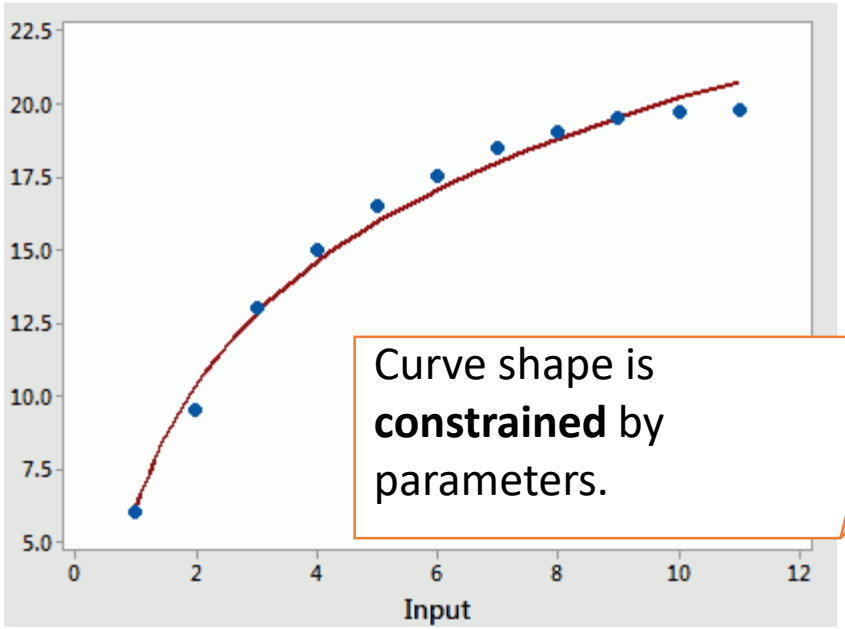
Gaussian Processes

Debaditya Roy

Slides Adapted from Prof. Piyush Rai, IITK (CS772A Gaussian Processes)

Parametric Models

- Discriminative models like **logistic regression** model $p(y \mid \boldsymbol{x})$ directly
- Functional form is **fixed**: $p(y|x) = \sigma(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{b})$
- **Number of parameters is fixed**: Only the weights $\boldsymbol{w}$ and bias $\boldsymbol{b}$, no matter how much data we have.
- **Limited flexibility**: Model can only fit functions within the chosen form (e.g., a linear decision boundary).



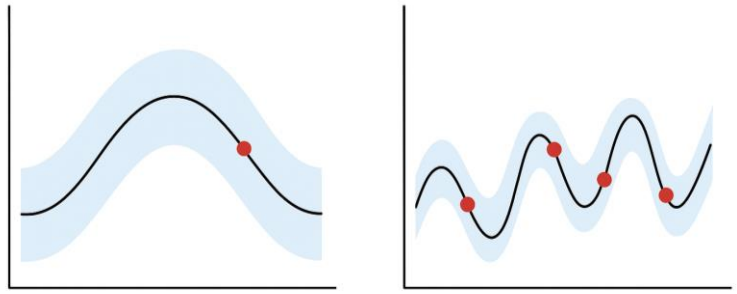
Non-parametric Models

- Gaussian Processes defines a **distribution over functions**:

$$f(x) \sim \mathcal{GP}(\mu(x), \kappa(x, x'))$$

- **No fixed number of parameters:** Predictions are influenced by **all training points**.

- **Model complexity grows with data:**
  More points ⟶ GP adapts its shape.



- **Flexible:** Kernel controls smoothness, but GP can model highly non-linear functions.

Gaussian Process (GP)

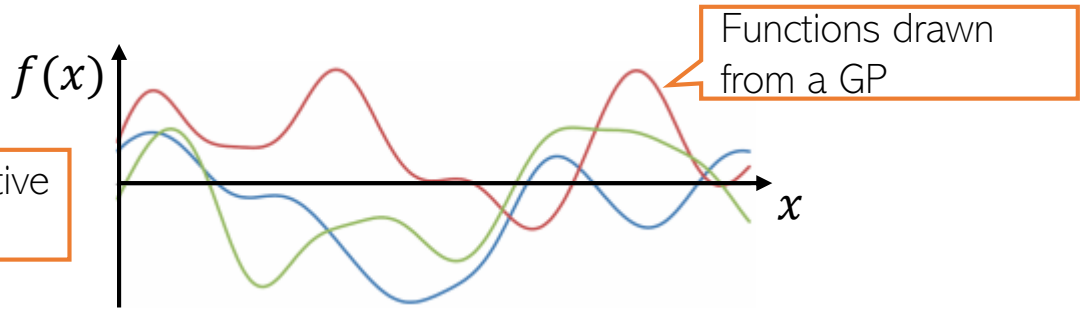
- A Gaussian Process (GP) defines a distribution over functions and is denoted as

