УНИВЕРЗИТЕТ У НИШУ

ЕЛЕКТРОНСКИ ФАКУЛТЕТ

Ведран Митић 18775

ЗАВРШНИ РАД – ИСТРАЖИВАЧКИ РАД

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

Ментор:

проф. др Мирослав Миловановић

2025.

Садржај

1. УВОД

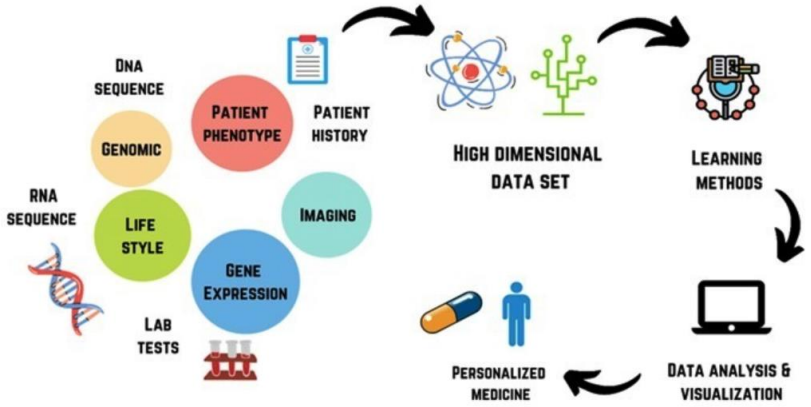
Вештачка интелигенција (ВИ) данас представља један од најдинамичнијих и најважнијих праваца у инжењерству и науци, са посебним фокусом на машинско учење (*Machine Learning*). [1] У домену анализе података, неуронске мреже су се доказале као супериорна алатка за решавање комплексних проблема класификације и регресије у различитим областима. Једна од критичних области примене је биомедицина, где тачна и брза дијагноза игра кључну улогу. Кардиоваскуларне болести (КВБ) остају водећи узрок смртности широм света. Унапређење прецизности раних дијагностичких алата може значајно утицати на клиничке исходе. Примена вишеслојних вештачких неуронских мрежа (*ANN*) на анализу биомедицинских параметара нуди велики потенцијал за аутоматизовану и прецизну дијагностику. Међутим, поузданост модела директно зависи од његове архитектуре и, што је најважније, од избора оптимизационог алгоритма који управља процесом учења.

Циљ овог рада је спровођење детаљне емпиријске анализе перформанси трослојне неуронске мреже при решавању проблема бинарне класификације КВБ. Фокус је на компаративној студији два најчешће коришћена алгоритма за *Backpropagation*:

1. *Scaled-Conjugate Gradient* (*SCG*) алгоритам, познат по брзој конвергенцији и меморијској ефикасности.
2. Левенберг-Марквартов (*LM*) алгоритам, познат по изузетно брзој и поузданој конвергенцији, али уз веће меморијске захтеве.

Коришћењем алатки програма *MATLAB*, истраживање ће квантитативно измерити:

1. Утицај броја неурона у скривеном слоју на појаву *Overfitting-a*.
2. Разлику у проценту грешке и брзини конвергенције између *LM* и *SCG* алгоритама.
3. Дефинисање оптималне архитектуре мреже за конкретни биомедицински скуп података, која обезбеђује највиши ниво тачности и поузданости модела.



Слика 1. Улога вештачке интелигенције у биомедицини

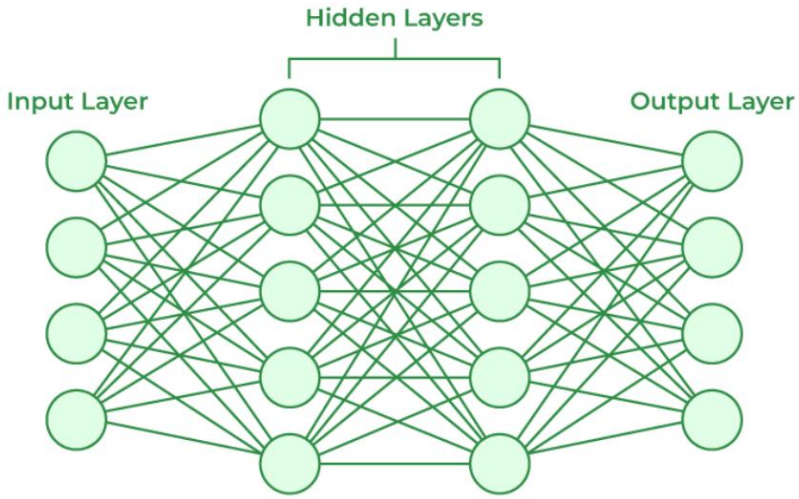
2. ТЕОРИЈСКА ОСНОВА

2.1. Вишеслојна неуронска мрежа

У раду је коришћена трослојна вештачка неуронска мрежа типа вишеслојни перцептрон (*Multi-Layer Perceptron, MLP*). Ова архитектура се састоји од:

