عنوان : تمرین دوم یادگیری عمیق

نگارنده : سحر داستانی اوغانی

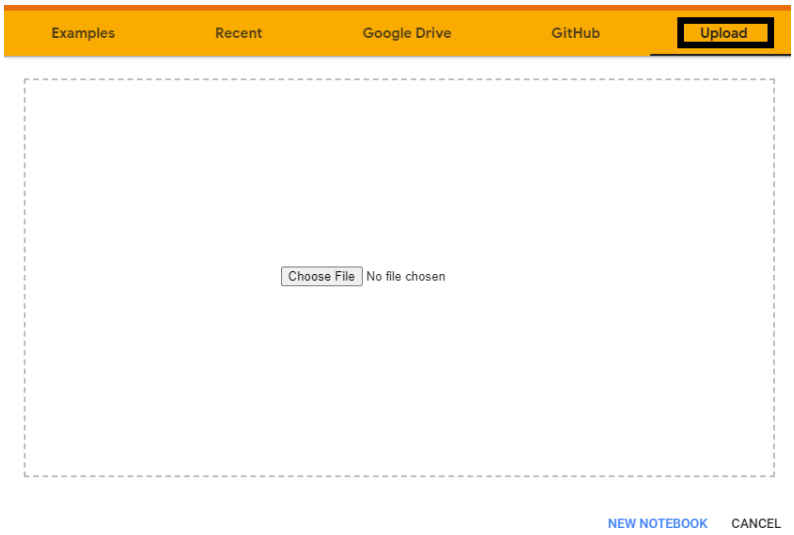
شماره دانشجویی : ۹۹۱۱۲۱۰۸

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

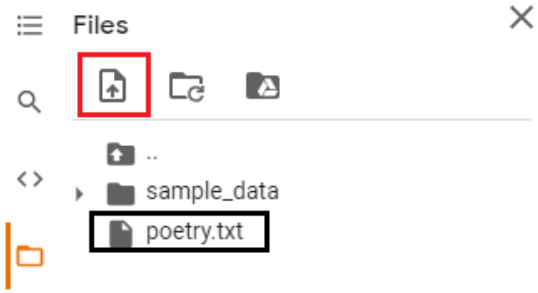
دانشکده ریاضی و علوم کامپیوتر

شیوه‌ی باز کردن کد

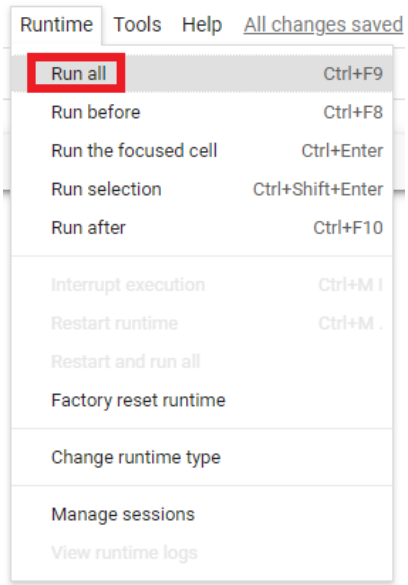
۱. برای این که کد ارسالی را مشاهده کنید به colab رفته و فایل را در قسمت **upload**، **drag** کنید.



۲. سپس دیتاست را در قسمت فایل اضافه کنید.



۳. حال می‌توانید به راحتی کد را ذخیره و از ابتدا اجرا کنید.



بخش اول: مجموع دو عدد باینری

ابتدا به تفصیل مفاهیمی که برای پیاده‌سازی این تمرین نیاز است، می‌پردازیم.

LSB[1]: پایین‌ترین بیت در سری اعداد باینری است. این رقم، معمولا در سمت راست‌ترین یا سمت چپ‌ترین قسمت عدد باینری قرار می‌گیرد. اگر LSB در سمت راست واقع شده باشد، معماری آن سیستم را "little-endian" و اگر در سمت چپ باشد، "big-endian" گویند. برای مثال: راست‌ترین ۱ در ۱۰۱۱۱۰۰۱، LSB این عدد خوانده می‌شود.

LSD[2]: پایین‌ترین رقم در اعداد معمولی را گویند که معمولا در راست‌ترین مکان عدد قرار گرفته است. برای مثال: ۶ در عدد ۲۰۰۶، LSD آن عدد خوانده می‌شود.

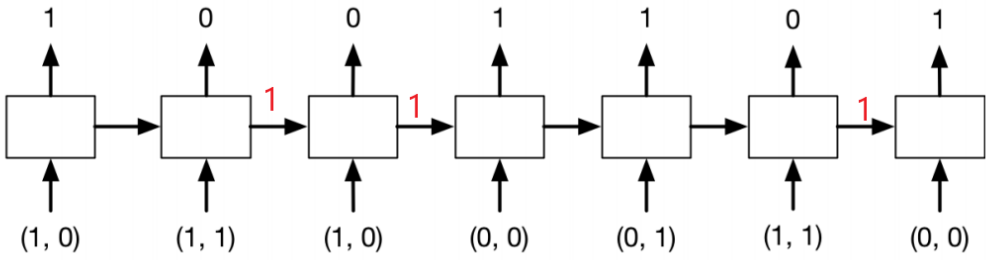
حال که به بررسی دو مفهموم قبل پرداختیم، جمع دو عدد باینری را در سیستم RNN با یکدیگر بررسی می‌کنیم.

