Class & Learning Problems | Reinforcement Learning

Supervised Data: (x, y) Goal: Learn function to map $x \to y$
x is data, y is label
Apple example

Unsupervised: Data: x x is data, no labels
Goal: Learn underlying structure

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

- states are the observations of the system
- actions are the behaviors of the system: system takes or that agent takes when it sees those states

Eat this thing (apple)

Scanned with CamScanner

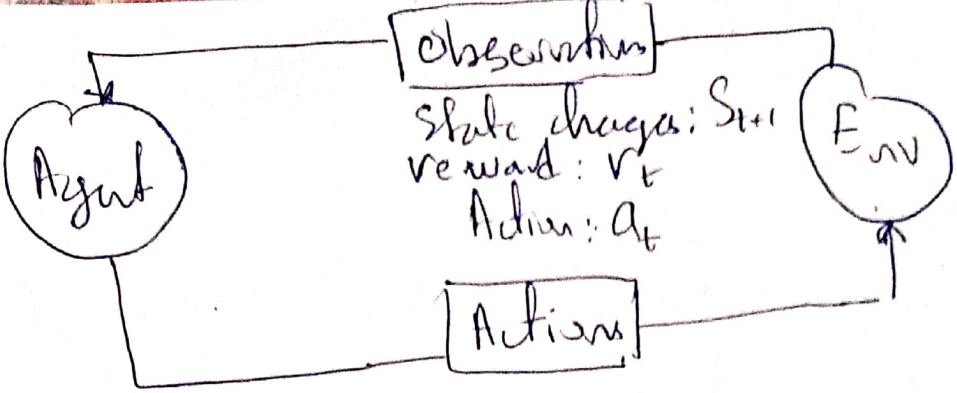
RL Concepts

Agent:. Take actions in the environment
e.g. drone delivery, navigation.
algorithm is agent
agent is each of you.

Environment:. The world in which
the agent lives., the agent
operates and conducts all these
actions.

Action:. a move the agent
can make in the environment
Agent can send command to the
environment in forms of actions

Scanned with CamScanner



Action Space A: the set of all possible actions

a_t: Action at time t

A: could have either discrete or continuous values. such as GPS coordinates

Observations: How the environment interacts back with the agent
That is how actions have affected its own state

State: the immediate situation in which the state finds itself
e.g. image feed.

Reward: a way of feedback from the environment to the agent

Scanned with CamScanner

environment can provide a way of feedback to measure the success or failure of an agent.

e.g. Mario touches coin, he wins points

from a given state an agent will send o/p in the form of actions to the environment

environment will respond with the new state as well as any rewards that may be given.

reward may be delayed or immediate

Total Reward (Return) $\longrightarrow R_t = \sum_{i=t}^{\infty} r_i$

$r_i \rightarrow$ reward at time t

R_t = reward from time t into the future.

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \cdots\cdots + r_{t+n} + \cdots\cdots$$

Scanned with CamScanner

rewards are collected by the agent

Discounted sum of rewards.

$$R_t = \sum_{i=1}^{\infty} \gamma^i r_i$$

discounting factor γ is multiplied by the future rewards that are discovered. in order to dampen those rewards effect on the agent's choice of action

this is to make future rewards less important than immediate rewards. enforces a short-term learning in the agent

$$R_t = \sum_{i=1}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \cdots\cdots + \gamma^{t+n} r_{t+n} + \cdots\cdots$$

γ: discount factor; $0 < \gamma < 1$

We would like to prioritize short-term rewards more

Scanned with CamScanner

The Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, R_t is the discounted sum of all rewards obtained from time t.

Q-function captures the expected total future reward an agent in state s can receive by executing a certain action a

$$Q(S_t, a_t) = E[R_t \mid S_t, a_t]$$

If an agent finds itself in some state in the world and it takes some action what is the expected return receive after that point

Or

$$Q(S_t, a_t) = E[R_t \mid S_t, a_t]$$

S_t ↓ state, a_t ↓ action

Scanned with CamScanner

$Q(s_t, a_t) = E[R_t \mid s_t, a_t]$

Expected return from your current time point.

given this function, can we determine what is the best action to take.

