Введение в обработку естественного языка

Урок 7. Сверточные нейронные сети для анализа текста

Практическое задание

Домашнее задание к уроку 7

Берем отзывы за лето (из архива с материалами или предыдущего занятия)

1. Учим conv сеть для классификации
2. Рассмотреть 2-а варианта сеточек
   2.1 Инициализировать tf.keras.layers.Embedding предобученными векторами взять к примеру с https://rusvectores.org/ru/
   2.2 Инициализировать слой tf.keras.layers.Embedding по умолчанию (ну то есть вам ничего не делать с весами)

Сравнить две архитектуры с предобученными весами и когда tf.keras.layers.Embedding обучается сразу со всей сеточкой, что получилось лучше

```
!pip install --upgrade xlrd
# !pip install --upgrade pandas
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: xlrd in /usr/local/lib/python3.7/dist-packages (2.0.1)
```

```
import numpy as np
import pandas as pd
from string import punctuation
print (pd.__version__)

# xls_file = "data/summer.xls"
# df = pd.read_excel(xls_file)
# df.to_csv('data/summer.csv', index=False)
```

```
1.3.5
```

```
# mkdir data
!ls
```

```
 data   sample_data  'отзывы за лето.xls'
```

```
from google.colab import files
upload = files.upload()
```

Выбрать файлы summer.csv

- **summer.csv**(text/csv) - 2433159 bytes, last modified: 16.06.2022 - 100% done

```
Saving summer.csv to summer.csv
```

```
# !mv 'data/отзывы за лето.xls' 'data/summer.xls'
!mv 'summer.csv' 'data/summer.csv'
```

```
# data = pd.read_excel(open('data/summer.xls', 'rb'))
data = pd.read_csv('data/summer.csv')
data.head(3)
```

	Rating	Content	Date
0	5	It just works!	2017-08-14
1	4	В целом удобноное приложение...из минусов хотя...	2017-08-14
2	5	Отлично все	2017-08-14

```
data.shape
```

```
(20659, 3)
```

```
data.drop('Date', axis=1, inplace=True)
```

```
max_words = 200
max_len = 150
num_classes = 1

# Training
epochs = 20
batch_size = 512
print_batch_n = 100
```

```
# !pip install stop-words
# !pip install pymorphy2
```

Предобработка

```
from string import punctuation
from stop_words import get_stop_words
from pymorphy2 import MorphAnalyzer
import re
```

```
sw = set(get_stop_words("ru"))
exclude = set(punctuation)
morpher = MorphAnalyzer()

def preprocess_text(txt):
    txt = str(txt)
    txt = "".join(c for c in txt if c not in exclude)
    txt = txt.lower()
    txt = re.sub("\sне", "не", txt)
    txt = [morpher.parse(word)[0].normal_form for word in txt.split() if word not in sw]
    return " ".join(txt)

# df_train['Content'] = df_train['Content'].apply(preprocess_text)
# df_val['Content'] = df_val['Content'].apply(preprocess_text)
# df_test['Content'] = df_test['Content'].apply(preprocess_text)

data['Content'] = data['Content'].apply(preprocess_text)
```

```
data.head()
```

	Rating	Content
0	5	it just works
1	4	целое удобноной приложениеиз минус хотеть боль...
2	5	отлично
3	5	зависать 1 работа антивирус ранее пользоваться...
4	5	удобно работать быстро

```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(data.Content, data.Rating, test_size=0.33, random_state=42)
```

```
# # Разбиваем на train, test, val
# # https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c
X = data.drop(columns = ['Rating']).copy()
y = data['Rating']

train_size=0.8
X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8)

test_size = 0.5
X_val, X_test, y_val, y_test = train_test_split(X_rem,y_rem, test_size=0.5)

print(X_train.shape), print(y_train.shape)
print(X_val.shape), print(y_val.shape)
print(X_test.shape), print(y_test.shape)
```

```
    (16527, 1)
    (16527,)
```

```
(2066, 1)
(2066,)
(2066, 1)
(2066,)
(None, None)
```

Токенизация

```
train_corpus = " ".join(X_train["Content"])
train_corpus = train_corpus.lower()
```

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download("punkt")

tokens = word_tokenize(train_corpus)
```

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

Отфильтруем данные

и соберём в корпус N наиболее частых токенов

```
tokens_filtered = [word for word in tokens if word.isalnum()]
```

```
from nltk.probability import FreqDist
dist = FreqDist(tokens_filtered)
tokens_filtered_top = [pair[0] for pair in dist.most_common(max_words-1)]
```

