Введение в обработку естественного языка

Урок 8. Рекуррентные нейронные сети RNN LSTM GRU

Практическое задание

Домашнее задание к уроку 7

Задание
Данные берем отызывы за лето
На вебинаре мы говорили, что долгое время CNN и RNN архитектуры были конурирующими выяснить какая архитектура больше подходит для нашей задачи

1. построить свёрточные архитектуры
2. построить различные архитектуры с RNN
3. построить совместные архитектуры CNN -> RNN или (RNN -> CNN)

Выполнил ***Соковнин ИЛ***

B [1]:
```
import numpy as np
import pandas as pd
from string import punctuation
```

```
1.4.2
```

B [2]:
```
data = pd.read_excel(open('отзывы за лето.xls', 'rb'))
```

Out[2]:

	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
3	5	Стал зависать на 1% работы антивируса. Дальше ...	2017-08-14
4	5	Очень удобно, работает быстро.	2017-08-14

B [3]:
```
data["Content"].str.len().min(), data["Content"].str.len().max(), data["Content"].str.len().mean()
```

Out[3]:
```
(1.0, 1147.0, 56.039941902687)
```

B [4]:

Out[4]:
```
(20659, 3)
```

B [5]:

B [6]:
```
max_words = 3000
max_len = 110
num_classes = 1

# Training
epochs = 20
batch_size = 512
```

Предобработка

B [7]:
```
import pandas as pd
from string import punctuation
from stop_words import get_stop_words
from pymorphy2 import MorphAnalyzer
```

B [8]:
```
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)
```

В [9]:

Out[9]:

	Rating	Content
0	5	it just works
1	4	целое удобноной приложениеиз минус хотеть боль...
2	5	отлично

В [10]:

В [11]:
```
# # 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)
```

```
(16527, 1)
(16527,)
(2066, 1)
(2066,)
(2066, 1)
(2066,)
```

Out[11]:
```
(None, None)
```

Токенизация

В [12]:
```
train_corpus = " ".join(X_train["Content"])
```

В [13]:

Out[13]:
```
'возможно перевести другой клиент карта деньга приложение тормозить обновление хороший приложение отл'
```

В [14]:
```
import nltk
from nltk.tokenize import word_tokenize
nltk.download("punkt")
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sil\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

В [15]:

Out[15]:
```
['возможно',
 'перевести',
 'другой',
 'клиент',
 'карта',
 'деньга',
 'приложение',
 'тормозить',
 'обновление',
 'хороший']
```

Отфильтруем данные

и соберём в корпус N наиболее частых токенов

В [16]:

В [17]:
```
from nltk.probability import FreqDist
dist = FreqDist(tokens_filtered)
```

В [18]:

Out[18]:

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

B [19]:

B [20]:
```
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))
```

B [21]:
```
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)
```

B [22]:

Out[22]: (16527, 110)

Создание модели

B [23]:
```
import tensorflow as tf
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation, Input, Embedding, Conv1D, GlobalMaxPool1D, Flatten
from keras.callbacks import TensorBoard
```

B [24]:
```
import pkg_resources
```

```
keras v2.7.0
```

B [25]:
```
y_train = pd.DataFrame(y_train)
```

B [26]:
```
num_classes = 6
y_train = tf.keras.utils.to_categorical(y_train['Rating'], num_classes)
```

B [27]:
```
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: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 110, 128)          384000

 conv1d (Conv1D)             (None, 108, 128)          49280

 activation (Activation)     (None, 108, 128)          0

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense (Dense)               (None, 10)                1290

 activation_1 (Activation)   (None, 10)                0

 dense_1 (Dense)             (None, 6)                 66

 activation_2 (Activation)   (None, 6)                 0

=================================================================
Total params: 434,636
Trainable params: 434,636
Non-trainable params: 0
_________________________________________________________________
```

B [28]:
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
```

B [29]:
```
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,
```

```
Epoch 1/20
30/30 [==============================] - 6s 170ms/step - loss: 1.2709 - accuracy: 0.6938 - val_loss: 0.9597 - val_acc
uracy: 0.7120
Epoch 2/20
30/30 [==============================] - 5s 146ms/step - loss: 0.8456 - accuracy: 0.7224 - val_loss: 0.7150 - val_acc
uracy: 0.7695
```

