Advances in Bayesian Optimization

Techniques for High Dimensional Bayesian Optimization

Jacob R. Gardner

Assistant Professor, University of Pennsylvania

Outline of the Tutorial

- Quick Overview of the BO Framework and GPs
- Summary of advances in GPs and Acquisition Functions
- Bayesian Optimization over Discrete/Hybrid Spaces

- High-Dimensional Bayesian Optimization
- BoTorch Hands-on Demonstration

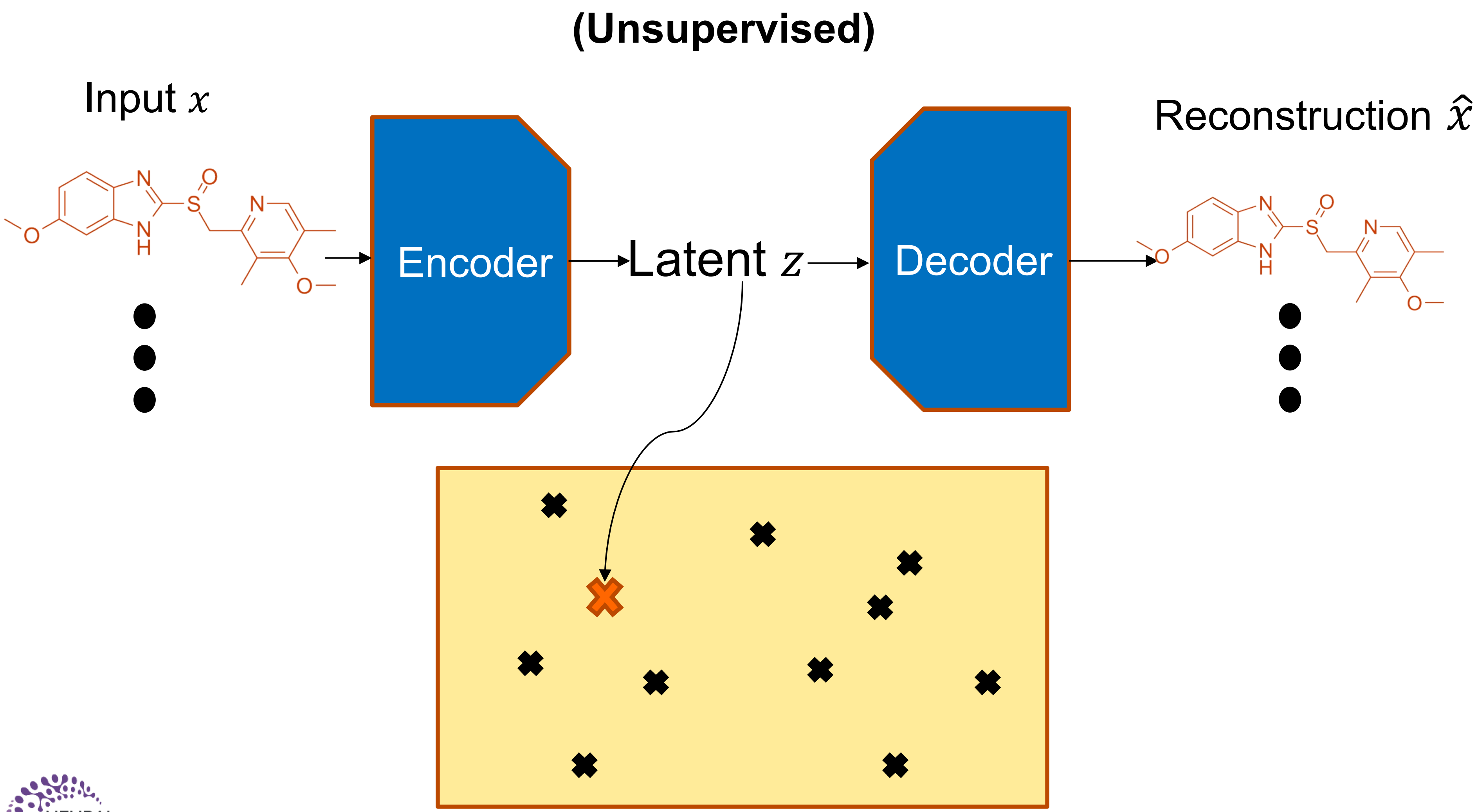
- Causal Bayesian Optimization
- Summary and Outstanding Challenges in BO

Dimensionality is a Key Challenge

Problem: covering a high dimensional search space takes **exponentially many** evaluations!

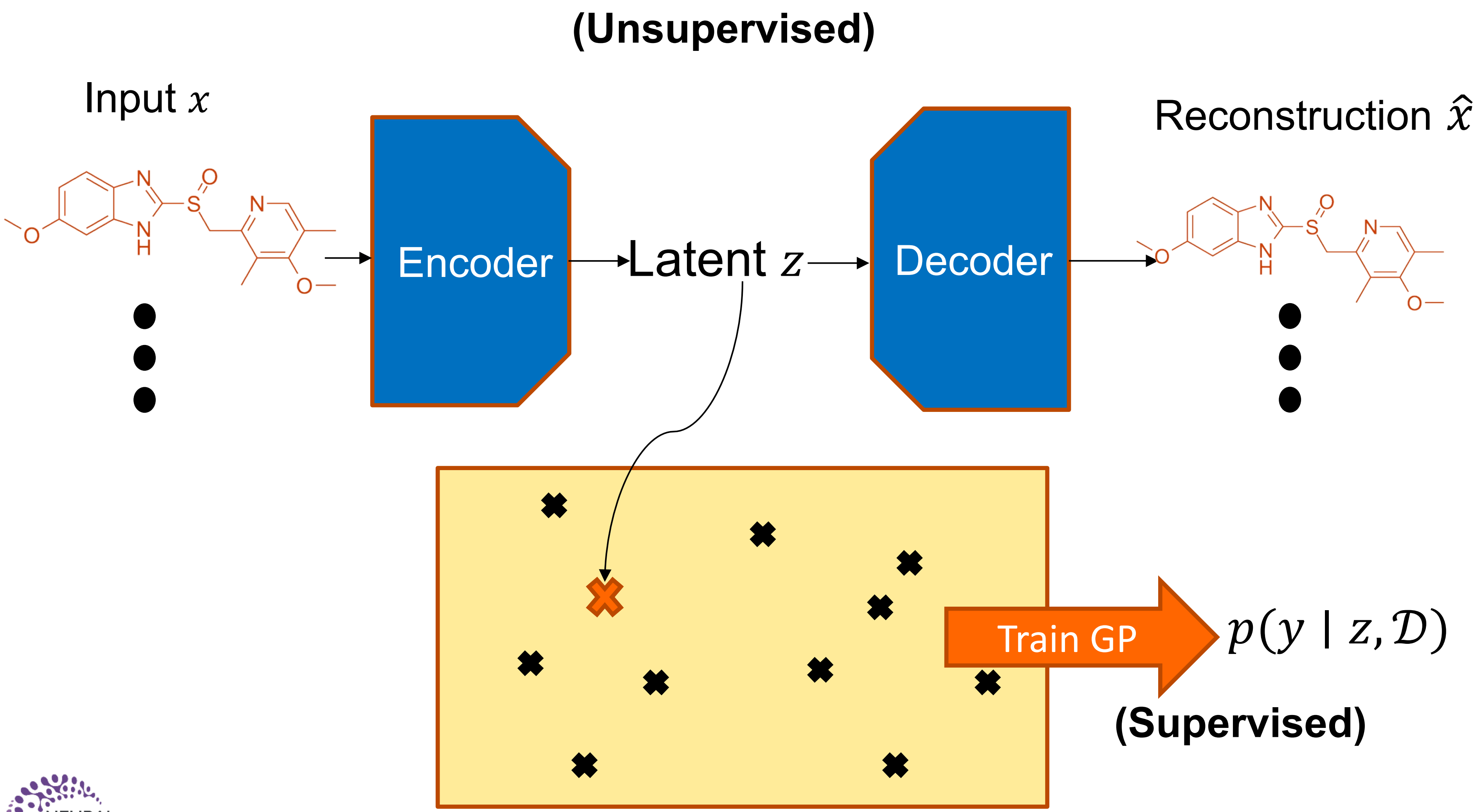
Dimensionality is a Key Challenge

Example



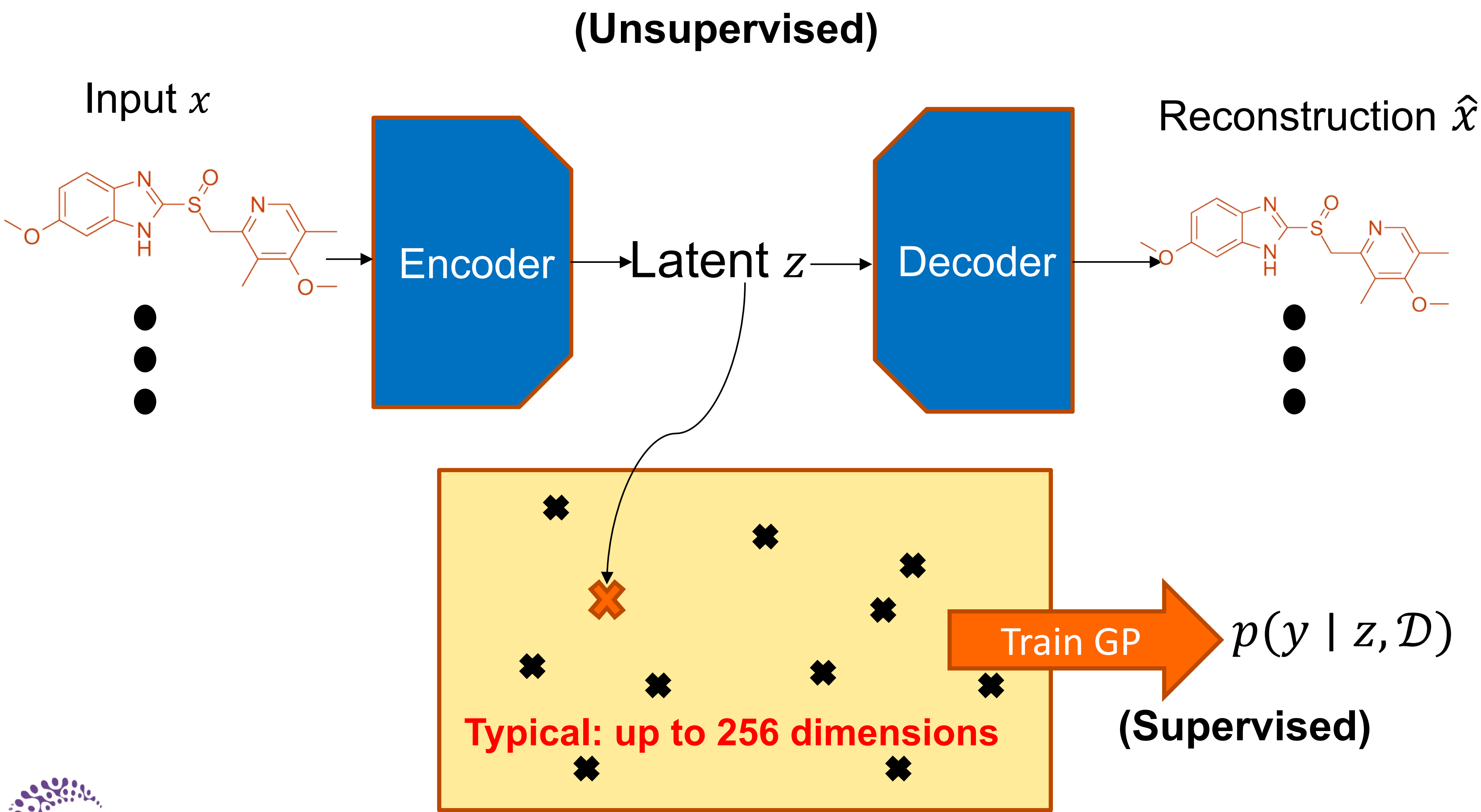
Dimensionality is a Key Challenge

Example



Dimensionality is a Key Challenge

Example

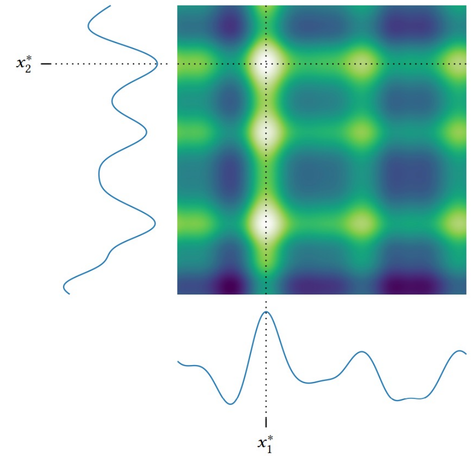


Approaches to High Dimensional BO

Approaches to High Dimensional BO

1. Additive Structure

(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



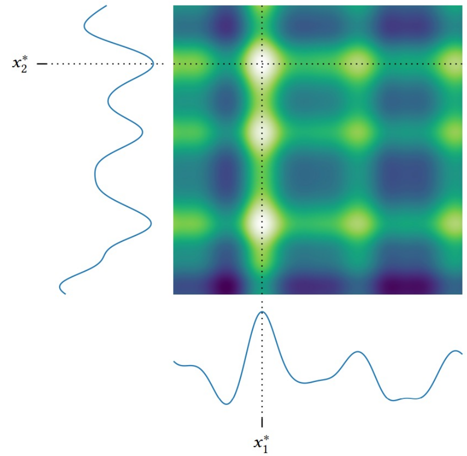
(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

Approaches to High Dimensional BO

1. Additive Structure

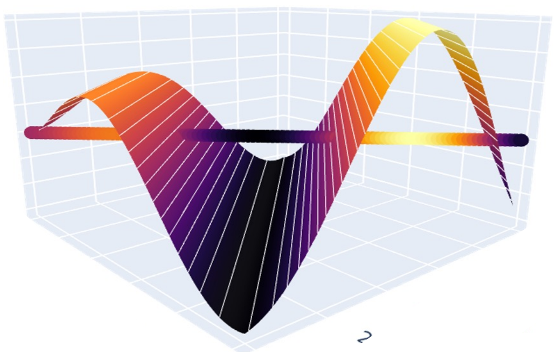
(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

