Лабораторная работа №7

Рекуррентные нейронные сети для анализа текста

Набор данных для предсказания оценок для отзывов, собранных с сайта *imdb.com*, который состоит из 50,000 отзывов в виде текстовых файлов.

Отзывы разделены на положительные (25,000) и отрицательные (25,000).

Данные предварительно токенизированы по принципу «мешка слов», индексы слов можно взять из словаря (*imdb.vocab*).

Обучающая выборка включает в себя 12,500 положительных и 12,500 отрицательных отзывов, контрольная выборка также содержит 12,500 положительных и 12,500 отрицательных отзывов.

Данные можно скачать ~~на сайте *Kaggle*~~: ~~https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset (https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset)~~ https://ai.stanford.edu/~amaas/data/sentiment/ (https://ai.stanford.edu/~amaas/data/sentiment/)

Задание 1

Загрузите данные. Преобразуйте текстовые файлы во внутренние структуры данных, которые используют индексы вместо слов.

Будем брать первые `MAX_LENGTH` слов, а если в отзыве слов меньше, чем это число, то применять паддинг.

In [1]:

```
from google.colab import drive

drive.mount('/content/drive', force_remount = True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=
email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2f
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%
2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleap
i.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803
-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=ur
n%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%
2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.co
m%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.re
adonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2'

import sys

sys.path.append(BASE_DIR)

import os
```

In [0]:

```
DATA_ARCHIVE_NAME = 'imdb-dataset-of-50k-movie-reviews.zip'

LOCAL_DIR_NAME = 'imdb-sentiments'
```

In [0]:

```
from zipfile import ZipFile

with ZipFile(os.path.join(BASE_DIR, DATA_ARCHIVE_NAME), 'r') as zip_:
    zip_.extractall(LOCAL_DIR_NAME)
```

In [0]:

```
DATA_FILE_PATH = 'imdb-sentiments/IMDB Dataset.csv'
```

In [0]:

```
import pandas as pd

all_df = pd.read_csv(DATA_FILE_PATH)
```

In [0]:

```
df_test = all_df.sample(frac = 0.1)

df_train = all_df.drop(df_test.index)
```

In [8]:

```
df_train.shape, df_test.shape
```

Out[8]:

```
((45000, 2), (5000, 2))
```

In [9]:

```
import nltk

nltk.download('punkt')
```

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

Out[9]:

```
True
```

In [0]:

```
MAX_LENGTH = 40

STRING_DTYPE = '<U12'

PADDING_TOKEN = 'PAD'

LIMIT_OF_TOKENS = 100000
```

In [0]:

```
from nltk import word_tokenize
import numpy as np
import string
import re

def tokenize_string(_string):
    return  [tok_.lower() for tok_ in word_tokenize(_string) if not re.fullmatch('[' + stri

def pad(A, length):
    arr = np.empty(length, dtype = STRING_DTYPE)
    arr.fill(PADDING_TOKEN)
    arr[:len(A)] = A
    return arr

def tokenize_row(_sentence):
    return pad(tokenize_string(_sentence)[:MAX_LENGTH], MAX_LENGTH)

def encode_row(_label):
    return 1 if _label == 'positive' else 0

def encode_and_tokenize(_dataframe):

    tttt = _dataframe.apply(lambda row: tokenize_row(row['review']), axis = 1)
    llll = _dataframe.apply(lambda row: encode_row(row['sentiment']), axis = 1)

    data_dict_ = { 'label': llll, 'tokens': tttt }

    encoded_and_tokenized_ = pd.DataFrame(data_dict_, columns = ['label', 'tokens'])

    return encoded_and_tokenized_
```

In [0]:

```
df_train_tokenized = encode_and_tokenize(df_train)
df_test_tokenized = encode_and_tokenize(df_test)
```

In [0]:

```
from collections import Counter

def get_tokens_list(_dataframe):

    all_tokens_ = []

    for sent_ in _dataframe['tokens'].values:
        all_tokens_.extend(sent_)

    tokens_counter_ = Counter(all_tokens_)

    return [t for t, _ in tokens_counter_.most_common(LIMIT_OF_TOKENS)]
```

In [0]:

```
tokens_list = get_tokens_list(pd.concat([df_train_tokenized, df_test_tokenized]))
```

In [0]:

```
word_to_int_dict = {}

word_to_int_dict.update(
    {t : i for i, t in enumerate(tokens_list)})
```

In [0]:

```
def intize_row(_tokens):
    return np.array([word_to_int_dict[t]
            if t in word_to_int_dict
            else 0
        for t in _tokens])

def encode_and_tokenize(_dataframe):

    iiii = _dataframe.apply(lambda row: intize_row(row['tokens']), axis = 1)

    data_dict_ = { 'label': _dataframe['label'], 'ints': iiii }

    intized_ = pd.DataFrame(data_dict_, columns = ['label', 'ints'])

    return intized_
```

In [0]:

```
df_train_intized = encode_and_tokenize(df_train_tokenized)
df_test_intized = encode_and_tokenize(df_test_tokenized)
```

Задание 2

Реализуйте и обучите двунаправленную рекуррентную сеть (*LSTM* или *GRU*).

