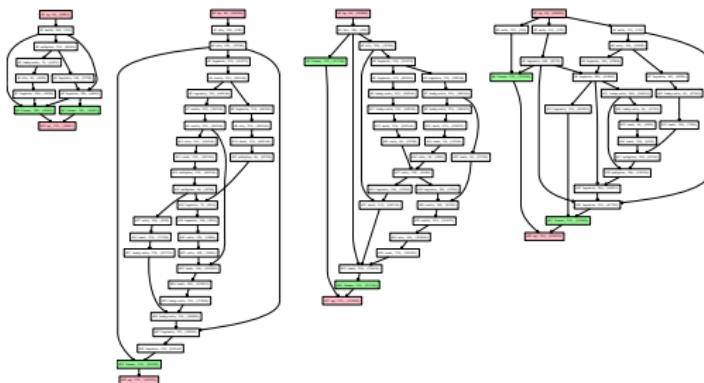


Neural Architecture Search with Bayesian Optimisation and Optimal Transport

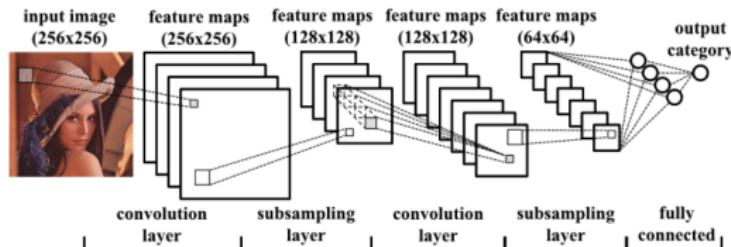
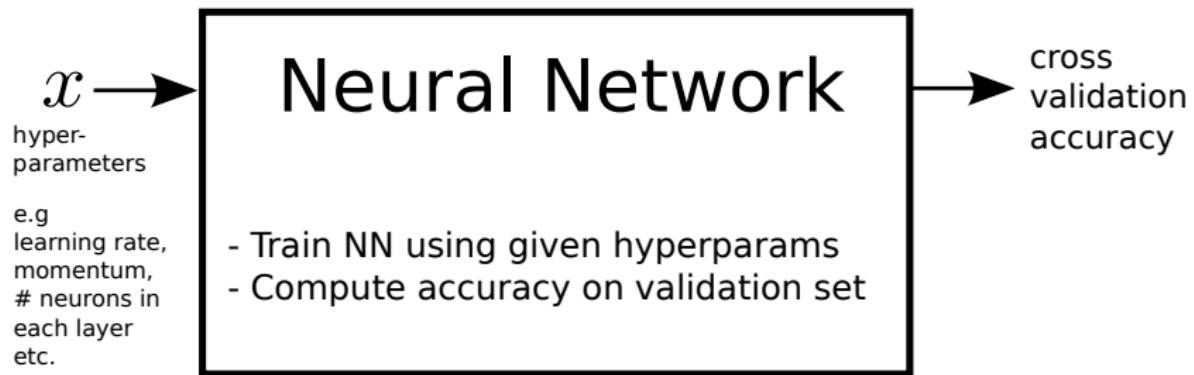


Kirthevasan Kandasamy
Carnegie Mellon University

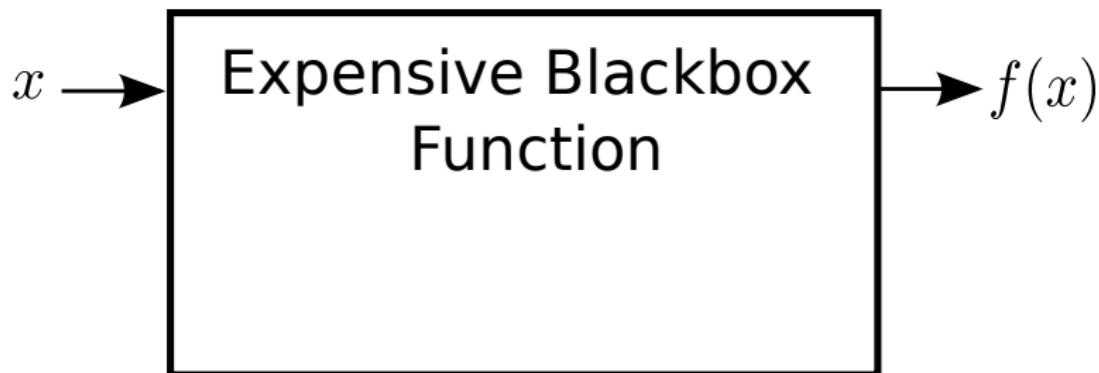
Nov 2, 2018
Uber AI Labs, CA

slides: www.cs.cmu.edu/~kkandas

Model Selection & Hyperparameter Tuning

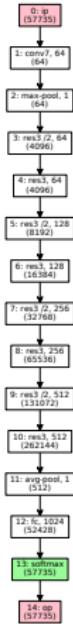


Model Selection is an Optimisation Problem



Many methods for optimising expensive zeroth order functions.
E.g. Bayesian Optimisation

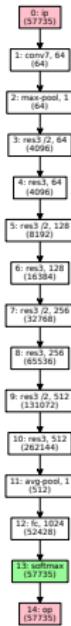
Neural Architecture Search



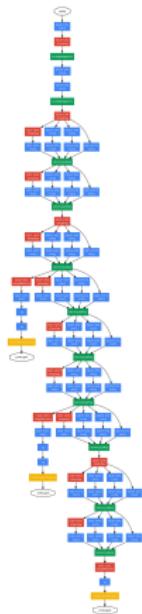
Feedforward
network

Neural Architecture Search

Feedforward
network



GoogLeNet
(Szegedy et
al. 2015)

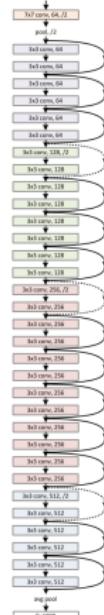
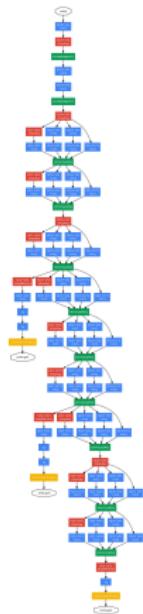


Neural Architecture Search

Feedforward
network



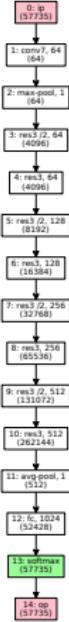
GoogLeNet
(Szegedy et
al. 2015)



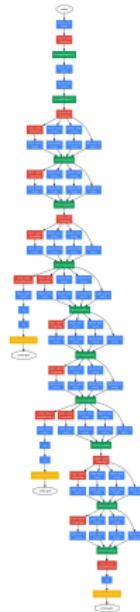
ResNet
(He et al.
2016)

Neural Architecture Search

Feedforward
network



GoogLeNet
(Szegedy et
al. 2015)

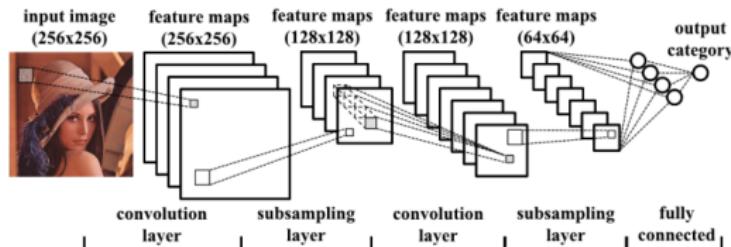
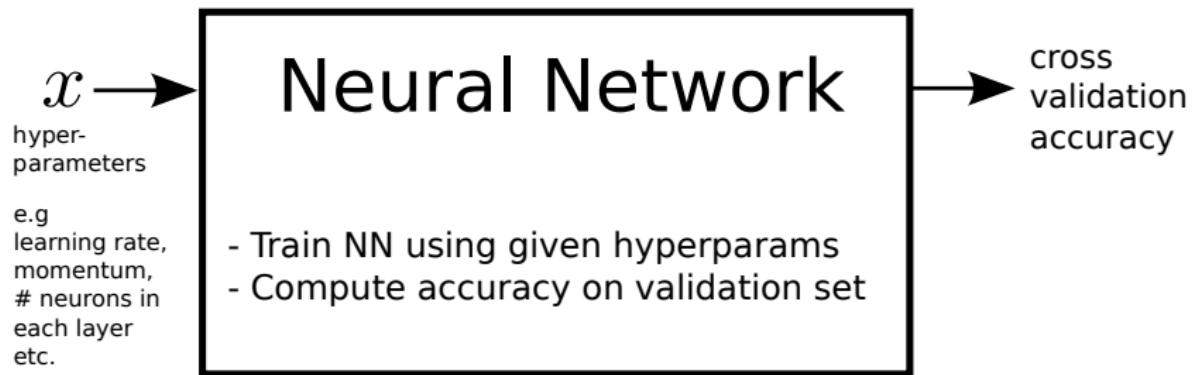


ResNet
(He et al.
2016)

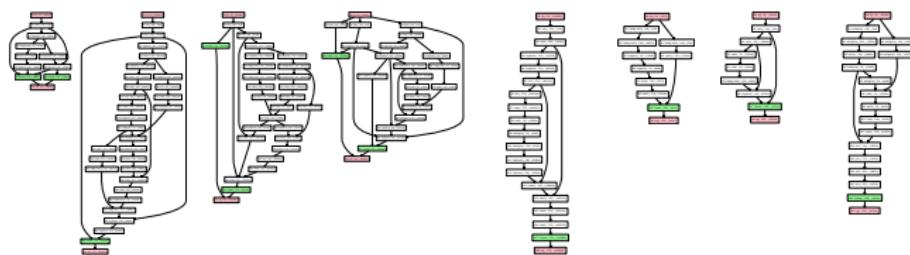
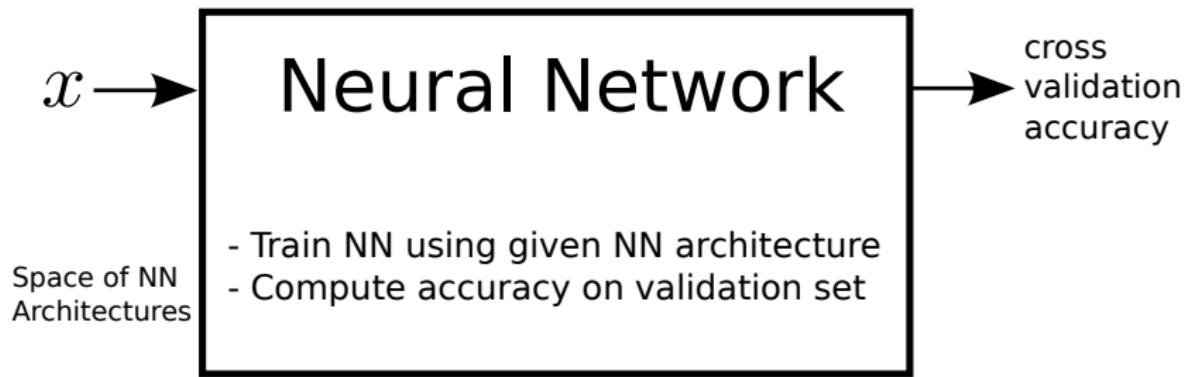


DenseNet
(Huang et
al. 2017)

Tuning Scalar/Categorical Hyperparameters



Neural Architecture Search



Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

RL is more difficult than optimisation (Jiang et al. 2016).

Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

RL is more difficult than optimisation (Jiang et al. 2016).

Based on Evolutionary Algorithms:

(Kitano 1990, Stanley & Miikkulainen 2002, Floreano et al. 2008, Liu et al. 2017,
Miikkulainen et al. 2017, Real et al. 2017, Xie & Yuille 2017)

Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

RL is more difficult than optimisation (Jiang et al. 2016).

Based on Evolutionary Algorithms:

(Kitano 1990, Stanley & Miikkulainen 2002, Floreano et al. 2008, Liu et al. 2017,
Miikkulainen et al. 2017, Real et al. 2017, Xie & Yuille 2017)

EA works well for optimising cheap functions, but not when
function evaluations are expensive.

Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

RL is more difficult than optimisation (Jiang et al. 2016).

Based on Evolutionary Algorithms:

(Kitano 1990, Stanley & Miikkulainen 2002, Floreano et al. 2008, Liu et al. 2017, Miikkulainen et al. 2017, Real et al. 2017, Xie & Yuille 2017)

EA works well for optimising cheap functions, but not when function evaluations are expensive.

Other (including BO):

(Swersky et al. 2014, Cortes et al. 2016, Mendoza et al. 2016, Negrinho & Gordon 2017, Jenatton et al. 2017)

Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

RL is more difficult than optimisation (Jiang et al. 2016).

Based on Evolutionary Algorithms:

(Kitano 1990, Stanley & Miikkulainen 2002, Floreano et al. 2008, Liu et al. 2017, Miikkulainen et al. 2017, Real et al. 2017, Xie & Yuille 2017)

EA works well for optimising cheap functions, but not when function evaluations are expensive.

Other (including BO):

(Swersky et al. 2014, Cortes et al. 2016, Mendoza et al. 2016, Negrinho & Gordon 2017, Jenatton et al. 2017)

Mostly search among feed-forward structures.

Neural Architecture Search – Prior Work

Based on Reinforcement Learning:

(Baker et al. 2016, Zhong et al. 2017, Zoph & Le 2017, Zoph et al. 2017)

RL is more difficult than optimisation (Jiang et al. 2016).

Based on Evolutionary Algorithms:

(Kitano 1990, Stanley & Miikkulainen 2002, Floreano et al. 2008, Liu et al. 2017, Miikkulainen et al. 2017, Real et al. 2017, Xie & Yuille 2017)

EA works well for optimising cheap functions, but not when function evaluations are expensive.

Other (including BO):

(Swersky et al. 2014, Cortes et al. 2016, Mendoza et al. 2016, Negrinho & Gordon 2017, Jenatton et al. 2017)

Mostly search among feed-forward structures.

And a few more in the last two years ...

Outline

1. Review
 - ▶ Bayesian optimisation
 - ▶ Optimal transport
2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport
 - ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
 - ▶ Optimising the acquisition via an evolutionary algorithm
 - ▶ Experiments
3. Multi-fidelity optimisation in NASBOT

Outline

1. Review
 - ▶ Bayesian optimisation
 - ▶ Optimal transport
2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport
 - ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
 - ▶ Optimising the acquisition via an evolutionary algorithm
 - ▶ Experiments
3. Multi-fidelity optimisation in NASBOT

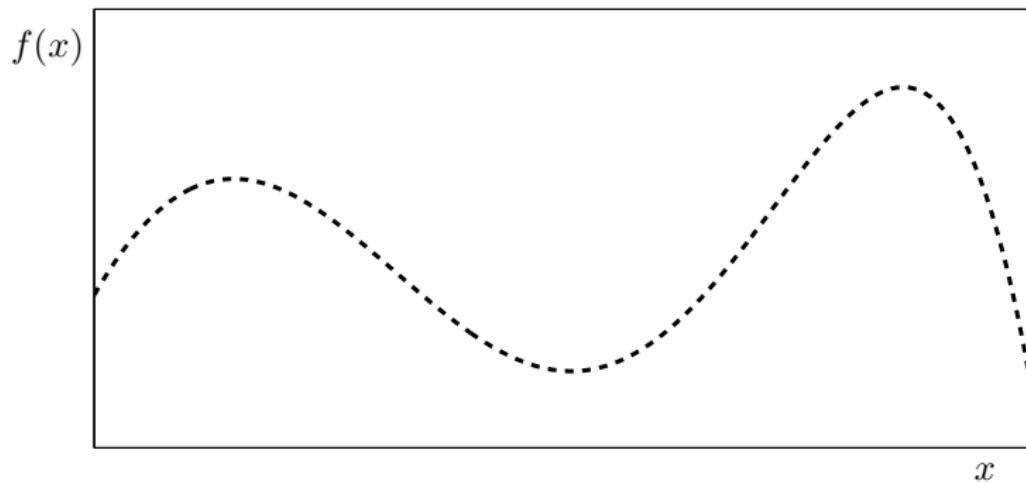
Gaussian Processes (\mathcal{GP})

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

Gaussian Processes (\mathcal{GP})

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

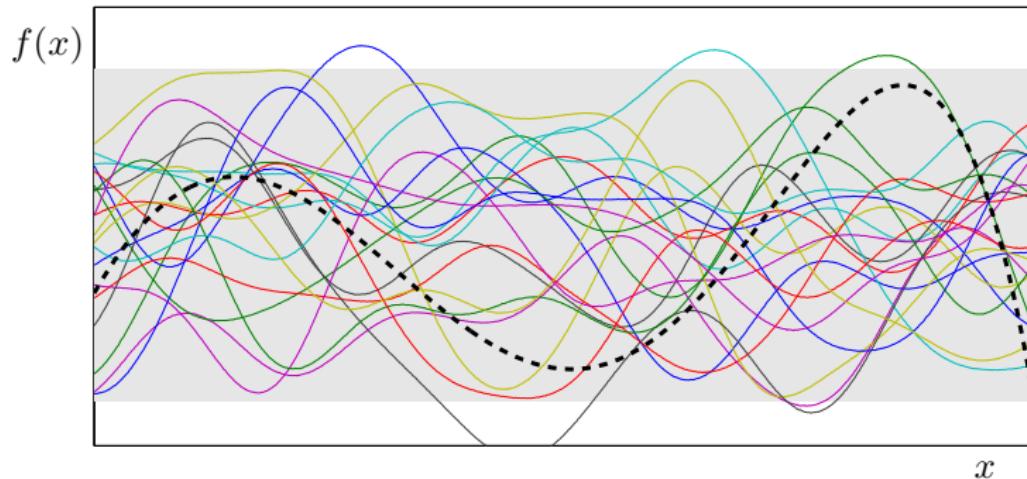
Functions with no observations



Gaussian Processes (\mathcal{GP})

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

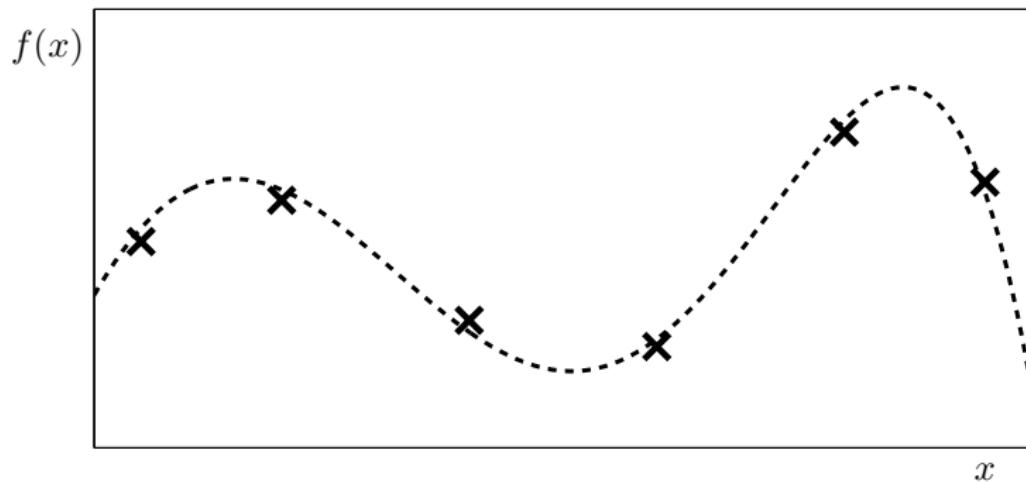
Prior \mathcal{GP}



Gaussian Processes (\mathcal{GP})

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

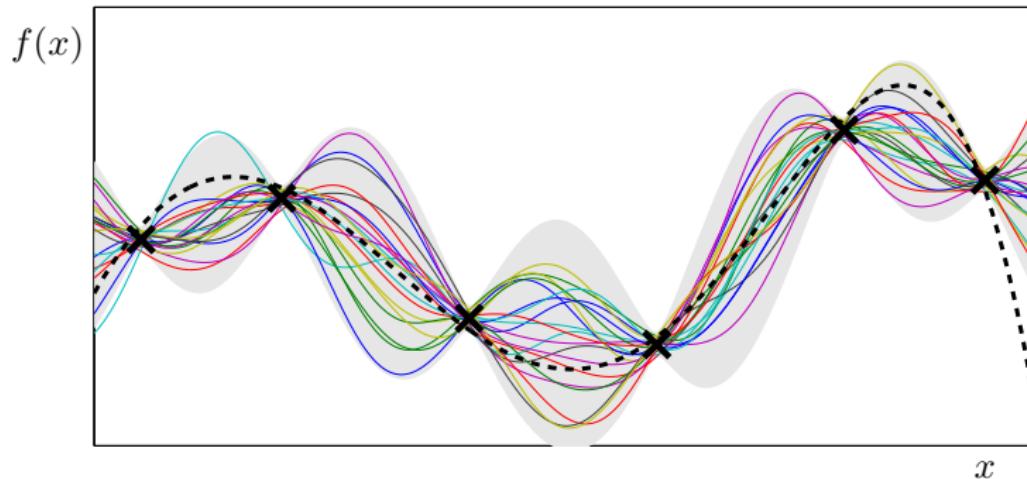
Observations



Gaussian Processes (\mathcal{GP})

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

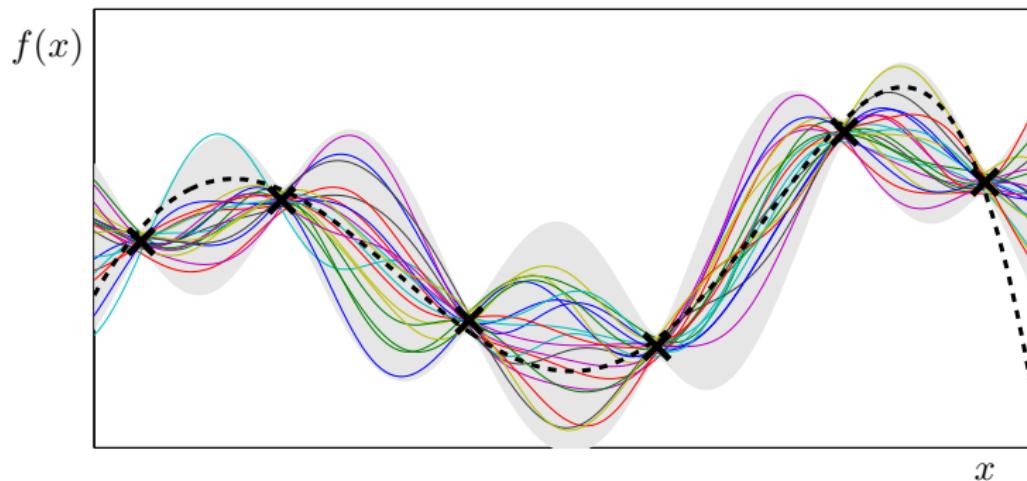
Posterior \mathcal{GP} given observations



Gaussian Processes (\mathcal{GP})

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

Posterior \mathcal{GP} given observations



Completely characterised by mean function $\mu : \mathcal{X} \rightarrow \mathbb{R}$, and covariance kernel $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

After t observations, $f(x) \sim \mathcal{N}(\mu_t(x), \sigma_t^2(x))$.

On the Kernel κ

a.k.a covariance function, covariance kernel, covariance

$\kappa(x, x')$: covariance between *random variables* $f(x)$ and $f(x')$.

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

On the Kernel κ

a.k.a covariance function, covariance kernel, covariance

$\kappa(x, x')$: covariance between *random variables* $f(x)$ and $f(x')$.

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

Some examples in Euclidean spaces

$$\kappa(x, x') = \exp(-\beta d(x, x'))$$

$$\kappa(x, x') = \exp(-\beta d(x, x')^2)$$

$d \leftarrow$ distance between two points.

E.g., $d(x, x') = \|x - x'\|_1$, or $d(x, x') = \|x - x'\|_2$.

On the Kernel κ

a.k.a covariance function, covariance kernel, covariance

$\kappa(x, x')$: covariance between *random variables* $f(x)$ and $f(x')$.

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

Some examples in Euclidean spaces

$$\kappa(x, x') = \exp(-\beta d(x, x'))$$

$$\kappa(x, x') = \exp(-\beta d(x, x')^2)$$

$d \leftarrow$ distance between two points.

E.g., $d(x, x') = \|x - x'\|_1$, or $d(x, x') = \|x - x'\|_2$.

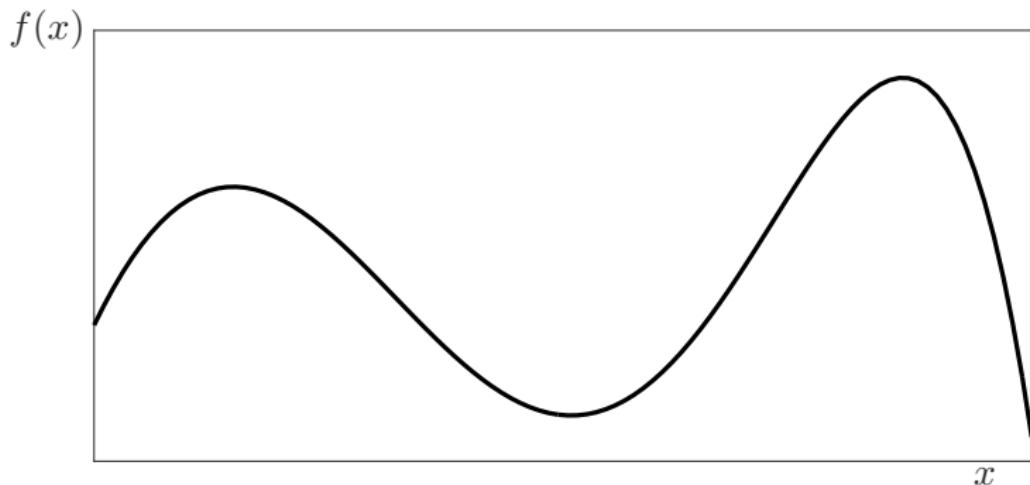
GP Posterior:

$$\mu_t(x) = \kappa(x, X_t)^\top (\kappa(X_t, X_t) + \eta^2 I)^{-1} Y$$

$$\sigma_t^2(x) = \kappa(x, x) - \kappa(x, X_t)^\top (\kappa(X_t, X_t) + \eta^2 I)^{-1} \kappa(X_t, x).$$

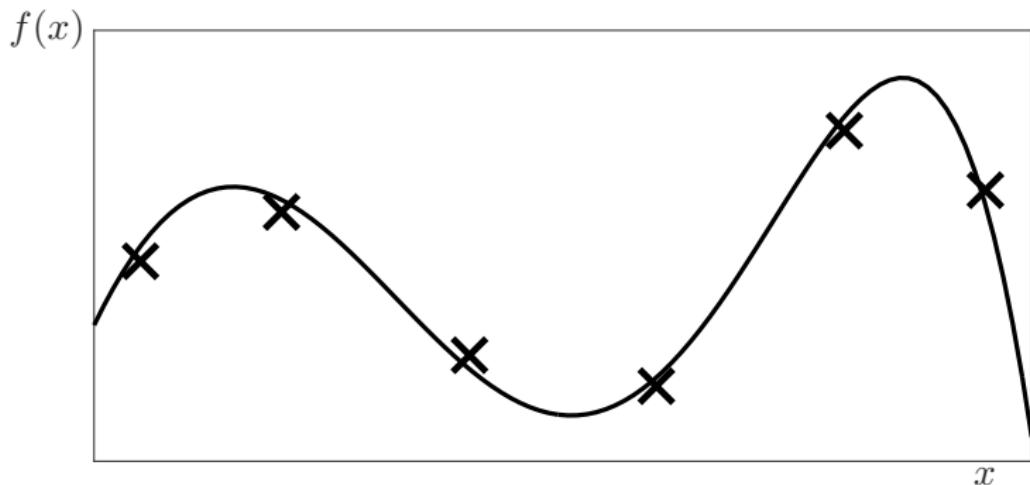
Bayesian Optimisation

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



Bayesian Optimisation

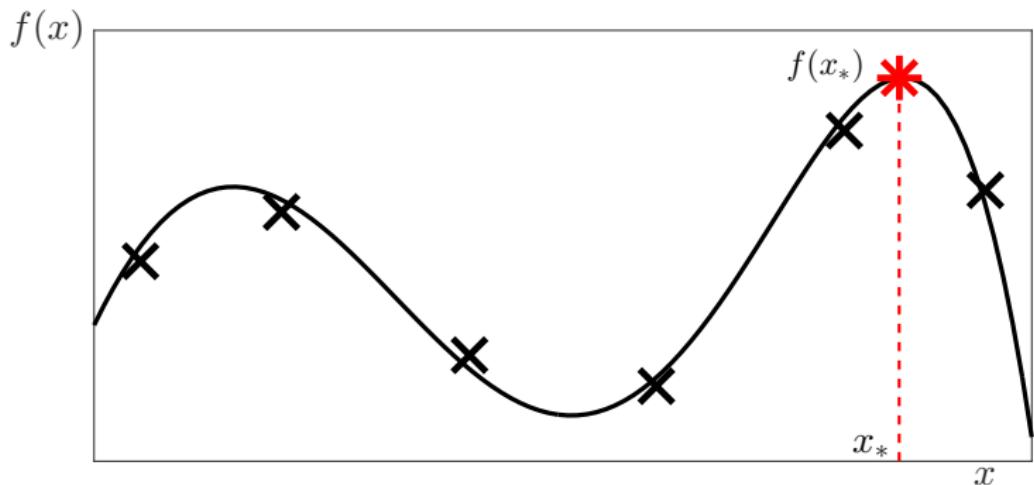
$f : \mathcal{X} \rightarrow \mathbb{R}$ is an expensive black-box function, accessible only via noisy evaluations.



Bayesian Optimisation

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

Let $x_* = \operatorname{argmax}_x f(x)$.

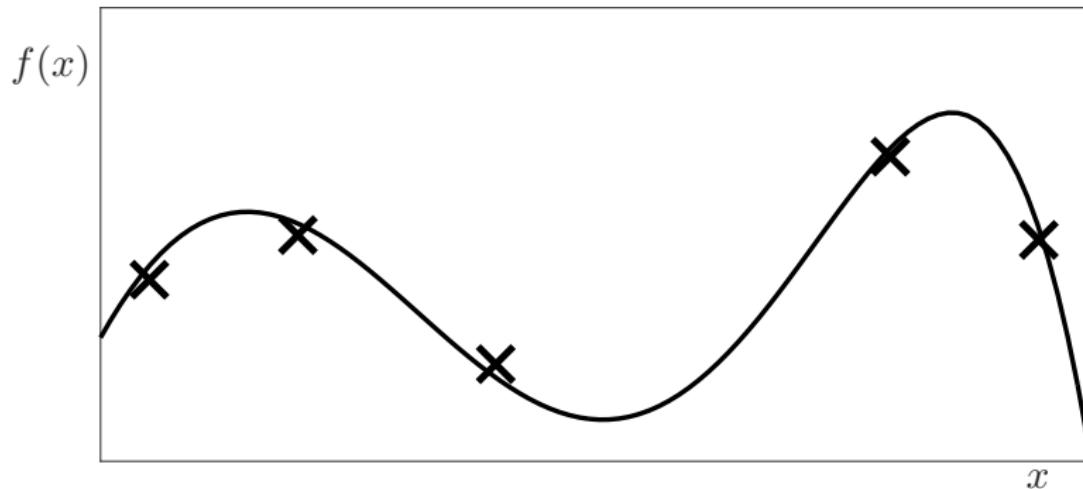


Algorithm 1: Upper Confidence Bounds for BO

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

Gaussian Process Upper Confidence Bound (GP-UCB)

(Srinivas et al. 2010)

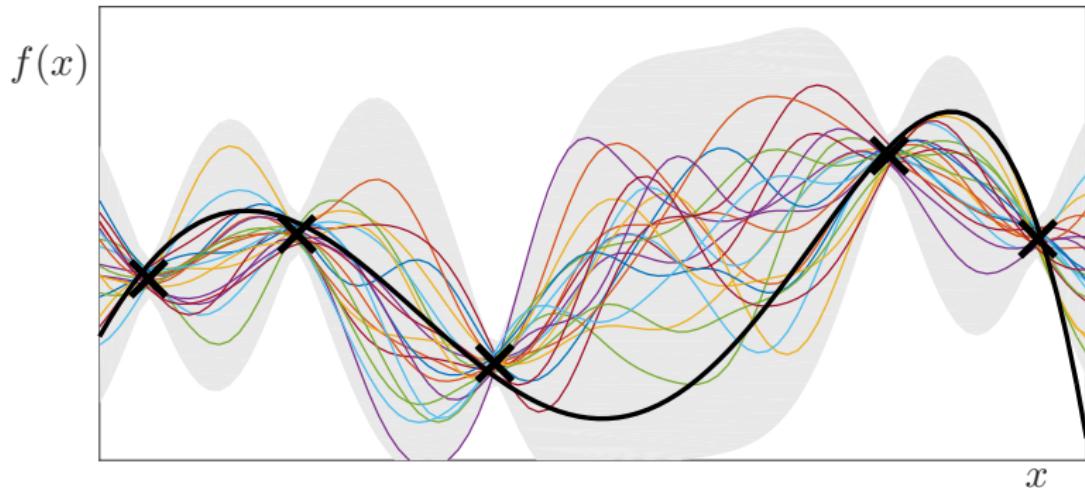


Algorithm 1: Upper Confidence Bounds for BO

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

Gaussian Process Upper Confidence Bound (GP-UCB)

(Srinivas et al. 2010)



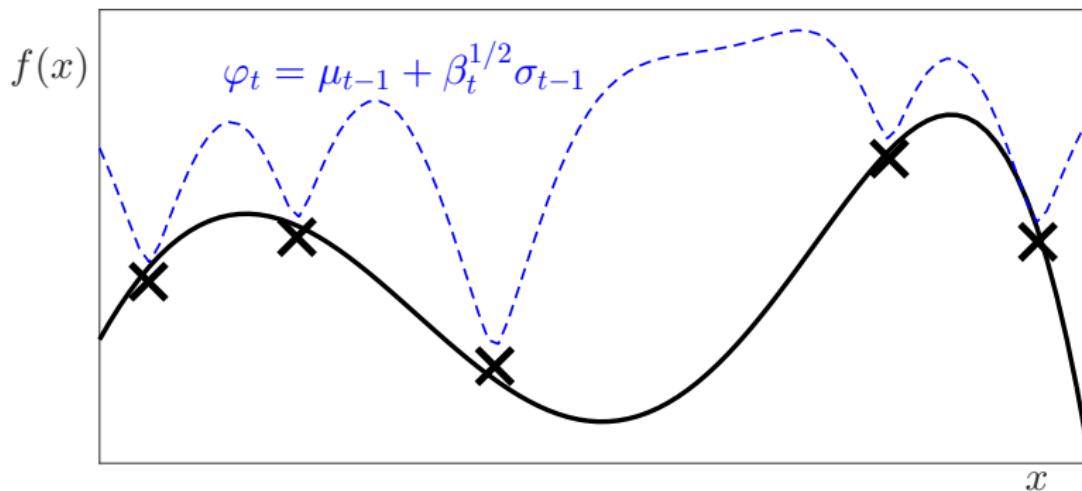
- 1) Compute posterior \mathcal{GP} .

Algorithm 1: Upper Confidence Bounds for BO

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

Gaussian Process Upper Confidence Bound (GP-UCB)

(Srinivas et al. 2010)



1) Compute posterior \mathcal{GP} .

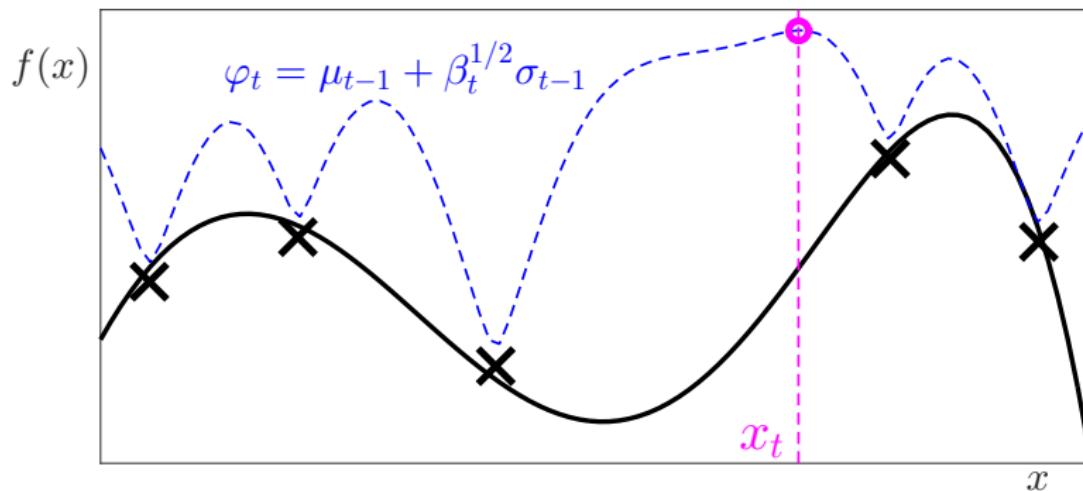
2) Construct UCB φ_t .

Algorithm 1: Upper Confidence Bounds for BO

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

Gaussian Process Upper Confidence Bound (GP-UCB)

(Srinivas et al. 2010)



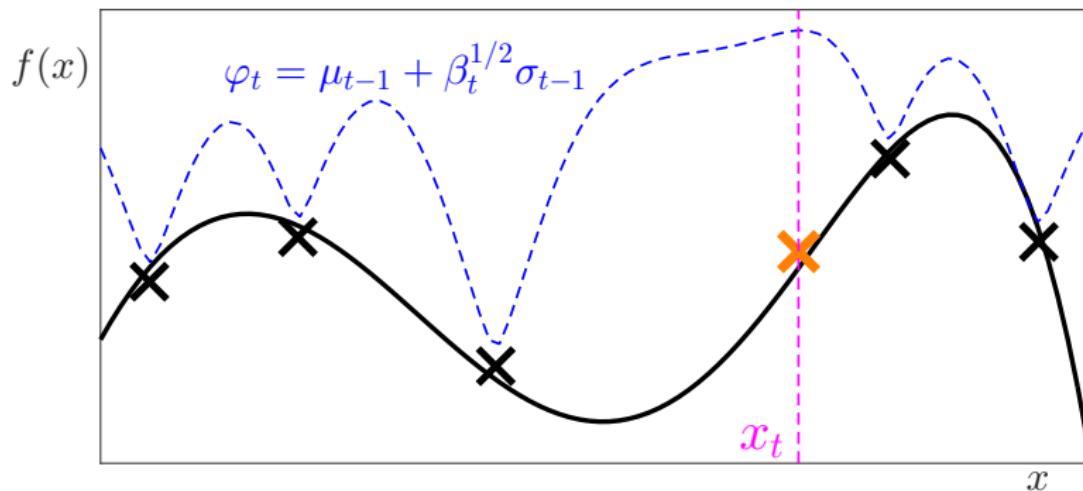
- 1) Compute posterior \mathcal{GP} .
- 2) Construct UCB φ_t .
- 3) Choose $x_t = \operatorname{argmax}_x \varphi_t(x)$.

Algorithm 1: Upper Confidence Bounds for BO

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

Gaussian Process Upper Confidence Bound (GP-UCB)

(Srinivas et al. 2010)

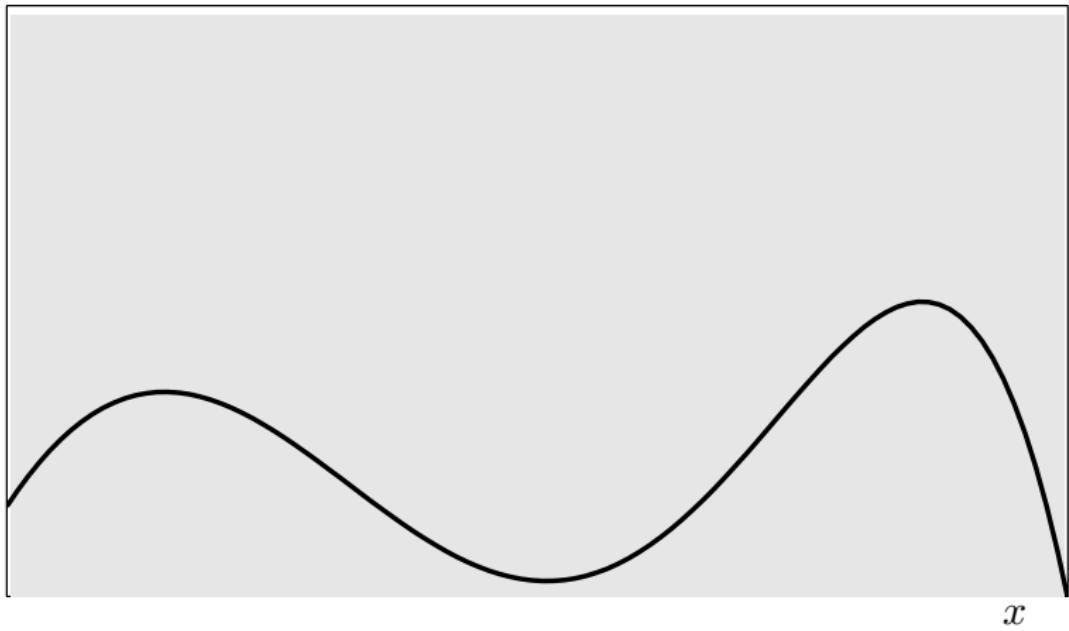


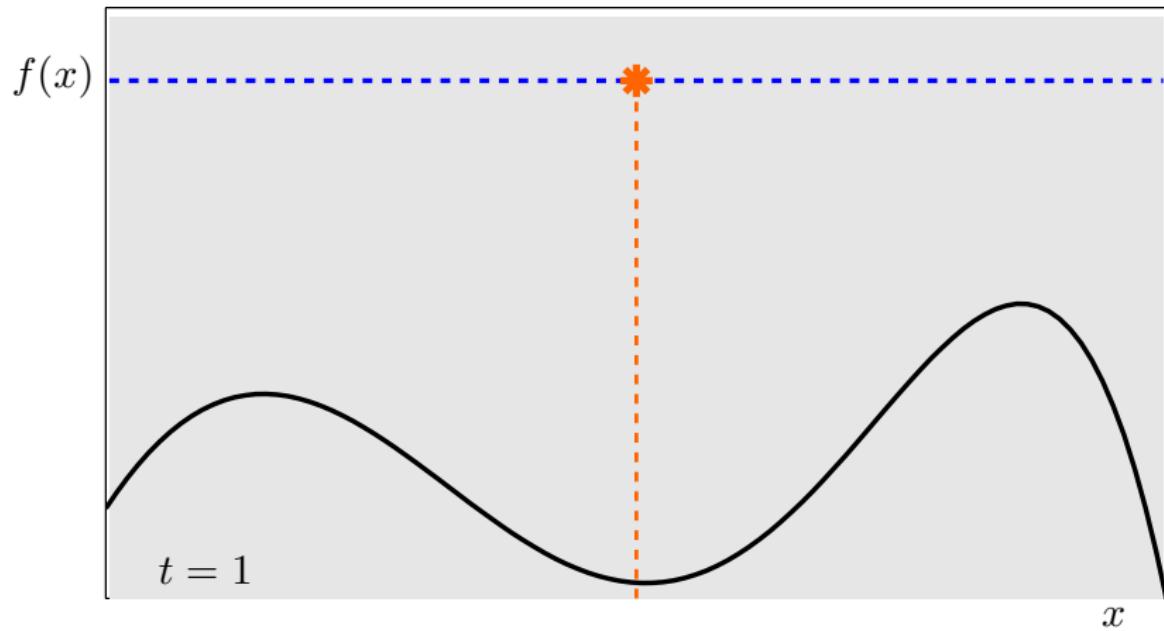
- 1) Compute posterior \mathcal{GP} .
- 2) Construct UCB φ_t .
- 3) Choose $x_t = \operatorname{argmax}_x \varphi_t(x)$.
- 4) Evaluate f at x_t .

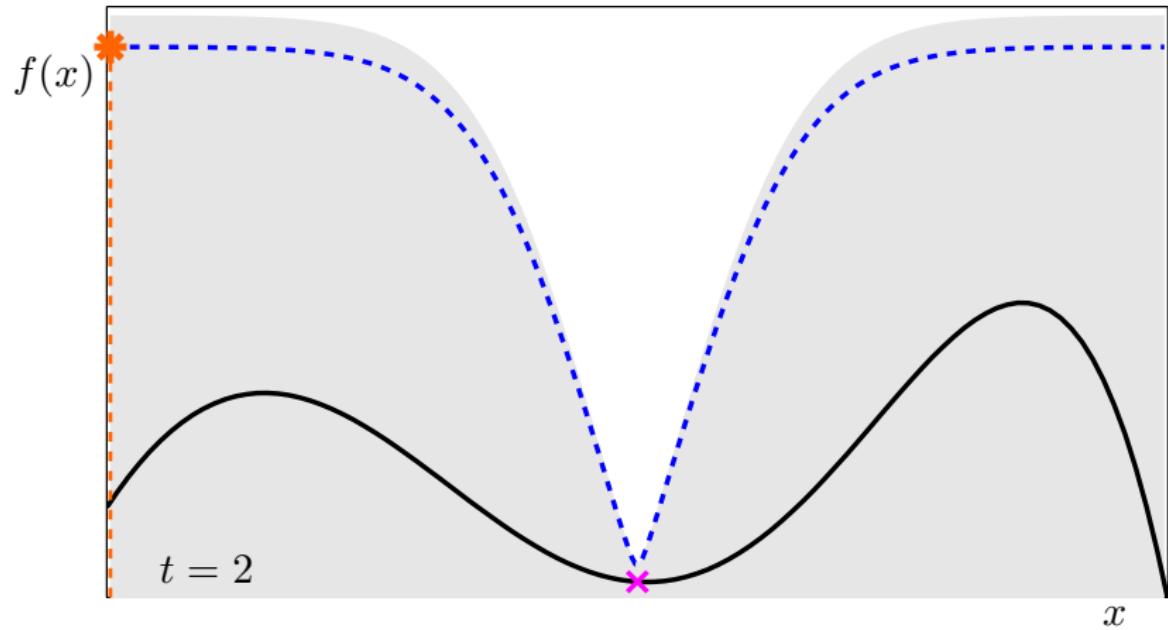
GP-UCB

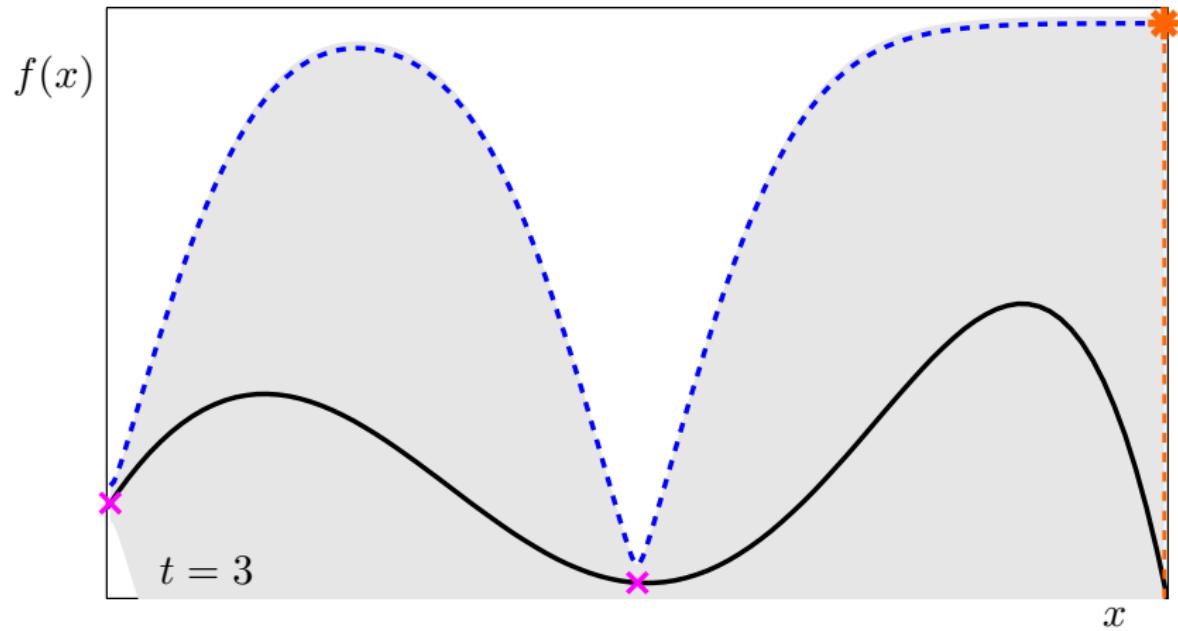
(Srinivas et al. 2010)

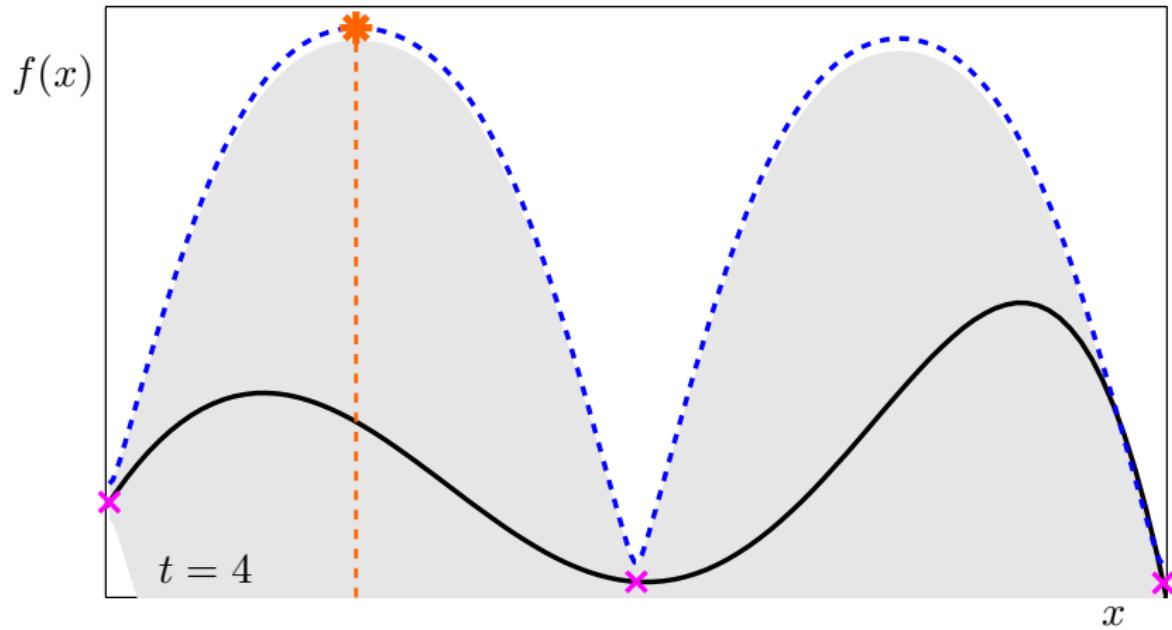
$$f(x)$$

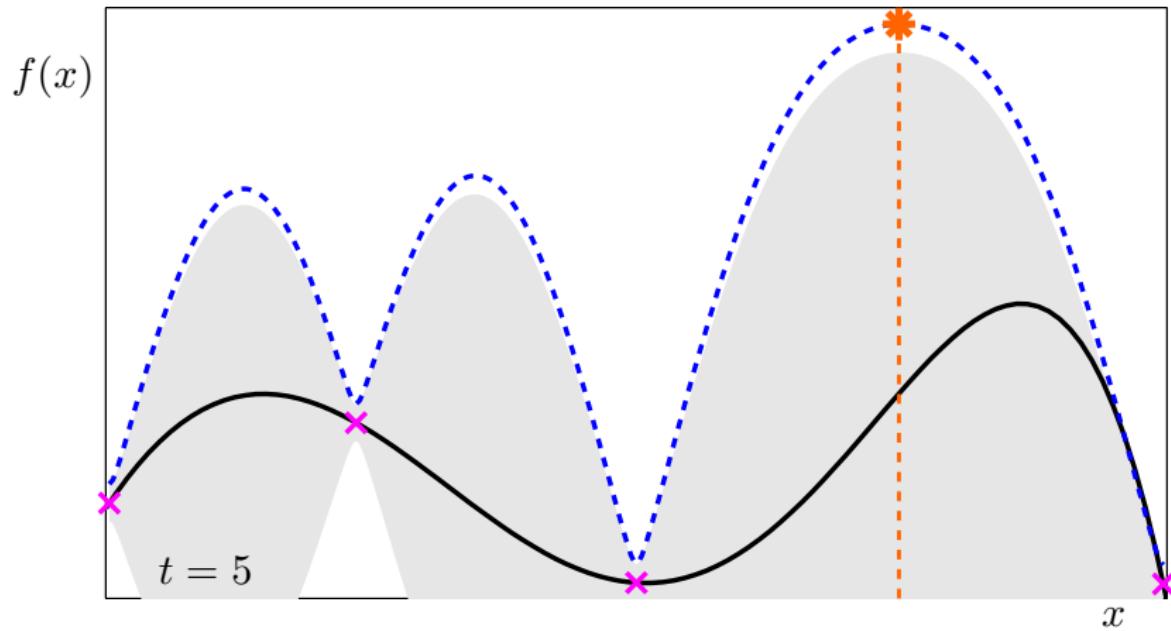


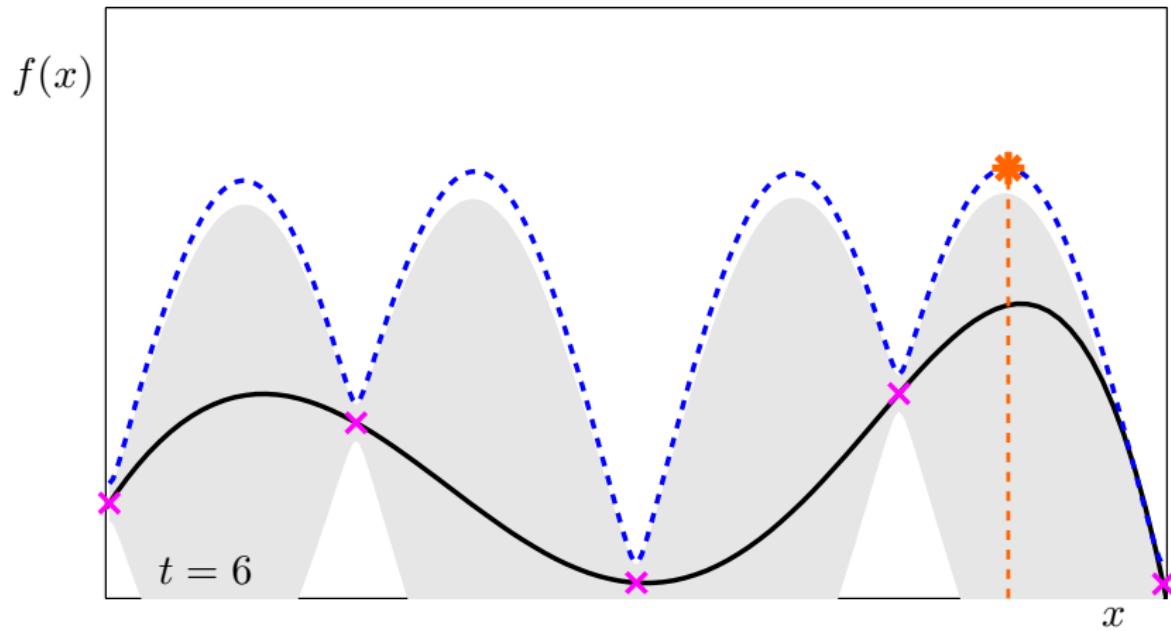


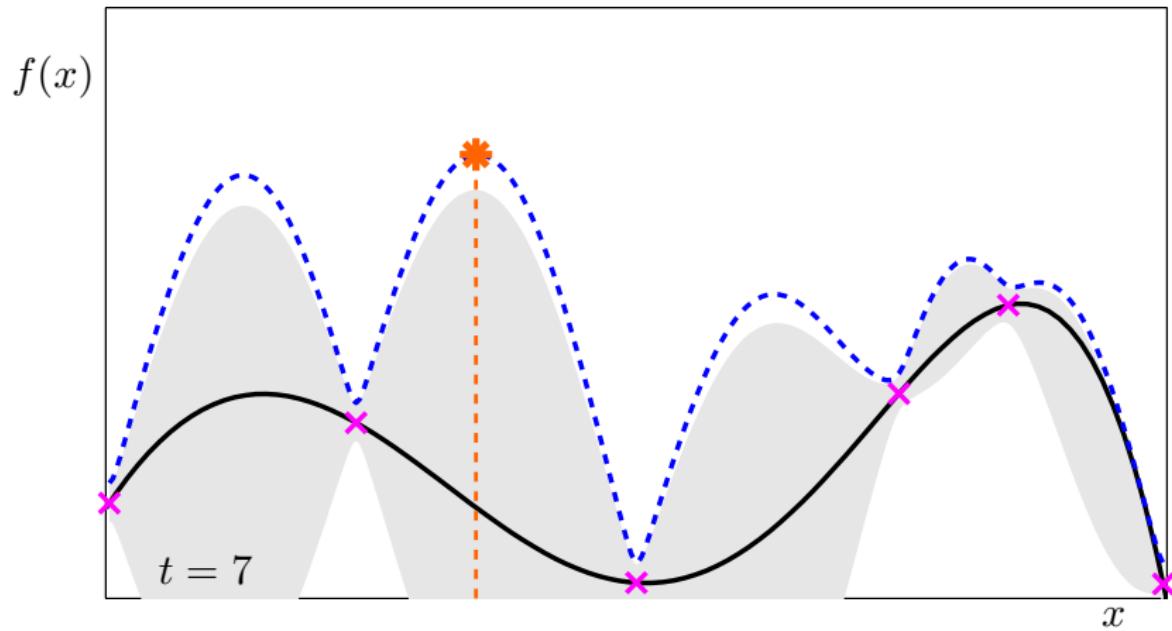






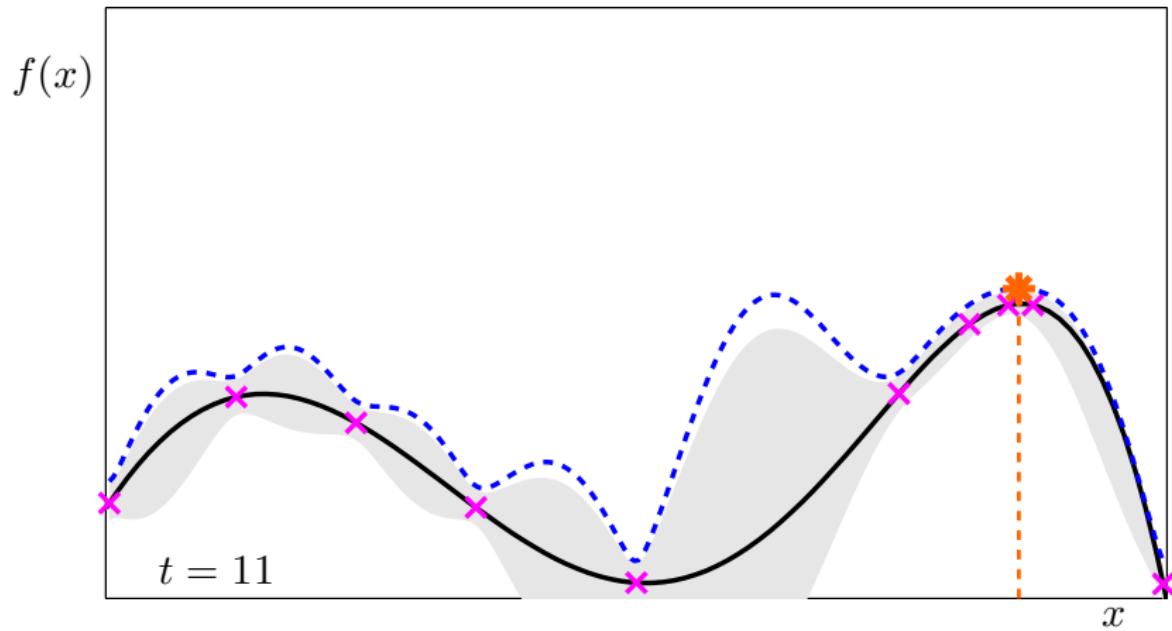






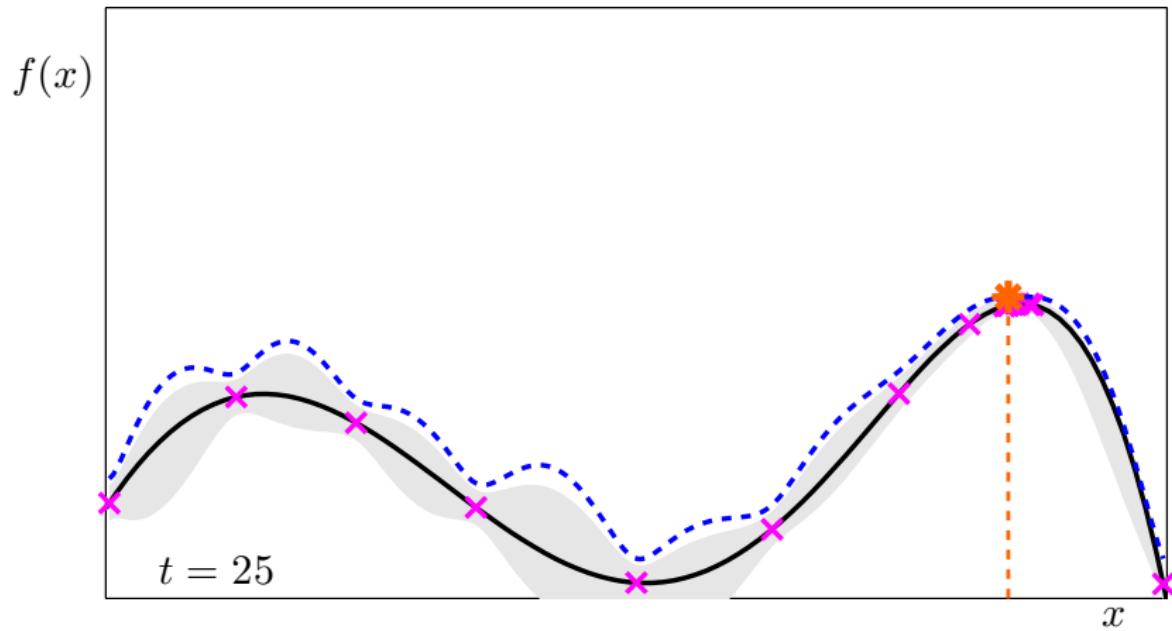
GP-UCB

(Srinivas et al. 2010)



GP-UCB

(Srinivas et al. 2010)



Theory

For BO with UCB

(Srinivas et al. 2010, Russo & van Roy 2014)

$$\mathbb{E} \left[f(x_*) - \max_{t=1,\dots,n} f(x_t) \right] \lesssim \sqrt{\frac{\Psi_n(\mathcal{X}) \log(n)}{n}}$$

$\Psi_n \leftarrow$ Maximum information gain

GP with SE Kernel in d dimensions, $\Psi_n(\mathcal{X}) \asymp \text{vol}(\mathcal{X}) \log(n)^d$.

