Aprendizaje por Refuerzo

Eduardo Morales, Jesús González

INAOE

Enero, 2012

Contenido

Aprendizaje por Refuerzo

- Uno de los enfoques más usados dentro de aprendizaje es el aprendizaje supervisado a partir de ejemplos (pares entradas – salida provistos por el medio ambiente), para después predecir la salida de nuevas entradas.
- Cualquier sistema de predicción puede verse dentro de este paradigma, sin embargo, ignora la estructura secuencial del mismo.
- En algunos ambientes, muchas veces se puede obtener sólo cierta retroalimentación o recompensa o refuerzo (e.g., gana, pierde).

Aprendizaje por Refuerzo

- El refuerzo puede darse en un estado terminal y/o en estados intermedios.
- Los refuerzos pueden ser componentes o sugerencias de la utilidad actual a maximizar (e.g., buena movida).
- En aprendizaje por refuerzo (RL) el objetivo es aprender cómo mapear situaciones a acciones para maximizar una cierta señal de recompensa.
- Promesa: *programar agentes mediante premio y castigo sin necesidad de especificar cómo realizar la tarea*

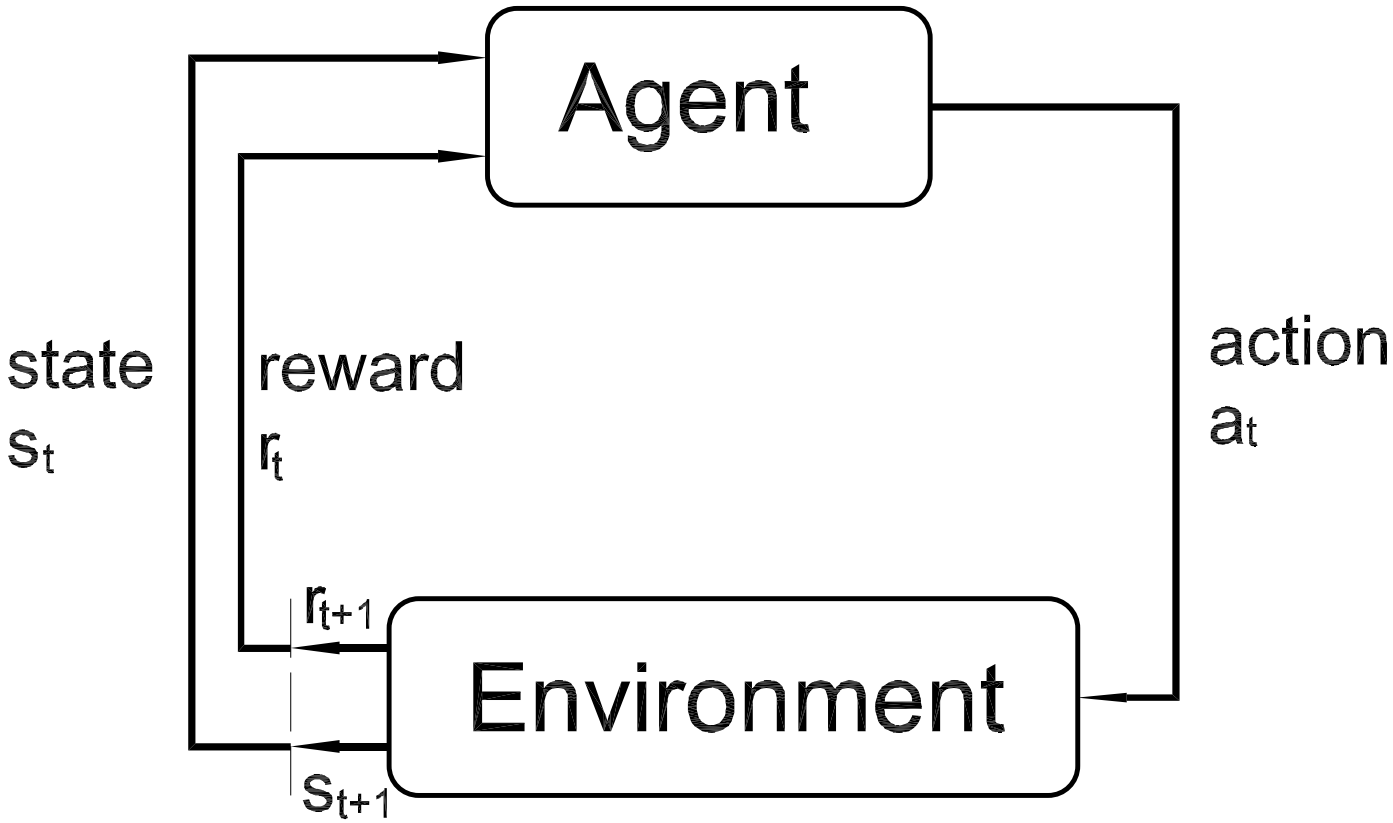
Diferencias con Otro Tipo de Aprendizaje

1. No se le presentan pares entrada - salida.
2. El agente tiene que obtener experiencia útil acerca de los estados, acciones, transiciones y recompensas de manera activa para poder actuar de manera óptima.
3. La evaluación del sistema ocurre en forma concurrente con el aprendizaje.

Aprendizaje por Refuerzo

- En RL un agente trata de aprender un comportamiento mediante interacciones de prueba y error en un ambiente dinámico e incierto
- En general, al sistema no se le dice qué acción debe tomar, sino que él debe de descubrir qué acciones dan el máximo beneficio
- En un RL estandar, un agente está conectado a un ambiente por medio de percepción y acción
- En cada interacción el agente recibe como entrada una indicación de su estado actual ($s \in S$) y selecciona una acción ($a \in A$). La acción cambia el estado y el agente recibe una señal de refuerzo o recompensa ($r \in \mathcal{R}$)

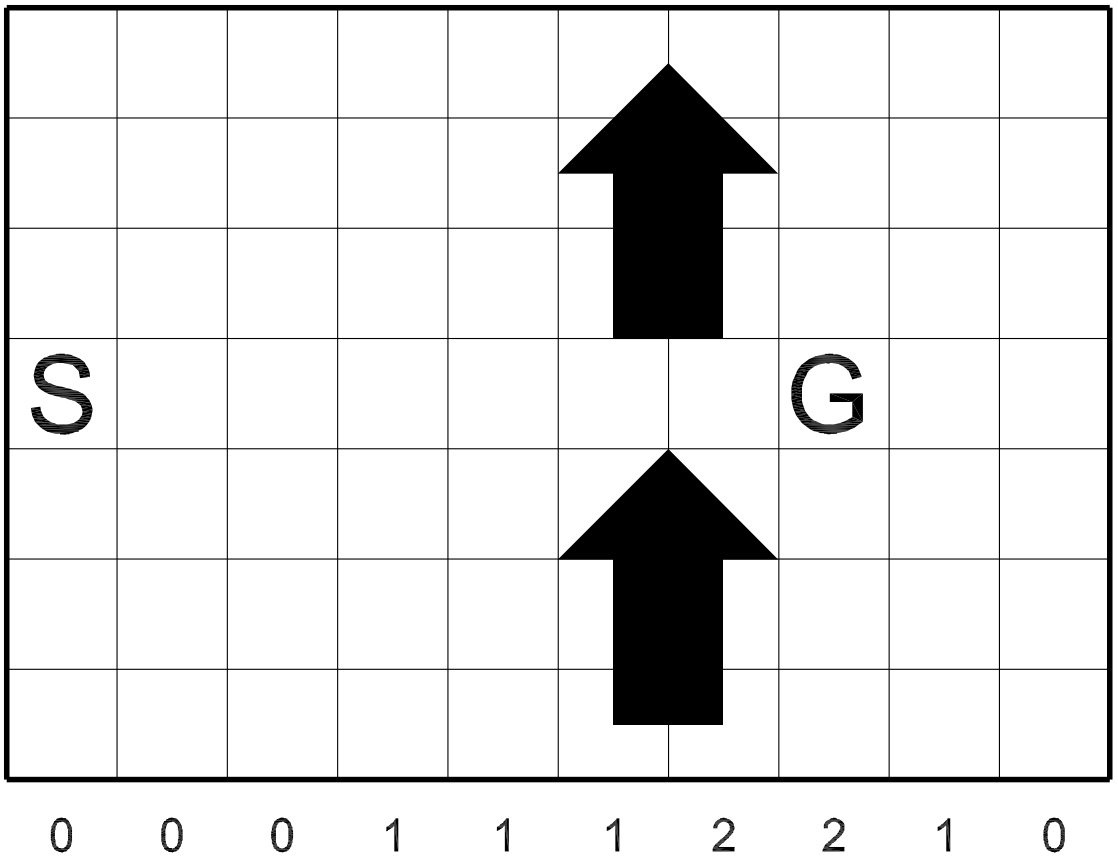
Aprendizaje por Refuerzo

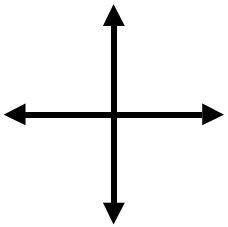


Aprendizaje por Refuerzo

- El comportamiento del agente debe de ser tal que escoga acciones que tiendan a incrementar a largo plazo la suma de las recompensas totales
- El objetivo del agente es encontrar una política (π), que mapea estados a acciones que maximice a largo plazo el refuerzo
- En general el ambiente es no-determinístico (tomar la misma acción en el mismo estado puede dar resultados diferentes)
- Sin embargo, se asume que el ambiente es estacionario (las probabilidades de cambio de estado no cambian o cambian muy lentamente)

Ejemplo de Problema





standard moves

Exploración y Explotación

- Aspectos importantes:
  1. se sigue un proceso de prueba y error
  2. la recompensa puede estar diferida
- Existe un balance entre exploración y explotación
- Para obtener buena ganancia uno prefiere seguir ciertas acciones, pero para saber cuáles, se tiene que hacer cierta exploración
- Muchas veces depende de cuánto tiempo se espera que el agente interactue con el medio ambiente

Procesos de Decisión de Markov

- En RL se tiene que decidir en cada estado la acción a realizar
- Este proceso de decisión secuenial se puede caracterizar como un procesos de decisión de Markov o MDP
- Un MDP modela un problema de decisión sequencial en donde el sistema evoluciona en el tiempo y es controlado por un agente
- La dinámica del sistema esta determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados

MDP

Formalmente, un MDP es una tupla $M =< S, A, \Phi, R >$ formada por:

- Un conjunto finito de estados $S(\{1, ..., n\})$.
- Un conjunto finito de acciones A, que pueden depender de cada estado
- Una función de recompensa (R), que define la meta y mapea cada estado–acción a un número (recompensa), indicando lo deseable del estado
- Un modelo del ambiente o función de transición de estados $\Phi : A \times S \rightarrow S$ que nos dice la probabilidad de alcanzar el estado $s' \in S$ al realizar la acción $a \in A$ en el estado $s \in S$, que se puede denotar como $\Phi(s'|s, a)$.

Elementos Adicionales

- Política (π): define cómo se comporta el sistema en cierto tiempo. Es un mapeo (a veces estocástico) de los estados a las acciones.
- Función de valor (V): indica lo que es bueno a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en un estado s ($V(s)$) o en un estado haciendo una acción a ($Q(s,a)$)
- Las recompensas están dadas por el ambiente, pero los valores se deben de estimar (aprender) en base a las observaciones.

Aprendizaje por refuerzo aprende las funciones de valor mientras interactua con el ambiente.

Modelos de Recompensas

- Dado un estado $s_t \in S$ y una acción $a_t \in \mathcal{A}(s_t)$, el agente recibe una recompensa r_{t+1} y se mueve a un nuevo estado s_{t+1}
- El mapeo de estados a probabilidades de seleccionar una acción particular es su política (π_t). Aprendizaje por refuerzo especifica cómo cambiar la política como resultado de su experiencia
- No trata de maximizar la recompensa inmediata, sino la recompensa a largo plazo (acumulada)

Modelos de Recompensas

- La recompensa debe de mostrar lo que queremos obtener y se calcula por el ambiente
- Si las recompensas recibidas después de un tiempo t se denotan como: r_{t+1}, r_{t+2}, r_{t+3}, ..., lo que queremos es maximizar lo que esperamos recibir de recompensa (R_t)
- Si se tiene un punto terminal se llaman tareas *episódicas*, si no se tiene se llaman tareas *continuas*.

Modelos de Recompensas

- Cuando no sabemos el punto terminal podemos hacer una suma con un factor de descuento:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

  donde γ se conoce como la *razón de descuento* y está entre: $0 \leq \gamma < 1$
- Si $\gamma = 0$ se trata sólo de maximizar tomando en cuenta las recompensas inmediatas

Modelos de Recompensas

- **Horizonte finito**: el agente trata de optimizar su recompensa esperada en los siguientes h pasos, sin preocuparse de lo que ocurra despues:

$$E(\sum_{t=0}^{h} r_t)$$

- Se puede usar como:
  - *política no estacionaria*: en el primer paso se toman los h siguientes pasos, en el siguiente los $h-1$, etc., hasta terminar. El problema principal es que no siempre se conoce cuántos pasos considerar
  - *receding-horizon control*: siempre se toman los siguientes h pasos

Modelos de Recompensas

- **Horizonte infinito** (la más utilizada): las recompensas que recibe un agente son reducidas geométricamente de acuerdo a un factor de descuento γ ($0 \leq \gamma \leq 1$):

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

- **Recompensa promedio**: optimizar a largo plazo la recompensa promedio:

$$lim_{h \to \infty} E(\frac{1}{h} \sum_{t=0}^{h} r_t)$$

Problema: no distingue políticas que reciban grandes recompensas al principio de las que no.

Modelo Markoviano

- En general, las acciones del agente determinan, no sólo la recompensa inmediata, sino también (por lo menos en forma probabilística) el siguiente estado del ambiente
- Los problemas con refuerzo diferido se pueden modelar como procesos de decisión de Markov (MDPs)
- El modelo es Markoviano si las transiciones de estado no dependen de estados anteriores

Modelos Markoviano

- RL asume que se cumple con la propiedad Markoviana y las probabilidades de transición están dadas por:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

El valor de recompensa esperado es:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

- Lo que se busca es estimar las funciones de valor. Esto es, qué tan bueno es estar en un estado (o realizar una acción)
- La noción de “qué tan bueno” se define en términos de recompensas futuras o recompensas esperadas

Funciones de Valor

- La política π es un mapeo de cada estado $s \in S$ y acción $a \in \mathcal{A}(s)$ a la probabilidad $\pi(s,a)$ de tomar la acción a estando en estado s
- El valor de un estado s bajo la política π, denotado como $V^\pi(s)$, es el refuerzo esperado estando en estado s y siguiendo la política π:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- El valor esperado tomando una acción a en estado s bajo la política π ($Q^\pi(s,a)$):

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{R_t \mid s_t = s, a_t = a\} \\ &= E_\pi\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \end{aligned}$$

Funciones de Valor Óptimas

- Las funciones de valor óptimas se definen como:

$$V^*(s) = max_\pi V^\pi(s) \text{ y } Q^*(s,a) = max_\pi Q^\pi(s,a)$$

- Las cuales se pueden expresar como las ecuaciones de optimalidad de Bellman:

$$V^*(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma max_{a'} Q^*(s',a')]$$

Métodos de Solución

Existen tres formas principales de resolver MDPs:

1. usando métodos de programación dinámica
2. usando métodos de Monte Carlo, y
3. usando métodos de diferencias temporales o de aprendizaje por refuerzo

Programación Dinámica

- Si se conoce el modelo del ambiente, osea las transiciones de probabilidad ($\mathcal{P}^a_{ss'}$) y los valores esperados de recompensas ($\mathcal{R}^a_{ss'}$), las ecuaciones de optimalidad de Bellman nos representan un sistema de $|S|$ ecuaciones y $|S|$ incognitas
- Consideremos primero como calcular la función de valor V^π dada una política arbitraria π.

Funciones de Valor para una Política

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')] \end{aligned}$$

donde $\pi(s,a)$ es la probabilidad de tomar la acción a en estado s bajo la política π.

Funciones de Valor para una Política

- Podemos hacer aproximaciones sucesivas, evaluando $V_{k+1}(s)$ en términos de $V_k(s)$.

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- Podemos entonces definir un algoritmo de evaluación iterativa de políticas

Funciones de Valor para una Política

Inicializa $V(s) = 0$ para toda $s \in S$
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
Regresa $V \approx V^\pi$

Iteración de Políticas

- Calculamos la función de valor de una política para tratar de encontrar mejores políticas
- Dada una función de valor, podemos probar una acción $a \neq \pi(s)$ y ver si su $V(s)$ es mejor o peor que el $V^{\pi}(s)$
- En lugar de hacer un cambio en un estado y ver el resultado, se pueden considerar cambios en todos los estados considerando todas las acciones de cada estado, seleccionando aquella que parezca mejor de acuerdo a una política *greedy*.
- Podemos entonces calcular una nueva política $\pi'(s) = \text{argmax}_a Q^{\pi}(s, a)$ y continuar hasta que no mejoremos.

Iteración de Políticas

- Esto sugiere, partir de una política (π_0) y calcular la función de valor (V^{π_0}), con la cual encontrar una mejor política (π_1) y así sucesivamente hasta converger a π^* y V^*.
- A este procedimiento se llama iteración de políticas

Iteración de Políticas

$V(s) \in \mathcal{R}$ y $\pi(s) \in \mathcal{A}(s)$ arbitrariamente $\forall s \in S$ (**Inicialización**)
Repite (**Evaluación de Política**)
    $\Delta \leftarrow 0$
    Para cada $s \in S$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} [\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
pol-estable $\leftarrow$ true (**Mejora de Política**)
Para cada $s \in S$:
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} \mathcal{P}_{ss'}^{a} [\mathcal{R}_{ss'}^{a} + \gamma V(s')]$
    if $b \neq \pi$, then *pol-estable* $\leftarrow$ false
If *pol-estable*, then stop, else go to 2.

Iteración de Valor

- Iteración de políticas en cada iteración evalúa la política y requiere recorrer todos los estados varias veces
- El paso de evaluación de política lo podemos truncar sin perder la garantía de convergencia, después de recorrer una sola vez todos los estados
- A esta forma se le llama iteración de valor (*value iteration*) y se puede escribir combinando la mejora en la política y la evaluación de la política truncada como sigue:

$$V_{k+1}(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- Se puede ver como expresar la ecuación de Bellman en una regla de actualización

Iteración de Valor

Inicializa $V(s) = 0$ para toda $s \in S$
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
Regresa una política determinística tal que:
$\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$

Bootstrapping

- Para espacios muy grandes, el ver todos los estados puede ser computacionalmente muy caro. Una opción es hacer estas actualizaciones al momento de estar explorando el espacio, y por lo tanto determinando sobre qué estados se hacen las actualizaciones.
- El hacer estimaciones en base a otras estimaciones se conoce también como *bootstrapping*.

Monte Carlo

- Los métodos de Monte Carlo, solo requieren de experiencia y la actualización se hace por episodio más que por cada paso.
- El valor de un estado es la recompensa esperada que se puede obtener a partir de ese estado.
- Para estimar V^{π} y Q^{π} podemos tomar estadísticas haciendo un promedio de las recompensas obtenidas

Monte Carlo para Estimar V^{π}

Repite
Genera un episodio usando π
Para cada estado s en ese episodio:
    $R \leftarrow$ recompensa después de la primera ocurrencia de s
    Añade R a $recomp(s)$
    $V(s) \leftarrow$ promedio($recomp(s)$)

Esquemas de Exploración

- Para estimar pares estado-acción (Q^{π}) corremos el peligro de no ver todos los pares, por lo que se busca mantener la exploración. Lo que normalmente se hace es considerar solo políticas estocásticas que tienen una probabilidad diferente de cero de seleccionar todas las acciones.

Esquemas de Exploración

- $\epsilon-$*greedy*: en donde la mayor parte del tiempo se selecciona la acción que da el mayor valor estimado, pero con probabilidad ϵ se selecciona una acción aleatoriamente.
- *softmax*, en donde la probabilidad de selección de cada acción depende de su valor estimado. La más común sigue una distribución de Boltzmann o de Gibbs, y selecciona una acción con la siguiente probabilidad:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

donde τ es un parámetro positivo (temperatura).

Monte Carlo para Mejorar Políticas

- Con Monte Carlo podemos alternar entre evaluación y mejoras en base a cada episodio
- La idea es que después de cada episodio las recompensas observadas se usan para evaluar la política y la política se mejora para todos los estados visitados en el episodio

Algoritmo de Monte Carlo para Mejorar Políticas

Repite
Genera un episodio usando π con exploración
Para cada par s, a en ese episodio:
    $R \leftarrow$ recompensa después de la primera ocurrencia de s, a
    Añade R a $recomp(s, a)$
    $Q(s, a) \leftarrow$ promedio$(recomp(s, a))$
Para cada s en el episodio:
    $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$

Allgoritmos “on” y “off-policy”

- Los algoritmos *on-policy*: Estiman el valor de la política mientras la usan para el control. Se trata de mejorar la política que se usa para tomar decisiones.
- Los algoritmos *off-policy*: Usan la política y el control en forma separada. La estimación de la política puede ser por ejemplo *greedy* y la política de comportamiento puede ser ϵ-*greedy*. Osea que la política de comportamiento está separada de la política que se quiere mejorar.
  Esto es lo que hace Q-learning, lo cual simplifica el algoritmo.

Diferencias Temporales

- Los métodos de TD combinan las ventajas de los dos anteriores: permite hacer *bootstrapping* (como DP) y no requiere tener un modelo del ambiente (como MC).
- Métodos tipo TD sólo tienen que esperar el siguiente paso.
- TD usan el error o diferencia entre predicciones sucesivas (en lugar del error entre la predicción y la salida final) aprendiendo al existir cambios entre predicciones sucesivas.

Aprendizaje por Refuerzo

- Los algoritmos son incrementales y fácil de computar
- Actualizan las funciones de valor usando el error entre lo estimado y la suma de la recompensa inmediata y lo estimado del siguiente estado
- El más simple TD(0) es:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

Algoritmo TD(0)

Inicializa $V(s)$ arbitrariamente y π a la política a evaluar
Repite (para cada episodio):
    Inicializa s
    Repite (para cada paso del episodio):
        $a \leftarrow$ acción dada por π para s
        Realiza acción a; observa la recompensa, r,
            y el siguiente estado, s'
        $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
        $s \leftarrow s'$
    hasta que s sea terminal

SARSA

- La actualización de valores tomando en cuenta la acción sería:

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

- y el algoritmo es prácticamente el mismo, solo que se llama SARSA

Algoritmo SARSA

Inicializa $Q(s, a)$ arbitrariamente
Repite (para cada episodio):
    Inicializa s
    Selecciona una a a partir de s usando la política dada por Q (e.g., ϵ–greedy)
    Repite (para cada paso del episodio):
        Realiza acción a, observa r, s'
        Escoge a' de s' usando la política derivada de Q
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    hasta que s sea terminal

Q-Learning

- Uno de los desarrollos más importantes en aprendizaje por refuerzo fué el desarrollo de un algoritmo "fuera-de-política" (*off-policy*) conocido como Q-learning.
- La idea principal es realizar la actualización de la siguiente forma (Watkins, 89):

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

Algoritmo Q-Learning

Inicializa $Q(s, a)$ arbitrariamente
Repite (para cada episodio):
    Inicializa s
    Repite (para cada paso del episodio):
        Selecciona una a de s usando la política dada por Q (e.g., ϵ–greedy)
        Realiza acción a, observa r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma max'_a Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$;
    hasta que s sea terminal

Estrategias para Espacios Grandes

- Uno de los problemas principales de RL es su aplicación a espacios grandes (muchos estados y acciones). Aunque los algoritmos convergen en teoría, en la práctica puede tomar un tiempo inaceptable.
- Se han propuesto diferentes estrategias para esto:
  1. Actualizar varias funciones de valor a la vez
  2. Usar aproximación de funciones
  3. Aprender un modelo y usarlo
  4. Utilizar abstracciones y jerarquías
  5. Incorporar ayuda adicional

Trazas de Elegibilidad

- Están entre métodos de Monte Carlo y TD de un paso.
- Los métodos Monte Carlo realizan la actualización considerando la secuencia completa de recompensas observadas.
- La actualización de los métodos de TD la hacen utilizando únicamente la siguiente recompensa.
- La idea de las trazas de elegibilidad es considerar las recompensas de n estados posteriores (o afectar a n anteriores).

Trazas de Elegibilidad

- Si recordamos:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$$

- Lo que se hace en TD es usar:

$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

donde $V_t(s_{t+1})$ reemplaza a los siguientes términos $(r_{t+2} + \gamma r_{t+3} \ldots)$

- Sin embargo, hace igual sentido hacer:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

y, en general, para n pasos en el futuro.

Trazas de Elegibilidad

- En la práctica, más que esperar n pasos para actualizar (*forward view*), se realiza al revés (*backward view*). Se puede probar que ambos enfoques son equivalentes.
- Se guarda información sobre los estados por los que se pasó y se actualizan hacia atrás los “errores” (descontadas por la distancia)
- Para esto se asocia a cada estado o par estado-acción una variable extra, representando su traza de elegibilidad (*eligibility trace*) que denotaremos por $e_t(s)$ o $e_t(s,a)$.
- Este valor va decayendo con la longitud de la traza creada en cada episodio

Trazas de Elegibilidad

- Para $TD(\lambda)$:

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

- Para SARSA se tiene lo siguiente:

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

SARSA con Trazas de Elegibilidad

Inicializa $Q(s,a)$ arbitrariamente y $e(s,a) = 0\ \forall s,a$
Repite (para cada episodio)
  Inicializa s,a
  Repite (para cada paso en el episodeo)
    Toma acción a y observa r, s'
    Selecciona a' de s' usando una política derivada de Q (e.g., $\epsilon-greedy$)
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    Para todos s,a
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'; a \leftarrow a'$
  hasta que s sea terminal

Q-Learning y Trazas de Elegibilidad

- Para Q-learning como la selección de acciones se hace, por ejemplo, siguiendo una política $\epsilon-greedy$, se tiene que tener cuidado, ya que a veces los movimientos, son movimientos exploratorios
- No queremos propagar en caminos buenos “errores” negativos por acciones exporatorias
- Se puede mantener historia de la traza solo hasta el primer movimiento exploratorio, ignorar las acciones exploratorias, o hacer un esquema un poco más complicado que considera todas las posibles acciones en cada estado.

Aprendiendo Modelos

- Con un modelo podemos predecir el siguiente estado y la recomepensa dado un estado y una acción
- La predicción puede ser un conjunto de posibles estados con su probabilidad asociada o puede ser un estado que es muestreado de acuerdo a la distribución de probabilidad de los estados resultantes
- Lo interesante es que podemos utilizar los estados y acciones simulados para aprender. Al sistema de aprendizaje no le importa si los pares estado-acción son dados de experiencias reales o simuladas.

Dyna-Q

- Dado un modelo del ambiente, uno puede seleccionar aleatoriamente un par estado–acción, usar el modelo para predecir el siguiente estado, obtener una recompensa y actualizar valores Q. Esto se puede repetir indefinidamente hasta converger a Q^*.
- El algoritmo Dyna-Q combina experiencias con planificación para aprender más rápidamente una política óptima.
- La idea es aprender de experiencia, pero también usar un modelo para simular experiencia adicional y así aprender más rápidamente

Algoritmo de Dyna-Q

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$
DO forever
- $s \leftarrow$ estado actual
- $a \leftarrow \epsilon-greedy(s,a)$
- realiza acción a observa s' y r
- $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- $Modelo(s,a) \leftarrow s', r$
- Repite N veces:
  - $s \leftarrow$ estado anterior seleccionado aleatoriamente
  - $a \leftarrow$ acción aleatoria tomada en s
  - $s', r \leftarrow Modelo(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Prioritized Sweeping

- El algoritmo de Dyna-Q selecciona pares estado-acción aleatoriamente de pares anteriores. Sin embargo, la planificación se puede usar mucho mejor si se enfoca a pares estado-acción específicos.
- Por ejemplo, enfocarnos en las metas e irnos hacia atrás o más generalmente, irnos hacia atrás de cualquer estado que cambie su valor.
- Los cambios en las estimaciones de valor V o Q pueden cambiar, cuando se está aprendiendo o si el ambiente cambia y un valor estimado deja de ser cierto.

Prioritized Sweeping

- Lo que se puede hacer es enfocar la simulación al estado que cambio su valor. Esto nos lleva a todos los estados que llegan a ese estado y que también cambiarían su valor.
- Esto proceso se puede repetir sucesivamente, sin embargo, algunos estados cambian mucho más que otros. Lo que podemos hacer es ordenarlos y cambiar solo los que rebacen un cierto umbral. Esto es precisamente lo que hace el algoritmo de *prioritized sweeping*

Algoritmo de Prioritized sweeping.

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$ y $ColaP = \emptyset$
DO forever
- $s \leftarrow$ estado actual
- $a \leftarrow \epsilon-greedy(s,a)$
- realiza acción a observa s' y r
- $Modelo(s,a) \leftarrow s', r$
- $p \leftarrow | r + \gamma \max_{a'} Q(s',a') - Q(s,a) |$
- if $p > \theta$, then inserta (s,a) a $ColaP$ con prioridad p
- Repite N veces, mientras $ColaP \neq \emptyset$:
  - $s, a \leftarrow$ primero($ColaP$)
  - $s', r \leftarrow Modelo(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  - Repite $\forall \overline{s}, \overline{a}$ que se predice llegan a s:
    - $\overline{r} \leftarrow$ recomensa predicha
    - $p \leftarrow | \overline{r} + \gamma \max_{a} Q(s,a) - Q(\overline{s},\overline{a}) |$
    - if $p > \theta$, then inserta $\overline{s}, \overline{a}$ a $ColaP$ con prioridad p

Aproximación de Funciones

- Hasta ahora hemos asumido que se tiene una representación explícita en forma de tabla, lo cual funciona para epacios pequeños, pero es impensable para dominios como ajedrez (10^{120}) o backgammon (10^{50}).
- Una alternativa es usar una representación implícita, i.e., una función.
- Por ejemplo en juegos, una función de utilidad estimada se puede representar como una función lineal pesada sobre un conjunto de atributos (F_i's):

$$V(i) = w_1 f_1(i) + w_2 f_2(i) + \ldots + w_n f_n(i)$$

- En ajedrez se tienen aproximadamente 10 pesos, por lo que es una compresión bastante significativa.

Aprendizaje de Funciones

- La compresión lograda por una representación implícita permite al sistema de aprendizaje, generalizar de estados visitados a estados no visitados
- Existen muchas alternativas que se pueden usar y en RL se han usado NN, SVM, árboles de decisión, procesos gaussianos, etc.
- Como en todos los sistemas de aprendizaje, existe un balance entre el espacio de hipótesis y el tiempo que toma aprender una hipótesis aceptable

Aprendizaje de Funciones

- Vamos a ver cómo se puede hacer con una combinación lineal de funciones base $\phi_1, \phi_2, \ldots, \phi_n$ de la forma $\sum_{i=1}^{n} w_i \phi_i$ donde los pesos w_i son los que tenemos que aprender (Θ)
- Muchos sistemas de aprendizaje supervisado tratan de minimizar el error cuadrado (MSE)
- Si $\vec{\Theta}_t$ representa el vector de parámetros de la función parametrizada que queremos aprender:

$$MSE(\vec{\Theta}_t) = \sum_{s \in S} P(s)[V^{\pi}(s) - V_t(s)]^2$$

donde $P(s)$ es una distribución pesando los errores de diferentes estados

Aprendizaje de Funciones

Para ajustar los parámetros del vector de la función que queremos optimizar, las técnicas de gradiente ajustan los valores en la dirección que produce la máxima reducción en el error:

$$\begin{aligned}\vec{\Theta}_{t+1} &= \vec{\Theta}_t - \tfrac{1}{2}\alpha\nabla_{\vec{\Theta}_t}[V^{\pi(s_t)} - V_t(s_t)]^2 \\ &= \vec{\Theta}_t + \alpha[V^{\pi(s_t)} - V_t(s_t)]\nabla_{\vec{\Theta}_t} V_t(s_t)\end{aligned}$$

donde α es un parámetro positivo $0 \leq \alpha \leq 1$ y $\nabla_{\vec{\Theta}_t} f(\Theta_t)$ denota un vector de derivadas parciales.

Aprendizaje de Funciones

- Como no sabemos $V^{\pi}(s_t)$ lo tenemos que aproximarla y podemos hacerlo con trazas de elegibilidad y actualizar la función Θ como sigue:

$$\vec{\Theta}_{t+1} = \vec{\Theta}_t + \alpha\delta_t\vec{e}_t$$

donde δ_t es el error:

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

y $\vec{e}_t$ es un vector de trazas de elegibilidad, una por cada componente de $\vec{\Theta}_t$, que se actualiza como:

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\Theta}_t} V_t(s_t)$$

con $\vec{e}_0 = 0$

Abstracciones y Jerarquías

- Agregación de estados: se juntan estados “parecidos” y a todos ellos se les asigna el mismo valor, reduciendo el espacio de estados. Por ejemplo: tile-coding, coarse coding, radial basis functions, Kanerva coding, y soft-state aggregation.
- Abstracciones basadas en máquinas de estado finito: el aprendizaje por refuerzo tiene que decidir que máquina utilizar (por ejemplo, HAM y PHAM).

Abstracciones y Jerarquías

- Definición de jerarquías: se divide el espacio en subproblemas, se aprenden políticas a los espacios de más bajo nivel y estas se usan para resolver problemas de más alto nivel (e.g., MAXQ, HEXQ). Algo parecido se usa con Macros y Options, en donde se aprenden políticas de subespacios que se usan para resolver problemas mas grandes.
- También se ha buscado utilizar representaciones relacionales dentro de aprendizaje por refuerzo, ya sea para representar las funciones de valor y/o para representar los estados y las acciones.

Incorporar Información Adicional

- En su forma tradicional, RL no utiliza prácticamente nada de conocimiento del dominio
- Una forma de ayudar a RL a coverger más rápidamente es incorporardo información adicional:
  1. La idea de *reward shaping* es incorporar información adicional a la función de recompensa
  2. También se han utilizado soluciones conocidas como guías o trazas que se usan para aprender más rápidamente las funciones de valor o para aprender un subconjunto de acciones relevantes.

Aplicaciones

- La primera aplicación en aprendizaje por refuerzo fué el programa para jugar damas de Samuel
- Usó una función lineal de evaluación con pesos usando hasta 16 términos
- Su programa era parecido a la ecuación de actualización de pesos, pero no usaba recompensa en los estados terminales, lo que hace que puede o no converger y puede aprender a perder.
- Logró evitar ésto haciendo que el peso para ganancia de material fuera siempre positivo.

Aplicaciones

- Una de las más conocidas es el control del péndulo invertido. Controlar la posición x para que se mantenga aproximadamente derecho ($\theta \approx \pi/2$), manteniendose en los límites de la pista. $X, \theta, \dot{X}$ y $\dot{\theta}$ son continuas. El control es de tipo bang–bang.
- Boxes (Michie, Chambers '68) balanceaba el pendulo por más de una hora después de 30 intentos (no simulado)
- Discretizaron el espacio en cajas. Se corria el sistema hasta que se caía el péndulo o se salia de los límites. Entonces se daba un refuerzo negativo a la última "caja" y se propagaba a la secuencia de "cajas" por las que pasó.

Aplicaciones

- TD-gammon (Tesauro ’92) representó una función de evaluación con una red neuronal de una sola capa intermedia con 40 nodos, que después de 200,000 juegos de entrenamiento mejoró notablemente su desempeño.
- Añadiendo atributos adicionales a una red con 80 nodos escondidos, después de 300,000 juegos de entrenamiento, juega como los 3 mejores jugadores del mundo.
- También se desarrolló un algoritmo de RL que actualiza las funciones de evaluación en un árbol de búsqueda en juegos. En ajedrez mejora el puntaje de un programa de 1,650 a 2,150 después de 308 juegos en 3 días.

Ejemplos

Ejemplos

Ejemplos

Ejemplos