МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета Рачунарство и информатика

НЕУРАЛНЕ МРЕЖЕ

Ученик

Срђан Марковић, IVд

Ментор

Јелена Хаџи-Пурић

Београд, мај 2016.

САДРЖАЈ

1. Увод у машинско учење

Проучавајући зашто је људска раса постала толико доминантна на читавом свету научници долазе до одговора да је кључ наше доминације у способности да брзо учимо, да се сами прилагођавамо на ситуације са којима се никада нисмо сусрели. За разлику од нас, наш заменик на све већем броју радних места, компјутер није имао ту способност. У данашње време, људи су нашли начин да науче компјутер да сам учи у машинском учењу.

Машинско учење је грана науке о компјутерима која се бави проучавањем и развијањем алгоритама који имају могућност да уче из претходно унетих примера и на основу њихових резултата обрађују наредне улазе, а не поштују нека унапред одређена правила.

Где наилазимо на примене машинског учења у свакодневном животу?

Сви сте сигурно корисници You Tube-а. Примећујете да када уђете на почетну страну сајт вам сам предлаже снимке за које претпоставља да би вам се могли свидети. Приметићете да уколико одете на You Tube користећи компјутер неког ватреног навијача он ће вам предлагати снимке са најбољим потезима из неких утакмица док би на компјутеру неког гитаристе наишли на предлоге који садрже снимке са неких концерата. Како сајт зна ког корисника интересује који садржај? Тако што учи и користи информације из свих претходних коришћења тог сајта. Он памти какве је упите корисник најчешће постављао и какве је садржаје тражио па му на основу тога нуди снимке са садржајима који ће га највероватније интересовати. Што више користимо тај сајт, то ће он о нама више научити и бити способан да боље задовољи наше потребе.

Још један пример су програми за аутоматско превођење текста. Уколико би желели са тачношћу да преведемо текст морали би прво у потпуности да га разумемо зато што неке речи у зависности од контекста могу имати разна значења. Како људи такве ситуације решавају користећи се осећајем и искуством које стичу читајући, то значи да ће и компјутер овакве проблеме најбоље решавати користећи се искуством; то јест, примерима већ постојећих превода.

1.1. Типови и класификација машинског учења

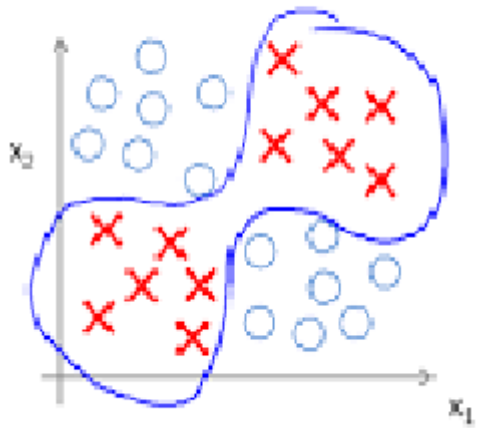
Машинско учење се најчешће дели у три категорије, у зависности од врсте података којима систем барата:

1. *Supervised learning* - Програму се унапред да група тест примара са познатим одговорима из којих он треба да научи правила по којима ће да решава нове улазе.

2. *Unsupervised learning* - Никакви примери нису унапред дати програму. Он сам мора да нађе начин да у зависности од различитих улаза даје различита решења.

3. *Reinforcment learning* - Програм интерагује са динамичким окружењем без тачно одређених тачних одговора; као на пример аутоматско вожење аутомобила.

Завршавамо ово поглавље појмом *класификације*. Проблеми класификације су најзначајнија група проблема из области машинског учења из групе *Supervised learning*. Код њих на основу примера, где су одређени подаци сврстани у неке категорије (класе), треба пронаћи правило по којем ће се подаци убудуће смештати у одређене категорије.



слика 1: Дељење објекта на кругове и икс-ове.

Најпознатији пример класификације из свакодневног живота је програм за одређивање да ли је *email spam* или није, то јест, да ли садржи информације од значаја за корисника или не. Тада имамо само две класе у које смештамо email-ове: корисни или не. Наизглед једноставан проблем ипак значајно компликује то што различити корисници различите ствари сматрају корисним. Чак и један корисник током времена може променити свој став о томе шта је *spam*, а шта није. Рецимо корисник може сматрати корисним информације о тренутним попустима у локалној продавници, а након евентуалне селидбе те информације ће за њега постати врло некорисне.

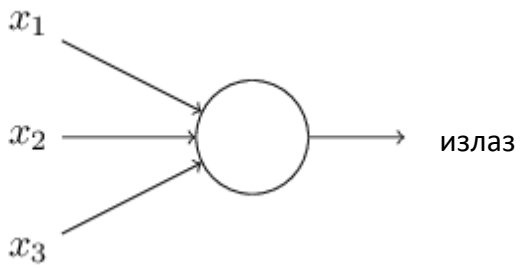
2. Увод у неуралне мреже

Инспирисани начином учења код људи, научници конструишу неуралне мреже, по угледу на нервни систем. Оне се састоје од вештачких неурна међусобно повезаних неким везама. Неуралне мреже су осмишљене да раде на принципима машинског учења и једна су од најкоришћенијих техника у области *supervised learning-a*.

Да би што боље разумели неуралне мреже најбоље је да их пратимо баш онако како су се оне и развијале. То значи да се прво треба упознати са појмом *percepron*-а осмишљеног 1958. од стране *Frank-a Rosenblatt-a*.

2.1. Perceptron

Perceptron је тип вештачког неурона који прима неколико бинарних улаза и даје тачно један бинарни излаз који прослеђује неком другом *perceptron*-u.



слика 2: Пример perceptrona са 3 улаза

У почетку је постојао веома једноставан начин за израчунавање излаза. Свакој вези била је додељена одређена тежина која је била неки рационалан број, која је одређивала важност те везе. Нека су улази означени са x_1, x_2,.. а нека су одговарајуће тежине веза w_1, w_2,... Вредност на излазу добијамо у зависности од тога да ли је $\sum_j w_j * x_j$ већа или мања од неке граничне вредности. Гранична вредност је карактеристика сваког неурона и он за излаз прослеђује 1 уколико је поменута сума већа од границе, а прослеђује 0 уколико је мања или једнака од границе.

