Математичка гимназија

Матурски рад
из предмета Програмирање и програмски језици

Увод у појачано учење

Ученик
Михаило Алексић, 4д

Ментор
Станка Матковић

Београд, јун 2019.

Садржај

Предговор

Свако од нас се некад сусрео са неким информацијама (подацима) које није знао шта представљају, чему служе... Упознати смо да у школама, после предавања градива, професори прелазе на израду примера. Такође, шеф ће највредније раднике наградити премијом, а оне мање вредне казнити, опоменути. Иако се чини да између ова два примера као и њихова повезаност са темом у наслову немају много смисла, они имају. Њих сам навео као алгоритме које надређени (професор, послодавалац...) користе како би одређену групу мотивисали или демотивисали на неку радњу. Делује да је ово тема коју би изучавала социјална психологија, али итекако постоје аналогије са данашњим програмирањем и његовим разумевањем.

У циљу већих прихода или већег утицаја, велике компаније или појединци, покушавају да одреде како ће се циљна група понашати по неким променама. Они ће пробати да направе алгоритам што бољи по њих тј. алгоритам који ће утицати на људе да се понашају сходно оном што одговара „програмерима“ тог алгоритма. С обзиром да још нисмо у могућности да у целости израчунамо будућност, да директно утичемо на избор целокупне популације, постаје јасно да је овакав систем по којем бисмо одредили ко ће и шта урадити немогуће направити. Ту на сцену ступа вештачка интелигенција (artificial intelligence) и машинско учење (machine learning).

Увод

2.1 Машинско учење

“Машинско учење је научно проучавање алгоритама и статистичких модела које компјутерски системи користе како би ефектно извршили одређени задатак, не користећи експлицитне наредбе, већ се ослањајући на неке њима својствене правилности, моделе користећи задате податке”, је реченица са *Википедије*, сирова дефиниција, док бих се ја још једном искористио аналогијом из „стварног“ живота, ради бољег разумевања.

Машинско учење се може повезати са рударењем, у овом случају информација. Ближе, велике количине информација асоциране са нехомогеним стенама, камењем, земљом, разбијене дају тражене руде, у нашем случају, групе података са сличним особинама које ће помоћи нама, односно, компјутеру, да предвидимо даље понашање система и тиме помогнемо у било ком виду, било то препознавање *спам* порука у вашем сандучету, обезбеђивање што прецизније обраде података научницима, до што прецизнијих медицинских анализа.

Ова информатичка дисциплина, иако једна од најмлађих (започета првом половином 20. века, своју наглу експазнију достигла је у његовој другој половини), данас је јако распрострањена и као таква има велики број начина имплементација решења на исти проблем, мање или више ефектних у зависности на захтеве истог. Најпопуларније две подобласти су *надгледано* и *ненадгледано* учење (supervised и unsupervised learning) и оне се највише асоцирају са представом човека о машинском учењу. Једна мање популарна подобласт је и тема овог рада, појачано учење (reinforcement learning), која иако веома слична наведеним има итекако више разлика и непоклапања са њима. У наставку рада посветићемо се опису овог учења и дубљом анализом.

2.2 Појачано учење

Сам назив, појачано учење, не даје никакав предзнак о томе шта би била тема тог приступа. *Обучение с подкреплением* је превод reinforcement learning на руски језик и грубо би преведен на српски значио „учење са поткрепљењем“, што би било ближе опису функционисања појачаног учења него што бисмо закључили из наслова. Једна од најједноставнијих, али и прилично тачна, аналогија појачаног учења са реалним светом би била интеракција живих бића са околином, промене њихових акција у односу на реакције природе на прошле акције тј. одабир што оптималнијег скупа „понашања “у дужем временском периоду ради што квалитетнијег начина живота, дужег времена преживљавања. У нашем свету такође постоје казне и награде, слично као у решавању проблема појачаног учења, а о чему ћемо касније причати.

За разумевање даљег текста неопхдоно је основно знање математичке алгебре, вероватноће и неких мање познатих математичких дисциплина. У њему ћемо пробати да што боље разумемо функционисање овог информатичког модела претежно у дискретним системима, док ћемо истраживање континуалних система оставити читаоцима са већим знањем горенаведених области и интеграбилног рачуна.

Основни принципи функционисања појачаног учења

3.1 Поставка проблема

Када нам на памет падне идеја о учењу, вероватно је сви први асоцирамо са интеракцијом са природом и учењу из исте. Било то да ћемо се испећи уколико додирнемо врућу ринглу или изгубити контролу над возилом ако не држимо руке на волану, јасно је да ће последице наших перцепција на реакције природе, као нашег окружења, довести до промене нашег понашања. Та чињеница навела је информатичке научнике да и учење програмских система поистовети са природним учењем. Основу појачаног учења чини интеракција и све што произилази из ње.

3.2 Основне компоненте

Као основне елементе појачаног учења поставићемо агента и окружење. Агент је синоним за човека, змијицу, док је окружење све око агента (а није исти) што интереагује са њим, кухиња, 3x3 табла, респективно. Такође, неопхдни елементи су циљ, наградни сигнал, политика, вредносна функција и произвољно, модел.

Циљ би био јасно дефинисан објектив и обавезно непроменљив, који агент покушава да испуни прогресивним скупом функција, а у интеракцији са окружењем.

Наградни сигнал је реакција окружења у односно на његово тренутно стање и извршену акцију агента.

Политика понашања је одабрани систем по којем ће агент тј. програм нама дат у руке, бирати акције, а научена је из претходних искустава, била она експлоративног или експлоативног типа, које ћемо у даљем тексту објаснити.

Модел је најкомплекснији елемент појачаног учење. Објашњава се као модел окружења и промена истог у односу на предузете акције. На основу њега, агент би лакше могао да коригује своје понашање тако да оно буде квалитетније.

3.2.1 Експлоација и експлорација

Експлоација и експлорација су као што им значења говоре: проширење познатог на најбољи нама познат начин и истраживање непознатог, редом. Слична значења имају и као компоненте појачаног учења и лепо укомпоновани га чине.

Експлоација је понашање агента такво да на крају неког временског интервал изађе у најповољнијој ситуацији и са најбољом наградом. Уско је повезана са надгледаним учењем и неопходан је део у појачаном учењу. Из прве реченице помислило би се да је ово и једини део учења с обиром да ћемо се сложити да ће на крају највећа награда бити оно чему агент тежи.

Експлорација је оно што појачано учење раздваја од осталог машинског учења. Оно је понашање које ће од могућих акција бира ону која можда није најбоља, рачунајући на то да ће њеним одабиром допринети бољој позицији агента у будућности, од оне у којој би се он обрео да је користио само експлоативни метод.

Заједно, ова два метода, добро бираних када ћемо који користити, чине појачано учење јако успешним и све више бираним када су у питању ситуације сличне реалном свету тј. такве да се окружење непредиктивно мења и реагује са агентом, бићем...

3.3 Икс-Окс

Генералну идеју појачаног учења и разлике ње са осталим приступима илустроваћемо на примеру једноставне игре, *Икс-окс*. Знајући да је ово игра за коју постоји стратегија против које не може бити победника, те да би ако је као такву посматрамо ово би бесмислен пример, сматраћемо да противник може да лоше одигра тј. да његов следећи потез са неком вероватноћом води ближе губитку партије, него победи.

> Икс-Окс је игра у којој два играча наизменично постављају X и O почевши од X. Игра се на табли 3x3 и победник је онај који свој знак успе да стави у један цео ред, целу колону или дијагонално три пута тј. да у истој нема ниједног противничког знака. Ако је табла цела попуњена, а ситуација из претходне реченице се није десила , игра се сматра нерешеном.

Како ми не знамо конкретан модел по којем противник игра, опција решавања овог проблема помоћу метода из теорије игара је немогућа. Такође, динамичко програмирање је овде неупотребљиво из разлога што је потребно да унесемо комплетне податке о супарнику, које немамо. У оба метода бисмо могли да направимо мање детаљан проблем, урачунамо несавршености са неком вероватноћом и да очекујемо што бољу апроксимацију понашања играча са друге стране табле, као и наших потеза у сваком моменту.

Ево како би ио проблем могао да се реши користећи вредносне функције.

3.3.1 Решавање помоћу вредносних функција

Прво ћемо направити табелу која ће за свако могуће стање таблице *Икс-Окс* имати вредност вероватноће да из тог стања ми изађемо као победници. Број придружен неком стању, зваћемо вредност стања. Стање А сматраћемо бољим од стања Б, уколико стање А има већу вредност стања од стања Б. Почевши од тога да ми играмо X, свим стањима где имамо три добро постављена знака (таквих да доносе победу играчу који их је одиграо) доделићемо вредност 1, јер је вероватноћа да ћемо после тог стања победити стопроцентна. У супротном случају тј. ако имамо три добро постављена O, таква стања имаће вредност 0, јер из њих не можемо добити партију. Остала стања ће имати вредност 0.5 јер је толика математичка вероватноћа да ћемо из њих победити.

