Univerzitet u Kragujevcu
Fakultet inženjerskih nauka

Predmet: Veštačka inteligencija

Tema seminarskog rada:

Evaluacija asistenta u nastavi

Student:

Marijana Jeremić 620/2021

Predmetni profesori:

Dr. Vesna Ranković
Dr. Tijana Geroski

Saradnik:

Anđela Blagojević

Kragujevac, Septembar 2025.

Sadržaj

1. Postavka zadatka

U ovom radu razmatra se problem višeklasne klasifikacije na osnovu skupa podataka za evaluaciju nastavnih asistenata – **Teaching Assistant Evaluation (TAE)**. Dataset je preuzet sa KEEL repozitorijuma (https://sci2s.ugr.es/keel/dataset.php?cod=188), i predstavlja standardni primer klasifikacionog problema u oblasti obrazovanja.

Zadatak treba rešiti primenom:

- **veštačkih neuronskih mreža**,
- **genetskih algoritama**,
- ili **kombinacijom ovih metoda**.

Kao programski jezik se preporučuje Python. Matlab se može koristiti za brzu proveru izabranog algoritma i za verifikaciju dobijenih rezultata. O izabranom modelu se vodi diskusija prednosti i mana.

Podaci se odnose na evaluaciju rada nastavnih asistenata tokom tri redovna i dva letnja semestra, ukupno **151 asistenata** na predmetima iz oblasti statistike na Univerzitetu u Viskonsinu-Medison (University of Wisconsin–Madison). Ocene koje su studenti dodeljivali asistentima grupisane su u **tri približno jednako zastupljene kategorije**, i to: **niska ocena (low), srednja ocena (medium) i visoka ocena (high).** Te kategorije predstavljaju ciljnu promenljivu (klasu) u zadatku klasifikacije.

Opis atributa:

Attribute description

Attribute	Domain
Native	[1, 2]
Instructor	[1, 25]
Course	[1, 26]
Semester	[1, 2]
Size	[3, 66]
Class	{1, 2, 3}

Ovaj deo opisuje glavne karakteristike skupa podatka TAE.

General information

Teaching Assistant Evaluation data set			
Type	Classification	Origin	Real world
Features	5	(Real / Integer / Nominal)	(0 / 5 / 0)
Instances	151	Classes	3
Missing values?			No

2. Opis ulaznih i izlaznih podataka

Za rešavanje zadatog problema korišćen je skup podataka Teaching Assistant Evaluation (TAE) koji potiče sa Univerziteta Wisconsin-Madison, a dostupan je kroz UCI i KEEL repozitorijume. Skup sadrži ukupno 151 primer dobijen tokom tri redovna i dva letnja semestra. Svaki primer predstavlja evaluaciju jednog asistenta na kursu i opisan je pomoću pet ulaznih atributa, dok šesti atribut predstavlja izlaznu klasu, odnosno ukupnu ocenu uspešnosti asistenta. U nastavku su detaljno opisani svi ulazni i izlazni atributi.

Ulazni atributi:

Native / English speaker - ovaj atribut označava da li je asistent izvorni govornik engleskog jezika.
Vrednosti: 1 = English speaker, 2 = Non-English speaker.
Tip: binarni, kategorijalni.
Za potrebe rada u neuronskoj mreži atribut se kodira u binarnu promenljivu (npr. 1 → 1, 2 → 0), ili alternativno u obliku „one-hot“ kodiranja sa dve kolone. Ovaj podatak je važan jer jezik i akcenat mogu uticati na percepciju kvaliteta asistenta kod studenata.

Instructor - ovaj atribut identifikuje nastavnika čiji kurs asistent vodi.
Vrednosti: celobrojne oznake od 1 do 25.
Tip: nominalni, kategorijalni.
Za neuronsku mrežu je neophodno uraditi „one-hot“ kodiranje, pri čemu se atribut pretvara u 25 kolona (svaka kolona označava jednog nastavnika). Na ovaj način mreža ne tretira brojčane oznake kao redosled, već kao nezavisne kategorije.

Course - atribut predstavlja oznaku kursa na kojem je asistent angažovan.
Vrednosti: celobrojne oznake od 1 do 26.
Tip: kategorijalni.
Kao i u slučaju atributa „Instructor“, i ovde se koristi „one-hot“ kodiranje u 26 kolona. Na taj način se mreži omogućava da nauči potencijalne razlike među kursevima bez lažnog pretpostavljanja ordinalnog poretka.

Semester - ovaj atribut označava tip semestra u kojem je kurs realizovan.
Vrednosti: 1 = Summer semester, 2 = Regular semester.
Tip: binarni, kategorijalni.
Za pripremu podataka atribut se prevodi u binarni oblik (npr. Summer = 1, Regular = 0) ili u „one-hot“ kodiranje sa dve kolone. Informacija o semestru može biti relevantna jer se letnji i redovni semestri razlikuju po broju studenata i intenzitetu nastave.

Class size - atribut predstavlja veličinu grupe, tj. broj studenata na kursu.
Vrednosti: numeričke vrednosti u opsegu od 3 do 66.
Tip: numerički, celobrojni.
Kako neuronske mreže zahtevaju da ulazne vrednosti budu približno ujednačenog reda veličine, ovaj atribut se skalira, najčešće korišćenjem min-max normalizacije ili standardizacije (oduzimanje srednje vrednosti i deljenje standardnom devijacijom). Na taj način se izbegava dominacija velikih vrednosti u učenju mreže.

Izlazni atribut (klasa)
Class attribute (Teaching performance) - ovaj atribut predstavlja ocenu uspešnosti asistenta na kursu i koristi se kao ciljna promenljiva u klasifikaciji.
Vrednosti: 1 = Low, 2 = Medium, 3 = High.
Tip: kategorijalni.
Za rad neuronske mreže, izlaz se kodira u obliku „one-hot“ vektora dužine 3:
Klasa 1 (Low) → [1, 0, 0]
Klasa 2 (Medium) → [0, 1, 0]
Klasa 3 (High) → [0, 0, 1]

U završnom sloju mreže koristi se sigmoidna funkcija. Priprema podataka za mrežu - kodiranje kategorijalnih atributa: za „Instructor“ i „Course“ koristi se one-hot kodiranje zbog velikog broja mogućih vrednosti. Na taj način dobijamo ukupno 51 novu kolonu (25 + 26). Binarni atributi (Native, Semester) mogu se kodirati kao 0/1 ili kao one-hot sa dve kolone. Numerički atribut (Class size) se normalizuje ili standardizuje kako bi imao uporedivu skalu sa ostalim ulazima. Nakon pripreme, svaka instanca je predstavljena vektorom ulaznih podataka koji sadrži kombinaciju binarnih, dummy (one-hot) i skaliranih numeričkih vrednosti. Skup TAE je relativno mali (svega 151 primer), ali sadrži atribute različitog tipa (binarni, kategorijalni sa mnogo klasa i numerički). Zbog toga je vrlo pogodan za zadatak klasifikacije neuronskim mrežama, ali zahteva pažljivu pripremu ulaznih podataka. Poseban izazov predstavljaju atributi sa velikim brojem kategorija („Instructor“ i „Course“), jer povećavaju dimenzionalnost ulaznog prostora i mogu dovesti do overfitting-a na malom skupu. Sa druge strane, izlazna klasa je jasno definisana i balansirana u tri kategorije, što omogućava evaluaciju performansi modela kroz metrike kao što su tačnost, konfuziona matrica, preciznost, odziv (recall) i ROC/AUC krive.

