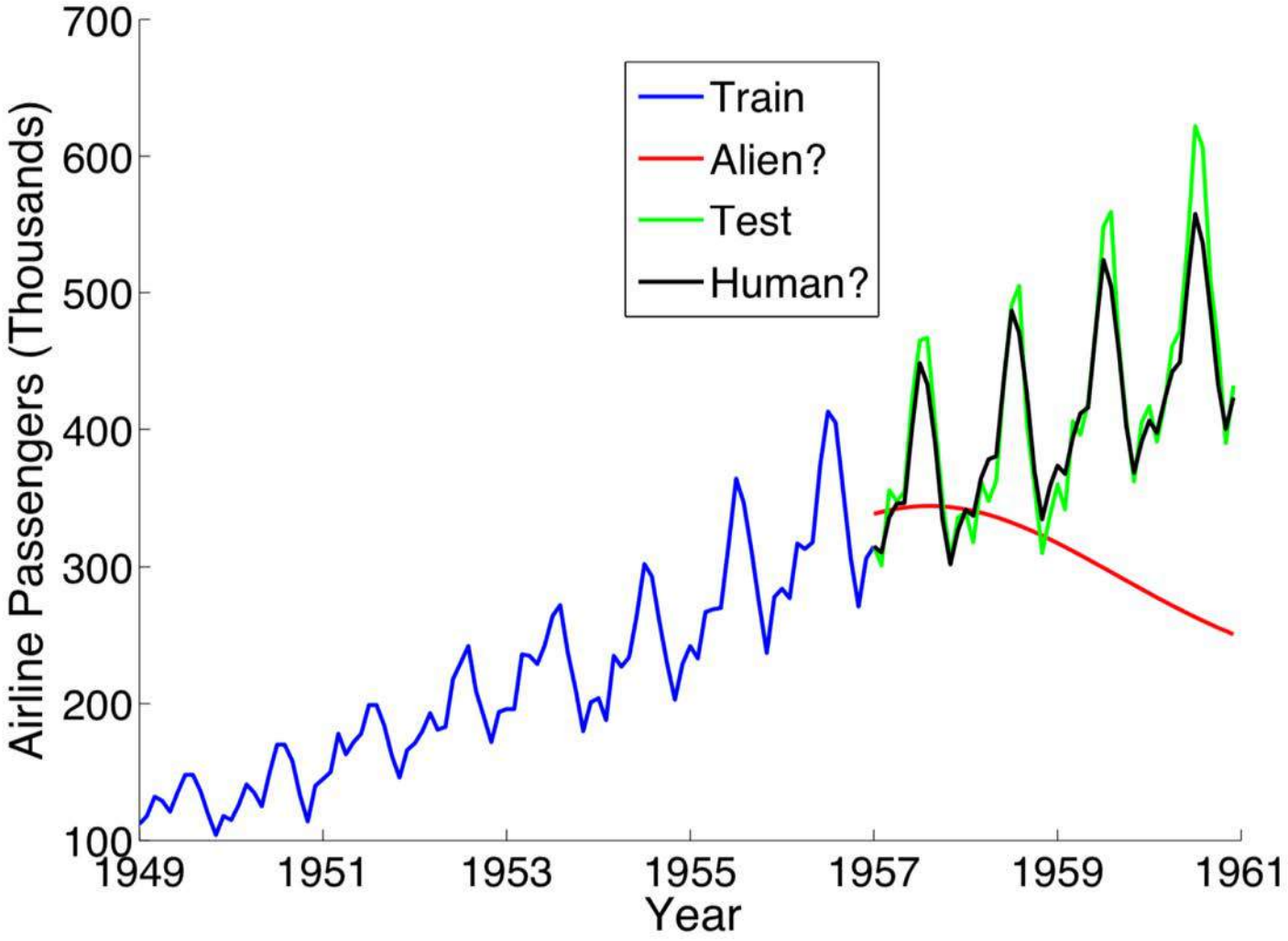
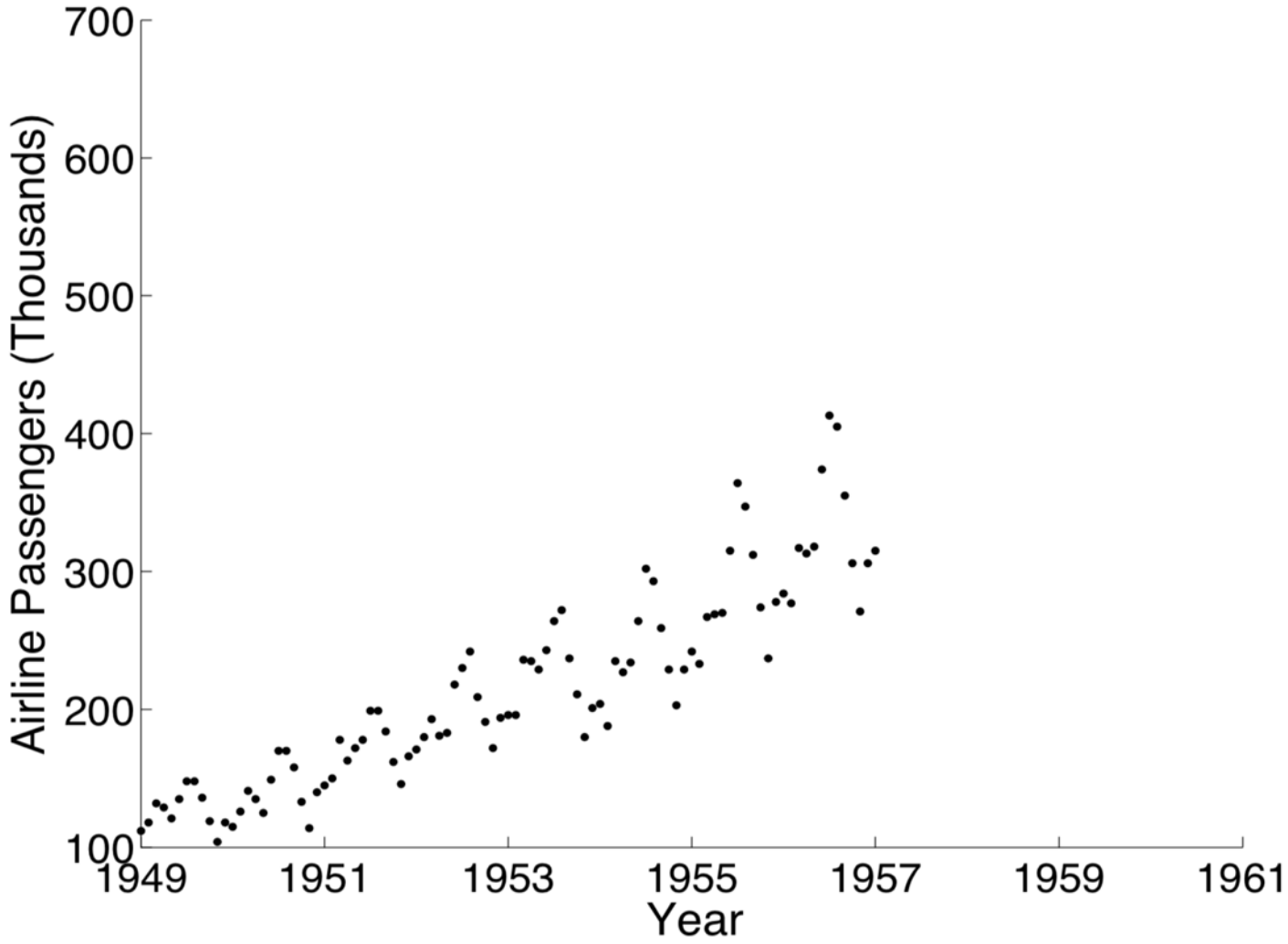
Function Learning Example



Slide credit: Andrew G. Wilson

Learning Functions from Data



Slide credit: Andrew G. Wilson

Learning Functions from Data

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

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

$\boldsymbol{\phi}(x)$ is a vector of basis functions. For example, if $\boldsymbol{\phi}(x) = (1, x, x^2)$ and $x \in \mathbb{R}^1$ then $f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2$ is a quadratic function.

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

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

Slide credit: Andrew G. Wilson

Learning Functions from Data

A probabilistic approach

We could explicitly account for noise in our model.

- $y(x) = f(x, \boldsymbol{w}) + \epsilon(x)$, 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

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

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

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

Slide credit: Andrew G. Wilson

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 σ^2.
- Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development.
- Both approaches are prone to *over-fitting* for flexible $f(x, \boldsymbol{w})$: low error on the training data, high error on the test set.

Regularization

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

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

- **But how should we define complexity, and how much should we penalize complexity?**
- Can set λ using *cross-validation*.

Slide credit: Andrew G. Wilson

Learning Functions from Data

Bayes' Rule

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

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

Slide credit: Andrew G. Wilson

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 $\mathbf{w}$.

Slide credit: Andrew G. Wilson

Parametric vs. Nonparameteric 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.

Nonparameteric models:

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

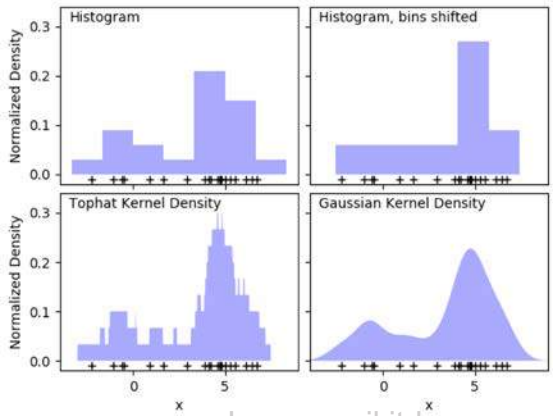


Image: scikit-learn.org

Bayesian nonparameterics:

- 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.

Parametric Bayesian Inference

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

- A parametric likelihood: $\mathbf{x} \sim p(\cdot|\theta)$
- Prior on θ: $\pi(\theta)$
- 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 → Gaussian posterior distribution
- Dirichilet distribution prior + 2D Multinomial likelihood → Dirichlet posterior distribution
- Sparsity-inducing priors + some likelihood models → 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

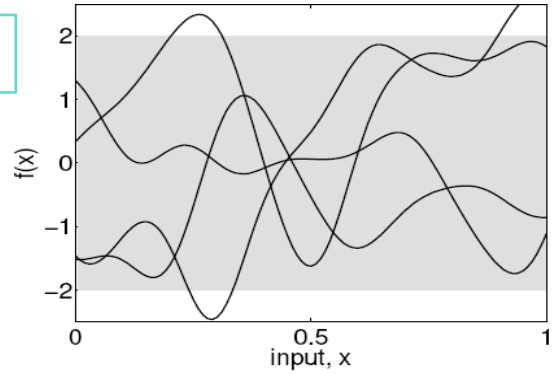
probability measure

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

binary matrix

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

function

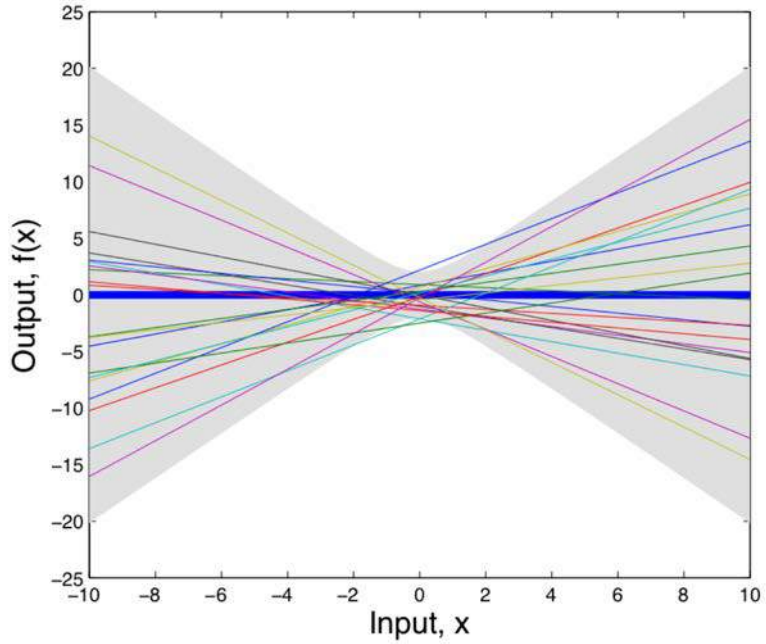


Gaussian Process Prior [Doob, 1944; Rasmussen & Williams, 2006]
+ Gaussian/Sigmoid/Softmax likelihood

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:

$$
\begin{aligned}
f(x|a_0, a_1) &= a_0 + a_1 x\,, \qquad a_0, a_1 \sim \mathcal{N}(0,1)\,. \\
\mathbb{E}[f(x)] &= \mathbb{E}[a_0] + \mathbb{E}[a_1]x = 0\,. \\
\mathrm{cov}[f(x_b), f(x_c)] &= \mathbb{E}[f(x_b)f(x_c)] - \mathbb{E}[f(x_b)]\mathbb{E}[f(x_c)] \\
&= \mathbb{E}[a_0^2 + a_0 a_1 (x_b + x_c) + a_1^2 x_b x_c] - 0 \\
&= \mathbb{E}[a_0^2] + \mathbb{E}[a_1^2 x_b x_c] + \mathbb{E}[a_0 a_1 (x_b + x_c)] \\
&= 1 + x_b x_c + 0 \\
&= 1 + x_b x_c\,.
\end{aligned}
$$

Function-space View

- Therefore any collection of values has a joint Gaussian distribution (not because of randomness in X, note that here we have lower-case x which means they are given as fixed, but because of randomness in the function f):

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

- 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

$$[f(x_1), \ldots, f(x_N)] \sim \mathcal{N}(\boldsymbol{\mu}, K) \quad (30$$
$$\boldsymbol{\mu}_i = m(x_i) \quad (31$$
$$K_{ij} = k(x_i, x_j) , \quad (32$$

for any collection of input values $x_1, \ldots, x_N$. 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:

$$f(x, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(x)$$
$$p(\boldsymbol{w}) = \mathcal{N}(0, \Sigma_w)$$

- Moments of the the induced distribution over functions:

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

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

Gaussian Processes

Interpretability:

- We are ultimately more interested in – and have stronger intuitions about – the *functions* that model our data and 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

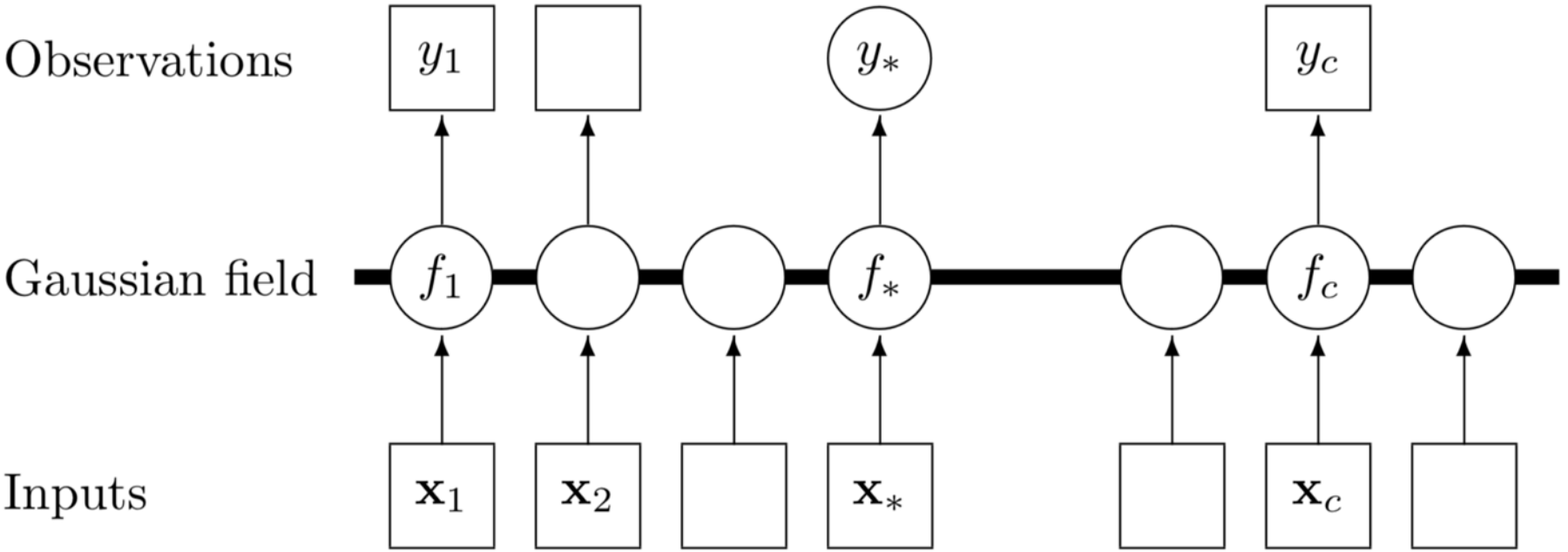


Image source: GPML book

Example: RBF kernel

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

- 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 ℓ control amplitudes and wiggliness of these functions.
- GPs with an RBF kernel have large support and are *universal approximators*.

