

Deep Reinforcement Learning

Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa CS 231n, Stanford

<http://cs231n.stanford.edu/>

Nadgledano Učenje

Podaci: (x, y)

x podatak, y klasa

Cilj: Naučiti funkciju koja mapira $x \rightarrow y$

Primeri: Klasifikacija, detekcija objekata, semantička segmentacija slike, dodeljivanje opisa slikama itd.



→ Cat

Klasifikacija

[This image is CC0 public domain](#)

Nenadgledano Učenje

Podaci: x

Samo podaci, nema klasa!

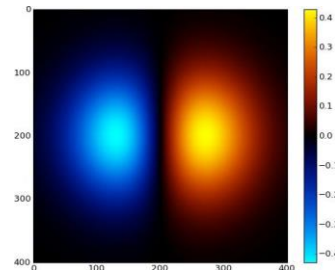
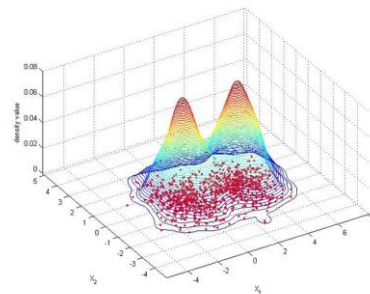
Cilj: Naučiti neku „skrivenu“
strukturu u podacima

Primeri: Klasterovanje,
redukcija dimenzionalnosti,
procena distribucije (gustine) itd.



Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d procena gustine



2-d procena gustine

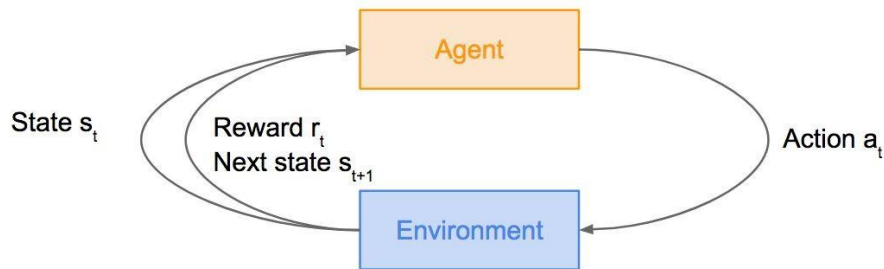
2-d density images [left](#) and [right](#)
are [CC0 public domain](#)

Danas: Učenje Uslovljavanjem

Reinforcement Learning

Problemi koji uključuju **agenta** koji je u interakciji sa **okruženjem**, koje pruža povratnu informaciju u vidu **nagrada** ili **kazni**.

Cilj: Naučiti koje akcije treba raditi da bi se dobila maksimalna moguća nagrada.



Atari games figure copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Sadržaj

- Šta je Učenje Uslovljavanjem?
- Markovljev Proces Odlučivanje
- Q-Učenje (*Q-Learning*)
- Gradijenti Politike (*Policy Gradients*)

Reinforcement Learning

Agent

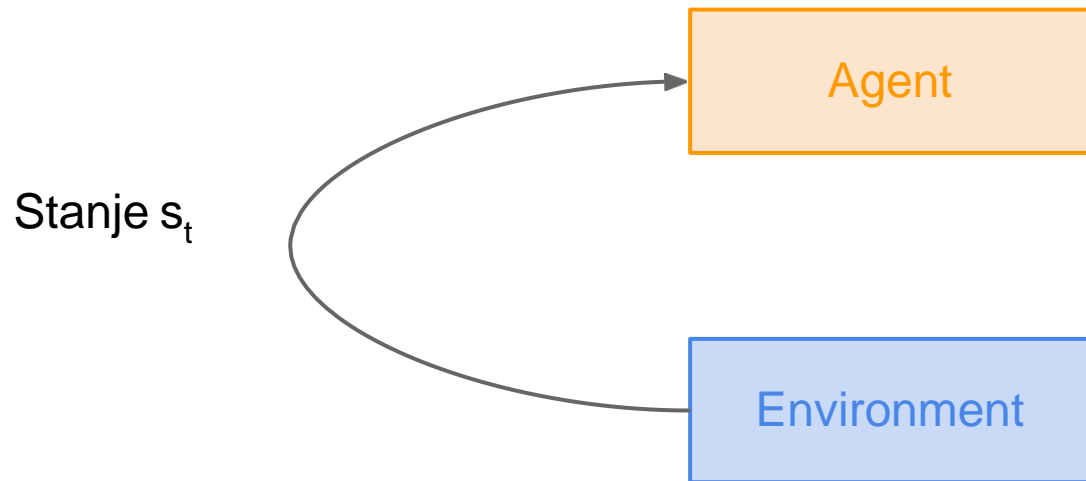


```
graph TD; Agent[Agent] --> Environment[Environment]; Environment --> Agent;
```

