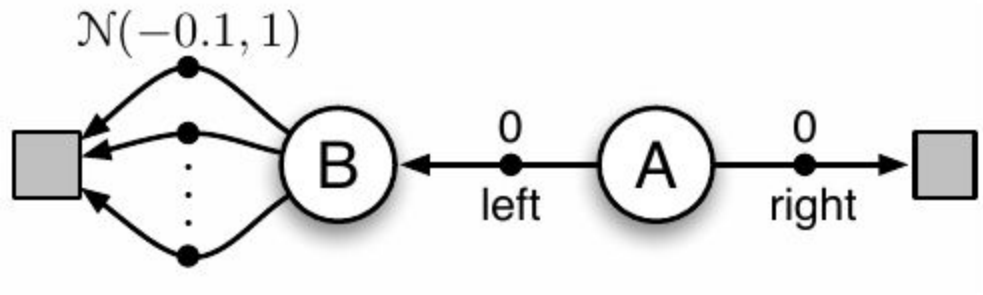
Lecture 6: Maximization Bias and State Aggregation

B. Ravindran

Maximization Bias

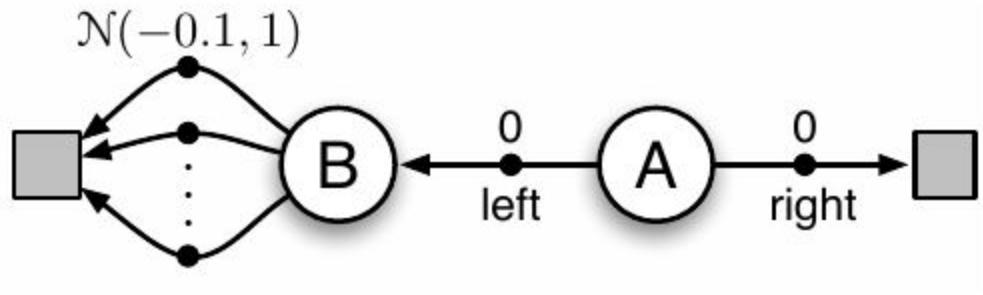
Consider the simple example below:



- A is the starting state.
- T(A, left, B) = 1
- R(A, left) = 0, R(A, right) = 0
- From B, there are |N| actions available, each of which results in a terminal state. And these |N| actions are normally distributed with mean = -0.1 and std = 1

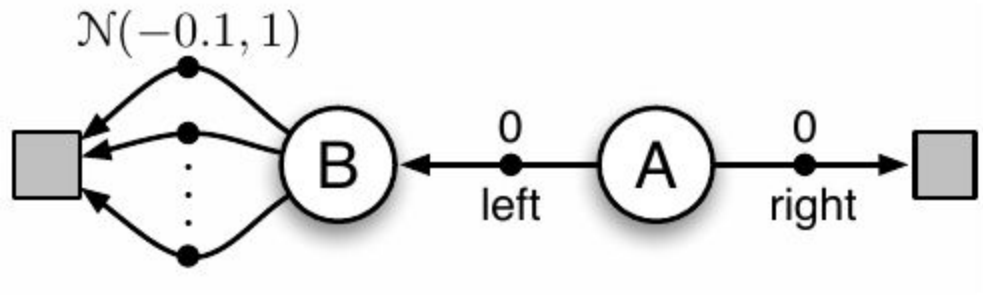
Maximization Bias

Which direction to move from A?



Maximization Bias

Which direction to move from A?

