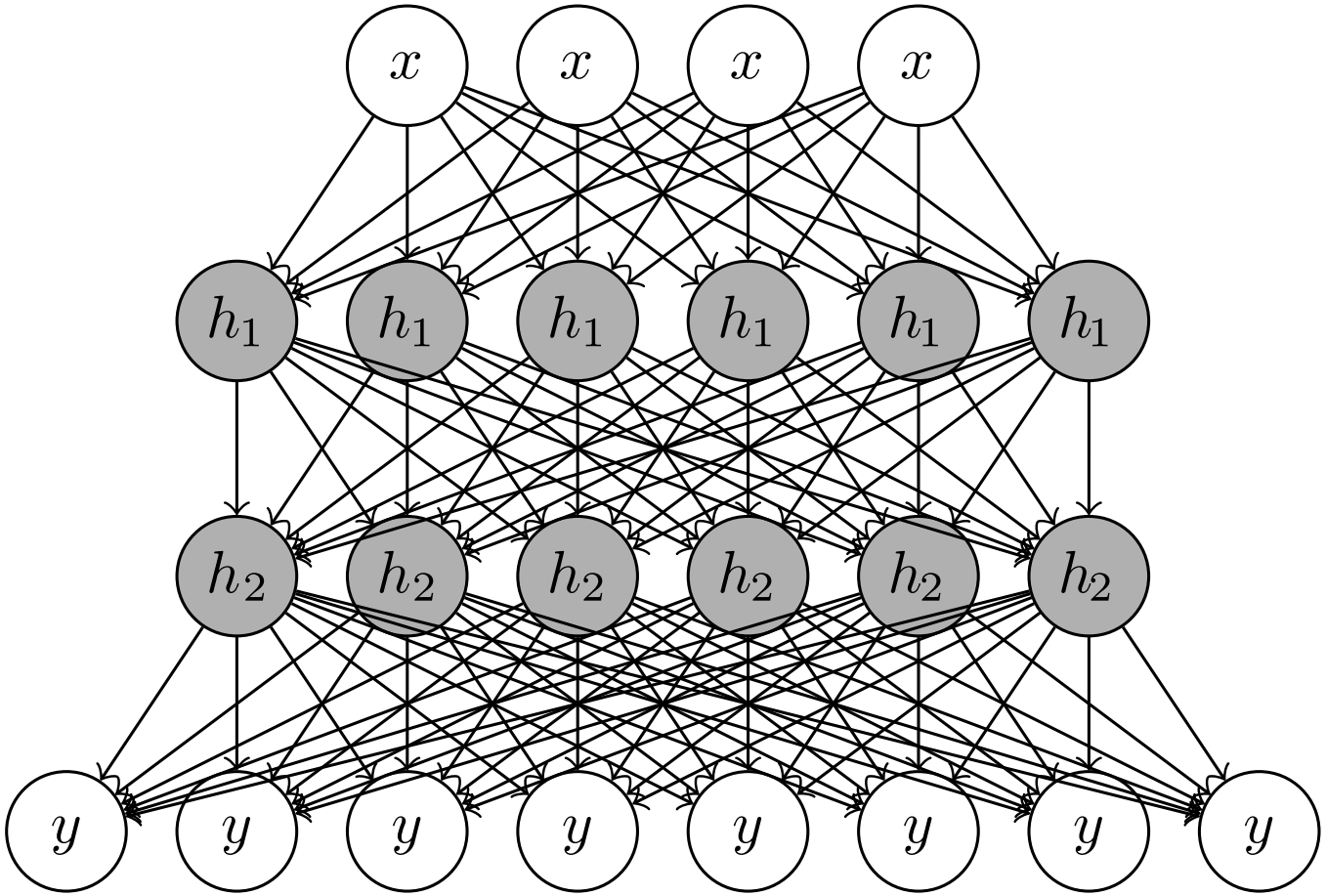
Deep Learning in the Function space

NeurIPS 2020 Nairobi Meetup
10 Dec. 2020

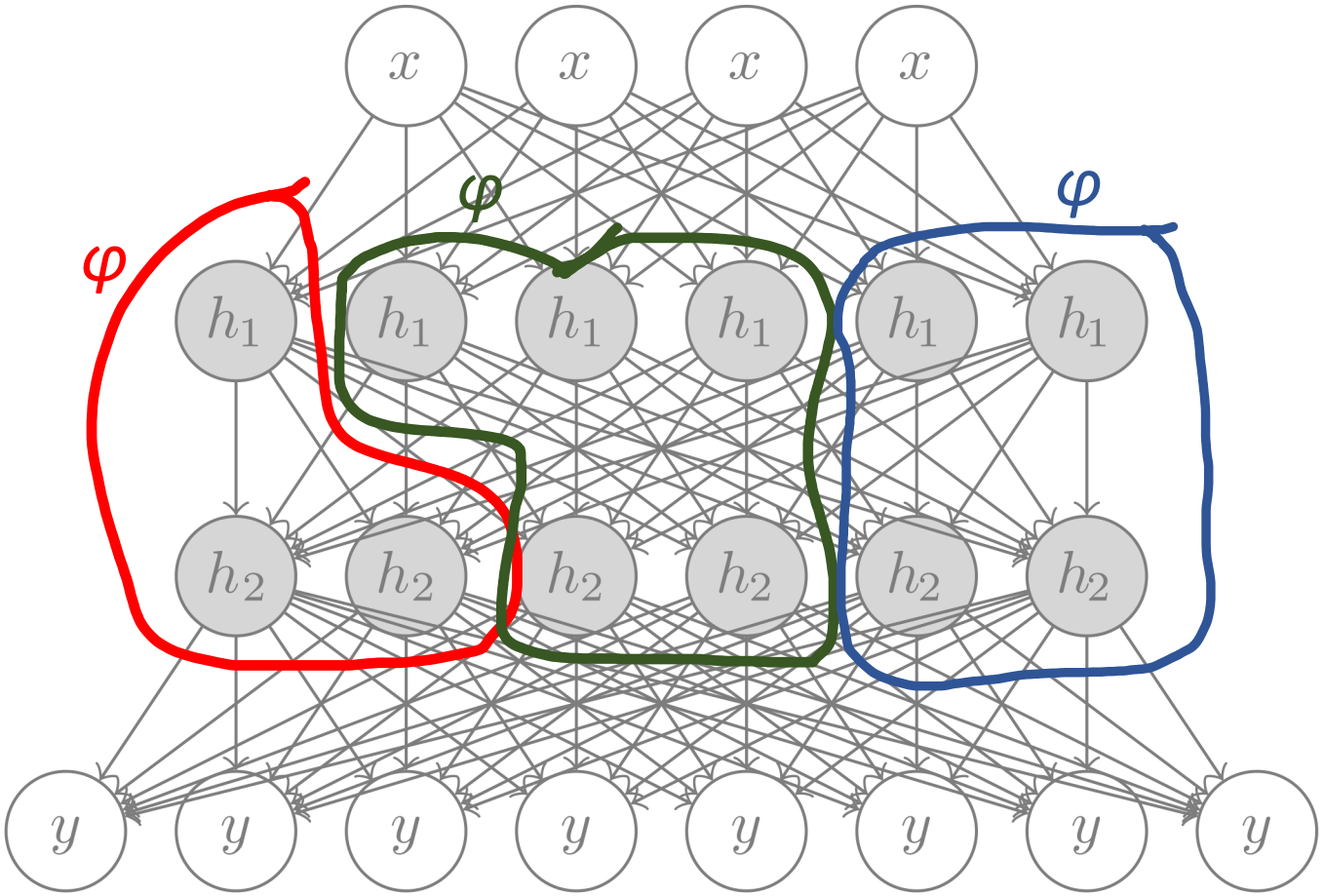
Andreas Damianou

What are the functional properties of a trained neural network?



$$y = f(x; w)$$

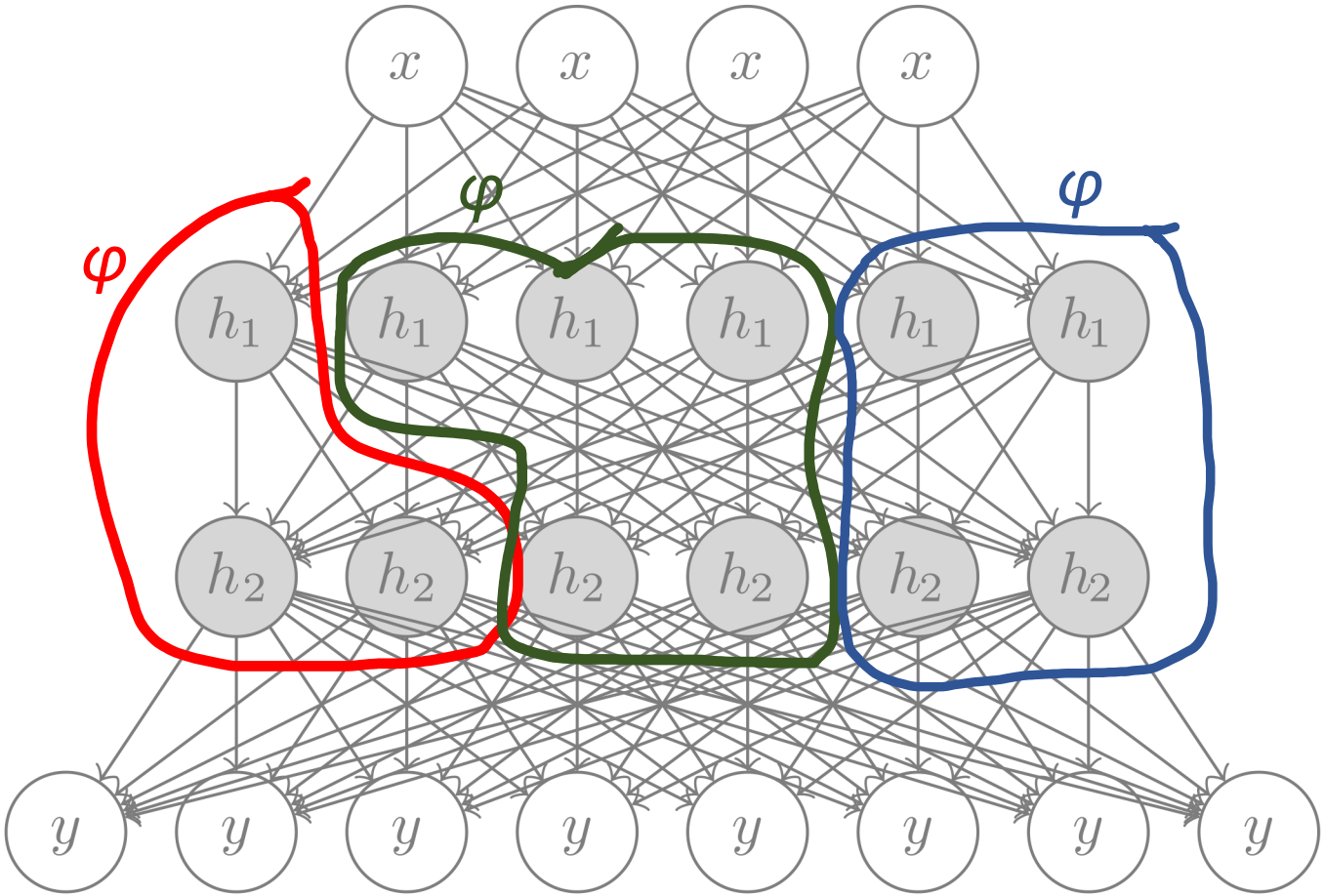
Sub-networks… sub-functions?



$$y = f(x; w)$$

Lottery Ticket Hypothesis, Frankle & Garbin, 2019

Sub-networks… sub-functions?