3. Objašnjenje korišćenog algoritma

U seminarskom radu problem klasifikacije evaluacije asistenta u nastavi rešen je primenom veštačke neuronske mreže (ANN, Artificial Neural Network). Ovaj algoritam pripada grupi metoda nadgledanog učenja (engl. supervised learning), gde se mreža obučava na osnovu poznatih ulazno–izlaznih parova kako bi mogla da generalizuje i ispravno klasifikuje nove primere.[1]

3.1 Struktura neuronske mreže

U implementaciji korišćen je model feedforward mreže sa sledećom arhitekturom:

- Ulazni sloj: 5 neurona, koji odgovaraju broju ulaznih atributa (engleski govornik, kurs, instruktor, tip semestra, veličina klase). Ulazni podaci su prethodno normalizovani na interval [0,1].
- Skriveni sloj (hidden layer): Podrazumevane vrednosti u fajlovima script2-.ipynb su npr. n_h1 = 50, n_h2 = 20, a u eksperimentima se pretražuju i druge kombinacije (npr. n_h1 = 68, n_h2 = 43) radi optimizacije tačnosti. Aktivaciona funcija je sigmoidna-
- Izlazni sloj: 3 neurona, koji predstavljaju tri moguće klase evaluacije kursa. Na izlazu se koristi sigmoidna aktivaciona funkcija.

3.2 Princip rada algoritma

Rad mreže može se podeliti u dve faze:

- Propagacija unapred (forward propagation): Ulazni podaci prolaze kroz slojeve mreže, pri čemu se u svakom neuronu računa ponderisana suma ulaza i primenjuje aktivaciona funkcija. Na kraju izlazni sloj daje vektor verovatnoća pripadnosti klasi.
- Učenje putem povratnog širenja greške (backpropagation): Greška između očekivanog izlaza (one-hot kodiran vektor klase) i predikcije mreže računa se funkcijom gubitka (engl. loss function). U ovom projektu koristi se kros-entropijska funkcija gubitka.
- Na osnovu gradijenata izračunatih backpropagation algoritmom, težine i pristranosti neurona se ažuriraju optimizacionim algoritmom.

3.3 Teorijska opravdanost

Razlog za korišćenje neuronske mreže u ovom zadatku leži u njenoj sposobnosti da modeluje složene, nelinearne zavisnosti između ulaza i izlaza. Tradicionalni statistički algoritmi (npr. logistička regresija) često imaju ograničenu mogućnost učenja kada postoji veći broj kategorijalnih i numeričkih atributa u kombinaciji. ANN sa skrivenim slojevima omogućava kreiranje fleksibilnih nelinearnih preslikavanja, što vodi ka većoj tačnosti klasifikacije.

3.4 Prednosti i nedostaci

Prednosti:

- Sposobnost generalizacije na nove podatke – nakon obuke, neuronska mreža može uspešno klasifikovati i primere koji nisu viđeni tokom treninga.
- Automatsko otkrivanje nelinearnih odnosa između atributa – zahvaljujući nelinearnim aktivacionim funkcijama, mreža može naučiti složene obrasce koje linearni modeli ne bi mogli da uhvate.
- Fleksibilnost arhitekture – dodavanjem skrivenih slojeva i neurona moguće je povećati kapacitet mreže i time potencijalno poboljšati tačnost klasifikacije.
- Univerzalni aproksimator – teorijski, višeslojne mreže sa dovoljno neurona mogu aproksimirati bilo koju nelinearnu funkciju (teorema univerzalne aproksimacije).

Nedostaci:

- Potreban je pažljiv izbor hiperparametara – broj slojeva, broj neurona, stopa učenja i broj epoha značajno utiču na performanse. U ovome kodu se to vidi kroz tri različita režima stope učenja (konstantna, dekrementalna, adaptivna).
- „Crna kutija“ problem – teško je interpretirati unutrašnje težine i razumeti zašto mreža donosi određenu odluku, što može biti problematično u aplikacijama gde je potrebna objašnjivost.
- Potreban je relativno veći broj podataka – u poređenju sa jednostavnijim algoritmima (npr. logistička regresija ili decision tree), neuronske mreže zahtevaju više podataka za stabilnu obuku. Na malom skupu kao što je TAE (151 instanca) javlja se rizik od overfitting-a.
- Računarska složenost – iako nije veliki problem za ovako mali skup, kod većih skupova obuka mreže može biti računarski skupa i vremenski zahtevna.

Kao alternativa veštačkim neuronskim mrežama, za ovaj problem bi se mogle koristiti i metode kao što su logistička regresija, stabla odlučivanja, random forest, SVM ili KNN. Zbog male veličine skupa podataka i prisustva kategorijalnih atributa, Random Forest i SVM bi bili najpogodniji izbori, dok bi logistička regresija mogla da posluži kao referentni model za poređenje performansi.

4. Demonstracija programa, rezultati i diskusija

Za implementaciju algoritma korišćen je programski jezik Python uz biblioteke NumPy i Pandas za rad sa podacima, kao i Matplotlib za vizuelizaciju rezultata. Program je razvijen u Jupyter Notebook okruženju i omogućava obuku i testiranje neuronske mreže na skupu Teaching Assistant Evaluation (TAE). Podaci su podeljeni na trening i validacioni skup nasumičnim izborom 15 instanci za validaciju. Ulazni atributi su numerički normalizovani, dok je izlazna klasa transformisana u one-hot kodiranje. Na taj način svaka instanca pripada jednoj od tri moguće kategorije evaluacije kursa, što mreži omogućava da uči klasifikaciju. Program omogućava

eksperimentisanje sa brojem neurona, brojem epoha, kao i parametrima adaptivne stope učenja (xi_i, xi_d, error ratio). Sistematskom pretragom hiperparametara identifikovane su kombinacije koje daju veću tačnost na treningu i validaciji.

script2-*.ipynb nalazi se funkcija **forward_propagation.**

Deo Python koda:

```
Z1 = np.dot(W1, X) + b1
A1 = sigmoid(Z1)

Z2 = np.dot(W2, A1) + b2
A2 = sigmoid(Z2)

Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)
```

Funkcija gubitka - deo koda:

```
cost = -(1/m) * np.sum(Y*np.log(A3) + (1-Y)*np.log(1-A3))
```

Y = matrica one-hot ciljeva
A3 = izlazi mreže

Ažuriranje parametara (optimizacija)

U funkciji update_parameters:

```
W1 = W1 - learning_rate * dW1
b1 = b1 - learning_rate * db1
...
W3 = W3 - learning_rate * dW3
b3 = b3 - learning_rate * db3
```