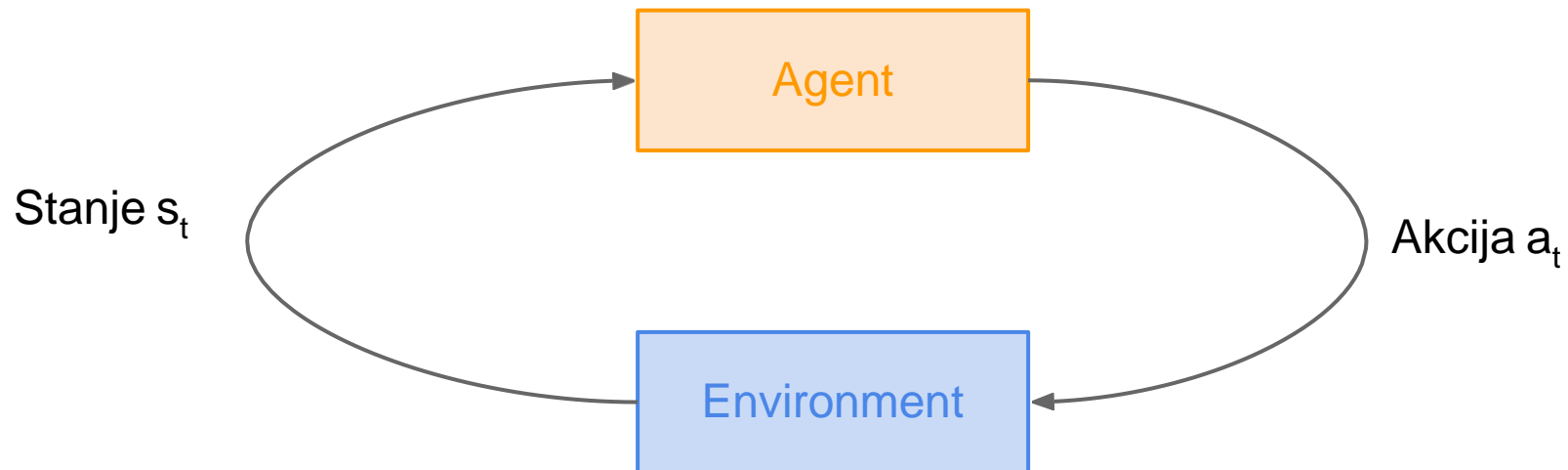
The diagram illustrates the Reinforcement Learning loop. It consists of two main components: the Agent and the Environment. The Agent is represented by an orange box at the top, and the Environment is represented by a blue box at the bottom. Arrows indicate a bidirectional flow of information between them: the Agent sends actions to the Environment, and the Environment sends observations and rewards back to the Agent.

