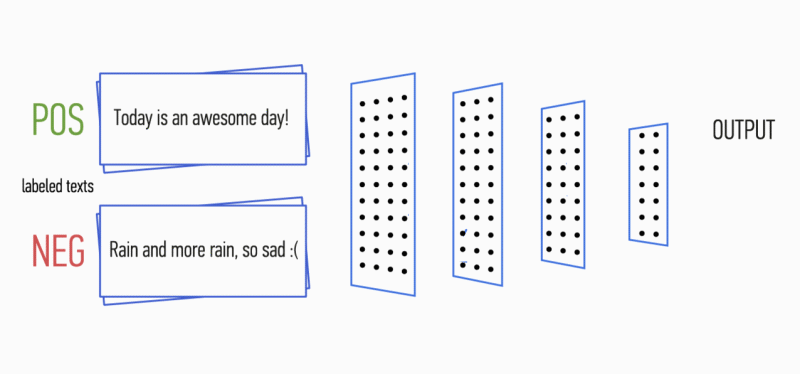
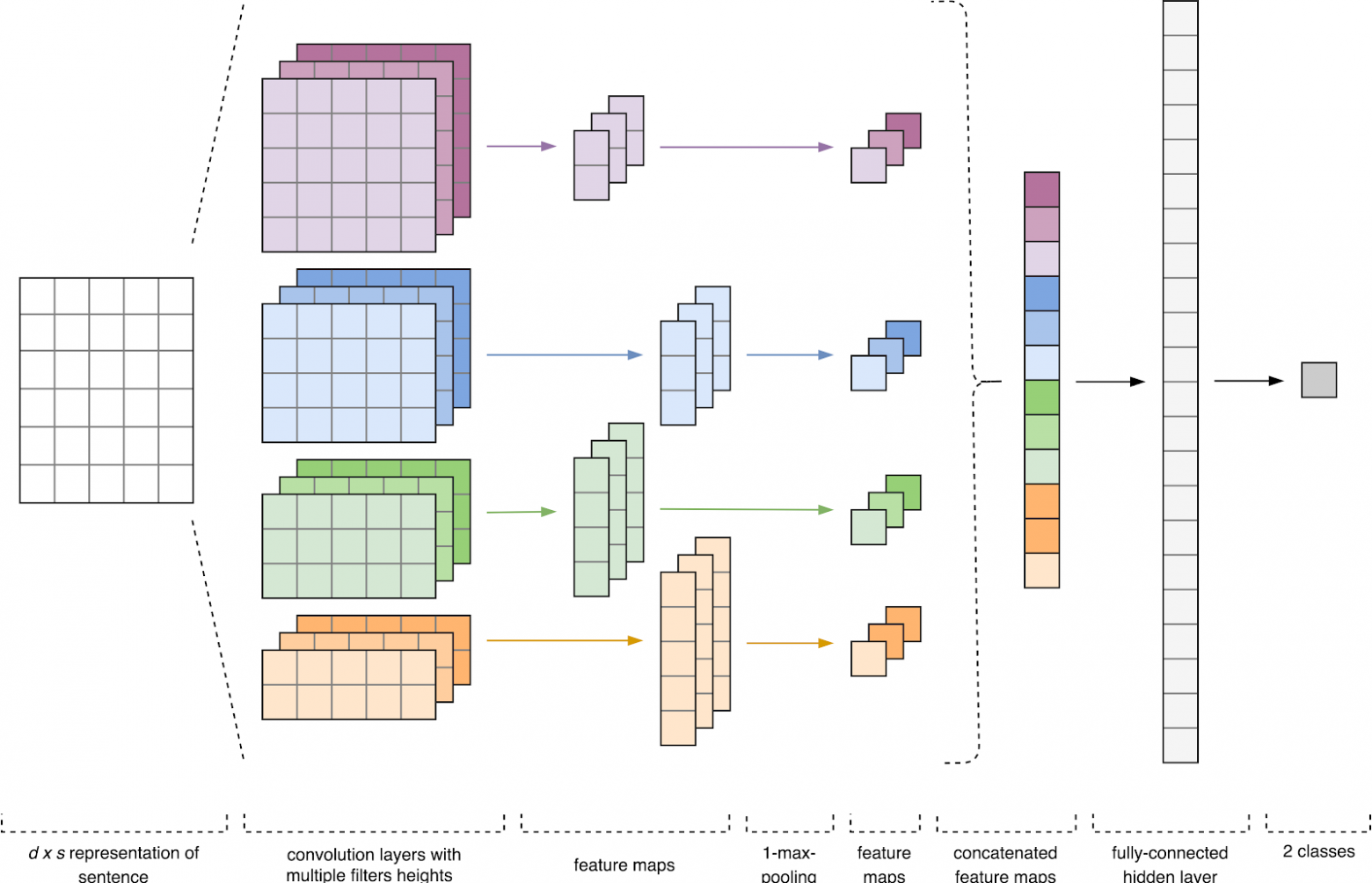
**Лекция 14**

