

**ВИСОКА ТЕХНИЧКА ШКОЛА СТРУКОВНИХ  
СТУДИЈА У КРАГУЈЕВЦУ**

**ПРЕПОЗНАВАЊЕ РУЧНО ПИСАНИХ ЦИФАРА  
ПОМОЋУ НЕУРОНСКИХ МРЕЖА**

**завршни рад**

**Студент  
Филип Лазовић  
094/2014**

**Ментор  
Др. Владимир Недић**

**Крагујевац  
Август, 2020**

# Табела садржаја

<b>1. Машино учење</b>	<b>3</b>
1.1. Машина Потпорних Вектора	4
1.2. К Најближих Комшија	4
1.3. Стабло одлуке	5
1.4. Линеарна и Полиномска регресија	6
<b>2. Невронске мреже</b>	<b>6</b>
2.1. Организација	7
2.2. Хипер Параметри	8
2.3. Учење	8
2.4. Стопа учења	8
2.5. Сума конекција	8
2.6. Активационе функције	8
2.6.1. Сигмоид	9
2.6.2. Софтмакс	10
2.6.3. РЕЛУ	10
2.7. Функција грешке	10
2.8. Регуларизација	11
2.9. Одбацивање неурона	11
2.10. Нормализација	11
2.11. Повратна пропагација	11
2.12. Спуштање градијентом	16
2.13. Пример учења	17
<b>3. Конволуције</b>	<b>18</b>
3.1. Филтери	19
3.2. Удруживање	20
3.3. Потпуно конектовани слој	21
3.4. Учење филтера	22
<b>4. Проблем</b>	<b>24</b>
4.1. Познати модели	28
4.1.1. ЛеНет (енг. LeNet)	28
4.1.2. АлексНет (енг. AlexNet)	28
4.1.3. ВГГ (енг. VGG)	29
4.2. Познате библиотеке	30
4.2.1. Тензорфлуу (енг. Tensorflow)	30
4.2.2. ПајТорч (енг. PyTorch)	30
4.3. Коришћење у продукцији	30
4.4. Закључак	31
<b>5. Ендноте</b>	<b>33</b>

# 1. Машино учење

Машинско учење (енг. Machine Learning) (у даљем тексту као МУ) је дисциплина која проучава алгоритме које уче на основу датих података. МУ алгоритми се користе за решавање комплексних проблема, који не могу бити решени методама заснованим на ручно програмираним логичким гранањима. Многи проблеми могу бити решени на тај начин, као нпр. калкулатор. Али, постоје проблеми чија решења немају јасно дефинисана правила, као нпр. да препознамо број са слике. Можемо да направимо правила на основу вредности пиксела са слике, али то би захтевало много времена. Такође не можемо да гарантујемо да ће програм дати тачан резултат када се вредности пиксела промене за исти број. Насупрот овом решењу, направићемо велику базу података са примерима и програм који ће да открије која цифра је написана на слици, аутоматски.<sup>1</sup>

Први програм који је имао могућност да учи је настао 1950-их година. Артур Самуел (енг. Arthur Samuel) је написао програм који ће да научи „игра даме”<sup>2</sup>. Користио је минимакс алгоритам (енг. Minimax) да израчуна који потез је следећи и био је први човек који ће да користи термин машинско учење. 1952 године, Франк Роузенблат (енг. Frank Rosenblatt) је направио перцептрон (енг. Perceptron) машину<sup>3</sup>, која је имала задатак да препозна визуелне шаблоне као што су људске лица. Резултати су били разочарајући што је довело до застоја у развоју машинског учења све до 1990-их година. Комплексни задаци као што је препознавање људског лица са слика постаје могуће након деведесетих година из следећих разлога:

- Могућност рачунара да извршавају захтевније задатке;
- Нови алгоритми;
- Веће базе података које се користе за тренирање;

Након ових промена МУ алгоритми показују велики потенцијал у стварању интелигентне машине, а нарочито под група алгоритама звана дубоко учење (енг. Deep Learning). Развојем графичких картица налази се начин да се убрза тренирање ових алгоритама користећи паралелно рачунање. Развојем интернета количина података за тренирање је драстично порасла. 2011 године проф. Феи-Феи Ли (енг. Fei-Fei Li) прави ИмиџНет (енг. ImageNet) базу података, који је инспирисала ИЛСВРЦ<sup>4</sup> (енг. ImageNet Large Scale Visual Recognition Challange-ILSVRC) такмичење које је изазвало револуцију у области МУ.

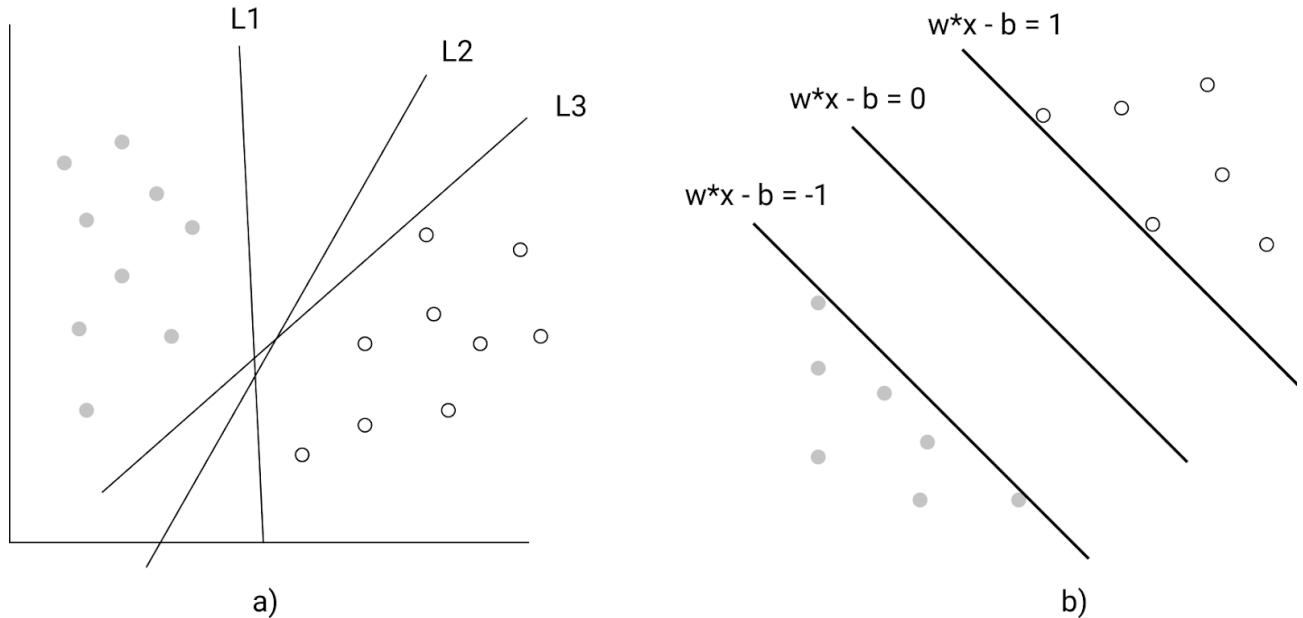
Постоје два начина учења: супервизовано и несупервизовано. Супервизовано учење захтева податке и ознаку. Ознака је уствари коначан резултат који желимо да предвидимо. Несупервизовано учење не захтева ознаку, и модел сам налази шаблоне.

МУ алгоритни се користе за решавање две врсте проблема: класификације и регресије. Код класификације постоје висе ознаке података које се називају класе. Код регресије покусавамо да нађемо тренд или везу између улазних и излазних параметара.

Алгоритми за класификацију су: Машина потпорних вектора (енг. Support Vector Machine-SVM), к најближих комшија (енг. K Nearest Neighbor), Стабло одлике (енг. Decision Tree).

## 1.1. Машина Потпорних Вектора

Позната је и као метода максимално граничне хиперравни, јер се тражи хиперраван која раздваја хиперпростор у два дела која одговарају двема категоријама (може се проширити на више категорија) тако да је удаљеност до најближих тачака максимална. Хиперраван је одређена вектором  $w$  и једначином  $w^*x + b = 0$ . Подаци морају бити линеарно сепарабилни, што значи да се могу поделити линеарном равни. Уколико нису, врши се трансформација података.<sup>5</sup>



Слика 1. а) Приказ неколико кандидата за линију која најврлоје раздваја податке. б) изглед функције која најбоље раздваја приказане податке у дво-димензијалном простору.

Ако имамо скуп примера  $\{(x_1, c_1), \dots, (x_n, c_n)\}$  где су  $c_i \in \{-1, 1\}$  вредност  $x_i$  је пожељно нормализовати између -1 и 1, и задатак је да се минимизује  $w$ , за шта се користи алгоритам опадајућег градијента.

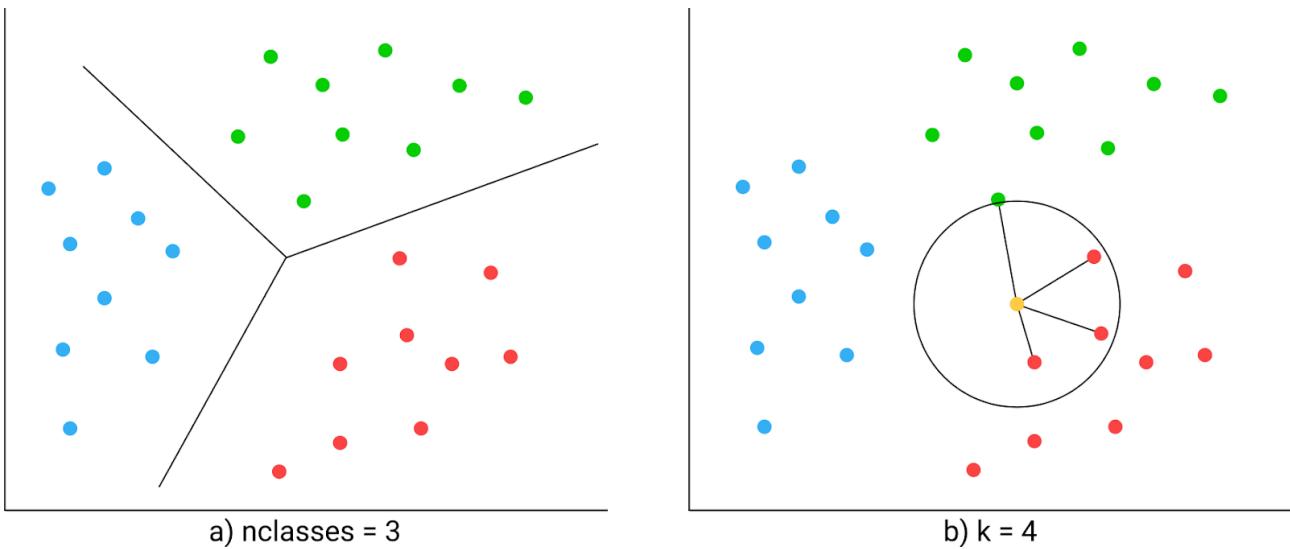
## 1.2. К Најближих Комшија

К најближих комшија је један од најједноставнијих МУ алгоритама. Класификација се може вршити на означеним или неозначеним подацима. У случају неозначених података потребна је кластеризација помоћу алгоритма као што је метода к-средњих вредности. Класификација се врши мерењем дистанце између нове вредности и свих осталих података.<sup>6</sup> Најуобичајнија мера дистанце је еуклидијска (енг. Euclidean) раздаљина.<sup>7</sup>

Ако су дате тачке  $x$  и  $y$  са димензијама  $N$ , дистанца  $d$  је:

$$d(x, y) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}$$

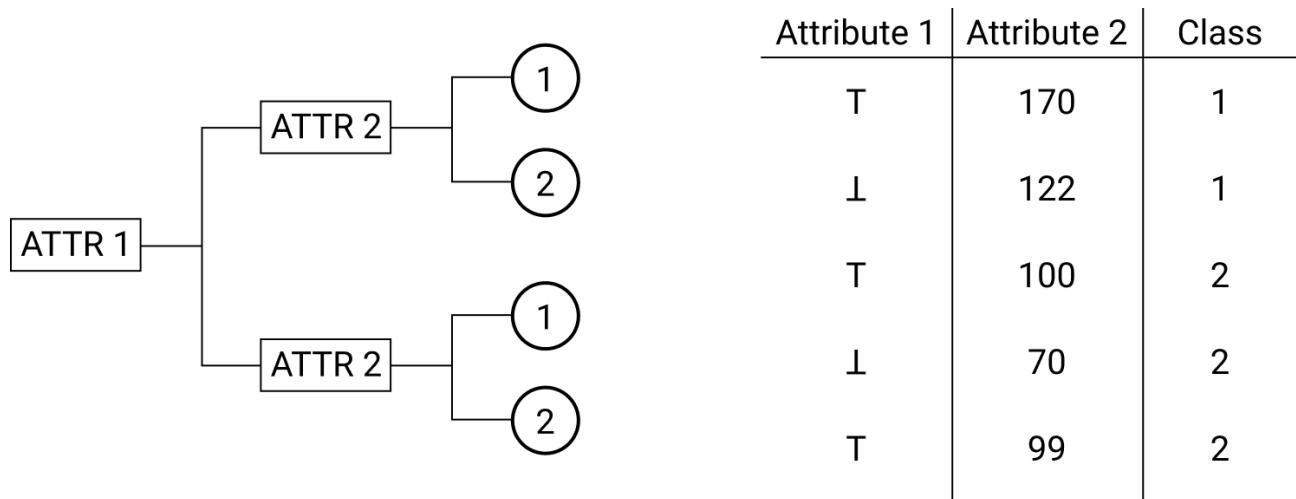
$k$  је број најближих тачака које ћемо узети у обзир када одређујемо класу. нпр. када је  $k = 4$  узећемо 4 најближе тачке. Класа са највећим бројем тачака је коначни резултат.