B [30]:
```
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('\n')
print(f'max_words: {max_words}')
print(f'max_len: {max_len}')
print('Test score:', score[0])
print('Test accuracy:', score[1])
# print('Test score:', score)

# max_words = 200
# max_len = 150
# Test score: 0.8588563203811646
# Test accuracy: 0.7216843962669373

# max_words = 350
# max_len = 150
# Test score: 0.8030889630317688
# Test accuracy: 0.7434656620025635


# max_words = 500
# max_len = 150 ?
# Test score: 0.9453510642051697
# Test accuracy: 0.7163600921630859

# max_words = 250
# max_len = 250
# Test score: 0.7557735443115234
# Test accuracy: 0.75121009349823


# max_words: 500
# max_len: 60
# Test score: 0.8064351677894592
# Test accuracy: 0.6974830627441406

# max_words: 500
# max_len: 80
# Test score: 0.8367857336997986
# Test accuracy: 0.7149080634117126

# max_words: 500
# max_len: 100
# Test score: 0.8913829326629639
# Test accuracy: 0.6892545819282532

# max_words: 500
# max_len: 100
# Test score: 0.8076151609420776
# Test accuracy: 0.7100677490234375

# max_words: 500
# max_len: 110
# Test score: 0.9021356105804443
# Test accuracy: 0.7144240140914917

# max_words: 500
# max_len: 130
# Test score: 0.8075022101402283
# Test accuracy: 0.7100677490234375

# max_words: 500
# max_len: 150
# Test score: 0.7409858107566833
# Test accuracy: 0.7579864263534546
```

```
# max_words: 500
# max_len: 170
# Test score: 0.8109299540519714
# Test accuracy: 0.7018393278121948

# max_words: 500
# max_len: 200
# Test score: 0.8282581567764282
# Test accuracy: 0.7115198373794556


# max_words: 800
# max_len: 110
# Test score: 0.8676804900169373
# Test accuracy: 0.6902226805686951

# max_words: 1000
# max_len: 110
# Test score: 0.8898418545722961
# Test accuracy: 0.733785092830658

# max_words: 1200?
# max_len: 110
# Test score: 0.9137847423553467
# Test accuracy: 0.7086156606674194

# max_words: 1200
# max_len: 110
# Test score: 0.8366225361824036
# Test accuracy: 0.7429816126823425

# max_words: 1250
# max_len: 110
# Test score: 0.8289342522621155
# Test accuracy: 0.7076476216316223

# max_words: 1300
# max_len: 110
# Test score: 0.7812488079071045
# Test accuracy: 0.7555662989616394

# max_words: 1500
# max_len: 110
# Test score: 0.8564026951789856
# Test accuracy: 0.7381413578987122

# max_words: 2000
# max_len: 110
# Test score: 0.8294047117233276
# Test accuracy: 0.7357211709022522

# max_words: 3000
# max_len: 110
# Test score: 0.9440693855285645
```

```
5/5 [==============================] - 0s 68ms/step - loss: 0.7582 - accuracy: 0.7512


max_words: 3000
max_len: 110
Test score: 0.7582032680511475
Test accuracy: 0.75121009349823
```

Предобработка

B [31]:

B [32]:

Out[32]: (20659, 2)

B [33]:
```
# Сокращаем количество классов до 2
df_w2v = df_w2v[df_w2v['Rating'] != 3]
df_w2v['target'] = (df_w2v['Rating'] > 3)*1
```

B [34]:
```
df_w2v['target'] = df_w2v['target'].astype(int)
```

Out[34]:
```
1    16724
0     3024
Name: target, dtype: int64
```

B [35]:
```
df_train = df_w2v.loc[:14000]
```

B [36]:
```
df_train['Content'] = df_train['Content'].apply(preprocess_text)
```

```
<ipython-input-36-18a355c3bd21>: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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_train['Content'] = df_train['Content'].apply(preprocess_text)
<ipython-input-36-18a355c3bd21>: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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_val['Content'] = df_val['Content'].apply(preprocess_text)
```

Токенизация

B [37]:
```
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)
```

```
<ipython-input-37-7fdbbb1602b8>:13: 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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_train['Content'] = df_train['Content'].apply(preprocess_text)
<ipython-input-37-7fdbbb1602b8>:14: 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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_val['Content'] = df_val['Content'].apply(preprocess_text)
```

B [38]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

B [39]:
```
text_corpus_train = df_train['Content'].values
text_corpus_valid = df_val['Content'].values
```

B [40]:
```
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_valid)
# sequences_test = tokenizer.texts_to_sequences(text_corpus_test)

#
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)
```

B [41]:

Out[41]:
```
13404
```

B [42]:
```
y_train = df_train['target'].values
```

B [43]:

Out[43]:
```
((13404, 100), (13404,))
```

B [44]:

Out[44]:
```
(array([1, 1, 1, ..., 1, 1, 1]),
                                                  Content  target
 0                                          it just works       1
 1      целое удобнона приложениеиз минус большой дост...       1
 2                                                 отлично       1
 3      зависать 1 работа антивирус ранее пользоваться...       1
 4                                удобно работать быстро       1
 ...                                                  ...     ...
 13996                                             норма       1
 13997  приложение ужасный пользоваться чтонуть выбор ...       0
 13998                                           устроить       1
 13999                                        удобнобыстро       1
 14000                                                           1

 [13404 rows x 2 columns])
```

B [45]:
```
import matplotlib.pyplot as plt

def plot_history(history):

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))

    ax1.plot(history.history['accuracy'])
    ax1.plot(history.history['val_accuracy'])
    ax1.set_title('model accuracy')
    ax1.set(xlabel='epoch', ylabel='accuracy')
    ax1.legend(['train', 'test'], loc='upper left')

    ax2.plot(history.history['loss'])
    ax2.plot(history.history['val_loss'])
    ax2.set_title('model loss')
    ax2.set(xlabel='epoch', ylabel='loss')
```

1. построить свёрточные архитектуры

B [46]:

B [47]:
```
num_classes = 1
model = Sequential()
# model.add(Embedding(input_dim=max_words, output_dim=128, input_length=max_len))
model.add(Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))

model.add(Masking(mask_value=0.0))
model.add(Conv1D(128, 3))
model.add(Activation("relu"))
model.add(GlobalMaxPool1D())
model.add(Dense(10, activation='relu'))
model.add(Dense(num_classes, activation='sigmoid'))
model.summary()
```

