MC & Optimization Lecture 16

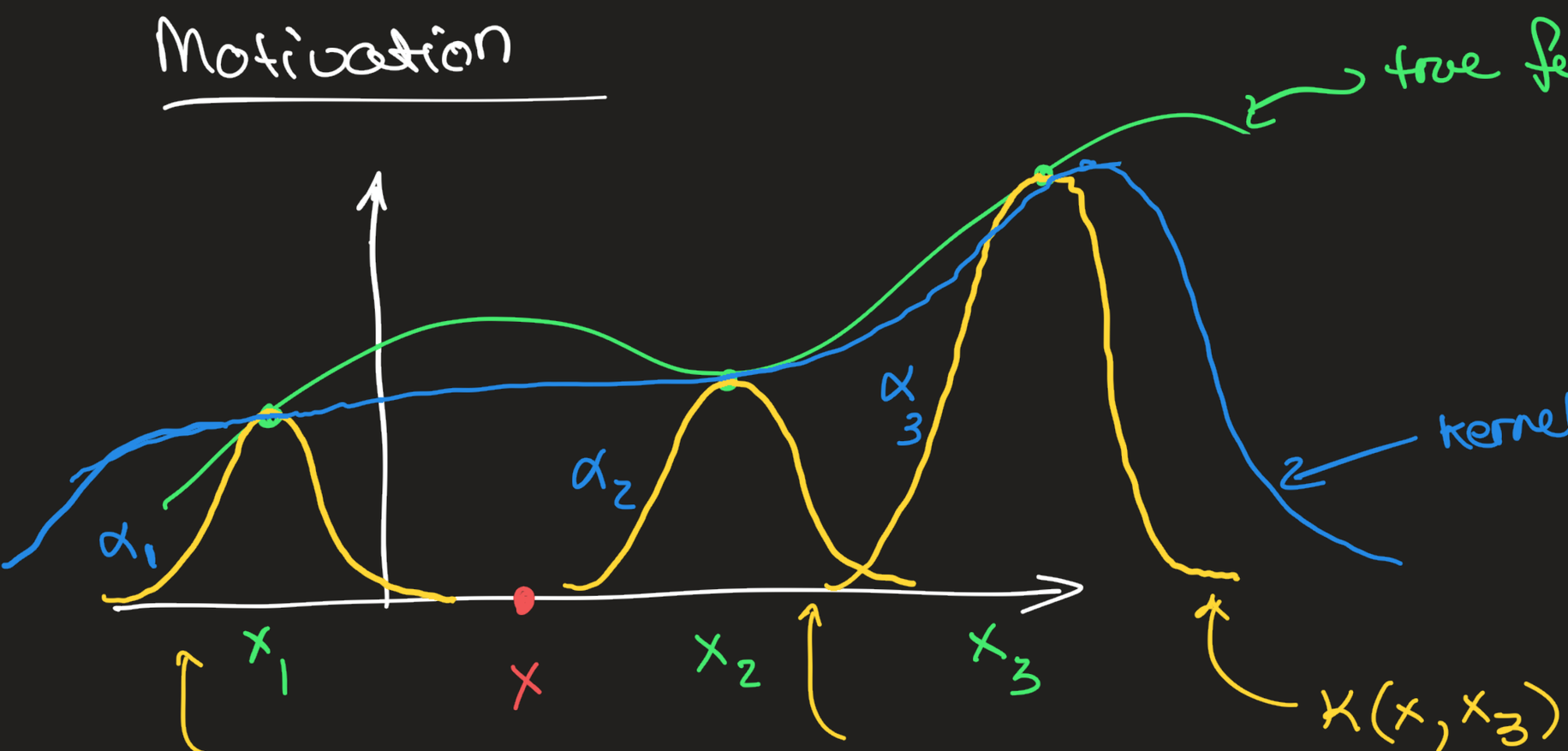
- Kernel Learning
- Approximate Kernel Learning

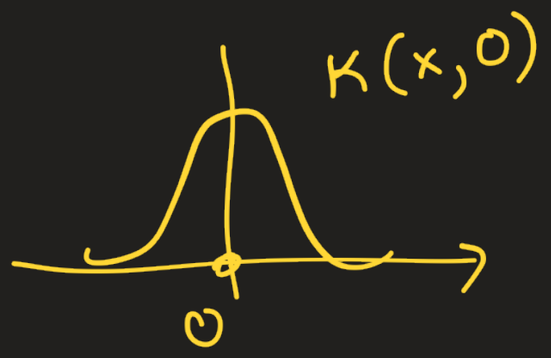
Brief History of ML

- linear models, learn w such that $w^T x$ is useful for a task:
  - SVMs: $\text{sgn}(w^T x + b)$
  - Logistic Regression: $\text{softmax}(Wx + b)$
  - linear regression: $y = Ax$
- feature-engineering: domain-dependent design $\phi : \mathbb{R}^d \to \mathbb{R}^D$ so that $w^T \phi(x)$ is useful
- neural networks: learn ϕ as a composition of multiple simpler functions, e.g. $\phi = \phi_3 \circ \phi_2 \circ \phi_1$, where $\phi_i : x \mapsto \sigma(W_i x + b_i)$, so that $w^T \phi(x)$
- <u>kernel methods</u> Fix a feature map ϕ. Learn weights w s.t. $f(x) = \sum_{i=1}^{n} w_i \langle \phi(x), \phi(x_i) \rangle$ is useful.

- Neural Networks redux (Deep Learning)
  - Neural networks again, but with:
    - more data
    - better architectures
    - more computational capabilities
    - more sophisticated optim algs.

Motivation


true function
kernel section
$\hat{f}(x) = \sum \alpha_i K(x, x_i)$
α_1
α_2
α_3
x_1
x
x_2
x_3
$K(x, x_1)$
$K(x, x_2)$
$K(x, x_3)$


$K(x, 0)$
0

In general we define a feature map $\phi: \mathbb{R}^d \to \mathbb{R}^D$ then the <u>kernel function</u>

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

Notice that K is a PSD, symmetric function. That is, for any $\{x_1, \dots, x_n\}$, then the corresponding <u>kernel matrix</u>

$$K = [K(x_i, x_j)]_{i,j=1}^{n} = \begin{bmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \cdots & \langle \phi(x_1), \phi(x_n) \rangle \\ \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \cdots & \langle \phi(x_n), \phi(x_n) \rangle \end{bmatrix}$$

$$= \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \end{bmatrix} \begin{bmatrix} \phi(x_1) & \cdots & \phi(x_n) \end{bmatrix} = \underline{\Phi}\,\underline{\Phi}^T$$

where $\underline{\Phi} \in \mathbb{R}^{n \times D}$

More generally (because we don't want to design ϕ) we pick a kernel function directly. This corresponds to some choice of ϕ.

Any function $K(x,y)$ that is PSD is a valid kernel.

Common choices:

- polynomial kernel of order r

$$K_{poly}(x,y) = (c\langle x,y\rangle + d)^r$$

corresponds to $\phi(x) =$ [all the monomials in the d variables $x_1, \dots, x_d$ and a constant term of order up to and including r]

$$D = \mathbb{R}(d^r)$$

- Gaussian kernel (RBF kernel, squared-exponential kernel)

$$K_{Gauss}(x, y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$$

$$\phi(x) = \left[\underline{\text{all}} \text{ monomials in } x, \text{ appropriately scaled}\right]$$

Most used in ML because of its universality

- Laplacian kernel

$$K_{Laplacian}(x, y) = \exp\left(-\frac{\|x-y\|_1}{\sigma}\right)$$

- many other standard kernels for dealing w/ data in non-Euclidean domains: graphs, strings, etc.

Motivation for working directly w/ kernels instead of ϕ

Consider an NLP problem where we use BOW encodings of input text spans.

"the dog ate my homework" $\longmapsto$ [0 1 0 ... 0 1 0 1 0 0 0 0 1 0 ...]

dog, cat, ate, homework, my

$\in \mathbb{R}^D$ where D is size of vocab

We want to consider interactions of pairs of words. Using ERM we would fit this model

$$v_* = \operatorname*{argmin}_v \frac{1}{n} \sum_{i=1}^{n} (y_i - v^T \phi(x_i))^2 + \lambda \|v\|_2^2$$

where

$$\phi\left(\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}\right) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1^2 \\ x_1 x_2 \\ \vdots \\ x_d^2 \end{bmatrix} \in \mathbb{R}^{1 + 2d + \binom{d}{2}}$$

In practice $d \approx 70K$ (say on Wikipedia) and that means $D = \mathcal{O}(100M)$. Let's see what happens if we use the standard RR formulation

$$v_* = \underset{v}{\operatorname{argmin}} \frac{1}{n} \| y - \bar{\Phi} v \|_2^2 + \lambda \| v \|_2^2$$

$$= \underbrace{\left(\bar{\Phi}^T \bar{\Phi} + n\lambda I \right)^{-1}}_{D \times D \text{ cost } \mathcal{O}(D^3) \text{ to invert}} \bar{\Phi}^T y$$

so we cannot find v_*.

We <u>only</u> care about predicting. So we will use the <u>kernel trick</u> : we only need to compute

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

Notice

$$f(x_{new}) = v_*^T \phi(x_{new}) = \phi(x_{new})^T v_*$$

$$= \phi(x_{new})^T \left(\Phi^T \Phi + n\lambda I \right)^{-1} \Phi^T y$$

observe (from the SVD) that for any matrix M and scalar $c > 0$

$$(M^T M + cI)^{-1} M^T = M^T (MM^T + cI)^{-1}$$

Take $M = \Phi$ and $c = n\lambda$, then

$$f(x_{new}) = \phi(x_{new})^T \left(M^T M + cI \right)^{-1} M^T y$$

$$= \phi(x_{new})^T M^T (MM^T + cI)^{-1} y$$

$$= \phi(x_{new})^T \Phi^T (\Phi \Phi^T + cI)^{-1} y$$

Notice that $K = \underline{\Phi}\underline{\Phi}^T = \left[K_{poly}(x_i, x_j)\right]_{ij=1}^{n}$

and

$$\phi(x_{new})^T \underline{\Phi}^T = \phi(x_{new})^T \left[\phi(x_1) \,\vdots\, \cdots \,\vdots\, \phi(x_n)\right]$$

$$= \left[K(x_{new}, x_1), \cdots, K(x_{new}, x_n)\right]$$

so our prediction is

$$f(x_{new}) = \left[K(x_{new}, x_1), \cdots, K(x_{new}, x_n)\right] \underbrace{\left(K + n\lambda I\right)^{-1} y}_{:= \alpha}$$

$$= \sum_{i=1}^{n} \alpha_i \, K(x_{new}, x_i)$$

Kernel Learning Paradigm

- Choose our kernel K
- This corresponds to a function space (RKHS)

$$\mathcal{H} = \overline{\text{span}\{ K(\cdot, x) : x \in \mathbb{R}^d \}}$$

$$= \overline{\left\{ \sum_{i=1}^{n} \alpha_i K(\cdot, x_i) : x_i \in \mathbb{R}^d, \alpha_i \in \mathbb{R} \right\}}$$

This space can be very large.

- Do ERM on this function space

$$f^* = \underset{f \in \mathcal{H}}{\text{argmin}} \ \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \lambda \|f\|_{\mathcal{H}}^2$$

- this is cvx.

- practically we have a <u>Representer Theorem</u>

The ERM

$$f^* = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2$$

has a solution of the form

$$f^*(x) = \sum_{i=1}^{n} K(x, x_i) \alpha_i$$

where K is the kernel associated with $\mathcal{H}$.

Drawbacks:

- dealing w/ $K \in \mathbb{R}^{n \times n}$ can be expensive when $n \gg 1$
- applying f requires you to store the training data set.

Mitigating approaches:

- Nyström approximation
- Random feature maps (Recht & Rahimi)

- because of the representer theorem we know in particular

$$f^*(x_i) = \sum_{j=1}^{n} K(x_i, x_j)\alpha_j = [K\alpha^*]_i$$

so we can fit f^* by finding α^*

$$\alpha^* = \underset{\alpha \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \ell([K\alpha]_i, y_i) + \lambda \alpha^T K \alpha$$