Универзитет у Крагујевцу
Факултет инжењерских наука

Анализа и пројектовање алгоритама

Пројектни задатак:

LSTM мрежа за класификацију реченица

Студент:
Огњен Павић 401/2021

Предметни наставник:
Проф. Владимир Миловановић

Садржај:

Увод

Задатак јесте креирање модела за класификаицју реченица на основу тога да ли је њихова конотација позитивна или негативна. Скуп података састоји се из коменатара остављених на сајту за оцењивање филмова.

Пошто је при класификацији реченица неопходно водити рачуна о редоследу речи, а не само скупу и броју речи у реченици, за решавање проблема користи се LSTM рекурентна неуронска мрежа, због своје способности рада са временски сензитивним подацима.

Модел је креиран у програмском језику python коришћењем библиотеке keras и њених алата за пројектовање секвенцијалних модела намењених дубоком учењу. Осим ове библиотеке коришћене су и библиотеке pandas, numpy и seaborn за обраду првобитног скупа података и каснију евалуацију перформанси добијеног модела.

Подаци

Приликом решавања проблема, неопходно је прво анализирати скуп података, пре почетка креирања модела за дубоко учење. Подаци су подељени у две категорије, реченице чији је смисао позитиван и реченице чији је смисао негативан. Оба скупа имају приближно подједнак број чланова.

Први проблем са скупом података јесте тај да су припадници различитих класа записани у различитим фајловима и да немају за себе везану никакву индикацију ком скупу припадају. Други проблем, који се истовремено може решити јесте постојање специјалних знакова који могу сметати приликом обучавања неуронске мреже.

На слици 1 приказан је начин решавања ових проблема. Првенствено се сви специјални знакови бришу. Затим се формирају листе реченица сегментисане контролним симболом за нови ред. На основу тога из ког фајла је реченица учитана добија се излазна вредност. На самом крају добијена очишћена реченица се уписује у табелу уз своју одговарајућу класу. Подаци се затим чувају у csv формату због једноставнијег учитавања и манипулације у каснијим деловима кода.

```
file=open("MR/rt-polarity.neg",'r')
negative=file.read()
file=open("MR/rt-polarity.pos",'r')
positive=file.read()

special=['.',',',':',';','=','+','-','*','/','\\','\'','\"','!','?','_','@','#','$','%','^','&','(',')','[',']','{','}']
for spec in special:
    negative=negative.replace(spec,"")
    positive=positive.replace(spec,"")


dataset=[]
output=[]

negativeS=negative.split('\n')
positiveS=positive.split('\n')

for x in negativeS:
    dataset.append(x.strip())
    output.append('negative')

for x in positiveS:
    dataset.append(x.strip())
    output.append('positive')


cols=['recenica','konotacija']
df=pd.DataFrame({'recenica' : dataset,
                 'konotacija' : output})
filename='recenice.csv'
df.to_csv(filename)
```

Слика 1. Формирање обједињеног скупа података

Следећи проблем на који се наилази везан је за чињеницу да је цео скуп састављен од великог броја различитих текстуалних података. Текстуални подаци се не могу користити директно за обучавање модела, па је потребно да се кодирају и претворе у нумеричке податке. Прво је неопходно пронаћи ширину вокабулара скупа података и дужину најдуже реченице у истом.

Приликом испитивања ширине вокабулара неопходно је креирати листу која ће имати улогу речника, а затим пролазити кроз све речи скупа. Уколико реч не постоји у речнику, додати је, у супротном је прескочити. Душина речника на крају представља број јединствених речи у скупу и износи 20520.

```
def vocabulary(X):
    reci=[]
    count=0

    for i in X:
        temp=i.split()
        for j in temp:
            if j not in reci:
                reci.append(j)
                count+=1
    return count
```

