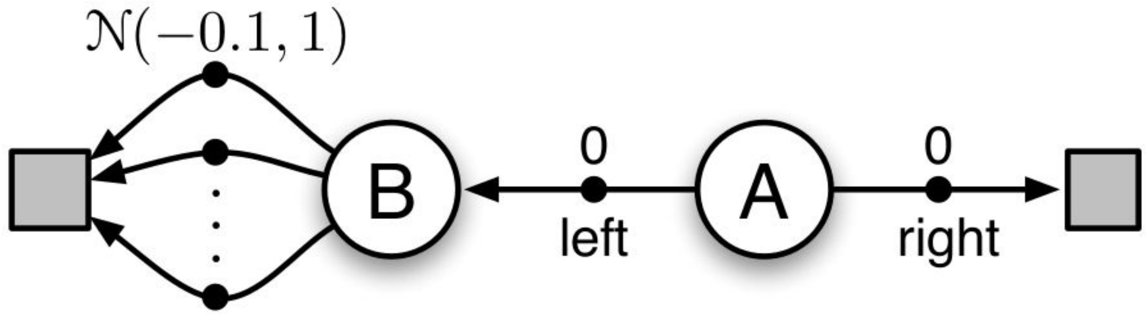
Lecture 6: Maximization Bias and State Aggregation

B. Ravindran

Maximization Bias

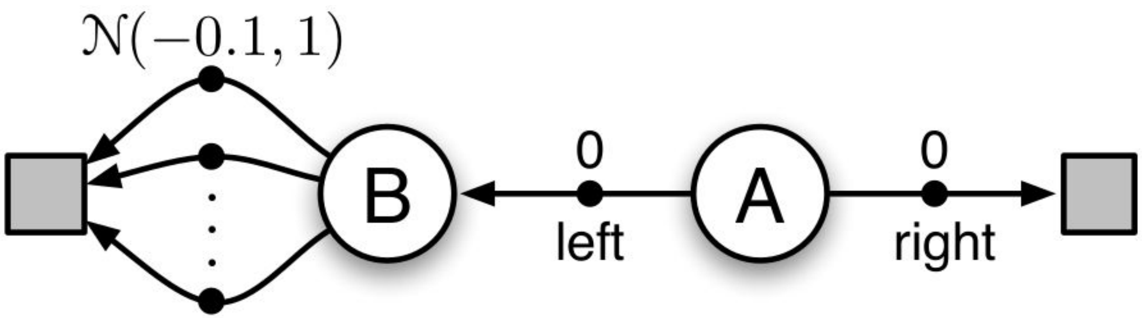
- Consider the simple example below:



- 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?