For intuitions: https://distill.pub/2019/visual-exploration-gaussian-processes/

Example: RBF kernel

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

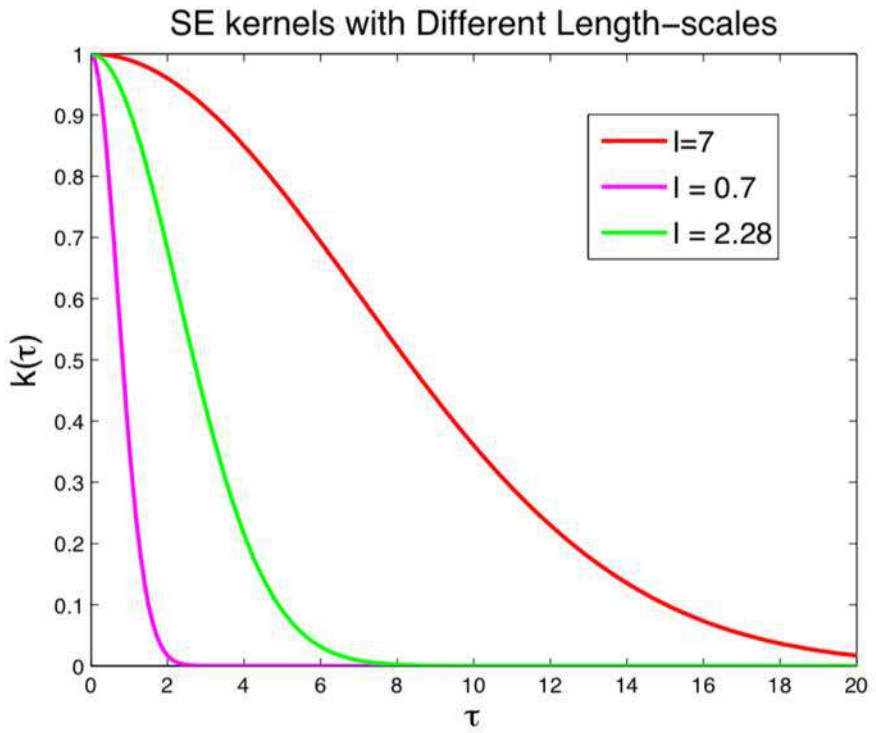
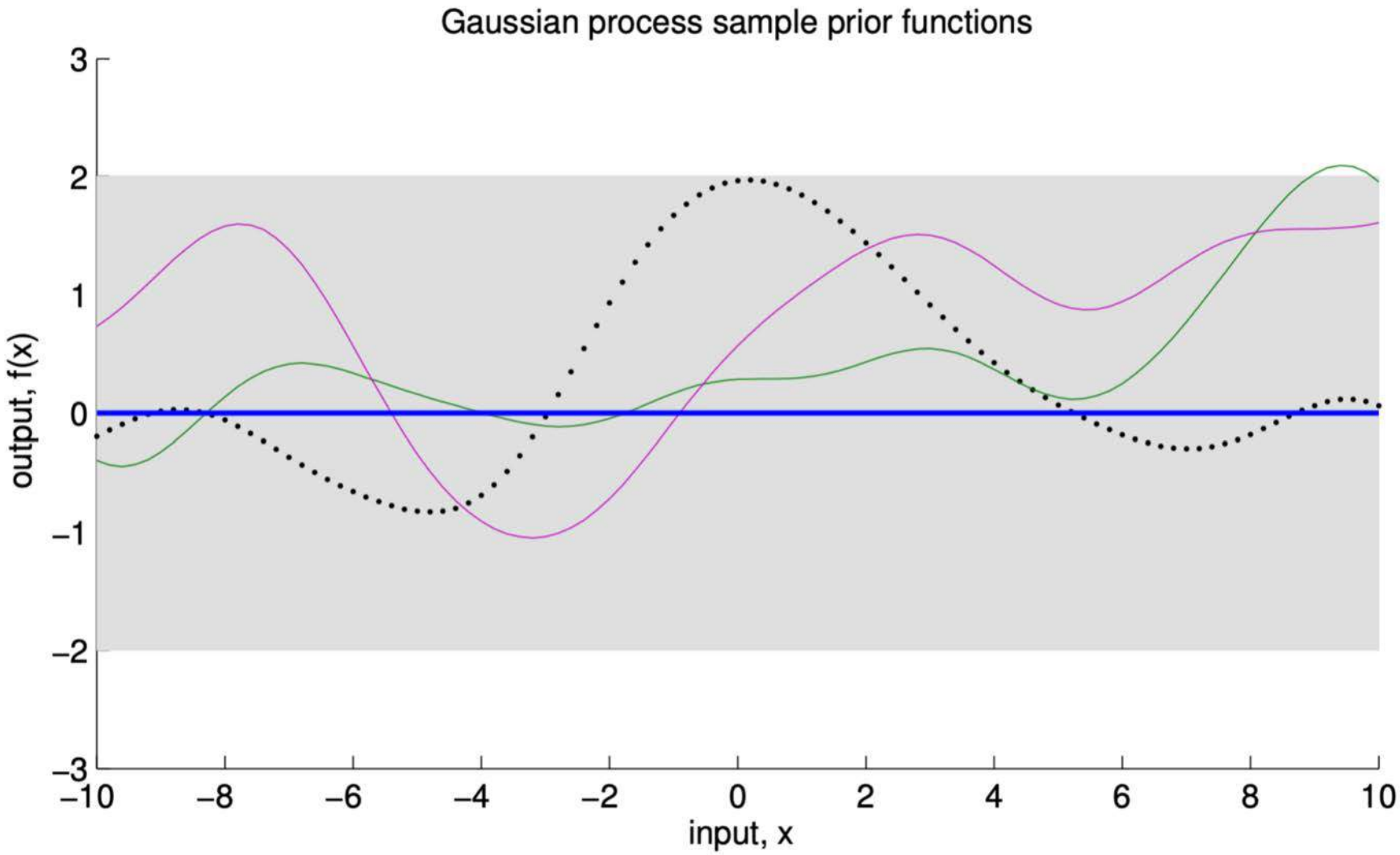


Figure: SE kernels with different length-scales, as a function of $\tau = x - x'$.

Slide credit: Andrew G. Wilson

Example: RBF kernel

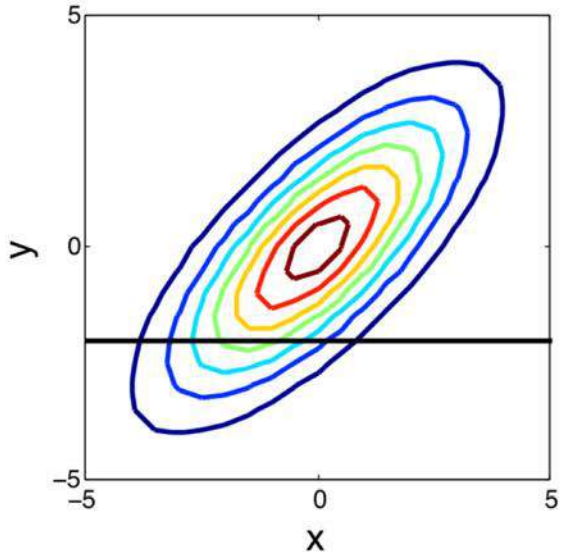


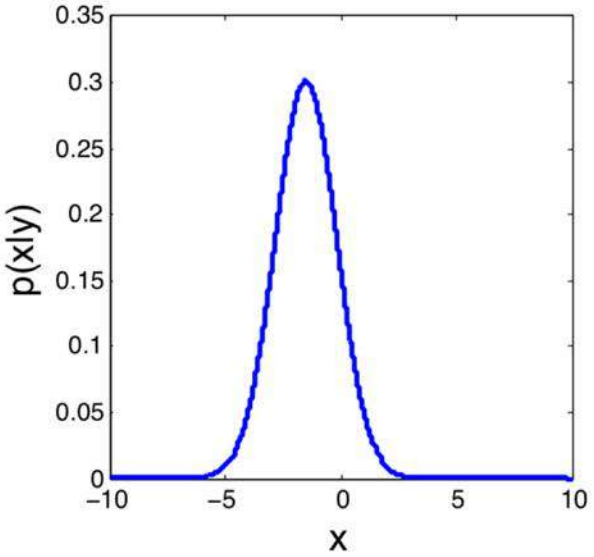
Slide credit: Andrew G. Wilson

Gaussian Process Inference

- Observed noisy data $\boldsymbol{y} = (y(x_1), \ldots, y(x_N))^{\mathrm{T}}$ 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 θ.
- Infer $p(\boldsymbol{f}_* | \boldsymbol{y}, X, X_*)$ 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 $\boldsymbol{y} = (y(x_1), \ldots, y(x_N))^{\mathrm{T}}$ 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 θ.
- Infer $p(\boldsymbol{f}_* | \boldsymbol{y}, X, X_*)$ for the noise free function f evaluated at test points X_*.

Joint distribution

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

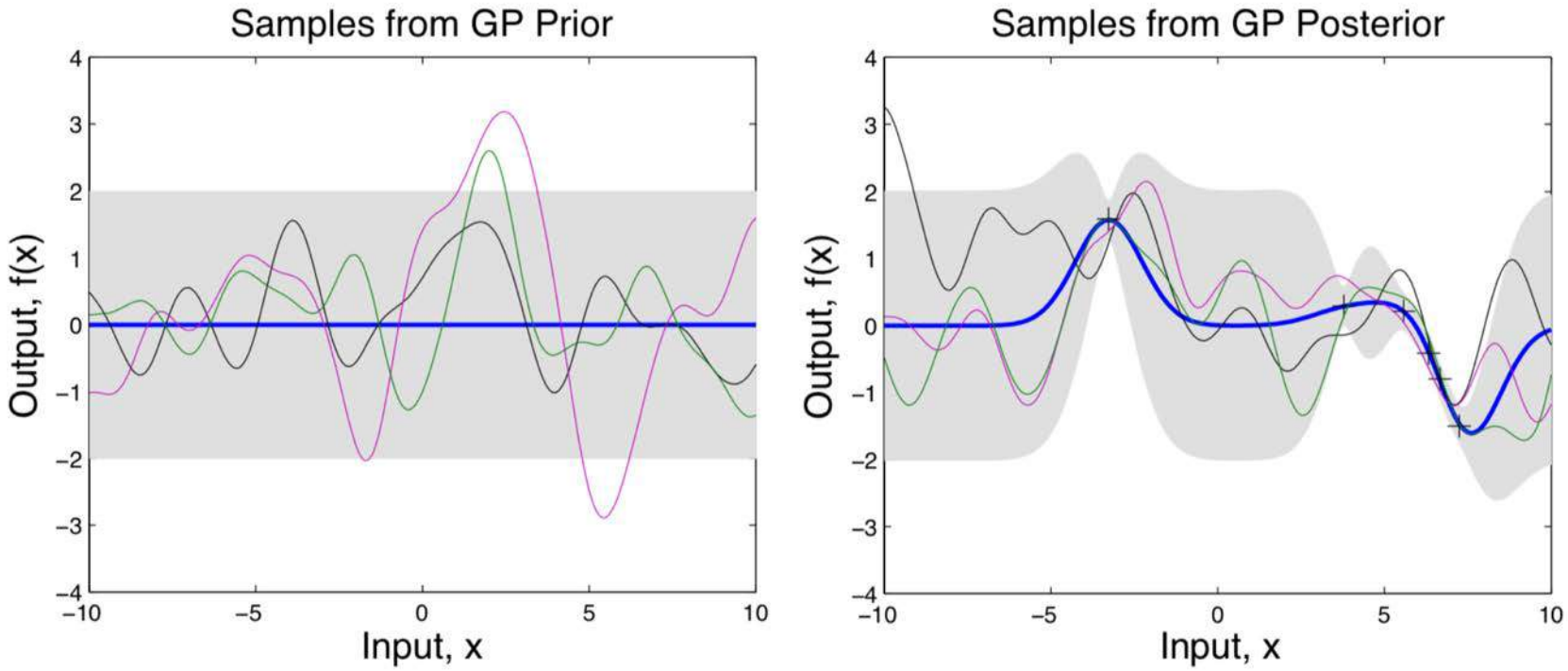
Conditional predictive distribution

$$\begin{aligned} \boldsymbol{f}_* | X_*, X, \boldsymbol{y}, \boldsymbol{\theta} &\sim \mathcal{N}(\bar{\boldsymbol{f}}_*, \mathrm{cov}(\boldsymbol{f}_*)), \\ \bar{\boldsymbol{f}}_* &= K_\theta(X_*, X)[K_\theta(X,X) + \sigma^2 I]^{-1} \boldsymbol{y}, \\ \mathrm{cov}(\boldsymbol{f}_*) &= K_\theta(X_*, X_*) - K_\theta(X_*, X)[K_\theta(X,X) + \sigma^2 I]^{-1} K_\theta(X, X_*). \end{aligned}$$

Slide credit: Andrew G. Wilson

Gaussian Process Inference

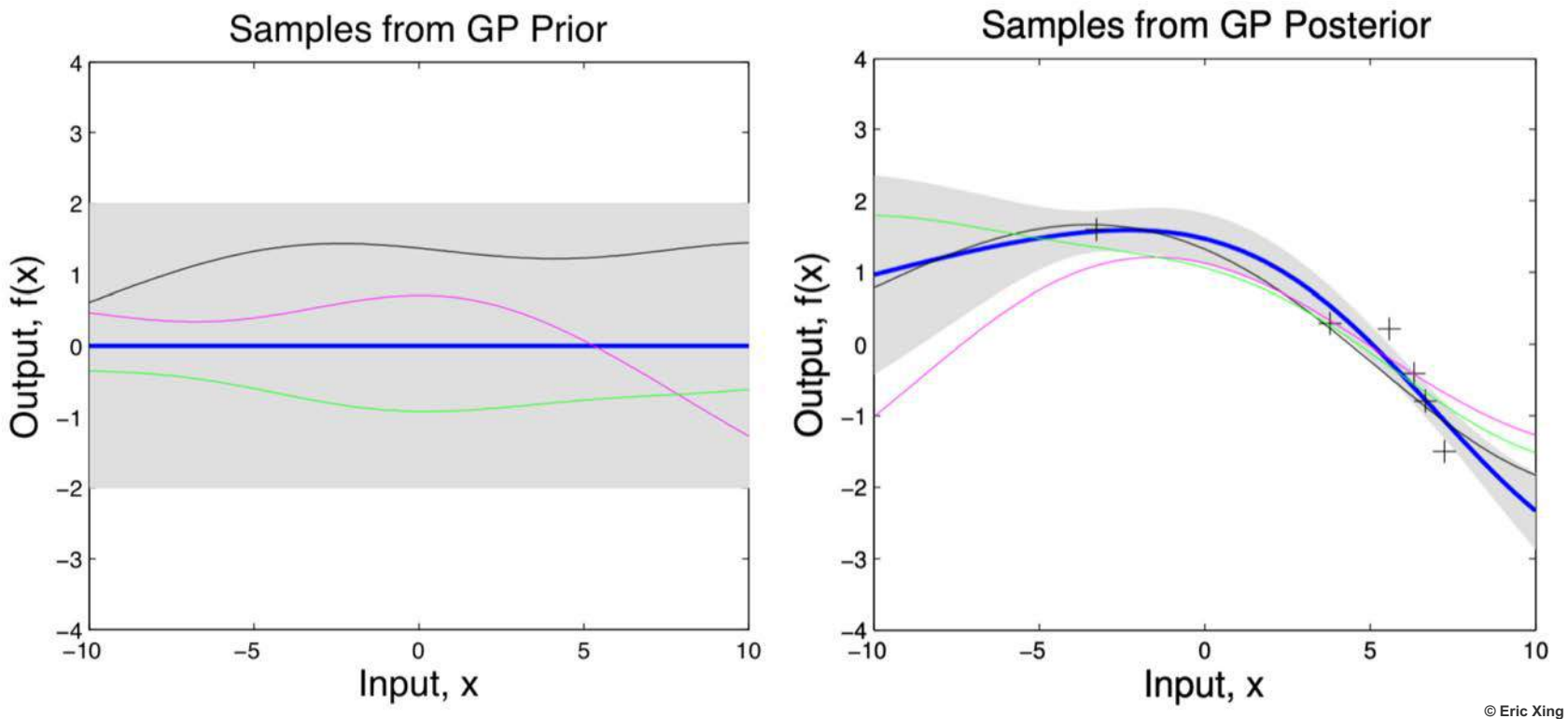
- Specify $f(x) \sim \mathcal{GP}(0, k)$.
- Choose $k_{\text{RBF}}(x, x') = a_0^2 \exp(-\frac{||x-x'||^2}{2\ell_0^2})$. Choose values for a_0 and ℓ_0.
- Observe data, look at the prior and posterior over functions.



Slide credit: Andrew G. Wilson

Gaussian Process Inference

Increase the length-scale ℓ.



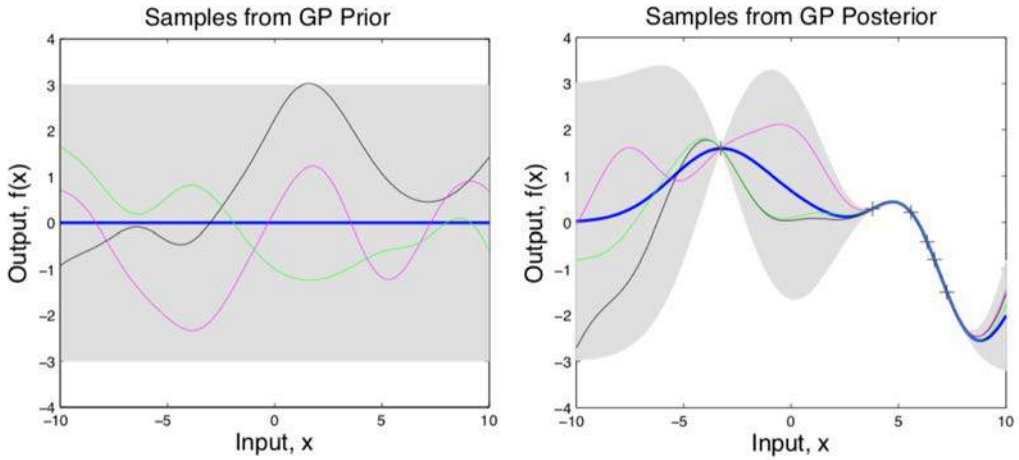
Slide credit: Andrew G. Wilson

Gaussian Process Learning

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

$$p(\boldsymbol{y}|\boldsymbol{\theta}, X) = \int p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}|\boldsymbol{\theta}, X)d\boldsymbol{f} \,. \tag{48}$$

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



Slide credit: Andrew G. Wilson

Gaussian Process Learning

1. Learning: Optimize marginal likelihood,

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

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

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

$$\begin{aligned}
\boldsymbol{f}_*|X_*, X, \boldsymbol{y}, \boldsymbol{\theta} &\sim \mathcal{N}(\bar{\boldsymbol{f}}_*, \mathrm{cov}(\boldsymbol{f}_*))\,, \\
\bar{\boldsymbol{f}}_* &= K_\theta(X_*, X)[K_\theta(X, X) + \sigma^2 I]^{-1}\boldsymbol{y}\,, \\
\mathrm{cov}(\boldsymbol{f}_*) &= K_\theta(X_*, X_*) - K_\theta(X_*, X)[K_\theta(X, X) + \sigma^2 I]^{-1}K_\theta(X, X_*)\,.
\end{aligned}$$

Slide credit: Andrew G. Wilson

Rich Literature on Other Types of Covariance Kernels

$$k_{\text{Matern}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{\ell}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu}r}{\ell}\right)$$

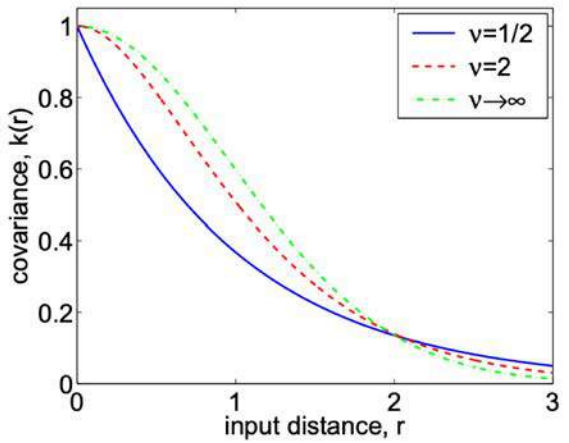
Kernels as functions of the distance:

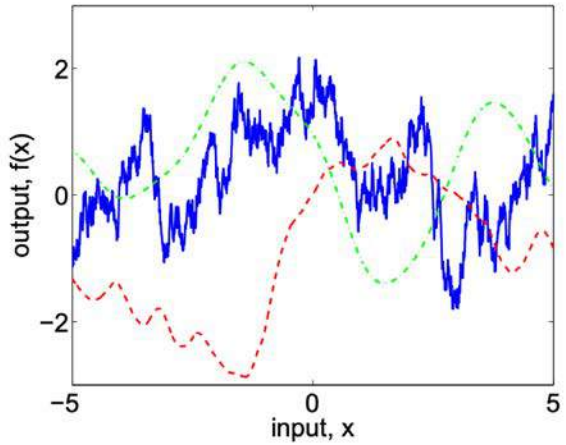
$$k_{\text{SE}}(\tau) = \exp(-0.5\tau^2/\ell^2)$$

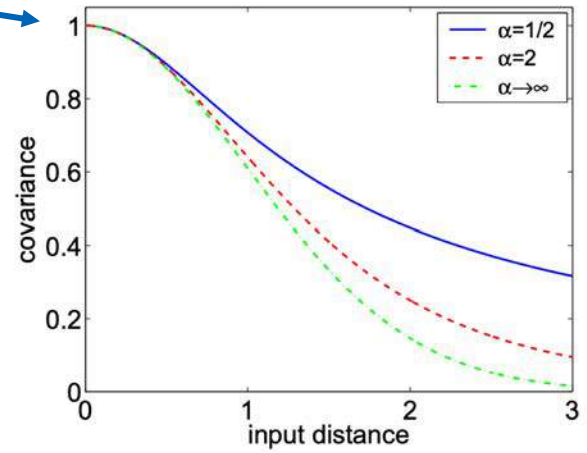
$$k_{\text{MA}}(\tau) = a(1+\frac{\sqrt{3}\tau}{\ell})\exp(-\frac{\sqrt{3}\tau}{\ell})$$

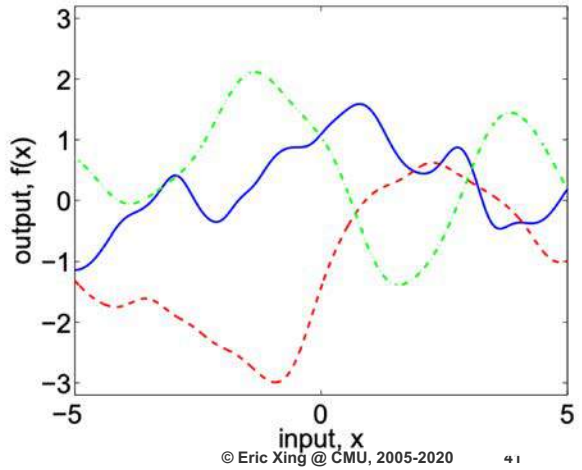
$$k_{\text{RQ}}(\tau) = (1+\frac{\tau^2}{2\,\alpha\,\ell^2})^{-\alpha}$$

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









Rich Literature on Other Types of Covariance Kernels

Kernels as functions of the distance:

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

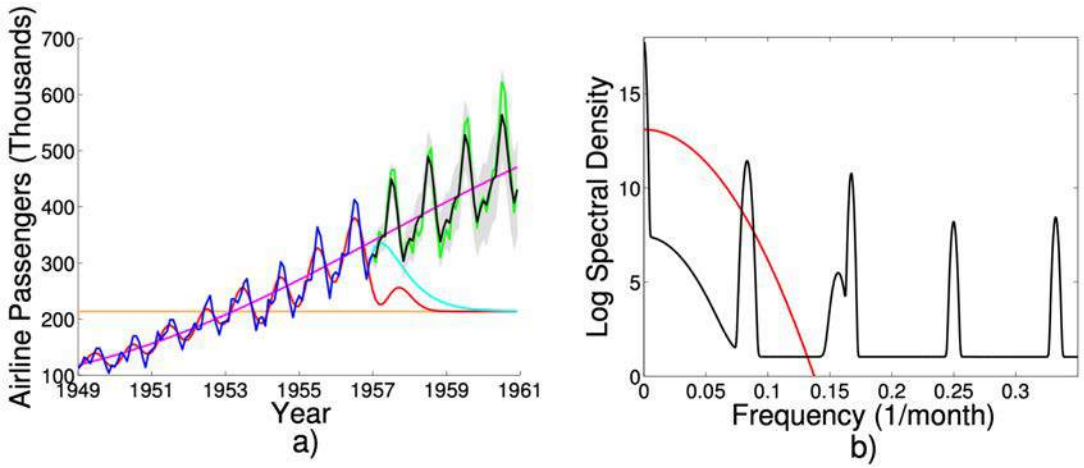
$$k_{\mathrm{MA}}(\tau) = a(1+\frac{\sqrt{3}\tau}{\ell})\exp(-\frac{\sqrt{3}\tau}{\ell})$$

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

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

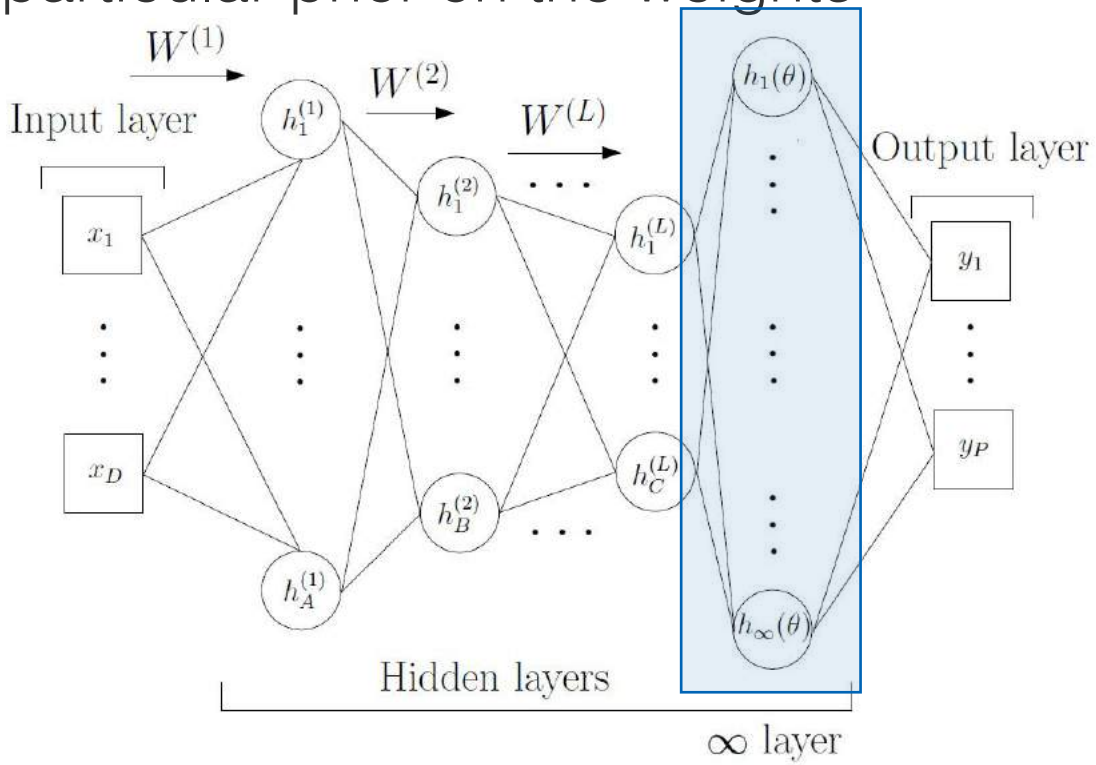
Spectral mixture kernels (Wilson & Adams, 2013)

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



Gaussian Process and Deep Kernel Learning

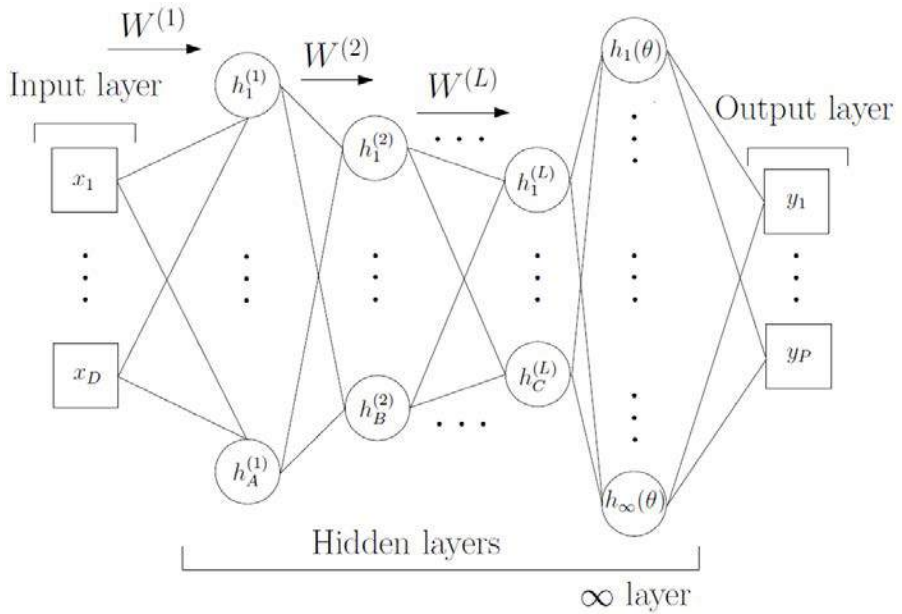
- By adding GP as a layer to a deep neural net, we can think of it as adding an infinite hidden layer with a particular prior on the weights
- Deep kernel learning [Wilson et al., 2016]
  - Combines the inductive biases of deep models with the non-parametric flexibility of Gaussian processes
  - GPs add powerful regularization to the network
  - Additionally, they provide predictive uncertainty estimates



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:

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



Wilson et al., NIPS 2016

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:

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

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

Wilson et al., NIPS 2016

Deep Kernel Learning for Regression

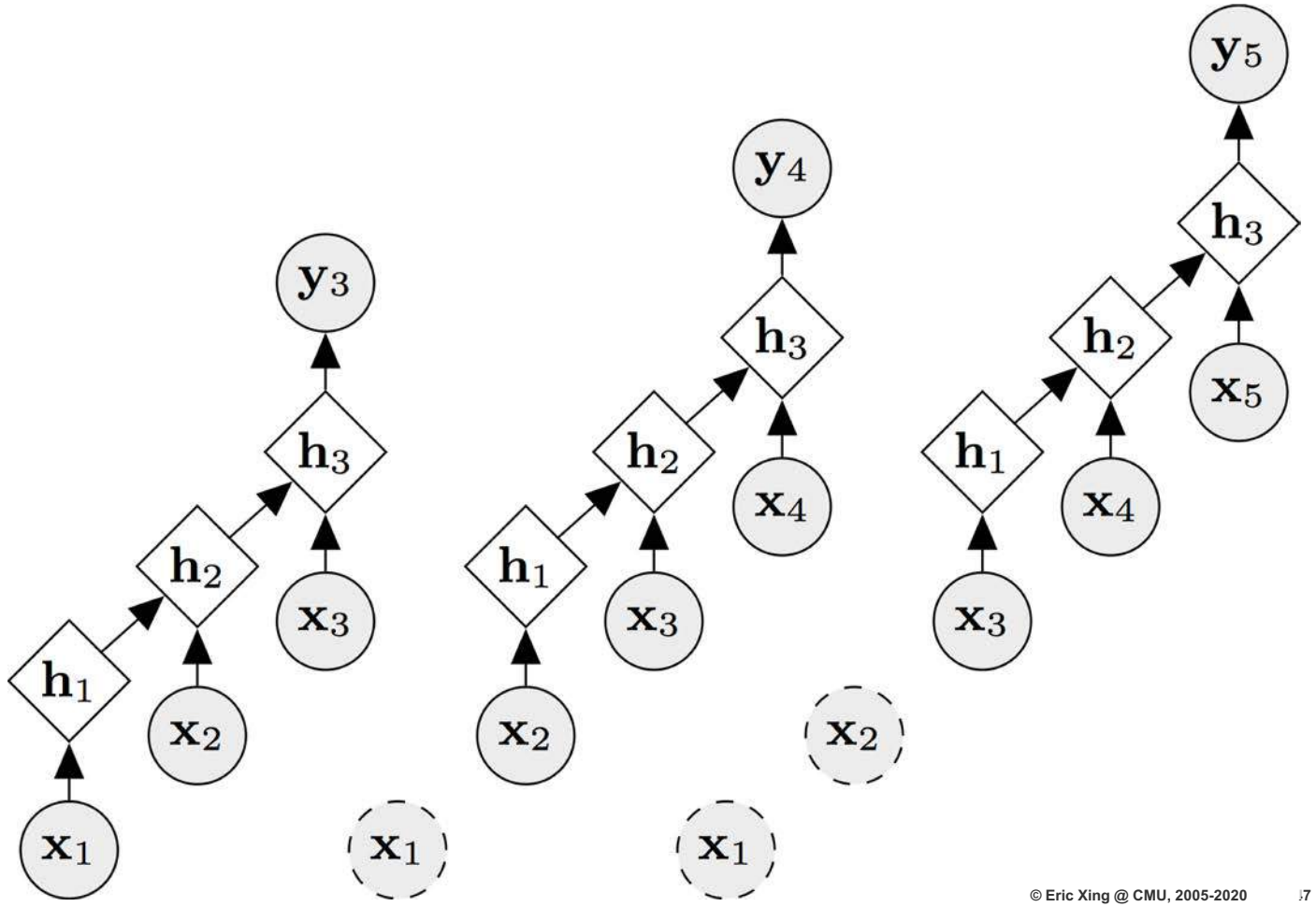
Datasets	n	d	RMSE					
			GP			DNN	DKL	
			RBF	SM	best		RBF	SM
Gas	2,565	128	0.21±0.07	0.14±0.08	0.12±0.07	0.11±0.05	0.11±0.05	**0.09±0.06**
Skillcraft	3,338	19	1.26±3.14	0.25±0.02	0.25±0.02	**0.25±0.00**	**0.25±0.00**	**0.25±0.00**
SML	4,137	26	6.94±0.51	0.27±0.03	0.26±0.04	0.25±0.02	0.24±0.01	**0.23±0.01**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

Wilson et al., NIPS 2016

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?

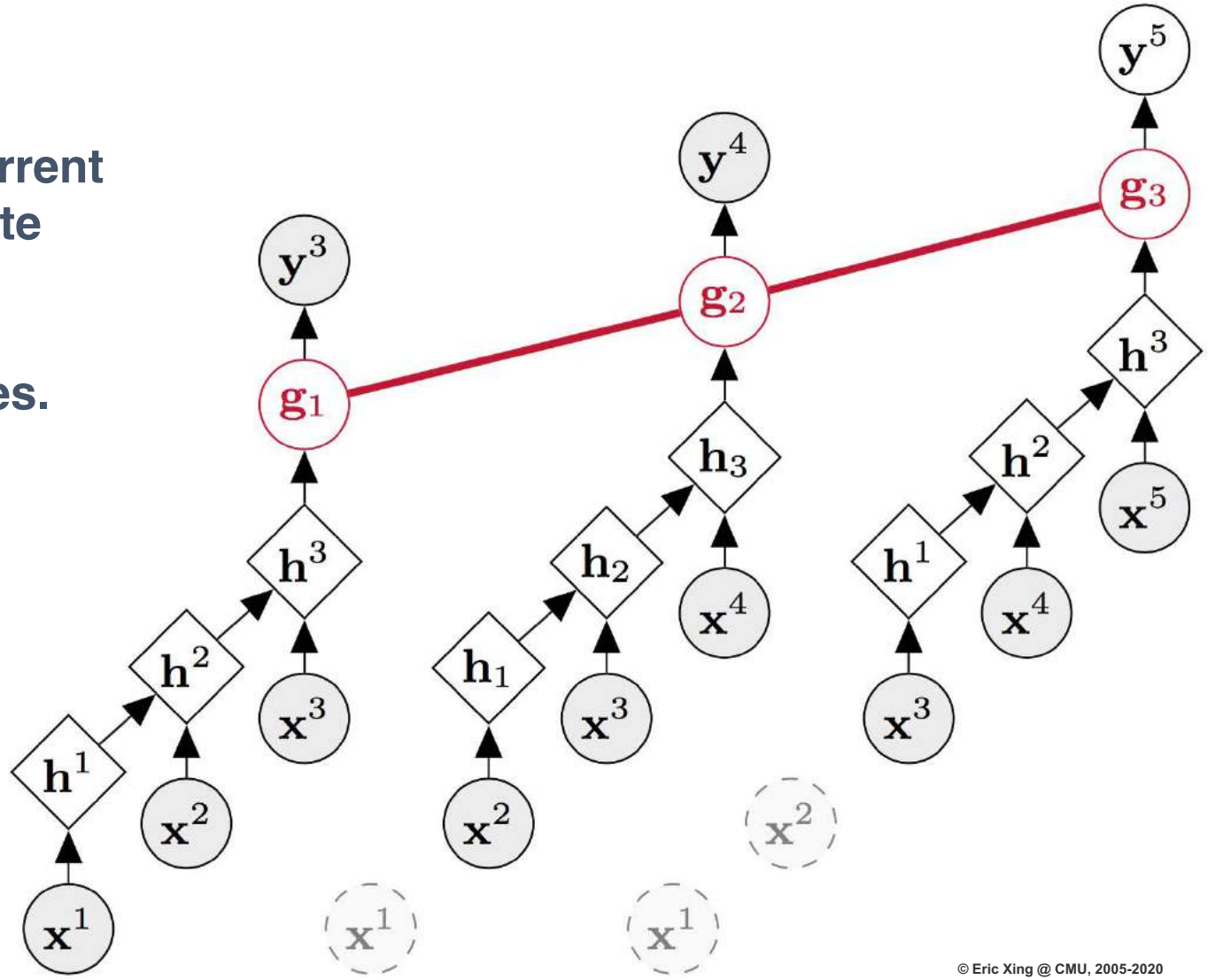


Al-Shedivat et al., JMLR 2017

Deep Kernel Learning on Sequential Data

The answer is YES!

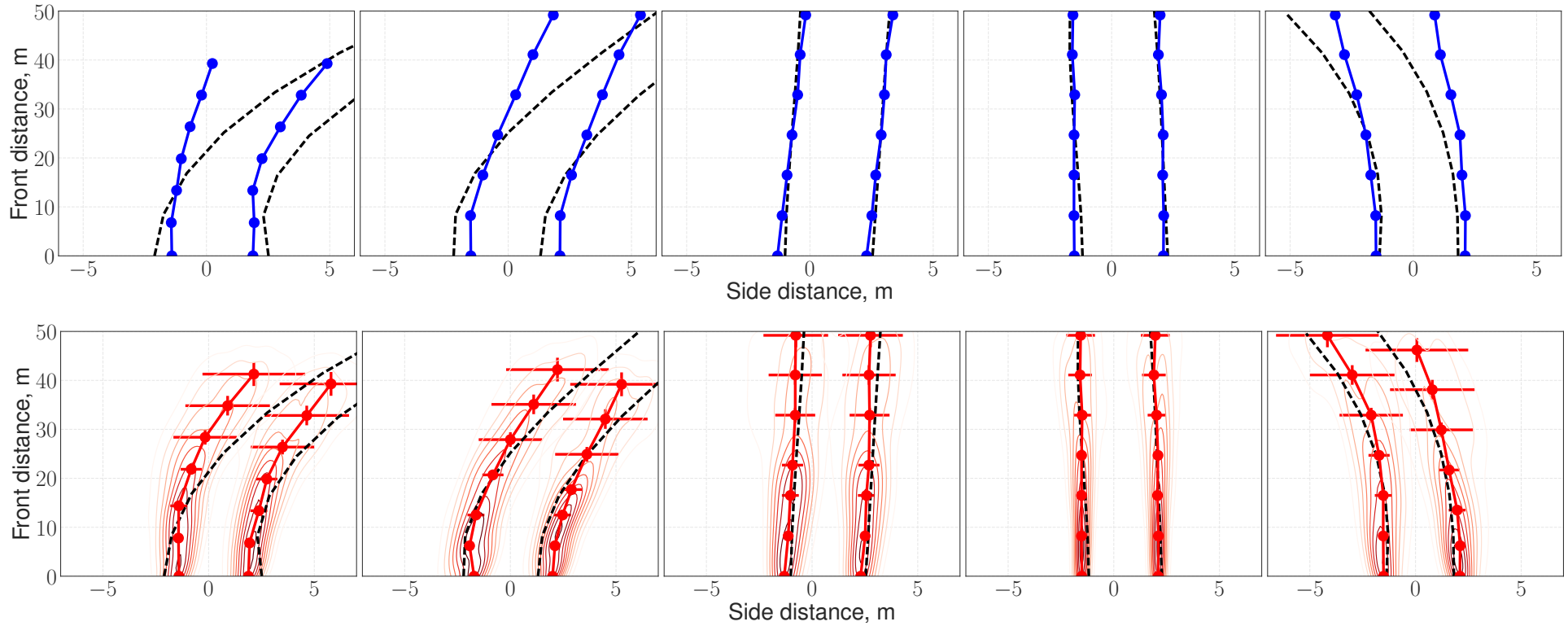
By adding a GP layer to a recurrent network, we effectively correlate samples across time and get predictions along with well calibrated uncertainty estimates.



Al-Shedivat et al., JMLR 2017

Deep Kernel Learning on Sequential Data

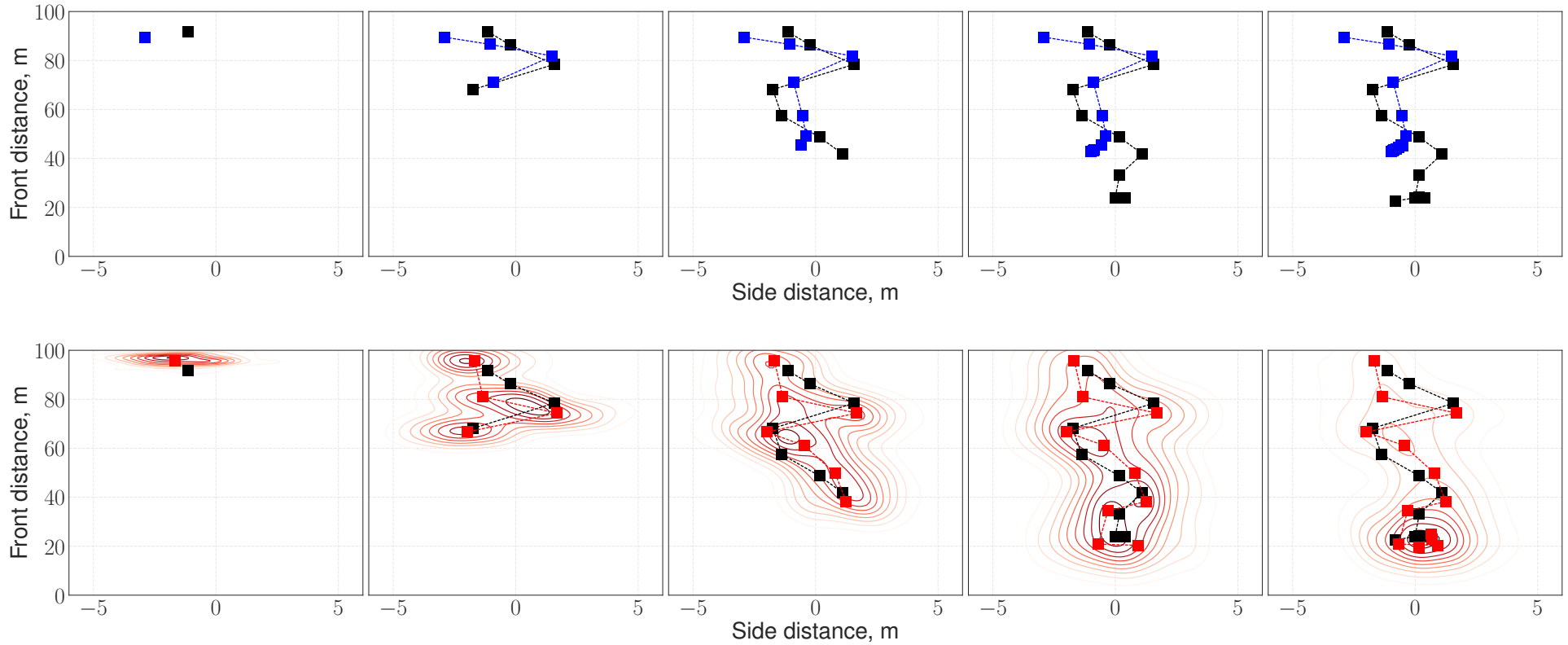
Lane prediction: LSTM vs GP-LSTM



Al-Shedivat et al., JMLR 2017

Deep Kernel Learning on Sequential Data

Lead vehicle prediction: LSTM vs GP-LSTM



Al-Shedivat et al., JMLR 2017

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 θ.
- 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.

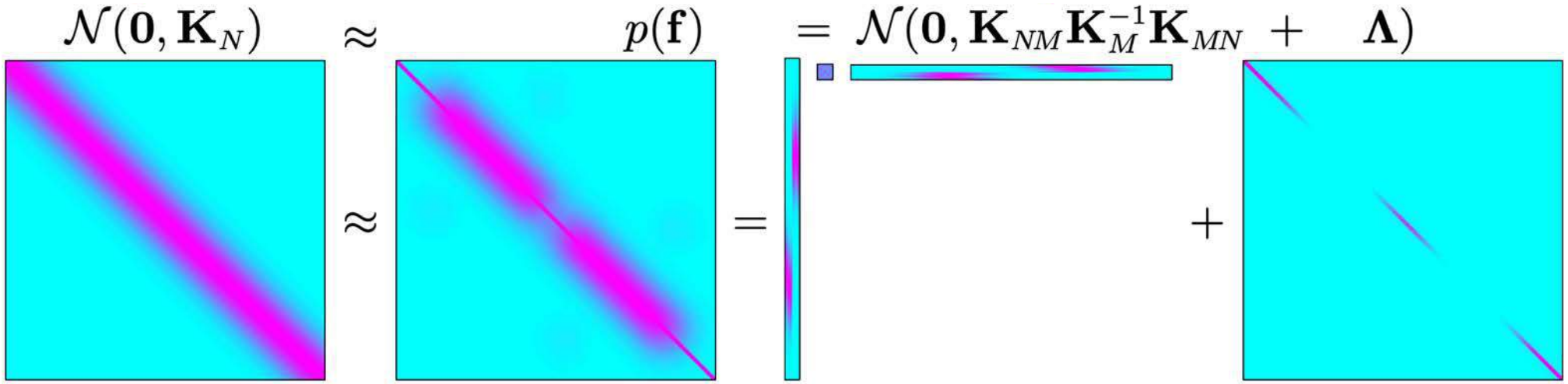
Scaling Up Gaussian Processes

Three Families of Approaches

- Approximate non-parametric kernels in a finite basis 'dual space'. Requires $\mathcal{O}(m^2 n)$ 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

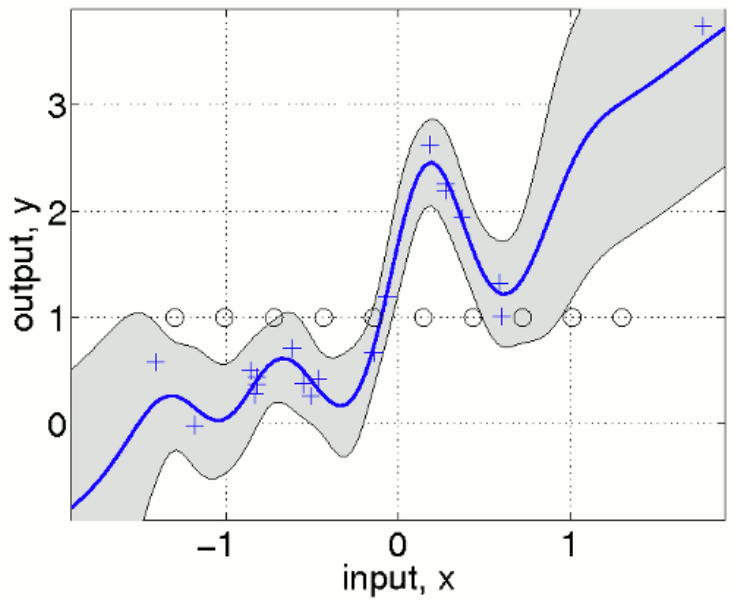
We can approximate GP through $M < N$ inducing points $\bar{\mathbf{f}}$ to obtain this Sparse Pseudo-input Gaussian process (SPGP) prior: $p(\mathbf{f}) = \int \mathrm{d}\bar{\mathbf{f}} \prod_n p(f_n|\bar{\mathbf{f}})\, p(\bar{\mathbf{f}})$

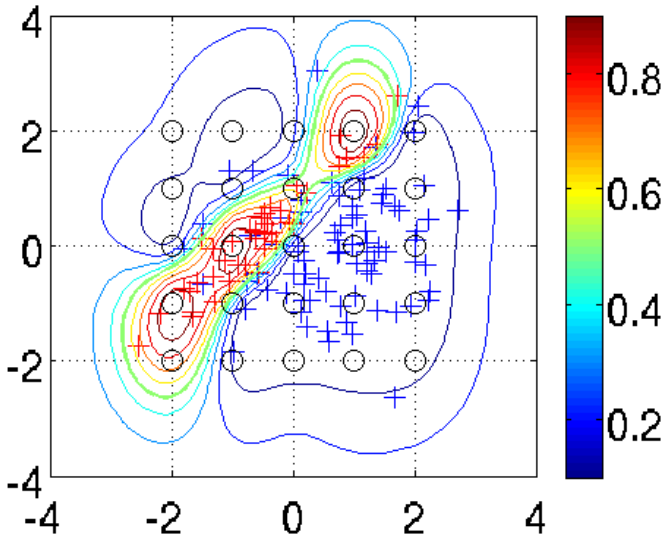


- SPGP covariance inverted in $\mathcal{O}(M^2N) \ll \mathcal{O}(N^3) \Rightarrow$ much faster

Slide credit: Zoubin Ghahramani (MLSS, 2011)

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.

Images: GPML library docs

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

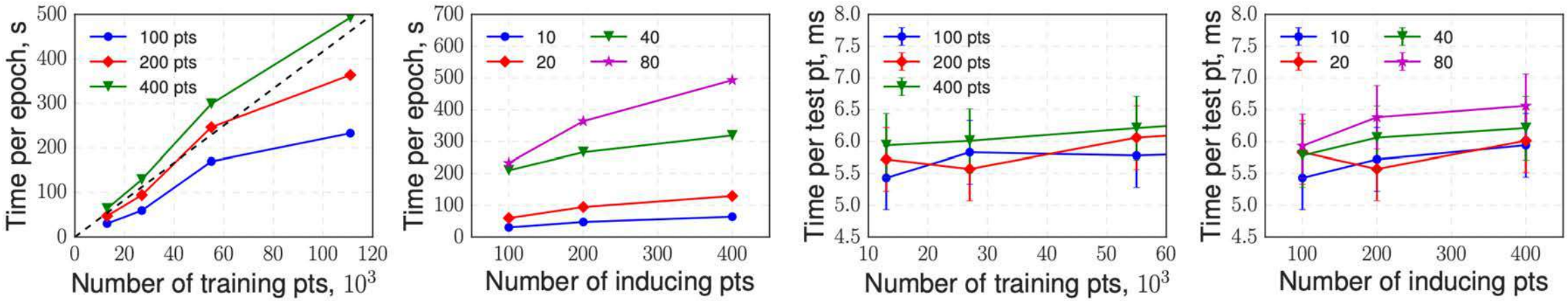
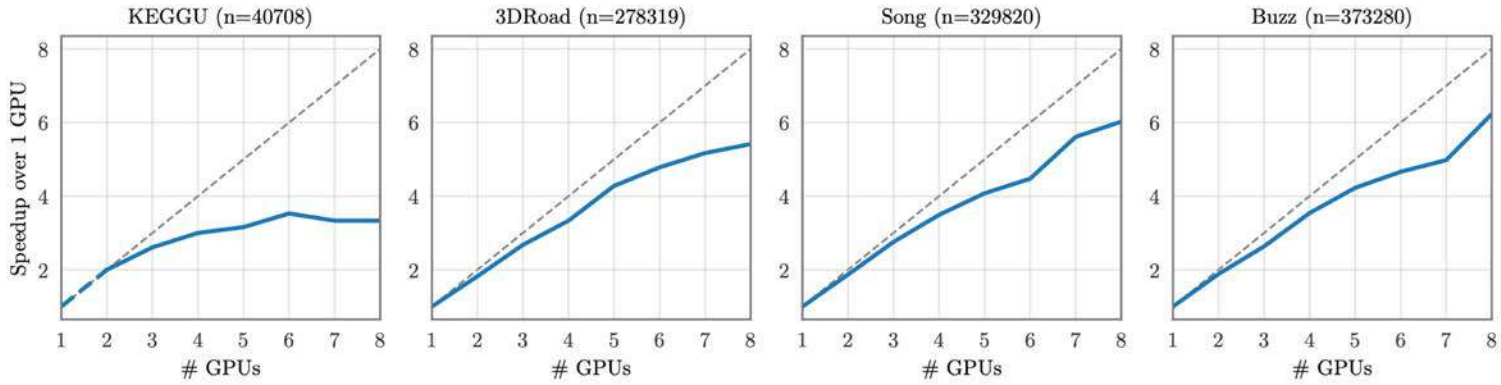


Image source: Al-Shedivat et al., JMLR 2017

Running Exact GPs on GPUs (recent)

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

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



Wang et al. (2019). Exact Gaussian Processes on a Million Data Points (arXiv:1903.08114)

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!)

alshedivat / keras-gp

Unwatch 10 | Unstar 165

Code | Issues 8 | Pull requests 1 | Projects 0 | Wiki | Insights | Settings

Keras + Gaussian Processes: Learning scalable deep and recurrent kernels.

keras | theano | tensorflow | gaussian-processes | neural-networks | machine-learning | Manage topics

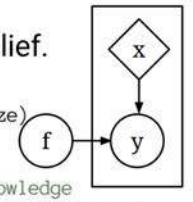
3) PyTorch-based

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)
kl = sum(model.losses)
loss = neg_log_likelihood + kl
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.

PETUUM

Carnegie Mellon University
School of Computer Science

Gaussian Process for Hyperparameter Tuning

Hyperparameter Tuning

- Existing methods
  - Grid search
  - Graduate student descent

- Problems
  - Time-consuming
  - Labor-intensive

Automatic Hyperparameter Tuning

- Generalization performance (e.g., error rate) is a **function** of hyperparameters.
- If knowing this function, we can perform optimization to search for the optimal hyperparameters yielding the lowest error.
- This function is a black-box and (almost surely) has no closed-form solutions.
- Solution: use a highly-expressive and easily-operable proxy function to approximate the true function and perform optimization on the proxy function.
- Family of proxy functions: Gaussian Process

Gaussian Process for Hyperparameter Tuning

- Obtain a set S of (hyperparameter-configuration, error) pairs using grid search or graduate student descent

- Repeat
  - Fit a Gaussian process on the (hyperparameter, error) pairs in S

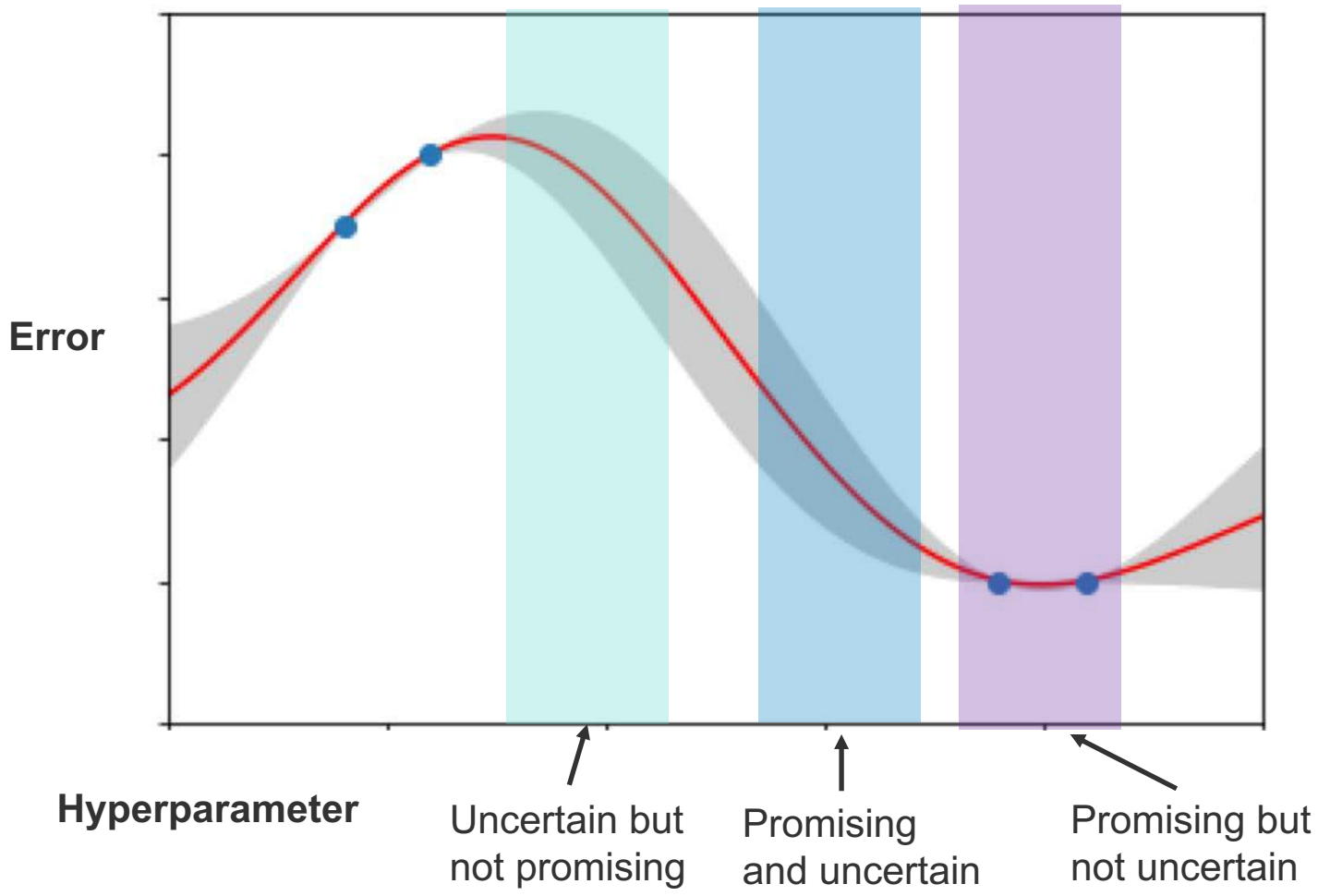
  - Based on the fitted Gaussian process, select a hyperparameter configuration H and measure the error E given H

  - Add the (H, E) pair to S

How to select hyperparameter configuration?

- Tradeoff between exploration and exploitation.
  - Exploitation: search over the “promising” hyperparameter space
    - The “promising” space is more likely to contain the best hyperparameters.
    - Hyperparameter space yielding lower GP function values is more promising.
  - Exploration: search over the entire hyperparameter space
    - The “promising” space may not contain the best hyperparameters.
    - Try other spaces as well
    - Space having more “uncertainty” is more worthwhile to try.

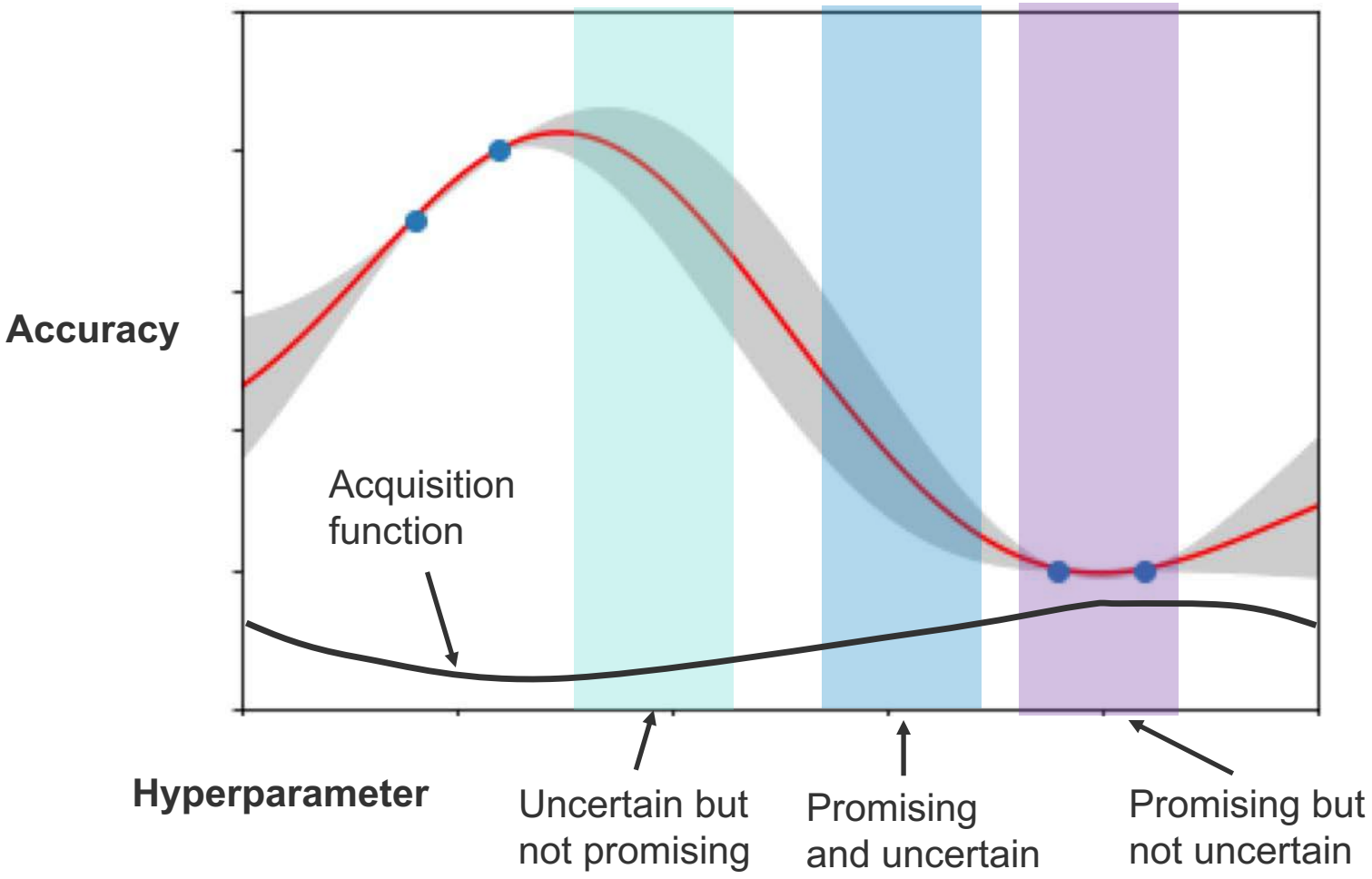
“Promising” and “Uncertain”



Promising: Hyperparameters yielding low GP **mean**

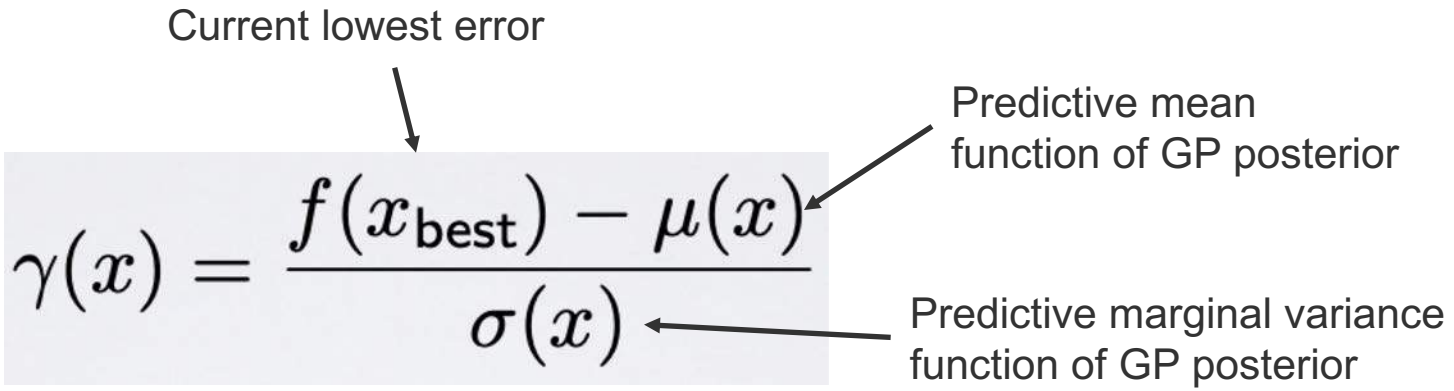
Uncertain: Hyperparameters yielding large GP **variance**

Acquisition Function



- Hyperparameters that are more promising and more uncertain have larger acquisition function value.
- Select the hyperparameter with the largest acquisition function value to try.

Define Acquisition Function



- Probability of Improvement (Kushner 1964):

$$a_{\text{PI}}(x) = \Phi(\gamma(x))$$

  - $\Phi(\cdot)$ is the cumulative density function of a normal distribution.

Define Acquisition Function (Cont'd)

- Expected Improvement (Mockus 1978):

$$a_{\mathsf{EI}}(x) = \sigma(x)(\gamma(x)\Phi(\gamma(x)) + \mathcal{N}(\gamma(x)\,;\,0,1))$$

- GP Upper Confidence Bound (Srinivas et al. 2010):

$$a_{\mathsf{LCB}}(x) = \mu(x) - \kappa\,\sigma(x)$$

Illustration

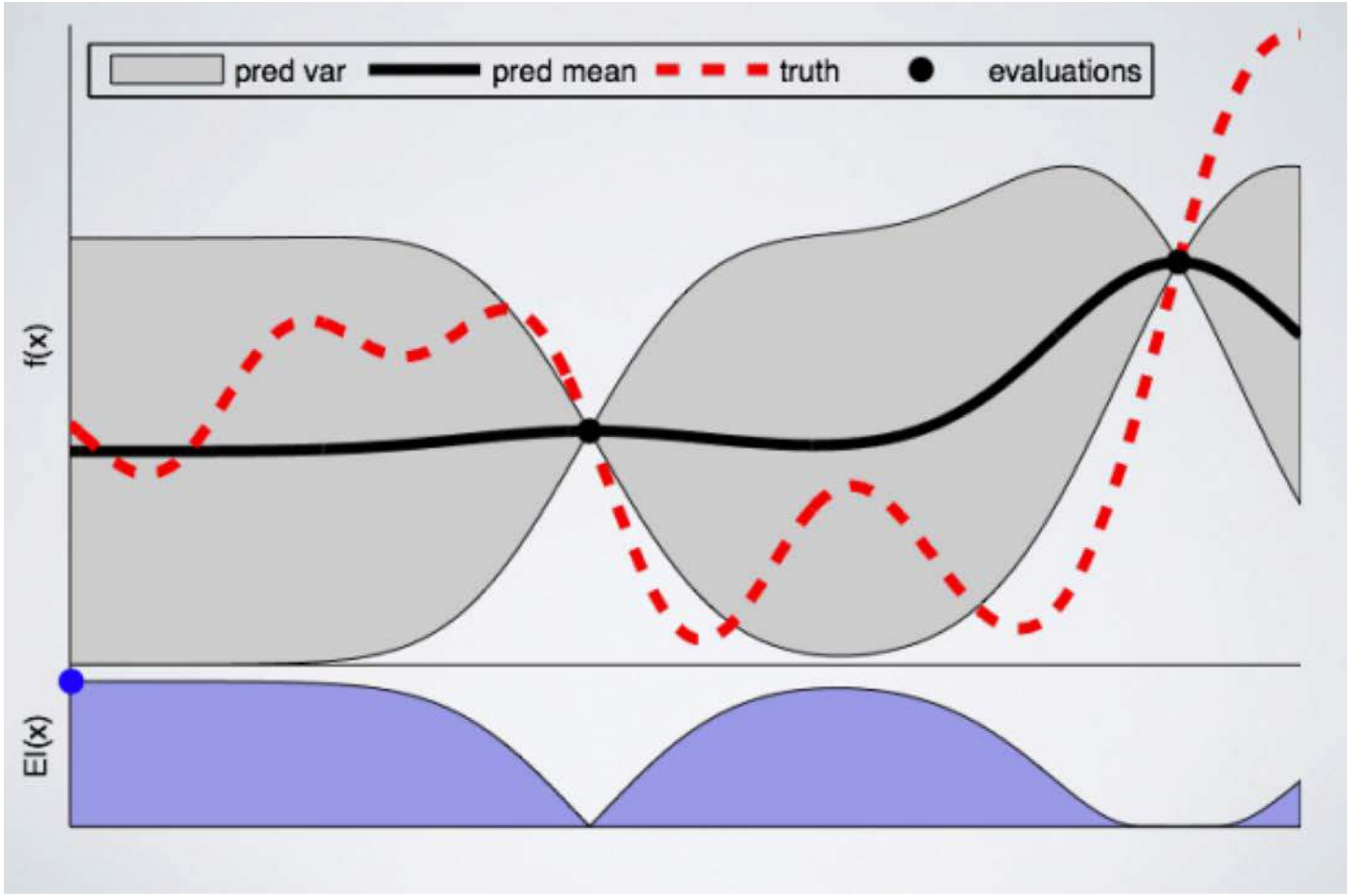


Figure Courtesy:
Ryan Adams

Illustration

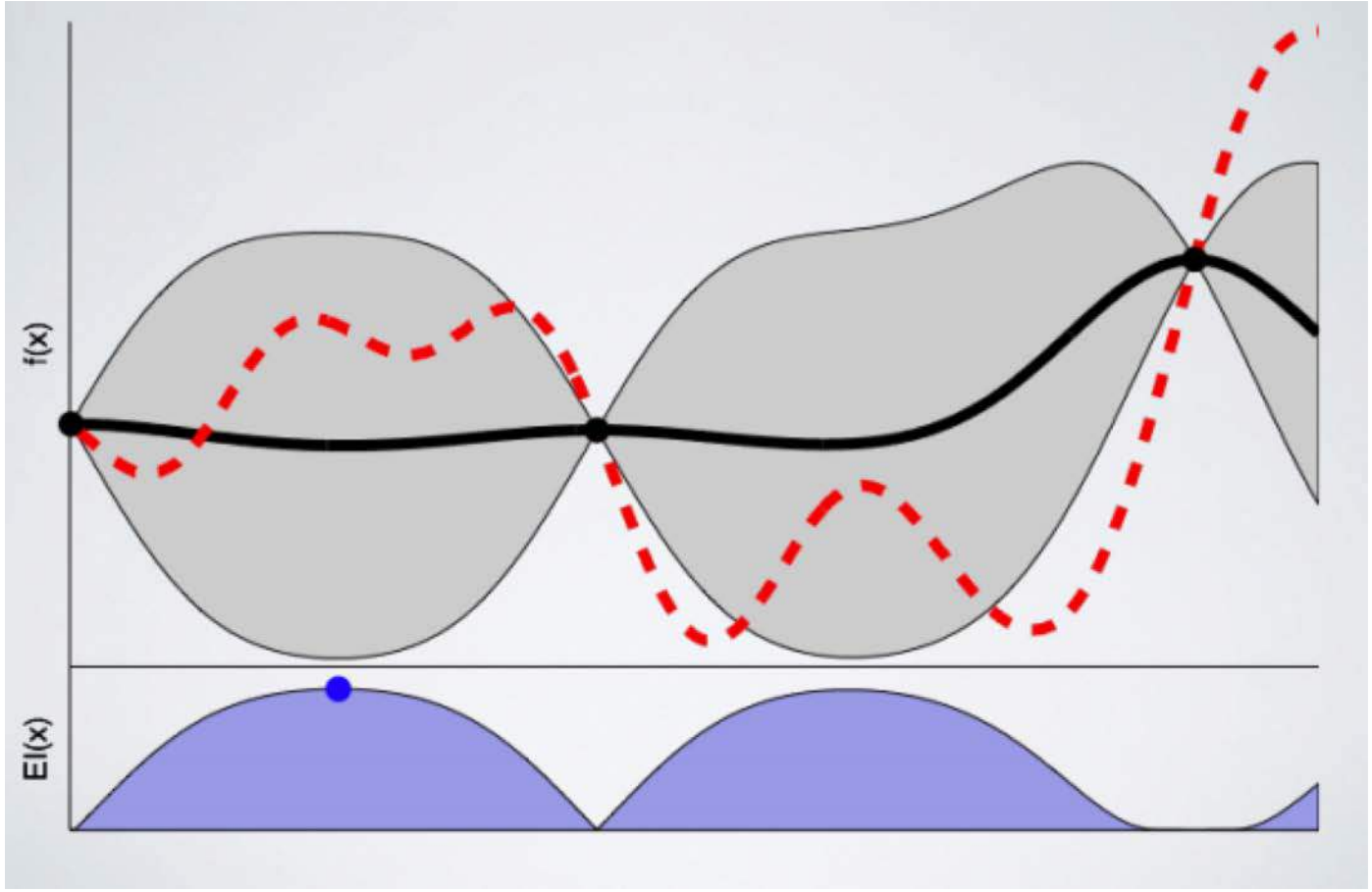


Figure Courtesy:
Ryan Adams

Illustration

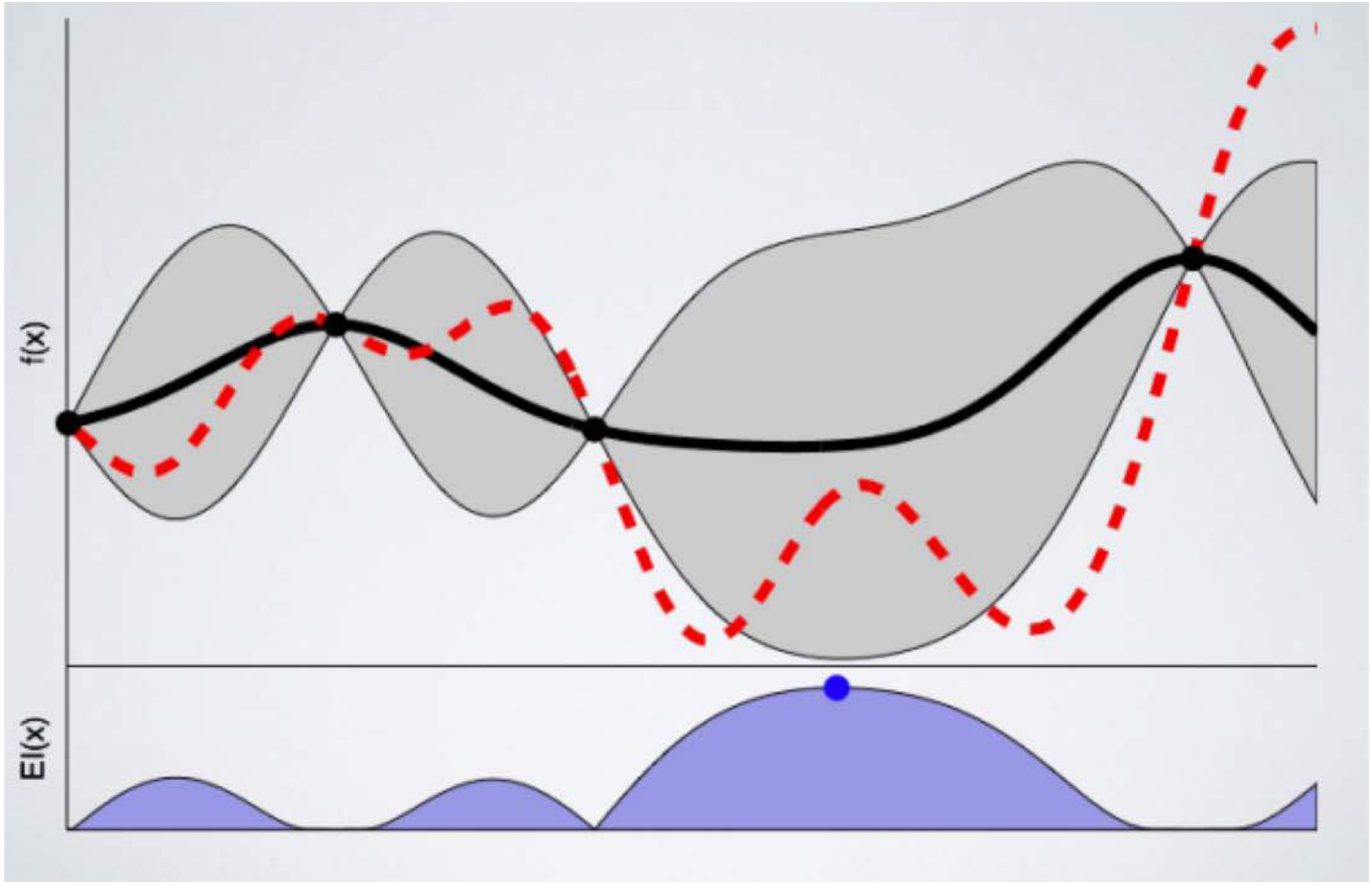


Figure Courtesy:
Ryan Adams

Illustration

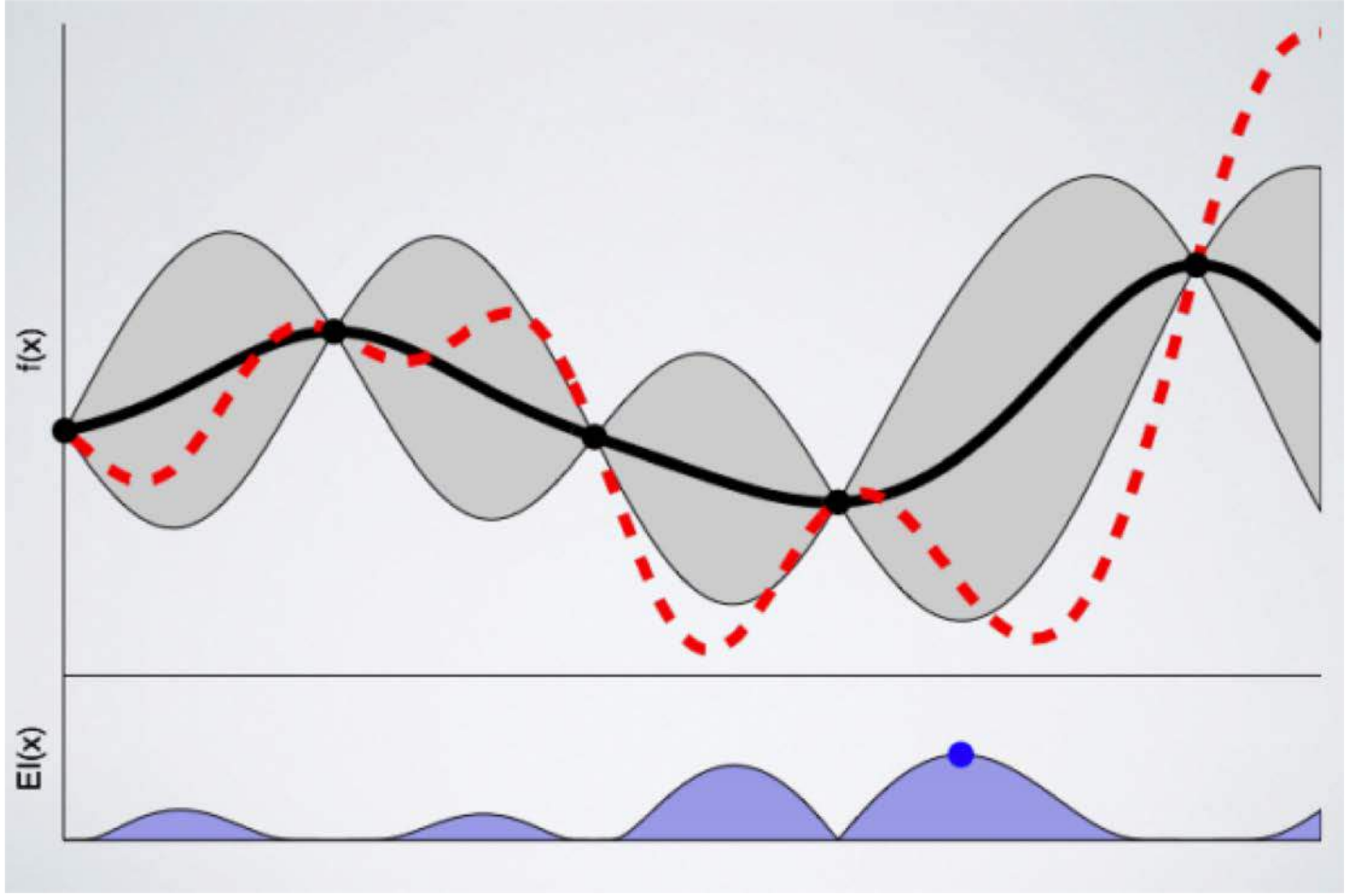


Figure Courtesy:
Ryan Adams

Illustration

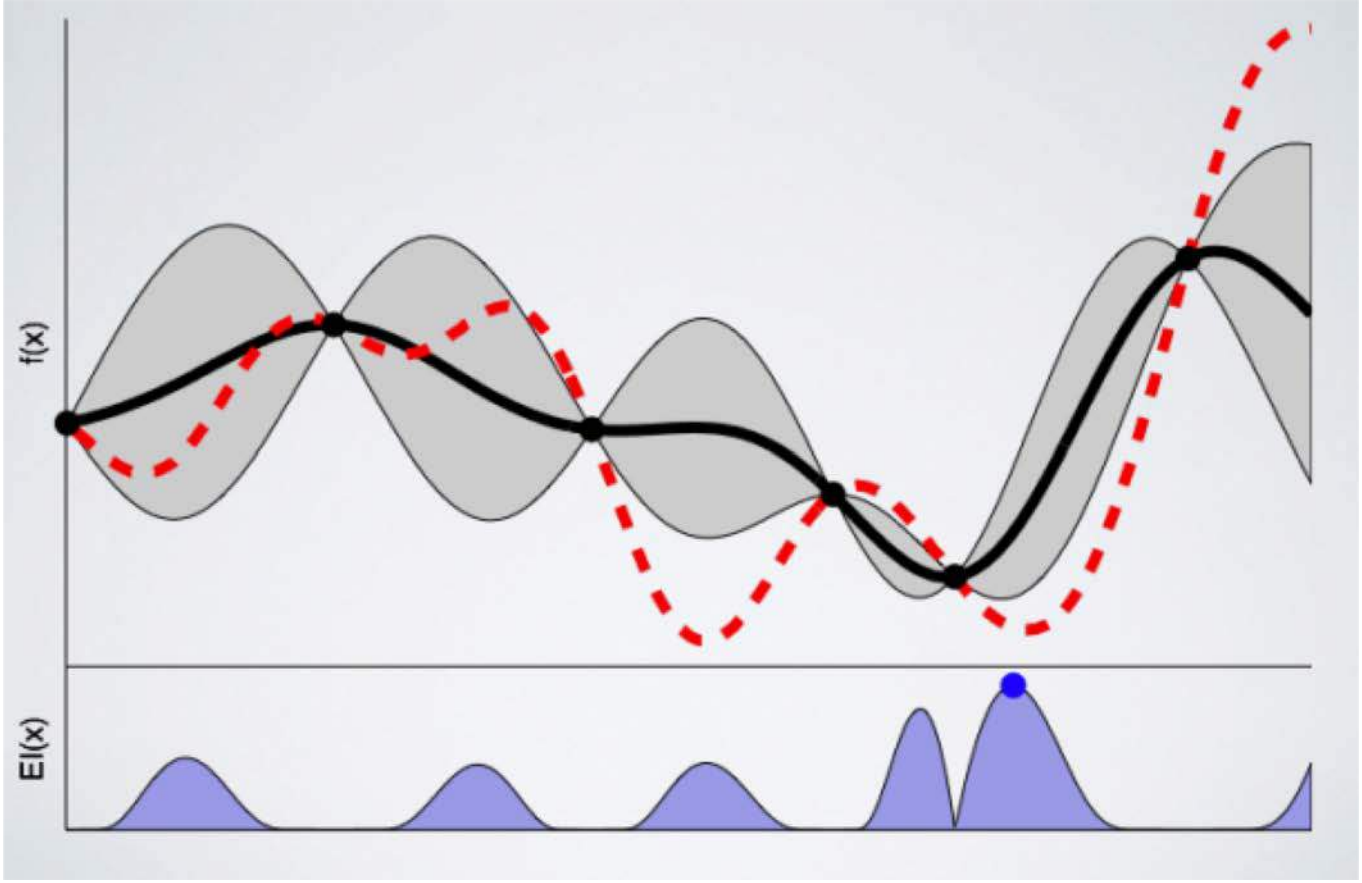


Figure Courtesy:
Ryan Adams

Illustration

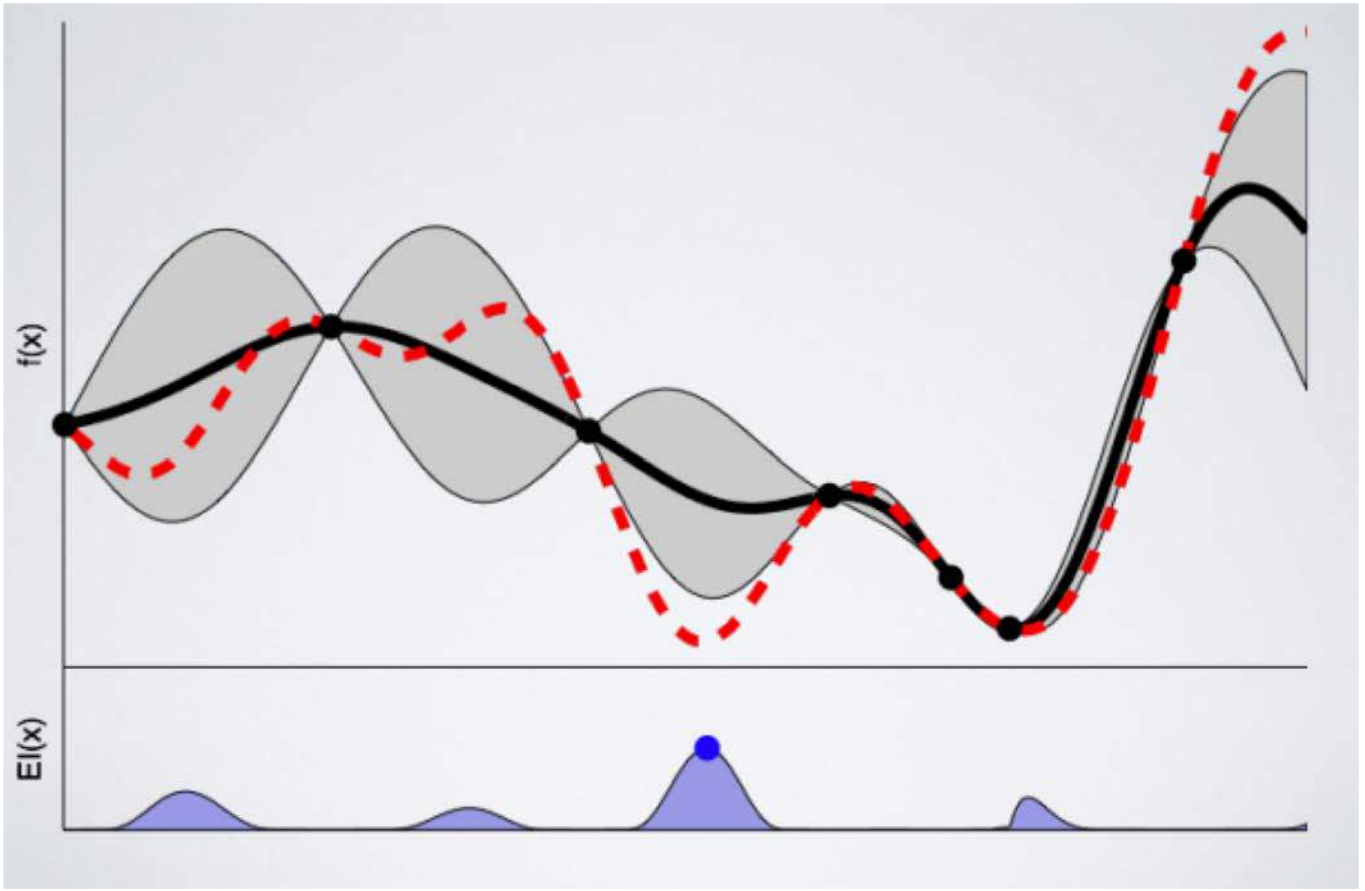


Figure Courtesy:
Ryan Adams

Illustration

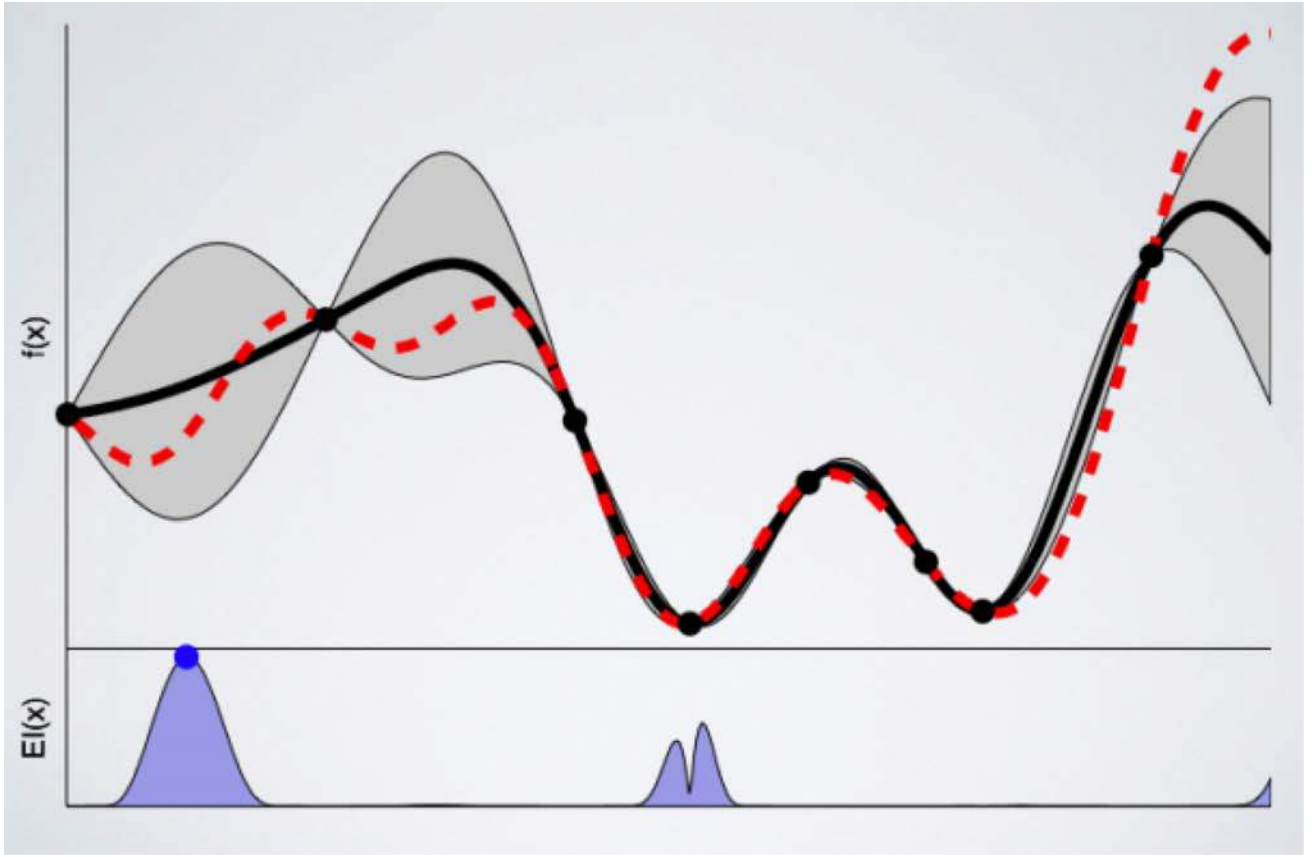


Figure Courtesy:
Ryan Adams

Summary

- Use GP to tune hyperparameters
- Iteratively fit GP to approximate the true hyperparameter-error function
- Select hyperparameters that have low GP mean and high GP variance to try
- Acquisition function simultaneously considers GP mean and variance.

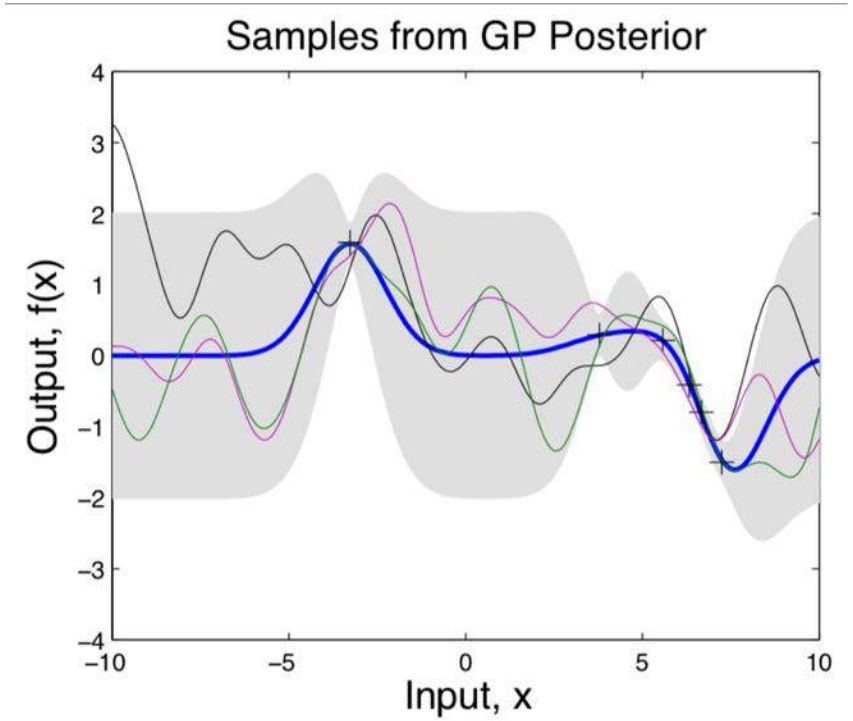
PETUUM

