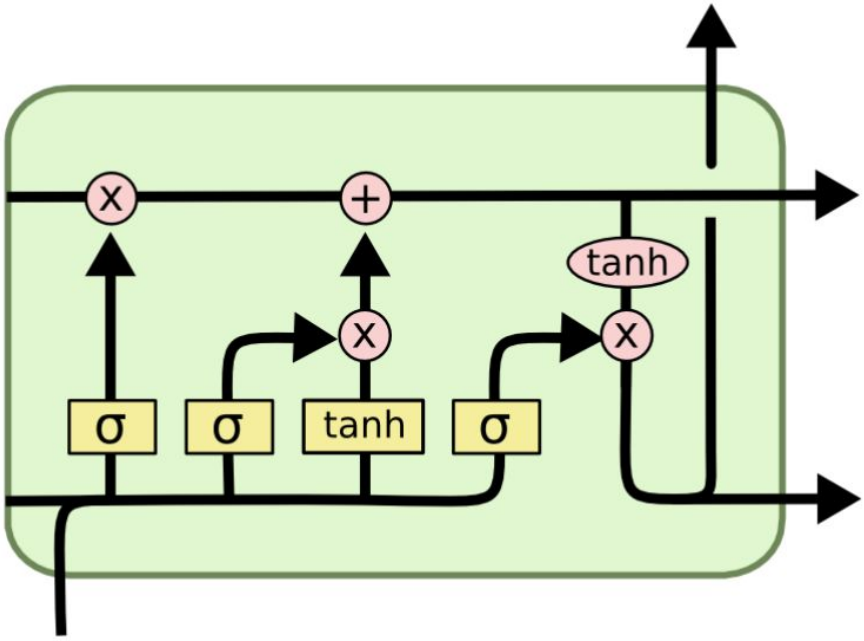
# گزارش پروژه چهارم یادگیری عمیق
# (LSTM)

**21 بهمن ماه 1399**

**غزاله محمودی**

96522249



## منابع :

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

https://github.com/masouduut94/Digikala_comments_verification

https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

(PDF) Investigation of the Effect of LSTM Hyperparameters on Speech Recognition Performance

What does SpatialDropout1D() do to output of Embedding() in Keras?

## لینک کدهای اجرایی:

- https://colab.research.google.com/drive/1uRSWa92QtplgFR_AH9lv5_ttgPygaTuT?usp=sharing

در این پروژه هدف این است به کمک embedding کلمات بر روی دیتاست دیجی کالا sentiment Analysis را انجام دهیم.

مراحل را به صورت زیر دنبال می کنیم. این قسمت مشابه کاری است که در پروژه قبل انجام شده است و برای تاکید و یادآوری مجدد آوردم.

با توجه به اینکه برچسب 1 به معنی نظر مثبت و -1 به معنی نظر منفی می باشد و صفر به معنا بدون لیبل گذاری است. بنابراین تصمیم گرفتم تنها label های 1 و -1 را نگه دارم و لیبل 0 را کنار بگذارم. و بدین ترتیب به ازای هر جمله ورودی به آن لیبل 1 یا -1 به معنای مثبت یا منفی بودن نظر می دهم. بنابراین با یک تسک binary classification روبرو هستیم.

در این قسمت لیبل های صفر را حذف کردم.

```
1  df = df[df.score != 0]
2  df.head()
```

| | score | comment |
|---|---|---|
| 5 | 1 | ... سلام من سه هفتس این گوشی رو خریدم واقعا عالیه |
| 6 | 1 | ...دیجی جون خدایی موجودش کن خیلی وقته منتظریم.. خ |
| 12 | 1 | همه چیزش معرکه ست تو این رنج قیمت مخصوصا (فورجیش) |
| 17 | 1 | ...خیلی عالیه بنظر من اگه این رو بخری خیلی سود کر |
| 20 | 1 | ...بابا تو رو خدا شگفت انگیزش کنین قبل عید این گو |

در ادامه با دیتاست را به نسبت 20 به 80 به داده های تست و آموزشی تقسیم می کنیم. این کار را به کمک train_test_split کتابخانه sklearn انجام دادم.

```
[ ]  1  x_train, x_test, y_train, y_test = train_test_split(df.comment, df.score, test_size=0.2)
```

```
[ ]  1  print(x_train.shape)
     2  print(x_test.shape)
```

```
(40013,)
(10004,)
```

در ادامه باید text ورودی را برای ورودی شبکه عصبی مناسب کنیم.
بنابر این مراحل زیر را بر روی داده های لغوی ورودی انجام می دهیم تا ورودی مناسب برای شبکه عصبی تولید شود.

- Tokenizer
  - توکن سازی فرآیند تبدیل داده های حساس به داده های غیر حساس به نام "توکن" است که می تواند در یک پایگاه داده یا سیستم داخلی مورد استفاده قرار گیرد بدون اینکه آنها را در دامنه خود قرار دهد. با جایگزینی داده های اصلی با مقدار نامرتبط با همان طول و قالب ، از رمزگذاری می توان برای ایمن سازی داده های حساس استفاده کرد. سپس نشانه ها برای استفاده به سیستم های داخلی سازمان ارسال می شوند و داده های اصلی در یک دیکشنری به ذخیره می شوند.
- Fit_to_text
  - در این قسمت به هر کلمه یک مقدار یکتای integer نسبت می دهیم. اعدادی که در این قسمت به vocab ها نسبت می دهیم همگی عددی بزرگتر از یک می باشند. عدد صفر هم برای padding در نظر میگیریم.
- Text_to_sequence
  - جمله ورودی را به دنباله ای از اعداد تبدیل می کند. این اعداد در واقع بع دیکشنری که در قسمت قبل تولید شده مپ می شوند. به جای ورود خوده کلمات به مدل عددی به نمایندگی آن ها به مدل وارد می شود.
- Pad_sequence
  - می دانیم ورودی شبکه عصبی طول ثابتی دارد. اما جملات طول متفاوتی دارند و این تناقضی به وجود می آورد. بنابراین برای جلوگیری از مشکلاتی که در ادامه به وجود می آید باید طول همه جملات یکسان باشد و بدین منظور به اول یا انتها جمله padding اضافه می کنیم که برابر مقدار صفر است و صرفا برای هم سایز شدن همه جملات است. طول padding را به حدی می گیریم که سایز جمله برابر سایز ماکزیمم جمله موجود در دیتاست شود.2 نوع padding به صورت های post , pre داریم که اولی به آخر جمله صفر اضافه می شود و در دومی به ابتدا جمله صفر اضافه می شود. برای یادآوری لازم به ذکر است که صفر همان padding است.

مراحلی که به اختصار در قسمت قبل توضیح دادم به صورت زیر پیاده سازی می شوند.
ابتدا برای جدا کردن لغات هر جمله با ابزار های موجود آن ها را tokenize می کنیم. سپس متن را بر روی دیتاست موجود fit می کنیم و در ادامه در تلاشیم هر جمله را به sequence مپ کنیم. در انتها باید طول sequence ها یکسان کنیم.

```
token = Tokenizer(num_words=n_unique_words, lower=False)
token.fit_on_texts(x_train)


train_sequences = token.texts_to_sequences(x_train)
train_sequences_matrix = pad_sequences(train_sequences, maxlen=max_text_length)


test_sequences = token.texts_to_sequences(x_test)
test_sequences_matrix = pad_sequences(test_sequences, maxlen=max_text_length)
```

ماکزیمم اندازه بردار را برابر اندازه بزرگترین جمله در نظر گرفتم.
به عنوان مثال برای جمله 'گوشی خوبیه' بردار متناظر مرحله text_to_sequence به صورت زیر است.

```
گوشیه خوبیه
[51, 52]
```

در نهایت چون باید سایز ورودی ها یکسان باشد، با پدینگ این اختلاف سایز را جبران می کنیم. دو نوع padding post , pre موجود است. در اینجا از 'pre' استفاده کردم.

```
گوشیه خوبیه
[51, 52]
[ 0  0  0 ...  0 51 52]
```

حال sequence تولید شده را به عنوان ورودی به شبکه دادم. این ورودی مناسب شبکه عصبی می باشد.

در اینجا طول بردار درون نگاری برابر با 100 در نظر گرفته شده است. ماکزیمم طول جملات را 2000 فرض کردم(با بررسی های انجام شده بلند ترین نظر طولی در حدودی 2600 دارد و من عدد 2000 را به عنوان بزرگترین طول در نظر گرفتم. جملات بزرگتر از آن قسمتی تز خود را از دست می دهند. جملات کوچکتر نیز تا رسیدن به طول 2000 پدینگ میگیرند)
همچنین در شبکه عصبی batch_size = 64گرفتم بدین معنی که هر 64داده یکبار وزن ها آپدیت می شود.

تعداد epoch = 4 می باشد بدین معنی که آموزش بر روی کل داده های آموزشی 4 بار تکرار می شود.
چون تسک binary classification است loss = 'binary_crossantropy' در نظر گرفتم. متریک را دقت و اپتیمایزر adam انتخاب شد.

در این مدل این پروژه از LSTM استفاده کردم.

## نتایج :

- LSTM به علت دارا بودن واحد های حافظه برای تحلیل کردن جملات طولانی بسیار مناسب است و تحلیل های بهتری ارائه می دهد.
- اگر از LSTM دو طرفه استفاده کنیم علاوه بر کلمات گذشته به کلمات بعد از کلمه اصلی هم برای آموزش دسترسی داریم و بدین ترتیب آموزش بهتر و دقت بالاتری داریم.
- در این قسمت با overfit مواجه شدم. همچنین گاهی با مینیمم محلی مواجه می شدم. برای حل مشکل از لایه drop out به دو صورت قبل و بعد LSTM , Classifier و همچنین در درون خوده LSTM انجام دادم.همان طور که در قسمت گذشته گفته شد.
- چالش اصلی که در این قسمت با آن مواجه شدم، عدم افزایش درست دقت است.
- استفاده از لایه dropout استفاده می‌کنیم که به طور تصادفی بخشی از ورودی به صفر را تعیین می‌کند. این لایه از overfitting شبکه جلوگیری می‌کند و یعنی جوری شبکه را توزیع می‌کند تا بر روی بخش‌های خاصی از ورودی تمرکز نکند.
- افزایش batch سایز موجب پیچیده شدن سیستم و overfitting می شود و می توان با کاهش این پارامتر از overfit جلوگیری کرد ولی در عین حال همگرا شدن مدل طولانی تر است.
- مقدار batch سایز در نحوه آموزش شبکه تاثیر گزار است. مقدار کم و زیاد ممکن است موجب عدم آموزش صحیح شود.
- با افزایش تعداد unit های LSTM دقت مدل افزایش می یابد.البته از یک تعدادی به بعد دقت ثابت می ماند و گویا مدل برای مسئله زیادی بزرگ است و همان مسئله بیش برازش پیش می یاید. باید این نکته را در نظر بگیریم که این عدد را مقداری قرار دهیم که موجب overfit شود.
- عملکرد شبکه در صورتی که 3 یا 4 لایه هیدن داشته باشیم افزایش می یابد و با افزایش بیش حد از این مقدار تاثیری ندارد و از جایی به بعد با افزایش تعداد لایه های پنهان بهبود عملکرد شبکه کم می شود.
- نتایج با 1 و 2و 3 لایه BILSTM به دست آمده است. تعداد unit ها 100 می باشد و batch size 128 می باشد.

- طبق نتایج بدست آمده optimizer عه Adam از همه بهتر عمل می کند و سریع تر است و تاثیر خوبی در روند یادگیری شبکه دارد. البته با تغییر learning rate و همچنین اپتیمایز RMS rope نتایج خوبی به دست می آید ولی SGD نتایج خوبی به دست نمی آورد.
- اگر drop out را هم بین لایه ها و هم به recurrent unit های LSTM بدهم نتایج بهتر است. به طور کلی استفاده از drop out تاثیر واضحی در عملکرد و جلوگیری از overfit دارد.
- اگر تعداد LSTM را ثابت نگه داریم، 2 تا Bidirectional LSTM پشت هم تاثیر مناسبی دارند. در کل افزایش تعداد لایه های LSTM تاثیر متوسطی در بهبود شبکه دارد.
- تعداد recurrent unit تا وقتی خیلی بزرگ یا کوچک نباشد تاثیر متوسطی در نتایج دارد. حدود 100 - 200 unit اعداد خوبی به نظر می رسد.

- مشاهده می شود که اگر از dropout درون LSTM استفاده کنیم، نتایج بهتری حاصل می شود و شبکه بهتر همگرا می شود نسبت به حالتی از لایه drop out استفاده کنیم. البته باید این نکته را ذکر کرد که به طور کلی استفاده از dropout زمان همگرا شدن شبکه را طولانی تر می کند. بنابراین با تعداد epoch برابر دقت پایین تر است و باید تعداد epoch بیشتری ران شود.
- می توان از CONV1D و Max Pooling بعد از embedding استفاده کرد تا دقت مدل بهتر شود و در واقع مفاهیم و فیچر های بهتری استخراج کند. به کمک CNN با وزن های کمتر در زمان سریع تر به نتایج مشابهی می رسیم.
- همان طور که قبلا هم ذکر کردم Bidirectional LSTM مناسب جملات طولانی است و با توجه به ویژگی که دارد جملات طولانی را بهتر یاد میگیرد و مفاهیم بهتری استخراج می کند. البته در نهایت هر دو می توانند عملکرد مشابهی داشته باشند.
- برای اینکه ورودی مناسب به شبکه بدهیم باید همه جملات طول برابر داشته باشند. به این منظور هم می توان جمله ها را به اندازه طولانی ترین جمله padding داد و هم می توان جملات را به قطعات کوچکتری تقسیم کرد ولی این کار موجب کاهش دقت می شود.
- با بررسی های انجام شده و سرچ و تحقیق اینطور به نظر می رسید که استفاده از spatial dropout 1D بعد از لایه embedding بسیار خوب و موثر است.

در این قسمت مدل های مختلفی تست شده است که به عنوان نمونه یکی از آن ها را می آوردم.

```
def LSTM_D_out():
  model = Sequential()
  model.add(Embedding(n_unique_words, n_dim, input_length=max_text_length))
  model.add(Dropout(0.2))
  model.add(LSTM(100))
  model.add(Dropout(0.2))
  model.add(Dense(1, activation='sigmoid'))
  return model
```

```
model = LSTM_D_out()
model.summary()
```

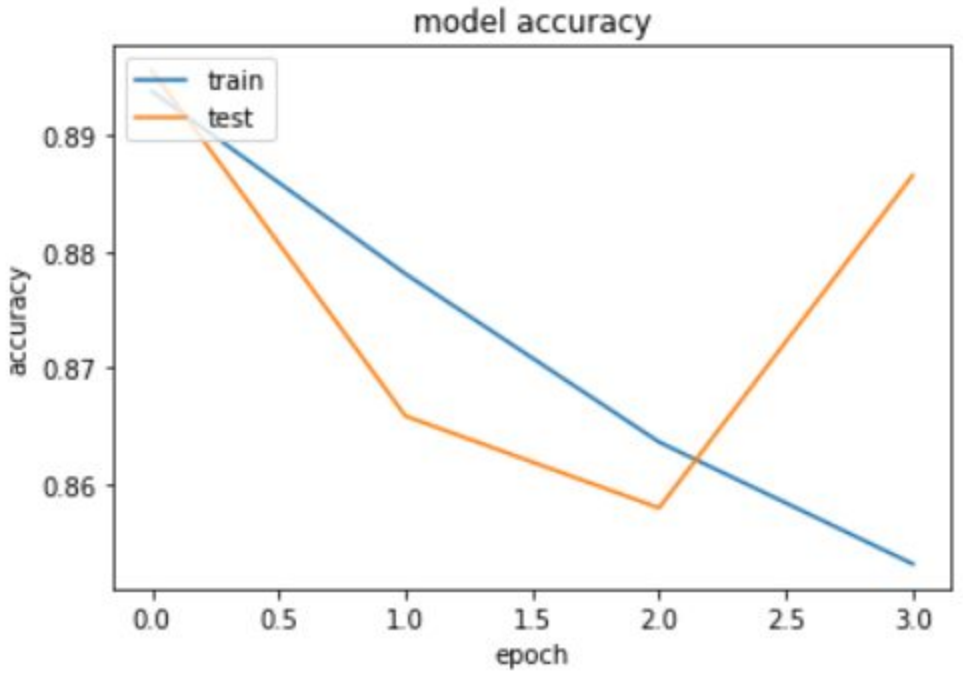
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 500, 64)           320000
_________________________________________________________________
dropout (Dropout)            (None, 500, 64)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               66000
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 386,101
Trainable params: 386,101
Non-trainable params: 0
_________________________________________________________________
```

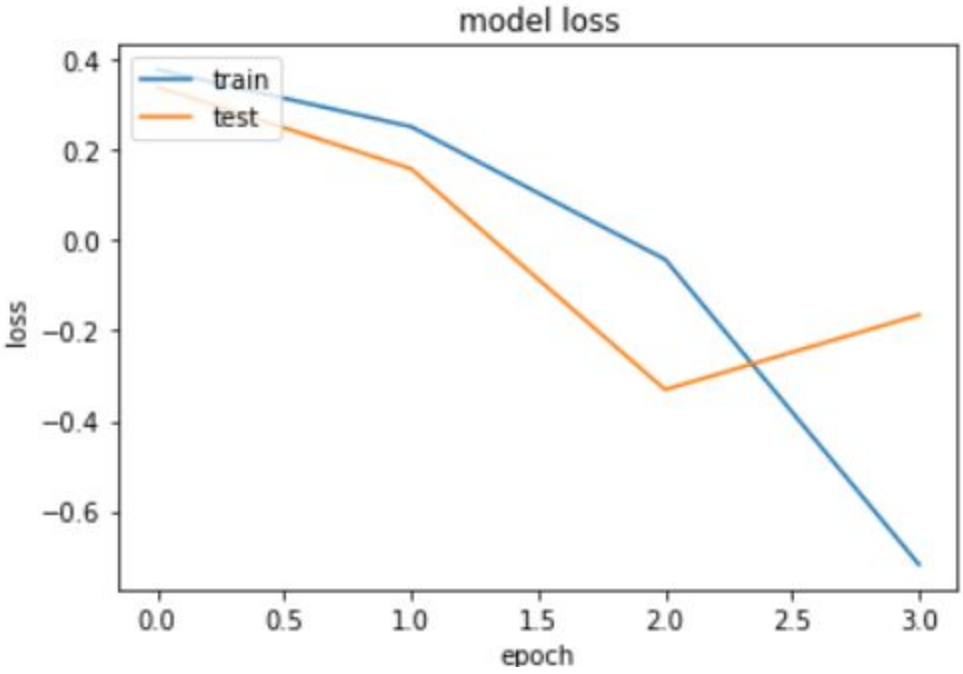
```
[40] model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

     history = model.fit(train_sequences_matrix, y_train,
               batch_size=batch_size,
               epochs=epochs,
               verbose=1,
               validation_data=(test_sequences_matrix, y_test))
```

```
Epoch 1/4
313/313 [==============================] - 16s 45ms/step - loss: 0.4463 - accuracy: 0.8934 - val_loss: 0.2987 - val_accuracy: 0.8846
Epoch 2/4
313/313 [==============================] - 14s 45ms/step - loss: 0.1898 - accuracy: 0.8765 - val_loss: 0.2232 - val_accuracy: 0.8168
Epoch 3/4
313/313 [==============================] - 14s 45ms/step - loss: -0.1090 - accuracy: 0.8631 - val_loss: -0.0835 - val_accuracy: 0.8609
Epoch 4/4
313/313 [==============================] - 14s 45ms/step - loss: -0.1785 - accuracy: 0.8631 - val_loss: -0.3234 - val_accuracy: 0.8797
```

نمودار loss , accuracy




```
[ ]  model.evaluate(test_sequences_matrix, y_test)
```

```
313/313 [==============================] - 9s 30ms/step - loss: -0.7627 - accuracy: 0.8525
[-0.7627005577087402, 0.8524590134620667]
```