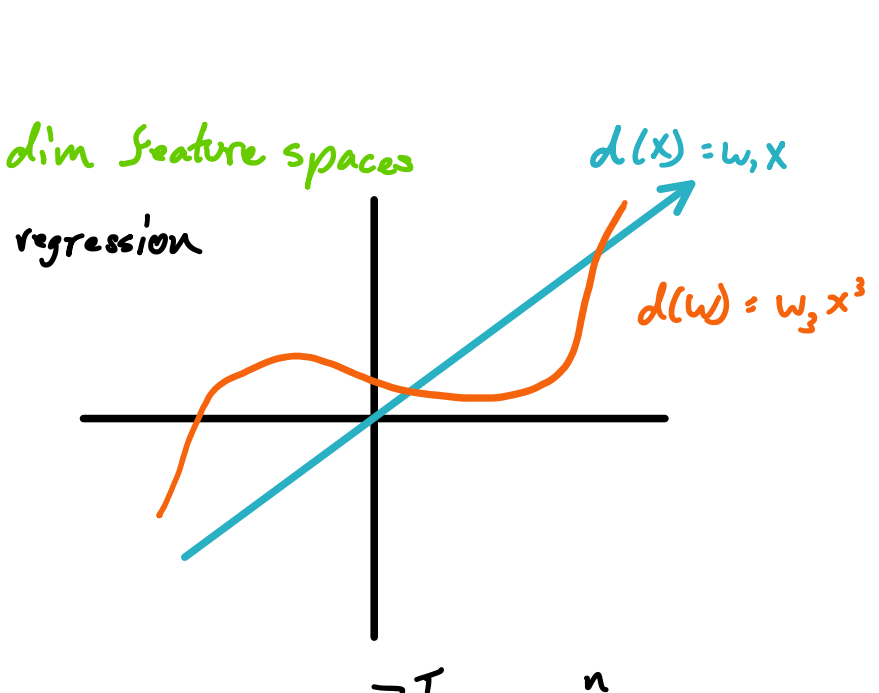
Unit 6 Kernel Regression and Neural Networks

Thursday, June 1, 2023 5:07 PM

6.1 Kernel regression

goals
- why use higher dim feature spaces
- reformulate regression in terms of Kernels
- look at some popular Kernels
- cautions and considerations

- why use higher dim feature spaces
  - it can extend regression



We let $\underline{x} = [x_1, x_2, x_3 \ldots x_n]^T \in R^n$

The consider $d(\underline{x}) = \phi^T(\underline{x})\underline{w}$, where $\phi(\underline{x}) \in R^P$, $P > n$

↑ regression or model

example) $\underline{x} = [x_1, x_2]^T$, $\phi(\underline{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, x_1, x_2, 1]$

we find w using training data through the process below

- $\underline{x}_i, d_i$ $i = 1, 2 \ldots N$
- Ridge regression: $\min_{\underline{w}} \sum_{i=1}^{N} (d_i - \phi^T(\underline{x}^i)\underline{w})^2 + \lambda\|\underline{w}\|_2^2$
- where $\underline{d} = [d^1, d^2 \ldots d^N]^T$
  $\Phi = [\phi(x^1)\ \phi(x^2) \ldots \phi(x^N)]^T$

so our sol is $\underline{w} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T \underline{d}$

now, $d(\underline{x}) = \phi^T(\underline{x})\underline{w} = \phi^T(\underline{x})(\Phi^T\Phi + \lambda I)^{-1}\Phi^T\underline{d}$ — $P \times P$

↳ which we say can be written as a weighted sum of "Kernels"

Regression

? $A = B$

$CA = BC^T$

Matrix Identity: $(\Phi^T\Phi + \lambda I)^{-1}\Phi^T = \Phi^T(\Phi\Phi^T + \lambda I)^{-1}$

$(\Phi\Phi + \lambda I)(\Phi^T\Phi + \lambda I)^{-1}\Phi^T = \Phi^T(\Phi\Phi^T + \lambda I)^{-1}(\Phi\Phi^T + \lambda I)$

