finance

- quants : portfolio calculations
- trading: hold,sell,buy
(HFT)

hedge funds
A 120/100x * 5
B 85/100y *5

long short : 120/100 x* 5* 12 long
 85/100y * 5* 3 short
- BFSI - IBs (they account for market inflation, risks and exchange defaults), hedge funds (risky)

weight
- Timeseries (statistical /math modelling)
- * Model this trading (RL environment)

IPO s - going public - listed in stock exchange
demand of the shares

RL : (What will we see today?? Train a DQN for trading on S&P ??)

Env : can be anything (States)

Agent : is ass. following actions and remembering (DL)

Rewards : functions

Actions: based on a set of environmental conditions bot will take actions

reward +=100
stop episode

- crash

if car.pos <u_thresh and car.pos > lower_thresh:
reward -=50

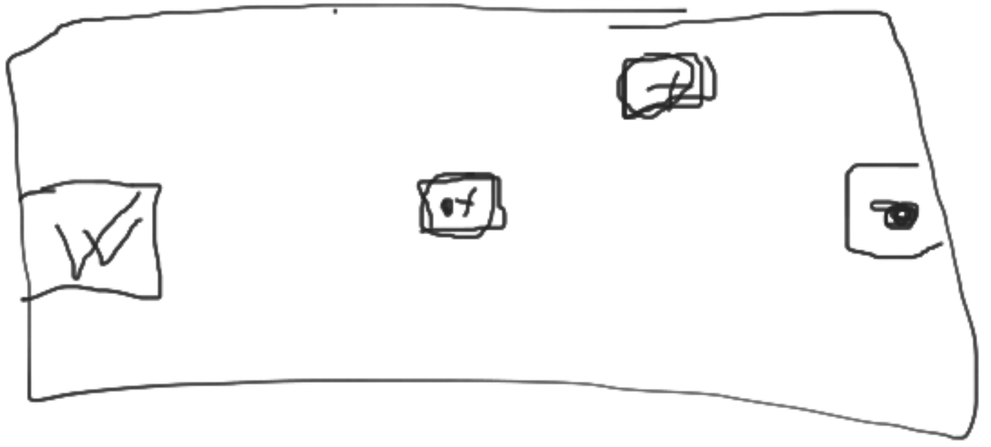
- go in opposite direction

RL is simulation of a robot to maximise rewards

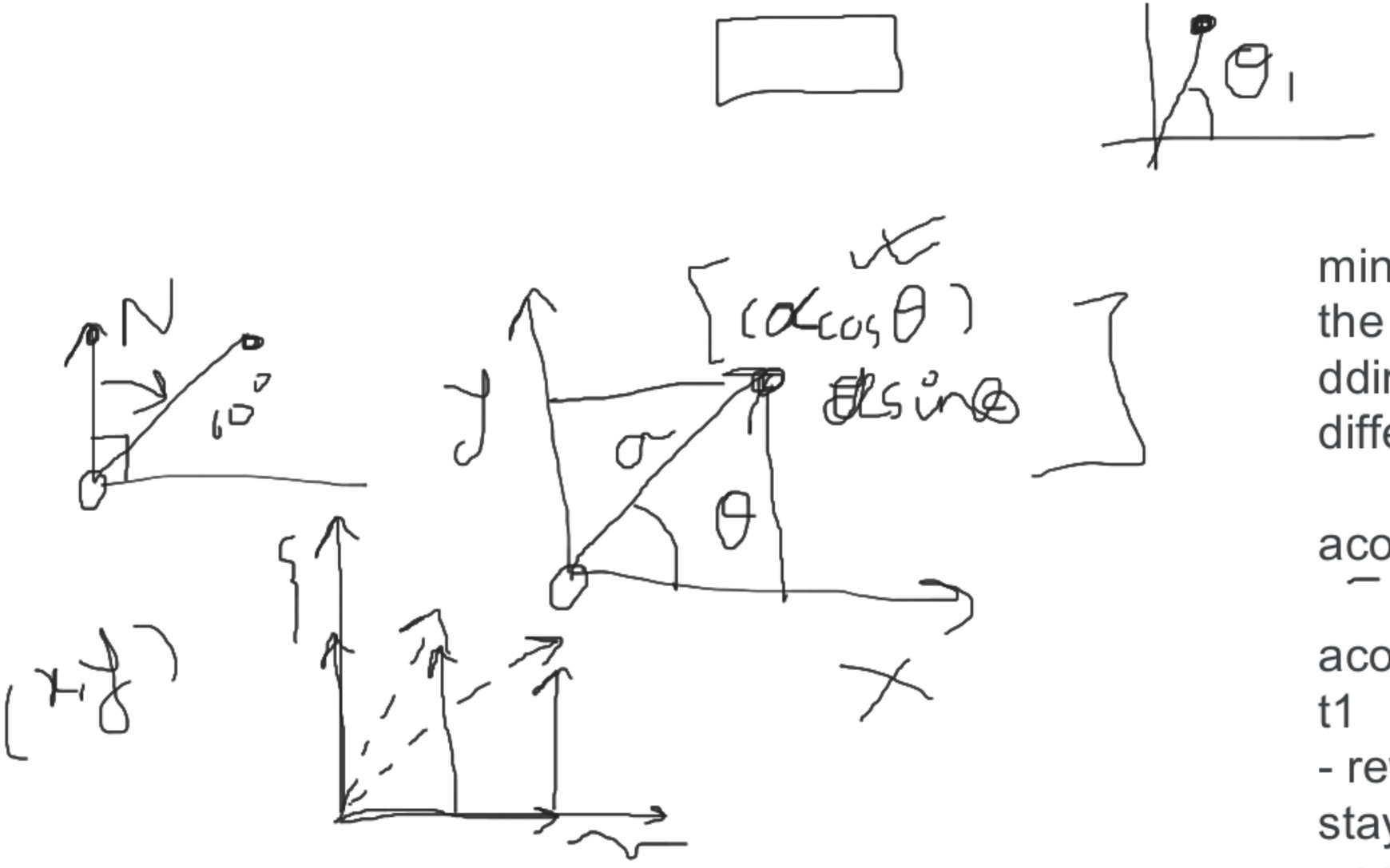
Camera mounted on a bot (Agent)
playground lot of obstacles (posts,signals) (Env)

Reward->??? Action
reward functions -> actions

pseudocode
if car.x >= location.x && car.y >=location.y



```
//reach the end
