PERSONAL NOTES

Principles of Large Scale ML

Hanfeng Zhai

Disclaimer: These notes are intended solely for personal reference and study purposes. They represent my own understanding of the course material and may contain errors or inaccuracies. The content presented here should not be considered as an authoritative source, and reliance solely on these materials is not recommended. If you notice any materials that potentially infringe upon the copyright of others, please contact me at hz253@cornell.edu so that appropriate action can be taken. Your feedback is greatly appreciated.

2023

Scaling → ML
↑
principles.

How scale impact performance of ML system.

Data Collection → Data cleaning → Feature Engineering (Embedding) → Training (Robust)

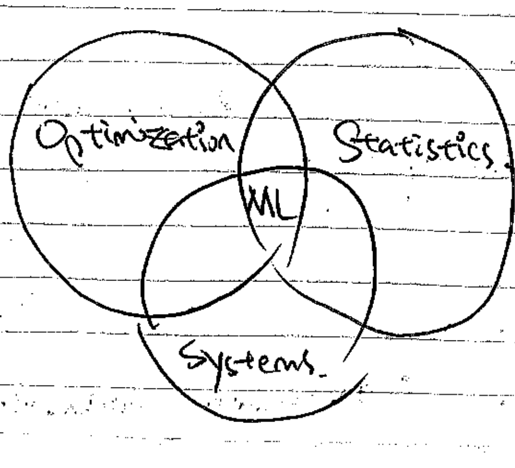
Train / Valide / Test. Splitting

Model selection (Hyperpara. Opt.)

(Online learning)

Deployment

Inference → Monitoring.



Principle #1: Optimization. Tasks

Solve it using gradient-base algorithms there are fast & "canned".

backpropagation + numerical linear algebra

- gradient descent.
- empirical risk min.

Principle #2: ~~whole dataset~~, → subsample

- Stochastic gradient d.
  - ↳ subsample the component losses.
- compute → faster.
- cross-validation / train ~~val test split~~
- bagging. → subsample with ~~[illegible]~~
- kernel sampling.
- in CNN
- dropout.
- data augmentation.
- VAE. (dimension reduction).

Principle #3. Use algorithms that are compatible w/ hardware. or vice versa.

- batch size. selection.
- numerical precision.
- GPU ⟷ NN.
- Caching
  - data loader / streaming
    - ↕
    - GPU.
- using fast numerical linear algebra kernels.
- ~~[illegible]~~ parallel / distributed computing.
- ~~[illegible]~~ neuromorphic computing. (chip → NN).

cs.cornell.edu/courses/cs4787.

+ Canvas. + Gradescope → problem.
+ ed discussion. + CMS - programming a. (python, numpy, torch)

Grading.

Psets

PAs.

Prelims

Final.

Paper reading.

Review sets - linear algebra

▷ vector calculus

▷ logistics

▷ computing w/ python

- Office hours

Wednesday 2-3 pm

Gates 426.

Week 1 - 2.

Aug. 24.

Principle 1 : ML as optimization.

continuous optimization.

↓

optimizing real numbers.

Data.

→ Embedding steps → $\mathbb{R}^d$

vectors.

e.g. pred. ppl.

- 33
- ~~g.~~ ~~w~~ 9.
- 2013 . current job / cornell alum.
- 58,000 / yr.

$\hookrightarrow \begin{bmatrix} - \\ - \\ - \\ - \end{bmatrix} \Rightarrow \mathbb{R}^4.$

Vector - element in vector space.

$x, y \in V \quad x+y \in V$

$+$ associated & commutative.

multiplication ..

~~AAA~~ Standard property of a vector space.

Typical example. $\mathbb{R}^n$. $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$

e.g. Matrix $\mathbb{R}^{n \times n}$.
$\mathbb{R}^{n \times n \times p}$ } can all be interpreted as vectors.

Core idea under: basis in linear algebra

how many numbers does it take / pinpoint that "vector space".

A basis for V is the subset for

$B = \{x_1 x_2 x_3\}$. s.t. for every v in V - v can be written as a linear combination of vectors in B.

exists some $x_1, x_2, \ldots x_n \in B$,

and $a_1, a_2, \ldots a_n$.

s.t. $v = \sum_{i=1}^{n} a_i x_i$ $\quad V = \text{span}(B)$

and $\forall x \in B$,

$|B|$ = "Dimension" $\quad x \notin \text{span}(B / \{x\})$

terminology confusion

Numerical linear algebra

numpy, sympy, ...

$V = \mathbb{R}^2$

$B = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$.

$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

numpy faster python.

- multiprocessing
- C++
- Cache

* & @ easily mixed

$\downarrow$ multiply per element. $\downarrow$ tensor / matrix multiply

broadcast $\rightarrow$ faster than manually comp.

(1, 3, 7) ✓ (1, 3, 7) ✓ i.e. loop replacement.
(3, 1, 7) $\rightarrow$ (3, 3, 7) (3, 1, 7) $\rightarrow$ (3, 3, 7)

(3, 1, 5)
(3, 1, 6) X

~~...~~ numpy broadcasting.

Week 2 (1) Aug. 29.

Automatic diff.

Definition — $\nabla f(w)^T$ is a linear map such that: $\Delta \in \mathbb{R}^d$

$$\nabla f(w)^T \Delta = \lim_{a \to 0} \frac{f(w + a\Delta) - f(w)}{a}$$

$f(x) = x^T A x$.

$$\lim_{a \to 0} \frac{(x + a\Delta)^T A (x + a\Delta) - x^T A x}{a}$$

$$\lim_{a \to 0} \frac{x^T A x + a\Delta^T A x + a x^T A \Delta + a^2 \Delta^T A \Delta - x^T A x}{a}$$

$$\lim_{a \to 0} \Delta^T A x + x^T A \Delta + a \Delta^T A \Delta$$

$$\Delta^T A x + x^T A \Delta$$

$$\Delta^T (A x + A^T x)$$

$$\nabla f(x) = A x + A^T x$$

all elements.

$$f(x) = \|x\|_2^2 = \sum_{i=1}^{d} x_i^2$$

$$\frac{\partial f}{\partial x_i} = \frac{\partial}{\partial x_i} x_i^2 = 2x_i$$

$$\nabla f(x) = 2x. \qquad f(x+a\Delta) = (x+a\Delta)^T (x+a\Delta)$$

Euclidean norm

L_1 Norm.

$$f(x) = \|x\|_1 = \sum_{i=1}^{d} |x_i|$$

$$\frac{\partial f}{\partial x_i} = \frac{\partial}{\partial x_i} |x_i| = \text{sign}(x_i) = \frac{x_i}{|x_i|}$$

$$\nabla f(x) = \text{sign}(x).$$

- Symbolic differentiation.

write function in mathematical expression.

↓

apply chain rule, ...

↓

code.

→ Problems.

had to do it manually.

- converting code into mathe not trivial
- no guarantee → comput. structure.

why important?

differentiating f_r function.

→ Numerical differentiation.

- high-order different.
-
- error accumulating,

Problems.

- numerical imprecision.
  - ↳ number of operations increases.
- f not smooth → wrong results.
- unclear how to setup ε
- for a vector → scalar function, you have to compute each partial individually, meaning $O(d)$ blowup in cost.
- if ε too small, may get zeros or NaN

~~AA~~ Automatic differentiation.

- Forward mode
- Reverse mode.

replace y with a tuple & a derivative

* operator overloading

Problems w/ Forward Mode AD.

- Differentiate with respect to one input
- Blow-up proportional to d. (dimension)

if we are computing a gradient of f:

$\mathbb{R}^d \to \mathbb{R}$.

Week 2 - 2

Aug 31.

Backpropagation.

Review for Forward Mode AD.

$x \in \mathbb{R}$.

$\downarrow$

$\frac{\partial y}{\partial x} \longrightarrow$ Store the tuple: $(y, \frac{\partial y}{\partial x})$.

☆ Python supports operator overloading

Reverse - Mode AD

→ fix one output l over $\mathbb{R}$.

↳ compute partial derivative $\frac{\partial l}{\partial y}$.

$(y, \nabla_y l)$.

Same shape

- Derive backprop. thru chain rule.

$l = f(u), \; u = g(y)$

$\ell = P(u_1, u_2, \ldots, u_k)$, $u_i = g_i(y)$

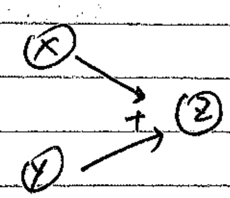
$$\nabla_y \ell = \sum_{i=1}^{k} Dg_i(y)^T \nabla_{u_i} \ell .$$

derivative

is a Jacobian

compute gradient w.r.t. y.

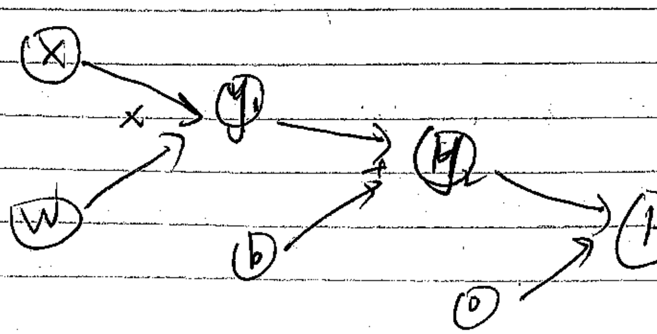
$$\underbrace{\nabla_y \ell}_{\text{process } y} = \sum_{i=1}^{k} \underbrace{Dg_i(y)^T \nabla_{u_i} \ell}_{\text{process } u_i}$$



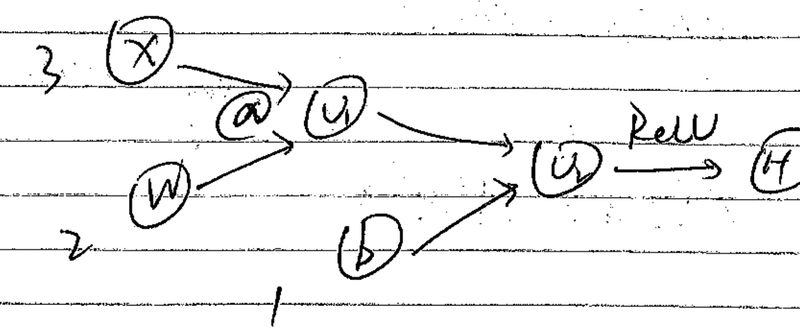
$z = x + y$

$H = \max(0, Wx + b)$. $W \in \mathbb{R}^{c \times d}$

$x \in \mathbb{R}^d$



$H \in \mathbb{R}^c$



$$\nabla_x H = \frac{\partial H}{\partial U_2} \cdot \frac{\partial U_2}{\partial g} \cdot \frac{\partial g}{\partial U_1} \cdot \frac{\partial U_1}{\partial x}$$

$H = f(U_2)$. $U_2 = g(U_1)$, $U_1 = h(x)$.

$\nabla_x H = \mathrm{ReLU}^{-1}$.

initialize.

$\nabla_{U_2} H = \nabla_{U_1} H = \nabla_W H = \nabla_x H = \nabla_b H = 0$.

$\nabla_{U_2} H \mathrel{+}= \mathrm{ReLU}'(U_2) \cdot \nabla_H H = 1 \cdot 1 = 1$.

Problems

1.2. Derivative of x^TAx.

b. $\Delta^T A x + x^T A \Delta$ why $x^TA\Delta = (x^TA\Delta)^T$?

$\downarrow$

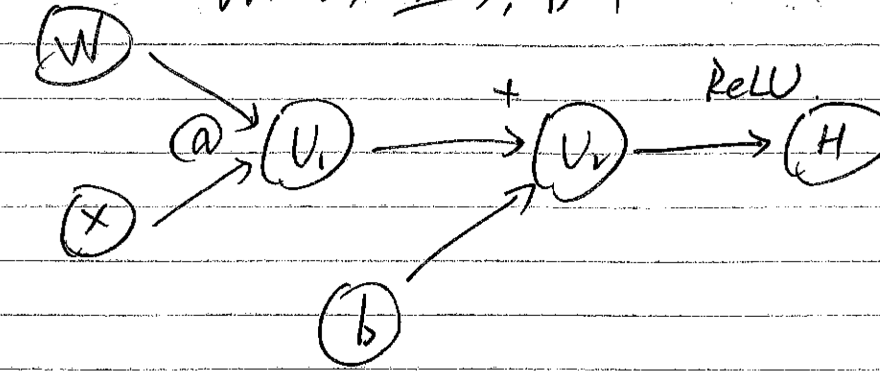
$\Delta^T A x + \Delta^T A^T x$

d. → a Jacobian.

2.5. why sum(log) in np.

Review: Backprop.