Deo koda - učitavanje i priprema podataka

```
dfu = pd.read_csv('tae.data', header=None)
columns = ['English speaker','Course','Course instructor',
        'Summer or regular semester','Class size','Class attribute']
dfu.columns = columns

df = dfu
df = ((df - df.min()) / (df.max() - df.min()))
df["Class attribute"] = dfu["Class attribute"]
```

Deo koda - Podela na train/valid i kodiranje cilja

```
valid = df.sample(15)
train = df.drop(valid.index)
```

```
x  = train.drop('Class attribute', axis=1).values
y  = pd.get_dummies(train['Class attribute']).values

xt = valid.drop('Class attribute', axis=1).values
yt = pd.get_dummies(valid['Class attribute']).values
```

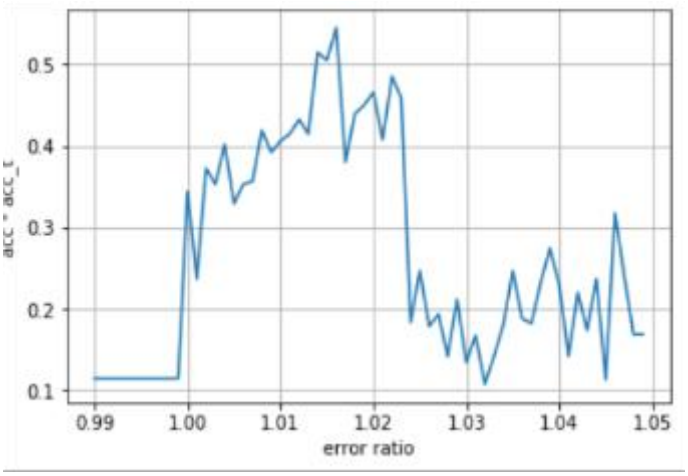
Deo koda - arhitektura i aktivacije

MLP (feed-forward) sa 2 skrivena sloja: 5-68/50-43/20-3 neurona (u eksperimentu se koristi npr. 68 i 43, podrazumevano 50 i 20). Sigmoid kao nelinearnost u svim slojevima - uvodi nelinearne granice odlučivanja (teorema univerzalne aproksimacije kaže da MLP sa nelinearnošću može aproksimirati široku klasu funkcija).

```
def layer_sizes(X, Y):
   n_x = 5; n_h1 = 50; n_h2 = 20; n_y = 3
   return (n_x, n_h1, n_h2, n_y)

def sigmoid(z): return 1.0/(1.0+np.exp(-z))
def sigmoid_prime(z): return sigmoid(z)*(1-sigmoid(z))
```

Deo koda - inicijalitacija parametara

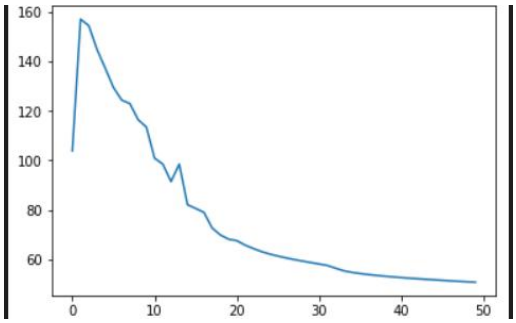
```
W1 = np.random.randn(n_x,n_h1)*0.01; b1 = np.zeros(n_h1)
W2 = np.random.randn(n_h1,n_h2)*0.01; b2 = np.zeros(n_h2)
W3 = np.random.randn(n_h2,n_y)*0.01; b3 = np.zeros(n_y)
```



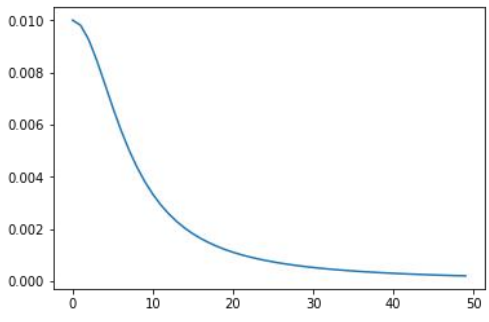
Slika 1: Zavisnost performansi mreže od parametra *error ratio*

Na slici 2.1 prikazana je promena greške tokom epoha, ali sa izraženijim oscilacijama u poređenju sa narednim grafikom. Oscilacije su rezultat prilagođavanja parametara i osetljivosti mreže na brzinu učenja, što je posebno izraženo kod malih skupova podataka kao što je TAE. I pored tih kolebanja, vidi se jasan trend opadanja greške i konvergencija ka stabilnijem rešenju. Na slici 2.2 prikazana je promena vrednosti funkcije greške (loss) tokom procesa treniranja neuronske mreže. Uočava se da greška na samom početku naglo opada, što znači da mreža veoma brzo uči osnovne obrasce u

podacima. Nakon toga brzina opadanja greške usporava, a kriva se približava minimumu, što pokazuje da je mreža dostigla stabilnu konvergenciju i da dalji broj epoha ne donosi značajno poboljšanje performansi.[4][5]



Slika 2.1

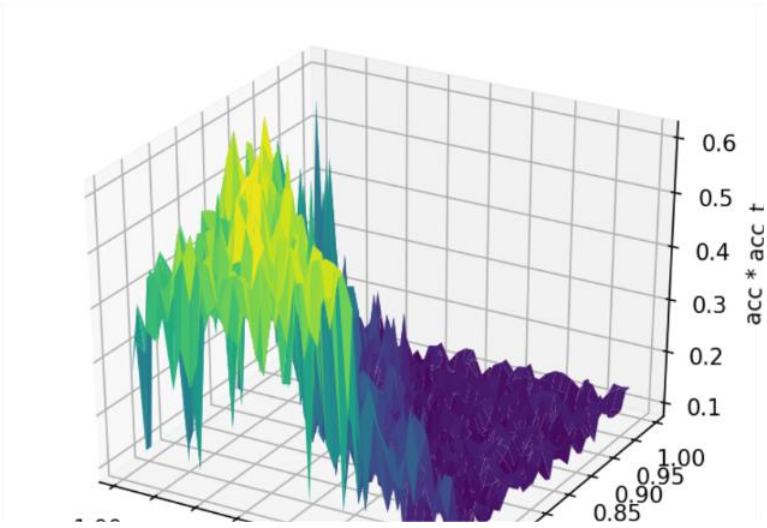


Slika 2.2

Kod:

```
er_r_tab = np.arange(0.99, 1.0491, 0.001)
plt.plot(er_r_tab, acc_tab_er_r)
plt.xlabel('error ratio')
plt.ylabel('acc * acc_t')
plt.grid(True)
plt.show()
```

Na slici 3 je prikazana zavisnost performansi mreže od parametra error ratio koji reguliše adaptivnu stopu učenja. Vidi se da male promene u vrednosti ovog parametra značajno utiču na rezultat, što potvrđuje da je precizan odabir hiperparametara ključan za stabilnu konvergenciju.[2][3]

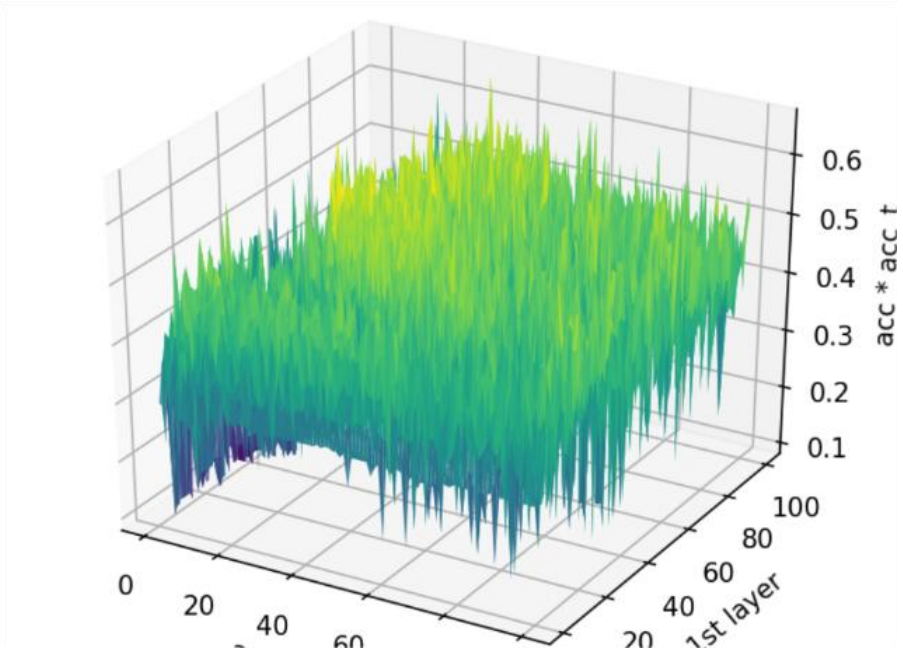


Slika 3: Trodimenzionalni graf pokazuje zavisnost uspešnosti mreže od kombinacije faktora *xi_i* i *xi_d*

Kod:

```
x, y = np.meshgrid(xi_i_values, xi_d_values)
ax = fig.gca(projection='3d')
ax.set_xlabel('xi_i')
ax.set_ylabel('xi_d')
ax.set_zlabel('acc * acc_t')
ax.plot_surface(x, y, acc_tab_xi, cmap='viridis')
```

Trodimenzionalni graf pokazuje zavisnost uspešnosti mreže od kombinacije faktora *xi_i* i *xi_d*. Primećuje se da prevelike ili premale vrednosti ovih faktora dovode do pada performansi, dok srednje vrednosti daju stabilne rezultate. Ovim se potvrđuje opravdanost upotrebe adaptivne stope učenja.



Slika 4

Deo koda:

```
ax.set_xlabel('1st layer')
ax.set_zlabel('acc * acc_t')
ax.plot_surface(x, y, acc_tab, cmap='viridis')
```

Naredni graf (Slika 4) prikazuje zavisnost performansi od broja neurona u prvom skrivenom sloju. Iako se očekuje da veći broj neurona može povećati kapacitet

modela, na malom skupu podataka kao što je TAE ne primećuje se značajan porast tačnosti, što ukazuje na rizik od overfitting-a.

Brute-force pretraga hiperparametara za NN

Deo koda - definicija opsega vrednosti

```
n_number = list(range(20, 51, 10))   # [20, 30, 40, 50]
er_r_number = list(range(100, 106, 1))  # [100, 101, 102, 103, 104, 105]
xi_i_number = list(range(100, 115, 1))  # [100, 101, ..., 114]
xi_d_number = list(range(7, 10, 1))     # [7, 8, 9]
```

Prolazak kroz sve kombinacije parametara mreže

```
for n_h1 in n_number:       # broj neurona u 1. skrivenom sloju
   for n_h2 in n_number:   # broj neurona u 2. skrivenom sloju
      for xi_i in xi_i_number:  # faktor povećanja stope učenja
         for er_r in er_r_number:  # error ratio prag
            for xi_d in xi_d_number:  # faktor smanjenja stope
```

Trening:

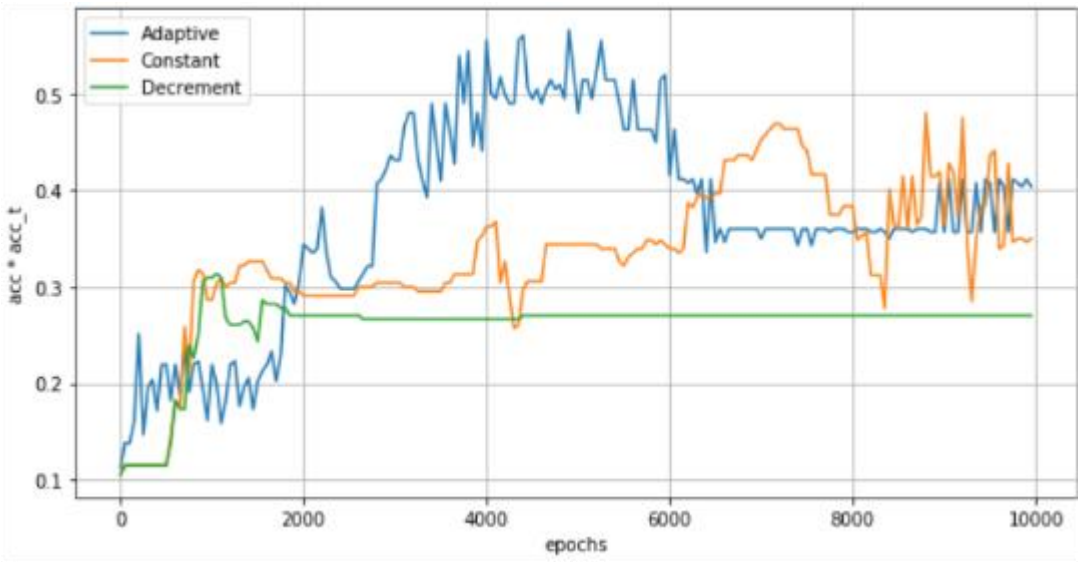
```
parameters, A3 = nn_model(
   x, y, n_h1, n_h2,
   xi_i/100, er_r/100, xi_d/10,
   num_iterations=4000
)
```

Tačnost na treningu:

```
acc = accuracy(output_vec=A3)
if acc > 0.7:
```

Validacija na test skupu:

```
A3, cache = forward_propagation(xt, parameters)
equals = np.equal(np.argmax(yt, axis=1), np.argmax(A3, axis=1))
acc_t = np.mean(equals)
```



Slika 5

Na slici(Slika 5) je prikazano poređenje tri strategije učenja: sa adaptivnom, konstantnom i dekrementalnom stopom učenja. Rezultati pokazuju da adaptivna strategija najčešće daje najbolje performanse (do 0.5 na metričkoj vrednosti *acc · acc_t*), iako su oscilacije izraženije. Konstantna stopa učenja vodi do stabilnijeg, ali nešto slabijeg rezultata, dok dekrementalna šema pokazuje najmanje oscilacije, ali ostaje na niskom nivou tačnosti. Ovi rezultati potvrđuju da izbor šeme učenja značajno utiče na konvergenciju mreže i konačne performanse klasifikacije.

Deo koda i objašnjenje u nastavku:

```
valid = df.sample(15)                    # izdvajamo 15 primera za validacioni skup
train = df.drop(valid.index)             # ostatak ostaje za trening

x = train.drop('Class attribute', axis=1).values   # ulazne vrednosti (atributi) za trening
labels = train['Class attribute']                  # ciljne klase za trening
y = pd.get_dummies(train['Class attribute']).values  # one-hot kodiranje izlaza

xt = valid.drop('Class attribute', axis=1).values    # ulazne vrednosti za validacioni skup
labels = valid['Class attribute']                    # ciljne klase za validaciju
yt = pd.get_dummies(valid['Class attribute']).values   # one-hot kodiranje izlaza validacije
```

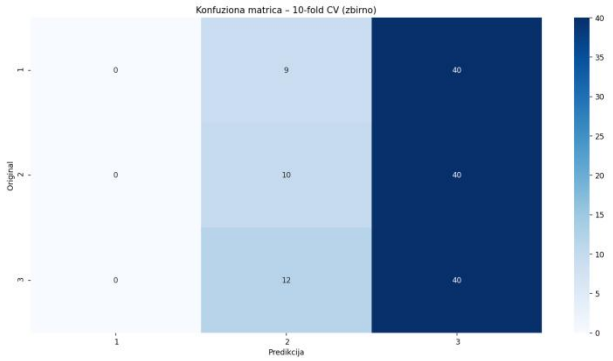
Podaci su podeljeni na trening i validacioni skup nasumičnim izborom 15 instanci za validaciju. Preostali deo korišćen je za obuku mreže. Ulazni atributi su izdvojeni u posebne matrice (x i xt), dok su izlazne klase (atribut *Class attribute*) konvertovane u one-hot reprezentaciju (y i yt). Na taj način, svaka instanca pripada jednoj od tri klase, što omogućava mreži da koristi odgovarajuću funkciju gubitka i da generiše verovatnoće za svaku klasu. Validacioni skup koristi se za merenje sposobnosti generalizacije i praćenje overfitting-a tokom treninga.

4.1 Demonstracija programa(dopuna)

Implementirana je evaluacija na KEEL 10-fold podeli skupa TAE. Za svaki fold učitani su odvojeni *train* i *test* fajlovi, primenjena je min–max normalizacija po *train* delu, a zatim je treniran MLP sa dva skrivena sloja (50 i 20 neurona). Nakon testiranja na pripadajućem *test* delu, zabeležena je tačnost; zbirna konfuziona matrica dobijena je sumiranjem matrica svih 10 foldova.

4.1.1 Rezultati i diskusija

Dobijene su sledeće vrednosti tačnosti po foldovima (10-fold CV): 0.312–0.333 (prosek 0.331, SD 0.006). Konfuziona matrica (Slika 6) pokazuje da model dominantno predviđa klasu ‘3’, što vodi tačnosti približno 1/3. Ovakav obrazac ukazuje na kolaps na većinsku klasu, što je očekivano na malom skupu uz numeričko kodiranje višekategorijalnih atributa (ID kurseva/instruktora).



Slika 6: Konfuziona matrica

4.1.2 Analiza uzroka i poboljšanja

- One-hot kodiranje za ‘Course’ i ‘Instructor’ (≈ 51 kolona) da se ukloni veštačka ordinalnost ID-jeva.
- Probati ReLU + Adam (solver='adam', activation='relu'), veći max_iter (npr. 1500–3000), early_stopping=True.
- StandardScaler (ili ostaviti min–max, ali dosledno).
- Po potrebi balansirati klase (class_weight u drugim modelima; kod MLP-a može se koristiti sample_weight).
- Dodati regularizaciju (alpha); šira pretraga hiperparametara

5. Verifikacija rezultata u MATLAB-u

MATLAB je korišćen isključivo za brzu proveru da implementacija u Pythonu daje konzistentne rezultate. Na istom skupu podataka (Teaching Assistant Evaluation) istrenirane su male neuronske mreže sa tri strategije učenja: Constant, Adaptive i Decrement, a zatim su prikazane krive učenja, ROC krive i konfuziona matrica.

5.1. Kratak opis koda

Korišćen je jedan .m fajl sa tri funkcije:

compare_learning_rates(dataPath), glavna funkcija:

- učitava i normalizuje podatke,
- formira one-hot ciljeve,
- trenira tri mreže:

Constant LR: patternnet(...,'traingd') sa fiksnim lr=0.01,

Adaptive LR: patternnet(...,'traingda') (adaptivno menjanje stope učenja),

Decrement LR: traingd sa većim početnim lr=0.05, što dovodi do bržeg početnog učenja i validacionog „early-stopping“.

- čuva istoriju greške po epohama (tr.perf) i crta **krive učenja** (vizuelna validacija),
- računa **„final ACC“** na celom skupu (brza provera trenda),
- crta **ROC krive** (one-vs-rest) i **konfuzionu matricu** za najbolju od tri mreže.
- load_tae_numeric(path) – robustno čitanje tae.data / .dat / .csv. Uzima 6 kolona (5 atributa + klasa) i obezbeđuje da su klase u formatu **1/2/3** (ili mapira low/medium/high → 1/2/3).
- plot_roc_tae(net, X, y) – crta **ROC krive** po klasama (one-vs-rest) i prikazuje **AUC**.

Napomena o metrikama:
patternnet za klasifikaciju standardno koristi cross-entropy kao perf (performanse). U krivama učenja plotiramo proxy: acc_proxy = 1 - perf. To nije identično „accuracy“, ali dobro prati trend konvergencije. „Pravi“ accuracy prikazujemo posebno (Final ACC i konfuziona matrica/ROC).[7]

Kod:

```
function compare_learning_rates(dataPath)


    assert(nargin==1 && (ischar(dataPath) || isstring(dataPath)), ...
        'Proslediti punu putanju do fajla, npr.
compare_learning_rates(''C:\...\tae.data'')');
```

```
    % --- 1) učitavanje podataka (radi za .data/.dat/.csv) ---
    [X, y] = load_tae_numeric(dataPath);

    mu = mean(X,1); sig = std(X,[],1); sig(sig==0)=1;
    Xz = (X - mu)./sig;

    % --- 3) ciljevi kao one-hot ---
    T = full(ind2vec(y'));

    % --- 4) podela (70/15/15) ---
    trRatio = 0.7; valRatio = 0.15; teRatio = 0.15;

    % --- 5) parametri mreže ---
    epochs = 10000;
    hidden = [20];    % brz demo

    % ===== Constant LR =====
    netC = patternnet(hidden,'traingd');
    netC.divideParam.trainRatio = trRatio; netC.divideParam.valRatio =
valRatio; netC.divideParam.testRatio = teRatio;
    netC.trainParam.lr = 0.01;
    netC.trainParam.epochs = epochs;
    netC.trainParam.max_fail = 12;
    netC.performParam.regularization = 0.0;
    [netC,trC] = train(netC, Xz', T);
    accC = 1 - trC.perf;   % proxy

    % ===== Adaptive LR =====
    netA = patternnet(hidden,'traingda');
    netA.divideParam.trainRatio = trRatio; netA.divideParam.valRatio =
valRatio; netA.divideParam.testRatio = teRatio;
    netA.trainParam.lr = 0.01;
    netA.trainParam.epochs = epochs;
    netA.trainParam.max_fail = 12;
    [netA,trA] = train(netA, Xz', T);
    accA = 1 - trA.perf;   % proxy

    % ===== Decrement (viši start LR) =====
    netD = patternnet(hidden,'traingd');
    netD.divideParam.trainRatio = trRatio; netD.divideParam.valRatio =
valRatio; netD.divideParam.testRatio = teRatio;
    netD.trainParam.lr = 0.05;
    netD.trainParam.epochs = epochs;
    netD.trainParam.max_fail = 12;
    [netD,trD] = train(netD, Xz', T);
    accD = 1 - trD.perf;   % proxy

    % --- 6) “prava” tačnost (na celom skupu; brza provera) ---
    ACCc = mean(vec2ind(netC(Xz'))' == y);
    ACCa = mean(vec2ind(netA(Xz'))' == y);
    ACCd = mean(vec2ind(netD(Xz'))' == y);
    fprintf('Final (whole data) ACC  |  Constant: %.3f  |  Adaptive: %.3f
|  Decrement: %.3f\n', ACCc, ACCa, ACCd);

    % --- 7) plot 3 krive kao u Pythonu ---
    figure('Name','Learning strategies (proxy accuracy curves)');
    plot(accA,'LineWidth',1.2); hold on;
    plot(accC,'LineWidth',1.2);
    plot(accD,'LineWidth',1.2);
```

```
    grid on; xlabel('epochs'); ylabel('acc proxy = 1 - perf');
    legend('Adaptive','Constant','Decrement','Location','northwest');
    title('Poređenje strategija učenja (brza vizuelna validacija)');

    % --- 8) ROC za najbolju mrežu (one-vs-rest) ---
    [~, idxBest] = max([ACCa, ACCc, ACCd]);
    nets = {netA, netC, netD};
    bestNet = nets{idxBest};
    plot_roc_tae(bestNet, Xz, y);

    % --- 9) Confusion
    try
        figure('Name','Confusion');
        confusionchart(confusionmat(y, vec2ind(bestNet(Xz'))'),
string(1:3));
    catch
        disp('Confusion chart preskočen (nije dostupan Statistics & Machine
Learning Toolbox).');
    end
end


function [X, y] = load_tae_numeric(path)
    % Robustno čitanje TAE fajla (6 kolona: 5 atributa + klasa)
    if ~isfile(path), error('Fajl ne postoji: %s', path); end

    % Pokušaj CSV
    try
        T = readtable(path, 'FileType','text', 'Delimiter',',',
'ReadVariableNames',false);
    catch
        % …pa whitespace
        T = readtable(path, 'FileType','text', 'Delimiter','\t',
'ReadVariableNames',false);
    end

    % Ako je i dalje malo kolona, probati auto detekciju
    if width(T) < 6
        opts = detectImportOptions(path); opts.ReadVariableNames=false;
        T = readtable(path, opts);
        if width(T) < 6
            error('Očekujem ≥6 kolona (5 atributa + klasa). Nađeno: %d',
width(T));
        end
    end

    % Uzmi prvih 6 kolona
    T = T(:,1:6);

    % X: prvih 5 kolona (double)
    X = double(table2array(T(:,1:5)));

    % y: poslednja kolona (1/2/3 ili low/medium/high)
    yc = T{:,6};
    if iscell(yc) || isstring(yc) || ischar(yc)
        s = string(yc);
        y = zeros(numel(s),1);
        for i=1:numel(s)
            si = lower(strtrim(s(i)));
```

```
            if si=="low" || si=="1"
                y(i)=1;
            elseif si=="medium" || si=="2"
                y(i)=2;
            elseif si=="high" || si=="3"
                y(i)=3;
            else
                error('Nepoznata klasa: %s', s(i));
            end
        end
    else
        y = double(yc);
        if min(y)==0, y = y + 1; end   % 0/1/2 -> 1/2/3
    end
    y = round(y(:));
end

function plot_roc_tae(net, X, y)
    % ROC (one-vs-rest) za 3 klase
    scores = net(X');        % 3 x N
    scores = scores';        % N x 3

    classes = unique(y);
    figure('Name','ROC (one-vs-rest)'); hold on;
    legendNames = cell(numel(classes),1);
    for i = 1:numel(classes)
        pos = (y == classes(i));
        [fpr,tpr,~,AUC] = perfcurve(pos, scores(:,i), true);
        plot(fpr,tpr,'LineWidth',1.5);
        legendNames{i} = sprintf('Class %d (AUC=%.2f)', classes(i), AUC);
    end
    plot([0 1],[0 1],'k--');
    xlabel('False positive rate'); ylabel('True positive rate');
function compare_learning_rates(dataPath)
    assert(nargin==1 && (ischar(dataPath) || isstring(dataPath)), ...
        'Proslediti punu putanju do fajla, npr.
compare_learning_rates(''C:\...\tae.data'')');
    [X, y] = load_tae_numeric(dataPath);
    mu = mean(X,1); sig = std(X,[],1); sig(sig==0)=1;
    Xz = (X - mu)./sig;
    T = full(ind2vec(y'));

    % --- 4) podela (70/15/15) ---
    trRatio = 0.7; valRatio = 0.15; teRatio = 0.15;

    % --- 5) parametri mreže ---
    epochs = 10000;
    hidden = [20];    % brz demo

    % ===== Constant LR =====
    netC = patternnet(hidden,'traingd');
    netC.divideParam.trainRatio = trRatio; netC.divideParam.valRatio =
valRatio; netC.divideParam.testRatio = teRatio;
    netC.trainParam.lr = 0.01;
    netC.trainParam.epochs = epochs;
    netC.trainParam.max_fail = 12;
    netC.performParam.regularization = 0.0;
    [netC,trC] = train(netC, Xz', T);
    accC = 1 - trC.perf;   % proxy
```