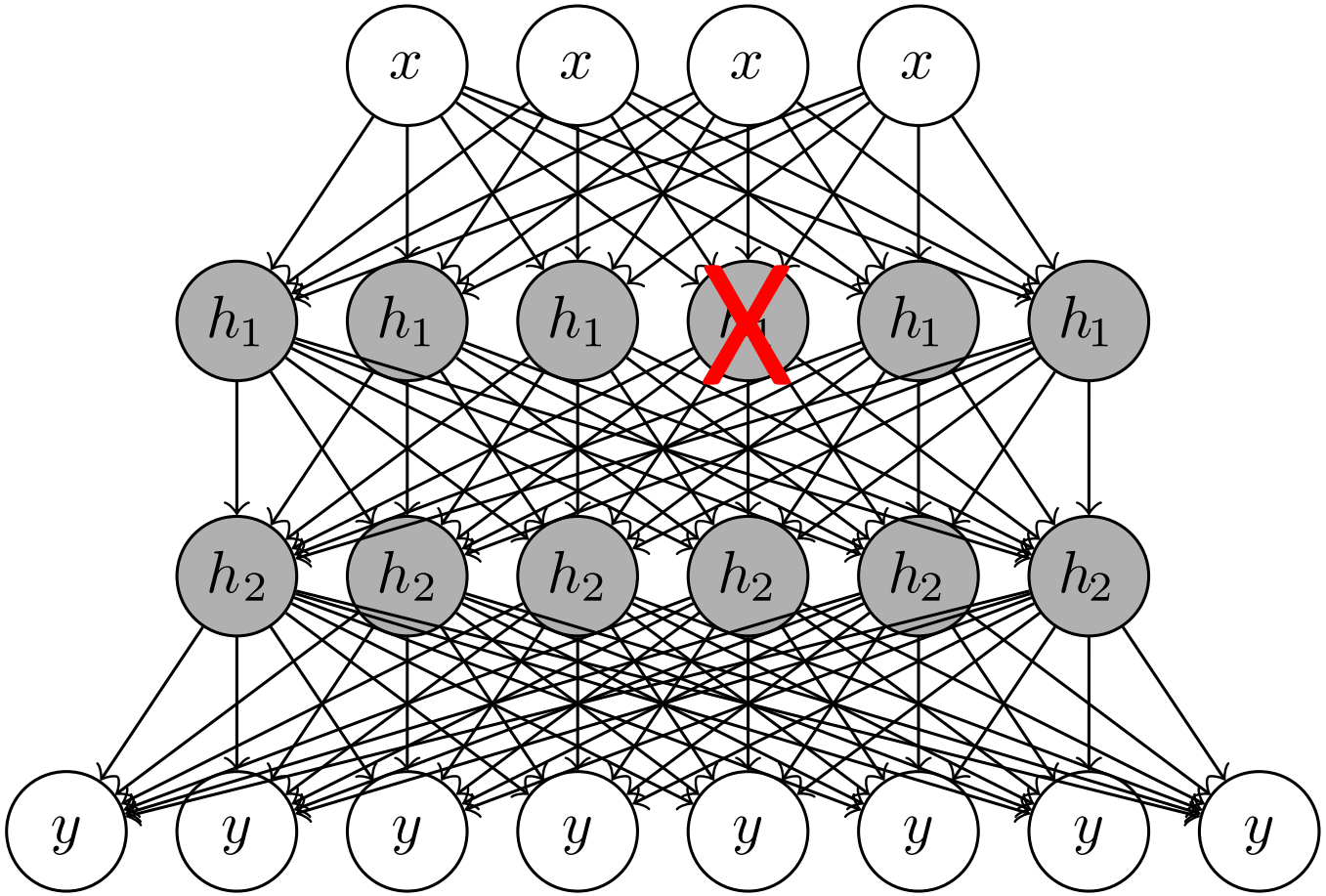


$$y = f(x; w)$$

$$f = \varphi \circ \varphi \circ \varphi$$

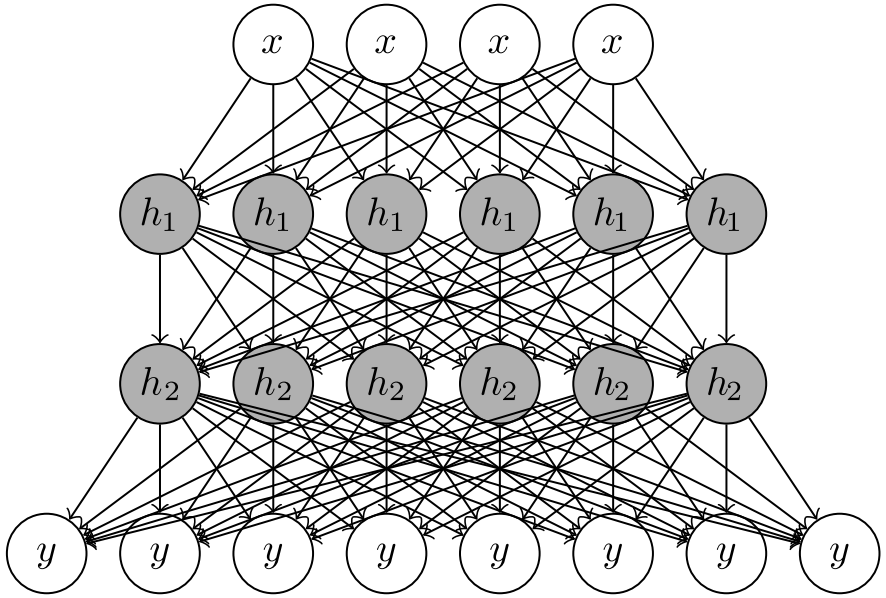
Lottery Ticket Hypothesis, Frankle & Garbin, 2019

How does the function change if I remove a node?



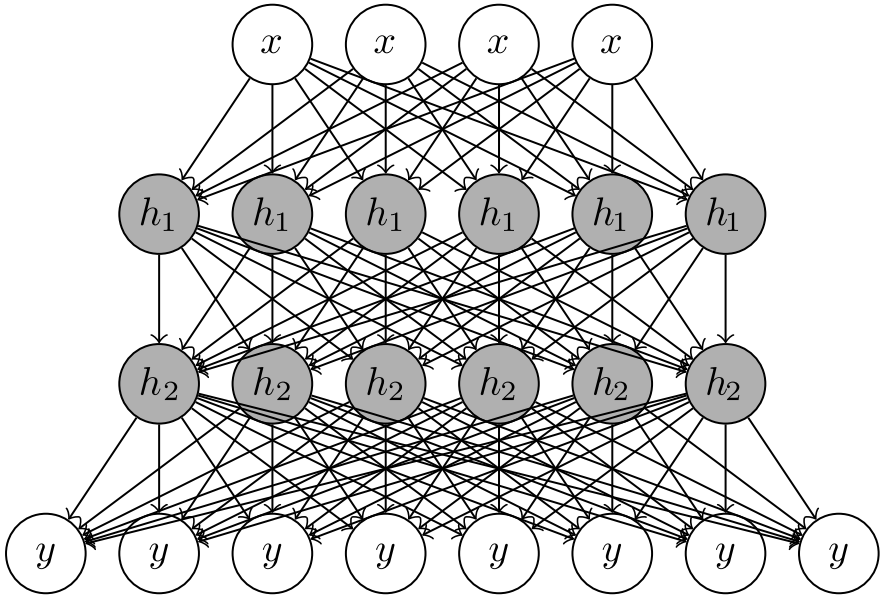
$$y = f'(x; w')$$

It’s not (only) about structure



$$y = f(x; w)$$

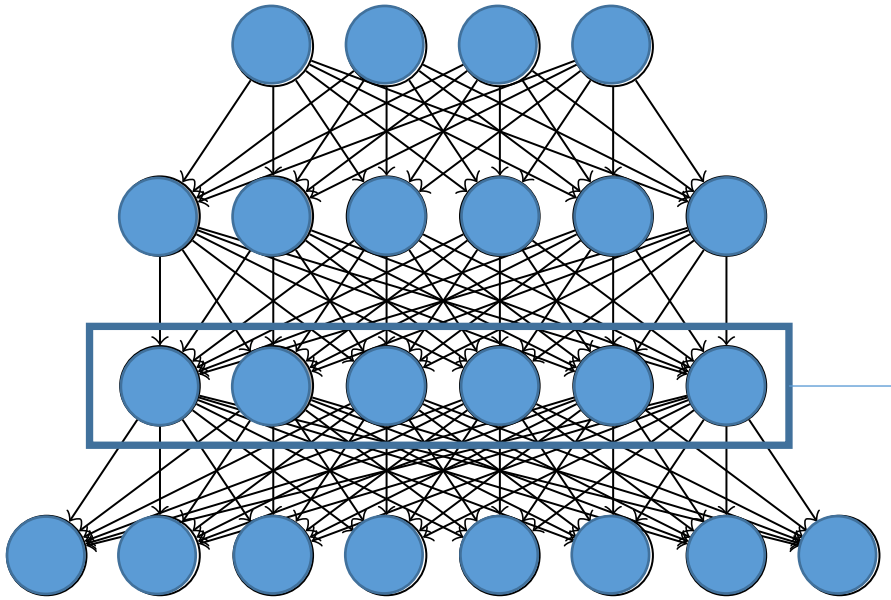
(optimized for **5** epochs)

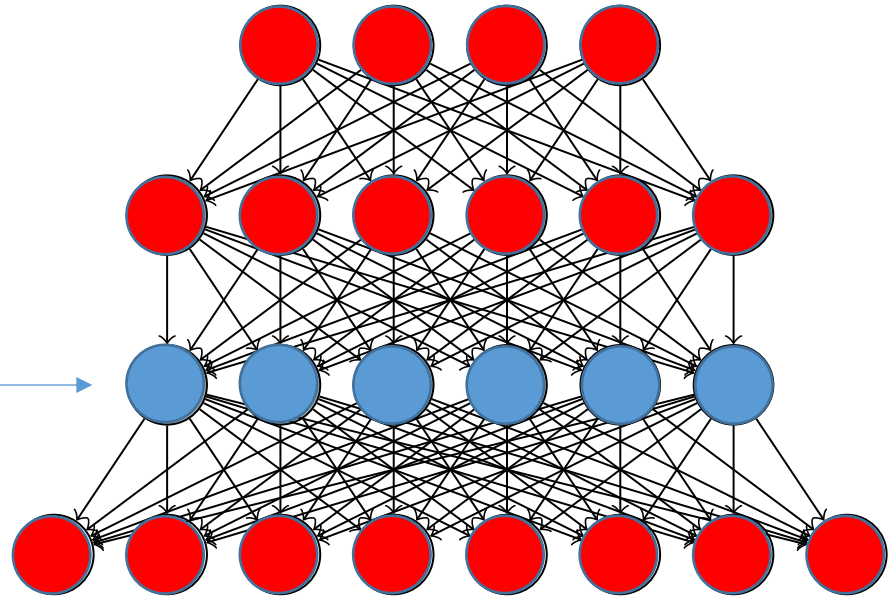


$$y = f'(x; w')$$

(optimized for **90** epochs)

What does it mean to “transfer” a layer?





