УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ


Огњен Милинковић

# ЛАТЕНТНИ МОДЕЛИ ОКРУЖЕЊА У УЧЕЊУ ПОТКРЕПЉИВАЊЕМ

мастер рад

Београд, 2023.

**Ментор**:

др Младен Николић, ванредни професор
Универзитет у Београду, Математички факултет

**Чланови комисије**:

др Јована Ковачевић, доцент
Универзитет у Београду, Математички факултет

др Александар Картељ, доцент
Универзитет у Београду, Математички факултет

**Датум одбране**: _____

*Породици*

**Наслов мастер рада**: Латентни модели окружења у учењу поткрепљивањем

**Резиме**: Учење поткрепљивањем је једна од најпопуларнијих истраживачких области машинског учења. Поред значаја на пољу истраживања, ова област има разноврсне домене примене, од аутоматског играња игара до аутономне вожње и летења. Алгоритми учења поткрепљивањем су познати по својој нестабилности и по потреби за великим бројем интеракција како би научили задовољавајуће понашање. Један од начина поправљања ових алгоритама је да се поред учења понашања агента, информације које се добијају из интеракција са окружењем паралелно користе за учење експлицитног модела неких релевантних аспеката тог окружења. Тако добијен модел се може користити у учењу агента као јефтинији извор информација о окружењу од самог интераговања са њиме. Латентни модели окружења, који уче апстрактне (латентне) репрезентације стања и/или акција и групишу их тако да слична стања и сличне акције имају сличне репрезентације, представљају активно поље текућег истраживања у учењу поткрепљивањем. У овом раду разматране су две методе за учење латентних модела окружења и анализирани су ефекти које учење на репрезентацијама добијеним из ових модела може имати на тренинг агента учењем поткрепљивањем. Посматрањем различитих примера окружења, приказани су случајеви када репрезентације добијене различитим методама учења латентних модела окружења имају позитиван, односно негативан ефекат на тренинг агента, као и када учење на овим, новодобијеним, репрезентацијама нема значајан ефекат на процес тренинга.

**Кључне речи**: машинско учење, учење поткрепљивањем, учење репрезентације

# Садржај

# Глава 1

# Увод

Учење поткрепљивањем је једна од три основне гране машинског учења и једна је од најактивнијих области тренутног истраживања. Нека од најзвучнијих достигнућа која су постигнута методама учења поткрепљивањем су аутономна вожња, аутономно управљање летелицама и аутономно играње игара као што су го, шах, дота и многе друге. Поред ових достигнућа која су често добијала велику медијску пажњу учење поткрепљивањем је нашло примене и у областима као што су самостално балансирање напона у електричној мрежи, а нашло је и примене у другим гранама машинског учења као што је аутоматско генерисање натписа на сликама.

Формалну дефиницију као и технике решавања проблема учењем поткрепљивањем оставићемо за главу 3, а основну идеју можемо видети у експериментима америчког психолога Б. Ф. Скинера [26]. Уколико неку животињу желимо да научимо да ради одређене задатке (на пример учење пацова да повуку одређену полугу) оно што можемо урадити је давати јој награду када задатак успешно изврши или, са упитним моралним импликацијама, кажњавати их када задатак не испуне. Овим процесом ће животиње, након првобитних случајних радњи, научити који редослед акција доводи до награде, или избегавања казне. У овом процесу „ученик" (животиња) учи да изврши задатак простом тежњом ка максималној награди, без знања о задатку који решава.

Поставка проблема учења поткрепљивањем се не разликује много од овог класичног експеримента. У жељи да направимо програм који успешно решава дати задатак прво дефинишемо *окружење* у коме ће *агент* извршавати *акције*. Окружење може имати различите облике у зависности од конкрет-

ног проблема који се решава, а типични примери су лавиринт, игрица, улице града, рука робота који управља рубиковом коцком и слични. Агент ће бити програм који покушавамо да направимо за решавање задатка, најчешће неуронска мрежа. У сваком кораку агент од окружења добија неко опажање на основу кога доноси одлуку коју акцију жели да предузме. Након извршене акције окружење се мења и агент добија *награду* у виду броја, као и информацију да ли се *епизода интеракција* (на пример, партија игре) завршила.

На основу искуства које ће агент стећи интеракцијама у окружењу, можемо постепено мењати агента, односно ажурирати параметре неуронске мреже која га контролише, у нади да ћемо повећати укупну награду коју агент добија од окружења. Уз претпоставку да прикупљање максималне награде у окружењу уједно и решава постављени задатак, овим изменама доћи ћемо до тог решења. Овај процес измена се назива *тренирање агента*.

Алгоритми тренирања агента су и даље предмет активног истраживања и показало се да приликом тренинга врло често наилазимо на различите потешкоће. Неке од њих ћемо и овде истаћи.

1. Опажања која агент добија од окружења најчешће су необрађени подаци. На пример опажање може бити слика екрана игрице коју покушавамо да играмо. Чак и када се на слици налазе све потребне информације за предузимање одговарајуће акције (играчев положај, положаји противника и сл.) неопходно је те информације извући из самих пиксела слике и превести их у неку репрезентацију која се лакше може користити. Неуронске мреже су добре у прављењу погодних репрезентација, али учење погодне репрезентације заједно са учењем исправних акција може бити врло нестабилан процес.

2. У машинском учењу високо димензиони подаци често представљају проблем. Сложеност модела и број параметара који нам је потребан за успешан тренинг расте експоненционално са порастом димензије улаза. Ово може бити посебно изазовно у случајевима када желимо на основу слика окружења (као што је случај приликом играња Атари игрица) донесемо одлуке о следећој акцији. Чак и релативно једноставна графика Атари игрица доноси десетине хиљада вредности на улазу у наш модел.

3. Како би агент научио да извршава задатке често је неопходно да има милионе интеракција са окружењем. Некада ово може бити изузетно скупо. Уколико желимо да научимо ауто да вози не желимо да га пустимо да врши милионе случајних радњи на улицама града. Поред финансијске и безбедоносне цене не треба занемарити ни временску цену прикупљања података. Скупљање више милиона интеракција са окружењем како би се добио користан агент, у зависности од времена потребног за једну интеракцију, може представљати временски најзахтевнији део тренинга.

4. Планирање је једна од основних техника које људи користе приликом избора својих акција. Шахисти потезе бирају на основу позиција које ће настати тек након неколико одиграних потеза, а не само на основу тренутне позиције. Како би ово симулирали са нашим агентом неопходно је да имамо могућност да симулирамо стање окружења након неколико примењених акција, што није доступно на једноставан начин у већини окружења.

Како би одговорили на наведене изазове, агента можемо посматрати као два модела који сарађују. Избор акције коју ће агент предузети у стању окружења $s$ можемо представити као композицију две функције $C(E(s))$. Задатак функције $E$ је прављење квалитетне репрезентације стања окружења, а задатак функције $C$ је избор одговарајуће акције на основу добијене репрезентације. Овом поделом добијена је могућност да се поставе неки додатни захтеви за функцију $E$ пре него што се почне са тренирањем агента.

Можемо захтевати од функције $E$ да простор у који пресликавамо стања чува неке особине окружења. На пример, можемо тражити постојање функције $T$ за коју важи да за свако стање окружења $s$ и акцију агента $a$ која окружење доводи у стање $s'$ важи следећа једнакост: $T(E(s), a) = E(s')$. Такође можемо захтевати постојање функција $R$ и $D$ за које важи $R(E(s), a) = r$ и $D(E(s), a) = d$, при чему је $r$ награда добијена при преласку у стање $s'$, а $d$ индикатор да ли је овим преласком завршена епизода интеракција са окружењем. Уређену четворку $(E, T, R, D)$ зваћемо *латеним моделом окружења*. Функције $E$, $T$, $R$ и $D$ можемо моделовати на основу случајних интеракција са окружењем пре него што кренемо са тренирањем модела $C$.

Овакво одвојено учење функција $E$ и $C$ може нам помоћи да одговоримо на горепоменуте изазове. Учење на корисним репрезентацијама, уместо на си-

ровим подацим добијеним од окружења, може нам значајно олакшати процес тренинга, а како је модел $E$ део латентног модела окружења, можемо претпоставити да су њиме добијене репрезентације корисне. Такође, с обзиром на то да димензију латентног простора бирамо ми, можемо добити репрезентације које су компактније од почетних опажања чиме смањујемо проблеме који су узроковани проклетством димензионалности (енг. curse of dimensionality). Поред тога, функције $T$, $R$ и $D$ добијеног модела окружења можемо користити за кретање кроз латентни векторски простор и тиме симулирати кретање кроз окружење. Ово нам даје могућност планирања у латентном простору, чиме можемо значајно побољшати перформансе модела.

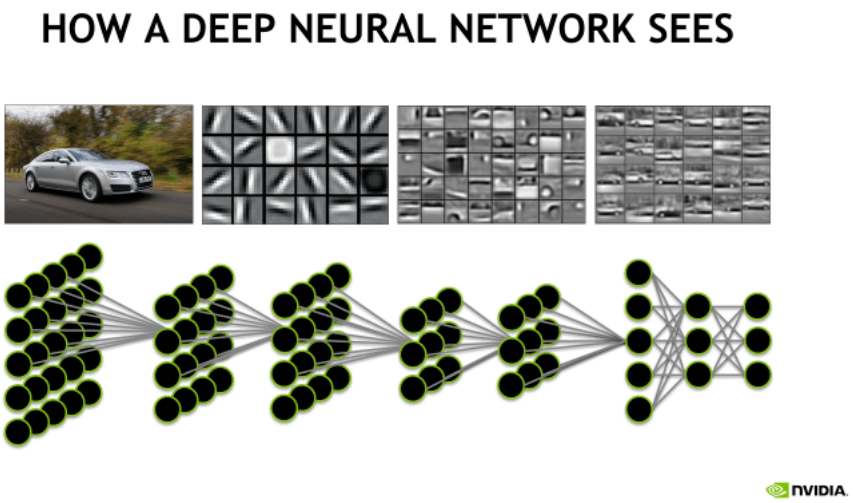
У овом раду приказаћемо две класичне технике за учење латентних модела окружења. Након тога ћемо анализирати ефекат који има учење на репрезентацијама добијеним латентним моделом окружења при тренингу агента. Окружења која ћемо користити за ове анализе биће представници три различите фамилије окружења чиме ћемо видети утицај латентних модела окружења у различитим ситуацијама.

# Глава 2

# Неуронске мреже

Иако идеја неуронских мрежа није нова [17] , класични модели као што су метода потпорних вектора (енг. support vector machines), стабла одлучивања и други дуго су имали примат у машинском учењу. Неки од разлога су боља интерпретабилност, статистичка гаранција за оцене грешака, добро познати услови конвергенције и пре свега рачунска захтевност. Ствари су се нагло промениле када је почело коришћење графичких картица за убрзање тренинга [14]. Ово убрзање омогућило нам је да користимо неуронске мреже чији је број параметара, а самим тим и њихов капацитет, за неколико редова величине већи него до тада. Ови нови моћнији модели постигли су до тада незапамћен резулат на чувеном изазову ImageNet [5] победивши конкуренцију у проблему класификације слика са чак 20% већом прецизношћу од другопласираног модела и тиме покренули ренесансу у дубоком учењу.

Један од разлога за ову огромну разлику у прецизности је могућност неуронских мрежа да, за разлику од класичних модела, у току израчунавања почетне атрибуте трансформишу и тиме добију нове репрезентације података на основу којих лакше могу доносити закључке. Ова могућност је посебно корисна за рад са сировим подацима као што су слике или звучни сигнали. На слици 2.1 можемо видети креирање нових репрезентација на делу. Након завршеног тренинга неуронске мреже на ImageNet-у анализирано је на које слике различити слојеви мреже највише реагују и тиме је добијена представа како мрежа трансформише улаз. Можемо видети да се прво на слици проналазе ивице објеката, након тога се на основу ивица проналазе сложенији облици као што су точкови и врата, а те информације мрежа склапа у проналазак целог аутомобила.




Слика 2.1: Илустрација како различити слојеви мреже праве нове атрибуте [16]

У овом делу рада увешћемо потпуно повезане и конволутивне неуронске мреже, а затим ћемо увести варијационе аутоенкодере. Ове архитектуре неуронских мрежа ће нам бити неопходне у прављењу модела окружења и агената.

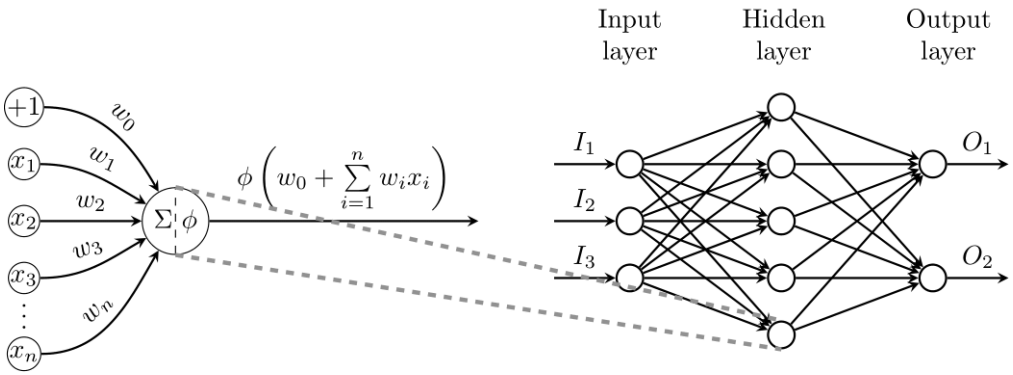
## 2.1 Потпуно повезане неуронске мреже

Потпуно повезане неуронске мреже (енг. fully connected neural networks или multi-layer perceptron) су прва и основна архитектура неуронских мрежа. Некада се називају и вештачке неуронске мреже (енг. artificial neural networks) како би се избегла забуна са мрежамама неурона у мозгу људи и животиња које су биле блага инспирација за њихову креацију.

Основна јединица неуронских мрежа је *неурон* који на основу више улазних сигнала производи излазни сигнал. Функција на основу које се рачуна излазни сигнал је:

$$output(\mathbf{x}) = \phi\left(\mathbf{w}^T\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix}\right) = \phi\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

где је $\mathbf{w} = (w_0,\ w_1, \ldots,\ w_n)$ вектор параметара неуронске мреже, а $\phi$ нелинеарна активациона функција. Неке од најчешће коришћених активационих




Слика 2.2: Дијаграм једног скривеног слоја потпуно повезане мреже [32]

функција су $ReLU(x) = \max(x, 0)$, $LeakyReLU(x) = \max(x, \lambda x)$, $\sigma(x) = \frac{1}{1+e^{-x}}$ и $\tanh(x) = \frac{e^x+e^{-x}}{e^x-e^{-x}}$.

