Aprendizaje por Refuerzo

Tratamiento Inteligente de la Información y Aplicaciones

Juan A. Botía Blaya

Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

November 24, 2009

Métodos de aprendizaje

Dependiendo de la información utilizada para el aprendizaje

1. Un profesor directo: que proporciona, en cada instante de tiempo, la acción de control correcta a ser aplicada por el aprendedor
2. Un profesor indirecto: que sin proporcionar las acciones correctas, suministra los efectos deseados del proceso
3. Una medida de rendimiento: indica la calidad del aprendedor en un conjunto de estados (i.e. como en la medida de adaptación de un individuo en una población de algoritmo evolutivo)
4. Un crítico: que asigna recompensas y castigos con respecto a los estados alcanzados por el aprendedor, pero no proporcionar acciones correctas ni trayectorias a seguir.

Problemas de decisión de Markov (MDP)

- Un proceso de decisión de Markov es un proceso estocástico que viene caracterizado por cinco elementos:
  1. pasos para la decisión
  2. estados
  3. acciones
  4. probabilidades de transición y
  5. recompensas

Problemas de decisión de Markov (MDP) y II

- En algunos casos, es posible encontrar un agente encargado de controlar el camino de decisión tomado en el proceso probabilístico
- En cada paso el sistema se encuentra en un estado de toma de decisiones
- como resultado de la ejecución de una acción en un estado determinado, el agente decisor recibe una recompensa
- Tras esto, el sistema pasa al estado siguiente con una cierta probabilidad
- Una regla de decisión es una función usada para seleccionar, en cada estado, una acción.
- Una política es un conjunto de esas reglas de decisión sobre el espacio de estados

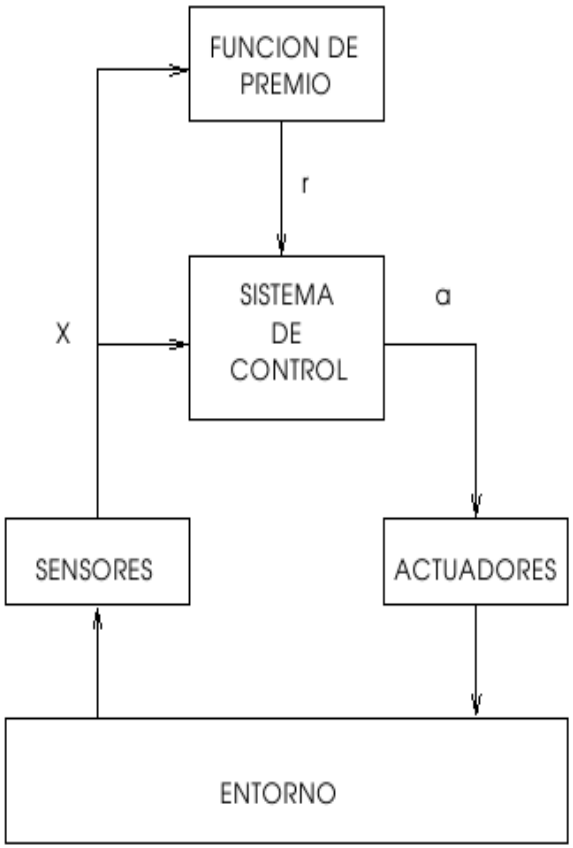
Prob. de decisión de Markov (MDP) (y II)

- Sea $MDP = \{X_n : n \in \mathcal{N}, X_n \in X\}$ un proceso de decisión de Markov
  - X_n es el estado en la fase de decisión $n-$ésima
  - $\mathcal{N}$ los enteros naturales
- Sea el paso n, donde $X_n = i \in X$, la acción escogida es $a_n \in A_i$, siendo A_i el conjunto de acciones que pueden tomarse en el estado i y $\bigcup A_i = A$. el conjunto de acciones posibles
- Dada $a \in A$, lleva asociada una matriz de transiciones a estados $P(a)$ de la cadena de Markov MDP, en donde $P(x_j|x_i, a)$ es la probabilidad de mover desde el estado i al j ejecutando la acción a
- Definimos también una función de recompensa

$$r : X \times A \rightarrow \Re,$$

  en donde $r(i, a)$ es la recompensa esperada al ejecutar la acción a en el estado i
- Asumimos: (a) rs acotadas superior e inferiormente, (b) probabilidades de la matriz de transiciones estacionarias y (c) espacio de estados finito.

Estructura de un aprendedor por refuerzo



El entorno

- La información que el sistema conoce sobre el entorno, en un instante t se puede representar en un vector de contexto x_t (vector de descripción del estado)
- La decisión acerca de qué acción se ha de ejecutar se tomará en base a este vector
  - si no incluye toda la información importante para el modelado, la calidad del aprendizaje se verá afectada
- $P(x_j|x_i, a)$ será cero en la mayoría de estados y para la mayoría de acciones
- Si las $P(x_j|x_i, a)$ se conocen, entonces disponemos de un *modelo del mundo*.
- Si no
  1. Podemos aprenderlo
  2. Podemos prescindir de él

Resultados de ejecución de acciones

- Los resultados (i.e. recompensas) son valores escalares $r(x_i, x_j)$, recibidas por el sistema a raíz de la transición de un estado a otro
- En el caso general, las recompensas pueden generarse apartir de una distribución de probabilidad
- En sistemas de aprendizaje por refuerzo simples, la acción más deseable es aquella que proporciona la recompensa inmediata más alta
  - el resultado que se va a alcanzar en cada instante es conocido pero la acción para ello no está clara
  - Ejemplo: mover la garra de un brazo de robot
- En el caso general se maximiza la suma esperada de recompensas descontadas recibidas (retorno)

$$E\left\{\sum_{t=0}^{\infty}\gamma^t r_t\right\}$$

  - r_t la recompensa recibida en la transición del estado x_t al x_{t+1} en el instante t
  - La constante $0 \leq \gamma \leq 1$ se denomina factor de descuento y asegura que la suma de recompensas es finita t $(r(x_t, x_{t+1}))$

