МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД
из предмета
Рачунарство и информатика

на тему

Карактеристичне примене машинског учења на самовозеће аутомобиле

Ученик:
Филип Весовић, $4_{\text{д}}$

Ментор:
Петар Величковић

Београд, мај 2017.

Садржај

Глава 1

Увод

Да ли знате да су друштвене мреже настале пре нешто више од десет година? Да су мобилни телефони и интернет настали пре око 25 година? Рачунари пре 70 година? Аутомобили пре око 130 година? Парна машина пре око 250 година? А да ли знате да је људска врста стара око 3 милиона година? Жеља за бољим је увек покретала људску врсту напред, од изласка из пећина, проналаска ватре и точка, па до данас. Међутим, тај развој је постајао временом све бржи и бржи. Све оно што ми данас сматрамо основом нашег живота настало је пре занемарљиво мало времена у поређењу са трајањем људске врсте. Овај рад је превасходно концентрисан на технологије машинског учења, које тренутно доживљавају вртоглаву брзину развоја, и њихове примене унутар аутономних возила, који представљају један од највећих отворених проблема 21. века. Са методолошког становништа, највећа пажња ће бити посвећена неуралним мрежама, које представљају носилац овог развоја у протеклим годинама.

Аутомобили су настали у 19. веку када је Карл Бенц конструисао први аутомобил који је уместо вуче коња користио мотор са унутрашњим сагоревањем. Убрзо, они су променили свет. Спојили су људе и омогућили им већу слободу кретања. Данас је немогуће замислити живот без њих. Међутим, док се свет мењао и напредовао под утицајем аутомобила, они су у суштини остали исти. Како и у 19 в. тако и данас, имају точкове, мотор, управљач, место намењено возачу који управља возилом... Идеја за аутомобиле без возача је постојала одавно, али је од почетка било јасно да се овоме не може приступити као традиционалном оптимизацијском проблему: ефективно и безбедно управљање моторним возилом неретко укључује доношење изузетно тешких одлука, које нису увек лако образложиве. Управо је употреба вештачке интелигенције за решавање важних потпроблема у овом простору и довела до највећих продора у претходним годинама; дотле да компаније попут Тесле већ имају половично аутономна возила на улицама.

1.1 Предности и изазови

Предности самовозећих аутомобила су бројне. Као прво, самовозећи аутомобили имају потенцијал да буду много безбеднији од класичних аутомобила. Бројне камере, радари, и системи уграђени у њих имају свакако бољу могућност да брже и исправније реагују од људи. Такође, самовозећи аутомобили би пружили олакшање људима који не воле да возе, али и дали више слободног времена за обављање других дневних обавеза, док би их њихов ауто возио ка одредишту. Развој самовозећих аутомобила би омогућио унапређење саобраћајног система кроз повећање максималне брзине вожње што би смањило време утрошено у саобраћају. Разлог за то је, поред тога што су самовозећи аутомобили сигурнији, отварање могућности да они међусобно комуницирају и на тај начин размењују информације везане за њихову планирану путању што доводи до бољег функсионисања целог система. Временом, може се очекивати да ће самовозећи аутомобили потпуно заменити класичне аутомобиле, као и то да ће аутомобили са мануелним управљањем постати забрањени ради безбедности и несметаног функционисања система. Ипак, управљање аутомобилима вероватно неће остати заборављено, него ће прећи у хоби и занимацију. Можемо очекивати да ће у будућности постојати паркови, градови па и читаве државе где ће аутомобиле и даље возити људи.

Током претходних година сведоци смо да многе светске компаније међу којима су Tesla, Google, NVIDIA и Uber покушавају да изграде прве самовозеће аутомобиле. Међутим, стварање система способног да се самостално креће у саобраћају је изузето тешко из више разлога:

- систем не сме да има простора за грешку из разлога што и најмања грешка може да изазове огромну штету (смрт људи, материјалну штету)
- постоје различити услови у којима систем треба да ради (дан, ноћ, слаба осветљеност, јака осветљеност, киша, магла, снежне падавине, снег по коловозу).
- систем мора да буде безбедан и заштићен од злонамерних напада на њега (заштита система од извршавања малициозног кода, као и отпорност на евентуалне намерне "замке" постављене на путу)
- постојање етичких дилема које још увек нису решене. На пример:
  - Испред аутомобила се налази камион са теретом који није добро причвршћен, у једном тренутку са тог камиона спада терет, ваш ауто има две опције: остаће на истом путу и тај терет ће убити вас, или ће скренути на страну и убити случајног пролазника. Уколико би изабрао прву опцију, да ли бисте се возили

у аутомобилу предпрограмираном да вас убије, а са друге стране, да ли бисте ишли улицом ако поред пролазе ауто-убице?

- Наш ауто долази у ситуацију у којој преостају само две опције, скренути и ударити у мотоциклисту или скренути и ударити у аутомобил пун људи. Ако ударимо у мотоциклисту, он има мале шансе да преживи, са друге стране, ако ударимо у ауто, повредићемо више људи који ће имати веће шансе да преживе, али потенцијално и убити више људи.
- Слично претходној ситуацији имамо да бирамо између двојице мотоциклиста, један носи кацигу, док је други не носи. Да ли треба ударити особу са кацигом која има шансу да преживи и на тај начин је наградити за поштовање закона, или особу без кациге која ће готово сигурно погинути?
- Уколико наш ауто убије човека, ко треба да буде одговоран? Аутор алгоритма због немогућности алгоритма да исправно реагује у реалном окружењу? Имплементатор система због постојања грешке у имплементацији система? Произвођач због лошег или недовољно доброг очитавања података о окружењу? Или особа у аутомобилу? Тренутно је веома тешко извући интерпретабилна правила закључивања из неуралних мрежа и сличних система на основу којих би било могуће да се утврди кривица.

1.2 Подела према нивоу аутоматизације

Аутомобиле можемо поделити у пет категорија по нивоу аутоматизације:

1. Аутомобил без било какве врсте аутоматизације
2. Аутомобил са врло мало аутоматизације (то могу бити кочнице како би се спречио судар)
3. Аутомобил способан да сам вози при одређеним условима (то може бити систем за праћење линија и спречавање судара какав би могао да се користи на аутопуту)
4. Скоро потпуно аутоматизовани аутомобили (ови аутомобили би били способни да у већини случајева функционишу сами, али би присуство возача било обавезно)
5. Потпуно аутоматизовани аутомобили (аутомобили који би могли да се крећу без присуства возача)

Тренутно већина доступних система имплементира трећи ниво аутоматизације, док многе компаније активно тестирају четврти ниво.

1.3 Тема рада

У овом раду биће разјашњене теоријске основе неуралних мрежа, као и неке коришћене методе при њиховој изради, а онда и специфични проблеми који се сусрећу приликом израде самовозећих аутомобила и решења тих проблема уз помоћ неуралних мрежа са анализом успешности.

Неки од проблема са којима ћемо се бавити су:

1. Препознавање саобраћајних знакова
2. Детекција саобраћајних знакова на слици
3. Детекција пешака, аутомобила, улице, и зграда
4. Управљање аутомобилом у симулатору
5. Самостално учење управљања уз помоћ учења са појачавањем

Ови проблеми представљају само део онога што аутономни системи треба да реше како би били способни за самосталну вожњу. Такође, они морају бити способни да се извршавају у реалном времену, интегришући велику количину података из бројних сензора (пример са три камере унутар Теслиног аутопилот система је дат на слици).

Слика 1.1: Пример функционисања Теслиног аутопилота

Глава 2

Теоријски темељи

2.1 Увод у машинско учење

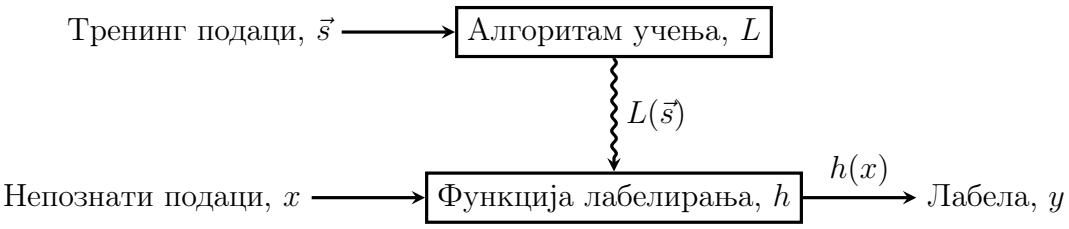
Машинско учење је посебна област вештачке интелигенције која се бави проучавањем и развојем алгоритама способних да "уче", односно уочавају законитости, и доносе претпоставке на основу података.

Машинско учење можемо поделити у три категорије:

1. **Супервизирано учење**
   Наш алгоритам учи на основу података и "лабела" које говоре шта ти подаци представљају. После тренинга, алгоритам постаје способан да додели "лабеле" до тада невиђеним улазним подацима. Овај тип учења се користи најчешће за проблеме класификације (пример: препознавање објеката на слици).

2. **Несупервизирано учење**
   Наш алгоритам учи само на основу података и има за циљ да пронађе интересантна својства тих података. Овај тип учења се користи најчешће за проблеме груписања (пример: recommender системи).

3. **Учење са појачавањем**
   Наш алгоритам комуницира са неким системом тако што извршава одређене акције, при чему се стање тог система мења, а као резултат добијамо награду (поене). Алгоритам треба да научи које акције треба да извршава како би максимизовао будуће награде (пример: играње игрица).



Слика 2.1: Приказ модела супервизираног учења.

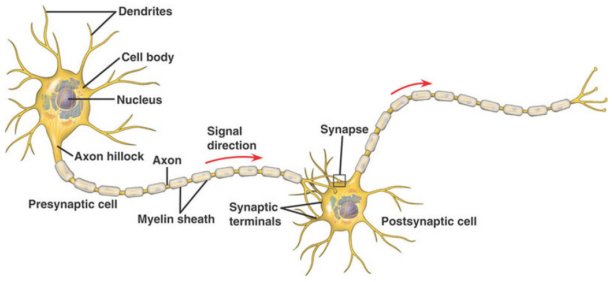
2.1.1 Супервизирано учење

Неуралне мреже, којима ћемо се бавити већином овог рада, су директно способне искључиво за супервизирано учење — ради њихове примене у другим проблемима, потребно је формулисати те проблеме као проблеме супервизираног учења. Стога ћемо се детаљније осврнути на формалну дефиницију овог типа машинског учења.

Дефиниција 1. Нека је $\vec{s} = \{(x_1, y_1), ..., (x_m, y_m)\}$ вектор познатих парова улаза и излаза који описују неку функцију $f : X \to Y$. Алгоритам учења $L : (X \times Y)^m \to (X \to Y)$ је способан да на основу овог вектора (тренинг вектора), дефинише функцију $L(\vec{s}) = h$, која преставља апроксимацију функције f, и стога се може користити за лабелирање нових, неозначених, података.

2.2 Неуронална мрежа

2.2.1 Мотивација



Слика 2.2: Нервна ћелија

Неуралне мреже су настале са циљем да, колико је могуће, реплицирају једини познат природни интелигентни систем—људски мозак. Стога, оне су настале као последица проучавања нервних система, као и нервних ћелија, живих бића.

Нервна ћелија живог бића је посебан тип ћелија специјализованих за пренос надражаја, као и пренос и чување информација. Она се састоји из:

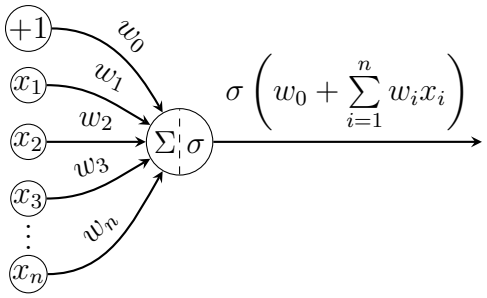
1. **Тела ћелије** — у њему се налази једро као и основне органеле неопходне за метаболичке процесе сваке ћелије.

2. **Дендрита** — специјалних структура у виду кратких и разгранатих наставака који проводе надражаје које примају из других ћелија до тела ћелије.

3. **Аксона** — дугачког цилиндричног наставка који преноси надражај од тела ћелије ка другим ћелијама. Полази из посебног подручја тела нервне ћелије који се назива аксонски брежуљак, где се улазни надражаји обједињују пре преноса на другу ћелију.

Свака ћелија има тачно један аксон, а може имати више дендрита. Синапса представља спој нервних ћелија, односно аксона једне и дендрита друге нервне ћелије. Према начину преноса могу се издвојити два типа: електричне синапсе, као и хемијске синапсе.

2.2.2 Структура перцептрона



Слика 2.3: Модел перцептрона.

Перцептрон представља јединицу грађе нашег математичког модела неуралне мреже. Функционише по принципу сличном функционисању стварне нервне ћелије. Имамо n улаза на које добијамо улазне податке, и i-том улазу (x_i) је придружена тежина w_i. Перцептрон рачуна скаларни производ улазних података и тежина, након чега се на резултат примењује активациона функција, чиме се добија коначни излаз.

Овако дефинисан модел обавезно представља функцију која пролази кроз координатни почетак; стога је пожељно додати и *bias* вредност, w_0, који дозвољава да контролишемо излаз мреже за нула-улаз.