То је начин рада једног *perceptron*-а. Можда за сада делује бесмислено зашто овакав начин рада може бити користан па ћемо га мало појаснити кроз једноставан пример.

Рецимо да вас је неко позвао да идете да идете у биоскоп. Да бисте донели одлуку да ли желите да одете на филм разматрате следеће факторе:

1. Да ли вам се допада филм?

2. Да ли је биоскоп близу?

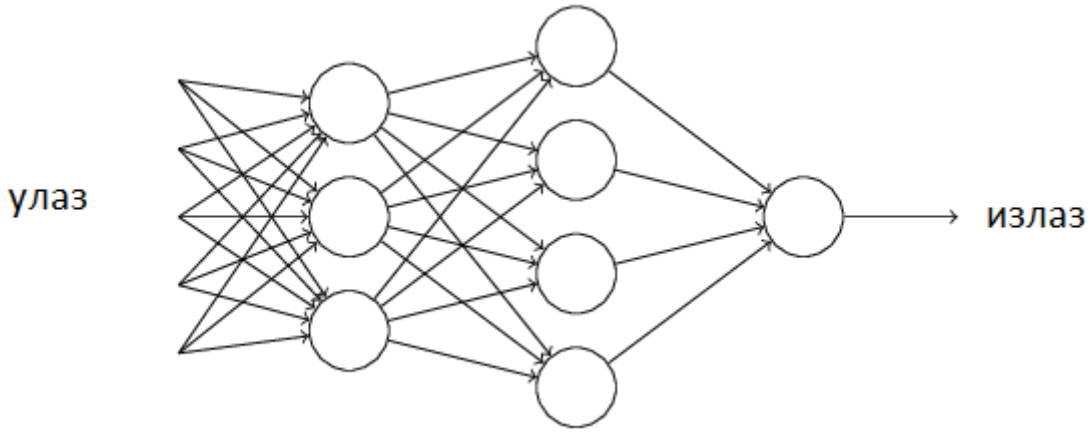
3. Да ли је особа која вас је позвала ваш добар пријатељ?

Ове факторе ћемо представити вредностима x_1, x_2 и x_3 редом и свакој променљивој доделићемо 1 уклико је услов испуњен, а 0 ако није. Рецимо x_1=1 ако вам се филм свиђа, а x_1=0 у супротном. Рецимо да су одговарајуће тежине w_1=3, w_2=3 и w_3=6, а граница за овај *perceptron* је 5. Ово значи да би ми свакако отишли на филм уколико је та особа наш добар пријатељ јер знамо да ћемо се са њим добро провести чак и ако идемо на лош филм који је далеко, а да ћемо филм гледати са особом која није наш пријатељ само уколико је филм добар и уколико је биоскоп близу јер знамо да би нам у путу било толико досадно да ни добар филм то не би надокнадио.

Приметићемо да би мењањем тежина веза или граничне вредности добили друге резултате. Спуштањем граничне вредности на 2 ми би у биоскоп ишли да се испуни ма који од три услова. Такође можемо приметити да оне гране које су нам битне, које желимо да имају већи утицај у нашој одлуци, имају већу тежину од мање битних.

2.1.1. Мрежа perceptron-а

Јасно нам је да користећи један *perceptron* не можемо доносити превише сложене одлуке већ да нам је за тако нешто потребна њихова мрежа.



слика 3: Пример мреже *perceptron*-а.

Објаснимо сада како изгледа мрежа. Она се дели на три основна слоја: улазни, излазни и скривени који у ствари може имати произвољно много слојева. Улазни слој служи да каже име за унос података у мрежу, док излаз служи за приказ резултата. Скривени слој се тако зове из разлога што он не врши директно никакву интеракцију са спољним светом већ обрађује информације које добија из првог омотача и прослеђује их у наредни омотач.

Приликом дефинисања *perceptorn*-а рекли смо да он има тачно један излаз, а на примеру мреже примећујемо да неки од њих имају више излаза. Ти излази су у ствари сви један излаз зато што сви носе исту вредност само је прослеђују разним *perceptorn*-има.

Да би *perceptron* начинили лакшим за баратање, уводим појам *bias*-а.

Bias, *b* - гранична вредност тако да на излазу *perceptron*-а, добијамо 0 уколико је

$\sum_j w_j * x_j + b \leq 0$,а 1 уколико $\sum_j w_j + x_j + b > 0$.

2.1.2. Недостаци perceptron-а

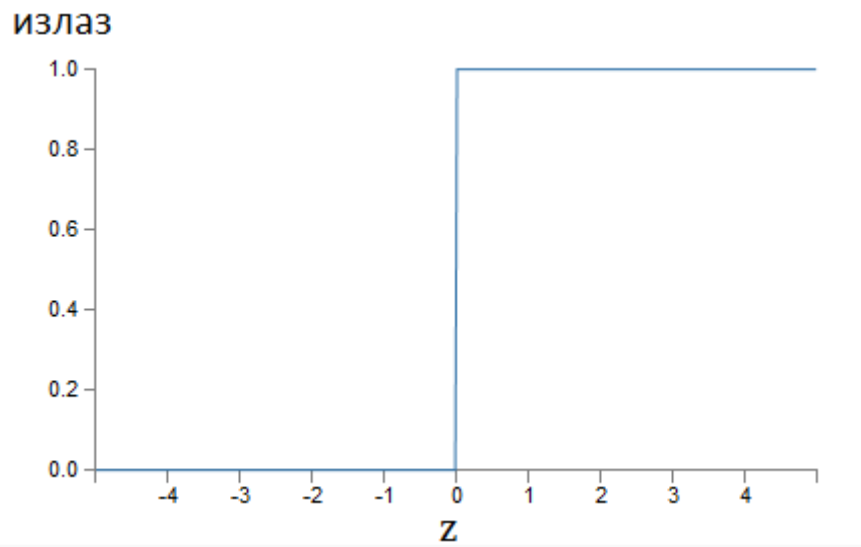
Вратимо се сада на проблем машинског учења. Да би натерали мрежу *perceptron*-а да учи морамо наћи одговарајући алгоритам који ће да мења вредности *bias*-а и тежине веза све док не добијемо жељени резултат. Рецимо да желимо да направимо мрежу која ће да препознаје ручно писане цифре. Претпоставимо да она грешком препознаје 9 као 8. Алгоритам направи малу промену у тежинама или *bias*-у и сада наша мрежа тачно препознаје 9. Међутим, овом приликом може се десити да сада она лоше препознаје 3 као 2. То се догађа зато што *perceptron* првенствено барата са целим бројевима и није могуће правити довољно мале промене.