if car.pos < u_thres.pos && car.pos>l_thresh.pos
   reward +=100
   stop episode

- crash

if car.pos <u_thresh_obs.pos  && car.pos > l_thresh_obs.pos:
    reward -=50

- do not go out of field
if car.pos >u_thresh_field.pos  && car.pos < l_thresh_field.pos:
    reward +=25


- go in opposite direction

distance measure

if(car.pos- u_thresh.pos)<thresh
   reward+=40
else:
   reward-=20
 - give a small reward for moving car.pos in direction of thresh_final.pos
```



minimise the ddirectional difference

acost,asint

acost1,asint1

- reward stay
- penalty make your direction correct!!

acost-acost1,asint-asint1

=a(cost-cost1)

```
if car.post.x - car.post1.x
<thresh_radians:
  reward+=15
else:
 reward-=15
```

(car.post.x = cost(tan-1(pos.y/pos.x)))

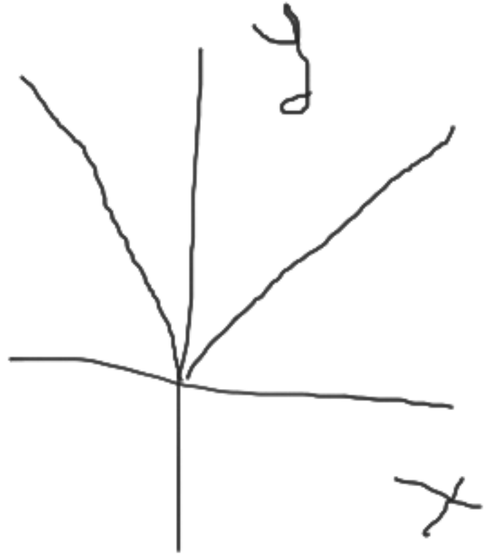
DRL:
(Model free RL)
- On policy
- Off policy
(Model Based RL) ~~~
2 functions:
Q V

Either you maximise the valu function V - off policy (be quick)
or you maximise the policy Q to maximse the V - on policy (actor critics)

discrete or continuous - off policy in cont spaces is inefficient
Q Learning (Sutton & Bellman equations),SARSA
Temporal Differece Learning

