Gaussian Process (Bayesian Learning meets Kernels)

CS772A: Probabilistic Machine Learning

Piyush Rai

Discriminative Models

The function f gives a real-valued score $f(\boldsymbol{x})$ to each input $\boldsymbol{x}$. Parameters of the likelihood model $p(y|f,\boldsymbol{x})$ depend on this score

- Discriminative models learn a function f that maps inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$

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

- We usually define the function f using weights (i.e., the model parameters), e.g.,
  - Linear: $f(\boldsymbol{x}) = \boldsymbol{w}^{\top}\boldsymbol{x}$, used in linear/logistic regression and GLM
  - Nonlinear: $f(\boldsymbol{x}) = \text{NN}(\boldsymbol{x},\boldsymbol{w})$, used in deep neural nets based models
- Since these models use parameters, we call them "parametric" models
- We can also define the function f "directly" without using parameters
  - Essentially, this can be done using a "nonparametric" approach
  - It would be similar to nearest neighbors or kernel SVMs which are also "nonparametric"
- Gaussian Process (GP) is such a Bayesian nonparametric approach

Gaussian Process (GP)

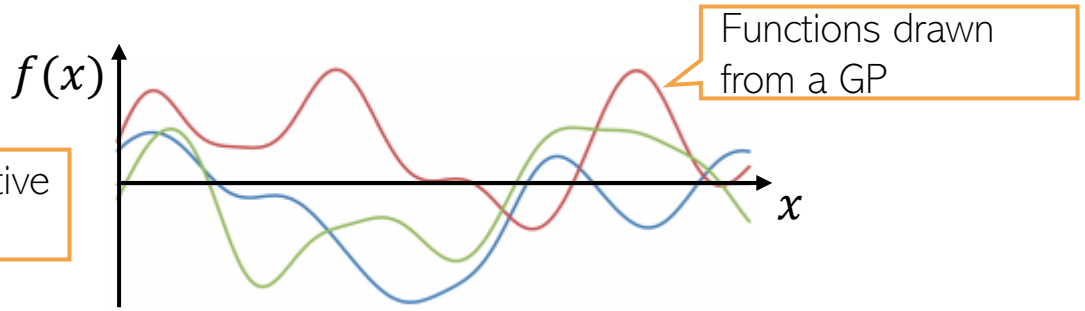
- A Gaussian Process (GP) defines a distribution over functions and is denoted as

Mean Function

Covariance Function

$$\mathcal{GP}(\mu(.), \kappa(.,.))$$

Should be a positive definite function



- Mean function defines what functions drawn from this GP look like on average

$$\mu(x) = \mathbb{E}_{f \sim \mathcal{GP}(\mu,\kappa)}[f(x)]$$

- Covariance/kernel function defines the similarity between a pair of inputs

$$K_{ij} = \kappa(x_i, x_j)$$

- Covariance/kernel function controls the shape of the functions drawn from GP
  - If $\kappa(x_i, x_j)$ is high then $f(x_i)$ and $f(x_j)$ will have similar values

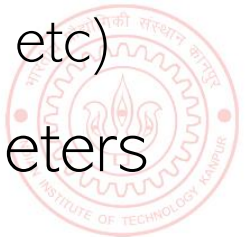
Covariance/kernel functions

- Kernel functions are popular in learning nonlinear functions (e.g., kernel SVMs)
- Using a kernel corresponds to applying a nonlinear mapping function ϕ on inputs, s.t.,

Kernel function $\kappa(.,.)$ gives the pairwise similarity between two inputs x and x' in the new feature space defined by mapping function $\phi(.)$

$$\kappa(x, x') = \phi(x)^\top \phi(x')$$

- A wide variety of kernel functions exists that suit different types of data
  - Linear kernel, polynomial kernel, Squared exponential (RBF) kernel
  - Automatic Relevance Determination (ARD) kernel
  - Matérn kernel, Periodic kernel, and many others
- We can combine multiple kernels and use them for GP, e.g., possible kernels
  - $\kappa_1 + \kappa_2$, $\kappa_1 \times \kappa_2$, $\alpha\kappa_1 + \kappa_2$, etc (add, mult, positive scalar mult, or combinations, etc)
- We can learn how to combine multiple kernels and learn their hyperparameters
  - Possible naturally with the Bayesian approach



Covariance/kernel functions

- Visualization of some kernel functions (how similarity changes with "distance")

Kernel for all the plots is $\kappa(x, x')$, i.e., fixing one input as x' and varying the other input x (along the X axis)

