# Value Function Approximation

# Large-Scale Reinforcement Learning

- So far we have represented value function by a lookup table
  - Every state s has an entry *V(s)*
  - Or every state-action pair *s*, *a* has an entry *Q(s,a)*

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: $10^{20}$ states
  - Computer Go: $10^{170}$ states
  - Helicopter: continuous state space

- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually

- How can we scale up the model-free methods for prediction and control?

# Value Function Approximation(VFA)

- Function approximation of (action-)state values, if the number of possible states exceeds any reasonable memory capability, $v_\pi(s) = \hat{v}_\pi(s, w)$ with w being a trainable weight vector.
- Solution for large MDPs:
  - Estimate value function with function approximation
    $\hat{v}(s, w) \approx v_\pi(s)$ or $\hat{q}(s, a, w) \approx q_\pi(s, a)$
  - Update parameter $w$ using MC or TD learning
  - Generalize from seen states to unseen states
    - Can estimate value for unseen states
  - Can represent continuous states

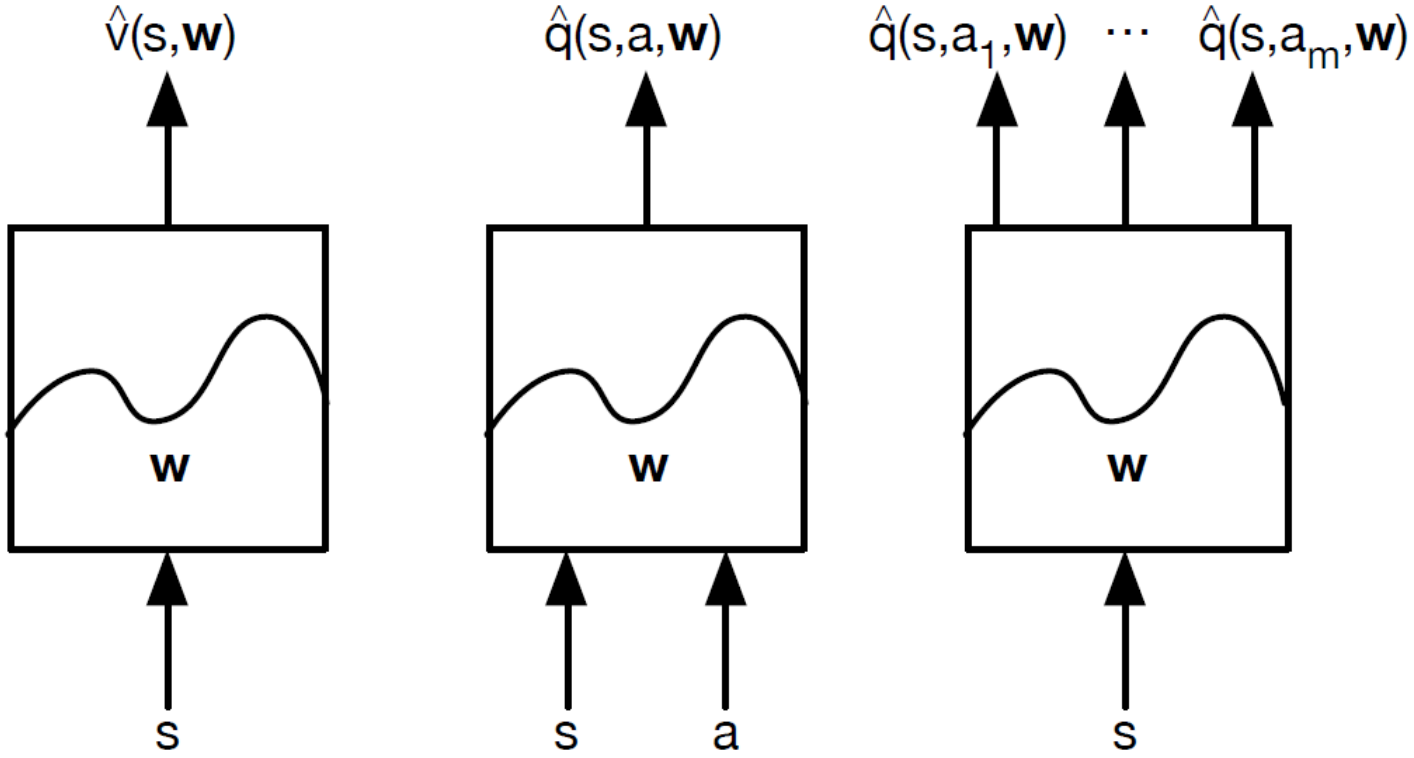
- $Q(s, a) \leftarrow Q(s, a) + \alpha(R + \gamma \max_{a'} Q(s', a') - Q(s, a))$ (no longer available)