Слика 2. Испитивање величине вокабулара скупа података

Осим броја комбинација потребних за енкодирање сваке речи неопходно је знати и дужину најдуже реченице, због тога што улаз у мрежу мора бити предефинисане дужине и свака реченица мора бити допуњена до исте дужине. Најдужа реченица у скупу састоји се из 51 речи.

```
def maxWords(X):
    maxWords = 0
    for i in X:
        temp=i.split()
        length=len(temp)
        if length > maxWords:
            maxWords=length
    return maxWords
```

Слика 3. Израчунавање броја речи најдуже реченице у скупу

Користећи методе one_hot и pad_sequence, библиотеке keras почетни скуп података се коначно доводи у стање које је могуће искористити за тренирање мреже [1]. Свака реч у реченици кодира се као целобројна вредност, тако да је свака реченица представљена у облику вектора дужине једнаке броју речи у реченици. Сваки вектор је затим проширен одговарајућим бројем нула са десне стране, како би имао дужину најдужег вектора.

```
Xencoded=[one_hot(i,vokabular) for i in X]
Xpadded=pad_sequences(Xencoded,maxlen=maxLength,padding='post')
```

Слика 4. Коначна измена података [1]

Након што су све измене скупа података извршене, потребно је поделити скуп података на улазни и излазни део, а затим њих поделити на тренинг и тест скупове података.

```
split=0.8
length=len(data)

splitPoint=int(split*length)

trainx=Xpadded[0:splitPoint]
trainy=Y[0:splitPoint]
testx=Xpadded[splitPoint:length]
testy=Y[splitPoint:length]
```

Слика 5. Формирање тренинг и тест скупова података

Вредност split представља однос података тренинг и тест скупова. У овом случају 80% података узето је за обучавање, док осталих 20% формира тест скуп.

Креирање мреже

Модел је креиран помоћу библиотеке keras и њене имплементације различитих слојева неуронских мрежа.

Први слој јесте Embedding слој. Овај слој је неопходно користити у ситуацијама када се врши обучавање мреже са текстуалим подацима. Његов задатак јесте да научи тежинске коефицијенте за сваку реч прослеђеног речника. Неопходно је првенствено кодирати текстуалне податке као целобројне вредности. Такође, неопходно је дефинисати величину вокабулара, број димензија векторског простора у ком ће се вредности представљати, као величину улазних података, односно у овом случају максималан број речи у реченици [1].

```
def LSTMmodel(trainx, trainy, testx, testy,vokabular, maxLength):
    model=Sequential()
    model.add(Embedding(vokabular,32,input_length=maxLength))
    model.add(LSTM(5,batch_input_shape=(128,len(trainx),32), dropout=0.7, recurrent_dropout=0.7, activation='tanh'))
    model.add(Dense(1,activation='sigmoid'))
    #adam pokazuje bolje rezultate nego sgd
    opt=Adam(learning_rate=0.0001)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Слика 6. Креирање модела неуронске мреже

Други слој мреже је LSTM слој сачињен од 5 LSTM јединица. Приликом пројектовања овог слоја било је неопходно дефинисати параметре dropout и recurrent_dropout, како би се успорило обучавање. На излазу из слоја користи се активациона функција *'tanh'*, односно тангенцијална хиперболичка крива, чији је задатак да мапира вредности у домен од -1 до 1.

На излазу из мреже налази се један неурон са сигмоидном функциојом активације, због тога што је у питању проблем бинарне класификације.

Функција губитака модела је binary_crossentropy због бинарне природе проблема који мрежа треба да решава. Као оптимизатор користи се *'adam'* који је знатно спорији, али уједно и показује далеко боље резултате од оптимизатора *'sgd'*. Оптимизатор ради са подразумеваним вредностима параметара, због тога што се показало да њихова измена не утиче на коначне резултате.

Подешавање хиперпараметара

Како би се постигл бољи резултати, над мрежом је употребљен grid search за тражење најбољих вредности хиперпараметара.

За тестирање коришћен је модел приказан на слици 7. који је готово идентичан моделу приказаном у претходном поглављу, с тим што је вредности хиперпараметара мреже могуће проследити као аргумент. Тестиране су различите вредности броја епоха тренирања, величине скупа података који се доводе на улаз мреже и коефицијената заборављања.

```
def gridLSTM(trainx,trainy,testx,testy,vokabular,maxLength,epochs,batch,dropout):
    model=Sequential()
    model.add(Embedding(vokabular,32,input_length=maxLength))
    model.add(LSTM(1,batch_input_shape=(32,len(trainx),batch),dropout=dropout,recurrent_dropout=dropout))
    model.add(Dense(1,activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=epochs/10)
    history=model.fit(trainx,trainy,validation_data=(testx,testy),epochs=epochs,batch_size=batch, callbacks=[es])
    return history
```

Слика 7. Модел мреже намењен тестирању различитих хиперпараметара

Ван модела креиране су листе предефинисаних вредности хиперпараметара са којима треба тестирати модел. Затим се модел креира и тестира са сваком комбинацијом хиперпараметара, при чему се чувају метрике тачности и функције губитака на тренинг и тест скупу података. Приликом израде задатка велики проблем је представљало преучавање мреже, због чега су најбољи параметри бирани на основу минималне вредности фунцкије губитака.

```
epochsParams=[50,100,200]
batchParams=[32,64,128]
dropoutParams=[0.2,0.5,0.7]

epochs=[]
batch=[]
dropout=[]
acc=[]
loss=[]
vacc=[]
vloss=[]

for i in epochsParams:
    for j in batchParams:
        for p in dropoutParams:
            history=gridLSTM(trainx,trainy,testx,testy,vokabular,maxLength,i,j,p)
            acc.append(history.history['accuracy'])
            loss.append(history.history['loss'])
            vacc.append(history.history['val_accuracy'])
            vloss.append(history.history['val_loss'])
            epochs.append(i)
            batch.append(j)
            dropout.append(p)

#parametri se biraju po validation loss metrici (bira se trenutak kada je ona minimalna)
getResults(vloss,epochs,batch,dropout)
```

Слика 8. Добијање метрика тачности модела за сваку комбинацју параметара

Најбољи параметри изабрани су на основу редног броја комбинације приликом које је модел показао минималну вредност функције губитака. Резултат претраге добија се у виду фајла у ком су наведене вредности хиперпараметара које треба користити као и тачност коју је модел показао са њима.

```
def getResults(vloss,epochs,batch,dropout):
    minloss=100.0
    position=0
    for i in range(len(vloss)):
        current=vloss[i][len(vloss[i])-1]
        if(current<minloss):
            minloss=current
            position=i

    bestEpochs=epochs[position]
    bestBatch=batch[position]
    bestDropout=dropout[position]
    file=open("parametri.txt","w")
    outputString= "najbolji rezultati dobijeni su za parametre :\nepochs: "
    + str(bestEpochs) +"\nbatch: " + str(bestBatch) +"\ndopout: "
    + str(bestDropout) + "\ndobijena tacnost na trening skupu: "
    + str(round(vacc[position][len(vacc[position])-1],2))
    file.write(outputString)
    file.close()
```

Слика 9. Бирање најбољих параметара

Евалуација перформанси

Евалуација перформанси неуронске мреже посматра се у два одвојена случаја. Првенствено неопходно је пратити перформансе током процеса тренирања, а затим тестирати мрежу након што је модел креиран и обучен.

Током тренирања, мрежа је првобитно наилазила на проблем преучавања који је могао да се примети праћењем функције губитака. Наиме, након што је мрежа достигла стање у ком је тачност испитивања над тренинг подацима досегла 100%, функција губитака валидационог сета је расла до великих вредности. Из овог разлога, на мрежи је примењено рано заустављање, које прати кретање функције губитака над валидационим сетом и прекида обучвање онда када се досегне минимална вредност функције, односно онда када она почне да расте. Применом ове методе, мрежа идаље боље класификује тренинг податке, међутим тачност над тест подацима је веродостојнија.

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=0, save_best_only=True)
history=model.fit(trainx,trainy,validation_data=(testx,testy),epochs=100, batch_size=64,callbacks=[es,mc])
```

Слика 10. обучавање модела уз коришћење раног заустављања [2]

Осим раног заустављања, мрежа креира ModelCheckpoint, односно чува стање у ком је имала најнижу вредност функције губитака током обучавања. Након што се обучвање прекине, модел се враћа у ово стање пошто је у њему имао најбољу вредност тачности над валидационим скупом пре него што је ова функција дивергирала.

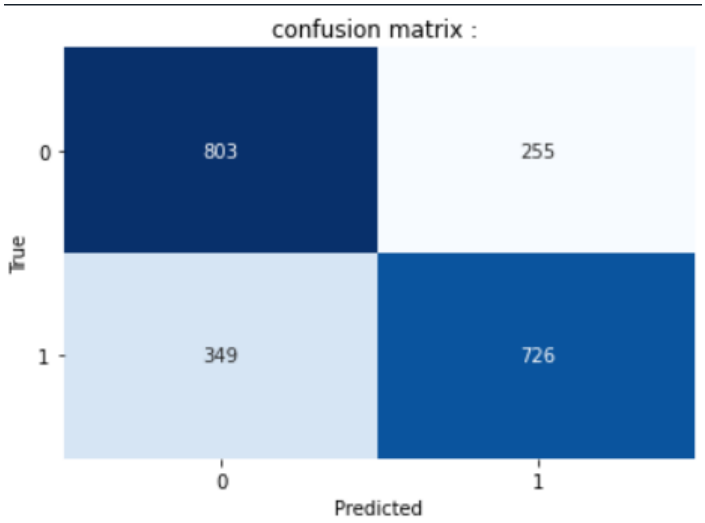
Након процеса тренирања, добијени модел се тестира са тест подацима и добијене вредности се упоређују са траженим вредностима. На овај начин се утврђују метрике тачности модела, коришћењем конфузионе матрице.

```
classes = np.unique(real)
fig, ax = pyplot.subplots()
cm = metrics.confusion_matrix(real, predicted, labels=classes)
sb.heatmap(cm, annot=True, fmt='d', cmap=pyplot.cm.Blues, cbar=False)
ax.set(xlabel="Predicted", ylabel="True", title=title)
ax.set_yticklabels(labels=classes, rotation=0)
pyplot.show()
```

Слика 11. Креирање конфузионе матрице

Коначни резултати

Коначни модел показује 72% тачности тренирања, с тим што мало боље класификује податке који припадају класи 0, односно реченице које имају негативну конотацију.

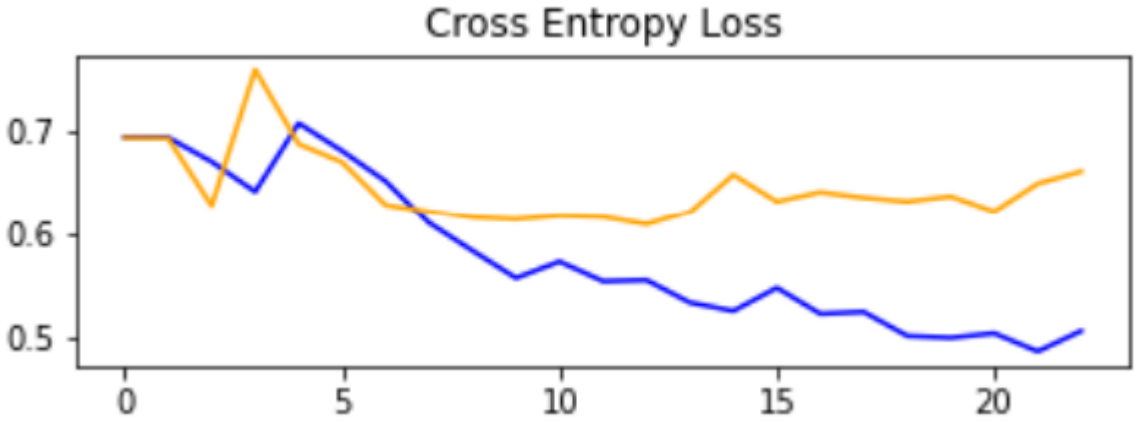


Слика 12. Конфузиона матрица

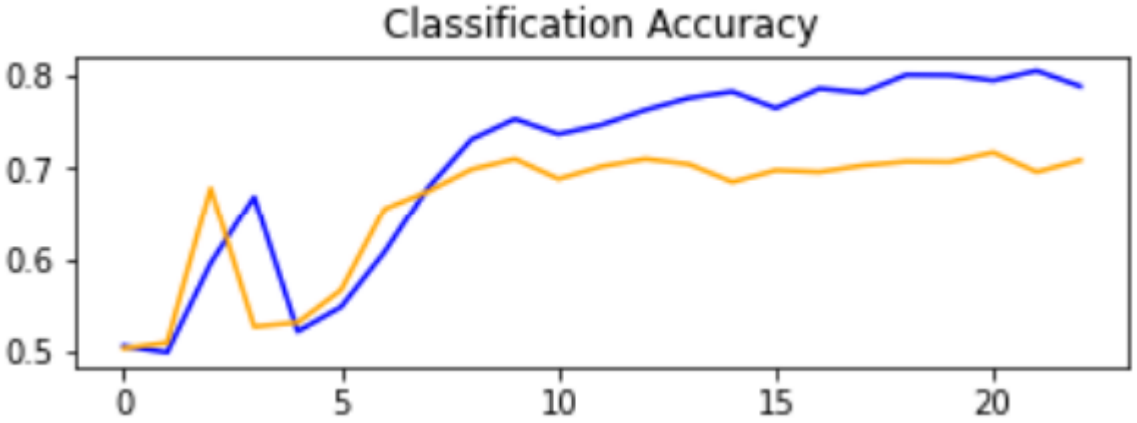
```
klasa1
senzitivnost: 0.76 specificnost: 0.68
PPV: 0.7 NPV: 0.74
f1 score: 0.73
preciznost: 0.72

klasa2
senzitivnost: 0.68 specificnost: 0.76
PPV: 0.74 NPV: 0.7
f1 score: 0.71
preciznost: 0.72
```

Слика 13. Добијене евалуационе метрике креираног модела

За оцењивање модела исправније је користити метрику прецизности због тога што су класе готово исте величине, међутим f1 score метрика даје приближно исте податке у овом случају. Метрика сензитивности класе 0 и специфичности класе 1 указују на то да модел боље класификује податке који припадају класи 0, односно да има више проблема са погађањем у случају када излаз не припада класи 0.



Слика 14. Функција губитака током тренирања модела



Слика 15. Кретање метрике тачности по епохама током тренирања

Када су у питању графици функције губитака и тачности током тренирања, ове функције над тренинг и тест сетом не конвергирају у потпуности. Међутим, пошто је проблем јако комплексне природе не може се очекивати 100% тачности над валидационим подацима. При крају процеса обучавања, након 20. епохе, графици полако почињу да дивергирају, али због постављених прекретница током тренирања, ова дивергенција неће имати утицаја на коначан модел, јер је најбоља вредност унапред сачувана.

Закључак

Добијени модел показује релативно добре перформансе приликом класификације реченица. Почетни проблеми лоше класификације и преучавања мреже пребројени су тестирањем разичитих комбинација хиперпараметара приликом тренирања мреже, као и коришћењем превременог заустављања и чувања најбољег модела током обуке. Претпоставка је да се перформансе модела могу унапредити проширењем почетног скупа података, али таквог да се поједине речи из доступног вокабулара, које се у примерима појављују мали број пута, користе чешће.

Такође, битно је напоменути да се вишекласна класификација реченица може извршити на сличан начин, с тим што би у том случају била неопходна измена излазног слоја који је прилагођен раду са само две класе, као и компајлирање модела коришћењем categorical_crossentropy или sparse_categorical_crossentropy јер је тренутна фунцкија губитака намењена искључиво бинарној класификацији.

Литература

[1] https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ (19.12.2021.)
[2] https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/ (23.12.2021.)