Agentes Inteligentes

Franz Mayr
mayr@ort.edu.uy

Universidad ORT Uruguay
2 de mayo de 2023

8. Métodos de aproximación de la función de valor

Repaso: Métodos Tabulares

Los métodos tabulares estiman las funciones de valor mediante tablas V y Q. Por ejemplo, la estimación Monte Carlo de v_π:

Inicializar:
    $V(s) \in \mathbb{R}$ arbitrariamente, $\forall s \in \mathcal{S}$
    $Retornos(s) \leftarrow$ lista vacía, $\forall s \in \mathcal{S}$
Repetir:
    Generar un episodio según π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
    $G \leftarrow 0$
    Para cada paso del episodio, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Si S_t no aparece en $S_0, S_1, \ldots, S_{t-1}$:
            Agregar G a $Retornos(S_t)$
            $V(S_t) \leftarrow$ promedio$(Retornos(S_t))$

Repaso: Métodos Tabulares

Los métodos tabulares estiman las funciones de valor mediante tablas V y Q. Por ejemplo, la estimación Monte Carlo de v_π:

Inicializar:
    $V(s) \in \mathbb{R}$ arbitrariamente, $\forall s \in \mathcal{S}$
    $Retornos(s) \leftarrow$ lista vacía, $\forall s \in \mathcal{S}$
Repetir:
    Generar un episodio según π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
    $G \leftarrow 0$
    Para cada paso del episodio, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Si S_t no aparece en $S_0, S_1, \ldots, S_{t-1}$:
            Agregar G a $Retornos(S_t)$
            $V(S_t) \leftarrow$ promedio$(Retornos(S_t))$

Este enfoque no sirve cuando $\mathcal{S}$ es demasiado grande.

Ejemplo: La cantidad de posibles imágenes en la pantalla del Mario Bros es un número combinatorio gigantesco.

Aproximación de funciones de valor

Dada una política π, vamos a aproximar $v_\pi(s)$ mediante $\hat{v}(s, \mathbf{w})$, una función con parámetros $\mathbf{w} \in \mathbb{R}^d$, con $d \ll |\mathcal{S}|$.

Ejemplo: $\hat{v}$ puede ser una red neuronal y $\mathbf{w}$ sus pesos.

Como $d \ll |\mathcal{S}|$, nuestra aproximación $\hat{v}$ tendrá cierto error.

Aproximación de funciones de valor

Dada una política π, vamos a aproximar $v_\pi(s)$ mediante $\hat{v}(s, \mathbf{w})$, una función con parámetros $\mathbf{w} \in \mathbb{R}^d$, con $d \ll |\mathcal{S}|$.

Ejemplo: $\hat{v}$ puede ser una red neuronal y $\mathbf{w}$ sus pesos.

Como $d \ll |\mathcal{S}|$, nuestra aproximación $\hat{v}$ tendrá cierto error.

Definimos el mean squared value error como:

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

donde $\mu(s)$ es una distribución $(\mu(s) \geq 0, \sum_s \mu(s) = 1)$ que indica la importancia relativa de cada estado: define cuáles estados nos parecen más relevantes en la tarea.

Suele tomarse $\mu(s)$ como la fracción de tiempo que se pasa en s.

Aproximación de funciones de valor

En un instante de tiempo t, usamos el vector de parámetros $\mathbf{w}_t$ para estimar el valor de cada estado $s \in \mathcal{S}$:

$$\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$$

Aproximación de funciones de valor

En un instante de tiempo t, usamos el vector de parámetros $\mathbf{w}_t$ para estimar el valor de cada estado $s \in \mathcal{S}$:

$$\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$$

Supongamos que observamos el valor real $v_\pi(S_t)$ para algún estado S_t.

¿Cómo podemos usar esta nueva información para mejorar nuestra aproximación?

Aproximación de funciones de valor

Querríamos reducir $[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2$ (el error cuadrático de la estimación para S_t), y así reducir $\overline{\text{VE}}(\mathbf{w}_t)$.

Aproximación de funciones de valor

Querríamos reducir $[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2$ (el error cuadrático de la estimación para S_t), y así reducir $\overline{\text{VE}}(\mathbf{w}_t)$.

Si $\hat{v}(s, \mathbf{w})$ es diferenciable con respecto a $\mathbf{w}$, podemos usar gradiente descendente estocástico (SGD) para ajustar $\mathbf{w}$ un poco en la dirección en que más se reduce el error:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2$$

Aproximación de funciones de valor

Querríamos reducir $[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2$ (el error cuadrático de la estimación para S_t), y así reducir $\overline{\text{VE}}(\mathbf{w}_t)$.

Si $\hat{v}(s, \mathbf{w})$ es diferenciable con respecto a $\mathbf{w}$, podemos usar gradiente descendente estocástico (SGD) para ajustar $\mathbf{w}$ un poco en la dirección en que más se reduce el error:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2 \\ &= \mathbf{w}_t + \alpha\left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

donde $\alpha \in (0, 1]$ dice cuánto avanzar en cada paso (*step size*).

