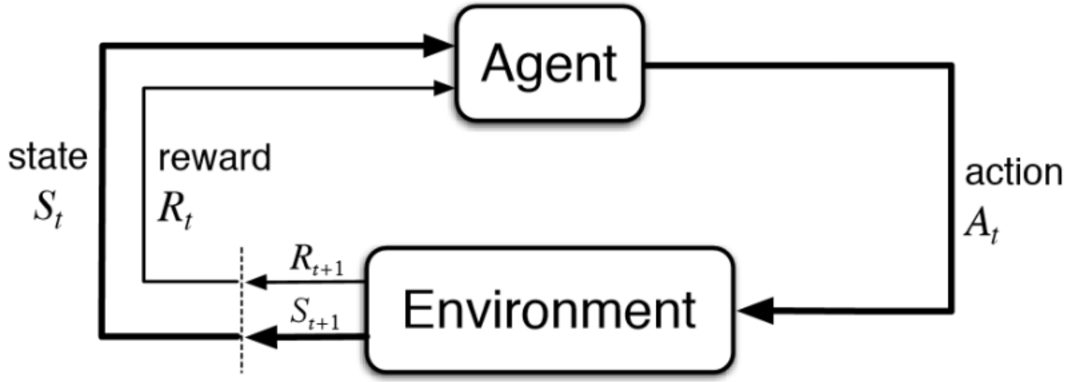
SELF DRIVING WITH REINFORCEMENT LEARNING

Ran Minerbi

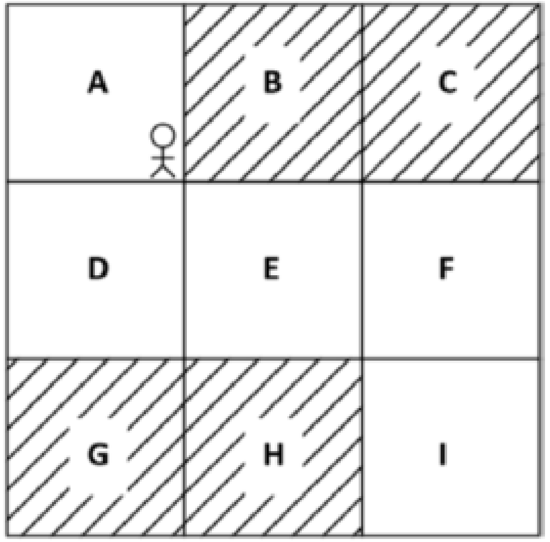
REINFORCEMENT LEARNING BASICS

- Reinforcement learning problems are basically Markov process
- Next action is determined based on current state
  - Agent perform action(s)
  - Environment return reward and new state to agent



EXAMPLE: FROZEN LAKE PROBLEM

State	Action	Reward	Next State
A	Right	-1	B
A	Down	1	D
D	Up	-1	A
D	Right	1	E
D	Down	-1	G
E	Down	-1	H
E	Up	-1	B
E	Right	1	F
F	Up	-1	C
F	Down	1	I
F	Left	-1	E



Need to make it to cross the grid only through white states

STATES AND ACTIONS SPACES FOR WHEEL STEERING

- Self wheel steering possible states :
  I. Car speed
  II. Distance from lane
  III. theta – car heading from lane heading angle
  IV. Theta dot - car heading from lane heading angle rate of change

Actions:

- Wheel steering [-1,1]

STATES AND ACTIONS FOR CRUISE CONTROL

- Self cruise control possible states :
  I. road slope
  II. Car speed
  III. Wight of passengers and lagage
  IV. Current gear
  V. Wheel air pressure

Actions:

- Gas pedal [0,1]

BELMAN EQUATION FOR STATE VALUE

- Belman equation for state value is given by the immediate reward and next state value

$$V(s) = R(s, a, s') + \gamma V(s')$$

- $R(s$, a, $s')$ implies the immediate reward obtained while performing an action *a* in state *s* and moving to the next state s'
- γ is the discount factor
- V(s') is the value of the next state value

BELMAN EQUATION FOR Q FUNCTION

- Same as bellman equation for value function but takes into account (state ,action) pair

$$Q(s,a) = R(s,a,s') + \gamma Q(s',a')$$

- $R(s, \text{a}, s')$ implies the immediate reward obtained while performing an action *a* in state *s* and moving to the next state s'
- γ is the discount factor
- $Q(\text{s}', a')$ is the Q value of the next state-action pair