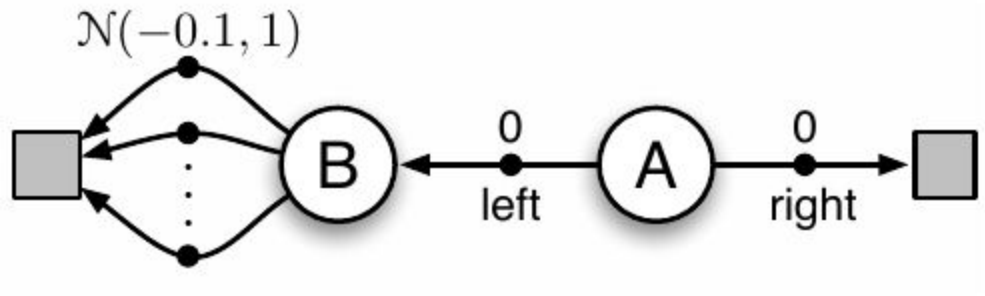


$E[G_t \mid s_0 = A, a_0 = \text{left}] = -0.1$

$E[G_t \mid s_0 = A, a_0 = \text{right}] = 0$

Maximization Bias

Which direction to move from A?

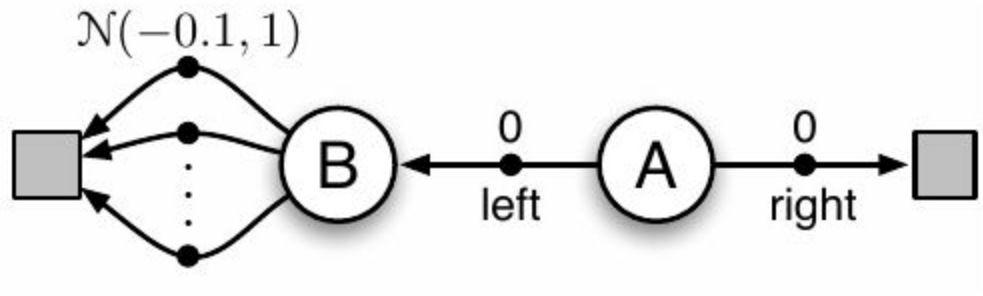


$E[G_t \mid s_0 = A, a_0 = \text{left}] = -0.1$

$E[G_t \mid s_0 = A, a_0 = \text{right}] = 0$

Maximization Bias

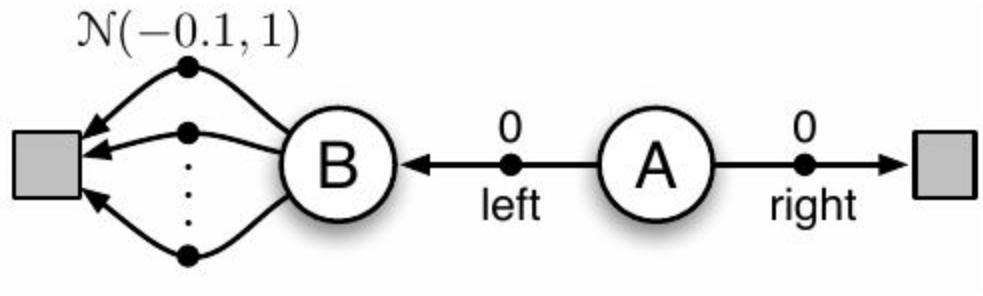
What happens when we learn a policy using Q-learning?



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Maximization Bias

What happens when we learn a policy using Q-learning?

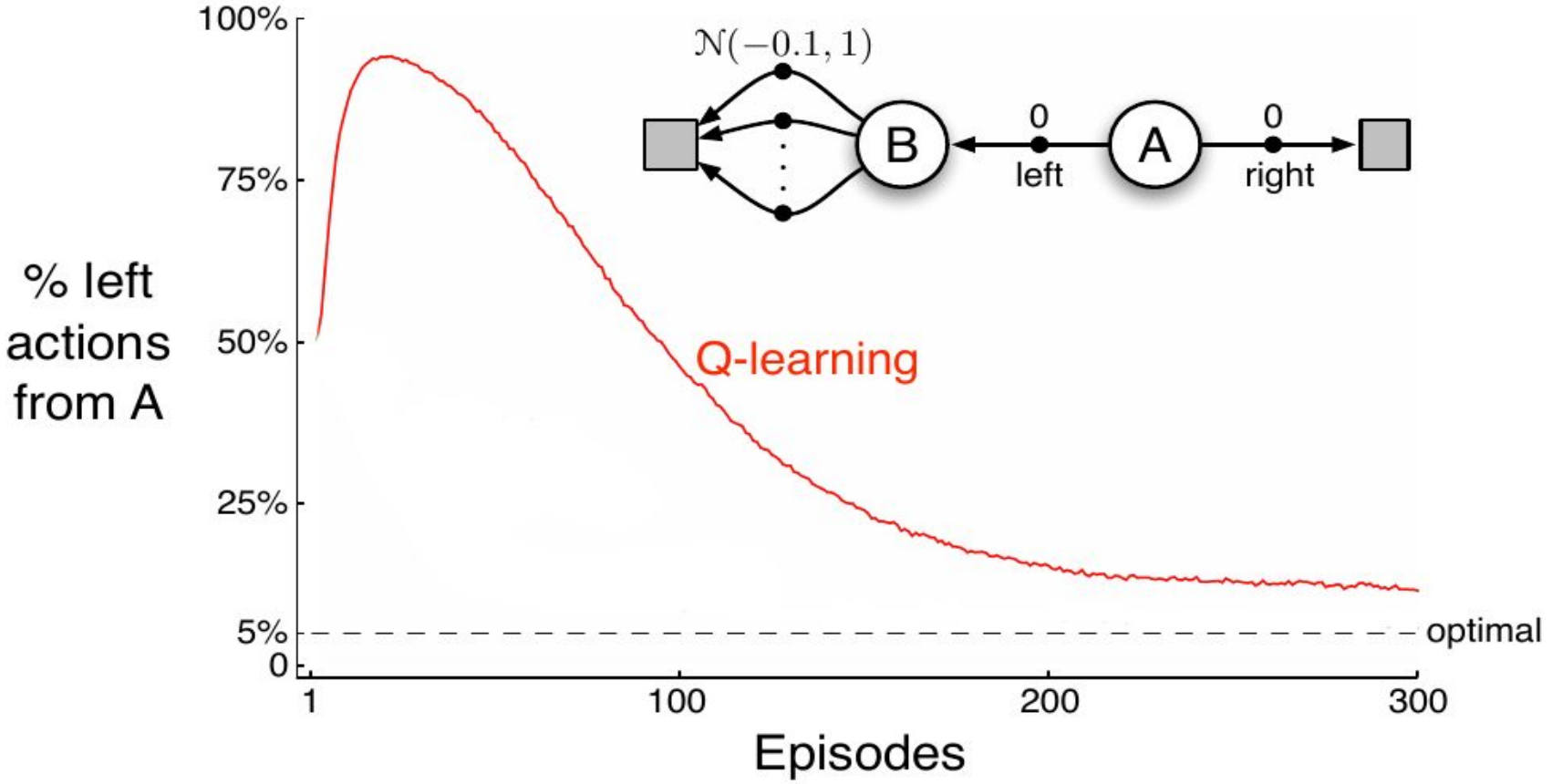


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Using maximum over estimate
as an estimate of the maximum!

Leads to a positive bias, called **maximization bias**.

Maximization Bias



ε = 0.1 for above example
Therefore, 10% of the actions are random.
Optimal => 5% can be right (random) and 95% should be left.