Thus: $d(\underline{x}) = \phi^T(\underline{x})\Phi^T(\Phi\Phi^T + \lambda I)^{-1}\underline{d}$ — $N \times N$

Question: Slide 3 of 6.1 The "Note" Part

where did this come from

$[\Phi\Phi^T]_{i,j} = \phi^T(\underline{x}^i)\phi(\underline{x}^j)$

$[\phi^T(\underline{x})\Phi^T]_j = \phi^T(\underline{x})\phi^T(x^j)$

Define "Kernel" as $K(\underline{u}, \underline{v}) = \phi^T(\underline{u})\phi(\underline{v})$

Let $\underline{\alpha} = [\alpha_1 \ldots \alpha_N]^T$
$= (\Phi\Phi^T + \lambda I)^{-1}\underline{d}$

Then $d(\underline{x}) = \sum_{i=1}^{N} \alpha_i \phi^T(\underline{x})\phi(x^i) = \sum_{i=1}^{N} \alpha_i K(\underline{x}, x^i)$

- Kernel methods find $d(\underline{x})$ without computing $\phi(\underline{x})$

$\alpha = (\Phi\Phi^T + \lambda I)^{-1}\underline{d}$ $K = \Phi\Phi^T$

$(K + \lambda I)^{-1}\underline{d}$

$[K]_{ij} = \phi^T(\underline{x}^i)\phi(\underline{x}^j) = K(\underline{x}^i, \underline{x}^j)$

not really sure how but, K can be computed efficiently!

ex) Monomials of degree q — Powers add to q

↳ $\phi(\underline{x}) \to x_1^q, x_1^{q-1}x_2, \ldots x_3^{q-5}x_6^2x_8$

↳ $K(\underline{u}, \underline{v}) = \phi^T(\underline{u})\phi(\underline{v}) = (u^Tv)^q$

$P = \frac{(q+m-1)!}{q!(m-1)!}$ Terms

Suppose — dim of ϕ

$m = 10, q = 5 \to P \sim 2000$
$m = 100, q = 5 \to P \sim 10^8$

- computing $O(P)$ vs $O(N)$
- memory $O(NP)$ vs $O(N^2)$

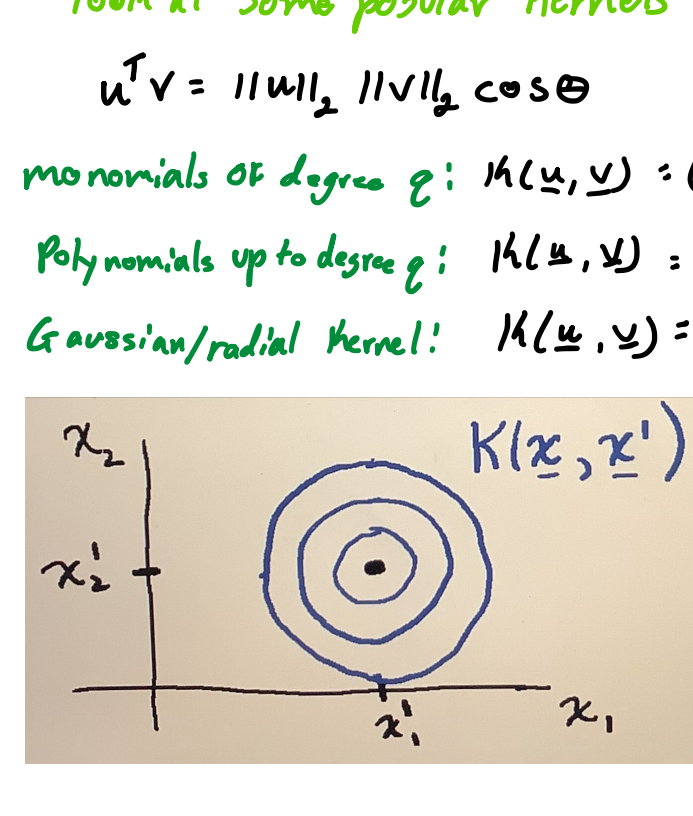
- look at some popular Kernels which one we use depends on $\underline{u}, \underline{v}$

