

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:

$$\begin{aligned}\hat{V}(s, \theta) &\approx V(s) \\ \hat{Q}(s, a, \theta) &\approx Q(s, a)\end{aligned}$$

Aproksymacja funkcji wartości

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

$$\begin{aligned}\hat{V}(s, \theta) &\approx V(s) \\ \hat{Q}(s, a, \theta) &\approx Q(s, a)\end{aligned}$$

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

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

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

- 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 funkcji wartości

- 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*):

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

Reprezentacja cech

- Kluczowym pytaniem jest jak reprezentować stan s .
- Oczywiście chcemy żeby wektor cech $x(s)$ był możliwie jak najmniejszy.
- Ale chcemy też żeby był w stanie reprezentować efektywnie reprezentować stan s :
 - W wielu przypadkach bazowe przedstawienie stanu s niesie ze sobą wiele niepotrzebnych informacji – chcemy, żeby reprezentacja $x(s)$ była w stanie je odpowiednio filtrować.
 - Dodatkowo chcemy żeby reprezentacja $x(s)$ uwzględniała interakcje pomiędzy atrybutami stanu s .

Reprezentacja cech

- Co w takim razie możemy zrobić:
 - Samodzielnie skonstruować wektor cech (np. implementując oczy lub czujniki za pomocą których agent odbiera bodźce z otoczenia).
 - Przetransformować zmienne korzystając z jednej z wielu dostępnych metod:

Reprezentacja cech

Przetransformować zmienne korzystając z jednej z wielu dostępnych metod:

- Wielomianów
- Transformat Fouriera

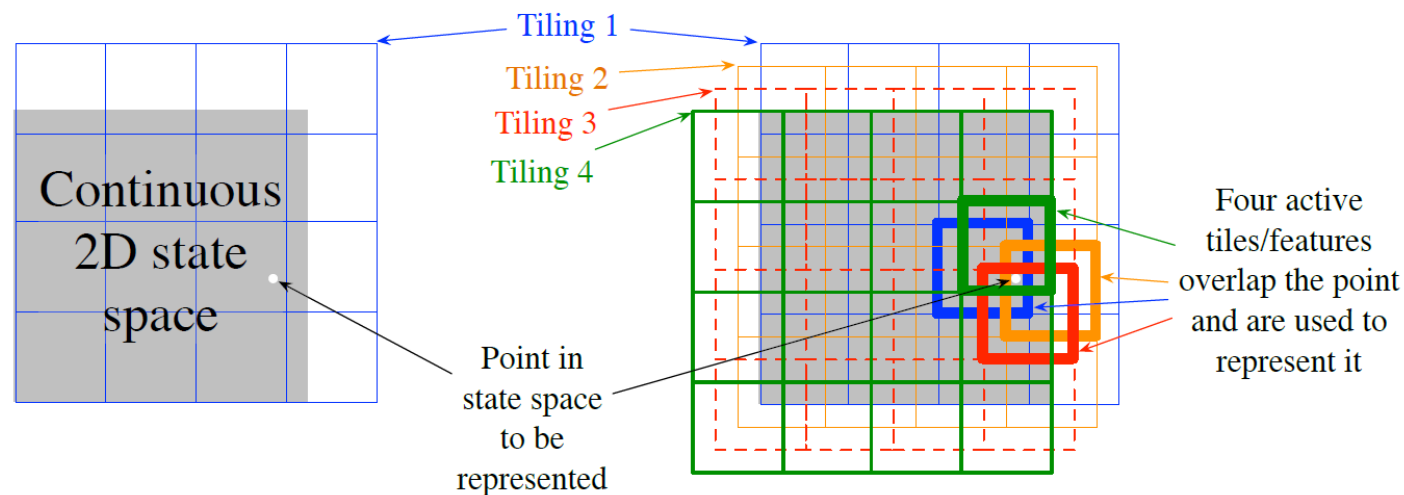
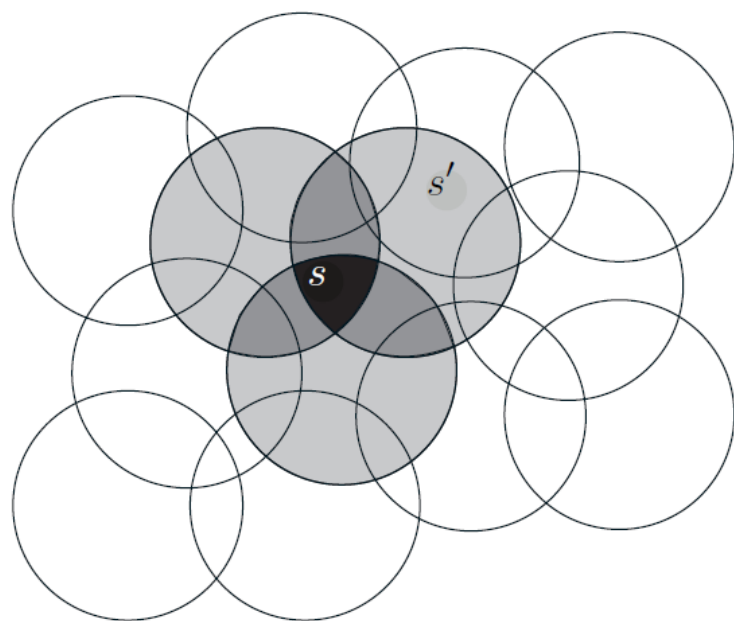
Reprezentacja cech