Како бисмо бирали потезе, посматрамо стања у која можемо стићи тим потезом. Најчешће ћемо бирати халапљиво (кратковремески најбоља), али ћемо и истраживати, бирати она која од понуђених нису најбоља.

Док играмо, мењамо вредности оних стања у којима се налазимо и то на начин:

$$V(s) \longleftarrow V(s) + \alpha\left[V(s) - V(s')\right]$$

где је s стање у којем се тренутно налазимо, $V(x)$ вредносна функција стања x, s' стање после потеза, а α дискретни параметар који утиче на брзину учења. Понављање оваквих потеза довољни број пута довешће на крају до жељеног понашања агента.

3.3.2 Закључак

Агент у овој игри бисмо били ми, окружење-противник, циљ-победа. Рачунајући и експлоративне потезе видимо корелацију са раније наведим описом појачаног учења. Такође, на овом једноставном примеру видимо предности овог приступа у односу на неке друге, као и да створимо неадекватну представу да је јако лимитиран нпр. ограничен на искључиво дискретне случајеве. Овде имамо као окружење само нашег супарника, док је код појачаног учења оно много веће или чак бесконачно.

3.4 Вишеручни бандит

Проблем вишеручног бандита је нешто сложенији од претходног и мање познат, али ће помоћи у схватању појачаног учења и значаја експлорације у њему.

3.4.1 Поставка проблема

> Вишеручни бандит је проблем где смо 1000 пута суочени са n-избора при чему после избора неког од тих n добијамо награду за исти по неком систему вероватноћа који зависи од избора.

Ово је оригинална дефиниција овог проблема, који се данас глобализује на више начина и друге проблеме, док ћемо се ми овде бавити само једноставнијим случајем (првобитним).

3.4.2 Решавање проблема

Очекивану награду за избор n-ручног бандита назваћемо вредност те акције. Ако бисмо знали вредности свих акција било би тривијално изабрати најбоље изборе, тако што бисмо сваки пут бирали акцију која би нам донела највећу награду и тако максимизовати коначни износ.

Халапљива акција ће бити она која од n тренутно понуђених има највећу вредност. Бирање халапљиве акције звaћемо експлоацијом знања о вредностима акција. Бирање оних са мањим вредностима од максималне звaћемо експлорацијом знања (истраживање), зато јер нам омогућује да евентуално на већем узорку достигнемо жељенију укупну вредност награда. Код експлоације ће награда бити већа у мањем броју корака, док ће код експлорације она бити већа на дуже стазе, јер ћемо њом открити неке повољније акције од оних које добијамо халапљивим путем. С обзиром да ниједан од ова два приступа није савршен, потребно их је балансирати, користити тако да „бандит “на крају буде најзадовољнији.

Метод коришћењем вредносних функција

Ако вредност акције a дефинишемо као $q(a)$, процењену вредност акције у t-ом потезу као $Q_t(a)$, број пута колико смо је изабрали до t-ог корака са $N_t(a)$, одговарајуће награде са R_1, R_2,..., $R_{N_t(a)}$, онда ће процењена вредност бити:

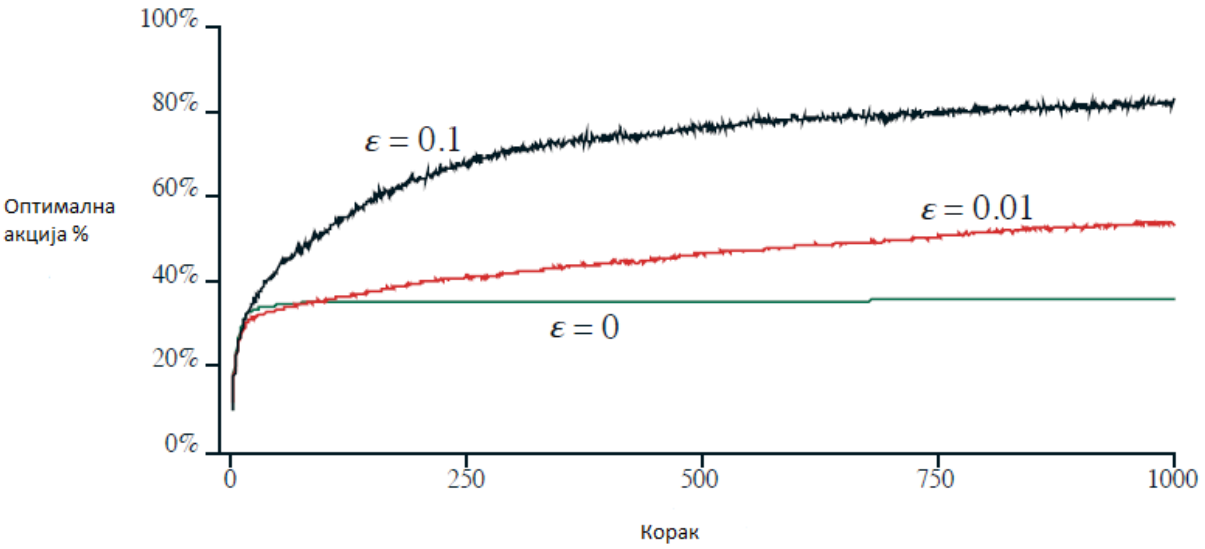
$$Q_t(a) = \frac{R_1 + R_2 + ... + R_{N_t(a)}}{N_t(a)}. \tag{3.1}$$

Ако је $N_t(a) = 0$ онда дефинишемо $Q_t(a)$ као почетну вредност, рецимо $Q_1(a) = 0$. Кад $N_t(a) \longrightarrow \infty$, $Q_t(a)$ конвергира $q(a)$, по закону великих бројева.

Најједноставији метод којим можемо да бирамо акције је да у сваком кораку бирамо ону са највећом процењеном вредношћу за тај корак. Овај начин се неће показати ефектним, па ћемо га убрзати ϵ-методом. ϵ-метод се од халапљивог разликује по томе што са неком мало вероватноћом ϵ повремено уместо оних максималних вредносних функција бира насумичне.

Како бисмо боље разумели побољшања која доноси ова промена, представићемо следећи проблем:

Имамо скуп 2000 насумично генерисаних n-ручних бандита , где је $n = 10$. За сваког бандита, вредносне функције $q(a)$, $= 1, 2, ...10$, биране су по нормалној (Гаусовој) расподели са очекивањем $\mathbf{E}(x) = 0$ и дисперзијом $\mathbf{D}(x) = 0$. Овај експеримент ћемо звати 10-ручни тестер.



Слика 3.1: График ефикасности халапљивог метода и два ϵ-метода

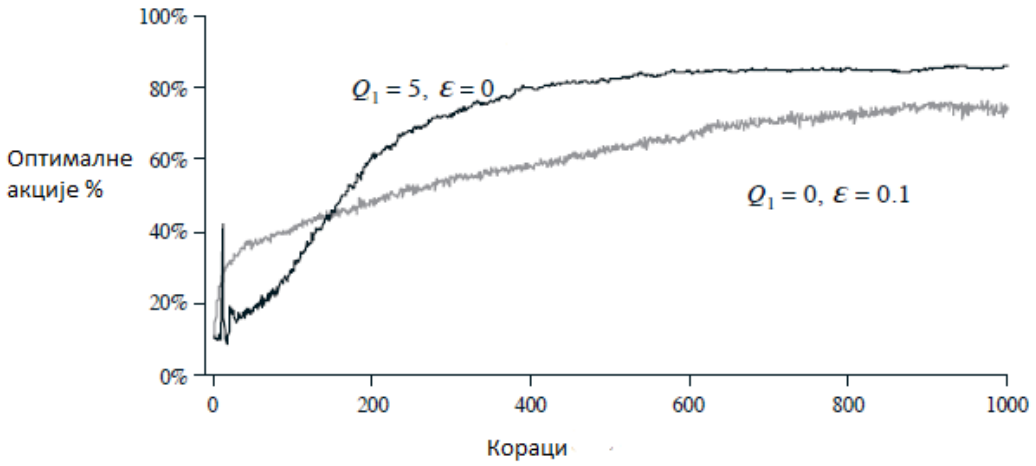
На горњој слици представљен је график 10-ручног тестера тако да је на доњој оси представљен број корака, док је на горњој оси број предузетих оптималних акција изражен у процентима. Видимо да је халапљиви метод ($\epsilon = 0$) лошији од оптимизованијих тј. да ће број оптималних акција код истог бити скоро константан и једнак 40% док код ϵ-метода где $\epsilon = 0.1$ достиже и број изнад 80%. Ако бисмо повећали број генерисаних бандита, испоставиће се да ће метод који користи $\epsilon = 0.01$ бити најефективнији од три са слике (ње-

гов график најбрже расте када посматрамо већи број бандита), што у овом проблему није случај.

Закључујемо да у зависности од тестера различити метод бива најкориснији, што имплицира да мењањем ϵ током експеримента можемо да постигнемо пожељнији резултат.

