Fakultet kemijskog inženjerstva i tehnologije
Zavod za mjerenja i automatsko vođenje procesa

**Metode umjetne inteligencije u kemijskom inženjerstvu**

Primjer razvoja i vrednovanja

modela neuronskih mreža

*Vodič kroz razvoj neuronskih mreža u Pythonu*

# Instalacija potrebnih paketa za rad u Pythonu

Za potrebe rada sa neuronskim mrežama u programskom jeziku Python, razvijen je jednostavan paket *Keras* u kojemu kroz intuitivan način ispisa sintakse i programer početnik s teorijskim predznanjem neuronskih mreža, može ispisati kod te mijenjati parametre i hiperparametre modela neuronskih mreža.

U nastavku je prikazan primjer razvoja modela neuronske mreže na realnim podacima iz naftne industrije.

Za početak će se u *Anaconda Navigatoru* pokrenuti *CMD.exe Prompt* što je crni prozor poznatiji kao *Command Prompt* u kojemu će se upisati sljedeća sintaksa:

* *pip install ime\_paketa*

Instalirajte sljedeće pakete:

**tensorflow, keras**

# Programski kod za razvoj neuronskih mreža

Za početak uvest će se potrebni paketi i moduli:

import pandas as pd

import numpy as np

from scipy import stats

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense

from keras.callbacks import EarlyStopping

Učitat ćemo podatke iz datoteke *MUI\_Primjer2.xls.*

Zapamtite, ovakvim kodom tražit će se datoteka isključivo na putu gdje je spremljen i sam programski kod. Ako se datoteka i kod ne nalaze na istome mjestu potrebno je u kod upisati put do datoteke.

(npr. *C:\Users\Korisnik\MUI\MUI\_Primjer2.xls*)

'''Load-anje podataka iz excel filea'''

Data = pd.read\_excel('/content/MUI\_Primjer2.xls', header = 0)

'''Odvajanje podataka na utjecajne varijable (Features) i praćene varijable (Target)'''

Y = Data['benz\_LR']

X = Data.drop('benz\_LR', axis=1)

''' Konverzija podataka u nizove '''

Y = Y.values

X = X.values

Normalizacija i skaliranje podataka bitan je dio predobrade podataka prije razvoja modela neuronskih mreža. Iako nije nužno potrebno predobrađivati podatke na ovakve načine, ono se sugerira na način da se prilagode podaci bilo na isti red veličine ili prema vrijednostima izlaznih nelinearnih funkcija.

''' Normalizacija i skaliranje podataka '''

Y = np.reshape(Y,(-1,1))

scaler = StandardScaler()

Y\_t = scaler.fit\_transform(Y)

MMscaler = MinMaxScaler()

X\_t = MMscaler.fit\_transform(X)

Određivanje skupa za učenje/skupa za vrednovanje iz cjelokupnih podataka također je bitan dio razvoja modela, jer bi skup za učenje trebao sadržavati kompletan raspon mogućih vrijednosti nekih podataka kako bi mogao simulirati modelom njihovo vladanje. Često se za učenje tj. trening set uzima što je više moguće podataka jer količina podataka može uvjetovati preciznost modela, dok je s druge strane nekad trening set ograničen, ovisno o tome koja je priroda problema kojega se pokušava riješiti.

U ovome primjeru od 10 tisuća dostupnih podataka prvih 80% je uzeto za trening/test set.

'''Odabir trening/test seta'''

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_t,Y\_t, train\_size=0.8, shuffle = True, random\_state = 42)

Iako postoji nekolicina načina kreiranja neuronske mreže unutar *Keras* paketa u nastavku je prikazan najčešći pristup.

U prvome redu inicijalizacija modela se odvija na primjeru gdje je varijabla model definirana kao *Sequential* što znači da će ovaj model imati linearno složene slojeve, od ulaznoga preko skrivenog sve do izlaznoga sloja.

Funkcijom *.add* (u primjeru *model.add()*) dodaju se slojevi u model. Odmah prvom *.add* naredbom dodaje se prvi **skriveni sloj** te se definira **koliko** će **neurona** imati, koja je **dimenzija ulaznog sloja**, te koja je **aktivacijska** funkcija.

Sloj se zove *Dense* te predstavlja standardni sloj umjetne neuronske mreže koji je duboko povezan s prethodnim i budućim neuronima u susjednim slojevima.

Na isti način dodaju se sljedeći skriveni ili izlazni sloj. Izlazni sloj će biti onaj koji je zadnji zapisan u nizu *.add* funkcija. U ovome primjeru definirana je jedna izlazna varijabla koju želimo naučiti, odnosno koju želimo modelom opisati, stoga izlazni sloj ima jedan neuron, dok mu je aktivacijska funkcija *linear* što znači da se izlazna varijabla iz zadnjega izlaznog sloja neće dodatno nelinearizirati.

Naredba *.compile* (u primjeru *model.compile()*) se dodaje nakon definiranja strukture neuronske mreže, a služi za određivanje **funkcije pogreške/cilja (**engl. ***Loss function)*** i **algoritma** učenja. Uz te dvije bitne informacije može se odrediti i veličina koja će se pratiti za vrednovanje modela na skupu za učenje.

Naredba *.fit* (u primjeru *history = model.fit()*) služi za podešavanje (fitanje) modela na trening podatcima, te se unutar naredbe obavezno postavljaju oni podaci predodređeni za trening (ulazi i izlazi), definira se *batch\_size* argument koji zapravo govori o količini podsetova danih trening podataka za koje će se optimirati unutrašnji parametri, a prema *epoch* tj. broju vrtnji po svim trening podacima po kojima se optimiraju. Povećanje jednoga i drugoga argumenta može povećati točnost modela, ali i uvelike usporiti cjelokupni proces, stoga je potrebno naći optimalne vrijednosti i *batch\_size* i *epoch* argumenata.

Uz navedene bitne argumente, mogu se dodati i još neki kao u primjeru gdje se argumentom *validation\_split­* trening set dodatno dijeli na trening i validation set, te *callbacks* argument koji može sadržavati neke od naprednih opcija prilikom učenja. U ovome primjeru pozvana je *EarlyStopping* naredba u kojoj je definirano da prati pogrešku na validacijskom setu, te ako se pogreška ne mijenja određeni broj puta (*patience*) da zaustavi učenje kako ne bi došlo do pretreniranja ili nepotrebno dugog treniranja bez jasnih napredaka.

Naredba *.fit*  ima i neke izlazne vrijednosti koje se mogu pratiti, stoga se, kao i u primjeru, ona može pozvati kroz varijablu koja će spremiti te podatke, u primjeru varijabla *history*. Ovo i nije nužno potrebno, te se naredba *fit* može napisati na isti način kao i *model.compile*, ali ovakav način nam omogućava da dodatno analiziramo podatke dobivene prilikom treninga.
Slično tome se ponaša i zadnja naredba *model.evaluate* koja se koristi za spremanje rezultata vrednovanja podataka na test setu podataka.

'''Razvoj modela'''

model = Sequential()

model.add(Dense(9, input\_dim = 5, activation = 'tanh'))

model.add(Dense(1, activation = 'linear')) # dodavanje izlaznog layera

model.compile(loss= 'mse', optimizer='Adam', metrics = 'mse') # definiranje loss funkcije, algoritma učenja i evaluacija modela na treningu

es = EarlyStopping(monitor = 'val\_loss', patience = 50)

history = model.fit(X\_train, y\_train,  batch\_size = 32, epochs = 1000, validation\_split = 0.2, callbacks=[es],  ) # fitanje modela na trening podatke

test = model.evaluate(X\_test, y\_test) #evaluacija modela na testnom skupu

Jedan od načina analize podataka prilikom učenja mreže jest vizualna analiza praćenja smanjenja funkcije cilja (*Loss function*) na trening i validacijskom setu, a po epohama. Prema tim rješenjima moguće je naći optimalan broj epoha kako trening ne bi bio predug, a da su rezultati prihvatljivi.

plt.figure(figsize=(10,2.5))

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss - trening')

plt.ylabel('Loss')

plt.xlabel('epoch')

plt.legend(['train','val'])

plt.show()

print('Evaluacija na testu %g mse' %test[0])

Zadnja naredba *.predict* (u primjeru Y\_train\_m **=** model**.**predict**(**X\_train**)** ) koristi prethodno naučen model za izračun vrijednosti izlazne varijable prema danim ulaznim varijablama. Ovime se lako može vizualno analizirati naučeni model. U ovom primjeru izračunate su vrijednosti izlazne varijable modela za trening set, test set i cjelokupni set ulaznih podataka. Također, prikazane su grafički te su izračunate vrijednosti korelacijskih faktora realnih rješenja te modelom dobivenih rješenja izlazne varijable.

Y\_train\_m = model.predict(X\_train)

Y\_test\_m = model.predict(X\_test)

Y\_mod = model.predict(X\_t)

plt.figure(figsize=(10,2.5))

plt.plot(y\_train)

plt.plot(Y\_train\_m)

plt.title('Trening podaci - predikcija')

plt.ylabel('normalizirani\_benzLR')

plt.xlabel('vrijeme')

plt.legend(['Realni','Model'])

plt.show()

plt.figure(figsize=(10,2.5))

plt.plot(y\_test)

plt.plot(Y\_test\_m)

plt.title('Test podaci - predikcija')

plt.ylabel('normalizirani\_benzLR')

plt.xlabel('vrijeme')

plt.legend(['Realni','Model'])

plt.show()

plt.figure(figsize=(10,2.5))

plt.plot(Y\_mod)

plt.plot(Y\_t)

plt.title('Svi podaci - predikcija')

plt.ylabel('normalizirani\_benzLR')

plt.xlabel('vrijeme')

plt.legend(['Realni','Model'])

plt.show()

Y\_train = np.reshape(y\_train,(-1,))

Y\_train\_m = np.reshape(Y\_train\_m,(-1,))

Y\_test = np.reshape(y\_test,(-1,))

Y\_test\_m = np.reshape(Y\_test\_m,(-1,))

Y\_t = np.reshape(Y\_t,(-1,))

Y\_mod = np.reshape(Y\_mod,(-1,))

Cor\_tren, \_ = np.corrcoef(Y\_train,Y\_train\_m)

Cor\_test, \_ = np.corrcoef(Y\_test,Y\_test\_m)

Cor\_sve, \_ = np.corrcoef(Y\_t,Y\_mod)

print ('Korelacijski faktor - trening: ', Cor\_tren[1])

print ('Korelacijski faktor - test', Cor\_test[1])

print ('Korelacijski faktor - sve', Cor\_sve[1])

Ako je, kao u primjeru, provedena neka vrsta predobrade podataka u vidu normalizacije ili skaliranja, potrebno je te vrijednosti „odskalirati“ odnosno inverzno transformirati kako bi poprimile svoje realne vrijednosti na realnom redu veličina.

''' vraćanje podataka u realni red veličine - inverse transform skaliranja'''

Y\_mod = np.reshape(Y\_mod,(-1,1))

Y\_mod\_unsc = scaler.inverse\_transform(Y\_mod)

plt.figure(figsize=(10,2.5))

plt.plot(Y\_mod\_unsc)

plt.plot(Y)

plt.title('Sve\_predikcija')

plt.ylabel('benzLR')

plt.xlabel('vrijeme')

plt.legend(['Realni','Model'])

plt.show()

**Napomena za daljnji rad**

U ovome primjeru prikazan je način razvoj modela neuronskih mreža pomoću **Keras** paketa unutar Python programskog jezika. Objašnjene su temeljne naredbe postavljanja parametara modela umjetnih neuronskih mreža. Ono što preostaje jest dobiti zadovoljavajuće dobar model mijenjanjem zadanih vrijednosti parametara, te dodavanjem dodatnih skrivenih slojeva.

Pokušajte dobiti što bolje korelacijske faktore mijenjanjem sljedećih parametara mreže:

* mijenjajte **broj neurona u skrivenom** sloju (ne manje od broja ulaznih varijabli (5)) od minimalnog do 50(ili više)
* mijenjajte **aktivacijsku funkciju** u skrivenom sloju (sigmoid, tanh i relu)
* promijenite **loss funkciju i algoritam učenja** (pogledajte dokumentaciju za vrste loss funkcija <https://www.tensorflow.org/api_docs/python/tf/keras/losses> i algoritama učenja <https://www.tensorflow.org/api_docs/python/tf/keras/optimizers> )
* promijenite argumente unutar ***EarlyStopping***funkcije
* mijenjajte ***batch\_size* i *epoch* vrijednosti** te zaključite koje bi bile optimalne vrijednosti dvaju navedenih argumenata te koliko su te vrijednosti utjecale na vrijeme učenja.
* dodajte **još jedan ili više skrivenih** slojeva te ponovite proceduru promjene broja neurona i aktivacijske funkcije unutar istih. Zaključite koliko su točnome rezultatu pridonjeli dodatni skriveni slojevi i jesu li uopće pomogli, te koliko je to utjecalo na vrijeme učenja.