```
tokens_filtered_top[:10]
```

```
['приложение',
 'удобно',
 'работать',
 'удобный',
 'отлично',
 'нравиться',
 'отличный',
 'хороший',
 'телефон',
 'супер']
```

```
vocabulary = {v: k for k, v in dict(enumerate(tokens_filtered_top, 1)).items()}
```

```
def text_to_sequence(text, maxlen):
    result = []
    tokens = word_tokenize(text.lower())
    tokens_filtered = [word for word in tokens if word.isalnum()]
    for word in tokens_filtered:
        if word in vocabulary:
            result.append(vocabulary[word])
    padding = [0]*(maxlen-len(result))
    return padding + result[-maxlen:]
```

```
x_train = np.asarray([text_to_sequence(text, max_len) for text in X_train["Content"]], dtype=np.int32)
x_test = np.asarray([text_to_sequence(text, max_len) for text in X_test["Content"]], dtype=np.int32)
x_val = np.asarray([text_to_sequence(text, max_len) for text in X_val["Content"]], dtype=np.int32)
```

```
x_train.shape
```

```
(16527, 150)
```

```
max_len
```

```
150
```

```
x_train[1]
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0, 177,  72,   1,   1,  18,
                61,  18,  65,  81,  81,  15,  19], dtype=int32)
```

Создание модели

```
import numpy as np
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation, Input, Embedding, Conv1D, GlobalMaxPool1D
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import TensorBoard
# from keras.objectives import categorical_crossentropy
from keras.callbacks import EarlyStopping
import tensorflow as tf

from sklearn.feature_extraction.text import TfidfVectorizer
```

```
import pkg_resources
print(f"keras v{pkg_resources.get_distribution('keras').version}")
```

```
    keras v2.8.0
```

```
print(type(y_train))
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
print(type(y_train))
```

```
    <class 'pandas.core.series.Series'>
    <class 'pandas.core.frame.DataFrame'>
```

```
pd.unique(data['Rating'])
```

```
    array([5, 4, 2, 3, 1])
```

```
num_classes = 6
y_train = tf.keras.utils.to_categorical(y_train['Rating'], num_classes)
y_test = tf.keras.utils.to_categorical(y_test['Rating'], num_classes)
```

```
# y_train = pd.DataFrame(y_train)
# y_test = pd.DataFrame(y_test)
# y_train.head()
```

```
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=128, input_length=max_len))
model.add(Conv1D(128, 3))
model.add(Activation("relu"))
model.add(GlobalMaxPool1D())
model.add(Dense(10))
model.add(Activation("relu"))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

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

```
tensorboard=TensorBoard(log_dir='./logs', write_graph=True, write_images=True)
early_stopping=EarlyStopping(monitor='val_loss')
```

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[tensorboard, early_stopping])
```

```
Epoch 1/20
30/30 [==============================] - 19s 593ms/step - loss: 1.2607 - accuracy: 0.7070 - val_loss: 1.0419 - val_accuracy: 0.
Epoch 2/20
30/30 [==============================] - 18s 617ms/step - loss: 0.9188 - accuracy: 0.7089 - val_loss: 0.8757 - val_accuracy: 0.
```

```
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('\n')
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
5/5 [==============================] - 5s 1s/step - loss: 0.8899 - accuracy: 0.6878


Test score: 0.8898655772209167
Test accuracy: 0.6878024935722351
```

Word2Vec

2.1. Модель со слоем tf.keras.layers.Embedding с предобученными векторами

Предобработка

```
df_w2v = data.copy()
```

```
# Сокращаем количество классов до 2
df_w2v = df_w2v[df_w2v['Rating'] != 3]
df_w2v['target'] = (df_w2v['Rating'] > 3)*1
df_w2v = df_w2v.drop(['Rating'], axis=1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
print(df_w2v.iloc[0])
```

```
Content    it just works
target                 1
Name: 0, dtype: object
```

```
df_w2v['target'] = df_w2v['target'].astype(int)
df_w2v['target'].value_counts()
```

```
1    16724
0     3024
Name: target, dtype: int64
```

```
df_train = df_w2v.loc[:4131]
df_val = df_w2v.loc[4132:]
```

```
df_train['Content'] = df_train['Content'].apply(preprocess_text)
df_val['Content'] = df_val['Content'].apply(preprocess_text)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
```

Токенизация

```
train_corpus = " ".join(df_train["Content"])
train_corpus = train_corpus.lower()
```

