

# بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

## فصل هفتم از کتاب

# Deep Learning Cookbook

### سرفصل مطالب

۴	.....مقدمه
۴	..... 1 ارائه مدلی برای پیشنهاد ایموجی به جای متن
۴	..... ۱-۱ ساخت یک طبقه بند احساسات ساده
۴	..... ۱-۱-۱ مسئله و راه حل
۵	..... ۲-۱-۱ دیتاست CrowdFlower
۷	..... ۳-۱-۱ بحث
۷	..... ۴-۱-۱ یادداشت
۸	..... ۲-۱ ارزیابی یک طبقه بند ساده
۸	..... ۱-۲-۱ نمایش نتایج
۹	..... ۲-۲-۱ بحث
۹	..... ۳-۱ استفاده از یک شبکه کانولوشن برای تحلیل احساسات
۱۰	..... ۱-۳-۱ روش One-hot و استفاده از CNN
۱۱	..... ۲-۳-۱ بحث

۴-۱	جمع آوری دیتای توئیتر.....	۱۱
۱-۴-۱	بحث.....	۱۴
۵-۱	یک پیش بینی کننده ایموجی ساده.....	۱۴
۱-۵-۱	بحث.....	۱۵
۶-۱ Dropout	و پنجره های چندگانه.....	۱۵
۱-۶-۱	بحث.....	۱۷
۷-۱	ساخت یک مدل در سطح کلمه.....	۱۷
۱-۷-۱	بحث.....	۱۸
۸-۱	ساخت embedding دلخواه خودمان.....	۱۹
۱-۸-۱	بحث.....	۲۰
۹-۱	استفاده از شبکه عصبی بازگشتی برای طبقه بندی.....	۲۱
۱-۹-۱	بحث.....	۲۱
۱۰-۱	مصور سازی توافقات.....	۲۲
۱۱-۱	نتیجه.....	۲۳
۱-۱۱-۱	بحث.....	۲۵
۱۲-۱	ادغام مدل ها.....	۲۶
۱-۱۲-۱	بحث.....	۲۷
۲	مراجع.....	۲۸

## فهرست شکل ها

شکل ۱:	روال کلی استخراج داده از توئیتر.....	۱۲
شکل ۲:	فرم مربوط به ثبت اپلیکیشن در توئیتر.....	۱۲
شکل ۳:	نمایش خروجی شبکه.....	۲۵

## فهرست جداول

جدول ۱:	نتایج مربوط به احساسات مختلف.....	۵
---------	-----------------------------------	---

- جدول ۲: تعداد حالات مختلف موجود در دیتاست ..... ۶
- جدول ۳: نمایش دقت های مختلف در طبقه بند ها ..... ۷

## مقدمه

هدف از این بخش، ایجاد مدلی است که به یک متن کوتاه ایموجی پیشنهاد بدهد. ابتدا کار را با یک طبقه‌بند ساده احساسات شروع می‌کنیم. این مدل براساس تعداد زیادی از توییت‌های برچسب خورده احساسات مانند خوشحالی، عشق، تعجب، و... ساخته شده‌است. در اولین مرحله، از یک طبقه‌بند بیزین برای درک عملکرد پایه استفاده کرده‌ایم. سپس کار با ایجاد یک شبکه کانولوشن و راه‌های متنوعی که می‌توانند کارایی این شبکه را بهبود ببخشند، ادامه می‌یابد. در ادامه، مشاهده می‌کنیم که چگونه می‌توان با استفاده از Twitter API به استخراج توییت‌ها پرداخت. سپس مدل کانولوشن ایجاد شده براساس دستورالعمل ۷،۳ را، قبل از رسیدن به یک مدل در سطح کلمه، اعمال می‌کنیم. در مرحله بعد، یک شبکه بازگشتی در سطح کلمه را ایجاد و اعمال کرده و مدل به دست آمده را، با سه مدل مقایسه می‌نماییم. در آخر، سه مدل را ادغام می‌کنیم تا بتوان از مزایای هر سه مدل بهره‌مند شد. مدل نهایی خیلی خوب کار کرده و برای یک اپ موبایلی بسیار کارآمد است. کد این قسمت در نوت‌بوک‌های زیر یافت می‌شود.

07.1 Text Classification

07.2 Emoji Suggestions

07.3 Tweet Embeddings

## ۱ ارائه مدلی برای پیشنهاد ایموجی به جای متن

### ۱-۱ ساخت یک طبقه‌بند احساسات ساده

#### ۱-۱-۱ مسئله و راه حل

##### مسئله

چگونه می‌توانیم احساسات موجود در یک متن را کشف کنیم؟

##### راه حل

ابتدا یک دیتاست از متون که با توجه به احساسات درون آنها برچسب خورده‌اند پیدا کرده و سپس یک طبقه‌بند ساده، روی آنها اعمال می‌کنیم.

قبل از این که یک عمل پیچیده را امتحان کنیم؛ بد نیست که ساده‌ترین کار را امتحان کنیم، ما می‌توانیم برای شروع از یک دیتاست آماده استفاده کنیم. در این جا می‌خواهیم یک طبقه‌بند احساسات ساده براساس دیتاستی که داریم بسازیم. در دستورالعمل پیش رو کار پیچیده تری را انجام می‌دهیم.

## ۱-۱-۲ دیتاست CrowdFlower

یک جستجوی سریع در گوگل ما را به دیتاست CrowdFlower می‌رساند. این دیتاست شامل توییت‌ها و برچسب‌های احساسی مربوط به آنهاست. از آنجایی که برچسب‌های احساسات به ایموجی‌ها شبیه هستند؛ این گام می‌تواند یک شروع خوب باشد. بگذارید فایل را دانلود کرده و نگاهی به آن بیندازیم.

```
import pandas as pd
from keras.utils.data_utils import get_file
import nb_utils

emotion_csv = get_file('text_emotion.csv',
                       'https://www.crowdflower.com/wp-content/uploads/2016/07/text_emotion.csv')
emotion_df = pd.read_csv(emotion_csv)
emotion_df.head()
```

نتیجه در جدول ۱: نتایج مربوط به احساسات مختلف، آمده است.

	tweet_id	sentiment	author	content
0	1956967341	empty	xoshayzers	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	wannamama	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	coolfunky	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	czareaquino	wants to hang out with friends SOON!
4	1956968416	neutral	xkilljoyx	@dannycastillo We want to trade with someone w...

جدول ۱: نتایج مربوط به احساسات مختلف

حتی می‌توانیم بفهمیم که هر احساس چه فراوانی در دیتاست دارد.

```
emotion_df['sentiment'].value_counts()
```

neutral	8638
worry	8459
happiness	5209
sadness	5165
love	3842
surprise	2187

جدول ۲: تعداد حالات مختلف موجود در دیتاست

بعضی از ساده‌ترین مدل‌ها، که بعضی اوقات در کمال تعجب نتایج بسیار خوبی نیز ارائه می‌دهند؛ از خانواده شبکه‌های بیزی هستند. ما ابتدا با استفاده از sklearn داده را کد می‌کنیم. TfidfVectorizer با توجه به معکوس تعداد تکرار، به کلمات وزن اختصاص می‌دهد؛ یعنی کلماتی که بیشتر تکرار می‌شوند وزن کمتری می‌گیرند چون بار معنایی کمتری دارند. LabelEncoder به هر برچسب یک عدد صحیح منحصر به فرد اختصاص می‌دهد.

```
VOCAB_SIZE = 50000

tfidf_vec = TfidfVectorizer(max_features=VOCAB_SIZE)
label_encoder = LabelEncoder()

X = tfidf_vec.fit_transform(emotion_df['content'])
y = label_encoder.fit_transform(emotion_df['sentiment'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)
```

با توجه به این اطلاعات، ما الان می‌توانیم یک مدل بیزی ساخته و آن را بسنجیم.

