Temporal Differencing [TD]

policy Evaluation

① start with $V_0(s)=0$

② Iterate until convergence:

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s,\pi_k(s))\left[R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')\right]$$

(we knew transition probability P, and Reward function)

How to extend policy evaluation when P, R are not known?

$P(s'|s,a)$ and $R(s,a,s')$ are unknown and only revealed through experience

* Every time you take action 'a' from state 's' you get a sample from the unknown $P(s'|s,a)$ and corresponding reward $R(s,a,s')$

Idea: treat single sample we get as representative of the distribution and apply an "incremental update to reduce the 'Bellman Error"

Sample of $V(s)$: sample $= R(s,\pi(s),s') + \gamma V_i^{\pi}(s')$

TD update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(\text{sample} - V^{\pi}(s))$

$Q[s,a]$ = immediate reward + discounted reward

By acting in the environment and slowly taking actions in the environment, you get more information about $P(s'|s,a)$ and $R(s,a,s')$

Bellman update rule:
Iterate until convergence -

taking expectation of this quantity under $P(s'|s,\pi_k(s))$

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s,\pi_k(s)) \left[R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s') \right]$$

Every time we perform an action on the environment, we get to see one sample of the quantity $R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')$

Noisy estimate to treat the single sample as the expectation of the distibution, so we don't iterate until convergence, we try to make $V_{i+1}^{\pi_k}(s)$ almost equivalent $R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')$

Sample of $V(s)$: $\text{sample} = R(s,\pi(s),s') + \gamma V_i^{\pi}(s')$

TD update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(\text{sample} - V^{\pi}(s))$

α: incremental update

Doing this for each sample = computing the running average over samples

Note :-

↳ Temporal differencing treats every single sample (information) we encounter as representative of the distribution and hence, we don't wait for many interactions in the environment to improve policy

↳ We get to learn from every single step in the environment (action)

↳ learning much more quickly, making use of every single interaction in the environment to update policy or value function

Applying Temporal Differencing to Learn Optimal Q-Function: $Q^*(s,a)$

* learning the optimal Q-Function equivalent to be able to perform optimal actions in the environment. (pick $a = \text{argmax}$ of Q-function)

Learning $Q^*(s,a)$

Bellman Equation for optimal Q^*:

$$Q^*(s,a) = \sum_{s' \in S} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

State action value iteration $\geq$ Q-value iteration

$$Q_{i+1}(s) \leftarrow \sum_{s' \in S} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

↓

Requires access to $P(s'|s,a)$ and $R(s,a,s')$ of which we only have samples from experience

↓

Apply TD trick to this

↓

Q-value iteration:

$$Q_{i+1}(s) \leftarrow \sum_{s' \in S} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

TD version:- Treat single sample you get as a representative of the distribution, and apply an incremental update to reduce the 'Bellman Error'

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(\text{sample} - Q(s,a)\right)$$

