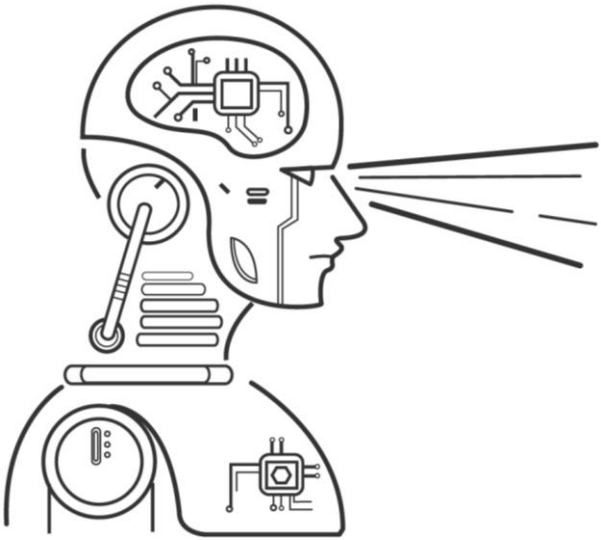
7915

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Ф. УТКИНА

МАШИННОЕ ОБУЧЕНИЕ

Методические указания
к практическим занятиям



Рязань 2024

УДК 004.8

Машинное обучение: методические указания к практическим занятиям/ Рязан. гос. радиотехн. ун-т; сост. И.С. Панина. – Рязань, 2024. – 48 с.

Содержат указания по выполнению практических работ для студентов, обучающихся по направлениям 02.03.03 «Математическое обеспечение и администрирование информационных систем» и 38.03.05 «Бизнес-информатика» уровня бакалавриата.

Предназначены для бакалавров дневного и заочного отделений.

Ил. 8

В подготовке и апробации материалов методических указаний принимали участие студентка группы 0714 Попова А.А. и студенты группы 040.

Алгоритмы машинного обучения, нейронные сети, сверточные сети, Python

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра электронных вычислительных машин Рязанского государственного радиотехнического университета (зав. кафедрой Б.В. Костров)

Машинное обучение

Составитель Панина Ирина Сергеевна

Редактор М.Е. Цветкова

Корректор С.В. Макушина

Подписано в печать 15.02.24. Формат бумаги 60х84 1/16.

Бумага писчая. Печать трафаретная. Усл. печ. л. 3,0.

Тираж 20 экз. Заказ

Рязанский государственный радиотехнический университет.

390005, Рязань, ул. Гагарина, 59/1.

Редакционно-издательский центр РГРТУ.

Практическое занятие № 1

Проектирование простейших нейронных сетей

Цель работы: изучить простейшие модели нейронных сетей.

Теоретическая часть

Искусственный интеллект — это область науки и инжиниринга, занимающаяся созданием машин и компьютерных программ, обладающих интеллектом. Она связана с задачей использования компьютеров для понимания человеческого интеллекта. При этом искусственный интеллект не должен ограничиваться только биологически наблюдаемыми методами.

Машинное обучение — обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. В машинном обучении используются различные алгоритмы анализа имеющейся информации для обучения аналитических платформ выявлению в данных всевозможных закономерностей, а в дальнейшем - самостоятельному принятию решений. Обучение происходит на множестве схожих примеров с учителем – когда данные заранее размечены, и без учителя - когда система пытается разметить данные самостоятельно.

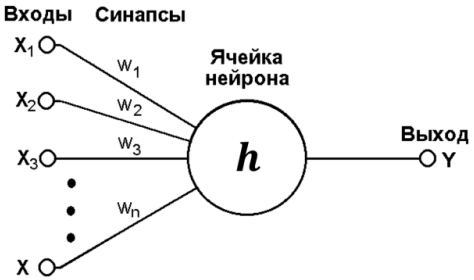
Нейросети — один из видов машинного обучения.

Искусственная нейронная сеть (ИНС) — упрощенная модель биологической нейронной сети, представляющая собой совокупность искусственных нейронов, взаимодействующих друг с другом.