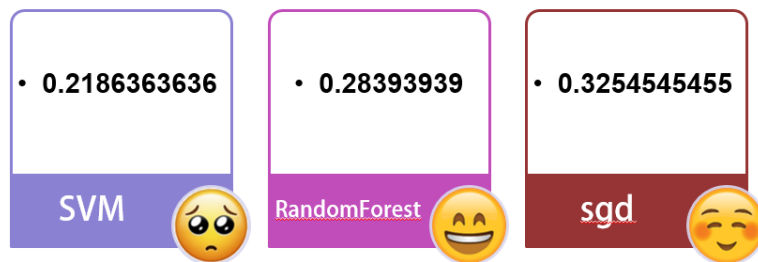
```
bayes = MultinomialNB()
bayes.fit(X_train, y_train)
predictions = bayes.predict(X_test)
precision_score(predictions, y_test, average='micro')
```

ما به دقت ۲۸ درصد رسیدیم. اگر ما همیشه بیشترین احتمال را پیش بینی می‌کردیم به دقتی کمی بالاتر از ۲۰ درصد می‌رسیدیم. لذا شروع خوبی بود. چند طبقه بند ساده دیگر هم وجود دارد که می‌توانند کمی بهتر عمل کنند، ولی سرعت کمتری دارند.

```
classifiers = {'sgd': SGDClassifier(loss='hinge'),
               'svm': SVC(),
               'random_forest': RandomForestClassifier()}

for lbl, clf in classifiers.items():
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
```

```
print(lbl, precision_score(predictions, y_test, average='micro'))
```



جدول ۳: نمایش دقت های مختلف در طبقه بند ها

### ۱-۱-۳ بحث

این که "ببینیم آیا با ساده ترین حالت نیز کار می کند" به ما کمک خواهد کرد تا شروع سریعی داشته باشیم و بفهمیم که داده آن قدر ظرفیت دارد که بتواند آن کاری که ما می خواهیم را انجام دهد.

طبقه‌بندهای بیزی در روزهای اولی که مسئله تشخیص ایمیل‌های اسپم مطرح شده بود، خیلی خوب کار کردند. اما این نوع طبقه‌بند، هر عامل را مستقل از عوامل دیگر در نظر می‌گیرد، بنابراین در این جا، هر کلمه در یک توییت، مستقل از کلمات دیگر، بر روی برجسب پیش بینی شده، موثر است؛ که کاملاً مشخص است در این مسئله، این اصل همواره برقرار نیست. به عنوان یک مثال ساده می‌توان دید که افزودن کلمه not به جمله می‌تواند احساس را برعکس کند.

از آنجایی که ساخت مدل آسان است، خیلی سریع نتایج را ارائه می‌دهد و نتایج قابل فهم نیز هستند. به عنوان یک قانون اگر یک مدل بیزی نتایج مناسبی را روی داده شما ندهد، استفاده از مدل پیچیده‌تر احتمالاً کمک زیادی به شما نکند.

### ۱-۱-۴ یادداشت

مدل‌های بیزی خیلی وقت‌ها بهتر از آنچه تصور می‌شود، کار می‌کنند. تحقیقاتی در رابطه با این که چرا این اتفاق می‌افتد؛ انجام شده است. قبل از یادگیری ماشین این مدل، به حل مسئله انیگما کمک کرد. کاربرد بعدی این شبکه در اولین نسل از تشخیص دهنده‌های ایمیل اسپم بود.

## ۱-۲ ارزیابی یک طبقه بند ساده

### مساله

چگونه می‌توانیم احساسات موجود در یک متن را کشف کنیم؟

### راه حل

یکی از مزایای استفاده از روش بیزی، فهم ساده آن می‌باشد. همانطور که در قسمت قبل توضیح دادیم؛ مدل‌های بیزی فرض می‌کنند که تاثیر هر کلمه در نتیجه مستقل از سایر کلمات است؛ لذا برای این که بفهمیم مدل‌مان چه یاد گرفته است: می‌توانیم نظر مدل را راجع به کلمات مستقل (تک کلمه) بپرسیم.

حالا به یاد بیاورید که، مدل انتظار یک سری از متون را دارد که هر کدام به صورت یک وکتور، که طول آن برابر به اندازه کلمه می‌باشد، کد شده است. هر کدام از عناصر وکتور بیانگر نسبت فرکانس رخداد کلمه مربوطه در این متن به رخداد آن در تمام متون است. بنابر آنچه بیان شد اگر یک مجموعه از متون داشته باشیم که در هر کدام از آن‌ها تنها یک و فقط یک کلمه مجزا آمده باشد، در این صورت مدل این مجموعه از متون، یک ماتریس مربعی خواهد بود که بر روی قطر اصلی آن عناصر ۱ قرار دارد و بقیه صفر هستند. این بدین معناست که برای متن شماره  $n$  تنها رخداد کلمه  $n$ ام مدل وجود دارد و بقیه کلمات در آن نیامده‌اند. حالا ما می‌توانیم احتمال انتساب برچسب برای هر کلمه را تخمین بزنیم:

```
from scipy.sparse import eye
d = eye(len(tfidf_vec.vocabulary_))
word_pred = bayes.predict_proba(d)
```

پس ما می‌توانیم با توجه به این پیش بینی‌ها، امتیاز کلمه برای هر کلاس را به دست آوریم. سپس این اطلاعات را در Counter ذخیره می‌کنیم؛ پس به راحتی می‌توانیم به کلماتی که بیشترین تاثیر را دارند دسترسی پیدا کنیم.

```
from collections import Counter, defaultdict
by_cls = defaultdict(Counter)
for word_idx, pred in enumerate(word_pred):
    for class_idx, score in enumerate(pred):
        cls = label_encoder.classes_[class_idx]
        by_cls[cls][inverse_vocab[word_idx]] = score
```

## ۱-۲-۱ نمایش نتایج

```
for k in by_cls:
    words = [x[0] for x in by_cls[k].most_common(5)]
    print(k, ': ', ' '.join(words))
```



```

happiness : excited woohoo excellent yay wars
hate : hate hates suck fucking zomberellamcfox
boredom : squeaking ouuut cleanin soooooooo candyland3
enthusiasm : lena_distractia foolproofdiva attending krisswouldhowse tatt
fun : xbox bamboozle sanctuary oldies toodaayy
love : love mothers mommies moms loved
surprise : surprise wow surprised wtf surprisingly
empty : makinitrite conversating less_than_3 shakeyourjunk kimbermuffin
anger : confuzzled fridaaaaaayyyy aaaaaaaaaaaa transtelecom filthy
worry : worried poor throat hurts sick
relief : finally relax mastered relief inspiration
sadness : sad sadly cry cried miss
neutral : www painting souljaboytellem link frenchieb

```

تصویر ۱: نتایج مربوط به احساسات مختلف

## ۱-۲-۲ بحث

وارسی این که مدل ساده چه چیزی یاد گرفته است، قبل از این که به یک مدل پیچیده‌تر برویم می‌تواند یک تمرین خوب باشد. به قدرتی که مدل‌های یادگیری عمیق دارند -شکی نیست- ولی واقعیت این است که توضیح آن که چه اتفاقی در این نوع مدل می‌افتد، مشکل است. ما می‌توانیم یک ایده کلی راجع به این که آن‌ها چگونه کار می‌کند، داشته باشیم ولی در واقع فهم میلیون‌ها وزن در فرآیند آموزش تقریباً غیرممکن است.

