08MIAR - Aprendizaje por refuerzo

Sesión 4 – Algoritmos base: Policy Gradient

Curso 21/22

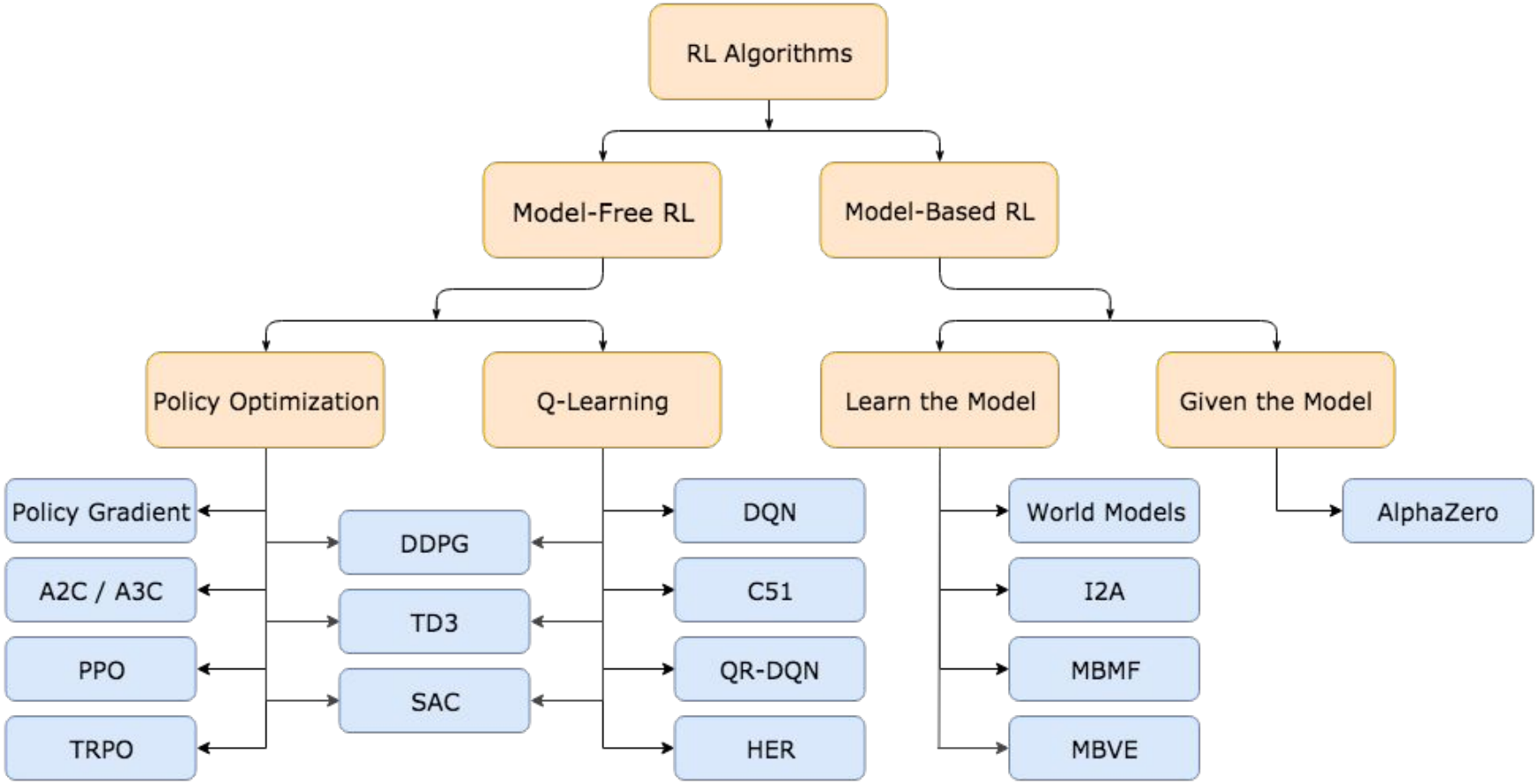
Universidad Internacional de Valencia

De:
Planeta Formación y Universidades

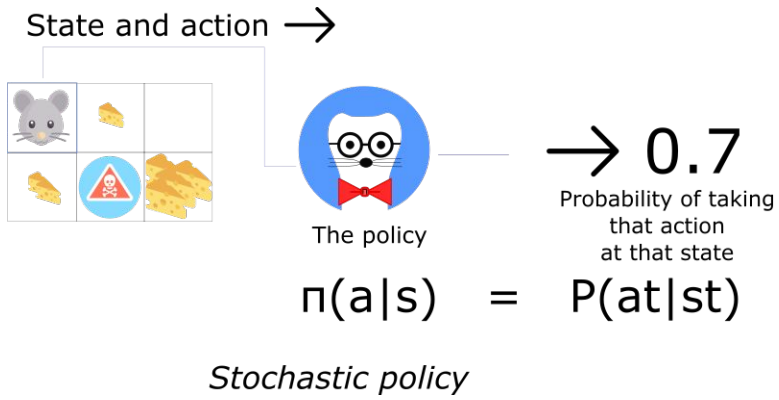
Índice

Índice

Definición Policy Gradient



Definición Policy Gradient



En el algoritmo anterior, nuestro objetivo era aproximar una función, la función Q.
Con esta función podríamos encontrar la estrategia óptima usando los valores de recompensa esperada a partir de la relación entre estado y acción.

Ahora, en vez de centrarnos en esos valores, trabajaremos directamente sobre la estrategia, es decir, sobre la distribución de probabilidades de las acciones siguiendo la estrategia que se está aprendiendo.

https://cdn-images-1.medium.com/max/1600/1*YCABimP7x1wZZZKqz2CoyQ.png

Definición Q-learning

Optimizar directamente la estrategia es una técnica ampliamente reconocida dentro de los algoritmos de aprendizaje por refuerzo. Algunos de los métodos que actualmente son el estado del arte en muchos problemas tienen como su base el algoritmo de *Policy Gradient*.

El nombre *Policy Gradient* viene justo de la idea de atacar directamente a la *policy* para maximizar la recompensa esperada, de ahí que se busca el gradiente de la *policy* para maximizar ese valor.

¿Cuál será la forma de maximizar la recompensa esperada? Intuitivamente, cuando estemos en un estado tendremos a nuestra disposición un conjunto de acciones. De entre estas acciones, potenciaremos las que nos devuelvan una recompensa positiva mientras que evitaremos (o ponderamos negativamente) las que nos devuelvan una recompensa negativa.

Índice

Conceptos importantes