$y = f(x; \mathbf{w})$

$y = f(x; [\mathbf{w}, \mathbf{w}])$

Cutting out the middleman… Dealing with functions directly!

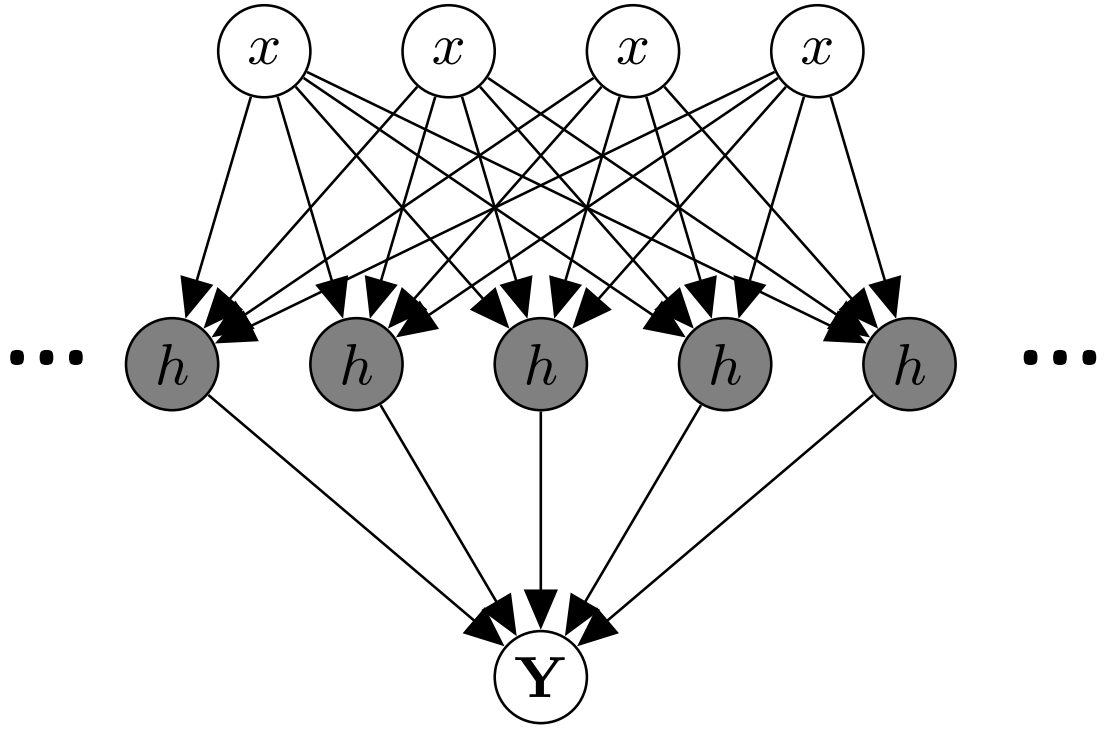
Deep learning over functions.

- Intuitive composition of signals — $f_1 \circ f_2 \quad \text{vs} \quad f([w_1, w_2])$
- Adding prior knowledge/assumptions — $p(f) \quad \text{vs} \quad p(w)$

NN at initialization is a GP (infinite width limit)

- Limit of infinite num. parameters:
  Optimization of finite num. parameters → Integration of infinite model

- *Closed form* solution for modelled function class!

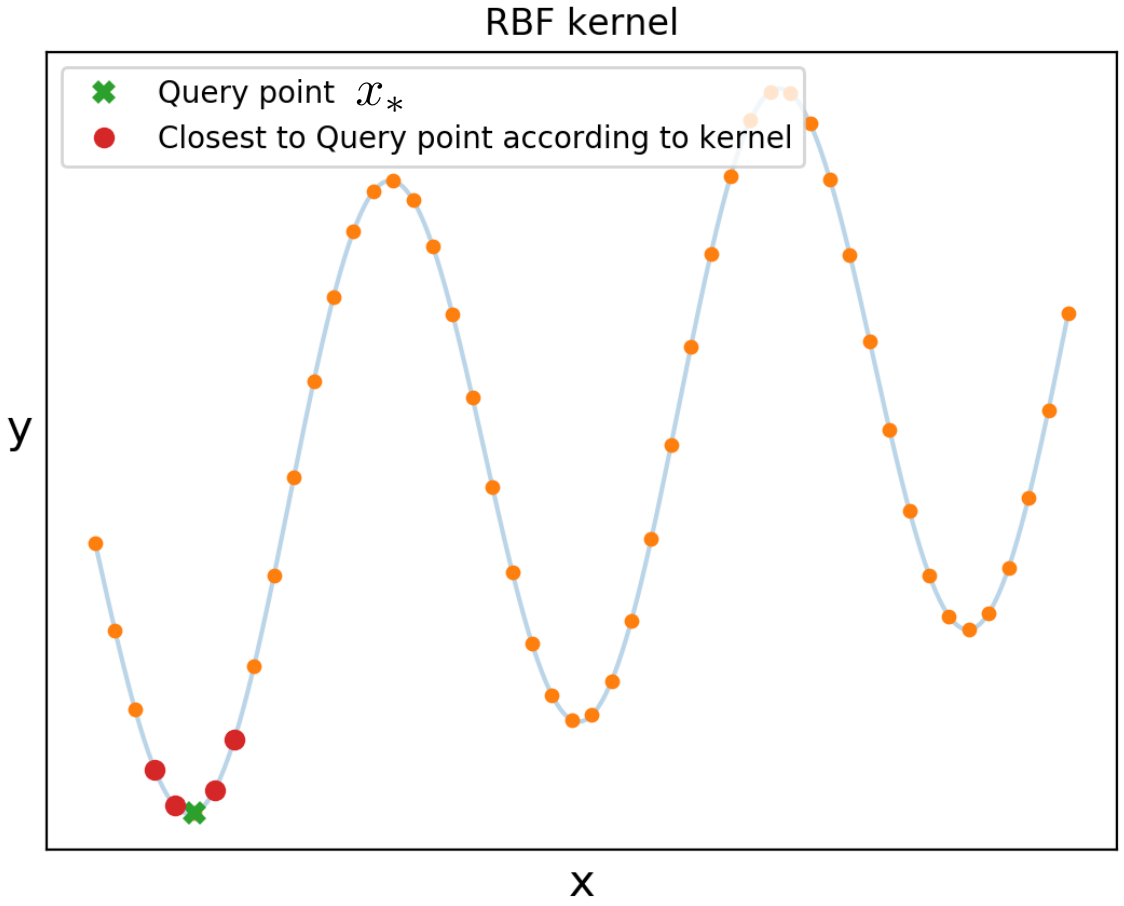


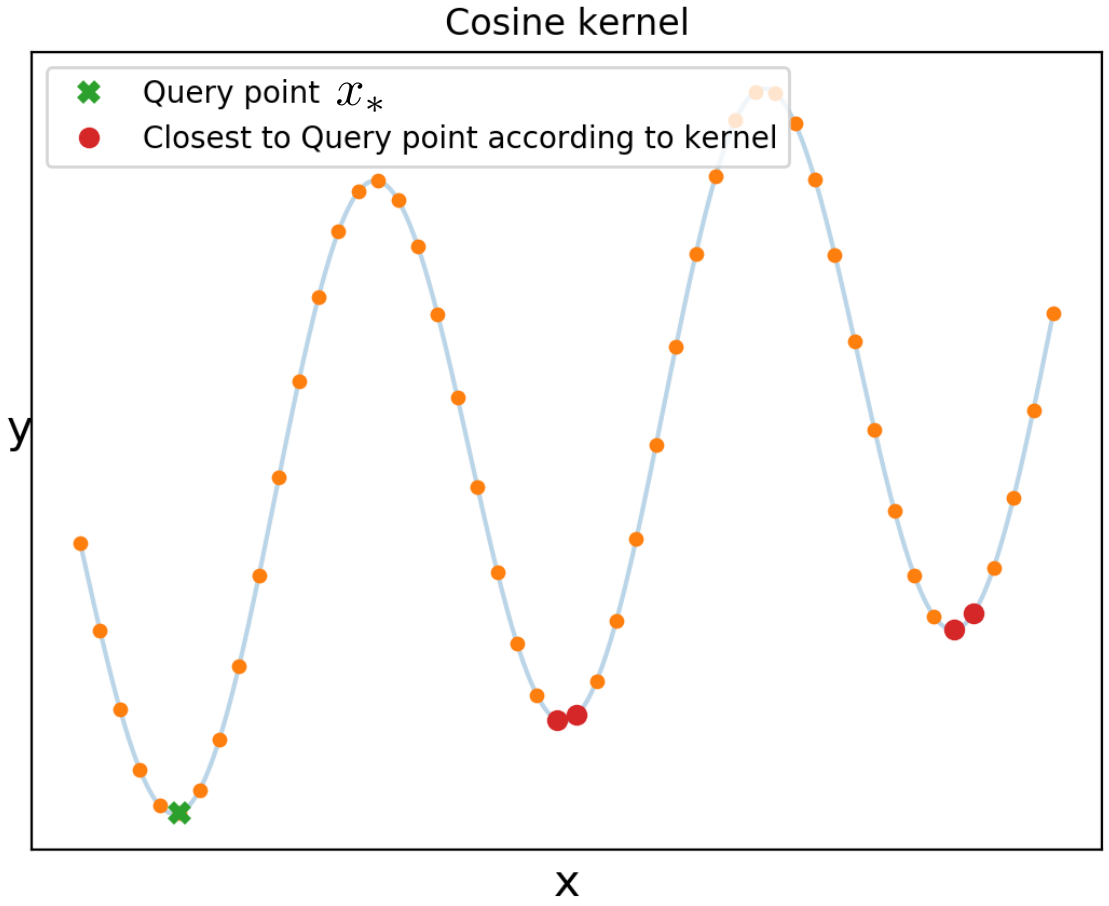
$$\mathcal{GP}\left(\mu(x), K(x,x)\right)$$