دیدیم که نتایج مدل بیزی در راستای آنچه ما انتظار داشتیم، بود. برای مثال، کلمه sad، یک نشانه (یک کلمه با امتیاز بالا)، برای کلاس sadness و کلمه wow یک نشانه برای کلاس surprise بود. همین‌طور مشاهده می‌شود که کلمه mothers یک نشانه قوی برای love است.

علاوه بر موارد فوق، ما یک سری کلمات عجیب برای نشانه پیدا کردیم مثلاً کلمه kimbermuffin، makinitrite یا مثلاً foolproofdiva که به آدمی گفته می‌شود که اشتیاق زیادی دارد. با توجه به هدفمان، می‌توانیم این نمونه‌ها را فیلتر کنیم.

## ۱-۳ استفاده از یک شبکه کانولوشن برای تحلیل احساسات

### مساله

می‌خواهیم که از یک شبکه عمیق برای فهم احساس به کاربرده شده در یک متن استفاده کنیم.

### راه حل

از یک شبکه کانولوشن استفاده کنید.

شبکه‌های کانولوشن معمولاً در تشخیص تصاویر به کار گرفته می‌شوند، ولی در دسته‌بندی متن نیز خوب کار می‌کنند. ایده این است که یک پنجره را روی متن حرکت بدهیم تا یک دنباله طولانی از آیتم‌ها تبدیل به یک دنباله کوتاه‌تر از ویژگی‌ها

تبدیل بشود. آیتم‌ها در این مورد ممکن است کاراکترها باشند. وزن‌های یکسانی برای هرگام استفاده شده است، لذا نیازی نیست که یک چیز را چند بار یاد بگیریم. برای مثال، کلمه cat هر جا در توییتی بیاید معنی گربه می‌دهد.

```
char_input = Input(shape=(max_sequence_len, num_chars), name='input')

conv_1x = Conv1D(128, 6, activation='relu', padding='valid')(char_input)
max_pool_1x = MaxPooling1D(6)(conv_1x)
conv_2x = Conv1D(256, 6, activation='relu', padding='valid')(max_pool_1x)
max_pool_2x = MaxPooling1D(6)(conv_2x)

flatten = Flatten()(max_pool_2x)
dense = Dense(128,
              activation='relu',
              kernel_regularizer=regularizers.l2(0.01))(flatten)
preds = Dense(num_labels, activation='softmax')(dense)

model = Model(char_input, preds)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

return model
```

### ۱-۳-۱ روش One-hot و استفاده از CNN

برای این که مدل اجرا بشود، ما ابتدا باید دیتای خود را برداری کنیم. ما از روش one-hot که در قسمت قبل دیدیم برای کد کردن استفاده می‌کنیم. روش استفاده به این صورت است که هر کلمه یک بردار می‌شود که همه درایه‌های آن غیر از nامی صفر هستند. N کاراکتری است که ما می‌خواهیم آن را کد کنیم.

```
chars = list(sorted(set(chain(*emotion_df['content']))))
char_to_idx = {ch: idx for idx, ch in enumerate(chars)}
max_sequence_len = max(len(x) for x in emotion_df['content'])

char_vectors = []
for txt in emotion_df['content']:
    vec = np.zeros((max_sequence_len, len(char_to_idx)))
    vec[np.arange(len(txt)), [char_to_idx[ch] for ch in txt]] = 1
    char_vectors.append(vec)
char_vectors = np.asarray(char_vectors)
char_vectors = pad_sequences(char_vectors)
labels = label_encoder.transform(emotion_df['sentiment'])
```

داده را به مجموعه آموزشی و تست تقسیم می‌کنیم:

```
def split(lst):
    training_count = int(0.9 * len(char_vectors))
    return lst[:training_count], lst[training_count:]

training_char_vectors, test_char_vectors = split(char_vectors)
training_labels, test_labels = split(labels)
```

حالا ما مدل را آموزش داده و ارزیابی می‌کنیم:

```
char_cnn_model = create_char_cnn_model(len(char_to_idx), char_vectors.shape[1], len(label_encoder.classes_))
char_cnn_model.summary()
```

بعد از ۲۰ گام، دقت در دیتای آموزشی به ۳۹٪ می‌رسد، در حالی که دقت داده تست تنها ۳۱٪ است. تفاوت این دو مقدار دقت با توجه به مفهوم بیش برآزش توضیح داده شده‌است: مدل جنبه‌های اطلاعاتی دادگان را یاد نگرفته و صرفاً قسمتی از داده‌ها را حفظ کرده است. مثل دانش‌آموزی که در یک امتحان ریاضی جواب‌ها را از حفظ نوشته و نمی‌داند که چرا جواب آن می‌شود!

## ۱-۳-۲ بحث

شبکه‌های کانولوشن، معمولاً زمانی که می‌خواهیم چیزهایی را یاد بگیرند، اما دقیقاً نمی‌دانیم چه زمانی رخ می‌دهند، خوب کار می‌کنند. برای تشخیص تصویر، ما از شبکه نمی‌خواهیم که مقادیر ویژگی را برای هر پیکسل تصویر به‌طور جداگانه یاد بگیرد بلکه می‌خواهیم برخی ویژگی‌ها را مستقل از محل وقوع آن‌ها درک کند.

به‌طور مشابه، برای متن، ما از مدل می‌خواهیم یاد بگیرد که اگر کلمه love در هر جای یک توییت آمد، این کلمه می‌تواند یک پرچسب خوب باشد. ما از مدل نمی‌خواهیم که این مسئله را برای هر جای متن به‌طور جدا یاد بگیرد. زمانی که در یک جا در متن، چیزی را یاد بگیرد، می‌تواند آن را نگه دارد. یک شبکه کانولوشن، با استفاده از حرکت دادن یک پنجره روی متن، این کار را انجام می‌دهد. در این جا ما یک پنجره به طول ۶ داریم که یعنی در هر لحظه ۶ کاراکتر را برمی‌داریم؛ برای یک توییت به طول ۱۲۵ کاراکتر ما باید ۱۲۰ گام این پنجره را حرکت بدهیم.

مسئله مهم در این جا این است که همه آن ۱۲۰ نورون، وزن‌های یکسان دارند، پس همه شان یک چیز را یاد گرفته‌اند. پس از لایه کانولوشن ما لایه maxpooling قرار می‌دهیم. این لایه گروهی از ۶ نورون را می‌گیرد و بیشینه را انتخاب می‌کند. این کار، اندازه را باتوجه به فاکتور ۶ کم می‌کند. به‌عبارتی، اندازه را یک ششم می‌کند.

در این مدل ما دو لایه کانولوشن - مکس پولینگ داریم؛ که اندازه را از  $100 * 167$  به  $256 * 3$  تبدیل می‌کند. ما می‌توانیم این‌گونه فکر کنیم که این سطح انتزاع<sup>۱</sup> است. در سطح ورودی، ما فقط ۱۶۷ موقعیت که هر یک از ۱۰۰ کاراکتر مختلف رخ می‌دهد را می‌دانیم. پس از کانولوشن آخر، ما ۳ وکتور از ۲۵۶ از هر توییتی داریم که مشخص می‌کند چه چیزی در شروع، وسط و پایان یک توییت رخ داده است.