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)

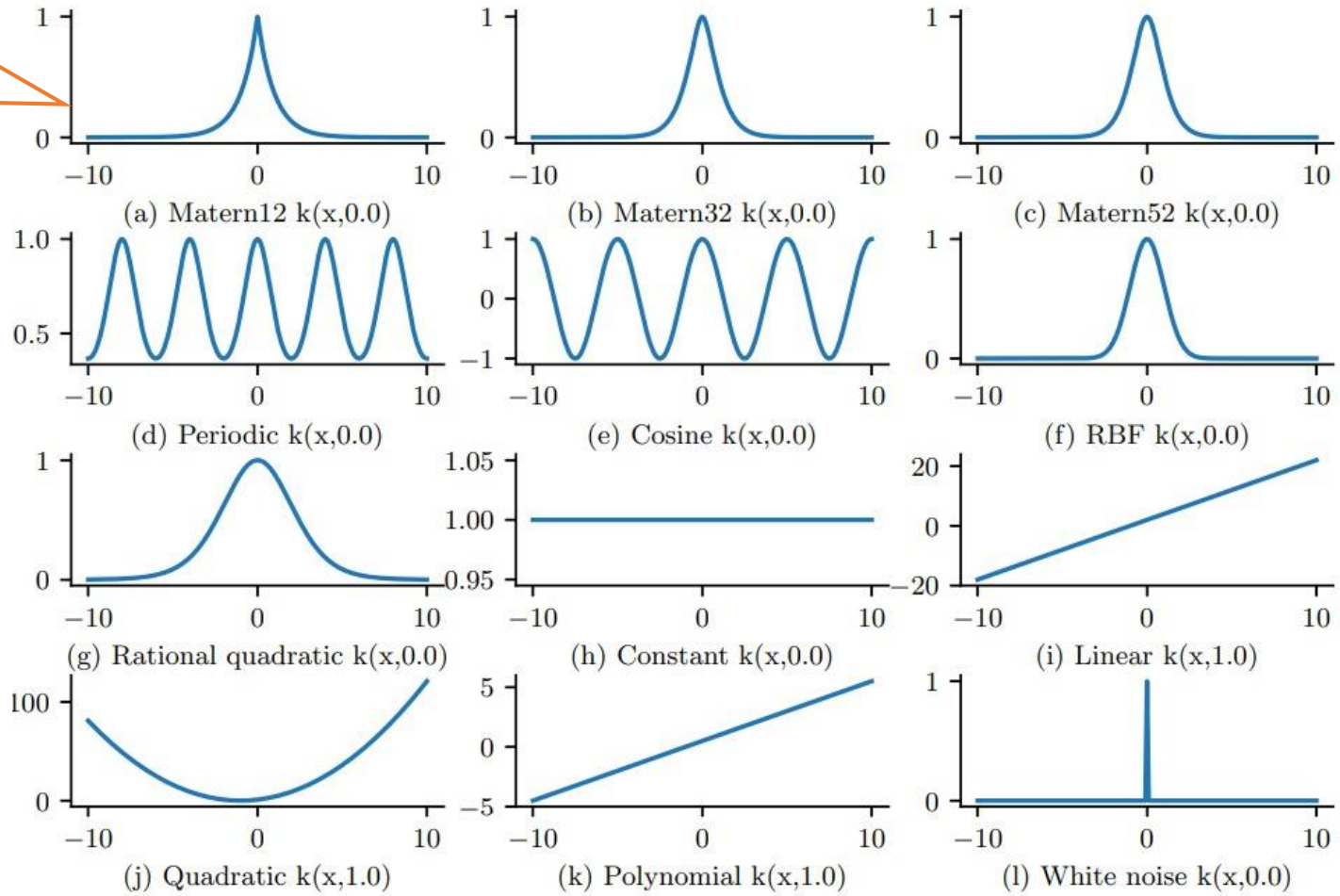


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

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

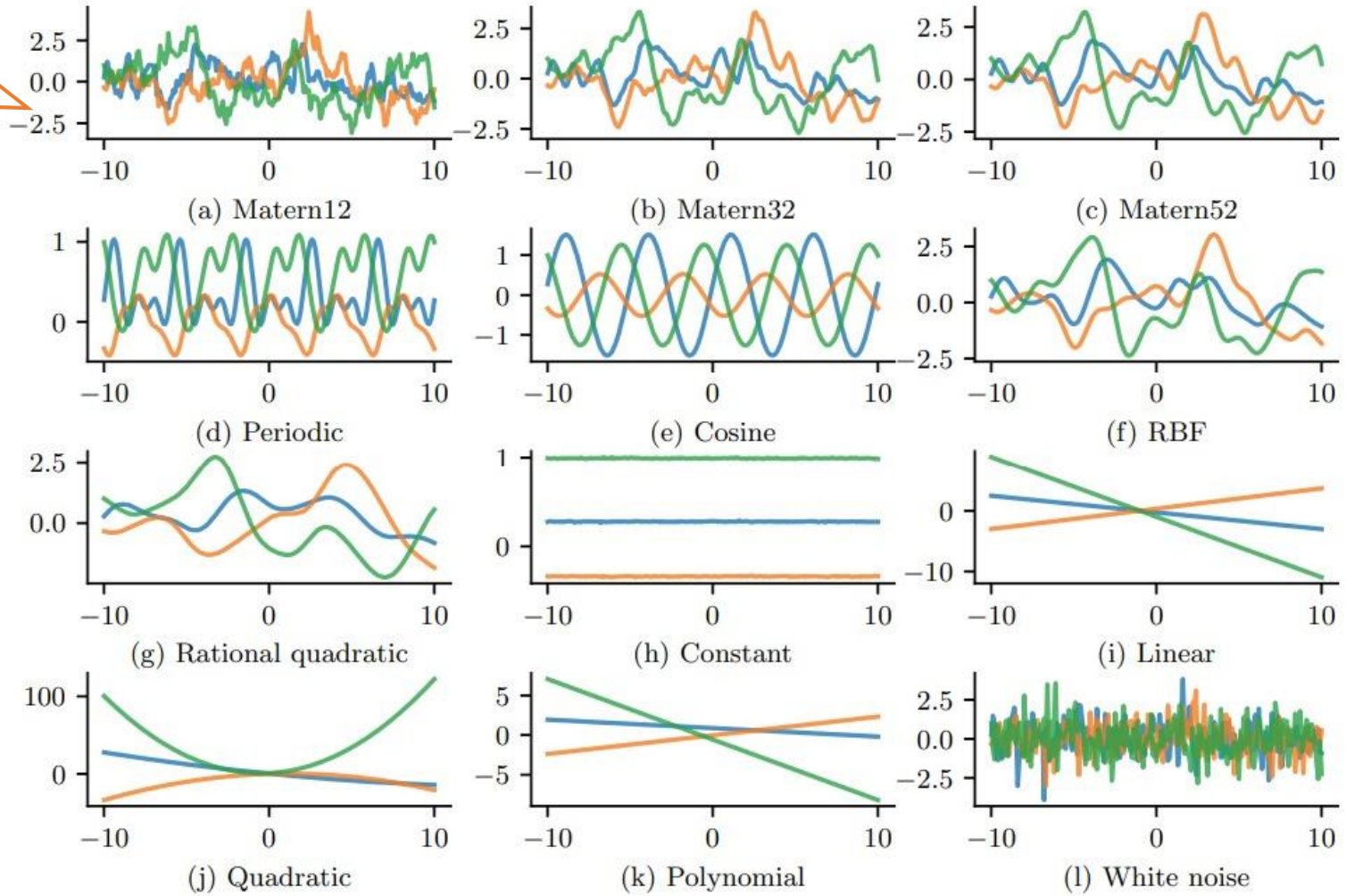


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

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

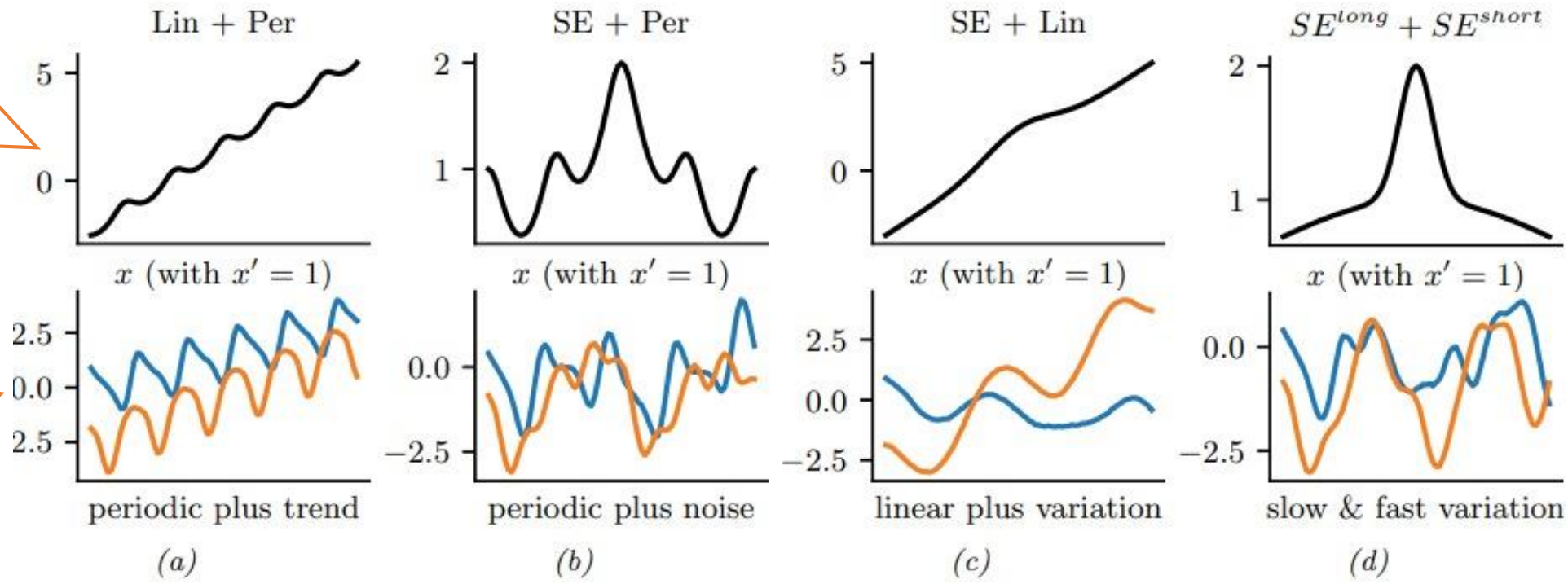