Przetransformować zmienne korzystając z jednej z wielu dostępnych metod:

- Wielomianów
- Transformat Fouriera
- Kodowania dyskretnego (*coarse coding/tile coding*) (pokrywamy m - wymiarową przestrzeń kulami o promieniu r lub kratą równomierną i kodujemy zmienne w taki sposób, że ich reprezentacją jest wektor binarny wskazujący do których kul/kwadratów wpadł dany stan)

Reprezentacja cech

Kodowania dyskretnego (*coarse coding/tile coding*) (pokrywamy m -wymiarową przestrzeń kulami o promieniu r lub kratą równomierną i kodujemy zmienne w taki sposób, że ich reprezentacją jest wektor binarny wskazujący do których kul/kwadratów wpadł dany stan)



Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

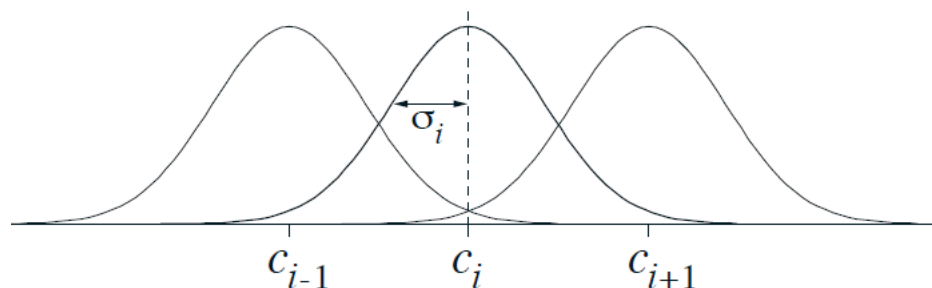
Reprezentacja cech

Przetransformować zmienne korzystając z jednej z wielu dostępnych metod:

- Wielomianów
- Transformat Fouriera
- Kodowania dyskretnego (*coarse coding/tile coding*)
- **Radialnych funkcji bazowych (*Radial Basis Functions, RBF*)**. RBF to taka funkcja ϕ , która zależy jedynie od odległości od określonego punktu referencyjnego (*centrum*) c : $\phi(x) = \phi(\|x - c\|)$. W przypadku funkcji gaussowskiej, taka reprezentacja przyjęłaby postać:

$$\phi(s) = \exp\left(-\frac{\|s - c_i\|^2}{\sigma_i^2}\right)$$

gdzie c_i to jeden z przyjętych m punktów referencyjnych, a σ_i to jego szerokość (odchylenie standardowe).



Reprezentacja cech

Przetransformować zmienne korzystając z jednej z wielu dostępnych metod:

- Wielomianów
- Transformat Fouriera
- Kodowania dyskretnego (*coarse coding/tile coding*)
- Radialnych funkcji bazowych (*Radial Basis Functions, RBF*).
- W przypadku danych relacyjnych reprezentowanych na grafie, do reprezentacji przestrzeni stanów można wykorzystać jedną z wielu technik osadzania (*embedding*).