## ۱-۴ جمع‌آوری دیتای توییت

مساله

چگونه می‌توانیم یک حجم زیادی از داده‌ها را از توییت برای آموزش مدل به‌طور اتوماتیک جمع‌آوری کنیم؟

راه حل

<sup>۱</sup> . level of abstraction

از API تویتر استفاده کنید.

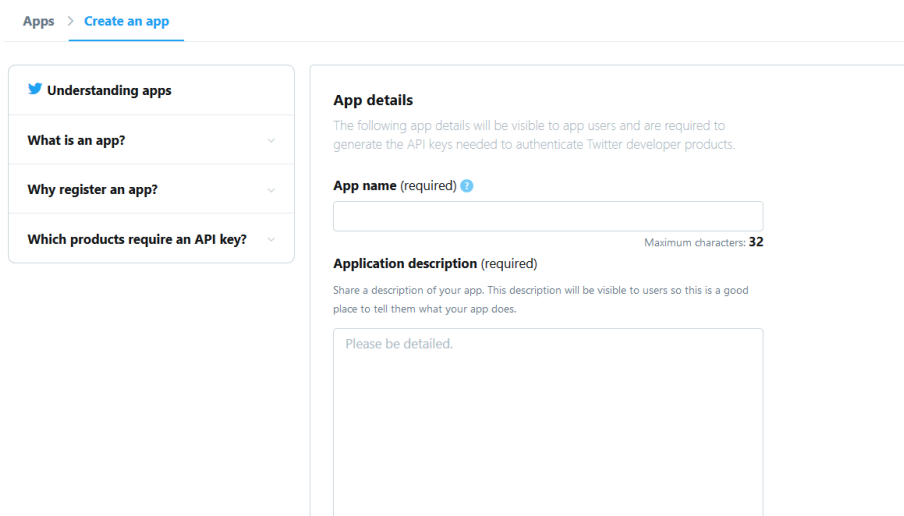
روال کلی به این صورت است:



شکل ۱: روال کلی استخراج داده از تویتر

اولین کاری که باید انجام دهید این است که به وب سایت <https://apps.twitter.com> رفته و برنامه خود را ثبت کنید.

روی Create New App کلیک کرده و فرم مربوطه را پر کنید. می‌توانید Callback URL را خالی بگذارید.



شکل ۲: فرم مربوط به ثبت اپلیکیشن در تویتر

بعد از اتمام فرم، و تایید توسط تویتر از طریق ایمیل، شما دو کلید و دو کد رمز دارید که به شما اجازه می‌دهند به

API دسترسی پیدا کنید. آنها را در متغیرهای مربوطه ذخیره می‌کنیم.

```
CONSUMER_KEY = '<your value>'
CONSUMER_SECRET = '<your value>'
ACCESS_TOKEN = '<your value>'
ACCESS_SECRET = '<your value>'
```

حال می‌توان یک شی برای احراز هویت ساخت.

```
auth=twitter.OAuth(
    consumer_key=CONSUMER_KEY,
    consumer_secret=CONSUMER_SECRET,
```

```
token=ACCESS_TOKEN,
token_secret=ACCESS_SECRET,
)
```

یک API توییتر دو بخش دارد: REST API که آن را قادر می‌سازد که با فراخوانی تعداد متنوعی از توابع در توییت‌ها جستجو کند، وضعیت مربوط به کاربران را بگیرد و حتی در توییتر پست بگذارد. بخش بعدی، بخش دسترسی محدود است که آن Streaming API است. در این دستورالعمل ما از streaming API استفاده می‌کنیم. چنانچه مبلغی را به توییتر بپردازید، شما یک جریان کامل از اطلاعات خواهید داشت که کل توییت‌ها را همزمان که ایجاد می‌شوند، در اختیار شما قرار می‌دهد. اگر این مبلغ را نپردازید، شما یک نمونه از کل توییت‌ها دارید، که همین برای ما کافی است.

```
status_stream = twitter.TwitterStream(auth=auth).statuses
```

شی sream یک تکرارکننده دارد به نام sample که توییت‌ها را برمی‌گرداند.

بیاپید نگاهی به بعضی از این‌ها که از itertools.islice استفاده می‌کنند، بیندازیم:

```
[x['text'] for x in itertools.islice(status_stream.sample(), 0, 5) if x.get('text')]
```

در این جا ما فقط توییت‌هایی را می‌خواهیم که به زبان انگلیسی بوده و حداقل یک ایموجی دارند.

```
def english_has_emoji(tweet):
    if tweet.get('lang') != 'en':
        return False
    return any(ch for ch in tweet.get('text', '') if ch in emoji.UNICODE_EMOJI)
```

ما الان می‌توانیم که ۱۰۰ توییت با حداقل یک ایموجی با کد زیر بگیریم:

```
tweets = list(itertools.islice(filter(english_has_emoji, status_stream.sample()),
, 0, 100))
```

ما حدوداً ۲ تا ۳ توییت در ثانیه می‌گیریم، که بد نیست، ولی به هر حال تا به اندازه ایده آل خود از مجموعه آموزش برسیم، زمان قالب توجهی را صرف می‌کند. ما فقط به توییت‌هایی توجه می‌کنیم که ایموجی‌هایی از یک دسته داشته باشند، و ما فقط آن ایموجی و تکست مربوط به آن را ذخیره می‌کنیم.

```
stripped = []
for tweet in tweets:
    text = tweet['text']
    emojis = {ch for ch in text if ch in emoji.UNICODE_EMOJI}
    if len(emojis) == 1:
        emoji = emojis.pop()
        text = ''.join(ch for ch in text if ch != emoji)
        stripped.append((text, emoji))
len(stripped)
```

## ۱-۴-۱ بحث

تویتر می‌تواند یک منبع بسیار مفید برای داده‌های آموزشی باشد. هر توییت منبع غنی از فراداده است که به صورت مجتمع قرار گرفته‌اند. اطلاعاتی مثل کاربری که این توییت را انجام داده و هشتک‌های به‌کار برده‌شده در متن در آن قرار می‌گیرند. در این فصل ما فقط فراداده‌های مربوط به زبان را استفاده می‌کنیم، ولی همین داده‌ها نیز فضای زیادی برای گشت و گذار دارد.

## ۱-۵ یک پیش‌بینی کننده ایموجی ساده

### مساله

شما چگونه می‌توانید بهترین ایموجی مربوط به یک متن را پیش‌بینی کنید؟

### راه حل

هدف طبقه بند احساسات را از دستورالعمل ۷،۳ به خاطر آورید.

اگر شما یک مقدار قابل توجه از توییت‌ها را در گام قبل به‌دست آورده باشید، می‌توانید از آن استفاده کنید. اگر نه، شما می‌توانید از data/emojis.txt به عنوان فایل داده نمونه استفاده کنید.

