გამოწრთობით დასწავლის მიმოხილვა

Reinforcement learning overview

ირაკლი კობერიძე

დოქტორანტის სემინარი 1

ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ფაკულტეტი: ზუსტ და საბუნებისმეტყველო მეცნიერებათა

პროგრამა: კომპიუტერული მეცნიერება

ხელმძღვანელი: პროფესორი მაგდა ცინცაძე, ფიზ.მათ. მეცნიერებათა კანდიდატი

10 ივლისი 2024 წ.

აბსტრაქტი

ეს ნაშრომი მიმოიხილავს გამოწრთობით დასწავლის (Reinforcement Learning, RL) ევოლუციას. განხილვას ვიწყებთ ფუნდამენტური პრინციპებიდან როგორიცაა მარკოვის გადაწყვეტილების პროცესი და მისი ამოხსნა დინამიური პროგრამირების მეშვეობით. შემდეგ კი განვიხილავთ დროებითი სხვაობის პრინციპს რომელიც საშუალებას გვაძლევს გარემოს მოდელის გარეშე გავწვრთნათ აგენტი ასევე ქიუ-დასწავლას (Q-Learning) რომელიც ეფექტურად მოიხმარს გამოცდილებას ოპტიმალური გადაწყვეტილებების მისაღებად. ასევე გავარჩევთ თუ როგორ შეიძლება მდგომარეობებს შორის განზოგადოება ფუნქციის მიახლოების გზით, მაგალითად ხელოვნური ნეირონული ქსელით.

1. შესავალი

გამოწრთობით დასწავლა არის მანქანური სწავლების მიდგომა რომელშიც აგენტი სწავლობს გამოწრთობის გზით, იგი ურთიერთქმედებს გარემოსთან კონკრეტული მიზნის მისაღწევად და იხვეწება თავისივე წარმატებულ და წარუმატებელ მცდელობების ხარჯზე. მეთოდი გამოიყენება რობოტიკაში, თამაშებში და სხვადასხვა ავტონომურ სისტემებში. [1]

დარგის ევოლუცია იწყება ისეთი კონცეფციებიდან როგორიცაა მარკოვის გადაწყვეტილების პროცესი და დინამიური პროგრამირება. მარკოვის გადაწყვეტილების პროცესი (Markov Decision Process, MDP) გვაძლევს მათემატიკურ ჩარჩოს რომელშიც შეგვიძლია გადაწყვეტილებების მიღების მოდელირება გარემოში, რომელშიც მდგომარეობის ცვლილება და ჯილდო ალბათობაზეა დამოკიდებული [3]. ხოლო ბელმანის გამოსახულებამ მოგვცა საშუალება დინამური პროგრამირების გზით დაგვედგინა ამა თუ იმ მდგომარეობისა და ქმედების სარგებლიანობა, რომელიც შემდეგ გამოიყენება ოპტიმალური გადაწყვეტილების მიღებისათვის [4].

გამოწრთობით დასწავლაში დიდი წინსვლა გამოიწვია დროებითი სხვაობით დასწავლამ (Temporal Difference, TD) [5]. რაც აერთიანებს დინამიურ პროგრამირებას მონტე კარლო მეთოდთან. ეს საშუალებას იძლევა აგენტმა ისწავლოს გადაწყვეტილებების სწორად მიღება გარემოს მოდელის გარეშე მაგალითად ისეთი ალგორითმით როგორიცაა SARSA [6].

შემდგომ დიდ მიღწევას ვატკინის ნამუშევარში ვიპოვნით. მან შეიმუშავა ქიუ დასწავლის (Q-learning) ალგორითმი [7], რომელიც წინამორბედ ალგორითმებისგან განსხვავებით შესაძლებელს ხდიდა აგენტს ესწავლა მაშინაც კი როდესაც იგი არ მიყვებოდა პოლისს (მოქცევის წესს) [8].

წარმატებების მიუხედავად ამ ალგორითმების დიდ პრობლემას წარმოადგენდა რომ პრაქტიკული პრობლემები იყო მრავალ-განზომილებიანი ხოლო მეთოდი კი ვერ ანხორციელებდა განზოგადოებას [9]. ამ პრობლემის გადასაჭრელად ფუნქციის მიახლოების მეთოდების გამოყენება გახდა საჭირო [10]. მათ შორის კი ყველაზე პოპულალური და ეფექტური ხელოვნური ნეირონული ქსელები აღმოჩნდნენ [11].

2. მარკოვის გადაწყვეტილების პროცესი

მარკოვის გადაწყვეტილების პროცესი ფართოდ გამოიყენება გამოწრთობით დასწავლაში რადგან მისი მეშვეობით შესაძლებელია გარემოში გადაწყვეტილებების მიღების და მათი შედეგების მოდელირება [3].