# Motivation for Value Function Approximation

- Don't want to have to explicitly store or learn for every
  - State value
  - State-action value
  - Policy (policy gradient)
  - Reward (dynamics) model

- Want more compact representation that generalizes across states and actions
  - Reduce memory needed to store V/Q/$\pi$/R
  - Reduce computation needed to compute V/Q/$\pi$/R
  - Reduce experience needed to find a good V/Q/$\pi$/R

# Types of Value Function Approximation




- Left: one function with single state input & single state value output
- Middle: one function with both states and actions as input
- Right: one function with *i = 1, 2, …:* outputs covering the action space (e.g. ANN with appropriate output layer)

# Which Function Approximator?

- We can consider many function approximators, e.g.
  - Linear combinations of features
  - Neural network
  - Decision tree
  - Nearest neighbor
  - ...
- Requirements of function approximator
  - Should be suitable for *non-stationary(time dependent), non-iid* data
  - Also be differentiable (gradient method)
- Furthermore, we require a training method that is suitable for non-stationary, non-iid data
- Two very popular classes of differentiable function approximator
  - Linear feature representations
  - (Deep) Neural networks

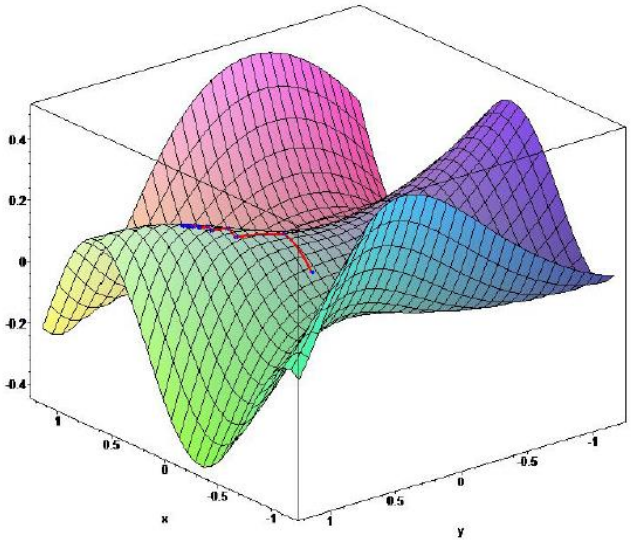
# VFA for Policy Evaluation with an Oracle

- The objective of VFA is to find the best approximate representation of $V_\pi$ given a parameterized function
- First assume we know the true value for $V_\pi(s)$ for any state s
  - Unrealistic assumption
- Can we define a function approximator where $V_\pi(s) \approx approximator(s), \forall s$ ?