Слика 2. а) Приказ података и њихових класа. б) Пример налажења класе за тачку обожену жутом бојом.

### 1.3. Стабло одлуке

Стабло одлуке је хијерархиска структура података заснована на одлукама.<sup>8</sup> Унутрашњи чворови представљају "тест" атрибута, свака грана представља резултат теста, и сваки лист представља класу.



Слика 3. Визуелни приказ стабла који је направљен на основу података у табели.

Код бинарних атрибута (тачно или нетачно), унутрашњи чворови се праве на основу цини методе (енг. Gini Impurity)<sup>9</sup>:

$$g(x_n, y_n) = 1 - \left( \frac{x_n}{x_n + y_n} \right)^2 - \left( \frac{y_n}{x_n + y_n} \right)^2$$

где је  $x_n$  број тачних, а  $y_n$  број нетачних атрибута. Што је цини вредност мања за дати атрибут, то је подела подата боља.

Код атрибута представљеним са бројевима, потребно је одредити добру вредност која ће поделити податке. Речимо да је  $a_i$  атрибут на локацији  $i$  која почиње од 1:

$$d_i = \frac{a_i + a_{i-1}}{2}$$

$$x_n = \left| \{n \mid n > d_i\} \right|$$

$$y_n = \left| \{n \mid n \leq d_i\} \right|$$

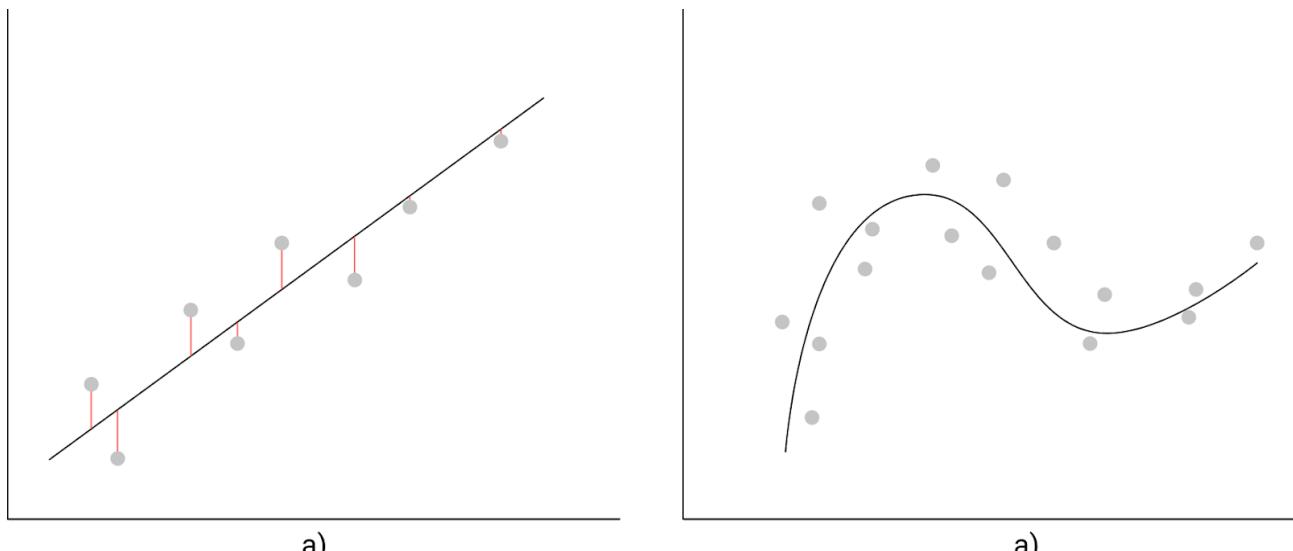
## 1.4. Линеарна и Полиномска регресија

Регресија је процес налажења везе између улазних и излазних података. Ако имамо податке  $\{(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)\}$  тражи се веза између  $x$  и  $y$ . Ако је веза линеарна<sup>10</sup>:

$$y_i = w * x_i + b$$

У колико се веза не може описати правом линијом, користи се полиномска регресија.<sup>11</sup> Нелинеарност се уводи довођењем  $n$ -тог степена променљиве  $x$ .

$$y = b + w_1 x + w_2 x^2 + w_n x^n$$



Слика 4. а) Оптимална линеарна функција која описује дате податке. б) полиномска функција која описује дате податке.

## 2. Неуронске мреже

Неуронске мреже (енг. Neural Network) су инспирисане радом биолошких неуронских мрежа. Састоје се од неурона, који задржавају биолошки концепт, што значи да примају улазни сигнал, комбинују тај сигнал са унутрашњим стањем и производе излазни сигнал. Улазни сигнални су слике или документи. Излаз извршава неки задатак (класификације или регресије).<sup>12</sup>

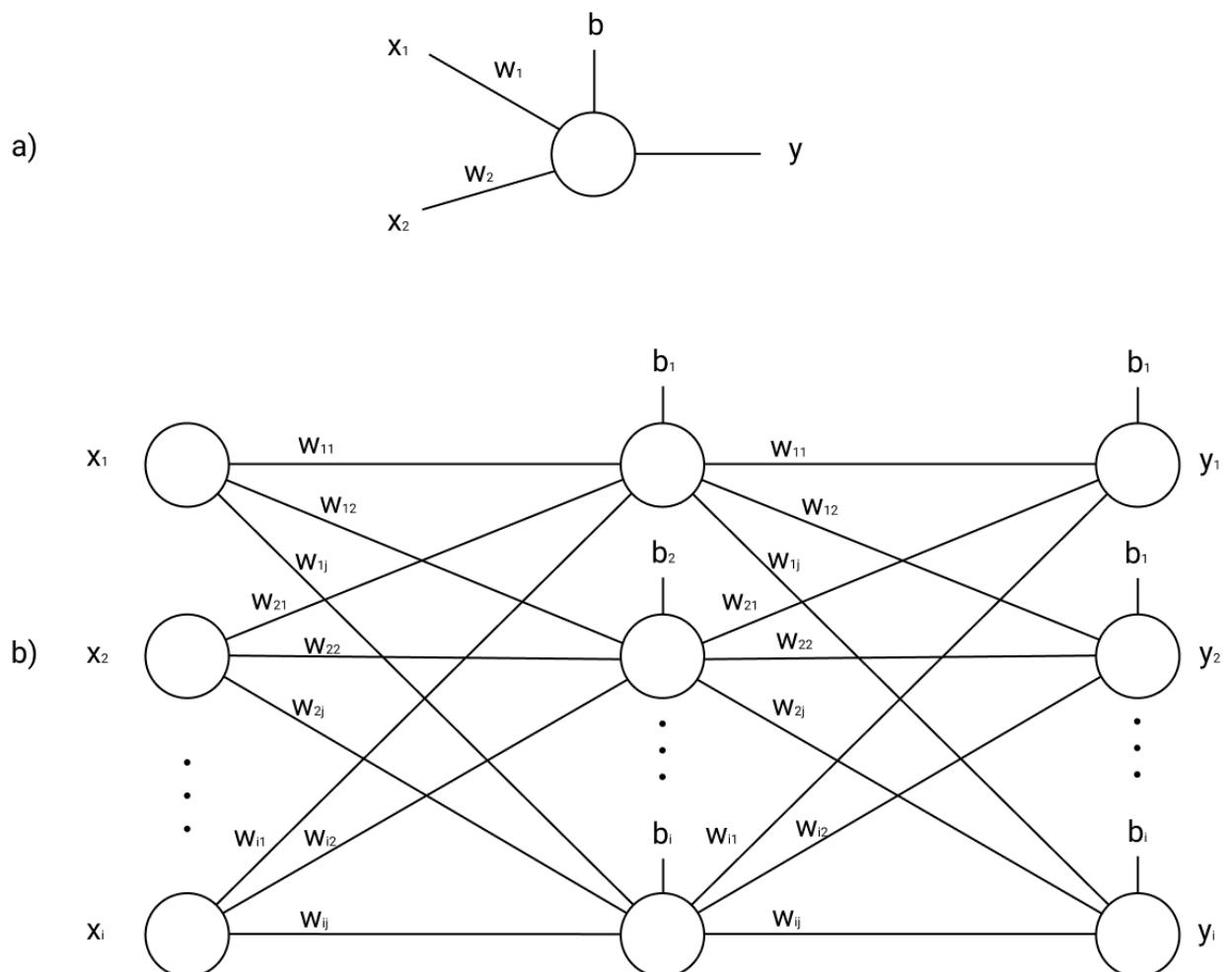
Пошто се перцептрон (енг. Perceptron) није добро показао као проналазач комплексних шаблона, друге методе МУ-а су биле коришћене све до појаве методе повратног ширења<sup>13</sup> 1975-е, што омогућава прављење вишеслојних неуронских мрежа. Развојем све бољих

графичких картица, омогућено је прављење дубоких неуронских мрежа<sup>14</sup> (са више слојева) чиме почиње револуција у области МУ-а.

Мрежа се садржи од конекција, свака конекција повезује излаз једног неурона са улазом другог. Свака конекција има тежину (енг. Weight), која представља њен утицај на коначни резултат. Неурони могу имати више улазних и излазних конекција. Сумом конекција се рачуна јачина улазног сигнала неурона на основу излаза предходног излаза.

## 2.1. Организација

Модели неуронских мрежа обично имају више слојева. Неурони једног слоја могу бити повезани са неуронима из предходног или следећег слоја. Слој који прима податке се назива улазни слој. Слој који производи излазни резултат се назива излазни слој. Слојеви између се називају скривени слојеви (енг. Hidden Layers). Постоје и модели без скривених слојева. Слојеве је могуће повезати на неколико начина. Постоје потпуно повезани, где су неурони једног слоја повезани са свим неуронима следећег слоја, слој за обједињавање, где се група неурона из једног слоја повезује са једним неуроном следећег слоја. Мреже у којима везе између слојева иду у једном смеру (од улаза ка излазу) називају се неуронске мреже са напредном спрегом (енг. Feedforward). Мреже у којима су неурони повезани са предходним слојевима, називају се повратне неуронске мреже.<sup>15</sup> Модел са само једним неуроном се назива перцептрон.



Слика 5. а) Перцептрон. б) Неуронска мрежа.

## 2.2. Хипер Параметри

Хипер параметар је константа која се иницијализује пре процеса тренирања. Примери хипер параметара су: стопа учења<sup>16</sup> (енг. Learning rate), тип и број скривених слијева, број епoha<sup>17</sup> (енг. Epochs). Програмер је тај који одлучује вредност параметара, али постоје и алгоритми који сами уче оптималне вердности за параметре. Та метода се назива мета учење<sup>18</sup> (енг. Meta Learning).

## 2.3. Учење

Учење је адаптација мреже да одради неки задатак. Подразумева подешавање јачине конекција између неурона са циљем побољшања прецизности резултата.<sup>19</sup> Ово се постиже минимизовањем грешака. Учење је завршено када се грешка више не може знатно смањити. После учења, грешка обично не достиже до нуле. Ако је грешка остала велика, то значи да се модел не може адаптирати на улазне податке. То се решава мењањем архитектуре нашег модела.

## 2.4. Стопа учења

Стопа учења дефинише величину корака који модел направи у циљу смањивања грешке. Велика стопа учења смањује време тренирања, али смањује и прецизност модела, то јест велика је шанса да модел не дође до стања у коме је грешка најмања. Мала стопа учења повећава вероватноћу да ће модел доћи у стање најмање грешке, али је тренирање спорије. То се решава тако што се на почетку постави велика стопа учења да би модел испрскакао локалне минимуме и нађе глобални минимум, затим се постепено смањује док се не дође до апсолутног минимума.