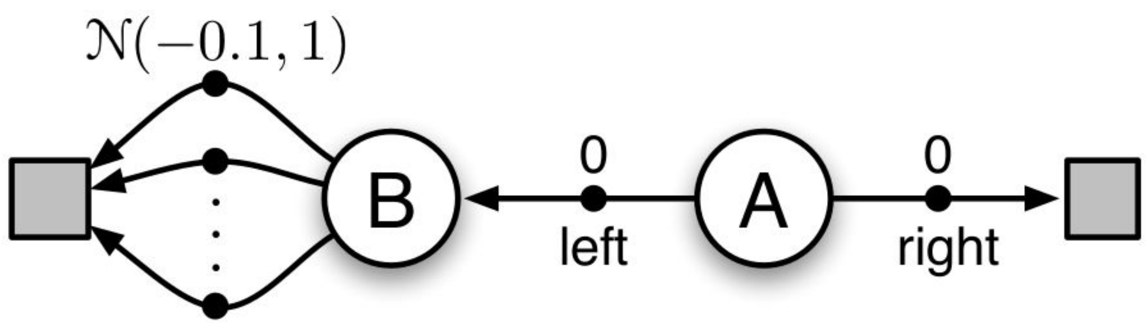


$E[G_t \,|s_0 = A, a_0 = \text{left}] = - 0.1$

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

Maximization Bias

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

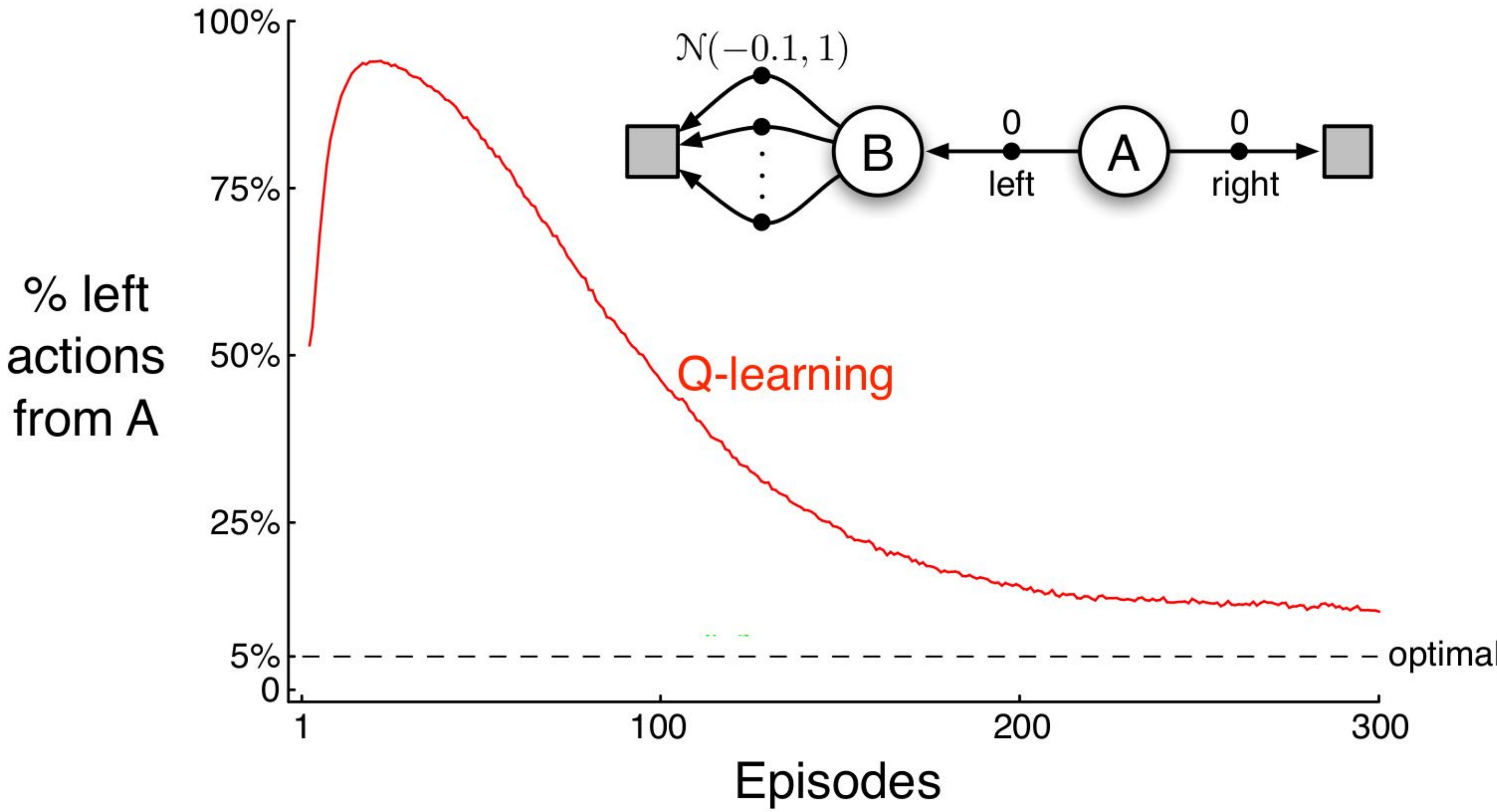


$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

Using maximum over estimate as an estimate of the maximum!

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

Maximization Bias



- ε = 0.1, 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 to determine both the maximizing action and to estimate its value

- Solution:

  Use different estimates to determine the maximizing action and to estimate its value