Environment

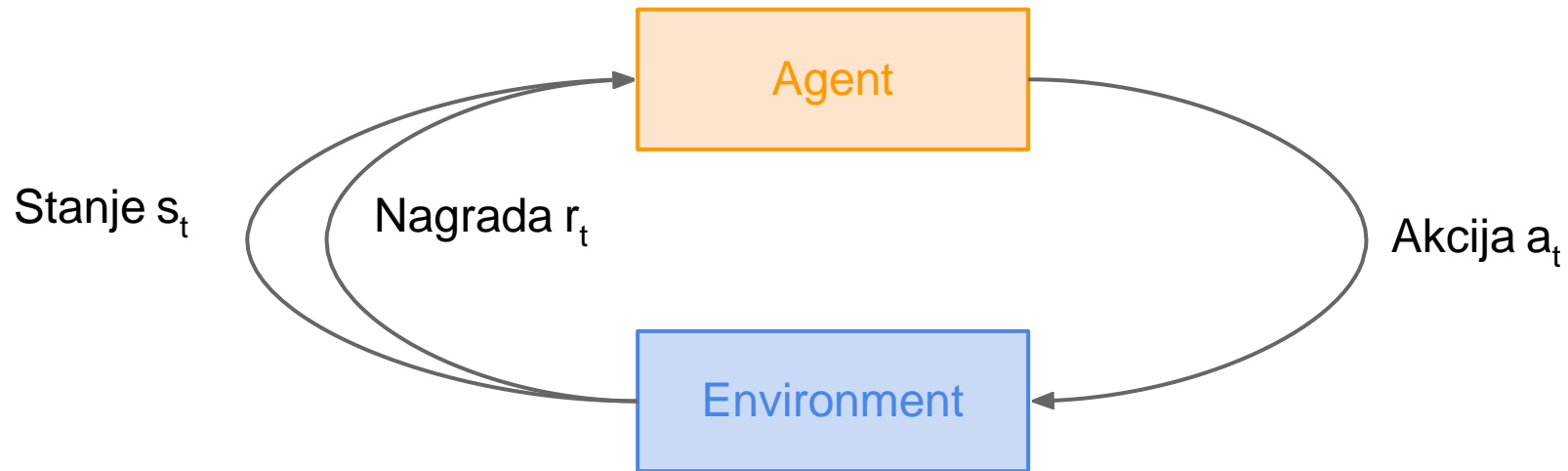
Reinforcement Learning



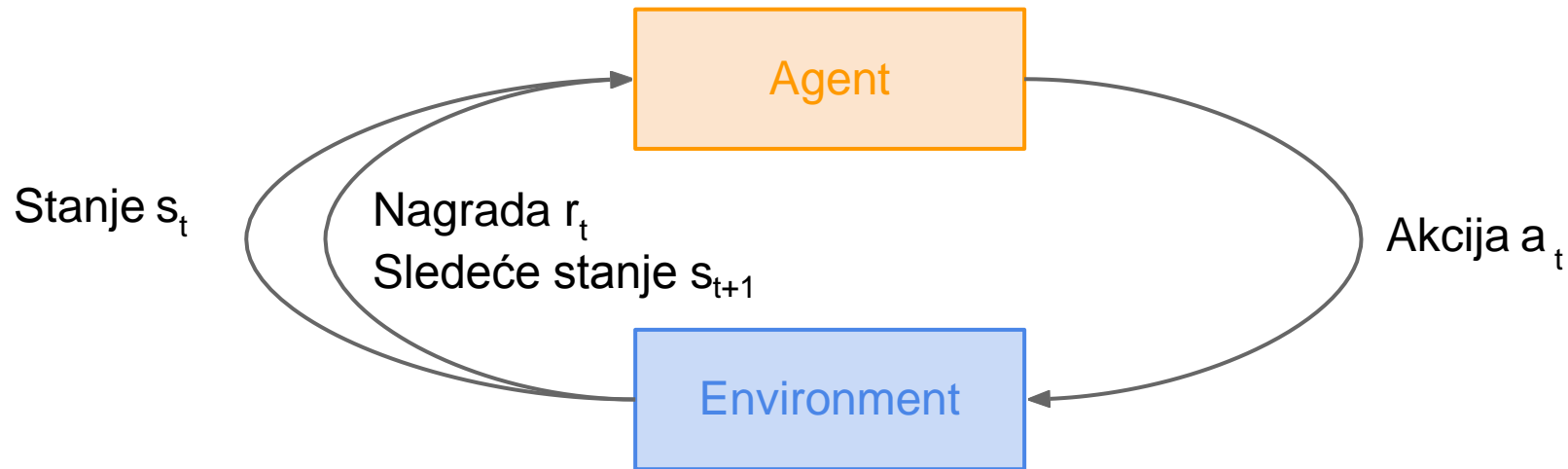
Reinforcement Learning



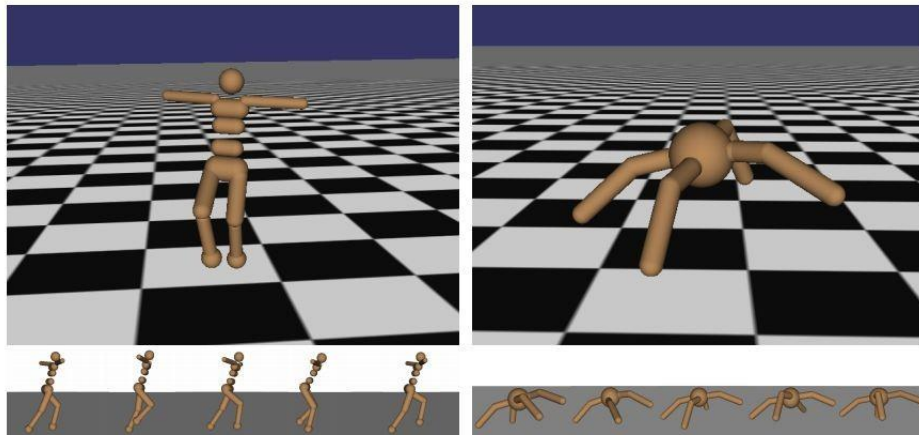
Reinforcement Learning



Reinforcement Learning



Kretanje Robota



Cilj: Naučiti robota da se kreće unapred

Stanje: Ugao i pozicija zglobova

Akcija: obrtni moment koji se primenjuje na zglobove

Nagrada: 1 za svaku jedinicu vremena kada je robot uspravno i hoda unapred

Figures copyright John Schulman et al., 2016. Reproduced with permission.

Atari Igre



Cilja: Završiti igru sa što više moguće bodova

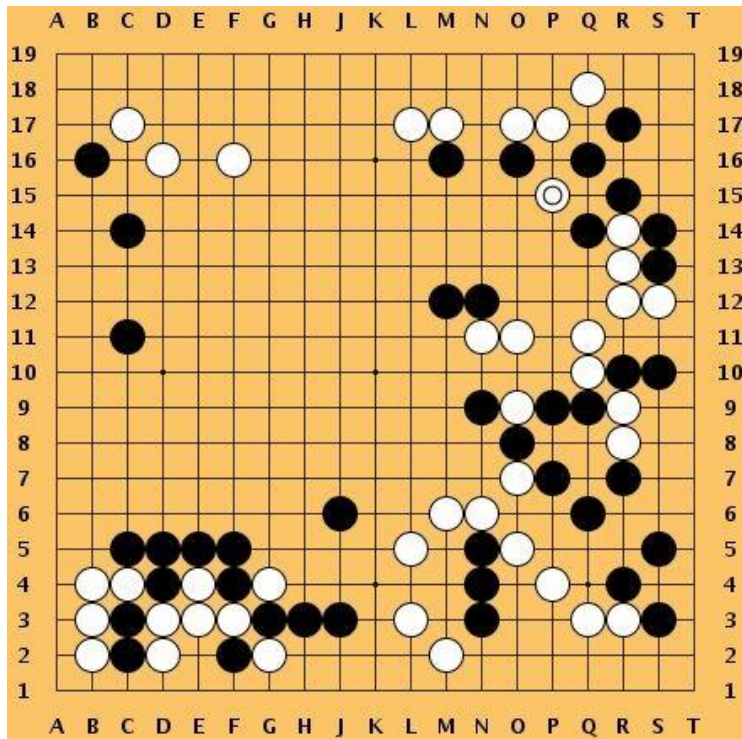
Stanje: Pikseli (frejmovi) iz igre

Akcije: Kontrole Atari kontrolera

Nagrada: Povećanje/Smanjenje bodova u igri

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Go



Cilj: Pobediti!

