Fakultet kemijskog inženjerstva i tehnologije
Zavod za mjerenja i automatsko vođenje procesa
Metode umjetne inteligencije u kemijskom inženjerstvu

Vodič kroz razvoj i vrednovanje modela neuronskih mreža u Pythonu

Prosinac, 2024.

Instalacija potrebnih paketa za rad u Pythonu

Za potrebe rada s neuronskim mrežama u Pythonu, razvijen je jednostavan i intuitivan alat Keras, koji omogućuje lako razumijevanje i implementaciju modela čak i početnicima s osnovnim predznanjem o neuronskim mrežama. Keras omogućuje korisnicima da na jednostavan način definiraju arhitekturu modela, postave njegove parametre i hiperparametre te prilagode model specifičnim potrebama.

U nastavku je prikazan primjer razvoja jednog takvog modela koristeći realne podatke iz naftne industrije.

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

- *pip install ime_paketa*

Instalirajte sljedeće pakete:

tensorflow, keras

Programski kod

Za početak će se uvesti 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 datoteka će se tražiti 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)

```
'''Loadanje podataka iz Excel file-a, reshape podataka u nizove''''''

Data = pd.read_excel('MUI_Primjer2.xls', header = 0)

ul1 = Data['T_ul'].values
ul2 = Data['T_dno'].values
ul3 = Data['T_sred'].values
ul4 = Data['p_sred'].values
ul5 = Data['m_ref'].values
izl1 = Data['benz_LR'].values


'''grupiranje ulaznih veličina u jednu varijablu'''

X = np.column_stack((ul1,ul2,ul3,ul4,ul5))
Y = izl1
```

Normalizacija i skaliranje podataka bitan je dio predobrade podataka prije razvoja modela neuronskih mreža. Iako nisu uvijek strogo nužni, preporučuju se kako bi se podaci prilagodili na isti red veličine ili prema vrijednostima izlaznih nelinearnih funkcija. Time se osigurava da ulazne varijable različitih veličina ne budu tretirane neproporcionalno, što može negativno utjecati na proces učenja modela.

```
''' Normalizacija i skaliranje podataka '''
```

```
scaler = StandardScaler()
Y_t = scaler.fit_transform(Y)

MMscaler = MinMaxScaler()
X_t = MMscaler.fit_transform(X)
```

Podjela podataka na skup za treniranje i skup za testiranje iz cjelokupnih podataka također je bitan dio, jer bi set za treniranje (učenje) trebao imati kompletan raspon mogućih vrijednosti nekih podataka kako bi mogao prepoznati obrasce i pravilno predviđati. Ova podjela osigurava da model može generalizirati i točno predvidjeti rezultate na neviđenim podacima. Često se za trening set uzima što je više moguće podataka jer količina podataka može uvjetovati kvalitetu modela.

U ovome primjeru od 10 000 podataka prvih 7 000 je uzeto za trening i test set.

```
'''odabir trening i test seta'''
X_tren = X_t[:7000,:]
Y_tren = Y_t[:7000,:]
```

Jednostavno odvajanje na četiri varijable za trening i test može se provesti *train_test_split()* funkcijom iz paketa *sklearn*, iako se može i „pješke".

```
''' odvajanje podataka na trening i test skupove'''
X_train, X_test, Y_train, Y_test =  train_test_split( X_tren, Y_tren,
train_size = 0.8 ,shuffle = False)
```

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

Inicijalizacija modela odvija se u 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 modelom predvidjeti, 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 arhitekture neuronske mreže, a služi za određivanje **funkcije pogreške/cilja (*Loss function*)** i **algoritma** učenja. Uz te dvije bitne informacije može se odrediti i statistička veličina koja će se pratiti za vrednovanje modela na trening setu.

Naredba *.fit* (u primjeru *history = model.fit()*) služi za podešavanje (fitanje) modela koristeći trening podatke, te se unutar te naredbe obavezno postavljaju podaci predodređeni za trening (ulazi i izlazi). Definira se i *batch_size*, argument koji određuje veličinu podsetova trening podataka za koje će se optimirati unutrašnji parametri. Optimizacija se provodi tijekom svake epohe, što predstavlja jedan prolaz kroz cijeli skup trening podataka. Povećanje jednoga i drugoga argumenta može pospješiti točnost modela, ali i usporiti cjelokupni proračun, stoga je potrebno naći optimalne vrijednosti *batch_size* i *epoch* argumenata. Uz navedene argumente, mogu se dodati i još neki kao u primjeru gdje se argumentom *validation_split* trening set dodatno razdijelio na trening i set za validaciju, te *callbacks* argument koji može sadržavati neke od naprednih opcija prilikom učenja. U ovom primjeru korištena je naredba *EarlyStopping*, kojom je definirano praćenje pogreške na validacijskom skupu podataka. Ako se pogreška ne smanji nakon određenog broja iteracija (zadanog parametrom *patience*), proces treniranja se zaustavlja kako bi se izbjeglo pretreniranje ili nepotrebno dugo treniranje bez značajnih poboljšanja.

Naredba *.fit* ima izlazne vrijednosti koje omogućuju praćenje različitih metrika tijekom procesa treniranja. Zbog toga se, kao u našem primjeru, ona može pozvati kroz varijablu koja će spremiti te podatke, u primjeru varijabla *history*. Iako nije nužno, ovakav pristup omogućuje dodatnu analizu podataka prikupljenih tijekom treniranja, za razliku od jednostavne primjene naredbe bez spremanja izlaza, poput načina na koji se koristi *model.compile*.

Slično tome, naredba *model.evaluate* omogućuje spremanje rezultata vrednovanja modela na testnom skupu podataka, čime se osigurava pregled točnosti i drugih metrika na neviđenim podacima.

```
'''Izrada modela neuronske mreže'''

model = Sequential()
model.add(Dense(9, input_dim = 5, activation = 'tanh'))
model.add(Dense(1, activation = 'linear')) # dodavanje izlaznog sloja
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
podacima
test = model.evaluate(X_test, Y_test) # evaluacija modela na testnom skupu
```

Jedan od načina analize podataka tijekom treniranja je vizualna analiza, kojom se prati smanjenje funkcije cilja (*Loss function*) na skupu za učenje i validaciju, a kroz epohe. Takva analiza omogućuje uvid u dinamiku učenja modela te pomaže u određivanju optimalnog broja epoha. Na taj način izbjegava se nepotrebno dugi proces učenja, a osiguravaju se prihvatljivi rezultati.

```
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 naučeni model za predviđanje vrijednosti izlazne varijable na temelju zadanih ulaznih varijabli. Ova naredba omogućuje vizualnu analizu performansi naučenog modela. U primjeru, model je korišten za predviđanje izlaznih vrijednosti na trening skupu, test skupu te na cjelokupnom skupu ulaznih podataka. Predviđeni rezultati modela su prikazani grafički, a dodatno su izračunate vrijednosti korelacija između stvarnih vrijednosti izlazne varijable (sadržaja benzena) i modelom predviđenih vrijednosti.

```
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, tijekom predobrade podataka primijenjena normalizacija ili skaliranje, potrebno je provesti inverznu transformaciju kako bi se predviđene vrijednosti vratile na svoje stvarne veličine i jedinice. Ova inverzna transformacija osigurava da rezultati budu interpretabilni u kontekstu stvarnih podataka i na stvarnom redu veličina.

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

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()
```

U ovom primjeru prikazan je postupak izrade neuronskih mreža korištenjem *Keras* paketa unutar programskog jezika Python. Objašnjene su osnovne naredbe za postavljanje parametara modela umjetnih neuronskih mreža. Preostaje optimizirati model prilagodbom početno zadanih hiperparametara, mijenjanjem broja neurona i slojeva te dodavanjem dodatnih skrivenih slojeva kako bi se postigao zadovoljavajući model s optimalnim performansama.

Pokušajte postići što bolje korelacijske faktore prilagođavanjem sljedećih parametara:

- mijenjajte **broj neurona u skrivenom sloju** (ne manje od broja ulaznih varijabli (minimalno 5)) do 50 (ili više)
- mijenjajte **aktivacijsku funkciju** u skrivenom sloju (sigmoid, tanh i relu)
- promijenite loss funkciju i optimizator - 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 i točnost modela.
- dodajte jedan ili više dodatnih skrivenih slojeva u mrežu te ponovite postupak promjene broja neurona i aktivacijske funkcije unutar tih slojeva. Zaključite jesu li i koliko točnosti modela pridonijeli dodatni skriveni slojevi, te koliko je to utjecalo na vrijeme učenja.