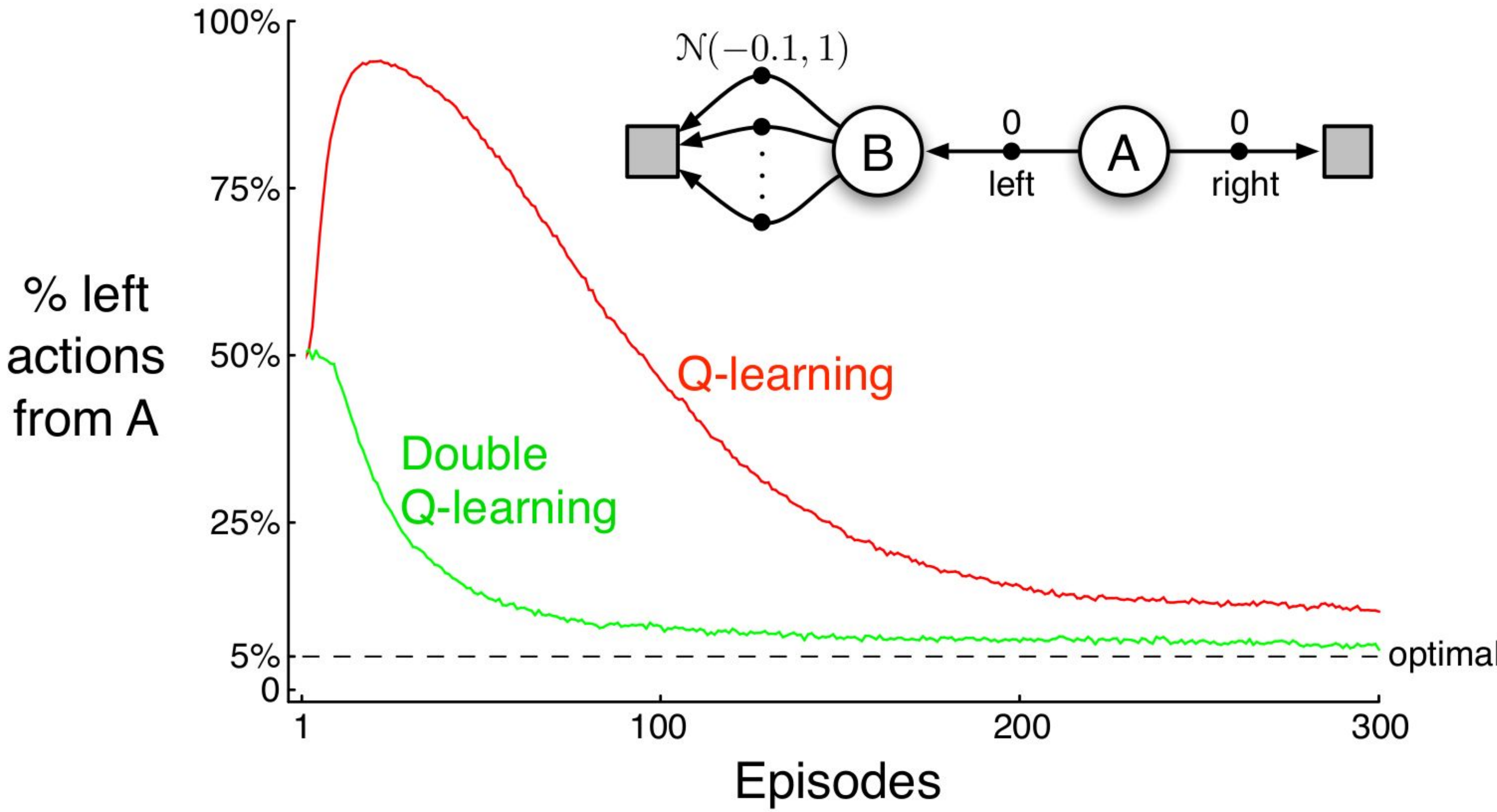
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q_2 \left(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a) \right) - Q_1(S_t, A_t) \right]$$

Double Q-learning

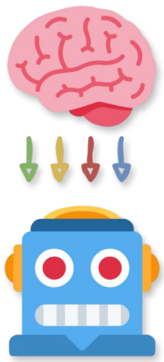
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

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:
$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$
    else:
$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$
    $S \leftarrow S'$
  until S is terminal

Double Q-learning



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



State Aggregation

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: $Q(s_t, a_t) = f(s_t, a_t; w_t)$

