Introduction to Gaussian Processes
(Kernel Methods meet Bayesian Learning)

CS772A: Probabilistic Machine Learning

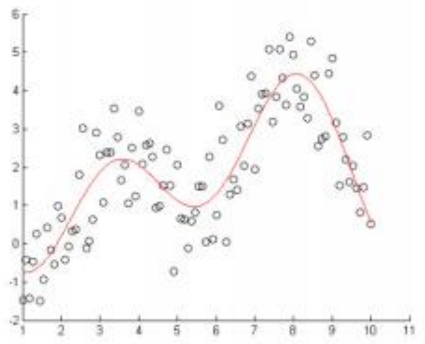
Piyush Rai

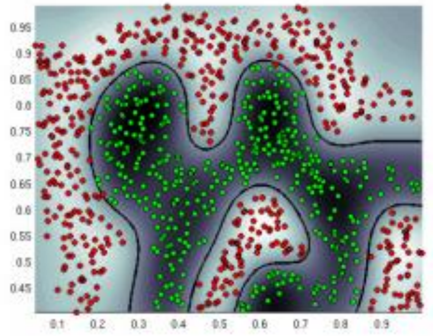
Linear Models and Their Limitations

- Consider learning to map an input $\boldsymbol{x}$ to the output $\boldsymbol{y}$
- We've seen various discriminative models (linear and generalized linear models)

$$p(y|\boldsymbol{w},\boldsymbol{x}) = \mathcal{N}(y|\boldsymbol{w}^\top\boldsymbol{x},\beta^{-1}) \quad \text{(Linear Regression)}$$
$$p(y|\boldsymbol{w},\boldsymbol{x}) = [\sigma(\boldsymbol{w}^\top\boldsymbol{x})]^y[1-\sigma(\boldsymbol{w}^\top\boldsymbol{x})]^{1-y} \quad \text{(Logistic Regression)}$$
$$p(y|\boldsymbol{w},\boldsymbol{x}) = \text{ExpFam}(\underbrace{\boldsymbol{w}^\top\boldsymbol{x}}_{\text{Natural param of canonical GLM}}) \quad \text{(Generalized Linear Model)}$$

- These have limited expressive power – can't learn nonlinear patterns


Nonlinear Regression


Nonlinear Classification

Learning Nonlinear Functions

- Assume the input to output relationship to be modeled by a nonlinear function $\boldsymbol{f}$

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

In all of these, the linear score $\boldsymbol{w}^\top\boldsymbol{x}$ has been replaced by a nonlinear function $\boldsymbol{f(x)}$

- Would like to model this function in a probabilistic/Bayesian manner
  - Nonlinearity + all the benefits of probabilistic/Bayesian modeling

Example: Assuming x is scalar, $\phi(x) = [1, x, x^2, \ldots, x^k]$, for some k

- Some ways to achieve this
  - Ad-hoc: Manually define nonlinear features $\boldsymbol{\phi(x)}$ + train Bayesian linear model
  - Ad-hoc: Use a pre-trained deep neural net to extract features $\boldsymbol{\phi(x)}$ + train Bayesian linear model
  - Bayesian Neural Networks (later)
  - Gaussian Processes (a Bayesian approach to kernel based nonlinear learning; today)

Gaussian Process

Any choice of the GP covariance function has an associated feature map $\phi(x)$ for the inputs x

Hmmm.. So GPs look like kernel methods with all the benefits of probabilistic/Bayesian modeling

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

Akin to how we define a Gaussian distribution over scalars/vectors, defined by a mean and variance/covariance matrix

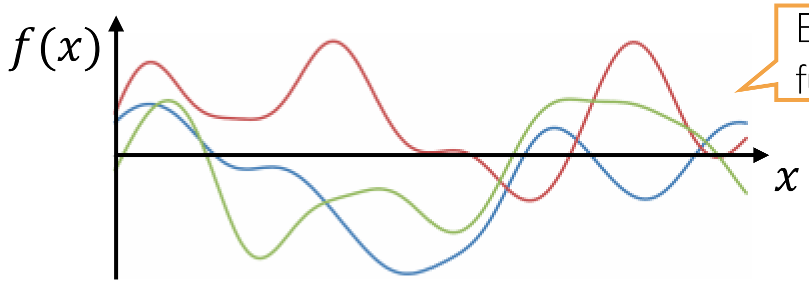
Mean Function

Covariance Function

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

Can also think of a function as an infinite dimensional vector of function's values at different inputs (x), i.e.,
$$\boldsymbol{f} = [f(x_1), f(x_2), f(x_3), \dots]$$