**Анализ тональности текстов с помощью нейронных сетей**

  
Представьте, что у вас есть абзац текста. Можно ли понять, какую эмоцию несет этот текст: радость, грусть, гнев? Можно. Упростим себе задачу и будем классифицировать эмоцию как позитивную или как негативную, без уточнений. Есть много способов решать такую задачу, и один из них — **свёрточные нейронные сети** (Convolutional Neural Networks). CNN изначально были разработаны для обработки изображений, однако они успешно справляются с решением задач в сфере автоматической обработки текстов. Я познакомлю вас с бинарным анализом тональности русскоязычных текстов с помощью свёрточной нейронной сети, для которой векторные представления слов были сформированы на основе обученной **Word2Vec** модели.   
  
Статья носит обзорный характер, я сделал акцент на практическую составляющую. И сразу хочу предупредить, что принимаемые на каждом этапе решения могут быть неоптимальными. Перед прочтением рекомендую ознакомиться с [вводной статьей](https://habr.com/company/ods/blog/353060/) по использованию CNN в задачах обработки естественных языков, а также прочитать [материал](https://habr.com/company/ods/blog/329410/) про методы векторного представление слов.

**Архитектура**

Рассматриваемая архитектура CNN основана на подходах [1] и [2]. Подход [1], в котором используется ансамбль сверточных и рекуррентных сетей, на крупнейшем ежегодном соревновании по компьютерной лингвистике SemEval-2017 занял первые места [3] в пяти номинациях в задаче по анализу тональности.  
  
  
*Рисунок 1. Архитектура CNN [2].*  
  
Входными данными CNN (рис. 1) является матрица с фиксированной высотой *n*, где каждая строка представляет собой векторное отображение токена в признаковое пространство размерности *k*. Для формирования признакового пространства часто используют инструменты дистрибутивной семантики, такие как Word2Vec, Glove, FastText и т.д.  
  
На первом этапе входная матрица обрабатывается слоями свертки. Как правило, фильтры имеют фиксированную ширину, равную размерности признакового пространства, а для подбора размеров у фильтров настраивается только один параметр — высота *h*. Получается, что *h* — это высота смежных строк, рассматриваемых фильтром совместно. Соответственно, размерность выходной матрицы признаков для каждого фильтра варьируется в зависимости от высоты этого фильтра *h* и высоты исходной матрицы *n*.   
  
Далее карта признаков, полученная на выходе каждого фильтра, обрабатывается слоем субдискретизации с определенной функцией уплотнения (на изображении — 1-max pooling), т.е. уменьшает размерность сформированной карты признаков. Таким образом извлекается наиболее важная информация для каждой свертки независимо от её положения в тексте. Другими словами, для используемого векторного отображения комбинация слоев свёртки и слоев субдискретизации позволяет извлекать из текста наиболее значимые *n*-граммы.  
  