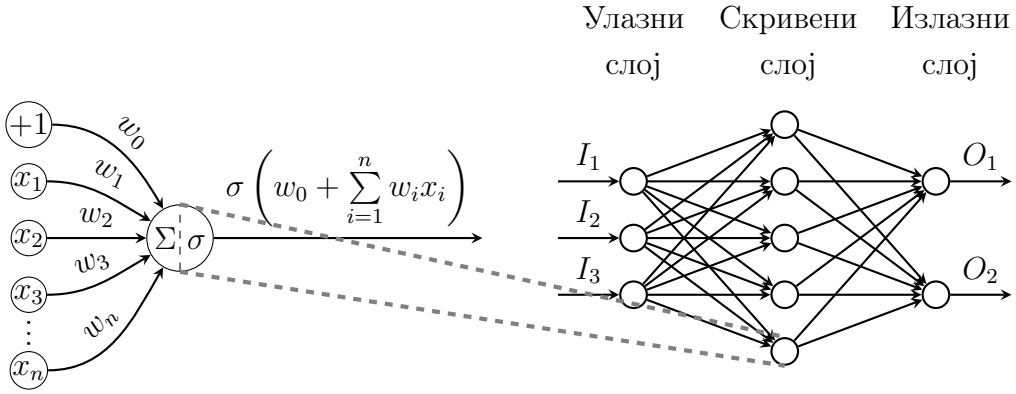
Дефиниција 2. Нека је $\vec{w} = (w_0, w_1, w_2, ..., w_n)$ вектор реалних бројева које представљају тежине, и σ функција активације. За улаз $\vec{x} = (x_1, x_2, ..., x_n)$ перцептрон ће произвести излаз $\sigma\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$. Излаз једног перцептрона обележаваћемо са $\Pi(\vec{x})$.

Перцептрони представљају изузетно једноставан модел—врло је лако конструисати проблеме на којима један перцептрон није довољан. Ради мотивације, прилажемо два примера:

Пример 1. Научити функцију $5x_1 - 6x_2 - 3 > 0$.
Наш перцептрон треба да на основу x_1 и x_2 предвиди излаз дате функције. За вектор тежина узећемо $\vec{w} = \{3, 5, -6\}$, а као функцију активације идентитет $f(x) = x$. Резултат нашег перцептрона ћемо тумачити на следећи начин: уколико је резултат већи од нула, то значи да је неједнакост $5x_1 - 6x_2 - 3 > 0$ тачна, а уколико је мања од нуле, неједнакост је нетачна. Лако је показати да резултат овако дефинисаног перцептрона у потпуности одговара жељеној функцији.

Пример 2. Научити функцију $x_1^3 + x_2^3 > 5$, користећи идентитет $f(x) = x$ као активациону функцију.
За овај пример се испоставља да ни једна комбинација тежина w_0, w_1 и w_2 неће довољно добро апроксимирати задату функцију, због инхерентне линеарности овог модела и нелинеарности проблема.

2.2.3 Структура неуралне мреже



Слика 2.4: Модел неуралне мреже.

Како би могли да боље да апроксимирамо сложеније функције јавила се идеја да повежемо више перцептона. Неуралне мрежа је мрежа састављена од више међусобно повезаних перцептрона. Према начину повезивања перцептрона у мрежу можемо издвојити два типа:

- Ацикличне (feedforward) мреже
  Мреже у којима не постоје циклуси, већ излаз једног перцептрона иде у следећи и никад више се не враћа у тај исти.

- Рекурентне мреже
  Мреже у којима постоји усмерени циклус. Често коришћене за проблеме са временско-зависним подацима, као што је нпр. препознавање гласа.

2.2.4 Feedforward мрежа

Састоји се од више слојева где се први слој назива улазни, последњи излазни, а слојеви између скривени слојеви. Ти слојеви су повезани међусобно тако да: сваки слој осим излазног шаље податке у сваки перцептрон наредног слоја, а излаз излазног слоја представља излаз неуралне мреже.

Дефиниција 3. Улазни слој перцептрона S је скуп перцептрона величине улазног вектора. Вредност i-тог одговара i-том улазу. Он не рачуна ништа већ служи да податке пропагира у наредни слој.

Дефиниција 4. Скривени слој перцептрона S је скуп перцептрона величине n. За улаз $\vec{x} = (x_1, x_2, ..., x_m)$, где је m број перцептрона претходног слоја, слој перцептрона ће на излаз дати вектор $\vec{o} = (\Pi_1(\vec{x}), \Pi_2(\vec{x}), ..., \Pi_n(\vec{x}))$. Дакле, сваки скривени перцептрон, као улаз узима излаз сваког перцептрона претходног слоја.

Дефиниција 5. Излазни слој перцептрона S је скуп перцептрона величине n. За улаз $\vec{x} = (x_1, x_2, ..., x_m)$, где је m број перцептрона претходног слоја, слој перцептрона ће на излаз дати вектор $\vec{o} = \{\Pi_1(\vec{x}), \Pi_2(\vec{x}), ..., \Pi_n(\vec{x})\}$. Излаз излазног слоја представља и излаз целокупне неуралне мреже.

Најчешће, Feedforward неуралну мрежа представља скуп слојева: улазног слоја, једног или више скривених слојева и излазног слоја. Слојеви су повезани тако да се на улазни слој настављају скривени слојеви, а након скривених слојева излазни слој. Излаз претходног представља улаз тренутног слоја. Сваки слој може имати другачију активациону функцију.

Показали смо да један перцептрон може потпуно апроксимирати неке функције, док друге не може довољно добро. Поставља се питање какве функције може овако дефинисан модел научити, односно апроксимирати, а одговор на њега даје следећа теорема.

Теорема 1. Cybenkova теорема из 1989. године каже да feedforward неуралну мрежа са само једним скривеним слојем може довољно добро апроксимирати било коју реалну

функцију (непрекидну и ограничену), уколико се за активациону функцију скривеног слоја користи сигмоидна функција, а излазног идентитет функција.

Дакле, за сваку функцију постоји скуп тежина који је описује. Ипак, доказ ове теореме није конструктиван, па нам ова теорема не открива и начин за тренирање такве мреже.

2.2.5 Дубоке неуралне мреже

Дубоке неуралне мреже [1] су посебан тип неуралних мрежа које имају више скривених слојева. Иако због претходне теореме не постоји разлог зашто би користили мреже са више слојева, испоставља се да је дубоке неуралне мреже много лакше истренирати и да имају већу моћ учења. Наиме, за разлику од обичних мрежа, дубоке неуралне мреже су способне да из **сирових података** издвајају нека занимљива својства на основу којих касније доносе предикцију, односно приликом тренирања ових мрежа излази првих слојева представљаће нека једноставна својства (нпр. код обраде слика то у већини случајева буду ивице), након тога процесирањем кроз дубље слојеве та својства ће се све више и више усложњавати, тако да ће крајни слој донети предикцију на основу тих издвојених својства.

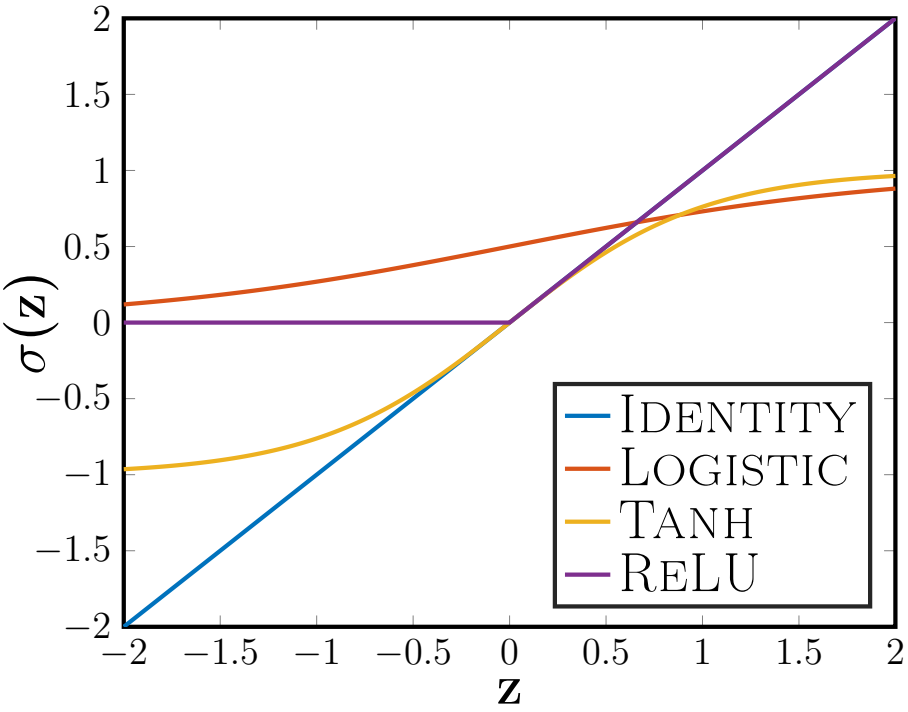
2.2.6 Функција активације

Неке од често коришћених активационих фукнција:

- функција идентитета $\sigma(x) = x$
- хиперболички тангенс $\sigma(x) = \tanh(x)$
- логистичка функција $\sigma(x) = \frac{1}{1+e^{-x}}$
- ReLU (Линеарни исправљач) функција $\sigma(x) = \begin{cases} 0 & \text{за } x < 0 \\ x & \text{за } x \geq 0 \end{cases}$

За функцију активације је важно да буде диференцијабилна. Такође, често се бирају неке нелинеарне функције како би се у систем увела нелинеарност и побољшале перформансе. Једна од најкоришћенијих функција данас је ReLU због доброг капацитета учења.

За проблеме пробабилистичке класификације користи се и Софтмакс (енг. Softmax)



Слика 2.5: Активационе функције

функција, монотона је и има особину да све излазе "сабија" у опсег $[0, 1]$. Такође, збир свих излаза једног слоја је 1. Дефинисана је на следећи начин: $\sigma(x)_j = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}$.

2.2.7 Feedforward алгоритам

У наставку је приказан алгоритам за рачунање излаза Feedforward мреже. Рачунање излаза се може представити и матричним операцијама које се могу паралелизовати на графичким процесорима.

CALCULATE(NODE, INPUT)

1 input $\leftarrow \{1\}$ + *input*
2 product $\leftarrow$ node.w $\cdot$ input
3 **return** node.σ(product)

```
FeedForward(network, x⃗)
1  input ← x⃗
2  for all layer ∈ network.hiddenLayers        // За сваки скривени слој
3       output ← {}
4       for all x ∈ layer                      // За сваки неурон у скривеном слоју
5            output ← output + Calculate(x, input)
6       input ← output
7  output ← {}
8  for all x ∈ network.outputLayer // За сваки неурон у излазном слоју
9       output ← output + Calculate(x, input)
10 return (output)
```

Знак + представља конкатенацију.

2.2.8 Функција грешке

Наш циљ је да научимо мрежу што боље можемо функцију f на основу тренинг примера, односно да одредимо тежине које ће описивати функцију f. Како бисмо одредили меру тачности предикција дефинисаћемо функцију грешке. Постоји много начина да дефинишемо функцију грешке и њено дефинисање директно утиче на перформансе нашег алгоритма. Један од начина је:

Дефиниција 6. Функција квадратног одступања E над тренинг примерима $\vec{s} = \{(x_1, y_1), ..., (x_m, y_m)\}$ је једнака $\sum_{i=1}^{m} \frac{1}{2}(h(x_i) - y_i)^2$.

2.2.9 Метод опадајућег градијента

Наш циљ је да минимизујемо функцију грешке, коригујући функцију h односно код неуралних мрежа коригујући тежине перцептрона. За тај проблем, користићемо метод опадајућег градијента.

$$\vec{x} \leftarrow \vec{x} - \lambda \nabla F(\vec{x})$$

где је $\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_1}, ..., \frac{\partial}{\partial x_n}\right)$ и λ коефицијент учења.

Овај метод се заснива на чињеници да функција у околини неке тачке најбрже опада у негативном смеру извода те функције у тој тачки. Улога коефицијента учења је

да одредимо за колико ћемо се померити у свакој итерацији. Њега морамо пажљиво бирати јер уколико је премали требаће нам много итерација да досегнемо минимум, а уколико је превелики осцилираћемо око тог минимума без достизања истог. Постоје оптимизациони алгоритми који се баве одабиром вредности коефицијената (видети више у оптимизацијама).

Један од недостатака овог метода је што ћемо за неке одређене почетне вредности остати заглављени у субоптималном локалном минимуму.

Дефиниција 7. За учење једног перцептрона
$E(\vec{s}) = \sum_{k=1}^{m} \frac{1}{2}(h(x_k) - y_k)^2 = \sum_{k=1}^{m} \frac{1}{2}(\sigma\left(w_0 + \sum_{i=1}^{n} w_i x_{k,i}\right) - y_k)^2$
$w_0 = w_0 - \sum_{k=1}^{m} (\sigma\left(w_0 + \sum_{i=1}^{n} w_i x_{k,i}\right) - y_k)\sigma'\left(w_0 + \sum_{i=1}^{n} w_i x_{k,i}\right)$
$w_a = w_a - \sum_{k=1}^{m} (\sigma\left(w_0 + \sum_{i=1}^{n} w_i x_{k,i}\right) - y_k)\sigma'\left(w_0 + \sum_{i=1}^{n} w_i x_{k,i}\right) x_{k,a}$ где је $a \neq 0$

Важно је да се ажурирања вредности при једној итерацији изврше симултано. У наредној секцији извешћемо промену тежина за целу неуралну мрежу.