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

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

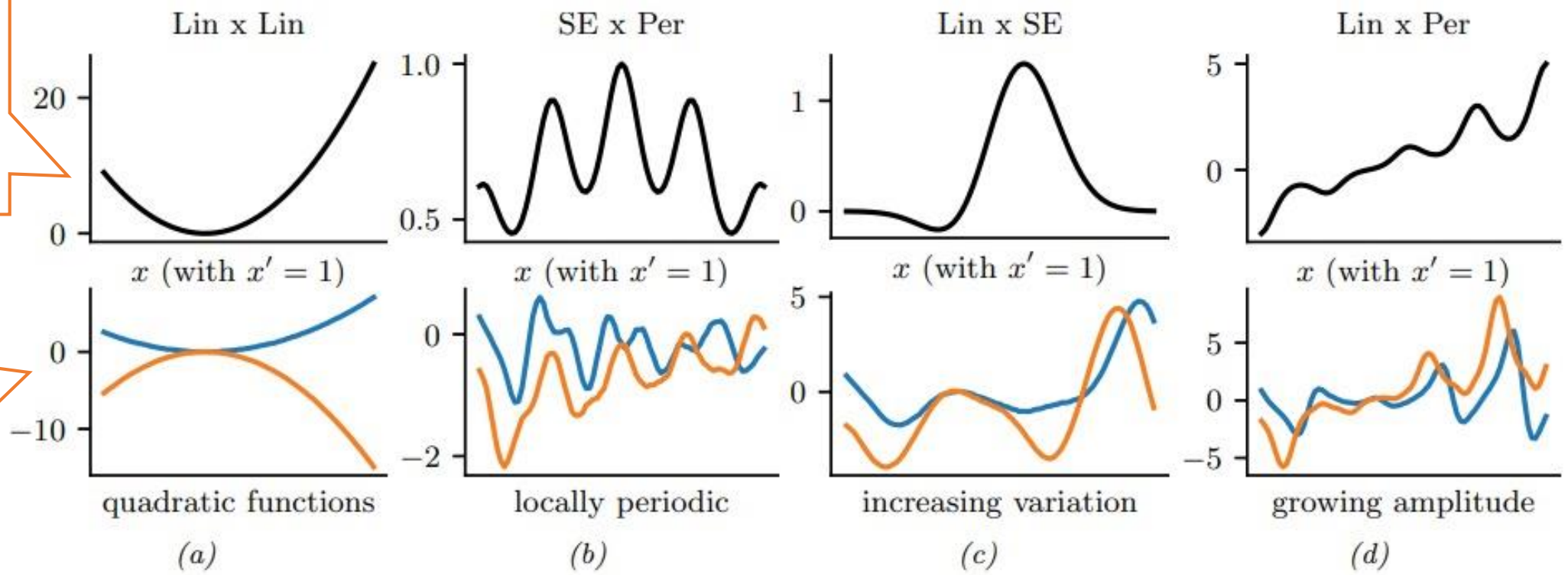


Fig source: Probabilistic Machine Learning - Advanced Topics (Murphy, 2023)

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

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

$N \times 1$ $\quad$ $N \times 1$ $\quad$ $N \times N$

- 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

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

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

Conditional Distribution of a Gaussian

$$\begin{bmatrix} a \\ b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix} \right)$$

$$p(b|a) = \mathcal{N}\left(\mu_b + \Sigma_{ba}\Sigma_{aa}^{-1}(a - \mu_a),\ \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab}\right)$$

