Gaussian Processes (Contd)

Piyush Rai

Topics in Probabilistic Modeling and Inference (CS698X)

Jan 30, 2019

Announcement

- Quiz 1 tomorrow - Jan 31, 7pm-8pm
- Y14, Y15, Y18: RM-101
- Y16, Y17: KD-101
- Bring a pencil and eraser (answers to be written on the question paper itself)
- Do not bring anything else

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$p(y|f,\boldsymbol{x}) = \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1})$$

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$\begin{aligned} p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\ p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^y[1-\sigma(f(\boldsymbol{x}))]^{1-y} \end{aligned}$$

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$\begin{aligned} p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\ p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^y[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\ p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x})) \end{aligned}$$

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$\begin{aligned} p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\ p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^y[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\ p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x})) \end{aligned}$$

- Not just interested in a point estimate but the full posterior over f

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$
\begin{aligned}
p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\
p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^{y}[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\
p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x}))
\end{aligned}
$$

- Not just interested in a point estimate but the full posterior over f
- Usually done in one of the following ways

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$
\begin{aligned}
p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\
p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^y[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\
p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x}))
\end{aligned}
$$

- Not just interested in a point estimate but the full posterior over f
- Usually done in one of the following ways
  - Ad-hoc: Define nonlinear features $\phi(\boldsymbol{x})$ + train Bayesian linear model ($f(\boldsymbol{x}) = \boldsymbol{w}^\top\phi(\boldsymbol{x})$)

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$
\begin{aligned}
p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\
p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^y[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\
p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x}))
\end{aligned}
$$

- Not just interested in a point estimate but the full posterior over f
- Usually done in one of the following ways
  - Ad-hoc: Define nonlinear features $\phi(\boldsymbol{x})$ + train Bayesian linear model ($f(\boldsymbol{x}) = \boldsymbol{w}^\top\phi(\boldsymbol{x})$)
  - Ad-hoc: Train a neural net to extract features $\phi(\boldsymbol{x})$ + train Bayesian linear model

Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$
\begin{aligned}
p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\
p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^{y}[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\
p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x}))
\end{aligned}
$$

- Not just interested in a point estimate but the full posterior over f
- Usually done in one of the following ways
  - Ad-hoc: Define nonlinear features $\phi(\boldsymbol{x})$ + train Bayesian linear model ($f(\boldsymbol{x}) = \boldsymbol{w}^\top\phi(\boldsymbol{x})$)
  - Ad-hoc: Train a neural net to extract features $\phi(\boldsymbol{x})$ + train Bayesian linear model
  - Bayesian Neural Networks (infer posterior over NN weights; compute posterior predictive)

Recap: Bayesian Modeling of Nonlinear Functions

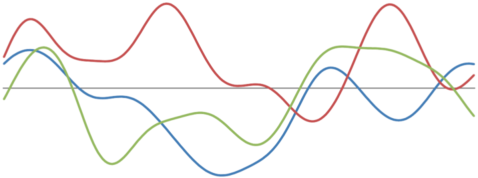
- Goal: Learn a nonlinear function f for discriminative models of the form $p(y|\boldsymbol{x})$, e.g.,

$$\begin{aligned} p(y|f,\boldsymbol{x}) &= \mathcal{N}(y|f(\boldsymbol{x}),\beta^{-1}) \\ p(y|f,\boldsymbol{x}) &= [\sigma(f(\boldsymbol{x}))]^{y}[1-\sigma(f(\boldsymbol{x}))]^{1-y} \\ p(y|f,\boldsymbol{x}) &= \text{ExpFam}(f(\boldsymbol{x})) \end{aligned}$$

- Not just interested in a point estimate but the full posterior over f
- Usually done in one of the following ways
  - Ad-hoc: Define nonlinear features $\phi(\boldsymbol{x})$ + train Bayesian linear model ($f(\boldsymbol{x}) = \boldsymbol{w}^\top\phi(\boldsymbol{x})$)
  - Ad-hoc: Train a neural net to extract features $\phi(\boldsymbol{x})$ + train Bayesian linear model
  - Bayesian Neural Networks (infer posterior over NN weights; compute posterior predictive)
  - Gaussian Processes (Bayesian modeling + kernels)

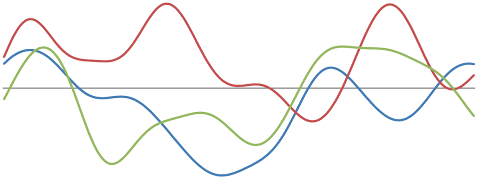
Recap: Gaussian Process

- A Gaussian Process is a distribution over functions
- Denoted as $\mathcal{GP}(\mu, \kappa)$; parametrized by a mean function μ and covariance/kernel function κ



Recap: Gaussian Process

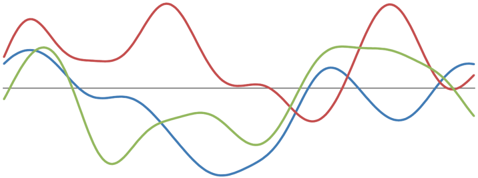
- A Gaussian Process is a distribution over functions
- Denoted as $\mathcal{GP}(\mu, \kappa)$; parametrized by a mean function μ and covariance/kernel function κ



- Mean function μ models the "average" function f from $\mathcal{GP}(\mu, \kappa)$: $\mu(\boldsymbol{x}) = \mathbb{E}_{f \sim \mathcal{GP}(\mu,\kappa)}[f(\boldsymbol{x})]$

Recap: Gaussian Process

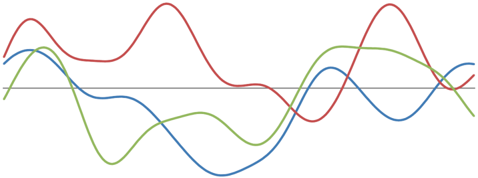
- A Gaussian Process is a distribution over functions
- Denoted as $\mathcal{GP}(\mu, \kappa)$; parametrized by a mean function μ and covariance/kernel function κ