Perform different queries on this function

Ultimately, the agent needs a policy $\pi(s)$, to infer the best action to take at its state, s.

The best action is the action which results in the highest expected total return.

<u>Strategy</u>:

So choose a policy that maximizes this future reward.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q(s,a)$$

the argmax of your q functions over all possible actions that you can take at the state

Scanned with CamScanner

given q function and a state that you are in you can feed in your state with every single action and evaluate what the q function would tell you the expected total reward would be given that (state, action) pair

- we can then pick the action that gives you the highest q value. and it should tell you the action

Computing policy:

given the q function

compute the argmax formulation

find the action that maximizes ~~you~~ the q function

Two classes of Deep Reinforcement Learning Algorithms

Value Learning	Policy Learning
Find $Q(s,a)$	Find $\pi(s)$
$a = \arg\max_a Q(s,a)$	Sample $a \sim \pi(s)$

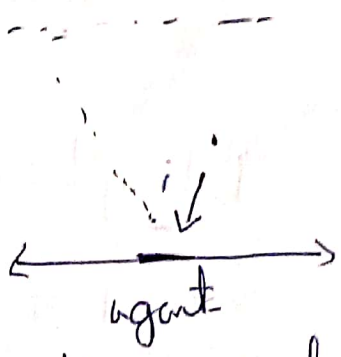
Scanned with CamScanner

In value Learning:
we will try to learn the q function and use the q function to infer the policy.

Policy Learning:
Find policy or directly Learn the policy $\pi(s)$ that governs what actions you should take.

The Q-function:
what is the total expected return that we can expect to see on any state in this game. and this is an example of one state

Atari example:



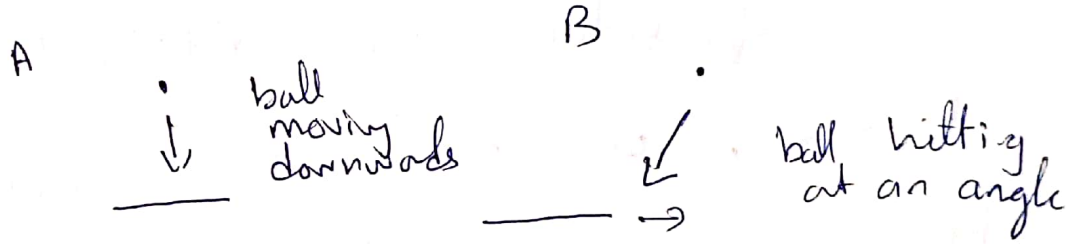
- Goal: bounce the ball at the right time and break the block
- ball is travelling moving down

Three actions:
1) stay calm
2) move left
3) move right

Scanned with CamScanner

function will tell us
what is the expected total return that we can expect in a given state action pair ~~and~~

- Some times it could be challenging
what is the q-value for a given state action pair
Given two options



which (s, a) pair has a higher Q-value?
which will give the higher expected total reward

Option A: does not move the paddle

Option B: hit the ball at the edge (extreme angle) of the paddle. Useful to hit the sides of the bricks