# Recap: Gradient Descent

- Let $J(w)$ be a differentiable loss/objective function of parameter vector $w$
- We want to find $w$ values which minimize $J(w)$
- Define the gradient of $J(w)$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

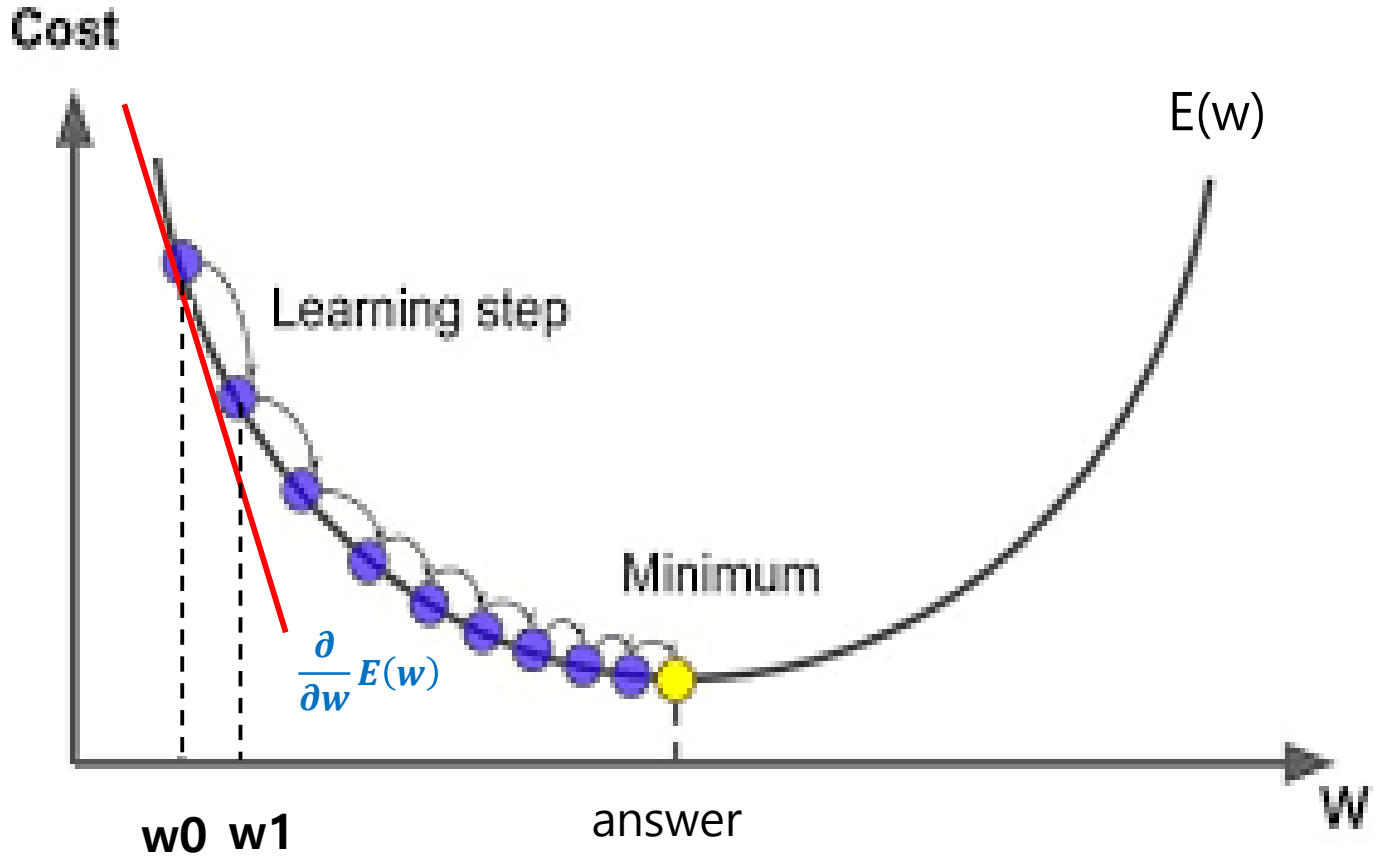


- Gradient Descent Rule: adjust $w$ in direction of gradient
  - find a local minimum of $J(w)$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2}\alpha \nabla_w J(w)$$

where $\alpha$ is a step-size parameter(learning rate)

# Gradient Descent Method




$$w \leftarrow w - \eta \frac{\partial}{\partial w} E(w)$$

# Value Function Approx. Using Gradient Descent

- The Goal now is to find parameter vector $w$ minimizing **mean-squared error** between *approximate value* function $\hat{v}(s,w)$ and *true value* function $v_\pi(s)$

$$J(w) = E_\pi[\big(v_\pi(s) - \hat{v}(s,w)\big)^2]$$

- Now it's optimization problem and, by using gradient descent, we can find a local minimum

$$\begin{aligned}\Delta w &= -\frac{1}{2}\alpha\nabla_w J(w) \\ &= \alpha E_\pi[\big(v_\pi(s) - \hat{v}(s,w)\big)\nabla_w \hat{v}(s,w)]\end{aligned}$$

- Stochastic gradient descent samples the gradient

$$\Delta w = \alpha\big(v_\pi(s) - \hat{v}(s,w)\big)\nabla_w \hat{v}(s,w)$$

# Feature Vectors

- State value or state-action is now represented as a function
- Therefore, each state should be defined as a set features, which becomes the input of function approximator
- First we represent state by a feature vector

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- Examples of feature:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

# Linear Value Function Approximation

- Assume value function can be represented by a linear combination of features
  - State value: linear combination of feature vector($x(s)$) and weight vector($w$)

$$\hat{v}(s,w) = x(s)^T w = \sum_{j=1}^{n} x_j(s) w_j$$

- Loss/objective function($J(w)$) is quadratic in parameters $w$

$$J(w) = E_\pi[\left(v_\pi(s) - x(s)^T w\right)^2]$$

- (Recall) stochastic GD: $\Delta w = \alpha\left(v_\pi(s) - \hat{v}(s,w)\right)\nabla_w \hat{v}(s,w)$
- Update rule is particularly simple. Since $\nabla_w \hat{v}(s,w) = x(s)$

$$\Delta w = \alpha\left(v_\pi(s) - \hat{v}(s,w)\right)x(s)$$

- Update = (step size)*(prediction error)*(feature value)
- Stochastic gradient descent converges on global optimum (since quadratic)

# Model Free VFA Policy Evaluation

- Until now, we assumed that true values of $V_\pi(s)$ are known (unrealistic assumption)
- Don't actually have access to an oracle to tell true $V_\pi(s)$ for any state s
- True values of $V_\pi(s)$ are NOT known now

# Incremental Prediction Algorithms

- Recall: The error/loss function of value function approximation is

$$J(w) = E_\pi[(v_\pi(s) - \hat{v}(s,w))^2]$$

  - Stochastic gradient is

$$\Delta w = \alpha(v_\pi(s) - \hat{v}(s,w))\nabla_w \hat{v}(s,w)$$

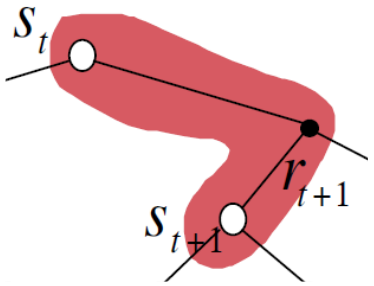
- However, we don't know the true value function $v_\pi(s)$
  - in RL there is no supervisor, only rewards
- In practice, we **substitute a target for $v_\pi(s)$**
  - For MC, the target is the return $G_t$ (actual return)

$$\Delta w = \alpha(G_t - \hat{v}(s,w))\nabla_w \hat{v}(s,w)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(s_{t+1},w)$

$$\Delta w = \alpha(R_{t+1} + \gamma\hat{v}(s_{t+1},w) - \hat{v}(s,w))\nabla_w \hat{v}(s,w)$$

  - semi-gradient method(only $\hat{v}(s,w)$ is differentiated)



# Monte-Carlo with Value Function Approximation

- Stochastic gradient with MC is

$$\Delta w = \alpha\big(G_t - \hat{v}(s,w)\big)\nabla_w \hat{v}(s,w)$$

- Return $G_t$ is an unbiased, noisy sample of true value $v_\pi(s_t)$
- Equivalent to applying supervised learning to training data:

$$< S_1, G_1 >, < S_2, G_2 >, \dots, < S_T, G_T >,$$

  - Find weights to minimize mean squared error:

$$J(w) = E_\pi[\big(G_s - \hat{v}(s,w)\big)^2]$$

- For example, using *linear* Monte-Carlo policy evaluation

$$\Delta w = \alpha\big(G_t - \hat{v}(s,w)\big)\nabla_w \hat{v}(s,w) = \alpha\big(G_t - \hat{v}(s,w)\big)x(s_t)$$

- Monte-Carlo evaluation converges to a local optimum even when using non-linear value function approximation

# TD(0) Learning with Value Function Approximation

- Stochastic gradient with TD(0) learning is

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s, w)\big)\nabla_w \hat{v}(s, w)$$

- The TD-target $R_{t+1} + \gamma\hat{v}(s_{t+1}, w)$ is a biased sample of true value $v_\pi(s_t)$
- Equivalent to applying supervised learning to training data:

$$< s_1, R_2 + \gamma\hat{v}(s_2, w) >, < s_2, R_3 + \gamma\hat{v}(s_3, w) >, \dots, < s_{T-1}, R_T >$$

- Find weights to minimize mean squared error:

$$J(w) = E_\pi[\big(R_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)\big)^2]$$

- For example, using *linear* TD(0)

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(s', w) - \hat{v}(s, w)\big)\nabla_w \hat{v}(s, w) = \alpha\delta x(s)$$

- Linear TD(0) converges (close) to global optimum

# TD($\lambda$) Learning with VFA

- Find weights to minimize mean squared error:

$$J(w) = E_\pi[\left(R_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)\right)^2]$$

- Instead of $R_{t+1} + \gamma\hat{v}(s_{t+1}, w)$, use $\lambda$-return

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

- Truncated $\lambda$-return: $G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t$
- Therefore, $\Delta w = \alpha\left(G_t^\lambda - \hat{v}(s_t, w)\right)\nabla_w \hat{v}(s_t, w)$

# Backward TD($\lambda$) Learning with VFA

- Recap: tabular eligibility trace (for each state)

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s), & if\ s_t \neq s \\ \lambda\gamma e_{t-1}(s) + 1, & if\ s_t = s \end{cases}$$

$$\delta_t = r_{t+1} + \gamma\hat{v}(s_{t+1}) - \hat{v}(s_t)$$

$$\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha\delta_t e_t(s_t)$$

- In VFA, tabular eligibility trace is not possible anymore.
- Instead, eligibility trace is defined for each parameter

- $e_t$: eligibility trace vector (of parameters) at time $t$
  - $\nabla_w \hat{v}(s_t, w)$ indicates how much a change in parameter affects the change in the state value v in a particular state $s_t$

$$e_t = \lambda\gamma e_{t-1} + \nabla_w \hat{v}(s_t, w)$$

$$\delta_t = r_{t+1} + \gamma\hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)$$

$$w \leftarrow w + \alpha\delta_t e_t$$

# Backward TD($\lambda$) Learning with VFA

**Backward TD(λ) learning**

initialize w
**repeat** (every episode)
  $e \leftarrow 0$
  initialize s
  **repeat** (every step in episode)
    perform action a in state s using policy $\pi$
    observe reward r and next state $s'$
    $\delta \leftarrow r + \gamma \hat{v}(s', w) - \hat{v}(s, w)$
    $e \leftarrow \gamma \lambda e + \nabla_w \hat{v}(s, w)$
    $w \leftarrow w + \alpha \delta_t e$
    $s \leftarrow s'$
  **until** s′ is final state
**end**

# Model-Free Control with VFA

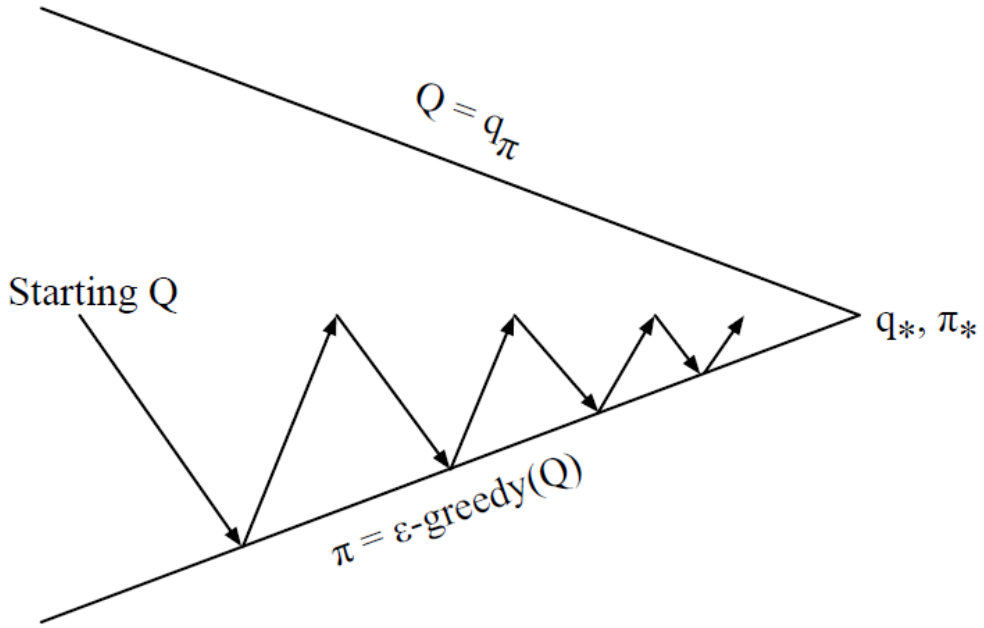
- Again, in model-free control, we need Q values, not V values.
  - Model-free: we don't know the value of $R_s^a, P_{ss'}^a$
  - $\pi'(s) = \text{argmax}_{a \in A} R_s^a + P_{ss'}^a V(s')$ (impossible)
  - $\pi'(s) = \text{argmax}_{a \in A}\ Q(s,a)$ (possible)

- Use value function approximation to represent state-action values

$$\hat{Q}_\pi(s,a,w) \approx Q_\pi$$

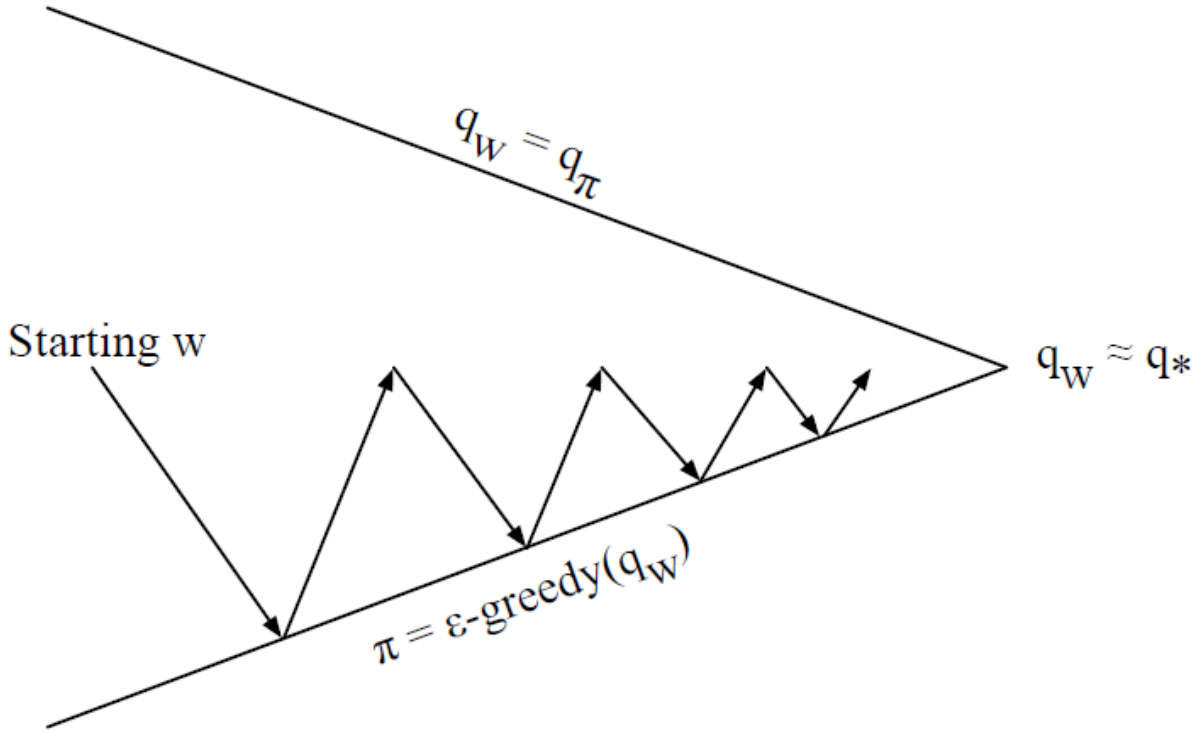
- Interleave
  - Approximate policy evaluation using value function approximation
  - Perform $\epsilon$-greedy policy improvement

# Recap: Monte-Carlo Control




- Policy evaluation: Monte-Carlo policy evaluation, $Q \approx q_\pi$
- Policy improvement: $\epsilon$-greedy policy improvement

# Model-Free Control with VFA




- Policy evaluation: Approximate policy evaluation, $\hat{q}(\cdot,\cdot,w) \approx q_\pi$
  - don't need to evaluate entire q value since we use approximate values
- Policy improvement: $\epsilon$-greedy policy improvement

# Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(s,a,w) \approx q_\pi(s,a)$$

- Minimize mean-squared error between approximate action-value function $\hat{q}(s,a,w)$ and true action-value function $q_\pi(s,a)$

$$J(w) = E_\pi[(q_\pi(s,a) - \hat{q}(s,a,w))^2]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_w J(w) = (q_\pi(s,a) - \hat{q}(s,a,w))\nabla_w \hat{q}(s,a,w)$$
$$\Delta w = \alpha(q_\pi(s,a) - \hat{q}(s,a,w))\nabla_w \hat{q}(s,a,w)$$

# Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$\mathbf{x}(S,A) = \begin{pmatrix} \mathbf{x}_1(S,A) \\ \vdots \\ \mathbf{x}_n(S,A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(s,a,w) = x(s,a)^T w = \sum_{j=1}^{n} x_j(s,a) w_j$$

- Stochastic gradient descent update

$$\text{Since } \nabla_w \hat{q}(s,a,w) = x(s,a)$$

$$\Delta w = \alpha\big(q_\pi(s,a) - \hat{q}(s,a,w)\big)x(s,a)$$

# Incremental Control Algorithms

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- Therefore, like prediction, we must substitute a target $q_\pi(s,a)$.

- From $\Delta w = \alpha(q_\pi(s,a) - \hat{q}(s,a,w))\nabla_w \hat{q}(s,a,w)$
  - For MC, the target is the return $G_t$

$$\Delta w = \alpha\big(G_t - \hat{q}(s_t,a_t,w)\big)\nabla_w \hat{q}(s_t,a_t,w)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1},w)$

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1},w) - \hat{q}(s_t,a_t,w)\big)\nabla_w \hat{q}(s_t,a_t,w)$$