فرض کنید می‌خواهیم حاصل جمع دو عدد باینری روبرو را با یکدیگر محاسبه کنیم. $100111 + 110010 = 1011001$

هر رقم این دو عدد، در یک بازه‌ی زمانی[3] وارد سیستم می‌شود. بنابراین لازم است دو ورودی و خروجی را به شکل زیر تعریف کنیم:

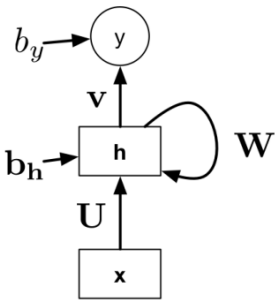
- **Input 1:** 1, 1, 1, 0, 0, 1, 0
- **Input 2:** 0, 1, 0, 0, 1, 1, 0
- **Correct output:** 1, 0, 0, 1, 1, 0, 1

حال که ورودی‌ها و خروجی مشخص شد، کافی است عملکرد سیستم را تعیین کنیم.



ارقام قرمز رنگ، نشان دهنده‌ی ارقام carry هستند، که با ورودی لایه‌ی بعد خود جمع می‌شوند و خروجی را تشکیل می‌دهند.

[1] Least significant bit
[2] Least significant digit
[3] Time step



شبکه‌ی ذکر شده به صورت روبرو خلاصه می‌گردد:

U، V و W، وزن‌های داخلی، x و y به ترتیب ورودی و خروجی شبکه و h، لایه پنهان شبکه می‌باشد. (bها نیز مقادیر بایاس را نمایش می‌دهند.)

پیاده‌سازی

برای پیاده‌سازی این روش می‌توان از دو راهکار بهره گرفت. یا می‌توان از کتابخانه‌های پایتون استفاده کرد یا آن را از scratch پیاده‌سازی کرد. برای بخش اول تنها از سایت‌های مختلف کمک گرفته شده است. ولی بخش پیاده‌سازی از scratch به صورت کامل از سایت peterroelants گرفته شده است. منتها گزارش آن به صورت مو شکافانه توضیح تمامی بخش‌ها را شامل می‌شود.

ساخت با کمک توابع پایتون

ابتدا باید یک دیتاست برای اجرای مدلی که در آینده می‌سازیم، به وجود آوریم. این دیتاست را به داده‌های آموزش و تست تقسیم کرده و عملیات تشکیل ساختار را شروع می‌کنیم.

ساختار مدل‌های RNN را به شرح زیر گزارش می‌کنیم:

- سه ورودی
- یک لایه‌ی پنهان (که دارای سه نورون است)
- یک لایه‌ی خروجی (که دارای دو نورون است)

برای ساخت مدل نیز از Sequential کمک گرفته و ساختار ساخته شده در مرحله‌ی قبل را به لایه‌ی مدل اضافه می‌کنیم. سپس مدل را با استفاده از تابع هزینه‌ی binary_crossentropy کامپایل کرده و بهینه ساز آن را adam قرار می‌دهیم. دلیل این انتخاب، مناسب بودن این تابع هزینه برای ورودی‌های باینری است و چون ورودی شبکه‌های RNN در این تمرین، اعدا باینری می‌باشند، استفاده از این تابع هزینه انتخاب هوشمندانه‌ای می‌باشد.

مدل ساخته شده به شرح زیر می‌باشد:

Creating Model

```
[4]  1 model = tf.keras.Sequential(name='full_adder')
     2 model.add(layers.RNN(FullAdderCell(3), return_sequences=True, input_shape=(None, 2)))
     3
     4 model.summary()
     5
     6 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
     7
     8 model.fit(x_train, y_train, batch_size=32, epochs=5)
     9 scores = model.evaluate(x_test, y_test, verbose=2)
```

```
Model: "full_adder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rnn (RNN)                    (None, None, 1)           20
=================================================================
Total params: 20
Trainable params: 20
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
2813/2813 [==============================] - 9s 3ms/step - loss: 0.6931 - accuracy: 0.5012
Epoch 2/5
2813/2813 [==============================] - 8s 3ms/step - loss: 0.4994 - accuracy: 0.7993
Epoch 3/5
2813/2813 [==============================] - 8s 3ms/step - loss: 0.0921 - accuracy: 1.0000
Epoch 4/5
2813/2813 [==============================] - 7s 3ms/step - loss: 0.0184 - accuracy: 1.0000
Epoch 5/5
2813/2813 [==============================] - 7s 3ms/step - loss: 0.0046 - accuracy: 1.0000
313/313 - 1s - loss: 0.0017 - accuracy: 1.0000
```

برای آزمایش آن از دو عدد به عنوان ورودی استفاده می‌کنیم. دو عدد ۴۸ و ۱۰ را (مانند اعداد استفاده شده در منبع) به شبکه می‌دهیم و شبکه بلافاصله آن‌ها را به اعداد باینری برعکس تبدیل می‌کند. سپس مدل ساخته شده را روی آن‌ها پیاده کرده و خروجی ۵۸ را به صورت باینری و سپس به صورت عددی گزارش می‌کند.

```
a: 48, b: 10
binary representations -> a: [0. 0. 0. 0. 1. 1. 0. 0.], b: [0. 1. 0. 1. 0. 0. 0. 0.]
a_b: [[[0. 0.]
  [0. 1.]
  [0. 0.]
  [0. 1.]
  [1. 0.]
  [1. 0.]
  [0. 0.]
  [0. 0.]]]
```

```
predictions: [0.00121176 0.9986558  0.00121257 0.9986558  0.9986553  0.9986553
 0.00121257 0.00121173]
binary representations -> summed: [0 1 0 1 1 1 0 0]
summed: 58
```

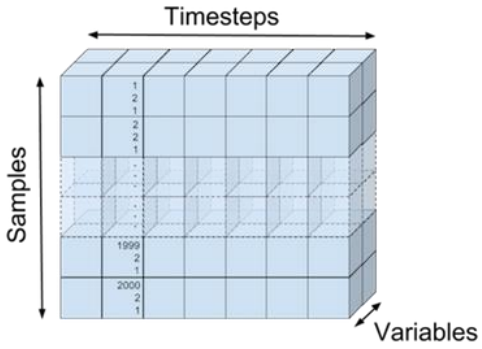
این خروجی نمایانگر درستی عملکرد مدل ساخته شده می‌باشد.