Scanned with CamScanner

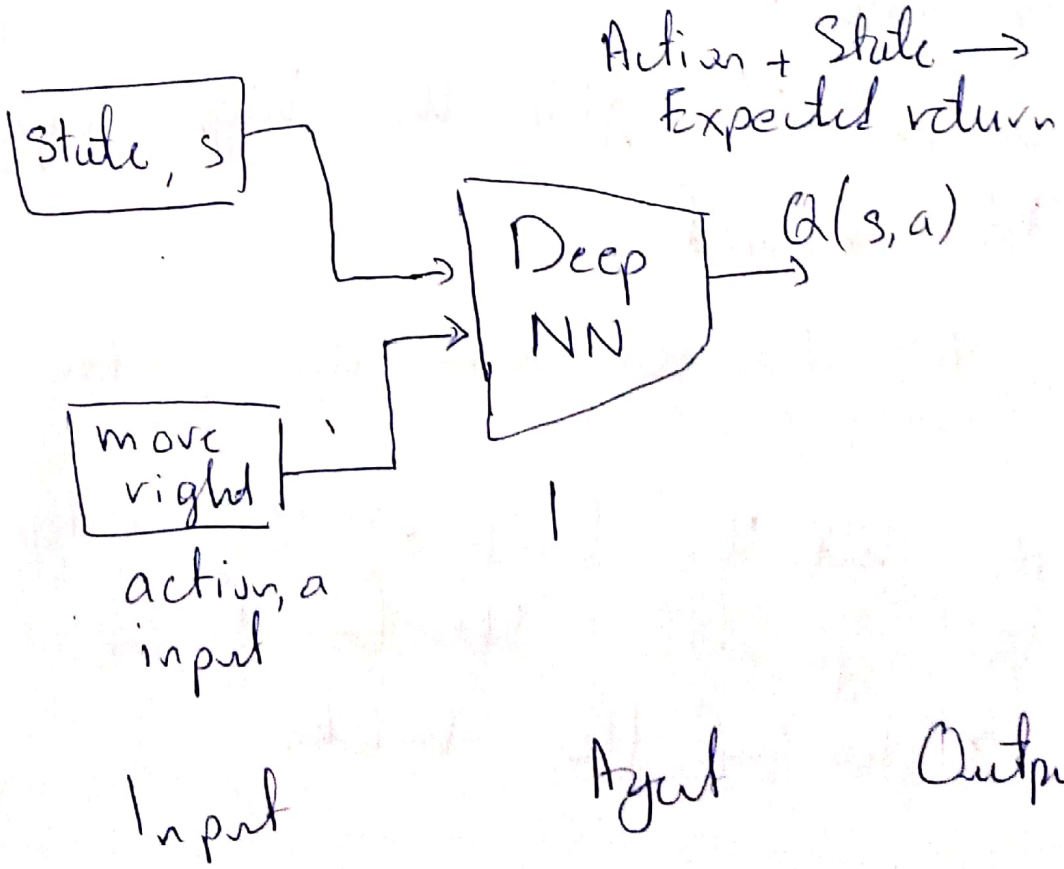
Option B is able to beat much faster

which state action pair has a higher q-value, option b

Unintuitive :: that option A was expected to be better

Since q-values are difficult to define, it is hard for humans

How can we use DNN to model Q-functions?



Scanned with CamScanner

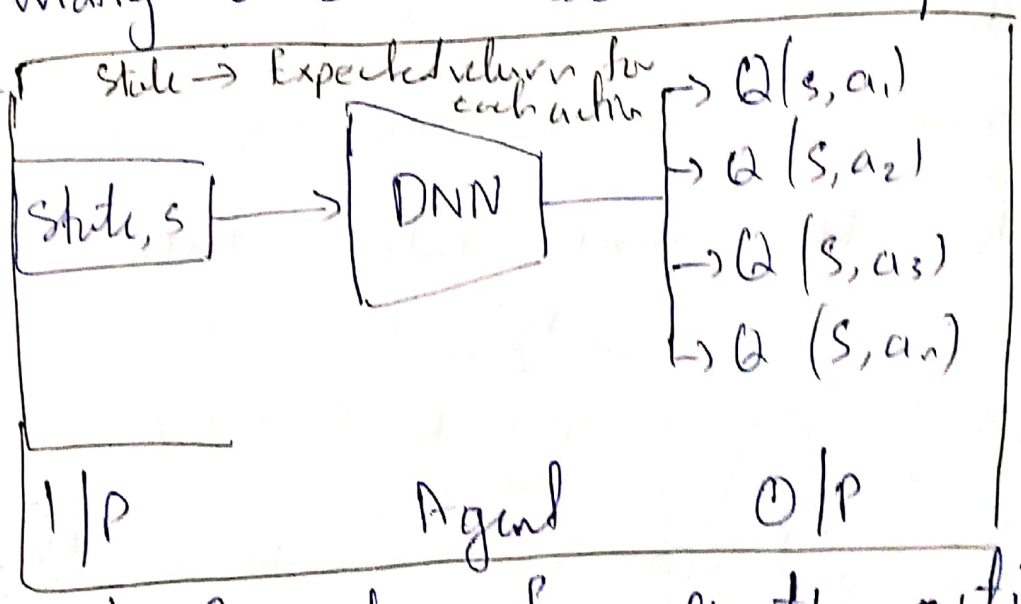
How can we use a DNN to model Q-functions (3)