Show how. Left as exercise

Sampling in GPs

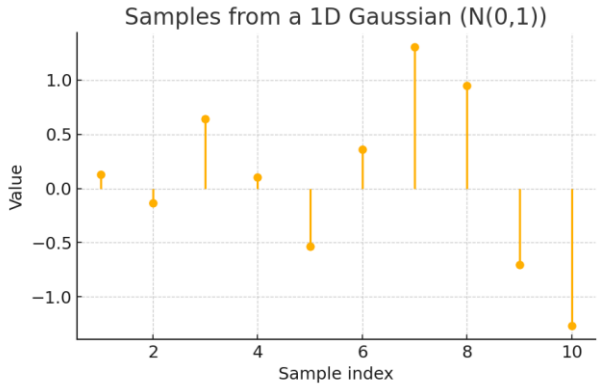
- Pick input points X (e.g., 50 evenly spaced points in [0,10]).
- Compute covariance matrix K using your kernel $k(x, x')$

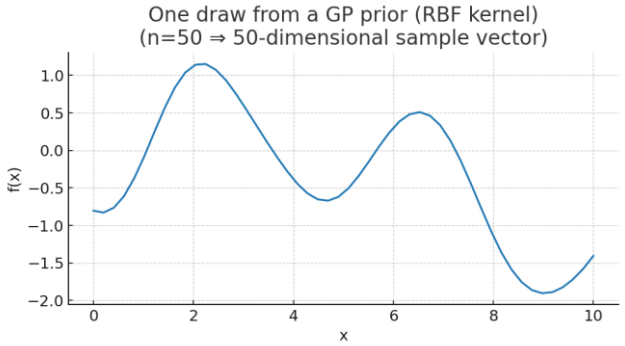
$$k_{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

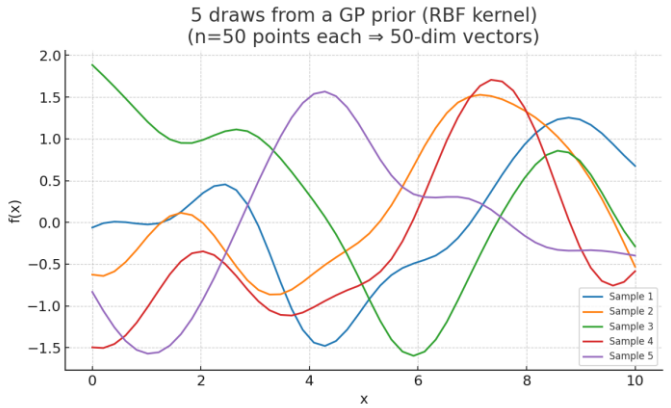
- Draw a sample from the multivariate normal:

$$f(X) \sim \mathcal{N}(0, K)$$
$$f(X) = [f(x_1), f(x_2), \ldots, f(x_n)]^\top$$

- Plot the sampled values against X.
- Each sample corresponds to **one function** from the GP prior.







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

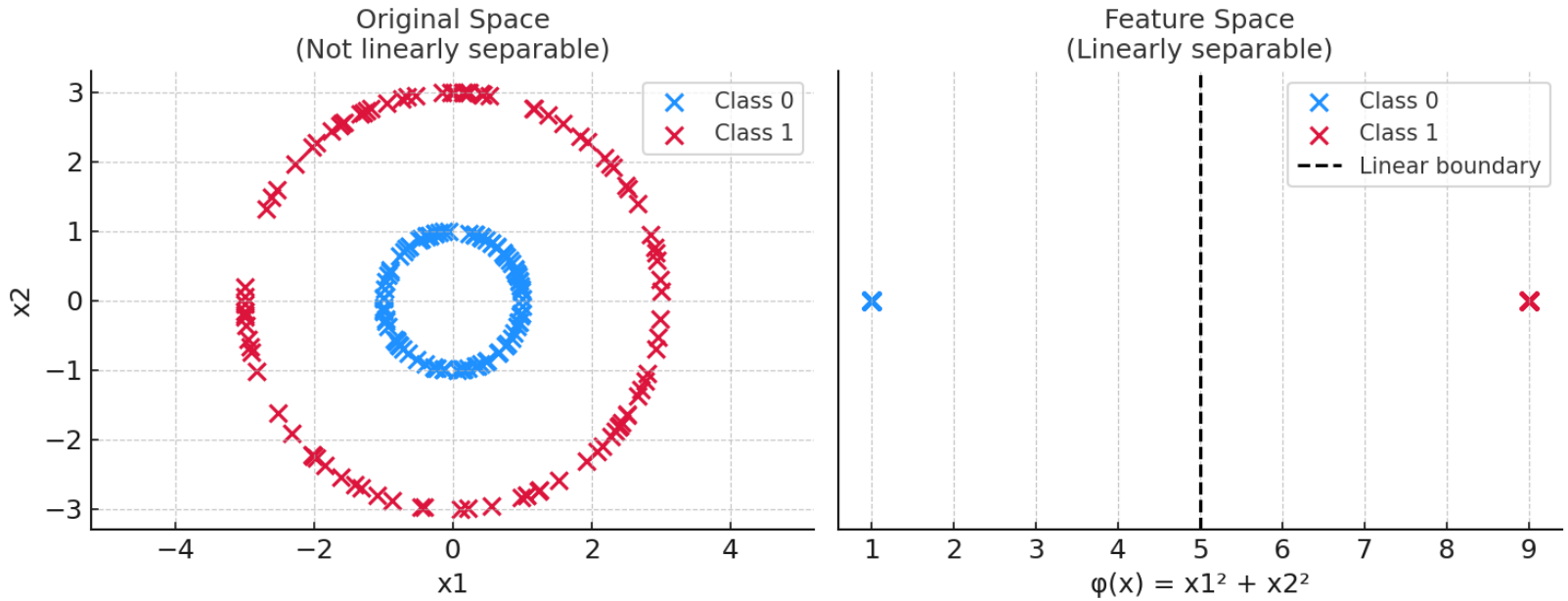
- Advantage: GP also gives the score's variance $\hat{\sigma}^2 = \kappa(x_*, x_*) - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*$