1 layer: Neal 1994; Williams 1997
>1 layers: Matthews et al. 2018, Lee et al. 2018

GP as kernel machine

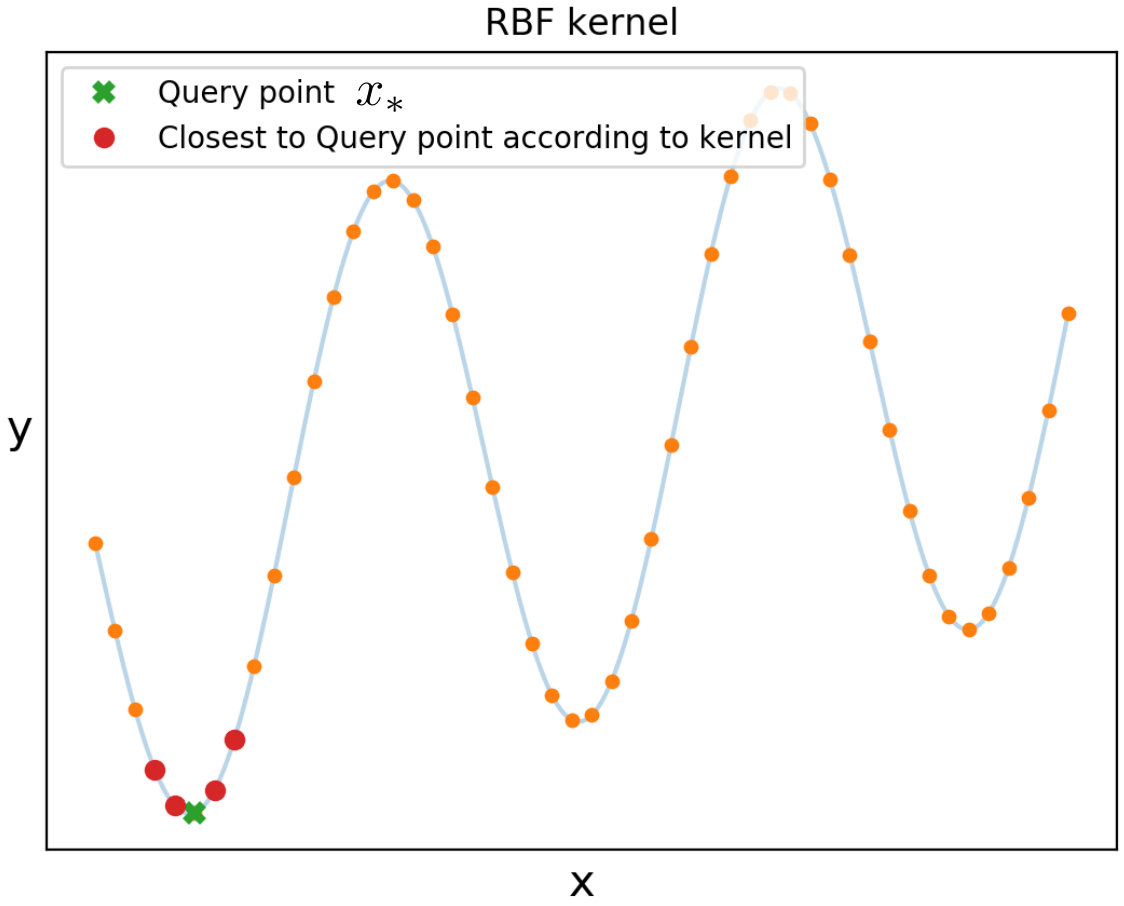
See also: adamian.github.io/talks/Damianou_GP_tutorial.html

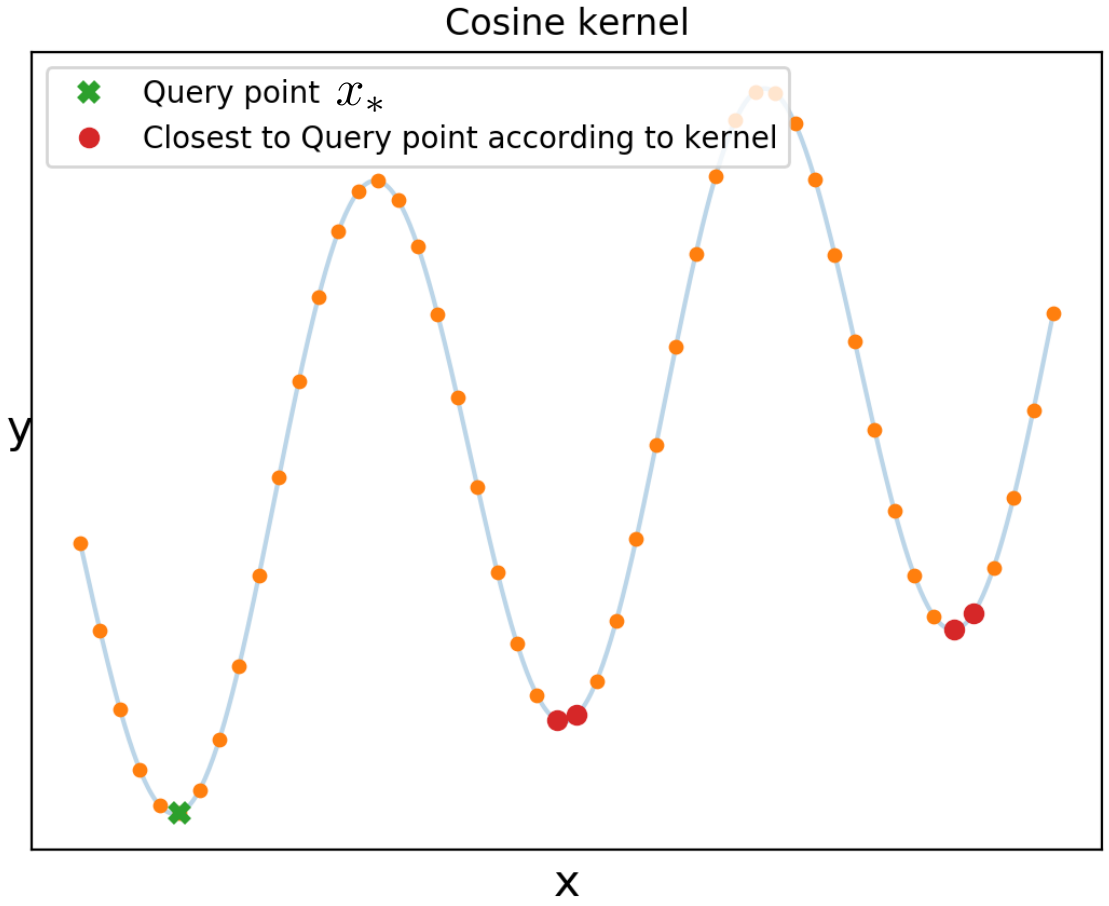




GP as kernel machine

See also: adamian.github.io/talks/Damianou_GP_tutorial.html





Predictions: $y_* \sim \mathcal{N}\left(k_* \tilde{\mathbf{K}}^{-1}\mathbf{y}, \mathbf{C}\right)$ where $k_* = [k(x_*, x_1), k(x_*, x_2), \ldots k(x_*, x_n)]$

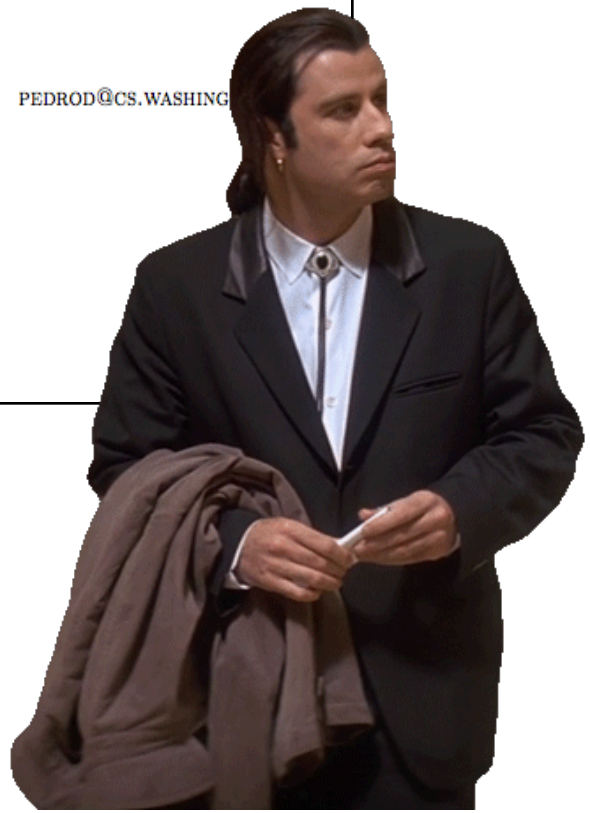
But… how can we then hierarchical features?

