RL with Linear Value Function Approximation

Roberto Capobianco

Sapienza Università di Roma

Recap

n-step Returns

---

n-step returns make using of temporal difference, but use more than one reward

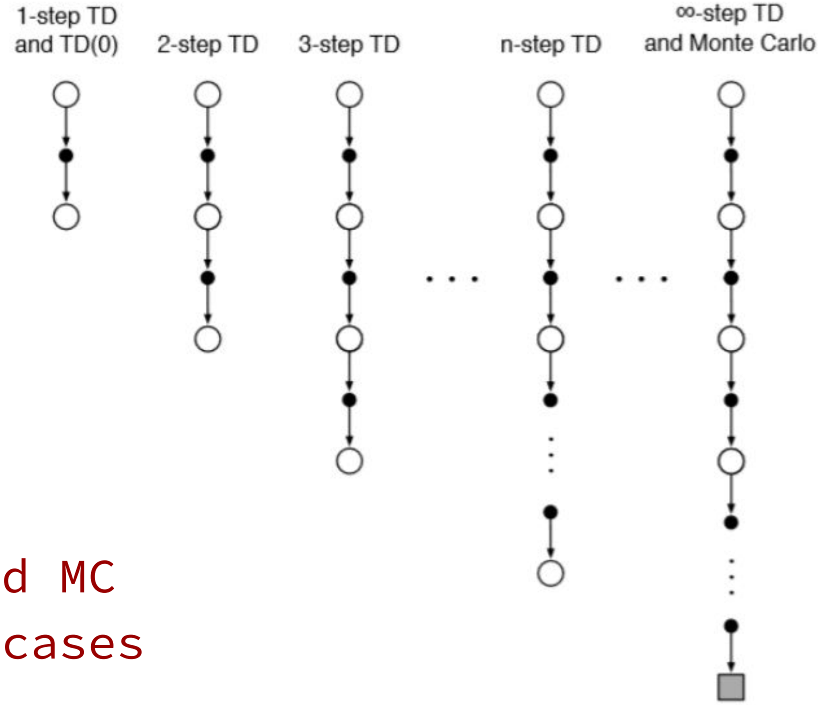
$n = 1 \quad y = G_t^{(1)} = r_t + \gamma V(s_{t+1})$

$n = 2 \quad y = G_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$

...

$n = \infty \quad y = G_t = y_{MC}$

1-step TD and MC just extreme cases



n-step TD in V

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

λ-returns

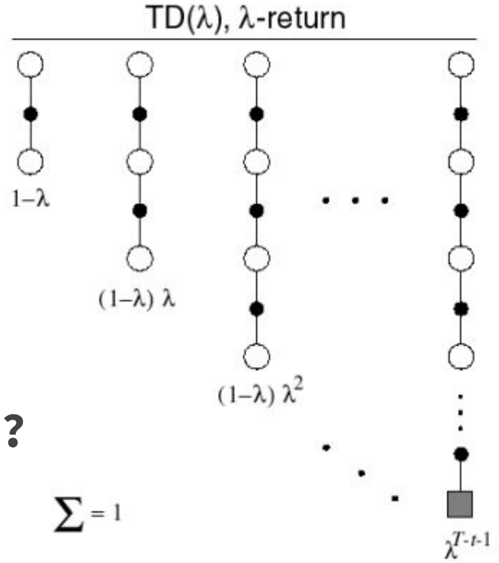
how do we pick n? Why should we commit to a specific n?

Can we combine information from all timesteps?

Yes, λ-returns G_t^λ combine all n-step returns

- using a certain weight $(1-\lambda)\lambda^{n-1}$
- doing a weighted average

$$y = G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$



Forward-View TD(λ)

We can combine all n-steps using λ-returns G_t^λ but we are back at the problem of MC updates: we need to wait for termination

Forward-View TD(λ)

We can combine all n-steps using λ-returns G_t^λ but we are back at the problem of MC updates: we need to wait for termination

We can actually do better and still preserve the updates at every timestep by maintaining a virtually equivalent intuition under a different perspective: backward-view

We will use **eligibility traces**

Eligibility Traces

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

TD error (in V): $\delta_t = r_t + \gamma V^{\wedge\pi}(s_{t+1}) - V^{\wedge\pi}(s_t)$