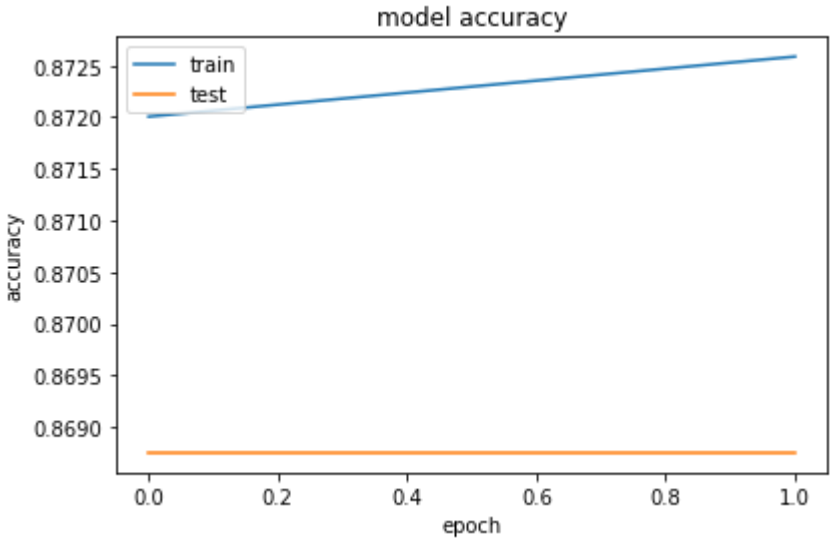
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 128)          1145344
```

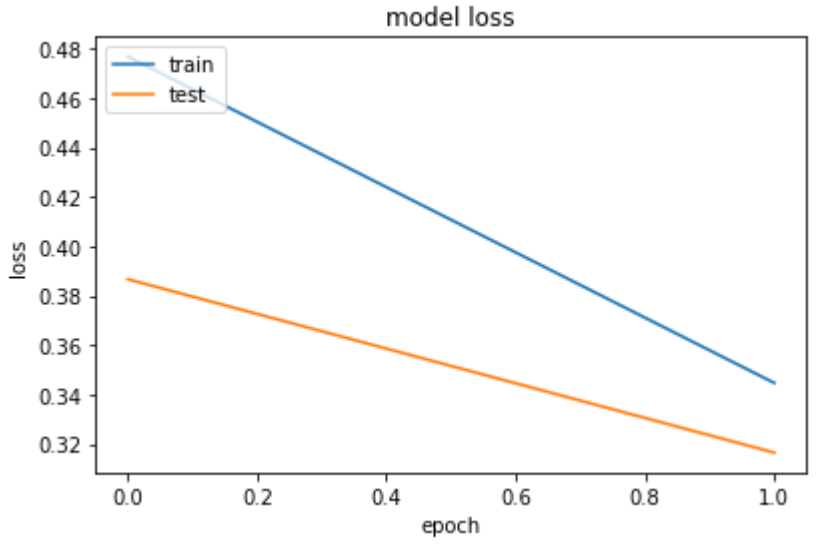
B [48]:
```
# batch_size
```

Out[48]:
```
((13404, 100), (13404,))
```

B [49]:
```
early_stopping=EarlyStopping(monitor='val_loss')
tensorboard=TensorBoard(log_dir='./logs', write_graph=True, write_images=True)

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
24/24 [==============================] - 6s 188ms/step - loss: 0.4768 - accuracy: 0.8720 - val_loss: 0.3867 - val_acc
uracy: 0.8688
Epoch 2/20
24/24 [==============================] - 4s 158ms/step - loss: 0.3447 - accuracy: 0.8726 - val_loss: 0.3165 - val_acc
uracy: 0.8688
```





B [50]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

```
13/13 [==============================] - 1s 63ms/step - loss: 0.4023 - accuracy: 0.7933


Test score: 0.40234702825546265
Test accuracy: 0.7933480739593506
```

2. Построить различные архитектуры с RNN

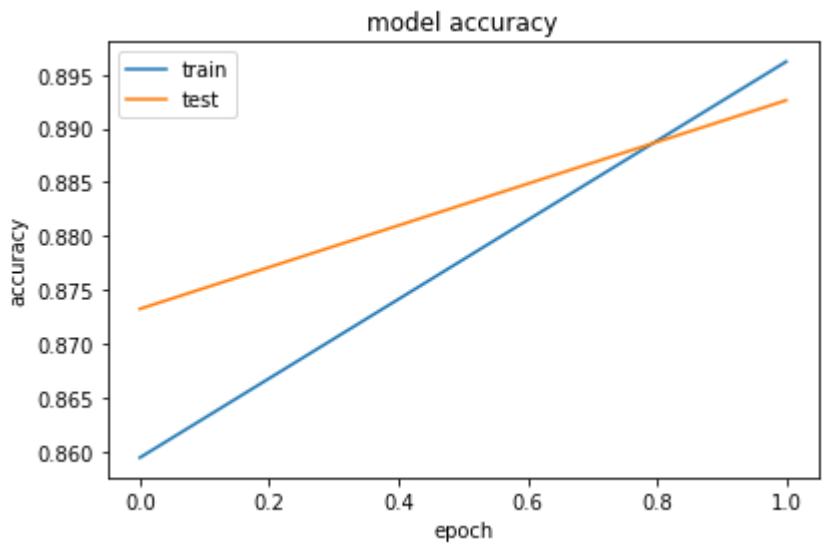
Простая рекурентная модель SimpleRNN

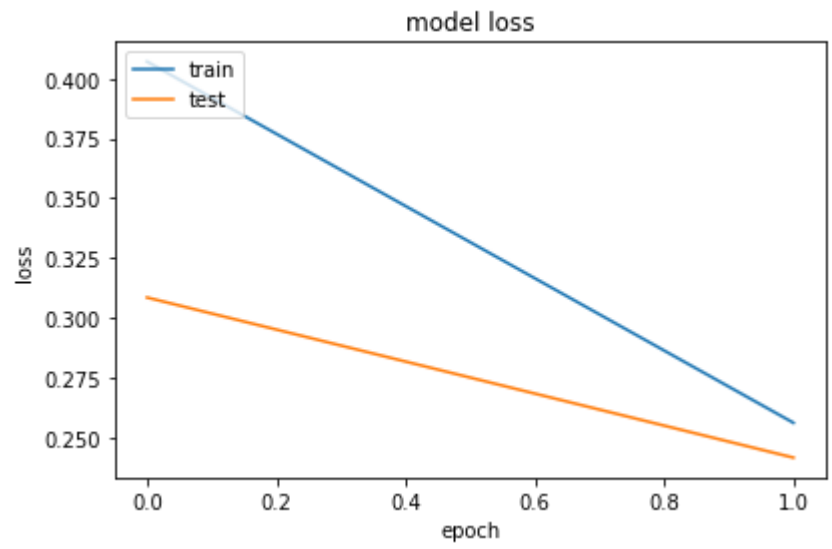
B [51]:
```
model = Sequential()