Al ser nuestro objetivo modelar la selección de acciones, en cada *step* obtendremos una lista de probabilidades, cada una relacionada con las acciones disponibles para el agente en un estado.

Las probabilidades de cada acción para ese estado se irán actualizando, acorde al factor con el que potenciamos la acción. Este proceso está guiado por la maximización de la recompensa esperada.

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + r^{T-t} r_T$$

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta]$$

$$= \sum_{t=i}^{T-1} P(s_t, a_t | \tau) r_{t+1}$$

Conceptos importantes

Si en las DQN usábamos la ecuación de Bellman para encontrar nuestra estrategia óptima, en *Policy Gradients* usaremos una función similar a la usada en el ámbito de Deep Learning, *Cross-Entropy function*.

$$\mathcal{L}(y - \hat{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

Siguiendo con la idea de potenciar las acciones “buenas”, en su definición más básica usaremos como ponderación la propia recompensa de tomar una acción determinada. Esta idea está bien como base para definir nuestros algoritmos, pero veremos cómo se producen algunas situaciones no deseadas con este enfoque que tendremos que evitar.

$$\mathbb{E}[f(x)] = \sum_{x} P(x) f(x)$$

Conceptos importantes

Algunas de las ventajas de usar Policy Gradients son:

1) Los métodos basados en Policy Gradient tienen mejores propiedades desde el punto de vista de la convergencia, ya que se optimizan los parámetros de la policy directamente en vez de utilizar los valores de las acciones.

2) Este tipo de métodos están más preparados para abordar retos que contengan un espacio de acciones *muy* grande o que trabajen en espacio de acciones continuos.