Importancia de las recompensas

- Las recompensas definen
  - el problema a resolver y
  - las restricciones impuestas a la política de control que usa el sistema aprendedor
- si no se da una buena recompensa a una acción considerada como buena o una mala recompensa a una acción considerada como mala
  - el sistema aprendedor puede no satisfacer los requerimientos del diseñador
- la tarea debe estar totalmente descrita mediante la función de recompensa

Ejemplo: el robot reciclador

- Robot móvil que recolecta latas vacías de bebidas, en un entorno de oficina.
  - Tiene sensores para detectar las latas
  - mediante una garra puede recogerlas y colocarlas en un depósito de abordo
  - se alimenta con una batería recargable
- En cada momento, el robot debe decidir si
  1. busca una lata activamente,
  2. permanece estacionario y espera a que alguien le lleve una lata o
  3. vuelve a la estación base a recargar su batería.

Ejemplo: el robot reciclador (y II)

- Entorno:
  - la mejor forma de encontrar latas es buscarlas activamente (batería sufre descarga rápidamente)
  - la espera no descarga la batería
  - Si se descarga el robot debe apagarse y esperar a ser rescatado, produciéndose así una recompensa baja.

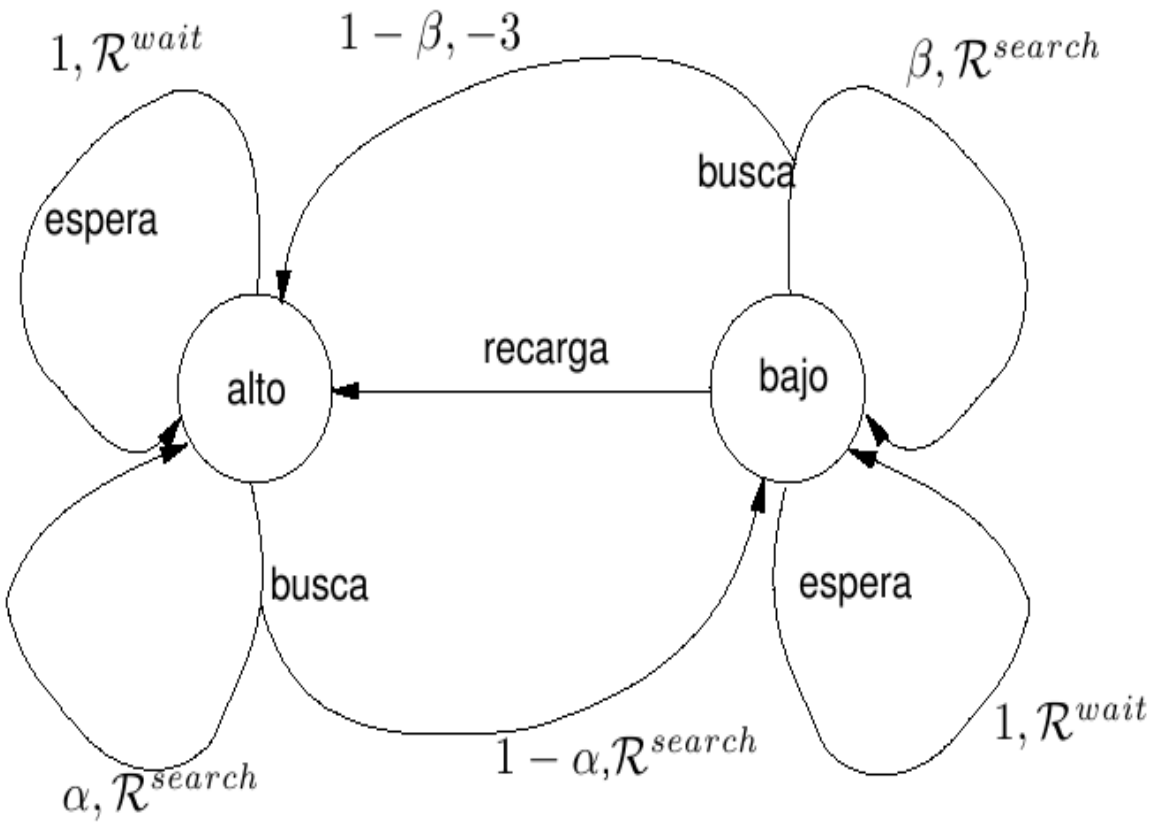
Ejemplo: decisiones

- Dependen únicamente del nivel de energía de su batería
- Consideraremos únicamente dos niveles de energía, alto y bajo: cjto. de estados es $S = \{alto, bajo\}$
- Cjto. acciones/estado es $\mathcal{A}(alto) = \{busca, espera\}$ y $\mathcal{A}(bajo) = \{busca, espera, recarga\}$
- búsqueda que comienza con un nivel alto permanece alto con una probabilidad α y cambia a bajo con una probabilidad $1 - \alpha$
- búsqueda realizada con nivel bajo mantiene la batería al mismo nivel con una probabilidad β y la agota con una probabilidad $1 - \beta$
- Cada lata recolectada resulta en una recompensa de una unidad
- rescatar al robot genera un refuerzo de -3
- Sean $\mathcal{R}^{search}$ y $\mathcal{R}^{wait}$, se tendrá que $\mathcal{R}^{search} > \mathcal{R}^{wait}$
- no puede recolectar latas mientras que se vuelve a recargar la batería y tampoco cuando la batería está agotada

Probs de transición entre estados

s	s'	a	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
alto	alto	busca	α	$\mathcal{R}^{search}$
alto	bajo	busca	$1-\alpha$	$\mathcal{R}^{search}$
bajo	alto	busca	$1-\beta$	-3
bajo	bajo	busca	β	$\mathcal{R}^{search}$
alto	alto	espera	1	$\mathcal{R}^{wait}$
alto	bajo	espera	0	$\mathcal{R}^{wait}$
bajo	alto	espera	0	$\mathcal{R}^{wait}$
bajo	bajo	espera	1	$\mathcal{R}^{wait}$
bajo	alto	recarga	1	0
bajo	bajo	recarga	0	0