Уколико посматрамо више овако дефинисаних неурона $\psi_i(\mathbf{x}), i = 1...m$ са истом активационом функцијом, слој неуронске мреже ће бити функција $f : \mathbb{R}^n \to \mathbb{R}^m$ задата на следећи начин:

$$f(\mathbf{x}) = (\psi_1(\mathbf{x}), \psi_2(\mathbf{x}), ..., \psi_m(\mathbf{x}))$$

Због имплементационих детаља, а и значајно погодније математичке репрезентације, ову формулу можемо записати и у следећем облику:

$$f(\mathbf{x}) = \boldsymbol{\phi}\left(\mathbf{W}\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix}\right)$$

где је $\mathbf{W} \in \mathbb{R}^{m\times(n+1)}$ матрица параметара неуронске мреже у којој се у редовима налазе параметри за сваки од неурона, а $\boldsymbol{\phi}(\mathbf{x})$ примена активационе функције $\phi$ на сваку од координата.

Коначно, потпуно повезану неуронску мрежу добијамо надовезивањем више слојева које смо управо описали. Уколико имамо слојеве $f_i(\mathbf{x}), i = 1 \dots k$ неуронска мрежа ће бити функција $NeuralNet(\mathbf{x}) = f_k(f_{k-1}(\dots(f_1(\mathbf{x}))\dots)$. Како излазе слојева $f_1 \dots f_{k-1}$ не користимо директно, ове слојеве често називамо скривени слојеви неуронске мреже. На слици 2.2 можемо видети како се гради неуронска мреже са једним скривеним слојем.

Битно је приметити да активациона функција у скривеним слојевима неуронске мреже не треба да буде линеарна. Уколико би активациону функцију заменили идентитетом, трансформацију коју врши један слој неуронске мреже би могли да запишемо као $f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ па би композиција два слоја $f_1$

и $f_2$ била:

$$f_2(f_1(\mathbf{x})) = \mathbf{A_2}(\mathbf{A_1}\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2} = (\mathbf{A_2}\mathbf{A_1})\mathbf{x} + (\mathbf{A_2}\mathbf{b_1} + \mathbf{b_2}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

за неку матрицу $\mathbf{A}$ и вектор $\mathbf{b}$. Можемо закључити да су све трансформације које можемо добити са ова два слоја само линеарна комбинација улаза, коју смо могли добити и са једним слојем. Слична рачуница се може применити и на линеарне активационе функције.

Штавише, постојање активационих функције не само да спречава свођење неуронских мрежа на линеарну трансформацију улазних атрибута, него их чини *универзалним апроксиматором* о чему говори следећа теорема.

**Теорема о универзалној апроксимацији.** *За свако $m, n \in \mathbb{N}$, сваки компактни скуп $K \subset \mathbb{R}^n$ и за сваку функцију $f \in C(K, \mathbb{R}^m)$*

$$(\forall \epsilon > 0)(\exists k \in \mathbb{N}, A \in \mathbb{R}^{k\times n}, b \in \mathbb{R}^k, C \in \mathbb{R}^{m\times k}) \sup_{x\in K} ||f(x) - C\sigma(Ax + b)|| < \epsilon$$

*акко $\sigma$ није полиномијална функција.*

Доказ теореме се може пронаћи у [9].

Израз $\sigma(Ax + b)$ препознајемо као један слој неуронске мреже тако да неформално теорему можемо интерпретирати: „неуронска мрежа са једним, довољно великим, скривеним слојем може произвољно добро апроксимирати произвољну непрекидну функцију на компактном скупу". Ово не значи да постоји ефикасан алгоритам за проналазак неуронске мреже која је произвољно добар апроксиматор, али дефинитивно даје добро оправдање за успешност неуронских мрежа.

## 2.2 Конволутивне неуронске мреже

Потпуно повезане неуронске мреже су моћне, али имају велики проблем са бројем параметара који користе. Уколико неки слој потпуно повезане мреже има $n$ улазних атрибута и $m$ излазних атрибута, матрица која одговара том слоју ће имати $(n + 1)m$ параметара. Типична слика високе резолуције има $1920 \times 1080$ пиксела, сваки пиксел се описује са 3 броја (црвена, зелена и плава компонента) што нам даје око 6 милиона улазних атрибута. Јасно је да ће број параметара потребан за прављење потпуно повезане неуронске мреже бити огроман, што је рачунски и меморијски неприхватљиво. Како би ефикасно

радили са сликама, неопходно је да користимо неку другу архитектуру мреже чији број параметара не расте са повећањем улаза.

Постоје два разлога за овако велики број параметара у потпуно повезаној неуронској мрежи. Први је то што у сваком слоју сваки неурон користи све улазне атрибуте за израчунавање свог излаза. Ово је пожељно уколико сматрамо да је комбиновање свих улазних атрибута неопходно за извлачење некаквих корисних информација. Међутим, у случају слика, вредности пиксела у једном углу слике врло вероватно немају много везе са пикселима у другом углу слике тако да овај број веза можемо смањити. Пожељно би било да, макар у почетним слојевима мреже, пиксели који учествују у формирању излаза једног слоја буду физички близу једни другима на слици. Други разлог за велики број параметара у потпуно повезаној мрежи је то што се за сваки излаз слоја користи нови скуп параметара. Уколико у различитим излазима неког слоја учествују пиксели из различитих делова слике, смањење броја параметара можемо постићи коришћењем истих параметара на сваком делу слике.

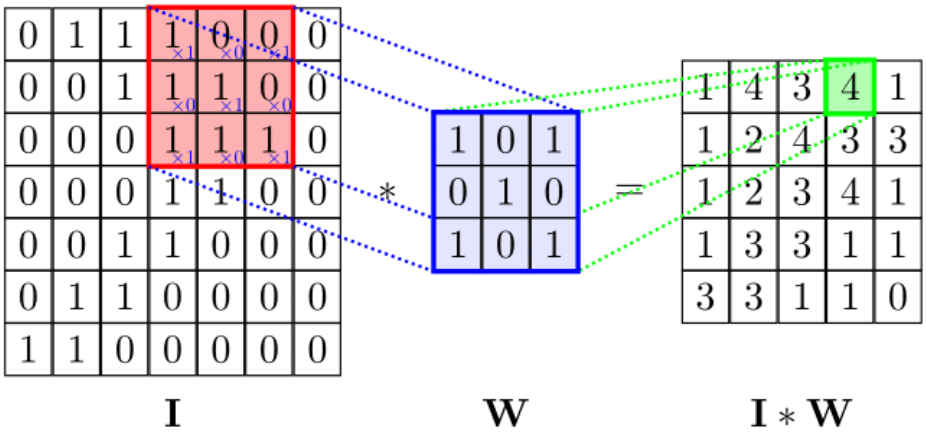
Операција конволуције природно даје могућност да одговоримо на ове проблеме. Конволуција две функције задате вредностима $f_{i,j} = f(x_i, y_j)$ и $w_{i,j} = w(x_i, y_j)$ за $i = 0...p-1$ и $j = 0...q-1$ дефинише се као:

$$(f * w)_{i,j} = \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} f_{k,l} w_{i-k,j-l} \tag{2.1}$$

При томе ћемо подразумевати да је $w_{i,j} = w_{p+i,j} = w_{i,q+j}$ како би елимнисали негативне индексе.

У случају да је $w_{i,j} = 0$ за скоро све $i$ и $j$ израчунавање суме може бити урађено у константном времену. Конволутивни слој неуронске мреже добијамо тако што операцију конволуције применимо на матрицу пиксела слике и, као и у случају потпуно повезаних неурноских мрежа, применом активационе функције на резултат конволуције. Вредности $w_{i,j}$ посматрамо као параметре конволутивног слоја.

Можемо приметити да код индекса параметара у формули 2.1 испред $k$ и $l$ стоји негативан знак, али се због једноставности и брже имплементације користи формула са позитивним знаком. Како су $w_{i,j}$ параметри конволутивног слоја, а ова промена знака је еквивалентна ротацији матрице параметара око споредне дијагонале, ово суштински нема никакав утицај на наша разматрања. Такође као што смо рекли већина вредности $w_{i,j}$ ће бити једнаки нули,




Слика 2.3: Пример операције конволуције [32]

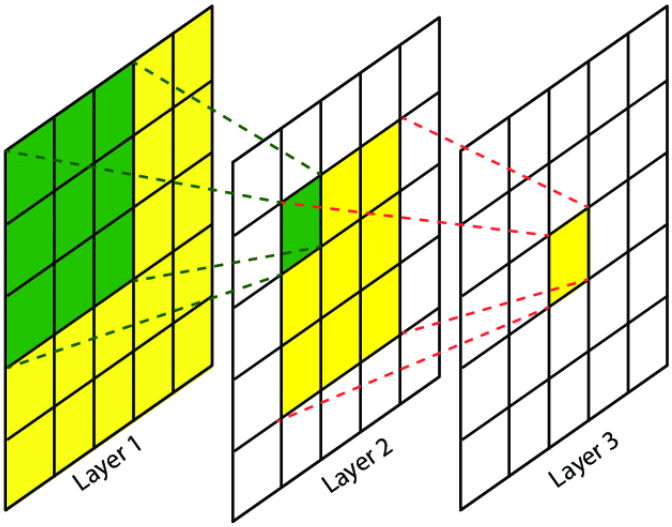
тако да је углавном представљамо као матрицу димензија $3 \times 3$ или $5 \times 5$ уз конвенцију да су остале вредности једнаке нули. Пример овако описане операције конволуције можемо видети на слици 2.3.

Матрица $W$ се често назива и филтером. Конволутивни слој неуронске мреже над једним улазом примењује више различитих филтера. У случају да улаз у конволутивни слој има више канала (као што су црвени, зелени и плави канал код слика) резултат примене једног филтера на тај улаз се добије применом одвојеног филтера на сваки од канала и сабирањем резултата.

Овако приказани слој добро разрешава проблеме које смо истакли на почетку секције. Присутан је локалност добијених излаза - сваки број на излазу из конволутивног слоја је комбинација бројева у малој околини (квадрата исте величине као филтер). Како применом филтера исте тежине користимо на сваком делу слике број параметара расте са бројем и величином филтера, а не са величином улаза и излаза што смо и желели да постигнемо.

Након сваког конволутивног слоја у мрежи можемо пратити који улазни пиксели су учествовали у прављењу пиксела на излазу. Као што можемо видети на слици 2.4 уколико користимо конволуције димензија $3 \times 3$ сваком излазном пикселу неког слоја је одговарало $3 \times 3$ пиксела претходног слоја, односно $5 \times 5$ пиксела два слоја уназад. Пикселе улазне слике који одговарају неком излазном пикселу неког слоја називамо рецептивно поље (енг. receptive field) излазног пиксела и може нам помоћи да закључимо шта тачно неуронска мрежа посматра приликом израчунавања тог пиксела.

Оно што такође можемо приметити је да ће суседни пиксели имати готово

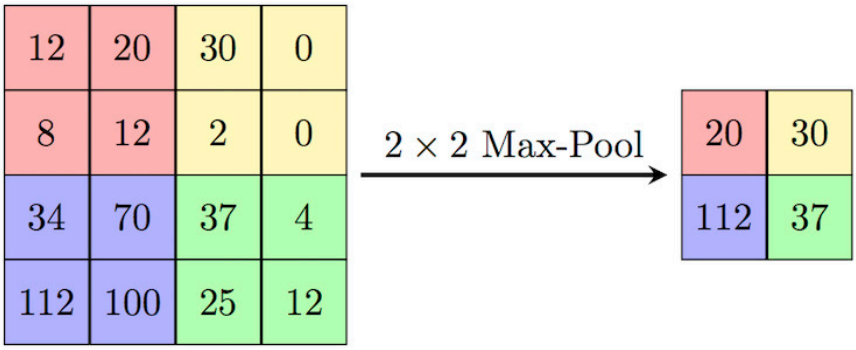



Слика 2.4: Рецептивно поље пиксела при конволуцији димензија $3 \times 3$ [15]

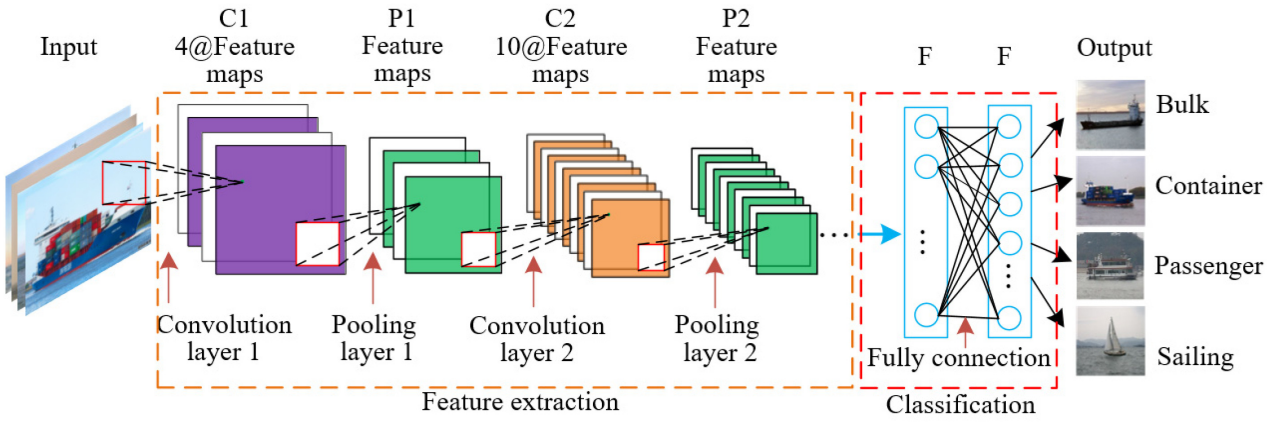
идентична рецептивна поља. Ово значи да ће они врло често имати сличне вредности зато што, као што смо описали, деле параметре приликом свог израчунавања. Можемо о овоме размишљати и на следећи начин. Уколико неки слој конволутивне неуронске мреже детектује точак аутомобила сви пиксели који у свом рецептивном пољу имају точак имаће високе вредности. Како нам понављање исте информације није корисно, има смисла све те информације спојити, односно агрегирати у један број.

