Vanishing Gradient Problem

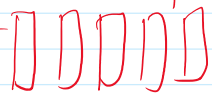
Thursday, April 7, 2022 7:45 AM

In machine learning, the **vanishing gradient problem** is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training. ~~As one example of the problem cause, traditional activation~~

GD

1) $0.1 \times 0.1 \times 0.1 \times 0.1 = \boxed{0.0001}$ → VGP

2) Deep NN



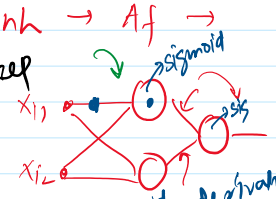
3) Sigmoid / tanh → Af →

$\boxed{0-1}$

Backprop

$\boxed{0-0.5}$

deep



$\hat{y} - y = L$

$L \to \hat{y} \to o_{11} \to w'_{11}$

$$\boxed{\frac{\partial L}{\partial w'_{11}}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w'_{11}}$$

sigmoid derivative

$0 < x < 1$ $\quad 0 < x < 1$ $\quad 0.1$

0.0001

$$w_n = w_0 - \eta \boxed{\frac{\partial L}{\partial w}}$$

→ derivative of L wrt weight

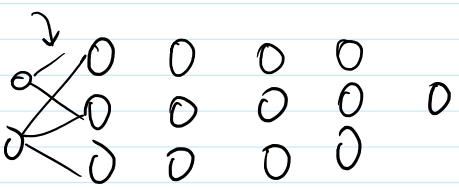
$w_0 = 1$ ↑ small change

$w_n = 1 - 0.01 \times 0.0001$

$w_n = 0.99999$ → vanishingly small

did not change

{ sigmoid, tanh }



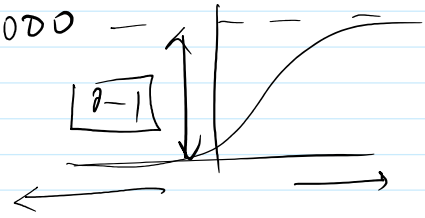
$\frac{\partial L}{\partial w'_{11}} = 0.000001$

Backprop → model training → x

80's, 90's → main reasons DL failed

$0 - 100000$

$\boxed{0-1}$



How to recognize?

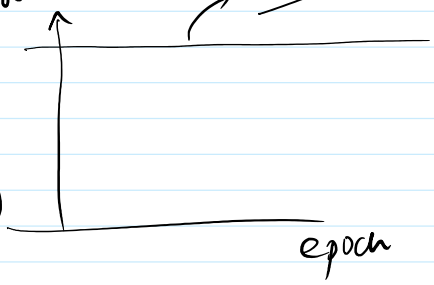
1) Loss focus → epoch → no changes → VGP

2) weights → graph value → Tensorboard → callbacks

w'_{11}

Keras

{ loss after epoch }



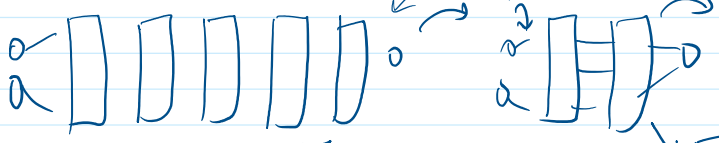
$$w_n = w_0 - \eta \boxed{\frac{\partial L}{\partial w}}$$

$$\boxed{\frac{w_0 - w_n}{\eta}} =$$

$$w_n = w_0 - \eta \boxed{\frac{\partial L}{\partial w}} \qquad \boxed{\frac{w_0 - w_n}{\eta}} =$$

How to handle Vanishing Gradient Problem →

1) Reduce model complexity



$\boxed{\frac{\partial L}{\partial w}}$ = bigger

$\boxed{d_1 d_2 d_3}$

$d_1 \times d_2 \times d_3 \times d_4 \times \ldots$

Applicable
↳ find out complex patterns

2) Using $\boxed{\text{ReLU}}$ Activation functions

max (0, z)