$u^Tv = \|u\|_2 \|v\|_2 \cos\theta$

monomials of degree q: $K(\underline{u}, \underline{v}) = (u^Tv)^q$

Polynomials up to degree q: $K(\underline{u}, \underline{v}) = (u^Tv + 1)^q$

Gaussian/radial Kernel: $K(\underline{u}, \underline{v}) = \exp\{-\frac{\|\underline{u} - \underline{v}\|_2^2}{2\sigma^2}\}$



- no explicit $\phi(\underline{x})$
- All Polynomial orders
- smoothness controlled by σ

- cautions and considerations

$d(\underline{x}) = \phi^T(\underline{x})\underline{w}$ vs $d(\underline{x}) = \sum_{i=1}^{N} \alpha_i K(x, x^i)$

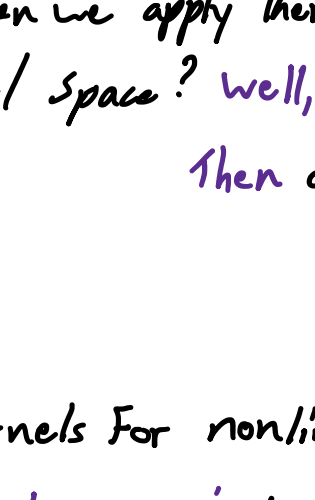
- store and compute α $(N \times 1)$ vs $\underline{w}$ $(P \times 1)$
- binary class $\text{sign}\{d(\underline{x})\}$ better boundaries
- avoid over fitting with cross validation CV

6.2 Kernel based SVM

Goals:
- reformulate linear max margin classifier in Terms of support vectors
- derive Kernel version of hinge loss with ridge regression
- Summarize features of support vector machines

- linear max margin classifier

if we have data points that are linearly seperable we can define a classifier that splits the data based on the max difference between data points. The points that are used to find max differences are Support vectors



$w = [\bar{w}^T\ b]^T$ depends only on the support vectors

high dim space derived from data

Recall Kernel regression: $d(\underline{x}) = \phi^T(\underline{x})\underline{w} = \sum_{j=1}^{N} \alpha_j K(x, x^j)$

Oh, what happens when we apply Kernel regression when we don't have a higher dimensional space? Well, Let $\phi(\underline{x}) = \underline{x} \Rightarrow K(x, x^j) = x^Tx^j$

Then $d(\underline{x}) = \sum_{j=1}^{N} \alpha_j x^Tx^j = \underline{x}^T \sum_{j=1}^{N} \alpha_j x^j$ — $\underline{w}$

we can see the explicit dependence of w on the training data. Also note that All $\alpha_j = 0$ except for support vectors

We can use Kernels for nonlinear decision boundaries

high-dim feature space: $\underline{x} \to \phi(\underline{x})$

ex) $\phi(\underline{x}) = [x_1^2, x_2^2, \ldots x_2x_4 \ldots x_{m-1}, x_M]$

$\hat{d}(\underline{x}) = \text{sign}(\phi^T(\underline{x})\underline{w})$

Hinge loss with ridge regression

$\min_{\underline{w}} \left(\sum_{i=1}^{N} (1 - d^i\phi^T(x^i)\underline{w})_+\right) + \lambda\|w\|_2^2$

Claim: $w = \sum_{j=1}^{N} \phi(x)^j \alpha_j$

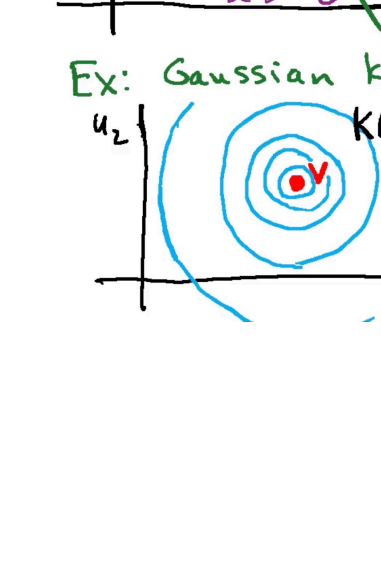
now we rewrite the hinge loss with ridge regression problem by replacing w and replacing $\phi^T(x^i)\phi(x^j)$ with $K(x^i, x^j)$

$\min_{\alpha} \sum_{i=1}^{N} (1 - d^i\phi^T(x^i)\sum_{j=1}^{N}\alpha_j\phi(x^j))_+ + \lambda\sum_{i=1}^{N}\alpha_i\phi^T(x^i)\sum_{j=1}^{N}\alpha_j\phi(x^j)$

$\min_{\alpha} \sum_{i=1}^{N} (1 - d^i\sum_{j=1}^{N}\alpha_j\phi^T(x^i)\phi(x^j))_+ + \lambda\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\phi^T(x^i)\phi(x^j)$ — $K(x^i, x^j)$

SVM $\min_{\alpha} \sum_{i=1}^{N} (1 - d^i\sum_{j=1}^{N}\alpha_j K(x^i, x^j))_+ + \lambda\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(x^i, x^j)$

Support vector machines have sparse $\underline{\alpha}$'s



decision boundary $d(\underline{x}) = 0 = \phi^T(\underline{x})\underline{w} = \sum_{j=1}^{N} \alpha_j K(\underline{x}, \underline{x}^j)$

Boundary (hinge loss) depends only on the support vectors

$K(\underline{u}, \underline{v})$ measures similarity/alignment of $\underline{u}, \underline{v}$

$K(\underline{u}, \underline{v}) = \exp\{-\frac{\|u - v\|_2^2}{2\sigma^2}\}$

Solve for α using gradient descent

by definition, the

Kernel $K(x, y) = \phi^T(x)\phi(y)$

measures the similarity between 2 points

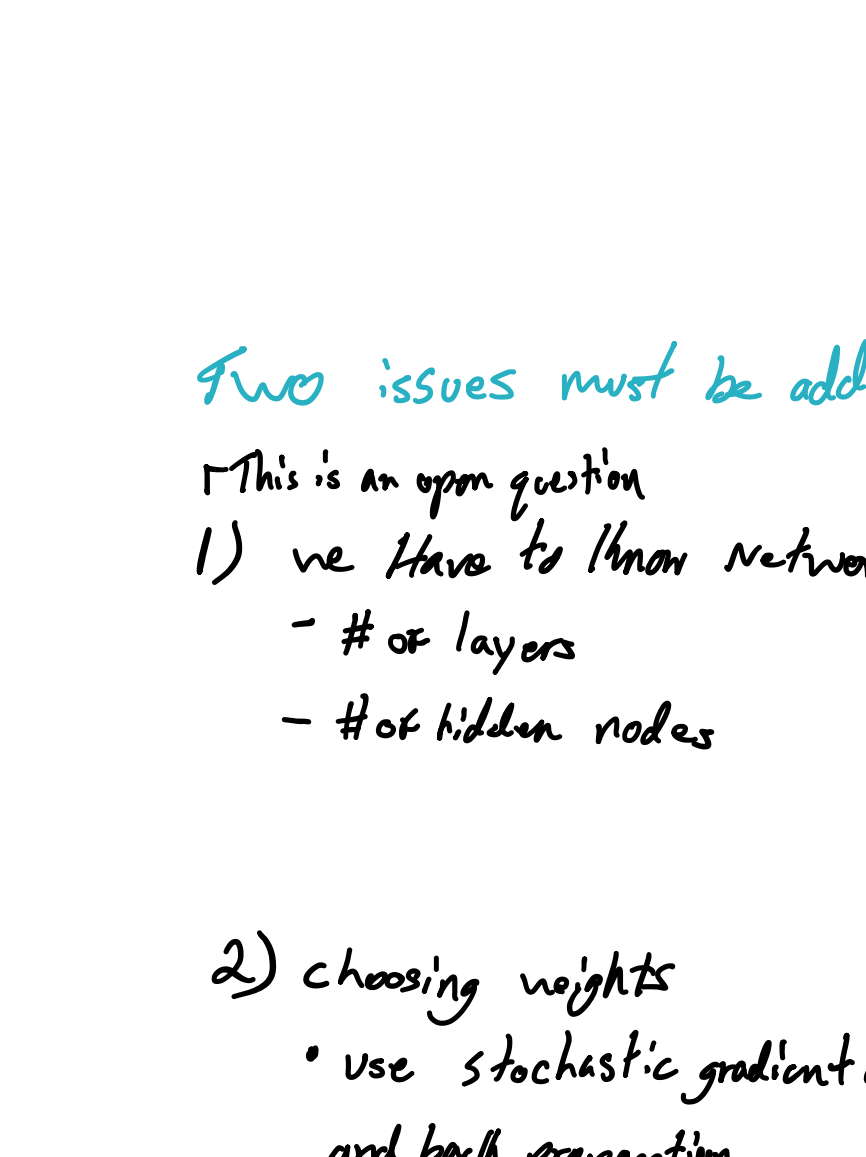
2 training data points

6.3 Intro to neural networks

- relate neural networks to linear classifiers
- define structure of multilayer neural network