model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))
model.add(Masking(mask_value=0.0))

model.add(SimpleRNN(64))  # 64 - рекурентных ячеёки (количество слоёв, RNN-ячеек)!

model.add(Dense(124, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

B [52]:
```
early_stopping=EarlyStopping(monitor='val_loss')
tensorboard=TensorBoard(log_dir='./logs', write_graph=True, write_images=True)
```

```
print(X_train.shape, y_train.shape)

history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    # callbacks=[early_stopping])
                    callbacks=[tensorboard, early_stopping])
```

```
(13404, 100) (13404,)
Epoch 1/10
24/24 [==============================] - 14s 461ms/step - loss: 0.4071 - accuracy: 0.8594 - val_loss: 0.3086 - val_ac
curacy: 0.8732
Epoch 2/10
24/24 [==============================] - 11s 459ms/step - loss: 0.2563 - accuracy: 0.8962 - val_loss: 0.2417 - val_ac
curacy: 0.8926
```





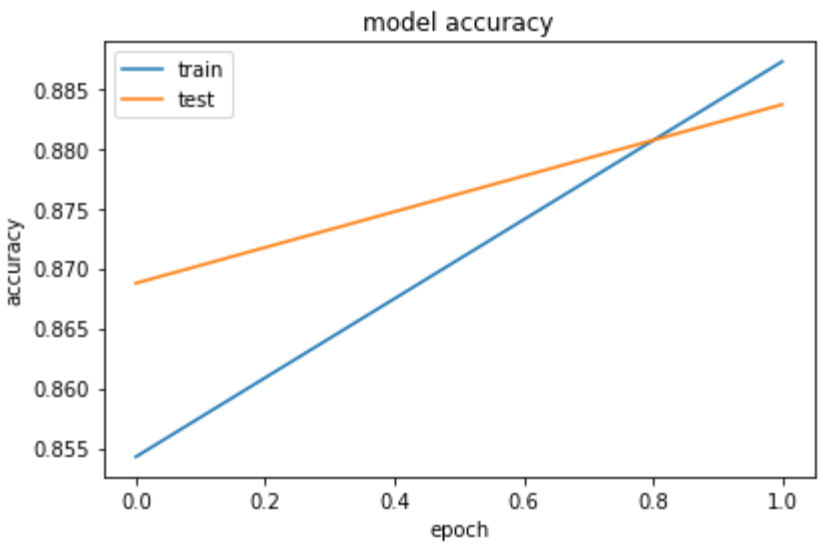
B [53]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

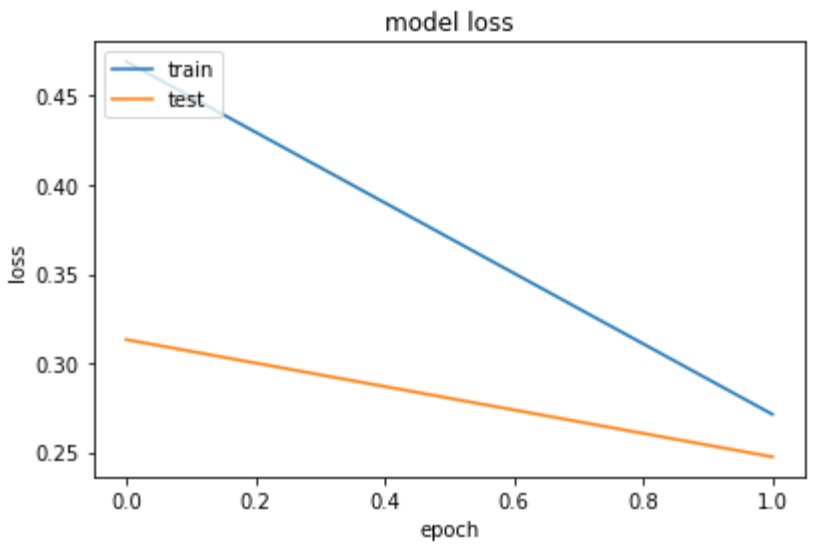
```
13/13 [==============================] - 2s 160ms/step - loss: 0.2989 - accuracy: 0.8671


Test score: 0.2988642454147339
Test accuracy: 0.8671185374259949
```