1) A DNN can take input of state and actions, the network will be trained to predict the q-value for that given state action pair

- for policy, the DNN has to be evaluated n times where n is the number of possible actions that it can make in this time step.

⟹ the DNN needs to be run many times for each time step.

Option 2



compute Q-values for all the actions

How do we train this?
challenging b/c we don't have dataset of q-values

we have observations, state, action and reward.

We can use the target return to serve as our dataset for training.

Formulate the expected return if we were to take all the best actions, the initial reward plus the action that we select that maximizes the expected return for the next future state.

$$\underbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}_{\text{target}} \qquad Q(s,a)$$

γ → balancing factor

r → initial reward

$Q(s,a)$ predicted

N/w is predicting Q-values for a given (s,a) pair.

Scanned with CamScanner

$$L = E\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s,a)}^{\text{predicted}} \right\|^2\right]$$

M.S.E

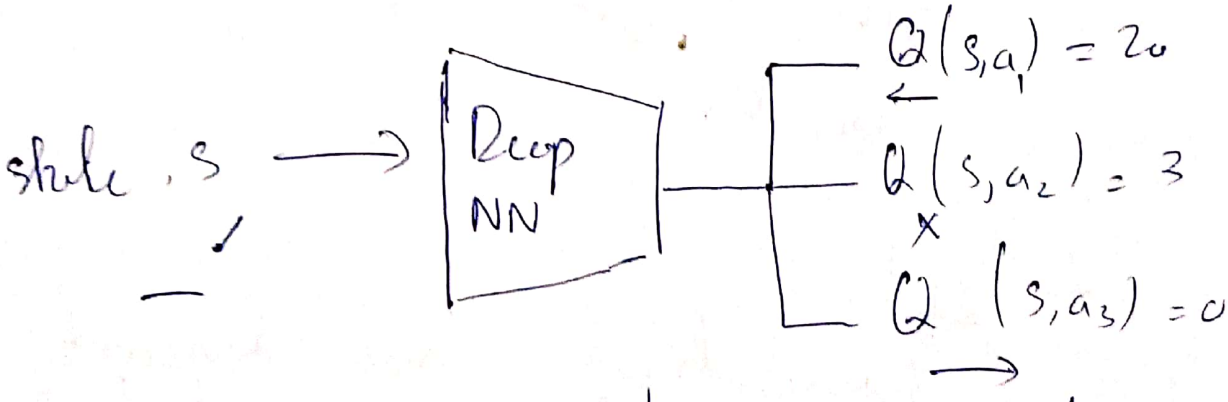
Q-loss

Use

DQN

sees as input a state and the state gets fed through the network and try to output the q-value for each possible actions.

Use NN to learn Q-function and then use to infer the optimal policy $\pi(s)$



$$\pi(s) = \arg\max_{a} Q(s,a)$$
$$= a_1$$

Scanned with CamScanner

In order to infer optimal policy it has to look for individual values

How do we compute the optimal policy.?

- looking at the max Q-value and picking the action

  pick a, and give it back to the game engine.
  and we receive our next state

- iterative process
- a variety of solutions for different games. not just breakout

- Input
  convolution → non-linear
  and posty- predict Q-values

Scanned with CamScanner

Surpass human-level	Below human level

Deep Mind
they were able
to surpass
human-level

Downside of Q-learning

Complexity.