2.2. Sigmoid неурон

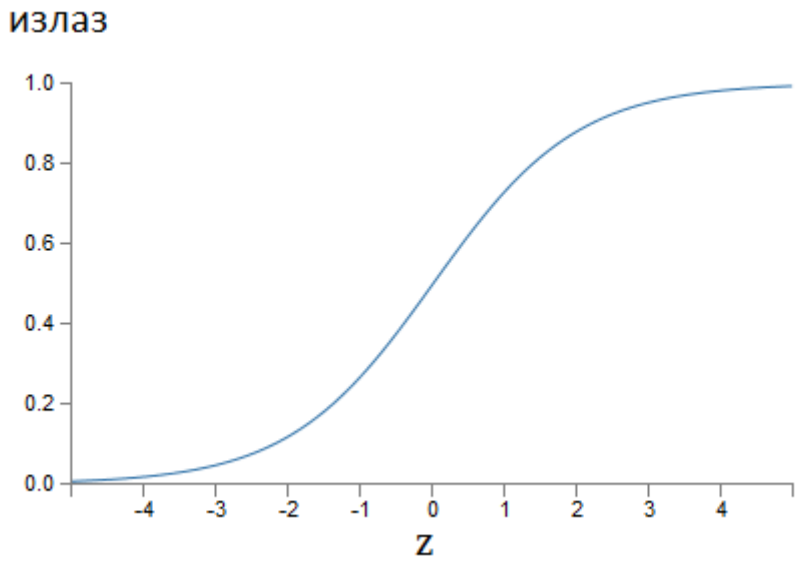
Да би превазишли овај проблем уводимо нови тип неурона који се зове *sigmoid* неурон. Као и код *perceptron*-а претставимо његове улазе са x_1, x_2,x_3... Разлика у односу на претходни тип неурона је у томе што сада улази не морају бити само јединице и нуле већ било који реални број између 1 и 0. Такође, свака веза има своју тежину и сваки неурон има свој *bias* - b. Нова разлика је у начину на који рачунамо излаз из неурона. Излаз се добија израчунавањем вредности *sigmoid* функције са параметром $\sum_j w_j * x_j + b$, то јест:

$\sigma(z) \equiv \frac{1}{1+e^{-z}}$, где је $z = \sum_j w_j * x_j + b$.

Сада се може чинити да ове две претстављене врсте неурона немају никаквих сличности, али то није сасвим тачно. Уколико би z->∞ онда би e^{-z}->0 па би $\sigma(z) \approx 1$, као и код *perceptron*-а. Такође за z->-∞ добијамо e^{-z}->∞ па би $\sigma(z) \approx 0$, баш као и код претходне врсте неурона. Ипак најбоље ћемо уочити сличност ако нацртамо графике обеју функција.



слика 4: Функција коју користи *perceptron*



слика 5: *Sigmoid* функција

Сада примећујемо да је *sigmoid* функција заобљена верзија функције коју користи *perceptron*, ова функција нема наглих скокова па је могуће вршити благе промене тежина и *bias*-а без већих последица.

Мреже неурона се у основи састоје од истих компонената као и мреже *perceptron*-а: улазни слој, излазни слој и кључни скривени слој. Од сложености последњег слоја зависи сложеност проблема који датом мрежом можемо решити.

Мреже се деле на два главна типа:

1. *feedforward мреже* - Њихова карактеристика је то да сваки неурон шаље сигнал ка неуронима из наредног слоја почев од улазног, све док не дођемо до излазног слоја. Један слој не почиње са радом све док сви неурони из претходног слоја не заврше свој посао. То значи да овакве мреже сваки улаз обрађују независно од другог и не могу решавати проблеме као што су препознавање говора, јер би за њих било потребно анализирати читаву изговорену реч одједном. За такве проблеме користи се други тип мреже.

2. *Рекурентне мреже* - Код овог типа мрежа везе између неурона могу чинити циклусе што у претходној групи није било могуће. То значи да излаз неког неурона може проћи кроз неколико других неурона,а потом поново постати улаз истог. На тај начин доћи ће до мешања утицаја више различитих улаза истовремено, што је нама и потребно у неким случајевима као што су програми за препознавање говора. Више о овом типу неуралних мрежа касније.

3. Учење код неуралних мрежа

Учење је најбитнији детаљ на који посебно треба обратити пажњу приликом проучавања неуралних мрежа. Учење подразумева проналажење начина за такво постављање тежина и *bias*-а тако да добијемо оптимално решење.

Функција цене C је основни појам када говоримо о машинском учењу. Она претставља меру колико је тренутно решење удаљено од траженог оптималног решења. Једном када изаберемо добру функцију цене, наш задатак постаје да минимизујемо њену вредност. О овој функцији можемо да говоримо само када говоримо о *supervised learning*-у зато што је нама потребно да знамо тачно решење да би израчунали њену вредност.

3.1. Backpropagnation

Backpropagnation је метод који се најчешће користи код тренирања неуралних мрежа. Овај метод се најчешће користи у комбинацији са *gradient descent*-ом о коме ће бити више речи касније. Принцип на којем се заснива овај метод је да прво израчуна излазне вредности свих неурона. Затим, почев од излазног слоја па уназад, одређује грешке између добијених и жељених вредности, а на крају *update*-ује тежине да би добио мање грешке.