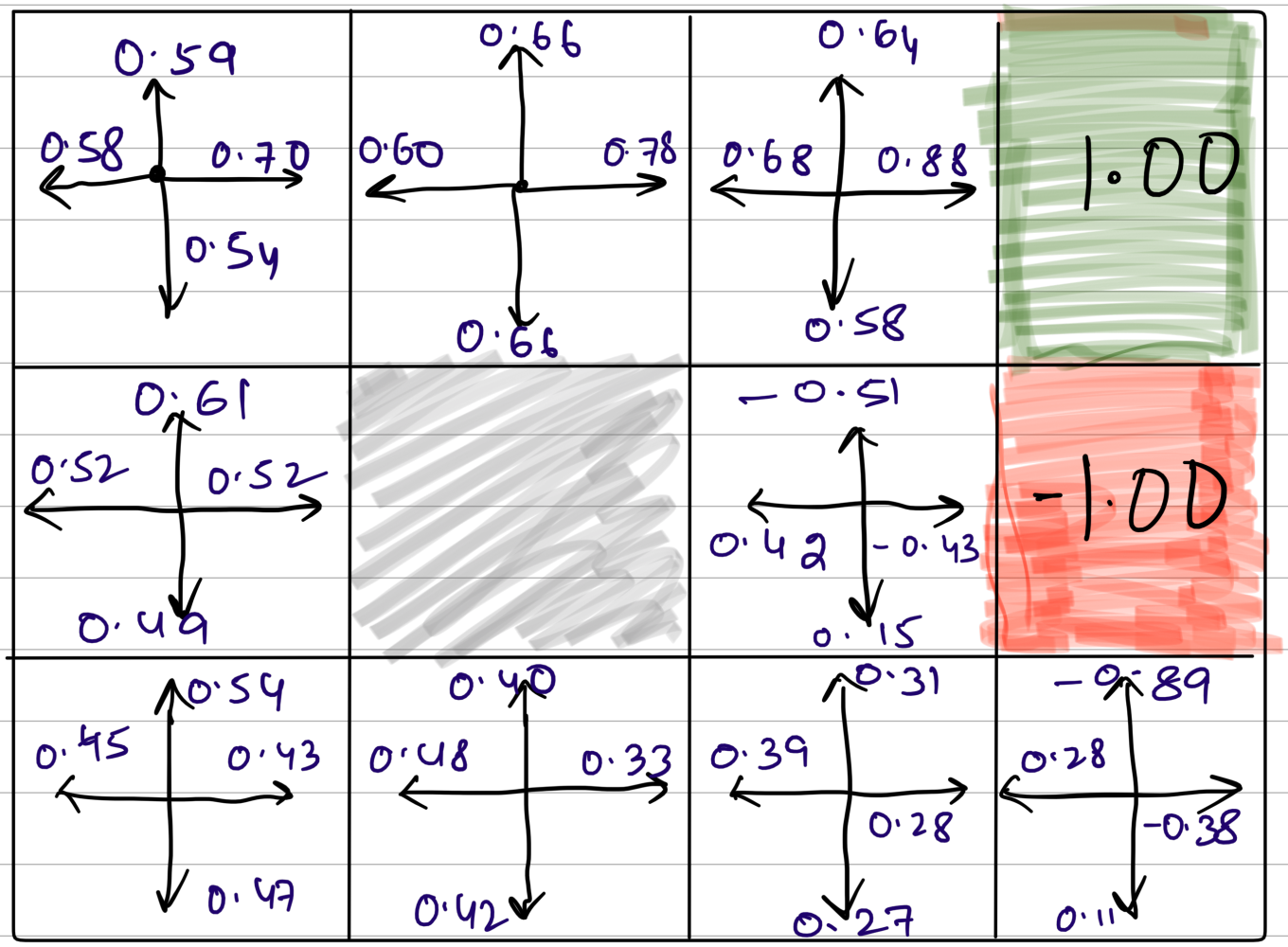
$$\text{sample} = R(s,a,s') + \gamma \max_{a'} Q_i(s',a')$$

Bellman Error

) Execute a single action a from state s and observe s' and R:

This is called Q-learning

Q-function - is a function of both state and the action



Q-values after 1000 Episodes

Deep Q-learning

Approximate Q-learning

=

Deep Q-Networks (DQN)

① Target Network

② Experience Replay Buffer

} a important components in a DQN

↳ Due to target network and an Experience Replay Buffer, DQN learns more efficiently and reliably.

↳ DQN shows better performance

Target network - provides a more stable estimate of Q-values

Experience replay Buffer - prevents overfitting.

The purpose of the target network is to stabilize the learning process and improve the overall performance of the agent

↳ In a DQN, a target network is a copy of the neural network used to predict

the Q-values of actions in a reinforcement learning agent

↳ During training, the Q-values of actions are estimated using the neural network

↳ Since the weights of the network are constantly being updated, this leads to instability in the Q-value iterations

↳ To address this issue, a target network is created by making a copy of the neural network and freezing its weights.

↳ The target network is then used to estimate the Q-values of future states during the learning process.

↳ The Q-values predicted by the target network are used as target values for updating the neural network.

↳ By periodically updating the target network with the weights of the main neural network, the learning process becomes more stable and the agent can learn to make better decisions.

Experience Replay Buffer:

- Experience replay buffer stores the experiences observed by an agent while interacting with the environment

- Experience replay buffer stores the agent's experience in the form of tuples (s, a, r, s')
  - s :- current state of agent
  - a :- action taken by the agent
  - r :- reward received by the agent for taking the action 'a'
  - s' :- resulting next state

- The tuples are stored in a buffer of fixed size

- During the training process, the agent samples a minibatch of experiences randomly from the replay buffer and uses them to update its policy

- It is not practical to perform learning by updating the agent's policy after every experience

- Instead, the agent can learn from a sample

of past experiences stored in the replay buffer, which helps to reduce the variance in the training process.

↳ This improves the agent's performance.

Why 'Deep' Q-learning?

when we huge number of states and actions, we need to create a huge Q-table

Limitations

- ↳ The amount of memory required to store q-table
- ↳ The amount of time to explore each state

The Idea :- Approximate Q-values with ML models such as neural networks.

Steps involved in RL using DQN (RL - Reinforcement Learning)

1. All past experience is stored by user in memory

2. The next action is determined by the maximum output of the Q-network