- overview of procedure for training neural networks

The "neuron" generalizes a linear classifier

Feature: $x^T = [x_1, x_2, x_3 \ldots]$
weights: $w^T = [w_1, w_2 \ldots w_N]$
linear classifier: $\hat{d} = \text{sign}(x^Tw)$
Neuron: $\hat{d} = \sigma(x^Tw)$



common $\sigma(z)$

A neural network is a network of neurons [3]

multiple outputs → solve multiple problems

$h_j = \sigma\left(\sum_{i=1}^{m} w_{ji}x_i\right)$

$\hat{d}_m = \sigma\left(\sum_{n=1}^{p} v_{mn}h_n\right) = \sigma\left(\sum_{n=1}^{p} v_{m,n}\sigma\left(\sum_{j=1}^{m} w_{nj}x_j\right)\right)$

Two issues must be addressed to use NN's

This is an open question

1) we have to know Network structure
   - # of layers
   - # of hidden nodes

2) choosing weights
   - use stochastic gradient descent and back propagation
   ↳ nonconvex: local minima

Back propagation updates each layer in sequence
- then we work back from deep (output) to shallow (input)

$\min_{w_{ij}, v_{ij} \ldots} \sum_{i=1}^{N}\sum_{q=1}^{Q}\frac{1}{2}(\hat{d}_{i,q} - d_{i,q})^2$ N Training samples, Q outputs

Process:
1) Initial guess on $w_{ij}, v_{ij} \ldots$
2) randomly choose training sample i
3) calculate $h_{i,p}, \hat{d}_{i,q}$
4) Gradient descent update v_{ij}, then w_{ij} (deep to shallow)