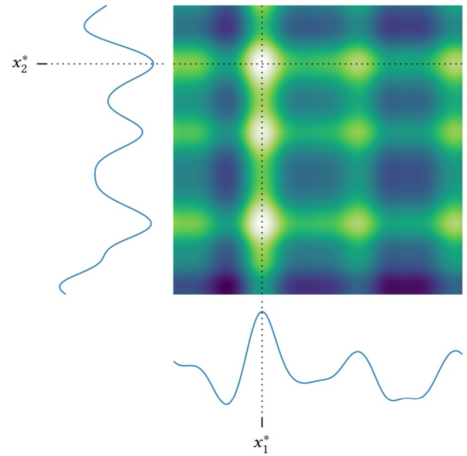
2. Linear Embeddings



$$f(x) = g(Ax)$$
$$g: \mathbb{R}^d \to \mathbb{R}$$

Approaches to High Dimensional BO

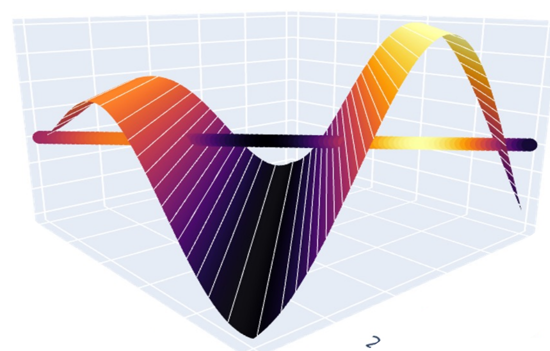
1. Additive Structure
(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

2. Linear Embeddings



$$f(x) = g(Ax)$$
$$g: \mathbb{R}^d \to \mathbb{R}$$

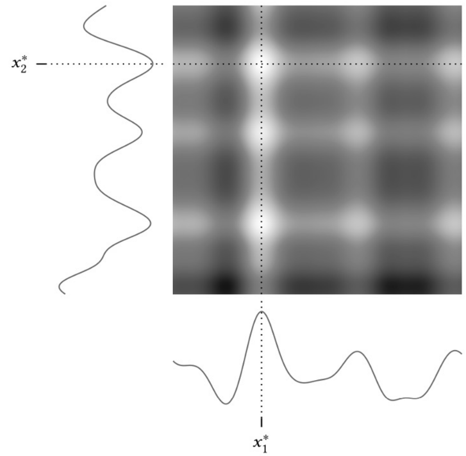
3. Local BayesOpt

"Any optimum will do"

Approaches to High Dimensional BO

1. Additive Structure

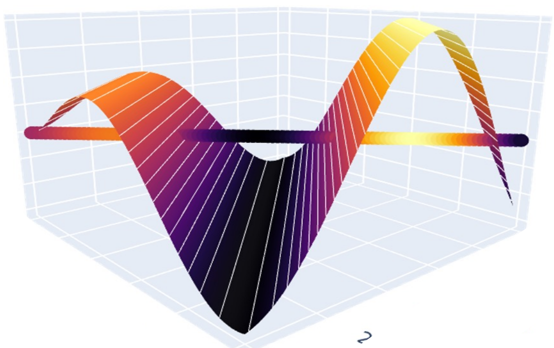
(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

2. Linear Embeddings



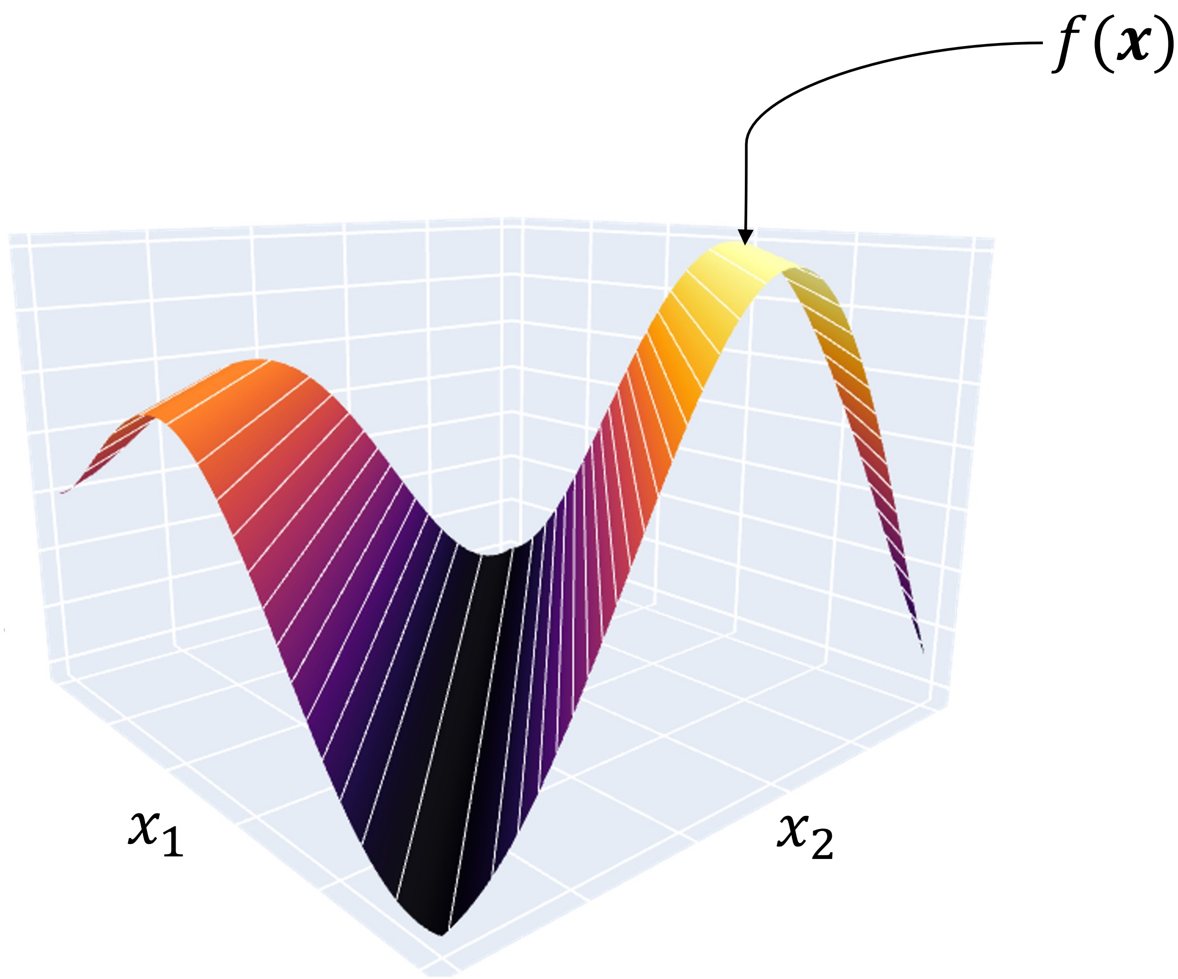
$$f(x) = g(Ax)$$
$$g: \mathbb{R}^d \to \mathbb{R}$$

3. Local BayesOpt

"Any optimum will do"

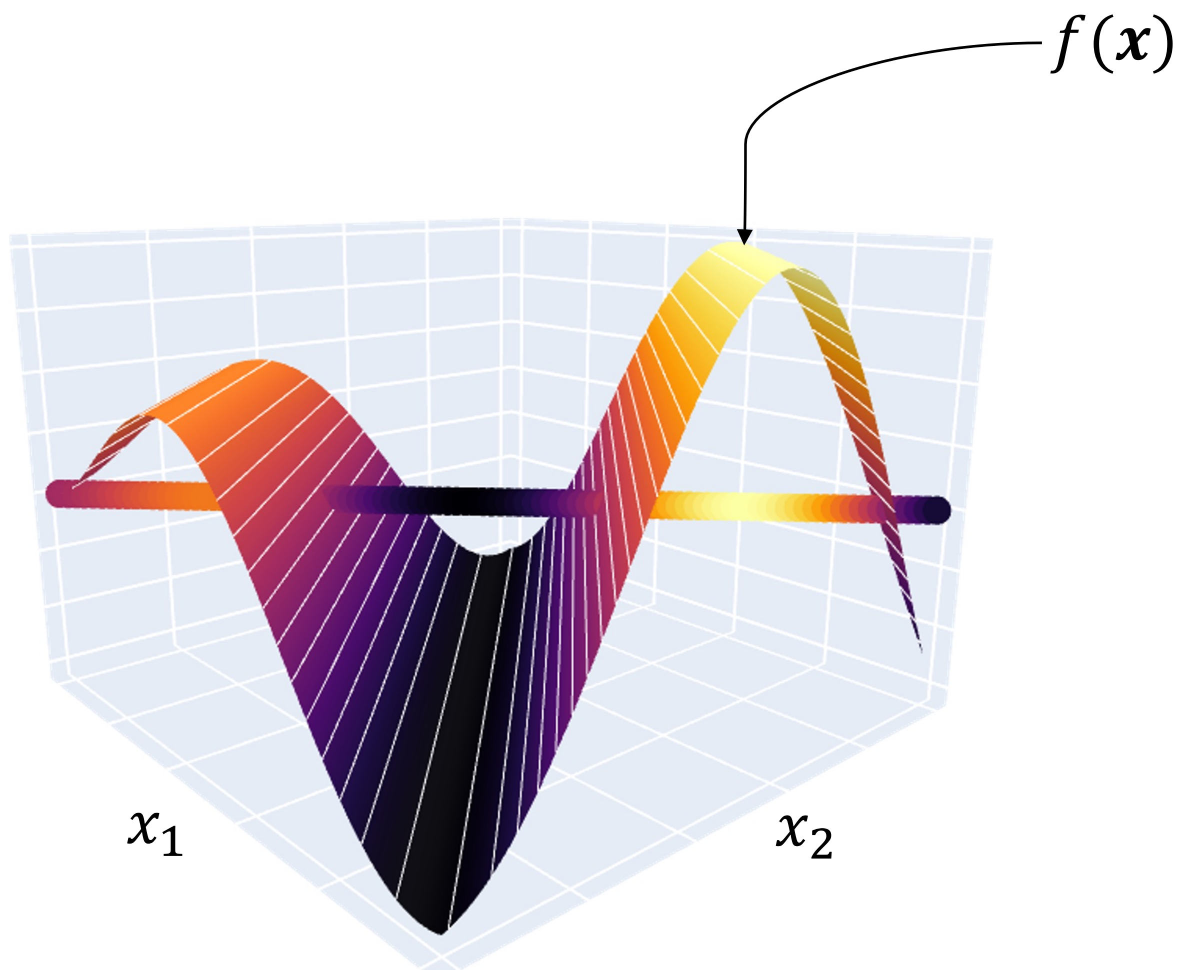
High Dimensional BO via Linear Embeddings