2.2.10 Пропагација уназад

Пропагација уназад [2] представља алгоритам за одређивање градијената сваког унутрашњег перцептрона у мрежи. У наредном извођењу, ради једноставности и прегледности функцију грешке E сматрамо за грешку на једном тренинг примеру, док се за тренирање на више тренинг примера ради само о суми изведених вредности. Такође, поједноставићемо bias вредност тако што ћемо сматрати да су улазни вектори облика $1, x_1, x_2, ..., x_n$ где први члан вектора одговара bias тежини. За активациону фунцкију сматраћемо да користимо логистичку, једначине се на исти начин изводе коришћењем било које друге активационе функције. Потребно је да израчунамо $\frac{\partial E}{\partial w_{ij}}$ где је E фунцкија грешке, а w_{ij} је тежина између i-тог и j-тог неурона, односно тежина којом множимо излаз i-тог неурона за рачунање j-тог. На основу правила извода композиције добијамо:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j}\frac{\partial out_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

где је out_i излаз i-тог чвора, net_i скаларни производ унутар i-тог чвора односно излаз без примењене активационе функције.
Како је $net_j = \sum_k w_{kj} o_k$ онда је $\frac{\partial net_j}{\partial w_{ij}} = o_i$ где је o_i излаз i-тог неурона.

Како је $out_j = \sigma(net_j)$ онда је $\frac{\partial out_j}{\partial net_j} = \sigma(net_j)(1 - \sigma(net_j)) = o_j(1 - o_j)$ на основу извода логистичке функције.
За $\frac{\partial E}{\partial out_j}$ разликоваћемо два случаја:

1. Чвор се налази у излазном слоју, тада је $out_j = y_j$ па је $\frac{\partial E}{\partial out_j} = y_j - t_j$ где је t_j излаз j-тог чвора у тренинг примеру.

2. Чвор се не налази у излазном слоју, тада нам парцијални извод зависи и од сваког чвора који директно добија на улаз излаз j-тог чвора. Коришћењем правила извода композиције добијамо да је $\frac{\partial E}{\partial out_j} = \sum_l \frac{\partial E}{\partial net_l} \frac{\partial net_l}{\partial out_j} = \sum_l \frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial net_l} w_{jl}$ где је l индекс чвора који директно зависи од излаза j-тог чвора.

Свеукупно, написаћемо уз помоћ параметра δ који ћемо дефинисати на следећи начин:
$\delta_j = \frac{\partial E}{\partial out_j} \frac{\partial out_j}{\partial net_j} = \begin{cases} (o_j - t_j)o_j(1 - o_j), \text{ ако је чвор } j \text{ у излазном слоју} \\ (\sum_l \delta_l w_{jl})o_j(1 - o_j), \text{ ако је чвор } j \text{ у скривеном слоју} \end{cases}$
Онда за тежину w_{ij} добијамо њен градијент: $\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$.

2.2.11 Алгоритам тренирања

Training(network, trainingSet)

```
1  while not converged
2      dw ← 0
3      for all (input, output) ∈ trainingSet // За сваки тренинг пример
4          FeedForward(network,input)
5          for all x_j∈ network.outputLayer // За неуроне у излазном слоју
6              δ_j ← (o_j - output_j)o_j(1 - o_j)
7          for all layer ∈ reversed(network.hiddenLayers) // За сваки скривени слој
8              for all x_j∈ layer        // За неуроне у скривеном слоју
9                  δ_j ← (∑ δ_l w_jl)o_j(1 - o_j)
10         dw_ij ← dw_ij - λ δ_j o_i
11     w ← w + dw
12 return (network)
```

2.3 Рекурентне мреже

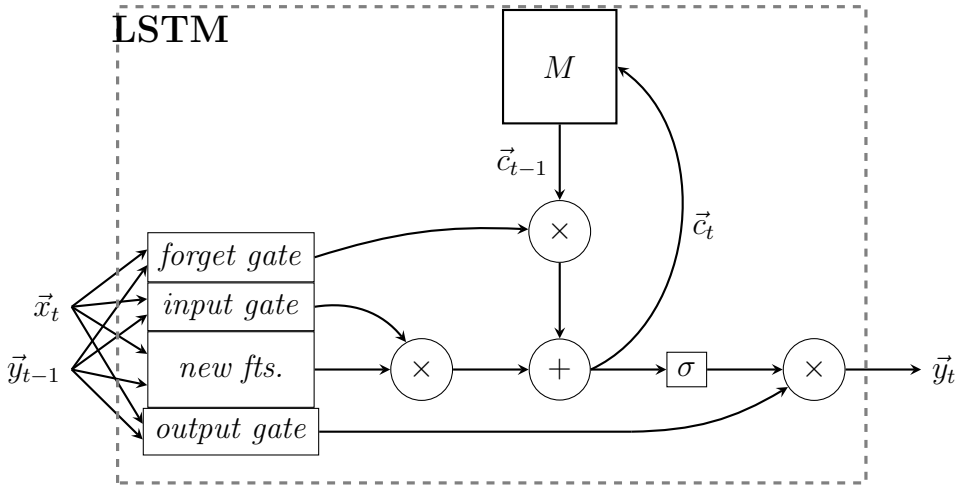
Рекурентне мреже представљају посебан тип неуралних мрежа које садрже усмерени циклус. Ове мреже су корисне за рад са временски зависним подацима односно подацима кроз време. Предност овог типа мреже је што нам омогућава да имамо зависности неодређене дужине.

2.3.1 LSTM

LSTM (Long Short-Term Memory) [4] представља основну јединицу рекурентних мрежа дизајнирану тако да нема проблем са нестајућим градијентом. Унутар јединице, налази се ћелија која може да чува стање из претходних пролаза. Постоје три капије: капија улаза, капија излаза и капија заборава. Оне на основу улаза одређују у којој мери ће подаци ући и изаћи из јединице, као и у коликој мери ће се претходно израчуната својства занемарити.

$$\vec{i}_t = logistic\left(\mathbf{W}_i\vec{x}_t + \mathbf{U}_i\vec{y}_{t-1} + \vec{b}_i\right)$$
$$\vec{f}_t = logistic\left(\mathbf{W}_f\vec{x}_t + \mathbf{U}_f\vec{y}_{t-1} + \vec{b}_f\right)$$
$$\vec{o}_t = logistic\left(\mathbf{W}_o\vec{x}_t + \mathbf{U}_o\vec{y}_{t-1} + \vec{b}_o\right)$$
$$\vec{ft}_t = \tanh\left(\mathbf{W}_{ft}\vec{x}_t + \mathbf{U}_{ft}\vec{y}_{t-1} + \vec{b}_{ft}\right)$$
$$\vec{c}_t = \vec{ft}_t \otimes \vec{i}_t + \vec{c}_{t-1} \otimes \vec{f}_t$$
$$\vec{y}_t = \tanh\left(\vec{c}_t\right) \otimes \vec{o}_t$$

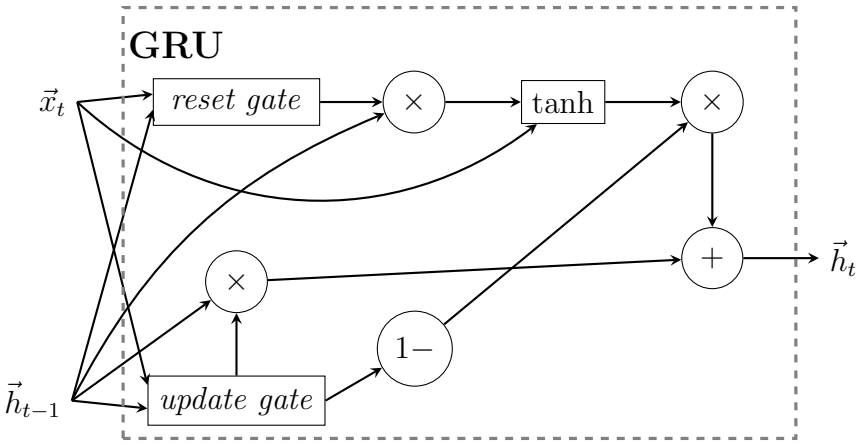
Симбол $\otimes$ представља оператор који за вектор $(a_1, a_2, ..., a_n)$ и вектор $(b_1, b_2, ..., b_n)$ даје резулатат $(a_1b_1, a_2b_2, ..., a_nb_n)$.



Слика 2.6: LTSM

2.3.2 GRU

GTU (Gated Recurrent Unit) представља поједностављен модел LTSM који постиже сличне резултате као и LTSM, али са мањим бројем параметара што значајно убрзава учење.



Слика 2.7: GRU

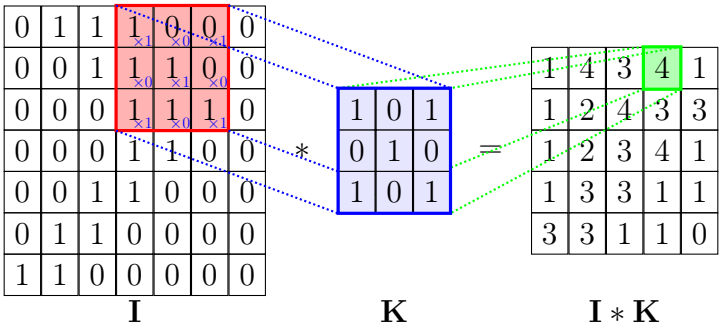
$$\vec{z}_t = logistic\left(\mathbf{W}_z\vec{x}_t + \mathbf{U}_z\vec{h}_{t-1} + \vec{b}_z\right)$$
$$\vec{r}_t = logistic\left(\mathbf{W}_r\vec{x}_t + \mathbf{U}_r\vec{h}_{t-1} + \vec{b}_r\right)$$
$$\vec{h}_t = \vec{z}_t \otimes \vec{h}_{t-1} + (1 - \vec{z}_t) \otimes \tanh\left(\mathbf{W}_h\vec{x}_t + \mathbf{U}_h(r_t \otimes h_{t-1}) + \vec{b}_h\right)$$

2.4 Конволуцијске неуралне мреже

2.4.1 Мотивација

Замислите да радимо класификацију на сликама у боји величине 1024 x 1024, за ту величину слике потребно нам је 1024 x 1024 x 3 (ширина x дужина x број канала за описивање боје) улазна чвора, што значи да би сваки чвор из скривеног слоја морао да чува толико тежина. Наша неуралне мрежа би била превелика, учење би трајало веома дуго, а и заузимало би велику количину ресурса. Прва идеја за решавање овог проблема је да смањимо слику пре него што је пустимо кроз неуралну мрежу. Међутим, на овај начин не би искористити сав потенцијал слике односно изгубили бисмо велику количину информација које би могле да нам буду од користи. Решење за овај проблем представљају конволуцијске неуралне мреже. Конволуцијске мреже функционишу по претпоставци да су подаци локално повезани, односно што је једна информација ближа другој то је повезанија. Код слика важи та локална повезаност односно пиксели у близини одређују неку ивицу, док пиксели на два различита краја слике немају толику повезаност.

2.4.2 Конволуција

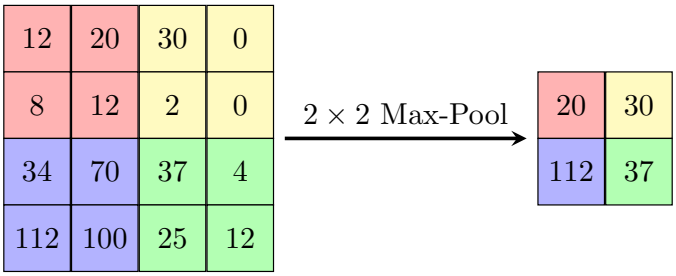


Слика 2.8: Конволуција.

Конволуција је математички оператор који на основу две матрице, где је прва наша слика, а друга тзв. кернел, формира трећу матрицу.

Дефиниција 8. Нека је I дводимензиона матрица која представља улазну матрицу и кернел K дводимензиона матрица величине $h \times w$. Конволуција $*$ представља оператор тако да је $I * K$ једнако матрици $(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \times I_{x+i-1,y+j-1}$
Овако дефинисана конволуција представља исти тип операција као и сам модел неурона (сабирање променљивих и множење скаларима), па ће исти алгоритми бити примењиви и за учење кернела.

2.4.3 MaxPool



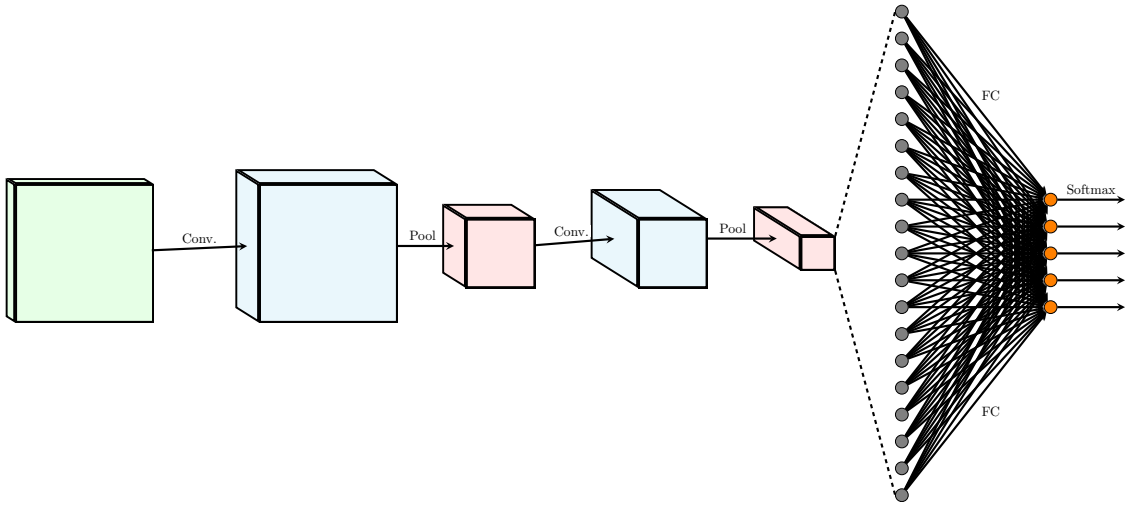
Слика 2.9: MaxPool операција

MaxPool је операција која смањује нашу слику.