ساخت از scracth

ساخت مجموعه دادگان

در این پیاده‌سازی از یک دیتاست با ۲۰۰۰ نمونه استفاده می‌کنیم که با استفاده از متد creat_dataset ساخته شده است. هر نمونه در این مجموعه دادگان، شامل دو عدد با ۶ رقم به عنوان ورودی است. مجموع دو عدد با ۶ رقم می‌تواند یک عدد ۷ رقمی تولید کند، بنابراین بهتر است یک صفر به ابتدای رقم اضافه کنیم تا یک خروجی target با ۷ رقم تولید کند. توضیحات قبل، در مجموع معادله‌ی زیر را تشکیل می‌دهد $x_{i1} + x_{i2} = t_i$. این اعداد با ارقام باینری نمایش داده شده‌اند و MSB[4] آن‌ها در سمت راست عدد واقع شده است. بنابراین مدل RNN، می‌تواند عمل جمع را از چپ به راست انجام دهد.

ورودی‌ها و بردار هدف[5]، در یک تنسور مرتبه سه ذخیره شدند. این بدان معنی است که هر مرتبه از تنسور وظیفه‌ی خاصی را بر عهده دارند. مرتبه‌ی اول تنسور، شامل تمامی نمونه‌ها (۲۰۰۰ نمونه) است. مرتبه‌ی دوم آن، به تعداد مراحل زمانی، خانه دارد. (7 time step) و مرتبه‌ی سوم آن تعداد متغیرها را نمایش می‌دهد.(دو ورودی)



[4] Most sugnificant bit
[5] Target

حال که دیتاست را تشکیل دادیم، می‌توانیم فرمت آن را مشاهده کنیم:

```
# Create training samples
X_train, T_train = create_dataset(nb_train, sequence_len)

print(f'X_train tensor shape: {X_train.shape}')
print(f'T_train tensor shape: {T_train.shape}')
```

```
X_train tensor shape: (2000, 7, 2)
T_train tensor shape: (2000, 7, 1)
```

همانطور که مشخص است، دادگان در دیتاست در سه بعد واقع شدند که دو ورودی و یک خروجی به ازای هر جمع داریم.

برای این‌که دادگان در دیتاست را به صورت visualize مشاهده کنیم، نمونه‌ای از آن را به شکل زیر چاپ می‌کنیم. برای این کار از تابع printSample استفاده می‌کنیم تا هر رقم را در کنار ارقام دیگر آن عدد باینری بگذارد و به فرم جمع دو عدد زیر هم نمایش دهد.

```
printSample(X_train[0,:,0], X_train[0,:,1], T_train[0,:,:])
```

```
x1:   1010010   37
x2: + 1101010   43
      -------   --
t:  = 0000101   80
```

پردازش ورودی و خروجی تنسور

شبکه‌های عصبی معمولا بردارهای ورودی را به وسیله‌ی ضرب ماتریسی و جمع برداری با استفاده از یک تابع انتقال غیر خطی تبدیل می‌کنند. همانطور که گفتیم، ورودی مسئله یک بردار دو بعدی است، این بردار به وسیله‌ی ماتریس وزن‌ها با اندازه‌ی 3×2 و یک بردار بایاس با اندازه‌ی ۳، انتقال می‌یابد. همچنین، بردارهای سه بعدی تنسور به بردار یک بعدی خروجی با استفاده از ماتریس وزن‌ها با اندازه‌ی 1×3 و بردار بایاس با اندازه‌ی یک تبدیل می‌شود تا خروجی را تشکیل دهند.

از آنجایی که می‌خواهیم ورودی شبکه به صورت بازه‌های زمانی وارد شود، می‌توانیم از تابع tensordot از کتابخانه‌ی numpy استفاده کنیم. این تابع، عمل ضرب داخلی را فراهم می‌کند. کلاسی با نام TensorLinear برای انتقال ورودی X به حالت S و انتقال حالت S به خروجی Y، تعریف می‌کنیم. پس از آن، کلاسی با نام LogisticClassifier تعریف می‌کنیم. این کلاس مشخص می‌کند که احتمال خروجی در گام زمانی K برابر با یک است. این کار را به وسیله‌ی طبقه بندی Logistic انجام می‌دهد.

Unfolding the recurrent states

حالت‌های بازگشت کننده نیاز دارند که در طول زمان باز شوند. این باز شدن در طول زمان، با استفاده از BPTT صورت می‌گیرد. همین امر در کلاس RecurrentStateUnfold پیاده‌سازی شده است. قسمت forward این کلاس، به صورت تکراری حالت‌ها را در طول زمان به روز رسانی می‌کند و نتایج را به صورت تنسور باز می‌گرداند. متد backward، گرادیان را برای خروجی هر حالت پیاده می‌کند. منتها توجه شود که در هر زمان k، گرادیانی که از خروجی Y حاصل می‌شود، نیاز دارد به گرادیانی که از حالت قبلی در زمان k+1 حاصل می‌شود نیز اضافه شود.