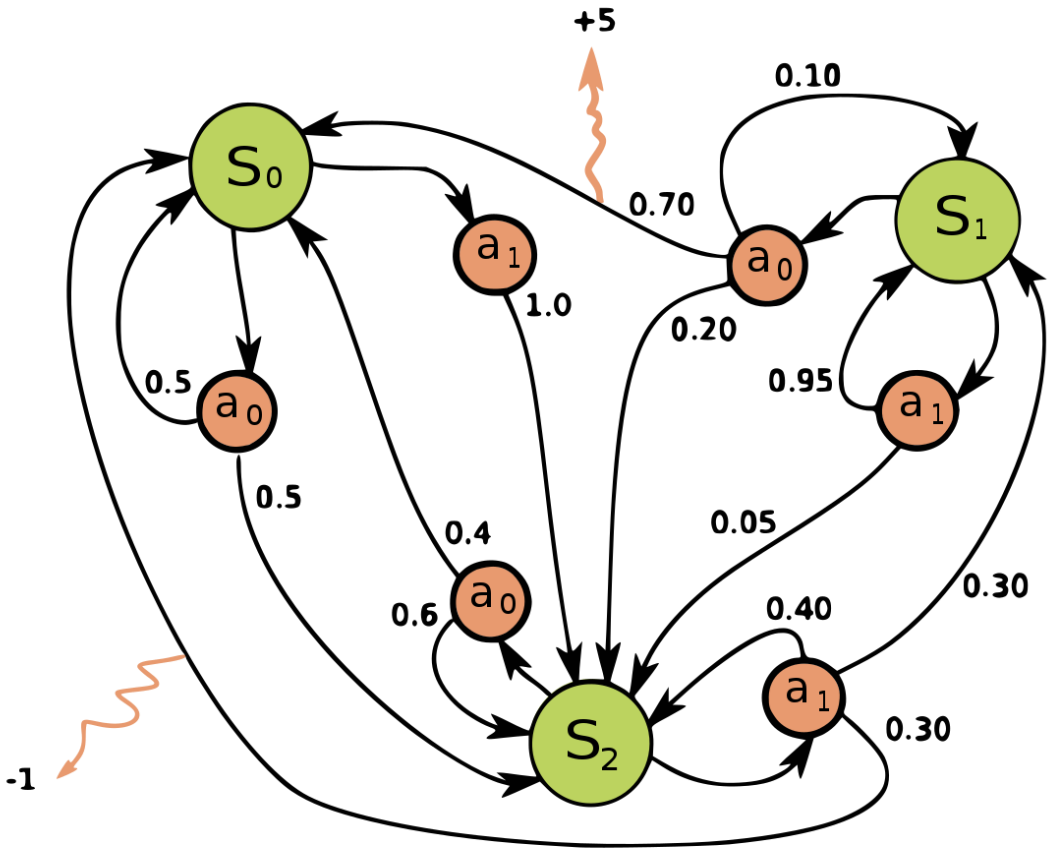
იგი შედგება შემდეგი კომპონენტებისაგან:

- მდგომარეობები (S): ეს არის სიმრავლე რომელიც მოიცავს ყველა იმ შესაძლო მდგომარეობას რომელშიც აგენტი შეიძლება აღმოჩნდეს. მდგომარეობა მოიცავს ყველაფერს რაც გვჭირდება რომ ვიცოდეთ გარემოს შესახებ კონკრეტულ მომენტში. მაგალითად ჭადრაკში მდგომარეობამ შეიძლება გვითხრას ფიგურების პოზიციები დაფაზე.

- მოქმედებები (A): ყველა იმ შესაძლო მოქმედებების სიმრავლე რომელიც აგენტმა შეიძლება მოიმოქმედოს. ქმედებები შეიძლება იყოს დამოკიდებული მდგომარეობაზე. მაგალითად ჭადრაკში შესაძლებელი ქმედებები იცვლება ფიგურების განლაგების მიხედვით.
- გადასვლის ფუნქცია (T, ან P): კონკრეტული მდგომარეობიდან სხვა მდგომარეობაში გადასვლის ალბათობა თუ მოვიმოქმედებთ კონკრეტულ ქმედებას. ხშირად აღინიშნება როგორც $P(s'|s,a)$, რაც აღნიშნავს s' მდგომარეობაში გადასვლას მდგომარეობა s იდან, თუ ქმედება a იყო განხორციელებული. მაგალითად თუ რობოტი არის კონკრეტულ პოზიციაში (მდგომარეობა) და ეცდება რომ გადაადგილდეს წინ (ქმედება), ეს ფუნქცია მოგვცემს ბევრ ახალ ადგილას აღმოჩენის ალბათობას ამ ქმედების შემდეგ.
- ჯილდოს ფუნქცია (R): მყისიერად მიღებული ჯილდო რომელსაც აგენტი იღებს ერთი მდგომარეობიდან მეორეში გადასვლისას მის მიერ განხორციელებული ქმედების შემდეგ. იგი შეიძლება იყოს აღნიშნული როგორც $R(s,a,s')$ ან $R(s,a)$, რაც გვეუბნება თუ რამხელა ჯილდოს მიიღებს აგენტი როდესაც მდგომარეობა s ში იგი განახორციელებს ქმედება a ს. მაგალითად ჯილდო შეიძლება იყოს პოზიტიური თუ რობოტი დანიშნულების ადგილს მიაღწევს, ან ნეგატიური თუ იგი გზაში რამეს დაეჯახება.
- ფასდაკლების ფაქტორი (γ): ფასდაკლების ფაქტორის ($0 \leq \gamma \leq 1$) რიცხვითი მნიშვნელობა განსაზღვრავს თუ რამდენად ნაკლებად ვაფასებთ შორ მომავალ ჯილდოებს ახლო მომავალში შესაძლო მისაღებ ჯილდოებთან შედარებით. მაგალითად თუ რიცხვითი მნიშვნელობა იქნება 1, მაშინ აგენტი შორ მომავალ ჯილდოებს ისე განიხილავს როგორც ახლანდელს. თუმცა რაც უფრო ვუახლოვდებით ნოლს მით უფრო მცირე შეფასებას მისცემს აგენტი შორ მომავალ ჯილდოებს. ეს კომპონენტი ხშირად არ არის განხილული მარკოვის გადაწყვეტილების პროცესში თუმცა გვხდება გამოწრთობით დასწავლაში.