- Can model scenarios where the action space is discrete and small.
- Cannot handle continuous action spaces. E.g. Continuous car drive

Flexibility

Policy is deterministically computed from the Q-function by maximizing the reward → cannot learn stochastic policy

- unseen probability distribution

Scanned with CamScanner

Policy gradient methods

~~Policy~~ Value Learning

a NN to learn Q-value
use this q-value to infer the best action to take → a policy

Policy learning :- inputs state, tries to directly learn the policy. that will tell us which action should we take, directly get the action by sampling from the policy fn.

Find π / s

Sample $a \sim \pi(s)$

~~How it works~~

Recall: DQN by first predicting given a state the q-value for each possible action and then pick the best (max expected return).

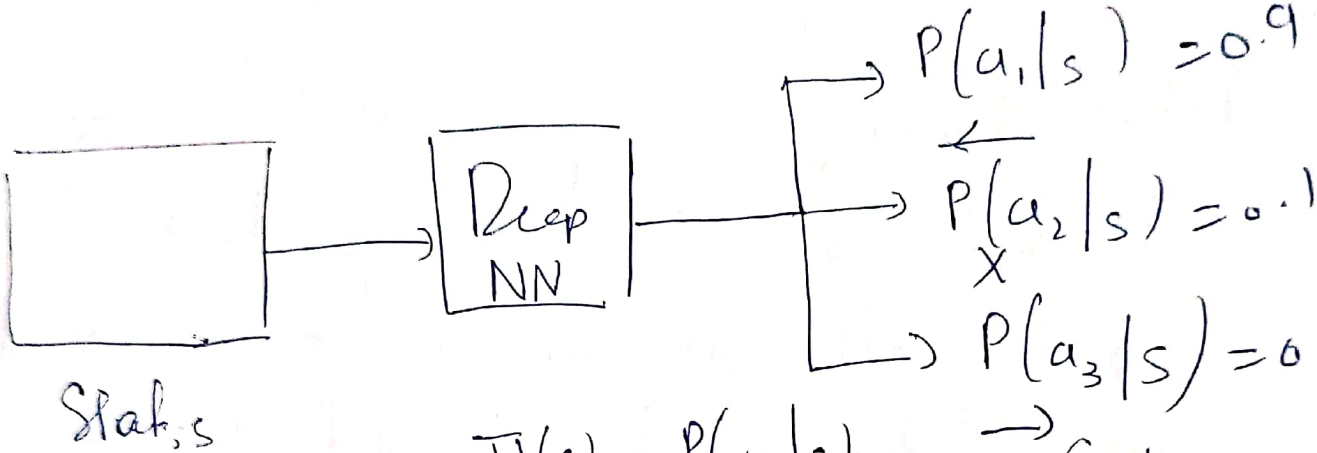
Scanned with CamScanner

Policy learning

instead of predicting the q-value, directly optimize the policy pi of s ~~so this~~

"how we should act given the state

give desired action directly. max reward



$\pi(s) \sim P(a|s)$
$= a_1$

$\sum_{a_i \in A} P(a_i|s) = 1$

Policy is defined by this PDF

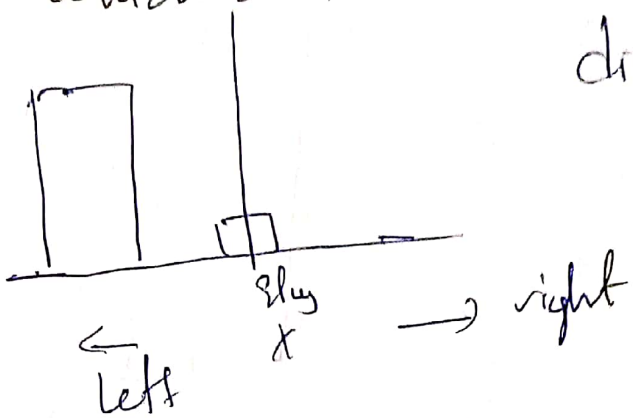
- for action we should sample from this distribution. to predict the action we should execute