- for **ALL** the states s in S update the value function estimate

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha\delta_t e_t(s)$$

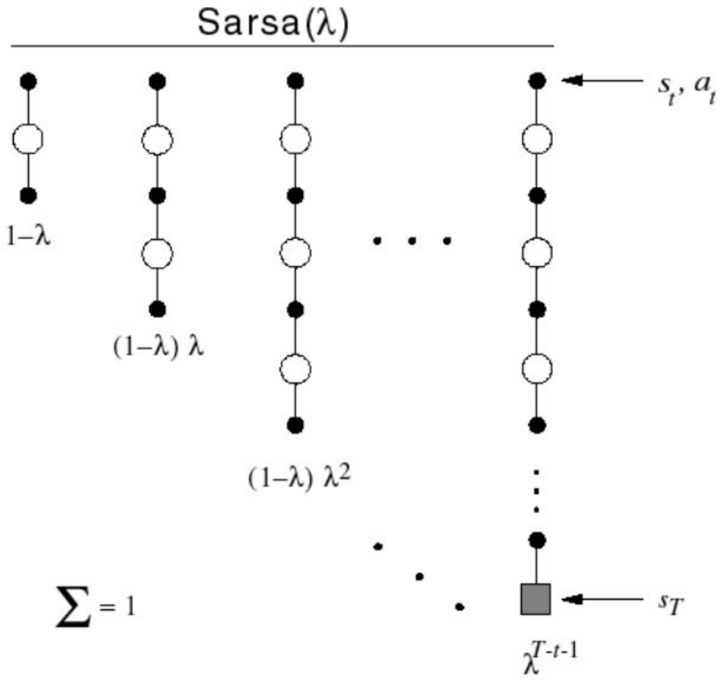
Forward & Backward View Equivalence

Theorem: The sum of offline updates is identical for forward-view and backward-view

$$\sum_{t=0}^{T-1} \alpha \delta_t e_t(s) = \sum_{t=0}^{T-1} \alpha (G_t^{\lambda} - V^{\wedge\pi}(s_t)) \mathbf{I}(s_t = s)$$

- **offline:** apply update in batch at the end of the episode
- **online:** apply update at every timestep

Sarsa-λ: Forward View



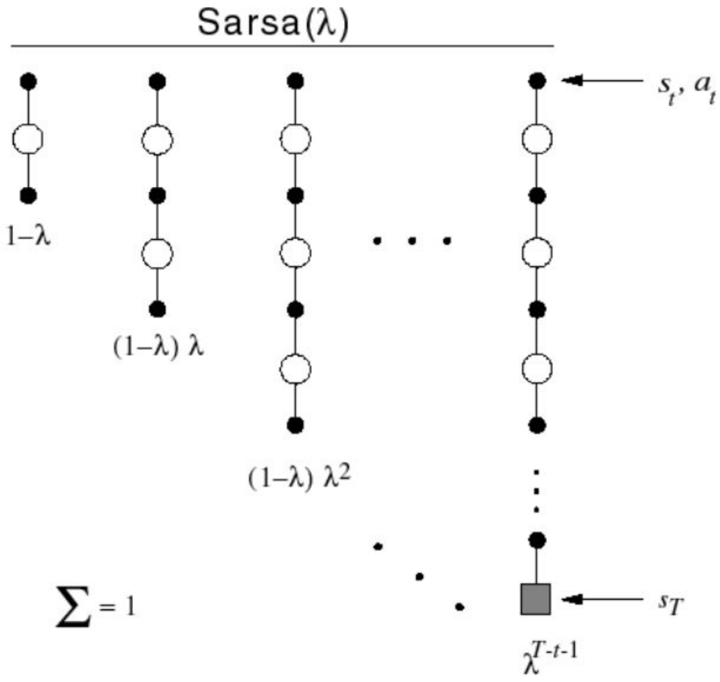
Forward view:

$y = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}Q_{(n)}^{\wedge\pi}$

$Q_{(n)}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1}r_{t+n-1} + \gamma^n Q^{\wedge\pi}(s_{t+n}, a\sim\pi(s_{t+n}))$