მარკოვის გადაწყვეტილების პროცესს ასევე აქვს თვისება რომელიც ცნობილია როგორც „მარკოვის თვისება“ [12]. რომელიც გვეუბნება რომ მომავალი მდგომარეობა

დამოკიდობულია მხოლოდ აწმყო მდგომარეობაზე და არა იმ ქმედებათა რიგზე რომელმაც ამ მდგომარეობამდე მოგვიყვანა. მაგალითად თუ ავიღებთ ჭადრაკში კონკრეტულ ფიგურების განლაგებას, ის თუ როგორ მოვედით ამ მდგომარეობამდე არ ცვლის თუ რა იქნება შემდეგი ოპტიმალური სვლა, ანუ ყველაფერი ისედაც ვიცით რაც საჭიროა სწორი გადაწყვეტილების მისაღბად. ფიგურა 1 გვაძლევს მარკოვის გადაწყვეტილების პროცესის ვიზუალურ სახეს.



ფიგურა 1: მარკოვის გადაწყვეტილების პროცესის მაგალითი. 3 მდგომარეობით 2 ქმედებით და 2 ჯილდოთი (სტაფილოსფერი კლაკნილი ისარი) [15]

შემდეგი სექცია განიხილავს თუ როგორ ვიპოვოთ ოპტიმალუქი ქმედება როდესაც მოცემული გვაქვს მარკოვის გადაწყვეტილების პროცესი დინამიური პროგრამირების გზით.

3. დინამური პროგრამირება გამოწრთობით დასწავლაში

დინამური პროგრამირების გზით შესაძლებელია ამოვხსნათ მარკოვის გადაწყვეტილების პროცესი. ერთ-ერთი ალგორითმი რომელიც ამისთვის გამოიყენება არის ცნობილი როგორც ფასეულობაზე იტერაცია (Value Iteration) [1].

3.1 ფასეულობაზე იტერაცია

ეს არის ალგორითმი რომელიც გამოითვლის V^* - ს, რომელიც არის ოპტიმალური ფასეულობის ფუნქცია. შესაბამისად იგი გვეუბნება თუ რომელი მდგომარეობა რამდენად ფასეულია აგენტისთვის. ამ ფუნქციის გამოყენებით აგენტს შეუძლია მიიღოს ოპტიმალური გადაწყვეტილება ნებისმიერ მდგომარეობაში.

ალგორითმი კი შემდეგნაირია:

- ინიციალიზაცია: ყველა შესაძლო მდგომარეობისთვის s თავიდან ვთვლით რომ $V(s)$ ტოლია 0 - ის.
- ფასეულობის განახლება: ყველა მდგომარეობისთვის მოვახდინოთ ფასეულობის განახლებას ბელმანის გამოსახულების მეშვეობით [4].

$$V_{k+1}(s) = \max_{\mathrm{a}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

სადაც:

  - $P(s'|s,a)$ არის გადასვლის ალბათობა მდგომარეობა s' ში თუ აგენტი იყო მდგომარეობა s - ში და განახორციელა ქმედება a.
  - $R(s,a,s')$ არის ჯილდო რომელსაც აგენტი მიიღებს როდესაც იგი არის მდგომარეობა s - ში და ქმედება a - ს შედეგად იგი აღმოჩნდება მდგომარეობა s' - ში.

- γ არის ფასდაკლების ფაქტორი
- k კი იტერაციის ინდექსი

- კონვერგენციის შემოწმება: ვამოწმებთ თუ ფუნქცია V აღარ განიცდის ცვლილებას ან თუ ცვლილება უფრო მცირეა ვიდრე რაიმე კონკრეტული შერჩეული ზღვარი, ამ შემთხვევაში ვაჩერებთ იტერაციას.
- პოლისის შემუშავება: იტერაციის დასრულების შემდეგ ვიყენებთ ფუნქცია V - ს რათა შევიმუშაოთ პოლისი π^* რომელიც არის ოპტიმალური მოქცევის წესი. ეს პოლისი გვეუბნება თუ რა მდგომარეობაში რა ქმედება არის ოპტიმალური. ანუ რა ქმედება უნდა განახორციელოს აგენტმა რომ გადაინაცვლოს უფრო მაღალი შეფასების მდგომარეობაში.

$$\pi^* = \arg\max_a \sum P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

3.2 ფასეულობის იტერაციის მაგალითი

მაგალითად გვაქვს მარტივი სამყარო 3 მდგომარეობით $S = \{1,2,3\}$ და 2 მოქმედებით $A = \{a_1, a_2\}$. შემდეგი გადასვლის ალბათობებით და ჯილდოებით:

ამჟამინდელი მდგომარეობა s	ქმედება a	მომდევნო მდგომარეობა s'	გადასვლის ალბათობა $P(s'\|s,a)$	ჯილდო $R(s,a)$
1	a_1	2	1	5
1	a_2	3	1	10
2	a_1	3	1	-1
2	a_2	1	1	2
3	a_1	1	1	7
3	a_2	2	1	0

მოვახდენთ ფასეულობის ფუნქციის ინციალიზაციას:

$$V_0(1) = 0, V_0(2) = 0, V_0(3) = 0$$

- პირველი იტერაცია:

მდგომარეობა s = 1

$$\begin{aligned} V_1(1) &= \max\{P(2|1,a_1)[R(1,a_1) + \gamma V_0(2)], P(3|1,a_2)[R(1,a_2) + \gamma V_0(3)]\} \\ &= \max\{1.0[5 + 0.9 \cdot 0], 1.0[10 + 0.9 \cdot 0]\} \\ &= \max\{5, 10\} \\ &= 10 \end{aligned}$$