High Dimensional BO: Linear Embeddings



NEURAL INFORMATION PROCESSING SYSTEMS

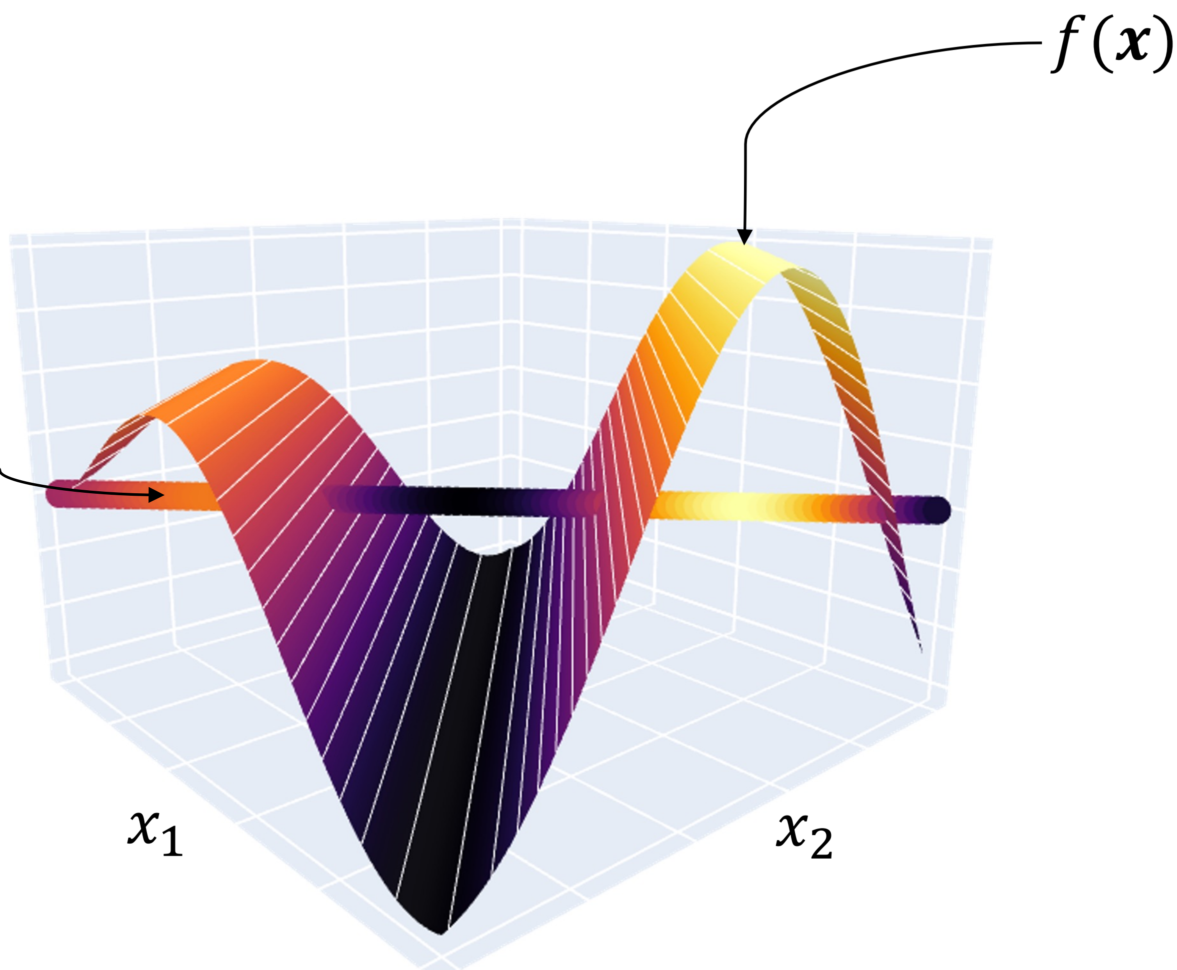
High Dimensional BO: Linear Embeddings



High Dimensional BO: Linear Embeddings

Assumes:
$f(x) = g(Ax)$

$A \in \mathbb{R}^{d \times D}$ x



High Dimensional BO: Linear Embeddings

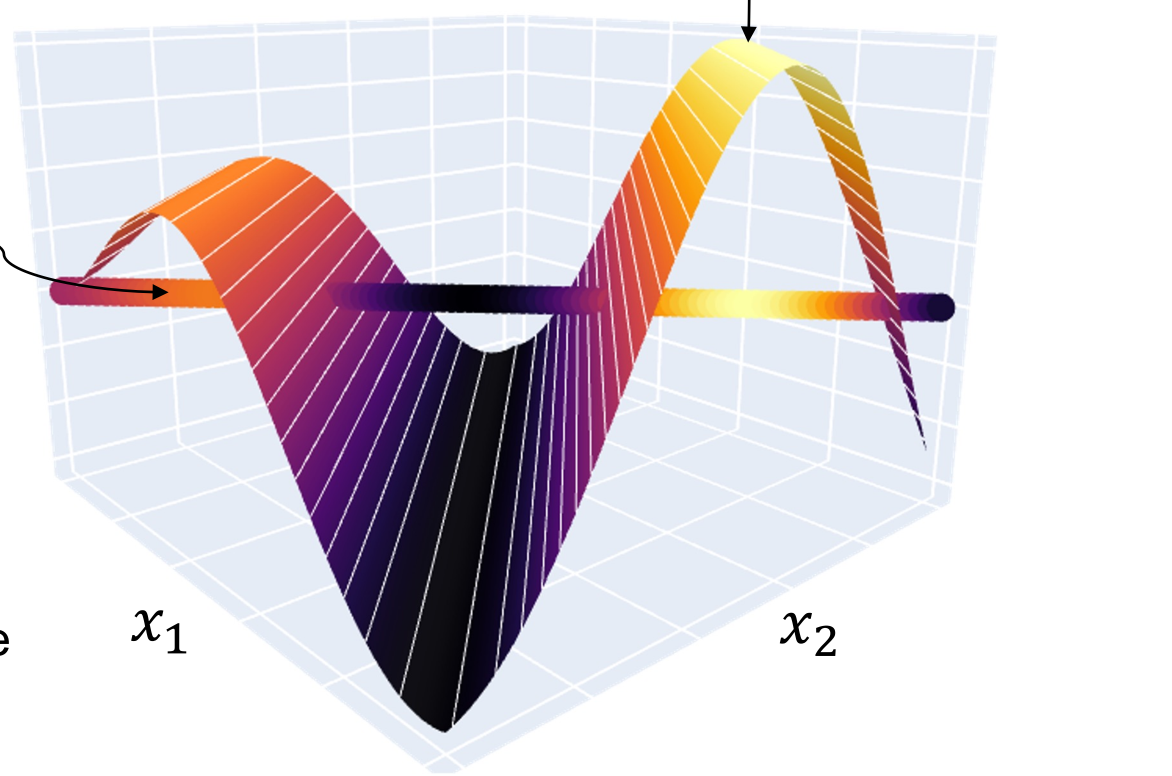
Wang et al., 2014 (REMBO): Random A

$A \in \mathbb{R}^{d \times D}$ x

When D=2, d=1,

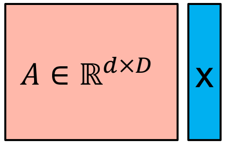
“how far along

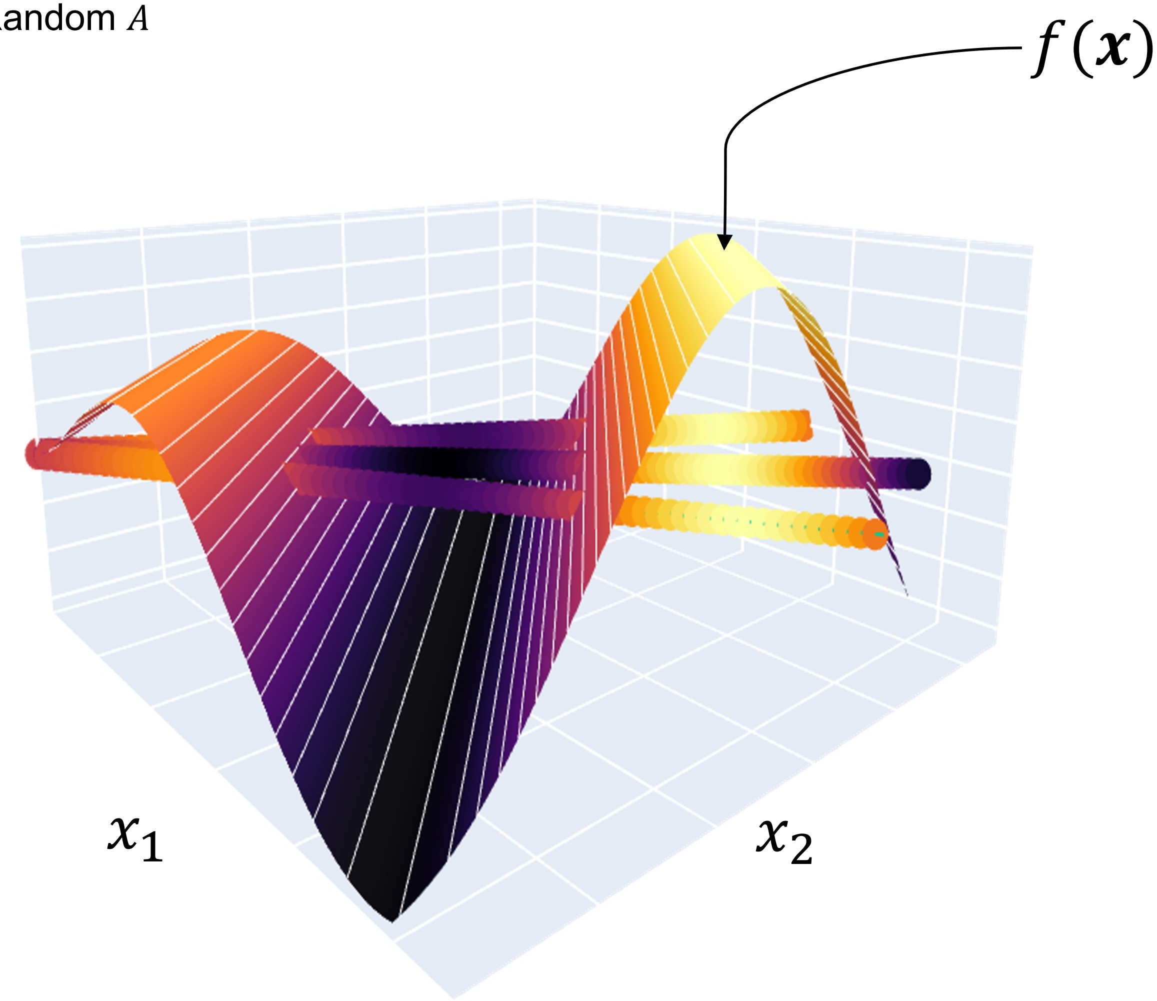
the vector” to move



High Dimensional BO: Linear Embeddings

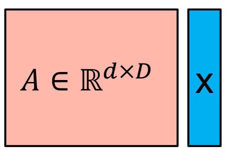
Wang et al., 2014 (REMBO): Random A

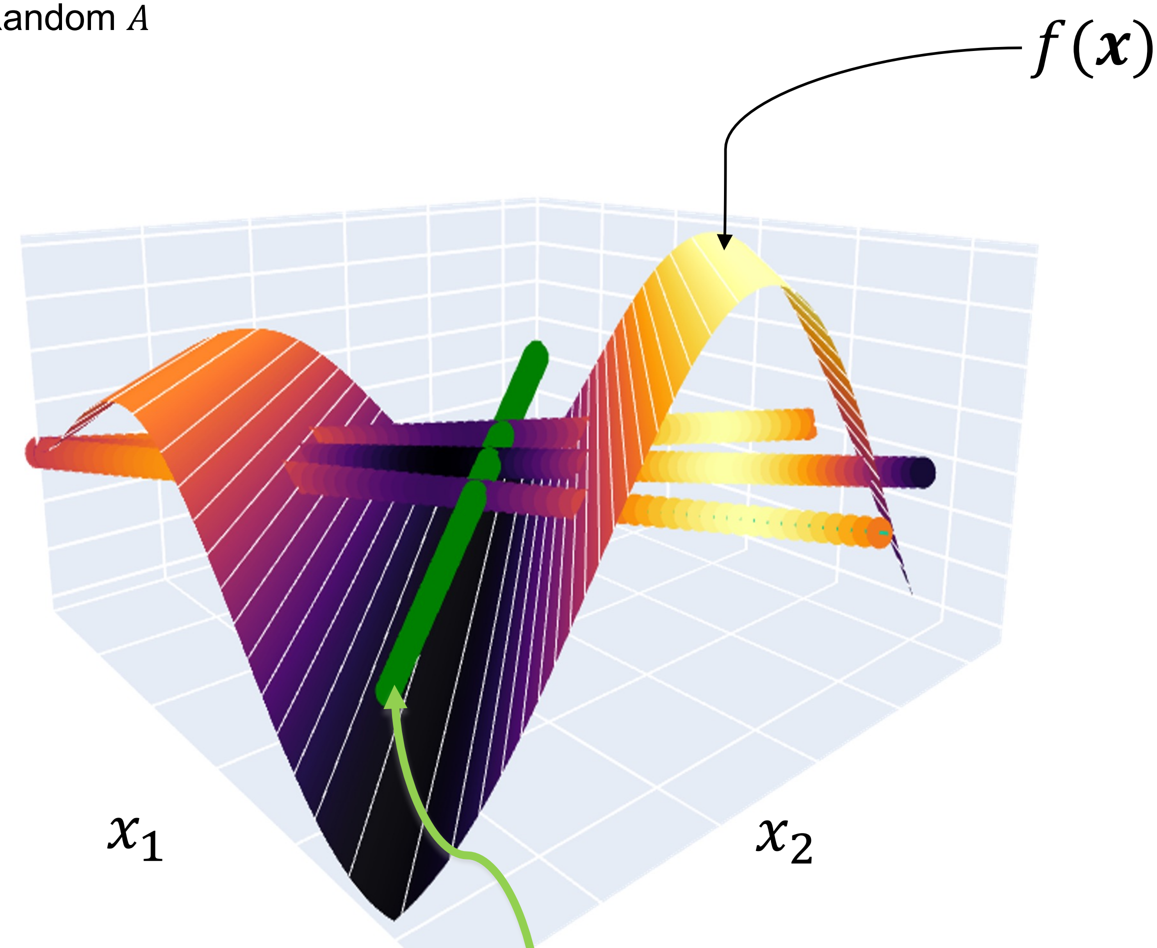




High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A



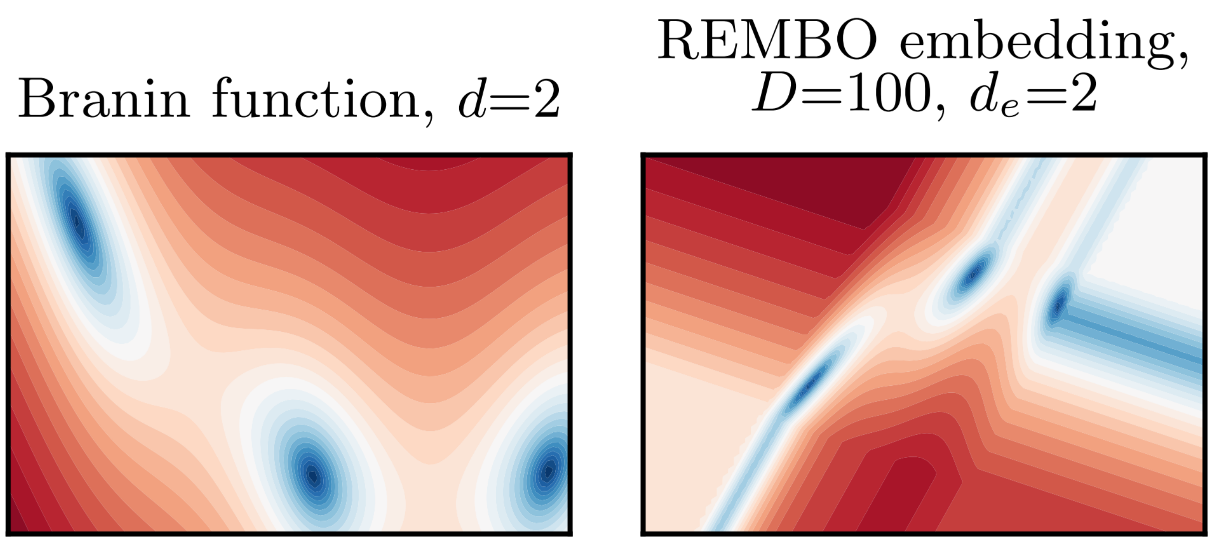


A “nearly orthogonal” to true subspace – optimum out of bounds!