```
tokens = word_tokenize(train_corpus)
tokens_filtered = [word for word in tokens if word.isalnum()]

max_words = 200
max_len = 40
```

```
dist = FreqDist(tokens_filtered)
tokens_filtered_top = [pair[0] for pair in dist.most_common(max_words-1)]
```

```
vocabulary = {v: k for k, v in dict(enumerate(tokens_filtered_top, 1)).items()}

def text_to_sequence(text, maxlen):
    result = []
    tokens = word_tokenize(text.lower())
    tokens_filtered = [word for word in tokens if word.isalnum()]
    for word in tokens_filtered:
        if word in vocabulary:
            result.append(vocabulary[word])
    padding = [0]*(maxlen-len(result))
    return padding + result[-maxlen:]

x_train = np.asarray([text_to_sequence(text, max_len) for text in df_train["Content"]], dtype=np.int32)
x_val = np.asarray([text_to_sequence(text, max_len) for text in df_val["Content"]], dtype=np.int32)

x_train
```

```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ..., 101, 102,  13],
       [  0,   0,   0, ...,   0,   0,   5],
       ...,
       [  0,   0,   0, ...,   0,   0,   0],
       [  0,   0,   0, ...,   0,   4,   1],
       [  0,   0,   0, ...,   0,   0,   4]], dtype=int32)
```

```
df_train["target"].unique()
```

```
array([1, 0])
```

Создание модели

```
from tensorflow.keras import utils as np_utils
num_classes = 2
y_train_w2v = np_utils.to_categorical(df_train["target"], num_classes)
y_val_w2v = np_utils.to_categorical(df_val["target"], num_classes)
```

```
from gensim.models import Word2Vec
```

```
modelW2V = Word2Vec(sentences=df_train['Content'].apply(str.split), size=40, window=5, min_count=1)
# modelW2V = Word2Vec(sentences=df_train['Content'].apply(str.split), window=5, min_count=1)
```

```
vect_idf = TfidfVectorizer()
vect_idf.fit_transform(df_train['Content'])
tfidf = dict(zip(vect_idf.get_feature_names(), vect_idf.idf_))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated
  warnings.warn(msg, category=FutureWarning)
```

```
def get_vect_mean(txt):
    vector_w2v = np.zeros(40)
    n_w2v = 0
    for wrd in txt.split():
        if wrd in modelW2V:
            vector_w2v += modelW2V[wrd]
            n_w2v += 1
    if n_w2v > 0:
        vector_w2v = vector_w2v / n_w2v
    return vector_w2v
```

```
from tqdm import tqdm_notebook

arr_vect = []
for txt in tqdm_notebook(df_train['Content']):
    arr_vect.append(get_vect_mean(txt))

arr_vect_valid = []
for txt in tqdm_notebook(df_val['Content']):
    arr_vect_valid.append(get_vect_mean(txt))

x_train_w2v = np.asarray(arr_vect)
x_val_w2v = np.asarray(arr_vect_valid)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: TqdmDeprecationWarning: This function will be removed in tqdm==
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  after removing the cwd from sys.path.
100%                                        3950/3950 [00:00<00:00, 13516.58it/s]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated `__contains__` (Method w
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: DeprecationWarning: Call to deprecated `__getitem__` (Method wi

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: TqdmDeprecationWarning: This function will be removed in tqdm==
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`


100%                                        15798/15798 [00:00<00:00, 15715.36it/s]
```

```
from tensorflow.keras.layers import Dense, Embedding, GlobalMaxPooling1D, Conv1D, Activation
```

```
model_w2v = tf.keras.Sequential()
model_w2v.add(Embedding(input_dim=max_words, output_dim=128, input_length=max_len))
model_w2v.add(Conv1D(128, 3))
model_w2v.add(Activation("relu"))
model_w2v.add(GlobalMaxPooling1D())
model_w2v.add(Dense(10))
model_w2v.add(Activation("relu"))
model_w2v.add(Dense(num_classes))
model_w2v.add(Activation('softmax'))

model_w2v.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 40, 128)           25600

 conv1d_3 (Conv1D)           (None, 38, 128)           49280

 activation_7 (Activation)   (None, 38, 128)           0

 global_max_pooling1d_2 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_4 (Dense)             (None, 10)                1290

 activation_8 (Activation)   (None, 10)                0

 dense_5 (Dense)             (None, 2)                 22

 activation_9 (Activation)   (None, 2)                 0

=================================================================
Total params: 76,192
Trainable params: 76,192
Non-trainable params: 0
_________________________________________________________________
```

Подготовка к обучению