## 2.5. Сума конекција

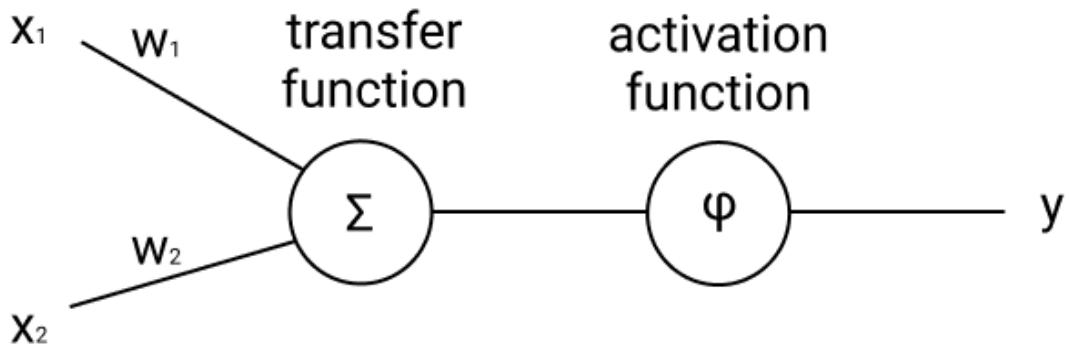
Улаз у сваки чвр је резултат суме конекција, што је уствари suma свих излаза из предходног слоја којима је додељена посебна тежина. Циљ ових тежина је да се пронађе шаблон или карактеристика у улазним подацима. Ако је suma велика, то значи да је карактеристика присутна, у супротном улазни подаци не садрже шаблон који овај чбор тражи. Ако имамо  $n$  конекција из слоја  $l-1$  у слој  $l$  који улазе у чвр  $j$ , где је тежина обележена као  $w$  и активација као  $a$  онда је suma:

$$z_j^{(l)} = \sum_{k=0}^n w_{kj}^{(l)} a_k^{(l-1)}$$

## 2.6. Активационе функције

Излаз сваког чвора је резултат активационе функције<sup>20</sup> (енг. Activation Function)  $\phi$  која за улаз има резултат суме конекција. Из предходног слоја добијамо суму линеарних функција која се прослеђује активационој функцији. Да би апроксимирали комплексне функције, потребне су нелинеарне активационе функције.

Активационе функције дају моделу две нове карактеристике: нелинеарност (што омогућава моделу да буде универзални апроксиматор функција) и ограниченост (ограничавају излаз чвора са доњом или горњом границом).



Слика 6. Приказ тока података. Сума конекција пролази кроз активациону функцију.

Најпознатије активационе функције су:

### 2.6.1. Сигмоид

Сигмоид (енг. Sigmoid) спада у групу логистичких функција. Има облик слова *S*, са једначином:

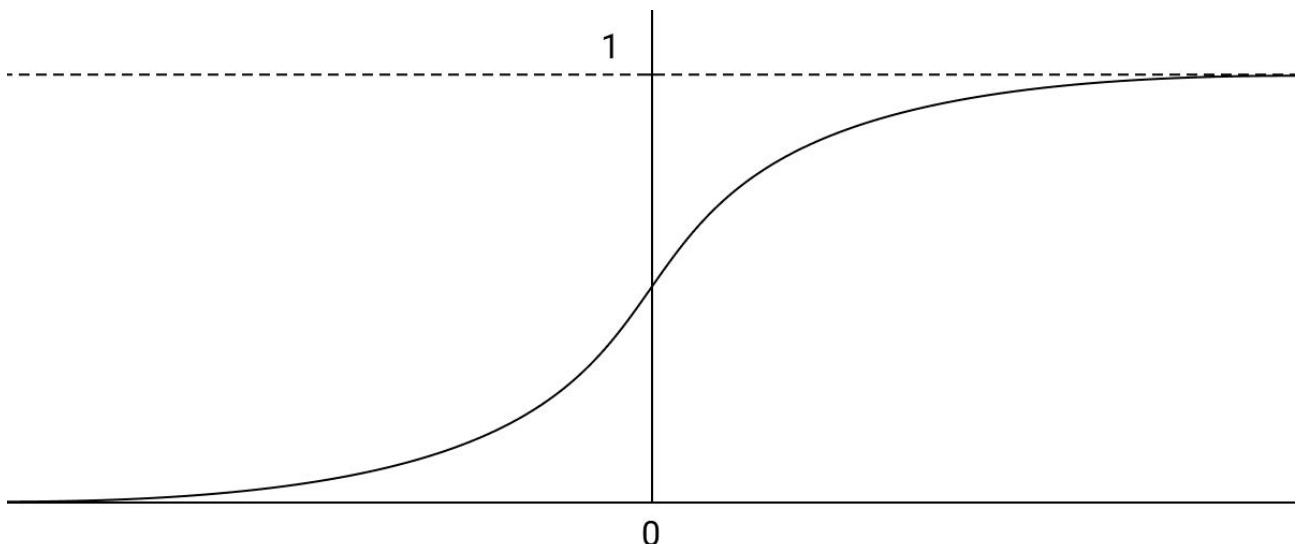
$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

где је:

$x_0$  = вредност  $x$ -а у средини функције

$L$  = максимална вредност

$k$  = стопа раста функције, или стрмост.



Слика 7. Сигмоид активациона функција.

## 2.6.2. Софтмакс

Софтмакс (енг. Softmax) функција за улаз узима вектор са реалним бројевима и нормализује их у дистрибуцију вероватноћа. Улаз су бројеви од 0 до 1, и сума свих излаза износи 1. Обично се користи у задњем слоју неуронске мреже ради предвиђања класе улазног податка. Стандардна софтмакс функција је дефинисана формулом:

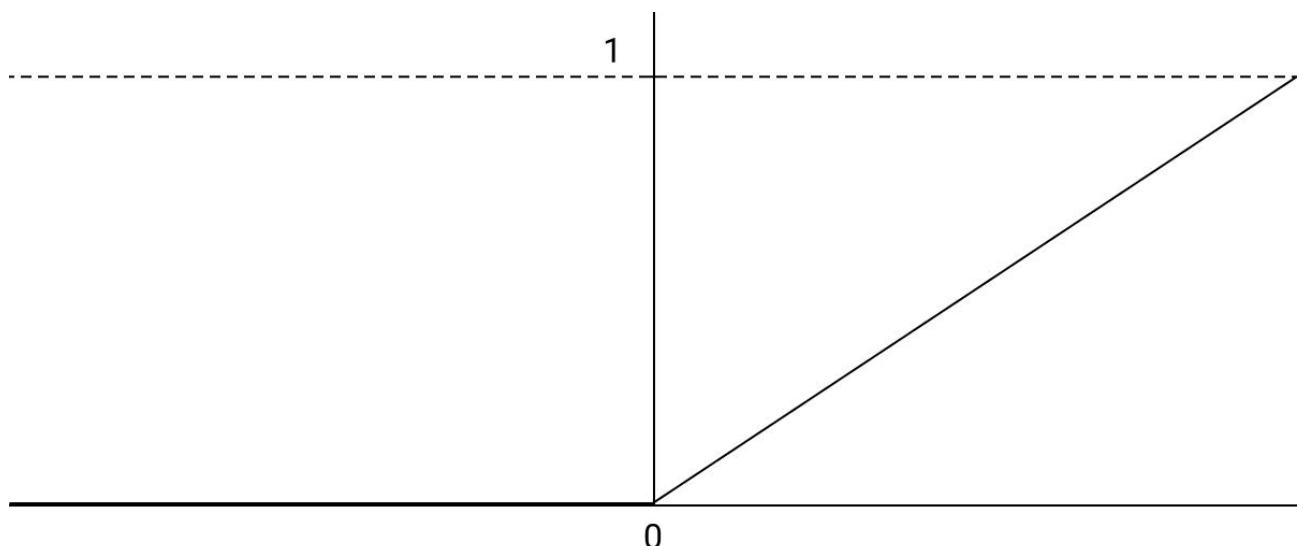
$$f(x)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

ако је  $i = 1, \dots, K$  и  $z = (z_1, \dots, z_K) \in R^K$ .

## 2.6.3. РЕЛУ

Исправљена линеарна јединица (енг. Rectified Linear Unit) или РЕЛУ. Нема лимит, али увек даје 0 ако је улаз испод нуле. После нуле се понаша као линеарна функција. Обично се користи у конволуционалним слојевима.

$f(x) = \max(0, x)$  где је  $x$  улаз за неурон.



Слика 8. РЕЛУ активациона функција.

## 2.7. Функција грешке

Сврха функције за грешку је да евалуира перформансе модела тј. да нам да нумеричку вредност која нам говори колико је модел близу тачног излаза. Два најчешћа начина да се израчуна грешка је као квадратна разлика између тачне и добијене вредности или абсолютна разлика између добијене и тачне вредности. Обично се означава као  $C$ , а тачна и добијена вредност као  $y_i$  и  $y$  респективно.

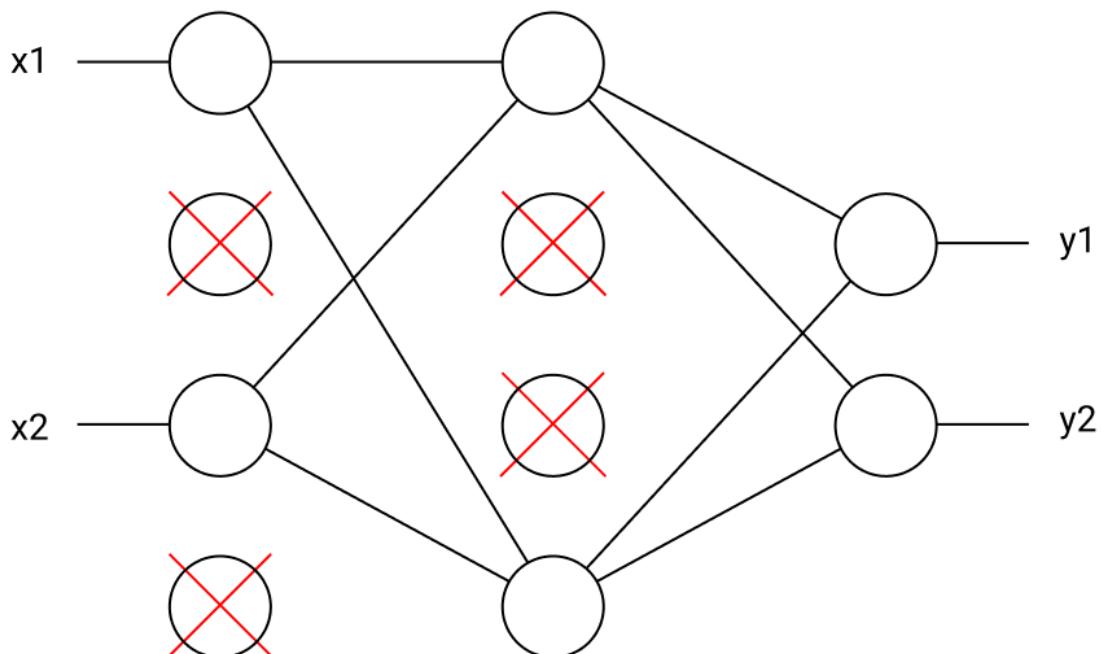
$$C = (y_i - y)^2$$

## 2.8. Регуларизација

Регуларизација је метода генерализовања модела уколико је превише прилагођен подацима.<sup>21</sup> Циљ модела је да нађе функцију која добро описује улазне податке. Уколико модел описује податке превише добро, нећемо добити добар резултат на подацима које модел никада није видео. Да би се решио овај проблем уводи се рестрикција на параметре у виду одбацивања неурона или нормализације.

## 2.9. Одбацивање неурона

Одбацивање неурона (енг. Dropout) је насумично одбацивање неурона из неког слоја са вероватноћом из Бернулијеве дистрибуције.<sup>22</sup> Посто се неурони насумично одбацују, модел мора да се прилагоди тако што ће прилагодити друге неуроне да препознају одговарајуће карактеристике.



Слика 9. Приказ мреже након одбацивања неурона.

## 2.10. Нормализација

Нормализација је алгоритам за регуларизацију који омогућава брже тренирање модела јер омогућава оптимизационом алгоритму да ради са већом стопом учења и да достигне минимум са мањим бројем епоха. Дефинише се средња вредност  $m$  и одступање од средње вредности  $p$ , тако да се подаци  $x$  налазе у границама  $m - p \leq x \leq m + p$ .<sup>23</sup>

## 2.11. Повратна пропагација

Повратна пропагација (енг. Backpropagation) је процес мењања јачине конекција у моделу на основу добијене грешке у циљу тачног класифицирања прослеђених података.<sup>24</sup>

Дефиниције и нотације:

$L$  = број слојева у мрежи

Слојеви су индексирани као  $l = 1, 2, \dots, L - 1, L$ .

Чвороби слоју  $l$  су индексирани као  $j = 0, 1, \dots, n - 1$ .

Чворови у слоју  $l - 1$  су индексирани као  $k = 0, 1, \dots, n - 1$ .

