Introducción a aprendizaje por refuerzo
Sesión 3

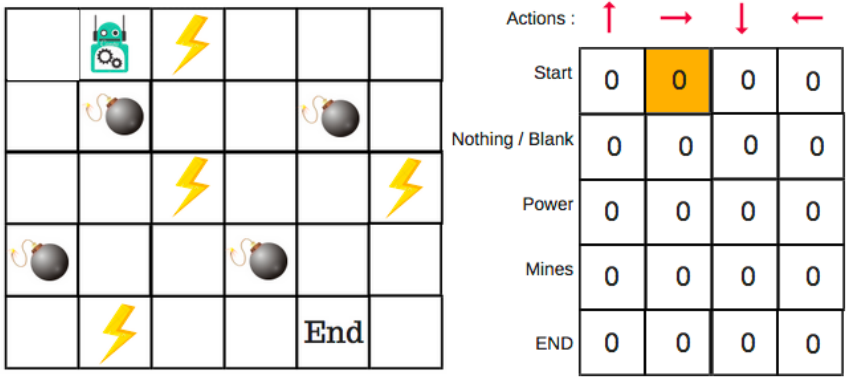
Gabriel Muñoz
Máster en Inteligencia Artificial, 2020-2021

viu Universidad Internacional de Valencia

Índice

Índice

Definición *Q-learning*



	Actions : ↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

Cuando hablamos de DQN, tenemos que comenzar hablando de Q-learning.

Q-learning es un algoritmo de aprendizaje por refuerzo que se basa en el aprendizaje a partir de diferencias temporales. Matemáticamente, su enfoque es como una cadena de Markov, en el que la transición entre estados sólo depende de la información que tenemos en el estado actual.

Por ello, la teoría nos dice que existe una función capaz de modelar qué acción es la mejor en cada estado.

https://cdn-images-1.medium.com/max/1600/1*iVVvxiaGajwZt4wKiIROsA.png

Definición *Q-learning*

Usando esta función Q podríamos encontrar la estrategia óptima para nuestro agente. Los parámetros de la función Q van a ser pares estado-acción. Es común usar una tabla que relacione estados con acciones.

Esta función nos devolverá, para cada par estado-acción, "el valor esperado de la suma de las recompensas futuras". De esta forma podemos ir midiendo en cada estado cuál es la acción que más nos conviene tomar para maximizar la recompensa futura.

Índice

Conceptos importantes

$$Q^{\pi}(s,a) = E[R_t]$$

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\pi^*(s) = arg\max_a Q(s,a)$$

Para encontrar la estrategia óptima, este algoritmo se basa en el proceso de exploración-explotación que vimos en la sesión anterior.

La estrategia irá tomando la acción que maximiza el valor de la recompensa esperada.

Respecto a la recompensa esperada, es común usar un *discount factor* para estimar la importancia que tendrán los valores futuros.

Conceptos importantes

Pero, ¿dónde queda la idea de diseñar como cadena de Markov?

Para ello usaremos la ecuación de Bellman. Como trabajaremos con la recompensa esperada a futuro, necesitamos alguna forma de modelar este comportamiento.

"La ecuación de Bellman proporciona una definición recursiva con la que podremos encontrar la función óptima Q. La fórmula Q(s, a) es igual a la suma de la recompensa inmediata, después de ejecutar la función A en el estado S, y la recompensa futura esperada después de la transición al siguiente estado S"*.

$$Q^{\pi^*}(s,a) = r + \gamma \max_{a'} Q(s',a')$$

Índice

Deep Q-network

Al ver Q-learning, hemos ido aproximando los *q_values* durante las iteraciones de nuestro proceso de aprendizaje. Nuestro agente ha ido aprendiendo una estrategia a partir de la tabla que se va creando.

Esto ha sido posible porque el problema no era demasiado complejo en términos de dimensionalidad (datos y atributos). En la realidad, la complejidad hace que esta forma de aprendizaje sea intratable.

Por ello, en vez de una tabla de *q_values* usaremos una red neuronal como función aproximadora . En vez de aprender los *q_values*, aprenderemos los parámetros de nuestro modelo. Normalmente, y debido a que la mayoría de simulaciones trabajan con la pantalla directamente, el tipo de red neuronal que usaremos serán redes convolucionales.

Deep Q-network

El uso de redes neuronales como función aproximadora conlleva más decisiones por nuestra parte a la hora del desarrollo de nuestras soluciones.

Ya comentamos en sesiones pasadas que a todos los hiperparámetros necesarios de los algoritmos de aprendizaje por refuerzo hay que sumarle los hiperparámetros de los modelos basados en Deep Learning. Esto hace que el proceso de aprendizaje sea empírico, en el que la prueba y error de nuestras hipótesis es fundamental.