MDP


$1, \mathcal{R}^{wait}$
$1-\beta, -3$
$\beta, \mathcal{R}^{search}$
espera
busca
recarga
alto
bajo
busca
espera
$\alpha, \mathcal{R}^{search}$
$1-\alpha, \mathcal{R}^{search}$
$1, \mathcal{R}^{wait}$

Políticas y funciones valor

- La política es el conjunto de decisiones sobre acciones hechas por el sistema, π
- No tiene que ser determinista, pudiéndose definir mediante una distribución de probabilidad
- Un sistema aprendedor por refuerzo busca encontrar la política que maximize la recompensa obtenida para todos los estados del entorno $x \in X$ alcanzados
- una función valor $V^{\pi}(x)$ encargada de predecir el retorno obtenido para cada estado, puede definirse para una política π como

$$V^{\pi}(x_t) = E\left\{\sum_{k=t}^{\infty} \gamma^{k-t} r_k\right\}. \quad (1)$$

- La política π^* para la cual $V^{\pi^*}(x) \geq V^{\pi}(x)$ para todo $x \in X$ se denomina política óptima
- encontrar π^* es el objetivo más ambicioso de todo sistema de aprendizaje por refuerzo

Refuerzo mediante programación dinámica

- Podemos rescribir la ecuación anterior en términos de las predicciones de la función valor sobre estados que pueden alcanzarse mediante transiciones al siguiente estado así:

$$V^{\pi}(x_i) = \sum_{x_j \in X} P(x_j|x_i, a)[r(x_i, x_j) + \gamma V^{\pi}(x_j)] \tag{2}$$

- Esta expresión es la clave del aprendizaje por refuerzo ya que nos va a permitir aproximar la función valor para una política cualquiera π, de manera iterativa.

Ecuación de Optimalidad de Bellman

- Enunciado: una función valor es optimal para cada estado $x_i \in X$ si y solo si se tiene que

$$V^{\pi^*}(x_i) = \max_{a \in A} \sum_{x_j \in X} P(x_j|x_i, a)[r(x_i, x_j) + \gamma V^{\pi^*}(x_j)]$$

- Considérese el caso en el que conocemos las probabilidades $P(x_j|x_i, a)$ para todo estado x_i, x_j y acción $a \in A$
  - Podemos determinar la función valor óptima, totalmente off-line, mediante

$$V(x_i) \leftarrow \max_{a \in A} \sum_{x_j \in X} P(x_j|x_i, a)[r(x_i, x_j) + \gamma V(x_j)]$$

  - Se exige que la ecuación se aplique en cada estado $x_i \in X$ y que
  - Cada estado debe visitarse un número suficiente de veces para asegurar convergencia

La política óptima

- Obsérvese que la política óptima se encuentra apartir de la función valor
- se hace mediante las acciones *a* que maximizan la ecuación anterior para cada estado x_i
- Estas acciones, así escogidas, se denominan *greedy* y la política que lleva a escogerlas, política *greedy*
- la política óptima π^* puede corresponderse con la política greedy dada una función valor que se haya calculado en un instante determinado sin necesariamente haber llegado a la función valor óptima
  - las acciones que actualmente tienen predicciones del mayor retorno podrían ser óptimas aun siendo no óptimas las predicciones
- No hay forma de saberlo

La política óptima (y II)

- La regla de actualización anterior puede aplicarse a los estados en cualquier orden con convergencia garantizada siempre que todos los estados se visiten repetidas veces
- Los métodos basados en PD pueden ser muy costosos computacionalmente: información puede necesitar muchas pasadas para propagarse hacia atrás en situaciones que requieren secuencias de estados muy largas hasta alcanzar estados finales.
- Solución: PD en tiempo real
  - las únicas regiones (i.e. grupos de estados) sobre las que se aprende son aquellas formadas por estados visitados realmente por el sistema durante su funcionamiento normal
  - los estados a actualizar se seleccionan mediante la ejecución de ensayos (i.e. *trials*)
  - el sistema realiza una actualización en el estado x_t y luego ejecuta la acción greedy para llegar al estado x_{t+1}

Mecanismos de selección de acciones

- Sea $Q*(a)$ el valor óptimo de retorno obtenido por la acción a
- Sea el valor estimado en el instante t, $Q_t(a)$
- la regla de selección más simple es la greedy, aquella que $Q_t(a^*) = \max_a Q_t(a)$
  - explota siempre el conocimiento actual que se tiene sobre la función valor
  - no emplea ningún tiempo extra en muestrear acciones aparentemente inferiores
- Alternativa ϵ-greedy
  - el algoritmo se comporte de manera greedy la mayor parte del tiempo
  - con una probabilidad pequeña ϵ, seleccionar una acción aleatoriamente, con probabilidad uniforme
  - Ventajas: efectiva y sencilla de implementar
  - Desventaja: escoge con igual probabilidad entre todas las acciones
- Alternativa *softmax*
  - las acciones se colocan en un ranking deacuerdo con sus valores de retorno
  - Distribución de Boltzman

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

Aprendizaje sin un modelo del mundo inicial

En el caso de que no tengamos un modelo del mundo disponible a priori, tenemos entonces dos opciones:

- Aprender un modelo apartir de la experiencia,
- Usar métodos que no requieren de ese modelo.

En ambos casos se debe explorar

- Para aprender un modelo del mundo, el sistema debe intentar diferentes acciones en cada estado para construir así una imagen de las transiciones entre estados que pueden darse
- Aunque no se aprenda el modelo, el sistema aprendedor debe explorar para actualizar su función valor de manera satisfactoria

Aprendizaje de un modelo del mundo

- Podemos aprender un modelo apartir de realizar ensayos en el entorno
  - off-line: identificación de sistemas clásica
  - de manera simultánea mientras aprendemos también la función valor
- Una vez aproximado el modelo puede usarse para realizar actualizaciones de la función valor completamente off-line
- El método básico de aprendizaje en entornos Markovianos se hace mediante la expresión

$$P(x_j|x_i, a) = \frac{n(x_i, a, x_j)}{n(x_i, a)}$$

Alternativas al aprendizaje de un modelo

Para este tipo de enfoque existen dos métodos principalmente:

- Critico heurístico adaptativo: son métodos que mantienen una estrategia de aprendizaje separada para la política actual y la función valor y
- Aprendizaje Q: métodos que aprenden una función valor que, a su vez, lleva definida implícitamente la política.

Las actualizaciones de la función valor se han de basar estrictamente en la experiencia del sistema en el entorno, de manera on-line

- en cada visita a un estado, solamente puede ejecutarse una acción
- Se trata de maximizar la utilidad de la información extraida de la experimentación para minimizar así el número de trials

Critico heurístico adaptativo (CHA)

- Estrategia basada en PD denominada de iteración sobre la política
- Funciones V y políticas se aprenden iterativamente repitiendo constantemente los dos pasos siguientes:
  1. Aprender una función valor usando una política fija y
  2. Aprender la política greedy con respecto a la actual función valor mantenida fija.
- Se seleccionan acciones mediante una distribución de probabilidad usada por el **actor**
- Se modifica la distribución mencionada con refuerzo generado por el **crítico**

CHA (II)

- V se aprende mediante una actualización incremental de sus valores según la expresión

$$V(x_t) \leftarrow V(x_t) + \alpha[r_t + \gamma V(x_{t+1}) - V(x_t)],$$

- El valor $V(x_t)$ solamente puede actualizarse ejecutando una acción y llegando tras esto al estado x_{t+1} (aquí ya se sabe qué $V(x_{t+1})$ utilizar)
- Como la política queda fijada la transición a los posibles estados x_j reflejarán las probabilidades $P(x_j|xi, a)$ y la predicción esperada $E\{V(x_i)\}$ tendrá que converger.

CHA (III)

- La política se actualiza mediante una crítica
- La crítica refleja el error entre predicciones consecutivas de V

$$\epsilon_t = r_t + \gamma V(x_{t+1}) - V(x_t)$$

- La manera exacta de utilizar ϵ depende del actor
  - En el caso de tener un conjunto de acciones discretas para cada estado
  - Sea $W(x_t, a_t)$ el peso que la acción a_t tiene con respecto al resto de acciones que pueden ejecutarse en el estado x_t

$$W(x_t, a_t) \leftarrow W(x_t, a_t) + \epsilon_t.$$

Aprendizaje Q

- Con el aprendizaje Q, usamos una función nueva que tiene en cuenta también las acciones y que puede usarse igual que V
- Se trabaja al mismo tiempo con el aprendizaje de la política y del retorno esperado a largo plazo
- Definimos la función Q como

$$Q^{\pi}(x_i, a) = \sum_{x_j \in X} P(x_j|x_i, a)[r(x_i, x_j) + \gamma V^{\pi}(x_j)]$$

- La equivalencia entre Q y V viene dada por

$$V^{\pi}(x) = \max_{a \in A} Q^{\pi}(x, a)$$

Q, actualización de valores

- Q puede aprenderse aun cuando las probabilidades de transición entre estados no son conocidas
- La actualización se hace mediante

$$Q(x_t, a) \leftarrow Q(x_t, a) + \alpha[r_t + \gamma V(x_{t+1}) - Q(x_t, a)]$$

- Si aplicamos la equivalencia anterior

$$Q(x_t, a) \leftarrow Q(x_t, a) + \alpha[r_t + \gamma \max_{a \in A} Q(x_{t+1}, a) - Q(x_t, a)].$$

- Una vez aprendida la función Q, la política se puede determinar fácilmente si en cada estado x_i se escoje la acción con mayor valor para la función Q
- Mientras se está realizando el aprendizaje, las predicciones de Q no van a ser precisas del todo
  - la política greedy no resulta apropiada
  - Compromiso exploración/explotación

Predicción mediante diferencias temporales

- Para Sutton hay dos tipos de problemas de predicción
  1. aquellos que se realizan en un solo paso
     - ★ toda la información sobre lo correcta que resulta cada predicción se revela en el instante
  2. los que se realizan en varios pasos
     - ★ no se revela hasta más de un paso después de haber hecho la predicción, revelándose, eso si, información parcial acerca de la exactitud de cada predicción en cada paso
- En un problema de predicción de varios pasos la experiencia viene dada en forma de secuencias observación-salida en la forma $x_1, x_2, \ldots, x_m, z$
- Para cada secuencia, el aprendedor va a producir una secuencia de predicciones $P_1, P_2, \ldots, P_m$, cada una de las cuales es una estimación de z
- Genéricamente, las predicciones se realizan mediante una función paramétrica, de parámetros w, por lo tanto $P(x_t, w)$.

Formulación del problema

- Cualquier procedimiento de aprendizaje se va a basar en la actualización de los valores de w
- Asumimos que w se actualiza una vez por cada par salida-secuencia de observaciones
- Para cada secuencia, se calcula Δw_t, y se actualiza w

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t . \tag{3}$$

- Desde la óptica del enfoque supervisado, cada secuencia de observaciones y su salida se transforman en pares observación-salida, esto es $(x_1, z), (x_2, z), \ldots, (x_m, z)$
  - La regla prototípica de actualización de los pesos es

$$\Delta w_t = \alpha(z - P_t)\Delta_w P_t , \tag{4}$$

siendo α el ratio de aprendizaje

Diferencias temporales

- Sea una secuencia de predicciones $P_t, P_{t+1}, \ldots,$ sobre el valor que va a tener una variable aleatoria r_T en un instante futuro T
- Una vez que se conoce r_T, las P_t para todo $t < T$ se pueden mejorar mediante

$$\Delta P_t = \alpha(r_T - P_t), \tag{5}$$

siendo, como siempre, α el ratio de aprendizaje

- Y podemos expandirla así

$$\begin{aligned} \Delta P_t &= \alpha[(P_{t+1} - P_t) + (P_{t+2} - P_{t+1}) + \cdots + (P_{T-1} - P_{T-2}) + (r_T - P_{T-1})] \\ &= \alpha \sum_{k=t}^{T-1} (P_{k+1} - P_k) \end{aligned}$$

en donde $r_T = P_T$

  - Interpretación: en cada instante t, cada predicción P_k siendo $k \geq t$ se puede actualizar haciendo uso del error TD (*Temporal Differences*) actual $(P_{t+1} - P_t)$) (se hace innecesario esperar hasta T)

Actualización de los Δw_t mediante TD

- Vamos a obtener una expresión para el aprendizaje de los w_t medinate los $(P_{t+1} - P_t)$
- Por la transparencia anterior sabemos que $z - P_t = \sum_{k=t}^{m} (P_{k+1} - P_k)$, siendo $P_{m+1} \stackrel{def}{=} z$.
- Si combinamos las ecuaciones 3 y 4 tenemos

$$\begin{aligned} w \leftarrow w + \sum_{t=1}^{m} \alpha(z - P_t)\Delta_w P_t \quad &= \quad w + \sum_{t=1}^{m} \alpha \sum_{k=t}^{m} (P_{k+1} - P_k)\Delta_w P_t \\ &= \quad w + \sum_{k=1}^{m} \alpha \sum_{t=1}^{k} (P_{k+1} - P_k)\Delta_w P_t \\ &= \quad w + \sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \Delta_w P_k \end{aligned}$$

- Con lo que al final, la cantidad con la que actualizar w_t queda

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \Delta_w P_k$$

$TD(\lambda)$

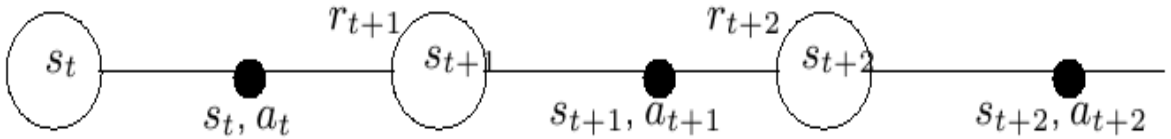
- Dependiendo de la importancia que se dé a los errores TD futuros para actualizar las predicciones actuales, tenemos una familia completa de algoritmos $TD(\lambda)$
- $0 \leq \lambda \leq 1$ es el parámetro que pondera esa importancia, mediante

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \Delta_w P_k$$

- Si $\lambda = 1$, es equivalente al procedimiento anterior
- Si $\lambda = 0$, cada actualización ΔP_t se basa solamente en el error TD del siguiente instante $(P_{t+1} - P_t)$
- Para valores de $\lambda < 1$, las actualizaciones son diferentes a las de procedimientos de aprendizaje supervisado

El algoritmo SARSA

- Los métodos de aprendizaje basados en diferencias temporales pueden también emplearse para control (i.e. para aproximar una función acción-valor)
- Tenemos que estimar $Q^{\pi}(s, a)$ para la política π actual y para todos los estados s y acciones a
- Un episodio consiste en una secuencia alternativa de estados y pares estado-acción



- La regla de actualización será:

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

  - La actualización se hará después de cada transición desde un estado no terminal s_t
  - Si s_{t+1} es terminal, entonces $Q(s_{t+1}, a_{t+1})$ se define como cero
  - Esta regla usa cada elemento de la quíntupla $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, de ahí el nombre de SARSA

SARSA

- Inicializar $Q(s, a)$ arbitrariamente
- Repetir para cada episodio
  - Inicializar s
  - Escoger a desde s usando una política derivada de Q (e.g. ϵ-greedy)
  - Repetir para cada paso del episodio
    - ★ Ejecutar acción a y observar estado s'
    - ★ Escoger a' desde s' usando una política derivada de Q (e.g. ϵ-greedy)
    - ★ $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    - ★ $s \leftarrow s'$; $a \leftarrow a'$
  - Hasta que s es terminal

Elegibilidades

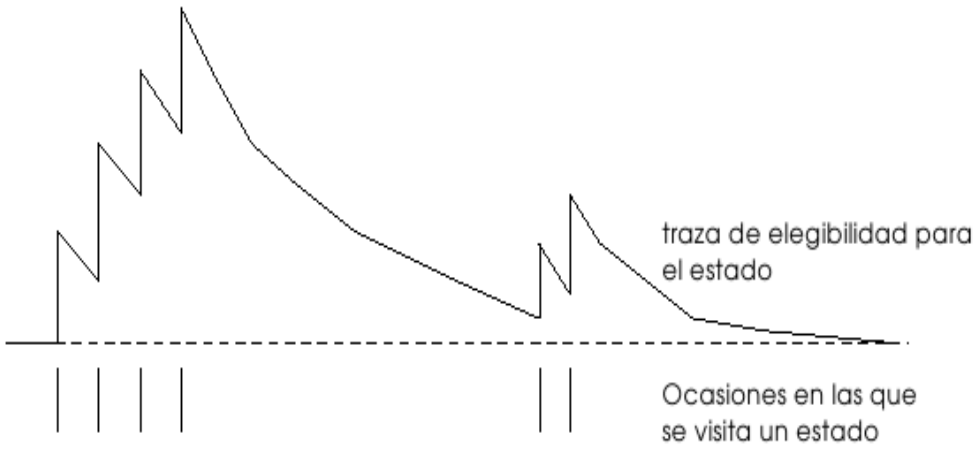
- Las eligibilidades son un mecanismo para acelerar la convergencia a una función valor aceptable ($\lambda > 0$).
- Consiste en una variable adicional asociada a cada estado, denominada traza de elegibilidad.
  - Para un estado s, su traza de elegibilidad en un instante t se va a denotar como $e_t(s) \in \Re$
  - Su actualización es la siguiente, en un instante t:

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

para todo $s \in S$ siendo γ el factor de descuento y λ el parámetro correspondiente a la familia de algoritmos TD

