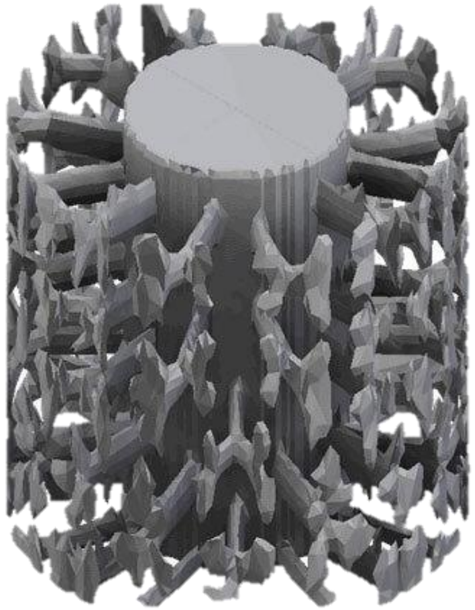
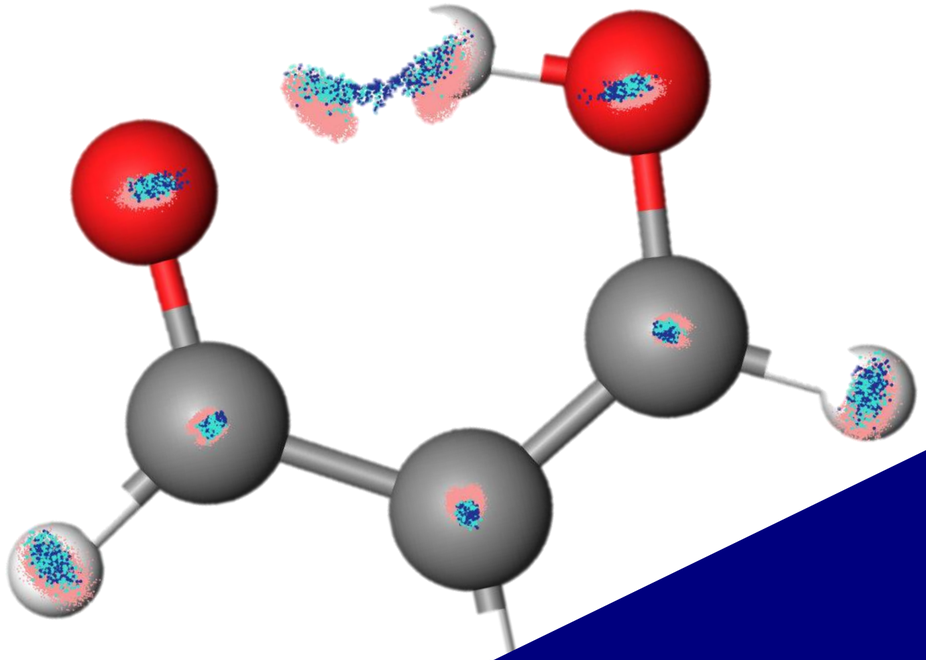
Bayesian Optimisation: Gaussian processes & Beyond

... towards effective data-driven optimisation





Haitham Bou Ammar
Reinforcement Learning Team Leader
Huawei R&D, Noah's Ark, London
H. Assistant Professor @ UCL

HUAWEI

We focus on the following black-box optimisation problem ...

Function not known analytically

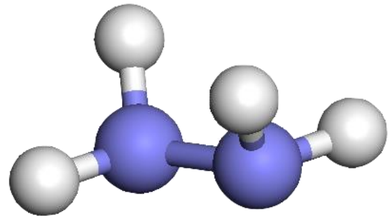
Function expensive to evaluate

Need a sample efficient solution

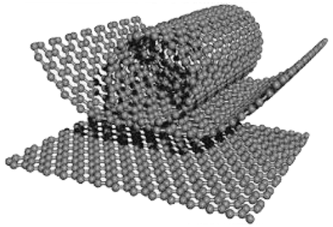
$$\max_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$$

Maximise a black-box function

... lots of real-world applications ...

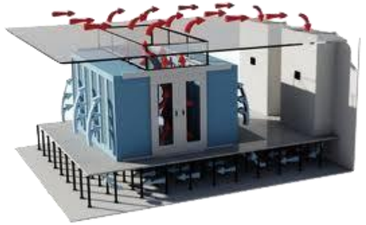


Molecule design



Material design

Power plant tuning



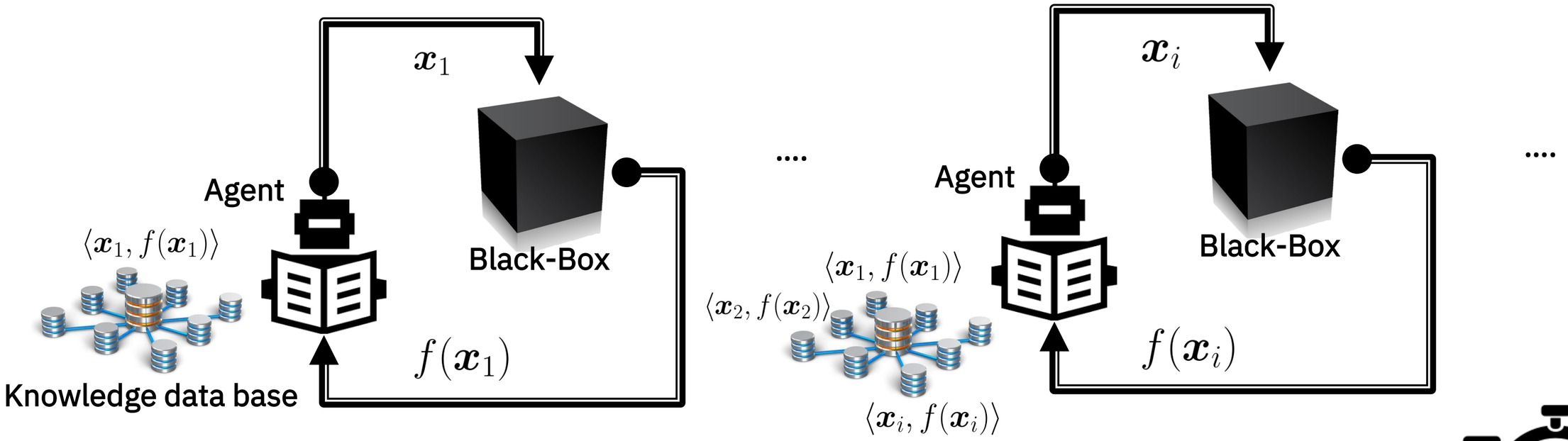
Data centre cooling

Hyper-param tuning

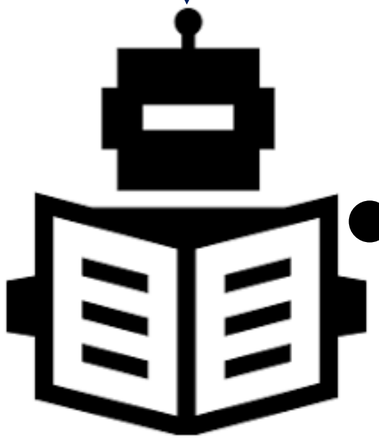
NAS & AutoML

… and optimise through a sequential data-driven scenario …

… improve the guess of the optimum while being data efficient in terms of how many queries we ask of the black-box …



… so, for example in hyper-parameter tuning of models …



Action/Decision

Feedback: Validation Loss

$\langle \boldsymbol{x} \equiv \text{hyper-params}, f(\boldsymbol{x}) \equiv \text{feedback} \rangle$

Save

… store in knowledge base …

$\mathcal{D}_{\text{train}}$

$\mathcal{A}_{\text{trainer}}(\text{hyper-params}, \mathcal{D}_{\text{train}})$

… e.g., minimise training loss …

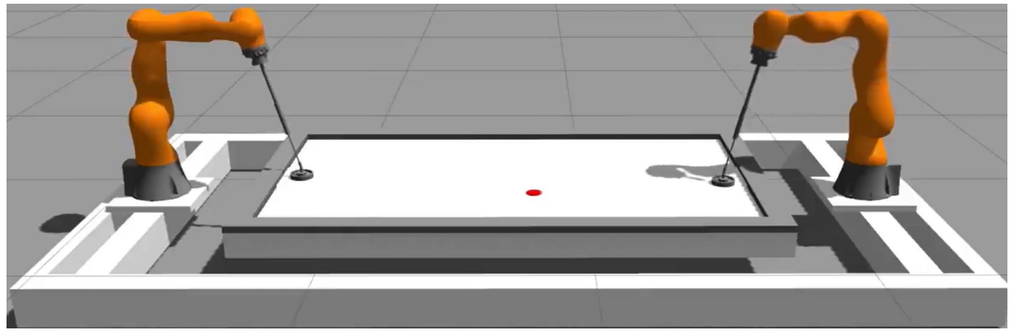
… trained network …

$\boldsymbol{W}^{\star}(\text{hyper-params}, \mathcal{D}_{\text{train}})$

$\mathcal{A}_{\text{validate}}(\boldsymbol{W}^{\star}(\text{hyper-params}, \mathcal{D}_{\text{train}}), \mathcal{D}_{\text{validate}})$

… or, in robotic scenarios for instance …

$$\text{sim-params} = \langle \mu_{\text{friction}}, \lambda_{\text{puck}}, \ldots \rangle$$



Simulated World

GAZEBO

Goal: Identify Simulation parameters

- Non-differentiable puck collision dynamics
- Need to differentiate through Gazebo
- Need quick way to tune sim-params

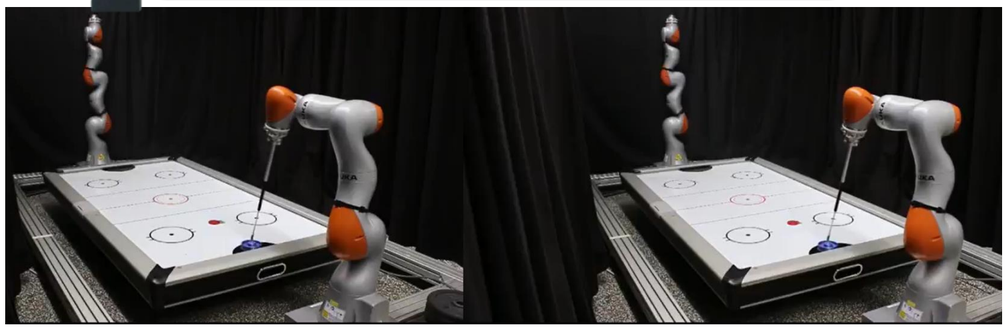
$$\mathcal{D}_{\text{sim}}(\text{sim-params})$$

Optitrack Sys

$$\mathcal{D}_{\text{real}}$$

$$\min_{\text{sim-params}} \mathcal{L}(\mathcal{D}(\text{sim-params}), \mathcal{D}_{\text{real}}) \equiv f(\text{simparams})$$

Feedback: Mismatch real-sim



Real-World

Action/Decision

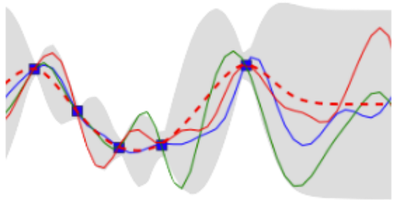
… hence, we need solvers that adhere to the following …

Can operate without the knowledge of the gradient and when the function in not known analytically

Efficient in terms of sample complexities (i.e., how many function evaluations we do)

Determine the global optimum of the black-box while being data driven.

… a **model-based data-driven method** achieving the above is **Bayesian Optimisation.**



Bayesian optimisation, operates in two main steps …

for $i = 0 : N - 1$:

Step I ⟷ Fit the GP model to the current dataset $\mathcal{D}_i$ by $\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ from Equation 1

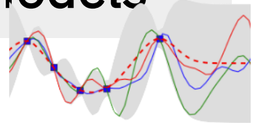
Step II ⟷ Find q points by solving $\mathbf{x}_{1:q}^{(\text{new})} = \arg\max_{\mathbf{x}_{1:q}} \alpha_{\text{q-type}}(\mathbf{x}_{1:q}|\mathcal{D}_i)$ (Round if categorical)

Evaluate new inputs by querying the black-box to acquire $\mathbf{y}_{1:q}^{(\text{new})} = f(\mathbf{x}_{1:q}^{(\text{new})})$

Update the dataset creating $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{\mathbf{x}_l^{(\text{new})}, y_l^{(\text{new})}\}_{l=1}^{q}$

end for

Step I: Learn Bayesian Models

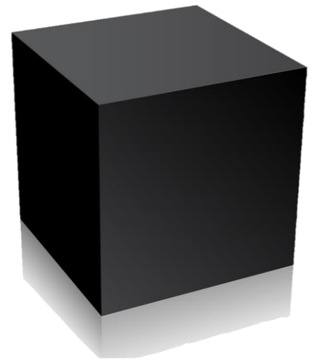


Step II: Maximise acquisition

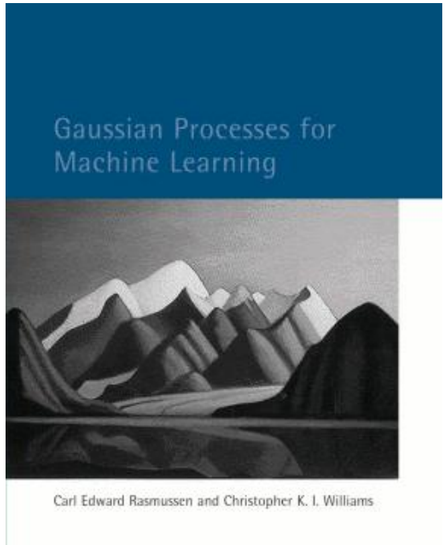
$$\max[\alpha_{\text{EI}}(\cdot), \alpha_{\text{PI}}(\cdot), \alpha_{\text{UCB}}(\cdot)]^{\text{T}}$$

… trades-off exploration vs exploitation …

$\boldsymbol{x}_i$



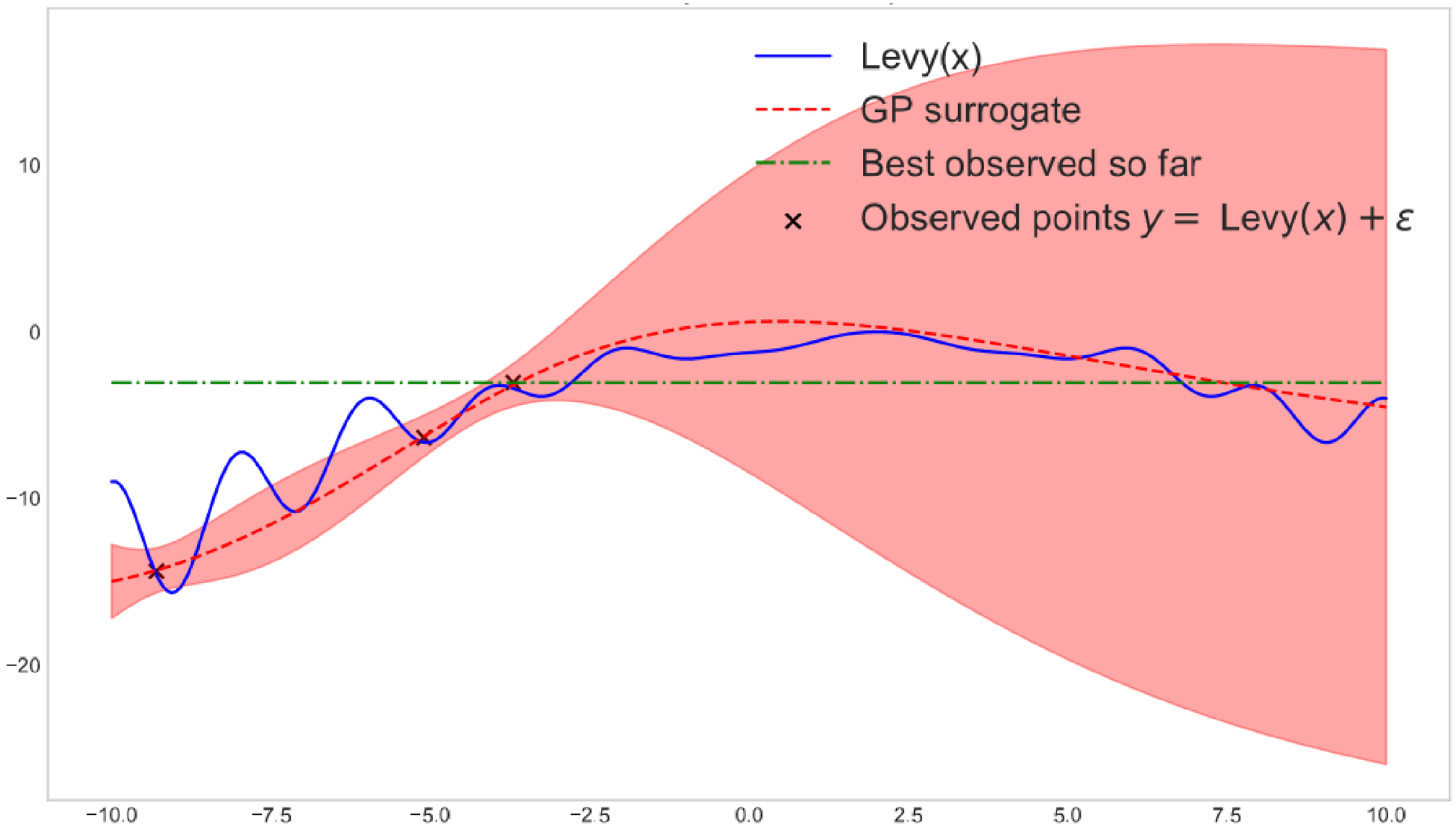
add data and repeat



Step I: Learning a Bayesian Model – **Gaussian Processes**

... some material is borrowed from James Hensman, Rich Turner, Nando de Freitas, and Neil Lawrence in addition to GP book itself ...

We would like to do inference **directly in the function space** ...

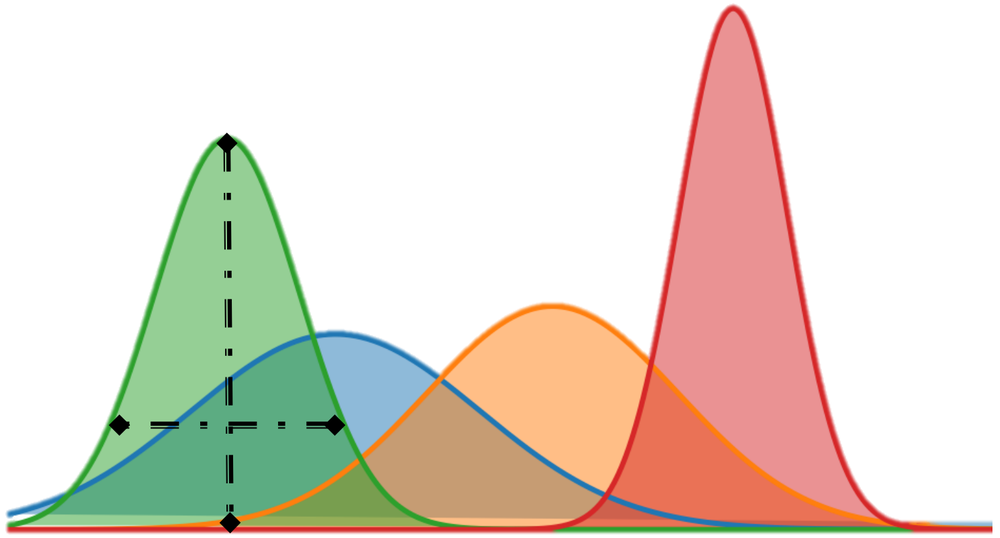


To do, let's go back to the basics of Gaussian distributions …

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

Variance

Mean



... in higher-dim. spaces, we can also define a Gaussian.

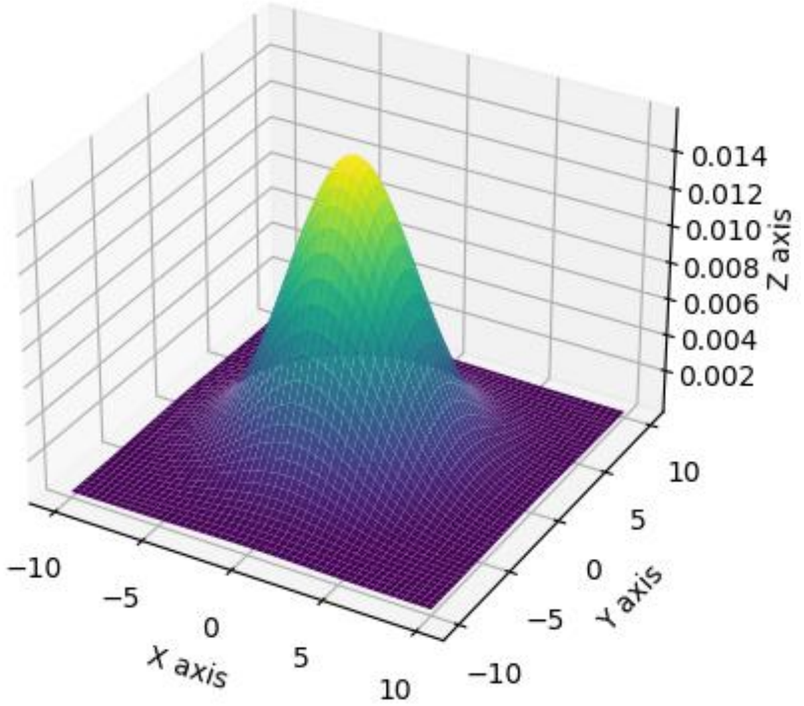
$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

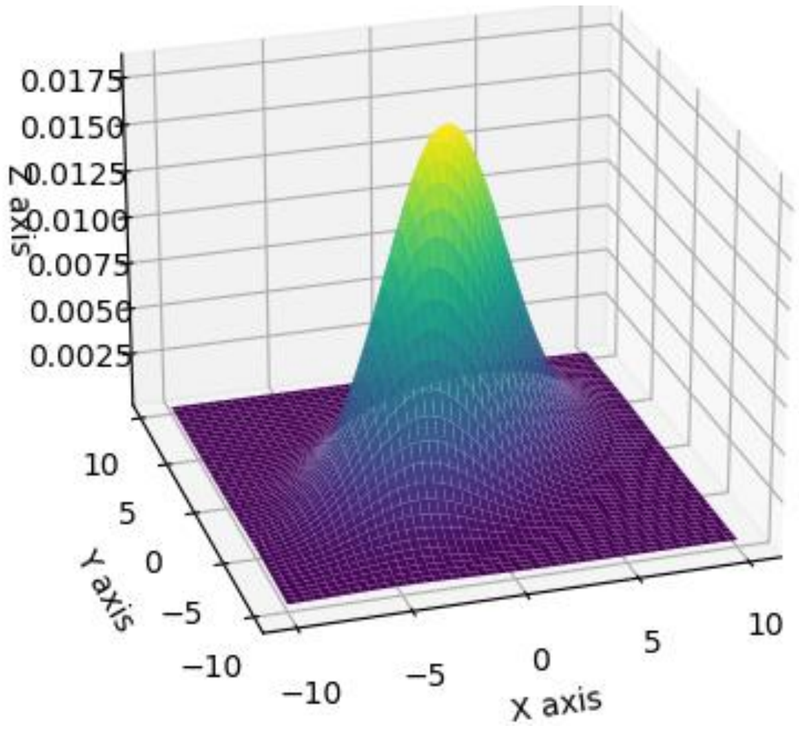
$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{X}]$$

$$\begin{aligned}\boldsymbol{\Sigma}_{i,j} &= \mathbb{E}\left[(\boldsymbol{X}_i - \boldsymbol{\mu}_i)(\boldsymbol{X}_j - \boldsymbol{\mu}_j)\right] \\ &= \mathrm{Cov}[\boldsymbol{X}_i, \boldsymbol{X}_j]\end{aligned}$$

$$\boldsymbol{\mu} = [0, 0]^{\mathsf{T}}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$





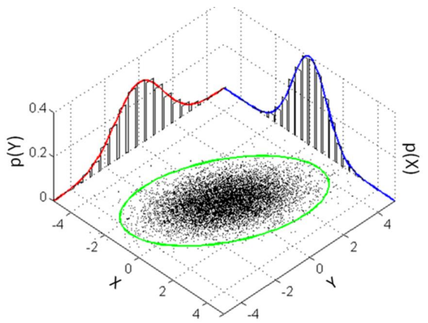
$$\boldsymbol{\mu} = [0, 0]^{\mathsf{T}}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 10 & 5 \\ 5 & 10 \end{bmatrix}$$

Some useful needed properties of a Gaussian include …

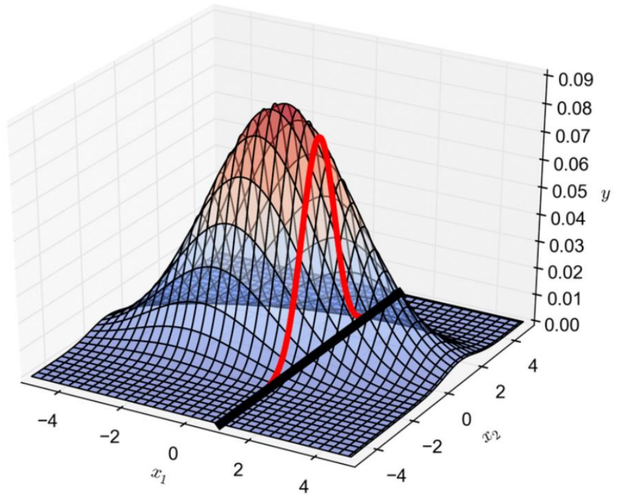
$$p(\boldsymbol{x}_1, \boldsymbol{x}_2) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{matrix} \boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_{1,2} \\ \boldsymbol{\Sigma}_{12}^{\mathsf{T}} & \boldsymbol{\Sigma}_2 \end{matrix} \right)$$

$$p(\boldsymbol{x}_1) = \int p(\boldsymbol{x}_1, \boldsymbol{x}_2) d\boldsymbol{x}_2 \ = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$



https://www.ritchievink.com/blog/2019/02/01/an-intuitive-introduction-to-gaussian-processes/

$$p(\boldsymbol{x}_1 | \boldsymbol{x}_2) = \mathcal{N}\left(\hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\Sigma}}_1\right)$$



$$\hat{\boldsymbol{\mu}}_1 = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{1,2}\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)$$ **Conditional Mean**

$$\hat{\boldsymbol{\Sigma}}_1 = \boldsymbol{\Sigma}_1 - \boldsymbol{\Sigma}_{1,2}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_{1,2}^{\mathsf{T}}$$ **Conditional Cov.**

https://www.ritchievink.com/blog/2019/02/01/an-intuitive-introduction-to-gaussian-processes/

Imagine we assume that outputs are jointly Gaussian ...

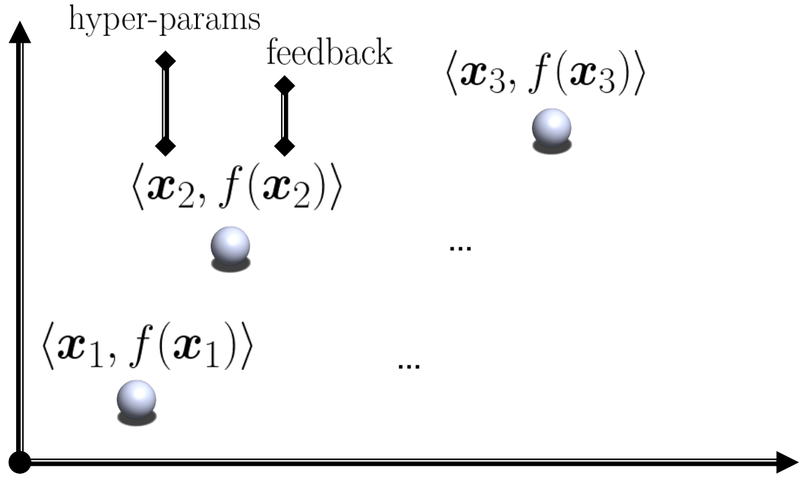
$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_3) \end{bmatrix} \Big\| \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{bmatrix} \sim \mathcal{N}(\underbrace{\mathbf{0}_{3\times 1}}_{\text{Mean-vector}}, \underbrace{\boldsymbol{K}_{3\times 3}}_{\text{Covariance Matrix}})$$

$$\boldsymbol{K}_{3\times 3} = \begin{bmatrix} \overbrace{k(\boldsymbol{x}_1, \boldsymbol{x}_1)}^{1} & \overbrace{k(\boldsymbol{x}_1, \boldsymbol{x}_2)}^{>} & k(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & k(\boldsymbol{x}_2, \boldsymbol{x}_3) \\ k(\boldsymbol{x}_3, \boldsymbol{x}_1) & k(\boldsymbol{x}_3, \boldsymbol{x}_2) & k(\boldsymbol{x}_3, \boldsymbol{x}_2) \end{bmatrix}$$

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) \propto \exp\left(-||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right) \quad \text{measures a similarity}$$

$$\begin{cases} 0 \quad \text{if } ||\boldsymbol{x}_i - \boldsymbol{x}_j|| \to \infty \\ 1 \quad \text{if } \boldsymbol{x}_i = \boldsymbol{x}_j \end{cases}$$

Closer points are more correlated



Nando de Freitas

44K subscribers • 69 videos

I am a machine learning professor at UBC. I am making my lectures available to the world with the hope that this will give more ...