→ $\boxed{0}, \boxed{1}$

$= 1 \times 1 \times 1 = 1$ → VGP

{ −1000 − 1000 squish
$\boxed{0}$ − 1000
$\boxed{0-1}$ }

dying relu → derivative
$w_n = w_0$

{leaky relu}

3) Proper weight init → Global, → xavier

4) Batch norm → layer →

5) Residual Network → CNN → ResNET
↳ building block → ANN

Exploding Gradient Problem → $\boxed{\text{RNN}}$

big number

10, 10, 10, 10 → 10000

$\frac{\partial L}{\partial w_{11}^1} = d_1 \times d_1 \times d_2 \times d_3$

$w_n = w_0 - \eta \frac{\partial L}{\partial w}$ (0.1, 1000)

random

Loss↓ ✗ 1 − 100 → 99 → 1000 →

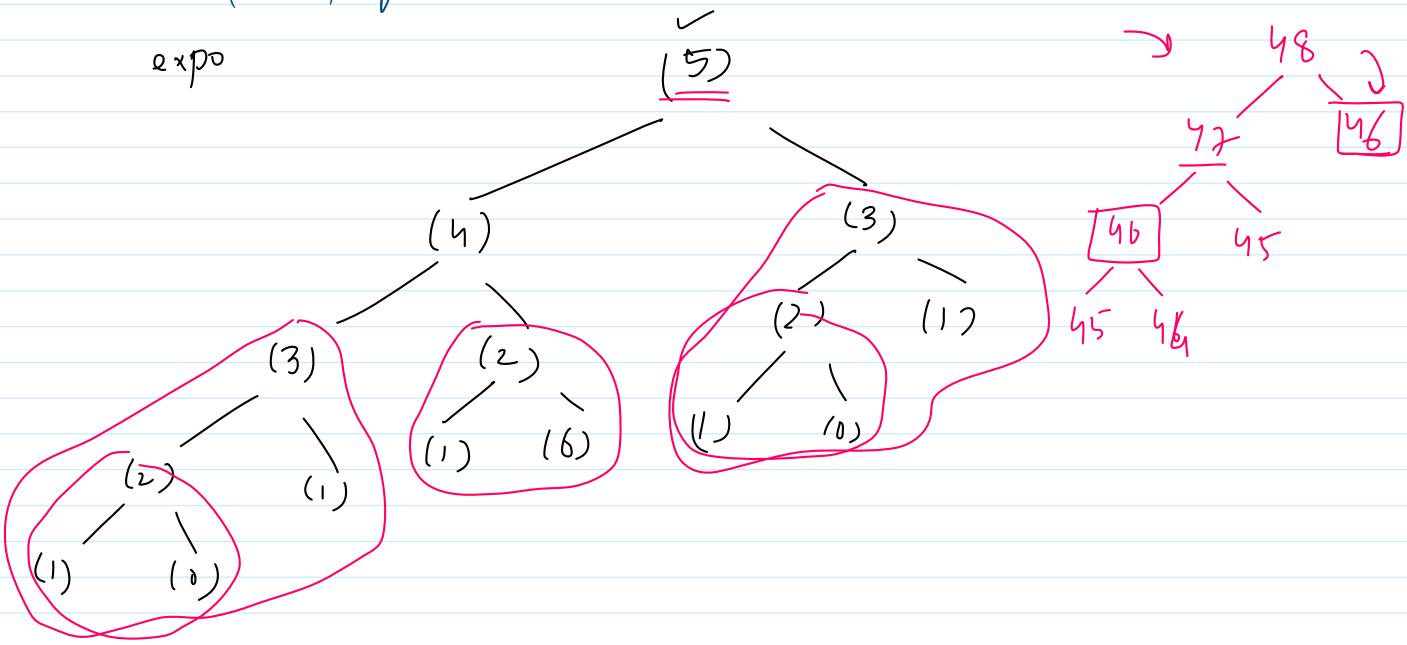
{Gradient clipping}

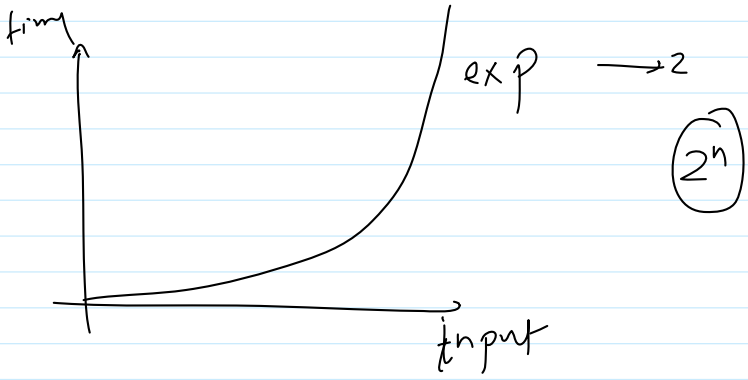
expo

(5)