Уводимо следеће појмове да би лакше извели сав потребан рачун:

net_j^l - сума свих вредности које улазе у чвор j у l-том слоју (укључујући *bias*)

w_{ij}^l - тежина везе која спаја i-ти чвор l-1-ог слоја и j-ти l-тог слоја

out_i^l - вредност на излазу i-тог чвора l-тог слоја

t_j - очекивана вредност на j-том излазном неурону

Нека је функција цене коју бирамо $E = (1/2) * \sum_k (out_j - t_j)^2$, то је сума квадрираних грешака на свим излазним чворовима. Фактор 1/2 смо увели из техничких разлога да би олакшали каснији рачун. Желимо да израчунамо $\partial E / \partial w_{jk}^l$ и да ову вредност што је више могуће приближимо 0. Посматрамо два случаја: када је неурон члан излазног слоја и када је неурон члан скривеног слоја.

Израчунаћемо извод *sigmoid* функције зато што ћемо га користити у даљим израчунавањима:

$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z}\frac{1}{1+e^{-z}} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \cdot \frac{-e^{-z}}{1+e^{-z}} = \sigma(z) \cdot (1-\sigma(z))$$

3.2. Излазни слој

Приликом извођења рачуна у овом случају изостављамо суперскрипт, зато што је јасно да се ради о последњем, излазном слоју.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \cdot \frac{\partial out_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} \qquad (1)$$

$$\frac{\partial E}{\partial out_j} = -(t_j - out_j)$$

$$\frac{\partial out_j}{\partial net_j} = out_j(1 - out_j)$$

$$\frac{\partial net_j}{\partial w_{ij}} = out_i$$

Заменом у једначину (1)

$$\frac{\partial E}{\partial w_{ij}} = -(t_j - out_j) \cdot out_j \cdot (1 - out_j) \cdot out_i$$

Сада уведемо ознаку δ_j као

$\delta_j = -(t_j - out_j) \cdot out_j \cdot (1 - out_j)$, то јест $\delta_j = \frac{\partial E}{\partial out_j} \cdot \frac{\partial out_j}{\partial net_j}$

Да би смањили грешку морамо увести ново w_{ij}

$w_{ij} = w_{ij} - \eta \cdot \frac{\partial E}{\partial w_{ij}}$, где је η стопа учења.

Стопа учења је неки врло мали рационалан број (нпр. 0.01) који бирамо у зависности од ситуације. Уколико изаберемо већи број онда ће наша мрежа брже да учи, али недостатак бирања великог броја је то што тада не можемо добити толико прецизно решење колико би добили уколико би изабрали мали број. Увек треба наћи баланс између брзине учења и прецизности добијеног решења.

3.3. Скривени слој

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial out_j} \cdot \frac{\partial out_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} \qquad (2)$$

$$\frac{\partial net_j}{\partial w_{ij}} = out_i$$

$$\frac{\partial out_j}{\partial net_j} = out_j \cdot (1 - out_j)$$

$\frac{\partial E}{\partial out_j} = \sum_l (\frac{\partial E}{\partial net_l} \cdot \frac{\partial net_l}{\partial out_j})$, где *l* иде по свим неуронима у следећем слоју.

$$\sum_l (\frac{\partial E}{\partial net_l} \cdot \frac{\partial net_l}{\partial out_j}) = \sum_l (\frac{\partial E}{\partial out_l} \cdot \frac{\partial out_l}{\partial net_l} \cdot \frac{\partial net_l}{\partial out_j})$$

$$\frac{\partial net_l}{\partial out_j} = w_{jl}$$

Из претходних израчунавања знамо да важи: $\frac{\partial E}{\partial out_l} \cdot \frac{\partial out_l}{\partial net_l} = \delta_l$

Заменом свега у *(2)* добијамо:

$$\frac{\partial E}{\partial w_{ij}} = \sum_l (\delta_l \cdot w_{jl}) \cdot out_j \cdot (1 - out_j) \cdot out_i$$

Сада као и у претходном израчунавању уведимо следећу ознаку:

$\delta_j = \sum_l (\delta_l \cdot w_{jl}) \cdot out_j \cdot (1 - out_j)$, то јест $\delta_j = \frac{\partial E}{\partial out_j} \cdot \frac{\partial out_j}{\partial net_j}$

Затим израчунавамо нову вредност тежине везе:

$$w_{ij}^* = w_{ij} - \eta \cdot \frac{\partial E}{\partial wij}$$

Такође морамо мењати и вредност *bias*-а са циљем да добијемо боље решење:

$b_j^* = b_j - \eta \cdot \delta_j$, за *bias* било ког неурон у мрежи.

3.4. Gradient descent

До сада нисмо објаснили зашто метод за одређивање нових тежина или *bias*-а доноси оптимално решење. Овде приказан метод се зове спуст низ градијент. Он служи за проналажење локалног минимума функције. Ради лакшег разумевања како овај метод ради замислимо да је график функције некаква ливада на коју ви поставите лопту. Како ће се та лопта тада кретати? Она ће ићи низбрдо све док не дође до локалног минимума. Тако ради и овај алгоритам, помера вредност тежина за негативну вредност пропорциоалну градијенту у посматраној тачки (градијент је $\frac{\partial E}{\partial w_{ij}}$). Када би узимали кораке пропорционалне позитивној вредности градијента, тада би дошли до локалног максимума.

4. Примери програма који користе *feedforward* мреже

Позабавићемо се примерима два програма: за почетак једноставан пример програма који учи да симулира рад *and* капије, а затим пример сложенијег програма који користећи неуралне мреже и препознаје ручно писане цифре.

4.1. Опис мреже

Посматрајмо проблем у којем треба написати програм који ће да класификује ручно писане цифре у цифре од 0 до 9. За потребе решавања овог проблема претпоставићемо да је улаз *grayscale* слика димензија 28x28 што значи да сваки пиксел може узимати вредност од 0.0 , за белу боју, до 1.0, за црну боју, док су све вредности између одређене нијансе сиве.