Aproximación de funciones de valor

Entonces, en cada paso vamos a ajustar $\mathbf{w}_t$ de esta manera:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t)$$

Aproximación de funciones de valor

Entonces, en cada paso vamos a ajustar $\mathbf{w}_t$ de esta manera:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t)$$

En general, si llamamos $U_t \in \mathbb{R}$ al objetivo del método, tenemos:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t) \right] \nabla \hat{v}(S_t, \mathbf{w}_t)$$

Por ejemplo, en los métodos Monte Carlo: $U_t \doteq G_t$

Ejemplo: Aproximación mediante una función lineal

Supongamos que $\hat{v}(s, \mathbf{w})$ es una función lineal en los pesos $\mathbf{w}$:

$$\hat{v}(s, \mathbf{w}) = \sum_{i=1}^{d} w_i \, x_i(s) = \mathbf{w}^\top \cdot \mathbf{x}(s)$$

donde $x_i : S \to \mathbb{R}$ son las características (*features*) de un estado.

Ejemplo: Aproximación mediante una función lineal

Supongamos que $\hat{v}(s, \mathbf{w})$ es una función lineal en los pesos $\mathbf{w}$:

$$\hat{v}(s, \mathbf{w}) = \sum_{i=1}^{d} w_i \, x_i(s) = \mathbf{w}^\top \cdot \mathbf{x}(s)$$

donde $x_i : S \to \mathbb{R}$ son las características (*features*) de un estado.

En este caso:

$$\begin{aligned} \nabla \hat{v}(s, \mathbf{w}) &\doteq \left(\frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_1}, \frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_2}, \ldots \right)^\top \\ &= \big(x_1(s), x_2(s), \ldots\big)^\top = \mathbf{x}(s) \end{aligned}$$

Ejemplo: Aproximación mediante una función lineal

Supongamos que $\hat{v}(s, \mathbf{w})$ es una función lineal en los pesos $\mathbf{w}$:

$$\hat{v}(s, \mathbf{w}) = \sum_{i=1}^{d} w_i \, x_i(s) = \mathbf{w}^\top \cdot \mathbf{x}(s)$$

donde $x_i : S \to \mathbb{R}$ son las características (*features*) de un estado.

En este caso:

$$\begin{aligned} \nabla \hat{v}(s, \mathbf{w}) &\doteq \left(\frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_1}, \frac{\partial \hat{v}(s, \mathbf{w})}{\partial w_2}, \ldots \right)^\top \\ &= \big(x_1(s), x_2(s), \ldots\big)^\top = \mathbf{x}(s) \end{aligned}$$

Entonces:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t, \mathbf{w}_t) \Big] \cdot \mathbf{x}(S_t) \\ &= \mathbf{w}_t + \alpha \Big[U_t - \mathbf{w}_t^\top \mathbf{x}(S_t) \Big] \cdot \mathbf{x}(S_t) \end{aligned}$$

Algoritmo gradiente Monte Carlo para estimar v_π

$\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ es una función parametrizable
con pesos $\mathbf{w} \in \mathbb{R}^d$ (ej.: red neuronal)

Inicializar $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente

Repetir:

Generar un episodio según π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.

$G \leftarrow 0$

Para cada paso del episodio, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma\, G + R_{t+1}$

Si S_t no aparece en $S_0, S_1, \ldots, S_{t-1}$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\,[G - \hat{v}(S_t, \mathbf{w})]\,\nabla \hat{v}(S_t, \mathbf{w})$$

Algoritmo gradiente Monte Carlo para estimar v_π

$\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ es una función parametrizable
con pesos $\mathbf{w} \in \mathbb{R}^d$ (ej.: red neuronal)

Inicializar $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente

Repetir:

Generar un episodio según π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.

$G \leftarrow 0$

Para cada paso del episodio, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma\, G + R_{t+1}$

Si S_t no aparece en $S_0, S_1, \ldots, S_{t-1}$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\,[G - \hat{v}(S_t, \mathbf{w})]\,\nabla \hat{v}(S_t, \mathbf{w})$$

En Monte Carlo, $U_t \doteq G_t$ es un estimador no sesgado:

$$v_\pi(s) \doteq \mathbb{E}_\pi(G_t \,|\, S_t = s)$$

Entonces, el algoritmo converge a una aproximación localmente óptima de $v_\pi(s)$.

Algoritmo semi-gradiente TD(0) para estimar v_π

En los métodos con bootstrapping:

$$U_t \doteq R_{t+1} + \gamma\, \hat{v}(S_{t+1}, \mathbf{w}_t)$$

Notar que U_t no es independiente de $\mathbf{w}_t$.

Algoritmo semi-gradiente TD(0) para estimar v_π

En los métodos con bootstrapping:

$$U_t \doteq R_{t+1} + \gamma\, \hat{v}(S_{t+1}, \mathbf{w}_t)$$

Notar que U_t no es independiente de $\mathbf{w}_t$.

Entonces, no vale este paso que hicimos antes:

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla \left[U_t - \hat{v}(S_t, \mathbf{w}_t)\right]^2 \\
&= \mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t)\right] \nabla \hat{v}(S_t, \mathbf{w}_t)
\end{aligned}$$

Algoritmo semi-gradiente TD(0) para estimar v_π

En los métodos con bootstrapping:

$$U_t \doteq R_{t+1} + \gamma\, \hat{v}(S_{t+1}, \mathbf{w}_t)$$

Notar que U_t no es independiente de $\mathbf{w}_t$.

Entonces, no vale este paso que hicimos antes:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\left[U_t - \hat{v}(S_t, \mathbf{w}_t)\right]^2 \\ &= \mathbf{w}_t + \alpha\left[U_t - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

En este caso, el método no es realmente un *gradiente*: toma en cuenta el efecto de ajustar $\mathbf{w}$ sobre la estimación $\hat{v}(S_t, \mathbf{w}_t)$, pero no sobre el objetivo U_t.

Por eso se lo llama semi-gradiente. Este método puede funcionar, pero sin buenas garantías de convergencia en general.

Algoritmo semi-gradiente TD(0) para estimar v_π

$\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ es una función parametrizable
con pesos $\mathbf{w} \in \mathbb{R}^d$ (ej.: red neuronal)

Inicializar $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente

Repetir:
- Inicializar S
- Repetir:
  - $A \leftarrow$ acción desde S según π
  - Ejecutar la acción A; observar R, S'
  - $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \, \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}) \right] \nabla \hat{v}(S, \mathbf{w})$
  - $S \leftarrow S'$
- hasta que S sea terminal

Algoritmos para estimar q_π

Extensión natural de los métodos para estimar q_π:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Algoritmos para estimar q_π

Extensión natural de los métodos para estimar q_π:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Para Monte Carlo: $U_t \doteq G_t$
(gradiente)

Para Sarsa: $U_t \doteq R_{t+1} + \gamma\, \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$
(semi-gradiente)

Estos algoritmos convergen del mismo modo que los algoritmos para estimar v_π.

Sarsa semi-gradiente para estimar q_* (on-policy)

$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$ es una función parametrizable
con pesos $\mathbf{w} \in \mathbb{R}^d$ (ej.: red neuronal)

Inicializar $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente

Repetir:
  Inicializar S
  $A \leftarrow$ acción desde S según política ε-greedy basada en $\hat{q}(S, \cdot, \mathbf{w})$
  Repetir:
    Ejecutar la acción A; observar R, S'
    $A' \leftarrow$ acción desde S' según polít. ε-greedy basada en $\hat{q}(S, \cdot, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \, \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w}) \right] \nabla \hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$
  hasta que S sea terminal

Q-learning semi-gradiente para estimar q_* (off-policy)

$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$ es una función parametrizable
con pesos $\mathbf{w} \in \mathbb{R}^d$ (ej.: red neuronal)

Inicializar $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente

Repetir:
  Inicializar S
  Repetir:
    $A' \leftarrow$ acción desde S' según polít. ε-greedy basada en $\hat{q}(S, \cdot, \mathbf{w})$
    Ejecutar la acción A; observar R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\right] \nabla \hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
  hasta que S sea terminal

Q-learning semi-gradiente para estimar q_* (off-policy)

$\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$ es una función parametrizable
con pesos $\mathbf{w} \in \mathbb{R}^d$ (ej.: red neuronal)

Inicializar $\mathbf{w} \in \mathbb{R}^d$ arbitrariamente

Repetir:
  Inicializar S
  Repetir:
    $A' \leftarrow$ acción desde S' según polít. ε-greedy basada en $\hat{q}(S, \cdot, \mathbf{w})$
    Ejecutar la acción A; observar R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \operatorname{máx}_a \hat{q}(S', a, \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\right] \nabla \hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
  hasta que S sea terminal

En estos algoritmos no hay garantía de convergencia. Igual se los usa mucho. Observación: Los algoritmos tabulares son casos particulares, y sí poseen buenas propiedades de convergencia.

Ejemplo: Atari con Deep Q-Learning: Mnih *et al.* 2013

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Resumen: Métodos de aproximación de la función de valor

- Aproximación de $v_\pi(s)$ mediante una función $\hat{v}_\pi(s, \mathbf{w})$
- Funciones lineales y no lineales (ej: redes neuronales)
- Objetivo de predicción $\overline{\text{VE}}(\mathbf{w})$
- Métodos gradiente y semi-gradiente para estimar v_π y q_π
- Métodos Sarsa y Q-learning semi-gradiente para estimar q_*

Próximas clases:

- Bootstrapping de n pasos
- Aprendizaje y planificación: Dyna-Q
- Métodos de aproximación de política