Learning Summary

Given a task, use

- data/experience
- bias/background knowledge
- measure of improvement or error

to improve performance on the task.
Learning Summary

Given a task, use
- data/experience
- bias/background knowledge
- measure of improvement or error
to improve performance on the task.

Representations for:
- Data (e.g., discrete values, indicator functions)
- Models (e.g., decision trees, linear functions, linear separators)
Learning Summary

- Given a task, use
  - data/experience
  - bias/background knowledge
  - measure of improvement or error

  to improve performance on the task.

- Representations for:
  - Data (e.g., discrete values, indicator functions)
  - Models (e.g., decision trees, linear functions, linear separators)

- A way to handle overfitting (e.g., trade-off model complexity and fit-to-data, cross validation).
Learning Summary

- Given a task, use
  - data/experience
  - bias/background knowledge
  - measure of improvement or error

to improve performance on the task.

- Representations for:
  - Data (e.g., discrete values, indicator functions)
  - Models (e.g., decision trees, linear functions, linear separators)

- A way to handle overfitting (e.g., trade-off model complexity and fit-to-data, cross validation).

- Search algorithm (usually local, myopic search) to find the best model that fits the data given the bias.
At the end of the class you should be able to:

- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning
Learning Objectives - Reinforcement Learning

At the end of the class you should be able to:

- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning
- Implement basic state-based reinforcement learning algorithms: Q-learning and SARSA
Learning Objectives - Reinforcement Learning

At the end of the class you should be able to:

- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning
- Implement basic state-based reinforcement learning algorithms: Q-learning and SARSA
- Explain the explore-exploit dilemma and solutions
At the end of the class you should be able to:

- Explain the relationship between decision-theoretic planning (MDPs) and reinforcement learning
- Implement basic state-based reinforcement learning algorithms: Q-learning and SARSA
- Explain the explore-exploit dilemma and solutions
- Explain the difference between on-policy and off-policy reinforcement learning
Reinforcement Learning

What should an agent do given:

- Prior knowledge
- Possible states of the world
- Possible actions
- Observations: current state of world
- Immediate reward / punishment
- Goal: act to maximize accumulated (discounted) reward

Like decision-theoretic planning, except:

- Model of dynamics and model of reward not given.
Reinforcement Learning

What should an agent do given:

- Prior knowledge
Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world, possible actions
- **Observations**

Like decision-theoretic planning, except model of dynamics and model of reward not given.
Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world possible actions
- **Observations** current state of world immediate reward / punishment
- **Goal**
Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world possible actions
- **Observations** current state of world immediate reward / punishment
- **Goal** act to maximize accumulated (discounted) reward
Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world possible actions
- **Observations** current state of world immediate reward / punishment
- **Goal** act to maximize accumulated (discounted) reward

Like decision-theoretic planning, except
Reinforcement Learning

What should an agent do given:

- **Prior knowledge** possible states of the world possible actions
- **Observations** current state of world immediate reward / punishment
- **Goal** act to maximize accumulated (discounted) reward

Like decision-theoretic planning, except model of dynamics and model of reward not given.
Reinforcement Learning Examples

- Game - reward winning, punish losing

- Dog - reward obedience, punish destructive behavior

- Robot - reward task completion, punish dangerous behavior
Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
- Robot - reward task completion, punish dangerous behavior
Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog -
Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
- Robot -
Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behavior
- Robot - reward task completion, punish dangerous behavior
We assume there is a sequence of experiences:

\[ \text{state, action, reward, state, action, reward, ...} \]
Experiences

- We assume there is a sequence of experiences:

  \[
  \text{state, action, reward, state, action, reward, ...}
  \]

- The agent has to choose its action as a function of its history.
We assume there is a sequence of experiences:

\[ \text{state, action, reward, state, action, reward, ...} \]

The agent has to choose its action as a function of its history.

At any time it must decide whether to
We assume there is a sequence of experiences:

\[ \text{state, action, reward, state, action, reward, ...} \]

The agent has to choose its action as a function of its history.