Нейрон — это узел искусственной нейронной сети, являющийся упрощенной моделью естественного нейрона. По сути, представляет собой вычислительную единицу, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Математически искусственный нейрон обычно представляют как некоторую нелинейную функцию от единственного аргумента — линейной комбинации всех входных сигналов. Они делятся на **три основных типа**: входной, скрытый и выходной**. Связи между двумя нейронами** называются синапсами. У синапсов есть один параметр – вес. Благодаря ему в процессе передачи от одного нейрона к другому данные способны изменяться.

Любая нейросеть — это набор нейронов и связей между ними. Нейрон лучше всего представлять как функцию с множеством входов и одним выходом (рис. 1). Задача нейрона — взять числа со своих входов, выполнить над ними функцию и отдать результат на выход. Простой пример полезного нейрона: просуммировать все цифры со входов, и, если их сумма больше N, — выдать на выход единицу, иначе — ноль.

Входной информацией для каждого из нейронов скрытого слоя является сумма всех входных данных, умноженных на соответствующие им веса. Поскольку данные на входе могут сильно отличаться друг от друга по абсолютной величине, необходимо провести нормализацию данных с помощью **функции активации.** Функций активации достаточно много, поэтому мы рассмотрим самые основные: линейная, сигмоида и гиперболический тангенс. Главные их отличия — это диапазон значений.



$$h = \sum_i x_i \cdot w_i, y = f(h)$$

Рис. 1. Формальный нейрон

Самым базовым типом нейронных сетей является **сеть прямого распространения**, то есть такая сеть, в которой поток информации движется только в одном направлении.

Нейронные сети используются для решения большого спектра задач, общим для которых является необходимость в аналитических вычислениях, на которые способен человеческий мозг. Самыми распространенными областями применения являются: классификация, предсказание, распознавание.

Последовательность данных, которыми оперирует нейронная сеть, называется **тренировочным набором.** Общее количество тренировочных наборов, которые прошла нейронная сеть, называется количеством итераций. Проход нейронной сетью всех тренировочных наборов называется **эпохой.** Количество этих эпох при инициализации нейросети устанавливается в 0 и имеет верхний потолок, задаваемый вручную.

Процентная величина, показывающая расхождение между ожидаемым и полученным результатом, называется **ошибкой.** Ошибка формируется каждую эпоху и должна идти на спад.

В остальном нейронные сети делятся:

– на однослойные;

– многослойные;

– сети прямого распространения;

– сети с обратными связями.

Многослойная нейронная сеть — нейронная сеть, состоящая из входного, выходного и расположенного(ых) между ними одного (нескольких) скрытых слоев нейронов (рис. 2).

Помимо входного и выходного слоев, эти нейронные сети содержат промежуточные, скрытые слои. Такие сети обладают гораздо большими возможностями, чем однослойные нейронные сети, однако методы обучения нейронов скрытого слоя были разработаны относительно недавно.

Входной (распределительный) псевдослой	Скрытые (промежуточные) слои	Выходной слой

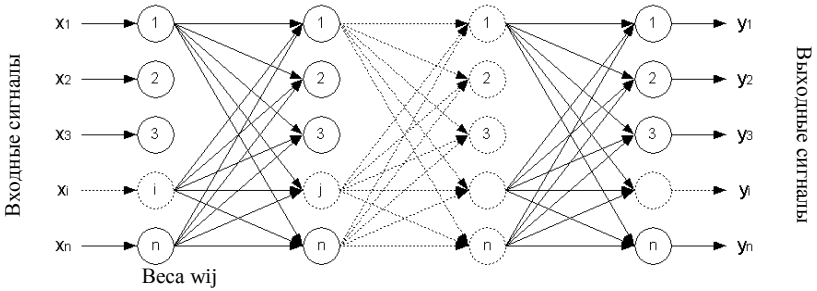


Рис. 2. Многослойная нейронная сеть

Сеть, состоящая из нескольких слоёв, между которыми связаны все нейроны, называется перцептроном (MLP) и считается самой простой архитектурой.

Практическая часть

Для начала необходимо импортировать библиотеку линейной алгебры numpy:

```
import numpy as np
```

Далее следует описать функцию активации:

```
def nonlin(x, deriv=False):
    if (deriv == True):
     return x*(1-x)
     return 1 / (1 + np.exp(-x))
```

Эта функция способна выдавать как значение самой сигмоиды при deriv == False, так и значение ее производной, соответствующее параметру x.