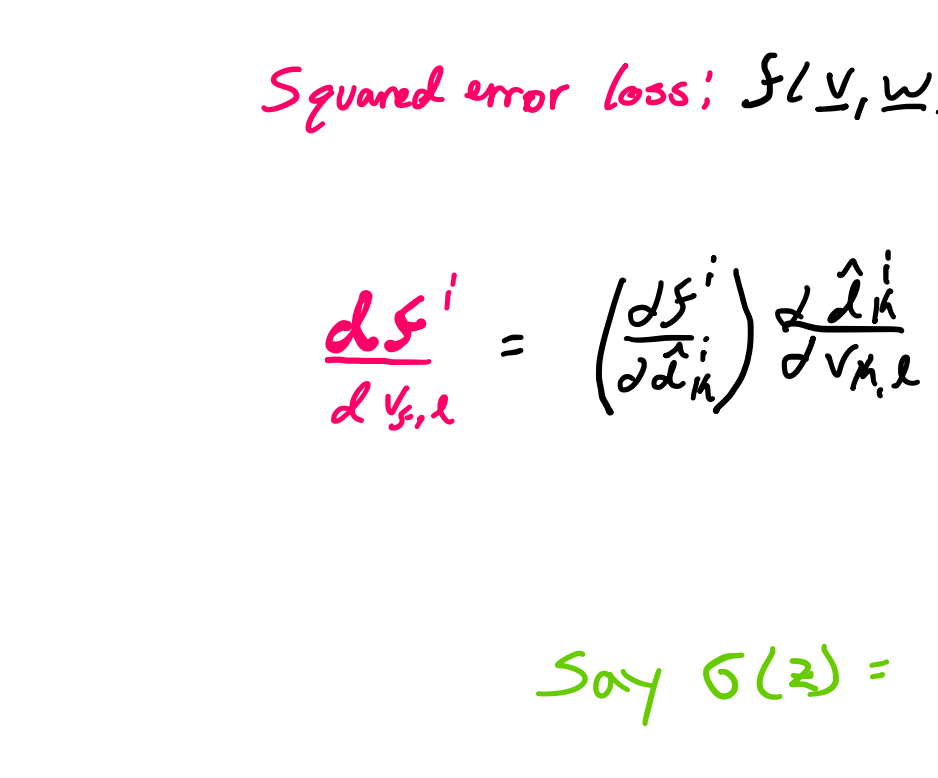
repeat until convergence

chain rule is key

6.4 back propagation of neural networks

- method for learning weights in a neural network (i.e. stochastic Grad. Descent)
- Apply stochastic Grad. Descent SGD
- Find grads of squared error loss with respect to weights

Backpropagation uses SGD to train the weights [2]



$\hat{d}_q = \sigma\left(\sum_{n=1}^{p} v_{q,n}h_n\right)$, $h_n = \sigma\left(\sum_{k=1}^{m} w_{n,k}x_k\right)$ ($= \frac{1}{1+e^{-z}}$ here)

1) initialize $w_{n,l}, v_{q,n}$
2) For t in $1, 2, 3 \ldots$
   - Pick $i_t \in \{1, 2 \ldots N\}$ randomly
   - compute $h_j^{i_t}, \hat{d}_j^{i_t}$
   - $v_{k,l}^{t+1} = v_{k,l}^t - \alpha_t \frac{d}{dv_{k,l}}(f^{i_t})$
   - $w_{m,j}^{t+1} = w_{m,j}^t - \alpha_t \frac{d}{dw_{m,j}}(f^{i_t})$

↳ now we just need to compute derivatives

Gradients are obtained using the chain rule

Squared error loss: $f(\underline{v}, \underline{w}) = \sum_{i=1}^{N}\frac{1}{2}\sum_{q=1}^{Q}(\hat{d}_q^i - d_q^i)^2$

$\frac{df^i}{dv_{k,l}} = \left(\frac{df^i}{d\hat{d}_k^i}\right)\frac{d\hat{d}_k^i}{dv_{k,l}} = (\hat{d}_k^i - d_k^i)\cdot\sigma'\left(\sum_{n=1}^{p} h_n^i v_{k,n}\right)\cdot\frac{d}{dv_{k,l}}\left(\sum_{n=1}^{p} h_n^i v_{k,n}\right)$

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

$\sigma'(z) = \frac{1}{(1+e^{-z})^2}e^{-z} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = \sigma(z)(1 - \sigma(z))$

and, $\sigma\left(\sum_{n=1}^{p} h_n^i v_{k,n}\right)\left(1 - \sigma\left(\sum_{n=1}^{p} h_n^i v_{k,n}\right)\right) = \hat{d}_k^i(1 - \hat{d}_k^i)$

$\frac{df^i}{dv_{k,l}} = (\hat{d}_k^i - d_k^i)\hat{d}_k^i(1 - \hat{d}_k^i)h_l^i$ (predicted value $\hat{d}_k^i$, actual value d_k^i) — δ_k

$= \delta_k h_l^i$ very simple gradient form

so now, $v_{k,l}^{t+1} = v_{k,l}^t - \alpha_t \delta_k h_l^i$ ("error" at output, layer input)

we can use a similar approach to find the gradient of a loss

$\frac{df^i}{dw_{m,j}} = \sum_{q=1}^{Q}\frac{df^i}{d\hat{d}_q^i}\cdot\frac{d\hat{d}_q^i}{dh_m^i}\cdot\frac{dh_m^i}{dw_{m,j}}$ (loss/output, output/hidden, hidden/Previous layer weights)

25.2 ↑ For A