CPSC 340/540 Tutorial 9

Winter 2024 Term 1

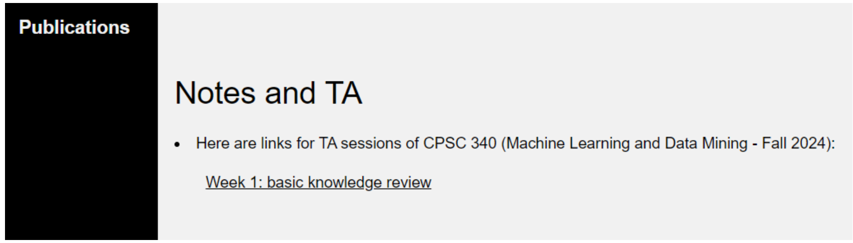
T1A: Tuesday 16:00-17:00;
T1C: Thursday 10:00-11:00;
Office Hour: Wednesday 15:00-16:00

Slides can be found at Piazza and my personal page after T1C.

Yi (Joshua) Ren

https://joshua-ren.github.io/
renyi.joshua@gmail.com
PhD with Danica

Machine Learning:
Learning dynamics, LLM, Compositional Generalization



Slides Credit: To various pervious TA's of this course

More helpful on theory

Less helpful on coding

- Kernel Regression
- MLE & MAP
- SGD

Kernel Trick: trade k by n

- Recall Linear Regression with poly-features
  - Recall the L2-regularized least squares objective with basis 'Z':

$$f(v) = \frac{1}{2}\|Zv - y\|^2 + \frac{\lambda}{2}\|v\|^2$$

  - We showed that the minimum is given by

$$v = (\underbrace{Z^T Z + \lambda I}_{k \times k})^{-1} Z^T y$$

(in practice you still solve the linear system, since inverse is less numerically unstable – see CPSC 302)

  - With some work (bonus slide), this can equivalently be written as:

$$v = Z^T(\underbrace{ZZ^T + \lambda I}_{n \times n})^{-1} y$$

  - This is faster if n << k:
    - After forming 'Z', cost is $O(n^2k + n^3)$ instead of $O(nk^2 + k^3)$.
    - But for the polynomial basis, this is still too slow since $k = O(d^p)$.

Kernel Trick: trade $\boldsymbol{k}$ by $\boldsymbol{n}$

- With the "other" normal equations we have $v = Z^T(ZZ^T + \lambda I)^{-1} y$
- Given test data $\tilde{X}$, predict $\hat{y}$ by forming $\tilde{Z}$ and then using:

$$\hat{y} = \tilde{Z} v$$

$$= \underbrace{\tilde{Z} Z^T}_{\tilde{K}} (\underbrace{ZZ^T}_{K} + \lambda I)^{-1} y$$

$$\underset{t \times 1}{} = \underbrace{\tilde{K}}_{t \times n} \underbrace{(K + \lambda I)^{-1}}_{n \times n} \underbrace{y}_{n \times 1} = \tilde{K} \underbrace{u}_{}$$

$n \times 1$ vector of "kernel weights" that we learn

- Notice that if you can from K and $\tilde{K}$ then you do not need Z and $\tilde{Z}$.

Kernel Trick: trade $\boldsymbol{k}$ by $\boldsymbol{n}$

- The matrix K = ZZ^T is called the Gram matrix K.

$$K = ZZ^T = \underbrace{\begin{bmatrix} \text{---} & z_1^T & \text{---} \\ \text{---} & z_2^T & \text{---} \\ & \vdots & \\ \text{---} & z_n^T & \text{---} \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} | & | & & | \\ z_1 & z_2 & \cdots & z_n \\ | & | & & | \end{bmatrix}}_{Z^T}$$

$$= \left.\underbrace{\begin{bmatrix} z_1^T z_1 & z_1^T z_2 & \cdots & z_1^T z_n \\ z_2^T z_1 & z_2^T z_2 & \cdots & z_2^T z_n \\ \vdots & & \ddots & \vdots \\ z_n^T z_1 & z_n^T z_2 & \cdots & z_n^T z_n \end{bmatrix}}_{n}\right\} n$$

- K contains the dot products between all training examples.
  - Similar to ‘Z’ in RBFs, but using dot product as “similarity” instead of distance.