- Every draw/sample from $\mathcal{GP}(\mu, \kappa)$ will give a random function f



Each of these curves is a random function drawn from the GP

μ and κ can be pre-defined or can even be learned

Mean Function $\mu(.)$ defines the "average" function looks like:
$$\mu(x) = \mathbb{E}[f(x)]$$

Covariance Function $\kappa(.,.)$ defines similarity between pairs of inputs and controls the shape of these curves (also needs to be pos-sem-def)

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

Can concisely write it as $p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$

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

$N \times 1$ vector of f's values: $\mathbf{f}$

$N \times 1$ mean vector: $\boldsymbol{\mu}$

$N \times N$ cov/kernel matrix (PSD): $\mathbf{K}$

Very useful property for making predictions: Knowing f's value at some N "training" inputs, say, $x_1, x_2, \dots, x_N$, we can easily compute its value at a new test input x_*, using the Gaussian joint-to-conditional formula

Predicting using GP

The results we saw here relating the score f_* to $\mathbf{f}$ will still hold ☺

We just need to use a likelihood model for y_n to handle such "noisy settings (will see soon)

For example
$p(y_n|f_n) = \mathcal{N}(y_n|f_n, \beta^{-1})$
$p(y_n|f_n) = \text{Bernoulli}(y_n|\sigma(f_n))$

The setting considered on this slide is the "noiseless" setting where the response y_n is simply given by $y_n = f_n = f(x_n)$. More realistic settings with have each output y_n as a transformation of a "score" given by GP: $f_n = f(x_n)$

- We have already seen that

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

- Let's assume the mean function $\mu(x) = 0$, thus $\boldsymbol{\mu} = \mathbf{0}$ and $p(\mathbf{f}) = \mathcal{N}(\mathbf{0}, \mathbf{K})$
- Assume we know $\mathbf{f} = [f(x_1), f(x_2), \dots, f(x_N)]$ and want to compute $f(x_*)$
- Due to the GP property, joint distribution of f's values will always be Gaussian

$$p\left(\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix}\right) = \mathcal{N}\left(\underbrace{\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}}_{(N+1)\times 1 \text{ vector}}, \underbrace{\begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^\top & \kappa(x_*, x_*) \end{bmatrix}}_{(N+1)\times(N+1) \text{ matrix}}\right)$$

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

$N \times 1$ vector of similarities of x_* with each of the N training inputs

PPD without computing posterior ☺

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

Form of prediction similar to kernel regression but also get variances σ_*^2

- Exercise: Show that predictive mean $\mu_* = \sum_{i=1}^N \beta_i f_i = \sum_{i=1}^N \alpha_i \kappa(x_i, x_*)$

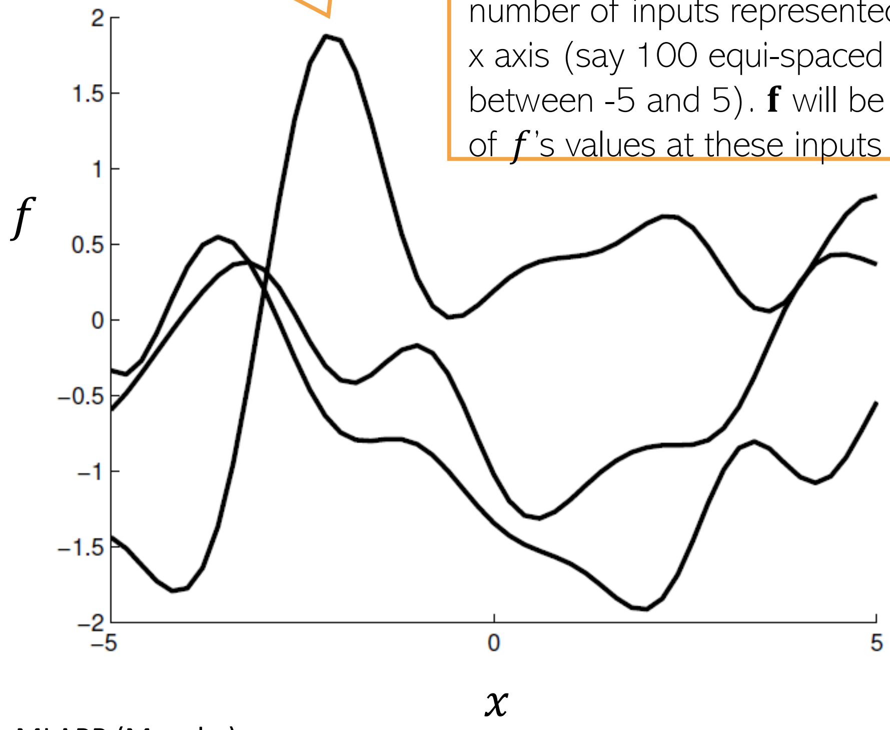
GP: A Visualization

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

- 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 the GP posterior $p(\mathbf{f}|\mathbf{f}_{train})$ which is also a Gaussian