$Q(s,a) \leftarrow Q(s,a) + \alpha(y - Q(s,a))$

Sarsa-λ: Backward View



Backward view:

$e_0(s,a) = 0$ and $e_t(s,a)=\gamma e_{t-1}(s,a)+\mathbf{I}(s_t=s,a_t=a)$ for all s,a

Update for all s,a every time:

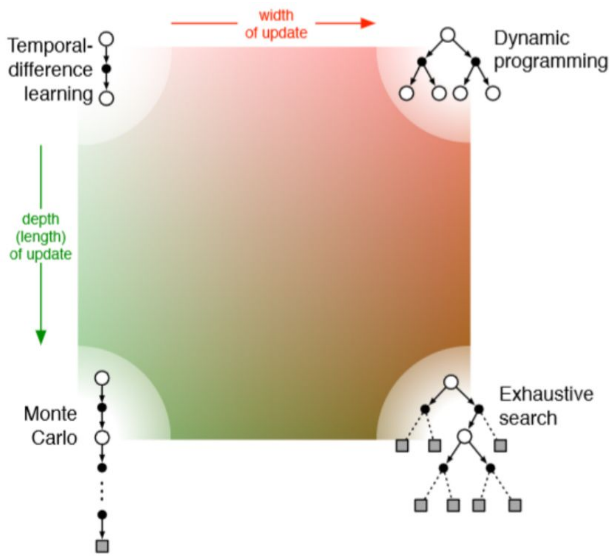
$\delta_t = r_t + \gamma Q^{\wedge\pi}(s_{t+1},a \sim \pi(s_{t+1})) - Q^{\wedge\pi}(s_t,a_t)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t e_t(s,a)$

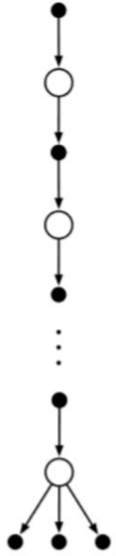
Going Wide vs Going Deep

So far we analyzed the depth of our updates, by considering the amount of sampling, which is:

- cheap computationally
- affected by sampling error
- easy to collect directly from the environment



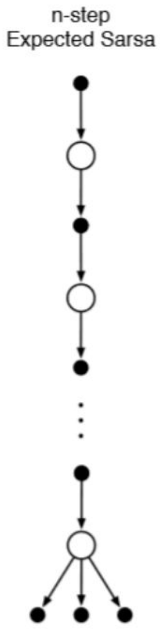
n-step Expected Sarsa



Just compute the target using n-steps and an expectation over the policy at the end

$y = Q_{(n)}{}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$

n-step Expected Sarsa



Just compute the target using n-steps and an expectation over the policy at the end

$$y = Q_{(n)}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$$

Can we handle the degree of sampling/expectation at each step?

Q(σ)

Use σ_t in [0,1] to denote the degree of sampling at each timestep:

- 1 means full sampling
- 0 means full expectation
- can be a function of the state

At each timestep the TD error is

$$\delta_t = r_t + \gamma(\sigma_t Q^{\wedge\pi}(s_{t+1},a\sim\pi(s_{t+1})) + (1-\sigma_t)\sum_a \pi(a|s_{t+1})Q^{\wedge\pi}(s_{t+1},a)) - Q^{\wedge\pi}(s_t,a_t)$$

End Recap

SAPIENZA
UNIVERSITÀ DI ROMA

The problems of tabular representation

So far, we have represented V and Q with tables. They are good for simple tasks, but they are affected by:

- High memory
- Require much time and data
- Lack in generalization

The problems of tabular representation

So far, we have represented V and Q with tables. They are good for simple tasks, but they are affected by:

- High memory
- Require much time and data
- Lack in generalization

What if we approximate them with any functions?

Value function approximators

Instead of using tables, we will represent V and Q with parametrized functions

$$V(s, \mathbf{w}) \approx V^{\pi}(s)$$
$$(\text{or } Q(s, a, \mathbf{w}) \approx Q^{\pi}(s, a))$$

where $V(s, \mathbf{w})$ can be a **linear combination of features**, a decision tree, K-nearest neighbor, a neural network.

How do we optimize V?