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

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

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

Linear Q-learning

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

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

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

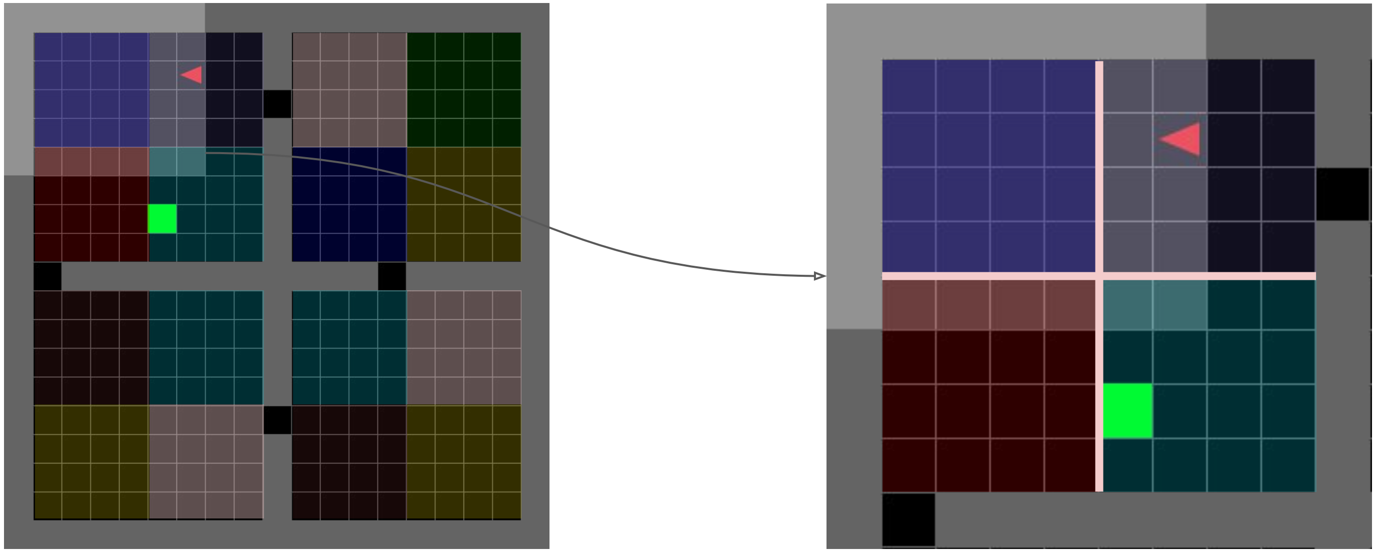
$$w_{t+1} = w_t + \alpha \delta_t \phi(s_t, a_t)$$

- 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 !

Coarse coding

- ❑ Exploit the possibility that q values of nearby states are similar
- ❑ Assume the task is to learn a policy on a big gridworld

- ❑ Naive method
  - ❑ Divide the grid into smaller grids
  - ❑ Abrupt change in Q-value of states that lie on the boundary

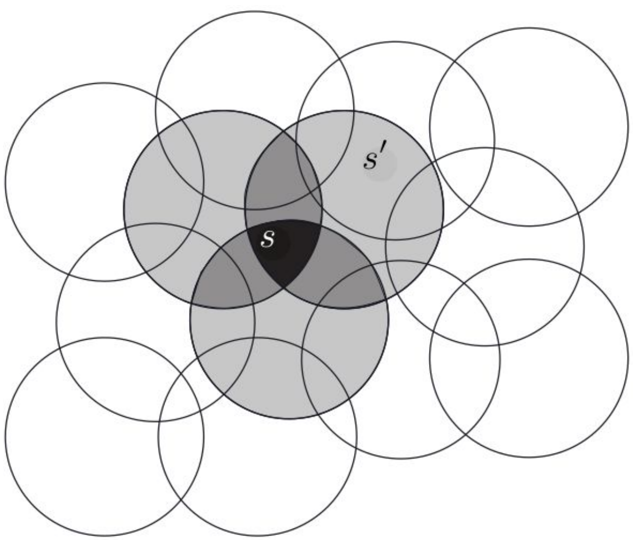


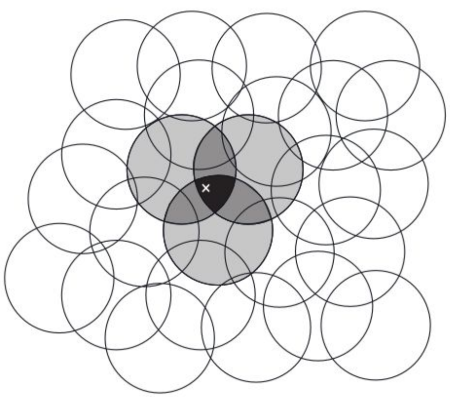
Coarse coding

- Each circle corresponds to a feature $s = [f_1, f_2, \ldots, f_N]$
- If the state is inside a circle, then the feature has value 1 and is *present* ; otherwise feature is 0 and is said to be *absent*