Какого качества классификации удалось достичь?

In [18]:

```
! pip install tensorflow-gpu --pre --quiet

! pip show tensorflow-gpu
```

```
     |████████████████████████████████| 516.2MB 30kB/s
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: keras-preprocessing, wrapt, gast, grpcio, astunparse, h5py, termco
lor, tensorflow-estimator, six, google-pasta, absl-py, opt-einsum, wheel, nu
mpy, protobuf, scipy, tensorboard
Required-by:
```

In [0]:

```
import tensorflow as tf
from tensorflow import keras
```

In [0]:

```
# To fix memory leak: https://github.com/tensorflow/tensorflow/issues/33009

tf.compat.v1.disable_eager_execution()
```

Здесь будем использовать такую конфигурацию рекуррентного *LSTM*-слоя, которая позволит использовать очень быструю *cuDNN* имплементацию.

In [21]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, LSTM, Dense

# The requirements to use the cuDNN implementation are:
# 1. `activation` == `tanh`
# 2. `recurrent_activation` == `sigmoid`
# 3. `recurrent_dropout` == 0
# 4. `unroll` is `False`
# 5. `use_bias` is `True`
# 6. `reset_after` is `True`
# 7. Inputs, if use masking, are strictly right-padded.

model = tf.keras.Sequential()

model.add(Bidirectional(LSTM(100, return_sequences = False), merge_mode = 'concat',
          input_shape = (MAX_LENGTH, 1)))
model.add(Dense(1, activation = 'sigmoid'))
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/py
thon/ops/resource_variable_ops.py:1666: calling BaseResourceVariable.__init_
_ (from tensorflow.python.ops.resource_variable_ops) with constraint is depr
ecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

In [22]:

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 200)               81600
_________________________________________________________________
dense (Dense)                (None, 1)                 201
=================================================================
Total params: 81,801
Trainable params: 81,801
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
X_train_intized = np.asarray(list(df_train_intized['ints'].values), dtype = float)[..., np.
X_test_intized = np.asarray(list(df_test_intized['ints'].values), dtype = float)[..., np.ne

y_train_intized = np.asarray(list(df_train_intized['label'].values))
y_test_intized = np.asarray(list(df_test_intized['label'].values))
```

In [24]:

```
model.fit(x = X_train_intized, y = y_train_intized, validation_split = 0.15, epochs = 20)
```

```
Train on 38250 samples, validate on 6750 samples
Epoch 1/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.6936
- accuracy: 0.5191 - val_loss: 0.6901 - val_accuracy: 0.5313
Epoch 2/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.6884
- accuracy: 0.5418 - val_loss: 0.6862 - val_accuracy: 0.5560
Epoch 3/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6853
- accuracy: 0.5501 - val_loss: 0.6826 - val_accuracy: 0.5612
Epoch 4/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.6828
- accuracy: 0.5585 - val_loss: 0.6889 - val_accuracy: 0.5498
Epoch 5/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6808
- accuracy: 0.5618 - val_loss: 0.6818 - val_accuracy: 0.5630
Epoch 6/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6787
- accuracy: 0.5702 - val_loss: 0.6788 - val_accuracy: 0.5695
Epoch 7/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6768
- accuracy: 0.5738 - val_loss: 0.6825 - val_accuracy: 0.5637
Epoch 8/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6742
- accuracy: 0.5739 - val_loss: 0.6775 - val_accuracy: 0.5720
Epoch 9/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6724
- accuracy: 0.5778 - val_loss: 0.6807 - val_accuracy: 0.5615
Epoch 10/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6701
- accuracy: 0.5816 - val_loss: 0.6777 - val_accuracy: 0.5693
Epoch 11/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6676
- accuracy: 0.5858 - val_loss: 0.6787 - val_accuracy: 0.5637
Epoch 12/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6654
- accuracy: 0.5880 - val_loss: 0.6768 - val_accuracy: 0.5680
Epoch 13/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6621
- accuracy: 0.5933 - val_loss: 0.6726 - val_accuracy: 0.5766
Epoch 14/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6598
- accuracy: 0.5970 - val_loss: 0.6755 - val_accuracy: 0.5701
Epoch 15/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6571
- accuracy: 0.5987 - val_loss: 0.6728 - val_accuracy: 0.5692
Epoch 16/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6542
- accuracy: 0.6018 - val_loss: 0.6747 - val_accuracy: 0.5661
Epoch 17/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6516
- accuracy: 0.6045 - val_loss: 0.6726 - val_accuracy: 0.5736
Epoch 18/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6485
- accuracy: 0.6095 - val_loss: 0.6807 - val_accuracy: 0.5759
Epoch 19/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6467
```