$$\tilde{K} = \tilde{Z} Z^T = \underbrace{\begin{bmatrix} \text{---} & \tilde{z}_1^T & \text{---} \\ \text{---} & \tilde{z}_2^T & \text{---} \\ & \vdots & \\ \text{---} & \tilde{z}_t^T & \text{---} \end{bmatrix}}_{\tilde{Z}} \underbrace{\begin{bmatrix} | & | & & | \\ z_1 & z_2 & \cdots & z_n \\ | & | & & | \end{bmatrix}}_{Z^T}$$

Kernel Trick: trade $\boldsymbol{k}$ by $\boldsymbol{n}$

Linear Regression

Training

1. Form basis Z from X.
2. Compute $\underbrace{v}_{k\times 1} = (Z^TZ + \lambda I)^{-1}(Z^Ty)$

$$v = Z^T\underbrace{(ZZ^T + \lambda I)^{-1}}y$$

Testing

1. Form basis $\tilde{Z}$ from $\tilde{X}$
2. Compute $\hat{y} = \underbrace{\tilde{Z}}_{t\times k}\underbrace{v}_{k\times 1}$

Both methods make the same predictions.

Kernel Regression

Training:

1. Form inner products K from X.
2. Compute $\underbrace{u}_{n\times 1} = (K + \lambda I)^{-1} y$

Non-parametric

Testing:

1. Form inner products $\tilde{K}$ from X and $\tilde{X}$
2. Compute $\hat{y} = \underbrace{\tilde{K}}_{t\times n}\underbrace{u}_{n\times 1}$

If you want explicit feature weights 'v' from kernel regression, you can use $v = Z^Tu$

Kernel Trick: other examples to understand kernel trick

$$x^9 + 9x^8 + 36x^7 + 84x^6 + 126x^5 + 126x^4 + 84x^3 + 36x^2 + 9x + 1 \quad \text{or} \quad (x+1)^9$$

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} \ldots \quad \text{or} \quad e^x$$

Kernel Trick: Gaussian Kernel and RBF

question

stop following 83 views

Why RBF-kernel not the same as RBF-basis?

I do not quite understand the two statements in red box? I think with k as defined that way, it is just the $g(||x_i - x_j||)$ as we saw in the last lecture of RBF basis? Why they are not equivalent? What does "equivalent" here mean?

Also, why now "we are using them as inner product"? Is it because we now regard $k(x_i, x_j)$ as the inner product of z_i and z_j, which are some magical transformation of x_i and x_j? (Like $k(x_i, x_j) = (1 + x_i^T x_j)^p$ is the inner product of z_i and z_j, which are polynomial transformation of x_i and x_j)?

Chenliang Zhou 8 months ago Oh so is my following reasoning correct?:

Let Z and $\tilde{Z}$ be as defined in lecture 22a.

In Gaussian RBF basis, $\tilde{y} = \tilde{Z}(Z^T Z + \lambda I)^{-1} Z^T y = \tilde{Z} Z^T (Z Z^T + \lambda I)^{-1} y$.

In Gaussian RBF kernel, we have $\tilde{y} = \tilde{K}(K + \lambda I)^{-1} y$ where where K and $\tilde{K}$ are those 2 horrible matrices for Gaussian RBF kernels. Since they are the same formula, K = Z and $\tilde{K}$ = $\tilde{Z}$, so $\tilde{y} = \tilde{Z}(Z + \lambda I)^{-1} y$.

So Gaussian RBF basis and Gaussian RBF kernel are different because in general,
$\tilde{Z} Z^T (Z Z^T + \lambda I)^{-1}$(for G-RBF basis) $\neq \tilde{Z}(Z + \lambda I)^{-1}$(for G-RBF kernel).

- Kernel Regression
- MLE & MAP
- SGD

MLE & MAP: definition

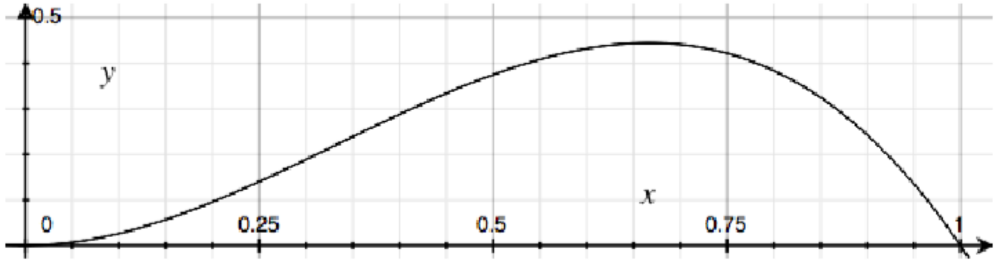
Suppose we have a dataset 'D' with parameters 'w'.

