УНИВЕРЗИТЕТ У НИШУ
ЕЛЕКТРОНСКИ ФАКУЛТЕТ

Емпиријски приступ избору оптималне архитектуре трослојне вештачке неуронске мреже за класификацију података

Завршни рад

Студијски програм: Електротехника и рачунарство

Модул: Управљање системима

Студент:

Ведран Митић, бр. инд. 18775

Ментор:

Проф. др Мирослав Миловановић

Ниш, октобар 2025. година

Универзитет у Нишу
Електронски факултет

ЕМПИРИЈСКИ ПРИСТУП ИЗБОРУ ОПТИМАЛНЕ АРХИТЕКТУРЕ ТРОСЛОЈНЕ ВЕШТАЧКЕ НЕУРОНСКЕ МРЕЖЕ ЗА КЛАСИФИКАЦИЈУ ПОДАТАКА

AN EMPIRICAL APPROACH TO OPTIMAL THREE-LAYER ARTIFICIAL NEURAL NETWORK ARCHITECTURE SELECTION FOR DATA CLASSIFICATION

Дипломски рад

Студијски програм: Електротехника и рачунарство

Модул: Управљање системима

Студент: Ведран Митић, бр. инд. 18775

Ментор: Проф. др Мирослав Миловановић

Задатак: *Приказати начин рада мреже за класификацију података и утицај промене параметара на резултат класификације користећи алатке програма MatLab.*

Датум пријаве рада:

Датум предаје рада:

Датум одбране рада:

Комисија за оцену и одбрану:

1. Проф. др Мирослав Миловановић, Председник комисије

2. , Члан

3. , Члан

РЕЗИМЕ РАДА

Завршни рад бави се емпиријском анализом перформанси трослојне вештачке неуронске мреже за решавање проблема бинарне и вишекласне класификације података. Основни циљ рада био је да се, коришћењем алатки програма MATLAB, дефинише методологија за одабир оптималне архитектуре мреже и да се квантитативно измери утицај кључних параметара на резултат класификације.

У оквиру методологије, детаљно су истражени: број неурона у скривеном слоју – анализиран је утицај промене комплексности мреже на појаве *Overfitting*-а и *Underfitting*-а; оптимизациони алгоритам – спроведена је компаративна анализа две критичне *Backpropagation* методе (*Scaled-Conjugate Gradient* и Левенберг-Марквартовог алгоритма); квалитет скупа података – перформансе су тестиране кроз три студије случаја са различитим нивоима нелинеарности и корелације.

Најважнији доприноси и закључци рада су: доминација Лавенберг-Марквартовог алгоритма – емпиријски је доказано да Лавенберг-Марквартов алгоритам постиже боље резултате у односу на *SCG*, смањујући проценат грешке теста и повећавајући сигурност модела при класификацији сложенијих података; правило *Overfitting*-а: показало се да повећање броја неурона у скривеном слоју изнад оптималног (у већини симулација 10 неурона) доводи до појаве *Overfitting*-а, што доводи до погоршања резултата на тест скупу, иако се грешка тренинга смањује; идентификација *Irreducible Error*-а – у симулацији са скупом података ниске корелације и квалитета, утврђено је постојање *Irreducible Error*-а (грешке шума) као последице ограничења модела скупом података.

Добијени резултати пружају практичну основу за избор архитектуре и алгоритма при решавању детерминистичких проблема класификације, посебно у областима које захтевају високу поузданост, уз јасно наглашавање критичне улоге квалитета улазних података.

ABSTRACT

This final thesis presents an empirical analysis of the performance of a three-layer artificial neural network (ANN) in solving binary and multi-class data classification problems. The main objective of the thesis was to use MATLAB tools to define a methodology for selecting the optimal network architecture and to quantitatively measure the impact of key parameters on the classification result.

Within the framework of the methodology, the following were investigated in detail: the number of neurons in the hidden layer – the impact of changing network complexity on the occurrences of Overfitting and Underfitting was analyzed; the optimization algorithm – a comparative analysis of two critical Backpropagation methods (Scaled-Conjugate Gradient and Levenberg-Marquardt algorithm) was conducted; the quality of the dataset – performance was tested through three case studies with varying levels of non-linearity and correlation.

The most important contributions and conclusions of the thesis are: the dominance of the Levenberg-Marquardt algorithm – it was empirically proven that the Levenberg-Marquardt algorithm achieves better results compared to SCG, reducing the test error percentage and increasing model confidence when classifying more complex data; the rule of Overfitting – it was shown that increasing the number of neurons in the hidden layer above the optimal number (10 neurons in most simulations) leads to the occurrence of Overfitting, which results in the deterioration of results on the test set, even though the training error decreases; the identification of Irreducible Error – in a simulation with a low correlation and quality dataset, the existence of Irreducible Error (noise error) was established as a consequence of the model being limited by the dataset itself.

The obtained results provide a practical basis for selecting the architecture and algorithm when solving deterministic classification problems, especially in fields that require high reliability, while clearly emphasizing the critical role of the quality of the input data.

Садржај

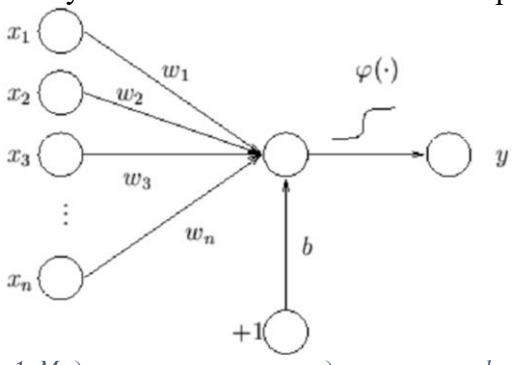
1. УВОД

Вештачка интелигенција (ВИ) данас представља један од најдинамичнијих и најпопуларнијих праваца у индустрији електронике и рачунарства. Машинско учење (Machine Learning), као њена кључна област, и дубоко учење (Deep Learning), као њена подгрупа, захтевају темељно разумевање неуронских мрежа. Концепт неуронске мреже није нов; његови корени сежу деценијама уназад као идеја модела који опонаша биолошки неурон и људски мозак, односно људско размишљање и интелигенцију. Савремени напредак у ВИ, посебно у дубоком учењу, умногоме је омогућен управо напредним конфигурацијама и архитектурама неуронских мрежа. Иако постоје и друге методе у машинском учењу, рад са сложеним подацима често зависи од примене неког облика неуронске мреже. Као што код људског неурона имамо синапсе, тела ћелије, дентрите, тако и код модела вештачког неурона имамо еквиваленте од којих сваки има своју сврху.

Први модел неурона је развијен у ратном и послератном периоду Другог светског рата, у доба када и почиње сама замисао и изучавање вештачке интелигенције у ратне сврхе. Као резултат студије под надзором *DARPA* (*Defence Advanced Research Projects Agency*) пројекта у САД, следећи закључак је понуђен и опште прихваћен: *Неуронске мреже обезбеђују значајне предности при решавању проблема процесирања који захтевају рад у реалном времену и интерпретацију односа између променљивих у вишедимензионалним просторима*. [1] И то све у доба када су данашња процесорска брзина и меморија биле незамисливе.

1.1. Перцептрон

Десетак година касније, 1958. јавља се први модел перцептрона који и дан данас представља основну јединицу неуронских мрежа. Он је једноставне архитектуре, састоји се од: улазних линија (x_i) од којих свака има своји тежински коефицијент (w_i), суматора који сумира производ улаза и тежина, активациони праг (b), активационе функције (f_{act}) која процесира добијену суму и излаза (y). Перцептрон користи простирање сигнала у једном смеру (*„feedforward”*) и примењује се у решавањима линарно сепарабилних проблема. Основни начин учења перцептрона је поређење добијеног излаза са стварим (жељеним) излазом и мењањем тежина и прага мреже на основу добијене грешке. Најпопуларнији пример таквог учења се назива *Roseblatt*-ово правило учења.

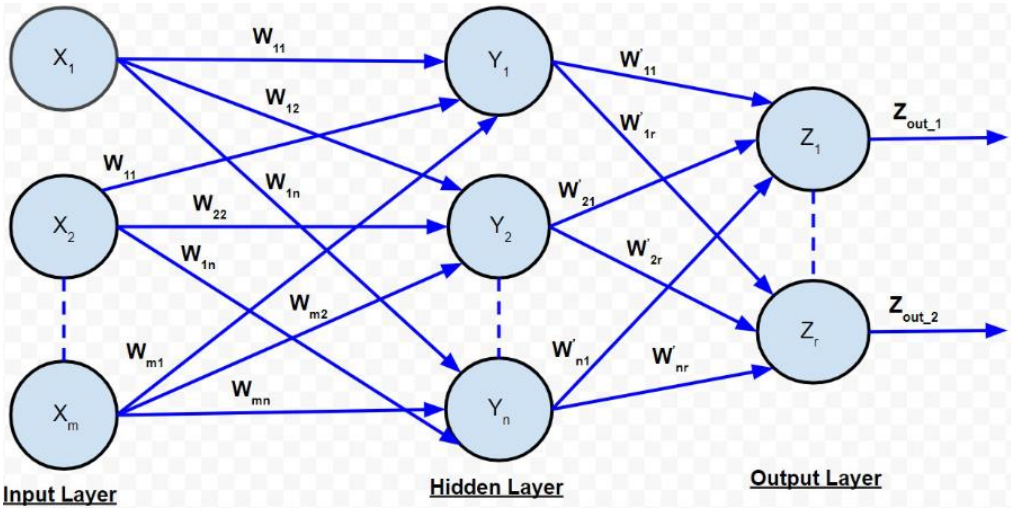


Слика 1. Модел перцептрона сигмоид активационе функције

1.2. Вишеслојне неуронске мреже

Како нас перцептрон ограничава на решавање линераних проблема и класификацију бинарних излаза (0 или 1), за решавање нелинеарности и вишегрупну класификацију сложенијих података, дошло је до комбинације више перцептрона и формирања неуронских мрежа. Свака неуронска мрежа састоји се од улазног слоја, скривеног слоја и излазног слоја, са тим да је број скривених слојева могуће повећавати у зависности од комплексности и захтева самог проблема. [2] Као и код самосталног перцептрона и овде се подаци пропагирају унапред:

- улазни подаци се доводе у неуроне улазног слоја кроз које се само дистрибуирају даље до неурона скривеног слоја. Број неурона у улазном слоју једнак је броју улазних података сваког од узорака. На пример можемо имати 10.000 узорака од којих сваки носи 5 различитих информација што значи да је број улазних неурона 10.
- скривени слој врши математичку обраду дефинисану активационом функцијом суме података и тежина веза. Овај део се може посматрати као *Black Box* где је готово немогуће пратити процесе који се догађају.
- на крају се обрађени подаци прослеђују излазном слоју који их обрађује у складу са врстом задатка, односно са бојем излазних класа којима треба придружити сваки од узорака. Бинарни проблем захтева две излазне класе и један излазни неурон, док је за *x* број класа потребно *x* излазних неурона и примењивање специфичних активационих функција као нпр. *Softmax*.