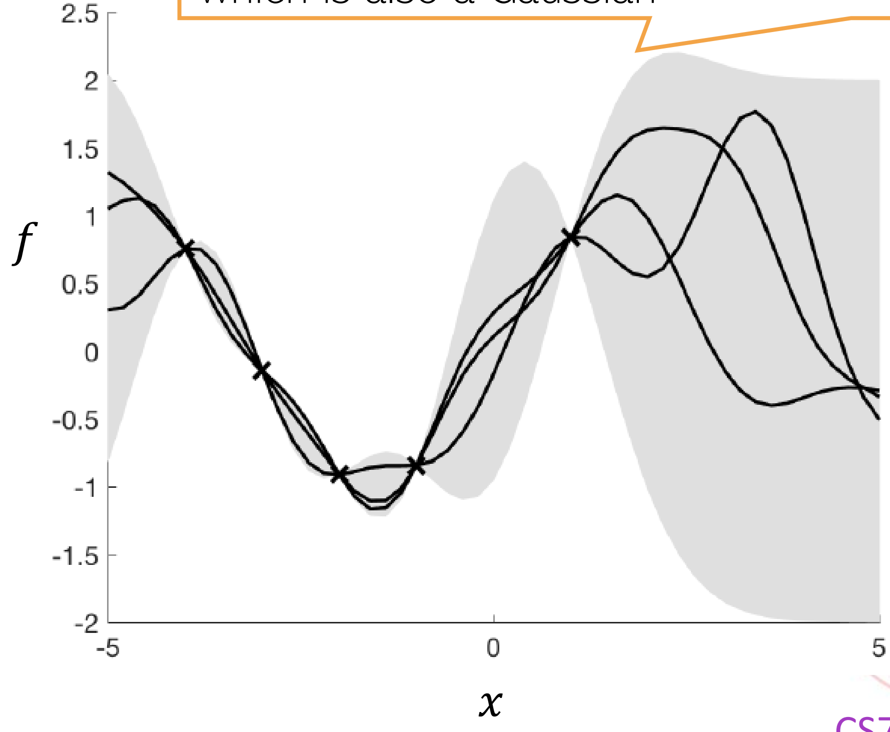


Figure courtesy: MLAPP (Murphy)

GP for Noisy Setting: Regression (Gaussian Lik.)

- For Gaussian lik, we can get PPD $p(y_*|\boldsymbol{y})$ without computing the GP posterior $p(\boldsymbol{f}|\boldsymbol{y})$
- Note that, in this case, the marginal likelihood is also a Gaussian

Also useful when learning hyperparams of the GP covariance/kernel

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

- The joint distribution of the training $\boldsymbol{y}$ and test response y_* is also a Gaussian

Note: All hyperparams assumed to be known

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

Identical to the noiseless case except the additional β^{-1} term on the diagonal

- Using the above, we can easily obtain $p(y_*|\boldsymbol{y})$ using Gaussian properties