BELMAN OPTIMAL Q FUNCTION

- Optimal Q function gives maximum state action value

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

- value function for state s can be denoted from Q function

$$V^{*}(s) = \max_{a} Q^{*}(s,a)$$

- Q function can be denoted out of value function

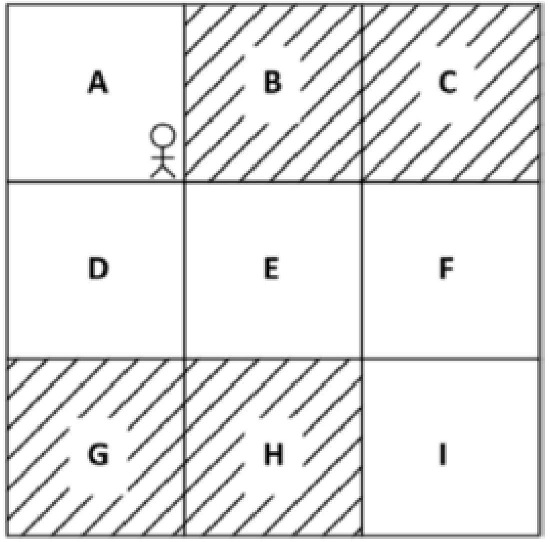
$$Q^{*}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{*}(s')]$$

EXAMPLE: REACH FROM A TO I

- Our objective is to find the optimal policy – with max return value
- In order to compute the policy - first need to compute Q functions
- Once we have Q functions – we extract policy by selecting action per each state that has max Q value

In this case we have:
9 States x ~4 actions per state = 36 Q functions
What if we have 50K states and 200 actions per state ?
Very clumsy for a very simple problem

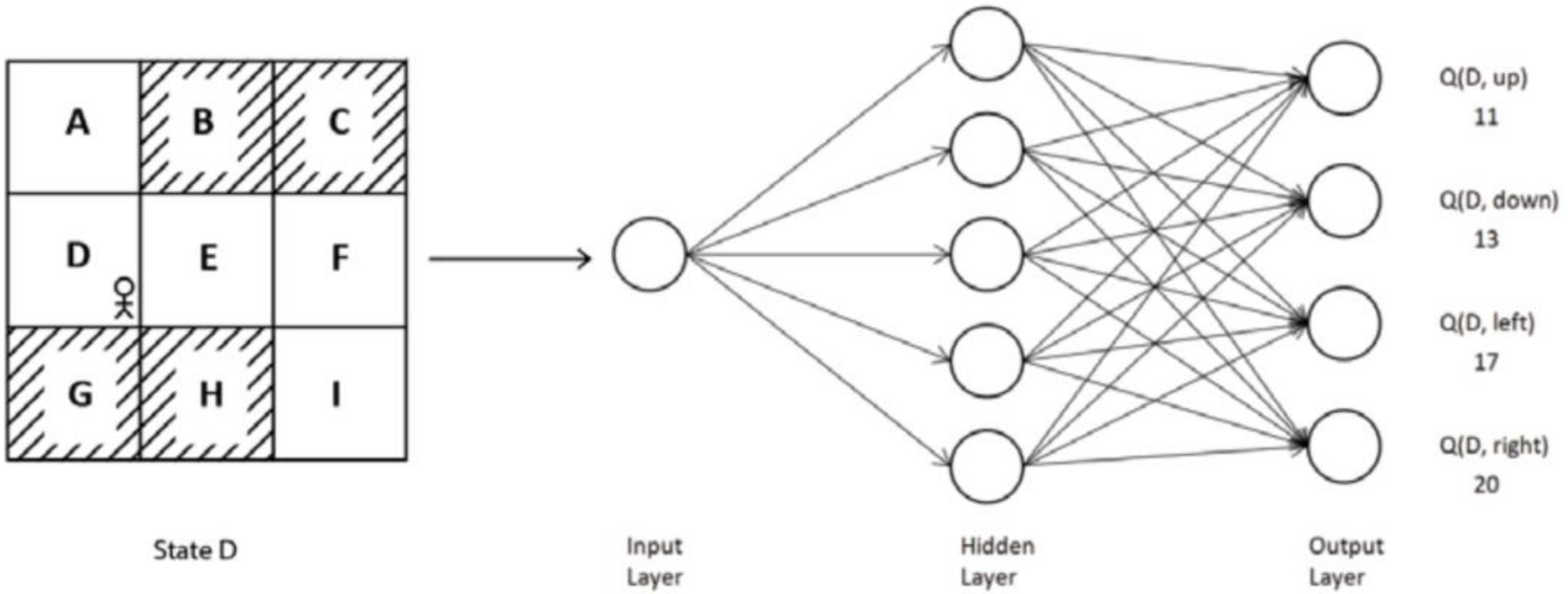
Can we Do better ?