Далее следует инициализация массива входных данных. Каждая его строка является тренировочным набором.

```
X = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
```

В следующей строчке кода присутствует функция «.T», которая означает транспонирование строки. Таким образом, у нейронной сети получается три входа и один выход.

```
y = np.array([[0, 0, 1, 1]]).T
```

После необходимо инициализировать матрицу весов. Так как строить нейросеть менее чем в три слоя не имеет смысла, понадобятся две матрицы.

```
syn0 = 2*np.random.random((3,4)) -1
syn1 = 2*np.random.random((4,1)) - 1
```

Когда начальные данные инициализированы, можно начинать процесс обучения сети.

```
for iter in range(10000):
```

В теле цикла располагается следующий фрагмент кода:

```
# Прямое распространение
l0 = X
```

```
l1 = nonlin(np.dot(l0, syn0))
l2 = nonlin(np.dot(l1, syn1))
# Вычисление ошибки и изменения весов
l2_error = y - l2
l2_delta = l2_error * nonlin(l2, True)
l1_error = l2_delta.dot(syn1.T)
l1_delta = l1_error * nonlin(l1, deriv=True)
# Обновление весов
syn1 += l1.T.dot(l2_delta)
syn0 += l0.T.dot(l1_delta)
```

После этого можно вывести полученные данные:

```
print(l2)
```

Производя небольшие изменения в коде программы, можно добиться того, чтобы она выдавала результат любой базовой логической операции.

Задание. Построить простейшую нейронную сеть, вычисляющую результат логической операции, в соответствии с вариантом.

Вариант задания	Вычисляемая операция
1	Логическое И
2	Стрелка Пирса
3	Логическое Следование
4	Логические ИЛИ
5	Сложение по модулю 2
6	Логическая Равнозначность
7	Логические ИЛИ
8	Штрих Шеффера

Содержание отчета. В отчете представить код программы и результат ее работы.

Контрольные вопросы

1. Что такое функция активации?
2. Дайте определение понятиям искусственного нейрона, искусственной нейронной сети.
3. Поясните структурную схему модели искусственного нейрона.
4. Что такое алгоритм обучения нейронной сети?
5. Что называют эпохой обучения?
6. Как работают сети прямого распространения?
7. Из каких слоев состоит ИСН?
8. Поясните механизм обучения с учителем, обучения без учителя.

9. Для чего используется обучающая выборка? Из каких множеств она состоит?

Практическое занятие № 2
Проектирование нейронных сетей с использованием библиотеки Keras

Цели работы: изучить простейшие модели активационных функций, научиться прогнозировать результат на основе анализа зависимости величин.

Теоретическая часть

Алгоритм обратного распространения ошибки — один из наиболее популярных способов обучения нейронных сетей прямого распространения. Относится к методам обучения с учителем, поэтому требуется, чтобы в обучающих примерах были заданы целевые значения. В основе идеи алгоритма лежит использование выходной ошибки нейронной сети.

Алгоритм является итеративным, когда веса нейронов сети корректируются после подачи на ее вход одного обучающего примера, т. е. в режиме on-line.

На каждой итерации происходит два прохода сети — прямой и обратный. На прямом входной вектор распространяется от входов сети к ее выходам и формирует некоторый выходной вектор, соответствующий текущему состоянию весов. Затем вычисляется ошибка нейронной сети как разность между фактическим и целевым значениями. На обратном проходе эта ошибка распространяется от выхода сети к ее входам, и производится коррекция весов нейронов. Данная коррекция прямо пропорциональна коэффициенту η, который позволяет дополнительно управлять величиной шага коррекции и подбирается экспериментально в процессе обучения. Метод обратного распространения ошибки использует градиентный спуск, поэтому активационная функция должна быть дифференцируема на всей числовой оси и не иметь экстремумов. По этой причине для применения настоящего метода используют сигмоидную активационную функцию, кусочно-линейные функции или гиперболический тангенс (рис. 3).

