Una Aplicación Exitosa de Deep Reinforcement Learning

Conceptos relevantes

- Proceso de Decisión de Markov (MDP):

$$\langle S, A, T(s,a), R(s,a) \rangle$$

- Función de valor:

$$V^{\pi}(s_t) = r_{t+1} + \gamma\, r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots = \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}$$

- Función Q:

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(T(s,a), a')$$

- Método iterativo ("model-free" y "online") para aprender Q:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Usando este algoritmo, lo que se aprende de un estado no se usa de ninguna manera en estados "similares".

Por ejemplo en Pacman, si usáramos (entre otras cosas) la cantidad de pastillas para definir un estado, las dos imágenes de abajo serían consideradas estados totalmente distintos.

El algoritmo de Q-learning teóricamente resuelve el problema; pero, en la práctica y con problemas medianos, requiere mucha exploración y tiempo de entrenamiento.

Una alternativa es aproximar Q(s, a) con una función.

$$Q(s, a) = F(s, a)$$

Por ejemplo, una combinación lineal de features predefinidos (muy parecido a regresión lineal).

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a) w_i$$

Ahora el problema se resume a aprender los w_i que mejor estiman Q. En el caso de un aproximador lineal esto se puede estimar de acuerdo a las siguientes expresiones:

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$
$$difference = (r + \gamma \max_{a'} Q(s', a')) - Q(s,a)$$

En el caso de Pacman, usando este algoritmo el sistema aprende bastante bien y rápido. Veamos el ejemplo tomando como features:

- La distancia a la pastilla más cercana.
- Si al moverse en una dirección hay un fantasma a un paso de distancia.
- Si, de no haber peligro de fantasma a un paso de distancia, Pacman come una pastilla al moverse en una dirección.

Problemas de aproximar Q de este esquema:

- Depende en gran medida de features construidos a mano (por qué Pacman no come a los fantasmas cuando puede). No parece que escale muy bien tener que hacer esto para todas las distintas tareas.

- Un aproximador lineal quizás no sea lo más potente (pero hace al algoritmo estable). Aproximadores no lineales en la práctica andan muy mal (ya vamos a ver por qué).

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Objetivo:
Proponer un algoritmo capaz de aprender múltiples tareas (49 juegos de Atari) con una misma configuración, creando de manera automática los features relevantes.

¿Cómo la hacen?
El artículo ataca los dos problemas mencionados aproximando Q con una red neuronal profunda y pudiéndola entrenar de manera exitosa.

Fuente: http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html

Ideas principales:

Combinar approximate q learning con redes neuronales convolucionales profundas.

Función a optimizar:

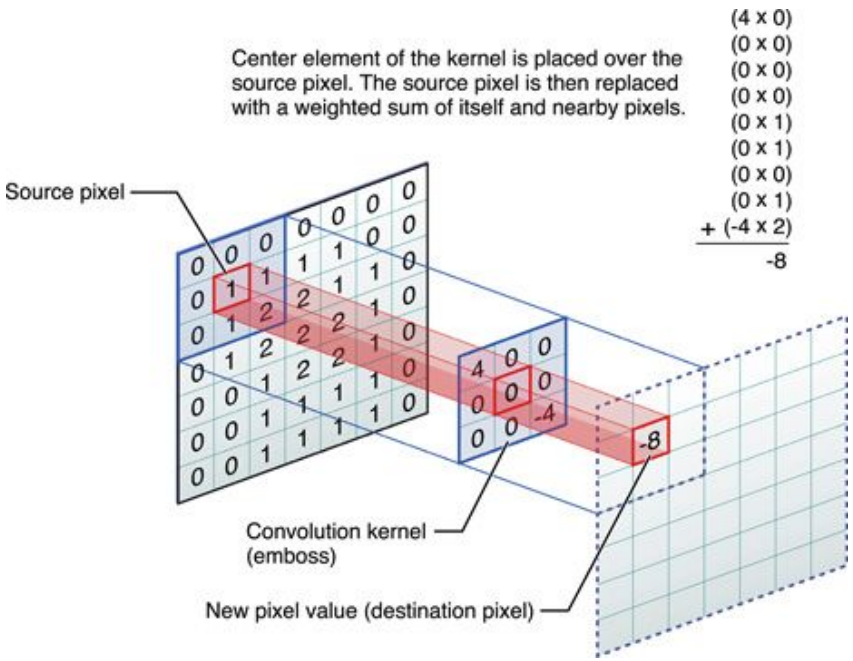
$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\, a_t = a,\, \pi\right]$$

Ahora Q se parametriza con una función que tiene la estructura de red convolucional profunda y se busca minimizar la siguiente expresión de costo:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i) \right)^2 \right]$$

El "chiste" para que el algoritmo converja es qué es D y qué es $Q(s',a';\theta_i^-)$

¿Qué hace una convolución con kernels en imágenes?



Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Fuentes: https://goo.gl/Xz4LJn , https://goo.gl/UafNf6

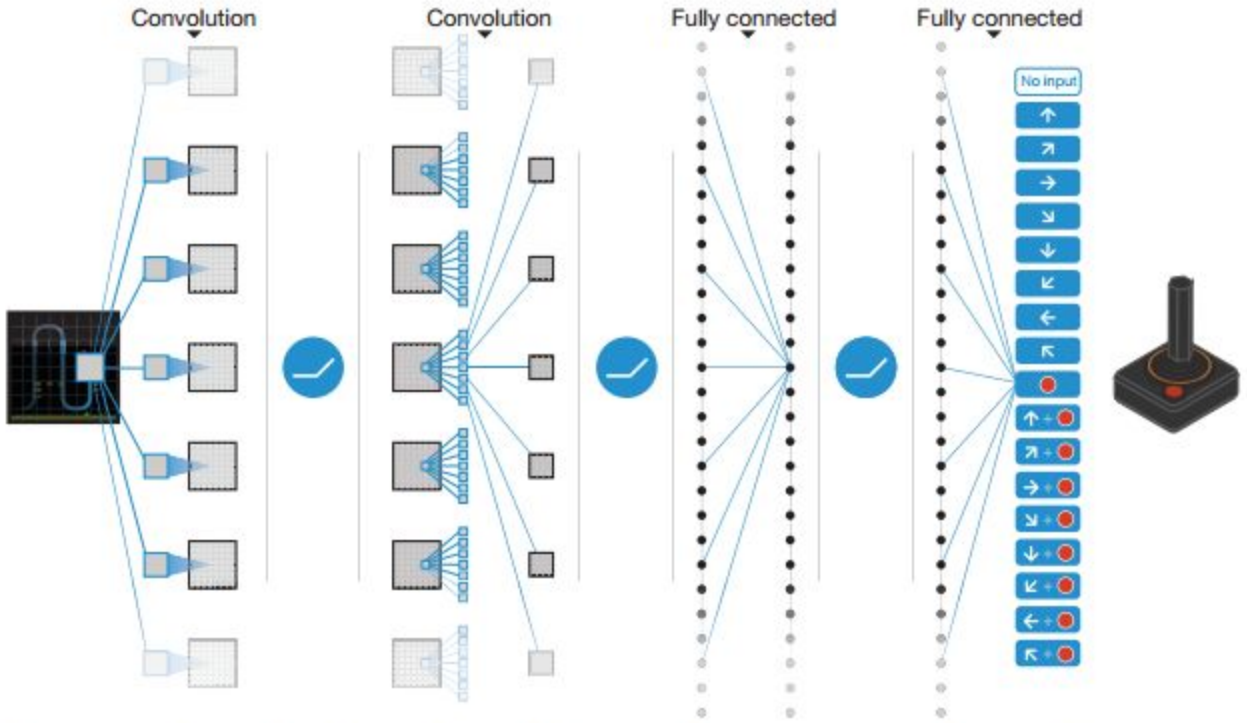


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

Noten que una misma red calcula el Q de un estado para todas las acciones posibles! (es computacionalmente más barato plantearlo así).

Sobre esta red se puede usar backpropagation y obtener la derivada del costo respecto a cada uno de los parámetros (que en el caso de redes convolucionales son "pocos").

Algoritmos de aprendizaje por refuerzos suelen ser inestables o incluso diverger cuando se usan aproximadores no lineales. Los autores mencionan los siguientes motivos:

- Existen correlaciones en la serie de estados observados (los estados están lejos de ser i.i.d).

- Pequeños cambios en los parámetros pueden cambiar dramáticamente Q, esto puede cambiar las acciones óptima rápidamente y en consecuencia sesgar las acciones y la distribución de los datos.

- Actualizaciones de los pesos que aumenten el valor de $Q(s_t,at)$ generalmente aumentan los valores de $Q(s_{t+1},a)$ para todas las acciones (lo que uso como target a predecir es muy inestable).

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

Truco #1

Experience Replay (supuestamente motivado por el comportamiento biológico de los animales):

Se guarda la experiencia del agente en el momento t, $e_t = (s_t, a, r_t, s_{t+1})$ en un dataset $D = \{e_1, e_2, \dots, e_t\}$ que puede contener datos de muchos episodios.

Se actualizan los pesos de la red usando muestras (mini-batchs) de este dataset y no de lo que efectivamente hizo el agente.

Esto implica aprender off-policy (los parámetros actuales son distintos de aquellos utilizados para generar la muestra).

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

Truco #2

Mejorar la estabilidad cada *C* actualizaciones de los pesos se clona la red actual y se usa esa red clonada (que no se actualiza en cada iteración) para crear los targets a optimizar.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

(Mini truco adicional para que generalice bien entre juegos con un mismo set de hiperparámetros: se pone como límite del error y los rewards -1 y 1).

Algoritmo

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Resultados

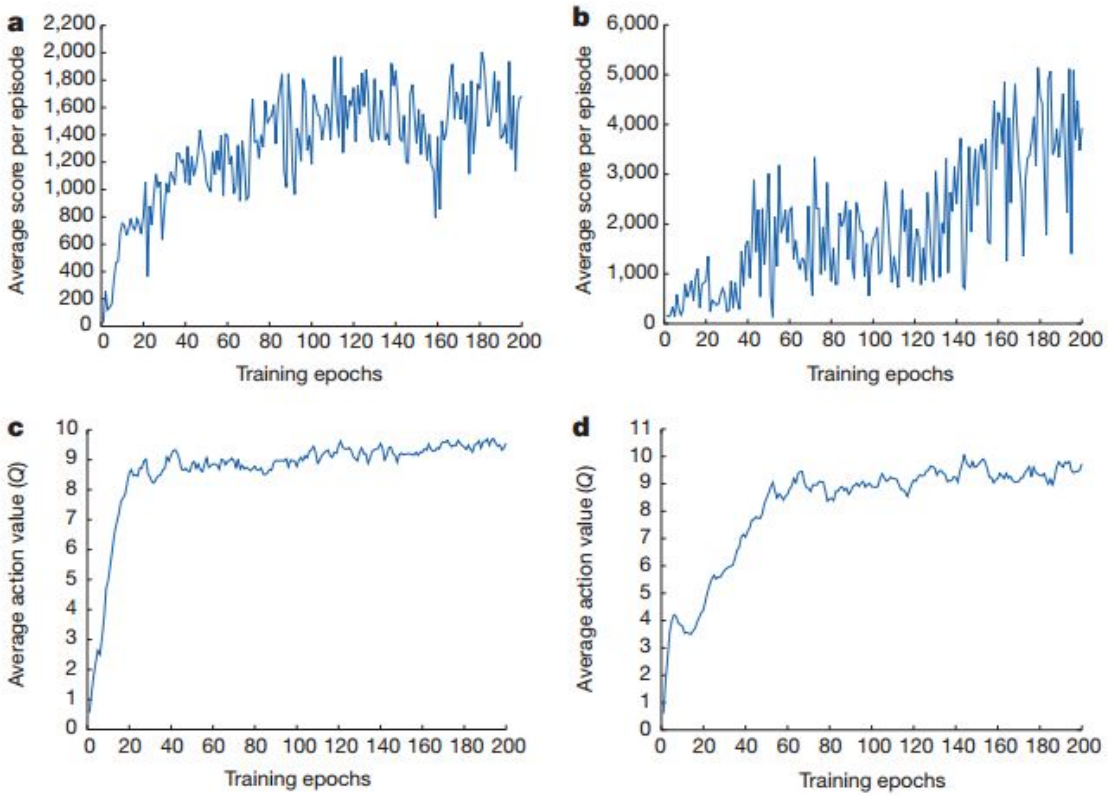
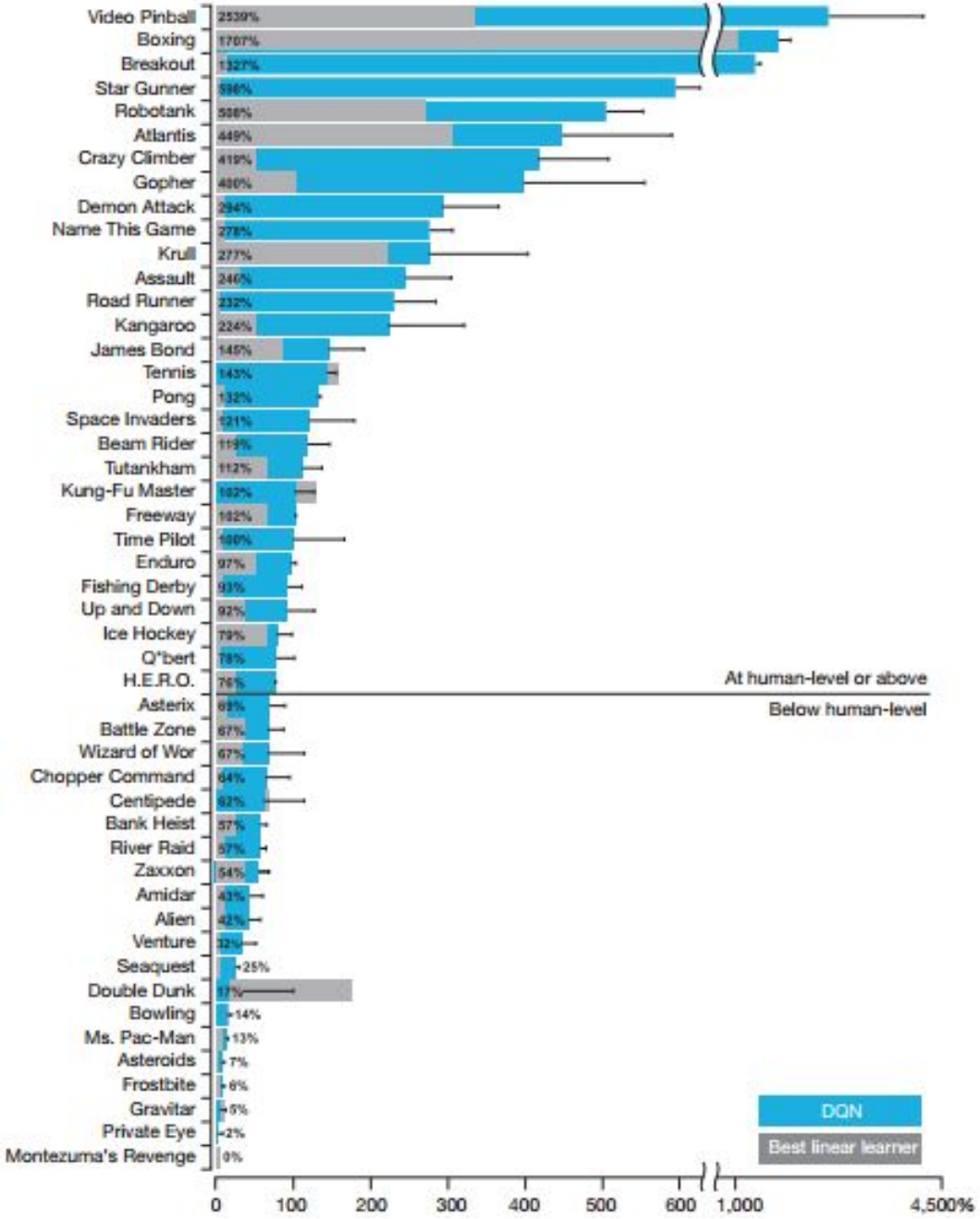


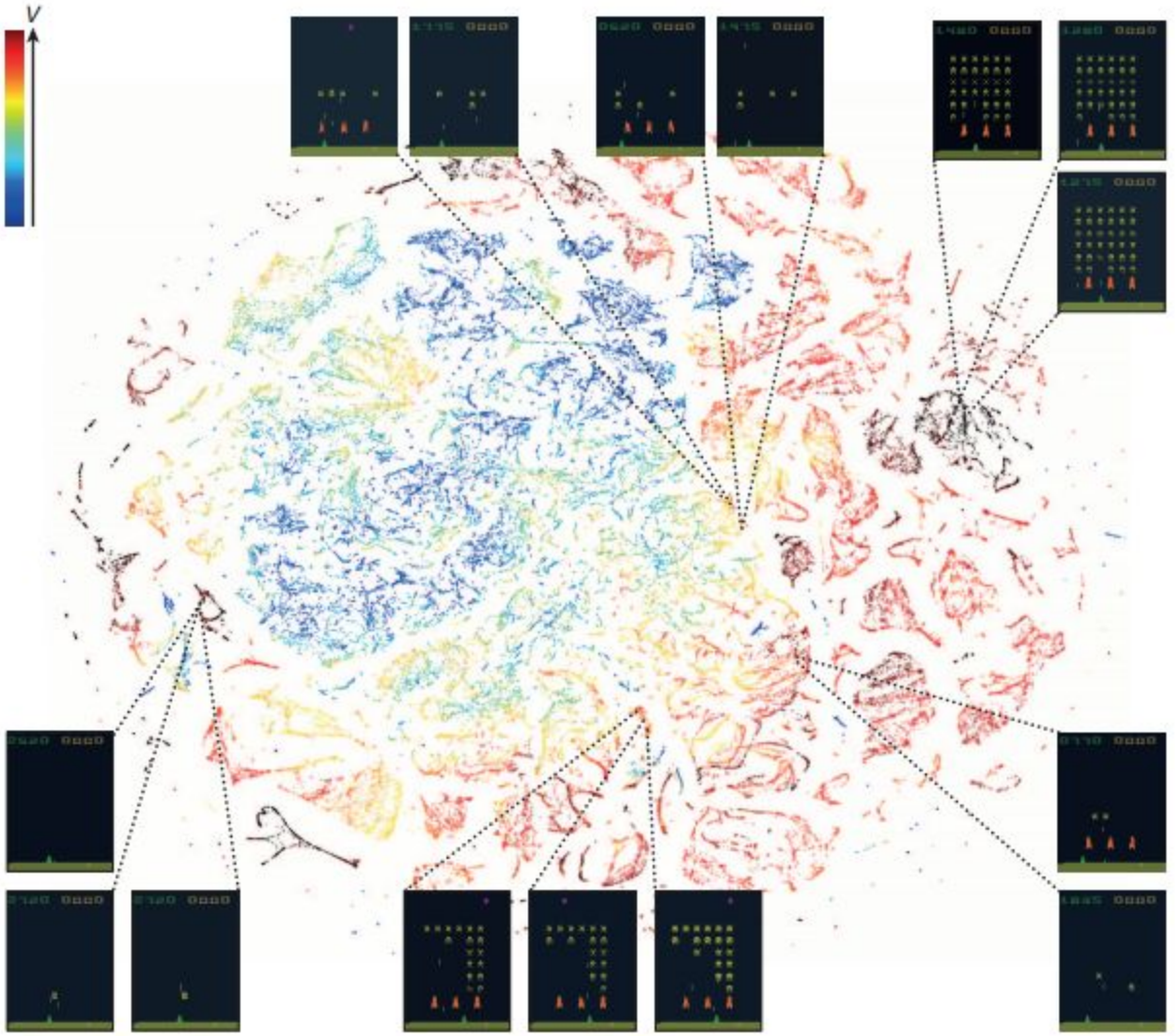
Figure 2 | Training curves tracking the agent's average score and average predicted action-value. **a**, Each point is the average score achieved per episode after the agent is run with ε-greedy policy ($\varepsilon = 0.05$) for 520 k frames on Space Invaders. **b**, Average score achieved per episode for Seaquest. **c**, Average predicted action-value on a held-out set of states on Space Invaders. Each point on the curve is the average of the action-value Q computed over the held-out set of states. Note that Q-values are scaled due to clipping of rewards (see Methods). **d**, Average predicted action-value on Seaquest. See Supplementary Discussion for details.

Video Breakout Video Space Invaders

Resultados


Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
596%
506%
449%
419%
400%
294%
278%
277%
246%
232%
224%
145%
143%
132%
121%
119%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
93%
67%
67%
64%
62%
57%
57%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Resultados



Conclusiones:

Usando aprendizaje automático hoy se puede aprender de manera automática una gran gama de juegos simples de Atari utilizando como input sólo imágenes (lo que es increíble).

¿Qué pasa con Montezuma's Revenge? (video)

(¡Por el momento estamos a salvo!¿?)