We iteratively optimize the approx. value function, minimizing the mean squared value error

$$\text{MSVE}(\mathbf{w}) = \Sigma_s d(s)[Q(s,a) - Q_\pi(s,a,\mathbf{w})]^2$$

where d: S -> [0,1], such that $\Sigma_s d(s)=1$,
is a distribution over the states

We seek for $\mathbf{w}^*$ s.t. $\text{RMSE}(\mathbf{w}^*) <= \text{RMSE}(\mathbf{w})$ for all w

How do we optimize V?

We iteratively optimize the approx. value function, minimizing the mean squared value error

$$MSVE(\mathbf{w}) = \Sigma_s d(s)[Q(s,a) - Q_\pi(s,a,\mathbf{w})]^2$$

where d: S -> [0,1], such that $\Sigma_s d(s)=1$,
is a distribution over the states

We seek for $\mathbf{w}^*$ s.t. $RMSE(\mathbf{w}^*) <= RMSE(\mathbf{w})$ for all w

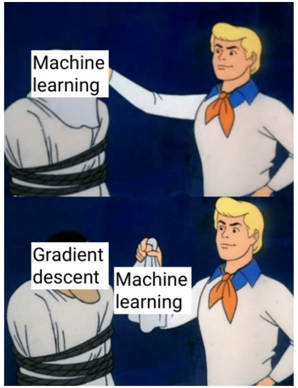
Cool, but how?

SAPIENZA
UNIVERSITÀ DI ROMA

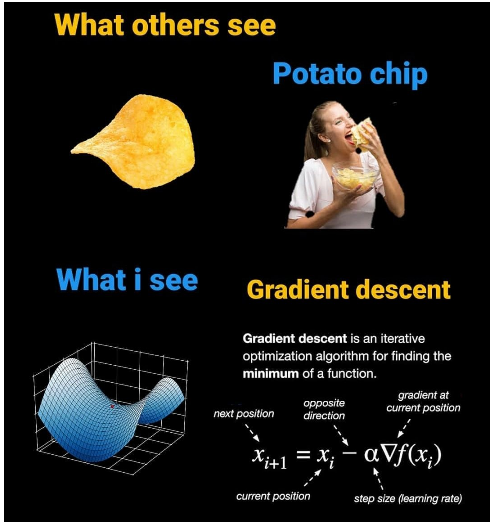
Optimization through Gradient Descent

Gradient Descent optimize a function f(x,**w**) by minimizing an error J(**w**). This is done by iteratively update w in the following way:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}_t)$$



Machine learning behind the scenes



SAPIENZA
UNIVERSITÀ DI ROMA

Optimization through Gradient Descent

Gradient Descent optimize a function $f(x,\mathbf{w})$ by minimizing an error $J(\mathbf{w})$. This is done by iteratively update w in the following way:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}_t)$$

What about J?

Optimization through Gradient Descent

Gradient Descent optimize a function f(x,**w**) by minimizing an error J(**w**). This is done by iteratively update w in the following way:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}_t)$$

Using the mean squared error (MSE):

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_w [½(f^* - f(x_t,\mathbf{w}_t))^2] =$$

$$= \mathbf{w}_t - \alpha(f^* - f(x_t,\mathbf{w}_t)) \nabla_w f(x_t,\mathbf{w}_t)$$

Optimization through Gradient Descent

Remembering that our function f is $Q(s,a,\mathbf{w})$ and that our f^* is Q^π we obtain:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}}[½(Q^\pi_t - Q(s_t,a_t,\mathbf{w}_t))^2] =$$

$$= \mathbf{w}_t - \alpha(Q^\pi_t - Q(s_t,a_t,\mathbf{w}_t)) \nabla_w Q(s_t,a_t,\mathbf{w}_t)$$

Ok cool again, but what about $Q(s_t,a_t,w_t)$?

Sapienza Università di Roma

Linear Combination State-Action Value Approx.

If we represent our state with a feature vector

$$\mathbf{x}(s,a) = (x_1(s,a), \ldots, x_n(s,a))^T$$

And consider a parametric Q function in the form

$$Q(s,a,\mathbf{w}) = \mathbf{w}^T \mathbf{x}(s,a)$$

Then

$$J(w_t) = \mathbb{E}(Q^\pi(s_t,a_t) - \mathbf{w}^T \mathbf{x}(s_t,a_t))^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(Q^\pi{}_t - \mathbf{w}^T{}_t \mathbf{x}(s_t,a_t))\ \mathbf{x}(s_t,a_t)$$

Linear Combination State-Action Value Approx.