```
- accuracy: 0.6125 - val_loss: 0.6771 - val_accuracy: 0.5714
Epoch 20/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6438
- accuracy: 0.6130 - val_loss: 0.6781 - val_accuracy: 0.5658
```

Out[24]:

```
<tensorflow.python.keras.callbacks.History at 0x7f8fdaa8b208>
```

In [25]:

```python
results = model.evaluate(X_test_intized, y_test_intized)

print('Test loss, test accuracy:', results)
```

```
Test loss, test accuracy: [0.6761944528579712, 0.5762]
```

На валидационной выборке удалось достичь точности 57%.

Задание 3

Используйте индексы слов и их различное внутреннее представление (*word2vec*, *glove*). Как влияет данное преобразование на качество классификации?

Используем 300-мерные вектора *FastTest* — лучшую на сегодняшний день имплементацию word2vec: https://fasttext.cc/docs/en/english-vectors.html (https://fasttext.cc/docs/en/english-vectors.html). Файл пришлось доработать — 9-я строка не читалась.

In [0]:

```python
# VECTORS_ARCHIVE_NAME = 'wiki-news-300d-1M-fixed.zip'

# VECTORS_FILE_NAME = 'wiki-news-300d-1M-fixed.vec'

# VECTORS_LOCAL_DIR_NAME = 'vectors'
```

In [0]:

```python
# with ZipFile(os.path.join(BASE_DIR, VECTORS_ARCHIVE_NAME), 'r') as zip_:
#     zip_.extractall(VECTORS_LOCAL_DIR_NAME)
```

Создадим уменьшенный словарь, содержащий только встреченные токены, чтобы уменьшить нагрузку на *Google Drive*:

In [0]:

```python
# def build_vectors_dict(_actual_tokens, _vectors_file_path, _unknown_token = 'unknown'):

#     vec_data_ = pd.read_csv(_vectors_file_path, sep = ' ', header = None, skiprows = [9])

#     actual_vectors_ = [x for x in vec_data_.values if x[0] in _actual_tokens or x[0] == _

#     return actual_vectors_
```

In [0]:

```
# actual_vectors = build_vectors_dict(tokens_list, os.path.join(VECTORS_LOCAL_DIR_NAME, VEC
```

In [0]:

```
# vectors_np = np.array(actual_vectors)

# vectors_dict = dict(zip(vectors_np[:, 0], vectors_np[:, 1:]))

# vectors_dict_file_name = 'word-vec-dict-{}-items'.format(len(vectors_dict))

# vectors_dict_file_path = os.path.join(BASE_DIR, vectors_dict_file_name)

# np.savez_compressed(vectors_dict_file_path, vectors_dict, allow_pickle = True)
```

In [0]:

```
vectors_dict_file_path = './drive/My Drive/Colab Files/mo-2/word-vec-dict-56485-items.npz'
```

In [0]:

```
vectors_dict_data = np.load(vectors_dict_file_path, allow_pickle = True)

vectors_dict = vectors_dict_data['arr_0'][()]
```

In [0]:

```
VECTORS_LENGTH = 300
```

In [0]:

```
def tokens_to_vectors(_word_to_vec_dict, _tokens, _unknown_token):
    return [_word_to_vec_dict[t]
                if t in _word_to_vec_dict
                else _word_to_vec_dict[_unknown_token]
            for t in _tokens]

def row_to_vectors(_tokens):
    return np.array(tokens_to_vectors(vectors_dict, _tokens, 'unknown'))

def vectorize(_dataframe):

    vvvv = _dataframe.apply(lambda row: row_to_vectors(row['tokens']), axis = 1)

    data_dict_ = { 'label': _dataframe['label'], 'vectors': vvvv }

    vectorized_ = pd.DataFrame(data_dict_, columns = ['label', 'vectors'])

    return vectorized_
```

In [0]:

```
df_train_vectorized = vectorize(df_train_tokenized)
df_test_vectorized = vectorize(df_test_tokenized)
```

In [0]:

```
X_train_vectorized = np.asarray(list(df_train_vectorized['vectors'].values), dtype = float)
X_test_vectorized = np.asarray(list(df_test_vectorized['vectors'].values), dtype = float)

y_train_vectorized = np.asarray(list(df_train_vectorized['label'].values))
y_test_vectorized = np.asarray(list(df_test_vectorized['label'].values))
```

In [37]:

```
model_2 = tf.keras.Sequential()

model_2.add(Bidirectional(LSTM(100, return_sequences = False), merge_mode = 'concat',
            input_shape = (MAX_LENGTH, VECTORS_LENGTH)))
model_2.add(Dense(1, activation = 'sigmoid'))
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
```

In [38]:

```
model_2.compile(optimizer = 'adam',
                loss = 'binary_crossentropy',
                metrics = ['accuracy'])

model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
bidirectional_1 (Bidirection (None, 200)               320800    
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 201       
=================================================================
Total params: 321,001
Trainable params: 321,001
Non-trainable params: 0
_________________________________________________________________
```

In [39]:

```
model_2.fit(x = X_train_vectorized, y = y_train_vectorized, validation_split = 0.15, epochs
```

```
Train on 38250 samples, validate on 6750 samples
Epoch 1/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.5456
- accuracy: 0.7143 - val_loss: 0.4991 - val_accuracy: 0.7511
Epoch 2/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.4851
- accuracy: 0.7566 - val_loss: 0.4786 - val_accuracy: 0.7603
Epoch 3/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.4589
- accuracy: 0.7734 - val_loss: 0.4749 - val_accuracy: 0.7640
Epoch 4/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.4354
- accuracy: 0.7882 - val_loss: 0.4676 - val_accuracy: 0.7686
Epoch 5/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.4117
- accuracy: 0.8025 - val_loss: 0.4917 - val_accuracy: 0.7613
Epoch 6/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.3849
- accuracy: 0.8179 - val_loss: 0.4727 - val_accuracy: 0.7667
Epoch 7/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.3511
- accuracy: 0.8372 - val_loss: 0.5012 - val_accuracy: 0.7529
Epoch 8/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.3122
- accuracy: 0.8599 - val_loss: 0.5162 - val_accuracy: 0.7553
Epoch 9/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.2662
- accuracy: 0.8848 - val_loss: 0.5886 - val_accuracy: 0.7609
Epoch 10/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.2165
- accuracy: 0.9092 - val_loss: 0.6468 - val_accuracy: 0.7603
Epoch 11/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.1680
- accuracy: 0.9329 - val_loss: 0.7128 - val_accuracy: 0.7470
Epoch 12/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.1269
- accuracy: 0.9524 - val_loss: 0.8222 - val_accuracy: 0.7526
Epoch 13/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0908
- accuracy: 0.9670 - val_loss: 0.8999 - val_accuracy: 0.7487
Epoch 14/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.0651
- accuracy: 0.9780 - val_loss: 1.0383 - val_accuracy: 0.7427
Epoch 15/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0495
- accuracy: 0.9842 - val_loss: 1.1222 - val_accuracy: 0.7464
Epoch 16/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0379
- accuracy: 0.9884 - val_loss: 1.2738 - val_accuracy: 0.7513
Epoch 17/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0326
- accuracy: 0.9897 - val_loss: 1.3313 - val_accuracy: 0.7434
Epoch 18/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0364
- accuracy: 0.9888 - val_loss: 1.2889 - val_accuracy: 0.7495
Epoch 19/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0237
```

```
- accuracy: 0.9933 - val_loss: 1.4557 - val_accuracy: 0.7465
Epoch 20/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.0216
- accuracy: 0.9940 - val_loss: 1.3947 - val_accuracy: 0.7459
```

Out[39]:

```
<tensorflow.python.keras.callbacks.History at 0x7f8fd4befa20>
```

In [40]:

```
results_2 = model_2.evaluate(X_test_vectorized, y_test_vectorized)

print('Test loss, test accuracy:', results_2)
```

```
Test loss, test accuracy: [1.3811187601089479, 0.7412]
```

Как и ожидалось, использование эмбеддингов показало лучший результат, чем кодирование слов просто целыми числами — 74%.

Задание 4

Поэкспериментируйте со структурой сети (добавьте больше рекуррентных, полносвязных или сверточных слоев). Как это повлияло на качество классификации?

In [41]:

```
model_3 = tf.keras.Sequential()

model_3.add(Bidirectional(LSTM(5, return_sequences = True), merge_mode = 'concat',
            input_shape = (MAX_LENGTH, VECTORS_LENGTH)))
model_3.add(LSTM(1, return_sequences = False))
model_3.add(Dense(10, activation = 'linear'))
model_3.add(Dense(1, activation = 'sigmoid'))
```

```
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
```

In [42]:

```
model_3.compile(optimizer = 'adam',
                loss = 'binary_crossentropy',
                metrics = ['accuracy'])

model_3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
bidirectional_2 (Bidirection (None, 40, 10)            12240     
_________________________________________________________________
lstm_3 (LSTM)                (None, 1)                 48        
_________________________________________________________________
dense_2 (Dense)              (None, 10)                20        
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11        
=================================================================
Total params: 12,319
Trainable params: 12,319
Non-trainable params: 0
_________________________________________________________________
```

In [44]:

```
model_3.fit(x = X_train_vectorized, y = y_train_vectorized, validation_split = 0.15, epochs
```

```
Train on 38250 samples, validate on 6750 samples
Epoch 1/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.6033
- accuracy: 0.6808 - val_loss: 0.5700 - val_accuracy: 0.7108
Epoch 2/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.5302
- accuracy: 0.7388 - val_loss: 0.5133 - val_accuracy: 0.7453
Epoch 3/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4987
- accuracy: 0.7546 - val_loss: 0.4906 - val_accuracy: 0.7566
Epoch 4/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4819
- accuracy: 0.7621 - val_loss: 0.4779 - val_accuracy: 0.7625
Epoch 5/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.4681
- accuracy: 0.7725 - val_loss: 0.4778 - val_accuracy: 0.7579
Epoch 6/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4602
- accuracy: 0.7770 - val_loss: 0.4755 - val_accuracy: 0.7636
Epoch 7/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4515
- accuracy: 0.7817 - val_loss: 0.4838 - val_accuracy: 0.7563
Epoch 8/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4444
- accuracy: 0.7843 - val_loss: 0.4796 - val_accuracy: 0.7613
Epoch 9/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.4363
- accuracy: 0.7929 - val_loss: 0.4616 - val_accuracy: 0.7683
Epoch 10/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4329
- accuracy: 0.7923 - val_loss: 0.4661 - val_accuracy: 0.7726
Epoch 11/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4258
- accuracy: 0.7961 - val_loss: 0.4658 - val_accuracy: 0.7674
Epoch 12/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4192
- accuracy: 0.8000 - val_loss: 0.4680 - val_accuracy: 0.7711
Epoch 13/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.4138
- accuracy: 0.8031 - val_loss: 0.4609 - val_accuracy: 0.7754
Epoch 14/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.4084
- accuracy: 0.8092 - val_loss: 0.4622 - val_accuracy: 0.7763
Epoch 15/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.4028
- accuracy: 0.8098 - val_loss: 0.4717 - val_accuracy: 0.7705
Epoch 16/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.3975
- accuracy: 0.8142 - val_loss: 0.4648 - val_accuracy: 0.7742
Epoch 17/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.3942
- accuracy: 0.8166 - val_loss: 0.4751 - val_accuracy: 0.7673
Epoch 18/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.3886
- accuracy: 0.8202 - val_loss: 0.4730 - val_accuracy: 0.7730
Epoch 19/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.3844
```

```
- accuracy: 0.8226 - val_loss: 0.4740 - val_accuracy: 0.7753
Epoch 20/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.3797
- accuracy: 0.8246 - val_loss: 0.4765 - val_accuracy: 0.7727
```

Out[44]:

```
<tensorflow.python.keras.callbacks.History at 0x7f8ef6e7a6a0>
```

In [45]:

```python
results_3 = model_3.evaluate(X_test_vectorized, y_test_vectorized)

print('Test loss, test accuracy:', results_3)
```

```
Test loss, test accuracy: [0.49010655212402343, 0.766]
```

Добавление ещё одного рекуррентного слоя ненамного улучшило результат — точность 76% на тестовой выборке.

Задание 5

Используйте предобученную рекуррентную нейронную сеть (например, *DeepMoji* или что-то подобное).

Какой максимальный результат удалось получить на контрольной выборке?

На своих моделях удалось достигнуть максимальной точности 76%.