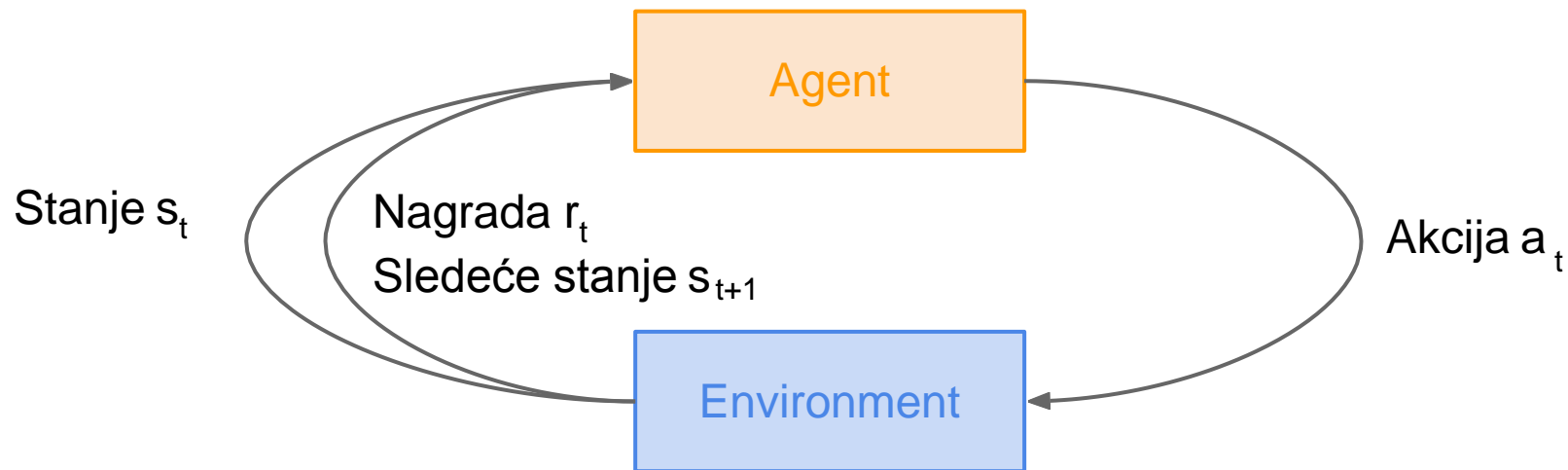
Stanje: Pozicije figura na tabli

Akcije: Premetšanje figura

Nagrada: 1 pobjeda na kraju igre, 0 inače

[This image is CC0 public domain](#)

Kako da matematički formalizujemo RL problem?



Markovljev Proces Odlučivanja

- Matematička formalizacija RL problema
- **Markovljevo svojstvo**: Ishod akcije u trenutnom stanju zavisi samo od trenutnog stanja i akcije, a ne prethodnih stanja i akcija

Definisan sa: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

\mathcal{S} : skup stanja

\mathcal{A} : skup akcija

\mathcal{R} : funkcija nagrade

\mathbb{P} : funkcija prelaza

γ : faktor zanemarivanja

Markovljev Proces Odlučivanja

- Počinjemo u vremenskom trenutku $t=0$,
- Krećemo od nekog datog ili slučajno odabranog početnog stanja s_0
- Ponavljamo od $t=0$ do završetka (definišemo ga na razne načine):
 - Agent bira na neki način i radi akciju a_t
 - Okruženje vraća nagradu $r_t \sim R(\cdot | s_t, a_t)$
 - Okruženje vraća sledeće stanje u koje agent prelazi $s_{t+1} \sim P(\cdot | s_t, a_t)$
- Politika je funkcija koja preslikava S na A i specificira koju akciju treba uraditi u kom stanju
- **Cilj:** pronaći optimalnu politiku u^* koja maksimizuje kumulativnu nagradu uz faktor zanemarivanja: $\sum_{t \geq 0} \gamma^t r_t$

Primer MDP: Grid World

akcija = {

1. desno →

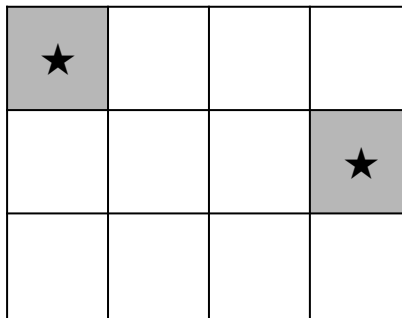
2. levo ←

3. gore ↑

4. dole ↓

}

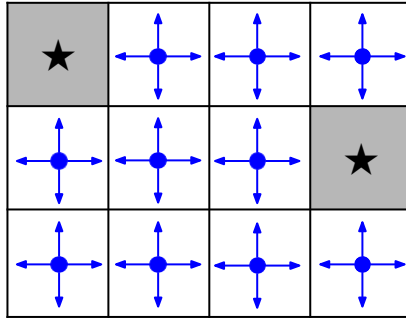
stanja



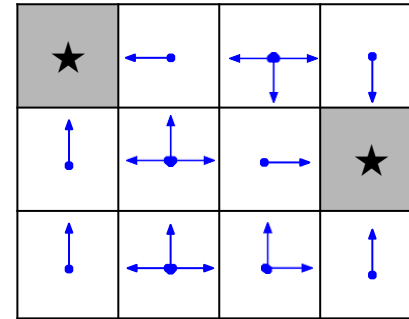
Možemo imati negativnu nagradu postojanja tj. svaka akcija izaziva npr. $r = -1$

Cilj: stići do jednog od dva ciljna stanja (označenja zvezdicom) uz što manje akcija

Primer MDP: Grid World



Politika
dobijena na
slučajan način



Optimalna
politika

Optimalna politika u^*

Hoćemo da pronađemo optimalnu politku u^* koja maksimizuje sumu nagrada.

Na koji način da rešimo stohastničnost okruženja?

Optimalna politika u^*

Hoćemo da pronađemo optimalnu politku u^* koja maksimizuje sumu nagrada.

Na koji način da rešimo stohastničnost okruženja?

Koristimo matematičko očekivanje

Maksimizujemo očekivanu sumu nagrada!

Formalnije: $\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right]$ uz $s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$

verovatnoća
da ćemo početi
iz stanja s_0

verovatnoća
da ćemo ako smo u
stanju s_t uradili akciju a_t
preći u stanje s_{t+1}

verovatnoća
da ćemo u stanju s_t
uraditi akciju a_t

Optimalna politika u^*

Hoćemo da pronađemo optimalnu politku u^* koja maksimizuje sumu nagrada.

Na koji način da rešimo stohastičnost okruženja?

Koristimo matematičko očekivanje

Maksimizujemo očekivanu sumu nagrada!

Formalnije: $\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right]$ uz $s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$

Napomena: Politika je u ovom slučaju stohastička tj. ona je distribucija verovatnoća akcija nad stanjima.

verovatnoća
da ćemo početi
iz stanja s_0

verovatnoća
da ćemo ako smo u
stanju s_t uradili akciju a_t
preći u stanje s_{t+1}

verovatnoća
da ćemo u stanju s_t
uraditi akciju a_t

Optimalna politika u^*

Hoćemo da pronađemo optimalnu politku u^* koja maksimizuje sumu nagrada.