LSTM

B [54]:
```
model = Sequential()

model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))
model.add(Masking(mask_value=0.0))
model.add(LSTM(64, recurrent_dropout=0.2))  # recurrent_dropout рвёт связи между RNN-ячейками
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

early_stopping=EarlyStopping(monitor='val_loss')


history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[early_stopping])
```

```
Epoch 1/10
24/24 [==============================] - 15s 446ms/step - loss: 0.4693 - accuracy: 0.8543 - val_loss: 0.3133 - val_ac
curacy: 0.8688
Epoch 2/10
24/24 [==============================] - 12s 493ms/step - loss: 0.2713 - accuracy: 0.8873 - val_loss: 0.2474 - val_ac
curacy: 0.8837
```





B [55]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

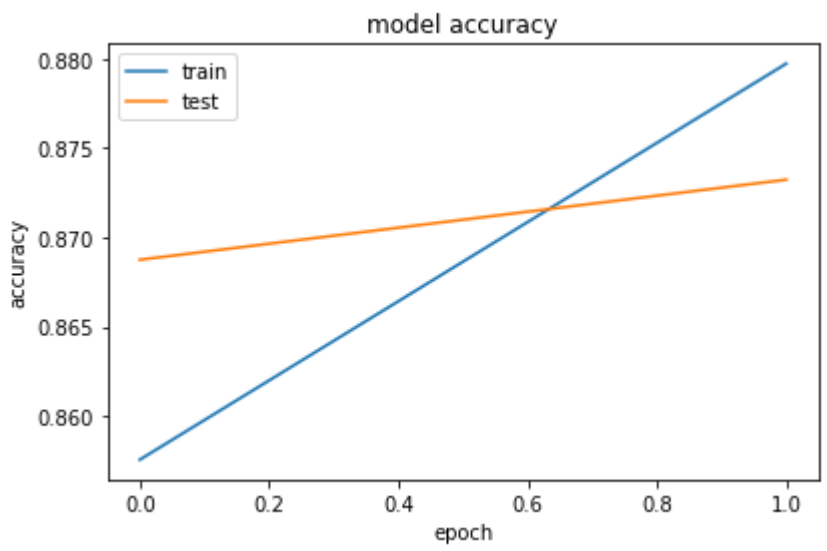
```
13/13 [==============================] - 3s 197ms/step - loss: 0.3091 - accuracy: 0.8677


Test score: 0.3091104030609131
Test accuracy: 0.867749035358429
```

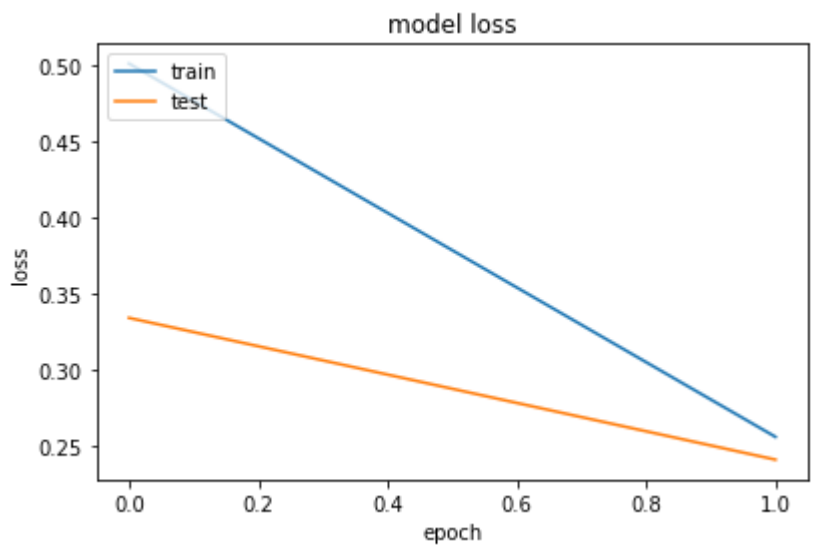
GRU

B [56]:
```
model = Sequential()

model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))
model.add(Masking(mask_value=0.0))
model.add(GRU(64, recurrent_dropout=0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

early_stopping=EarlyStopping(monitor='val_loss')
```

```
history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[early_stopping])
```

```
Epoch 1/10
24/24 [==============================] - 15s 509ms/step - loss: 0.5016 - accuracy: 0.8576 - val_loss: 0.3342 - val_ac
curacy: 0.8688
Epoch 2/10
24/24 [==============================] - 11s 439ms/step - loss: 0.2559 - accuracy: 0.8797 - val_loss: 0.2409 - val_ac
curacy: 0.8732
```





B [57]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

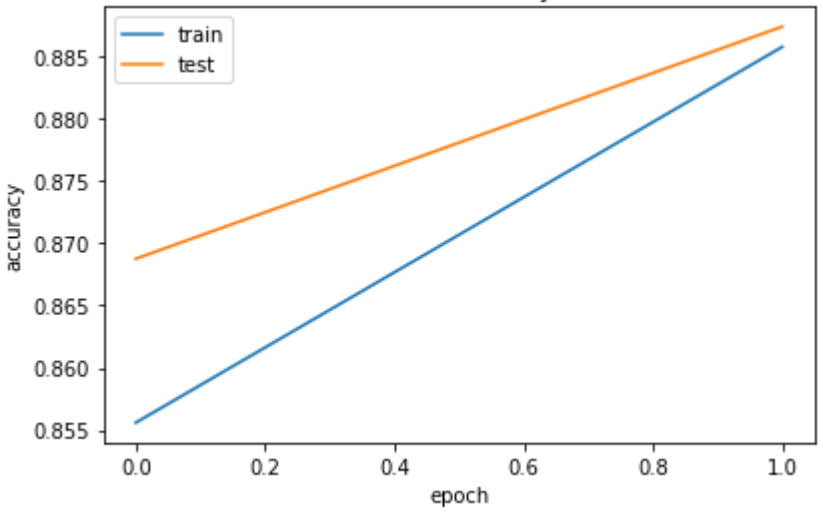
```
13/13 [==============================] - 3s 261ms/step - loss: 0.3140 - accuracy: 0.8428


Test score: 0.31398484110832214
Test accuracy: 0.8428436517715454
```