Weighted average of the training responses

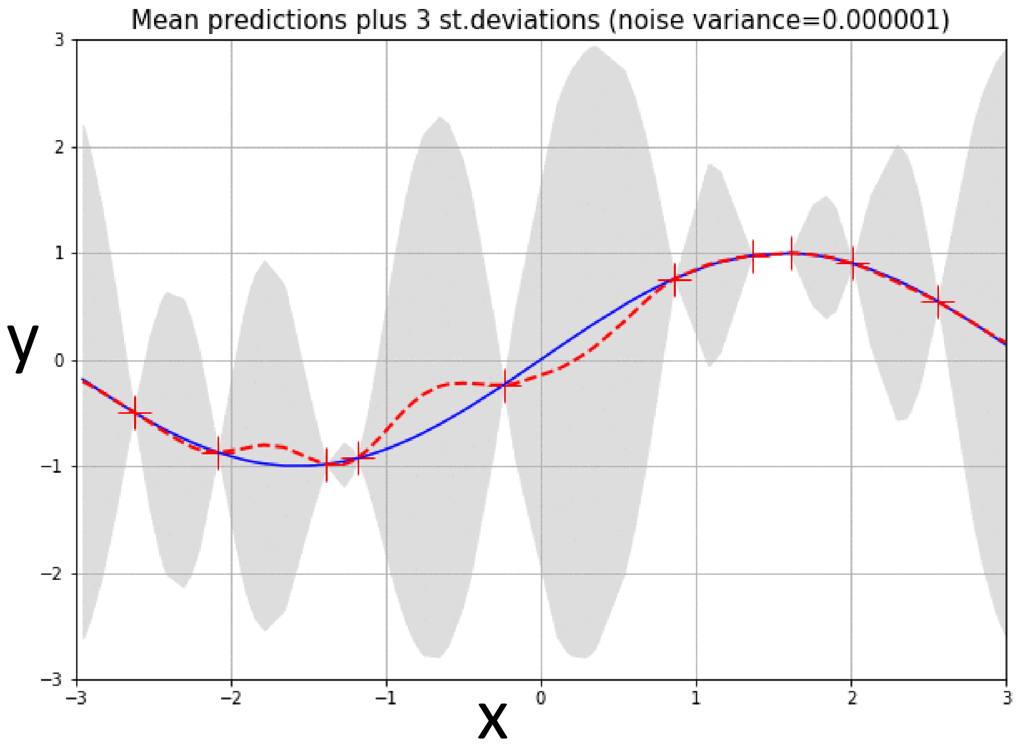
$$p(y_*|\boldsymbol{y}) = \mathcal{N}(\mathbf{k}_*^\top\mathbf{C}_N^{-1}\boldsymbol{y}, \kappa(x_*,x_*) - \mathbf{k}_*^\top\mathbf{C}_N^{-1}\mathbf{k}_* + \beta^{-1}) = \mathcal{N}(\mu_*,\sigma_*^2)$$

μ_* has a similar interpretation as in the noiseless case

- This is almost identical to the expression of $p(f_*|\boldsymbol{f})$ from the noiseless case except $\mathbf{K}$ there is replaced by $\mathbf{C}_N$ and extra β^{-1} term in variance

GP Regression: An Illustration

- The figure below shows GP predictive mean and variance as noise variance changes



Blue curve: True function
Red point: Training inputs (noisy)
Red curve: Learned predictive mean
Shaded region: +/- 3 std-dev

- As expected, the predictive mean worsens and predictive variance increases as the noise variance increases

Figure courtesy: https://sandipandey.wixsite.com/simplydatascience/post/gaussian-process-regression-with-python

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 distribution of the N responses $y_1, y_2, \ldots, y_N$ is a multivariate Gaussian

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

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

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

GP for Noisy Setting: Classification and GLM

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

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

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

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

Learning Hyperparameters in GP based Models

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

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

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

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

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 $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$
- For non-Gaussian likelihoods, the marg-lik itself will need to be approximated

Coming Up

- Some aspects of GPs
  - Scalability
  - Connections with neural nets
  - Some recent advances

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

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

- 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

GP: Some Comments

- GP is sometimes referred to as a nonparametric model because
  - 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

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

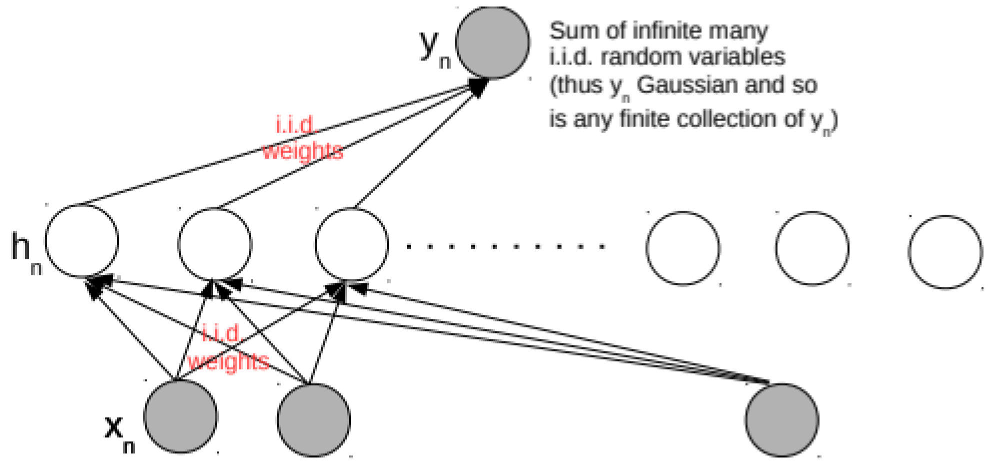
  - It implies that $f(.) = \sum_{n=1}^{N} \alpha_n k(., \mathbf{x}_n)$ which means f is written in terms of all training examples
  - Thus the representation size of f depends on the number of training examples