Слика 2. Архитектура трослојне неуронске мреже

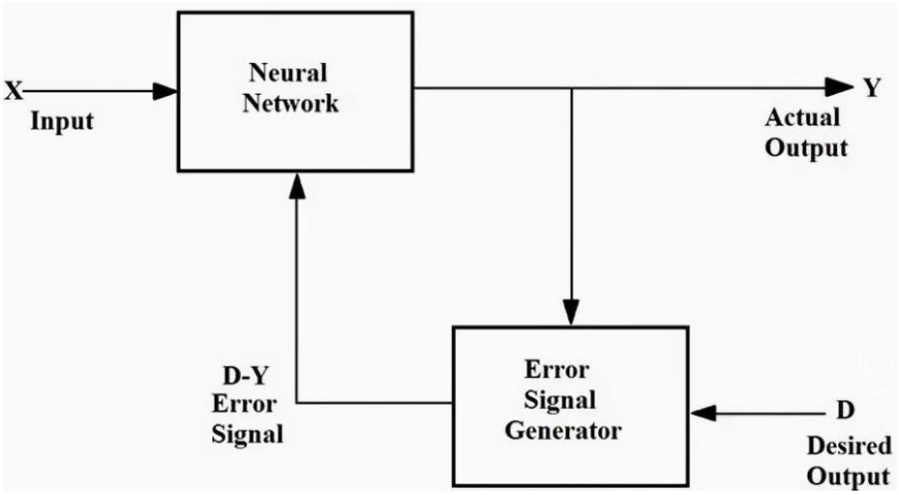
Поред мрежа које пропагирају информацију унапред, постоје и мреже са повратном спрегом (рекурентне). Ове мреже се разликују по томе што су динамичније због могућности меморисања података.

1.3. Алгоритми учења

Сваки процес учења, проблем и архитектура мреже захтевају пажљив одабир алгоритма учења. Алгоритми учења су процедуре које итеративно одређују тежине веза у мрежи тако што их прилагођавају на основу мерене грешке. Одабир алгоритма је један од најбитнијих задатака при раду, јер добијање жељених резултата често може зависити управо од њега.

Основна подела алгоритама је на *offline* и *online* алгоритме учења. *Offline* учење прима цео сет података (*batch*) пре модификације тежина. Мрежа пролази кроз цео скуп података, обрађује их, израчунава агрегатну грешку, и тек након тога једанпут модификује све тежине веза. Један такав пролаз кроз цео скуп података назива се епоха. Код *online* учења, мрежа обрађује по један улазни узорак, генерише излаз, израчунава грешку и одмах модификује тежине. Ово омогућава мрежи да учи и током саме реалне употребе (*real-time*), прилагођавајући се сваком новом податку који јој се доведе.

Учење се још може подетили и по начину обраде излазних информација на ненадгледано и надгледано. Ненадгледано учење подразумева довођење улазних података на мрежу без спољашње помоћи при обради тих података и процесу учења. То значи да систем ни у једном тренутку нема жељене излазне вредности, а самим тим ни вредности грешке на основу којих би могао да врши подешавање својих параметара. Мрежа идентификује податке са сличним карактеристикама и класификује их по наученим категоријама, користећи самоорганизујуће мреже. Са друге стране надгледано учење захтева и жељене излазне податке на основу којих се креира грешка. Циљ је модификовати тежине тако да мрежа генерише жељене излазе за доведене тренинг улазе.

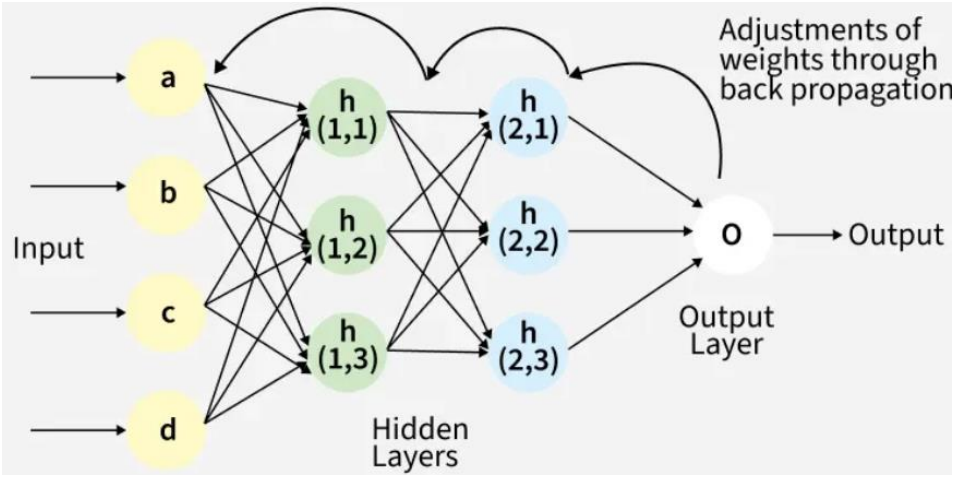


Слика 3. Блок дијаграм надгледаног учења

1.4. Backpropagation

Један од најзаступљенијих алгоритама учења назива се *Backpropagation* и спада у тип *offline* надгледаног учења. Овај дипломски рад ће се управо заснивати на *Backpropagation* алгоритму.

„ВР мрежа користи градијентни поступак при обучавању, који је аналоган процесу минимизације грешке. То значи да учи пресликавања из улазног простора узорка у излазни простор, кроз процес минимизације грешке између актуелног излаза који је остварила мрежа и захтеваног излаза, на основу скупа обучавајућих парова, односно примера. Процес учења почиње са презентацијом улазног облика узорка ВР мрежи, који простирањем кроз мрежу остварује излазни облик. ВР мрежа затим примењује генерализано делта правило да би се утврдила грешка на излазу, коју простирањем уназад преко скривеног слоја, користи за „лагано" модификовање сваког тежинског односа између неурона, што се понавља за сваки нови узорак. Генерализано делта правило обезбеђује конвергенцију процеса учења до задатог нивоа тачности кроз итеративни процес адаптације тежинских односа." [4]

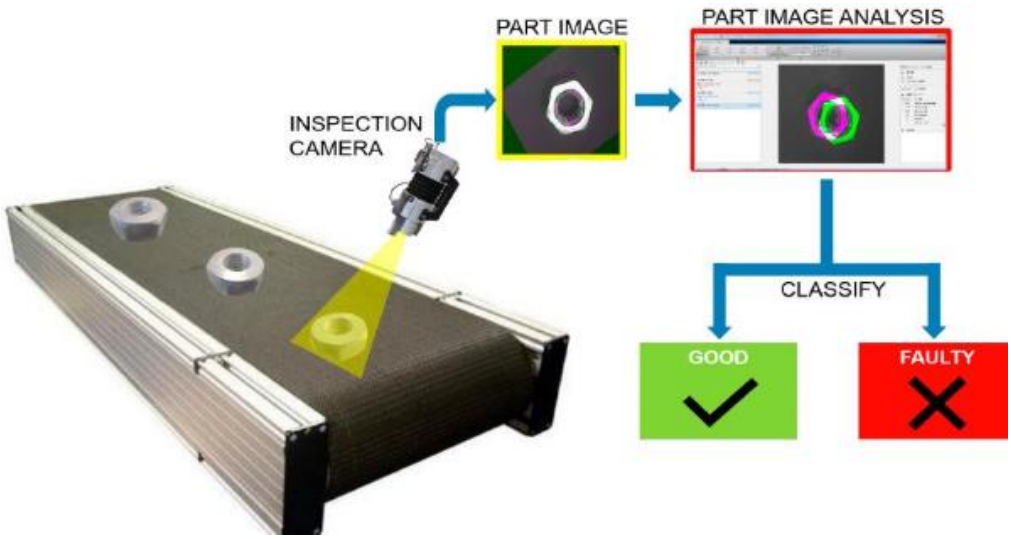


Слика 4. Шема Backpropagation алгоритма

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Препознавање образаца

Pattern recognition или препознавање образаца је проблем (процес) чијим се решавањем неуронска мрежа оспособљава (учи) за решавање разних задатака, попут сегментације слика, компјутерског вида, препознавања гласа, медицинских дијагноза, идентификација отисака прстију, итд. Може се рећи да је све од наведеног неки вид класификације која функционише на тај начин што открива нелинеарне везе и шаблоне између података чиме се обезбеђује мапирање сваког од улазних узорака за једану од излазних класа. Крајњи циљ оваквог тренинга је добијање модела који поседује високу способност генерализације, односно модела који ће бити способан да добије високу тачност класификације за неке нове тест податке које модел пре није видео. Наравно, у зависности од класификационог проблема, зависи и архитектура саме мреже. Математички можемо рећи да се за одређени улазни узорак тражи којој од k категорија он припада, односно тражимо функцију $f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$. [3]



Слика 5. Пример употребе Pattern recognition-а у индустрији

Архитектура која ће се користити у овом раду, за унапред срећене нумеричке примере, је трослојна неуронска мрежа са сигмоидном и *Softmax* функцијом које се користе у скривеном и излазном слоју респективно.

2.1.1. Сигмоидна функција

Сигмоидална функција (или Сигмоид) је математичка функција која има карактеристичну криву у облику латиничног слова *„S”*. Њен домен обухвата све реалне бројеве, а повратна вредност се монотоно повећава. Једна од најважнијих особина јесте то што је њен излаз увек ограничен у опсегу од 0 до 1, што је корисно у статистици као кумулативна функција расподеле. Сигмоидна функција је ограничена, диференцијабилна (има извод у свакој тачки), реална функција и има ненегативни извод. Улога ове функције је управо у омогућавању својства нелинеарности, односно решавању проблема који нису линеарно сепарабилни, као на пример проблем класификације. Њена диференцијабилност има велику улогу у *Backpropagation* алгоритму, јер омогућава ефикасно израчунавање градијента грешке који је неопходан за ажурирање тежина. Сигмоидална функција је дефинисана једначином:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

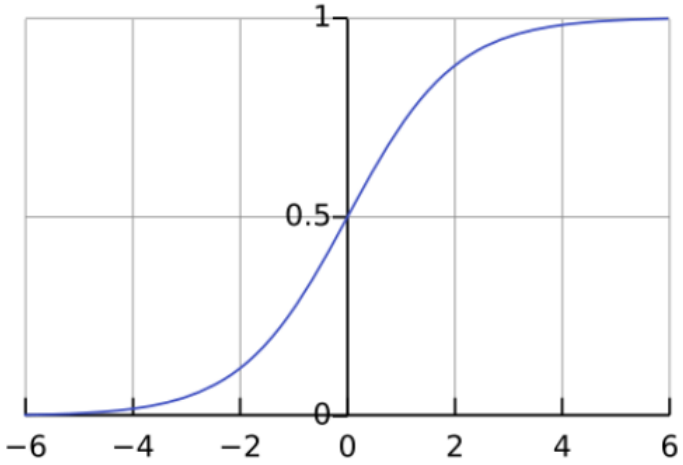
Извод ове функције се израчунава као:

$$\frac{d\,\sigma(x)}{d\,x} = \sigma(x) \cdot (1 - \sigma(x)) \tag{2}$$

Где се за неурон m, x може записати као сума свих производа улаза и њихових тежина у тај неурон, односно:

$$x_m(k) = \sum_{i=1}^{n} x_i(k) \cdot w_{im}\,(k) - b_m \tag{3}$$

При чему је:
n – број неурона у неком слоју
b_m – праг осетљивости за неурон m.