High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A

Letham et al., 2020 (ALEBO): Use projection pseudo-inverse to avoid bound clipping via constrained acquisition maximization, introduce pseudo inverse into Mahalanobis metric in kernel



(Figure 1, Letham et al., 2020)

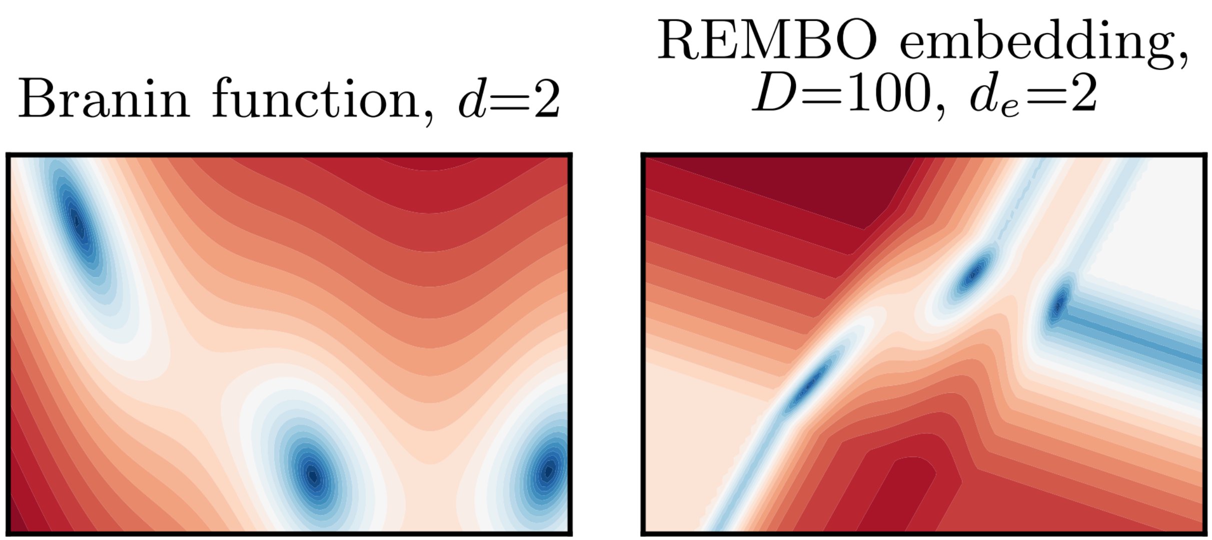
High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A

Letham et al., 2020 (ALEBO): Use projection pseudo-inverse to avoid bound clipping via constrained acquisition maximization, introduce pseudo inverse into Mahalanobis metric in kernel

Binois et al., 2020: The original space bounds clipping problem is not trivially solved by heuristic bounds in the embedding.

Munteanu et al., 2019 (HeSBO): Avoids bounds clipping via their embedding method.

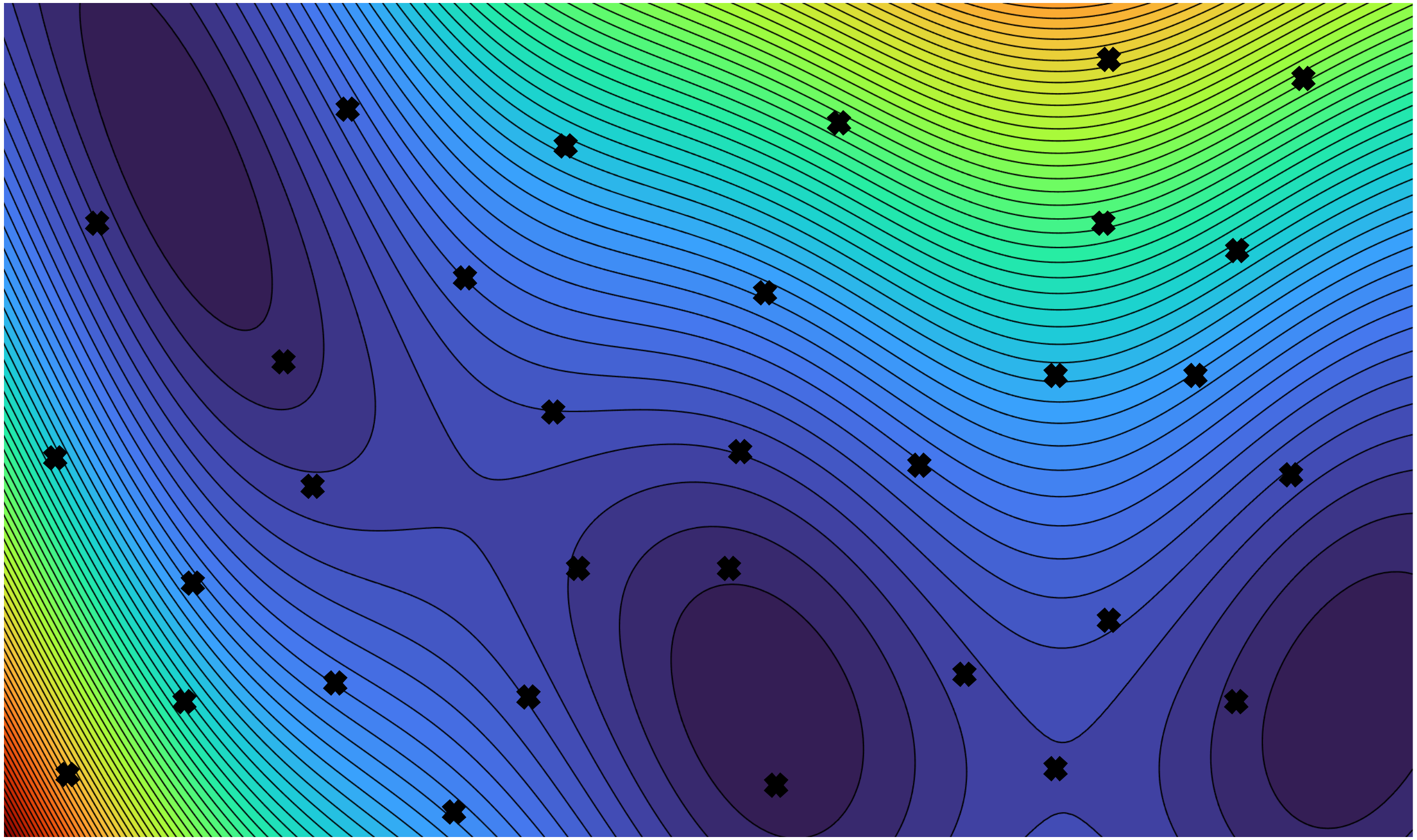


(Figure 1, Letham et al., 2020)

High Dimensional BO via Local BO

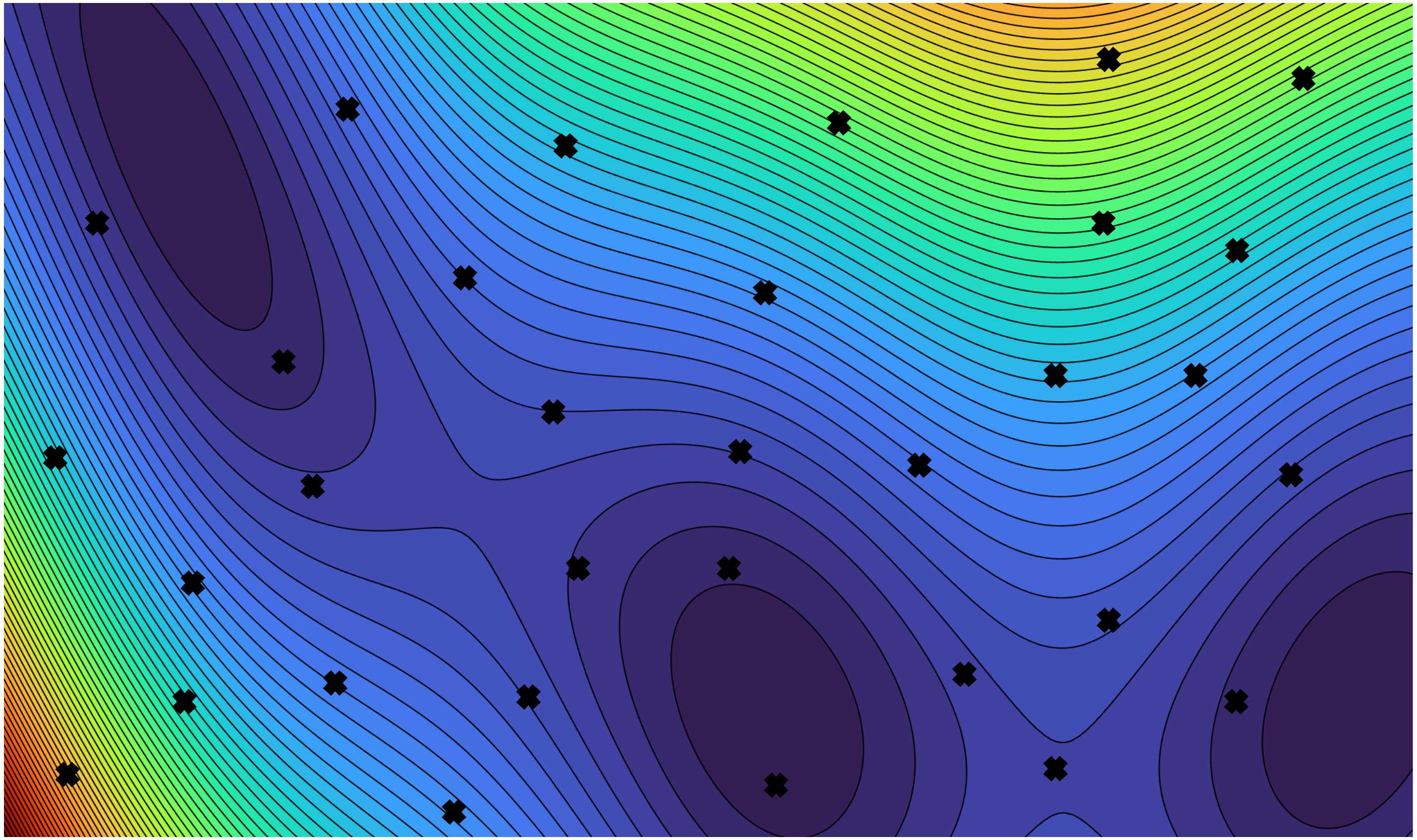
High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

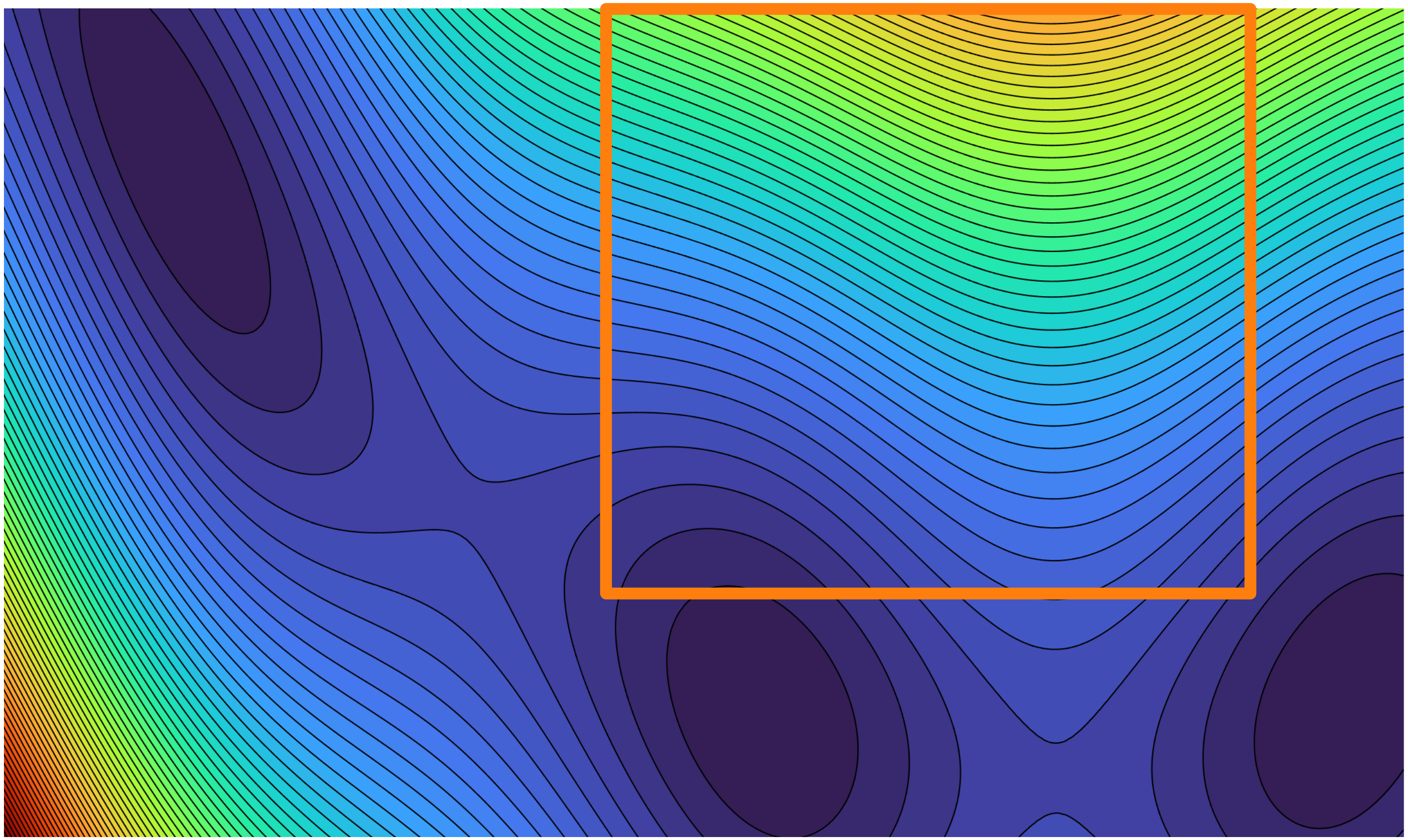


Idea: Perform BO inside a *trust region* (TuRBO)

NEURAL INFORMATION PROCESSING SYSTEMS

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

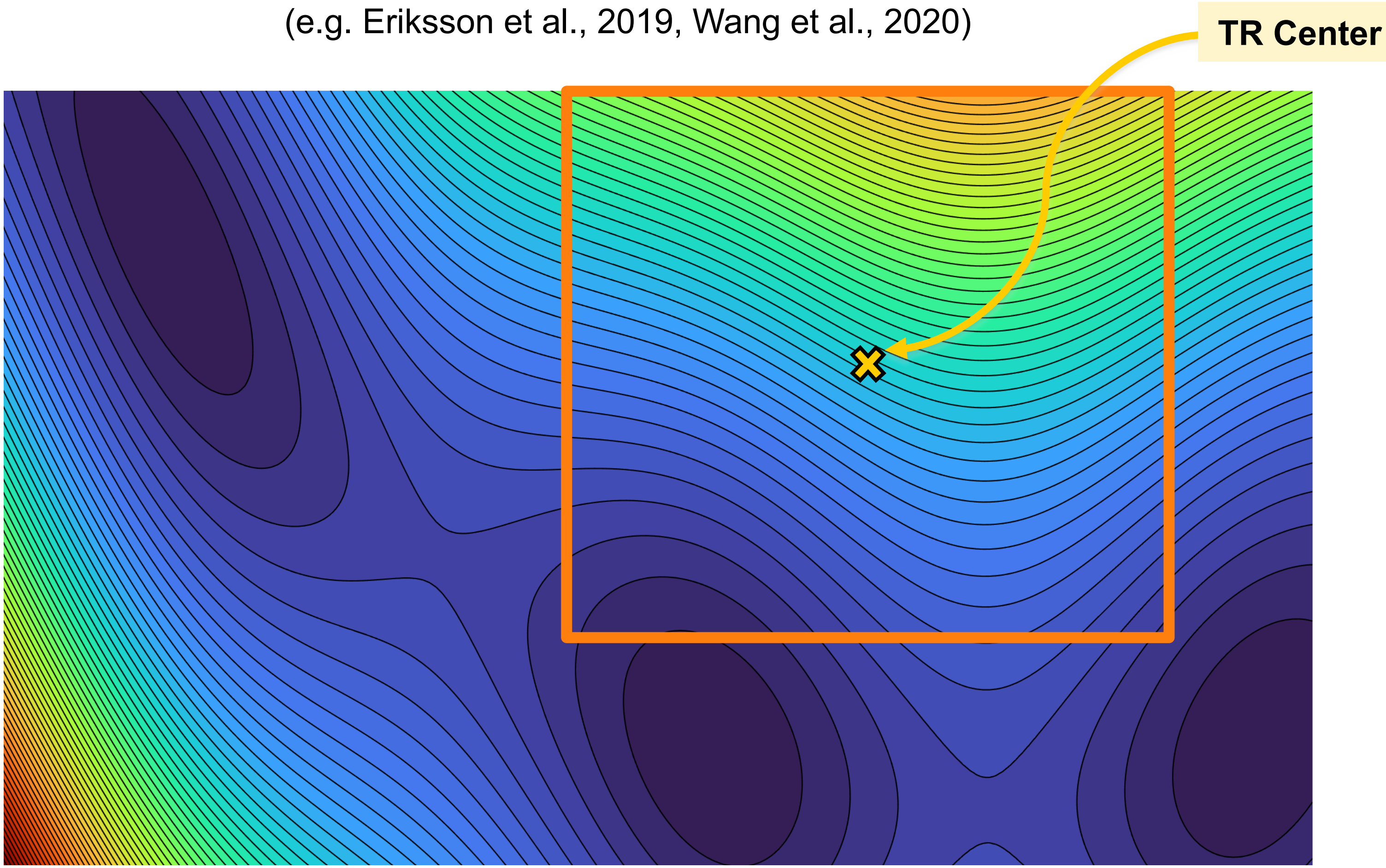


Idea: Perform BO inside a *trust region* (TuRBO)

NEURAL INFORMATION PROCESSING SYSTEMS

High Dimensional BO: Local BO

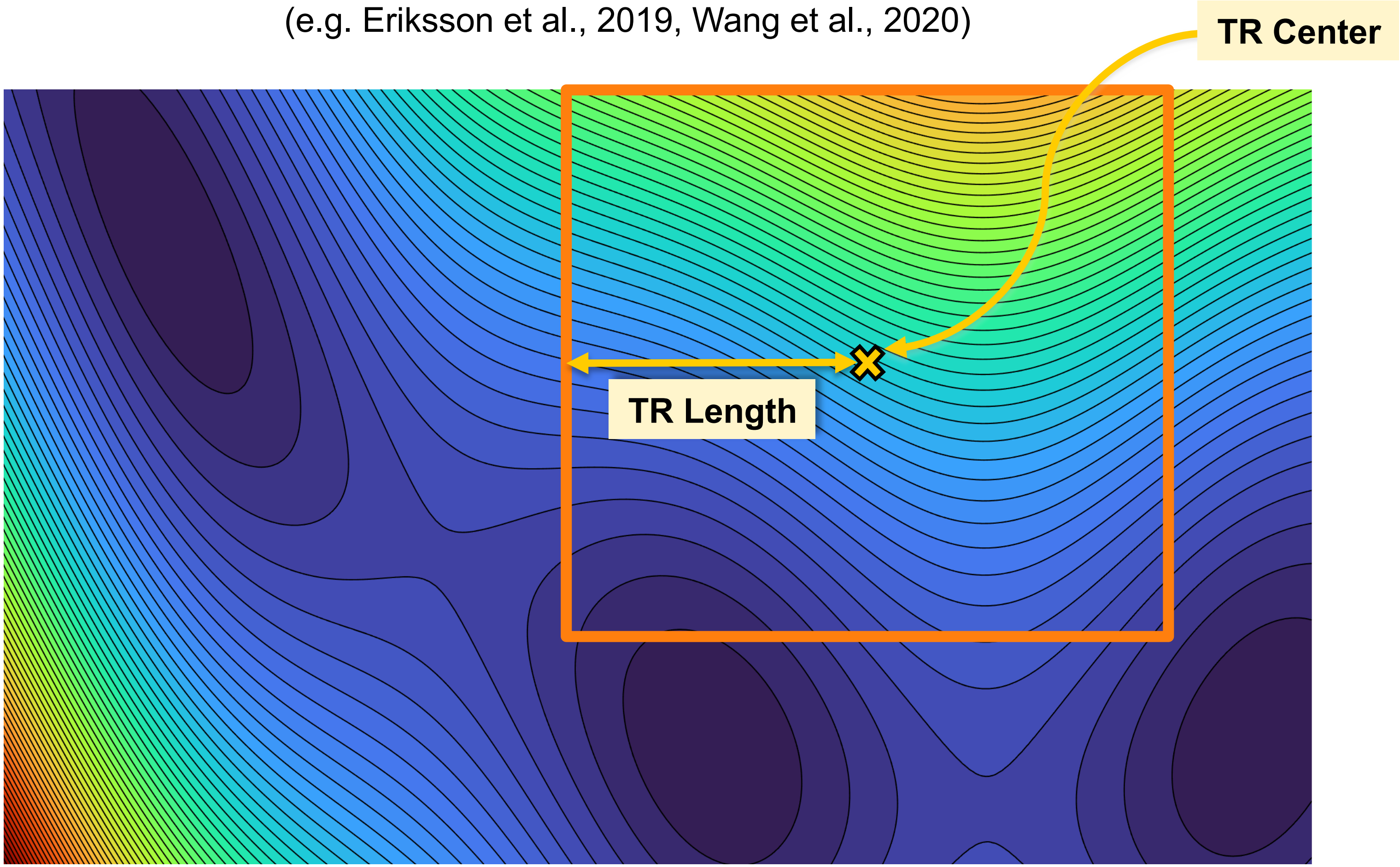
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

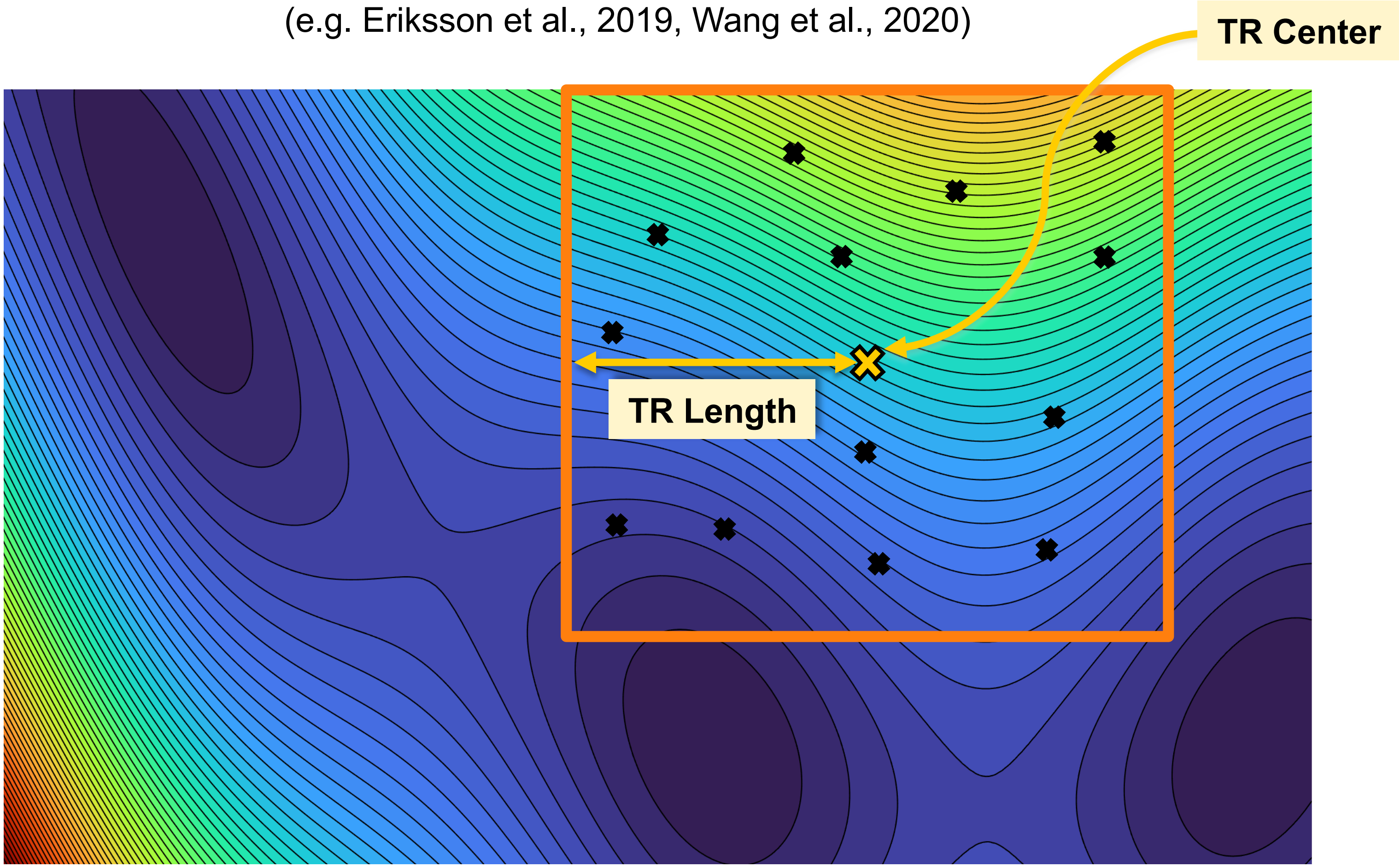
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

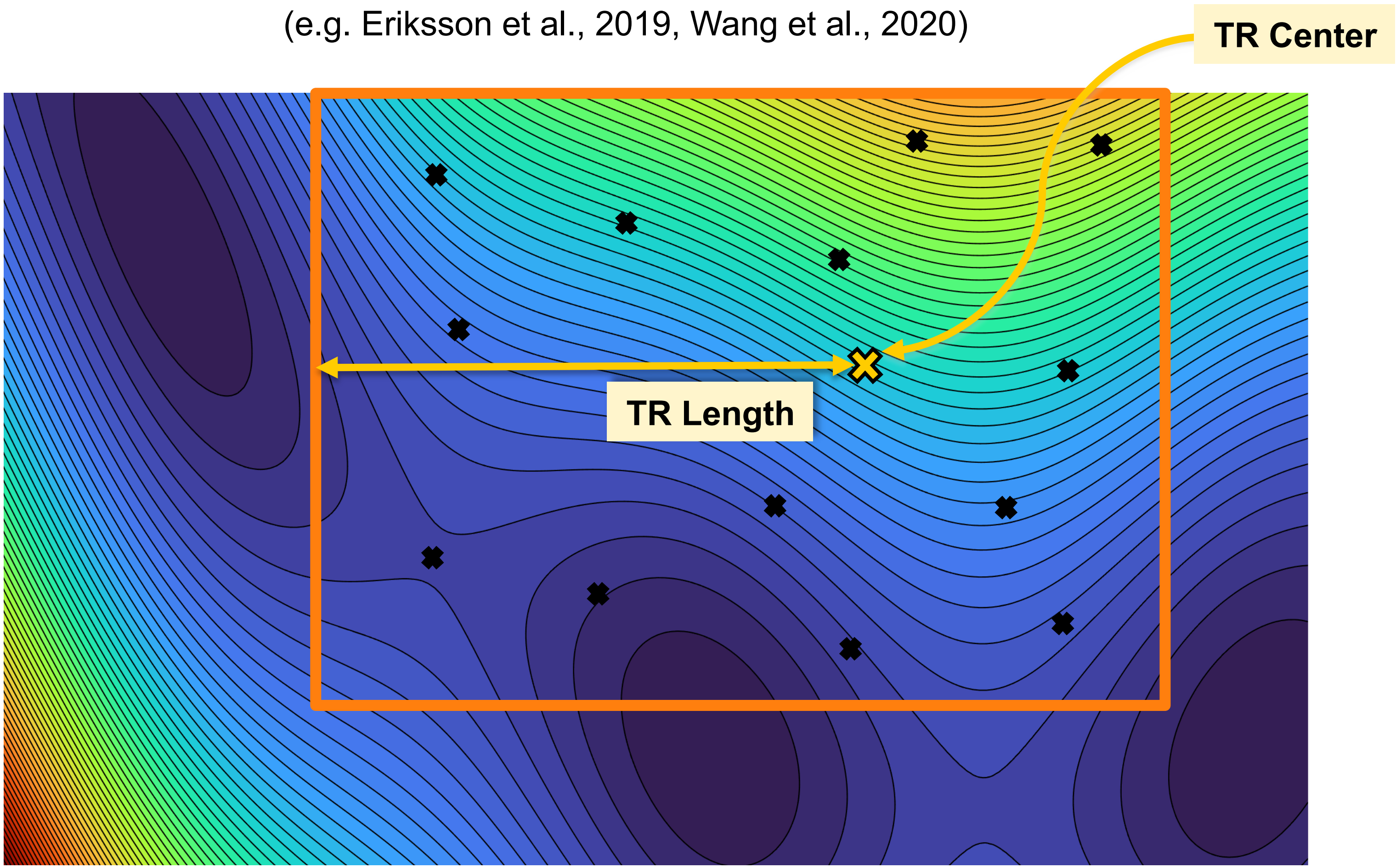


Idea: Perform BO inside a *trust region* (TuRBO)

NEURAL INFORMATION PROCESSING SYSTEMS

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

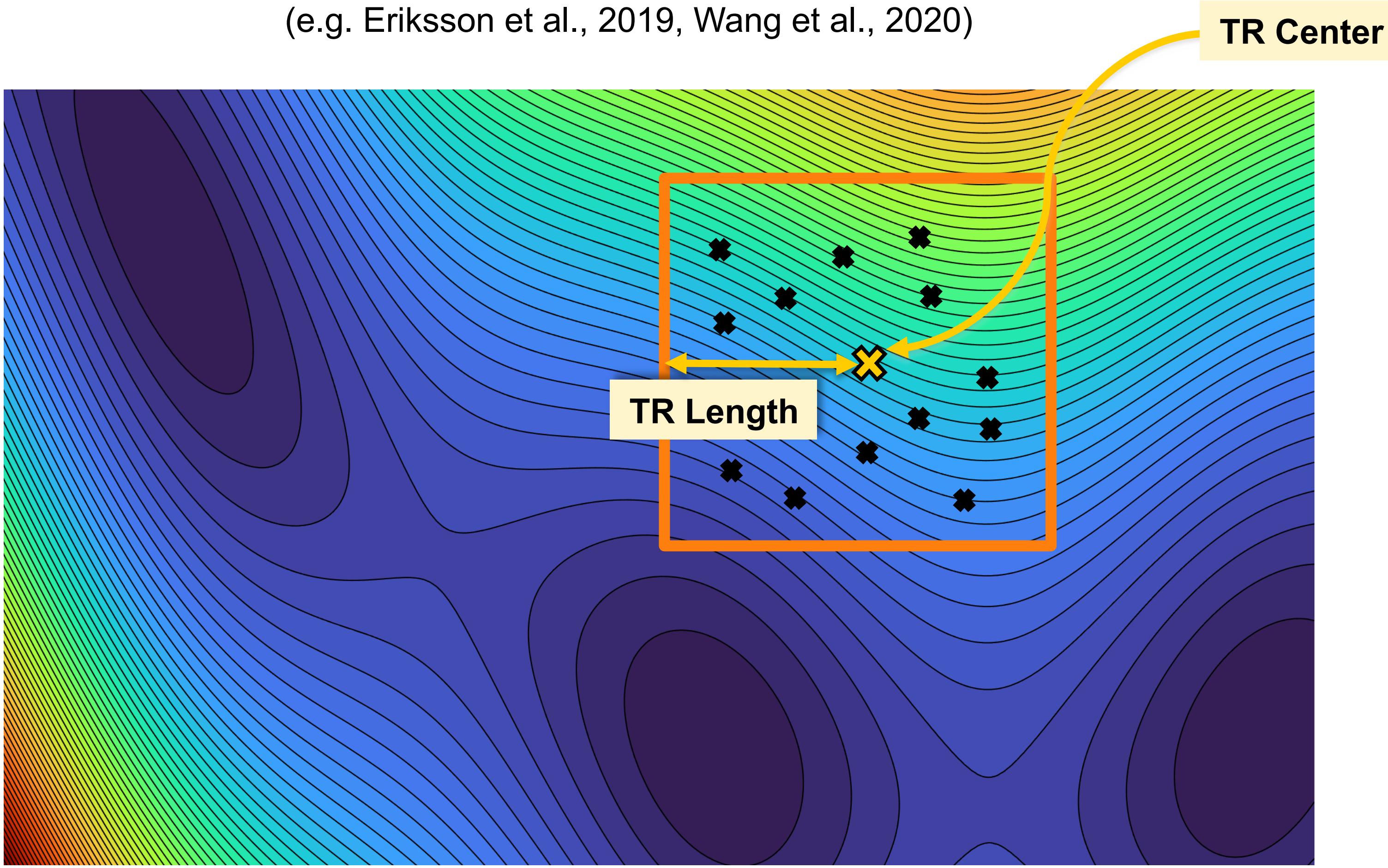


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



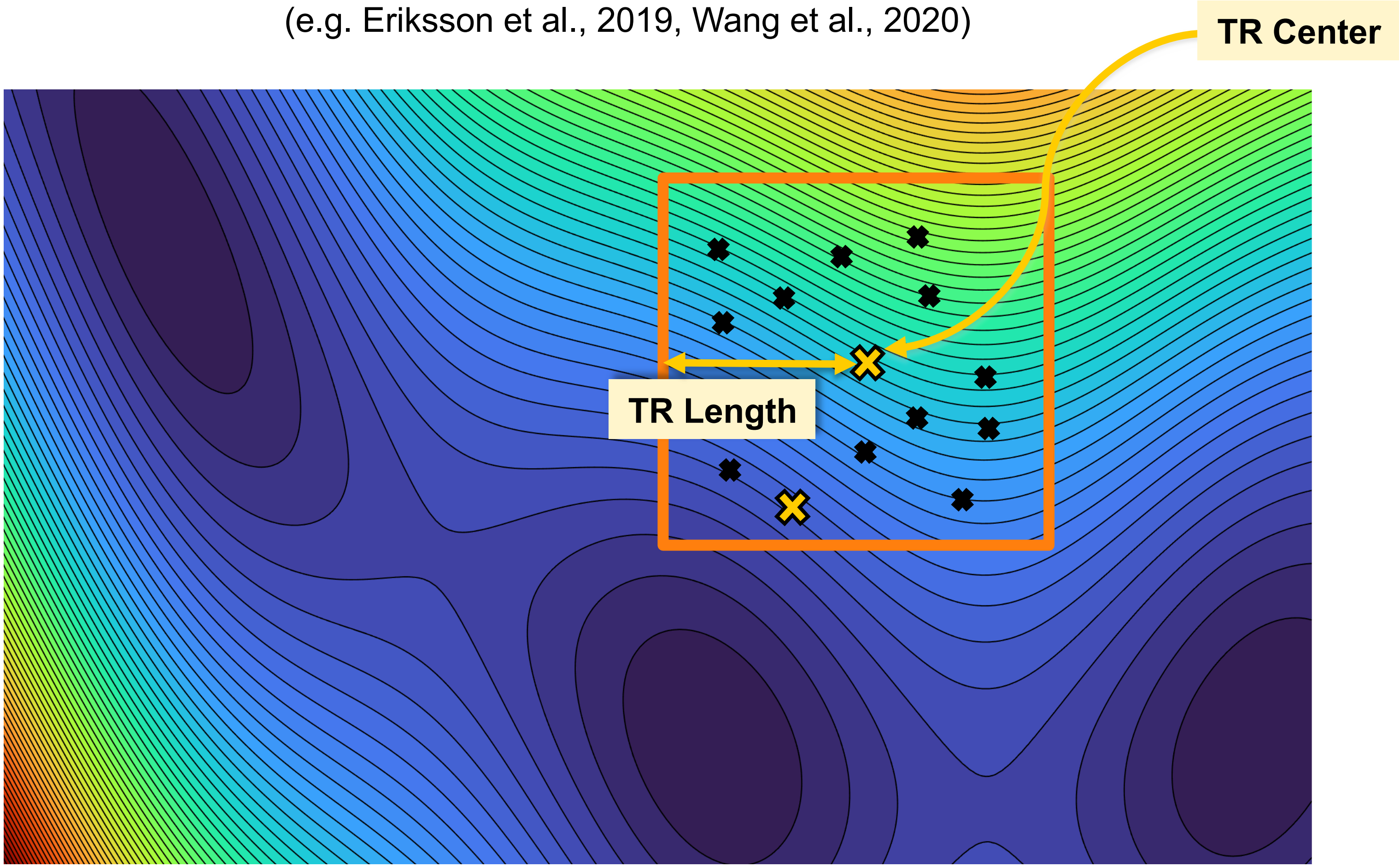
Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR**

NEURAL INFORMATION PROCESSING SYSTEMS

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



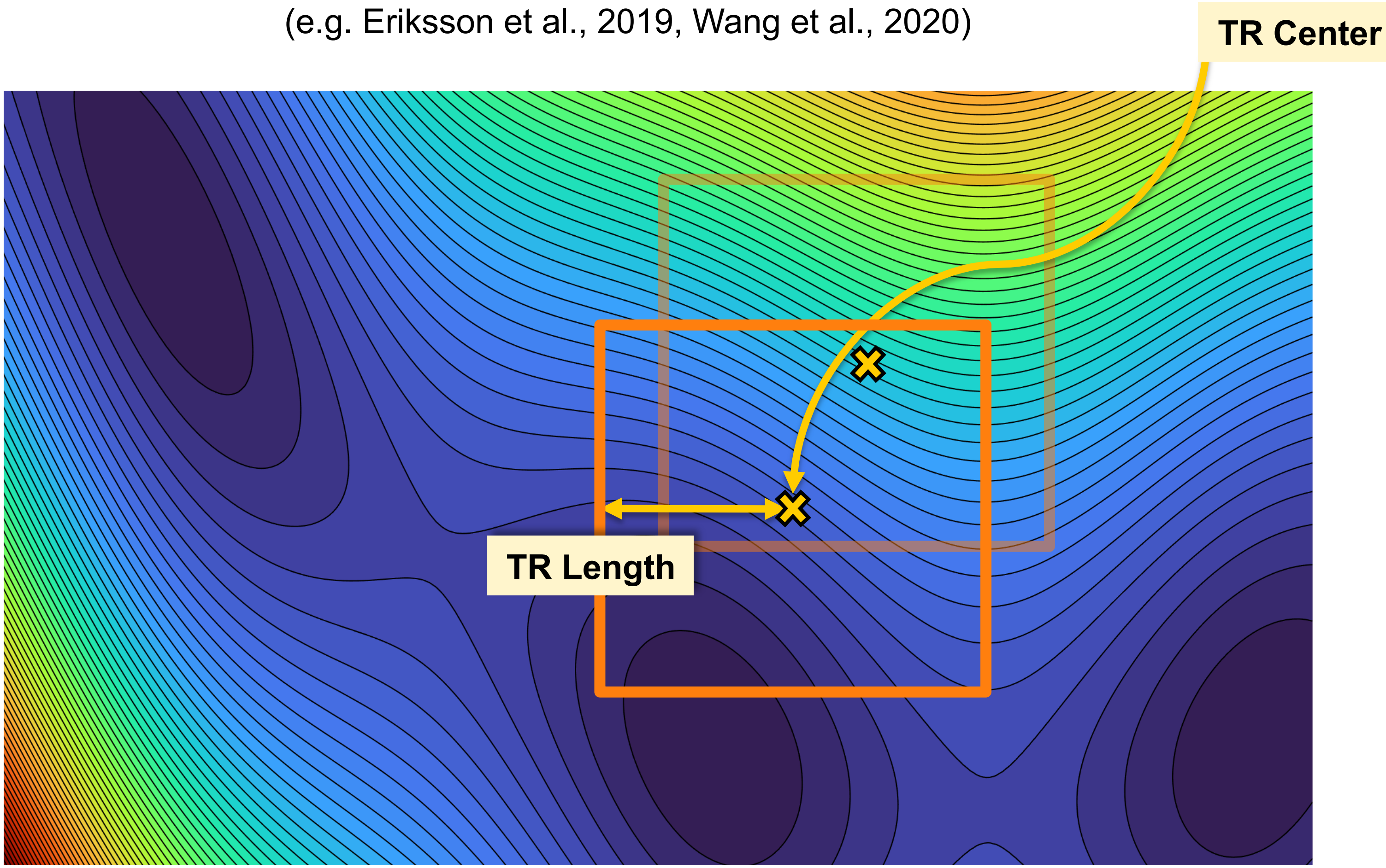
Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR

Op #2: Shrink the TR

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

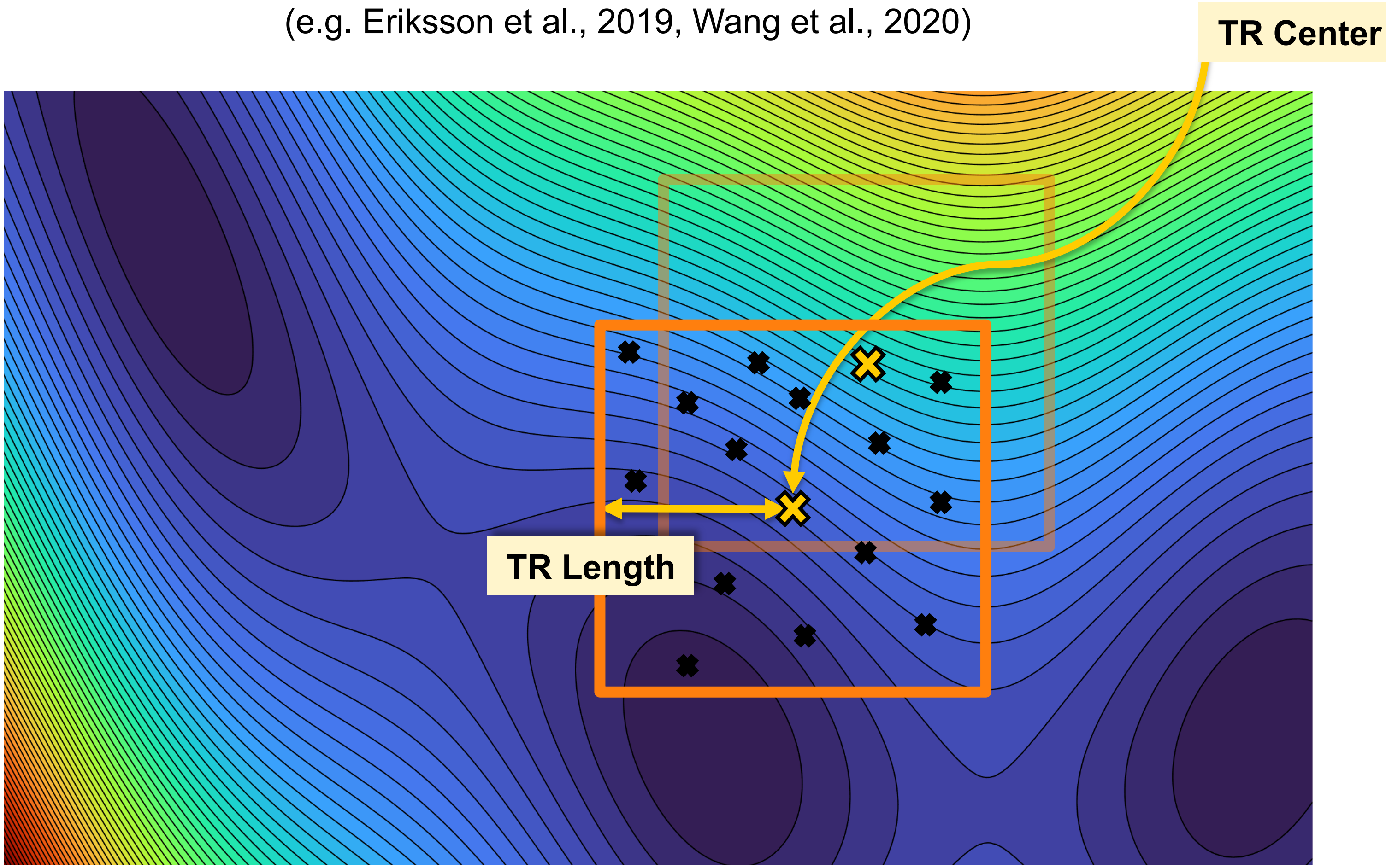


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR** **Op #3: Move the TR**

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

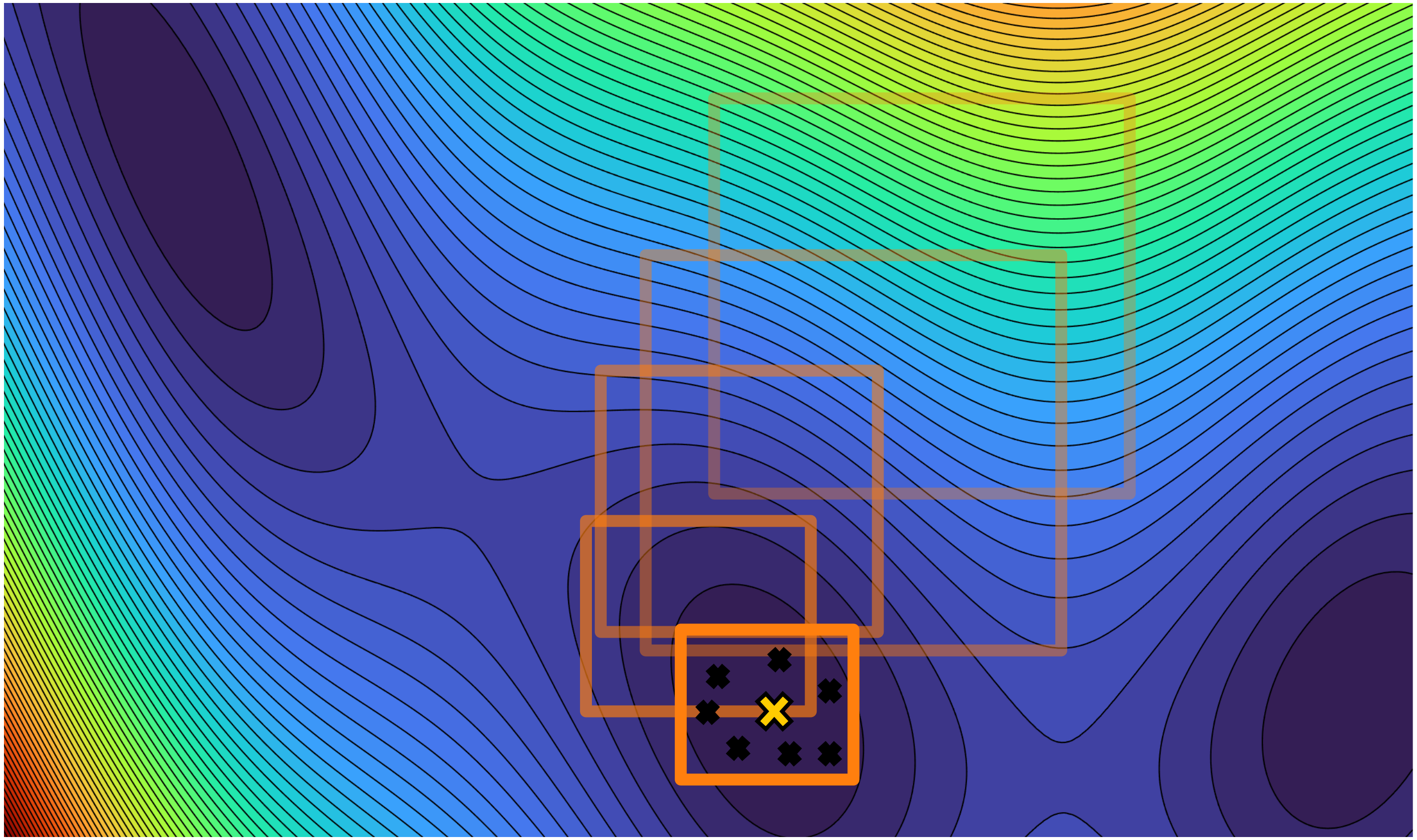


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR** **Op #3: Move the TR**

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



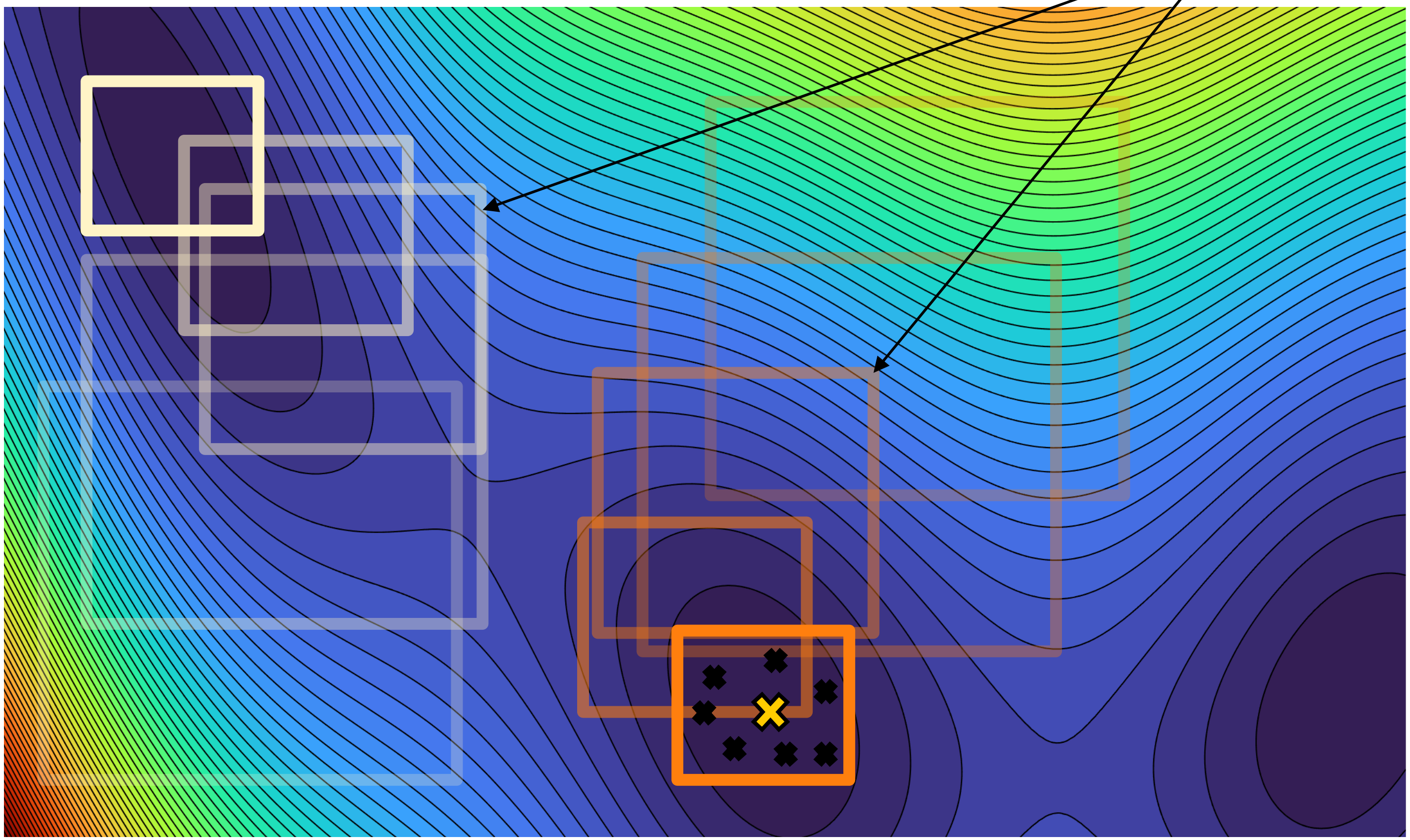
Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR** **Op #3: Move the TR**

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

Consider bandit / MCTS over multiple TRs



Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR** **Op #3: Move the TR**

GPyTorch + BoTorch Demo: TuRBO + LS-BO

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.

Decode from PyTorch VAE Model

VAE Turns real vectors into molecules:

```
In [11]: z = torch.randn(5, 256)
         smiles = vae_decode(z)
         for s in smiles:
             print(f"{s}")
```

```
CC(CCO)C(=O)CN(O)C=CC=C(NC=NC=CCC)OCC
CCOC(=O)C(C1=CC2=CC=NC=C2)N=CC(=O)CCOCN1C(=O)O
CC1=NC=CC=C1N(C=CC=CC(=CC#N)C=CC2=NC3)CC(C#N)=C2[NH1]3
CCCC(=O)C(O)CC1=CC=CC(=C1)[NH1]C(=O)CCCNC(=O)C(=O)O
CCC1=CC=CC(=C1)NC(=O)C=CC=CC(C(=O)OC2=CC=CC(C)=C2[NH1]C=N)C
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.

Decode from PyTorch VAE Model

VAE Turns real vectors into molecules:

In [11]:
```
z = torch.randn(5, 256)
smiles = vae_decode(z)
for s in smiles:
    print(f"{s}")
```

```
CC(CCO)C(=O)CN(O)C=CC=C(NC=NC=CCC)OCC
CCOC(=O)C(C1=CC2=CC=NC=C2)N=CC(=O)CCOCN1C(=O)O
CC1=NC=CC=C1N(C=CC=CC(=CC#N)C=CC2=NC3)CC(C#N)=C2[NH1]3
CCCC(=O)C(O)CC1=CC=CC(=C1)[NH1]C(=O)CCCNC(=O)C(=O)O
CCC1=CC=CC(=C1)NC(=O)C=CC=CC(C(=O)OC2=CC=CC(C)=C2[NH1]C=N)C
```

In [12]:
```
obj_func = lambda z: smile_to_penalized_logP(vae_decode(z)[0])
obj_func(torch.randn(1, 256))
```

Out[12]: -29.951505411029295

In [13]:
```
obj_func = lambda z: smile_to_guacamol_score('rano', vae_decode(z)[0])
obj_func(torch.randn(1, 256))
```

Out[13]: 0.1917855978072532

Brown et al., 2019

Take away: The "LS" part is pretty much done for you. If that's what you want.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.

2. Train a surrogate model

Pre-baked BoTorch model:

Inducing points initialized using Pivoted Cholesky (Burt et al., 2020)

In [25]:
```
from gpytorch.kernels import MaternKernel, ScaleKernel
from gpytorch.likelihoods import GaussianLikelihood
from botorch.models.approximate_gp import SingleTaskVariationalGP

likelihood = GaussianLikelihood().to(device='cuda:0')
covar_module = ScaleKernel(MaternKernel(nu=2.5)).to(device='cuda:0')
model = SingleTaskVariationalGP(X, Y, inducing_points=1024, likelihood=likelihood, covar_module=covar_module)
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.

2. Train a surrogate model

Inducing points initialized using Pivoted Cholesky (Burt et al., 2020)

Pre-baked BoTorch model:

In [25]:
```python
from gpytorch.kernels import MaternKernel, ScaleKernel
from gpytorch.likelihoods import GaussianLikelihood
from botorch.models.approximate_gp import SingleTaskVariationalGP

likelihood = GaussianLikelihood().to(device='cuda:0')
covar_module = ScaleKernel(MaternKernel(nu=2.5)).to(device='cuda:0')
model = SingleTaskVariationalGP(X, Y, inducing_points=1024, likelihood=likelihood, covar_module=covar_module)
```

Training:

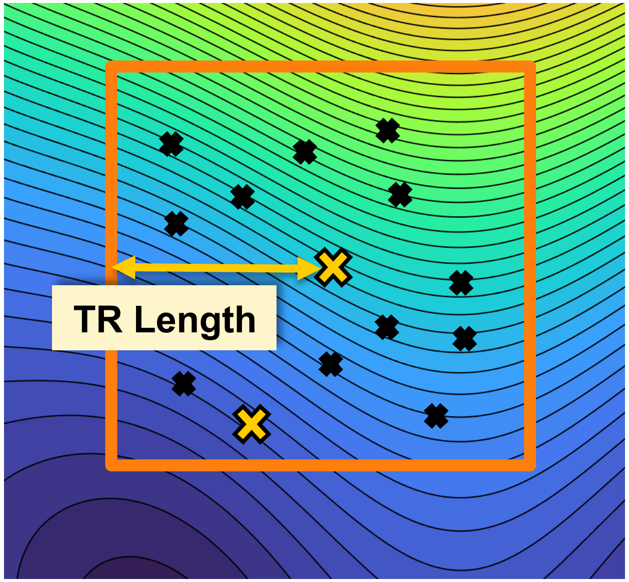
```python
# PPGPR (Jankowiak et al., 2020)
mll = PredictiveLogLikelihood(likelihood, model.model, num_data=X.size(-2))
# SVGP (Hensman et al., 2013)
mll = VariationalELBO(likelihood, model.model, num_data=X.size(-2))
```

```python
model = model.train()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
train_dataset = TensorDataset(train_z, train_y)
train_loader = DataLoader(train_dataset, batch_size=min(len(train_y), 128), shuffle=True)
for _ in range(n_epochs):
    for (inputs, scores) in train_loader:
        optimizer.zero_grad()
        output = model(inputs)
        loss = -mll(output, scores)
        loss.backward()
        optimizer.step()
```

(Soon: https://github.com/pytorch/botorch/pull/1439/)

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**
3. **Trust region state**

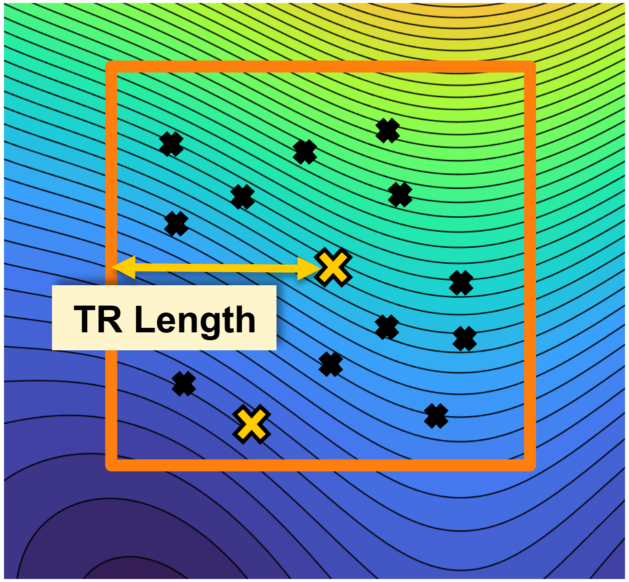


GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.

2. Train a surrogate model

3. Trust region state

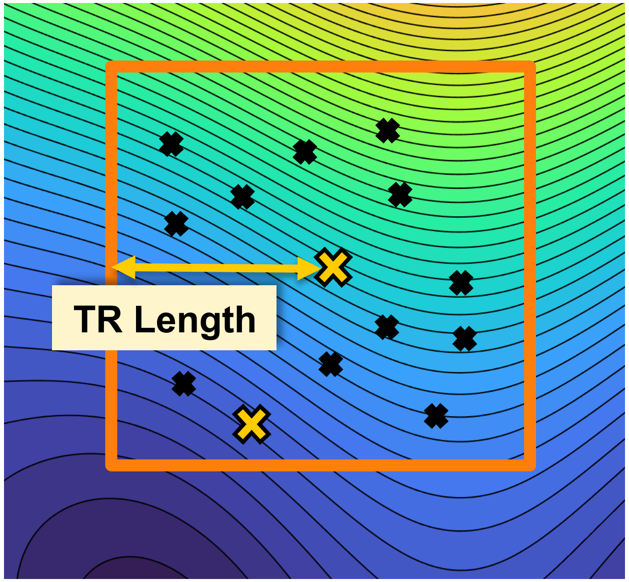



```
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

TR length $\in [0, 1]$
($\times$ some fixed init. Length)

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**
3. **Trust region state**

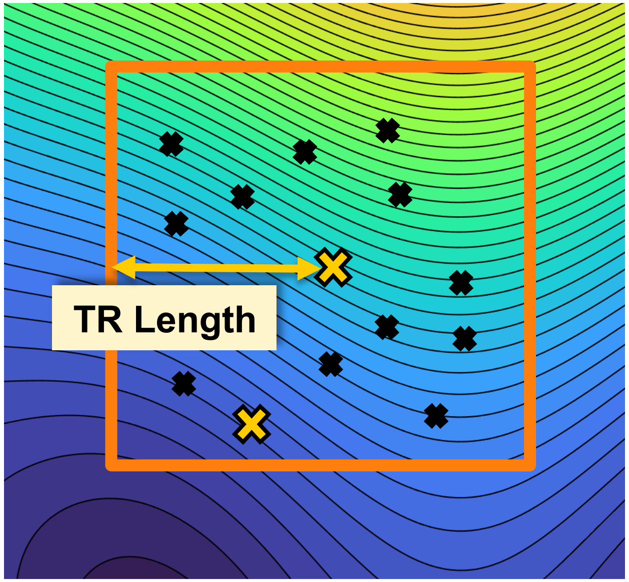



```
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

Shrink TR if we fail to make progress 5 times in a row.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the “LS” bit.**
2. **Train a surrogate model**
3. **Trust region state**



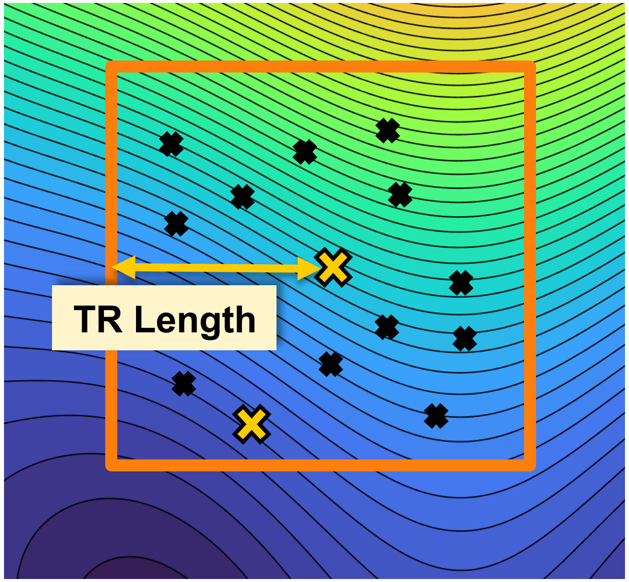

```
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

Grow TR if we make progress 5 times in a row.

NEURAL INFORMATION PROCESSING SYSTEMS

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state




```python
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

Grow TR if we make progress 5 times in a row.

```python
def update_state(state, Y_next):
    if max(Y_next) > state.best_value + 1e-3 * math.fabs(state.best_value):
        state.success_counter += 1
        state.failure_counter = 0
    else:
        state.success_counter = 0
        state.failure_counter += 1

    if state.success_counter == state.success_tolerance:  # Expand trust region
        state.length = min(2.0 * state.length, state.length_max)
        state.success_counter = 0
    elif state.failure_counter == state.failure_tolerance:  # Shrink trust region
        state.length /= 2.0
        state.failure_counter = 0

    state.best_value = max(state.best_value, max(Y_next).item())
    if state.length < state.length_min:
        state.restart_triggered = True
    return state
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function

NEURAL INFORMATION PROCESSING SYSTEMS

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function

```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

Monte-Carlo Acquisition functions (Wilson et al., 2018)

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**
3. **Trust region state**
4. **Maximize an acquisition function**

```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

Monte-Carlo Acquisition functions (Wilson et al., 2018)

Batch size = 10

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function

```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

Run local optimization from 10 initial conditions.

ICs chosen from among 512 Sobol samples.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**
3. **Trust region state**
4. **Maximize an acquisition function**

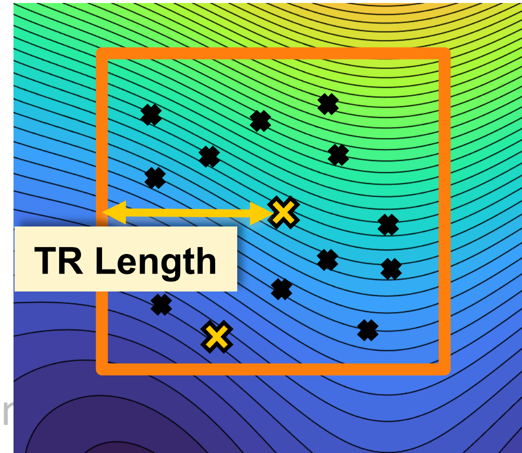
```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**
3. **Trust region state**
4. **Maximize an acquisition function**

```
INIT_TR_LENGTH = 3

x_center = X[Y.argmax(), :]
lb = x_center - INIT_TR_LENGTH * state.length
ub = x_center + INIT_TR_LENGTH * state.length
```

```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```



GPyTorch + BoTorch Demo: TuRBO + LS-BO

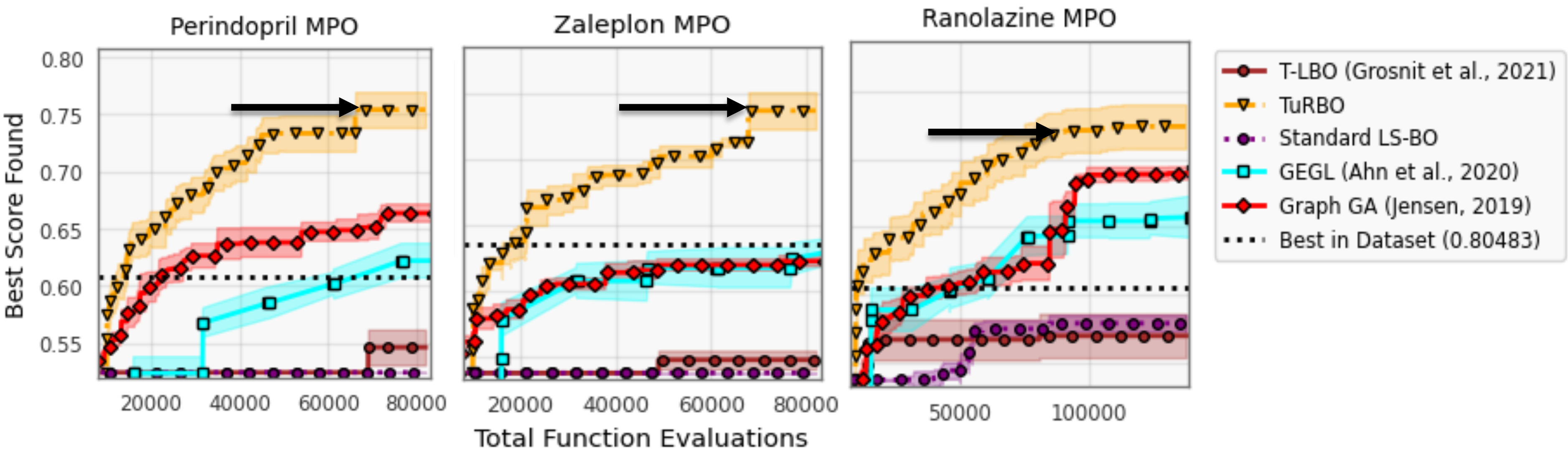
1. Handling the "LS" bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function
4. Update state

```
# Decode batch to smiles, get logP values.
Y_next = torch.tensor([obj_func(x) for x in X_next], dtype=vae.dtype, device=vae.device).unsqueeze(-1)

# Update TuRBO state
state = update_state(state=state, Y_next=Y_next)

# Add data
X = torch.cat((X, X_next), dim=-2)
Y = torch.cat((Y, Y_next), dim=-2)
```

Results

$\log P$: ~140 in 200-300 steps.

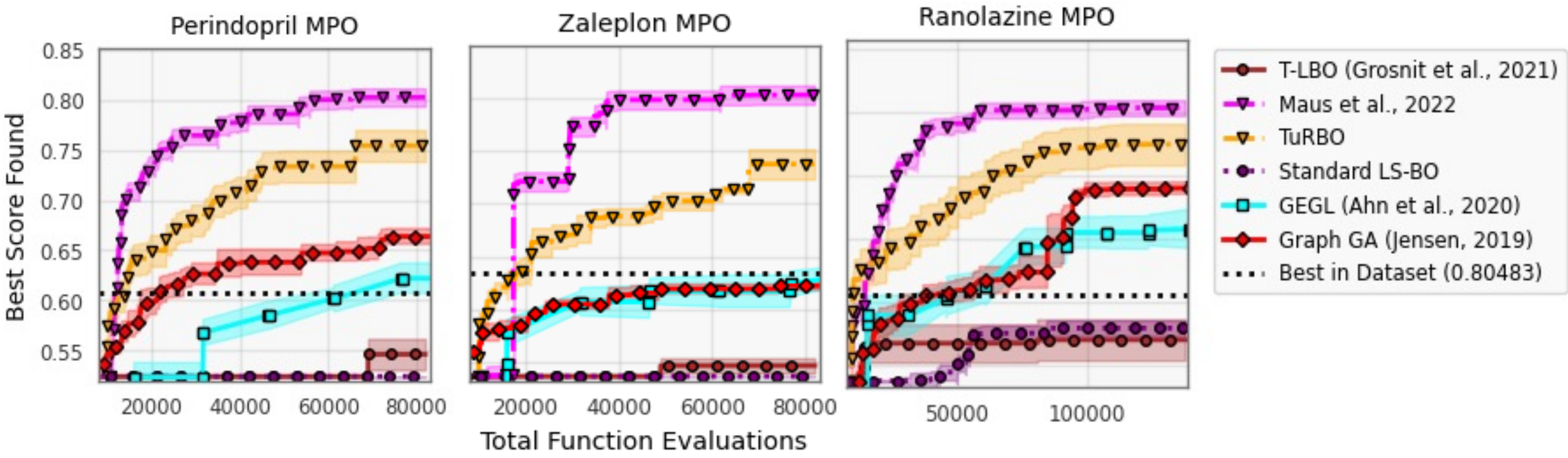
Guacamol:



Results

$\log P$: ~140 in 200-300 steps.

Guacamol:



Thanks!