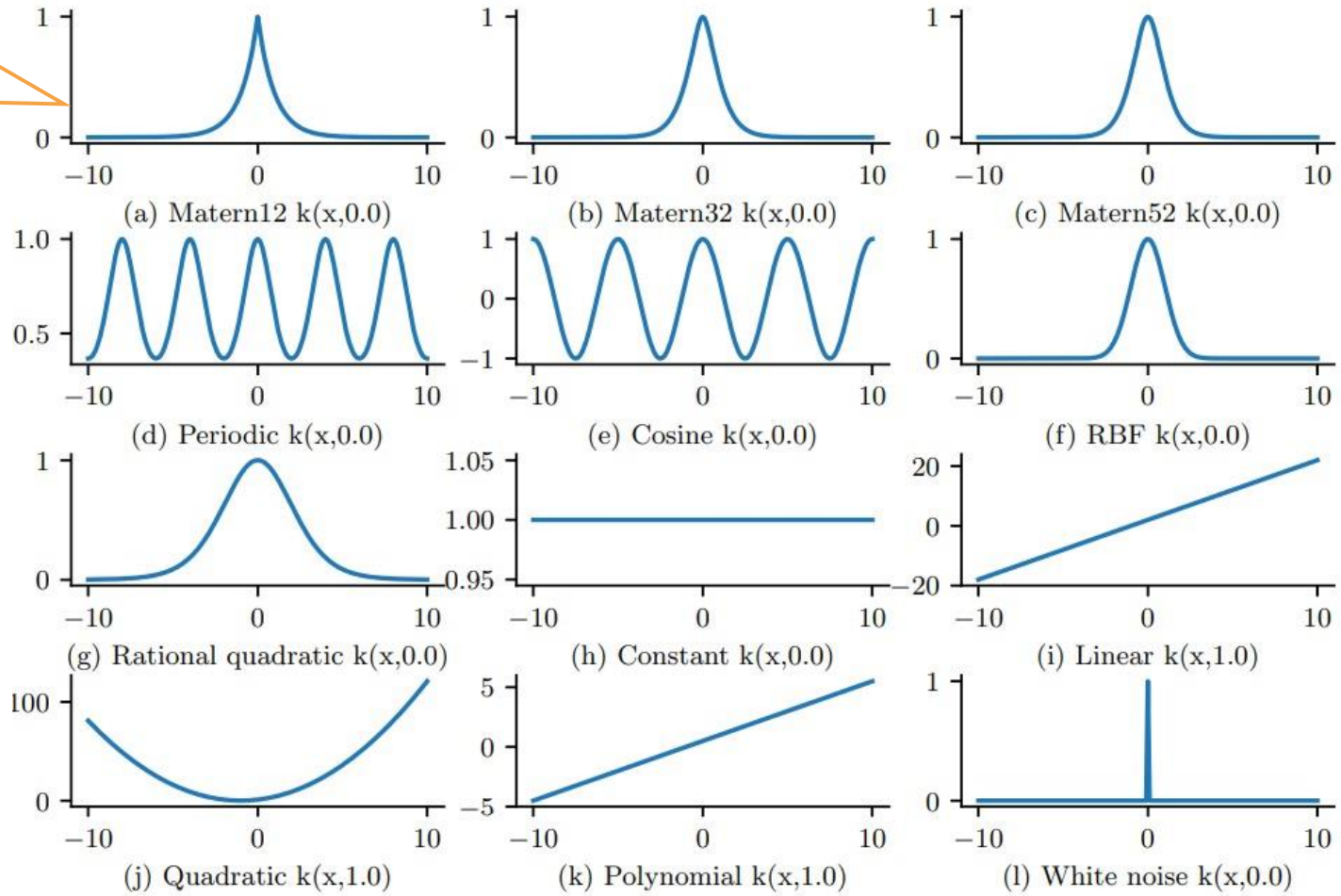


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

Covariance/kernel functions

- Examples of functions $f \sim \mathbf{GP}(\mu, \kappa)$ drawn from a GP using some standard kernels

Each plot shows 3 random functions drawn from the corresponding GP with the specified kernel function

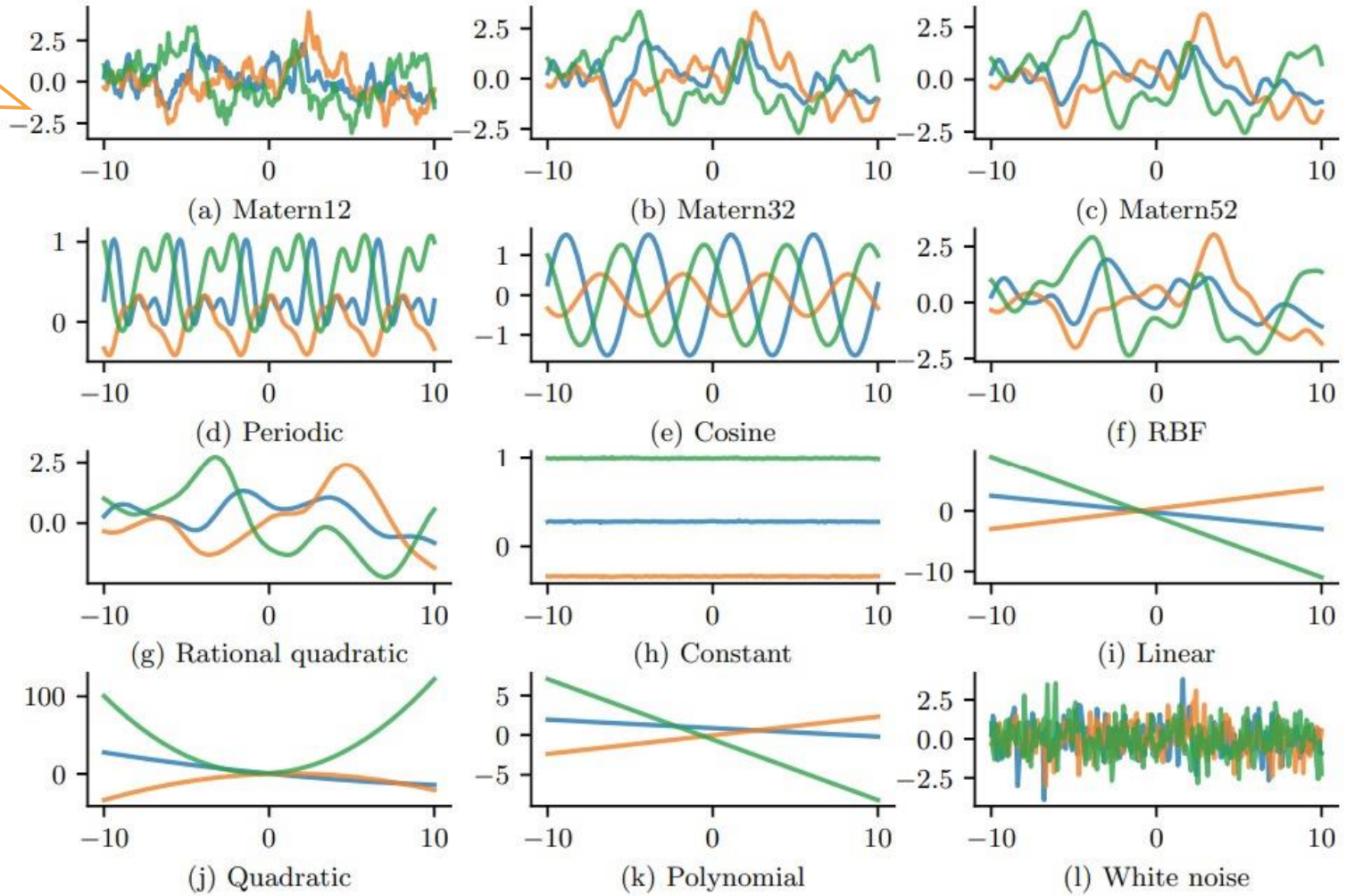


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

Combining two (or more) covariance/kernel functions

- Adding two kernels and its effect on the function f defined by the resulting kernel

Kernel for all 4 plots is $\kappa(x, x')$, i.e., fixing x' as 1 and varying the other input x (along the X axis)

Two randomly drawn functions from the GP with this kernel combination

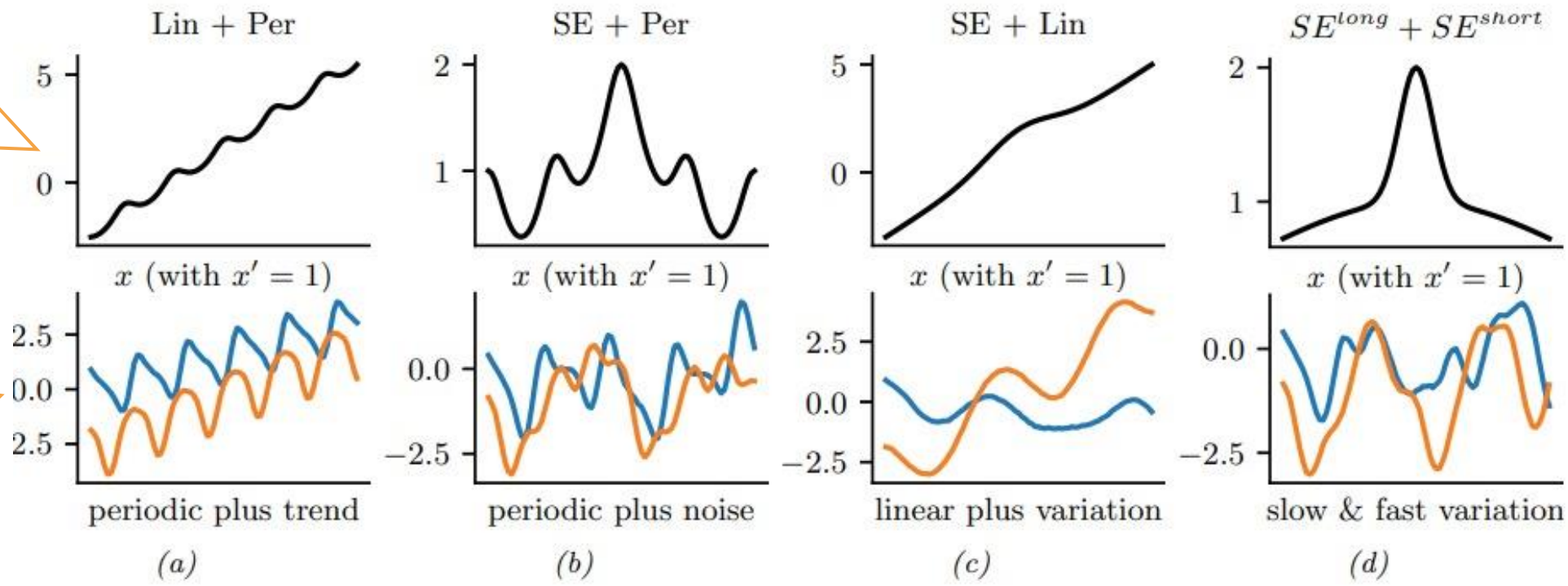


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

Combining two (or more) covariance/kernel functions

- Multiplying two kernels and its effect on the function f defined by the resulting kernel

Kernel for all 4 plots is $\kappa(x, x')$, i.e., fixing x' as 1 and varying the other input x (along the X axis)

Two randomly drawn functions from the GP with this kernel combination

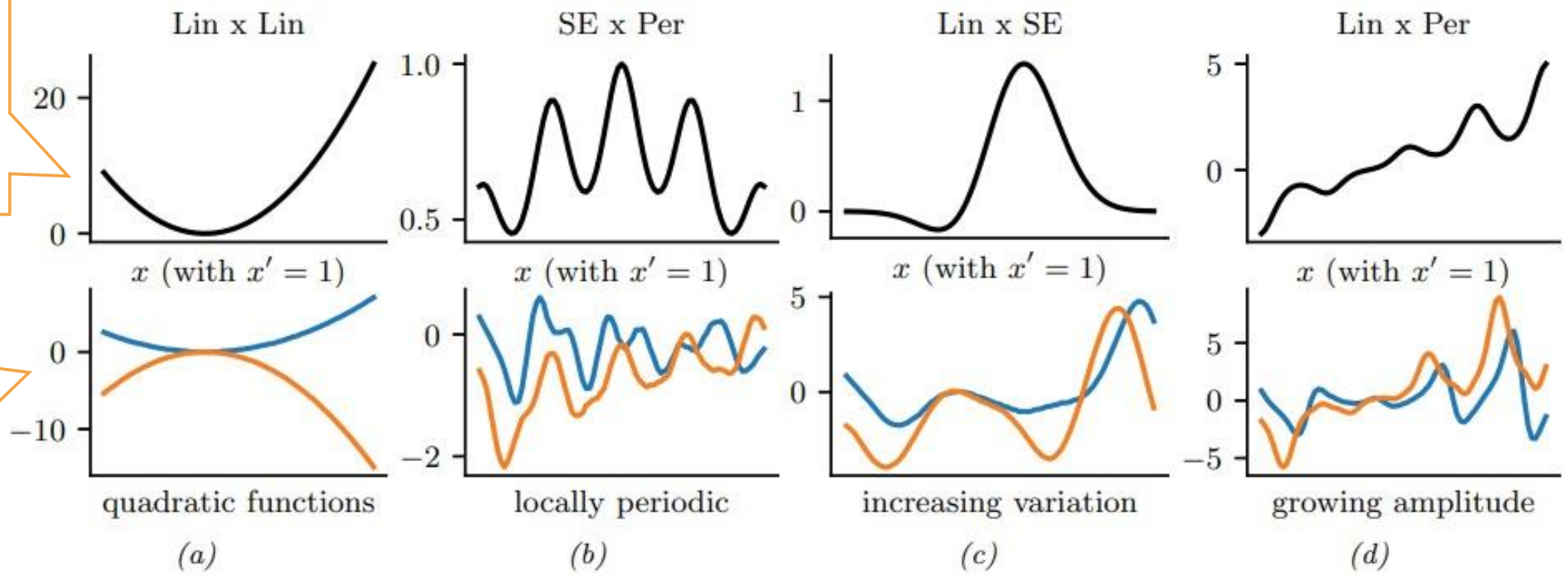


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

Gaussian Process: The Predictive Model

- If $f \sim \mathcal{GP}(\mu, \kappa)$ then f's value at any finite set of inputs is jointly Gaussian