1. Улазног слоја: Прима улазне особине (*features*) скупа података.
2. Скривеног слоја: Састоји се од одређеног броја неурона, одговорних за извођење нелинеарних трансформација улазних података. Број неурона у овом слоју је кључни параметар који одређује комплексност и способност генерализације модела.
3. Излазног слоја: Генерише финални резултат класификације.

Сваки неурон у скривеном и излазном слоју користи активациону функцију која уводи нелинеарност, омогућавајући мрежи да научи сложене односе. [2] За проблеме класификације, најчешће се користи *Softmax* функција у излазном слоју, која излазе претвара у дистрибуцију вероватноће припадности класама. Активациона функција у скривеном слоју која ће се користити у раду је Сигмоид.



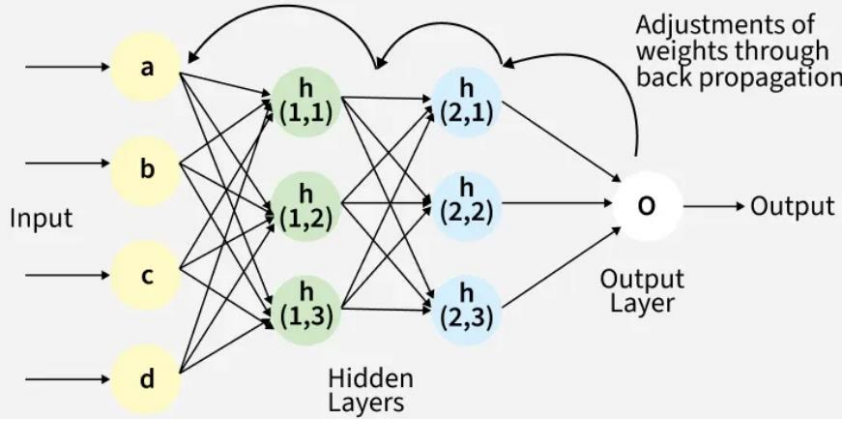
Слика 2. Архитектура неуронске мреже

2.2. Алгоритам пропагације уназад

Процес учења вишеслојне неуронске мреже заснива се на алгоритму пропагирања грешке уназад - *Backpropagation*. Овај алгоритам је ефикасна метода за израчунавање градијената функције грешке у односу на све тежине у мрежи. Његов рад се одвија кроз две фазе:

1. Фаза Унапред (*Forward Pass*): Улазни подаци пролазе кроз мрежу до излазног слоја, где се израчунава грешка (у раду *Cross-Entropy*) у односу на жељени излаз.
2. Фаза Уназад (*Backward Pass*): Градијент грешке се пропагира уназад кроз мрежу, од излазног слоја ка улазном, израчунавајући тако делимичне изводе функције грешке по свакој појединачној тежини.

Оптимизациони алгоритам (оптимизатор) тада користи ове израчунате градијенте за модификовање тежина мреже.



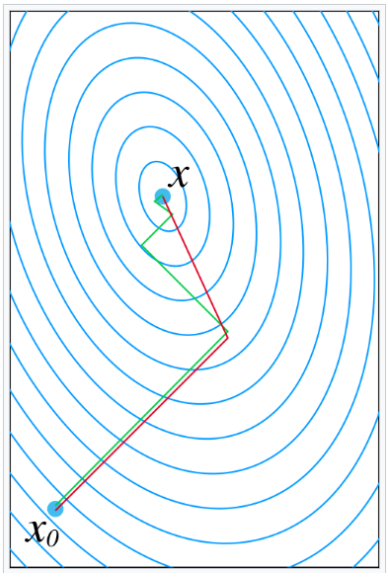
Слика 3. Шема Backpropagation алгоритма

2.3. Оптимизатори

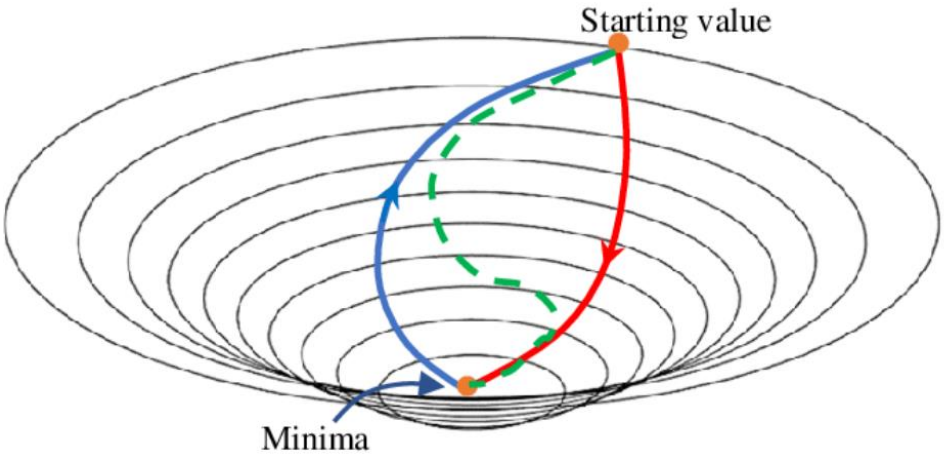
Улога оптимизатора у неуронској мрежи је да пронађе најбољи сет тежинских коефицијената који ће минимизовати функцију губитка (Cross-entropy) и тиме смањити грешку мреже. Као што је раније објашњено, *Backpropagation* алгоритам је универзални механизам за рачунање градијента грешке. Међутим, оптимизационе 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 [7] , 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 [7]. 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*.



Слика 4. Поређење конвергенције метода Gradien Deescent (зелено) и SCG (црвено)



Слика 5. Левемберг-Марквартов метод (зелено), који почиње као Gradient Deescent (плаво) и пребацује се на Гаус-Њутнов метод (црвено) у околини минумума

3. СИМУЛАЦИЈА

3.1. Скуп података

За емпиријску анализу коришћен је анализирани скуп података [12] који се односи на класификацију ризика од кардиоваскуларних болести (КВБ). Скуп се састоји од осам улазних нумеричких особина: 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.

Пре увођења у мрежу, спроведене су следеће фазе прераде података:

- Скалирање (Нормализација): Све нумеричке улазне особине скалиране су на опсег [0, 1] помоћу *Min-Max* нормализације. Овај поступак обезбеђује да ниједна особина не доминира над другима због своје јединичне скале.
- Кодирање излаза: Излазне класе су кодиране коришћењем *One-Hot encoding*-ом, где је вредности *DA* додељена вредност [1, 0], а вредности *NE* [0, 1], што је неопходно за рад са функцијом грешке *Cross-Entropy*.

3.2. Експериментални дизајн

Симулација је спроведена на трослојној неуронској мрежи са 8 улазних неурона и 2 излазна, док је број неурона скривеног слоју постепено мењан почевши од 10, на 30, па на 50. Скуп података је ради подељен у три категорије у сразмери 70% / 15% / 15% на тренинг, валидационе и тест податке респективно. За оптимизационе алгоитме најпре је коришћен *SCG*, а затим *LM*. Након десетоструког покретања сваке од конфигурација и записивања резултата сваког од покретања, израчунате су просечне вредности и смештене у табелу коначних резултата. Ова табела носи информацију о брзини, сигурности и успешности сваке од конфигурација и служи за визуелизацију поређења истих.

3.3. Резултат *SCG* оптимизатора

Софтвер *MATLAB* и његов алат за рад са неуронским мрежама *Neural Network Toolbox*, пружају детаљан опис рада и резултата мреже након покретања у виду табела и графикона. Један приказ резултата за мрежу од 10 неурона у скривеном слоју, користећи SCG оптимизациони метод изгледа овако:

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	19	1000
Elapsed Time	-	00:00:01	-
Performance	0.827	0.579	0
Gradient	0.55	0.0605	1e-06
Validation Checks	0	6	6

Слика 6. Табела резултата тренажног процеса

Табела на *Слици 6* 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.

Model Summary

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

Data

Predictors: HeartAttack - [1319x8 double]

Responses: HeartAttack_1 - [1319x1 double]

HeartAttack: double array of 1319 observations with 8 features.

HeartAttack_1: double array of 1319 observations with 1 classes.

Algorithm

Data division: Random

Training algorithm: Scaled conjugate gradient

Performance: Cross-entropy error

Training Results

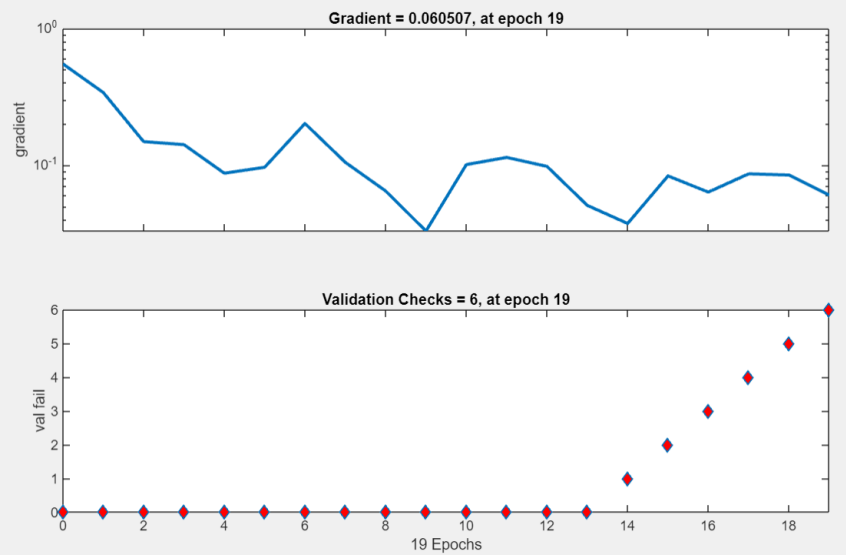
Training start time: 28-Oct-2025 18:42:47

Layer size: 10

	Observations	Cross-entropy	Error
Training	923	0.5956	0.3174
Validation	198	0.5635	0.2828
Test	198	0.6040	0.3333

Слика 7. Прозор резултата модела

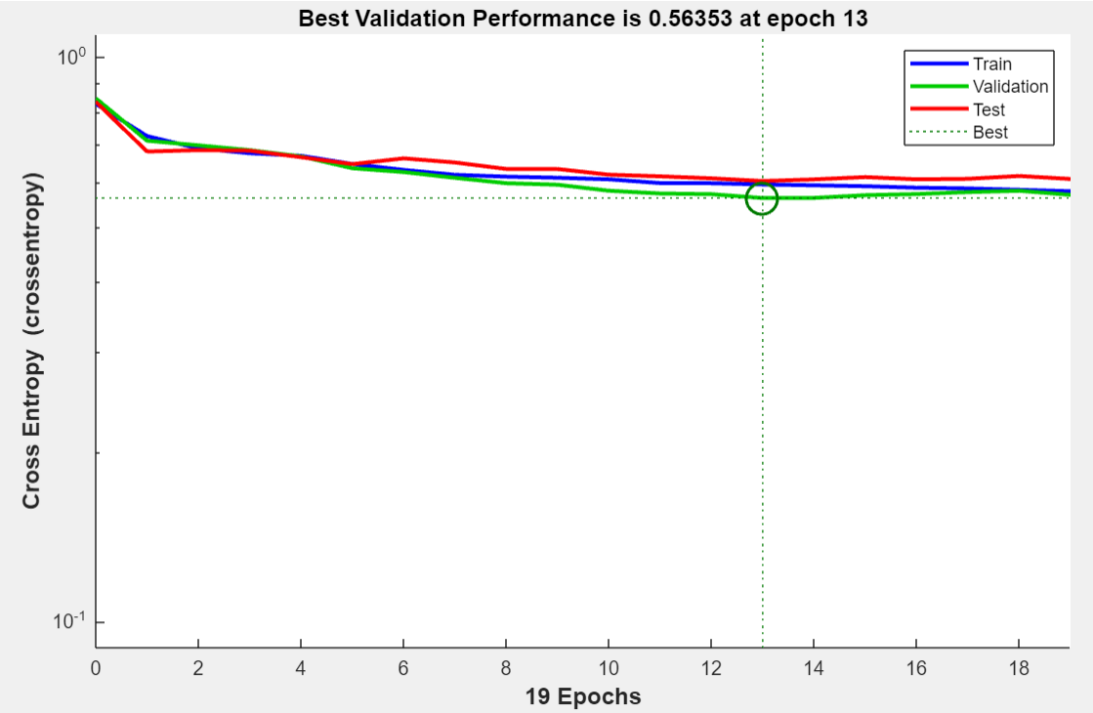
Вредност *Observations* представља број узорака узетих за сваку од група података, *Cross-entropy* вредност грешке у најбољој тачки валидације (*Best Validation Performance*), а *Error* грешку у односу на број узорака за дату групу. За дат пример, *Error* од 0,2828 при валидацији значи да је од 198 узорака, погрешно класификовано 28,28% , што је 56 узорака.



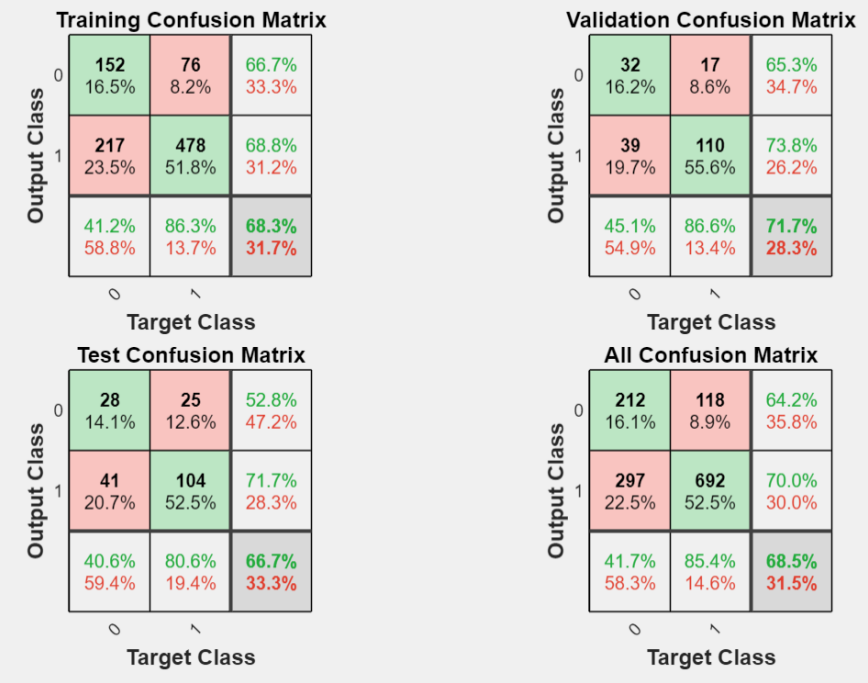
Слика 8. Training state plot

Training State Plot је график који приказује промену вредности градијента и бројања *Validation Check*-а при тренингу кроз епохе. Поред њега постоји и График перформансе. То је 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, што је подразумевано намештање у *MATLAB*-у. На *Слици 9.* се види да је мрежа отприлике и остварила своју најбољу перформансу у *Best Validation Performance*-у (вредност грешке је престала да опада и наставила са устаљењем и након епохе 13) што не мора увек да буде случај. Дешавају се случајеви и где је грешка тренинг и тест података у истој (и наредним) епохи била мања. На наведену особину графика највише утиче насумична расподела података у ове три групе. За тренутни пример, расподела је перфектна јер мрежа за све три групе података подједнако брзо и (не)тачно учи, а то се јасно види скоро поклапањем трију линија. Како год, овај график је и приказ ненаучене мреже, где је грешка недовољно ниског реда врло брзо престала да опада.



Слика 9. График перформанси модела



Слика 10. Матрице конфузије

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. По датом примеру се јасно види да су проценти веома лоши и да је модел погрешно класификовао сваки трећи узорак (33,3% грешке теста).

Следећи изабрану емпиријску методу из *Експериментални дизајн*, након покретања свих конфигурација добијају се следећи 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 1. Резултати модела коришћи Scaled-Conjugate Gradient оптимизациони алгоритам

3.4. Резултат LM оптимизатора

Пошто *MATLAB* софтвер нема неки вид прозора за ручну промену оптимизационог метода, за прелаз на *LM* метод потребно је најпре генерисати код постојеће мреже кликом на дугме *Generate Code*, а онда изменити линију кода `trainFcn = 'trainscg';` у `trainFcn = 'trainlm';`. Цео код се може наћи у *Додатак 1*. Покретањем се добијају следећи резултати:

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:05	-
Performance	0.293	0.0427	0
Gradient	0.17	0.0017	1e-07
Mu	0.001	0.01	1e+10
Validation Checks	0	1	6

Слика 11. Табела резултата другог тренажног процеса

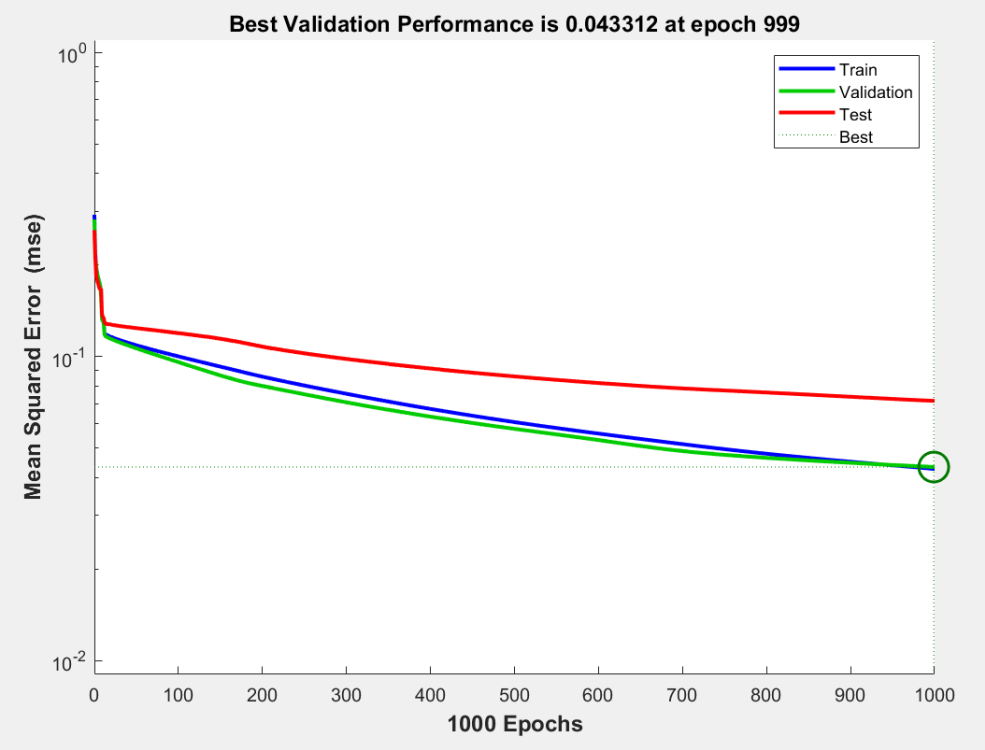
Слика 11. показује да је тренинг трајао значајно дуже него претходног пута, а зауставио се након предефинисаног броја епоха – 1000. *Mu* или μ је параметар који контролише промену између метода које *LM* користи – Гаус-Њутнов и *Gradient Descent*. За мање вредности означава да алгоритам користи Гаус-Њутнов, а за веће *Gradient Descent* метод.

Покретање кода не генерише прозор попут оног на Слици 7. већ избацује бројке директно у командни прозор:

```
performance =

    0.0471


trainPerformance =

    0.0427


valPerformance =

    0.0433


testPerformance =

    0.0715
```