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

Detour: Kernel Methods

- Many datasets are not linearly separable
- **Kernel trick:** implicitly map data into higher-dimensional space
- Kernel function: $K(x, z) = \phi(x) \cdot \phi(z)$

$$\phi(x) = x_1^2 + x_2^2$$

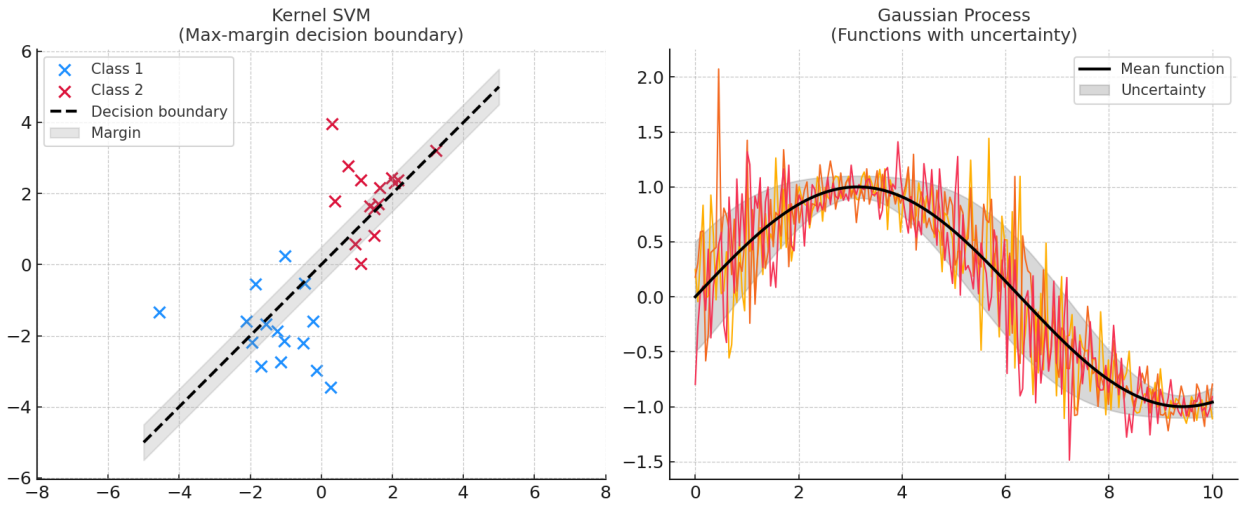


Kernel SVM

- Support Vector Machine: finds max-margin separating boundary
- Works well with kernels for non-linear decision boundaries
- Key properties:
  - Discriminative model (focuses on boundary)
  - Deterministic outputs (class/margin score)
  - Kernel defines shape of boundary

Gaussian process

- Each set of points → multivariate normal distribution
- Key properties:
  - Generative (Bayesian) model
  - Probabilistic predictions (mean + uncertainty)
  - Kernel encodes prior assumptions (smoothness, periodicity)
- Difference from SVM:
  - SVM → deterministic boundary
  - GP → distribution of plausible functions



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 $\boldsymbol{y}_*$ 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 $\mu = \mathbf{0}$, the hyperparams of GP are cov/kernel function hyperparams

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

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

Different RBF kernel bandwidth γ_d for each feature

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

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

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

- 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

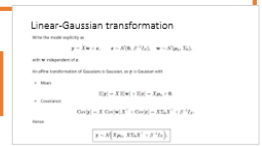
Weight Space View vs Function Space View

- 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

Linear-Gaussian transformation

Write the model explicitly as

$$\mathbf{y} = X\mathbf{w} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0},\, \beta^{-1} I_N), \quad \mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_0,\, \Sigma_0),$$

with $\mathbf{w}$ independent of $\boldsymbol{\varepsilon}$.

An affine transformation of Gaussians is Gaussian, so $\mathbf{y}$ is Gaussian with

- Mean:

$$\mathbb{E}[\mathbf{y}] = X\,\mathbb{E}[\mathbf{w}] + \mathbb{E}[\boldsymbol{\varepsilon}] = X\boldsymbol{\mu}_0 + \mathbf{0}.$$

- Covariance:

$$\mathrm{Cov}[\mathbf{y}] = X\ \mathrm{Cov}[\mathbf{w}]\, X^\top + \mathrm{Cov}[\boldsymbol{\varepsilon}] = X\Sigma_0 X^\top + \beta^{-1} I_N.$$

Hence

$$\boxed{\mathbf{y} \sim \mathcal{N}\Big(X\boldsymbol{\mu}_0,\ X\Sigma_0 X^\top + \beta^{-1} I_N\Big).}$$

GP: A Visualization

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

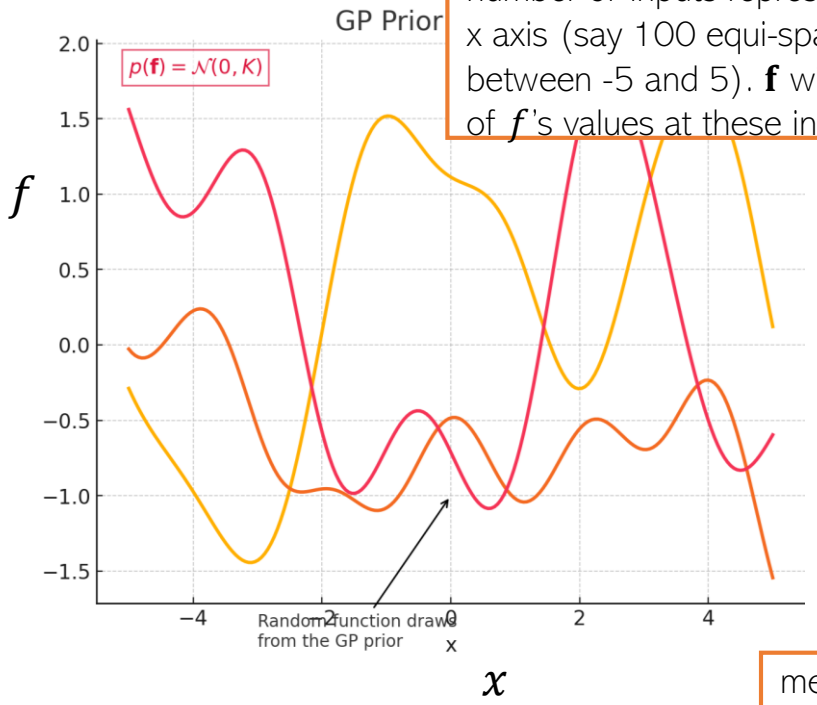
- Assume zero mean function and a squared exponential kernel

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.

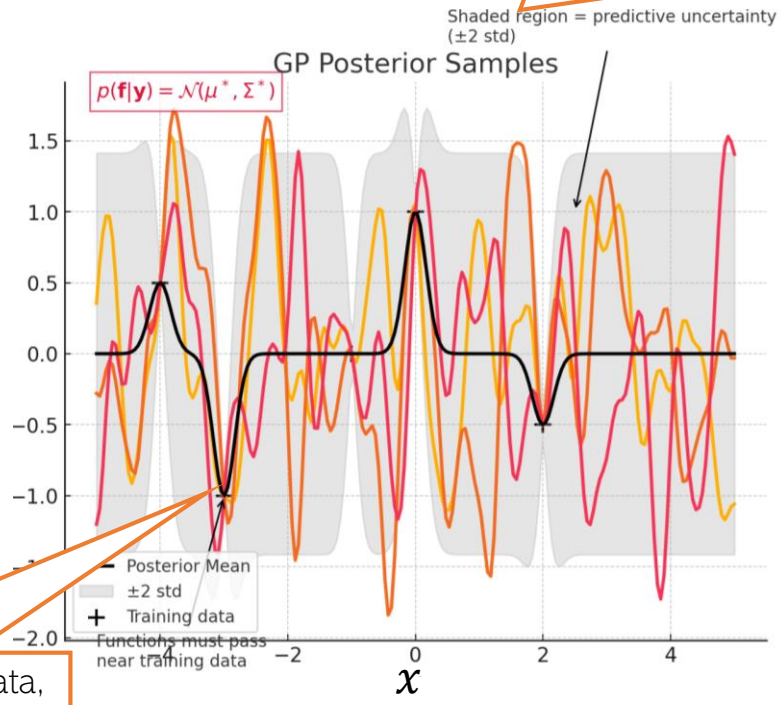
Shaded area shows the predictive uncertainty for each of the test inputs (+/- 2 std)

$\mathbf{K}$ 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

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., $y_i = f_i$) .



