Aproksymacja funkcji wartości

Aproksymacja funkcji wartości

- Problemy z którymi mieliśmy dotychczas do czynienia były relatywnie niewielkie.
- W przypadku klasycznych gier złożoność (mierzona jako rząd wielkości przestrzeni stanów) wygląda następująco:
  - Warcaby (8x8): 10^{20}
  - Szachy: 10^{47}
  - Hex (11x11): 10^{57}
  - Go (19x19): 10^{170}
- Problemy niebędące grami potrafią być jeszcze bardziej skomplikowane.

Aproksymacja funkcji wartości

- Bazowy sposób rozwiązywania MDP, polegający na uaktualnianiu macierzy funkcji wartości V lub Q jest nieefektywny dla tak dużych problemów.
  - Wymagałby nakładów czasu i mocy obliczeniowej dążących do nieskończoności.

Aproksymacja funkcji wartości

- Bazowy sposób rozwiązywania MDP, polegający na uaktualnianiu macierzy funkcji wartości V lub Q jest nieefektywny dla tak dużych problemów.
  - Wymagałby nakładów czasu i mocy obliczeniowej dążących do nieskończoności.
- Zamiast tego można estymować wartość funkcji wartości dla stanu s stosując do tego odpowiednią aproksymatę funkcji wartości:

$$\hat{V}(s,\theta) \approx V(s)$$
$$\hat{Q}(s,a,\theta) \approx Q(s,a)$$

Aproksymacja funkcji wartości

- Zamiast tego można estymować wartość funkcji wartości dla stanu s stosując do tego odpowiednią aproksymatę funkcji wartości:

$$\hat{V}(s,\theta) \approx V(s)$$
$$\hat{Q}(s,a,\theta) \approx Q(s,a)$$

Gdzie θ jest wektorem wag krótszym niż przestrzeń stanów s.

- Dzięki temu jesteśmy w stanie wyestymować funkcję wartości dla wielu przyszłych stanów bazując na relatywnie niewielkim doświadczeniu.
- Intuicja stojąca za takim rozwiązaniem jest prosta – generalizujemy nasze poprzednie doświadczenia na teraźniejsze i przyszłe doświadczenia, które są do nich podobne.

Aproksymacja funkcji wartości

- Sposobów aproksymacji funkcji wartości jest wiele, ich wybór zależy od problemu, który jest rozpatrywany.
- Skupmy się na metodach, które są **różniczkowalne** i dodatkowo pozwalają na aproksymację procesów, które są **niestacjonarne** i nie są **niezależnymi zmiennymi losowymi o jednakowym rozkładzie.**

Aproksymacja funkcji wartości

- Zdefiniujmy **średniokwadratowy błąd wartości** *(Mean Squared Value Error)*:

$$MSVE(\theta) = \sum_{s} d(s)\left(V_\pi(s) - \hat{V}(s,\theta)\right)^2$$

- Gdzie $d(s)$ oznacza przeciętny czas spędzony w stanie s w wyniku korzystania ze strategii π.
- Przy tak zdefiniowanym błędzie jesteśmy w stanie przedstawić problem aproksymacji funkcji wartości jako problem minimalizacji średniokwadratowego błędu wartości.
- Dzięki temu możemy wykorzystać **kombinacje liniowe** lub **metody gradientowe** do rozwiązania tego zadania.

Aproksymacja liniowa

- Przedstawmy stan s jako wektor pewnych cech *(feature vector)*:

$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

- W szczególności możemy przechowywać wszystkie stany w formie odpowiedniej tablicy *(table lookup)*:

$$x_{table}(S) = \begin{pmatrix} I(S = s_1) \\ \vdots \\ I(S = s_m) \end{pmatrix}$$