At any time it must decide whether to

- explore to gain more knowledge
- exploit knowledge it has already discovered
Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
- The long-term effect of an action depend on what the agent will do in the future.
Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
- The long-term effect of an action depend on what the agent will do in the future.
- The explore-exploit dilemma: at each time should the agent be greedy or inquisitive?
Reinforcement learning: main approaches

- search through a space of policies (controllers)
Reinforcement learning: main approaches

- search through a space of policies (controllers)
- learn a model consisting of state transition function $P(s'|a,s)$ and reward function $R(s,a,s')$; solve this as an MDP.
Reinforcement learning: main approaches

- search through a space of policies (controllers)
- learn a model consisting of state transition function $P(s'|a, s)$ and reward function $R(s, a, s')$; solve this as an MDP.
- learn $Q^*(s, a)$, use this to guide action.
Recall: Asynchronous VI for MDPs, storing $Q[s, a]$

(If we knew the model:)

Initialize $Q[S, A]$ arbitrarily
Repeat forever:

- Select state $s$, action $a$
- $Q[s, a] := \sum_{s'} P(s'|s, a) \left( R(s, a, s') + \gamma \max_{a'} Q[s', a'] \right)$
Reinforcement Learning (Deterministic case)

- flat or modular or hierarchical
- explicit states or features or individuals and relations
- static or finite stage or indefinite stage or infinite stage
- fully observable or partially observable
- deterministic or stochastic dynamics
- goals or complex preferences
- single agent or multiple agents
- knowledge is given or knowledge is learned
- perfect rationality or bounded rationality
Asynchronous VI for Deterministic RL

initialize $Q[S, A]$ arbitrarily
observe current state $s$

repeat forever:
  select and carry out an action $a$
  observe reward $r$ and state $s'$
  \[ Q[s, a] := r + \gamma \max_{a'} Q[s', a'] \]

What do we know now?
Asynchronous VI for Deterministic RL

initialize $Q[S, A]$ arbitrarily
observe current state $s$

repeat forever:
  select and carry out an action $a$
  observe reward $r$ and state $s'$
  $Q[s, a] := r + \gamma \max_{a'} Q[s', a']$
  $s := s'$
Reinforcement Learning

- flat or modular or hierarchical
- explicit states or features or individuals and relations
- static or finite stage or indefinite stage or infinite stage
- fully observable or partially observable
- deterministic or stochastic dynamics
- goals or complex preferences
- single agent or multiple agents
- knowledge is given or knowledge is learned
- perfect rationality or bounded rationality
Temporal Differences

- Suppose we have a sequence of values:

  \[ v_1, v_2, v_3, \ldots \]

  and want a running estimate of the average of the first \( k \) values:

  \[
  A_k = \frac{v_1 + \cdots + v_k}{k}
  \]
Temporal Differences (cont)

Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

"TD formula"

Often we use this update with $\alpha$ fixed. We can guarantee convergence to average if $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$.

E.g., $\alpha_k = \frac{10}{9 + k}$ treats more recent experiences more, but converges to average.
Temporal Differences (cont)

Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

$$= \frac{k - 1}{k} A_{k-1} + \frac{1}{k} v_k$$

Let $\alpha_k = \frac{1}{k}$, then

$$A_k =$$
Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

$$= \frac{k-1}{k}A_{k-1} + \frac{1}{k}v_k$$

Let $\alpha_k = \frac{1}{k}$, then

$$A_k = (1 - \alpha_k)A_{k-1} + \alpha_k v_k$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1})$$

“TD formula”
Temporal Differences (cont)

Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

$$= \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k$$

Let $\alpha_k = \frac{1}{k}$, then

$$A_k = (1 - \alpha_k) A_{k-1} + \alpha_k v_k$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1})$$

“TD formula”

Often we use this update with $\alpha$ fixed.
Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

$$= \frac{k - 1}{k} A_{k-1} + \frac{1}{k} v_k$$

Let $\alpha_k = \frac{1}{k}$, then

$$A_k = (1 - \alpha_k) A_{k-1} + \alpha_k v_k$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1})$$

“TD formula”

Often we use this update with $\alpha$ fixed.

We can guarantee convergence to average if
Temporal Differences (cont)

Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

$$= \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k$$

Let $\alpha_k = \frac{1}{k}$, then

$$A_k = (1 - \alpha_k) A_{k-1} + \alpha_k v_k$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1})$$

“TD formula”

Often we use this update with $\alpha$ fixed.

We can guarantee convergence to average if

$$\sum_{k=1}^{\infty} \alpha_k = \infty \text{ and } \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$
Suppose we know $A_{k-1}$ and a new value $v_k$ arrives:

$$A_k = \frac{v_1 + \cdots + v_{k-1} + v_k}{k}$$

$$= \frac{k-1}{k} A_{k-1} + \frac{1}{k} v_k$$

Let $\alpha_k = \frac{1}{k}$, then

$$A_k = (1 - \alpha_k) A_{k-1} + \alpha_k v_k$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1})$$

“TD formula”

Often we use this update with $\alpha$ fixed.

We can guarantee convergence to average if

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty.$$

E.g., $\alpha_k = 10/(9 + k)$ treats more recent experiences more, but converges to average.
Q-learning

- **Idea**: store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
Q-learning

- **Idea:** store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience $\langle s, a, r, s' \rangle$.
- This provides one piece of data to update $Q[s, a]$.
- An experience $\langle s, a, r, s' \rangle$ provides a new estimate for the value of $Q^*(s, a)$:

  $$r + \gamma \max_{a'} Q[s', a']$$

  which can be used in the TD formula giving:

  $$Q[s, a] := Q[s, a] + \alpha (r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$$
Q-learning

- **Idea:** store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience $\langle s, a, r, s' \rangle$
- This provides one piece of data to update $Q[s, a]$.
- An experience $\langle s, a, r, s' \rangle$ provides a new estimate for the value of $Q^*(s, a)$:

$$r + \gamma \max_{a'} Q[s', a']$$

which can be used in the TD formula giving:
Q-learning

- **Idea:** store $Q[State, Action]$; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience $\langle s, a, r, s' \rangle$
- This provides one piece of data to update $Q[s, a]$.
- An experience $\langle s, a, r, s' \rangle$ provides a new estimate for the value of $Q^*(s, a)$:

$$r + \gamma \max_{a'} Q[s', a']$$

which can be used in the TD formula giving:

$$Q[s, a] := Q[s, a] + \alpha \left( r + \gamma \max_{a'} Q[s', a'] - Q[s, a] \right)$$
Q-learning

initialize $Q[S, A]$ arbitrarily
observe current state $s$

repeat forever:

select and carry out an action $a$
observe reward $r$ and state $s'$

$Q[s, a] := Q[s, a] + \alpha (r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

$s := s'$
Properties of Q-learning

- Q-learning converges to an optimal policy, no matter what the agent does, as long as it tries each action in each state enough.
- But what should the agent do?
  - exploit: when in state $s$,
  - explore:
Properties of Q-learning

- Q-learning converges to an optimal policy, no matter what the agent does, as long as it tries each action in each state enough.

- But what should the agent do?
  - exploit: when in state $s$, select an action that maximizes $Q[s, a]$
  - explore: select another action
The $\epsilon$-greedy strategy: choose random action with probability $\epsilon$ & choose a best action with probability $1 - \epsilon$. 

"optimism in the face of uncertainty": initialize $Q$ to values that encourage exploration.

"upper confidence bounds" - take into account average + variance.
Exploration Strategies

- The $\epsilon$-greedy strategy: choose random action with probability $\epsilon$ & choose a best action with probability $1 - \epsilon$.
- Softmax action selection: in state $s$, choose $a$ with probability
  
  $$\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$

  where $\tau > 0$ is the temperature.
Exploration Strategies

- The $\epsilon$-greedy strategy: choose random action with probability $\epsilon$ & choose a best action with probability $1 - \epsilon$.
- Softmax action selection: in state $s$, choose $a$ with probability

$$
\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}
$$

where $\tau > 0$ is the temperature.
- “optimism in the face of uncertainty”: initialize $Q$ to values that encourage exploration.
Exploration Strategies

- The $\epsilon$-greedy strategy: choose random action with probability $\epsilon$ & choose a best action with probability $1 - \epsilon$.
- Softmax action selection: in state $s$, choose $a$ with probability
  
  \[
  \frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}
  \]

  where $\tau > 0$ is the temperature.
- “optimism in the face of uncertainty”: initialize $Q$ to values that encourage exploration.
- “upper confidence bounds” - take into account average + variance
Exploration Strategies

- The $\epsilon$-greedy strategy: choose random action with probability $\epsilon$ & choose a best action with probability $1 - \epsilon$.
- Softmax action selection: in state $s$, choose $a$ with probability

$$\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}$$

where $\tau > 0$ is the temperature.
- “optimism in the face of uncertainty”: initialize $Q$ to values that encourage exploration.
- “upper confidence bounds” - take into account average + variance
- Maintain a stochastic policy (distribution over actions)
Problems with Q-learning

- It does one backup between each experience.
  - Is this appropriate for a robot interacting with the real world?
Problems with Q-learning

- It does one backup between each experience.
  - Is this appropriate for a robot interacting with the real world?
  - An agent can make better use of the data by...
Problems with Q-learning

- It does one backup between each experience.
  - Is this appropriate for a robot interacting with the real world?
  - An agent can make better use of the data by
    — remember previous experiences and use these to update model (action replay)
    — building a model, and using MDP methods to determine optimal policy.
    — doing multi-step backups

- It learns separately for each state.
On-policy Learning

- Q-learning does **off-policy learning**: it learns the value of an optimal policy, no matter what it does.
- This could be bad if

```latex
\text{SARSA uses the experience } \langle s, a, r, s' \rangle \text{ to update } Q[s, a].
```
Q-learning does off-policy learning: it learns the value of an optimal policy, no matter what it does.

This could be bad if the exploration policy is dangerous.

On-policy learning learns the value of the policy being followed.

e.g., act greedily 80% of the time and act randomly 20% of the time

Why?
On-policy Learning

- Q-learning does **off-policy learning**: it learns the value of an optimal policy, no matter what it does.
- This could be bad if the exploration policy is dangerous.
- **On-policy learning** learns the value of the policy being followed.
  e.g., act greedily 80% of the time and act randomly 20% of the time
- Why? If the agent is actually going to explore, it may be better to optimize the actual policy it is going to do.
- SARSA uses the experience \(\langle s, a, r, s', a' \rangle\) to update \(Q[s, a]\).
initialize $Q[S, A]$ arbitrarily
observe current state $s$
select action $a$
repeat forever:
    carry out action $a$
    observe reward $r$ and state $s'$
    select action $a'$ using a policy based on $Q$
    $Q[s, a] :=$
initialize $Q[S, A]$ arbitrarily
observe current state $s$
select action $a$
repeat forever:
  carry out action $a$
  observe reward $r$ and state $s'$
  select action $a'$ using a policy based on $Q$
  $Q[s, a] := Q[s, a] + \alpha (r + \gamma Q[s', a'] - Q[s, a])$
  $s := s'$
  $a := a'$
Q-learning with Action Replay

initialize $Q[S, A]$ arbitrarily
$E = \{\}$
observe current state $s$
select action $a$

repeat forever:
  carry out action $a$
  observe reward $r$ and state $s'$
  $E := E \cup \{⟨s, a, r, s'⟩\}$
  $Q[s, a] :=$
Q-learning with Action Replay

initialize $Q[S, A]$ arbitrarily
$E = \{\}$
observe current state $s$
select action $a$

