Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе #7

«Логистическая регрессия в качестве нейронной сети»

Карп Александр Игоревич

магистрант кафедры информатики

группа №858641

Минск 2019

Исходный код:

<https://colab.research.google.com/drive/1u4FE41qu8KkowNLYwWQvEbHulsJQv3d4>

**Задание 1.**

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

Для создания словаря используем класс Tokenizer

Загрузим датасет и подготовим его для токенайзера:

def read\_file(file):

    f=open(file)

    try:

        text = f.read()

    except:

        text =  None

    finally:

        f.close()

    return text

def clean\_text(text):

    text = text.lower()

    text = re.sub(re.compile('<.\*?>'), '', text)

    text = text.translate(str.maketrans('', '', punctuation))

    return text

positive\_files = glob.glob('./aclImdb/train/pos/\*.txt')

positive\_reviews\_list = []

for file in positive\_files:

    text = read\_file(file)

    if text:

        positive\_reviews\_list.append(text)

negative\_files = glob.glob('./aclImdb/train/neg/\*.txt')

negative\_reviews\_list = []

for file in negative\_files:

    text = read\_file(file)

    if text:

        negative\_reviews\_list.append(clean\_text(text))

Обучим токенайзер на тренировочных данных

t = Tokenizer()

t.fit\_on\_texts(positive\_reviews\_list + negative\_reviews\_list)

**Задание 2.**

Реализуйте и обучите двунаправленную рекуррентную сеть (LSTM или GRU). Какого качества классификации удалось достичь?

Создадим простую архитектуру нейронной сети с LSTM слоем:

model = Sequential()

model.add(Embedding(20000, 128))

model.add(LSTM(128, dropout=0.2, recurrent\_dropout=0.2, kernel\_regularizer=keras.regularizers.l2(0.01)))

model.add(Dense(1, activation='sigmoid'))

Model: "sequential\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_2 (Embedding) (None, None, 128) 2560000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_2 (LSTM) (None, 128) 131584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 1) 129

=================================================================

Total params: 2,691,713

Trainable params: 2,691,713

Non-trainable params: 0

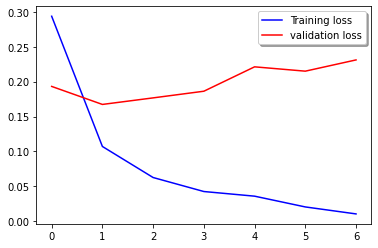


Рис 1. Зависимость функции потерь от эпохи обучения

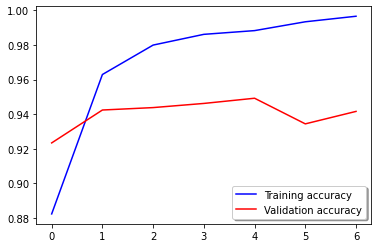


Рис 2. Зависимость точности от эпохи

Можно увидеть результаты переобучения данной модели, с ростом точности на тренировочных данных точность на валидационных меняется немонотонно

Посмотрим точность на тестовом датасете

25000/25000 [==============================] - 13s 513us/step

0.9348800182342529

Точность составила 93.5%

**Задание 3.**

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

Воспользуемся преобученным word embedding - glove

Для этого скачаем вектора и составим матрицу:

word\_index = t.word\_index

import numpy as np

embeddings\_index = {}

with open('glove.6B.100d.txt') as f:

    for line in f:

        word, coefs = line.split(maxsplit=1)

        coefs = np.fromstring(coefs, 'f', sep=' ')

        embeddings\_index[word] = coefs

num\_words = min(20000, len(word\_index) + 1)

embedding\_matrix = np.zeros((num\_words, 100))

for word, i in word\_index.items():

    if i >= 20000:

        continue

    embedding\_vector = embeddings\_index.get(word)

    if embedding\_vector is not None:

        # words not found in embedding index will be all-zeros.

        embedding\_matrix[i] = embedding\_vector

print('Found %s word vectors.' % len(embeddings\_index))

Теперь используем эту матрицу в качестве инициализации для Embedding слоя и обучим модель:

model = Sequential()

from keras.initializers import Constant

model.add(Embedding(20000,100,embeddings\_initializer=Constant(embedding\_matrix),input\_length=80,trainable=False))

model.add(LSTM(100, dropout=0.2, recurrent\_dropout=0.2,))

model.add(Dense(1, activation='sigmoid'))

# try using different optimizers and different optimizer configs

model.compile(loss='binary\_crossentropy',

              optimizer='adam',

              metrics=['accuracy'])

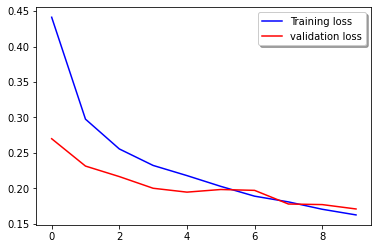


Рис 1. Зависимость функции потерь от эпохи обучения

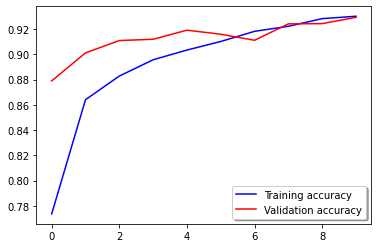


Рис 2. Зависимость точности от эпохи

Графики в данном случае ведут себя более монотонно, что говорят о меньшем результате переобучения – точность растет как на тренировочном, так и на валидационном сетах

Точность, полученная на тестовом сете составила 93%

**Задание 4.**

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

Создадим новую сеть, добавив больше реккурентных и полносвязных слоев

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_3 (Embedding) (None, None, 128) 2560000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_4 (LSTM) (None, None, 128) 131584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_5 (LSTM) (None, 64) 49408

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 128) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_5 (Dense) (None, 32) 4128

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_6 (Dense) (None, 1) 33

=================================================================

Total params: 2,753,473

Trainable params: 2,753,473

Non-trainable params: 0

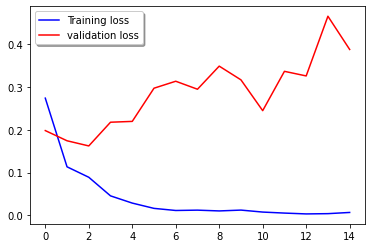


Рис 1. Зависимость функции потерь от эпохи обучения

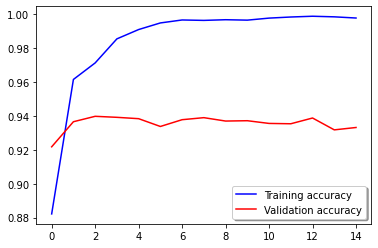


Рис 2. Зависимость точности от эпохи

На графиках видны результаты переобучения сети

Достигнутая точность:

25000/25000 [==============================] - 24s 973us/step

0.9314799904823303

Таким образом, расширением модели удалось добиться только увеличения времени обучения сети.