Compute Q values

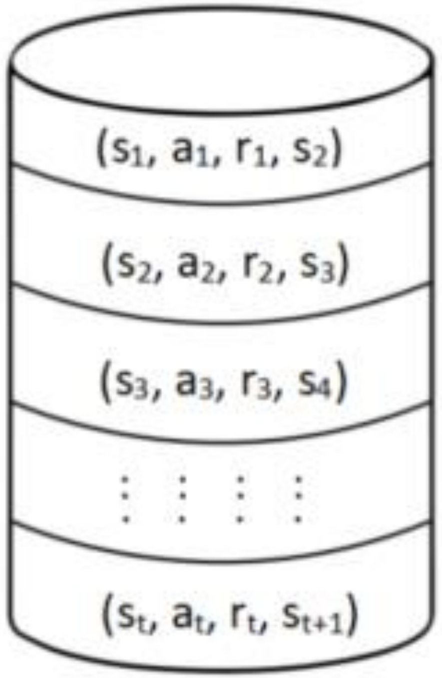
	State	Action	Reward	Next State
Q(s=A,a)	A	Right	-1	B
	A	Down	1	D
Q(s=D,a)	D	Up	-1	A
	D	Right	1	E
	D	Down	-1	G
Q(s=E,a)	E	Down	-1	H
	E	Up	-1	B
	E	Right	1	F
Q(s=F,a)	F	Up	-1	C
	F	Down	1	I
	F	Left	-1	E

DEEP Q NETWORK – APPROXIMATE Q VALUE WITHIN NEURAL NETWORK



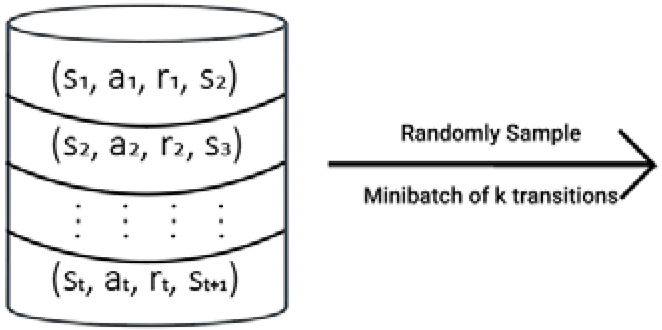
DEEP Q NETWORK – REPLAY BUFFER

- How shall we train our DQN ? Record every movement !
    - Make random actions a from state s to state s' and get reward r
    - Store s,a,r,s' in the replay buffer for future prediction



DEEP Q NETWORK – TRAINING PROCEDURE

- Get next action from network $a = \arg\max_{a} Q_\theta(s, a)$
- Execute selected action , and store in replay buffer (s,a,r,s')
- randomly sample K values from replay buffer
- Calculate target value within next state (out of replay buffer)
- Calculate predicted value
- Compute loss within the below formula based on the K samples



$$L(\theta) = \frac{1}{K}\sum_{i=1}^{K}(r_i + \gamma \max_{a'} Q_\theta(s'_i, a') - Q_\theta(s_i, a_i))^2$$

DEEP Q NETWORK – LOSS FUNCTION CALCULATION

$$\text{MSE} = \frac{1}{K}\sum_{i=1}^{K}(y_i - \hat{y}_i)^2$$

- Calculate loss following
- Yi – is the target value
- Y^i – is the predicted value from network
- K –is nimi batch size

$$Q^*(s,a) = r + \gamma \max_{a'} Q^*(s',a')$$

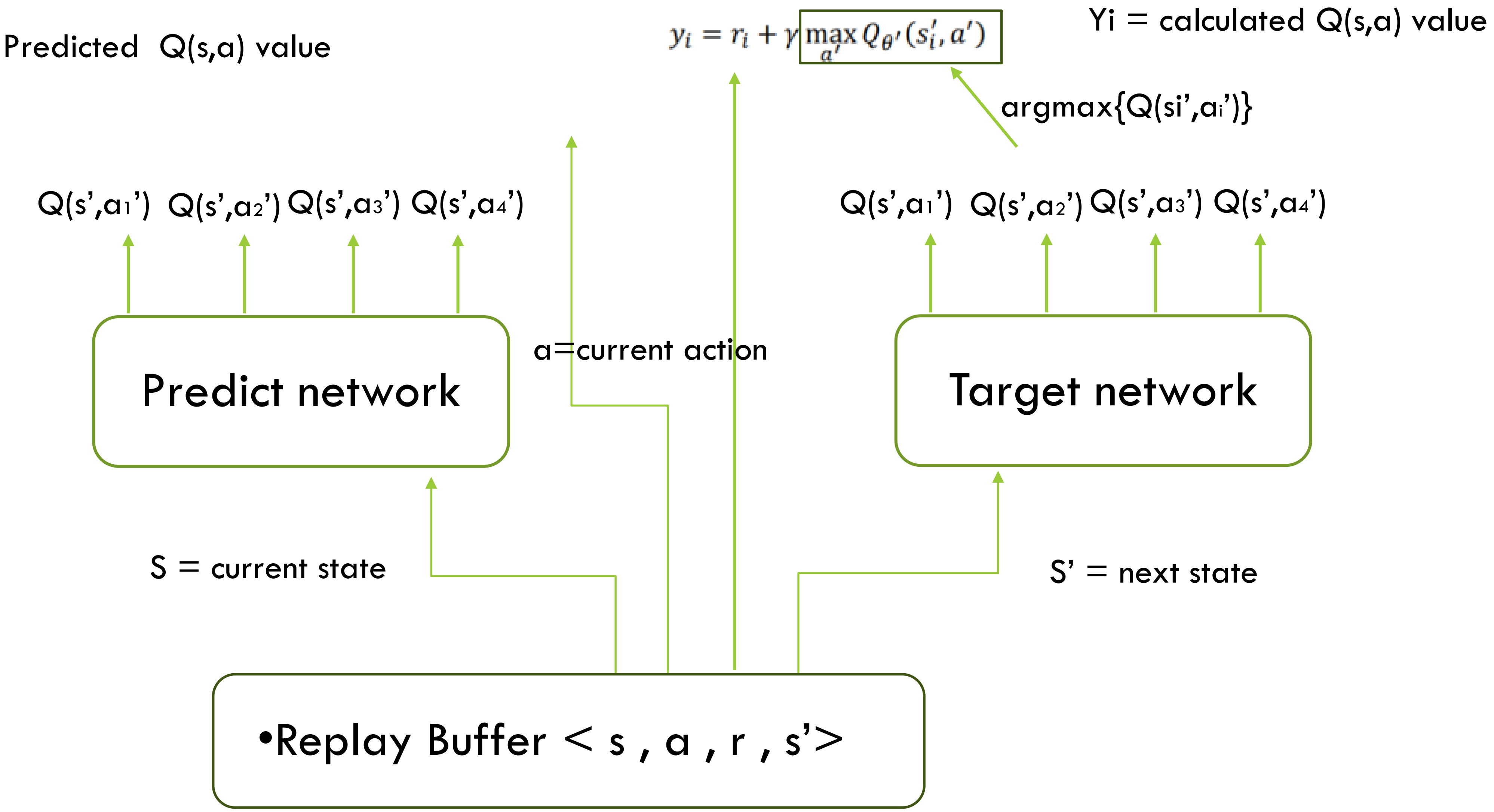
- Calculate target value with belman equation

$$L(\theta) = Q^*(s,a) - Q_\theta(s,a)$$

- Calculate loss

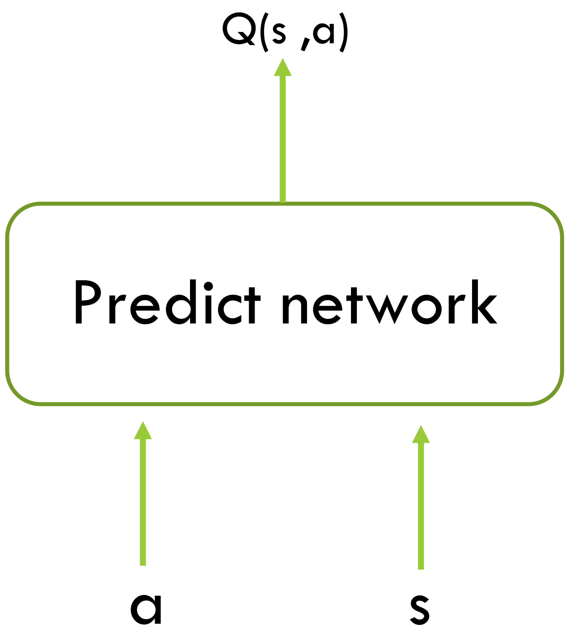
$$L(\theta) = r + \gamma \max_{a'} \underbrace{\boxed{Q(s',a')}}_{\text{How do we compute this?}} - Q_\theta(s,a)$$

DEEP Q NETWORK – STRUCTURE OUTLINE



WHAT IF OUR ACTION SPACE IS CONTINUOUS ?

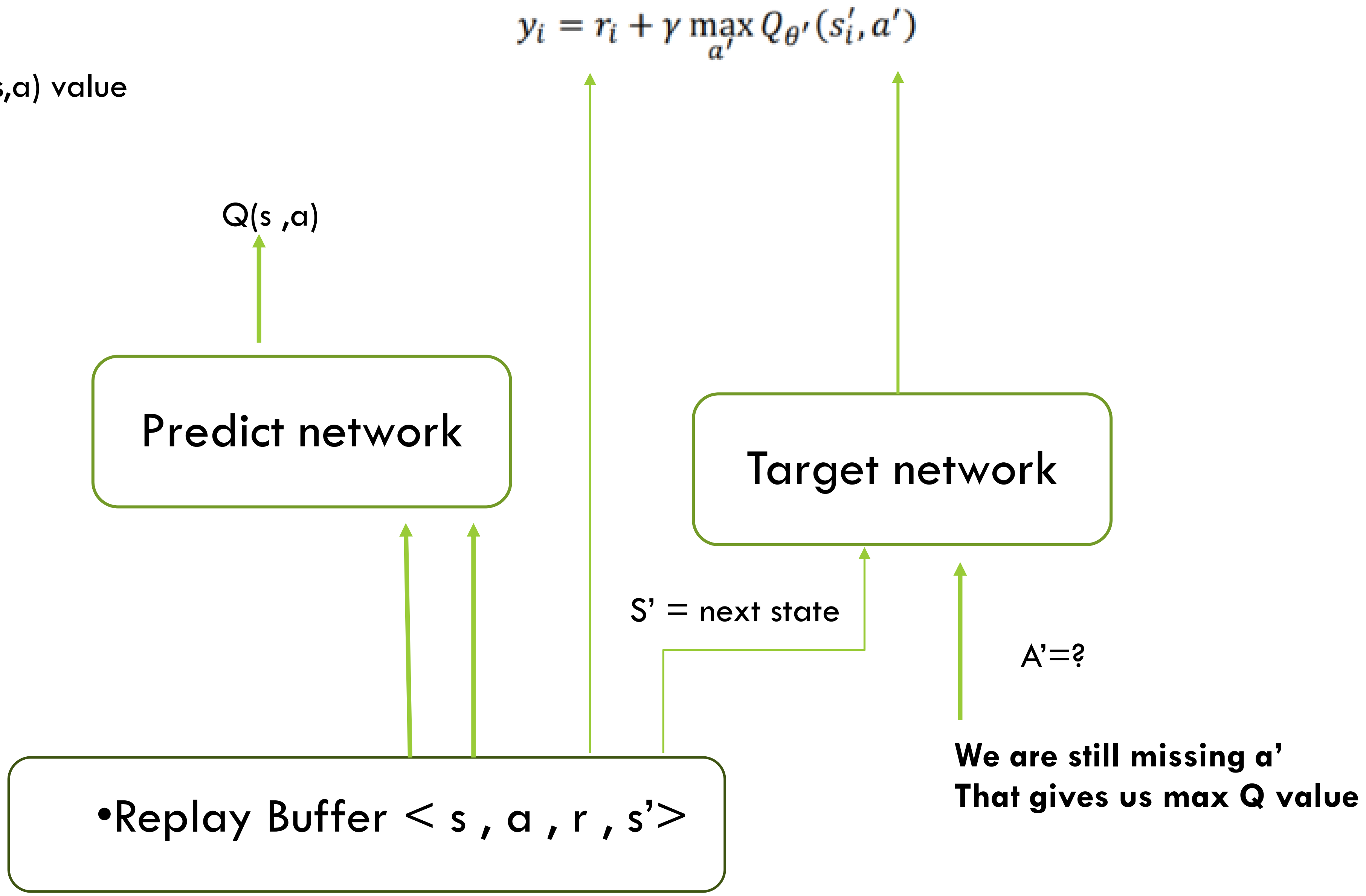
- it might be difficult to use DQN since its impossible to associate our action to the predicting network output
- What if we could have network that predict Q value based on state and action ?



S = state
a= action

CONTINUOUS ACTION SPACE

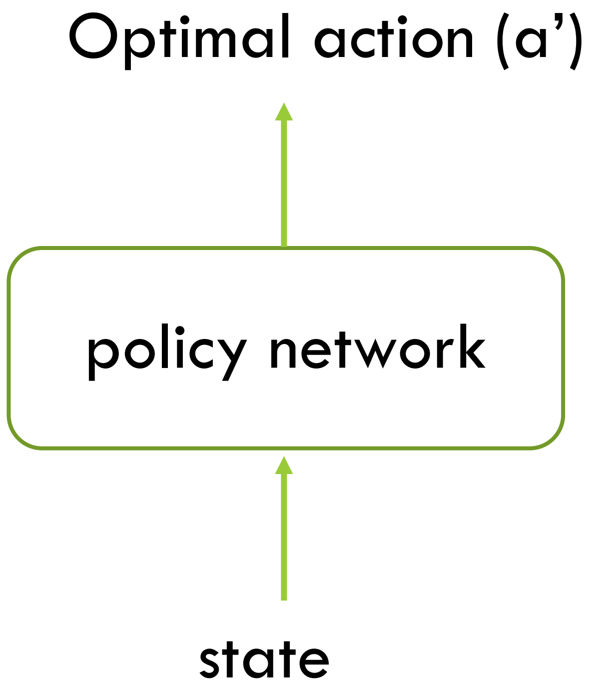
Predicted Q(s,a) value



S = current state
a= action

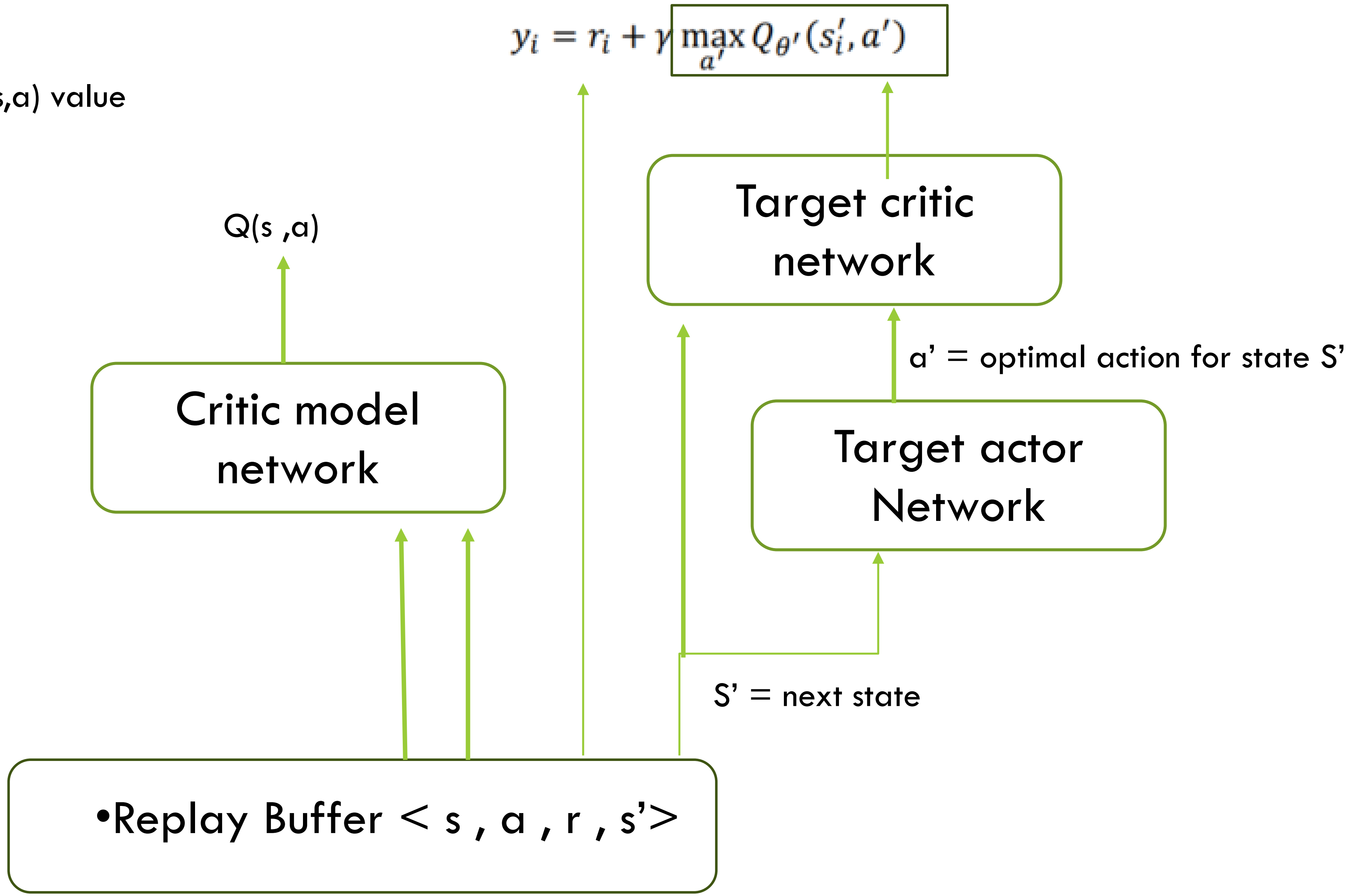
CONTINUOUS ACTION SPACE

- We are still missing optimal next action input
- Lets initiate policy network - getting State as an input and issue optimal action for the state



TRAINING CRITIC NETWORK

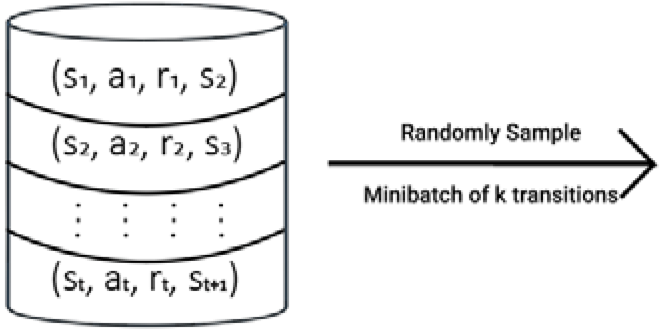
Predicted Q(s,a) value



S = current state
a= action

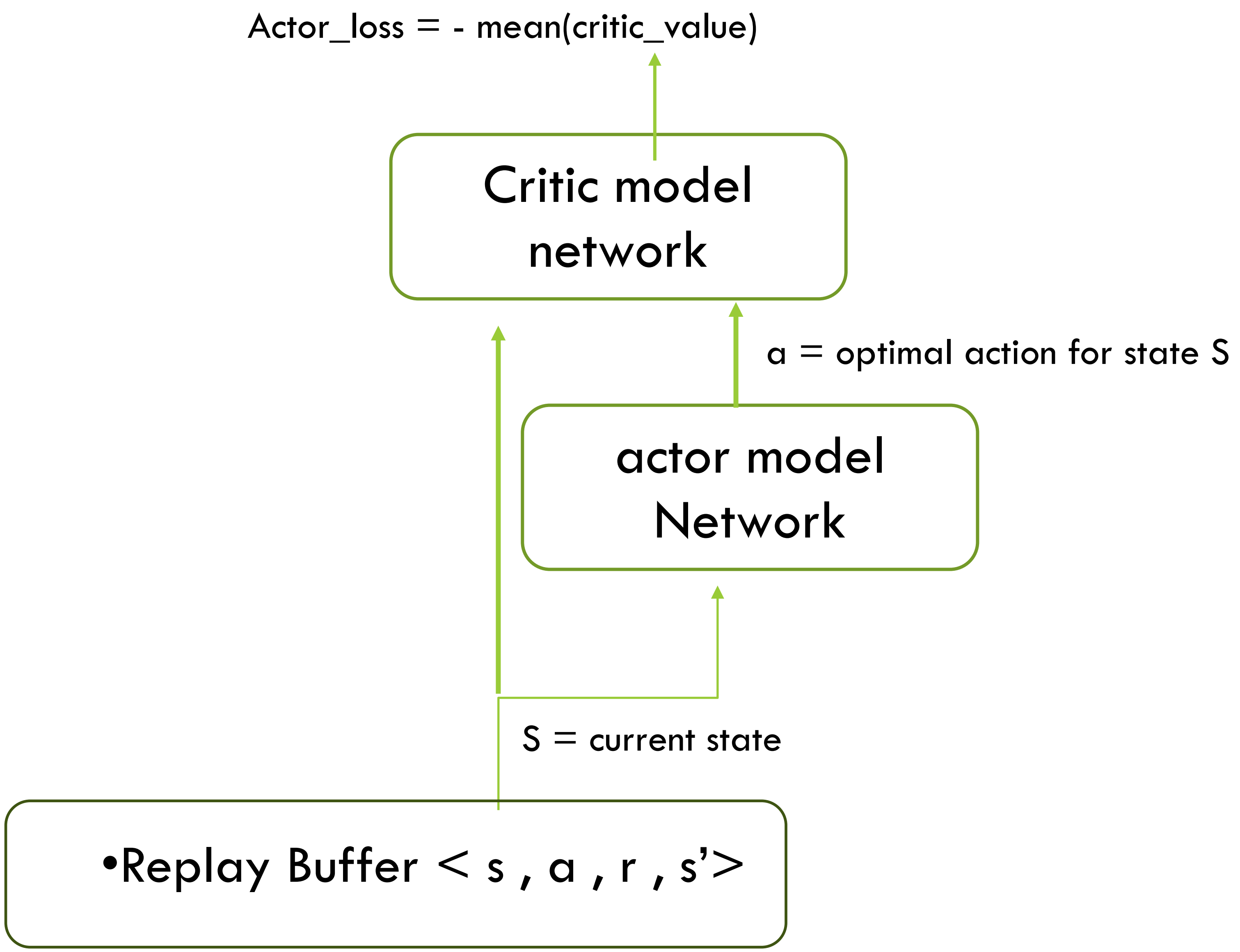
VALUE NETWORK LOSS CALCULATION

- Collect min batch of K size from replay buffer
- predict Q(s,a) and max{Q(s',a')} with value network
- Calculate loss within the below equation



$$L(\theta) = \frac{1}{K}\sum_{i=1}^{K}(r_i + \gamma \max_{a'} Q_\theta(s'_i, a') - Q_\theta(s_i, a_i))^2$$

TRAINING ACTOR NETWORK



S = current state
a= action

ACTOR-CRITIC PROCEDURE OUTLINE

- Get next action from policy network (added OU noise)
- Execute selected action , log in replay buffer (s,a,r,s')
- randomly sample K values from replay buffer
- Calculate target value of critic network (out of replay buffer)
- Calculate predicted value of critic
- Compute loss within the above equation based on the K samples
- Compute gradient for critic network
- Calculate actor loss
- Calc gradient for actor network
- Update auxiliary networks

ACTOR-CRITIC SUMMARY

•Actor –critic method

- Actor - learn the mapping between the state and action. learn the optimal policy that gives the maximum return
- Critic – value network - evaluate the action produced by the actor network