Metody gradientowe

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$

Require: Initial parameter θ

$k \leftarrow 1$

while stopping criterion not met **do**

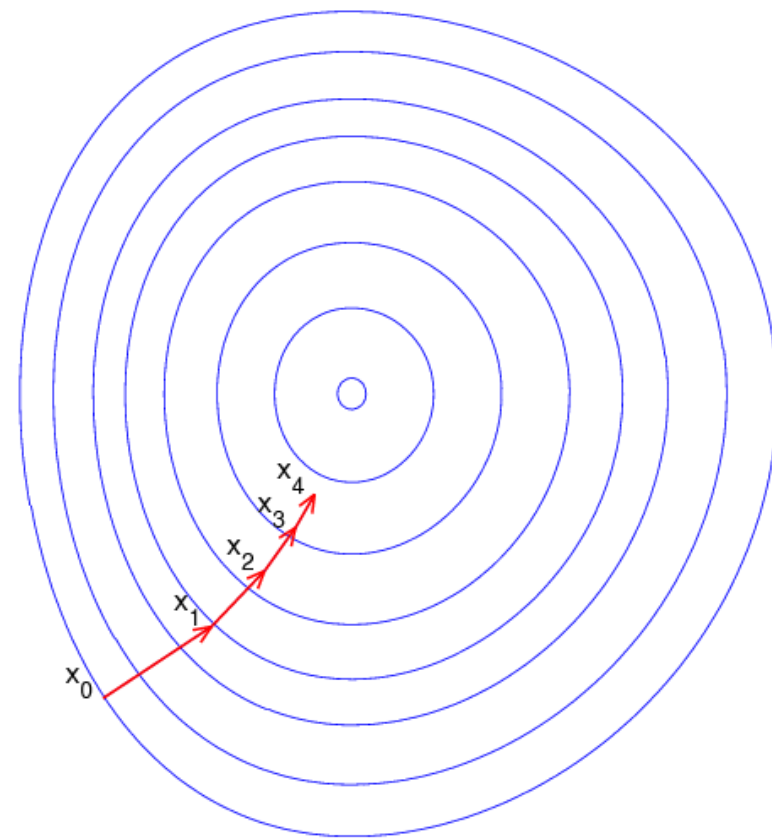
Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

Apply update: $\theta \leftarrow \theta - \epsilon_k \hat{\mathbf{g}}$

$k \leftarrow k + 1$

end while



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:

$$\begin{aligned} \nabla_{\theta}(\hat{V}(s_t, \theta)) &= x(s) \\ \Delta\theta &= \eta \left(V_{\pi}(s) - \hat{V}(s, \theta) \right) x(s) \end{aligned}$$

Metody gradientowe

- Problemem jest jednak to, że w trakcie uczenia nie dysponujemy dokładną wartością funkcji $V_{\pi}(s)$.
- Z tego powodu musimy ją zastąpić jej **przybliżeniem**.

Metody gradientowe

- W zależności od rodzaju algorytmu uaktualnienie wygląda następująco:

MC:

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

TD(0):

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

Forward – view TD(λ):

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

Backward– view TD(λ):

$$\begin{aligned} \delta &= \left(r_{t+1} + \beta \hat{V}(s_{t+1}, \theta) - \hat{V}(s_t, \theta) \right) \\ e_t &= \beta e_{t-1} + \nabla_{\theta}(\hat{V}(s_t, \theta)) \\ \Delta\theta &= \eta \delta e_t \end{aligned}$$

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:
 - Charakteru aproksymowanej 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:
 - Charakteru aproksymowanej 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