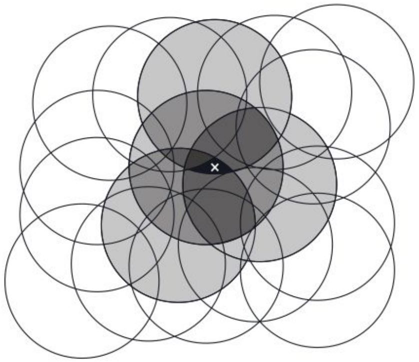
- Generalization from state s to state s' depends on the number of overlapping features

- No uniformity in the number of 'ON' bits used to represent a state





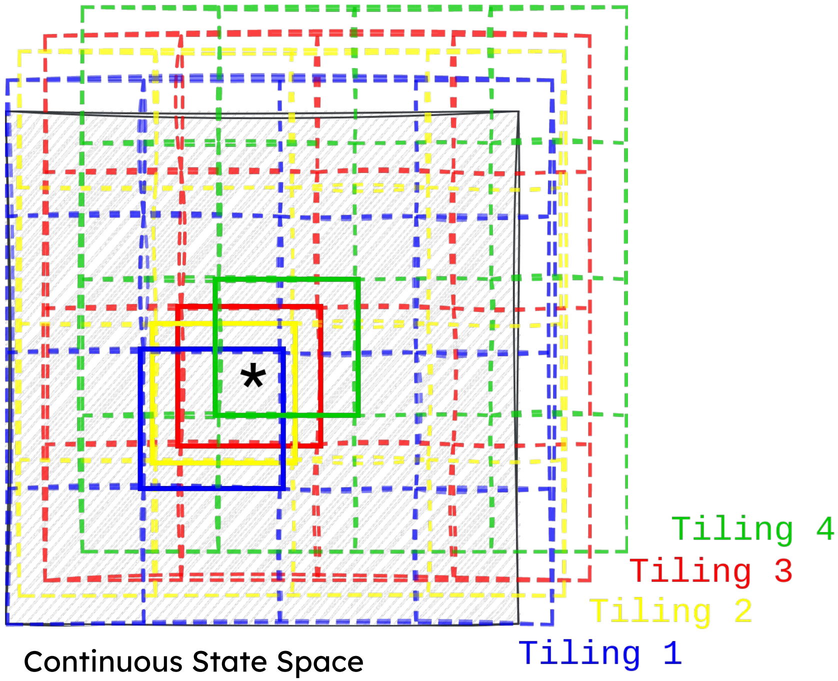
Narrow generalization

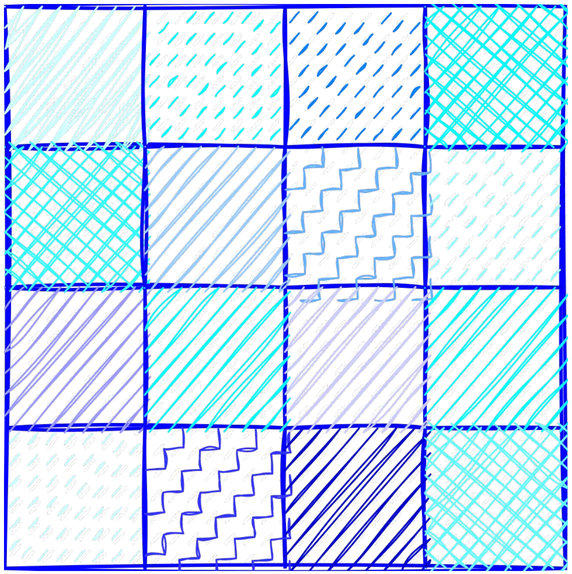


Broad generalization

Tile coding

- ❑ Form of coarse coding but systematic
- ❑ Number of 'ON' bits == Number of tiles used
- ❑ Number of features that are active at one time is the same for any state
- ❑ Easier hyper-parameter tuning





Tiling - 4 x 4 tiles

Additional Linear Approximators

- CMAC (Cerebellar Model Articulation Controller)
  - A form of coarse coding - James Albus in 1975
  - Has a value of 1 inside of k square regions and 0 elsewhere
  - Hash function makes sure that k squares are randomly scattered
  - 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 & 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
- Requires a lot of data and compute