Дефиниција 9. Нека је I дводимензиона матрица величине $x \times y$ која представља улазну матрицу. Применом MaxPool оператора величине $a \times b$ добијамо нову матрицу такву да је $A_{x,y} = \max_{ax \leq p < a(x+1), by \leq q < b(y+1)} I_{p,q}$

Разлог за максимизацију, уместо нпр. узимања средње вредности активација, је зато што ће конволуција кернелима из претходног слоја произвести излаз чија величина представља јачину препознатог шаблона—стога је најбоље сачувати најјаче препознат шаблон у том региону.

2.4.4 Архитектура конволуцијске мреже



Слика 2.10: Архитектура конволуцијске мреже

Када смо дефинисали конволуцију и MaxPool операцију, можемо описати архитектуру ових мрежа. На улаз ће бити примењена серија ових конволуција и MaxPool операција које ће издвојити нека занимљива својства са слике, а онда смањити ту слику при чему ће се та издвојена својства сачувати. Након тога, та смањена слика улази у потпуно повезану неуралну мрежу која ће процесирати податке исто као и модели из претходне секције. Код класификације слика, испоставља се да скоро увек прве конволуције буду неки детектори ивица (енг. edge detector) што и није чудно с обзиром да ивице садрже највише информација о објекту.

2.5 Учење са појачавањем

2.5.1 Увод

Учење са појачавањем (енг. reinforcement learning) представља посебан тип машинског учења где наш алгоритам има могућност да извршава одређене акције над системом након којих добија награду. Циљ алгоритма је извршавати праве акције како би се

максимизовала награда. Оно за разлику од супервизираног учења не захтева никакве лабеле или било какве информације о систему, већ се својства система морају откривати методом пробе и грешке. Захваљујући овој особини један исти алгоритам је способан да научи, и да се исправно понаша у тотално различитим системима.

2.5.2 Формална дефиниција

Формално ћемо дефинисати основне појмове овог типа учења.

Дефиниција 10. Нека је S скуп свих могућих стања система, и нека је A скуп свих могућих акција. Дефинисаћемо функцију промене стања $\mathcal{S}\colon S \times A \to S$ и функцију награде $\mathcal{R}\colon S \times A \to \mathbb{R}$

Дефиниција 11. Функцију $p\colon S \to A$ ћемо звати функцијом полисе, тј. функција која нам враћа акцију коју агент треба да изврши када је систем у датом стању.

Задатак нашег алгоритма је да нађе функцију полисе која ће му доносити највећу награду. Међутим, желимо да та награда дође што пре, па ћемо дефинисати и следећу функцију. Такође, за случај да сума награда дивергира, ова функција ће нам омогућити да разликујемо и такве полисе.

Дефиниција 12. Нека је $\gamma \in [0, 1)$. Функција $V^p(s) = \sum_{k=1}^{+\infty} \gamma^{k-1} r_k$ где је r_k награда добијена после k корака од стања s примењивајући полису p.

Наш циљ је наћи оптималну полису p_{opt} где је $p_{opt}(s) = \operatorname{argmax}_p V^p(s)$ за свако s.

Дефиниција 13. Нека је $Q(s, a) = \mathcal{R}(s, a) + \gamma V_{opt}(S(s, a))$. Ова функција нам даје кумулативну награду уколико у стању s применимо акцију a, а након тога се понашамо оптимално.

Уколико знамо функцију Q, знамо и полису $p_{opt}(s) = \operatorname{argmax}_a\{Q(s, a)\}$.
Такође, знамо да важи $V_{opt}(s) = \max_\alpha\{Q(s, \alpha)\}$
Онда је $Q(s, a) = \mathcal{R}(s, a) + \gamma \max_\alpha\{Q(\mathcal{S}(s, a), \alpha)\}$.

Теорема 2. Ако је Q' процена Q, онда ће $\mathcal{R}(s, a) + \gamma \max_\alpha\{Q'(\mathcal{S}(s, a), \alpha)\}$ бити боља или иста процена $Q(s, a)$ од $Q'(s, a)$.

Претходна теорема нам каже да уколико ми имамо неку процену вредности функције Q, заменом вредности процене са датим изразом добићемо сигурно бољу или исту процену

Q што значи да уколико довољан број пута то исто будемо урадили конвергираћемо ка стварној вредност функције Q. На основу овог својства базираћемо наш алгоритам учења.

2.5.3 Алгоритам учења

Сада можемо описати алгоритам учења:

1. Налазимо се у стању s, изабраћемо акцију a.
2. Извршити је и добити награду $\mathcal{R}(s, a)$.
3. Тренутно смо у стању $\mathcal{S}(s, a)$.
4. Ажурираћемо $Q'(s, a) \leftarrow \mathcal{R}(s, a) + \gamma \max_\alpha \{Q'(\mathcal{S}(s, a), \alpha)\}$
5. Идемо на корак 1.

Међутим, ово би значило да би за свако могуће стање и сваку могућу акцију чували вредност функције Q како би наш алгоритам могао да се понаша оптимално. Нажалост, ово често није могуће зато што је простор могућих стања превелик да би се чувала вредност сваког могућег. Уместо тога, ми ћемо уз помоћ неуралне мреже научити да апроксимирамо функцију Q.

2.5.4 Одабир акције приликом учења

Акције приликом учења можемо бирати на два начина:

- **Истраживање** (енг. explore) представља бирање следеће акције на случајан начин, предност овог приступа је што нам даје могућност да истражимо нешто "ново", односно, даје нам могућност да видимо да ли ће нам нека акција дати награду или неће.
- **Искоришћавање** (енг. exploit) представља бирање следеће акције на основу већ стеченог знања, односно изабраћемо акцију која неуралној мрежи делује да је најбоља за ту ситуацију.

Како бисмо успешно тренирали не можемо се определити за само један начин него морамо користити и један и други, истраживање нам омогућује да сазнамо нешто ново о систему и акцијама, док нам искоришћавање служи да би нас водило ка оптималној стратегији односно стању које ће истраживање побољшавати. Комбиновање ова два начина радимо тако што доделимо вероватноћу да радимо један и други начин и на

основу тога на случајан начин бирамо. Ту вероватноћу можемо мењати како траје учење односно на почетку када немамо никакво знање о систему разумно је да све акције бирамо на истраживачки начин, ипак како временом стичемо неко знање логичније је да почнемо то знање да користимо и да се мање ослањамо на истраживање.

2.5.5 Учење мреже и меморија искуства

Преостаје нам да научимо неуралну мрежу да апроксимира функцију Q. Потребни су нам тренинг примери и лабеле (процењене вредности функције Q) како бисмо могли да учимо неуралну мрежу. Процењене вредности функције Q имамо на основу претходног алгоритма. Сада нам још треба начин да бирамо тренинг примере. Испоставља се да неуронска мрежа најбоље учи на тренинг примерима међу којима не постоји велика корелација. Према томе, ако будемо учили на основу последњих N стања, наша мрежа неће постизати добре резултате. Због тога ћемо увести меморију искуства, то је привремена меморија која ће чувати одређени број претходних стања. Из тог скупа ћемо сваки пут насумичано бирати узорак и тренирати на њему. На тај начин ћемо обезбедити већу независност тренинг примера, па самим тим и квалитентније учење.

2.5.6 Алгоритам DQN

У наставку описаћемо алгоритам [3].

```
ExperienceReplay.init()
1   list ← {}

ExperienceReplay.remember(state,action,reward,newState)
1   list ← list + (state,action,reward,newState)
2   if size(list) > maxSize
3         list.pop()
```

ExperienceReplay.getBatch()

```
1  subset ← list.getRandomSubset()
2  inputs ← {}
3  targets ← {}
4  for x ∈ subset
5        target ← network.predict(x.state)
6        Qnew ← network.predict(x.next_state)
7        if x.gameover
8              target ← reward
9        else
10             target ← reward + discountRate · Qnew
11       inputs ← inputs + x.state
12       targets ← targets + target
13 return (inputs, targets)
```

Q-Learn()

```
1  ExperienceReplay.init()
2  while not converged
3        while not gameover
4              if random() < exploreRate
5                    action ← choose_random_action()
6              else action ← argmax(network.predict(state))
7              newState, reward, gameover ← enviroment.perform(action)
8              ExperienceReplay.remember(state, action, reward, newState)
9              network.trainOnBatch(ExperienceReplay.getBatch())
10             state ← newState
11             exploreRate ← exploreRate − step
12 return (network)
```

2.5.7 A3C алгоритам

A3C [5] (енг. Asynchronous Advantage Actor-Critic) алгоритам представља унапређену верзију алгоритма за учење са појачањем који постиже значајно боље перформансе за разлику од претходног. Порекло имена објаснићемо након што објаснимо алгоритам.

Дефиниција 14. Функција вредности полисе је једанка $V(s) = \mathbb{E}_{\pi(s)}(r + \gamma V(s'))$ где $\mathbb{E}$ представља математичо очекивање. Она нам у суштини каже просечну кумулативну награду за стање s.

Дефиниција 15. Нека је $Q(s,a) = \mathcal{R}(s,a) + \gamma V(S(s'))$. Ова функција нам даје кумулативну очекивану награду уколико у стању s применимо акцију a и након тога наставимо да играмо.

Дефиниција 16. Сада ћемо дефинисати функцију предности A као $A(s,a) = Q(s,a) - V(s)$.

Функција A нам показује колику предност добијамо ако применимо акцију a у стању s. Ако је позитивно то ће значи да је акција пожељна односно да је боља од просека, ако је негативна, лошија је и пожељно је избећи овакву акцију.

У претходном алгоритму циљ наше мреже је био да апроксимира што боље функцију Q на основу које смо бирали акције које ћемо извршити. Сада, желимо да нам наша мрежа, уместо функције Q, врати дистрибуцију вероватноћа за сваку акцију. Акције можемо бирати тако што ћемо узети акцију са највећом вероватноћом, или тако што ћемо према тој дистрибуцији изабрати случајну акцију.

Дефиниција 17. Функција полисе $\pi(s)$ је функција која нам враћа дистрибуцију вероватноћа за стање s.

Сада, желимо неку величину која нам каже колико је добра наша полиса. Дефинисаћемо следећу функцију:

Дефиниција 18. Нека је функција $J(\pi) = \mathbb{E}_{s_0}(V(s_0))$ где s_0 представља свако могуће почетно стање. Она нам за полисз π каже очекивану кумулативну награду за свако почетно стање, што је функције већа то је наша полиса боља.
Циљ нам је максимизовати ову функцију, па нам је потребан извод.

$$\nabla J(\pi) = \mathbb{E}_{sp^{\pi}, a\pi(s)}(A(s,a)\nabla log\pi(a|s))$$

Формулу нећемо детаљно изводити, $\nabla \log \pi(a|s)$ нам говори у ком смеру вероватноћа расте, док је $A(s,a)$ скаларна величина која нам говори колико је та акција добра. Пошто очекивање не можемо тачно израчунати, користићемо његову апроксимацију односно средњу вредност на случајном узорку. Остаје нам још да израчунамо $A(s,a)$, а за то можемо користити да је $A(s,a) = R(s,a) + V(s') - V(s)$, функцију $V(s)$ можемо такође апроксимирати помоћу неуралне мреже, а пошто треба да апроксимирамо и $\pi(s)$ и $V(s)$ можемо користити исту неуралну мрежу, што ће довести до бржег и стабилнијег тренирања. За Q-learning алгоритам користили смо меморију искуства како би смо омогућили независност тренинг примера, међутим сада ћемо користити више агената који у исто време интерагују са својим окружењем и на тај начин обезбедити независност.

Сада када знамо како A3C функционише објасићемо порекло имена:

- **Asynchronous** (Асихрони) - означава могућност тренирања на више окружења одједном чиме се постиже независност тренинг примера.
- **Advantage** (Предност) - означава функцију предности на основу које се базира учење овог алгоритма
- **Actor-Critic** (Извођач-Критичар) означава два излаза неуралне мреже односно функцију дистрибуције $\pi(s)$ и награде $V(s)$, прва функција представља извођача који извршава акције, док друга функција представља критичара који оцењује колико су те акције добре.

2.5.8 Детаљи имплементације A3C алгоритма

Функцију грешке ћемо дефинисати: $L = L_\pi + c_v L_v + c_{reg} L_{reg}$

L_π је грешка полисе.
L_v је грешка функције вредности.
L_{reg} предсавља функцију ентропије над расподелом.
c_{reg} и c_v представљају параметре који одређују важност сваког члана.

