МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета

Програмирање и програмски језици

на тему

DQN агент за дубоко учење са појачавањем

Ученик:
Слободан Јенко, IV_d

Ментори:
Јелена Хаџи-Пурић
Петар Величковић

Београд, мај 2018.

Садржај

1 Увод

Машинско учење је посебна област компјутерских наука која се бави вештачком интелигенцијом. Циљ алгоритама из ове области је да почну од нуле, и успешно "науче" да решавају разне проблеме.
Постоје три основна облика машинског учења:

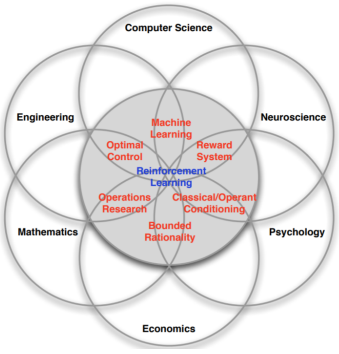
- **Супервизирано учење**.
  Ови алгоритми на основу датих тренинг података и "ознака" уче да означавају до тада невиђене податке. Пример супервизираног учења је препознавање објекта на слици.
- **Несупервизирано учење**
  Сада су алгоритму дати само подаци, а он из њих треба да извуче скривене правилности. Користи се за груписање података.
- **Учење са појачавањем**.

Суштина учења са појачавањем је проблем одлучивања. Због тога се оно налази у пресеку више научних дисциплина као што су рачунарске науке, психологија и неурологија. Мотивација за алгоритме често долази из открића о начину на који наш мозак перципира награде и доноси одлуке. Неки од основнох алгоритама учења са појачавањем су врло слични начину на који функционише допамински систем у људском телу.
У проблеме одлучивања спадају трговина на берзи, управљање возилима, играње игрица... Да би на проблем могли применити учење са појачавањем, довољно је да постоје награде и доношење одлука, па уз добро дефинисане награде у ову категорију можемо сврстати сваку људску активност. Због тога је оно кључ развоја генералне вештачке интелгенције која ће бити једна од највећих прекретница у људској историји.

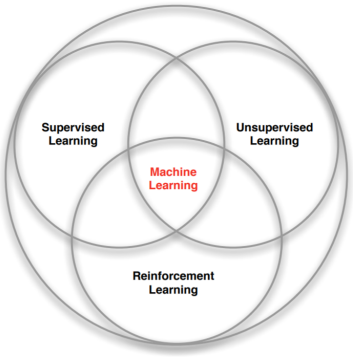
Учење са појачавањем је теже од осталих облика из више разлога:

- Не постоји супервизор, тј. неко ко нам говори који је најбољи потез у одређеној ситуацији. Уместо тога добијамо награде које нам дају меру ваљаности наших акција.
- Награде долазе са закашњењем. Неке акције нам могу донети тренутну позитивну награду а за пар потеза довести до катастрофалног исхода (У шаху можемо замислити ситуацију у којем потез којим смо појели фигуру, противнику омогућили да нас матира).
- Подаци које добијамо зависе од акција које смо изабрали. Дакле морамо на паметан начин бирати акције, да би добили корисне податке.

Такође, супервизирано учење можемо посматрати као поједностављени облик учења са појачавањем.



(a) Учење са појачавањем.



(b) Гране машинског учења.

2 Основни појмови

2.1 Агент и окружење

Наведимо сада не тако строге дефиниције основних појмова које ћемо користити:

Дефиниција 2.1.1. Агент је наш алгоритам који интереагује са окружењем. Те интеракције називамо акције и означавамо их са A_t.

Дефиниција 2.1.2. Опсервација O_t је сигнал који нам даје неку информацију о промени стања система у тренутку t.

Дефиниција 2.1.3. Стање је скуп информација које неки објекат памти. Стањем је одређено понашање тог објекта.

Разликујемо стање окружења (S_t^e) и стање агента (S_t^a). Стање окружења је његова унутрашња репрезентација и углавном је недоступно. Стање агента је агентово виђење окружења и то су подаци које агент користи да би изабрао акцију. Нпр. у видео игрицама стање окружења је садржај меморије, а стање агента слика екрана. У даљем тексту реч стање ће се односити на стање агента и означаваћемо га са S_t.

Треба правити разлику између појмова опсервација и стање. Опсервацију агент добија од окружења после сваке извршене акције. Стање је нека функција свих претходних опсервација, $S_t = f(O_1, ..., O_t)$. Нпр. то могу бити све претходне опсервације или само последња.

Дефиниција 2.1.4. Награда R_{t+1} је скаларни повратни сигнал који нам говори колико је **моментално** добра акција коју је агент изабрао у тренутку t.

Истакнимо још једном да нам тренутна награда не говори колико је заправо добра изабрана акција, зато што акција утиче на читаву будућност. Права вредност акција је кључ решавања неког проблема, јер ако знамо колико акције стварно вреде, знамо и како да се понашамо.

Агент изводи акције унутар окружења, чиме изазива да окружење промени стање, добија наградни сигнал и опсервацију. Циљ агента је да максимизује укупну стечену награду. При игрању видео игрица акције су могући притисци тастера а награде су промене у броју поена.

Дакле, агент је у стању S_t, бира акцију A_t, затим добија опсервацију O_{t+1} и награду R_{t+1}. Индекси су ствар договора, можемо ставити и O_t или R_t. У овом случају, промену стања окружења замишљамо као промену тренутка, па агент награду и опсервацију добија тренутак након што бира акцију.

Универзалност учења са појачавањем се базира на хипотези да све циљеве можемо представити као максимизацију очекиване укупне награде.

Дефиниција 2.1.5. Стање S_t има Марковљево својство акко:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{2.1}$$

Марковљево својство нам говори да будућност зависи само од тренутног стања. Захваљујући томе не морамо да чувамо сва претходна стања.

Дефиниција 2.1.6. Кажемо да је окружење потпуно прегледно ако агент има увид у стање окружења, $O_t = S_t^a = S_t^e$. У супротном се ради о делимично прегледном окружењу.

Ако важи потпуна прегледност, процес интеракције агента са окружењем називамо **Марковљев процес одлучивања (MDP)**. У супротном ради се о **делимично прегледном Марковљевом процесу одлучивања**.

Потпуна прегледност значајно олакшава учење. Да би решили делимично прегледна окружења прво ћемо се позабавити Марковљевим процесима одлучивања. Већина проблема у стварном свету су делимично прегледни, нпр. робот са камером чији је циљ кретање кроз простор види само слику света испред себе, не и своју апсолутну локацију, у видео игрицама доступна нам је само слика екрана, не и меморија.

У случају делимично прегледног MDP-а, агент мора да конструише своје стање као функцију виђених опсервација.

2.2 Компоненте агента

Наш агент може да садржи једну или више следећих компоненти:

- **Полиса** (π): одређује понашање агента. То је функција која слика стања у акције. Може бити детерминистичка $a = \pi(s)$ или стохастична $\pi(a|s) = \mathbb{P}[A = a|S = s]$.
- **Вредносна функција** (v): одређује колико су добра стања. Вредност је наше предвиђање очекиване укупне будуће награде. Приметимо да вредност стања зависи од полисе коју пратимо. Такође ако знамо праве вредности за свако стање, лако можемо конструисати полису тако што "похлепно" бирамо акцију која ће нас довести у стање највеће вредности.

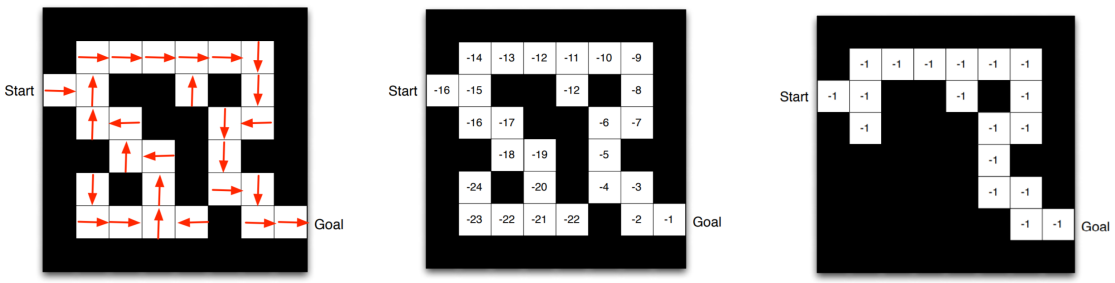
  $$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...|S_t = s] \tag{2.2}$$

  Касније ћемо видети чему служи коефицијент γ.
- **Модел**: предвиђа шта ће окружење урадити. Можемо моделовати:
  - Прелазе(транзиције), тј. вероватноћу да из стања s дођемо у стање s' ако смо изабрали акцију a:

    $$\mathcal{P}^a_{ss'} = \mathbb{P}[S' = s'|S = s, A = a] \tag{2.3}$$

  - Награде, тј. очекивану укупну будућу награду ако почињемо из стања s и бирамо акцију a:

    $$\mathcal{R}^a_s = \mathbb{E}[R|S = s, A = a] \tag{2.4}$$



(a) Полиса. (b) Вредносна функција. (c) Модел.

Слика 2: Компоненте агента на примеру лавиринта. Циљ је што брже изаћи из лавиринта. За сваки корак губи се један поен (добија се награда -1).

Категоризација агената према томе да ли садрже полису или вредносну фукцију:

- Засновани на вредносној функцији.
- Засновани на полиси.
- Актер-критичари: садрже и полису и вредносну функцију.

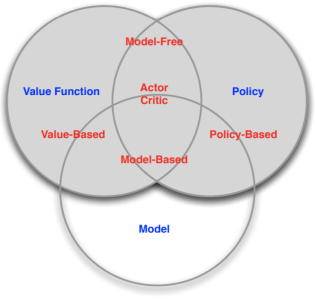
Поред овога агенти могу бити без модела или засновани на моделу.

Чак и када експлицитно не памтимо полису, у случају агената заснованих на вредности, она и даље постоји. У том случају, полиса је бирање акције која има максималну очекивану награду. Полиса је оно што нам је заправо битно. Ако знамо оптималну полису, знамо како да доносимо најбоље одлуке, што је и циљ учења са појачавањем. Међутим, често ће бити ефективније да полису не учимо директно, већ преко вредносне функције.

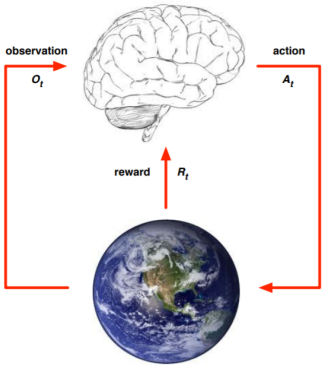
Два основна проблема у секвенцијалном доношењу одлука којима ћемо се бавити су **учење** и **планирање**. Битно је да разумемо разлику између ова два појма. При учењу, агент не зна како окружење фунционише, док је код планирања начин функционисања окружења "уграђен" у агентов систем доношења одлука.

Дилема истраживања и експлоатације је једно од кључних питања учења са појачавањем. Када да истражујемо а када да експлоатишемо стечено знање? Учење са појачавањем је учење методом покушаја и грешки. Да би учили морамо да истражујемо али истовремено желимо да минимизујемо успут изгубљену награду. Нпр. аутомобил који учи да вози, при скретању мора да покуша различита окретања волана да би научио које је правилно, али није исплативо покушати све могућности јер би на тај начин слупали јако велики број аутомобила. Још један пример је систем реклама на интернету. Страница углавном кориснику приказује рекламу за коју верује да ће највероватније бити кликнута, али понекад је корисно приказати нову рекламу јер се она кориснику може више свидети.

Још једна подела учења са појачавањем је на проблем предвиђања и проблем контроле. Предвиђање подразумева да предвидимо очекивану укупну награду за дату полису. Приликом контроле желимо да нађемо најбољу полису. Видећемо да је решавање проблема предвиђања битан корак за проблем контроле.



(a) Таксономија учења са појачавањем



(b) Интеракција агента са окружењем.

3 Марковљев процес одлучивања

MDP формално описује окружење, у случају када је оно потпуно прегледно. У овом поглављу ћемо дефинисати најбитније појмове, којима карактеришемо сваки MDP и над којима вршимо учење.

3.1 Марковљев ланац

Захваљујући Марковљевом својству можемо дефинисати вероватноћу прелаза из стања s у стање s' као:

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s] \tag{3.1}$$

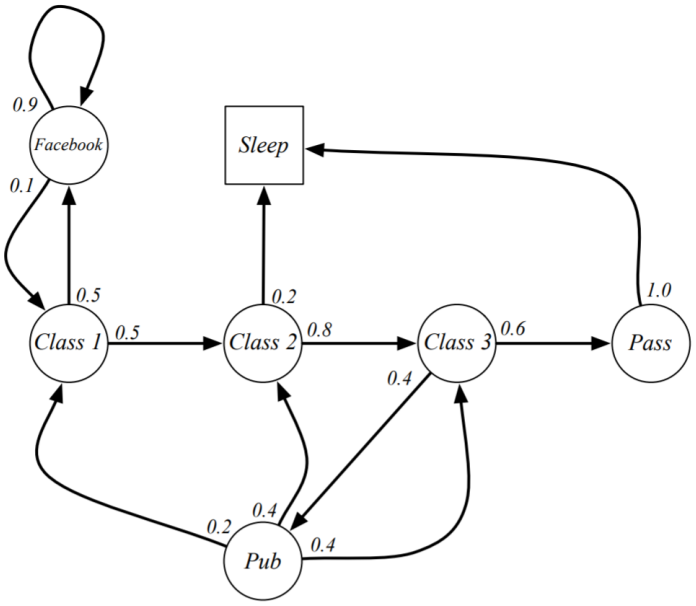
Све вероватоће можемо сместити у матрицу чиме добијамо матрицу вероватноће прелаза: ...

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \tag{3.2}$$

Дефиниција 3.1.1. Марковљев процес (или Марковљев ланац) (MP) је уређени пар $\langle \mathcal{S}, \mathcal{P} \rangle$, где:

- $\mathcal{S}$ је коначан скуп стања.
- $\mathcal{P}$ је матрица вероватноће прелаза

Марковљев процес је насумичан процес без меморије, тј. низ насумичних стања са Марковљевим својством.



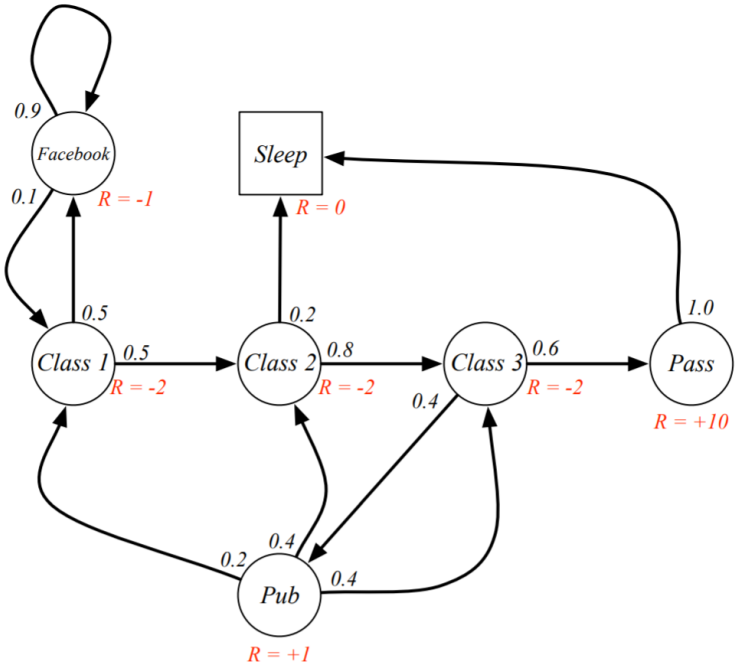
Слика 4: Пример студентског MP-а [1]. Сваку епизоду (дан) студент започиње у стању *Class 1*. Епизода се завршава када дође у стање *Sleep*. Бројеви на стрелицама представљају вероватноће преласка између стања. Једне од могућих епизода су *C1, C2, C3, Pass, Sleep* и *C1,FB,FB,C1,C2,Sleep*.

3.2 Марковљев наградни процес

Дефиниција 3.2.1. Марковљев наградни процес је уређена четворка $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$,где:

- $\mathcal{S}$ је коначан скуп стања.
- $\mathcal{P}$ је матрица вероватноће прелаза.
- $\mathcal{R}$ је наградна функција, $\mathcal{R}_s = \mathbb{E}[R_{t+1}|S_t = s]$.
- γ је фактор попуста, $\gamma \in [0, 1]$.

Марковљев наградни процес (MRP) је Марковљев ланац са наградама.



Слика 5: Студентски MRP. Награде се добијају по изласку из стања. Награда зависи само од стања из ког се излази, не и од изабране акције.

Дефиниција 3.2.2. Повраг G_t је укупна кумулативна награда са попустом, почевши од тренутка t.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.3}$$

Фактор попуста одређује колико су нам тренутно битне будуће награде. Што је ближи 0, више преферирамо тренутне награде, а што је ближи 1, небитније нам је када награде стижу.

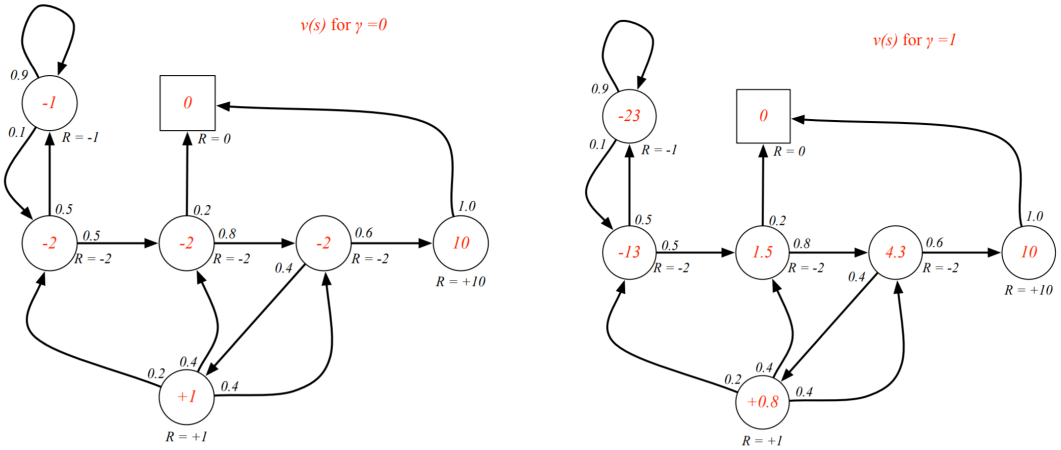
Зашто користимо фактор попуста?

- Избегавамо бесконачне поврате у случају цикличних или бесконачних MDP-ева. Проблем са бесконачним повратима је то што не можемо разликовати трајекторије. Нпр. агент неће моћи да разликује циклус који нам у сваком потезу доноси награду $+1000$ од оног који нам сваки потез доноси $+1$, јер је $\infty * 1000 = \infty * 1 = \infty$, а први циклус је очигледно бољи.
- На овај начин урачунавамо непрецизност нашег модела. Много је вероватније да ће награде у далекој будућности бити погрешно процењене, па оне тренутно за нас вреде мање.