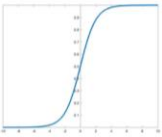
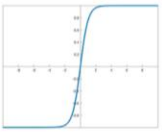
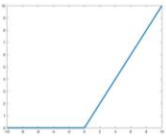
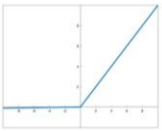
Sigmoid	$S(x) = \frac{1}{1+e^{-x}}$	
Tanh	$\tanh x = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
ReLU	$RELU(x) = \begin{cases} 0 \; if \; x<0 \\ x \; if \; x>=0 \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$	

Рис. 3. Функции активации

К преимуществам данного метода относятся простота реализации и устойчивость к аномалиям и выбросам данных. Основными недостатками являются неопределенность длительности обучения и уязвимость алгоритма к попаданию в локальные минимумы функции ошибки.

Практическая часть

Для начала следует подключить к проекту необходимые для выполнения библиотеки. К ним относятся numpy, который уже использовался ранее, а также составные части библиотеки Keras.layers, такие как Dense, облегчающая создание слоев нейронной сети, Activation, содержащая множество функций активации и Sequential, включающая в себя модель нейронной сети прямого распространения.

```
import os.path
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
```

Далее следует импортировать данные из файла следующим образом:

```
from array import array
```

```
data = np.loadtxt('E:\test.txt', delimiter = ';', dtype=np.float64)
X = data[:,[0,1,2,3]]
Y = data[:,[4]]
```

Здесь **E:\test.txt -** путь до файла с данными. Данные необходимо подготовить самостоятельно.

После успешного импорта данных необходимо указать, какая модель нейросети используется, сколько нейронов в каждом слое, и выбрать активационные функции для каждого из скрытых слоев:

```
model = Sequential()
model.add(Dense(units = 8, input_shape = (4,)))
model.add(Activation("sigmoid"))
model.add(Dense(units = 24))
model.add(Activation("sigmoid"))
model.add(Dense(units = 12))
model.add(Activation("sigmoid"))
model.add(Dense(1))
model.add(Activation("sigmoid"))
```

Библиотека Keras позволяет автоматизировать калибровку функций потерь относительно модели нейросети с помощью различных **оптимизаторов**, которые и обновляют модель в ответ на данные функции потерь. По этой причине нужно не забыть указать, какой оптимизатор следует использовать, а также указать количество эпох обучения:

```
model.compile(optimizer = 'RMSprop', loss = 'binary_crossentropy', metrics = ['accuracy'])
model.fit(X,Y, nb_epoch = 500, batch_size = 100)
```

На этом этапе нейросеть готова к обучению, но необходимо указать данные, не входящие в обучающий набор, которые она будет анализировать и вывести результат:

```
model.fit(X,Y, nb_epoch = 500, batch_size = 100)
test = np.asarray([[6,3,5,2],[5,3,1,0],[6,3,4,2]])
res = model.predict(test)
print(res)
```

Задание. Построить простейшую нейронную сеть, предназначенную для определения вида цветка ириса по его размерам в соответствии с вариантом.

Вариант задания	Параметр 1	Параметр 2	Параметр 3
1	Длина чашелистика	Ширина чашелистика	Длина лепестка
2	Длина	Ширина	Длина лепестка

	чашелистика	чашелистика	
3	Длина чашелистика	Длина лепестка	Ширина лепестка
4	Длина чашелистика	Длина лепестка	Ширина лепестка
5	Ширина чашелистика	Длина чашелистика	Длина лепестка
6	Ширина чашелистика	Длина чашелистика	Длина лепестка
7	Ширина чашелистика	Длина лепестка	Ширина лепестка
8	Ширина чашелистика	Длина лепестка	Ширина лепестка

Контрольные вопросы

1. Что такое функция активации?
2. Какие функции активации вы знаете? Для чего используются разные функции активации?
3. Оптимизаторы, их применение и виды.
4. Применение библиотеки Keras для машинного обучения.
5. Для чего используют функцию потерь?

Практическое занятие № 3
Линейная регрессия

Цель работы: изучить модель линейной регрессии.

Теоретическая часть

Одной из самых распространенных и простых в понимании моделей нейронных сетей является линейная регрессия. Линейная регрессия (англ. Linear regression) — используемая в статистике регрессионная модель зависимости одной (объясняемой зависимой) переменной *y* от другой или нескольких других переменных (факторов, регрессоров, независимых переменных) *x* с линейной функцией зависимости. Эта модель принадлежит к категории «обучение с учителем». С ее помощью можно прогнозировать значение *y* при новых значениях *x*.