$y_j$  = тражена вредност чвора  $j$  у излазном слоју  $L$  за једну итерацију тренирања

$C$  = функција за рачунање грешке за једну итерацију тренирања

$C_j$  = функција за рачунање грешке једног чвора за једну итерацију тренирања

$w_{kj}^{(l)}$  = јачина конекције између чвора  $k$  у слоју  $l - 1$  и чврода  $j$  у слоју  $l$

$w_j^{(l)}$  = вектор који садржи све конекције од слоја  $l - 1$  према чврду  $j$  у слоју  $l$

$z_j^{(l)}$  = улаз за чврд  $j$  у слоју  $l$

$g^{(l)}$  = активациона функција за слој  $l$

$a_j^{(l)}$  = излаз активационе функције из чврда  $j$  у слоју  $l$

Грешка је квадратна разлика између излаза активације и тачног излаза за чврда  $j$  у слоју  $L$ .

$$C_j = (a_j^{(L)} - y_j)^2$$

Да бих добили укупну грешку, рачунамо суму грешака свих чврдова на излазном слоју.

$$C = \sum_{j=0}^{n-1} C_j$$

Излаз активације чврда  $j$  у слоју  $l$  је резултат прослеђивања  $z_j^{(l)}$  кроз активациону функцију  $g^{(l)}$ .

$$a_j^{(l)} = g^{(l)}(z_j^{(l)})$$

Улаз у чврд  $j$  у слоју  $l$  је пондерисана suma активационих излаза из предходног слоја  $l - 1$ .

$$z_j^{(l)} = \sum_{k=0}^{n-1} w_{kj}^{(l)} a_k^{(l-1)}$$

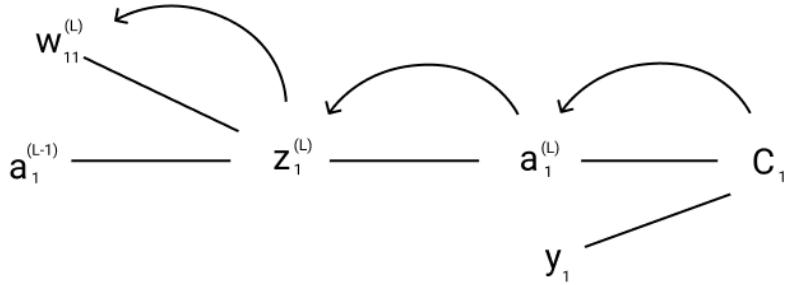
Закључујемо да се рачунање грешке рачуна композицијом функција:

$$C = \sum_{j=0}^{n-1} C_j(a_j^{(l)}(z_j^{(l)}(w_j^{(l)})))$$

Пошто не можемо да мењамо грешку директно, мењамо променљиву  $w_j^{(l)}$ , сразмерно величини грешке. Задатак је израчунати колико се грешка мења са променом  $w_j^{(l)}$ , што се постиже диференцирањем  $C_j$  у односу на  $w_j^{(l)}$ , коришћењем извода сложене функције.

Ако желимо да нађемо извод  $C_1$  у односу на променљиву  $w_{11}^{(L)}$ :

$$\frac{\partial C_1}{\partial w_{11}^{(L)}} = \frac{\partial C_1}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial w_{11}^{(L)}}$$



Слика 10. Приказ тока повратне пропагације.

Прва ставка:

$$\frac{\partial C_1}{\partial a_1^{(L)}} = \frac{\partial}{\partial a_1^{(L)}} (a_1^{(L)} - y_1)^2$$

Узима се извод само од чланова који садрже  $\partial a_1^{(L)}$ , остали чланови се гледају као константе, чији је извод 0. Своди се на:

$$\frac{\partial C_1}{\partial a_1^{(L)}} = 2(a_1^{(L)} - y_1)$$

Друга ставка:

$$\frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} = \frac{\partial}{\partial z_1^{(L)}} g^{(L)}(z_1^{(L)}) = g'^{(L)}(z_1^{(L)})$$

Трећа ставка:

$$\frac{\partial z_1^{(L)}}{\partial w_{11}^{(L)}} = \frac{\partial}{\partial w_{11}^{(L)}} w_{00}^{(L)} a_0^{(L-1)} + \frac{\partial}{\partial w_{11}^{(L)}} w_{11}^{(L)} a_1^{(L-1)} + \dots + \frac{\partial}{\partial w_{11}^{(L)}} w_{kj}^{(L)} a_j^{(L-1)}$$

$$\frac{\partial z_1^{(L)}}{\partial w_{11}^{(L)}} = a_1^{(L-1)}$$

Комбиновањем ових ставки добијамо:

$$\frac{\partial C_1}{\partial w_{11}^{(L)}} = 2(a_1^{(L)} - y_1) g'(z_1^{(L)}) a_1^{(L-1)}$$

Предходни пример показује како се грешка мења у односу на јачину конекције представљену са променљивом  $w$  приликом једне итерације тј. једног улаза током тренирања. Больи резултати се добијају ако узмемо средњу вредност извода од више итерација.

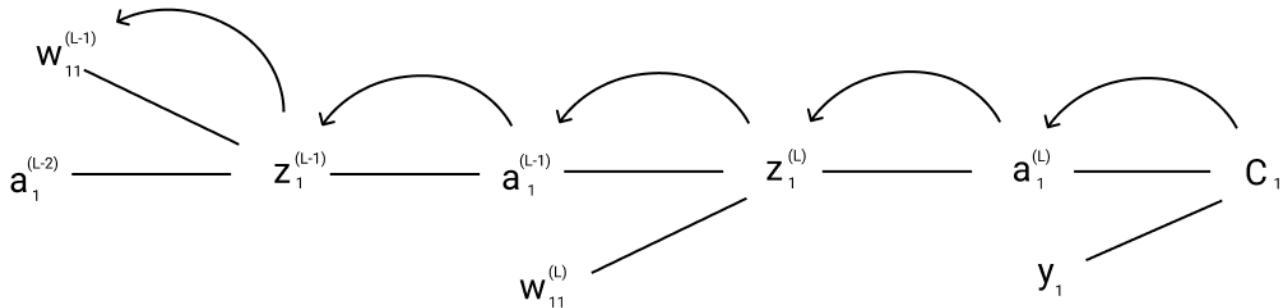
$w_{11}^{(L)}$  се затим мења формулом:

$$w_{11}^{'(L)} = w_{11}^{(L)} - r \frac{\partial C_1}{\partial w_{11}^{(L)}}$$

где је  $r$  стопа учења, број од 0 до 1. Смер у коме ће се  $w_{11}^{(L)}$  кретати зависи од знака извода.

Ако желимо да променимо конекцију дубље у мрежи, све што треба да је да откријемо како се коначна грешка мења у односу на ту конекцију.

$$\frac{\partial C_1}{\partial w_{11}^{(L-1)}} = \frac{\partial C_1}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial a_1^{(L-1)}} \frac{\partial a_1^{(L-1)}}{\partial z_1^{(L-1)}} \frac{\partial z_1^{(L-1)}}{\partial w_{11}^{(L-1)}}$$



Слика 11. Приказ тока повратне пропагације кроз мрежу са више слојева.

Као додатан улаз у чврот користи се и померај. Обично се означава словом  $b$ . Уз суму конекција, активациону функција неурона такође прима и померај, све остале калкулације остају исте.

$$z_j^{(l)} = \left( \sum_{k=0}^{n-1} w_{kj}^{(l)} a_k^{(l-1)} \right) + b_k$$

Учење помераја се врши на сличан начин као и код конекција:

$$\frac{\partial C_1}{\partial b_1^{(L)}} = \frac{\partial C_1}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial b_1^{(L)}}$$

Зависност улаза у активациону функцију од помераја је 1:

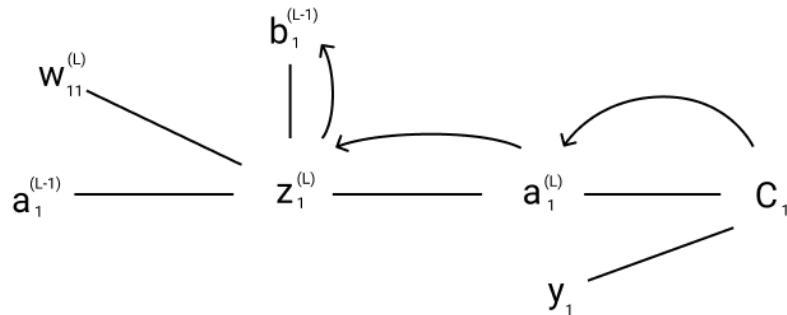
$$\frac{\partial z_1^{(L)}}{\partial b_1^{(L)}} = 1$$

Коначна једначина је затим:

$$\frac{\partial C_1}{\partial b_1^{(L)}} = 2(a_1^{(L)} - y_1) g'(z_1^{(L)})$$

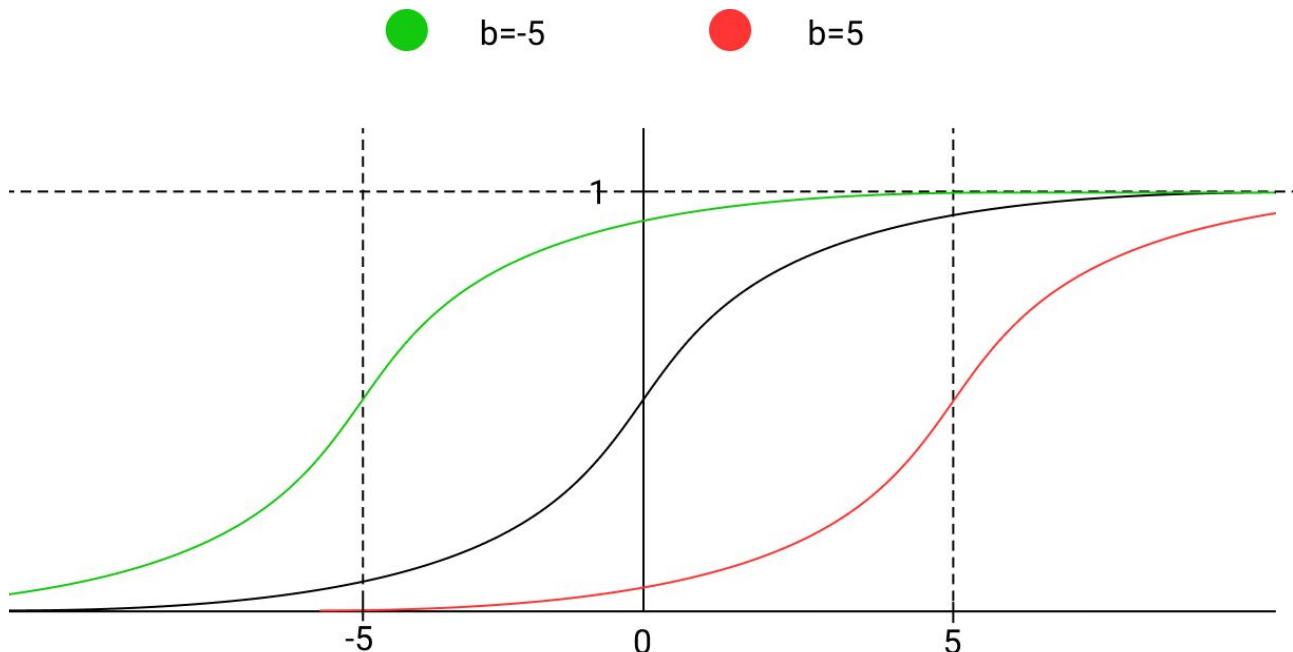
Мењање помераја<sup>25</sup> (енг. Bias) се врши истом једначином као и код конекција:

$$b_1'^{(L)} = b_1^{(L)} - r \frac{\partial C_1}{\partial b_1^{(L)}}$$



Слика 12. Приказ тока повратне пропагације у мрежи са померајем.

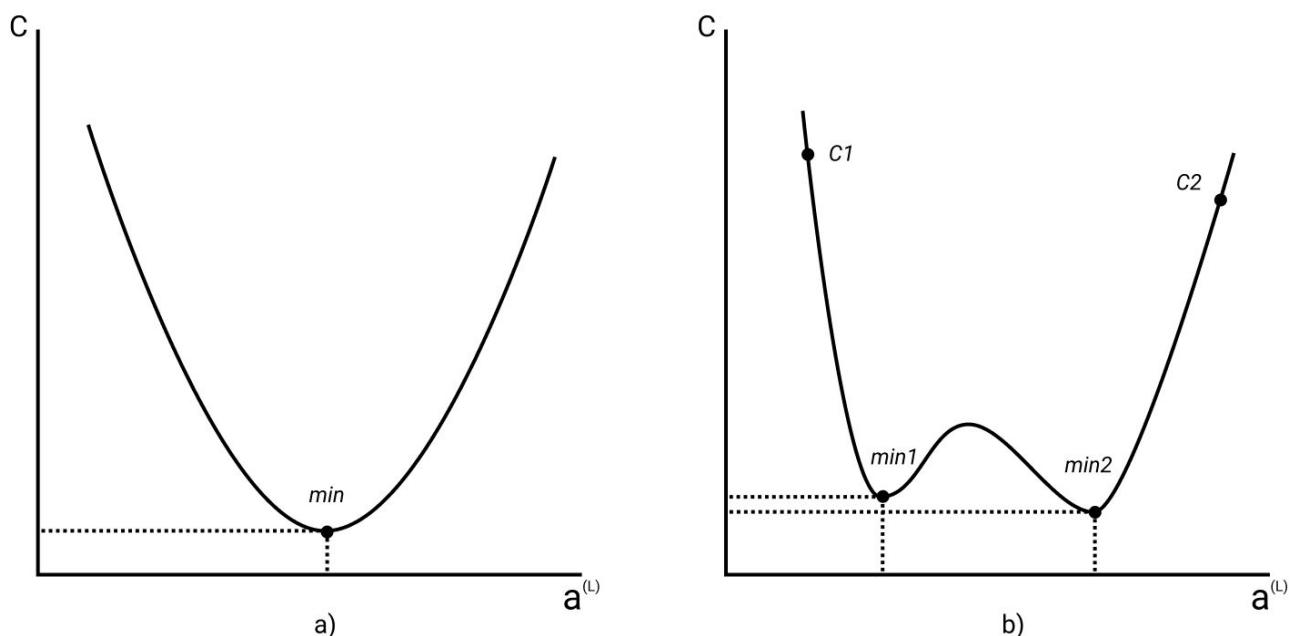
Како што име каже, он помера активациону функцију у правцу знака помераја по x оси. Пример: Ако је  $b = -5$ , и користимо сигмоид активациону функцију излаз ће бити потиснут ако је улаз мањи од 5.



