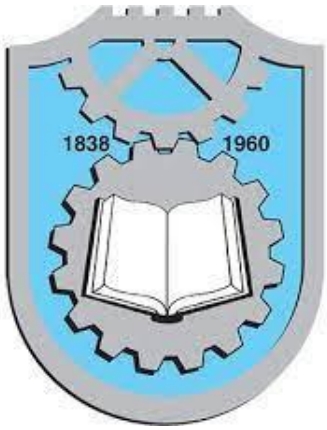
Факултет инжењерских наука
Универзитет у Крагујевцу



Četvrti domaći zadatak iz neuronskih mreža

- Implementacija LSTM arhitekture –

Student:

Ilija Todorović 313/2023

Predmetni nastavnik:

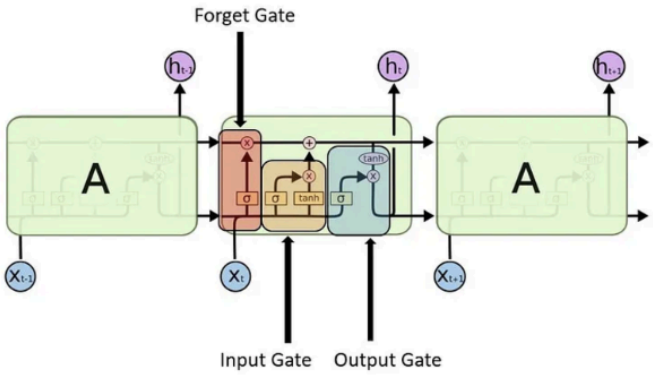
Dr. Vesna Ranković

Kragujevac, 2024

Sadržaj

Uvod

LSTM (Long Short-Term Memory) modeli su posebna vrsta rekurentnih neuronskih mreža (RNN) dizajnirana da efikasno uči sekvencijalne podatke i prevaziđe problem "nestajanja gradijenta", čest kod klasičnih RNN-ova. Ključna prednost LSTM-a leži u njegovim ćelijskim stanjima i strukturi sa "vratima" (gate) – ulaznim, izlaznim i zaboravnim, koja omogućavaju selektivno zadržavanje ili odbacivanje informacija tokom učenja. Ova arhitektura ih čini pogodnim za zadatke poput analize vremenskih serija, obrade prirodnog jezika (NLP) i predikcije vremenskih obrazaca.



slika 1 - Primer arhitekture LSTM-a

U daljem radu biće predstalvjena i objašnjena poglavlja iz knjige *Long Short-Term Memory Networks With Python | Develop Sequence Prediction Models With Deep Learning - Jason Bronwlee.*

Kod

Preprocesuiranje podataka

Jedna jako bitna stavka u pred-procesuiranju podataka jeste normalizacija podataka. Sledeći kod upravo to i radi. Normalizuje vrednosti data seta tako da one budu u opsegu od 0 do 1. [1]

```
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
data = [10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
series = Series(data)
print(series)
values = series.values
values = values.reshape((len(values), 1))
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
print("Min: %f, Max: %f" % (scaler.data_min_, scaler.data_max_))
normalized = scaler.transform(values)
print(normalized)
inversed = scaler.inverse_transform(normalized)
print(inversed)
```

Takođe još jedna bitna stavka je standardizacija podataka. Standardizacija je proces transformacije podataka tako da imaju srednju vrednost (mean) 0 i standardnu devijaciju (standard deviation) 1. To se postiže oduzimanjem srednje vrednosti od svake vrednosti u skupu podataka i deljenjem rezultata standardnom devijacijom. U kodu ispod prikazana je funkcija u pythonu koja to radi za nas [1].

```
from pandas import Series
from sklearn.preprocessing import StandardScaler
from math import sqrt
data=[1.0,5.5,9.0,2.6,8.8,3.0,4.1,7.9,6.3]
series=Series(data)
print(series)
values=series.values
values=values.reshape((len(values),1))
scaler=StandardScaler()
scaler=scaler.fit(values)
print("Mean: %f, StandardDeviation: %f"%(scaler.mean_,sqrt(scaler.var_)))
standardized=scaler.transform(values)
print(standardized)
inversed=scaler.inverse_transform(standardized)
print(inversed)
```

Kada se radi sa podacima koje treba klasifikovati, veoma često se koristi one-hot metoda. Ona predstavlja pretvaranje svih podataka u vektore čije se vrednosti sastoje od nula i jedinica. Primer: Pas [0, 1] , Macka [1 , 0]. S obzirom da se u primeru ispod u data setu javljaju 3 različite klase imaćemo trodimenzionalni vektor [1].

```
from numpy import array, argmax
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
data=["cold","cold","warm","cold","hot","hot","warm","cold","warm","hot"]
values=array(data)
print(values)
label_encoder=LabelEncoder()
integer_encoded=label_encoder.fit_transform(values)
print(integer_encoded)
onehot_encoder=OneHotEncoder(sparse=False)
integer_encoded=integer_encoded.reshape(len(integer_encoded),1)
onehot_encoded=onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
inverted=label_encoder.inverse_transform([argmax(onehot_encoded[0,:])])
print(inverted)
```

Postoje određene funkcije za padding, kada nemamo ravnomeran skup podataka, tj kada sekvenca ide [[1,2,3,4], [1,2,3],[1]]. Tada se moze primenjivati funkcija pad_sequences, koja dodaje nule na mesta nedostajucih vrednosti tako da svi skupovi u sekvenci budu jedanke velicine.

pad_sequences(sekvenca) - ce dodati nule na pocetak skupa (za primer iznad: [[1,2,3,4], [0,1,2,3] ,[0,0,0,1]]), ako zelimo da dodamo nule na kraj skupova u sekvecni u zagradu dodajemo uslov ‘post’: pad_sequences(sekvenca, **padding = ‘post’**).

Takodje postoji i nesto sto se zove Sequence Truncation, sto nam omogucava da skratimo duze sekvence. Implementira se na sledeci nacin:
pad_sequences(sekvenca, maxlen=2, truncating = “post”)
U ovom primeru maxlen oznacava velicinu, dok post znaci da ce se brisati svi clanovi sekvence nakon drugog u nizu. Post se moze zameniti sa ‘pre’, sto ce uraditi suprotno. [1]

Implementacija modela

LSTM model se kreira na sledeći način[1]:

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(2))
model.add(Dense(1))
```

Prvo se kreira prazan sekvencijalni model i potom se definišu 2 LSTM jedinice. Ovaj broj označava broj neurona tj. dimenziju izlaznog vektora. I dodaje se jedan Dense sloj koji uglavnom služi kao izlazni sloj.

```
data = data.reshape((data.shape[0], data.shape[1], 1))
```

Kod iznad transformiše oblik (shape) niza podataka kako bi bio u skladu sa očekivanim ulazom LSTM sloja.

```
model = Sequential()
model.add(LSTM(5, input_shape=(2,1)))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

U kodu iznad je dodato 5 LSTM memorijskih jedinica sa definisanim oblikom ulaznih podataka, i na kraj je dodata Sigmoidna aktivaciona funkcija koja pakuje izlaz u opseg od 0 do 1.

```
model.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])
```

Linija iznad služi za kompajliranje modela i u njoj se definišu optimizator, funkcija gubitka kao i metrika na osnovu koje će se evaluirati to je model dobro trenira.

```
model.fit(X, y, batch_size=32, epochs=100)
```

Model je potrebno i fitovati u ovoj liniji se definišu ulazi i izlazi, batch size tj. koliko će podataka model prihvatati odjednom kao i broj epoha.

Na kraju je potrebno izvršiti predikciju dobijenih rezultata.

```
predictions = model.predict(X)
```

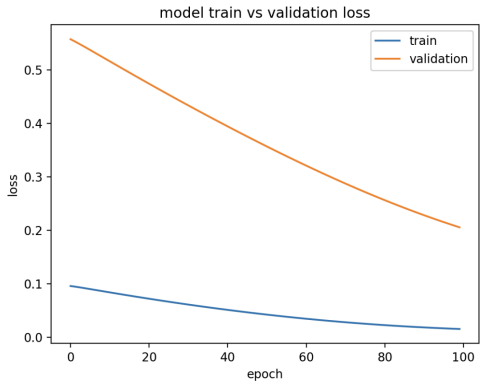
Primeri

U ovom delu rada bice predstavljena tri primera modela.Underfit model je model koji ima dobre performanse na trening skupu, a niske na test[1].

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from matplotlib import pyplot
from numpy import array

def get_train():
seq = [[0.0, 0.1], [0.1, 0.2], [0.2, 0.3], [0.3, 0.4], [0.4, 0.5]]
seq = array(seq)
X, y = seq[:, 0], seq[:, 1]
X = X.reshape((len(X), 1, 1))
return X, y

def get_val():
seq = [[0.5, 0.6], [0.6, 0.7], [0.7, 0.8], [0.8, 0.9], [0.9, 1.0]]
seq = array(seq)
X, y = seq[:, 0], seq[:, 1]
X = X.reshape((len(X), 1, 1))
return X, y

model = Sequential()
model.add(LSTM(10, input_shape=(1,1)))
model.add(Dense(1, activation= linear ))

model.compile(loss= mse , optimizer= adam )

X,y = get_train()
valX, valY = get_val()
history = model.fit(X, y, epochs=100, validation_data=(valX, valY), shuffle=False)
pyplot.plot(history.history[ loss ])
pyplot.plot(history.history[ val_loss ])
pyplot.title( model train vs validation loss )
pyplot.ylabel( loss )
pyplot.xlabel( epoch )
```

```
pyplot.legend([ train , validation ], loc= upper right )
pyplot.show()
```

To se može primetiti ako se pogleda plotovani grafik na kome se može videti da je gubitak na trening skupu manji od onog na validacionom.



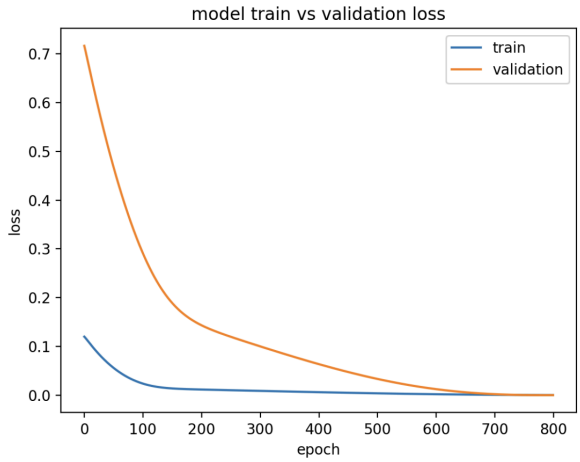
Slika 2 *- Grafik koji pokazuje underfit[1]*

Sledeći slučaj je kada je mreža dobar fit. Mreža je dobar fit kada su rezultati na trening i test skupu dobri[1].

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from matplotlib import pyplot
from numpy import array

def get_train():
seq = [[0.0, 0.1], [0.1, 0.2], [0.2, 0.3], [0.3, 0.4], [0.4, 0.5]]
seq = array(seq)
X, y = seq[:, 0], seq[:, 1]
X = X.reshape((5, 1, 1))
return X, y

def get_val():
seq = [[0.5, 0.6], [0.6, 0.7], [0.7, 0.8], [0.8, 0.9], [0.9, 1.0]]
seq = array(seq)
X, y = seq[:, 0], seq[:, 1]
X = X.reshape((len(X), 1, 1))
return X, y

model = Sequential()
model.add(LSTM(10, input_shape=(1,1)))
model.add(Dense(1, activation= linear ))
```

```
model.compile(loss= mse , optimizer= adam )
X,y = get_train()
valX, valY = get_val()
history = model.fit(X, y, epochs=800, validation_data=(valX, valY), shuffle=False)
pyplot.plot(history.history[ loss ])
pyplot.plot(history.history[ val_loss ])
pyplot.title( model train vs validation loss )
pyplot.ylabel( loss )
pyplot.xlabel( epoch )
pyplot.legend([ train , validation ], loc= upper right )
pyplot.show()
```

Dobri rezultati se mogu videti na grafiku ispod:

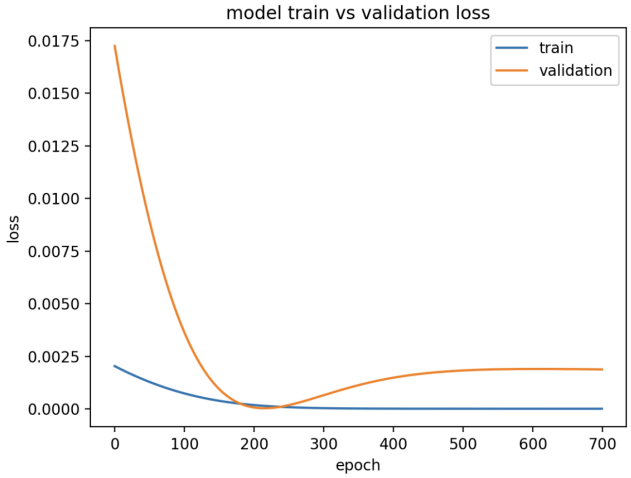


Slika 3 *- Grafik koji prikazuje dobar fit [1]*

Treći slučaj je overfit. Overfitovanje je stanje u kojem model previše dobro nauči šablone iz trening skupa, uključujući šum i nebitne detalje, što dovodi do lošeg performansa na novim, nepoznatim podacima [1].

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from matplotlib import pyplot
from numpy import array

def get_train():
seq = [[0.0, 0.1], [0.1, 0.2], [0.2, 0.3], [0.3, 0.4], [0.4, 0.5]]
seq = array(seq)
```

```
X, y = seq[:, 0], seq[:, 1]
X = X.reshape((5, 1, 1))
return X, y

def get_val():
seq = [[0.5, 0.6], [0.6, 0.7], [0.7, 0.8], [0.8, 0.9], [0.9, 1.0]]
seq = array(seq)
X, y = seq[:, 0], seq[:, 1]
X = X.reshape((len(X), 1, 1))
return X, y

model = Sequential()
model.add(LSTM(10, input_shape=(1,1)))
model.add(Dense(1, activation= linear ))

model.compile(loss= mse , optimizer= adam )

X,y = get_train()
valX, valY = get_val()
history = model.fit(X, y, epochs=1200, validation_data=(valX, valY), shuffle=False)

pyplot.plot(history.history[ loss ][500:])
pyplot.plot(history.history[ val_loss ][500:])
pyplot.title( model train vs validation loss )
pyplot.ylabel( loss )
pyplot.xlabel( epoch )
pyplot.legend([ train , validation ], loc= upper right )
pyplot.show()
```



Slika 4 *- Grafik koji prikazuje overfit [1].*

Zaključak

Implementacija LSTM mreža zahteva pažljivo balansiranje modela kako bi se postigla optimalna tačnost. Kroz prikazane primere underfit, dobro podešenog (good fit) i overfit modela, jasno se vidi kako različiti parametri i arhitekture utiču na performanse. Dok underfit model ne uspeva da uhvati ključne šablone u podacima, overfit model previše prilagođava šablone trening skupu, zanemarujući generalizaciju. Najbolji rezultati se postižu modelom koji ostvaruje balans između ovih krajnosti, čime se obezbeđuje stabilnost i primenljivost na nepoznate podatke. Ovi primeri naglašavaju značaj pravilnog podešavanja hiperparametra i procene performansi kako bi LSTM mreže bile uspešno implementirane u praktičnim zadacima.

Literatura

[1] - *Long Short-Term Memory Networks With Python | Develop Sequence Prediction Models With Deep Learning - Jason Bronwlee*