Слика 6. График сигмоидне функције

2.1.2. *Softmax* функција

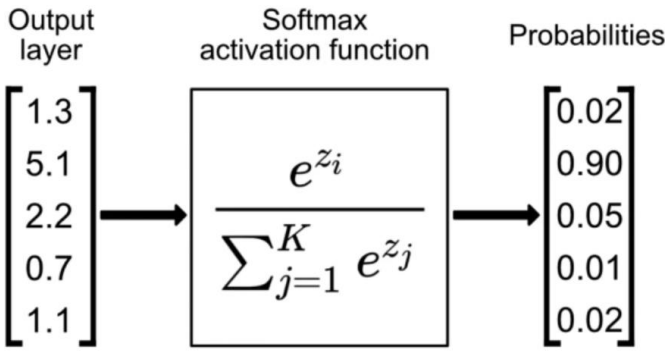
Због своје функционалности, ова функција је најчешћи избор за активациону функцију излазног слоја неуронске мреже за вишекласну класификацију. Како доведени улази (улазне суме слоја) представљају само сирове податке, реалне бројеве који не пружају директно неке смислене информације, *Softmax* функција их обрађује и претвара у вероватноће припадности свакој од излазних класа. Збир таквих излаза, вероватноћа је увек 1, а главни задатак функције је да, користећи експоненцијалну трансформацију, изврши нормализацију и ефективно истакне највећу вредност (потискујући остале ка нули), чиме класа са највећом вероватноћом постаје предвиђена класа. Знајући ово, у зависности од добијених излаза (вероватноћа), можемо и проценити (не)сигурност саме мреже при класификацији. На пример, уколико за један узорак добијемо излазе 0,30, 0,33, 0,37 – мрежа ће тај узорак класификовати под последњом класом (са вероватноћом од 37%), што не мора да представља тачан избор. Док уколико за узорак имамо излазе 0,1, 0,75, 0,15 – мрежа је врло вероватно успешно класификовала узорак у другу класу (са вероватноћом од 75%). *Softmax* функција дефинисана је једначином:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4}$$

Где је:

z_i – излаз претходног слоја мреже за *i*-ту класу, често називан и логит (*Logit*),

K – број класа.



Слика 7. Пример улоге Softmax функције у претварању улаза у излазне вероватноће

2.1.3. Унакрсна ентропија

Након што надгледана неуронска мрежа генерише стварни излаз (у процесу *Feedforward*-а), најпре је потребно израчунати вредност грешке на излазу. Ова грешка се најпростије може добити као апсолутна разлика (*L1 loss*) жељеног и стварног излаза или као квадрат ове разлике (*Mean-square error*), али је тако добијени губитак непогодан за рачунање градијента при *Backpropagation*-у због проблема засићења (*Saturation problem*), као и за проблеме класификације код којих баратамо вероватноћама и предвиђамо припадност, а не континуалне нумеричке вредности. На пример, ове функције губитка не могу да креирају довољно висок градијент када је наша предикиција са сигурношћу погрешна.

Решавање ових недостатака решавамо увођењем Унакрсне ентропије (*Cross-entropy loss*). Ова функција губитка пореди предвиђену расподелу вероватноће, добијену Softmax функцијом, са стварном расподелом вероватноће и кажњава мрежу уколико је предвиђена вероватноћа за тачну класу ниска. Тиме се заправо процењује перформанса модела. За разлику од једноставне квадратне грешке (*MSE*), унакрсна ентропија је посебно осетљива на нетачна предвиђања са високом сигурношћу. Ако мрежа предвиди погрешну класу са веома високом вероватноћом (нпр. 99%), функција губитка ће казнити мрежу знатно већом вредношћу грешке, што ће приморати алгоритам за учење (*Backpropagation*) да брзо коригује тежине. Тиме се мрежа снажније подстиче да исправи своју грешку током процеса тренирања, што доводи до брже и поузданије конвергенције.

Математички, Унакрсна ентропија се може изразити формулом:

$$L = -\sum_{i=1}^{K} y_i \cdot \log_2(a_i) \tag{5}$$

Где је:

K – укупан број класа,

y_i – стварна вредност вероватноће за класу i,

a_i – предвиђена вероватноћа за класу i, излаз *Softmax* функције.

Ако заменимо a_i фомулом (4), имамо:

$$L = -\sum_{i=1}^{K} y_i \cdot \log_2\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right) \tag{6}$$

У пракси, због *One-hot* енкодирања (Поглавље МЕТОДОЛОГИЈА), у суми ће само једна класа (тачна) имати стварну вредност вероватноће 1, а све остале класе (нетачне) 0. Па ће се формула (5) свести на:

$$L = -\log_2(a_p) \tag{7}$$

Где је a_p предвиђена вероватноћа за тачну класу p. Што значи да ће вредност губитка зависити само од вероватноће коју је мрежа доделила тачној класи. Из формуле видимо да уколико је додељена вероватноћа велика (близу 1), губитак ће бити мали, док уколико је вероватноћа мала (близу 0), губитак ће бити велики.

Пример 1.

Уколико за класификацију у две класе *x* и *y*, где је тачна класа *x*, имамо добијено предвиђање:

x = 0.1

y = 0.9

Видимо да је модел погрешио у класификацији, изабрао је класу **y** са веома великим процентом сигурности (90%). Применивши формулу (7), добијамо:

$$L = 2.302$$

Што је веома висока вредност казне.

Пример 2.

Уколико за за исти класификациони проблем имамо добијено предвиђање:

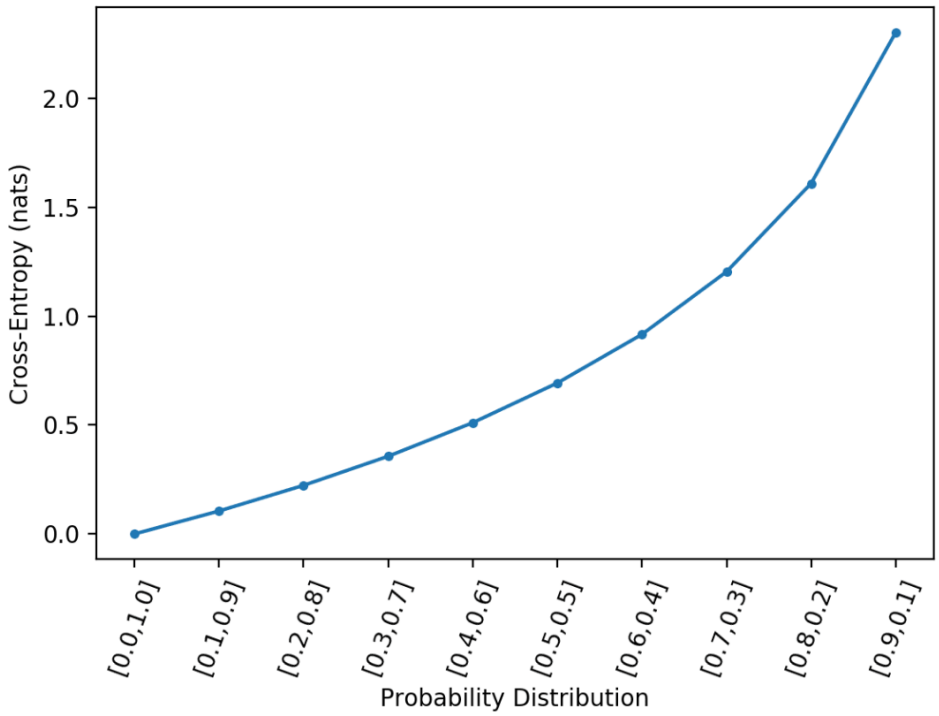
x = 0.55

y = 0.45

Видимо да је модел поново погрешио у класификацији, али са мањим процентом сигруности (55%). Сада добијамо:

$$L = 0.6$$

Овога пута „казна" је много мања, па закључујемо да је наш циљ да заправо смањимо сигурност система у погрешне одлуке, односно повећамо његову сигурност у тачне. Промену вредности казне у односу на вредност добијене вероватноће можемо видети на *Слици 8*. Због ове особине и њене математичке везе са *Softmax* функцијом, Унакрсна ентропија представља најефикаснији начин за мерење грешке у задацима класификације.



Слика 8. График зависности вредости губитка Унакрсне ентропије у односу на вредност расподеле вероватноће

2.2. Градијент грешке

Након што је вредност губитка (L) израчуната на излазном слоју помоћу Унакрсне ентропије, следећи кључни корак је да се одреди како и колико треба променити тежине у мрежи да би се тај губитак смањио. Ово се постиже израчунавањем градијента који представља основни концепт математичке оптимизације. Уопштено говорећи, градијент вишеваријабилне функције је вектор чији елементи представљају прве изводе те функције. Он показује смер најбржег раста функције и код неуронских мрежа користимо га у склопу оптимизационог алгоритма првог реда – Опадајућег градијента (*Gradient descent*), који налази локални минимум функције тако што се креће у супротном смеру (уназад).

То постижемо рачунањем градијента грешке (δ) [19] који представља локалну вредност израчунату посебно за сваки неурон у мрежи током *Backpropagation*-а. Он се може и посматрати као допринос сваког од неурона укупној грешци мреже. Као што је речено у подпоглављу *Backpropagation*, грешка се простире уназад кроз мрежу почевши од излазног слоја, где се директно повезује са вредностима добијеним Унакрсном ентропијом, ка улазним слојевима, притом коригујући вредности тежинских фактора. Градијент грешке се математичком формулом дефинише као:

$$\delta_p = \frac{dy_p}{dX_p} \cdot e_p \tag{8}$$

Где је:

δ_p – градијент грешке неурона p,
y_p – излаз неурона p,
X_p – улаз неурона p,
$\frac{dy_p}{dX_p}$ – извод активационе функције у односу на улаз неурона
e_p – грешка на излазу неурона p.

За излазни слој где за рачунање грешке користимо Унакрсну ентропију, грешка на излазу неурона p је извод функције губитака у односу на предвиђену вероватноћу $e_p = \frac{dL}{da_p}$, улаз излазног неурона је логит $X_p = z_p$, а излаз неурона је излаз *Softmax* функције $y_p = a_j$. Па на основу тога формула (8) постаје:

$$\delta_p = \sum_{j=1}^{K} \frac{dL}{da_j} \cdot \frac{da_j}{dz_p} \tag{9}$$

Како у мрежи можемо имати два случаја, када је узорак тачно класификован (j = p) и када је погрешно класификован (j ≠ p), формулу можемо разложити:

$$\delta_p = \left(\frac{dL}{da_p}\right) \cdot \left(\frac{da_p}{dz_p}\right) + \sum_{j \neq p} \frac{dL}{da_j} \cdot \frac{da_j}{dz_p} \tag{10}$$

Заменом формуле (5) у L и одређивањем извода по a_j добијамо:

$$\frac{dL}{da_j} = -\frac{y_j}{a_j} \tag{11}$$

Извод Softmax функције по z_p је:

$$\begin{aligned} \frac{da_p}{dz_p} &= a_p \cdot (1 - a_p), \qquad j = p \\ \frac{da_j}{dz_p} &= -a_j a_p, \qquad j \neq p \end{aligned} \tag{12}$$

Заменом (11) и (12) у (10) добијамо:

$$\begin{aligned} \delta_p &= \left(-\frac{y_p}{a_p}\right) \cdot \left(a_p(1 - a_p)\right) + \sum_{j \neq p} \left(-\frac{y_j}{a_j}\right) \cdot (-a_j a_p) \\ \delta_p &= -y_p \cdot (1 - a_p) + \sum_{j \neq p} y_j a_p \end{aligned} \tag{13}$$

Како је $y_j = 1$ само за једну тачну класу, а за све остале 0, постоје два случаја за неурон *p*:

Ако је неурон p тачна класа ($y_p = 1$), онда је $y_j = 0$, па је $\sum_{j \neq p} y_j a_p = 0$, одакле је $\delta_p = a_p - 1$, односно:

$$\delta_p = a_p - y_p \tag{14}$$

Ако је неурон p нетачна класа ($y_p = 0$), онда је $\sum_{j \neq p} y_j a_p = a_p$, одакле је $\delta_p = -0 + a_p$, односно:

$$\delta_p = a_p - y_p$$

Дакле видимо да се у сваком случају градијент грешке излазног слоја рачуна на исти начин и да увек представља разлику између предвиђене вероватноће припадности и реалне припадности.