Слика 13. Приказ утицаја помераја на активациону функцију.

## 2.12. Спуштање градијентом

Поменули смо да је циљ учења да се смањи грешка. Процес постепеног смањивања грешке се зове спуштање градијентом (енг. Gradient Descent), што значи налажење минимума функције за грешку.<sup>26</sup> Неке функције су једноставне и локација минимума је очигледна, али функција грешке може бити много комплекснија и може имати више минимума који се не разликују превише. Налажење најмањег минимума зависи од тога како су иницијализовани параметри мреже. При иницијализацији, тежине конекција се додељују насумично, тако да не можемо знати дали ће се грешка спустити у најмањи минимум. Због тога би било пожељно да се тренирање понови неколико пута.



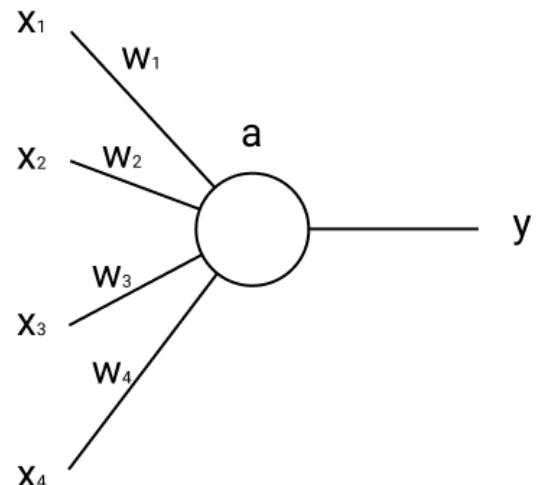
Слика 14. а) Функција грешке са једним минимумом. б) Функција грешке са два минимума.

На слици видимо да лева функција за грешку има очигледан минимум, али функција на десној страни има два минимума. Ако при иницијализацији параметара, добијемо грешку  $C_1$  минимум ће бити  $min_1$ , у супротном ако је иницијална грешка близу  $C_2$ , минимум ће бити у  $min_2$ . Приметимо да између  $min_1$  и  $min_2$  постоји брдо које спречава грешку да нађе најмањи минимум. Иако се то брдо може прескочити ако користимо већу стопу учења, рекли смо да са великим стопом учења је теже наћи минимум, тако да је решење опадајућа стопа учења.

Опадајућа стопа учења подразумева да се стопа учења смањује временом. Иницијализовањем велике стопе учења би значило да су кораци учења велики на почетку, чиме би испрескакали брда која би се нашла на путу, али би било теже наћи најмању грешку због чега постепеним смањивањем стопе учења при свакој итерацији би резултовало у лакшем налажењу минимума при крају учења.

## 2.13. Пример учења

id	$x_1$	$x_2$	$x_3$	$x_4$	$y_i$
0	0.9	0.9	0.1	0.1	1
1	0.1	0.1	0.9	0.9	0



Слика 15. Перцептрон са 2 примерака податка са 4 димензије.

На примеру имамо 4 улаза и један излазни неурон. Као активациону функцију користимо сигмоид. Ако насумично дефинишемо вредности конекција тако да су:  $w_1 = 0.2, w_2 = 0.5, w_3 = 0.3, w_4 = 0.6$ .

Као што смо рекли, грешка се рачуна одузимањем излаза из активационе функције и тачне вредности класификације.

$$C_1 = (sig(x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4) - y_i)^2$$

Додељивањем променљиви подацима са идентификационим бројем 0 добијамо да је грешка 0.107:

$$\begin{aligned} C_1 &= (sig(0.9 * 0.2 + 0.9 * 0.5 + 0.1 * 0.3 + 0.1 * 0.6) - 1)^2 \\ C_1 &= (sig(0.72) - 1)^2 = (0.672 - 1)^2 = 0.107 \end{aligned}$$

Извод активационе функције сигмоид:

$$sig'(x) = sig(x)(1 - sig(x))$$

Промена грешке у односу на промену  $w_1$ :

$$\frac{\partial L}{\partial w_1} = 2(0.672 - 1) * 0.672 * (1 - 0.672) * 0.9 = -0.129$$

Нова вредност  $w_1$ :

$$w_1 = 0.2 - (-0.129) = 0.329$$

Исти процес се примени и за остале конекције:

$$w_2 = 0.629$$

$$w_3 = 0.314$$

$$w_4 = 0.614$$

При следећој итерацији користимо улаз са идентификационим бројем 1:

$$C_1 = (sig(0.9 * 0.329 + 0.9 * 0.629 + 0.1 * 0.314 + 0.2 * 0.614) - 1)^2 = 0.07$$

Грешка за улаз са идентификационим бројем 2 је затим:

$$C_2 = 0.51$$

Нове вредности конекција:

$$w_1 = 0.299$$

$$w_2 = 0.599$$

$$w_3 = 0.052$$

$$w_4 = 0.352$$

Видимо да је грешка мања у односу на грешку пре итерације:

$$C_2 = 0.373$$

Када би покренули алгоритам више итерација, конекције  $w_1$  и  $w_2$  порасле, а  $w_3$  и  $w_4$  би се смањиле испод нуле чиме би се грешка смањила и резултат класификације би био тачан.

### 3. Конволуције

Концепт конволуције (енг. Convolution) потиче из области обрађивања слика. Идеја је да се нека слика проследи кроз филтер који је трансформише и изражава неке карактеристике док потискује друге.<sup>27</sup>



3x3 filter		
.5	.5	.5
.5	.5	.5
.5	.5	.5



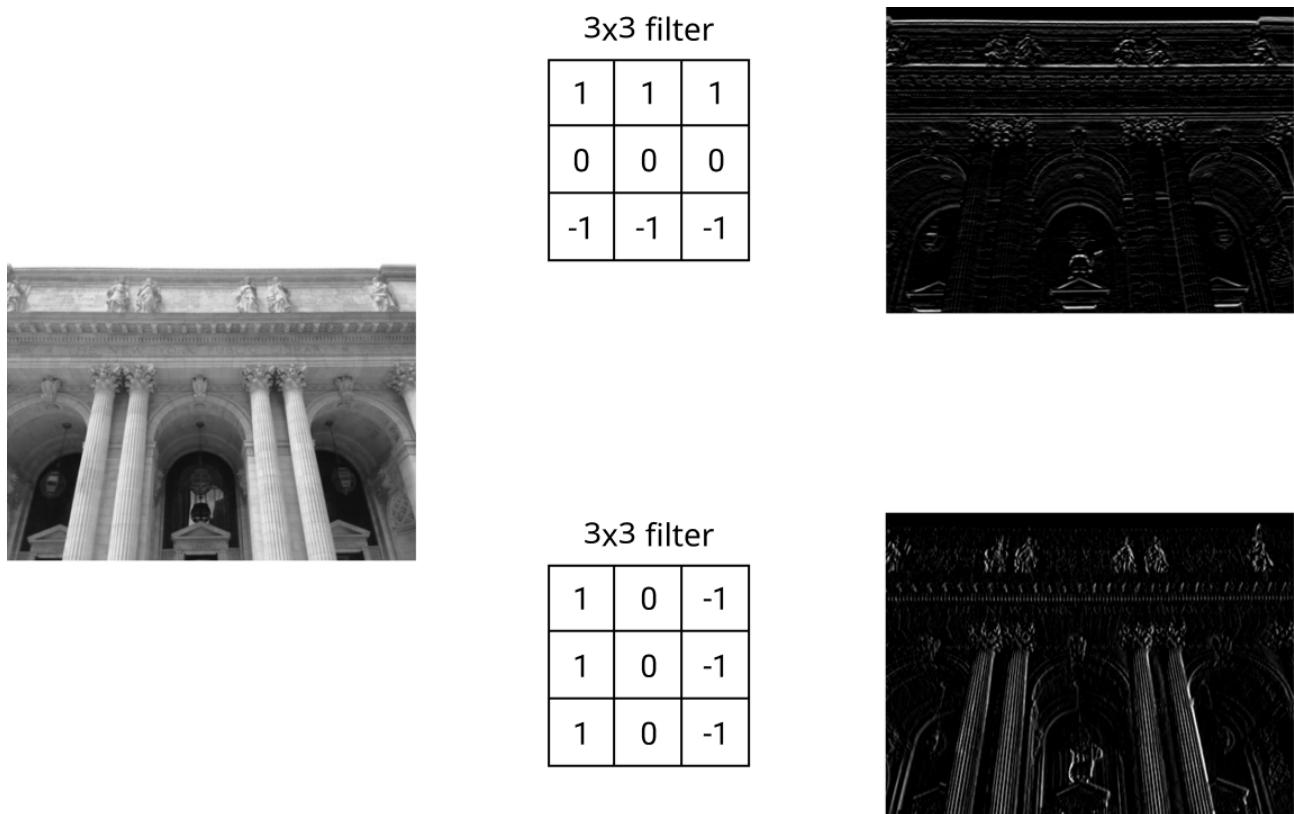
Слика 16. Пример замагљивања слике помоћу филтера.

Ефекат замагљивања (енг. Blurring) слике се постиже помоћу филтера. На слици је дат пример филтера величине 3x3, који поставља преко сваког пиксела слике. Затим се примењује елементно множење између филтера и одговарајућих пиксела испод филтера.

Узима се сума свих елемената у добијеној матрици и добијена вредност се поставља уместо пиксела у центру филтера.

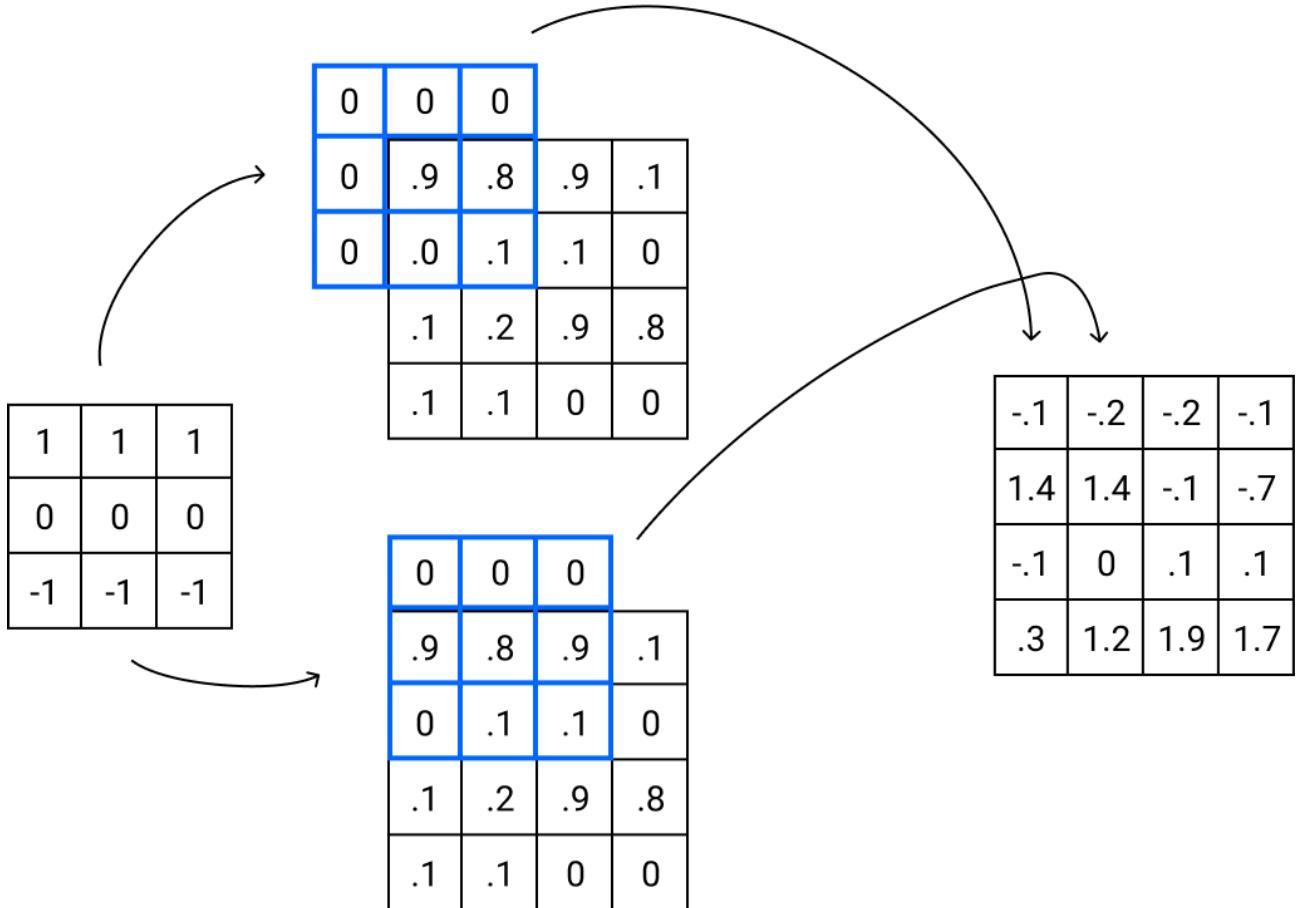
### 3.1. Филтери

У области обраде слика, човек је тај који дефинише одређене филтере у циљу постизања одговарајућих ефеката. Напротив, у области машинског учења, циљ модела је да сам нађе одговарајуће вредности који ће филтер садржати, како би тачно класифицирао одређену слику. Филтери могу да линије, ивице, кругове, па чак и комплексне облике као што су делови тела животиња и људи, делове аутомобила, зграда итд.<sup>28</sup>



Слика 17. Приказ филтера који детектује хоризонталне и вертикалне линије.