Na koji način da rešimo stohastičnost okruženja?

Koristimo matematičko očekivanje

Maksimizujemo očekivanu sumu nagrada!

Formalnije: $\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right]$ uz $s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$

Napomena 2: Stohastička politika olakšava rešavanje različitih vrsta optimizacionih problema vezanih za RL.

verovatnoća
da ćemo početi
iz stanja s_0

verovatnoća
da ćemo ako smo u
stanju s_t uradili akciju a_t
preći u stanje s_{t+1}

verovatnoća
da ćemo u stanju s_t
uraditi akciju a_t

Optimalna politika u^*

Hoćemo da pronađemo optimalnu politku u^* koja maksimizuje sumu nagrada.

Na koji način da rešimo stohastičnost okruženja?

Koristimo matematičko očekivanje

Maksimizujemo očekivanu sumu nagrada!

Formalnije: $\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right]$ uz $s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$

Napomena 3: Stohastička politika omogućava da nekad radimo akcije koje imaju malu verovatnoću tj. nisu optimalne, ali nam omogućavaju istraživanje našeg okruženja.

verovatnoća
da ćemo početi
iz stanja s_0

verovatnoća
da ćemo ako smo u
stanju s_t uradili akciju a_t
preći u stanje s_{t+1}

verovatnoća
da ćemo u stanju s_t
uraditi akciju a_t

Definicije: Funkcija Vrednosti i Funkcija Q-vrednosti

Ako agent prati neku politiku on kao rezultat proizvodi neki niz akcija i stanja (koji ćemo nazvati putanjom) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

Definicije: Funkcija Vrednosti i Funkcija Q-vrednosti

Ako agent prati neku politiku on kao rezultat proizvodi neki niz akcija i stanja (koji ćemo nazvati putanjom) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

Koliko je dobro neko stanje?

Vrednost stanja s za politiku π , je očekivana suma nagrada uz zanemarivanje ako od stanja s pratimo politku π :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

Definicije: Funkcija Vrednosti i Funkcija Q-vrednosti

Ako agent prati neku politiku on kao rezultat proizvodi neki niz akcija i stanja (koji ćemo nazvati putanjom) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

Koliko je dobro neko stanje?

Vrednost stanja s za politiku π , je očekivana suma nagrada uz zanemarivanje ako od stanja s pratimo politiku π :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

Koliko je dobro raditi neku akciju u nekom stanju?

Q-vrednost akcije a u stanju s i a za politiku π , je očekivana suma nagrada ako u s uradimo a i nakon toga pratimo politiku π :

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Belmanova jednačina

Optimalna funkcija Q-vrednosti Q^* je ona koja za svaki par stanje, akcija vraća najveću moguću očekivanu nagradu:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Belmanova jednačina

Optimalna funkcija Q-vrednosti Q^* je ona koja za svaki par stanje, akcija vraća najveću moguću očekivanu nagradu:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Q^* zadovoljava **Belmanovu jednačinu**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Intuicija: ako je optimalna vrednost za sledeći korak tj. $Q^*(s', a')$ poznata, onda je optimalna akcija a u s takva da daje najveću vrednost za:

$$r + \gamma Q^*(s', a')$$

Belmanova jednačina

Optimalna funkcija Q-vrednosti Q^* je ona koja za svaki par stanje, akcija vraća najveću moguću očekivanu nagradu:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Q^* zadovoljava **Belmanovu jednačinu**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Intuicija: ako je optimalna vrednost za sledeći korak tj. $Q^*(s', a')$ poznata, onda je optimalna akcija a u s takva da daje najveću vrednost za:

$$r + \gamma Q^*(s', a')$$

Optimalna politika u^* je ona koja kaže agentu da svaki put radi optimalnu akciju tj. onu koja ima maksimalnu Q^*

Kako da dobijemo optimalnu politiku?

Iteracija vrednosti: Koristimo Belmanovu jednačinu iterativno

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i konvergira ka Q^* kada $i \rightarrow \infty$

Kako da dobijemo optimalnu politiku?

Iteracija vrednosti: Koristimo Belmanovu jednačinu iterativno

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i konvergira ka Q^* kada $i \rightarrow \infty$

U čemu je problem sa ovim postupkom?

Kako da dobijemo optimalnu politiku?

Iteracija vrednosti: Koristimo Belmanovu jednačinu iterativno

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i konvergira ka Q^* kada $i \rightarrow \infty$

U čemu je problem sa ovim postupkom?

Nije skalabilan. Moramo da izračunamo $Q(s, a)$ za svaki mogući par stanje, akcija. Ako su nam npr. stanja frejmovi (pikseli) iz neke igre nikad nećemo moći da izračunamo sve moguće Q -vrednosti.

Kako da dobijemo optimalnu politiku?

Iteracija vrednosti: Koristimo Belmanovu jednačinu iterativno

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i konvergira ka Q^* kada $i \rightarrow \infty$

U čemu je problem sa ovim postupkom?

Nije skalabilan. Moramo da izračunamo $Q(s, a)$ za svaki mogući par stanje, akcija. Ako su nam npr. stanja frejmovi (pikseli) iz neke igre nikad nećemo moći da izračunamo sve moguće Q -vrednosti.

Rešenje: koristimo neku funkciju (neki aproksimator) da procenimo $Q(s, a)$.
Na primer Neuronsku Mrežu!

Q-učenje

Q-učenje: koristimo neku funkciju (neki aproksimator) da procenimo $Q(s,a)$. Funkcija ima neke parametre θ . Kod NN to su težine, kod lin. regresije to su koeficijenti.