Evolución de elegibilidades para s

Cada paso, la traza de elegibilidad de cada estado va a decaer por el factor $\gamma\lambda$



Las trazas indican qué estados son *elegibles* para aplicar los cambios correspondientes a la función valor

SARSA(λ)

- También pueden aplicarse a control, como en el algoritmo SARSA
- Vamos a necesitar una traza de elegibilidad para cada par estado-acción
- la expresión de actualización de la función Q es la siguiente:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a), \text{ para todo par s, a}$$

en donde

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s,a)$$

y

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

SARSA(λ), Algoritmo

Inicializar $Q(s, a)$ arbitrariamente y $e(s, a) = 0$ para todo s, a
Repetir para cada episodio

- Inicializar s, a
- Repetir para cada paso del episodio
  - Ejecutar acción a y observar estado r y s'
  - Escoger a' desde s' usando una política derivada de Q (e.g. ϵ-greedy)
  - $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
  - $e(s, a) \leftarrow e(s, a) + 1$
  - Para todo 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 es terminal

Representación de funciones V y Q

- ¿cómo almacenar la información que se obtiene de la interacción con el entorno?
- El sistema ideal debe
  - preservar toda la información útil aprendida
  - generalizarla en otros estados para disminuir en la medida de lo posible el tiempo de aprendizaje
  - Compromiso almacenaje-generalización
- Aprendizaje supervisado vs. refuerzo
  - Supervisado: conjunto de entrenamiento fijo y creado de antemano, presentado de manera repetida
  - Refuerzo: datos se generan de manera continua a medida que se experimenta con el entorno (incluso una única vez); estados no visitadas muy frecuentemente pueden *olvidarse*

Aproximador de funciones V y Q

- Los siguientes son requerimientos que debe satisfacer un aproximador de funciones para ser adecuado en sistemas de aprendizaje por refuerzo que operan en entornos con un número elevado de dimensines:
  - Un algoritmo de aprendizaje que sea capaz de aprender on-line de ejemplos de entrenamiento individuales (debe prescindir del lote de ejemplares prefijado de antemano)
  - Capacidad para proporcionar salidas continuas ante entradas continuas,
  - Capacidad de generalización para permitir convergencia rápida en predicciones para todos los puntos de espacios de estados continuos.
- Ejemplo: optimización de pesos de los arcos de un MLP mediante gradiente descendente.

Métodos de aproximación tabulares

- Es posible usar simples tablas en donde almacenar simplemente todos los valores de la función Q para cada estado y acción
- El entorno debe transformarse en estados discretos, tal y como se exige en la formulaceión de entornos Markovianos
- Desventajas
  - necesidad de un enorme espacio en memoria para representar el retorno correspondiente a todos los estados junto con sus acciones
  - en una tabla no hay generalización posible y la convergencia hacia aproximaciones aceptables puede resultar considerablemente lenta

Métodos de aproximación tabulares

En el caso de espacios de estados continuos

- particionamos el espacio de estados en regiones separadas y asociamos cada bloque de estados continuos con una entrada en la tabla
- El éxito de la generalización implícita al colapsar varios estados en una entrada de la tabla depende de cómo de bien puede representarse la función aprendida con un espacio de estados continuo discretizado

Representación mediante redes neuronales

Algoritmo $TD(\lambda)$

- Sea z las salidas, en forma de predicciones del algoritmo $TD(\lambda)$ bien en forma de escalares bien en forma de vectores.
- Sea y_i^t la salida, en el instante t del nodo de salida i-ésimo del perceptrón multicapa
- Sea O el conjunto de índices para los nodos de salida de la red
- Siendo, $k \in O$, y_k^t se denota alternativamente como P_k^t para hacer énfasis en que es una predicción
- Para todo $k \in O$, z_k es la componente del vector de salida correspondiente al nodo de salida k
- z_k se predice mediante cada P_k^t, $t = 1, \ldots, m$, en donde m es el número de ejemplos en el conjunto de entrenamiento.
- Por definición, tenemos que $P_k^{m+1} = z_k$.

Más notación

- Sea $w_i ij^t$ el peso, en el instante t del arco que va desde el nodo i al j
- Sea FO_j el fan-out de la unidad j y sea FI_j el fan-in del nodo j
- Los nodos con índices en el conjunto FI_j contribuyen a la suma ponderada s_j^t en la unidad j mediante la expresión

$$s_j^t = \sum_{i \in FI_j} w_{ij}^t y_i^t$$

- La salida final del nodo j-ésimo se obtiene mediante una sigmoide tal que así:

$$y_j^t = f(s_j^t) = \frac{1}{1 + e^{-s_j^t}}$$

- Debemos distinguir dos casos de implementación, cuando $\lambda = 0$ y cuando $\lambda > 0$

Implementación para el caso $TD(0)$

- La función de error viene dada por sucesivas diferencias en las predicciones temporales

$$E^t = \sum_{k \in O} (P_k^{t+1} - P_k^t)^2$$

- Para componer la regla de actualización de los pesos, solo hemos de tener en cuenta los pesos de la red para la predicción P_k^t y no los de la predicción en el instante siguiente, P_k^{t+1}.

Implementación para el caso $TD(0)$

- La regla es

$$\begin{aligned} w_{ij}^{t+1} &= w_{ij}^t - \alpha \sum_{k \in O} \left(\frac{\partial E^t}{\partial P_k^t} \frac{\partial P_k^t}{\partial w_{ij}^t} \right) \\ &= w_{ij}^t - \alpha \frac{\partial E^t}{\partial P_j^t} \frac{\partial P_j^t}{\partial w_{ij}^t} \\ &= w_{ij}^t - \alpha \frac{\partial E^t}{\partial s_j^t} \frac{\partial s_j^t}{\partial w_{ij}^t} \\ &= w_{ij}^t + \alpha \delta_j^t y_i^t, \end{aligned}$$

en donde δ_j^t se calcula haciendo uso de la expresión

$$\delta_i^t = -\frac{\partial E^t}{\partial s_i^t} = \begin{cases} (P_i^{t+1} - P_i^t) y_i^t (1 - y_i^t), & i \in O \\ \sum_{j \in FO_i} -\frac{\partial E^t}{\delta s_j^t} \frac{\partial s_j^t}{\partial y_i^t} \frac{\partial y_i^t}{\partial s_i^t} = \sum_{j \in FO_i} \delta_j^t w_{ij}^t y_i^t (1 - y_i^t) & \text{sino} \end{cases}$$

Implementación para el caso $\lambda > 0$

- El algoritmo para el caso $TD(0)$ es bastante similar al backpropagation convencional
- El caso genérico para $TD(\lambda)$ es ligeramente diferente ya que el proceso de retropropagación del error producirá una eligibilidad para cada parámetro a optimizar (i.e. peso)
- Cuando se calcula una diferencia temporal, se propaga a todos los pesos y esa propagación se ha de combinar con las elegibilidades.
- La regla de actualización de pesos es ahora

$$w_{ij}^{t+1} = w_{ij}^{t} + \alpha \sum_{k \in O} (P_k^{t+1} - P_k^t) e_{ijk}^t,$$

en donde e_{ijk}^t es la elegibilidad k-ésima en el instante t para el peso del arco desde el nodo i al nodo j

  - Para el cálculo de e_{ijk}^t se obtiene la suma desde el primer instante, hasta el actual t de cada término $\partial P_k^n / \partial w_{ij}^n$ que es, en definitiva, la variación de la predicción en el instante t para el nodo de salida k, con respecto al arco del cual se calcula la misma.

Implementación para el caso $\lambda > 0$ (II)

De manera iterativa, las eligibilidades se calculan usando las expresiones

$$\begin{aligned} e_{ijk}^{t+1} &= \lambda e_{ijk}^{t} + \frac{\partial P_k^{t+1}}{\partial w_{ij}^{t+1}} \\ &= \lambda e_{ijk}^{t} + \frac{\partial P_k^{t+1}}{\partial s_j^{t+1}} \frac{\partial s_j^{t+1}}{\partial w_{ij}^{t+1}} \\ &= \lambda e_{ijk}^{t} + \delta_{kj}^{t+1} y_i^{t+1}, \end{aligned}$$

en donde $\delta_{kj}^{t+1} = \partial P_k^{t+1} / \partial s_j^{t+1}$ se calcula mediante un procedimiento recursivo que propaga el error hacia atrás como sigue

$$\delta_{ki}^{t} = \frac{\partial P_k^t}{\partial s_i^t} = \begin{cases} y_i^t(1 - y_i^t), & \text{si } k = i \\ 0, & \text{si } k \in O \text{ y } k \neq i \\ \sum_{j \in FO_i} \frac{\partial P_k^t}{\partial s_j^t} \frac{\partial s_j^t}{\partial y_i^t} \frac{\partial y_i^t}{\partial s_i^t} = \sum_{j \in FO_i} \delta_{kj}^t w_{ij}^t y_i^t (1 - y_i^t) & \text{, si no} \end{cases}$$

Aprendizaje por refuerzo de sistemas de inferencia difusos

- Los controladores difusos son sistemas expertos basados en reglas difusas
- En el control difuso las reglas que gobiernan el control se construyen a mano incorporando conocimiento de un experto
- La extracción natural del conocimiento experto es una tarea difícil de realizar
- Proponemos un algoritmo de ajuste de los consecuentes (crisp) para aprendizaje on-line

Crítico heurístico adaptativo difuso

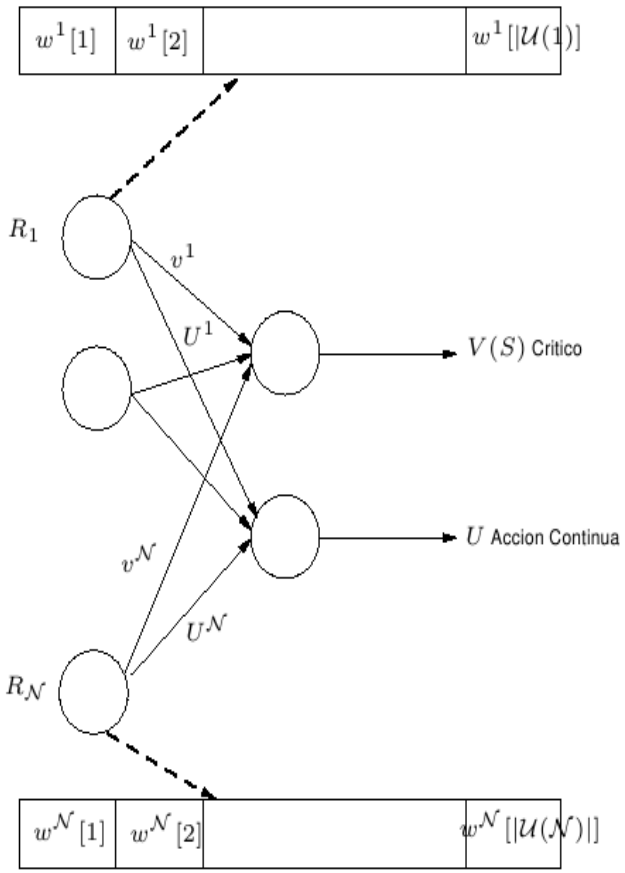
- En la arquitectura de aprendizaje por refuerzo del crítico heurístico adaptativo teníamos un actor y un crítico.
  - El actor se encargaba de, mediante el uso de una política fija, ejecutar la acción escogida probabilísticamente.
  - El crítico generaba una determinada recompensa, positiva o negativa, que actualizaba directamente la política.
- En este caso vamos a usar un FIS para representar la función valor, en lugar de una tabla de tipo *look-up*.
- El crítico se modelará mediante el vector de conclusiones asociadas al estado, v
  - Cada componente es la contribución al valor de aproximación de V de una regla del FIS
- El actor tiene un conjunto discreto de acciones para cada estado
  - En realidad, cada regla tiene un conjunto de acciones posible con sus correspondientes probabilidades

Reglas del sistema difuso

Cada regla R_i tiene:

- un valor v^i que aproxima a a la función valor V^*,
- un conjunto discreto de acciones $\mathcal{U}(i)$ y
- un vector de parámetros w^i que proporciona las probabilidades de las diferentes acciones disponibles para la regla, y así aproximar una política optimal.

Esquema del Sistema de Inferencia Difuso



El crítico

- Aproxima la función valor $V(S)$
- En el instante de tiempo t, el valor $V_t(s_t)$ se calcula con la salida de todas las reglas (i.e. el vector de conclusiones v) mediante

$$V_t(s_t) = \sum_{R_i \in \mathcal{A}_t} v_t^i \alpha_{R_i} = v_t \Phi_t^T,$$

  - s_t el estado en que se encuentra el aprendedor en el instante t
  - α_{R_i} el grado de disparo de la regla R_i aplicando la T-norma del producto

- El error que hay que aplicar a la función valor viene dado por

$$\epsilon_{t+1} = r_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$$

  siendo γ el factor de olvido o descuento

- El ajuste de los valores del vector de conclusiones v, se hace aplicando la regla delta

$$v_{t+1} = v_t + \beta \epsilon_{t+1} \Phi_t$$

  en donde β es el ratio de aprendizaje del crítico.

El actor

- Para representar la política, se usa un vector de parámetros w^i para cada regla R_i difusa.
- Cuando la regla se activa, la acción local de R_i se toma en cuenta para componer la acción global

$$U_t(S_t) = \sum_{R_i \in \mathcal{A}_t} \epsilon - greedy_{\mathcal{U}(i)}(w_t^i)\alpha_{R_i},$$

- Para el ajuste de w_i, la regla de aprendizaje del actor se basa en el uso de la cantidad ϵ_{t+1}
  - Si la del crítico es una medida relevante (i.e. corresponde a la función valor con política optimal en el instante t), un error TD positivo implica que la acción que se acaba de aplicar es mejor que la correspondiente a la política t-optimal. Por lo tanto, es necesario incrementar la calidad de esa acción para el estado correspondiente.
  - si el error TD es negativo, se debe decrementar la calidad de la acción porque esa acción ha llevado al sistema a un estado con una función valor peor que la prevista.

  Finalmente

$$w_{t+1}^i(U_t^i) = w_t^i(U_i^i) + \epsilon_{t+1}\alpha_{R_i}, \forall R_i \in \mathcal{A}_t,$$

siendo U_t^i la acción escogida mediante la estrategia ϵ-greedy para la regla R_i en el instante t

Trazas de elegibilidad

- El procedimiento que acabamos de describir únicamente modifica los parámetros de las reglas activadas en el instante de tiempo t ($TD(0)$)
- Aplicaremos elegibilidades a los términos que intervienen en la actualización de v_{t+1} y w_{t+1}
- Si planteamos un $TD(\lambda)$ con $\lambda \in [0,1]$, la traza del crítico, $\overline{\Phi}_t$ en el instante t se define

$$\overline{\Phi}_t = \sum_{n=0}^{t} (\gamma\lambda)^{t-n}\Phi_n = \Phi_t + \gamma\lambda \sum_{n=0}^{t-1} (\gamma\lambda)^{(t-1)-n}\Phi_n,$$

con lo que al final tenemos que

$$\overline{\Phi}_t = \Phi_t + \gamma\lambda\overline{\Phi}_{t-1}.$$

Con esta traza llevaremos la cuenta de la salida que ha ido obteniendo cada regla.

Trazas de elegibilidad (II)

- Ahora determinamos las elegibilidades para la actualización de w^i. Sea $e_t^i(U^i)$ la traza asociada con la acción discreta U^i de la regla R_i en el instante T, entonces

$$e_t^i(U^i) = \begin{cases} \lambda' e_{t-1}^i(U^i) + \Phi_t^i & U^i = U_t^i \\ \lambda' e_{t-1}^i(U^i) & sino \end{cases}$$

  siendo λ' el del actor.

- Las ecuaciones de actualización se modifican para obtener

$$\begin{aligned} v_{t+1} &= v_t + \beta \epsilon_{t+1} \overline{\Phi}_t. \\ w_{t+1} &= w_t + \epsilon_{t+1} e_t. \end{aligned}$$

Algoritmo Final

Inicialización: Sea $t+1$ el instante de tiempo actual, el aprendedor ha llevado acabo la acción U_t escogida mediante la estrategia ϵ-greedy, recibiendo una señal de refuerzo r_{t+1} deacuerdo con la transición al siguiente estado. Tras el paso de fuzzificación y el cálculo de la fuerza de cada regla (i.e. Φ_{t+1} describe el nuevo estado S_{t+1}), los cinco pasos son

1. Estimación de la función valor correspondiente al estado actual: $V_t(S_{t+1}) = v_t \Phi_{t+1}^T$
2. Cálculo del error TD: $\epsilon_{t+1} = r_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$
3. Ajuste de parámetros mediante las ecuaciones

$$\begin{aligned} v_{t+1} &= v_t + \beta \epsilon_{t+1} \overline{\Phi}_t . \\ w_{t+1} &= w_t + \epsilon_{t+1} e_t . \end{aligned}$$

4. Estimación de la función valor correspondiente al estado actual con el vector de conclusiones nuevo v_{t+1} para ser usada en la siguiente iteración, en el cálculo del error TD
5. Para cada regla hacer
   1. Elección de la acción local
   2. Actualización de la traza de elgibilidad
6. Computación y disparo de la acción global U_{t+1}

Conclusiones

- El aprendizaje por refuerzo permite ajuste de modelos on-line
- Podemos trabajar sin un modelo del mundo
- Adecuado cuando no se dispone de casos obtenidos en un proceso off-line
- Incluso aplicable a modelos difusos