تابع فعالیت را tanh انتخاب کردیم، زیرا ماکسیمم گرادیان این تابع از ماکسیمم گرادیان تابع logistic بزرگتر است.

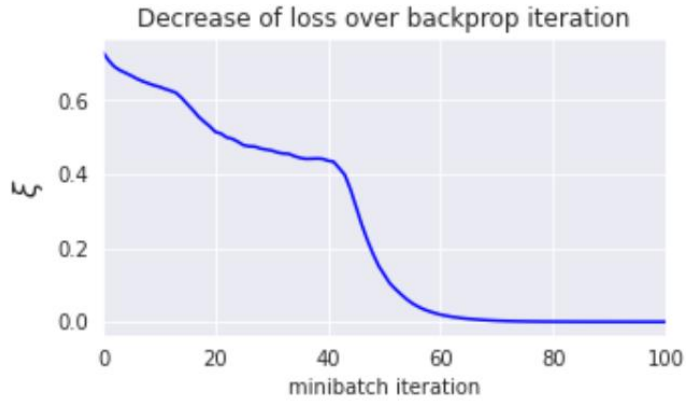
طراحی شبکه

شبکه‌ای که برای محاسبه‌ی مجموع دو عدد باینری طراحی شده است، در کلاس RnnBinaryAdder معرفی شده است. متد forward و backward به ترتیب عملیات مربوط به propagate کردن لایه‌ها را به سمت جلو و عقب برعهده دارند.

عملیاتی تحت عنوان Gradient Checking برای اطمینان حاصل کردن از اینکه هیچ خطایی در مسیر محاسبه‌ی گرادیان وجود ندارد انجام می‌دهیم. پیغام خروجی نشان می‌دهد که خطایی وجود ندارد.

```
No gradient errors found
```

از شبکه انتظار داریم که یاد بگیرد چگونه به نحو احسن عمل جمع باینری را رو روی مثال‌های آموزشی انجام دهد. این موضوع مشخص می‌کند که آموزش شبکه باید دارای هزینه‌ی صفر باشد و به این سمت همگرا شود. نمودار زیر حاکی از این موضوع است:



خروجی

برای آزمایش عملکرد شبکه‌ی طراحی شده، کافی است از داده‌های تست استفاده کنیم. در این صورت مشاهده می‌کنیم که برای هر دو عدد باینری یک خروجی target و یک خروجی y با عنوان خروجی شبکه داریم. به دلیل مشابه بودن این موارد در پنج مورد زیر، می‌توان گفت که عملکرد شبکه به درستی شکل می‌گیرد.

```
x1:   0100010   34
x2: + 1100100   19
      -------   --
t:  = 1010110   53
y:  = 1010110

x1:   1010100   21
x2: + 1110100   23
      -------   --
t:  = 0011010   44
y:  = 0011010

x1:   1111010   47
x2: + 0000000    0
      -------   --
t:  = 1111010   47
y:  = 1111010

x1:   1000000    1
x2: + 1111110   63
      -------   --
t:  = 0000001   64
y:  = 0000001

x1:   1010100   21
x2: + 1010100   21
      -------   --
t:  = 0101010   42
y:  = 0101010
```

بخش دوم: تولید شعر به سبک سعدی با استفاده از RNN

متد زنجیره‌ی مارکوف:

قبل از شروع توضیح این بخش و شرح عملکردهای موجود لازم است متدی را تحت عنوان Markov chain معرفی کنیم. این متد، جملات را بر پایه‌ی ترکیب مجدد عناصر جملات شناخته شده می‌سازد. لغات را آنالیز کرده و احتمال ظهور دو لغت پی در پی را بدست می‌آورد. بنابراین تولید متن توسط این متد به صورت رندوم و بر پایه‌ی احتمال هر لغت می‌باشد. در ابتدا شعری از دیتاست را توسط این روش تولید کرده و در نهایت با اشعار تولیدی توسط RNN آن را مقایسه می‌کنیم. شعر تولیدی به شرح روبرو می‌باشد:

نهیست این نظر خونم بریخت

باور مکن عقل ببرد نگار اوست
نهیست این نظر خونم بریخت

None
کارم زلف یار منست منست
لاله گلستان نمی‌رود دل توست

ندهد چنین دل دست خلاف جان

مقتول زلف یار پریشان درهمست
وصف نیاید مطبوع درختی

متد LSTM:

پیش‌پردازش داده‌ها

برای عملیات preprocessing، سه مرحله در صورت تمرین تعریف شده است. در ادامه به شرح و چگونگی اجرای هر کدام می‌پردازیم.

۱) حذف کلمات و کاراکترهای ایستدر ابتدا نیاز به لیستی از کلمات ایست[6] داریم. به دلیل عدم تعریف این لیست در پایتون و کتابخانه‌های nltk.corpus، مجبور به تعریف آن‌ها به صورت دستی هستیم. برای اینکار می‌توان به سایت‌هایی نظیر لینک۱ و لینک۲ برای دریافت کلمات ایست فارسی مراجعه کرد. برای انجام این تمرین از کلمات لینک۲ به همراه کاراکترهای تعریفی در لینک۱ استفاده شده است. کتابخانه‌های مورد نیاز برای حذف در لیست زیر خلاصه شده‌اند.

```
1 import nltk
2 from nltk.corpus import stopwords
3 from nltk.tokenize import word_tokenize
4 nltk.download('punkt')
5 nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

[6] Stop words

* راهنمایی تعریف کلمات ایست به صورت دستی در پایتون:

در ابتدا لیست لغات را در کلمه‌ای دلخواه مانند stop_word کپی می‌کنید. برای وضوح در تصویر، تمامی کلمات آورده نشده است و ادامه‌ی آن‌ها با علامت ... نمایش داده شده است.

