

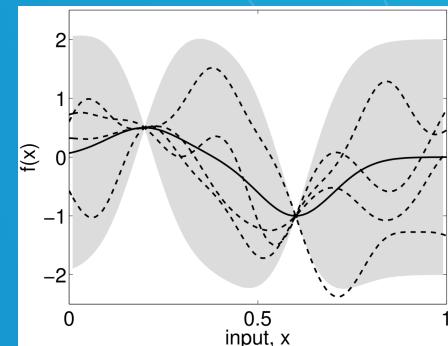
# Probabilistic Graphical Models

## Gaussian Processes

Maruan Al-Shedivat

Lecture 26, April 22, 2019

Reading: see class homepage

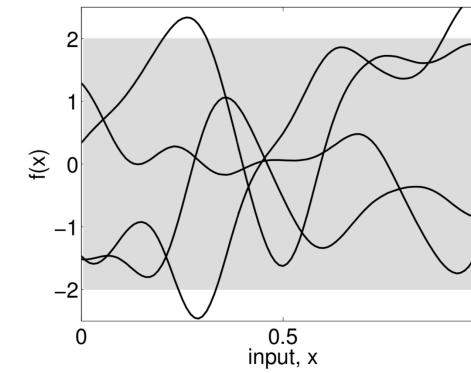




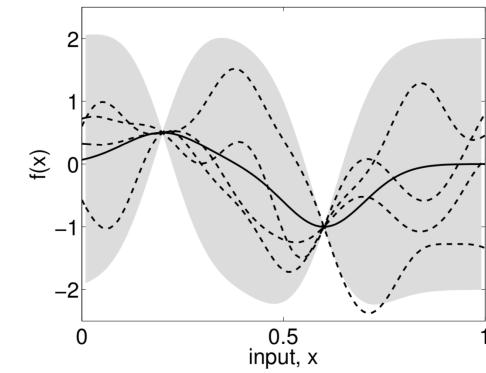
GPM book

# Outline

- Part 1: Gaussian Processes (GPs)
  - Definition of the GP
  - Covariance kernels over various input spaces
  - Inference and learning with GPs
  - Deep (structured) kernel learning
  - Scalability of GPs
- Part 2: Elements of meta-learning
  - The one-/few-shot learning problem
  - Neural processes: conditional, attentive, etc.
  - Conditional parameter generation models



(a), prior

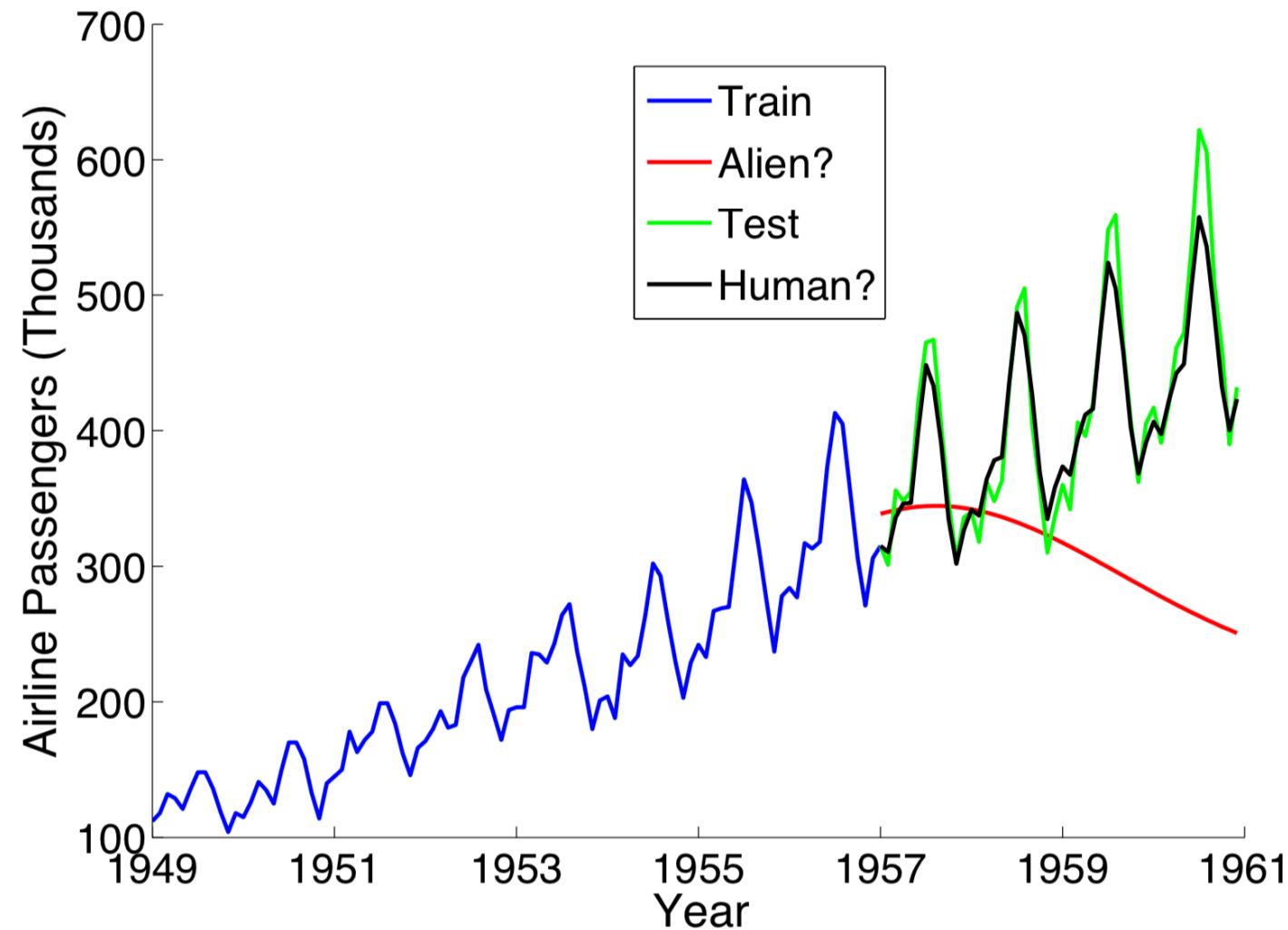


(b), posterior



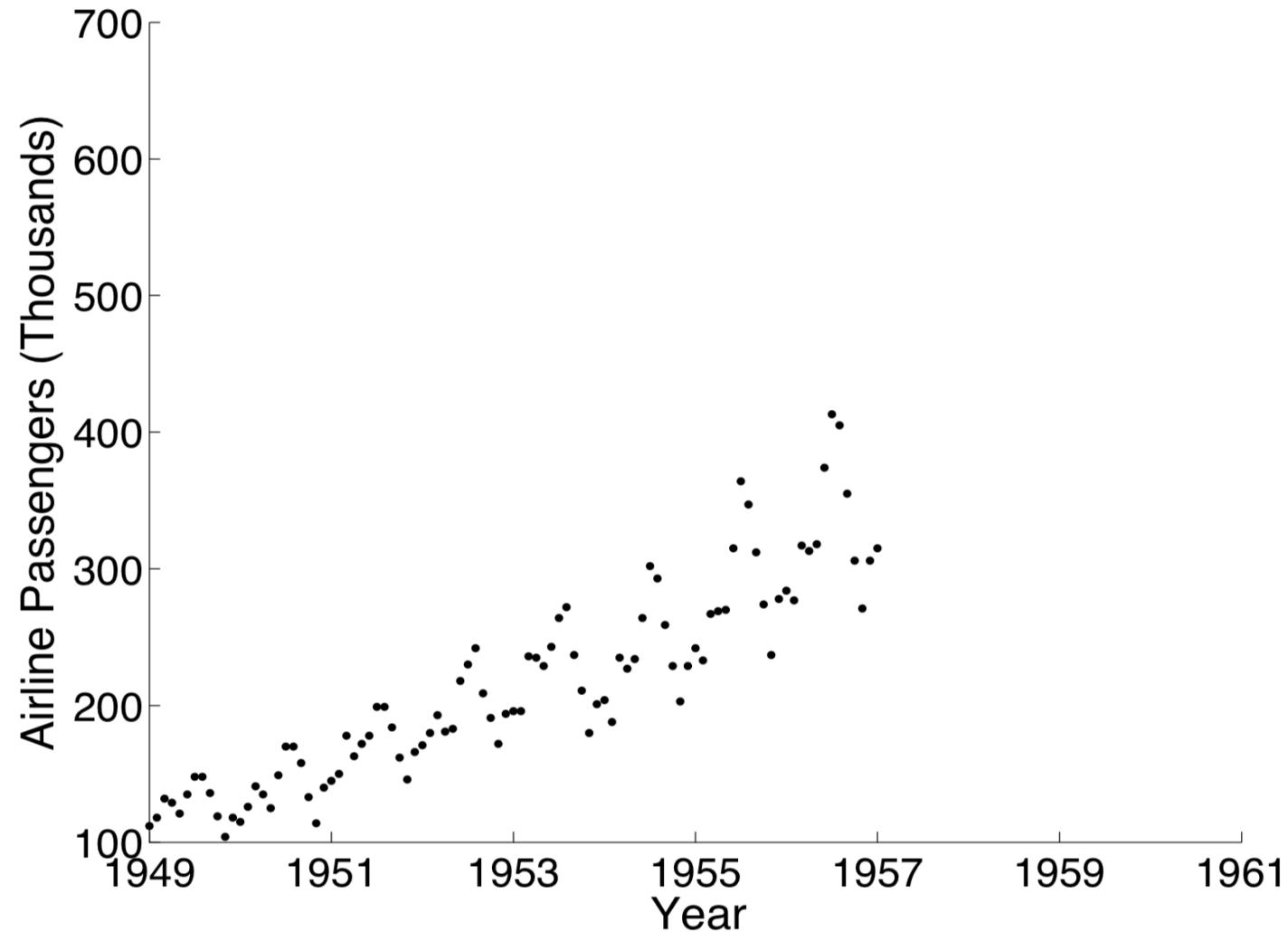


# Function Learning Example





# Learning Functions from Data





# Learning Functions from Data

Guess the parametric form of a function that could fit the data

- ▶  $f(x, \mathbf{w}) = \mathbf{w}^T x$  [Linear function of  $\mathbf{w}$  and  $x$ ]
- ▶  $f(x, \mathbf{w}) = \mathbf{w}^T \phi(x)$  [Linear function of  $\mathbf{w}$ ] (Linear Basis Function Model)
- ▶  $f(x, \mathbf{w}) = g(\mathbf{w}^T \phi(x))$  [Non-linear in  $x$  and  $\mathbf{w}$ ] (E.g., Neural Network)

$\phi(x)$  is a vector of basis functions. For example, if  $\phi(x) = (1, x, x^2)$  and  $x \in \mathbb{R}^1$  then  $f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$  is a quadratic function.





# Learning Functions from Data

Guess the parametric form of a function that could fit the data

- ▶  $f(x, \mathbf{w}) = \mathbf{w}^T x$  [Linear function of  $\mathbf{w}$  and  $x$ ]
- ▶  $f(x, \mathbf{w}) = \mathbf{w}^T \phi(x)$  [Linear function of  $\mathbf{w}$ ] (Linear Basis Function Model)
- ▶  $f(x, \mathbf{w}) = g(\mathbf{w}^T \phi(x))$  [Non-linear in  $x$  and  $\mathbf{w}$ ] (E.g., Neural Network)

$\phi(x)$  is a vector of basis functions. For example, if  $\phi(x) = (1, x, x^2)$  and  $x \in \mathbb{R}^1$  then  $f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$  is a quadratic function.

Choose an error measure  $E(\mathbf{w})$ , minimize with respect to  $\mathbf{w}$

- ▶  $E(\mathbf{w}) = \sum_{i=1}^N [f(x_i, \mathbf{w}) - y(x_i)]^2$





# Learning Functions from Data

## A probabilistic approach

We could explicitly account for noise in our model.

- ▶  $y(x) = f(x, \mathbf{w}) + \epsilon(x)$ , where  $\epsilon(x)$  is a noise function.





# Learning Functions from Data

## A probabilistic approach

We could explicitly account for noise in our model.

- ▶  $y(x) = f(x, \mathbf{w}) + \epsilon(x)$ , where  $\epsilon(x)$  is a noise function.

One commonly takes  $\epsilon(x) = \mathcal{N}(0, \sigma^2)$  for i.i.d. additive Gaussian noise, in which case

$$p(y(x)|x, \mathbf{w}, \sigma^2) = \mathcal{N}(y(x); f(x, \mathbf{w}), \sigma^2) \quad \text{Observation Model}$$

$$p(\mathbf{y}|x, \mathbf{w}, \sigma^2) = \prod_{i=1}^N \mathcal{N}(y(x_i); f(x_i, \mathbf{w}), \sigma^2) \quad \text{Likelihood}$$





# Learning Functions from Data

## A probabilistic approach

We could explicitly account for noise in our model.

- ▶  $y(x) = f(x, \mathbf{w}) + \epsilon(x)$ , where  $\epsilon(x)$  is a noise function.

One commonly takes  $\epsilon(x) = \mathcal{N}(0, \sigma^2)$  for i.i.d. additive Gaussian noise, in which case

$$p(y(x)|x, \mathbf{w}, \sigma^2) = \mathcal{N}(y(x); f(x, \mathbf{w}), \sigma^2) \quad \text{Observation Model}$$

$$p(\mathbf{y}|x, \mathbf{w}, \sigma^2) = \prod_{i=1}^N \mathcal{N}(y(x_i); f(x_i, \mathbf{w}), \sigma^2) \quad \text{Likelihood}$$

- ▶ Maximize the likelihood of the data  $p(\mathbf{y}|x, \mathbf{w}, \sigma^2)$  with respect to  $\sigma^2, \mathbf{w}$ .





# Learning Functions from Data

- ▶ The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level  $\sigma^2$ .
- ▶ Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development.





# Learning Functions from Data

- ▶ The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level  $\sigma^2$ .
- ▶ Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development.
- ▶ Both approaches are prone to *over-fitting* for flexible  $f(x, \mathbf{w})$ : low error on the training data, high error on the test set.





# Learning Functions from Data

- ▶ The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level  $\sigma^2$ .
- ▶ Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development.
- ▶ Both approaches are prone to *over-fitting* for flexible  $f(x, \mathbf{w})$ : low error on the training data, high error on the test set.

## Regularization

- ▶ Use a penalized log likelihood (or error function), such as

$$\log p(\mathbf{y}|X, \mathbf{w}) \propto \underbrace{-\frac{1}{2\sigma^2} \sum_{i=1}^n (f(x_i, \mathbf{w}) - y(x_i))^2}_{\text{model fit}} - \underbrace{\lambda \mathbf{w}^T \mathbf{w}}_{\text{complexity penalty}} .$$





# Learning Functions from Data

- ▶ The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level  $\sigma^2$ .
- ▶ Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development.
- ▶ Both approaches are prone to *over-fitting* for flexible  $f(x, \mathbf{w})$ : low error on the training data, high error on the test set.

## Regularization

- ▶ Use a penalized log likelihood (or error function), such as

$$\log p(\mathbf{y}|X, \mathbf{w}) \propto \underbrace{-\frac{1}{2\sigma^2} \sum_{i=1}^n (f(x_i, \mathbf{w}) - y(x_i))^2}_{\text{model fit}} - \underbrace{\lambda \mathbf{w}^T \mathbf{w}}_{\text{complexity penalty}} .$$

- ▶ **But how should we define complexity, and how much should we penalize complexity?**
- ▶ Can set  $\lambda$  using *cross-validation*.





# Learning Functions from Data

## Bayes' Rule

$$p(a|b) = p(b|a)p(a)/p(b), \quad p(a|b) \propto p(b|a)p(a).$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \quad p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \sigma^2) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2)p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X}, \sigma^2)}$$





# Learning Functions from Data

## Predictive Distribution

$$p(y|x_*, \mathbf{y}, X) = \int p(y|x_*, \mathbf{w})p(\mathbf{w}|\mathbf{y}, X)d\mathbf{w}.$$





# Statistics From Scratch

## Predictive Distribution

$$p(y|x_*, \mathbf{y}, X) = \int p(y|x_*, \mathbf{w})p(\mathbf{w}|\mathbf{y}, X)d\mathbf{w}.$$

- ▶ Average of infinitely many models weighted by their posterior probabilities.
- ▶ No over-fitting, automatically calibrated complexity.
- ▶ Typically more interested in distribution over functions than in parameters  $\mathbf{w}$ .





# Parametric vs. Nonparametric Modeling

## Parametric models:

- Assume that all data can be represented using a fixed, finite number of parameters.
  - Mixture of K Gaussians, polynomial regression, neural nets, etc.

## Nonparametric models:

- Number of parameters can grow with sample size.
- Number of parameters may be random.
  - Kernel density estimation.

## Bayesian nonparametrics:

- Allow for an infinite number of parameters a priori.
- Models of finite datasets will have only finite number of parameters.
- Other parameters are integrated out.

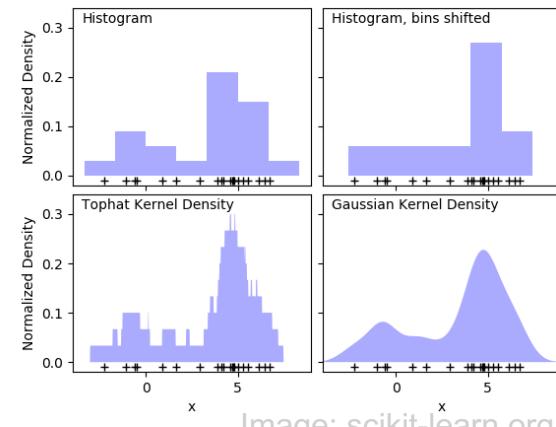


Image: scikit-learn.org





# Parametric Bayesian Inference

$\mathcal{M}$  is represented as a finite set of parameters  $\theta$

- ◆ A **parametric** likelihood:  $\mathbf{x} \sim p(\cdot|\theta)$
- ◆ Prior on  $\theta$ :  $\pi(\theta)$
- ◆ Posterior distribution

$$p(\theta|\mathbf{x}) = \frac{p(\mathbf{x}|\theta)\pi(\theta)}{\int p(\mathbf{x}|\theta)\pi(\theta)d\theta} \propto p(\mathbf{x}|\theta)\pi(\theta)$$

## Examples:

- Gaussian distribution prior + 2D Gaussian likelihood  $\rightarrow$  Gaussian posterior distribution
- Dirichilet distribution prior + 2D Multinomial likelihood  $\rightarrow$  Dirichilet posterior distribution
- Sparsity-inducing priors + some likelihood models  $\rightarrow$  Sparse Bayesian inference





# Nonparametric Bayesian Inference

$\mathcal{M}$  is a richer model, e.g., with an infinite set of parameters

- ◆ A nonparametric likelihood:  $\mathbf{x} \sim p(\cdot|\mathcal{M})$
- ◆ Prior on  $\mathcal{M}$ :  $\pi(\mathcal{M})$
- ◆ Posterior distribution

$$p(\mathcal{M}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{M})\pi(\mathcal{M})}{\int p(\mathbf{x}|\mathcal{M})\pi(\mathcal{M})d\mathcal{M}} \propto p(\mathbf{x}|\mathcal{M})\pi(\mathcal{M})$$

## Examples:

→ see next slide





# Nonparametric Bayesian Inference

probability measure



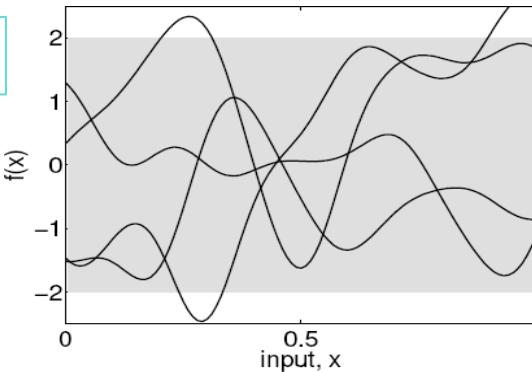
Dirichlet Process Prior [Antoniak, 1974]  
+ Multinomial/Gaussian/Softmax likelihood

binary matrix

|                | K |   |   |     |
|----------------|---|---|---|-----|
|                | 1 | 1 | 1 | ... |
| z <sub>1</sub> | 0 | 1 | 0 | ... |
| z <sub>2</sub> | 1 | 1 | 0 | ... |
| ⋮              | ⋮ | ⋮ | ⋮ | ⋮   |
| z <sub>n</sub> | 0 | 1 | 1 | ... |

Indian Buffet Process Prior [Griffiths & Gharamani, 2005]  
+ Gaussian/Sigmoid/Softmax likelihood

function



Gaussian Process Prior [Doob, 1944; Rasmussen & Williams, 2006]  
+ Gaussian/Sigmoid/Softmax likelihood

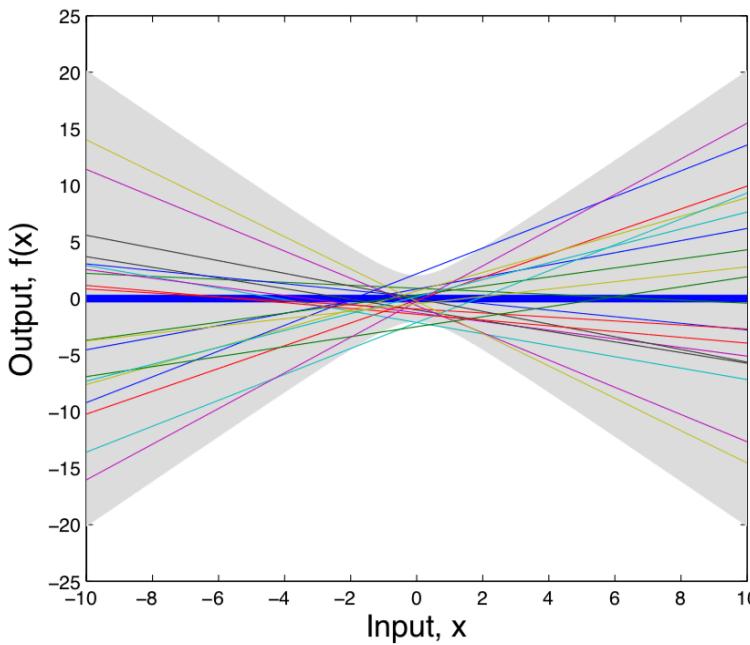




# Weight-space View

- Consider a simple linear model

$$f(x) = a_0 + a_1 x, \\ a_0, a_1 \sim \mathcal{N}(0, 1).$$





# Function-space View

- We are interested in the distribution over functions induced by the distribution over parameters...
- In fact, we can characterize the properties of these functions directly:

$$f(x|a_0, a_1) = a_0 + a_1 x, \quad a_0, a_1 \sim \mathcal{N}(0, 1).$$





# Function-space View

- We are interested in the distribution over functions induced by the distribution over parameters...
- In fact, we can characterize the properties of these functions directly:

$$f(x|a_0, a_1) = a_0 + a_1 x, \quad a_0, a_1 \sim \mathcal{N}(0, 1).$$

$$\mathbb{E}[f(x)] = \mathbb{E}[a_0] + \mathbb{E}[a_1]x = 0.$$





# Function-space View

- We are interested in the distribution over functions induced by the distribution over parameters...
- In fact, we can characterize the properties of these functions directly:

$$f(x|a_0, a_1) = a_0 + a_1 x, \quad a_0, a_1 \sim \mathcal{N}(0, 1).$$

$$\mathbb{E}[f(x)] = \mathbb{E}[a_0] + \mathbb{E}[a_1]x = 0.$$

$$\text{cov}[f(x_b), f(x_c)] = \mathbb{E}[f(x_b)f(x_c)] - \mathbb{E}[f(x_b)]\mathbb{E}[f(x_c)]$$





# Function-space View

- We are interested in the distribution over functions induced by the distribution over parameters...
- In fact, we can characterize the properties of these functions directly:

$$f(x|a_0, a_1) = a_0 + a_1 x, \quad a_0, a_1 \sim \mathcal{N}(0, 1).$$

$$\mathbb{E}[f(x)] = \mathbb{E}[a_0] + \mathbb{E}[a_1]x = 0.$$

$$\begin{aligned}\text{cov}[f(x_b), f(x_c)] &= \mathbb{E}[f(x_b)f(x_c)] - \mathbb{E}[f(x_b)]\mathbb{E}[f(x_c)] \\ &= \mathbb{E}[a_0^2 + a_0 a_1 (x_b + x_c) + a_1^2 x_b x_c] - 0 \\ &= \mathbb{E}[a_0^2] + \mathbb{E}[a_1^2 x_b x_c] + \mathbb{E}[a_0 a_1 (x_b + x_c)] \\ &= 1 + x_b x_c + 0 \\ &= 1 + x_b x_c.\end{aligned}$$





# Function-space View

- Therefore any collection of values has a joint Gaussian distribution:

$$[f(x_1), \dots, f(x_N)] \sim \mathcal{N}(0, K),$$
$$K_{ij} = \text{cov}(f(x_i), f(x_j)) = k(x_i, x_j) = 1 + x_b x_c.$$

- Definition: A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. We write  $f(x) \sim \mathcal{GP}(m, k)$  to mean

$$[f(x_1), \dots, f(x_N)] \sim \mathcal{N}(\boldsymbol{\mu}, K)$$
$$\boldsymbol{\mu}_i = m(x_i)$$
$$K_{ij} = k(x_i, x_j),$$

for any collection of input values  $x_1, \dots, x_N$ . In other words,  $f$  is a GP with mean function  $m(x)$  and covariance kernel  $k(x_i, x_j)$ .





# Example: Linear Basis Function Models

- Model specification:

$$f(x, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x)$$

$$p(\mathbf{w}) = \mathcal{N}(0, \Sigma_w)$$

- Moments of the induced distribution over functions:

$$\mathbb{E}[f(x, \mathbf{w})] = m(x) = \mathbb{E}[\mathbf{w}^T] \boldsymbol{\phi}(x) = 0$$

$$\begin{aligned}\text{cov}(f(x_i), f(x_j)) &= k(x_i, x_j) = \mathbb{E}[f(x_i)f(x_j)] - \mathbb{E}[f(x_i)]\mathbb{E}[f(x_j)] \\ &= \boldsymbol{\phi}(x_i)^T \mathbb{E}[\mathbf{w} \mathbf{w}^T] \boldsymbol{\phi}(x_j) - 0 \\ &= \boldsymbol{\phi}(x_i)^T \Sigma_w \boldsymbol{\phi}(x_j)\end{aligned}$$

- ▶  $f(x, \mathbf{w})$  is a Gaussian process,  $f(x) \sim \mathcal{N}(m, k)$  with mean function  $m(x) = 0$  and covariance kernel  $k(x_i, x_j) = \boldsymbol{\phi}(x_i)^T \Sigma_w \boldsymbol{\phi}(x_j)$ .





# Gaussian Processes

## Interpretability:

- We are ultimately more interested in – and have stronger intuitions about – the *functions* that model our data than weights  $w$  in a parametric model. We can express these intuitions using a covariance kernel.

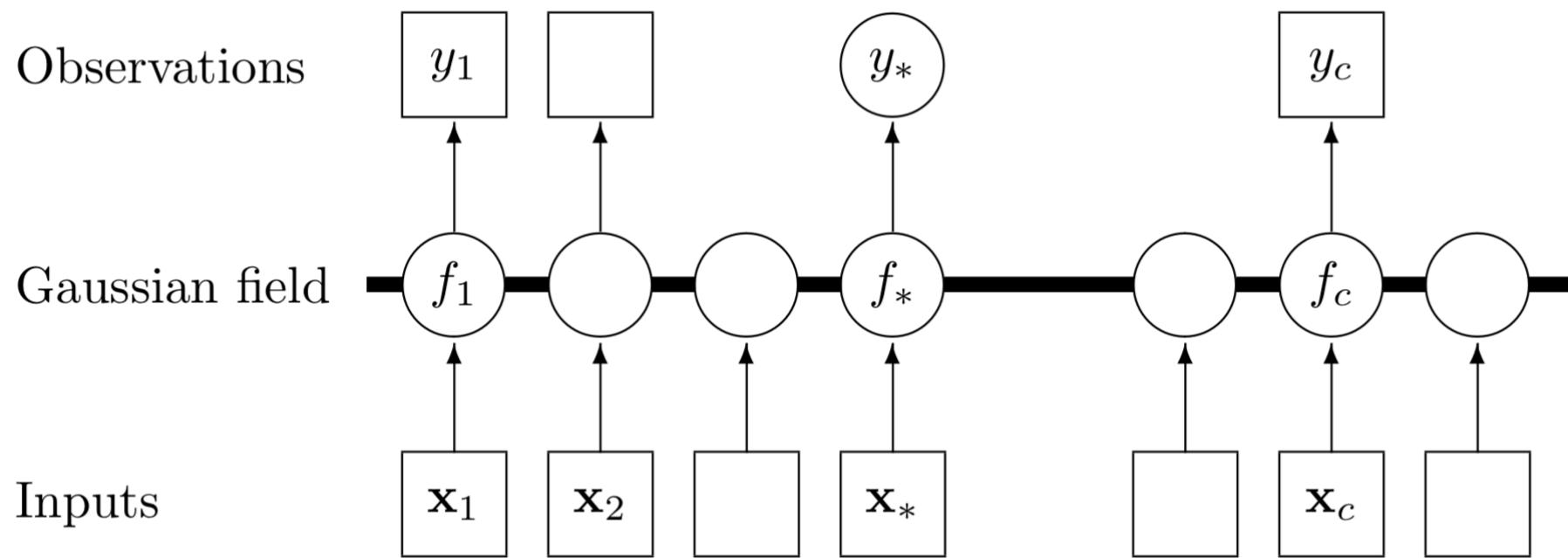
## Generalization:

- The kernel controls the support and inductive biases of our model, and thus its ability to generalize to unseen.





# Gaussian Process: Graphical Model





# Example: RBF kernel

$$k_{\text{RBF}}(x, x') = \text{cov}(f(x), f(x')) = a^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right)$$

- ▶ Far and above the most popular kernel.
- ▶ Expresses the intuition that function values at nearby inputs are more correlated than function values at far away inputs.
- ▶ The kernel *hyperparameters*  $a$  and  $\ell$  control amplitudes and wiggliness of these functions.
- ▶ GPs with an RBF kernel have large support and are *universal approximators*.





# Example: RBF kernel

$$k_{\text{RBF}}(x, x') = \text{cov}(f(x), f(x')) = a^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right)$$

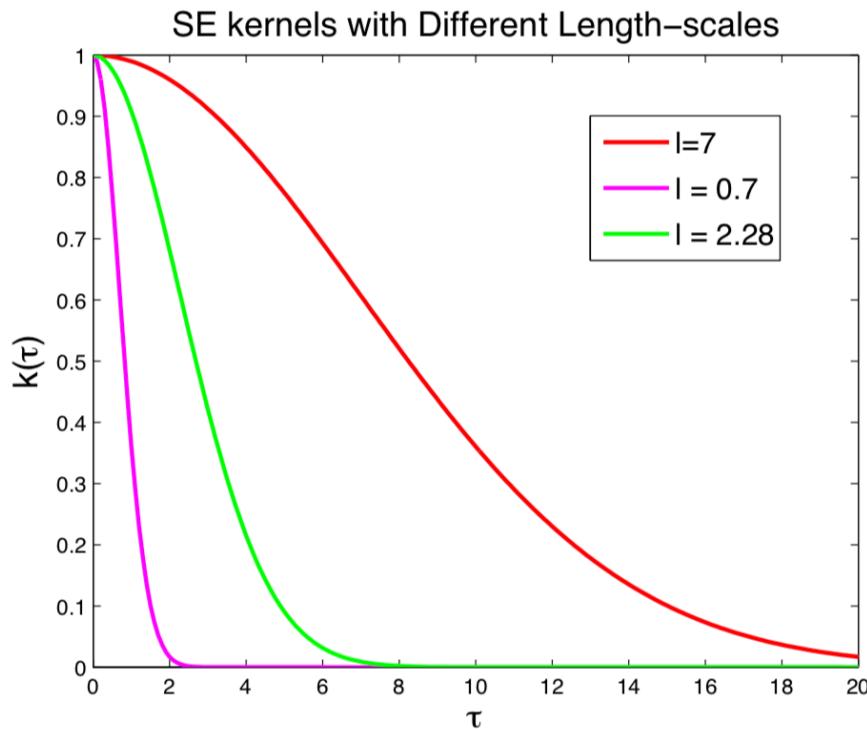
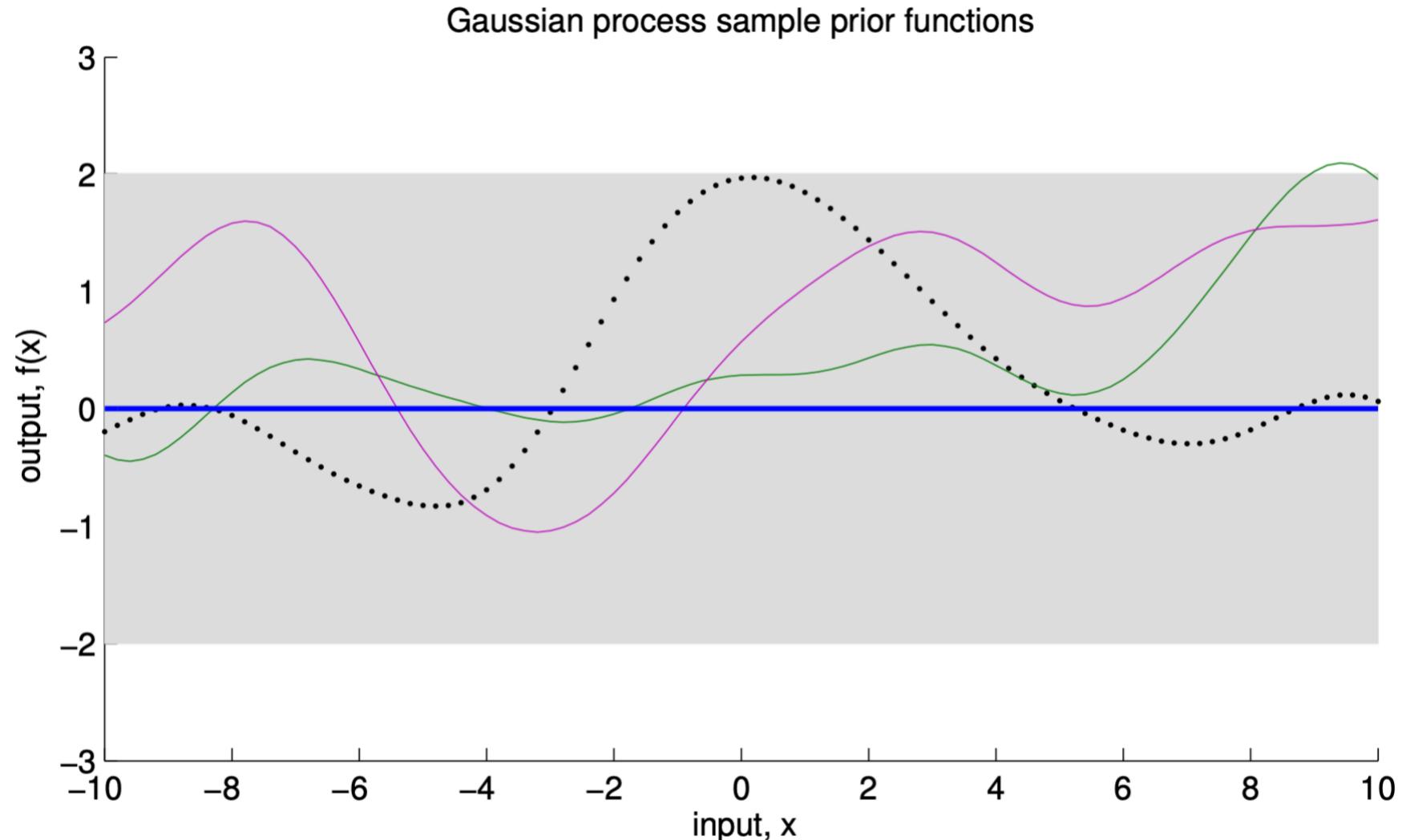


Figure: SE kernels with different length-scales, as a function of  $\tau = x - x'$ .





# Example: RBF kernel





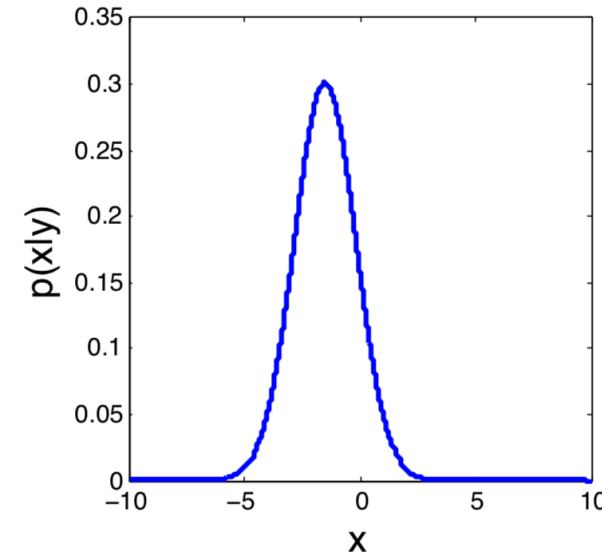
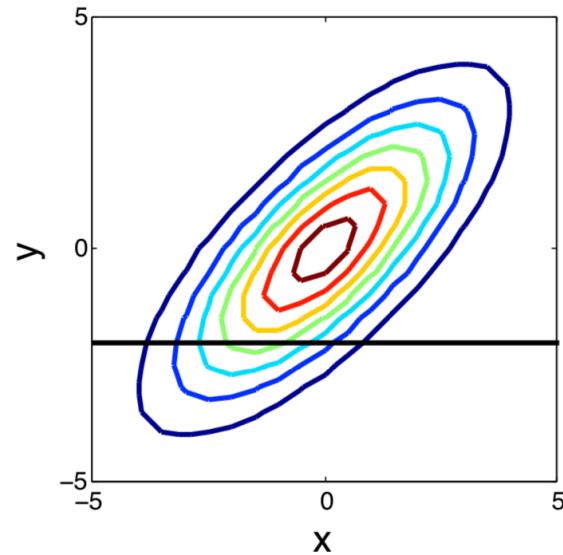
# Gaussian Process Inference

- ▶ Observed noisy data  $\mathbf{y} = (y(x_1), \dots, y(x_N))^T$  at input locations  $X$ .
- ▶ Start with the standard regression assumption:  $\mathcal{N}(y(x); f(x), \sigma^2)$ .
- ▶ Place a Gaussian process distribution over noise free functions  $f(x) \sim \mathcal{GP}(0, k_\theta)$ . The kernel  $k$  is parametrized by  $\theta$ .
- ▶ Infer  $p(f_* | \mathbf{y}, X, X_*)$  for the noise free function  $f$  evaluated at test points  $X_*$ .





# Recap: Multivariate Gaussian Distribution



If

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} k_{1,1} & k_{1,2} \\ k_{1,2}^T & k_{2,2} \end{bmatrix} \right)$$

then

$$y_1 | y_2 \sim \mathcal{N} \left( \mu_1 + k_{1,2} k_{2,2}^{-1} (y_2 - \mu_2), k_{1,1} - k_{1,2} k_{2,2}^{-1} k_{1,2}^T \right)$$





# Gaussian Process Inference

- ▶ Observed noisy data  $\mathbf{y} = (y(x_1), \dots, y(x_N))^T$  at input locations  $X$ .
- ▶ Start with the standard regression assumption:  $\mathcal{N}(y(x); f(x), \sigma^2)$ .
- ▶ Place a Gaussian process distribution over noise free functions  $f(x) \sim \mathcal{GP}(0, k_\theta)$ . The kernel  $k$  is parametrized by  $\theta$ .
- ▶ Infer  $p(f_* | \mathbf{y}, X, X_*)$  for the noise free function  $f$  evaluated at test points  $X_*$ .

## Joint distribution

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_\theta(X, X) + \sigma^2 I & K_\theta(X, X_*) \\ K_\theta(X_*, X) & K_\theta(X_*, X_*) \end{bmatrix}\right).$$





# Gaussian Process Inference

- ▶ Observed noisy data  $\mathbf{y} = (y(x_1), \dots, y(x_N))^T$  at input locations  $X$ .
- ▶ Start with the standard regression assumption:  $\mathcal{N}(y(x); f(x), \sigma^2)$ .
- ▶ Place a Gaussian process distribution over noise free functions  $f(x) \sim \mathcal{GP}(0, k_\theta)$ . The kernel  $k$  is parametrized by  $\theta$ .
- ▶ Infer  $p(f_* | \mathbf{y}, X, X_*)$  for the noise free function  $f$  evaluated at test points  $X_*$ .

## Joint distribution

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_\theta(X, X) + \sigma^2 I & K_\theta(X, X_*) \\ K_\theta(X_*, X) & K_\theta(X_*, X_*) \end{bmatrix}\right).$$

## Conditional predictive distribution

$$f_* | X_*, X, \mathbf{y}, \boldsymbol{\theta} \sim \mathcal{N}(\bar{f}_*, \text{cov}(f_*)) ,$$

$$\bar{f}_* = K_\theta(X_*, X)[K_\theta(X, X) + \sigma^2 I]^{-1} \mathbf{y} ,$$

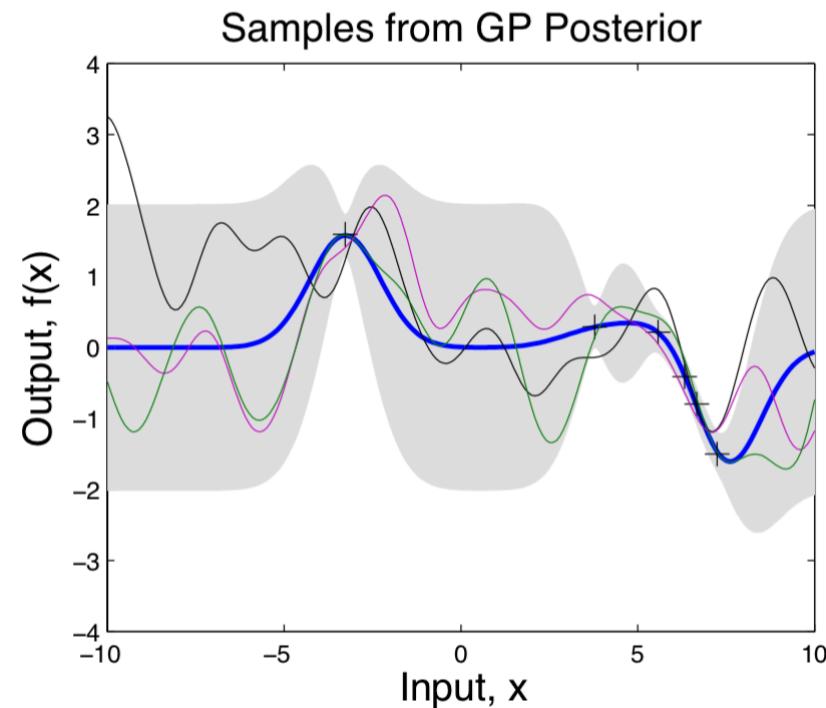
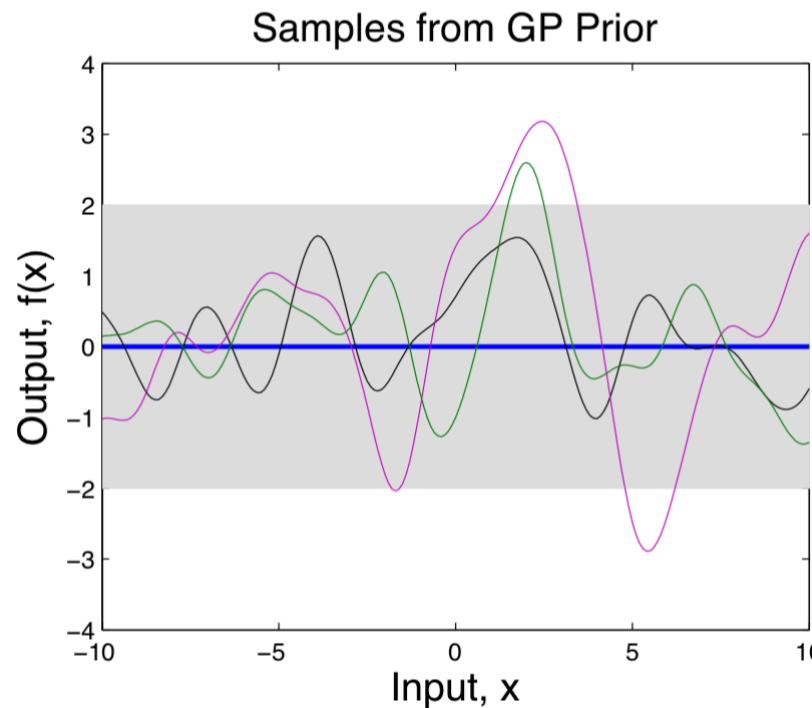
$$\text{cov}(f_*) = K_\theta(X_*, X_*) - K_\theta(X_*, X)[K_\theta(X, X) + \sigma^2 I]^{-1} K_\theta(X, X_*) .$$





# Gaussian Process Inference

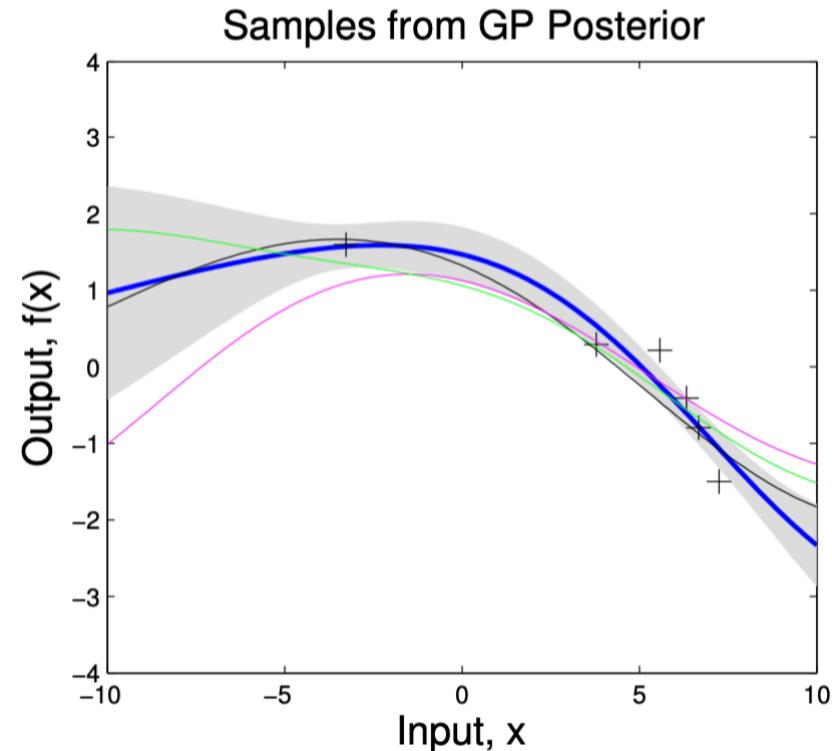
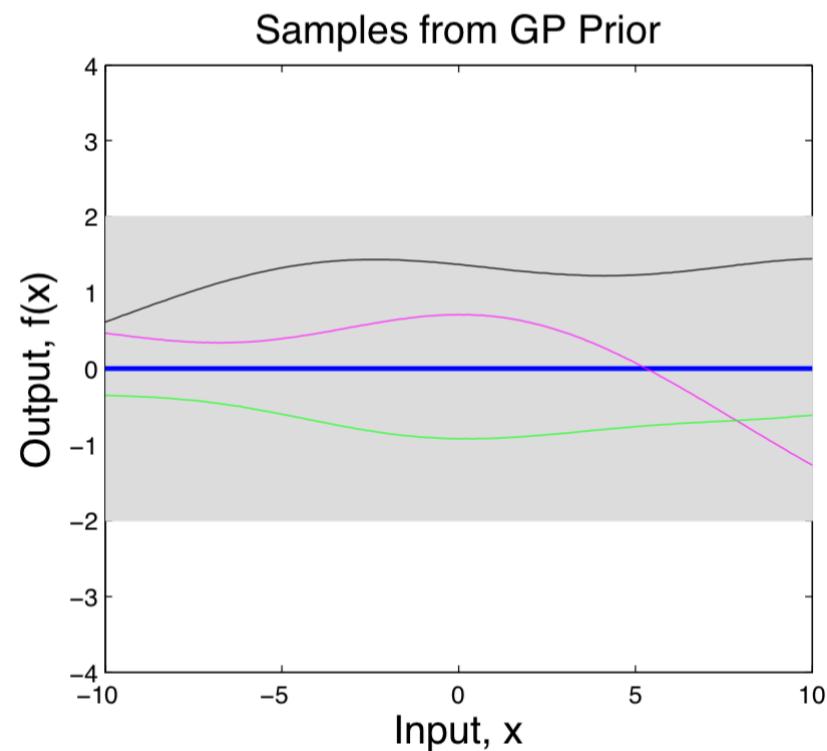
- ▶ Specify  $f(x) \sim \mathcal{GP}(0, k)$ .
- ▶ Choose  $k_{\text{RBF}}(x, x') = a_0^2 \exp(-\frac{\|x-x'\|^2}{2\ell_0^2})$ . Choose values for  $a_0$  and  $\ell_0$ .
- ▶ Observe data, look at the prior and posterior over functions.





# Gaussian Process Inference

Increase the length-scale  $\ell$ .



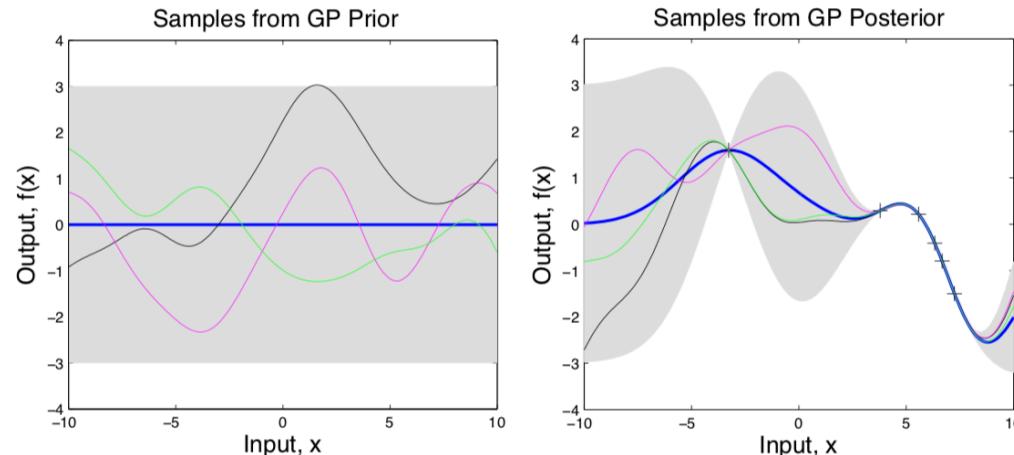


# Gaussian Process Learning

- We can integrate away the entire Gaussian process  $f(x)$  to obtain the marginal likelihood, as a function of kernel hyperparameters  $\theta$  alone.

$$p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X}) = \int p(\mathbf{y}|f, \mathbf{X})p(f|\boldsymbol{\theta}, \mathbf{X})df. \quad (48)$$

$$\log p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X}) = \underbrace{-\frac{1}{2}\mathbf{y}^T(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1}\mathbf{y}}_{\text{model fit}} - \underbrace{\frac{1}{2} \log |K_{\boldsymbol{\theta}} + \sigma^2 I| - \frac{N}{2} \log(2\pi)}_{\text{complexity penalty}}. \quad (49)$$





# Gaussian Process Learning

1. Learning: Optimize marginal likelihood,

$$\log p(\mathbf{y}|\boldsymbol{\theta}, X) = \underbrace{-\frac{1}{2}\mathbf{y}^T(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1}\mathbf{y}}_{\text{model fit}} - \underbrace{\frac{1}{2}\log|K_{\boldsymbol{\theta}} + \sigma^2 I|}_{\text{complexity penalty}} - \frac{N}{2}\log(2\pi),$$

with respect to kernel hyperparameters  $\boldsymbol{\theta}$ .

2. Inference: Conditioned on kernel hyperparameters  $\boldsymbol{\theta}$ , form the predictive distribution for test inputs  $X_*$ :

$$f_* | X_*, X, \mathbf{y}, \boldsymbol{\theta} \sim \mathcal{N}(\bar{f}_*, \text{cov}(f_*)) ,$$

$$\bar{f}_* = K_{\boldsymbol{\theta}}(X_*, X)[K_{\boldsymbol{\theta}}(X, X) + \sigma^2 I]^{-1}\mathbf{y} ,$$

$$\text{cov}(f_*) = K_{\boldsymbol{\theta}}(X_*, X_*) - K_{\boldsymbol{\theta}}(X_*, X)[K_{\boldsymbol{\theta}}(X, X) + \sigma^2 I]^{-1}K_{\boldsymbol{\theta}}(X, X_*) .$$





# Rich Literature on Other Types of Covariance Kernels

Kernels as functions of the distance:

$$k_{\text{SE}}(\tau) = \exp(-0.5\tau^2/\ell^2)$$

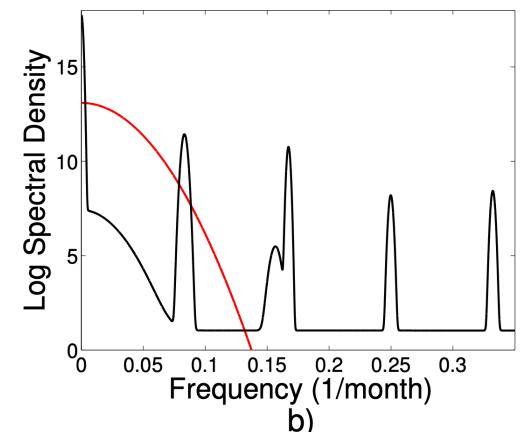
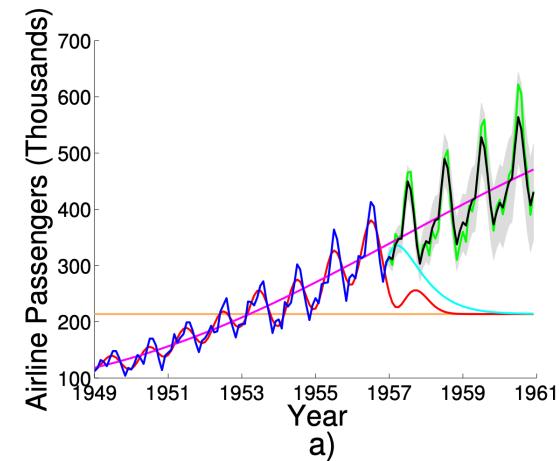
$$k_{\text{MA}}(\tau) = a(1 + \frac{\sqrt{3}\tau}{\ell}) \exp(-\frac{\sqrt{3}\tau}{\ell})$$

$$k_{\text{RQ}}(\tau) = (1 + \frac{\tau^2}{2\alpha\ell^2})^{-\alpha}$$

$$k_{\text{PE}}(\tau) = \exp(-2 \sin^2(\pi \tau \omega)/\ell^2)$$

Spectral mixture kernels (Wilson & Adams, 2013)

$$k(\tau) = \sum_{q=1}^Q w_q \prod_{p=1}^P \exp\{-2\pi^2\tau_p^2 v_q^{(p)}\} \cos(2\pi\tau_p \mu_q^{(p)})$$



Other kernels: on strings/sequences (Lodhi et al., 2002), Fisher kernels (Jaakkola et al., 2000), etc.

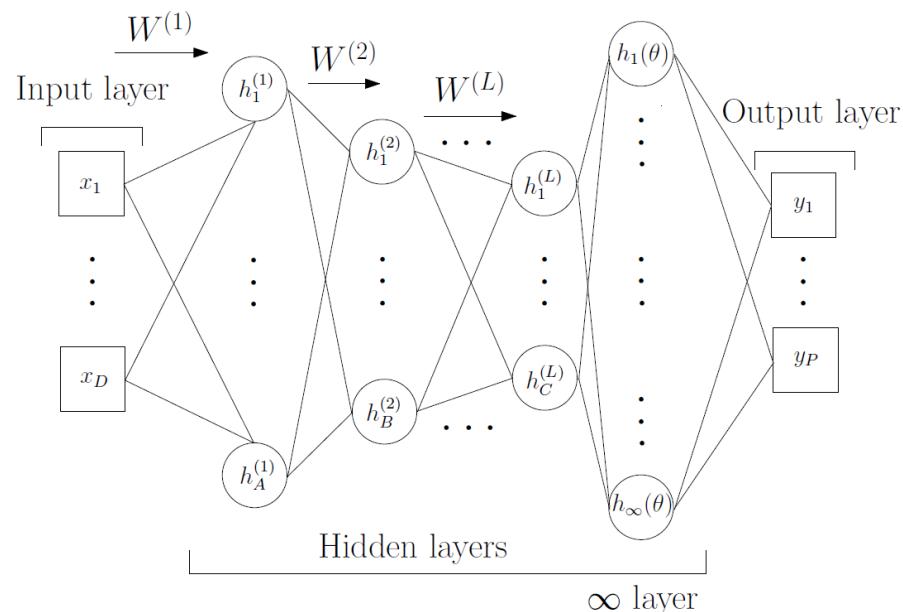




# Deep Kernel Learning

- Combines inductive biases of deep learning architectures with the nonparametric flexibility of Gaussian processes.
- Starting from some base kernel, we can get a deep kernel using functional composition:

$$\kappa(x, x') = k(h(x), h(x'))$$





# Learning Deep Kernels

- Learn base kernel hyperparameters and neural network parameters jointly.
- Use the chain rule to compute derivatives of the log marginal likelihood w.r.t. the deep kernel hyperparameters:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial K_\gamma} \frac{\partial K_\gamma}{\partial \boldsymbol{\theta}}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial K_\gamma} \frac{\partial K_\gamma}{\partial g(\mathbf{x}, \mathbf{w})} \frac{\partial g(\mathbf{x}, \mathbf{w})}{\partial \mathbf{w}}$$

- To make the model scalable, inducing point methods can be applied.





# Deep Kernel Learning for Regression

| Datasets   | n         | d   | RMSE        |                  |                  |                  |                    |                    |
|------------|-----------|-----|-------------|------------------|------------------|------------------|--------------------|--------------------|
|            |           |     | GP          |                  |                  | DNN              | DKL                |                    |
|            |           |     | RBF         | SM               | best             |                  | RBF                | SM                 |
| Gas        | 2,565     | 128 | 0.21±0.07   | 0.14±0.08        | 0.12±0.07        | 0.11±0.05        | 0.11±0.05          | <b>0.09±0.06</b>   |
| Skillcraft | 3,338     | 19  | 1.26±3.14   | 0.25±0.02        | 0.25±0.02        | <b>0.25±0.00</b> | <b>0.25±0.00</b>   | <b>0.25±0.00</b>   |
| SML        | 4,137     | 26  | 6.94±0.51   | 0.27±0.03        | 0.26±0.04        | 0.25±0.02        | 0.24±0.01          | <b>0.23±0.01</b>   |
| Parkinsons | 5,875     | 20  | 3.94±1.31   | <b>0.00±0.00</b> | <b>0.00±0.00</b> | 0.31±0.04        | 0.29±0.04          | 0.29±0.04          |
| Pumadyn    | 8,192     | 32  | 1.00±0.00   | 0.21±0.00        | <b>0.20±0.00</b> | 0.25±0.02        | 0.24±0.02          | 0.23±0.02          |
| PoleTele   | 15,000    | 26  | 12.6±0.3    | 5.40±0.3         | 4.30±0.2         | 3.42±0.05        | 3.28±0.04          | <b>3.11±0.07</b>   |
| Elevators  | 16,599    | 18  | 0.12±0.00   | 0.090±0.001      | 0.089±0.002      | 0.099±0.001      | <b>0.084±0.002</b> | <b>0.084±0.002</b> |
| Kin40k     | 40,000    | 8   | 0.34±0.01   | 0.19±0.02        | 0.06±0.00        | 0.11±0.01        | 0.05±0.00          | <b>0.03±0.01</b>   |
| Protein    | 45,730    | 9   | 1.64±1.66   | 0.50±0.02        | 0.47±0.01        | 0.49±0.01        | 0.46±0.01          | <b>0.43±0.01</b>   |
| KEGG       | 48,827    | 22  | 0.33±0.17   | 0.12±0.01        | 0.12±0.01        | 0.12±0.01        | 0.11±0.00          | <b>0.10±0.01</b>   |
| CTslice    | 53,500    | 385 | 7.13±0.11   | 2.21±0.06        | 0.59±0.07        | 0.41±0.06        | 0.36±0.01          | <b>0.34±0.02</b>   |
| KEGGU      | 63,608    | 27  | 0.29±0.12   | 0.12±0.00        | 0.12±0.00        | 0.12±0.00        | <b>0.11±0.00</b>   | <b>0.11±0.00</b>   |
| 3Droad     | 434,874   | 3   | 12.86±0.09  | 10.34±0.19       | 9.90±0.10        | 7.36±0.07        | <b>6.91±0.04</b>   | <b>6.91±0.04</b>   |
| Song       | 515,345   | 90  | 0.55±0.00   | 0.46±0.00        | 0.45±0.00        | 0.45±0.02        | 0.44±0.00          | <b>0.43±0.01</b>   |
| Buzz       | 583,250   | 77  | 0.88±0.01   | 0.51±0.01        | 0.51±0.01        | 0.49±0.00        | 0.48±0.00          | <b>0.46±0.01</b>   |
| Electric   | 2,049,280 | 11  | 0.230±0.000 | 0.053±0.000      | 0.053±0.000      | 0.058±0.002      | 0.050±0.002        | <b>0.048±0.002</b> |

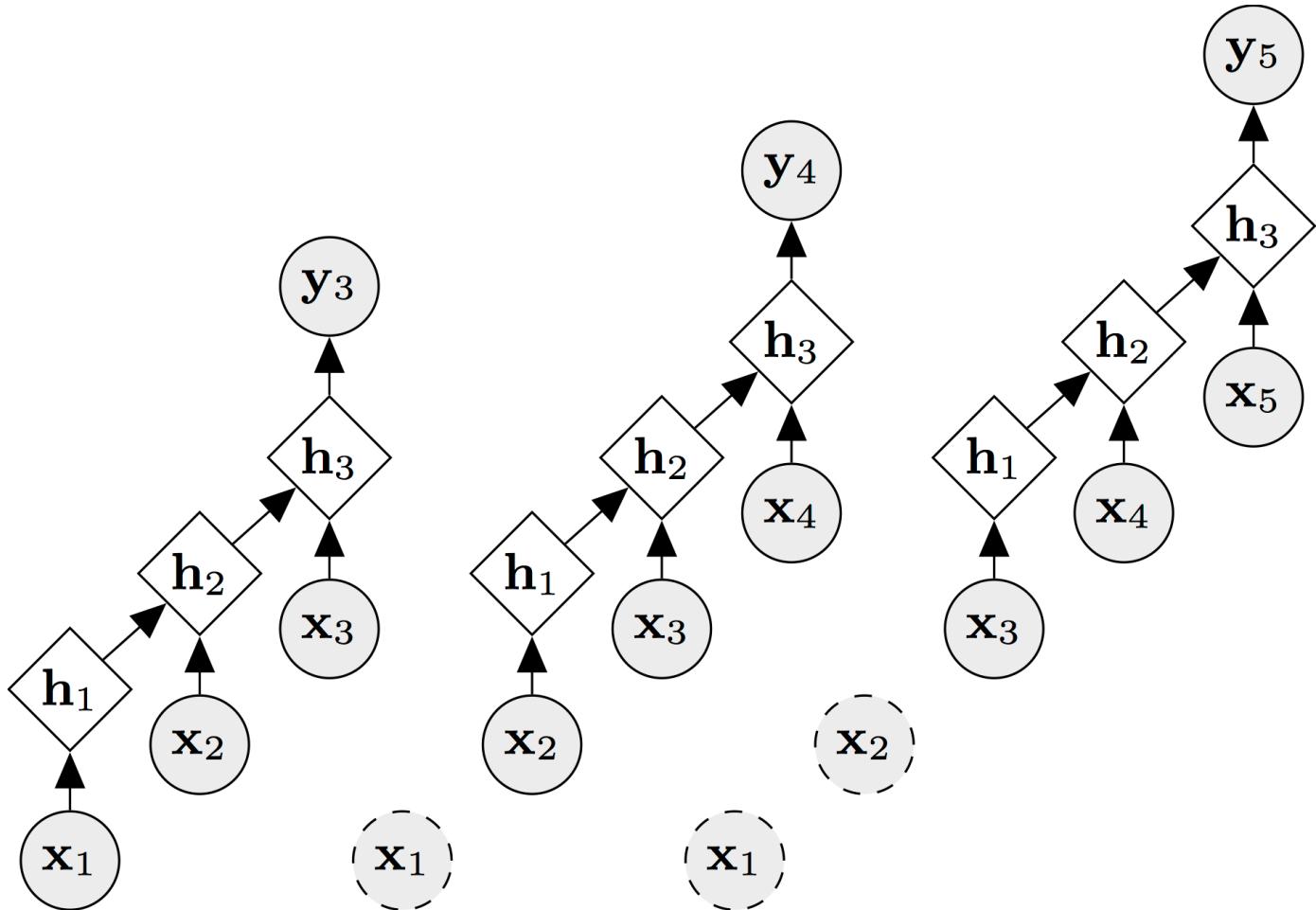




# Deep Kernel Learning on Sequential Data

What if we have data of sequential nature?

Can we still apply the same reasoning and build rich nonparametric models on top recurrent nets?

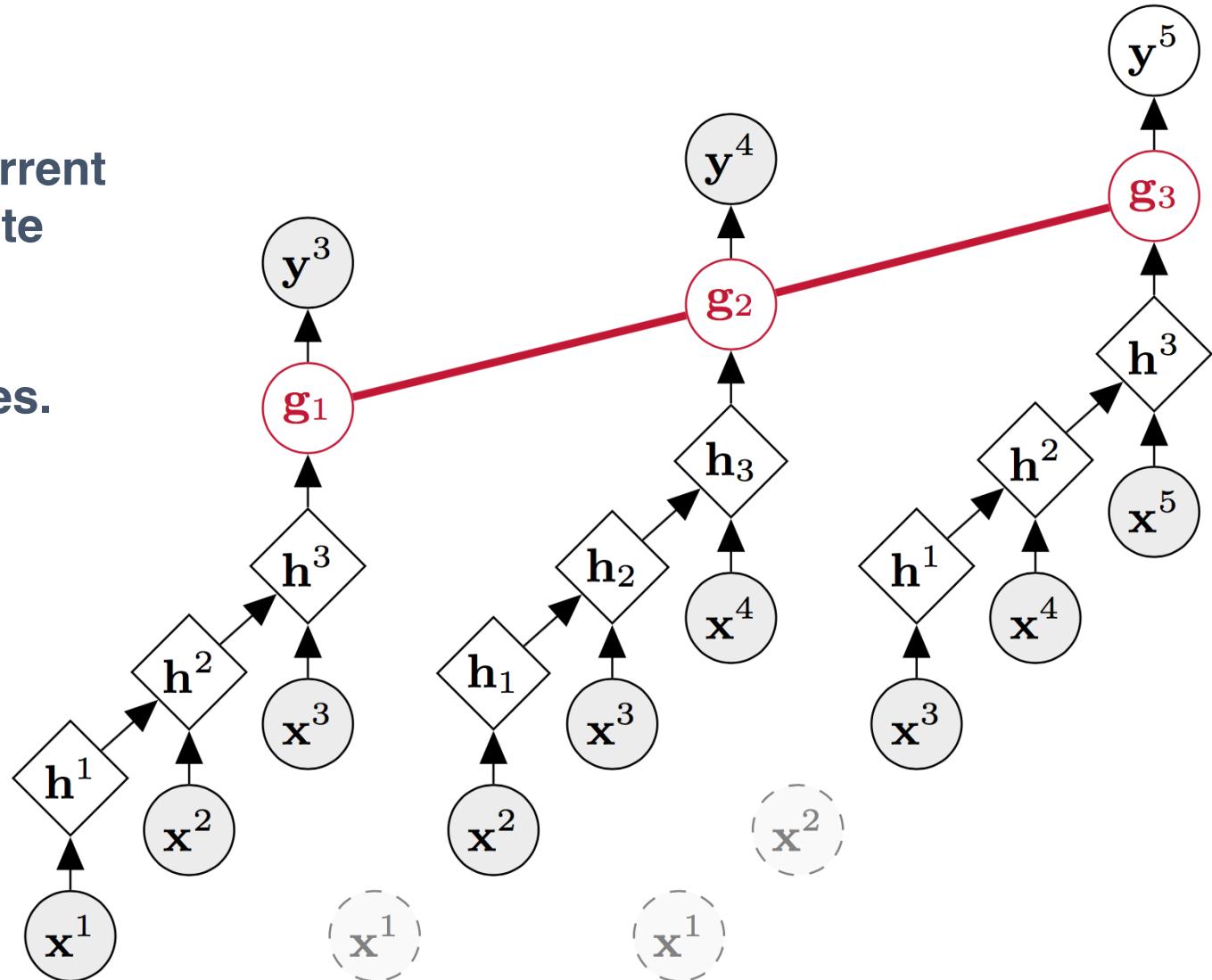




# Deep Kernel Learning on Sequential Data

The answer is YES!

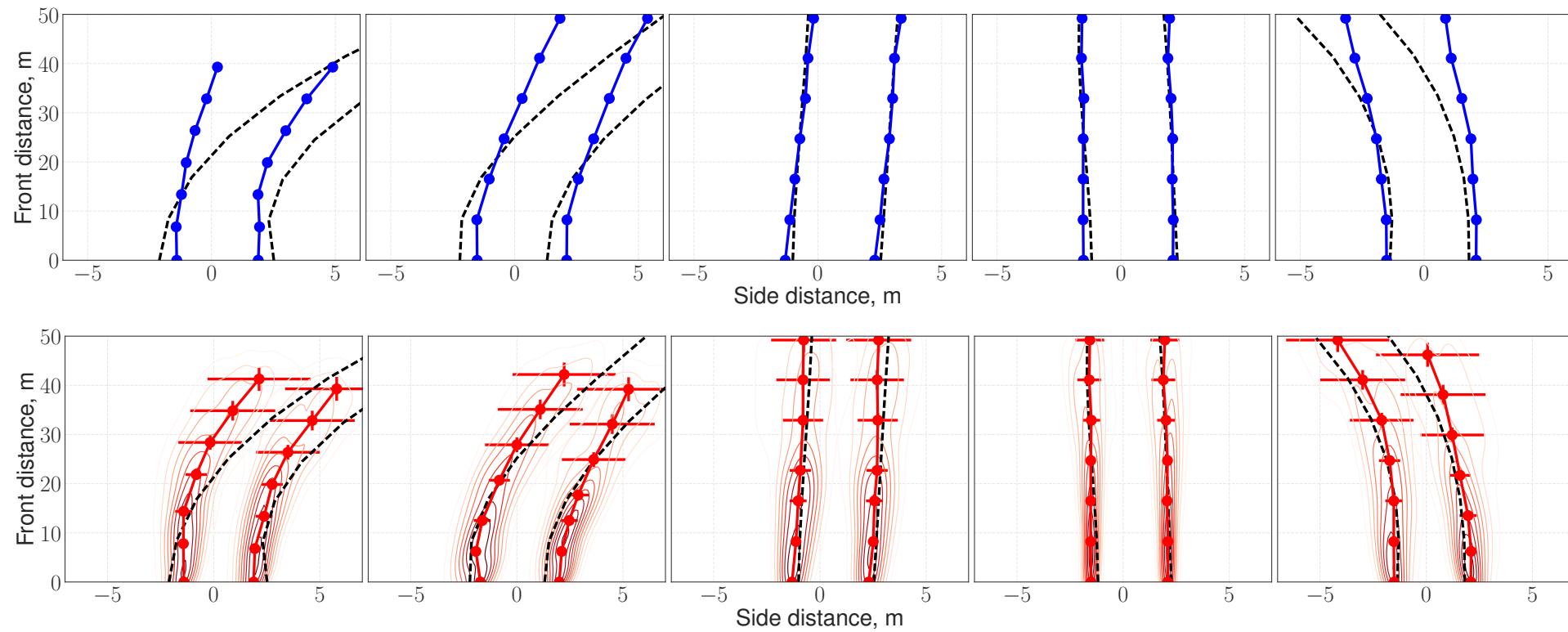
By adding a GP layer to a recurrent network, we effectively correlate samples across time and get predictions along with well calibrated uncertainty estimates.





# Deep Kernel Learning on Sequential Data

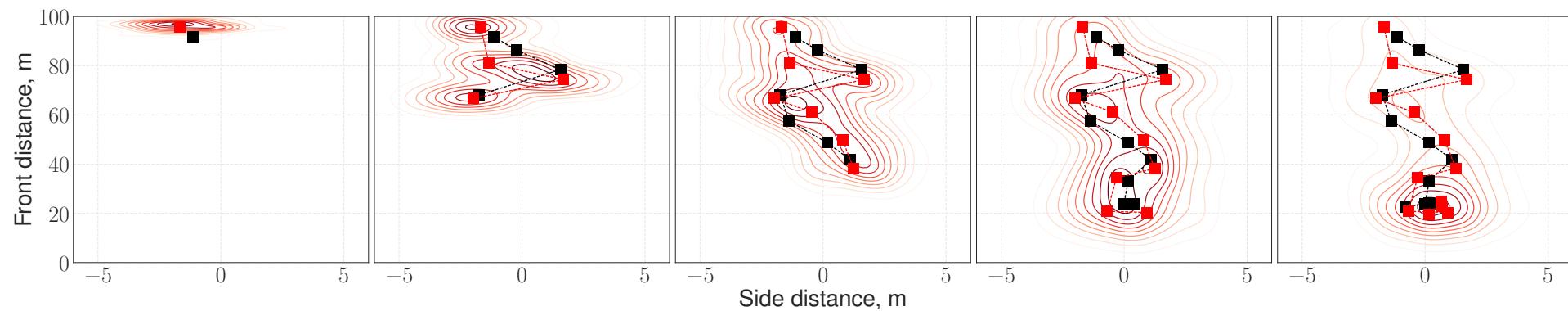
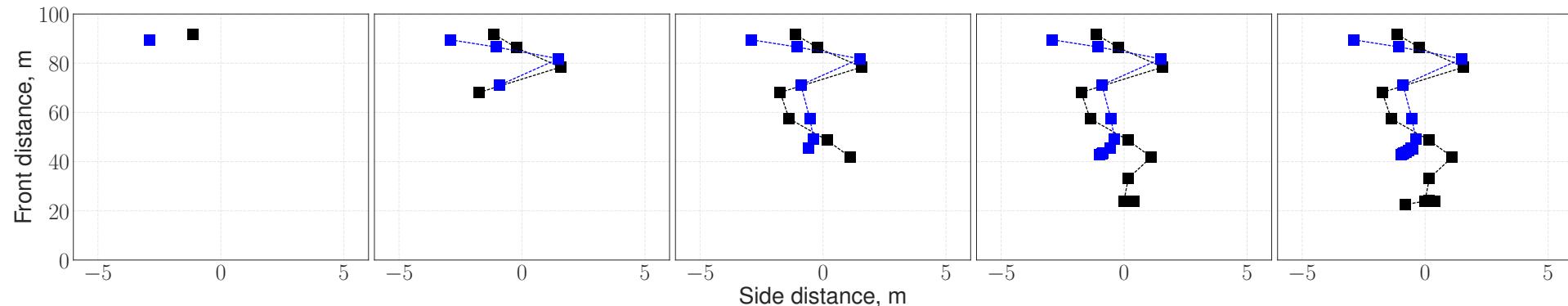
Lane prediction: LSTM vs GP-LSTM





# Deep Kernel Learning on Sequential Data

Lead vehicle prediction: LSTM vs GP-LSTM





# The Scalability Issue

- ▶ Computational bottlenecks for GPs:
  - ▶ Inference:  $(K_\theta + \sigma^2 I)^{-1} \mathbf{y}$  for  $n \times n$  matrix  $K$ .
  - ▶ Learning:  $\log |K_\theta + \sigma^2 I|$ , for marginal likelihood evaluations needed to learn  $\theta$ .
- ▶ Both inference and learning naively require  $\mathcal{O}(n^3)$  operations and  $\mathcal{O}(n^2)$  storage (typically from computing a Cholesky decomposition of  $K$ ). Afterwards, the predictive mean and variance cost  $\mathcal{O}(n)$  and  $\mathcal{O}(n^2)$  per test point.





# Scaling Up Gaussian Processes

## Three Families of Approaches

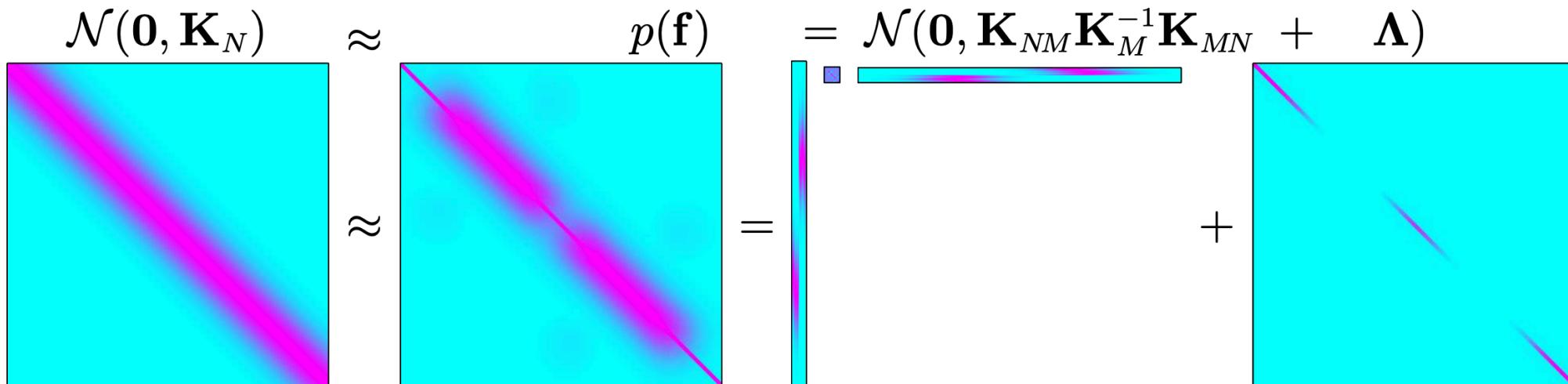
- ▶ Approximate non-parametric kernels in a finite basis ‘dual space’. Requires  $\mathcal{O}(m^2n)$  computations and  $\mathcal{O}(m)$  storage for  $m$  basis functions. Examples: SSGP, Random Kitchen Sinks, Fastfood, À la Carte.
- ▶ Inducing point based sparse approximations. Examples: SoR, FITC, KISS-GP.
- ▶ Exploit existing structure in  $K$  to quickly (and exactly) solve linear systems and log determinants. Examples: Toeplitz and Kronecker methods.





# Inducing Point Methods

We can approximate GP through  $M < N$  inducing points  $\bar{\mathbf{f}}$  to obtain this Sparse Pseudo-input Gaussian process (SPGP) prior:  $p(\mathbf{f}) = \int d\bar{\mathbf{f}} \prod_n p(f_n|\bar{\mathbf{f}}) p(\bar{\mathbf{f}})$

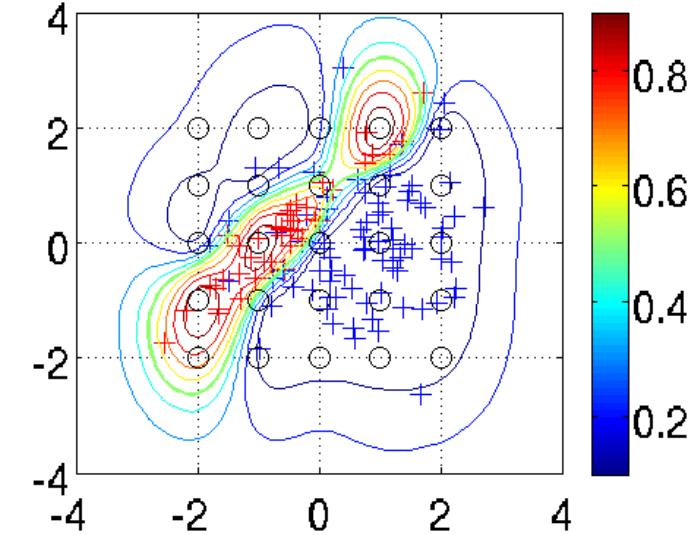
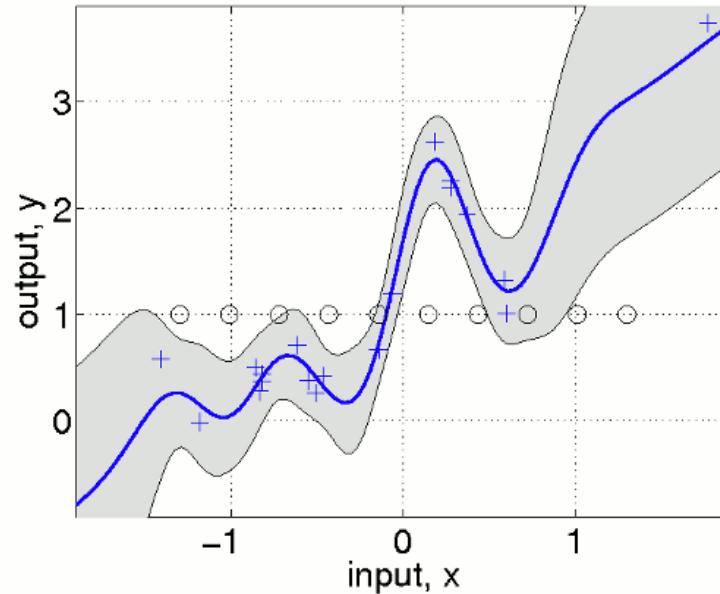


- SPGP covariance inverted in  $\mathcal{O}(M^2N) \ll \mathcal{O}(N^3) \Rightarrow$  much faster





# Inducing Point Methods



**Grids are tricky:**

In high dimensions, one would need a LOT of inducing points to build a high-dimensional grid.  
This might drastically affect efficiency.

**Further reading:**

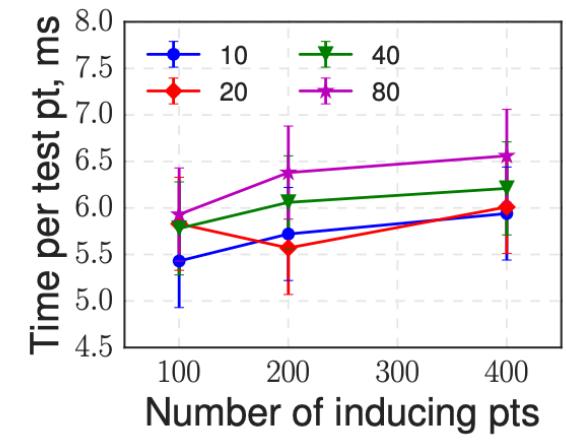
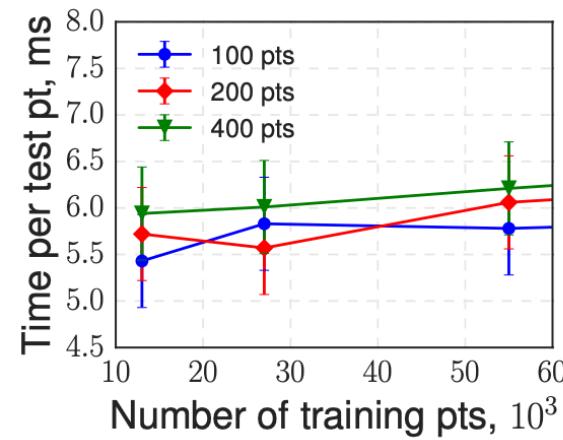
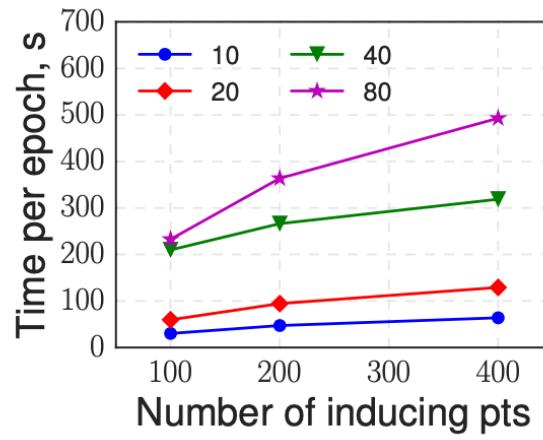
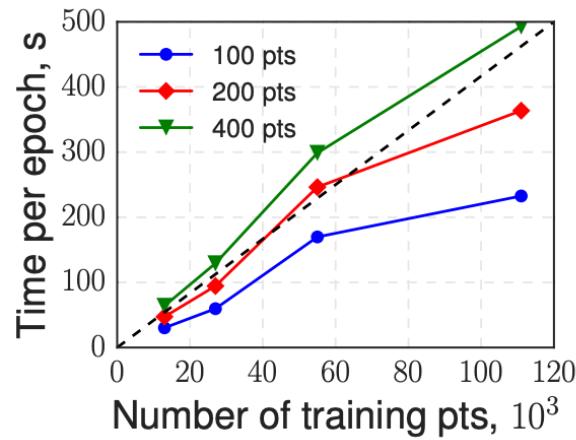
Wilson, Dann, Nickisch (2015). Thoughts on Massively Scalable Gaussian Processes

Bauer, van der Wilk, Rasmussen (2016). Understanding Probabilistic Sparse Gaussian Process Approximations.





# Massively Scalable GPs: $O(n)$ training, $O(1)$ inference

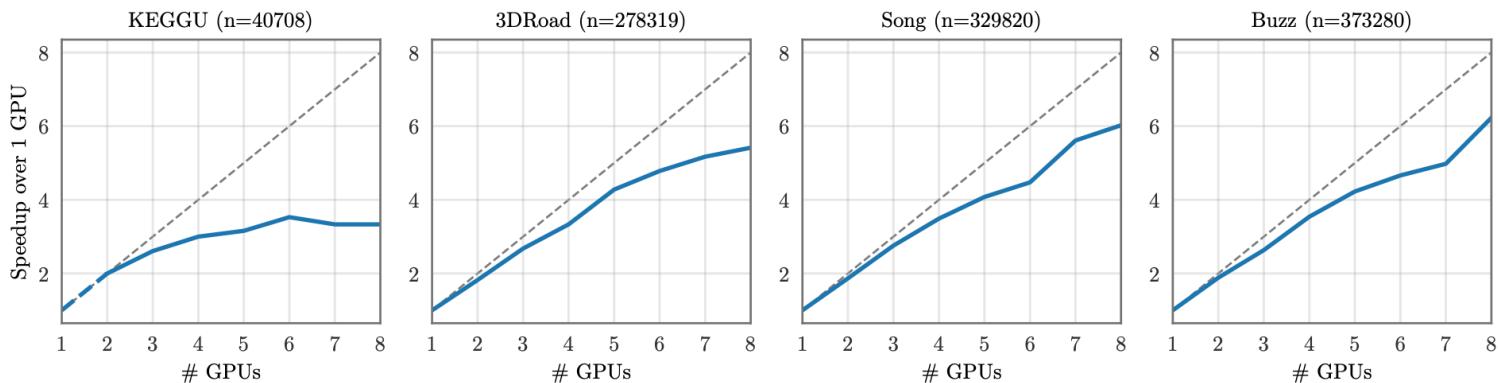




# Running Exact GPs on GPUs (recent)

**Key idea:** Use a clever distributed GP learning and inference algorithm that runs on multiple GPUs.

| Dataset       | n         | d   | RMSE (random = 1)  |                 |                   | Training Time      |                 |                   |      |     |
|---------------|-----------|-----|--------------------|-----------------|-------------------|--------------------|-----------------|-------------------|------|-----|
|               |           |     | Exact GP<br>(BBMM) | SGPR<br>(m=512) | SVGP<br>(m=1,024) | Exact GP<br>(BBMM) | SGPR<br>(m=512) | SVGP<br>(m=1,024) | #GPU | p   |
| PoleTele      | 9,600     | 26  | <b>0.154</b>       | 0.219           | 0.218             | <b>22.1 s</b>      | 40.6 s          | 68.1 s            | 1    | 1   |
| Elevators     | 10,623    | 18  | <b>0.374</b>       | 0.436           | 0.386             | <b>17.1 s</b>      | 41.2 s          | 112 s             | 1    | 1   |
| Bike          | 11,122    | 17  | <b>0.216</b>       | 0.345           | 0.261             | <b>18.8 s</b>      | 41.0 s          | 109 s             | 1    | 1   |
| Kin40K        | 25,600    | 8   | <b>0.093</b>       | 0.257           | 0.177             | 83.3 s             | <b>56.1 s</b>   | 297 s             | 1    | 1   |
| Protein       | 29,267    | 9   | <b>0.545</b>       | 0.659           | 0.640             | 120 s              | <b>65.5 s</b>   | 300 s             | 1    | 1   |
| KeggDirected  | 31,248    | 20  | <b>0.078</b>       | 0.089           | 0.083             | 107 s              | <b>67.0 s</b>   | 345 s             | 1    | 1   |
| CTslice       | 34,240    | 385 | <b>0.050</b>       | 0.199           | 1.011             | 148 s              | <b>77.5 s</b>   | 137 s             | 1    | 1   |
| KEGGU         | 40,708    | 27  | <b>0.120</b>       | 0.133           | 0.123             | <b>50.8 s</b>      | 84.9 s          | 7.61 min          | 8    | 1   |
| 3DRoad        | 278,319   | 3   | <b>0.106</b>       | 0.654           | 0.475             | 7.06 hr            | <b>8.53 min</b> | 22.1 min          | 8    | 16  |
| Song          | 329,820   | 90  | <b>0.761</b>       | 0.803           | 0.999             | 6.63 hr            | <b>9.38 min</b> | 18.5 min          | 8    | 16  |
| Buzz          | 373,280   | 77  | <b>0.265</b>       | 0.387           | 0.270             | 11.5 hr            | <b>11.5 min</b> | 1.19 hr           | 8    | 19  |
| HouseElectric | 1,311,539 | 9   | <b>0.049</b>       | —               | 0.086             | 3.29 days          | —               | <b>4.22 hr</b>    | 8    | 218 |





# Gaussian Process Software

## 1) Classic MATLAB-based:

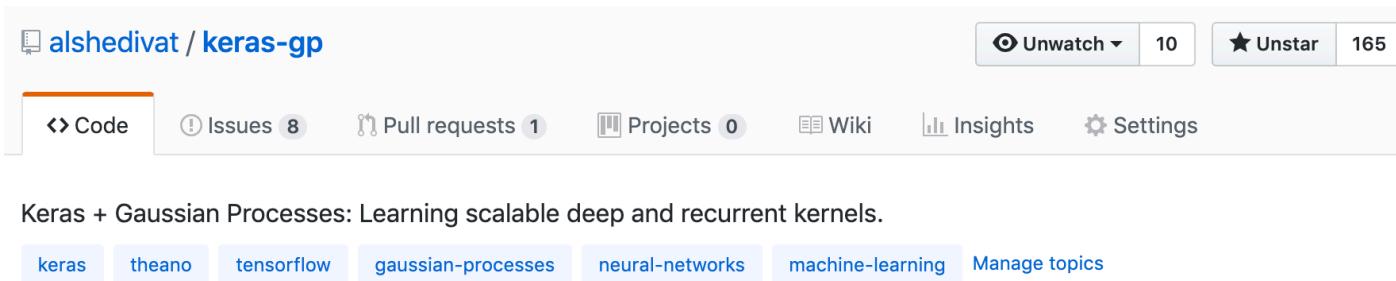
### Documentation for GPML Matlab Code version 4.2

#### 1) What?

The code provided here originally demonstrated the main algorithms from Rasmussen and Williams: [Gaussian Processes for Machine Learning](#). It has since grown to allow more likelihood functions, further inference methods and a flexible framework for specifying GPs. Other GP packages can be found [here](#).

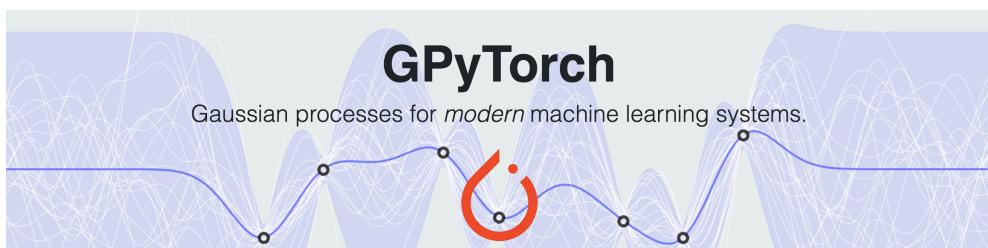
The code is written by Carl Edward Rasmussen and Hannes Nickisch; it runs on both [Octave](#) 3.2.x and [Matlab®](#) 7.x and later. The code is based on [previous versions](#) written by Carl Edward Rasmussen and Chris Williams.

## 2) Keras-based (GPs as DL layers!)



A screenshot of a GitHub repository page for 'alshedivat / keras-gp'. The page shows basic statistics: 10 stars and 165 forks. Navigation links include 'Code', 'Issues 8', 'Pull requests 1', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. Below the navigation bar, a brief description reads: 'Keras + Gaussian Processes: Learning scalable deep and recurrent kernels.' A topic bar at the bottom includes 'keras', 'theano', 'tensorflow', 'gaussian-processes', 'neural-networks', 'machine-learning', and 'Manage topics'.

## 3) PyTorch-based



Gardner et al. (2018) arXiv:1809.11165

## 4) TensorFlow (T2T library)

### 2 Gaussian Process Layers

GP layers map tensor to tensor and internally sample from the function belief.

```
batch_size = 256
features, labels = load_spatial_data(batch_size)

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(), # no spatial knowledge
    layers.SparseGaussianProcess(units=256, num_inducing=512),
    layers.SparseGaussianProcess(units=256, num_inducing=512),
    layers.SparseGaussianProcess(units=10, num_inducing=512),
])
predictions = model(features)
neg_log_likelihood = tf.losses.mean_squared_error(labels=labels,
    predictions=predictions)
k1 = sum(model.losses)
loss = neg_log_likelihood + k1
train_op = tf.train.AdamOptimizer().minimize(loss)
```

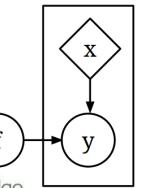


Figure: Deep GP

Tran et al. (2018) arXiv:1812.03973





# Summary

- Gaussian process are Bayesian nonparametric models that can represent distributions over smooth functions.
- Using expressive covariance kernel functions, GPs can model a variety of data (scalar, vector, sequential, structured, etc.).
- Inference can be done fully analytically (in case of Gaussian likelihood).
- Inference and learning are very computationally costly since exact methods require large matrix inversions.
- There is a variety of approximation methods to GPs that can bring down the learning and inference cost to  $O(n)$  and  $O(1)$ , respectively.
- Many new libraries based on TF, PyTorch, Keras – GP models despite computational constraints, GPs are certainly quite popular.

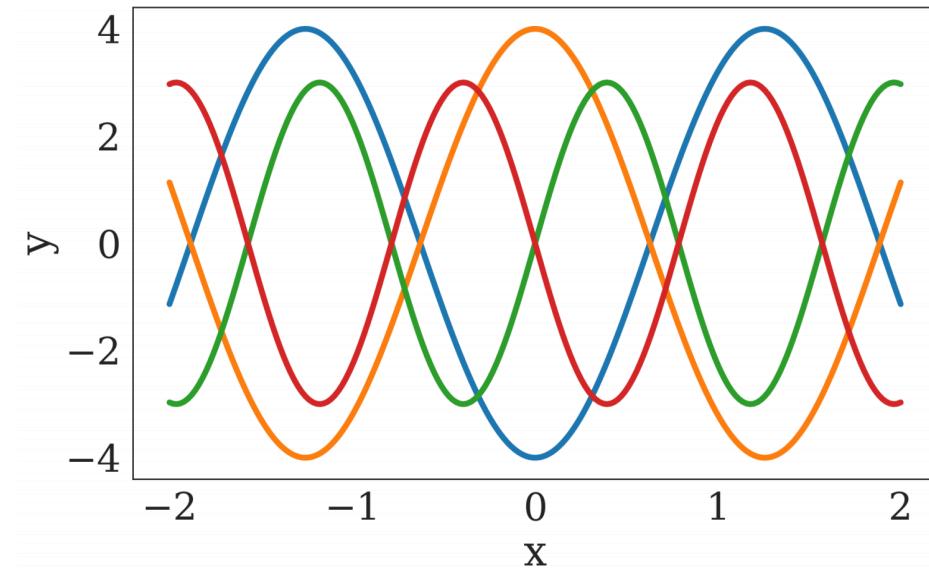
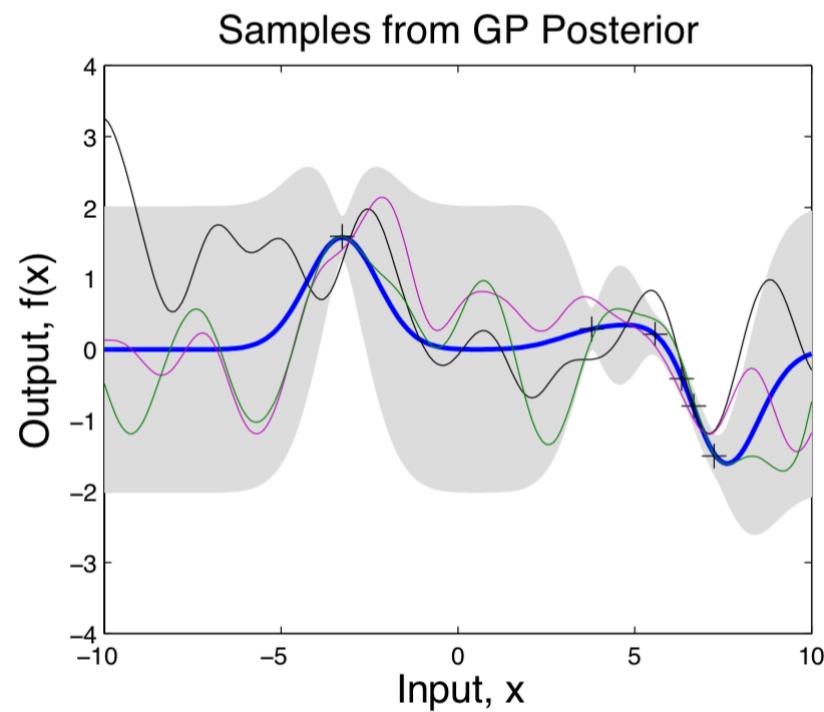


# Elements of Meta-learning and Neural Processes



# Example: Fast Learning of Functions

- So far, we assumed that data was generated by a single function.
- What if there are multiple data-generating functions, and each time we get only a few points from one of them. Can we identify it?





# What is meta-learning?

- Standard learning: Given a distribution over examples (single task), learn a function that minimizes the loss

$$\hat{\phi} = \arg \min_{\phi} \mathbb{E}_{z \sim \mathcal{D}} [l(f_{\phi}(z))]$$

- Learning-to-learn: Given a distribution over tasks, output an adaptation rule that can be used at test time to generalize from a task description

distribution over  
tasks/datasets

$$\hat{\theta} = \arg \min_{\theta} \mathbb{E}_{T \sim \mathcal{P}} \{ \mathcal{L}_T[g_{\theta}(T)] \}, \quad \text{where}$$

$$\mathcal{L}_T[g_{\theta}(T)] := \mathbb{E}_{z \sim \mathcal{D}_T} [l(f_{\phi}(z))], \quad \phi := g_{\theta}(T)$$

distribution over  
examples for task T

adaptation rule takes  
a task description as input  
and outputs a model





# Example: Few-shot Image Classification

$$\hat{\theta} = \arg \min_{\theta} \mathbb{E}_{T \sim \mathcal{P}} \{ \mathcal{L}_T[g_{\theta}(T)] \}, \quad \text{where}$$

$$\mathcal{L}_T[g_{\theta}(T)] := \mathbb{E}_{z \sim \mathcal{D}_T} [l(f_{\phi}(z))], \phi := g_{\theta}(T)$$

Considered in:

- Lake et al., '15
  - Vinyals et al., '16
  - Santoro et al., '16
  - Ravi, Larochelle, '17
  - Finn et al., '17
- ...

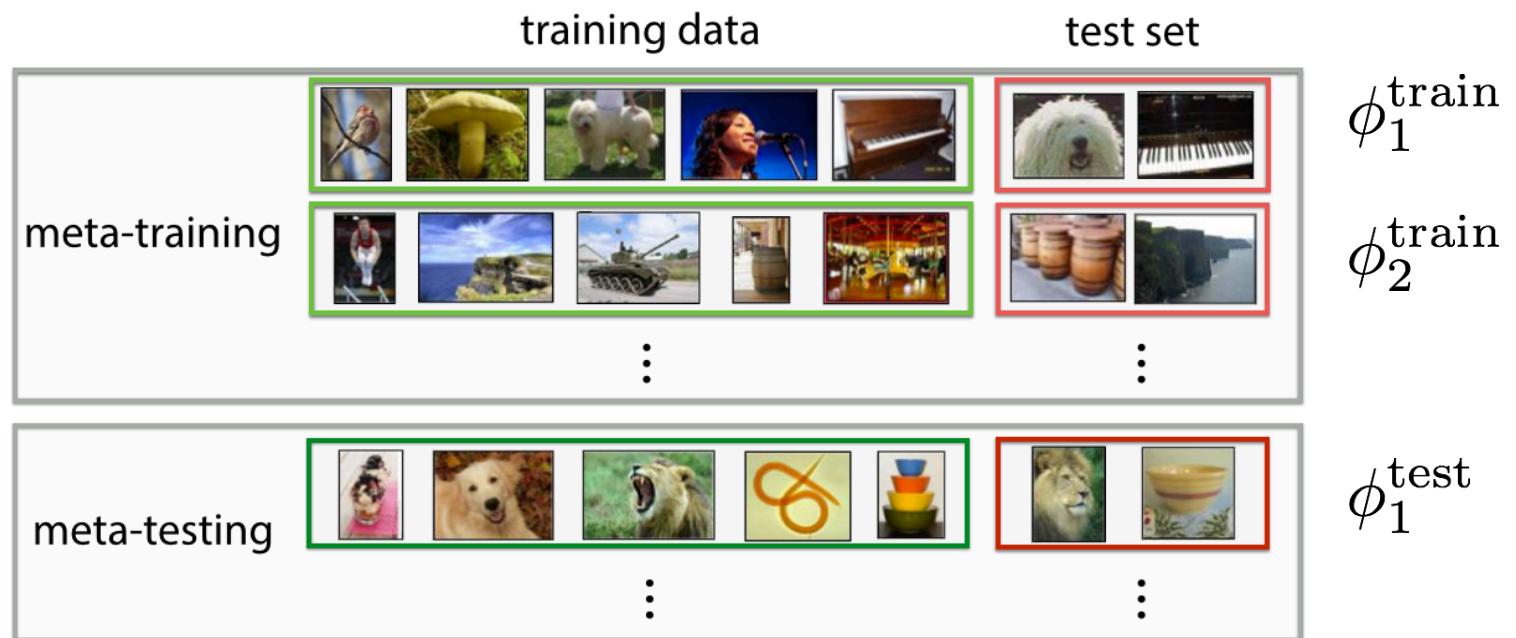
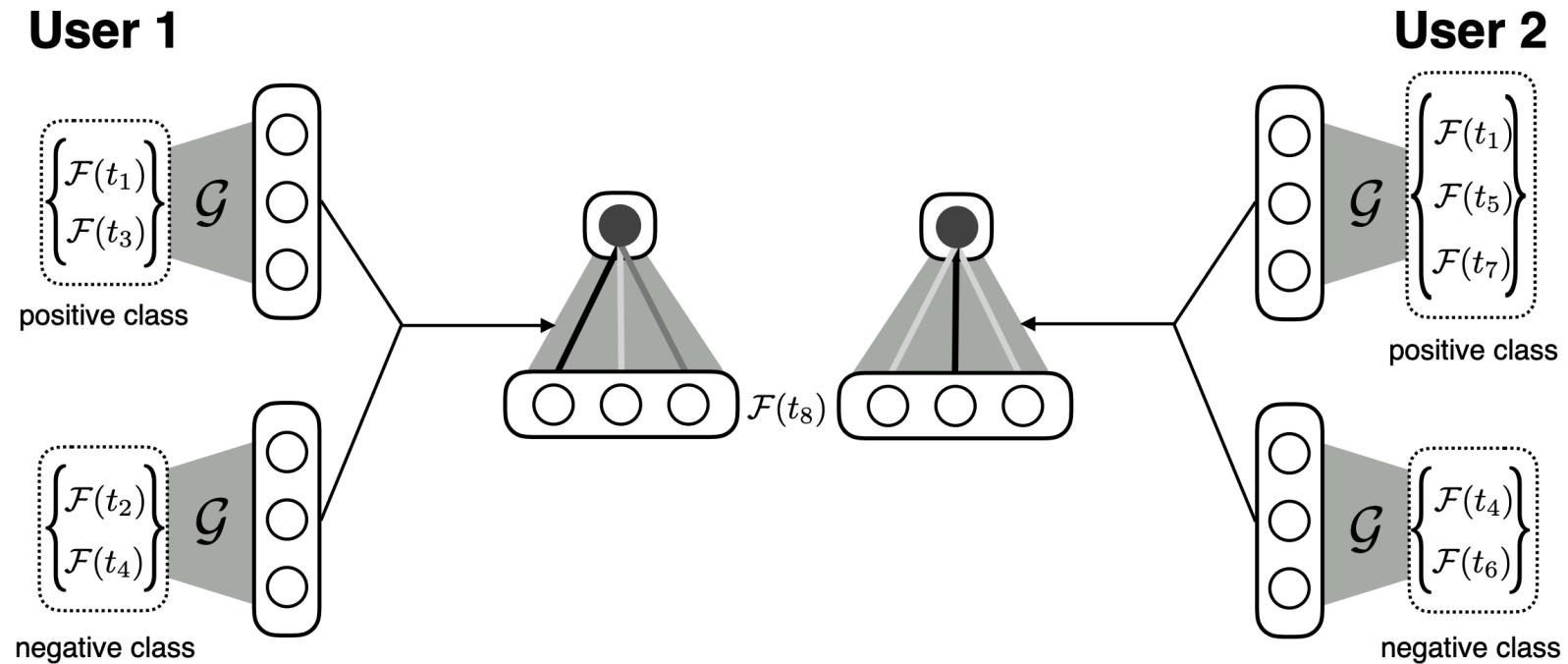


Image: bair.berkeley.edu



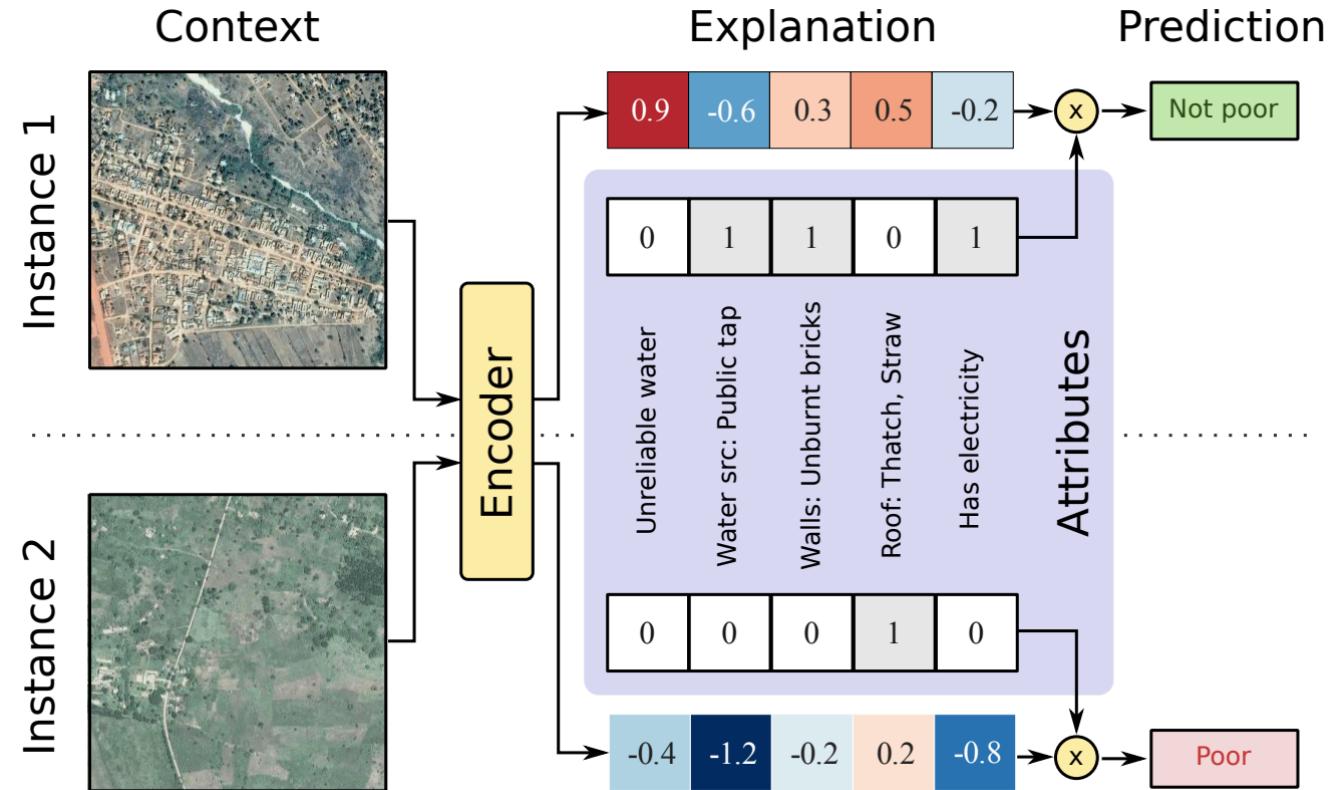


# Example: Few-shot User-specific Recommendation



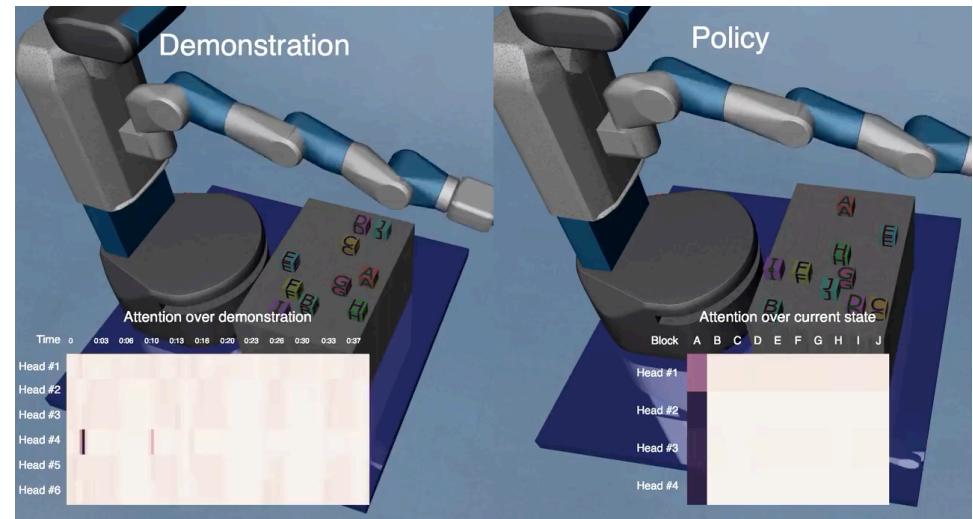
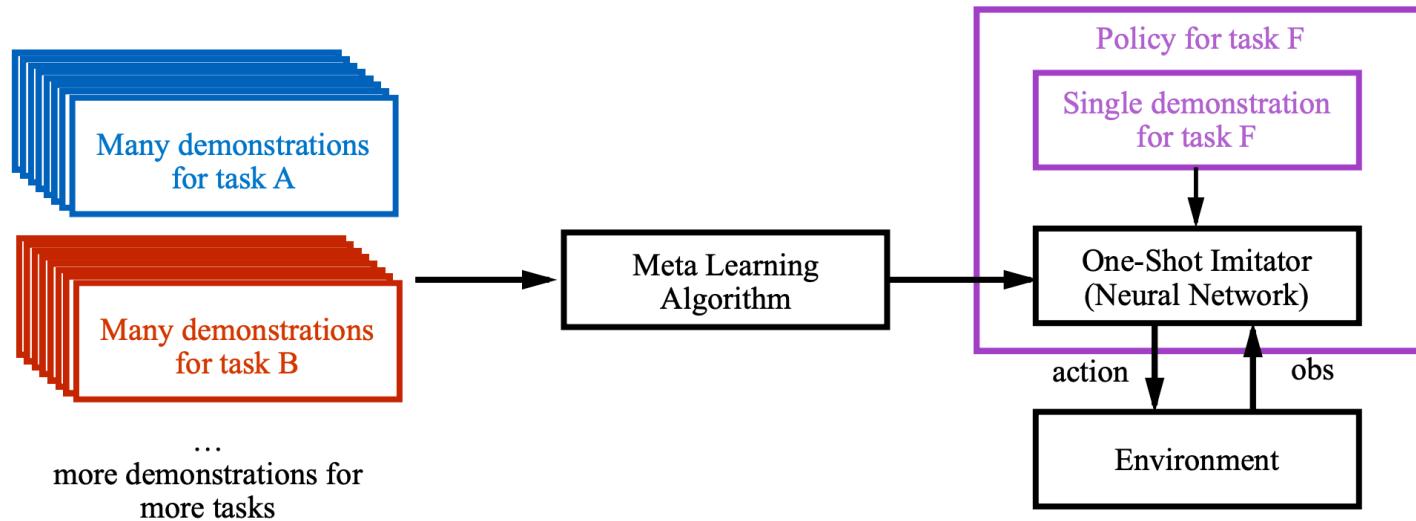


# Example: Contextual Interpretability





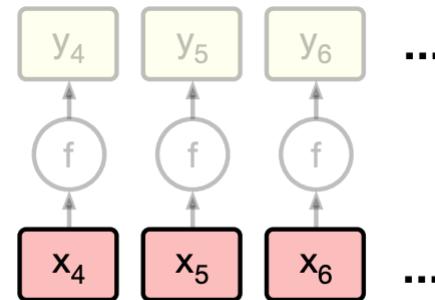
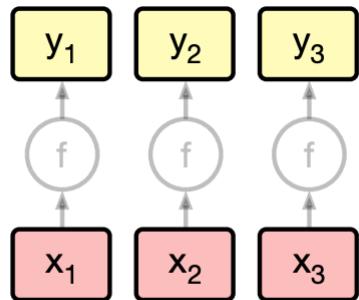
# Example: One-shot Imitation Learning



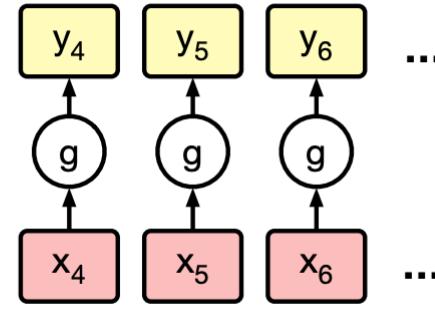
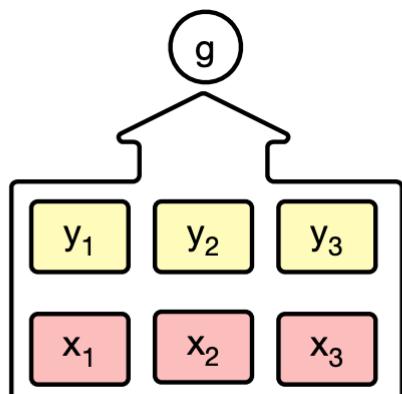


# Conditional Neural Processes

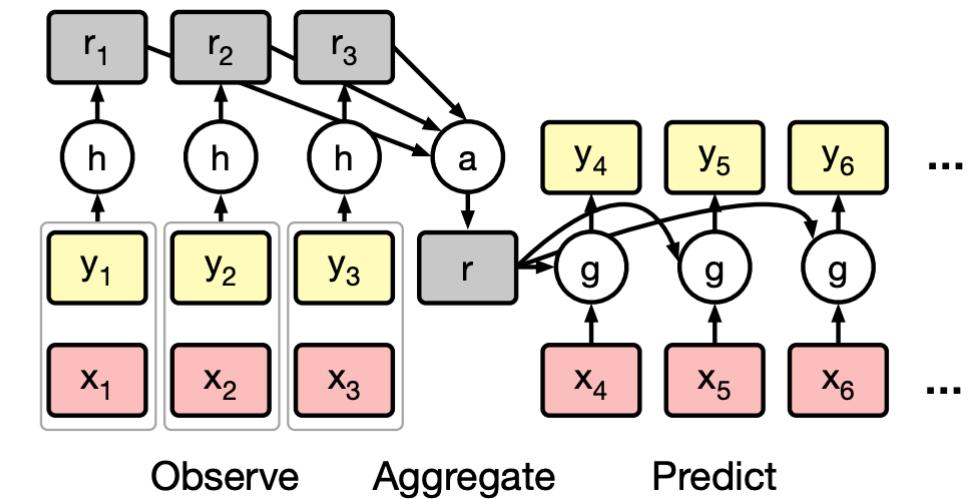
a Data



b Supervised Learning

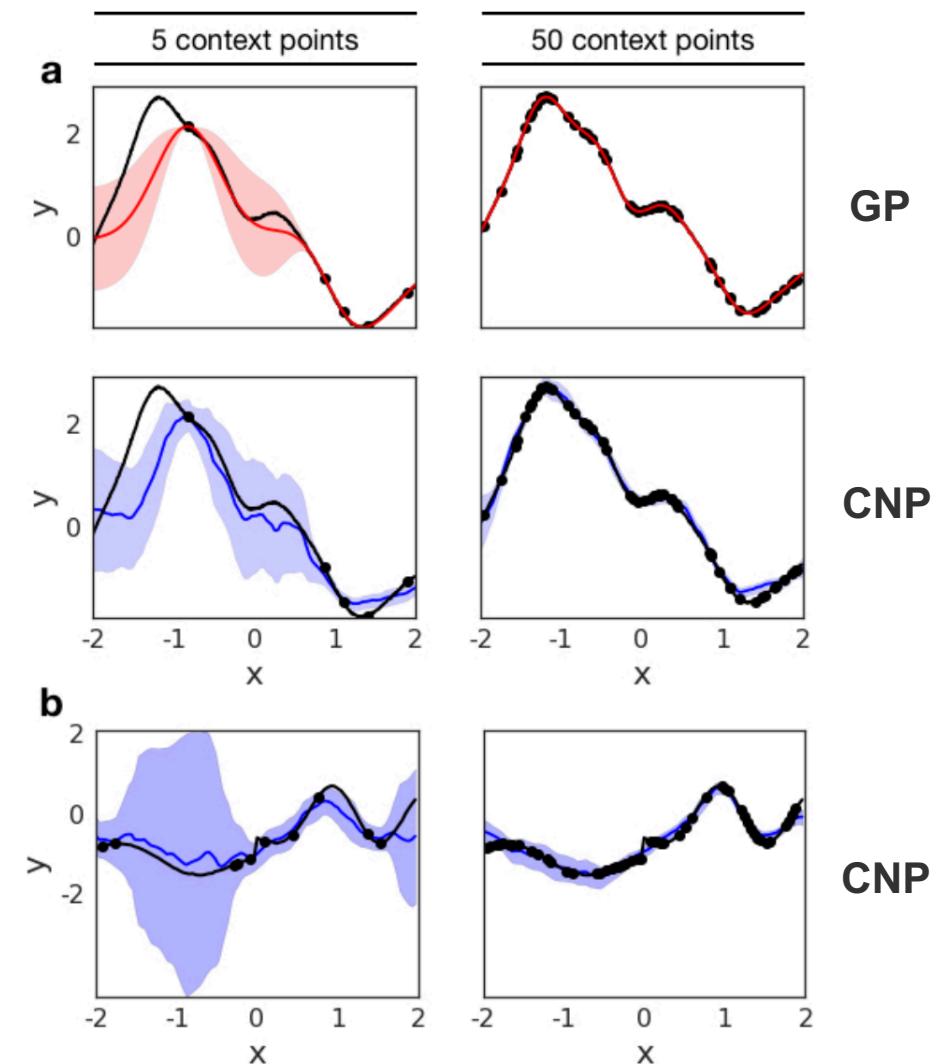
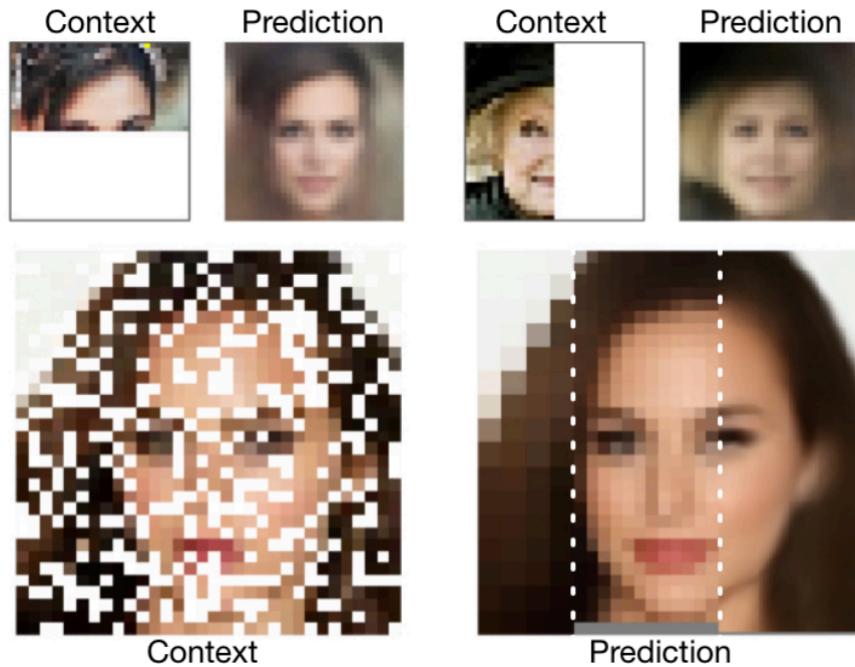


CNP architecture:





# Conditional Neural Processes

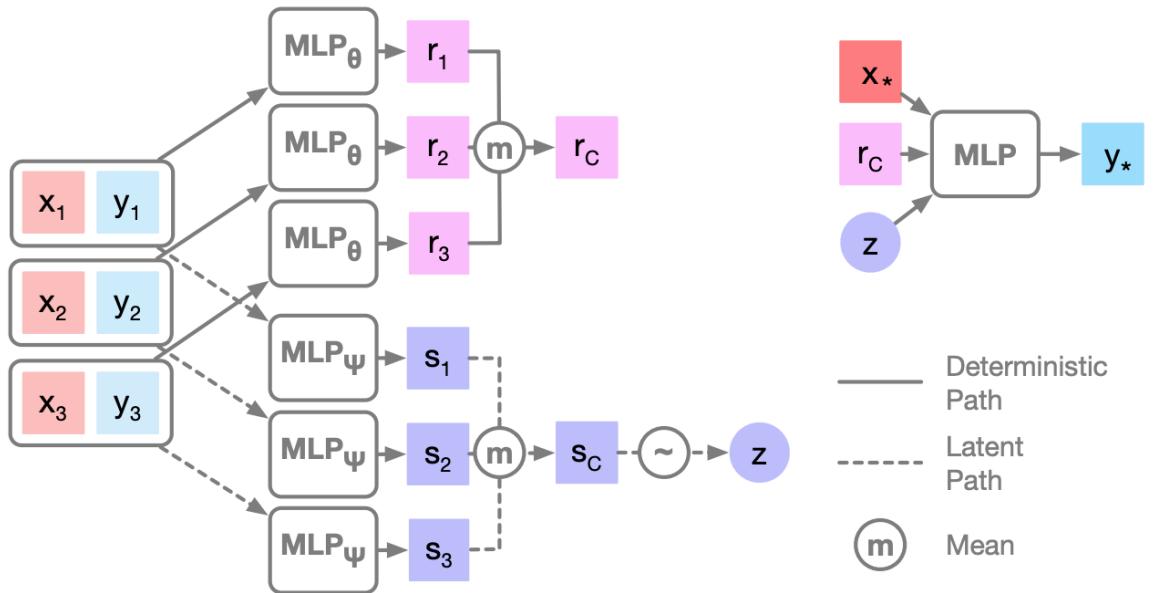




# Attentive Neural Processes

## NEURAL PROCESS

### ENCODER

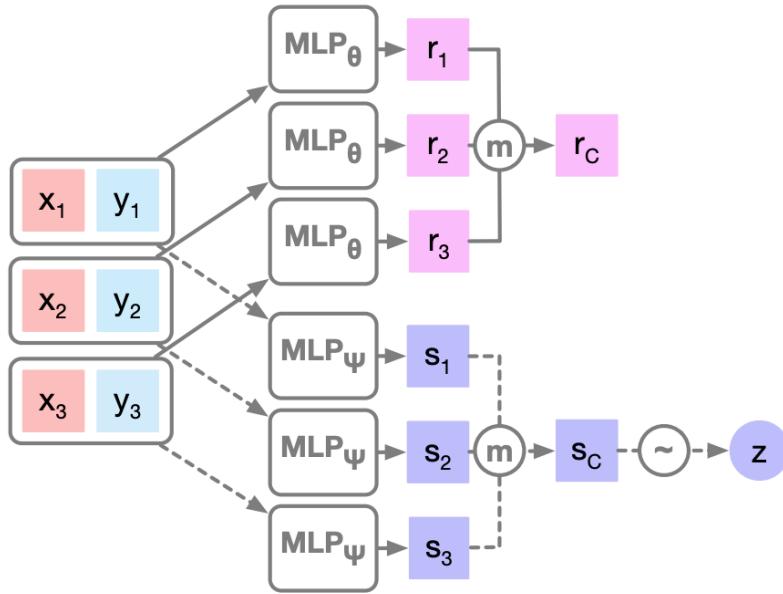




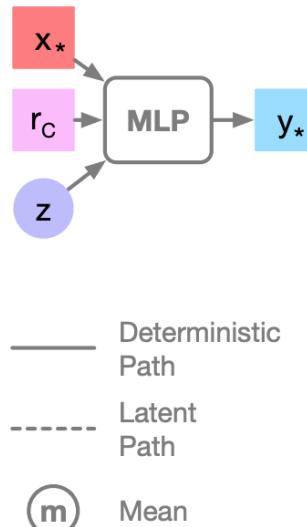
# Attentive Neural Processes

## NEURAL PROCESS

### ENCODER

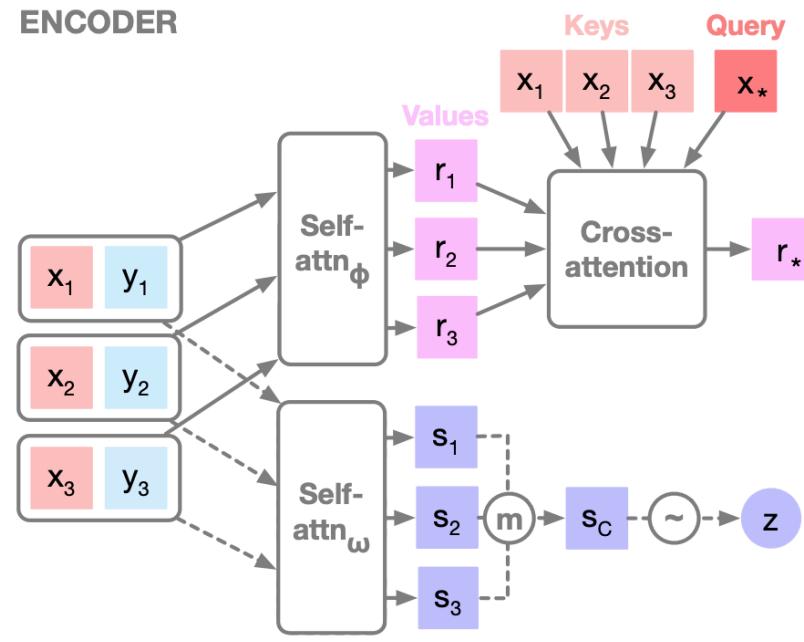


### DECODER

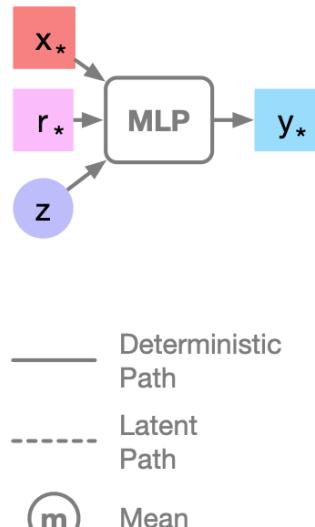


## ATTENTIVE NEURAL PROCESS

### ENCODER

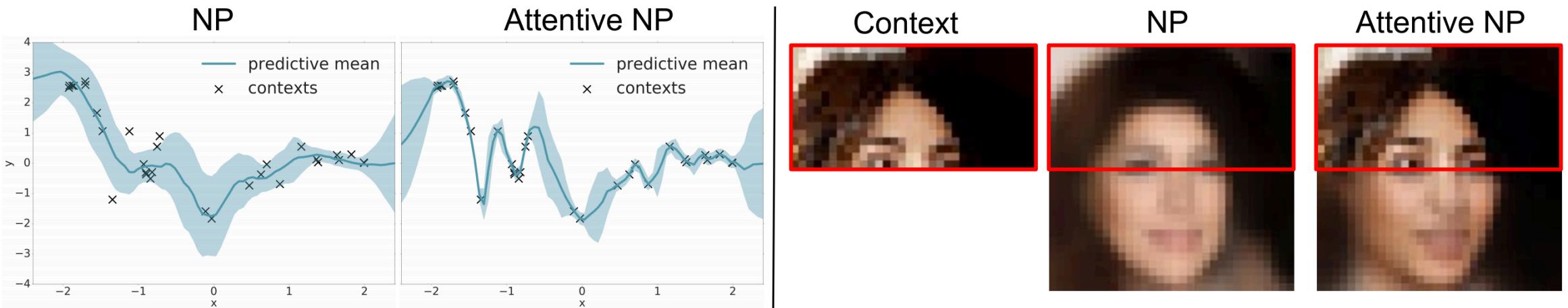


### DECODER



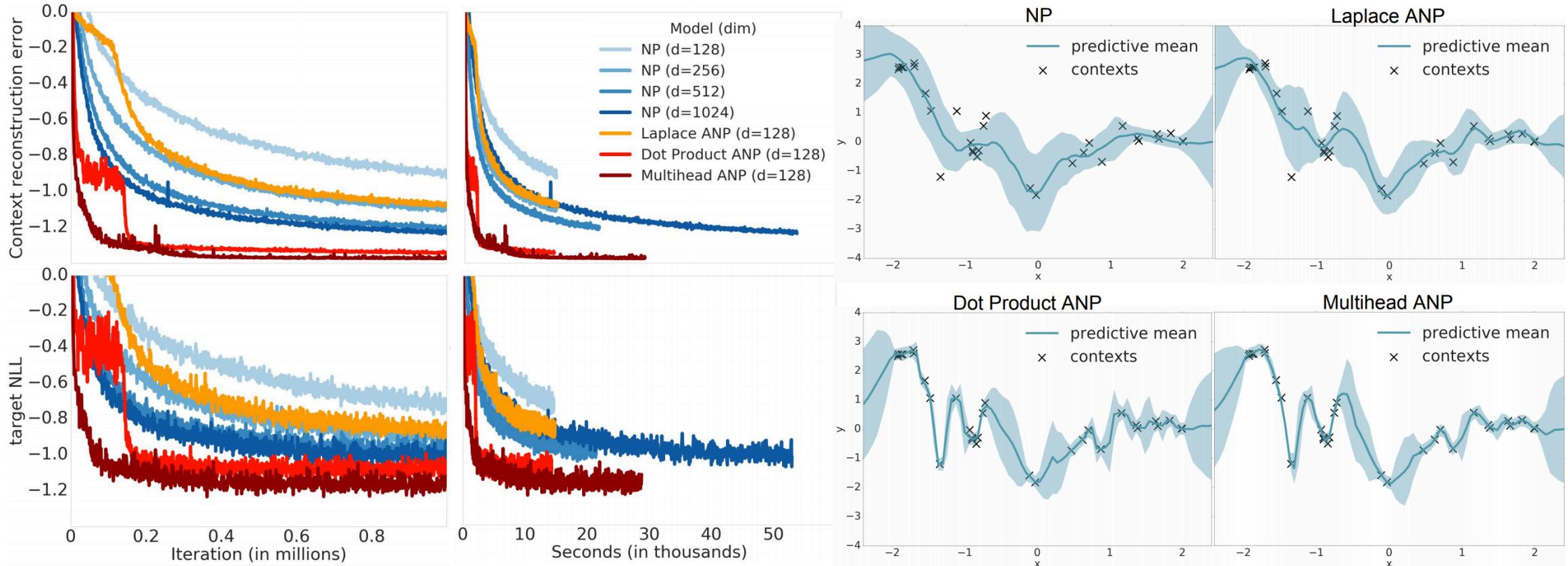


# Attentive Neural Processes





# Attentive Neural Processes





# Summary

- There are cases when learning a single function is not enough – contextual models are used in such case.
- Few-shot learning is a popular application of meta-learning, where contextual models are trained on distributions of different tasks.  
Examples:
  - Solve different sub-problems
  - Imitate different demonstrations
  - Make predictions about different user preferences
- Neural processes propose an alternative to kernel learning (kernel becomes fully implicit; the model is scalable without approximations)