$W=2,\ X=3,\ b=1$



Forward

- $U_1 = W * X = 2*3 = 6$.
- $U_2 = U_1 + b = 7$
- $H = ReLU(U_2) = ReLU(WX+b) = ReLU(7)$

- $\nabla_H H = 1$.
- initialize: $\nabla_{U_2}H = \nabla_{U_1}H = \nabla_W H$
  $= \nabla_x H = \nabla_b H = 0$
- $\nabla_{U_2}H \mathrel{+}= ReLU'(U_2)\nabla_H H = 1*1 = 1$.
- $\nabla_{U_1}H \mathrel{+}= \nabla_{U_2}H = 1$.
- $\nabla_b H \mathrel{+}= \nabla_{U_2}H = 1$
- $\nabla_W H \mathrel{+}= x\nabla_{U_1}H = 3*1 = 3$

$\nabla_x H + = W \nabla_u H = 2 * 1 = 2$

$F(x) \longrightarrow DF(x)$

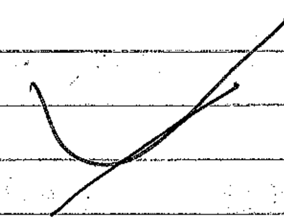
$\Delta \mapsto$

A^{-1}

Labor day.

Week 3 - 9/2.

Machine learning systems



$f : U \mapsto V$

$Df(x) \quad \mathcal{L}(U, V)$

$f : \mathbb{R}^m \to \mathbb{R}^n$

$Df(x) \in \mathcal{L}(\mathbb{R}^m, \mathbb{R}^n)$

$\mathbb{R}^{m \times n}$

if $F(x) = X^2$ $\quad A \in \mathbb{R}^{n \times n}$

$$\lim_{\alpha \to 0} \frac{F(X + \alpha\Delta) - F(X)}{\alpha}$$

$$\lim_{\alpha \to 0} \frac{X^2 + \alpha \Delta X + \alpha X \Delta + \alpha^2 \Delta^2 - X^2}{\alpha}$$

$= \Delta X + X \Delta$

$DF(x) = (\Delta \mapsto \Delta X + X \Delta)$

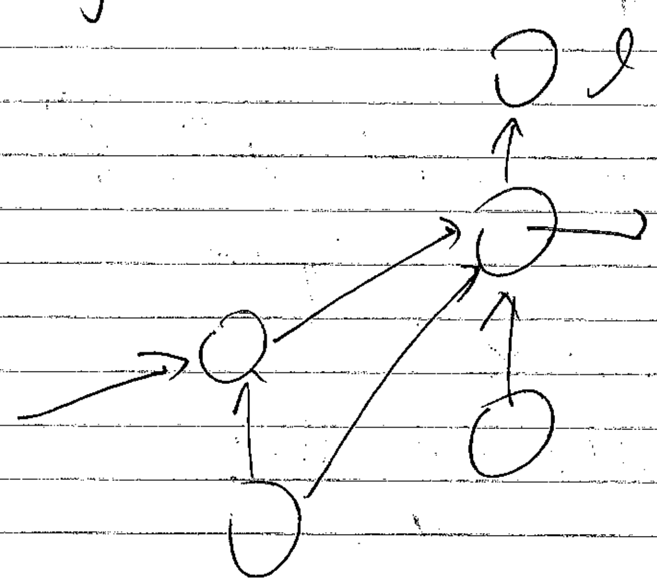
or $(DF(x))(\Delta) = \Delta X + X \Delta$

Why Backprop.

gradient has 1 output l.

Backprop manifest gradient

$\nabla_y l$ for intermediates y.



class / object

↓

knows how
to compute itself

↓

knows its dependencies

+

def __add__(self, other):

two ways:

(lazy)

EAGER v.s. GRAPH (only later)

compute immediately

don't compute, but know the dependencies

immaterialized

↑

Eager	Graph
↓ cutoff	∘ fast computation.
↓ Error	∘ simplification (transform. ✓)
∘ take storage	∘ reuse the graph. (✓)
∘ debugging & development (X).	∘ good for caching (?).
∘ if / branch / control flow (✓).	∘ memory locality (✓).
	∘ less python overhead (✓)

☆ Deployment.

↓

convert it into some form ~~that~~ "in python"

- optimization

TF $\xrightarrow{\text{Default}}$ Graph

PT ⟶ Eager

ML framework.

- numerical linear algebra. library.
- hardware accelerator support.
- backprop engine. (gradient. auto).
- library for writing Deep Nets.

PyTorch & TensorFlow

MxNet (C++)   Jax.   Flux (Julia)

Caffe (ancient) → 1st GPU acc.

```
model = torch.nn.Sequential(
            nn.Linear( ... )
            nn.Linear( ... )
        )
```

hardware - accelerator.

```
x.to("cuda")
x.to("cpu")
y.to("cpu")
z = x + y.
```

throw an exception when two diff vars on diff hardwares.

Programming Assignment #1.

def get_next_order

return to a "+1" order

def to_backprop

convert array to backprop

* class BackpropArray.

def _init_ (np array).

def _repr ( )

↦ String.

1.1

use graph search alg.

☆ def all_dependencies( )

find all the dependencies

1.2

☆ def backward( )

backward ↦ find gradients

Sort the dependencies in reverse node

def grad_fn ( )

"Define math operators in backprop"

def add

→ BA_Add( )

def sub

→ BA_Sub( )

def mul

→ BA_Mul( )

def truediv

→ BA_Div( )

def sum ( )

→ BA_Sum( ).

def reshape( )

→ BA_Reshape( )

def transpose ( )

→ BA_Transpose( )

Different classes

implement for scalar_arrays.

1.3

Tensor operation

2.1

2.3

compute "grad"

doesn't modified for tensor operation

2.2

compare ad, nd, sd

1.5

global funs.

- def numerical_diff( ).
- def numerical_grad( ).
- def backprop_diff( ).

~~2.3~~ 2.4 2.5 2.6

Store test functions in a class.

class TestFxs( ):

def {f1, df1dx, f2, ...}

wish'd input ~~test implementation: TestFxs. df3dx~~ 1.4

if __name__ == "__main__":

scalar output

write the script to execute

convergence ... main functions.

for #3:

$$\lim_{\alpha \to 0} \frac{\log(x+\alpha\Delta) - \log(x)}{\alpha} = D(f(x))\Delta$$

$$\frac{\exp(x+\alpha\Delta) - \exp(x)}{\alpha}$$

$$= \frac{\exp(x)\cdot\exp(\alpha\Delta) - \exp(x)}{\alpha}$$

$$= \frac{\exp(x)\cdot(\exp(\alpha\Delta) - \exp(0))}{\alpha\cdot\exp(0)}$$

$$=$$

Week 4 - Lecture 7.

GD

we compute gradient in large-scale opt. problem to solve learning task.

examples will most be on supervised learning, but not limited to ...

in SL : $f(w) = \frac{1}{n} \sum_{x,y \in D} \mathcal{L}(h_w(x), y)$

n size of dataset.

$= \frac{1}{n} \sum_{x \in D} f(w; x)$

in framework, we strive to:

minimize $f(w)$ over $w \in \mathbb{R}^d$.

computing f takes $O(n)$ time.

Init $w_0 = 0$ → not nec. zero. gradient.

$w_{t+1} = w_t - \alpha \nabla f(w_t)$. → gradient descent.

☆ How much time does it cost?

→ $O(ndk)$.

☆ How much memory required?

→ $O(nd)$ or $O(d)$

parallel way

running backprop thru whole thing.

in a NAIVE way → $O(nd)$.

Newton's Method

$w_{t+1} = w_t - [\nabla^2 f(w_t)]^{-1} \nabla f(w_t)$

↳ converges faster.

"BUT" more expensive than GD

So, ☆ How much time? & ☆ How m. mem? store Hessian matrix.

→ $O(nd^2 k)$. → $O(d^2)$

ppl scared away from Newton's method cuz the large memory required, i.e. quadratic GD.

☆ why are we confident that GD will converge?

the value of gradient are computed based on particular weight, if move a lil' bit, it will drastically change grads

we want to make it robust.

How close GD ~?

⟶ bound Derivative.

↳ Second-order D. is also Bd.

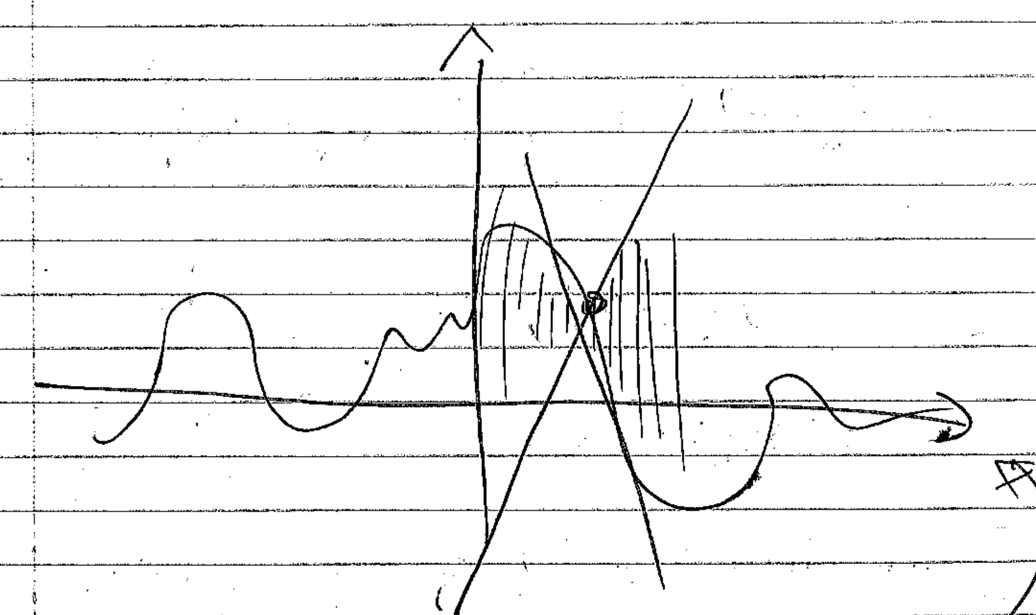
Gradient ∇f is L- Lipschitz continuous

$\|\nabla f(u) - \nabla f(v)\|_2 \le L \|u - v\|_2$

Assume $\exists f$ s.t. $f(w) \ge f^*$.

$\forall w, \forall u, \|u\| = 1.$

$$\left\| \frac{\partial^2}{\partial \alpha^2} f(w + \alpha u) \right\| \le L.$$



Lipschitz continuous

↳ L- smooth.

How to prove GD converges???

$f(w_{t+1}) = f(w_t - \alpha \nabla f(w_t))$

fundamental thm. of calc.

$$\Downarrow$$

$$= f(w_t) + \int_0^{\alpha} \frac{\partial}{\partial \eta} f(w_t - \eta \nabla f(w_t))\, d\eta$$

$$\Updownarrow$$

$$f(b) - f(a) = \int_a^b \frac{\partial}{\partial x} f(x)\, dx.$$

$$= f(w_t) + \int_0^{\alpha} (-\nabla f(w_t))^T \nabla f(w_t - \eta \nabla f(w_t))\, d\eta.$$

how?

$\hookrightarrow$ imagine $= 0$

close to grad.

let $h(\eta) = f(w_t - \eta \nabla f(w_t))$

$$h(\alpha) = h(0) + \int_0^{\alpha} h'(\eta)\, d\eta$$

Expect η to be small $\leftarrow$ expect

α : small

$$\frac{\partial}{\partial \eta} f(w_t - \eta \nabla f(w_t))$$

$$= \lim_{\delta \to 0} \frac{f(w_t - (\eta + \delta) \nabla f(w_t)) - f(w_t - \eta \nabla f(w_t))}{\delta}$$

$$= \lim_{\delta \to 0} \frac{f(w_t - \eta \nabla f(w_t) - \delta \nabla f(w_t)) - f(w_t - \eta \nabla f(w_t))}{\delta}$$

$$= \lim_{\delta \to 0} \frac{f(\hat{w} + \delta(-\nabla f(w_t))) - f(\hat{w})}{\delta}$$

$$= (-\nabla f(w_t))^T \nabla f(\hat{w}).$$

$\Rightarrow$ close to norm. $\rightarrow$ always positive $\Rightarrow$ gd works to min

$\rightarrow$ cont.

$$= f(w_t) + \int_0^{\alpha} \left(-\nabla f(w_t)^T \nabla f(w_t)\right) d\eta$$

$$+ \int_0^{\alpha} \left(-\nabla f(w_t)\right)^T \left(\nabla f(w_t - \eta \nabla f(w_t)) - \nabla f(w_t)\right) d\eta.$$

Apply Cau. theorem:

$$A \cdot B \leq \|A\| \, \|B\|$$

$$\{ \|A \, B\| \}$$

$$\leq f(w_t) + \int_0^{\alpha} \left(-\nabla f(w_t)^T \nabla f(w_t)\right) d\eta$$

$$+ \int_0^{\alpha} \|-\nabla f(w_t)\| \, \|\nabla f(w_t - \eta \nabla f(w_t)) - \nabla f(w_t)\| \, d\eta.$$

$$\leq f(w_t) + \int_0^{\alpha} -\|\nabla f(w_t)\| \qquad d\eta$$

$$\frac{\alpha^2}{2} = \int_0^{\alpha} \eta \, d\eta$$

$$+ \int_0^{\alpha} \|-\nabla f(w_t)\| \cdot L \cdot \|\eta \nabla f(w_t)\| \, d\eta.$$

$$f(w_t) \leq f(w_t) - \alpha \|\nabla f(w_t)\|^2$$

$$+ \frac{\alpha^2 L}{2} \|\nabla f(w_t)\|^2$$

$$\leq f(w_t) - \alpha\left(1 - \frac{\alpha L}{2}\right) \|\nabla f(w_t)\|^2.$$

$\Rightarrow$ How does constant L affect how we learn tasks in scales

$\downarrow$

~~there's~~ why

assume: $\alpha L < 1$.

$$\leq f(w_t) - \frac{\alpha}{2} \|\nabla f(w_t)\|^2$$

$$\frac{\alpha}{2}\|\nabla f(w_t)\|^2 \le f(w_t) - f(w_{t+1})$$

Single iteration of GD.

$$\frac{\alpha}{2}\sum_{t=0}^{k-1}\|\nabla f(w_t)\|^2 \le \sum_{t=0}^{k-1}\left(f(w_t) - f(w_{t+1})\right)$$

k iterations $< f(w_0) - f(w_k)$.

$$\frac{\alpha}{2}\sum_{t=0}^{k-1}\|\nabla f(w_t)\|^2 \le \underline{f(w_0) - f(w_t)}$$

assume

f is bounded from below. ☆☆☆

$$= \le f(w_0) - f^*.$$

$$\frac{\alpha}{2}k \min_{t\in\{0,k-1\}}\|\nabla f(w_t)\|^2 \le f(w_0) - f(w_t)$$

$$\min_{t\in\{0,k-1\}}\|\nabla f(w_t)\|^2 \le \frac{2(f(w_0) - f^*)}{\alpha k}.$$

require $\alpha L \le 1$.

LHS run k iteration of GD

return the optimal weights.

RHS upper bound of func.

Step size α to be large as possible to $\frac{1}{\alpha} \to$ small

the largest step

$$\le \frac{2L(f(w_0) - f^*)}{k}.$$

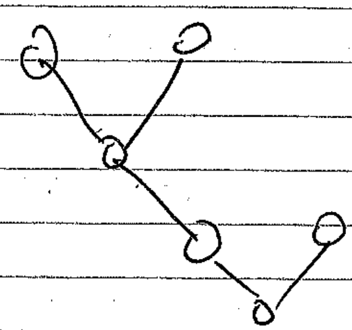
GD only find local opt.

OH (Sep. 13)

(1.1)

- Breadth-first search ?

what are the dependencies



(1.2)

what is the utility of the grad-fn

why zero out ".grad" ?

(1.3)

1.3.4. — why ?

(2)

generally, how write the help function to manipulate the dimensions ?

(2.5) numerical grad. — hints ?

(2.7).

$\mathbb{R}^d \to \mathbb{R}$

confused on the implementation.

② writing a for loop.

while the len

① add empty row / cols. for extra dim. -> new axis.

check the dims are targetting the same

-> broadcasting.

A: $1 \times 6 \times 5$

B: $1 \times 1 \times 2$

→ iterating over the axis.

1: adding B.
2: repeating axis (iterating)

helper_func

BA_And

just recall $+=$ helper(a, b)

Lecture 8. (wk. 4).

HW Review:

$\nabla_X l$ has same shape as X

X.grad.shape = X.data.shape.

* "+=" in numpy mutates the vector

self.X.grad += stuff

self.X.grad = self.X.grad + stuff.

* init both as arrays as all zeros

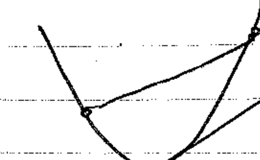
GD continued.

... last time.

$W_{t+1} = W_t - \alpha \nabla f(W_t)$.

→ Most simple condition - convexity

0th-order convexity



$f((1-\alpha)x + \alpha y) \le (1-\alpha)f(x) + \alpha f(y)$

for $\alpha \in [0, 1]$

▽ 1st order def. — considering whether func. const.

$$f(x) + (y-x)^T \nabla f(x) \le f(y)$$

▽ 2nd order def.

$$\frac{\partial^2}{\partial \alpha^2} f(x + \alpha u) \ge 0 \quad \text{for any } \alpha, u \in \mathbb{R}^d$$

"parabola has a const. sec. deriv."

- convex has a unique global optimum.

"that's why we care 'bout con"

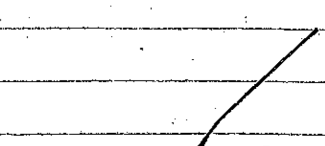
Strongly convex function

"a function that can be bounded from below"

μ- strongly convex function. — scalar positive num.

$$\frac{\partial^2}{\partial \alpha^2} f(x + \alpha u) \ge \mu \quad (\text{for } \|u\| = 1)$$

E.g. "Strongly convex"



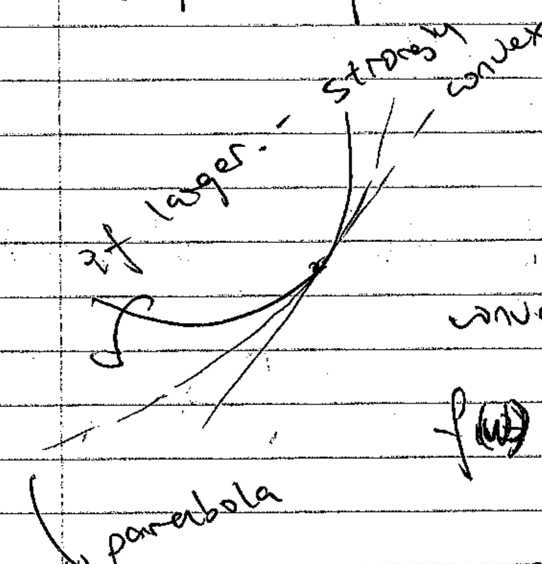
$f(x) = \max(x, 0)$

Convexity: vector space $\mapsto$ real num.
i.e. scalar.

Strictly convex vs. Strongly convex

e.g. $f(x) = e^x$ — but not larger than a positive val.

$$f(y) \ge f(x) + (y-x)^T \nabla f(x) + \frac{\mu}{2}\|y - x\|^2$$



f is μ- strongly convex, iff $\exists g$ s.t.

$$f(w) = g(w) + \frac{\mu}{2}\|w\|^2$$

& g is convex.

☆☆☆

Use in a lot of analysis.

Polyak - Lojasiewicz (PL)

... TBC

min val.

$$\|\nabla f(w)\|^2 \geq 2\mu\,(f(w) - f^*)$$

strongly convex.

but does not imply strongly convex

e.g. $f(w) = \frac{\mu}{2}\|w\|^2$

$$\nabla f(w) = \mu w$$

$$\|\nabla f(w)\|^2 = \|\mu w\|^2 = \mu^2\|w\|^2$$

$$= 2\mu\left(\frac{\mu}{2}\|w\|^2\right) = 2\mu\,(f(w) - f^*)$$

☆ the gradient is only close to zero when its only ~~ete~~ around global optimum.

Statisfies PL yet not strongly cv.

$$f(w_{t+1}) = f(w_t - \alpha\nabla f(w_t))$$

$$\leq f(w_t) - \alpha\left(1 - \frac{\alpha L}{2}\right)\|\nabla f(w_t)\|^2$$

Non-general conver of gd.

$$\leq f(w_t) - 2\mu\alpha\left(1 - \frac{\alpha L}{2}\right)(f(w_t) - f^*)$$

$$f(w_{t+1}) - f^* \leq (f(w_t) - f^*) - 2\mu\alpha\left(1 - \frac{\alpha L}{2}\right)(f(w_t) - f^*)$$

$$\leq \underbrace{\left(1 - 2\mu\alpha\left(1 - \frac{\alpha L}{2}\right)\right)}_{\text{call it } \gamma}(f(w_t) - f^*)$$

$$\leq \gamma\,(f(w_t) - f^*)$$

min

min is γ

$$f(w_k) - f^* \leq \gamma^k (f(w_0) - f^*)$$

if $\gamma > 0$.

to show $1 - \frac{\mu}{L}$ is non-n

☆ How should we chose α?

$$2\mu\alpha - \frac{2\mu\alpha^2 L}{2} \to 0$$

$$1 - \alpha L = 0$$

$$\alpha L = 1$$

$$\alpha = \frac{1}{L}.$$

we got $\gamma = 1 - 2\mu\frac{1}{L}\left(1 - \frac{\frac{1}{L}L}{2}\right) = 1 - \frac{\mu}{L}$

$$f(w_k) - f^* \leq \left(1 - \frac{\mu}{L}\right)^k (f(w_0) - f^*)$$

$$\leq \exp\left(-\frac{\mu k}{L}\right) \cdot (f(w_0) - f^*)$$

convergence @ a linear rate

even tho is decreasing exponentially

converges with errors ~ num. of digits.

going to be linear wrt. ...

"roughly linear with digits wrt. score".

Goal: output $\hat{w}$ with

$$f(\hat{w}) - f^* \leq \varepsilon \mapsto \text{error tol.}$$

$$(\varepsilon \in \mathbb{R},\ \varepsilon > 0).$$

it suffices to run T iterations of (num timestep) (GD), s.t.

$$\exp\left(-\frac{\mu}{L}T\right)(f(w_0) - f^*) \leq \varepsilon.$$

we can solve this for T

$$\exp\left(\frac{\mu}{L}T\right) \geq \frac{f(w_0) - f^*}{\varepsilon}.$$

$$\frac{\mu}{L}T \geq \log\left(\frac{f(w_0) - f^*}{\varepsilon}\right).$$

$$T \geq \frac{L}{\mu}\log\left(\frac{f(w_0) - f^*}{\varepsilon}\right)$$

hessian for s.c.

pos. def.

symm.

ratio of largest eigen val of ~~hessian~~ / the smallest val.

$$\frac{L}{\mu} = K \quad \text{"condition number".}$$

$\|\nabla f(x) - \nabla f(y)\| \le L \|x-y\|$

$\left|\frac{\partial^2}{\partial \alpha^2} f(x+\alpha u)\right| \le L$

(for $\|u\| = 1$)

eqly. $-LI \preceq \underbrace{Hf(x)}_{\nabla^2 f} \preceq LI$

k large → lot of steps

k small → a few steps.

$k \geq 1$ ⟼ $k = 1$ isotropic quadratic

Condition number ∝ num. of steps to solve the opt. problem.

the other thing impact time of GD.

→ size of dataset.

↓

Stochastic GD ⟸ GD goes ~~bad~~ when DS are large.

batch size

Subsampling ← approx.

↓

Principle #2 of LS ML.

Spoiler: ~~g~~ faster than GD

GD # steps. $\sim \mathcal{O}(K)$.

~~SGD~~ GD # ~~steps~~ time $\sim \mathcal{O}(ndK)$

Lecture 7

Week 3.

GD time $\quad O(nK \log(\frac{f_0 - f^*}{\varepsilon}))$

Subsampling

training set size

num. of iterations of GD.

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w).$$

$$\text{GD} \begin{cases} w_{t+1} = w_t - \alpha \nabla f(w_t) \\ \quad = w_t - \frac{\alpha}{n}\sum_{i=1}^{n} \nabla f_i(w_t) \end{cases}$$

SGD: sample @ random $i \in \{1, n\}$.

$$w_{t+1} = w_t - \alpha \nabla f_i(w_t).$$

mini-batch SGD: Sampling B

$$w_{t+1} = w_t - \frac{\alpha}{|B|}\sum_{b=1}^{B} \nabla f_{i_b}(w_t)$$

batch size

sample i_b uniformly from $\{1, n\}$ for each $b \in \{1, B\}$

Stochastic GD (mini-batch)

$$E[w_{t+1} \mid w_t] = w_t - \alpha \nabla f(w_t)$$

$$E[\nabla f_i(w_t) \mid w_t] = \sum_{j=1}^{n} P(i=j) \nabla f_j(w_t)$$

$$= \sum_{j=1}^{n} \frac{1}{n} \nabla f_j(w_t).$$

$$= \nabla f(w_t).$$

Assume $\exists L > 0$ s.t.

$$\|\nabla f(w) - \nabla f(v)\| \le L\|w - v\|$$

Assume. $f(w) \ge f^*$

let g_t denote $\frac{1}{B}\sum_{b=1}^{B} \nabla f_{i_b}(w_t) = g_t$

$$f(w_{t+1}) = f(w_t - \alpha g_t) = f(w_t) + \int_0^\alpha \frac{\partial}{\partial \eta} f(w_t - \eta g_t)\, d\eta. = f(w_t) + \int_0^\alpha (\nabla f(w_t - \eta g_t))(-g_t)\, d\eta$$

full loss function

$$= f(w_t) - \alpha \nabla f(w_t)^T g_t + \int_0^\alpha (\nabla f(w_t - \eta g_t) - \nabla f(w_t))^T (-g_t)\, d\eta$$

$$\le f(w_t) - \alpha \nabla f(w_t)^T g_t + \int_0^\alpha \|\nabla f(w_t - \eta g_t) - \nabla f(w_t)\| \|g_t\|\, d\eta$$

$$\le f(w_t) - \alpha \nabla f(w_t)^T g_t + \int_0^\alpha L \|\eta g_t\| \|g_t\|\, d\eta$$

$$\le f(w_t) - \alpha \nabla f(w_t)^T g_t + \frac{\alpha^2 L}{2} \|g_t\|^2$$

full-batch grad

$$f(w_{t+1}) \le f(w_t) - \alpha \nabla f(w_t)^T g_t + \frac{\alpha^2 L}{2} \|g_t\|^2$$