```
stop_word = '''
و
در
به
از
که
مي
اين
است
...
...
...
را
با
هاي
براي
آن
يك
مان
تان
'''
```

سپس با استفاده از word_tokenize، متن تعریف شده را به صورت لغات مستقل از یک دیگر جدا می‌کنیم.

```
tokenize_stop_word = word_tokenize(stop_word)
tokenize_stop_word
```

```
['و',
 'در',
 'به',
 'از',
 'كه',
 'مي',
 'اين',
 'است',
 'را',
 'با',
 'هاي',
```

حال کافی است لغات را به یکدیگر بچسبانیم.

```
filtered_sentence = (" ").join(tokenize_stop_word)
filtered_sentence
```

```
'و در به از كه مي اين است را با هاي براي آن يك مان تان'
```

اکنون به راحتی می‌توان متن تولید شده را کپی کرد و در متغیری ذخیره کرد.

```
Final_stop_words = 'و در به از كه مي اين است را با هاي براي آن يك مان تان'
```

* برای حذف کلمات ایست (که در تعریف شده در مرحله‌ی قبل) از متن دیتاست،کافی است متن را پیمایش کرده و لغاتی که در stop_words وجود دارد را از آن حذف کرد. نمونه‌ای از حذف را در تصویر زیر مشاهده می‌کنید.

جمله‌ی اصلی " همه وقتی غم آن تا چه کند با غم دوست " است که به "غم کند غم دوست" تغییر یافته است.

Data Preprocessing

Delete StopWords

```
[4]  1 list = []
     2 stop_words = '؟ ؛ ، « » [ ] : / . - , * ( ) # '' ! … ۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ همین پیش منّتی هنگامی مان تان
     3 with open('/content/poetry.txt', 'r') as fh:
     4     for line in fh:
     5         text_tokens = word_tokenize(line)
     6         tokens_without_sw = [word for word in text_tokens if not word in stop_words]
     7         filtered_sentence = (" ").join(tokens_without_sw)
     8         filtered_paragraph = ('\n').join(filtered_sentence)
     9     print(filtered_sentence)
```

```
غم کند غم دوست
```

۲) حذف فاصله‌های اضافی

برای انجام این مرحله، کافی است spaceهای اضافی را از متن آماده شده در مرحله‌ی قبل حذف کنیم.

Delete Extra Space

```
[7]  1 re.sub(' +', ' ', text)
```

```
'منعم درویش\u200cقسمت می\nمرغ هوا نصیب ماهی دریا\nبختبندگی بنده نوازی\nخوب آفرید سیرت زیبا\nاکبر اعظم خدای عالم آدم\nصانع پروردگار حی توانا\nدفتر ایزد دانا\n\n'
جلالتی\nعالم نهان پیدا\nنیاز مشفق\u200cهمگان بی\nنخل تناور کند دانه خرما\nشربت نوش آفرید مگس نحل\nتر چوب خشک چشمه خارا\u200cبرگ\nکند شکر نی\u200cه می
خرم صبا\n\nکمال وهم رسد\nسعدی فهم اوست سخن\nکروبیان عالم\nتوانیم حمد گفتن\nعیبی مقدسی میرا\nبارخدایا مهیمنی مدبر\nحیف نصیب رحمت فردا\nنداند سپاس نعمت\nاعضا
پیمان عهد\nجان بقا\u200cکند بی\nرمقی بیش نماند ضعیف\nبگذری پیک نسیم صبا\nدگر کوی دوست\nقدم خوف روم رجا\nای خلاف\u200cصلح آمده\nرود اندر رضا\u200cمی
دوست\nدوست نباشد حقیقت\nدست دامن نکنیمت رها\n...'
```

۳) تبدیل کلمات به ایندکس

در شبکه‌های عصبی قادر به کار با جملات نیستیم. بنابراین بهتر است آن‌ها را به اعداد تبدیل کنیم و اعداد مختص به هر جمله یا کلمه را به NN بدهیم. در ابتدا تمامی کاراکترهای یونیک را فیلتر کرده و آن‌ها را به صورت list شده در متغیر chars می‌ریزیم. این عمل بدین معنی است که هیچ متغیری نمی‌تواند بیشتر از یکبار در متن ظاهر شود. سپس سعی می‌کنیم به هر یک از کاراکترهای یونیک یک عدد اختصاص دهیم. برای اینکار از دو ساختار استفاده می‌کنیم که با دیکشنری کار می‌کنند. ساختار اول کاراکتر را به عدد و ساختار دوم عدد را به کاراکتر تبدیل می‌کند. در این مرحله می‌توان طول کاراکتر یا اندازه‌ی لغات را در متن محاسبه کرد. نتیجه‌ی این محاسبه نشان می‌دهد که سایز کاراکتر ۴۲ در نظر گرفته شده است.

```
Vocabulary size: 42
```

برای آموزش شبکه‌ی عصبی نیاز به داده‌های آموزشی به صورت x و y داریم. برای تولید این نوع داده‌ها، متن دیتاست را پیمایش می‌کنیم و در این میان، sequenceهایی که تولید می‌کنیم را در x ذخیره می‌کنیم. مقادیر واقعی نیز در y ذخیره می‌گردند. تداد sequenceهای تولید شده در متن به شرح زیر است:

```
Number of sequences: 42597
```

لازم به ذکر است که طول sequence را مانند resource، 100 تعریف می‌کنیم. این بدان معناست که تعداد کاراکترهایی که قبل از پیش‌بینی یک کاراکتر در نظر گرفته می‌شود، ۱۰۰ است. سپس ابعاد متن را به شکلی تغییر می‌دهیم که قابل پردازش شوند. درواقع یک بعد برای جملات، یک بعد برای موقعیت کاراکترها در جملات و یک بعد برای مشخص کردن این موضوع که کدام کاراکتر در این موقعیت قرار دارد تعریف می‌شود.

```
((42597, 100, 1), (42597, 42))
```

ساخت مدل RNN

دو مدل برای RNN قابل طراحی است. یک مدلی که متن را تولید کن. و دو مدلی که یک خط را تولید کند.

در ابتدا مدل اول را مورد بررسی قرار می‌دهیم. به دلیل کار با داده‌های sequential، الزامی است که از شبکه‌های RNN برای ساخت مدل استفاده کنیم. در این بخش از Sequential() برای مدل، از Dropout، Flatten، Dense و LSTM برای لایه‌ها و از RMSprop برای بهینه ساز استفاده شده است. لازم به ذکر است که LSTM، نوعی از شبکه‌های RNN است. مدل ذکر شده به شرح زیر است.

▾ Character-Level LSTM Text Generation

▾ Building RNN

```
[10]  1 model = Sequential()
      2
      3 model.add(LSTM(150, input_shape = (X_new.shape[1], X_new.shape[2]), return_sequences = True))
      4
      5 model.add(Dropout(0.1))
      6
      7 model.add(Flatten())
      8
      9 model.add(Dense(Y_new.shape[1], activation = 'softmax'))
     10
     11 model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')
     12
     13 model.fit(X_new, Y_new, epochs = 1, verbose = 1)
```

```
1332/1332 [==============================] - 190s 141ms/step - loss: 3.0989
<tensorflow.python.keras.callbacks.History at 0x7f0df9da8610>
```

شرح لایه‌ها:

در این شبکه از چهار لایه استفاده شده است. ورودی این شبکه عددی است که از حاصلضرب طول جملات در طول کاراکترها حاصل می‌شود. این ورودی به سرعت وارد لایه‌ی LSTM می‌شود، که دارای ۱۵۰ نورون است. این لایه حافظه‌ی شبکه است که وظیفه‌ی یادآوری کاراکترهای مهم گذشته را بر عهده دارد. برای جلوگیری از overfitting از یک لایه‌ی dropout استفاده می‌شود. این لایه به صورت رندوم، بخشی از نورون‌ها و اتصال‌های آن‌ها را حذف می‌کند. لایه‌ی بعدی وظیفه‌ی Flattenz را برعهده دارد. نقش لایه‌ی پنهان در این شبکه را لایه‌ی Dense بازی می‌کند که وظیفه‌ی اضافه کردن پیچیدگی به شبکه را بر عهدا دارد. تعداد نورون‌های این لایه نیز به اندازه‌ی طول کاراکترها در نظر گرفته شده است.

Compile and fit:

تابع هزینه برای انجام compile، categorical_crossentropy انتخاب شده است. دلیل آن شیوه‌ی نمایش و خروجی این تابع است. این تابع آرایه‌ای one-hot تولید می‌کند که شامل احتمال برای هر category می‌باشد. از طرفی تابع بهینه ساز برای optimizer، adam انتخاب شده است. این optimizer، نرخ یادگیری را برای هر وزن در شبکه با unfold کردن یادگیری، به روز رسانی می‌کند.

مدل طراحی شده را بر روی داده‌های آموزشی فیت می‌کنیم. Batch_size را ۲۵۶ می‌گیریم که نشان‌دهنده‌ی تعداد نمونه‌هایی است که می‌توانیم به یکباره داخل شبکه قرارشان دهیم. همچنین تعداد تکرار را ۴ بار در نظر می‌گیریم.

تولید متن:

حال برای تولید متن کافی است یک نقطه‌ی شروع را به صورت رندوم در نظر بگیریم و با استفاده از متد predict متن را پیش‌بینی کنیم. متن پیش‌بینی شده به شرح زیر گزارش شده است.

ای
میان جمع دلم جای
شاهد میان شمع بمیر
چراغ نباشد منور
ابنای روزگار صحرا روند باغ

حال نوبت به بررسی مدل دوم می‌رسد. این مدل برای تولید خط بر مبنای لغاتی است که به آن به عنوان ورودی می‌دهیم. شکل ظاهری مدل به شرح زیر است:

```
# Input layer takes sequence of words as input
input_len = max_seq_len - 1
model = Sequential()
model.add(Embedding(total_words, 10, input_length = input_len))
model.add(LSTM(150))
model.add(Dropout(0.1))
model.add(Dense(total_words, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')

# Use 10 epoch for efficacy
model.fit(predictors, label, epochs = 10, verbose = 1)
```

```
Epoch 1/10
188/188 [==============================] - 6s 22ms/step - loss: 7.9683
Epoch 2/10
188/188 [==============================] - 4s 23ms/step - loss: 7.5007
Epoch 3/10
188/188 [==============================] - 4s 22ms/step - loss: 7.3862
Epoch 4/10
188/188 [==============================] - 4s 22ms/step - loss: 7.2894
Epoch 5/10
188/188 [==============================] - 4s 22ms/step - loss: 7.2471
Epoch 6/10
188/188 [==============================] - 4s 22ms/step - loss: 7.1667
Epoch 7/10
188/188 [==============================] - 4s 23ms/step - loss: 7.0822
Epoch 8/10
188/188 [==============================] - 4s 22ms/step - loss: 6.9840
Epoch 9/10
188/188 [==============================] - 4s 22ms/step - loss: 6.8362
Epoch 10/10
188/188 [==============================] - 4s 22ms/step - loss: 6.6876
<tensorflow.python.keras.callbacks.History at 0x7f1406da7c10>
```

دقت این مدل بعد از ۱۰۰ epoch، ٪۶۷ گزارش شده است.

این مدل دارای لایه‌های گزارش شده در مدل قبلی است. منتها تنها تفاوت آن، وجود لایه Embedding در آن است. کراس این لایه را برای استفاده‌ی شبکه عصبی بر روی داده‌های متنی ارائه داده است. ورودی این لایه باید اعداد صحیح encode شده باشند تا هر لغت بتواند با یک عدد یونیک نمایش داده شود. در ادامه نیز تابعی برای تولید خطوط از دیتاست طراحی شده است که با گرفتن ورودی چند کلمه، قادر به تولید اشعاری به سبک سعدی است. نمونه‌ای از اشعار را مشاهده می‌کنید:

Line generating output

```
[15]  1 generate_line("مرغ هوا را", 5, max_seq_len, model)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450:
  warnings.warn('`model.predict_classes()` is deprecated and '
'مرغ هوا را این این لیلی نگاه سبوست'
```

```
generate_line("نخل تناور کند", 5, max_seq_len, model)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450:
  warnings.warn('`model.predict_classes()` is deprecated and '
'نخل تناور کند دل این پنجه نگاه سکاری'
```

```
generate_line("حمد و ثنا", 5, max_seq_len, model)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450:
  warnings.warn('`model.predict_classes()` is deprecated and '
'حمد و ثنا دل دل دوست عمر زهد'
```

```
generate_line("ما نتوانیم حق", 5, max_seq_len, model)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450:
  warnings.warn('`model.predict_classes()` is deprecated and '
'ما نتوانیم حق دل این لیلی خوشترست جام'
```

```
generate_line("سلام من سحر", 5, max_seq_len, model)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450:
  warnings.warn('`model.predict_classes()` is deprecated and '
'سلام من سحر دل دل این ابروی خوشترست'
```

بخش کارهای اضافه

لازم به ذکر است که می‌توان برای تولید متون به سبک سعدی از مدل زیر نیز استفاده کرد. در این صورت چندین متن با توجه به ویژگی تعریف شده تولید می‌شود که می‌توان از میان آن‌ها بهترین را برگزید.

مدل طراحی شده به شرح زیر است:

```
model = Sequential()
model.add(LSTM(128, input_shape=(SEQ_LENGTH, len(characters))))
model.add(Dense(len(characters)))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=0.01))

model.fit(x, y, batch_size=256, epochs=4) #how many examples we put in to the network at once
```

با استفاده از توابع زیر می‌توان متن را تولید کرد.

توابع کمکی:

۱. Sample

این تابع مقدار پیش‌بینی شده توسط مدل را دریافت می‌کند و یک کاراکتر را به وسیله‌ی آن فیکس می‌کند. کد این قسمت از keras.tutorial برداشته شده است. این تابع مقدار دیگری را نیز به نام temperature دریافت می‌کند. این مقدار نشان-دهنده‌ی میزان خطرناک بودن انتخاب فعلی است. اگر temperature بالا انتخاب کنیم یعنی کاراکتری را برای انتخاب کردن برداشتیم که احتمال وقوع آن بسیار کم است و در غیر این صورت، انتخاب ما محتاطانه است. در واقع از میان احتمال‌های خروجی از softmax، یکی را انتخاب می‌کند.

۲. Generate_text

همانطور که از نام این تابع مشخص است، وظیفه‌ی آن تولید شعر است. در ابتدا یک نقطه‌ی شروع به صورت رندوم را انتخاب کرده و عمل پیش‌بینی را از آنجا شروع می‌کنیم. بنابراین اولین n کاراکتر از روی خود متن کپی می‌شود (n طول sequence را نشان می‌دهد). ولی این موضوع مشکلی به وجود نمی‌آورد زیرا در نهایت می‌توان این تکه را از متن تولید شده حذف کرد و متن نهایی را در اختیار داشت. سپس با در اختیار داشتن طول متن تولیدی می‌توانیم متن را با استفاده از یک حلقه‌ی for تولید کنیم. می‌توان این متن را با ۱۰۰ کاراکتر یا بیشتر تولید کرد. جملات در این متن آرایه‌ای هستند که با مقدار یک یا True نمایش داده می‌شوند. این نمایش زمانی اتفاق می‌افتد که کاراکتر مورد نظر در موقعیت ذکر شده اتفاق افتد. پس از آن از متد predict برای پیش‌بینی احتمال وقوع کاراکتر بعدی استفاده می‌کنیم. در نهایت نیز خروجی را از فرمت عددی به فرمت متنی تبدیل می‌کنیم. وقتی این عملیات را انجام دادیم، کاراکتر را به متن تولیدی اضافه کرده و عملیات ذکر شده را دوباره تکرار می‌کنیم تا تمامی کاراکترهای مورد نظر یافت شوند. نکته‌ای که باید مورد توجه قرار گیرد، ورودی مسئله است. در این قسمت برای تولید شعرهایی به سبک سعدی، از لغات رندوم در خود اشعار سعدی استفاده کردیم و با دادن آن‌ها به شبکه اسشعار زیر را تولید کردیم.

خروجی مدل ← خروجی را با طول ۳۰۰ و میزان temperature 0.2 تا 0.8 بدست آوردیم.

```
----------0.2--------
بر آشنایی
ساحت یار مهربان
می‌روان می‌رود می‌کند بروی می‌کند می‌ارد می‌رود بازی بروان کند می‌رود می‌روان می‌رود می‌رود می‌رود می‌رود می‌رود باز بین بازد می‌رود بوست
بروی می‌رود می‌کند می‌کند می‌کند می‌کند می‌رود می‌کند می‌رود برای می‌روا
----------0.4--------
ای ببری
بندگان بنی سعد خوان یغما
این روشم بیشت می‌ای دلست
می‌سوان خوان مروی بوشت بروید نکند خواب نهای بین شوی می‌رور ناید جویم شور شوی عشق خواش بویش نایش خوشت
خویش روی بروان مشت نمی‌پند خوشت
نیست غمان می‌کوین می‌رود برای خوان برود خواش خوشتی بروای مست دوست دیران برویی می‌رود می‌رود خوشت خوابی بروان جوان فروان بون
----------0.5--------
ی
معرکه زره کنی
دیار اقالیم ترک بسیارند
خوام نیست زید بیز دلان می‌اوده ناید می‌کن میک نیم خوشه خاتی خلان می‌کان حستو
باوی صبر بین این بوست
بینم براز بیر دست عدل بویی می‌رود گوی بار نشاری باب نینه می‌کند گویی نشان ساتی حلار می‌رود خوشت خوشت دوست معنه عشق بدید دل برمید
چشم این دید بی
----------0.6--------
میان این فرصت شمار

بازبینم یار خویش
قیاب بوشتی نوشت عابان خواشت بواشای تیدان ملان بیزی بروی برویر خوشتم زوش بوزم خوشت
دلمی می‌رود سعدی عشق خوست می‌کوی عشق خواب
خوید گوست مروش باده‌ای خوشت
صبر برود شیر بین می‌لوو نظر گوی حهای جار نکوان نسیده مشکوی بگایی بدفت می‌روا نیبان گارم دوستی عوشمن بران ندید
دست آمی گوی ملیر معوست
عشم می‌ریز دوست گو
----------0.7--------

خیمه شعاع جبینست
عالم لوح فکر بشویند
عشک نیاه نهاز جام عورانی بوست شوست می‌یاد می‌یل زوهم بسیان چوشمی انتین دوست
خوشت می‌ترم نفاشد عشق بنده عشقی بوید دلود سعزی بایی عشق بروین انوام بروی بیاید گویمان منهاتهار شوی ترد سعدی  جار خوشت شبای دوست
دلر می‌تای می‌رین باند چند این عشقی عاحب بوست جان می‌نوم عشق بصالدت می‌تنی نهرین
عایش دوب زلمی سعد
----------0.8--------
لله فراموش مکن صحبت
قیمت عشق نداند قدم صبریم ایشی عال عدل گوایت
مین بدیی بان حگوی نهاشی حستیم عادق گواه بون شمان قواز ندادت گویی مور ایی نمیند پیماب دوست
چش می‌رفت خلاشت گوی می‌بوی هوشت
دین می‌ایند خوشهی برار
استان شوم خوشتا بروان زاان نشای
خوی برزایی کنی وین سعدی عیاب
زده خورت
بیاوید دویخت نوبی خوشمی کند بور ووش
بروان بی‌ورمی آشازی طیر م
```

نتایج بالا نشان می‌دهد که هرچه مقدار temperature بالا رود، معنای اشعار نیز بهتر می‌شود. منتها در مقدار 0.8، overfit رخ داده و معانی تغییر می‌کند.

منابع و مراجع

بخش اول

Yet Another Recurrent Neural Network (RNN) Tutorial: An Explicit Explanation | by Yao Peng | Medium

Recurrent Neural Network from scratch — Binary Addition Task | by Vincent | Vincent's Blog | Medium

Recurrent neural network - Binary addition.py · GitHub

Learn to Add Numbers with an Encoder-Decoder LSTM Recurrent Neural Network

https://peterroelants.github.io/posts/rnn-implementation-part02/

بخش دوم

Generating Texts with Recurrent Neural Networks in Python - NeuralNine

Yet Another Recurrent Neural Network (RNN) Tutorial: An Explicit Explanation | by Yao Peng | Medium

Poetry Generator (RNN Markov) | Kaggle

Poetry Text Generation (LSTM) | Kaggle

Deep Learning #4: Why You Need to Start Using Embedding Layers | by Rutger Ruizendaal | Towards Data Science

Understanding RMSprop — faster neural network learning | by Vitaly Bushaev | Towards Data Science

Persian (Farsi) Stopwords

persian-stopwords/chars at master · kharazi/persian-stopwords · GitHub

Removing Stop Words from Strings in Python

How to Generate Music using a LSTM Neural Network in Keras | by Sigurður Skúli | Towards Data Science

Writing Persian Poetry with GPT-2.0 | by Afshin Khashei | Medium

Text Generation in Deep Learning with Tensorflow & Keras | by Murat Karakaya | Deep Learning Tutorials with Keras | Medium

Generation of poems with a recurrent neural network | by Denis Krivitski | Medium