Оптимистичне почетне вредности

Све вредносне функције без сумње зависе од базичне, којој смо ми у горњем делу текста доделили вредност $Q_1(a) = 0$. Ако бисмо ту вредност за сваку акцију поставили на $+5$ знатно бисмо убрзали процес учења. Агент би за сваку изабрану акцију после иницијалне добио вредносну функцију која је доста мања од почетне што ће га навести на то да се пребацује на друге. Колико је овај метод ефикасан видимо на слици испод, где ће због промене прве вредности функције график где је $Q_1(a) = +5$ брже расти од оног са $\epsilon = 0.01$, који се у претходном делу показао најбољим.



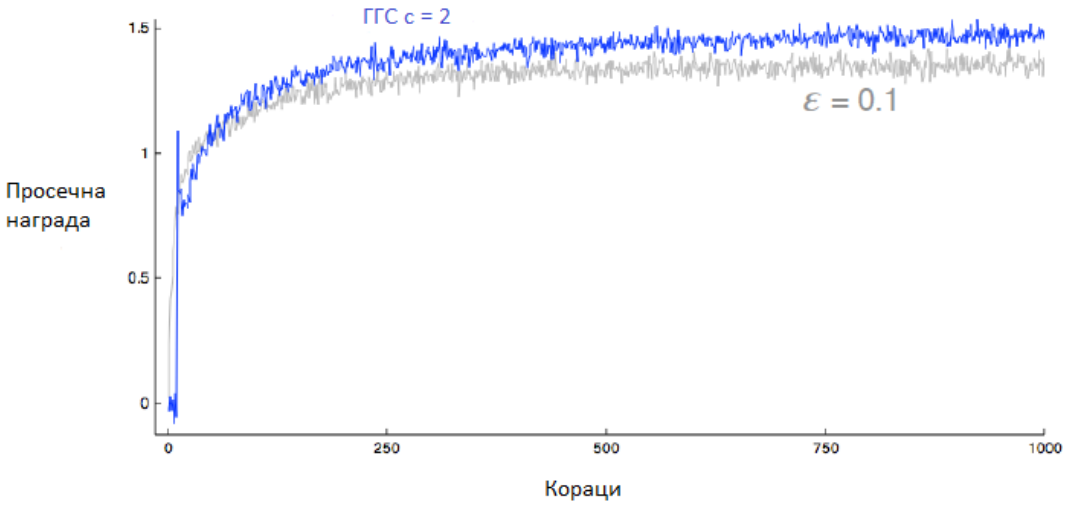
Слика 3.2: График ефикасности ϵ-метода и метода оптимистичне почетне вредности

Горња граница сигурности

Увођењем ϵ у решавање проблема вишеручног бандита добили смо експлорацију. То што овај алгоритам тотално насумично бира акције различите од оне експлоатичне је нешто што можемо додатно поправити. Бираћемо акцију са највећим $F(a,t)$ где:

$$F(a,t) = Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \tag{3.2}$$

Ако $N_t(a) = 0$ важиће да за исте t, a, $F(a,t) \longrightarrow \infty$ што говори да ће она бити изабрана следећа. Ако је изабрана акција , повећаће се $N_t(a)$ тј. смањиће се $F(a,t)$ и то спушта шансу да ћемо њу на даље поново изабрати. Такође, уколико је не изаберемо разломак ће се повећати, тиме и $F(a,t)$. c из једначине 3.2 представља ниво сигурности и подесив је у односу на проблем. На графику испод се уочава побољшање добијено методом горње границе сигурности (ГГС) у односу на ϵ метод.



Слика 3.3: График ефикасности ϵ-метода у односу на метод који користи ГГС

На доњој оси налази се број корака, а на горњој оси просечна награда у неком корака. Разлика између графика ГГС-а и ϵ-метода није велика, али постоји.

3.4.3 Нестационаран вишеручни бандит

Сва решења која смо посматрали у 3.4.2 односе се на проблем када се бандит не мења, стационаран је. Ако се он ипак мења, што је ближе уопштености појачаног учења, постоје разлике са већ анализираним проблемом непроменљивог бандита. Те разлике се уочавају у томе да ће награде из ближе будућности бити повољније него оне што следе касније.

Нека је код стационарног проблем Q_k очекивање за k-ту награду, њу ћемо рачунати као просек претходних $k-1$. Тај просек, заједно са k-том наградом, R_k, даће просек k награда:

$$Q_{k+1} = Q_k + \frac{1}{k}[R_k - Q_k]. \tag{3.3}$$

У нестационарном проблему $\frac{1}{k}$ заменићемо са α :

$$Q_{k+1} = Q_k + \alpha[R_k - Q_k] \tag{3.4}$$

и то ће рачуном довести до

$$Q_{k+1} = (1-\alpha)^k Q_1 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} R_i \tag{3.5}$$

Из једначине 3.5 закључујемо да очекивана вредност највише зависи од почетне вредности и оних ближих награда. Оне су и даље такође урачунате, али не можемо да будемо сигурни за њих као за оне на које ћемо раније наићи зато ће и мање утицати на вредност Q_{k+1} из једначине са почетка реченице.

3.4.4 Закључак

Идеја представљања проблема вишеручног бандита била је да читаоцу приближи начин рада појачаног учења кроз оптимизације најбазичније идеје (халапљиве). Експлорација, експлотација, дискретни параметар, нестационаран систем су све ствари са којима ћемо се сретати у наставку рада.

3.5 Марковљев процес одлучивања

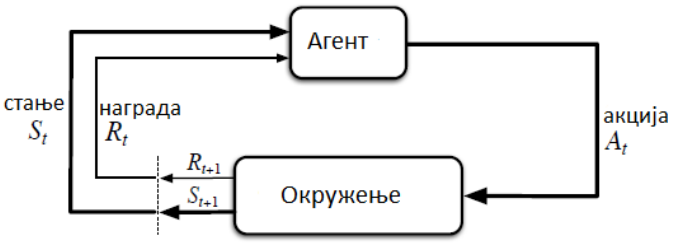
У овом делу уводимо проблем, Марковљев процес одлучивања (МПО), који дефинише поље појачаног учења и сваки метод који буде решавао овај проблем. Пробаћемо да кроз њега разумемо природу појачаног учења, увођењем кључних математичких елемената, давањем примера апликација где се користи појачано учење.

3.5.1 Основни елементи МПО

Као што смо рекли на почетку рада и из скоро тоталне аналогије у дефинисању МПО-а и појачаног учења знамо да су њихови најосновнији елементи агент и окружење. Такође, основу чине циљ и награде. Агент је оно што испуњава задатак, учи, интерагује са окружењем. Примера ради, агент у МПО-у неће бити човек, а окружење природа. Оваква аналогија није добра јер у стварно животу, природа утиче на човека, мења га, што код МПО није дозвољено. У том случају, руке би утицале на чуло мириса, или, биле би њихово окружења. Агент мора да буде униформан и зато је граница између ове две компоненте неретко различита него што природа налаже.

Основна ствар коју ћемо посматрати и коју смо посматрали је интеракција између агента и окружења, како функционише, како можемо на њу да утичемо, шта из ње произилази итд.

Поделимо време интеракције на дискретне интервале нумерисане, $t = 0, 1, 2, ...$ Нека је $\mathcal{S}$ скуп стања у којима може окружење да се налази и $S_t \in \mathcal{S}$, стање у којем се оно налази у моменту t. $\mathcal{A}(S_t)$ скуп акција могућих из стања S_t і $A_t \in \mathcal{A}(S_t)$, акција селектована на основу стања. У следећем кораку, агент ће добити награду $R_{t+1} \in \mathcal{R}$ и наћи се у стању S_{t+1}. Наравно, није неопходно да интервали у којима се дешавају акције и промене стања те давање награда буду дискретни, већ систем може бити континуалан, чиме се ми нећемо бавити. За континуалне системе је неопходно шире математичко знање, а представа о раду МПО-а се довољно јасно може објаснити и кроз дискретни систем. Дијаграм испод ближе појаснiће ове односе.



Слика 3.4: Дијаграм функционисања дискретног МПО-а

Политика $\pi_t(a|s)$ је вероватноћа да изаберемо акцију такву да $A_t = a$ у стању које је $S_t = s$. Агент ће своју политику мењати сходно искуству стеченом у прошлости.

Циљ је оно што агент тежи да достигне и он и награде су међусобно условљени. Награде су дефинисане на основу природе циља, док ће циљ агента тако постати максимизовање награда.

Пример 1. ***Рециклирајући робот*** *Посао покретног робота је да сакупља бачене лименке. Има сензоре за детектовање лименки, руке што убацују лименке у канту закачену за њега и ради на пуњиву батерију. Алгоритам којим одлучује шта ради у неком тренутку функционише по систему појачаног учења. Агент бира између тога да тражи за новом лименком (1), стане и чека да неко дође и сам убаци лименку, не трошећи батерију (2), врати се до базе и тамо напуни батерију (3). Награде су позитивне кад је у робота убачена лименка, а јако негативне ако му се испразни. Неопходно је да се ове три одлуке дешавају периодично или сваки пут када неки сигнал стигне до робота, нова лименка детектована сензором, пражњење батерије... У овом примеру агент није цео робот.*

Комплетније ћемо се овим примером позабавити касније, како он ради детаљније, када бира тренутак када ће да се врати у базу, стане, тражи.