If we represent our state with a feature vector

$$\mathbf{x}(s,a) = (x_1(s,a), \dots, x_n(s,a))^T$$

And consider a parametric Q function in the form

$$Q(s,a,\mathbf{w}) = \mathbf{w}^T \mathbf{x}(s,a)$$

Then

$$J(w_t) = \mathbb{E}(Q^\pi(s_t,a_t) - \mathbf{w}^T \mathbf{x}(s_t,a_t))^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(Q^\pi_t - \mathbf{w}^T_t \mathbf{x}(s_t,a_t))\ \mathbf{x}(s_t,a_t)$$

mind the sign

Incremental Prediction Algorithms (forward-view)

What exactly is the target Q^{π}_{t}?

- For MC, the target is the return G_t
- For TD(0) the TD target is $R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}, \mathbf{w})$
- For forward-view TD(λ), the target is G_t^{λ}

Incremental Prediction Algorithms (backward-view)

For backward-view TD(λ) the update is

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\delta_t e_t$$

$$\delta_t = R_{t+1} + \gamma Q(s_{t+1},a_{t+1},\mathbf{w}) - Q(s_t,a_t,\mathbf{w})$$

$$e_t = \gamma\lambda e_{t-1} + \nabla_w Q(s_t,a_t,\mathbf{w})$$

Batch Methods

Incremental methods use Gradient Descent:

- simple
- not sample efficient (need to calculate it for every sample)

Batch methods seek to find the best fitting value function over a batch of samples (like in supervised learning)

Least Squares

Given a Q function approx. $Q(s,a,\mathbf{w}) \sim Q^{\pi}(s,a)$

And an experience $D = \{(s_1,a_1,Q^{\pi}{}_1),(s_2,a_2,Q^{\pi}{}_2),...,(s_n,a_n,Q^{\pi}{}_n)\}$

The least squares algorithm finds a parameter vector **w** minimizing the error:

$$J(\mathbf{w}) = \Sigma_i \, (Q^{\pi}{}_i - Q(s_i,a_i,\mathbf{w}))^2$$

Least Squares

Given a Q function approx. $Q(s,a,\mathbf{w}) \sim Q^{\pi}(s,a)$

And an experience $D = \{(s_1,a_1,Q^{\pi}{}_1),(s_2,a_2,Q^{\pi}{}_2),...,(s_n,a_n,Q^{\pi}{}_n)\}$

The least squares algorithm finds a parameter vector **w** minimizing the error:

$$J(\mathbf{w}) = \Sigma_i \, (Q^{\pi}{}_i - Q(s_i,a_i,\mathbf{w}))^2$$

with the Stochastic Gradient Descent or with the closed form solution (linear version)

Least Squares Closed Form Solution

In the case of linear value function approx.:

$$Q(s,a,\mathbf{w}) = \mathbf{w}^T \mathbf{x}(s,a)$$

$$\arg\min_{\mathbf{w}} [\Sigma_i\, Q^\pi(s_i,a_i) - \mathbf{w}^T \mathbf{x}(s_i,a_i))^2]$$

$$\mathbf{w} = (X^TX)^{-1}X\, Q^\pi$$

where $(X^TX)^{-1}X$ is the pseudo-inverse of X

Least Squares Closed Form Solution (contd.)

$$\mathbf{w} = (X^T X)^{-1} X\, Q^\pi$$

where $(X^T X)^{-1} X$ is the pseudo-inverse of X

$$X = [x(s_1, a_1) \quad x(s_2, a_2) \quad \dots \quad x(s_n, a_n)]^T$$

$$Q^\pi = [Q^\pi_1 \quad Q^\pi_2 \quad \dots \quad Q^\pi_n]^T$$

SGD with Experience Replay

Given experience $D = \{(s_1,a_1,Q^\pi_1),(s_2,a_2,Q^\pi_2),...,(s_n,a_n,Q^\pi_n)\}$

Repeat:

Sample $(s,a,Q^\pi) \sim D$

Apply SGD update $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha(Q^\pi_t - Q(s,a,\mathbf{w}))\ \nabla_w Q(s,a,\mathbf{w})$

What about the features?

- Feature representation plays an important role
- Most times features should be inspected in combination:

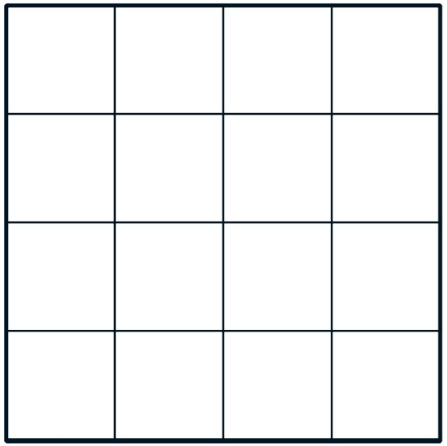
What about the features?

- Feature representation plays an important role

- Most times features should be inspected in combination:

Is the pendulum swinging up or down? The angle and the angular velocity shouldn't be used independently

Tile Coding



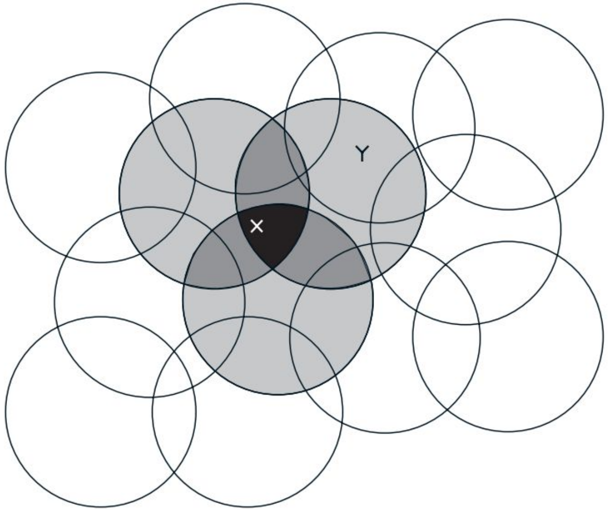
Consider the pendulum task:

- continuous 2-D state space
  (θ and ω)

Tile coding:

- binary feature representation
  that encodes the presence of the current state
  in a discretized space

Coarse Coding



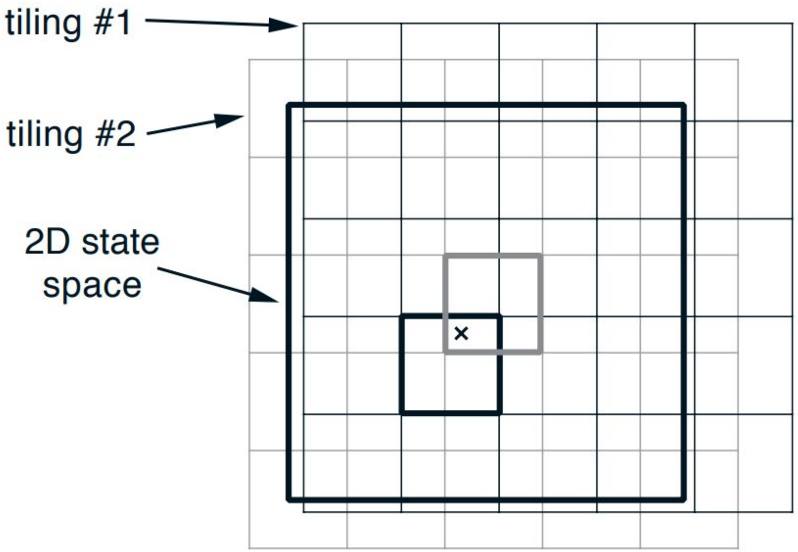
Generalization of tile coding

Coarse coding:

- binary feature representation
  that encodes the presence of the current state
  in the circles

Radial Basis Function

Generalization of coarse coding



RBF:

Discretization is done at multiple resolutions

$$x_i(s) = \exp((-||s-c_i||^2)/(2\sigma^2_i))$$

for different distributions centered in c_i with σ^2_i