Овим размишљањем долазимо до још једне операције која је често коришћена у конволутивним неуронским мрежама, а то је операција агрегације (eng. pooling operation). Она се дефинише тако што слику изделимо на сегменте и сваки од сегмената заменимо једним бројем, најчешће просеком бројева у сегменту или њиховим максимумом. Типична величина сегмената које користимо су $2 \times 2$ или $3 \times 3$, а принцип рада ове операције можемо видети на слици 2.5.

Слојеви конволуције и слојеви агрегације су врло погодни за извлачење корисних информација о слици, али уколико је репрезентација података коју имамо довољно добра, потпуно повезани слојеви се чешће користе. Зато након слојева конволуције и агрегације који су задужени за прављење нове репрезентације (енг. feature extraction) најчешће додајемо још неколико потпуно повезаних слојева које смо описали у претходној секцији. Овако представљена фамилија архитектура неуронских мрежа коришћена је за решавање много-




Слика 2.5: Пример операције агрегације максимумом [32]

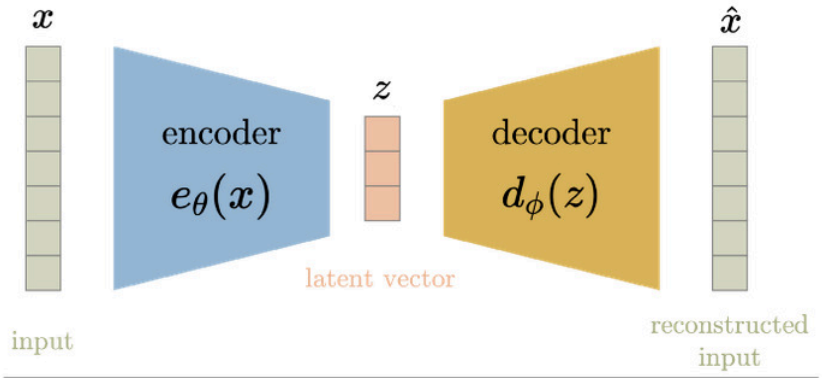



Слика 2.6: Пример архитектуре конволутивне неуронске мреже [21]

бројних проблема рачунарског вида и једну од архитектура можете видети на слици 2.6.

## 2.3 Варијациони аутоенкодери

Варијациони аутоенкодери су једна од најпопуларнијих група модела ненадгледаног учења. Припадају фамилији аутоенкодера чија је најчешћа улога смањење димензионалности података односно пресликавање података у неки релативно ниско дименизиони простор без губитка битних информација из почетне расподеле података. Смањење димензионалности је доста пожељно у машинском учењу због велике рачунске цене израчунавања над високо димензионим подацима, а у овом делу видећемо још неке корисне особине варијационих аутоенкодера.

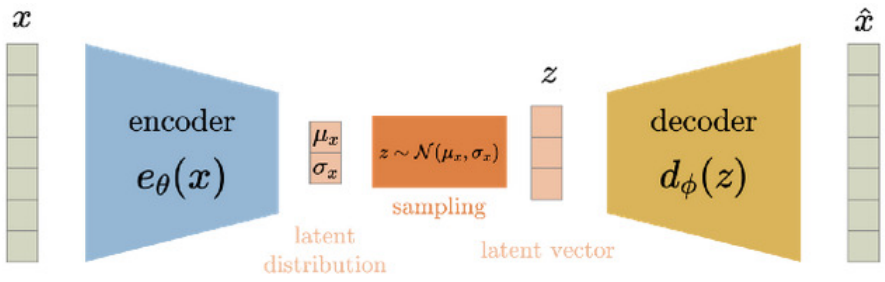



Слика 2.7: Блок дијаграм аутенкодера [2]

За почетак посматрајмо класичну архитектуру аутоенкодера 2.7. Овакву неуронску мрежу можемо третирати као два модела која удружено покушавају да науче нову репрезентацију података. Први модел који се назива енкодер задужен је за пресликавање почетних атрибута у неки нискодимензиони простор и његову примену можемо записати као $z = e_\theta(x)$ где су $\theta$ параметри модела. Други модел који се назива декодер задужен је за реконструкцију почетних атрибута из овако добијених репрезентација и његову примену можемо записати као $x_{recon} = d_\psi(z)$ где су $\psi$ параметри декодера. Тренирање оваквог модела се ради класичним техникама надгледаног учења где максимизујемо $p_{\theta,\psi}(x_{recon}|x)$ што се може свести на минимизацију средње квадратне функције грешке $L(x, \theta, \psi) = ||x - d_\psi(e_\theta(x))||^2$.

Овај основни приступ даје доста добре резултате и може се користити за смањење шума у подацима, детекцију аномалија и многе друге, али није идеални кандидат за наш рад. Као што знамо из увода, ми ћемо желети да се крећемо у латентном простору који је добијен применом енкодера $e_\theta$ и лоша расподела елемената у латентном простору нам може значајно отежати моделовање овог кретања. Како коришћењем класичних аутоенкодера немамо једноставан начин да контролишемо расподелу података у латентном простору, за тај задатак ће нам бити потребан нешто сложенији математички апарат варијационих техника које ћемо овде изложити.

Уколико посматрамо цео аутоенкодер, то ће бити модел који покушава да реконструише неку инстанцу из нашег скупа података. Како би то радио он мора да научи (приближну) расподелу вероватноћа нашег скупа података $p_\psi(x)$ при чему ће $\psi$ бити параметри модела. Подаци са којима радимо могу



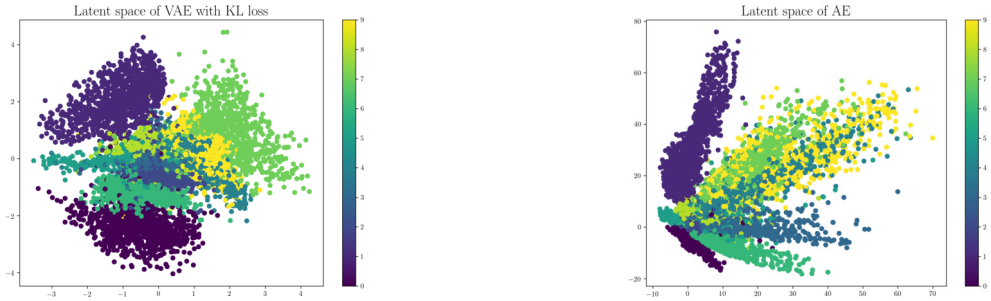

Слика 2.8: Блок дијаграм архитектуре варијационог аутоенкодера[2]

имати врло сложене расподеле тако да њихово учење може да се покаже као изазован проблем. Један од разлога овоме је што, ма колико детаљно скупљали податке, неке информације нећемо моћи да сазнамо. Односно постојаће неки атрибути који на нетривијалан начин утичу на расподелу, тако да учење модела $p_\psi(x, z)$ заједничке расподеле промењивих $x$ и латентне промењиве $z$ може бити потенцијално једноставније. Како би знали да ли смо ову расподелу добро научили можемо на основу ње израчунати расподелу почетног скупа података са $p_\psi(x) = \frac{p_\psi(x,z)}{p_\psi(z|x)} = \frac{p_\psi(x|z)p_\psi(z)}{p_\psi(z|x)}$. Како расподела $p_\psi(z)$ не зависи од $x$, можемо наметнути да то буде нормална расподела, $p_\psi(x|z)$ ће бити моделовано нашим декодером $d_\psi$, а једини преостали проблем је шта урадити са $p_\psi(z|x)$. Код аутонекодера смо користили енкодер за предвиђање тачне вредности латентне промењиве $e_\theta(x) = z$, док ће сада он моделовати параметре расподеле $q_\theta$ коју ћемо користити за апроксимацију расподеле $p_\psi(z|x)$. Најчешће се за $q_\theta$ узима нормална расподела са очекивањем и стандардном девијацијом коју предвиђа $e_\theta$ и ову архитектуру можемо видети на слици 2.8.

Оптимизацију параметара оваквог модела вршићемо максимизацијом функције $ELBO_{\theta,\psi}(x) = \mathbb{E}_{q_\theta(z|x)}[\log p_\psi(x, z) - \log q_\theta(z|x)]$. Како на први поглед не мора бити јасно шта постижемо максимизацијом ове функције, било би добро да је разложимо:

$$\begin{aligned} ELBO_{\theta,\psi}(x) &= \mathbb{E}_{q_\theta(z|x)}[\log p_\psi(x, z) - \log q_\theta(z|x)] \\ &= \mathbb{E}_{q_\theta(z|x)}[\log(p_\psi(z|x)p_\psi(x)) - \log q_\theta(z|x)] \\ &= \mathbb{E}_{q_\theta(z|x)}[\log p_\psi(z|x) + \log p_\psi(x) - \log q_\theta(z|x)] \\ &= \mathbb{E}_{q_\theta(z|x)}[\log p_\psi(x)] - D_{KL}(q_\theta(z|x)||p_\psi(z|x)) \end{aligned}$$

Први део последњег израза је ништа друго него очекивање логаритма вероватноће $p_\theta(x)$, и његовом максимизацијом умањујемо средње квадратну грешку




Слика 2.9: Слика латентног простора аутоенкодера(лево) и варијационог аутоенкодера(десно) након тренирања на mnist [6] скупу података [2]

између $x$ и реконструкције добијене моделом $d_\psi$. Други део израза је KL одступање (енг. Kullback-Leibler divergence) између расподела $q_\theta(z|x)$ и $p_\psi(z|x)$. Како је испред њега негативан знак, максимизацијом функције $ELBO_{\theta,\psi}$ ћемо смањивати KL одступање ове две дистрибуције. Минимум KL одступања две дистрибуције се постиже када су оне једнаке, тако да ћемо максимизацијом функције $ELBO_{\theta,\psi}$ приближавати расподелу $p_\psi(z|x)$ и њену апроксимацију $q_\theta(z|x)$.

Као што смо рекли расподела $q_\theta(z|x)$ ће најчешће бити нормална расподела са параметрима добијеним моделом $e_\theta$, тако да максимизацијом $ELBO_{\theta,\psi}$ форсирамо да расподела латентне промењиве $p_\psi(z|x)$ буде што ближа нормалној. На слици 2.9 су приказане расподеле латентне промењиве $z$ након тренирања аутоенкодера и варијационог аутоенкодера на скупу mnist [6] који садржи слике руком писаних цифара. Можемо приметити да је расподела латентне промењиве $z$ за слике истих цифара заиста ближа нормалној расподели на слици лево, за разлику од линија које видимо на слици десно.

Максимизацију функције $ELBO$ ћемо радити стохастичким градијентним спустом и вреди се осврнути на рачунање непристрасне оцене њеног градијента.

За почетак можемо приметити да израчунавање градијента $\nabla_\psi ELBO_{\theta,\psi}(x)$ не представља проблем. Како расподела $q_\theta$ не зависи од $\psi$ градијент једноставно може да уђе у очекивање и тиме добијемо

$$\begin{aligned}\nabla_\psi ELBO_{\theta,\psi}(x) &= \nabla_\psi \mathbb{E}_{q_\theta(z|x)}[\log p_\psi(x,z) - \log q_\theta(z|x)] \\ &= \mathbb{E}_{q_\theta(z|x)}[\nabla_\psi(\log p_\psi(x,z) - \log q_\theta(z|x))] \\ &= \mathbb{E}_{q_\theta(z|x)}[\nabla_\psi \log p_\psi(x,z)]\end{aligned} \tag{2.2}$$

а како користимо стохастички градијентни спуст довољно је да знамо градијент $\nabla_\psi \log p_\psi(x, z)$ који је могуће израчунати зато што је декодер неуронска мрежа која је по дизајну диференцијабилна.

Оно што ће нам правити проблем јесте градијент по параметрима $\theta$ зато што је и расподела по којој вршимо очекивање параметризована тим параметрима. Како бисмо решили тај проблем користимо репараметризациони трик [11]. Уместо да $z$ буде узорак расподеле $q_\theta(x)$ идеја је да представимо узорковање као диференцијабилну функцију $z = g(\epsilon, \theta, x)$ где ће $\epsilon$ бити нова случајна промењива чија расподела не зависи од $\theta$ и $x$. Како важи $E_{q_\theta(z|x)}[f(z)] = E_\epsilon[f(g(\phi, \theta, x)]$, а расподела $\epsilon$ не зависи од $\theta$, применом овог трика градијент $\nabla_\theta ELBO_{\theta,\psi}(x)$ можемо непристрасно оценити рачуном сличним 2.2. У пракси за $q_\theta$ користимо нормалну расподелу $\mathcal{N}(\mu, \sigma^2)$ где су $(\mu, \theta) = e_\theta(x)$, па за примену репараметризационог трика користимо $\epsilon \sim \mathcal{N}(0, 1)$ и $z = \mu + \sigma\epsilon$.

И поред својих добрих особина као и математике која стоји иза њих, варијациони аутоенкодери нису без лоших страна. Као што је наведено у [12], њихово тренирање лако може дати лоше резултате. Процес тренинга често наилази до лоших локалних минимума, а и слике које добијамо реконструкцијом врло често су прилично замућене и недостају им ситни детаљи. Ово значи да у случају да у скупу података имамо пуно слика где се само по неки детаљ разликује, резултујуће реконструкције ће вероватно изгледати идентично, што може правити посебне проблеме приликом играња игрица где су често ситни детаљи најбитнији на екрану.

# Глава 3

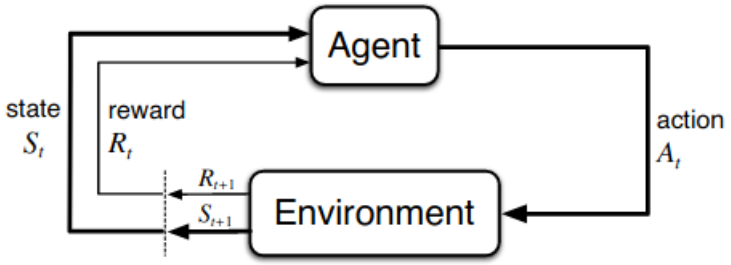
# Учење поткрепљивањем

Основне појмове учења поткрепљивањем смо већ видели у Уводу, а то су *опажања* и *награде* које добијамо од *окружења*, као и *агент* који у том окружењу предузима *акције*. У овој секцији ћемо ове појмове формално дефинисати као Марковљев процес одлучивања (енг. Markov decision process), анализираћемо шта значи решити овако задат процес и затим ћемо приказати једну од техника за приближно решавање овог проблема коју ћемо даље користити у експериментима.