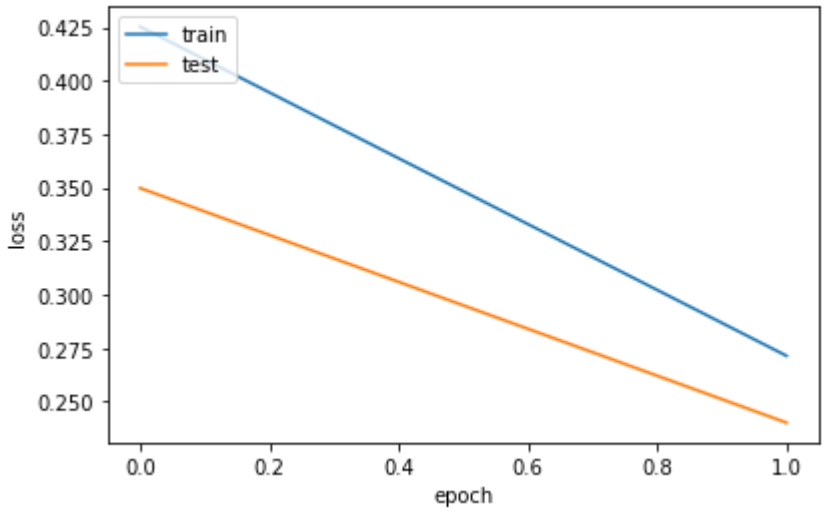
3. Построить совместные архитектуры CNN -> RNN или (RNN -> CNN)

CNN + RNN

B [58]:
```
model = Sequential()

model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))
model.add(Masking(mask_value=0.0))

model.add(Conv1D(filters=64, kernel_size=3, activation='relu', padding="same"))

model.add(SimpleRNN(64))  # 64 - рекурентных ячеёки (количество слоёв, RNN-ячеек)!

model.add(Dense(124, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

model.summary()


history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[early_stopping])
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 100, 128)          1145344

 masking_4 (Masking)         (None, 100, 128)          0

 conv1d_2 (Conv1D)           (None, 100, 64)           24640

 simple_rnn_1 (SimpleRNN)    (None, 64)                8256

 dense_10 (Dense)            (None, 124)               8060

 dropout_3 (Dropout)         (None, 124)               0

 dense_11 (Dense)            (None, 1)                 125

=================================================================
Total params: 1,186,425
Trainable params: 1,186,425
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
24/24 [==============================] - 7s 205ms/step - loss: 0.4253 - accuracy: 0.8556 - val_loss: 0.3498 - val_acc
uracy: 0.8688
```





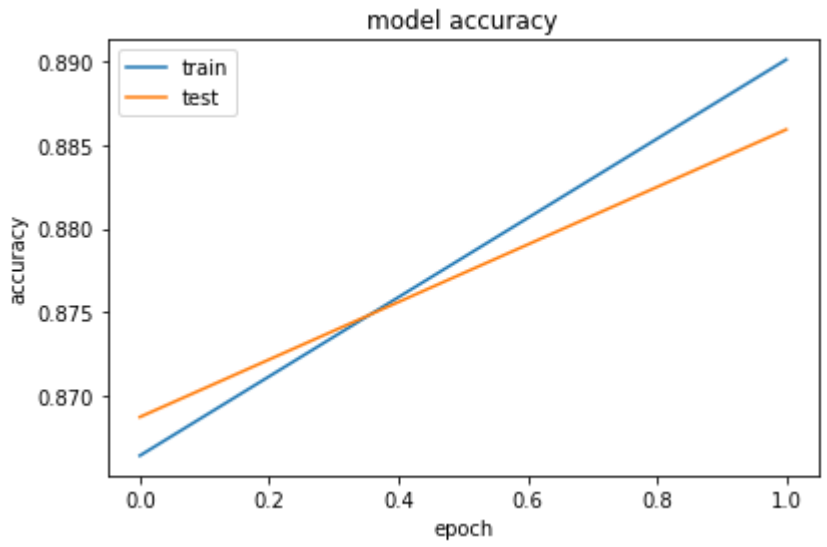
B [59]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