После этого карты признаков, рассчитанные на выходе каждого слоя субдискретизации, объединяются в один общий вектор признаков. Он подаётся на вход скрытому полносвязному слою, а потом поступает на выходной слой нейронной сети, где и рассчитываются итоговые метки классов.

**Данные для обучения**

Для обучения я выбрал корпус коротких текстов Юлии Рубцовой, сформированный на основе русскоязычных сообщений из Twitter [4]. Он содержит 114 991 положительных, 111 923 отрицательных твитов, а также базу неразмеченных твитов объемом 17 639 674 сообщений.

import pandas as pd

import numpy as np

# Считываем данные

n = ['id', 'date', 'name', 'text', 'typr', 'rep', 'rtw', 'faw', 'stcount', 'foll', 'frien', 'listcount']

data\_positive = pd.read\_csv('data/positive.csv', sep=';', error\_bad\_lines=False, names=n, usecols=['text'])

data\_negative = pd.read\_csv('data/negative.csv', sep=';', error\_bad\_lines=False, names=n, usecols=['text'])

# Формируем сбалансированный датасет

sample\_size = min(data\_positive.shape[0], data\_negative.shape[0])

raw\_data = np.concatenate((data\_positive['text'].values[:sample\_size],

data\_negative['text'].values[:sample\_size]), axis=0)

labels = [1] \* sample\_size + [0] \* sample\_size

Перед началом обучения тексты прошли процедуру предварительной обработки:

* приведение к нижнему регистру;
* замена «ё» на «е»;
* замена ссылок на токен «URL»;
* замена упоминания пользователя на токен «USER»;
* удаление знаков пунктуации.

import re

def preprocess\_text(text):

text = text.lower().replace("ё", "е")

text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL', text)

text = re.sub('@[^\s]+', 'USER', text)

text = re.sub('[^a-zA-Zа-яА-Я1-9]+', ' ', text)

text = re.sub(' +', ' ', text)

return text.strip()

data = [preprocess\_text(t) for t in raw\_data]

Далее я разбил набор данных на обучающую и тестовую выборку в соотношении 4:1.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=1)

**Векторное отображение слов**

Входными данными сверточной нейронной сети является матрица с фиксированной высотой *n*, где каждая строка представляет собой векторное отображение слова в признаковое пространство размерности *k*. Для формирования embedding-слоя нейронной сети я использовал утилиту дистрибутивной семантики Word2Vec [5], предназначенную для отображения семантического значения слов в векторное пространство. Word2Vec находит взаимосвязи между словами согласно предположению, что в похожих контекстах встречаются семантически близкие слова. Подробнее о Word2Vec можно прочитать в [оригинальной статье](https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf), а также [тут](https://habr.com/company/ods/blog/329410/) и [тут](https://habr.com/post/249215/). Поскольку твитам характерна авторская пунктуация и эмотиконы, определение границ предложений становится достаточно трудоемкой задачей. В этой работе я допустил, что каждый твит содержит лишь одно предложение.  
  
База неразмеченных твитов хранится в SQL-формате и содержит более 17,5 млн. записей. Для удобства работы я конвертировал её в SQLite с помощью [этого](https://github.com/dumblob/mysql2sqlite) скрипта.

import sqlite3

# Открываем SQLite базу данных

conn = sqlite3.connect('mysqlite3.db')

c = conn.cursor()

with open('data/tweets.txt', 'w', encoding='utf-8') as f:

# Считываем тексты твитов

for row in c.execute('SELECT ttext FROM sentiment'):

if row[0]:

tweet = preprocess(row[0])

# Записываем предобработанные твиты в файл

print(tweet, file=f)

Далее с помощью библиотеки Gensim обучил Word2Vec-модель со следующими параметрами:

* *size = 200* — размерность признакового пространства;
* *window = 5* — количество слов из контекста, которое анализирует алгоритм;
* *min\_count = 3* — слово должно встречаться минимум три раза, чтобы модель его учитывала.

import logging

import multiprocessing

import gensim

from gensim.models import Word2Vec

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

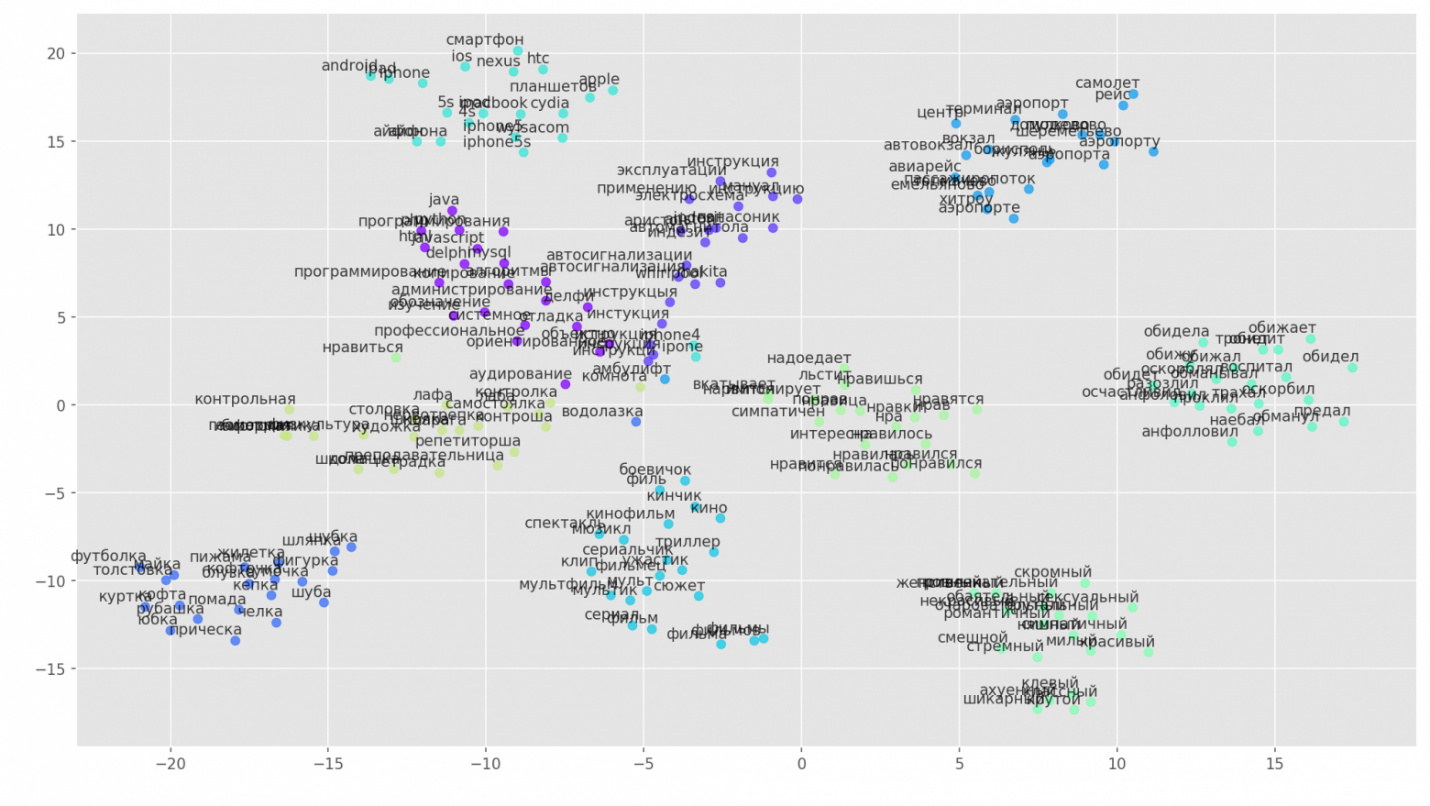
# Считываем файл с предобработанными твитами

data = gensim.models.word2vec.LineSentence('data/tweets.txt')

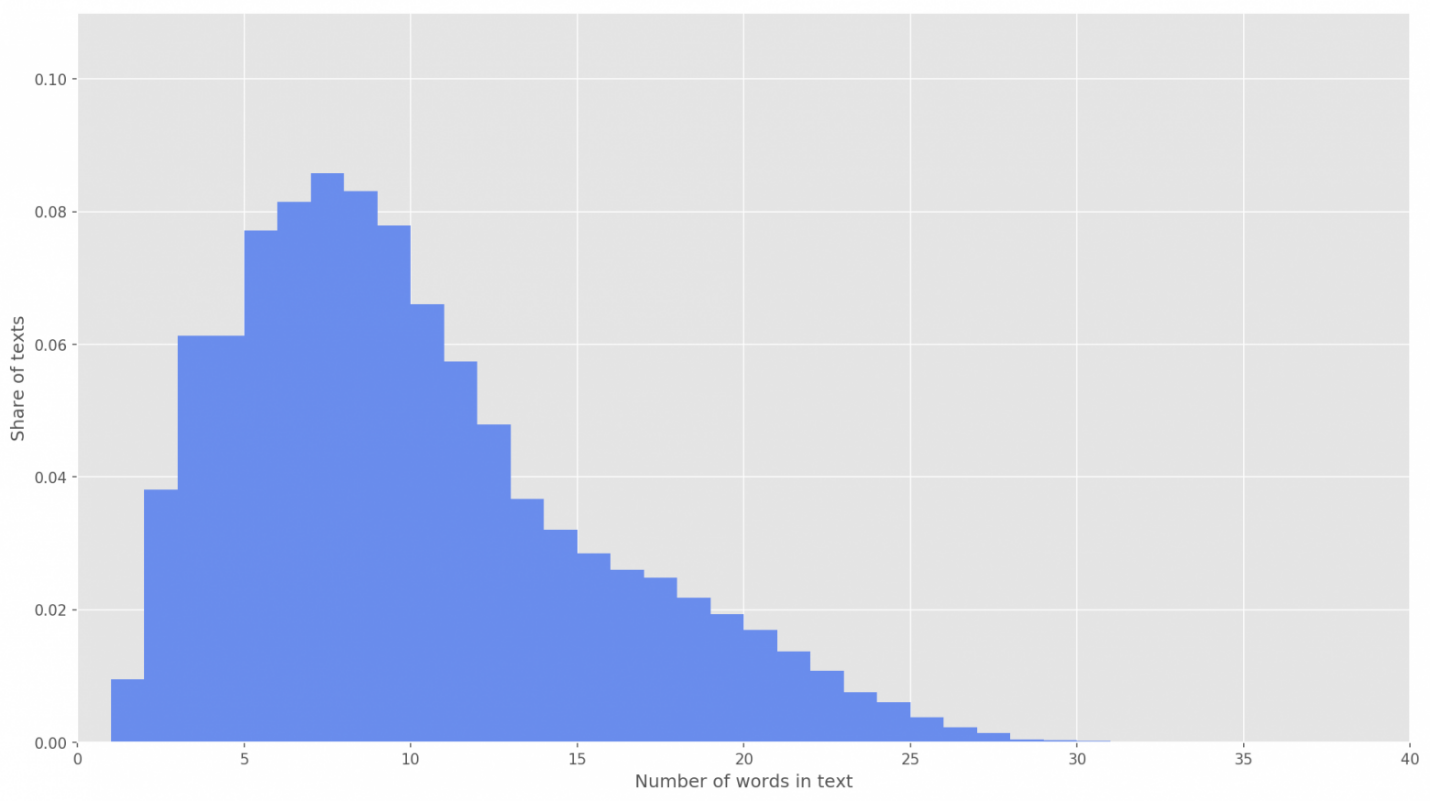
# Обучаем модель

model = Word2Vec(data, size=200, window=5, min\_count=3, workers=multiprocessing.cpu\_count())

model.save("models/w2v/model.w2v")

  
*Рисунок 2. Визуализация кластеров похожих слов с использование t-SNE.*   
  
Для более детального понимания работы Word2Vec на рис. 2 представлена визуализация нескольких кластеров похожих слов из обученной модели, отображенных в двухмерное пространство с помощью [алгоритма визуализации t-SNE](https://habr.com/post/267041/).