## 3.1 Учење поткрепљивањем као Марковљев процес одлучивања

Како би боље описали појмове које ћемо користити можемо посматрати две игре, Pac-Man и познату игру са картама таблић. *Агент* је јединица за избор акција које се предузимају. Средина којом агент интерагује називамо *окружење*. У случају игре Pac-Man агент ће бити замена за људског играча, а окружење ће бити сама игрица.

Једна партија игре коју ћемо звати и *епизода* ће се састојати од, потенцијално бесконачног, низа узастопних интеракција са окружењем. Ове интеракције се могу дешавати непрекидно, као што је случај са игрицом Pac-Man или у дискретним временским интервалима (потезима) као што је случај са игром таблић. Иако можда на први поглед ово може представљати битну разлику, можемо природно дискретизовати непрекидно време на коначне сегменте (на пример ограничимо број интеракција на 30 по секунди). Након ове дискре-




Слика 3.1: Начин интеракције између окружења и агента, преузето из [28]

тизације, ако је она била неопходна, добијамо низ интеракција у дискретним временским тренуцима $t = 0, 1, 2, 3...$ које се састоје од следећих корака:

1. Од окружења агент добија опажање $s_t$ које представља неку репрезентацију стања окружења.

2. На основу опажања $s_t$ агент бира акцију коју предузима $a_t$.

3. Окружење агенту даје награду $r_{t+1} \in \mathbb{R}$ и ново опажање $s_{t+1}$.

Ради лакше нотације скуп свих могућих опажања обележаваћемо са $\mathcal{S}$. Једно опажање које добијемо од окружења може бити један фрејм игрице у случају игрице Pac-Man или вредности карата које имамо у руци и карата које са налазе на табли у случају таблића. Битна разлика коју овде можемо приметити је да у случају игрице Pac-Man један фрејм даје довољно информација да поступимо оптимално у сваком тренутку, док то није случај са таблићем. Како бисмо одиграли најбољи потез потребно је знати и карте противника, а та информација нам није на располагању. Алгоритми које ћемо приказати неће правити разлику између ова два случаја, али је битно да ово узмемо у обзир када оцењујемо перформансе наших агената.

Скуп свих акција које агент може предузети у стању $s$ обележаваћемо са $\mathcal{A}(s)$. Овај скуп се може мењати између различитих стања окружења, на пример код игрице Pac-Man не постоји могућност скретања унутар тунела. Алгоритми које ћемо приказати могу радити и у случају промењивог скупа акција, али то значајно отежава њихову имплементацију. Из овог разлога, у пракси најчешће посматрамо случај где увек имамо исти скуп акција (на пример унију свих могућих акција), а накнадно игноришемо недоступне акције уколико агент покуша да их предузме [10, 25].

Награда $r_t$ биће кључан фактор у одлучивању које акције је добро предузети, а које акције су потенцијално лоше. За резоновање о томе колико су акције добре, односно лоше неопходно је да разумемо како уопште наше акције утичу на награду. Како право стање окружења агенту није доступно, већ о њему резонује на основу опажања која добија од окружења, у наставку ћемо кад кажемо стање подразумевати опажање које агент добија од окружења.

Једини начин на који агент добија информације о утицају својих акција су опажања и награде које добија од окружења. Као што смо рекли опажања не дају целу слику о окружењу тако да ће опажање које добијамо након примењене акције, као и награда бити недетрминистички. У општем случају оне ће се мењати на основу расподели вероватноћа

$$P\{r_{t+1} = r, s_{t+1} = s|s_0, a_1, r_1, ..., s_{t-1}, a_{t-1}, r_t, s_t, a_t\}$$

Битно је приметити да у овој расподели фигуришу све прећашње акције које смо предузели, награде које смо добили и сва стања у којима смо били. У неким окружењима зависност од прећашњих стања је неопходна, на пример у таблићу карте које су се раније појавиле неће се опет појавити на табли, али у великом броју окружења можемо је избећи. Својство које важи код таквих окружења можемо формално дефинисати на следећи начин:

$$P\{r_{t+1} = r, s_{t+1} = s|s_t, a_t\} = P\{r_{t+1} = r, s_{t+1} = s|s_0, a_1, r_1, ..., s_{t-1}, a_{t-1}, r_t, s_t, a_t\} \tag{3.1}$$

Ово својство називамо *Марковљево својство*, а уређену тројку $(\mathcal{S}, \mathcal{A}, P)$ називамо *Марковљев процес одлучивања* (енг. Markov Descision process, MDP). Неформално можемо рећи да окружење испуњава Марковљево својство уколико је оно у потпуности описано тренутним стањем. Како нам ово омогућава да бирамо акције само на основу тренутног стања, окружења која испуњавају Марковљево својство биће нам посебно погодна за анализу. Када радимо на неком окружењу које није Марковљево најчешће ћемо за опажања која дајемо агенту узимати комбинацију неколико последњих опажања која смо добили од окружења. Овиме поред тренутног стања, агенту дајемо и информацију о ограниченој историји која је до тог стања довела, чиме настојимо да умањимо грешку коју добијамо претпоставком да је Марковљево својство испуњено.

У зависности од контекста некада нам неће бити неопходна заједничка расподела $p(s', r|s, a) = P\{r_{t+1} = r, s_{t+1} = s'|s_t = s, a_t = a\}$ већ само расподела награда или следећих стања. Из овог разлога ћемо увести још две ознаке

$r(r|s,a) = \sum_{s'} p(s', r|s, a)$ и $t(s'|s,a) = \sum_r p(s', r|s, a)$ за одговарајуће маргиналне расподеле.

## 3.2 Решење Марковљевог процеса одлучивања

У преходној глави смо описали окружење као један Марковљев процес одлучивања, али и даље нисмо формализовали концепт прикупљања максималне награде. У овој секцији ћемо формално дефинисати овај појам и дефинисаћемо шта то значи решити Марковљев процес одлучивања.

Нека је наш агент у једној епизоди интеракција са окружењем добијао награде $R_i, i = 1...T, T \in \mathbb{N} \cup \{\infty\}$. *Добит* $G_t$ у тренутку $t$ дефинишемо као суму награда коју смо добили од тренутка $t$, односно $G_t = \sum_{i=t+1}^{T} R_i$. Наш циљ ће бити максимизација ове добити. Можемо приметити да овако дефинисана сума за $T = \infty$ може бити бесконачна, у ком случају максимизација не би имала смисла. Из овог разлога се често уместо добити посматра *умањена добит* дефинисана као $G_t = \sum_{i=t+1}^{T} \gamma^{i-t-1} R_i$ коју ћемо онда максимизовати. Коефицијент $\gamma \in [0, 1)$ називамо *фактор умањења* и можемо приметити да у случају да су награде ограничене (што је у пракси увек случај) ова сума конвергира, тако да можемо причати о њеној максимизацији и за $T = \infty$.

Начин на који наш агент бира акције које предузима звaћемо политика. У општем случају политика неће бити детерминистичка већ ће агент примењивати акције у складу са расподелом $\pi(a_t|s_t, r_t, a_{t-1}, s_{t-1}, r_{t-1}, ..., s_0)$. Као што смо већ напоменули у случају да посматрамо Марковљево окружење неће бити потребе да прошла стања и акције утичу на акцију коју ћемо предузети, тако да ћемо најчешће сматрати да политика зависи само од тренутног стања и обележаваћемо је са $\pi(a|s)$.

Са овако дефинисаним функцијама $G_t$ и $\pi$ можемо дефинисати следеће три функције које ће нам бити корисне у следећој секцији. Прва је *функција вредности стања* која ће нам описивати колико су стања вредна при тренутној политици.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi \left[ \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k | s_t = s \right] \tag{3.2}$$

Друга је *функција вредности стања и акција*, у литератури често названа Q-функција, која нам описује колика је очекивана награда уколико применимо акцију у неком стању.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k | s_t = s, a_t = a\right] \tag{3.3}$$

Трећа је *функција предности* (енг. advantage function) која нам говори колико је нека акција добра у контексту наше тренутне политике.

$$A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$$

Након што смо увели дефиниције политике природно и формализовали појам награде, можемо формално дефинисати шта значи да је једна политика боља од друге. Природан начин да ово дефинишемо је коришћењем функције стања $v_\pi$:

$$\pi_1 \preceq \pi_2 \Leftrightarrow (\forall s)(v_{\pi_1}(s) \leq v_{\pi_2}(s))$$

Видимо да је овиме дефинисано парцијално уређење међу свим политикама где важи да је једна политика боља од друге уколико добија већу награду у свим стањима. Оптимална политика $\pi^*$ биће она која није гора ни од једне друге политике, и њу сматрамо за решење Марковљевог процеса одлучивања.

Можемо приметити на основу претходних формула да фактор умањења $\gamma$ има утицај и на то коју политику сматрамо оптималном, а не само на конвергенцију функције добити. Како су касније награде експоненцијално отежане са $\gamma \in [0, 1)$ за мале вредности оптимална политика ће преферирати краткорочне награде, док ће за веће вредности политика преферирати каснију добит.

## 3.3 Оптимизација политике

Проналажење оптималне политке коју смо описали у претходном делу у пракси представља изузетно тежак проблем. Иако постоје алгоритми за проналазак оптималне политике, они су најчешће изузетно рачунски захтевни. Из тог разлога алгоритми за оптимизацију политике врло често неће имати гаранцију конвергенције ка оптималној политици. Битно је приметити да, и поред одрицања гаранције о оптималности политике, ови алгоритми могу надмашити људске перформансе у неким изазовним задацима.

Прва фамилија алгоритама која се користи у дубоком учењу поткрепљивањем заснива се на апроксимацији Q-функције оптималне политике и зато има назив дубоко Q-учење. Знање вредности Q-функције оптималне политике нам омогућава једноставан начин израчунавања оптималне политике као $\pi_*(s) = \text{argmax}_a(q_*(a, s))$. Неки од алгоритама који припадају овој фамиилији су DQN [19], SARSA [22], и RAINBOW [8], али они неће бити предмет овог рада.

Друга фамилија алгоритама која се користи у дубоком учењу поткрепљивањем заснива се на апроксимацији градијената политике. Наш циљ је да направимо модел који ће нам давати политику и да градијентним успоном пронађемо што боље параметре овог модела. Неки од алгоритама који припадају овој фамилији су A2C[18], TRPO[23] и PPO[24] који ћемо у овом раду користити и чији ћемо метод описати у овом делу рада.

У свим алгоритмима заснованим на градијентима политике, политика коју оптимизујемо биће $\pi_\theta$ где су $\theta$ параметри политике (најчешће параметри неуронске мреже). Наш циљ ће бити да итеративно ажурирамо ове параметре градијентним успоном, при чему желимо да након сваког ажурирања параметара неуронске мреже политика буде боља, односно желимо да важи $\pi_{\theta_{old}} \preceq \pi_{\theta_{new}}$.

Како би ово постигли анализирајмо разлику функције вредности стања за неке две политике:

$$
\begin{aligned}
v_{\pi_{new}}(s) - v_{\pi_{old}}(s) &= \mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k r_{k+1}\middle| s_0 = s\right] - v_{\pi_{old}}(s) \\
&= \mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k r_{k+1} - v_{\pi_{old}}(s_0)\middle| s_0 = s\right] \\
&= \mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k r_{k+1} + \sum_{k=0}^{\infty}(\gamma^{k+1} v_{\pi_{old}}(s_{k+1}) - \gamma^k v_{\pi_{old}}(s_k))\middle| s_0 = s\right] \\
&= \mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k (r_{k+1} + \gamma v_{\pi_{old}}(s_{k+1}) - v_{\pi_{old}}(s_k))\middle| s_0 = s\right] \\
&= \mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k (q_{\pi_{old}}(s_k, a_k) - v_{\pi_{old}}(s_k))\middle| s_0 = s\right] \\
&= \mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k A_{\pi_{old}}(s_k, a_k)\middle| s_0 = s\right]
\end{aligned}
$$

Одавде видимо да уколико важи $\mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k A_{\pi_{old}}(s_k,a_k)\middle|s_0=s\right]\geq 0$ за свако $s\in\mathcal{S}$ онда знамо да је $\pi_{old}\preceq\pi_{new}$. Овај израз се може даље разложити на следећи начин:

$$\begin{aligned}
&\mathbb{E}_{\pi_{new}}\left[\sum_{k=0}^{\infty}\gamma^k A_{\pi_{old}}(s_k,a_k)\right]= \\
&=\sum_{t=0}^{\infty}\sum_{s}P(s_t=s|\pi_{new})\sum_{a}\pi_{new}(a|s)\gamma^t A_{\pi_{old}}(s,a) \\
&=\sum_{s}\sum_{t=0}^{\infty}\gamma^t P(s_t=s|\pi_{new})\sum_{a}\pi_{new}(a|s)A_{\pi_{old}}(s,a) \\
&=\sum_{s}\rho_{\pi_{new}}(s)\mathbb{E}_{\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)] \qquad (3.4)
\end{aligned}$$

При чему је $\rho_{\pi_{new}}(s)=\sum_{t=0}^{\infty}\gamma^t P(s_t=s|\pi_{new})$ сума вероватноћа да се неко стање посети експоненцијално отежана бројем корака које смо до њега направили. Како би политика $\pi_{new}$ била боља од политике $\pi_{old}$ довољно је да очекивање $\mathbb{E}_{\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)]$ буде увек позитивно.

Овај услов је могуће тривијално испунити тако што се политика $\pi_{new}$ изабере детерминистички $\pi_{new}(s)=\operatorname{argmax}_a A_{\pi_{old}}(s,a)$, али овако једноставно унапређивање политике у пракси неће бити могуће. Први проблем на који наилазимо је то што тачне вредности функције предности $A_\pi$ најчешће нећемо имати на располагању већ само њене апроксимације што неизбежно доводи до грешака у израчунавању. Други проблем је што ће политике са којима радимо бити реализоване путем неуронских мрежа, тако да проналажење параметара мреже за који ће важити $\pi_\theta(s)=\operatorname{argmax}_a A_{\pi_{old}}(s,a)$ може бити практично неизводљиво.