Циљ филтера је да изрази одлике које које неуронска мрежа може да искористи при класификацији неке слике. Представљају се у виду матрица који пролазе преко сваког пиксела на слици и извршавају елементарно множење над пикセルима преко којих пролазе.

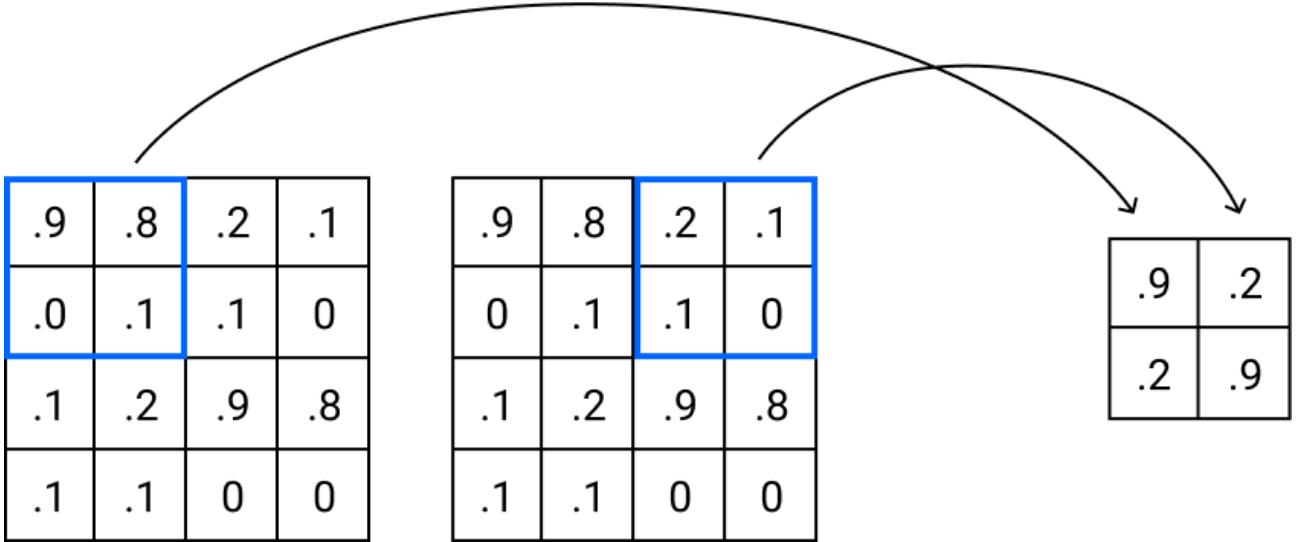


Слика 18. Приказ примене 3x3 филтера на 4x4 матрицу.

### 3.2. Удружијање

У неким случајевима, смањивањем димензија слике се постиже боље изражавање карактеристика слике, јер се тиме одклањају шумови и смањује комплексност слике.

Најпознатија метода удружијања је удружијање по максималној вредности (енг. MaxPooling). Ако дефинишемо да је филтер величине 2x2 са померањем од 2 пиксела:

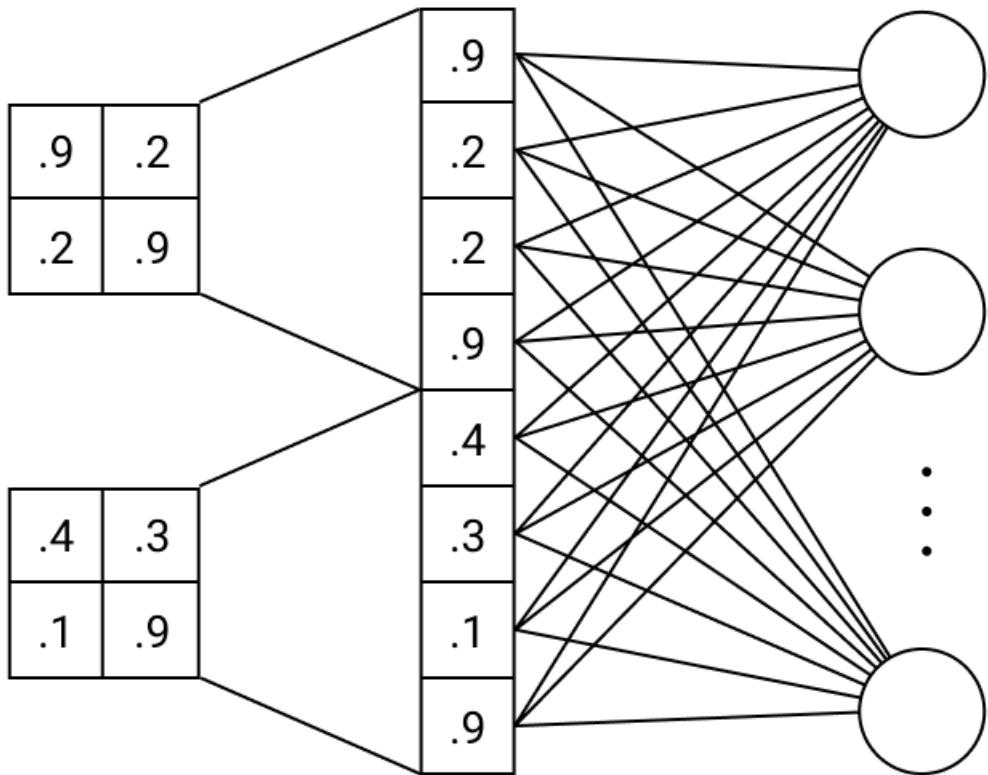


Слика 19. Приказ удруживања пиксела користећи 2x2 филтера и помераја од 2 пиксела.

Конволуционалне мреже могу имати више филтера за одлике и више филтера за удруживање пре него што се пренесу у потпуно конектовани слој.

### 3.3. Потпуно конектовани слој

Иако су конволуције добре за препознавање различитик одлика слике, нису добри у разликовању једне класе слика од других. Зато се користе традиционалне неуронске мреже. Да бих филтере претворили у слој чвррова морамо их поравнати. Поравњавање (енг. Flattening) је процес претварања вишедимензионалних матрица у једнодимензионални вектор.<sup>29</sup> Пошто ће сваки филтер утицати на коначну одлуку модела, поравнати филтери се спајају у један велики вектор. Ако смо имали 2 филтера са 2x2 димензијама, резултат је један вектор са 8 бројева.



Слика 20. Поравњавање филтера и прослеђивања у потпуно конектовани слој.

### 3.4. Учење филтера

Процес налажења или учења филтера се ради на сличан начин као налажења јачине конекција у неуронким мрежама.

Узећемо у обзир једноставан модел који садрижи један филтер, и један чврт потпуно конектованог слоја.

$C$  = функција за рачунање грешке

$y$  = тачан резултат

$w_i^F$  = јачина конекције у потпуно конектованом слоју

$w_{jk}^C$  = јачина једног пиксела у филтеру конволуционог слоја

$z^F$  = улаз у чврт поптпuno конектованог слоја

$z^C$  = улаз у пиксел конволуционог слоја

$g^F$  = активациона функција за потпуно конектовани слој

$g^C$  = активациона функција за конволуциони слој

$a^F$  = излаз активационе функције потпуно конектованог слоја

$a_i^C$  = излаз активационе функције филтера конволуционог слоја

$r$  = стопа учења

За рачунање грешке користићемо квадратну разлику између тачне вредност и добијене вредности.

$$C = (y - a^F)^2$$

Излаз активационе  $g^F$  у потпуно конектованом слоју:

$$a^F = g^F(z^F)$$

Улаз у чвор у потпуно конектованом слоју:

$$z^F = \sum_{i=0}^I a_i^C w_i^F$$

Излаз активационе функције  $g^C$  у конволуционом слоју:

$$a_i^C = g^C(z_i^C)$$

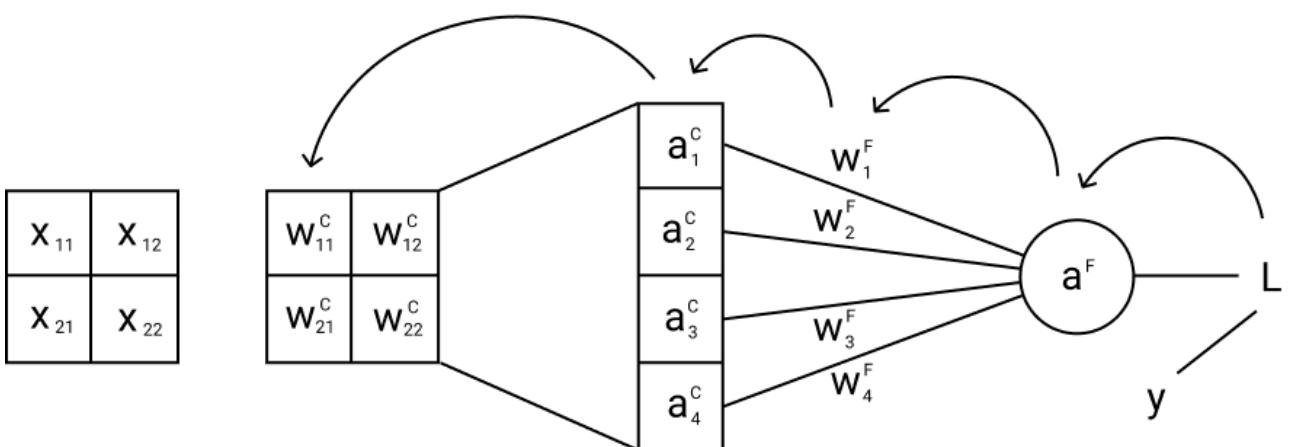
Излаз филтера у конволуционалном слоју:

$$z_i^C = \sum_{j=0}^J \sum_{k=0}^K x_{jk} w_{jk}^C$$

Ако желимо да нађемо како се вредност  $w_{jk}^C$  у филтеру конволуционог слоја мења у односу на грешку:

$$\frac{\partial L}{\partial w_{jk}^C} = \frac{\partial L}{\partial a^F} \frac{\partial a^F}{\partial z^F} \frac{\partial z^F}{\partial a_i^C} \frac{\partial a_i^C}{\partial z_C} \frac{\partial z_C}{\partial w_{jk}^C}$$

У следећем примеру је приказано како се пиксел  $w_{11}^C$  у филтеру мења у односу на грешку  $L$ :



Слика 21. Пример повратне пропагације у конволуционалним слојевима.

$$\frac{\partial L}{\partial w_{11}^C} = 2(y - a^F) (g^F(z^F)) w_1^F (g^F(z_1^C)) x_{11}$$

А затим се вредност ажурира у смеру у коме је грешка мања, са величином корака  $r$ :

$$w_{11}^{'C} = w_{11}^C - r \frac{\partial L}{\partial w_{11}^C}$$

## 4. Проблем

Решавамо проблем класификације ручно писаних цифара помоћу неуронских мрежа. База података коју користимо се зове МНИСТ (енг. Modified National Institute of Standards and Technology-MNIST) и садржи 65,000 примерака цифара од 0 до 9. Сваки пример са собом носи ознаку која говори који је то број.<sup>30</sup>

label = 5



label = 0



label = 4



label = 1



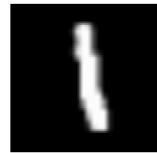
label = 9



label = 2



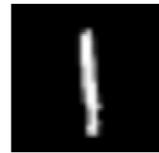
label = 1



label = 3



label = 1



label = 4



label = 3



label = 5



label = 3



label = 6



label = 1



label = 7



label = 2



label = 8



label = 6

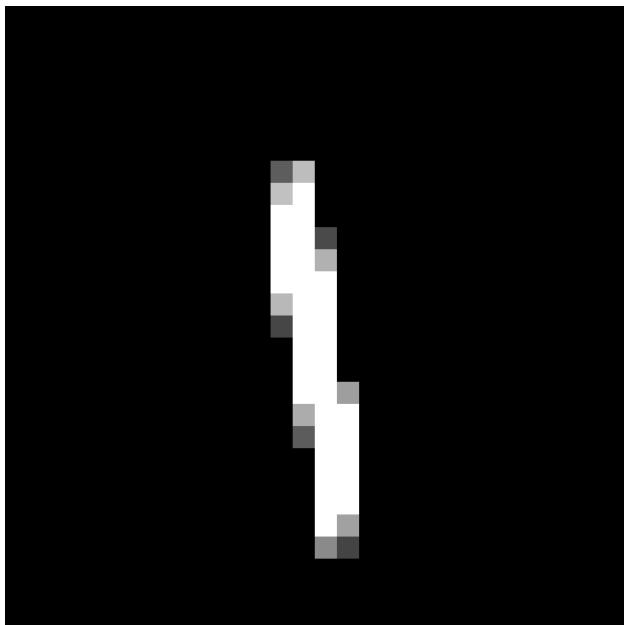


label = 9



Слика 22. Пимери из МНИСТ базе података.

Примери су црно-беле слике димензија 28x28. Сваки пиксел је представљен бројем од 0 до 255, који ће у нашем случају бити претворени у вредност изнеђу 0 и 1 и представљени као матрица.

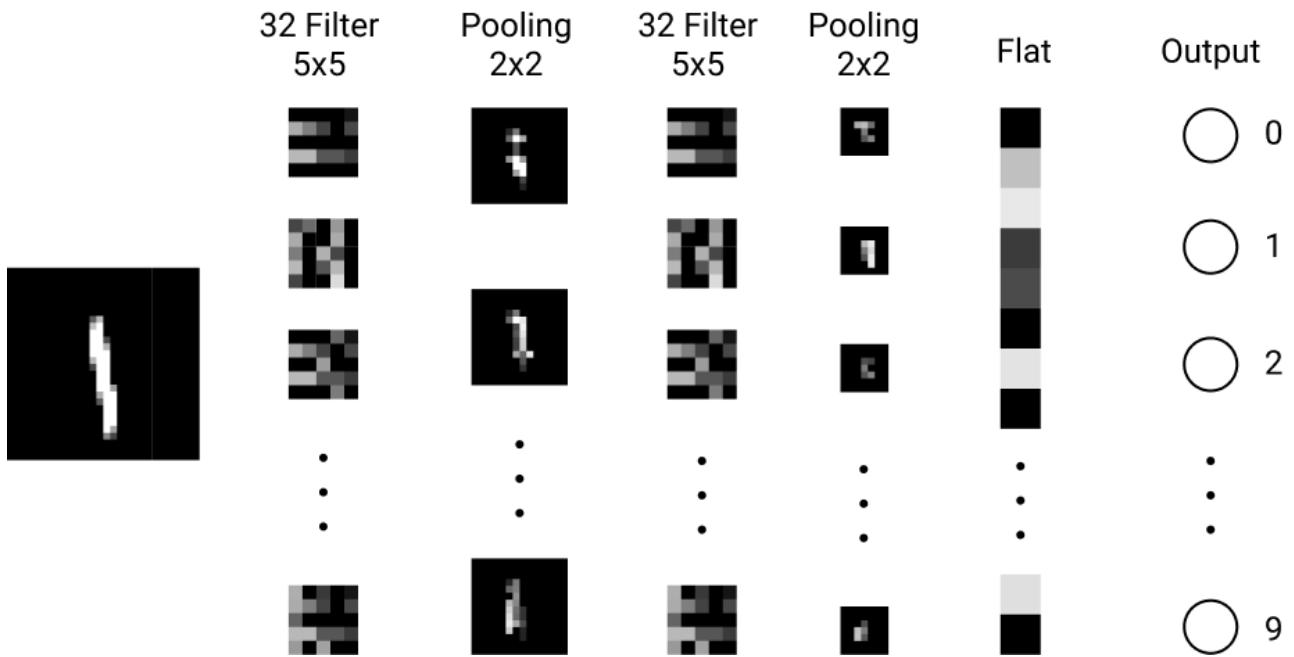


Слика 23. Представљање слике у виду матрице.

Подаци ће бити подељени у две групе. Група за тренирање и група за валидацију. Ова процедура је стандардна у области машинског учења, јер нам омогућава да боље генерализујемо модел. Пошто ћемо део података оставити за тестирање перформанси то нам даје јаснију слику како ће се модел понашати над подацима које никад није видео, што нам може помоћи у побољшању нашег модела.

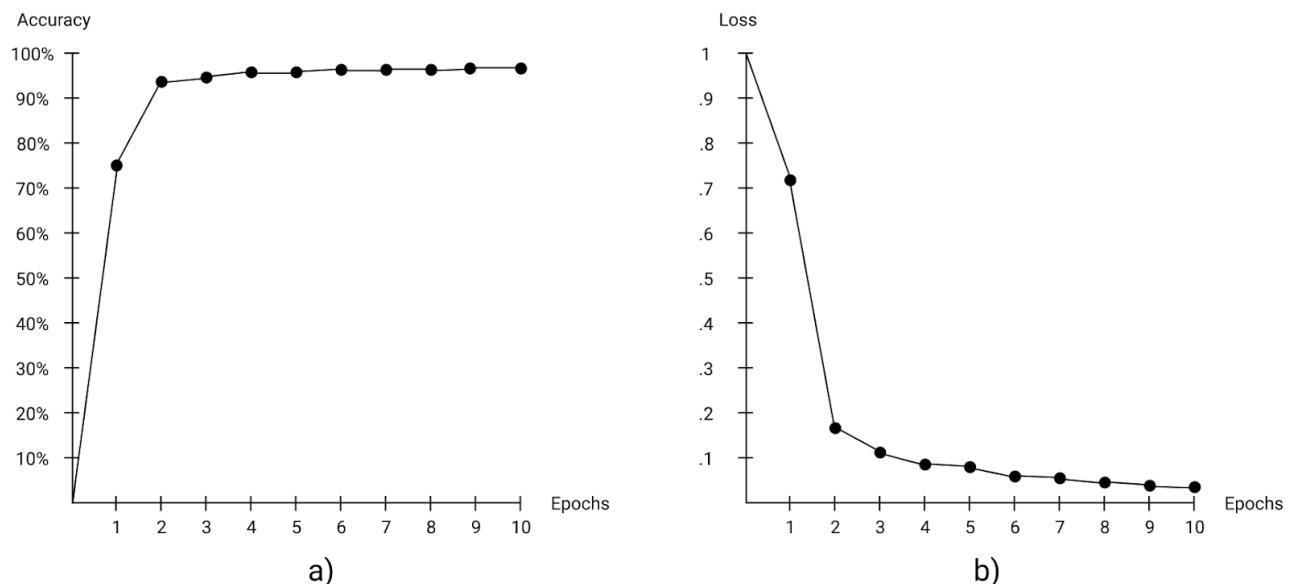
Модел има 3242 параметара и архитектура се состоји од:

- 1) Слоја са 32 филтера величине  $5 \times 5$ . Активациона функција за овај слој је РЕЛУ.
  - 2) Слоја за удрживање пиксела величине  $2 \times 2$  и померањем од 2 пиксела. Излаз је матрица величине  $14 \times 14$ .
  - 3) Још једног слоја са 32 филтера величине  $5 \times 5$ . Активациона функција за овај слој је РЕЛУ.
  - 4) Другог слоја за удржавање филтера величине  $2 \times 2$  и померањем од 2 пиксела. Излаз је матрица величине  $7 \times 7$ .
  - 5) Слој за поравњавање . Пошто је улаз матрица  $7 \times 7$  и број филтера из предходног слоја је 32, величина вектора је  $7 \times 7 \times 32 = 1568$ .
  - 6) Потпуно конектовани слој са 10 неурона. Сваки неурон представља једну класу (од 0 до 9). Активациона функција је Софтмакс јер ће суме излаза свих неурона бити 1, што нам даје вектор вероватноћа што је погодно за класификацију.



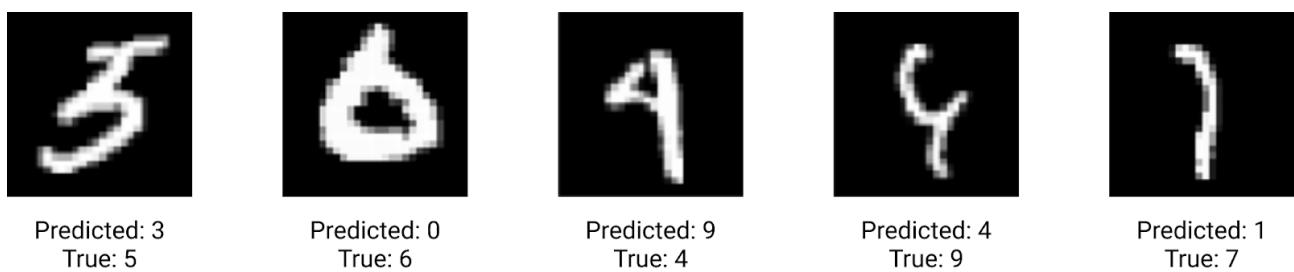
Слика 24. Архитектура нашег модела.

Тренинг траје 10 епоха, што значи да ће кроз модел проћи цео сет за тренирање 10 пута. Модел класифицира податке за тестирање у 99.3% случајева. Грешка после тренирања је смањена на 0.02.



Слика 25. Историја прецизности и грешке нашег модела током 10 епоха.

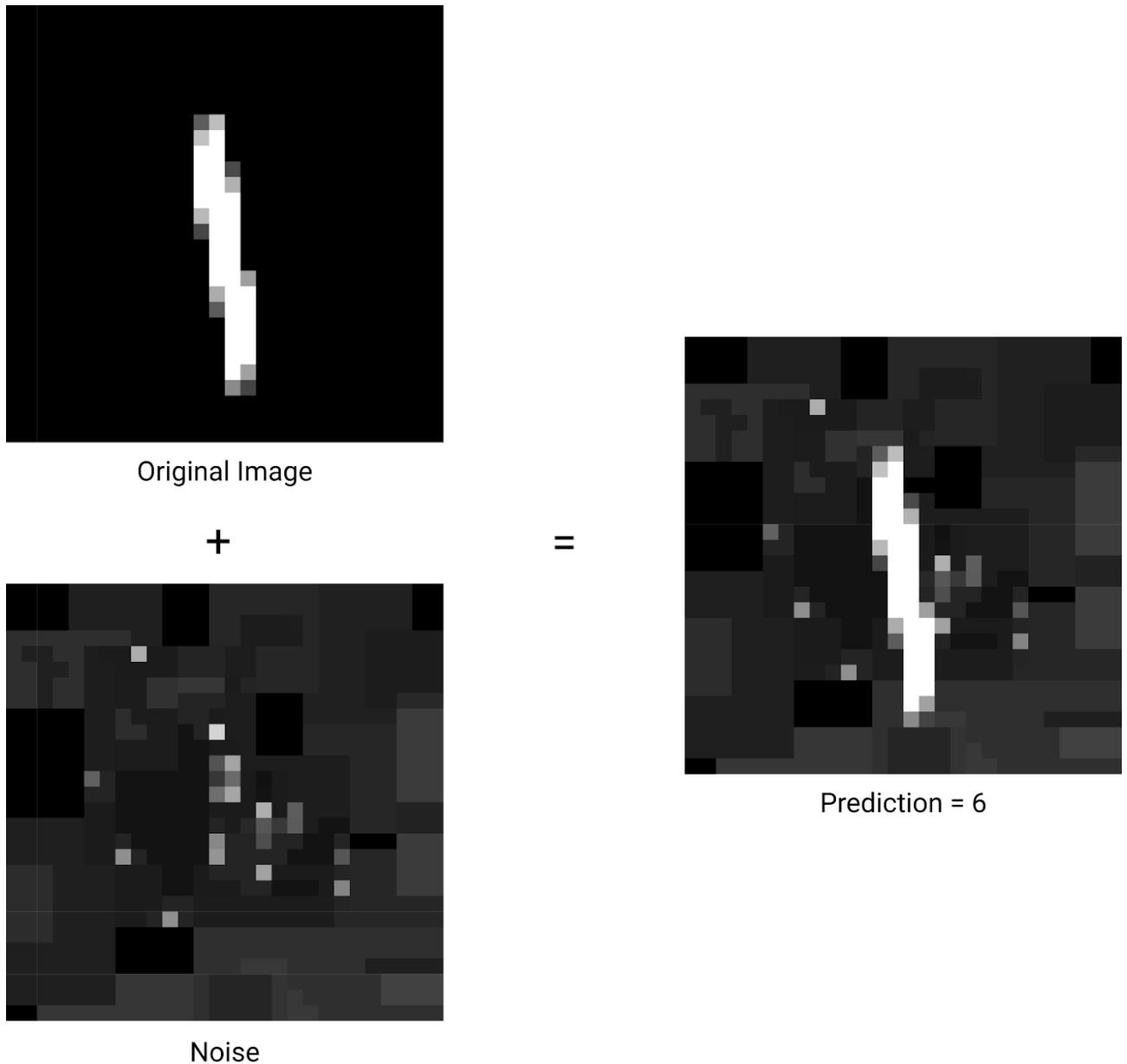
Примери погрешно класификованих слика:



Слика 26. Примери погрешно класификованих слика.