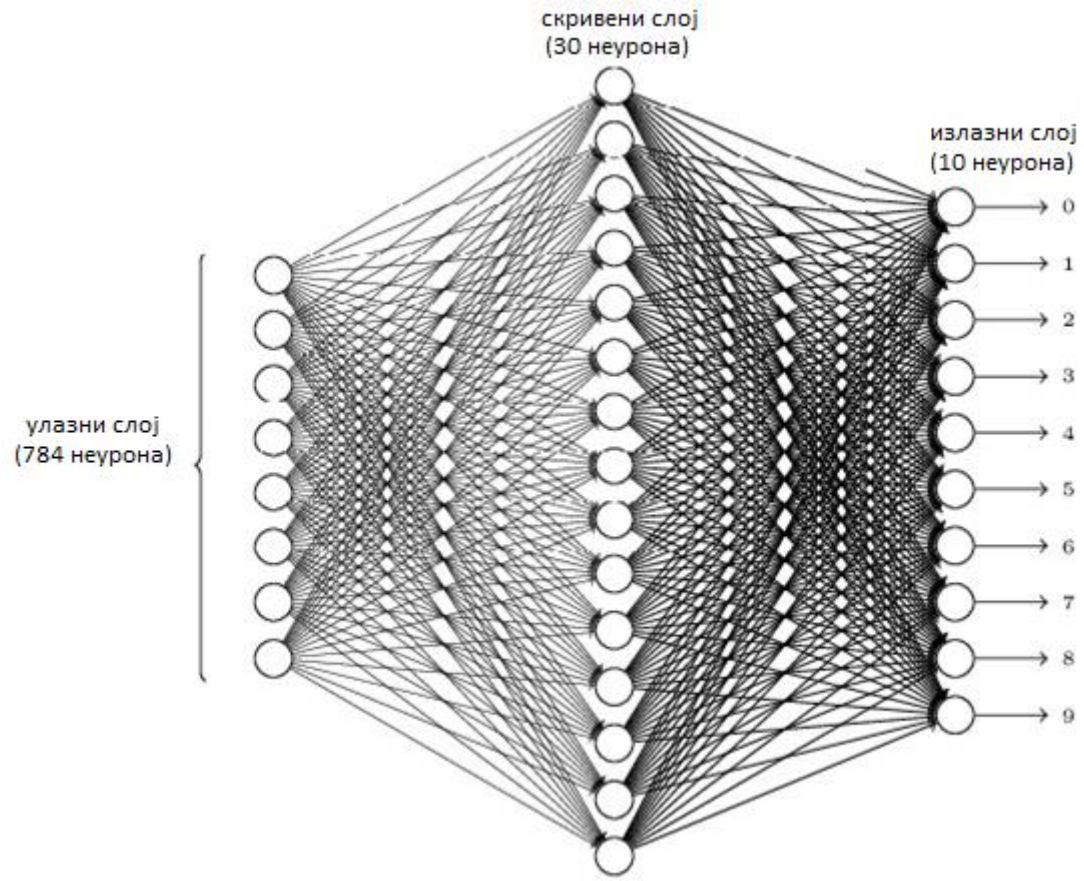
За решавање проблема користићемо неуралну мрежу која се састоји од три слоја. Улазни слој чиниће 28*28=784 неурона, то јест сваки неурон ће узимати вредост боје тачно једног пиксела. Излазни слој чиниће 10 неурона, сваки ће претстављати по једну цифру. Сада се поставља питање како ћемо помоћу десет излазних неурона одредити који број наша неуралне мреже приказује као решење, када ће се на сваком излазу наћи вредност између 0 и 1? Решење ће означавати онај неурон који на излазу даје највећу вредност. Сада нам остаје да дизајнирамо још скривени слој који је уједно и најтеже дизајнирати јер он не врши никакву интеракцију са спољним светом. Са бројем неурона у том слоју морамо мало експериментисати, али претпоставимо за сада да их има 30.

4.2. Тест узорци

Да би било могуће да користимо нашу неуралану мрежу морамо прво да је истренирамо, а за то нам је потребна база ручно писаних цифара. За сличне пројекте препознавања ручно писаних цифара најчешће је коришћена базе података MNIST[1]-а.

Она садржи велики број слика димензија 28*28 пиксела које су писале различите особе. По узору на њу и ми коритимо 5000 цифара које је писало 100 особа. Цифре ћемо поделити у две групе: 4000 на којима ће мрежа учити и 1000 на којима ће се тестирати њен рад.

слика 6: Примери цифара које користимо



слика 7: Илустрација претходно описане мреже

[1] MNIST је скраћено од Modified National Institute of Standards and Technology, то је национални институт за стандарде и технологију Сједињених Америчких Држава.

4.3. Програм за препознавање ручно писаних цифара

Програм ћемо имплементирати у програмском језику Java зато што је он погодан за рад са сликом. У раду ће бити изложене само најбитније функције док ће се комплетан код налазити у прилогу. Делови кода који су под коментаром представљају рад са *bias* вредностима зато што се у пракси неурална мрежа може имплементирати и без коришћења *bias*-а

За почетак обратимо пажњу на функцију која креира неуралну мрежу као почетну фазу извршавања програма. Позивањем ове функције на случајан начин генеришемо иницијалне вредности тежина веза и *bias*-а.

```
public mreza()
{
    Random rand= new Random();
    s1=784;  //broja neurona u svakom od slojeva
    s2=30;
    s3=10;
    for (i=1;i<=s1;i++)
    {
        for (j=1;j<=s2;j++)
        {
            w[2][i][j]=rand.nextDouble();  //slučajno generišemo težine veza između ulaznog
            w[2][i][j]=w[2][i][j]*0.01;    //i skrivenog sloja
        }
    }
    for (i=1;i<=s2;i++)
    {
        for (j=1;j<=s3;j++)
        {
            w[3][i][j]=rand.nextDouble();  //slučajno generišemo težine veza između skrivenog
            w[3][i][j]=w[3][i][j]*0.01;    //i izlaznog sloja
        }
    }
    /*for (i=1;i<=30;i++)
    {
        b[2][i]=rand.nextDouble();
        b[2][i]=b[2][i]*0.0001;
    }
    for (i=1;i<=10;i++)
    {
        b[3][i]=rand.nextDouble();
        b[3][i]=b[3][i]*0.0001;
    }*/
}
```

Функција коју користимо да случајно генеришемо тежину веза *rand.nextDouble()* враћа случајно изабран рационалан број између 0 и 1. Приметимо да сваку везу множимо са 0.01. То радимо зато што уколико би тежине веза биле веће онда би улазна вредност за неуроне у скривеном слоју била релативно велика па би $\mathbf{e}^{-\text{улаз}}$ било запамћено као нула па би *sigmoid* функција увек давала вредност 1 што није довољно прецизно.