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Q-učenje

Q-učenje: koristimo neku funkciju (neki aproksimator) da procenimo $Q(s,a)$. Funkcija ima neke parametre θ . Kod NN to su težine, kod lin. regresije to su koeficijenti.


$$Q(s, a; \theta) \approx Q^*(s, a)$$

Ako je aproksimator duboka neuronska mreža onda imamo **deep q-learning**!

Q-učenje

Q-učenje: koristimo neku funkciju (neki aproksimator) da procenimo $Q(s,a)$. Funkcija ima neke parametre θ . Kod NN to su težine, kod lin. regresije to su koeficijenti.

$$Q(s, a; \theta) \approx Q^*(s, a)$$

 težine neuronske mreže

Ako je aproksimator duboka neuronska mreža onda imamo **deep q-learning**!

Q-učenje

Cilj nam je da pronađemo Q-funkciju koja zadovoljava Belmanovu jednačinu:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Q-učenje

Cilj nam je da pronađemo Q-funkciju koja zadovoljava Belmanovu jednačinu:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Izračunavanje Unapred Funkcija greške:

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

gde je:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Napomena: Q-vrednost u formuli za y_i izračunavamo pomoću modela iz prethodne iteracije obučavanja ($i-1$).

Q-učenje

Cilj nam je da pronađemo Q-funkciju koja zadovoljava Belmanovu jednačinu:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Izračunavanje Unapred Funkcija greške:

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

gde je:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Na primer, trenuta iteracija mreže za stanje 5 i akciju 2 daje $Q(5,2)=12$. Agent je upravo uradio akciju 2 u stanju 5 i dobio neku nagradu r . Greška y_i meri koliko se procena $Q(5,2)=12$ razlikuje od nagrade koju smo upravo dobili + maksimalne očekivane nagrade (uz zanemarivanje) od stanja u koje smo prešli. Greška meri koliko moramo da korigujemo procenu od 12 da bi ona bila u skladu sa našim iskustvom iz okruženja + onoga što očekujemo od stanja u koje smo došli (to dobijamo iz prethodne iteracije mreže).

Q-učenje

Cilj nam je da pronađemo Q-funkciju koja zadovoljava Belmanovu jednačinu:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Izračunavanje Unapred Funkcija greške:

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

gde je:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Izračunavanje unazad (backpropagation):

Promena gradijenta (u odnosu na Q-funkciju i parametre θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i)$$

Q-učenje

Cilj nam je da pronađemo Q-funkciju koja zadovoljava Belmanovu jednačinu:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Izračunavanje Unapred Funkcija greške:

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

gde je:

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Izračunavanje unazad (backpropagation):

Promena gradijenta (u odnosu na Q-fukciju i parametre θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Iterativno pomeramo Q vrednost ka ciljnoj vrednosti y_i koju bi trebalo da ima da je politika optimalna

Primer: Igranje Atari Igara



Cilja: Završiti igru sa što više moguće bodova

Stanje: Pikseli (frejmovi) iz igre

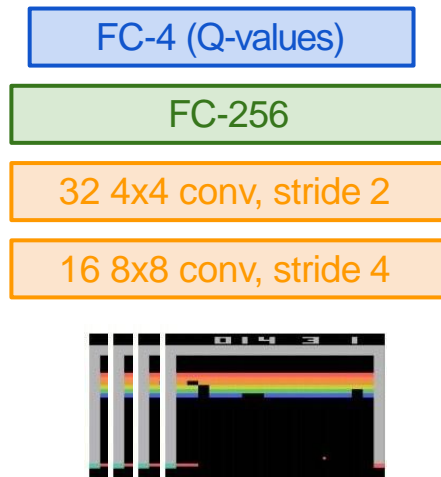
Akcije: Kontrole na Atari kontrolera

Nagrada: Povećanje/Smanjenje bodova u igri

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Arhitektura Q-mreže (Q-network)

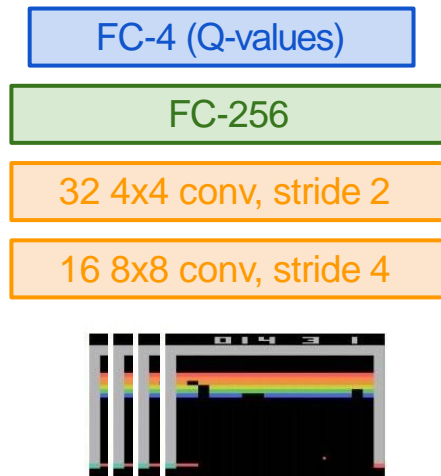
$Q(s, a; \theta)$:
neuronska mreža
sa težinama θ



Trenutno stanje s_t : 84x84x4 poslednja 4 frejma
(malo predprocesiranja: RGB->grayscale, *downsampling*, i *cropping*)

Arhitektura Q-mreže (Q-network)

$Q(s, a; \theta)$:
neuronska mreža
sa težinama θ

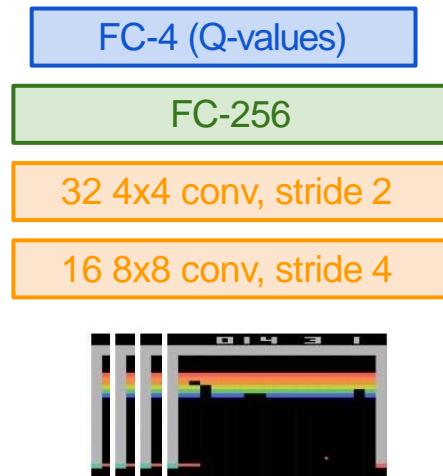


← Ulaz: stanje s_t

Trenutno stanje s_t : 84x84x4 poslednja 4 frejma
(malo predprocesiranja: RGB->grayscale, *downsampling*, i *cropping*)

Arhitektura Q-mreže (Q-network)

$Q(s, a; \theta)$:
neuronska mreža
sa težinama θ

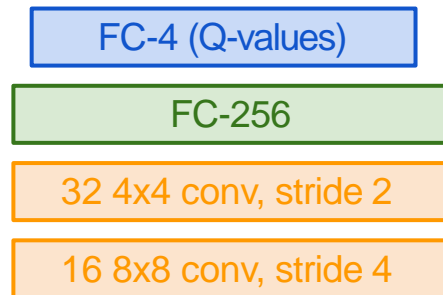


← Kovolutivni slojevi i
Potpuno Povezani
(Fully Connected, FC)
slojevi

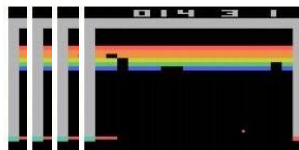
Trenutno stanje s_t : 84x84x4 poslednja 4 frejma
(malo predprocesiranja: RGB->grayscale, *downsampling*, i *cropping*)

Arhitektura Q-mreže (Q-network)

$Q(s, a; \theta)$:
neuronska mreža
sa težinama θ



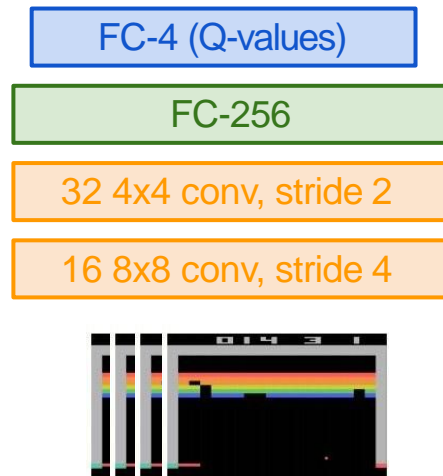
← Poslednji FC sloj ima 4 izlazna neurona (za 4 akcije), to su ustvari vrednosti $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$



Trenutno stanje s_t : 84x84x4 poslednja 4 frejma
(malo predprocesiranja: RGB->grayscale, *downsampling*, i *cropping*)

Arhitektura Q-mreže (Q-network)

$Q(s, a; \theta)$:
neuronska mreža
sa težinama θ



← Poslednji FC sloj ima 4 izlazna neurona (za 4 akcije), to su ustvari vrednosti $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

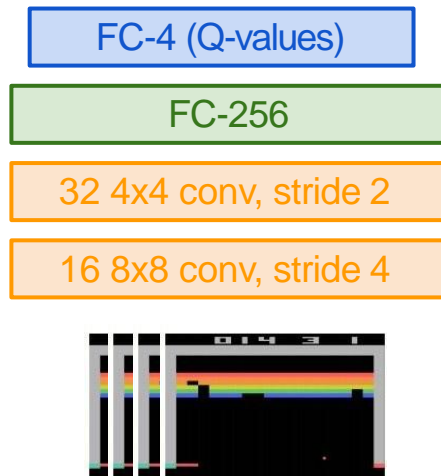
Broj akcija može da varira između 4 i 18 u zavisnosti od konkretne Atari igre

Trenutno stanje s_t : 84x84x4 poslednja 4 frejma
(malo predprocesiranja: RGB->grayscale, *downsampling*, i *cropping*)

Arhitektura Q-mreže (Q-network)

$Q(s, a; \theta)$:
neuronska mreža
sa težinama θ

Samo jedan prolaz unapred
(*feedforward pass*) da bi se
izračunale Q-vrednosti za sve
akcije u trenutnom stanju => ovo
je vrlo efikasno, ne radimo
prolaz za svako Q posebno!



← Poslednji FC sloj ima 4
izlazna neurona (za 4
akcije), to su ustvari
vrednosti $Q(s_t, a_1)$, $Q(s_t, a_2)$,
 $Q(s_t, a_3)$, $Q(s_t, a_4)$

Broj akcije može da varira
između 4 i 18 u zavisnosti od
konkretne Atari igre

Trenutno stanje s_t : 84x84x4 poslednja 4 frejma
(malo predprocesiranja: RGB->grayscale, *downsampling*, i *cropping*)

Obučavanje Q-mreže: *Experience Replay*

Učenje na osnovu niza uzastopnih semplova se empirijski pokazalo kao problematično:

- Semplovi su u korelaciji*=> neefikasno učenje
- Trenutni parametri Q-mreže faktički diktiraju akcije tj. koji ćemo sledeći sempl dobiti za učenje (npr. ako trenutno očekivanu nagradu maksimizuje akcija „levo“, semplovi na osnovu kojih učimo biće puni samo stanja iz levog dela porostora problema) => ovo nas može dovesti do petlji**

*Formalno mi koristimo MDP i ishod akcije koju radimo u trenutnom stanju zavisi samo od tog stanja i akcije, ali nizovi akcija koje radimo su u korelaciji, tačnije trenutna akcija je na neki način uslovljena prethodnom. Recimo ako robot želi da hoda, svako savijanje zgloba je akcija i naravno da se mora uraditi određeni niz savijanja kod koga uvek nakon određene akcije mora da sledi neka određena druga (da robot ne bi pao). Te akcije su onda u korelaciji.

**Zašto je to loše, jer ako učimo iz takvog niza nećemo mnogo istraživati, vrtećemo se oko nekog određenog niza akcija.

Obučavanje Q-mreže: *Experience Replay*

Učenje na osnovu niza uzastopnih semplova se empirijski pokazalo kao problematično:

- Semplovi su u korelaciji=> neefikasno učenje
- Trenutni paramettri Q-mreže faktički diktiraju akcije tj. koji ćemo sledeći simpl dobiti za učenje (npr. ako je trenutno očekivanu nagradu maksimizuje akcija „levo“, semplovi na osnovu koga učimo biće puni samo stanja iz levog dela porostora problema) => ovo nas može dovesti do petlji

Problem se rešava pomoću tehnike ***experience replay***

- Kako radimo akcije ishode (s_t, a_t, r_t, s_{t+1}) čuvamo u memoriji (**replay memory**) – ovo je ustvari skup našeg iskustva (**experience**)
- Q-mrežu obučavamo na mini-podskupovima odabranim na slučajan način iz **replay** memorije. Pošto bismo na slučajan način, mreža neće učiti iz niza uzastopnih akcija.