Грешка полисе

$J(\pi) = \mathbb{E}[A(s,a) \log \pi(a|s)]$
$L_\pi = -\frac{1}{n} \sum_{i=1}^{n} A(s_i, a_i) \log \pi(a_i|s_i)$

Грешка функције вредности

За функцију $V(s_0) = r_0 + \gamma V(s_1)$ можемо увести оптимизацију тако да уместо гледања само једног потеза унапред, гледамо више корака унапред $V(s_0) = r_0 + \gamma r_1 + ... + \gamma^n V(s_n)$
Грешку $e = r_0 + \gamma r_1 + ... + \gamma^n V(s_n) - V(s_0)$
$L_V = \frac{1}{n} \sum_{i=1}^{n} e_i^2$

Регулациони члан

Представља функцију ентропије над дистрибуцији $H(\pi(s)) = -\sum_{k=1}^{n} \pi(s)_k log \pi(s)_k$
$L_{reg} = -\frac{1}{n} \sum_{i=1}^{n} H(\pi(s_i))$
Улога регулационог члана је да дозволи мрежи да истраживање.

2.6 Чести проблеми

У овој секцији биће обрађене неке грешке које могу утицати на перформансе нашег алгоритма, и о којима треба водити рачуна приликом пројектовања оваквих система.

2.6.1 Грешке у тренинг подацима

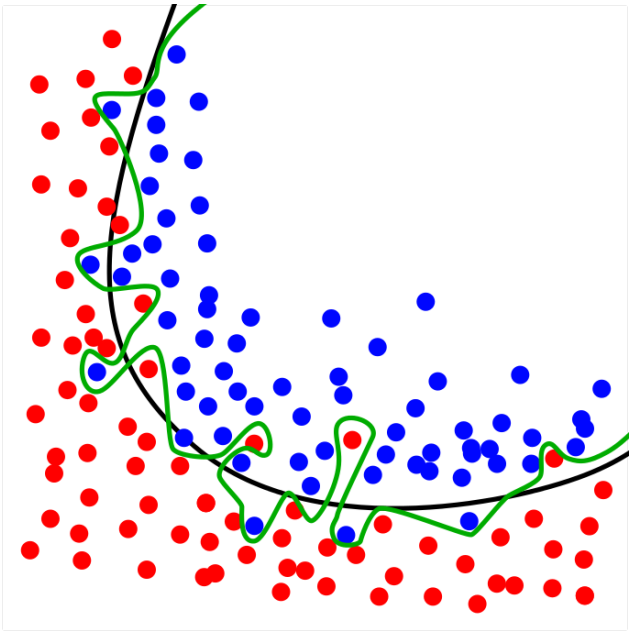
Неуралне мреже често захтевају огроман број тренинг примера, код супервизираног учења често се јавља потреба да људи означе (анотирају) велики скуп података. У тако великом скупу података, можемо очекивати да ће постојати неке грешке настале приликом означавања. Овакви пропусти могу утицати на квалитет учења као и на тачност алгоритма.

2.6.2 Недовољан или нерепрезентативан тренинг скуп

Други тип грешке представљају мали или недовољно репрезентативни тренинг скупови. Односно да би наш алгоритам добро научио потребно је изложити систему што више тренинг података, исто тако, ти подаци би требало да буду варијабилни тј. да покривају што већи опсег могућих улаза за систем. Нпр. уколико радимо са сликама, пожељно је имати слике различитог осветљења, контраста, фокуса; ако радимо неку врсту класификације аутора књиге по исечку, пожељно је имати исечке из различитих књига једног истог аутора, а не само из једне. Неки од ових недостатака могу се решити модифакцијом тренинг скупа која је покривена у наредној секцији (оптимизације), док се други могу отклонити пажљивим одабиром тренинг података, као и начинима за њихово прикупљање. Јасно је да креирање доброг скупа података представља науку за себе, и да је веома важан сегмент креирања успешног система. Пажљиво прикупљање података или руковање подацима спада више у домен data science-а (науке о подацима) него машинског учења, због тога ова област неће бити детаљније обрађена у овом раду.

2.6.3 Overfitting

Overfitting представља један од најчешћих проблема који се јављају у машинском учењу. Циљ тренирања у машинском учењу је да уочи неке "унутрашње везе" које се крију међу подацима, и да буде способан да те везе генерализује на до тада невиђене примере. Међутим, дешава се да наш алгоритам престане да учи те везе на подацима, него крене да учи шум који се налази међу подацима, односно учи саме тренинг примере. Иако се смањује вредност функције грешке на тренинг скупу, губи се способност генерализације, па се перформансе над невиђеним скупом података смањују. Развијени су бројни алгоритми



Слика 2.11: Приказ overfittinga

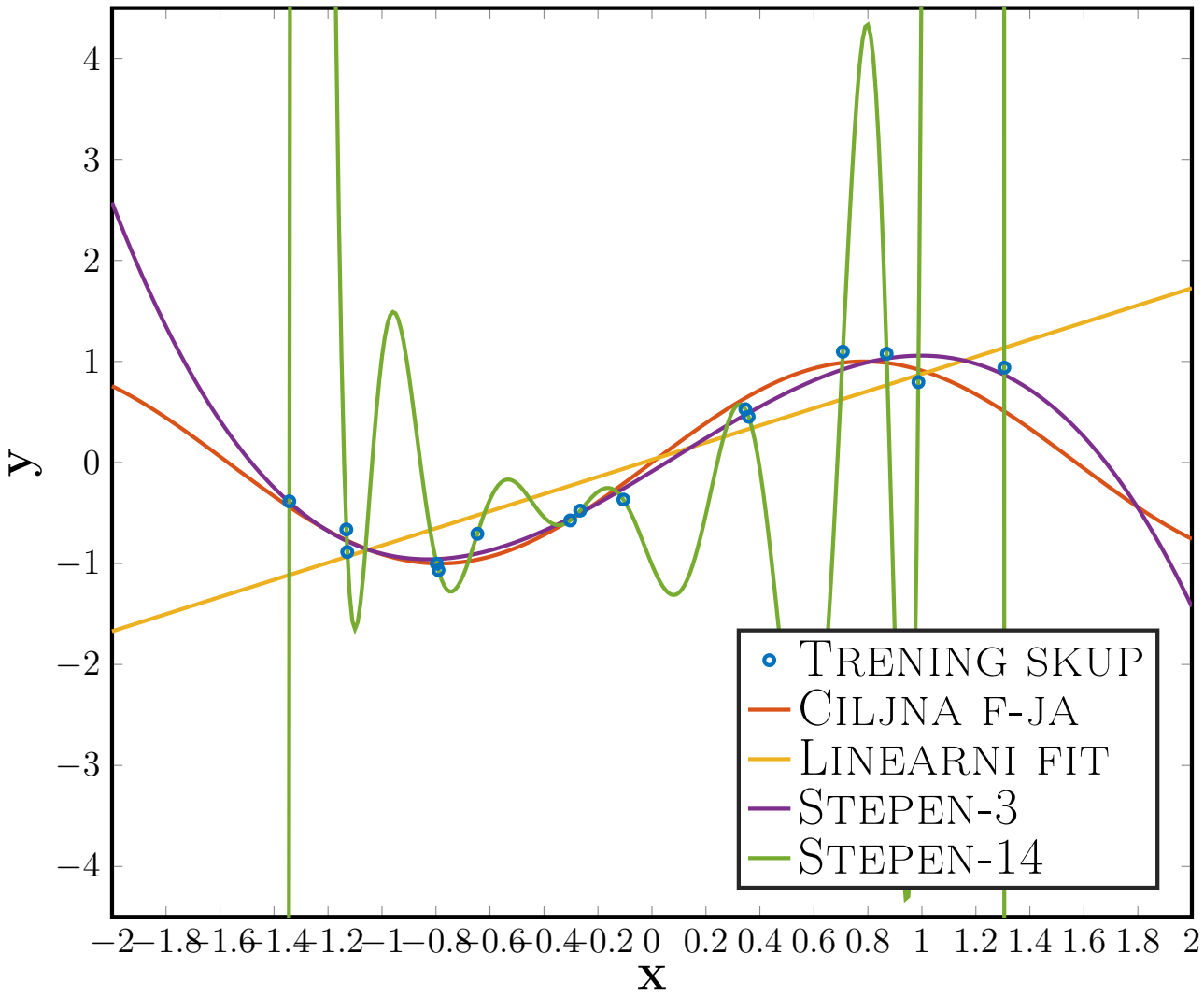
и методе које имају за циљ да спрече overfitting. Једна од таквих оптимизација је и стохастични опадајући градијент који је детаљније описан у наредној секцији.

2.6.4 Underfitting

Underfitting представља појаву супротну overfitting-у. Тада наш алгоритам не успева да научи много из тренинг скупа. Пример 2 из секције о Перцептрону представља underfitting, јер ми покушавамо да научимо нелинеарну фунцију уз помоћ линеарног модела.

2.6.5 Нестајући градијент

Проблем нестајућег градијента представља појаву да се вредност градијентног ажурирања смањује експоненцијално на сваком претходном слоју. Последица тога је да ће код дубљих мрежа вредност тог ажурирања бити веома мала на првим слојевима што отежава и успорава тренирање. Једно од решења за проблем нестајућег градијента представљају и нестандардне мреже описане у наредним секцијама, као што су ResNets и DenseNets.



Слика 2.12: Приказ overfittinga и underfittinga на синусној функцији

2.6.6 Експлодирајући градијент

Експлодирајући градијент је појава супротна нестајућем градијенту, односно вредност градијентног ажурирања постаје огромна што не доприноси много учењу. Један од начина за решење овог проблема је одсецањем градијента.

2.7 Оптимизације

У овој секцији биће приказане неке од најчешћих метода које се користе ради постизања бољих резултата током тренинга; превасходно избегавањем ефеката overfitting-a, који је и најчешћи проблем при тренирању неуралних мрежа.

2.7.1 Стохастични опадајући градијент

Стохастични опадајући градијент или инкрементални опадајући градијент представља оптимизациону технику где се ажурирање тежина врши на сваком примеру одвојено уместо као сума ажурирања над целим тренинг скупом.
Ипак, најкоришћенији приступ је комбинација ова два принципа, односно, тренираћемо на малим насумично одабраним узорцима тренинг примера, и над њима вршити ажурирања. Предност овог начина у односу на класични је што се постиже стабилнија конвергенција ка минимуму. Предност се, такође, огледа у томе што се користи мање ресурса по ажурирању што је погодно за системе са великом количином тренинг примера. Такође, предност ове мале групе је и у томе што се отвара могућност за паралелизацију тренинга.

2.7.2 Модификација тренинг података

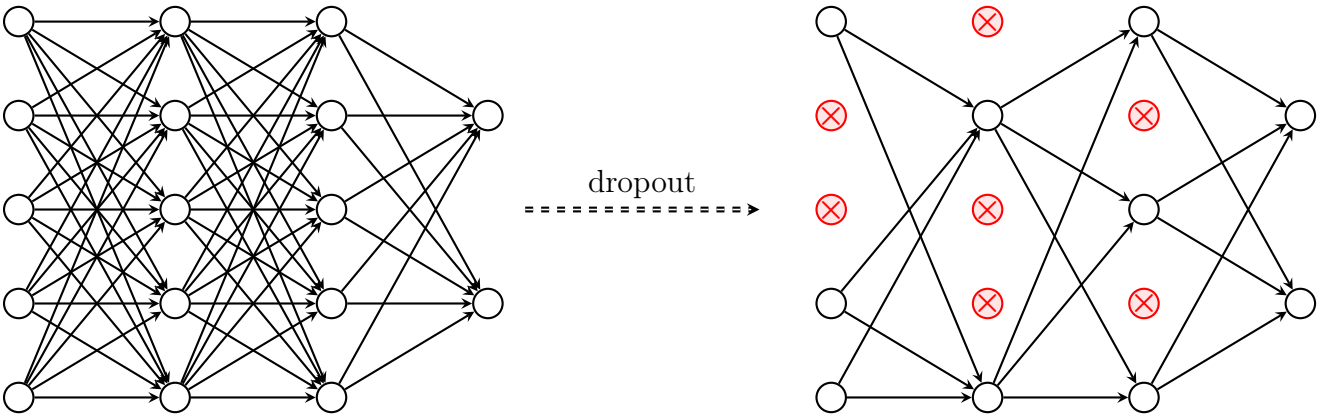
Слика 2.13: Приказ модификованих верзија исте слике

Како би учење било успешније, тренинг подаци треба да имају широк спектар варијација. То можемо постићи и модификовањем тренутног скупа на неке од следећих начина:

- Можемо вештачки додавати шум на наше податке (било да су то слике, аудио фајлови, текстуални подаци).
- Слике можемо ротирати, транслирати или рефлектовати.
- На слике можемо примењивати филтере, повећавати и смањивати осветљеност.

2.7.3 Dropout

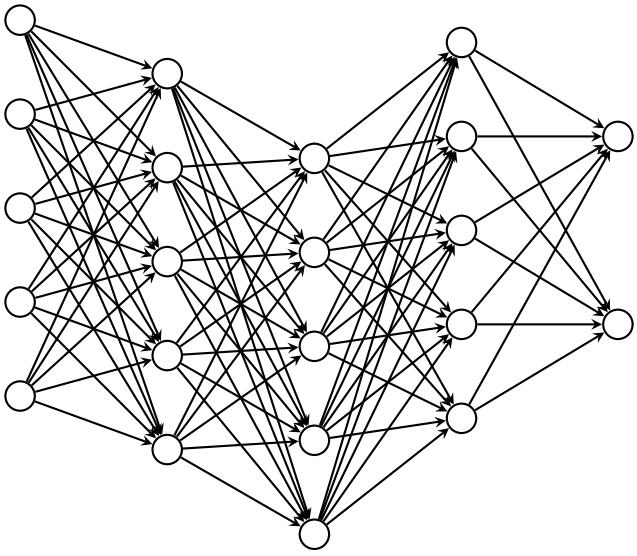
Dropout [6] представља оптимизациону технику која служи да спречи појаву overfitting-а. Наиме, често се дешава да неки перцептрони постану активнији од других, тј. они



Слика 2.14: Функционисање Dropout методе.

много више утичу на резултат од осталих. Проблем који се јавља је да уколико тај активнији перцептрон није добро научио функцију коју апроксимира, и греши, грешиће и цела мрежа. Dropout решава овај проблем тако што перцептронима додељује вероватноћу да буду мртви (неактивни) током једне епохе. На тај начин, мрежа ће се навикнути на постојање неисправних неурона, па ће се ослањати на "заједничке одлуке" већине неурона, радије него на мали скуп "јаких" неурона. Dropout је активан само током тренинга.

2.7.4 Нормализација узорка



Слика 2.15: Померај унутрашње коваријансе

Нормализација узорка [7] (енг. Batch Normalization) представља оптимизациону техинку за смањење помераја унутрашње коваријенсе. Наиме, дубоке неуралне мреже су склоне

томе да мала промена параметара у почетним слојевима, драстично утиче на дубље слојеве. Због тога сваки слој треба да се навикне на различите дистрибуције улазних података. То резултира у потреби да се почетни параметри пажљиво одаберу као и да се користе мале брзине учења. Нормализација узорка решава тај проблем тако што узорак поставља тако да му је очекивање једнако 0, а стандардна девијација једнака 1, али и дајући могућност да мрежа врати узорак на одређено очекивање и стандардну девијацију ако је тако боље. Као резултат, наша мрежа је способна да много брже учи.

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

$$y_i = \gamma\hat{x}_i + \beta$$

где су γ и β параметри који се тренирају.

2.7.5 Хиперпараметри

Хиперпараметри представљају параметре који контролишу понашање и учење наше мреже. То могу бити број узорака у једној итерацији тренинга, брзина учења, али и број слојева, број неурона у сваком слоју, одабир активационе функције и функције губитка. Вредности ових параметара морају бити познати и фиксирани пре почетка учења. Они у многоме одређују перформансе наше мреже, стога је веома важно, понекад и пресудно, подесити хиперпараметре на одговарајуће вредности. Због тога, постоје и многа истраживања која се баве методама за тражење тих оптималних вредности.
Један од начина да одаберемо хиперпараметаре је да одвојимо један део тренинг скупа над којим ћемо тестирати дискретан скуп комбинација хиперпараметара, а након тога изабрати оне који дају најбоље резултате на том делу тренинг скупа.

2.7.6 Регуларизација

Регуларизација представља оптимизациону технику којом мрежу приморавамо да користи мање тежине. Разлог за то је што мање тежине омогућавају бољу контролу и стабилност учења, али и штите од overfittinga. То се постиже додавањем члана на функцију грешку који расте са растом тежина. Размотрићемо две често коришћене регуларизације:

L1 регуларизација $\sum_{i=1}^{n} |w_i|$
L2 регуларизација $\sum_{i=1}^{n} (w_i)^2$
Такође, уводи се и хиперпараметар λ који представља снагу регуларизације, односно у којој мери се тежи ка мањим тежинама.

2.8 Неке функције грешке

2.8.1 Унакрсна ентропија

Функција грешке унакрсне ентропије је функција грешке која се користи за пробабилистичку класификацију и дефинисана је на следећи начин:

$$E(y, y') = \sum_i y'_i \log y_i$$

где је y предвиђен излаз, а y' прави излаз.

2.9 Неки алгоритми оптимизације

2.9.1 Импулс

Један од проблема стохастичног опадајућег градијента је то што се дешава да не иде директно ка минимуму, већ иде цик-цак путањом услед различитог тренинг подскупа. Један од начина да тај ефекат умањимо јесте увођење импулса који у некој мери чувати претходне помераје.

$$\vec{v} \leftarrow \gamma \vec{v} - \lambda \nabla F(\vec{x})$$

$$\vec{x} \leftarrow \vec{x} + \vec{v}$$

где је γ хиперпараметар који нам говори у којој мери чувамо претходно ажурирање.

2.9.2 RMSProp и Adam

Следећи корак у унапређену метода учења је и промена брзине учења у току самог учења тј. њено прилагођавање тренутној ситацији. Као што смо видели када смо дефинисали метод опадајућег градијента, уколико је брзина учења велика, брзо ћемо се кретати ка минимуму а онда ћемо почети да осцилујемо око њега, са друге стране ако је брзина мала, доћи ћемо до минимума али за то ће нам требати много времена. Логичан спој ова два дела је да у почетку имамо велику брзину и да се крећемо ка минимуму брзо, а након

тога да смањимо брзину како не би осцилирали већ достигли минимум. Ово постижемо коришћењем алгоритама као што су RMSProp и Adam, који су описани испод.

RMSProp(λ, γ)

$m \leftarrow 0$
$g \leftarrow 0$
while not converged
    $\vec{m} \leftarrow \gamma\vec{m} + (1-\gamma)\nabla F(\vec{x})$
    $\vec{g} \leftarrow \gamma\vec{g} + (1-\gamma)\nabla F(\vec{x})^2$
    $\vec{v} \leftarrow \gamma\vec{v} - \frac{\lambda \nabla F(\vec{x})}{\sqrt{g - m^2 + \epsilon}}$
    $\vec{x} \leftarrow \vec{x} + \vec{v}$

Adam(α, β_1, β_2) [8]

$t \leftarrow 0$
$m \leftarrow 0$
$v \leftarrow 0$
while not converged
    $t \leftarrow t + 1$
    $g \leftarrow \nabla F(\vec{x})$
    $m \leftarrow \beta_1 m + (1-\beta_1) g$
    $v \leftarrow \beta_2 v + (1-\beta_2) g^2$
    $\hat{m} \leftarrow \frac{m}{1-\beta_1^t}$
    $\hat{v} \leftarrow \frac{v}{1-\beta_2^t}$
    $\vec{x} \leftarrow \vec{x} - \alpha\hat{m}/(\sqrt{\hat{v}} + \epsilon)$

Параметар ϵ представља веома мали број који служи за постизање нумеричке стабилности, односно спречавање дељења са нулом.

2.10 Неки модели мрежа

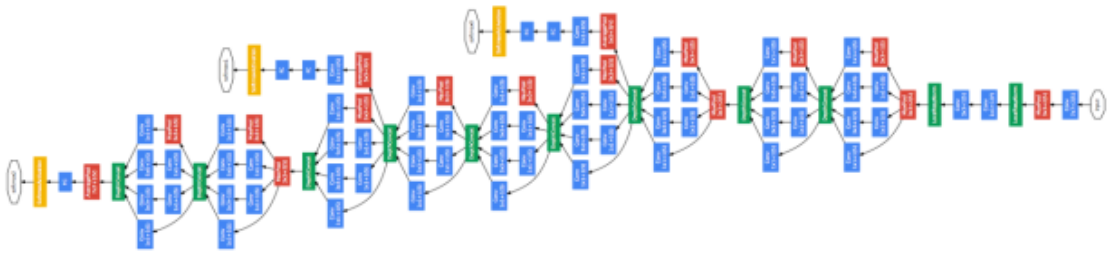
У овом делу биће приказани неки нестандардни модели неуралних мрежа, који често постижу боље резултате неко стандардни модел и представљају активан правац научног истраживања.

2.10.1 Ансамбл мрежа

Ансамбл мрежа представља низ различитих мрежа које су независно једна од друге трениране и независно доносе своју предикцију, а након тога ми одлучујемо финалну предикцију ансамбла. Мотивација за овакву врсту архитектуре је претпоставка да

ће различите мреже правити различите грешке, а да ћемо слушајући предикције више мрежа направити праву предикцију. Доношење финалне предикције можемо урадити на два начина: методом гласања — предикција за коју се највећи број мрежа сложи је финална, методом просечне вероватноће — предикција која имају највећу просечну вероватноћу је финална предикција. Да би осигурали да различите мреже праве различите резултате можемо извршити додатну обраду улазних података за сваку мрежу другачије, или можемо из тренинг скупа извлачити са понављањем тренинг примере и извучене користи као тренинг скуп.

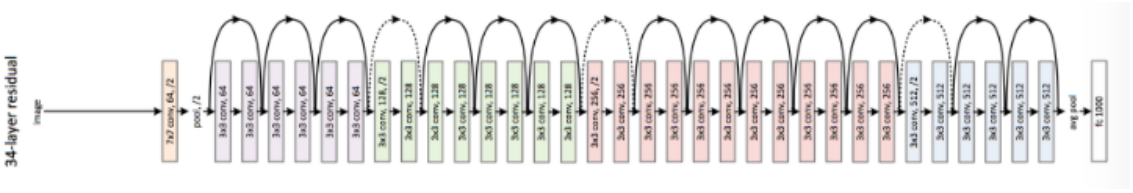
2.10.2 Inception



Слика 2.16: Приказ Inception модела

Inception [9] модел представља тип који се још назива мрежа у мрежи, односно архитектура где је више мрежа повезано паралелно.

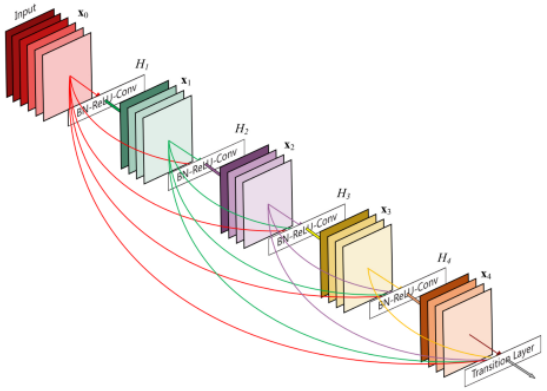
2.10.3 ResNets



Слика 2.17: Приказ ResNet

Residual Networks [10] представљају нови тип архитектуре који значајно решава проблем нестајућег градијента и тиме омогућава ширење у дубину мрежа до 100-1000 слојева. До сада излаз неког слоја је био $F(x)$ где је F функција тог слоја, међутим резидуалне мреже за сваки слој уводе пречицу која га заобилази па је излаз $F(x) + x$. На тај начин не само да решавамо проблем нестајућег градијента, већ и омогућавамо самој мрежи да "одлучи" колико жели да иде у дубину.

2.10.4 DenseNets



Слика 2.18: Приказ DenseNet

Dense Networks [11] представљају надоградњу претходног типа где уместо додавања пречице за само један слој, додајемо пречице од сваког до сваког слоја.

2.10.5 Енкодер/Декодер мрежа

Мотивација за ову мрежу потиче из компресије података са губитком, где желимо да компресујемо што више, а да при томе можемо да реконструишемо довољно добро почетне податке. Како смо видели да дубоке неуралне мреже имају способност да уочавају нека занимљива својства између података, користићемо њих да одлуче које информације о подацима ће сачувати. Направићемо мрежу којој је улаз и излаз димензионо исти, а неки од унутрашњих слојева садржи количински мање информација од оригиналних података. Такву мрежу тренираћемо тако да на излаз враћа исто оно што је на улазу. Процес кодирања података би био рачунање излаза тог слоја са мање података, док би процес декодирања рачунање излаза мреже уколико у тај слој убацимо енкодоване податке.

Поред оригиналне примене, овакву мрежу можемо тренирати и да примени одређен филтер на податке, тј. можемо учити мреже на пару (улаз: оригинални податак, излаз: примењен филтер), након тога мрежа ће приближно примењивати такав филтер на податке.

Глава 3

Имплементација

У овом поглављу описаћемо проблеме које решавамо, анализирати могућа решења, објаснити алгоритме које користимо и приказати струкуре наших мрежа.

3.1 Препознавање саобраћајних знакова

Слика 3.1: Саобраћајни знаци

Једна од важних ствари које самовозећи аутомобили треба да уради је да препознају саобраћајне знаке и да се понашају у складу са њима. Применом конволуцијске неуралне мреже направљен је систем који успешно препознаје који је саобраћајни знак на слици.

Скуп података који је коришћен је **German Traffic Signs**[12]. Тренинг скуп има 39209 слике, на којима се налазе 43 различита знака, док тест скуп садржи 12630 слика. Поред оригиналних слика, постоје и различите верзије сваке слике направљене променом осветљености.

Сваки знак је исечен тако да заузима комплетну слику и примењена је нормализација боја. Такође, пре тренинга примењена је и модификација тренинг скупа, како би се повећала варијабилност података. Подаци су модификовани додавањем ротације, транслације и извртања. Такође, како би постигли бољи резултат коришћен је асамбл састављен од 25 мрежа.

Слика 3.2: Примери знакова

У табели на слици 3.3 приказана је архитектура мреже која је постигла најбоље резултате на тест скупу. Тренирана је помоћу Adam алгоритма, а као функција грешке коришћена је унакрсна ентропија.

3.2 Детекција саобраћајних знакова

Детекција саобраћог знака представља проширење претходног проблема. Наиме, решење претходног проблема нам може успешно рећи који је знак на слици, при чему знак мора бити преко целе слике. У овом делу, направићемо систем који ће на слици улице успешно проналазити и препознавати знакове.
Систем за препознавање састојаће се из:

- Неуралне мреже која нам може рећи да ли се на слици налази знак или не.
  Ова мрежа је истренирана на тренинг скупу од око 750 000 слика. Слике на којима се налази знак су креиране сечењем већ постојећих знакова, а слике на којима се не налази знак су креиране сечењем слика улица на насумичне парчиће.
- Други део представља алгоритам клизећег прозора (енг. sliding window algorithm) који на слици где тражимо знакове превлачи прозор односно квадрат величине 25 x 25 пиксела и те парчиће пропушта кроз мрежу. Оне квадрате на којима мрежа каже да се налази знак чувамо.

Структура неуралне мреже		
BatchNormalization		
Convolution 2D	3x3	32 филтера
Activation	ReLU	BatchNormalization
Convolution 2D	3x3	32 филтера
Activation	ReLU	BatchNormalization
MaxPool2D		2x2
Dropout		0.25
Convolution 2D	3x3	64 филтера
Activation	ReLU	BatchNormalization
Convolution 2D	3x3	64 филтера
Activation	ReLU	BatchNormalization
MaxPool2D		2x2
Dropout		0.25
Convolution 2D	3x3	128 филтера
Activation	ReLU	BatchNormalization
Convolution 2D	3x3	128 филтера
Activation	ReLU	BatchNormalization
Dense		512 неурона
Activation	ReLU	BatchNormalization
Dropout		0.5
Dense		43 неурона
Activation		SoftMax

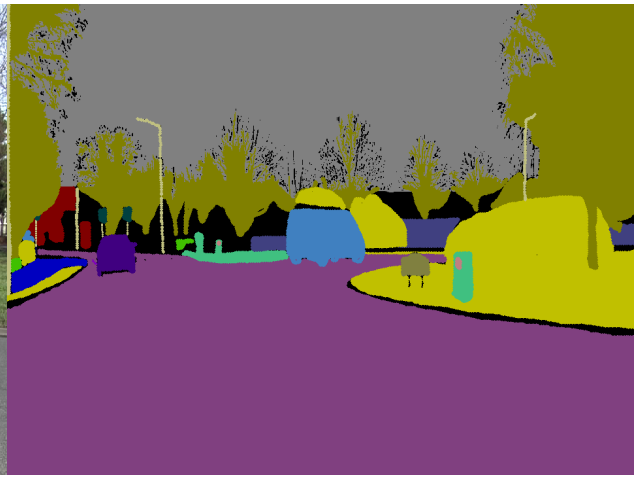
Слика 3.3: Структура мреже за препознавање знакова

Слика 3.4: Пример детектованих квадрата на којима се налазе знаци (без уклоњених грешака)

- Трећи део представља алгоритам који ће препознате квадрате око једног знака спојити и извести ограничавајући правоугаоник око тог знака. Такође, уклониће шум односно погрешно препознате квадрате.
- Сада, те препознате знакове пуштамо кроз прву мрежу која их класификује.

3.3 Сегментација слике на битне регионе

Слика 3.5: Слика улице



Слика 3.6: Сегментована слика улице

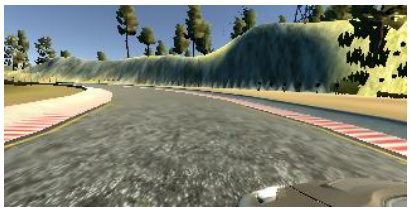
Структура неуралне мреже			
Convolution 2D	3x3	64 филтера	ReLu
MaxPool2D		2x2	
Convolution 2D	3x3	128 филтера	ReLu
MaxPool2D		2x2	
Convolution 2D	3x3	256 филтера	ReLu
MaxPool2D		2x2	
Convolution 2D	3x3	512 филтера	ReLu
MaxPool2D		2x2	
Convolution Transpose 2D	3x3	512 филтера	
Convolution Transpose 2D	3x3	256 филтера	
Convolution Transpose 2D	3x3	128 филтера	
Convolution Transpose 2D	3x3	64 филтера	
Convolution Transpose 2D	3x3	32 филтера	

Слика 3.7: Структура мреже за сегментацију слике

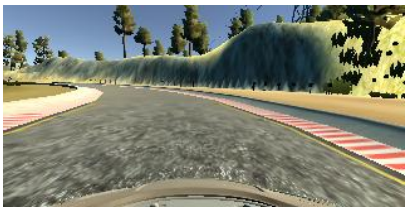
За овај проблем направићемо систем који ће слику улице раставити на повезане сегменте који ће бити класификовани према садржају (улица, небо, аутомобили, објекти са стране). На улазу ће бити слика улице, док ће излаз представљати исту ту слику где ће сваки битан сегмент бити означен одговарајућом бојом (нпр црвена за улице, плава за аутомобиле, сива за небо). Коришћен је скуп података **CamVid** [15]. Постоји 32 различите класе, а скуп података се састоји од 1402 слика, од тога је 701 слика настала рефлектовањем по X оси оригиналне. За валидациони скуп се користи 100 слика, док је остатак коришћен за тренинг. Све слике су скалиране на величину 128 x 128. Улаз у мрежу представља матрица 128 x 128 x 3, а излаз је 128 x 128 x 32, излаз представља за сваки пиксел низ вероватноћа да он припада некој класи. За решење овог проблема користићемо енкодер-декодер мрежу чија структура је приказана на слици 3.7.

3.4 Управљање у симулацији

У овом делу имплементираћемо управљање аутомобила у симулацији уз помоћ машинског учења. Наш алгоритам ће учити на основу снимка управљања и преузетих акција. Тест пример ће се састојати од низа слика са централне, леве и десне камере, а излаз тога ће бити угао скретања. Дакле наш алгоритам ће гледајући како неко други вози, научити тј. успешно клонирати понашање возача из тренинг примера, и на основу њега и сам возити аутомобил.



Слика 3.8: Лева



Слика 3.9: Центарална

Слика 3.10: Десна

Наш алгоритам ће учити да апроксимира угао скретања на основу слике, и на тај начин учити да вози по стази.
За учење је коришћен тренинг скуп генерисан правилним вожењем аутомобила по стази, тај скуп података је проширен креирањем рефлектованих слика и негирања угла, а алгоритам је након тога тестиран тако што је алгоритму дата потпуна контрола да управља возилом.

3.5 Учење са појачавањем

У овој секцији биће описана три система која представљају управљачке задатке релевантне за навигацију возила, над којима су тестирани алгоритми учења са појачавањем описани у претходној глави.

3.5.1 Систем 1

За први задатак, самостално сам израдио систем приказан на слици 3.12.
Систем је матрица 10 x 10, играч је представљен 2 x 2 блоком у првом реду одоздо, одозго долазе блокови са насумично распоређеним рупама, који у сваком кораку падају једно поље вертикално наниже. Уколико коцкица удари у нашег играча, игра се завршава, уколико не дође до судара, добијамо један поен.

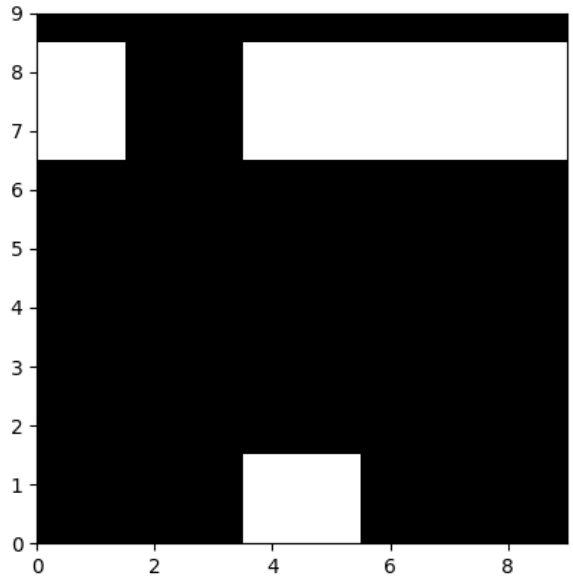
Постоје три могуће акције:

- померити играча једно поље улево
- остати на истом пољу
- померити играча једно поље удесно

Наш алгоритам ће научити да управља системом како би максимизовао број поена без икаквог претходног познавања система.

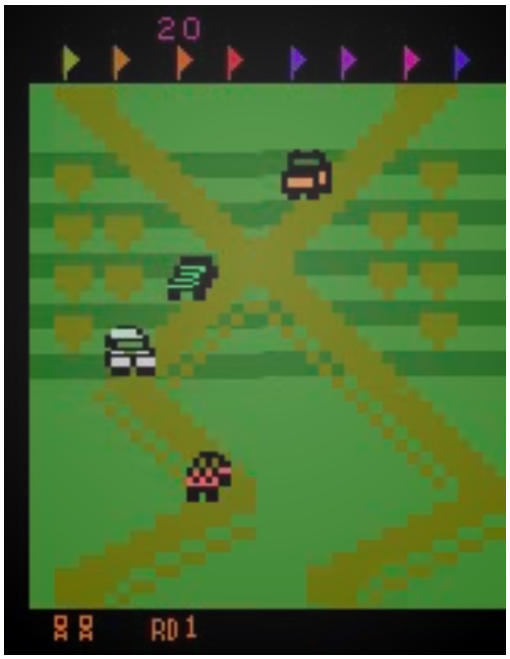
Структура неуралне мреже		
Convolution 2D	3x3	8 филтера
Activation	ReLU	BatchNormalization
MaxPool2D		2x2
Convolution 2D	3x3	16 филтера
Activation	ReLU	BatchNormalization
MaxPool2D		2x2
Dropout		0.25
Convolution 2D	3x3	32 филтера
Activation	ReLU	BatchNormalization
MaxPool2D		2x2
Dropout		0.25
Convolution 2D	3x3	64 филтера
Activation	ReLU	BatchNormalization
MaxPool2D		2x2
Convolution 2D	3x3	64 филтера
Dense		512 неурона
Activation	ReLU	BatchNormalization
Dense		256 неурона
Activation	tanh	BatchNormalization
Dense		25 неурона
Dense		1 неурон

Слика 3.11: Структура мреже за управљање у симулацији



Слика 3.12: Графички приказ система

3.5.2 Систем 2 — Игра UpNdown



Слика 3.13: Атари игрица UpNdown

Систем 2 представља Атаријеву игрицу UpNdown. Циљ игрице је преживети што дуже управљајући аутићем, и уништити што више противника скакањем на њих, за шта се добијају и додатни поени. Улаз представља матрицу 210 x 160 x 3, која представља

снимак екрана игрице, могуће је извести укупно 6 акција. После сваке акције, систем нам враћа награду односно број поена који смо освојили. Наш алгоритам ће научити да управља аутићем како би освајао што више поена. За учење је коришћен алгоритам A3C.

3.5.3 Систем 3 — Игра Enduro

Слика 3.14: Атари игрица Enduro

Систем 3 представља Атаријеву игрицу Enduro. Циљ игрице је избегавати судare са другим аутомобилима, као и престићи што више играча. Улаз представља матрицу 210 x 160 x 3, која представља снимак екрана игрице, могуће је извести укупно 9 акција. После сваке акције, систем нам враћа награду односно број поена који смо освојили. Наш алгоритам ће научити да управља аутићем како би освајао што више поена. За учење је коришћен алгоритам A3C.

Структура неуралне мреже		
Convolution 2D	8x8	16 филтера
Convolution 2D	4x4	32 филтера
$V(s)$	$\pi(s)$	
Dense 256 neurons	Dense action num neurons	
Dense 1 neuron	/	

Слика 3.15: Структура мреже за A3C алгоритам

Глава 4

Евалуација

У овом делу приказаћемо неке величине који су показатељи колико успешно су наши системи из претходног поглавља научили да препознају или класификују објекте. Такође, биће приказани неки занимљиви графици који приказују процес учења и разна поређења различитих архитектура и поменутих оптимизационих техника.

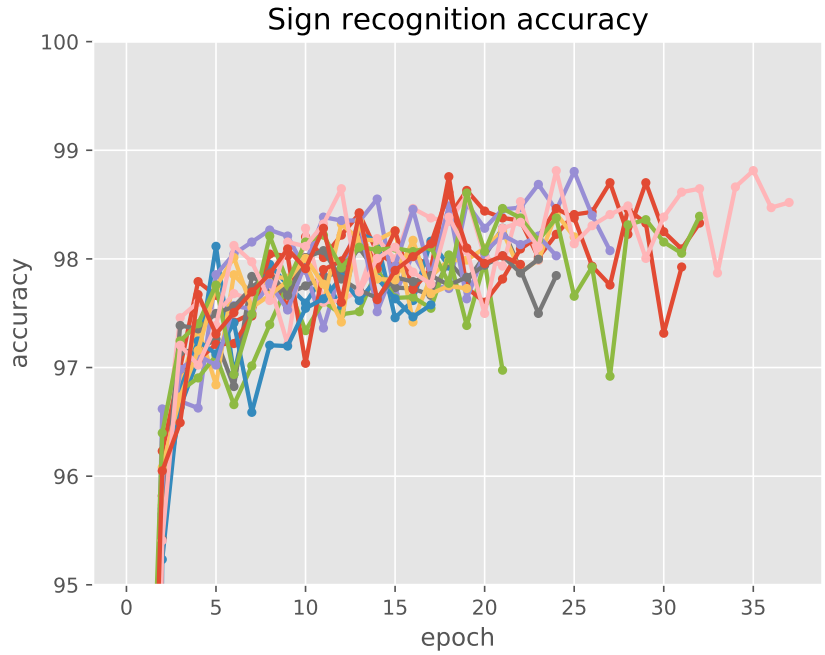
4.1 Препознавање саобраћајних знакова

Тачност
99.29%

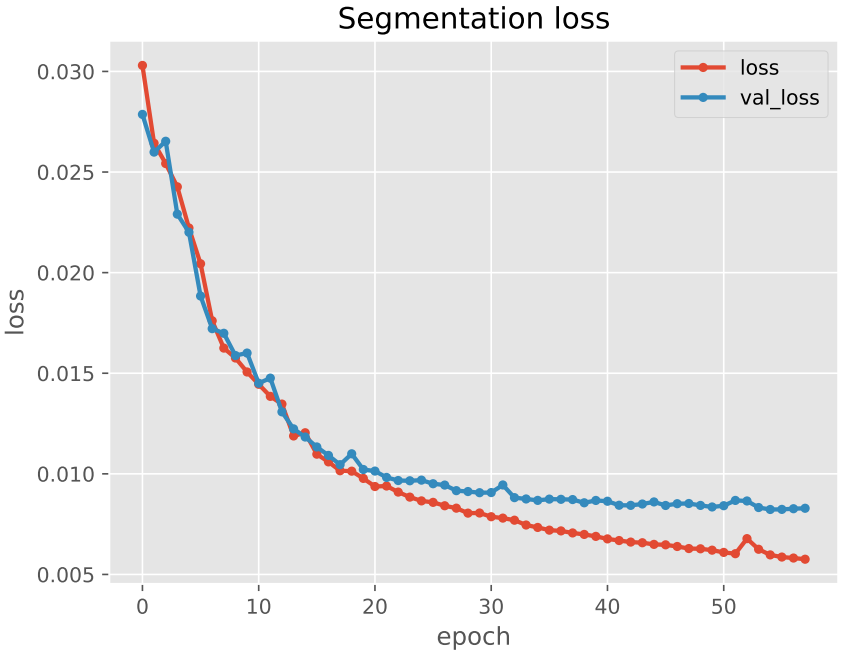
Најбоља постигнута тачност на тест скупу је 99.29%, што је за око 0.5% више од људског перформанса. На слици 4.1 приказан је график тачности појединачних модела кроз епоху. Можемо приметити да иако је највећа тачност појединачних модела испод 99%, удружени у асамбл постижу већу тачност.

4.2 Сегментација слике

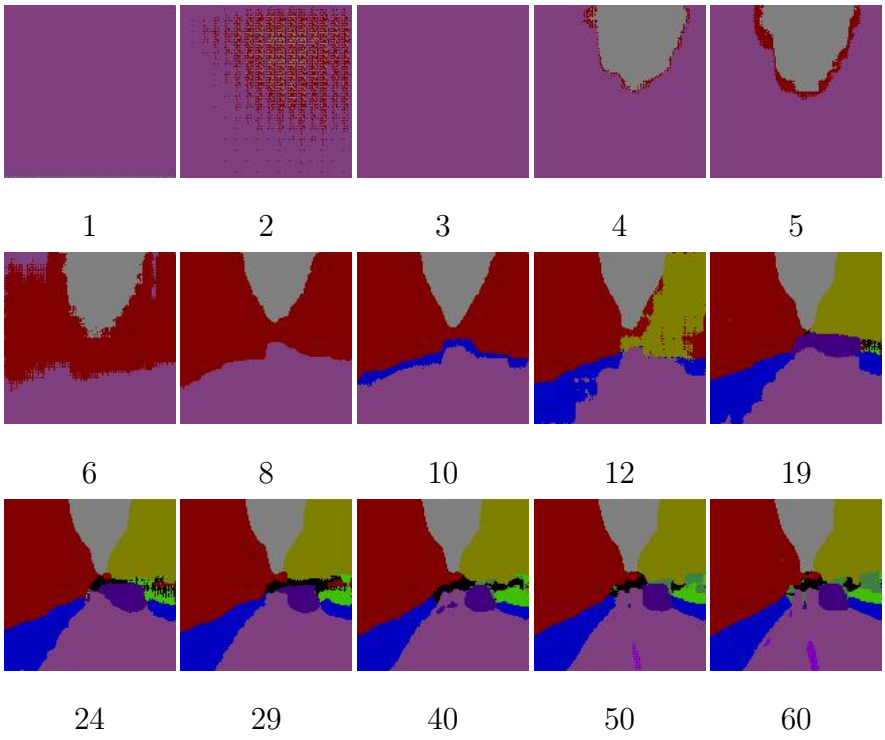
На слици 4.2 је приказан график вредности функције грешке у зависности од епохе на тренинг сету као и на тест сету. На слици 4.3 је приказана предикција мреже кроз епохе на валидационој слици. На слици 4.4 су приказани примери оригиналне, циљане слике као и предикције мреже.



Слика 4.1: График тачности појединачних модела кроз епоху



Слика 4.2: Приказ функције грешке по епохи



Слика 4.3: Приказ предикције кроз епоху на валидационом примеру

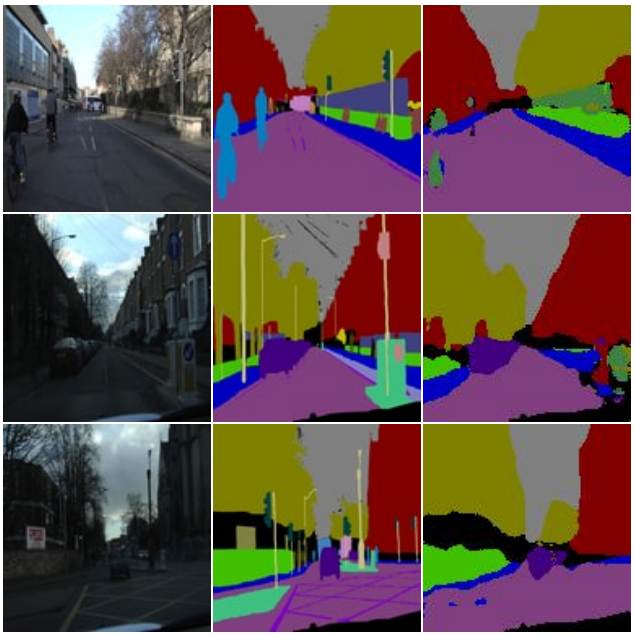
4.3 Симулација вожње

Истренирана мрежа је способна да самостално управља аутомобилом. На слици 4.5 приказане су слике настале током вожње од стране алгоритма.[1]

4.4 Учење са појачањем

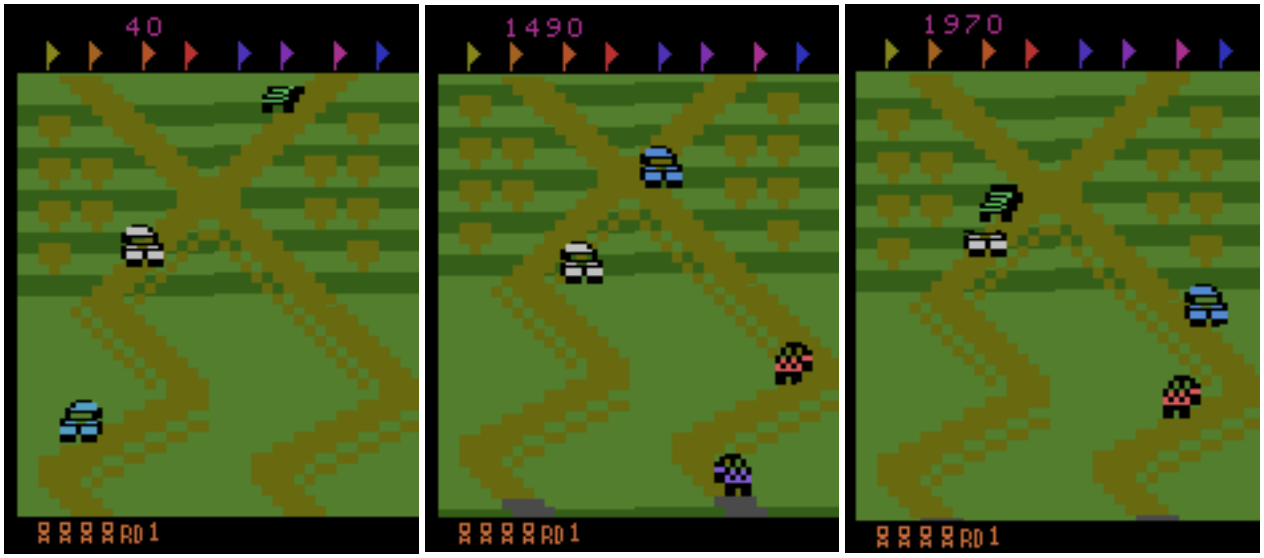
A3C алгоритам успева да осваја преко 30 хиљада поена у игрици UpNDown. На слици 4.6. приказно је окружење у току игре. Можемо приметити да је алгоритам нашао неку необичну стратегију, па уместо да вози ауто, он ће стајати у месту и скакати, и на тај начин уништавати непријатеља.[2]

[1]На линку **https://youtu.be/1qbdMxSO1zo** може се погледати видео вожње.
[2]На линку **https://youtu.be/-KNs9uTIdvs** може се погледати видео играња игре UpNDown.



Слика 4.4: Приказ оригиналне, циљане слике и предикције мреже

Слика 4.5: Приказ аутомобила управљаног помоћу неуралне мреже у току вожње



Слика 4.6: Приказ окружења у току игре

Глава 5

Закључак

Кроз овај рад упознали смо се са основама машинског учења и неуралним мрежама кроз супервизирано учење, као и кроз учење са појачањем. Приказали смо честе грешке, оптимизационе технике, као и неке нестандардне моделе неуралних мрежа, које пружају боље резултате учења. Затим су кроз решавање проблема везаних за вожњу, приказани различити приступи за решавање тих проблема, али и права способност неуралних мрежа да уоче и науче законитости међу подацима.

На самом крају овог рада желео бих да изразим захвалност:

- **Петру Величковићу**, мом ментору, на помоћи приликом израде и писања овог рада, посвећеном времену, на огромном доприносу у ширењу мојих информатичких знања све четири године из алгоритама као и из других области. Исто тако као једном од оснивача и организатора Недеље информатике, која ме је приближила и заинтересовала за многе до тада невиђене области рачунарских наука.
- **Јелени Хаџи Пурић**, лидеру тима Математичке гимназије за информатичка такмичења.
- **Жељку Лежаји**, **Наташи Чалуковић**, као и свим другим професорима у Математичкој гимназији, на пренетом знању, уложеном труду и разумевању.
- **свим осталима** који нису наведени, а допринели су стварању овог рада или унапређивању мог информатичког знања.

У Београду, мај 2017.
Филип Весовић

Литература

[1] Deep learning, Yann LeCun, Yoshua Bengio and Geoffrey Hinton, Nature, 2015

[2] Learning representations by back-propagating errors, David Rumelhart, Geoffrey Hinton, and Ronald Williams, Nature, 1986

[3] Human-level control through deep reinforcement learning, Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg and Demis Hassabis, Nature, 2015

[4] Long short-term memory. Sepp Hochreiter, Jürgen Schmidhuber, 1997

[5] Asynchronous Methods for Deep Reinforcement Learning, Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver and Koray Kavukcuoglu, ICML, 2016

[6] Dropout: a simple way to prevent neural networks from overfitting, Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov, Journal of Machine Learning Research, 2014

[7] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Sergey Ioffe and Christian Szegedy, ICML, 2015

[8] Adam: A Method for Stochastic Optimization, Diederik P. Kingma, Jimmy Ba, 2014

[9] Going Deeper with Convolutions, Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, 2014

[10] Deep Residual Learning for Image Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, 2015

[11] Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Kilian Q. Weinberger and Laurens van der Maaten,2016

[12] Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, J. Stallkamp, M. Schlipsing, J. Salmen and C. Igel

[13] Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark, Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing and Christian Igel

[14] Udacity Self-Driving Car Simulator
https://github.com/udacity/self-driving-car-sim.

[15] Semantic object classes in video: A high-definition ground truth database, Brostow, Gabriel J. and Fauqueur, Julien and Cipolla, Roberto

[16] Supervised learning, Petar Veličković
https://github.com/PetarV-/TikZ/tree/master/Supervised%20learning%20setup.

[17] 2D Convolution image, Petar Veličković
https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution.

[18] Dropout image, Petar Veličković
https://github.com/PetarV-/TikZ/tree/master/Dropout.

[19] Multilayer perceptron image, Petar Veličković
https://github.com/PetarV-/TikZ/tree/master/Multilayer%20perceptron.

[20] Long short-term memory image, Petar Veličković
https://github.com/PetarV-/TikZ/tree/master/Long%20short-term%20memory.

[21] Activation functions and Overfitting image, Petar Veličković, MG CSWeek seminar
http://www.csnedelja.mg.edu.rs/static/resources/v3.0/sre1_neuralne_pv.pdf.

[22] Overfitting image, Wikipedia
https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Overfitting.svg/685px-Overfitting.svg.png.

[23] Data augmenation image, Keras blog
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

[24] Residual networks image, Felix Lau
http://felixlaumon.github.io/2015/01/08/kaggle-right-whale.html.

[25] Dense networks image
https://github.com/liuzhuang13/DenseNet

[26] Inception model image
http://joelouismarino.github.io/blog_posts/blog_googlenet
_keras.html