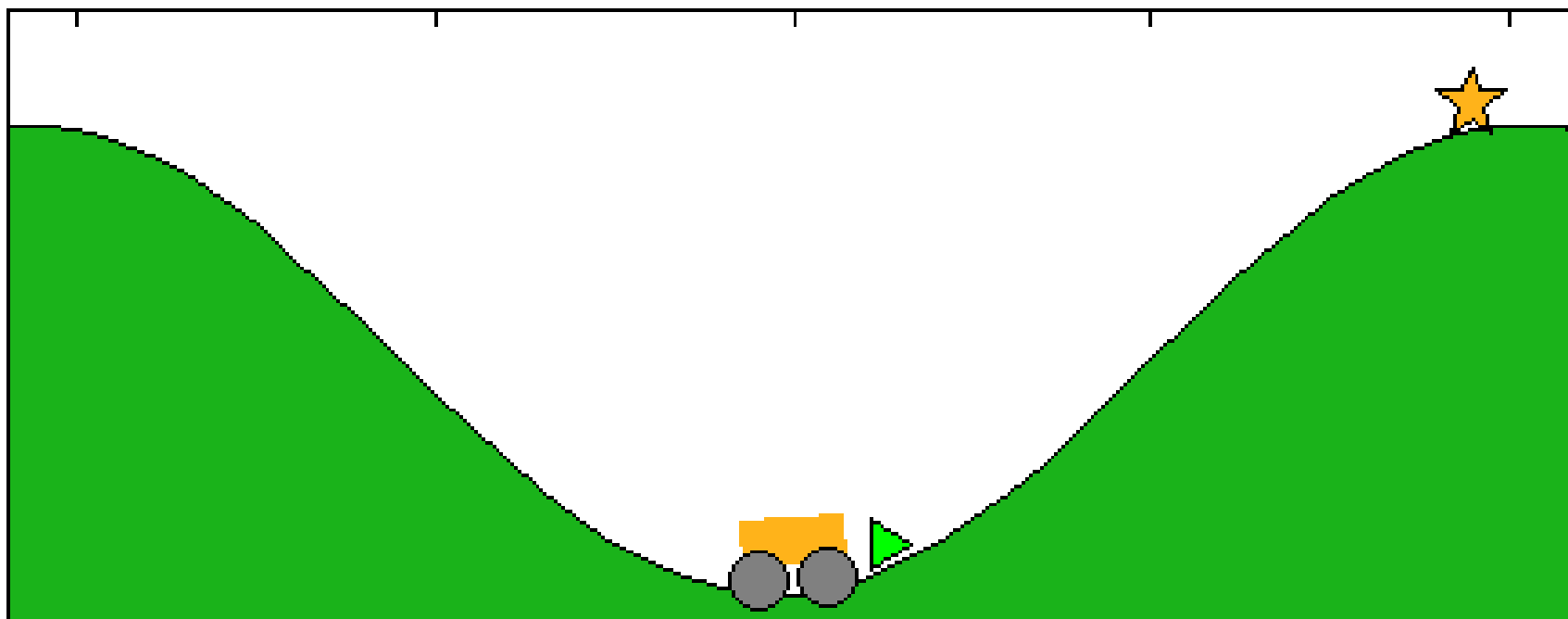
Metody gradientowe

- Próbkowanie ważności (*importance sampling*) pozwala na choć częściowe obejście tego problemu.
- Istnieją też modyfikacje algorytmów usprawniające uczenie na podstawie aproksymowanej funkcji wartości:
 - **Gradient-TD** (Maei 2011)
 - **Proximal-gradient-TD** (Mahadevan 2015)
 - **Emphatic-TD** (Sutton, White & Mahmood 2015, Yu 2015).
- Innym sposobem jest wykorzystanie doświadczenia agenta (danych zebranych na podstawie jego poprzednich kroków) do aproksymacji funkcji wartości.

Przykład

Mountain Car

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



Experience Replay

Experience Replay

- Zamiast uaktualniać gradient w każdym kroku możemy rozwiązać problem za pomocą metody *experience replay*.
- Aby oszacować $\hat{V}(s, \theta) \approx V(s)$ zdefiniujemy doświadczenie D :
$$D = \{(s_1, V_1^\pi), (s_2, V_2^\pi), \dots, (s_T, V_T^\pi)\}$$
- Wtedy problem aproksymacji funkcji wartości sprowadza się do problemu uczenia nadzorowanego.
- Do jego rozwiązania możemy wykorzystać różne algorytmy.
- Ale:
 - Nadal musimy dokonywać rewaluacji $\hat{V}(s, \theta)$ w każdym kroku algorytmu używając do tego uaktualnionego doświadczenia D !

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:

$$LS(\theta) = \sum_{t=1}^T (V_t^\pi(s) - \hat{V}_t(s, \theta))^2 = E_D \left[\left(V_\pi(s) - \hat{V}(s, \theta) \right)^2 \right]$$

Metoda najmniejszych kwadratów

- W przypadku ewaluacji strategii uczenie polega na zebraniu odpowiedniego doświadczenia i rozwiązaniu równania:

$$\theta_{\pi} = \operatorname{argmax}_{\theta} LS(\theta)$$

Metoda najmniejszych kwadratów

- Korzystając z liniowej aproksymacji funkcji wartości $\hat{V}(s, \theta) = x(s)^\top \theta$
- W minimum oczekiwany przyrost $LS(\theta)$ musi być równy 0:

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

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

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

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

Metoda najmniejszych kwadratów

- Korzystając z liniowej aproksymacji funkcji wartości $\hat{V}(s, \theta) = x(s)^\top \theta$
- W minimum oczekiwany przyrost $LS(\theta)$ musi być równy 0:

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

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

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

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

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

Metoda najmniejszych kwadratów

- Algorytm predykcji za pomocą metody najmniejszych kwadratów wygląda następująco:

Metoda najmniejszych kwadratów

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $Q_0(s, a) = 0$ i dowolną strategię π .
2. W każdej iteracji k :
 - Wygeneruj T kroków doświadczenia $D = (s_0, a_0, r_1), \dots, (s_{T-1}, a_{T-1}, r_T)$ na podstawie strategii π .
 - Bazując na doświadczeniu D oszacuj wartość funkcji $\hat{Q}(s, a)$ za pomocą metody najmniejszych kwadratów
 - Popraw zaproponowaną strategię zachowując się zachłannie:

$$\pi(s) = \operatorname{argmax}_{a \in A} \hat{Q}(s, a)$$

Metoda najmniejszych kwadratów

Lub:

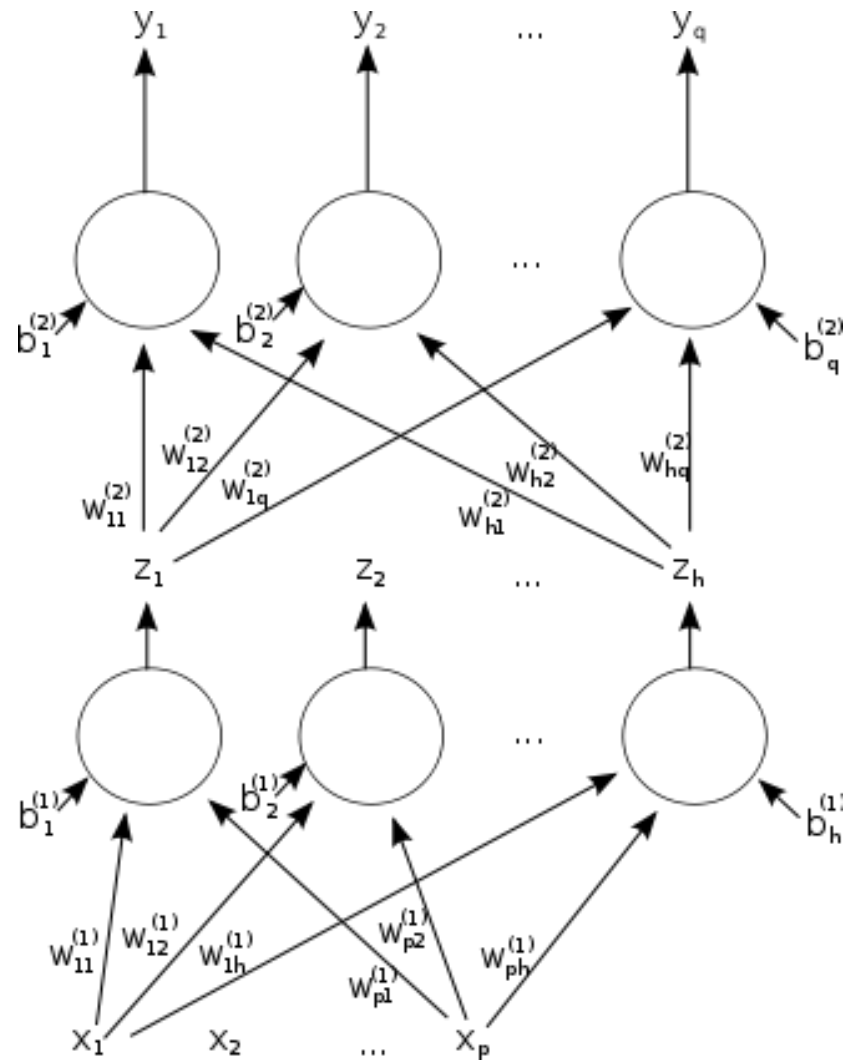
1. Zainicjuj algorytm wybierając dowolne początkowe przybliżenie wartości funkcji wartości $\hat{Q}(s, a)$.
2. W każdej iteracji t :
 - Wybierz akcję a_t za pomocą strategii ϵ – *zachłannej*.
 - Zapisz krok $(s_t, a_t, r_{t+1}, s_{t+1})$ w pamięci D .
 - Wylosuj batch T kroków $\{(s_i, a_i, r_i, s'_i)\}_{i \in \{1, \dots, T\}}$ z D .
 - Bazując na wylosowanym doświadczeniu oszacuj wartość funkcji $\hat{Q}(s, a)$ za pomocą metody najmniejszych kwadratów

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

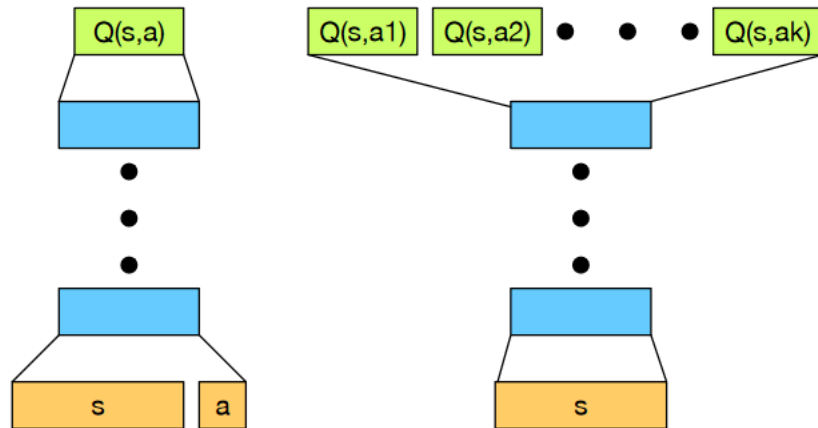
- Naszym celem jest przedstawienie aproksymacji funkcji wartości akcji $\hat{Q}(s, a, \theta)$ jako sieci neuronowej.

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:

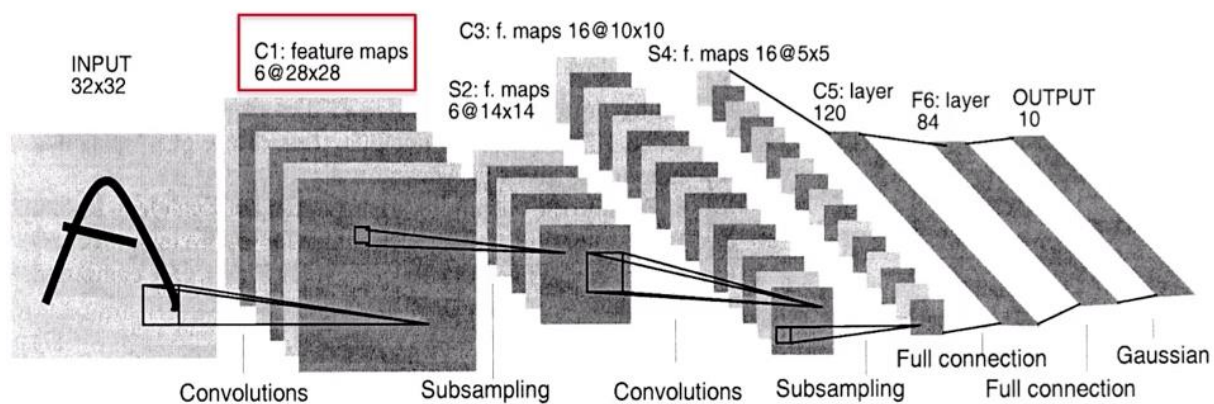
1. Q-network takes an input s, a and produces $Q(s, a)$
2. Q-network takes an input s and produces a vector $Q(s, a_1), \dots, Q(s, a_k)$



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. Zainicjuj algorytm wybierając dowolne początkowe przybliżenie wartości funkcji wartości $\hat{Q}(s, a)$.
2. W każdej iteracji t :
 - Wybierz akcję a_t za pomocą strategii ϵ – *zachłannej*.
 - Zapisz krok $(s_t, a_t, r_{t+1}, s_{t+1})$ w pamięci D .
 - Wylosuj batch m kroków $\{(s_i, a_i, r_i, s'_i)\}_{i \in \{1, \dots, m\}}$ z D .
 - Zoptymalizuj $MSVE(\theta_t)$ pomiędzy siecią neuronową \hat{Q} i przybliżeniem docelowego Q :
$$MSVE(\theta_t) = E_{s,a,r,s' \sim D} \left[\left(r + \beta \max_{a'} \hat{Q}(s', a', \theta_k^-) - \hat{Q}(s, a, \theta_t) \right)^2 \right]$$
 - Co k iteracji (jeżeli $\text{mod}(t, k) = 0$) przypisz nowe wagi $\theta_k^- := \theta_t$

Przykład

Mountain Car

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