repeat forever:

  carry out action $a$
  observe reward $r$ and state $s'$
  $E := E \cup \{\langle s, a, r, s'\rangle\}$
  $Q[s, a] := Q[s, a] + \alpha (r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

repeat for a while:

  select $\langle s_1, a_1, r_1, s'_1\rangle \in E$
  $Q[s_1, a_1] := $
Q-learning with Action Replay

initialize $Q[S, A]$ arbitrarily
$E = \{\}$
observe current state $s$
select action $a$

repeat forever:
  carry out action $a$
  observe reward $r$ and state $s'$
  $E := E \cup \{\langle s, a, r, s'\rangle\}$
  $Q[s, a] := Q[s, a] + \alpha (r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

repeat for a while:
  select $\langle s_1, a_1, r_1, s'_1\rangle \in E$
  $Q[s_1, a_1] := Q[s_1, a_1] + \alpha (r_1 + \gamma \max_{a'_1} Q[s'_1, a'_1] - Q[s_1, a_1])$
  $s := s'$
  $a := a'$
Model-based reinforcement learning uses the experiences in a more effective manner.

It is used when collecting experiences is expensive (e.g., in a robot or an online game); an agent can do lots of computation between each experience.
Model-based reinforcement learning uses the experiences in a more effective manner.

It is used when collecting experiences is expensive (e.g., in a robot or an online game); an agent can do lots of computation between each experience.

Idea: learn the MDP and interleave acting and planning.
Model-based reinforcement learning uses the experiences in a more effective manner.

It is used when collecting experiences is expensive (e.g., in a robot or an online game); an agent can do lots of computation between each experience.

Idea: learn the MDP and interleave acting and planning.

After each experience, update probabilities and the reward, then do some steps of asynchronous value iteration.
Model-based learner


Assign $Q$, $R$ arbitrarily, $C = 0$, $T = 0$

observe current state $s$

repeat forever:
    select and carry out action $a$
    observe reward $r$ and state $s'$
Model-based learner


Assign \( Q, R \) arbitrarily, \( C = 0, T = 0 \)

observe current state \( s \)

repeat forever:

select and carry out action \( a \)

observe reward \( r \) and state \( s' \)

\[
T[s, a, s'] := T[s, a, s'] + 1
\]

\[
C[s, a] := C[s, a] + 1
\]

\[
R[s, a] := R[s, a] + (r - R[s, a])/C[s, a]
\]

What goes wrong with this?
Model-based learner


Assign $Q, R$ arbitrarily, $C = 0, T = 0$

observe current state $s$

repeat forever:

- select and carry out action $a$
- observe reward $r$ and state $s'$

$$T[s, a, s'] := T[s, a, s'] + 1$$

$$C[s, a] := C[s, a] + 1$$

$$R[s, a] := R[s, a] + (r - R[s, a])/C[s, a]$$

repeat for a while:

- select state $s_1$, action $a_1$

$$Q[s_1, a_1] :=$$

What goes wrong with this?

Assign $Q$, $R$ arbitrarily, $C = 0$, $T = 0$

observe current state $s$

**repeat forever:**

select and carry out action $a$

observe reward $r$ and state $s'$

$T[s, a, s'] := T[s, a, s'] + 1$

$C[s, a] := C[s, a] + 1$

$R[s, a] := R[s, a] + (r - R[s, a])/C[s, a]$

**repeat for a while:**

select state $s_1$, action $a_1$

$Q[s_1, a_1] := R[s_1, a_1] + \sum_{s_2} \frac{T[s_1, a_1, s_2]}{C[s_1, a_1]} \left( \gamma \max_{a_2} Q[s_2, a_2] \right)$

$s := s'$
Model-based learner


Assign \( Q, R \) arbitrarily, \( C = 0, T = 0 \)

observe current state \( s \)

repeat forever:

select and carry out action \( a \)

observe reward \( r \) and state \( s' \)

\[
T[s, a, s'] := T[s, a, s'] + 1
\]

\[
C[s, a] := C[s, a] + 1
\]

\[
R[s, a] := R[s, a] + (r - R[s, a])/C[s, a]
\]

repeat for a while:

select state \( s_1 \), action \( a_1 \)

\[
Q[s_1, a_1] := R[s_1, a_1] + \sum_{s_2} \frac{T[s_1, a_1, s_2]}{C[s_1, a_1]} \left( \gamma \max_{a_2} Q[s_2, a_2] \right)
\]

\( s := s' \)

What goes wrong with this?
Usually we don’t want to reason in terms of states, but in terms of features.

In state-based methods, information about one state cannot be used by similar states.

If there are too many parameters to learn, it takes too long.

Idea: Express the value (Q) function as a function of the features. Most typical is a linear function of the features, or a neural network.
Reinforcement Learning

- flat or modular or hierarchical
- explicit states or features or individuals and relations
- static or finite stage or indefinite stage or infinite stage
- fully observable or partially observable
- deterministic or stochastic dynamics
- goals or complex preferences
- single agent or multiple agents
- knowledge is given or knowledge is learned
- perfect rationality or bounded rationality
Review: Gradient descent

To find a (local) minimum of a real-valued function \( f(x) \):

1. assign an arbitrary value to \( x \)
2. repeat

\[
x := x - \eta \frac{df}{dx}
\]

where \( \eta \) is the step size

To find a local minimum of real-valued function \( f(x_1, \ldots, x_n) \):

1. assign arbitrary values to \( x_1, \ldots, x_n \)
2. repeat:
   - for each \( x_i \)
     \[
x_i := x_i - \eta \frac{\partial f}{\partial x_i}
\]
Review: Gradient descent

To find a (local) minimum of a real-valued function $f(x)$:

- assign an arbitrary value to $x$
- repeat

$$x := x - \eta \frac{df}{dx}$$

where $\eta$ is the step size
Review: Gradient descent

To find a (local) minimum of a real-valued function $f(x)$:

- assign an arbitrary value to $x$
- repeat

\[ x := x - \eta \frac{df}{dx} \]

where $\eta$ is the step size

To find a local minimum of real-valued function $f(x_1, \ldots, x_n)$:

- assign arbitrary values to $x_1, \ldots, x_n$
- repeat:
  - for each $x_i$

\[ x_i := \]
Review: Gradient descent

To find a (local) minimum of a real-valued function $f(x)$:

- assign an arbitrary value to $x$
- repeat

$$x := x - \eta \frac{df}{dx}$$

where $\eta$ is the step size

To find a local minimum of real-valued function $f(x_1, \ldots, x_n)$:

- assign arbitrary values to $x_1, \ldots, x_n$
- repeat:
  - for each $x_i$

$$x_i := x_i - \eta \frac{\partial f}{\partial x_i}$$
A linear function of variables $x_1, \ldots, x_n$ is of the form

$$f^\overline{w}(x_1, \ldots, x_n) = w_0 + w_1x_1 + \cdots + w_nx_n$$

$\overline{w} = \langle w_0, w_1, \ldots, w_n \rangle$ are weights. (Let $x_0 = 1$).

Given a set $E$ of examples.

Example $e$ has input $x_i = e_i$ for each $i$ and observed value, $o_e$:

$$\text{Error}_E(\overline{w}) = \sum_{e \in E} (o_e - f^\overline{w}(e_1, \ldots, e_n))^2$$

Minimizing the error using gradient descent, each example should update $w_i$ using:

$$w_i :=$$
A linear function of variables $x_1, \ldots, x_n$ is of the form

$$f^\overline{w}(x_1, \ldots, x_n) = w_0 + w_1 x_1 + \cdots + w_n x_n$$

$\overline{w} = \langle w_0, w_1, \ldots, w_n \rangle$ are weights. (Let $x_0 = 1$).

Given a set $E$ of examples.
Example $e$ has input $x_i = e_i$ for each $i$ and observed value, $o_e$:

$$Error_E(\overline{w}) = \sum_{e \in E} (o_e - f^\overline{w}(e_1, \ldots, e_n))^2$$

Minimizing the error using gradient descent, each example should update $w_i$ using:

$$w_i := w_i - \eta \frac{\partial Error_E(\overline{w})}{\partial w_i}$$
Given $E$: set of examples over $n$ features
each example $e$ has inputs $(e_1, \ldots, e_n)$ and output $o_e$:
Assign weights $\overline{w} = \langle w_0, \ldots, w_n \rangle$ arbitrarily
repeat:
   For each example $e$ in $E$:
      let $\delta = o_e - f^\overline{w}(e_1, \ldots, e_n)$
      For each weight $w_i$:
         $w_i := w_i + \eta \delta e_i$
One step backup provides the examples that can be used in a linear regression.

Suppose $F_1, \ldots, F_n$ are the features of the state and the action.

So $Q_w(s, a) = w_0 + w_1 F_1(s, a) + \cdots + w_n F_n(s, a)$

An experience $\langle s, a, r, s', a' \rangle$ provides the “example”:

- old predicted value:
- new “observed” value:
SARSA with linear function approximation

- One step backup provides the examples that can be used in a linear regression.
- Suppose $F_1, \ldots, F_n$ are the features of the state and the action.
- So $Q_w(s, a) = w_0 + w_1 F_1(s, a) + \cdots + w_n F_n(s, a)$
- An experience $\langle s, a, r, s', a' \rangle$ provides the “example”:
  - old predicted value: $Q_w(s, a)$
  - new “observed” value: $r + \gamma Q_w(s', a')$
One step backup provides the examples that can be used in a linear regression.

Suppose $F_1, \ldots, F_n$ are the features of the state and the action.

So $Q_w(s, a) = w_0 + w_1 F_1(s, a) + \cdots + w_n F_n(s, a)$

An experience $\langle s, a, r, s', a' \rangle$ provides the "example":

- old predicted value: $Q_w(s, a)$
- new "observed" value: $r + \gamma Q_w(s', a')$

Treat $r + \gamma Q_w(s', a')$ as a new training example for $Q(s, a)$ in linear regression (or other supervised learning algorithm).
SARSA with linear function approximation

Given $\gamma$: discount factor; $\eta$: step size
Assign weights $\overline{w} = \langle w_0, \ldots, w_n \rangle$ arbitrarily
observe current state $s$
select action $a$
repeat forever:
  carry out action $a$
  observe reward $r$ and state $s'$
  select action $a'$ (using a policy based on $Q_{\overline{w}}$)
SARSA with linear function approximation

Given $\gamma$: discount factor; $\eta$: step size
Assign weights $\overline{w} = \langle w_0, \ldots, w_n \rangle$ arbitrarily
observe current state $s$
select action $a$
repeat forever:
  carry out action $a$
  observe reward $r$ and state $s'$
  select action $a'$ (using a policy based on $Q_{\overline{w}}$)
  let $\delta = r + \gamma Q_{\overline{w}}(s', a') - Q_{\overline{w}}(s, a)$
  for $i = 0$ to $n$
    $w_i := w_i + \eta \delta F_i(s, a)$
  $s := s'$
  $a := a'$
SARSA with linear function approximation

Given \( \gamma \): discount factor; \( \eta \): step size
Assign weights \( \overline{w} = \langle w_0, \ldots, w_n \rangle \) arbitrarily
observe current state \( s \)
select action \( a \)
repeat forever:

- carry out action \( a \)
- observe reward \( r \) and state \( s' \)
- select action \( a' \) (using a policy based on \( Q_{\overline{w}} \))

let \( \delta = r + \gamma Q_{\overline{w}}(s', a') - Q_{\overline{w}}(s, a) \)

For \( i = 0 \) to \( n \)

\[ w_i := w_i + \eta \delta F_i(s, a) \]
SARSA with linear function approximation

Given $\gamma$: discount factor; $\eta$: step size

Assign weights $\overline{w} = \langle w_0, \ldots, w_n \rangle$ arbitrarily

observe current state $s$

select action $a$

repeat forever:

- carry out action $a$
- observe reward $r$ and state $s'$
- select action $a'$ (using a policy based on $Q_{\overline{w}}$)
- let $\delta = r + \gamma Q_{\overline{w}}(s', a') - Q_{\overline{w}}(s, a)$
- For $i = 0$ to $n$
  - $w_i := w_i + \eta \delta F_i(s, a)$
- $s := s'$
- $a := a'$
Example Features

- $F_1(s, a) = 1$ if $a$ goes from state $s$ into a monster location and is 0 otherwise.
- $F_2(s, a) = 1$ if $a$ goes into a wall, is 0 otherwise.
- $F_3(s, a) = 1$ if $a$ goes toward a prize.
- $F_4(s, a) = 1$ if the agent is damaged in state $s$ and action $a$ takes it toward the repair station.
- $F_5(s, a) = 1$ if the agent is damaged and action $a$ goes into a monster location.
- $F_6(s, a) = 1$ if the agent is damaged.
- $F_7(s, a) = 1$ if the agent is not damaged.
- $F_8(s, a) = 1$ if the agent is damaged and there is a prize in direction $a$.
- $F_9(s, a) = 1$ if the agent is not damaged and there is a prize in direction $a$. 
Example Features

- $F_{10}(s, a)$ is the distance from the left wall if there is a prize at location $P_0$, and is 0 otherwise.
- $F_{11}(s, a)$ has the value $4 - x$, where $x$ is the horizontal position of state $s$ if there is a prize at location $P_0$; otherwise is 0.
- $F_{12}(s, a)$ to $F_{29}(s, a)$ are like $F_{10}$ and $F_{11}$ for different combinations of the prize location and the distance from each of the four walls.

For the case where the prize is at location $P_0$, the $y$-distance could take into account the wall.
Problems and Variants of function approximation

This algorithm tends to overfit to current experiences. “Catastrophic forgetting”. Solution:
This algorithm tends to overfit to current experiences. “Catastrophic forgetting”.
Solution: remember old \( \langle s, a, r, s' \rangle \) experiences and to carry out some steps of action replay.
Problems and Variants of function approximation

- This algorithm tends to overfit to current experiences. “Catastrophic forgetting”.
  Solution: remember old \(\langle s, a, r, s'\rangle\) experiences and to carry out some steps of action replay

- Different function approximations, such as
  - a decision tree with a linear function at the leaves (regression tree)
  - a neural network

  could be used, but they requires a representation of the states and actions.
Problems and Variants of function approximation

- This algorithm tends to overfit to current experiences. “Catastrophic forgetting”.
  Solution: remember old $\langle s, a, r, s' \rangle$ experiences and to carry out some steps of action replay
- Different function approximations, such as
  ▶ a decision tree with a linear function at the leaves (regression tree)
  ▶ a neural network
  could be used, but they require a representation of the states and actions.
- Use the policy to do more than one-step lookahead (better estimate of $Q(s', a')$)
Evolutionary Algorithms

Idea:
- maintain a population of controllers
- evaluate each controller by running it in the environment
- at each generation, the best controllers are combined to form a new population of controllers
Evolutionary Algorithms

Idea:

- maintain a population of controllers
- evaluate each controller by running it in the environment
- at each generation, the best controllers are combined to form a new population of controllers

If there are $n$ states and $m$ actions, there are $m^n$ policies.
Evolutionary Algorithms

- Idea:
  - maintain a population of controllers
  - evaluate each controller by running it in the environment
  - at each generation, the best controllers are combined to form a new population of controllers

- If there are $n$ states and $m$ actions, there are $m^n$ policies.
- Experiences are used wastefully: only used to judge the whole controller. They don’t learn after every step.
- Performance is very sensitive to representation of controller.