Сада се позабавимо функцијом *feedforward* која за унете вредности на улазни слој израчунава вредности на излазном слоју мреже.

```
public void feedforward ()
{
    for (j=1;j<=30;j++)
    {
        net[2][j]=0;     //suma svih ulaza u neuron skrivenog sloja inicijalno postavljena na 0
        for (i=1;i<=784;i++)
        {
            net[2][j]=net[2][j]+w[2][i][j]*out[1][i];  //sumiranje svih ulaznih neurona
        }
        //net[2][j]=net[2][j]+b[2][j];
        out[2][j]=sig(net[2][j]);       //izračunavanje izlazne vrednosti pomoću sigmoid funkcije
    }
    for (j=1;j<=10;j++)
    {
        net[3][j]=0;   //suma svih ulaza u neuron izlaznog sloja inicijalno postavljena na 0
        for (i=1;i<=30;i++)
        {
            net[3][j]=net[3][j]+w[3][i][j]*out[2][i]; //sumiranje svih vrednosti dobijenih iz skrivenog sloja
        }
        //net[3][j]=net[3][j]+b[3][j];
        out[3][j]=sig(net[3][j]);    //izračunavanje izlazne vrednosti pomoću sigmoid funkcije
    }
}
```

У њој израчунавамо излазне вредности свих неурона.

За крај остаје функција која врши *backpropagnation* и рачуна нове вредности тежине веза и *bias*-а.

```
public void backpropagnation ()
{
    e=0;      //ukupna vrednost kvadrata odstupanja
    lr=0.05;  //stopa učenja
    for (i=1;i<=10;i++)
    {
        e=e+(t[i]-out[3][i])*(t[i]-out[3][i]);
    }
    e=e*0.5;
    for (i=1;i<=10;i++)
    {
        dt[3][i]=-(t[i]-out[3][i])*out[3][i]*(1-out[3][i]);   //izračunavamo vrednost delta za neurone u izlaznom sloju
    }
    for (i=1;i<=30;i++)
    {
        dt[2][i]=0;
        for (j=1;j<=10;j++)
        {
            dt[2][i]=dt[2][i]+dt[3][j]*w[3][i][j];    //izračunavamo vrednost delta za neurone u skrivenom sloju
        }
        dt[2][i]=dt[2][i]*out[2][i]*(1-out[2][i]);
    }
    for (i=1;i<=784;i++)
    {
        for (j=1;j<=30;j++)
        {
            w[2][i][j]=w[2][i][j]-lr*dt[2][j]*out[1][i];   //update težina veza koje povezuju ulazni i skriveni sloj
        }
    }
    /*for (j=1;j<=30;j++)
    {
        b[2][j]=b[2][j]-lr*dt[2][j];
    }*/
    for (i=1;i<=30;i++)
    {
        for (j=1;j<=10;j++)
        {
            w[3][i][j]=w[3][i][j]-lr*dt[3][j]*out[2][i];  //upadte težina veza koje povezuju skriveni i izlazni sloj
        }
    }
    /*for (j=1;j<=10;j++)
    {
        b[3][j]=b[3][j]-lr*dt[3][j];
    }*/
}
```

У овој функцији на раније описан начин мењамо вредности веза и *bias*-а и тако постижемо све бољи и бољи резултат.

4.4. Тестирање рада неуралне мреже

Сада треба проверити да ли се овако компликован проблем као што је препознавање ручно писаних цифара, може довољно добро решити помоћу нешто више од 100 линија кода.

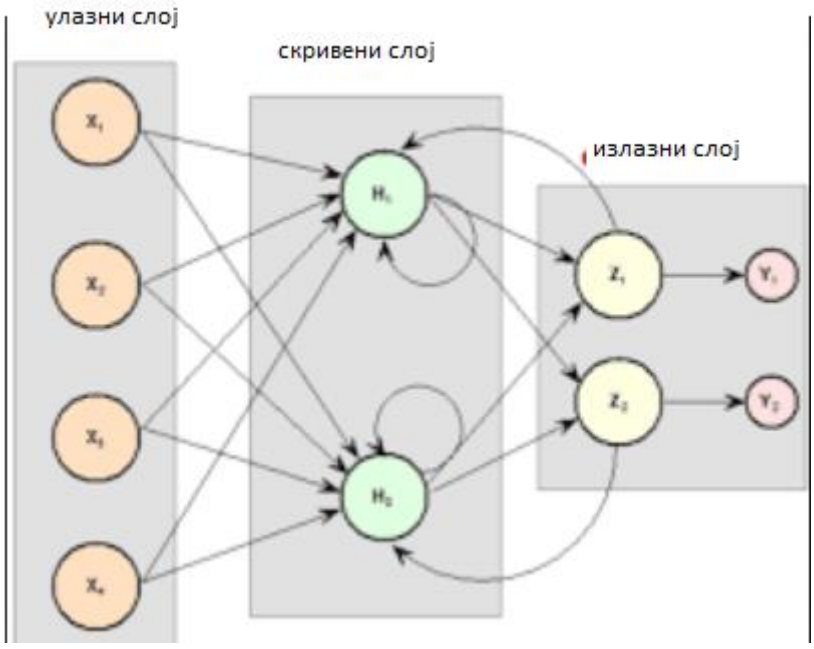
За почетак, проверимо да ли мрежа уопште ради на нешто једноставнијем примеру. Тренирајмо је на мањем узорку од 3000 слика и проверимо да ли наша мрежа постиже неки напредак на препознавању слика које смо јој већ показали. Након само 300 итерација учења ових слика (око 12 минута), мрежа са тачношћу препознаје 2653 цифре што је приближно 88.5%. Након још 300 итерација препознаје 2806 цифара, тј. има тачност око 93.5%, а након још 300 итерција, 2829 цифара, што је око 94.3%.