Double Q-learning

The problem can also be viewed as:

Using the same samples both to determine the maximizing action and to estimate its value

Solution: Use different estimates for maximizing the action and estimating its value

Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

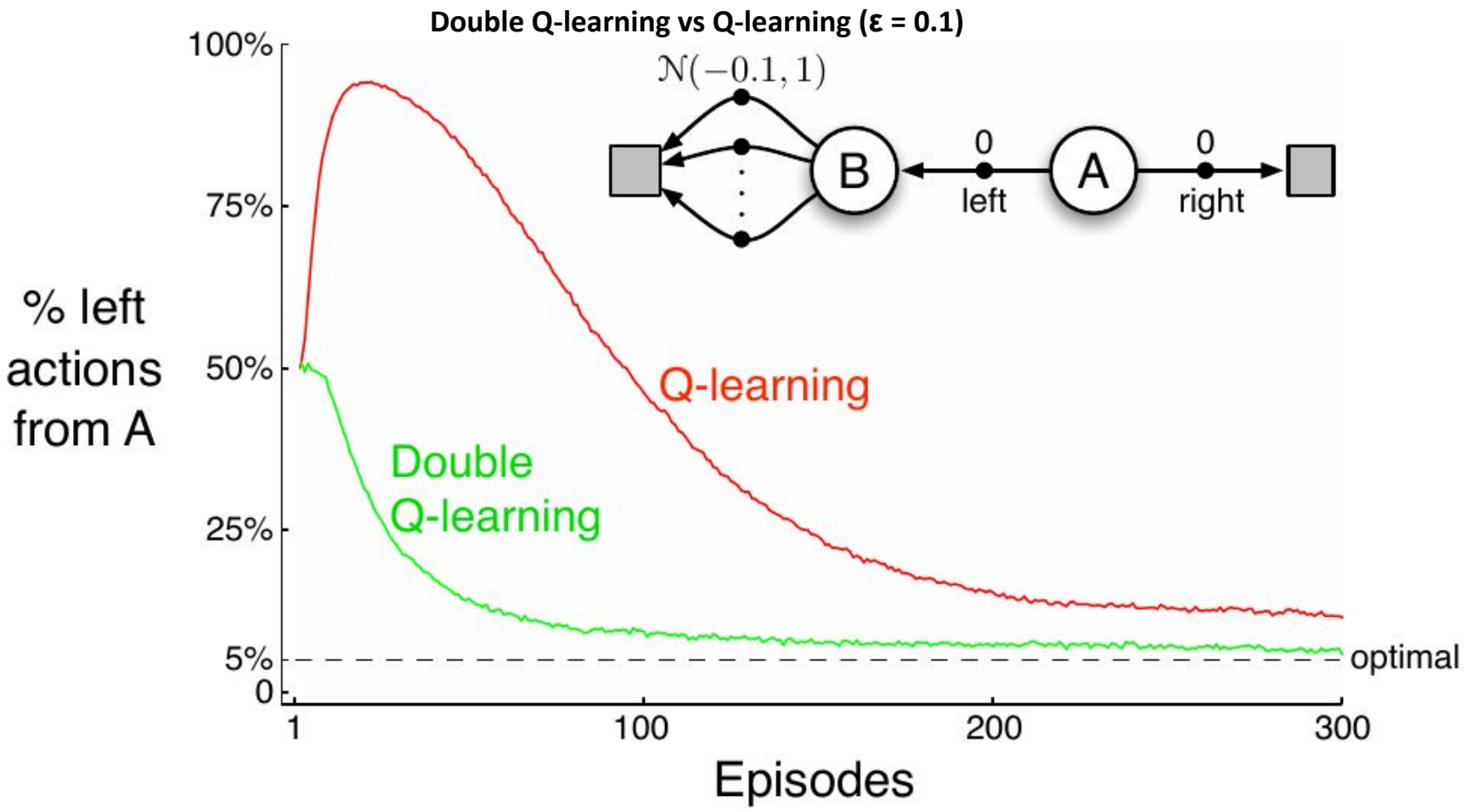
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

Double Q-learning



Towards Function Approximation

Disadvantages of Tabular representation

- Issues with large state/action spaces:
  - not memory efficient
  - data sparsity
  - continuous state/action spaces
  - **Generalisation**