3.5.2 Повраћај

Сада ћемо да формализујемо оно о чему смо информативно причали. Речено је да агент тежи да максимизује добијене награде током неког временског периода.

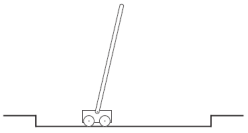
Глобално, агент ће тежити да максимизује *очекивани повраћај*, где је повраћај G_t дефинисан као функција зависна од неког скупа награда. У најједноставнијем случају, може бити представљен као:

$$G_t = \sum_{i=t+1}^{T} R_i. \tag{3.6}$$

Горе су t тренутак у ком тражимо повраћај, T крајњи тренутак, R_i награда у моменту i.

Још један елемент је битан код повраћаја, а то је *снижење*. Повраћаје са снижењем зваћемо *снижени повраћај* и дефинишемо га:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1}, \tag{3.7}$$



Слика 3.5: Балансирајући штап

где је γ дискретни параметар, важи $0 \leq \gamma \leq 1$, и зовемо га *брзина снижавања*.

Уколико је низ R_x ограничен и пошто важи $\gamma < 1$, бесконачна сума из 3.6 је ограничена, што је нама било неопходно да бисмо могли да бирамо максимуме.

Још, ако је $\gamma = 0$, агент ће бити халапљив и неће марити за будуће награде. Ако је $\gamma = 1$, агент ће награде које су нам далеке поистоветити по значају са оним јаснијим, ближим, што такође није добро.

Пример 2. ***Балансирање шипке*** *Поента овог примера је још једна илустрација појачаног учења. Циљ у овом проблему је да искористимо силе природе и померајући ауто на*

коме се налази зглобно закачена шипка по хоризонталној оси тако да она не падне. Уколико шипка падне, враћа се у позицију у којој је нормална на осу кретања возила. Награду (+1) треба давати ако у моменту провере позиције шипке она није пала. Провере треба вршити у константним временским интервалима. Ако, пак, овај систем посмтрамо континуално, сваки пут када шипка падне треба дати награду -1, у осталим случајевима нема награде. Повраћај би био одређен са $-\gamma^K$, где је K број пута када шипка није пала пре свог пада (оног који посматрамо). У оба случаја награда ће бити максимизована балансирањем штапа у вертикалном положају, што смо и хтели (Слика 3.5).

Код сниженог повраћаја видимо да је у дефиницији G_t-а горња граница суме недефинисана, бесконачна. Сада ћемо ту формулу предефинисати као:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}. \tag{3.8}$$

Представу G_t-а из формуле 3.8 користимо ради представљања паралела између континуалног и епизодичног проблема где може бити $\gamma = 1$ за први, а $T = \infty$ за други, али не оба (због конвергенције).

3.5.3 Марковљеви елементи

Када бисмо посматрали агента који игра блекџек, од њега не бисмо могли да очекујемо да зна карте које следе, од брокера која ће акција када „скочити“ или „пасти“. То нам говори да не можемо знати све сигнале стања и да треба исфилтрирати оне битне. Сви сигнали стања који дају корисне информације о окружењу зваћемо *Марковљевим* или да су једни од *Марковљевих елемената*. Они такође доносе информације које су независне од стања осим прошлог, што је јако битно. Боље ћемо разумети круцијалност Марковљевих елемената преко математичке дефиниције.

Најопштији случај реаговања окружења у моменту $t + 1$ на акцију учињену у моменту t је да исти зависи од свега што му је претходило:

$$Pr\{R_{t+1} = r, S_{t+1} = s|S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t\} \tag{3.9}$$

У формули 3.9 посматрамо све могуће награде r, такође, сва могућа следећа стања s' и све могуће информације из прошлости $S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t$, док она сама представља вероватноћу да на основу тих информација, награда у $t+1$ буде баш r, а стање s'.

Ако је окружење један од Марковљевих елемената, његова следећа стања и награде можемо закључити из вероватноће која зависи само од стања и акције предузете у том стању у претходном тренутко и пишемо:

$$p(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s'|S_t, A_t\} \tag{3.10}$$

Марковљеве елементе је очигледно јако лакше анализирати него остале, из разлога што зависе само од информација у прошлом моменту, док ће они који нису Марковљеви елементи зависити из свега прошлог, што може бити велики број стања, акција, награда. Зато је лакше да сваки стање који није овакав елемент, апроксимирамо тако да га можемо посматрати као да јесте, због једноставности.

Пример 3. ***Балансирајући штап*** *Још једном ћемо споменути пример из горњег дела текста. Како бисмо одредили где би возило требало да се креће у следећем моменту, ми*

можемо посматрати кретање од самог почетка и све информације о њему. Такав приступ би био неоптималан па ћемо посматрати само малопређашње стање и где је ауто тада био, која му је била брзина, позицију штапа, његово угаоно убрзање. У идеалнизованом систему, ове информације са још неким минорнијим биле би довољне да проценимо даље кретање оба објекта. С обзиром да не можемо да инструменте којима меримо направимо идеалним и да нећемо рачунати температуру тачкова, трење у зглобу где је штап везан за возило, овакав систем можемо апроксимизовати идеалним.

3.5.4 Коначни Марковљев процес одлучивања

На почетку овог дела рада рекли смо шта је Марковљев процес одлучивања и која би његова улога требало да буде. Коначни Марковљеви процеси одлучивања су најближи појачаном учењу и њихово разумевање би имплицирало и разумевање 90% појачаног учења.

Одређени коначни МПО дефинисан је претходним стањем и акцијом учињеном у том стању. Нека су они дати као s и a, а потенцијалне награда и стање као r и s'. Онда:

$$p(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a\} \tag{3.11}$$

и баш ова вероватноћа спецификује динамику коначног МПО-а. Сва даља теорија у наставку биће око оваквих МПО-а.

Из 3.11 можемо извући сличну формулу за скоро сваку информацију о окружењу, рецимо награди:

$$r(s, a) = \mathbf{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathbf{R}} r \sum_{s'\in\mathbf{S}} p(s', r|s, a). \tag{3.12}$$

Испод појаснићемо функционисање МПО-а једним једноставним примером.

Пример 4. ***Рециклирајући робот МПО*** *Рециклирајући робот из Примера 1. лако може послужити као пример МПО-а његовим упрошћавањем и додавањем неких детаља. У споменутом примеру већ смо рекли да ће робот у екзактно дефинисаним тренуцима да бира између тога да тражи за новом лименком (1), стане и чека да неко дође и сам убаци лименку, не трошећи батерију (2), врати се до базе и тамо напуни батерију (3). Окружење ради тако да ће највеће награде бити за самостално налажење лименки и да то троши батерију која доводи до ригорозне негативне награде ако се она испразни., док је чекање не празни. Док робот тражи, наравно постоји могућност да се батерија испразни и ту ће морати да сачека док га неко не покупи и премести до „куће“ где би напунио батерију па поново кренуо у потрагу.*

Агент ће своје одлуке доносити апсолутно на основу нивоа батерије. Батерија може бити `скоро пуна` *и* `скоро празна`*. Тако да ће скуп стања бити* $\mathbf{S}$=`{скоро пуна, скоро празна}`*. Дефинишемо потенцијалне одлуке агента - агентове акције, као* `чекај, тражи, напуни`*.*

Ако је ниво батерије `скоро пуна`*, пуњење би било беспотребно те га нећемо убацити у скуп акција из тога стања. Скупови агентових акција у односу на стања су:*

$\mathbf{A}$(`скоро пуна`) = {`тражи`, `чекај`}

$\mathbf{A}$(`скоро празна`) = {`тражи`, `чекај`, `напуни`}

Ако је ниво енергије `скоро пуна`*, робот може без стрепње да тражи за лименкама.*

Вероватноћу да робот из стања са нивоом батерије `скоро пуна` *остане у истом стању док активно тражи звaћемо* α*, а вероватноћу преласка из стања* `скоро пуна` *у стање* `скоро празна` *док такође тражи,* $1 - \alpha$*. Слично, посматрајмо период времена у ком он*

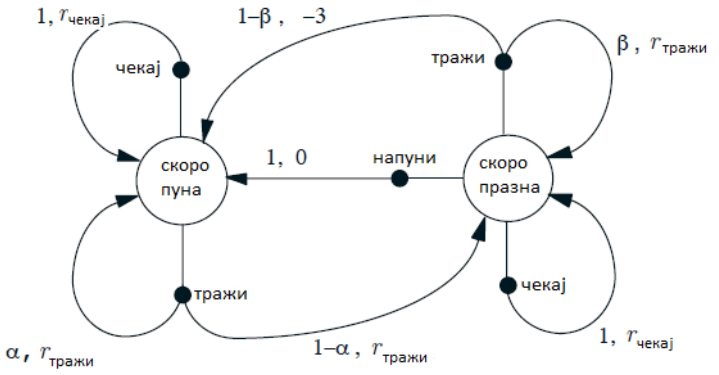
тражи и почевши стањем **скоро пуна** *остане у том стању, вероватноћу за такав случај зваћемо* β*, док ће се тотално пражњење батерије по тражењу за лименкама десити са вероватноћом* $1-\beta$*. У том случају робот ће бити пренешен до базе, вратити се у стање* **скоро пуна** *и то ће бити обележено казном -3.*

Награда за сакупљање лименке при потрази биће $r_{тражи}$*, док ће за сакупљање лименке док стоји бити* $r_{чекај}$*. Рећи ћемо да* $r_{тражи} > r_{чекај}$*. На крају, због једноставности, сматраћемо да је немогуће убацити лименку у празног робота, као и да робот не може да сакупља лименке на путу до базе.*

Сада је овај систем коначан МПО и може се написати преко транзиционих вероватноћа и очекиваних награда.

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
скоро пуна	скоро пуна	тражи	α	$r_{тражи}$
скоро пуна	скоро празна	тражи	$1-\alpha$	$r_{тражи}$
скоро празна	скоро пуна	тражи	$1-\beta$	-3
скоро празна	скоро празна	тражи	β	$r_{тражи}$
скоро пуна	скоро пуна	чекај	1	$r_{чекај}$
скоро пуна	скоро празна	чекај	0	$r_{чекај}$
скоро празна	скоро пуна	чекај	0	$r_{чекај}$
скоро празна	скоро празна	чекај	1	$r_{чекај}$
скоро празна	скоро пуна	напуни	1	0
скоро празна	скоро празна	напуни	0	0

Транзициони граф је користан начин да прикажемо динамику коначног МПО-а. У њему имамо две врсте чворова: чворови стања и чворови акција. За свако стање постоји одговарајући чвор стања, док за сваки пар стање-акција постоји чвор акције, мали црни попуњени кружић на графу, назван именом акције, а који повезује чворове стања. Почевши од стања и чинећи акцију померамо се линијом од чвора стања до чвора акције. Окружење тада реагује транзицијом до следећег чвора стања преко једне од линија које излазе од полазног. Свака линија одговара триплету $r(s,a,s')$*, где је следеће стање* s'*, а тренутно стање* s *и предузета акција* a *и поред сваке линије пишемо транзициону вероватноћу* $p(s'|s,a)$ *и очекивану награду. На Слика 3.6 видимо такав граф, док је изнад овог пасуса табела горенаведених елемената и заједно са те две презентације овог проблема овај пример нас знатно приближава коначним МПО-вима.*



Слика 3.6: Транзициони граф понашања рециклирајућег робота

3.5.5 Вредносне функције у МПО-у

Посматрајући вредносне функције као процене колико је стање у ком се тренутно налазимо добро, очекивано је да ћемо на њих наићи и у овом одељку (стање је „добро“ онолико колико је процењено да ће из тог стања у будућности проистећи награда). С обзиром да награде које ћемо добити зависе од акција које ћемо предузети, вредносне функције ће зависити од одређених политика понашања агента.

Користећи се ознакама из текста и ослањајући се на представу очекиваног сниженог повраћаја G_t према 3.7 вредносну функцију v_π добијену неком политиком понашања π пишемо:

$$v_\pi(s) = \mathbf{E}_\pi[G_t|S_t = s] = \mathbf{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right] \tag{3.13}$$

Слично, дефинисаћемо и *вредносну функцију акције* q_π као:

$$q_\pi(s) = \mathbf{E}_\pi[G_t|S_t = s, A_t = a] = \mathbf{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = a\right] \tag{3.14}$$

Оптимална вредносна функција

Сама вредносна функција функционише тако што представља **E** очекивану вредност насумичне променљиве зависне од политике π у било ком тренутку t и не говори довољно о томе коју акцију треба учинити. Зато дефинишемо *оптималну вредносну функцију* v_* као:

$$v_*(s) = \max_\pi(v_\pi(s)), \tag{3.15}$$

а тако и *оптималну вредносну функцију акције* q_*:

$$q_*(s, a) = \max_\pi(q_\pi(s, a)). \tag{3.16}$$

Овакав запис представља бирање оптималне вредносне функције упоређујући вредносне функције према политикама које користе.

3.5.6 Белманова једнакост

У прошлом одељку ближе смо одредили како би требало да бирамо акције из одређених стања како би стигли до пожељног скупа награда. *Белманова једнакост* представља везу између вредности тренутног стања и будућег стања. Приближиће нам математички принцип по ком бирамо најоптималнију акцију и са својом особином да за v_π има уникатно решење олакшаће систем бирања поменуте акције.

У свом оригиналном облику, изведена од v_π, гласи:

$$v_\pi(s) = \sum_a \pi(a|s)\sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \tag{3.17}$$

Како смо дошли до једнакости 3.17 нећемо анализирати, већ ћемо се позабавити извођењем

Белманове једнакости оптималности:

$$\begin{aligned} v_*(s) &= \max_{a\in\mathbf{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbf{E}_{\pi^*}[G_t|S_t=s,A_t=a] \\ &= \max_a \mathbf{E}_{\pi^*}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t=s,A_t=a\right] \\ &= \max_a \mathbf{E}_{\pi^*}\left[R_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\middle|S_t=s,A_t=a\right] \\ &= \max_a \mathbf{E}[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s,A_t=a] \\ &= \max_{a\in\mathbf{A}(s)}\sum_{s',r} p(s',r|s,a)[r+\gamma v_*(s')] \end{aligned} \tag{3.18}$$

Сада ћемо особине Белманове једнакости оптималности искористити у примеру.

Пример 5. ***Белманова једнакост оптималности на примеру рециклирајућег робота*** *Још једном ћемо искористити већ описивани пример како би у овом случају видели примену Белманове једнакости оптималности. Из једначина добијених поступком 3.18, ми можемо дати тачну Белманову једнакост за рециклирајућег робота. Због лакшег записа, стања* `скоро пуно`, `скоро празно` *и акције* `тражи`, `чекај`, `напуни` *писаћемо као ,* `спу`, `спр`, `т`, `ч`, `н`, *редом.*

Зато што имамо два стања, имаћемо и две Белманове једнакости оптималности. За стање `спр`, *она ће гласити:*

$$\begin{aligned} v_*(\texttt{спу}) &= max\begin{cases} p(\texttt{спу}|\texttt{спу},\texttt{т})[r(\texttt{спу},\texttt{т},\texttt{спу})+\gamma v_*(\texttt{спу})]+p(\texttt{спр}|\texttt{спу},\texttt{т})[r(\texttt{спу},\texttt{т},\texttt{спр})+\gamma v_*(\texttt{спр})], \\ p(\texttt{спу}|\texttt{спу},\texttt{ч})[r(\texttt{спу},\texttt{ч},\texttt{спу})+\gamma v_*(\texttt{спу})]+p(\texttt{спр}|\texttt{спу},\texttt{ч})[r(\texttt{спу},\texttt{ч},\texttt{спр})+\gamma v_*(\texttt{спр})] \end{cases} \\ &= max\begin{cases} \alpha[r_{\texttt{т}}+\gamma v_*(\texttt{спу})]+(1-\alpha)[r_{\texttt{т}}+\gamma v_*(\texttt{спр})], \\ 1[r_{\texttt{ч}}+\gamma v_*(\texttt{спу})]+0[r_{\texttt{ч}}+\gamma v_*(\texttt{спр})] \end{cases} \\ &= max\begin{cases} r_{\texttt{т}}+\gamma[\alpha v_*(\texttt{спу})+(1-\alpha)v_*(\texttt{спр})], \\ r_{\texttt{ч}}+\gamma v_*(\texttt{спу}) \end{cases} \end{aligned}$$

Слично пишемо и за $v_*(\texttt{спр})$*:*

$$v_*(\texttt{спр}) = max\begin{cases} \beta r_{\texttt{т}}-3(1-\beta)+\gamma[(1-\beta)v_*(\texttt{спу})+\beta v_*(\texttt{спр})], \\ r_{\texttt{ч}}+\gamma v_*(\texttt{спр}), \\ \gamma v_*(\texttt{спу}) \end{cases}$$

Уређени пар $(v_*(\texttt{спу}), v_*(\texttt{спр}))$ *је увек јединствен, као што је речено у објашњењу Белманове једнакости, за било које одабране* r, r, α, β, γ, *где* $0\leq\gamma,\alpha,\beta\leq 1$.

Ово би представљало начин на који бисмо бирали акције тако да нам укупан повраћај на крају користећи Белманове једнакости и МПО.

3.5.7 Закључак

Лепота оптималне вредносне акције је та што у сваком стању (сваком кораку), можемо да између њих бирамо „халапљиво“ тј. да се не обазиремо на будуће функције, него на

максимизацију оне тренутне. Још ближе, тај одабир смо приказали Белмановом једнакости, али ипак познато нам је да на примеру шаха, иако игру засновану на искуству играча, још није направљен такав програм (агент) који би могао да победи сваког играча и да тачно процени сваки потез. То је због тога зато што је за рачунање решења Белманове једнакости потребно јако доста меморије и времена, колико нама у данашње време није доступно. Зато се ту, избацивањем неких вероватноћа и разним апроксимацијама постиже довољна тачност у предикцијама које тражимо.

Иако дефиниције појачаног учења и проблем Марковољевог процеса одлучивања нису индентичне, они нераскидиво зависе међусобно и разумевање овог другог итекако чини леп увод у разумевање основних принципа појачаног учења.

3.6 Методе учења

У претходном одељку бавили смо се оптимизацијом информација потребним за учење о што бољој интеракцији агента и окружења (Марков процес одлучивања), те начином избора акција таквих да оне доводе до највећег повраћаја по некој задатој политици понашања, али се нисмо бавили како одабрати такву политику тако да поново њен максимизован повраћај доноси најпожељнији исход. У овом одељку ћемо прокоментарисати најзначајније од таквих метода.

Почећемо од методе динамичким програмирањем, а такође описаћемо и Монте Карлов метод.

Динамичко програмирање

Динамичко програмирање представља скуп алгоритама који решавају проблем појачаног учења када нам је познат комплетан и савршен модел окружења.

Алгоритми који користе *динамичко програмирање* (ДП) су врло лимитирани из више разлога. Да би интеракцију агента и окружења могли да посматрамо кроз оквир динамичког програмирања потребно је да окружење буде перфектно (буде Марковљев елемент, буде тотално предвидљиво...). Осим тог разлога, ДП се сматра лимитираним због своје брзине и неопходном великом меморијом за свој рад. Упркос томе, ови алгоримтми јако су битни као базични за све остале методе, јер ће се они послуживати сличним методама, само ефективнијим и без неопходног услова за савршеност окружења. Генерална идеја динамичког програмирања је да користи вредносне функције описане у секцији 3.5 у потрази за што бољом политиком понашања.

Монте Карлов метод

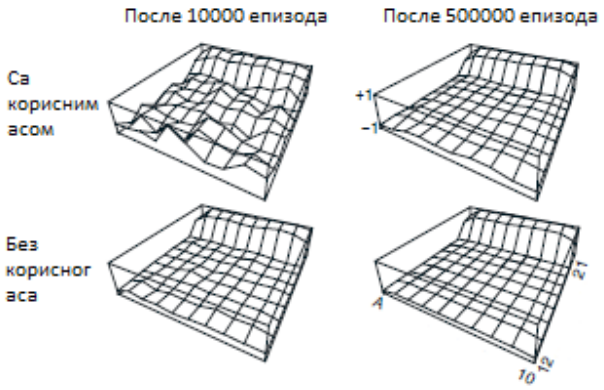
Монте Карлов метод је метод који се заснива на идеји ДП-а само што је јако ефикаснији од истог. Код овог метода није неопходно да знамо све информације о окружењу већ су нам довољне иформације из симулиране интеракције (стања, акције, награде...) те је погоднији због своје једноставности. Пробаћемо да додатно објаснимо разлику између ова два приступа кроз предстојећи пример.

Пример 6. ***Блекџек*** *Блекџек је популарна игра са картама у којој је циљ да вам збир карата подељених од стране дилера буде што ближи броју 21, али не и већи од њега. Игра почиње тако што дилер подели две карте себи и две карте играчу, тако да су обе карте које су код играча окренуте лицем, док је једна од две дилерове, окренута полеђином. Овде ћемо посматрати нешто једноставију верзију игре у којој играч може само да* `настави`

или да `стане` *и играчи не играју један против другог, него је игра усмерена само између крупијеа и играча, појединачно. Играч* `наставља` *све док не одлучи да* `стане` *и тада је на потезу дилер, или не пређе број 21 у збиру својих карата, када је победа моментално противникова. Дилер игра према фиксној стратегији, тако што све док не пређе збир карата од 17* `наставља` *и тада* `стаје`*. Ако премаши 21, победа је играчева док се у супротном гледа удаљеност од траженог збира и тако одређује победник (ако је збир исти, игра се сматра нерешеном).*

Играње блекуек можемо посматрати као епизодични МПО, где је свака епизода једна партија игре. Награде се дају по систему: +1 за победу, 0 за нерешено и -1 за пораз. С обзиром да су игре независне једна од друге нећемо снижавати повраћај већ ћемо сматрати да је $\gamma = 1$*. Акције могу да буду или настављање и заустављање на тренутним картама. Стање зависи у односу на карте које тренутно играч има и у односу на дилерову карту коју види. Агентова одлука о акцији заснивађесе на тренутном збиру карата (12 до 21; у супротном ће сигурно узети још једну карту), карти коју види код дилера (ас до 10; све карте са сликом се сматрају као „десетке“) и на томе да ли аса може рачунати као 11 (ас се може рачунати и као 1 и као 11 при рачунању збира карата у зависности од тога да како више одговара, било то када је играч или дилер на потезу) без да пређе дозвољену границу (таквог аса звaћемо* `корисним`*) или не, што је 2 случаја, што доводи до укупног броја од 200 стања. Здраворазумски је да ћемо стати ако нам је збир 21, док ћемо посматрати политику по којој агент додатно* `стаје` *и када је збир 20 (шансе да је следећа карта ас је много мања него да је било шта друго). Да не би било популарног „бројања“ карата, сматраћемо да у шпилу има неограничено много карата тј. да није класичан са одређеним бројем карата у себи.*

На слици испод видимо колико је случај у ком имамо аса нејаснији од оног када га немамо при мањем броју епизода, док су оба добро апроксимизована после 500000 епизода.



Слика 3.7: Приказ рељефа вредносних функција зависних од стања помоћу Монте Карловог метода

Насупрот овом методу, а иако нам је у овом случају познато цело окружење, метод динамичког програмирања је неупотребљив. Потребно је да из сваког стања одредимо даље награде и вероватноће за дешавање свих опција што је овде јако тешко урадити (споро и комплексно), подлежно је грешкама, док је као што смо видели Монте Карлов метод способан да апроксимира понашање играча у овој не тако компликованој игри.

3.7 Q-учење

У овом раду анализирали смо две методе учења (динамичким програмирањем и Монте Карлова метода), док методу тренутним разлика нисмо споменули.

Метод *тренутних разлика* (ТР) је заснован на идејама Монте-Карловог метода и ДП метода и својом компликованошћу превазилази захтевност у односу на остали део рада. Ради по сличном принципу као два која смо описали, па сам сматрао да ће нешто заинтересованији читаоци лако у референцама наћи књиге и радове из којих ће се моћи нешто више информисати о овом методу.

Иако га нећемо анализирати, у коду који следи у следећем одељку користимо један од делова метода ТР-а, *Q-учење*.

3.7.1 Поставка проблема

Q-учење је још један начин којим унапређујемо понашање агента у интеракцији са окружењем, а ради жељеног исхода. Сматра се неполитичним алгоритмом због тога што се не ослања на неку политику понашања, него учењем прави своју, где насумичним бирањем акција, бира оне које ће се касније испоставити најпогоднијим. Функционише по систему сличном већ наведеним методама, а рад овог типа учења се може најлакше приказати табелом.

Основа Q-учења је дводимензионална табела у којој је једна координата стање, а друга акција (слично као код стање-акција парова у ранијем делу текста). На почетку сва поља табеле биће иницијално постављена на нулу, баш из разлога споменутог на почетку, а то је да се појачано учење заснива на искуству, а не на подацима унетим од стране корисника раније. Мењањем табеле помажемо агенту да изабере најбоље акције. У систему рада биће потребно више епизода како бисмо достигли или се бар довољно приближили споменутом q_*.

Два базична корака у раду Q-учења су :

1. Бирање акције ослањајући се на Q-табелу бирајући ону са највећом вредношћу или бирати насумично
2. Промена садржаја Q-табеле

3.7.2 Принцип мењања Q-табеле

Математички принцип по ком ћемо мењати Q-табелу гласи:

$$Q(S_t, A_t) \longleftarrow (1-\alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, A_t)), \tag{3.19}$$

где је $Q(S_t, A_t)$ вредност коју мењамо, стање S_t је стање у ком се тренутно налазимо, A_t је предузета акција, S_{t+1} је надолазеће стање, R_{t+1} је одговарајућа награда, а γ-ом множимо највећу вредност у табели гледајући по акцији и за дато стање.

Оно што нас више интересује јесу параметри α и γ. α је параметар који одређује колико ће претходна искуства „остарити“, када ће престати да нам буду битна за даље бирање акција и подесив је по жељи корисника и у односу на решавани проблем. γ се односи на будуће награде и њихов значај на тренутно бирање акције (као што смо већ спомињали, γ снижава значај будућих награда у односу на оне ближе нама, баш због непредвидивости система). Уз све ово, ова два параметра нам доносе неопходну експлорацију без које би овај проблем био решаван на скоро халапљив начин.

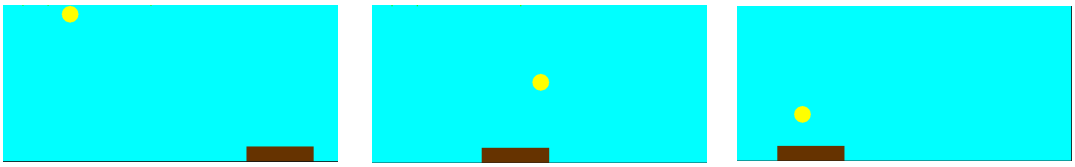
У апликацији чије објашњење кода следи искористићемо знање научено у овом делу.

Апликација "Сакупљач лоптица"

Све ово претходно научено о појачаном учењу искористићемо у овој апликацији. Да би читалац могао да испрати потребно је да има основно знање из програмског језика *Python* и нешто веће генерално програмерско знање, уз неопходно основно знање о појачаном учењу.

4.1 Поставка задатка

У почетном тренутку лоптице почињу да падају са неба константом брзином. Оне падају кроз свих 10000 итерација и појављују се једна по једна, а нестају када буду ухваћене. Циљ агента (корпе) је да научи да их хвата са што већом прецизношћу. Неке од епизода видимо на сликама испод.



Слика 4.1: Случајни моменти у процесу бирања акције

4.2 Увод у решавање задатка

У решавању овог задатка користићемо Q-учење. Поделићемо систем на епизоде, свака епизода почиње почетком падања лоптице док је крај епизоде или хватање лоптице или њено падање на дно. Позитивну награду ћемо агенту доделити за успешно хватање лоптице, док ће она бити негативна за њено нехватање. Акције које може да предузме су да остане на истом месту, помери се лево или да се помери десно. Агент ће бити сама корпа, док ће окружење бити све остало.

4.2.1 Опис библиотека

Осим системских библиотека, које су инсталирањем пајтоновг окружења и оне уграђене, потребно је за покретање ове апликације скинути и библиотеке pygame и numpy.

pygame је библиотека која помаже у креирању мултимедијалних апликација и игрица и значајно олакшава израду ове апликације.

numpy је библиотека која помаже у оном научном делу програмирања у *Python*-у. Обезбеђује нам n-димензионалне низове, лакшу израду функција...

4.3 Анализирање кода

У овом одељку анализираћемо код део по део и коментарисати улогу сваког од тог дела.

4.3.1 Иницијализација

У коду који следи испод иницијализоваћемо класе и елементе задатка.

```
import numpy as np

# definisemo velicine prozora u kojem se desava interakcija

prozorD = 800
prozorV = 400

# neophodne boje neugradjene sistemski

ZUTA = (255, 255, 0)
PLAVA = (0, 255, 255)
BRAON = (102, 51, 0)

# definisanje konstantnih parametara loptica kruzica

krugR = 20
brzina_padanja = prozorV / 10

# definisanje velicina korpe (pravougaonika)

korpaL = 100
korpaG = 365
korpaD = 160
korpaV = 35

# niz sa indeksima unikatnih stanja

QIDic = {}

#Q-tabela

Q = np.zeros([5000, 3])

# klasa Krug sa X i Y koordinatama centra kao parametrima

class Krug:

    def __init__(x, kX, kY):
        x.kX = kX
        x.kY = kY

# klasa Centar sa pravougaonikom i krugom kao parametrima

class Stanje:

    def __init__(x, pu, kg):
        x.pu = pu
        x.kg = kg
```

У горњем коду, поред коментара у истом, мислим да је потребно искоментарисати шта је то QIDic низ.

Једна од особина Q табеле дефинисане у коду је да не можемо узимати индексе као што смо навикли у осталим порграмским језицима. Зато ћемо у видети како да сваки стање-акција пар из табеле уникатно индексирамо и тим индексима напунимо горенаведини низ.

4.3.2 Функције

У овом делу видећемо функције које су коришћене у изради главног дела.

```
# unosenje biblioteka i koda iz fajla inicijalizacija

import random
import pygame as pg
from inicijalizacija import *

# novo stanje je samo promena pozicije korpe i loptice; u odnosu na jednu od tri
    pomenute akcije pravougaonik moze da se krece levo, desno ili da stoji

def novostanjeposleakcije(stanje, akcija):
    prug = None
    # promena pozicije pravougaonika
    if akcija == 2: # 2==desno; 0==stoji; 1==levo
        if stanje.pu.right + stanje.pu.width > prozorD:
            prug = stanje.pu
        else:
            prug = pg.Rect(stanje.pu.left + stanje.pu.width,
                           stanje.pu.top, stanje.pu.width,
                           stanje.pu.height) # Rect(left, top, width, height)
    elif akcija == 1:
        if stanje.pu.left - stanje.pu.width < 0:
            prug = stanje.pu
        else:
            prug = pg.Rect(stanje.pu.left - stanje.pu.width,
                           stanje.pu.top, stanje.pu.width,
                           stanje.pu.height)
    else:
        prug = stanje.pu
        # promena pozicije kruga
    noviKrug = Krug(stanje.kg.kX, stanje.kg.kY + brzina_padanja)
    return Stanje(prug, noviKrug)

# ova funkcija je samo deo prosle, ali sam je napisao zbog lakseg koriscenja u
    daljem kodu

def noviprugposleakcije(pravougaonik, akcija):
    if akcija == 2: # 2==desno; 0==stoji; 1==levo
    # promena pozicije pravougaonika
        if pravougaonik.right + pravougaonik.width > prozorD:
            return pravougaonik
        else:
            return pg.Rect(pravougaonik.left + pravougaonik.width,
                           pravougaonik.top, pravougaonik.width,
                           pravougaonik.height)
    elif akcija == 1:
        if pravougaonik.left - pravougaonik.width < 0:
            return pravougaonik
        else:
            return pg.Rect(pravougaonik.left - pravougaonik.width,
                           pravougaonik.top, pravougaonik.width,
```

```
                            pravougaonik.height)
    else:
        return pravougaonik

# kada u korpu upadne loptica ili ona dotakne dno, potrebno je napraviti novu i to
    na vrhu prozora tako sto cemo je postaviti na nasumicnoj X koordinati, a uz
    vec poznati poluprecnik

def padanjekruga(krugR):
    noviX = 100 - krugR
    ran = random.randint(1, 8)
    noviX *= ran
    return noviX

# racunanje rezultata se radi tako sto ce rezultat biti +1 ako smo lopticu
    uhvatili, a -1 ako nismo

def racunajrezultat(prug, krug):
    if prug.left <= krug.kX <= prug.right:
        return 1
    else:
        return -1

# kao sto smo vec rekli, nemoguce je uzeti indeks neke vrednosti iz Q-tabele, tako
    da cemo svako od unikatnih stanja preko njenog pravougaonika i kruga
    indeksirati u listi QIDic

def klasifikacija(stanje):
    a = stanje.pu.left
    b = int(stanje.kg.kY)
    c = int(str(a) + str(b) + str(stanje.kg.kX))
    if c in QIDic:
        return QIDic[c]
    else:
        if len(QIDic):
            maxi = max(QIDic, key=QIDic.get)
            QIDic[c] = QIDic[maxi] + 1
        else:
            QIDic[c] = 1
    return QIDic[c]

# u funkciji ispod biramo najbolju akciju u odnosu na trenutno stanje i to bas na
    nacin opisan u odeljku 3.7.2

def besteakcija(stanje):
    return np.argmax(Q[klasifikacija(stanje), :])
```

4.3.3 Главни део кода

Сумирани кодови иницијализација и функција заједно са дизајном и делом учења, уз коментаре се налази у коду доле:

```
# unos biblioteka, kao i dela koda sa funkcijama

import sys

from pygame.locals import *
from rad import *

# definisanje parametara okruzenja i agenta, definisanje diskretnih konstanti alfa
    i gama (njih postavlja korisnik prilagodjavajuci njima brzinu ucenja) itd.
```

```
9
10 FPS = 20
11 vremefps = pg.time.Clock()
12
13 pg.init()
14
15 prozor = pg.display.set_mode((prozorD, prozorV))
16 pg.display.set_caption('Skupljac loptica')
17
18 prug = pg.Rect(korpaL, korpaG, korpaD, korpaV)
19
20 skor = 0
21 ukupno = 0
22 nagrada = 0
23 font = pg.font.Font(None, 30)
24
25 procenat = 0
26
27 uci = .85
28 y = .99
29 i = 0
30 krugCentarX = int(padanjekruga(krugR))
31 krugCentarY = 50
32
33 # pokretanje i zaustavljanje programa
34
35 while True:
36     for event in pg.event.get():
37         if event.type == QUIT:
38             pg.quit()
39             sys.exit()
40     prozor.fill(PLAVA)
41
42 #dodavanje odredjene nagrade agentu u odnosu na to da li je ili nije uhvatio
      lopticu, a kako smo opisali u odeljku 4.2
43
44     if krugCentarY >= prozorV - korpaV - krugR:
45         nagrada = racunajrezultat(prug, Krug(krugCentarX, krugCentarY))
46         krugCentarX = int(padanjekruga(krugR))
47         krugCentarY = 50
48     else:
49         nagrada = 0
50         krugCentarY += int(brzina_padanja)
51
52 # promena Q-tabele
53
54     s = Stanje(prug, Krug(krugCentarX, krugCentarY))
55     potez = besteakcija(s)
56     rez = racunajrezultat(s.pu, s.kg)
57     s1 = novostanjeposleakcije(s, potez)
58     Q[klasifikacija(s), potez] += uci * (rez + y
59             * np.max(Q[klasifikacija(s1), :]) - Q[klasifikacija(s),
60             potez])
61
62 # promena pozicija kako pravougaonika, tako i kruga
63
64     prug = noviprugposleakcije(s.pu, potez)
65     krugCentarX = int(s.kg.kX)
66     krugCentarY = int(s.kg.kY)
67     pg.draw.circle(prozor, ZUTA, (krugCentarX, krugCentarY), krugR, 0)
68     pg.draw.rect(prozor, BRAON, prug)
69
```

```
70 # promena skora, ukupnog broja proslih loptica i procenta uspesnosi agenta
71
72     if nagrada == 1:
73         skor += nagrada
74         ukupno += nagrada
75         procenat = skor / ukupno * 100
76     elif nagrada == -1:
77         ukupno -= nagrada
78         procenat = skor / ukupno * 100
79
80 # promena interfejsa
81
82     tekst = font.render('skor: ' + str(skor), True, (10, 250, 40))
83     tekst1 = font.render('ukupno loptica: ' + str(ukupno), True, (10,
84                          250, 40))
85     tekst2 = font.render('procenat:' + str(procenat) + '%', True, (50,
86                          250, 0))
87     prozor.blit(tekst, (prozorD - 600, 10))
88     prozor.blit(tekst1, (prozorD - 790, 10))
89     prozor.blit(tekst2, (prozorD - 450, 10))
90
91     pg.display.update()
92
93 # promena epizode (otkucaj)
94
95     vremefps.tick(FPS)
```

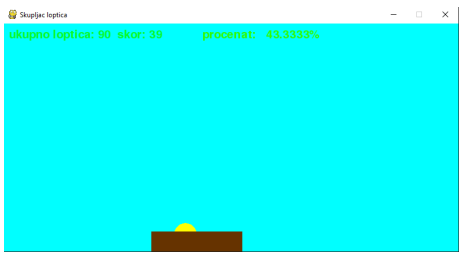
С обзиром на то да смо почетну позицију лоптице изабрали насумично, није неопходно да тако бирамо и почетну позицију корпе, јер је онда (насумичним бирањем лоптице) и само почетно стање насумично, тако да смо базичну позицију корпе фиксирали.

Брзина падања, број епизода, као и вредности дискретних параметара сам сам наместио тако да су мени брзина учења, као и изглед интерфејса, изгледали најбоље.

4.4 Закључак о апликацији „Скупљач лоптица“

С обзиром на то да у раду не можемо дигитално приказати како ова апликација ради, пробаћемо то да дочарамо кроз график ефикасности на узорку од 10000 лоптица.

На Слика 4.2 видимо како би требало да игледа интерфејс апликације у неком неком тренутку.

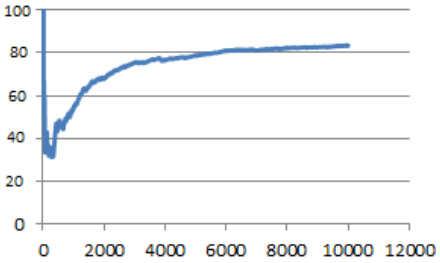


Слика 4.2: Интерфејс *Скупљача лоптица у произвољном тренутку*

Како ефикастост алгоритма можемо посматрати кроз проценат ухваћених лоптица у неком моменту, баш такав график ћемо нацртати: на његовој доњој оси биће укупан број

лоптица које су пале, док ће на горњој оси бити ефикасност изражена у процентима ($\frac{\text{ухваћене}}{\text{укупно}} \cdot 100\%$).

На слици видимо да на почетку имамо висок пик на 100% који траје нешто дуже, што има везе са независношћу случајева (у неком другом случају може се десити да на почетку ухвати мање и тиме би график изгледао другачије). Такође, примећујемо очигледне осцилације до $\sim$ 6000. итерације што нам казује несавршеност агента (до тог момента није научио шта треба да ради). Од $\sim$ 6000. итерације график константно расте, што доказује да је учење било ефикасно. Шумови на том делу графика производ су експлорација, која иако све ређа, и даље постоји.



Слика 4.3: График ефикасности

Овај график ће асимптотски тежити линији која обележава 100%, само што ће се то десити после великог броја итерација.

Кроз ову игрицу успели смо да покажемо велики број ствари научених у овом раду, почевши од експлорације и експлоације, дискретних константи, па преко снижених повраћаја и наравно Q-учења. Иако незахтеван задатак, успео је да нам ове појмове уско повезане са појачаним учењем још више дочара.

Закључак

Појачано учење као асоцијација на модерно програмирање, балон који се надувава и који неће пући, а тек је кренуо да расте. Научно поље чије су тек основе откривене и којем се тешко види крај.

У овом раду ја сам хтео да читаоцима прикажем основе ове информатичке дисциплине и разним примерима покажем да програмирање није толики баук и да је анимозитет који широка маса има према науци генерално, ипак неоправдан. Хтео бих да напоменем да је једина примена примера и паралела са стварним светом у сврси дочаравања и илустрализације неких ствари везаних за појачано учење, јер сам сматрао да ће они бити веза која ће повезивати интуитивне ствари са овим нешто мање интуитивним и тако помоћи разумевање текста, а ни у коју руку подстицање или рекламирање поменутих ствари. Успут, циљ овог рада је да оне који мисле да је машинско, појачано учење, или учење генерално, нешто у шта ће тешко упливати, охрабре да, не само ову научну дисциплину, него и све остале, пригрле и почну да их истражују. Рад ипак није могао да остане на најпростијој основи, из разлога што је математика у њему била неопходна како би се све елементе, од најједноставнијих па до оних компликованих, којих у овом тексту нема много, разумели нешто ближе. Такође се надам да ће та повезаност појачаног учења са вероватноћом, математичком алгебром, теоријом графова и генерално математиком, обе популаризовати кроз потребе једне за другом.

Апликација, чију смо презентацију имали прилику да видимо, није ни у једну руку захтевна, али је као и већина примера описиваних кроз рад, чини ми се одлично објаснила примене неких ствари које на први поглед нису биле очигледне. Преко *Змијице*, *Не љути се човече*, па до сензора на аутомобилима протеже се поље на ком појачано учење има своју примену. Несагледиво је и још расте.

Сматрам да би у школама требало увести наставу колико-толико усмерену ка овим модернијим информатичким дисциплинама јер су оне наша будућност. Управо зато, моје основно знање о појачаном учењу проистекло је са интернет страница и књига, непреведених на наш језик, што је популаран, а врло штетан тренд, не преводити књиге које иду у корак са светским научним развојем.

Неизбежно морам да се захвалим свом ментору и четворогодишњем професору, професорки Станки Матковић, која ме је охрабрила да пишем рад на тему о којој до пре пар месеци ништа нисам знао, а сад тек видим колико не знам о њој и колико хоћу да научим и наравно дала информатичку подлогу која је неизоставна у раду као што је овај и не само у овом раду, него и надам се мом будућем животу.

Такође, захвалност дугујем и свим осталим који су допринели знању и подршци који су били круцијални за завршетак школског образовања и овог рада.

Литература

[1] Reinforcement Learning: An Introduction, Sutton Richard S, Barto, MIT Press, 1998

[2] Learning from Delayed Rewards, Watkins Christopher J.C.H, PhD thesis, King's College, Cambridge, UK

[3] Advances in Reinforcement Learning, Mellouk Abdelhamid, InTech, 2011

[4] Algorithms for Reinforcement Learning, Csaba Szepesvári, Morgan Claypool, 2010

[5] Reinforcement and Systemic Machine Learning for Decision Making, Parag Kulkarni, John Wiley Sons, 2012

[6] Python Reinforcement Learning Projects, Sean Saito, Yang Wenzhuo, Rajalingappaa Shanmugamani, Packt Publishing, 2018

[7] Practical Reinforcement Learning, Farrukh Akhtar, Packt Publishing, 2017

[8] Simple Reinforcement Learning: Q-learning, Andre Violante, Towards Data Science, 2018
`https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56`

[9] Reinforcement learning, Wikipedia
`https://en.wikipedia.org/wiki/Reinforcement_learning`

[10] Q-learning Pyhton example, Iqbal Hasan
`https://github.com/hasanIqbalAnik/q-learning-python-example`