Уместо тога на основу политике $\pi_\theta$ и искуства који ћемо сакупити коришћењем те политике нове параметре желимо пронаћи максимизацијом леве стране неједнакости 3.4. Једна од препрека за коришћење градијентног успона за максимизацију ове функције је то што је функцију $\rho_{\pi_{new}}$ и њен градијент тешко израчунати. Зато ћемо уместо ње користити следећу апроксимацију:

$$\sum_{s}\rho_{\pi_{new}}(s)\mathbb{E}_{\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)]\approx\sum_{s}\rho_{\pi_{old}}(s)\mathbb{E}_{\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)] \qquad (3.5)$$

Као што видимо $\rho_{\pi_{new}}(s)$ заменићемо са $\rho_{\pi_{old}}(s)$ и тиме смо елеминисали потребу за рачунање њеног градијента. Оно што можемо очекивати је да

ова апроксимација има смисла само уколико се ове политике не разликују превише. Овај увид нам карактерише следећа оцена чије извођење можемо пронаћи у [23]:

$$\mathbb{E}(v_{\pi_{new}}(s_0)) \geq \mathbb{E}(v_{\pi_{old}}(s_0)) + L_{\pi_{old}}(\pi_{new}) - C \max_s D_{KL}(\pi_{new}(\cdot|s)||\pi_{old} \cdot |s)) \quad (3.6)$$

$$L_{\pi_{old}}(\pi_{new}) = \sum_s \rho_{\pi_{old}}(s)\mathbb{E}_{\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)]$$
$$\epsilon = |A_{\pi_{old}}(s,a)|$$
$$C = \frac{4\gamma \max_{s,a} \epsilon}{(1-\gamma)^2}$$

Уколико се за тренутак ослонимо на то да можемо израчунати $A_\pi(s,a)$ ова оцена је прилично погодна за нас. Максимизацијом израза

$$L_{\pi_{old}}(\pi_{new}) - C \max_s D_{KL}(\pi_{new}(\cdot|s)||\pi_{old} \cdot |s))$$

долазимо до алгоритма који има гаранцију да ће очекивање вредности функције стања итеративно расти [23]. Максимизацију овог израза можемо вршити, на пример, градијентним успоном чиме долазимо до следећег алгоритма.

**Алгоритам 1:** Алгоритам са гаранцијом растуће вредности политике

Иницијализуј $\pi_0$
**for** $i = 1, 2, \ldots$ **do**
  Израчунај вредности функције предности $A_{\pi_i}(s,a)$
  Израчунај $\pi_{i+1}$ као решење оптимизационог проблема:
  $\pi_{i+1} = \text{argmax}_\pi \left[L_{\pi_i}(\pi_{i+1}) - C \max_s D_{KL}(\pi_{i+1}(\cdot|s)||\pi_i \cdot |s))\right]$

У пракси се показало да иако се политике које добијамо овим алгоритмом стално побољшавају, кораци које правимо су исувише мали што чини процес непрактично спорим. Део $C \max_s D_{KL}(\pi_{new}(\cdot|s)||\pi_{old}(\cdot|s))$ у 3.5 нам ограничава величину корака како би апроксимације које смо правили биле довољно добре, али је такође превише ограничавајући тако да га морамо релаксирати. Зато га можемо заменити оптимизацијом функције $L_{\pi_i}(\pi_{i+1})$ при ограничењу $\max_s D_{KL}(\pi_{i+1}(\cdot|s)||\pi_i(\cdot|s)) \leq \delta$. Параметар $\delta$ биће један од хиперпараметара

нашег модела, и ово ограничење се назива ограничењем интервала поверења по чему је алгоритам TRPO добио име (енг. trust region policy optimization).

Само ова промена не би била довољна да убрза конвергенцију алгоритма, већ ћемо поред ње такође направити низ апроксимација како би смањили рачунску захтевност. За почетак при рачунању $L_{\pi_{old}}(\pi_{new})$ се морамо ослободити суме по свим стањима што ћемо урадити на следећи начин:

$$\begin{aligned}
L_{\pi_{old}}(\pi_{new}) &= \sum_s \rho_{\pi_{old}}(s)\mathbb{E}_{a\sim\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)] \\
&\approx \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\pi_{old}}}\left[\mathbb{E}_{a\sim\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)]\right] \\
&\approx \frac{1}{1-\gamma}\mathbb{E}_{s\sim\pi_{old}}\left[\mathbb{E}_{a\sim\pi_{new}(\cdot|s)}[A_{\pi_{old}}(s,a)]\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\pi_{old}}\left[\sum_a [\pi_{new}(a|s)A_{\pi_{old}}(s,a)]\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\pi_{old}}\left[\sum_a \left[\pi_{old}(a|s)\frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}A_{\pi_{old}}(s,a)\right]\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s,a\sim\pi_{old}}\left[\frac{\pi_{new}(a|s)}{\pi_{old}(a|s)}A_{\pi_{old}}(s,a)\right]
\end{aligned}$$

Након тога се желимо ослободити ограничења КЛ-одступања на целом скупу стања

$$\overline{DKL_\pi}(\pi_{i+1}(\cdot|s)||\pi_i(\cdot|s)) \coloneqq \mathbb{E}_{s\sim\pi}[D_{KL}(\pi_{i+1}(\cdot|s)||\pi_i(\cdot|s)]$$

Уместо ограничења $\max_s D_{KL}(\pi_{i+1}(\cdot|s)||\pi_i(\cdot|s)) \leq \delta$ можемо користити ограничење $\overline{DKL_{\pi_i}}(\pi_{i+1}(\cdot|s)||\pi_i(\cdot|s)) \leq \delta$. Овим апроксимацијама добили смо могућност да уместо целог скупа стања посматрамо само стања која добијамо коришћењем текуће политике. Ове вредности онда можемо апроксимирати једноставним упросечавањем вредности на стањима које посетимо коришћењем старе политике. Јасно је да ће обе апроксимације довести до непрецизности, али како обе вредности апроксимирамо вредностима у стањима које посећујемо политиком очекујемо да у том региону грешка не буде превелика. Коришћењем ових апроксимација долазимо до алгоритма TRPO [2]. Како би поједноставили нотацију у алгоритму и надаље у раду користићемо следеће

ознаке:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$
$$A_t = A_{\theta_{old}}(a_t, s_t)$$
$$\hat{\mathbb{E}}_t[f(a,s)] = \frac{1}{T}\sum_{t=1}^{T} f(a_t, s_t)$$

**Алгоритам 2:** TRPO

Иницијализуј параметре $\theta$
**for** *iteration* $= 1, 2, \ldots$ **do**
  $\theta_{old} = \theta$
  Крећи се кроз окружење политиком $\pi_{\theta_{old}}$ за T корака
  Израчунај функције предности $A_1, \ldots, A_T$
  Нове параметре пронађи као приближно решење проблема
  $\theta = \text{argmax}\left[\hat{\mathbb{E}}_t[r_t(\theta)A_t]\right]$ при услову $\overline{DKL}_{\pi_{\theta_{old}}}(\pi_\theta(\cdot|s)||\pi_{\theta_{old}}(\cdot|s)) \leq \delta$

Најсложенији корак овог алгоритма је приближно решавање проблема тражења условног минимума функције $\max_\theta \left[\hat{\mathbb{E}}_t[r_t(\theta)A_t]\right]$. Уколико би игнорисали услов ограниченог КЛ-одступања политика би се превише брзо мењала што би имало деструктивни ефекат [23]. Коришћење константе као у алгоритму са гаранцијом растуће политике има проблем компликованог рачуна константе, тако да би било добро наћи још неки начин за ограничавање превеликих корака. У раду [24] је показано да ово можемо постићи заменом функције $\hat{\mathbb{E}}_t[r_t(\theta)A_t]$ коју максимизујемо са:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{3.7}$$

Размотримо израз $\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)$. Уколико је $A_t > 0$ желимо чешће да вршимо ове акције тако да ће градијент тежити да повећа $r_t$. Ово повећање нам се свиђа, али након промене од $1+\epsilon$ улазимо у опасност да вршимо превелику промену политике, те је ту раст ограничен и градијент ће бити нула. Аналогно, ако је $A_t < 0$ направили смо релативно лошу акцију тако да желимо да смањимо колико често је предузимамо што ће смањити $r_t$. Како би спречили да се $r_t$ превише смањи ограничење након промене на $r_t < 1-\epsilon$ градијент ће бити нула. Део $r_t(\theta)\hat{A}_t$ је ту како би у случају да направимо погрешан корак у тражењу максимума могли лакше да се вратимо у интервал $r_t \in [1-\epsilon, 1+\epsilon]$.

Коначно можемо прокоментарисати и како долазимо до вредности функције предности $A_t$. У општем случају ова вредност нам неће бити доступна те је морамо апроксимирати. Ово ћемо урадити тиме што ћемо поред неуронске мреже која нам рачуна политику $\pi_\theta$ имати неуронску мрежу која апроксимира функцију вредности стања $v_\phi$. Ове неуронске мреже могу делити параметре што је генерално честа пракса. Вредности које ће нам мрежа $v_\phi$ дати користићемо за апроксимацију функције предности $A_t$:

$$\hat{A}_t = -v_\phi(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{t-1} + \gamma^{T-t} v_\phi(s_T)$$

Како је потребно да се функција $v_\phi$ такође оптимизује то радимо минимизацијом функције $L_t^{VF} = (v_\theta(s_t) - v_t^{targ})^2$ приликом тренинга. За $L_t^{VF}$ се може узети исти израз као и за $\hat{A}_t$. Коначно, како би подстакли агента да истражује на почетку тренинга у коначну функцију коју максимизујемо у алгоритму PPO додајемо максимизацију ентропије расподеле акција $S[\pi_\theta](s_t)$. Када спојимо све ове делове добијамо следећу функцију коју ћемо максимизовати:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[ L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{3.8}$$

Константе $c_1$ и $c_2$ су хиперпараметри алгоритма и у раду ћемо користити вредности које се могу пронаћи у имплементацији [13]. Како бисмо убрзали процес скупљања интеракција са окружењем са старом политиком крајњи алгоритам користи више паралелних агената што нам даје следећи алгоритам:

**Алгоритам 3:** PPO, Actor-Critic Style

**for** *iteration* $= 1, 2, \ldots$ **do**
  **for** *actor* $= 1, 2, \ldots, N$ **do**
    Крећи се кроз окружење са политиком $\pi_{\theta_{old}}$ за $T$ корака
    Израчунај функције предности $\hat{A}_1, \ldots, \hat{A}_T$
  Са $K$ епоха стохастичног градијентног успона максимизуј функцију $L(\theta)$ коришћењем $N \cdot T$ сачуваних корака
  $\theta_{old} \leftarrow \theta$

# Глава 4

# Латентни модели окружења

Шахиста приликом избора свог следећег потеза анализира позиције које ће настати у будућности. Позиције које он анализира најчешће посматра на замишљеној шаховској табли на којој и помера потезе које планира да одигра. Ова замишљена табла представља експлицитан модел окружења, односно шахиста зна сва правила на основу којих ће се позиције мењати што му омогућава да верно симулира будуће ситуације.

Ово експлицитно моделовање је могуће у (релативно) једноставним окружењима, као што је игра шах, али није могуће у реалном свету. Приликом вожње аутомобила ми не размишљамо о сваком детаљу нашег окружења, већ користимо апстрактније репрезентације аутомобила, пешака и других објеката око нас. То нам омогућава да брзо доносимо одлуке без фокусирања на детаље који нису од суштинске важности. Задатак латентни модели окружења биће моделовање ових апстрактних репрезентација.

Како би ову идеју искористили у учењу поткрепљивањем циљ ће нам бити да научимо пресликавање $E : \mathcal{S} \to \mathbb{R}^n$ које ће нам давати компактнију репрезентацију стања окружења. Ову, као и све друге функције везане за латентне моделе окружења, моделоваћемо неуронским мрежама.

Наш циљ је да ова репрезентација добро моделује промене окружења које настају услед нашег деловања у њему, тако да ћемо моделовати и функцију $T : \mathbb{R}^n \times \mathcal{A} \to \mathbb{R}^n$. Њен циљ ће бити да промене стања окружења услед примене одређене акције осликава у латентном простору. Ово можемо изразити следећим условом:

$$(\forall s \in \mathcal{S})(\forall a \in \mathcal{A}(s))(T(E(s), a) = \mathbb{E}_{s' \sim t(s,a)}(E(s'))) \tag{4.1}$$

Сваки Марковљев процес одлучивања се може представити графом, тако да за избор модела функције $T$ можемо користити технике учења репрезентације графовских структура података [3, 27, 30, 31]. Избор функције преласка може утицати на квалитет наших репрезентација и бирамо је најчешће на начин који најбоље осликава симетрије које имамо у скупу података. Због њене једноставности у овом раду користићемо облик функције предложене у [3] у коме ће функција $T$ бити представљена транслацијом између стања:

$$T(z, a) = z + f_\theta(s, a) \tag{4.2}$$

, где је $f_\theta$ неуронска мрежа са параметрима $\theta$

Поред структуре окружења, коју у потпуности можемо описати функцијом транзиције, за потпун модел окружења неопходно је да имамо још две функције $R : \mathbb{R}^n \times \mathcal{A} \to \mathbb{R}$ и $D : \mathbb{R}^n \times \mathcal{A} \to [0, 1]$. Задатак функције $R$ биће да симулира награде у латентном простору што можемо записати на следећи начин:

$$(\forall s \in \mathcal{S})(\forall a \in \mathcal{A}(s))(R(E(s), a) = \mathbb{E}(r(s, a))) \tag{4.3}$$

Функција $D$ ће нам бити корисна у окружењима која имају завршно стање и служиће нам да разликујемо завршна стања од осталих. Како окружење не мора бити детерминистичко моделоваћемо следећу вероватноћу:

$$(\forall s \in \mathcal{S})(\forall a \in \mathcal{A}(s))(D(E(s), a) = P(s_n \text{ је завршно стање} | s_{n-1} = s, a_{n-1} = a)) \tag{4.4}$$

Тренинг модела окружења ће се састојати из два корака. Прво ћемо кретањем кроз окружење коришћењем неке (потенцијално случајне) политике сакупити низ интеракција. Ове интеракције ћемо запамтити као скуп уређених четворки (стање, предузета акција, ново стање, награда, да ли је стање завршно), а податке из тог скупа ћемо користити као тренинг скуп са којим ћемо тренирати функције $E$, $T$, $R$ и $D$.