داده‌ها را به صورت Pandas DataFrame می‌خوانیم. ما از هر ایموجی که کمتر از ۱۰۰۰ بار تکرار شده باشد، صرف‌نظر می‌کنیم.

```
tweets = all_tweets.groupby('emoji').filter(lambda c:len(c) > 1000)
tweets['emoji'].value_counts()
```

این دیتاست برای این‌که به‌صورت برداری ذخیره شود، بسیار بزرگ است. لذا ما از یک مولد استفاده می‌کنیم. pandas به راحتی می‌تواند با استفاده از متد sample، برای ما data\_generator بسازد.

```
def data_generator(tweets, batch_size):
    while True:
        if batch_size is None:
            batch = tweets
            batch_size = batch.shape[0]
        else:
            batch = tweets.sample(batch_size)
        X = np.zeros((batch_size, max_sequence_len, len(chars)))
        y = np.zeros((batch_size,))
        for row_idx, (_, row) in enumerate(batch.iterrows()):
            y[row_idx] = emoji_to_idx[row['emoji']]
            for ch_idx, ch in enumerate(row['text']):
                X[row_idx, ch_idx, char_to_idx[ch]] = 1
        yield X, y
```

ما حالا می‌توانیم با استفاده از دستورالعمل ۷،۳ بدون تغییر مدل را آموزش دهیم:

```
train_tweets, test_tweets = train_test_split(tweets, test_size=0.1)
BATCH_SIZE = 512
char_cnn_model.fit_generator(
```

```
data_generator(train_tweets, batch_size=BATCH_SIZE),
epochs=20,
steps_per_epoch=len(train_tweets) / BATCH_SIZE,
verbose=2,
)
```

مدل تا صحت ۴۰٪ آموزش می‌بیند. این دقت خوب به نظر می‌رسد، حتی وقتی این را به حساب آوریم که ایموجی‌هایی که زیاد استفاده می‌شوند، احتمال بیشتری نسبت به ایموجی‌های کم کاربرد دارند. اگر ما این مدل را روی مجموعه تست اجرا کنیم، دقت کمی پایین آمده و به ۳۵٪ می‌رسد.

```
char_cnn_model.evaluate_generator(
data_generator(test_tweets, batch_size=BATCH_SIZE),
steps=len(test_tweets) / BATCH_SIZE )
```

نتیجه نهایی به صورت زیر است:

```
[3.1370692166729248, 0.3634086277173913]
```

## ۱-۵-۱ بحث

بدون هیچ تغییری در اصل مدل، ما می‌توانیم بدون استفاده از طبقه بند احساسات، برای هر توییت، ایموجی پیشنهاد بدهیم. این خیلی عجیب نیست؛ به عبارتی ایموجی‌ها برچسب احساساتی هستند که از طریق نویسنده به متن زده شده‌اند. این که کارایی این دو عمل تقریباً مشابه است، در مواردی که داده‌ها نویزی هستند یا برچسب‌ها زیاد می‌شوند، این مورد ممکن است کمتر از حد انتظار شود.

## ۱-۶ Dropout و پنجره‌های چندانگانه

### مساله

چگونه می‌توانیم عملکرد شبکه را افزایش دهیم؟

### راه حل

تعداد متغیرهای قابل یادگیری را هنگام تعریف دراپ اوت افزایش دهید؛ این تکنیکی است که بیش برآزش را برای شبکه سخت می‌کند.

یکی از راه‌های ساده ای که برای افزایش عملکرد شبکه به کار می‌رود، بزرگ‌تر کردن آن است. این کار با افزایش نورون‌ها در هر لایه و افزایش لایه‌ها انجام می‌شود. یک شبکه با متغیرهای بیشتر، ظرفیت بیشتری برای یادگیری دارد و بهتر می‌تواند تعمیم پیدا کند. البته این برای ما تبعاتی دارد، که یکی از آنها مسئله بیش برآزش است.

در این مرحله می‌خواهیم شبکه فعلی را توسعه دهیم. در دستورالعمل قبلی ما از گام ۶ برای کانولوشن استفاده می‌کردیم. ۶ تا کاراکتر برای اطلاعات محلی به نظر منطقی می‌رسد، ولی این همچنین تا حدی دلخواه است. چرا ۴ یا ۵ نه؟ درواقع می‌توانیم پارامتر را با سه عدد (۴ و ۵ و ۶) تنظیم کرده و نتایج همه‌شان را به کار گیریم.

```
layers = []
for window in (4, 5, 6):
    conv_1x = Conv1D(128, window, activation='relu', padding='valid')(char_input)
    max_pool_1x = MaxPooling1D(4)(conv_1x)
    conv_2x = Conv1D(256, window, activation='relu', padding='valid')(max_pool_1x)
    max_pool_2x = MaxPooling1D(4)(conv_2x)
    layers.append(max_pool_2x)

merged = Concatenate(axis=1)(layers)
```

در حین آموزش میزان دقت به ۴۷٪ می‌رسد. ولی متأسفانه در مجموعه تست تنها به ۳۷ درصد می‌رسد. ولی به‌هرحال تا حدی بهتر از مرحله قبل است. البته فاصله بیش برآزشی کمی افزایش یافته است.

چند تکنیک برای مبارزه با بیش برآزش وجود دارد که یکی از آن تکنیک‌ها دراپ اوت است. در طول آموزش، دراپ اوت به طور تصادفی می‌آید و با احتمالی وزن‌های متصل به نورون‌ها را حذف می‌کند. که این کار شبکه را مجبور می‌کند تا مدل را با robustness بیشتری یاد بگیرد و تکیه‌اش به برخی نورون‌های خاص کم شود. در حین پیش‌بینی همه نورون‌ها کار می‌کنند که باعث می‌شود نتیجه میانگین بشود و کمتر حالت‌های استثنا رخ دهد. این روند بیش برآزش را کاهش می‌دهد. در کراس، دراپ اوت را مانند هر لایه دیگری اضافه می‌کنیم. مدل‌مان بعد از آن به این‌صورت درخواهد آمد:

```
layers = []
for window in (4, 5, 6):
    conv_1x = Conv1D(128, window, activation='relu', padding='valid')(char_input)
    max_pool_1x = MaxPooling1D(4)(conv_1x)
    dropout_1x = Dropout(drop_out)(max_pool_1x)
    conv_2x = Conv1D(256, window, activation='relu', padding='valid')(dropout_1x)
    max_pool_2x = MaxPooling1D(4)(conv_2x)
    dropout_2x = Dropout(drop_out)(max_pool_2x)
    layers.append(dropout_2x)

merged = Concatenate(axis=1)(layers)

dropout = Dropout(drop_out)(merged)
```

تعیین مقدار دراپ اوت کمی تجربه می‌خواهد. مقدار بالاتر باعث robustness بالاتر شده ولی سرعت آموزش را کند می‌کند. اجرای کد با دراپ اوت ۰.۲ دقت را در داده آموزشی به ۴۳٪ و در داده تست به ۳۹٪ می‌رساند که به نظر می‌رسد هنوز هم می‌توانیم به دقت بالاتر برسیم.



## ۱-۶-۱ بحث

این دستورالعمل به ما این ایده را می‌دهد که اگر بعضی از تکنیک‌ها را استفاده کنیم، دقت ما بیشتر می‌شود. با افزایش تعداد لایه‌ها، امتحان کردن اندازه مختلف پنجره و استفاده از دراپ‌اوت، می‌توانیم شبکه را بهینه کنیم. به این فرایند پیدا کردن بهترین این مقادیر hyperparameter tuning می‌گویند.

چارچوب‌هایی وجود دارد که می‌توانند به طور اتوماتیک این مقادیر را با امتحان کردن تعداد زیادی از ترکیب این مقادیر پیدا کنند. از آنجایی که آنها احتیاج دارند تا مدل را بارها آموزش دهند، شما دو راه دارید یا به اندازه کافی صبور باشید تا این مدل اجرا بشود یا به طور موازی شبکه را اجرا کنید.

## ۱-۷ ساخت یک مدل در سطح کلمه:

### مساله

توییت‌ها از کلمه تشکیل شده‌اند، نه کاراکترهایی که به صورت تصادفی آمده‌اند. چگونه می‌توانیم از این اتفاق به خوبی استفاده کنیم؟

### راه حل

مدلی را آموزش بدهید که به عنوان ورودی دنباله کلمات را به جای دنباله کاراکترها بگیرد.

اولین کاری که باید در این رابطه انجام بدهیم، جداسازی (توکنایزیشن)<sup>۱</sup> توییت‌هاست. ما یک جداساز کلمه که ۵۰۰۰۰ کلمه را نگه می‌دارد، می‌سازیم. سپس آن را به مجموعه آموزشی و تست اعمال می‌کنیم. بعد به آنها صفر اضافه کرده تا طول منحصر به فرد داشته باشند.

```
VOCAB_SIZE = 50000
tokenizer = Tokenizer(num_words=VOCAB_SIZE)
tokenizer.fit_on_texts(tweets['text'])
training_tokens = tokenizer.texts_to_sequences(train_tweets['text'])
test_tokens = tokenizer.texts_to_sequences(test_tweets['text'])
max_num_tokens = max(len(x) for x in chain(training_tokens, test_tokens))
training_tokens = pad_sequences(training_tokens, maxlen=max_num_tokens)
test_tokens = pad_sequences(test_tokens, maxlen=max_num_tokens)
```

ما می‌توانیم با استفاده از embedding‌های از پیش آموزش داده شده، مدل را خیلی سریع آموزش بدهیم. ما وزن‌ها را با استفاده از یک تابع load\_wv2 بارگزاری می‌کنیم. این تابع word2vec embedding را بارگزاری کرده و آن‌ها را با کلمات داخل پیکره مطابقت می‌دهد. این یک ماتریس می‌سازد که برای هر کدام از توکن‌ها یک ردیف دارد که از وزن‌هایی که از مدل word2vec لود شده‌اند، تشکیل شده است.

```
def load_w2v(tokenizer=None):
```

<sup>۱</sup> . Tokenization

```
word2vec_gz = download('https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz')
word2vec_vectors = word2vec_gz.replace('.gz', '')
if not os.path.exists(word2vec_vectors):
    assert os.system('gunzip -d --keep "%s"' % word2vec_gz) == 0
w2v_model=gensim.models.KeyedVectors.load_word2vec_format(word2vec_vectors,
binary=True)
total_count = sum(tokenizer.word_counts.values())
idf_dict = { k: np.log(total_count/v) for (k,v) in
tokenizer.word_counts.items() }
w2v = np.zeros((tokenizer.num_words, w2v_model.syn0.shape[1]))
idf = np.zeros((tokenizer.num_words, 1))
for k, v in tokenizer.word_index.items():
    if v >= tokenizer.num_words:
        continue
    if k in w2v_model:
        w2v[v] = w2v_model[k]
        idf[v] = idf_dict[k]

del w2v_model
return w2v, idf
```

حال یک مدل می‌سازیم که بسیار شبیه به مدل کاراکتری ماست، اکثر تفاوتش در روش پردازش ورودی است. ورودی

دنباله ای از توکن‌ها را می‌گیرد و لایه های embedding به دنبال هر یک از توکن‌ها که در ماتریس ساخته‌ایم، می‌گردد:

```
def create_cnn_model(vocab_size, embedding_size=None, embedding_weights=None,
drop_out=0.2):
    message = Input(shape=(max_num_tokens,), dtype='int32', name='cnn_input'
)
    embedding = Embedding(mask_zero=False, input_dim=vocab_size,
output_dim=embedding_weights.shape[1],
weights=[embedding_weights],
trainable=True,
name='cnn_embedding')(message)
```

این مدل کار می‌کند، ولی به خوبی مدل کاراکتری عمل نمی‌کند. ما می‌توانیم با تغییر مقدار ابرپارامترها کمی دقت را

افزایش بدهیم، ولی فاصله دقت‌شان کمی زیاد است. می‌توان برای رفع این مشکل یک رویه استفاده کرد (تا درست بشود)

در لایه embedding ویژگی trainable را تبدیل به true کنیم. این کار باعث می‌شود که صحت مدل سطح کلمه تا ۳۶٪

افزایش یابد ولی این همچنین به این معناست که ما از embeddingهای اشتباه استفاده کرده ایم. ما در دستورالعمل بعدی

سعی می‌کنیم این را اصلاح کنیم.

## ۱-۷-۱ بحث

یک مدل سطح کلمه یک دید بزرگتری از داده ورودی نسبت به مدل سطح کاراکتر دارد. به خاطر این که مدل سطح

کلمه به خوشه کلمات به جای خوشه کاراکترها نگاه می‌کند. به جای استفاده از کدگذاری one-hot که برای کاراکترها

استفاده کردیم، از word embedding برای شروع سریع‌تر استفاده می‌کنیم. این جا، ما هر کلمه را با یک وکتور که مقدار معنایی را نشان می‌دهد نمایش می‌دهیم، و آن را به عنوان ورودی به مدل می‌دهیم.

مدلی که در این دستورالعمل توضیح داده شد، بهتر از مدل سطح کاراکتر از لحاظ صحت نبود، و خیلی هم از مدل بیزی بهتر کار نمی‌کرد. این مسئله بیان می‌کند که وزن‌های که از word embedding پیش آموزش داده شده، استفاده کردیم، بد بوده و مشکل همین است. این مدل، وقتی که آن را تبدیل به trainable کنیم، بسیار بهتر کار می‌کند. همچنین وقتی ما به لایه embedding اجازه بدهیم خودش تغییر کند، مدل بهتر می‌شود. در دستورالعمل بعدی با جزییات بیشتری توضیح می‌دهیم.

این که وزن‌ها خوب نیستند، خیلی عجیب نیست. مدل word2vec، بر اساس Google news آموزش دیده شده است، که از نظر زبانی، کمی متفاوت از میانگین رسانه‌های اجتماعی است. برای مثال هشتگ‌های معروف، در پیکره خبر گوگل یافت نمی‌شوند، در حالی که برای طبقه‌بندی توییت‌ها بسیار مهم هستند.

## ۸-۱ ساخت embedding دلخواه خودمان

### مساله

چگونه می‌توانیم embedding بسازیم که با پیکره ما همخوانی داشته باشد؟

### راه حل

Word embedding را خودتان آموزش بدهید!

پکیج gensim نه تنها به شما اجازه استفاده از مدل‌های از پیش آموزش داده شده را می‌دهد، بلکه شما می‌توانید embedding‌های جدید را نیز آموزش دهید. تنها چیزی که نیاز دارد انجام دهد، یک generator هست که دنباله‌ای از توکن‌ها را ایجاد می‌کند. از این عمل استفاده می‌شود تا واژه ایجاد شود. پس از آن مدل را بوسیله مراجعه مداوم به generator، آموزش می‌دهد. شی زیر در جریان توییت‌ها قرار گرفته، آن‌ها را مرتب کرده و آن‌ها را جدا می‌کند.

```
class TokensYielder(object):
    def __init__(self, tweet_count, stream):
        self.tweet_count = tweet_count
        self.stream = stream

    def __iter__(self):
        count = self.tweet_count
        for tweet in self.stream:
            if tweet.get('lang') != 'en':
                continue
            text = tweet['text']
            text = html.unescape(text)
            text = RE_WHITESPACE.sub(' ', text)
            text = RE_URL.sub(' ', text)
            text = strip_accents(text)
            text = ''.join(ch for ch in text if ord(ch) < 128)
```

```
if text.startswith('RT '):
    text = text[3:]
text = text.strip()
if text:
    yield text_to_word_sequence(text)
    count -= 1
if count <= 0:
    break
```