მდგომარეობა s = 2

$$\begin{aligned} V_1(2) &= \max\{P(3|2,a_1)[R(2,a_1) + \gamma V_0(3)], P(1|2,a_2)[R(2,a_2) + \gamma V_0(1)]\} \\ &= \max\{1.0[-1 + 0.9 \cdot 0], 1.0[2 + 0.9 \cdot 0]\} \\ &= \max\{-1, 2\} \\ &= 2 \end{aligned}$$

მდგომარეობა s = 3

$$\begin{aligned} V_1(3) &= \max\{P(1|3,a_1)[R(3,a_1) + \gamma V_0(1)], P(2|3,a_2)[R(3,a_2) + \gamma V_0(2)]\} \\ &= \max\{1.0[7 + 0.9 \cdot 0], 1.0[0 + 0.9 \cdot 0]\} \\ &= \max\{7, 0\} \\ &= 7 \end{aligned}$$

- მეორე იტერაცია

მდგომარეობა s = 1

$$\begin{aligned} V_2(1) &= \max\{P(2|1,a_1)[R(1,a_1) + \gamma V_1(2)], P(3|1,a_2)[R(1,a_2) + \gamma V_1(3)]\} \\ &= \max\{1.0[5 + 0.9 \cdot 2], 1.0[10 + 0.9 \cdot 7]\} \\ &= \max\{5 + 1.8, 10 + 6.3\} \\ &= \max\{6.8, 16.3\} \\ &= 16.3 \end{aligned}$$

მდგომარეობა s = 2

$$\begin{aligned} V_2(2) &= \max\{P(3|2,a_1)[R(2,a_1)+\gamma V_1(3)], P(1|2,a_2)[R(2,a_2)+\gamma V_1(1)]\} \\ &= \max\{1.0[-1+0.9\cdot 7], 1.0[2+0.9\cdot 10]\} \\ &= \max\{-1+6.3, 2+9\} \\ &= \max\{5.3, 11\} \\ &= 11 \end{aligned}$$

მდგომარეობა s = 3

$$\begin{aligned} V_2(3) &= \max\{P(1|3,a_1)[R(3,a_1)+\gamma V_1(1)], P(2|3,a_2)[R(3,a_2)+\gamma V_1(2)]\} \\ &= \max\{1.0[7+0.9\cdot 10], 1.0[0+0.9\cdot 2]\} \\ &= \max\{7+9, 0+1.8\} \\ &= \max\{16, 1.8\} \\ &= 16 \end{aligned}$$

საკმარისი იტერაციების შემდეგ ჩვენ მივალთ პოლისთან რომლის მიხედვითაც პირველ და მეორე მდგომარეობებში მეორე ქმედება ითვლება ოპტიმალურად. ხოლო მესამე მდგომარეობიაში კი პირველი ქმედება.

3.3 პოლისზე იტერაცია

არსებობს კიდევ ერთი პოპულალური ალგორითმი რომელიც ცნობილია როგორც პოლისზე იტერაცია (Policy iteration). იგი არის უფრო ეფექტური ვიდრე ფასეულობაზე იტერაცია იმ მხრივ თუ რამდენი იტერაცია არის საჭირო რომ სწავლების პროცესი დასრულდეს.

ალგორითმი:

- ინიციალიზაცია: ვიწყებთ შემთხვევითი ქმედებებით გავსებული პოლისით π
- პოლისის შეფასება: გამოვთვალოთ V^{π} ყოველი მდგომარეობისთვის

$$V^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

- პოლისის განახლება: შევარჩევთ ყველა მდგომარეობისთვის ისეთ ქმედებას რომელიც მოსალოდნელი ჯილდოების მაქსიმიზაციას ახდენს

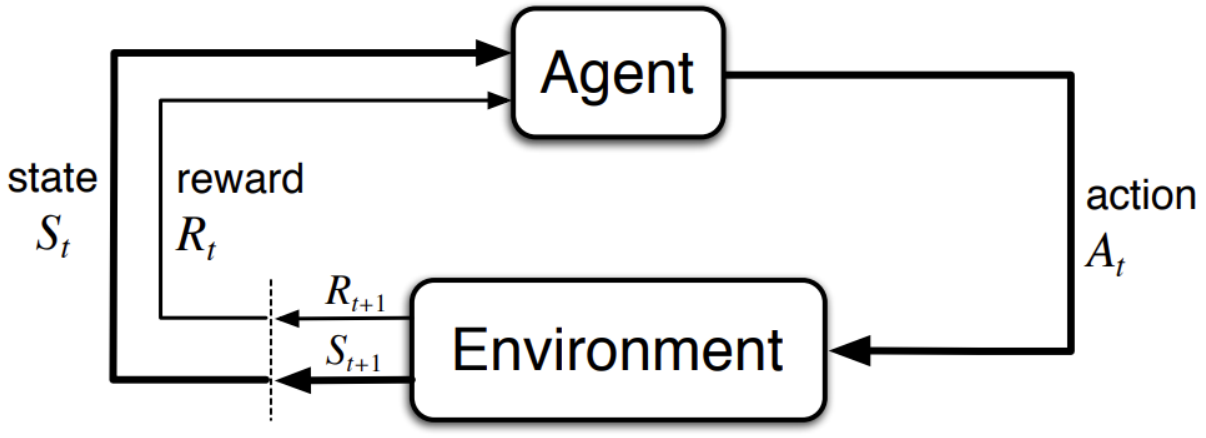
$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

- როდესაც პოლისი შეწყვეტს ცვლილებას შევწყვეტთ იტერაციას

თუმცა პრაქტიკულ პრობლემებში ამგვარი მიდგომა ნაკლებად გამოსადეგია რადგან გადასვლის ფუნქცია იშვიათად გვაქვს. შემდეგი სექცია განიხილავს თუ როგორ ავუაროთ ამ პრობლემას გვერდი.