Disadvantages of Tabular representation

- Issues with large state/action spaces:
  - not memory efficient
  - data sparsity
  - continuous state/action spaces
  - **Generalisation**

Idea: Use a parameterized representation (eg. neural networks)

Value Function Approximation

- Least squares:

$$w_{t+1} = w_t - \frac{1}{2}\alpha \nabla_{w_t}\left[q_*(s_t, a_t) - Q(s_t, a_t)\right]^2$$

- But we don't know the target!
- Use the TD *target.*

$$w_{t+1} = w_t - \frac{1}{2}\alpha \nabla_{w_t}\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]^2$$

Linear Q-learning

$$Q(s_t, a_t) = \phi^T(s_t, a_t) \times w_t$$

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$$

TD Error

$$\nabla_{w_t}\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]^2 = -2\,\delta_t \phi(s_t, a_t)$$

$$w_{t+1} = w_t + \alpha\ \delta_t \phi(s_t, a_t)$$

- Known to converge to close to the RMSE minimizer if the policy is **fixed**
- No such results for Q learning
- No strong results for other complex parameterizations
  - But many successful examples: TD Gammon, Atari,…
- Deep RL!!

Tile and Coarse Coding

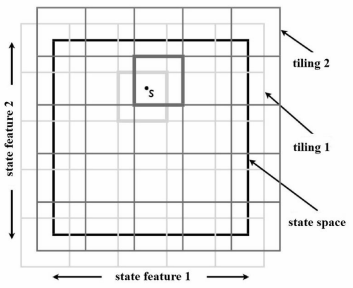
Can exploit the possibility that Q values of nearby states wouldn't change a lot!

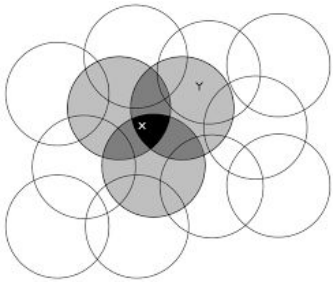
Assume the task is to learn a policy on a big gridworld:

- **Naive method:-**
  - Divide the grid into smaller 10x10 grid.
  - Abrupt change in Q value of states that lie on the boundary.

- **Coarse coding:-**
  - Avoids abrupt changes in the value of the state. Smoothens the Q values during transition from one cluster to another.
- Issue:-
  - No uniformity in the number of 'ON' bits used to represent a state.

- **Tile coding:-**
  - Form of coarse coding but systematic.
  - Number of 'ON' bits == Number of tiles used.





Additional Linear Approximators

CMAC(Cerebellar Model Articulation Controller):

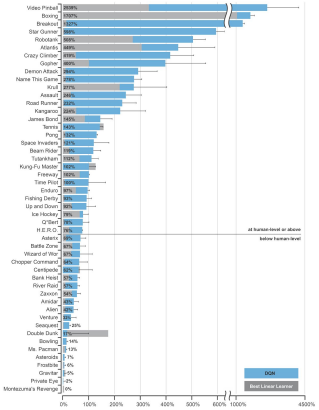
- A form of coarse coding.
- Has a value of 1 inside of k square regions and 0 elsewhere.
- Hash function is used to make sure that the k squares are randomly scattered.
- Implemented in such a way there are exactly c different functions which will be active for any given input.
- The hash function makes generalization even better.
- Typically used in places where the input is high dimensional.

Radial Basis Function:

- Output of radial basis function depends on the distance between input and some fixed point c.
- Sum of many radial basis functions can be used to approximate Q(s,a).

Non-Linear Function Approximator

- Linear function approximators are very restrictive. Can only model linear functions. Basis expansion does help to generate non-linear functions in the original input space.
- Non-linear approximators can model complex functions and are very powerful.
- The features are learnt on the fly and are not hard-coded as is the case with tile and sparse coding.
- Can generalize to unseen states.



Disadvantage:-

Requires a lot of data and compute.