Дефиниција 3.2.3. Вредносна функција стања за MRP је очекивани повратак почевши из стања s:

$$v(s) = \mathbb{E}[G_t|S_t = s] \tag{3.4}$$

Циљ при раду са Марковљевим наградним процесима је наћи ову функцију.



Слика 6: Вредносна функција стања за студентски MRP за два избора попуста γ.

3.2.1 Белманове једначине за MRP

Теорема 3.2.1. Вредносну функцију можемо раставити на два дела: тренутну награду R_{t+1} и вредност наредног поља са попустом $\gamma v(S_{t+1})$:

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \tag{3.5}$$

Доказ.

$$\begin{aligned} v(s) &= \mathbb{E}[G_t|S_t = s] && (3.6)\\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] && (3.7)\\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...)|S_t = s] && (3.8)\\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] && (3.9)\\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] && (3.10) \end{aligned}$$

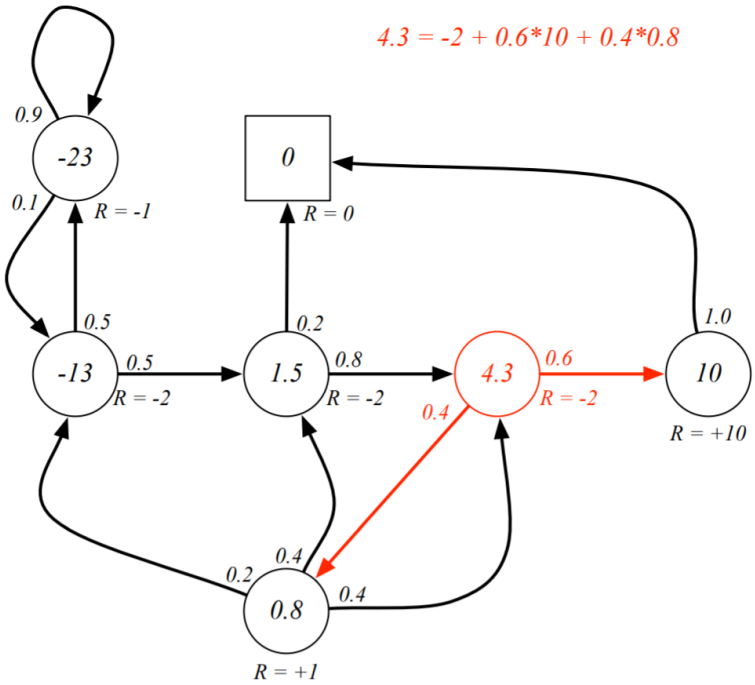
Белманову једначину је могуће концизно записати у матричном облику: $v = \mathcal{R} + \gamma\mathcal{P}$.

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & ... & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & ... & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} \tag{3.11}$$

Ово је линеарна једначна па је можемо директно решити:

$$\begin{aligned} v &= \mathcal{R} + \gamma\mathcal{P}v && (3.12)\\ (1 - \gamma\mathcal{P})v &= \mathcal{R} && (3.13)\\ v &= (1 - \gamma\mathcal{P})^{-1}\mathcal{R} && (3.14) \end{aligned}$$

У општем случају, временска сложеност налажења инверзне матрице је $O(n^3)$, ако имамо n стања. То је превелика сложеност за велике MDP-еве. Бавићемо се итеративним методама за израчунавање вредносне функције као што су: динамичко програмирање, Монте-Карло и Temporal-Difference учење.



Слика 7: Белманове једначине за студентски MRP.

3.3 Марковљев процес одлучивања

Дефиниција 3.3.1. Марковљев процес одлучивања је уређена петорка $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ је коначан скуп стања.
- $\mathcal{A}$ је коначан скуп акција.
- $\mathcal{P}$ је матрица вероватноће прелаза, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$.
- $\mathcal{R}$ је наградна функција, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$.
- γ је фактор попуста, $\gamma \in [0, 1]$.

Марковљев процес одлучивања је Марковљев наградни процес са доношењем одлука.

Дефиниција 3.3.2. Полиса (π) је дистрибуција акција по стањима,

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s] \tag{3.15}$$

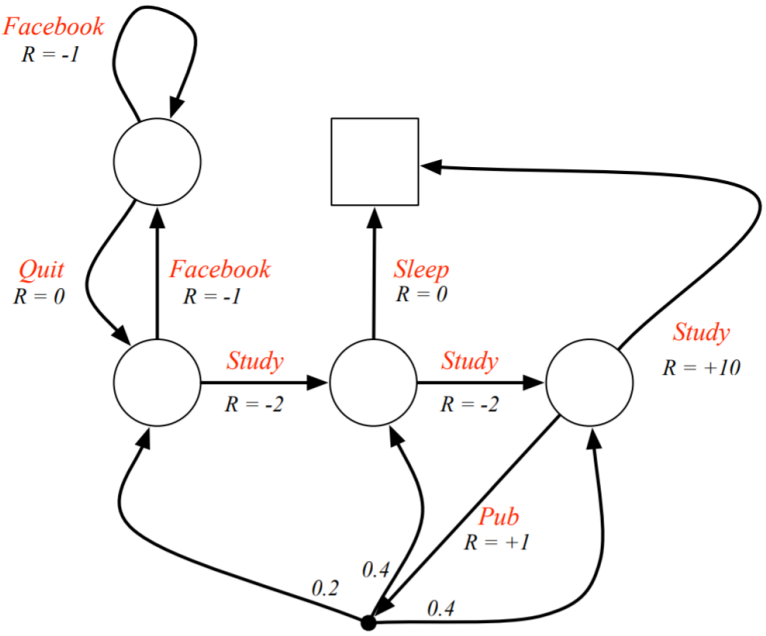
Полиса је стационарна (не мења се током времена) захваљујући Марковљевом својству.

За дати MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ и полису π:

- Низ стања $S_1, S_2, ...$ је Марковљев ланац $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- Низ стања и награда $S_1, R_1, S_2, ...$ је Марковљев наградни процес $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$,
- $\mathcal{P}^\pi$ i $\mathcal{R}^\pi$ су укупна вероватноћа по свим могућим акцијама да из стања s дођемо у стање s' и просечна награда по свим могућим акцијама,

$$\mathcal{P}^\pi_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{s,s'} \tag{3.16}$$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s \tag{3.17}$$



Слика 8: Студентски MDP. Сада награде зависе и од акције коју бирамо. Овог пута Pub није стање, већ пример стохастичне транзиције из стања $Class3$ у неко од наредних стања.

3.3.1 Вредносне функције

Дефиниција 3.3.3. Вредносна функција стања (вредност стања) $v_\pi(s)$ за MDP је очекивана кумулативна награда (повратак) почевши из стања s, и пратећи полису π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{3.18}$$

Дефиниција 3.3.4. Вредносна функција акција (вредност акције) $q_\pi(s, a)$ за MDP је очекивана кумулативна награда (повратак) ако почињемо из стања s, бирамо акцију a, и пратимо полису π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{3.19}$$

3.3.2 Белманове једначине очекивања

Поново можемо вредносну функцију стања раставити на тренутну награду и вредност наредног стања са попустом,

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{3.20}$$

Слично се раставља и вредносна функција акција,

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{3.21}$$

Можемо комбиновати ове две функције, чиме добијамо:

$$v_\pi(s) = \sum_{a\in A} \pi(a|s) q_\pi(s, a) \tag{3.22}$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \tag{3.23}$$

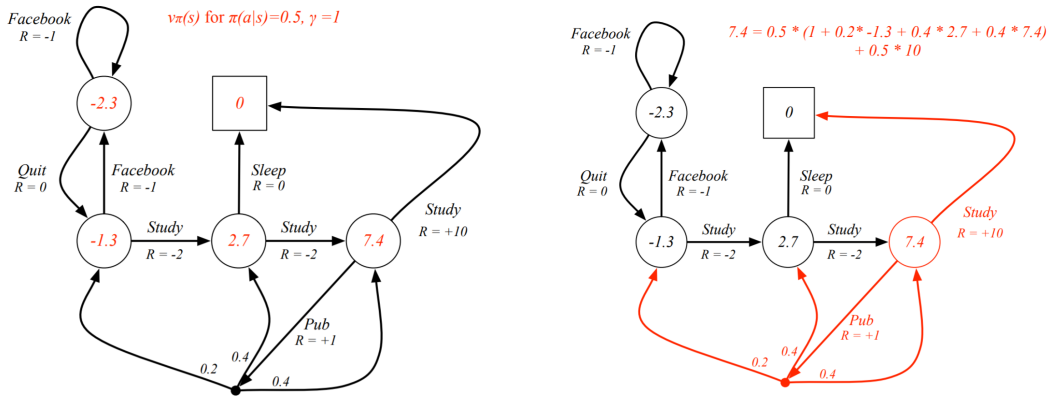
Ако повежемо претходне две једначине, добијамо рекурзивне формуле:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right) \tag{3.24}$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \tag{3.25}$$

Све што нам ове једначине у суштини говоре је да је вредносна функција у неком тренутку једнака збиру тренутне награде и наредне вредносне функције.

Претходне једначине нам говоре како да нађемо вредносне функције за дату полису, али не и како да нађемо најбољу полису.



Слика 9: Вредносне функције и Белманове једначине очекивања за студентски MDP.

Дефиниција 3.3.5. Оптимална вредносна функција стања $v_*(s)$ је највећа могућа вредност неког стања по свим могућим полисама

$$v_*(s) = \max_\pi v_\pi(s) \tag{3.26}$$

Дефиниција 3.3.6. Оптимална вредносна функција акције $q_*(s,a)$ је највећа могућа вредност неке акције по свим могућим полисама

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{3.27}$$

Да би пронашли оптималну полису довољно је да знамо оптималну вредност свих акција. Према томе, решавање MDP-а се може свести на проналажење $q_*(s,a)$.

3.3.3 Оптимална полиса

Дефиниција 3.3.7. За две дате полисе π и π' важи $\pi \geq \pi'$ ако важи $v_\pi(s) \geq v_{\pi'}(s), \forall s$.

Теорема 3.3.1. За сваки MDP:

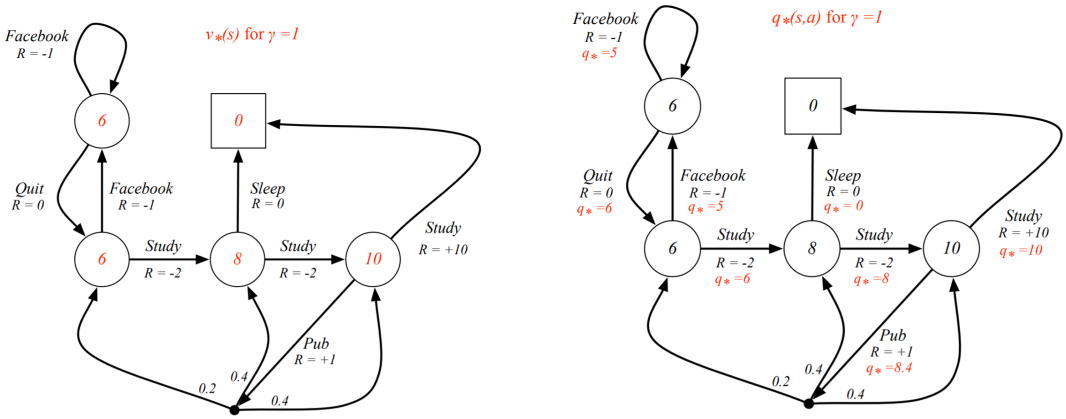
- Постоји бар једна оптимална полиса π_*, која је боља од или једнака свим осталим полисама,
  $\pi_* \geq \pi, \forall \pi$.
- Све оптималне полисе достижу оптималну вредносну функцију стања, $v_{\pi_*}(s) = v_*(s)$.
- Све оптималне полисе достижу оптималну вредносну функцију акција, $q_{\pi_*}(s,a) = q_*.(s,a)$

Доказ ове теореме следи на крају наредног поглавља.

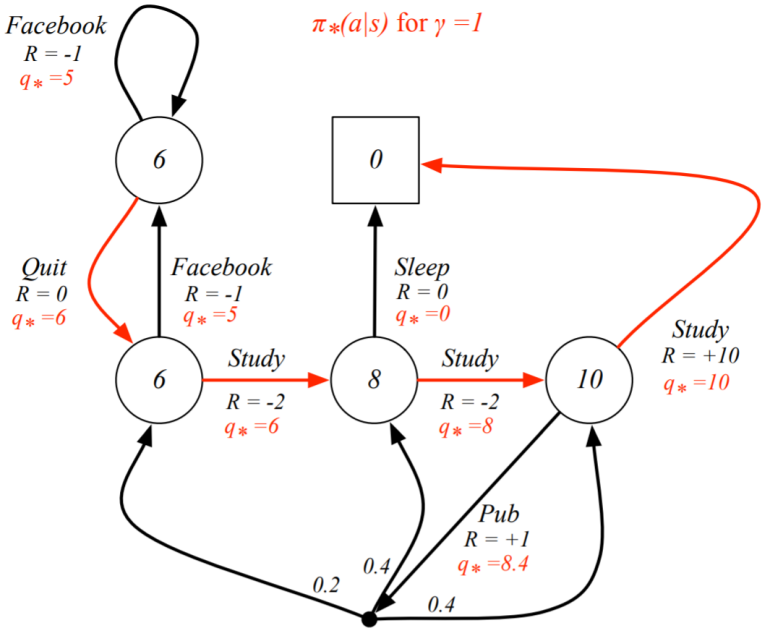
Када знамо $q_*(s,a)$, оптималну полису можемо наћи максимизацијом по $q_*(s,a)$,

$$\pi_*(a|s) = \begin{cases} 1, & a = \underset{a \in \mathcal{A}}{\text{argmax}} q_*(s,a) \\ 0, & \text{u suprotnom} \end{cases} \tag{3.28}$$

Дакле, за сваки MDP постоји детерминистичка оптимална полиса.



Слика 10: Оптималне вредносне функције стања и акције.



Слика 11: Оптимална полиса за студентски MDP.

3.3.4 Белманове једначине оптималности

Белманове једначине оптималности служе налажењу оптималне полисе. И до Белманових једначина оптималности долазимо раздвајањем вредносних функција тако што гледамо корак у будућност.

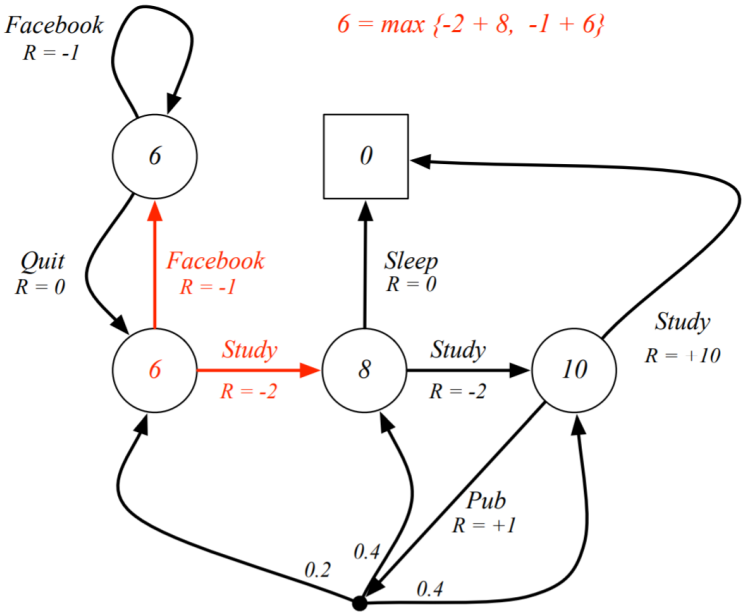
$$v_*(s,a) = \max_a q_*(s,a) \tag{3.29}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \tag{3.30}$$

$$v_*(s,a) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right) \tag{3.31}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a') \tag{3.32}$$

Белманове једначине оптималности нису линеарне, па их не можемо директно решити. Постоји мноштво итеративних решења којима ћемо се бавити: итерација вредности, итерација полисе, Q-учење.



Слика 12: Белманова једначина оптималности за студентски MDP.

Случај итеративне примене неке од једначина на стање или пар стање акција, називамо Белманов оператор (нпр. Белманов оператор оптималности).

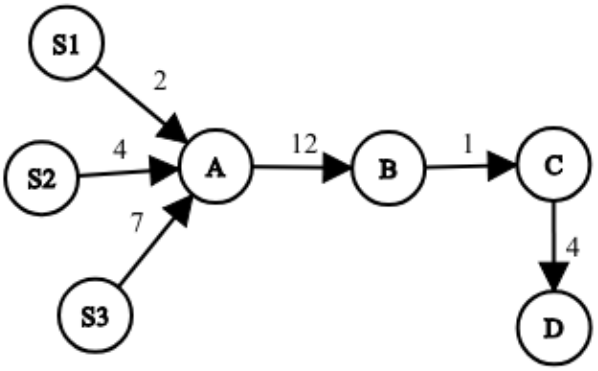
4 Планирање динамичким програмирањем

У претходном поглављу смо се упознали са основним појмовима везаним за Марковљеве процесе одлучивања. Сада ћемо видети како уз помоћ Белманових једначина можемо оценити неку полису, па потом доћи и до оптималне.

Подсетимо се да при планирању агент има увид у начин функционисања окружења, па се ради о MDP-у. Шта у нашем контексту значе речи **динамичко** и **програмирање**? Проблем је динамички ако има секвенцијалну или временску компоненту. Дакле, за све што се одвија у "корацима" кажемо да је динамичко. Програмирање у математичком смислу представља оптимизацију неког "програма", у нашем случају полисе. Заједно, ови појмови означавају методе оптимизације секвенцијалних проблема.

Динамичко програмирање се као метода састоји из три дела: разбијање проблема на мање подпроблеме, решавање тих подпроблема и комбиновање њихових решења.
Генерално за проблеме које на овај начин решавамо морају да важе два својства:

- **Оптимална структура** подразумева да проблем можемо поделити на делове чија искомбинована решења дају решење полазног проблема.
- **Подпроблеми који се преклапају**. У току решавања, више пута наилазимо на исти подпроблем. Решења памтимо, и следећи пут када наиђемо на исти проблем довољно је да прочитамо већ израчунато, чиме штедимо време.



Слика 13: Пример проблема са ова два својства. Циљ је наћи дужину пута од чворова S1,S2 и S3 до чвора D. Дужину пута од S1 до D можемо разложити на збир дужина од S1 до A и од A до D. Слично делимо путеве из S2 и S3. Дужину пута од A до D користимо 3 пута, приликом тражења решења за свако од почетних поља. Када би је сваки пут рачунали морали би да прођемо $4 + 4 + 4 = 12$ чворова (A, B, C, D сваки пут). Ако запамтимо дужину пута из чвора A, мораћемо да прођемо $4 + 1 + 1 = 6$ чворова. За већи граф ова разлика била би још већа.