3) Algoritmos basados en Policy Gradient pueden aprender estrategias estocásticas, lo que implica que no se necesite utilizar el enfoque de exploración/explotación que vimos en DQN (más sobre esto en la siguiente sección).

Índice

Deep Policy Gradient

En el caso de Policy Gradient, la combinación con técnicas de Deep Learning es similar al ejemplo visto en DQN.

Como se ha comentado en las secciones anteriores, el detalle más importante es el hecho de que ahora estamos modelando la probabilidad de las acciones y no el valor de recompensa esperado.

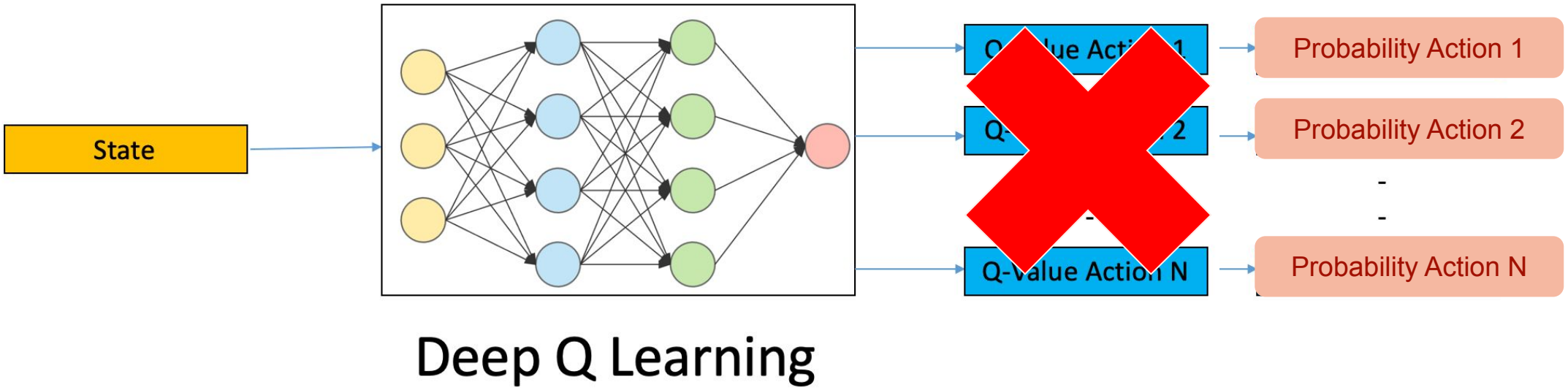
Por ello, normalmente encontraremos como función de activación en la salida del modelo de Deep Learning una función *Softmax*, para poder modelar nuestra capa de salida como una distribución de probabilidad. Si el conjunto de acciones es reducido, podemos encontrar otras opciones como *Sigmoid*.

Deep Policy Gradient

Otro detalle importante es cómo se realiza el proceso de exploración. A diferencia de DQN, en Policy Gradient (y toda su familia de algoritmos) la exploración va intrínseca en el modo en el que se seleccionan las acciones ya que se escogen siguiendo una **distribución aleatoria de probabilidad**.

En cada step de la ejecución, se calcularán las probabilidades de las acciones para el estado en el que se encuentra el agente y se utilizará esa distribución de probabilidades como los pesos de la selección de la acción de manera aleatoria.

Deep Policy Gradient



https://cdn.analyticsvidhya.com/wp-content/uploads/2019/04/Screenshot-2019-04-16-at-5.46.01-PM.png

Índice

Proceso de aprendizaje

Como indicábamos hace unas diapositivas, en *Policy Gradients* usaremos un factor que nos dirá cómo de bueno es tomar una acción o no.

Ese factor de escala será *Rt* y decidirá como la probabilidad *P(a)* debe cambiar para maximizar la recompensa futura esperada.

“Si una acción es buena (por ejemplo, *Rt* con valor muy grande), *P(a)* será multiplicado por un peso grande. Por otro lado, si la acción es mala, la probabilidad *P(a)* se descartará. Eventualmente, acciones buenas incrementarán su probabilidad para ser seleccionadas en iteraciones futuras.”

$$\nabla_\theta E[R_t] = E[\nabla_\theta log P(a) R_t]$$

Proceso de aprendizaje

La función de coste que se utiliza en Policy Gradient surge de la aplicación de lo que se conoce como *Log Derivative Trick.* Básicamente, nuestra función de coste calcula el *ratio* entre la actualización de la policy y la probabilidad de la acción tomada, para de esa manera no impactar negativamente al proceso de optimización cuando una acción tiene probabilidad alta:

$$\nabla \ln f(x) = \frac{\nabla f(x)}{f(x)} \longrightarrow \frac{\nabla \pi_\theta(a|s)}{\pi_\theta(a|s)} = \nabla_\theta \log \pi_\theta(s|a)$$

Ese *ratio* lo podemos transformar en el logaritmo que utilizaremos en la función de coste, manteniendo las mismas propiedades desde un punto de vista matemático para nuestro proceso de optimización.

https://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/

Proceso de aprendizaje

Durante el proceso de aprendizaje debemos tener algunos conceptos y situaciones presentes, para entender qué está ocurriendo:

- Una de las primeras decisiones que debemos tomar es "¿Cuántos *steps* vamos a usar para ir modificando la estrategia?". El proceso de aprendizaje se puede ver más o menos impactado dependiendo del número de iteraciones que realicemos para ir almacenando nuestra experiencia.

- Además, al trabajar con la trayectoria, las recompensas obtenidas se procesarán siguiendo un enfoque conocido como *discounted rewards*. Al tener una trayectoria finita, utilizaremos las recompensas en sentido inverso para ir estimando la recompensa esperada en los siguientes estados y de esta forma poder ponderar las acciones de manera adecuada.

- Por otro lado, usar la recompensa como factor de las probabilidades de las acciones produce una varianza en los datos muy grande. Tened en cuenta que con esta definición la probabilidad de una acción en un estado puede cambiar dependiendo de si la recompensa cambia también. Esto dificulta el aprendizaje ya que no se encuentra una correlación entre estado y probabilidad de acción fácilmente. Esta situación se puede dar en muchos escenarios, sobre todo en las simulaciones basadas en videojuegos.

Proceso de aprendizaje

Para suavizar el problema con el uso de la recompensa como factor, hay otras definiciones como:

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function)}. \tag{3}$$

Índice

Algoritmo: REINFORCE

function REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

for $t = 1$ to $T - 1$ **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

end for

end for

return θ

end function

Índice

Algoritmo: Vanilla Policy Gradient

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b

for iteration=$1, 2, \ldots$ **do**

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,

summed over all trajectories and timesteps.

Update the policy, using a policy gradient estimate $\hat{g}$,

which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$

end for

Índice

Conclusiones

- Policy Gradient es uno de los algoritmos base dentro del aprendizaje por refuerzo. Este algoritmo es el origen de algunos de los algoritmos más potentes actualmente. Es un algoritmo de tipo on-policy.

- En comparación con DQN, algunas de las características a destacar son una mayor capacidad de convergencia, apto para trabajar con espacios de acciones grandes (y continuos) y posibilidad de aprender policies estocásticas.

- La función de coste en Policy Gradient se centra en ir optimizando la propia policy aplicando *gradient ascent* sobre la probabilidad de la acción seleccionada y su recompensa obtenida.

- Al ser un algoritmo base, veremos en las siguientes sesiones de la asignatura las evoluciones que se han ido produciendo para estabilizar y mejorar el proceso de aprendizaje del agente.

Bibliografía recomendada

- “Reinforcement Learning: An Introduction”, Sutton y Barto:
http://incompleteideas.net/book/bookdraft2017nov5.pdf
(Capítulo 13, *Policy Gradient Methods)*

- *An Intuitive explanation on Policy Gradients, Adrien Lucas, Towards data science / Medium*
https://towardsdatascience.com/an-intuitive-explanation-of-policy-gradient-part-1-reinforce-aa4392cbfd3c

Universidad Internacional de Valencia

universidadviu.com

De:
Planeta Formación y Universidades