Битно је приметити да је за константну функцију $E(s)$, и функцију $T(z, a) = z$ услов 4.1 испуњен. Како нам ово тривијално испуњавање овог услова не даје информације, функције $E$ и $T$ ћемо додатно ограничити како би избегли ово понашање. Постоје два основна приступа регуларизацији ових функција и њих ћемо изложити у секцијама 4.1 и 4.2.

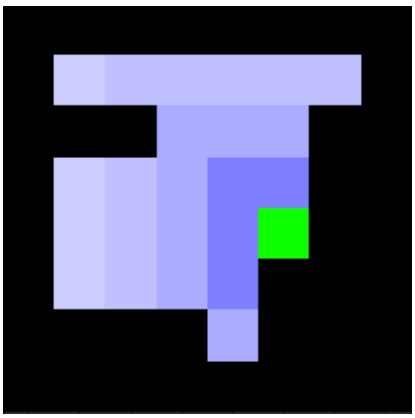
## 4.1 Модели окружења засновани на аутоенкодерима

Први начин да избегнемо тривијално решење једначине 4.1 је да се осигурамо да можемо извршити реконструкцију стања на основу његове латентне репрезентације. Овиме ћемо избећи решење које сва стања слика у исти вектор.

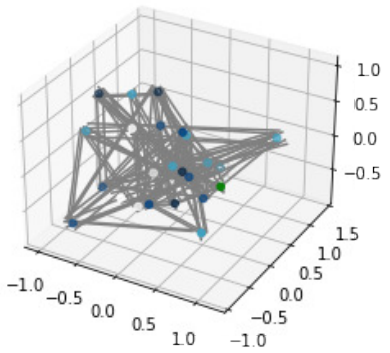
У оваквом приступу за функцију $E$ узећемо енкодер аутоенкодера, а како нам нормална расподела сличних улазних података која настаје коришћењем варијационих аутоенкодера одговара, они су најчешће коришћени код овог приступа што можемо видети у раду [7].

Функције $T$, $R$ и $D$ можемо учити на репрезентацијама које добијамо од раније истренираног аутоенкодера $E$, као што је рађено у раду [7] или можемо учити све четири функције истовремено. Првим приступом значајно умањујемо скуп хиперпараметара које морамо да бирамо - како сваку од ових функција одвојено тренирамо скупове хиперпараметара сваке од њих одвојено претражујемо. Други приступ нам може дати квалитетније репрезентације. Паралелним учењем функција $R$ и $T$ са функцијом $E$ латентни простор ће верније осликавати стварни скуп стања. Ипак приликом тестирања први приступ се показао значајно приступачнијим тако да смо се одлучили за њега.

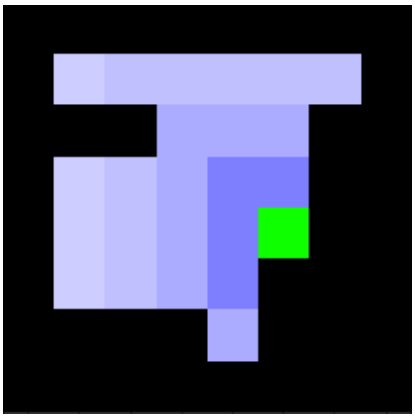
Пример структуре латентног простора можемо видети на слици 4.2.



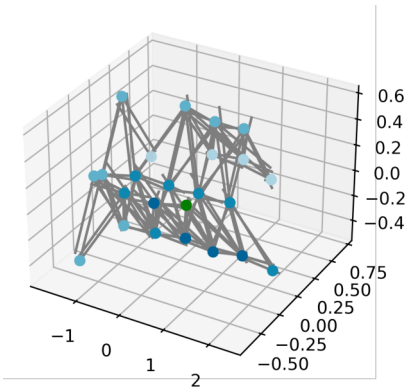
Слика 4.1: Лавиринт димензије 8x8 са пољима која су обојена у зависности од удаљености од циља




Слика 4.2: Латентни простор добијен коришћењем аутоенкодера на лавиринт лево, пројектован на 3 главне компоненте



Слика 4.3: Лавиринт димензије 8x8 са пољима која су обојена у зависности од удаљености од циља




Слика 4.4: Латентни простор добијен коришћењем аутоенкодера на лавиринт лево, пројектован на 3 главне компоненте

## 4.2 Модели окружења засновани на контрастивном учењу

Други приступ ограничења простора решења једначине 4.1 заснива се на техникама контрастивног учења. На основу скупа података који смо сакупили кретањем кроз окружење имамо информације да смо у стању $s$, применом акције $a$ дошли у стање $s'$. Једноставном минимизацијом функције $d(T(E(s), a), E(s'))$, као што смо већ рекли, лако долазимо до тривијалног решења које сва стања слика у исту тачку.

Ово можемо да избегнемо тиме што ћемо поред стања $s'$ у које смо дошли применом акције, посматрати и неко друго стање $\hat{s}$ у које нисмо дошли. Поред услова да функцијом транзиције желимо доћи што ближе стању $s'$, желимо да будемо удаљени од стања $\hat{s}$. Овај приступ се често користи у области учења метрика и функција коју ћемо тежити да минимизујемо биће

$$L(s, a, s', \hat{s}) = d(T(E(s), a), E(s')) + \max(0, 1 - d(T(E(s), a), E(\hat{s}))) \tag{4.5}$$

Како се овим приступом пажња посвећује само структури латентног простора, а не и могућности реконструкције оригиналног опажања, добијена структура је много уређенија него у случају тренирања аутоенкодерима. Пример структуре може се видети на слици 4.4. Обратимо пажњу на позицију циља (зелене боје) и распоред његових суседа (тамно плаве боје), можемо

приметити да је њихов распоред у оригиналном лавиринту 4.3 и у латентном простору 4.4 готово идентичан. Ово важи и за суседе осталих стања, тако да добијена структура у потпуности одговара структури почетног лавиринта, за разлику од структуре добијене применом аутоенкодера на слици 4.2.

Битно питање је како одредити стање $\hat{s}$ које ћемо користити приликом рачунања грешке. Уколико је то стање сувише слично стању $s'$ њихово раздвајање може бити контрапродуктивно, док избором стања које се исувише разликује од стања $s'$ функција се може опет тривијализовати (тако тренирана функција $E$ ће можда само разликовати значајно велика стања). Технике за избор погодних негативних примерака су предмет активног истраживања, а у нашем раду се показало довољно добрим коришћење стања у које би дошли применом неке друге акције у стању $s$, када нам је окружење то дозвољавало, и коришћењем насумичног стања из исте епизоде када то није било могуће.

# Глава 5

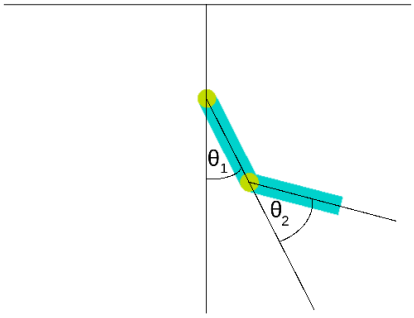
# Експерименти

У овој глави описаћемо експерименте које смо користили за оцењивање квалитета репрезентација добијених учењем латентних модела окружења. У првом делу главе описаћемо окружења у којима смо тренирали агенте, након тога ћемо описати начин избора хиперпараметара коришћеним у евалуацији и на крају ћемо описати које су метрике које смо пратили и начин на који смо дошли до резултата експеримената у терминима тих метрика приказаних у глави 6.

## 5.1 Посматрана окружења

### Акробот

Ово окружење смо изабрали као представника окружења код којих су стања која добијамо од окружења изражена у терминима малог скупа атрибута за које је познато да су релевантни. Отуд није потребно да модел открије нове релевантне атрибуте. Окружење представља симулацију двоструког клатна и агент од окружења добија информације о позицији и брзни клатна што можемо видети на слици 5.1. Агент може утицати на двоструко клатно применом позитивног или негативног угаоног момента фиксне вредности на фиксираном зглобу двоструког клатна, а може одлучити и да у тренутном кораку не предузме никакву акцију. Циљ у окружењу је да неки део двоструког клатна достигне одређену висину што можемо видети на слици 5.2. Агент ће од окружења добијати награду -1 приликом сваке интеракције, а како би све епизоде биле коначне дужине, ограничени смо на 500 интеракција са окружењем до




Слика 5.1: Слика окружења Акробот, опажање се састоји од $(\sin(\theta_1), \cos(\theta_1), \sin(\theta_2), \cos(\theta_2), \frac{d\theta_1}{dt}, \frac{d\theta_2}{dt})$

Слика 5.2: Пример завршног стања окружења

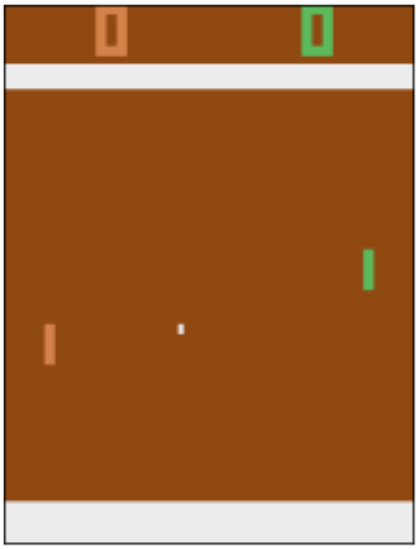
краја епизоде.

Како би дошли до завршног стања биће потребно да применом одговарајућег обртног момента у одговарајућим тренуцима зањишемо клатно довољно да оно достигне тражену висину. Уколико би у сваком тренутку вршили оптималну акцију било би потребно бар 50 корака да би дошли до завршног стања, а сваки погрешан корак овај број значајно повећава. Ова чињеница представља највећу тешкоћу приликом тренирања агента. Пошто је потребно доћи до завршног стања како би агент сазнао који је његов циљ он може дуго лутати док не дође први пут до ове информације.

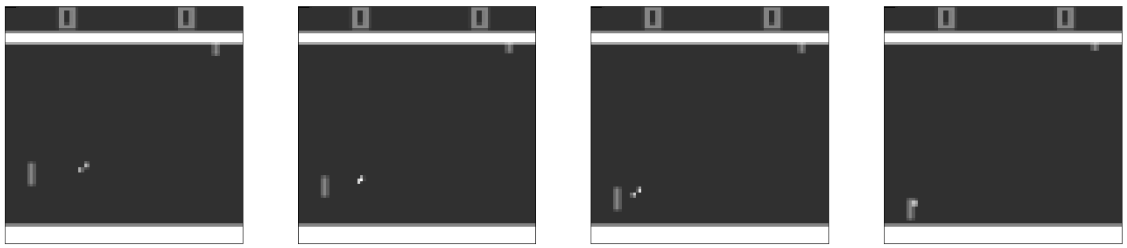
## Понг

Ово окружење изабрали смо као представника окружења у коме се учи на основу сирове репрезентације слике у виду пиксела. Отуд модел сам треба да научи релевантне промењиве проблема. Окружење представља једну од најпопуларнијих игрица за систем Atari2600. У овој игрици контролишемо рекет којим одбијамо лоптицу и циљ нам је да лоптица удари у зид који се налази иза противничког рекета.

Опажања која добијамо од окружења су фрејмови игрице и један његов пример можемо видети на слици 5.3. Можемо приметити да један фрејм игрице није довољан да се закључе брзина и смер лоптице, тако да окружење само

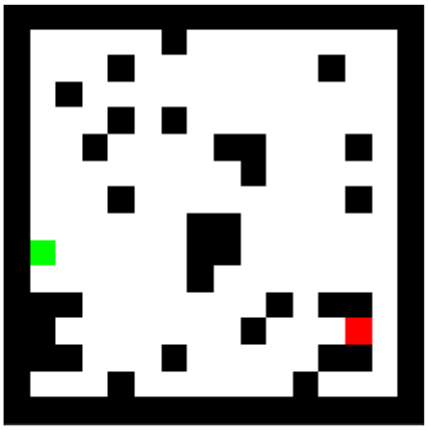


Слика 5.3: Слика окружења Понг



Слика 5.4: Слика окружења Понг након извршених трансформација

по себи није Марковљево. Како би се изборили са овим, уместо само тренутног опажања, агенту ћемо прослеђивати последња четири фрејма игрице. Поред ове модификације окружења, пребацивање слике у боји у црно-белу слику, као и скалирање слике на квадрат димензије $84 \times 84$ пиксела су се усталили као типични кораци претпроцесирања који олакшавају тренирање агента на Атари окружењима. Пример овако добијеног опажања можемо видети на слици 5.4. Акције које можемо предузети у овом окружењу су померање рекета горе или доле и остављање рекета у истој позицији. Награде које добијамо су -1 за сваки примљени го, +1 за сваки дати го и 0 у свим осталим ситуацијама.



Слика 5.5: Слика окружења лавиринт - циљ је обележен зеленом бојом, а агент црвеном

## Лавиринт

Ово окружење је пример мрежних окружења [4] у којима је планирање врло често један од кључева за њихово успешно решавање. Окружење се састоји од табеле 16 × 16 која представља лавиринт и наш циљ ће бити спровођење агента до циља. Пример опажања овог окружења можемо видети на слици 5.5.

Свако опажање се састоји од 3 матрице нула и јединица димензије 16 × 16. Прва матрица описује где се налазе зидови лавиринта, друга матрица описује где се налази циљ и трећа матрица описује положај агента. Позиције зидова, као и позиције циљева су преузете из скупа података коришћеног у раду [29]. У сваком кораку агент ће покушати да пређе на једно од 8 суседних поља. Уколико агент покуша да се помери на поље са зидом добија награду -1 и епизода се завршава. Долазак до циља доноси награду од 1, а како би подстакли агента да направи најмањи број корака прелазак на празно поље доноси награду -0.01.

Као и у случају окружења Акробот, чињеница да се награда добија тек када достигнемо циљ представља велику препреку у тренирању успешних агената. Такође уколико агент случајно лута кроз окружење, он врло лако може доћи до зида што доводи до прекидања епизоде. Ово додатно смањује шансе за долазак до циља и самим тим чини почетак тренинга врло нестабилним. Како ми можемо контролисати позиције са којих ће агент кренути у лавиринту, можемо олакшати тренинг тако што ћемо на почетку давати агенту

позиције у којима почиње директно поред циља. Када агент научи да долази до циља у бар 90% случајева повећаћемо максимално растојање од циља на коме ће агент почињати епизоду. Максимално растојање на коме агент може почети епизоду зваћемо ниво лавиринта, а повећање максималног растојања називаћемо прелазак нивоа.