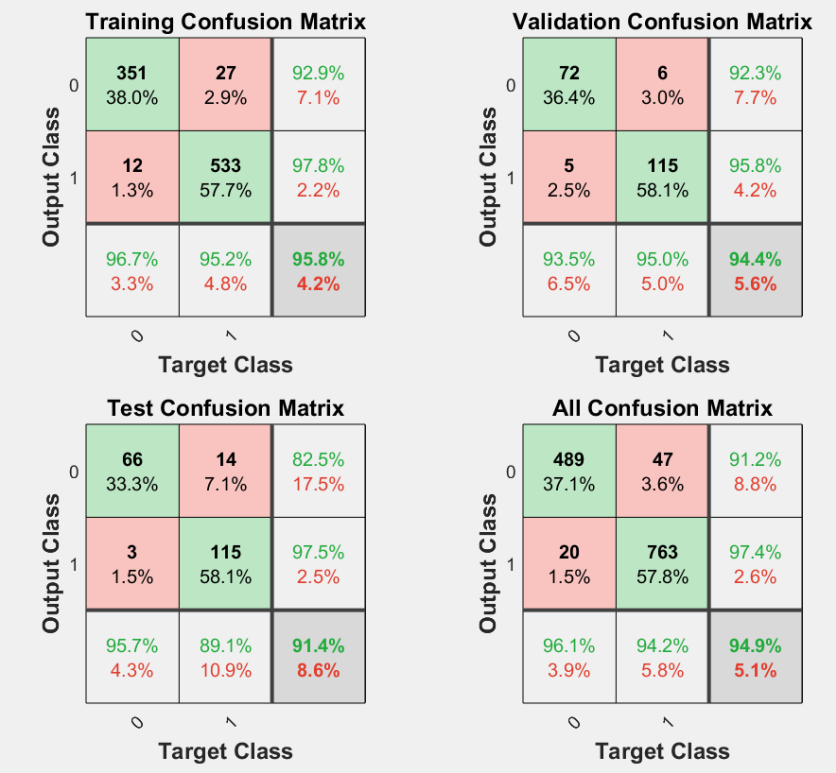
Слика 12. Резултати другог модела

Ниска вредност ових вредности показује да је модел био сигуран у своје одлуке при класификацији.



Слика 13. График перформанси другог модела

График перформансе овога пута изгледа знатно боље, најпре нагло пада (што је честа карактеристика мрежа), а онда линеарно опада без видљивих осцилација. График тест података је мало изнад тренинг графика што указује да је расподела била таква да су ти подаци били за нијансу тежи, а због сличности ова два графика може се наговестити и појава *Overfitting*-а.



Слика 14. Матрице конфузије другог модела

Из матрица конфузија се јасно може увидети побољшање перформанси модела променом оптимизационог алгоритма, али и појаву благог Overfitting-a где је успешност теста мало мања од успешности тренинга. Ово се догодило јер се мрежа превише навикла на тренинг узорке, док је међу тест узорцима било нових, до сада невиђених које мрежа није успела тачно да класификује.

Следећи изабрану емпиријску методу из *Експериментални дизајн*, након покретања свих конфигурација добијају се следећи резултати:

	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%	12,4%
Грешка теста	8,34%	11,07%	14,15%
Грешка укупно	6,1%	6,92%	7,49%

Табела 2. Резултати другог модела користећи Левенберг-Марквартов оптимизациони алгоритам

3.5. Анализа

Код SCG оптимизационог алгоритма, 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 перформансе 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. ЗАКЉУЧАК

Истраживачки рад је спровео емпиријску студију перформанси два оптимизациона алгоритма, *Scaled-Conjugate Gradient* и Левенберг-Марквартовог алгоритма, у циљу дефинисања оптималне архитектуре трослојне неуронске мреже за класификацију ризика од кардиоваскуларних болести.

Емпиријска анализа је недвосмислено доказала супериорност *LM* алгоритма. *LM* је постигао најнижу просечну грешку теста од 8,34%(10 неурона), што је значајно боље од најбољег резултата *SCG*-а од 22,47%. Дефинисана је оптимална архитектура мреже као 10 неурона у скривеном слоју. Док су конфигурације са 30 и 50 неурона постизале боље резултате на тренинг скупу, њихова грешка теста је била већа (на пример, 14,03% за 50 неурона) због појаве *Overfitting*-а.

Добијени резултати пружају практичну основу за избор модела при развоју дијагностичких алата. У домену биомедицине, *LM* алгоритам са минималном архитектуром (10 неурона) представља оптимални баланс између брзине, поузданости и избегавања *Overfitting*-а.

Будући рад може да истражи примену напредних регуларизационих техника (нпр. *Dropout*) у комбинацији са *LM* алгоритмом, како би се проценила могућност даљег побољшања перформанси мрежа веће комплексности.

5. ЛИТЕРАТУРА

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

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

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

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

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

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

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

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

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

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

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

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

[13] Neuronske mreze 4 - BP algoritam ucenja.pptx

6. ДОДАТАК 1 КОД MATLAB ПРОГРАМА

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 28-Oct-2025 19:52:07
%
% This script assumes these variables are defined:
%
%   HeartAttack - input data.
%   HeartAttack_1 - target data.

x = HeartAttack';
t = HeartAttack_1';

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
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)

% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotconfusion(t,y)
%figure, plotroc(t,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
```