**Векторное отображение текстов**

  
*Рис 3. Распределение длины текстов.*  
  
На следующем этапе каждый текст был отображен в массив идентификаторов токенов. Я выбрал размерность вектора текста *s=26*, поскольку при данном значении полностью покрываются 99,71% всех текстов в сформированном корпусе (рис. 3). Если при анализе количество слов в твите превышало высоту матрицы, оставшиеся слова отбрасывались и не учитывались в классификации. Итоговая размерность матрицы предложения составила *s×d=26×200*.

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

# Высота матрицы (максимальное количество слов в твите)

SENTENCE\_LENGTH = 26

# Размер словаря

NUM = 100000

def get\_sequences(tokenizer, x):

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

return pad\_sequences(sequences, maxlen=SENTENCE\_LENGTH)

# Cоздаем и обучаем токенизатор

tokenizer = Tokenizer(num\_words=NUM)

tokenizer.fit\_on\_texts(x\_train)

# Отображаем каждый текст в массив идентификаторов токенов

x\_train\_seq = get\_sequences(tokenizer, x\_train)

x\_test\_seq = get\_sequences(tokenizer, x\_test)

**Свёрточная нейронная сеть**

Для построения нейронной сети я использовал библиотеку Keras, которая выступает высокоуровневой надстройкой над TensorFlow, CNTK и Theano. У Keras есть отличная документация, а также блог, который освещает многие задачи машинного обучения, к примеру, [инициализацию embedding-слоя](https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html). В нашем случае embedding-слой был инициирован весами, полученными при обучении Word2Vec. Чтобы минимизировать изменения в embedding-слое, я заморозил его на первом этапе обучения.

from keras.layers import Input

from keras.layers.embeddings import Embedding

tweet\_input = Input(shape=(SENTENCE\_LENGTH,), dtype='int32')

tweet\_encoder = Embedding(NUM, DIM, input\_length=SENTENCE\_LENGTH,

weights=[embedding\_matrix], trainable=False)(tweet\_input)

В разработанной архитектуре использованы фильтры с высотой *h=(2, 3, 4, 5)*, которые предназначены для параллельной обработки биграмм, триграмм, 4-грамм и 5-грамм соответственно. Добавил в нейронную сеть по 10 свёрточных слоев для каждой высоты фильтра, функция активации — ReLU. С рекомендациями по поиску оптимальной высоты и количества фильтров можно ознакомиться в работе [2].   
  
После обработки слоями свертки, карты признаков поступали на слои субдискретизации, где к ним применялась операция 1-max-pooling, тем самым извлекая наиболее значимые n-граммы из текста. На следующем этапе происходило объединение в общий вектор признаков (слой объединения), который подавался в скрытый полносвязный слой с 30 нейронами. На последнем этапе итоговая карта признаков подавалась на выходной слой нейронной сети с сигмоидальной функцией активации.  
  
Поскольку нейронные сети склонны к переобучению, после embedding-слоя и перед скрытым полносвязным слоем я добавил dropout-регуляризацию c вероятностью выброса вершины p=0.2.

from keras import optimizers

from keras.layers import Dense, concatenate, Activation, Dropout

from keras.models import Model

from keras.layers.convolutional import Conv1D

from keras.layers.pooling import GlobalMaxPooling1D

branches = []

# Добавляем dropout-регуляризацию

x = Dropout(0.2)(tweet\_encoder)

for size, filters\_count in [(2, 10), (3, 10), (4, 10), (5, 10)]:

for i in range(filters\_count):

# Добавляем слой свертки

branch = Conv1D(filters=1, kernel\_size=size, padding='valid', activation='relu')(x)

# Добавляем слой субдискретизации

branch = GlobalMaxPooling1D()(branch)

branches.append(branch)

# Конкатенируем карты признаков

x = concatenate(branches, axis=1)

# Добавляем dropout-регуляризацию

x = Dropout(0.2)(x)

x = Dense(30, activation='relu')(x)

x = Dense(1)(x)

output = Activation('sigmoid')(x)

model = Model(inputs=[tweet\_input], outputs=[output])

Итоговую модель сконфигурировал с функцией оптимизации Adam (Adaptive Moment Estimation) и бинарной кросс-энтропией в качестве функции ошибок. Качество работы классификатора оценивал в критериях макро-усредненных точности, полноты и f-меры.

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=[precision, recall, f1])

model.summary()

На первом этапе обучения заморозил embedding-слой, все остальные слои обучались в течение 10 эпох:

* Размер группы примеров, используемых для обучения: 32.
* Размер валидационной выборки: 25%.

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint("models/cnn/cnn-frozen-embeddings-{epoch:02d}-{val\_f1:.2f}.hdf5", monitor='val\_f1', save\_best\_only=True, mode='max', period=1)

history = model.fit(x\_train\_seq, y\_train, batch\_size=32, epochs=10, validation\_split=0.25, callbacks = [checkpoint])

**Логи**

Затем выбрал модель с наивысшими показателями F-меры на валидационном наборе данных, т.е. модель, полученную на восьмой эпохе обучения (F1=0.7791). У модели разморозил embedding-слой, после чего запустил еще пять эпох обучения.

from keras import optimizers

# Загружаем веса модели

model.load\_weights('models/cnn/cnn-frozen-embeddings-09-0.77.hdf5')

# Делаем embedding слой способным к обучению

model.layers[1].trainable = True

# Уменьшаем learning rate

adam = optimizers.Adam(lr=0.0001)

model.compile(loss='binary\_crossentropy', optimizer=adam, metrics=[precision, recall, f1])

model.summary()

checkpoint = ModelCheckpoint("models/cnn/cnn-trainable-{epoch:02d}-{val\_f1:.2f}.hdf5", monitor='val\_f1', save\_best\_only=True, mode='max', period=1)

history\_trainable = model.fit(x\_train\_seq, y\_train, batch\_size=32, epochs=5, validation\_split=0.25, callbacks = [checkpoint])

**Логи**

Наивысший показатель *F1=76.80%* на валидационной выборке был достигнут на третьей эпохе обучения. Качество работы обученной модели на тестовых данных составило *F1=78.1%*.   
  
Таблица 1. Качество анализа тональности на тестовых данных.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метка класса | Точность | Полнота | F1 | Количество объектов |
| Negative | 0.78194 | 0.78243 | 0.78218 | 22457 |
| Positive | 0.78089 | 0.78040 | 0.78064 | 22313 |
| avg / total | 0.78142 | 0.78142 | 0.78142 | 44770 |

**Результат**

В качестве baseline-решения я [обучил](https://towardsdatascience.com/sentiment-analysis-of-tweets-using-multinomial-naive-bayes-1009ed24276b) наивный байесовский классификатор с мультиномиальной моделью распределения, результаты сравнения представлены в табл. 2.  
  
Таблица 2. Сравнение качества анализа тональности.

|  |  |  |  |
| --- | --- | --- | --- |
| Классификатор | Precision | Recall | F1 |
| MNB | 0.7577 | 0.7564 | 0.7560 |
| CNN | **0.78142** | **0.78142** | **0.78142** |

Как видите, качество классификации CNN превысило MNB на несколько процентов. Значения метрик можно увеличить еще больше, если поработать над оптимизацией гиперпараметров и архитектуры сети. К примеру, можно изменить количество эпох обучения, проверить эффективность использования различных векторных представлений слов и их комбинаций, подобрать количество фильтров и их высоту, реализовать более эффективную предобработку текстов (исправление опечаток, нормализация, стемминг), настроить количество скрытых полносвязных слоев и нейронов в них.