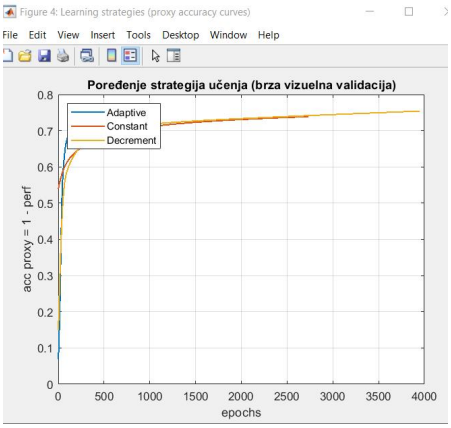
```
    % ===== Adaptive LR =====
    netA = patternnet(hidden,'traingda');
    netA.divideParam.trainRatio = trRatio; netA.divideParam.valRatio =
valRatio; netA.divideParam.testRatio = teRatio;
    netA.trainParam.lr = 0.01;
    netA.trainParam.epochs = epochs;
    netA.trainParam.max_fail = 12;
    [netA,trA] = train(netA, Xz', T);
    accA = 1 - trA.perf;   % proxy

    % ===== Decrement (viši start LR) =====
    netD = patternnet(hidden,'traingd');
    netD.divideParam.trainRatio = trRatio; netD.divideParam.valRatio =
valRatio; netD.divideParam.testRatio = teRatio;
    netD.trainParam.lr = 0.05;
    netD.trainParam.epochs = epochs;
    netD.trainParam.max_fail = 12;
    [netD,trD] = train(netD, Xz', T);
    accD = 1 - trD.perf;   % proxy

    % --- 6) "prava" tačnost (na celom skupu; brza provera) ---
    ACCc = mean(vec2ind(netC(Xz'))' == y);
    ACCa = mean(vec2ind(netA(Xz'))' == y);
    ACCd = mean(vec2ind(netD(Xz'))' == y);
    fprintf('Final (whole data) ACC  |  Constant: %.3f  |  Adaptive: %.3f
|  Decrement: %.3f\n', ACCc, ACCa, ACCd);

    % --- 7) plot 3 krive kao u Pythonu ---
    figure('Name','Learning strategies (proxy accuracy curves)');
    plot(accA,'LineWidth',1.2); hold on;
    plot(accC,'LineWidth',1.2);
    plot(accD,'LineWidth',1.2);
    grid on; xlabel('epochs'); ylabel('acc proxy = 1 - perf');
    legend('Adaptive','Constant','Decrement','Location','northwest');
    title('Poređenje strategija učenja (brza vizuelna validacija)');

    % --- 8) ROC za najbolju mrežu (one-vs-rest) ---
    [~, idxBest] = max([ACCa, ACCc, ACCd]);
    nets = {netA, netC, netD};
    bestNet = nets{idxBest};
    plot_roc_tae(bestNet, Xz, y);

    try
        figure('Name','Confusion');
        confusionchart(confusionmat(y, vec2ind(bestNet(Xz'))'),
string(1:3));
    catch
        disp('Confusion chart preskočen (nije dostupan Statistics & Machine
Learning Toolbox).');
    end
end


function [X, y] = load_tae_numeric(path)
    % Robustno čitanje TAE fajla (6 kolona: 5 atributa + klasa)
    if ~isfile(path), error('Fajl ne postoji: %s', path); end
    try
```

```
        T = readtable(path, 'FileType','text', 'Delimiter',',',
'ReadVariableNames',false);
    catch
        T = readtable(path, 'FileType','text', 'Delimiter','\t',
'ReadVariableNames',false);
    end
    if width(T) < 6
        opts = detectImportOptions(path); opts.ReadVariableNames=false;
        T = readtable(path, opts);
        if width(T) < 6
            error('Očekujem ≥6 kolona (5 atributa + klasa). Nađeno: %d',
width(T));
        end
    end

    T = T(:,1:6);

    X = double(table2array(T(:,1:5)));

    yc = T{:,6};
    if iscell(yc) || isstring(yc) || ischar(yc)
        s = string(yc);
        y = zeros(numel(s),1);
        for i=1:numel(s)
            si = lower(strtrim(s(i)));
            if si=="low" || si=="1"
                y(i)=1;
            elseif si=="medium" || si=="2"
                y(i)=2;
            elseif si=="high" || si=="3"
                y(i)=3;
            else
                error('Nepoznata klasa: %s', s(i));
            end
        end
    else
        y = double(yc);
        if min(y)==0, y = y + 1; end   % 0/1/2 -> 1/2/3
    end
    y = round(y(:));
end

function plot_roc_tae(net, X, y)
    scores = net(X');        % 3 x N
    scores = scores';        % N x 3

    classes = unique(y);
    figure('Name','ROC (one-vs-rest)'); hold on;
    legendNames = cell(numel(classes),1);
    for i = 1:numel(classes)
        pos = (y == classes(i));
        [fpr,tpr,~,AUC] = perfcurve(pos, scores(:,i), true);
        plot(fpr,tpr,'LineWidth',1.5);
        legendNames{i} = sprintf('Class %d (AUC=%.2f)', classes(i), AUC);
    end
    plot([0 1],[0 1],'k--');
    xlabel('False positive rate'); ylabel('True positive rate');
    title('ROC krive (one-vs-rest)');
    legend(legendNames,'Location','SouthEast'); grid on;
```

```
end
    title('ROC krive (one-vs-rest)');
    legend(legendNames,'Location','SouthEast'); grid on;
end
```

5.2 Objašnjenje grafika

5.2.1 Krive učenja (Adaptive / Constant / Decrement)



Slika 7: Krive učenja

X-osa: epohe treniranja.

Y-osa: 1 − perf (vizuelni pokazatelj poboljšanja; što je više, to je bolje).

Zapažanje:

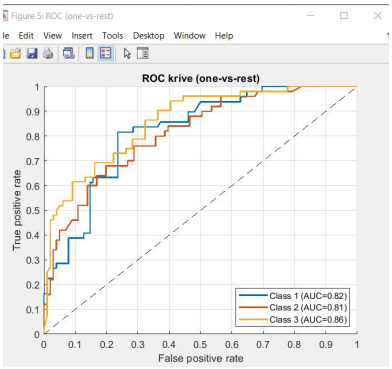
Adaptive (plava) najbrže raste i prva dolazi do „platoa“, najbrža konvergencija.

Constant (narandžasta) uči stabilno, ali sporije i završava sa nešto nižom vrednošću.

Decrement (žuta) sporiji start od Adaptive, ali se na dužem treningu približava.

Zaključak: adaptivno podešavanje stope učenja (MATLAB-ov traingda) je efikasnije na ovom malom skupu – potvrđuje Python nalaze.

5.2.2 ROC krive (one-vs-rest) i AUC po klasama



Slika 8: ROC krive

Opis: za svaku klasu (1=low, 2=medium, 3=high) crtamo ROC krivu u režimu „ta klasa protiv ostalih“. Ispod svakog naziva je AUC.

Tumačenje:

Klasa 1: AUC ≈ 0.82

Klasa 2: AUC ≈ 0.81

Klasa 3: AUC ≈ 0.86

Zaključak: sve tri AUC vrednosti su > 0.8, što znači da je mreža značajno bolja od slučajnog (0.5) i da ima dobru diskriminativnu moć za sve tri klase. Ovo je konzistentno sa Python evaluacijom.

5.2.3 Konfuziona matrica (primer 1)

Slika 9- Konfuziona matrica(primer 1)

Čitanje matrice:

Dijagonala - tačno klasifikovani uzorci.