MLE: $\underset{w}{\operatorname{argmax}}\, p(D|w)$

MAP: $\underset{w}{\operatorname{argmax}}\, p(D|w)p(w)$

MLE & MAP: definition

- We can plot the likelihood p(HHT | w) as a function of ‘w’:



- Notice:
  - Data has probability 0 if w=0 or w=1 (since we have ‘H’ and ‘T’ in data).
  - Data doesn’t have highest probability at 0.5 (we have more ‘H’ than ‘T’).
  - This is a probability distribution over ‘D’, not ‘w’ (area isn’t 1).
- Maximum likelihood estimation (MLE):
  - Choose parameters that maximize the likelihood: $\hat{w} \in \underset{w}{\text{argmax}}\{p(D|w)\}$
    - In this example, MLE is 2/3.

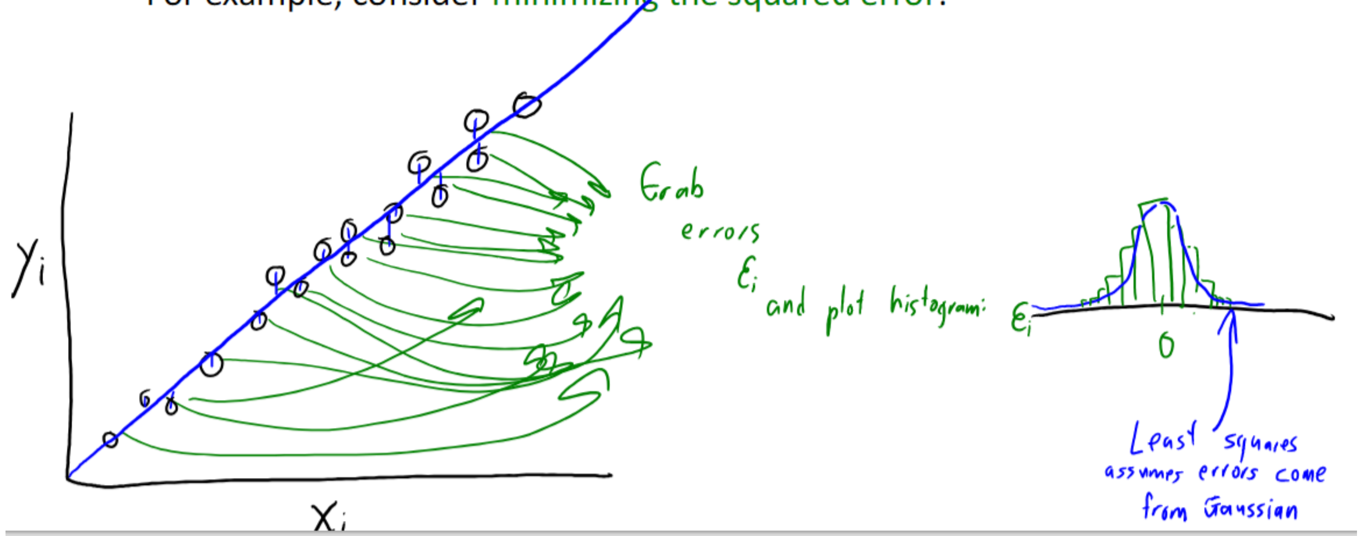
MLE & MAP: definition

$$\text{MAP: } \underset{w}{\operatorname{argmax}}\; p(D|w)p(w)$$

- Given D={HHT}
- What about: we have no information about the coin?
- What about: the coin comes from National Bank?
- What about: the coin comes from a private Casino, and H means you lose?