$$p\left(\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \vdots \\ \mu(x_N) \end{bmatrix}, \begin{bmatrix} \kappa(x_1,x_1) & \cdots & \kappa(x_1,x_N) \\ \kappa(x_2,x_1) & & \kappa(x_2,x_N) \\ \vdots & \ddots & \vdots \\ \kappa(x_N,x_1) & \cdots & \kappa(x_N,x_N) \end{bmatrix}\right) \Rightarrow p(\mathbf{f}) = \mathcal{N}(\mathbf{f} \mid \boldsymbol{\mu}, \mathbf{K})$$

$\mathbf{f}$: $N \times 1$; $\boldsymbol{\mu}$: $N \times 1$; $\mathbf{K}$: $N \times N$

- Denoting f's score for a new test input $\boldsymbol{x}_*$ as $f_* = f(\boldsymbol{x}_*)$, we must also have

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

$\mathbf{k}_* = [\kappa(x_1, x_*), \kappa(x_2, x_*), \ldots, \kappa(x_N, x_*)]^\top$

$N+1$ dim Gaussian

- Very useful result: Easy to see that, given the above, the GP predictive distribution

$$p(f_* \mid \mathbf{f}) = \mathcal{N}(f_* \mid \mu_* + \mathbf{k}_*^\top \mathbf{K}^{-1}(\mathbf{f} - \boldsymbol{\mu}), \kappa(x_*, x_*) - \mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{k}_*)$$

- Thus score of f on $\boldsymbol{x}_*$ given scores on training inputs has a Gaussian distribution

Gaussian Process: The Predictive Model

- Assuming the mean function $\mu(\boldsymbol{x}) = \mathbf{0}$, the conditional distribution of score becomes

$$p(f_*|\mathbf{f}) = \mathcal{N}(f_*|\mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{f}, \kappa(x_*, x_*) - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*) = \mathcal{N}(f_*|\hat{\mu}, \hat{\sigma}^2)$$

- Note that the predictive mean $\hat{\mu}$ can be written in the following two equivalent ways

Weighted sum of the scores of the N training inputs

$$\hat{\mu} = \sum_{i=1}^{N} \beta_i f_i$$

Like doing a "weighted" nearest neighbors using all the N training inputs as neighbors with weight β_i given to input $\boldsymbol{x}_i$ with score $f_i = f(\boldsymbol{x}_i)$

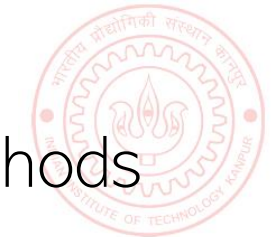
Weighted sum of kernel based similarities of $\boldsymbol{x}_*$ with the N training inputs

$$\hat{\mu} = \sum_{i=1}^{N} \alpha_i \kappa(x_i, x_*)$$

Methods like kernel regression or kernel SVM have their predictions in this form

- Advantage: GP also gives the score's variance $\hat{\sigma}^2 = \kappa(x_*, x_*) - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*$

- Thus GP can be viewed as a probabilistic/Bayesian version of kernel methods



From GP Scores ($\mathbf{f}$) to Actual Outputs ($\boldsymbol{y}$)

- Assume a supervised learning problem with N training examples $(\boldsymbol{X}, \boldsymbol{y}) = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$
- Denoting $f_i = f(\boldsymbol{x}_i)$, for regression with added noise $\mathcal{N}(0, \beta^{-1})$

This GP score $f_i = f(\boldsymbol{x}_i)$ is the mean of this Gaussian likelihood

$$y_i = f_i + \epsilon_i \quad \Rightarrow \quad p(y_i|f_i) = \mathcal{N}(y_i|f_i, \beta^{-1})$$

The likelihood function for all the training outputs (assuming i.i.d.)

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

For multi-class case with C classes, we will use a multinoulli with probability vector softmax(f_i) where f_i will be a C-dim vector of logits

- Likewise, for binary classification, likelihood $p(\boldsymbol{y}|\mathbf{f}) = \prod_{i=1}^N \text{Bernoulli}(y_i|\sigma(f_i))$
- In general, when using GP, the PPD $p(y_*|\boldsymbol{y})$ of output y_* for a new test input $\boldsymbol{x}_*$

Averaging over f

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

Skipping the training and test inputs from the PPD notation

Likelihood

GP predictive: Always a Gaussian

The GP posterior

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

The GP posterior

Likelihood function for training output

$$p(\mathbf{f}|\boldsymbol{y}) \propto \boldsymbol{p}(\mathbf{f})\boldsymbol{p}(\boldsymbol{y}|\boldsymbol{f})$$

The GP prior: Gaussian $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K})$ or $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|0, \mathbf{K})$ is mean function is zero

GP Prediction with Gaussian Likelihood

- In general, the PPD when using GP is defined as

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

And don't even have to compute/use the posterior $p(\mathbf{f}|\mathbf{y})$ (which in this case is a Gaussian by the way ☺) to get the PPD

- For Gaussian likelihood (and fixed hyperparams), we don't need to do above integral
- Reason: The marginal likelihood is Gaussian

Gaussian likelihood (assuming β is fixed)

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \beta^{-1}\mathbf{I}_N)$$

GP prior

Assuming zero mean function

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$$

Marginal likelihood of training outputs

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})d\mathbf{f} = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \beta^{-1}\mathbf{I}_N) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_N)$$

Marginal likelihood of training and test outputs

$$p\left(\begin{bmatrix}\mathbf{y}\\ y_*\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{y}\\ y_*\end{bmatrix}|\begin{bmatrix}\mathbf{0}\\ 0\end{bmatrix}, \begin{bmatrix}\mathbf{C}_N & \mathbf{k}_*\\ \mathbf{k}_*^\top & \kappa(x_*, x_*) + \beta^{-1}\end{bmatrix}\right)$$

PPD obtained using joint to conditional results of Gaussians

$$p(y_*|\mathbf{y}) = \mathcal{N}(y_*|\mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{y}, \kappa(x_*, x_*) - \mathbf{k}_*^\top \mathbf{C}_N^{-1}\mathbf{k}_* + \beta^{-1})$$

- $p(y_*|\mathbf{y})$ is almost identical to $p(f_*|\mathbf{f})$ with $\mathbf{K}$ replaced by $\mathbf{C}_N$ + extra β^{-1} noise variance

Learning Hyperparameters in GP based Models

- Can learn the hyperparameters of the GP prior as well as of the likelihood model
- Assuming $\mu = 0$, the hyperparams of GP are cov/kernel function hyperparams

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

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

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

Can help in feature selection (irrelevant features will tend to have very large γ_d)

Different RBF kernel bandwidth γ_d for each feature

Ability to learn kernel hyperparams (without cross-valid) is another very appealing property of GP

- MLE-II is a popular choice for learning these hyperparams (otherwise MCMC, VI, etc)
- Denoting the covariance/kernel matrix as $\mathbf{K}_\theta$, for Gaussian likelihood case, the marg-lik

$$p(\boldsymbol{y}|\theta, \beta^{-1}) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \mathbf{K}_\theta + \beta^{-1}\mathbf{I}_N)$$

- This can be maximized to learn θ and β
- For non-Gaussian likelihoods, the marg-lik itself will need to be approximated

Weight Space View vs Function Space View

- GPs are defined w.r.t. a function space that models input-output relationship
- In contrast, we have seen models that are defined w.r.t. a weight space, e.g.,

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{w},\beta^{-1}\boldsymbol{I}_N)$$ Likelihood

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_0,\boldsymbol{\Sigma}_0)$$ Prior over weight vector

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w} = \mathcal{N}(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{\mu}_0,\beta^{-1}\boldsymbol{I}_N + \boldsymbol{X}\boldsymbol{\Sigma}_0\boldsymbol{X}^\top)$$ Marginal likelihood after integrating out the weights

$$p(\boldsymbol{y}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0},\beta^{-1}\boldsymbol{I}_N + \boldsymbol{X}\boldsymbol{X}^\top)$$ Marginal likelihood assuming $\boldsymbol{\mu}_0 = \boldsymbol{0}$ and $\boldsymbol{\Sigma}_0 = \boldsymbol{I}$

$$p(\boldsymbol{y}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0},\boldsymbol{X}\boldsymbol{X}^\top)$$ Assuming noise-free likelihood

- Thus the joint marginal of the N responses $y_1, y_2, \ldots, y_N$ is a multivariate Gaussian

$$p\left(\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} x_1^\top x_1 & \cdots & x_1^\top x_N \\ x_2^\top x_1 & & x_2^\top x_N \\ \vdots & \ddots & \vdots \\ x_N^\top x_1 & \cdots & x_N^\top x_N \end{bmatrix}\right)$$

This equivalence also shows that Bayesian linear regression is a special case of GP with linear kernel

Same as a GP $f(x_i) = y_i$, $\mu(x) = 0$ and linear covariance/kernel function $\kappa(x_i, x_j) = x_i^\top x_j$

- Thus GPs can be seen as bypassing the weight space and directly defining the model using a marginal likelihood via a function space defined by the GP

Scalability of GPs

- Computational costs in some steps of GP models scale in the size of training data
- For example, prediction cost is $O(N)$

$O(N)$ cost assuming $\mathbf{C}_N$ is already inverted

$$p(y_*|\boldsymbol{y}) = \mathcal{N}(y_*|\hat{\mu}, \hat{\sigma}^2) \qquad \hat{\mu} = \mathbf{k}_*^\top \mathbf{C}_N^{-1} \boldsymbol{y} \qquad \hat{\sigma}^2 = \kappa(x_*, x_*) - \mathbf{k}_*^\top \mathbf{C}_N^{-1} \mathbf{k}_* + \beta^{-1}$$

- GP models often require matrix inversions (e.g., in marg-lik computation when estimating hyperparameters) – takes $O(N^3)$
- Storage also requires $O(N^2)$ since need to store the covariance matrix
- A lot of work on speeding up GPs[1]. Some prominent approaches include
  - Inducing Point Methods (condition predictions only on a small set of "learnable" points)
  - Divide-and-Conquer (learn GP on small subsets of data and aggregate predictions)
  - Kernel approximations

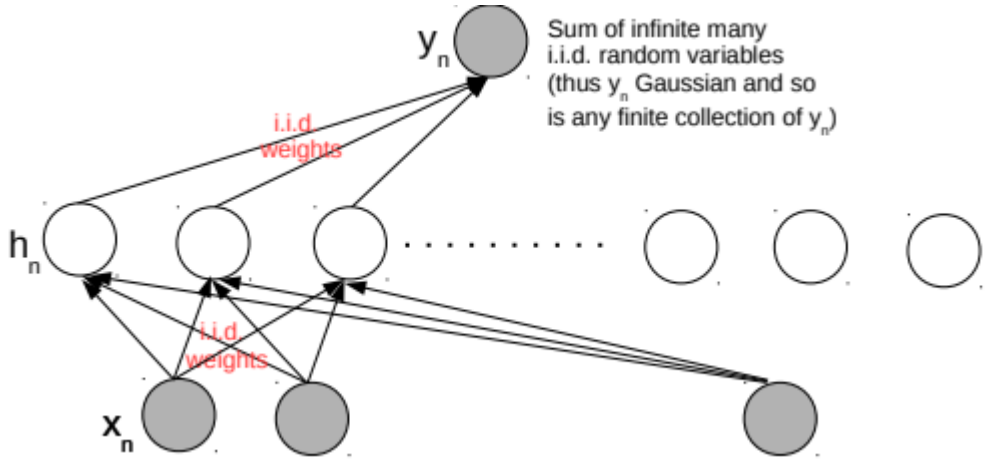
$M \ll N$ pseudo-inputs and pseudo-outputs

- Note that nearest neighbor methods and kernel methods also face similar issues
  - 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

Neural Networks and Gaussian Process

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = GP
- Shown formally by (Neal[2], 1994). Based on applying the central limit theorem



- This equivalence is useful 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 generalized to infinitely wide multiple hidden layer NN (Lee et al[3], 2018)

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

GP: Some other comments

- GPs can be thought of as Bayesian analogues of kernel methods
- Can get estimate of the uncertainty in the function and its predictions
- Can learn the kernel (by learning the hyperparameters of the kernels)
- In some ways, GPs and (Bayesian/ensembles of) deep neural nets have same goals
  - These methods are also very related (though appear different based on their formulation)
  - Several recent papers have investigated these connections
- GP can be a nice alternative to (Bayesian/ensembles of) deep neural networks
  - GP may be preferable if we don't have that much training data (deep networks requires lots of data to train well)
  - When we have lots of training data, training and test speed may be an issue for GP (but faster versions exist)
- Not limited to supervised learning problems
  - f could even define a mapping of low-dim latent variable $\boldsymbol{z}_n$ to an observation $\boldsymbol{x}_n$

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

GP latent variable model for dimensionality reduction (like a kernel version of probabilistic PCA)

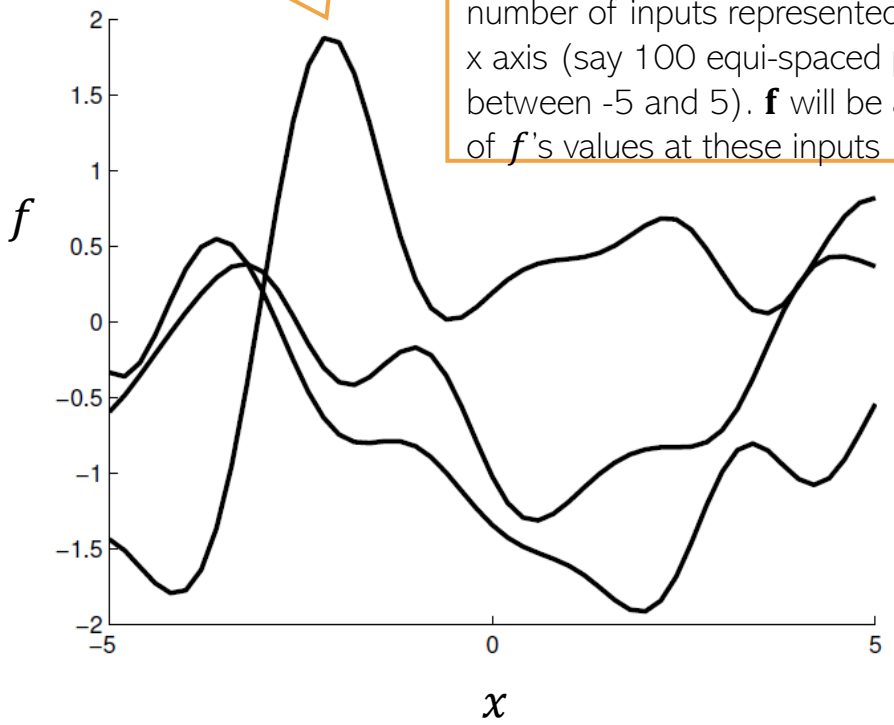
GP: A Visualization

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

- Assumed zero mean function and a squared exponential kernel

Each curve below is obtained by drawing a random $\mathbf{f}$ from the GP prior $p(\mathbf{f}) = \mathcal{N}(\mathbf{0}, \mathbf{K})$ and plotting it.

$\mathbf{K}$ is the kernel matrix of a finite number of inputs represented on the x axis (say 100 equi-spaced points between -5 and 5). $\mathbf{f}$ will be a vector of f's values at these inputs



Shaded area shows the predictive uncertainty for each of the test inputs (+/- 2 std)

Each curve below is obtained by drawing random $\mathbf{f}$'s from the GP posterior $p(\mathbf{f}|\boldsymbol{y})$ which is also a Gaussian (The + symbols denote the training data and we assume noiseless outputs, i.e., $y_i = f_i$) .

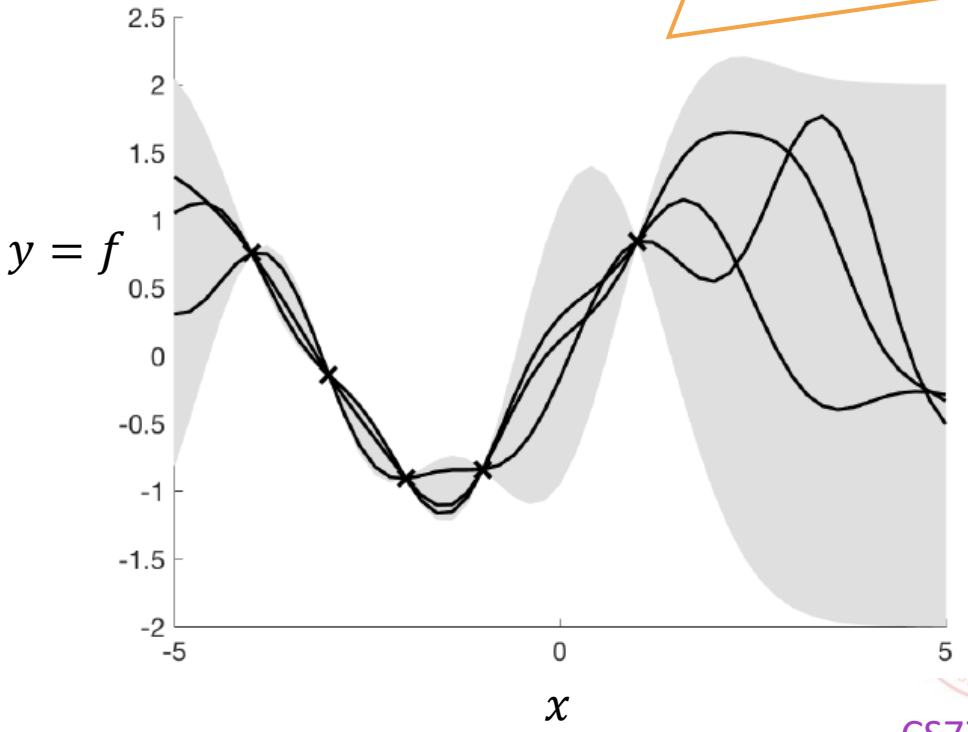


Figure courtesy: MLAPP (Murphy)

GP packages

- Many mature implementations of GP exist. You may check out
  - GPyTorch (PyTorch), GPFlow (Tensorflow)
  - sklearn (Python with some basic GP implementations)
  - GPML (MATLAB), GPsuff (MATLAB/Octave)
  - Many others such as Stan, GPJax
- A comparison of the various packages: https://en.wikipedia.org/wiki/Comparison_of_Gaussian_process_software