4. დროებითი სხვაობით დასწავლა (Temporal Difference Learning)

დროებითი სხვაობით დასწავლა აერთიანებს დინამიური პროგრამირების და მონტე კარლოს მეთოდების სიძლიერეს. მონტე კარლოს მეთოდი გამოწრთობის დასწავლის კონტექსტში გულისხმობს რომ სწავლების პროცესში ჩვენ ვახდენთ გარემოსთან ურთიერთბის სიმულაციას. ვაგენერირებთ ეპიზოდებს და მათი შედეგების მიხედვით ვახდენთ ფასეულობის ფუნქციის განახლებას. ეპიზოდი არის ერთი სრული სიმულაცია სანამ აგენტი არ მივა საბოლოო მდგომარეობამდე. მაგალითად ჭადრაკში ერთი ეპიზოდი შეიძლება იყოს ერთი მთლიანი ხელი თავიდან ბოლომდე. ფიგურა 2 ახდენს ამ დინამიკის ვიზუალიზაციას.



ფიგურა 2: აგენტისა და გარემოს ურთიერთქმედების დინამიკა [1]

დრებითი სხვაობა არის მეთოდი რომლითაც შეგვიძლია ვივარაუდოდ ფასეულობის ფუნქცია სხვადასხვა მდგომარეობებისთვის, რაც წარმოადგენს მოსალოდნელ სარგებელს რომელსაც აგენტი მიიღებს როდესაც აღმოჩნდება კონკრეტულ მდგომარეობაში და მიყვება კონკრეტულ პოლისს [13].

ამ მეთოდის ძირითადი აზრი არის რომ გავანალხოთ ფასეულობის ვარაუდი $V(s)$ შემდეგი მდგომარეობის $V(s')$-ის მიხედვით და მყისიერი ჯილდოს გათვალისწინებით. ეს პროცესი შეიძლება იყოს ფორმალიზებული შემდეგი გამოსახულებით:

$$V(s) \leftarrow V(s) + \alpha[R + \gamma V(s') - V(s)]$$

სადაც:

- $V(s)$ არის ამჟამინდელი ფასეულობის ვარაუდი მდგომარეობა s - ისთვის
- α არის სწავლის ინტენსიურობა
- R არის ჯილდო რომელსაც აგენტი იღებს მდგომარეობა s იდან s' ში გადასვლისას
- γ არის ფასდაკლების ფაქტორი რომელიც განსაზღვრავს თუ რამდენად მნიშვნელოვანია მომავალი ჯილდოები
- $V(s')$ არის ვარაუდი თუ რამდენად ფასეულია შემდეგი მდგომარეობა s'

წევრი $R + \gamma V(s') - V(s)$ არის ცნობილი როგორც დროებითი სხვაობის ცდომილება. რომელიც ზომავს სხვაობას აგენტის ნავარაუდებ ფასეულობასა და სინამდვილეში მიღებულ ფასეულობასთან შორის [13].

წინამდებარე ცოდნიდან გამომდიარე შემუშავებული იქნა ალგორითმი SARSA (State-Action-Reward-State-Action). SARSA არის დროებით სხვაობაზე აგებული ალგორითმი რომელიც ქმედება-ფასეულობის ფუნქციას ანახლებს იმ ქმედებების ხარჯზე რომელსაც იგი ნამდვილად აკეთებს და არა ჰიპოტეთურად აკეთებს. [2]

ალგორითმი შემდეგნაირია:

- ინიციალიზაცია: ყველა შესაძლო ქმედება მდგომარეობა წყვილისთვის ფასეულობის ინციალიზაცია ხდება შემთხვევითი მნიშვნელობებით.
- პოლისი: აგენტმა უნდა აირჩიოს ქმედება a მდგომარება s ში პოლისი π - ს მიხედვით. (ხშირად ეპსილონ ხარბი, განხილული შემდეგ სექციაში)
- იტერაცია: სანამ ეპიზოდი დამთავრდება აგენტი იმეორებს შემდეგს
    - მოიმოქმედოს ქმედება a, ნახოს მიღებული ჯილდო r და შემდეგი მდგომარეობა s'
    - აირჩიოს შემდეგი ქმედება a' მდგომარეობა s' - ისთვის პოლისი π-ს მიხედვით.
    - გაანახლოს ქმედება-ფასეულობის ფუნქცია

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

    - შემდეგი იტერაციისთვის გაანახლოს $s \leftarrow s'$ და $a \leftarrow a'$
- გამეორდეს პროცესი შემდეგი ეპიზოდისთვის მანამ სანამ Q ფუნქცია განიცდის საგრძნობ ცვლილებას. [14]

5. ექსპლუატაციასა და ექსპერიმენტს შორის დილემა

წინა სექციაში აგენტს ჭირდებოდა ქმედებების შერჩევა კონკრეტული პოლისის მიხედვით. როდესაც აგენტი სწავლობს იგი დგება დილემის ქვეშ, მოახდინოს იმ ქმედების ექსპლუატაცია რომელიც უკვე იცის რომ სავარაუდოდ უმეტეს ჯილდოს მისცემს მას, თუ ცადოს რამიე ახალი და იქნებ აღმოჩნდეს რომ ეს ახალი ქმედება მას უფრო მეტ ჯილდოს მოაპოვებინებს. ამ დილემის გადასაჭრელად ყველაზე პოპულალური მიდგომაა ეპსილონ ხარბი ალგორთმი. ამ ალგორითმში ჩვენ გვაქვს ეპსილონი

$$\epsilon$$

რომელის მნიშვნელობაც უნდა იყოს 0-სა და 1-ს შორის. იგი განსზაღვრავს იმის ალბათობას რომ აგენტმა გადაწყვიტეს ექსპერიმენტის ჩატარება, ანუ ცადოს ისეთი ქმედება რომელიც მისი ცოდნის მიხედვით არ არის ოპტიმალური კონკრეტულ მდგომარეობაში. შესაბამისად $1-\epsilon$ არის იმის ალბათობა რომ აგენტმა გადაწყვიტოს ექსპლუატაცია. ანუ გადაწყვიტოს ისეთი ქმედება რომელიც მისი ცოდნიდან გამომდინარე ითვლება ოპტიმალურად [1].

შემდეგი სექცია განიხილავს SARSA ზე უფრო ეფექტულ ალგორითმს იგივე მიზნის მისაღწევად.

6. ქიუ დასწავლა (Q-learning)

ქიუ დასწავლა არის მოდელისგან თავისუფალი გამოწრთობით დასწავლის ალგორითმი რომელშიც აგენტი სწავლობს ოპტიმალურ ქმედების შერჩევის პოლისს გარემოსთან ურთიერთმქედების გზით. ესეც არის დროებითი სხვაობის მეთოდი რომელიც ახდენს მისი ვარაუდის განახლებას გამოცდილებიდან მიღებული ჯილდოების ხარჯზე. ალგორითმი შემდეგნაირად გამოიყურება:

- მოვახდინოთ Q ფუნქციის ინიციალიზაცია შემთხვევითი მნიშვლობებით $Q(s,a)$ - სთვის.
- იტერაციულად ეპიზოდებში:
    - აგენტი აკვირდება მდგომარეობა s-ს
    - ირჩევს ქმედება a-ს პოლისით როგორიცაა მაგალითად ეპსილონ ხარბი
    - ანვახორციელებს ქმედებას
    - ანახლებს ქიუ ფუნქციას შემდეგნაირად

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

    - გადავიდის შემდეგ მდგომარეობაში და იმეორებს პროცესს

6.1 SARSA და Q-Learning შორის განსხვავება

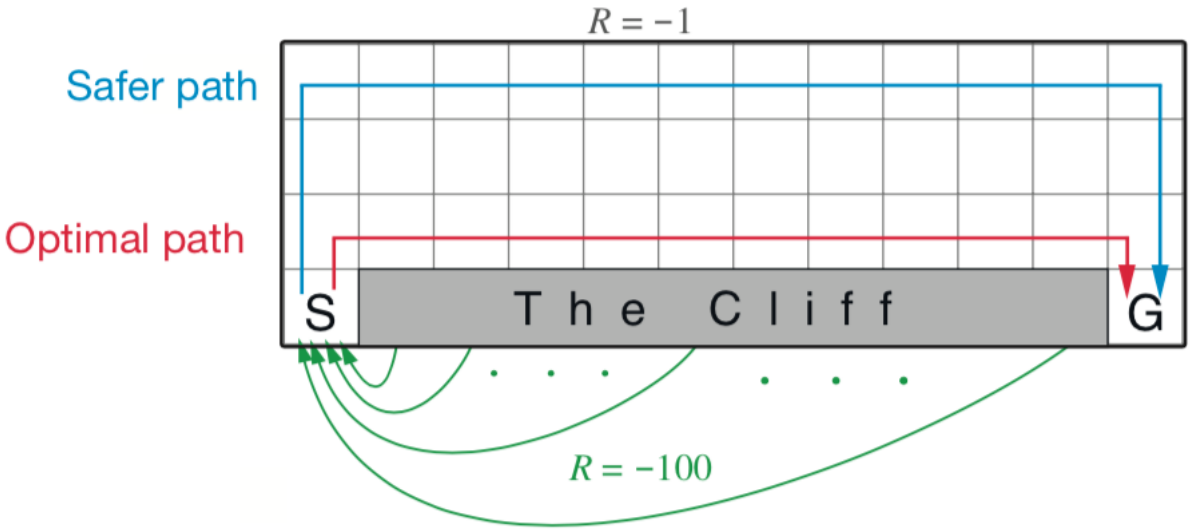
როგორც ზემოთ ვნახეთ ქიუ ფუნქციის განახლება SARSA-სგან განსხვავებულად ხდება. SARSA ქიუ ფუნქციას ანახლებს იმის მიხედვით თუ რომელ პოლისსაც მიყვება აგენტი კონკრეტულ მომენტში. ხოლო Q-Learning აგენტი სწავლობს ოპტიმალურ პოლისს მაშინც კი როდესაც იგი ექსპერიმენტს ატარებს. ეს გამომდინარეობს იქიდან რომ მდგომარეობა-ქმედების ფუნქციის განახლებისას Q-Learning აკვირდება მომდევნო მდგომარეობის მაქსიმალურ ფასეულობას ყველა ქმედებას შორის, და არა იმ ფასეულობას რომელსაც იგი მიიღებდა პოლისის მიყოლის შემთხვევაში. ეს უზრუნველყოფს რომ Q-Learning - მა უფრო მალე ისწავლოს ოპტიმალური პოლისი. თუმცა SARSA უფრო უსაფრთხო პოლისს სწავლობს რადგან იგი ექსპერიმენტულ პოლისსაც სწავლობს.

მაგალითისთვის ავიღოთ თამაში რომელშიც გვაქვს ცხრილისებრი სამყარო სადაც ზოგიერთი უჯრედი აღნიშნავს კლდიდან ჩასავარდნს როგორ ეს არის ფიგურა 3 შია. ამ თამაშში აგენტი იწყებს ეპიზოდს უჯრედში რომელიც აღნიშნულია S ით და მისი მიზანია მოხვდეს G უჯრაში. ყოველ გადადგმულ ნაბიჯზე აგენტი იღებს ჯილდოს მინუს ერთს (შესაბამისად უნდა ეცადოს რომ მალე დაამთავროს), ხოლო თუ კლდიდან გადავარდება მინუს ასს და ბრუნდება პოზიცია S - ში.

როდესაც აგენტი სწავლობს SARSA ალგორითმის მიხედვით მან შეიძლება როდესაც იგი კლდის კიდეშია გადაწყვიტოს კლდიდან გადახტომა. და წინა მდგომარეობას მიანიჭებს ნეგატიურ ქულას რადგან სწორედ იმ მდგომარეობიდან მოხდა გადავარდნა.

თუმცა, როდესაც ქიუ ალგორითმი გადაწყვეტს გადახტომას, იგი წინა მდგომარეობას არ მიანიჭებს ნეგატიურ ქულას რადგან წინა მდგომარეობაში სხვა უფრო კარგი ქმედებებიც არსებობს და ისევ იმ საუკეთესო ქმედების მნიშვნელობას მიანიჭებს იმ უჯრედს რომლიდანაც იგი შემდეგ გადავარდა.

შესაბამისად სარსა აგენტი ჩათვლის ოპტიმალურად რომ გაივლის უსაფრთხო გზით რომელიც ლურჯადაა ფიგურა 3 - ზე, ხოლო ქიუ დასწავლის ალგორითმი იპოვნის ოპტიმალურ უმოკლეს მარშუტს რომელიც წითლადაა აღნიშნული.



ფიგურა 3: კლდეზე სეირნობა, ცხრილისებრი სამყარო [16]

7. ქიუ დასწავლის ალგორითმის დახვეწა ფუნქციის მიახლობით

აქამდე განხილულ მეთოდს აქვს კიდევ ერთი ნაკლი. Q ფუნქცია რომელიც ჩვენ განვიხილეთ არის ტაბულალური ფორმის რომელიც ეფექტურია ისეთ პრობლემებში რომელშიც მცირე მდგომარეობა-ქმედების სივრცეა. თუმცა როდესაც მდგომარეობის რეპრეზენტაცია შეიძლება იყოს უწყვეტი მნიშვნელობების მქონე ტაბულალური ფუნქცია უკვე ხდება არაპრაქტიკული და ზღუდავს მთლიანად ალგორითმის გამოყენებადობას. ფუნქციის დაახლოების ტექნოლოგია ამ შეუზღუდვის გადალახვაში გვეხმარება რადგან მათ შეუძლიათ აიტანონ დიდი და უწყვეტი სივრცეები ეფექტურად.

7.1 რატომ ფუნქციის დაახლოება

- **ტაბულალური Q-learning - ის შეზღუდვები**
    - **სკალირება:** Q ცხრილის ზომა იზრდება ექსპონენციალურად რაც შეუძლებელს ხდის მის გამოყენებას დიდი მასშტაბების პრობლემების გადასაჭრელად.
    - **განზოგადოება:** ტაბულალურ მეთოდს არ აქვს უნარი მოახდინოს განზოგადოება ისეთ მდგომარეობა-ქმედების წყვილებთან რომელიც წვრთნისას აგენტს არ შეხვედრია.
    - **კომპიუტერული მეხსიერების მოთხოვნილება:** ყველა შესაძლებელი მდგომარეობა-ქმედების წყვილის შენახვამ შეიძლება უზარმზარი რესურსი მოითხოვოს კომპიუტერისგან და შესაბამისად იყოს არაპრაქტიკული.
- **ფუნქციის მიახლოების უპირატესობები**
    - **განზოგადოება**: ფუნქციის მიახლოებას შეუძლია განაზაგოდოს მდგომარეობა-ქმედების ისეთ წყვილებზე რომელიც აგენტს წვრთნისას არ შეხვედრია.
    - **მეხსიერების მხრივ ეფექტურობა:** ასეთი ფუნქციები ზოგადად უფრო ნაკლებ კომპიუტერულ მეხსიერებას მოითხოვენ ვიდრე ტაბულალური მეთოდები.
    - **მოქნილობა:** შეუძლიათ ბევრ სხვადასხვა სახის სენსორულ მონაცემებთან მუშაობა როგორიც შეიძლება იყოს სურათები და ხმა.

7.2 Q-Learning განზოგადოება ხელოვნური ნეირონული ქსელით

ალგორითმს შეგვიძლია შევმატოთ განზოგადოების უნარი ტაბულალური Q ფუნქციის ხელოვნური ნეირონული ქსელის ჩანაცვლებით.

7.2.1 ხელოვნური ნეირონული ქსელი

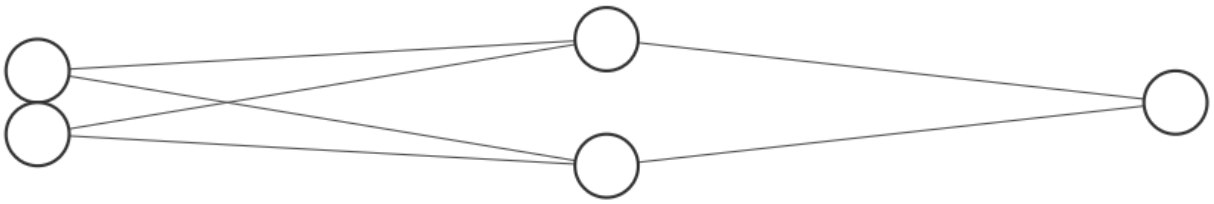
ხელუვნური ნეირონული ქსელი არის გამოთვლადი მოდელი რომლიც ბიოლოგიური ნეირონული ქსელებს გავს. [17]

ძირითადი კომოპნენტები:

- **ნეირონი:** ფუნდამენტური ერთეული რომელიც იღებს მნიშვნელობას, ამუშავებს მას და გადასცემს მომდევნო შრეს.
- **წონები:** პარამეტრები რომელთა მნიშვნელობებიც წვრთნის შედეგად დგინდება.
- **აქტივაციის ფუნქცია:** ფუნქცია რომელიც განსაზღვრავს ნეირონის სიგნალის ინტენსიობას შემდეგ ნეირონებთან

მაგალითი

ვთქვათ შემავალი შრე იღებს ორ მნიშვნელობას x_1 და x_2. გვაქვს მხოლოდ ერთი დამალული შრე (შრეები საწყის და საბოლოო შრეებს შორის) 2 ნეირონით და საბოლოო შრეში მხოლოდ ერთი ნეირონი. ვიზუალურად ისეთი როგორც ფიგურა 4 ზე.



ფიგურა 4: 2x2x1 ხელოვნური ნეირონული ქსელი.

როდესაც ეს ქსელი იღებს შემავალ მნიშვნელობას, საბოლოო შედეგი გამოითვლება შემდეგნაირად:

საწყისი შრე არ ცვლის შემავალ მნიშვნელობებს და აწვდის მათ შემდეგ შრეს. შემდეგ დამალულ შრეში კი ყოველ ნეირონს აქვს იმდენი წონა რამდენი კავშირიც არის მასთან წინა შრიდან და ბიასი. ეს წონები აღინიშნება w - თი ხოლო ბიასი b - თი.

პირველი დამალული ნეირონისთვის:

$$h1 = \sigma(w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + b_1)$$

მეორე დამალული ნეირონისთვის:

$$h2 = \sigma(w_{2,1} \cdot x_1 + w_{2,2} \cdot x_2 + b_2)$$

სადაც $\sigma = \frac{1}{1+e^{-x}}$ და არის აქტივაციის ფუნქცია (სხვა აქტივაციის ფუნქციაც შეიძლება იყოს გამოყენებული).

და ბოლოს გამოვითვლით შედეგს ბოლო შრეში:

$$y = \sigma(w_{out,1} \cdot h1 + w_{out,2} \cdot h2 + b_{out})$$

8. შეჯამება

ამ ტექსტში ჩვენ მიმოვიხილეთ გამოწრთობით დასწავლის ევოლუცია. ჩვენ განვიხილეთ საწყისი პრინციპები როგორიცაა მარკოვის გადაწყვეტილების პროცესი და მისი ამოხნა დინამიური პროგრამირების მეშვეობით. განვიხილეთ ამ მიდგომის შეზღუდვები, რომ იგი დამოკიდებულია გარემოს მოდელზე. შემდეგ განვიხილეთ დროებითი სხვაობის მეთოდი რომელიც საშუალებას იძლევა მოდელის გარეშე განხორციელდეს სწავლება SARSA ალგორითმის მეშვეობით. შემდეგ მის დახვეწილ ვერსიასაც შევეხეთ რომელშიც აგენტი სწავლობს ოპტიმალურ პოლისს. ამ მიდგომის შეზღუდვებზეც ვისაუბრეთ რომელიც ეხება განზოგადოებას და განვიხილეთ თუ როგორ შეიძლება ამ შეზღუდვების გადალხვა ფუნქციის მიახლოების მეთოდით როგორიცაა მაგალითად ხელოვნური ნეირონული ქსელი.

წყაროები

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.
2. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4, 237-285.
3. Puterman, M. L. (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons.
4. Bellman, R. (1957). Dynamic Programming. Princeton University Press.
5. Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3(1), 9-44.
6. Rummery, G. A., & Niranjan, M. (1994). On-line Q-learning using connectionist systems (Vol. 37). University of Cambridge, Department of Engineering.
7. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292.
8. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
9. Kearns, M. J., & Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3), 209-232.
10. Bertsekas, D. P., & Tsitsiklis, J. N. (1996). Neuro-Dynamic Programming. Athena Scientific.
11. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
12. Howard, R. A. (1960). Dynamic Programming and Markov Processes. MIT Press.
13. Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. Communications of the ACM, 38(3), 58-68.
14. Szepesvári, C. (2010). Algorithms for Reinforcement Learning. Morgan & Claypool.

15. By waldoalvarez - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=59364518
16. Jeremy Zhang (2019). Reinforcement Learning — Cliff Walking Implementation https://towardsdatascience.com/reinforcement-learning-cliff-walking-implementation-e40ce98418d4
17. LeCun, Y., Bengio, Y., & Hinton, G. (2015). "Deep learning." Nature, 521(7553), 436-444.