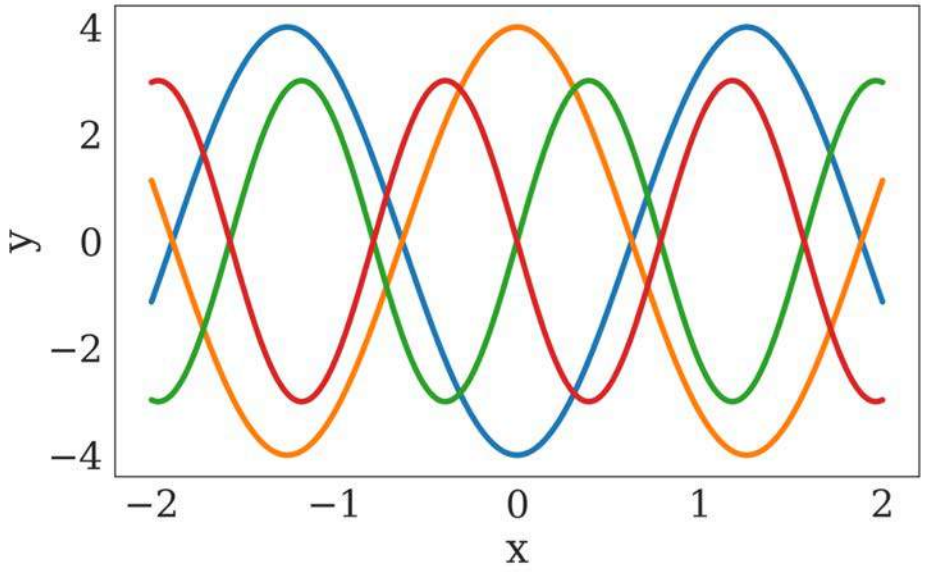
Carnegie Mellon University
School of Computer Science

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

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

- 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

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

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

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

distribution over examples for task T

Example: Few-shot Image Classification

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

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

Considered in:

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

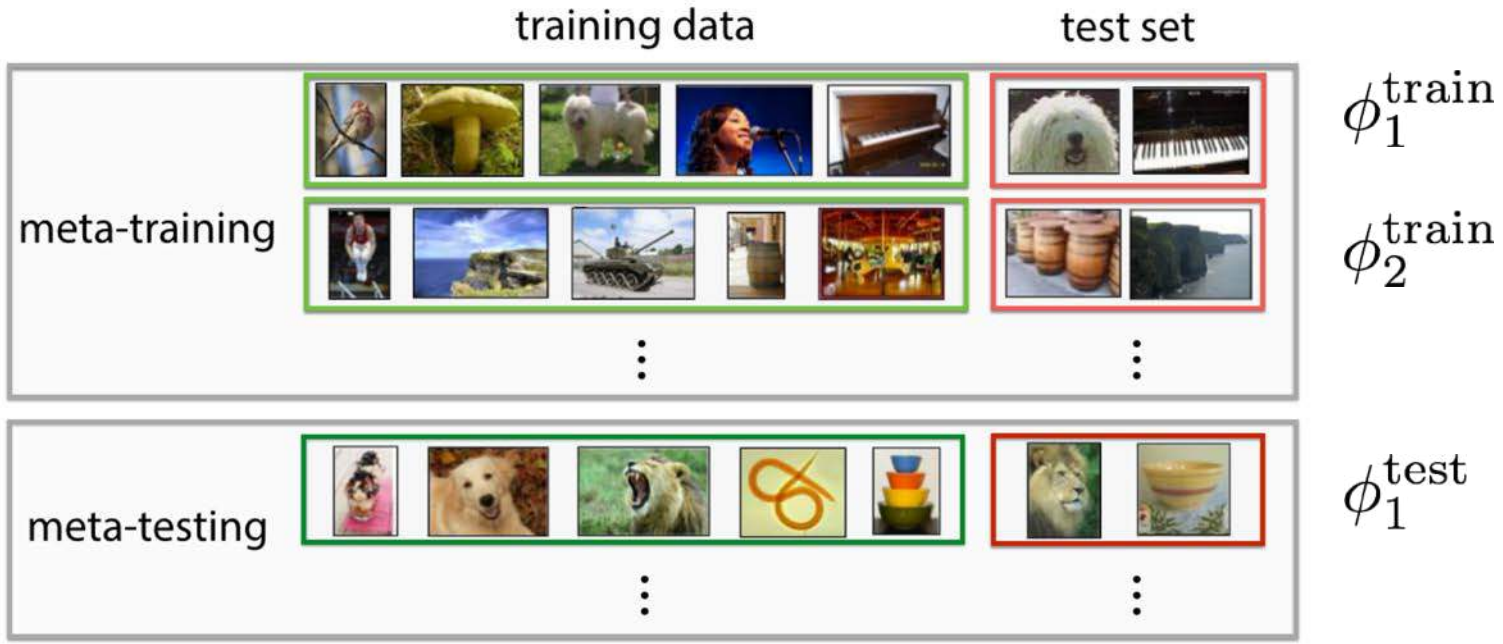
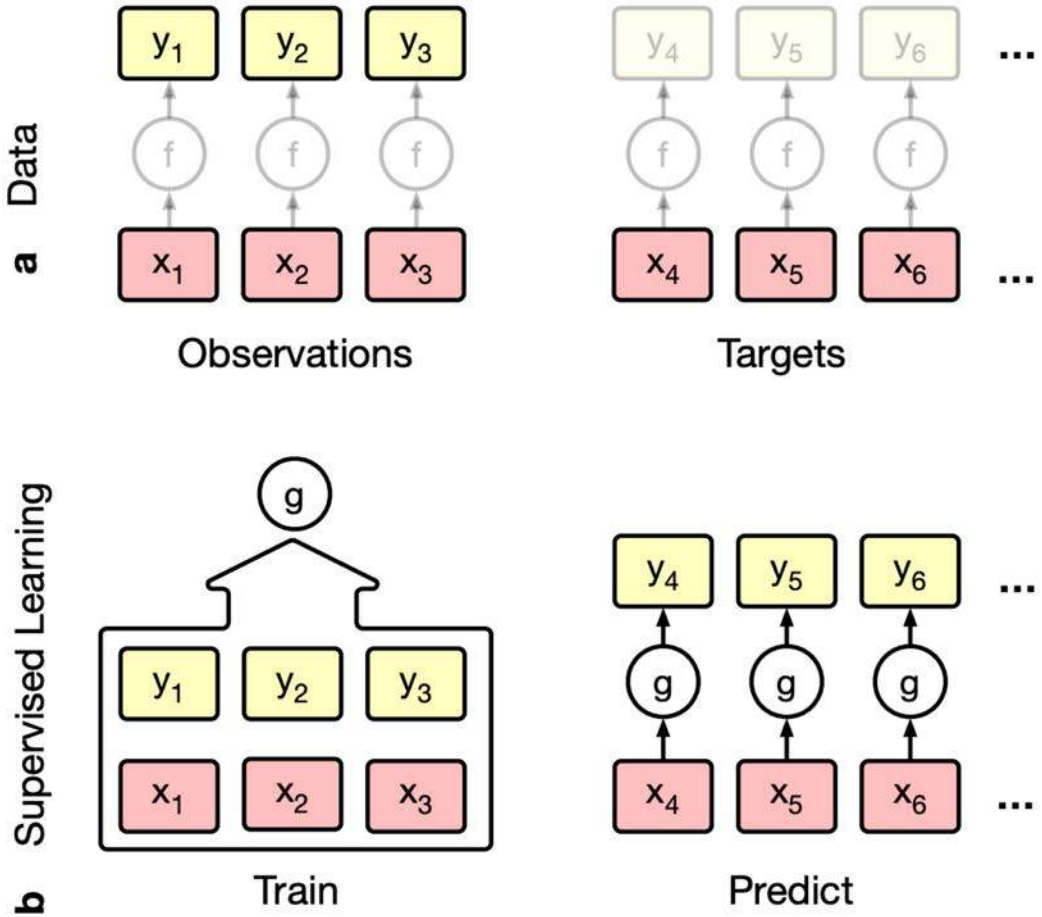
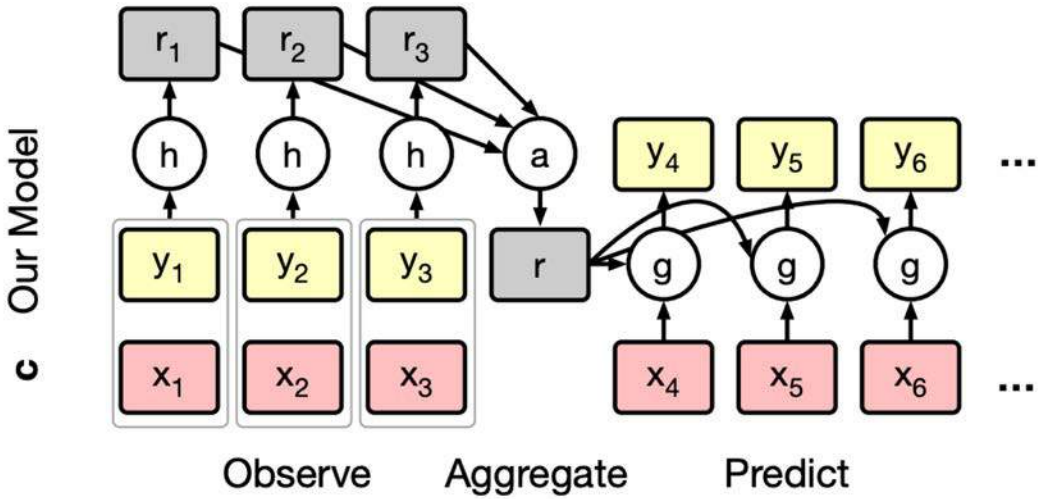


Image: bair.berkeley.edu

Conditional Neural Processes



CNP architecture:



Garnelo et al., ICML 2018

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)