Овако израчунат градијент пропагира се назад кроз мрежу коригујући тежине између неурона m у скривеном слоју и неурона p у излазном слоју по формули:

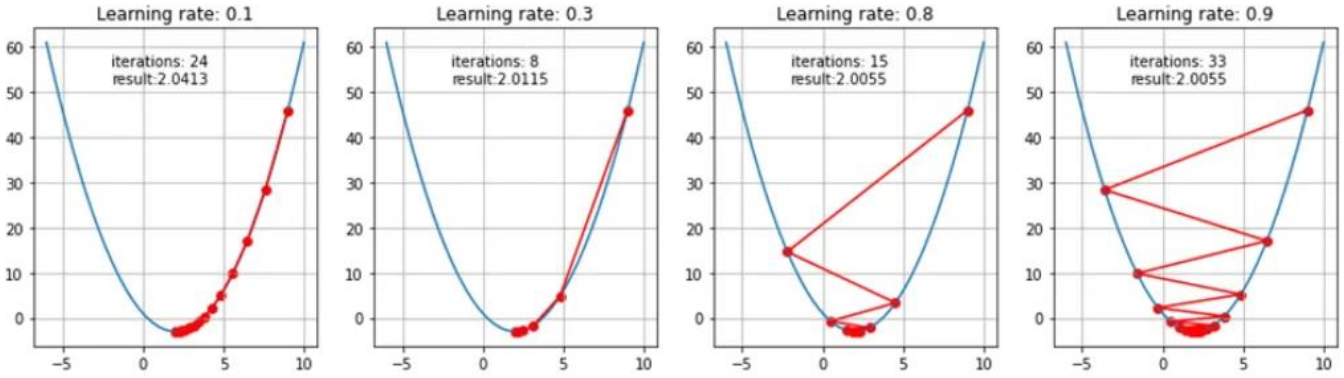
$$\Delta w_{mp}(k) = \eta \cdot y_m(k) \cdot \delta_p(k) \tag{15}$$

Где је:
$\Delta w_{mp}(k)$ – промена тежине неурона m у итерацији k,
η – стопа учења
$y_m(k)$ – излаз неурона m и улаз у неурон p у итерацији k,
$\delta_p(k)$ – градијент грешке излазног неурона p у итерацији k.

Па ће се тежине у наредној итерацији мењати по формули:

$$w_{mp}(k+1) = w_{mp}(k) + \Delta w_{mp}(k) \tag{16}$$

Стопа учења (η) је хиперпараметар који одређује величину корака при ажурирању тежина веза неуронске мреже, односно колико брзо модел учи приликом тренинга. Вредност стопе учења је обично између 0 и 1, а циљ је пронаћи и одабрати оптималну вредност која ће минимизирати функцију губитка и ефикасно доћи до глобалног минимума. У пракси обично се креће са већом вредношћу стопе учења и онда смањује до жељене. Висока стопа учења чини да модел прави веће кораке, тежине имају велике промене и тиме је тренинг бржи, али врло често изазива осциловање око минимума или дивергенцију од оптимума због чега модел заправо никада не конвергира и тренира узалуд. Са друге стране, ниска стопа учења је прецизнија и пружа више контроле, али чини тренинг споријим и може довести до тога да модел заглави у локалном минимуму. Оптимална стопа учења би требала да омогући моделу да превазиђе плитке локалне минимуме и седласте тачке, да време трајања тренирања не буде сувише дугачко и да се постигне најбоља конвергенција ка глобалном минимуму.



Слика 9. Утицај вредности стопе учења на конвегензију ка минимуму

Градијент грешке скривеног слоја по формули (8), који користи сигмоидну активациону функцију за неурон m, након коришћења формуле (2) биће:

$$\delta_m = y_m(k) \cdot [1 - y_m(k)] \cdot \sum_{i=1}^{K} \delta_i(k) \cdot w_{mi}(k) \tag{17}$$

Где је:

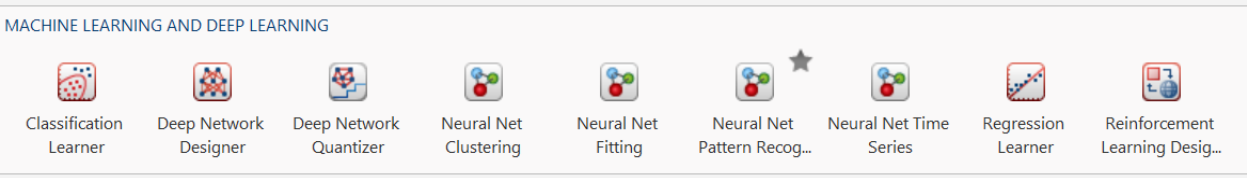
$y_m(k)$ – излаз неурона m и улаз у неурон p у итерацији k,

$\sum_{i=1}^{K} \delta_i(k) \cdot w_{mi}(k)$ – сума производа градијента грешке свих K излазних неурона и тежинских коефицијената веза које су повезане за сваки од њих, познатија као повратна грешка.

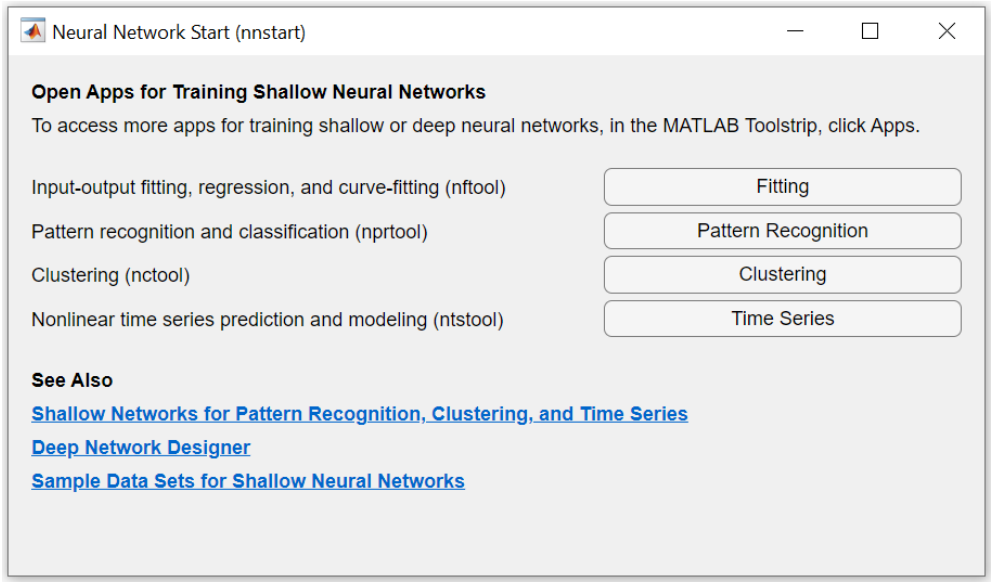
3. МЕТОДОЛОГИЈА

Без обзира на постојеће теоријске основе и принципе рада неуронских мрежа, једини прави показатељ и усмеравач при самом раду јесте тестирање мреже конфигурисањем њених особина за дати скуп података. Ове особине су најчешће бројеви неурона у скривеном слоју или тип активационе функције, али су и повремено сам број скривених слојева, вредност стопе учења, оптимизациони алгоритми, итд. Врло често, добивши жељене резултате једним моделом, након његовог експлоатисања и тестирања при другим скуповима података успех модела се не понавља или његови резултати и понашање бивају недолични. Зато је, сем саме поставке неуронске мреже, битан и рад са скуповима података који се у мрежу доводе. Јако је важно разумети повезнатост улазних и излазних података, односно колико су доведене улазне вредности на основу којих мрежа учи репрезентативне за учење. Јер уколико нису, мрежа ће тешко проналазити логичку повезаност и учити шумове, што ће довести до лоших резултата не само тестирања, већ и самог тренинга. Дакле модел неће бити адекватан за даље експлоатисање. Зато је потребно да пре уноса података у мрежу обавимо процес који се у машинском учењу назива *Data Preprocessing*.

У овом раду, за потребе рада са неуронским мрежама, коришћен је софтвер *MATLAB* и његов алат за рад са мрежама *Neural Network Toolbox*. Овом алату се може приступити или избором алата из опадајуће листе *APPS* или уносом команде *nnstart* у командном прозору, а затим одабиром *Neural Net Pattern Recognition*. За директан одабир *Neural Net Pattern Recognition*-а може се унети и команда *nprtool*. [18]

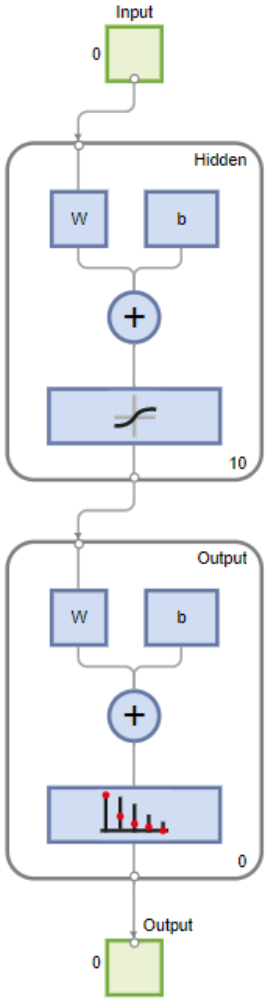


Слика 10. Опадајућа листа APPS



Слика 11. Прозор nnstart команде

Као резултат ће се аутоматски генерисати трослојна неуронска мрежа коју ћемо користити за тренинг и сликовно приказати на екрану.



Слика 12. Изглед трослојне неуронске мреже за класификацију

Најпре се може видети почетна репрезентација мреже без доведеног скупа података, где је број десет у скривеном слоју подразумевана почетна вредност броја неурона у *MATLAB*-у, али се она може мењати уписом другог броја у поље *Layer size* опадајућег менија. Такође се може и блоковски видети смер кретања улаза мреже (*Feedforward*), принцип суме (*Logits*) и одабране активационе функције у сваком од слојева (Сигмоид и *Softmax*). Притиском на дугме *Import* у горњем левом углу опадајућег менија и увезивањем улазних и излазних података у мрежу, бројеви неурона улазног и излазног слоја ће се променити у зависности од броја улазних параметара учења сваког од узорака и броја излазних класа.

3.1. Крос валидација

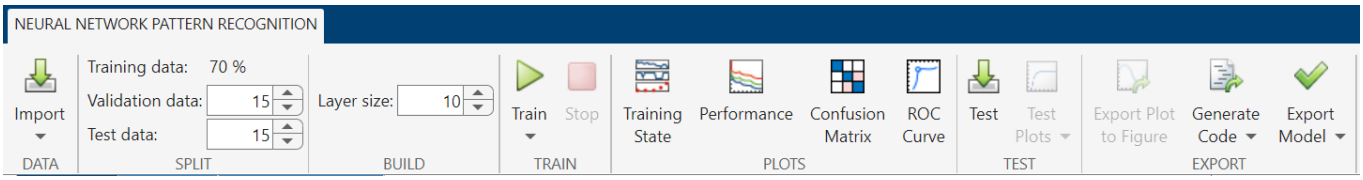
Крос-валидација (*Cross-Validation*) представља кључну методологију у машинском учењу која се користи за процену способности модела да се генерализује на независни скуп података. Њена основна сврха је избегавање проблема *Overfitting*-а и добијање поуздане процене грешке. Сваки скуп података је подељен на три подскупа за потребе *Cross-Validation*-а:

- Тренинг подаци – ово је највећи скуп података јер се управо мрежа најпре на њима тренира. Користе се за прорачун градијента и ажурирање тежинских коефицијената. Квалитет модела ће у највећој мери зависити од резултата овог скупа података јер се у односу на њега може говорити о *Overfitting*-у или *Underfitting*-у. Ниска успешност тренинга ће врло ретко (готово никада) довести до високе успешности теста, док висока успешност тренинга може предусловити *Overfitting*, али не и тиме нужно покварити успешност теста.
- Валидациони подаци – користе се за праћење перформанси тренинга, односно грешке на излазу сваке епохе. На овај начин се детектује почетак привикавања мреже на тренинг податаке на конто чега се зауставља сам процес учења. Уколико валидациона грешка из епохе у епоху расте, док тренинг грешка опада, процес учења ће се зауставити. У *MATLAB*-у се тренинг зауставља уколико за шест узастопних епоха валидациона грешка не дође до новог минимума. Дакле, уколико се у некој x епохи дође до до тада највећег минимума валидационе грешке, тренинг ће се зауставити у $x+6$-ој епохи. Број од шест епоха се у *MATLAB*-у може променити кодирањем. Са једне стране, овиме се спречава појављивање *overfitting*-а, али са друге, прерано заустављање може утицати на неефикасност тест података на коју *Overfitting* можда не би утицао (благи *Overfitting*).
- Тест подаци – овај скуп података је најбитнији и одређује успешност истренираног модела. Ови подаци најчешће представљају неки нови скуп података који до сада није виђен у тренингу и треба да процени генерализацију модела, да ли је он способан да се прилагоди и реши новонастали проблем класификације, односно да ли поседује интелигенцију, а не само учење напамет.

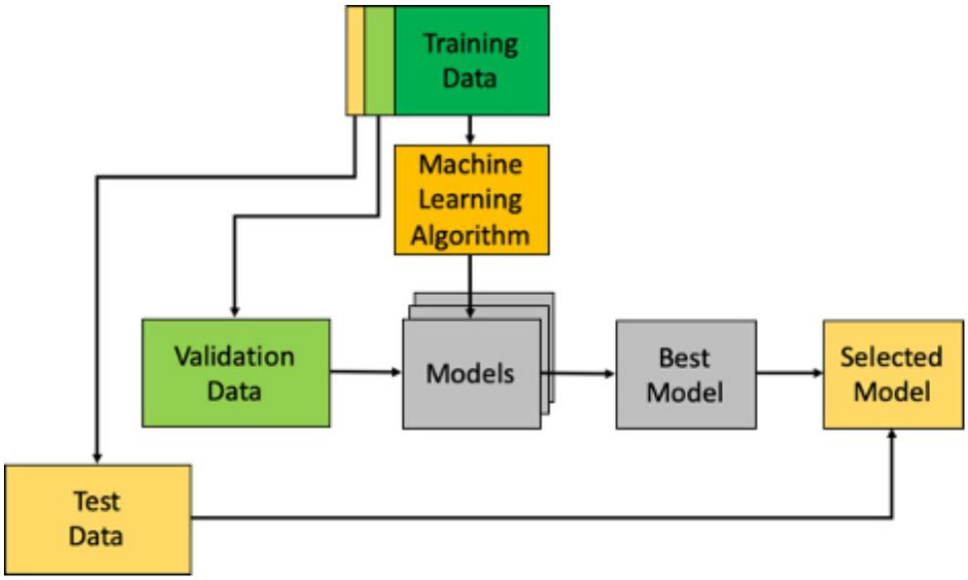
Расподела података је обично у мери 70% / 15% / 15% , за тренинг, валидационе и тест податке респективно. По овом принципу се на основу коначних резултата процењује способност генерализације неуронске мреже за дати скуп података. Уколико се тренинг не заустави услед достигнутог *Validation Check*-а, зауставиће се када достигне жељени градијент (често реда 10^{-6}), или, уколико оба услова нису испуњена након унапред одређеног броја епоха (у *MATLAB*-у по *deafoult*-у 1000).

Споменути појмови *Overfitting* и *Underfitting* означавају пренаучену и недовољно научену мрежу. Као што је већ речено, *Overfitting* настаје када модел превише добро запамти структуру и шум тренинг података, због чега губи способност генерализације и показује значајно лошије перформансе приликом тестирања на новим, до сада невидљивим подацима. Висок проценат успешности тренинг података, а ниска успешност тест података је класичан пример лошег утицаја *Overfitting*-а. Благи *Overfitting* ће имати свега

неколико процента између успешности тренинг и тест података. Низак проценат успешности тренинга, *Underfitting*, значи да мрежа није успела да разуме везе између улаза и излаза и интерпретира их као логичке образце, па даје низак проценат успешности и тренинг и тест података. То је најчешће последица лоше одрађеног *Data Preprocessing*-а, прејединоставне мреже или прераног заустављања тренинга.



Слика 13. Опадајући мени алата неуронских мрежа за препознавање образца



Слика 14. Блок дијаграм процесуирања скупа података при Cross-Validation-у

3.2. Преобрада података

Као што је већ истакнуто, пре него што су подаци увезени у неуронску мрежу, они су претходно ручно обрађени и припремљени (преобрађени). Ова фаза је кључна јер сирови подаци често садрже формате који нису компатибилни са математичким операцијама неуронске мреже. На пример, уколико имамо скуп података код кога ће улази у мрежу бити особине аутомобила, а излази типови квара који су се догодили код истог, међу улазним колонама можемо наћи оне које садрже назив модела, земља произвођача, боју возила, информације о возачу... Такве информације често нам нису параметри који би могли логички утицати на квар, па их можемо обрисати из скупа података. Међутим, ако одлучимо да нам је модел аутомобила битна карактеристика (јер се можда ради о моделима једне те исте фирме), такав улазни параметар би смо морали да трансформишемо у нумерички и то путем технике која се назива *One-Hot Encoding* (*OHE*). *OHE* подразумева да групу текстуалних података претворимо у матрице нула и јединица.

Пример 3. За колону *Models* постоје три могућности: *Model-1, Model-2* и *Model-3*. Примена OHE би изгледала тако што би направили нове посебне три колоне са називима *Model-1, Model-2* и *Model-3* и сваку кодирали са:

Models	Model-1	Model-2	Model-3
Model-1	1	0	0
Model-2	0	1	0
Model-3	0	0	1

Табела 1. Пример OHE за групу модела

Овако осигуравамо да мрежа не прави разлику у тежини података, већ да препознаје да ли је улаз активан или не. Употреба *OHE* се чешће користи за излазне класе јер су скупови података тако организовани да се различите врсте излазних класа (рецимо типови квара аутомобила) смештају у једну колону (рецимо *Failures*).

Поред процесуирања текстуалних података, јако је битно обрадити и нумеричке податке због неповезаности опсега вредности између различитих колона скупа података, као и због опсега активационих функција. Другим речима, потребно је скалирати ове податке на вредности између 0 и 1. Скалирање се врши примењивањем формуле:

$$X_{skalirano} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{18}$$

Где је X група података која се скалира.

Овакво скалирање се још назива и *Min-Max* нормализација, а *MATLAB* садржи функцију којом се наведени процес аутоматски извршава уколико радимо са мрежом путем кодирања.

Пример 4. Рецимо да за колону *RMP* која представља број обртаја мотора у минуту имамо вредности чији опсег иде од -2636 до 2886, скалирање ће изгледати на следећи начин:

RPM	RPM_{scaled}
1551	0,75824
1408	0,732343
1498	0,748642
1433	0,736871
1408	0,732343
1425	0,735422
1558	0,759507
1527	0,753894
1667	0,779247
1741	0,792648
1782	0,800072
1423	0,73506
1339	0,719848

Табела 2. Пример скалирања вредности броја обртаја мотора у минуту

3.3. Емпиријска метода

Овако припремљена неуронска мрежа са унешеним процесираним подацима је спремна за рад притиском на дугме *Train*. У зависности од величине мреже и обима скупа података, као и брзине рада процесора, време тренирања може трајати од секунде до неколико минута или сати. *MATLAB*-ов алат нам након тога пружа све нумеричке и графичке репрезентације добијених резултата и рада мреже. За један насумичан скуп података и мрежу од 10 неурона у скривеном слоју, то су:

- Резултати тренажног процеса – дати су у виду табеле која садржи иницијалне вредности, вредности након заустављања тренинга и циљане вредности епохе, времена тренирања, перформансе, градијента, *Validation Checks*-а. Вредност перформансе заправо представља вредност грешке Унакрсне ентропије након последње епохе.

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	15	1000
Elapsed Time	-	00:00:01	-
Performance	0.308	0.000249	0
Gradient	0.217	0.000701	1e-06
Validation Checks	0	6	6

Слика 15. Табела резултата тренажног процеса

- Резултати модела – под чиме се поред резултата тренинга подразумевају и резултати валидације и теста. У датом прозору можемо видети и информације о податицма скупа података и о коришћеном алгоритму.

Model Summary

Train a neural network to classify predictors into a set of classes.

Data

Predictors: wineInputs - [13x178 double]
Responses: wineTargets - [3x178 double]

wineInputs: double array of 178 observations with 13 features.
wineTargets: double array of 178 observations with 3 classes.

Algorithm

Data division: Random
Training algorithm: Scaled conjugate gradient
Performance: Cross-entropy error

Training Results

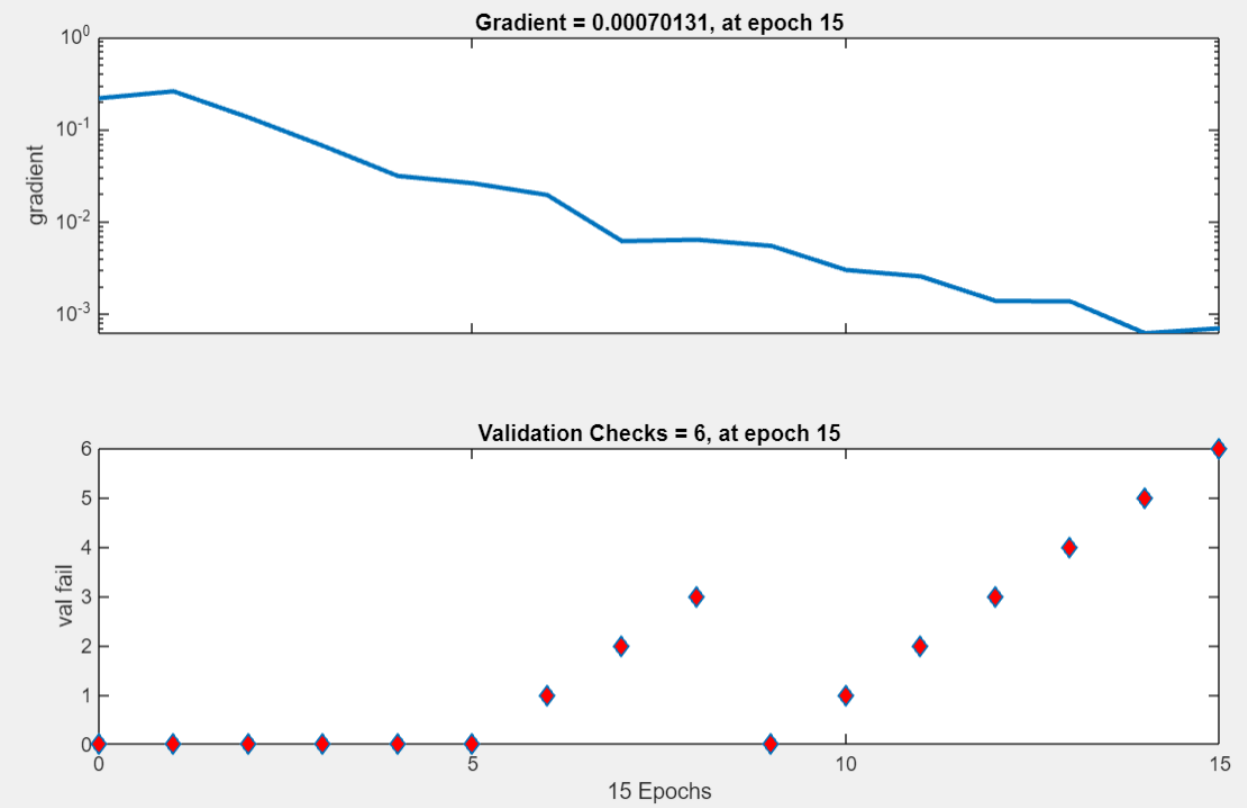
Training start time: 21-Oct-2025 13:01:06
Layer size: 10

	Observations	Cross-entropy	Error
Training	124	0.0025	0
Validation	27	0.0456	0.0370
Test	27	0.0050	0

Слика 16. Прозор резултата модела

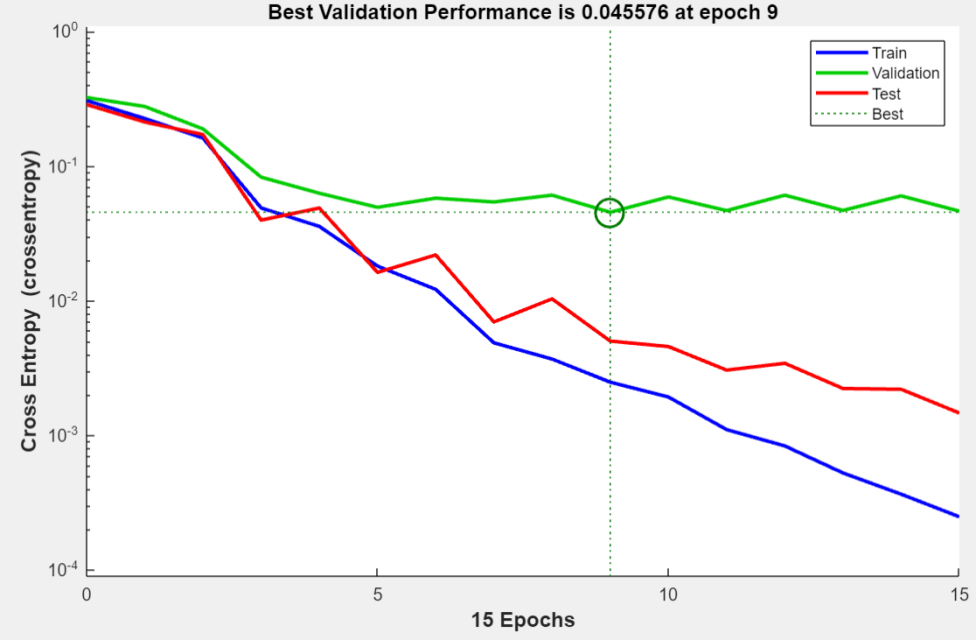
Вредност *Observations* представља број узорака узетих за сваку од група података, *Cross-entropy* вредност грешке у најбољој тачки валидације (*Best Validation Performance*), а *Error* грешку у односу на број узорака за дату групу. За дат пример, *Error* од 0,0370 при валидацији значи да је од 27 узорака, погрешно класификовано 3,7%, што је један узорак.

- *Training State Plot* – график који приказује промену вредности градијента и бројања *Validation Check*-а при тренингу кроз епохе.



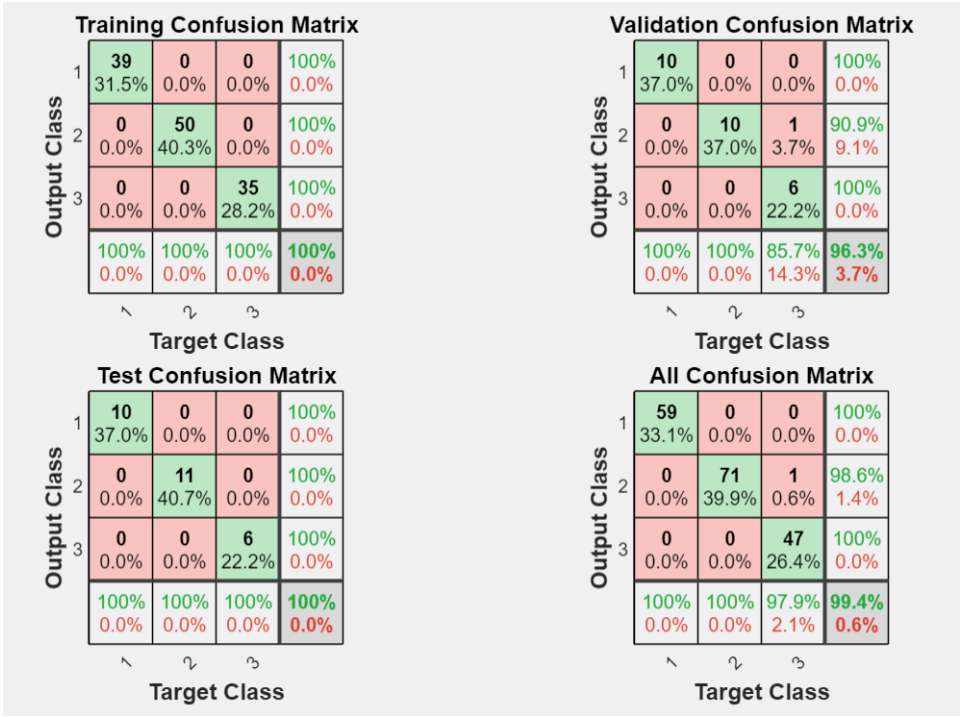
Слика 17. Графици промене градијента и бројања Validation Check-а

- График перформансе – јединствен график који обухвата промену грешке Унакрсне ентропије сва три скупа података (тренинг, валидација, тест) кроз епохе. Помоћу графика перформансе се могу пратити вредности грешке у свакој епохи појединачно и изводити закључци о брзини конвергенције модела, о тежини података (мисли се на то колико су подаци тешки за класификацију) који су додељени свакој од група, о присутности *Overfitting*-а, о квалитету рада мреже, итд. Такође је истакнута и тачка *Best Validation Performance* у којој је валидација достигла свој минимум. Може се приметити да је тренинг стао шест епоха након ове тачке (подпоглавље *Cross-Validation*). Без обзира што се ова тачка назива *„Best“*, не мора да значи да је мрежа у њој достигла своје најбоље перформансе, на *Слици 18.* се види да је грешка тренинг и тест података у истој епохи била за читав децимални ред мања. То је и један од разлога зашто је валидациони скуп података имао један погрешно класификован узорак.



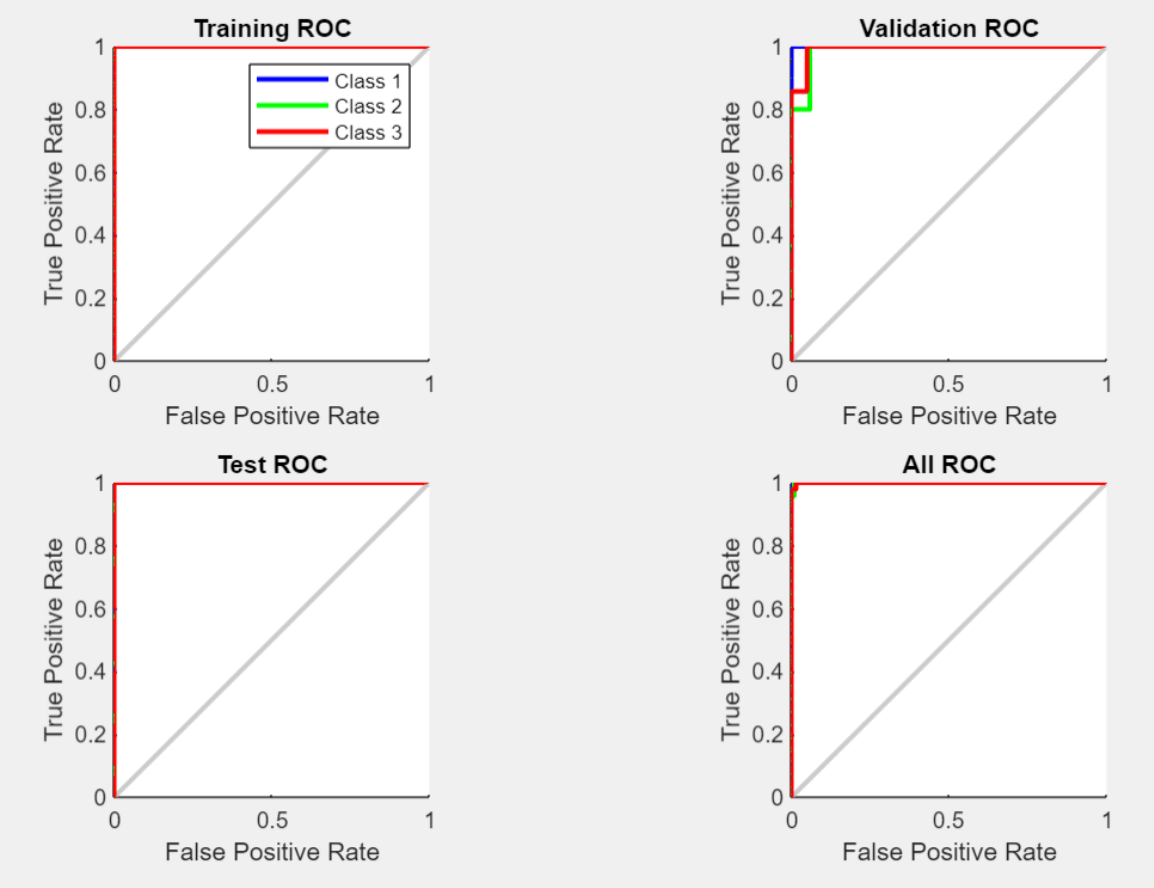
Слика 18. График перформансе модела

- Матрице конфузије – пружају најдиректнији преглед у резултате модела. Садрже проценат грешке и успешности тренинг, валидационих, тест, као и свеукупних података. По редовима су распоређени излазне класе у које је модел класификовао узорак (*Output class*), а по колонама класе у које би узорак требао бити класификован (*Target class*). Сем процентуално, у матрицама имамо увид и у тачне бројке класификације.



Слика 19. Матрице конфузије

Reciever Operating Characteristic (*ROC*) – је крива која графички приказује компромис између Истински позитивне стопе (*True Positive Rate* на *y*-оси) и Лажно позитивне стопе (*False Positive Rate* на *x*-оси), чиме се мери дијагностичка способност класификатора како се његов праг одлуке (*Threshold*) за сигурност класификације мења. Истински позитивна стопа представља проценат тачно квалификованих позитивних узорака, док Лажно позитивна оса представља проценат негативних узорака које је модел погрешно класификовао као позитивне.



Слика 20. Графици ROC криве

Емпиријска метода која ће се користити у овом раду састојаће се од десетоструког покретања мреже са неким *X* бројем неурона скривеног слоја, записивањем добијених резултата у мрежи, а затим понављање процеса за другачији број неурона. На крају ће се табеларно поредити просечне вредности ових тренинга са циљем да се одабере која од конфигурација даје најбоље резултате и може бити изабрана за наш модел који ће се експлоатисати. Поред оваквог приступа, у зависности од добијених резултата, а са циљем смањења грешке класификације, испробаће се и упоредити резултати добијени оптимизационом методом Левенберг-Маркварт (*Levenberg-Marquardt*) у односу на резултате добијене стандардним оптимизацоним методом *Scaled-Conjugate Gradient*.

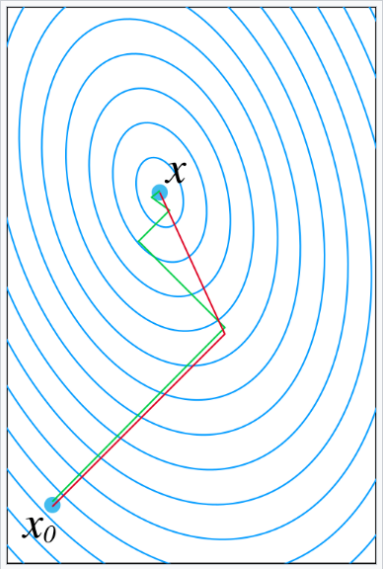
3.4. *Scaled-Conjugate Gradient* и Левенберг-Маркварт

Улога оптимизатора у неуронској мрежи је да пронађе најбољи сет тежинских коефицијената који ће минимизовати функцију губитка (Унакрсну ентропију) и тиме смањити грешку мреже. Као што је раније објашњено, *Backpropagation* алгоритам је универзални механизам за рачунање градијента грешке. Међутим, оптимизационе методе одлучују како да тај градијент искористе за промену тежина и проналажење локалног минимума. Одабир оптимизатора је критичан јер директно утиче на брзину конвергенције и квалитет пронађеног минимума.

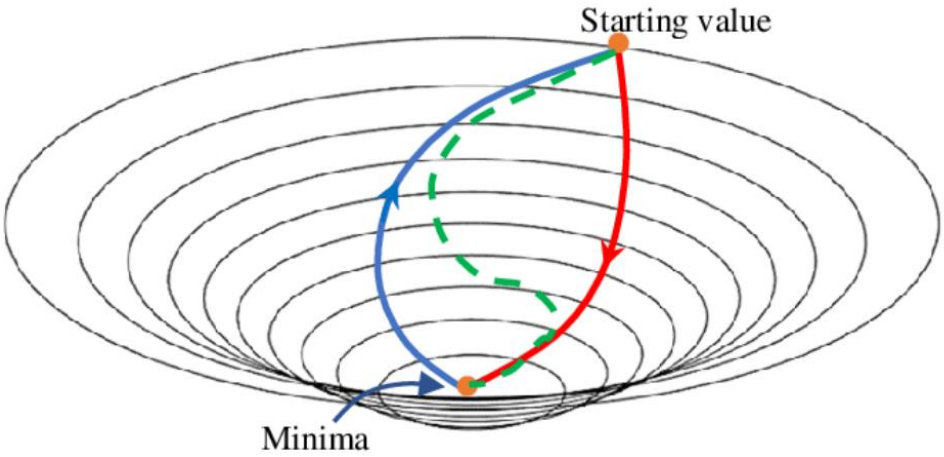
Scaled-Conjugate Gradient (*SCG*) је метод оптимизације првог реда који избегава скупе прорачуне Хесијан матрице [10] , ослањајући се само на градијент првог реда. Он користи коњуговане смерове (*conjugate directions*) за претрагу, уместо да се креће само у смеру најбржег спуштања (који је најчешће неуспешан у уским долинама функције грешке). Ови коњуговани смерови осигуравају да се неће изгубити напредак постигнут у претходним корацима. Алгоритам је скалиран како би се убрзало налажење оптималног корака коришћењем ефикаснијих смерова, па је тако много бржи од стандардне методе најбржег спуштања – *Gradient Descent*-а.

Иако је *SCG* ефикасан, он и даље спада у методе првог реда јер не користи информације о кривини (Хесијан матрица) функције губитка. Због тога је његова брзина конвергенције често ограничена у поређењу са методама које користе апроксимацију другог реда. Једна од таквих оптимизационих метода је управо Левенберг-Маркварт која комбинује Гаус-Њутнову методу и *Gradient Deescent* методу. Левенберг-Маркварт алгоритам ради на принципу апроксимације Хесијан матрице коришћењем Јакобијан матрице [11]. Јакобијан је матрица првих парцијалних извода градијената грешке у односу на тежине. Коришћењем ове апроксимације, алгоритам ефективно добија информације о кривини функције грешке, што му омогућава да предвиди идеалну локацију минимума. Дакле, наведена метода је другог реда, па је тиме и бржа од методе *SCG*.

Употреба ових алгоритама се кодирањем у *MATLAB*-у позива линијом *trainFcn = 'trainscg'* или *'trainlm'*.



Слика 21. Поређење конвергенције метода Gradien Deescent (зелено) и SCG (црвено)



Слика 22. Левемберг-Марквартов метод (зелено), који почиње као Gradient Deescent (плаво) и пребацује се на Гаус-Њутнов метод (црвено) у околини минумума

4. СИМУЛАЦИЈА

У наредном поглављу приказаће се суштина рада тако што ће се применом претходно изнете методологије приказати три случаја тренинга мреже за различите скупове података.

4.1. Симулација 1

Симулација 1 показаће утицај промене броја неурона скривеног слоја на резултате мреже.

4.1.1. Скуп података

Предметни скуп података [23] је синтетички скуп података креиран да симулира податке о предиктивном одржавању који се обично срећу у индустријском окружењу. Улазни подаци представљају карактеристике процеса на основу којих мрежа треба да класификује сваки од узорака (машина) у класу машинског квара који се код узорка јавио. Укупно има 10000 узорака.

Улазне карактеристике:

- Температура ваздуха [K], генерисана коришћењем насумичног хода и нормализована око 300 K.
- Температура процеса [K], добијена додавањем 10 K на температуру ваздуха, уз нормализовани насумични ход.
- Брзина обртаја [rpm], израчуната из снаге од 2860 W са наслоњеним нормално расподељеним шумом.
- Момент силе [Nm], нормално расподељен око 40 Nm.
- Истрошеност алата[min].

Излазне класе:

- TWF (Tool Wear Failure), квар истрошености алата. Настаје ако је истрошеност алата између 200–240 минута. Спада 46 узорака.
- HDF (Heat Dissipation Failure), квар расипања топлоте. Настаје ако је разлика у температури испод 8.6 K и брзина ротације испод 1380 rpm. Спада 115 узорака.
- PWF (Power Failure), квар напајања/снаге. Настаје ако је снага испод 3500 W или изнад 9000 W. Спада 95 узорака.
- OSF (Overstrain Failure), квар пренапрезања. Настаје ако производ истрошености алата и обртног момента прелази одређене прагове. Спада 98 узорака.
- RNF (Random Failures), насумични кварови. Сваки процес има 0.1% шансе за квар без обзира на параметре. Спада 19 узорака.
- No Failure, нема квара. Спада 9652 узорака.

4.1.2. Резултати

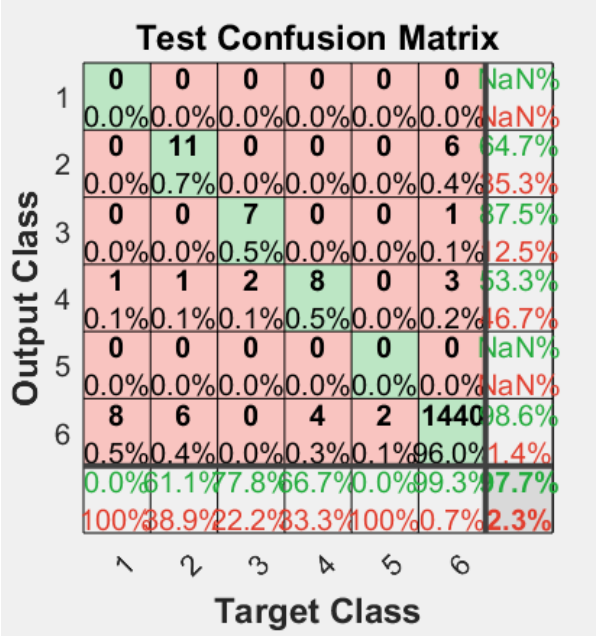
	5. неурона	10. неурона	30. неурона	50. неурона
Епохе	87,9	74,6	74,2	66,3
Перформанса	0,0133	0,01166	0,01104	0,0112
Градијент	0,00126	0,0014225	0,00167	0,00163
	5. неурона	10. неурона	30. неурона	50. неурона
Грешка тренинга	2,35%	2,21%	2,11%	2,15%
Грешка валидације	2,51%	2,1%	2,37%	2,30%
Грешка теста	2,48%	2,22%	2,54%	2,37%
Грешка укупно	2,38%	2,18%	2,22%	2,21%

Табела 3. Резултати симулације 1

4.1.3. Анализа

Анализа показује да је мрежа постигла изузетно висок ниво успешности у класификацији кварова, независно од броја неурона у скривеном слоју. Проценат грешке теста кретао се у врло уском опсегу, што указује да је модел готово савршено савладао дати скуп података. Најбољи проценат успешности је дала мрежа са 10 неурона (са грешком теста од 2,22%). Разлика између грешке тренинга (2,21%) и грешке теста (2,22%) износи свега 0,01%. Добијена, готово идентична вредност доказује да је дати модел научио да одлично генерализује проблем. Са повећањем броја неурона на 30 и 50, мрежа је савладавала тренинг за мањи број епоха, што је довело до мање грешке тренинга (2,11% и 2,15%), али је значајно повећало грешку теста (2,54% и 2,37%). То указује на постојање благог *Overfitting*-а, што би се са даљим повећањем неуорона врло вероватно повећавало. Ниска просечна вредност перформансе указује на то да је мрежа била изузетно сигурна при својим одлукама класификације.

Мали проценат грешке се крије у неуравнотеженој расподели класа, где 96.52% узорака припадају класи *No Failure*. Из овакве расподеле, мрежа највећи проценат својих грешака базира на лоше класификованим узорцима у некој од преостале четири класе, док за класу *No Failure* има потпуну контролу и образац учења, што се види и из матрице конфузије. Овакавом моделу промена оптимизационог алгоритма није нужна ради повећања квалитета.



Слика 23. Матрица конфузије тест података неуронске мреже са 10 скиверних неруона

4.2. Симулација 2

Симулација 2 показаће утицај промене оптимизационог метода на резултате мреже.

4.2.1. Скуп података

Анализирани скуп података [25] се састоји од прикупљених карактеристика и фактора који доприносе срчаном удару, што је уједно и главни узрок смртности широм света, обухватајући коронарну срчану болест, цереброваскуларне болести и друге проблеме са срцем и крвним судовима. Састоји се од 1319 узорака (пацијената) који се бинарно класификују по томе да ли постоји присуство срчаног удара или не.

Улазне карактеристике:

- Године старости пацијента
- Пол
- Срчани пулс
- Горња вредност крвног притиска
- Доња вредност крвног притиска
- Ниво шећера у крви
- Ниво креатина у крви
- Ниво тропонина – индикатор оштећења срца

Број пацијената који су класификовани са ризиком од срчаног удара је 810, а који нису 509.

4.2.2. Резултати

	10. неурона	30. неурона	50. неурона
Епохе	53	41,7	43,1
Перформанса	0,5031	0,4697	0,4553
Градијент	0,1179	0,06987	0,0938
	10. неурона	30. неурона	50. неурона
Грешка тренинга	23,49%	24,51%	22,65%
Грешка валидације	21,17%	26,58%	23,57%
Грешка теста	22,47%	25,92%	24,90%
Грешка укупно	23,00%	25,93%	23,12%

Табела 4. Резултати симулаиције 2 коришћени Scaled-Conjugate Gradient оптимизациони алгоритам

	10. неурона	30. неурона	50. неурона
Епохе	232,9	90,1	89,1
Перформанса	0,04722	0,04382	0,1302
Градијент	0,06626	0,15237	0,1164
	10. неурона	30. неурона	50. неурона
Грешка тренинга	6,19%	4,9%	5,05%
Грешка валидације	10,11%	12,16%	12,40%
Грешка теста	8,34%	11,07%	14,15%
Грешка укупно	6,10%	6,92%	7,49%

Табела 5. Резултати симулације 2 користећи Левенберг-Марквартов оптимизациони алгоритам

4.2.3. Анализа

Резултати *SCG*, као и у прошлој симулацији, су слични једни другима. Међутим, овога пута проценти грешака су много виши (22,47% - 25,92%), мрежа погрешно класификује у просеку сваки четврти или пети узорак, што је изузетно лоше. Проценат високе грешке теста је производ високе грешке тренинга, односно појаве *Underfitting*-а. Из вредности перформанске може се закључити и да је модел био изузетно несигуран при класификацији, готово да је нагађао са вероватноћом 50% - 50%.