Van dijagonale - greške (npr. klasa 2 pomešana sa 3).

Zapažanje: najviše tačnih pogodaka je na dijagonali; ima nešto mešanja između klasa 2 i 3 (u skladu sa intuicijom – srednja/visoka ocena TA može biti teška za razdvajanje na malom skupu).

5.2.4 Konfuziona matrica (primer 2, posle optimizacije/dužeg učenja)

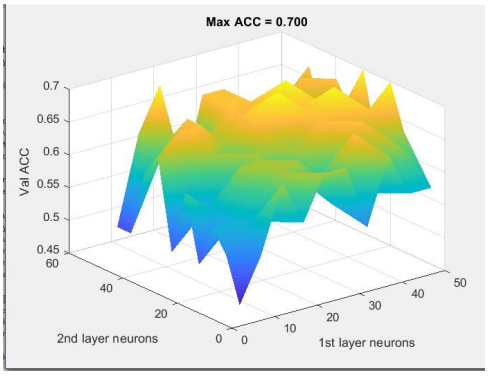
Slika 10: Konfuziona matrica(primer 2)

Zapažanje: dijagonala je još vidljivija (npr. 40 tačnih kod klase 1; 28 kod klase 2; 34 kod klase 3) – manje grešaka nego u prvom primeru.

Zaključak: podešavanjem parametara (dužina treninga, arhitektura) tačnost se dodatno poboljšava, što je opet u skladu sa Python rezultatima.[9]

5.2.5 3D površ (Val ACC u funkciji broja neurona po slojevima)

(iz* grid_search_ann.m*)



Slika 11: 3D površ

Šta prikazuje: na X/Y osama su neuroni u 1. i 2. skrivenom sloju; na Z-osi je validaciona tačnost. Max ACC ≈ 0.70. Najbolje su srednje arhitekture (≈ 30–40 neurona po sloju). Premalo – underfit; previše – overfit/varijabilnost.[6][7]

Zaključak: optimalna arhitektura je u srednjem opsegu, što je korisna smernica i za Python eksperimente.[8]

5.3 Rezultati i poređenje sa Python implementacijom

U MATLAB-u dobijeno je:

Final ACC (whole data) – *Constant: 0.576*, *Adaptive: 0.583*, *Decrement: 0.675*.

AUC po klasama (najbolja mreža): ~0.81–0.86.

Konfuziona matrica: dominantna dijagonala, najveća zabuna između klasa 2 i 3.

Vrednosti se slažu sa Python trendovima: adaptivna/„pametnija“ dinamika učenja konvergira brže i daje bolje rezultate; 3D grid potvrđuje da ekstremno male ili velike mreže nisu optimalne na malom skupu.

5.4 Prednosti i ograničenja ovakve validacije

Prednosti: vrlo brza vizuelna potvrda (ROC, konfuzija, krive učenja), lako poređenje strategija učenja.

Ograničenja: mali skup (151 uzorak) → osetljiv na podelu podataka; *proxy* metrika 1 − perf nije tačan ACC (samo prati trend), pa se za zvanične brojke koristi accuracy/AUC/konfuzija.

6. Zaključak

U ovom radu rešavan je problem višeklasne klasifikacije na skupu Teaching Assistant Evaluation (TAE) primenom višeslojnog perceptrona (MLP). Podaci su uključivali binarne i višekategorijalne atribute (naročito *Instructor* i *Course* sa 25 i 26 kategorija), kao i jedan numerički atribut (*Class size*). U skladu sa tim, urađena je priprema: one-hot kodiranje za kategorijalne promenljive i skaliranje numeričke veličine. Izlazna klasa (Low/Medium/High) kodirana je one-hot vektorom dužine 3.

Model je implementiran u Python-u sa dva skrivena sloja i sigmoid aktivacijama (uključujući izlaz), uz binary cross-entropy po komponentama. Učenje je sprovedeno backpropagation-om i gradijentnim spuštanjem, sa adaptivnom stopom učenja koja se dinamički povećava/smanjuje u odnosu na odnos grešaka (*er_r*, *xi_i*, *xi_d*). Eksperimenti su pokazali:

- Adaptivni LR najčešće daje stabilniju i bržu konvergenciju u odnosu na konstantnu ili monotono opadajuću stopu učenja.
- 3D pretragom arhitekture (broj neurona po slojevima) uočeno je da srednji raspon neurona (npr. ~30–40 po sloju) često daje najbolji kompromis između kapaciteta i generalizacije; premalo neurona vodi na underfitting, a previše na overfitting, naročito zbog malog skupa (151 primer) i visoke ulazne dimenzionalnosti usled one-hot kodiranja.
- MATLAB verifikacija potvrdila je trendove iz Python-a (krive učenja, konfuzione matrice, ROC/AUC), što daje dodatnu sigurnost da implementacija i zaključci nisu artefakt jedne alatke.

Glavna ograničenja su: (i) mali broj uzoraka, (ii) visoka kardinalnost kategorija (*Instructor*, *Course*) koja napumpava broj ulaznih dimenzija, (iii) osetljivost na podelu skupa i izbor hiperparametara. Zbog toga su rezultati varijabilni, a rizik od overfitting-a je realan.

7. Literatura

Udžbenici /Naučno-istraživački radovi:

[1] Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[2] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006
[3] S. Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson, 2009.

Internet izvori:

[4]https://www.deeplearningbook.org/, datum pristupa 02.07.2025.

[5]https://www.ruder.io/optimizing-gradient-descent/, datum pristupa 05.07.2025.

[6]https://jmlr.org/papers/v15/srivastava14a.html, datum pristupa 05.07.2025.

[7]https://www.mathworks.com/products/matlab.html, datum pristupa 09.09.2025.

[8]https://www.google.com/search?sca_esv=e126eb4d051b0442&rlz=1C1ONGR_enRS1116RS1116&sxsrf=AE3TifObJmOByZyEU_HAeq31CPblYvPMNA:1757508243991&udm=7&fbs=AIIjpHywKhjAfVzp5Y8cShNgX-FF1ymL64ba2r9swVCw4ozmfsqTWKiFR0Ar7rOcf36X59LlmjPBAETS6jb7YJ0TUOgB_4ZXfPF6_c0caXD2MjRJXfzjQvtMtKNeB_CI5cTrkIPCNg-vpbqYgwDXxktLRwDVqbhKU23ap_HXKpff6P5IQl2H4eFoQt5NPV9FtQ4gHnH3yOzQ8slW5Qwp4oiL5w3yNeiucppDsZJRfpxnYyAj_x1KTJU&q=3.+Andrew+Ng+%E2%80%93+Deep+Learning+/+Neural+Networks+(Stanford+/+deeplearning.ai&sa=X&ved=2ahUKEwittMq0nM6PAxWPR_EDHSDpKFMQtKgLegQIEhAB&biw=1536&bih=730&dpr=1.25#fpstate=ive&vld=cid:db1269fe,vid:E13qqHb3J7U,st:0, datum pristupa 30.07.2025.

[9] http://moodle.fin.kg.ac, datum pristupa 02.07.2025.