BSc Informatics & Telecommunications

BSc Informatics & Telecommunications

MSc Informatics & AI

BSc Informatics & Telecommunications

MSc Informatics & AI

PhD Machine Learning

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- **Researcher ML & Robotics**

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- Researcher ML & Robotics
- **Start-up**

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- Researcher ML & Robotics
- Start-up

ATI Fellowship

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- Researcher ML & Robotics
- Start-up

  ATI Fellowship
- **Amazon**

What this talk is about

- Deep Bayesian learning over functions.

• Adding prior knowledge/assumptions	$p(f) \quad \text{vs} \quad p(w)$
• Generalization/generation by averaging hypotheses	$\int_f p(y\|f)p(f) \quad \text{vs} \quad p(y\|f(\hat{w}))$
• Unsupervised learning for generation	$p(\text{generated\_speech} \mid \text{latent\_style})$
• Intuitive composition of signals	$f_1 \circ f_2 \quad \text{vs} \quad f([w_1, w_2])$

What this talk is about

- Deep Bayesian learning over functions.

• Adding prior knowledge/assumptions	$p(f) \quad \text{vs} \quad p(w)$
• Generalization/generation by averaging hypotheses	$\int_f p(y\|f)p(f) \quad \text{vs} \quad p(y\|f(\hat{w}))$
• Unsupervised learning for generation	$p(\text{generated\_speech} \mid \text{latent\_style})$
• Intuitive composition of signals	$f_1 \circ f_2 \quad \text{vs} \quad f([w_1, w_2])$

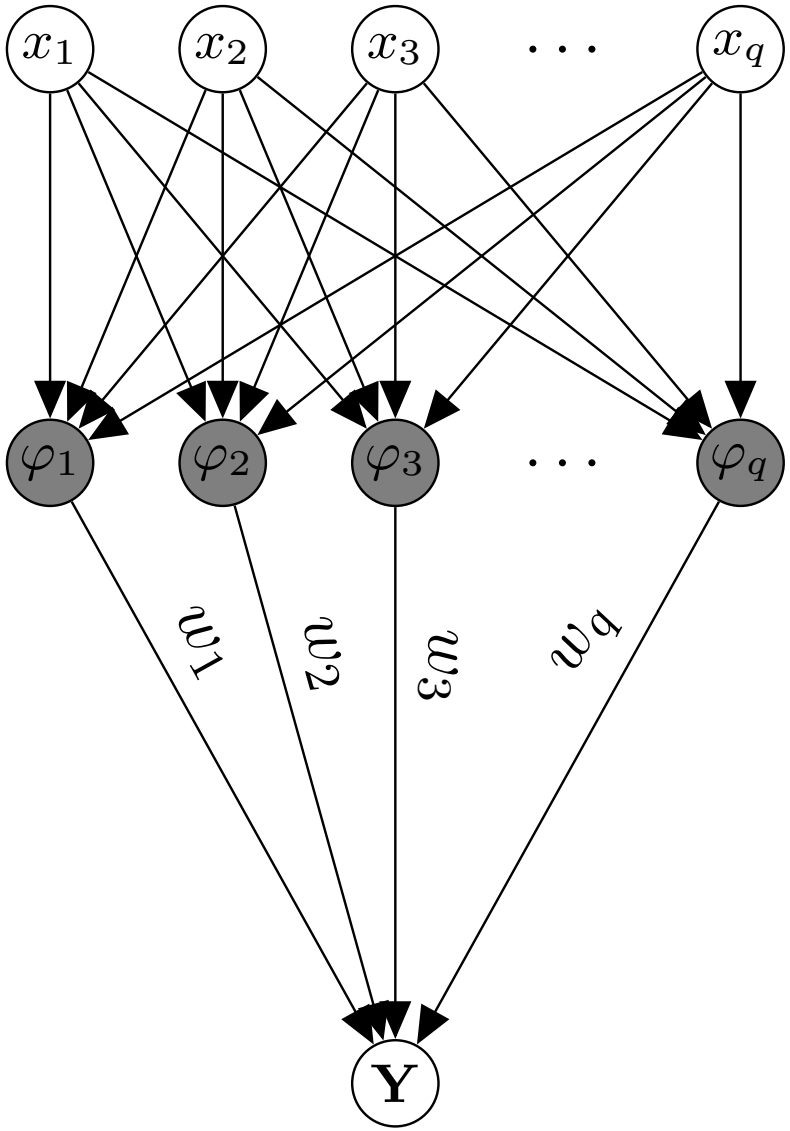
- Particularly important for:
  - Generative modeling
  - Domain adaptation

What this talk is ***not*** about

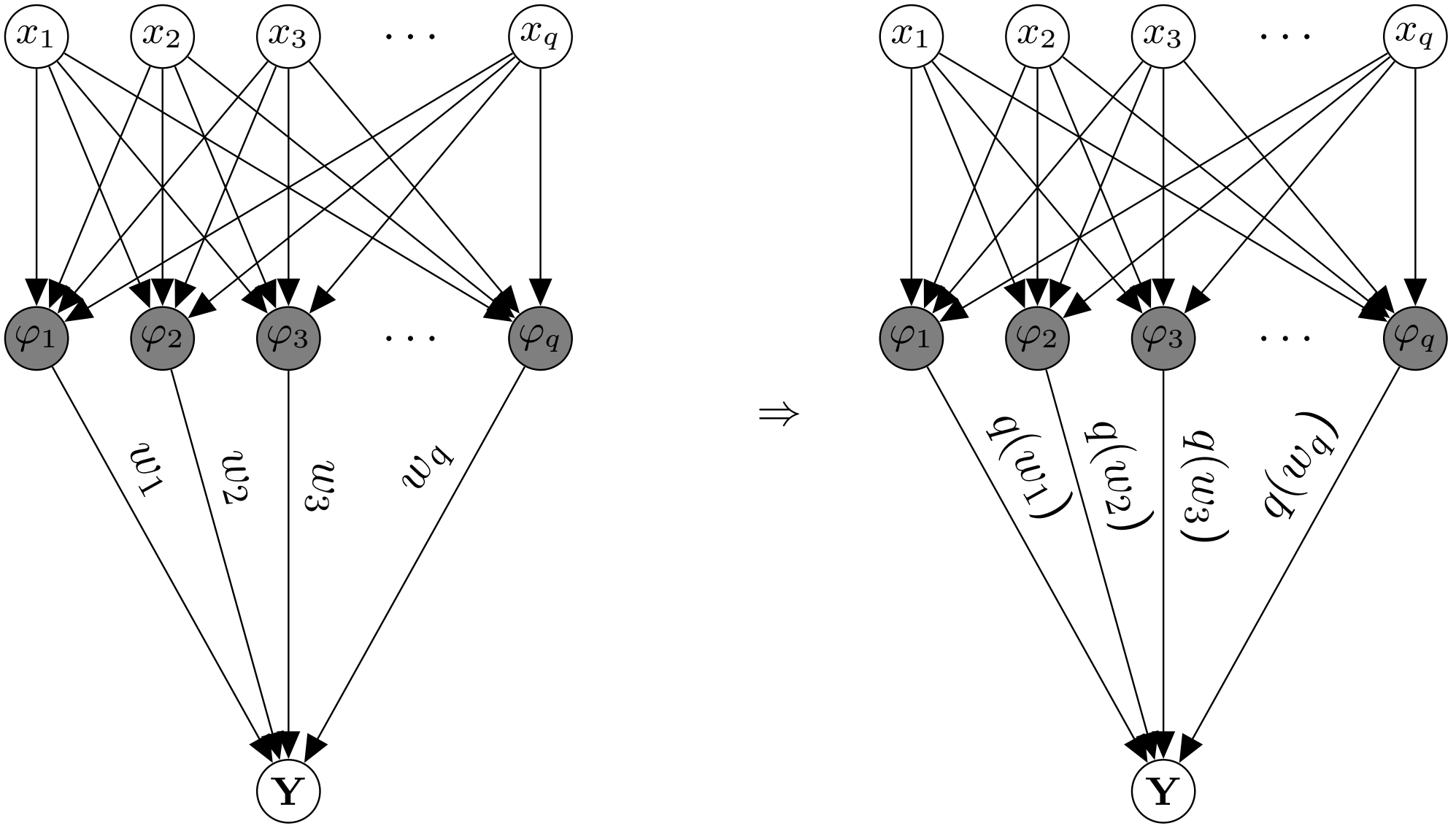
Reverend T. Bayes

- We don't dogmatically motivate Bayesian methods.
- In fact, Bayesian methods lack the practical *learnability* of DNNs.
- *Hence, we wish to combine function space modeling with the learnability of DNNs.*

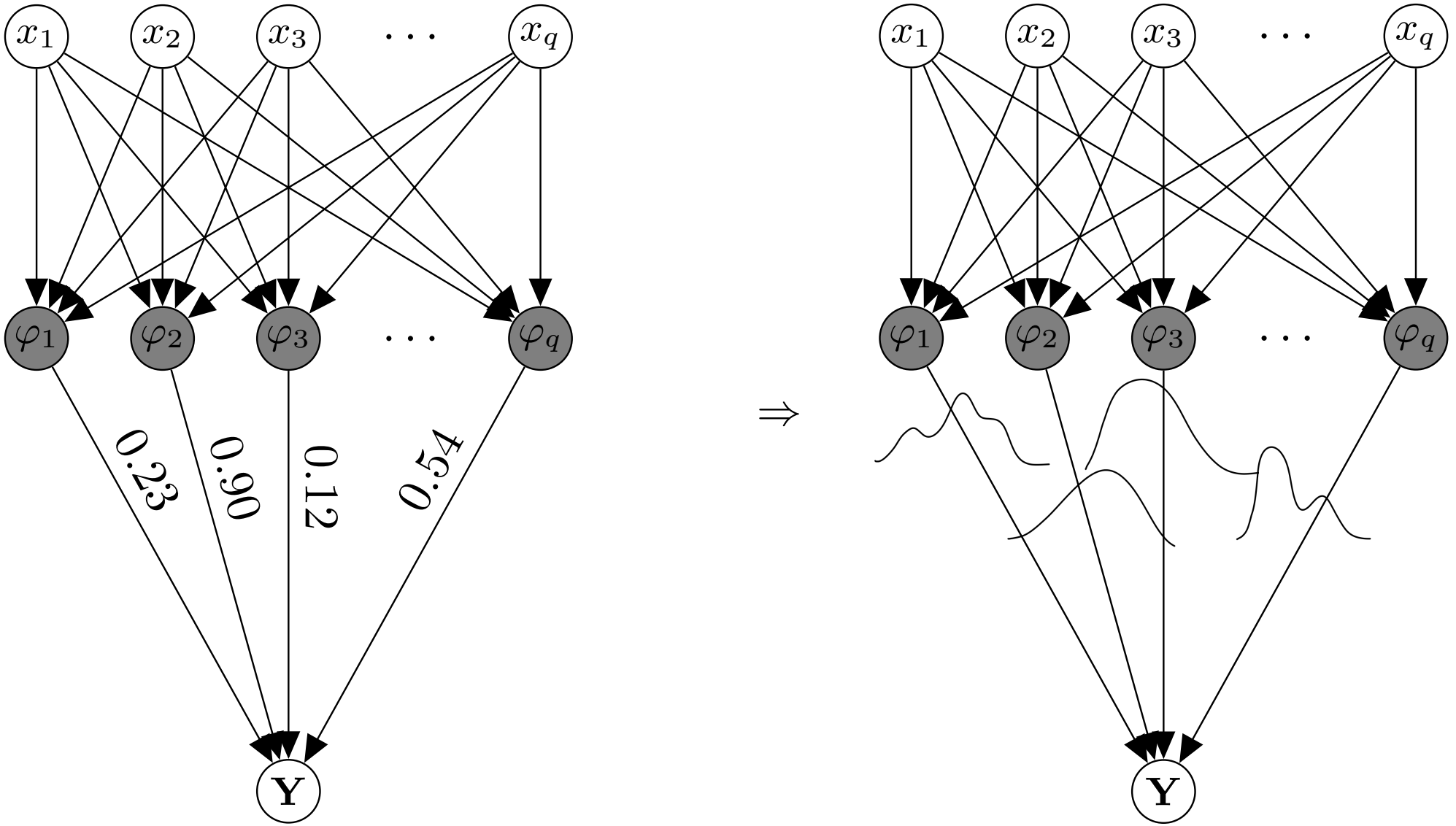
A standard neural network



BNN with priors on its weights



BNN with priors on its weights



Learning in function space

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\textit{parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*, \hat{\mathbf{w}}_\mathbf{D}) \quad = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D})$$

Learning in function space

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\textit{parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*, \hat{\mathbf{w}}_\mathbf{D}) = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D})$$

$$\textit{Bayesian parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*) = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D})$$

Learning in function space

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\begin{aligned}
\textit{parametric}: \quad & p(\mathbf{y}_*|\mathbf{x}_*, \hat{\mathbf{w}}_\mathbf{D}) && = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D}) \\
\textit{Bayesian parametric}: \quad & p(\mathbf{y}_*|\mathbf{x}_*) && = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D}) \\
\textit{Gaussian process} : \quad & p(\mathbf{y}_*|\mathbf{x}_*) && = \int_{\mathbf{f}_*} \; p(\mathbf{y}_*|f_*(\mathbf{x})) \quad p(f_*|\mathbf{D})
\end{aligned}$$

Learning in function space

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\begin{aligned}
\textit{parametric}: \quad & p(\mathbf{y}_*|\mathbf{x}_*, \hat{\mathbf{w}}_\mathbf{D}) && = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D}) \\
\textit{Bayesian parametric}: \quad & p(\mathbf{y}_*|\mathbf{x}_*) && = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D}) \\
\textit{Gaussian process} : \quad & p(\mathbf{y}_*|\mathbf{x}_*) && = \int_{\mathbf{f}_*} p(\mathbf{y}_*|f_*(\mathbf{x})) \; p(f_*|\mathbf{D})
\end{aligned}$$

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 $f_i = f(\mathbf{x}_i)$:

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

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!

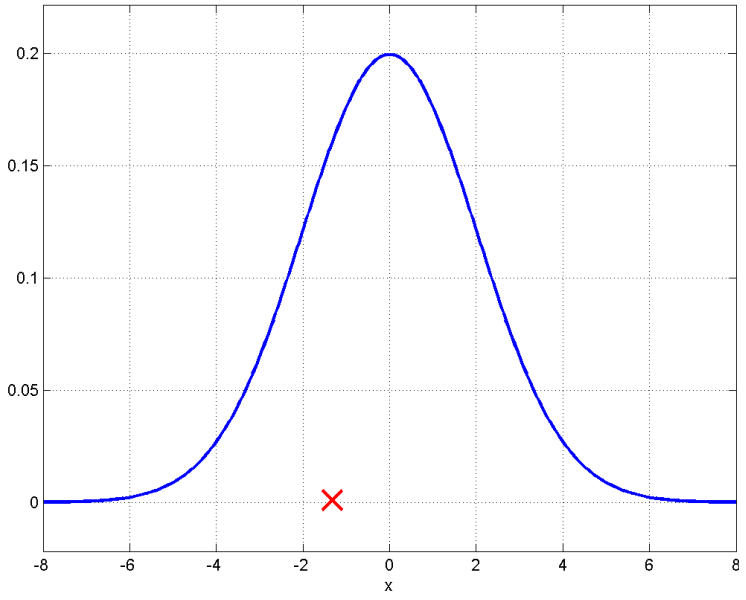
Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

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

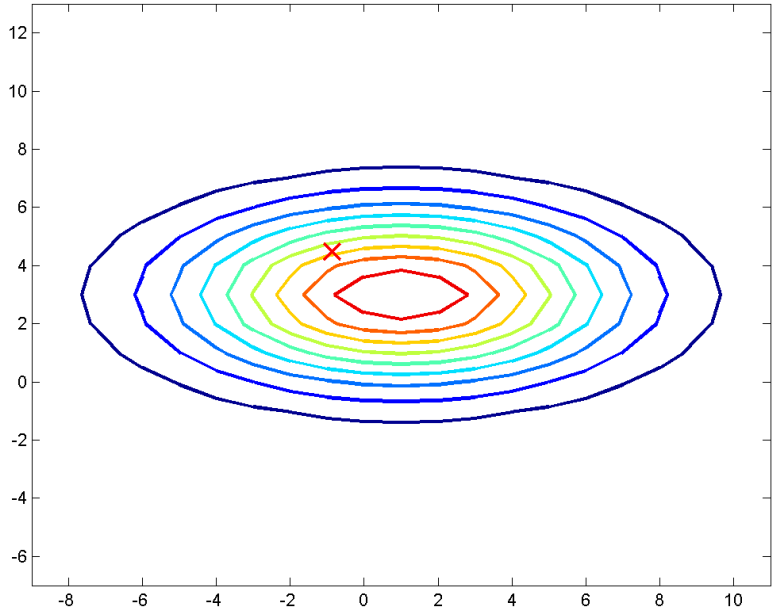
$$\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j; \theta)$$

Train a GP means fitting θ

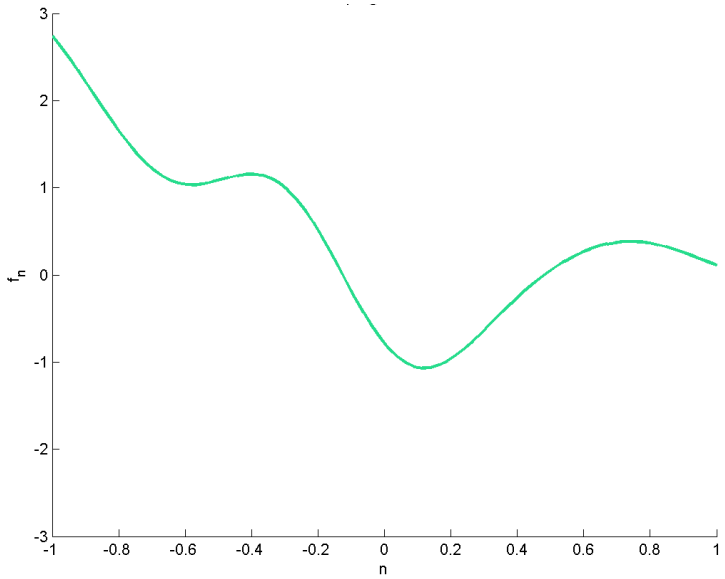
GP: Infinite dimensional Gaussian distribution



1-dim



2-dim



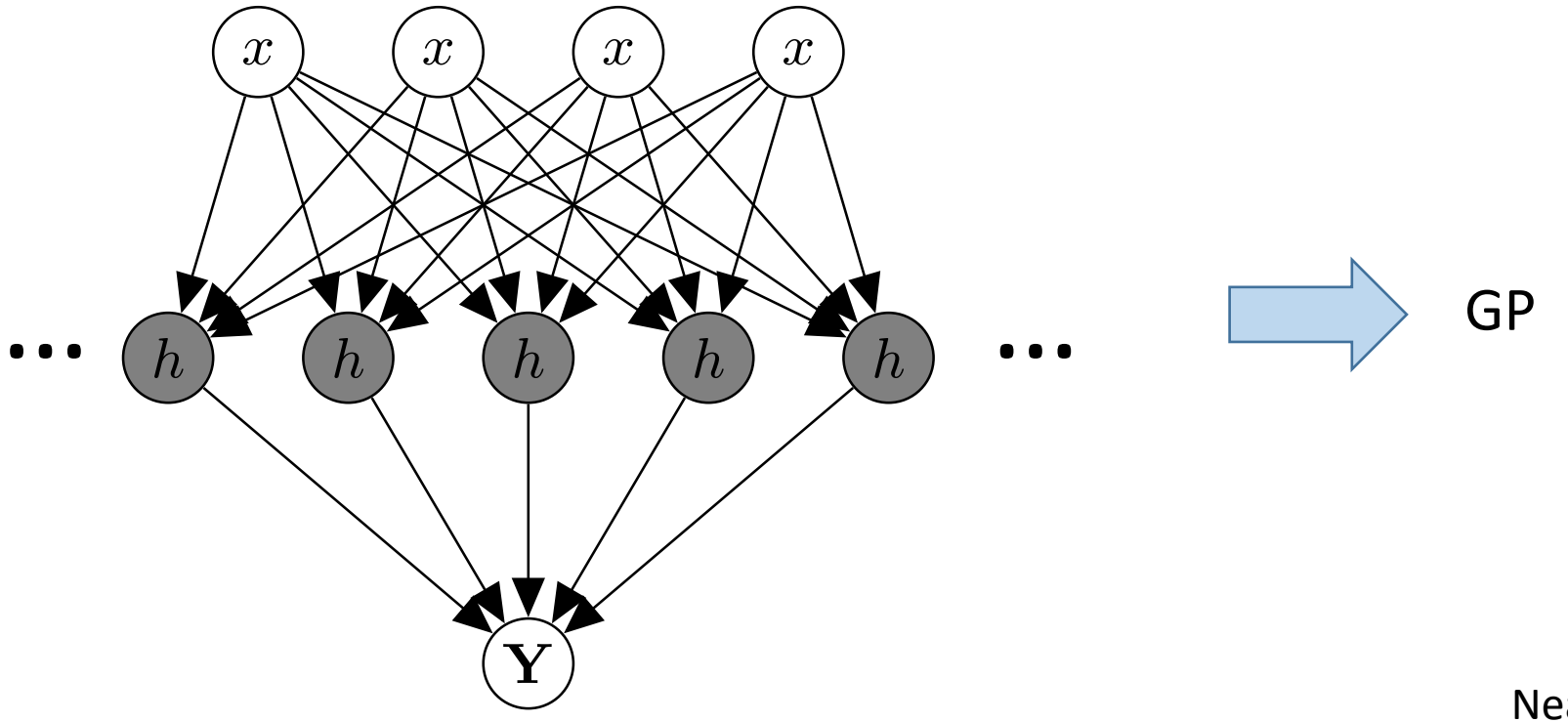
∞-dim

A GP is a distribution over functions.

GP as the limit of a Bayesian NN

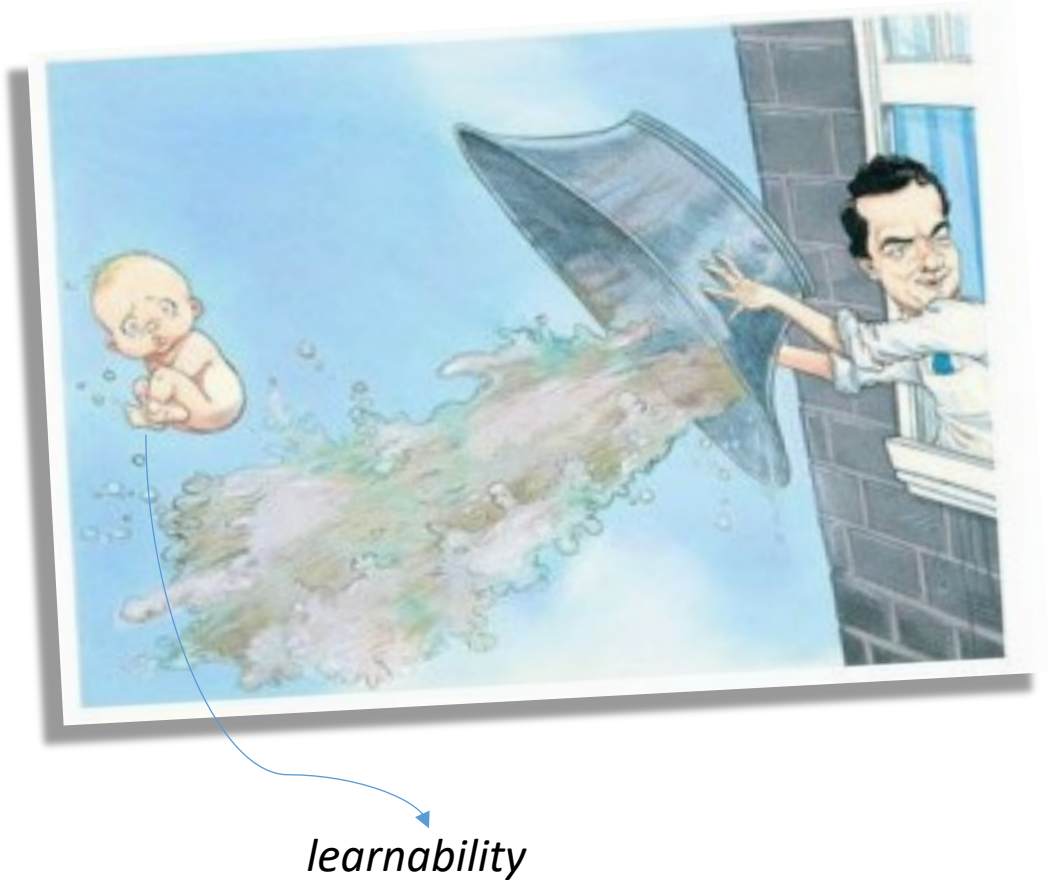
- Limit of infinite num. parameters:
  Optimization of finite num. parameters → Integration of infinite model

- *Closed form* solution for modelled function class!



Neal 1994; Williams 1997

Did we throw the baby out with the bathwater? (D. MacKay)



learnability

- In the presence of finite data the analytic solution might not be the best! Neural networks, instead, perform **hierarchical concept learning**.

For the rest of the talk…

Function space modeling
(benefits for tasks beyond supervised learning)

+

Hierarchical concept learning
(benefits for learnability)

Deep Gaussian process

Hierarchical feature learning *and* function space learning.

$$g(x) = f(f(f \cdots (x)))$$ where $$f \sim \mathcal{GP}$$

[Damianou, Lawrence. "Deep Gaussian processes", 2013]
[Damianou, "Deep Gaussian processes and variational propagation of uncertainty", PhD Thesis]
[Kumar, Singh, Srijith, Damianou, "Deep Gaussian processes with convolutional kernels", 2018]

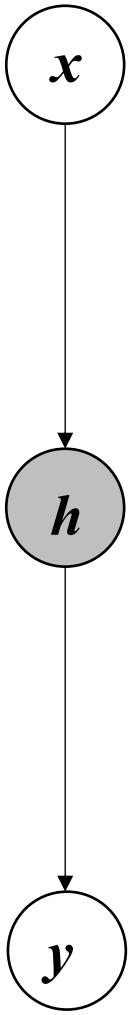
Deep Gaussian process

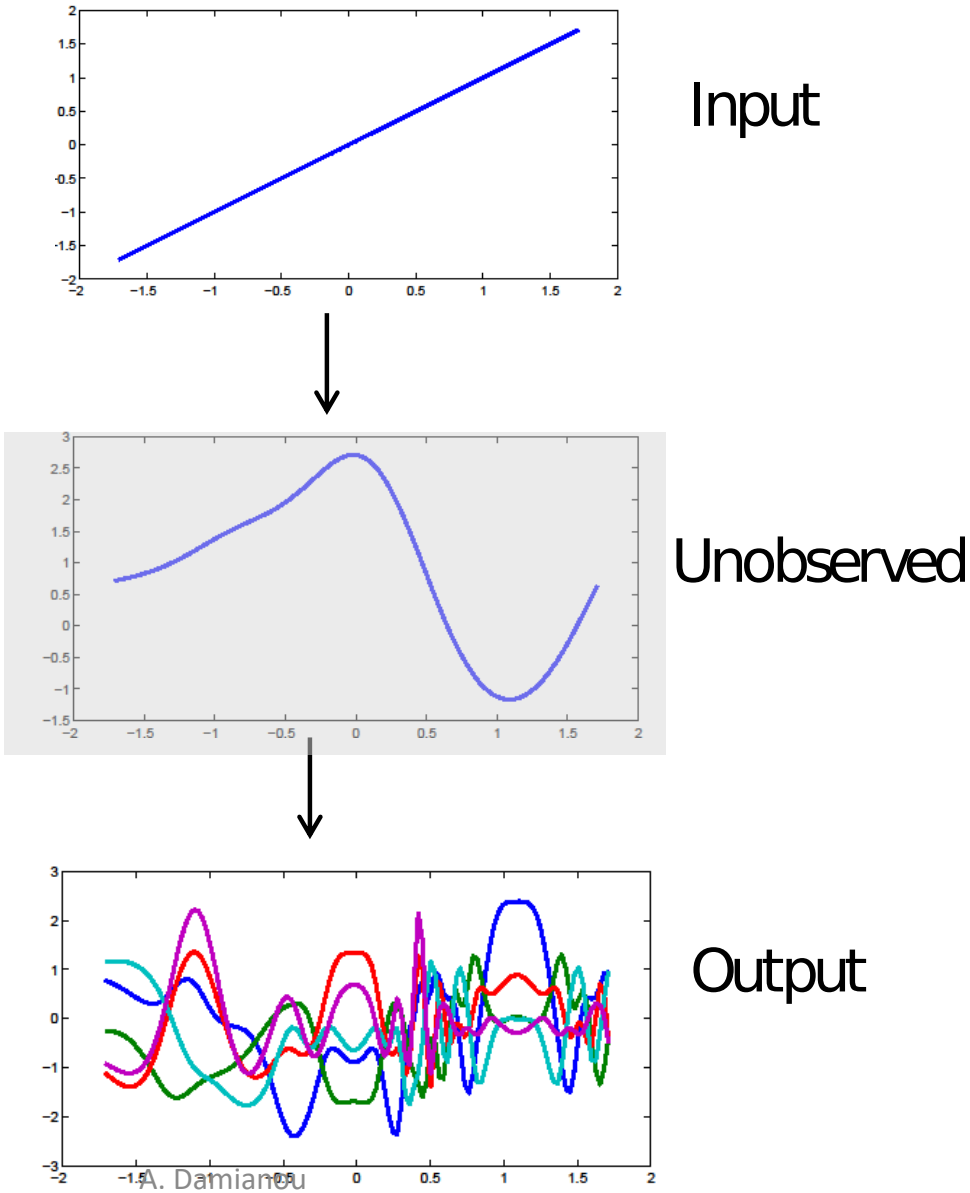
$$p(y|x) = \int_{f_3} p(y|f_3) \int_{f_2} p(f_3|f_2) \int_{f_1} p(f_2|f_1)p(f_1|x)$$

Damianou & Lawrence, 2013, Damianou, PhD Thesis 2015

Hierarchical function learning

[Damianou, PhD Thesis]

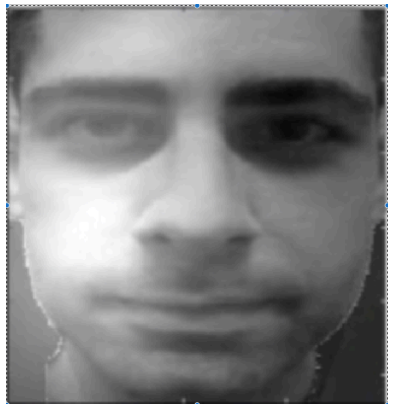




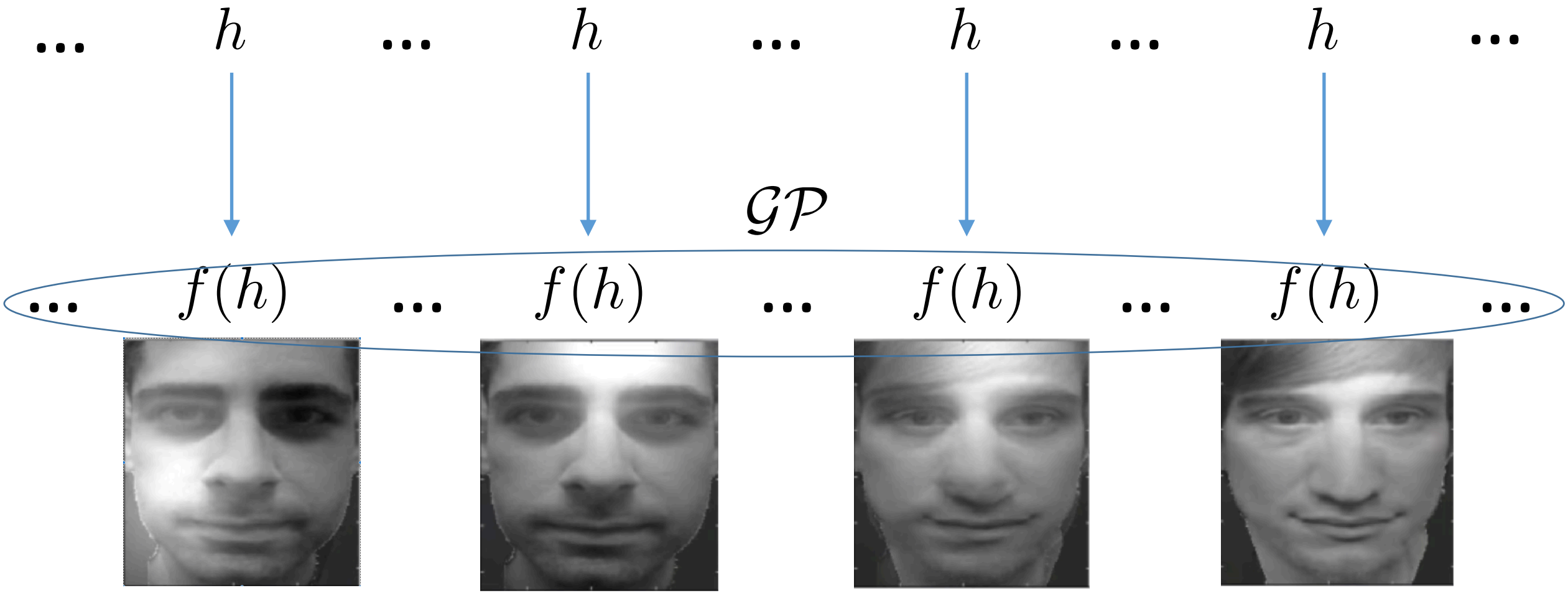
Function space ≈ the "*infinite data*" space

$\ldots \quad h \quad \ldots \quad h \quad \ldots \quad h \quad \ldots \quad h \quad \ldots$

$\ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots$



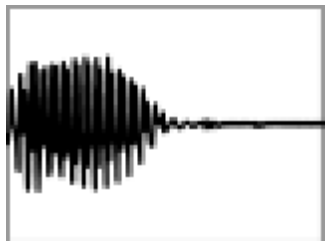
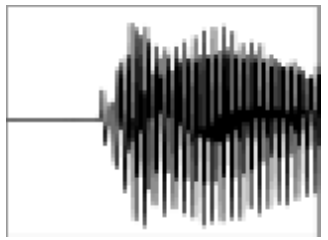

Function space ≈ the "*infinite data*" space



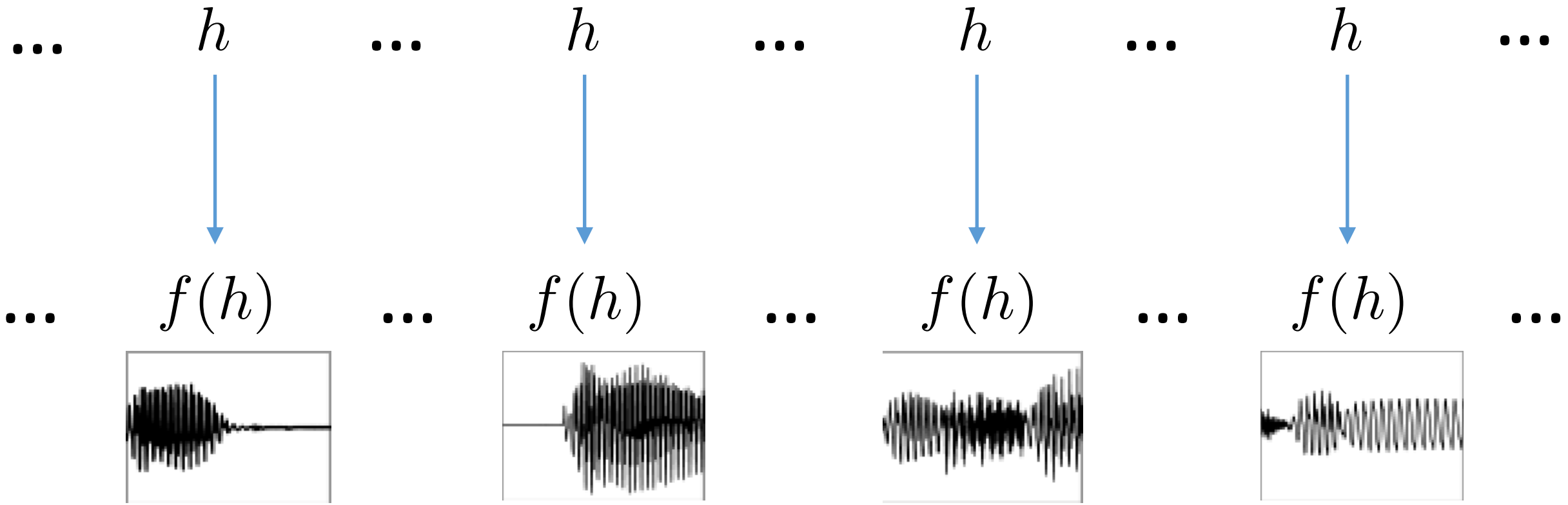
Function space ≈ the "*infinite data*" space

$\ldots \quad h \quad \ldots \quad h \quad \ldots \quad h \quad \ldots \quad h \quad \ldots$

$\ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots$

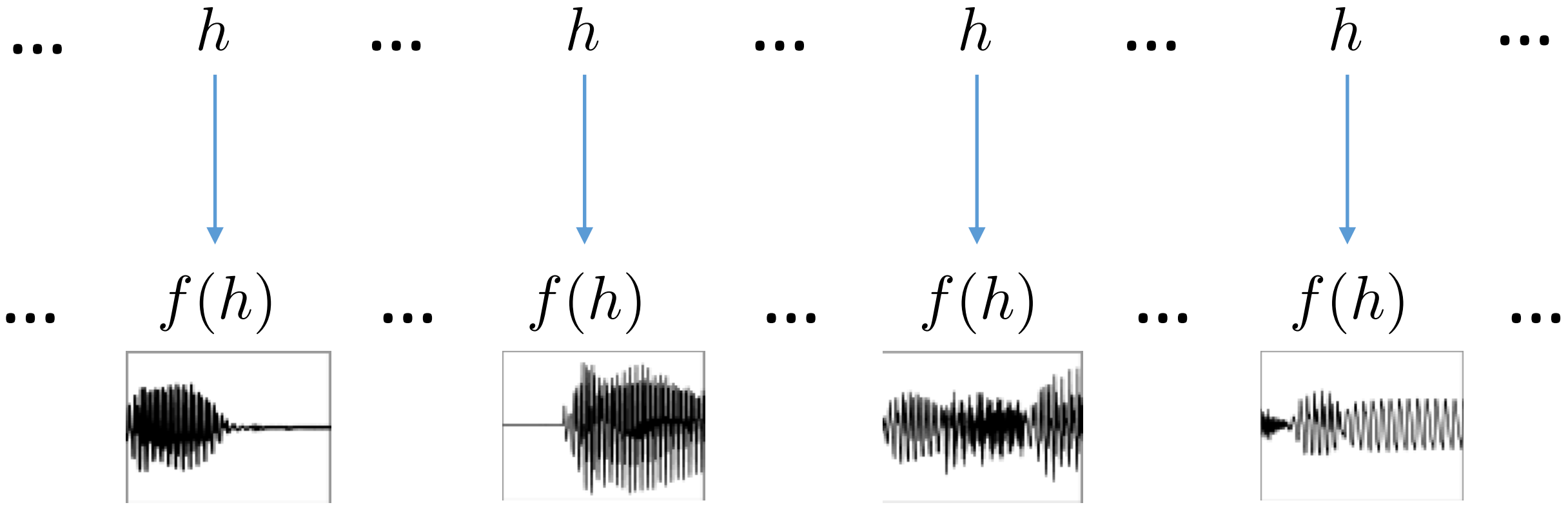
Function space ≈ the "*infinite data*" space



$$p(f_*|x_*, \mathbf{x}, \mathbf{f}) \sim \mathcal{N}(\mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}, k_*)$$

Predictions: linear combination of training utterances

Function space ≈ the "*infinite data*" space



$$p(f_*|x_*, \mathbf{x}, \mathbf{f}) \sim \mathcal{N}(\mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}, k_*)$$ Predictions: linear combination of training utterances

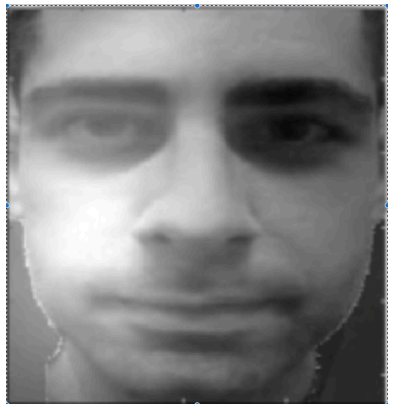
Hybrid between "unit selection" and "parametric" (functional) generation?

Function space ≈ the "*infinite data*" space

$$\ldots \quad h \quad \ldots \quad h \quad \ldots \quad h \quad \ldots \quad h \quad \ldots$$

$$\ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots$$

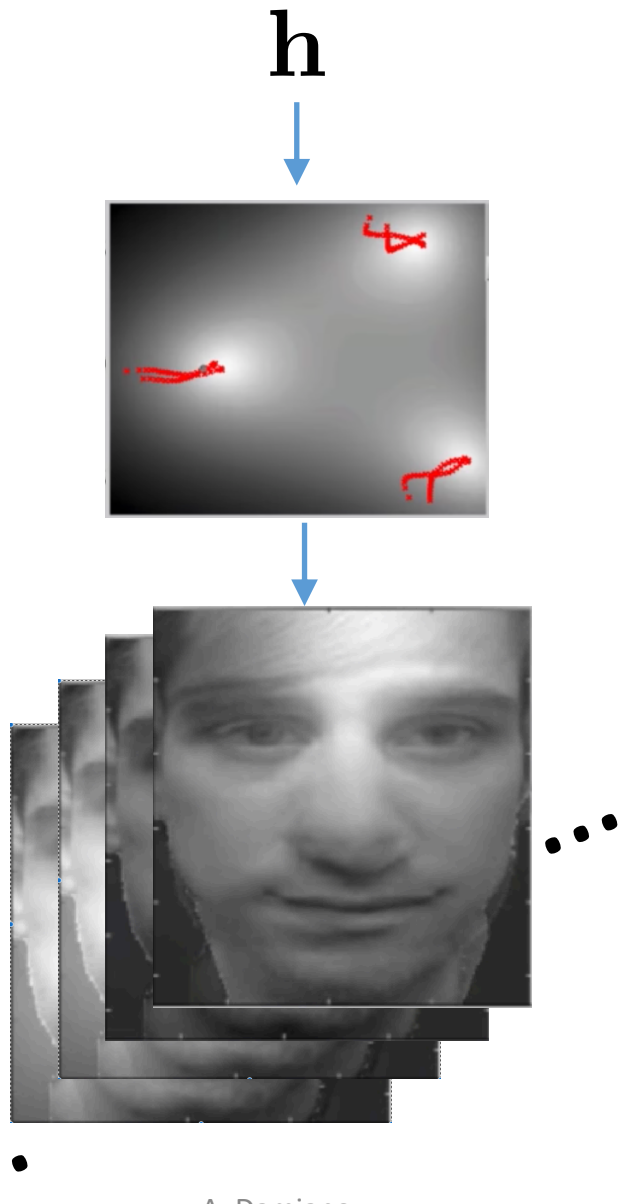
$$\ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots \quad f(h) \quad \ldots$$




Function space ≈ the "*infinite data*" space

Intermediate layer
function space
(2-dim multivariate function)

Observation layer
(#pixels-dim multivariate function)



Generative modelling

- Training set: *150* images
- Raw pixels, no pre-processing
- No convolutions
- No explicit information about variables such as *“person_id”* or *“light_angle”*

https://youtu.be/rIPX3CIOhKY?t=91

Audio generation

https://youtu.be/Q3oBt2WhaXc

- Training set: ~ *200* short pieces of classical music
- No pre-processing

Function generative modeling advantages

- Controls are learned “automatically” in function space, where is easier to (de)compose because we work directly with the *function* rather than with *the parameter that induces the function*.

- Can learn from very small data by averaging hypotheses which is a useful property even when we have massive data, because it shows *we can cover the space of variability well*.

State-of-the-art for deep GPs

- Focus on extensions:
  - Recurrent DGPs *[Mattos, Dai, Damianou, Barreto, Lawrence, 2017]*
  - Convolutional DGPs *[Kumar et al. 2018; Blomqvist et al. 2018; Dutordoir et al. 2019]*
  - IRL-DGP *[Jin, Damianou, Abbeel, Spanos, 2015]*
  - Active learning/BO *[Yang et al. 2020; Fei et al. 2019]*
  - Physical sciences & Engineering (molecule dynamics, aerospace vehicle design, geospatial modeling)
  - ***Hybrids with DNNs*** *[Dai, Damianou, Gonzalez, Lawrence ICLR 2016; Tran et al. 2019]*

- Focus on approximations:
  - EP, MCMC etc. VI dominates *[Damianou NeurIPS workshop keynote 2017; Yu et al. 2019; Salimbeni et al. 2019]*

- Opinion: Time to re-visit generative modelling with functional models

Gaussian processes with trained neural network inductive biases

(best of both worlds?)

- Train neural networks in situations where they are superior.
- Encode them in function space through a GP.
- Manipulate the resulting function e.g. for domain adaptation in function space.
  - *Transfer inductive biases from the **trained** DNN, not its architectural structure!*

GP with NN inductive biases

[Maddox, Tang, Moreno, Wilson, Damianou (2020)]

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP f with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
- Adaptation: $p(f_*|x_*, \mathbf{x}, \mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_*, k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*})$

  where e.g. $\mathbf{k}_{*\mathbf{x}} = k(x_*, \mathbf{x}) = J(x_*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$.

GP with NN inductive biases

[Maddox, Tang, Moreno, Wilson, Damianou (2020)]

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP f with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
- Adaptation: $p(f_*|x_*, \mathbf{x}, \mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_*, k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*})$

  where e.g. $\mathbf{k}_{*\mathbf{x}} = k(x_*, \mathbf{x}) = J(x_*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$.

GP with NN inductive biases

[Maddox, Tang, Moreno, Wilson, Damianou (2020)]

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP f with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
- Adaptation: $p(f_*|x_*, \mathbf{x}, \mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_*, k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*})$

  where e.g. $\mathbf{k}_{*\mathbf{x}} = k(x_*, \mathbf{x}) = J(x_*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$.

GP with NN inductive biases

[Maddox, Tang, Moreno, Wilson, Damianou (2020)]

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP f with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
- Adaptation: $p(f_*|x_*, \mathbf{x}, \mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_*, k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*})$

  where e.g. $\mathbf{k}_{*\mathbf{x}} = k(x_*, \mathbf{x}) = J(x_*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$.

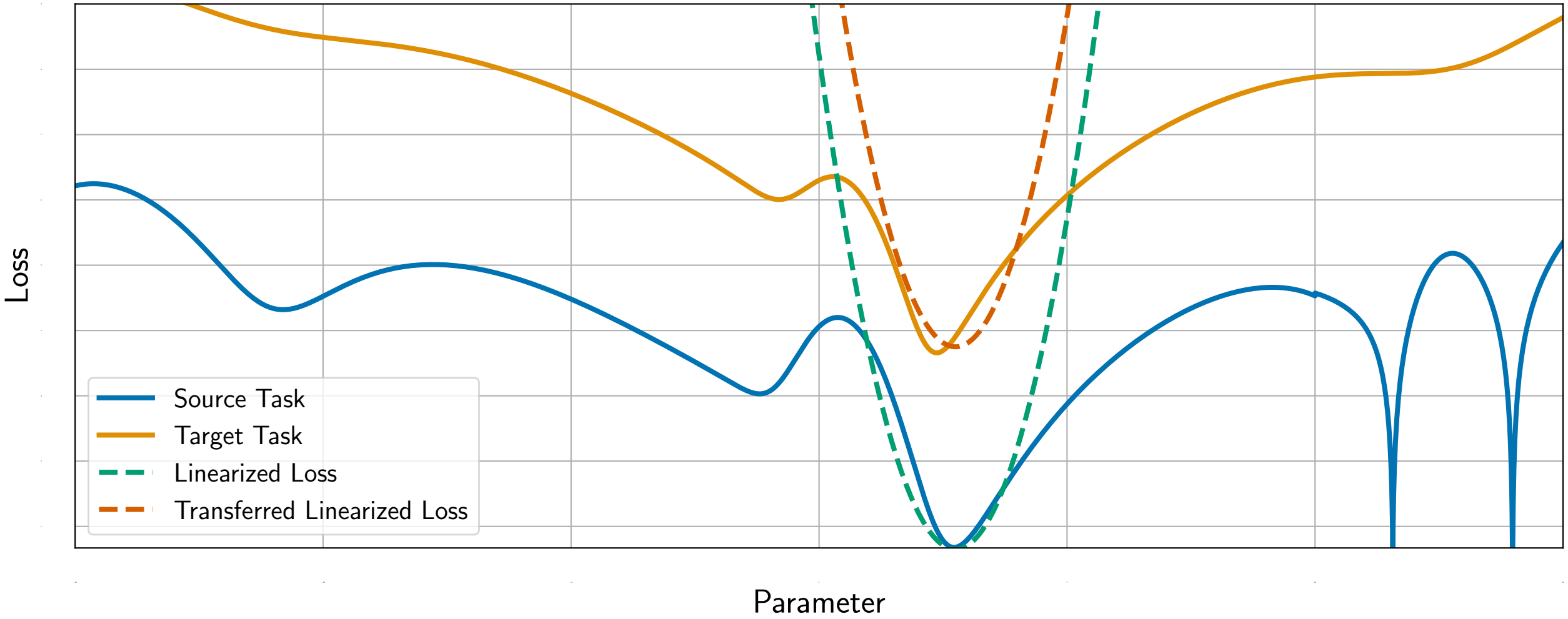
Fast, closed-form adaptation in the functional space with uncertainty!

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:

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


Loss
Parameter
Source Task
Target Task
Linearized Loss
Transferred Linearized Loss

Efficient computations

$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

  $g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has maximum at $\hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Efficient computations

$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f} \text{ has maximum at } \hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$$

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Efficient computations

$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has maximum at $\hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$ *[Pleiss et al. 2018]*

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Efficient computations

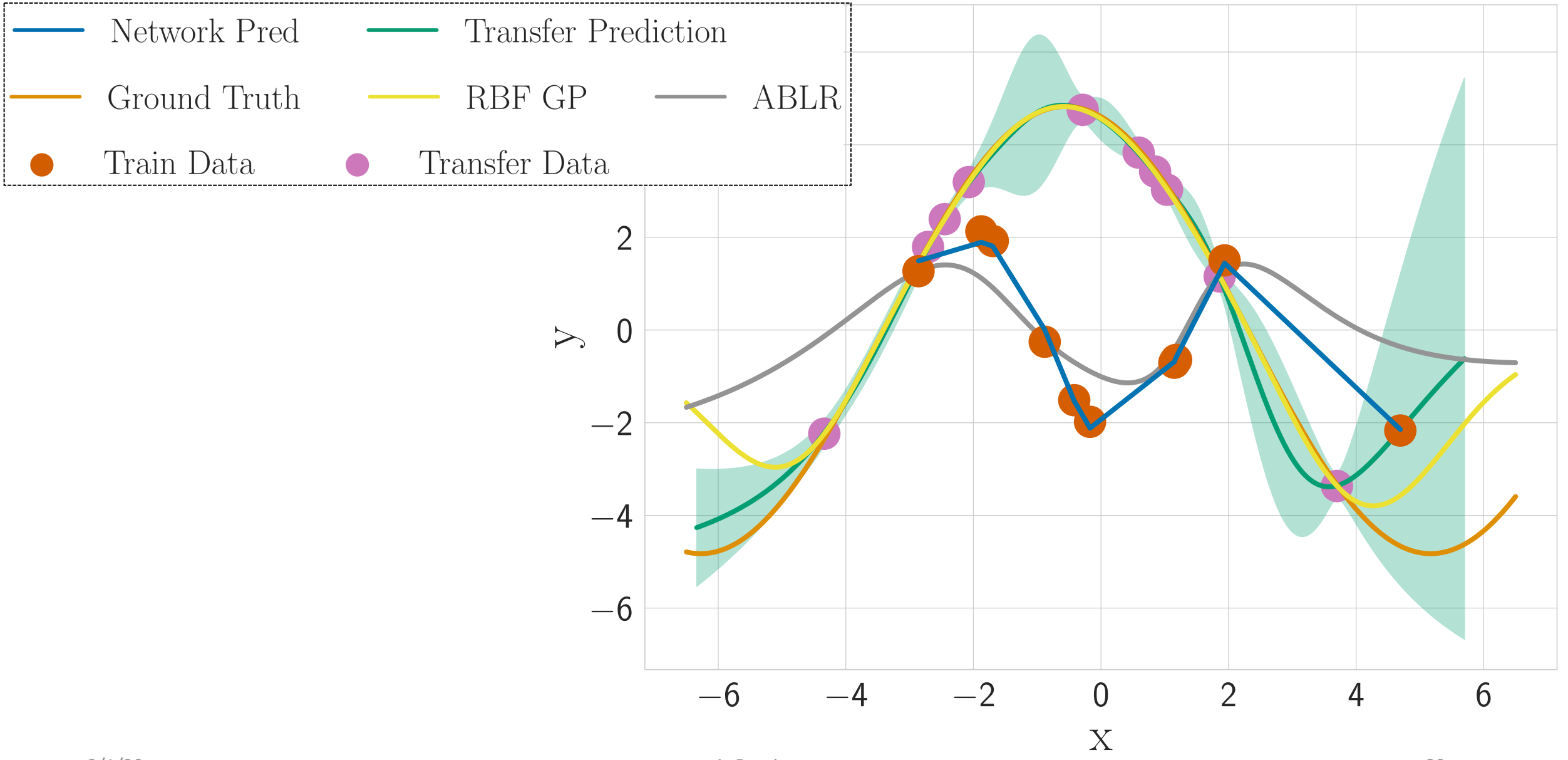
$$\boldsymbol{\mu}_* = \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*\mathbf{x}}\mathbf{K}_{\mathbf{xx}}^{-1}\mathbf{k}_{\mathbf{x}*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

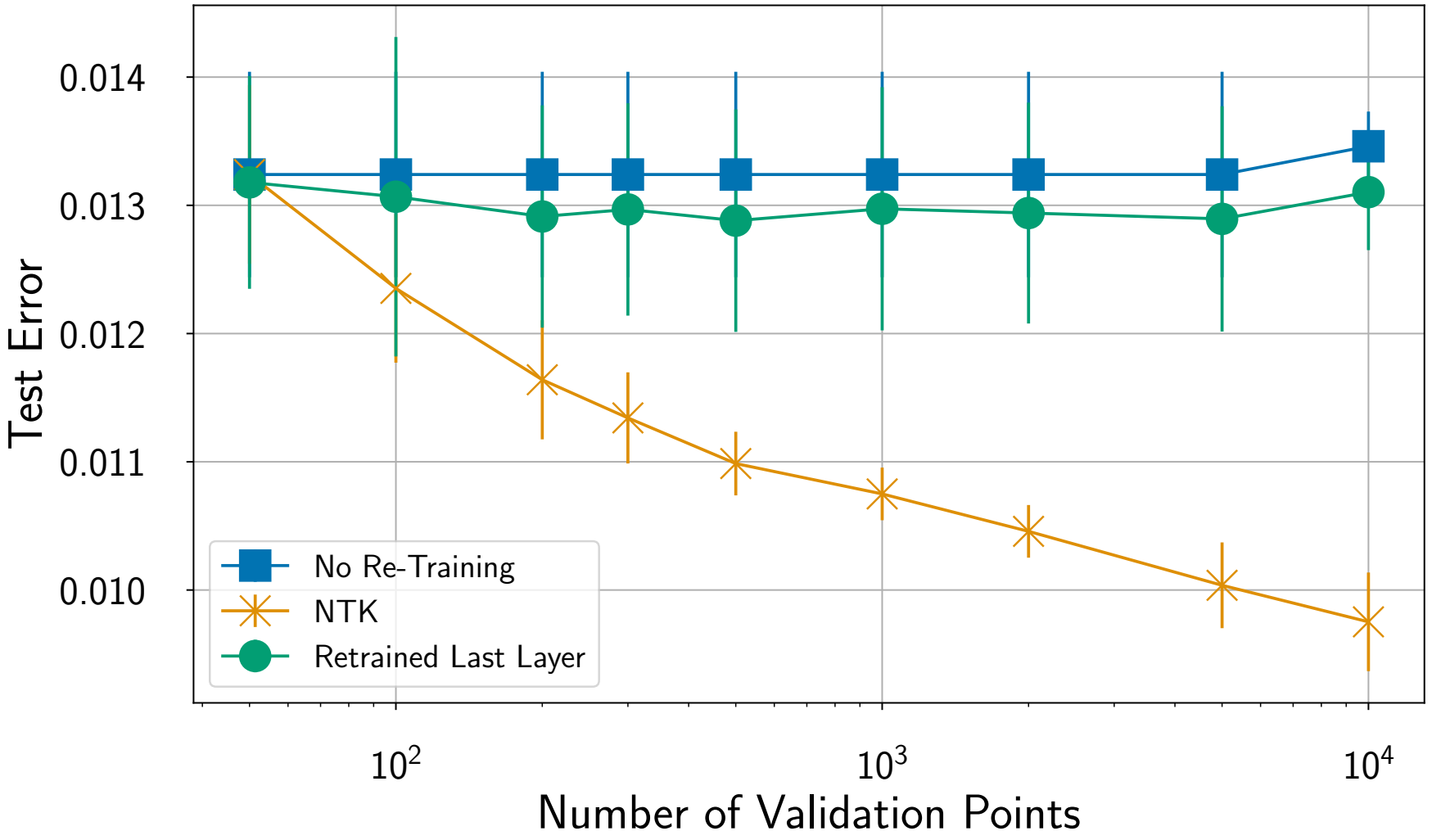
$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has maximum at $\hat{\mathbf{a}} = \mathbf{K}^{-1}\mathbf{f}$ *[Pleiss et al. 2018]*

- For t iterations the covariance expressed in terms of matrices with t columns.
- Allows to access $\mathbf{K}^{-1}\mathbf{f}$ lazily using fast MVP.

Transfer learning (toy regression data)



Malaria spread data



Conclusions

- DNNs are great predictors (supervised learning), thanks to learning algorithms.
- Beyond supervised learning: modelling in function space offers an alternative.
- Leverage hierarchical concept learning together with function space modelling:
  - NNs as inductive biases for GPs
  - Deep Gaussian process (stack of GPs)
- Demonstrated success in transfer learning.
- Re-visiting hybrid functional/DNN models for generation might be timely.

Questions?

See also:

- *adamian.github.io/talks/Damianou_deep_learning_rss_2018.pdf*
- *adamian.github.io/talks/Damianou_GP_tutorial.html*

Appendix

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

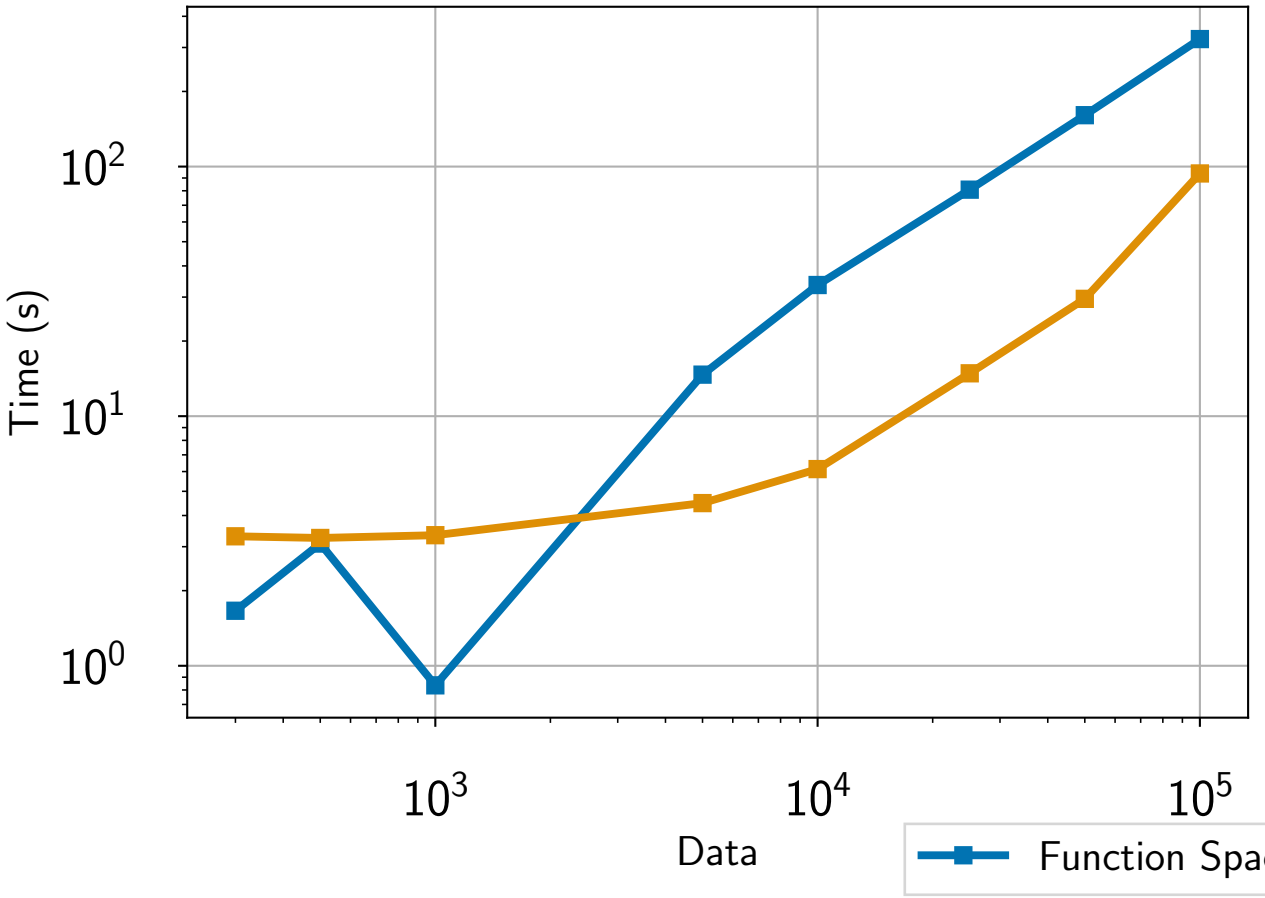
Model: A degenerate GP from DNNs obtained at convergence.

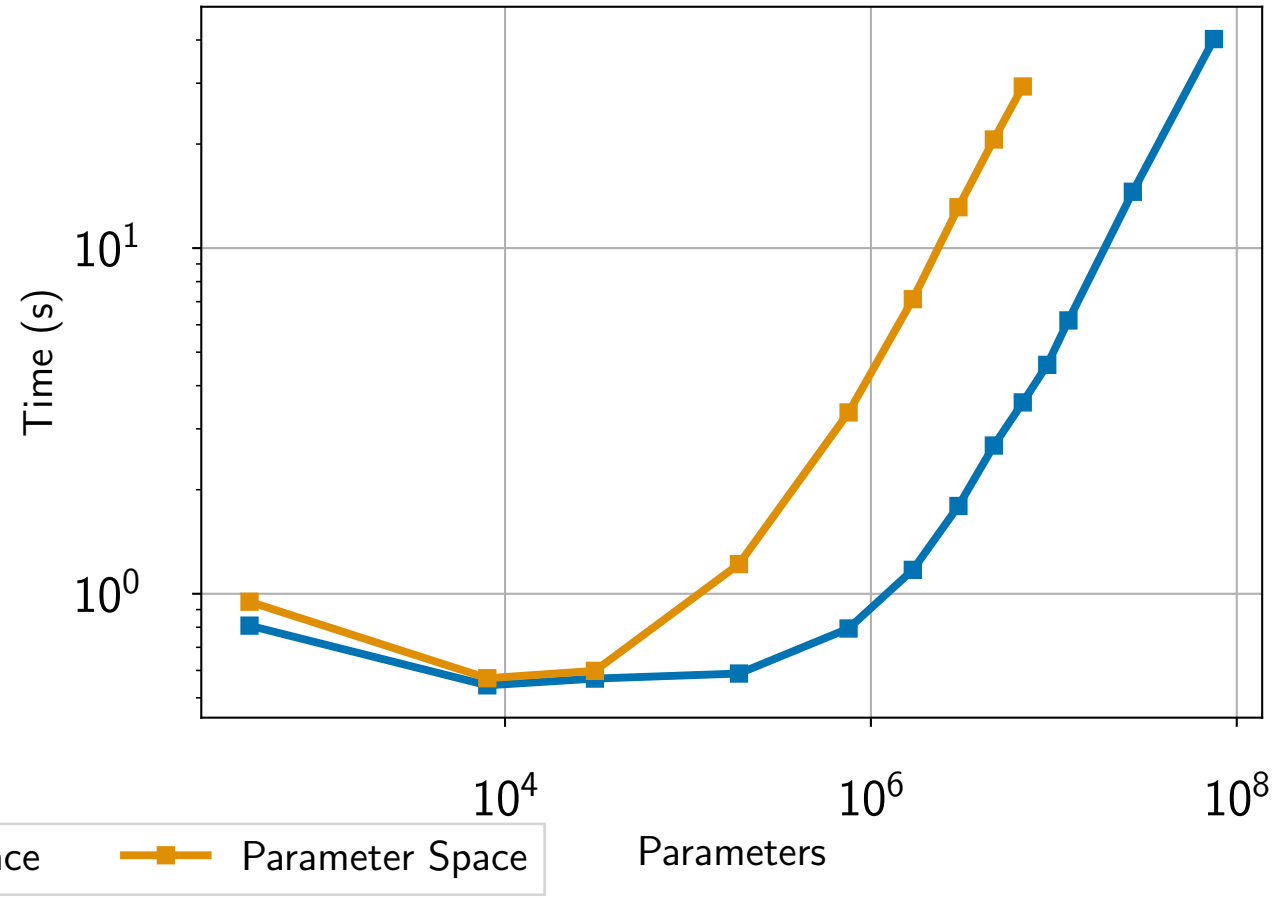
W. Maddox, S. Tang, P. Moreno, A. Wilson, A. Damianou: *Fast Adaptation with Linearized Neural Networks*. 2019

Other relevant works

- Perrone et al. 2018: Explicitly uses output features of DNN (we use NTK with Jacobian information from all layers)
- Wilson et al. 2015: DKL (whole NN embedded in kernel)
- Jaakkola et al. 1998 and others: gradients as features

Scaling

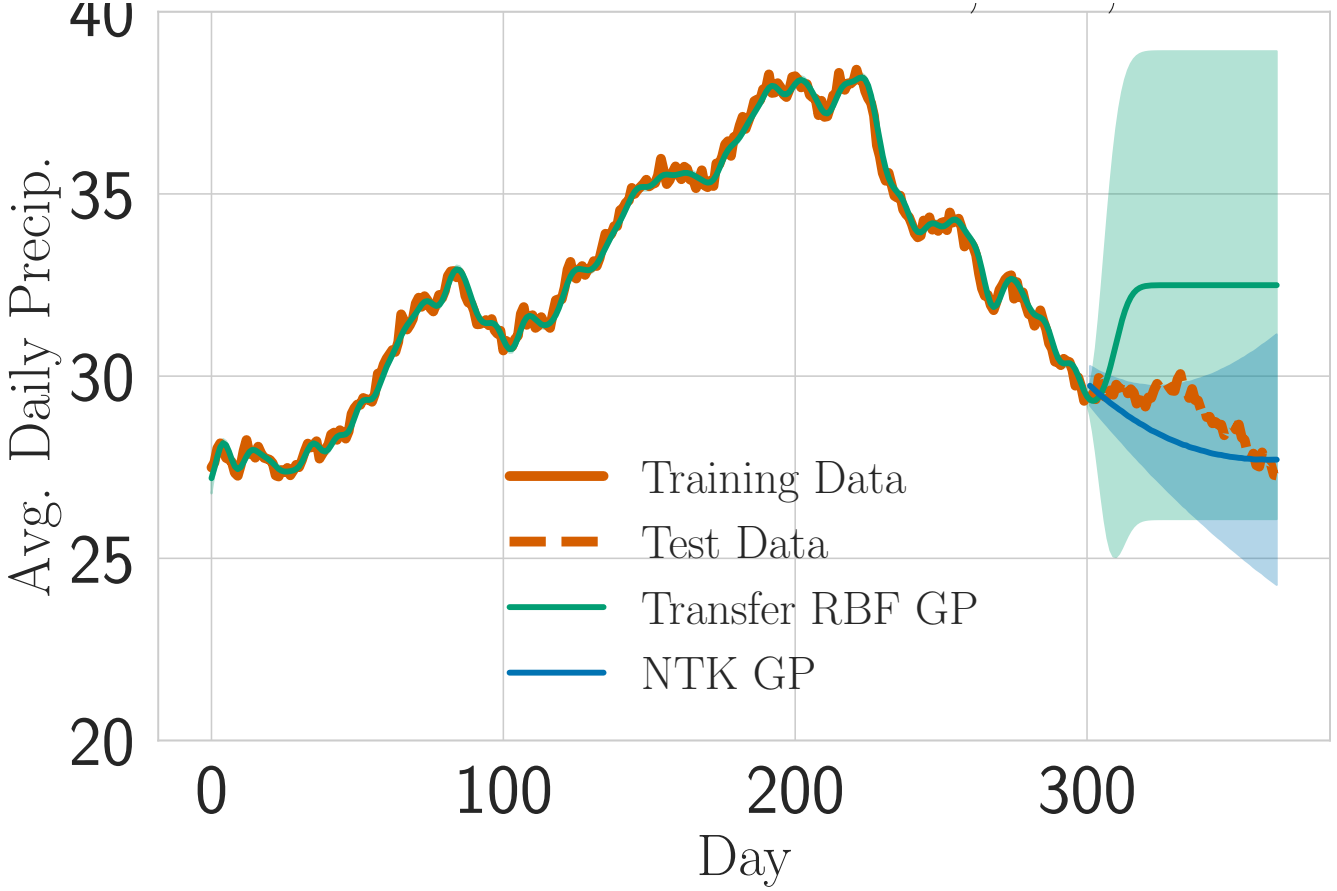

10^2
10^1
10^0
Time (s)
10^3
10^4
10^5
Data


10^1
10^0
Time (s)
10^4
10^6
10^8
Parameters

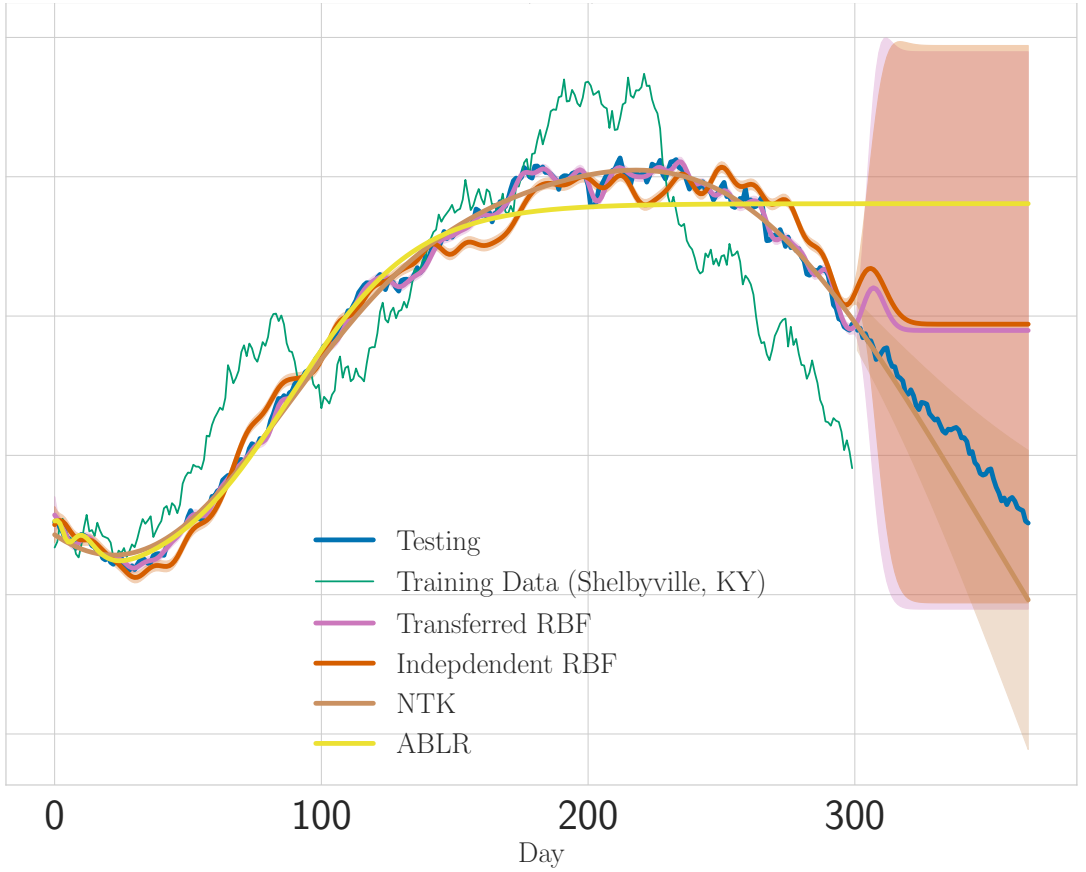
Function Space
Parameter Space

Transfer learning for precipitation data

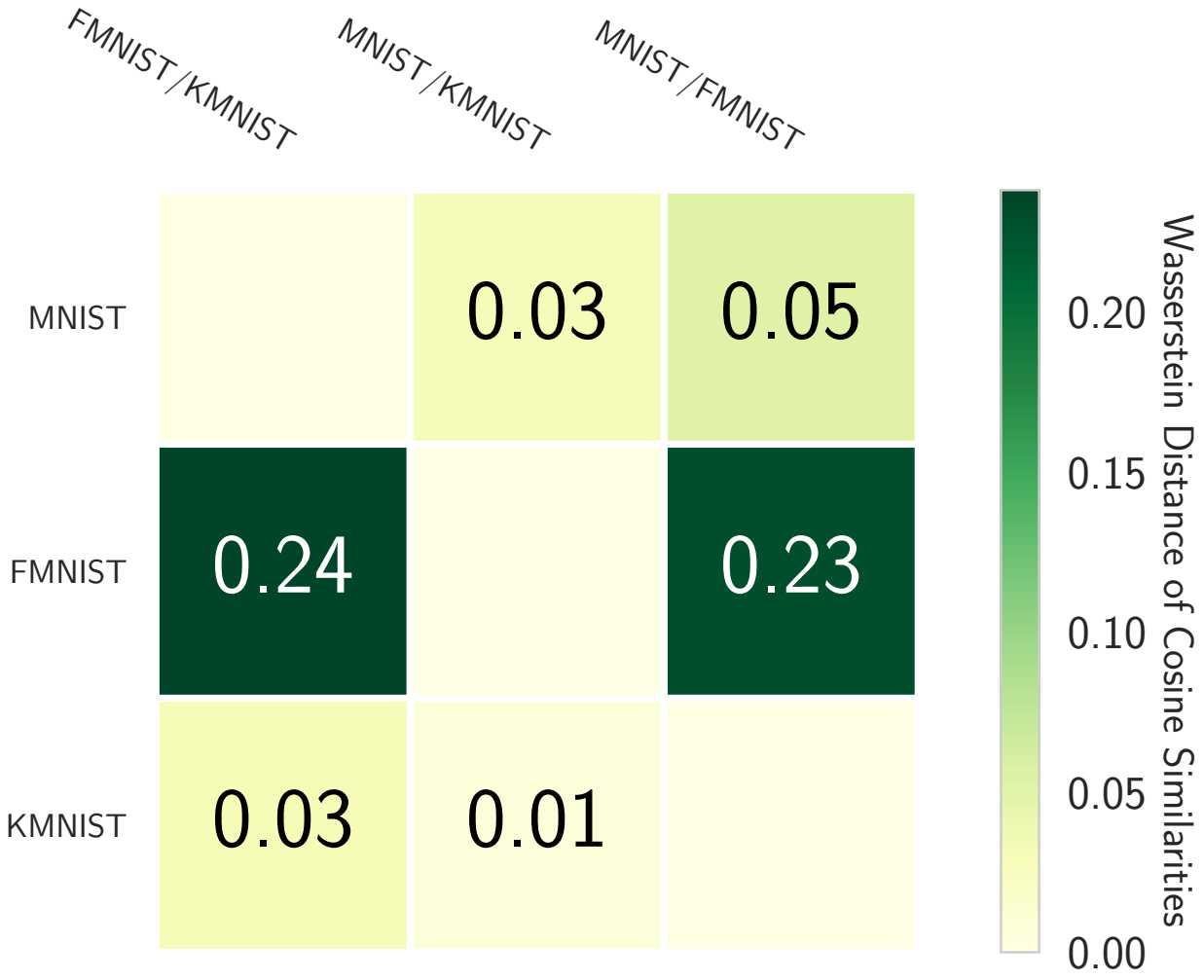
Source task



Target task



Similar tasks have similar Jacobians



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

Finite-NTK complexity

Computational complexities

- exact-GP : O(n^2)
- KISS-GP+Lanczos : O(t).
- exact-GP+Lanczos : O(tn) but k is very small
- sparseGP : O(m^2) but m is typically large
- sparseGP+Lanczos : O(km)

Storage costs

- exact-GP : O(n^2)
- KISS-GP+Lanczos : O(tm)
- exact-GP+Lanczos : O(tn) [but parallelizable]
- sparseGP : O(m^2)
- sparseGP+Lanczos : O(tm)

Complexities

- In training, $\mathbf{K}_{XX}$ is computed by $k(\mathbf{X}, \mathbf{X}; \theta)$. During optimization we change θ and thus have to compute the inverse in each optimization step. Cholesky is $O(n^3)$ and with ind. pts it can be done $O(nm^2)$. With KISS-GP it can be $O(n + m \log m)$ but requires grid.
- In predictions θ and hence $\mathbf{K}_{XX}$ is fixed. If we precompute whatever doesn't include $*$, then we have a standard vector-vector mult. for the mean, i.e. (O(n)). No linear solves required, so it can be done in a single GPU. For variance, if we precompute again, the remainder still costs $O(n^2)$ or $O(m^2)$ for Sparse GP or $O(k(n + mlogm))$ for KISS-GP.
- WIth KISS-GP + Lanczos (KISS-GP LOVE), instead of $\mathbf{k}_{*X}const$ we have $\mathbf{w}_{\mathbf{x}_*}$ which has only 4 nonzero elements, hence mean pred. is O(1). Pred. variance can then be O(k) with k very small, so they consider it O(1).
- With just Lanczos (and no KISS-GP), the variance goes back to O(nk) or O(mk) if used with sparse GPs. In the exact gp paper they say O(n), possibly because k is typically small. The mean remains O(n), I think, but can be parallelized.