ما حالا می‌توانیم مدل را آموزش دهیم. یک راه معقول برای این کار این است که توییت های مثلا یک هفته را جمع کنیم و آن ها را در فایل ذخیره کنیم، سپس یک generator را از آن ها عبور می‌دهد. قبل از این که ما بخواهیم این کار را انجام دهیم، و یک هفته برای توییت هایمان صبر کنیم، می‌توانیم روی آن ۱۰۰۰۰۰ داده فیلتر شده کار کنیم، ببینیم اصلا کار می‌کند یا نه؟

```
tweets = list(TokensYielder(100000,
                             twitter.TwitterStream(auth=auth).status-
                             es.sample()))
```

و بعد از آن مدل را با gensim بسازیم:

```
model = gensim.models.Word2Vec(tweets, min_count=2)
```

نگاه کردن به نزدیک ترین همسایه کلمه love به ما نشان می‌دهد که در واقع ما embedding های مخصوص یک دامنه را داریم. در توییت ۴۵۳ به love مربوط است! از آن جایی که مخفف cool story, bro است.

```
model.wv.most_similar(positive=['love'], topn=5)
```

نتایج بدست آمده به صورت زیر است:

```
[('miss', 0.822679877281189),
 ('hope', 0.8068050146102905),
 ('loved', 0.8038904666900635),
 ('appreciate', 0.8034697771072388),
 ('ramblingsloa', 0.8009338974952698)]
```

## ۱-۸-۱ بحث

استفاده از word embedding موجود یک راه عالی برای شروع است، ولی فقط زمانی مناسب است که پیکره ای که مدل روی آن آموزش داده شده است با پیکره ما همخوانی داشته باشد. در موقعیت دیگر، که ما دیتاست بزرگ و مناسبی داریم به راحتی می‌توانیم embedding خودمان را بسازیم.

همانطور که در دستورالعمل قبلی دیدیم، یک گزینه دیگر برای embedding جدید، استفاده از embedding موجود ولی با تنظیم ویژگی trainable به true است. این کار شبکه را قادر می‌سازد تا وزن هایی که مغایرت دارند را اصلاح کند.

## ۹-۱ استفاده از شبکه عصبی بازگشتی برای طبقه بندی

### مساله

مطمئناً راهی برای استفاده از این واقعیت که یک توییت دنباله ای از کلمات است، وجود دارد. حالا چگونه این کار را

انجام دهیم؟

### راه حل

از یک شبکه بازگشتی سطح کلمه برای طبقه بندی استفاده کنید.

شبکه های کانولوشن برای پیدا کردن الگو های محلی در ورودی خوب است. برای تحلیل احساسات، اغلب خوب کار می کند. عبارت های خاص روی احساسات موجود در جمله مستقیماً از این که کجای جمله بیایند، اثر می گذارند. تسک پیشنهاد ایموجی، یک عنصر زمان در خود دارد، پس، ما از مزایای CNN استفاده نمی کنیم. ایموجی که با یک توییت مجتمع شده اند؛ معمولاً در انتهای توییت و در نتیجه گیری ایموجی می آید. در این موقعیت ها، یک شبکه RNN بهتر کار می کند. ما در فصل ۵ دیدیم که یک RNN چگونه متن می سازد. ما می توانیم از این روش مشابهی برای پیشنهاد ایموجی استفاده کنیم. کاملاً مانند یک CNN در سطح کلمه، ما کلمات را ورودی داده و به معادل های embeddingشان تبدیل می شود. یک لایه LSTM خیلی خوب کار می کند.

```
def create_lstm_model(vocab_size, embedding_size=None, embedding_weights=None):
    message = layers.Input(shape=(None,), dtype='int32', name='title')
    embedding = make_embedding('message_vec', vocab_size, embedding_size, embedding_weights)(message)

    lstm_1 = layers.LSTM(units=128, return_sequences=False)(embedding)
    # lstm_2 = layers.LSTM(units=128, return_sequences=False)(lstm_1)
    category = layers.Dense(units=len(label_encoder.classes_), activation='softmax')(lstm_1)

    model = models.Model(
        inputs=[message],
        outputs=[category],
    )
    model.compile(loss='sparse_categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    return model
```

بعد از ۱۰ گام ما به صحت ۵۰٪ در آموزش و ۴۰٪ در تست می رسیم، که بسیار بهتر از CNN است.

### ۹-۱-۱ بحث

مدل LSTM که ما این جا استفاده کردیم، بسیار بهتر از مدل CNN عمل کرد. ما می توانیم این عملکرد بسیار بهتر را به این واقعیت نسبت دهیم که توییت ها دنباله هستند. مثلاً چیزی که در پایان جمله رخ داده است با چیزی که در ابتدای جمله آمده است، کاملاً متفاوت است.

از آنجایی که CNN سطح کاراکتر، بهتر از CNN سطح کلمه عمل کرد و LSTM سطح کلمه بهتر از CNN سطح کاراکتر عمل کرد، ما احتمالاً تعجب می‌کنیم از این که یک LSTM سطح کاراکتر از این بهتر عمل نکند. دلیل این است که اگر ما یک LSTM را با یک کاراکتر در هر لحظه، ورودی بدهیم، این احتمال زیاد تا آخرش فراموش می‌کند که در اول متن چه رخ داده بود. اگر ما به LSTM در هر لحظه یک کلمه به عنوان ورودی بدهیم، شبکه قادر خواهد بود تا به مسئله فراموشی غلبه کند. توجه کنید که CNN سطح کاراکتر ما در واقع این مسئله یک کاراکتر در هر لحظه نمی‌تواند انجام بدهد. ما مثلاً یک دنباله از ۴ تایی، ۵ تایی و یا ۶ تایی را در هر لحظه استفاده می‌کنیم، و کانولوشن‌های مختلفی به ازای مقادیر مختلف که داشتیم فقط ۳ بردار ویژگی خروجی می‌دهد. ما می‌توانیم این دو مدل را با هم ادغام کرده، این گونه که از یک CNN برای خلاصه کردن توییت‌ها به فرگمنت و سپس این وکتورها را به یک LSTM به عنوان ورودی، برای به دست آوردن خروجی نهایی بدهیم. این مدل نزدیک به مدل LSTM سطح کلمه است. به جای استفاده از یک CNN که برای دسته بندی تکه های متن، ما از یک Word embedding آماده برای مدل سطح متن استفاده کردیم.

## ۱۰-۱ مصور سازی توافقات

### مساله

شما احتمالاً دوست دارید که به صورت تصویری ببینید که چگونه مدل های متفاوت که ساختیم، در عمل با هم تفاوت دارند.

### راه حل

از pandas برای نشان دادن شباهت ها و تفاوت ها استفاده کنید. دقت، به ما این ایده را می‌دهد که چقدر مدل ما خوب کار می‌کند. پیشنهاد ایموجی یک تسک نویزی است، پس این می‌تواند خیلی مفید باشد که یک نگاهی به انبوه مدل های استفاده شده بیندازیم. pandas یک ابزار مناسب برای این کار است. بگذارید با گرفتن دیتای تست برای مدل کاراکتری مان که به صورت یک وکتور به جای یک generator ذخیره شده است، شروع کنیم.

```
test_char_vectors, _ = next(data_generator(test_tweets, None))
```

حالا بگذارید برای ۱۰۰ مورد تست بگیریم.





















```
predictions = {
    label: [emojis[np.argmax(x)] for x in pred]
    for label, pred in (
        ('lstm', lstm_model.predict(test_tokens[:100])),
        ('char_cnn', char_cnn_model.predict(test_char_vectors[:100])),
        ('cnn', cnn_model.predict(test_tokens[:100])),
    )
}
```












































حالا ما می توانیم pandas dataframe را با ۲۵ پیش بینی اول برای هر کدام از مدل ها، ایموجی اصلی و پیش بینی شده را، ساخته و نمایش بدهیم.

```
pd.options.display.max_colwidth = 128
test_df = test_tweets[:100].reset_index()
eval_df = pd.DataFrame({
    'content': test_df['text'],
    'true': test_df['emoji'],
    **predictions
})
eval_df[['content', 'true', 'char_cnn', 'cnn', 'lstm']].head(25)
```






































## ۱-۱۱ نتیجه

با مرور این نتایج، ما متوجه می شویم که اغلب زمانی مدل اشتباه می کند که دو تا برچسب خیلی به هم شبیه اند. بعضی اوقات پیش بینی ها خیلی بیشتر از متنی که در توییت آمده است، حس می دهد، و بعضی اوقات هیچ کدام از مدل ها خوب کار نمی کند.

#	content	true	char_cnn	cnn	lstm
0	@Gurmeetramrahim @RedFMIndia @rjraunac #8DaysToLionHeart Great				
1	@suchsmallgods I can't wait to show him these tweets				
2	@Captain_RedWolf I have like 20 set lol WAYYYYYYY ahead of you				
3	@OtherkinOK were just at @EPfesti- val, what a set! Next stop is @whelanslive on Friday 11th Novem- ber 2016.				
4	@jochendria: KathNiel with GForce Jorge. #PushAwardsKathNiels				

5	Okay good	   
6	“Distraught means to be upset” “So that means confused right?” -@ReevesDakota	   
7	@JennLiri babe wtf call bck I’m tryna listen to this ring tone	   
8	does Jen want to be friends? we can so be friends. love you, girl. #BachelorIn-Paradise	   
9	@amwalker38: Go Follow these hot accounts @the1stMe420 @DanaDeelish @So_deelish @aka_teemoney38 @CamPromoXXX @SexyLThings @l...	   
10	@gspisak: I always made fun of the parents that show up 30+ mins early to pick up their kids today thats me At least I got a...	   
11	@ShawnMendes: Toronto Billboard. So cool! @spotify #ShawnXSpotify go find them in your city	   
12	@kayleeburt77 can I have your number? I seem to have lost mine.	   
13	@KentMurphy: Tim Tebow hits a dinger on his first pitch seen in professional ball	   
14	@HailKingSoup...	   
15		  



@RoxeteraRibbons Same and I have to figure to prove it	
16 @theseoulstory: September come-backs: 2PM, SHINee, INFINITE, BTS, Red Velvet, Gain, Song Jieun, Kanto...	   
17 @VixenMusicLabel - Peace & Love	   
18 @iDrinkGallons sorry	   
19 @StarYouFollow: 19- Frisson	   
20 @RapsDaiIy: Don't sleep on Ugly God	   
21 How tf do all my shifts get picked up so quickly?! Wtf	   
22 @ShadowhuntersTV: #Shadowhunters fans, how many s would YOU give this father-daughter #FlashbackFriday bonding moment between...	   
23 @mbaylisxo: thank god I have a uniform and don't have to worry about what to wear everyday	   
24 Mood swings like...	   

شکل ۳: نمایش خروجی شبکه

## ۱-۱۱-۱ بحث

اگر به داده اصلی توجه کنیم، می‌توانیم بفهمیم که کجا مدل دارد اشتباه می‌کند. در اینجا یک چیز ساده برای افزایش کارایی، این است که تمام emoji هایی که مشابه و دارای معنای یکسان هستند، به صورت ایموجی یکسان به حساب بیایند. مثلاً حالت های مختلف قلب یا خنده

یک جایگزین دیگر برای یادگیری embedding برای emoji ها وجود دارد. این ممکن است که به ما این دید را بدهد که ایموجی هایی که به هم مربوط اند چگونه اند؟ ما بعد از آن یک تابع خطا داریم که این شباهت را به حساب می‌آورد.

## ۱-۱۲ ادغام مدل ها

### مساله

می خواهیم از ادغام کردن مدل ها برای افزایش قدرت مدل و به دست آوردن جواب بهتر استفاده کنیم.

### راه حل

مدل های مختلف را به صورت یک مدل گروهی در بیاوریم.

ایده استفاده از خرد جمعی: استفاده از میانگین تعداد زیادی نظر معمولاً بهتر از یک خاص شده عمل می کند، معمولاً برای مدل های یادگیری ماشین هم استفاده می کنند. ما می توانیم سه مدل را به یک مدل تبدیل کنیم. ورودی را به سه مدل می دهیم سپس برای لایه خروجی، یک لایه average با استفاده از keras قرار می دهیم.

```
def prediction_layer(model):
    layers = [layer for layer in model.layers if layer.name.endswith('_predictions')]
    return layers[0].output

def create_ensemble(*models):
    inputs = [model.input for model in models]
    predictions = [prediction_layer(model) for model in models]
    merged = Average()(predictions)
    model = Model(
        inputs=inputs,
        outputs=[merged],
    )
    model.compile(loss='sparse_categorical_crossentropy', optimizer='rmsprop',
        metrics=['accuracy'])
    return model
```

ما به data generator های متفاوتی برای آموزش این مدل احتیاج داریم. به جای یک ورودی الان سه تا داریم. از آنجایی که آنها اسامی متفاوتی دارند؛ ما می توانیم data generator طوری بسازیم که سه ورودی را پوشش دهد. همچنین باید یک سری کارها در سطح کاراکتر انجام دهیم تا دیتای سطح کلمه را تنظیم کنیم.

```
def combined_data_generator(tweets, tokens, batch_size):
    tweets = tweets.reset_index()
    while True:
        batch_idx = random.sample(range(len(tweets)), batch_size)
        tweet_batch = tweets.iloc[batch_idx]
        token_batch = tokens[batch_idx]
        char_vec = np.zeros((batch_size, max_sequence_len, len(chars)))
        token_vec = np.zeros((batch_size, max_num_tokens))
        y = np.zeros((batch_size,))
        for row_idx, (token_row, (_, tweet_row)) in enumerate(zip(token_batch, tweet_batch.iterrows())):
            y[row_idx] = emoji_to_idx[tweet_row['emoji']]
            for ch_idx, ch in enumerate(tweet_row['text']):
                char_vec[row_idx, ch_idx, char_to_idx[ch]] = 1
            token_vec[row_idx, :] = token_row
        yield {'char_cnn_input': char_vec, 'cnn_input': token_vec, 'lstm_input': token_vec}, y
```

ما می‌توانیم مدل را این گونه آموزش دهیم:

```
BATCH_SIZE = 512
ensemble.fit_generator(
    combined_data_generator(train_tweets, training_tokens, BATCH_SIZE),
    epochs=20,
    steps_per_epoch=len(train_tweets) / BATCH_SIZE,
    verbose=2,
    callbacks=[early]
)
```

## ۱-۱۲-۱ بحث

مدل‌های ادغام شده یا مدل‌های گروهی یک راه عالی برای ادغام روش‌های مختلف برای یک مسئله است. معمولاً تو مسابقات هوش مصنوعی مانند kaggle از این تکنیک استفاده می‌شود. به جای این که مدل‌ها را کاملاً جدا از هم نگهداریم و سپس آخرش نتیجه را به هم بچسبانیم از average layer استفاده می‌کنیم.

## ۲ مراجع

- [1] D. Osinga, *Deep Learning Cookbook: Practical Recipes to Get Started Quickly*, 1 edition. Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly Media, 2018.