… to predict, on a new input, we just condition …

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_3) \\ f(\boldsymbol{x}_\star) \end{bmatrix} \Bigg\| \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}_{4\times 1}, \boldsymbol{K}_{4\times 4}\right)$$

$$\boldsymbol{K}_{4\times 4} = \begin{bmatrix} \overbrace{\begin{matrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & k(\boldsymbol{x}_1,\boldsymbol{x}_2) & k(\boldsymbol{x}_1,\boldsymbol{x}_2) \\ k(\boldsymbol{x}_2,\boldsymbol{x}_1) & k(\boldsymbol{x}_2,\boldsymbol{x}_2) & k(\boldsymbol{x}_2,\boldsymbol{x}_3) \\ k(\boldsymbol{x}_3,\boldsymbol{x}_1) & k(\boldsymbol{x}_3,\boldsymbol{x}_2) & k(\boldsymbol{x}_3,\boldsymbol{x}_2) \end{matrix}}^{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{x})} & \begin{matrix} k(\boldsymbol{x}_1,\boldsymbol{x}_\star) \\ k(\boldsymbol{x}_2,\boldsymbol{x}_\star) \\ k(\boldsymbol{x}_3,\boldsymbol{x}_\star) \end{matrix} \\ \underbrace{\begin{matrix} k(\boldsymbol{x}_\star,\boldsymbol{x}_1) & k(\boldsymbol{x}_\star,\boldsymbol{x}_2) & k(\boldsymbol{x}_\star,\boldsymbol{x}_3) \end{matrix}}_{\boldsymbol{k}(\boldsymbol{x}_\star,\boldsymbol{x})} & k(\boldsymbol{x}_\star,\boldsymbol{x}_\star) \end{bmatrix}$$

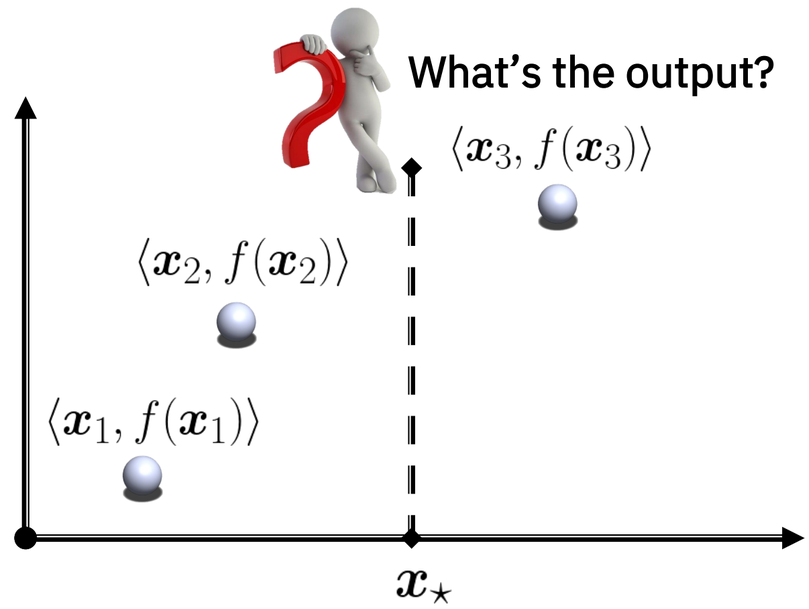
… assuming training and test are from the same distribution …

$$f(\boldsymbol{x}_\star)|\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3,\boldsymbol{x}_\star,f(\boldsymbol{x}_1),\ldots \sim \mathcal{N}\left(\mu_\star,\sigma_\star\right)$$

$$\mu_\star = \boldsymbol{k}(\boldsymbol{x}_\star,\boldsymbol{x})\boldsymbol{K}^{-1}(\boldsymbol{x},\boldsymbol{x})\boldsymbol{f}$$

Linear combinations of kernel basis functions

$$\sigma_\star = k(\boldsymbol{x}_\star,\boldsymbol{x}_\star) - \boldsymbol{k}(\boldsymbol{x}_\star,\boldsymbol{x})\boldsymbol{K}^{-1}(\boldsymbol{x},\boldsymbol{x})\boldsymbol{k}(\boldsymbol{x},\boldsymbol{x}_\star)$$



Now, let us step back and define a GP …

Definition: *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution*

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \qquad k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))]$$

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$$

… the random variable is the value of the function itself …

In other words, we can say that ...

... if $\boxed{f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))}$

... then ...

$$K(\mathbf{x}, \mathbf{x}') = \sigma_0^2 \exp\left[-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\lambda_d}\right)^2\right]$$

$\boldsymbol{\theta}$

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}\right)$$

... we will assume zero mean function ...

To learn, the kernel hypermeters ...

Assumptions

$$y = f(\boldsymbol{x}) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2)$$
$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$$

Given some data $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i \geq 1}$

Marginal Likelihood

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}, \boldsymbol{f}|\boldsymbol{X}) d\boldsymbol{f} = \int p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{X}) p(\boldsymbol{f}|\boldsymbol{X}) d\boldsymbol{f}$$

Likelihood

$$p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}, \sigma_n^2 \boldsymbol{I})$$

Prior $p(\boldsymbol{f}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{\boldsymbol{\theta}})$

Maximise

$$\log p(\boldsymbol{y}|\boldsymbol{X}) = \underbrace{-\frac{1}{2}\boldsymbol{y}^\mathsf{T}[\boldsymbol{K}_{\boldsymbol{\theta}} + \sigma_n^2 \boldsymbol{I}]^{-1}\boldsymbol{y}}_{\textbf{Data fit}} \underbrace{-\frac{1}{2}\log|\boldsymbol{K}_{\boldsymbol{\theta}} + \sigma_n^2 \boldsymbol{I}|}_{\textbf{Model Complexity}} - \text{cnst}$$

… so, let us derive it …

$$\log p(\mathbf{y}|\mathbf{X}) = \log \int p(\mathbf{y}, \mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the marginalisation property ("summing-out" latent)

$$= \log \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the chain rule of probability

Note I: Understanding notation

$\mathbf{y} = [y_1, \ldots, y_n]^\mathsf{T}$ vector of all outputs

$\mathbf{f} = [f_1, \ldots, f_n]^\mathsf{T}$ vector of all latents

$\mathbf{X} \in \mathbb{R}^{n \times d}$ matrix of all inputs

n # data pts.

d # dims

Note II: Given the notation, note that

all outputs all latents all inputs

$$p(\mathbf{y}|\mathbf{f}, \mathbf{X}) = p(y_1, \ldots, y_n | f_1, \ldots, f_n, \mathbf{x}_1, \ldots, \mathbf{x}_n)$$ … data **likelihood** given latents & inputs

$$p(\mathbf{f}|\mathbf{X}) = p(f_1, \ldots, f_n | \mathbf{x}_1, \ldots, \mathbf{x}_n)$$ … **prior** given a set of input data points

… so, let us derive it …

$$\log p(\mathbf{y}|\mathbf{X}) = \log \int p(\mathbf{y}, \mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the marginalisation property (“summing-out” latent)

$$= \log \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the chain rule of probability

Realisation I: Understanding the Likelihood

$p(\mathbf{y}|\mathbf{f}, \mathbf{X}) = \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I})$

Step I: Let’s look at the joint distribution

$$\mathbf{y} = \mathbf{f} + \boldsymbol{\epsilon}_n$$
$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}})$$
$$\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \sigma_n^2 \boldsymbol{I})$$

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I})$$

sum of 2 multi-variate Gaussian random vectors

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I} & ? \\ ? & \mathbf{K}_{\boldsymbol{\theta}} \end{bmatrix} \right)$$

$$? = \text{cov}(\mathbf{y}, \mathbf{f}) = \text{cov}(\mathbf{f} + \boldsymbol{\epsilon}_n, \mathbf{f}) = \text{cov}(\mathbf{f} + \boldsymbol{\epsilon}_n, \mathbf{f})$$
$$= \text{cov}(\mathbf{f}, \mathbf{f}) + \text{cov}(\boldsymbol{\epsilon}_n, \mathbf{f})$$
$$= \mathbf{K}_{\boldsymbol{\theta}}$$

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I} & ? \\ ? & \mathbf{K}_{\boldsymbol{\theta}} \end{bmatrix} \right)$$

… therefore, our likelihood can be written as …

$$\log p(\mathbf{y}|\mathbf{X}) = \log \int p(\mathbf{y}, \mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the marginalisation property (“summing-out” latent)

$$= \log \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the chain rule of probability

Realisation I: Understanding the Likelihood

$p(\mathbf{y}|\mathbf{f}, \mathbf{X}) = \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I})$

Step I: Let’s look at the joint distribution

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I} & \mathbf{K}_{\boldsymbol{\theta}} \\ \mathbf{K}_{\boldsymbol{\theta}} & \mathbf{K}_{\boldsymbol{\theta}} \end{bmatrix} \right)$$

Step II: Simply use conditioning property

$$\mathbf{y}|\mathbf{f}, \mathbf{X} \sim \mathcal{N}\left(\mathbf{0} + \mathbf{K}_{\boldsymbol{\theta}} \mathbf{K}_{\boldsymbol{\theta}}^{-1} (\mathbf{f} - \mathbf{0}), \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I} - \mathbf{K}_{\boldsymbol{\theta}} \mathbf{K}_{\boldsymbol{\theta}}^{-1} \mathbf{K}_{\boldsymbol{\theta}}^{\top} \right) \implies p(\mathbf{y}|\mathbf{f}, \mathbf{X}) = \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I})$$

Going back to our integral & plugging-in our results …

$$\log p(\mathbf{y}|\mathbf{X}) = \log \int p(\mathbf{y}, \mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the marginalisation property (“summing-out” latent)

$$= \log \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f}$$ … using the chain rule of probability

$$= \log \int \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I}) \mathcal{N}(\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}}) d\mathbf{f}$$ … plugging-in our previous results

Realisation II: Product of Gaussians & integrals

$$\mathcal{N}(\mathbf{a}, \mathbf{A}) \mathcal{N}(\mathbf{b}, \mathbf{B}) = Z^{-1} \mathcal{N}(\mathbf{c}, \mathbf{C}) \quad \text{with} \quad \mathbf{c} = \mathbf{C}(\mathbf{A}^{-1}\mathbf{a} + \mathbf{B}^{-1}\mathbf{b}) \;\; \mathbf{C} = (\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1}$$

$$Z^{-1} = (2\pi)^{-n/2} |\mathbf{A} + \mathbf{B}|^{-1/2} \exp\left(-\frac{1}{2} (\mathbf{a} - \mathbf{b})^{\mathsf{T}} (\mathbf{A} + \mathbf{B})^{-1} (\mathbf{a} - \mathbf{b}) \right)$$

… normalisation term is a Gaussian itself …

Going back to our integral & plugging-in our results ...

$$\log p(\mathbf{y}|\mathbf{X}) = \log \int p(\mathbf{y}, \mathbf{f}|\mathbf{X}) d\mathbf{f}$$ ... using the marginalisation property ("summing-out" latent)

$$= \log \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f}$$ ... using the chain rule of probability

$$= \log \int \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I}) \mathcal{N}(\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}}) d\mathbf{f}$$ ... plugging-in our previous results

$$= \log \ \mathcal{N}(\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I})$$

... no approximation, rather exact ...

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top [\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2 \mathbf{I}| - \text{cnst}$$

Some implementations include …

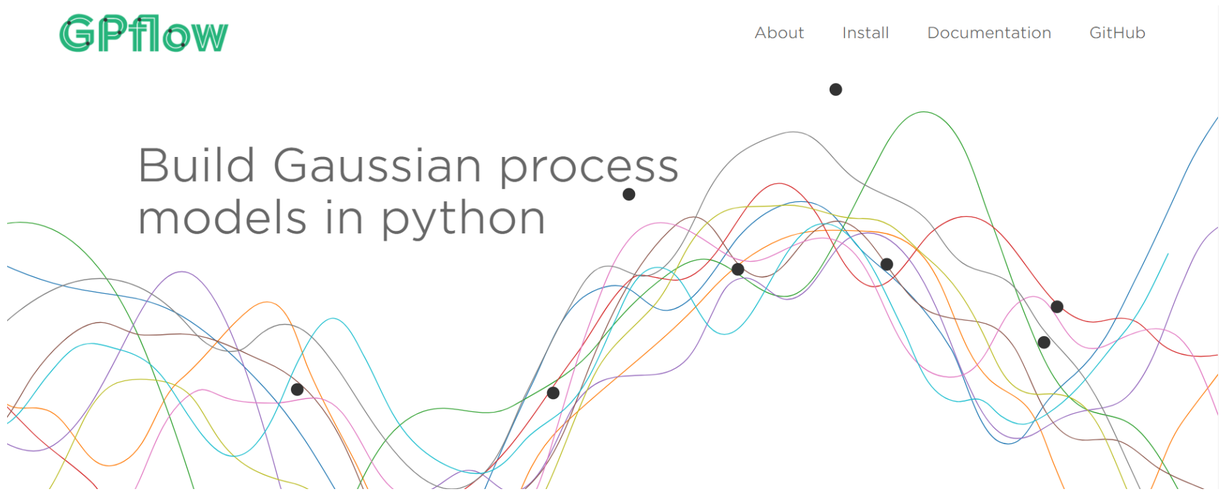
https://www.gpflow.org/

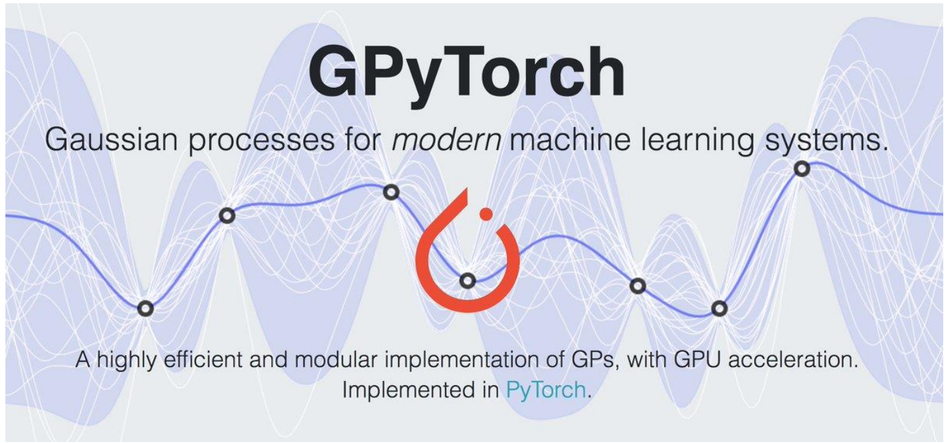
```
[3]: k = gpflow.kernels.Matern52()

[5]: m = gpflow.models.GPR(data=(X, Y), kernel=k, mean_function=None)

[8]: opt = gpflow.optimizers.Scipy()

[9]: opt_logs = opt.minimize(m.training_loss, m.trainable_variables, options=dict(maxiter=100))
     print_summary(m)
```

Train

```
[10]: ## generate test points for prediction
      xx = np.linspace(-0.1, 1.1, 100).reshape(100, 1)  # test points must be of shape (N, D)

      ## predict mean and variance of latent GP at test points
      mean, var = m.predict_f(xx)

      ## generate 10 samples from posterior
      tf.random.set_seed(1)  # for reproducibility
      samples = m.predict_f_samples(xx, 10)  # shape (10, 100, 1)
```

Predict





https://gpytorch.ai/

Train

```
self.covar_module = ScaleKernel(RBFKernel() + LinearKernel())
```

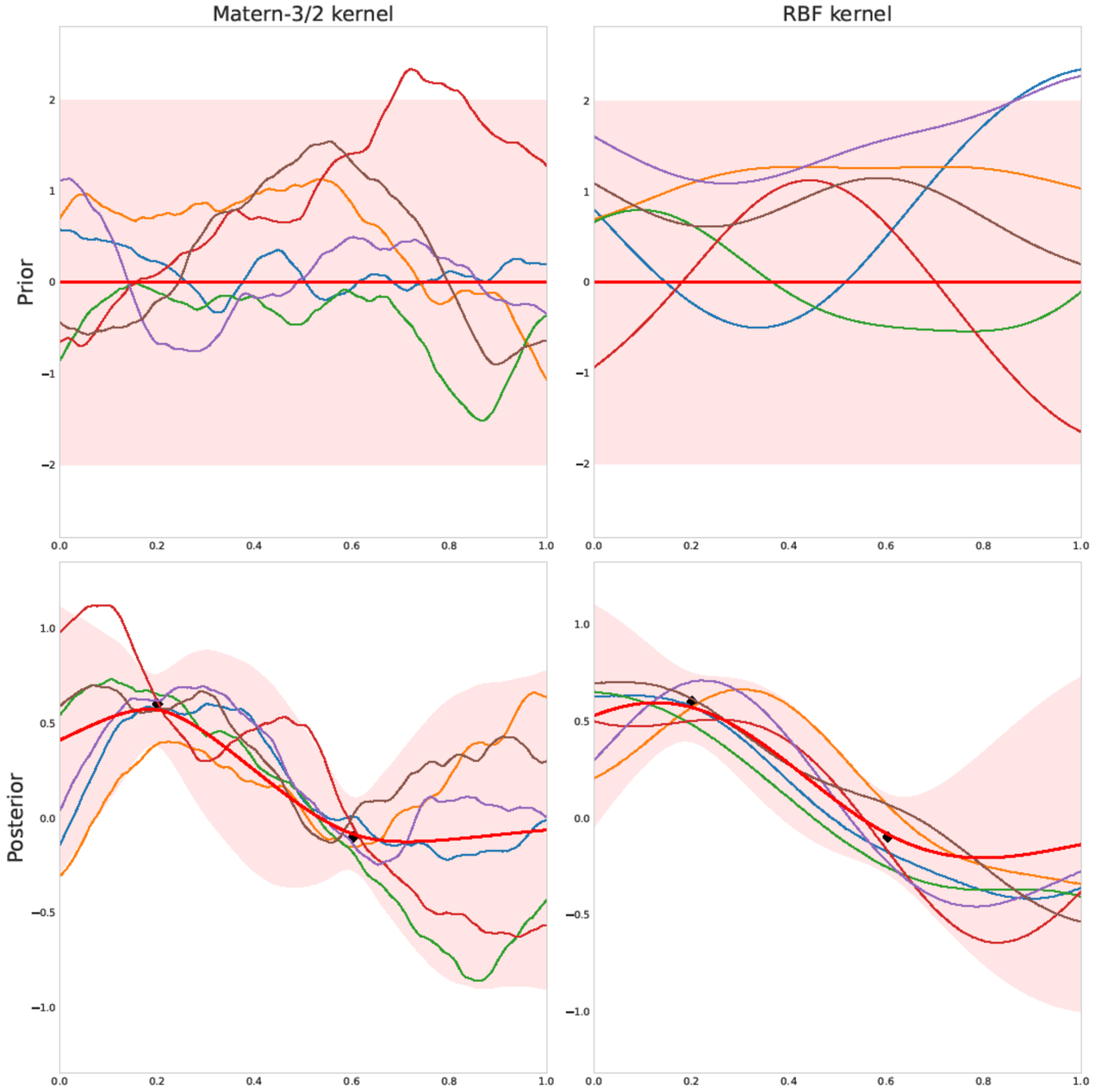
```
# initialize likelihood and model
likelihood = gpytorch.likelihoods.GaussianLikelihood()
model = ExactGPModel(train_x, train_y, likelihood)
```

```
# Use the adam optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)  # Includes GaussianLikelihood parameters

# "Loss" for GPs - the marginal log likelihood
mll = gpytorch.mlls.ExactMarginalLogLikelihood(likelihood, model)

for i in range(training_iter):
    loss = -mll(output, train_y)
    loss.backward()
    optimizer.step()
```

Predict

```
f_preds = model(test_x)
y_preds = likelihood(model(test_x))

f_mean = f_preds.mean
f_var = f_preds.variance
f_covar = f_preds.covariance_matrix
f_samples = f_preds.sample(sample_shape=torch.Size(1000,))
```

… giving you an accurate fit in a couple lines of code.



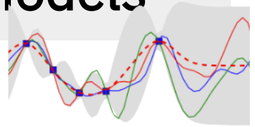

```python
x = np.linspace(0, 1, 200).reshape(-1, 1)
train_x = np.array([x[40], x[120]])
train_y = np.array([[.6], [-.1]])

def plot_prior(kernel, n_samples:int, kname=None, ax=None):
    if ax is None:
        ax = plt.subplot()

    # Sample multivariate Gaussian (zero-mean GP)
    cov = kernel(x, x).numpy()
    samples = np.random.multivariate_normal(np.zeros(len(x)), cov, size=n_samples)

    # plot
    for i in range(n_samples):
        ax.plot(x.flatten(), samples[i])
    ax.plot([0, 1], [0, 0], c='r', linewidth=2)
    ax.fill_between([0, 1], [-2, -2], [2, 2], alpha=.1, color='r')
    lim =max(np.max(np.abs(samples)), 2) * 1.2
    ax.set_ylim(-lim, lim)
    ax.set_xlim(0, 1)
    ax.grid(False)
    return ax

def plot_posterior(gp, n_samples:int, kname=None, ax=None):
    if ax is None:
        ax = plt.subplot()
    # get posterior mean and std
    predictions = gp.predict_y(x)
    mean = predictions[0].numpy().flatten()
    std = predictions[1].numpy().flatten()

    # sample from GP
    samples = gp.predict_f_samples(x, n_samples).numpy()

    # plot
    for i in range(n_samples):
        ax.plot(x.flatten(), samples[i].flatten())
    ax.plot(x.flatten(), mean, c='r', linewidth=2)
    ax.fill_between(x.flatten(), mean - 2 * std, mean + 2 * std, alpha=.1, color='r')
    ax.scatter(train_x.flatten(), train_y.flatten(), marker='D', s=50, c='k')
    lim =max(np.max(np.abs(samples)), np.max(mean + 2 * std)) * 1.2
    ax.set_ylim(-lim, lim)
    ax.set_xlim(0, 1)
    ax.grid(False)
    return ax
```


BREAK
TIME!!!

Step II: Find new Probes to Test – **Acquisition Functions**

With the GP at hand now we look at step II …

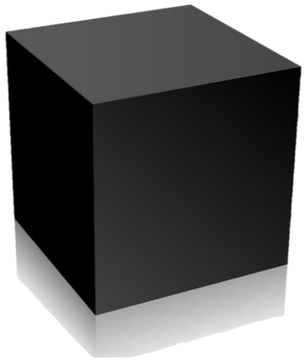
Step I: Learn Bayesian Models



Step II: Maximise acquisition

$$\max[\alpha_{\mathrm{EI}}(\cdot), \alpha_{\mathrm{PI}}(\cdot), \alpha_{\mathrm{UCB}}(\cdot)]^{\mathsf{T}}$$

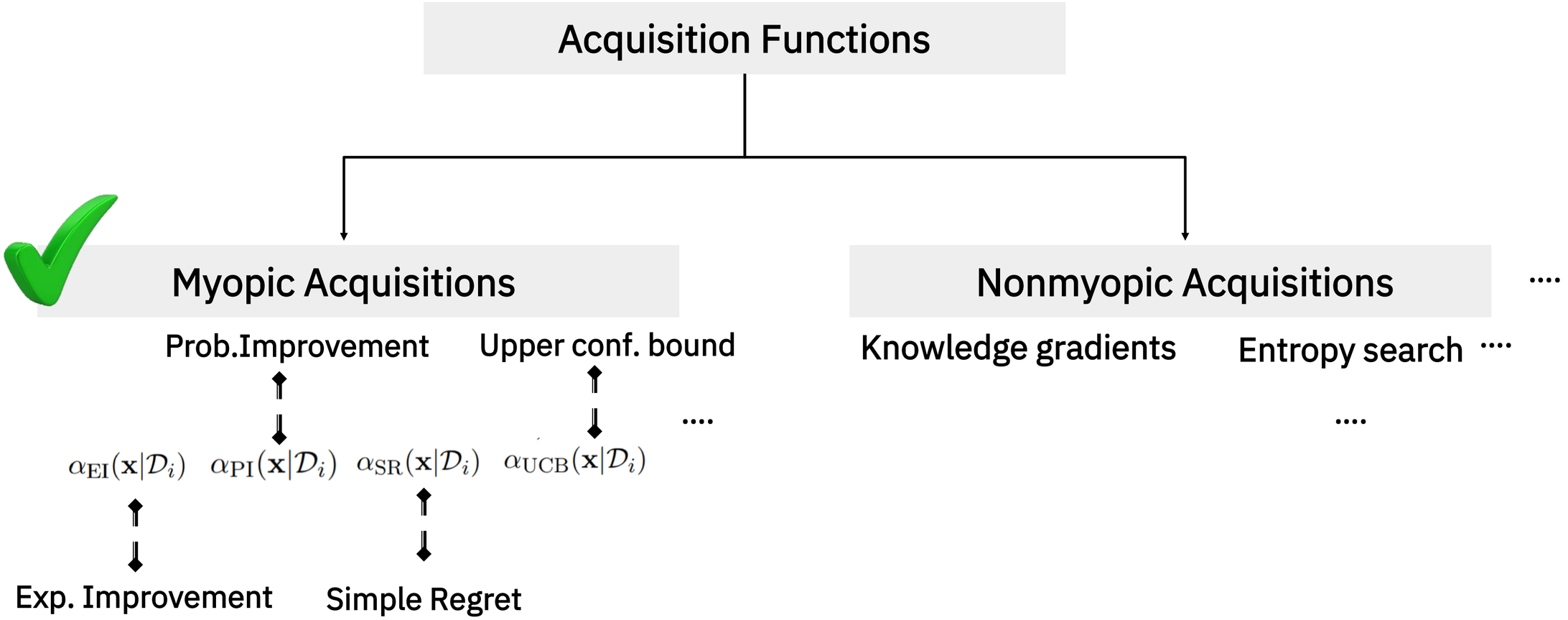
$\boldsymbol{x}_i$



Step II $\longleftrightarrow$ Find q points by solving $\mathbf{x}_{1:q}^{(\mathrm{new})} = \arg\max_{\mathbf{x}_{1:q}} \alpha_{\text{q-type}}(\mathbf{x}_{1:q}|\mathcal{D}_i)$

add data and repeat

… acquisitions trade-off exploration & exploitation …



… considering, expected improvement for example …

Best in collected data so-far

$$\alpha_{\mathrm{EI}}(\mathbf{x}|\mathcal{D}_i) = \mathbb{E}_{f(\mathbf{x})|\mathcal{D}_i,\boldsymbol{\theta}}\left[\max\{(f(\mathbf{x}) - f(\mathbf{x}_i^+)), 0\}\right]$$

$$= \mathbb{E}_{f(\mathbf{x})|\mathcal{D}_i,\boldsymbol{\theta}}\left[\mathrm{ReLU}(f(\mathbf{x}) - f(\mathbf{x}_i^+))\right]$$

… find new probe that does better on average than best seen so-far …

```
d = 1
STD = 0.25

test_func_ = Levy(dim=1, negate=True)
test_func = lambda x: test_func_(x - 1)

bounds = torch.stack([-torch.ones(d), torch.ones(d)]) * 10
X = torch.linspace(-10, 10, 300)
Y = test_func(X.unsqueeze(1))
```

```
acq_func_name = 'Expected Improvement'
for it in tqdm(range(50)):

    # normalise / standardise
    train_X_step = normalize(train_X, bounds)
    mu_y, std_y = train_Y.mean(), train_Y.std()
    train_Y_step = train_Y.add(-mu_y).div(std_y)

    model = SingleTaskGP(train_X_step, train_Y_step)
    mll = ExactMarginalLogLikelihood(model.likelihood, model)
    fit_gpytorch_model(mll);
```

```
posterior = model.posterior(normalize(X.unsqueeze(1), bounds))
sur_std_Y_ = posterior.variance.sqrt().detach().numpy().flatten()
sur_mean_Y_ = posterior.mean.detach().numpy().flatten() * std_y.numpy().flatten() + mu_y.numpy().flatten()
sur_std_Y_ *=  std_y.numpy().flatten()  # de-standardise std
```

Gaussian Process prediction

```
        optimizer.zero_grad()
        acq_val_ = acq_func(X_start)
        acq_val_loss = - acq_val_.sum() # maximise acquisition -> minimise (- acquisition)
        acq_val_loss.backward()
        optimizer.step()
```

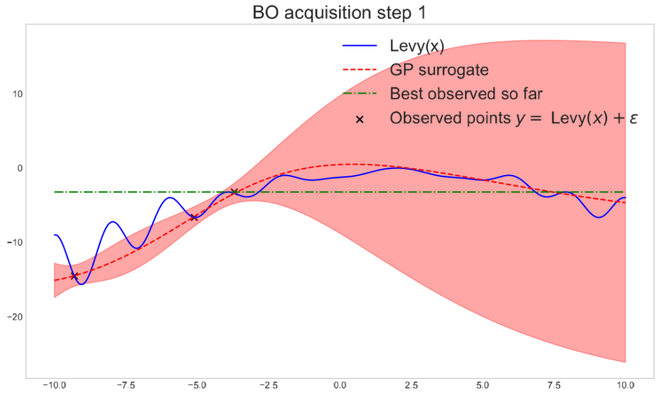
```
best_value = train_Y_step.max()
acq_func = ExpectedImprovement(model=model, best_f=best_value)
```

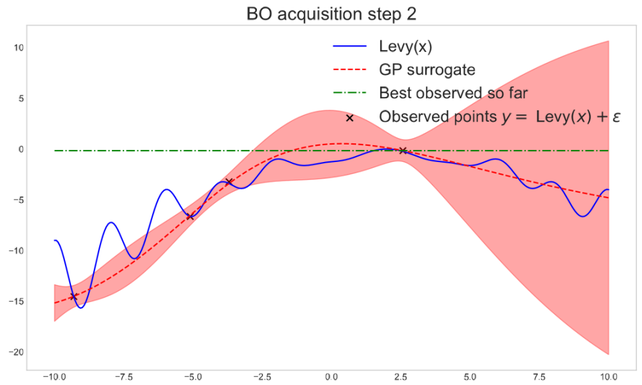
```
acq_val = acq_func(normalize(X.reshape(-1, 1, 1), bounds))
```

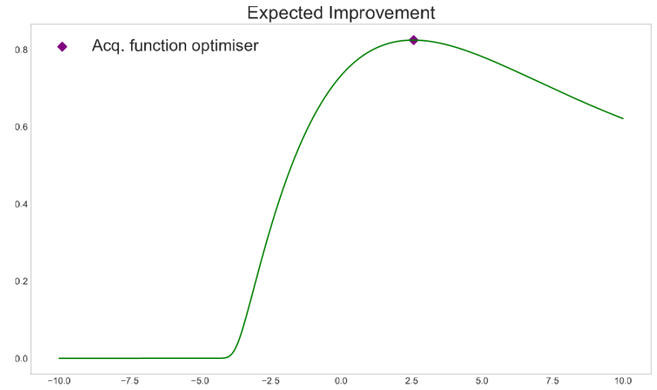
```
new_x_ind = acq_val.argmax()
X_new = X[None, new_x_ind]
Y_new = test_func(X_new) + STD * torch.randn(1)  # new noisy observation

# Add newly observed point to training set
train_X = torch.cat([train_X, X_new.unsqueeze(1)])
train_Y = torch.cat([train_Y, Y_new.unsqueeze(1)])
```

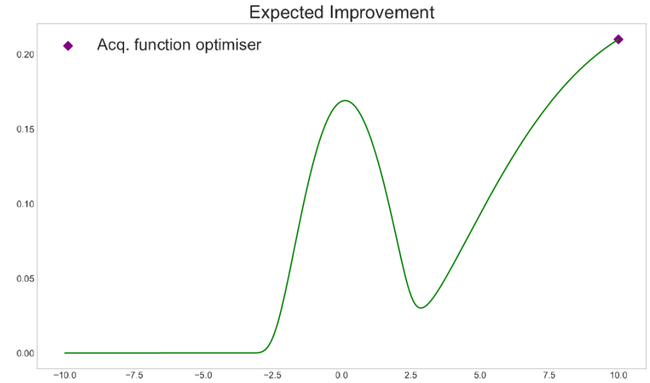
Acquisition & Evaluation

… considering, expected improvement for example …

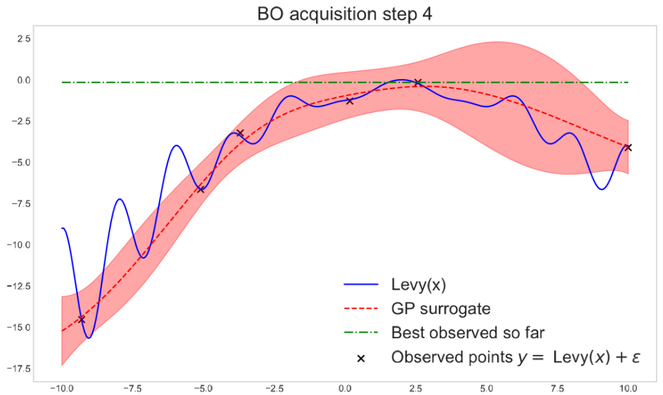


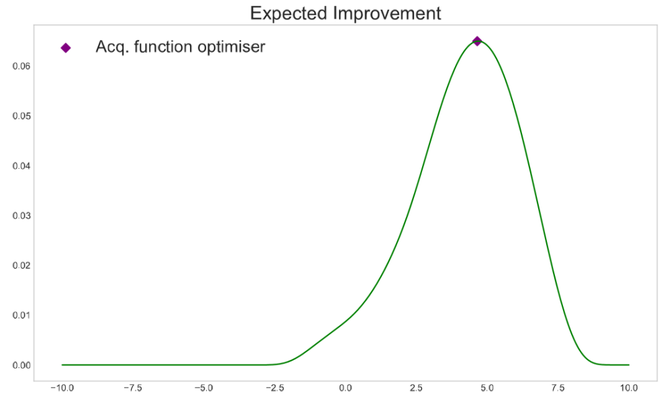


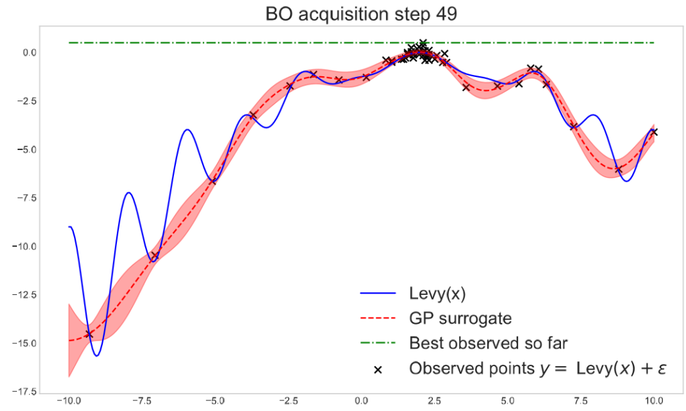




…







… considering, probability of improvement …

$$\alpha_{\mathrm{PI}}(\mathbf{x}|\mathcal{D}_i) = \mathbb{E}_{f(\mathbf{x})|\mathcal{D}_i,\boldsymbol{\theta}}\left[\mathbb{1}\{f(\mathbf{x}) - f(\mathbf{x}_i^+)\}\right]$$

Best in collected data so-far

… find new probe that does better in probability than best seen so-far …

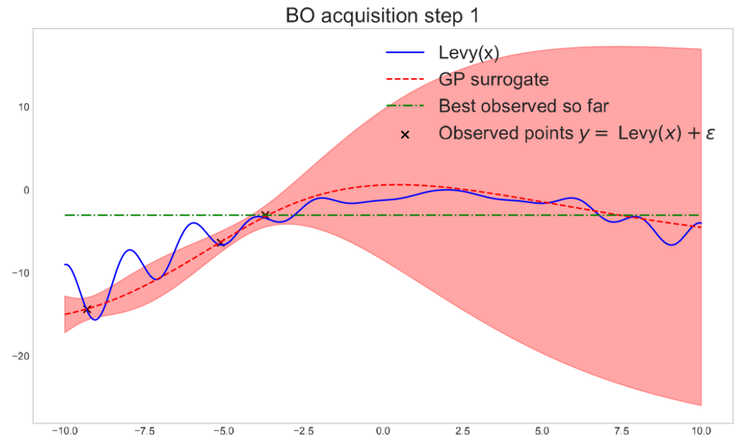
… simple change …

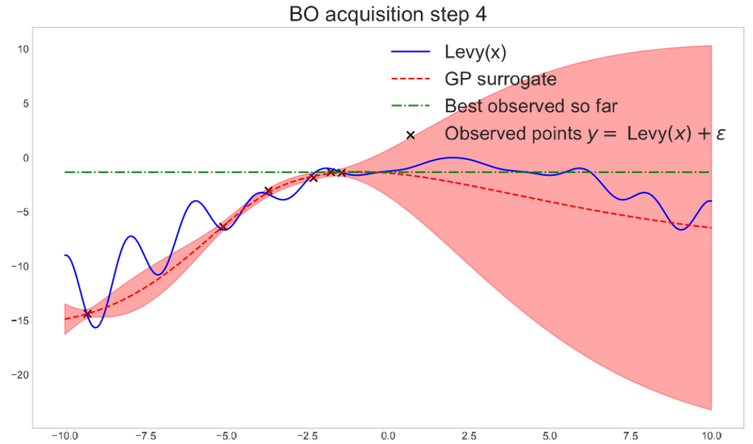
```
acq_func_name = 'Probability of Improvement'
for it in tqdm(range(50)):

    mu_y, std_y = train_Y.mean(), train_Y.std()
    train_X_step = normalize(train_X, bounds)
    train_Y_step = train_Y.add(-mu_y).div(std_y)
    model = SingleTaskGP(train_X_step, train_Y_step)
    mll = ExactMarginalLogLikelihood(model.likelihood, model)
    fit_gpytorch_model(mll);

    best_value = train_Y_step.max()
    acq_func = ProbabilityOfImprovement(model=model, best_f=best_value)

    posterior = model.posterior(normalize(X.unsqueeze(1), bounds))
    sur_std_Y_ = posterior.variance.sqrt().detach().numpy().flatten()
    sur_mean_Y_ = posterior.mean.detach().numpy().flatten() * std_y.numpy().flatten() + mu_y.numpy().flatten()
    sur_std_Y_ *=  std_y.numpy().flatten()
```

```
acq_val = acq_func(normalize(X.reshape(-1, 1, 1), bounds))
```

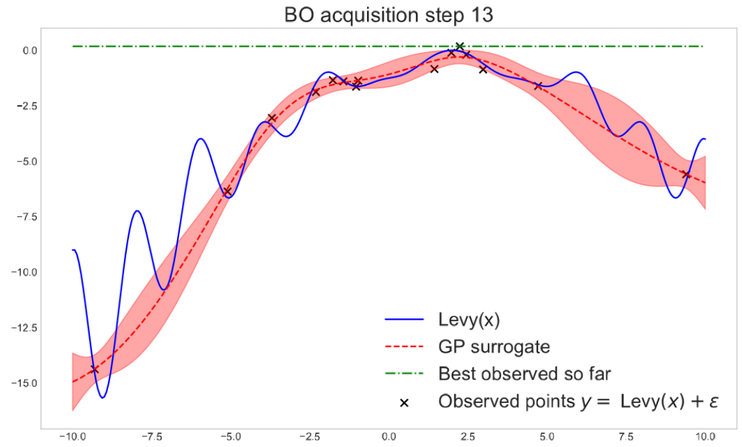
```
new_x_ind = acq_val.argmax()
X_new = X[None,new_x_ind]
Y_new = test_func(X_new) + STD * torch.randn(1)
```

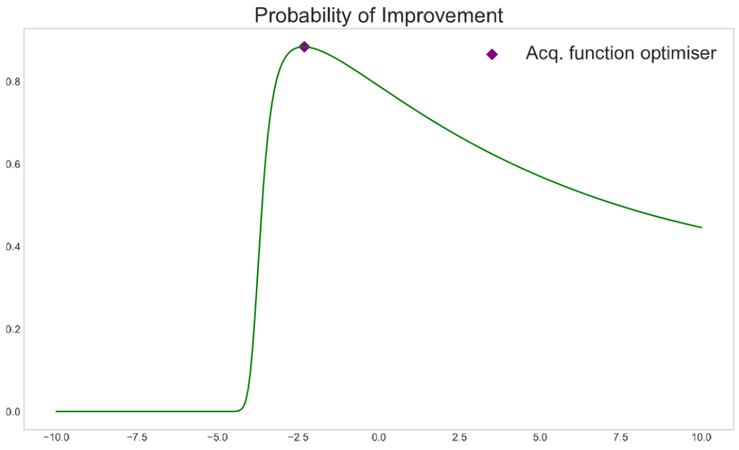
… considering, probability of improvement …

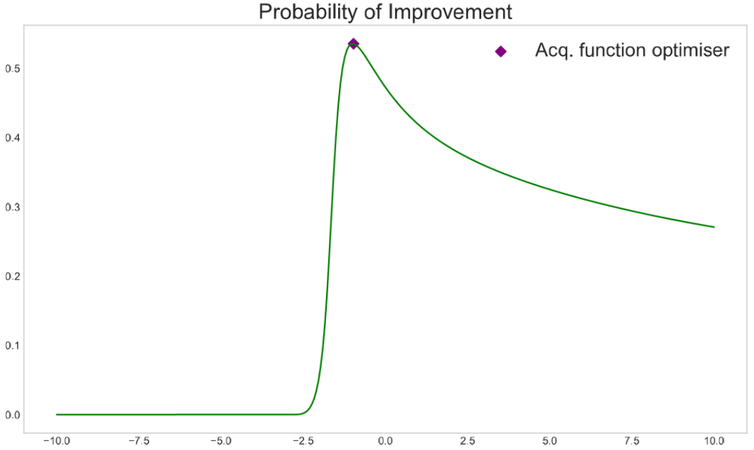


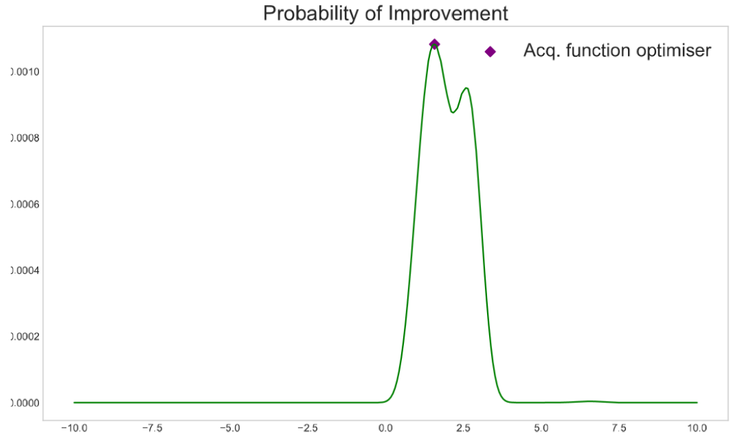


…

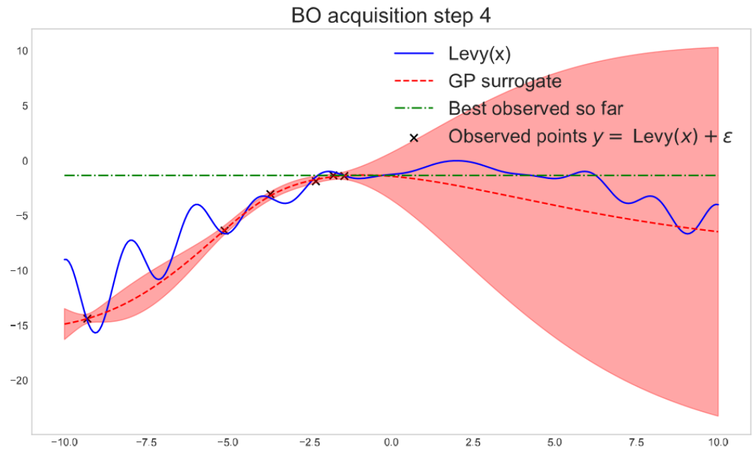


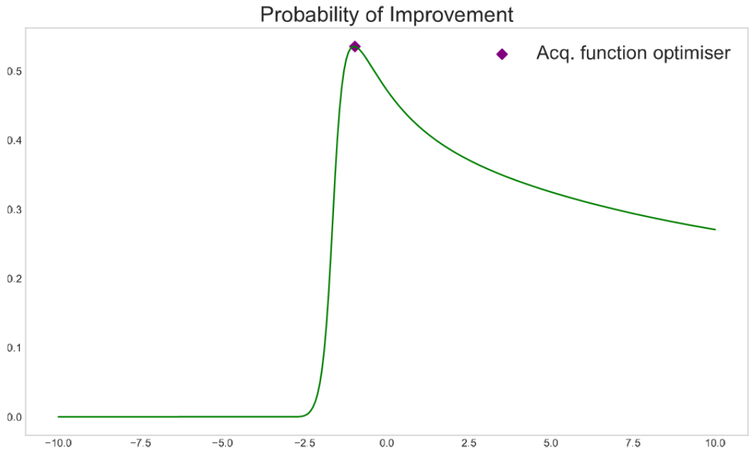


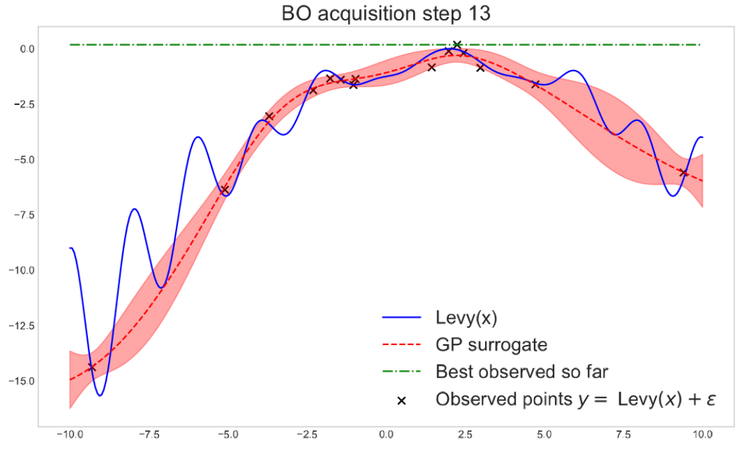


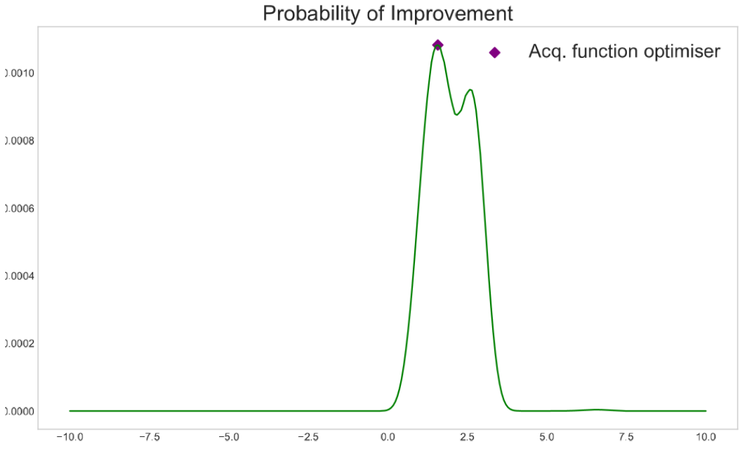


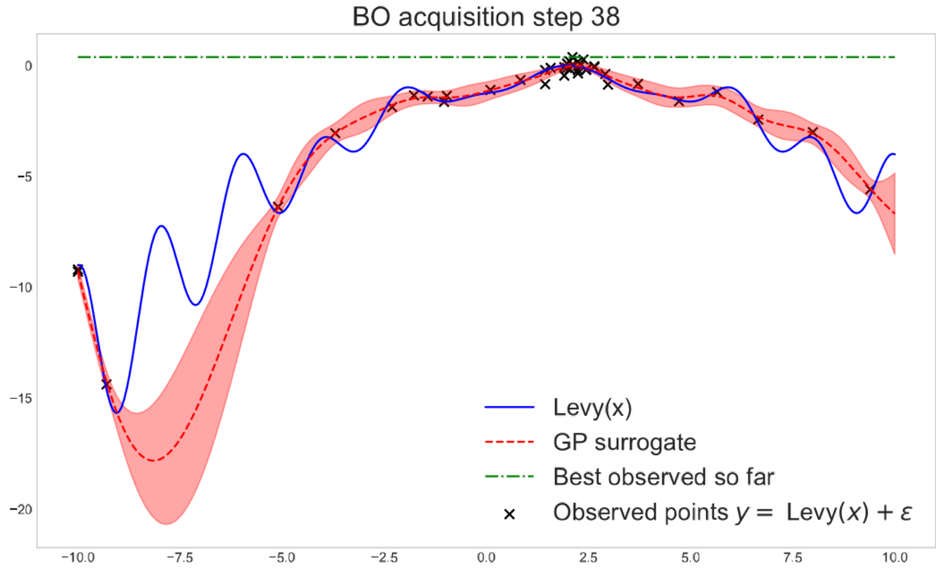
… considering, probability of improvement …











HEBO: New BO Lib with SOTA methods

Bayesian Optimisation Research

This directory contains official implementations for Bayesian optimisation works developped by Huawei R&D, Noah's Ark Lab.

- HEBO: Heteroscedastic Evolutionary Bayesian Optimisation
- T-LBO
- Bayesian Optimisation with Compositional Optimisers

Further instructions are provided in the README files associated to each project.

HEBO

https://github.com/huawei-noah/HEBO

Bayesian optimsation library developped by Huawei Noahs Ark Decision Making and Reasoning (DMnR) lab. The **winning submission** to the NeurIPS 2020 Black-Box Optimisation Challenge.

HEBO: NeurIPS BBO Winners

Easy to use

Flexible to use

Supports MOO

Supports parallel BO

Supports High-D BO

Supports Evol. Algos

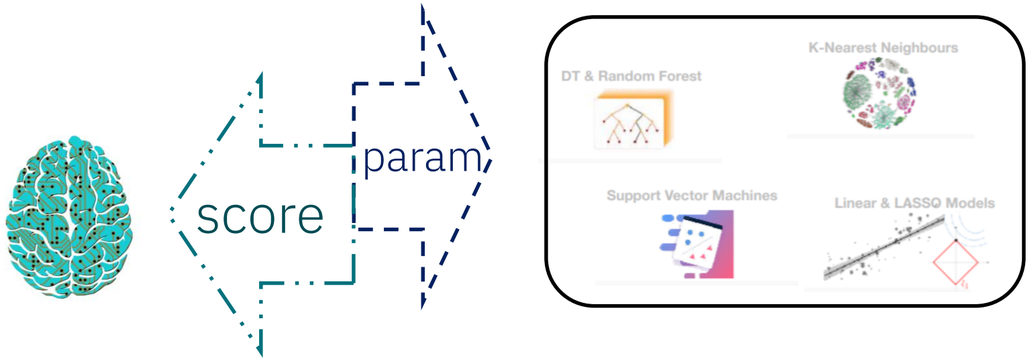
Supports Mixed-Variable Optimization

Includes a variety of solvers and benchmarks

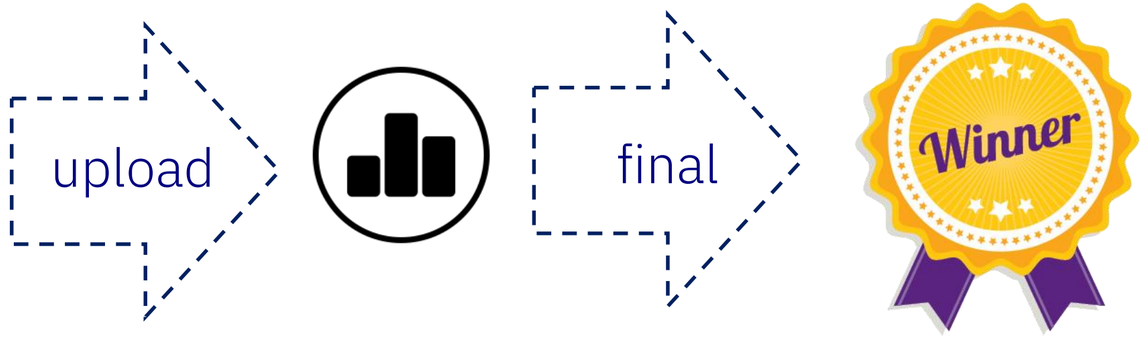
Interfaces with GPyTorch, GPy, GPFLow, Pymoo

NeurIPS BBO Competition

Mixed variables (cont. & discrete)



… as such we want a robust solution …



Offline training phase

Online testing phase

Final testing phase (hidden set of tasks)

Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020

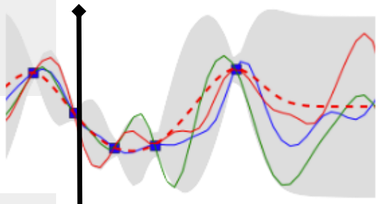
An Empirical Study of Assumptions in Bayesian Optimisation

For the purposes of this article we use the term ensemble in a broad sense. Ensembles could be built from multiple surrogates, acquisition functions, or potentially entire optimization algorithms, each of which could independently be used to fully power the optimization. The level at which these ensembling decisions were made varied across each of the teams. The first-placed team Huawei Noah's Ark Lab [11] used an elaborate compilation of acquisition functions and incorporated them into a multi-objective optimization strategy to select the next suggestions from the Pareto frontier. Other popular approaches were to alternate the kernel in the GP model or to use multiple surrogate models such as a GP and an XGBoost model.

… warping and multi-objectivity for non-stationary processes..

… use the acquired data to find a better rule …

… learn an effective surrogate model…



… optimise novel objective functions …

… novel model & objective …

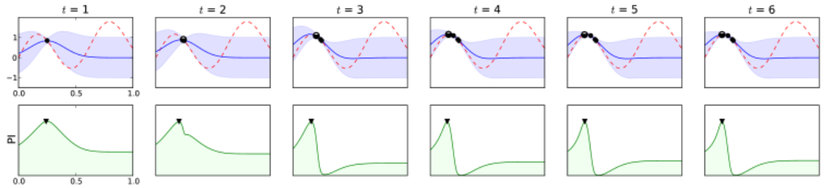
HEBO 河伯
Heteroscedastic Evolutionary Bayesian Optimisation

HEBO: Heteroscedastic & Evolutionary BO $\max[\alpha_{\text{EI}}(\cdot), \alpha_{\text{PI}}(\cdot), \alpha_{\text{UCB}}(\cdot)]^{\mathsf{T}}$

… determines which points to test next by trading-off exploration and exploitation …

$$\alpha_{\text{EI}}(\mathbf{x}|\mathcal{D}_i) = \mathbb{E}_{f(\mathbf{x})|\mathcal{D}_i,\boldsymbol{\theta}}\left[\max\{(f(\mathbf{x}) - f(\mathbf{x}_i^+)), 0\}\right]$$ … find best average gain over best known …

$$\alpha_{\text{PI}}(\mathbf{x}|\mathcal{D}_i) = \mathbb{E}_{f(\mathbf{x})|\mathcal{D}_i,\boldsymbol{\theta}}\left[\mathbb{1}\{f(\mathbf{x}) - f(\mathbf{x}_i^+)\}\right]$$ … find maximal probability of improvement over best known …



… we argue for a model-based Bayesian solution with constrained objectives for real-world SDM …

… we identify simplistic assumptions in literature …

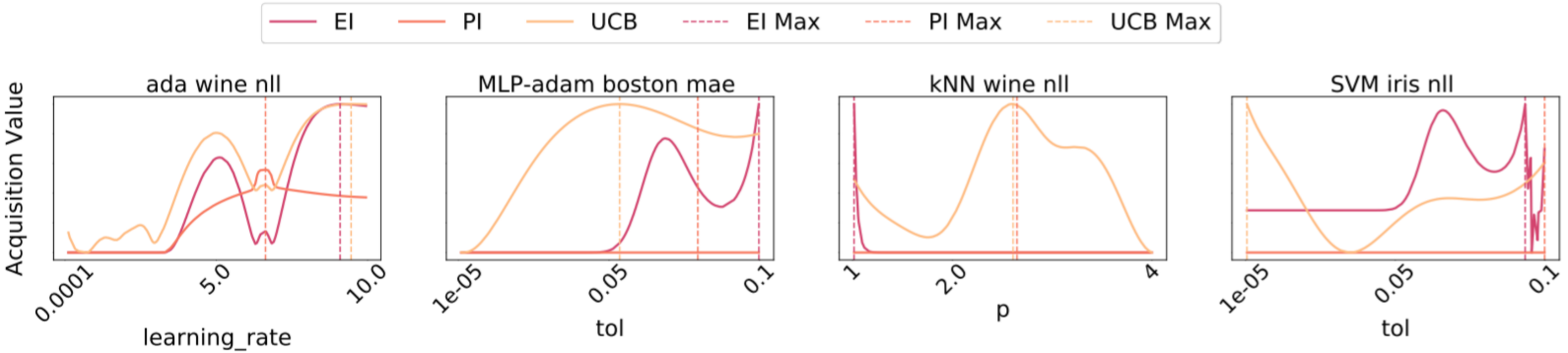
… use the acquired data to find a better rule …

Prob1: Data likelihood is highly non-Gaussian

… learn an effective surrogate model…

… optimise novel objective functions …

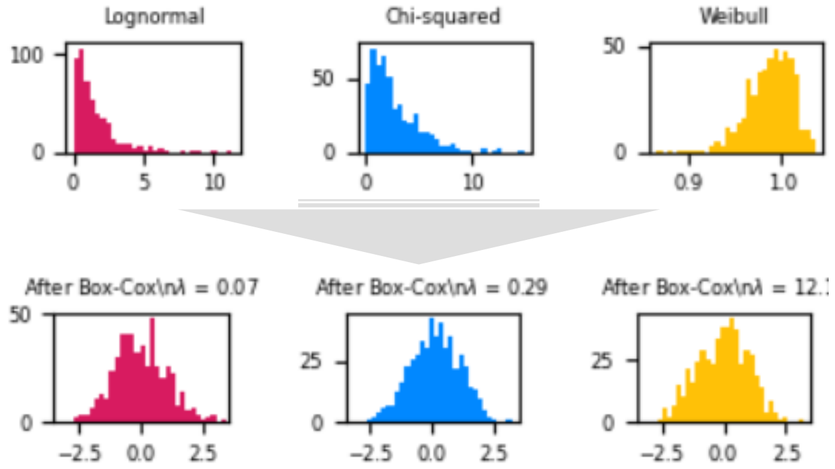
Prob2: Different Acquisitions give conflicting results



… fixing those two problems, HEBO achieves SOTA BO.

Prob1: Data likelihood is highly non-Gaussian

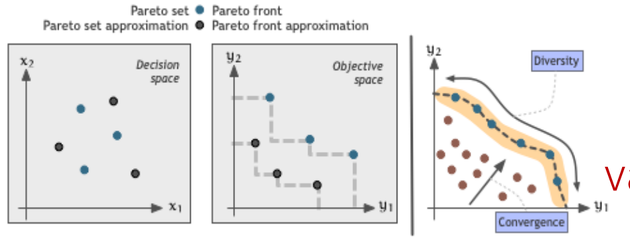
(1) Used a warping process (2) Used a box-cox transform



Prob2: Different Acquisitions give conflicting results

(1) Used an MOO acquisition (2) Looked for a Pareto front

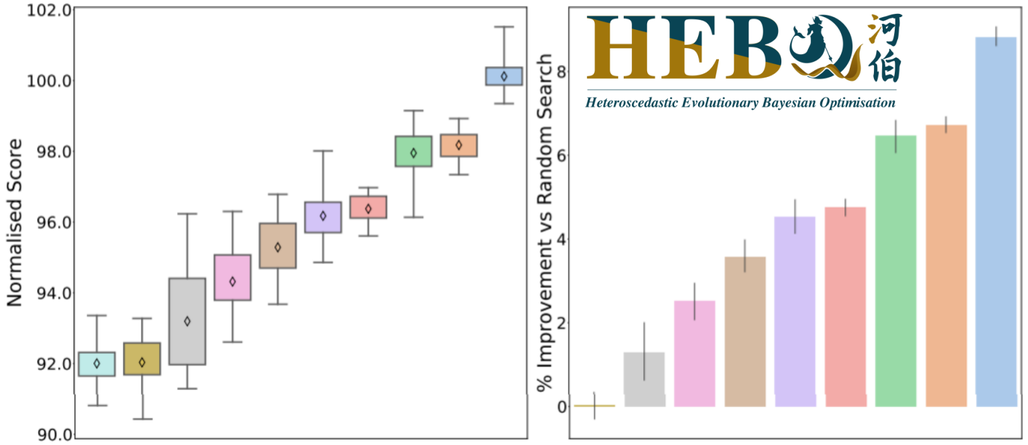
$$\max[\alpha_{\mathrm{EI}}(\cdot), \alpha_{\mathrm{PI}}(\cdot), \alpha_{\mathrm{UCB}}(\cdot)]^{\mathsf{T}}$$



NSGAII that supports mixed variable mutation and cross-over

LEADERBOARD

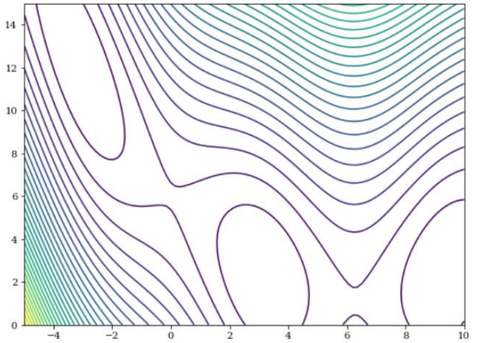
Team	Score	Prize
Huawei Noah's Ark Lab	93.519	$6,000
NVIDIA RAPIDS.AI	92.928	$4,000
JetBrains Research	92.509	$3,000
Duxiaoman DI	92.212	$1,000
Optuna Developers (Preferred Networks & CyberAgent)	91.806	$1,000



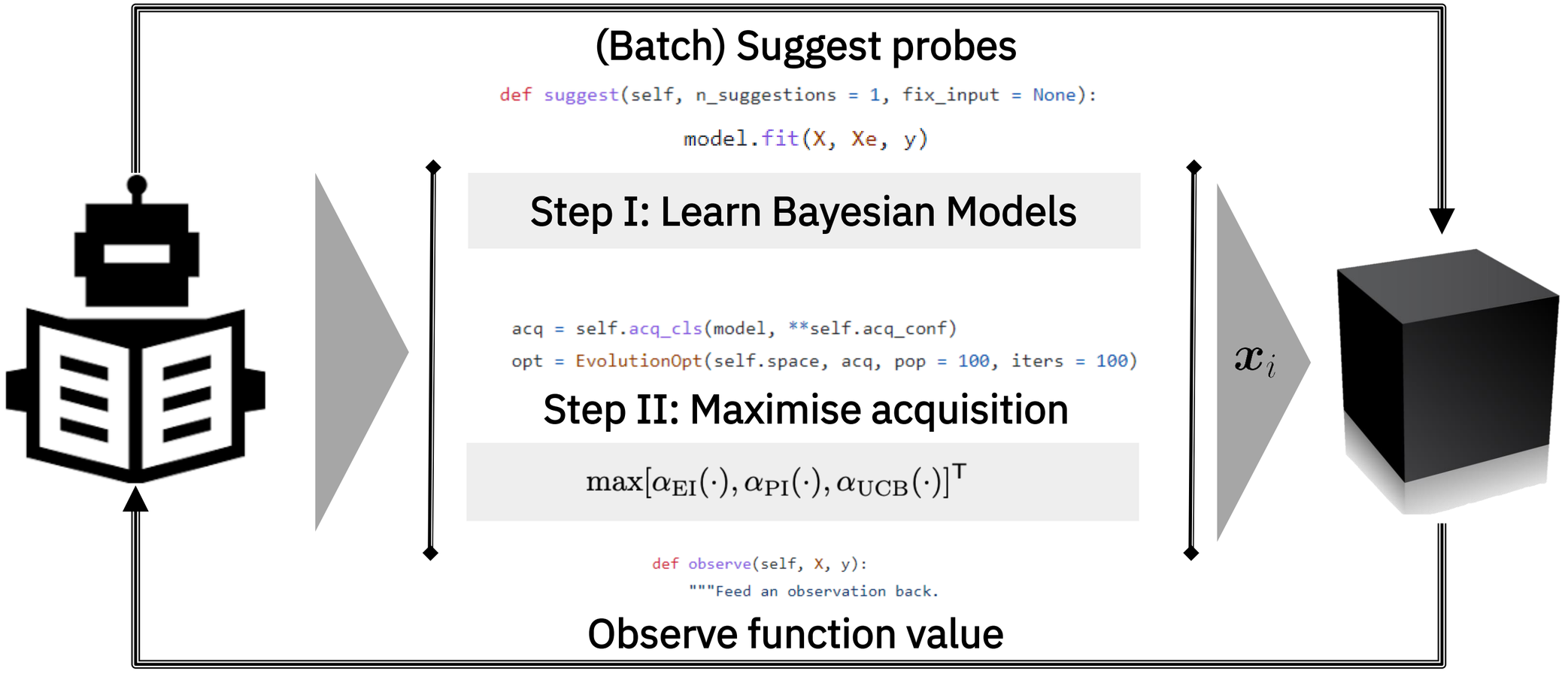
https://github.com/huawei-noah/HEBO

It’s implementation is really simple with our library …

```
problem = get_problem('go-branin01')

get_visualization("fitness-landscape", problem, angle=(45, 45), _type="contour").show()
```



```
from hebo.design_space.design_space import DesignSpace
space = DesignSpace().parse([
    {'name' : 'x0', 'type' : 'num', 'lb' : -5, 'ub' : 10},
    {'name' : 'x1', 'type' : 'num', 'lb' : 0,  'ub' : 15}
])
```



… then you could run any flavor of BO algorithms you'd like …

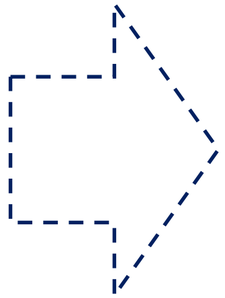
```python
bo    = BO(space)
for i in range(64):
    rec_x = bo.suggest()
    bo.observe(rec_x, obj(rec_x))
    if i % 4 == 0:
        print('Iter %d, best_y = %.2f' % (i, bo.y.min()))
```

```python
hebo_seq    = HEBO(space, model_name = 'gpy', rand_sample = 4)
for i in range(64):
    rec_x = hebo_seq.suggest(n_suggestions=1)
    hebo_seq.observe(rec_x, obj(rec_x))
    if i % 4 == 0:
        print('Iter %d, best_y = %.2f' % (i, hebo_seq.y.min()))
```

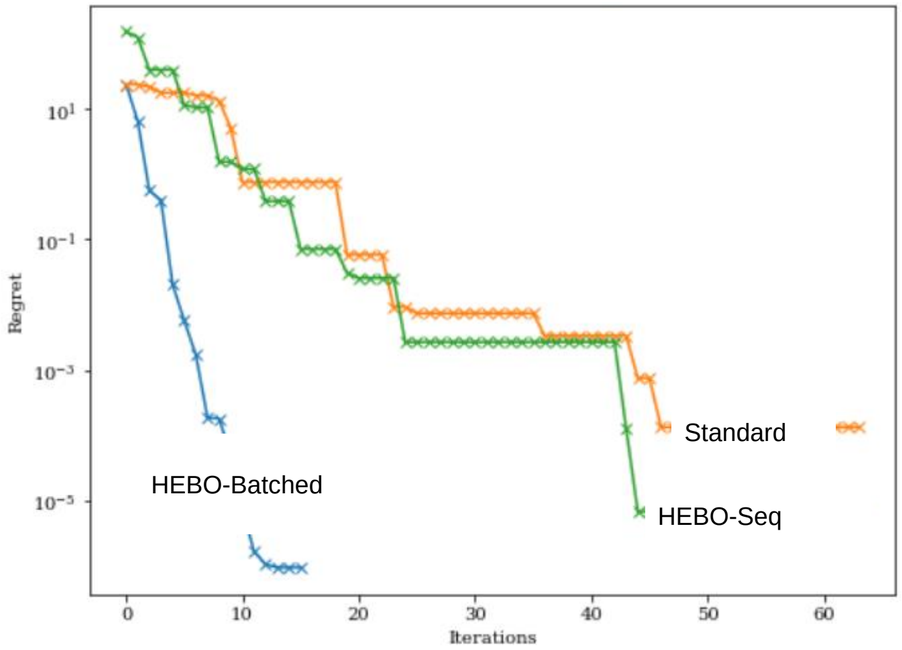
```python
def suggest(self, n_suggestions = 1, fix_input = None):
    assert n_suggestions == 1
    if self.X.shape[0] < self.rand_sample:
        sample = self.space.sample(n_suggestions)
        if fix_input is not None:
            for k, v in fix_input.items():
                sample[k] = v
        return sample
    else:
        X, Xe     = self.space.transform(self.X)
        y         = torch.FloatTensor(self.y)
        num_uniqs = None if Xe.shape[1] == 0 else [len(self.space.paras[name].categories) for name in self.space.enum_names]
        model     = get_model(self.model_name, X.shape[1], Xe.shape[1], y.shape[1], num_uniqs = num_uniqs, warp = False)
        model.fit(X, Xe, y)

        acq = self.acq_cls(model, **self.acq_conf)
        opt = EvolutionOpt(self.space, acq, pop = 100, iters = 100)

        suggest = self.X.iloc[[np.argmin(self.y.reshape(-1))]]
        return opt.optimize(initial_suggest = suggest, fix_input = fix_input)


        X, Xe = self.space.transform(self.X)
        try:
            if self.y.min() <= 0:
                y = torch.FloatTensor(power_transform(self.y / self.y.std(), method = 'yeo-johnson'))
            else:
                y = torch.FloatTensor(power_transform(self.y / self.y.std(), method = 'box-cox'))
                if y.std() < 0.5:
                    y = torch.FloatTensor(power_transform(self.y / self.y.std(), method = 'yeo-johnson'))
            if y.std() < 0.5:
                raise RuntimeError('Power transformation failed')
            model = get_model(self.model_name, self.space.num_numeric, self.space.num_categorical, 1, **self.model_config)
            model.fit(X, Xe, y)
        except:
            y     = torch.FloatTensor(self.y).clone()
            model = get_model(self.model_name, self.space.num_numeric, self.space.num_categorical, 1, **self.model_config)
            model.fit(X, Xe, y)
```

… and we also support batched suggestions …

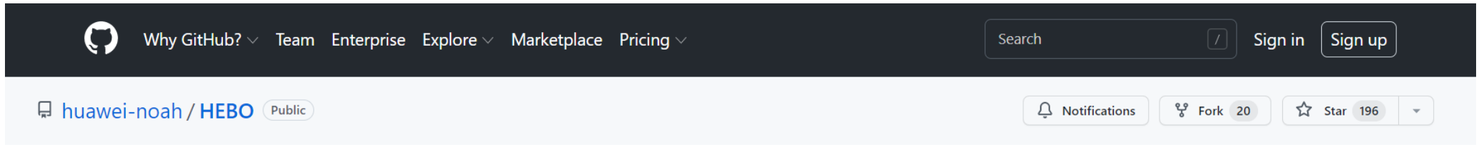
```python
hebo_seq   = HEBO(space, model_name = 'gpy', rand_sample = 4)
for i in range(64):
    rec_x = hebo_seq.suggest(n_suggestions=1)
    hebo_seq.observe(rec_x, obj(rec_x))
    if i % 4 == 0:
        print('Iter %d, best_y = %.2f' % (i, hebo_seq.y.min()))
```



```python
hebo_batch = HEBO(space, model_name = 'gpy', rand_sample = 4)
for i in range(16):
    rec_x = hebo_batch.suggest(n_suggestions=8)
    hebo_batch.observe(rec_x, obj(rec_x))
    print('Iter %d, best_y = %.2f' % (i, hebo_batch.y.min()))
```



https://github.com/huawei-noah/HEBO



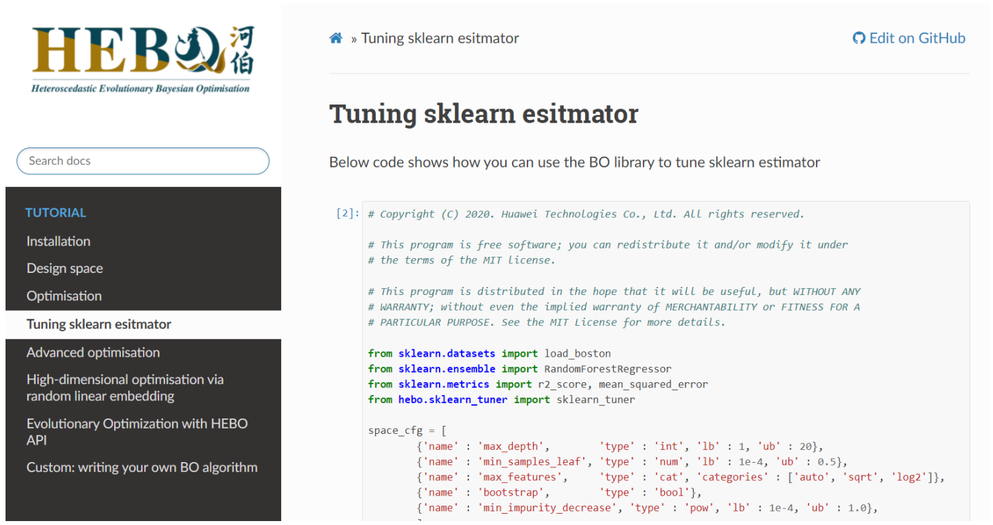
… of course, our methods work as well on hyper-param tasks …

Tuning sklearn estimators

```
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error

from hebo.sklearn_tuner import sklearn_tuner

space_cfg = [
    {'name' : 'max_depth', 'type' : 'int', 'lb' : 1, 'ub' : 20},
    {'name' : 'min_samples_leaf', 'type' : 'num', 'lb' : 1e-4, 'ub' : 0.5},
    {'name' : 'max_features', 'type' : 'cat', 'categories' : ['auto', 'sqrt', 'log2']},
    {'name' : 'bootstrap', 'type' : 'bool'},
    {'name' : 'min_impurity_decrease', 'type' : 'pow', 'lb' : 1e-4, 'ub' : 1.0},
    ]
X, y   = load_boston(return_X_y = True)
result = sklearn_tuner(RandomForestRegressor, space_cfg, X, y, metric = r2_score, max_iter = 16)
```

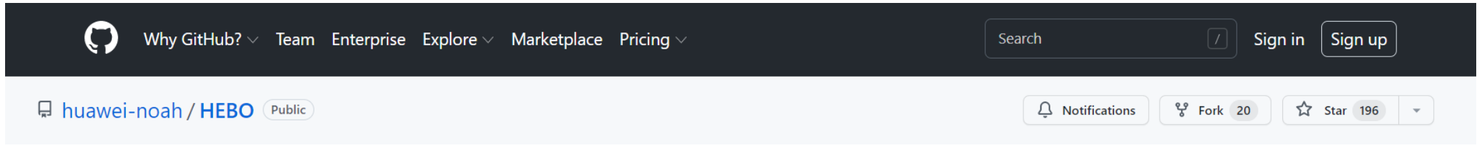
```
Iter 0, best metric: 0.492082
Iter 1, best metric: 0.5526
Iter 2, best metric: 0.5526
Iter 3, best metric: 0.5526
Iter 4, best metric: 0.5526
Iter 5, best metric: 0.715855
Iter 6, best metric: 0.862657
Iter 7, best metric: 0.862657
Iter 8, best metric: 0.862657
Iter 9, best metric: 0.862657
Iter 10, best metric: 0.869785
Iter 11, best metric: 0.882127
Iter 12, best metric: 0.882127
Iter 13, best metric: 0.882127
Iter 14, best metric: 0.882127
Iter 15, best metric: 0.882127
```

```
{'max_depth': 15,
 'min_samples_leaf': 0.00011814573477638075,
 'max_features': 'log2',
 'bootstrap': False,
 'min_impurity_decrease': 0.0001074304107055820 9}
```



https://hebo.readthedocs.io/en/latest/sklearn-tuner.html

https://github.com/huawei-noah/HEBO



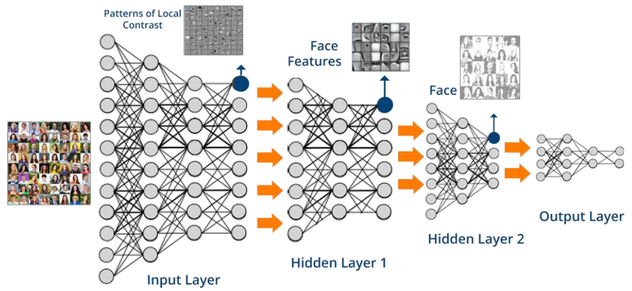

BREAK
TIME!!!

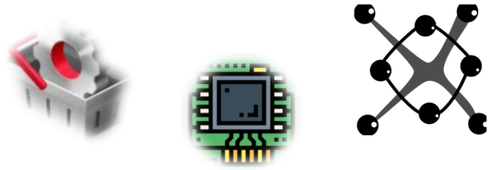
OK! What about other variable types?

In many problems, variables can be discrete (or even) mixed ...

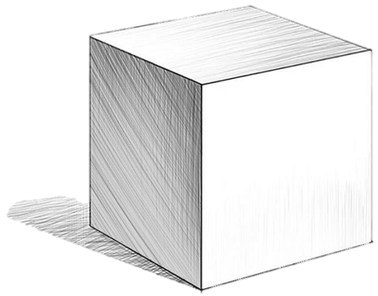
Some examples

- Hyper-parameter tuning (e.g., learning rates and number of layers)
- Any standard combinatorial problem (e.g., SAT solvers, TSPs, and others)
- Some drug design problems (e.g., CDR3 regions on antibodies)

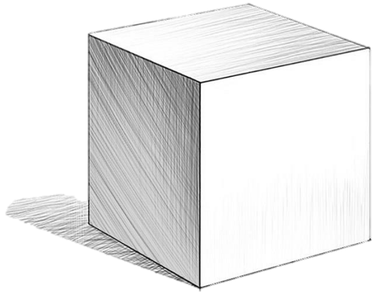




... same as before, but the nature of input space is different ...

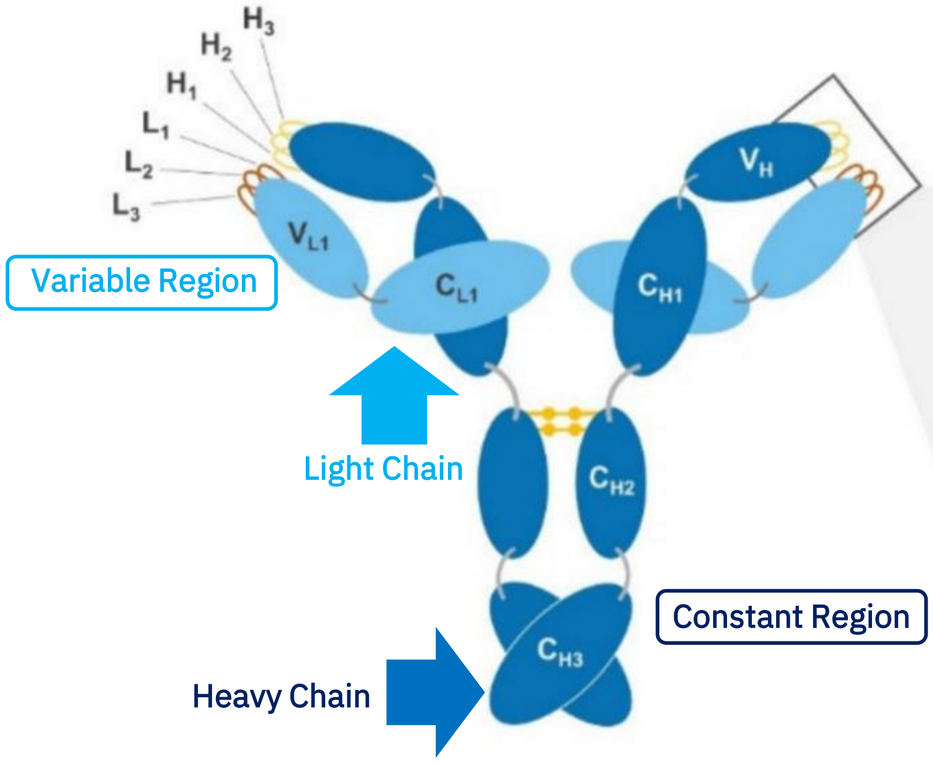


$$\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$



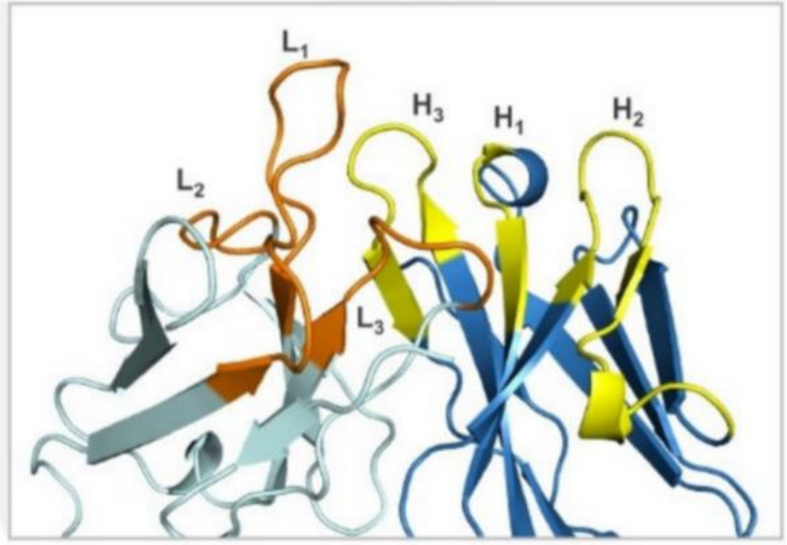
Let us, elaborate such settings with an antibody design example …

… antibodies are a type of protein produced as an immune response to invading pathogens …



… antibody's binding region (paratope) …

6 variable loops (3 loops on light & 3 on heavy)



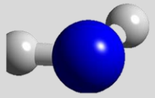
Complementarity-determining region (CDR)

… made of 2 heavy and 2 light chains …

… those are made from amino-acids …

… amino acids are chemical compounds that combine to form proteins (e.g., CDR sequences) …

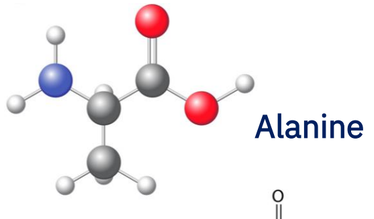
Amino



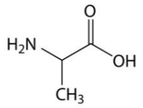
+

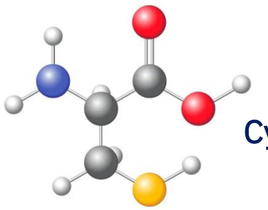
Carboxylic

… contains the following groups …

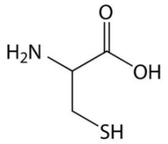


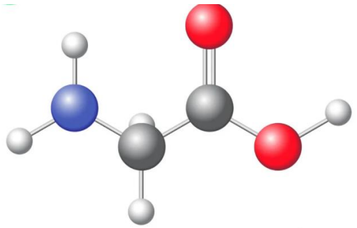
Alanine



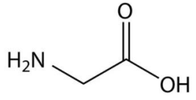


Cysteine

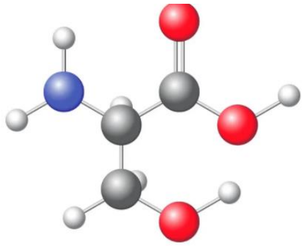




Glycine

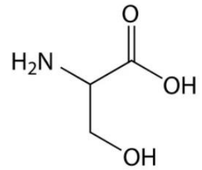


…



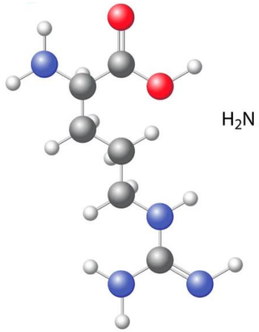
…

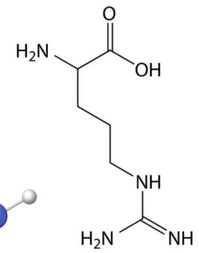
Serine



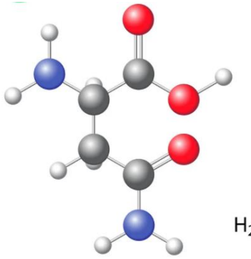
…

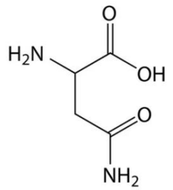
… only 20 – 22 amino acids comprise proteins …



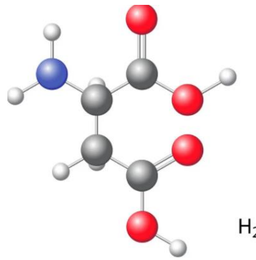


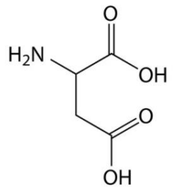
Arginine





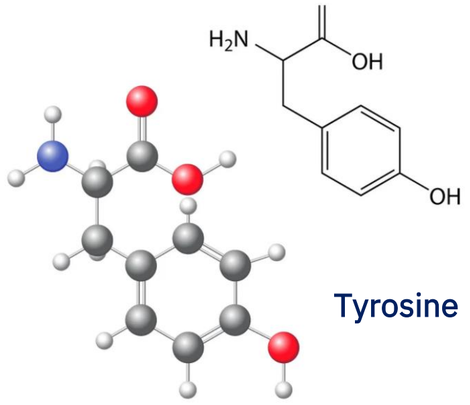
Asparagine





Aspartic acid

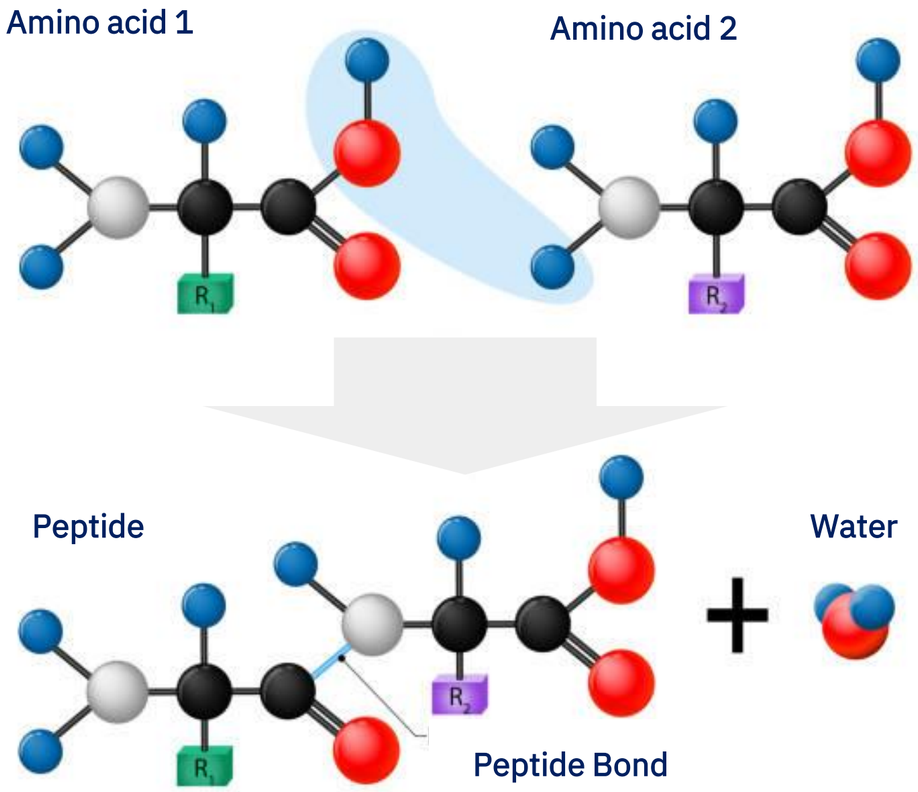
…

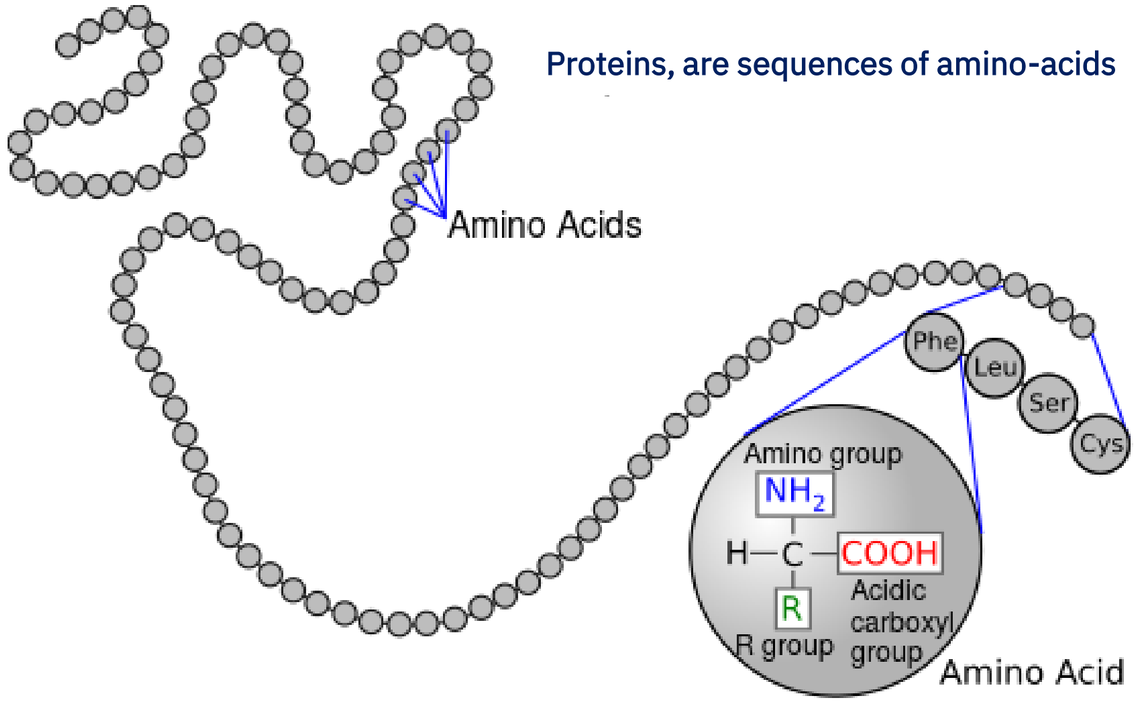


Tyrosine

… that when combining create protiens …

… proteins are linear polymers formed through peptide bonds (or amide bonds) …





… typically contains 50 to 4000 amino-acids …

… given a new antigen, we want to find a **good CDR3 fast.**

Goodness of CDR3, e.g., [negative] binding energy or affinity to the target antigen

CDR3 region – sequence of amino-acids (i.e., a string), e.g., $\mathbf{x} = [\text{'AL'}, \text{'Ag'}, \text{'Ty'}, \cdots]$

$$\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

Allowable CDR3 regions, e.g., repetition on amino-acids, pH levels, hydrophobicity, …

$$\min_{\text{CDR3-Sequence}} \text{Binding-Energy}(\text{CDR3-Sequence}, \text{Antigen}) \quad \text{s.t. Consrtaints}(\text{CDR3-Sequence})$$

… expensive to evaluate binding energies

… not differentiable in optimisation var.

But wait, why can’t we just use our old BO tech as is?

Recalling our pipeline, we need two ingredients to work ...

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

$$\text{Binding-Energy} \sim \mathcal{GP}(m(\text{CDR3-Sequence}), k_{\boldsymbol{\theta}}(\text{CDR3-Sequence}, \text{CDR3-Sequence}'))$$

... some notion of similarity ...

Acquisition Function Maximisation

Given fitted GP, we then maximise acquistions $\|\nabla_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D}, \boldsymbol{\theta})\|$

Not computable ... $\nabla_{\text{CDR3-Sequence}} \alpha(\text{CDR3-Sequence}|\mathcal{D}, \boldsymbol{\theta})$

... discrete GP kernels for combinatorial opt.

Starting with kernels on discrete spaces, we find ...

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

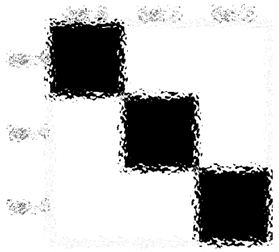
Overlap (counting) kernels:

Dimension/seq length

Simply count overlap $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \frac{\sigma}{d} \sum_{i=1}^{d} l_i \mathbb{1}\{\mathbf{x}[i] = \mathbf{x}'[i]\}$ with $\boldsymbol{\theta} = [\sigma, \mathbf{l}_{1:d}]$ being params. to opt.

Indicator function

Consider the following example:



Only active component

$$\text{CDR-Sequence} = [\texttt{'AL'}, \texttt{'Ag'}, \texttt{'Ty'}] \qquad \text{CDR-Sequence}' = [\texttt{'Gly'}, \texttt{'Al'}, \texttt{'Ty'}]$$

$$k_{\boldsymbol{\theta}}(\text{CDR-Sequence}, \text{CDR-Sequence}') = \frac{\sigma}{d}(l_1 \times 0 + l_2 \times 0 + l_3 \times 1)$$

… another powerful kernel **is the sub-string kernel** …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}] \qquad \text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'},$$

… first, list all **length 1 sub-sequences** …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}] \qquad \text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'},$$

… first, list all **length 1 sub-sequences** …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}] \qquad \text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'},$$

… then, list all **length 2 sub-sequences** …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}] \qquad \text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \{\texttt{'Al'}, \texttt{'Ty'}\},$$

… then, list all **length 2 sub-sequences** …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}] \qquad \text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \{\texttt{'Al'}, \texttt{'Ty'}\}, \{\texttt{'Al'}, \texttt{'Ag'}\}\}$$

… leading us to all subsequences.

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}] \qquad \text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \{\texttt{'Al'}, \texttt{'Ty'}\}, \{\texttt{'Al'}, \texttt{'Ag'}\}\}$$

Now, we shift attention to computing each's contribution ...

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

... length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ ... gap parameter

... all subsequence of a certain length

Working example:

... let's consider this subsequence ...

$$\text{CDR-Sequence} = [\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \texttt{'Gly'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \{\texttt{'Al'}, \texttt{'Ty'}\}, \{\texttt{'Al'}, \texttt{'Ag'}\}\}$$

Seq. of length 2

$$c_{\{\texttt{'Al'},\texttt{'Tl'}\}}(\underbrace{\text{CDR-Sequence}}_{\mathbf{x}}) = \theta_m^2 \theta_g^1$$

a gap of 1

… while in the other CDR-Seq …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

… let's consider this subsequence …

$$\text{CDR-Sequence}' = [\texttt{'Al'}, \texttt{'Ty'}, \texttt{'As'}, \texttt{'Ag'}]$$

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \{\texttt{'Al'}, \texttt{'Ty'}\}, \{\texttt{'Al'}, \texttt{'Ag'}\}\}$$

Seq. of length 2

$$c_{\{\texttt{'Al'},\texttt{'Ty'}\}}(\text{CDR-Sequence}') = \theta_m^2 \theta_g^0$$

a gap of 0

$\mathbf{x}$

… hence, all together this subsequence contributes as …

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Substring kernels:

… length of subsequence parameter

Look at substrings $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$ with $c_{\mathbf{u}}(\mathbf{x}) = \theta_m^{|\mathbf{u}|} \sum_i \theta_g^{\text{gap}(\mathbf{x}_i)}$ … gap parameter

… all subsequence of a certain length

Working example:

… let's consider this subsequence …

$$\mathbf{u} \in \{\texttt{'Al'}, \texttt{'Ag'}, \texttt{'Ty'}, \{\texttt{'Al'}, \texttt{'Ty'}\}, \{\texttt{'Al'}, \texttt{'Ag'}\}\}$$

… it's contribution …

$$c_{\{\texttt{'Al'},\texttt{'Tl'}\}}(\text{CDR-Sequence})\, c_{\{\texttt{'Al'},\texttt{'Ty'}\}}(\text{CDR-Sequence}') = \theta_m^2 \theta_g^1 \theta_m^2 \theta_g^0 = \theta_m^4 \theta_g$$

OK! We can model, but what about acquisition maximisation?

Now, shifting attention to acquisition maximisation ...

Gaussian Process fitting

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

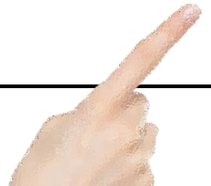
$$\text{Binding-Energy} \sim \mathcal{GP}(m(\text{CDR3-Sequence}), k_{\boldsymbol{\theta}}(\text{CDR3-Sequence}, \text{CDR3-Sequence}'))$$

... some notion of similarity ...

Acquisition Function Maximisation

Given fitted GP, we then maximise acquistions $\|\nabla_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D}, \boldsymbol{\theta})\|$

Not computable ... $\nabla_{\text{CDR3-Sequence}} \alpha(\text{CDR3-Sequence}|\mathcal{D}, \boldsymbol{\theta})$



… acquisition optimisation is a combinatorial problem in itself …

This is a combinatorial optimisation problem, where we can query as much as we want:

Particle Swarms … Simulated Annealing

Genetic Algos

$$\max_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D})$$

… …

Local Search …

… any off-the shelf algorithm …

… but

Global GP fit across the combinatorial space is difficult - (lots of data even if trend existed)

As dimensionalities grow, the (combinatorial) search space becomes notoriously large

Leverage local search in trust regions

Trust-region constraint opt. $\max_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D}) \;\; \text{s.t. } \mathbf{x} \in \text{Trust-Region}(\mathbf{x}_+)$

Best candidate so-far

… locally around best so-far with $\text{Trust-Region}(\mathbf{x}_+) = \left\{ \mathbf{x} \middle| \sum_{i=1}^{d} \mathbb{1}\{\mathbf{x}_i, \mathbf{x}_i^+\} \leq \rho \right\}$

… optimise within the trust-region and adapt its size.

Apply local search within the trust-region and adapt its size based on performance & heuristics:

Global GP fit across the combinatorial space is difficult - (lots of data even if trend existed)

As dimensionalities grow, the (combinatorial) search space becomes notoriously large

Leverage local search **in trust regions**

$$\max_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D}) \quad \text{s.t. } \mathbf{x} \in \text{Trust-Region}(\mathbf{x}_+)$$

Adaptive Trust Region (TR): restrict the search space the current candidate optimum.

- **Increase TR size** when better points are found
- **Decrease TR size** when the optimisation is stuck

Local Search within the trust region

The overall Pipeline of BO-4-Comb problems

Step I: Fit Gaussian Process

$$k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \frac{\sigma}{d} \sum_{i=1}^{d} l_i \mathbb{1}\{\mathbf{x}[i] = \mathbf{x}'[i]\}$$

… use comb. kerns

$$k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{u} \in \Sigma^l} c_{\mathbf{u}}(\mathbf{x}) c_{\mathbf{u}}(\mathbf{x}')$$

Step II: Maximise constraint acquisitions

$$\max_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D}) \quad \text{s.t. } \mathbf{x} \in \text{Trust-Region}(\mathbf{x}_+)$$

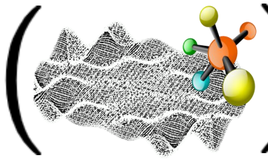
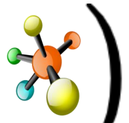
Step III: Query black-box & evaluate new probe

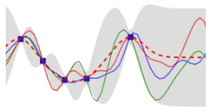
Repeat

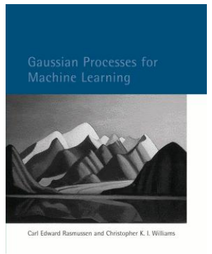
Taking this pipeline to our antibody design problems yields …

Step 1

Efficient model with uncertainty estimates

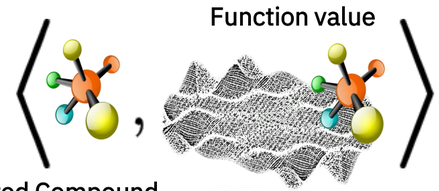
$$p\left(f\left(\text{  }\right)\middle|\ \text{  }\right)$$



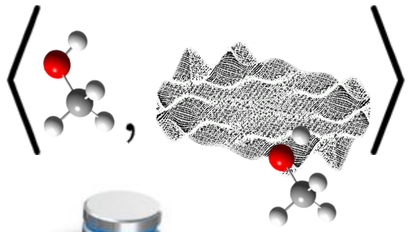


Probabilistic Modelling

Function value

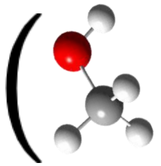


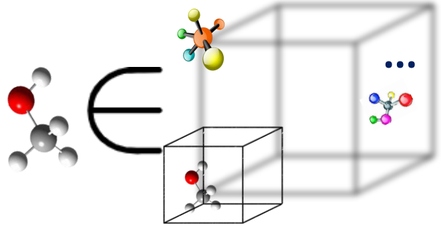
Attempted Compound



Data-Collected So-Far @ some round t

Step 2

$$\max\ \alpha\left(\text{  }\middle|\text{prob. model}\right)$$

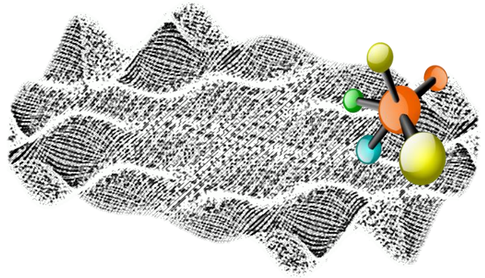


Searching within neighbourhood

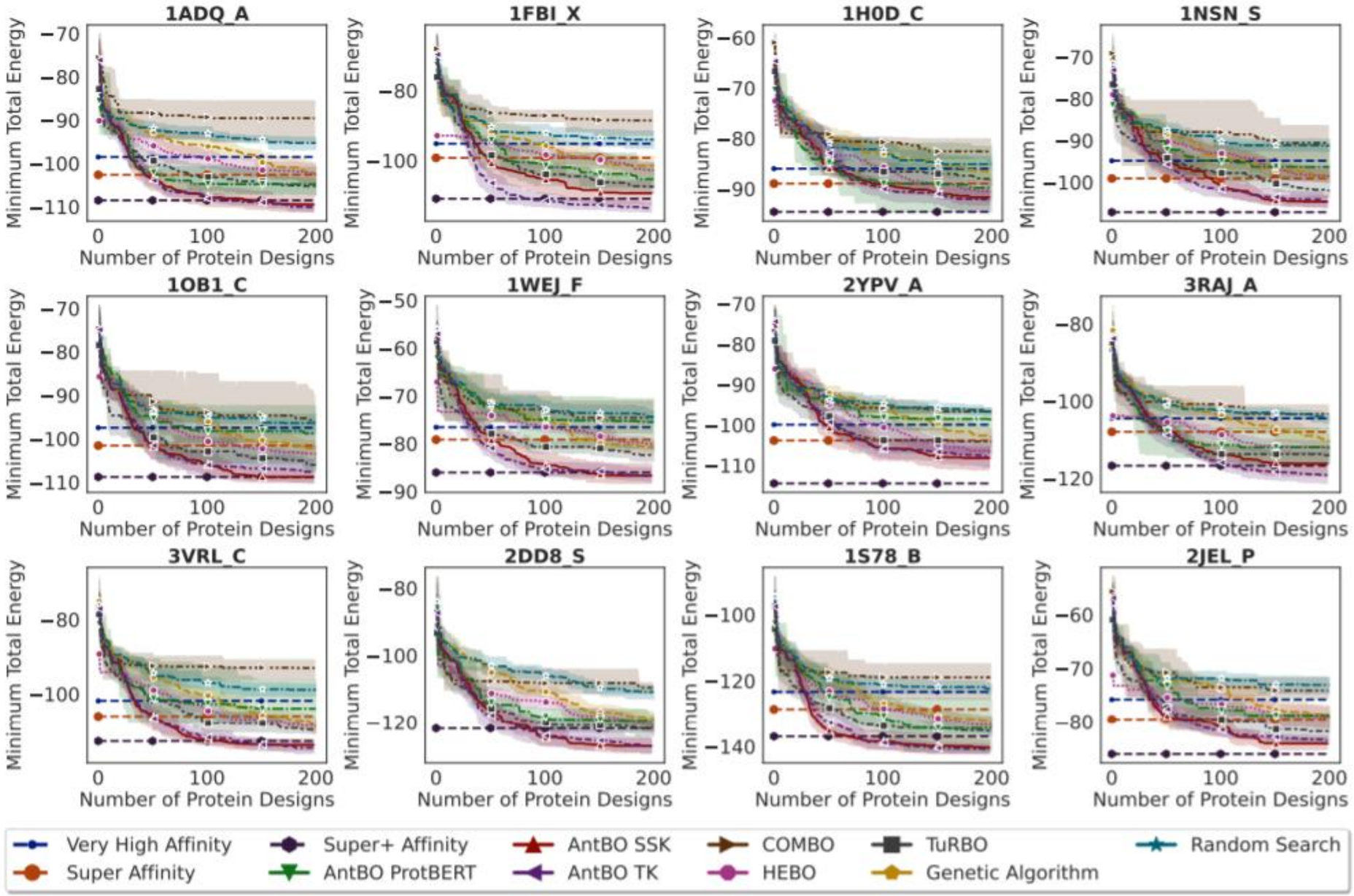
- Trades-off exploration versus exploitation based on the model's uncertainty
- Leads to novel unexplored compounds with potential to success

New compound to test with potential of success under our model

Evaluate, add evaluation to data and repeat



… leading to reduction in number of trials.



Great! We can do discrete opt. what about mixed-variable type?

Again, we need to decide on kernels first …

Gaussian Process fitting

$$\mathbf{x} = [\mathbf{x}_{\text{discrete}}, \mathbf{x}_{\text{cont}}] \qquad f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$$

Combination Kernel:

Product kernel over both spaces

$$k_{\boldsymbol{\theta}}^{\text{joint}}(\mathbf{x}, \mathbf{x}') = \lambda(k_{\boldsymbol{\theta}}^{\text{cont}}(\mathbf{x}_{\text{cont}}, \mathbf{x}'_{\text{cont}}) k_{\boldsymbol{\theta}}^{\text{discrete}}(\mathbf{x}_{\text{discrete}}, \mathbf{x}'_{\text{discrete}}))$$

Any continuous kernel
(e.g., RBF, Matern)

Additive kernel over both spaces

$$+ (1 - \lambda)(k_{\boldsymbol{\theta}}^{\text{cont}}(\mathbf{x}_{\text{cont}}, \mathbf{x}'_{\text{cont}}) + k_{\boldsymbol{\theta}}^{\text{discrete}}(\mathbf{x}_{\text{discrete}}, \mathbf{x}'_{\text{discrete}}))$$

Combination parameter

Any discrete kernel
(e.g., Overlap, SSK)

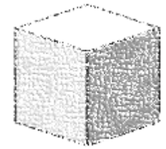
… fit all parameters

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top \left[\mathbf{K}_{\boldsymbol{\theta}}^{\text{joint}} + \sigma_n^2 \mathbf{I}\right]\mathbf{y} - \frac{1}{2}\left|\mathbf{K}_{\boldsymbol{\theta}}^{\text{joint}} + \sigma_n^2 \mathbf{I}\right| - \text{cnst}$$

… when it comes to acquisition optimisation, we will interleave.

Given the GP, we now iterate over both variables to solve the following constraint problem:

… box around best cont. var so-far…



… same as before …

$$\max_{\mathbf{x}_{\text{cont}},\mathbf{x}_{\text{discrete}}} \alpha(\mathbf{x}_{\text{cont}}, \mathbf{x}_{\text{discrete}}|\mathcal{D}) \quad \text{s.t. } \mathbf{x}_{\text{discrete}} \in \text{Trust-Region}(\mathbf{x}^{+}_{\text{discrete}}) \quad \mathbf{x}_{\text{cont}} \in \text{Trust-Region}(\mathbf{x}^{+}_{\text{cont}})$$

… new cont. trust region …

Interleave optimisation between the two types (grads followed by adaptive local search):

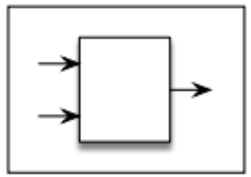
Step I: Fix discrete & grad cont.

$$\mathbf{x}^{[t+1]}_{\text{cont}} = \text{Proj}_{\text{Trust-Region}}[\mathbf{x}^{[t]}_{\text{cont}} + \eta_t \nabla_{\mathbf{x}_{\text{cont}}[t]} \alpha(\mathbf{x}_{\text{cont}}, \mathbf{x}_{\text{discrete}})]$$

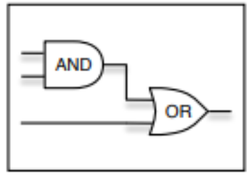
Step II: Given continuous vars, local search discrete

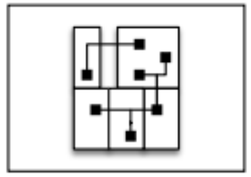
$$\max_{\mathbf{x}_{\text{discrete}}} \alpha(\mathbf{x}_{\text{cont}}{}^{[t+1]}, \mathbf{x}_{\text{discrete}}|\mathcal{D}) \quad \text{s.t. } \mathbf{x}_{\text{discrete}} \in \text{Trust-Region}(\mathbf{x}^{+}_{\text{discrete}})$$

We applied this solution with SSKs to Logic synthesis …

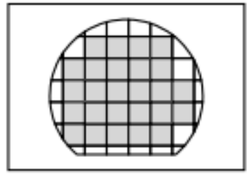


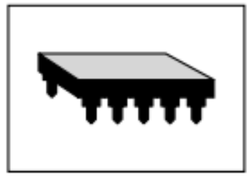
module test
input in[3];
…
endmodule





DRC
LVS
STA





System Specification

Architectural Design

Functional Design and Logic Design (RTL)

Logic Synthesis ✓

Physical Design

Physical Verification and Signoff

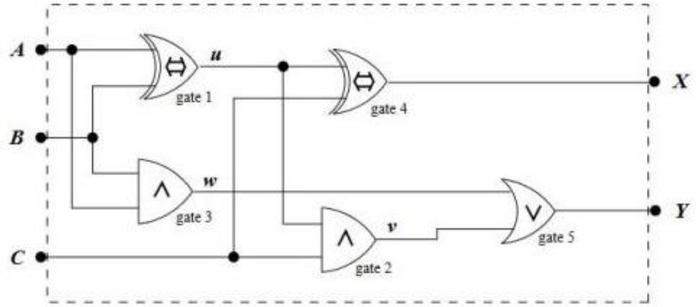
Fabrication

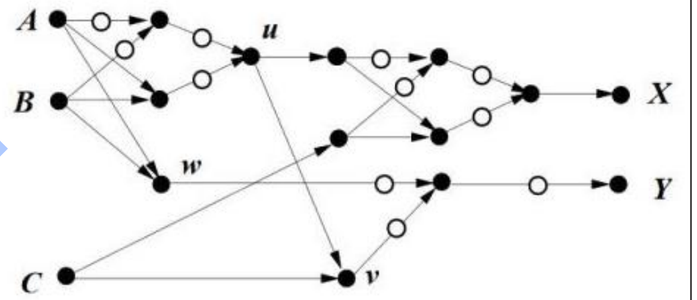
Packaging and Testing

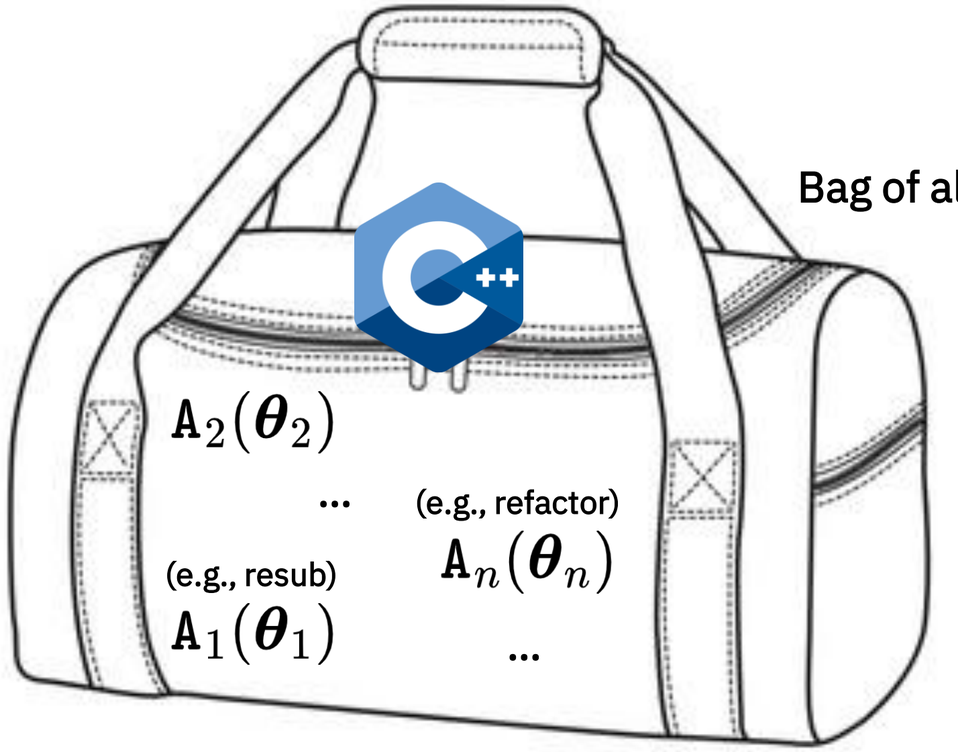
Chip

What's Given

Circuits as AND/Inverted Graphs (AIGs)







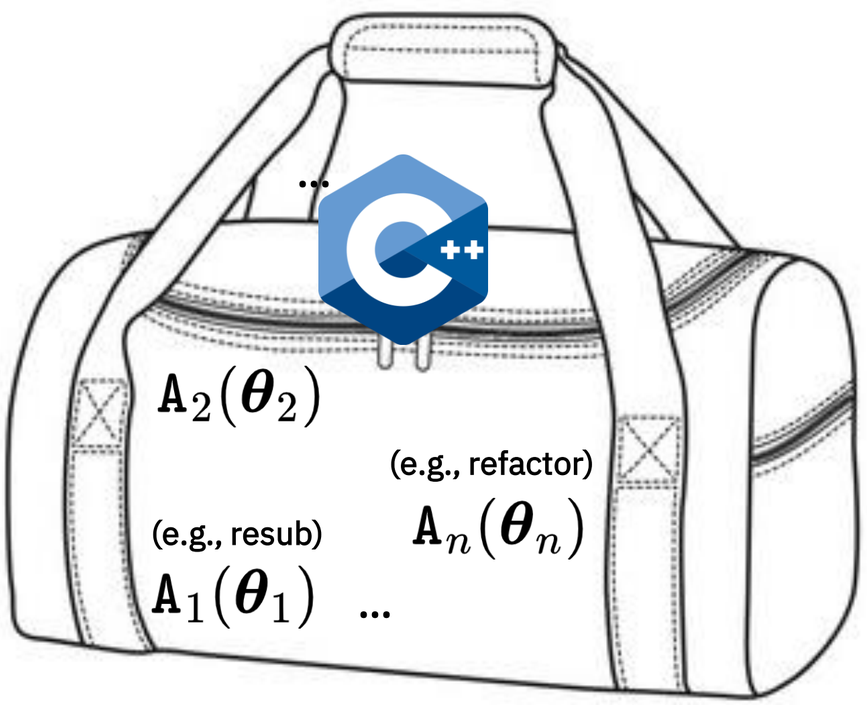
Bag of algorithms (operators)

… digging a bit deeper into an example operator…

```
abc 01> refactor -h
usage: refactor [-N <num>] [-lzvh]
                  performs technology-independent refactoring of the AIG
        -N <num> : the max support of the collapsed node [default = 10]
        -l       : toggle preserving the number of levels [default = yes]
        -z       : toggle using zero-cost replacements [default = no]
```

$\boldsymbol{\theta}_n$

… can be mixed/cont. or both …




```
int Abc_NtkRefactor( Abc_Ntk_t * pNtk, int nNodeSizeMax, int nConeSizeMax, int fUpdateLevel, int fUseZeros, int fUseDcs, int fVerbose )
{
    extern void          Dec_GraphUpdateNetwork( Abc_Obj_t * pRoot, Dec_Graph_t * pGraph, int fUpdateLevel, int nGain );
    ProgressBar * pProgress;
    Abc_ManRef_t * pManRef;
    Abc_ManCut_t * pManCut;
    Dec_Graph_t * pFForm;
    Vec_Ptr_t * vFanins;
    Abc_Obj_t * pNode;
    abctime clk, clkStart = Abc_Clock();
    int i, nNodes;

    assert( Abc_NtkIsStrash(pNtk) );
    // cleanup the AIG
    Abc_AigCleanup((Abc_Aig_t *)pNtk->pManFunc);
    // start the managers
    pManCut = Abc_NtkManCutStart( nNodeSizeMax, nConeSizeMax,  nNodeFanStop: 2,  nConeFanStop: 1000 );
    pManRef = Abc_NtkManRefStart( nNodeSizeMax, nConeSizeMax, fUseDcs, fVerbose );
    pManRef->vLeaves   = Abc_NtkManCutReadCutLarge( pManCut );
    // compute the reverse levels if level update is requested
    if ( fUpdateLevel )
        Abc_NtkStartReverseLevels( pNtk,  nMaxLevelIncrease: 0 );
```

$\mathbf{A}_n(\boldsymbol{\theta}_n)$

… and now, **our goal is to find a good sequence of algos.** …

optimal sequence

performance measure

$$\mathbf{seq}^{\star} \in \arg\max_{\mathtt{Alg},\boldsymbol{\Theta}} \mathrm{QoR}(\mathbf{seq})$$

Represents improvements in area & delay

ranges, allowable sequences

$$\mathtt{A}_1(\boldsymbol{\theta}_1)\,\mathtt{A}_n(\boldsymbol{\theta}_n)\cdots\mathtt{A}_2(\boldsymbol{\theta}_2)$$

$$\text{s.t.}\,\mathrm{Constraints}(\mathbf{seq})$$

Optimise operation parameters

QoR computation can become very slow (especially in real-world validations)

QoR computation involves nesting of algorithms (i.e., hard to get a closed form)

QoR function is NOT differentiable with respect to sequences and parameters

Constraints are involve complex algorithmic checking (i.e., also hard to get closed form)

… on EPFL benchmark leadership board …

EPFL Combinational Benchmark Suite: "The EDA community heavily relies on public benchmarks to evaluate the performance of academic and commercial design tools."

Introduced in 2015: **defines a comparative standard for the logic optimisation community.**

Provides a set of 23 logic circuits and **reports the best circuit mappings** found so far **using any technique** (latest update: **March, 2022**) in 2 types of leaderboards:

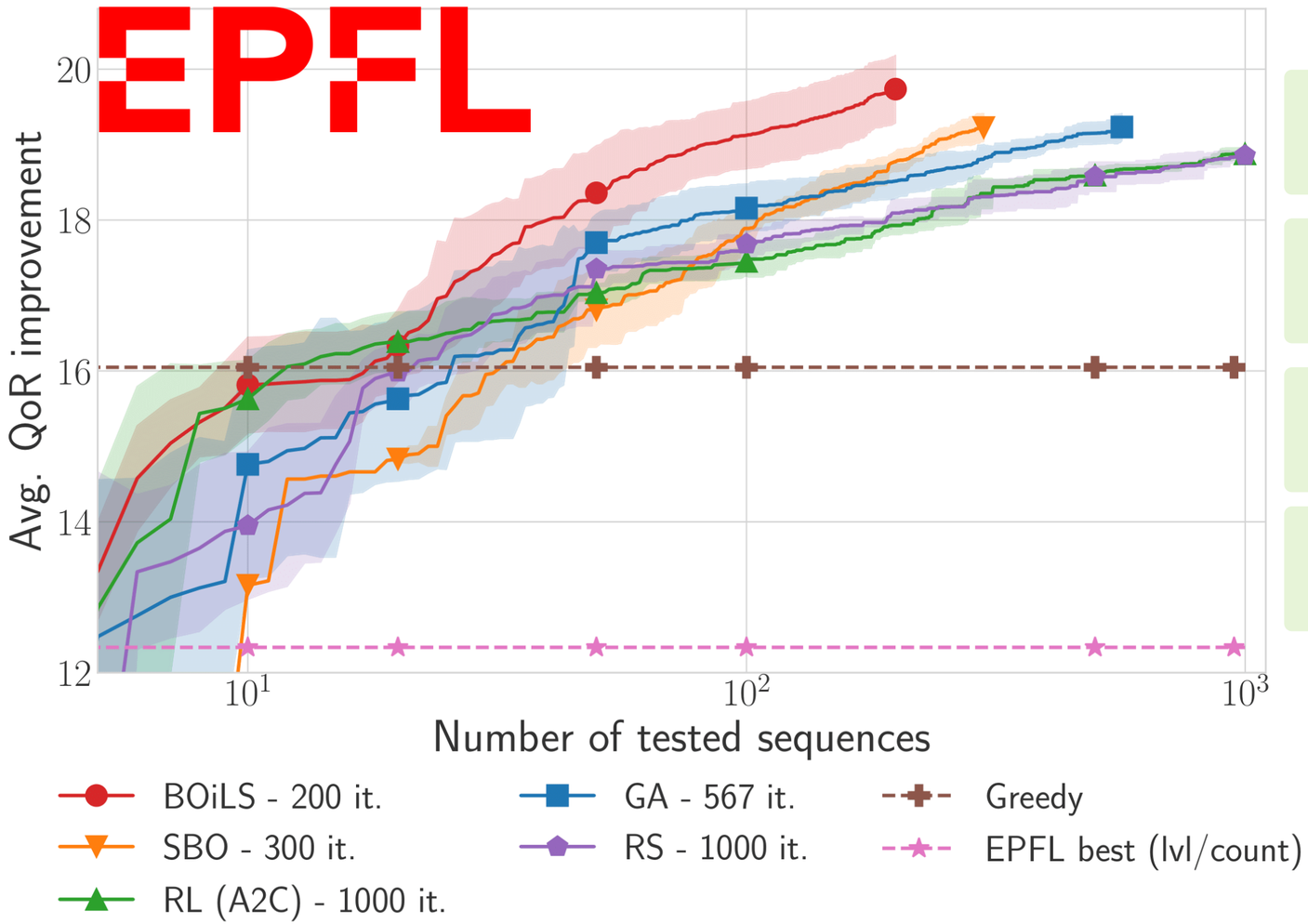
- Leaderboard for **best LUT count (=Area)** (compare Level in case of tie in LUT count)
- Leaderboard for **best Level (=Delay)** (compare LUT count in case of tie in Level)

In March, 2022: HIMap and Support-Reducing Decomp were the methods obtaining the largest numbers of SOTA mappings (8 best mappings each)!

Method	Nb. of best LUT count	Nb. of best Level	Total Nb. Best
HIMap	5	3	8
Support-Reducing Decomp	4	4	8
scaleSyn mapper	2	3	5
LUT-optimization	5	0	5
iMap	0	5	5
ABC Extreme Mapper	2	2	4
Original	2	2	4
XAG optimization	0	3	3
PIMap	1	1	2
Boolean Methods	1	0	1
Resubstitution	1	0	1

… we arrived at a sample efficient solver …

$$\mathtt{seq}^\star \in \arg\max_{\mathtt{Alg},\Theta} \mathrm{QoR}(\mathtt{seq}) \quad \mathrm{s.t.Constraints}(\mathtt{seq})$$



Fully **automated solver (no need for heuristics)**

5x sample complexity reduction w.r.t. to SOTA RL

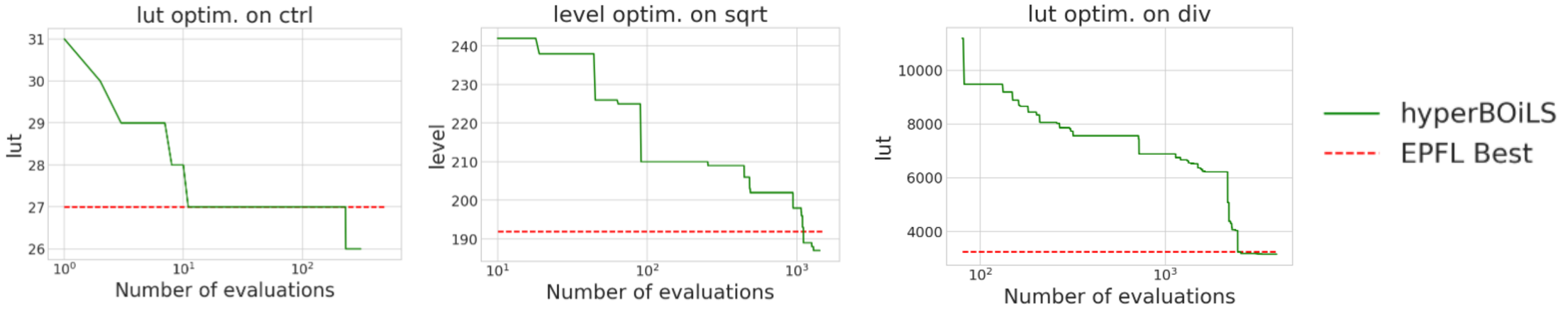
2.8x reduction w.r.t. to Genetic & evolutionary algos.

1.5x reduction w.r.t. to standard Bayesian Opt.

… yielding SOTA results so-far.

Run our algorithms on EPFL benchmark circuits to optimise LUT count or Level

Examples of regret curves obtained when optimising Level (for ‘sqrt’ and ‘ctrl’), and LUT-count (for ‘div’):



→ Our method recovers 12 of the best LUT / Level and found 9 new SOTA circuit mappings making it the method providing the largest number of SOTA circuit mappings

Method	Nb. of best LUT count	Nb. of best Level	Total Nb. Best
HIMap	5	3	8
Support-Reducing Decomp	4	4	8
scaleSyn mapper	2	3	5
LUT-optimization	5	0	5
iMap	0	5	5
ABC Extreme Mapper	2	2	4
Original	2	2	4
XAG optimization	0	3	3
PIMap	1	1	2
Boolean Methods	1	0	1
Resubstitution	1	0	1

Method	Nb. of best LUT count	Nb. of best Level	Total Nb. Best
hyperBOiLS	5	4	9
Support-Reducing Decomp	4	4	8
HIMap	3	2	5
LUT-optimization	5	0	5
ABC Extreme Mapper	2	2	4
Original	2	2	4
scaleSyn mapper	0	3	3
iMap	0	3	3
XAG optimization	0	3	3
Boolean Methods	1	0	1
Resubstitution	1	0	1

New best results #15

Open AntGro wants to merge 66 commits into lsils:master from AntGro:master


BREAK
TIME!!!

OK! GPs are great but slow, can we do better?

Though successful, **our previous solution suffers from** …

Step I: Learn Bayesian Models (Time & memory of GPs)

Standard GP training $\mathcal{O}(n^3)$

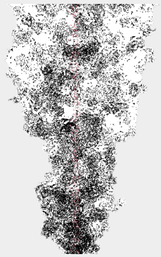
Standard GP prediction (variances) $\mathcal{O}(n^2)$

High-time demands

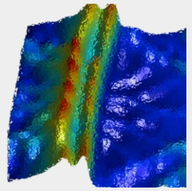
Standard GP memory $\mathcal{O}(nd + n^2)$

High-memory demands

Step I: … flexibility of GPs

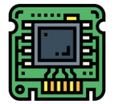


Heteroscedasticity



Non-smoothness

… resolve for more effective and efficient BO …



(Time & memory of GPs) + flexibility of GPs

Bayesian Optimisation **v.2.0**

未 来

Our strategy is to stand-on shoulders of deep nets, but ...

Direction I: Bayesian Neural Networks (Priors on weights & Units)

Flexible in priors & can scale

Needs access to GP-BO to tune
Highly sample intensive

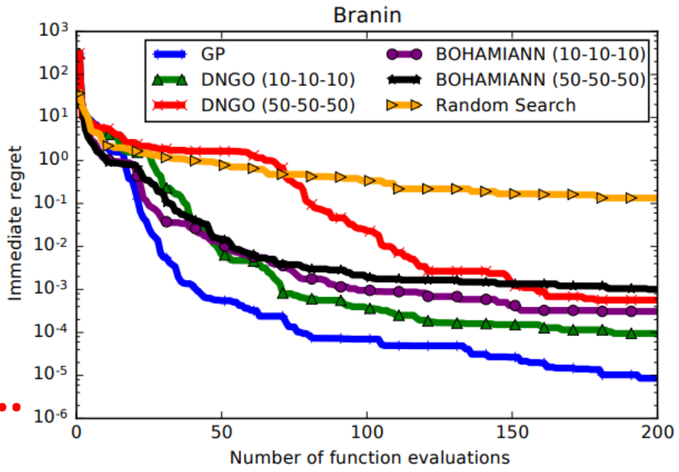
Direction II: Neural Processes (Flexible models)

Flexible in priors & can scale

Unstable end-2-end training
Highly sample intensive

Experiment	Gaussian Process	DNGO	BOHAMIANN	NPBO
Branin	**0.3996**	0.4019	**0.3979**	**0.3980**
Camelback	**−1.011**	**−1.026**	**−1.027**	−0.9999
Hartmann3	−1.028	**−3.862**	**−3.861**	**−3.498**
Forrester	**−6.021**	−5.846	**−6.021**	−5.301
GoldsteinPrice	**4.916**	**6.379**	11.39	**8.654**
Hartmann6	**−3.255**	**−3.249**	**−3.264**	−3.214
SinOne	**0.04292**	**0.04292**	0.04292	**0.04292**

... on classic benchmarks GPs are still the best ...

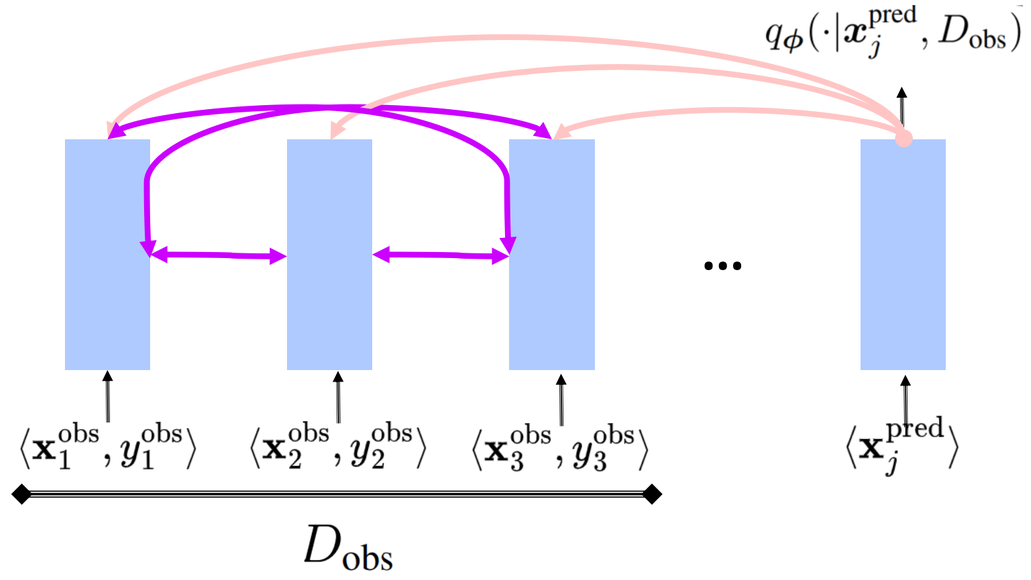


… we develop a new model based on **prob. transformers** …

Standard cross-entropy loss (model-output dists.)

$$\mathcal{J}(\phi) = \mathbb{E}_{(\langle \boldsymbol{x}_j^{\text{pred}}, y_j^{\text{pred}} \rangle, D_{\text{obs}}) \sim p(\mathcal{D})} [-\log q_\phi(\cdot | \boldsymbol{x}_j^{\text{pred}}, D_{\text{obs}})]$$

Predictive distribution



Prior-data sets **not related to black-box**

$$p(D) = \mathcal{HGP}(\cdot) \quad p(D) = \mathcal{BNN}(\cdot) \quad \ldots$$

Staying sample-efficient to opt. problem during training

[Transformer can do Bayesian inference, Muller et. al 2022]

… as is, this model, however, fails BO in two aspects.

P.I: Uniform training and non-uniform predictions hurt model accuracies!

… improve training procedure – BO-tailored masking …

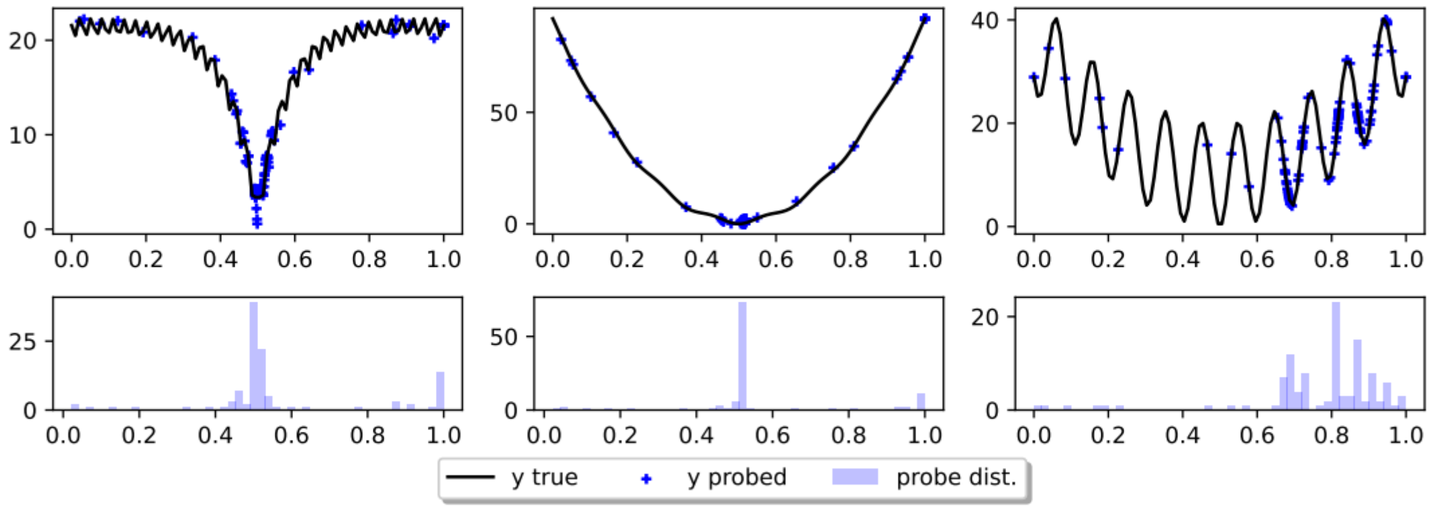


Figure 1: GP predictions and BO run probed point distributions on Ackley, Griewank and Rastrigin in 1 dimension. Clearly, probed points during BO are far from being uniform in the search space.

P.II: Sensitivity to unexplored regions is low, increasing sampling complexities!

… introduce regularisers inducing favourable inductive biases …

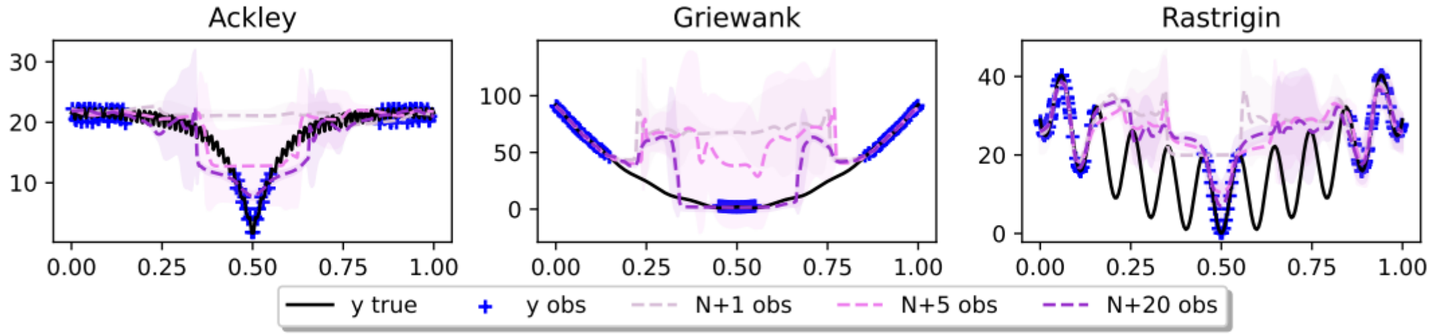


Figure 2: PT predictive distribution with growing number of observations in unexplored regions. As we can see, the PT does change its posterior prediction but only upon observing many probes.

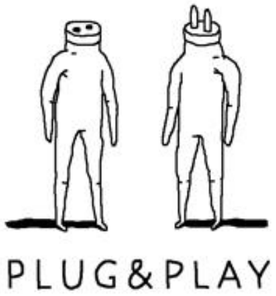
… we tackle P.I and P.II as follows …

P.I: Uniform training and non-uniform predictions hurt model accuracies!

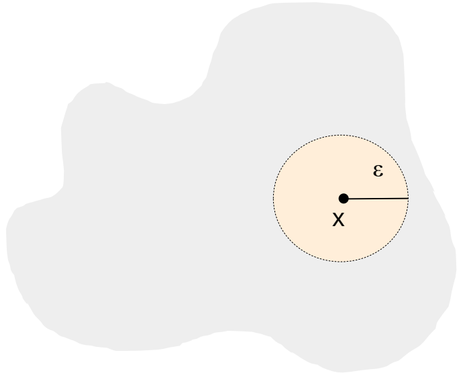
… improve training procedure – BO-tailored masking …

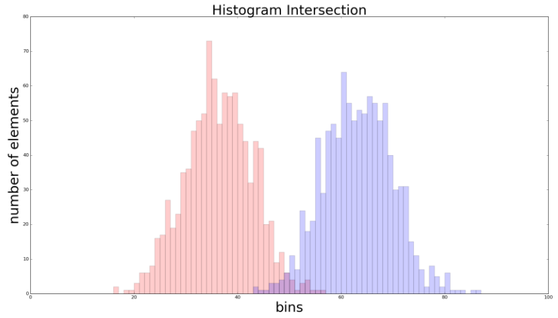
Simple Plug-n-Play Solution

$$\text{add pt. } \langle \mathbf{x}_i, y_i \rangle \; D_{\text{obs}} \text{ w.p. } \propto e^{y_i} / \sum_{n=1}^{N} e^{y_n}$$



… non-uniform distribution prop to y-vals …





P.II: Sensitivity to unexplored regions is low, increasing sampling complexities!

… introduce regularisers inducing favourable inductive biases …

Stationary Inductive Biases (close inputs, close output behaviours)

$$+ \sum_{\boldsymbol{x}_i^{\text{pred}} \in \mathcal{B}_\epsilon(\boldsymbol{x}_j^{\text{pred}})} \omega(\boldsymbol{x}_j^{\text{pred}}, \boldsymbol{x}_i^{\text{pred}}) \text{KL}(q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}_j^{\text{pred}}, D_{\text{obs}}) || q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}_i^{\text{pred}}, D_{\text{obs}}))$$

… new regulariser inducing stationarity inductive biases

… hence our overall BO-pipeline involves the following steps.

Step I: Sample non-uniform data training for a prior

$$\text{add pt. } \langle \mathbf{x}_i, y_i \rangle \; D_{\text{obs}} \text{ w.p. } \propto e^{y_i} / \sum_{n=1}^{N} e^{y_n}$$

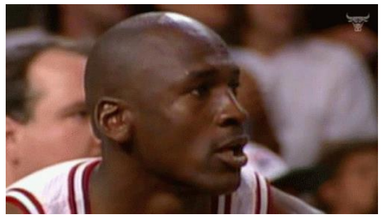
… no black-box data needed …

Step II: Train using cross-entropy and our regulariser

$$\mathcal{J}(\phi) + \text{Reg}(\phi)$$

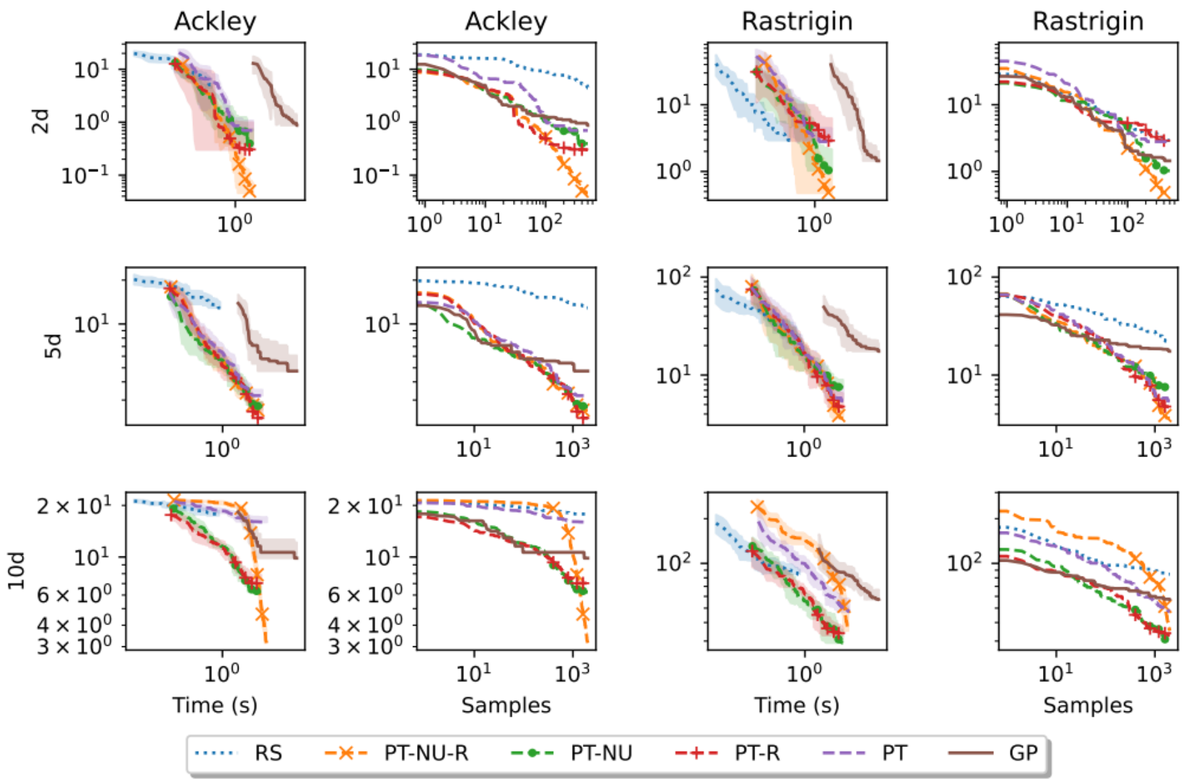
… no fine-tuning needed (one model rules all!!) …

Step III: Guide exploration using this surrogate



… on benchmark tasks, we beat GPs and are 1 order faster …

Upon running on all BO-Torch benchmarks, we demonstrate ...



Dimension	PT	PT-$\mathcal{R}_\varepsilon$	PT-ν	PT-$\nu\mathcal{R}_\varepsilon$	GP	# Tasks
1	20.5	88.1	79.2	73.6	**93.6**	6
2	103.3	112.4	111.8	**114.3**	111	16
5	73	77	74.7	**78**	65.8	8
10	36	71.1	71.8	**72.1**	63.8	8

Table 6: Approximate total running times, including training for PT methods (hours).

Process	1D (6 tasks)	2D (16 tasks)	5D (8 tasks)	10D (8 tasks)
PT-$\nu\mathcal{R}_\varepsilon$-BO	3.1	3.2	5.7	6.9
GP-BO	3.2	4.7	21.4	26.2

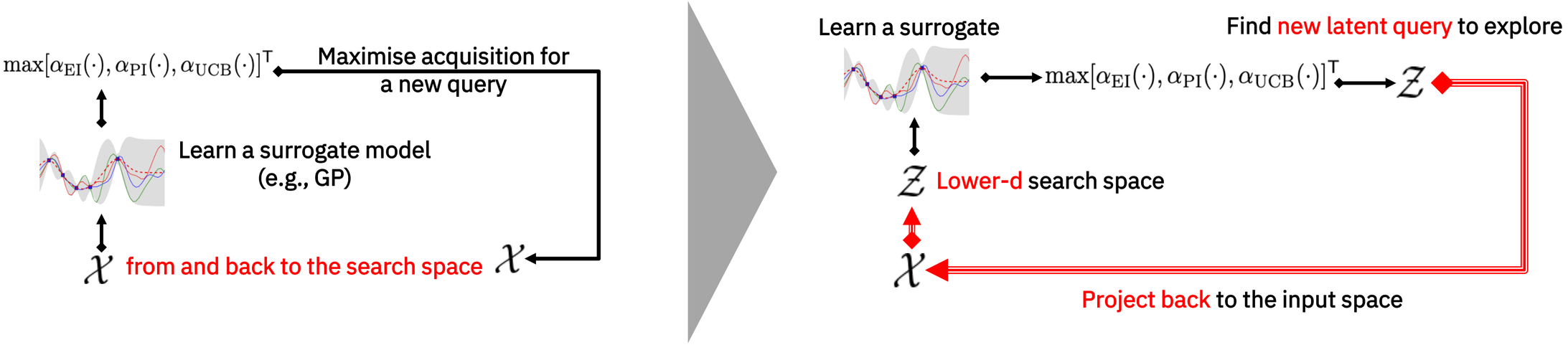
Table 1: Algorithms notation.

Notation	Components
RS	Random Search
GP	Gaussian Process BO
PT	[30] unchanged
PT-$\mathcal{R}_\varepsilon$	PT with Regularised Training (Equation 3)
PT-ν	PT with Non-Unif. Inputs Prior
PT-$\nu\mathcal{R}_\varepsilon$	PT with Non-Unif. Inputs Prior & with Regularised Training


BREAK
TIME!!!

OK! Off to high-d worlds.
How to scale BO beyond low-dimensional domains?

Almost all techniques involve a dimensionality reduction step ...



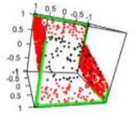
... learning effective latent space is critical to the success differentiating many algorithmic forms ...

Linear Projection Techniques

Nonlinear Projection Techniques

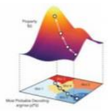

Wang et al. 2016


Lethman et al. 2020


Qian et al. 2016

Binois et al. 2020


Jin al. 2019

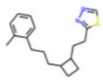

Trippe al. 2020

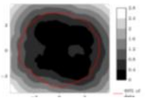
Moriconi et al. 2020

Gomez-B. et al. 2018

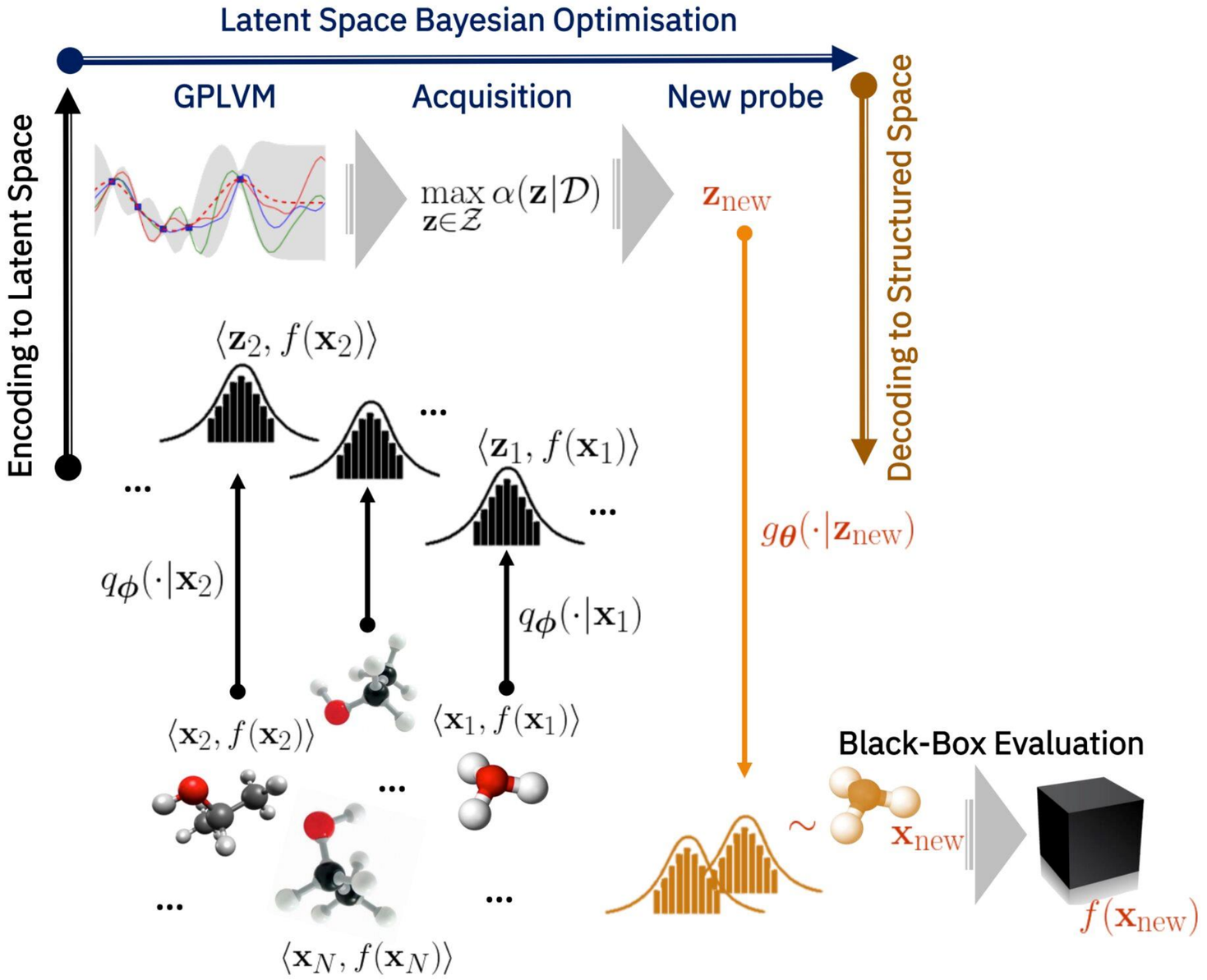
Siivola et al. 2020


Griffiths et al. 2020

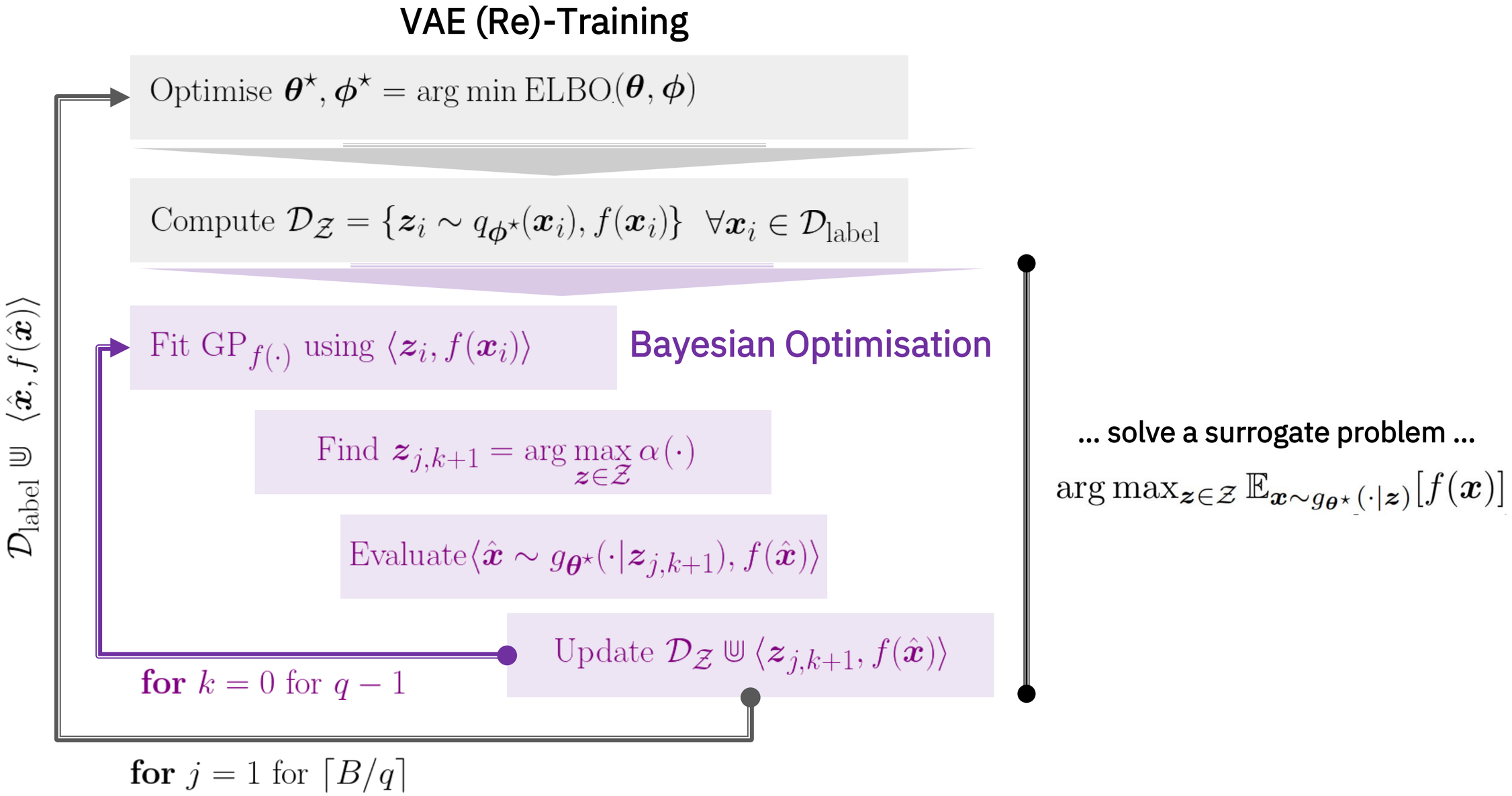

Kusner al. 2017


Mutner et al. 2019

… we focus on those using VAEs for latent spaces …

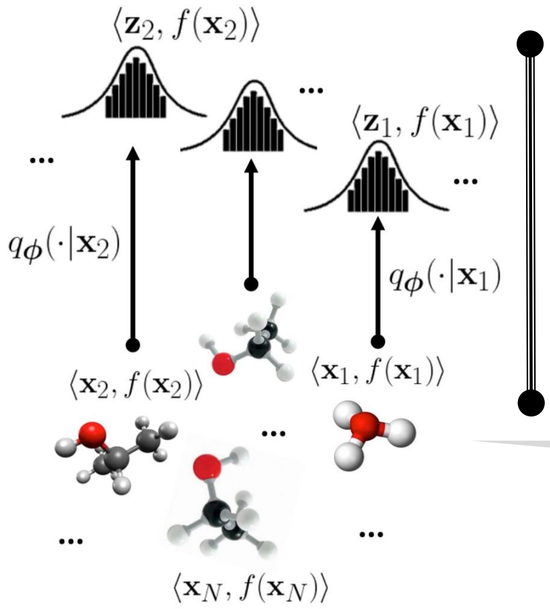


… that exhibit a nesting of VAE training and Bayesian Opt.



Hold on! What about those latent features?

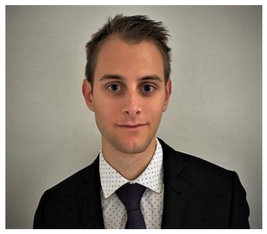
Latent features need to generalise beyond reconstruction ...



Tripp et al. 2020 $w(\boldsymbol{x}_i) \propto f(\boldsymbol{x}_i) \longrightarrow \mathbb{E}_{\boldsymbol{x}\sim\mathcal{U}(\cdot)}[\mathbf{ELBO}_{\text{weighted}}(\boldsymbol{\theta}, \boldsymbol{\phi})] = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{U}(\cdot)}[w(\boldsymbol{x})\mathbf{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi})]$

Property I: Label guidance for informative feature spaces reflective of the black-box

Property II: Useful features facilitating Gaussian process training in the latent space



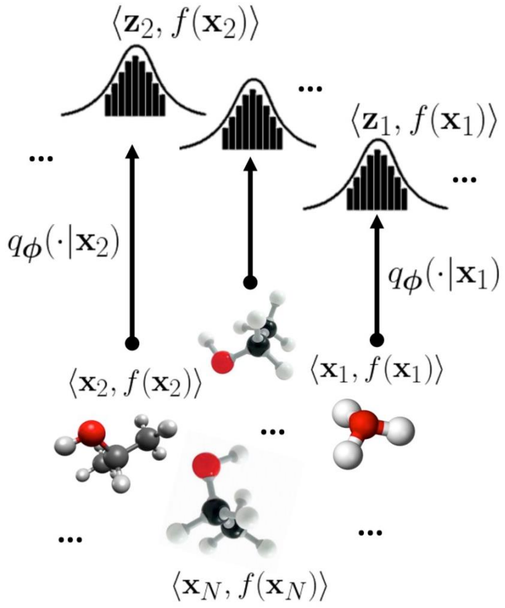
Antoine Gronsit

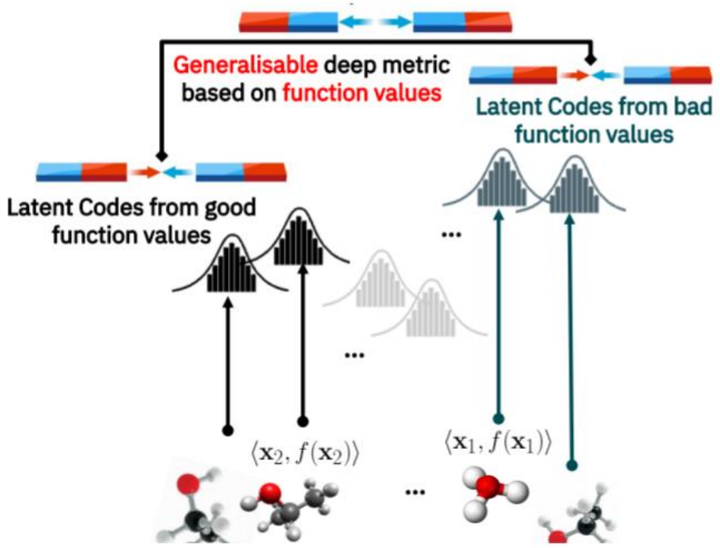
Alexandre Maraval

Ali Cowen Rivers

Rasul Tutunov

Ryan Griffins





... combine weighted ELBOs with smooth metric losses ...

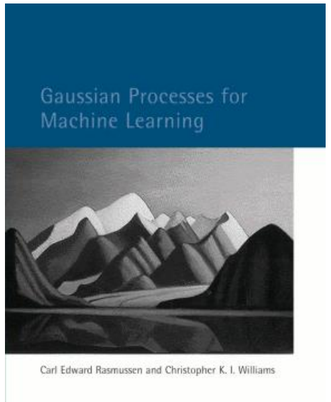
… when training the VAE, we know GPs are coming next …

Can we encourage such an inductive property in the latent space?

Build directly into the GP

Incorporate in the VAE's ELBO

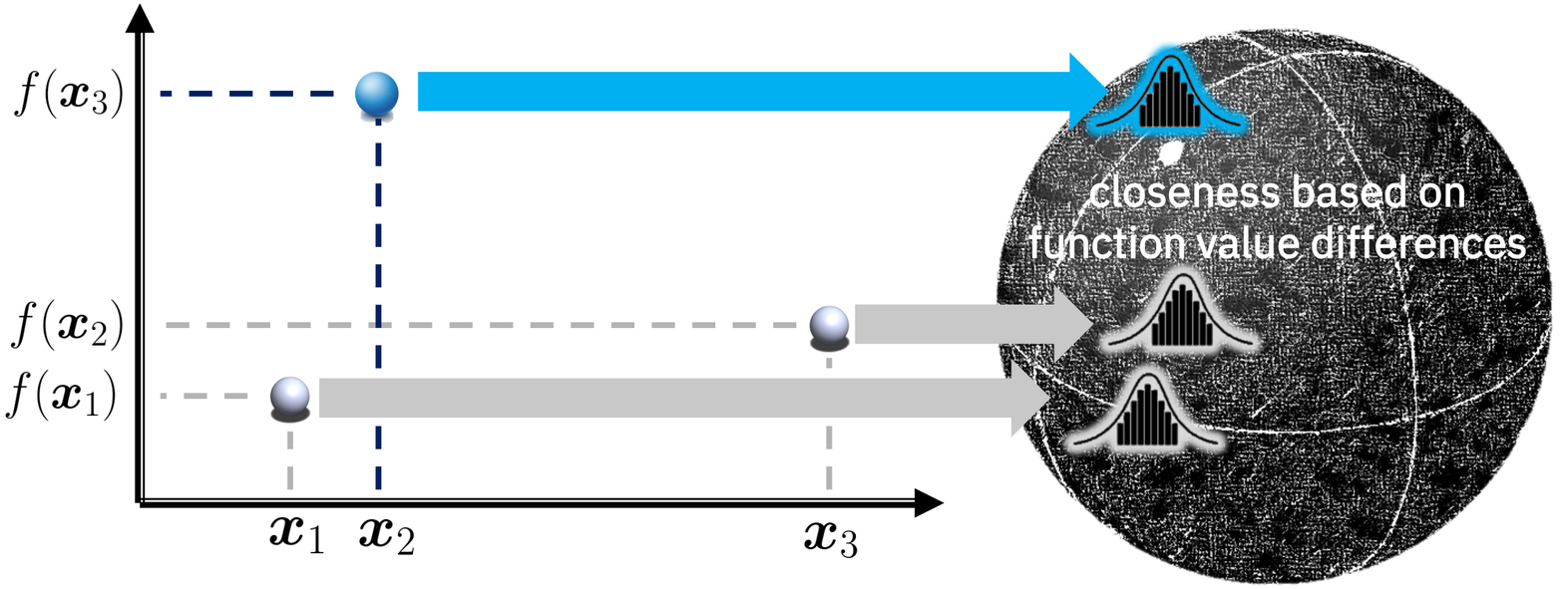
Exciting direction for future work

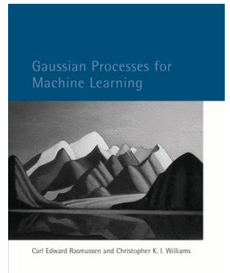


However, for all their variety, supervised learning algorithms are based on the idea that similar input patterns will usually give rise to similar outputs (or output distributions), and it is the precise notion of similarity that differentiates the algorithms. For example some algorithms may do feature selection and decide that there are input dimensions that are irrelevant to the predictive task. Some algorithms may construct new features out of those provided and measure similarity in this derived space. As we have seen, many regression techniques can be seen as linear smoothers (see section 2.6) and these techniques vary in the definition of the weight function that is used.

… connects to deep metric learning (e.g., contrastive and triplet losses) …

… ideally, we require a latent structure that preserves …





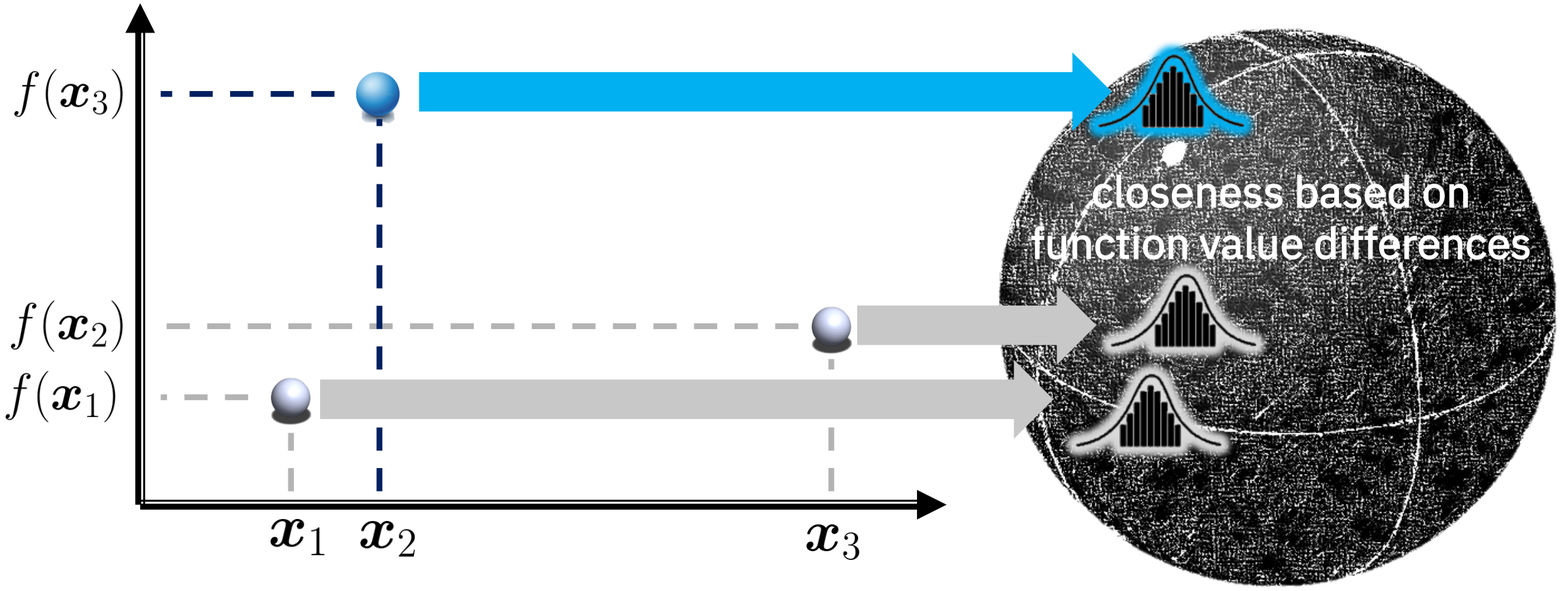
easier for GPs since close latent codes correspond to close function values

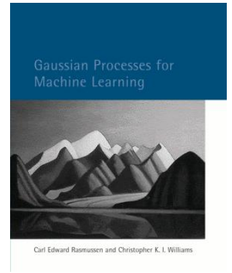
… we can easily transform this intuition into a loss function as follows …

$$|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)| \leq r \;\Rightarrow\; \boldsymbol{z}_i \text{ and } \boldsymbol{z}_j \text{ should be close} \;\Rightarrow\; \mathcal{L}(\cdot) \propto ||\boldsymbol{z}_i - \boldsymbol{z}_j||_q$$

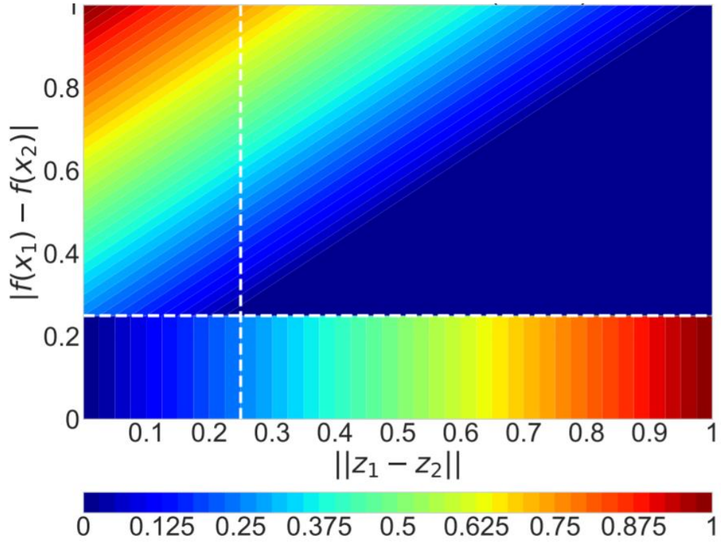
$$|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)| > r \;\Rightarrow\; \boldsymbol{z}_i \text{ and } \boldsymbol{z}_j \text{ should seperate by a margin} \;\Rightarrow\; \mathcal{L}(\cdot) \propto \max\{0, \rho - ||\boldsymbol{z}_i - \boldsymbol{z}_j||_q\}$$

… we also smoothen the loss for more effective spaces …

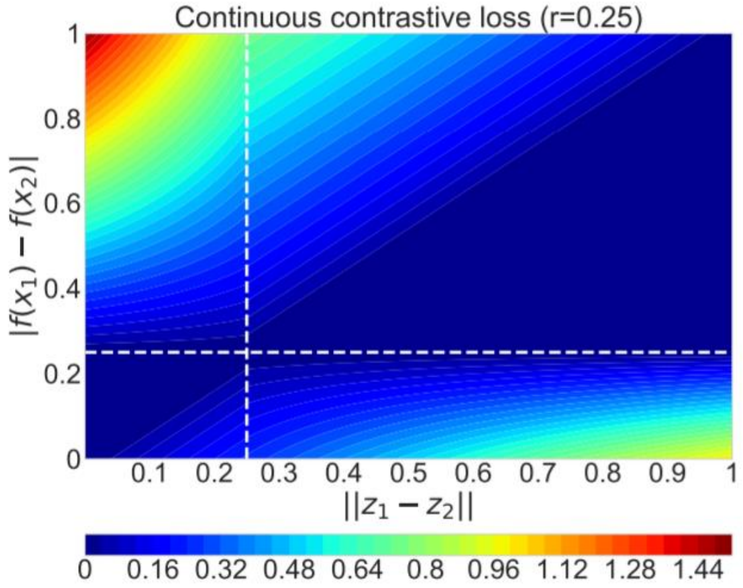




easier for GPs since close latent codes correspond to close function values

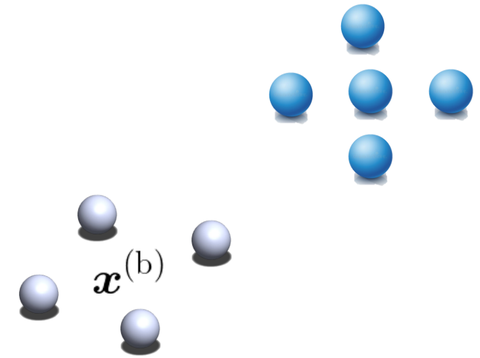


Smoothening



… we could have done the same but with triple points instead.

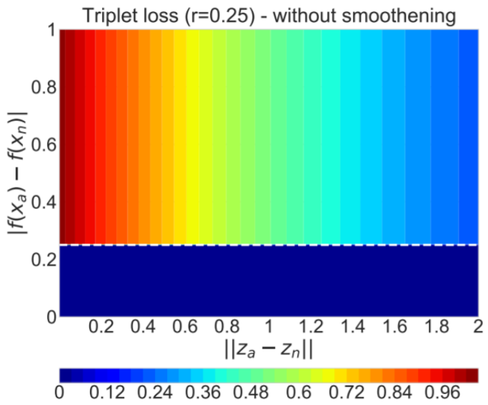
$$\mathcal{D}_{\mathrm{n}}(\boldsymbol{x}^{(\mathrm{b})}; r) = \langle \boldsymbol{x} \in \mathcal{D} : |f(\boldsymbol{x}^{(\mathrm{b})}) - f(\boldsymbol{x})| \geq r \rangle$$

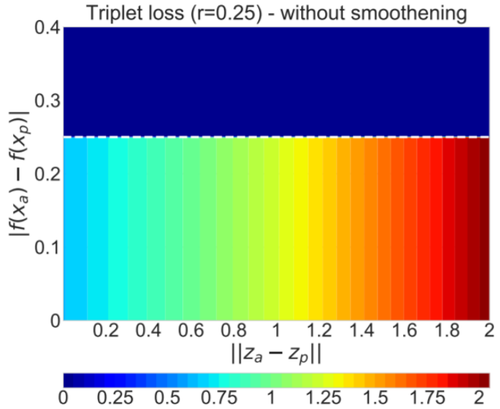


$$\mathcal{D}_{\mathrm{p}}(\boldsymbol{x}^{(\mathrm{b})}; r) = \langle \boldsymbol{x} \in \mathcal{D} : |f(\boldsymbol{x}^{(\mathrm{b})}) - f(\boldsymbol{x})| < r \rangle$$

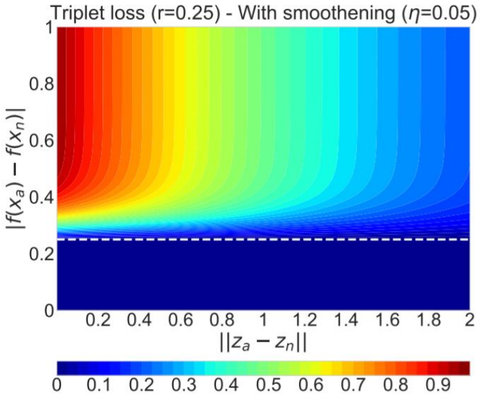
$$\mathcal{L}^{(\mathrm{BO})}_{\mathrm{triple}}(\cdot) \propto \log(1 + \exp(\Delta^{+}_{\boldsymbol{z}} - \Delta^{-}_{\boldsymbol{z}})) \text{ with } \Delta^{+}_{\boldsymbol{z}} = ||\boldsymbol{z}^{(\mathrm{b})} - \boldsymbol{z}^{(\mathrm{p})}||_q \text{ and } \Delta^{-}_{\boldsymbol{z}} = ||\boldsymbol{z}^{(\mathrm{b})} - \boldsymbol{z}^{(\mathrm{n})}||_q$$

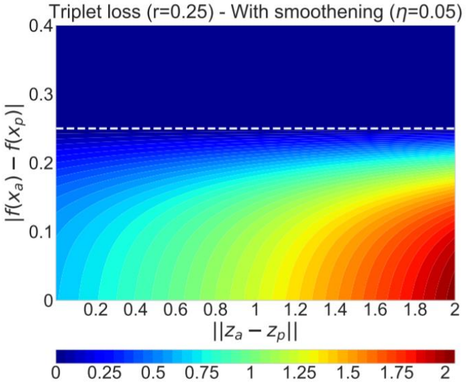
$$\text{such that } \boldsymbol{z}^{(\mathrm{p})} \sim q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(\mathrm{p})}) \text{ and } \boldsymbol{z}^{(\mathrm{n})} \sim q_{\boldsymbol{\phi}}(\cdot|\boldsymbol{x}^{(\mathrm{n})})$$





Smoothening





OK! We can structure latent spaces but how about data efficiency?

Solutions to both properties require label information ...

Tripp et al. 2020

Property I: Label guidance for informative feature spaces reflective of the black-box

$$w(\boldsymbol{x}_i) \propto f(\boldsymbol{x}_i)$$

Property II: Useful features facilitating Gaussian process training in the latent space

$$|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)| \leq r$$

... but this goes against BO that seeks sample efficient solutions ...

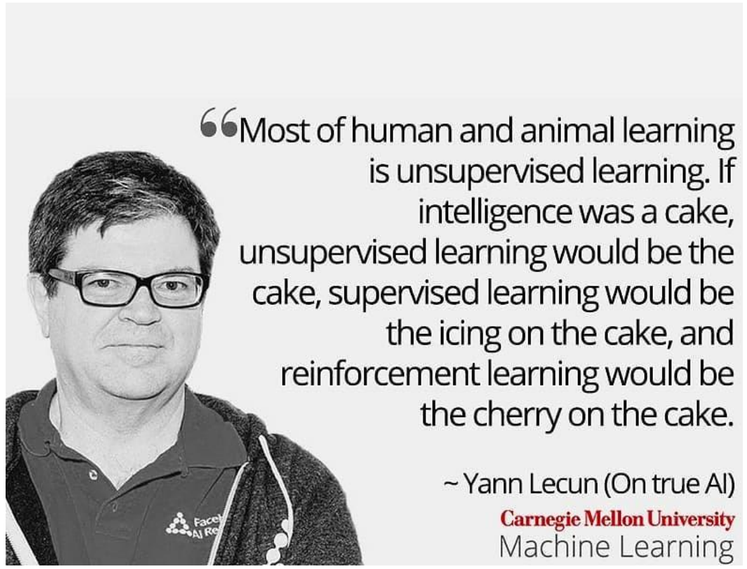
Use Self-Supervised Learning

Exciting direction for future work.

How to augment data for a black-box?

Leverage Unsupervised Learning

Simply, use USL (no labels needed) for bootstrapping features



… we assume unlabelled data and fine-tune with labels…

… unlabelled component …

$$\mathbf{ELBO}^{\text{s-triple}}_{\text{weighted}}(\boldsymbol{\phi}, \boldsymbol{\theta} | \mathcal{D}_{\mathbb{L}}, \mathcal{D}_{\mathbb{U}}, \mathcal{C}) = \sum_{m=1}^{M} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}^{(\mathbb{U})}(\boldsymbol{z}_m^{(\mathrm{u})} | \boldsymbol{x}_m^{(\mathrm{u})})} \left[\log[p_{\boldsymbol{\theta}}(\boldsymbol{x}_m^{(\mathrm{u})} | \boldsymbol{z}_m^{(\mathrm{u})})] \right] - \mathrm{KL}(q_{\boldsymbol{\phi}}^{(\mathbb{U})}(\boldsymbol{z}_m^{(\mathrm{u})} | \boldsymbol{x}_m^{(\mathrm{u})}) || p(\boldsymbol{z}_m^{(\mathrm{u})})) \right]$$

Features that can reconstruct

Fine-Tune as we gather function values (during BO)

$$+ \sum_{n=1}^{N} w(\boldsymbol{x}_n^{(\mathrm{l})}) \left[\mathbb{E}_{q_{\boldsymbol{\phi}}^{(\mathbb{L})}(\boldsymbol{z}_n^{(\mathrm{l})} | \boldsymbol{x}_n^{(\mathrm{l})})} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n^{(\mathrm{l})} | \boldsymbol{z}_n^{(\mathrm{l})}) + \log p_{\boldsymbol{\theta}}(f(\boldsymbol{x}_n^{(\mathrm{l})}) | \boldsymbol{z}_n^{(\mathrm{l})})] - \mathrm{KL}(q_{\boldsymbol{\phi}}^{(\mathbb{L})}(\boldsymbol{z}_n^{(\mathrm{l})} | \boldsymbol{x}_n^{(\mathrm{l})}) || p(\boldsymbol{z}_n^{(\mathrm{l})})) \right]$$

$$- \sum_{i,j,k=1}^{N,N,N} w_{i,j,k} \mathbb{E}_{q_{\boldsymbol{\phi}}^{(\mathbb{L})}(\boldsymbol{z}_i^{(\mathrm{l})}, \boldsymbol{z}_j^{(\mathrm{l})}, \boldsymbol{z}_k^{(\mathrm{l})} | \boldsymbol{x}_{i,j,k}^{(\mathrm{l})})} \left[\mathcal{L}^{(\mathrm{BO})}_{\text{s-triple}}(\boldsymbol{z}_i^{(\mathrm{l})}, \boldsymbol{z}_j^{(\mathrm{l})}, \boldsymbol{z}_k^{(\mathrm{l})}) \right]$$

$N <<< M$

Overall Algorithm

Algorithm 1 High-D BO with VAEs and Deep Metric Learning

Inputs: Budget B, frequency q, datasets $\mathcal{D}_{\mathbb{L}}$ and $\mathcal{D}_{\mathbb{U}}$, stopping criteria τ
for $\ell = 1$ to $L \equiv \lceil B/q \rceil$:
  Solve $\boldsymbol{\theta}_\ell^\star, \boldsymbol{\phi}_\ell^\star = \arg\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbf{ELBO}^{(\text{s-triple})}_{\text{weighted}}(\boldsymbol{\theta}, \boldsymbol{\phi})$
  Compute $\mathcal{D}_{\mathbb{Z}} = \langle \boldsymbol{z}_i, f(\boldsymbol{x}_i^{(\mathrm{l})}) \rangle_{i=1}^{N}$ s.t. $\boldsymbol{z}_i \sim q_{\boldsymbol{\phi}_\ell^\star}(\cdot | \boldsymbol{x}_i^{(\mathrm{l})})$, $\forall \boldsymbol{x}_i^{(\mathrm{l})} \in \mathcal{D}_{\mathbb{L}}$
  for $k = 0$ to $q - 1$ and $\alpha_{\mathrm{EI}}(\hat{\boldsymbol{z}}_{\ell,k+1}) \geq \tau$:
    Fit surrogate GP on $\langle \boldsymbol{z}_i, f(\boldsymbol{x}_i) \rangle_{i=1}^{N}$
    Optimise acquisition $\hat{\boldsymbol{z}}_{\ell,k+1} = \arg\max_{\boldsymbol{z} \in \mathcal{Z}} \alpha_{\mathrm{EI}}(\boldsymbol{z})$ and compute $\hat{\boldsymbol{x}} = g_{\boldsymbol{\theta}_l^\star}(\cdot | \hat{\boldsymbol{z}}_{\ell,k+1})$
    Evaluate $f(\hat{\boldsymbol{x}})$ and augment data $\mathcal{D}_{\mathbb{Z}}$ and $\mathcal{D}_{\mathbb{L}}$
    end for
  end for
Output: $\boldsymbol{x}^\star = \arg\max_{\boldsymbol{x} \in \mathcal{D}_{\mathbb{L}}} f(\boldsymbol{x})$

… we also demonstrate **convergence in the original space** .

… we analyse outer and inner regret …

$$\text{Regret}_{L,q}(\langle \hat{\boldsymbol{z}}_{\ell,k} \rangle_{\ell,k}^{L,q}) = \sum_{\ell=1}^{L} \sum_{k=1}^{q} \left(f(\boldsymbol{x}^\star) - \mathbb{E}_{\boldsymbol{x}^\star_{\ell,k} \sim g_{\boldsymbol{\theta}^\star_\ell}(\cdot|\hat{\boldsymbol{z}}_{\ell,k})}[f(\boldsymbol{x}^\star_{\ell,k})] \right)$$

VAE retraining loop

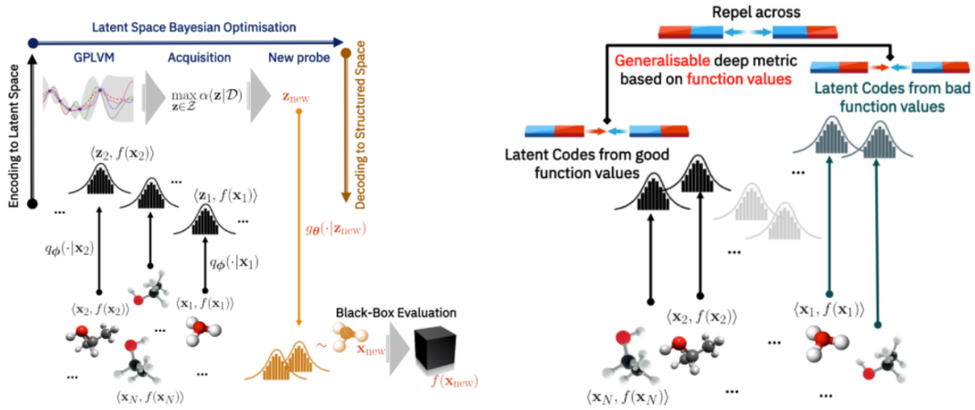
Bayesian optimisation regret

$$\Pr\left[\boldsymbol{x}^\star \sim g_{\boldsymbol{\theta}^\star_\ell}(\cdot|\bar{\boldsymbol{z}}(\ell))\right] \geq 1 - \gamma(\ell)$$

Theorem 1. *Algorithm 1 with* $q = \lceil B^{\frac{2}{3}} \rceil$, $L = \lceil B^{\frac{1}{3}} \rceil$ *and under the assumptions in Appendix C admits sub-linear averaged regrets of the form:* $\lim_{B\to\infty} \frac{1}{B} \text{Regret}_{L,q}(\langle \hat{\boldsymbol{z}}_{\ell,k} \rangle_{\ell,k}^{L,q}) \to 0$, *with a probability of at least* $1 - \delta$ *for* $\delta \in (0, 1)$.

Experimental Results

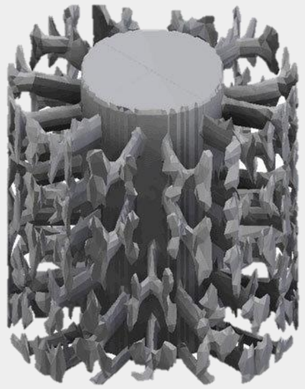
T-LBO



Codebase associated to: High-Dimensional Bayesian Optimisation withVariational Autoencoders and Deep Metric Learning

https://github.com/huawei-noah/HEBO

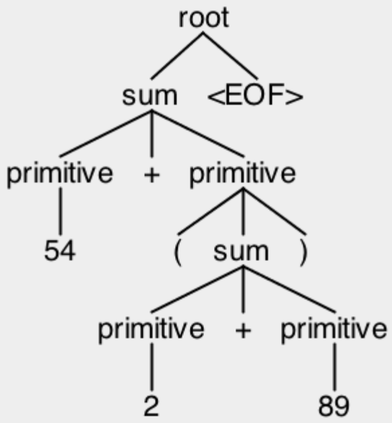
We considered the following three tasks ...



Topology toy domain

Starting from a blank image generate a target structure

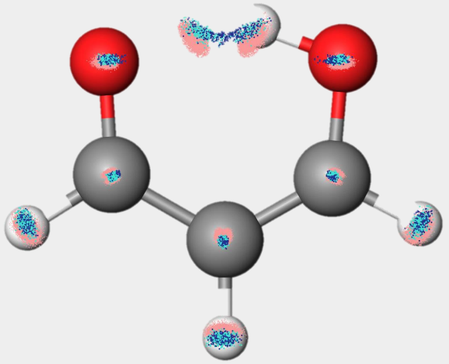
$\boldsymbol{x}$	40x40 image
$f(\boldsymbol{x})$	dist. to target
$\mathcal{Z}$	20-dimensional



Arithmetic Expression

Find a target expression from a formal grammar

$\boldsymbol{x}$	180 binary mat.
$f(\boldsymbol{x})$	dist. to target
$\mathcal{Z}$	25-dimensional



Molecule generation

Find a molecule that maximises penalised logP

$\boldsymbol{x}$	SMILE graphs
$f(\boldsymbol{x})$	plogP values
$\mathcal{Z}$	56-dimensional

… and we are chiefly interested in the following questions …

Question I: Does deep metric learning enable latent spaces that promote generalisation of GPs?

Question II: Does improved generalisation help Bayesian Opt when lots of labels are available?

Question III: Do those improvements in generalisation carry to the case when data is limited?

… we answer question I affirmatively, DML helps to generalise …

Question I: Does deep metric learning enable latent spaces that promote generalisation of GPs?

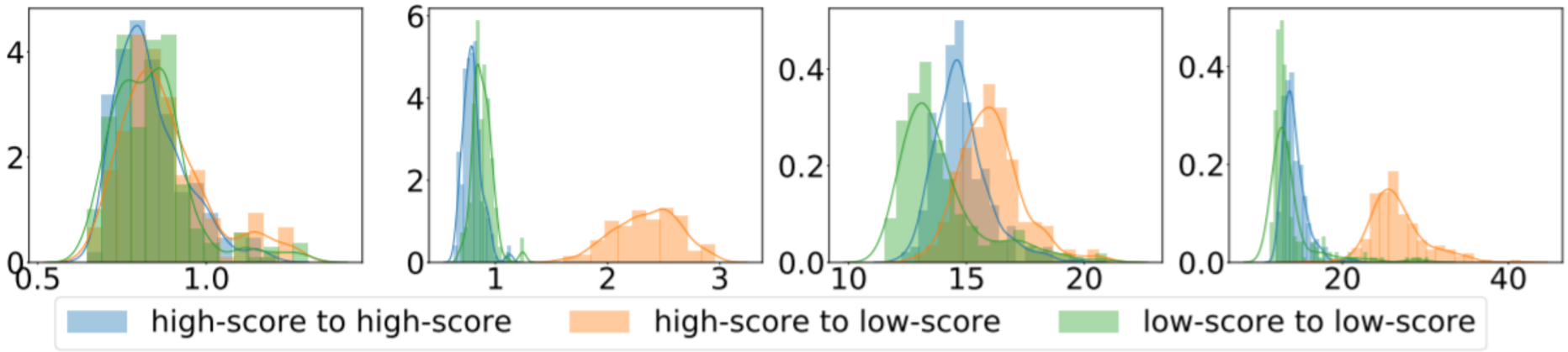


Table 1: GP predictive log-likelihood on validation set.

	Vanilla	Target Prediction	Contrastive	Triplet
Topology	-1.87 (0.06)	**-1.35 (0.01)**	-1.75 (0.02)	-2.03 (0.02)
Expression	-2.99 (0.06)	-2.02 (0.06)	**-1.39 (0.04)**	**-1.91 (0.08)**
Molecule	-1.79 (0.21)	-2.68 (0.82)	**-1.75 (0.18)**	**-1.55 (0.35)**

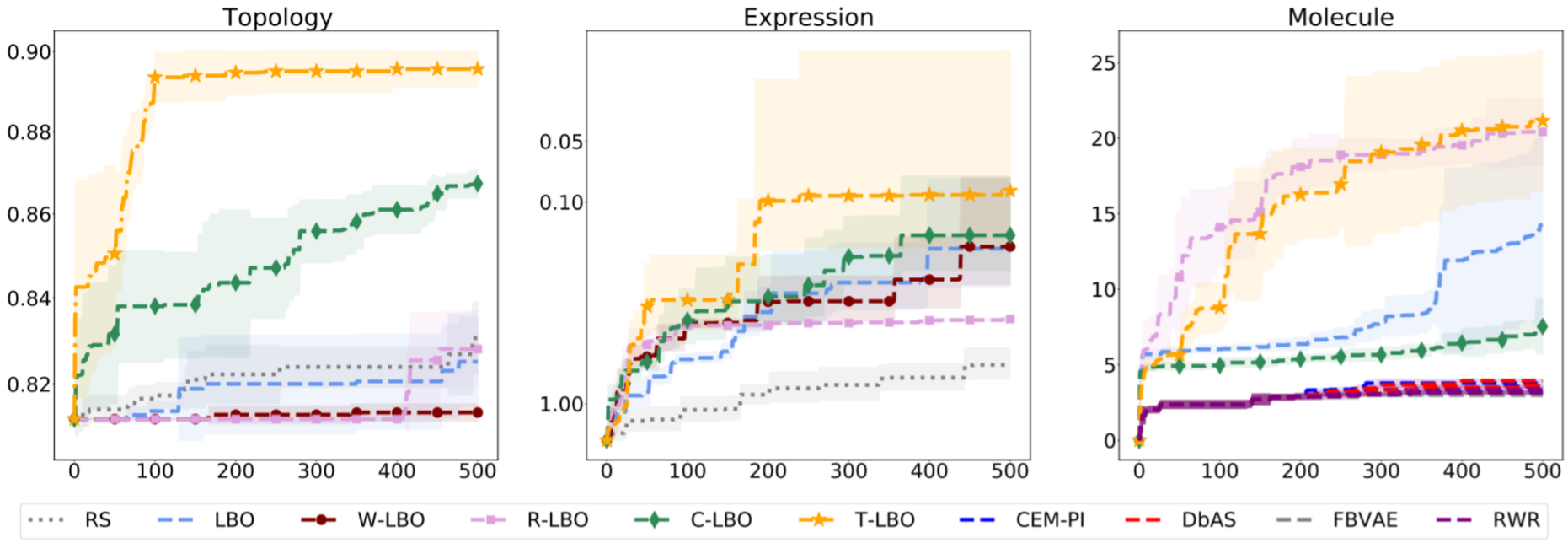
Question II: Does improved generalisation help Bayesian Opt when lots of labels are available?

Question III: Do those improvements in generalisation carry to the case when data is limited?

… we answer question II affirmatively, DML helps to BO …

Question I: Does deep metric learning enable latent spaces that promote generalisation of GPs?

Question II: Does improved generalisation help Bayesian Opt when lots of labels are available?



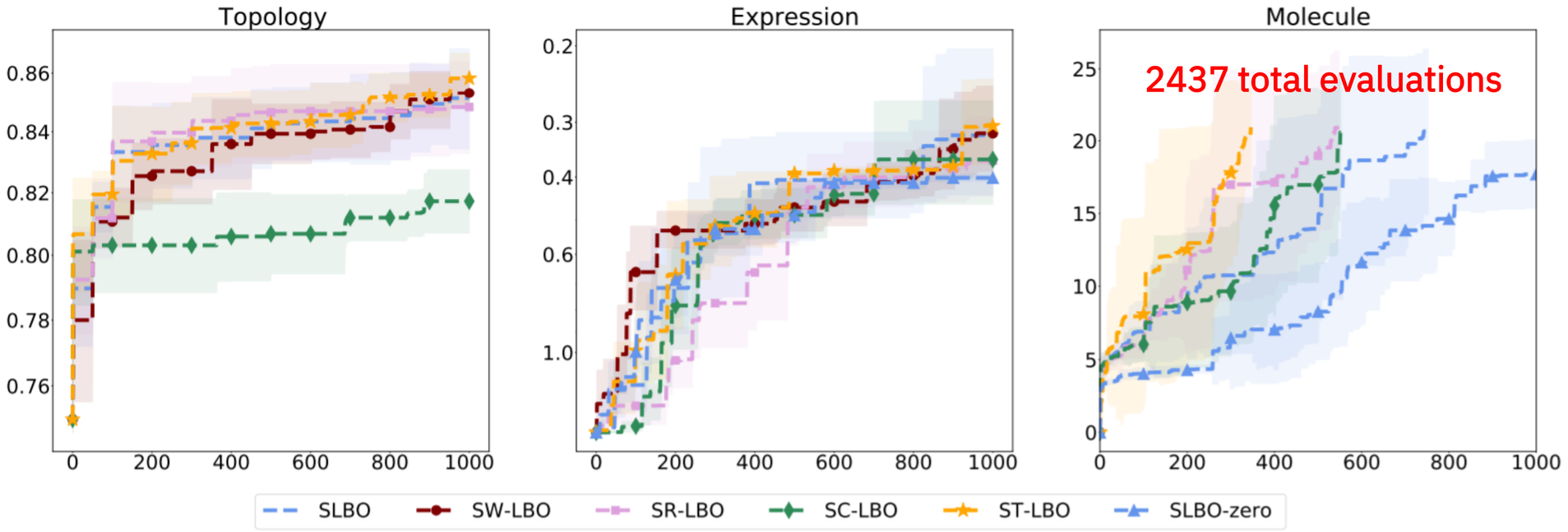
Question III: Do those improvements in generalisation carry to the case when data is limited?

… and we answer question III affirmatively as well.

Question I: Does deep metric learning enable latent spaces that promote generalisation of GPs?

Question II: Does improved generalisation help Bayesian Opt when lots of labels are available?

Question III: Do those improvements in generalisation carry to the case when data is limited?



99% less labels

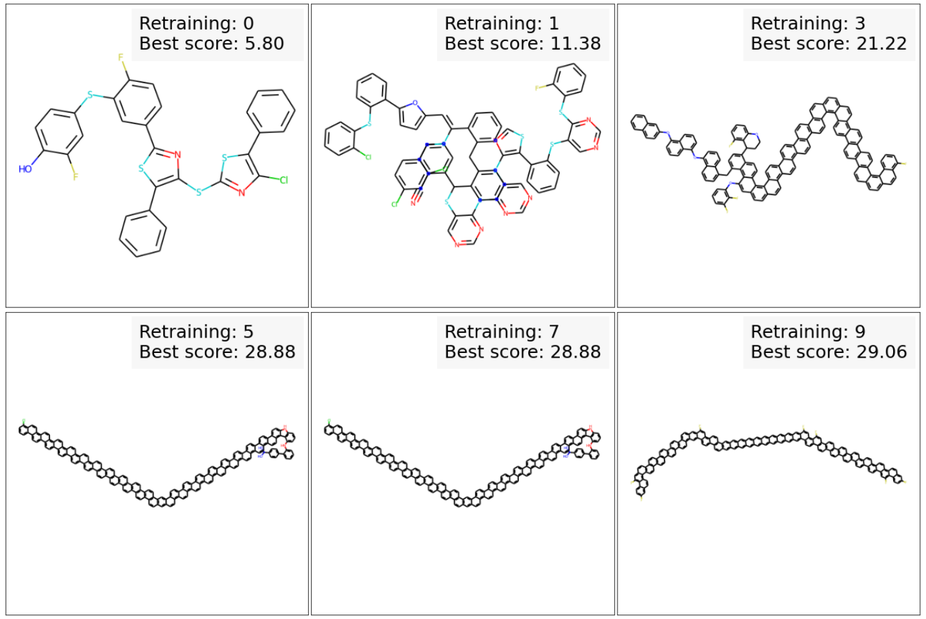
… SOTA molecule results using only 1% of labels from ZINC250K data-set …

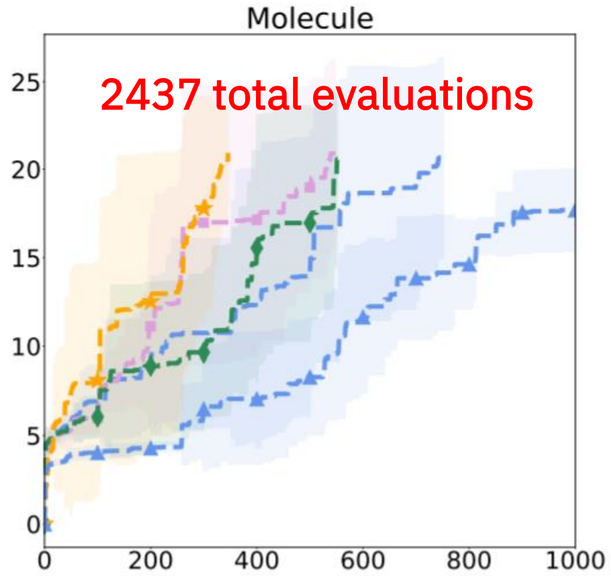
… and we answer question III affirmatively as well.

Question I: Does deep metric learning enable latent spaces that promote generalisation of GPs?

Question II: Does improved generalisation help Bayesian Opt when lots of labels are available?

Question III: Do those improvements in generalisation carry to the case when data is limited?





99% less labels

… SOTA molecule results using only 1% of labels from ZINC250K data-set …

All those, are again in our library, e.g., linear embeddings …

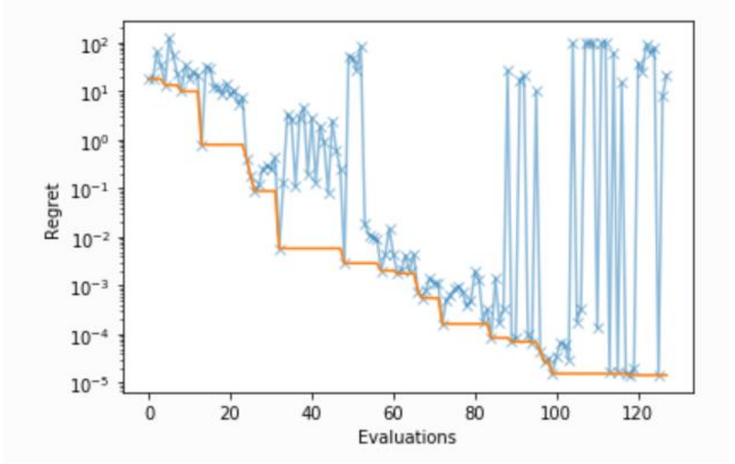
High-dimensional optimisation via random linear embedding

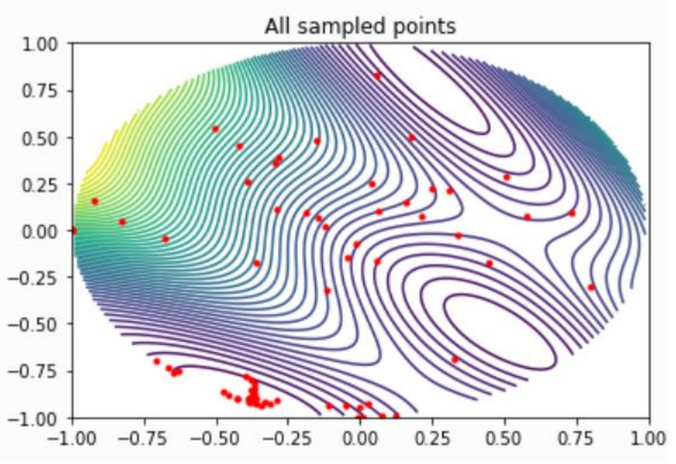
[3]:
```
prob  = BraninDummy(1000) # 1000-D Branin function where the frist two dimensions are active
opt   = HEBO_Embedding(prob.space, rand_sample = 10, eff_dim = 2, scale = 1)
```

[4]:
```
def f(x : np.ndarray, y : np.ndarray):
    data = pd.DataFrame(np.hstack([x.reshape(-1,1), y.reshape(-1,1)]),columns = opt.eff_space.numer
    x_hd = opt.project(data)
    valid = (x_hd.abs() < 1).all(axis = 1).values.reshape(-1)
    f    = prob(x_hd).reshape(-1)
    f[~valid] = np.nan
    return f

x = np.linspace(-1*opt.scale,1*opt.scale,200)
y = np.linspace(-1*opt.scale,1*opt.scale,200)

if opt.eff_dim == 2:
    X,Y = np.meshgrid(x,y)
    Z   = f(X,Y).reshape(200,200)
    plt.contour(X, Y, Z, 50);
    plt.title('2D function seen by optimizer')
    print('Best value is %.3f' % Z[np.isfinite(Z)].min())
```

[5]:
```
for i in range(16):
    rec   = opt.suggest(8)
    rec_h = opt.project(rec)
    y     = prob(rec_h) # optimizer return points in embedding space
    opt.observe(rec,y)
    print('Iter %d, y = %.2f, best_y = %.2f' % (i, y.min(), opt.mace.y.min()))
```

```
Iter 0, y = 13.40, best_y = 13.40
Iter 1, y = 0.80, best_y = 0.80
Iter 2, y = 5.58, best_y = 0.80
Iter 3, y = 0.09, best_y = 0.09
Iter 4, y = 0.01, best_y = 0.01
Iter 5, y = 0.08, best_y = 0.01
Iter 6, y = 0.00, best_y = 0.00
Iter 7, y = 0.00, best_y = 0.00
Iter 8, y = 0.00, best_y = 0.00
Iter 9, y = 0.00, best_y = 0.00
Iter 10, y = 0.00, best_y = 0.00
Iter 11, y = 0.00, best_y = 0.00
Iter 12, y = 0.00, best_y = 0.00
Iter 13, y = 0.00, best_y = 0.00
Iter 14, y = 0.00, best_y = 0.00
Iter 15, y = 0.00, best_y = 0.00
```



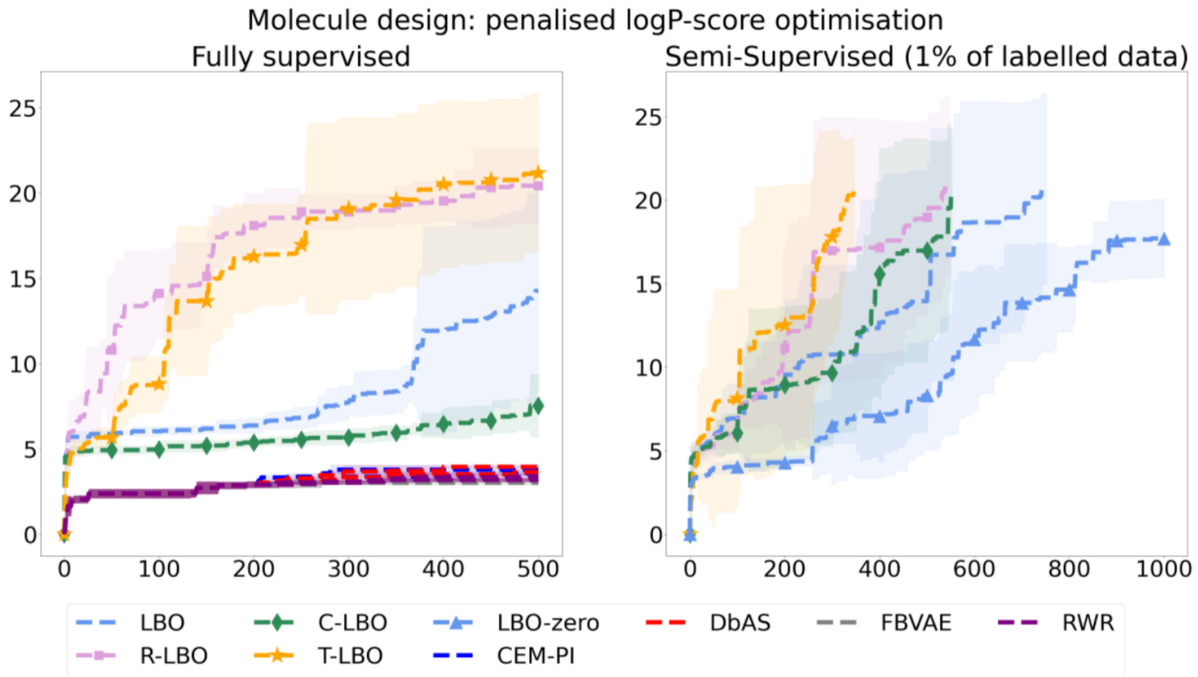


… or using VAEs like described above.

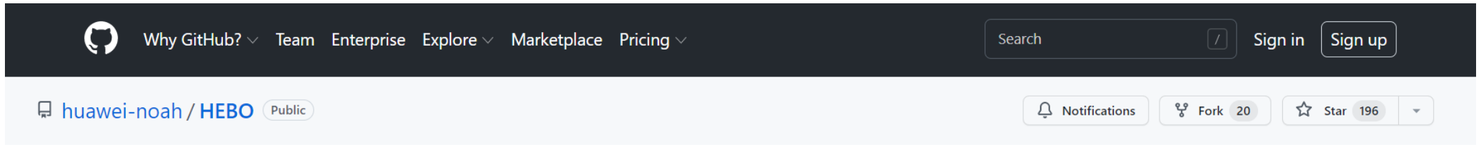
```python
import weighted_retraining.weighted_retraining.chem.jtnn.datautils as datautils
from weighted_retraining.weighted_retraining.chem.chem_model import JTVAE
from weighted_retraining.weighted_retraining.chem.chem_utils import standardize_smiles, penalized_logP, QED_score
from weighted_retraining.weighted_retraining.chem.jtnn import MolTreeFolder, MolTreeDataset, Vocab, MolTree
from weighted_retraining.weighted_retraining.chem.jtnn.datautils import TargetMolTreeDataset
from weighted_retraining.weighted_retraining.utils import print_flush

NUM_WORKERS = 4


################################################
# Data preprocessing code
################################################
def get_vocab_from_tree(tree: MolTree):
    cset = set()
    for c in tree.nodes:
        cset.add(c.smiles)
    return cset


def get_vocab_from_smiles(smiles):
    """ Get the set of all vocab items for a given smiles """
    mol = MolTree(smiles)
    return get_vocab_from_tree(mol)
```

```python
class WeightedJTNNDataset(pl.LightningDataModule):
    """ dataset with property weights. Needs to load all data into memory """

class WeightedMolTreeFolder(MolTreeFolder):
    """ special weighted mol tree folder """
```

Optimisation of penalised logP



https://github.com/huawei-noah/HEBO



In the future, we are planning on …

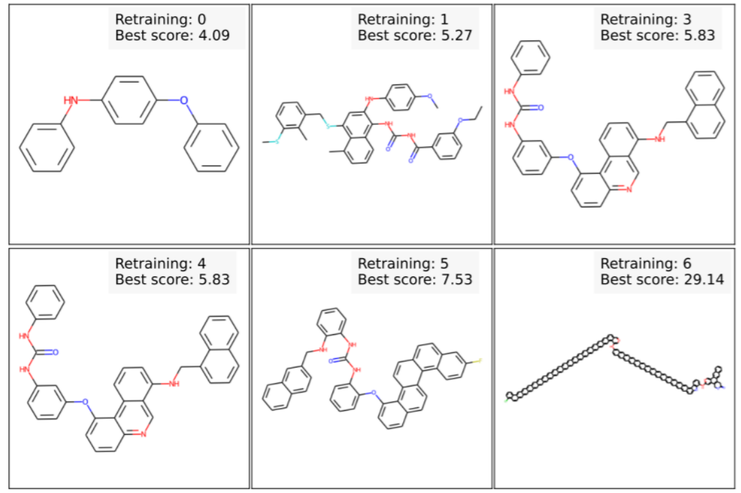
… can we avoid this assumption …

$$\mathbb{P}\mathrm{r}\left[\boldsymbol{x}^{\star} \sim g_{\boldsymbol{\theta}^{\star}_{\ell}}(\cdot|\bar{\boldsymbol{z}}(\ell))\right] \geq 1-\gamma(\ell)$$

Theorem 1. *Algorithm 1 with* $q = \lceil B^{\frac{2}{3}} \rceil$, $L = \lceil B^{\frac{1}{3}} \rceil$ *and under the assumptions in Appendix C admits sub-linear averaged regrets of the form:* $\lim_{B\to\infty} \frac{1}{B} Regret_{L,q}(\langle \hat{\boldsymbol{z}}_{\ell,k} \rangle_{\ell,k}^{L,q}) \to 0$, *with a probability of at least* $1-\delta$ *for* $\delta \in (0,1)$.

… plogP's aren't the best measure of molecules …

… currently, we are looking at more realistic settings, e.g., IC50s of molecules …



Noah's Ark

Huawei R&D, London

Thank you!

@hbouammar Haitham Bou Ammar

https://github.com/huawei-noah/HEBO

Easy to use

Flexible to use

Supports MOO

Supports parallel BO

Supports High-D BO

Supports Evol. Algos

Supports Mixed-Variable Optimization

Includes a variety of solvers and benchmarks

Interfaces with GPyTorch, GPy, GPFLow, Pymoo

Bayesian Optimisation Research

This directory contains official implementations for Bayesian optimisation works developped by Huawei R&D, Noah's Ark Lab.

- HEBO: Heteroscedastic Evolutionary Bayesian Optimisation
- T-LBO
- Bayesian Optimisation with Compositional Optimisers

Further instructions are provided in the README files associated to each project.

HEBO

HEBO 河伯

Heteroscedastic Evolutionary Bayesian Optimisation

Bayesian optimsation library developped by Huawei Noahs Ark Decision Making and Reasoning (DMnR) lab. The **winning submission** to the NeurIPS 2020 Black-Box Optimisation Challenge.

Jonas Mockus. On Bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

A. Zhilinskas. Single-step bayesian search method for an extremum of functions of a single variable. *Cybernetics, 11:160–166*, 1975.

Bruno Betrò. Bayesian methods in global optimization. *Journal of Global Optimization*, 1(1):1–14, 1991.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions*. Journal of Global Optimization, 13(4):455–492*, 1998.

Eric P Xing, Andrew Y Ng, Michael I Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In Proceedings of the 15th International Conference on Neural Information Processing Systems, pages 521–528, 2002.

H. Lodhi, C. Saunders, J. Shawe-Taylor et al., ” Text classification using string kernels ,” *Journal of Machine Learning Research*, 2002.

Carl Edward Rasmussen and Christopher KI Williams. Gaussian Processes for Machine Learning, volume 2. *MIT press Cambridge*, *MA*, 2006

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of Proceedings of Machine Learning Research, pages 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

Michael A. Osborne, Roman Garnett, and Stephen J. Roberts. Gaussian processes for global optimization. In in *LION*, 2009.

Miguel Lázaro-Gredilla and Michalis K. Titsias. Variational heteroscedastic gaussian process regression. In *Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML’11*, page 841–848, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195. 2011.

Chunyi Wang and Radford M. Neal. Gaussian process regression with heteroscedastic or nongaussian residuals. *CoRR, abs/1212.6246*, 2012.

Jasper Snoek, Ryan P Adams, and Hugo Larochelle. Nonparametric guidance of autoencoder representations using label information. Journal of Machine Learning Research, 13:2567–2588, 2012.

Carl Ivar Branden and John Tooze. Introduction to protein structure. *Garland Science*, 2012.

Andreas Damianou and Neil D. Lawrence. Deep Gaussian processes. In Carlos M. Carvalho and Pradeep Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of Proceedings of Machine Learning Research, pages 207–215, Scottsdale, Arizona, USA, 29 Apr–01 May 2013.

Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, pages 1674–1682. PMLR, 2014.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In 2nd *International Conference on Learning Representations*, ICLR, 2014.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabas Poczos. High dimensional bayesian optimisation and bandits via additive models. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research*, pages 295–304, Lille, France, 07–09 Jul 2015. PMLR.

Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer, 2015.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE, 104(1):148–175*, 2016.

Javier I. González, Michael A. Osborne, and Neil D. Lawrence. Glasses: Relieving the myopia of bayesian optimisation. In *AISTATS*, 2016.

Santu Rana, Cheng Li, Sunil Gupta, Vu Nguyen, and Svetha Venkatesh. High dimensional Bayesian optimization with elastic Gaussian process. In *International conference on machine learning, pages 2883–2891. PMLR*, 2017.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–1954. PMLR, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In Jennifer Dy and Andreas Krause, editors*, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research*, pages 2323–2332. PMLR, 10–15 Jul 2018.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

C. Yu et al., Developing Synthesis Flows Without Human Knowledge, *Proceedings of the 55th Annual Design Automation Conference*, 2018.

Stephan Eissman, Daniel Levy, Rui Shu, Stefan Bartzsch, and Stefano Ermon. Bayesian optimization and attribute adjustment. In *Proc. 34th Conference on Uncertainty in Artificial Intelligence*, 2018.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

ChangYong Oh, Efstratios Gavves, and Max Welling. BOCK : Bayesian optimization with cylindrical kernels. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research*, pages 3868–3877. PMLR, 10–15 Jul 2018.

Mahmut Kaya and Hasan ¸Sakir Bilge. Deep metric learning: A survey. *Symmetry*, 11(9):1066, 2019.

Ryan-Rhys Griffiths, Alexander A Aldrick, Miguel Garcia-Ortegon, Vidhi R Lalchand, and Alpha A Lee. Achieving robustness to aleatoric uncertainty with heteroscedastic Bayesian optimisation. *arXiv preprint arXiv:1910.07779*, 2019.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Peter Wu, SaiKrishna Rallabandi, Alan W. Black, and Eric Nyberg. Ordinal Triplet Loss: Investigating Sleepiness Detection from Speech. In *Proc. Interspeech 2019*, pages 2403–2407, 2019.

Riccardo Moriconi, Marc Peter Deisenroth, and KS Sesh Kumar. *High-dimensional Bayesian optimization using low-dimensional feature spaces. Machine Learning, 109(9):1925–1943*, 2020.

Haque Ishfaq, Assaf Hoogi, and Daniel Rubin. Tvae: Triplet-based variational autoencoder using metric learning. *arXiv preprint arXiv:1802.04403*, 2018.

Peter Wu, SaiKrishna Rallabandi, Alan W. Black, and Eric Nyberg. Ordinal Triplet Loss: Investigating Sleepiness Detection from Speech. In Proc. Interspeech 2019, pages 2403–2407, 2019.

Hao Liu, Jiwen Lu, Jianjiang Feng, and Jie Zhou. Ordinal deep learning for facial age estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(2):486–501, 2019.

Aryan Deshwal, Syrine Belakaria, and Janardhan Rao Doppa. Mercer features for efficient combinatorial bayesian optimization. *arXiv preprint arXiv:2012.07762*, 2020.

Aryan Deshwal, Syrine Belakaria, and Janardhan Rao Doppa. Mercer features for efficient combinatorial bayesian optimization. *arXiv preprint arXiv:2012.07762*, 2020.

Austin Tripp, Erik Daxberger, and José Miguel Hernández-Lobato. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11259–11272. Curran Associates, Inc., 2020

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020*. CoRR, abs/2104.10201*, 2021.

Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chemical Science., 11:577–586*, 2020.

Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In *International Conference on Artificial Intelligence and Statistics*, pages 3393–3403. PMLR, 2020.

Henry Moss, David Leslie, Daniel Beck, Javier Gonzalez, and Paul Rayson. Boss: Bayesian optimization over string spaces. *Advances in Neural Information Processing Systems*, 33, 2020.

Aryan Deshwal, Syrine Belakaria, and Janardhan Rao Doppa. Mercer features for efficient combinatorial bayesian optimization. *arXiv preprint arXiv:2012.07762*, 2020.

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 33, pages 21524–21538, 2020.

Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Hao Jianye, Jun Wang, and Haitham Bou Ammar. An empirical study of assumptions in Bayesian optimisation. *arXiv preprint arXiv:2012.03826*, 2020.

Ankush Chakrabarty, Gordon Wichern, and Christopher R. Laughman. Attentive neural processes and batch bayesian optimization for scalable calibration of physics-informed digital twins. *CoRR, abs/2106.15502*, 2021

Xingchen. Wan et al., Think Global and Act Local: Bayesian Optimisation over High-Dimensional Categorical and Mixed Search Spaces, *ICML,* 2021.

Antoine Grosnit, Alexander I Cowen-Rivers, Rasul Tutunov, Ryan-Rhys Griffiths, Jun Wang, and Haitham Bou-Ammar. Are we forgetting about compositional optimisers in bayesian optimisation? *Journal of Machine Learning Research, 22(160):1–78*, 2021.

Antoine Grosnit, Rasul Tutunov, Alexandre Max Maraval, Ryan-Rhys Griffiths, Alexander Imani Cowen-Rivers, Lin Yang, Lin Zhu, Wenlong Lyu, Zhitang Chen, Jun Wang,

David Eriksson and Martin Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In Cassio de Campos and Marloes H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of Proceedings of Machine Learning Research, pages 493–503. PMLR, 27–30 Jul 2021.

Mohammed AlQuraishi and Peter K Sorger. Differentiable biology: Using deep learning for biophysics-based and data-driven modeling of molecular mechanisms. *Nature Methods*, 18(10):1169–1180, 2021.

Juan Maroñas, Oliver Hamelijnck, Jeremias Knoblauch, and Theodoros Damoulas. Transforming gaussian processes with normalizing flows. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of Proceedings of Machine Learning Research, pages 1081–1089. PMLR, 13–15 Apr 2021.

Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. Proteinbert: A universal deep-learning model of protein sequence and function. *bioRxiv*, 2021.

Asif Khan, Alexander I. Cowen-Rivers, Derrick-Goh-Xin Deik, Antoine Grosnit, Kamil Dreczkowski, Philippe A. Robert, Victor Greiff, Rasul Tutunov, Dany Bou-Ammar, Jun Wang, and Haitham Bou-Ammar. Antbo: Towards real-world automated antibody design with combinatorial bayesian optimisation, *arXiv preprint arXiv:2201.12570*, 2022.

Alexandre Maraval, Matthieu Zimmer, Antoine Grosnit, Rasul Tutunov, Jun Wang, and Haitham Bou Ammar. Sample-Efficient Optimisation with Probabilistic Transformer Surrogates. *arxiv.2205.13902*, 2022.