## 5.2 Претрага хиперпараметара

За претрагу хиперпараметара коришћена је популарна библиотека optuna [1]. У овом делу рада приказаћемо како смо користили ову библиотеку за претрагу хиперпараметара.

Библиотека optuna нам даје могућност за аутоматску претрагу скупа хиперпараметара које дефинишемо у коду. Посебну популарност је добила на такмичењима на платформи Kaggle због динамичког одређивања скупа хиперпараметара што омогућава једноставну интеграцију у већ постојећи кôд.

Основни објекат у овој библиотеци је `study` и он ће водити рачуна о свим хиперпараметрима које смо до сада користили, као и резултатима које смо добили. Пример инстанцирања објекта `study: Study` можемо видети у следећем коду:

```
sampler = optuna.samplers.TPESampler(seed=args.sampler_seed)
study = optuna.create_study(
    study_name=study_name, # idetifikator nase studije
    direction='maximize',
    storage=storage_path,  # baza podataka za cuvanje rezultata
    sampler=sampler,
    load_if_exists=True)

...

study.optimize(objective, n_trials=num_trials)
```

Код 5.1: Пример инстанцирања објекта `study`

Као што видимо покушаћемо да оптимизујемо циљ `objective` који може бити функција или функтор који прима само један параметар `trial`. Овај параметар ће нам прослеђивати објекат `study` и његова улога је предлагање нових

хиперпараметара које желимо да тестирамо. Пример како ћемо користити овај објекат можемо видети у следећем коду:

```
1  import torch.nn as nn
2  def define_model(trial):
3      n_layers = trial.suggest_int(''n_layers'', 1, 3)
4      layers = []
5
6      in_features = 28 * 28
7      for i in range(n_layers):
8          out_features = trial.suggest_int(f''n_units_l{i}'', 4, 128)
9          layers.append(nn.Linear(in_features, out_features))
10         layers.append(nn.ReLU())
11         p = trial.suggest_float(f''dropout_l{i}'', 0.2, 0.5)
12         layers.append(nn.Dropout(p))
13
14         in_features = out_features
15     layers.append(nn.Linear(in_features, CLASSES))
16     layers.append(nn.LogSoftmax(dim=1))
17
18     return nn.Sequential(*layers)
```

Код 5.2: Пример коришћења објекта `trial`

Можемо приметити да смо у линијама 3, 8 и 11 користили методе `suggest_int`, односно `suggest_float`. Ове, као и многе друге методе сличног имена, предлагаће нам нове хиперпараметре за тестирање, а за избор хиперпараметара за тестирање биће задужен `sampler`. Подразумевано се користи `TPESampler` који користи Парзенов дрволики оцењивач (eng. Tree-structured Parzen estimator, TPE) [20] за предлагање нових хиперпараметара, а уколико желимо да му поставимо семе генератора случајних бројева, или да променимо хеуристику за избор хиперпараметара, можемо га проследити функцији `optuna.create_study` као што смо урадили у примеру 5.2.

Поред избора нових хиперпараметара за тестирање, објекат `trial` користимо за пријављивање тренутне вредности функције циља позивом метода `report(self, value: float, step: int)-> None`. Доста често у току тренинга можемо закључити да су перформансе тренутног скупа хиперпараметара доста лошије од до сада виђених вредности. У том случају има смисла тренирање модела превремено прекинути, а како смо помоћу метода `report` пријављива-

ли перформансе модела, ову одлуку можемо аутоматски доносити на основу неке хеуристике. Метод који ћемо користити како би сазнали да ли тренирање треба превремено завршити је `should_prune(self)-> bool`. Ова функција ће подразумевано вратити `True` уколико је тренутна вредност гора од медијане досадашњих вредности у том кораку. Уколико желимо да променимо ово понашање, као и у случају `sampler`-а можемо проследити нашу класу објекту `study`.

Коришћењем свих горе поменутих функција `objective` функција за максимизацију тачности неког класификатора може изгледати овако:

```
def objective(trial):
    model = define_model(trial)

    optimizer_name = trial.suggest_categorical(''optimizer'',
        [''Adam'', ''RMSprop'', ''SGD''])
    lr = trial.suggest_float(''lr'', 1e-5, 1e-1, log=True)
    optimizer = getattr(optim, optimizer_name)(model.parameters(),
        lr=lr)

    train_data, valid_data = get_data()

    for epoch in range(EPOCHS):
        model.train()
        for data, target in train_data:
            optimizer.zero_grad()
            output = model(data)
            loss = loss(output, target)
            loss.backward()
            optimizer.step()

        model.eval()
        correct = 0
        with torch.no_grad():
            for data, target in valid_loader:
                output = model(data)
                pred = output.argmax(dim=1, keepdim=True)
                correct +=
                    pred.eq(target.view_as(pred)).sum().item()
    accuracy = correct / len(valid_data)

    trial.report(accuracy, epoch)
```

```
29
30      if trial.should_prune():
31          raise optuna.exceptions.TrialPruned()
32
33      return accuracy
```

Код 5.3: Пример дефиниције функције `objective`

## 5.3 Евалуација

Приликом евалуације агената пратићемо како се циљна метрика мења са бројем интеракција са окружењем. Ово ће нам омогућити да пратимо утицај модела окружења у различитим фазама тренинга. Циљна метрика ће у окружењима Понг и Акробот бити просечна укупна награда коју агент остварује у једној епизоди, док ћемо у окружењу лавиринт пратити колико брзо прелазимо нивое лавиринта.

Како би избегли оцењивање перформанси агента на истом окружењу на коме је и трениран, приликом тренирања агента смо користили различита семена генератора псеудо случајних бројева (енг. seed of pseudo random number generator) у окружењима за тренинг и валидацију. Такође, како пријављене метрике не би биле везане само за један пар семена генератора псеудо случајних бројева, приликом евалуације смо користили 10 различитих парова семена генератора псеудо случајних бројева. На графицима у глави 6 биће приказани просек и стандардна девијација измерених метрика.

За избор хиперпараметара користили смо билиотеку `optuna` и за сваког агента и за свако окружење смо покушавали по 150 различитих конфигурација. Од тога за агенте који су користили латентне моделе окружења 50 конфигурација је посвећено избору хиперпараметара модела окружења, а преосталих 100 избору хиперпараметара самог агента. Семена генератора псеудо случајних бројева за избор хиперпараметара, као и семе за генерисање интеракција са окружењем које су коришћене за тренирање модела, разликовала су се од семена за евалуацију перформанси модела како би у што већој мери избегли непримећено преприлагођавање. У наставку ове главе налазе се хиперпараметри који су разматрани, као и семена генератора псеудо случајних бројева коришћена за добијање резултата које можемо видети у глави 6.

## 5.4 Тестирани хиперпараметри

У овом делу рада биће наведени хиперпараметри модела које смо тестирали. При запису архитектура мреже са $D(n_1, n_2, ..., n_m)$ подразумеваћемо низ потпуно повезаних слојева неуронске мреже са $n_1, n_2, ..., n_m$ неурона у слоју, са $C(n_1, n_2, ..., n_m)$ подразумеваћемо низ конволутивних слојева са $n_1, n_2, ..., n_m$ филтера по слоју и са $DC(n_1, n_2, ..., n_m)$ подразумеваћемо низ транспонованих конволутивних слојева са $n_1, n_2, ..., n_m$ филтера по слоју. За латентне моделе окружења, поред модела које смо описали у глави 4, бирали смо одговарајућу димензију латентног простора, у табелама обележену са Дим., као и стопу учења (енг. learning rate), у табелама обележену са с.у. За агенте смо бирали број скривених слојева између репрезентације добијене моделом окружења и линеарног слоја коришћеног за агента, и критичара у алгоритму PPO 3.

У табелама 5.1, 5.2 и 5.3 приказани су хиперпараметри коришћени за тренирање латентног модела окружења заснованог на контрастивном учењу, латентног модела окружења заснованог на аутоенкодерима и агента. У случају агента који није користио претренирани енкодер, поред архитектуре агента бирана је и архитектура енкодера. Тестиране конфигурације енкодера су исте као у случају модела окружења заснованог на контрастивном учењу и могу се видети у табели 5.1.

| **Акробот** | | | | |
|---|---|---|---|---|
| Дим. | Енкодер | Транзиција | Награда | С.у. |
| 3 | D(16) | D(16) | D(16) | |
| 4 | D(16-32) | D(16-32) | D(16-16) | $[10^{-2}, 10^{-5}]$ |
| 6 | D(16-32-32) | D(16-32-32) | | |
| **Понг** | | | | |
| Дим. | Енкодер | Транзиција | Награда | С.у. |
| 256 | C(32-64-32)-D(256) | D(512) | D(512) | |
| 512 | C(32-64-32)-D(256-256) | D(512-512) | D(512-512) | $[10^{-2}, 10^{-5}]$ |
| 1024 | C(32-64-32)-D(256-256-256) | D(512-512-512) | | |
| **Лавиринт** | | | | |
| Дим. | Енкодер | Транзиција | Награда | С.у. |
| 10 | D(256-256-128) | D(32) | D(16) | |
| 16 | C(8-16)-D(256-128) | D(16-32) | D(16-32) | $[10^{-2}, 10^{-5}]$ |
| 32 | C(8-16-16)-D(256-128) | D(16-32-32) | | |

Табела 5.1: Хиперпараметри модела окружења заснованог на контрастивном учењу

| **Акробот** | | | | | |
|---|---|---|---|---|---|
| Дим. | Енкодер | Декодер | Транзиција | Награда | С.у. |
| 3 | D(16) | D(16) | D(16) | D(16) | |
| 4 | D(16-32) | D(16-32) | D(16-32) | D(16-16) | $[10^{-2}, 10^{-5}]$ |
| 6 | D(16-32-32) | D(16-32-32) | D(16-32-32) | | |
| **Понг** | | | | | |
| Дим. | Енкодер | Декодер | Транзиција | Награда | С.у. |
| 256 | C(32-64-32)-D(256) | D(256)-DC(32-64-32) | D(512) | D(512) | |
| 512 | C(32-64-32)-D(256-256) | D(256-256)-DC(32-64-32) | D(512-512) | D(512-512) | $[10^{-2}, 10^{-5}]$ |
| 1024 | C(32-64-32)-D(256-256-256) | D(256-256-256)-DC(32-64-32) | D(512-512-512) | | |
| **Лавиринт** | | | | | |
| Дим. | Енкодер | Декодер | Транзиција | Награда | С.у. |
| 10 | D(256-256-128) | D(256-256-128) | D(32) | D(16) | |
| 16 | C(8-16)-D(256-128) | D(256-128)-DC(8-16) | D(16-32) | D(16-32) | $[10^{-2}, 10^{-5}]$ |
| 32 | C(8-16-16)-D(256-128) | D(256-128)-DC(8-16-16) | D(16-32-32) | | |

Табела 5.2: Хиперпараметри модела окружења заснованог на аутоенкодерима

| **Акробот** | **Понг** | **Лавиринт** | С.у. |
|---|---|---|---|
| - | - | - | |
| D(16) | D(256) | D(64) | $[10^{-2}, 10^{-5}]$ |
| D(16, 16) | D(256, 256) | D(64, 64) | |

Табела 5.3: Хиперпараметри агента

У табели 5.4 приказана су семена генератора случајних бројева коришћених за (1) генерисање интеракција коришћених за тренирање латентног модела окружења, (2) тренинг окружења приликом избора хиперпараметара агента, (3) валидациона окружења приликом избора хиперпараметара агента, (4) тренинг окружења приликом евалуације агента и (5) валидациона окружења приликом евалуације агента.

| **Интеракције** | **Тренинг хп.** | **Валидација хп.** | **Тренинг евал.** | **Валидација евал.** |
|---|---|---|---|---|
| 42 | 50 | 100 | 200, 220, 240, 260, 280, 300, 320, 340, 360, 380 | 500, 520, 540, 560, 580, 600, 620, 640, 660, 680 |

Табела 5.4: Семена генератора псеудо-случајних бројева

# Глава 6

# Резултати и дискусија

У овој глави приказујемо резултате добијене приликом тренинга агента помоћу различитих модела окружења за окружења Акробот, Понг и Лавиринт.

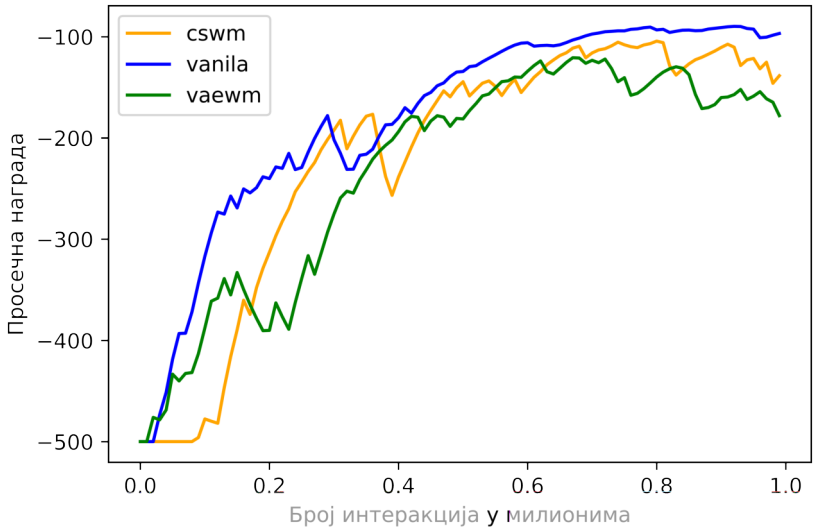
## Акробот

Као што смо најавили у уводу, учење у овом окружењу је врло изазовно због проређености награде. Приликом претраге хиперпараметара, као и приликом евалуације, велики број агената никада није дошао до циља окружења, тако да је кретање ових агената представљало насумичан одабир акција.