Bayesian Optimisation

Other criteria for selecting x_t :

- ▶ Expected improvement (Jones et al. 1998)
- ▶ Thompson Sampling (Thompson 1933)
- ▶ 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)
- ▶ Thompson Sampling (Thompson 1933)
- ▶ 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)

Outline

1. Review

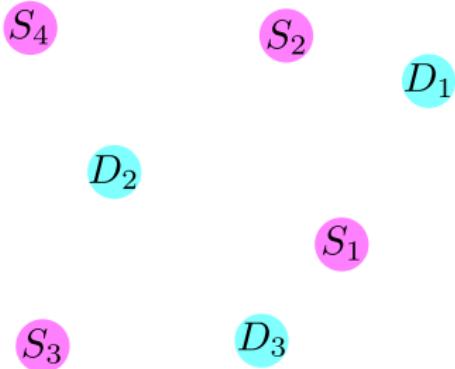
- ▶ Bayesian optimisation
- ▶ Optimal transport

2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport

- ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
- ▶ Optimising the acquisition via an evolutionary algorithm
- ▶ Experiments

3. Multi-fidelity optimisation in NASBOT

Optimal Transport



Optimal Transport

$S_4^{s_4}$

$S_2^{s_2}$

$D_1^{d_1}$

$D_2^{d_2}$

$S_1^{s_1}$

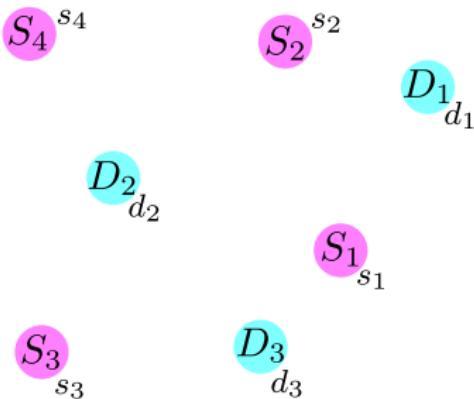
$S_3^{s_3}$

$D_3^{d_3}$

total supply = total demand

$$\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$$

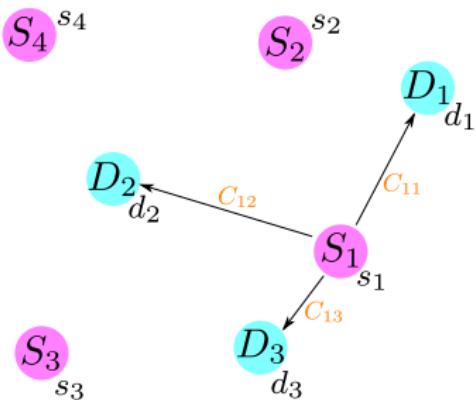
Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

	D_1	D_2	D_3
S_1			
S_2			
S_3			
S_4			

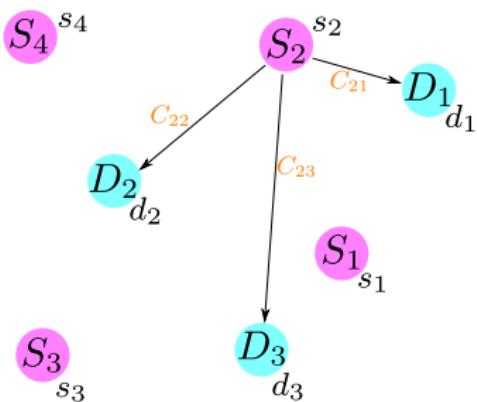
Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

	D_1	D_2	D_3
S_1	C_{11}	C_{12}	C_{13}
S_2			
S_3			
S_4			

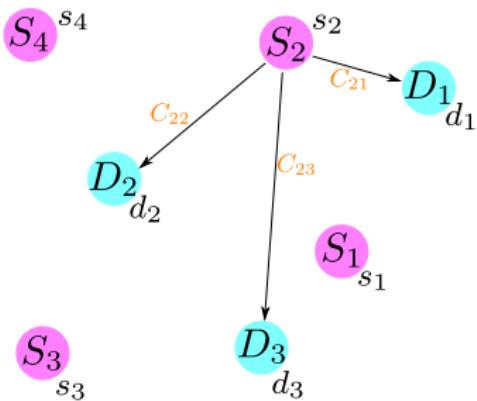
Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

	D_1	D_2	D_3
S_1	C_{11}	C_{12}	C_{13}
S_2	C_{21}	C_{22}	C_{23}
S_3			
S_4			

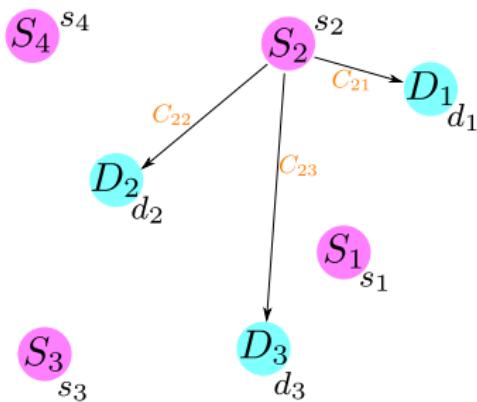
Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

	D_1	D_2	D_3
S_1	C_{11}	C_{12}	C_{13}
S_2	C_{21}	C_{22}	C_{23}
S_3	C_{31}	C_{32}	C_{33}
S_4	C_{41}	C_{42}	C_{43}

Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

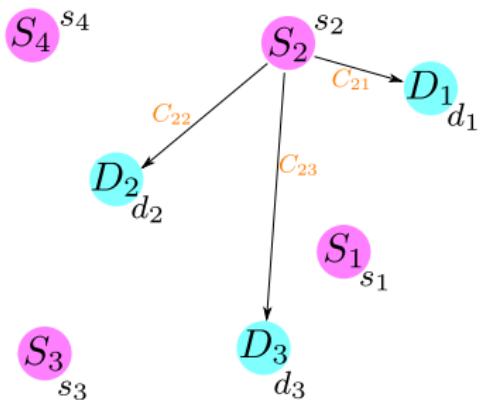
	D_1	D_2	D_3
S_1	C_{11}	C_{12}	C_{13}
S_2	C_{21}	C_{22}	C_{23}
S_3	C_{31}	C_{32}	C_{33}
S_4	C_{41}	C_{42}	C_{43}

Optimal Transport Program: Let $Z \in \mathbb{R}^{n_s \times n_d}$ such that
 $Z_{ij} \leftarrow$ amount of mass transported from S_i to D_j .

$$\text{minimise} \quad \sum_{i=1}^{n_s} \sum_{j=1}^{n_d} C_{ij} Z_{ij} = \langle Z, C \rangle$$

$$\text{subject to} \quad \sum_i Z_{ij} = s_i, \quad \sum_j Z_{ij} = d_j, \quad Z \geq 0$$

Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

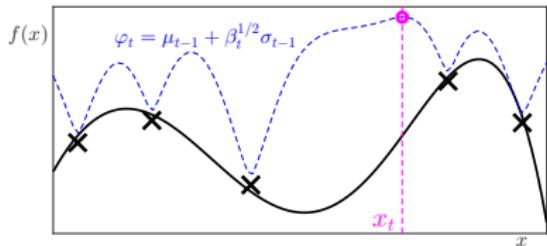
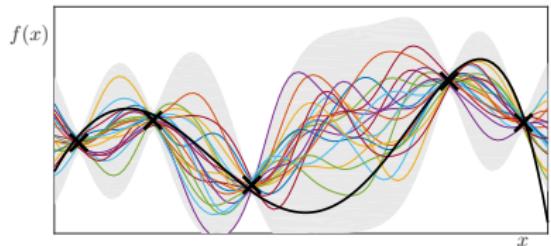
	D_1	D_2	D_3
S_1	C_{11}	C_{12}	C_{13}
S_2	C_{21}	C_{22}	C_{23}
S_3	C_{31}	C_{32}	C_{33}
S_4	C_{41}	C_{42}	C_{43}

Properties of OT

- ▶ OT is symmetric: solution the same if we swap sources and destinations.
- ▶ Connections to Wasserstein (earth mover) distances.
- ▶ Several efficient solvers (Peyré & Cuturi 2017, Villani 2008).
- ▶ OT can also be viewed as a minimum cost matching problem.

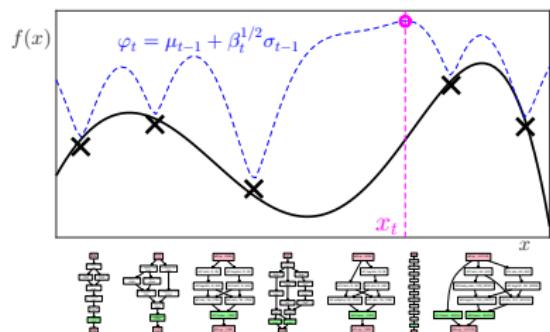
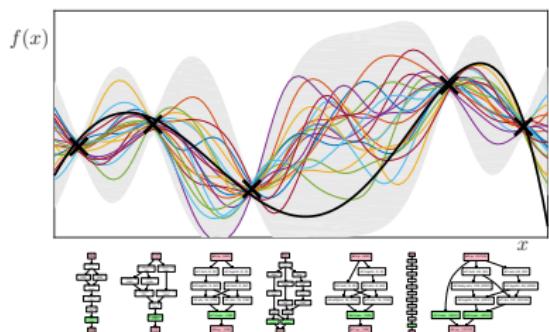
Bayesian Optimisation for Neural Architecture Search?

At each time step



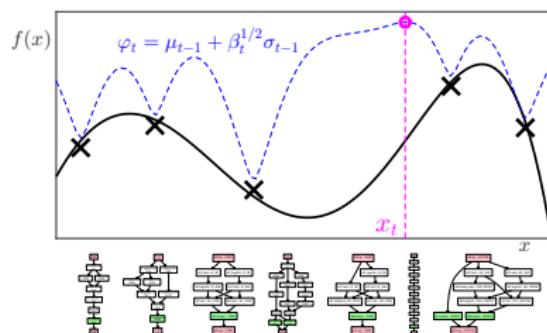
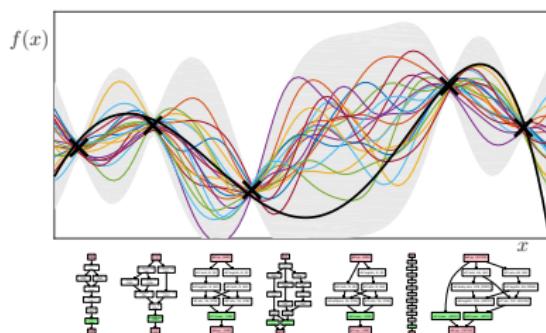
Bayesian Optimisation for Neural Architecture Search?

At each time step



Bayesian Optimisation for Neural Architecture Search?

At each time step



Main challenges

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

Outline

1. Review
 - ▶ Bayesian optimisation
 - ▶ Optimal transport
2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport
 - ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
 - ▶ Optimising the acquisition via an evolutionary algorithm
 - ▶ Experiments
3. Multi-fidelity optimisation in NASBOT

OTMANN: A distance between neural architectures

(Kandasamy et al. NIPS 2018)

Plan: Given a distance d , use " $\kappa = e^{-\beta d^p}$ " as the kernel.

OTMANN: A distance between neural architectures

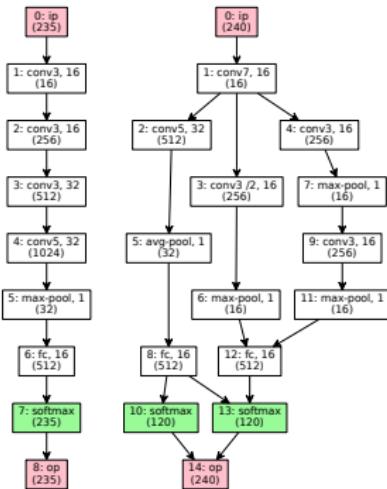
(Kandasamy et al. NIPS 2018)

Plan: Given a distance d , use " $\kappa = e^{-\beta d^p}$ " as the kernel.

Key idea: To compute distance between architectures G_1 , G_2 ,
match computation (layer mass) in layers in G_1 to G_2 .

$$Z \in \mathbb{R}^{n_1 \times n_2}.$$

$Z_{ij} \leftarrow$ amount matched between layer
 $i \in G_1$ and $j \in G_2$.



OTMANN: A distance between neural architectures

(Kandasamy et al. NIPS 2018)

Plan: Given a distance d , use " $\kappa = e^{-\beta d^p}$ " as the kernel.

Key idea: To compute distance between architectures G_1 , G_2 ,
match computation (layer mass) in layers in G_1 to G_2 .

$$Z \in \mathbb{R}^{n_1 \times n_2}.$$

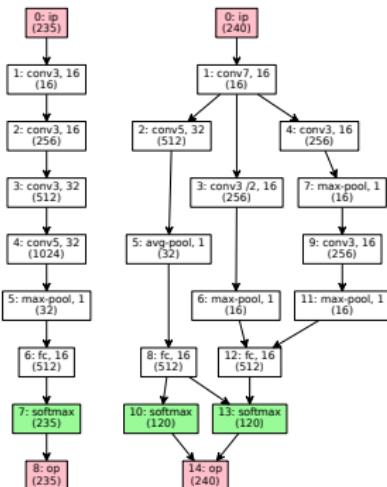
$Z_{ij} \leftarrow$ amount matched between layer
 $i \in G_1$ and $j \in G_2$.

Minimise $\phi_{lmm}(Z) + \phi_{str}(Z) + \phi_{nas}(Z)$

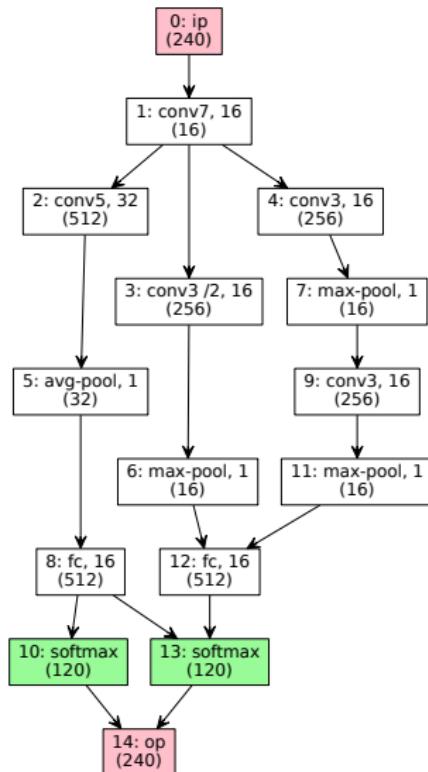
$\phi_{lmm}(Z)$: label mismatch penalty

$\phi_{str}(Z)$: structural penalty

$\phi_{nas}(Z)$: non-assignment penalty

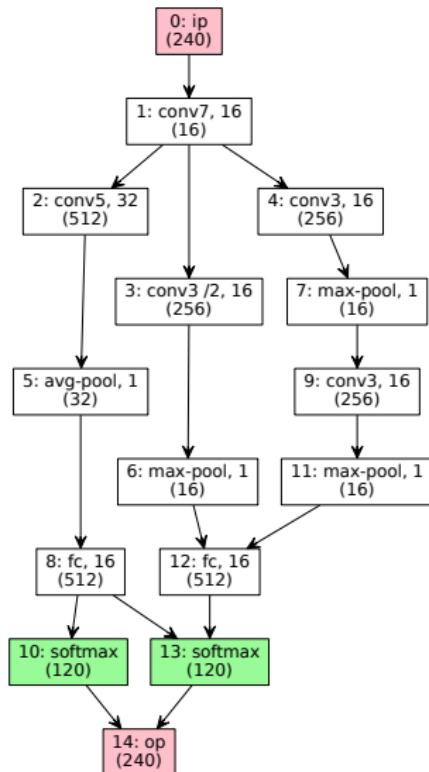


The layer masses at each layer is proportional to the amount of computation at each layer



Typically computed as,
incoming units × # units in layer

The layer masses at each layer is proportional to the amount of computation at each layer



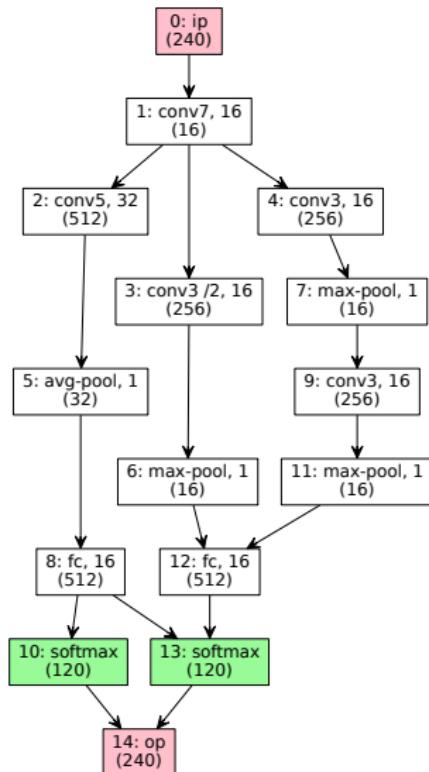
Typically computed as,
incoming units \times # units in layer

E.g.

$$\ell m(2) = 16 \times 32 = 512.$$

$$\ell m(12) = (16 + 16) \times 16 = 512.$$

The layer masses at each layer is proportional to the amount of computation at each layer



Typically computed as,
 $\# \text{ incoming units} \times \# \text{ units in layer}$

E.g.

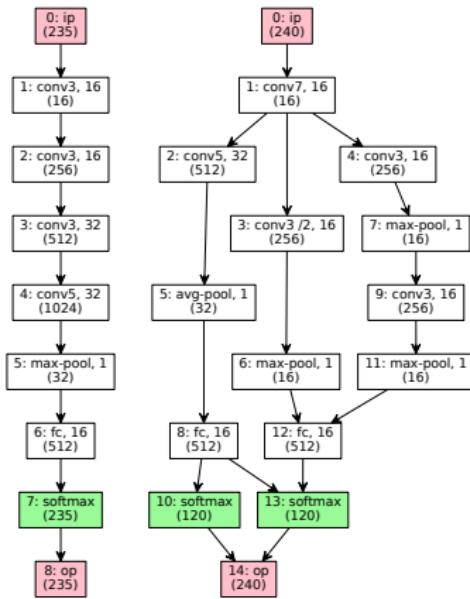
$$\ell m(2) = 16 \times 32 = 512.$$

$$\ell m(12) = (16 + 16) \times 16 = 512.$$

A few exceptions:

- input, output layers
- softmax/linear layers
- fully connected layers in CNNs

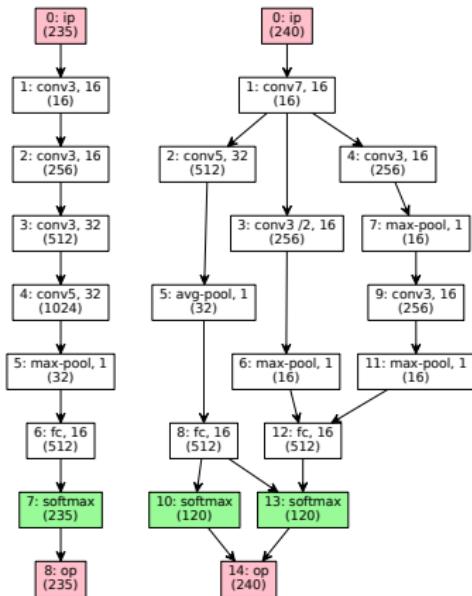
Label mismatch penalty



Define M ,

	c3	c5	mp	ap	fc
c3	0	0.2			
c5	0.2	0			
ap			0	0.25	
mp			0.25	0	
fc					0

Label mismatch penalty



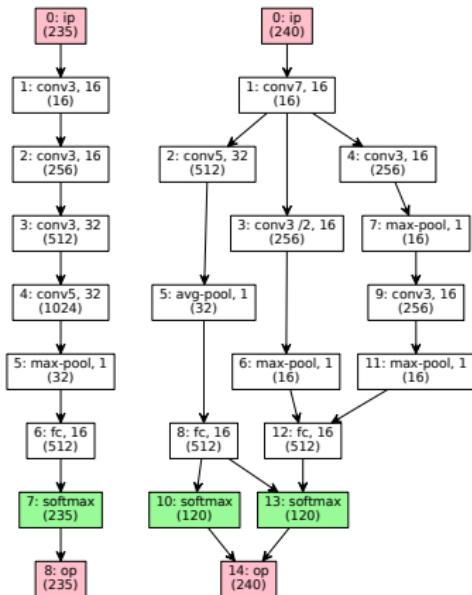
Define M ,

	c3	c5	mp	ap	fc
c3	0	0.2			
c5	0.2	0			
ap			0	0.25	
mp			0.25	0	
fc					0

Define $C_{lmm} \in \mathbb{R}^{n_1 \times n_2}$ where,

$$C_{lmm}(i, j) = M(\ell\ell(i), \ell\ell(j)).$$

Label mismatch penalty



Define M ,

	c3	c5	mp	ap	fc
c3	0	0.2			
c5	0.2	0			
ap			0	0.25	
mp			0.25	0	
fc					0

Define $C_{lmm} \in \mathbb{R}^{n_1 \times n_2}$ where,

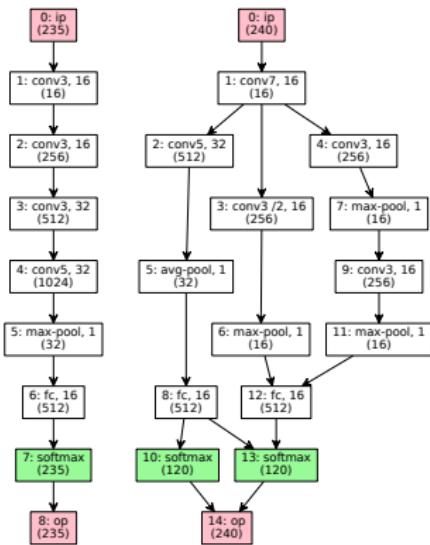
$$C_{lmm}(i, j) = M(\ell\ell(i), \ell\ell(j)).$$

Label mismatch penalty,

$$\phi_{lmm}(Z) = \langle Z, C_{lmm} \rangle.$$

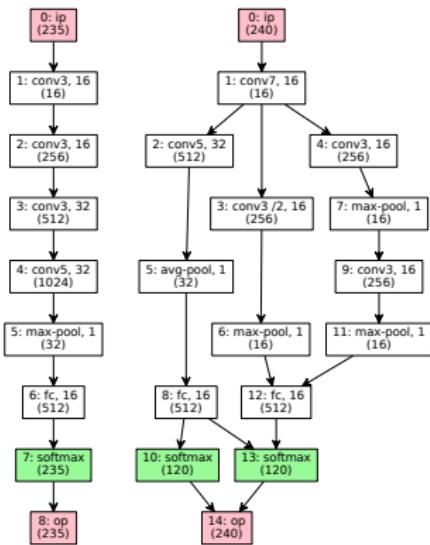
Structural Penalty

$\delta_{\text{op}}^{\text{sp}}(i), \delta_{\text{op}}^{\text{lp}}(i), \delta_{\text{op}}^{\text{rw}}(i) \leftarrow$ shortest, longest, random walk path lengths from layer i to output.