> *Have we thrown the baby out with the bathwater?*
>
> *David MacKay.*

Every Model Learned by Gradient Descent Is Approximately a Kernel Machine

Pedro Domingos PEDROD@CS.WASHING
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350, USA

30 Nov. 2020



Posted by u/thegregyang 1 day ago

[R] Wide Neural Networks are Feature Learners, Not Kernel Machines

Research

Hi Reddit,

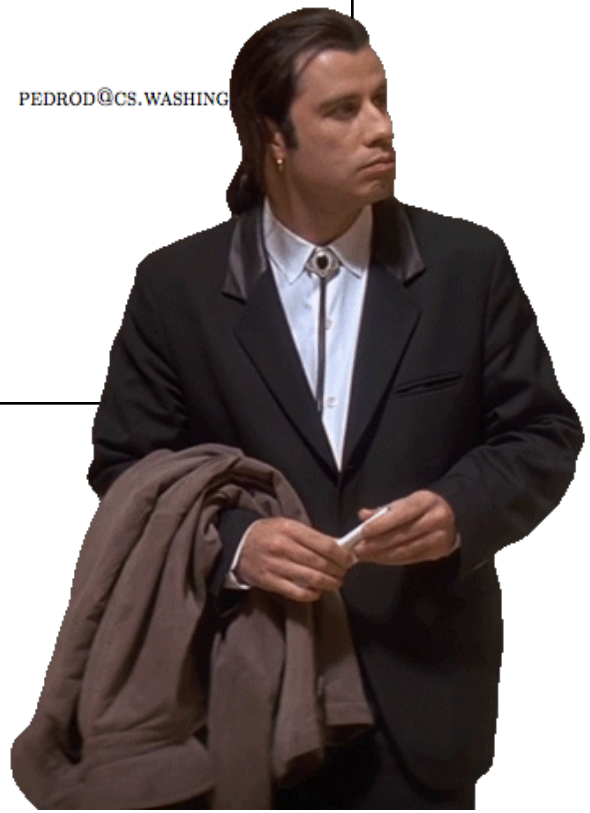
I'm excited to share with you my new paper [2011.14522] Feature Learning in Infinite-Width Neural Networks (arxiv.org).

6 Dec. 2020

Every Model Learned by Gradient Descent Is Approximately a Kernel Machine

Pedro Domingos PEDROD@CS.WASHING
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350, USA

30 Nov. 2020



Posted by u/thegregyang 1 day ago

[R] Wide Neural Networks are Feature Learners, Not Kernel Machines

Research

Hi Reddit,

I'm excited to share with you my new paper [2011.14522] Feature Learning in Infinite-Width Neural Networks (arxiv.org).

6 Dec. 2020

> *According to the hype of 1987, neural networks were meant to be intelligent models that discovered features and patterns in data. Gaussian processes in contrast are simply smoothing devices. How can Gaussian processes possibly replace neural networks? Were neural networks over-hyped, or have we underestimated the power of smoothing methods? I think both these propositions are true.*
>
> *David MacKay.*

Every Model Learned by Gradient Descent Is Approximately a Kernel Machine

Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350, USA

30 Nov. 2020

Posted by u/thegregyang 1 day ago

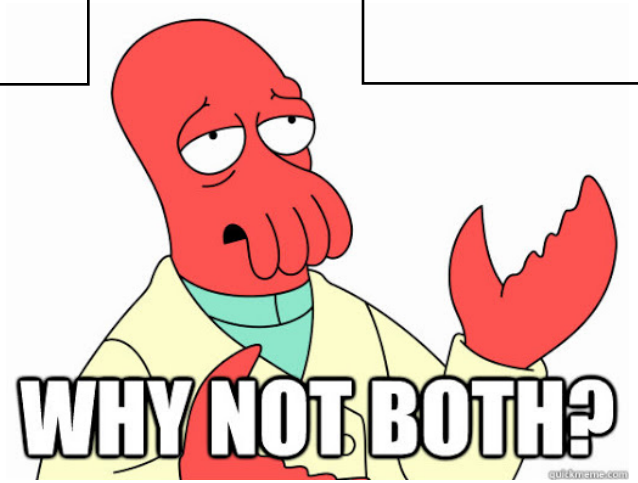
[R] Wide Neural Networks are Feature Learners, Not Kernel Machines

Research

Hi Reddit,

I'm excited to share with you my new paper [2011.14522] Feature Learning in Infinite-Width Neural Networks (arxiv.org).

6 Dec. 2020

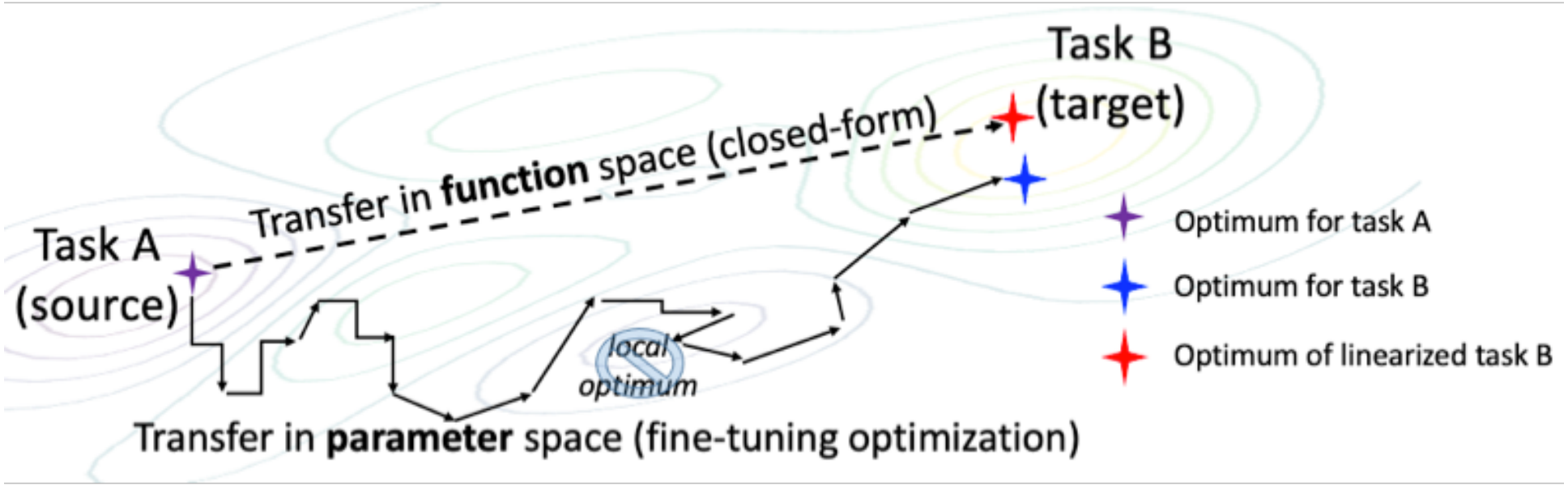


- Nuance: wide NNs result in *data-dependent* kernels (≈ "functional features" learned from data).
- So, we can have both: hierarchical feature learning and function space inference.

NNs fine-tuned in function space

github.com/amzn/xfer/

- Fit DNN's parameters w on data X
- Encode the trained DNN *function* as a kernel $k_{DNN}(X, X; w)$
- For new inputs x_* predict with a GP having kernel k_{DNN}



Jacot et a. 2019, Maddox et al. 2019

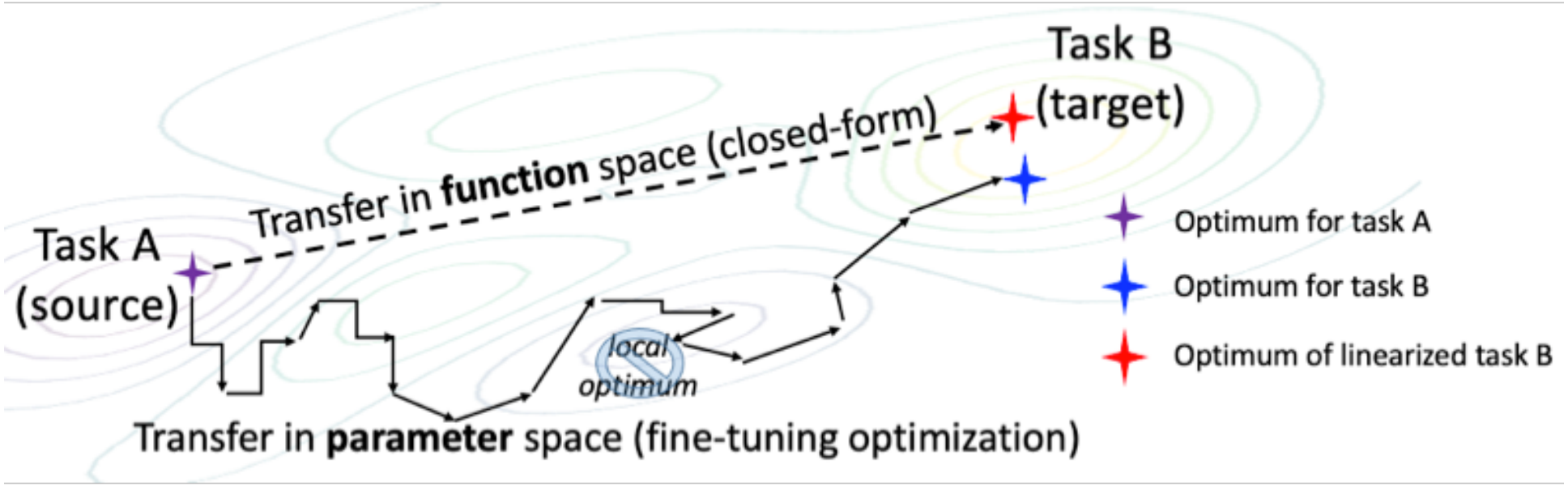
NNs fine-tuned in function space

github.com/amzn/xfer/

- Fit DNN's parameters w on data X
- Encode the trained DNN *function* as a kernel
- For new inputs x_* predict with a GP having kernel k_{DNN}