Како уочавамо да за изузетно кратко време (мало више од пола сата) програм научи да препознаје познате цифре са довољно великом прецизношћу, треба се позабавити правим проблемом. Тренирајмо програм на 4000 цифара и проверимо колику прецизност постиже на 1000 непознатих примера. За пола сата учења програм са тачношћу препознаје 605 цифара За око сат времена програм препознаје 621 цифру што је 62.1%, а за два сата 632 – 63.2%.

На основу овог примера можемо утврдити да неуралне мреже чине компијутер способним да учи можда и брже него сам човек. С обзиром на то да се користимо релативно малом базом података и да је трненинг био релативно кратак програм достиже задовољавајуће резултате.

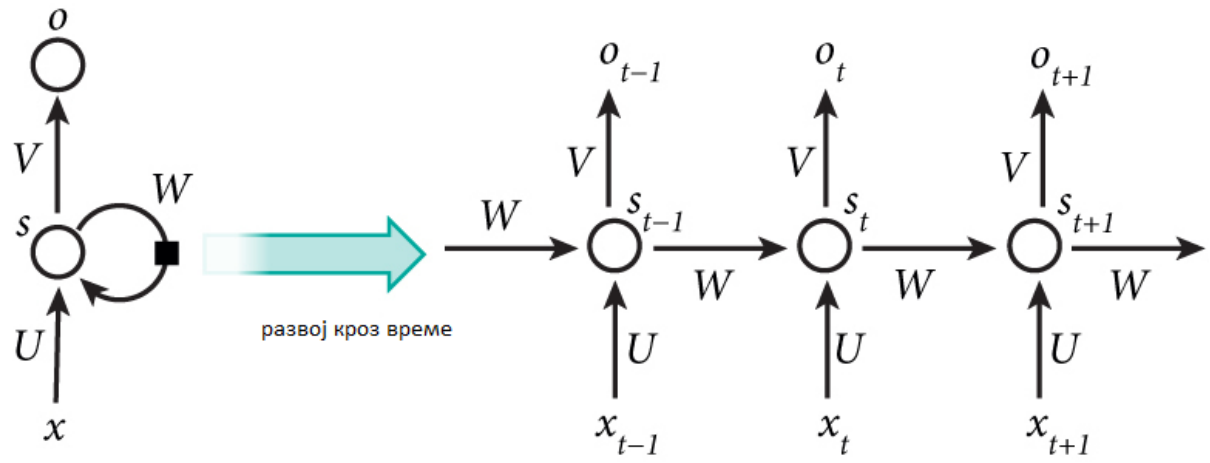
5. Рекурентне неуралне мреже

Рекурентне неуралне мреже су посебна врста неуралних мрежа која за разлику од претходне врсте не прослеђује информације из једног слоја искључиво у наредни већ у себи може садржати циклусе. То је врло битна карактеристика захвљујући којој је овај тип мрежа нашао много ширу примену него *feedforward* мреже. До сада смо улазе могли да обрађујемо само један по један, независно. Сваки улаз би ишао редом слој по слој и не би било никаквог мешања са другим улазима. У рекурентним неуралним мрежама излаз из неког неурона може ући у неки циклус и касније опет постати улаз истог тог неурона и тако се мешати са неким каснијим улазима. Узмимо за пример да желимо да направимо мрежу која треба да преводи реченице. Потребно је да за сваку реч одредимо могућа значења и онда у комбинацији са остатком реченице одредимо контекст посматране речи. Уколико би покушали да користимо *feedforward* мрежу она би као улаз у једном тренутку узимала само једну реч. Не би могла да памти и комбинује са остатком реченице јер она нема могућност памћења. За овакве проблеме користе се рекурентне мреже јер оне уз помоћ петљи увек задржавају део информације у себи. Овакве мреже нашле су значајну примену у области препознавања говора и превода текстова.



слика 8: Пример рекурентне мреже

Често ради лакшег праћења догађаја у рекурентним мрежама ми посматрену рекурентну мрежу 'развијемо кроз време' и тако добијамо нешто слично познатим *feedforward* мрежама само што се сваки блок одвија у једној јединици времена.



слика 9: Пример развоја мреже кроз време

5.1. Проблем нестајућег градијента

Да би разумели проблем нестајућег градијента прво треба разумети како уче рекуренте неуралне мреже. Јасно је да није могуће применити *backpropagation* зато што нећемо моћи да одређујемо δ слој по слој јер постоје циклуси, тако да ћемо поменути алгоритам мало модификовати. На већ поменут начин развићемо мрежу кроз време и на тај начин добићемо класичну *feedforward* мрежу и на њој је могуће применити *backpropagation*. Овај алгоритам се назива *backpropagnation through time*.

Проблем на који су научници наилазили у првим покушајима тренирања рекурентних неуралних мрежа је проблем нестајућег градијента. То се догађало када се у *backpropagnation-у* користио управо метод спуштања по градијенту који смо и ми користили. До проблема је долазило зато што је вредност извода која се користила за тренирање расла екпоненцијално са бројем слојева у мрежи. То значи да је до проблема долазило не само код рекурентних мрежа било које величине (јер се код

њих на један чвор због присуства циклуса враћамо много пута), већ и код веома дубоких неуралних мрежа.

Посматрајмо једноставну рекурентну неуралну мрежу задату на следећи начин:

$h_t = w * \sigma(h_{t-1}) + w_x * x_t$ и $y_t = w_y * \sigma(h_t)$, x_t је улазна вредност у мрежу у тренутку t, h_t вредност на улазу у скривени слој у посматраном тренутку и y_t вредност на излазу.

Кренимо да рачунамо потребне изводе:

$\frac{\partial E}{\partial w} = \sum_{t=1}^{s} \frac{\partial E_t}{\partial w}$, где је s број који представља са колико смо корака развили мрежу.

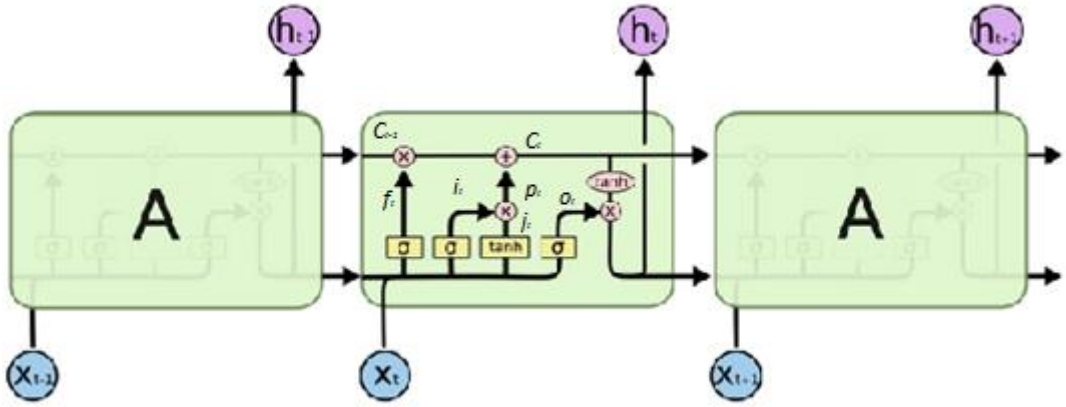
$$\frac{\partial E_t}{\partial w} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} * \frac{\partial y_t}{\partial h_t} * \frac{\partial h_t}{\partial h_k} * \frac{\partial h_k}{\partial w}$$

$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} w * \sigma(h_{i-1}) * (1 - \sigma(h_{i-1}))$, што ће за велику разлику између t и i и уз чињеницу да су чиниоци мањи од 1, тежити 0. Уколико би посматрали неку другу функцију, а не *sigmoid*, могло би се десити да су чиниоци већи од нуле и бројеви ће тада постати много велики што се зове *проблем експлодирајућег градијента*.

5.2. Long short-term memory (LSTM)

Long short-term memory је нови тип рекурентних неуралних мрежа који су 1997. дизајнирали *Sepp Hochreiter* и *Jürgen Schmidhuber*. Овај тип мреже је дизајниран тако да нема проблем са нестајућим градијентом, па је тренутно најпопуларнији тип неуралне мреже. Чак је победио у такмичењу у препознавању рукописа одржаном 2009. године.

Слично као и обичне рекурентне мреже, *LSTM* могу да се развију и тако добијамо низ копија једног модула. Кључна разлика је у томе што ће функција активације бити значајно сложенија од једноставне *sigmoid* функције.



слика 10: Пример који се састоји од три модула *LSTM*-а

Кључ за разузмевање *LSTM*-а је стање ћелије, а њега на слици изнад представља линија која непрекинута пролази кроз врх свих модула и тако преноси одређену информацију. *LSTM* има начин да ту информацију мења, дода нове информације или је комплетно избрише. Утицај на ову информацију се врши на три места приликом протицања кроз један модул, а та места се називају капије.

Капија заборава- Прво треба одредити шта ћемо избрисати из стања ћелије. То се одрађује преко *sigmoid* неурона. Он као свој излаз даје вредност између 0 и 1 која се потом множи са стањем ћелије. Јасно је да уколико добијемо 1 то значи да задржавамо све, а уколико добијемо 0 онда комплетно бришемо досадашње стање ћелије.

$$f_t = \sigma(w_f * (h_{t-1} * x_t) + b_f$$

Улазна капија- Следећа на реду је улазна капија. Она служи за додавање нових информација у стање ћелије. Она се користи тако што на стање ћелије додајемо производ *sigmoid* функције и функције хиперболичког тангенса.

$$i_t = \sigma(w_i * (h_{t-1} + x_t) + b_i)$$

$$j_t = \tanh(w_j * (h_{t-1} + x_t) + b_j)$$

$$C_t = C_{t-1} * f_t + i_t * j_t$$

На крају нам преостаје само да одредимо која вредност ће бити излаз овог модула. Њу добијамо тако што помножимо вредност хиперболичког тангенса који за аргумент узима стање ћелије и *sigmoid* функције која за аргумент има излаз из претходног модула.

$$o_t = \sigma(w_o * (h_{t-1} + x_t) + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

6. Закључак

У овом раду је укратко приказан значај машинског учења, а посебно неуралних мрежа у решавању неких проблема. Увиђамо да нам за њихово коришћење нису потребни комплексни кодови већ само добро дизајнирана мрежа која ће посао радити уместо вас. Надам се да сам у предходних тридесетак страна успео да објасним основе рада са неуралним мрежама и да је програм, чији се комплетан код може наћи у прилогу, помогао у њиховом разумевању.

Желео бих да се посебно захвалим мом ментору Јелени Хаџи-Пурић, која је била мој професор информатике током читавог школовања у Математичкој гимназији, без које овај рад не би био оно што јесте. Такође бих желео да се захвалим и професорки Станки Матковић, такође професорки информатике, која је посебно допринела мом знању програмског језика Java.

Литература

1. Introduction to machine learning- Alex Smola,
   http://alex.smola.org/drafts/thebook.pdf, 21.05.2016.

2. Using neural nets to recognize handwritten digits,
   neuralnetworksanddeeplearning.com/chap1.html, 21.05.2016.

3. A step by step Backpropagation Example,
   http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/, 21.05.2016.

4. Machine learning,
   https://en.wikipedia.org/wiki/Machine_learning, 22.05.2016.

5. Artificial neural network,
   https://en.wikipedia.org/wiki/Artificial_neural_network, 22.05.2016.

6. Backpropagation,
   https://en.wikipedia.org/wiki/Backpropagation, 22.05.2016.

7. Gradient descent,
   https://en.wikipedia.org/wiki/Gradient_descent, 22.05.2016.

8. Recurrent neural network,
   https://en.wikipedia.org/wiki/Recurrent_neural_network, 23.05.2016.

9. Vanishing gradient problem,
   https://en.wikipedia.org/wiki/Vanishing_gradient_problem, 23.05.2016.

10. Long short-term memory,
    https://en.wikipedia.org/wiki/Long_short-term_memory, 23.05.2016.

11. Recurrent Neural Nets and LSTMs, Department of Computer Science- University of Oxford,
    https://www.youtube.com/watch?v=56TYLaQN4N8, 23.05.2016

12. Understanding LSTM Networks,
    http://colah.github.io/posts/2015-08-Understanding-LSTMs, 23.05.2016.