Формула линейной регрессии похожа на обычную линейную зависимость $y = kx+b$, но с учетом случайной ошибки e. Таким образом, формула, представляющая простейшую линейную регрессию, будет выглядеть так: $y = kx + b + e$. В случае же зависимости y от нескольких x формула примет вид:

$$y = a_0 + \sum_{i=1}^{n} a_i x_i + e ,$$

где a_i – параметры модели;

y, x_i – исходные данные;

e – случайное значение ошибки.

Такая модель называется **множественной регрессией**.

Обучение модели зачастую осуществляется с помощью метода наименьших квадратов (МНК). **Метод наименьших квадратов (МНК)** — математический метод, применяемый для решения различных задач, основанный на минимизации суммы квадратов отклонений некоторых функций от искомых переменных.

Зачастую построение модели затрудняется шумами – элементами, резко отличающимися от остальных в данной выборке, что довольно часто бывает на реальных данных.

Графическое представление простейшей линейной регрессии может выглядеть так, как представлено на рис. 4.

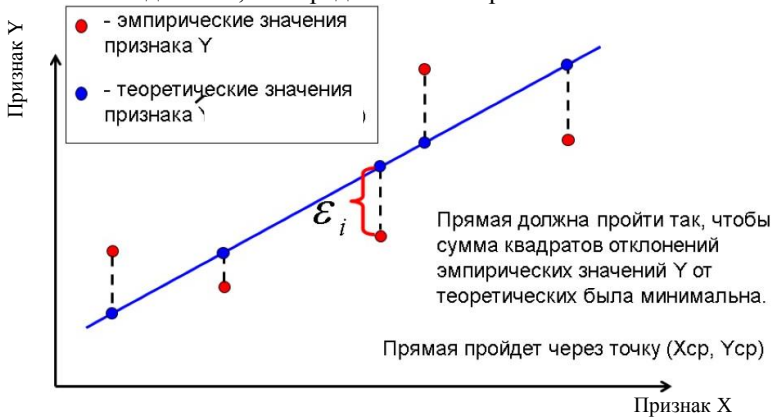


Рис. 4. Графическое представление линейной регрессии

Довольно часто вместо линейной регрессии применяется нелинейная регрессионная модель, способная во многих случаях более точно описать зависимости исходных данных.

Практическая часть

В практической части мы разберем создание модели линейной регрессии с помощью инструментов Scikit-learn на языке Python и с использованием средств графического интерфейса.

Произведем импорт требуемых библиотек:

```
from tkinter import *
import matplotlib.pyplot as plt #библиотека для визуализации результатов построения нашей модели
```

```
from sklearn import datasets, linear_model #данные импорты помогут собрать набор данных и построить линейную регрессию
```

Опишем окно для ввода исходных значений:

```
window = Tk()
window.title('Построение модели линейной регрессии')
 window.geometry('400x400')
frame = Frame(
    window,
    padx=15,
    pady=15)
frame.pack(expand=True)
lb1 = Label(
    frame, text='Введите количество точек:')
lb1.grid(row=3, column=1)
tf1 = Entry(frame)
tf1.grid(row=3, column=2)
k_samples = tf1.get()
lb2 = Label(frame, text='Введите разброс точек:')
lb2.grid(row=4, column=1)
tf2 = Entry( frame)
tf2.grid(row=4, column=2)
n_features=1
```

Создаем набор данных с помощью предусмотренных в библиотеке scikit инструментов и разбиваем наши данные на обучающий и тестовый наборы случайным образом. Создаем и обучаем модель линейной регрессии:

```
#функция построения модели
def calc():
k_noise = int(tf2.get())
k_samples = int(tf1.get())
X, y = datasets.make_regression(n_samples=k_samples, n_features=1, noise=k_noise)
#  обучающие и тестовые данные
train_size = int(k_samples * 0.8)
test_size = int(k_samples * 0.2)
# нарезаем входные данные на обучающие и тестовые
X_train = X[:-train_size]
X_test = X[-test_size:] # указываем, сколько будет выходных обучающих и тестовых данных
y_train = y[:-train_size]
y_test = y[-test_size:]
```

```
# Создали регрессионную модель
regr = linear_model.LinearRegression()
# обучение модели
regr.fit(X_train, y_train)
# получение значений y
y_pred = regr.predict(X_test)
# вывод картинки
plt.scatter(X_test, y_test, color='purple', s=10)
plt.plot(X_test, y_pred, color='black', linewidth=2)
plt.title('Модель линейной регрессии')
plt.xticks(())
plt.yticks(())
plt.savefig('model.png')
plt.show()
btn = Button(
   frame,
   text = "Построить модель",
   command = calc)
btn.grid(row=5, column=2)
window.mainloop()
```