```
LEARNING_RATE = 0.0001
```

```
optimizer = tf.keras.optimizers.Adam(lr=LEARNING_RATE)

model_w2v.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['AUC'])

tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir='logs/my_model_plus_w2v',
    write_graph=False, update_freq=100, profile_batch=0)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learn
  super(Adam, self).__init__(name, **kwargs)
```

Обучение модели

```
# Training
NUM_EPOCHS = 8
batch_size = 1024
```

```
%%time

history = model_w2v.fit(
    x_train_w2v, y_train_w2v,
    #batch_size=batch_size,
    epochs=NUM_EPOCHS,
    validation_split=0.1,
    callbacks=[tensorboard_callback])
```

```
Epoch 1/8
112/112 [==============================] - 4s 23ms/step - loss: 0.6909 - auc: 0.7025 - val_loss: 0.6850 - val_auc: 0.8759
Epoch 2/8
112/112 [==============================] - 2s 20ms/step - loss: 0.6809 - auc: 0.8796 - val_loss: 0.6770 - val_auc: 0.8759
Epoch 3/8
112/112 [==============================] - 2s 20ms/step - loss: 0.6728 - auc: 0.8811 - val_loss: 0.6691 - val_auc: 0.8759
Epoch 4/8
112/112 [==============================] - 2s 19ms/step - loss: 0.6649 - auc: 0.8804 - val_loss: 0.6614 - val_auc: 0.8759
Epoch 5/8
112/112 [==============================] - 3s 24ms/step - loss: 0.6571 - auc: 0.8831 - val_loss: 0.6538 - val_auc: 0.8759
Epoch 6/8
112/112 [==============================] - 3s 28ms/step - loss: 0.6496 - auc: 0.8782 - val_loss: 0.6464 - val_auc: 0.8759
Epoch 7/8
112/112 [==============================] - 3s 28ms/step - loss: 0.6422 - auc: 0.8790 - val_loss: 0.6392 - val_auc: 0.8759
Epoch 8/8
112/112 [==============================] - 4s 33ms/step - loss: 0.6350 - auc: 0.8802 - val_loss: 0.6322 - val_auc: 0.8759
CPU times: user 25.8 s, sys: 2.42 s, total: 28.2 s
Wall time: 42 s
```

Оценка качества модели

```
loss, accuracy = model_w2v.evaluate(x_train_w2v, y_train_w2v, batch_size=batch_size, verbose=False)
print("Training Loss:  {:.4f}".format(loss))
print("Training Accuracy:  {:.4f}".format(accuracy))
print('\n')
loss, accuracy = model_w2v.evaluate(x_val_w2v, y_val_w2v, batch_size=batch_size, verbose=False)
print("Testing Loss:  {:.4f}".format(loss))
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Loss:  0.6314
Training Accuracy:  0.8803


Testing Loss:  0.6386
Testing Accuracy:  0.8385
```

2.2 Модель со слоем tf.keras.layers.Embedding по умолчанию

```
exclude = set(punctuation)
sw = set(get_stop_words("ru"))
morpher = MorphAnalyzer()

def preprocess_text(txt):
    txt = str(txt)
    txt = "".join(c for c in txt if c not in exclude)
    txt = txt.lower()

    txt = re.sub("\sне", "не", txt)

    txt = [morpher.parse(word)[0].normal_form for word in txt.split() if word not in sw]
    txt = [word for word in txt if len(word)>1] # условие "более одного слова в тексте"

    return " ".join(txt)

data['text'] = data['Content'].apply(preprocess_text)
```

```
# Сокращаем количество классов до 2-х
data = data[data['Rating'] != 3]
data['target'] = (data['Rating'] > 3)*1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
print(data.iloc[0])
```

```
Rating                 5
Content    it just works
text       it just works
target                 1
Name: 0, dtype: object
```

```
df = data.drop(['Content', 'Rating'], axis=1)
print(df.iloc[0])
```

```
text      it just works
target                1
Name: 0, dtype: object
```

```
df = data.drop(['Content', 'Rating'], axis=1)
print(df.iloc[0])
```

```
text      it just works
target                1
Name: 0, dtype: object
```

```
df_train, df_val = train_test_split(df, test_size=0.2,random_state=13)
```

Токенизация

```
text_corpus_train = df_train['text'].values
text_corpus_val = df_val['text'].values
```