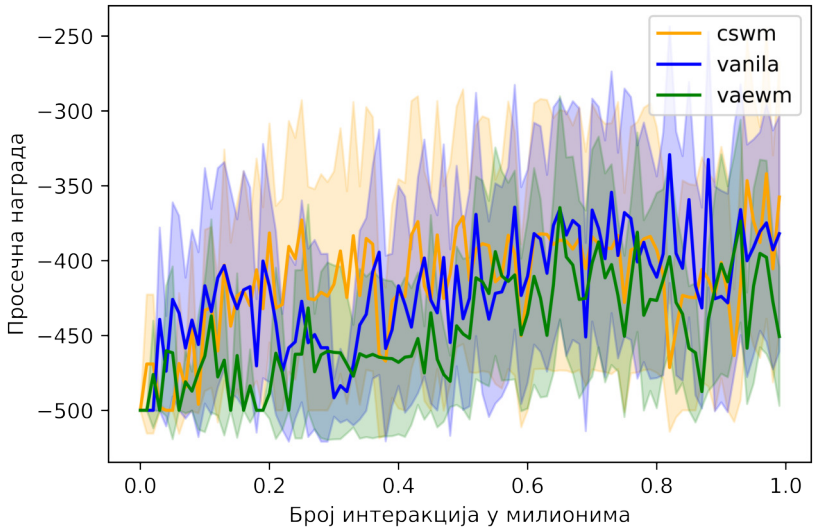
Како окружење само по себи није претерано сложено, након што би агент добио прву позитивну награду, углавном би релативно брзо напредовао у окружењу. Ово значи да успешност агента практично зависи само од времена потребног за прво успешно решавање окружења. Док графици просечне награде најбољих агената најчешће личе на график 6.1[1], најгори агенти у току целог тренинга имају награду -500 (најмању могућу). Ово својство резултује нестабилним тренингом са веома великом варијансом што можемо видети на графику 6.2[1].

Можемо приметити да нема суштинских разлика између ових агената. Чак и ако гледамо перформансе најбољих агената оне личе једне на друге. Разлог овоме можемо пронаћи у чињеници да стања која добијамо од окружења имају мали скуп атрибута и знамо да су сви атрибути корисни. Ово значи да нове

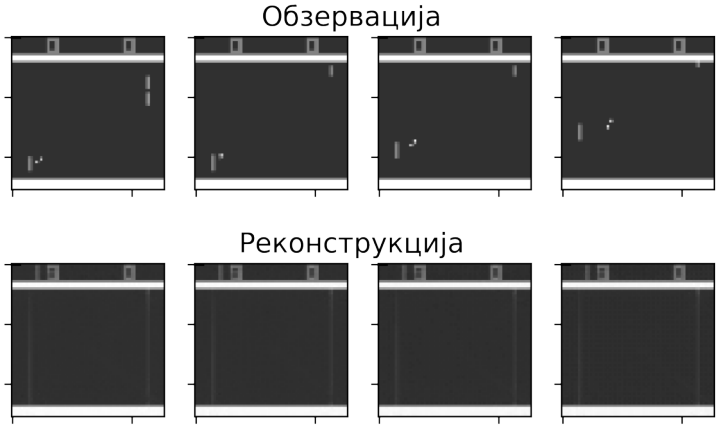
[1]На графицима смо резултате класичног агента, агента добијеног коришћењем латентног модела окружења заснованог на контрастивном учењу и агента добијеног коришћењем латентног модела окружења заснованог на аутоенкодерима обележавали са VANILA, CSWM и VAEWM, редом




Слика 6.1: Награда најуспешнијег агента у окружењу Акробот




Слика 6.2: Просечна награда агента у окружењу Акробот




Слика 6.3: Реконструкција једног опажања окружења Понг применом варијационог аутоенкодера

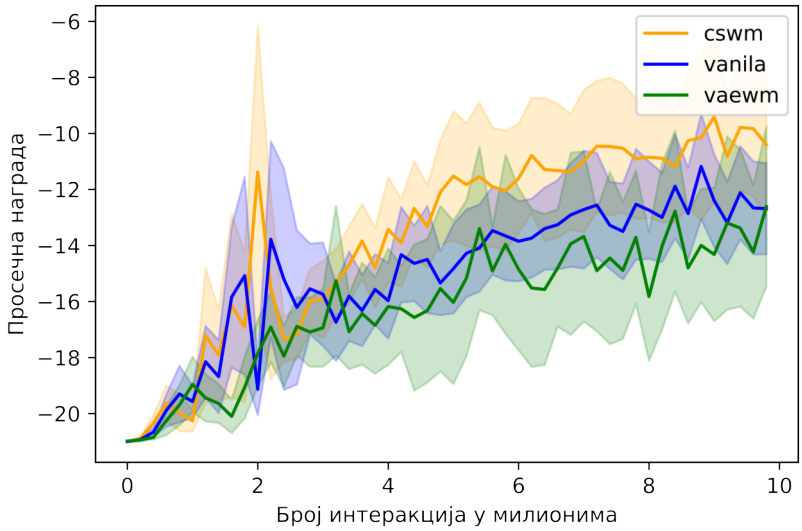
репрезентације које добијамо латеним моделима окружења неће имати велике предности у односу на стања која добијамо од окружења, па су и резултати учења на овим репрезентацијама слични.

## Понг

Битно је приметити да се опажања код овог окружења већински састоје од позадине, док су најбитнији делови екрана (рекети и лоптице) релативно мали. Од улазне слике која је димензије $84 \times 84$ пиксела, рекети заузимају по 14, а лоптица само 4 пиксела. То значи да за нас битан садржај представља мање од 1% слике. Варијациони аутоенкодери имају тенденцију да овако мале детаље игноришу, што можемо видети на слици реконструкције опажања 6.3.

Поред опажања које је изазовно за варијационе аутоенкодере, разлог овако лоше реконструкције може бити и у избору архитектуре неуронске мреже. Хиперпараметри које смо користили за енкодер су већински преузети из рада [19] и представљају стандардне хиперпараметре за учење играња Атари игрица. Мењање архитектуре неуронске мреже коришћене за агента мења њен капацитет, што ће неминовно имати утицај на перформансе агента. Како нисмо хтели да мењамо класичног агента, за кога је ова архитектура установљена као добра, ову архитектуру смо оставили и за латентне моделе окружења.

Проблем код овако добијене репрезентације је то што она свакако неће бити корисна за нашег агента (осим резултата који делимично разликујемо, рекети



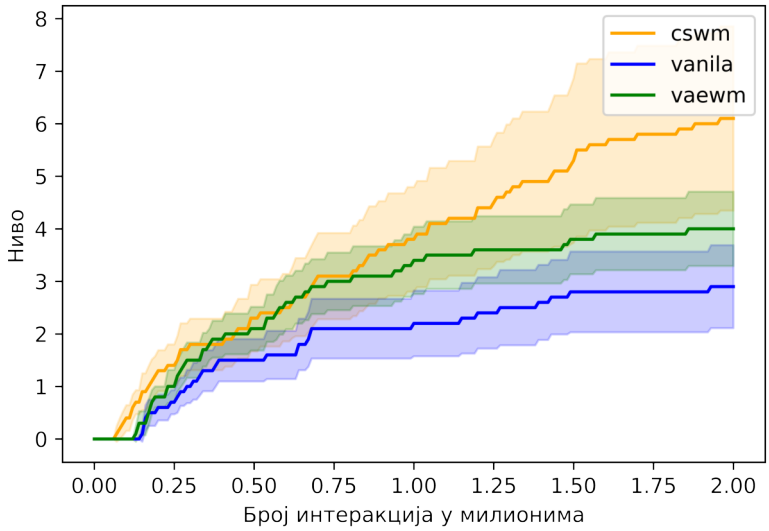

Слика 6.4: Просечна награда у окружењу Понг

и лоптице очигледно нису присутни у реконструкцији). Из овог разлога смо у овом окружењу дозволили да се параметри модела окружења такође мењају у току тренинга агента.

Можемо видети на графику 6.4 да се модел окружења заснован на варијационим аутоенкодерима (VAEWM) најгоре показао, као што је очекивано. Претренирање варијационим аутоенкодером довело је до тога да смо изгубили тачну позицију лоптице и рекета у репрезентацији коју користи агент, а ове информације су кључне приликом играња игре. Ово значи да је за успешно играња игрице неопходно прво изменити параметре енкодера, и тек онда ће агент моћи да научи неке корисне информације из новодобијених репрезентација.

За разлику од агента који користи енкодер из VAEWM, код класичног агента (VANILA) параметри енкодера су на почетку тренинга иницијализовани случајним вредностима. Репрезентације које агент добија од овако случајно иницијализоване мреже неће бити корисне, али највероватније неће изгубити информације о положајима лоптице и рекета. Из тог разлога, као што можемо видети на графику 6.4, агент VANILA је имао бољи резултат од агента VAEWM.

Приликом учења латентног модела окружења заснованог на контрастивном учењу (CSWM) кључна информација за учење функције транзиције биће




Слика 6.5: Просечан напредак по нивоима у окружењу лавиринт

положај рекета и лоптице, ово нам гарантује да ће ове информације бити присутне у репрезентацији добијеној енкодером овог модела окружења. Можемо приметити да су перформансе агента CSWM и VANILA врло сличне у првој трећини тренинга, док након тога агент CSWM постаје бољи. У првом делу тренинга агент је учио како исправно да искористи добијене репрезентације, тако да можемо да закључимо да су репрезентације добијене енкодером заиста квалитетне и да представљају разлог за ово побољшање перформанси.

## Лавиринт

У овом окружењу највећи проблем представљају зидови, зато што покушај преласка на поље са зидом завршава епизоду уз негативну награду. Агент на почетку тренинга неће имати много сусрета са зидовима, што значи да агент неће имати потребу да научи шта је потребно урадити приликом сусрета са њима све док не кренемо да га постављамо на већим растојањима од циља. На графику 6.5 можемо приметити да је класичан агент (VANILA) имао проблем да научи ову интеракцију. На удаљеностима већим од 3-4 корака од циља чешће наилазимо на зидове, а како агент није научио да их заобилази, имао је потешкоће са конзистентним доласком до циља.

Агент који је добијен коришћењем варијационих аутоенкодера (VAEWM)

се боље показао, разлог овоме је највероватније компактнија репрезентација на којој је агент учио. На почетним нивоима, како епизоду почињемо на малим удаљеностима од циља, препознавање релативних позиција агента и циља и директно кретање ка циљу је најбоља стратегија. Агент VAEWM се показао бољим од VANILA на овим почетним нивоима, одакле можемо закључити да је учење препознавања релативних позиција агента и циља из репрезентација добијених варијационим аутоенкодером било лакше од препознавања релативних позиција агента и циља на основу сирове слике.

Можемо приметити и да је око удаљености од 4-5 корака до циља агент стагнирао. Видели смо на слици 4.2 да распоред репрезентација стања у латентном простору није превише уређен. Чак и ако је агент успешно научио како да препозна циљ у овом простору, учење положаја зидова се показало као сувише изазовно на овако неуређеним репрезентацијама.

Агент који је добијен коришћењем контрастивног учења (CSWM) се зато најбоље показао у овом окружењу. Видимо да је на мањим растојањима он сличан агенту VAEWM, што додатно потврђује чињеницу да је на тим мањим растојањима најбитније препознати релативну позицију циља и агента, али након тога уређеност латентног простора коју смо видели на слици 4.4 омогућава препознавање и избегавање зидова у окружењу.

# Глава 7

# Закључак

Латентни модели окружења нам дају могућност да научимо нове репрезентације података. У овом раду приказали смо основни начин на који овако добијене репрезентације можемо да користимо у учењу поткрепљивањем.

Приметили смо да су репрезентације добијене контрастивним учењем врло корисне и да нам могу дати значајна побољшања у перформансама агента. Репрезентације добијене коришћењем варијационих аутоенкодера се нису добро показале приликом тренирања агента на окружењу Понг, али смо видели да могу бити корисне у окружењу лавиринт. Такође смо видели да, очекивано, учење нове репрезентације података неће донети драстичне промене у перформансама уколико је стање које добијамо од окружења представљено малим скупом корисних атрибута.

Битно је приметити да смо у овом раду анализирали утицај самих репрезентација добијених од латентних модела окружења на процес тренинга. Модел окружења ван тога нисмо користили. Примене латентих модела окружења у планирању, као и њихово коришћење у симулацијама окружења остали су ван оквира овог рада и предмет су активних истраживања.

# Литература


[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] Aqeel Anwar. Difference between Autoencoder (AE) and Variational Auto-Encoder (VAE). `https://tinyurl.com/4red7uwe`, 2021.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[4] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic Gridworld Environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[7] David Ha and Jürgen Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018.

[8] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and

David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning, 2017.

[9] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[10] Shengyi Huang and Santiago Ontañón. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *CoRR*, abs/2006.14171, 2020.

[11] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes, 2013.

[12] Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[13] Ilya Kostrikov. PyTorch Implementations of Reinforcement Learning Algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[15] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. Maritime Semantic Labeling of Optical Remote Sensing Images with Multi-Scale Fully Convolutional Network. *Remote Sensing*, 9:480, 05 2017.

[16] Scott Martin. What's the Difference Between a CNN and an RNN? `https://tinyurl.com/3c2pcpy8`, 2018.

[17] Warren Mcculloch and Walter Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[18] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, abs/1602.01783, 2016.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, 2013.

[20] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. Multiobjective Tree-Structured Parzen Estimator for Computationally Expensive Optimization Problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 533–541, New York, NY, USA, 2020. Association for Computing Machinery.

[21] Yongmei Ren, Jie Yang, Qingnian Zhang, and Zhiqiang Guo. Multi-Feature Fusion with Convolutional Neural Network for Ship Classification in Optical Images. *Applied Sciences*, 9(20), 2019.

[22] G. A. Rummery and M. Niranjan. On-Line Q-Learning Using Connectionist Systems. Technical report, 1994.

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, 2015.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[26] B. F. SKINNER. The experimental analysis of behavior. *American Scientist*, 45(4):343–371, 1957.

[27] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space, 2019.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[29] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. *CoRR*, abs/1602.02867, 2016.

[30] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction, 2016.

[31] Pengcheng Yin and Graham Neubig. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation, 2018.

[32] Петар Величковић. TikZ. `https://github.com/PetarV-/TikZ`, 2022.

# Биографија аутора

**Огњен Д. Милинковић** рођен је 05.03.1998. године у Београду. Основну и средњу школу завршио је у Математичкој гимназији у Београду. Основне академске студије уписао је на смеру рачунарство и информатика на Математичком факултету 2016. године. На наведеном смеру дипломирао је 2020. године са просеком 9,90 након чега је на истом смеру уписао и мастер академске студије. Од октобра 2020. држи вежбе на Математичком факултету на позицији сарадника у настави. На конференцији NeurIPS (Conference on Neural Information Processing Systems), 2021. године био је коаутор рада „Neural Algorithmic Reasoners are Implicit Planners". Области његовог интересовања укључују компајлере и машинско учење.