Задание. Выполнить пример, представленный в работе. Изменить значения аргументов функции datasets.make_regression, сделать выводы по выполненной работе и подготовить отчет.

Контрольные вопросы

1. Какова основная задача регрессионного анализа?
2. С какой целью используется МНК? Чем характеризуется этот метод?
3. Что показывает коэффициент детерминации R^2? Какое значение R^2 лучше: более низкое или более высокое?
4. Как интерпретируется стандартная ошибка оценки для линии регрессии? Какое значение стандартной ошибки лучше: более низкое или более высокое?
5. Каким образом осуществляется прогнозирование в построенной линейной модели регрессии?
6. Как выглядит математическая модель множественной регрессии?

Практическое занятие № 4
Кластерный анализ данных

Цели работы: изучить простейший метод кластерного анализа данных.

Теоретическая часть

Кластеризация — это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться наиболее похожие между собой объекты, а объекты разных групп должны быть как можно более различны. Главное отличие кластеризации от классификации состоит в том, как четко заданы свойства каждой группы.

Кластерный анализ в общем случае сводится к следующим нескольким этапам:

- отбор выборки объектов;
- определение множества переменных, по которым будут оцениваться объекты в данной выборке;
- вычисление значения меры сходства между объектами;
- применение кластеризации для создания групп объектов;
- представление результатов анализа.

В данной лабораторной работе будет рассматриваться метод k-средних, подразумевающий наличие массива данных, которые необходимо сгруппировать в k кластеров.

Входными данными в методе k-средних является только матрица X. Как правило, мы формируем её так, чтобы каждая строка представляла отдельный пример, а каждый столбец – отдельный признак. Обычно мы говорим, что есть N примеров и D признаков, так что X представляет собой матрицу размерности NxD.

В алгоритме метода k-средних есть два основных этапа. Сначала выбирается k разных центров кластеров – как правило, это просто случайные точки в наборе данных. Затем следует основной этап, также разделенный на два. Первый – это выбор, к какому из кластеров принадлежит каждая точка из X. Для этого выбираются пример и кластер, чей центр ближе всего. Второй этап – повторное вычисление центров кластеров на основе множества точек, которые к ним приписаны. Для этого берутся все соответствующие примеры и вычисляется их среднее значение. Всё это делается до тех пор, пока алгоритм не сойдётся, то есть пока не прекратится изменение в распределении точек по кластерам или в координатах центров кластеров.

Практическая часть

Начать необходимо с импорта библиотек Numpy и Matplotlib, а также с написания функции **main()**. В этом примере данные для кластеризации будут сгенерированы сами в виде распределенных по Гауссу облаков. Переменные **mu** отвечают за координаты, где будут находиться данные облака, содержащие по 300 объектов. В итоге будет

создано 900 образцов, содержащихся в матрице X, размерностью NxD. После этого можно вывести всё на экран.

```
import numpy as np
import matplotlib.pyplot as plt
def main():
    D = 2
    s = 4
    mu1 = np.array([0, 0])
    mu2 = np.array([s, s])
    mu3 = np.array([0, s])
    N = 900
    X = np.zeros((N, D))
    X[:300, :] = np.random.randn(300, D) + mu1
    X[300:600, :] = np.random.randn(300, D) + mu2
    X[600:, :] = np.random.randn(300, D) + mu3
    plt.scatter(X[:,0], X[:,1])
    plt.show()
```

Далее используется функция plot_k_means, которая ещё не написана:

```
K = 5
plot_k_means(X, K)
```

Теперь необходимо перед функцией main() прописать функцию, вычисляющую расстояние между векторами:

```
def d(u, v):
    diff = u – v
    return diff.dot(diff)
```