Aproksymacja liniowa

- Na tej podstawie możemy przedstawić aproksymację funkcji wartości jako kombinację liniową cech:

$$\hat{V}(s,\theta) = x(s)^{\top}\theta = \sum_{i=1}^{n} x_i(s)\theta_i$$

- I w rezultacie zdefiniować funkcję celu jako:

$$MSVE(\theta) = E_{\pi}[(V_{\pi}(s) - x(s)^{\top}\theta)^2]$$

- Otrzymując:

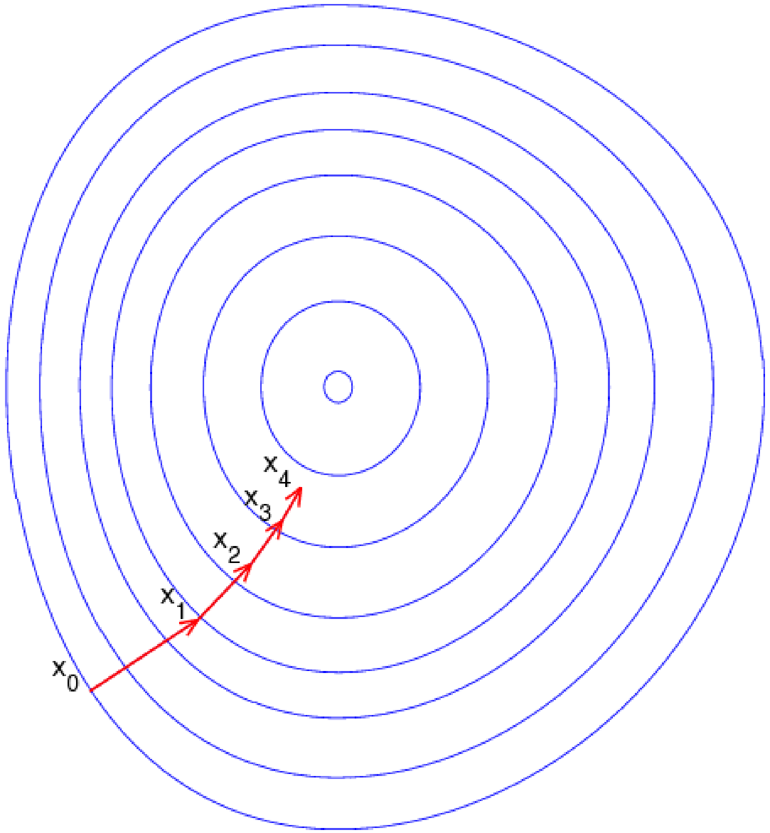
$$\nabla_{\theta}(\hat{V}(s_t,\theta)) = \mathrm{x(s)}$$

$$\Delta\theta = \alpha\left(V_{\pi}(s) - \hat{V}(s,\theta)\right)x(s)$$

Metody gradientowe

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
$k \leftarrow 1$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
$k \leftarrow k + 1$
end while



Metody gradientowe

- Problemem jest jednak to, że w trakcie uczenia nie dysponujemy dokładną wartością funkcji $V_\pi(s)$, a jedynie jej estymacją (niekoniecznie nieobciążoną) $U_\pi(s)$.
- O ile w przypadku metod Monte Carlo nie jest to problemem, to w przypadku metod opartych o bootstrapping (programowanie dynamiczne i temporal difference learning):

MC:

$$\Delta\theta = \alpha\left(R_t - \hat{V}(s_t,\theta)\right)\nabla_\theta(\hat{V}(s_t,\theta))$$

TD:

$$\Delta\theta = \alpha\left(r_{t+1} + \beta\hat{V}(s_t,\theta) - \hat{V}(s_t,\theta)\right)\nabla_\theta(\hat{V}(s_t,\theta))$$

Metody gradientowe

- Ze względu na zależność od aktualnej wartości parametru θ metody te nazywa się metodami **semi-gradientowymi**.
- Ryzyko rozbieżności w ich przypadku zależy przede wszystkim od trzech charakterystyk:
  - Aproksymacji funkcji
  - Bootstrapowania
  - Bycia metodą off-policy