48

expo





Saturday, April 9, 2022 10:01 AM

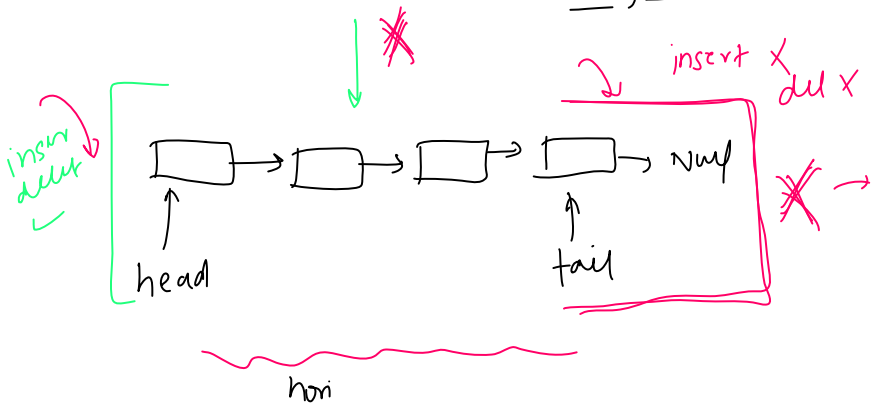
DS Algo → Time complex
→ Arrays → Dynamic Arrays
→ Linked list

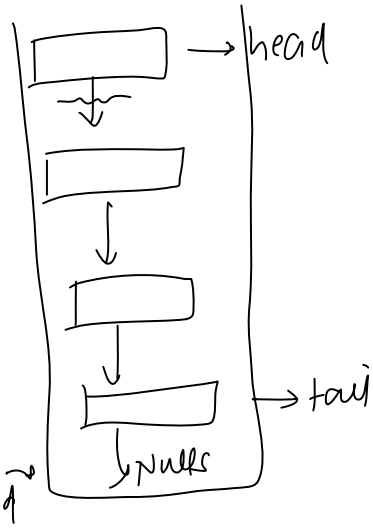
Today → Stacks → LL
→ Arrays

→ 3 → Tuesday
↓
last 3 videos

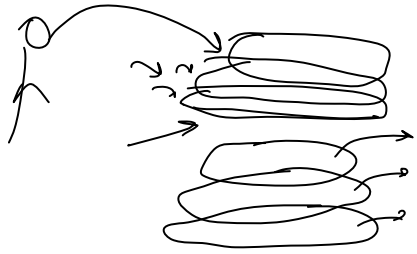
Next → Queues
→ Practice →

→ Stacks → DS
↳ useful





LL head



Stack LIFO

LIFO → Stack

last in first out

ADT → conceptual thing
↓
Normal data type

push → adding new node from top
pop → removing from top
peek → top node → value
size → kitne node

is_empty → empty

LIFO
→ Expression matching
→ Undo redo
→ browser back button

s = Stack()

Arrays

Linked List

s = Stack()

empty stack

top = head

data

address

head

new

None

top

None

3
2
1

None

head

head

head = head.next

top = None

None → Next

peek (5)

top = None

LIFO

Advantage of Stack

Write

Insert → O(?) → O(1)

push

LL →

Arrays

Static → int a[50]

Dynamic

1) push

2) delete → O(2) → O(1)

int a[50]

size = 4

top = -1 → 0 → 1 → 2 → 3

1, 2, 3, 4 (5)

-1 = 4-1 ×

0 = 4-1

1 = 4-1

2 = 4-1

3 = 4-1

overf-

1	2	3	4
0	1	2	3

top

0 → top+1

1	2	3

top top

123

321

1	2	3	4

top

1	2	3

top

1	2

top

top = -1

1	2	3

-1 top

pop → 3

pop → 2

pop → 1

pop → 1

stack[-1]

x, y : x+y

1 2 3 4 5

x y

2 3 4 5

x y

9 5

x y

14

5 4 5

x + y

try

error

No error

Online

Time

Arrays → Dynamic

DSA

Linked List

Stacks

3 RL

→ M/T → Queues → practice

Stacks

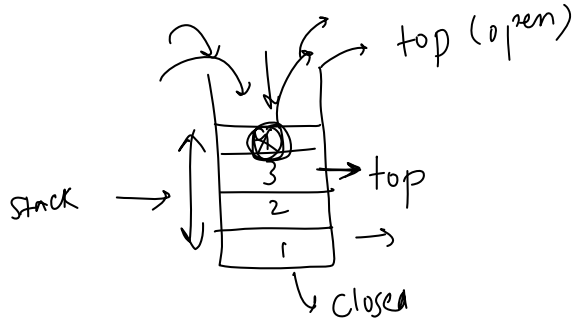
Data structure (linear)

LIFO

LIFO → Real world

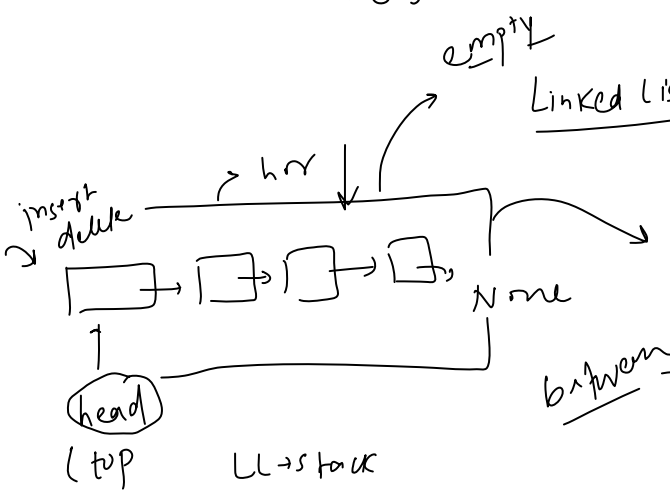
Linked list

Arrays



push
pop
peek
is empty
size

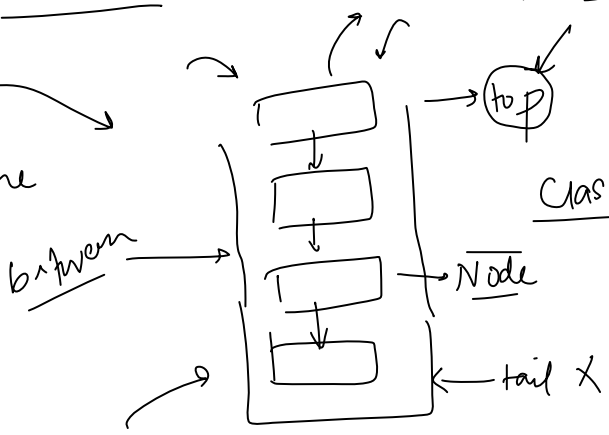
Simple



head = None

stack

top = None



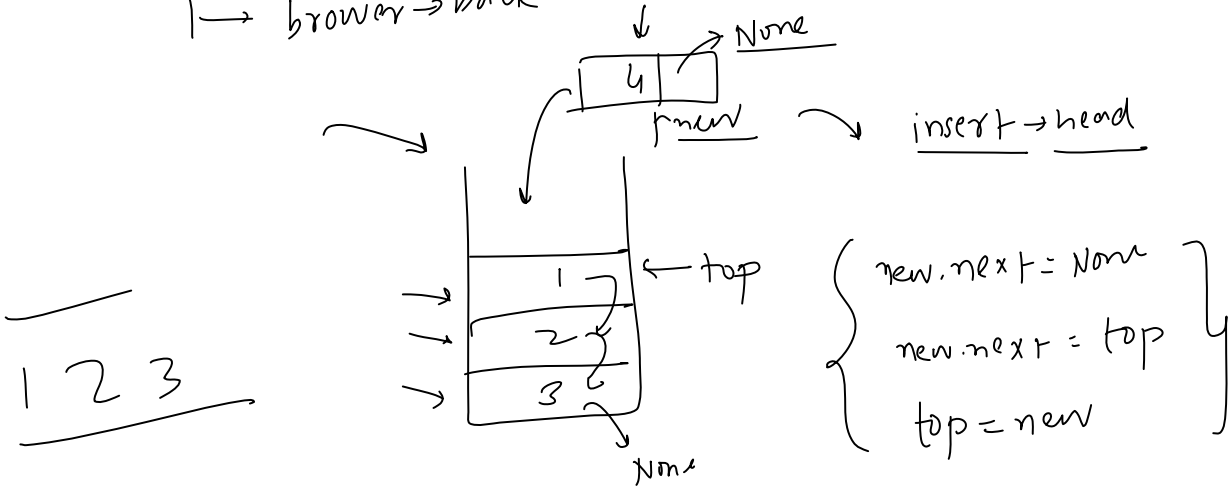
Class → LL

LL

Application

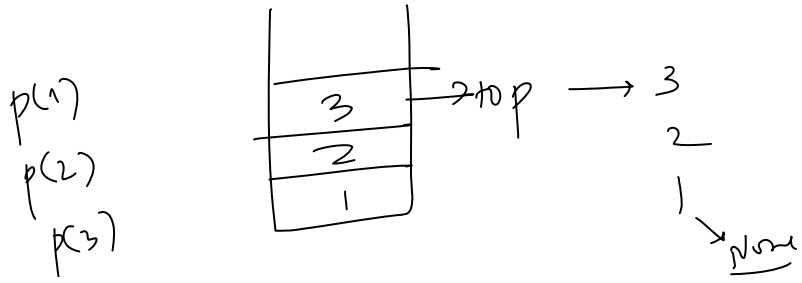
- → expression
- → undo/redo
- → recursion →
- → browser → back

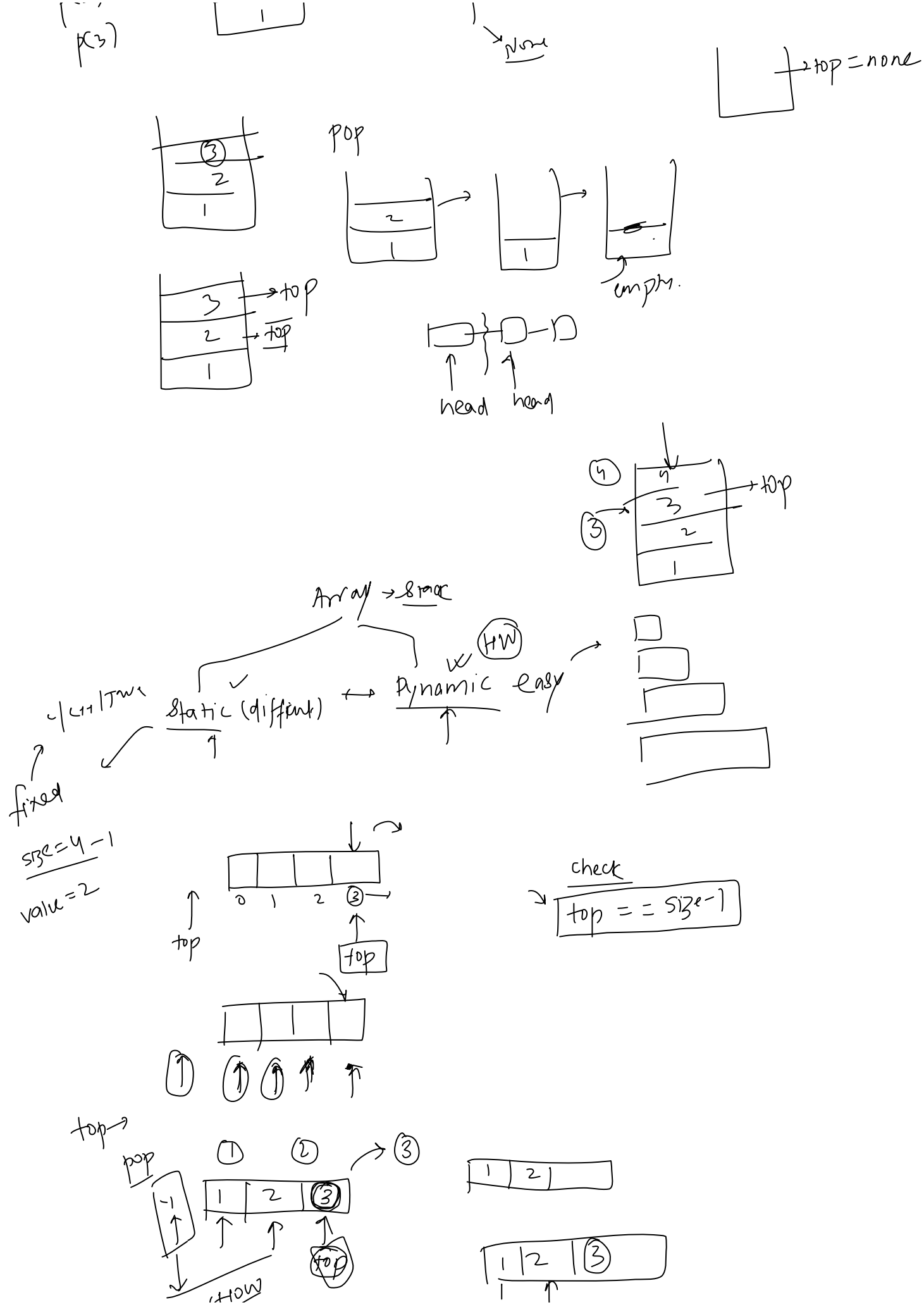
LIFO

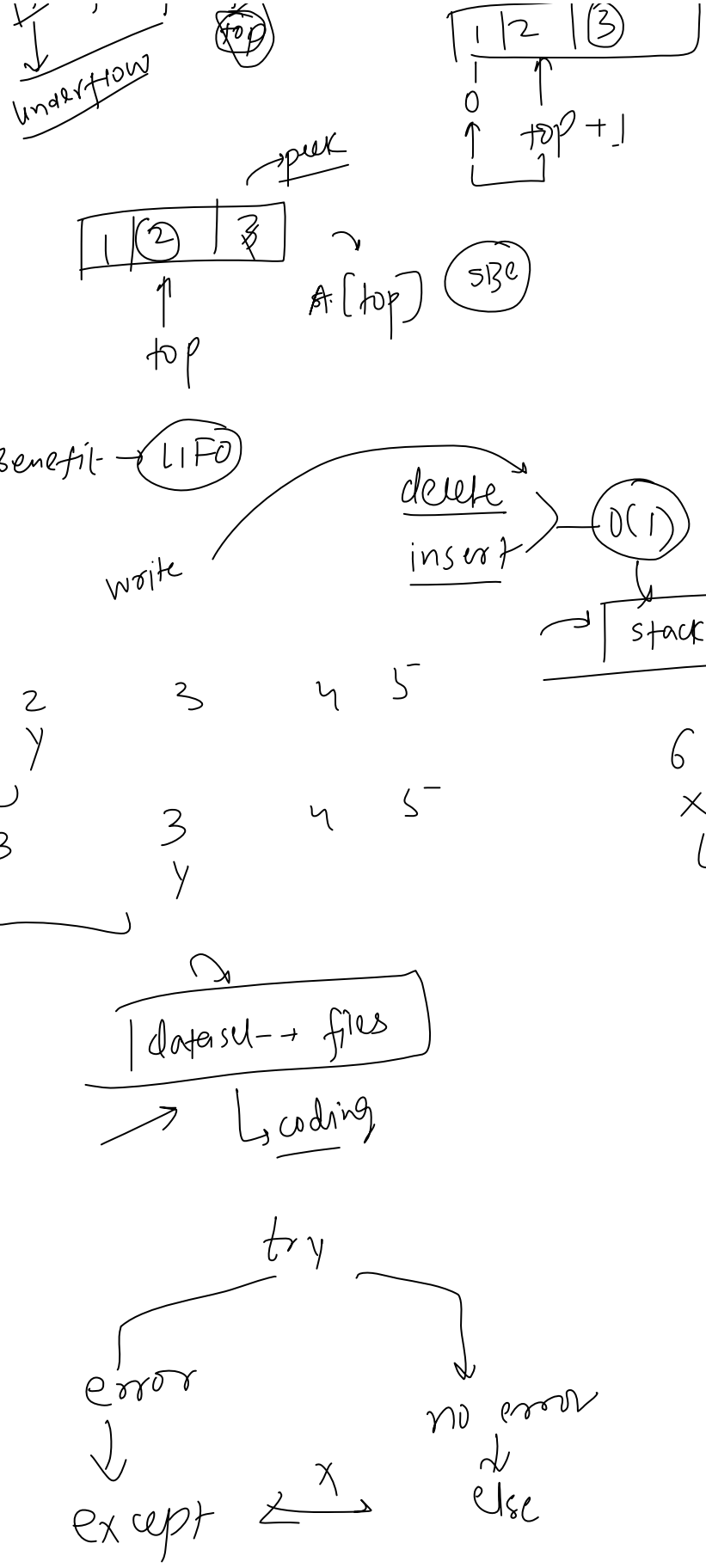


new.next = None
new.next = top
top = new

1 2 3




p(3)
None
top = none
pop
empty.
top
top
head
head
top
Array → Stack
HW
Static (difficult)
Dynamic easy
fixed
size = 4 - 1
value = 2
top
top
check
top = = size-1
top →
pop
-1
top
HOW


top
Underflow
1 2 3
0
top + 1
peek
1 2 3
top
A[top]
size
Benefit → LIFO
write
delete
insert
O(1)
stack size
1 2 3 4 5
x y
3 3 4 5
x y
6 4 5
x y
10 15 5
x y
dataset → files
↳ coding
try
error
no error
except
else
x