Видимо да велики проценат цифара који су мискласификовани имају карактеристике више цифара, што објашњава зашто модел није у стању да са сигурношћу класификује цифру. нпр. прва цифра садржи карактеристике цифре 3 и 5. Пошто ни човек не може са сигурношћу рећи која је цифра у питању, постоји шанса да примери нису добро означени, у којем случају би значило да модел боље распознаје цифре од човека.

Једна од највећих мана је то што са додавањем шумова модел може да мискласификује слику.<sup>31</sup>



Слика 27. Резултат додавања шумова у један од примера.

То се дешава јер модел не зна абстрактне појмове као што су линија, круг или било који други облици. Да би модел знао за појам круга или неких других облика, мора имати много више параметара и много већу базу података на располагању која саджи много више појмова из реалног света. Све што модел види јесу бројеви, и карактеристике прави на основу тога који бројеви дају најбољи резултат, и то некада не мора да буде нешто што би човеку било

логично. Због тога, додавањем мало шумова модел види потпуно другу цифру, иако човек види веома малу разлику.

## 4.1. Познати модели

### 4.1.1. ЛеНет (енг. LeNet)

Популарни конволуционални модел направљен 90-их година. Данас је стандард за почетнике који хоће да уђу у ову област. Састоји је од 8 слојева и 60000 параметара и може се покренути на процесорима. Достиже прецизност од 97.9% и грешку од 0.06.<sup>32</sup>

Табела 1. Архитектура ЛеНет модела.

Слој	Филтери	Величина слоја	Величина филтера	Померај	Активациона функција
Улаз	-	28x28	-	-	-
Конволуција	6	28x28	5x5	1	танх
Удружицање	6	14x14	2x2	2	танх
Конволуција	16	10x10	5x5	1	танх
Удружицање	16	5x5	2x2	2	танх
Конволуција	120	1x1	5x5	1	танх
Потпуно конекнтован	-	84	-	-	танх
Излаз	-	10	-	-	софтмакс

### 4.1.2. АлексНет (енг. AlexNet)

Освојио ИЛСВРЦ такмичење 2012-е године. Има 60 милиона параметара и 650 хиљада неурона. Иако је овај модел дизајниран за базе са 1000 класа и милион примерака, постиже преко 97% прецизности на МНИСТ подацима.<sup>33</sup>

Табела 2. Архитектура АлексНет модела.

Слој	Филтери	Величина слоја	Величина филтера	Померај	Активациона функција
Улаз	-	227x227	11x11	-	-
Конволуција	96	55x55	3x3	4	релу
Удружицање	96	27x27	5x5	2	релу

Конволуција	256	27x27	3x3	1	релу
Удружицање	256	13x13	3x3	2	релу
Конволуција	384	13x13	3x3	1	релу
Конволуција	384	13x13	3x3	1	релу
Конволуција	256	13x13	3x3	1	релу
Удружицање	256	6x6	3x3	2	релу
Потпуно конектован	-	9216	-	-	релу
Потпуно конектован	-	4096	-	-	релу
Потпуно конектован	-	4096	-	-	релу
Излаз	-	1000	-	-	софтмакс

#### 4.1.3. ВГГ (енг. VGG)

Добио друго место на ИмиџНет такмичењу 2014 године. Има 138 милиона параметара и постиже прецизност од 98.4% на МНИСТ подацима.<sup>34</sup>

Табела 3. Архитектура ВГГ модела.

Слој	Филтери	Величина слоја	Величина филтера	Померај	Активациона функција
Улаз	-	224x224	-	-	-
2x Конволуција	64	224x224	3x3	1	релу
Удружицање	64	112x112	3x3	2	релу
2x Конволуција	128	112x112	3x3	1	релу
Удружицање	128	56x56	3x3	2	релу
2x Конволуција	256	56x56	3x3	1	релу
Удружицање	256	28x28	3x3	2	релу
3x	512	28x28	3x3	1	релу

Конволуција					
Удруживање	512	14x14	3x3	2	релу
3x Конволуција	512	14x14	3x3	1	релу
Удруживање	512	7x7	3x3	2	релу
Потпуно конектован	-	25088	-	-	релу
Потпуно конектован	-	4096	-	-	релу
Потпуно конектован	-	4096	-	-	релу
Излаз	-	1000	-	-	софтмакс

## 4.2. Познате библиотеке

Пошто неуронске мреже нису имале много промена током година што се тиче математике која их покреће, неби било продуктивно да свако ко жели да користи неуронске мреже напише сав код испочетка.

### 4.2.1. Тензорфлоу (енг. Tensorflow)

Развијен и коришћен од стране Гугл-а за унутрашње потребе још од 2011-е године. 2015-е изворни код је постадо доступан, бесплатно. Популарност му је расла од самог почетка и данас је стандард за прављење неуронских мрежа јер је оптимизована да ради на графичким картицама што знатно убрзава време тренирања. Доступна је у Пајтону (енг. Python), Џ++ (енг. C++), Јаваскрипти (енг. Javascript), Јави (енг. Java) и Го (енг. Go) програмском језику.

### 4.2.2. ПајТорч (енг. PyTorch)

Прва верзија је изашла годину дана после Тензорфлоу-а и убрзо је постала његов главни конкурент. Популарност је добила када је Тесла објавио да је користи за своја аутономна возила. Компаније се обично опредељују за ПајТорч због ненаклоности према Гугл-у. Доступна је само у Пајтон програмском језику и оптимизована је за рад на графичким картицама.

## 4.3. Коришћење у производњи

У производњи ови модели се користе у сврхе распознавања цифара са неких докумената, уличних знакова, регистарских таблица, итд. Гугл користи ове моделе ради распознавања уличних бројева на објектима, како би направио прецизнију мапу и бољу претрагу адреса.<sup>35</sup>



Слика 28. Примери уличних бројева из реалног света.

Иако је могуће да се направе модели који могу да препознају више цифара у једној слици или слике са одређеним шумовима сто је често у подацима из реалног света, то би захтевало модел са много параметара, самим тим и дуже тренирање. Када би правили модел са сличном архитектуром као што смо направили у овом раду, прво би морали прерадити слике на одређен начин пре тренирања. Први корак је конвертовати слику у црно-бели формат, затим исећи слику тако да изолујемо сваку цифру, што се може постићи користећи другог модела или користећи неке технике обраде слика. Затим је потребно ротирати слику и ако је могуће одстрanити шумове.

#### 4.4. Закључак

Фундаменталне математиче операције се ретко мењају зато што је тешко измислiti неку иновативну методу и зато што перформансе неке методе нису одмах очигледне. Најбољи пример су ГАН-ови (енг. GAN), који је 2014 измислио Гудфелоу (енг. Goodfellow). Модел скажи за генерисање нових примера на основу података на којима је тренирао, нпр. да направи слику људског лица. У почетку, модел није успео да генерише добре примере, али додавањем више података и истраживањем нових архитектура могу се постићи добре перформансе.

Највећи фактор у успеху модела јесте архитектура модела и квалитет и количина података. Што се архитектуре тиче, није битно само додати још слојева или повећати број неурона. Архитектура се одлучује на основу проблема који се решава, нпр. веома познати АлексНет има 60 милиона параметара, али се показао лошије над МНИСТ подацима него наш модел који има само 3242 параметара, али зато је много бољи на подацима са милион примерака и 1000 класа. Може се рећи да је прављење модела више истраживање архитектура и обрада података него програмирање.

Такође, не можемо од модела очекивати да ће пронаћи шаблоне на било којим подацима који му дамо. Пре тренирања подаци морају бити обрађени на тај начин да је лако моделу да извуче корисне карактеристике, јер је то највећи услов за успех. То не мора да значи да подаци не треба да имају неку врсту шумова; као што смо видели модел може бити преварен ако му за улаз дамо слику са шумовима а да он није истрениран да их препозна. Чак је и препоручљиво да се примери модификују пре тренирања тако да садрже одређене шумове које можемо да очекујемо из реалног света, али да притом пример не губи своје карактеристике.

## 5. Ендноте

1. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 96-101, <https://www.deeplearningbook.org/contents/ml.html>
2. Foote, K. (2019), „A Brief History of Machine Learning”, *Dataversity*, <https://www.dataversity.net/a-brief-history-of-machine-learning/>
3. Foote, K. (2019), „A Brief History of Machine Learning”, *Dataversity*, <https://www.dataversity.net/a-brief-history-of-machine-learning/>
4. Brownlee, J. (2019), „A Gentle Introduction to the ImageNet Challenge”, *Machine Learning Mastery*, <https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/>
5. Stecanella, B. (2017), „An Introduction to Support Vector Machines (SVM)”, *MonkeyLearn*, <https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/>
6. Panchal, S. (2017), „k Nearest Neighbor Classifier (kNN) - Machine Learning Algorithms”, <https://medium.com/@equipintelligence/k-nearest-neighbor-classifier-knn-machine-learning-algorithms-ed62feb86582>
7. Greenacre, M. (2008), „Measures of Distance Between Samples”, <http://econ.upf.edu/~michael/stanford/maeb4.pdf>
8. Brid, R. (2018), „Decision Trees - A simple way to visualize a decision trees”, *GreyAtom*, <https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-tree-dc506a403aeb>
9. Zhou, V. (2019), „A Simple Explanation of Gini Impurity”, <https://victorzhou.com/blog/gini-impurity/>
10. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp 105-108, <https://www.deeplearningbook.org/contents/ml.html>
11. Pant, A. (2019), „Introduction to Linear Regression and Polynomial Regression”, *Towards Data Science*, <https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb>
12. Mehlig, B., „Artificial Neural Network”, <https://arxiv.org/pdf/1901.05639.pdf>
13. Jaksa, R., Katrak, M. (2015), „Neural Network Model of the Backpropagation Algorithm”, pp. 1-3, <https://pdfs.semanticscholar.org/f7de/25eb027083d4e7144c1ef6831baff35d6d06.pdf>
14. Brownlee, J. (2019), „What is Deep Learning?”, <https://machinelearningmastery.com/what-is-deep-learning/>
15. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 367-369, <https://www.deeplearningbook.org/contents/rnn.html>
16. Surmenok, P. (2017), „Estimating an Optimal Learning Rate For a Deep Neural Network”, *Towards Data Science*, <https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0>
17. Brownlee, J. (2018), „Difference Between a Batch and an Epoch in a Neural Network”, *Machine Learning Mastery*, <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>
18. Weng, L. (2018), „Meta-Learning: Learning to Learn Fast”, *Lil’Log*, <https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html>
19. Torres, J. (2018), „Learning process of a neural network”, *Towards Data Science*, <https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7>
20. Sharma, A. (2017), „Understanding Activation Functions in Neural Networks”, <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0>
21. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 224-225, <https://www.deeplearningbook.org/contents/regularization.html>
22. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, <https://www.deeplearningbook.org/contents/regularization.html>
23. Jordan, J. (2018), „Normalizing your data (specifically, input and batch normalization)”, <https://www.jeremyjordan.me/batch-normalization/>
24. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 200-211, <https://www.deeplearningbook.org/contents/mlp.html>
25. De Luca, G. (2020), „Bias in Neural Network”, *Baeldung*, <https://www.baeldung.com/cs/neural-networks-bias>
26. Kathuria, A., (2018), cIntro to optimization in deep learning: Gradient Descent”, *Paperspace*, <https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/>
27. Goodfellow, I., Bengio, Y., Courville, A., (2016), *Deep Learning*, pp. 326-335, <https://www.deeplearningbook.org/contents/convnets.html>
28. Powell, V., (2015), „Image Kernels explained visually”, <https://setosa.io/ev/image-kernels/>
29. Jeong, J. (2019), „The Most Intuitive and Easiest Guide for Convolutional Neural Network”, *Towards Data Science*, <https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480>
30. Olah, C. (2014), „Visualizing MNIST: An Exploration of Dimensionality Reduction”, <https://colah.github.io/posts/2014-10-Visualizing-MNIST/>

31. Jain, A. (2019), „Breaking neural networks with adversarial attacks”, *Towards Data Science*,  
<https://towardsdatascience.com/breaking-neural-networks-with-adversarial-attacks-f4290a9a45aa>
32. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., (1998), „Gradient-Based Learning Applied to Document Recognition”, <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>
33. Krizhevsky, A., Sutskever, I., Hinton, G., (2012), „ImageNet Classification with Deep Convolutional Neural Networks”, <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
34. Simonyan, K., Zisserman, A. (2015), „Very Deep Convolutional Networks For Large-Scale Image Recognition”  
<https://arxiv.org/pdf/1409.1556.pdf>
35. Goodfellow, I., Bulatov, Y., Ibarz, J., Arnoud, S., Shet, V. (2012), „Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks”,  
<https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf>