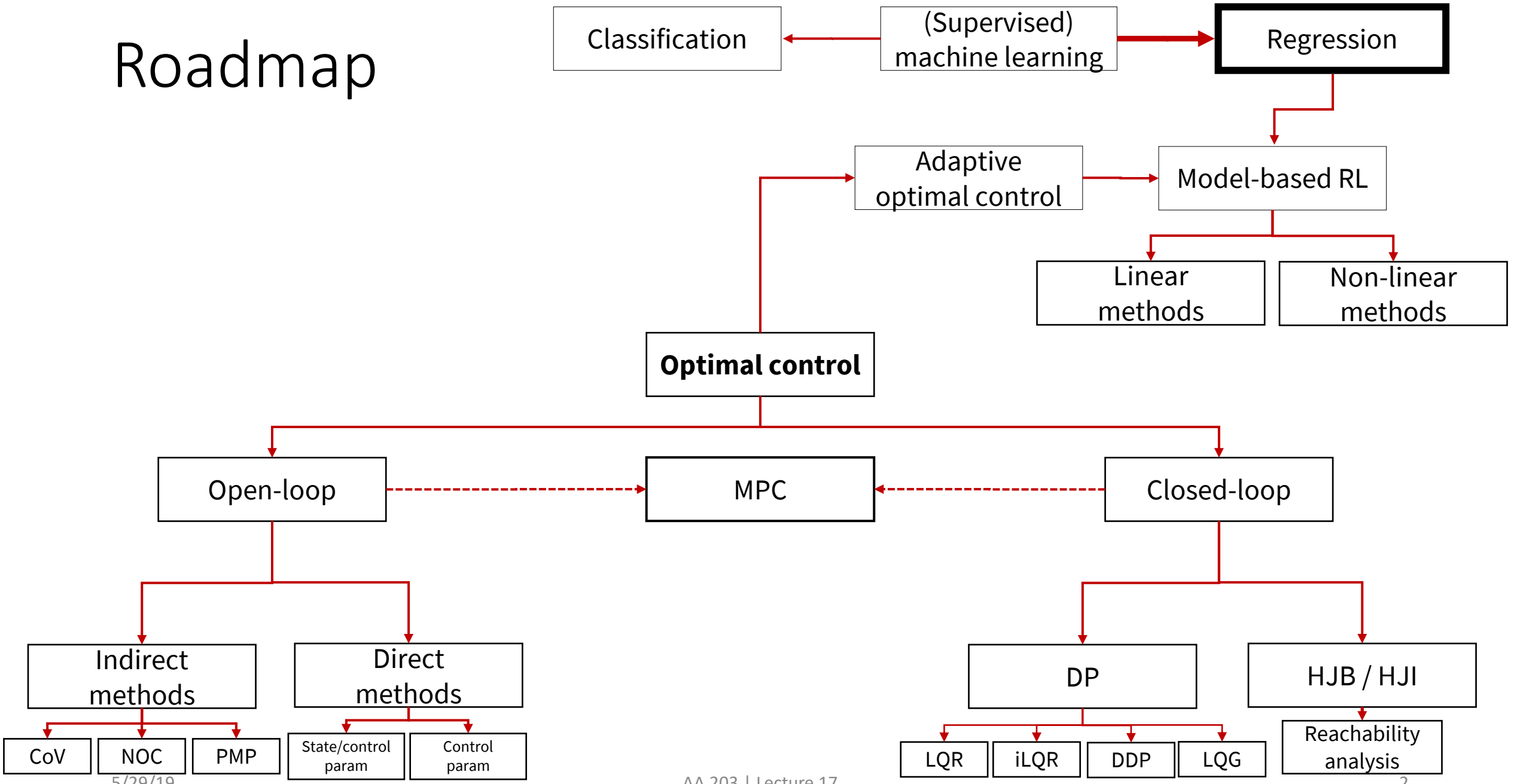
AA203
Optimal and Learning-based Control

Nonlinear Regression

Stanford University

ASL
Autonomous Systems Lab

Roadmap


Classification
(Supervised) machine learning
Regression
Adaptive optimal control
Model-based RL
Linear methods
Non-linear methods
Optimal control
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
CoV
NOC
PMP
State/control param
Control param
DP
HJB / HJI
LQR
iLQR
DDP
LQG
Reachability analysis

Linear Regression

- We will begin by revisiting linear regression (in the scalar output case, for simplicity).
- We will assume a model of the form $y_i = \boldsymbol{\theta}^T \boldsymbol{x}_i + \epsilon_i$, where ϵ_i is zero-mean Gaussian with known covariance σ_ϵ^2.
- Then, assuming data of the form $y_1, \dots, y_n, \boldsymbol{x_1}, \dots, \boldsymbol{x_n}$, we can write the least squares problem as

$$\min_{\boldsymbol{\theta}} || \boldsymbol{y} - X\boldsymbol{\theta}||^2$$

where $\boldsymbol{y} = [y_1, \dots, y_n]^T$, $X = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^T$.

Least Squares via MLE

- This minimization problem is intuitive – we want to reduce the error of our model predictions, so we minimize the error. But formally, where does it come from?
- One answer: Maximum Likelihood Estimation (MLE)
  - Define the **likelihood function**,

$$\mathcal{L}(\boldsymbol{\theta} \mid X, \boldsymbol{y}) = p(\boldsymbol{y} \mid X, \boldsymbol{\theta}) = \prod_{\boldsymbol{i=1}}^{\boldsymbol{n}} p(y_i \mid \boldsymbol{x_i}, \boldsymbol{\theta})$$

  where because we assumed zero mean gaussian errors,

$$p(y_i \mid \boldsymbol{x_i}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}^T \boldsymbol{x_i}, \sigma_{\boldsymbol{\epsilon}}^2).$$

  - This is the probability of our output variable as a function of the parameter

Least Squares via MLE

- MLE maximizes the likelihood function (i.e. the likelihood of our data).
  - For convenience, we will minimize **negative log likelihood –** the optimization problem yields the same solution
  - So, $-\log \mathcal{L}(\boldsymbol{\theta} \mid X, \boldsymbol{y}) = -\sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma_\epsilon} e^{-\frac{\left(y_i - \boldsymbol{\theta}^T \boldsymbol{x_i}\right)^2}{2\sigma_\epsilon^2}}$
  - Can drop $\log \frac{1}{\sqrt{2\pi}\sigma_\epsilon}$ from optimization problem, which yields

$$-\log \mathcal{L}(\boldsymbol{\theta} \mid X, \boldsymbol{y}) = \sum_{i=1}^{n} \frac{\left(y_i - \boldsymbol{\theta}^T \boldsymbol{x_i}\right)^2}{2\sigma_\epsilon^2}$$

  Which, ignoring the covariance scaling factor, yields the least squares problem.

Least Squares: Bayesian perspective

- MLE gives a point estimate of $\boldsymbol{\theta}$. What if we instead take a Bayesian approach, in which we specify a **prior** over $\boldsymbol{\theta}$, and compute the posterior distribution after observing data?
- This is based on **Bayes' rule**:

$$p(\boldsymbol{\theta} \mid X, \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid X, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y} \mid X)}$$

- Ignoring $p(\boldsymbol{y} \mid X)$ for now, we will fix a Gaussian prior, $\boldsymbol{\theta} \sim \mathcal{N}(0, \Sigma_0)$, thus

$$\begin{aligned} p(\boldsymbol{y} \mid X, \boldsymbol{\theta})p(\boldsymbol{\theta}) &= e^{-\frac{1}{2\sigma_\epsilon}(\boldsymbol{y}-X^T\boldsymbol{\theta})^T(\boldsymbol{y}-X^T\boldsymbol{\theta})} e^{-\frac{1}{2}\boldsymbol{\theta}^T\Sigma_0^{-1}\boldsymbol{\theta}} \\ &= e^{-\frac{1}{2}(\boldsymbol{\theta}-\overline{\boldsymbol{\theta}})^T(\boldsymbol{\sigma_\epsilon^{-1}} X\, X^T+\Sigma_0^{-1})(\boldsymbol{\theta}-\overline{\boldsymbol{\theta}})} \end{aligned}$$

where $\overline{\boldsymbol{\theta}} = \sigma_\epsilon^{-2}\left(\sigma_\epsilon^{-2}XX^T + \Sigma_0^{-1}\right)^{-1}X\boldsymbol{y}$

Bayesian Least Squares

- So, the posterior over $\boldsymbol{\theta}$ is a Gaussian of the form

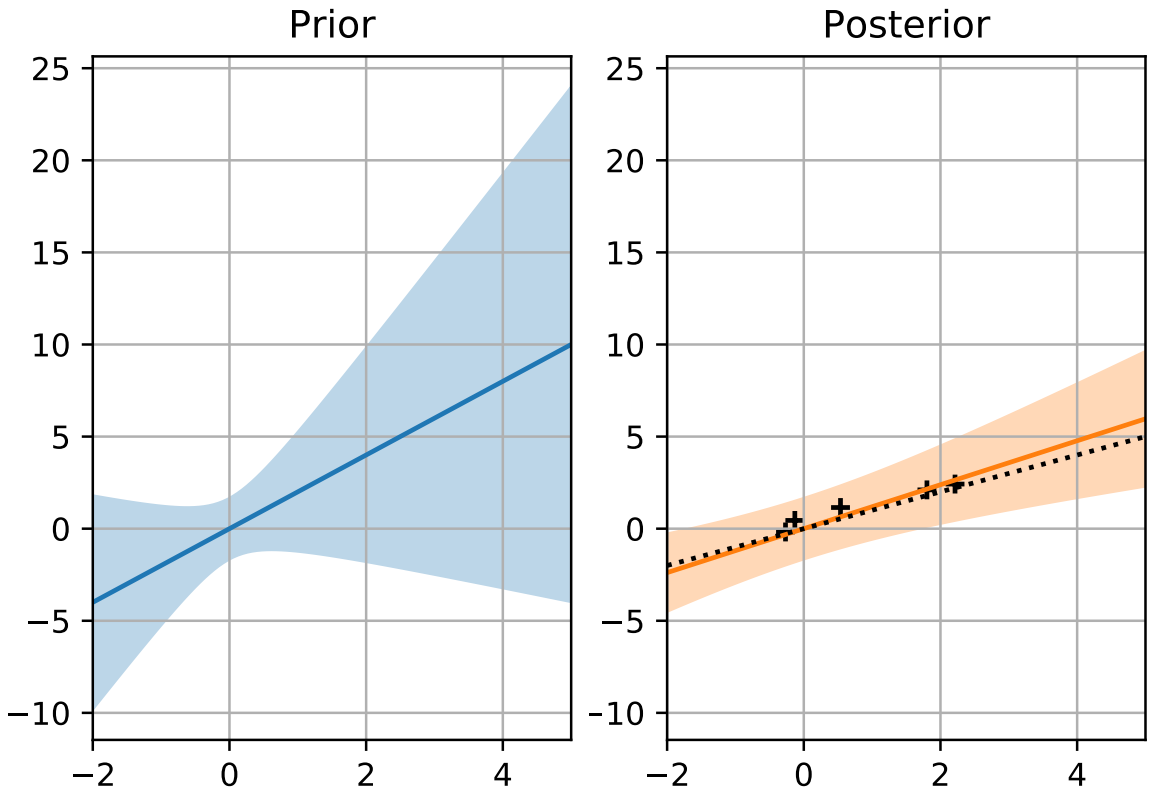
$$p(\boldsymbol{\theta} \mid X, \boldsymbol{y}) = \mathcal{N}(\sigma_\epsilon^{-1}\Sigma_n X\boldsymbol{y}, \Sigma_n)$$

  where $\Sigma_n = \left(\sigma_\epsilon^{-1}XX^T + \Sigma_{\mathbf{0}}^{-1}\right)^{-1}$

- What just happened? If the prior is Gaussian and the data likelihood is Gaussian, the posterior over the weights is Gaussian
- What does this mean about the predictive distribution, $p(\boldsymbol{y}_* \mid \boldsymbol{x}_*, X, \boldsymbol{y})$?
- Note that for constant $\mathbf{a}$, $E[\boldsymbol{a}^T\boldsymbol{x}] = \boldsymbol{a}^T E[\boldsymbol{x}]$ and $Var(\boldsymbol{a}^T x) = \boldsymbol{a}^T Var(x)\boldsymbol{a}$
- Thus, $p(\boldsymbol{y}_* \mid \boldsymbol{x}_*, X, \boldsymbol{y}) = \mathcal{N}(\sigma_\epsilon^{-1}\boldsymbol{x}_*^T\Sigma_n X\boldsymbol{y}, \boldsymbol{x}_*^T\Sigma_{\boldsymbol{n}}\boldsymbol{x}_* + \sigma_\epsilon^2)$

Bayesian Least Squares

- What features does this model have?
  - Our posterior over weights is Gaussian
  - Our posterior predictive distribution is Gaussian
  - We have a closed form representation of the posterior density $p(\boldsymbol{y}_* \mid \boldsymbol{x}_*, X, \boldsymbol{y})$; **useful for capturing model confidence**!
  - Note that in this derivation, we could replace $\boldsymbol{x}$ with $\boldsymbol{\phi}(\boldsymbol{x})$, derivation proceeds the same
  - Note: Tikhonov regularization (e.g. ridge regression) is a consequence of this model!



BLR with nonlinear basis functions

- Let $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\boldsymbol{x}_*)$, and $f_* = \boldsymbol{\phi}_*^T \boldsymbol{\theta}$. Then, $f_* \mid \boldsymbol{x}_*, X, \boldsymbol{y}$ is Gaussian with

$$E[f_* \mid \boldsymbol{x}_*, X, \boldsymbol{y}] = \boldsymbol{\phi}_*^T \Sigma_0 \Phi (K + \sigma_\epsilon^2 I)^{-1} \boldsymbol{y}$$
$$Var(f_* \mid \boldsymbol{x}_*, X, \boldsymbol{y}) = \boldsymbol{\phi}_*^T \Sigma_0 \boldsymbol{\phi}_* - \boldsymbol{\phi}_*^T \Sigma_0 \Phi (K + \sigma_\epsilon^2 I)^{-1} \Phi^T \Sigma_0 \boldsymbol{\phi}_*$$

  where $K = \Phi^T \Sigma_0 \Phi$

- This is a mess, but one important fact: the basis function $\boldsymbol{\phi}$ always enters through $\boldsymbol{\phi}_*^T \Sigma_0 \boldsymbol{\phi}_*$, $\boldsymbol{\phi}_*^T \Sigma_0 \Phi$, or $\Phi^T \Sigma_0 \Phi$, so entries are always of the form $\boldsymbol{\phi}(\boldsymbol{x})^T \Sigma_0 \boldsymbol{\phi}(\boldsymbol{x}')$
- We will define **kernel function** $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \Sigma_0 \boldsymbol{\phi}(\boldsymbol{x}')$
- Kernel trick: we can skip computing $\boldsymbol{\phi}(\boldsymbol{x})$, only ever have to compute $k(\boldsymbol{x}, \boldsymbol{x}')$!

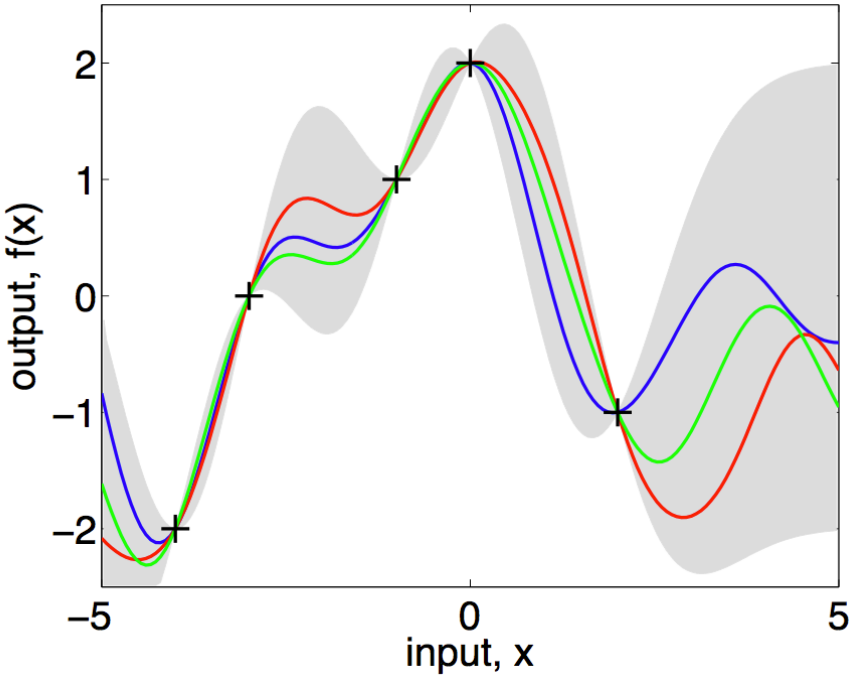
Gaussian processes

- What does the kernel trick buy us? We can use kernels that correspond to very high dimensional (even infinite-dimensional) basis functions, since we never have to compute them!

- Example: squared exponential kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{1}{2l^2}||\boldsymbol{x} - \boldsymbol{x}'||^2}$$

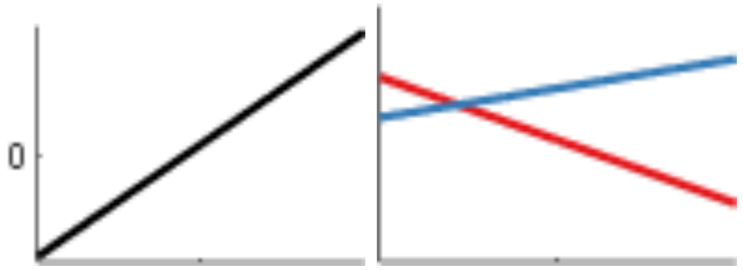
  - Often a default for GP regression
  - Captures smooth local correlations
  - Corresponds to infinite-dimensional basis function



Choice of kernel function

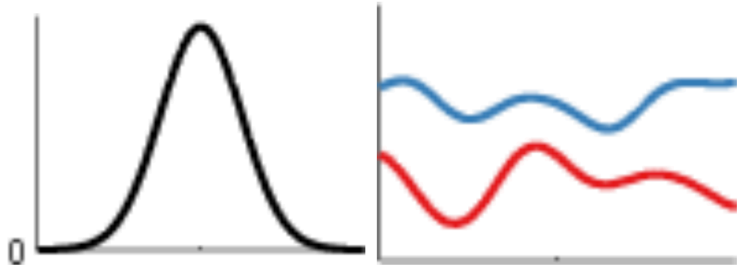
- The choice of kernel function allows us to encode prior knowledge into regression problem
- Can we choose any function as a kernel function?
  - Short answer: no. Kernel must correspond to inner product between some basis functions
- Let k_1, k_2 be valid kernels. Then, the following are also valid kernels:
  - $a\ k_1(\boldsymbol{x}, \boldsymbol{x}') + b\ k_2(\boldsymbol{x}, \boldsymbol{x}')$for $a, b \geq 0$
  - $k_1(\boldsymbol{x}, \boldsymbol{x}')\ k_2(\boldsymbol{x}, \boldsymbol{x}')$
  - $k_1(f(\boldsymbol{x}), f(\boldsymbol{x}'))$

Kernel function examples



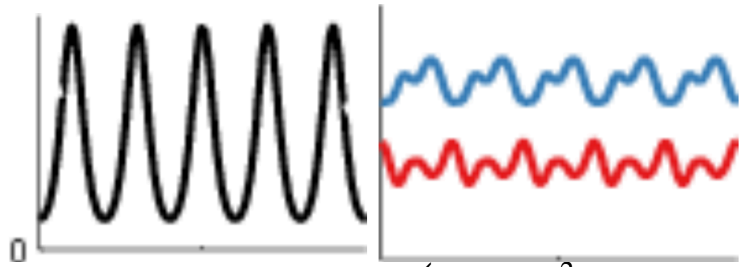
$$k_{\text{Lin}}(x, x') = \sigma_b^2 + \sigma_v^2(x - c)(x' - c)$$

Linear



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

Squared Exponential



$$k_{\text{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

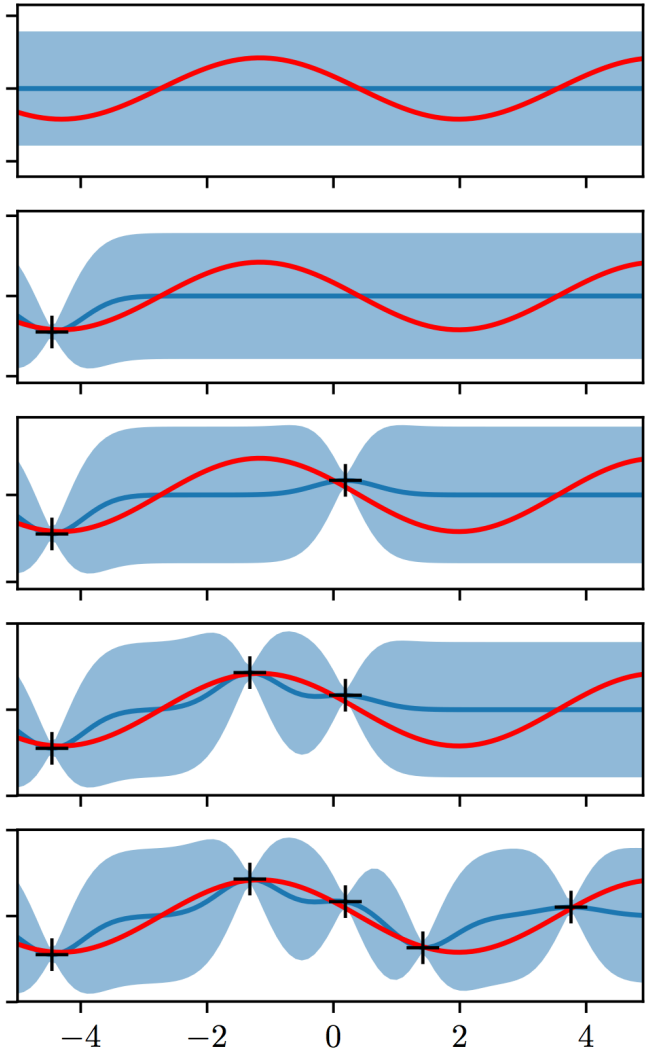
Periodic

Plus many more (e.g. polynomial, Matern, rational quadratic, etc.).

See: https://www.cs.toronto.edu/~duvenaud/cookbook/

Function space view of GPs

- A GP is a collection of random variables, any finite number of which follow a multivariate Gaussian distribution
- GPs generalize the notion of Gaussian distributions to the function space (i.e. the infinite-dimensional case); thus, they are distributions over functions (where this distribution is implied by choice of covariance function/kernel)
- Finite-dimensional distributional view: distribution of data points and test point(s) is jointly Gaussian with correlation governed by the chosen kernel



Limitations of Gaussian process regression

- The largest limitation of GP regression is computational complexity
  - Computation of posterior for a point requires computation of $(K + \sigma_\epsilon^2 I)^{-1}$, where $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - Thus, K has dimension equal to the number of data points conditioned on
  - Inversion of dense matrix has complexity $O(n^3)$
  - Thus GP regression scales extremely poorly with the amount of data
  - One approach: use small collection of "inducing points", ignore the rest of the data
- Other limitations:
  - encode prior structure directly via kernels (possibly limiting the capacity of features)
  - Relies on known noise covariance (relaxing this assumption leads to t processes)

Further reading on GP methods

- *Gaussian Processes for Machine Learning*, Rasmussen and Williams
  - Great, comprehensive coverage, free online
- https://distill.pub/2019/visual-exploration-gaussian-processes/
  - Interactive GP visualizations

Neural networks

- Linear models are restrictive; while nonlinear basis functions are more expressive, GPs are limited in terms of choice of kernels
- Neural networks directly optimize nonlinear features
- Example: 2-layer neural network

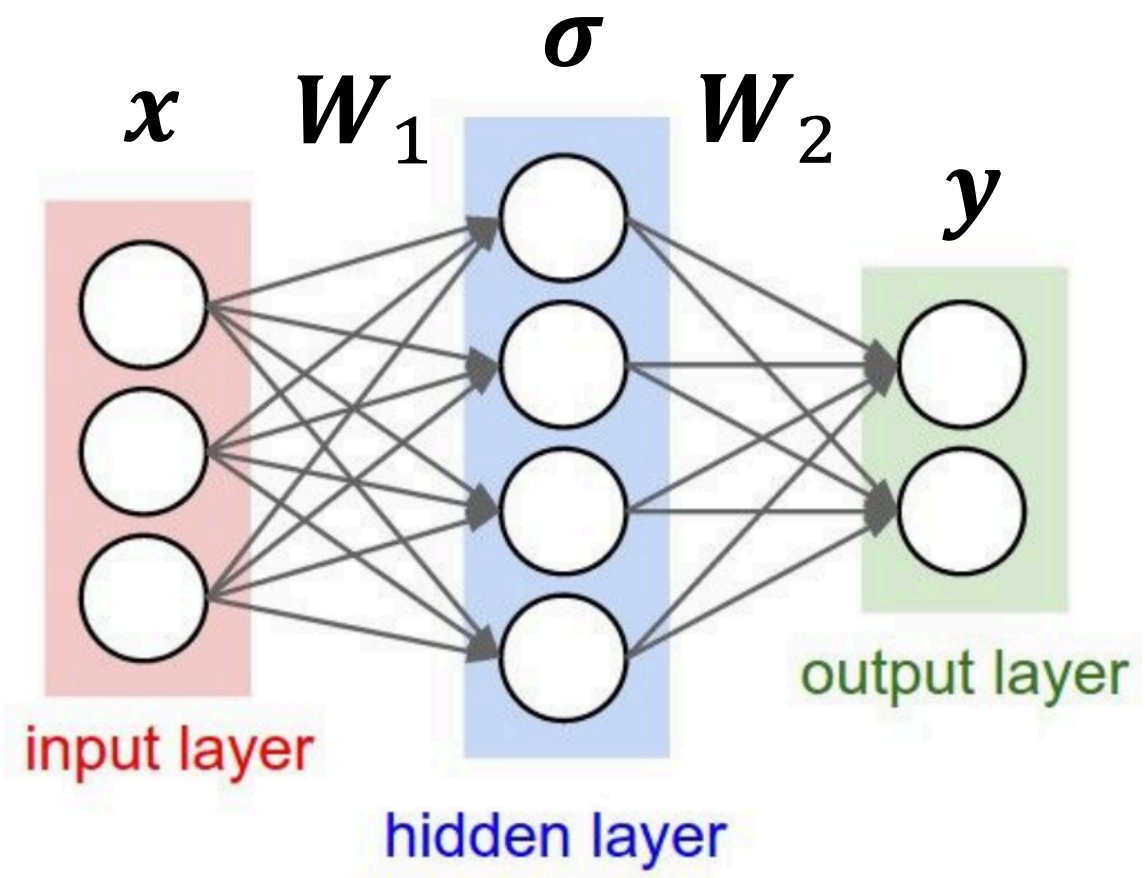
$$\boldsymbol{y} = f(\boldsymbol{x}) = W_2\boldsymbol{\sigma}(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

where $\boldsymbol{\sigma}$ is some nonlinear function (applied element-wise), typically called the **activation function**. Examples:

  - Relu: $\sigma(x_i) = \max(0, x_i)$
  - Tanh: $\sigma(x_i) = \tanh(x_i)$
  - Sigmoid/logistic: $\sigma(x_i) = \frac{1}{1+e^{-x_i}}$
  - and many many more

- Can also have nonlinearities that are not applied element-wise, but they are recently less popular

Neural networks



Training neural networks

- Given this model, we wish to solve the optimization problem

$$\min_{W_1,W_2} \sum_{i=1}^{n} L(\boldsymbol{x}_i, \boldsymbol{y}_i)$$

where L is some loss function.

- In least squares, we had an analytical solution to minimize our empirical loss
- Because of nonlinear dependence on parameters, neural network optimization problem is non-convex, and so we must turn to gradient descent
- Gradient descent can be done efficiently via **error backpropagation**: basically using chain rule backwards on the network
  - Network activation functions (and losses) are usually chosen to be cheap/easy to differentiate

Computation graphs and deep learning

- Calculating gradients for networks can be done automatically/efficiently for networks by building computation graphs and performing automatic differentiation
  - This is what deep learning frameworks such as e.g. tensorflow do
- What is deep learning? Stacking more layers/more exciting "neural network architectures"/collection of tricks to improve training
  - Examples: convolutional networks (popular in vision), recurrent networks (popular in language modelling), etc.
- These architectures have, for the most part, not been used very much in model learning for RL and control

Limitations of neural networks

- Data hungry: may require a large amount of data to train effectively
- Hard to incorporate prior knowledge into (compared to GPs), e.g. smoothness
- Possibly unstable to train; definitely time/compute intensive
- No confidence measure/expression of uncertainty
  - there are some architectures that address this but fully Bayesian treatment of neural networks are still an active research area

Further reading on neural networks

- *Deep Learning*, Goodfellow, Bengio and Courville
  - The standard on deep learning methods
- CS231n lecture notes and slides
  - Introduction to neural networks + vision models
- CS229 notes on deep learning
- http://playground.tensorflow.org

Next time

- Nonlinear model-based reinforcement learning