Off policy - DQN

Also note: In RL we take expectations of rewards rather than only rewards
E[f(x)] not f(x)



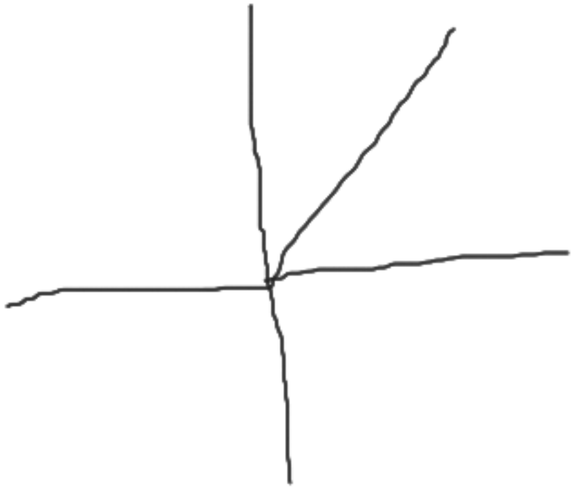
$f(x) = |x|$

3 signals - B,H,S

Agent: Sequential Dense prediction of 3 softmax logits
memory: a container for previous observations
cache memory: 30 % of previous observations recent (buffer)
actions: predict porbabilities
learn: update agents memory ,states
(update your capital)

f(x)=x for all x>0
f(x)=0 otherwise

relu



Can we do derivatives??????

Memory: states, actions, past rewards, next steps , met my goal

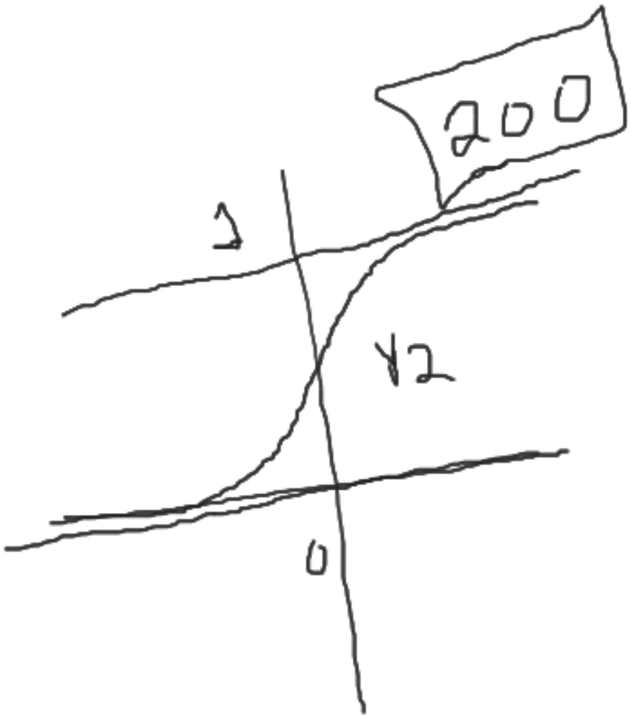
For off policy there is always an experience replay ????
Bcs in DQN we need to Agent to perform everything

states -> actions (3) -> rewards (+/-R) -> next_steps(3) ->done(T/F)

DQN is very optimistic
does not account for margins of errors

Agent will have 2 brains

updating the weights from second brain to the first brain -> ??? margin of error found by the second brain is now updated in the first brain . Now the first brain knows the errors which might have come had he bought at 7 and sold at 10
brain->model() DQNs
second brain->target model() DQNs
DDQN -> they are not competing against each other



Environment -? Portfolio
Capital
Stock (1)
Quantity
previous stock prices (upto certain timesteps)
-> dates
previous rewards