Structural Penalty

$\delta_{\text{op}}^{\text{sp}}(i), \delta_{\text{op}}^{\text{lp}}(i), \delta_{\text{op}}^{\text{rw}}(i) \leftarrow$ shortest, longest, random walk path lengths from layer i to output.

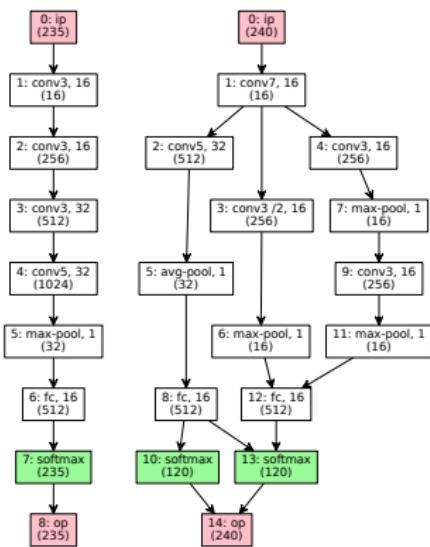


E.g. (in right network): $\delta_{\text{op}}^{\text{sp}}(1) = 5$,
 $\delta_{\text{op}}^{\text{lp}}(1) = 7$, $\delta_{\text{op}}^{\text{rw}}(1) = 5.67$.

Structural Penalty

$\delta_{\text{op}}^{\text{sp}}(i), \delta_{\text{op}}^{\text{lp}}(i), \delta_{\text{op}}^{\text{rw}}(i) \leftarrow$ shortest, longest, random walk path lengths from layer i to output.

Similarly define $\delta_{\text{ip}}^{\text{sp}}(i), \delta_{\text{ip}}^{\text{lp}}(i), \delta_{\text{ip}}^{\text{rw}}(i)$ from input to layer i .

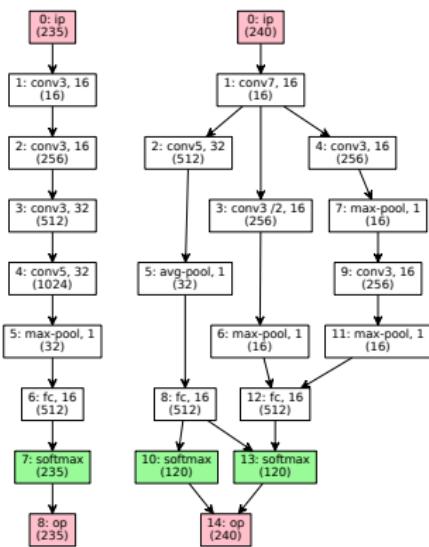


E.g. (in right network): $\delta_{\text{op}}^{\text{sp}}(1) = 5$,
 $\delta_{\text{op}}^{\text{lp}}(1) = 7$, $\delta_{\text{op}}^{\text{rw}}(1) = 5.67$.

Structural Penalty

$\delta_{\text{op}}^{\text{sp}}(i), \delta_{\text{op}}^{\text{lp}}(i), \delta_{\text{op}}^{\text{rw}}(i) \leftarrow$ shortest, longest, random walk path lengths from layer i to output.

Similarly define $\delta_{\text{ip}}^{\text{sp}}(i), \delta_{\text{ip}}^{\text{lp}}(i), \delta_{\text{ip}}^{\text{rw}}(i)$ from input to layer i .



E.g. (in right network): $\delta_{\text{op}}^{\text{sp}}(1) = 5$,
 $\delta_{\text{op}}^{\text{lp}}(1) = 7$, $\delta_{\text{op}}^{\text{rw}}(1) = 5.67$.

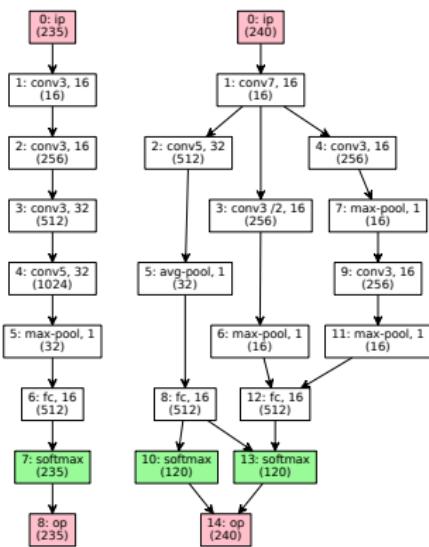
Let $C_{\text{str}} \in \mathbb{R}^{n_1 \times n_2}$ where,

$$C_{\text{str}}(i, j) = \frac{1}{6} \sum_s \sum_t |\delta_t^s(i) - \delta_t^s(j)|.$$
$$s \in \{\text{sp, lp, rw}\}, t \in \{\text{ip, op}\}$$

Structural Penalty

$\delta_{\text{op}}^{\text{sp}}(i), \delta_{\text{op}}^{\text{lp}}(i), \delta_{\text{op}}^{\text{rw}}(i) \leftarrow$ shortest, longest, random walk path lengths from layer i to output.

Similarly define $\delta_{\text{ip}}^{\text{sp}}(i), \delta_{\text{ip}}^{\text{lp}}(i), \delta_{\text{ip}}^{\text{rw}}(i)$ from input to layer i .



E.g. (in right network): $\delta_{\text{op}}^{\text{sp}}(1) = 5$,
 $\delta_{\text{op}}^{\text{lp}}(1) = 7$, $\delta_{\text{op}}^{\text{rw}}(1) = 5.67$.

Let $C_{\text{str}} \in \mathbb{R}^{n_1 \times n_2}$ where,

$$C_{\text{str}}(i, j) = \frac{1}{6} \sum_s \sum_t |\delta_t^s(i) - \delta_t^s(j)|.$$
$$s \in \{\text{sp, lp, rw}\}, t \in \{\text{ip, op}\}$$

Structural penalty,

$$\phi_{\text{str}}(Z) = \langle Z, C_{\text{str}} \rangle.$$

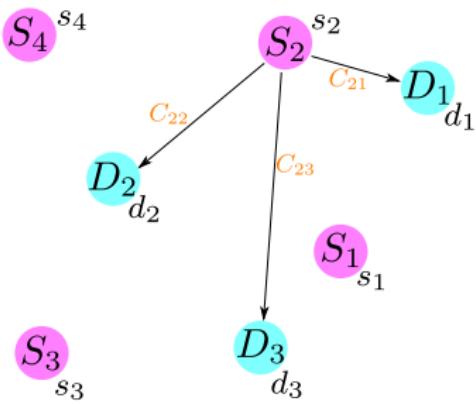
Non-assignment penalty

The non-assignment penalty is the amount of mass unmatched in both networks,

$$\phi_{\text{nas}}(Z) = \sum_{i \in \mathcal{L}_1} (\ell m(i) - \sum_{j \in \mathcal{L}_2} Z_{ij}) + \sum_{j \in \mathcal{L}_2} (\ell m(j) - \sum_{i \in \mathcal{L}_1} Z_{ij}).$$

The cost per unit for unassigned mass is 1.

Optimal Transport



total supply = total demand
 $\sum_{i=1}^{n_s} s_i = \sum_{j=1}^{n_d} d_j.$

	D_1	D_2	D_3
S_1	C_{11}	C_{12}	C_{13}
S_2	C_{21}	C_{22}	C_{23}
S_3	C_{31}	C_{32}	C_{33}
S_4	C_{41}	C_{42}	C_{43}

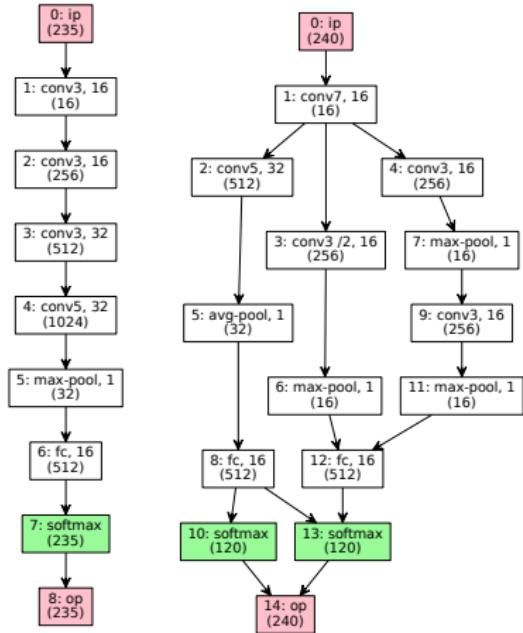
Optimal Transport Program: Let $Z \in \mathbb{R}^{n_s \times n_d}$ such that
 $Z_{ij} \leftarrow$ amount of mass transported from S_i to D_j .

$$\text{minimise} \quad \sum_{i=1}^{n_s} \sum_{j=1}^{n_d} C_{ij} Z_{ij} = \langle Z, C \rangle$$

$$\text{subject to} \quad \sum_i Z_{ij} = s_i, \quad \sum_j Z_{ij} = d_i, \quad Z \geq 0$$

Computing OTMANN via Optimal Transport

Introduce sink layers with mass equal to total mass of other network. Unit cost for matching sink nodes is 0.

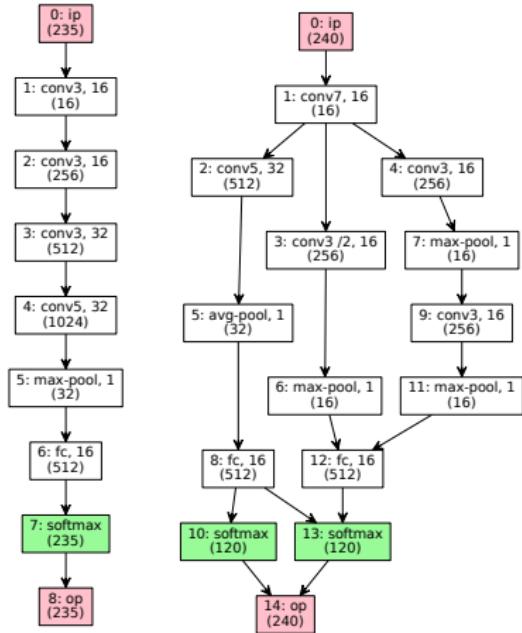


sink_1
(3120)

sink_2
(3055)

Computing OTMANN via Optimal Transport

Introduce sink layers with mass equal to total mass of other network. Unit cost for matching sink nodes is 0.

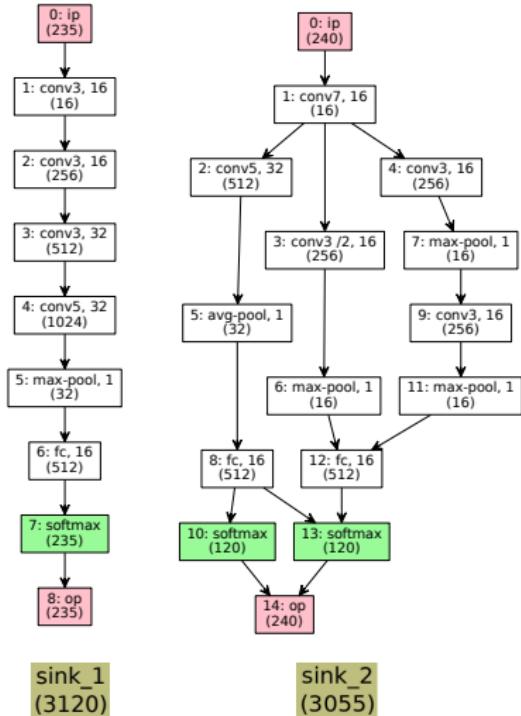


OT variable Z' and cost matrix C' , $C', Z' \in \mathbb{R}^{(n_1+1) \times (n_2+1)}$.

sink_1
(3120)

sink_2
(3055)

Computing OTMANN via Optimal Transport



Introduce sink layers with mass equal to total mass of other network. Unit cost for matching sink nodes is 0.

OT variable Z' and cost matrix C' , $C', Z' \in \mathbb{R}^{(n_1+1) \times (n_2+1)}$.

For $i \leq n_1, j \leq n_2$,

$$C'(i, j) = C_{\text{lmm}}(i, j) + C_{\text{str}}(i, j)$$

C' looks as follows,

		1
		\vdots
		1
$C_{\text{lmm}} + C_{\text{str}}$		
1	...	1
		0

Theoretical Properties of OTMANN

(Kandasamy et al. NIPS 2018)

1. It can be computed via an optimal transport scheme.
2. Under mild regularity conditions, it is a pseudo-distance.
That is, given neural networks $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$,

- ▶ $d(\mathcal{G}_1, \mathcal{G}_2) \geq 0$.
- ▶ $d(\mathcal{G}_1, \mathcal{G}_2) = d(\mathcal{G}_2, \mathcal{G}_1)$.
- ▶ $d(\mathcal{G}_1, \mathcal{G}_3) \leq d(\mathcal{G}_1, \mathcal{G}_2) + d(\mathcal{G}_2, \mathcal{G}_3)$.

Theoretical Properties of OTMANN

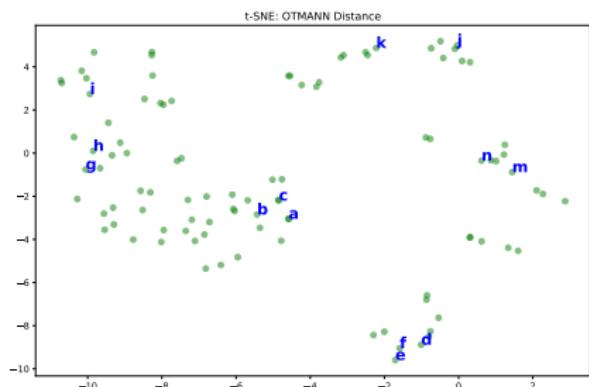
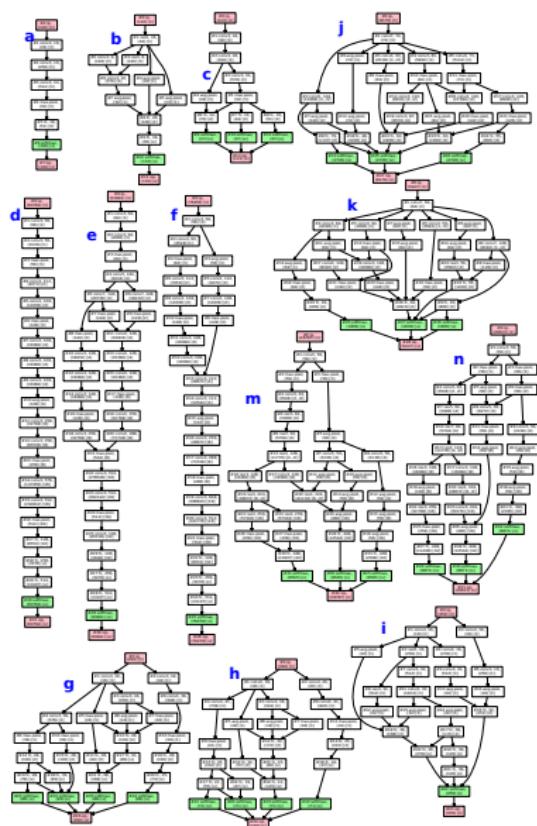
(Kandasamy et al. NIPS 2018)

1. It can be computed via an optimal transport scheme.
2. Under mild regularity conditions, it is a pseudo-distance.
That is, given neural networks $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$,
 - ▶ $d(\mathcal{G}_1, \mathcal{G}_2) \geq 0$.
 - ▶ $d(\mathcal{G}_1, \mathcal{G}_2) = d(\mathcal{G}_2, \mathcal{G}_1)$.
 - ▶ $d(\mathcal{G}_1, \mathcal{G}_3) \leq d(\mathcal{G}_1, \mathcal{G}_2) + d(\mathcal{G}_2, \mathcal{G}_3)$.

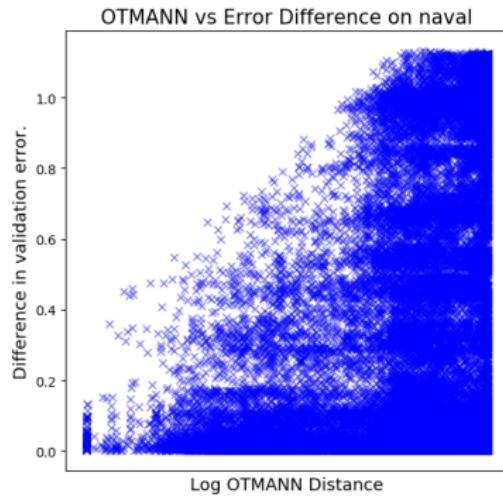
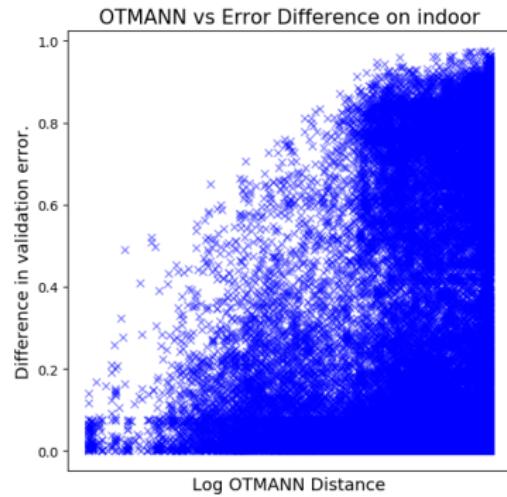
From distance to kernel:

Given OTMANN distance d , use $\kappa = e^{-\beta d}$ as the “kernel”.

OTMANN: Illustration with tSNE Embeddings



OTMANN correlates with cross validation performance



Outline

1. Review
 - ▶ Bayesian optimisation
 - ▶ Optimal transport
2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport
 - ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
 - ▶ Optimising the acquisition via an evolutionary algorithm
 - ▶ Experiments
3. Multi-fidelity optimisation in NASBOT

Optimising the acquisition via an Evolutionary Algorithm

EA navigates the search space by applying a sequence of local modifiers to the points already evaluated.

Optimising the acquisition via an Evolutionary Algorithm

EA navigates the search space by applying a sequence of local modifiers to the points already evaluated.

Each modifier:

- ▶ Takes a network, modifies and returns a new one.
- ▶ Each modifier can change the number of units in each layer, add/delte layers, or modify the architecture of the network.

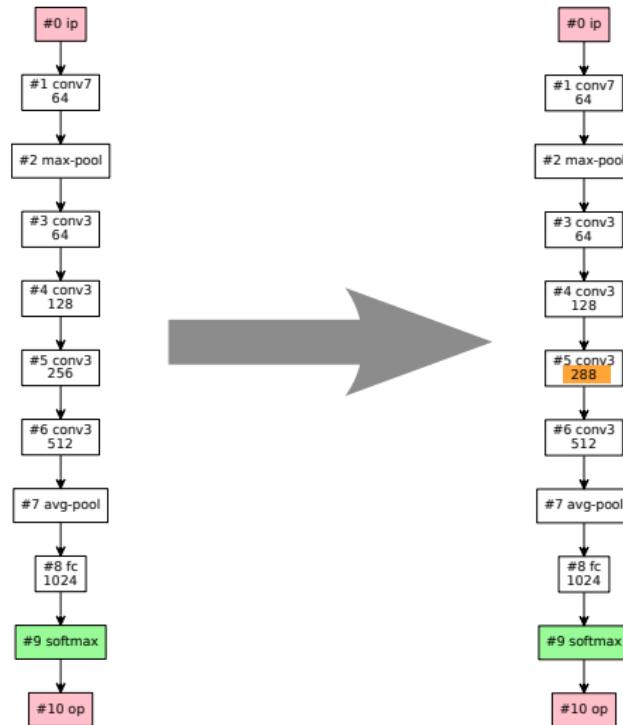
Optimising the acquisition via an Evolutionary Algorithm

EA navigates the search space by applying a sequence of local modifiers to the points already evaluated.

Each modifier:

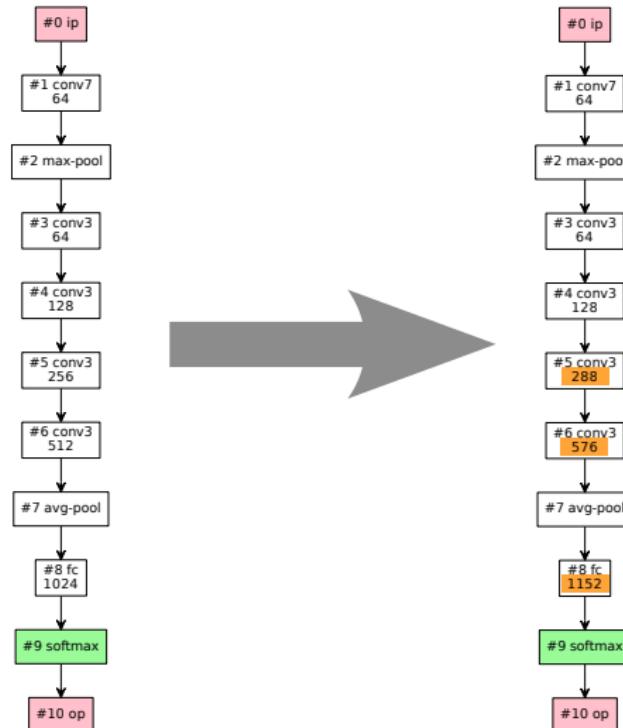
- ▶ Takes a network, modifies and returns a new one.
- ▶ Each modifier can change the number of units in each layer, add/delte layers, or modify the architecture of the network.
- ▶ Care must be taken to ensure that the resulting networks are still “valid”.

inc_single



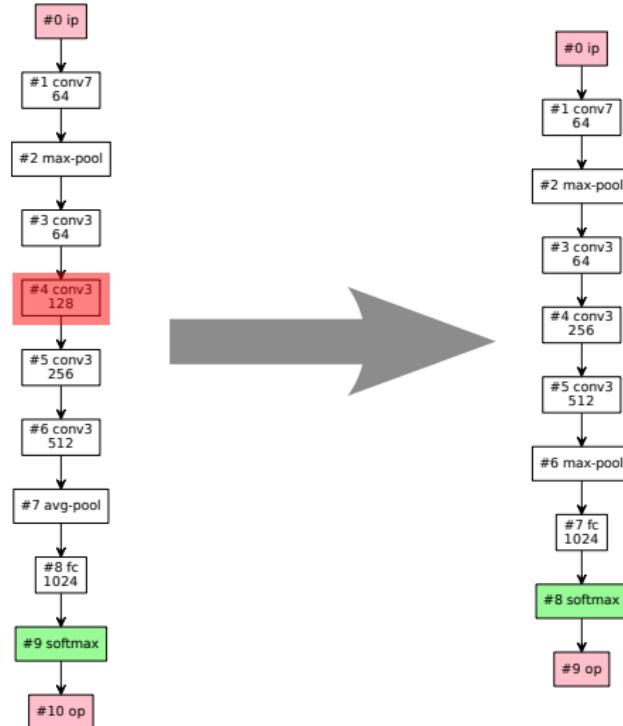
Similarly define dec_single

inc_en_masse

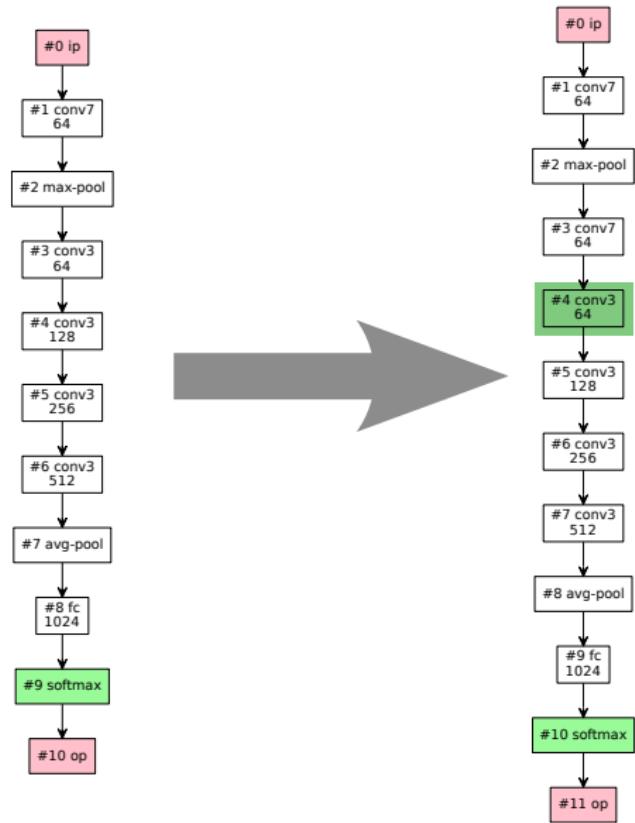


Similarly define `dec_en_masse`

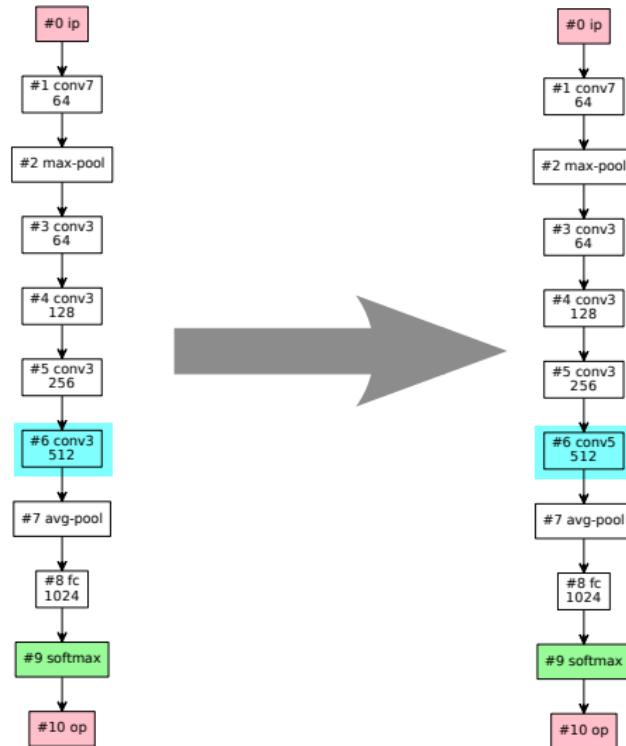
remove_layer



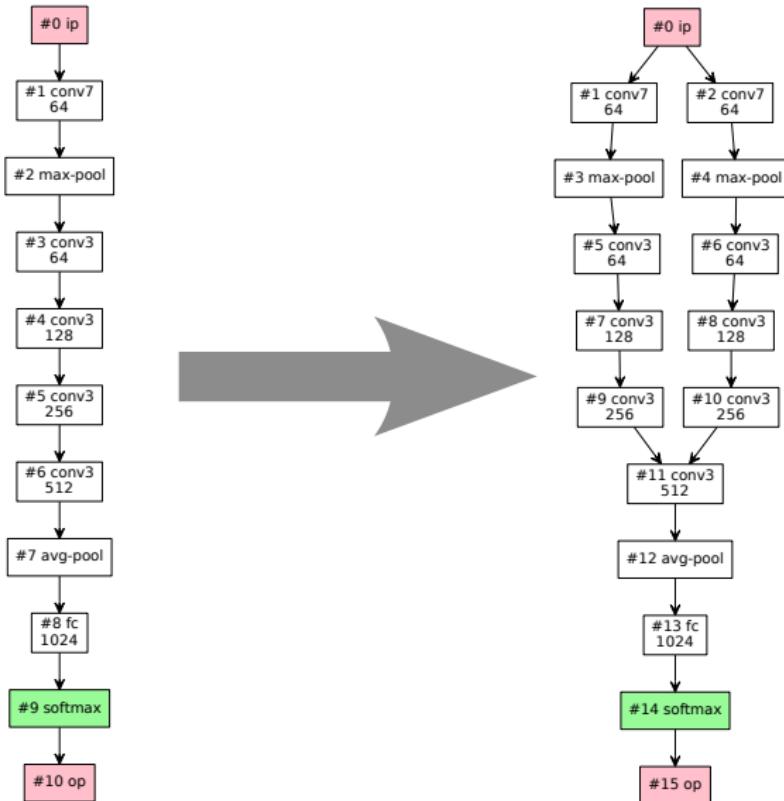
wedge_layer



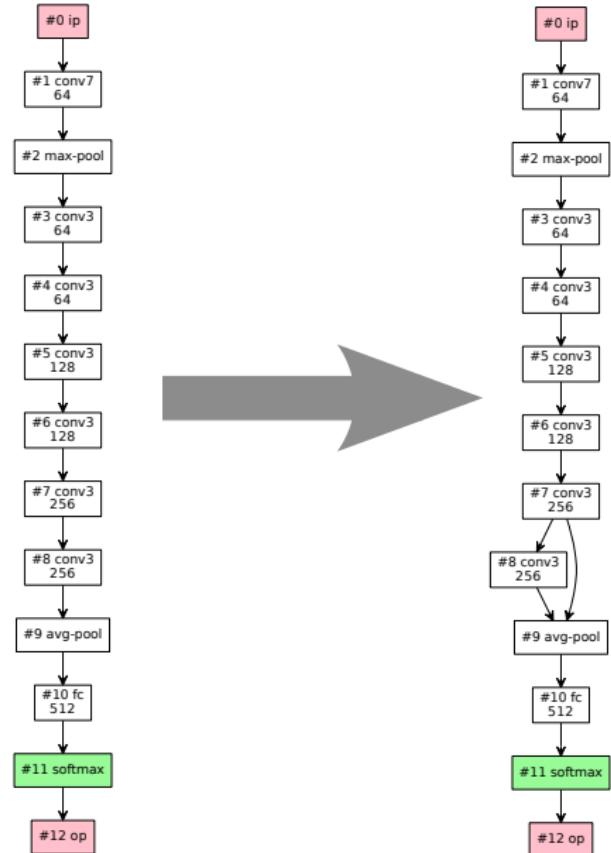
swap_label



dup_path



skip



Optimising the acquisition via EA

Operation	Description
dec_single	Pick a layer at random and decrease the number of units by $1/8$.
dec_en_masse	Pick several layers at random in topological order and decrease the number of units by $1/8$ for all of them.
inc_single	Pick a layer at random and increase the number of units by $1/8$.
inc_en_masse	Pick several layers at random in topological order and increase the number of units by $1/8$ for all of them.
dup_path	Pick a random path $u_1, u_2, \dots, u_{k-1}, u_k$, duplicate layers u_2, \dots, u_{k-1} and connect them to u_1 and u_k .
remove_layer	Pick a layer at random and remove it. Connect the layer's parents to its children if necessary.
skip	Randomly pick layers u, v where u is topologically before v . Add (u, v) to \mathcal{E} .
swap_label	Randomly pick a layer and change its label.
wedge_layer	Randomly remove an edge (u, v) from \mathcal{E} . Create a new layer w and add $(u, w), (w, v)$ to \mathcal{E} .

Goal: optimise the acquisition (e.g. UCB, EI etc.)

Optimising the acquisition via EA

Operation	Description
dec_single	Pick a layer at random and decrease the number of units by $1/8$.
dec_en_masse	Pick several layers at random in topological order and decrease the number of units by $1/8$ for all of them.
inc_single	Pick a layer at random and increase the number of units by $1/8$.
inc_en_masse	Pick several layers at random in topological order and increase the number of units by $1/8$ for all of them.
dup_path	Pick a random path $u_1, u_2, \dots, u_{k-1}, u_k$, duplicate layers u_2, \dots, u_{k-1} and connect them to u_1 and u_k .
remove_layer	Pick a layer at random and remove it. Connect the layer's parents to its children if necessary.
skip	Randomly pick layers u, v where u is topologically before v . Add (u, v) to \mathcal{E} .
swap_label	Randomly pick a layer and change its label.
wedge_layer	Randomly remove an edge (u, v) from \mathcal{E} . Create a new layer w and add $(u, w), (w, v)$ to \mathcal{E} .

Goal: optimise the acquisition (e.g. UCB, EI etc.)

- ▶ Evaluate the acquisition on an initial pool of networks.

Optimising the acquisition via EA

Operation	Description
dec_single	Pick a layer at random and decrease the number of units by $1/8$.
dec_en_masse	Pick several layers at random in topological order and decrease the number of units by $1/8$ for all of them.
inc_single	Pick a layer at random and increase the number of units by $1/8$.
inc_en_masse	Pick several layers at random in topological order and increase the number of units by $1/8$ for all of them.
dup_path	Pick a random path $u_1, u_2, \dots, u_{k-1}, u_k$, duplicate layers u_2, \dots, u_{k-1} and connect them to u_1 and u_k .
remove_layer	Pick a layer at random and remove it. Connect the layer's parents to its children if necessary.
skip	Randomly pick layers u, v where u is topologically before v . Add (u, v) to \mathcal{E} .
swap_label	Randomly pick a layer and change its label.
wedge_layer	Randomly remove an edge (u, v) from \mathcal{E} . Create a new layer w and add $(u, w), (w, v)$ to \mathcal{E} .

Goal: optimise the acquisition (e.g. UCB, EI etc.)

- ▶ Evaluate the acquisition on an initial pool of networks.
- ▶ Stochastically select those that have a high acquisition value and apply modifiers to generate a pool of candidates.

Optimising the acquisition via EA

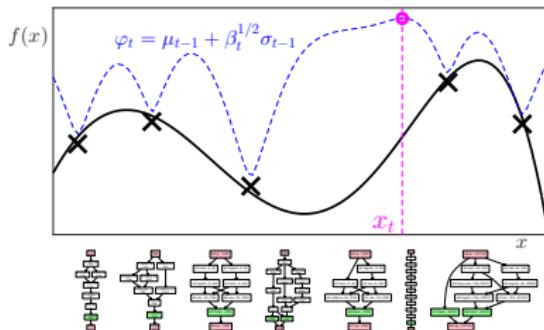
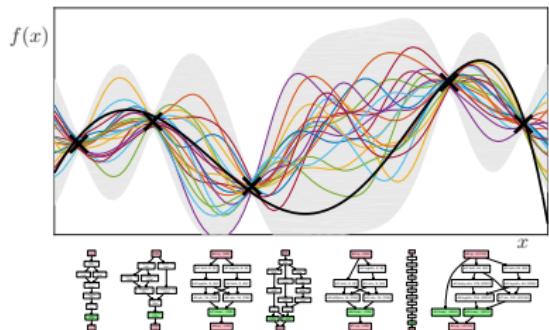
Operation	Description
dec_single	Pick a layer at random and decrease the number of units by $1/8$.
dec_en_masse	Pick several layers at random in topological order and decrease the number of units by $1/8$ for all of them.
inc_single	Pick a layer at random and increase the number of units by $1/8$.
inc_en_masse	Pick several layers at random in topological order and increase the number of units by $1/8$ for all of them.
dup_path	Pick a random path $u_1, u_2, \dots, u_{k-1}, u_k$, duplicate layers u_2, \dots, u_{k-1} and connect them to u_1 and u_k .
remove_layer	Pick a layer at random and remove it. Connect the layer's parents to its children if necessary.
skip	Randomly pick layers u, v where u is topologically before v . Add (u, v) to \mathcal{E} .
swap_label	Randomly pick a layer and change its label.
wedge_layer	Randomly remove an edge (u, v) from \mathcal{E} . Create a new layer w and add $(u, w), (w, v)$ to \mathcal{E} .

Goal: optimise the acquisition (e.g. UCB, EI etc.)

- ▶ Evaluate the acquisition on an initial pool of networks.
- ▶ Stochastically select those that have a high acquisition value and apply modifiers to generate a pool of candidates.
- ▶ Evaluate the acquisition on those candidates and repeat.

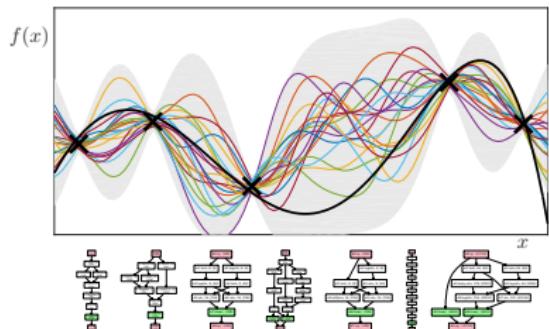
Neural Architecture Search via Bayesian Optimisation

At each time step

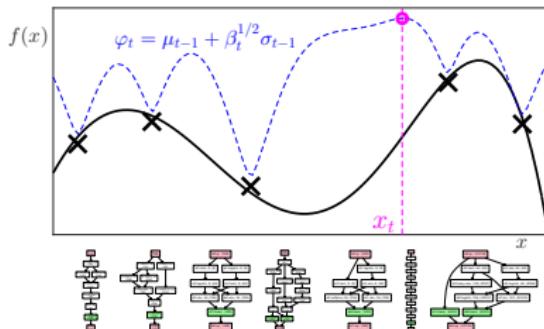


Neural Architecture Search via Bayesian Optimisation

At each time step



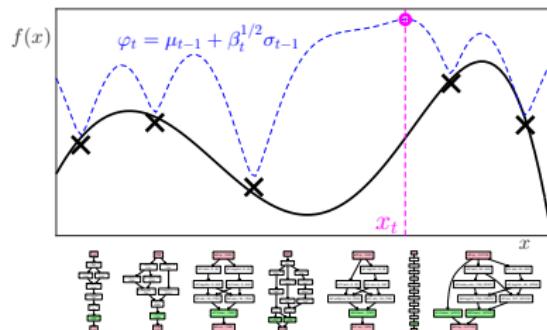
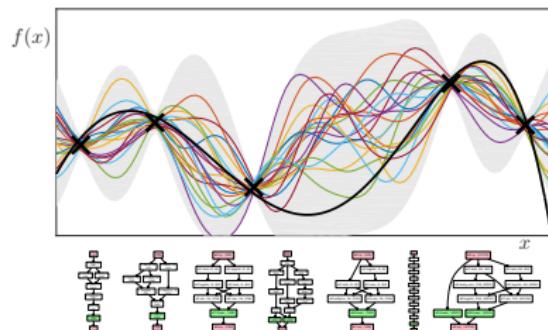
OTMANN



Evolutionary Algorithm

Neural Architecture Search via Bayesian Optimisation

At each time step



Resulting Procedure: NASBOT

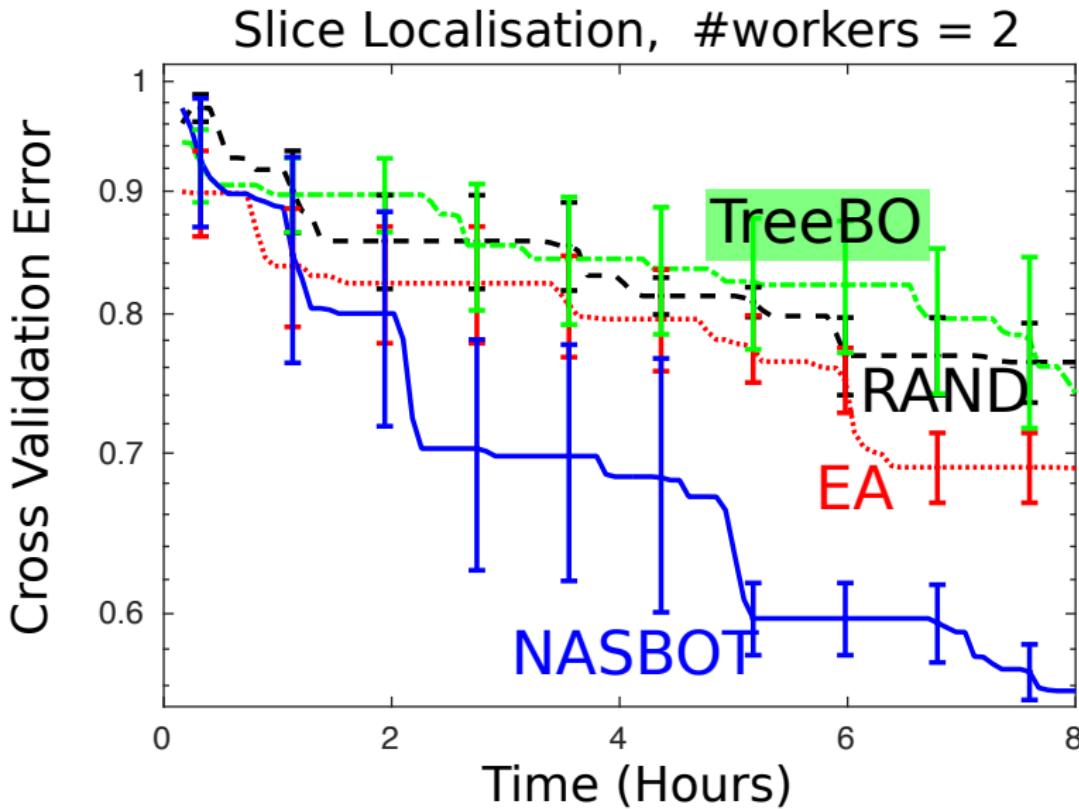
Neural Architecture Search with Bayesian Optimisation and
Optimal Transport

(Kandasamy et al. NIPS 2018)

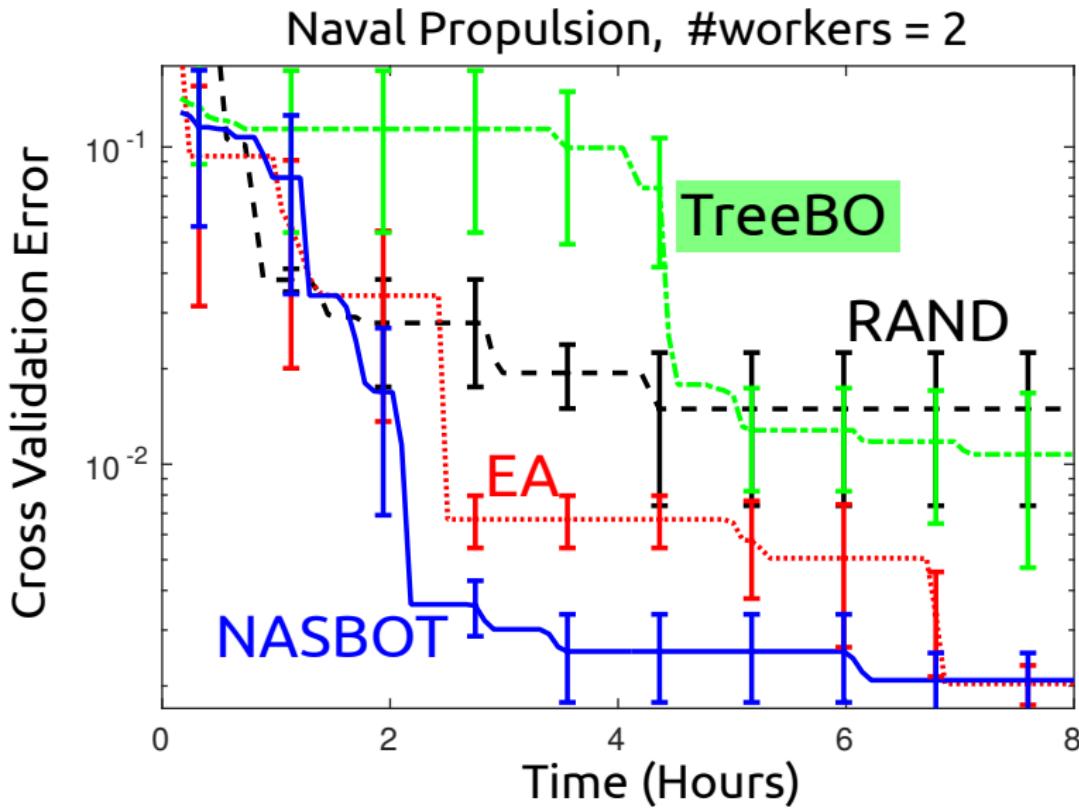
Outline

1. Review
 - ▶ Bayesian optimisation
 - ▶ Optimal transport
2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport
 - ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
 - ▶ Optimising the acquisition via an evolutionary algorithm
 - ▶ Experiments
3. Multi-fidelity optimisation in NASBOT

NAS Results

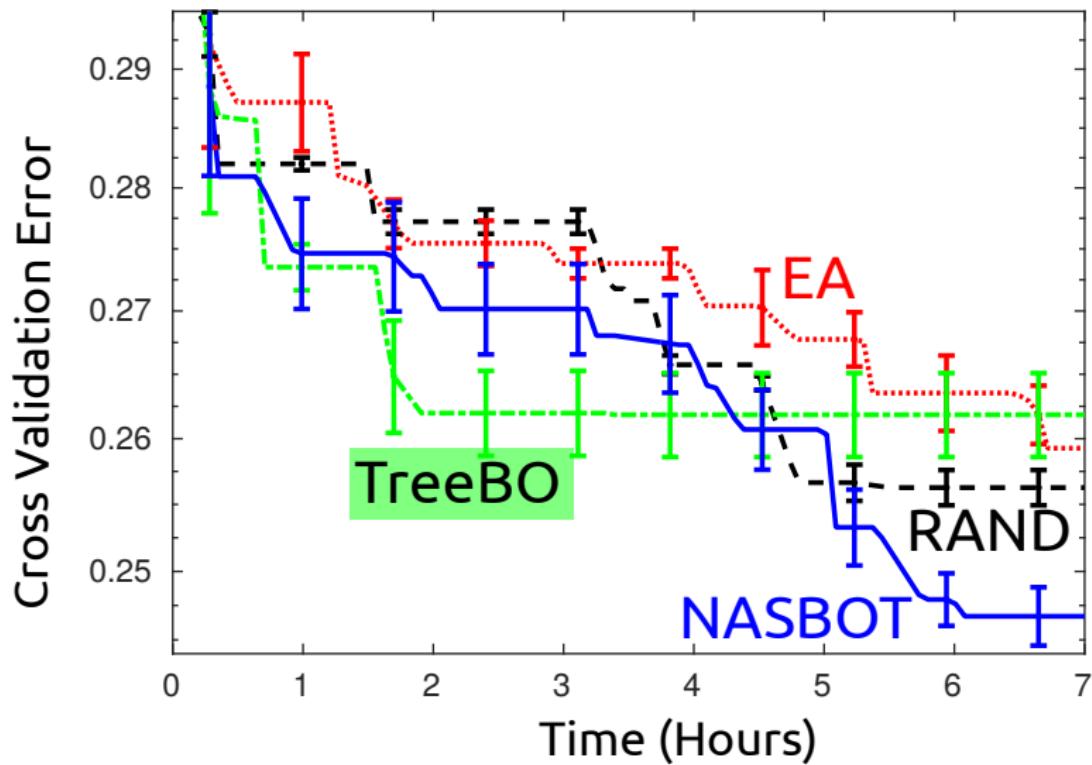


NAS Results



NAS Results

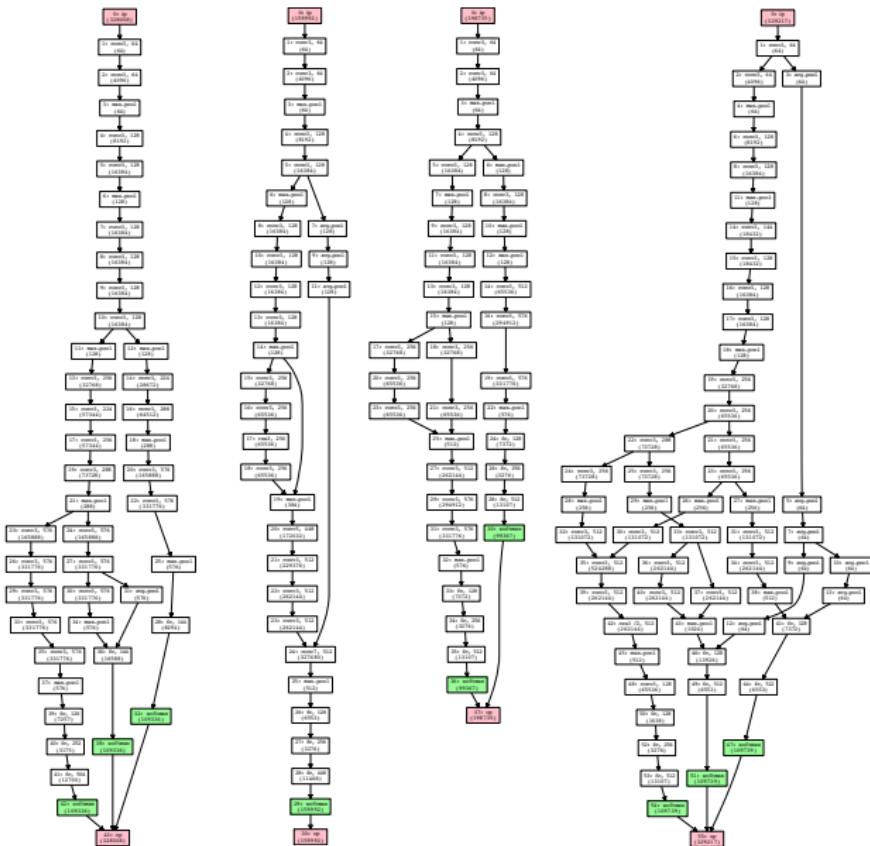
Cifar10, #workers = 4



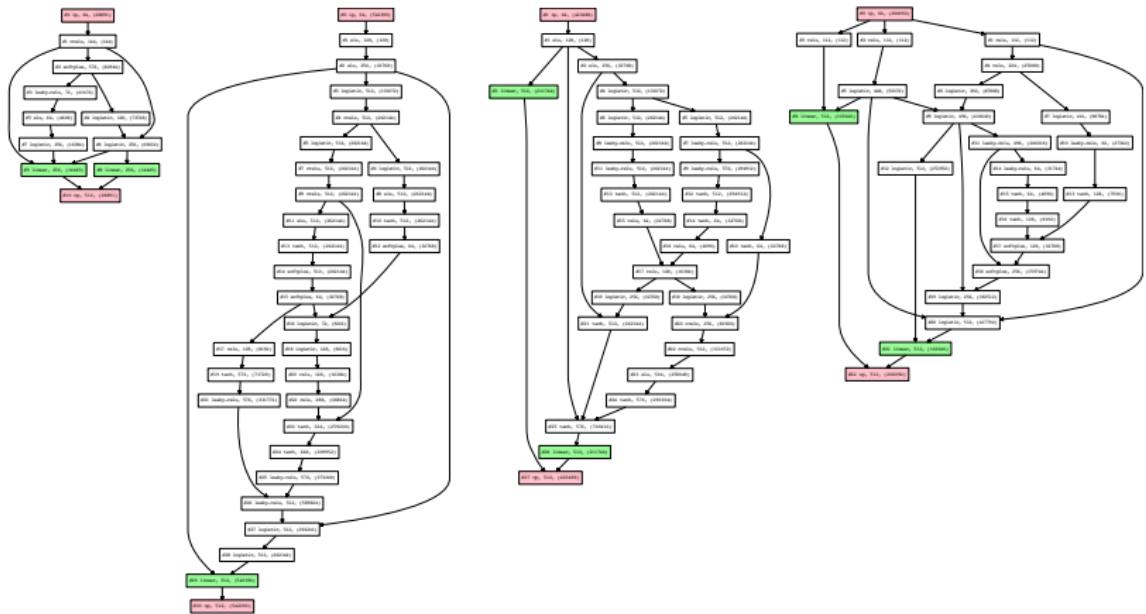
Test Error on 7 Datasets

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	0.115 ± 0.023	0.758 ± 0.041	0.0103 ± 0.002	0.948 ± 0.024	0.762 ± 0.013	0.1342 ± 0.002	0.0914 ± 0.008
EA	0.806 ± 0.040	0.147 ± 0.010	0.733 ± 0.041	0.0079 ± 0.004	1.010 ± 0.038	0.758 ± 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	0.731 ± 0.029	0.117 ± 0.008	0.615 ± 0.044	0.0075 ± 0.002	0.902 ± 0.033	0.752 ± 0.024	0.1209 ± 0.003	0.0869 ± 0.004

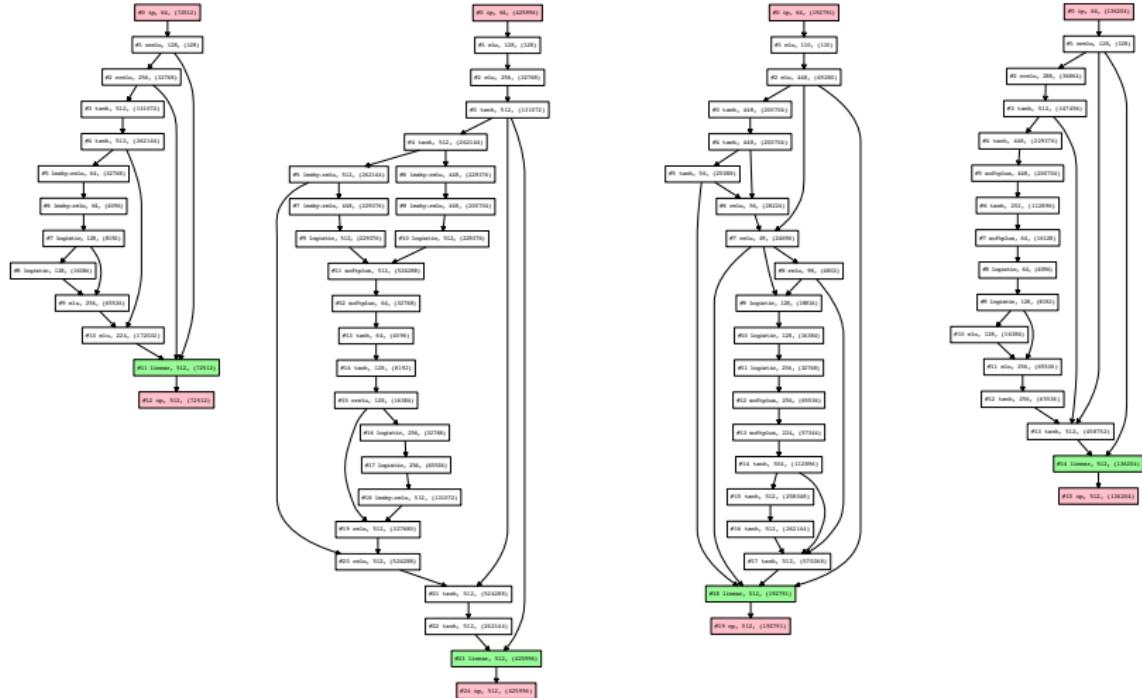
Architectures found on Cifar10



Architectures found on Indoor Location



Architectures found on Slice Localisation



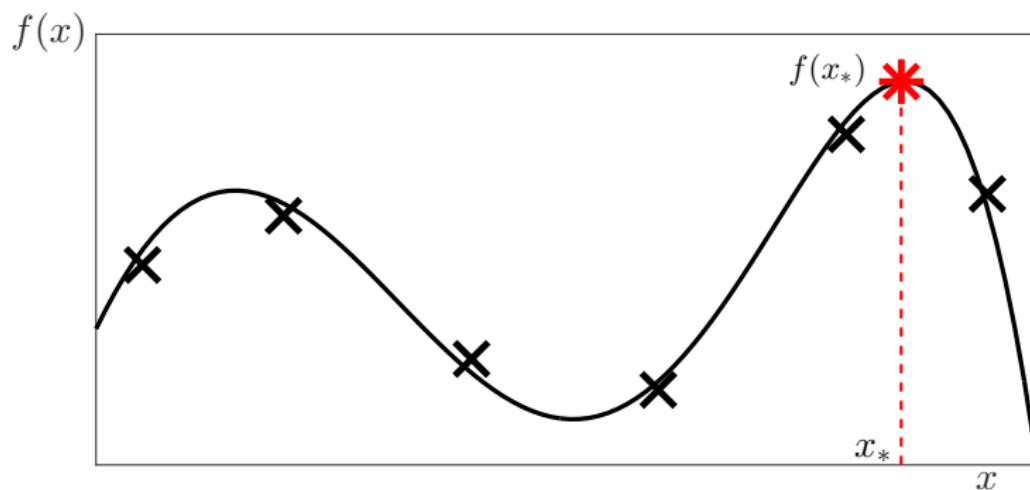
Outline

1. Review
 - ▶ Bayesian optimisation
 - ▶ Optimal transport
2. NASBOT: Neural Architecture Search with Bayesian Optimisation & Optimal Transport
 - ▶ OTMANN: Optimal Transport Metrics for Architectures of Neural Networks
 - ▶ Optimising the acquisition via an evolutionary algorithm
 - ▶ Experiments
3. Multi-fidelity optimisation in NASBOT

Bayesian Optimisation

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

Let $x_* = \operatorname{argmax}_x f(x)$.



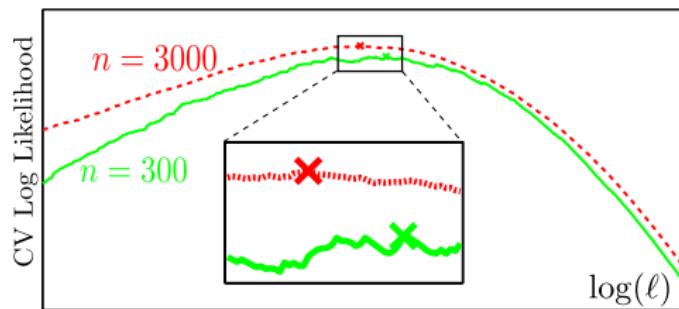
Multi-fidelity Bandits

Motivating question:

What if we have cheap approximations to f ?

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

E.g. Bandwidth (ℓ) selection in kernel density estimation.



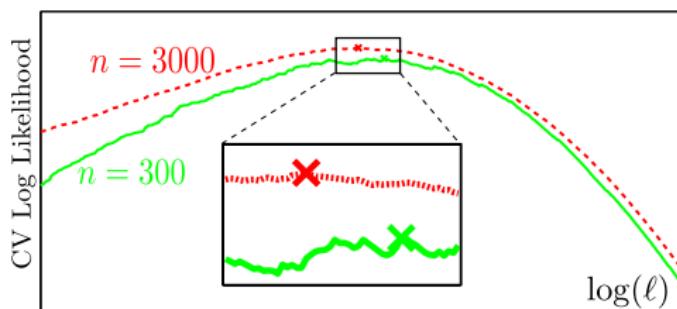
Multi-fidelity Bandits

Motivating question:

What if we have cheap approximations to f ?

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

E.g. Bandwidth (ℓ) selection in kernel density estimation.



2. Computational astrophysics: cosmological simulations and numerical computations with less granularity.
3. Autonomous driving: simulation vs real world experiment.

Multi-fidelity Bandits for Hyper-parameter tuning

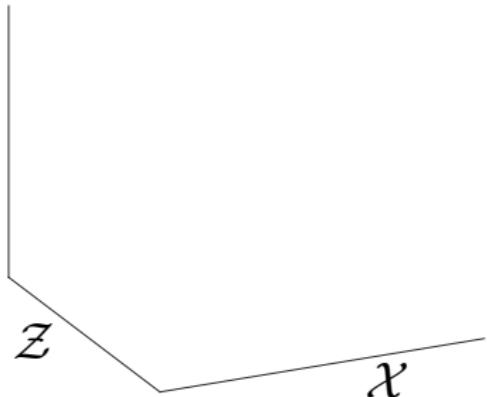
E.g. Train an ML model with N_\bullet data and T_\bullet iterations.

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

Multi-fidelity Bandits for Hyper-parameter tuning

- E.g. Train an ML model with N_{\bullet} data and T_{\bullet} iterations.
- But 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) .



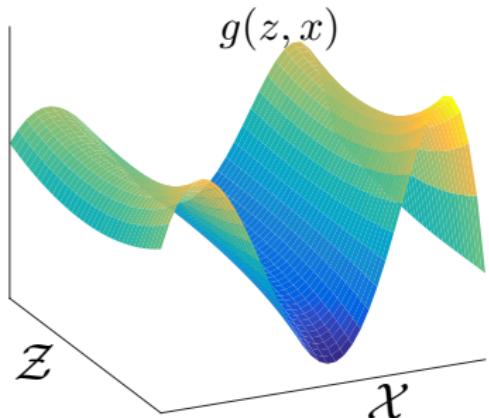
A fidelity space \mathcal{Z} and domain \mathcal{X}

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

$\mathcal{X} \leftarrow$ all hyper-parameter 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.

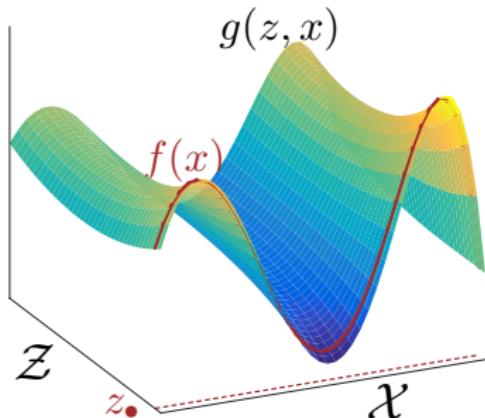
$\mathcal{X} \leftarrow$ all hyper-parameter values.

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

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

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

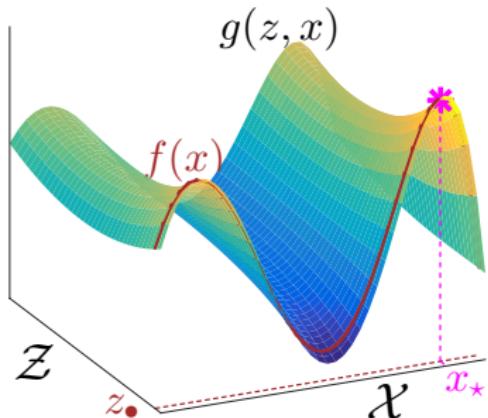
$g([N, T], x) \leftarrow$ cv accuracy when training with N data for T iterations at hyper-parameter x .

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 hyper-parameter 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 hyper-parameter x .

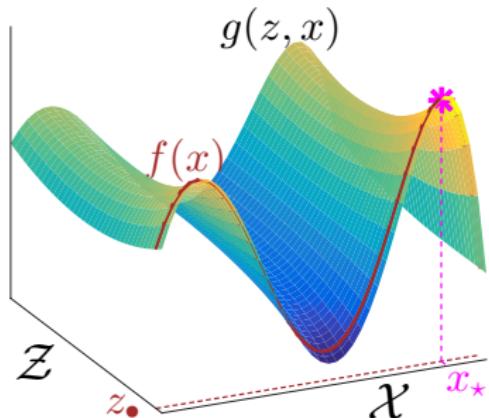
Denote $f(x) = g(z_\bullet, x)$ where $z_\bullet \in \mathcal{Z}$.

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

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

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 hyper-parameter 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 hyper-parameter x .

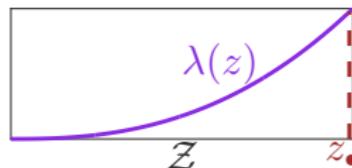
Denote $f(x) = g(z•, x)$ where $z• \in \mathcal{Z}$.

$z• = [N•, T•]$.

End Goal: Find $x* = \operatorname{argmax}_x f(x)$.

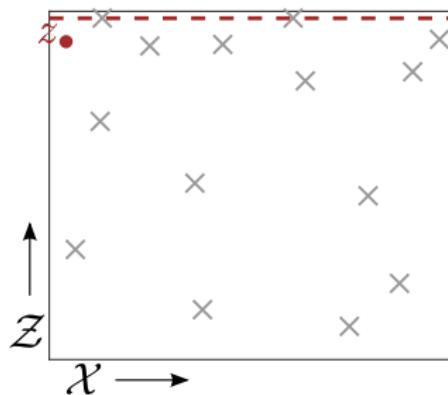
A cost function, $\lambda : \mathcal{Z} \rightarrow \mathbb{R}_+$.

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



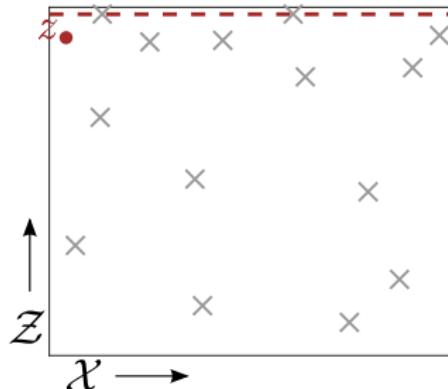
Algorithm: BOCA

(Kandasamy et al. ICML 2017)



Algorithm: BOCA

(Kandasamy et al. ICML 2017)



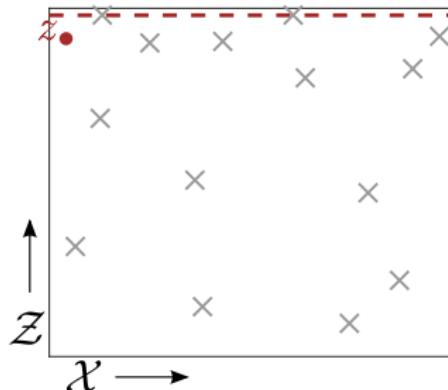
Model $g \sim \mathcal{GP}(0, \kappa)$ and compute posterior \mathcal{GP} :

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

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

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



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

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

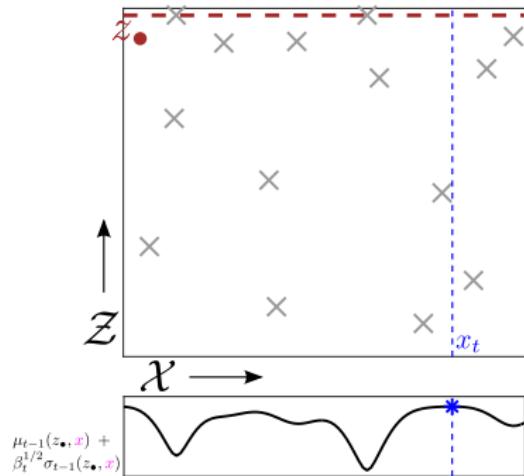
std-dev $\sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}_+$

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

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

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



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

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

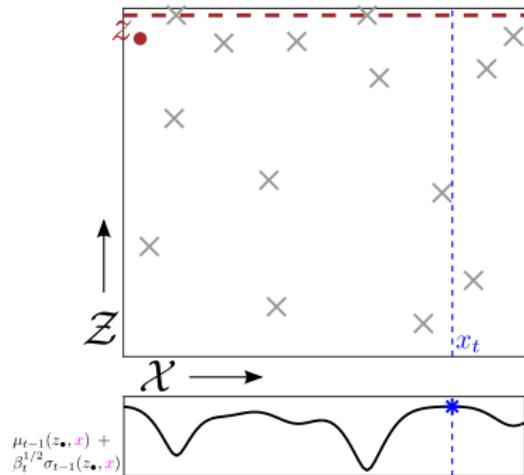
std-dev $\sigma_{t-1} : \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}_+$

(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 $\mu_{t-1} : \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}$

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

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

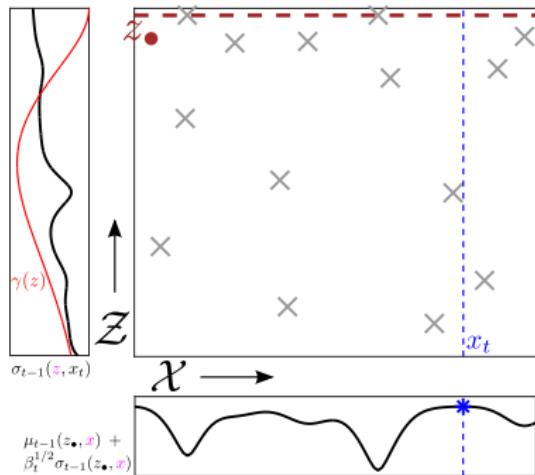
$$x_t = \operatorname{argmax}_{x \in \mathcal{X}} \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2} \sigma_{t-1}(z_\bullet, x)$$

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

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

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



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

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

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

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

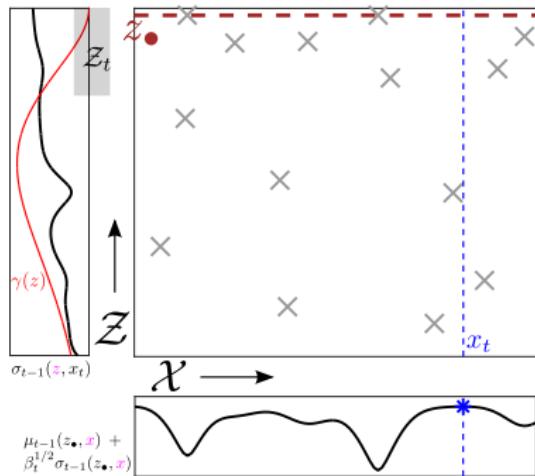
$$x_t = \operatorname{argmax}_{x \in \mathcal{X}} \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2} \sigma_{t-1}(z_\bullet, x)$$

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

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

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



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

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

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

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

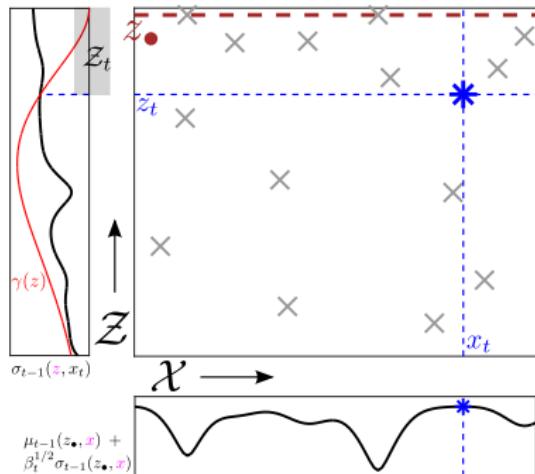
$$x_t = \operatorname{argmax}_{x \in \mathcal{X}} \mu_{t-1}(z_•, x) + \beta_t^{1/2} \sigma_{t-1}(z_•, x)$$

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

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

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



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

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

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

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

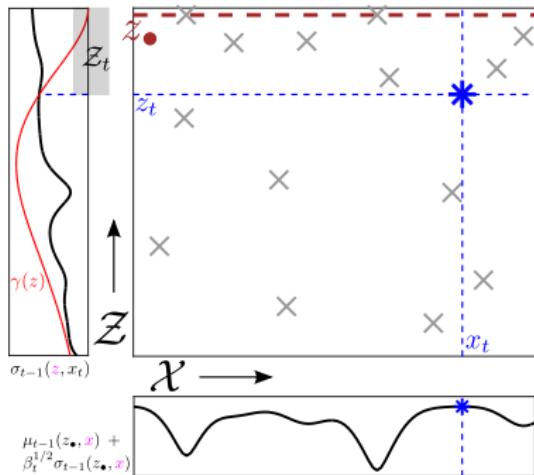
$$x_t = \operatorname{argmax}_{x \in \mathcal{X}} \mu_{t-1}(z_\bullet, x) + \beta_t^{1/2} \sigma_{t-1}(z_\bullet, x)$$

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

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

Algorithm: BOCA

(Kandasamy et al. ICML 2017)



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

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

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

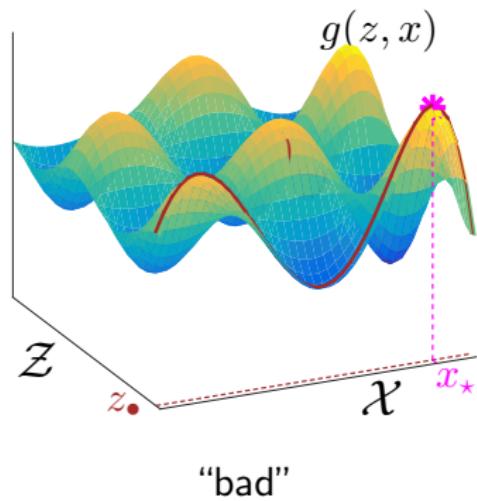
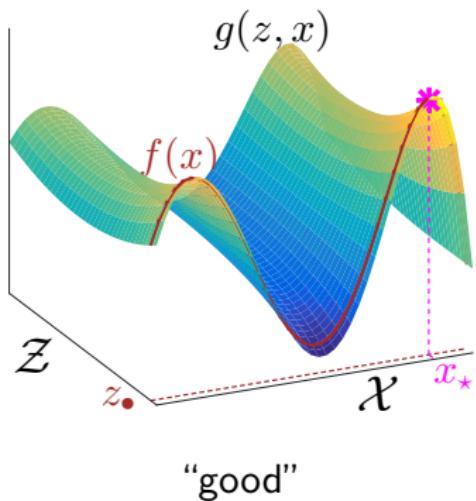
(1) $x_t \leftarrow$ maximise upper confidence bound for $f(x) = g(z_\bullet, x)$.

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

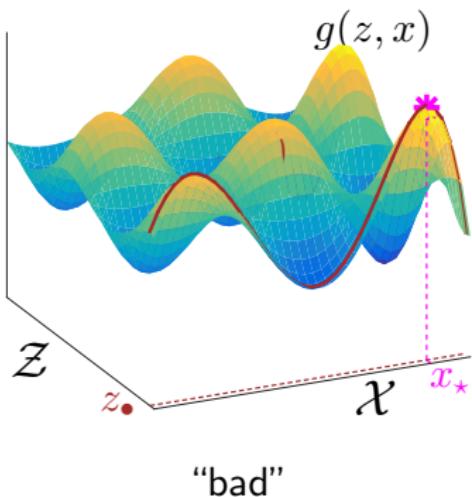
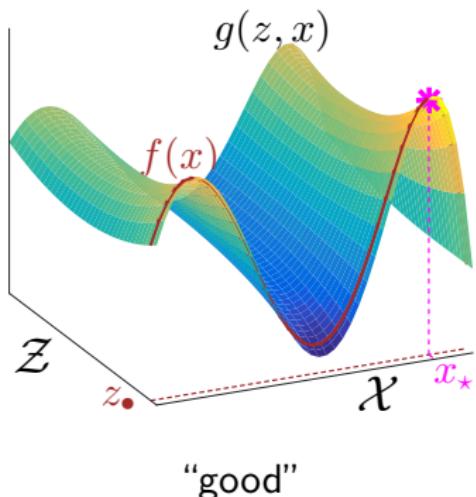
(2) $\mathcal{Z}_t \approx \{z_\bullet\} \cup \left\{ z : \sigma_{t-1}(z, x_t) \geq \gamma(z) = \left(\frac{\lambda(z)}{\lambda(z_\bullet)} \right)^q \xi(z) \right\}$

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

Theoretical Results for BOCA



Theoretical Results for BOCA

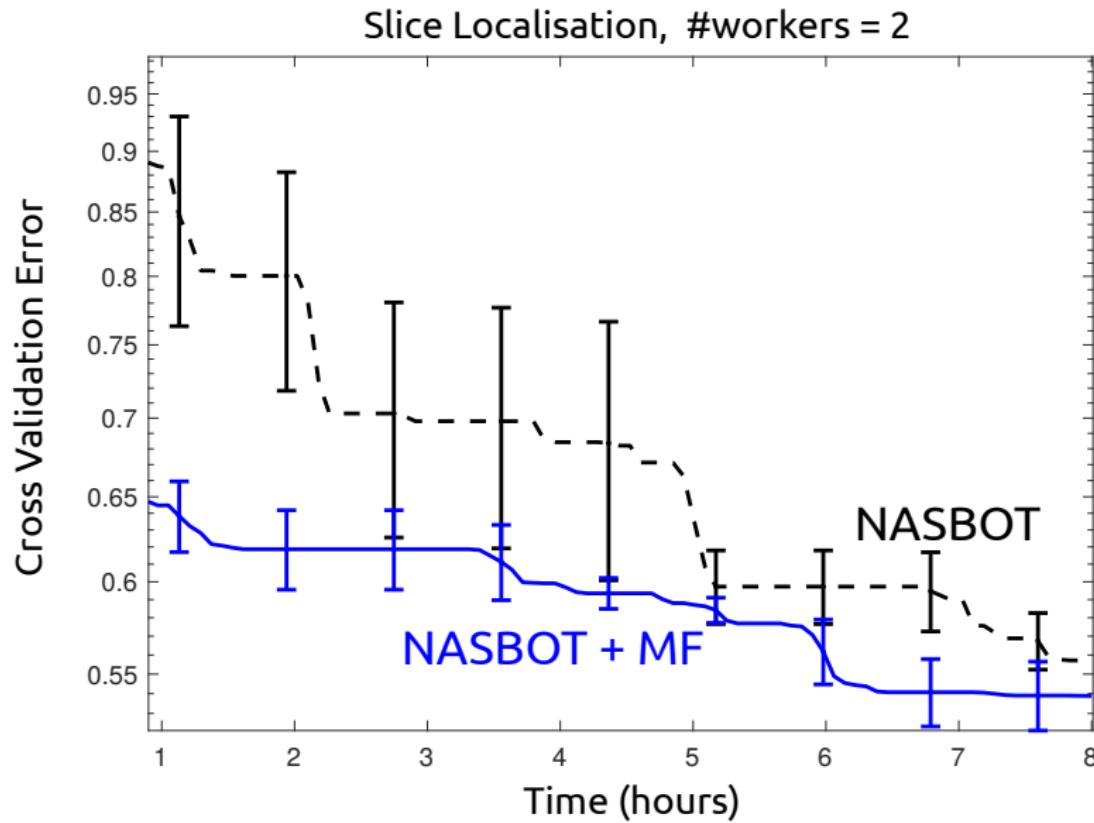


Theorem: (Informal)

(Kandasamy et al. ICML 2017)

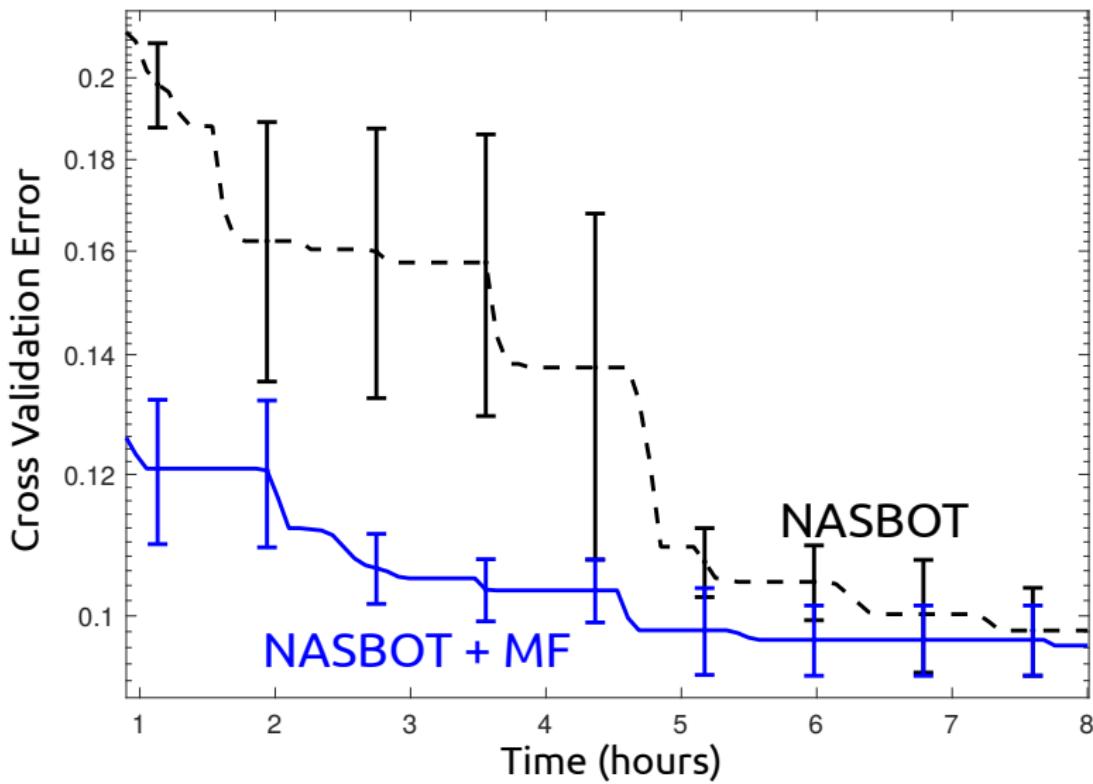
BOCA does better, i.e. achieves better Simple regret, than GP-UCB. The improvements are better in the "good" setting when compared to the "bad" setting.

NASBOT with BOCA



NASBOT with BOCA

Indoor Location, #workers = 2

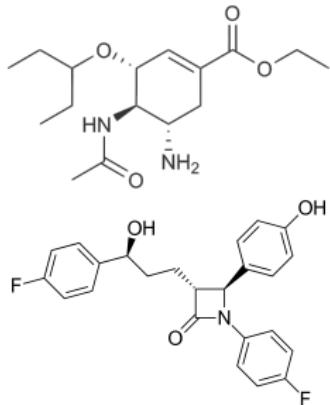


Summary

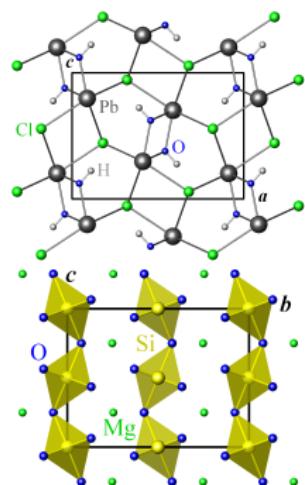
- ▶ Neural Architecture Search: finding the best deep neural network architecture for a given problem.
- ▶ NASBOT: A GP based Bayesian optimisation framework for neural architecture search.
- ▶ OTMANN: A pseudo-distance on the space of neural networks.
- ▶ Faster tuning when we combine NASBOT with multi-fidelity Bayesian optimisation.

Bayesian Optimisation on other graphical structures

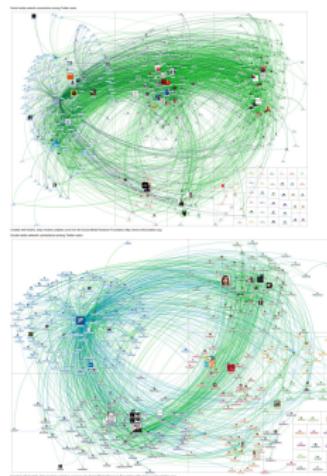
Drug Discovery with Small molecules



Crystal Structures



Social networks & viral marketing





Barnabás



Eric



Gautam



Jeff



Karun



Willie

Thank You

dragonfly.github.io
github.com/kirthevasank/nasbot

Appendix

From distance to kernel

From distance to kernel

Define the normalised distance,

$$\bar{d}(\mathcal{G}_1, \mathcal{G}_2) = \frac{d(\mathcal{G}_1, \mathcal{G}_2)}{\text{tm}(\mathcal{G}_1) + \text{tm}(\mathcal{G}_2)}$$

where $\text{tm}(\mathcal{G}_i) = \sum_{u \in \text{layers}(\mathcal{G}_1)} \ell m(u)$ is the total mass of the network.

From distance to kernel

Define the normalised distance,

$$\bar{d}(\mathcal{G}_1, \mathcal{G}_2) = \frac{d(\mathcal{G}_1, \mathcal{G}_2)}{\text{tm}(\mathcal{G}_1) + \text{tm}(\mathcal{G}_2)}$$

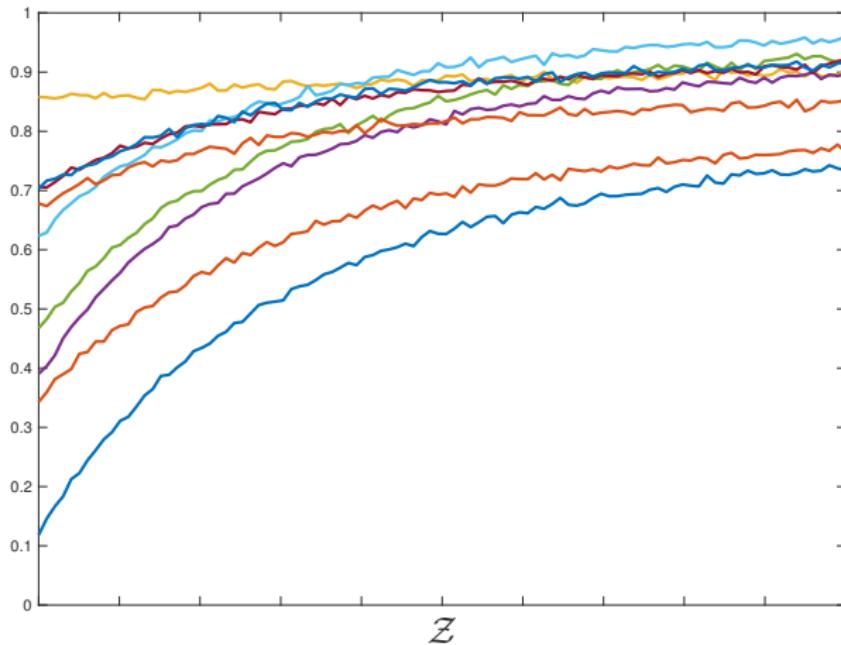
where $\text{tm}(\mathcal{G}_i) = \sum_{u \in \text{layers}(\mathcal{G}_1)} \ell m(u)$ is the total mass of the network.

Given OTMANN distances d, \bar{d} , we use,

$$\kappa(\mathcal{G}_1, \mathcal{G}_2) = \alpha e^{-\beta d(\mathcal{G}_1, \mathcal{G}_2)} + \bar{\alpha} e^{-\bar{\beta} \bar{d}(\mathcal{G}_1, \mathcal{G}_2)}$$

as the “kernel”.

Exponential Decay Kernel for Multi-fidelity BO



$$\kappa_{\mathcal{Z}}(u, u') = \frac{1}{(1 + u + u')^{\alpha}}$$