```
tokenizer = Tokenizer(num_words=None,
                     filters='#$%&()*+-<=>@[\\]^_`{|}~\t\n',
                     lower = False, split = ' ')
tokenizer.fit_on_texts(text_corpus_train)

sequences_train = tokenizer.texts_to_sequences(text_corpus_train)
sequences_val = tokenizer.texts_to_sequences(text_corpus_val)

word_count = len(tokenizer.index_word) + 1
training_length = max([len(i.split()) for i in text_corpus_train])

x_train = pad_sequences(sequences_train, maxlen=training_length)
x_val = pad_sequences(sequences_val, maxlen=training_length)
```

```
from tensorflow.keras import utils as np_utils
num_classes = 2
y_train = np_utils.to_categorical(df_train["target"], num_classes)
y_val = np_utils.to_categorical(df_val["target"], num_classes)
```

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

```
model.add(Embedding(input_dim=word_count,
                    output_dim=128,
                    input_length=training_length))
model.add(Conv1D(128, 3))
model.add(Activation("relu"))
model.add(GlobalMaxPooling1D())
model.add(Dense(10))
model.add(Activation("relu"))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_4 (Embedding)     (None, 131, 128)          1371392   
                                                                 
 conv1d_4 (Conv1D)           (None, 129, 128)          49280     
                                                                 
 activation_10 (Activation)  (None, 129, 128)          0         
                                                                 
 global_max_pooling1d_3 (Glo  (None, 128)              0         
 balMaxPooling1D)                                                
                                                                 
 dense_6 (Dense)             (None, 10)                1290      
                                                                 
 activation_11 (Activation)  (None, 10)                0         
                                                                 
 dense_7 (Dense)             (None, 2)                 22        
                                                                 
 activation_12 (Activation)  (None, 2)                 0         
                                                                 
=================================================================
Total params: 1,421,984
Trainable params: 1,421,984
Non-trainable params: 0
_________________________________________________________________
```

```
LEARNING_RATE = 0.0001
optimizer = tf.keras.optimizers.Adam(lr=LEARNING_RATE)

model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir='logs/my_model',
    write_graph=False, update_freq=100, profile_batch=0)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learn
  super(Adam, self).__init__(name, **kwargs)
```

```
batch_size=1024
NUM_EPOCHS=8
```

```
%%time

history = model.fit(
    x_train, y_train,
    epochs=NUM_EPOCHS,
    validation_data=(x_val,y_val),
    callbacks=[tensorboard_callback])
```

```
Epoch 1/8
494/494 [==============================] - 47s 92ms/step - loss: 0.3614 - accuracy: 0.8504 - val_loss: 0.2647 - val_accuracy: 0
Epoch 2/8
494/494 [==============================] - 35s 71ms/step - loss: 0.2051 - accuracy: 0.9140 - val_loss: 0.1976 - val_accuracy: 0
Epoch 3/8
494/494 [==============================] - 38s 77ms/step - loss: 0.1556 - accuracy: 0.9378 - val_loss: 0.1813 - val_accuracy: 0
Epoch 4/8
494/494 [==============================] - 35s 70ms/step - loss: 0.1275 - accuracy: 0.9518 - val_loss: 0.1774 - val_accuracy: 0
Epoch 5/8
494/494 [==============================] - 35s 70ms/step - loss: 0.1046 - accuracy: 0.9637 - val_loss: 0.1785 - val_accuracy: 0
Epoch 6/8
494/494 [==============================] - 33s 68ms/step - loss: 0.0865 - accuracy: 0.9701 - val_loss: 0.1905 - val_accuracy: 0
Epoch 7/8
494/494 [==============================] - 34s 68ms/step - loss: 0.0706 - accuracy: 0.9771 - val_loss: 0.1950 - val_accuracy: 0
Epoch 8/8
```

```
494/494 [==============================] - 36s 72ms/step - loss: 0.0577 - accuracy: 0.9825 - val_loss: 0.2046 - val_accuracy: 0
CPU times: user 7min 3s, sys: 24.9 s, total: 7min 28s
Wall time: 5min 23s
```

```
loss, accuracy = model.evaluate(x_train, y_train, batch_size=batch_size, verbose=False)
print("Training Loss:  {:.4f}".format(loss))
print("Training Accuracy:  {:.4f}".format(accuracy))
print('\n')
loss, accuracy = model.evaluate(x_val, y_val, batch_size=batch_size, verbose=False)
print("Testing Loss:  {:.4f}".format(loss))
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Loss:  0.0461
Training Accuracy:  0.9885


Testing Loss:  0.2046
Testing Accuracy:  0.9147
```

Вывод

Модель со слоем Embedding по умолчанию показал лучший скор, чем модель со слоем Embedding с предобученными векторами (Word2Vec).

✓ 4 сек. выполнено в 14:59