Obučavanje Q-mreže: *Experience Replay*

Učenje na osnovu niza uzastopnih semplova se empirijski pokazalo kao problematično:

- Semplovi su u korelaciji=> neefikasno učenje
- Trenutni paramettri Q-mreže faktički diktiraju akcije tj. koji ćemo sledeći sempl dobiti za učenje (npr. ako je trenutno očekivanu nagradu maksimizuje akcija „levo“, semplovi na osnovu koga učimo biće puni samo stanja iz levog dela porostora problema) => ovo nas može dovesti do petlji

Problem se rešava pomoću tehnike ***experience replay***

- Kako radimo akcije ishode (s_t, a_t, r_t, s_{t+1}) čuvamo u memoriji (**replay memory**) – ovo je ustvari skup našeg iskustva (**experience**)
- Q-mrežu obučavamo na mini-podskupovima odabranim na slučajan način iz **replay** memorije. Pošto biramo na slučajan način, mreža neće učiti iz niza uzastopnih akcija.

Na ovaj način svako iskustvo (s_t, a_t, r_t, s_{t+1}) može da bude izvučeno više puta i da doprinese obučavanju mreže u različitim momentima – kod Q-učenja jedno iskustvo menja Q-vrednost samo tada kada ga iskusimo => sada imamo efikasnije korišćenje podataka (*data efficiency*)

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

← Inicijalizujemo replay memoriju i težine Q-mreže

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

← Igramo M epizoda igre (jedna epozioda je igranje igre od početka do kraja)

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

← na početku svake epizode inicijalizujemo početno stanje (frejmovi sa početka igre)

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

← Za svaki vremenski korak t u igri

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

← Sa malom
verovatnoćom akciju
birmo slučajno
(istraživanje), inače
birmo po trenutnoj
politici.

Politiku tj. mrežu optimizujemo tako da je u
svakom stanju najbolja akcija ona koja
maksimizuje nagradu, pa će politika odabrati
baš tu akciju sada. To je *greedy* algoritam ili
pojam *exploitation* u RL obasti.

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

← Radimo akciju a_t u okruženju i iskusimo odgovor okruženja tj. nagradu r_t i sledeće stanje s_{t+1}

Obučavanje Q-mreže: *Deep Q-Learning with Experience Replay*

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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

Iskustvo koje smo
dobili od okruženja
skladištimo u
replay memoriji



Obučavanje Q-mreže: Deep Q-Learning with Experience Replay

Sklapamo sve što smo do sada prikazali u jedan algoritam:

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**

 Initialize 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}

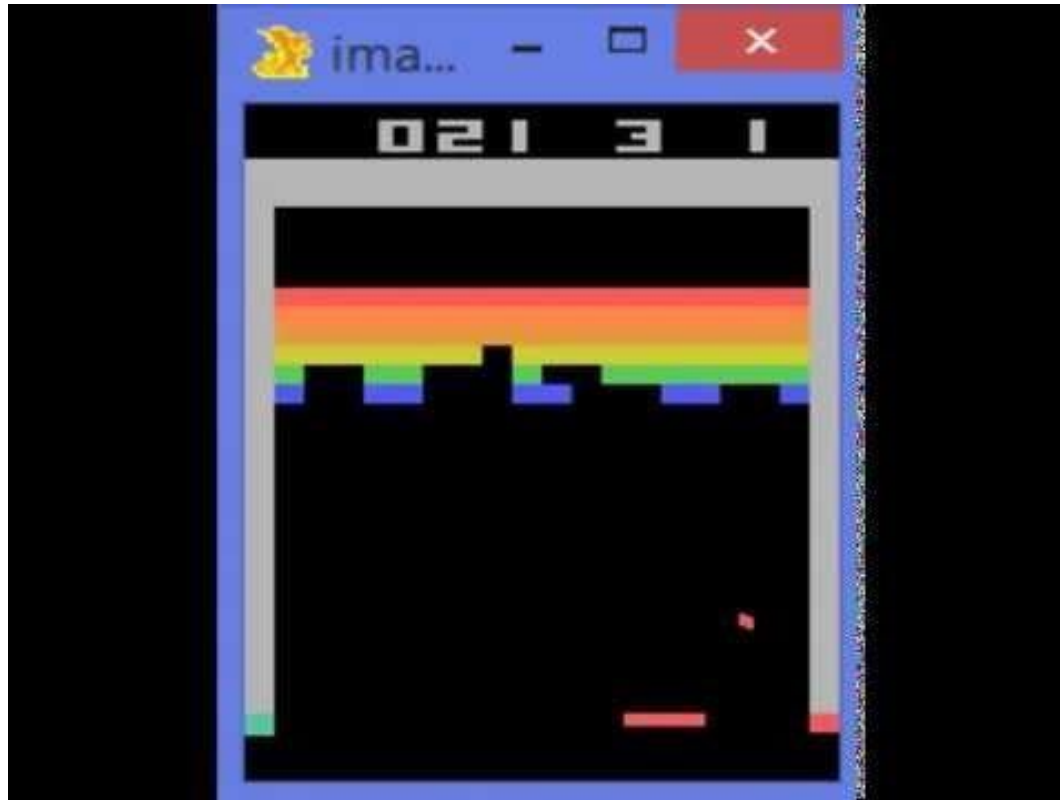
 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

Experience Replay:
Na slučajan način
biramo mini-podskup
iskustava iz replay
memorije i radimo jedan
korak gradient descent
algoritma



<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

Video by Károly Zsolnai-Fehér. Reproduced with permission.