MLE & MAP: Relationship between MLE and Gaussian, L2 loss

– For example, consider minimizing the squared error:



$$f(w) = \frac{1}{2}\|Xw - y\|^2$$

$$y_i = w^T x_i + \varepsilon_i$$

where each ε_i is sampled independently from standard normal

$$\hat{w} \in \underset{w}{\operatorname{argmax}}\{p(D|w)\} \equiv \underset{w}{\operatorname{argmin}}\{-\log(p(D|w))\}$$

- Let's assume that $y_i = w^T x_i + \varepsilon_i$, with ε_i following standard normal:

$$p(\varepsilon_i) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\varepsilon_i^2}{2}\right)$$

also known as "Gaussian" distribution

- This leads to a Gaussian likelihood for example 'i' of the form:

$$p(y_i | x_i, w) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)$$

- Finding MLE (minimizing NLL) is least squares:

$$f(w) = -\sum_{i=1}^{n} \log(p(y_i | w, x_i))$$

$$= -\sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)\right)$$

$$= -\sum_{i=1}^{n}\left[\underbrace{\log\left(\frac{1}{\sqrt{2\pi}}\right)}_{\text{constant in 'w'}} + \underbrace{\log\left(\exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)\right)}_{\text{operations cancel}}\right]$$

$$= -\sum_{i=1}^{n}\left[(\text{constant}) - \frac{1}{2}(w^T x_i - y_i)^2\right]$$

$$= (\text{constant}) + \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2$$

$$= (\text{constant}) + \frac{1}{2}\|Xw - y\|^2$$

MLE & MAP: what about MAP?

- By again taking the negative of the logarithm as before we get:

$$\hat{w} \in \underset{w}{\operatorname{argmin}} \left\{ -\sum_{i=1}^{n} \underbrace{\left[\log\left(p(D_i \mid w)\right)\right]}_{\text{loss}} - \underbrace{\log\left(p(w)\right)}_{\text{regularizer}} \right\}$$

Observing Evidence

Domain Knowledge

- Kernel Regression
- MLE & MAP
- SGD

SGD: popular methods in DL

Motivation: Big-N Problems

- Consider fitting a least squares model:

$$f(w) = \frac{1}{2} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

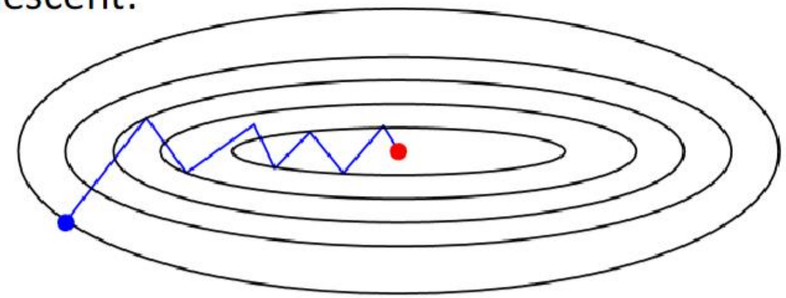
- Gradient methods are effective when ‘d’ is very large.
  - O(nd) per iteration instead of $O(nd^2 + d^3)$ to solve as linear system.
- But what if number of training examples ‘n’ is very large?
  - All Gmails, all products on Amazon, all homepages, all images, and so on.

SGD: popular methods in DL

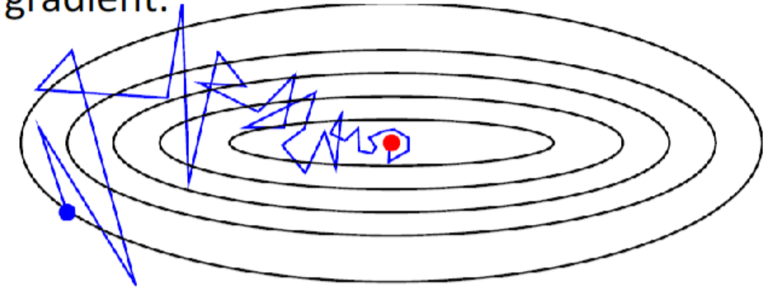
$$\nabla f(w) = \sum_{i=1}^{n} \underbrace{(w^T x_i - y_i)}_{\text{scalar}} \underbrace{x_i}_{d \times 1}$$

$$\nabla f_i(w) = \underbrace{(w^T x_i - y_i)}_{\text{scalar}} \underbrace{x_i}_{d \times 1}$$

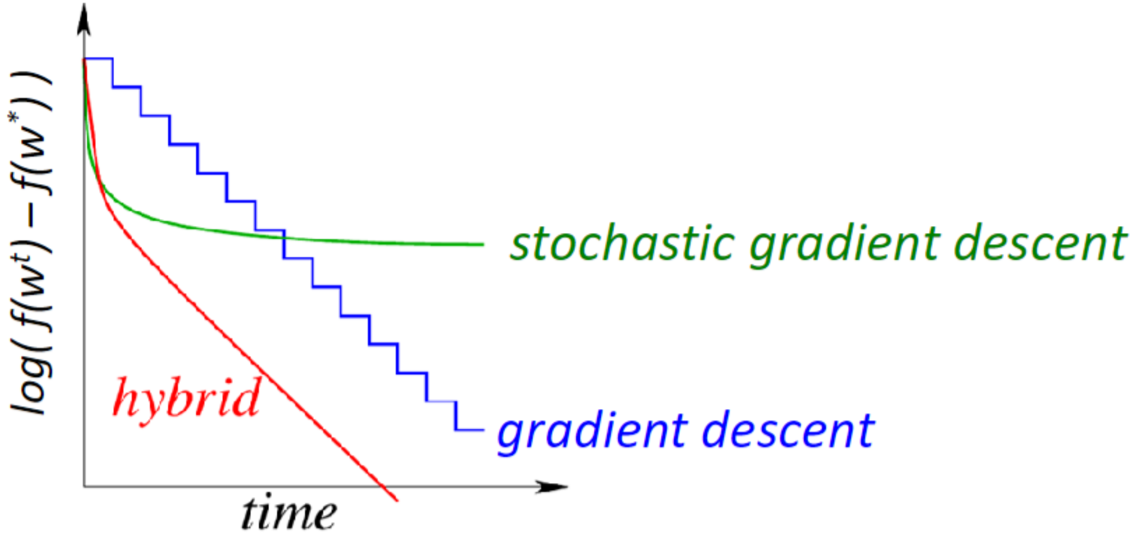
- Gradient descent:

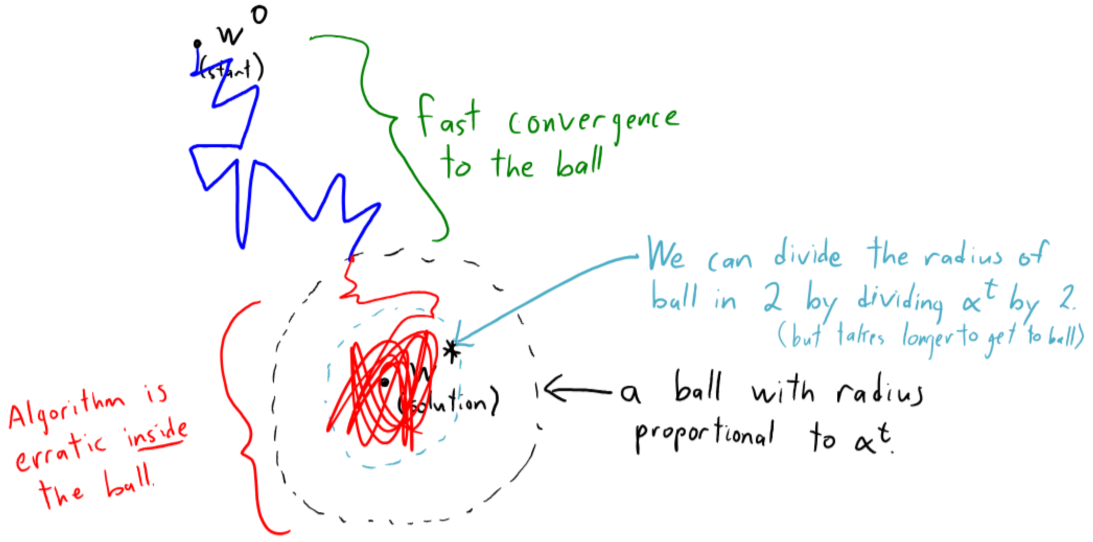


- Stochastic gradient:



Gradient Descent vs. Stochastic Gradient vs. Hybrid





Thanks for your time!
Questions?