- In contrast, a parametric model has a size that doesn't grow with training data
  - E.g., a linear model learns a weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (D parameters, size independent of N)

- Nonparametric models more flexible since their complexity is not limited beforehand
  - Note: Methods like nearest neighbors and kernel SVMs are also nonparametric (but not Bayesian)

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

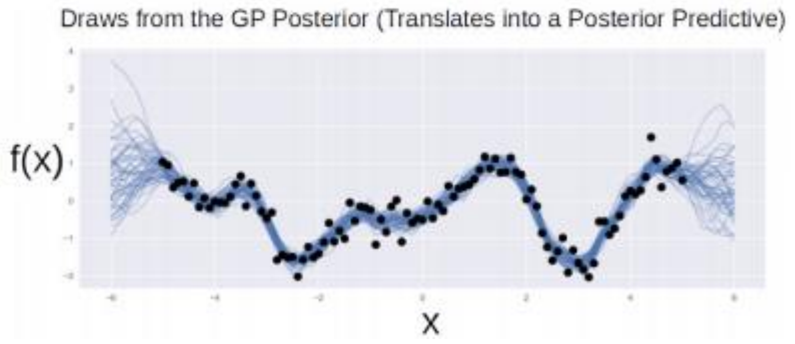

y_n
Sum of infinite many i.i.d. random variables (thus y_n Gaussian and so is any finite collection of y_n)
i.i.d weights
h_n
i.i.d weights
x_n

- 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: A Few Other Comments

- GPs 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
  - f could even define a mapping of low-dim latent variable z_n to an observation x_n

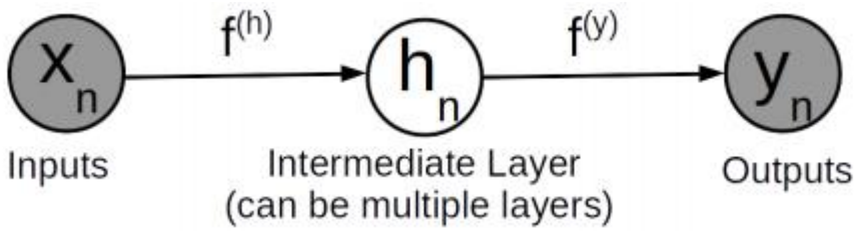
$$\boldsymbol{x}_n = f(\boldsymbol{z}_n) + \text{"noise"}$$

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

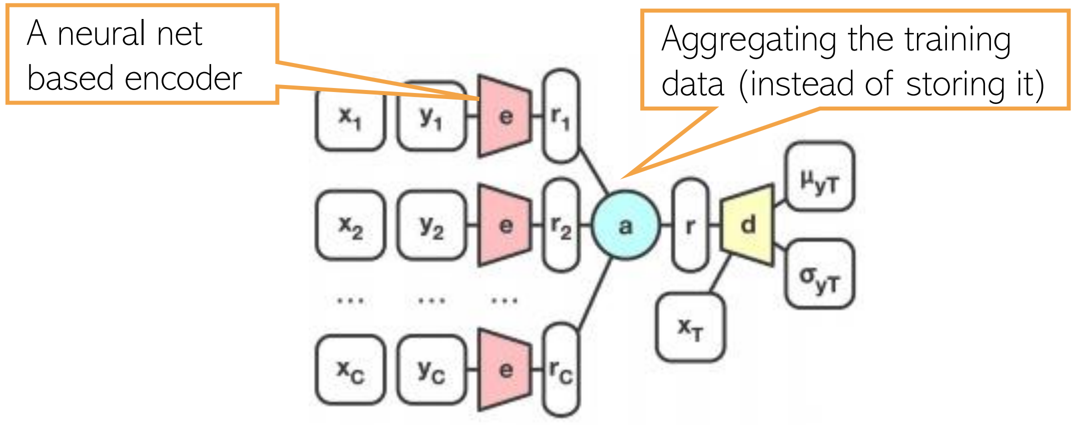
- Many mature implementations of GP exist. You may check out
  - GPyTorch (PyTorch), GPFlow (Tensorflow)
  - GPML (MATLAB), GPsuff (MATLAB/Octave)

GP: Some Other Recent Advances

- 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 in neural nets, e.g., $\boldsymbol{h} = \tanh(\boldsymbol{W}\boldsymbol{x})$
- GPs with deep kernels defined by neural nets
- Neural Processes (GP + neural nets): Faster way to do GPs


A neural net based encoder
Aggregating the training data (instead of storing it)
x_1
y_1
e
r_1
x_2
y_2
e
r_2
...
x_C
y_C
e
r_C
a
r
d
x_T
μ_{yT}
σ_{yT}