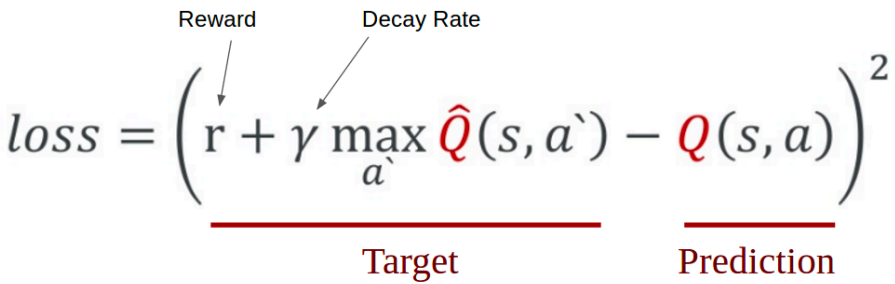
Índice

Proceso de aprendizaje

Como queremos aproximar la función Q con un modelo de Deep Learning, necesitamos una función de coste que mida cómo lo estamos haciendo.

Podemos usar la ecuación de Bellman para, iterativamente, aproximar la función Q usando aprendizaje basado en diferencia temporal.

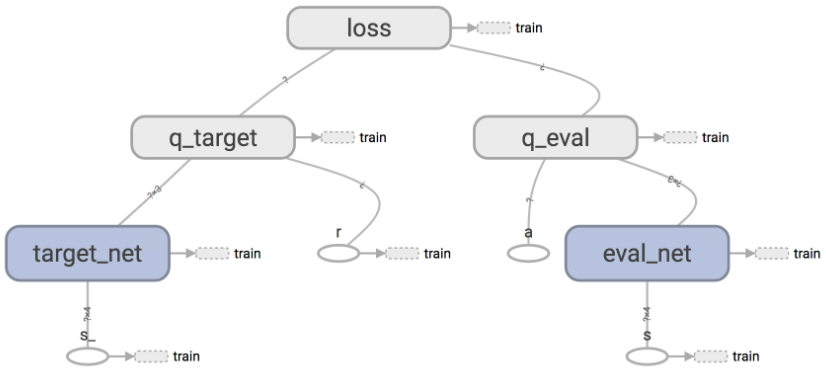
Concretamente, en cada instante de tiempo buscaremos minimizar el error que se produce entre Q(s, a) (término que se predice) y la ecuación de Bellman (término objetivo).

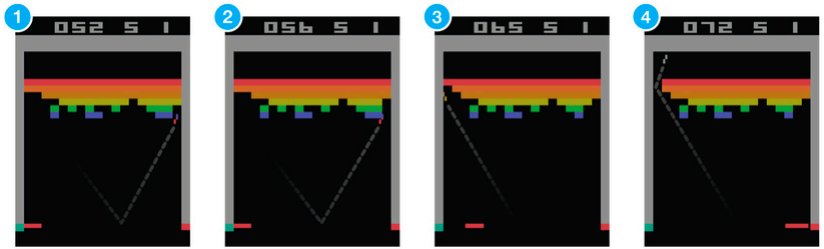


https://keon.io/images/deep-q-learning/deep-q-learning.png

Proceso de aprendizaje

Respecto al proceso de aprendizaje y a las DQN, hay otros dos elementos que son fundamentales para asegurar la convergencia de nuestro modelo. Estos elementos son la *target network* y el uso de una *secuencia de frames*.





http://neuro.cs.ut.ee/wp-content/uploads/2015/12/breakout_deepmind.png

Índice

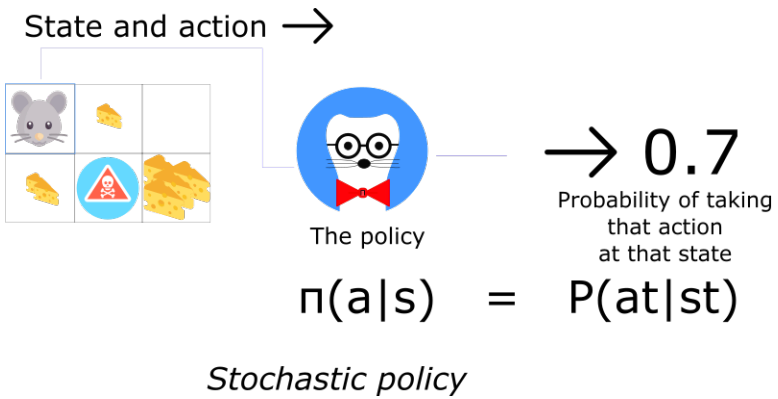
Algoritmo de *DQN*

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Índice

Definición de *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 de *Policy Gradient*

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 ponderaremos 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 muy familiar en el ámbito del 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.

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

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

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.

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 de *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

Algoritmo de *REINFORCE*

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

- Hemos visto dos de los algoritmos fundamentales para nuestros procesos de aprendizaje por refuerzo, *Deep Q-networks* y *Policy Gradients*.

- En *DQN* nuestro objetivo es aproximar los valores de recompensa esperada que relacionan estados con acciones.

- En *Policy Gradient* trabajamos directamente sobre la distribución de probabilidad del espacio de acciones.

- Los procesos de aprendizaje están guiados por la ecuación de Bellman en el caso de *DQN* y una función similar a Cross-entropy en el caso de *Policy Gradients*.

Enlaces de interés

- Deep Q-networks

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html
https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0

- Policy Gradients

http://karpathy.github.io/2016/05/31/rl/
https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Gracias

viu Universidad Internacional de Valencia