$y = f$



mean shifted toward the training data, variance reduced around data

Figure courtesy: MLAPP (Murphy)

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

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

- GP models often require matrix inversions (e.g., in marginal-likelihood 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

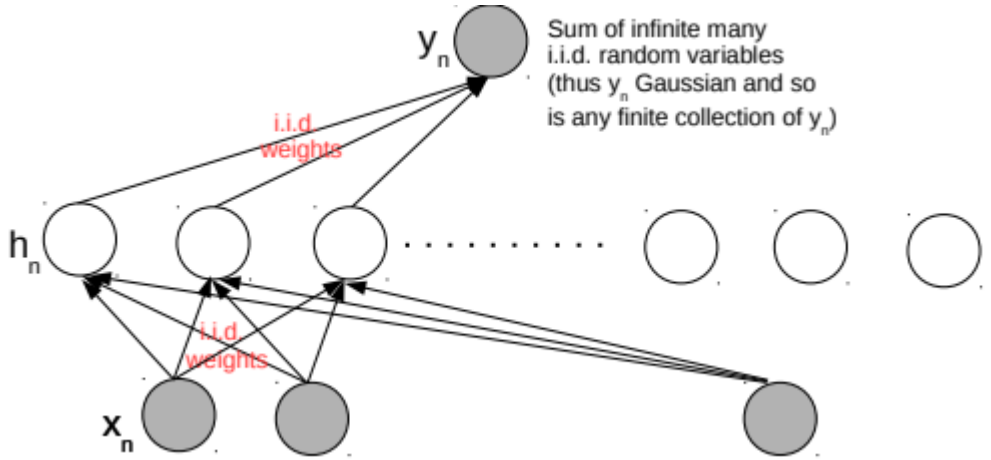
$M \ll N$ pseudo-inputs and pseudo-outputs

- 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)

Infinite Width NNs converge to GPs

1. Neural network setup

Consider a simple feedforward neural network (1 hidden layer, scalar output):

$$f(x) = \frac{1}{\sqrt{H}} \sum_{i=1}^{H} v_i \, \phi(w_i^\top x + b_i),$$

- H: number of hidden units.
- w_i, b_i: weights and biases of hidden layer.
- v_i: weights to the output.
- $\phi(\cdot)$: activation function.
- Assume all weights are i.i.d. Gaussian:

$$w_i \sim \mathcal{N}(0, \sigma_w^2 I), \quad b_i \sim \mathcal{N}(0, \sigma_b^2), \quad v_i \sim \mathcal{N}(0, \sigma_v^2).$$

Infinite Width NNs converge to GPs

2. Distribution over outputs as $H \to \infty$

- For fixed input x, $f(x)$ is a **sum of many i.i.d. random terms**.
- By the **Central Limit Theorem (CLT)**, as $H \to \infty$, $f(x)$ becomes Gaussian distributed.

Now, for multiple inputs $(x_1, \ldots, x_N)$, the joint distribution of $(f(x_1), \ldots, f(x_N))$ is **multivariate Gaussian** because each pair of outputs shares weights in common.

Thus, in the infinite-width limit:

$$f(x) \sim \mathrm{GP}(0, k(x, x')),$$

where the kernel k comes from the covariance structure of the NN.

Infinite Width NNs converge to GPs

3. The induced GP kernel

The kernel arises from the expectation over the random weights:

$$k(x, x') = \mathbb{E}\big[f(x) f(x')\big].$$

For the 1-hidden-layer network:

$$k(x, x') = \sigma_v^2 \, \mathbb{E}_{w,b}\big[\phi(w^\top x + b)\, \phi(w^\top x' + b)\big].$$

- Different activations ϕ give different kernels:
  - Linear $\phi(z) = z$ → linear kernel.
  - ReLU → ArcCos kernel.
  - Tanh → "Neal kernel" (from Neal, 1996).

This is sometimes called the **Neural Network Gaussian Process (NNGP)** kernel.

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)

References

- **Chapter 18**, Kevin Murphy, Probabilistic Machine Learning: Advanced Topics, MIT Press, 2022 (freely available online)

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