Metody gradientowe

- Ze względu na zależność od aktualnej wartości parametru θ metody te nazywa się metodami **semi-gradientowymi**.
- Ryzyko rozbieżności w ich przypadku zależy przede wszystkim od trzech charakterystyk:
  - Aproksymacji funkcji
  - Bootstrapowania
  - Bycia metodą off-policy

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Metody gradientowe

- Istnieją sposoby rozwiązania tego problemu:
  - **Gradient-TD** (Maei 2011)
  - **Proximal-gradient-TD** (Mahadevan 2015)
  - **Emphatic-TD** (Sutton, White & Mahmood 2015, Yu 2015).
- Innym sposobem jest wykorzystanie **metody najmniejszych kwadratów** do aproksymacji funkcji wartości.

Metoda najmniejszych kwadratów

- Dla $\hat{V}(s,\theta) \approx V(s)$ i doświadczenia D:
$$D = \{(s_1, V_1^\pi), (s_2, V_2^\pi), \dots, (s_T V_T^\pi)\}$$
- Możemy zdefiniować problem minimalizacji sumy kwadratów błędów:
$$LS(\theta) = \sum_{t=1}^{T} \left(V_t^\pi(s) - \hat{V}_t(s,\theta)\right)^2 = E_D[\left(V_\pi(s) - \hat{V}(s,\theta)\right)^2]$$

Metoda najmniejszych kwadratów

- W minimum oczekiwany przyrost $LS(\theta)$ musi być równy 0:

$$E_D[\Delta\theta] = 0$$

$$\alpha \sum_{t=1}^{T} x(s_t)(V_t^\pi - x(s_t)^\top \theta) = 0$$

$$\sum_{t=1}^{T} x(s_t)V_t^\pi = \sum_{t=1}^{T} x(s_t)x(s_t)^\top\theta$$

$$\theta = \left(\sum_{t=1}^{T} x(s_t)x(s_t)^\top\right)^{-1} \sum_{t=1}^{T} x(s_t)V_t^\pi$$

Metoda najmniejszych kwadratów

- W minimum oczekiwany przyrost $LS(\theta)$ musi być równy 0:

$$E_D[\Delta\theta] = 0$$

$$\alpha \sum_{t=1}^{T} x(s_t)(V_t^{\pi} - x(s_t)^{\top}\, \theta) = 0$$

$$\sum_{t=1}^{T} x(s_t)V_t^{\pi} = \sum_{t=1}^{T} x(s_t)x(s_t)^{\top}\theta$$

$$\theta = \left(\sum_{t=1}^{T} x(s_t)x(s_t)^{\top}\right)^{-1} \sum_{t=1}^{T} x(s_t)V_t^{\pi}$$