Променом на Левенберг-Марквартов оптимизациони алгоритам грешка теста је успешно смањена (за око 10% - 15%), што је изузетно побољшање. Може се приметити да је овај алгоритам трајао значајније већи број епоха и најпре побољшао резултате тренинг података, а онда самим тим и резултате теста. Са повећањем броја неурона теорија је још једном доказана – број епоха се смањио, а успешност тренинга повећала. Међутим, како је посматрани алгоритам успешно решио проблем *Underfitting*-а, тако је изазвао појаву *Overfitting*-а – грешка теста је константно већа од грешке теста. Мрежа је сада била много

сигурнија при класификацији са перформасном блоској нули. Најбољи резултат показала је мрежа са десет скривених неурона – најспорија, али са најблажим *Overfitting*-ом и тиме најбољим резултатима теста (91,6% успештности). Мрежа са тридесет скривених неурона је дала слабији резултат (88,93% успешности), али је идаље била изузетно сигурна у класификацији, па чак и у то да је лоше класификовала. Већ са даљим повећањем скривених неурона (50 и више), број епоха се смањује и успешност тренинга побољшава, али се уједно повећава и утицај *Overfitting*-а, па се успешност теста смањује јер се мрежа превише навикава на тренинг податке.

4.3. Симулација 3

Трећа и последња симулација показаће да постоје случајеви код којих мрежа не може да постигне значајно побољшање упркос променама параметара алгоритма.

4.3.1. Скуп података

Испитивани, едукативни скуп података [26] прикупљен је из система за управљање учењем (*LMS*) под називом *Kalboard 360*. *Kalboard 360* је мулти-агентски *LMS* дизајниран да олакша учење и пружи синхрони приступ едукативним ресурсима са било ког уређаја. Подаци су прикупљени помоћу алата за праћење активности ученика под називом *experience API* (*xAPI*). *xAPI* омогућава праћење напретка ученика и њихових акција (нпр. читање чланка или гледање видеа), помажући да се одреди ученик, активност и објекти који описују искуство учења. Састоји се од 480 узорака (ученика) који се класификују у три групе у зависности од њихове коначне оцене.

Улазне карактеристике:

- Број подигуте руке на часу
- Број посећених часова
- Број прегледане литературе
- Број вођених дискусија
- Број изостанака

Излазне класе:

- Класа L (Low-Level), спадају ученици са оценом од 0 до 69.
- Класа M (Middle-Level), спадају ученици са оценом од 70 до 89.
- Класа H (High-Level), спадају ученици са оценом од 90 до 100.

4.3.2. Резултати

	10. неурона	30. неурона	50. неурона
Епохе	13,8	18	13,2
Перформанса	0,2268	0,2142	0,2194
Градијент	0,01714	0,03952	0,06978
	10. неурона	30. неурона	50. неурона
Грешка тренинга	35,18%	32,66%	36,4%
Грешка валидације	31,96%	32,48%	35,56%
Грешка теста	41,96%	39,16%	40,26%
Грешка укупно	35,69%	33,62%	36,84%

Табела 6. Резултати симулације 3 користећи SCG оптимизациони алгоритам

	10. неурона	30. неурона	50. неурона
Епохе	9	8,4	12
Перформанса	0,1138	0,156	0,05562
Градијент	0,01908	0,0186	0,03879
	10. неурона	30. неурона	50. неурона
Грешка тренинга	33%	26,44%	19,96%
Грешка валидације	37,1%	35,88%	31,4%
Грешка теста	39,45%	38,51%	36,1%
Грешка укупно	33,76%	34,83%	24,1%

Табела 7. Резултати симулације 3 користећи LM оптимизациони алгоритам

4.3.3. Анализа

По овим резултатима види се да је промена оптимизационог алгоритма минимално спустила грешку теста за свега 2% до 3% у просеку, па иако је *LM* технички бољи, ниједан алгоритам не може да постигне значајно побољшање (грешка остаје висока). Даљим повећањем броја неурона у скривеном слоју, при коришћењу *LM* оптимизационог алгоритма, не постиже се ништа сем повећања успешности тренинга, док успешност теста у просеку остаје иста. Обе мреже су сигурне и свесне у своје одлуке, али просто не успевају да савладају дати скуп података у коме се управо и налази разлог овако велике грешке. Најпре, сами улазни подаци ученика нису довољно репрезентативни и повезани са оценом, па их мрежа у великој мери види као шумове и не може да направи довољно смислене и јаке везе међу њима. На пример, њихова активност није логички параметар који би могао да указује на то да ли ће имати нижу или вишу оцену јер посећеност неком предмету може бити ниска, али просек положених испита и високих оцена из истог могу бити високе. Због оваквог скупа података, мрежа није успела да пронађе алгоритам веза између улазних карактеристика и излазних класа, па је остала ненаучена (*Underfitting*). Други разлог се налази у излазним опсезима трију класа. Уколико постоји велики број

ученика који имају поене склоне прелазним оценама (око 70 или око 90 поена), мрежа ће имати недоумице у коју од две могуће класе да класификује ученика. То значи да су два ученика могла да имају исте карактеристике, а да један припада класи *Low-Level*, а други класи *Middle-Level*, док би их мрежа учила као само једну од две класе. Уколико узмемо да је просечна грешка теста симулације око 38% , ова грешка се назива *Irreducible Error* или Noise Error и директно је везана за сам скуп података и шумове у њему.

4.4. Напомена: Утицај насумичне поделе скупова података

Још један од узрока добијених резултата у све три симулације може бити и насумична расподела података. Већ је речено да се скуп података дели на тренинг, валидационе и тест податке у размери 75% / 15% / 15%. Наведена подела је стандардни и најчешћи приступ, који се свакако може мењати по потреби. У сваком покретању тренажног процеса мреже, ови подаци бивају насумично распоређени свакој од група, па иста мрежа за исти скуп података некада може имати теже податке за тренинг, а некада теже за тест. Наведени фактор, коришћењем MATLAB-а на тај начин може утицати код неких скупова података где припадност класама није распоређена у најбољој мери. У пракси, обично тренинг, валидациони и тест подаци бивају одвојени. Најпре се мрежа посебно учи на тренинг и валидационим подацима, а онда јој се доводе тест подаци.

5. ЗАКЉУЧАК

Овај рад је спровео детаљну емпиријску анализу перформанси неуронске мреже за препознавање алгоритма при проблемима бинарне и вишекласне класификације у односу на промену броја неурона скривеног слоја, промене оптимизационог алгоритма (*SCG* и *LM*) и комплексности скупа података. У свим студијама случаја, *LM* алгоритам се показао као супериорнији оптимизатор у односу на *SCG*, с тим што се јавило и постојање ограничења перформанси у виду *Irreducible Error*-а. На скупу података ниског квалитета и слабе корелације, мреже су стигле више грешке теста, док су логички оптималнији скупови постигли већу успешност. Резултати су доследно показали да повећање броја неурона у скривеном слоју изнад оптималног броја (најчешће 10 неурона) доводи до појаве Overfitting-а, што резултира погоршањем грешке на тест скупу, иако се грешка тренинга смањује.

Како је оваква мрежа погодна за једноставније детерминистичке проблеме, даље унапређивање се може извршити проширењем броја скривених слојева, употребом адаптивнијих оптимизационих алгоритама попут Адама (*Adam*) и покушај превазилажења *Irreducible Error*-а, изградња дубоких и конволуцијских мрежа за препознавање објеката, лица, итд. Овакав модел би могао на основу података прикупљених из околине у реалном времену путем сензора или *IoT (Internet Of Things)* уређаја могао да класификује кварове машине, објекте на путу у виду возних трака или рупа, учеснике у саобраћају и на основу тога да примени одговарајућу дијагностику или пружи одговарајућу графику (слику) окружења. Модел способан за класификацију све више је применљив и у медицини за препознавање обољења на основу ћелијских снимака и задавања дијагнозе где постиже процентуално боље резултате од данашњих доктора.

6. ЛИТЕРАТУРА

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Pearson, 2021.

[2] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. Cham, Switzerland: Springer, 2023.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York, NY, USA: Springer, 2006.

[4] https://www.mathworks.com/discovery/pattern-recognition.html

[5] https://www.geeksforgeeks.org/machine-learning/pattern-recognition-introduction/

[6] M. A. Nielsen, *Neural Networks and Deep Learning*. San Francisco, CA, USA: Determination Press, 2015

[7] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, 2nd ed. Cham, Switzerland: Springer, 2023.

[8] https://www.geeksforgeeks.org/machine-learning/backpropagation-in-neural-network/

[9] https://cent.mas.bg.ac.rs/nastava/ksivi_mo/Sajt%20KSiVI%20i%20Metode%20odlucivanja/Materijali/Metode_odlucivanja/5/AT-5_MO_Mart2009.pdf

[10] J. E. Marsden and A. Tromba, *Vector Calculus*, 6th ed. New York, NY, USA: W. H. Freeman, 2012.

[11] J. Stewart, *Calculus*, 8th ed. Boston, MA, USA: Cengage Learning, 2015.

[12] K. Binmore and J. Davies, *Calculus: Concepts and Methods*. Cambridge, UK: Cambridge University Press, 2002.

[13] https://www.ibm.com/think/topics/gradient-descent

[14] https://www.youtube.com/watch?v=znqbtL0fRA0

[15] https://medium.com/data-science/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1

[16] https://www.mathworks.com/products/deep-learning.html#

[17] D. Hughes-Hallett, A. M. Gleason, and W. G. McCallum, *Calculus: Single and Multivariable*, 8th ed. Hoboken, NJ, SAD: Wiley, 2020

[18] H. Demuth and M. Beale, *Neural Network Toolbox User's Guide*. Natick, MA, USA: The MathWorks, 2004.

[19] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge, UK: Cambridge University Press, 2020.

[20] https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm

[21] https://en.wikipedia.org/wiki/Conjugate_gradient_method

[22] T. Danka, *Mathematics of Machine Learning: Master Linear Algebra, Calculus, and Probability for Machine Learning*. Birmingham, UK: Packt Publishing, 2025.

[23] https://www.kaggle.com/datasets/abdelazizsami/predictive-maintenance-dataset

[24] https://www.sciencedirect.com/science/article/pii/S2949863524000219#sec0010

[25] https://www.kaggle.com/datasets/bharath011/heart-disease-classification-dataset

[26] https://www.kaggle.com/datasets/aljarah/xAPI-Edu-Data.
[27] Neuronske mreze 4 - BP algoritam ucenja.pptx

7. ДОДАТАК 1. КОД *MATLAB* ПРОГРАМА

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 23-Oct-2025 13:32:04
%
% This script assumes these variables are defined:
%
%   xAPI_Edu_DataS3 - input data.
%   xAPI_Edu_DataS4 - target data.

x = xAPI_Edu_DataS3';
t = xAPI_Edu_DataS4';

% Choose a Training Function
% For a list of all training functions type: help nntrain
trainFcn = 'trainlm';

% Create a Pattern Recognition Network
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize, trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'crossentropy';  % Cross Entropy

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
```

```
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)


% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
```