**Лабораторная работа № 3**

**«Сети LSTM»**

**Задание № 1:** Классификация текстов.

LSTM-сеть должна классифицировать новости по 4 темам: World, Sports, Business, Sci/Tech.

Реализация на языке Python:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Dense, Dropout, LSTM, InputLayer

from tensorflow.keras import regularizers

from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras import utils

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras import utils

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

# Максимальное количество слов

num\_words = 10000

# Максимальная длина новости

max\_news\_len = 30

# Количество классов новостей

nb\_classes = 4

train = pd.read\_csv('train.csv', header=None, names=['class', 'title', 'text'])

news = train['text']

y\_train = utils.to\_categorical(train['class'] - 1, nb\_classes)

tokenizer = Tokenizer(num\_words=num\_words)

tokenizer.fit\_on\_texts(news)

sequences = tokenizer.texts\_to\_sequences(news)

N = len(sequences)

List = list()

for i in range(N):

List.append([sequences[i], y\_train[i]])

List2 = sorted(List, key=lambda data: len(data[0]))

sequences2 = list()

y\_train2 = np.ndarray(shape = y\_train.shape, dtype = np.float32())

for i in range(N):

sequences2.append(List2[i][0])

y\_train2[i] = np.float32(List2[i][1])

x\_train = pad\_sequences(sequences, maxlen=max\_news\_len, padding = 'post')

x\_train2 = pad\_sequences(sequences2, maxlen=max\_news\_len, padding = 'post')

l2\_lambda = 0.0001

model\_lstm = Sequential()

model\_lstm.add(Embedding(num\_words, 32, input\_length=max\_news\_len))

model\_lstm.add(Dropout(0.9))

model\_lstm.add(LSTM(16, dropout=0.5, recurrent\_dropout=0.2, kernel\_regularizer=regularizers.l2(l2\_lambda)))

model\_lstm.add(Dense(4, activation='softmax'))

model\_lstm.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model\_lstm.summary()

history\_lstm = model\_lstm.fit(x\_train2, y\_train2, epochs=50, batch\_size=1536, validation\_split=0.1, callbacks=[EarlyStopping(monitor='val\_loss', patience=5)])

Результаты обучения:

Train on 108000 samples, validate on 12000 samples

Epoch 1/50

WARNING:tensorflow:Large dropout rate: 0.9 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep\_prob. Please ensure that this is intended.

108000/108000 [==============================] - 19s 173us/sample - loss: 1.3594 - accuracy: 0.3302 - val\_loss: 1.1126 - val\_accuracy: 0.5033

Epoch 2/50

108000/108000 [==============================] - 10s 93us/sample - loss: 1.0726 - accuracy: 0.5172 - val\_loss: 0.8210 - val\_accuracy: 0.6224

Epoch 3/50

108000/108000 [==============================] - 10s 92us/sample - loss: 0.8743 - accuracy: 0.6145 - val\_loss: 0.7098 - val\_accuracy: 0.6823

Epoch 4/50

108000/108000 [==============================] - 10s 94us/sample - loss: 0.7700 - accuracy: 0.6635 - val\_loss: 0.6694 - val\_accuracy: 0.7583

Epoch 5/50

108000/108000 [==============================] - 10s 94us/sample - loss: 0.7155 - accuracy: 0.6949 - val\_loss: 0.6268 - val\_accuracy: 0.7962

Epoch 6/50

108000/108000 [==============================] - 10s 94us/sample - loss: 0.6697 - accuracy: 0.7288 - val\_loss: 0.5691 - val\_accuracy: 0.8317

Epoch 7/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.6271 - accuracy: 0.7600 - val\_loss: 0.5198 - val\_accuracy: 0.8446

Epoch 8/50

108000/108000 [==============================] - 10s 95us/sample - loss: 0.5920 - accuracy: 0.7811 - val\_loss: 0.4997 - val\_accuracy: 0.8474

Epoch 9/50

108000/108000 [==============================] - 10s 92us/sample - loss: 0.5646 - accuracy: 0.7950 - val\_loss: 0.4823 - val\_accuracy: 0.8532

Epoch 10/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.5480 - accuracy: 0.8029 - val\_loss: 0.4745 - val\_accuracy: 0.8544

Epoch 11/50

108000/108000 [==============================] - 10s 92us/sample - loss: 0.5280 - accuracy: 0.8107 - val\_loss: 0.4689 - val\_accuracy: 0.8571

Epoch 12/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.5177 - accuracy: 0.8160 - val\_loss: 0.4660 - val\_accuracy: 0.8555

Epoch 13/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.5087 - accuracy: 0.8201 - val\_loss: 0.4593 - val\_accuracy: 0.8581

Epoch 14/50

108000/108000 [==============================] - 10s 92us/sample - loss: 0.4999 - accuracy: 0.8240 - val\_loss: 0.4555 - val\_accuracy: 0.8594

Epoch 15/50

108000/108000 [==============================] - 10s 92us/sample - loss: 0.4939 - accuracy: 0.8256 - val\_loss: 0.4531 - val\_accuracy: 0.8615

Epoch 16/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4830 - accuracy: 0.8301 - val\_loss: 0.4532 - val\_accuracy: 0.8603

Epoch 17/50

108000/108000 [==============================] - 10s 96us/sample - loss: 0.4771 - accuracy: 0.8321 - val\_loss: 0.4535 - val\_accuracy: 0.8602

Epoch 18/50

108000/108000 [==============================] - 10s 95us/sample - loss: 0.4722 - accuracy: 0.8340 - val\_loss: 0.4474 - val\_accuracy: 0.8613

Epoch 19/50

108000/108000 [==============================] - 10s 94us/sample - loss: 0.4689 - accuracy: 0.8352 - val\_loss: 0.4470 - val\_accuracy: 0.8626

Epoch 20/50

108000/108000 [==============================] - 10s 95us/sample - loss: 0.4629 - accuracy: 0.8380 - val\_loss: 0.4466 - val\_accuracy: 0.8622

Epoch 21/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4575 - accuracy: 0.8392 - val\_loss: 0.4431 - val\_accuracy: 0.8648

Epoch 22/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4522 - accuracy: 0.8412 - val\_loss: 0.4438 - val\_accuracy: 0.8638

Epoch 23/50

108000/108000 [==============================] - 10s 97us/sample - loss: 0.4457 - accuracy: 0.8431 - val\_loss: 0.4456 - val\_accuracy: 0.8639

Epoch 24/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4469 - accuracy: 0.8438 - val\_loss: 0.4457 - val\_accuracy: 0.8630

Epoch 25/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4439 - accuracy: 0.8445 - val\_loss: 0.4438 - val\_accuracy: 0.8640

Epoch 26/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4446 - accuracy: 0.8440 - val\_loss: 0.4386 - val\_accuracy: 0.8658

Epoch 27/50

108000/108000 [==============================] - 10s 93us/sample - loss: 0.4384 - accuracy: 0.8477 - val\_loss: 0.4393 - val\_accuracy: 0.8643

Epoch 28/50

108000/108000 [==============================] - 10s 94us/sample - loss: 0.4345 - accuracy: 0.8494 - val\_loss: 0.4403 - val\_accuracy: 0.8642

Epoch 29/50

108000/108000 [==============================] - 10s 95us/sample - loss: 0.4320 - accuracy: 0.8489 - val\_loss: 0.4396 - val\_accuracy: 0.8653

Epoch 30/50

108000/108000 [==============================] - 10s 96us/sample - loss: 0.4306 - accuracy: 0.8493 - val\_loss: 0.4386 - val\_accuracy: 0.8652

Epoch 31/50

108000/108000 [==============================] - 10s 95us/sample - loss: 0.4292 - accuracy: 0.8507 - val\_loss: 0.4406 - val\_accuracy: 0.8645

Построим график, иллюстрирующий зависимость точности нейронной сети от количества пройденных эпох обучения:

plt.plot(history\_lstm.history['accuracy'],

label='Доля верных ответов на обучающем наборе')

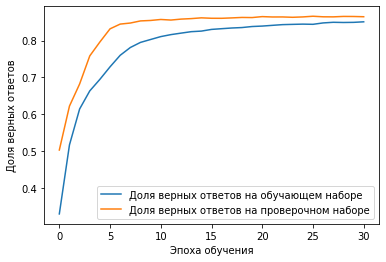
plt.plot(history\_lstm.history['val\_accuracy'],

label='Доля верных ответов на проверочном наборе')

plt.xlabel('Эпоха обучения')

plt.ylabel('Доля верных ответов')

plt.legend()

plt.show()

Сохраним результат и проверим работу сети:

model\_lstm.save('LSTM.h5')

test = pd.read\_csv('test.csv', header=None, names=['class', 'title', 'text'])

test\_sequences = tokenizer.texts\_to\_sequences(test['text'])

x\_test = pad\_sequences(test\_sequences, maxlen=max\_news\_len, padding = 'post')

y\_test = utils.to\_categorical(test['class'] - 1, nb\_classes)

res = model\_lstm.evaluate(x\_test, y\_test, verbose=0)

print('val\_loss:',res[0],'\nval\_accuracy:',res[1])

Получим:

val\_loss: 0.32516818248911905

val\_accuracy: 0.8975

Таким образом, наша сеть предсказывает принадлежность новости к одной из тем с точностью более 89 %.

**Задание № 2:** Предсказание времени отказа для ракетных двигателей.

LSTM-сеть должна предсказывать количество циклов работы двигателя, оставшееся до отказа работы.

Реализация на языке Python:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Dense, Dropout, LSTM, InputLayer

from tensorflow.keras import regularizers

from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras import utils

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from tensorflow.keras import utils

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

str\_train = './CMAPSSData/train\_FD001.txt'

train\_data = pd.read\_csv(str\_train, sep = ' ', header=None)

train\_data\_np = np.array(train\_data)

count\_n = np.int32(max(train\_data\_np[:,0]))

Len\_seqs = np.zeros(count\_n)

count = 1

j = 0

for i in range(1,train\_data\_np.shape[0]):

if (train\_data\_np[i,0] == train\_data\_np[i - 1,0]):

count += 1

else:

Len\_seqs[j] = count

j += 1

count = 1

Len\_seqs[j] = count

max\_len = np.max(Len\_seqs)

mas = list()

for i in range(train\_data\_np.shape[1]):

if (np.max(train\_data\_np[:,i]) == np.min(train\_data\_np[:,i])):

mas.append(i)

mas.append(26)

mas.append(27)

train\_data = train\_data.drop(mas, axis=1)

train\_data\_np = np.array(train\_data)

for i in range(2,train\_data\_np.shape[1]):

m = np.mean(train\_data\_np[:,i])

train\_data\_np[:,i] = (train\_data\_np[:,i] - m)/np.std(train\_data\_np[:,i], ddof = 1)

temp = max\_len - 1

x\_train = np.zeros([int(count\_n\*max\_len),train\_data\_np.shape[1] - 2])

i = 1

j = 1

x\_train[0,:] = train\_data\_np[0,2:]

while (i < train\_data\_np.shape[0]):

if (train\_data\_np[int(i),0] == train\_data\_np[int(i - 1),0]):

x\_train[int(j),:] = train\_data\_np[int(i),2:]

temp -= 1

else:

if (j + temp < x\_train.shape[0]):

x\_train[int(j + temp),:] = train\_data\_np[int(i),2:]

j += temp

temp = max\_len - 1

else:

break

j += 1

i += 1

y\_train = np.zeros(int(count\_n))

y\_train[int(count\_n - 1)] = train\_data\_np[train\_data\_np.shape[0] - 1,1]

j = count\_n - 1

i = train\_data\_np.shape[0] - 2

while (i != -1):

if (train\_data\_np[int(i),0] != train\_data\_np[int(i + 1),0]):

j -= 1

y\_train[int(j)] = train\_data\_np[int(i),1]

i -= 1

for i in range(y\_train.shape[0]):

if (y\_train[i] > 150):

y\_train[i] = 150

x\_train = np.reshape(x\_train, [y\_train.shape[0], int(max\_len), train\_data\_np.shape[1] - 2])

idxTransform = sorted(range(len(Len\_seqs)), key=lambda k: -Len\_seqs[k])

x\_train = x\_train[idxTransform]

y\_train = y\_train[idxTransform]

str\_test = './CMAPSSData/test\_FD001.txt'

test\_data = pd.read\_csv(str\_test, sep = ' ', header=None)

test\_data\_np = np.array(test\_data)

count\_n = np.int32(max(test\_data\_np[:,0]))

mas = list()

for i in range(test\_data\_np.shape[1]):

if (np.max(test\_data\_np[:,i]) == np.min(test\_data\_np[:,i])):

mas.append(i)

mas.append(26)

mas.append(27)

test\_data = test\_data.drop(mas, axis=1)

test\_data\_np = np.array(test\_data)

for i in range(2,test\_data\_np.shape[1]):

m = np.mean(test\_data\_np[:,i])

test\_data\_np[:,i] = (test\_data\_np[:,i] - m)/np.std(test\_data\_np[:,i], ddof = 1)

temp = max\_len - 1

x\_test = np.zeros([int(count\_n\*max\_len),test\_data\_np.shape[1] - 2])

i = 1

j = 1

x\_test[0,:] = test\_data\_np[0,2:]

while (i < test\_data\_np.shape[0]):

if (test\_data\_np[int(i),0] == test\_data\_np[int(i - 1),0]):

x\_test[int(j),:] = test\_data\_np[int(i),2:]

temp -= 1

else:

if (j + temp < x\_test.shape[0]):

#x\_train[int(j):int(j + temp),:] = 0

x\_test[int(j + temp),:] = test\_data\_np[int(i),2:]

j += temp

temp = max\_len - 1

else:

#x\_train[int(j):,:] = 0

break

j += 1

i += 1

handle = open('./CMAPSSData/RUL\_FD001.txt', 'r')

a = list()

for line in handle:

a.append(np.float32(line))

handle.close()

y\_test = np.array(a)

x\_test = np.reshape(x\_test, [y\_test.shape[0], int(max\_len), test\_data\_np.shape[1] - 2])

l2\_lambda = 0.0001

model\_lstm2 = Sequential()

model\_lstm2.add(InputLayer(input\_shape=[max\_len, train\_data\_np.shape[1] - 2]))

model\_lstm2.add(BatchNormalization(axis=1))

model\_lstm2.add(LSTM(300, kernel\_regularizer=regularizers.l2(l2\_lambda)))

model\_lstm2.add(Dropout(0.7))

#model\_lstm2.add(LSTM(100, kernel\_regularizer=regularizers.l2(l2\_lambda), return\_sequences = True))

model\_lstm2.add(Dense(50))

model\_lstm2.add(Dropout(0.6))

model\_lstm2.add(Dense(1))

model\_lstm2.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

model\_lstm2.summary()

checkpoint = ModelCheckpoint('LSTM2.h5', monitor='val\_loss', save\_best\_only=True, verbose=1) # сохраним веса, дающие наименьшее значение функции потерь

history\_lstm2 = model\_lstm2.fit(x\_train, y\_train, epochs=50, batch\_size=15, validation\_data = (x\_test, y\_test), callbacks=[checkpoint, EarlyStopping(monitor='val\_loss', patience=5)])

Результаты обучения:

Train on 100 samples, validate on 100 samples

Epoch 1/50

90/100 [==========================>...] - ETA: 1s - loss: 22280.6589 - accuracy: 0.0000e+00

Epoch 00001: val\_loss improved from inf to 6923.17375, saving model to LSTM2.h5

100/100 [==============================] - 19s 191ms/sample - loss: 22247.5068 - accuracy: 0.0000e+00 - val\_loss: 6923.1738 - val\_accuracy: 0.0000e+00

Epoch 2/50

90/100 [==========================>...] - ETA: 0s - loss: 18666.5734 - accuracy: 0.0000e+00

Epoch 00002: val\_loss improved from 6923.17375 to 6315.59603, saving model to LSTM2.h5

100/100 [==============================] - 9s 91ms/sample - loss: 18533.2379 - accuracy: 0.0000e+00 - val\_loss: 6315.5960 - val\_accuracy: 0.0000e+00

Epoch 3/50

90/100 [==========================>...] - ETA: 0s - loss: 14893.2949 - accuracy: 0.0000e+00

Epoch 00003: val\_loss improved from 6315.59603 to 5518.60110, saving model to LSTM2.h5

100/100 [==============================] - 11s 106ms/sample - loss: 14713.1880 - accuracy: 0.0000e+00 - val\_loss: 5518.6011 - val\_accuracy: 0.0000e+00

Epoch 4/50

90/100 [==========================>...] - ETA: 0s - loss: 11008.8112 - accuracy: 0.0000e+00

Epoch 00004: val\_loss improved from 5518.60110 to 5394.35460, saving model to LSTM2.h5

100/100 [==============================] - 10s 100ms/sample - loss: 10668.2755 - accuracy: 0.0000e+00 - val\_loss: 5394.3546 - val\_accuracy: 0.0000e+00

Epoch 5/50

90/100 [==========================>...] - ETA: 0s - loss: 6767.3076 - accuracy: 0.0000e+00

Epoch 00005: val\_loss did not improve from 5394.35460

100/100 [==============================] - 10s 98ms/sample - loss: 6549.7086 - accuracy: 0.0000e+00 - val\_loss: 5557.5505 - val\_accuracy: 0.0000e+00

Epoch 6/50

90/100 [==========================>...] - ETA: 0s - loss: 3859.7071 - accuracy: 0.0000e+00

Epoch 00006: val\_loss improved from 5394.35460 to 3094.54386, saving model to LSTM2.h5

100/100 [==============================] - 10s 102ms/sample - loss: 3676.3052 - accuracy: 0.0000e+00 - val\_loss: 3094.5439 - val\_accuracy: 0.0000e+00

Epoch 7/50

90/100 [==========================>...] - ETA: 0s - loss: 1941.1155 - accuracy: 0.0000e+00

Epoch 00007: val\_loss did not improve from 3094.54386

100/100 [==============================] - 11s 107ms/sample - loss: 1844.9639 - accuracy: 0.0000e+00 - val\_loss: 4761.4502 - val\_accuracy: 0.0000e+00

Epoch 8/50

90/100 [==========================>...] - ETA: 0s - loss: 1055.0333 - accuracy: 0.0000e+00

Epoch 00008: val\_loss did not improve from 3094.54386

100/100 [==============================] - 11s 111ms/sample - loss: 1020.4685 - accuracy: 0.0000e+00 - val\_loss: 6431.3985 - val\_accuracy: 0.0000e+00

Epoch 9/50

90/100 [==========================>...] - ETA: 0s - loss: 1024.4600 - accuracy: 0.0000e+00

Epoch 00009: val\_loss did not improve from 3094.54386

100/100 [==============================] - 10s 99ms/sample - loss: 1021.2726 - accuracy: 0.0000e+00 - val\_loss: 7457.3012 - val\_accuracy: 0.0000e+00

Epoch 10/50

90/100 [==========================>...] - ETA: 0s - loss: 834.6285 - accuracy: 0.0000e+00

Epoch 00010: val\_loss did not improve from 3094.54386

100/100 [==============================] - 9s 93ms/sample - loss: 883.2034 - accuracy: 0.0000e+00 - val\_loss: 7621.1423 - val\_accuracy: 0.0000e+00

Epoch 11/50

90/100 [==========================>...] - ETA: 0s - loss: 997.1629 - accuracy: 0.0000e+00

Epoch 00011: val\_loss did not improve from 3094.54386

100/100 [==============================] - 9s 90ms/sample - loss: 997.0316 - accuracy: 0.0000e+00 - val\_loss: 7171.8593 - val\_accuracy: 0.0000e+00

Результаты работы сети:

model\_lstm2.load\_weights('LSTM2.h5')

a = model\_lstm2.predict(x\_test)

res = model\_lstm2.evaluate(x\_test, y\_test, verbose=0)

print('val\_loss:',res[0],'\nval\_accuracy:',res[1])

val\_loss: 3094.54373046875

val\_accuracy: 0.0

Таким образом, получилось, что среднеквадратическая ошибка на тестовых данных равна примерно 3094, однако это суммарная ошибка, и так как у нас в тестовой выборке 100 значений, ошибка, приходящаяся на одно предсказание, составляет примерно ±31 рабочий цикл.

Построим график, иллюстрирующий зависимость значений функции потерь нейронной сети от количества пройденных эпох обучения:

plt.plot(history\_lstm2.history['loss'],

label='Среднеквадр. ошибка на обучающем наборе')

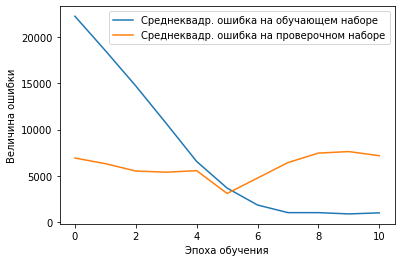
plt.plot(history\_lstm2.history['val\_loss'],

label='Среднеквадр. ошибка на проверочном наборе')

plt.xlabel('Эпоха обучения')

plt.ylabel('Доля верных ответов')

plt.legend()

plt.show()