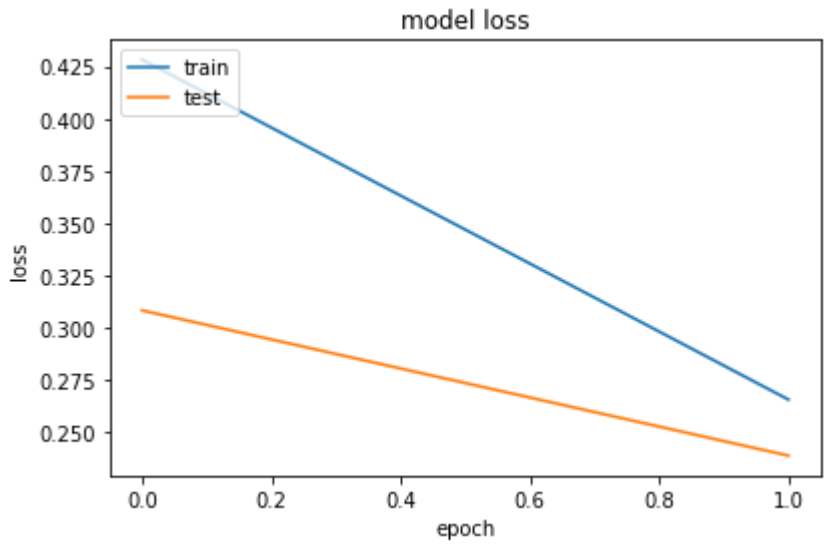
```
13/13 [==============================] - 1s 101ms/step - loss: 0.3130 - accuracy: 0.8515


Test score: 0.313017874956131
Test accuracy: 0.8515132665634155
```

CNN + LSTM

B [60]:
```
model = Sequential()
model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))
model.add(Masking(mask_value=0.0))

model.add(Conv1D(filters=64, kernel_size=3, activation='relu', padding="same"))
model.add(LSTM(64, recurrent_dropout=0.2))  # recurrent_dropout рвёт связи между RNN-ячейками
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

early_stopping=EarlyStopping(monitor='val_loss')


history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[early_stopping])
```

```
Epoch 1/10
24/24 [==============================] - 14s 442ms/step - loss: 0.4282 - accuracy: 0.8665 - val_loss: 0.3081 - val_ac
curacy: 0.8688
Epoch 2/10
24/24 [==============================] - 11s 429ms/step - loss: 0.2653 - accuracy: 0.8901 - val_loss: 0.2385 - val_ac
curacy: 0.8859
```





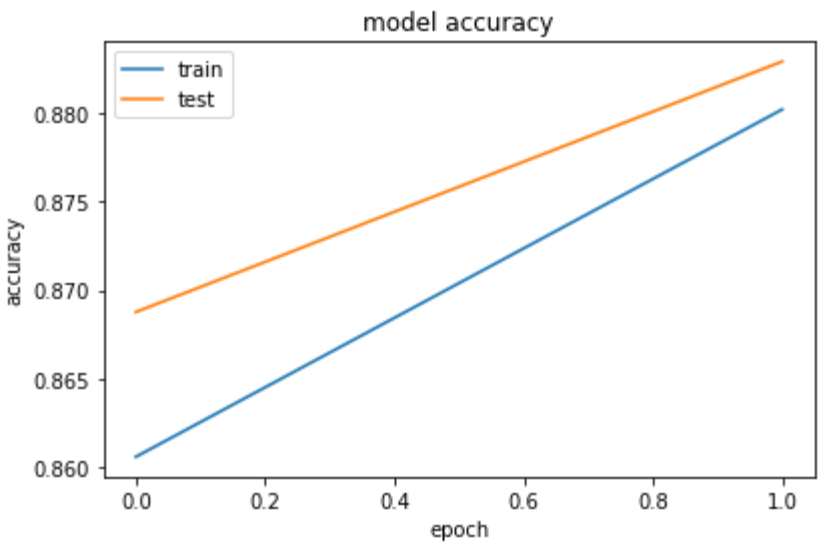
B [61]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

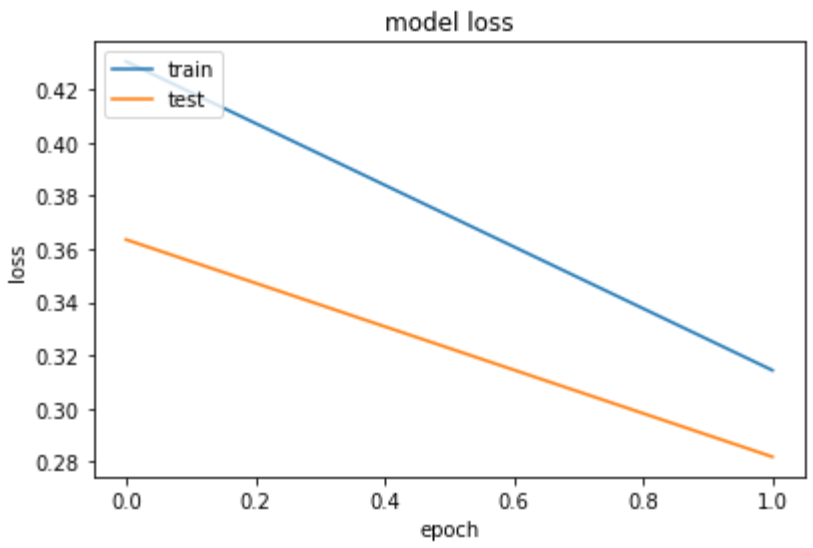
```
13/13 [==============================] - 3s 197ms/step - loss: 0.3049 - accuracy: 0.8635


Test score: 0.30487334728240967
Test accuracy: 0.8634930849075317
```

LSTM + CNN

B [62]:
```
model = Sequential()

model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=30,
              trainable=True,
              mask_zero=True)
)

model.add(Masking(mask_value=0.0))

model.add(LSTM(64, recurrent_dropout=0.2, return_sequences=True))  # recurrent_dropout рвёт связи между RNN-ячейками
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', padding="same"))


model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

early_stopping=EarlyStopping(monitor='val_loss')


history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[early_stopping])
```

```
Epoch 1/10
24/24 [==============================] - 13s 425ms/step - loss: 0.4306 - accuracy: 0.8606 - val_loss: 0.3636 - val_ac
curacy: 0.8688
Epoch 2/10
24/24 [==============================] - 10s 396ms/step - loss: 0.3143 - accuracy: 0.8802 - val_loss: 0.2818 - val_ac
curacy: 0.8829
```