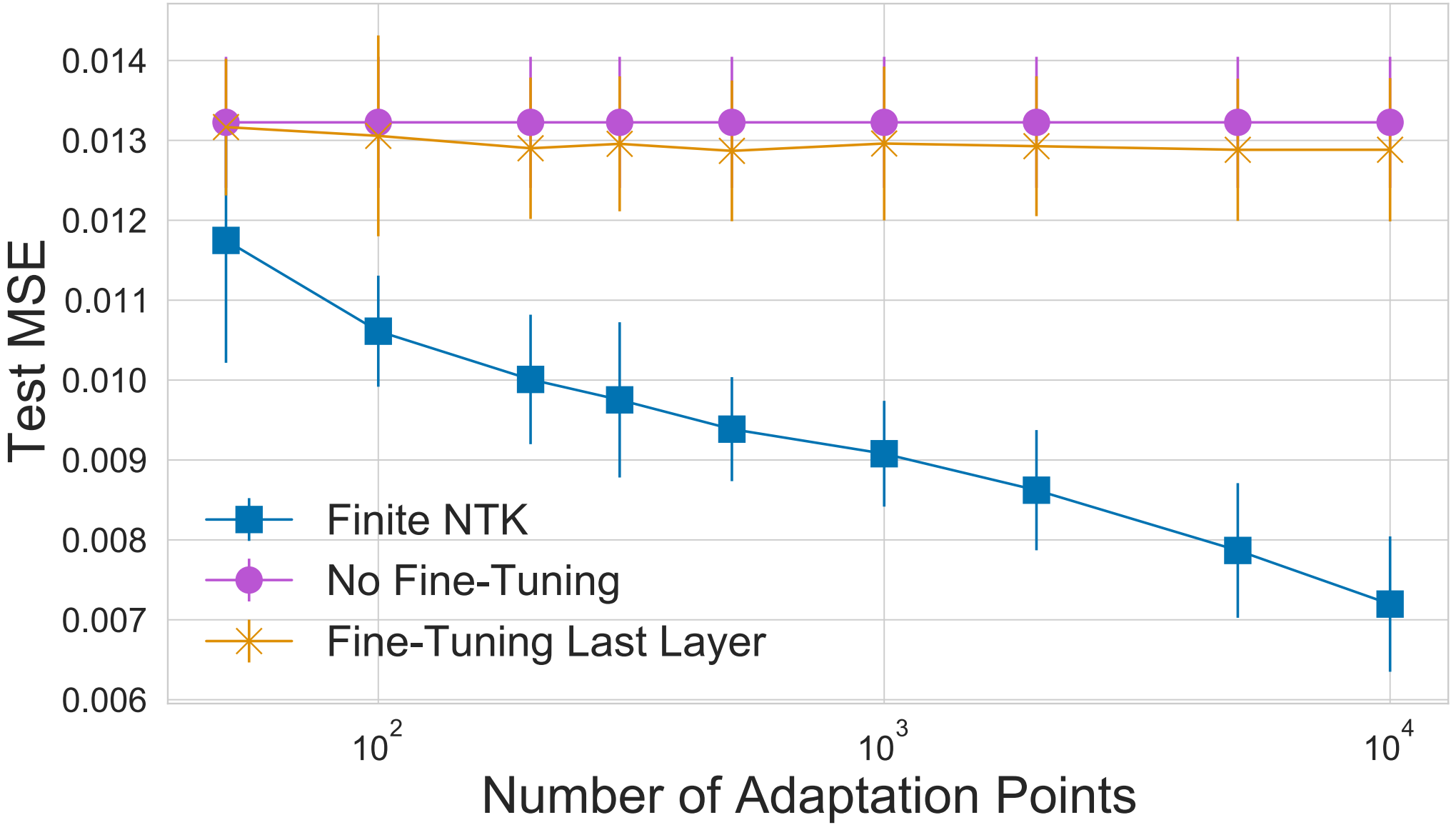
Kernel governing SGD dynamics around convergence

$$k_{DNN}(X, X; w) = J(x; w)^T J(x'; w)$$



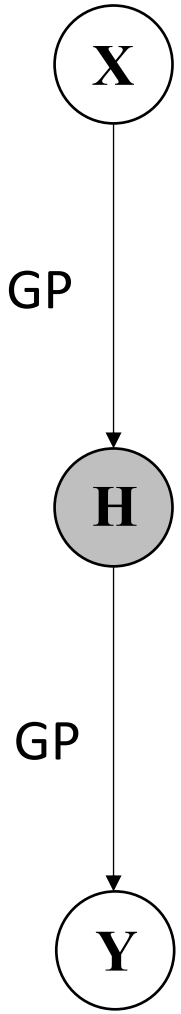
Jacot et a. 2019, Maddox et al. 2019

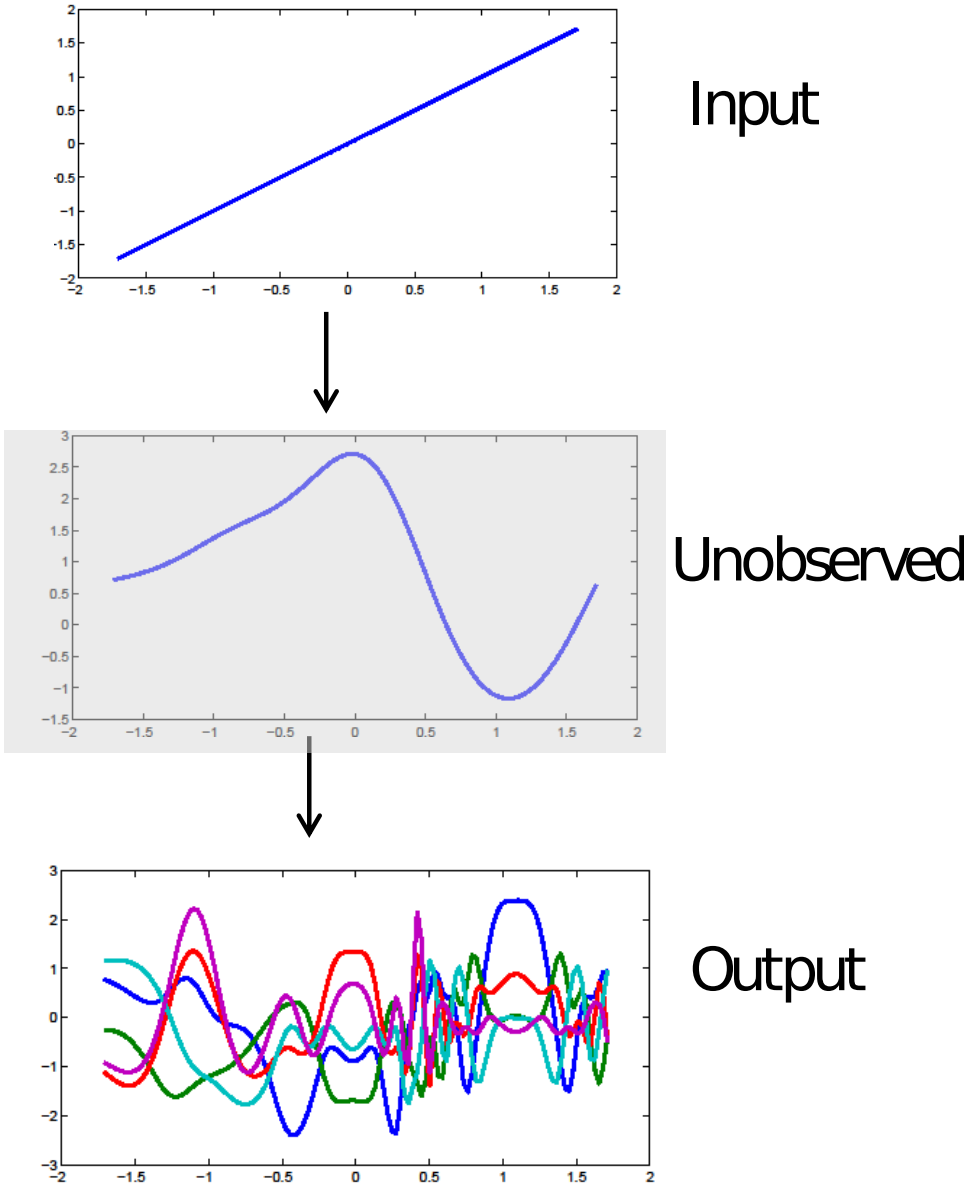
github.com/amzn/xfer/



Hierarchical function learning with Deep GPs

Hierarchical learning can also happen in the function space directly!





See: *Damianou and Lawrence 2013, Damianou 2015*

Take home messages

- DNNs: hierarchical concept learning, learns well from large data
- Function space modeling: interpretable, avoids local optima, data efficient
- Can combine both by considering the kernel limit of trained DNNs
- and/or we can perform hierarchical learning on the function space directly

Questions?

See also: github.com/amzn/xfer/

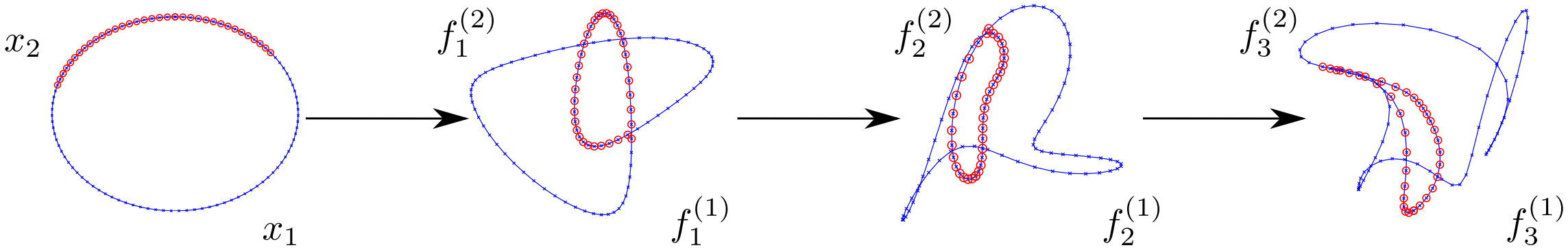
Thanks to my collaborators:

- Wesley Maddox
- Pablo G. Moreno
- Andrew Wilson
- Shuai Tang
- Neil Lawrence
- Jordan Massiah

Appendix I: DGPs

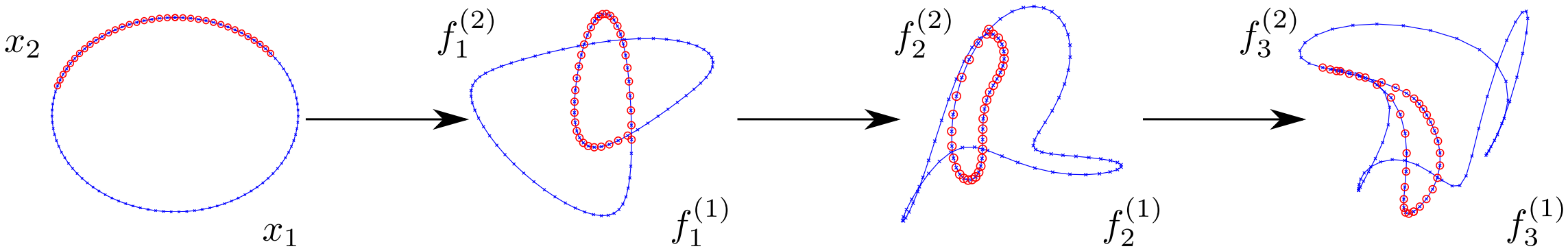
Feature learning

Features are learned as “knots” in the latent space, carried over from layer to layer.



Feature learning

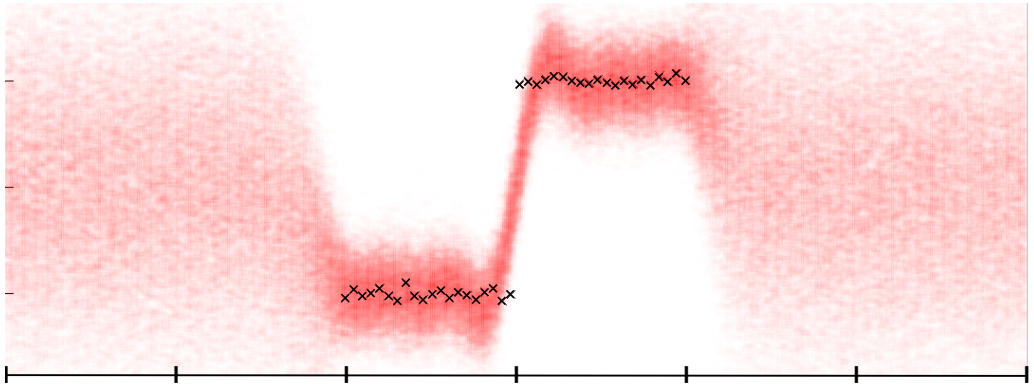
Features are learned as “knots” in the latent space, carried over from layer to layer.



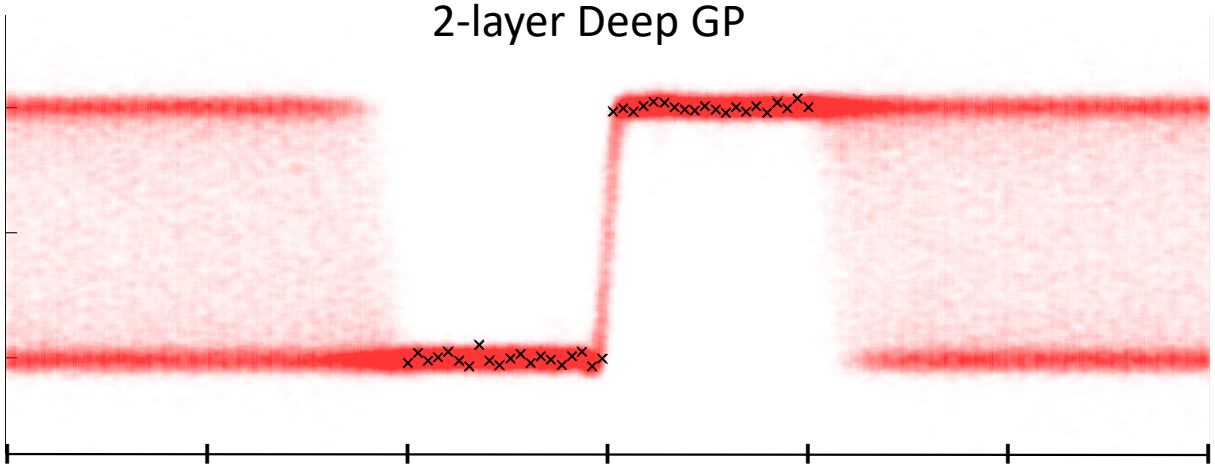
Narrow intermediate layers do give hierarchical feature learning.

Step function example

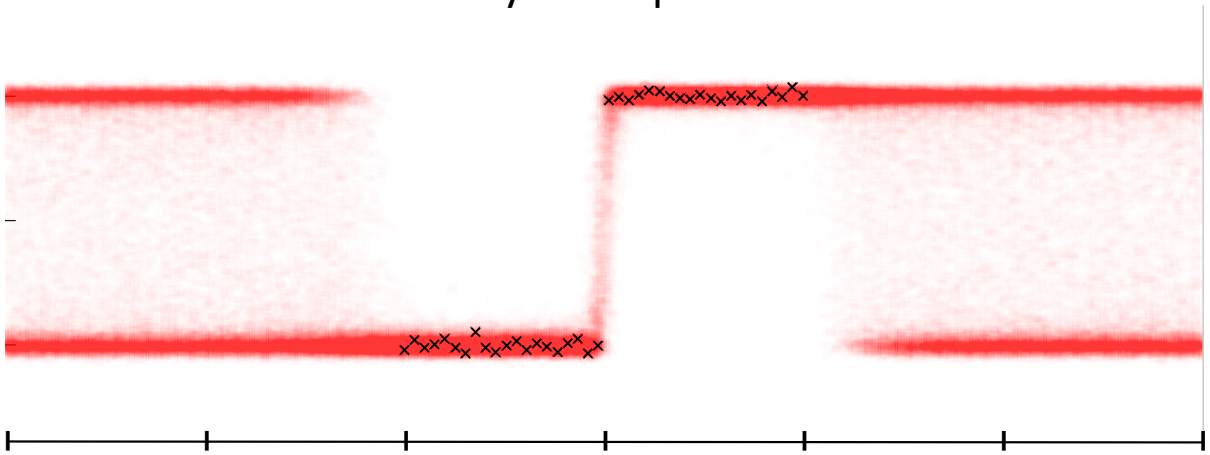
Single GP



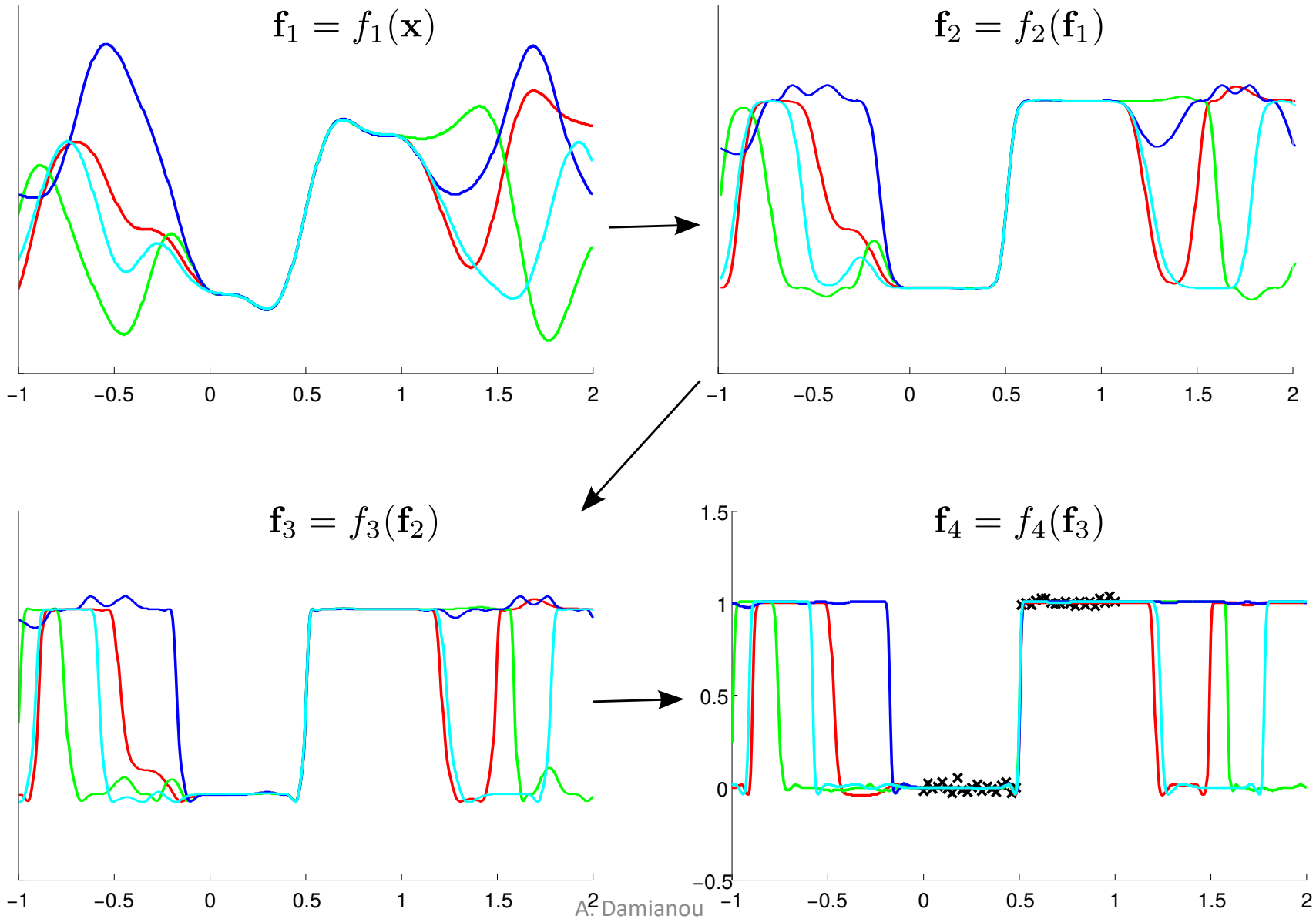
2-layer Deep GP



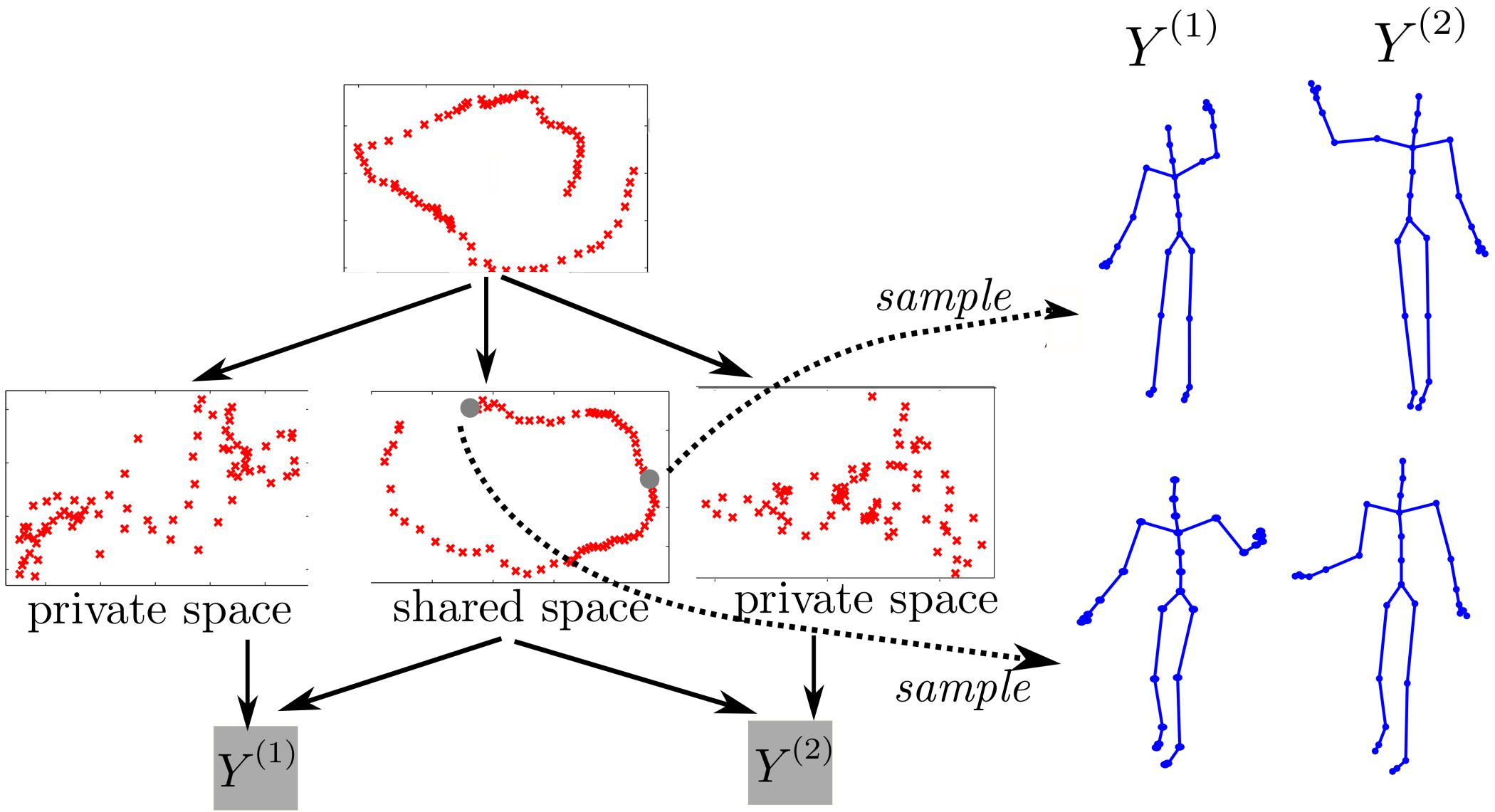
4-layer Deep GP



Successive warping to learn the step function



Unsupervised learning for multiple views



Appendix II: NTK

GP ⟷ DNN beyond initialization (and practical use)

Initialization	During training	**Convergence**
Neal ‘94 Lee et al. ‘18 Matthews et al. ‘18	Jacot et al. ‘18 NTK Lee et al. ‘19 Hayou et al. ‘19	Maddox et al. ‘19

Model: A degenerate GP from DNNs obtained at convergence.

W. Maddox, S. Tang, P. Moreno, A. Wilson, A. Damianou: *Fast Adaptation with Linearized Neural Networks*. 2019

DNN training dynamics in GD

$$w_{t+1} = w_t - \eta \nabla_w L(f(w_t)) \Rightarrow$$
$$\frac{w_{t+1} - w_t}{\eta} = -\nabla_w L(f(w_t)) \Rightarrow$$
$$\frac{dw(t)}{dt} = -\nabla_w f(w_t) \nabla_f L(f(w_t))$$

Then:

$$\frac{d\ f(w(t))}{dt} = -\underbrace{\nabla_w^\top f(w_t) \nabla_w f(w_t)}_{\text{NTK}} \nabla_f L(f(w_t))$$

DNN training dynamics in GD

$$w_{t+1} = w_t - \eta \nabla_w \underbrace{L(f(x; w_t))}_{\log p(y|f(x;w))} \Rightarrow \frac{w_{t+1} - w_t}{\eta} = -\nabla_w L(f(w_t)) \Rightarrow$$

$$\frac{dw(t)}{dt} = -\nabla_w L(f(w_t)) \tag{1}$$

But: $\frac{dL(f(w_t))}{dw_t} = \frac{L(f(w_t))}{df} \frac{df(w_t)}{dw_t}$ (chain rule). So writing nabla notation, (1) becomes:

$$\frac{dw(t)}{dt} = -\nabla_w f(w_t) \nabla_f L(f(w_t)) \tag{2}$$

Then: $\frac{d\ f(w(t))}{dt} = \frac{df(w(t))}{dw(t)} \frac{dw(t)}{dt} = -\underbrace{\nabla_w^\top f(w_t) \nabla_w f(w_t)}_{\text{NTK}} \nabla_f L(f(w_t))$

Notation, $\nabla_w f(x; w_t) = J_w(x)$

Appendix III: GPs

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$p(\underbrace{f_1, f_2, \cdots, f_n}_{\mathbf{f}_n}, \underbrace{f_{n+1}, f_{n+2}, \cdots, f_s}_{\mathbf{f}_s}) = p(\mathbf{f}_n, \mathbf{f}_s) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}).$$

with:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_n \\ \boldsymbol{\mu}_s \end{bmatrix} \text{ and } \mathbf{K} = \begin{bmatrix} \mathbf{K}_{nn} & \mathbf{K}_{ns} \\ \mathbf{K}_{sn} & \mathbf{K}_{ss} \end{bmatrix}$$

Marginalisation:

$$p(\mathbf{f}_n) = \int_{\mathbf{f}_s} p(\mathbf{f}_n, \mathbf{f}_s) \mathrm{d}\mathbf{f}_s = \mathcal{N}(\boldsymbol{\mu}_n, \mathbf{K}_{nn})$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$p(\underbrace{f_1, f_2, \cdots, f_n}_{\mathbf{f}_n}, \underbrace{f_{n+1}, f_{n+2}, \cdots, f_\infty}_{\mathbf{f}_\infty}) = p(\mathbf{f}_n, \mathbf{f}_\infty) \sim \mathcal{GP}(\boldsymbol{\mu}_\infty, \mathbf{K}_\infty).$$

with:

$$\boldsymbol{\mu}_\infty = \begin{bmatrix} \boldsymbol{\mu}_n \\ \cdots \end{bmatrix} \quad \text{and} \quad \mathbf{K}_\infty = \begin{bmatrix} \mathbf{K}_{nn} & \cdots \\ \cdots & \cdots \end{bmatrix}$$

Marginalisation:

$$p(\mathbf{f}_n) = \int_{\mathbf{f}_\infty} p(\mathbf{f}_n, \mathbf{f}_\infty) \mathrm{d}\mathbf{f}_\infty = \mathcal{N}(\boldsymbol{\mu}_n, \mathbf{K}_{nn})$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$\mathbf{K}_\infty = \begin{bmatrix} \mathbf{K}_{nn} & \cdots \\ \cdots & \cdots \end{bmatrix}$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$\mathbf{K}_\infty = \begin{bmatrix} \mathbf{K}_{nn} & \cdots \\ \cdots & \cdots \end{bmatrix}$$

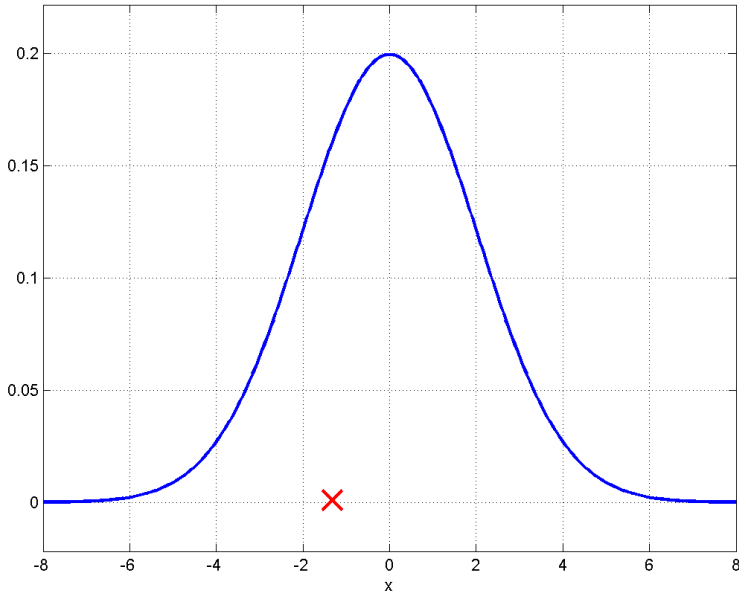
$$\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j; \theta)$$

Train a GP means fitting θ

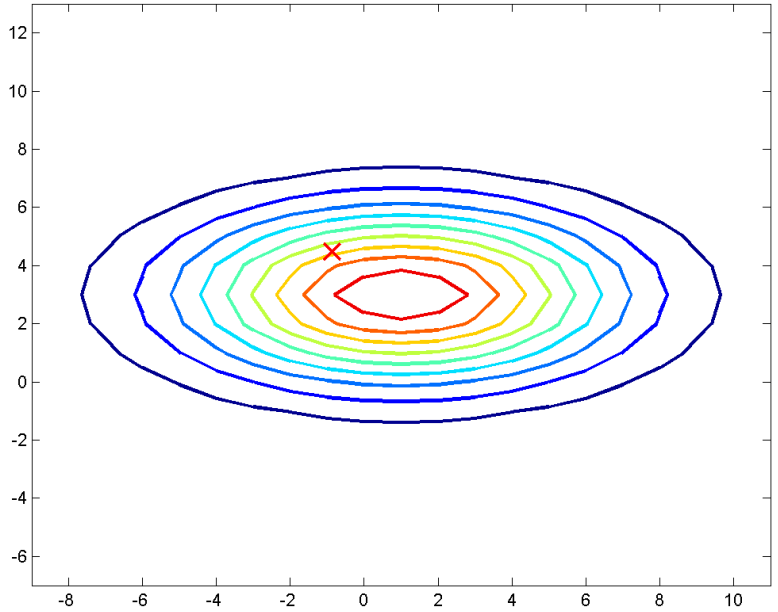
Introducing Gaussian Processes:

- A Gaussian distribution depends on a mean and a covariance matrix.
- A Gaussian process depends on a mean and a covariance function.

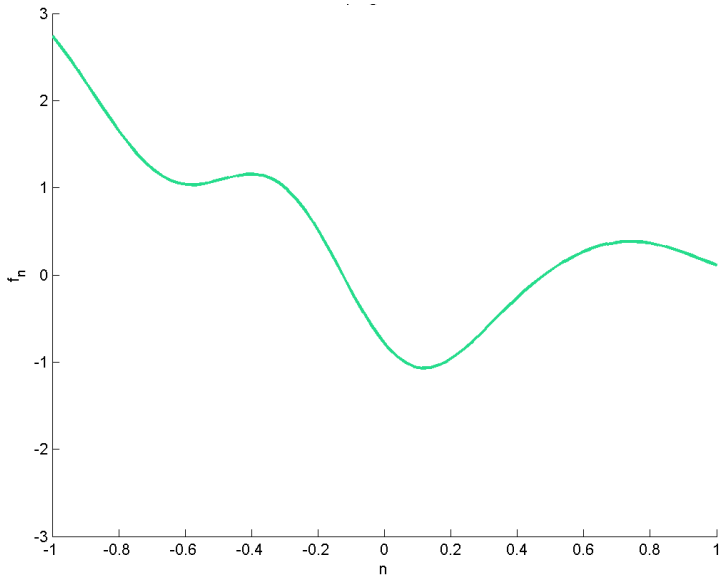
GP: Infinite dimensional Gaussian distribution



1-dim



2-dim



∞-dim

A GP is a distribution over functions.