Operations:
env update operations
reset_portfolio ?????? -> on completion of episode (agent reaches the target)

generate price
related to reward differences between the Agent and the env/portfolio ??
NN -> bias , forgetting

lim n**x/x**n
x->inf

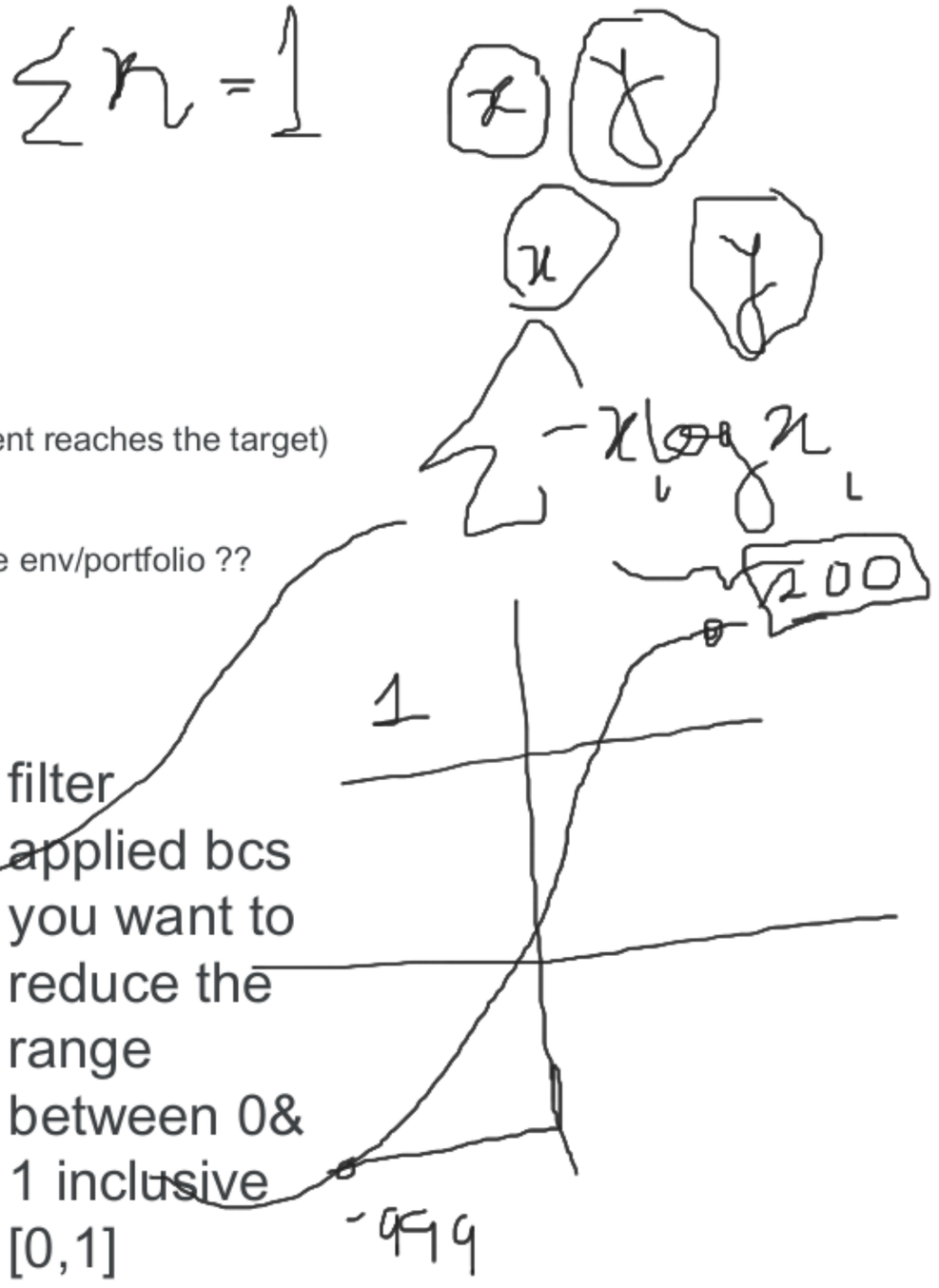
E[f(x)] = E[x(t)+ gamma* (x(t-1))*scalar values]
policy gradients -> starting point of ~~on policy~~

sigmoid: ???
1/(1+e**(-x))=f(x)=0
lim ~~x this will happen~~

e**(x)/
sum(e**(xi)
) =softmax

1=1+e**(-x)
e**(-x)=0

filter
applied bcs
you want to
reduce the
range
between 0&
1 inclusive
[0,1]



Env :
Train script

pseudocode:

Agent should be able to buy and sell
agent should be able to predict (agent.act())
agent should be able to update the weights
agent.experience _replay()
agent needs to update its history/memory
agent.remember()

Env:
stock[buydate] ,stock[enddate]
based on the rewards Agent is getting
env.reset()
env.generate_()
inherit parameters from portfolio -> scalars like balance etc

- there should be a training loop iterating upto the number of episodes provided

```
for i in training_iter:
  do
      agent.reset()
      states=env.generate_()
      while(j<trading_days):
          do
              set reward to 0
              state=env.generate_()
               get prev_portfolio_value()
               Agent.forward() -> agent.act(state)
               agent.buy() | agent.sell() | agent.hold()
                if agent.missed_opportunity():
                     agent.penalize()


               agent.get_rewards()
               if agent.rewards()==threshold:
                   j++
```

#other functions for plotting.tb what not

sum of any three non recurring
real numbers where all are equal !=1 [0,1]
(True)
3x=1
x=1/3 R
1/3 =
0.33333333333333333333333333333333333333
33333333333333333333333333333333333333
33333333333333333333333333333333333333
333333

at any time atleast one value > greater than other 2

- there should be a training loop iterating upto the number of episodes provided

```
for i in training_iter:
   do
       agent.reset()
       states=env.generate_()
       for j in trading_days:
           do
              set reward to 0
              state=env.generate_()
               get prev_portfolio_value()
               Agent.forward() -> agent.act(state)
               agent.buy() | agent.sell() | agent.hold()
                if agent.missed_opportunity():
                     agent.penalize()


               agent.get_rewards()
```

Training script as it is ...

When 2 brains are competing against each other

Q=v+A
uptil D2QNs , Q=v
advantage A

A=Q-v
Dueling DDQN

for some simulations D3QN > D2QN (more rewards)
Never ending dueling creates stagnation

D3QN
Q=V+ max(A- 1/|A|<A>)
A-> policy based methods (on policy)

On policy methods

policy person(brains??) + Agent (brains ???)
A
Q=V+A

actor critics
policy gradients -> minimise the advantage loss
A=Q-V =
increase the E[A]
- gradient descent
- gradient ascent

Q up and V down
A up
policy guy -> his eestimated returns are quite higher than the Agents returns --T/F??
 (Agent becomes the Actor)

When you have -rewards

A -> negative
Q down -V up ??

(policy guy becomes Actor)

r(t) + gamma*(r(t-1)) = R(t) = discrete

Bellman eq: Q= V(t)+ R(t)
r(t) + gamma*(V(t-1)) = Q(t)updated

A(t)= Q(t)-V(t)
= r(t)+gamma*(v(t-1)) -v(t)
=E[()]

(optima curve)
max(A(t)) : ???
- d(v(t))/d(t)
- continuously differentiable within the same range
- curl should exist

J(t)= E[Q(t).pi(t)] ->gradient ascent

param update= s(t+1)=s(t)+ scalar* J(t)

Actor Critic Model
3 actions : B,H,S

Ingredients:
(Env and all are static)
- Actor Network (GD)
- Critic Network (GA)
- Synchronizing Module

pseudocode

- Actor pseudocode
 - def network():
    Dense()
    mse loss with adam
    model
- def weight_update():
   return gradients([action,states])

- Critic psuedocode
  - def network():
      return model
  - def weight_update():
     return gradients([actions,states]) (added up ?????)

- def weight_transfer():
    weight_update(actor)
    weight_update(critic)

Combined ACAgent()
- Actor class
- Critic Class

AcAgent.actor
Acagent.critic

- there should be a training loop iterating upto the number of episodes provided

```
for i in training_iter:
   do
       AcAgent.actor.reset()
       AcAgent.critic.reset()
       states=env.generate_()
       for j in trading_days:
           do
              set reward to 0
              state=env.generate_()


               get prev_portfolio_value()
               Agent.forward() -> agent.act(state)
               agent.buy() | agent.sell() | agent.hold()
                if agent.missed_opportunity():
                     agent.penalize()
                env.update()
                if agent.rewards()==threshold
                 do
                    agent.update()
                     end episode
```

ac -> you dont have memory?? no cache memory ?

off policy:
C51
Rainbow

on policy:
AC
A3c,TRPO,P
PO,SAC

Timeseries
Arima,arima
x,
DQNs to
solve
timeseries