- Mean function μ models the "average" function f from $\mathcal{GP}(\mu, \kappa)$: $\mu(\boldsymbol{x}) = \mathbb{E}_{f \sim \mathcal{GP}(\mu,\kappa)}[f(\boldsymbol{x})]$
- Cov. function κ models "shape/smoothness" of functions from this GP

Recap: Gaussian Process

- A Gaussian Process is a distribution over functions
- Denoted as $\mathcal{GP}(\mu, \kappa)$; parametrized by a mean function μ and covariance/kernel function κ



- Mean function μ models the "average" function f from $\mathcal{GP}(\mu, \kappa)$: $\mu(\boldsymbol{x}) = \mathbb{E}_{f \sim \mathcal{GP}(\mu,\kappa)}[f(\boldsymbol{x})]$
- Cov. function κ models "shape/smoothness" of functions from this GP
    - $\kappa(.,.)$ is a function that computes similarity between two inputs

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$
- Can use it to easily compute $f_* = f(\boldsymbol{x}_*)$ for a new input $\boldsymbol{x}_*$. To see this, note that for $\boldsymbol{\mu} = \mathbf{0}$

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*{}^\top & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right)$$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$
- Can use it to easily compute $f_* = f(\boldsymbol{x}_*)$ for a new input $\boldsymbol{x}_*$. To see this, note that for $\boldsymbol{\mu} = \mathbf{0}$

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*{}^\top & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right)$$

where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$
- Can use it to easily compute $f_* = f(\boldsymbol{x}_*)$ for a new input $\boldsymbol{x}_*$. To see this, note that for $\boldsymbol{\mu} = \mathbf{0}$

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^\top & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right)$$

where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$

- Can now apply the Gaussian conditioning to get $p(f_* | \boldsymbol{f}) = \mathcal{N}(\mu_*, \sigma_*^2)$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$
- Can use it to easily compute $f_* = f(\boldsymbol{x}_*)$ for a new input $\boldsymbol{x}_*$. To see this, note that for $\boldsymbol{\mu} = \mathbf{0}$

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ {\mathbf{k}_*}^\top & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right)$$

where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$

- Can now apply the Gaussian conditioning to get $p(f_* | \boldsymbol{f}) = \mathcal{N}(\mu_*, \sigma_*^2)$ where

$$\mu_* \quad = \quad {\mathbf{k}_*}^\top \mathbf{K}^{-1} \boldsymbol{f}$$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$
- Can use it to easily compute $f_* = f(\boldsymbol{x}_*)$ for a new input $\boldsymbol{x}_*$. To see this, note that for $\boldsymbol{\mu} = \mathbf{0}$

$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^\top & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right)$$

  where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$

- Can now apply the Gaussian conditioning to get $p(f_*|\boldsymbol{f}) = \mathcal{N}(\mu_*, \sigma_*^2)$ where

$$\mu_* = \mathbf{k}_*^\top \mathbf{K}^{-1} \boldsymbol{f} \quad (= \sum_{n=1}^N w_n f_n = \sum_{n=1}^N \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}_*))$$

Recap: Gaussian Process

- For $f \sim \mathcal{GP}(\mu, \kappa)$, f's values at any finite set of input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are jointly Gaussian

$$\begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}_1) \\ \mu(\boldsymbol{x}_2) \\ \vdots \\ \mu(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots \quad \ddots \quad \vdots \\ \kappa(\boldsymbol{x}_N, \boldsymbol{x}_1) \ldots \kappa(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \right)$$

- In a more compact notation, $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$, where $\boldsymbol{f}$ and $\boldsymbol{\mu}$ are $N \times 1$ and $\mathbf{K}$ is $N \times N$
- Can use it to easily compute $f_* = f(\boldsymbol{x}_*)$ for a new input $\boldsymbol{x}_*$. To see this, note that for $\boldsymbol{\mu} = \mathbf{0}$

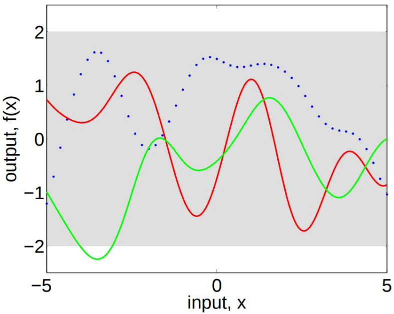
$$p\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{f} \\ f_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*{}^\top & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right)$$

  where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$

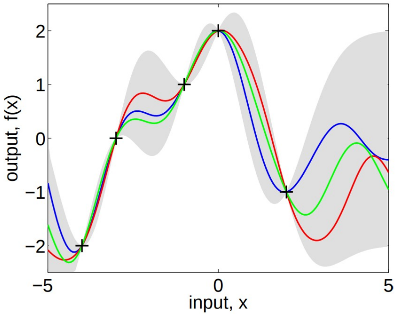
- Can now apply the Gaussian conditioning to get $p(f_* | \boldsymbol{f}) = \mathcal{N}(\mu_*, \sigma_*^2)$ where

$$\begin{aligned} \mu_* &= \mathbf{k}_*{}^\top \mathbf{K}^{-1} \boldsymbol{f} \quad \left(= \sum_{n=1}^N w_n f_n = \sum_{n=1}^N \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}_*)\right) \\ \sigma_*^2 &= \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) - \mathbf{k}_*{}^\top \mathbf{K}^{-1} \mathbf{k}_* \end{aligned}$$

GP: A Visualization



Some functions drawn from a GP prior (note: Blue dots are values of a randomly drawn function at a small number of inputs; the solid curves are generated by evaluating the functions at a large # of inputs)



Some functions drawn from the GP posterior after observing 5 (x,f(x)) pairs

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$p(y_n|f_n) = \mathcal{N}(y_n|f_n, \beta^{-1})$$

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$
\begin{aligned}
p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\
p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1 - \sigma(f_n)]^{1-y_n}
\end{aligned}
$$

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$\begin{aligned} p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\ p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1-\sigma(f_n)]^{1-y_n} \\ p(y_n|f_n) &= \text{ExpFam}(f_n) \end{aligned}$$

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$
\begin{aligned}
p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\
p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1 - \sigma(f_n)]^{1-y_n} \\
p(y_n|f_n) &= \text{ExpFam}(f_n)
\end{aligned}
$$

- Here making predictions for a new input $\boldsymbol{x}_*$ requires not $p(f_*|\boldsymbol{f})$ but $p(y_*|\boldsymbol{y})$

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_*$$

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$\begin{aligned} p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\ p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1 - \sigma(f_n)]^{1-y_n} \\ p(y_n|f_n) &= \text{ExpFam}(f_n) \end{aligned}$$

- Here making predictions for a new input $\boldsymbol{x}_*$ requires not $p(f_*|\boldsymbol{f})$ but $p(y_*|\boldsymbol{y})$

$$p(y_*|\mathbf{y}) = \int p(y_*|f_*)p(f_*|\mathbf{y})df_* = \int p(y_*|f_*){\color{red}p(f_*|\mathbf{f})}p(\mathbf{f}|\mathbf{y})d\boldsymbol{f}\,df_*$$

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$\begin{aligned} p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\ p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1 - \sigma(f_n)]^{1-y_n} \\ p(y_n|f_n) &= \text{ExpFam}(f_n) \end{aligned}$$

- Here making predictions for a new input $\boldsymbol{x}_*$ requires not $p(f_*|\boldsymbol{f})$ but $p(y_*|\boldsymbol{y})$

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}df_*$$

- For the above, $p(y_*|f_*)$ and $p(\mathbf{f}|\boldsymbol{y}) \propto p(\boldsymbol{f})p(\boldsymbol{y}|\boldsymbol{f})$ will depend on likelihood model $p(y_n|f_n)$

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$\begin{aligned} p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\ p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1-\sigma(f_n)]^{1-y_n} \\ p(y_n|f_n) &= \text{ExpFam}(f_n) \end{aligned}$$

- Here making predictions for a new input $\boldsymbol{x}_*$ requires not $p(f_*|\boldsymbol{f})$ but $p(y_*|\boldsymbol{y})$

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)\color{red}{p(f_*|\mathbf{f})}p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

- For the above, $p(y_*|f_*)$ and $p(\mathbf{f}|\boldsymbol{y}) \propto p(\boldsymbol{f})p(\boldsymbol{y}|\boldsymbol{f})$ will depend on likelihood model $p(y_n|f_n)$
- However $p(f_*|\mathbf{f})$ will be the same as in the noiseless setting (i.e., a Gaussian as we saw) :-)

GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs y_n that are "noisy" versions of $f_n = f(\boldsymbol{x}_n)$, e.g.,

$$\begin{aligned} p(y_n|f_n) &= \mathcal{N}(y_n|f_n, \beta^{-1}) \\ p(y_n|f_n) &= [\sigma(f_n)]^{y_n}[1-\sigma(f_n)]^{1-y_n} \\ p(y_n|f_n) &= \text{ExpFam}(f_n) \end{aligned}$$

- Here making predictions for a new input $\boldsymbol{x}_*$ requires not $p(f_*|\boldsymbol{f})$ but $p(y_*|\boldsymbol{y})$

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

- For the above, $p(y_*|f_*)$ and $p(\mathbf{f}|\boldsymbol{y}) \propto p(\boldsymbol{f})p(\boldsymbol{y}|\boldsymbol{f})$ will depend on likelihood model $p(y_n|f_n)$
- However $p(f_*|\mathbf{f})$ will be the same as in the noiseless setting (i.e., a Gaussian as we saw) :-)
- Note: For GP Regression (with Gaussian noise), $p(y_*|\boldsymbol{y})$ is very easily computable!

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

where all the 3 distributions in the integrand are Gaussians in case of GP regression!

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

  where all the 3 distributions in the integrand are Gaussians in case of GP regression!
- Therefore it is an easy to compute integral!

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

  where all the 3 distributions in the integrand are Gaussians in case of GP regression!
- Therefore it is an easy to compute integral!
- However, we cancompute $p(y_*|\boldsymbol{y})$ even without using the above method

GP Regression

- The likelihood model: $p(\boldsymbol{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

  where all the 3 distributions in the integrand are Gaussians in case of GP regression!
- Therefore it is an easy to compute integral!
- However, we cancompute $p(y_*|\boldsymbol{y})$ even without using the above method
- Reason: The marginal distribution of the training data responses $\boldsymbol{y}$

$$p(\boldsymbol{y}) = \int p(\boldsymbol{y}|\mathbf{f})p(\mathbf{f})d\mathbf{f} = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \mathbf{K} + \sigma^2\mathbf{I}_N) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \mathbf{C}_N)$$

GP Regression

- The likelihood model: $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2\mathbf{I}_N)$. The prior distribution: $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \mathbf{X})$ or $p(y_*|\boldsymbol{y})$ (skipping $\mathbf{X}, \boldsymbol{x}_*$ from the notation) will be

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

  where all the 3 distributions in the integrand are Gaussians in case of GP regression!
- Therefore it is an easy to compute integral!
- However, we cancompute $p(y_*|\boldsymbol{y})$ even without using the above method
- Reason: The marginal distribution of the training data responses $\boldsymbol{y}$

$$p(\boldsymbol{y}) = \int p(\boldsymbol{y}|\mathbf{f})p(\mathbf{f})d\mathbf{f} = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \mathbf{K} + \sigma^2\mathbf{I}_N) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \mathbf{C}_N)$$

- Using the same result, the marginal distribution $p(y_*) = \mathcal{N}(y_*|0, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2)$

GP Regression: Making Predictions

- Let's consider the joint distr. of N training responses $\boldsymbol{y}$ and test response y_*

$$p\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_* \\ \mathbf{k}_*^\top & c \end{bmatrix}\right)$$

where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$, $c = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2$

GP Regression: Making Predictions

- Let's consider the joint distr. of N training responses $\boldsymbol{y}$ and test response y_*

$$p\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_* \\ \mathbf{k}_*^\top & c \end{bmatrix}\right)$$

where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$, $c = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2$

- The desired predictive posterior will be (using conditional from joint property of Gaussian)

$$p(y_*|\boldsymbol{y}) = \mathcal{N}(y_*|\mu_*, \sigma_*^2)$$

GP Regression: Making Predictions

- Let's consider the joint distr. of N training responses $\boldsymbol{y}$ and test response y_*

$$p\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_* \\ \mathbf{k}_*^\top & c \end{bmatrix}\right)$$

  where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$, $c = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2$

- The desired predictive posterior will be (using conditional from joint property of Gaussian)

$$\begin{aligned} p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\ \mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1} \boldsymbol{y} \end{aligned}$$

GP Regression: Making Predictions

- Let's consider the joint distr. of N training responses $\boldsymbol{y}$ and test response y_*

$$p\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_* \\ \mathbf{k}_*^\top & c \end{bmatrix}\right)$$

  where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$, $c = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2$

- The desired predictive posterior will be (using conditional from joint property of Gaussian)

$$\begin{aligned} p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\ \mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1} \boldsymbol{y} \\ \sigma_*^2 &= \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1} \mathbf{k}_* \end{aligned}$$

GP Regression: Making Predictions

- Let's consider the joint distr. of N training responses $\mathbf{y}$ and test response y_*

$$p\left(\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_* \\ \mathbf{k}_*^\top & c \end{bmatrix}\right)$$

where $\mathbf{k}_* = [\kappa(\mathbf{x}_*, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}_*, \mathbf{x}_N)]^\top$, $c = \kappa(\mathbf{x}_*, \mathbf{x}_*) + \sigma^2$

- The desired predictive posterior will be (using conditional from joint property of Gaussian)

$$\begin{aligned} p(y_*|\mathbf{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\ \mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{y} \\ \sigma_*^2 &= \kappa(\mathbf{x}_*, \mathbf{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{k}_* \end{aligned}$$

- Note that this is almost identical to the noiseless case (with σ^2 added to the predictive variance)

GP Regression: Making Predictions

- Let's consider the joint distr. of N training responses $\boldsymbol{y}$ and test response y_*

$$p\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k}_* \\ \mathbf{k}_*{}^\top & c \end{bmatrix}\right)$$

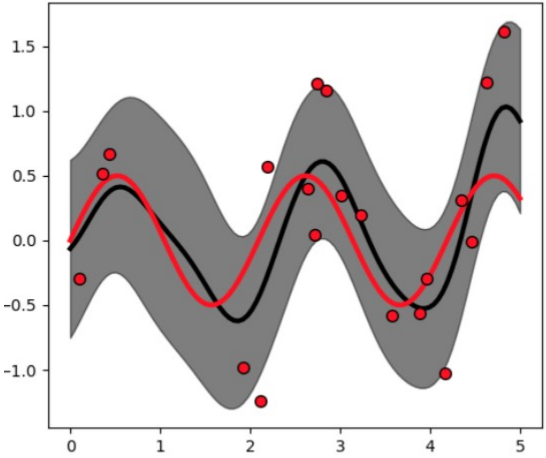
  where $\mathbf{k}_* = [\kappa(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_N)]^\top$, $c = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2$

- The desired predictive posterior will be (using conditional from joint property of Gaussian)

$$\begin{aligned} p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\ \mu_* &= \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \boldsymbol{y} \\ \sigma_*^2 &= \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \mathbf{k}_* \end{aligned}$$

- Note that this is almost identical to the noiseless case (with σ^2 added to the predictive variance)
- Can interpret predictive mean μ_* as kernelized SVM or nearest neighbor based prediction

GP Regression: An Illustration



Red curve: True function
Red points: Noisy training examples
Black curve: Predictive mean
Shaded part: Predictive variance

GP Regression: Learning Hyperparameters

- There are two hyperparameters in the GP regression model

GP Regression: Learning Hyperparameters

- There are two hyperparameters in the GP regression model
  - Variance of the Gaussian noise σ^2

GP Regression: Learning Hyperparameters

- There are two hyperparameters in the GP regression model
  - Variance of the Gaussian noise σ^2
  - Assuming $\mu = 0$, the hyperparameters θ of the covariance/kernel function κ, e.g.,

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp\left(-\frac{||\boldsymbol{x}_n - \boldsymbol{x}_m||^2}{\gamma}\right) \quad \text{(RBF kernel)}$$

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp\left(-\sum_{d=1}^{D}\frac{(\boldsymbol{x}_{nd} - \boldsymbol{x}_{md})^2}{\gamma_d}\right) \quad \text{(ARD kernel)}$$

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) = \kappa_{\theta_1}(\boldsymbol{x}_n, \boldsymbol{x}_m) + \kappa_{\theta_2}(\boldsymbol{x}_n, \boldsymbol{x}_m) + \ldots + \kappa_{\theta_M}(\boldsymbol{x}_n, \boldsymbol{x}_m) \quad \text{(flexible composition of multiple kernels)}$$

GP Regression: Learning Hyperparameters

- There are two hyperparameters in the GP regression model
  - Variance of the Gaussian noise σ^2
  - Assuming $\mu = 0$, the hyperparameters θ of the covariance/kernel function κ, e.g.,

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp\left(-\frac{||\boldsymbol{x}_n - \boldsymbol{x}_m||^2}{\gamma}\right) \quad \text{(RBF kernel)}$$

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp\left(-\sum_{d=1}^{D}\frac{(\boldsymbol{x}_{nd} - \boldsymbol{x}_{md})^2}{\gamma_d}\right) \quad \text{(ARD kernel)}$$

$$\kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) = \kappa_{\theta_1}(\boldsymbol{x}_n, \boldsymbol{x}_m) + \kappa_{\theta_2}(\boldsymbol{x}_n, \boldsymbol{x}_m) + \ldots + \kappa_{\theta_M}(\boldsymbol{x}_n, \boldsymbol{x}_m) \quad \text{(flexible composition of multiple kernels)}$$

- Type-II MLE is a popular choice for learning these hyperparams, by maximizing marginal likelihood

$$p(\boldsymbol{y}|\sigma^2, \theta) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \sigma^2\mathbf{I}_N + \mathbf{K}_\theta)$$

GP Regression: Learning Hyperparameters

- There are two hyperparameters in the GP regression model
  - Variance of the Gaussian noise σ^2
  - Assuming $\mu = 0$, the hyperparameters θ of the covariance/kernel function κ, e.g.,

$$\begin{aligned}
\kappa(\mathbf{x}_n, \mathbf{x}_m) &= \exp\left(-\frac{||\mathbf{x}_n - \mathbf{x}_m||^2}{\gamma}\right) && \text{(RBF kernel)} \\
\kappa(\mathbf{x}_n, \mathbf{x}_m) &= \exp\left(-\sum_{d=1}^{D}\frac{(\mathbf{x}_{nd} - \mathbf{x}_{md})^2}{\gamma_d}\right) && \text{(ARD kernel)} \\
\kappa(\mathbf{x}_n, \mathbf{x}_m) &= \kappa_{\theta_1}(\mathbf{x}_n, \mathbf{x}_m) + \kappa_{\theta_2}(\mathbf{x}_n, \mathbf{x}_m) + \ldots + \kappa_{\theta_M}(\mathbf{x}_n, \mathbf{x}_m) && \text{(flexible composition of multiple kernels)}
\end{aligned}$$

- Type-II MLE is a popular choice for learning these hyperparams, by maximizing marginal likelihood

$$p(\mathbf{y}|\sigma^2, \theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \sigma^2\mathbf{I}_N + \mathbf{K}_\theta)$$

- MLE-II for GP regression maximizes the log marginal likelihood w.r.t. the hyperparameters

$$\log p(\mathbf{y}|\sigma^2, \theta) = -\frac{1}{2}\log|\sigma^2\mathbf{I}_N + \mathbf{K}_\theta| - \frac{1}{2}\mathbf{y}^\top(\sigma^2\mathbf{I}_N + \mathbf{K}_\theta)^{-1}\mathbf{y} + \text{const}$$

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution
- The prior is still GP, therefore $p(\boldsymbol{f}) = \mathcal{N}(0, \mathbf{K})$

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution
- The prior is still GP, therefore $p(\boldsymbol{f}) = \mathcal{N}(0, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{y})$ can again be written as

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution
- The prior is still GP, therefore $p(\boldsymbol{f}) = \mathcal{N}(0, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{y})$ can again be written as

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

- This in general is not as easy to compute as in case of GP regression

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution
- The prior is still GP, therefore $p(\boldsymbol{f}) = \mathcal{N}(0, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{y})$ can again be written as

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

- This in general is not as easy to compute as in case of GP regression
  - $p(f_*|\boldsymbol{f})$ is still not a problem (will be Gaussian)

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution
- The prior is still GP, therefore $p(\boldsymbol{f}) = \mathcal{N}(0, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{y})$ can again be written as

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

- This in general is not as easy to compute as in case of GP regression
  - $p(f_*|\boldsymbol{f})$ is still not a problem (will be Gaussian)
  - $p(\mathbf{f}|\boldsymbol{y}) \propto p(\boldsymbol{f})p(\boldsymbol{y}|\boldsymbol{f})$ will require approximation (e.g., Laplace, MCMC, variational, etc.)

GP for Classification and GLMs

- Binary classification: Now the likelihood $p(\boldsymbol{y}|\boldsymbol{f})$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$
- For multiclass GP ($K > 2$ classs), $p(y_n|f_n)$ will be multinoulli (note: f_n will be a $K \times 1$ vector)
- For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution
- The prior is still GP, therefore $p(\boldsymbol{f}) = \mathcal{N}(0, \mathbf{K})$
- The posterior predictive $p(y_*|\boldsymbol{y})$ can again be written as

$$p(y_*|\boldsymbol{y}) = \int p(y_*|f_*)p(f_*|\boldsymbol{y})df_* = \int p(y_*|f_*)p(f_*|\mathbf{f})p(\mathbf{f}|\boldsymbol{y})d\boldsymbol{f}\,df_*$$

- This in general is not as easy to compute as in case of GP regression
  - $p(f_*|\boldsymbol{f})$ is still not a problem (will be Gaussian)
  - $p(\mathbf{f}|\boldsymbol{y}) \propto p(\boldsymbol{f})p(\boldsymbol{y}|\boldsymbol{f})$ will require approximation (e.g., Laplace, MCMC, variational, etc.)
  - The overall integral will require approximation as well

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1}\boldsymbol{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{k}_*
\end{aligned}
$$

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \boldsymbol{y} \qquad\qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1} \boldsymbol{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1} \mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\boldsymbol{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1} \boldsymbol{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1} \mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]. Some approaches for speeding up GPs

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\mathbf{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1}\boldsymbol{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]. Some approaches for speeding up GPs
  - Inducing Point Methods (condition the predictions only on a small set of "learnable" points)

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\mathbf{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\mathbf{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\mathbf{x}_*, \mathbf{x}_*) + \sigma^2 - \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]. Some approaches for speeding up GPs
  - Inducing Point Methods (condition the predictions only on a small set of "learnable" points)
  - Divide-and-Conquer methods (learn GP on small subsets of data and aggregate predictions)

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\mathbf{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\mathbf{x}_*, \mathbf{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]. Some approaches for speeding up GPs
  - Inducing Point Methods (condition the predictions only on a small set of "learnable" points)
  - Divide-and-Conquer methods (learn GP on small subsets of data and aggregate predictions)
  - Kernel approximations

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$\begin{aligned} p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\ \mu_* &= \mathbf{k}_*^\top \mathbf{C}_N^{-1}\boldsymbol{y} \qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\ \sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{k}_* \end{aligned}$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]. Some approaches for speeding up GPs
  - Inducing Point Methods (condition the predictions only on a small set of "learnable" points)
  - Divide-and-Conquer methods (learn GP on small subsets of data and aggregate predictions)
  - Kernel approximations
- Note that nearest neighbor methods and kernel methods also face similar issues w.r.t. scalability

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$
\begin{aligned}
p(y_*|\boldsymbol{y}) &= \mathcal{N}(y_*|\mu_*, \sigma_*^2) \\
\mu_* &= \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\boldsymbol{y} \qquad\qquad (O(N) \text{ cost assuming } \mathbf{C}_N^{-1} \text{ is pre-computed}) \\
\sigma_*^2 &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\mathbf{k}_*
\end{aligned}
$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space
- A lot of work on speeding up GPs[1]. Some approaches for speeding up GPs
  - Inducing Point Methods (condition the predictions only on a small set of "learnable" points)
  - Divide-and-Conquer methods (learn GP on small subsets of data and aggregate predictions)
  - Kernel approximations
- Note that nearest neighbor methods and kernel methods also face similar issues w.r.t. scalability
  - Many tricks to speed up kernel methods can be used for speeding up GPs too

[1] When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018

GP: A few comments

- GP is a nonparametric model

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*^\top \mathbf{C}_N^{-1}\boldsymbol{y} = \mathbf{k}_*^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*^\top \mathbf{C}_N^{-1} \boldsymbol{y} = \mathbf{k}_*^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \boldsymbol{x}_n)$, which means f is written in terms of all training examples

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \boldsymbol{y} = \mathbf{k}_*{}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \boldsymbol{x}_n)$, which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*^\top \mathbf{C}_N^{-1}\boldsymbol{y} = \mathbf{k}_*^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \boldsymbol{x}_n)$, which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples
- In contrast, a parametric model has a size that doesn't grow with training data

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\boldsymbol{y} = \mathbf{k}_*{}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^N \alpha_n k(., \boldsymbol{x}_n)$, which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples
- In contrast, a parametric model has a size that doesn't grow with training data
  - E.g., a linear model learns a fixed-sized weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (D parameters, size independent of N)

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \boldsymbol{y} = \mathbf{k}_*{}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \boldsymbol{x}_n)$, which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples
- In contrast, a parametric model has a size that doesn't grow with training data
  - E.g., a linear model learns a fixed-sized weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (D parameters, size independent of N)
- Nonparametric models therefore are more flexible since their complexity is not limited beforehand

GP: A few comments

- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\boldsymbol{x}_*) = \mathbf{k}_*{}^\top \mathbf{C}_N^{-1}\boldsymbol{y} = \mathbf{k}_*{}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_*, \boldsymbol{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \boldsymbol{x}_n)$, which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples
- In contrast, a parametric model has a size that doesn't grow with training data
  - E.g., a linear model learns a fixed-sized weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (D parameters, size independent of N)
- Nonparametric models therefore are more flexible since their complexity is not limited beforehand
  - Note: Methods such as nearest neighbors and kernel SVMs are also nonparametric (but not Bayesian)

GP: A few comments

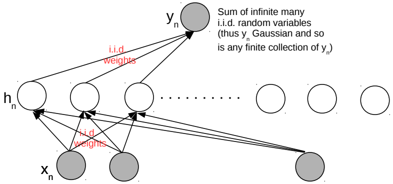
- GP is a nonparametric model. Why called "nonparametric"?
  - Complexity (representation size) of the function f grows in the size of training data
  - To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

$$\mu_* = f(\mathbf{x}_*) = \mathbf{k}_*{}^\top \mathbf{C}_N^{-1} \mathbf{y} = \mathbf{k}_*{}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_*, \mathbf{x}_n)$$

  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \mathbf{x}_n)$, which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples
- In contrast, a parametric model has a size that doesn't grow with training data
  - E.g., a linear model learns a fixed-sized weight vector $\mathbf{w} \in \mathbb{R}^D$ (D parameters, size independent of N)
- Nonparametric models therefore are more flexible since their complexity is not limited beforehand
  - Note: Methods such as nearest neighbors and kernel SVMs are also nonparametric (but not Bayesian)
- GPs equivalent to infinitely-wide single hidden-layer neural net (under some technical conditions)

Neural Networks and Gaussian Processes

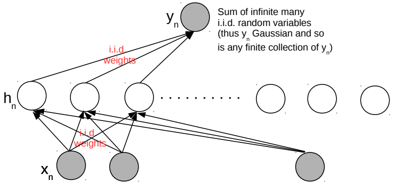
- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)[2]. Based on a simple application of central limit theorem



[2]Priors for infinite networks, Tech Report, 1994
[3]Deep Neural Networks as Gaussian Processes (ICLR 2018)

Neural Networks and Gaussian Processes

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)[2]. Based on a simple application of central limit theorem

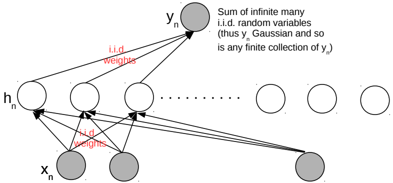


- A useful result for several reasons

[2]Priors for infinite networks, Tech Report, 1994
[3]Deep Neural Networks as Gaussian Processes (ICLR 2018)

Neural Networks and Gaussian Processes

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)[2]. Based on a simple application of central limit theorem



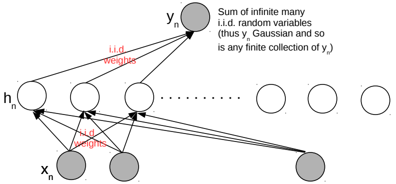
- A useful result for several reasons
  - Can use a GP instead of an infinitely wide Bayesian NN (which is impractical anyway)

[2] Priors for infinite networks, Tech Report, 1994

[3] Deep Neural Networks as Gaussian Processes (ICLR 2018)

Neural Networks and Gaussian Processes

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)[2]. Based on a simple application of central limit theorem

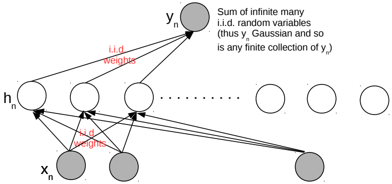


- A useful result for several reasons
  - Can use a GP instead of an infinitely wide Bayesian NN (which is impractical anyway)
  - With GPs, inference is easy (at least for regression and with known hyperparams)

[2] Priors for infinite networks, Tech Report, 1994
[3] Deep Neural Networks as Gaussian Processes (ICLR 2018)

Neural Networks and Gaussian Processes

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)[2]. Based on a simple application of central limit theorem

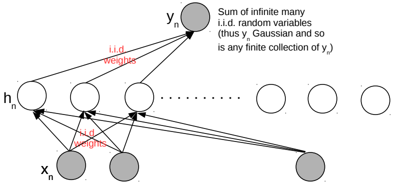


- A useful result for several reasons
  - Can use a GP instead of an infinitely wide Bayesian NN (which is impractical anyway)
  - With GPs, inference is easy (at least for regression and with known hyperparams)
  - A proof that GPs can also learn any function (just like infinitely wide neural nets - Hornik's theorem)

[2] Priors for infinite networks, Tech Report, 1994
[3] Deep Neural Networks as Gaussian Processes (ICLR 2018)

Neural Networks and Gaussian Processes

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)[2]. Based on a simple application of central limit theorem



- A useful result for several reasons
  - Can use a GP instead of an infinitely wide Bayesian NN (which is impractical anyway)
  - With GPs, inference is easy (at least for regression and with known hyperparams)
  - A proof that GPs can also learn any function (just like infinitely wide neural nets - Hornik's theorem)
- Connection recently generalized to infinitely wide multiple hidden layer NN (Lee et al, 2018)[3]

[2] Priors for infinite networks, Tech Report, 1994

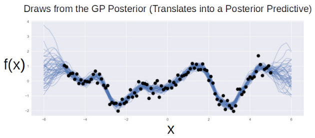
[3] Deep Neural Networks as Gaussian Processes (ICLR 2018)

GP: A few other comments

- Can be thought of as Bayesian analogues of kernel methods

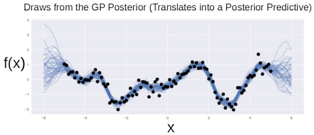
GP: A few other comments

- Can be thought of as Bayesian analogues of kernel methods
  - Can get estimate in the uncertainty in the function and its predictions



GP: A few other comments

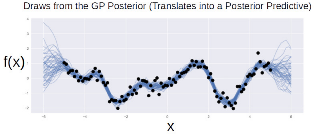
- Can be thought of as Bayesian analogues of kernel methods
  - Can get estimate in the uncertainty in the function and its predictions



  - Can learn the kernel (by learning the hyperparameters of the kernels)

GP: A few other comments

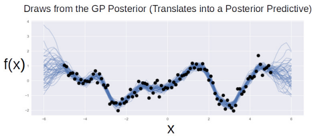
- Can be thought of as Bayesian analogues of kernel methods
  - Can get estimate in the uncertainty in the function and its predictions



  - Can learn the kernel (by learning the hyperparameters of the kernels)
- Not limited to supervised learning problems

GP: A few other comments

- Can be thought of as Bayesian analogues of kernel methods
  - Can get estimate in the uncertainty in the function and its predictions



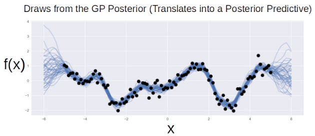
  - Can learn the kernel (by learning the hyperparameters of the kernels)
- Not limited to supervised learning problems
  - The function f could even be a mapping of an unknown quantity to an observed quantity

$$\boldsymbol{x}_n = f(\boldsymbol{z}_n) + \text{“noise”}$$

where $\boldsymbol{z}_n$ is a latent representation of $\boldsymbol{x}_n$ (“GP latent variable models” for nonlin. dim. red.)

GP: A few other comments

- Can be thought of as Bayesian analogues of kernel methods
  - Can get estimate in the uncertainty in the function and its predictions



  - Can learn the kernel (by learning the hyperparameters of the kernels)
- Not limited to supervised learning problems
  - The function f could even be a mapping of an unknown quantity to an observed quantity

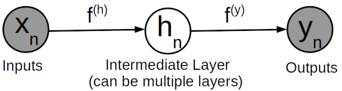
$$\boldsymbol{x}_n = f(\boldsymbol{z}_n) + \text{“noise”}$$

where $\boldsymbol{z}_n$ is a latent representation of $\boldsymbol{x}_n$ (“GP latent variable models” for nonlin. dim. red.)

- Many mature implementations of GP exist. You may check out
  - GPML (MATLAB), GPsuff (MATLAB/Octave), GPy (Python), GPyTorch (PyTorch)

Other Recent Advances on Gaussian Processes

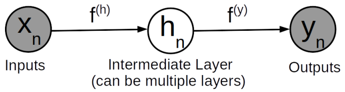
- Deep Gaussian Processes (DGP)
  - Akin to a deep neural network where each hidden node is modeled by a GP



  - A nice alternative to linear transform + nonlinearity based neural nets, e.g., $\boldsymbol{h} = \tanh(\mathbf{W}\boldsymbol{x})$

Other Recent Advances on Gaussian Processes

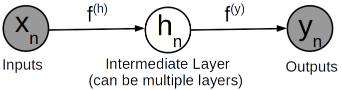
- Deep Gaussian Processes (DGP)
  - Akin to a deep neural network where each hidden node is modeled by a GP



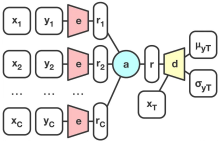
  - A nice alternative to linear transform + nonlinearity based neural nets, e.g., $\boldsymbol{h} = \tanh(\mathbf{W}\boldsymbol{x})$
- GPs with deep kernels defined by neural nets

Other Recent Advances on Gaussian Processes

- Deep Gaussian Processes (DGP)
  - Akin to a deep neural network where each hidden node is modeled by a GP

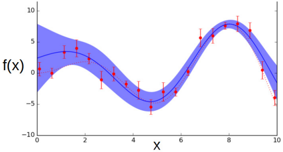


  - A nice alternative to linear transform + nonlinearity based neural nets, e.g., $\boldsymbol{h} = \tanh(\mathbf{W}\boldsymbol{x})$
- GPs with deep kernels defined by neural nets
- Neural Processes and Conditional Neural Processes (GP + neural nets): Most recent development



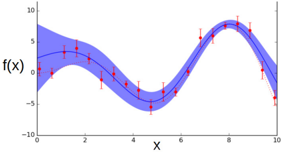
GPs are very versatile!

- GPs enable us to learn nonlinear functions while also capturing the uncertainty



GPs are very versatile!

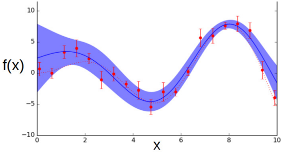
- GPs enable us to learn nonlinear functions while also capturing the uncertainty



- Uncertainty can tell us where to acquire more training data to improve the function's estimate

GPs are very versatile!

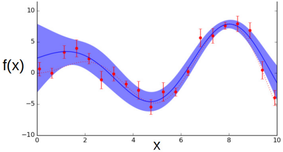
- GPs enable us to learn nonlinear functions while also capturing the uncertainty



- Uncertainty can tell us where to acquire more training data to improve the function's estimate
  - Especially useful if we can't get too many training examples (e.g., expensive inputs and/or labels)

GPs are very versatile!

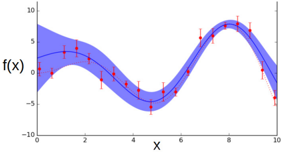
- GPs enable us to learn nonlinear functions while also capturing the uncertainty



- Uncertainty can tell us where to acquire more training data to improve the function's estimate
  - Especially useful if we can't get too many training examples (e.g., expensive inputs and/or labels)
- This is very useful in a wide range of applications involving sequential decision-making

GPs are very versatile!

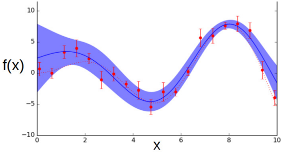
- GPs enable us to learn nonlinear functions while also capturing the uncertainty



- Uncertainty can tell us where to acquire more training data to improve the function's estimate
  - Especially useful if we can't get too many training examples (e.g., expensive inputs and/or labels)
- This is very useful in a wide range of applications involving sequential decision-making
  - Active Learning: Learning a function by gathering the most informative training examples

GPs are very versatile!

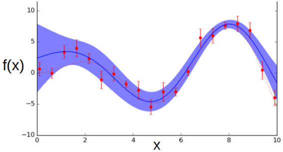
- GPs enable us to learn nonlinear functions while also capturing the uncertainty



- Uncertainty can tell us where to acquire more training data to improve the function's estimate
  - Especially useful if we can't get too many training examples (e.g., expensive inputs and/or labels)
- This is very useful in a wide range of applications involving sequential decision-making
  - Active Learning: Learning a function by gathering the most informative training examples
  - Bayesian Optimization: Optimizing an expensive to evaluate functions (and maybe we don't even know it form) – boils down to simultaneous function learning and optimization

GPs are very versatile!

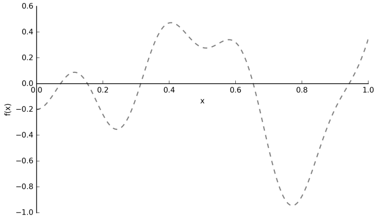
- GPs enable us to learn nonlinear functions while also capturing the uncertainty



- Uncertainty can tell us where to acquire more training data to improve the function's estimate
  - Especially useful if we can't get too many training examples (e.g., expensive inputs and/or labels)
- This is very useful in a wide range of applications involving sequential decision-making
  - Active Learning: Learning a function by gathering the most informative training examples
  - Bayesian Optimization: Optimizing an expensive to evaluate functions (and maybe we don't even know it form) – boils down to simultaneous function learning and optimization

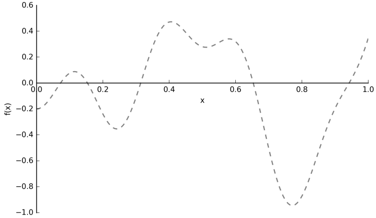
Bayesian Optimization: The Basic Formulation

- Consider finding the optima x_* (say minima) of a function $f(x)$



Bayesian Optimization: The Basic Formulation

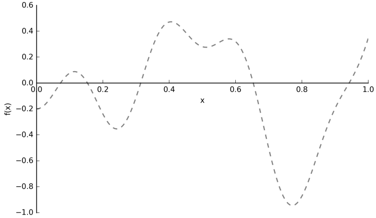
- Consider finding the optima x_* (say minima) of a function $f(x)$



- Caveat: We don't know the form of the function; can't get its gradient,Hessian, etc

Bayesian Optimization: The Basic Formulation

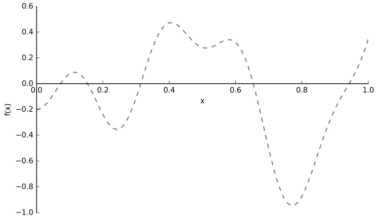
- Consider finding the optima x_* (say minima) of a function $f(x)$



- Caveat: We don't know the form of the function; can't get its gradient,Hessian, etc
- Suppose we can only query the function's values at certain points (i.e., only black-box access)

Bayesian Optimization: The Basic Formulation

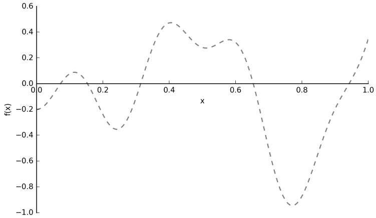
- Consider finding the optima x_* (say minima) of a function $f(x)$



- Caveat: We don't know the form of the function; can't get its gradient,Hessian, etc
- Suppose we can only query the function's values at certain points (i.e., only black-box access)
- Thus we have to learn the function as well as find its optima

Bayesian Optimization: The Basic Formulation

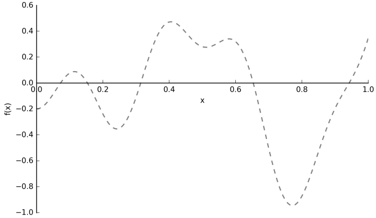
- Consider finding the optima x_* (say minima) of a function $f(x)$



- Caveat: We don't know the form of the function; can't get its gradient,Hessian, etc
- Suppose we can only query the function's values at certain points (i.e., only black-box access)
- Thus we have to learn the function as well as find its optima
- Can learn the function using GP and use the uncertainty to decide which $f(x)$ value to query next

Bayesian Optimization: The Basic Formulation

- Consider finding the optima x_* (say minima) of a function $f(x)$



- Caveat: We don't know the form of the function; can't get its gradient,Hessian, etc
- Suppose we can only query the function's values at certain points (i.e., only black-box access)
- Thus we have to learn the function as well as find its optima
- Can learn the function using GP and use the uncertainty to decide which $f(x)$ value to query next
- Will look at it in more detail later this semester