



УНИВЕРСИТЕТ
ИСКУССТВЕННОГО
ИНТЕЛЛЕКТА

ОБРАБОТКА ТЕКСТОВ





Обработка текстов с помощью нейросети

Входные данные для нейросети

Передать необработанные текстовые данные на вход нейросети невозможно, так как сеть работает только с числовыми данными. Для того чтобы была возможность работать с текстами, необходимо перевести текст в числа, если быть точнее, в набор чисел. Для представления существует несколько подходов.

Обработка текстов

Токенизация

Токенизация - это разбиение текста на отдельные токены/блоки.

Одним из способов является сопоставление символа и кода из таблицы ASCII. Например,

для словосочетания "Библиотека Keras"

б - 10000110000

и - 10000111000

б - 10000110000

л - 10000111011

и - 10000111000

о - 10000111110

т - 10000111110

е - 10000110101

к - 10000111010

а - 10000110000

' - 001000000

К - 01001011

е - 01100101

г - 01110010

а - 01100001

с - 01110011

Однако такой метод применяется редко, так как основная проблема его использования - это ограничения памяти.

Другой способ - это кодировать по словам, т. е. когда значение сопоставляется не с символом, а со словом.

Рассмотрим на примере:

«Марк Корнуэлл ел хлеб с сыром, когда раздался стук в дверь.

Комната была маленькой и холодной: горстка горевших прутьев в маленьком камине не согревала ее».

Обработка текстов

Например, можно пронумеровать слова:

марк	1
корнуэлл	2
ел	3
хлеб	4
с	5
сыром	6
когда	7
раздался	8
стук	9
в	10
дверь	11
комната	12
была	13
маленькой	14
и	15
холодной	16
горстка	17
горевших	18
прутьев	19
в	10
маленьком	20
камине	21
не	22
согревала	23
ее	24

либо отсортировать их в алфавитном порядке

была	1
в	2
горевших	3
горстка	4
дверь	5
ее	6
ел	7
и	8
камине	9
когда	10
комната	11
корнуэлл	12
маленькой	13
маленьком	14

Обработка текстов

марк	15
не	16
прутьев	17
раздался	18
с	19
согревала	20
стук	21
сыром	22
хлеб	23
холодной	24

Другой способ - это сопоставление слов с частотностью появления слова, то есть чем чем чаще встречается слово, тем меньше у него индекс, например,

и	2
в	3
не	4
я	5
что	6
на	7
с	8
он	9
а	10
как	11
то	12
это	13
но	14
все	15
у	16
по	17
...	...
раздался	1295
...	...
комната	2001
...	...
зияющая	133072

Обработка текстов

тогда наш текст можно представить как

марк	39585
корнуэлл	134234
ел	4028
хлеб	2866
с	7
сыром	19591
когда	40
раздался	1295
стук	2723
в	2
дверь	181
комната	2001
была	95
маленькой	2174
и	1
холодной	2950
горстка	108963
горевших	120123
прутьев	59542
в	2
маленьком	6276
камине	10454
не	3
согревала	125417
ее	59

Если посмотреть на корнуэлл = 134234 и предлог и = 1, видим, что наши данные разбалансированы. Чтобы этого избежать, существуют способы, построенные методе сбалансированных данных.

Bag Of Words (BOW)

BOW заключается в том, что фраза разбивается на отдельные составляющие части фиксированной длины и представляется в виде вектора заданной длины, размер которого задается параметров `max_words` - индекс максимального слова, который будет анализироваться, при этом все слова, у которых индексы больше `max_words`, заменяются на индекс 1. В результате у нас получается вектор длины `max_words` из 0 и 1 (0 - там, где нет слов, и 1 - на месте индекса встречающего слова).

Прежде чем представлять в форме BOW, надо привести весь текст к нормальной форме, а потом уже конвертировать в BOW.

Обработка текстов

Embedding

Еще одним из способов предобработки текста для использования в сети является Embedding. Embedding - это представление текста в n-мерной форме.

В библиотеке Keras для представления в формате Embedding есть уже готовый слой, например,

`model.add (Embedding (1000, 64, input_length=10))`, где

`input_dim = 1000` - размер словаря, т. е. максимальный целочисленный индекс в входном векторе + 1 (`max_words` в BOW),

`output_dim = 64` - размер плотного embedding-a,

`input_length = 10` - длина входных последовательностей.

Инструкцию можно посмотреть: <https://ru-keras.com/embedding-layers/>.

Для примера рассмотрим фрагмент текста, что и для BOW,

«Марк Корнуэлл ел хлеб с сыром, когда раздался стук в дверь. Комната была маленькой и холодной: горстка горевших прутьев в маленьком камине не согревала ее...»

при `Embedding (1000, 64, input_length=8)` при подаче на вход,

марк	корнуэлл	ел	хлеб	с	сыром	когда	раздался
39585	34234	4028	2866	7	19591	40	1295

на выходе получим,

[0.54	[2.78	[2.12	[6.54	[5.54	[8.54	[0.74	[8.53
1.43	1.23	1.24	2.33	6.43	2.43	1.23	2.63
0.12	1.12	-5.3	1.22	2.12	1.12	2.12	3.52
0.34	-3.65	3.4	3.44	3.34	-4.4	1.54	4.44
0.56	2.45	2.12	4.36	-2.56	2.56	2.56	5.36
...
0.67]	-2.65]	4.1]	3.37]	-2.67]	3.67]	-3.63]	5.27]

Вычисления Embedding-ов требуются большие вычислительные мощности, чтобы их не вычислять, существуют предварительно обученные векторные представления.

Обработка текстов

Предварительно обученные векторные представления слов

Для работы с текстом существуют предварительно обученные векторные представления:

- GloVe (Global Vectors)

Стэнфордский университет

<https://nlp.stanford.edu/projects/glove/>

- Word2Vec

Google

<https://code.google.com/archive/p/word2vec/>

- FastText

Facebook

<https://fasttext.cc>

Word2Vec

Принцип работы Word2Vec заключается в нахождении связей между контекстами слов, то есть, получив на вход текст большого объема, определить слова близкие по смыслу.

В результате расстояние между векторами, например, *мужчина* - *женщина* и *король* - *королева* будет одинаковым, при этом слова *мужчина* и *король* будут находиться недалеко в n-мерном векторном пространстве. В случае с обученными представлениями обучение на корпусе данных не происходит, а данные подаются на вход обученного представления и словам из нашего набора данных присваиваются вектора в n-мерном пространстве.

Обработка текстов

Пример. Классификация текстов писателей

Для примера рассмотрим классификацию текстов писателей: О. Генри, Стругацкие, Булгаков, Клиффорд Саймак, Макс Фрай, Рэй Брэдбери.

После загрузки произведений писателей в обучающую и тестовую выборку необходимо преобразовать тексты к виду, который можно использовать в нейронных сетях. Преобразуем тексты методом Bag Of Words. Для этого существует в Keras встроенный класс Tokenizer (<https://ru-keras.com/text-preprocessing/>).

В общем виде:

```
Tokenizer(  
    num_words=None,  
    filters='!>#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',  
    lower=True,  
    split=' ',  
    char_level=False,  
    oov_token=None,  
    document_count=0),
```

где

num_words: максимальное количество слов, для которых нужно провести токенизацию, основываясь на частоте слов в тексте. Будут переведены в токены только самые распространенные слова в количестве *num_words-1*.

filters: строка, содержащая символы, которые будут исключены из текста. По умолчанию используются все символы пунктуации, плюс символы табуляции и разрыва строки, но исключая символ ' (одиночный апостроф).

lower: логическое значение. Следует ли приводить текст к нижнему регистру.

split: строка. Символ, по которому будет происходить разделение слов в тексте.

Обработка текстов

char_level: если указано значение True, тогда каждый символ будет рассматриваться как токен.

oov_token: (Out-Of-Vocabulary - Вне словарного запаса) если передан этот параметр, то он будет добавлен в word_index и использован для замены слов, не входящих в словарь, во время вызовов text_to_sequence.

По умолчанию вся пунктуация удаляется и текст превращается в последовательности слов, разделенные пробелами (слова могут включать символ ' - одиночный апостроф). Затем эти последовательности разбиваются на списки из токенов. Потом они будут индексированы или векторизованы.

0 - это зарезервированный индекс, который не будет присвоен ни одному слову.

Максимальное количество слов, которое будем учитывать, maxWordsCount = 20 000. Создадим экземпляр класса:

```
tokenizer = Tokenizer(  
    num_words=maxWordsCount, filters='!"#$%&()*+,----./...:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufe0f',  
    lower=True,  
    split=' ',  
    oov_token='unknown',  
    char_level=False)
```

Таким образом, из исходного текста будут удалены все символы !"#\$%&()*+,----./...:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufe0f, все слова будут приведены к нижнему регистру, в качестве разделителя слов - пробел.

Для того чтобы собрать словарь частотности в классе Tokenizer, есть метод .fit_on_texts(), который на вход принимает текст, на выходе - словарь: «слово - индекс»

Соберем словарь частотности для текстов наших писателей, для этого подадим тексты из обучающей выборки:

```
tokenizer.fit_on_texts(trainText)
```

Для того чтобы посмотреть, какие индексы у слов, воспользуемся методом .word_index, который объект класса преобразует в словарь, и методом словаря .items(), который возвращает кортеж пары ключ-значение типа dict_items, который не индексируется, для того, чтобы

Обработка текстов

можно было обращаться к элементам по индексу, необходимо преобразовать в список:

```
items = list(tokenizer.word_index.items())
```

посмотрим часто встречающиеся слова:

```
print(items[-10:])
```

выведет:

```
[('поджарьте', 133061), ('заполните', 133062), ('мучающие', 133063),  
(('погремушкой', 133064), ('свистком', 133065), ('потерян', 133066),  
(('расплывающиеся', 133067), ('миллионе', 133068), ('зияющая', 133069),  
(('ничтонавстречу', 133070))]
```

Теперь необходимо преобразовать текст в набор индексов вместо слов, согласно созданному нами частотному словарю, для этого воспользуемся встроенным методом класса `.texts_to_sequences`, который принимает на вход текст, а возвращает список индексов слов (т. к. в выборке у нас тексты хранились в списке для каждого писателя, то получим список списков:

```
trainWordIndexes = tokenizer.texts_to_sequences(trainText)  
testWordIndexes = tokenizer.texts_to_sequences(testText)
```

Чтобы наши тексты (индексы слов) обучающей и тестовой выборки нарезать на куски и сформировать обучающий набор (`xTrain` - двумерный массив, список фиксированного размера), состоящий из индексов слов (`yTrain` - номер класса в форме one-hot-encoding) и проверочной выборки, необходимы вспомогательные функции:

```
# Формирование обучающей выборки по листу индексов слов  
# (разделение на короткие векторы)  
def getSetFromIndexes(wordIndexes, xLen, step): # функция  
    принимает последовательность индексов, размер окна, шаг  
    окна  
    xSample = [] # Объявляем переменную для векторов  
    wordsLen = len(wordIndexes) # Считаем количество слов  
    index = 0 # Задаем начальный индекс
```


Обработка текстов

```
while (index + xLen <= wordsLen):# Идём по всей длине
вектора индексов
    xSample.append(wordIndexes[index:index+xLen]) #
"Откусываем" векторы длины xLen
    index += step # Смещаемся вперёд на step

return xSample

-----

# Формирование обучающей и проверочной выборки
# Из двух листов индексов от двух классов
def createSetsMultiClasses(wordIndexes, xLen, step): #
Функция принимает последовательность индексов, размер
окна, шаг окна

    # Для каждого из 6 классов
    # Создаём обучающую/проверочную выборку из индексов
    nClasses = len(wordIndexes) # Задаем количество классов
выборки
    classesXSamples = []          # Здесь будет список
размером "кол-во классов*кол-во окон в тексте*длину окна
(например, 6 по 1341*1000)"
    for wI in wordIndexes:        # Для каждого текста выборки
из последовательности индексов
        classesXSamples.append(getSetFromIndexes(wI, xLen,
step)) # Добавляем в список очередной текст индексов,
разбитый на "кол-во окон*длину окна"

    # Формируем один общий xSamples
    xSamples = [] # Здесь будет список размером "суммарное
кол-во окон во всех текстах*длину окна (например,
15779*1000)"
    ySamples = [] # Здесь будет список размером "суммарное
кол-во окон во всех текстах*вектор длиной 6"
```


Обработка текстов

```
for t in range(nClasses): # В диапазоне кол-ва
классов(6)
    xT = classesXSamples[t] # Берем очередной текст вида
"кол-во окон в тексте*длину окна"(например, 1341*1000)
    for i in range(len(xT)): # И каждое его окно
        xSamples.append(xT[i]) # Добавляем в общий список
выборки
    ySamples.append(utils.to_categorical(t, nClasses)) #
Добавляем соответствующий вектор класса

xSamples = np.array(xSamples) # Переводим в массив numpy
для подачи в нейронку
ySamples = np.array(ySamples) # Переводим в массив numpy
для подачи в нейронку
return (xSamples, ySamples) #Функция возвращает выборку
и соответствующие векторы классов
```

Функции на вход принимают список символов (wordIndexes), длину списка (xLen, другими словами, количество слов), шаг (step, на сколько индексов (слов) происходит смещение при формировании).

Получим выборку:

```
xTrain, yTrain = createSetsMultiClasses(trainWordIndexes,
xLen, step) #извлекаем обучающую выборку
```

```
xTest, yTest = createSetsMultiClasses(testWordIndexes,
xLen, step) #извлекаем тестовую выборку
```

При xLen = 50 первый набор будет:

```
print(xTrain[0])
```

```
[ 9296   5 3069  951 14918   3   1   1  11   1 5809   3
 4920   1   1  177  749  15  296 1988   5   1   1  750
   7 3282   1   1   3 1509 6209 6653 1020  11  29  73
   1  102  124   1  138 1716 2608   1  676   1 1086 10331
   1  40]
```


Обработка текстов

Чтобы данный набор символов представить в виде Bag Of Words (BOW), можно либо написать свою функцию, либо воспользоваться методом класса `Tokenizer .sequences_to_matrix()`, которая на вход принимает список (list), а на выходе numpy массив. Преобразуем наш набор индексов из numpy в список: `xTrain.tolist()`,

```
xTrain01 = tokenizer.sequences_to_matrix(xTrain.tolist())
xTest01 = tokenizer.sequences_to_matrix(xTest.tolist())
```

Посмотрим размерность и данные, которые получились (т. к. `maxWordsCount = 20000`, выведем первые 100):

```
print(xTrain01.shape)
print(xTrain01[0][0:100])
```

вернет:

```
(17640, 20000)
[0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 0. 0. 1. 1. 0. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0.
 0. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 0.
 1. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1.
 0. 0. 1. 1.]
```

Таким образом, все наши данные представлены в виде 0 и 1, входные в представлении BOW и выходные в представлении `one_hot_encoding`, которые можно использовать в нейросетях.

Создадим простую нейронную сеть, состоящую из полносвязного слоя с 200 нейронами, слоя Dropout с 25% входных блоков для исключения, слоя нормализации и выходного полносвязного слоя с 6 нейронами (количество классов):

```
model01 = Sequential()
model01.add(Dense(200, input_dim=maxWordsCount,
activation="relu"))
model01.add(Dropout(0.25))
model01.add(BatchNormalization())
model01.add(Dense(6, activation='sigmoid'))
model01.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```


Обработка текстов

Обучим нашу сеть и посмотрим результат:

```
history = model01.fit(xTrain01,  
                      yTrain,  
                      epochs=10,  
                      batch_size=128,  
                      validation_data=(xTest01, yTest))  
plt.plot(history.history['accuracy'],  
          label='Доля верных ответов на обучающем наборе')  
plt.plot(history.history['val_accuracy'],  
          label='Доля верных ответов на проверочном  
наборе')  
plt.xlabel('Эпоха обучения')  
plt.ylabel('Доля верных ответов')  
plt.legend()  
plt.show()
```

В результате:

Epoch 10/10

138/138 [=====] - 5s 38ms/step - loss:
5.2432e-04 - accuracy: 1.0000 - val_loss: 0.2186 - val_accuracy: 0.9251



Как видим, точность классификации на проверочной выборке 92%.