$\nabla f(w_t)$ Stochastic grad.

$$\mathbb{E}[f(w_{t+1}) | w_t] \le f(w_t) - \alpha \nabla f(w_t)^T \mathbb{E}[g_t | w_t] + \frac{\alpha^2 L}{2} \mathbb{E}[\|g_t\|^2 | w_t]$$

$$\le f(w_t) - \alpha_t \nabla f(w_t)^T \nabla f(w_t) + \frac{\alpha^2 L}{2} \mathbb{E}[\|g_t\|^2 | w_t]$$

$$\le f(w_t) - \alpha \|\nabla f(w_t)\|^2 + \frac{\alpha^2 L}{2} \mathbb{E}[\|g_t\|^2 | w_t]$$

New Assumption

uniform over w_t

$$\frac{1}{n} \sum_{i=1}^n \|\nabla f_i(w) - \nabla f(w)\|^2 \le \sigma^2$$

for any $w \in \mathbb{R}^d$.

$$\mathbb{E}[\|g_t\|^2 | w_t] = \mathbb{E}\left[\left\|\frac{1}{B} \sum_{b=1}^B \nabla f_{i_b}(w_t)\right\|^2 \middle| w_t\right]$$

$$= \mathbb{E}\left[\left(\frac{1}{B} \sum_{b=1}^B \nabla f_{i_b}(w_t)\right)^T \left(\frac{1}{B} \sum_{c=1}^B \nabla f_{i_c}(w_t)\right) \middle| w_t\right]$$

$$= \frac{1}{B^2} \sum_{b=1}^B \sum_{c=1}^B \mathbb{E}[\nabla f_{i_b}(w_t)^T \nabla f_{i_c}(w_t) | w_t]$$

independent each other

$b \ne c$

samples that $b \ne c$

$$= \frac{1}{B^2}(B^2 - B) \|\nabla f(w_t)\|^2 + \frac{1}{B^2} \sum_{b=1}^B \mathbb{E}[\|\nabla f_{i_b}(w_t)\|^2 | w_t]$$

same

$$= \frac{B-1}{B} \|\nabla f(w_t)\|^2 + \frac{1}{B^n} \sum_{i=1}^{n} \|\nabla f_i(w_t)\|^2$$

Back to the assumption:

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w_t)\|^2 - \underbrace{\frac{2}{n}\sum_{i=1}^{n} \nabla f_i(w)^T \nabla f(w)}_{-2\|\nabla f(w)\|^T \nabla f(w)}}$$

$$+ \underbrace{\frac{1}{n}\sum_{i=1}^{n} \|\nabla f(w)\|^2}_{\text{Easy to bound } \|\nabla f(w)\|^2} \le \sigma^2$$

$$\Downarrow$$

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w)\|^2 - \|\nabla f(w)\|^2 \le \sigma^2$$

$$\le \frac{B-1}{B} \| f(w_t)\|^2 + \frac{1}{B}(\sigma^2 + \|\nabla f(w)\|^2)$$

$$= \|\nabla f(w_t)\|^2 + \frac{\sigma^2}{B}$$

~~coming back to old assumption~~

$$\mathbb{E}[f(w_{t+1}) | w_t] \le f(w_t) - \alpha \|\nabla f(w_t)\|^2$$

$$+ \frac{\alpha^2 L}{2} \|\nabla f(w_t)\|^2 + \underbrace{\frac{\alpha^2\sigma^2 L}{2B}}_{\text{variance of the square error of SGD}}$$

$$\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y|X]]$$

Also assume: $\underbrace{\|\nabla f(w)\|^2 \ge 2\mu(f(w) - f^*)}_{\text{PL condition}}$

$$\sim \le f(w_t) - \alpha\left(1 - \frac{\alpha L}{2}\right) 2\mu (f(w_t) - f^*) + \frac{\alpha^2\sigma^2 L}{2B}$$

$$\mathbb{E}[f(w_{t+1}) - f^* | w_t] \le \left(1 - \alpha\left(1 - \frac{\alpha L}{2}\right)2\mu\right)(f(w_t) - f^*) + \frac{\alpha^2\sigma^2 L}{2B}$$

SGD additional "noise" term.

$$\Leftarrow \rho_t = \mathbb{E}[f(w_t) - f^*]$$

$$\rho_{t+1} \le \left(1 - \alpha\left(1 - \frac{\alpha L}{2}\right)2\mu\right)\rho_t + \frac{\alpha^2\sigma^2 L}{2B}$$

$\Rightarrow$ Assume $\alpha L \le 1$.

$$\le (1 - \alpha\mu)\rho_t + \frac{\alpha^2\sigma^2 L}{2B}$$

$$\rho_\alpha = (1-\alpha\mu)\rho_\infty + \frac{\alpha^2\sigma^2 L}{2B}$$

$$\alpha\mu\rho_\infty = \frac{\alpha^2\sigma^2 L}{2B}$$

$$\rho_\alpha = \frac{\alpha\sigma^2 L}{2\mu B} = \frac{\alpha\sigma^2 K}{2B}$$

$$\rho_{t+1} - \frac{\alpha\sigma^2 L}{2\mu B} \le (1-\alpha\mu)\rho_t - \frac{\alpha\sigma^2 L}{2\mu B}$$

$$+ \frac{\alpha^2\sigma^2 L}{2B}$$

$$\le (1-\alpha\mu)\left(\rho_t - \frac{\alpha\sigma^2 L}{2\mu B}\right)$$

$$\rho_T - \frac{\alpha\sigma^2 L}{2\mu B} \le (1-\alpha\mu)^T\left(\rho_0 - \frac{\alpha\sigma^2 L}{2\mu B}\right)$$

Step

$$\le (1-\alpha\mu)^T\rho_0$$

$$\mathbb{E}\left[f(w_T) - f^*\right] \le (1-\alpha\mu)^T(f(w_0)-f^*)$$

$$+ \frac{\alpha^2\sigma^2 L}{2\mu B}$$ noise wall / noise floor

$\Rightarrow 1 - x \le \exp(-x)$ (plug in)

LHS $\le \exp(-\alpha\mu T)(f(w_0) - f^*) + \dfrac{\alpha\sigma^2 L}{2\mu B}$

Lecture 9. Week 5 -2.

Rev - last time.

$$\mathbb{E}\left[f(w_T) - f^*\right] \le \exp(-\alpha\mu T)(f(w_0) - f^*) + \frac{\alpha L \sigma^2}{2B\mu}.$$

Goal: $\mathbb{E}\left[f(w_{out}) - f^*\right] \le \varepsilon$.

suffice for $\varepsilon \ge \exp(-\alpha\mu T)(f(w_0) - f^*) + \frac{\alpha L\sigma^2}{2B\mu}$.

also suffice $\frac{\varepsilon}{2} \ge \exp(-\alpha\mu T)(f(w_0) - f^*)$

and $\frac{\varepsilon}{2} \ge \frac{\alpha L \sigma^2}{2B\mu}$.

$$\Rightarrow \alpha \le \frac{B\mu\varepsilon}{L\sigma^2}.$$

$$\log\left(\frac{\varepsilon}{2(f(w_0) - f^*)}\right) \ge -\alpha\mu T.$$

$$\Rightarrow T \ge \frac{1}{\alpha\mu}\cdot\log\left(\frac{2(f(w_0) - f^*)}{\varepsilon}\right)$$

also $\alpha L \le 1$.

$$\alpha \le \min\left(\frac{B\mu\varepsilon}{\sigma^2}, 1\right)\cdot\frac{1}{L}$$

$$T \ge \max\left(\frac{\sigma^2}{B\mu\varepsilon}, 1\right)\frac{L}{\mu}\log\left(\cdots \frac{2\,f(w_0) - f^*}{\varepsilon}\right)$$

$$\ge \max\left(\frac{\sigma^2}{B\mu\varepsilon}, 1\right)\cdot K\cdot\log(\sim)$$

runtime of SGD: $\mathcal{O}\left(\max\left(\frac{\sigma^2}{B\mu\varepsilon}, 1\right)\cdot B\cdot K\cdot\log\left(\frac{1}{\varepsilon}\right)\right)$

usually, ε are small

$$\tilde{\mathcal{O}}\left(\frac{\sigma^2}{\cancel{B}\mu\varepsilon}K\right)$$

batch B cancels

ignoring "log" term.

runtime of GD. $\mathcal{O}(nK \log(\frac{1}{\varepsilon}))$

for GD $n \longrightarrow \frac{\sigma^2}{B\varepsilon}$.

GD	SGD
◦ convex (strongly).	◦ non-convex
◦ Global opt. guaranteed (unique).	◦ multi-objective
	◦ huge dataset
◦ large memory. (GPU)	◦ huge n
parallel	ε → large "ish"
◦ really small ε	or not too small
◦ large σ^2	

How to set batch size B

⇓

☆ Hardware consideration.

e.g. $f_i(w) = \frac{1}{2}(x_i^T w - y_i)^2$.

$\nabla f_i(w) = x_i(\widehat{x_i^T w} - y_i)$ dot axpy

minibatch $X \in \mathbb{R}^{B \times d}$. $Y \in \mathbb{R}^B$.

$\nabla f_{batch}(w) = \frac{1}{B} X^T (Xw - Y)$

faster mmpy $(B, d) @ (d, 1)$,

$(d, B) @ (B, 1)$

$(1024, 1024) @ (1024, 1024)$ } a lot less than 1024 ×
$(1024, 1024) @ (1024, 1)$

Minibatch size ∝ hardware itself

e.g. $B = 256, 64, \ldots$

SGD → GD: sampling w/o replacemnt.

$\max(n, 1)$ → is GD scenario

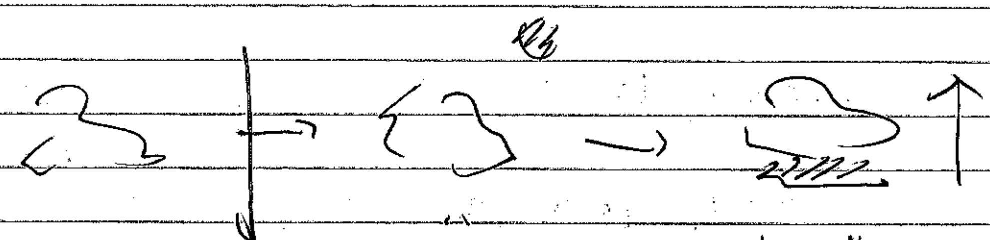
larger, slower

random reshuffling

sample w/o repl., only reuse example after every example has been used..

epoch $\Rightarrow$ 1 pass thru the training set (e.g. 100~200).

shuffle - once



"same - order"

Statistically not -

Use a fixed step size.

Diminishing step size (assuming $\alpha_t L \le 1$)

$$\mathbb{E}[f(w_{t+1}) - f^*] \le (1-\alpha_t\mu)\,\mathbb{E}[f(w_{t+1}) - f^*] + \frac{\alpha_t^2\sigma^2 L}{2B}$$

$$\rho_{t+1} \le \underbrace{(1-\alpha_t\mu)\rho_t + \frac{\alpha_t^2\sigma^2 L}{2B}}$$

$$\Rightarrow 0 = -\mu\rho_t + \frac{\alpha_t\sigma^2 L}{B}$$

$$\Rightarrow \alpha_t \simeq \frac{\rho_t\mu B}{\sigma^2 L}$$

$$\rho_{t+1} \le \rho_t - \frac{\mu^2 B}{2\sigma^2 L}\rho_t^2$$

$$\frac{1}{\rho_{t+1}} \ge \frac{1}{\rho_t - \frac{\mu^2 B}{2\sigma^2 L}\rho_t^2} \quad \left(\frac{1}{\rho_t}\right)$$

$$= \frac{1}{\rho_t}\left(1 - \frac{\mu^2 B}{2\sigma^2 L}\rho_t^2\right)^{-1}.$$

$$\ge \frac{1}{\rho_t}\left(1 + \frac{\mu^2 B}{2\sigma^2 L}\rho_t\right) = \frac{1}{\rho_t} + \frac{\mu^2 B}{2\sigma^2 L}.$$

$$\Rightarrow \frac{1}{\rho_T} \ge \frac{1}{\rho_0} + \frac{\mu^2 B T}{2\sigma^2 L} \Rightarrow \rho_T = \mathbb{E}[f(w_T) - f^*]$$

$$\leq \left(\frac{1}{f(w_0) - f^*} + \frac{\mu^2 BT}{2\sigma^2 L} \right)^{-1}$$

$$= \mathcal{O}\left(\frac{1}{T}\right)$$

$$\simeq \mathcal{O}\left(\frac{\sigma^2 K}{\mu BT}\right)$$

Lecture 11 Week 6.

Momentum.

Rev: $K \simeq \frac{L}{\mu}$; grad. noise $\sim\left(\frac{\sigma^2}{\mu}\right)$

Announcement: • Pset 3

• PA 2.

Rev: GD.

$\ell \Rightarrow f(w) - f^* \leq \exp\left(-\frac{T}{K}\right)(f(w_0) - f^*)$

gap after T epochs

Q. Simplist model with large K?

a 2D problem → Quadratic

e.g. $f(w) = \frac{1}{2}(w_1^2 L + w_2^2 \mu)$

$= \frac{1}{2} w^T \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix} w$

$\nabla^2 f(w)$

$\Rightarrow H(w) = \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix}$ Suppose $L > \mu > 0$

$$\nabla f(w) = \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix} w$$

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

$$= w_t - \alpha \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix} w_t$$

$$= \begin{bmatrix} 1-\alpha L & 0 \\ 0 & 1-\alpha \mu \end{bmatrix} w_t$$

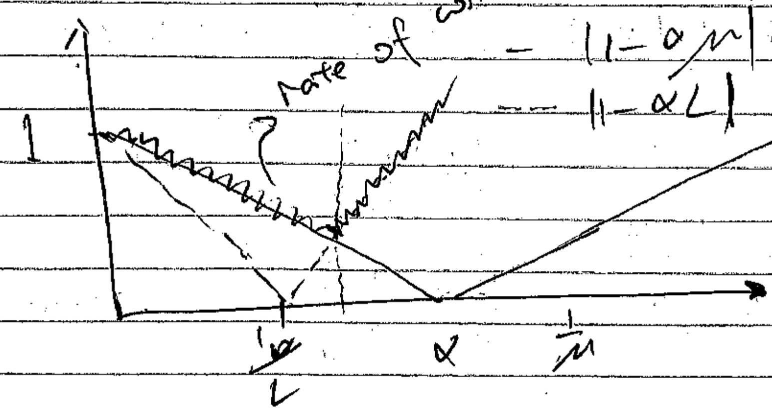
power "T"

$$\Rightarrow w_T = \begin{bmatrix} 1-\alpha L & 0 \\ 0 & 1-\alpha \mu \end{bmatrix}^T w_0$$

$$= w_T - \begin{bmatrix} (1-\alpha L)^T & 0 \\ 0 & (1-\alpha \mu)^T \end{bmatrix} w_0$$

$$f(w_T) = \frac{1}{2}\left(L(1-\alpha L)^{2T} (w_0)_1^2 + \mu(1-\alpha\mu)^{2T}(w_0)_2^2 \right)$$

rate of convergence



$$\alpha = \frac{2}{L+\mu}$$

$$= \frac{1-\mu}{L+\mu} =$$

$$= 1 - \frac{2}{K+1}$$

$$\Rightarrow f(w_T) = O\left(\left(1-\frac{2}{K+1}\right)^{2T}\right) = O\left(\exp\left(-\frac{4T}{K+1}\right)\right)$$

$$w_{t+1} = w_t - \alpha \nabla f(w_t) + \beta(w_t - w_{t-1})$$

previous step ~~curve~~ the next step, step size too large, overshooting the object.

"Self-tune" the step size

$$w_{t+1} - w_t = \cancel{-\alpha\nabla f(w_t)}$$

$$\beta(w_t - w_{t-1}) - \alpha \nabla f(w_t)$$

we need

$0 < \beta < 1$

"friction term"

or momentum.

particle undergoes "some" force term

How Polyak momentum applies to 2D system?

$$W_{t+1} = W_t - \alpha \underbrace{\begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix}}_{A} W_t + \beta(W_t - W_{t-1})$$

$$\begin{bmatrix} W_{t+1} \\ W_t \end{bmatrix} = \begin{bmatrix} W_t - \alpha A W_t + \beta(W_t - W_{t-1}) \\ W_t \end{bmatrix}$$

$$= \begin{bmatrix} (1+\beta)I - \alpha A & -\beta \\ I & 0 \end{bmatrix} \begin{bmatrix} W_t \\ W_{t-1} \end{bmatrix}$$

$$\begin{bmatrix} W_{T+1} \\ W_T \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)I - \alpha A & -\beta I \\ I & 0 \end{bmatrix}}^{T} \begin{bmatrix} W_1 \\ W_0 \end{bmatrix}$$

~~4x4~~ block matrix

$$\begin{bmatrix} (1+\beta) - \alpha L & -\beta & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & (1+\beta) - \alpha\mu & -\beta \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Eigenvalue

use x to find eigenvalues.

$$\left| \begin{bmatrix} (1+\beta) - \alpha\lambda - x & -\beta \\ 1 & 0 - x \end{bmatrix} \right| = x^2 - (1+\beta + \alpha\lambda)x + \beta \quad (2a)$$

$$x = \frac{1+\beta-\alpha\lambda \pm \sqrt{(1+\beta-\alpha\lambda)^2 - 4\beta}}{2}$$

complex vals.

Polyak gives:

$$\alpha = \frac{L+2\beta}{L+\mu}, \quad \sqrt{\beta} = \frac{\sqrt{K}-1}{\sqrt{K}+1}.$$

$$|x|^2 = \frac{1}{4}\left[(1+\beta-\alpha\lambda)^2 + 4\beta - (1+\beta-\alpha\lambda)^2\right]$$

$T \approx \sqrt{K}$. $\quad \sqrt{\beta}^T \approx \exp\left(-\frac{2T}{\sqrt{K}+1}\right)$

$T \approx \sqrt{K}$

Works when $4\beta \geq (1+\beta-\alpha\lambda)^2$

$|2\sqrt{\beta}| \geq |1+\beta+\alpha\lambda|$

$\hookrightarrow 2\sqrt{\beta} \geq 1+\beta+\alpha\lambda \geq -2\sqrt{\beta}$

$\mu \le \lambda \le L$

Lecture 11. - Week 6.

Conditioning.

Rev. - momentum.

$$T = O\left(nK \log\left(\tfrac{1}{\varepsilon}\right)\right)$$

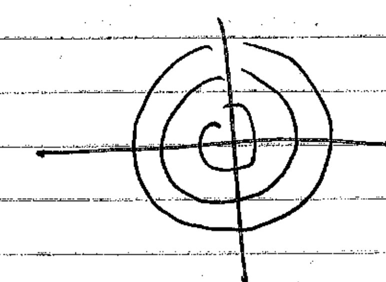
condition num.

Could be too large / too small.

last lec.

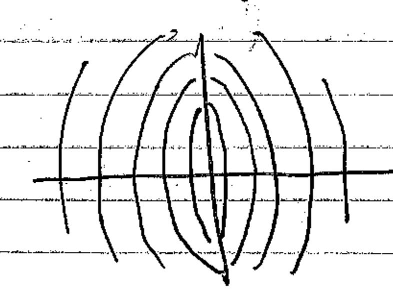
$$T_{mom} = O\left(n\sqrt{K} \log\left(\tfrac{1}{\varepsilon}\right)\right).$$

$$f(w_1, w_2) = \tfrac{1}{2}w_1^2 + \tfrac{1}{2}w_2^2 \quad (K=1)$$



$C = w_1^2 + w_2^2$

$$f(w_1, w_2) = \tfrac{10}{2}w_1^2 + \tfrac{1}{2}w_2^2$$



$C = aw_1^2 + bw_2^2$

"poorly-conditioned" prob.

$w = Pu$

$\min_w f(w) = \min_u f(Pu)$

ALA. $|P| \neq 0$

invertible

any matrix, invertible

$\arg\min_w f(w) = P\left(\arg\min_u f(Pu)\right)$

let $g(u) = f(Pu)$.

graphically, this is essentially "stretching" the poorly-conditioned problem, i.e. apply linear operation

$\Downarrow$

map back $\leftarrow$ reduced K

$\Downarrow$

$\underbrace{\text{solve for optimal}}_{\text{preconditioning.}}$

Instead of saying minimize $f(w)$ over w - solve minimize $f(Pu)$ over u (for clever P).

e.g. $f(w) = \frac{1}{2} w^T A w$. $A^T = A$

- how to set P?. what cond can we get?. (PD, $A > 0$, all eigenval. A are positive)

symmetric.

$P^T A P = I$

$2 f(Pu) = (Pu)^T A (Pu) = u^T P^T A P u$

$= u^T u = \|u\|^2$

suppose P symmetric, invertible

- $PAP = I$

- $P^{-1} P A P P^{-1} = P^{-1} I P^{-1}$

$A = P^{-2} = P = A^{-\frac{1}{2}}$

replace eigenval.

in transformed space.

$g(u) = f(Pu) \rightarrow \nabla g(u) = P^T \nabla f(Pu)$

$u_{t+1} = u_t - \alpha \nabla g(u_t) = u_t - \alpha P^T \nabla f(Pu_t)$

assert: $w = Pu$; $w_t = Pu_t$

$Pu_{t+1} = Pu_t - \alpha PP^T \nabla f(Pu_t)$.

$w_{t+1} = w_t - \alpha \underbrace{PP^T}_{R} \nabla f(w_t)$

Preconditioner

$w_{t+1} = w_t - \alpha R \nabla f(w_t)$

positive invertible definite symmetric.

Preconditioned GD

$R = A^{-1}$

$x^T R x = x^T PP^T x$

$= (P^T x)^T (P^T x)$

$= \|P^T x\|^2 > 0$

chose R to be diagonal preconditioner

R diagonal $\longrightarrow$ stretch along the coordinates' axis

memory $O(d)$.

time multiply $O(d)$.

$(w_{t+1})_i = (w_t)_i - \alpha R_{ii} (\nabla f(w_t))_i$.

let $\alpha_i = \alpha R_{ii}$.

$(w_{t+1})_i = (w_t)_i - \alpha_i (\nabla f(w_t))_i$

$w_{t+1} = w_t - \alpha \odot \nabla f(w_t)$

element-wise

run GD in implicit scaled space.

AdaGrad

loop

$g \leftarrow \nabla f_i(w)$, random i

$r_i \leftarrow r_i + g_i^2$

$$\left((r_i)_t = \sum_{k=0}^{t} (g_i^2)_t \right)$$

$$W_i \leftarrow W_i - \frac{\alpha}{\sqrt{r_i}} g_i \quad \left(\alpha_i = \frac{\alpha}{\sqrt{r_i}} \right)$$

- AdaGrad.

↳ "Adaptive Optimization" Alg.

Lecture 12. Week 7.

AdaGrad.

init $w \in \mathbb{R}^d$, $r = 0 \in \mathbb{R}^d$.

loop:

get example grad $g \in \mathbb{R}^d$

$g = \nabla f(w; x)$ minibatch etc

large example

update $\underline{\underline{r}} \leftarrow r + g^2$

update $w \leftarrow w - \frac{\alpha}{\sqrt{r} + \varepsilon} \cdot g$

small number.

depends too much on the history

infinite memory.

- done in element-wise

implicitly. $r_i \approx T\,\mathbb{E}[g_i^2]$

mul. num. of steps.

↳ $\approx T(\mathbb{E}(g_i)^2 + \mathrm{Var}(g_i))$

could be dominant

→ fix this: RMSProp.

RMSProp

hyperparameter $\alpha > 0$, $\rho \in (0, 1)$.

init $w \in \mathbb{R}^d$, $r = 0 \in \mathbb{R}^d$

loop:

get gradient sample g at w.

update $r \leftarrow \rho r + (1-\rho) g^2$

update $w \leftarrow w - \frac{\alpha}{\sqrt{r} + \varepsilon} g$

$r_i \approx \mathbb{E}[g_i^2]$

typical $\rho = 0.999$

Combining momentum + RMSProp + correction for 0-init

init $w \in \mathbb{R}^d$, $r = 0 \in \mathbb{R}^d$

$s = 0 \in \mathbb{R}^d$

momentum:

$$w_{t+1} = w_t - \alpha g_t + \beta (w_t - w_{t-1})$$

$$(w_{t+1} - w_t) = \beta (w_t - w_{t-1}) - \alpha g_t$$

$$\begin{cases} v_{t+1} = \beta v_t - \alpha g_t = \beta v_t - (1-\beta)\left[\frac{\alpha}{(1-\beta)} g_t\right] \\ w_{t+1} = w_t + v_t \end{cases}$$

α, ρ_1, ρ_2 . hyperpara.

loop: get gradient sample g @ w.

$s \leftarrow \rho_1 s + (1-\rho_1) g$

$r \leftarrow \rho_2 r + (1-\rho_2) g^2$

Update:

$w \leftarrow w - \frac{\alpha}{\sqrt{\hat{r}} + \varepsilon} \hat{s}$

$\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$

$t = \#$ of steps

$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

if g_t denotes the g drawn at step t, $\underbrace{s_t}_{s \text{ after } t \text{ steps}} = \sum_{k=0}^{t-1} \rho_1^k (1-\rho_1) g_{t-k}$

$$\sum_{k=0}^{t-1} \rho_1^k (1-\rho_1)$$

$$= \cancel{(1-\rho_1)} \left(\frac{1-\rho_1^t}{\cancel{1-\rho_1}} \right)$$

SI: exponential MA: recurrence

sequence $x_0, x_1, x_2, \ldots$

$S_0 = 0$; $S_{t+1} = \rho S_t + (1-\rho) x_t$.

tradeoff - parameters:

subsampling - batch number n.

momentum - k.

variance of the gradient.

↳ avoid having sample

Variance Reduction

→ SVRG

↳ strive to behave GD

- yet SGD each steps.

↳ Large-scale convex optimization

→ great !!

for DL - Not so good.

GD SVRG.

$\mathcal{O}(nk\log(\frac{1}{\varepsilon}))$ $\mathcal{O}((n+k)\log(\frac{1}{\varepsilon}))$

Polyak Averaging

- Output an average:

if W_t is parameter vector after t steps

$$\hookrightarrow \frac{1}{T}\sum_{t=0}^{T-1} W_t$$

Stochastic Weight averaging (SWA).

(wed.)

Lecture 15/14. Week 7.

Rev...

$\mathcal{O}(n k \log(\frac{1}{\varepsilon}))$ - GD.

$\mathcal{O}(K/\varepsilon^2)$ - SGD

momentum: $\mathcal{O}(n \sqrt{K} \log(\frac{1}{\varepsilon}))$ - MSGD.

$\mathcal{O}((n+K) \log(\frac{1}{\varepsilon}))$ SVRG.

Polyak averaging / Stochastic weight averaging

$\mathcal{O}(n K d \cdot \log(\frac{1}{\varepsilon})) \Leftrightarrow \mathcal{O}(K \sigma^2 d / \varepsilon^2)$

$\mathcal{O}(n \sqrt{K} d \log(\frac{1}{\varepsilon}))$

linear model: $w \in \mathbb{R}^d$.

$f(w) = \frac{1}{n} \sum_{x,y \in D} \mathcal{L}(w^T x, y)$.

time to compute an example grad

$\mathcal{O}(d)$

PCA $\Rightarrow$ covariance matrix $(d \times d)$.

computationally expensive

Random Projection. (Johnson-Lindenstrauss

$x \in \mathbb{R}^D \Rightarrow Ax \in \mathbb{R}^d$ Transform)

feature length

$A \in \mathbb{R}^{d \times D}$ random.

$A_{ij} \sim \mathcal{N}(0, \sigma^2)$.

~~$\leq \|Ax - Ay\|^2 \leq$~~

$(1-\varepsilon)\|x_i - x_j\|^2 \leq \|Ax_i - Ax_j\|^2 \leq (1+\varepsilon)\|x_i - x_j\|^2$

$\forall x_i, x_j \in D$

$d \geq \frac{8 \log(n)}{\varepsilon^2}$.

Why? Choose σ^2 s.t.

$\mathbb{E}[\|Ax_i - Ax_j\|^2] = \|x_i - x_j\|^2$.

$\mathbb{E}[\|A(x_i - x_j)\|^2] = \mathbb{E}[(x_i - x_j)^T A^T A (x_i - x_j)]$

$= (x_i - x_j)^T \mathbb{E}[A^T A] (x_i - x_j)$

↳ Gaussian matrix.

$\mathbb{E}[A^T A]_{ij} = \sum_{k=1}^{d} \mathbb{E}(A_{ki} A_{kj})$

$= \begin{cases} 0 & \text{if } i \neq j \\ \underbrace{\sum_{k=1}^{d} \mathbb{E}[A_{ki}^2]}_{\sigma^2} & \text{if } i = j \end{cases}$

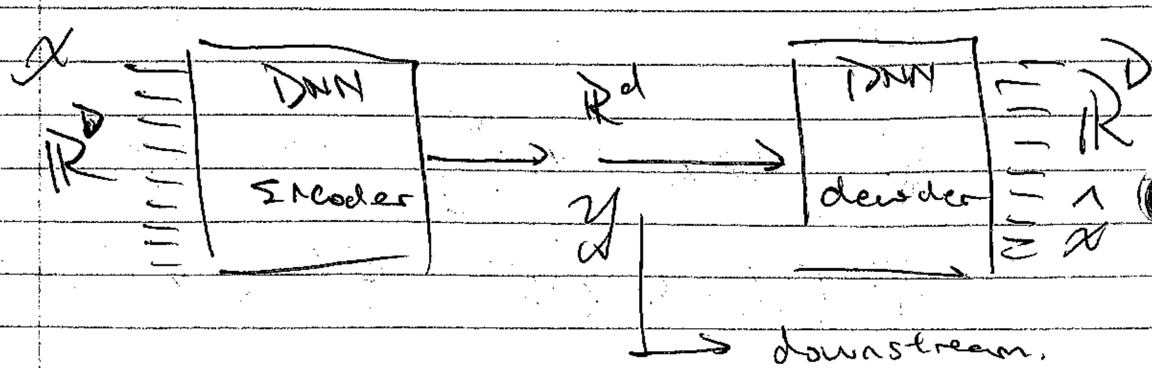
$\mathbb{E}[A^T A] = \sigma^2 d I$

$= (x_i - x_j)^T (\sigma^2 d I)(x_i - x_j)$

$= \sigma^2 d \|x_i - x_j\|^2 \overset{\text{if}}{=} \|x_i - x_j\|^2$

↳ $\sigma^2 = \frac{1}{d}$

AutoEncoders



$l(x, \hat{x})$

e.g. $\sum_{\|D\|, x \in D} \|x - \hat{x}\|_2^2$ ← unsupervised learning.

Sparsity

density of x = $\frac{\text{\# of nonzero entries } x}{\text{size of } x, d}$

$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 3 \end{bmatrix}_{\mathbb{R}^8} = \{(4, 2.0), (7, 3.0)\} =$ - indices $= [4, 7]$
- values $= [2.0, 3.0]$

16 ish

Sparse matrix.

COO
- expensive to comp.
- Easy to write

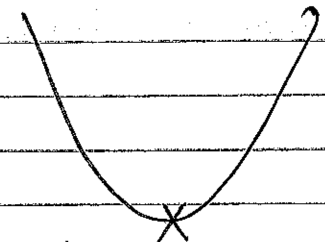
$\begin{bmatrix} 0 & 0 & 5 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \mapsto$
- row idx. $[1, 1, 2]$
- col. idx. $[3, 6, 2]$
- values $[5, 3, 1]$

CSR $\xrightarrow{\text{tr.}}$ CSC

- row offsets. $[0, 2]$ $[3, 6]$ $[2]$ ⇒ $[3, 6, 2]$
- col idxs. $[5.0, 3.0]$ $[1.0]$ $[5.0, 3.0, 1.0]$
- values

Week 8. lecture 15/16.

Deep learning.



Neural Networks

- MLP

$x \in \mathbb{R}^{d_0}$

$x \to \boxed{\text{linear}} \to O_1 \to \boxed{\sigma} \to a_1 \to O_2 \to O \to a_2 \to \cdots$

$O_1 = W_1 x + b_1$ $\quad a_1 = \sigma(O_1)$

$$\begin{cases} W_1 \in \mathbb{R}^{d_1 \times d_0} \\ b \in \mathbb{R}^{d_1} \\ O_1 \in \mathbb{R}^{d_1} \end{cases}$$

nonlinear: function acts element-wise

as the activation function.

$\sigma(x) = \text{ReLU}(x) = \max(x, 0)$.

universal: for continuous input → output.

there exists a function the approx.

with bounded parameters (good accuracy)

capacity $\xrightarrow{\text{imply}}$ overfit.

NN tend not to overfit!

→ larger models generalize better.

▷ double-descent.

$(x, y) \in \mathbb{R}^d \times \mathbb{R}$.

$|\mathbb{D}| = n$. $\quad n = d$.

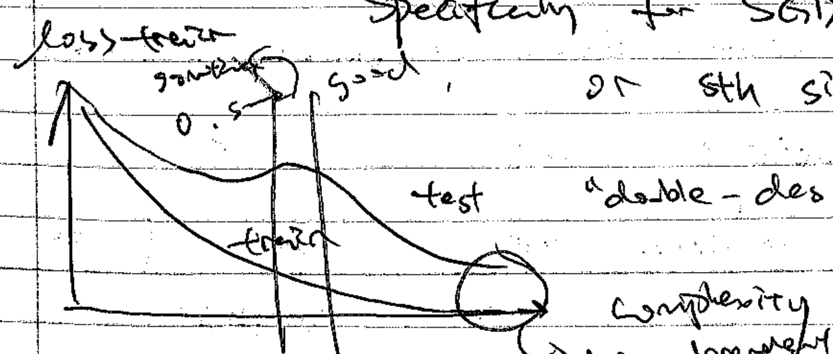
$\ell(w) = \frac{1}{n} \sum_{i=1}^{n} (x_i^T w - y_i)^2$ + regularization

larger model generalize better

Specifically for SGD, Adam.

or sth similar.

"double-descent" phenomenon.



$a_\ell, o_\ell \in \mathbb{R}^{d_\ell}$

$d_1(d_0 + d_0 - 1) + d_1$

$a_\ell = W_\ell o_{\ell-1} + b_\ell \to O(\cancel{d_0})$

$\mathbb{R}^{d} \quad \mathbb{R}^{d\times d} \quad \in \mathbb{R}^{d}$

$o_\ell = \sigma_\ell(a_\ell) \to O(d_1)$

$W_1 \in \mathbb{R}^{d_1 \times d_0}$ $\quad$ $3d \cdot \ell$.

$b_1 \in \mathbb{R}^{d_1}$

$x \in \mathbb{R}^{d_0}$

Image processing ← CNN

$x \in \mathbb{R}^{C\times W\times H}$ → height of image

width of image

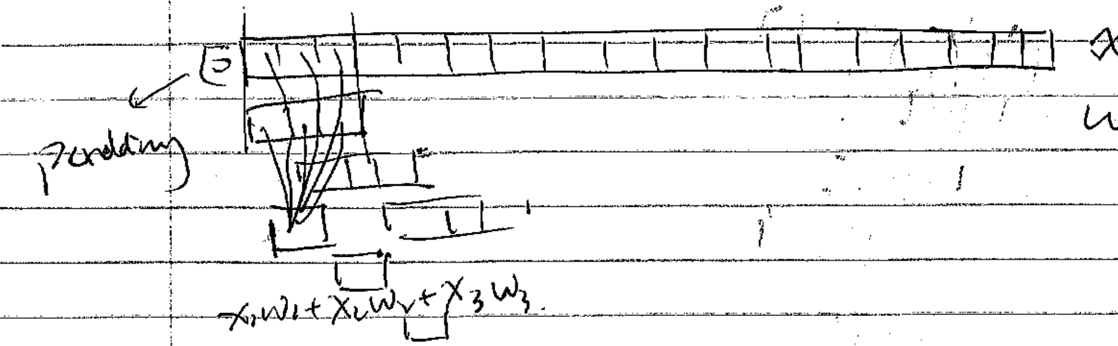
channels.

1) convolutional neural network

– restricted subset

convolutional layer in place of the linear layers.

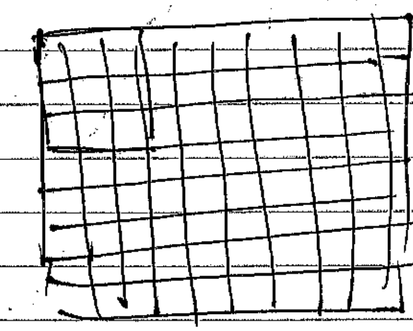
Convolution in 1D:



filter: arbitrarily set of parameters.

$x_{k+1}w_1 + x_{k+2}w_2 + x_{k+3}w_3$

in 2D



diff. channels

Conv. layer

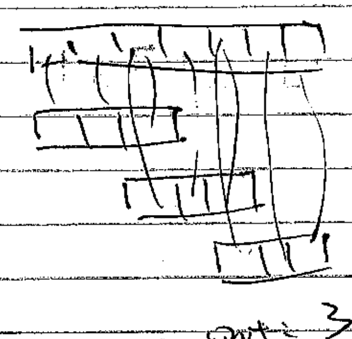
input $x \in \mathbb{R}^{C_0\times W_0\times H_0}$ $\quad$ output $a \in \mathbb{R}^{C_1\times W_1\times H_1}$

parameters $W \in \mathbb{R}^{C_1\times C_0\times K_W\times K_H}$.

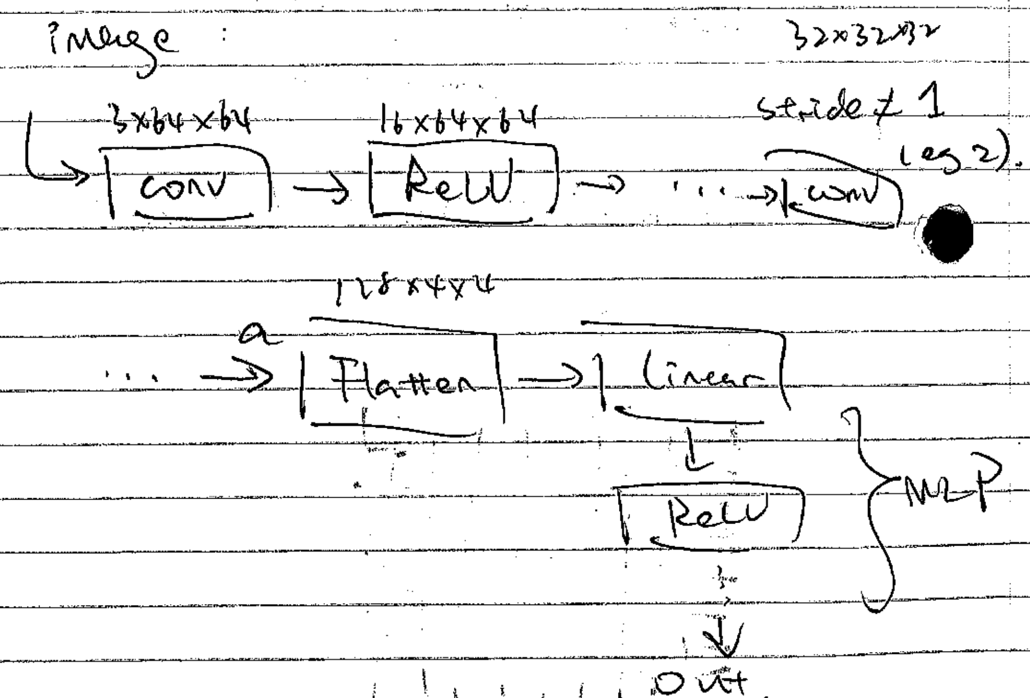
$a_i = \sum_{j=1}^{C_0} \text{conv}(\underbrace{W_{ij}}_{\text{matrix}}, x_j)$ $\quad$ $W \in \mathbb{R}^{C_1\times C_0\times K_W\times K_H}$ i o w image

$b \in \mathbb{R}^{C_1}$

· stride.



input image :



Week 9. lecture 17.

Overfitting / Underfitting ← low capacity

← model has high capacity.

(overparametrized).

DNNs high ← representational

~~many~~ weights.

weights coming ← effective.

from the learning algorithm.

Dropout.

↳ increase the representation capacity.

During training → randomly drop

(independently each step). neurons.

↓

forces the learn to employ

most neurons.

Batch Norm.

let $u \in \mathbb{R}^n$.

$$u \to \frac{u-\mu}{\sqrt{\sigma^2}} = \frac{u-\mu}{\sigma}$$

↳ instead. look at $u \in \mathbb{R}^B$

— conv. ⇒ image → per channel

— saves a running exp. average of the mean & variance

μ σ.

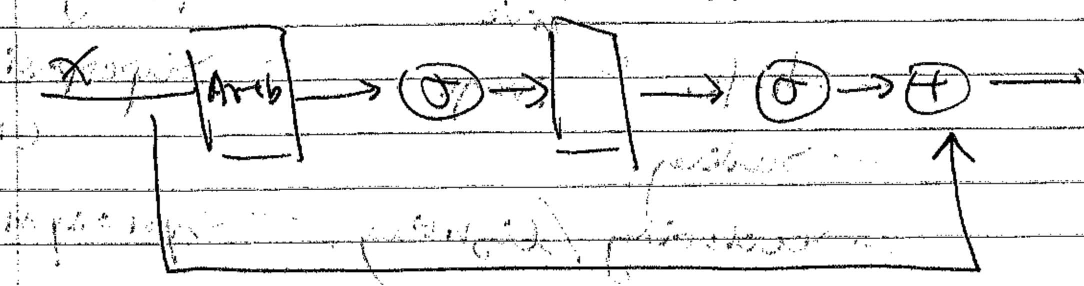
Model.train() ⇒ turning on/off
Model.eval() batch norm.

$$u \to \frac{u-\mu}{\sigma}\cdot\gamma + \beta$$

broadcast scalar learned param (SGD)

Week 9 Lecture 18

Residual connections / skip connections



ResNet.

DenseNet

$$\min \ell(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w, x_i, y_i)$$

$$(x_i, y_i) \in D \subseteq \mathcal{X} \times \mathcal{Y}$$

▷ sparse data

n too small.

* Data Augmentation.

randomness

Aug function $A: \mathcal{X} \times [0,1] \to \mathcal{X}$. random noise

$$w_{t+1} = w_t - \alpha \nabla \ell(w_t; A(x_t; \eta_t), y_t)$$

example chosen @t.

IMAGES:

- Translation
- Rotation
- Shift
- Scaling
- recoloring / lighting

move image

- Noise / Artifact Addition
- Cropping
- Superposition (mixup)
- synonym swapping
- deletion
- back-trans.

Semi-supervised learning

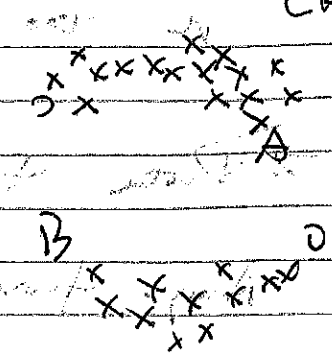
"data - cheap" "label - expens."

lots of data, few labels

assump.

Similar inputs have similar outputs.

→ (KNN)



Clustering → data is clustered

labels by cluster.

Manifold → data lie on low-dimensional manifold

PCA   Autoencoders

Week 10. lecture 19.

Sequence models.

$x \in \mathbb{R}^{d \times n}$. ← different per example.

discussion:

takes sequence as input.

↳ generate sequence output.

○ handle sequence input / output?

- padding to max length.

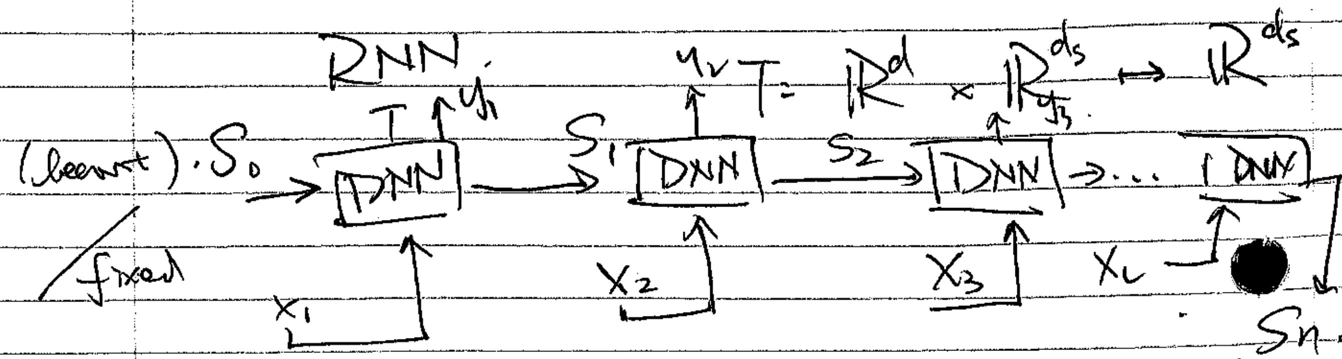
- counts / sum

- RNN ≈ sequential

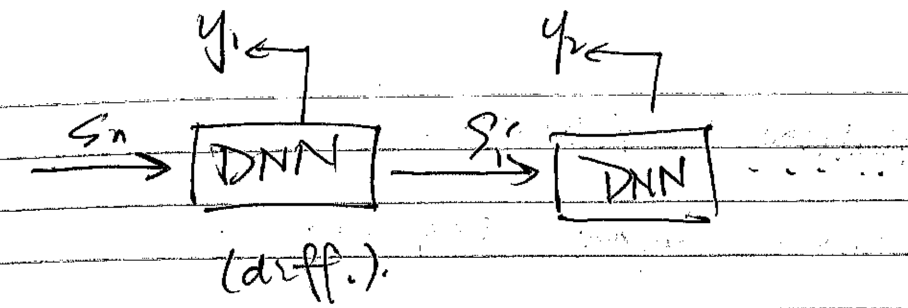
- Transformers. ≈ parallel.

DFA: $S_{next} = T(S_{current}, x_i)$. $\times \mathbb{R}^c$

RNN. $T: \mathbb{R}^d \times \mathbb{R}^{d_s} \mapsto \mathbb{R}^{d_s}$



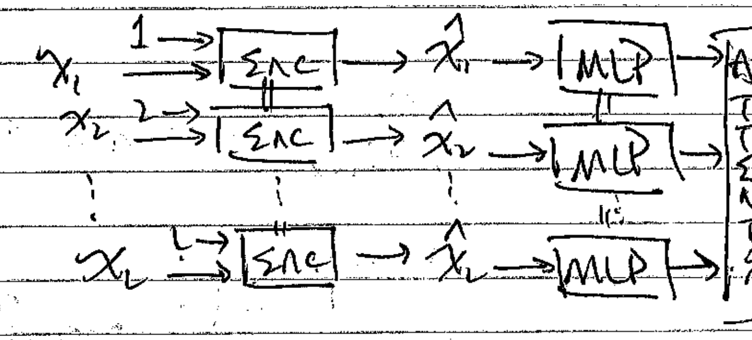
y_2 seq2seq task $y \leftarrow$ DNN



- LSTM.

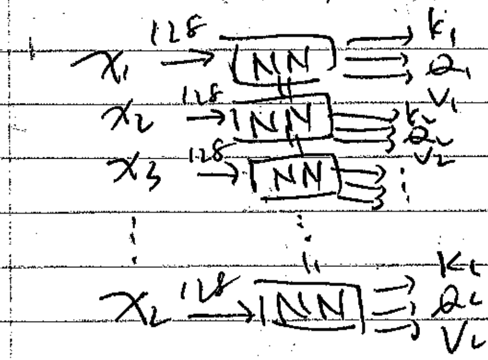
- Transformers.

input sequence



$\hat{x}_i = \text{Enc}(x_i, i)$.

Mathematically attention layer:



$k_i \in \mathbb{R}^{d_k}$

$V_i \in \mathbb{R}^{d_v}$.

$Q_i \in \mathbb{R}^{d_k}$.

$Q, k \in \mathbb{R}^{n \times d_k}$.

$V \in \mathbb{R}^{n \times d_v}$.

$\text{Softmax}\left(\frac{Qk^T}{\sqrt{d_k}}\right) \cdot V$

approximate table lookup: Attention block.

Multihead Attention

Same input sequence

↓

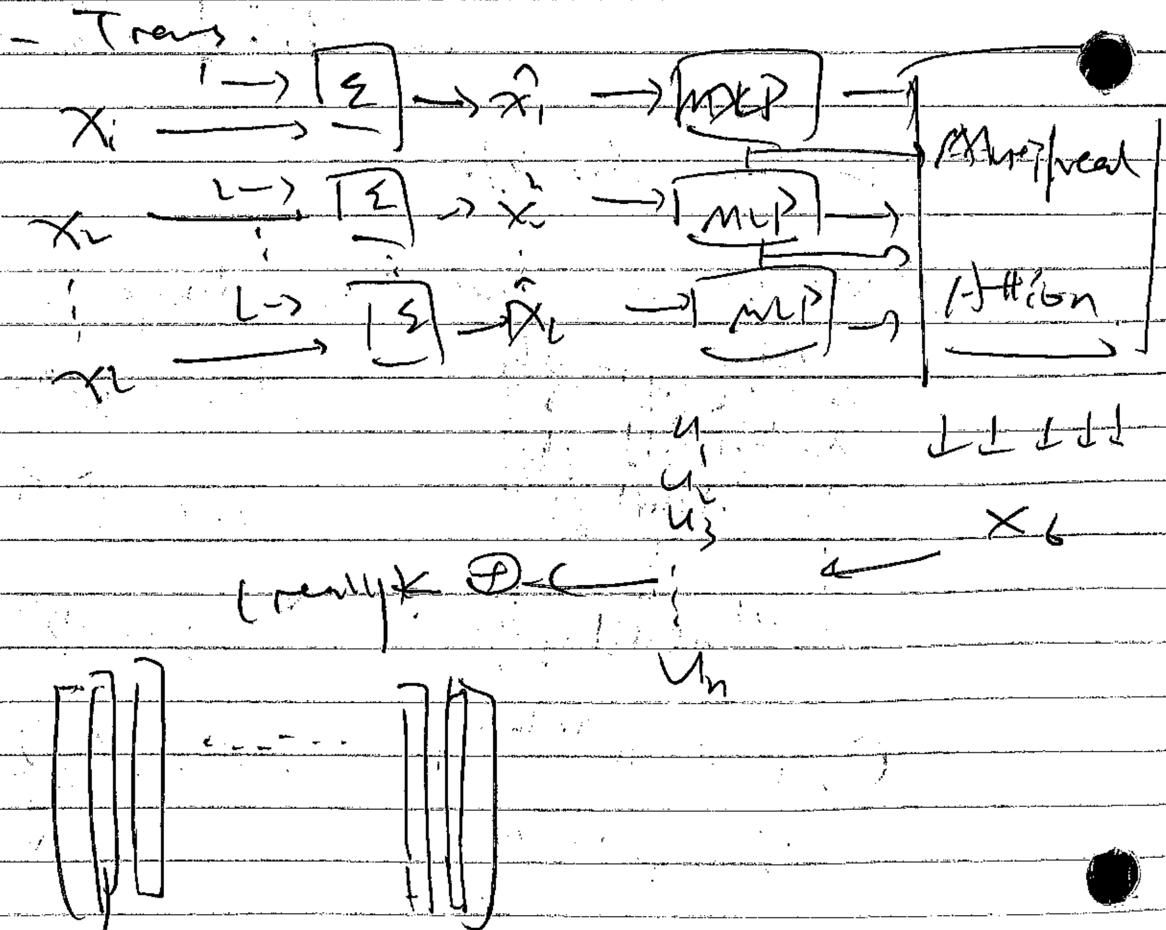
Same no. attention blocks

↓

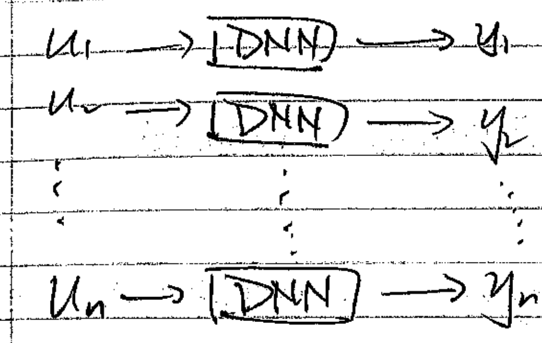
diff weights → concatenate output

$\hat{X}_{i,1:8} = Enc(x_i, i)$

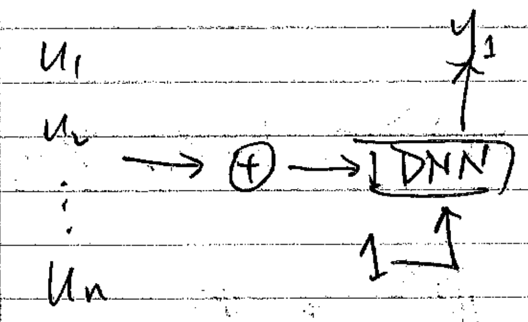
- Trans.



Seq.



instead, we do



Lecture 20.

Kernels.

opposite of dimension reduction.

map into higher dimensional space.

$$\phi(x) = \mathbb{R}^d \mapsto \mathbb{V}$$

$$\langle \phi(x_1), \phi(x_2) \rangle = k(x_1, x_2).$$

$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

Examples.

- Gaussian Kernels

$$k(x_1, x_2) = \exp(-\gamma \| x_1 - x_2 \|^2).$$

a.k.a. RBF kernel

- linear kernel

$$k(x_1, x_2) = x_1^T x_2.$$

- exponential kernel (Laplacian)

$$k(x_1, x_2) = \exp(-\gamma \| x_1 - x_2 \|).$$

- Polynomial kernels

$$k(x_1, x_2) = (1 + x_1^T x_2)^P.$$

Kernel definition.

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}.$$

$$k(u, v) = k(v, u)$$

→ kernel must be symmetric.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0.$$

$\parallel$

if $K_{ij} = k(x_i, x_j) \Rightarrow K$ is positive semidefinite

k is a kernel implies:

$\exists$ $\mathbb{V}$ vector space & ϕ s.t.

$$\langle \phi(x_1), \phi(x_2) \rangle = k(x_1, x_2)$$

if k_1 & k_2 are kernels w/ feature maps ϕ_1 & ϕ_2, then $k = k_1 + k_2$

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \end{bmatrix}$$

$k(u, v) = k_1(u, v) + k_2(u, v)$ ← is a kernel.

$k(u,v) = c\,k_1(u,v)$ if $c>0$.

$\phi(x) = \sqrt{c}\,\phi_1(x)$.

$k(u,v) = k_1(u,v)\,k_2(u,v)$.

$\phi(x) = \phi_1(x) \otimes \phi_2(x)$.

$k(u,v) = f_0(u)\,k_1(u,v)\,f_0(v)$.

$\phi(x) = f(x)\,\phi(x)$.

$f: \mathcal{X} \to \mathbb{R}$.

component loss $\in$ $f(w, x, y) = \ell(w^T\phi(x), y)$.

$$= \ell(\langle w, \phi(x)\rangle, y).$$

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f(w, x_i, y_i).$$

$$= \frac{1}{n}\sum_{i=1} \ell(\langle w, \phi(x_i)\rangle, y_i).$$

$$\nabla_w f(w, x, y) = \ell'(w^T\phi(x), y)\,\phi(x).$$

$\Rightarrow$ for GD / SGD.

$w_k \in \text{span}\{\phi(x_1), \phi(x_2), \dots \phi(x_n)\}$

$w_k = \sum_{i=1}^{n} \alpha_i \phi(x_i)$

$w_k = \sum_{i=1}^{n} a_i \phi(x_i)$.

$$F(a, x, y) = \ell\left(\left\langle \sum_{i=1}^{n} a_i \phi(x_i), \phi(x)\right\rangle, y\right).$$

$$= \ell\left(\sum_{i=1}^{n} a_i \langle \phi(x_i), \phi(x)\rangle, y\right)$$

$$= \ell\left(\sum_{i=1}^{n} a_i k(x_i, x), y\right)$$

$$f(a) = \frac{1}{n}\sum_{j=1}^{n} \ell\left(\sum_{i=1}^{n} a_i k(x_i, x_j), y\right).$$

let $K_{ij} = k(x_i, x_j)$

$$= \frac{1}{n}\sum_{j=1}^{n} \ell((Ka)_j, y).$$

① compute $\phi(x_i)$ on-the-fly as needed.

② precompute & store $\phi(x_i)$ before training. train in the transformed space

③ compute K on-the-fly as needed

④ precompute K & store.

Lecture 21.

<u>Kernels</u> function $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

ways to do learning:

1. Learn in the transformed space, compute ϕ on-the-fly.

2. learn " ", precompute ϕ.

3. learn in the "kernel" space while computing k on-the-fly.

4. learn in "kernel trick" space. precompute $K_{ij} = k(x_i, x_j)$

$f_i(w) = \ell(w^T \phi(x_i), y_i)$

$\nabla f_i(w) = \ell'(w^T \phi(x_i), y) \phi(x_i)$

suppose n examples, $x_i \in \mathbb{R}^d$, and k takes $\Theta(d)$ to compute

$\phi(x_i) \in \mathbb{R}^m$,

run for T steps.

cost to compute $\phi \Rightarrow \Theta(md)$.

- Vanilla non-kernel linear model: $\mathbb{R}^d$.

	Compute	Memory
1	$\Theta(Td)$	$\Theta(d)$ (not inc. train.s)
2		
3		
4		

- learn in the transformed space

1	$\Theta(Tmd)$. precompute	$\Theta(m)$.	"tnd" (training d..)
2	$\Theta(Tm + nmd)$.	$\Theta(nm)$,	
3	$\Theta(Tnd)$.	$\Theta(n)$.	
4	$\Theta(Tn + n^2 d)$	$\Theta(n^2)$.	

$\ell'(\sum_{j=1}^n u_j k(x_i, x_j), y_i) \phi(x_i)$

$\nabla f_i(w) = \ell'(\phi(x_i)^T \sum_{j=1}^n u_j \phi(x_j), y_i) \phi(x_i)$

an SGD step $\Rightarrow$ compute i

$$u_i \leftarrow u_i - \alpha\, l'\Big(\sum_{j=1}^{n} u_j k(x_i, x_j); y\Big)$$

For kernels - subsampling.

$$k(x_i, x_j) \approx \langle \psi(x_i), \psi(x_j) \rangle.$$

$$\psi(x) \in \mathbb{R}^D.$$

Approximate feature map.

- Random Fourier features.

if kernel : $k(x_i, x_j) = k(x_i - x_j)$.

then $k(x_i, x_j) = \mathbb{E}\big[2\cos(w^T x_i + b),$

$\cos(w^T x_j + b)\big]$.

$b \sim \text{Unif}\,[0, 2\pi]$

$w \sim \mathcal{F}(k)$ Fourier transform of k.

"RBF" k

RBF kernel:

$$w \sim \mathcal{N}(0, 2\gamma I),$$

$$b \sim \text{Uniform}([0, 2\pi]).$$

$$\mathbb{E}\big[2\cos(w^T x_i + b) \cdot \cos(w^T x_j + b)\big]$$

$$= \exp(-\gamma \|x_1 - x_2\|^2).$$

draw $w_1, w_2, \dots, w_D \sim \mathcal{N}(0, 2\gamma I)$.

$b_1, b_2, \dots, b_D \sim \text{Unif}([0, 2\pi])$

Set $\psi_i(x) = \cancel{\tfrac{1}{\sqrt{D}}} \sqrt{\tfrac{2}{D}} \cos(w_i^T x + b_i)$

$$\Rightarrow \mathbb{E}\big[\langle \psi(x_i), \psi(x_j) \rangle\big] = \exp(-\gamma \|x_i - x_j\|^2)$$

⋮ (TBC)

⑤ Learn w/ approx. feature map comp.

ψ on-the-fly.

⑥. Learn w/ approx. f. m. compute & cache

Vanilla linear model.

compute. Memory.

(f) $\Theta(TDd)$. $\Theta(D)$)

(b) $\Theta(TD + nDd)$ $\Theta(nD)$).

large D - high "accuracy" in repr. K.

- high compute & memory usage.

Lecture 22.

▷ Hyperparameter optimization.

- Step size / lr / α → diverge.

$\alpha \Uparrow$ — too large noise ball.

get "stuck".

$\alpha \Downarrow$. — nonconvex.

- λ for regularization

↓

over/under regularize.

converge too slowly. ← wrong level of expressivity.

poor generalization.

- β momentum. → if $\beta \notin (0, 1)$.

$\Downarrow$

bad.

- B batch size → too small can

affect noise ball.

too large → much like GD.

performance matching parallel capabilities of HW.

- choice of architecture : DNN

↓

generalize poorly.

systems implications.

- - - - - - - - - - - - - - Non-DNN - -

- tree depth → weak learner

overfit & lots of mem.

- feature vector length for Random features. → poor fidelity,

high compute/memory cost.

- # of epochs → high loss/error

compute cost.

- dimension → poor accuracy.

high compute cost

- k in KNN & k-means → under/overfit

more compute for longer k.

- the task for assigning hp:

→ HPO - Any process assign hps.

Standard: β 0.9 0.99

(β_1, β_2) (0.9, 0.999)

B power of 2, e.g. 256.

- Grid Search

↳ CoD

- Random Search

less reproducible than GS.

Parallelism

Early stopping

- droping bad hyperparameters that perform poor each epoch goal: minimize $F(\alpha, \beta, B, k, \lambda)$.

"derivative-free" optimization (DFO)

Encode our knowledge abt the space

& the problem $\Longrightarrow$ HPO

Bayesian Opt — continued.

Mean / Median $f_0,\ f_1,\ f_3,\ f_4,\ \ldots$

HPO

$P(f \mid f(x_1) = y_1,\ f(x_2) = y_2)$.

look at: $P(f(x_*) \mid f(x_1) = y_1,\ f(x_2) = y_2, \ldots)$

$f \in \mathbb{R}^d,\quad f \sim \mathcal{N}(0, \Sigma)$

mean covariance

* Fact: if (X, Y) are jointly $\wedge$ Gaussian.

multivariate

{ Set by ~~the~~ according to intuition.

the $X \mid Y = y$ is also Gaussian.

* Fact: " $\sim$ ", then X is Gaussian.

$f(x_*) \mid f(x_1) = y_1,\ f(x_2) = y_2, \ldots$

$\sim \mathcal{N}(\mu, \sigma^2)$

How to pick the next point?

$$x_{next} = \arg\min_{x^*} a(\mu(x^*), \sigma^2(x^*))$$

$$a(\mu, \sigma) = \mu - K\sigma.$$

(lower confidence bound)

a = acquisition function.

$$a(\mu, \sigma) = P(f(x^*) \le f_{best}).$$

$\hookrightarrow P \sim \ldots$

(Probability of Improvement).

$$f(x^*) \sim \mathcal{N}(\mu, \sigma^2)$$

$$= \Phi\left(\frac{f_{best} - \mu}{\sigma}\right)$$

accumulated distribution function.

$$\Phi(u) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{u} e^{-\frac{z^2}{2}} dz$$

$$P(f(x^*) \le f_{best} \mid obs) = P(Z \le f_{best})$$

for $Z \sim \mathcal{N}(\mu, \sigma^2)$

$$= P(\sigma u + \mu \le f_{best})$$

for $u \sim \mathcal{N}(0, 1)$

$$= P\left(u \le \frac{f_{best} - \mu}{\sigma}\right) = \int_{-\infty}^{(f_{best}-\mu)/\sigma} P(u)\, du$$

$$a(\mu, \sigma) = \mathbb{E}[\underbrace{f(x_*) - f_{best}}_{\min(f(x_*),\, f_{best})} \mid obs)$$

Expected improvement.

by convention:

$$= \mathbb{E}[\min(f(x_*) - f_{best}, 0) \mid obs]$$