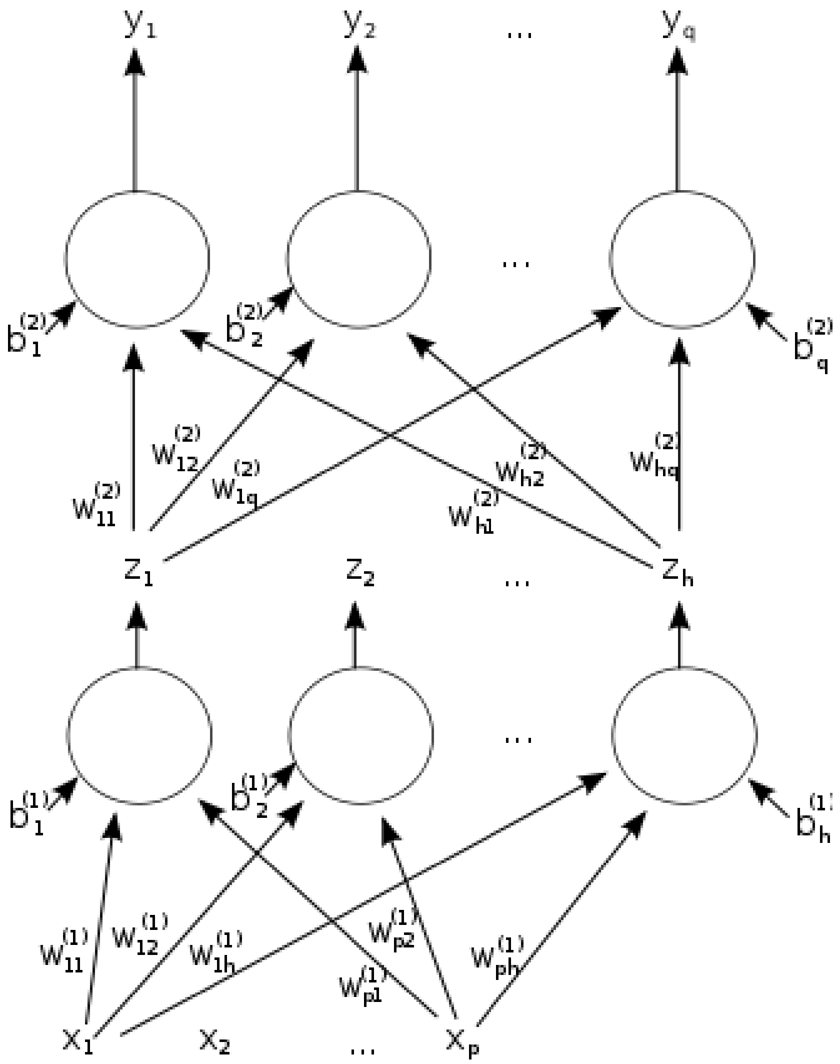
- Niestety dla N cech złożoność obliczeniowa (najlepszym przypadku) wynosi $\mathrm{O}(N^2)$.

Deep Reinforcement Learning

Deep Reinforcement Learning

- Niestety metody liniowej aproksymacji niekoniecznie są efektywne w przypadku nietrywialnych problemów.
- Rozsądnym rozwiązaniem jest zastosowanie efektywniejszej funkcji aproksymującej.
- Na przykład **sieci neuronowej**.

Deep Reinforcement Learning



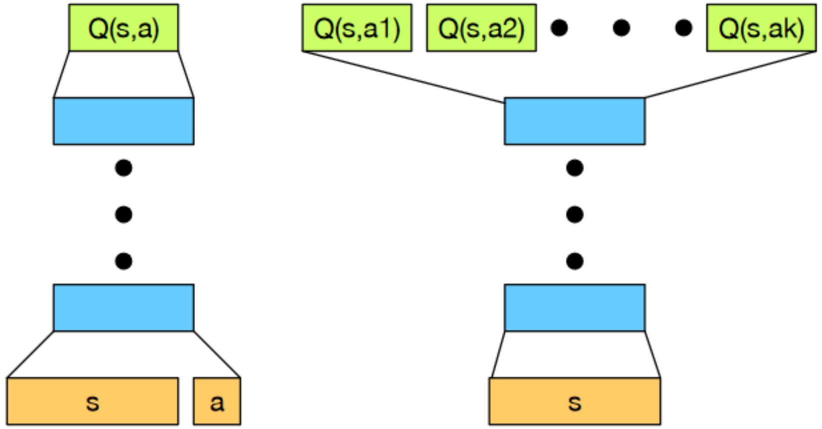
Deep Reinforcement Learning

- Naszym celem jest przedstawienie aproksymacji funkcji wartości akcji $\hat{Q}(s, a, \theta)$ jako sieci neuronowej.

Deep Q-Learning

- Naszym celem jest przedstawienie aproksymacji funkcji wartości akcji $\hat{Q}(s, a, \theta)$ jako sieci neuronowej.
- Można to zrobić na dwa sposoby:

- There are two architectures:
  1. Q-network takes an input s, a and produces $Q(s, a)$
  2. Q-network takes an input s and produces a vector $Q(s, a_1), \cdots, Q(s, a_k)$

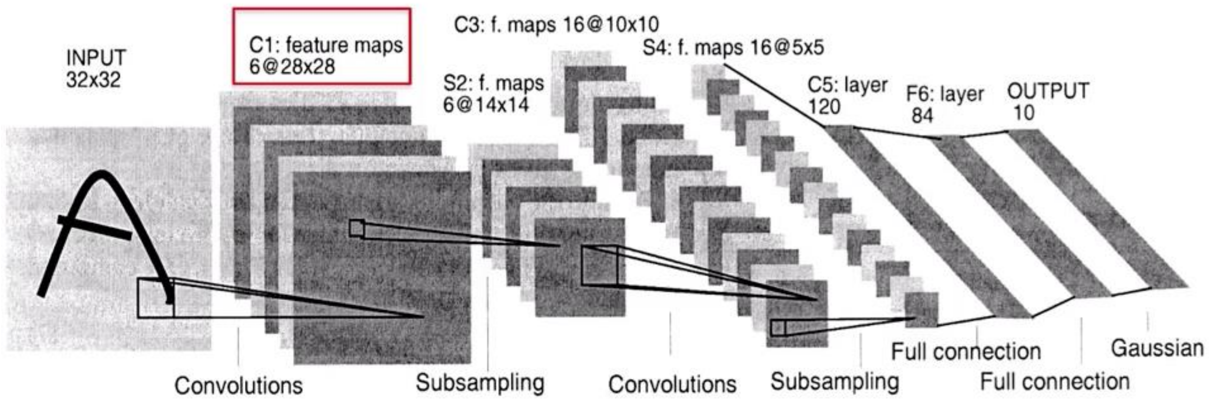


Źródło: http://mi.eng.cam.ac.uk/~mg436/LectureSlides/MLSALT7/L6.pdf

Deep Q-Learning

- Typowymi architekturami wykorzystywanymi w deep Q-learningu są sieci konwolucyjne:

The architecture of LeNet5



Deep Q-Learning

- Naszym celem jest przedstawienie aproksymacji funkcji wartości akcji $Q(s, a, \theta)$ jako sieci neuronowej.
- Uczenie sieci polega na minimalizacji średniokwadratowego błędu wartości:

$$MSVE = \left(r + \beta \max_{a'} Q(s', a', \theta) - Q(s, a, \theta) \right)^2$$

- Jak wiemy jednak ten estymator będzie obciążony.
  - Stany są skorelowane
  - Funkcja celu Q jest niestacjonarna
- Co zrobić w takim wypadku?

Deep Q-Learning

- Należy założyć, że agent będzie wykorzystywał w procesie uczenia swoje doświadczenie z przeszłości *(experience replay)* i że uczyć się będzie na podstawie stałych wag θ^{-} *(fixed Q-targets)*.
- Losowanie kroków służących do uczenia się na bazie swojego poprzedniego doświadczenia rozwiązuje problem skorelowania stanów.
- Przyjęcie stałych wag θ^{-} (i uaktualnianie ich co pewien czas) rozwiązuje problem niestacjonarnej funkcji celu.

Deep Q-Learning

1. W każdej iteracji t:
   - Wybierz akcję a_t za pomocą strategii $\epsilon - zachłannej$.
   - Zapisz krok $(s_t, a_t, r_{t+1}, s_{t+1})$ w pamięci D.
   - Wylosuj dowolny krok (s, a, r, s') z D.
   - Zoptymalizuj $MSVE(\theta_t)$ pomiędzy siecią neuronową Q i docelowym Q:

$$MSVE(\theta_t) = E_{s,a,r,s' \sim D}\left[\left(r + \beta \max_{a'} Q(s', a', \theta_t^-) - Q(s, a, \theta_t)\right)^2\right]$$

Przykład

Mountain Car

https://en.wikipedia.org/wiki/Mountain_car_problem