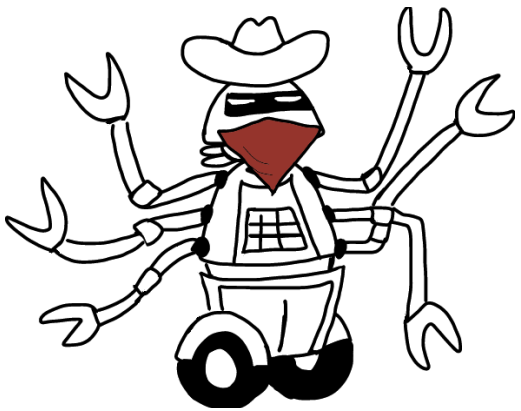
Scaling up Bandits and Friends



Kirthevasan Kandasamy

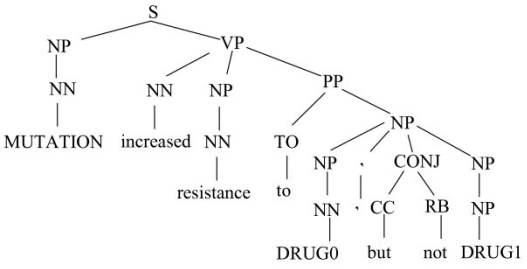
Carnegie Mellon University

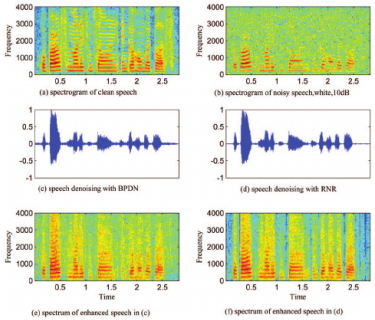
Apr 30, 2019

UC Berkeley

AI & ML Today

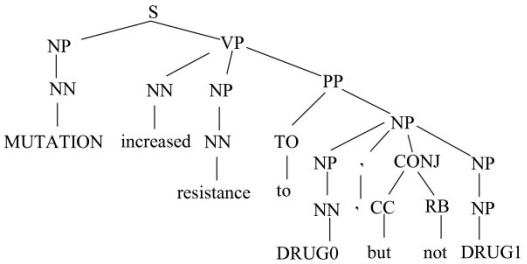
Supervised learning

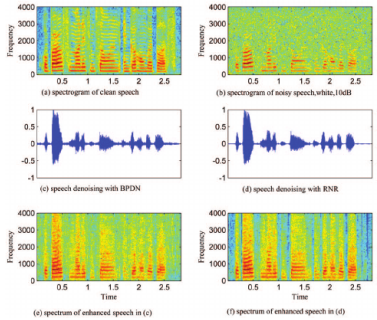




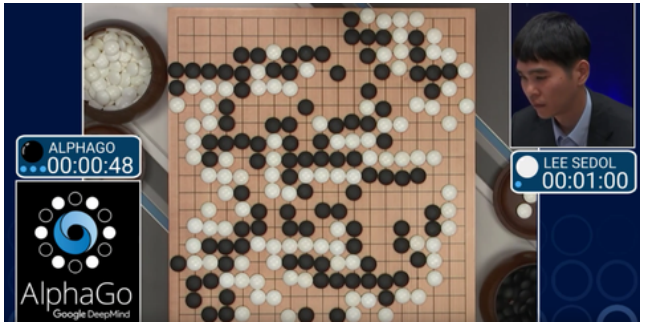
AI & ML Today

Supervised learning



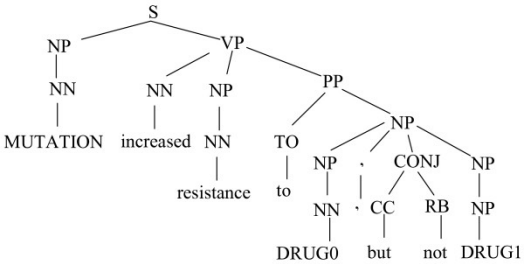


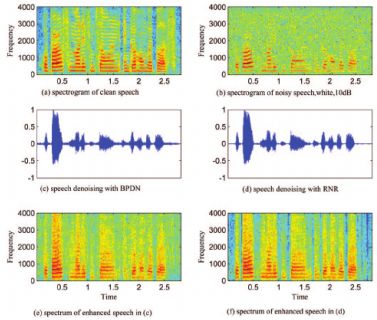
Reinforcement learning



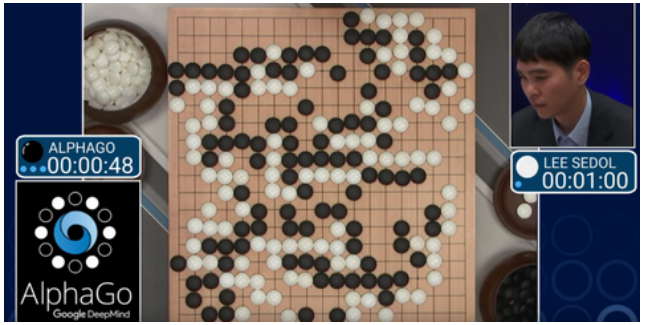
AI & ML Today

Supervised learning





Reinforcement learning



A lot of success in RL when data is abundant.

Black-box Optimisation: Model Selection

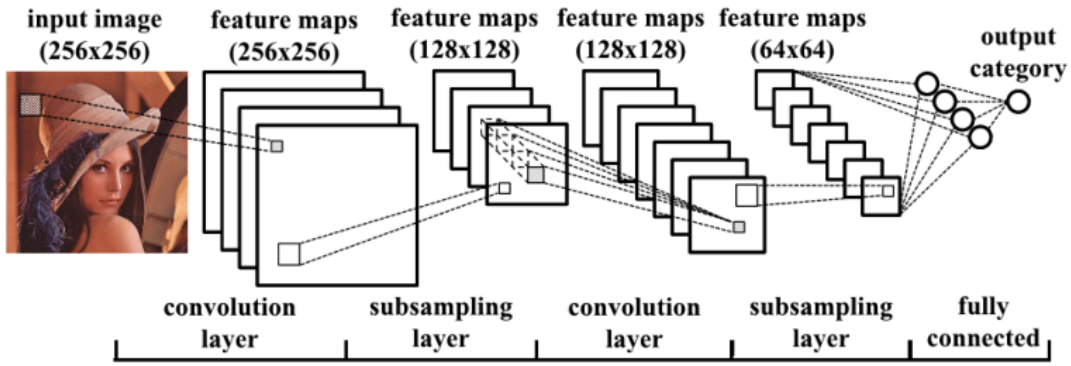
$x \longrightarrow$ **Neural Network** $\longrightarrow Y_x$

cross validation accuracy

$\mathcal{X}$: hyper-parameter values

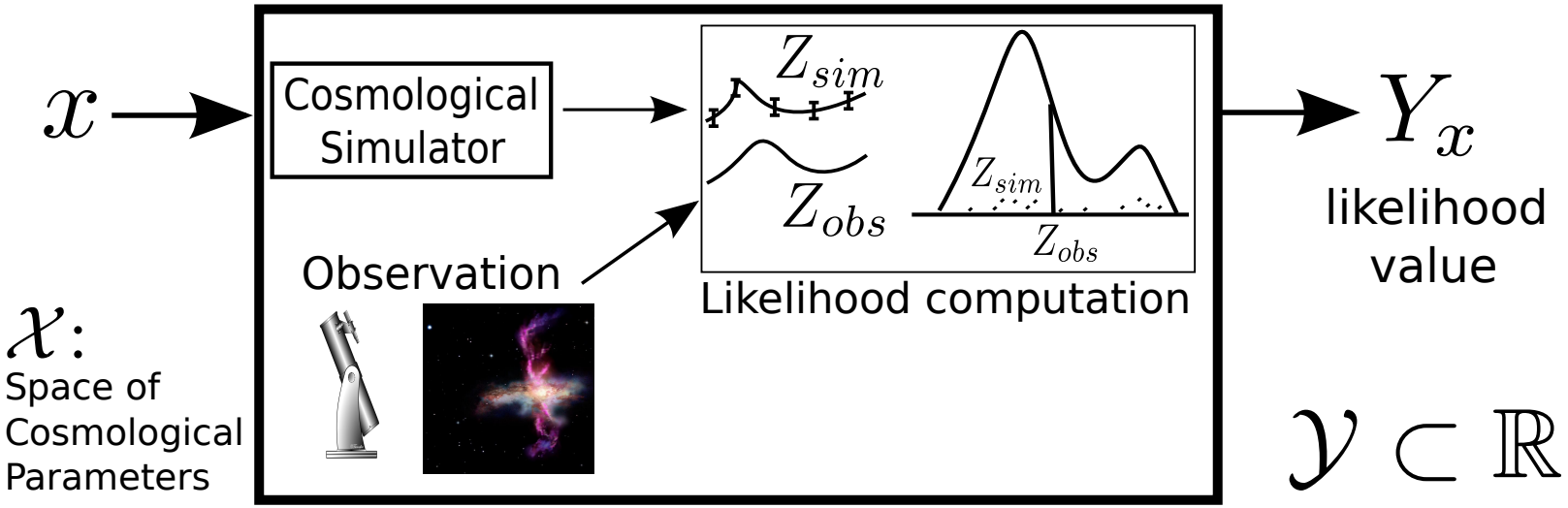
- Train NN using given hyper-params
- Compute accuracy on validation set

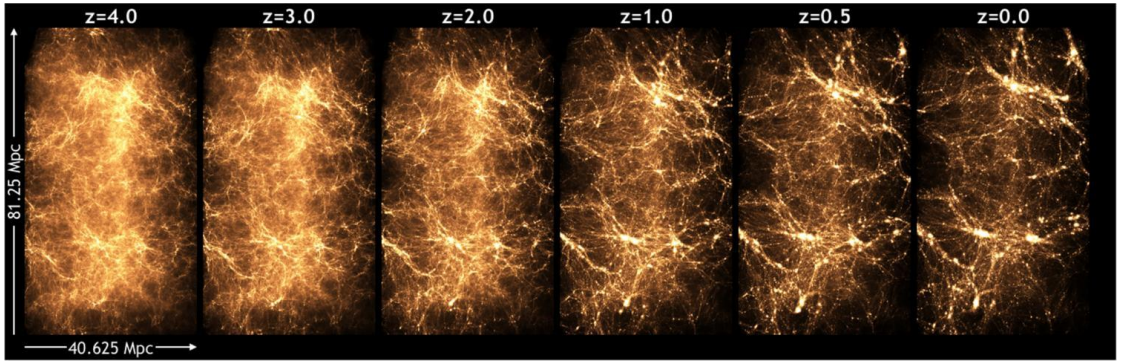
$\mathcal{Y} = [0, 1]$



Goal: Find hyperparameters with highest validation accuracy.

Black-box Optimisation: Astrophysics





Goal: Estimate cosmological parameters by maximising likelihood.

Outline

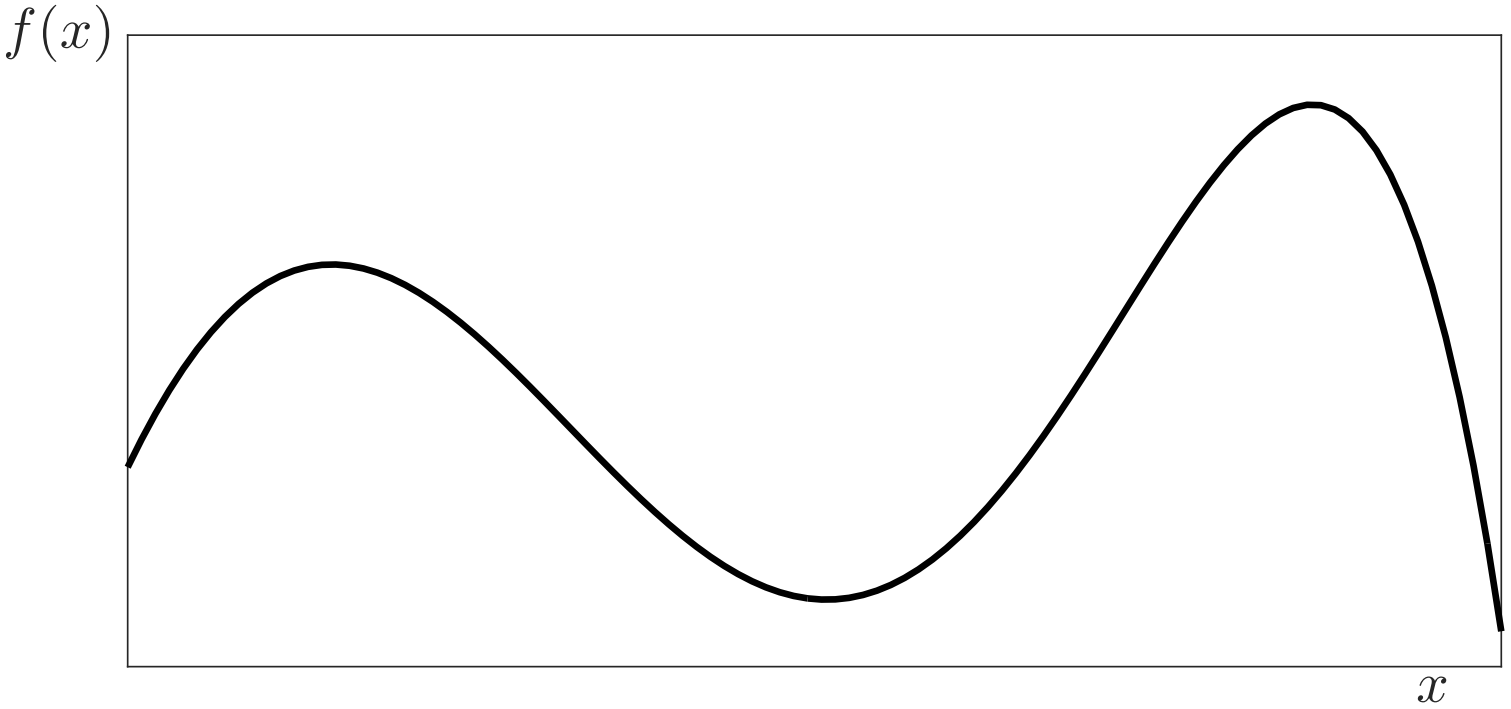
- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

Outline

- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

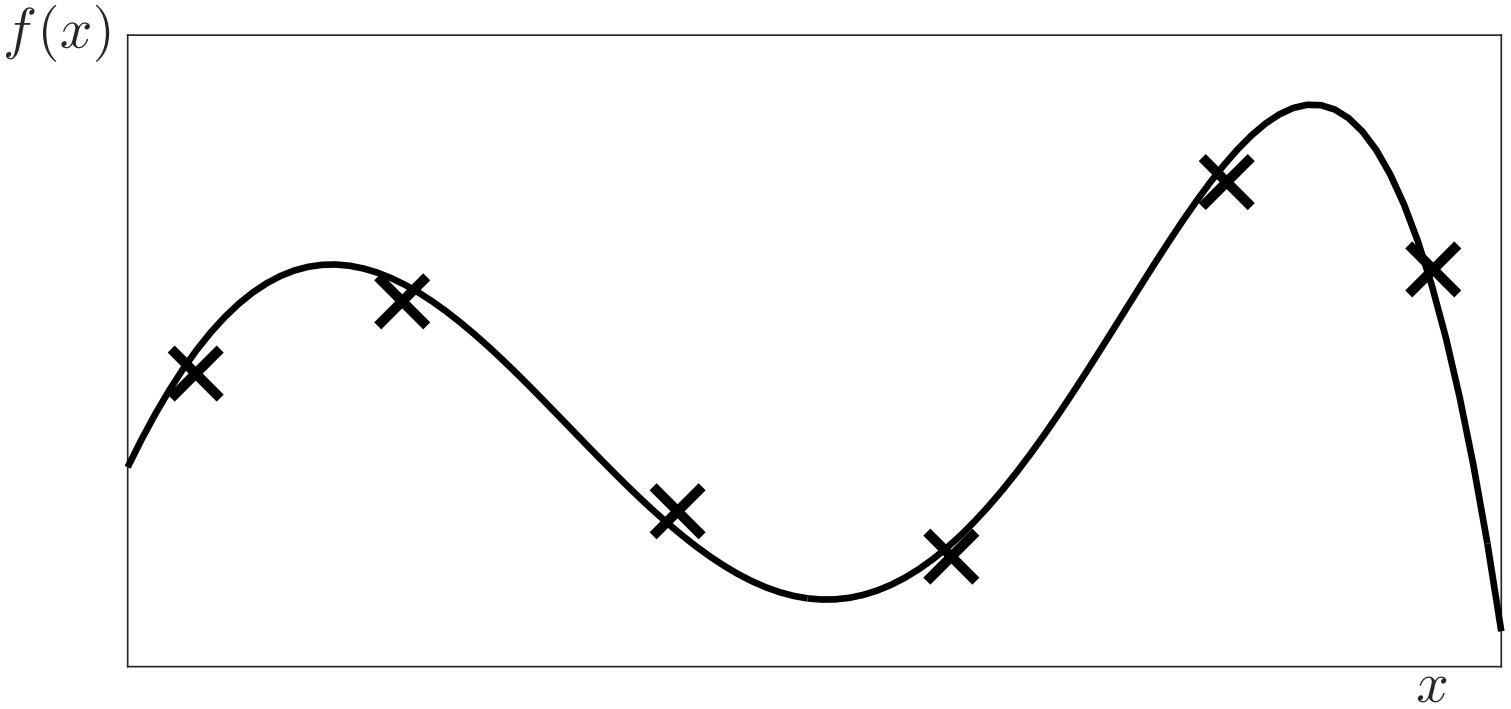
Bandits/Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.



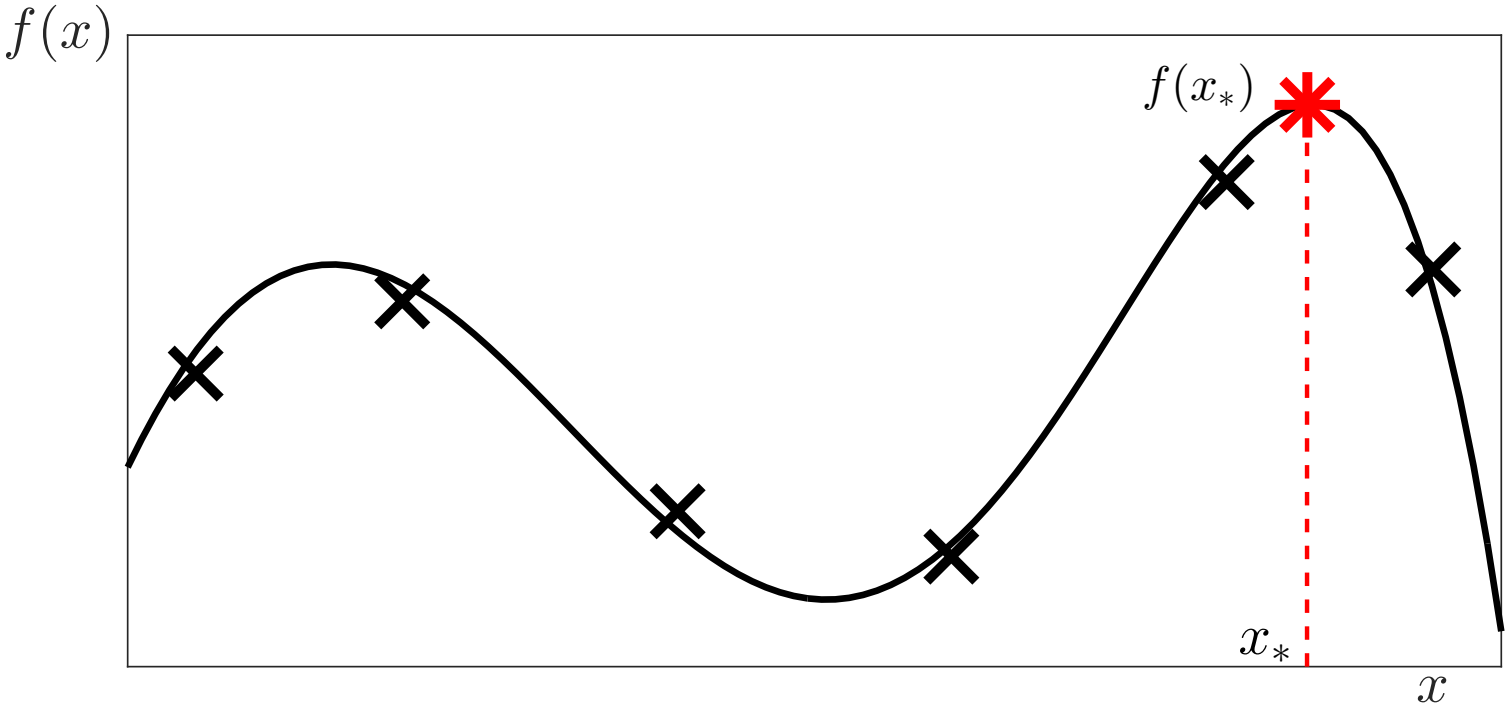
Bandits/Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.



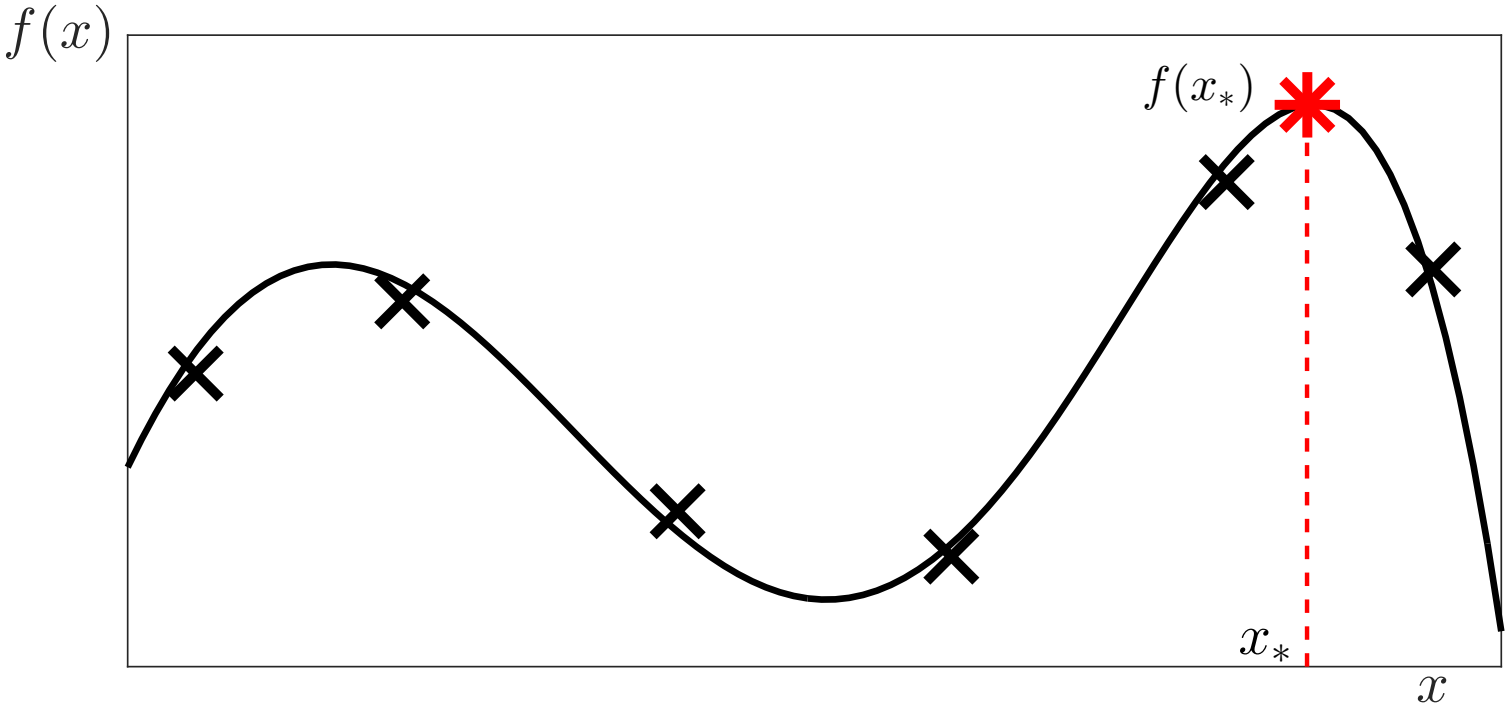
Bandits/Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.



Bandits/Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.

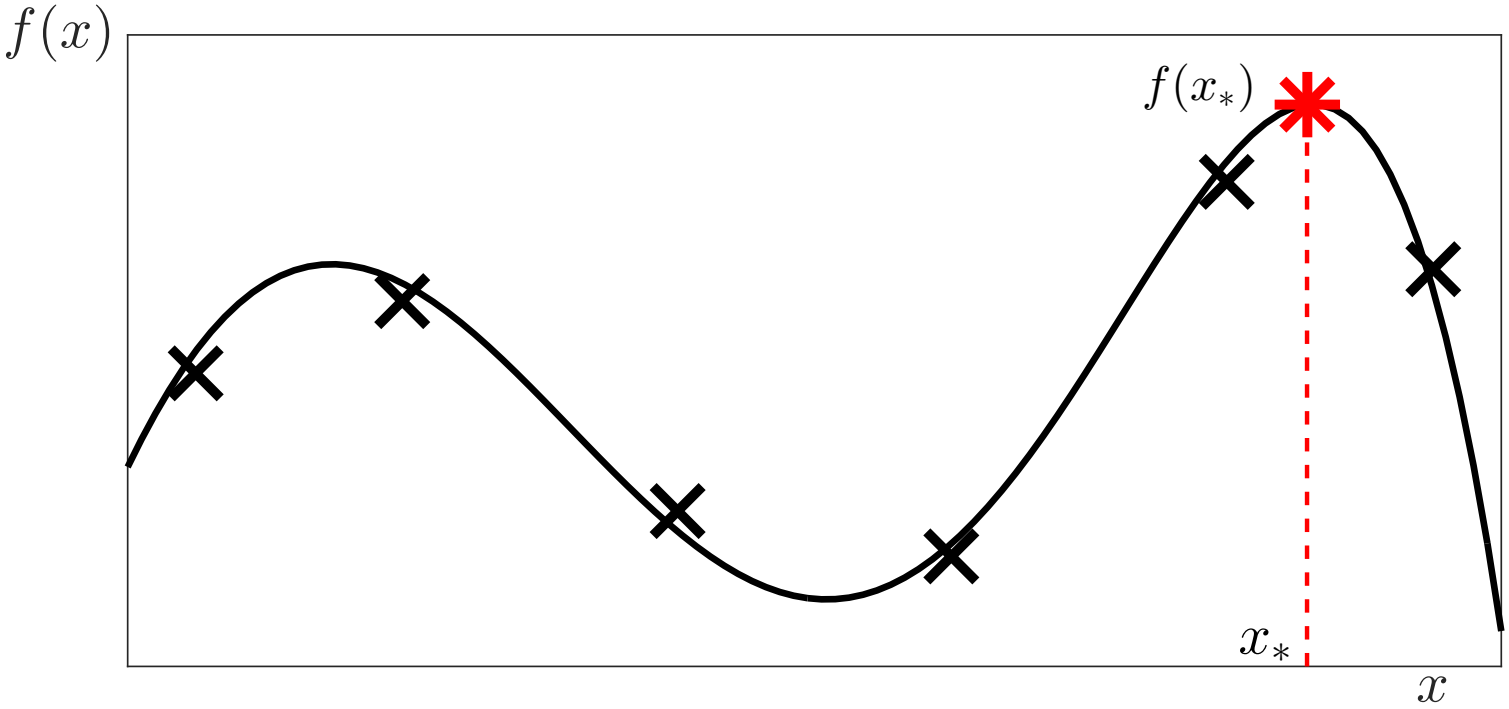


Simple Regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t).$$

Bandits/Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x\, f(x)$.

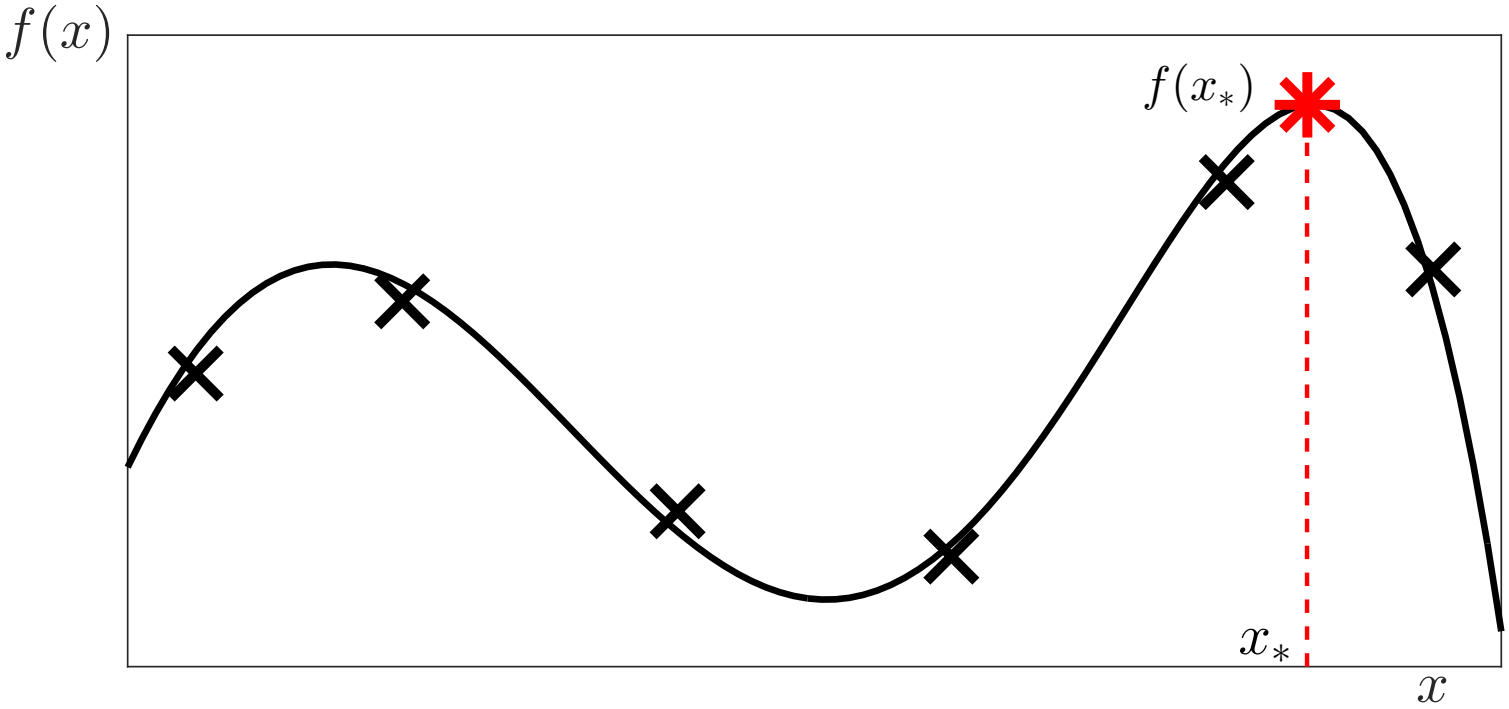


Cumulative Regret after n evaluations

$$\mathrm{CR}(n) = \sum_{t=1}^{n} \Big(f(x_\star) \; - f(x_t) \Big)$$

Bandits/Black-box Optimisation

$f : \mathcal{X} \to \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.
Let $x_\star = \mathrm{argmax}_x f(x)$.



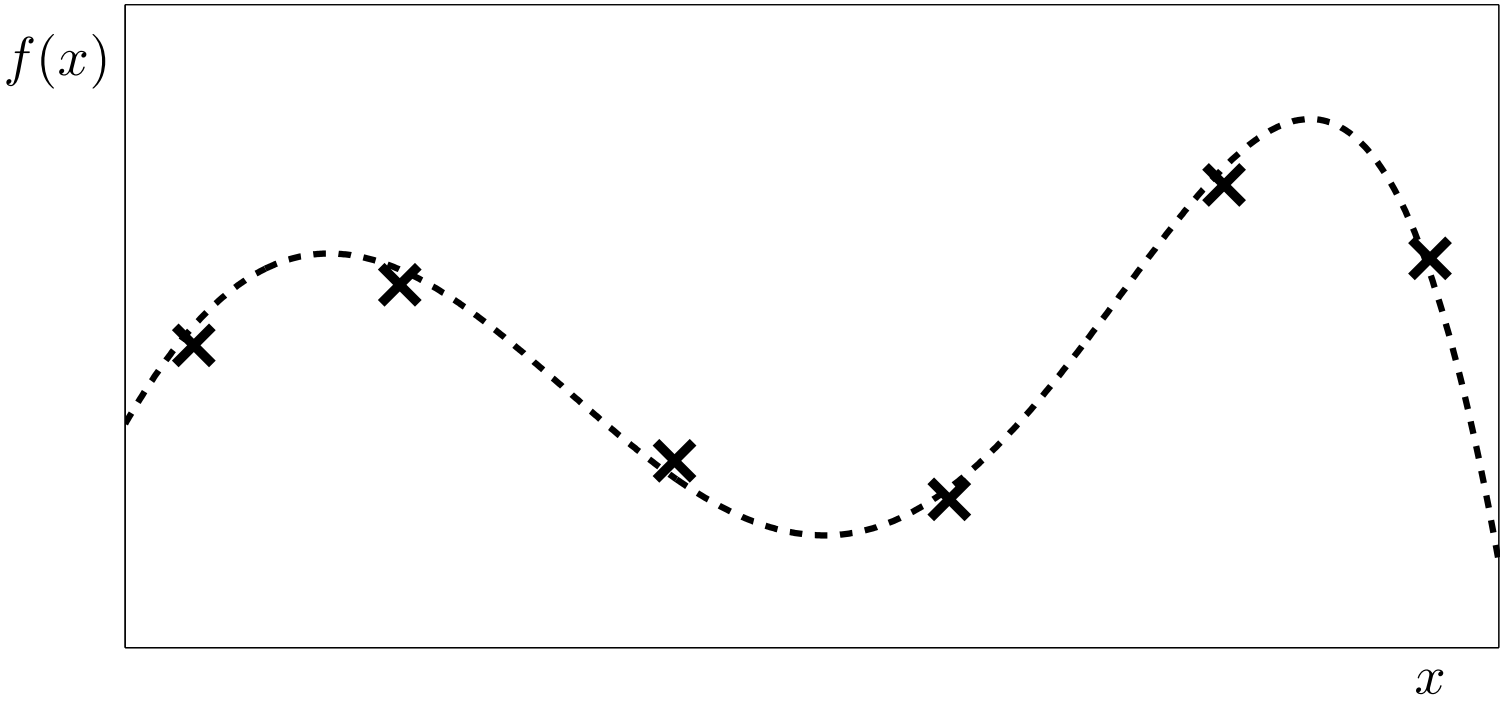
Simple Regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) - \max_{t=1,\ldots,n} f(x_t).$$

Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

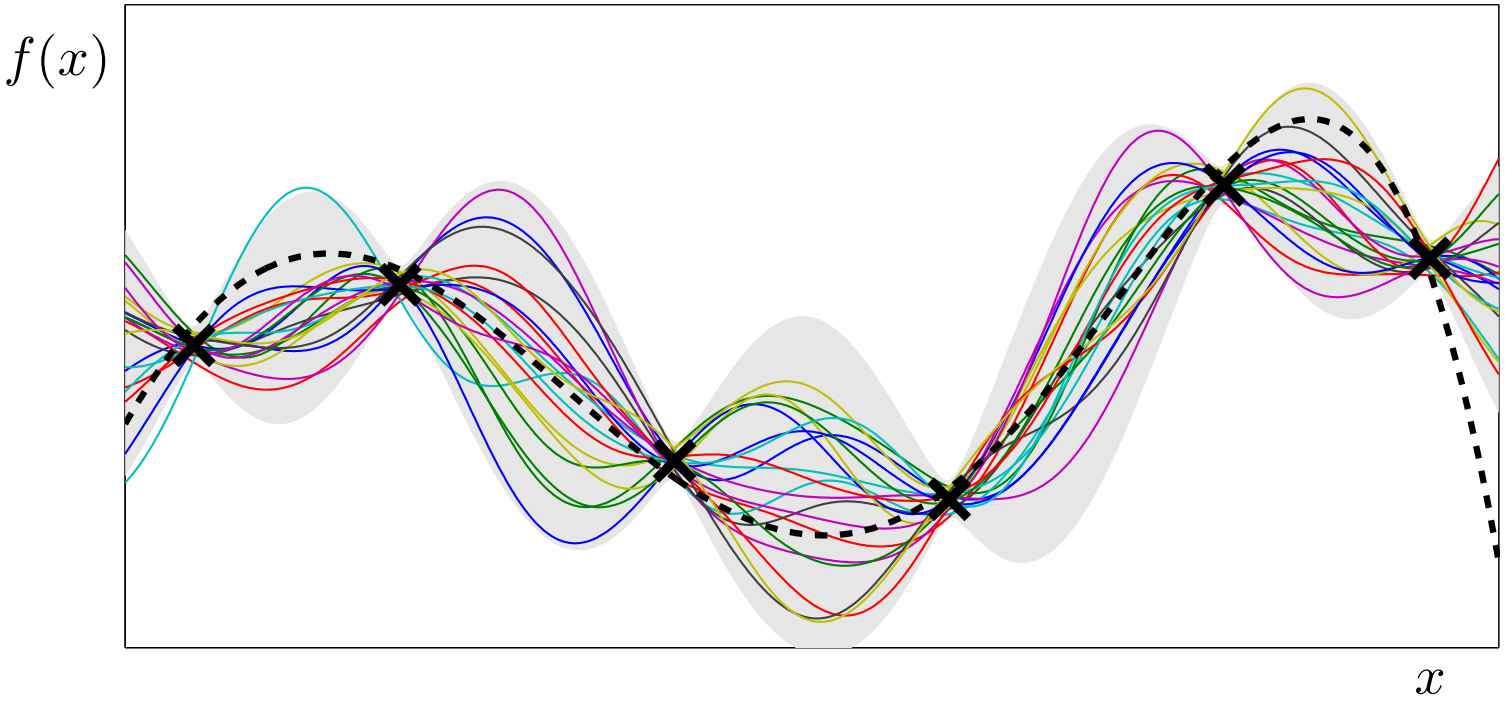
Observations



Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

Posterior $\mathcal{GP}$ given observations

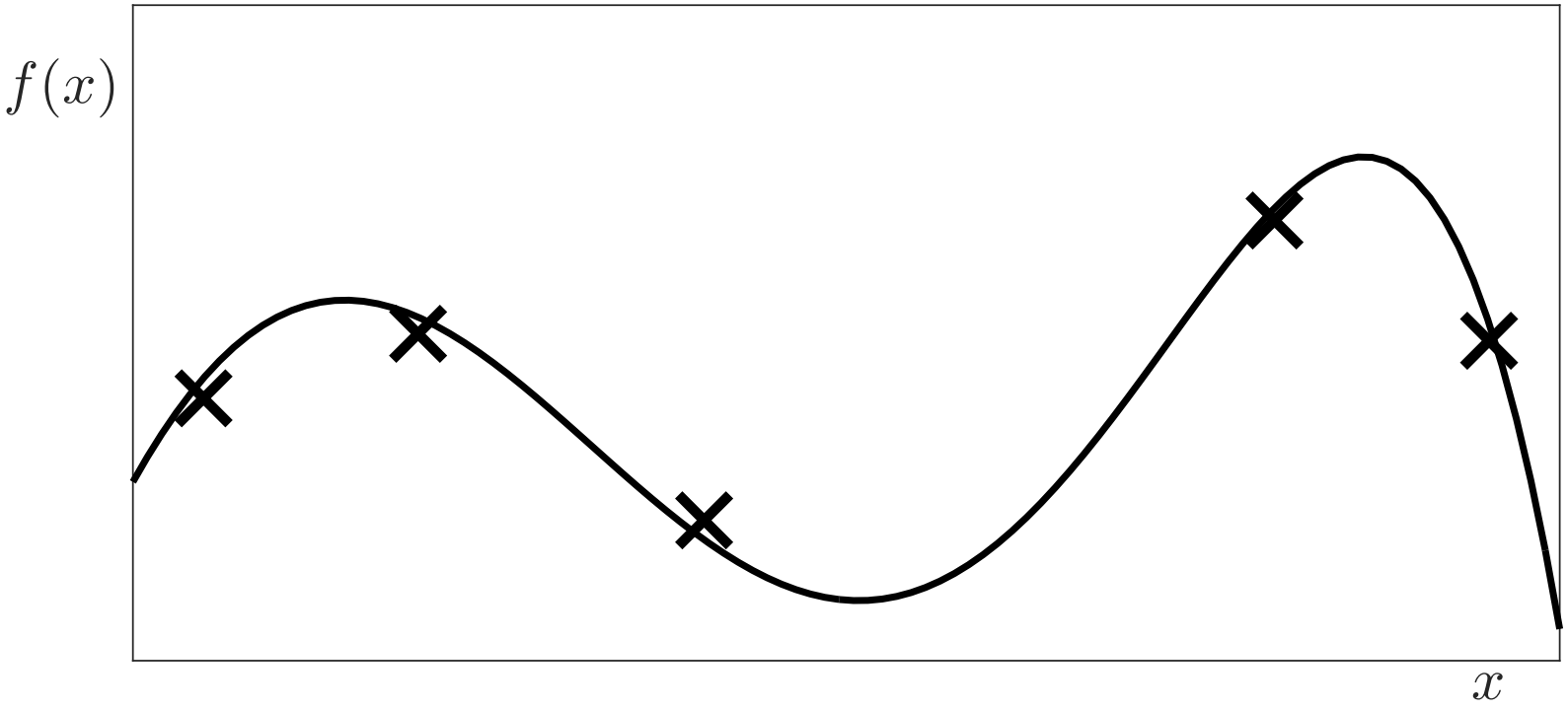


Algorithm 1: Upper Confidence Bounds

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Gaussian Process Upper Confidence Bound (GP-UCB)

(Auer el al. 2003, Srinivas et al. 2010)

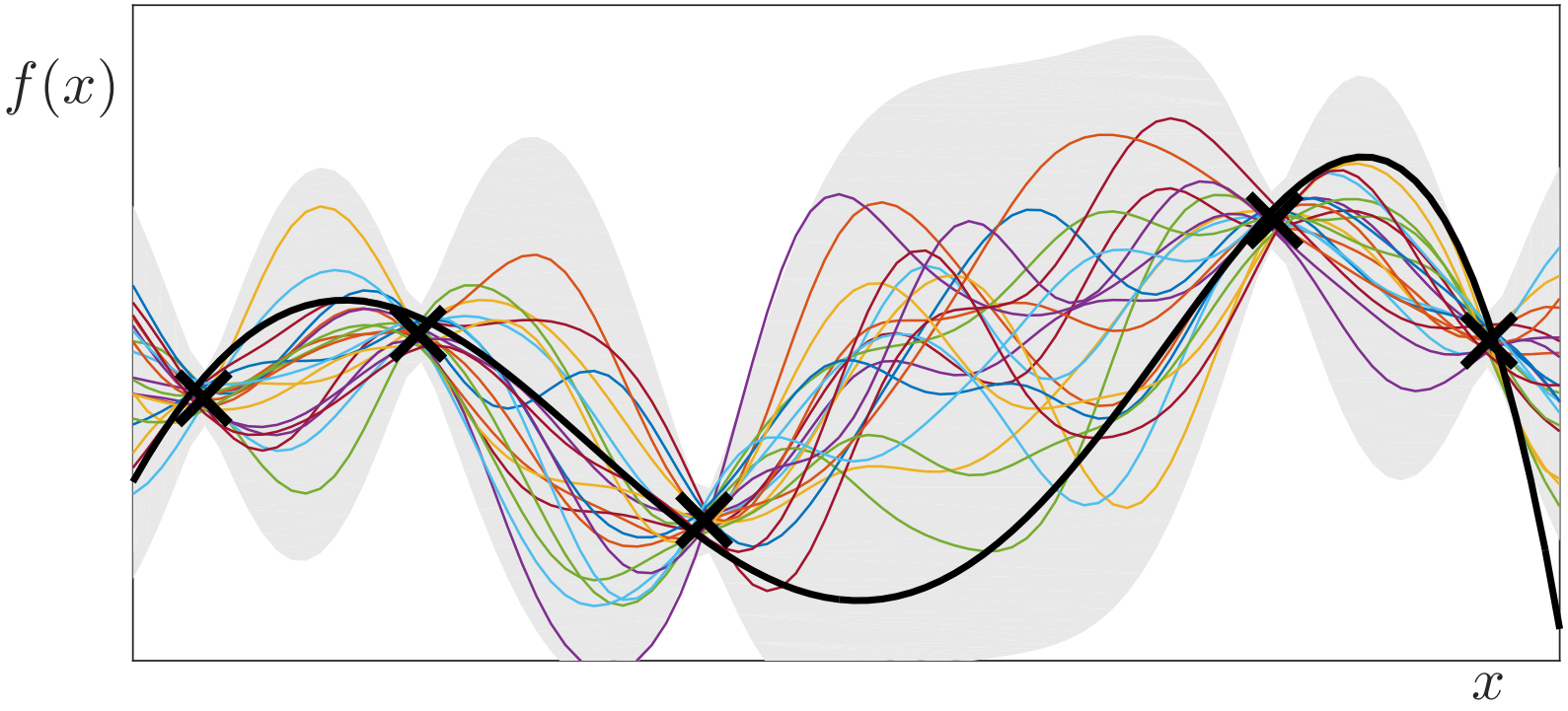


Algorithm 1: Upper Confidence Bounds

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Gaussian Process Upper Confidence Bound (GP-UCB)

(Auer el al. 2003, Srinivas et al. 2010)



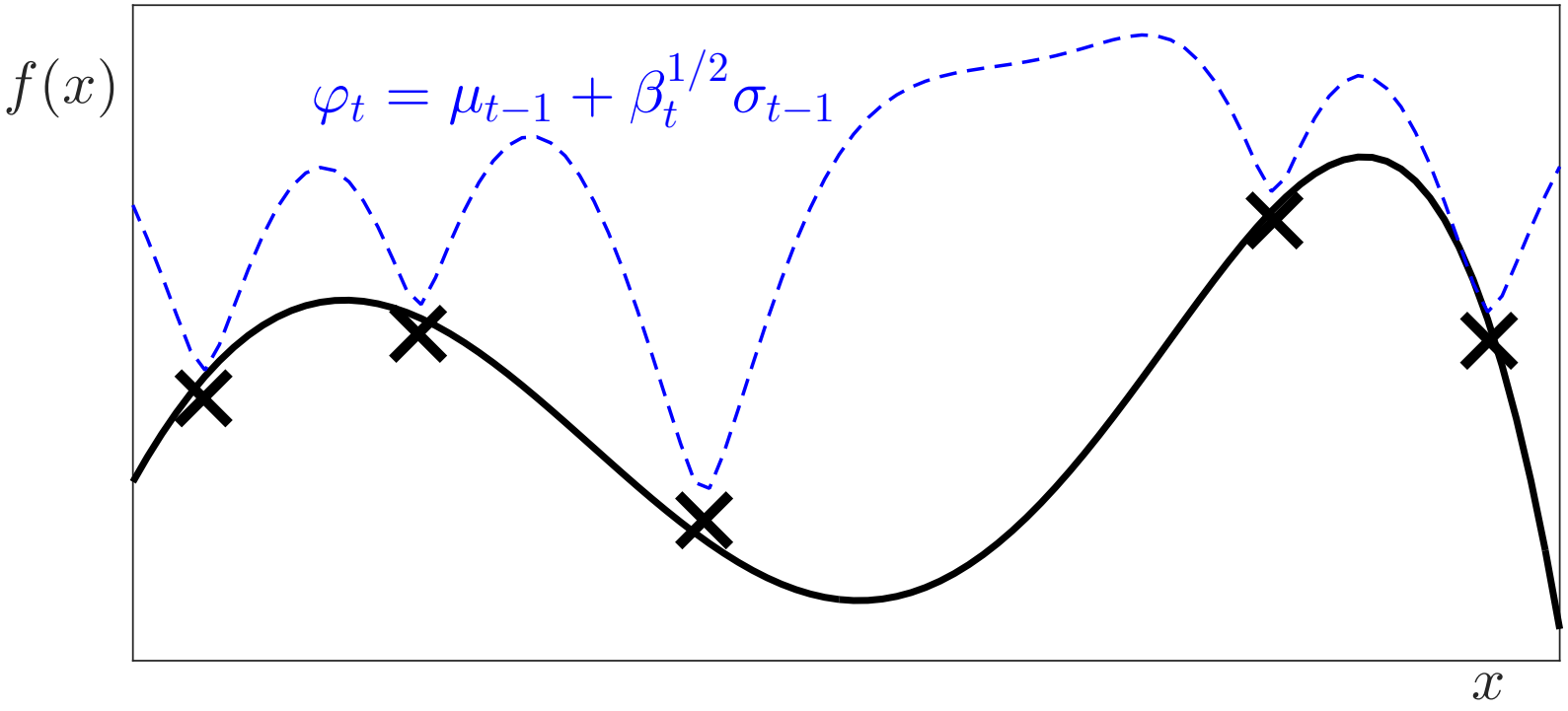
1) Compute posterior $\mathcal{GP}$.

Algorithm 1: Upper Confidence Bounds

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Gaussian Process Upper Confidence Bound (GP-UCB)

(Auer el al. 2003, Srinivas et al. 2010)



1) Compute posterior $\mathcal{GP}$.

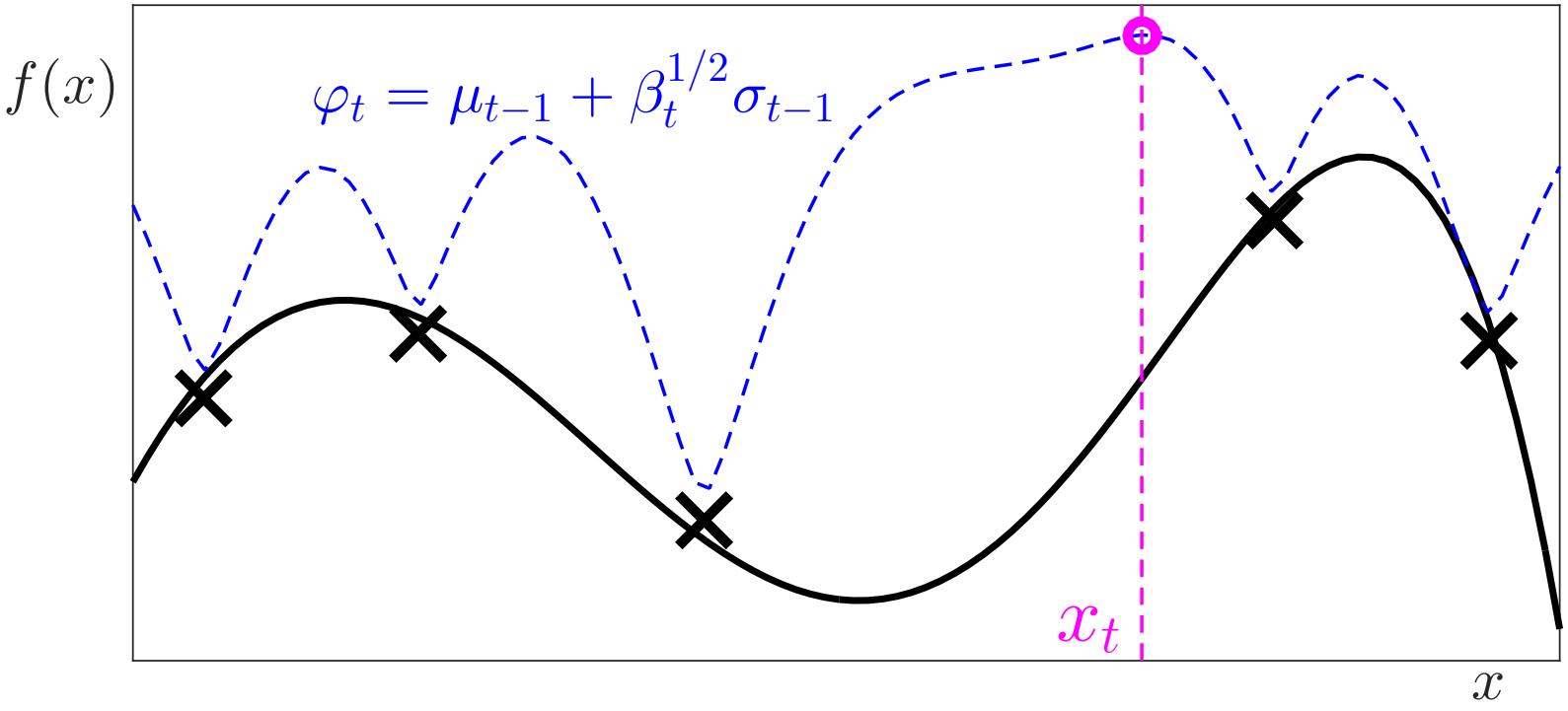
2) Construct UCB φ_t.

Algorithm 1: Upper Confidence Bounds

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Gaussian Process Upper Confidence Bound (GP-UCB)

(Auer el al. 2003, Srinivas et al. 2010)



1) Compute posterior $\mathcal{GP}$.

2) Construct UCB φ_t.

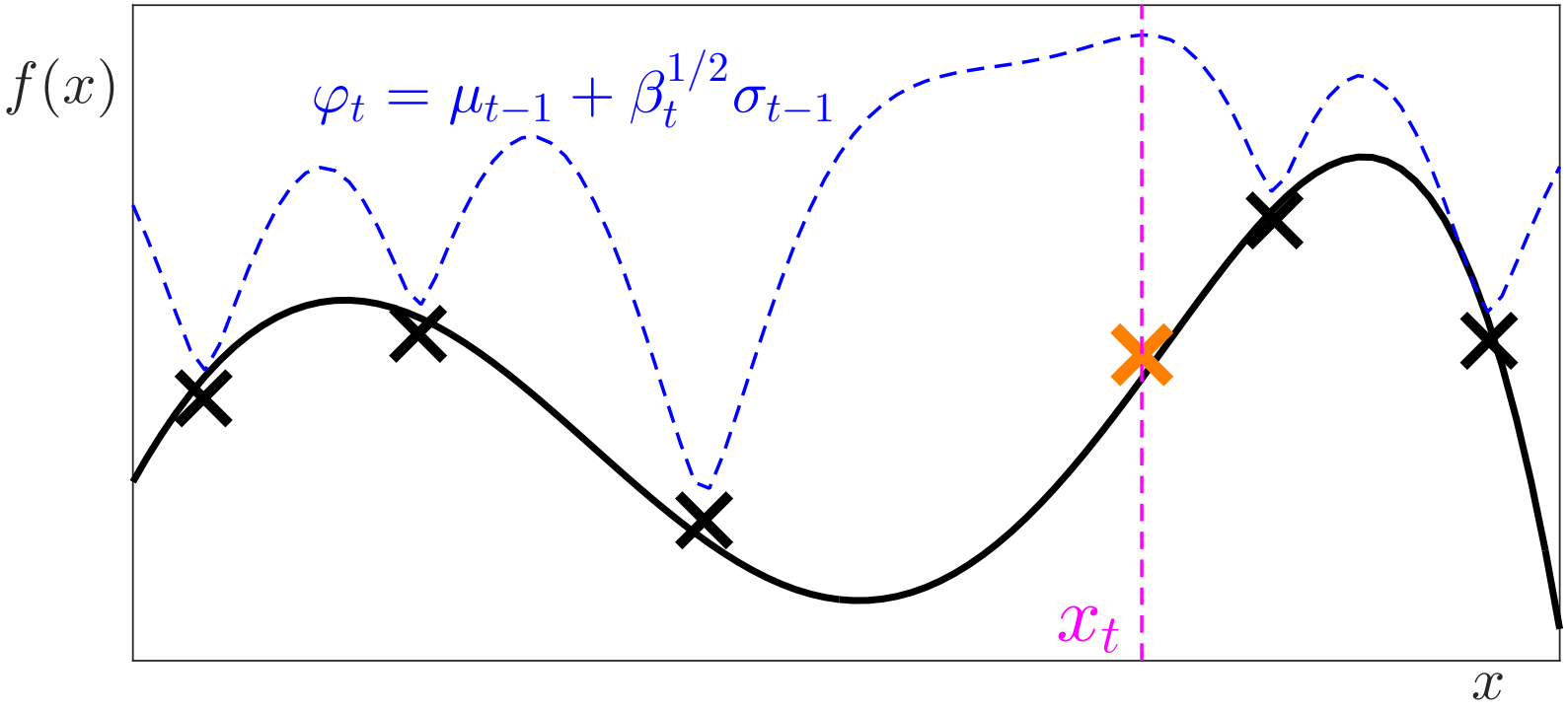
3) Choose $x_t = \operatorname{argmax}_x \varphi_t(x)$.

Algorithm 1: Upper Confidence Bounds

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$.

Gaussian Process Upper Confidence Bound (GP-UCB)

(Auer el al. 2003, Srinivas et al. 2010)



1) Compute posterior $\mathcal{GP}$.

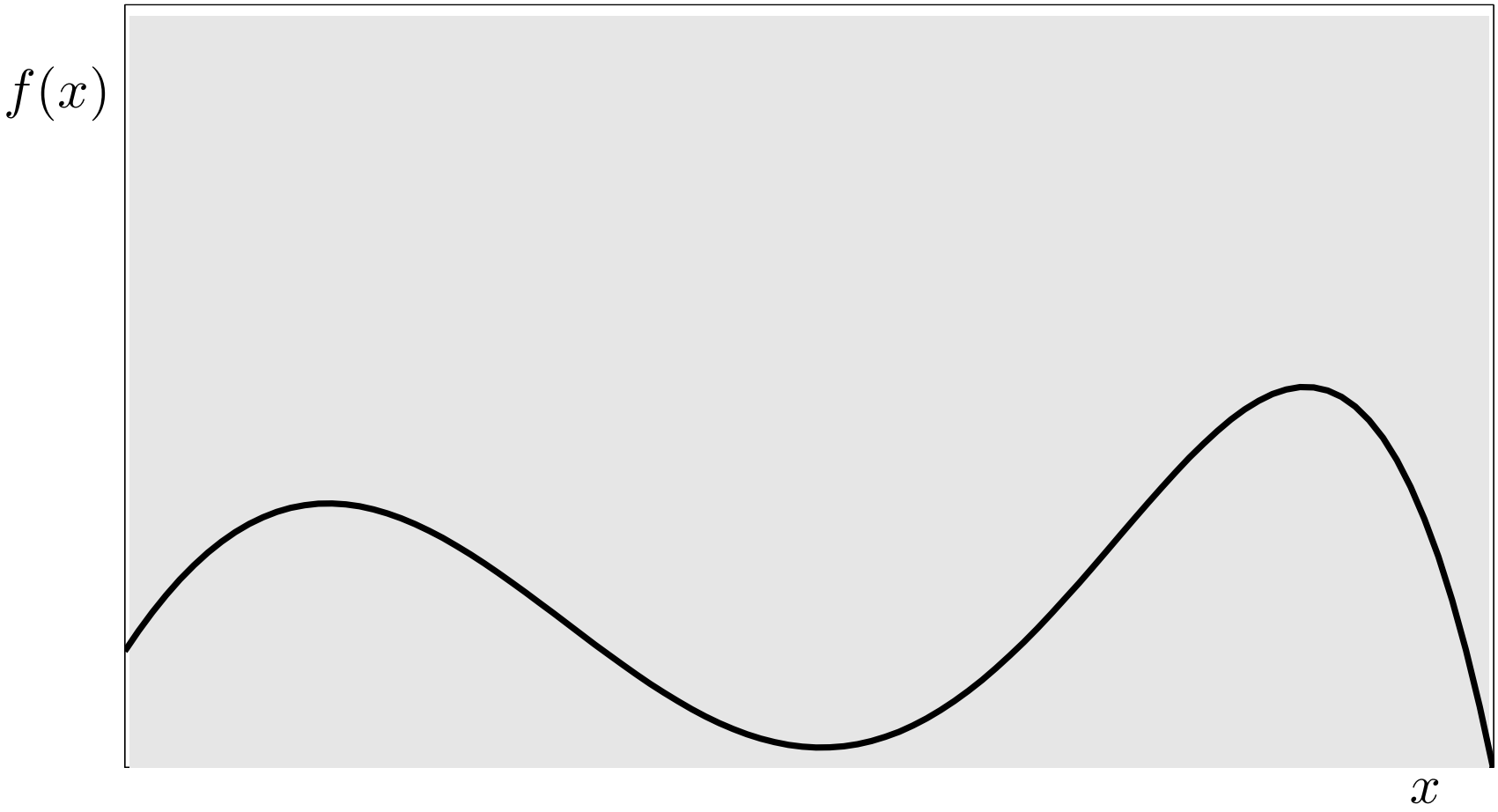
2) Construct UCB φ_t.

3) Choose $x_t = \operatorname{argmax}_x \varphi_t(x)$.

4) Evaluate f at x_t.

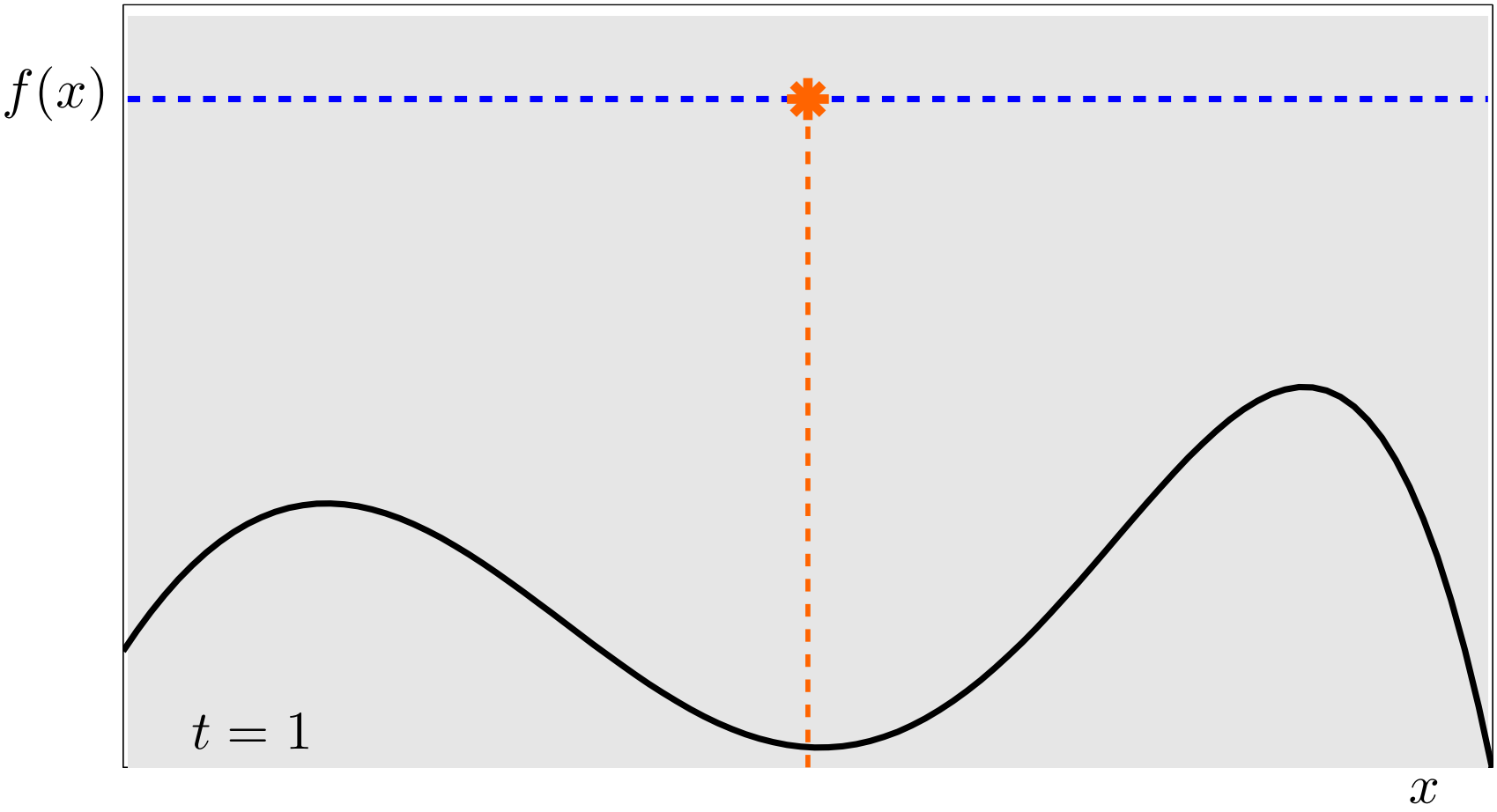
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



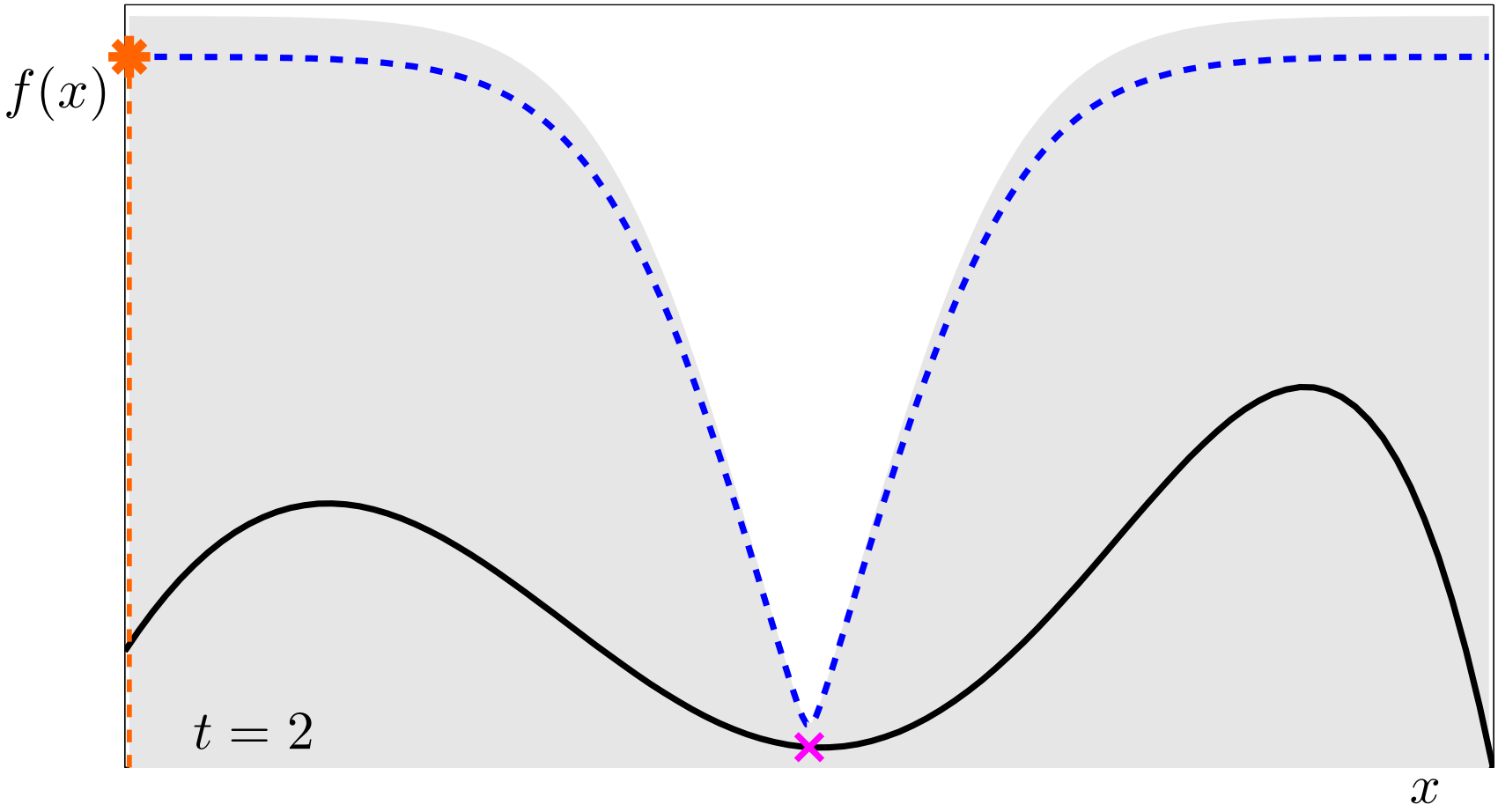
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



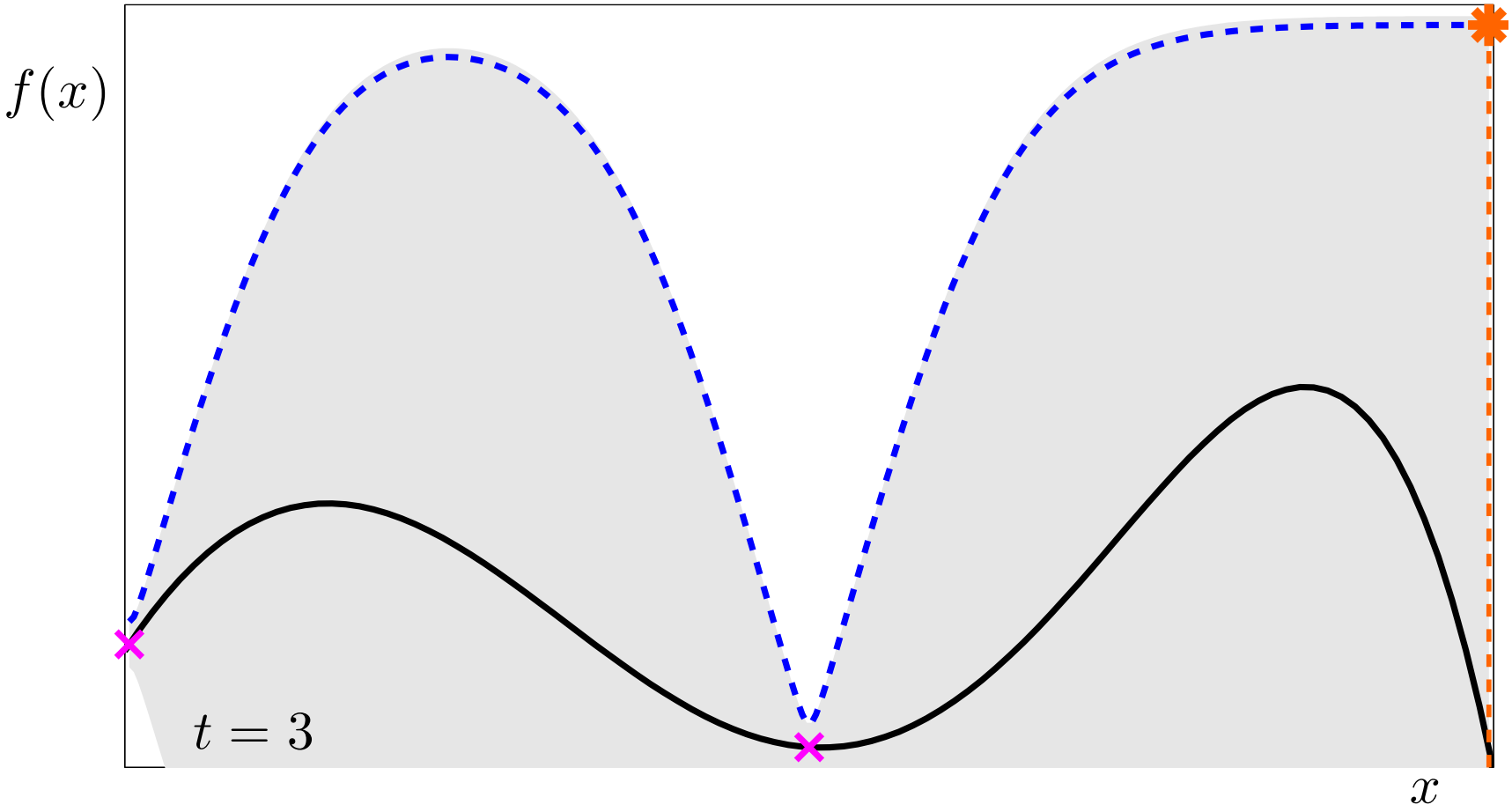
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



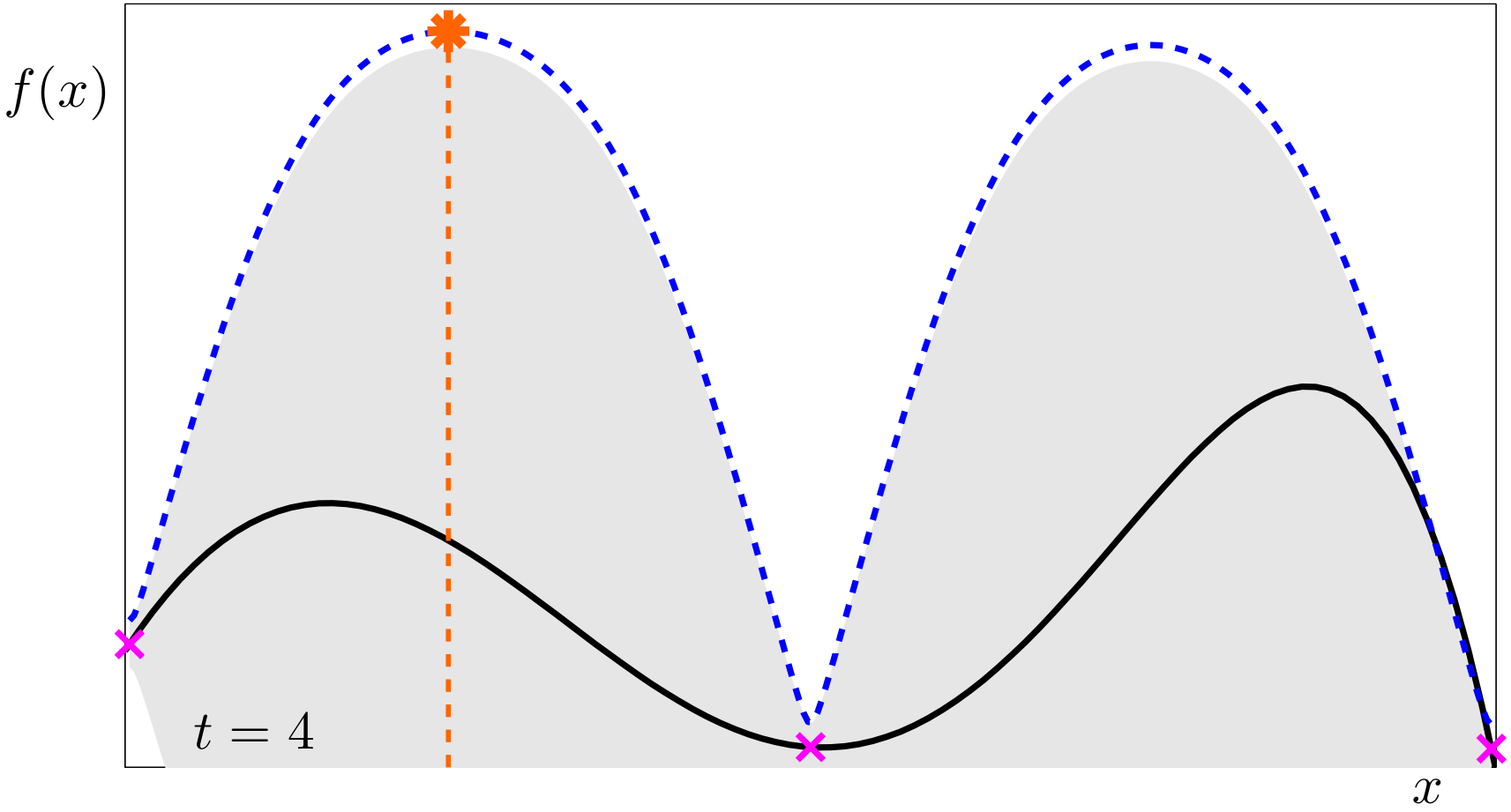
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



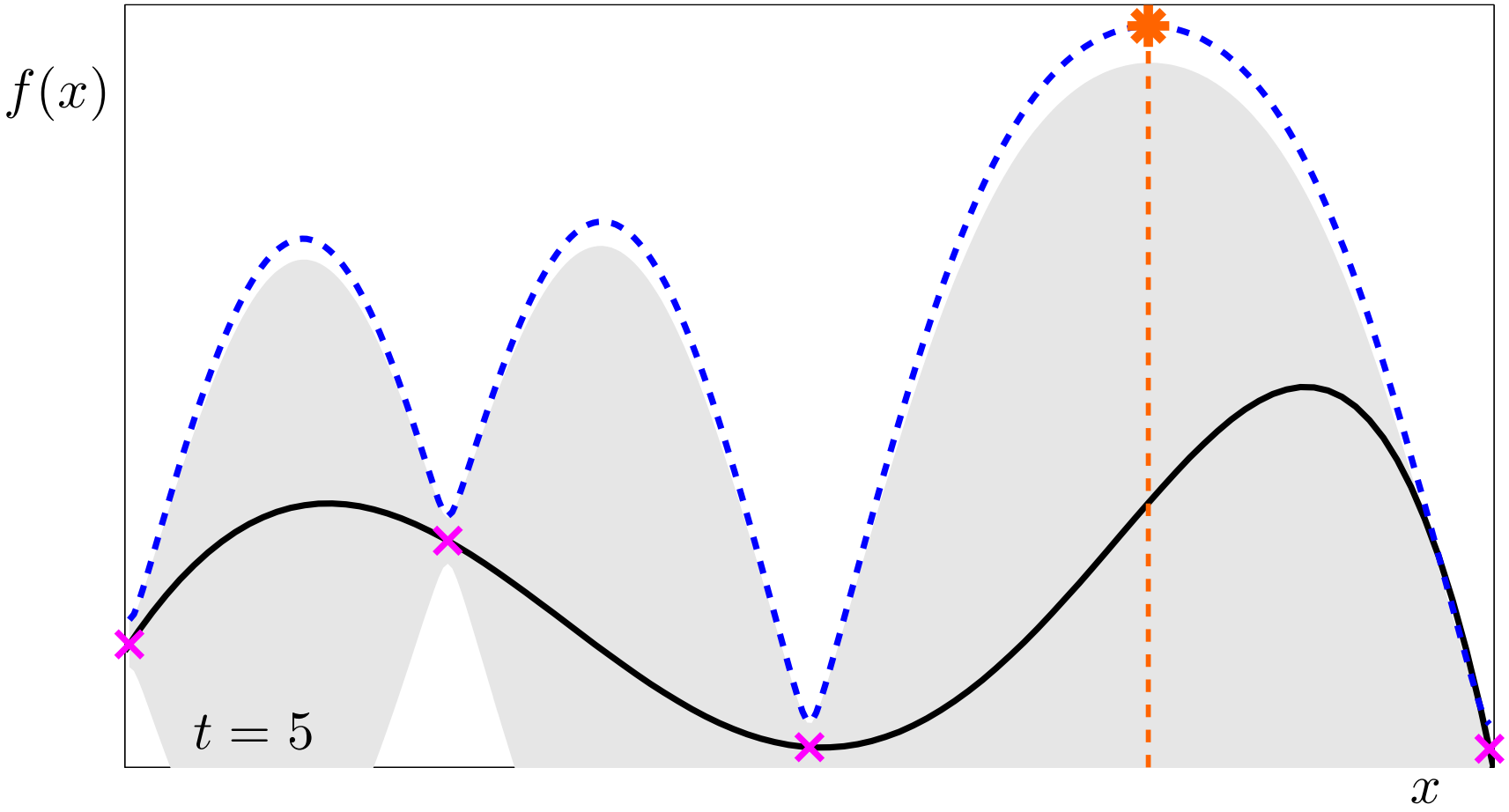
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



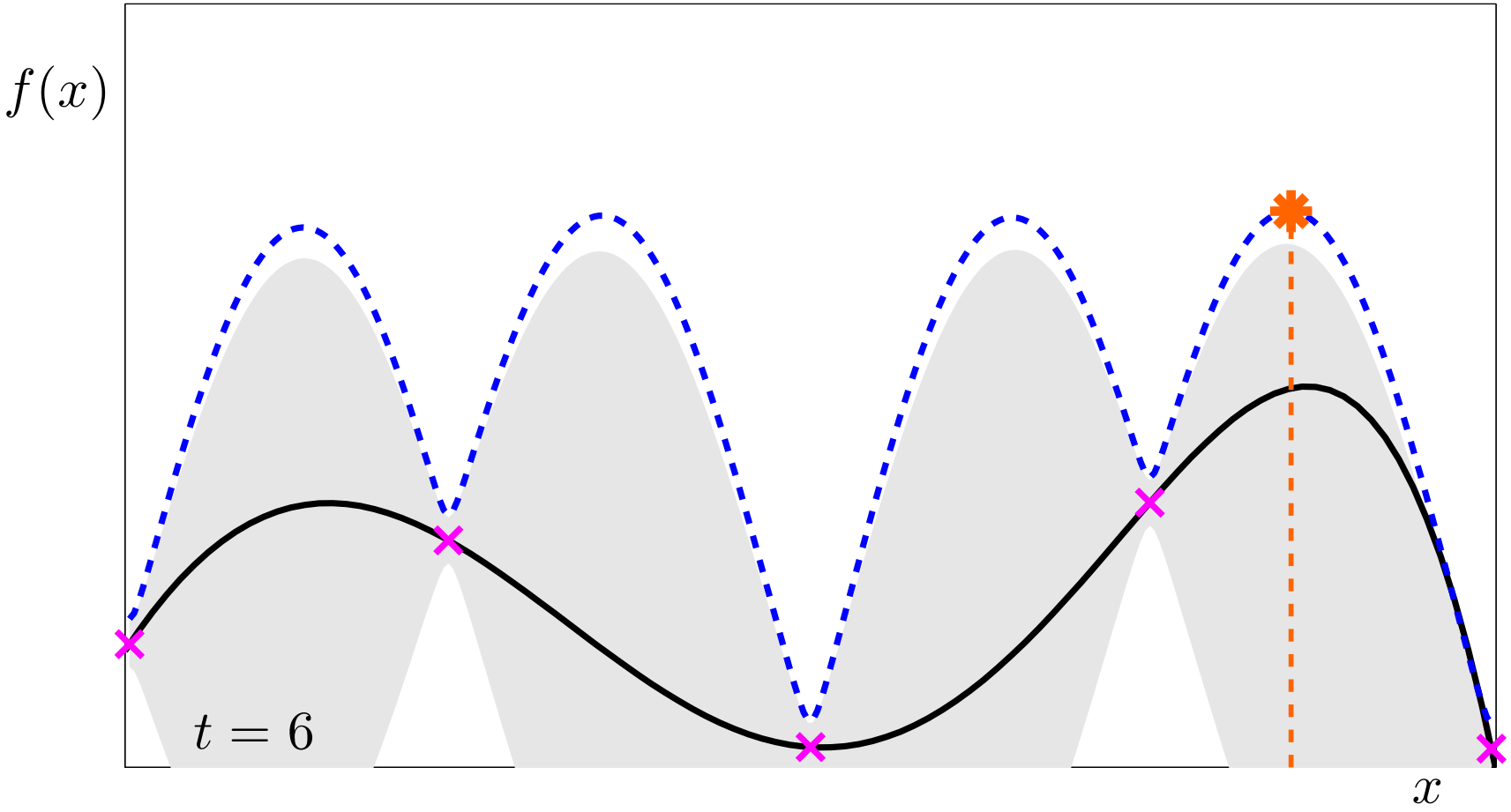
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



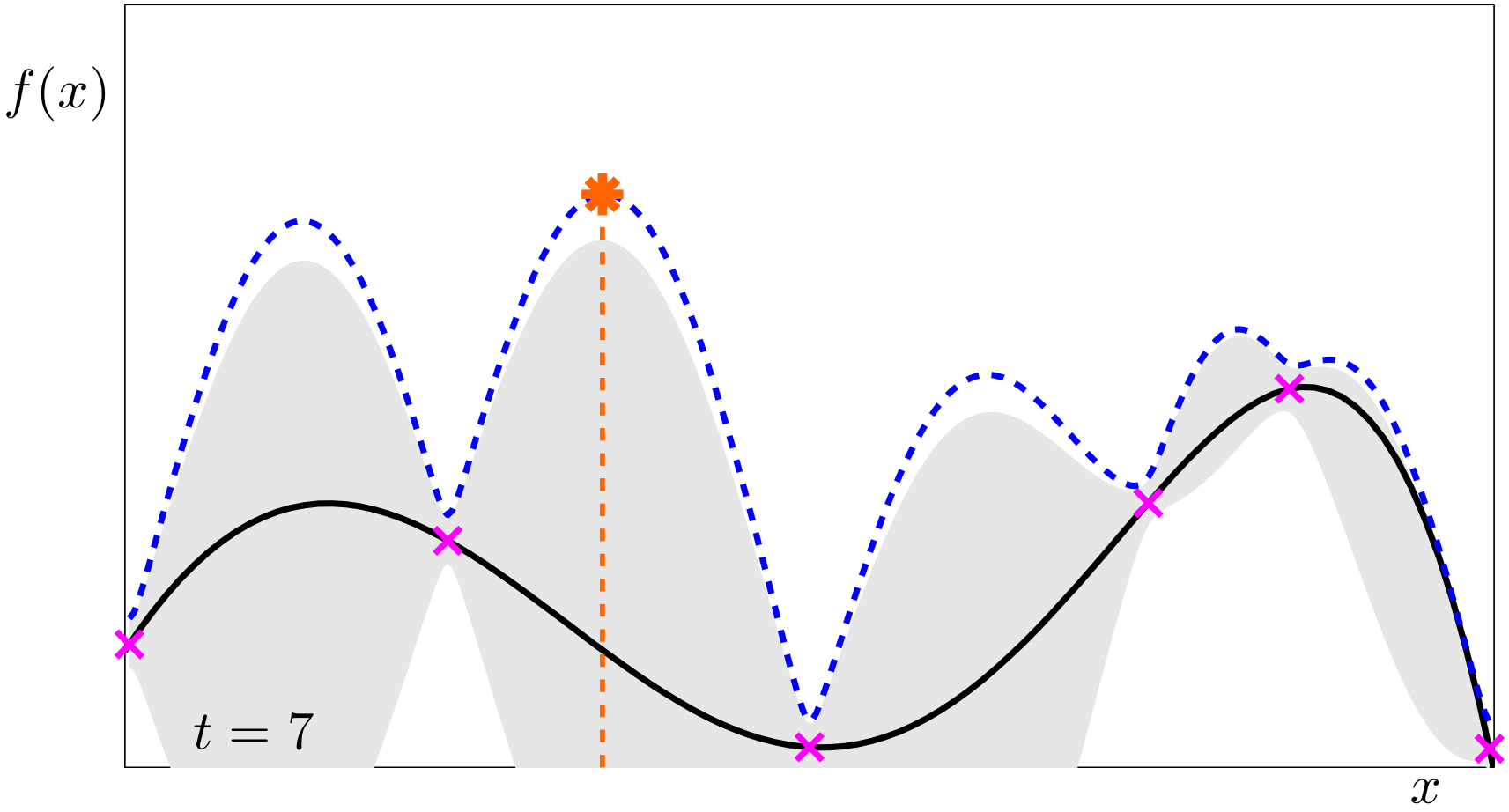
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



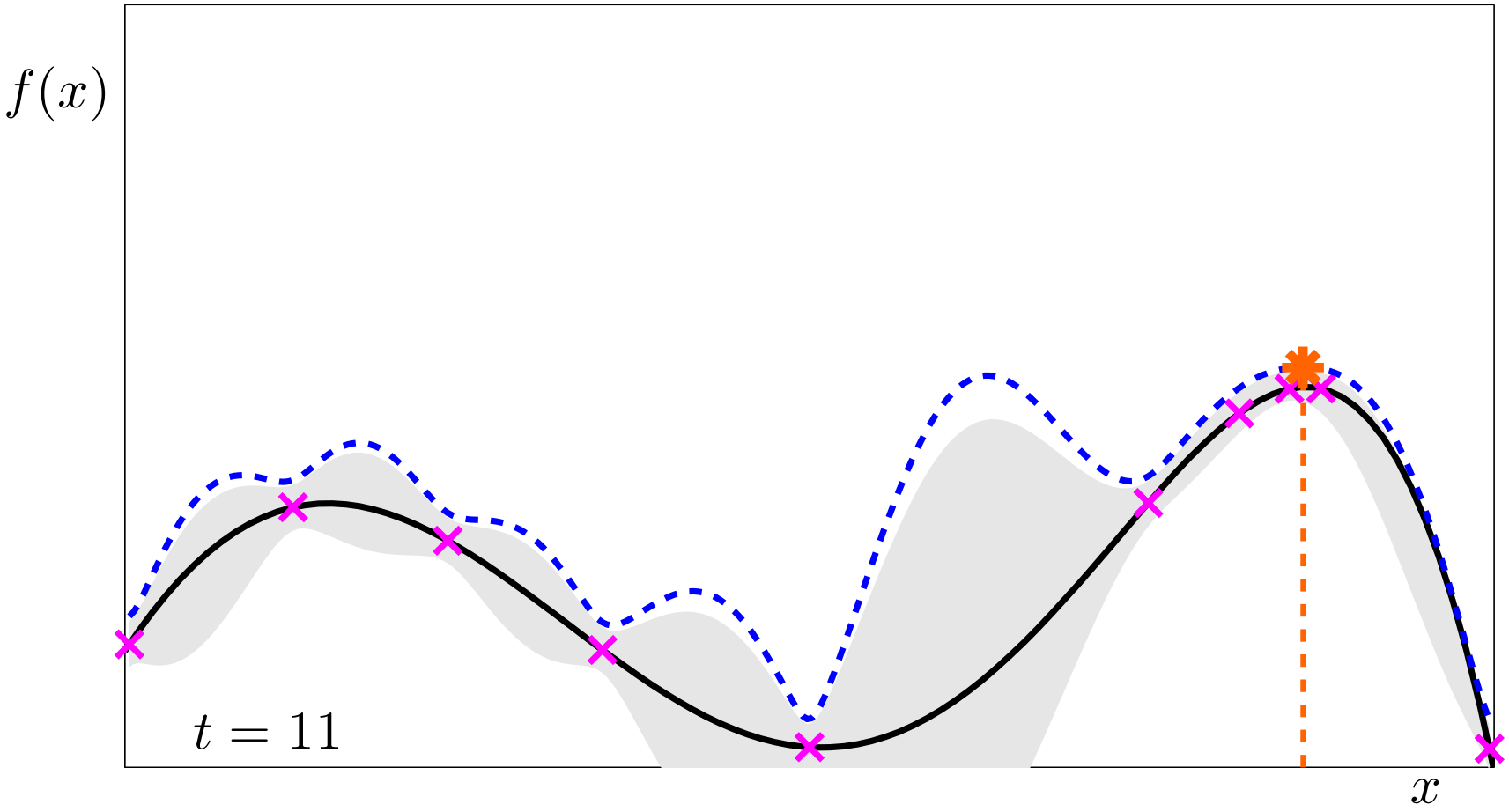
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



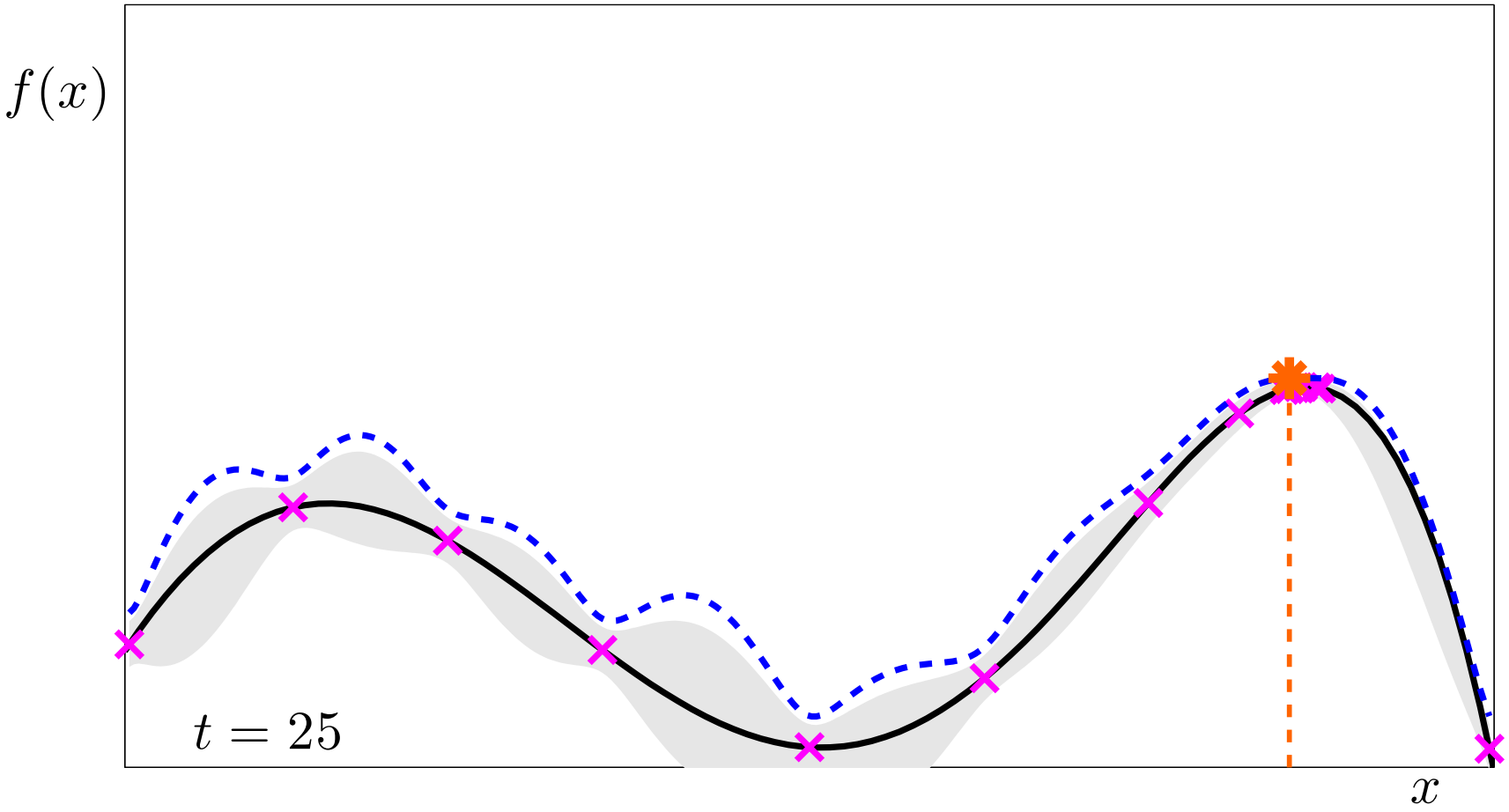
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



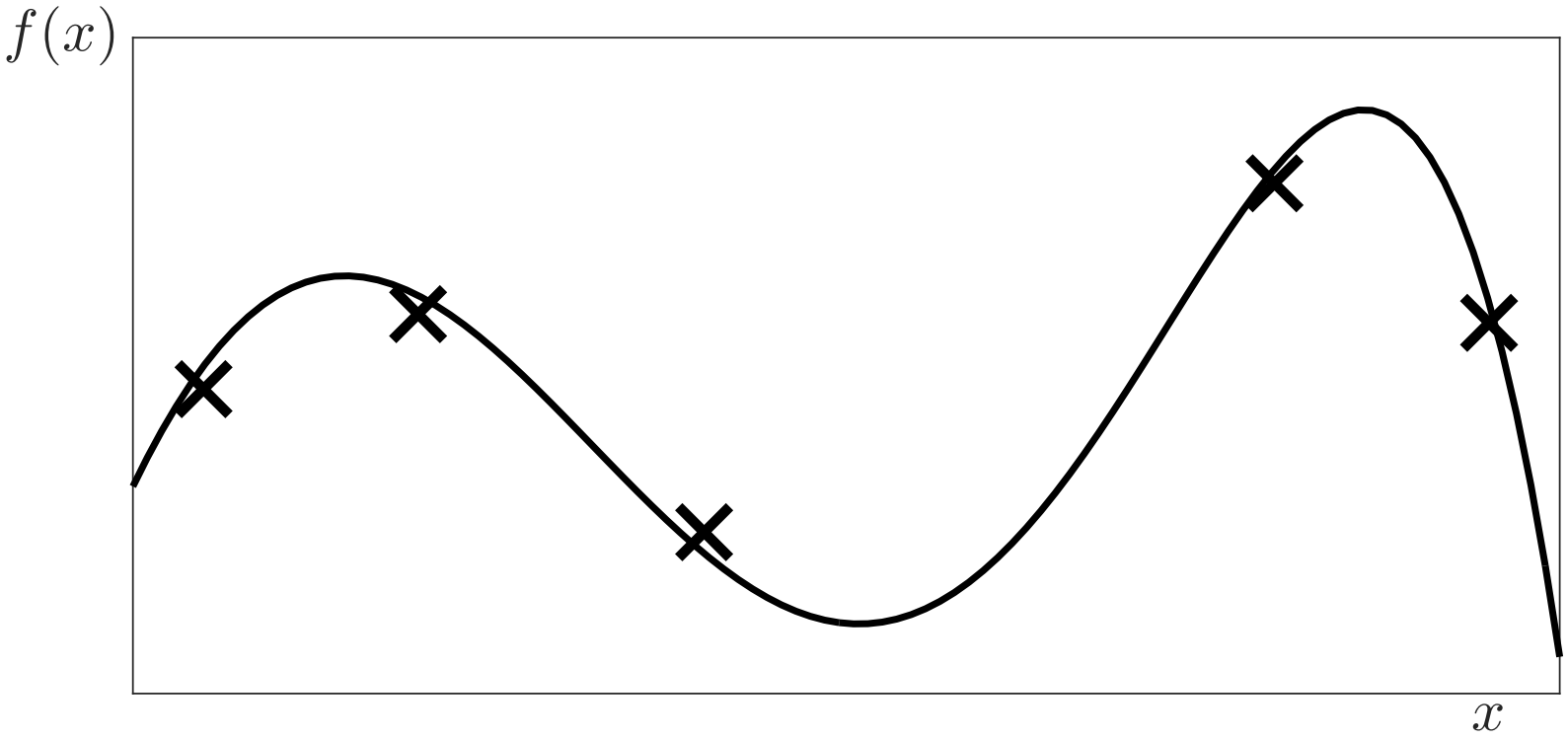
GP-UCB

(Auer el al. 2003, Srinivas et al. 2010)



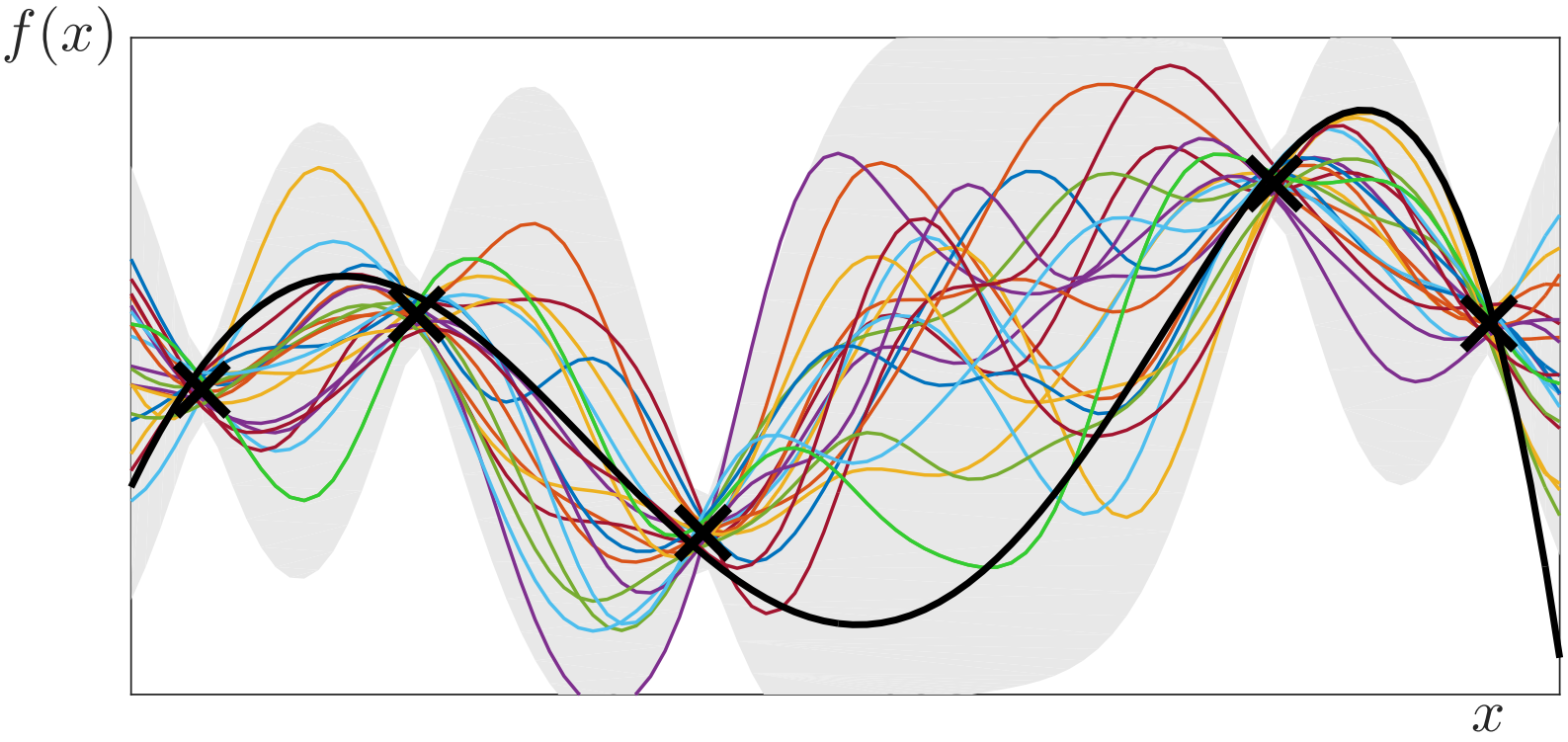
Algorithm 2: Thompson (Posterior) Sampling

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$. (Thompson, 1933)



Algorithm 2: Thompson (Posterior) Sampling

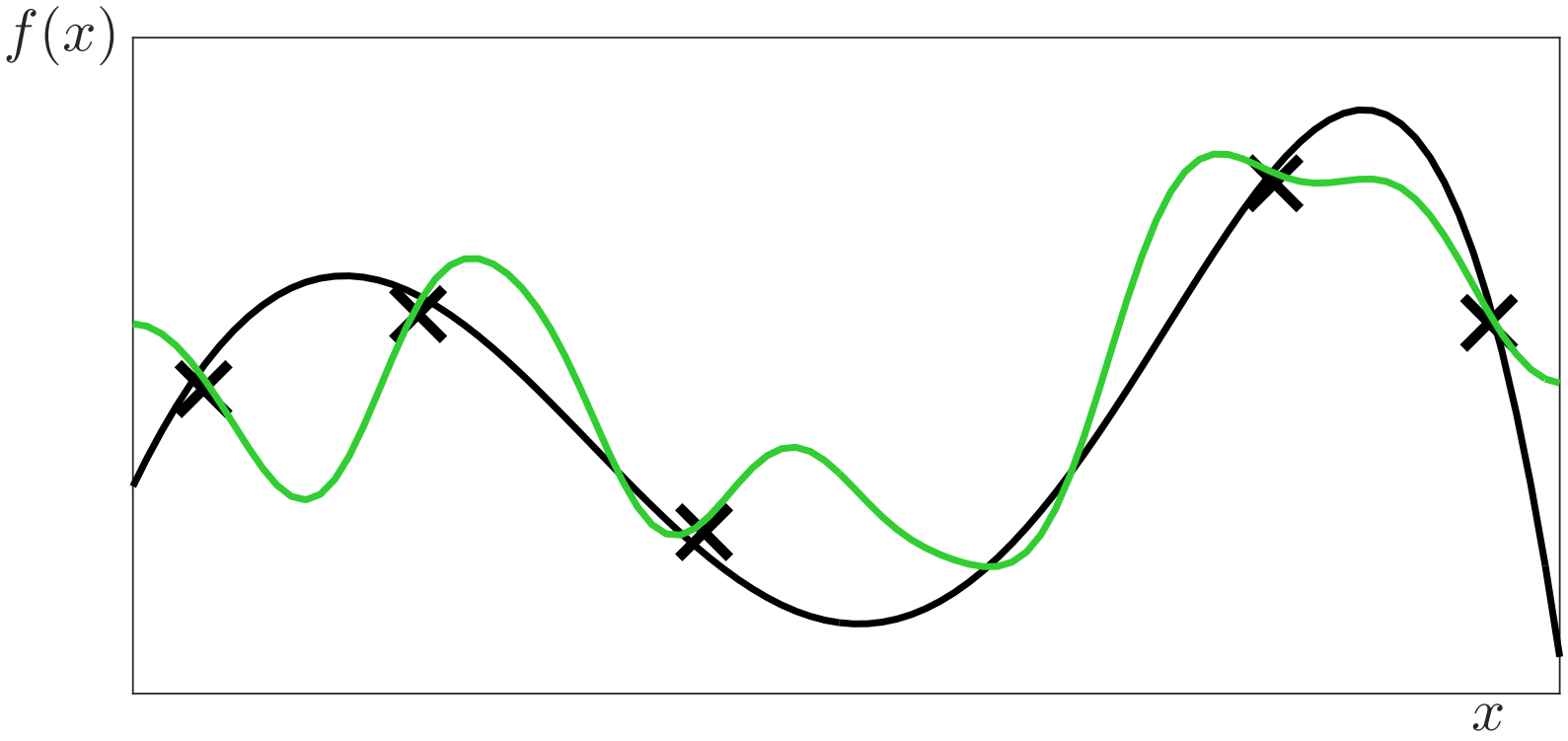
Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$. (Thompson, 1933)



1) Construct posterior $\mathcal{GP}$.

Algorithm 2: Thompson (Posterior) Sampling

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$. (Thompson, 1933)

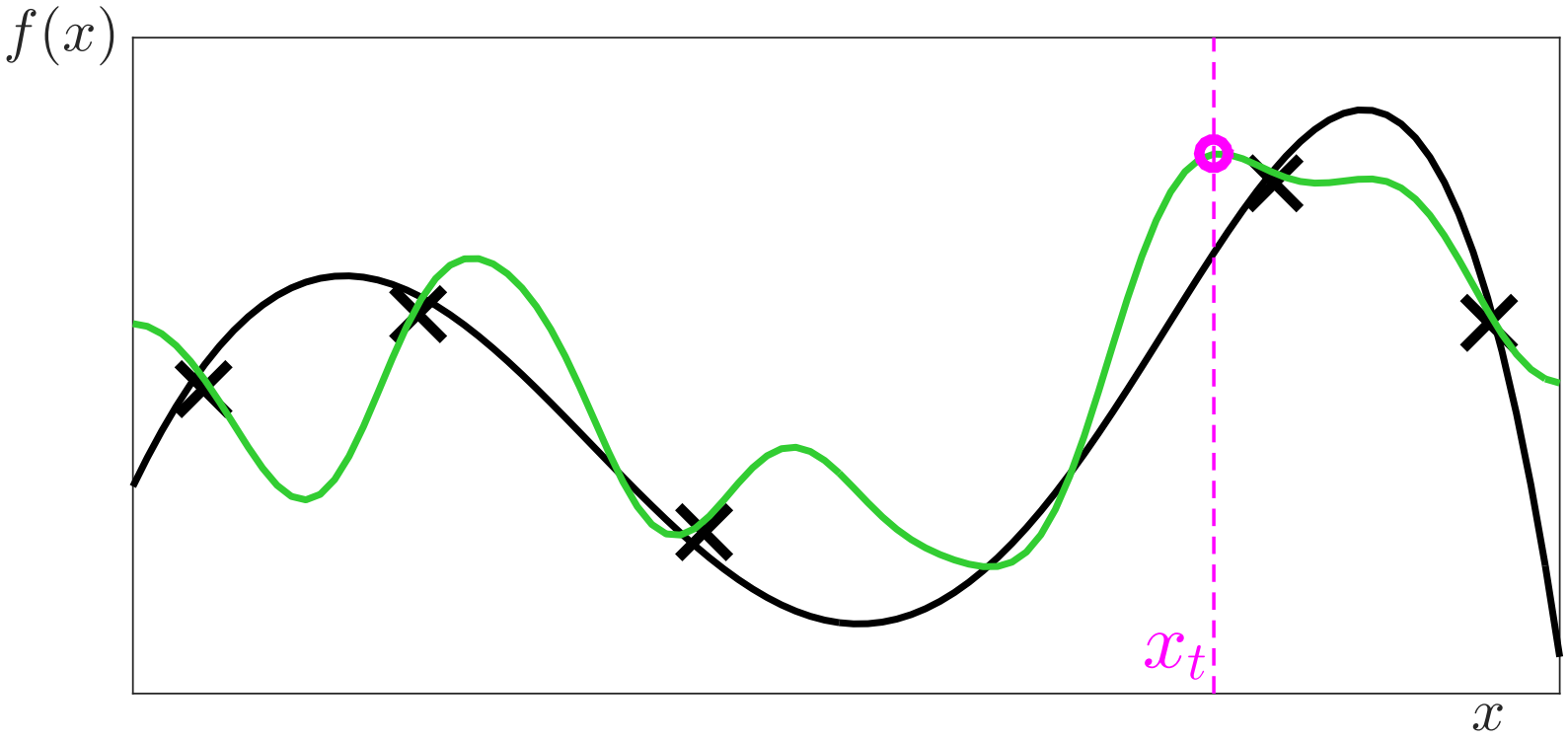


1) Construct posterior $\mathcal{GP}$.

2) Draw sample g from posterior.

Algorithm 2: Thompson (Posterior) Sampling

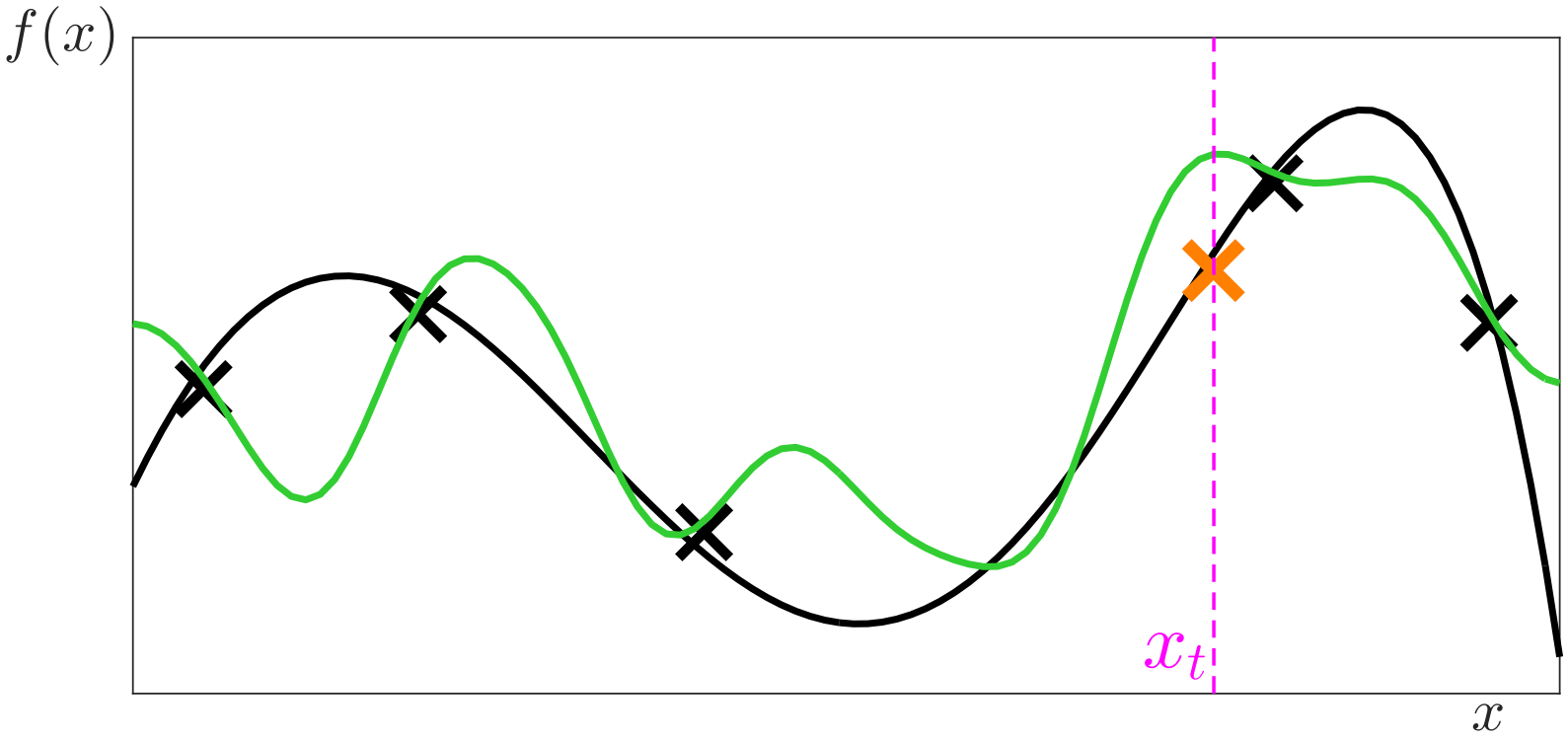
Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$. (Thompson, 1933)



1) Construct posterior $\mathcal{GP}$.
2) Draw sample g from posterior.
3) Choose $x_t = \operatorname{argmax}_x g(x)$.

Algorithm 2: Thompson (Posterior) Sampling

Model $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$. (Thompson, 1933)



1) Construct posterior $\mathcal{GP}$.
2) Draw sample g from posterior.
3) Choose $x_t = \operatorname{argmax}_x g(x)$.
4) Evaluate f at x_t.

Bayesian Optimisation

Other criteria for selecting x_t:

- Expected improvement (Jones et al. 1998)
- Probability of improvement (Kushner et al. 1964)
- Entropy search (Hernández-Lobato et al. 2014, Wang et al. 2017)
- ...and a few more.

Bayesian Optimisation

Other criteria for selecting x_t:

- Expected improvement (Jones et al. 1998)
- Probability of improvement (Kushner et al. 1964)
- Entropy search (Hernández-Lobato et al. 2014, Wang et al. 2017)
- ...and a few more.

Other Bayesian models for f:

- Neural networks (Snoek et al. 2015)
- Random Forests (Hutter 2009)
- Customised models

Outline

- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

Multi-fidelity Bandits

Motivating question:
What if we have cheap approximations to f?

1. Hyperparameter tuning: Train & validate with a subset of the data, and/or early stopping before convergence.

Multi-fidelity Bandits

Motivating question:
What if we have cheap approximations to f?

1. Hyperparameter tuning: Train & validate with a subset of the data, and/or early stopping before convergence.
2. Simulation based studies: perform simulations and numerical computations with less granularity.

Multi-fidelity Bandits

Motivating question:
What if we have cheap approximations to f?

1. Hyperparameter tuning: Train & validate with a subset of the data, and/or early stopping before convergence.
2. Simulation based studies: perform simulations and numerical computations with less granularity.

Example: Tune hyperparameters for an ML model with $N_\bullet$ data and $T_\bullet$ iterations.

Multi-fidelity Bandits

Motivating question:
What if we have cheap approximations to f?

1. Hyperparameter tuning: Train & validate with a subset of the data, and/or early stopping before convergence.
2. Simulation based studies: perform simulations and numerical computations with less granularity.

Example: Tune hyperparameters for an ML model with $N_\bullet$ data and $T_\bullet$ iterations.

- Can we use $N < N_\bullet$ data and $T < T_\bullet$ iterations to approximate cross validation performance at $(N_\bullet, T_\bullet)$?

Multi-fidelity Bandits

Motivating question:
What if we have cheap approximations to f?

1. Hyperparameter tuning: Train & validate with a subset of the data, and/or early stopping before convergence.
2. Simulation based studies: perform simulations and numerical computations with less granularity.

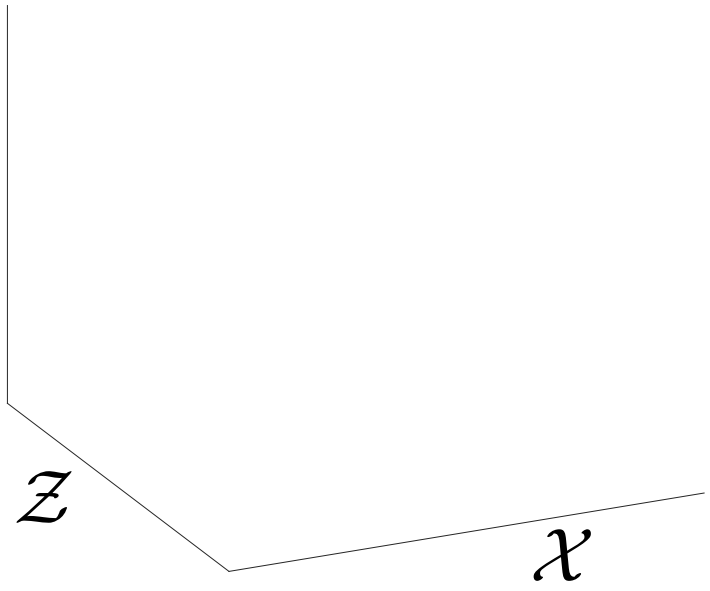
Example: Tune hyperparameters for an ML model with $N_\bullet$ data and $T_\bullet$ iterations.

- Can we use $N < N_\bullet$ data and $T < T_\bullet$ iterations to approximate cross validation performance at $(N_\bullet, T_\bullet)$?

Approximations from a *continuous* 2D "fidelity space" (N, T).

Multi-fidelity Bandits

(Kandasamy et al. ICML 2017)



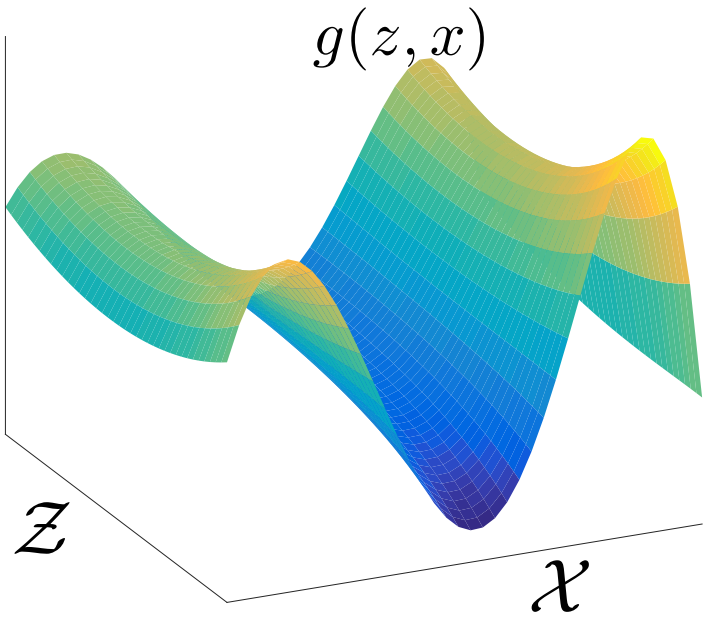
A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all (N, T) values.

$\mathcal{X} \leftarrow$ all hyperparameter values.

Multi-fidelity Bandits

(Kandasamy et al. ICML 2017)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all (N, T) values.

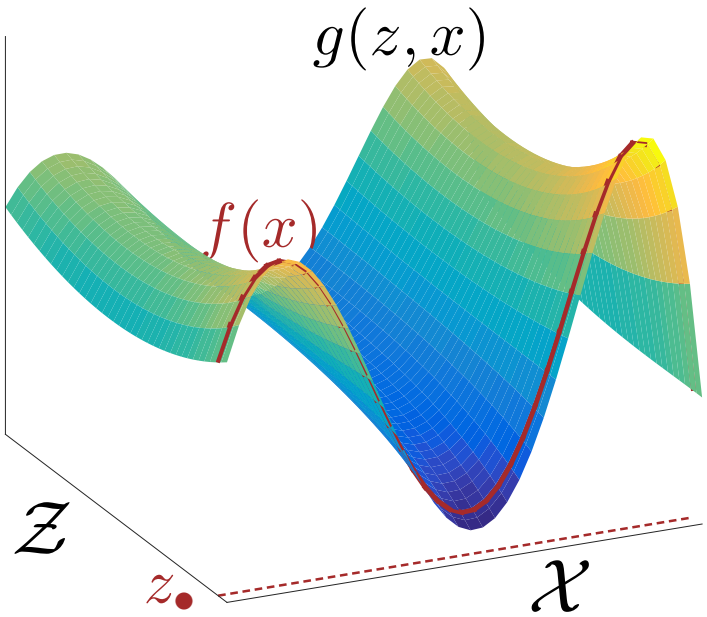
$\mathcal{X} \leftarrow$ all hyperparameter values.

$g : \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}$.

$g([N, T], x) \leftarrow$ cv accuracy when training with N data for T iterations at hyperparameter x.

Multi-fidelity Bandits

(Kandasamy et al. ICML 2017)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all (N, T) values.

$\mathcal{X} \leftarrow$ all hyperparameter values.

$g : \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}$.

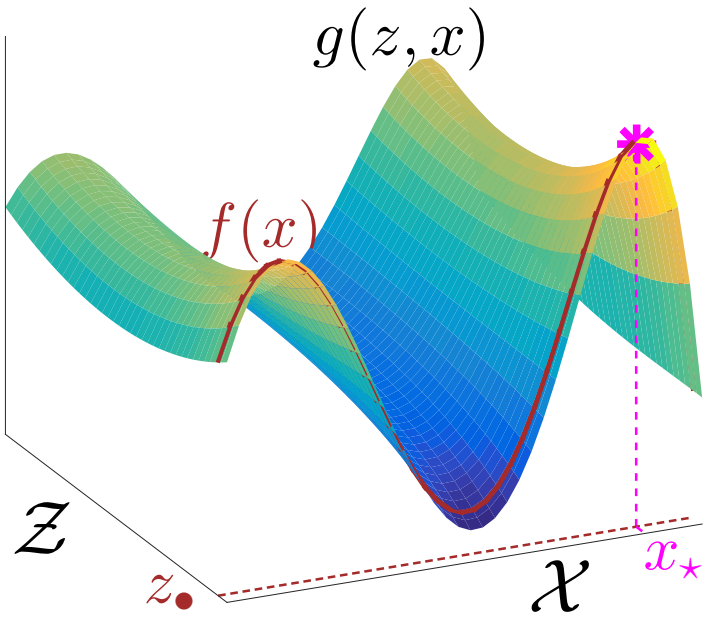
$g([N, T], x) \leftarrow$ cv accuracy when training with N data for T iterations at hyperparameter x.

Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$.

$z_\bullet = [N_\bullet, T_\bullet]$.

Multi-fidelity Bandits

(Kandasamy et al. ICML 2017)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all (N, T) values.

$\mathcal{X} \leftarrow$ all hyperparameter values.

$g : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$.

$g([N, T], x) \leftarrow$ cv accuracy when training with N data for T iterations at hyperparameter x.

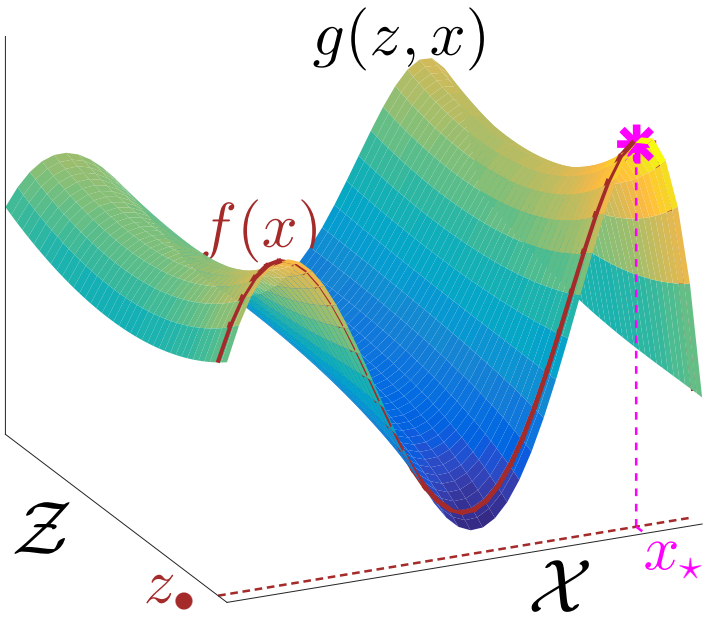
Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$.

$z_\bullet = [N_\bullet, T_\bullet]$.

End Goal: Find $x_\star = \mathrm{argmax}_x f(x)$.

Multi-fidelity Bandits

(Kandasamy et al. ICML 2017)



A fidelity space $\mathcal{Z}$ and domain $\mathcal{X}$

$\mathcal{Z} \leftarrow$ all (N, T) values.

$\mathcal{X} \leftarrow$ all hyperparameter values.

$g : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$.

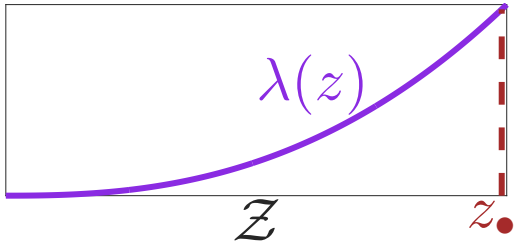
$g([N, T], x) \leftarrow$ cv accuracy when training with N data for T iterations at hyperparameter x.

Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$. $z_\bullet = [N_\bullet, T_\bullet]$.

End Goal: Find $x_\star = \mathrm{argmax}_x f(x)$.

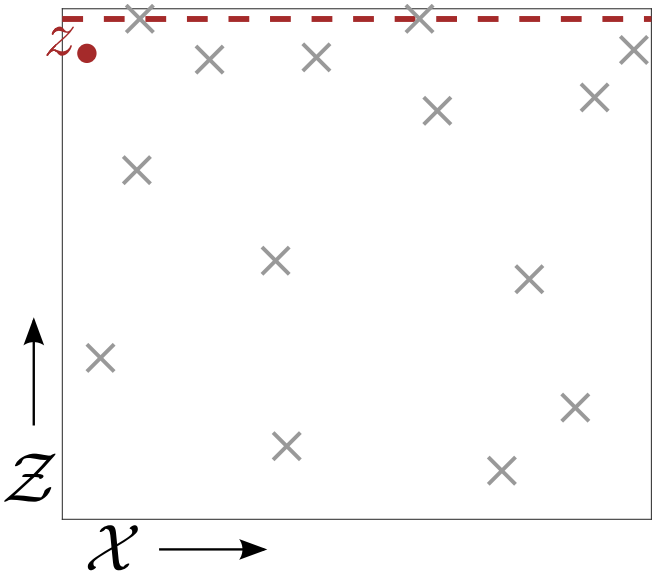
A cost function, $\lambda : \mathcal{Z} \to \mathbb{R}_+$.

$\lambda(z) = \lambda(N, T) = \mathcal{O}(N^2 T)$ (say).



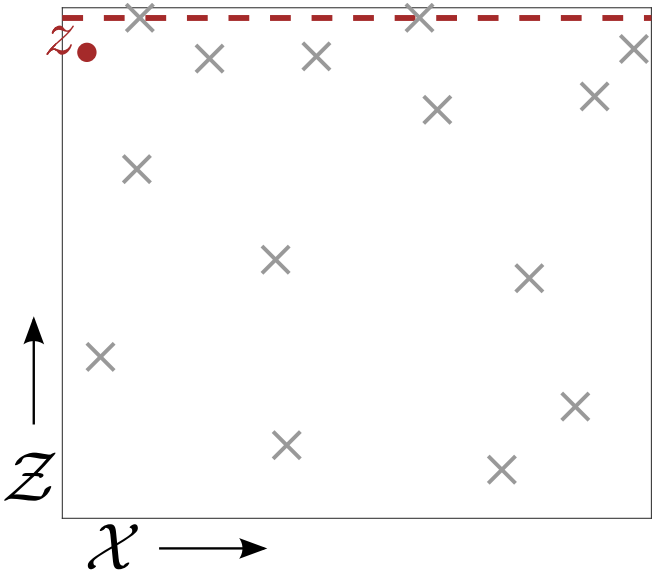
Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Algorithm: BOCA

(Kandasamy et al. ICML 2017)

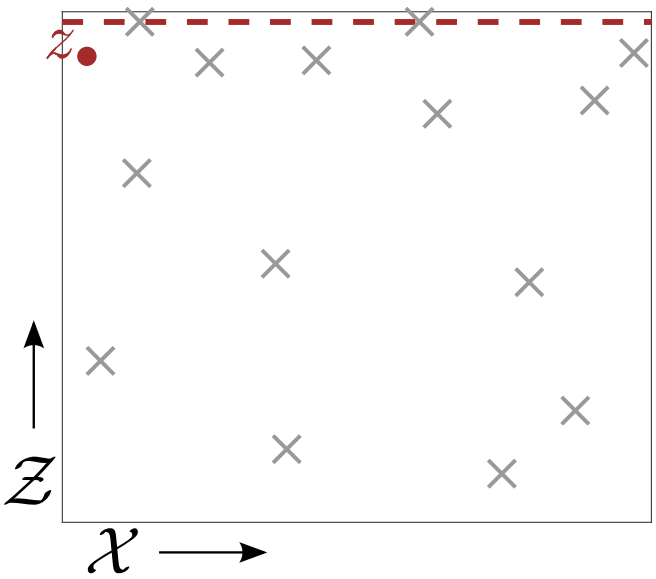


Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean	$\mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$
std-dev	$\sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

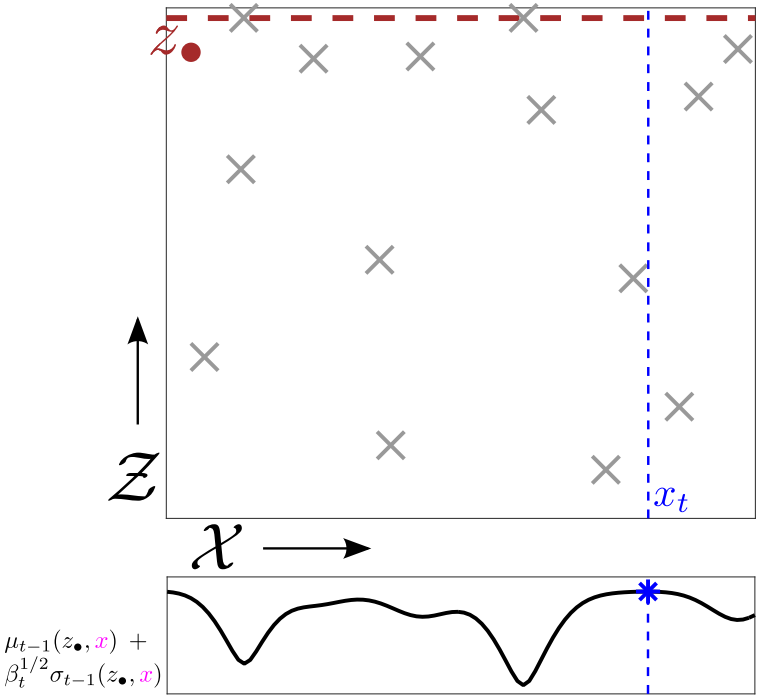
mean	$\mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$
std-dev	$\sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2} \sigma_{t-1}(z_\bullet, x)$$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

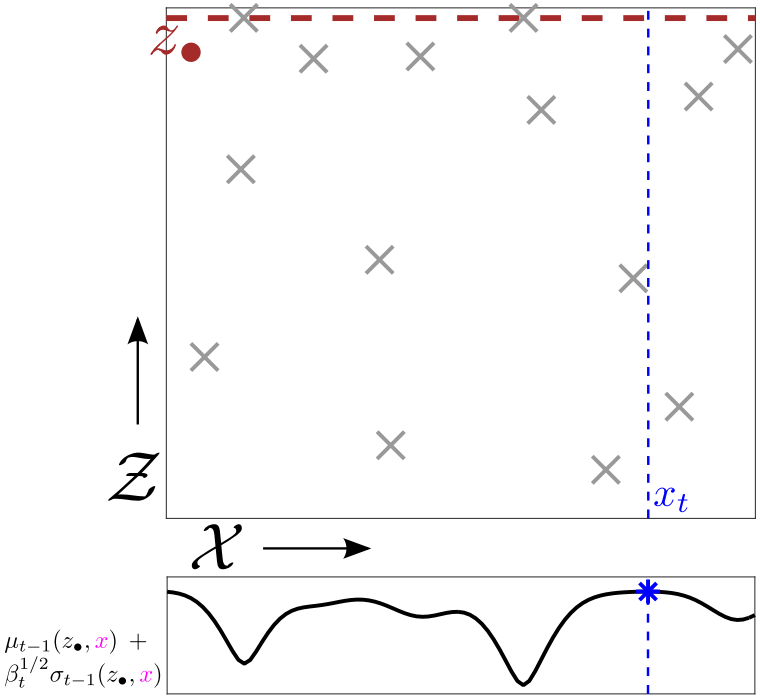
mean	$\mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$
std-dev	$\sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean $\quad \mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$

std-dev $\quad \sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

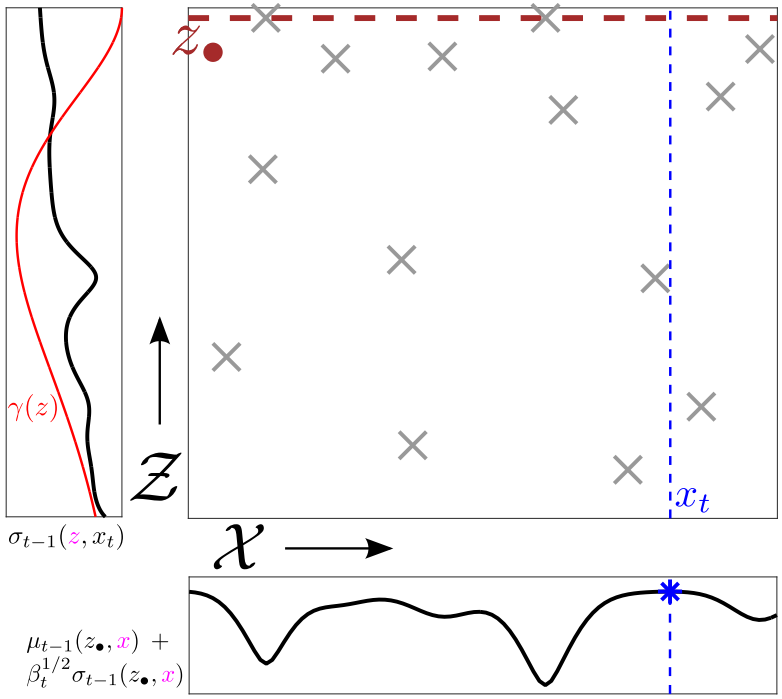
$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

(2) $$\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z : \sigma_{t-1}(z, x_t) \geq \gamma(z) \right\}$$

(3) $$z_t = \operatorname*{argmin}_{z \in \mathcal{Z}_t} \lambda(z) \qquad \text{(cheapest } z \text{ in } \mathcal{Z}_t\text{)}$$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean $\quad \mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$

std-dev $\quad \sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

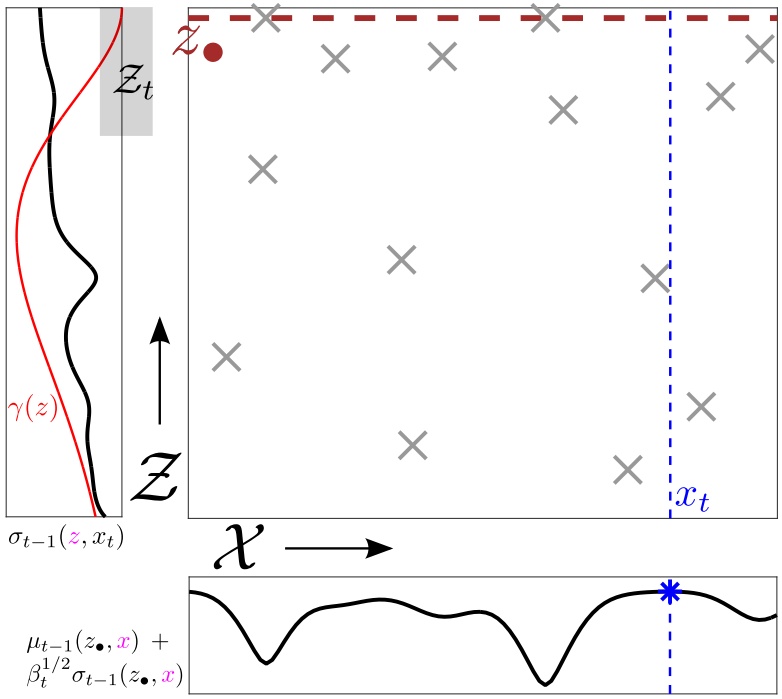
$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

(2) $$\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z : \sigma_{t-1}(z, x_t) \geq \gamma(z) \right\}$$

(3) $$z_t = \operatorname*{argmin}_{z \in \mathcal{Z}_t} \lambda(z) \qquad \text{(cheapest } z \text{ in } \mathcal{Z}_t\text{)}$$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean $\quad \mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$

std-dev $\quad \sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

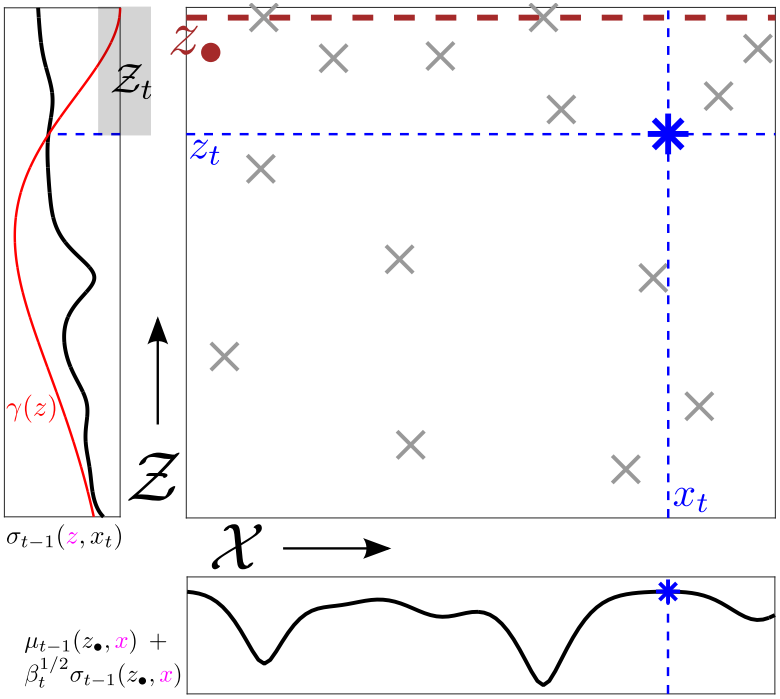
$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

(2) $$\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z : \sigma_{t-1}(z, x_t) \geq \gamma(z) \right\}$$

(3) $$z_t = \operatorname*{argmin}_{z \in \mathcal{Z}_t} \lambda(z) \qquad \text{(cheapest } z \text{ in } \mathcal{Z}_t\text{)}$$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean $\quad \mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$

std-dev $\quad \sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

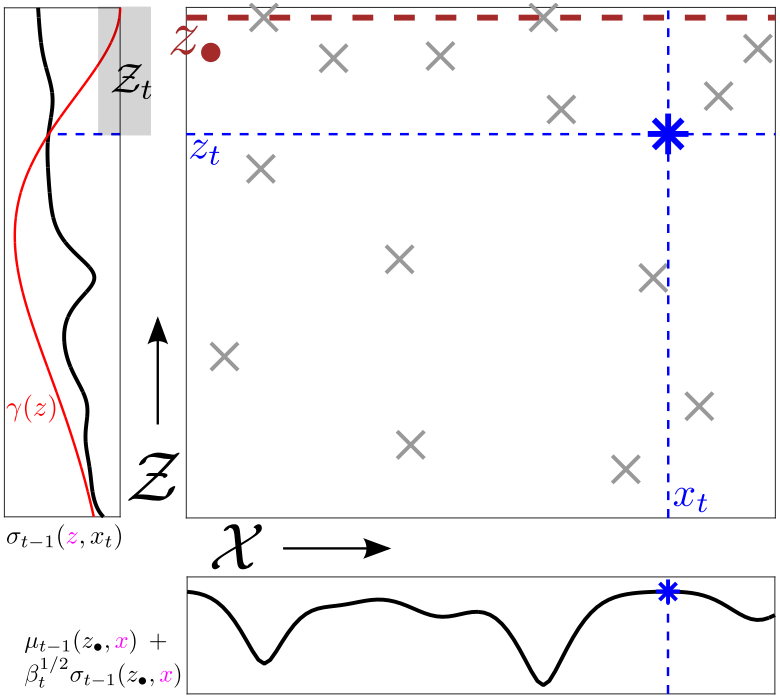
$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

(2) $$\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z : \sigma_{t-1}(z, x_t) \geq \gamma(z) \right\}$$

(3) $$z_t = \operatorname*{argmin}_{z \in \mathcal{Z}_t} \lambda(z) \qquad \text{(cheapest } z \text{ in } \mathcal{Z}_t\text{)}$$

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior $\mathcal{GP}$:

mean $\quad \mu_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}$

std-dev $\quad \sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \to \mathbb{R}_+$

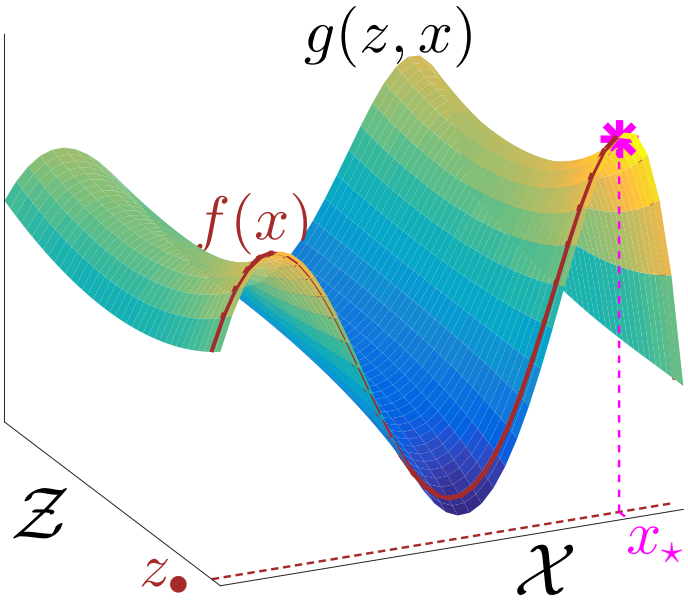
(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

$$x_t = \operatorname*{argmax}_{x \in \mathcal{X}} \; \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2}\sigma_{t-1}(z_\bullet, x)$$

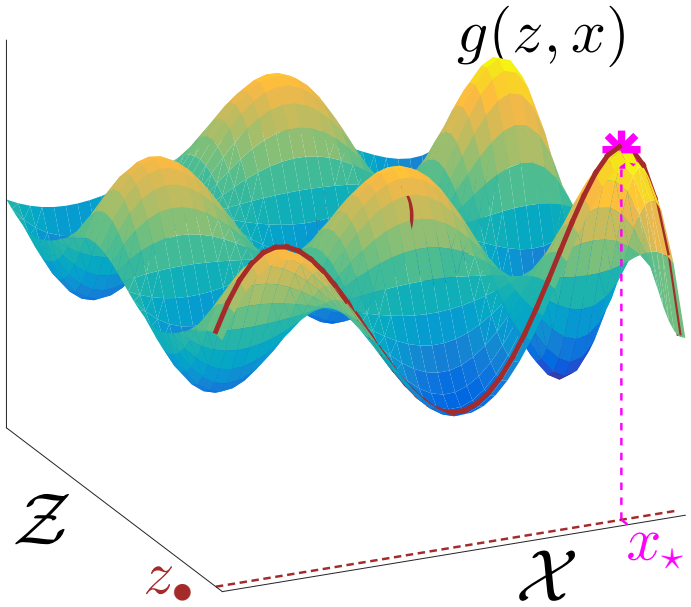
(2) $$\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z : \sigma_{t-1}(z, x_t) \geq \gamma(z) = \left(\frac{\lambda(z)}{\lambda(z_\bullet)}\right)^q \xi(z) \right\}$$

(3) $$z_t = \operatorname*{argmin}_{z \in \mathcal{Z}_t} \lambda(z) \qquad \text{(cheapest } z \text{ in } \mathcal{Z}_t\text{)}$$

Theoretical Results for BOCA

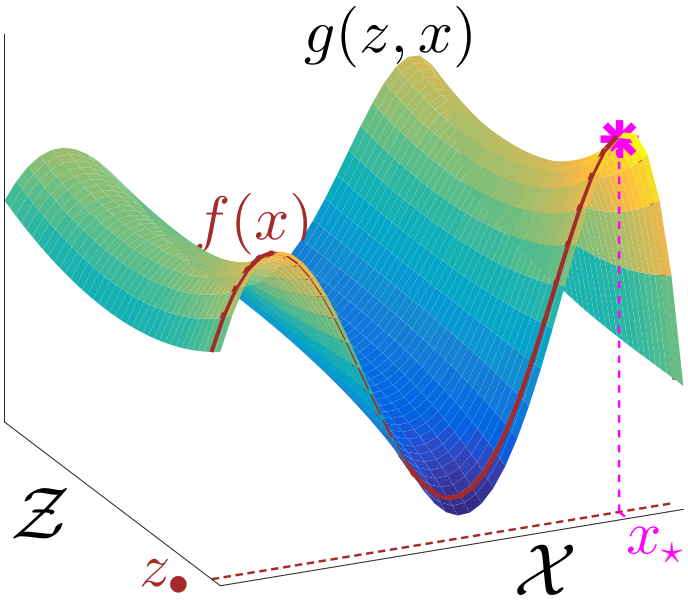


"good"

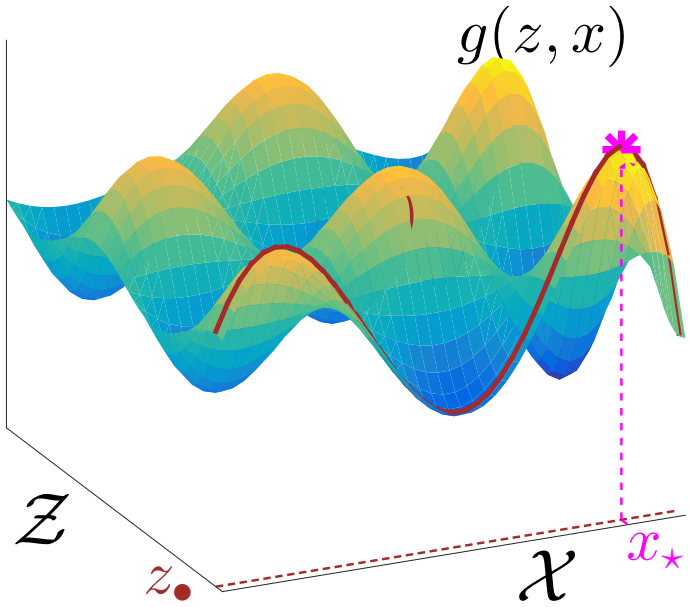


"bad"

Theoretical Results for BOCA



"good"



"bad"

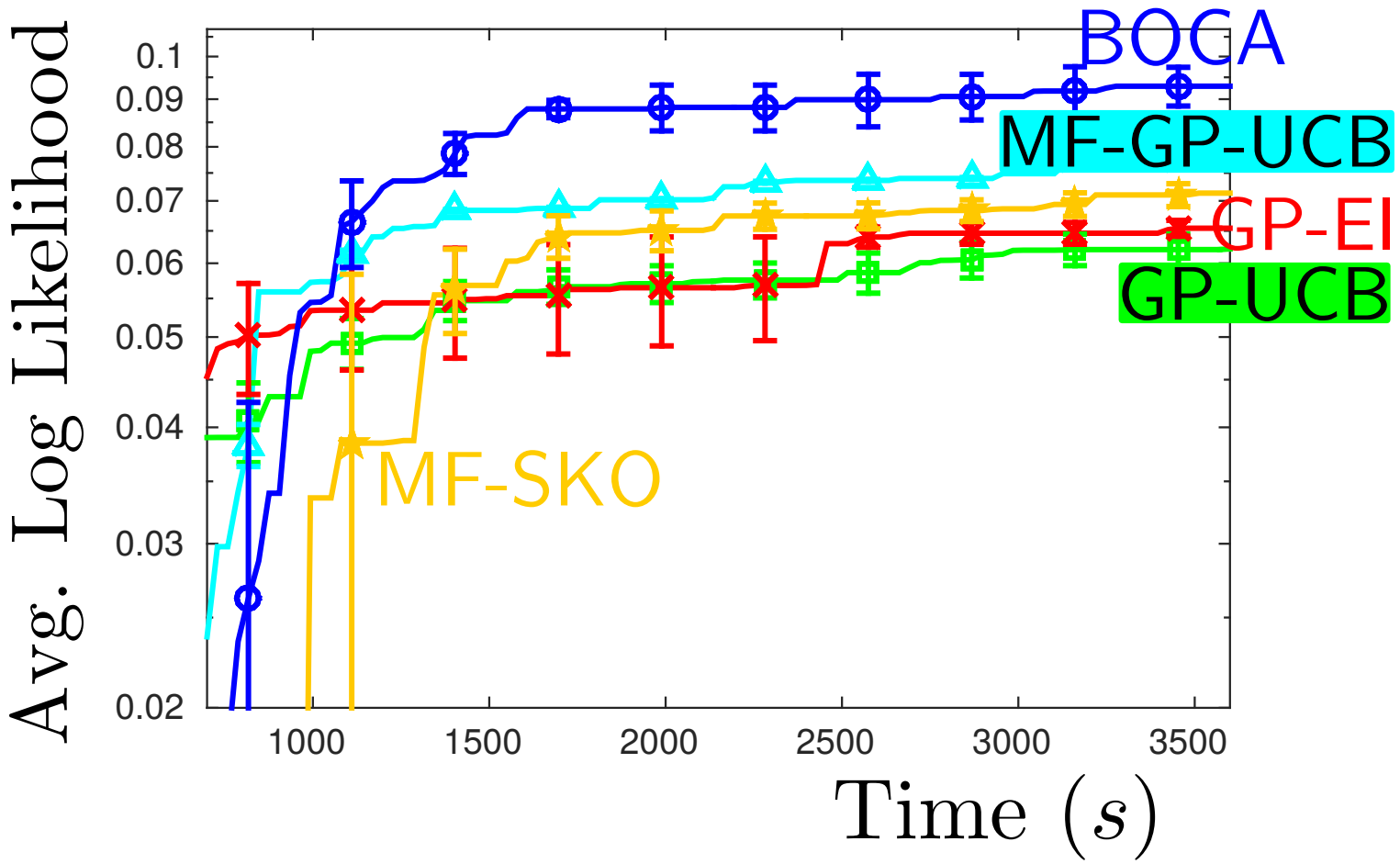
Theorem: (Informal)
BOCA does better, i.e. achieves better Simple regret, than GP-UCB. The improvements are better in the "good" setting when compared to the "bad" setting.

Experiment: Cosmological inference on Type-1a supernovae data

Estimate Hubble constant, dark matter fraction & dark energy fraction using data from $N_\bullet = 192$ supernovae.
Requires numerical integration on a grid of size $G_\bullet = 10^6$.
Approximate with $N \in [50, 192]$ or $G \in [10^2, 10^6]$ (2D fidelity space).

Experiment: Cosmological inference on Type-1a supernovae data

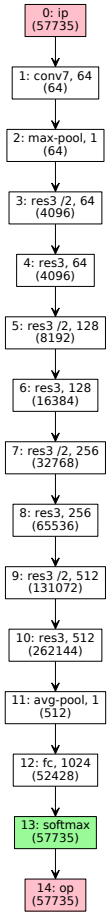
Estimate Hubble constant, dark matter fraction & dark energy fraction using data from $N_\bullet = 192$ supernovae.
Requires numerical integration on a grid of size $G_\bullet = 10^6$.
Approximate with $N \in [50, 192]$ or $G \in [10^2, 10^6]$ (2D fidelity space).



Outline

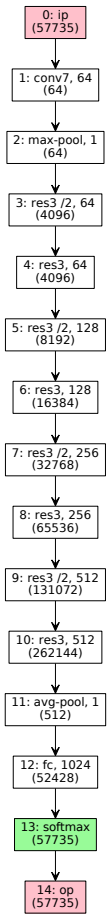
- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

Bandits on Graph Structured Domains
Neural Architecture Search



Feedforward network

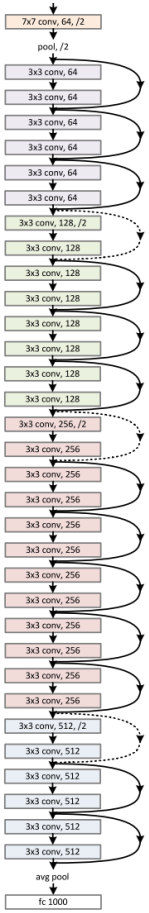
Bandits on Graph Structured Domains
Neural Architecture Search



Feedforward network

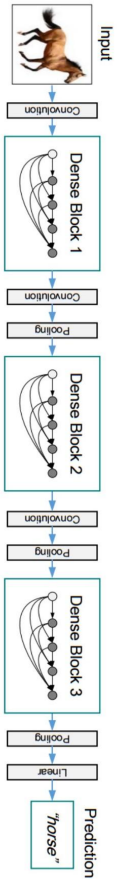
GoogLeNet

(Szegedy et al. 2015)



ResNet

(He et al. 2016)

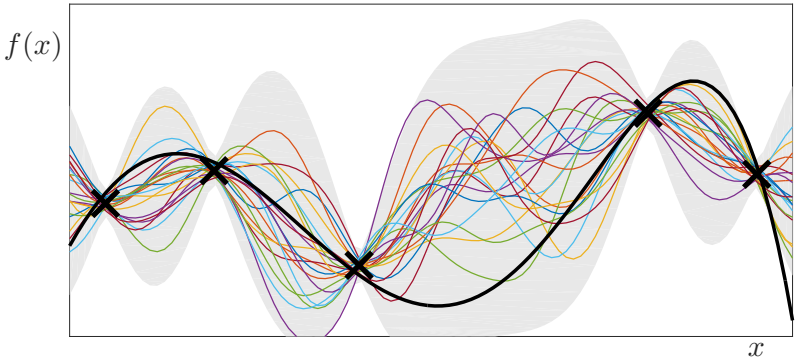


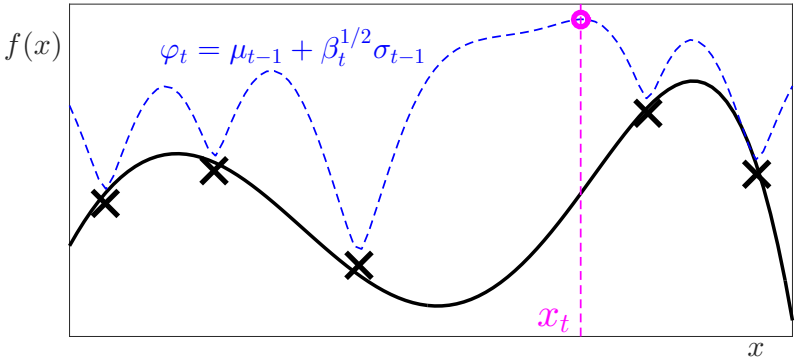
DenseNet

(Huang et al. 2017)

Tuning Neural Network Architectures

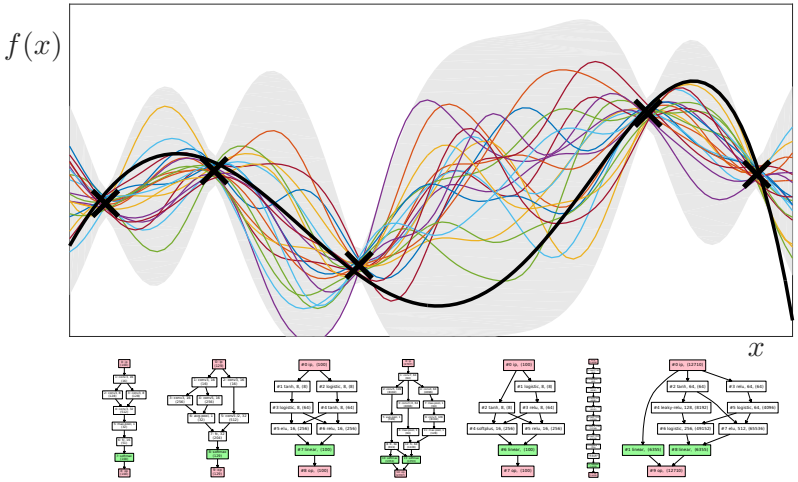
At each time step

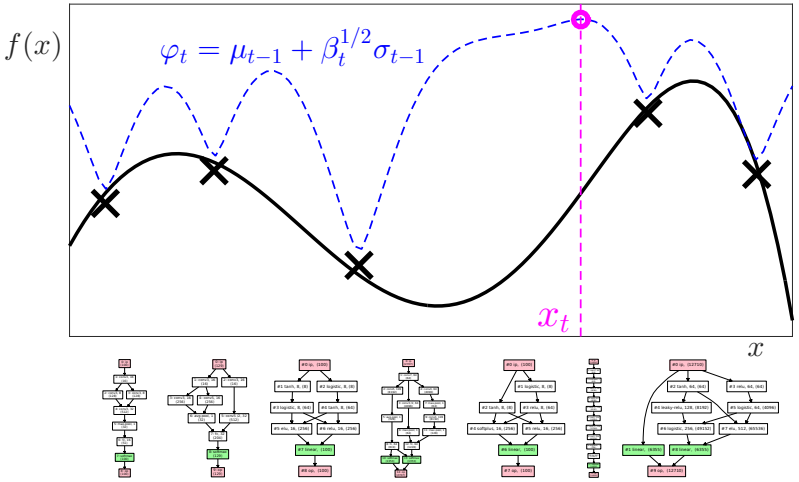




Tuning Neural Network Architectures

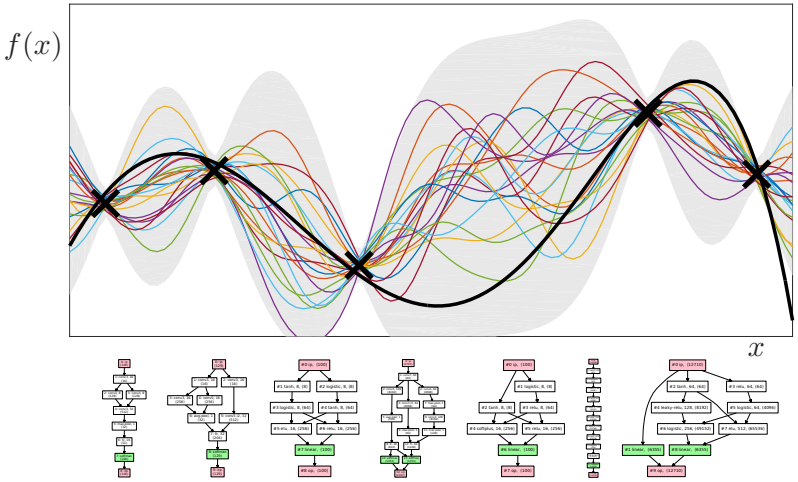
At each time step

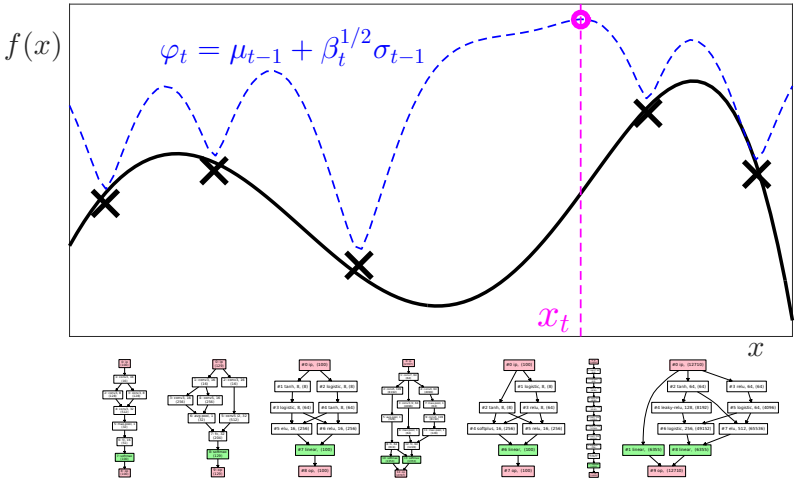




Tuning Neural Network Architectures

At each time step



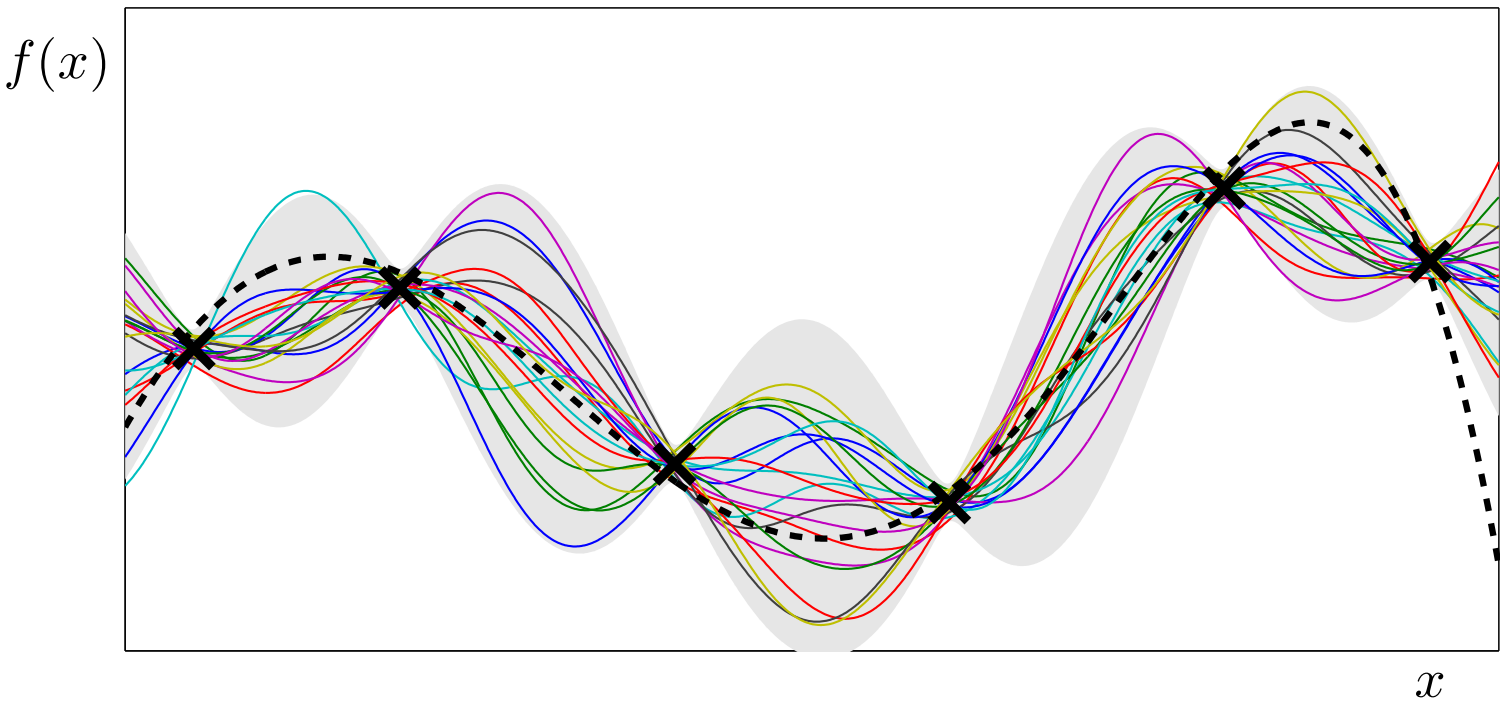


Main challenges

- Define a kernel between neural network architectures.
- Optimise φ_t on the space of neural networks.

Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.

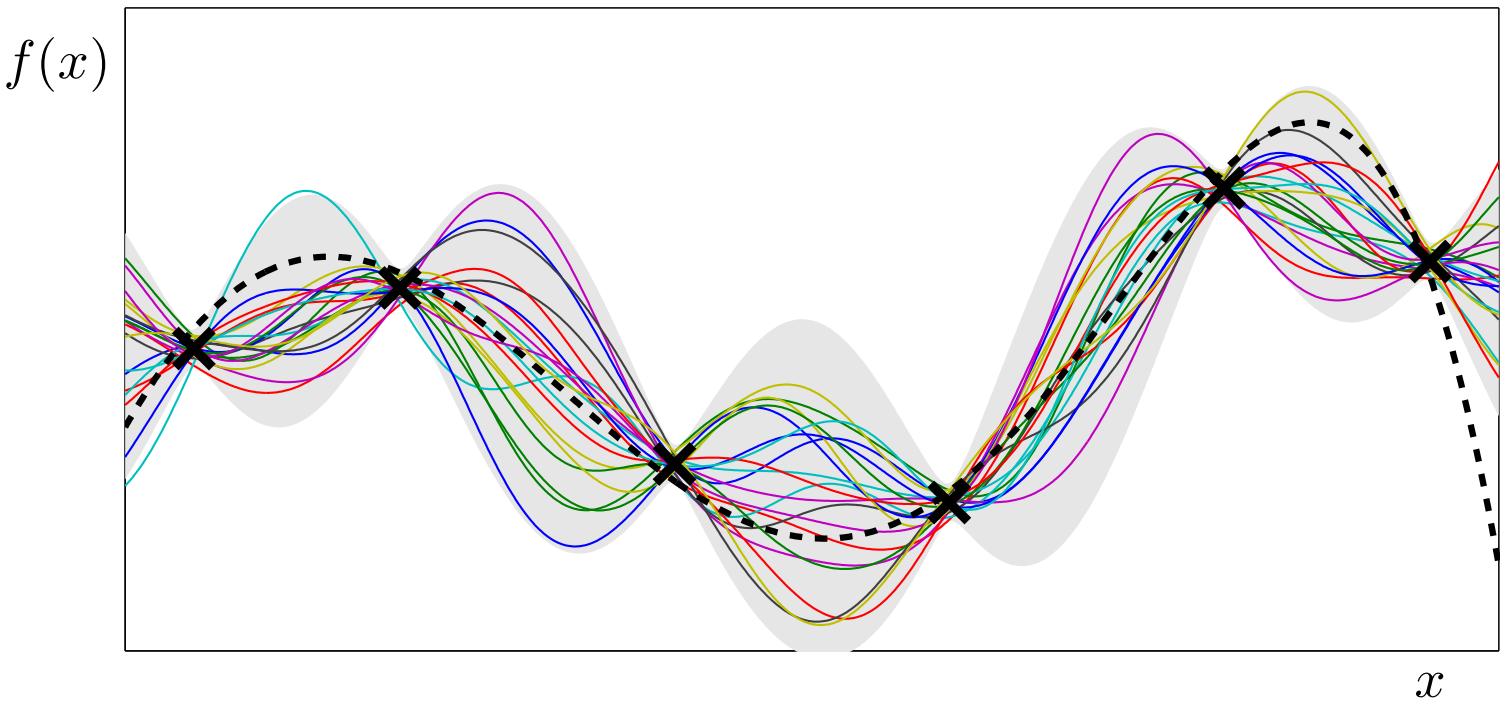


Statistical properties determined by the kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

- intuitively, $\kappa(x, x')$ is a measure of similarity between x and x'.

Gaussian Processes ($\mathcal{GP}$)

$\mathcal{GP}(\mu, \kappa)$: A distribution over functions from $\mathcal{X}$ to $\mathbb{R}$.



Statistical properties determined by the kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

- intuitively, $\kappa(x, x')$ is a measure of similarity between x and x'.

For many kernels, $\kappa(x, x') = e^{-\beta d(x,x')}, e^{-\beta d^2(x,x')}$,

- e.g. In Euclidean spaces, $e^{-\beta \|x-x'\|_1}$, $e^{-\beta \|x-x'\|_2^2}$.

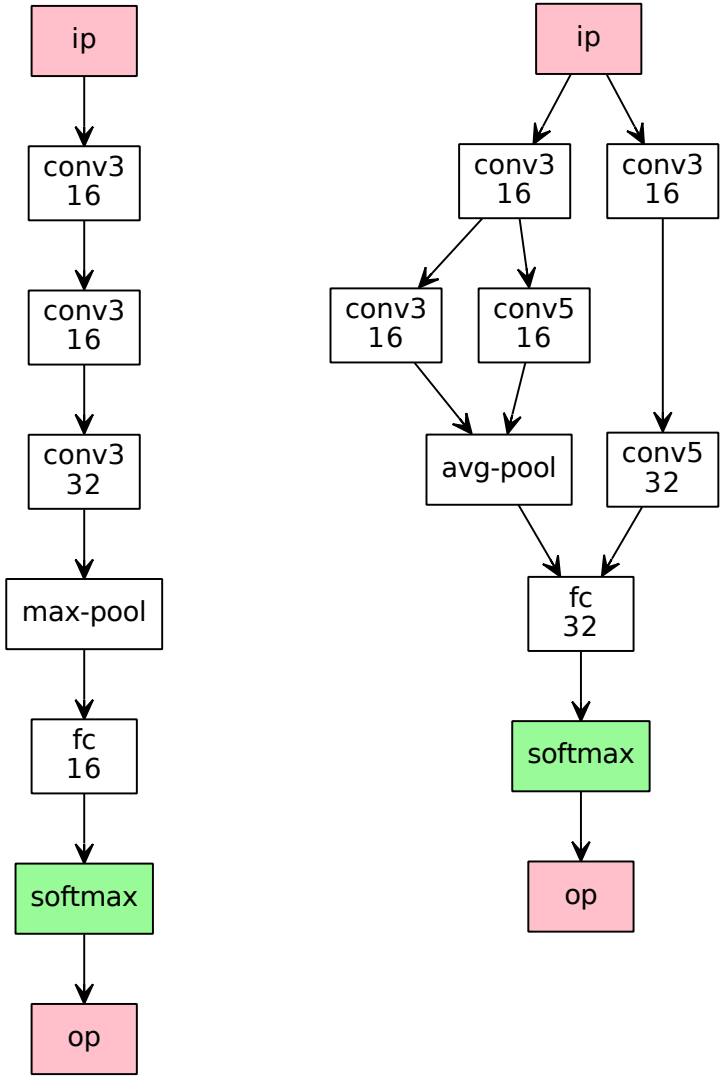
OTMANN: An optimal transport based distance for neural architectures. (Kandasamy et al. NeurIPS 2018)

Given this distance d, we can use use $e^{-\beta d}$ or $e^{-\beta d^2}$ as the kernel.

OTMANN: An optimal transport based distance for neural architectures.

(Kandasamy et al. NeurIPS 2018)

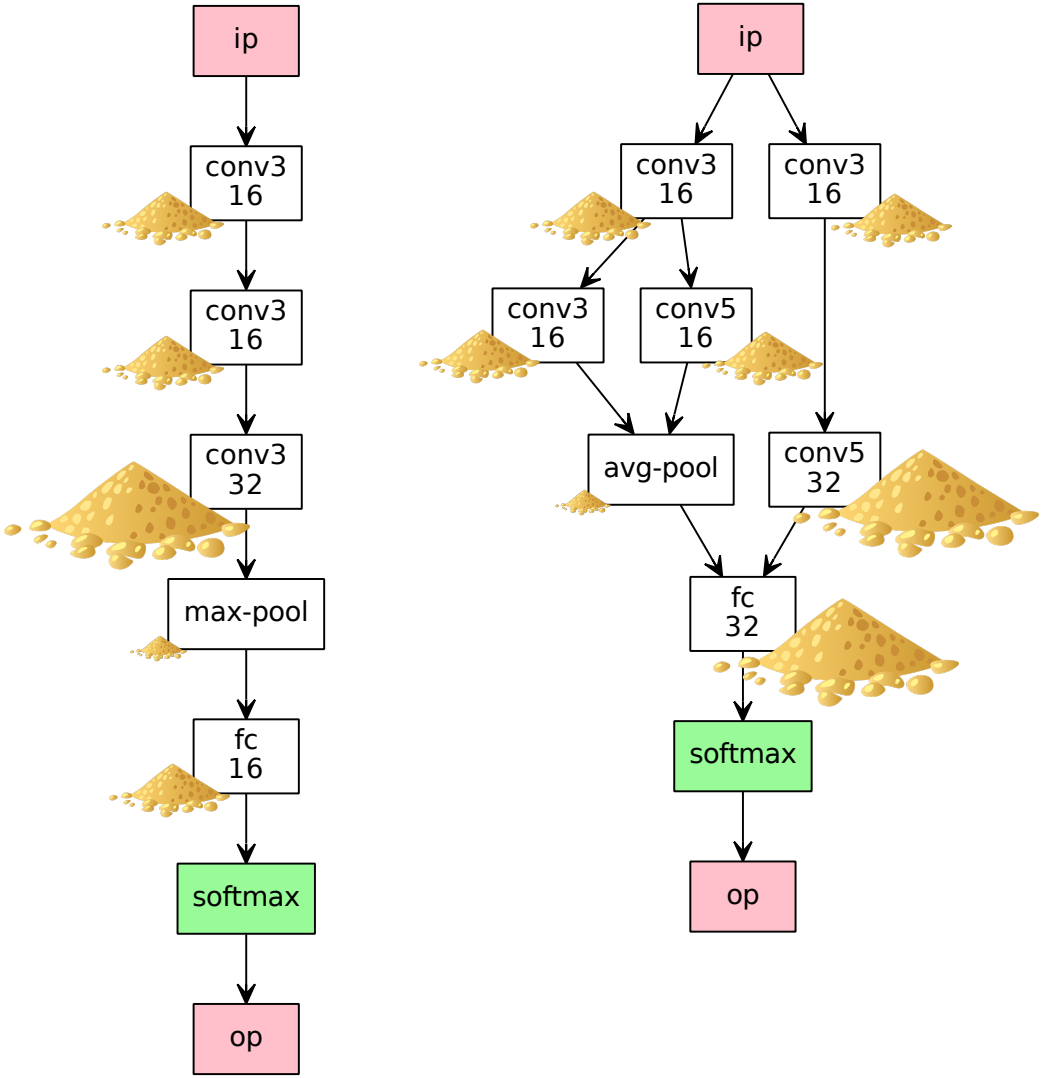
Given this distance d, we can use use $e^{-\beta d}$ or $e^{-\beta d^2}$ as the kernel.



OTMANN: An optimal transport based distance for neural architectures.

(Kandasamy et al. NeurIPS 2018)

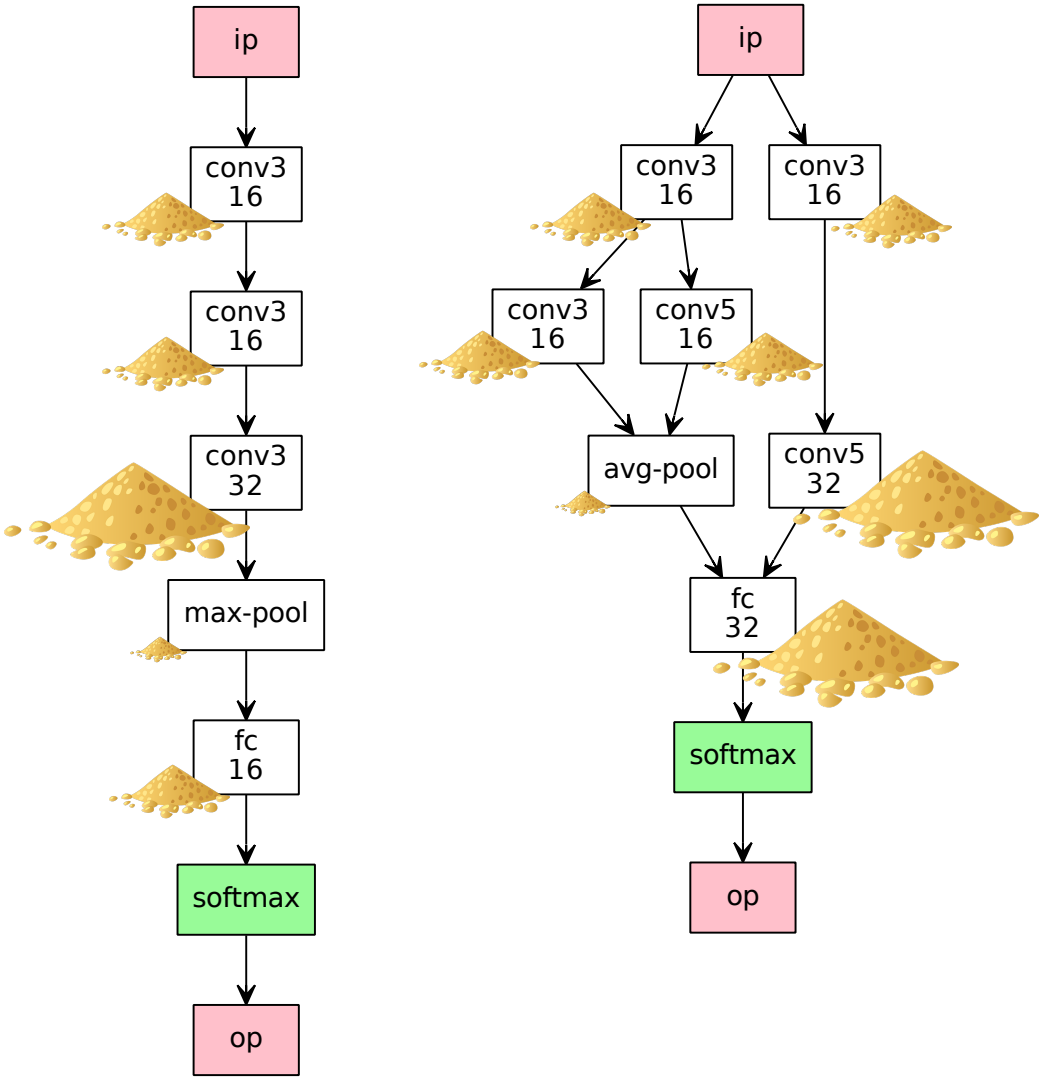
Given this distance d, we can use use $e^{-\beta d}$ or $e^{-\beta d^2}$ as the kernel.



OTMANN: An optimal transport based distance for neural architectures.

(Kandasamy et al. NeurIPS 2018)

Given this distance d, we can use use $e^{-\beta d}$ or $e^{-\beta d^2}$ as the kernel.



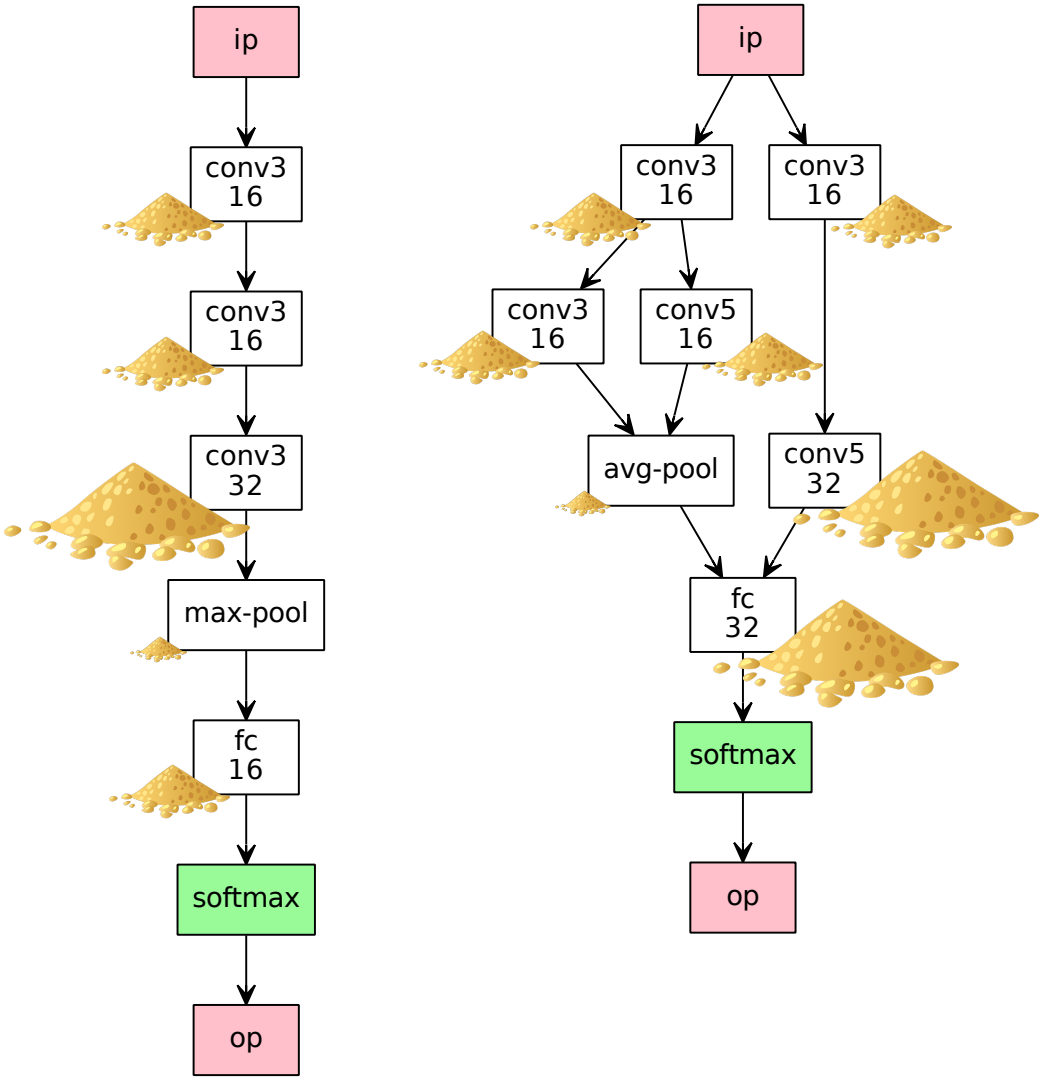
Penalty function:

- type of operation.
- structural position.

OTMANN: An optimal transport based distance for neural architectures.

(Kandasamy et al. NeurIPS 2018)

Given this distance d, we can use use $e^{-\beta d}$ or $e^{-\beta d^2}$ as the kernel.



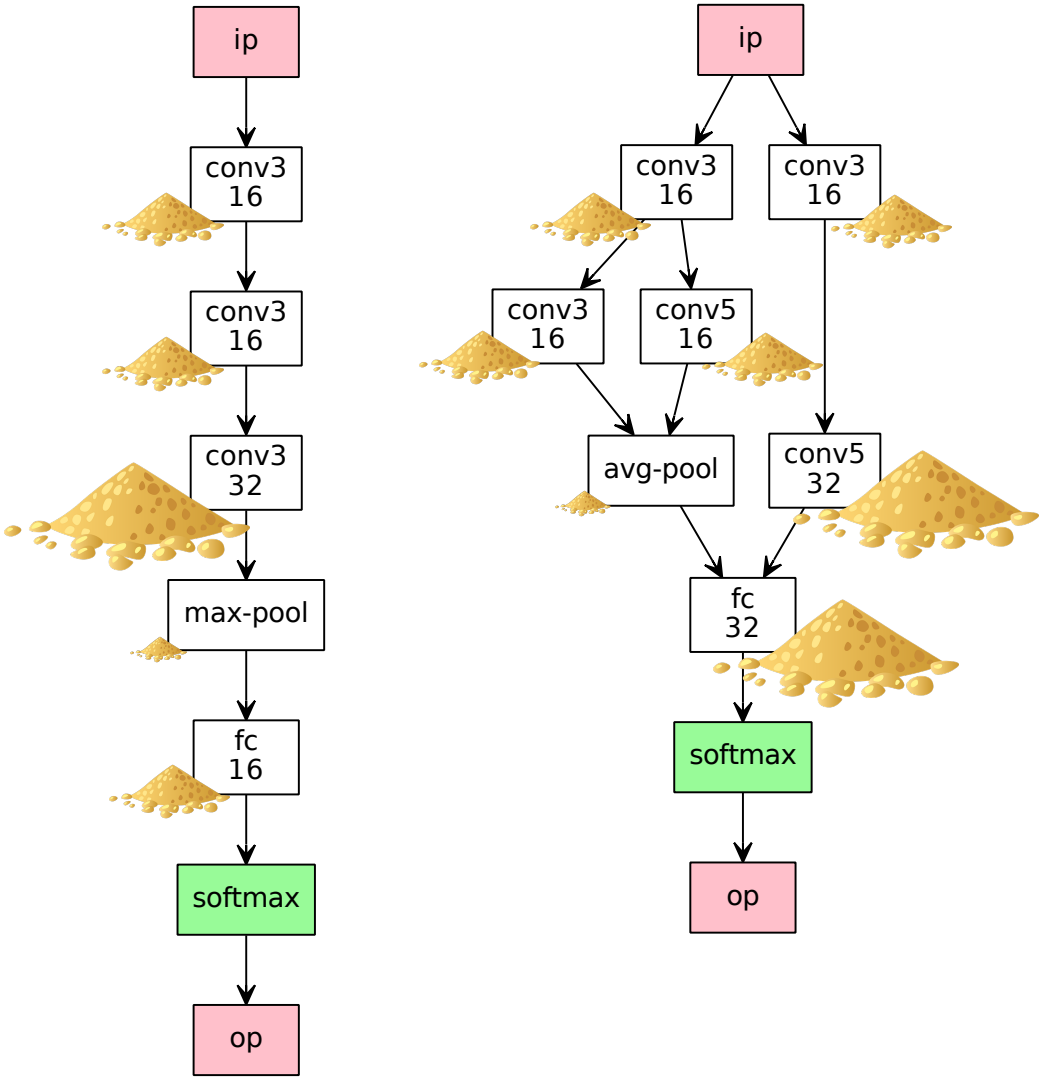
Penalty function:
- type of operation.
- structural position.

Can be computed via an optimal transport scheme.

OTMANN: An optimal transport based distance for neural architectures.

(Kandasamy et al. NeurIPS 2018)

Given this distance d, we can use use $e^{-\beta d}$ or $e^{-\beta d^2}$ as the kernel.

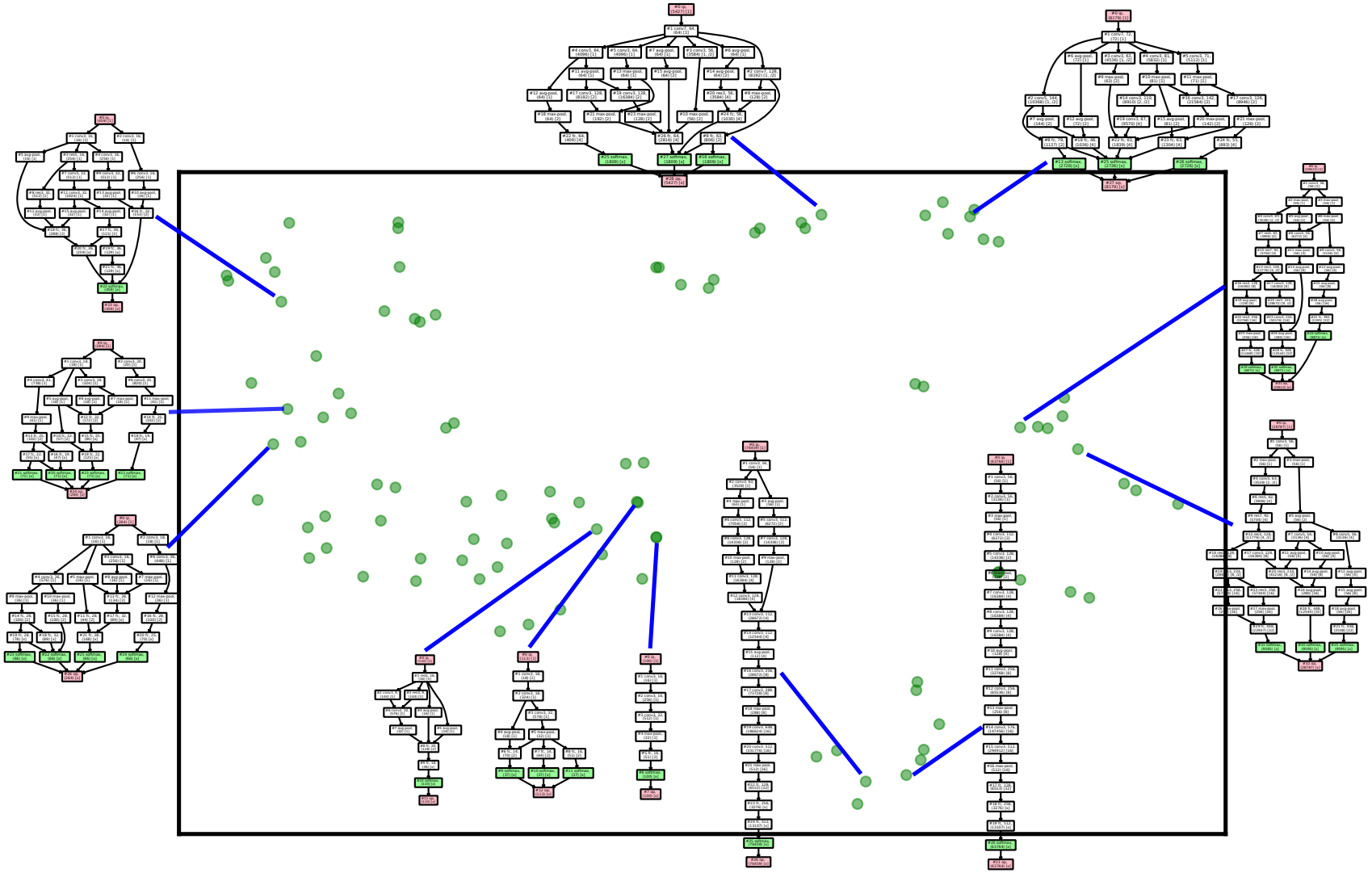


Penalty function:
- type of operation.
- structural position.

Can be computed via an optimal transport scheme.

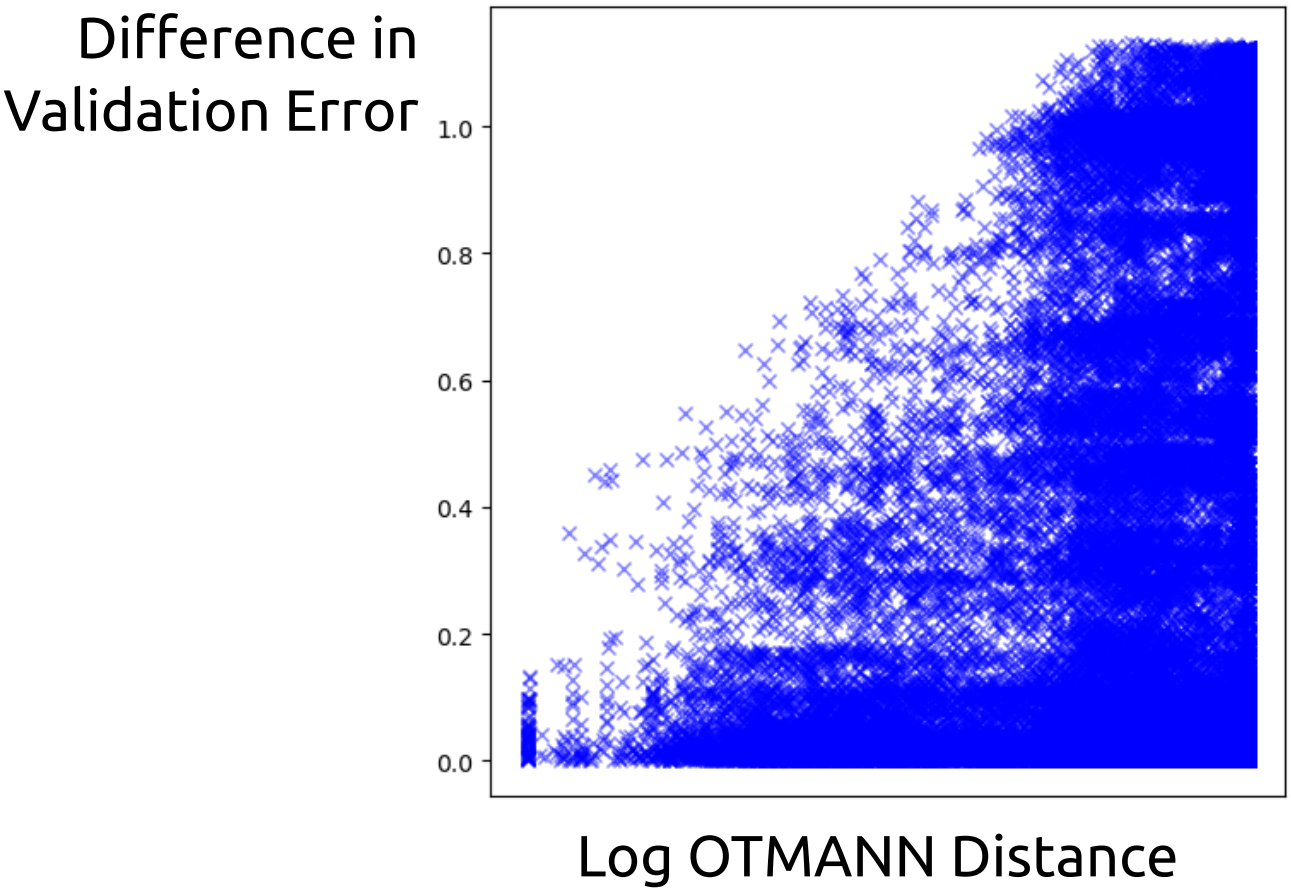
Theorem: OTMANN is a pseudo-distance.

OTMANN: Illustration with tSNE Embeddings



OTMANN correlates with cross validation performance

On the Naval propulsion Dataset:



Optimising the acquisition (UCB, TS, GP-EI etc.)

Modifiers to navigate search space:
inc_single, dec_single, inc_en_masse, dec_en_masse, remove_layer, wedge_layer, swap_layer, dup_path, skip_path.

Apply an evolutionary algorithm using these modifiers.

Optimising the acquisition (UCB, TS, GP-EI etc.)

Modifiers to navigate search space:
inc_single, dec_single, inc_en_masse, dec_en_masse, remove_layer, wedge_layer, swap_layer, dup_path, skip_path.

Apply an evolutionary algorithm using these modifiers.

Resulting procedure: NASBOT
Neural Architecture Search with Bayesian Optimisation and Optimal Transport (Kandasamy et al. NeurIPS 2018)

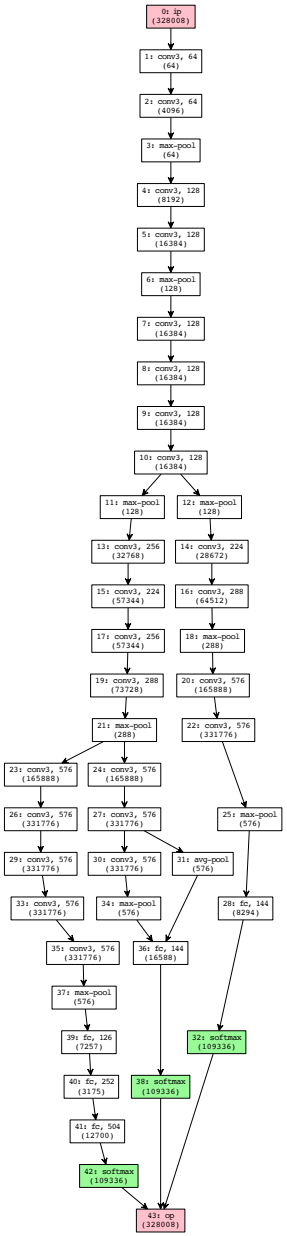
Test Error on 7 Datasets

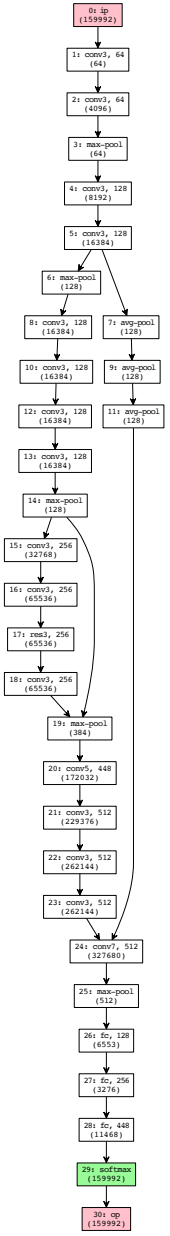
Method	Blog $(60K, 281)$	Indoor $(21K, 529)$	Slice $(54K, 385)$	Naval $(12K, 17)$	Protein $(46K, 9)$	News $(40K, 61)$	Cifar10 $(60K, 1K)$	Cifar10 $150K$ iters
RAND	0.780 ± 0.034	$\mathbf{0.115 \pm 0.023}$	0.758 ± 0.041	0.0103 ± 0.002	0.948 ± 0.024	$\mathbf{0.762 \pm 0.013}$	0.1342 ± 0.002	0.0914 ± 0.008
EA	0.806 ± 0.040	0.147 ± 0.010	0.733 ± 0.041	$\mathbf{0.0079 \pm 0.004}$	1.010 ± 0.038	$\mathbf{0.758 \pm 0.038}$	0.1411 ± 0.002	0.0915 ± 0.010
TreeBO	0.928 ± 0.053	0.168 ± 0.023	0.759 ± 0.079	0.0102 ± 0.002	0.998 ± 0.007	0.866 ± 0.085	0.1533 ± 0.004	0.1121 ± 0.004
NASBOT	$\mathbf{0.731 \pm 0.029}$	$\mathbf{0.117 \pm 0.008}$	$\mathbf{0.615 \pm 0.044}$	$\mathbf{0.0075 \pm 0.002}$	$\mathbf{0.902 \pm 0.033}$	$\mathbf{0.752 \pm 0.024}$	$\mathbf{0.1209 \pm 0.003}$	$\mathbf{0.0869 \pm 0.004}$

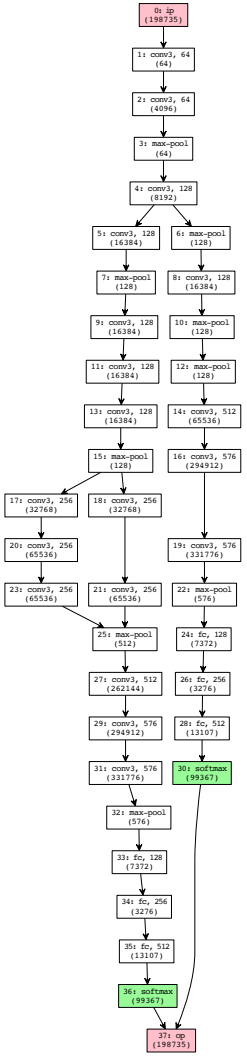
EA: (Floreano etl al. 2008, Kitano et al. 1990)

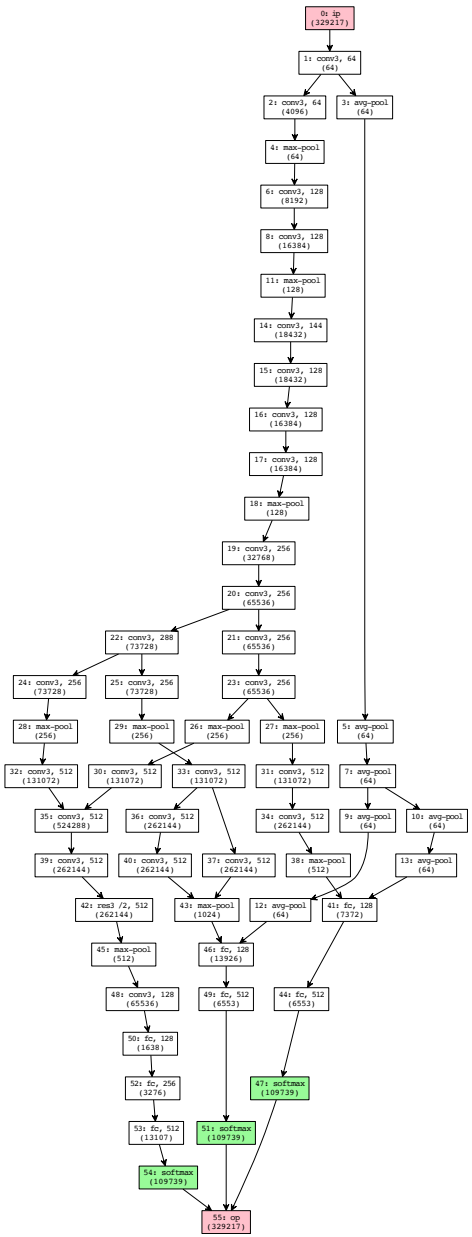
TreeBO: (Jenatton et al. 2017)

Architectures found on Cifar10

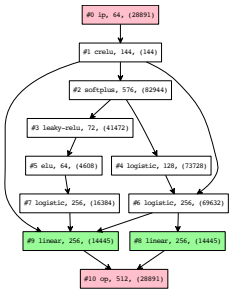


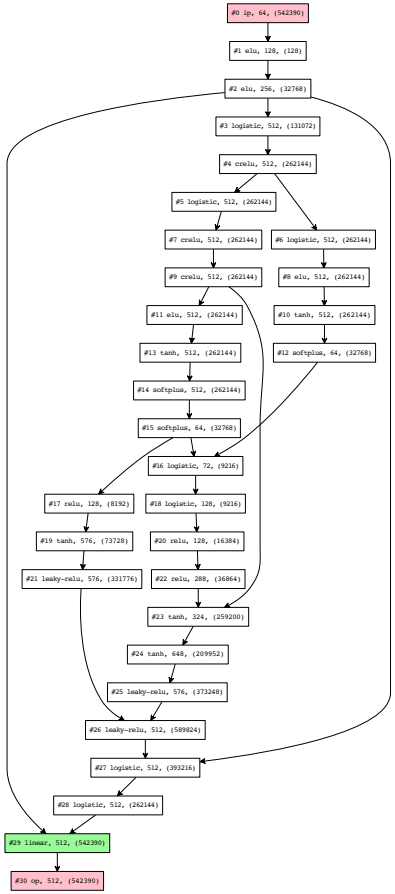


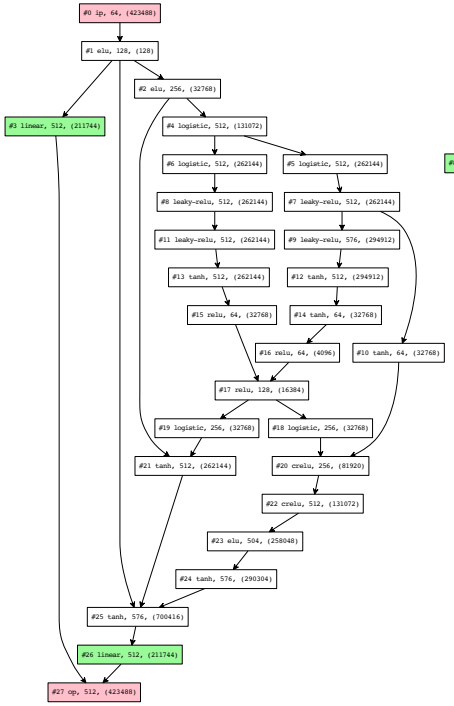


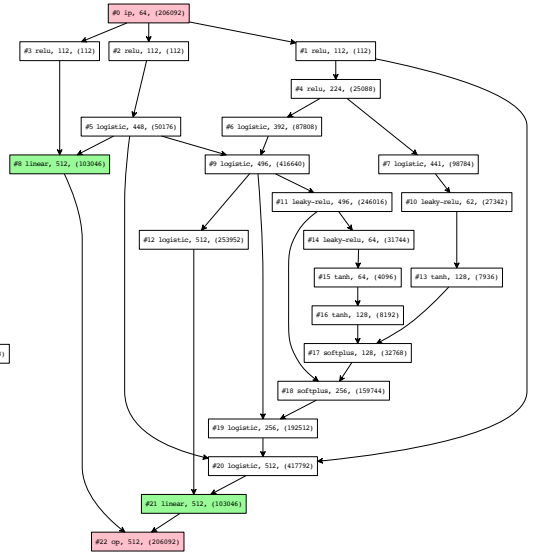


Architectures found on Indoor Location

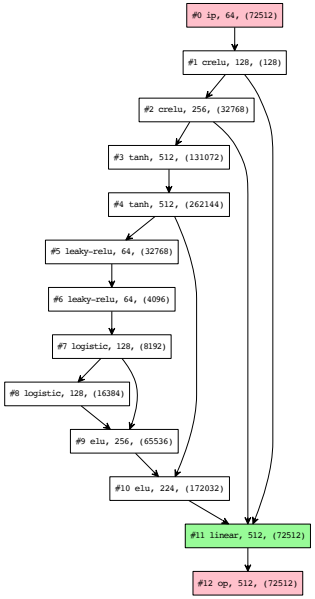


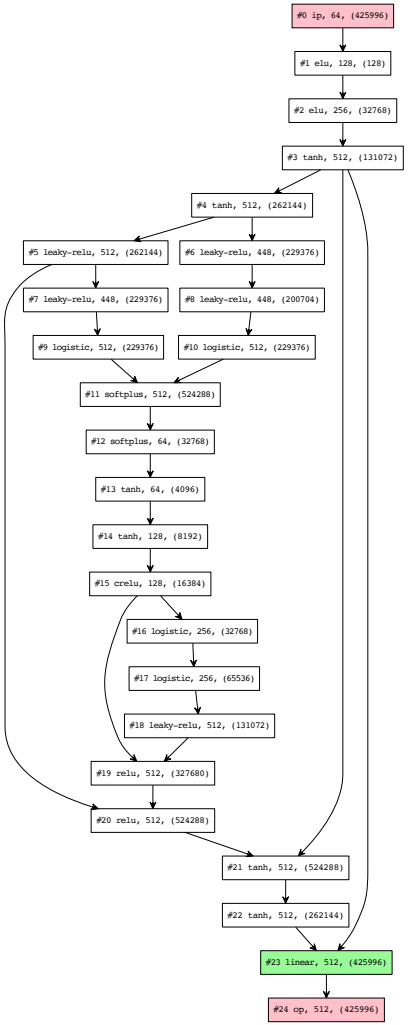


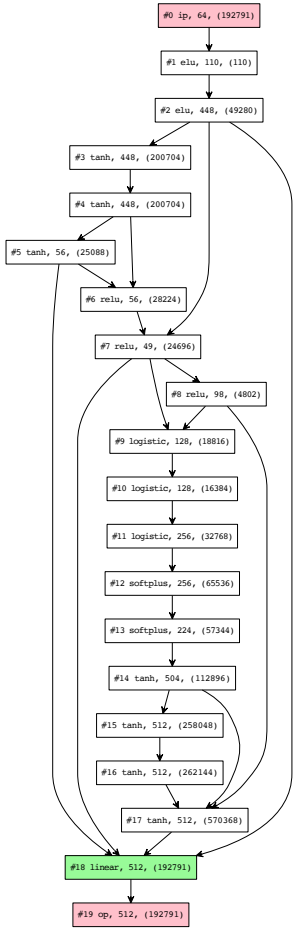


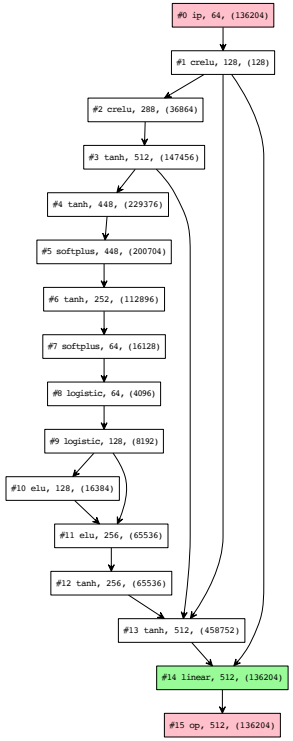


Architectures found on Slice Localisation



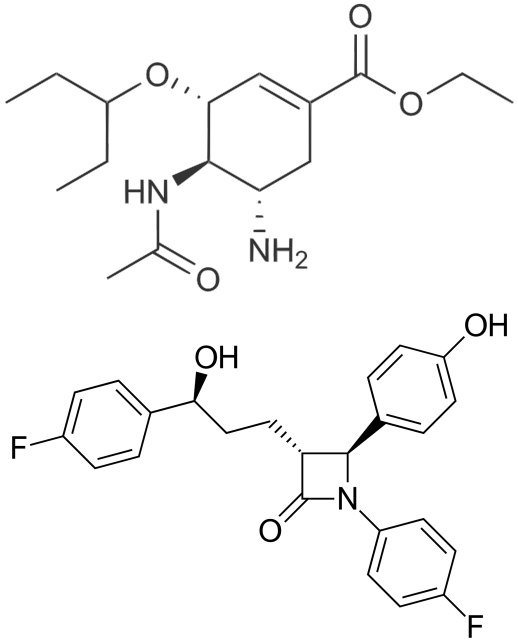




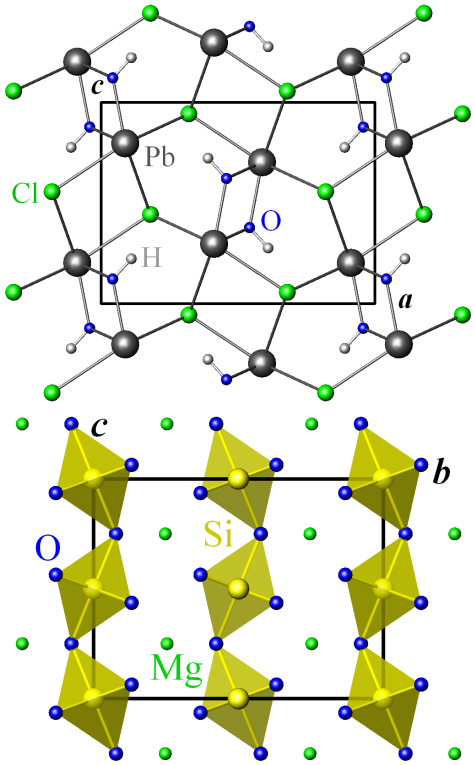


Bandits on Graph-structured Domains — On-going work

Drug Discovery with Small molecules



Crystal Structures



Outline

- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

High Dimensional Bandits

Optimise $f : [0, 1]^d \to \mathbb{R}$ when d is very large.

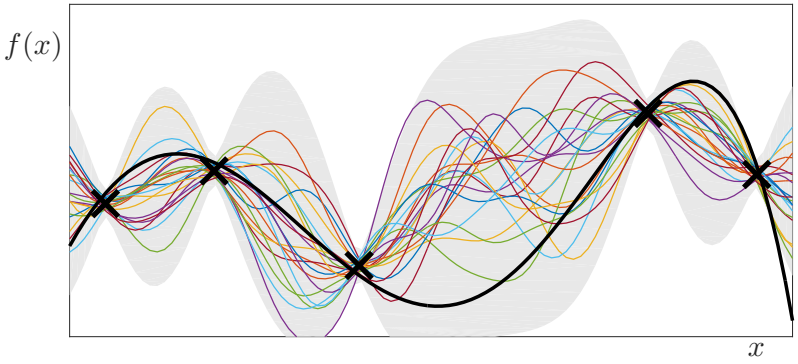
E.g. Tuning a machine learning model with several hyperparameters

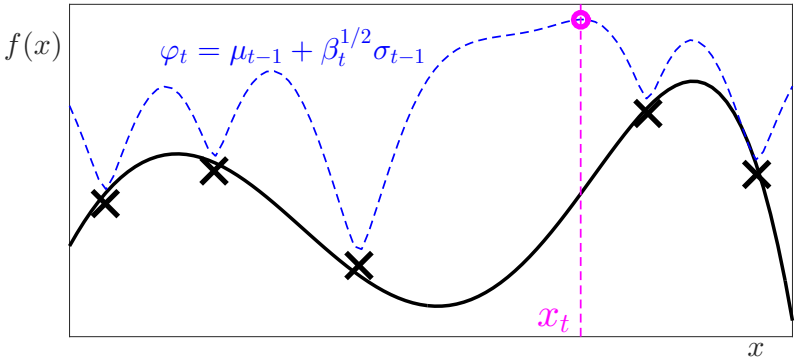
High Dimensional Bandits

Optimise $f : [0,1]^d \to \mathbb{R}$ when d is very large.

E.g. Tuning a machine learning model with several hyperparameters

At each time step



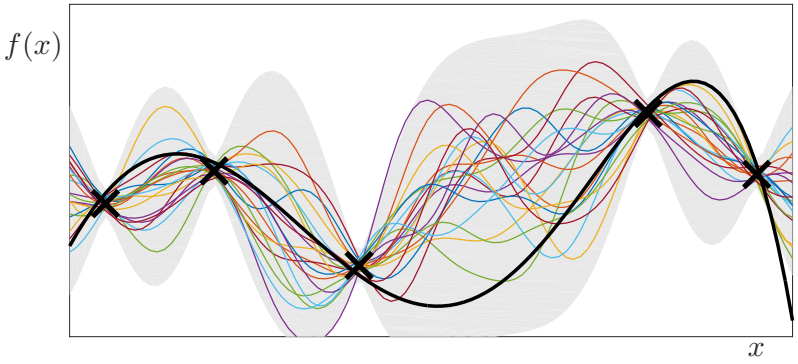


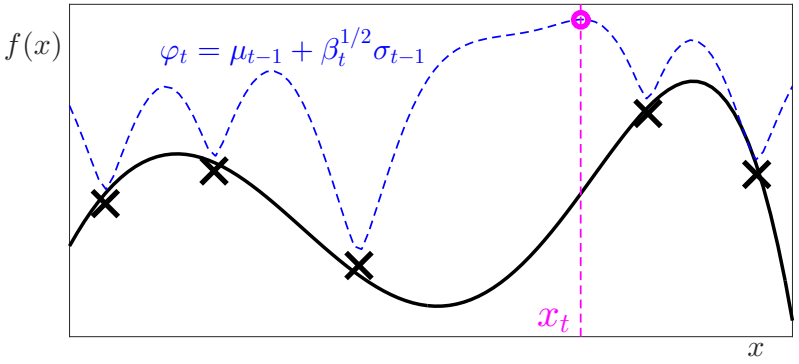
High Dimensional Bandits

Optimise $f : [0,1]^d \to \mathbb{R}$ when d is very large.

E.g. Tuning a machine learning model with several hyperparameters

At each time step





1. **Statistical Difficulty:** estimating a high dimensional GP requires several samples.
2. **Computational Difficulty:** maximising a high dimensional acquisition (e.g. upper confidence bound) φ_t.

Additive Models for High Dimensional BO

(Kandasamy et al. ICML 2015)

Structural assumption:

$$f(x) = f^{(1)}(x^{(1)}) + f^{(2)}(x^{(2)}) + \ldots + f^{(M)}(x^{(M)}).$$

$x^{(j)}$'s are p-dimensional, $p \ll d$.

Additive Models for High Dimensional BO

(Kandasamy et al. ICML 2015)

Structural assumption:

$$f(x) = f^{(1)}(x^{(1)}) + f^{(2)}(x^{(2)}) + \ldots + f^{(M)}(x^{(M)}).$$

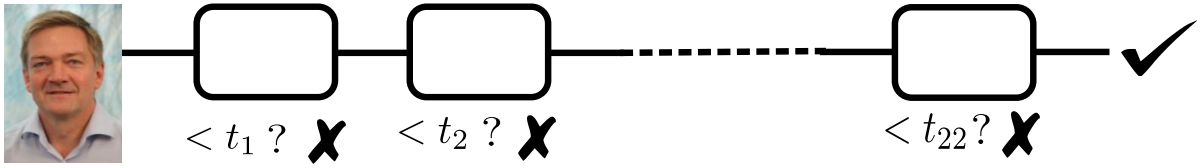
$x^{(j)}$'s are p-dimensional, $p \ll d$.

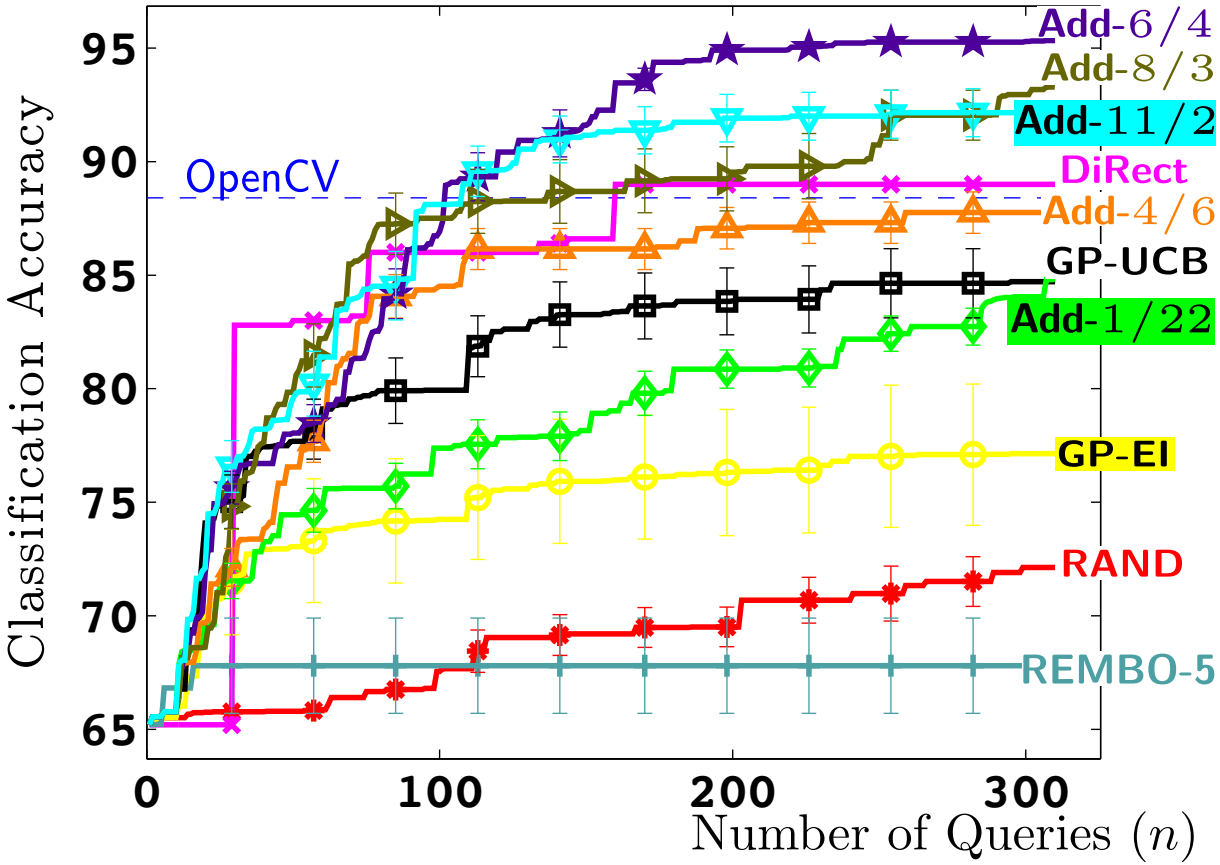
- Theory: Dependence on dimension improves from exponential to linear for UCB and TS.
- Better bias-variance trade-off even if f is not additive.
- Add-GP-UCB: algorithm with attractive computational properties.

Experiment: Viola & Jones Face Detection

A cascade of 22 weak classifiers.

Image classified negative if the score $<$ threshold at any stage.



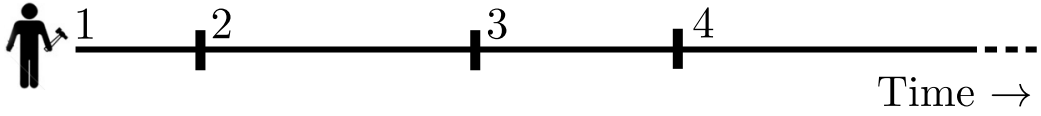


Outline

- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

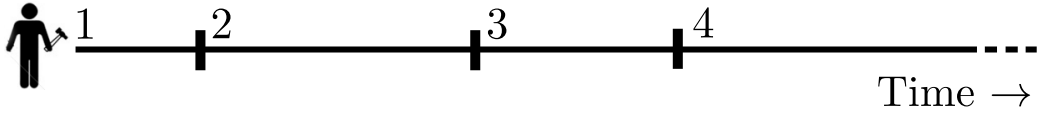
Parallel Evaluations

Sequential evaluations with one worker

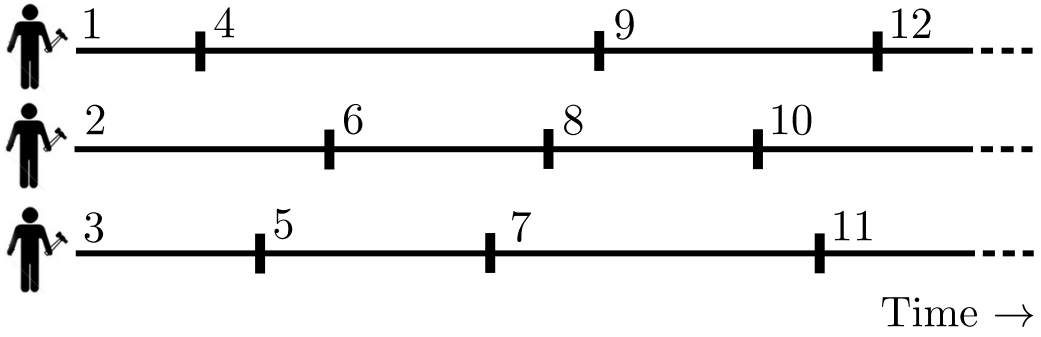


Parallel Evaluations

Sequential evaluations with one worker

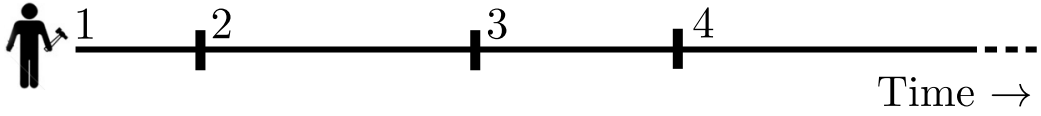


Parallel evaluations with M workers (Asynchronous)

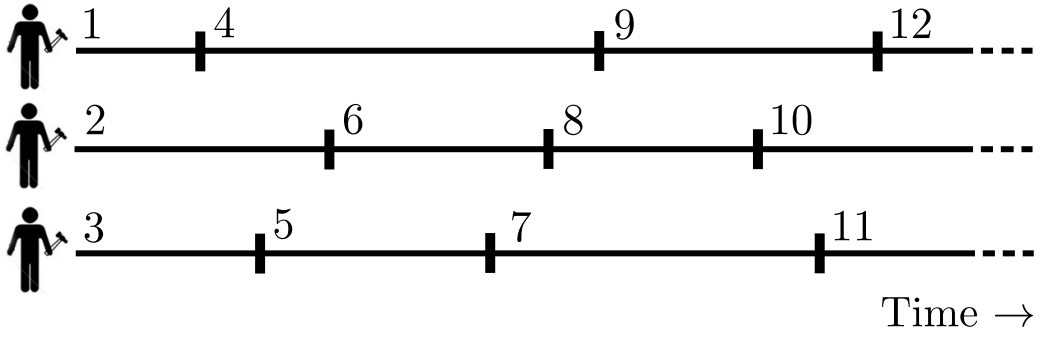


Parallel Evaluations

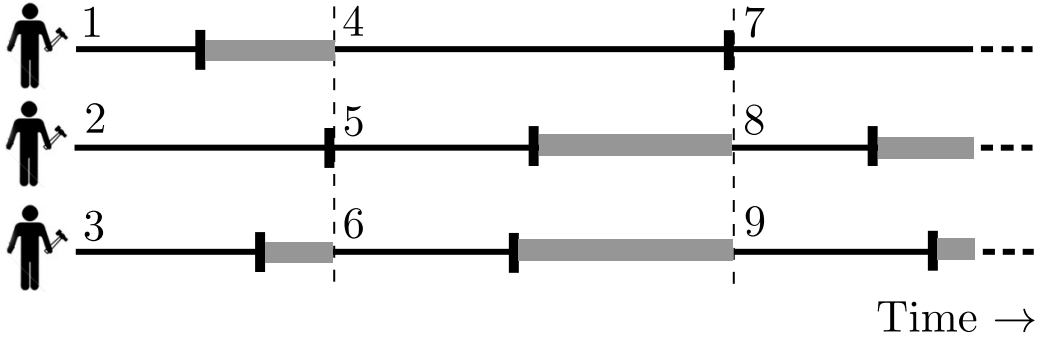
Sequential evaluations with one worker



Parallel evaluations with M workers (Asynchronous)

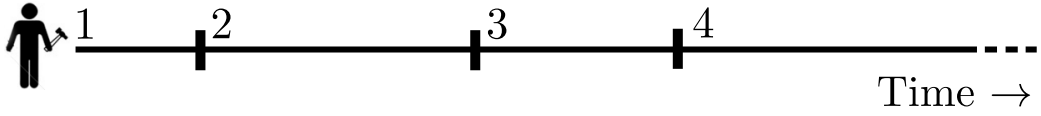


Parallel evaluations with M workers (Synchronous)



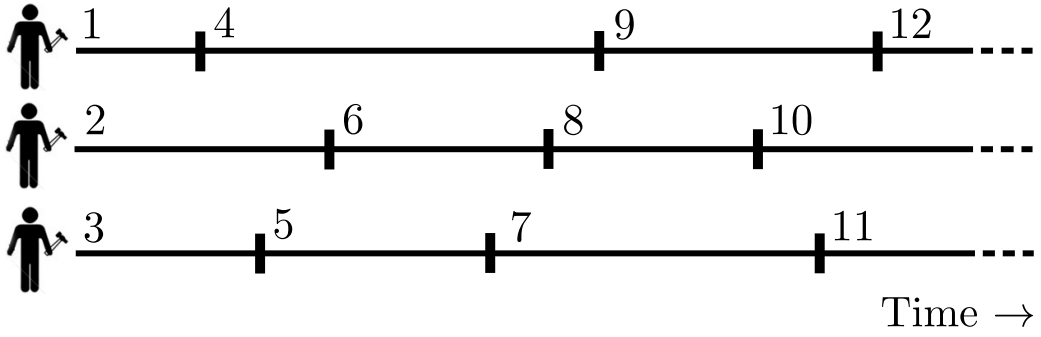
Parallel Evaluations

Sequential evaluations with one worker



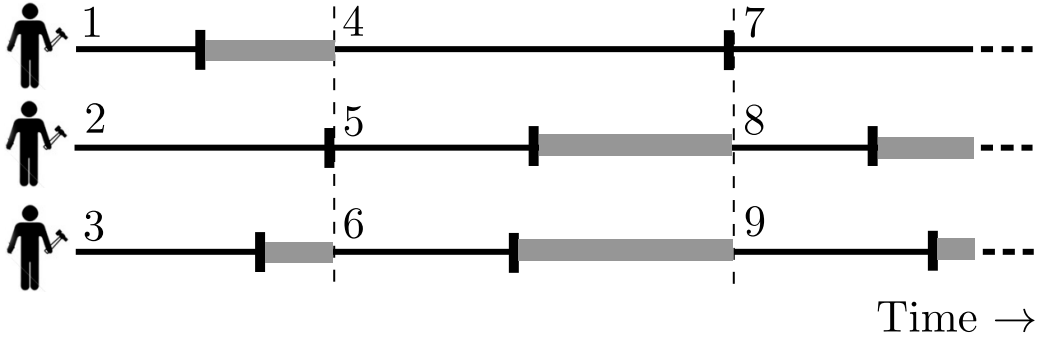
j^{th} job has feedback from all previous $j-1$ evaluations.

Parallel evaluations with M workers (Asynchronous)



j^{th} job missing feedback from exactly $M-1$ evaluations.

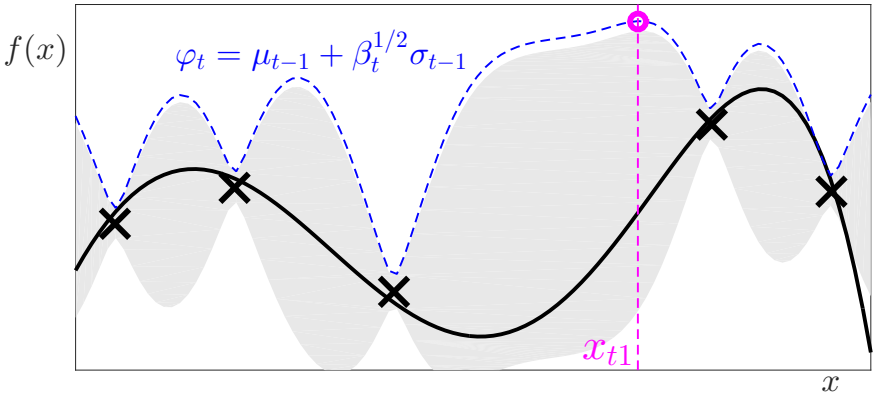
Parallel evaluations with M workers (Synchronous)



j^{th} job missing feedback from $\leq M-1$ evaluations.

Challenges in parallel BO: encouraging diversity

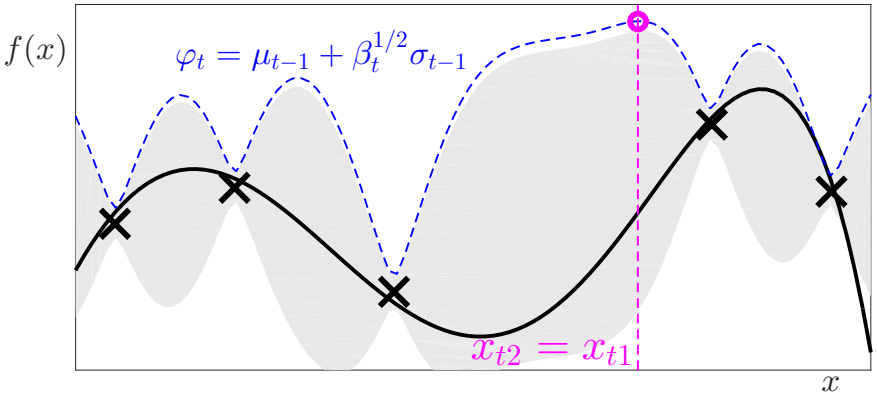
Direct application of UCB in the synchronous setting . . .



- First worker: maximise UCB, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.

Challenges in parallel BO: encouraging diversity

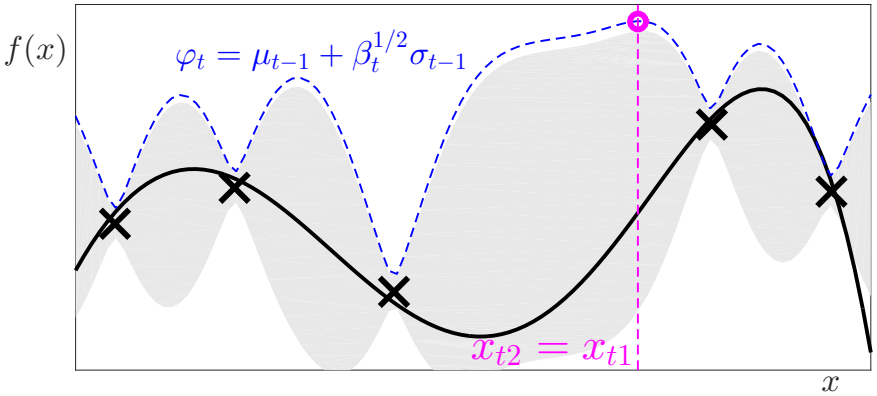
Direct application of UCB in the synchronous setting . . .



- First worker: maximise UCB, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.
- Second worker: UCB is the same! $x_{t1} = x_{t2}$

Challenges in parallel BO: encouraging diversity

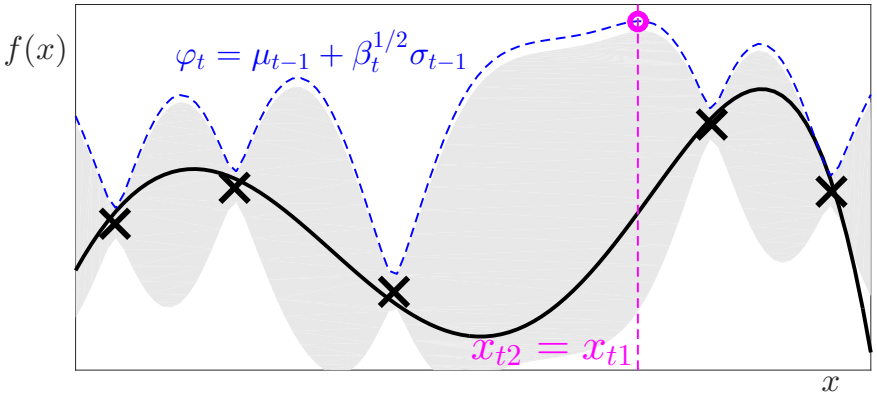
Direct application of UCB in the synchronous setting ...



- First worker: maximise UCB, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.
- Second worker: UCB is the same! $x_{t1} = x_{t2}$
- $x_{t1} = x_{t2} = \cdots = x_{tM}$.

Challenges in parallel BO: encouraging diversity

Direct application of UCB in the synchronous setting ...



- First worker: maximise UCB, $x_{t1} = \operatorname{argmax} \varphi_t(x)$.
- Second worker: UCB is the same! $x_{t1} = x_{t2}$
- $x_{t1} = x_{t2} = \cdots = x_{tM}$.

Direct application of popular (deterministic) strategies, e.g. UCB, EI, etc. do not work. Need to "encourage diversity".

Challenges in parallel BO: encouraging diversity

- Add hallucinated observations.

  (Ginsbourger et al. 2011, Janusevkis et al. 2012)
- Optimise an acquisition over $\mathcal{X}^M$ (e.g. M-product UCB).

  (Wang et al 2016, Wu & Frazier 2017)
- Resort to heuristics, typically requires additional hyperparameters and/or computational routines.

  (Contal et al. 2013, Gonzalez et al. 2015, Shah & Ghahramani 2015, Wang et al. 2017, Wang et al. 2018)

Challenges in parallel BO: encouraging diversity

- Add hallucinated observations.

  (Ginsbourger et al. 2011, Janusevkis et al. 2012)
- Optimise an acquisition over $\mathcal{X}^M$ (e.g. M-product UCB).

  (Wang et al 2016, Wu & Frazier 2017)
- Resort to heuristics, typically requires additional hyperparameters and/or computational routines.

  (Contal et al. 2013, Gonzalez et al. 2015, Shah & Ghahramani 2015, Wang et al. 2017, Wang et al. 2018)

In (Kandasamy et al. AISTATS 2018): Direct application of Thompson sampling works!

Challenges in parallel BO: encouraging diversity

- Add hallucinated observations.

  (Ginsbourger et al. 2011, Janusevkis et al. 2012)
- Optimise an acquisition over $\mathcal{X}^M$ (e.g. M-product UCB).

  (Wang et al 2016, Wu & Frazier 2017)
- Resort to heuristics, typically requires additional hyperparameters and/or computational routines.

  (Contal et al. 2013, Gonzalez et al. 2015, Shah & Ghahramani 2015, Wang et al. 2017, Wang et al. 2018)

In (Kandasamy et al. AISTATS 2018): Direct application of Thompson sampling works!

- Conceptually and computationally simple in practice: *does not require explicit diversity strategies.*
- Comes with theoretical guarantees.

Theoretical Results – Simple regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) \; - \max_{t=1,\ldots,n} f(x_t).$$

seqTS

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

(Russo & van Roy 2014)

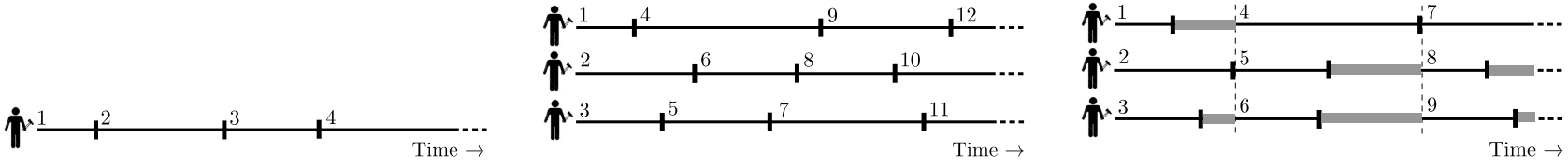
Theoretical Results – Simple regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) \; - \max_{t=1,\ldots,n} f(x_t).$$

seqTS

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$

(Russo & van Roy 2014)

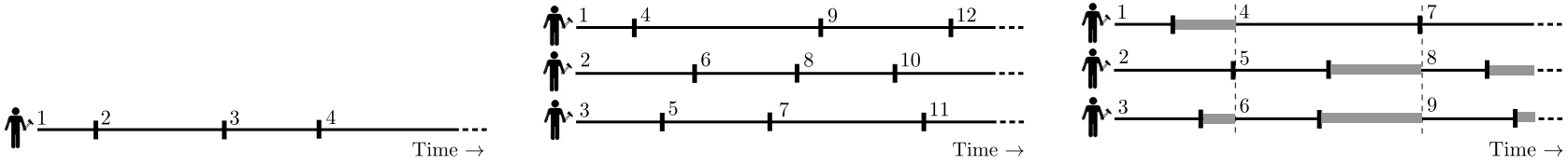


Theoretical Results – Simple regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) \; - \max_{t=1,\ldots,n} f(x_t).$$

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$



Theorem: synTS (Kandasamy et al. AISTATS 2018)

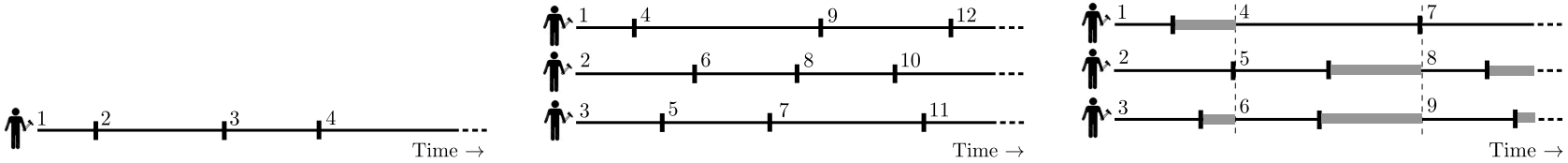
$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\sqrt{\log(M)}}{n} + \sqrt{\frac{\Psi_n \log(n+M)}{n}}$$

Theoretical Results – Simple regret after n evaluations

$$\mathrm{SR}(n) = f(x_\star) \; - \max_{t=1,\ldots,n} f(x_t).$$

seqTS (Russo & van Roy 2014)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \sqrt{\frac{\Psi_n \log(n)}{n}}$$



Theorem: synTS (Kandasamy et al. AISTATS 2018)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\sqrt{\log(M)}}{n} + \sqrt{\frac{\Psi_n \log(n+M)}{n}}$$

Theorem: asyTS (Kandasamy et al. AISTATS 2018)

$$\mathbb{E}[\mathrm{SR}(n)] \lesssim \frac{M\mathrm{polylog}(M)}{n} + \sqrt{\frac{C\Psi_n \log(n+M))}{n}}$$

Theoretical Results – Simple Regret with a Time Budget

Theorem (Informal): TS with M parallel workers

(Kandasamy et al. AISTATS 2018)

- Both synTS and asyTS are better than seqTS.

Theoretical Results – Simple Regret with a Time Budget

Theorem (Informal): TS with M parallel workers

(Kandasamy et al. AISTATS 2018)

- Both synTS and asyTS are better than seqTS.
- If evaluation times are the same, synTS is marginally better than asyTS.

Theoretical Results – Simple Regret with a Time Budget

Theorem (Informal): TS with M parallel workers

(Kandasamy et al. AISTATS 2018)

- Both synTS and asyTS are better than seqTS.
- If evaluation times are the same, synTS is marginally better than asyTS.
- When there is high variability in evaluation times, asyTS is better than synTS.
  - Sub-Gaussian: $\sqrt{\log(M)}$ factor
  - Sub-exponential: $\log(M)$ factor

Dragonfly

(Kandasamy et al. Arxiv 2019)

Code | Issues 1 | Pull requests 1 | Projects 0 | Wiki | Insights | Settings

An open source python library for scalable Bayesian optimisation. Edit

Manage topics

218 commits | 2 branches | 0 releases | 4 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

kirthevasank Updated tree_reg demos — Latest commit a472c07 17 hours ago

bin	Minor bug fixes	10 days ago
dragonfly	Minor updates	5 days ago
examples	Updated tree_reg demos	17 hours ago
.gitignore	Minor bug fixes	6 days ago
.travis.yml	Update TravisCI config to install cython	9 days ago
AUTHORS.txt	Updated readme, setup and authors	8 days ago
LICENSE.txt	added face recognition demo	8 months ago
MANIFEST.in	Added MANIFEST.in	6 days ago
README.md	Minor updates	5 days ago
requirements-dev.txt	Update requirements-dev.txt	9 days ago
requirements.txt	Add six to requirements.txt	2 months ago
run_all_tests.sh	Simplify run_all_tests.sh to use nosetests	10 days ago
setup.py	Minor bug fixes	6 days ago

README.md

Dragonfly

Scalable Bayesian Optimisation

Dragonfly is an open source python library for scalable Bayesian optimisation.

Bayesian optimisation is used for optimising black-box functions whose evaluations are usually expensive. Beyond vanilla optimisation techniques, Dragonfly provides an array of tools to scale up Bayesian optimisation to expensive large scale problems. These include features/functionality that are especially suited for high dimensional optimisation (optimising for a large number of variables), parallel evaluations in synchronous or asynchronous settings (conducting multiple evaluations in parallel), multi-fidelity optimisation (using cheap approximations to speed up the optimisation process), and multi-objective optimisation (optimising multiple functions simultaneously).

Dragonfly is compatible with Python2 (>= 2.7) and Python3 (>= 3.5) and has been tested on Linux, macOS, and

Installation

Set up: We recommend installation via `pip`. In most Linux environments, it can be installed via the commands below, depending on the version.

```
$ sudo apt-get install python-pip    # for Python2
$ sudo apt-get install python3-pip   # for Python3
$ pip install --upgrade pip
```

Alternatively, if you prefer to work in a Python virtual environment, `pip` is automatically available. If so, you need to install the appropriate version in your system. In most Linux environments, this can be done via `sudo apt-get install virtualenv` if you are using Python2, or `sudo apt-get install python3-venv` if you are using Python3. You can also follow the instructions here, here, here, here, or here for Linux, macOS and Windows environments.

The next step is recommended but not required to get started with Dragonfly. Dragonfly uses some Fortran dependencies which require a NumPy compatible Fortran compiler (e.g. gnu95, pg, pathf95) and the `python-dev` package. In most Linux environments, they can be installed via `sudo apt-get install python-dev gfortran` if you are using Python2, or `sudo apt-get install python3-dev gfortran` if you are using Python3. These packages may already be pre-installed in your system. If you are unable to install these packages, then you can still use Dragonfly, but it might be slightly slower.

You can now install Dragonfly via one of the four steps below.

1. Installation via pip (recommended): Installing dragonfly properly requires that numpy is already installed in the current environment. Once that has been done, the library can be installed with pip.

```
$ pip install numpy
$ pip install dragonfly-opt -v
```

2. Installation via source: To install via source, clone the repository and proceed as follows.

```
$ git clone https://github.com/dragonfly/dragonfly.git
$ cd dragonfly
$ pip install -r requirements.txt
$ python setup.py install
```

3. Installing in a Python Virtual Environment: Dragonfly can be pip installed in a python virtualenv, by following the steps below. You can similarly install via source by creating/sourcing the virtualenv and following the steps above.

```
$ virtualenv env        # For Python2
$ python3 -m venv env   # For Python3
$ source env/bin/activate
(env)$ pip install numpy
(env)$ pip install git+https://github.com/dragonfly/dragonfly.git
```

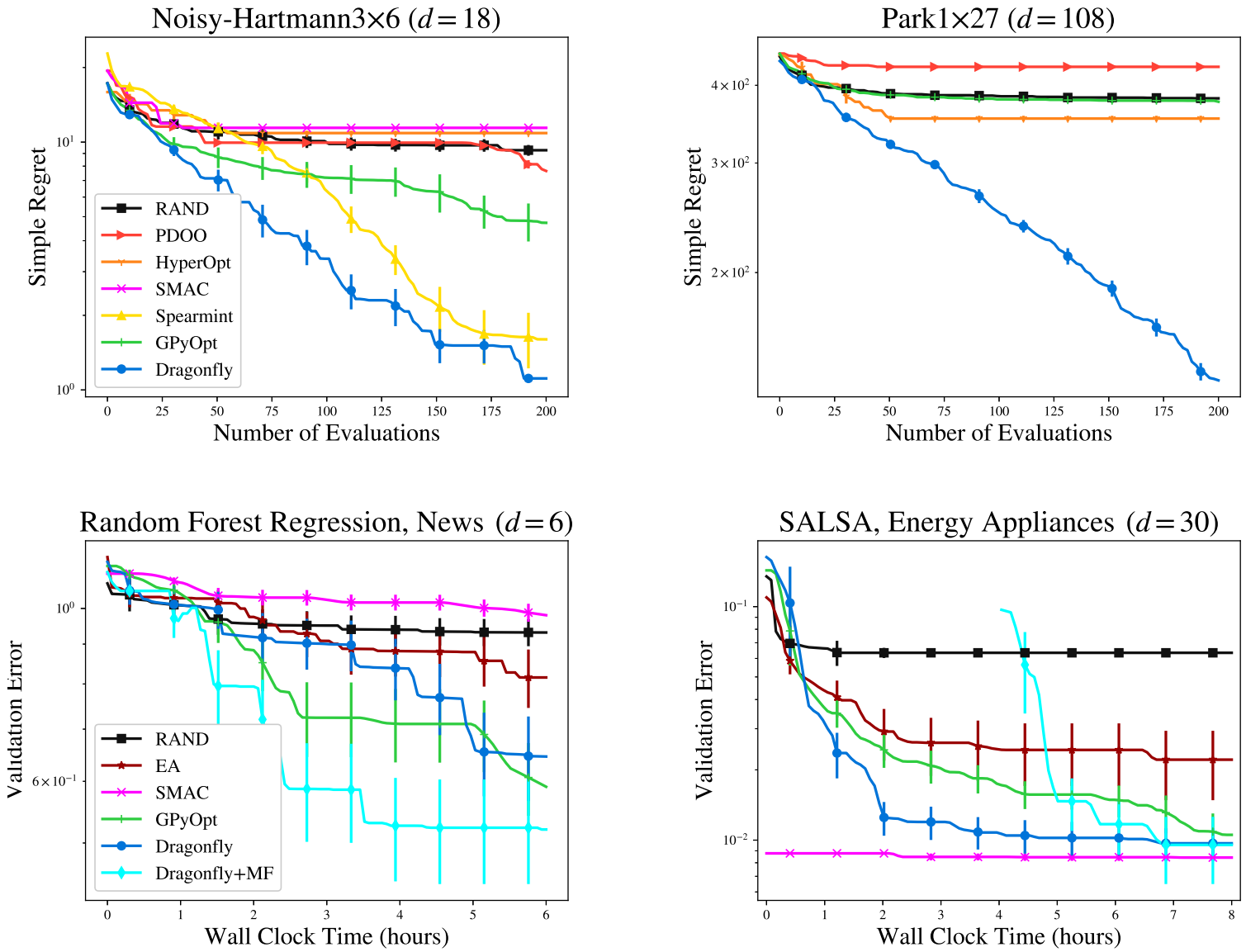
4. Using Dragonfly without Installation: If you prefer to not install Dragonfly in your environment, you can use it by following the steps below.

```
$ git clone https://github.com/dragonfly/dragonfly.git
$ cd dragonfly
$ pip install -r requirements.txt
$ cd dragonfly/utils/direct_fortran
$ bash make_direct.sh
```

dragonfly.github.io

Dragonfly

(Kandasamy et al. Arxiv 2019)



Outline

- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

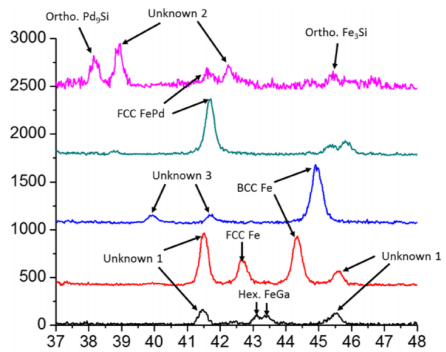
Decision-Making in Stateless Environments without a Reward Signal

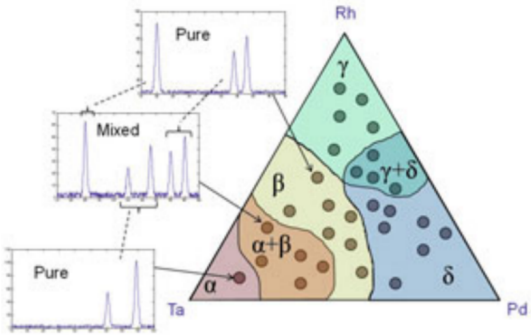
$x \longrightarrow$ **X-Ray Diffraction Studies** $\longrightarrow Y_x$

$\mathcal{X}$: composition of alloy

Diffraction patterns

$\mathcal{Y} \subset \mathbb{R}^K$





Goal: Identify phase transitions in crystal structure in an alloy.

A Framework for Goal Oriented Design of Experiments

(Kandasamy et al. ICML 2019)

- MPS: A myopic algorithm inspired by Thompson sampling.

A Framework for Goal Oriented Design of Experiments

(Kandasamy et al. ICML 2019)

- MPS: A myopic algorithm inspired by Thompson sampling.

- Achieve a broad range of application-specific DoE objectives.
- Flexibility to incorporate domain expertise.

A Framework for Goal Oriented Design of Experiments

(Kandasamy et al. ICML 2019)

- MPS: A myopic algorithm inspired by Thompson sampling.

- Achieve a broad range of application-specific DoE objectives.
- Flexibility to incorporate domain expertise.
- Regret bounds with a globally optimal oracle policy.
  - Proof ideas from submodular optimisation and reinforcement learning.

Outline

- Review: Bandits in the Bayesian Setting
- Scaling up Bandits
  1. Multi-fidelity Bandits
  2. Bandits on Graph Structured Domains
  3. High Dimensional Bandits
  4. Parallel Bandits
  5. Beyond Bandits: Adaptive Decision-making in Stateless Environments
- Conclusions, Applications in Systems, etc.

Autonomous Decision-making in Computer Systems

- Systematic noise/variations: performance depends on several extraneous factors (e.g. hardware, resource contention).

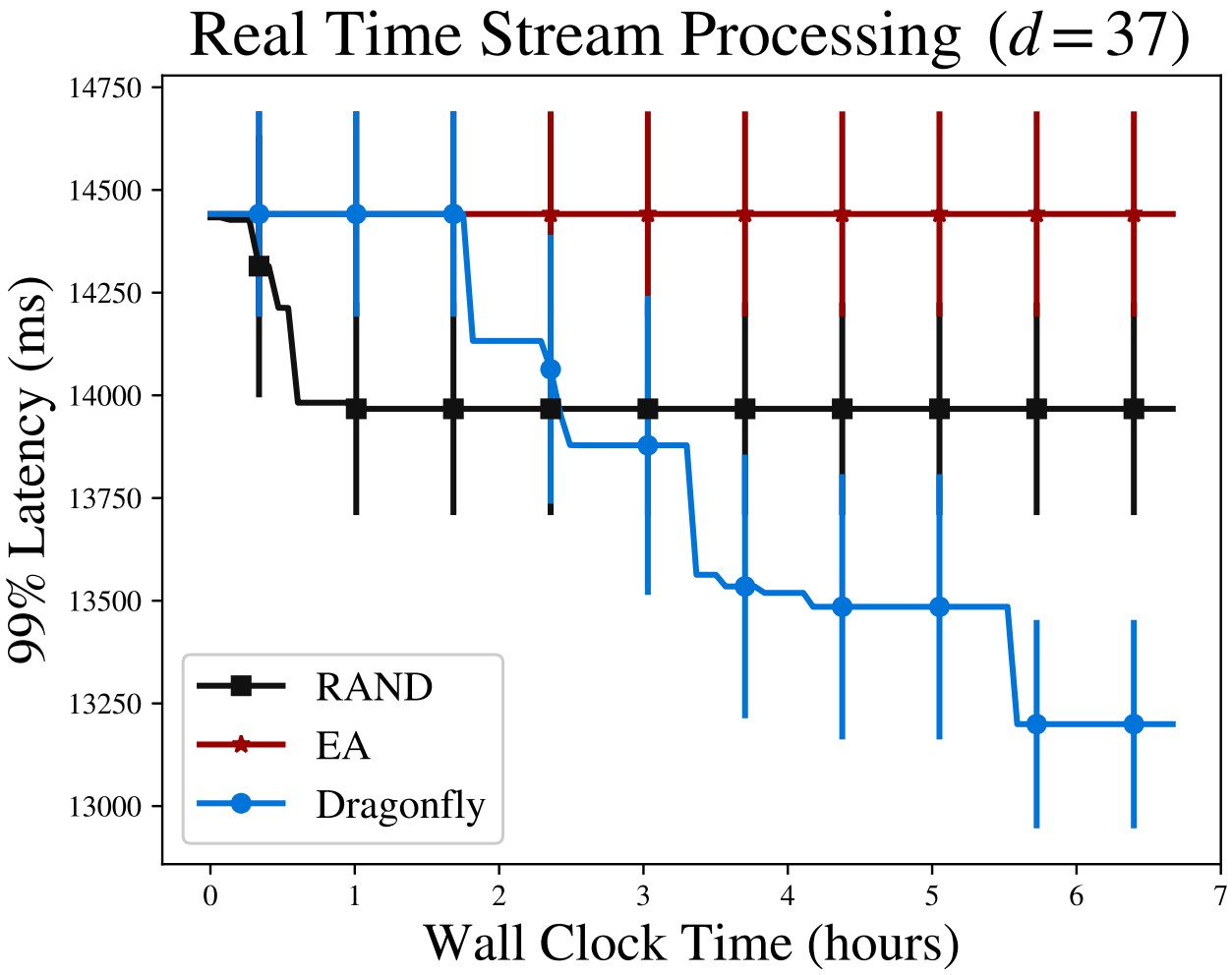
Autonomous Decision-making in Computer Systems

- Systematic noise/variations: performance depends on several extraneous factors (e.g. hardware, resource contention).
- Performance of a configuration varies depending on the workload, which may be completely or partially unknown.

Autonomous Decision-making in Computer Systems

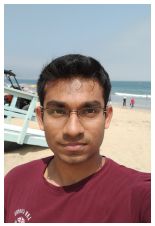
- Systematic noise/variations: performance depends on several extraneous factors (e.g. hardware, resource contention).
- Performance of a configuration varies depending on the workload, which may be completely or partially unknown.
- Systems in production can only be changed gradually.

Tuning Spark Configurations in YSB



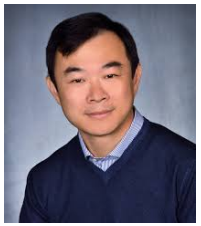

Akshay


Barnabás


Biswajit


Chris


Chun-liang

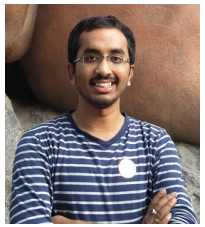

Eric


Gautam


Hai

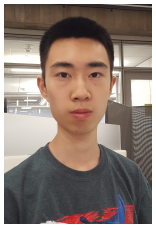

Jeff

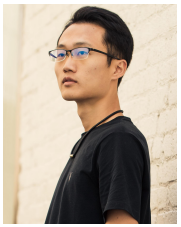

Junier


Karun


Ksenia


Rajat


Reed


Sailun


Sanjay


Willie


Yusha

Thank You

Summary

- **Multi-fidelity bandits:** Use cheap approximations to a an expensive experiment to speed up optimisation.
  (Kandasamy et al. NeurIPS 2016a, Kandasamy et al. NeurIPS 2016b, Kandasamy et al. ICML 2017)
- **Bandits on graph-domains** Neural Architecture Search.
  (Kandasamy et al. NeurIPS 2018)
- **High dimensional bandits:** Additive models have favourable statistical and computational properties.
  (Kandasamy et al. ICML 2015)
- **Parallel bandits:** Conduct multiple simultaneous evaluations.
  (Kandasamy et al. AISTATS 2018)
- **Goal Oriented Design of Experiments:** A myopic strategy inspired by Thompson sampling.
  (Kandasamy et al. ICML 2019)