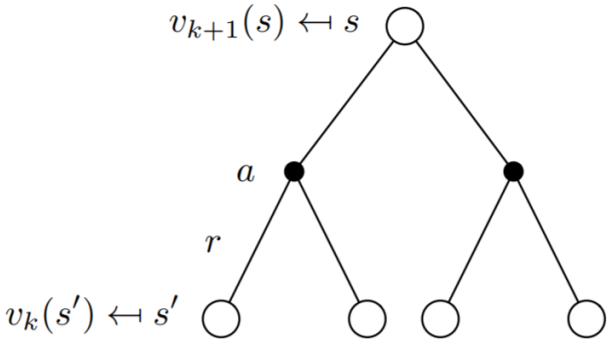
Марковљеви процеси одлучивања задовољавају оба својства. Белманове једначине дају начин да проблем поделимо, а вредносне функције чувају израчуната решења.
Динамичко програмирање се у MDP-у користи за:

- Предвиђање: за дати MDP и полису (што је исто што и MRP) израчунава вредносну функцију.
- Контролу: за дати MDP даје оптималну вредносну функцију и оптималну полису.

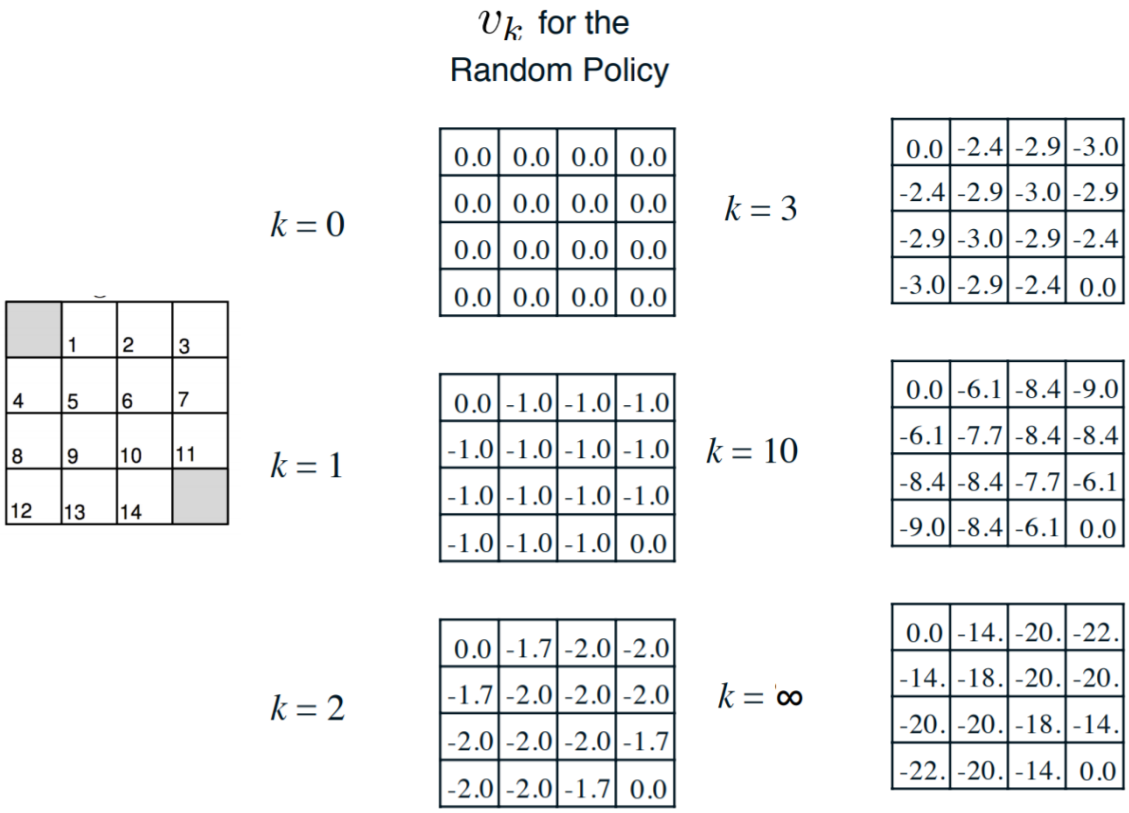
4.1 Оцена полисе

Оценити дату полису значи наћи њену вредносну функцију стања. Ово постижемо итеративном применом Белманове једначине очекивања, синхронисано, на сва стања одједном ($V_\pi^1 \to V_\pi^2 \to ... \to v_\pi$). Са V означавамо оцену вредности а са v праву вредност. У сваком кораку сва стања ажурирамо формулом:

$$V_\pi^{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi^k(s') \right) \tag{4.1}$$



Слика 14: Визуелизација Белманове једначине очекивања.



Слика 15: Пример итеративне оцене полисе [2]. Стања су поља матрице. У сваком кораку агент може прећи у једно од суседних поља матрице (горе, доле, лево или десно). За сваки протекли корак добија се награда -1, сива поља су терминална. Агент прати насумичну полису.

4.2 Итерација полисе

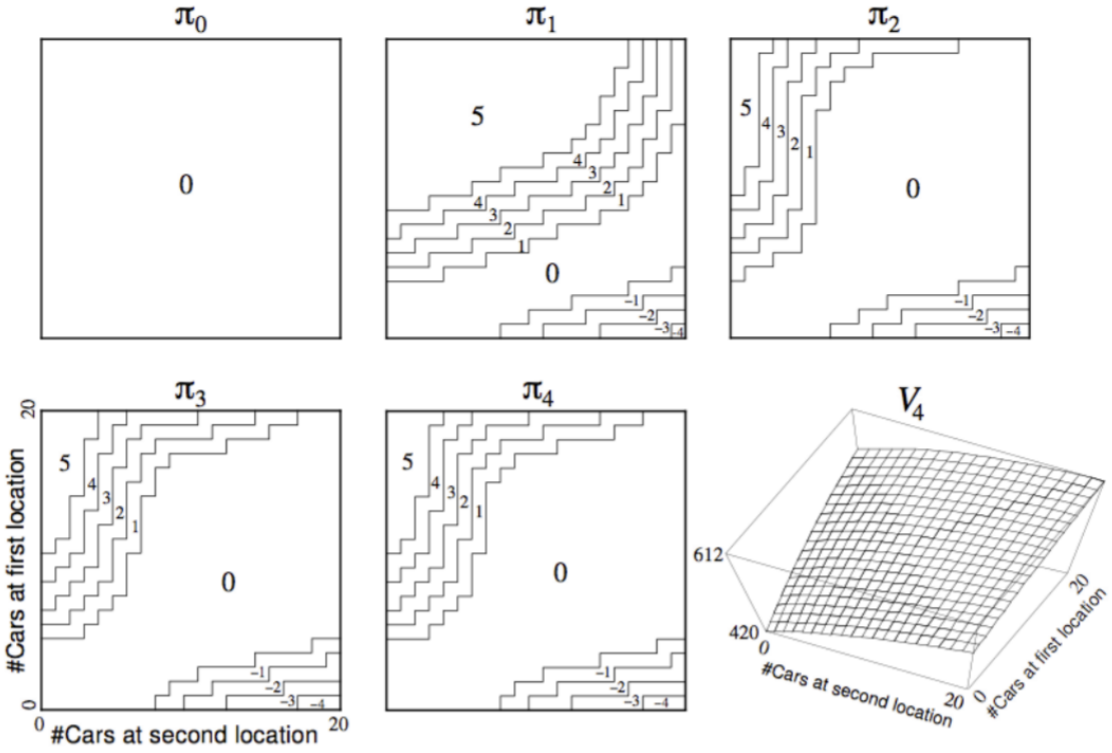
Итерацијом долазимо до оптималне полисе. Алгоритам се састоји из два дела: за дату полису π:

- Вршимо оцену полисе, чиме налазимо v_π,
- Унапређујемо полису тако што бирамо акције похлепно у односу на v_π :

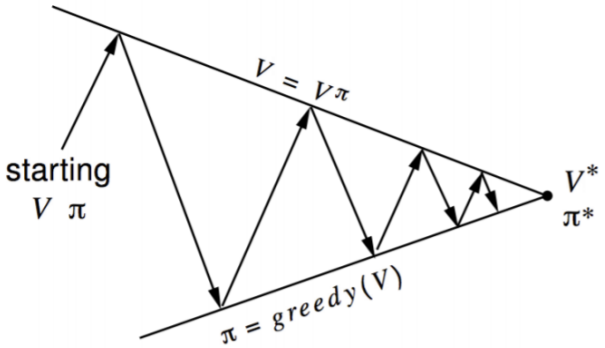
$$\pi' = greedy(v_\pi) \tag{4.2}$$

Похлепно значи да бирамо акцију која максимизује $\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$.

Овај поступак понављамо до конвергенције. У општем случају биће потребан велики број корака.



Слика 16: Итерација полисе. Ово је пример продавнице која изнајмљује аутомобиле [3] на две локације. На свакој од њих може бити највише 20 аутомобила. Стања су уређени парови (број возила на првој локацији, број возила на другој локацији) на крају дана. У току ноћи можемо пребацити до 5 аутомобила са једне локације на другу, и то су акције. Добијамо награду 10$ за сваки изнајмљен ауто. Број захтева и враћених аутомобила у току дана је насумичан, али знамо да у просеку на првој локацији имамо 3 захтева и 3 враћена возила, а на другој 4 захтева и 2 враћена возила. На сликама, свака тачка представља неко стање, а бројеви унутар ограничених површина представљају број аутомобила које треба да преместимо са прве на другу локацију. Алгоритам већ после 4 итерације долази до оптималне полисе. Видимо да ће чешће бити оптимално да пребацујемо аутомобиле са прве на другу локацију, што је и логично јер са друге одлази више аутомобила него што долази. Последња слика представља оптималну вредносну функцију сваког стања.



Слика 17: Визуелизација итерације полисе.

4.2.1 Доказ конвергенције

Лема 4.2.1. Нека нам је дата детерминистичка полиса π и почетно стање s. Ако један корак пратимо полису π', добијену као $\pi'(s) = \underset{a\in\mathcal{A}}{\text{argmax}} q_*(s,a)$, повећаћемо функцијску вредност почетног стања.

Доказ. Нека је нова вредност стања једнака је $q_\pi(s,\pi'(s))$. Тада важи:

$$q_\pi(s,\pi'(s)) = \max_{a\in\mathcal{A}} q_\pi(s,a) \geq q_\pi(s,\pi(s)) = v_\pi(s) \tag{4.3}$$

□

Теорема 4.2.1. Итерација полисе увек конвергира оптималној полиси.

Доказ. Нека је почетна полиса π, а π' полиса добијена као $\pi'(s) = \underset{a\in\mathcal{A}}{\text{argmax}} q_*(s,a)$. Покажимо прво да је $\pi' \geq \pi$. На основу леме 4.2.1 следи:

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s,\pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] && (4.4)\\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1},\pi'(S_{t+1}))|S_t = s] && (4.5)\\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2},\pi'(S_{t+2}))|S_t = s] && (4.6)\\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] = v_{\pi'}(s) && (4.7) \end{aligned}$$

Ако побољшање престане, то јест ако $v_{\pi'}(s) = v_\pi(s), \forall s$, важиће:

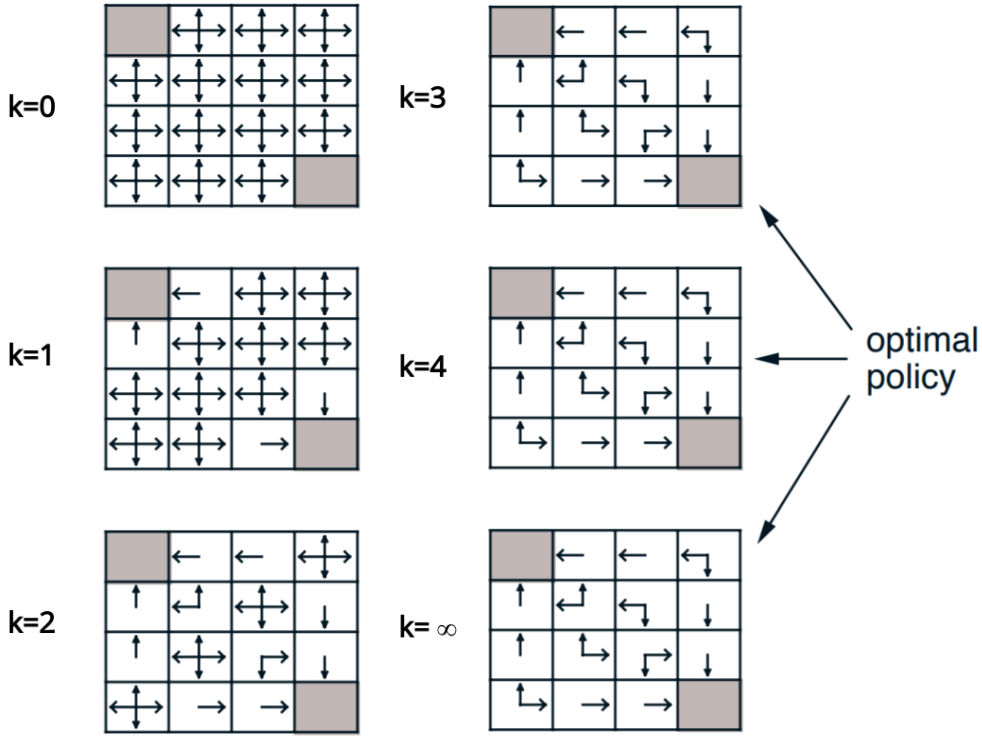
$$q_\pi(s,\pi'(s)) = \max_{a\in\mathcal{A}} q_\pi(s,a) = q_\pi(s,\pi(s)) = v_\pi(s) \tag{4.8}$$

Али тада је задовољена једна од Белманових једначина оптималности:

$$v_\pi(s) = \max_{a\in\mathcal{A}} q_\pi(s,a) \tag{4.9}$$

Одатле следи да је $v_\pi(s) = v_*(s), \forall s$, чиме је теорема доказана. □

Да ли је неопходно да сваки пут оцена полисе конвергира v_π? Можемо се зауставити и раније. Да би извршили корак побољшања полисе потребно је да за свако стање знамо акцију са највећом вредносном функцијом. Није увек неопходно знати баш тачне вредности акција да би знали која је најбоља. Могли би да се зауставимо када промене вредносних функција постану довољно мале, или једноставно након сваких k итерација.
Заправо уопште не морамо да итерирамо до вредносне функције која нам за свако поље даје најбољу акцију пратећи дату полису. Зашто бисмо чекали на сва стања, ако можемо одмах искористити стечено знање у неким од њих? Када је $k = 1$, тј. када радимо само један корак процене и одмах потом похлепно побољшавамо полису долазимо до алгоритма **итерације вредносне функције**.



Слика 18: Наставак примера са матрицом из секције 4.1. Ово су полисе које би добили похлепним побољшањем после сваке од итерација. Видимо да већ у трећој итерацији знамо која је најбоља акција у сваком од стања.

4.3 Итерација вредносне функције

Као што смо рекли, итерација вредносне функције је специјалан случај итерације полисе. При сваком кораку вршимо операцију:

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right) \tag{4.10}$$

Сада π ажурирамо после сваке примене ове операције. То значи да ће, за разлику од итерације полисе, овде π увек бити похлепна полиса. Због тога њу није потребно експлицитно памтити. Замењујемо је једним оператором максимизације:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right) \tag{4.11}$$

Сада немамо експлицитну полису, радимо директно на вредносној фукцији, па отуд и назив итерација вредносне фукције. Приметимо да је претходна једначина заправо Белманова једначина оптималности. То значи да је Белманова једначина очекивања, у случају када је полиса похлепна у односу на вредносне функције, ништа друго него једначина оптималности.

Сагледајмо овај алгоритам и на други начин, почевши из Белманових једначина оптималности, без разматрања било какве полисе. Знамо да смо нашли оптималну вредносну функцију када су оне задовољене. Осврнимо се на једно својство оптималних полиса које нам може помоћи да додатно мотивишемо итеративну примену Белмановог оператора оптималности.

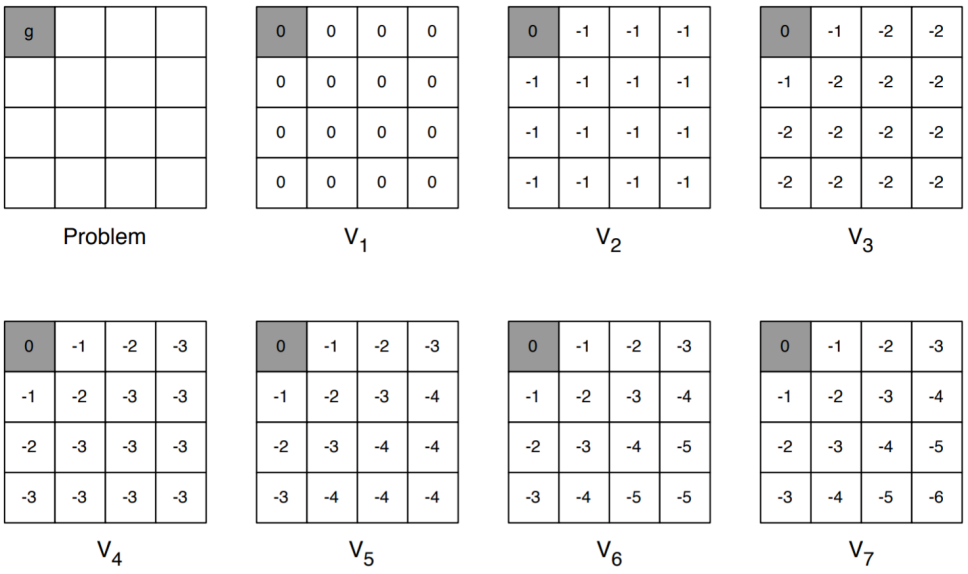
Теорема 4.3.1. (Принцип оптималности): Полиса $\pi(a|s)$ достиже оптималну вредност из стања s, $v_\pi(s) = v_*(s)$, ако и само ако за свако стање s' доступно из s, π достиже оптималну вредност за то стање, $v_\pi(s') = v_*(s)$.

Дакле, ако знамо решење подпроблема $v_*(s')$, вредност $v_*(s)$ можемо наћи гледањем један корак у будућност:

$$v_*(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right) \tag{4.12}$$

Намеће се идеја о итеративној примени Белманових оператора оптималности. Али да ли она заиста конвергира? Одговор на ово питање је потврдан и у наредном одељку ћемо видети како да то и докажемо.

Како можемо интуитивно схватити начин функционисања овог алгоритма? Замислимо окружење у којем постоји једно терминално стање. За улазак у то стање добија се велика позитивна награда (+100), а за остала мала негативна (−1). Иницијализујемо вредносну функцију за свако стање на 0. После првог корака ће вредност сваког стања бити вредност тренутне награде, дакле +100 за терминално и −1 за остала. При другом кораку, стања суседна терминалном ће узети у обзир велику награду која се из њега добија, јер се она сада налази у његовој вредносној функцији. Зато ће после другог корака вредносна функција и ових стања бити велика $(-1 + \gamma V(terminal))$. У наредном кораку, велика награда ће доћи до стања удаљених два корака од терминалног, итд. Награде се на овај начин пропагирају кроз цео простор. Тек када све битне награде дођу до неког стања алгоритам ће знати која је најбоља акција у њему.



Слика 19: Пример итерације вредносне функције. Циљ је доћи у горње лево (терминално) поље. За сваку транзицију добија се награда −1.

4.4 Доказ конвергенције оцене полисе и итерације вредносне функције

Како знамо да итерација вредносне функције конвергира v_*, или да итеративна оцена полисе конвергира v_π, а самим тим и да итерација полисе конвергира v_*? Да ли су ова решења јединствена или постоје локални максимуми? Колико брзо конвергирају наши алгоритми? Одговор на ова питања даје нам теорема о контракцијама.

Размотримо векторски простор $\mathcal{V}$ свих могућих вредносних функција стања. Овај простор има $|\mathcal{S}|$ димензија. Свака тачка у простору представља једну вредносну функцију v. Показаћемо да применом Белманових једначина приближавамо ове тачке захваљујући чему оне конвергирају јединственом решењу.
Меримо удаљеност између тачака u и v преко ∞-норме, која представља максимални елемент

вектора:

$$||u - v||_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)| \tag{4.13}$$

Означимо Белманов оператор очекивања са T^π,

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v \tag{4.14}$$

Контракције су функције на метричком простору $(\mathcal{V}, d)$ за које важи $d(f(x), f(y)) \leq kd(x, y)$, где је $\mathcal{V}$ векторски простор а d функција која мери удаљеност.
Оператор T^π је γ контракција, тј. приближава тачке бар γ пута,

$$\begin{aligned} ||T^\pi(u) - T^\pi(v)||_\infty &= ||(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi u) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi v)||_\infty && (4.15)\\ &= ||\gamma \mathcal{P}^\pi (u - v)||_\infty && (4.16)\\ &\leq ||\gamma \mathcal{P}^\pi ||u - v||_\infty ||_\infty && (4.17)\\ &\leq \gamma ||u - v||_\infty && (4.18)\\ & && (4.19) \end{aligned}$$

Теорема 4.4.1. За сваки метрички простор $\mathcal{V}$ који је затворен под оператором $T(v)$, где је T γ-контракција, важи: T конвергира јединственој фиксној тачки брзином γ

На основу претходне теореме, Белманов оператор очекивања има фиксну тачку, а знамо да v_π задовољава ту Белманову једначину, што значи да је баш то фиксна тачка. Овим смо доказали да итеративна оцена полисе конвергира v_π, чиме смо допунили претходни доказ да итерација полисе конвергира v_*.
И Белманову једначину оптималности можемо посматрати као оператор над простором вредносних функција. На сличан начин се доказује да тај оператор представља контракцију чија је фиксна тачка v_*

4.5 Преглед обрађених алгоритама

Проблем	**Белманова једначина**	**Алгоритам**
Предвиђање	Белманова једначина очекивања	Итеративна оцена полисе
Контрола	Белманова једначина очекивања + похлепно побољшање полисе	Итерација полисе
Контрола	Белманова једначина оптималности	Итерација вредносне функције

За алгоритме које смо у овом поглављу обрадили довољна је била вредносна функција стања. Временска сложеност итерације вредносне функције је $O(mn^2)$ по итерацији, за n стања и m акција (из сваког стања разматрамо: сваку акцију и сва стања у која том акцијом можемо доћи). Када би у овим алгоритмима користили вредносну функцију акција, сложеност би била $O(m^2n^2)$ по итерацији. Видећемо да је у случају контроле без модела, и поред лошије сложености, неопходно користити $q(s, a)$.

5 Предвиђање без модела

У прошлом поглављу научили смо како се решавају Марковљеви процеси одлучивања. Користили смо чињеницу да имамо увид у динамику MDP-а (вероватноће прелаза у одређена стања при одређеним акцијама). Ово није случај у већини реалних проблема. За сва окружења и даље постоји MDP на основу којег се мења само окружење, али га ми или не можемо сазнати, или је једноставно превише компликован. Замислимо робота који учи да шутира лопту. Стање MDP-а у том случају је распоред свих молекула на фудбалском терену, што очигледно није погодна репрезентација. Сада ћемо размотрити алгоритме који врше оцену полисе када динамика окружења није позната.

5.1 Монте Карло (MC) учење

Монте Карло методе користе врло једноставну идеју: вредносна функција је једнака очекиваној вредности повратка. Циљ нам је да научимо v_π из епизода виђених пратећи полису π:

$$S_1, A_1, R_2, ..., S_k \sim \pi \tag{5.1}$$

Подсетимо се да је повратак једнак укупној награди са попустом:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T \tag{5.2}$$

А да је вредносна функција једнака очекиваном повратку:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \tag{5.3}$$

Монте Карло оцена полисе користи експериментално добијену средњу вредност уместо очекиване вредности.
Да би оценили вредност стања s, за свако појављивање тог стања у свакој епизоди:

- Повећавамо бројач посета: $N(s) = N(s) + 1$
- Повећавамо укупни повратак: $S(s) = S(s) + G_t$

Вредост стања се рачуна као $V(s) = S(s)/N(s)$. Према закону великих бројева $V(s) \to v_\pi(s)$ када $N(s) \to \infty$.

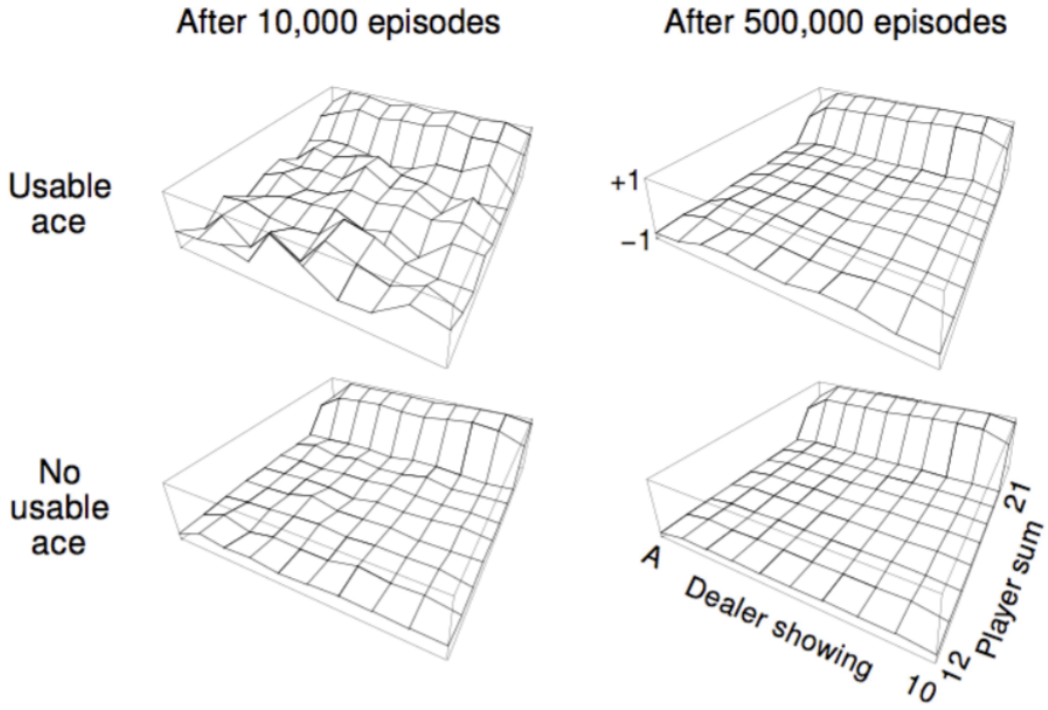
Размотримо Монте Карло оцену на примеру поједностављеног Блек Џека [4]. Свако стање је одређено са три броја: сума агентових карата (12-21), карта коју дилер показује (ас-10) и да ли агент има аса (да-не).
Могуће акције су stick (агент више не добија карте, епизода се завршава) и twist (агент добија још једну карту).
Награде за stick су:

- $+1$ ако је сума агентових карата $>$ сума дилерових карата
- 0 ако су једнаке
- -1 ако је мања

Награде за twist су:

- -1 ако сума агентових карата > 21 (и епизода се завршава)
- 0 у супротном

Агент аутоматски бира twist ако му је сума карата мања од 12 јер тиме сигурно неће прећи 21.
Агент прави једноставну полису: stick ако је сума карата ≥ 20, иначе twist.
После 10 и 500 хиљада епизода, добијамо следеће вредносне функције:



Слика 20: Вредносне функције у примеру Блек Џека.

Није неопходно чувати и N и S, можемо итеративно рачунати средњу вредност:

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j \tag{5.4}$$

$$= \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) \tag{5.5}$$

$$= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \tag{5.6}$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) \tag{5.7}$$

Разлику $x_k - \mu_{k-1}$ можемо схватити као грешку процене а $\frac{1}{k}$ као величину корака којим се крећемо у смеру x_k. За $k = 1$, било би $\mu_k = x_k$. Све итеративне оцене које се користе имају овакав облик: нова процена = стара процена + величина корака $*$ грешка.

За проблеме који нису стационарни, корисно је увести могућност заборављања искуства. Ово можемо постићи фиксном величином корака α ($\alpha \in [0, 1]$):

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t)) \tag{5.8}$$

На тај начин, удео вредности у суми опада експоненцијално, што се лако види ако претходну формулу презапишемо:

$$V(S_t) = (1-\alpha)V(S_t) + \alpha G_t \tag{5.9}$$

$$= \alpha G_t + (1-\alpha)(\alpha G_{t-1} + \alpha(1-\alpha)G_{t-2} + \cdots + \alpha(1-\alpha)^{t-2}G_1) \tag{5.10}$$

$$= \alpha G_t + \alpha(1-\alpha)G_{t-1} + \alpha(1-\alpha)^2 G_{t-2} + \cdots + \alpha(1-\alpha)^{t-1}G_1 \tag{5.11}$$

Заборављање искуства ће бити битно при тражењу оптималне полисе, зато што ћемо тада мењати полису а самим тим ће се мењати и праве вредности вредносне функције.

МС методе уче директно из виђених епизода, али морамо да сачекамо крај епизоде да би ажурирали вредносне функције, што доноси низ ограничења (нпр. не може се применити на случај MDP-ева у којима епизоде јако дуго трају или се никада не завршавају).

5.2 Temporal-Difference (TD) учење

Слично Монте Карло учењу, TD методе уче директно из виђених епизода и не користе модел. Међутим, ове методе уче из непотпуних епизода, поступком који се назива бутстрепинг. Бутстрепинг значи да тренутну процену вршимо на основу неке друге процене. У МС учењу смо морали да сачекамо да се епизода заврши,а у најједноставнијој верзији TD учења ћемо оцену померати у смеру збира тренутне награде и процене вредносне функције наредног стања.

Формула за итеративно ажурирање вредносне функције:

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{5.12}$$

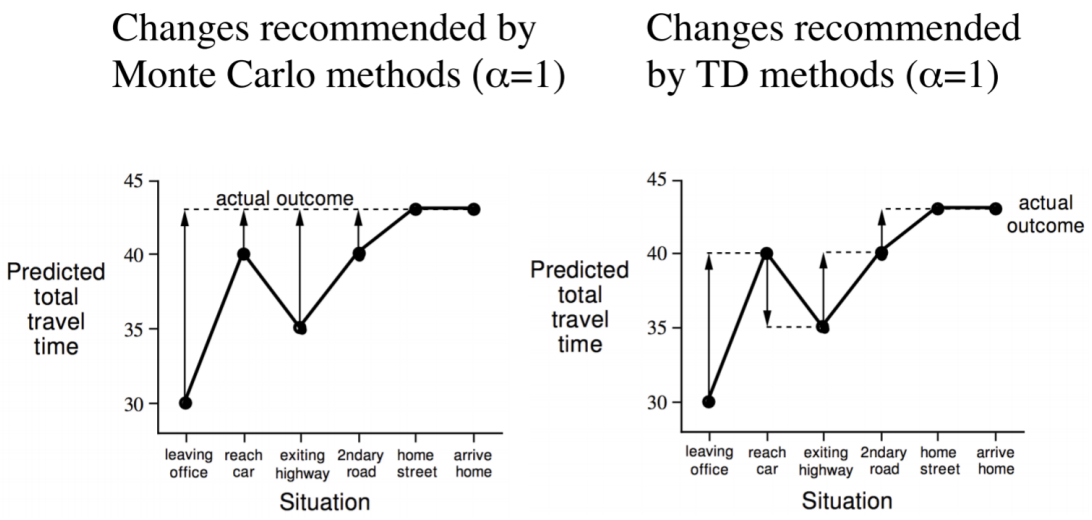
$R_{t+1} + \gamma V(S + t + 1)$ се назива TD мета. $\delta_t = R_{t+1} + \gamma V(S + t + 1) - V(S_t)$ је TD грешка.

5.3 Разлике TD и МС учења

Замислите самовозећи аутомобил којем изненада на пут излети друго возило. Аутомобил није на време схватио да треба да закочи, али неким чудом возила у пуној брзини су се мимоишла и није дошло до судара. Ако користимо TD, велика негативна награда коју аутомобил очекује када се приближи возилу ће се пропагирати у претходна стања и научиће да је требао почети кочење раније. Међутим, ако користимо МС, аутомобил ће видети само позитивну крајњу награду и неће схватити да је био у лошој ситуацији. У овом примеру TD очигледно даје боље резултате.

Размотримо још један пример. Колима се враћамо кући из канцеларије и меримо време путовања.

Стање	**Протекло време**	**Процењено преостало време**	**Процењен укупно време**
излазимо из канцеларије	0	30	30
стижемо до кола, пада киша	5	35	40
на аутопуту нема гужве	20	15	35
испред нас је камион	30	10	40
улазимо у своју улицу	40	3	43
стигли смо кући	43	0	43



Слика 21: Разлике у променама вредносних функција за TD и MC [4].

5.3.1 Компромис између пристрасности и дисперзије

Нека је θ параметар чију вредност желимо да оценимо, а T_n наша оцена тог параметра. Кажемо да је оцена непристрасна ако важи $\mathbb{E}[T_n] = \theta$.

Дисперзија нам говори колико је оцена "раширена", тј. колико је велик опсег могућих вредности те оцене. Процењена вредносна функција стања представља случајну променљиву, јер ће различита покретања алгоритма наилазити на различито искуство, па ће се и оцена вредносних функција разликовати. Када кажемо дисперзија у контексту алгоритма учења са појачавањем, мислимо на дисперзију вредносне функције одређеног стања. Дакле, дисперзија нам даје информацију о томе колико је наша процена осетљива на различито искуство.

Дисперзија и пристрасност су извори грешке. Они одређују колико ће процена вредносне функције успети да се приближи правој вредности. Углавном су повезани тако да смањење једног доводи до повећања другог.

Повратак $G_t = R_{t+1} + \gamma R_{t+2} + ...\gamma^{T-1} R_T$ је непристрасна оцена $v_\pi(S_t)$. TD мета $R_{t+1} + \gamma V(S_{t+1})$ јесте пристрасна, јер користи вредност $V(S_{t+1})$, која је наша процена и у општем случају не мора бити тачна.

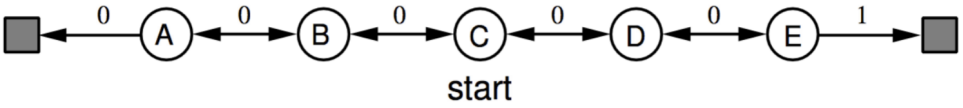
Са друге стране, TD мета има мању дисперзију од повратака. Она зависи само од једне случајне променљиве, R_{t+1}. $V(S_{t+1})$ је константа па има дисперзију 0. Дисперзија повратака је збир дисперзија великог броја променљивих.

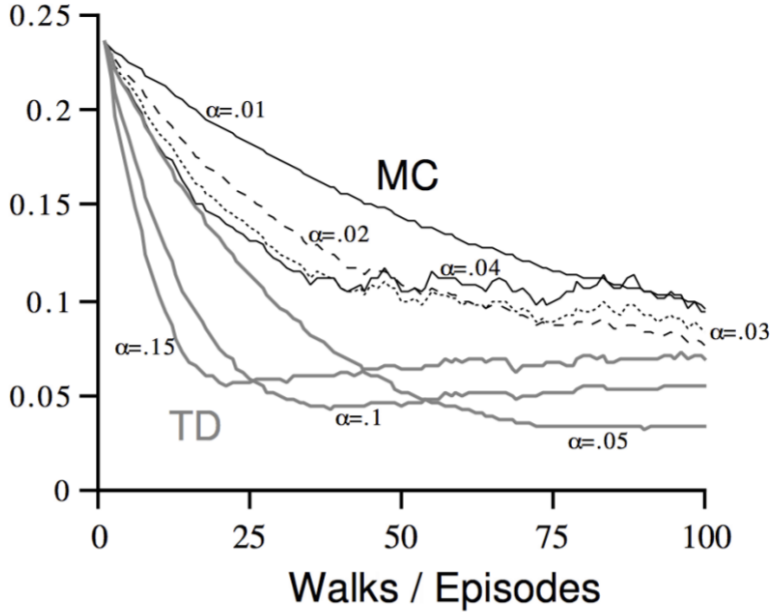
Због дисперзије ће алгоритам осциловати око правог решења. Величина тих осцилација зависиће и од величине корака α.

Такође, зато што врши бутстрепинг, TD је много осетљивији на почетне вредности.

У пракси се показује да су TD методе углавном ефикасније.

Пример насумичног хода. Сива стања су терминална. На графику је приказана зависност просечне грешке по свим стањима од броја епизода које је агент прошао. Дати су графици за различите α. Видимо да TD брже тежи решењу и има мању минималну грешку. То значи да је дисперзија MC већи извор грешке него пристрасност.





5.3.2 Разлике у случају ограниченог искуства

Размотримо прво један пример. Постоје два стања, A и B, немамо попуст, дато нам је 8 епизода:

$$A, 0, B, 0$$

$$B, 1$$

$$B, 1$$

$$B, 1$$

$$B, 1$$

$$B, 1$$

$$B, 1$$

$$B, 0$$

Које су вредности A и B?

MC конвергира решењу које минимизује средње квадратно одступање решења од података.

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} (G_t^k - V(s_t^k)) \tag{5.13}$$

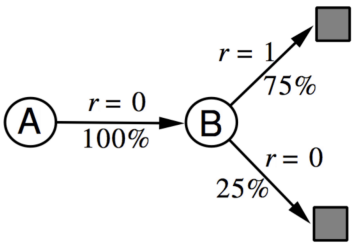
k је број епизоде, а t корак у епизоди. У AB примеру, $V(A) = 0$.

TD конвергира решењу највероватнијег Марковљевог модела, тј. $\langle \mathcal{S}, \mathcal{A}, \overline{\mathcal{P}}, \overline{\mathcal{R}}, \gamma \rangle$ које најбоље одговара виђеним подацима:

$$\overline{\mathcal{P}}^{a}_{s,s'} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} I(s_t^k, a_t^k, s_{t+1}^k = s, a, s') \tag{5.14}$$

$$\overline{\mathcal{R}}^{a}_{s,s'} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} I(s_t^k, a_t^k = s, a) r_t^k \tag{5.15}$$

I је индикатор догађаја. Има вредност 1 ако је једнакост тачна, а 0 у супротном.
У AB примеру, $V(A) = 0.75$.



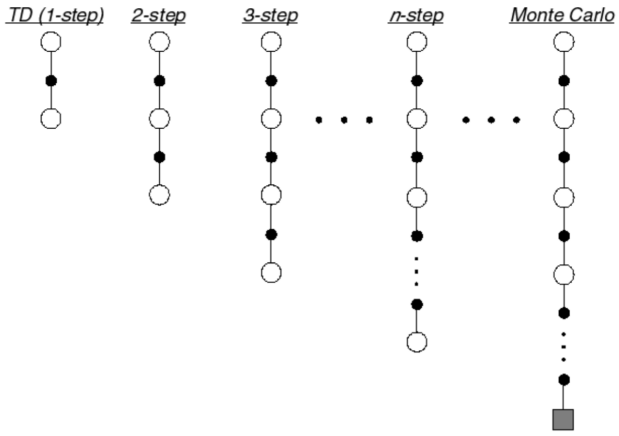
Слика 22: Највероватнији Марковљев модел.

Ова разлика је последица чињенице да TD, за разлику од MC-а, искоришћава Марковљево својство. Подсетимо се Марковљевог својства: будућност зависи само од тренутног стања, што значи да очекивани повратак стања не зависи од начина на који смо у њега дошли. Захваљујући томе TD је углавном ефикаснији у Марковљевим окружењима.

5.4 TD(λ) учење

5.4.1 Повратак у n корака

Зашто не бисмо дозволили да TD мета узме у обзир n корака пре него што бутстрепује? На овај начин уводимо генералнију верзију TD учења, TD(λ), која нам даје алгоритме између TD и MC метода.



Слика 23: Повратак у n корака.

Размотримо следеће примере поврата у n корака, за n=1,2,∞:

$$n = 1 \quad (TD) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \tag{5.16}$$

$$n = 2 \qquad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \tag{5.17}$$

$$\vdots \qquad \vdots \tag{5.18}$$

$$n = \infty \ (MC) \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T \tag{5.19}$$

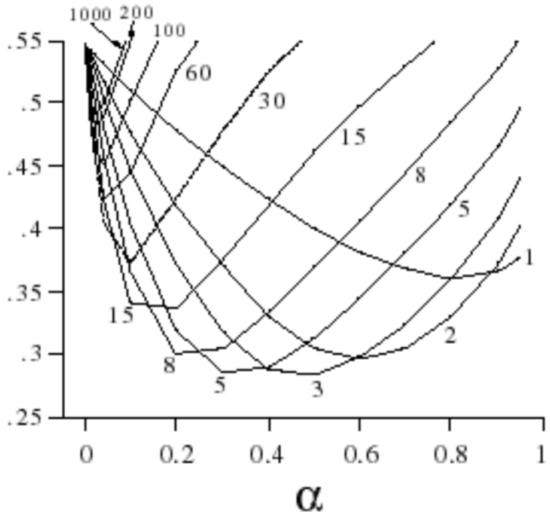
Дефинишимо поврат у n корака:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{5.20}$$

Сада уместо TD мете користимо нови поврат:

$$V(S_t) = V(S_t) + \alpha(G_t^{(n)} - V(S_t)) \tag{5.21}$$

Проблем са повратом у n корака је избор одговарајућег n. Идеално n ће се значајно мењати од проблема до проблема. Циљ машинског учења је да нађе алгоритме који могу без промене да раде на што већем броју проблема. Желимо да избегнемо потребу за одређивањем најбољег n.



Пример већег насумичног хода. На графику се види зависност просечне грешке по свим стањима од различитих вредности α при коришћењу поврата у n корака. Различите криве се односе на различите вредности n.

5.4.2 Просек за различите n

Најједноставнија идеја је узети просечну вредност поврата у n корака за више различитих n. Поставља се питање да ли можемо на ефикасан начин искомбиновати поврате за све могуће вредности n?

5.4.3 λ поврат

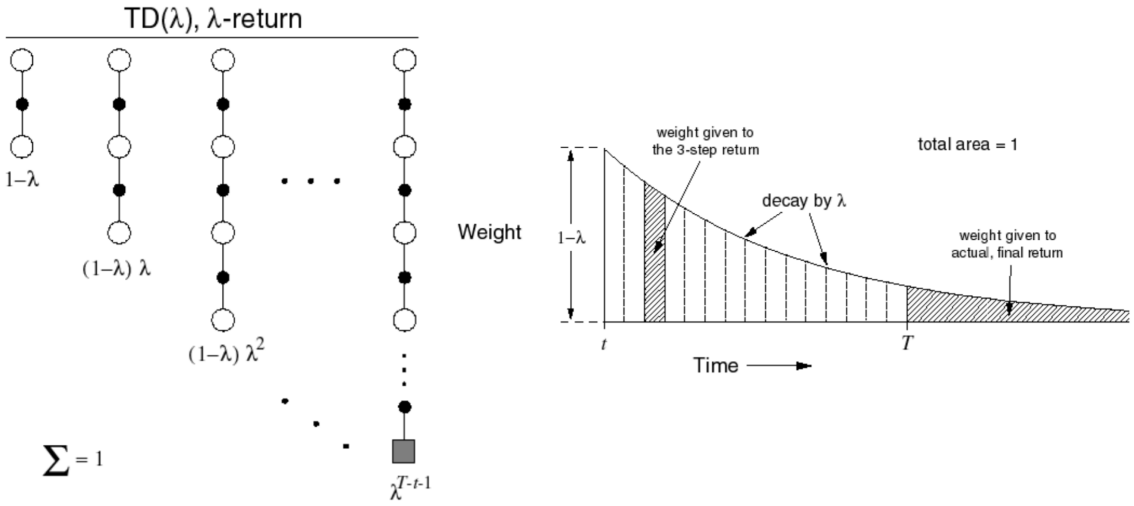
Идеја λ поврата је да геометријски отежинимо све поврате. Сума тежина мора бити једнака 1. Ово постижемо увођењем параметра $\lambda \in [0, 1]$. Тежина n-тог поврата једнака је $(1 - \lambda)\lambda^{n-1}$:

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{5.22}$$

Сада TD(λ) узима облик:

$$V(S_t) = V(S_t) + \alpha(G_t^{\lambda} - V(S_t)) \tag{5.23}$$

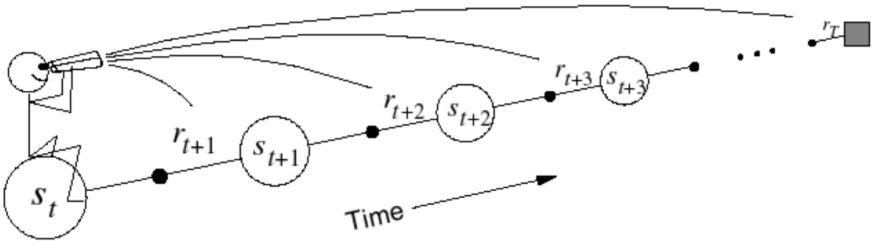
Специјални случајеви: за $\lambda = 0$ добијамо TD, а за $\lambda = 1$ MC учење.



Слика 24: λ поврати.

5.4.4 TD(λ) са погледом унапред

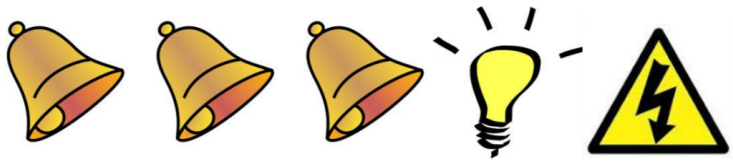
У досадашњем приступу TD(λ) алгоритам је гледао "напред у будућност"да би израчунао λ повратак. Награде које су нам потребне за ажурирање неког стања добијамо тек када то стање напустимо. У пракси ово можемо извести тако што сачекамо да се епизода заврши и потом посећујемо стања уназад. Код Монте Карло метода смо већ видели да није повољно чекати завршетак епизоде. Ово можемо заобићи памћењем такозваних **трагова одговорности**.



Слика 25: Поглед унапред.

5.4.5 Трагови одговорности

Потребан нам је начин да после сваког корака агента у окружењу ажурирамо претходна стања, тако да укупно ажурирање сваког стања остане исто као при погледу унапред.



Слика 26: Шта је узрок електрошока, звоно или лампица?

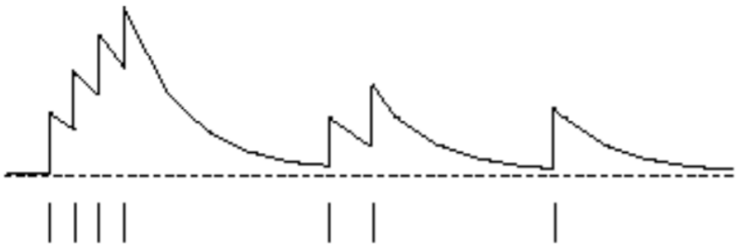
Постоје два основна начина на које можемо рачунати колико је сваки догађај допринео исходу: на основу броја појављивања догађаја (фреквенције) и на основу тога колико се скоро догађај десио.

Трагови одговорности комбинују ове две идеје:

$$E_0(s) = 0 \tag{5.24}$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + I(S_t = s) \tag{5.25}$$

Где је $E_t(s)$ траг одговорности стања s у тренутку t. Сваки пут када наиђемо на неко стање s, његов траг одговорности ће се увећати за 1 (I је индикатор). Трагови одговорности свих стања у којима нисмо били у тренутку t експоненцијално опадају брзином $\gamma\lambda$. Множимо са $\gamma\lambda$ јер ће на тај начин укупно ажурирање бити исто као при погледу унапред са та два параметра.



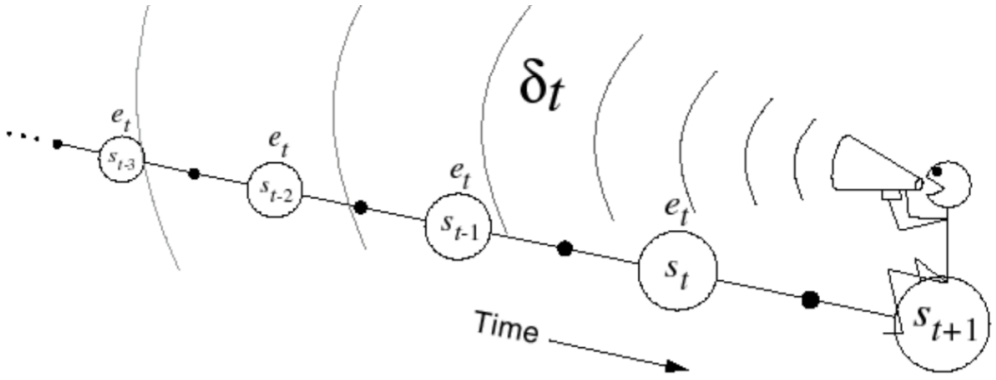
Слика 27: Трагови одговорности

5.4.6 TD(λ) са погледом уназад

За свако стање чувамо траг одговорности и вредносне функције ажурирамо пропорционало TD грешци δ_t и трагу одговорности $E_t(s)$:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{5.26}$$

$$V(s) = V(s) + \alpha\delta_t E_t(s) \tag{5.27}$$



Слика 28: Поглед уназад.

У случају λ=0, ажурирамо само тренутно стање:

$$E_t(s) = I(S_t = s) \tag{5.28}$$

$$V(s) = V(s) + \alpha\delta_t E_t(s) \tag{5.29}$$

Што је еквивалентно са TD(0). Слично, за λ=1, алгоритам је еквивалентан MC.

Теорема 5.4.1. Сума свих промена вредности неког стања је једнака за TD(λ) са погледом у напред и погледом у назад:

$$\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \sum_{t=1}^{T} \alpha(G_t^\lambda - V(S_t))I(S_t = s) \tag{5.30}$$

Доказ. Претпоставимо да је у току епизоде стање S_t посећено једном у тренутку t. Тада важи:

$$\begin{aligned}
G_t^\lambda - V(S_t) = \quad & -V(S_t) \quad + \quad (1-\lambda)\lambda^0(R_{t+1} + \gamma V(S_{t+1})) && (5.31)\\
& + \quad (1-\lambda)\lambda^0(R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})) && (5.32)\\
& + \quad (1-\lambda)\lambda^0(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \gamma^3 V(S_{t+3})) && (5.33)\\
& + \quad ... && (5.34)\\
= \quad & -V(S_t) \quad + \quad (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})) && (5.35)\\
& + \quad (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})) && (5.36)\\
& + \quad (\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - \gamma\lambda V(S_{t+3})) && (5.37)\\
& + \quad ... && (5.38)\\
= \quad & \quad (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) && (5.39)\\
& + \quad (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1})) && (5.40)\\
& + \quad (\gamma\lambda)^2(R_{t+3} + \gamma V(S_{t+3}) - V(S_{t+2})) && (5.41)\\
& + \quad ... && (5.42)\\
= \quad & \quad \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + ... && (5.43)
\end{aligned}$$

Пошто је стање посећено само једном:

$$\sum_{t=1}^{T} \alpha\delta_t E_t(s) = \alpha \sum_{t=k}^{T} (\gamma\lambda)^{t-k}\delta_t = \alpha(G_k^\lambda - V(S_k)) \tag{5.44}$$

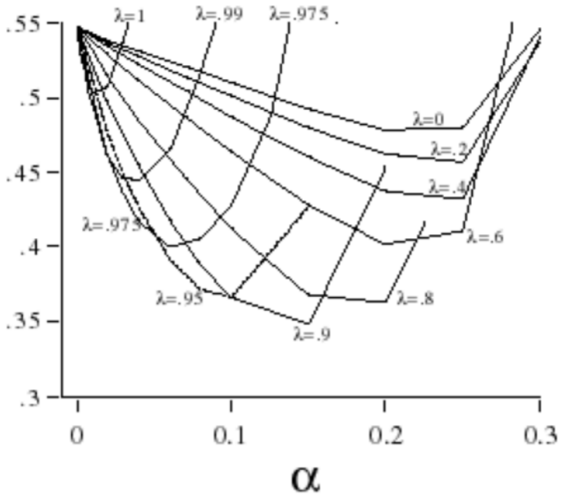
У случају да је стање посећено више пута, довољно је да одвојено посматрамо посете. Како смо већ доказали да је за сваку посету сума промена уназад једнака промени унапред, исто мора важити и за њихове суме. □

Поглед унапред и уназад се значајно разликују по питању временске сложености. При погледу унапред, довољно је да после завршене епизоде у обрнутом редоследу једном прођемо стања на агентовој трајекторији, јер ако је λ-повратак стања посећеног у тренутку t једнак:

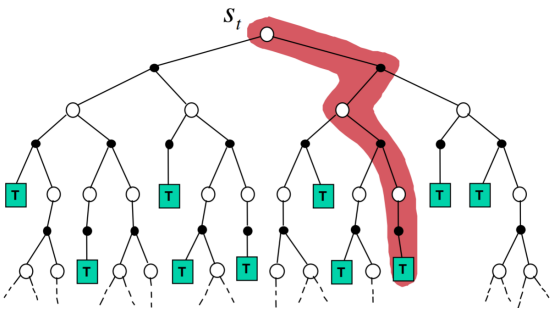
$$\begin{aligned}
G_{t-1}^\lambda &= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t-1}^{(n)} && (5.45)\\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}(R_t + \gamma R_{t+1} + ...) && (5.46)\\
&= (1-\lambda)R_t + \lambda(1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}(R_{t+1} + \gamma R_{t+2} + ...) && (5.47)\\
&= (1-\lambda)R_t + \lambda G_t^\lambda && (5.48)
\end{aligned}$$

Сложеност погледа унапред је $O(T)$, где је T дужина епизоде. При погледу уназад при сваком кораку ажурирамо свако стање, па је сложеност $O(nT)$, за n стања.
Видимо да поглед уназад има доста већу сложеност, па је боље користити поглед унапред уколико је практично сачекати крај епизоде.

Пример већег насумичног хода. На графику се види зависност просечне грешке по свим стањима од различитих вредности α при коришћењу λ-повратa. Различите криве се односе на различите вредности λ.

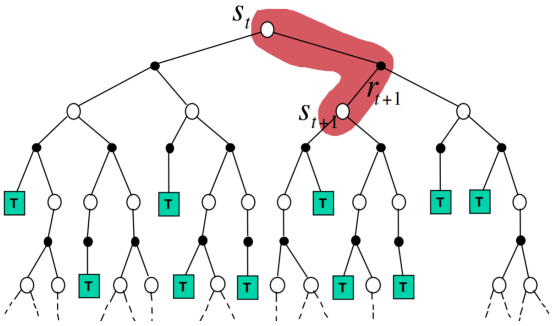


$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



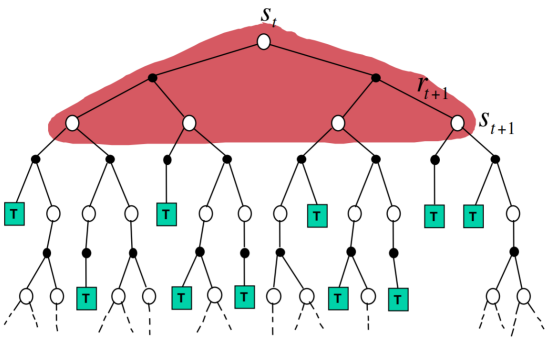
(a) Монте Карло

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



(b) Temporal difference

$$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$$



(c) Динамичко програмирање

Слика 29: Преглед различитих начина ажурирања вредносних функција у обрађеним алгоритмима

6 Контрола без модела

У претходна два поглавља видели смо како итеративном применом оцене и побољшања полисе долазимо до оптималне вредносне функције у случају MDP-а, и како да оценимо полису у случају да модел није познат. Спајањем ових идеја долазимо до алгоритама за контролу без модела.

Дефиниција 6.0.1. Кажемо да алгоритам учи **на полиси** када унапређује полису π на основу искуства прикупљеног пратећи ту полису.

Дефиниција 6.0.2. Кажемо да алгоритам учи **ван полисе** када унапређује полису π на основу искуства прикупљеног пратећи неку другу полису μ.

У случају учења на полиси агент мора да интереагује са окружењем и учи само из својих трајекторија. Учење ван полисе му омогућава да учи гледајући неког другог агента. Ово доноси низ предности. Може учити гледајући човека. Такође, када су интеракције са окружењем јако скупе, можемо тренирати више различитих агената по цени једног (на примеру самовозећих аутомобила, велика је разлика да ли ћемо их слупати 2 или 20). Међутим, оно често повећава дисперзију. У овом поглављу ћемо обрадити оба типа алгоритама.
Подсетимо се да се итерација полисе састоји из два дела: оцене и побољшања полисе. И даље користимо ова два корака, али за њих уводимо нове алгоритме.

6.1 Монте Карло контрола на полиси

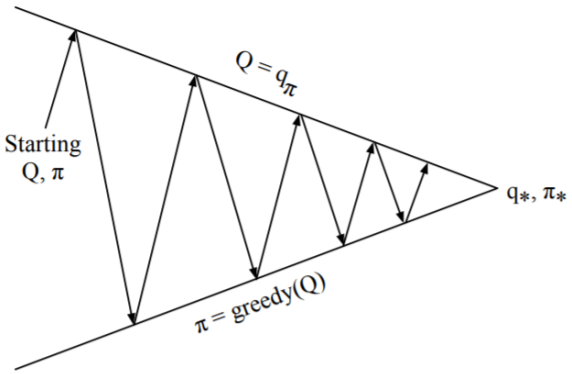
Да ли је комбинација Монте Карло оцене и похлепног побољшања полисе довољна за налазак оптималне полисе? Испоставља се да није. Постоје два проблема са овим приступом.
Први је то што похлепно побољшање полисе у односу на вредносну функцију $V(s)$ захтева познавање Марковљевог процеса одлучивања:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}(\mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')) \tag{6.1}$$

Решење: учимо вредносну функцију акција $Q(s, a)$:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a) \tag{6.2}$$

Подсетимо се да учење $Q(s, a)$ повећава сложеност, али је овде неизбежно.



Слика 30

Други проблем је то што радимо похлепна побољшања полисе. Увек похлепно бирање акција доводи до проблема истраживања. Наиме, како сада не знамо MDP, при учењу не можемо у обзир узети све могуће транзиције. Лако се можемо заглавити на неком лошем локалном максимуму и оставити већи део простора неистраженим.

Слика 31: Испред вас су двоја врата. Бирате једна од њих и добијате награду, која је стохастична. Пробали сте прва врата и добили награду 0. Потом друга, и овај пут добијате +1. Пратећи похлепну полису поново отварате друга врата и добијате награду +3. Процењена вредност бирања других врата је +2, а првих 0. Ако друга увек дају награде +1 и +3, изнова и изнова ћемо њих отварати, иако смо из првих видели само једну награду.

6.1.1 ϵ-похлепно побољшање полисе

Ово је вероватно најједноставнији начин да уведемо истраживање. У пракси се испоставља да тешко можемо боље од њега. При сваком доношењу одлуке, са вероватноћом ϵ бирамо насумичну акцију:

$$\pi'(s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{за } a* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s,a) \\ \epsilon & \text{у супротном} \end{cases} \tag{6.3}$$

Докажимо да ϵ-похлепно побољшање полису заиста чини бољом.

Теорема 6.1.1. За сваку ϵ-похлепну полису π, ϵ-похлепна полиса π' по q_π има већу или једнаку вредносну функцију, $v_{\pi'}(s) \geq v_\pi(s)$.

Доказ. Прво ћемо доказати да ако један корак користимо нову полису, а потом пратимо стару, повећавамо вредносну функцију:

$$q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \tag{6.4}$$

$$= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \max_{a \in \mathcal{A}} q_\pi(s,a) \tag{6.5}$$

$$\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi(s,a) \tag{6.6}$$

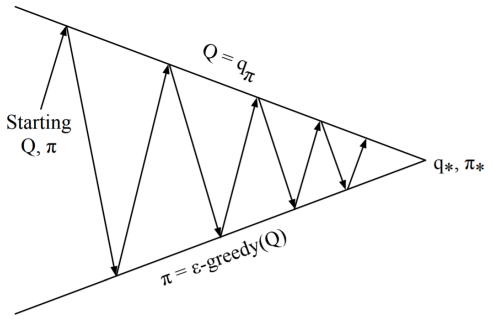
$$= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) = v_\pi(s) \tag{6.7}$$

Сада слично теореми о побољшању полисе коју смо доказали у поглављу о динамичком програмирању, неједнакост за један корак можемо применити на све кораке дуж путање, одакле следи $v_{\pi'}(s) \geq v_\pi(s)$.□

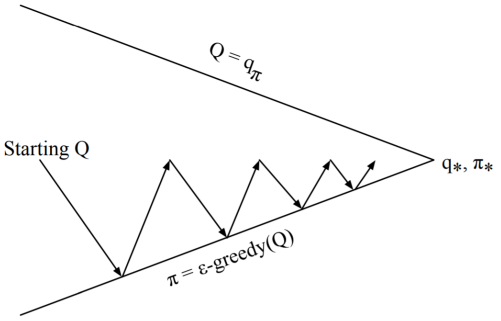
6.1.2 Монте Карло итерација полисе

Алгоритам сада узима следећи облик:

- MC оцена полисе, $Q = q_\pi$
- ϵ-похлепно побољшање



Већ смо видели да је у пракси често неефикасно чекати да оцена полисе исконвергира правој вредности. Можемо одмах искористити стечено знање ϵ-похлепним побољшањем после сваке епизоде.



Да ли ϵ-похлепни алгоритми заиста налазе оптималну вредносну функцију? Овде постоји баланс између две ствари. Неопходно је да довољно истражују, да им не би промакла нека информација, али асимптотски морају да теже похлепној полиси, јер је свака оптимална полиса похлепна.

Дефиниција 6.1.1. Учење је **похлепно у границама са бесконачним истраживањем (GLIE)** акко:

- Све парове стање-акција виђамо бесконачно много пута:

$$\lim_{k\to\infty} N_k(s,a) = \infty \tag{6.8}$$

- Полиса конвергира похлепној полиси:

$$\lim_{k\to\infty} \pi_k(a|s) = I(a = \underset{a'\in\mathcal{A}}{\operatorname{argmax}} Q_k(s,a')) \tag{6.9}$$

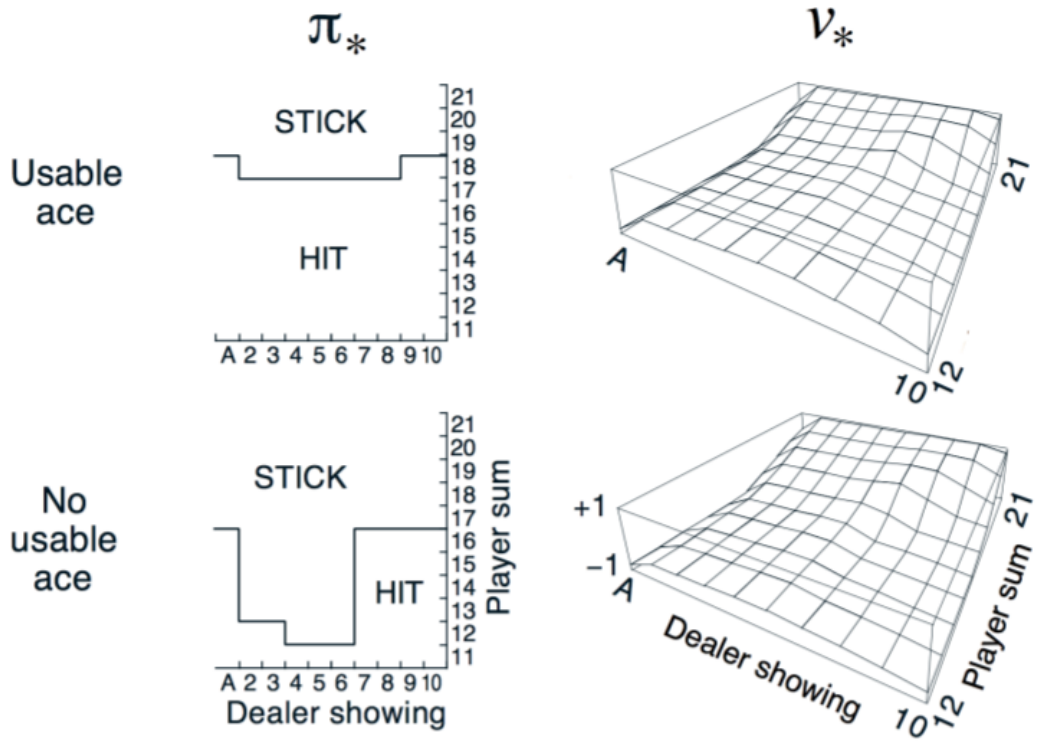
Идеја је да ϵ поступно смањујемо. На пример, ϵ-похлепно учење је GLIE за $\epsilon_k = 1/k$.

Теорема 6.1.2. GLIE Монте Карло контрола конвергира оптималној вредносној функцији акција, $Q(s,a) \to q_*(s,a)$

Дакле, GLIE својство гарантује да алгоритам истражује баш онолико колико је потребно.

6.2 Temportal Difference контрола на полиси

Већ смо видели више предности TD учења у односу на MC: мања дисперзија, учење из незавршених и непотпуних епизода, већа брзина конвергенције и мања минимална грешка. Природно се јавља идеја да користимо TD уместо MC оцене полисе.

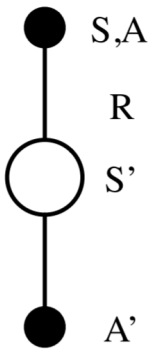


Слика 32: Монте Карло контрола на примеру поједностављеног Блек Џека из прошлог поглавља.

6.2.1 SARSA

Дошли смо до алгоритма по имену SARSA (State, Action, Reward, next State, next Action). Она је облик TD оцене, када уместо $V(s)$ користимо $Q(s,a)$. Ажурирамо вредност стања и акције за неку полису помоћу награде, наредног стања и наредне акције изабране истом том полисом:

$$(s,a) = Q(s,a) + \alpha(R + \gamma Q(s',a') - Q(s,a)) \tag{6.10}$$



Сада алгоритам контроле **у сваком кораку** врши:

- SARSA оцену полисе, $Q \approx q_\pi$
- ϵ-похлепно побољшање полисе

Псеудокод алгоритма:

SARSA

initialise $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and Q(terminal-state, $\cdot$) $= 0$
repeat (for each episode):
    initialise S
    choose A from S using policy derived from Q(e.g., ϵ-greedy)
    repeat (for each step of episode):
        take action A, observe R, S'
        choose A' from S' using policy derived from Q(e.g., ϵ-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

Да ли SARSA увек конвергира оптималној вредносној функцији? Овог пута нам је потребно још једно додатно својство.

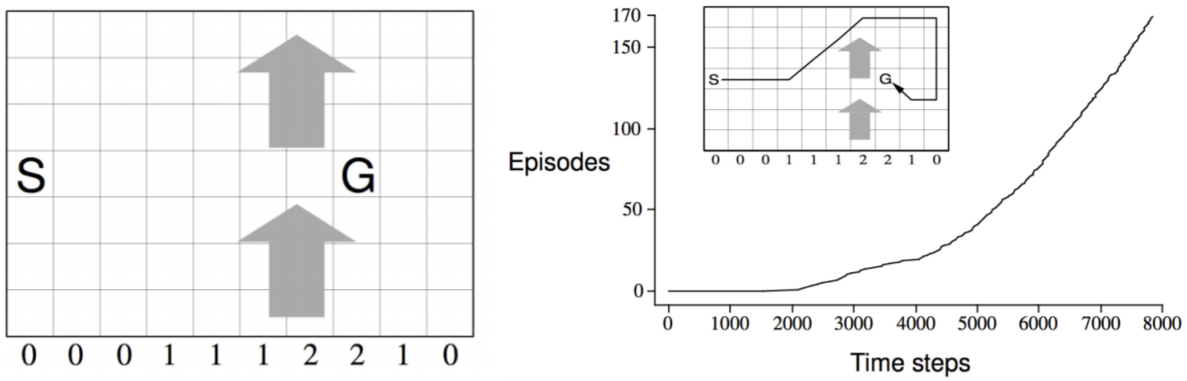
Теорема 6.2.1. SARSA конвергира оптималној вредносној функцији акција, $Q(s,a) \to q_*(s,a)$, под следећим условима:

- алгоритмом добијамо GLIE низ полиса
- и Робин-Монро низ величина корака α_t:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \tag{6.11}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty \tag{6.12}$$

Први услов нам говори да су кораци довољно велики, па ће почетна вредносна функција успети да достигне праву, а други да се кораци временом смањују, захваљујући чему ће вредносна функција исконвергирати правој.



Слика 33: Ветровити мрежни свет [5]. У сваком кораку агент може прећи на неко од суседних поља (горе, доле, лево или десно), а на одређеним местима дува ветар који га гура на горе. Циљ је доћи до поља G. На другој слици видимо оптималну полису добијену SARSA-ом. Генерално, за овај алгоритам карактеристично је дуго време до првог проналаска циља. Након тога, бутстрепинг омогућава да агент учи све брже и брже, па добијена крива има такав изглед.

Поново можемо уопштити TD учење на начин који смо то урадили у претходном поглављу.

6.2.2 SARSA у n-корака

Дефинишимо Q-поврат у n корака:

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}) \tag{6.13}$$

Сада SARSA ажурира $Q(s,a)$ у смеру овог поврата:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(q_t^{(n)} - Q(S_t, A_t)) \tag{6.14}$$

6.2.3 SARSA(λ) са погледом унапред

Као што смо већ видели, λ поврат комбинује поврате у n корака за све n:

$$q_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)} \tag{6.15}$$

SARSA(λ) ажурирања:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(q_t^{\lambda} - Q(S_t, A_t)) \tag{6.16}$$

Избором λ добијамо могућност балансирања између TD и MC, тј. између пристрасности и дисперзије.

6.2.4 SARSA(λ) са погледом уназад

Користимо трагове одговорности:

$$E_0(s,a) = 0 \tag{6.17}$$
$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + I(S_t = s, A_t = a) \tag{6.18}$$

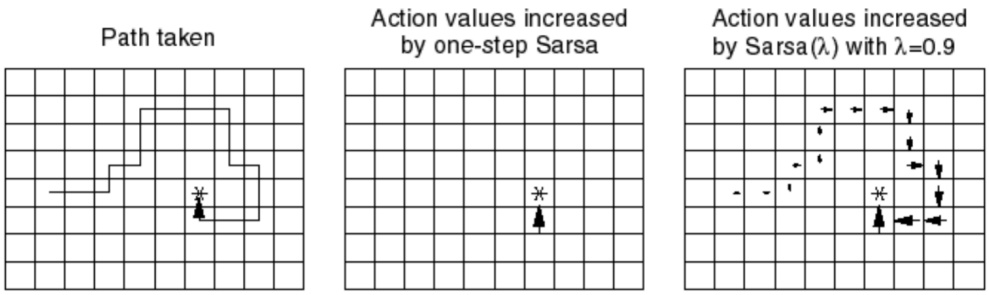
Ажурирања вршимо пропорционало TD грешци δ_t и трагу одговорности:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \tag{6.19}$$
$$Q(s,a) = Q(s,a) + \alpha\delta_t E_t(s,a) \tag{6.20}$$

Псеудокод SARSA(λ) алгоритма:

SARSA(λ)

initialise $Q(s,a), \forall s \in \mathcal{S},\ a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
repeat (for each episode):
    $E(s,a) = 0$, for all $s \in \mathcal{S},\ a \in \mathcal{A}(s)$
    initialise S, A
    repeat (for each step of episode):
        take action A, observe R, S'
        choose A' from S' using policy derived from Q(e.g.,ϵ-greedy)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $E(S,A) \leftarrow E(S,A) + 1$
        for all $s \in \mathcal{S},\ a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
            $E(s,a) \leftarrow \gamma\lambda E(s,a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal



Слика 34: Пример разлике између обичног и λ SARSA алгоритма. На првој слици је приказана агентова трајекторија. Величине стрелица представљају величину вредносне функције одговарајућих акција.

6.3 Учење ван полисе

Видели смо да истраживање представља велики проблем када немамо модел. Учење ван полисе нам омогућава да оценимо **циљну полису** $\pi(a|s)$ пратећи **полису понашања** $\mu(a|s)$. На овај начин можемо наћи оптималну полису док пратимо неку истраживачку полису. Такође, можемо поново искористити старо искуство, што ће бити од велике важности у наредном поглављу.

Погледајмо сада два механизма којима ово постижемо.

6.3.1 MC и TD учење ван полисе коришћењем Importance sampling-a

Генерална идеја importance sampling-a је да врши процену расподеле P на основу неке друге расподеле Q, тако што узимамо узорке из Q, и множимо их са количником вероватноће да тај узорак добијемо из расподеле P и Q:

$$\mathbb{E}_{X\sim P}[f(X)] = \sum P(X)f(X) \tag{6.21}$$

$$= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \tag{6.22}$$

$$= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right] \tag{6.23}$$

Importance sampling је у суштини користан када је лакше узимати узорке из Q него P.

Интегришемо га у MC тако што отежинимо поврат:

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})}\cdots\frac{\pi(A_T|S_T)}{\mu(A_T|S_T)}G_t \tag{6.24}$$

$$V(S_t) = V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t)) \tag{6.25}$$

Иако на први поглед делује примамљиво, MC ван полисе доноси низ проблема који га чине практично неупотребљивим. Постоји могућност дељења са нулом, ако је $\mu(A_t|S_t) = 0$ када $\pi(A_t|S_t)$ није, што је врло могућа појава. Такође, како стања и акције добијамо праћењем полисе μ, именилац разломка ће углавном бити доста већи него бројилац, па ће поврат постати безначајно мали. Уз све то, ово значајно повећава дисперзију.

Када учимо ван полисе, неопходно је да користимо TD:

$$V(S_t) = V(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})) - V(S_t)\right) \tag{6.26}$$

У пракси, ни TD са importance sampling-ом не даје претерано добре резултате, па користимо друге методе.

6.3.2 Q-учење у ширем смислу

Ово је најефикаснија метода за учење ван полисе. Идеја је врло једноставна. Када смо у неком стању S, наредну акцију A_t бирамо помоћу полисе понашања, чиме долазимо у ново стање S_{t+1}. Након тога **разматрамо** алтернативну акцију коју би изабрали у новом стању пратећи циљну полису. Потом $Q(S_t, A_t)$ ажурирамо у смеру алтернативне акције:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)) \tag{6.27}$$

Ово можемо схватити као уопштење TD(0) учења. При TD(0) учењу, циљна и полиса понашања су исте. Елеганција Q-учења је у томе што циљна полиса не узима у обзир никакве специфичности полисе понашања, већ само виђене награде, које немају никакву пристрасност. Овде нам μ служи да прикупи што више информативних транзиција.
На пример, могли би да пратимо насумичну полису, а да учимо оптималну, и да тиме на неки начин симулирамо исцрпну претрагу какву смо имали у динамичком програмирању.

6.3.3 Q-учење у ужем смислу

Када се спомене Q-учење, углавном се мисли на овај специјални случај. Сада нам је циљна полиса похлепна у односу на $Q(s, a)$:

$$\pi(S_{t+1}) = \underset{a'}{\text{argmax}} Q(S_{t+1}, a') \tag{6.28}$$

А полиса понашања ϵ-похлепна у односу на $Q(s, a)$. Обе полисе се побољшавају.
Овим добијамо поједностављену мету:

$$R_{t+1} + \gamma Q(S_{t+1}, A') \tag{6.29}$$
$$= R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\text{argmax}} Q(S_{t+1}, a')) \tag{6.30}$$
$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \tag{6.31}$$

па ажурирања узимају облик:

$$Q(s, a) = Q(s, a) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right) \tag{6.32}$$

Зашто пратимо ϵ-похлепну полису? Зашто не насумичну? ϵ-похлепна полисе ја углавном веома ефикасна, јер довољно истражује, а не расплињује се превише на све делове порстора, чиме омогућава брже учење вредносних функција у битним тачкама.

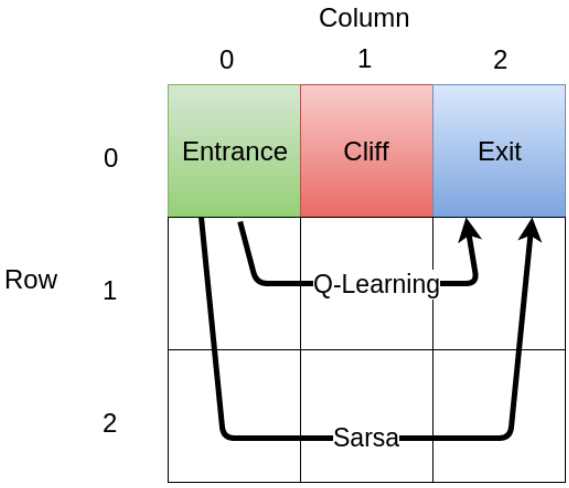
Може се доказати да Q-учење заиста конвергира.

Теорема 6.3.1. Q-учење конвергира оптималној вредносној функцији акција, $Q(s, a) \to q_*(s, a)$

Направимо контраст између ДП и TD метода и Белманових једначина које имплицитно користе.

Белманова једначина	ДП	TD
Белманова једначина очекивања за $v_\pi(s)$	Итеративна оцена полисе	TD учење
Белманова једначина очекивања за $q_\pi(s, a)$	Q-итерација полисе	SARSA
Белманова једначина оптималности за $q_*(s, a)$	Q-итерација вредносне функције	Q-учење

Пример разлике између учења на и ван полисе. Претпоставимо да ϵ има фиксну вредност. ϵ-похлепна полиса има не нулту вероватноћу да насумично скрене ка провалији. SARSA учи на полиси, стога урачунава ризик од слетања у провалију, па иде заобилазним путем. Са друге стране, Q-учењем налази најкраћи пут до решења. (Када би GLIE својство било задовољено и SARSA би нашла најкраћи пут.)



7 Апроксимација вредносне функције

У шаху постоји око 10^{45} стања. Ако управљамо роботом, простор могућих очитавања сензора је континуалан. Већина реалних проблема имају својство да је скуп стања превише велики, па не можемо експлицитно памтити вредносне функције за свако стање. Очигледно морамо надоградити методе које смо до сада користили. Циљ наших алгоритама је да раде са произвољно великим простором стања.

7.1 Апроксиматори вредносне функције

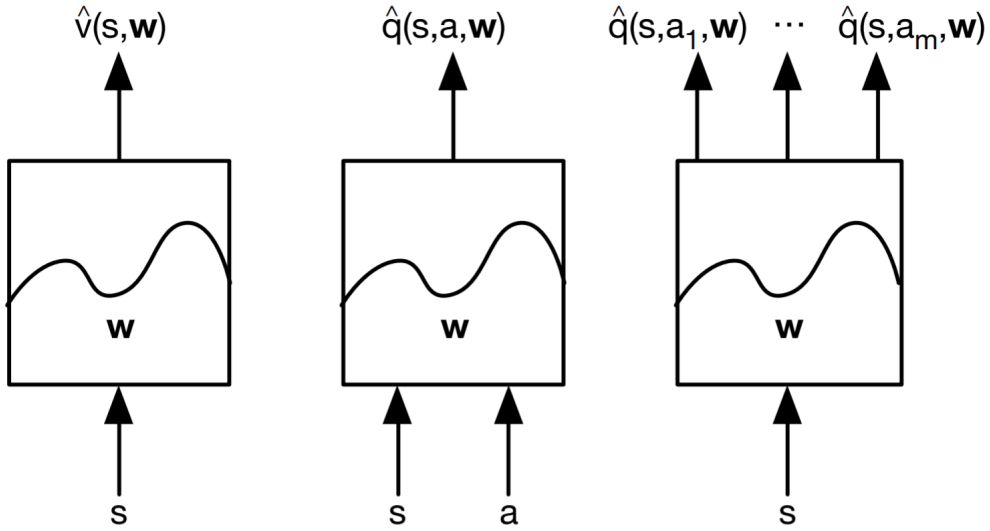
Када човек научи да вози аутомобил, његово знање није ограничено само на полигон на којем је учио. За људе, аутопут, градска улица, пут по којем пада киша, колона возила, пут у Новом Саду и пут у Београду нису потпуно различита стања. Ово је могуће зато што људи поседују способност генерализације. Ово постижемо коришћењем апроксиматора вредносне функције.

До сада смо вредносне функције представљали "таблицом", у којој се налазило поље за сваки пар стање, акција. Чак и када би имали довољно меморије да чувамо сва стања, учење би једноставно било преспоро.
Могућност генерализације смањује меморију и време извршавања, али смањује и прецизност.
Арпоксиматоре означавамо на следећи начин:

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s) \tag{7.1}$$

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a) \tag{7.2}$$

$\boldsymbol{w}$ представља вектор параметара апроксиматора. Апроксиматор је функција стања (акције) и параметара, па и од њих зависи како ћемо процењивати стања. Учење вршимо мењањем параметара, што нам омогућава да мењамо процене вредносне функције.



Слика 35: Различите врсте апроксиматора у учењу са појачавањем. Прва слика је апроксимација вредносне функције стања, друга апроксимира вредносну функцију акције, а трећа за неко стање даје вредносне функције свих могућих акција. У зависности од проблема, један од ових приступа ће бити најефикаснији.

Постоји велики број различитих апроксиматора функција. За учење са појачавањем потребно нам је да апроксиматор буде диференцијабилан, то јест да можемо наћи парцијални извод за сваки параметар. Када знамо градијент апроксиматора, знамо на који начин промена параметара утиче на излаз функције, па самим тим и можемо оцену вредности за неко стање померити

у жељеном смеру.

Градијент представља уопштење извода у случају функција више променљивих (функција над векторима). Градијент је вектор, садржи парцијалне изводе по свакој променљивој. Знак извода нам говори да ли функција расте или опада (што можемо схватити и као смер у ком функција расте), а апсолутна вредност колико се брзо функција мења. Слично, k-ти елемент градијента нам говори како ће се функција мењати ако улаз мењамо само у k-тој оси. Знак сваког елемента градијента нам говори у ком смеру да померамо тај елемент да би функција расла. Захваљујући томе, градијент даје смер најбржег раста функције, и то је својство које ћемо користити.

Велики број апроксиматора најбоље раде када их тренирамо над подацима који су стационарни (што би значило да од почетка имамо исто искуство), и да су сви једнако расподељени, тј. да над сваким податком подједнако често вршимо учење. У нашем случају, подаци су транзиције агента. Очигледно, ниједно од ова два својства није задовољено. Агент свакако може открити потпуно нова стања, и нека може посетити само једном, док друга посећује врло често.
У овом поглављу ћемо се фокусирати на линеарну комбинацију карактеристика, ради једноставности. У пракси се најчешће користе неуралне мреже [10].

7.2 Инкременталне методе

Инкрементални алгоритми ажурирања примењују на стања оним редом којим на њих наилазе и након тога одбацују виђено искуство. У наредном одељку видећемо како можемо ефикасније користи искуство, што ће бити изузетно битно у случају нелинеарних апроксиматора.

7.2.1 Градијентни спуст

Нека је $J(\boldsymbol{w})$ диференцијабилна функција вектора параметара $\boldsymbol{w}$. Градијент дефинишемо као:

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w_1}} \\ \vdots \\ \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}_n} \end{pmatrix} \tag{7.3}$$

Да би нашли локални минимум за $J(\boldsymbol{w})$ довољно је да вектор параметара $\boldsymbol{w}$ итеративно малим корацима померимо у смеру негативног градијента:

$$\triangle \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \tag{7.4}$$

α је величина корака, $\frac{1}{2}$ није битна, служи да би се формула коју ћемо касније видети лепше средила. Ово је основна идеја градијентног спуста.

У случају оцене вредносне функције циљ је наћи вектор параметара $\boldsymbol{w}$ који минимизује очекивано средње квадратно одступање апроксимације $\hat{v}_\pi(s, \boldsymbol{w})$ и праве вредносне функције $v_\pi(s)$:

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \boldsymbol{w}))^2 \right] \tag{7.5}$$

Градијентни спуст налази локални минимум итеративним ажурирањем вектора параметара:

$$\triangle \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \tag{7.6}$$

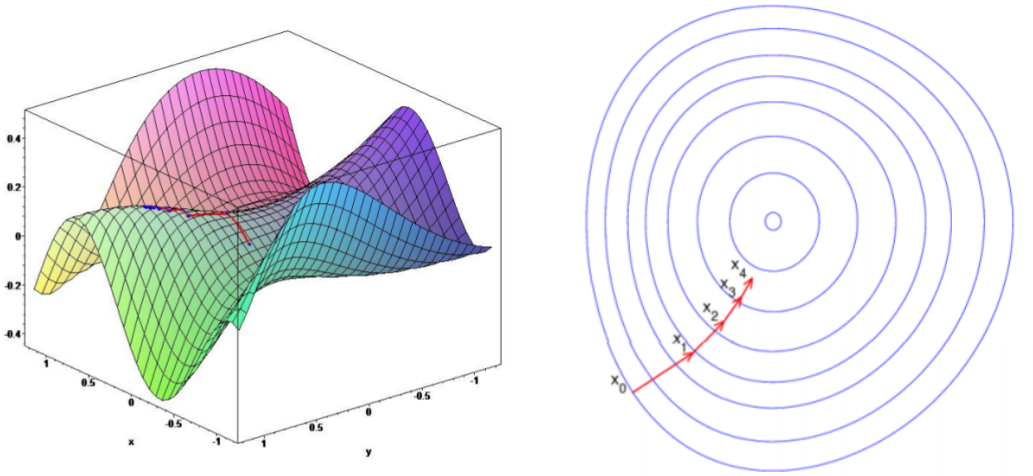
$$= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \boldsymbol{w})) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) \right] \tag{7.7}$$

Да би израчунали очекивану вредност неопходно је да знамо динамику окружења, па је чист градијентни спуст за нас неупотребљив. Уместо тога користимо стохастични градијентни спуст, који ажурирања врши у правцу појединачних градијената:

$$\triangle \boldsymbol{w} = \alpha(v_\pi(S) - \hat{v}(S, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{v}(S, \boldsymbol{w}) \tag{7.8}$$

После довољно итерација, стохастични градијентни спуст тежи обичном.

v_π које смо у претходним формулама користили нам није познато, њега ћемо заменти MC и TD метама које смо обрадили.



Слика 36: Визуелизација градијентног спуста.

7.2.2 Вектор карактеристика

Циљ вектора карактеристика је да стање представи преко низа вредости које га карактеришу. Те карактеристичне вредности треба што боље да описују стање у којем се агент налази. Када користимо линеарни апроксиматор, на улазу му дајемо овај вектор. Уколико је избор карактеристика добар, тј. ако у себи садржи све битне информације проблема који решавамо, може довести до значајно бржег учења.

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix} \tag{7.9}$$

На пример, робот који жели да дође до неке значајне тачке, може уместо своје апсолутне локације у универзуму користити удаљеност од те тачке.

7.2.3 Апроксимација линеарном комбинацијом карактеристика

Вредносну функцију представљамо линеарном комбинацијом карактеристика:

$$\hat{v}(S, \boldsymbol{w}) = x(S)^T \boldsymbol{w} = \sum_{j=1}^{n} x_j(S)\boldsymbol{w}_j \tag{7.10}$$

$x(S)^T$ је транспонована матрица $x(S)$.

Тежине одређују колико је битна свака од карактеристика. Линеарном комбинацијом у већини случајева не добијамо савршену апроксимацију, али имамо једноставну репрезентацију стања са којом је лако радити.

Такође битно својство је то што је циљна функција сада квадратна:

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - x(S)^T \boldsymbol{w}))^2 \right] \tag{7.11}$$

Ово значи да нема локалних минимума, па стохастични градијентни спуст увек проналази глобални минимум. Ажурирања добијају следећи облик:

$$\nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}) = x(S) \tag{7.12}$$

$$\triangle \boldsymbol{w} = \alpha(v_\pi(S) - \hat{v}(S, \boldsymbol{w}))x(S) \tag{7.13}$$

Дакле, k-ту тежину ажурирамо производом величине корака, грешке процене и вредности k-те карактеристике. Што је већа активираност неке карактеристике, она више утиче на тренутно стање, па ће и њена промена бити већа. Ово и интуитивно има смисла, нпр. FALI PRIMER

Табеларни приступ (чување свих стања) који смо до сада користили је само специјални случај линеарне апроксимације. Да би се у ово уверили, довољно је да вектор карактеристика има низ индикатора за свако од могућих стања.

$$x^{table}(S) = \begin{pmatrix} I(S = s_1) \\ \vdots \\ I(S = s_n) \end{pmatrix} \tag{7.14}$$

Сада сваки параметар w_k даје вредност k-тог стања:

$$\hat{v}(S, \boldsymbol{w}) = \begin{pmatrix} I(S = s_1) \\ \vdots \\ I(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_n \end{pmatrix} \tag{7.15}$$

7.2.4 Алгоритми предвиђања

У досадашњем делу поглавља претпоставили смо да знамо праву вредносну функцију v_π. Ово у учењу са појачавањем наравно није случај. Потребно је да v_π заменимо метом. Ово постижемо једним од алгоритама које смо већ видели:

7.2.4.1 Монте Карло оцена

У случају Монте Карло оцене, мета је поврат G_t:

$$\triangle \boldsymbol{w} = \alpha(G_t - \hat{v}(S_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) \tag{7.16}$$

Подсетимо се да је поврат непристрасна оцена вредносне функције, високе дисперзије. Овај алгоритам можемо схватити и као супервизирано учење над "тренинг подацима:"

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \ldots, \langle S_T, G_T \rangle \tag{7.17}$$

На пример, ако користимо линеарну Монте Карло оцену добијамо:

$$\triangle \boldsymbol{w} = \alpha(G_t - \hat{v}(S_t, \boldsymbol{w}))x(S_t) \tag{7.18}$$

Монте Карло оцена конвергира локалном оптимуму чак и када користимо нелинеарне апроксиматоре, што је последица непристрасности поврата. Ово својство нећемо доказивати.

7.2.4.2 TD оцена

За TD(0) оцену, користимо TD мету $R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w})$:

$$\triangle \boldsymbol{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{v}(S_t, \boldsymbol{w}) \tag{7.19}$$

TD мета је пристрасна оцена вредносне функције. Као и за MC, овај алгоритам можемо схватити као супервизирано учење над виђеним искуством:

$$\langle S_1, R_2 + \gamma\hat{v}(S_2, \boldsymbol{w})\rangle, \langle S_2, R_3 + \gamma\hat{v}(S_3, \boldsymbol{w})\rangle, \ldots, \langle S_{T-1}, R_T\rangle, \tag{7.20}$$

У случају линеарне TD(0) оцене, промене тежина добијају следећи облик:

$$\triangle \boldsymbol{w} = \alpha(R_{t+1} + \gamma\hat{v}(S', \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}))x(S_t) \tag{7.21}$$

Може се доказати да линеарна TD(0) оцена конвергира глобалном оптимуму. Ово није гарантовано ако користимо нелинеарне апроксиматоре. Ипак, у пракси се показује да TD методе на већини проблема раде и боље него MC, у смислу брже конвергенције и мање крајње грешке коју достижу.

7.2.4.3 TD(λ) оцена

За TD(λ), мета је λ-повратак G_t^λ:

$$\triangle \boldsymbol{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}}\hat{v}(S_t, \boldsymbol{w}) \tag{7.22}$$

λ-повратак је оцена мање пристрасности од TD мете и мање дисперзије од повратка. Поново алгоритам можемо схватити као супервизирано учење над виђеним искуством:

$$\left\langle S_1, G_1^\lambda\right\rangle, \left\langle S_2, G_2^\lambda\right\rangle, \ldots, \left\langle S_T, G_T^\lambda\right\rangle \tag{7.23}$$

Ажурирања параметара на примеру линеарне TD(λ) са погледом унапред имају следећи облик:

$$\triangle \boldsymbol{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \boldsymbol{w}))x(S_t) \tag{7.24}$$

Ако користимо поглед уназад, потребни су нам трагови одговорности. Овог пута, траг не чувамо за свако стање, већ за сваку карактеристику. Када наиђемо на неко стање, сваки траг одговорности увећавамо за вредност одговарајуће карактеристике:

$$\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}) \tag{7.25}$$

$$E_t = \gamma\lambda E_{t-1} + x(S_t) \tag{7.26}$$

$$\triangle \boldsymbol{w} = \alpha\delta_t E_t \tag{7.27}$$

Погледи унапред и уназад су еквивалентни.

7.2.5 Алгоритми контроле

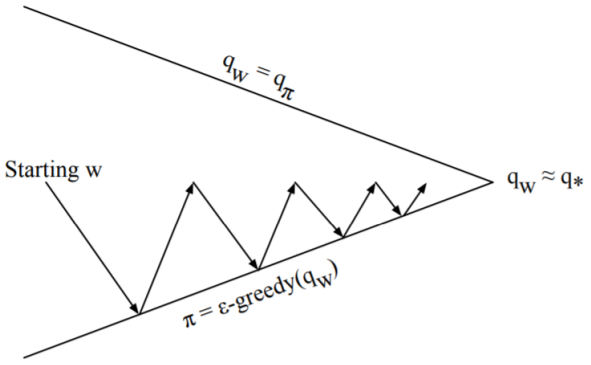
Већ смо видели два битна додатка која контрола захтева у односу на предвиђање: коришћење вредносне функције акција $q(s, a)$ (сада користимо апроксиматор ове функције $\hat{q}(S, A, \boldsymbol{w})$) и ϵ-похлепно истраживање.

Као и до сада, алгоритми контроле са апроксиматором се састоје из два дела:

- Оцена полисе. Као и у случају без апроксиматора, није неопходно да оцена дође до праве вредносне функције. Уместо тога радимо пар корака оцене (често 1) и одмах побољшавамо полису. Када користимо апроксиматор, чак је и бесмислено чекати праву вредносну функцију, јер самим коришћењем апроксиматора углавном губимо могућност да је потпуно прецизно одредимо.

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

Слика 37: Конвергенције различитих алгоритама предвиђања.



- ϵ-похлепно побољшање полисе.

Како користимо апроксиматор, немамо гаранцију да ће алгоритам контроле конвергирати оптималној вредносној функцији. Најбоље чему можемо да се надамо је да наша оцена постаје ближа и ближа q_*. Испоставља се да ни ово није случај. У пракси ови алгоритми осцилују око решења као у некој посуди, и некада изађу из посуде па се поново врате. Иако теријски немају никакве гаранције конвергенције, алгоритми углавном после довољно великог броја корака долазе прилично близу оптималне вредносне функције.

Погледајмо сада како изгледају коришћени појмови у случају вредносне функције акција. Желимо да минимизујемо средње квадратно одступање:

$$J(\boldsymbol{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}))^2 \right] \tag{7.28}$$

Формула за ажурирања вектора параметара стохастичним градијентним спустом:

$$\triangle \boldsymbol{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(S, A, \boldsymbol{w}) \tag{7.29}$$

Сада у случају линеарног апроксиматора, пар (стање, акција) представљамо вектором карактеристика:

$$x(S, A) = \begin{pmatrix} x_1(S, A) \\ \vdots \\ x_n(S, A) \end{pmatrix} \tag{7.30}$$

Ажурирања параметара у случају линеарног апроксиматора:

$$\triangle \boldsymbol{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}))x(S, A) \tag{7.31}$$

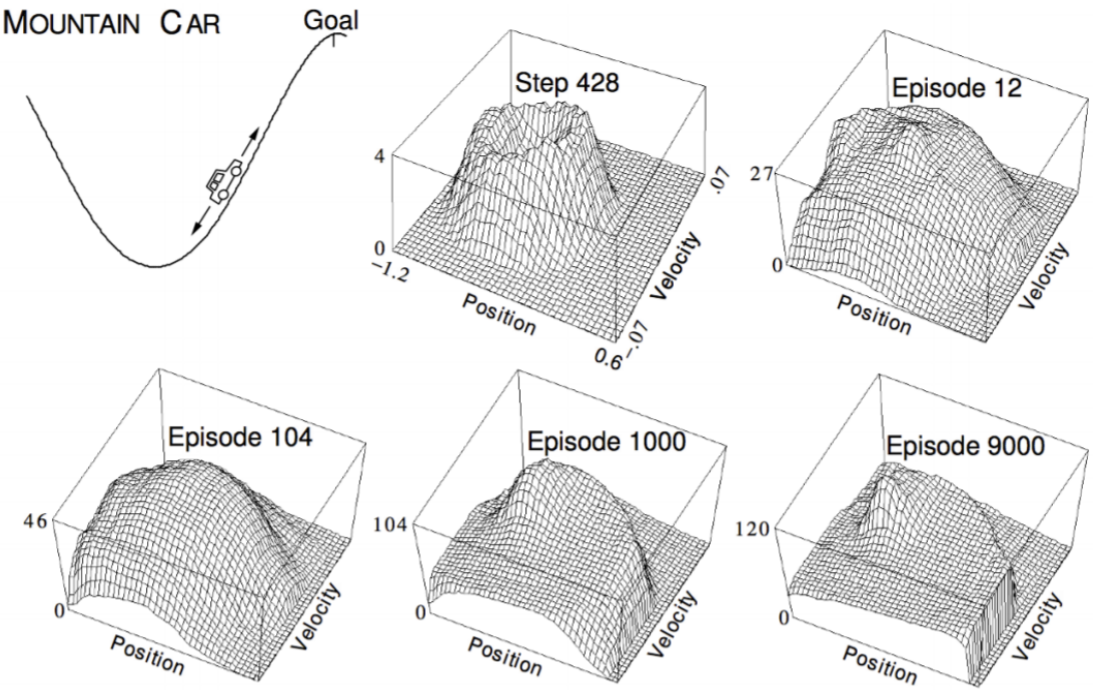
Ажурирања са убаченом метом за редом MC, TD (SARSA) и TD(λ) са погледом унапред:

$$\triangle \boldsymbol{w} = \alpha(G_t - \hat{q}(S, A, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(S, A, \boldsymbol{w}) \tag{7.32}$$

$$\triangle \boldsymbol{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S, A, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(S, A, \boldsymbol{w}) \tag{7.33}$$

$$\triangle \boldsymbol{w} = \alpha(q_t^\lambda(S, A) - \hat{q}(S, A, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(S, A, \boldsymbol{w}) \tag{7.34}$$

$$\tag{7.35}$$



Слика 38: SARSA алгоритам са апроксиматором функције на примеру аутомобила на планини. Циљ је попети се на десни врх планине. Стање нам је дато са два броја: x координатом аутомобила и његовом брзином, који су континуални. У сваком кораку аутомобил може да убрза на десно или лево. Нема довољно снаге да се одједном попне до циља, већ мора да се "заљуља", тј. да оде на једну страну, потом се пусти и оде што више на другу, и тако више пута док не стекне довољну брзину. На сликама је приказан апроксиматор вредносне функције. Нећемо улазити у његове детаље. Битно је знати да је састављен од великог броја плочица које се преклапају, тако да ажурирања вредносне функције у једној тачки повлаче у истом смеру и њој суседне тачке.

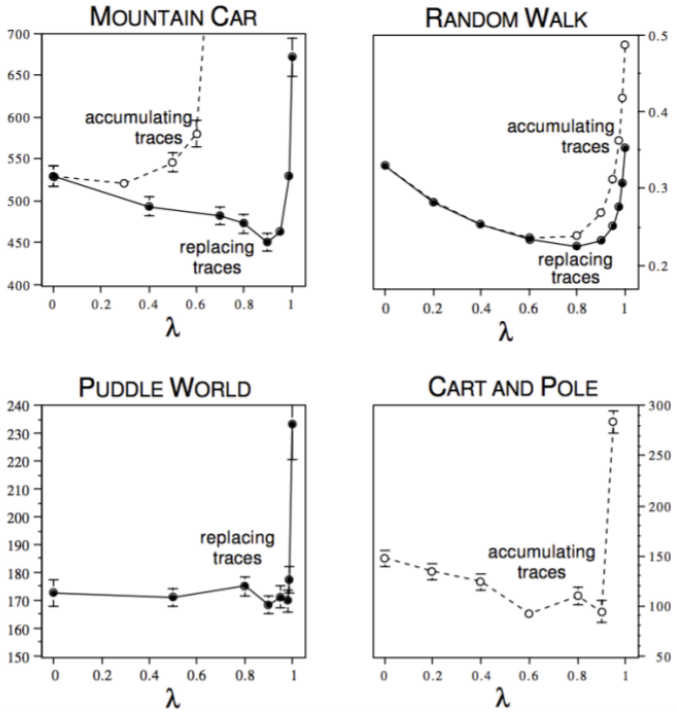
Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

Слика 39: Конвергенција различитих алгоритама контроле. ✓значи да алгоритам равномерно конвергира, тј. сваким кораком је све ближи решењу. (✓) представља "цвокотаву" конвергенцију, што значи да се алгоритам једно време приближава па потом одаљава од решења, али у целини конвергира релативно близу оптималне вредносне функције.

7.3 Скуповне методе

Инкременталне методе одбацују виђене транзиције после једног ажурирања. Ово није ефикасно јер једном искоришћено искуство и даље има сазнајну вредност. Поновно коришћење искуства памтимо у меморији искуства. Ове методе су скуповне, јер сада при сваком кораку нећемо ажурирати вектор параметара само једном транзицијом, већ скупом транзиција (величине скупова у пракси су углавном 16, 32 или 64).

Најчешће коришћени апроксиматор функција у пракси су неуралне мреже. Да би оне научиле добру апроксимацију функције, потребно је да подаци којима је тренирамо буду што више



Слика 40: На графицима су приказане просечне грешке апроксимације за различите изборе λ у случају 4 позната проблема. Као и без апроксиматора, MC методе поново дају најгоре резулате, иако нам гарантују конвергенцију локалном минимуму. Бутстрепинг повећава ефикасност. Али морамо бити опрезни јер постоје примери у којима ова метода дивергира. Градијентно Q-учење

једнако расподељени. У нашем контексту, ово би значило да је при сваком кораку једнако вероватно да било коју могућу транзицију користимо за ажурирање. Како немамо увид у све могуће транзиције, најбоље што можемо је да памтимо и искористимо виђено искуство. Инкременталне методе очигледно немају ово својство. Стања на трајекторијама које агент пролази су суседна, па самим тим и јако корелисана. Неуронална мрежа ће се превише фокусирати на уску групу стања, за њих ће научити добру апроксимацију, али ће за остала катастрофално дивергирати.

7.3.1 Минимално квадратно предвиђање

Сада, уз апроксиматор вредносне функције $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$, чувамо и читаво виђено искуство $\mathcal{D}$ сачињено од парова (стање, акција):

$$\mathcal{D} = \langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle \tag{7.36}$$

Потребан нам је начин да одредимо колико добро апроксиматор процењује вредности из искуства.

Алгоритми минималног квадратног предвиђања налазе вектор параметара $\boldsymbol{w}$ који минимизује укупно квадратно одступање између апроксимације $\hat{v}(s_t, \boldsymbol{w})$ и праве вредности v_t^π:

$$LS(\boldsymbol{w}) = \sum_{t=1}^{T} \left(v_t^\pi - \hat{v}(s_t, \boldsymbol{w})\right)^2 \tag{7.37}$$

$$= \mathbb{E}_{\mathcal{D}} \left[v^\pi - \hat{v}(s, \boldsymbol{w})\right] \tag{7.38}$$

7.3.2 Стохастични градијентни спуст са меморијом искуства

Ово су алгоритми минималног квадратног предвиђања који итеративно примењује два корака:

- Узимамо узорак (стање, акција) из искуства:

$$\langle s, v^\pi \rangle \sim \mathcal{D} \tag{7.39}$$

- Вршимо ажурирање стохастичним градијентним спустом:

$$\triangle \boldsymbol{w} = \alpha(v^\pi - \hat{v}(s, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w}) \tag{7.40}$$

Овим поступком вектор параметара конвергира минималном квадратном решењу:

$$\boldsymbol{w}^\pi = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ LS(\boldsymbol{w}) \tag{7.41}$$

Наравно, нама права вредносна функција v_π није позната. Уместо ње морамо убацити неку мету. Како користимо стара искуства, учење мора бити ван полисе, па се алгоритам Q-учења намеће као логичан избор.

7.3.3 Дубоке Q-мреже (DQN) са меморијом искуства

Ово је алгоритам Q-учења који као апроксиматор користи дубоку неуралну мрежу [10], и уз то имплементира две битне идеје: **меморију искуства** и **фиксне Q-мете**.

Шта подразумевају фиксне Q мете? Када користимо апроксиматор вредносне функције, i-то Q-ажурирање узимаја следећи облик:

$$\triangle w_i = \alpha(r + \gamma \max_{a'}\ \hat{q}(s', a', w_i) - \hat{q}(s, a, w_i))\nabla_{w_i} \hat{q}(S, A, w_i) \tag{7.42}$$

Подсетимо се да из стања s акцију a бирамо на основу ϵ-похлепне полисе.
Идеја је да замрзнемо параметре који се користе за процену мете, тј. да $r + \gamma \max_{a'} \hat{q}(s', a', w_i)$ заменимо са $r + \gamma \max_{a'} \hat{q}(s', a', w_i^-)$, при чему вектор w_i^- на почетку постављамо да буде баш w_i, а потом га неки број корака не мењамо (у пракси углавном пар хиљада корака), након чега га поново изједначавамо са w_i.

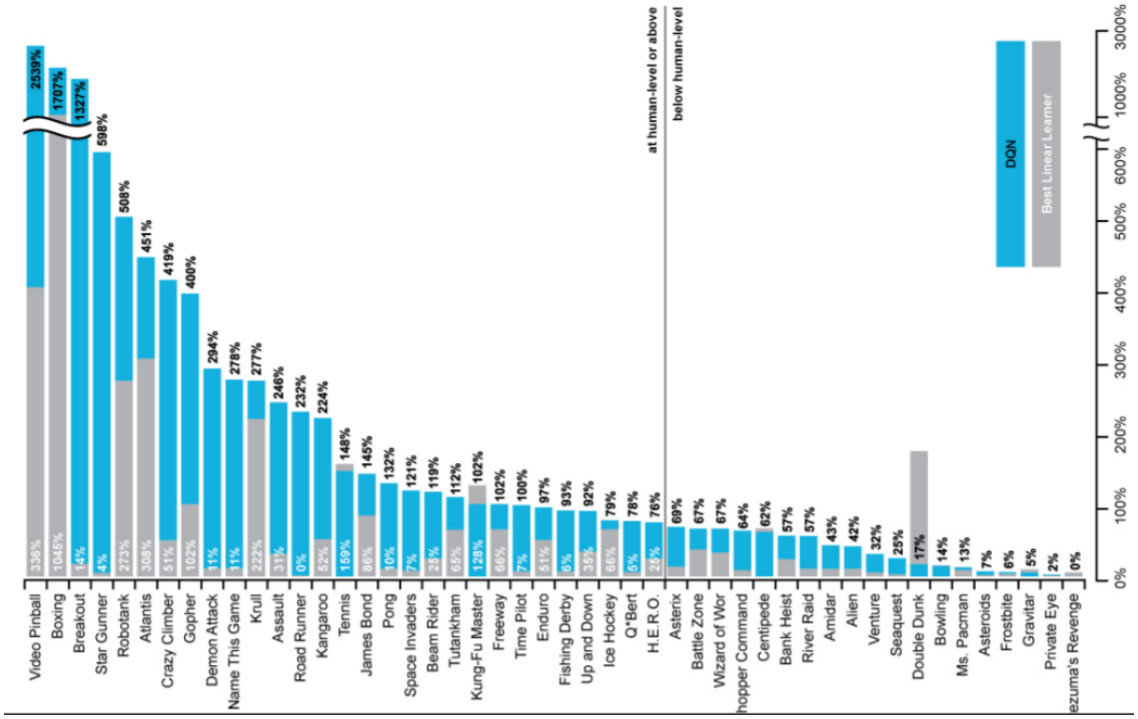
Зашто уводимо фиксне Q-мете? Оне значајно стабилизују учење. Дубоке неуралне мреже су врло нестабилне, и ако им дамо низ корелисаних стања лако дивергирају. Иако коришћењем меморије искуства омогућавамо насумично бирање транзиција, насумично можемо више пута извући мање-више корелисана стања и потпуно упропастити до тада изграђену апроксимацију. Иако на први поглед овај ефекат не делује значајно, у пракси се испоставља да јесте. Тек када фиксирамо параметре мете на пар хиљада корака, вероватноћа да нам се ово деси постаје занемарљиво мала.

DQN је од великог значаја. У време када је изашао (2013) био је први агент који је успешно играо игрице из Атари [11] домена и покренуо је низ других истраживања у овој области. Изузетно је импресивно то што су агенту дате само слике екрана. Он нема никакву информацију о томе шта ствари на екрану значе. Ово значајно отежава учење, јер агент сам мора да научи да тумачи слике. Тип апроксиматора који се користи за рад над сликама је конволуциона неуралнa мрежа [10], у чије детаље нећемо улазити.

Псеудокод алгоритма, при сваком кораку ажурирања вршимо помоћу мини-скупова:

DQN WITH EXPERIENCE REPLAY

INITIALIZE REPLAY MEMORY D
INITIALIZE PARAMETER VECTORS w AND w^- WITH RANDOM NUMBERS
FOR ALL EPISODES
    OBSERVE INITIAL STATE s
    REPEAT
        WITH PROBABILITY ϵ SELECT A RANDOM ACTION
        OTHERWISE SELECT $a = \underset{a'}{\operatorname{argmax}}\, Q(s, a')$
        EXECUTE ACTION a AND OBSERVE REWARD r AND NEW STATE s'
        STORE EXPERIENCE (s, a, r, s') IN REPLAY MEMORY D
        SAMPLE RANDOM MINIBATCH OF TRANSITIONS (s_j, a_j, r_j, s'_j) FROM D
        SET $y_j = \begin{cases} r_j \text{ for terminal } s'_j \\ r_j + \gamma \max_{a'} \; Q(s'_j, a', w^-) \text{ otherwise} \end{cases}$
        PERFORM GRADIENT DESCENT STEP ON $(y_j - Q(s_j, a_j, w))^2$
        IF NEEDED SET $w^- = w$
    UNTIL END OF EPISODE
END FOR



Слика 41: Резултати на Атари игрицама [6]. DQN постиже боље резултате од стручног људског играча на свим игрицама лево од вертикалне линије.

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

Слика 42: Просечна укупна награда по епизоди на више игрица [6], у зависности од тога да ли користимо меморију искуства и фиксне Q-мете.

8 Закључак

Циљ овог матурског рада био је да теоријски обради учење са појачавањем. Фокус није био толико на конкретним применама, колико на схватању генералних идеја и алгоритама. Почели смо од решавања потпуно познатих окружења и редом смо усложњавали проблем. На крају смо видели како можемо скалирати научене методе на реалне примере и упознали се са једним скорашњим успешним моделом.

Последњих пар година настао је велики број надоградњи на DQN модел. Могућност за евентуални даљи рад су њихова обрада и поређење. Такође, изостављена је имплементација обрађених алгоритама.

Током израде овог матурског рада, нисам успео да нађем било какву литературу на српском језику. Идеје у овом матурском раду су уведене од нуле и уз мноштво примера, па сматрам да он може да послужи као квалитетна почетна тачка сваком ентузијасти. Такође сматрам да је обрађено знање довољно за даљи самосталан рад и разумевање најсавременијих алгоритама.

На самом крају желео бих да изразим захвалност:

- **Петру Величковићу** – мом ментору, који ми је и поред великог броја обавеза значајно помогао при изради овог рада низом сугестија, предлогом теме и упућивањем на одговарајућу литаратуру. Такође, као једном од оснивача Недеље информатике, која ме је упозанала са великим бројем до тада невиђених области рачунарских наука и заинтересовала за машинско учење.
- **Јелени Хаџи-Пурић** – мом ментору, чија ме безгранична енергија увек мотивише.
- **Филипу Марићу** – мом професору информатике из средње школе, који увек предаје на забаван начин, уз мноштво инспиративних примера.
- **Момчилу Топаловићу и Алекси Милисављевићу** – другарима из школе, захваљујући којима сам значајно напредовао на пољу такмичарског програмирања.

9 Литература

[1] David Silver, *Introduction to Reinforcement Learning*. Доступно на: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf`. [Прегледано 28.5.2017].

[2] David Silver, *Markov Decision Processes*. Доступно на: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf`link. [Прегледано 28.5.2017].

[3] David Silver, *Planning by Dynamic Programming* Доступно на: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/DP.pdf`. [Прегледано 28.5.2017].

[4] David Silver, *Model-Free Prediction* Доступно на: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MC-TD.pdf`. [Прегледано 28.5.2017].

[5] David Silver, *Model-Free Control* Доступно на: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/control.pdf`. [Прегледано 28.5.2017].

[6] David Silver, *Value Function Approximation* Доступно на: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf`. [Прегледано 28.5.2017].

[7] Richard S. Sutton, Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998

[8] Петар Величковић, *Учење са појачавањем*, 2018. Доступно на: `http://www.csnedelja.mg.edu.rs/static/resources/v4.0/sre1_ucenje_pv.pdf`. [Прегледано 28.5.2017.]

[9] Петар Величковић, *Reinforcement learning fundamentals*, 2018. Доступно на: `http://www.cl.cam.ac.uk/~pv273/slides/RLBasics.pdf`. [Прегледано 28.5.2017.]

[10] Петар Величковић, *Неуралне мреже*, 2016. Доступно на: `http://www.csnedelja.mg.edu.rs/static/resources/v3.0/sre1_neuralne_pv.pdf`. [Прегледано 28.5.2017.]

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmill, *Playing Atari with Deep Reinforcement Learning*