- It can handle continuous spaces

Scanned with CamScanner

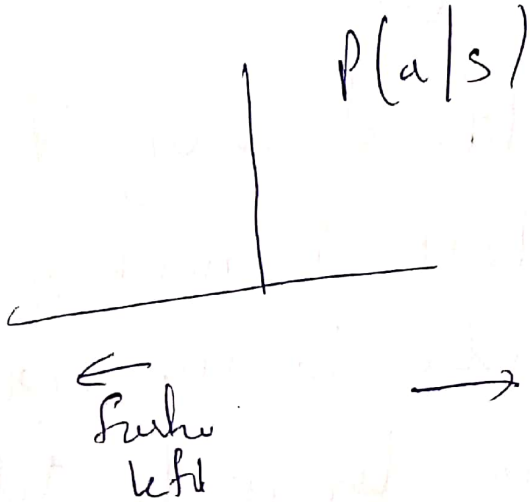
Discrete Vs continuous

Discrete: Which action should I move?



discrete

Continuous:



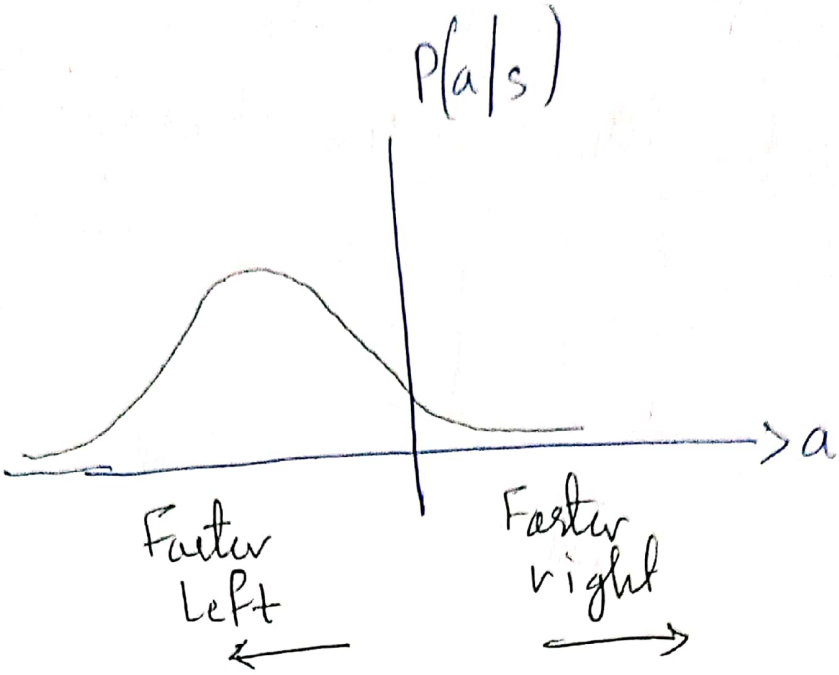
continuous

how faster we should move

Not just the direction but how fast as a real number that we should move. speed as well as direction +-

Visualize this as a continuous action space. something like gaussian distribution

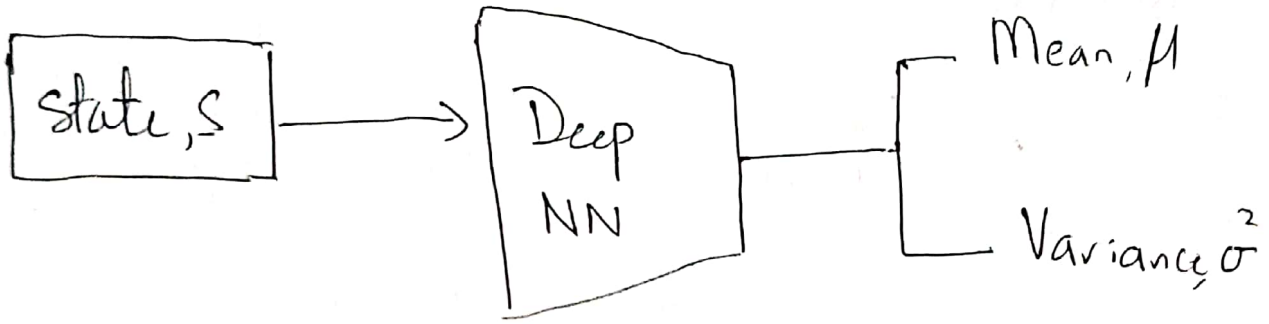
Scanned with CamScanner



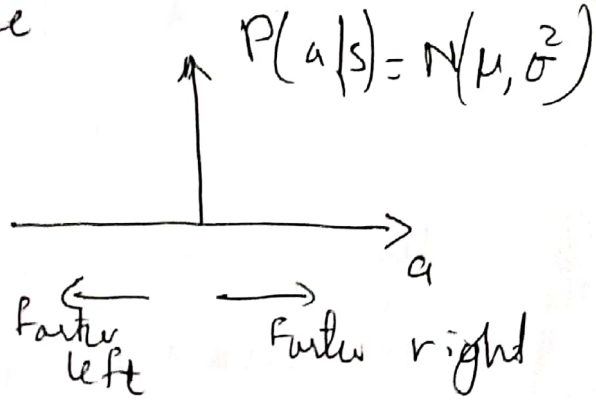
Probability of moving faster to the left is much greater than moving faster to the right.
The mean, the peak, tells us the exact numerical value of how fast we should be moving to the left and not just it should be moving

Policy Gradient (PG): Key idea

How to model continuous action space with a policy gradient method.



Policy gradient: Enables modelling of continuous action space

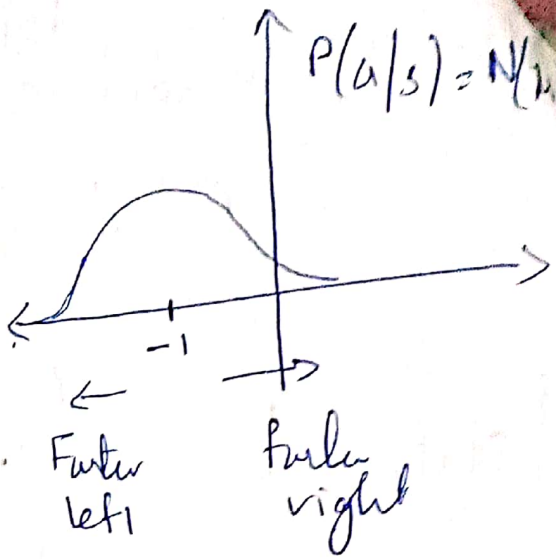


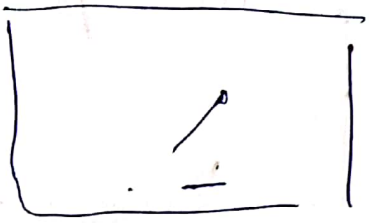
Instead of predicting the probability taking an action given a possible state

we have two o/ps but it allows us to distribute over continuous space otherwise we will have infinite no. of o/ps.

Scanned with CamScanner

-1 m/s on the left with some ~~confidence~~ variance





the paddle needs to move on the left

Mean $\mu = -1$

Variance, $\sigma^2 = 0.5$

mass of the distribution is on the left side of the number line

$P(a|s) = N(\mu, \sigma^2)$

$\pi(s) \sim P(a|s)$

$= -0.8$ [m/s]

when we sample from this distribution we can see that in this case we are getting that the action that we should move at a speed of 0.8 m/s

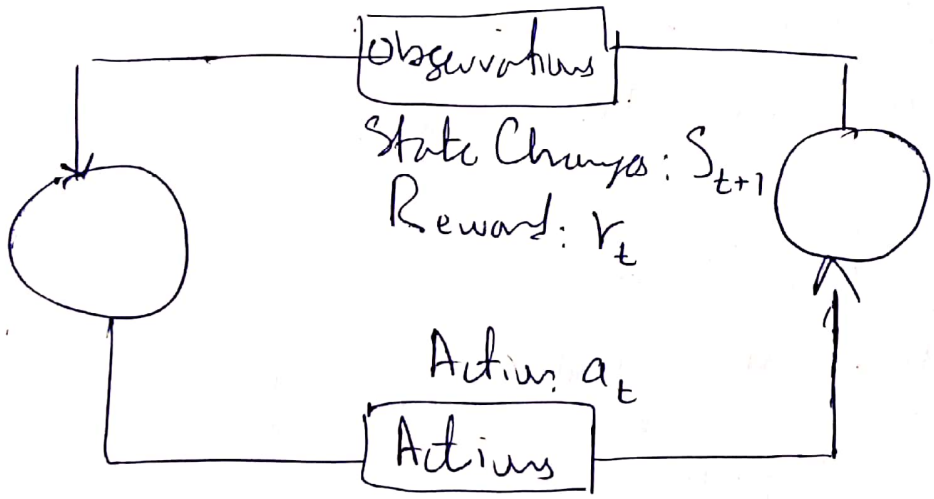
this is continuous probability distribution so we are not confined to that mean number. When we sampled our value is not mean

We can model infinite number of actions.

This probability distribution has all the ~~possible~~ properties

$$\int_{a=-\infty}^{\infty} P(a|s) = 1$$

Training Policy Gradients: Case Study



Reinforcement Learning Loop:

Scanned with CamScanner

How to use policy gradient for an autonomous vehicle problem.

Agent: vehicle

State: camera, lidar etc

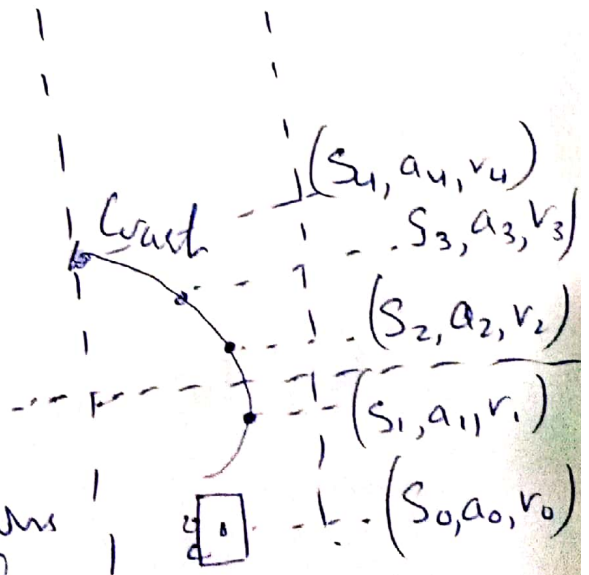
Action: steering wheel angle

Reward: distance travelled

An example of continuous data.
infinite data
How to train this car.

Training Algorithm:

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards.
4. Decrease probability of actions that resulted in low reward
5. Increase the probability of actions that resulted in higher reward.



Scanned with CamScanner

Take all the action reward pairs and decrease the probability of taking any action that it took close to the time where the termination happened (so close to the crash) we want to decrease the probability of making any of those actions again in the future.

Increase the probability in the beginning.

We are assuming that since the crash occured in the second part, something really bad occured in this part.

An unintelligent approach decrease the probability of low reward and increase the probability of high reward.

gt doesn't know which one is bad specially in the beginning

Scanned with CamScanner

b/c no feedback

- keep doing it again and the car will ~~keep~~ keep running longer.

decrease the probability of low reward and increase the probability of high reward.

keep doing until it eventually learns to stand following lanes without crashing.

Not to do any action that occurs very close to crash.

How can we update policy on every training iteration to decrease the probability of bad events and increase the probability of good events.

Scanned with CamScanner

Loss function:

$$\text{Loss} = -\log P(a_t \mid s_t)\, R_t$$

Log likelihood

total discounted ~~reward~~ return

~~high log likelihood will~~

~~high reward~~

G.D.

$$w' = w - \nabla \text{loss}$$

$$= w + \boxed{\nabla \log P(a_t \mid s_t)\, R_t}$$

likelihood of action

Reward

Policy gradient

If the action is +ve very good reward will be good.

Scanned with CamScanner