B [63]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

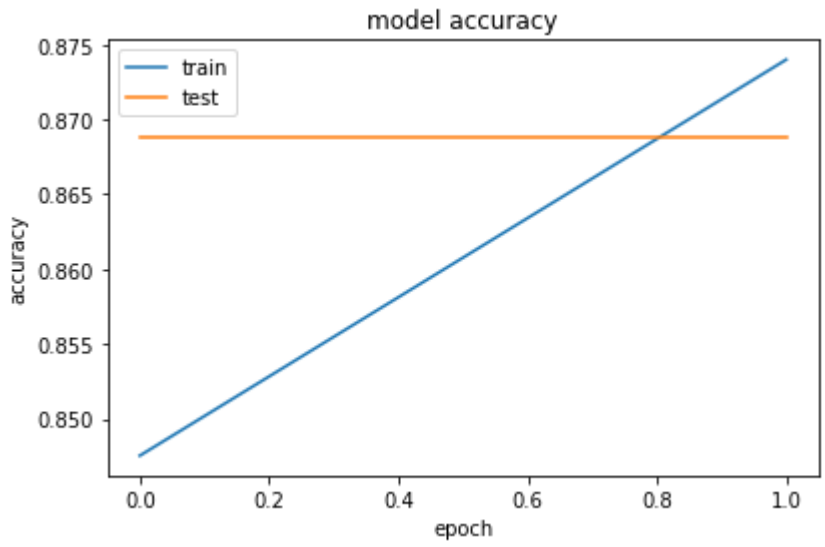
```
13/13 [==============================] - 4s 309ms/step - loss: 0.3625 - accuracy: 0.8482


Test score: 0.3624567687511444
Test accuracy: 0.8482030034065247
```

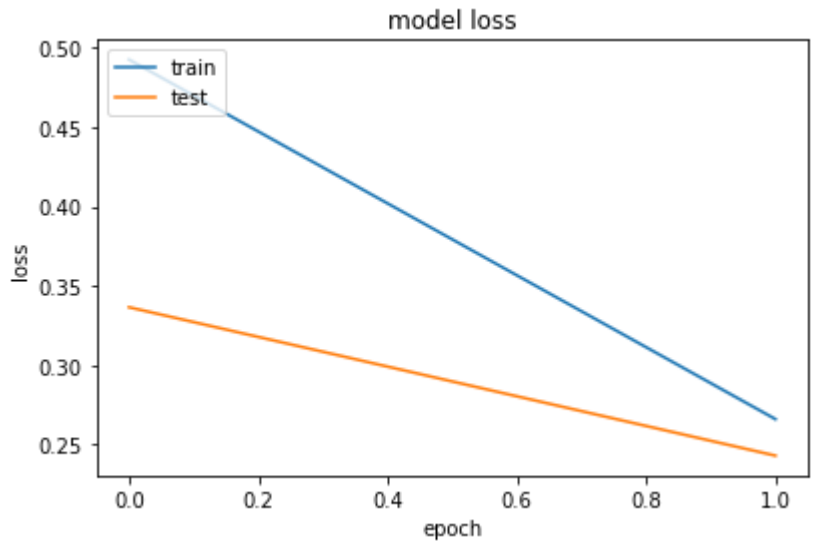
CNN + GRU

B [64]:
```
model = Sequential()

model.add(
    Embedding(input_dim=word_count,
              input_length=training_length,
              output_dim=128,
              trainable=True,
              mask_zero=True))
model.add(Masking(mask_value=0.0))

model.add(Conv1D(filters=64, kernel_size=3, activation='relu', padding="same"))
model.add(GRU(64, recurrent_dropout=0.2))
```

```
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

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

early_stopping=EarlyStopping(monitor='val_loss')


history = model.fit(X_train, y_train,
                    batch_size=512,
                    epochs=10,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[early_stopping])
```

```
Epoch 1/10
24/24 [==============================] - 12s 380ms/step - loss: 0.4926 - accuracy: 0.8476 - val_loss: 0.3364 - val_ac
curacy: 0.8688
Epoch 2/10
24/24 [==============================] - 9s 363ms/step - loss: 0.2659 - accuracy: 0.8740 - val_loss: 0.2428 - val_acc
uracy: 0.8688
```





B [65]:
```
score = model.evaluate(X_valid, y_val, batch_size=512, verbose=1)
print('\n')
print('Test score:', score[0])
```

```
13/13 [==============================] - 4s 279ms/step - loss: 0.3275 - accuracy: 0.8222


Test score: 0.32753103971481323
Test accuracy: 0.8221942186355591
```

Выводы. Лучше всего сработали (в порядке убывания результата) следующие архитектуры:

1. SimpleRNN
2. LSTM
3. GRU
4. CNN

Их результаты отличаются не сильно.

SimpleRNN: (0.3091506361961365, 0.8668032884597778)
LSTM: (0.31315749883651733, 0.8668032884597778)
GRU: (0.31110599637031555, 0.8477301597595215)
CNN: (0.35694921016693115, 0.7933480739593506)

CNN + SimpleRNN: (0.2870107591152191, 0.8748423457145691)
CNN + LSTM: (0.3044297993183136, 0.8628625273704529)
CNN + GRU: (0.31573957204818726, 0.8294451236724854)

B [ ]: