AA203
Optimal and Learning-based Control

Fundamentals of Regression

Stanford University

ASL
Autonomous Systems Lab

Today's lecture

- Aim
  - Review regression models that will be used throughout this class

- Outline:
  - Maximum likelihood and least squares via ML
  - Bayesian inference and Bayesian linear regression
  - Gaussian process regression
  - Intro to neural networks

Regression

- Main tool in learning dynamics models, value functions, and policies
- Aims to model a continuous output as a function of the input
- Contrast to classification setting which has a finite set of outputs

Supervised learning

- We are given training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{d}$
- Want to find some f such that $\hat{y}_i = f(\boldsymbol{x}_i)$ is close to y_i

What does it mean to be “close”?

How do we represent f ?

How do we represent uncertainty?

Maximum likelihood

- Dominant method in statistical inference due to flexibility and computational tractability
- Assume model is parameterized by $\boldsymbol{\theta}$
- Likelihood $L(\boldsymbol{\theta}) = p(\mathcal{D} \mid \boldsymbol{\theta})$
- Principle of maximum likelihood: choose $\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$
- In practice, will maximize *log likelihood*

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{d} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$$

From max likelihood to least squares

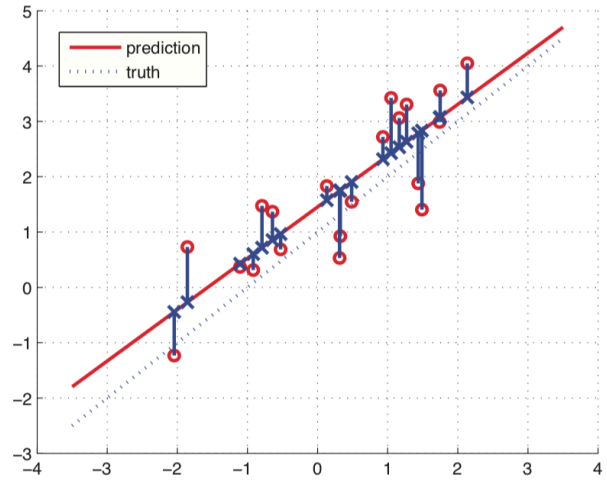
- We have not chosen a model parameterization: how to represent f?
- We will assume data is generated by $y = \boldsymbol{x}^T \boldsymbol{\theta}^* + \epsilon$
- Likelihood is

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}^T \boldsymbol{\theta}, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y - \boldsymbol{x}^T \boldsymbol{\theta})^2}{2\sigma^2}}$$

Least squares

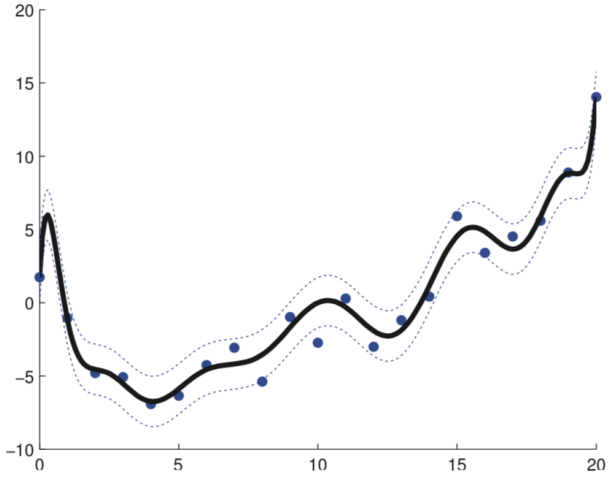
- Loss function is convex + smooth in $\boldsymbol{\theta}$, so first order necessary conditions imply global optimum

$$\nabla_{\boldsymbol{\theta}} \|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2 = 2X^T X\boldsymbol{\theta} - 2X^T\boldsymbol{y}$$



Murphy, 2012.

- Observe $\boldsymbol{x}$ without noise, so can transform nonlinearly to construct *features* $\phi(\boldsymbol{x})$



Bayesian inference

- Previously, we have computed a point estimate of model parameters, have not incorporated prior information
- Bayesian inference: specify prior belief over parameter, use Bayes' rule to compute the posterior distribution

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

- As opposed to point estimate, gives full distribution over parameter
- Allows incorporation of prior information

Bayesian inference: difficulties

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

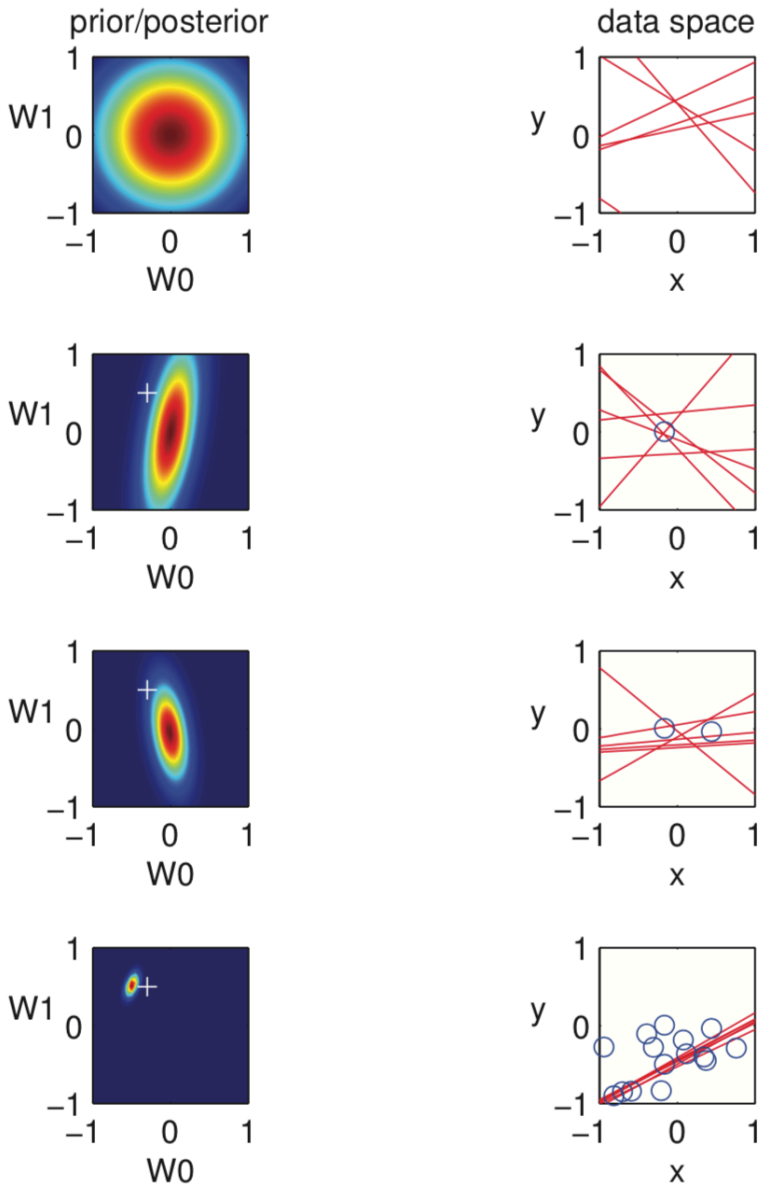
- Requires computing marginal likelihood $p(\mathcal{D})$
- Typically results in intractable integral

$$\int p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$$

- In general, we will not be able to exactly compute posterior
  - Instead, turn to sampling-based approximations
- We will examine special cases where posterior computation is exact

Bayesian least squares

- Assume linear model with iid Gaussian noise
- Choose prior $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$



Murphy, 2012.

Maximum a Posteriori Estimation

- Want to factor in prior information but computing full posterior is irrelevant/too difficult
- Maximum a Posteriori (MAP) estimator

$$\hat{\boldsymbol{\theta}}_{MAP} = \text{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \mathcal{D}) = \text{argmax}_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})$$

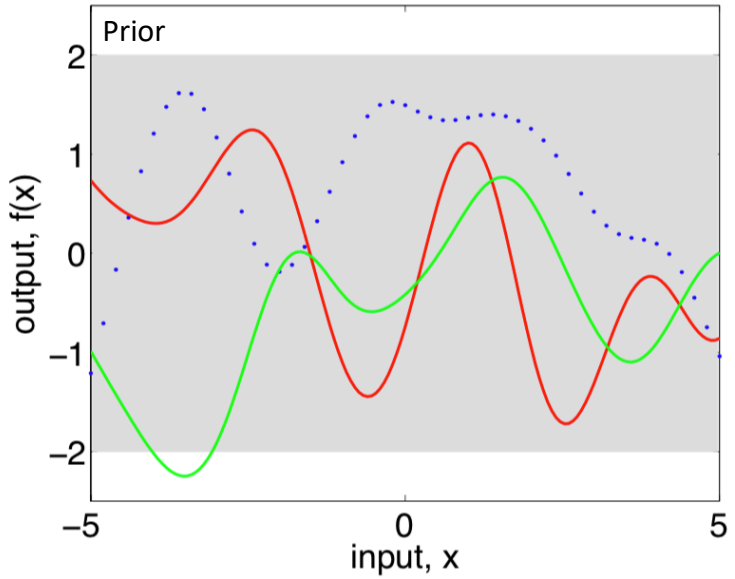
- As prior becomes flatter, approaches maximum likelihood estimator

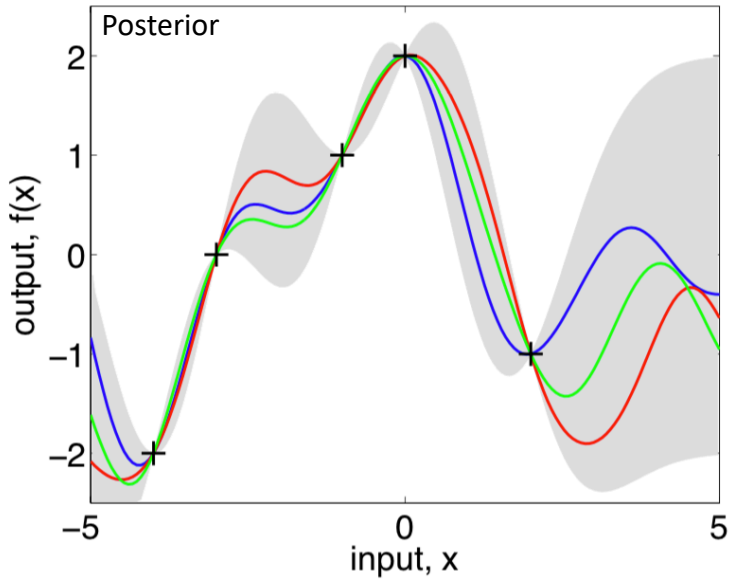
Gaussian process regression

- Gaussian process: can think of as infinite dimensional Gaussian distribution, or Gaussian distribution over functions
- Fully specified by mean and variance

$$\mu(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - \mu(\boldsymbol{x}))(f(\boldsymbol{x}') - \mu(\boldsymbol{x}'))]$$

- The function $k(\boldsymbol{x}, \boldsymbol{x}')$ is referred to as the *kernel*; must be
  - Symmetric: $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$
  - Positive definite: $\begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_k) \\ \vdots & & \vdots \\ k(\boldsymbol{x}_k, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_k, \boldsymbol{x}_k) \end{bmatrix} \succ 0$





Rasmussen and Williams, 2006.

Kernel functions

- Consider Bayesian linear regression with prior $\boldsymbol{\theta} \sim \mathcal{N}(0, \Sigma_0)$

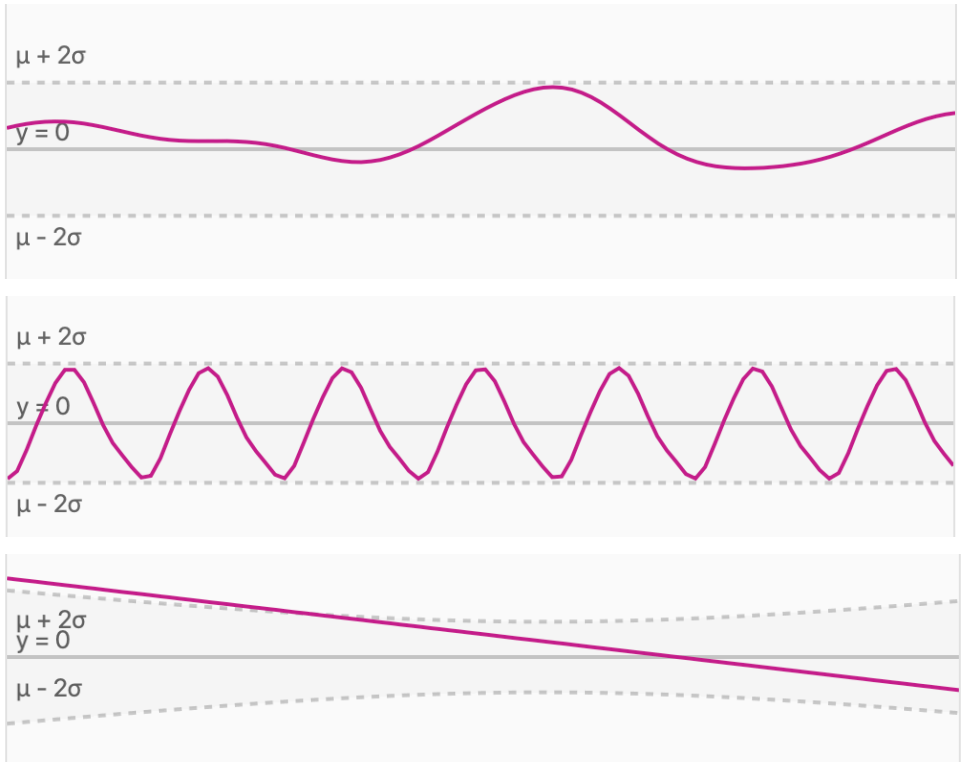
$$\mu(\boldsymbol{x}) = \mathbb{E}[\boldsymbol{\theta}^T \phi(\boldsymbol{x})] = 0$$
$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^T] \phi(\boldsymbol{x}) = \phi(\boldsymbol{x})^T \Sigma_0 \phi(\boldsymbol{x})$$

- So kernel is a simple inner product of feature vectors; can rewrite BLR in this form
- In general: every positive definite kernel corresponds to a (possible infinite dimensional) set of features; e.g

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2\ell} \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2)$$

Kernel function

- Choice of kernel determines features/shape of the posterior belief over function

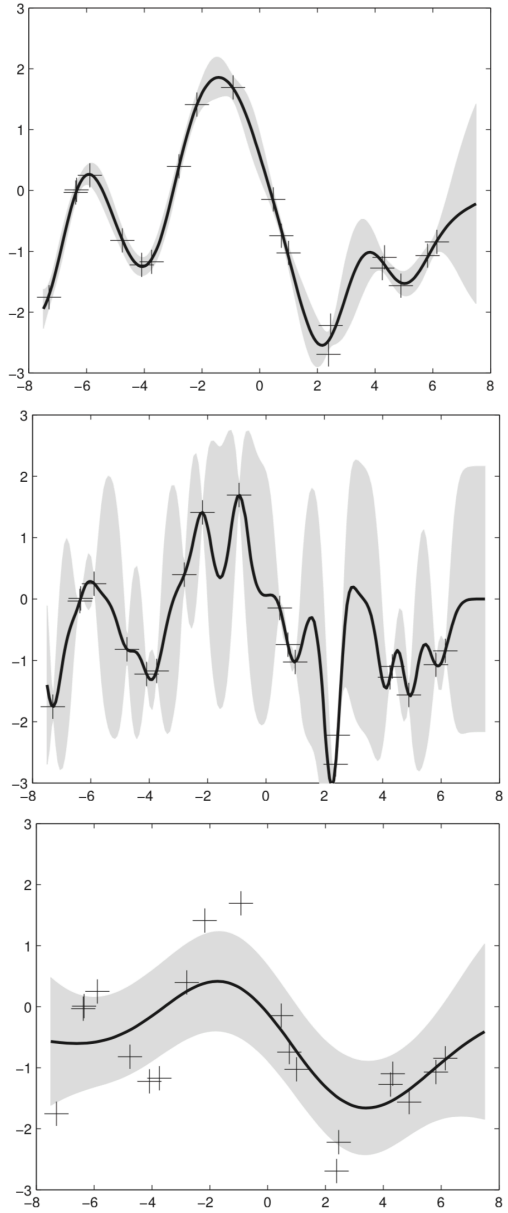


Squared exponential kernel

Periodic kernel

Linear kernel

Gortler et al., Distill, 2019



Murphy, 2012.

Gaussian process: machinery

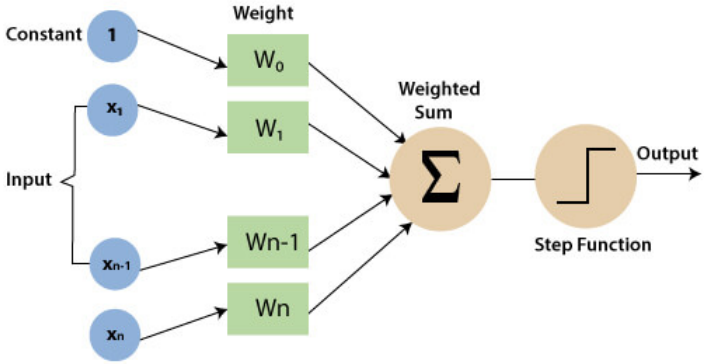
- Let $\begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix})$ then $p(\boldsymbol{y}_1 \mid \boldsymbol{y}_2) = \mathcal{N}(\boldsymbol{\mu}_1 + \Sigma_{21}\Sigma_{11}^{-1}(\boldsymbol{y}_2 - \boldsymbol{\mu}_2), \Sigma_{11} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12})$
- So have $\begin{bmatrix} f(\boldsymbol{x}^*) \\ \boldsymbol{y} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}^*, \boldsymbol{x}^*) & K(X, \boldsymbol{x}^*) \\ K(\boldsymbol{x}^*, X) & K(X, X) + \sigma^2 I \end{bmatrix})$
- With $K(\boldsymbol{x}^*, X) = \begin{bmatrix} k(\boldsymbol{x}^*, \boldsymbol{x}_1) \\ \vdots \\ k(\boldsymbol{x}^*, \boldsymbol{x}_d) \end{bmatrix} \quad K(X, X) = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_d) \\ \vdots & & \vdots \\ k(\boldsymbol{x}_d, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_d, \boldsymbol{x}_d) \end{bmatrix}$
- Gives
$$\mathbb{E}[f(\boldsymbol{x}^*)] = K(\boldsymbol{x}^*, X)(K(X, X) + \sigma^2 I)^{-1}\boldsymbol{y}$$
$$\text{var}(f(\boldsymbol{x}^*)) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - K(\boldsymbol{x}^*, X)(K(X, X) + \sigma^2 I)^{-1}K(X, \boldsymbol{x}^*)$$

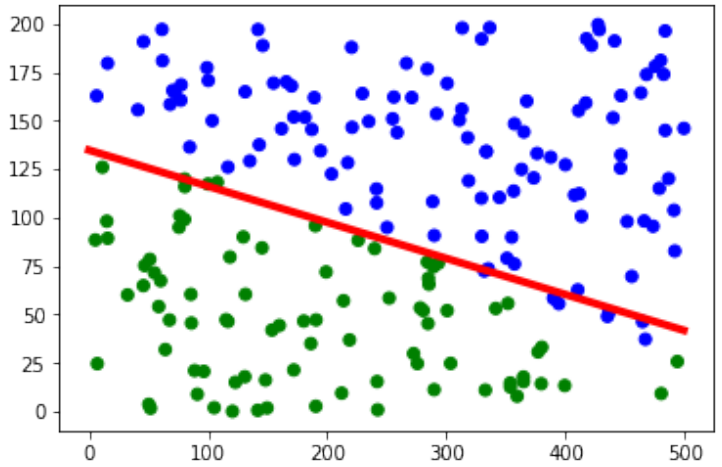
Neural networks: perceptron

- Have so far pre-specified features
- Since 2012, significant progress in learning features via neural networks
- Perceptron: $h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$
- The nonlinearity $\sigma(\cdot)$ is the *threshold* function
- Example: sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Results in linear binary classifier





Stack exchange

Neural networks

- Feed-forward neural network: stack hidden units
- Single hidden layer:

$$\boldsymbol{h} = \sigma(W_1 \boldsymbol{x})$$
$$\boldsymbol{y} = W_2 \boldsymbol{h}$$

- Can choose loss function
  - Squared error common for regression
- Problem: optimization no longer convex
  - Bottleneck until empirical performance became state of the art