Fast predictive variances

$$\tilde{\mathbf{k}}_{X\mathbf{x}_i^*} = W_X^\top K_{UU} \mathbf{w}_{\mathbf{x}_i^*}, \qquad \tilde{K}_{XX} = W_X^\top K_{UU} W_X$$

Approximate covariance using points in grid with weight W

$$k_{f|\mathcal{D}}(\mathbf{x}_i^*, \mathbf{x}_j^*) \approx k_{\mathbf{x}_i^*\mathbf{x}_j^*} - \tilde{\mathbf{k}}_{X\mathbf{x}_i^*}^\top (\tilde{K}_{XX} + \sigma^2 I)^{-1} \tilde{\mathbf{k}}_{X\mathbf{x}_j^*}. \quad (7)$$

By fully expanding the second term in (7), we obtain

$$\mathbf{w}_{\mathbf{x}_i^*}^\top \underbrace{K_{UU} W_X (\tilde{K}_{XX} + \sigma^2 I)^{-1} W_X^\top K_{UU}}_{C} \mathbf{w}_{\mathbf{x}_j^*} \quad (8)$$

Replace in GP predictive equation. Precomputing C with Lanczos.

$$C = K_{UU} W_X \underbrace{(\tilde{K}_{XX} + \sigma^2 I)^{-1}}_{\text{Apply Lanczos}} W_X^\top K_{UU}$$

$$\approx K_{UU} W_X (Q_k T_k^{-1} Q_k^\top) W_X^\top K_{UU}$$

$$= \underbrace{K_{UU} W_X Q_k}_{R^\top} \underbrace{T_k^{-1} Q_k^\top W_X^\top K_{UU}}_{R'}$$

C precomputed with t iterations, giving us structure R^T R

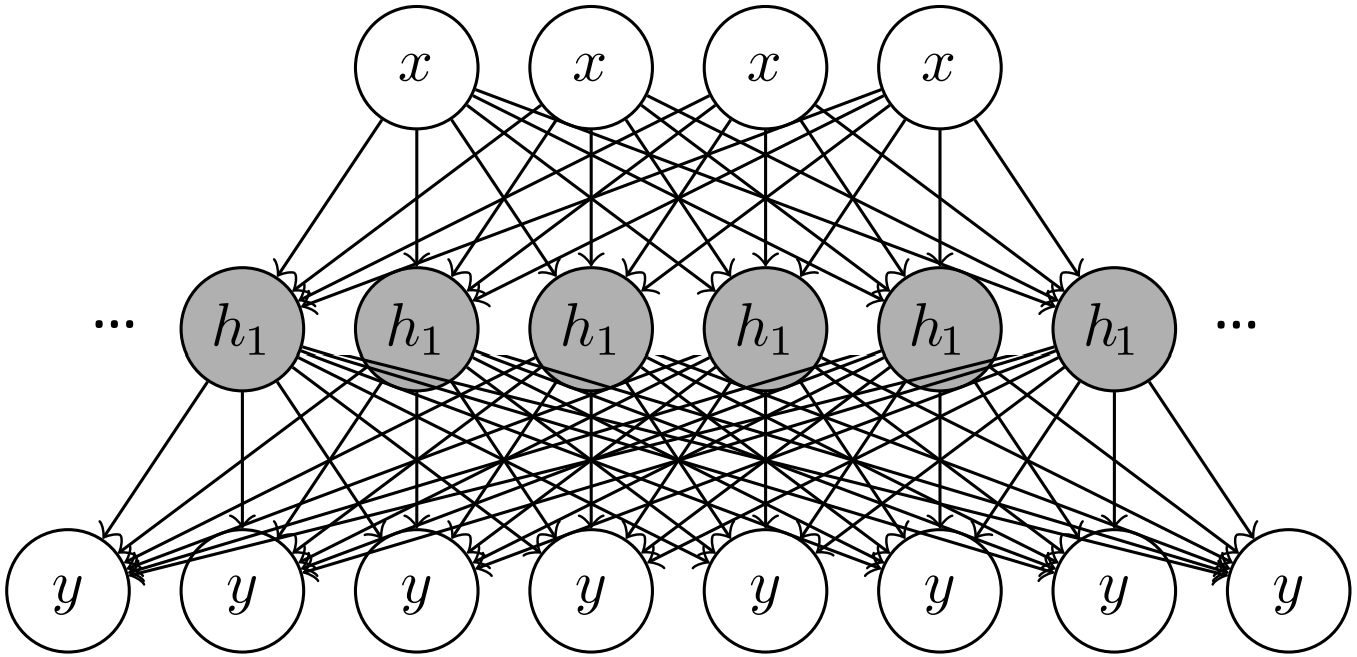
To compute R and R', we perform k iterations of Lanczos to achieve $(\tilde{K}_{XX} + \sigma^2 I) \approx Q_k T_k Q_k^\top$ using the av-

R is *m x t*

GP Limit properties

Single layer, infinite width NN → GP

Also assuming Gaussian i.i.d noise for weights and biases



GP

Neal 1994; Williams 1997

Conditions for limit results

- Independence (i.i.d. noise)
- Bounded variance
- #nodes → Inf

Then use multivariate Central Limit Theorem.

Analytic covariance function (1 layer)

Single layer network:

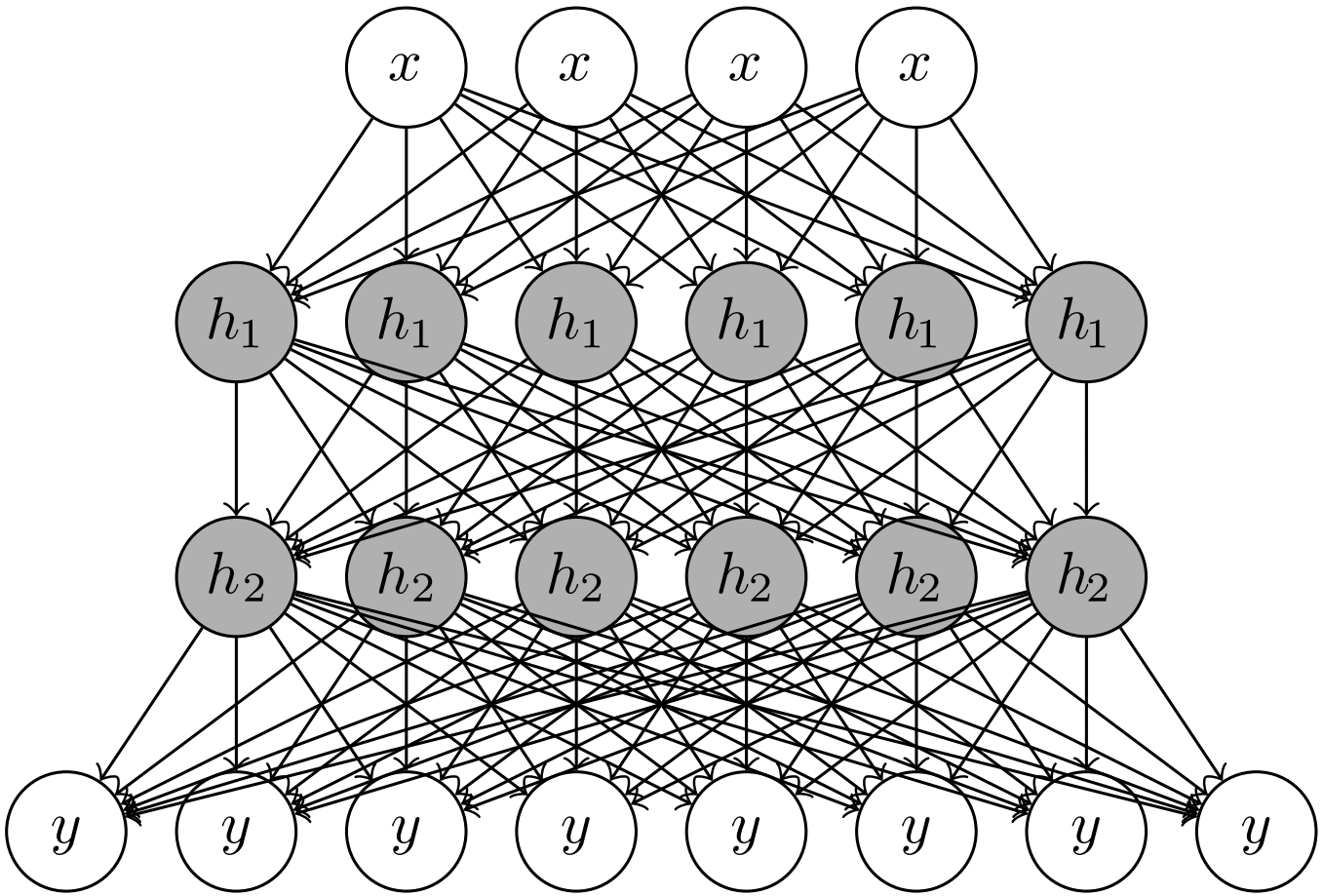
$$z_i^l(x) = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l x_j^l(x), \quad x_j^l(x) = \phi(z_j^{l-1}(x)).$$

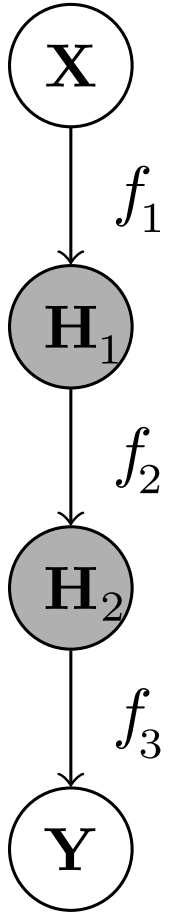
Corresponding GP cov. function:

$$K^1(x, x') \equiv \mathbb{E}\left[z_i^1(x) z_i^1(x')\right] = \sigma_b^2 + \sigma_w^2 \, \mathbb{E}\left[x_i^1(x) x_i^1(x')\right] \equiv \sigma_b^2 + \sigma_w^2 C(x, x').$$

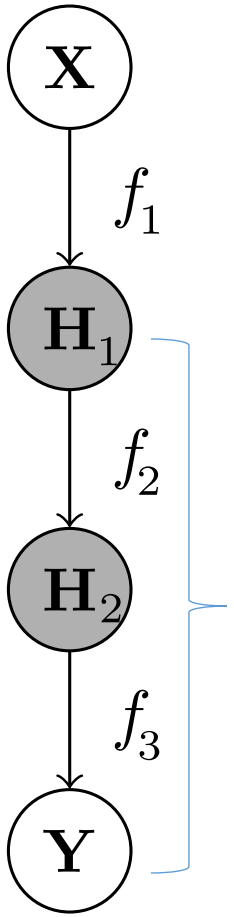
Lee et al. 2018, Matthews et al. 2018

Extension to deep models





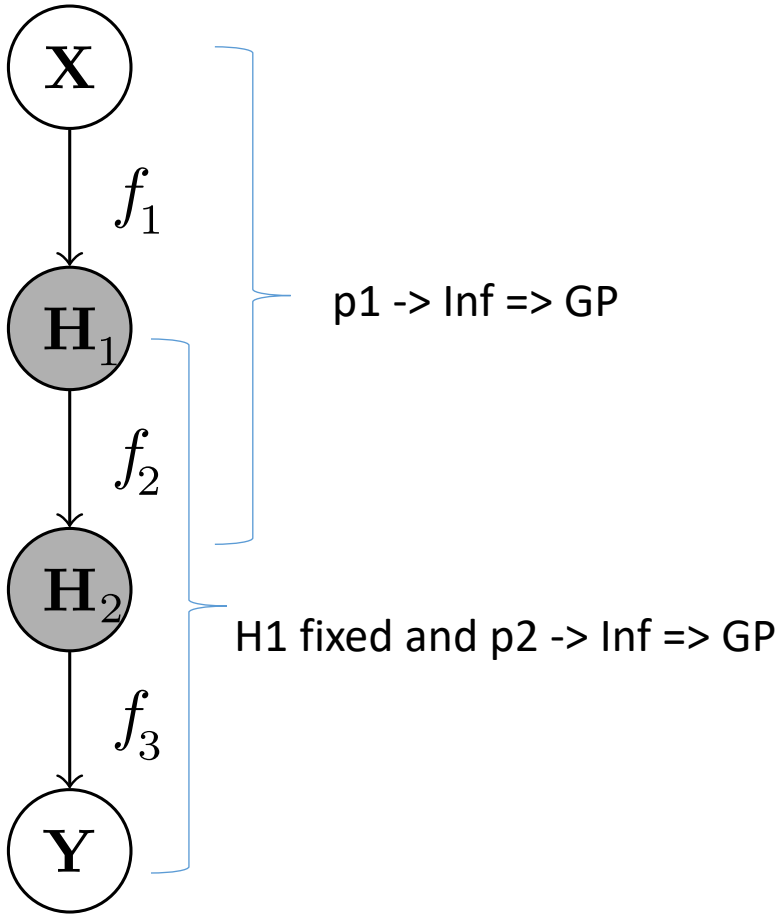
How to generalize the argument for >1 layers?



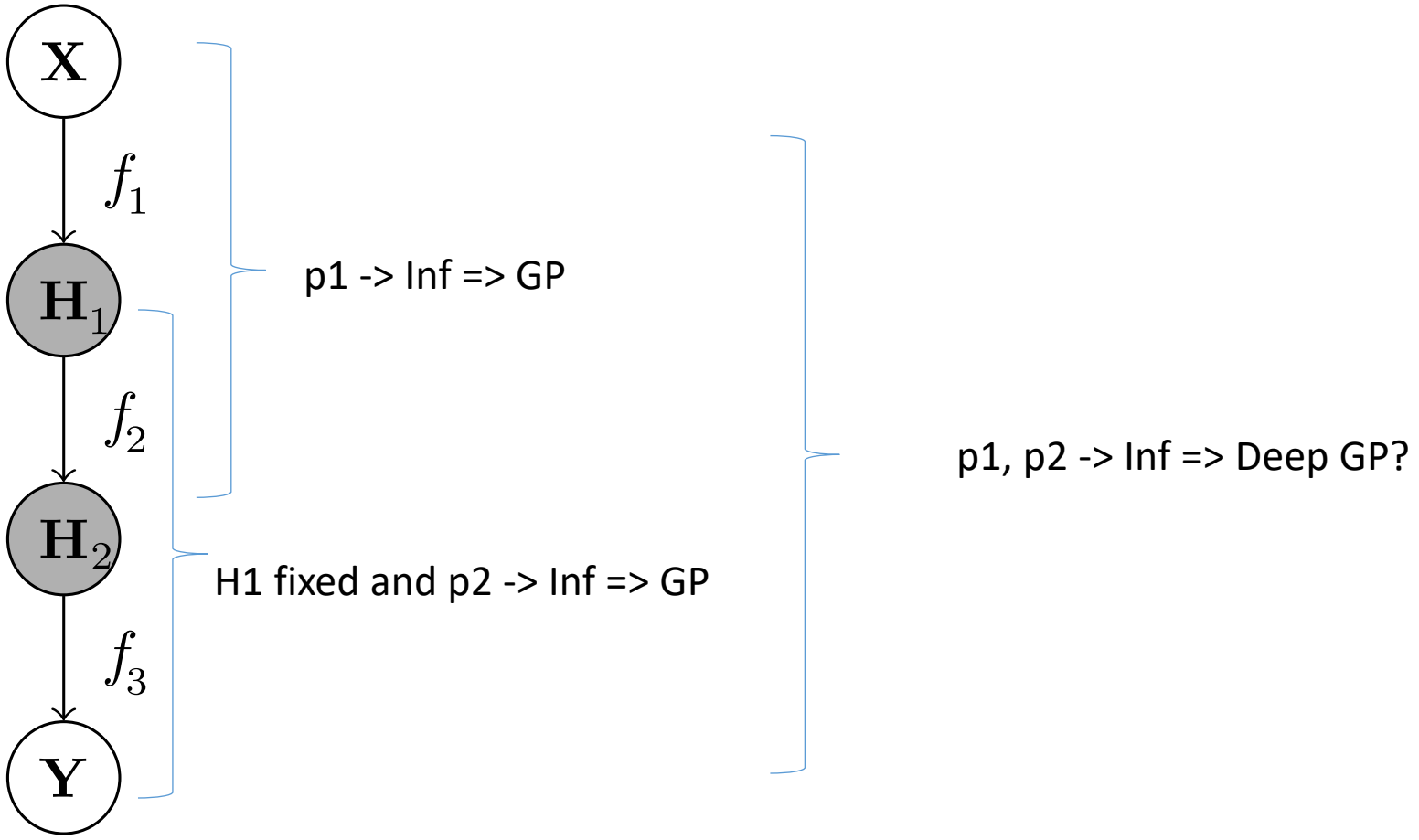
H1 fixed and p2 -> Inf => GP

(Neal 1994)

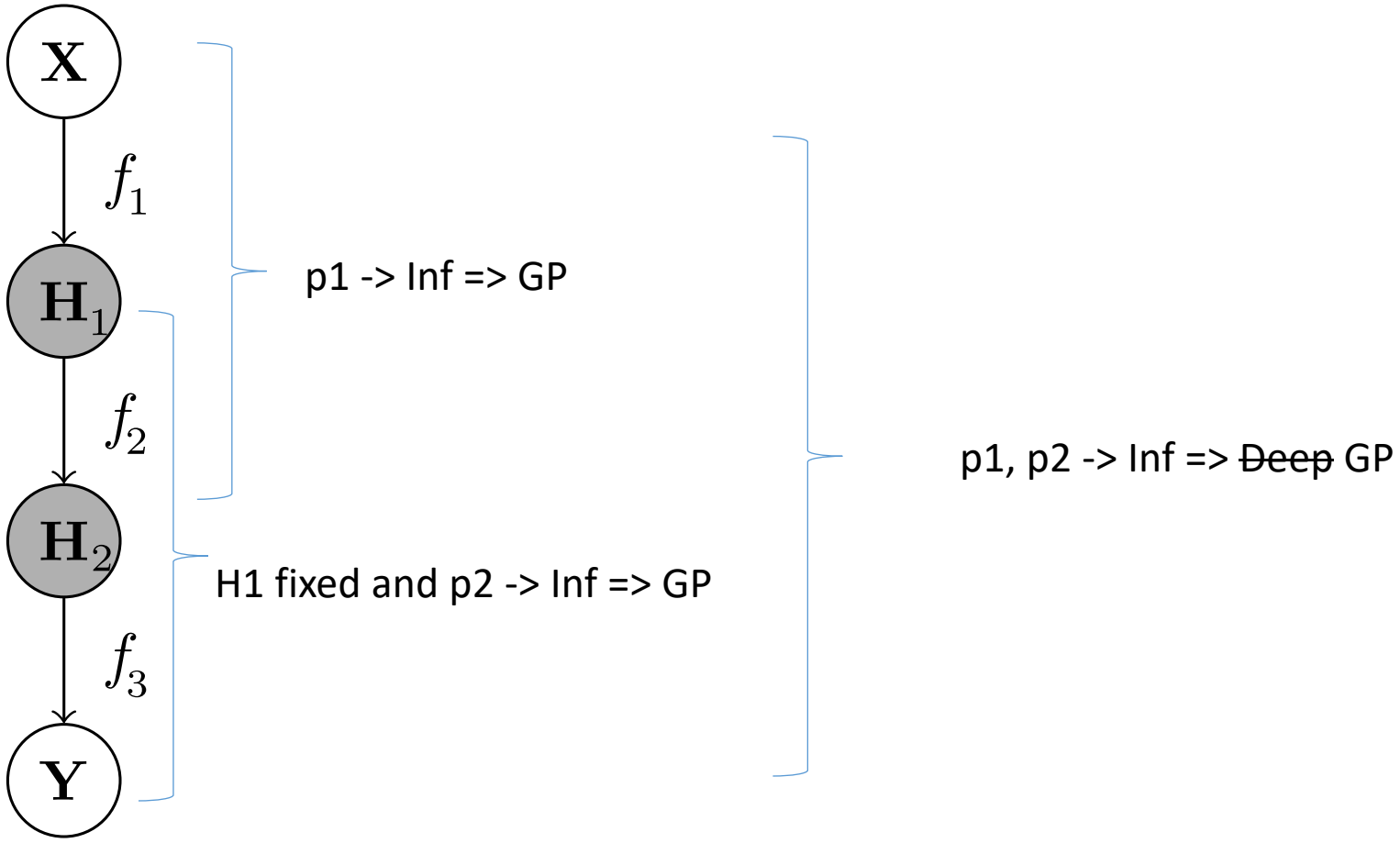
How to generalize the argument for >1 layers?



How to generalize the argument for >1 layers?



How to generalize the argument for >1 layers?



Lee et al., Matthews et al.

Recursive cov. function formulation

$$K^l(x, x') \equiv \mathbb{E}\left[z_i^l(x) z_i^l(x')\right]$$

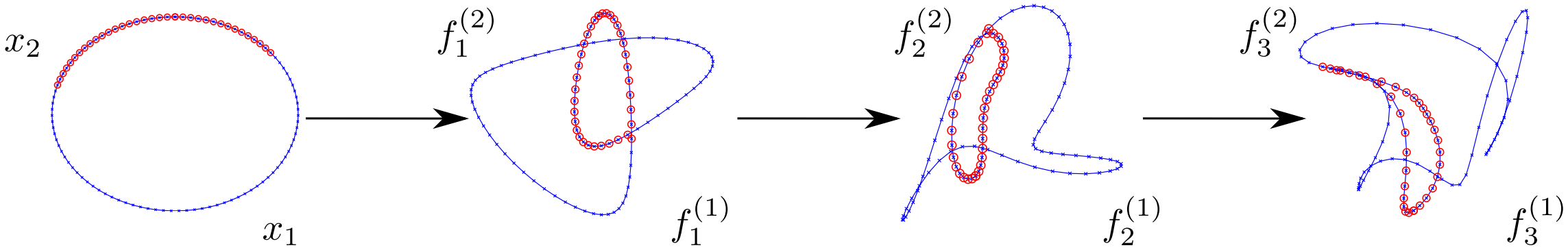
$$= \sigma_b^2 + \sigma_w^2 \, \mathbb{E}_{z_i^{l-1} \sim \mathcal{GP}(0, K^{l-1})}\left[\phi(z_i^{l-1}(x))\phi(z_i^{l-1}(x'))\right]$$

$$= \sigma_b^2 + \sigma_w^2 \, F_\phi\Big(K^{l-1}(x, x'),\, K^{l-1}(x, x),\, K^{l-1}(x', x')\Big)$$

DEEP GP

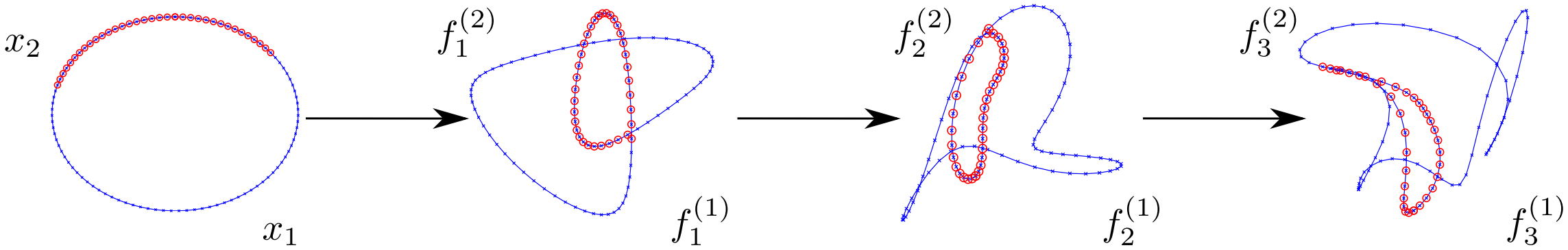
Feature learning

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.

DNN vs DGP

- Shallow NN → GP
- Deep NN → GP

DNN vs DGP

- Shallow NN → GP
- Deep NN → GP

Deep GP > Deep NN?
(Discussion)

Analysing DGPs

Limit properties

- Distribution of derivatives in deep models [Duvenaud et al. ‘14]
- How are effective depth and ergodicity connected? [Dunlop et al. ’18]

Few layers analysis through approximation:

- DGP moments and approximation of DGP with GP [Lu et al. ‘19]

Analysing DGPs

Limit properties

- Distribution of derivatives in deep models [Duvenaud et al. ‘14]
- How are effective depth and ergodicity connected? [Dunlop et al. ’18]

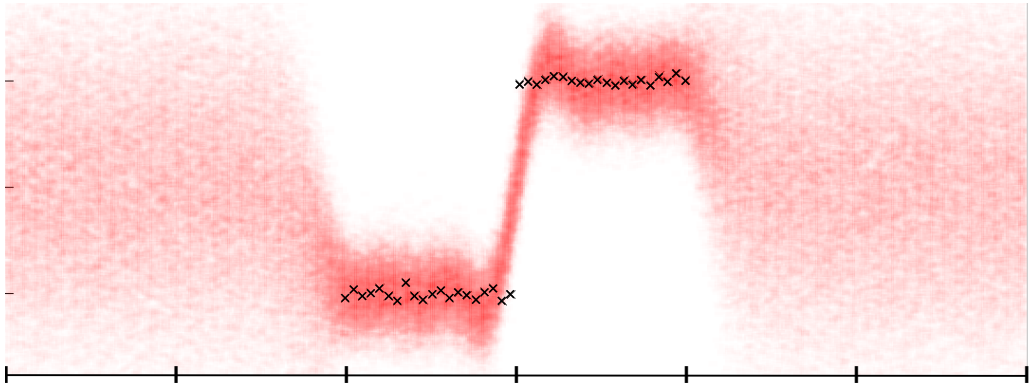
Few layers analysis through approximation:

- DGP moments and approximation of DGP with GP [Lu et al. ‘19]

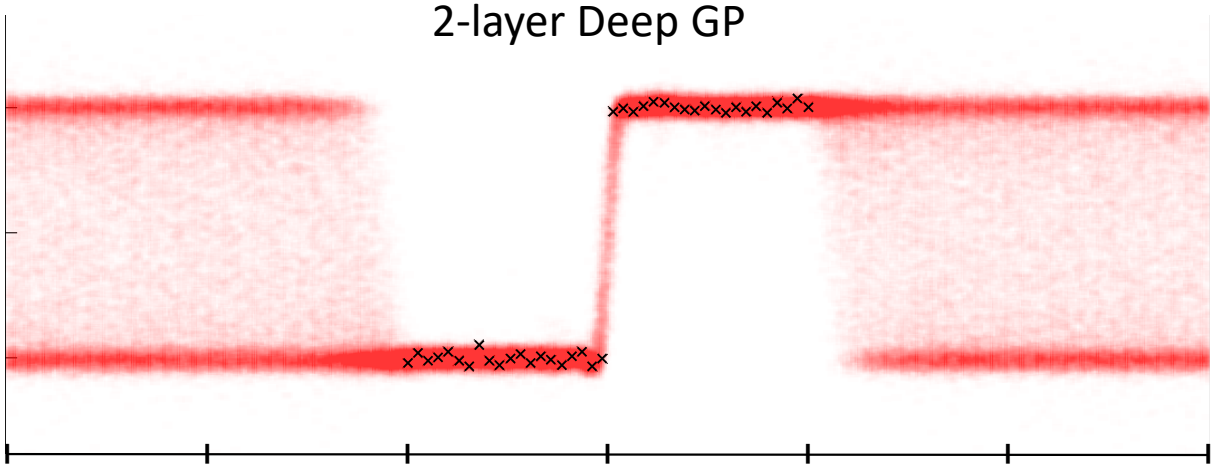
Discussion: What does a DGP with > 2 layers mean?

Step function example

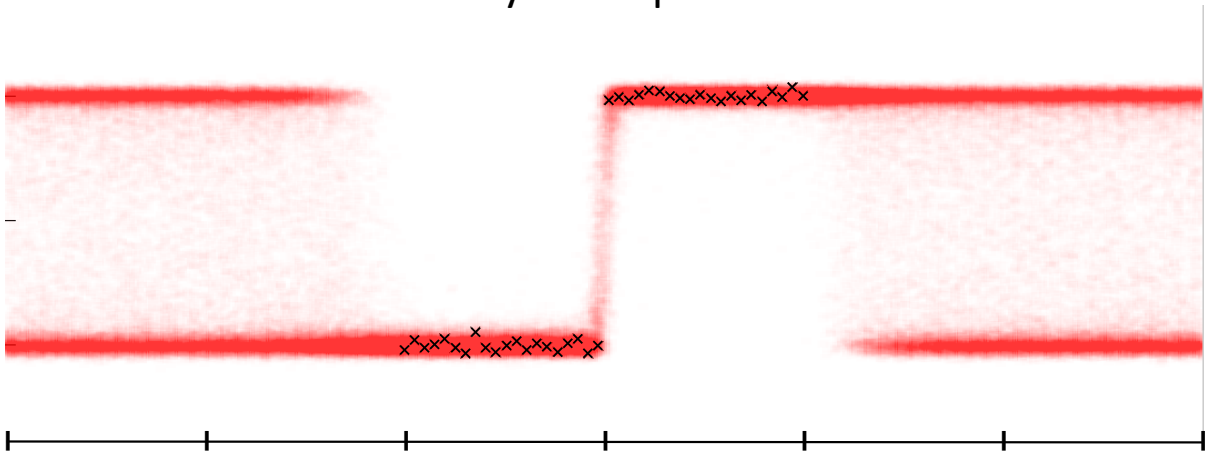
Single GP



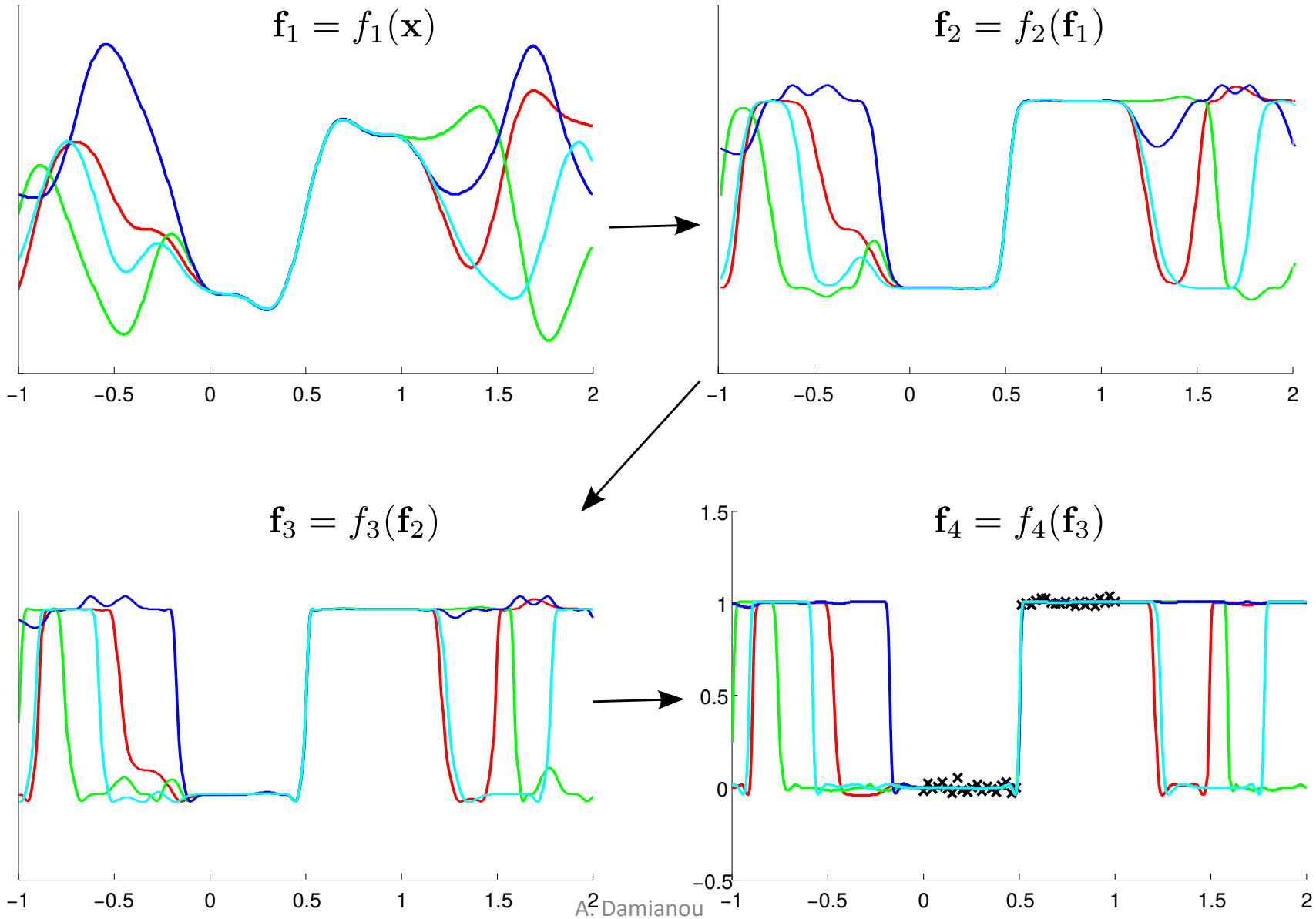
2-layer Deep GP



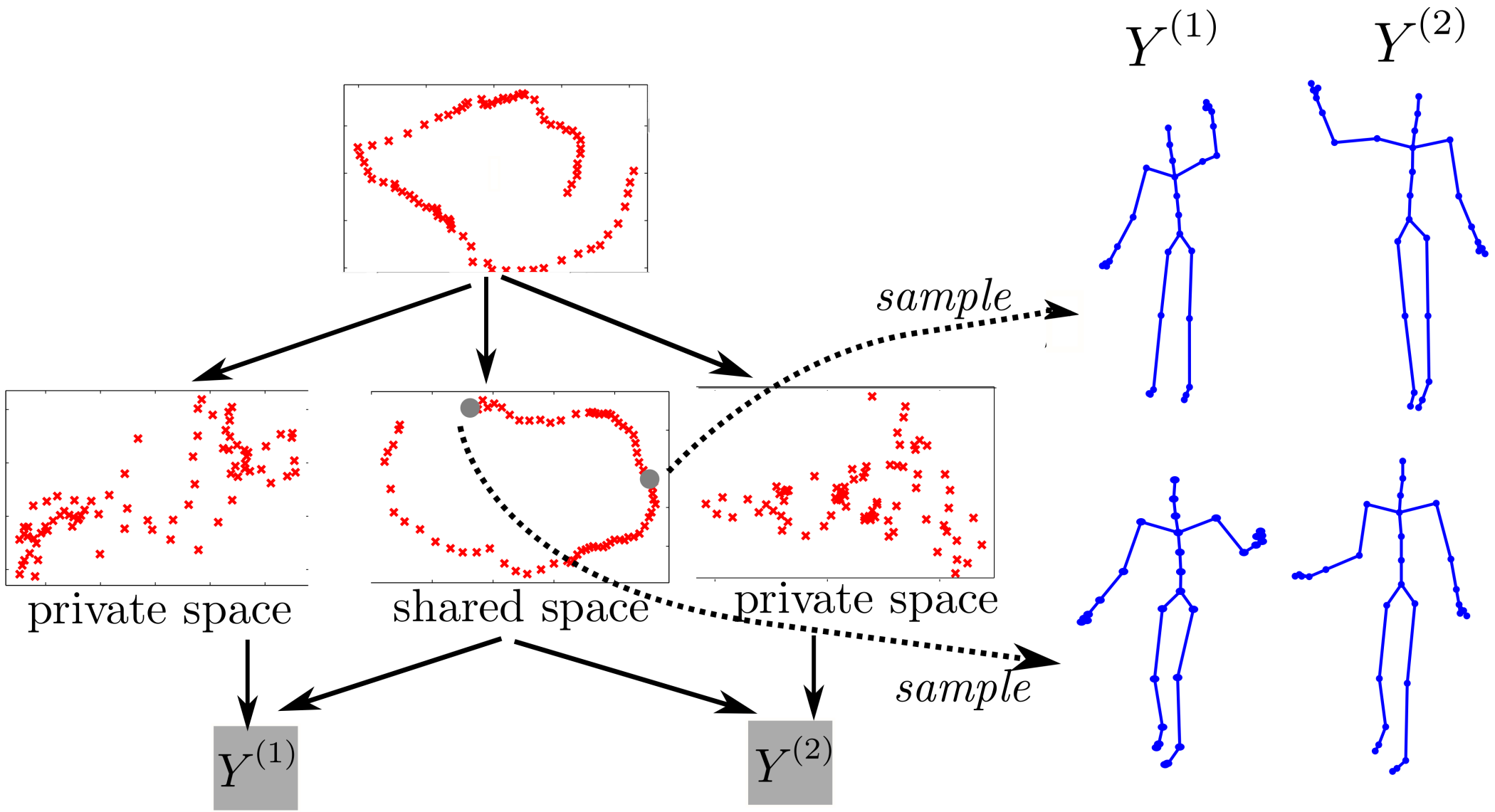
4-layer Deep GP



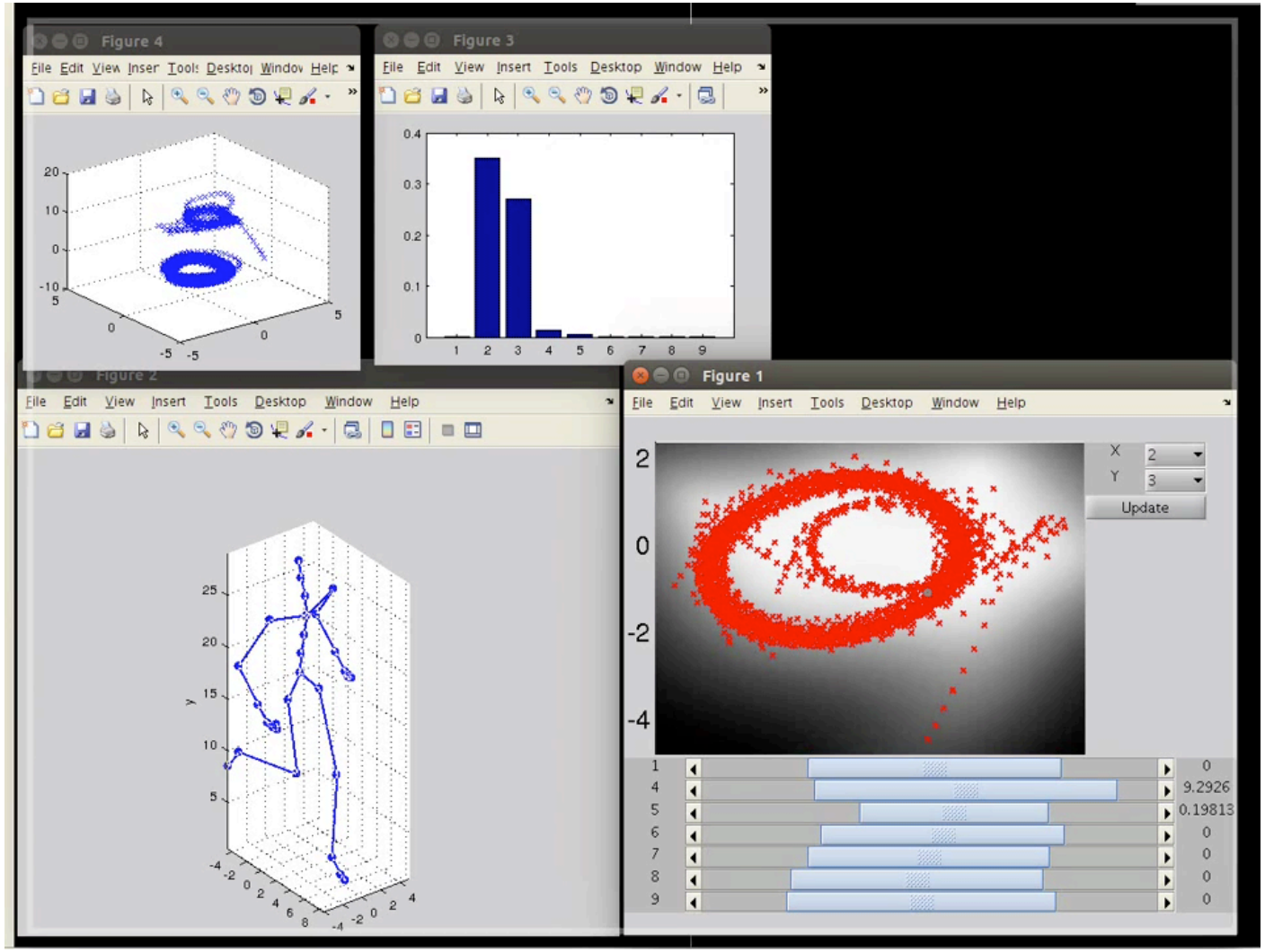
Successive warping to learn the step function



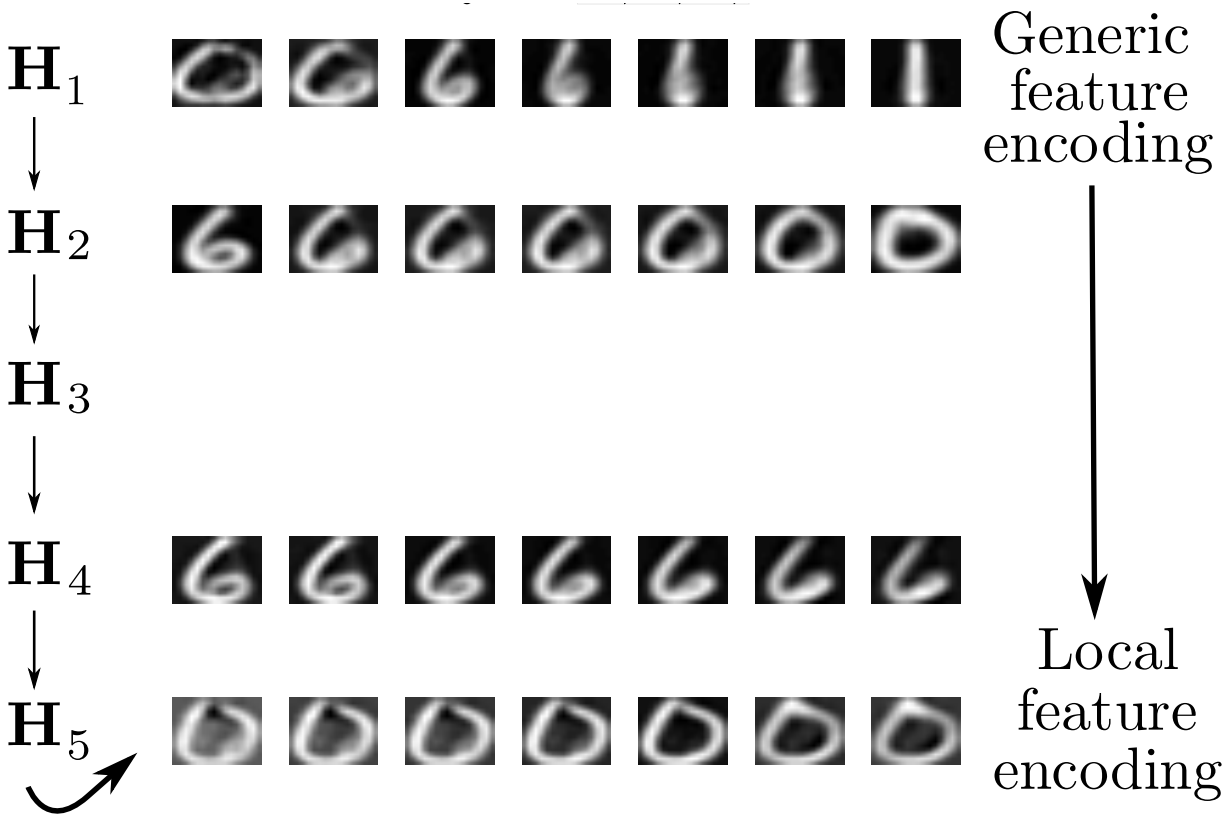
Unsupervised learning for multiple views



Generative modelling



https://youtu.be/fHDWloJtgk8


H1
H2
H3
H4
H5
Generic feature encoding
Local feature encoding

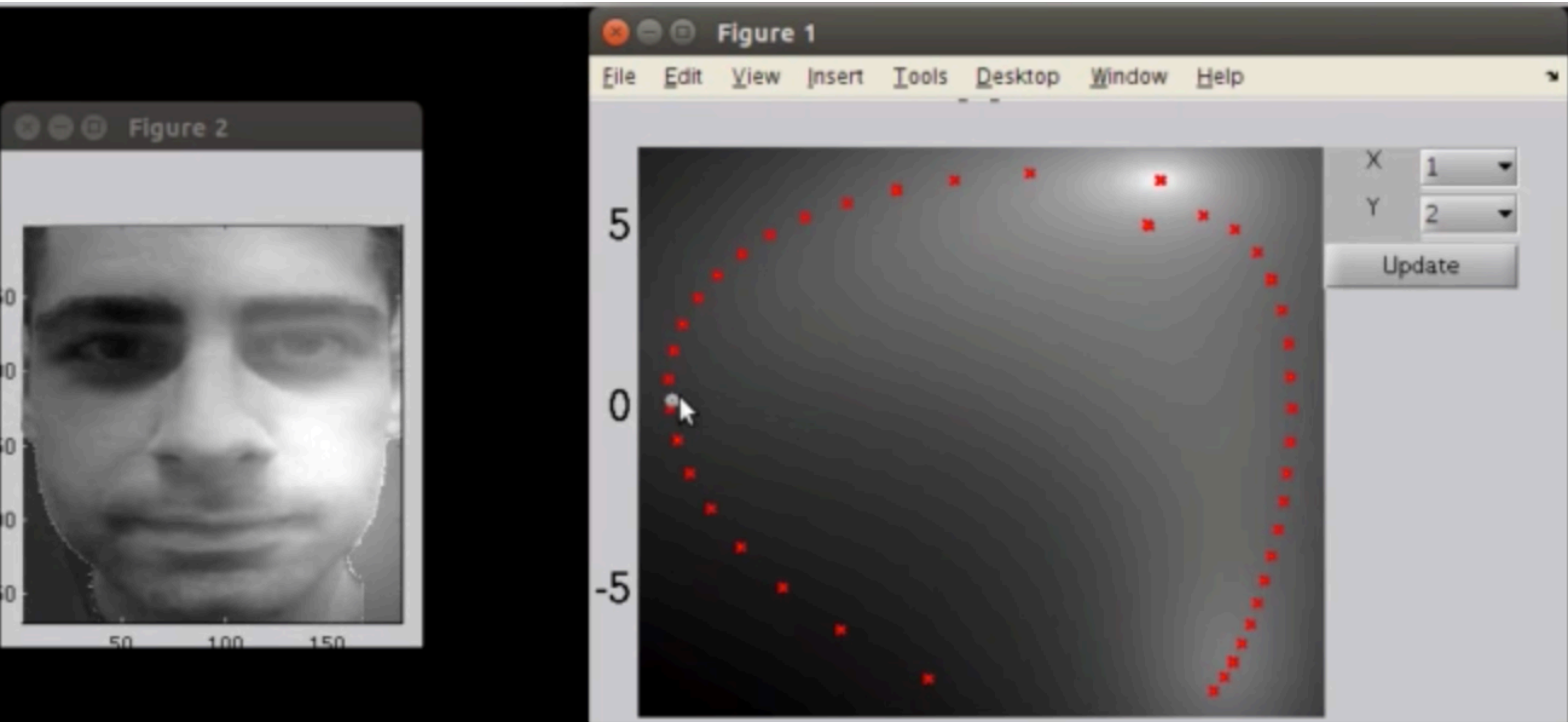


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
5
0
-5
X
1
Y
2
Update

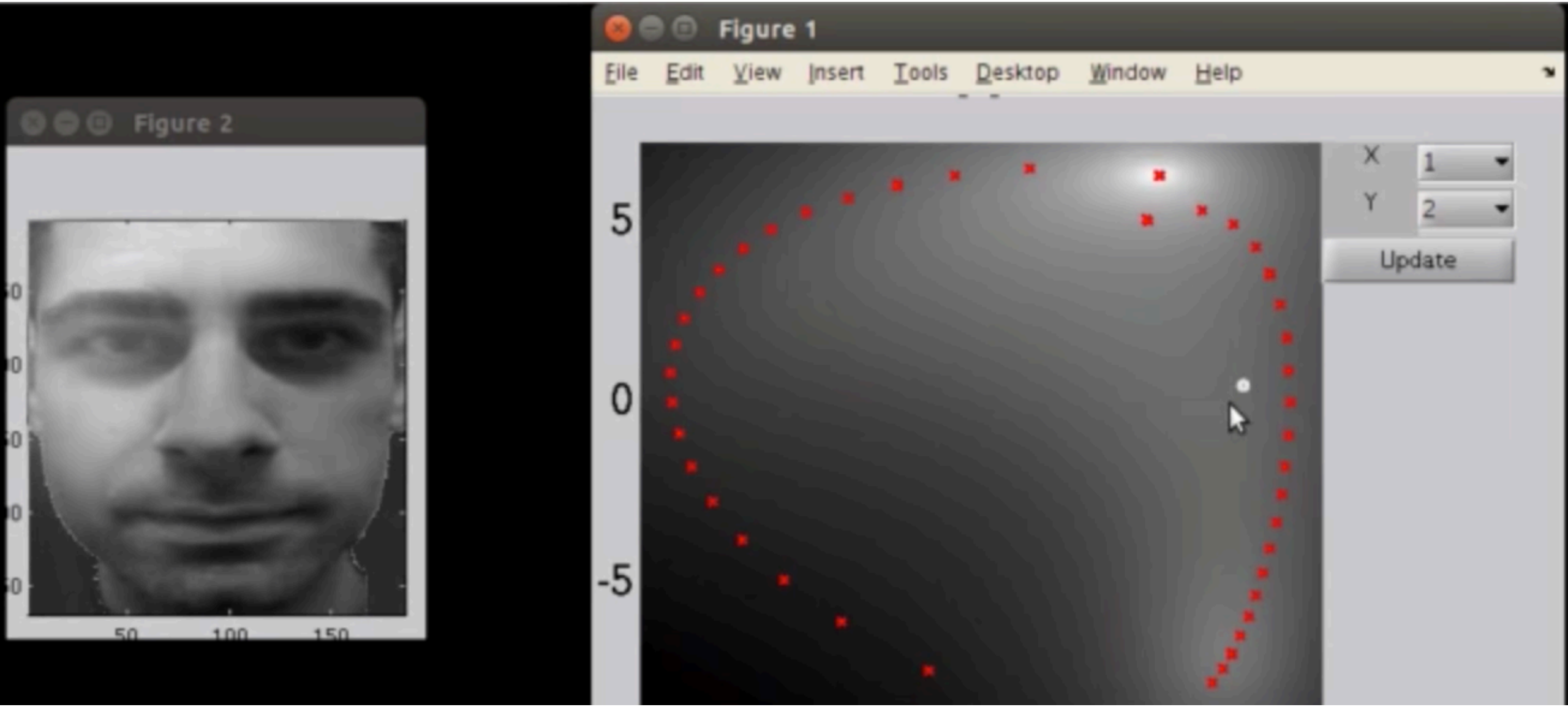


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
5
0
-5
X
1
Y
2
Update

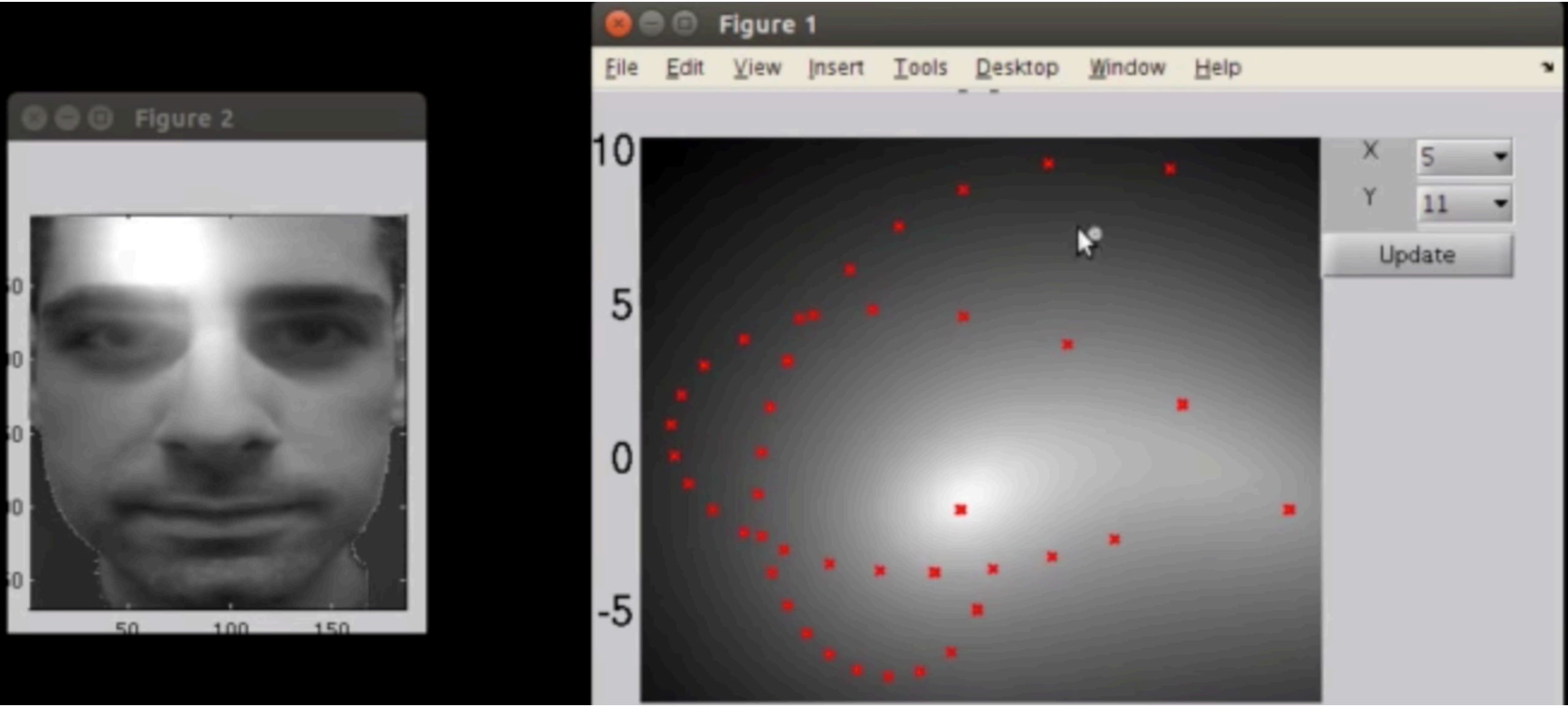


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
10
5
0
-5
X
5
Y
11
Update

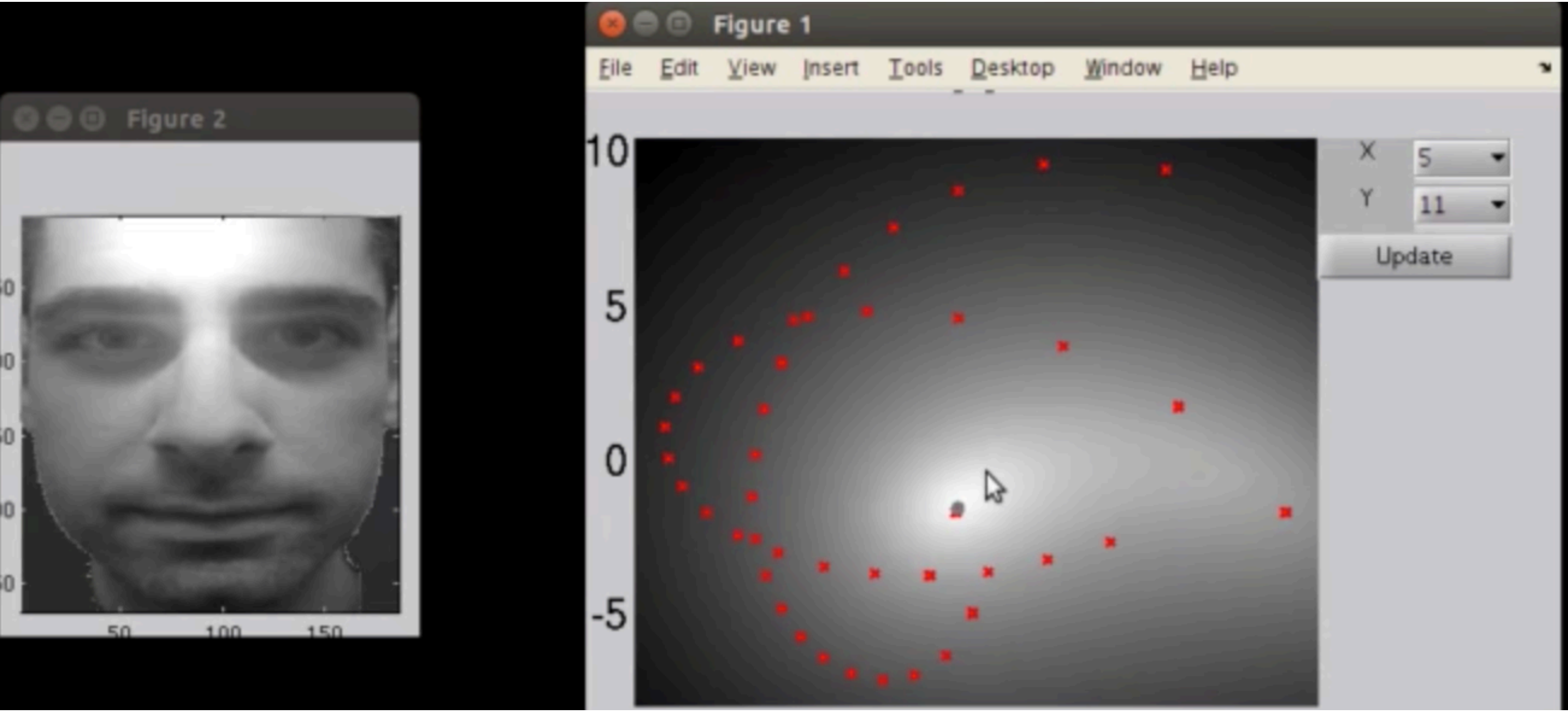


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
10
5
0
-5
X
5
Y
11
Update

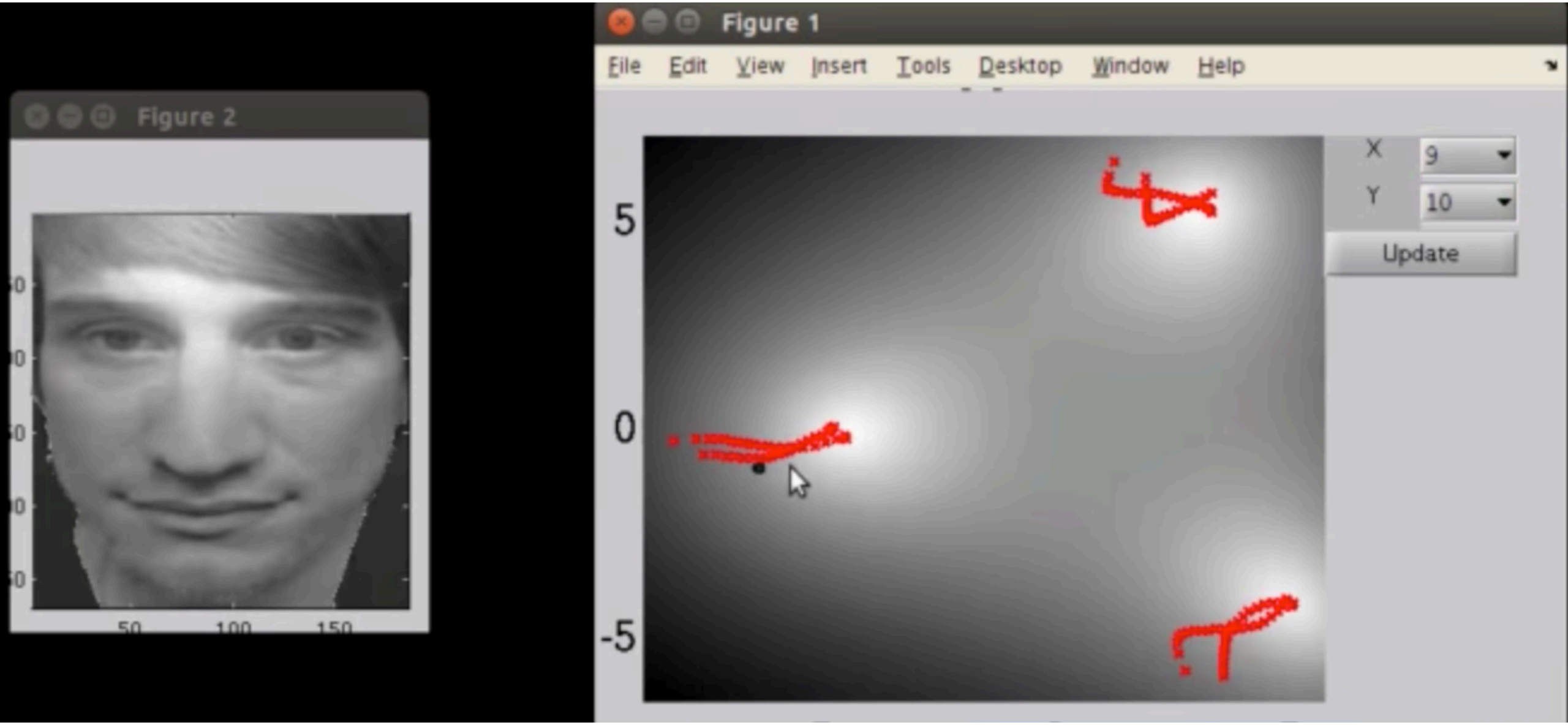


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
5
0
-5
X
9
Y
10
Update

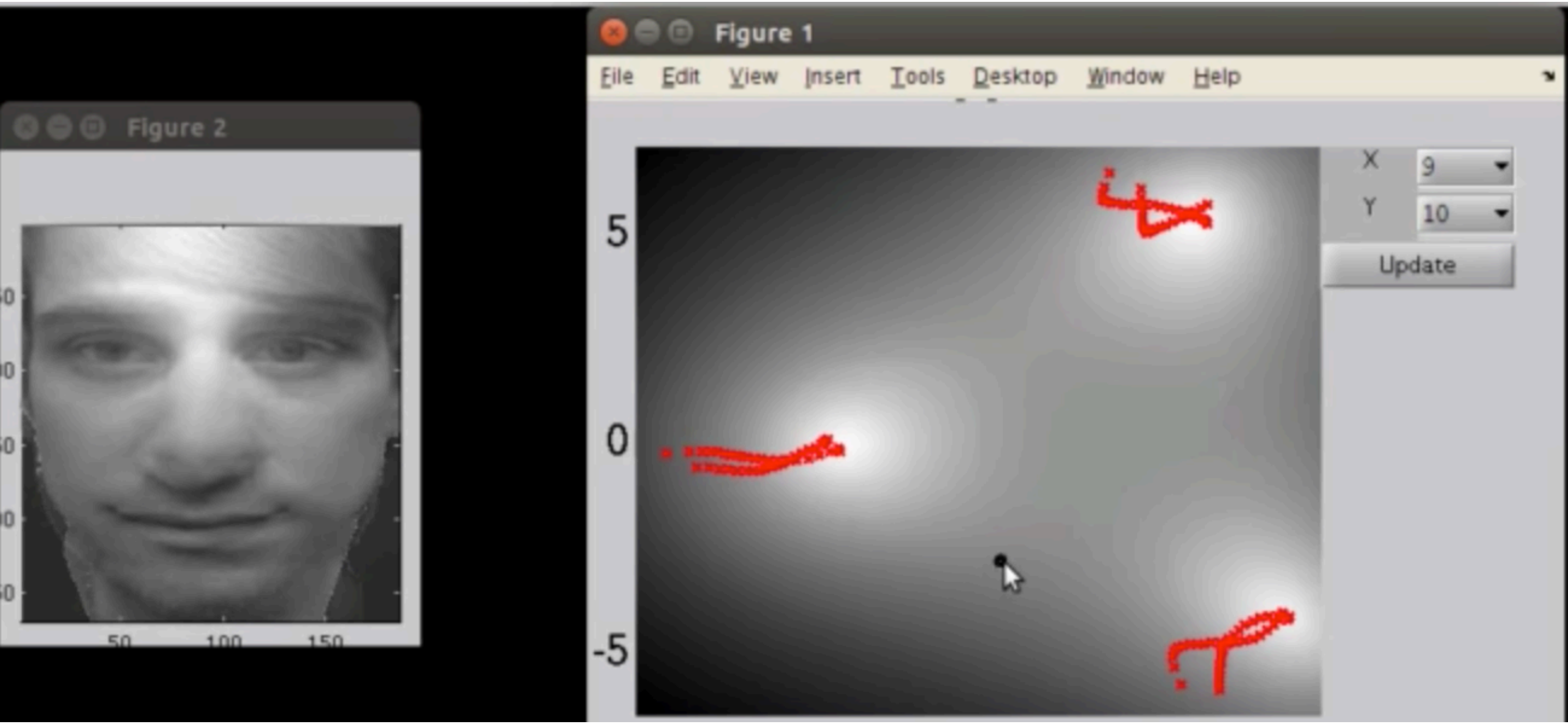


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
5
0
-5
X
9
Y
10
Update

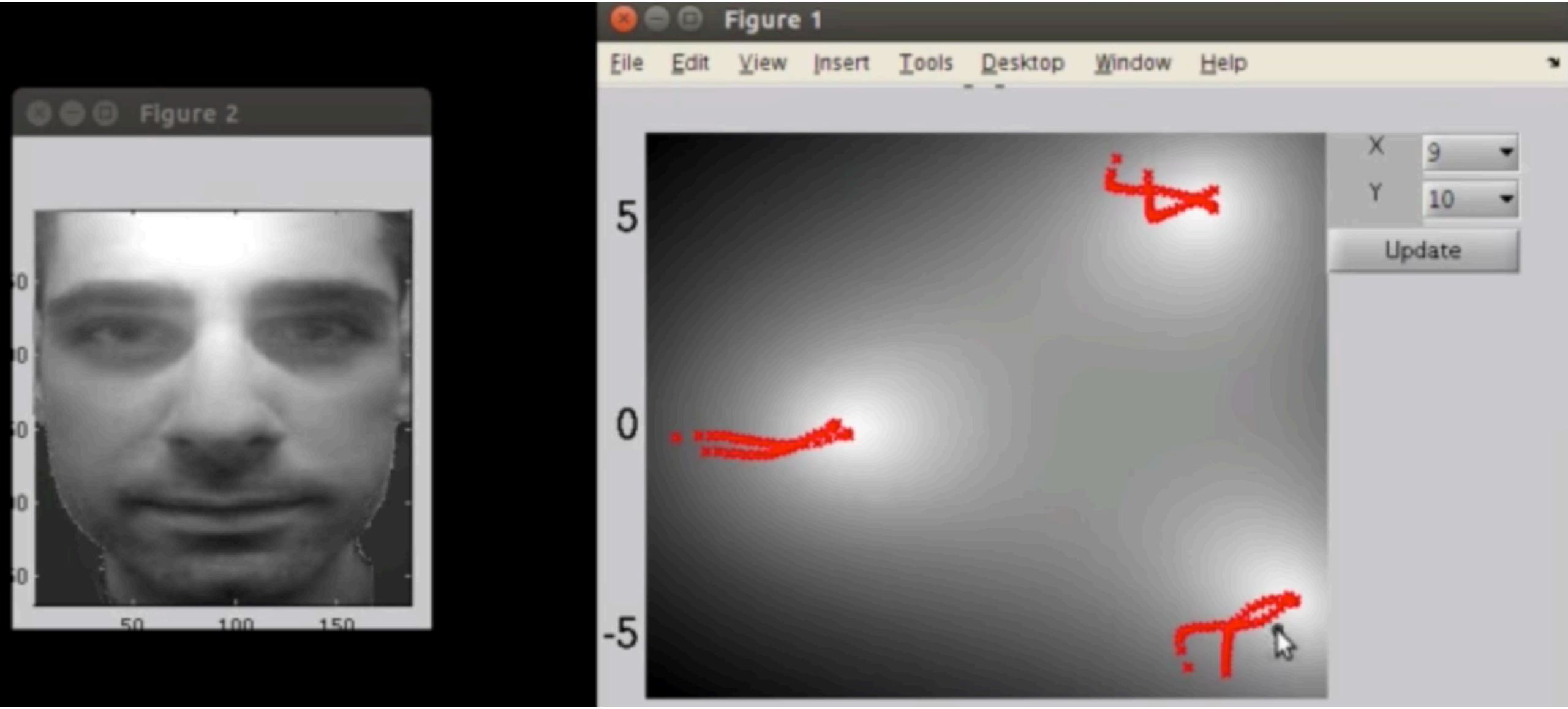


Figure 2
50
100
150
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
5
0
-5
X
9
Y
10
Update

Properties

- Unsupervised learning possible due to Bayesian regularization
- Very data efficient
- Scalability also possible with newer techniques

- Intractable objective
- Classification is more challenging

Properties

- Unsupervised learning possible due to Bayesian regularization
- Very data efficient
- Scalability also possible with newer techniques

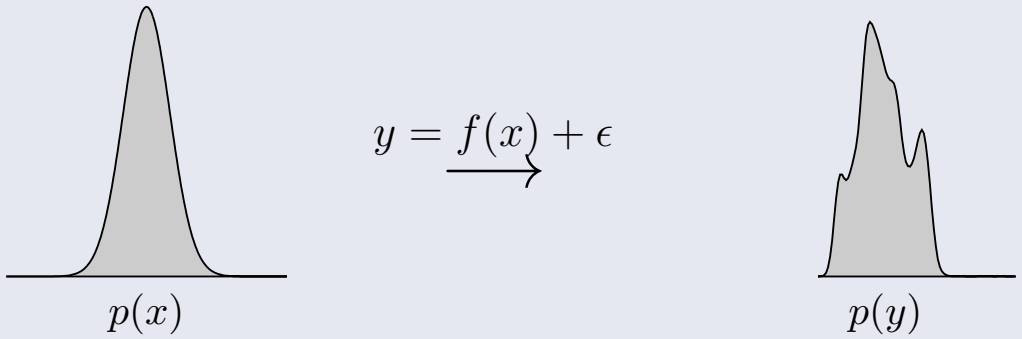
- Intractable objective
- Classification is more challenging

Inference in Deep GPs: uncertainty propagation

- Objective: $p(y|x) = \int_{h_2} \left(p(y|h_2) \int_{h_1} p(h_2|h_1) p(h_1|x) \right)$

- $p(h_2|x) = \int_{h_1, f_2} p(h_2|f_2) \underbrace{p(f_2|h_1)}_{\text{contains } (k(h_1, h_1))^{-1}} p(h_1|x)$

Propagating uncertainty through non-linearities:



Multi-fidelity

Multi-fidelity data

High fidelity observations

Low fidelity observations

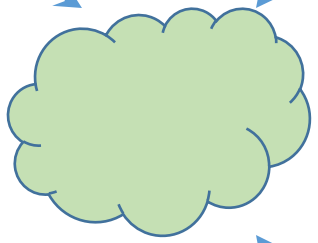
High fidelity simulations

Low fidelity simulations

Multi-fidelity data

High fidelity observations

Low fidelity observations



High fidelity simulations

Low fidelity simulations

X_H

Y_H =

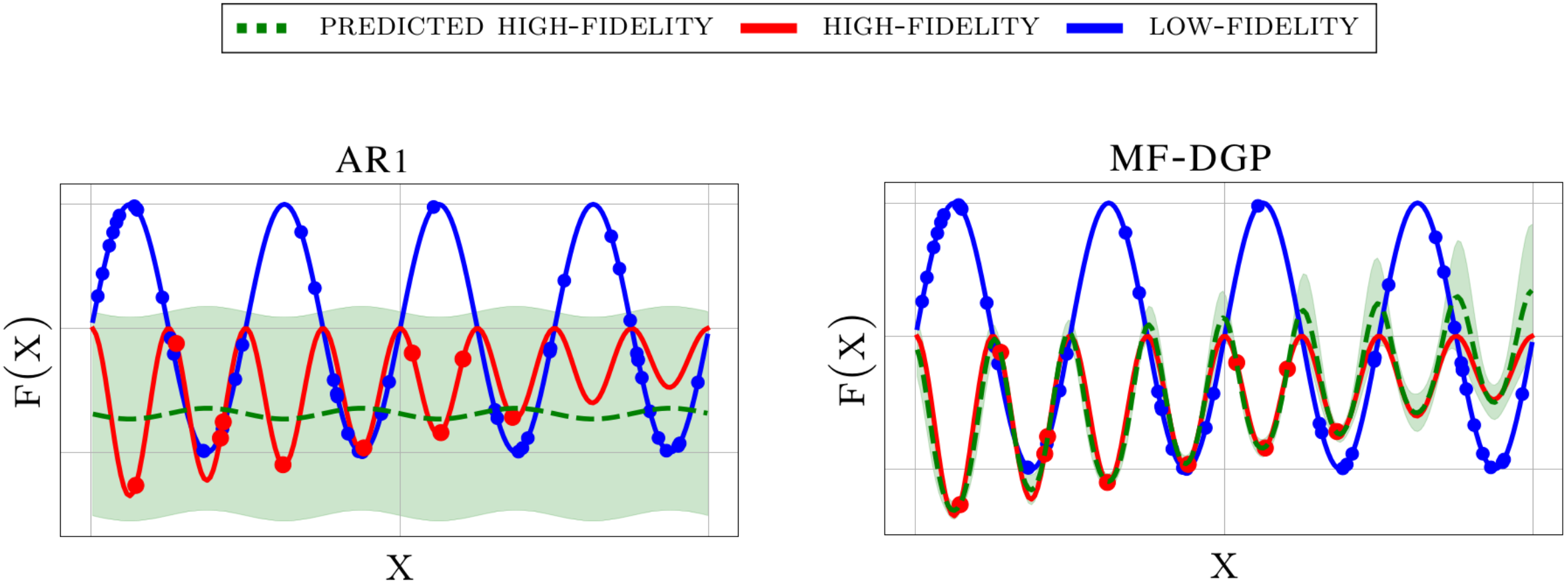
OK
OK
OK
OK
ERROR!
OK
OK

X_L

Y_L =

OK
OK
ERROR!
OK
ERROR!
OK
ERROR!

Fusing information from multiple fidelities



We want to trust the high-fidelity data, where we have them, and where we don't have them to learn how to reason based on low-fidelity data.

Linear GP multi-fidelity

$$f_H(x) = \rho_H f_L(x) + \delta_H(x)$$

f_H High fidelity function (GP)

ρ_H Contribution of low fidelity (const)

f_L Low fidelity function (GP)

δ_H Bias between fidelities (GP)

Kennedy & O'Hagan 2000, Le Gratiet & Garnier 2014

Non-linear multi-fidelity GP => Deep GP

$$f_H(x) = \rho_H f_L(x) + \delta_H(x)$$

Linear relationship between fidelities

$$f_H(x) = \rho_H(f_L(x), x) + \delta_H(x)$$

Non-linear relationship between fidelities
(if ρ is a GP -> overall a deep GP!)

Non-linear multi-fidelity GP => Deep GP

$$f_H(x) = \rho_H f_L(x) + \delta_H(x)$$

Linear relationship between fidelities

$$f_H(x) = \rho_H(f_L(x), x) + \delta_H(x)$$

Non-linear relationship between fidelities
(if ρ is a GP -> overall a deep GP!)

$$\Downarrow$$

$$f_H(x) = g_H(f_L^*(x), x)$$

δ absorbed into g

$f_L^*(x)$ denotes the posterior of the GP modeling the low-fidelity data.

Perdikaris et al., Proc. Royal Society, 2017

Algorithm

1. Train f_L on (X_L, Y_L)
2. Compute $f_L^*(X_H)$
3. Train f_H on $((X_L, Y_L), f_L^*(X_H))$

Predictions

$$p(f_H^*(x^*)) = \int \underbrace{p(f_H(x^*, f_L^*(x^*))|y_H, x_H, x^*)}_{\text{Local posterior from fidelity } H} \underbrace{p(f_L^*(x^*))}_{\text{Predictive from fidelity } L} df_L^*$$