После неё следует описать уже использованную функцию plot_k_means. Прежде всего, необходимо определить переменные и форму матрицы X, размерности NxD. Далее случайным образом инициализируются центры кластеров, являющиеся точками X, и происходит перерасчёт средних значений:

```
def plot_k_means(X, K, max_iter=20, beta=1.0):
    N, D = X.shape
    M = np.zeros((K, D))
    R = np.zeros((N, K))
    for k in range(K):
        M[k] = X[np.random.choice(N)]
    costs = []
    for i in range(max_iter):
        for k in range(K):
            for n in range(N):
```

```
                R[n,k] = np.exp(-beta*d(M[k], X[n])) / np.sum( np.exp(-beta*d(M[j], X[n])) for j in range(K) )
            M[k] = R[:,k].dot(X) / R[:,k].sum()
        c = cost(X, R, M)
        costs.append(c)
        if i > 0:
            if np.abs(costs[-1] - costs[-2]) < 1e-5:
                break
```

Дополнительным этапом рассчитывается и выводится функция затрат при данных R и M. Для этого допишем в текущую функцию следующий фрагмент кода:

```
def cost(X, R, M):
    cost = 0
    for k in range(len(M)):
        diff = X - M[k]
        sq_distances = (diff * diff).sum(axis=1)
        cost += (R[:,k] * sq_distances).sum()
    return cost
```

Осталось лишь добавить в функцию plot_k_means фрагмент кода для вывода, выводящий уже раскрашенные кластеры:

```
random_colors = np.random.random((K, 3))
colors = R.dot(random_colors)
plt.scatter(X[:, 0], X[:, 1], c=colors)
plt.show()
return M, R
```

Задание. Построить нейронную сеть, производящую кластеризацию методом k-means в соответствии с вариантом, создав по 600 объектов на облако.

Вариант задания	Количество кластеров	Количество облаков
1	3	5
2	4	4
3	6	2
4	3	5
5	4	4
6	5	3
7	3	5
8	4	3

Контрольные вопросы

1. Какие признаки вносят наибольший вклад в кластеризацию?
2. Что такое кластеризация и классификация?
3. Как оценить результаты кластеризации?

4. Какие признаки вносят наибольший вклад в кластеризацию?

5. Какое правило связи между кластерами используется в методе k-средних?

6. Какое расстояние между объектами в кластерах используется в методе k-средних?

7. Поясните сущность метода k-means.

Практическое занятие № 5
Ассоциативные правила

Цель работы: изучить методы построения ассоциативных правил.

Теоретическая часть

Ассоциативные правила позволяют находить закономерности между связанными событиями. Примером такого правила служит утверждение, что покупатель, приобретающий "Хлеб", приобретет и "Молоко". Впервые эта задача была предложена для поиска ассоциативных правил для нахождения типичных шаблонов покупок, совершаемых в супермаркетах, поэтому иногда ее еще называют анализом рыночной корзины (market basket analysis).

Транзакция – это множество событий, произошедших одновременно. Пусть имеется база данных, состоящая из покупательских транзакций. Каждая транзакция – это набор товаров, купленных покупателем за один визит. Такую транзакцию еще называют рыночной корзиной.

Целью анализа является установление следующих зависимостей: если в транзакции встретился некоторый набор элементов X, то на основании этого можно сделать вывод о том, что другой набор элементов Y также должен появиться в этой транзакции. Установление таких зависимостей дает возможность находить очень простые и интуитивно понятные правила. Основными характеристиками таких правил являются **поддержка** и **достоверность**. Правило "Из X следует Y" имеет поддержку *s*, если *s %* транзакций из всего набора содержат наборы элементов X и Y. Достоверность правила показывает, какова вероятность того, что из X следует Y. Правило "Из X следует Y" справедливо с достоверностью *c*, если *c %* транзакций из всего множества, содержащих набор элементов X, также содержат набор элементов Y.

Алгоритмы поиска ассоциативных правил предназначены для нахождения всех правил вида "из X следует Y", причем поддержка и достоверность этих правил должны находиться в рамках некоторых

наперед заданных границ, называемых соответственно минимальной и максимальной поддержкой и минимальной и максимальной достоверностью. Границы значений параметров поддержки и достоверности выбираются таким образом, чтобы ограничить количество найденных правил.

Если поддержка имеет большое значение, то алгоритмы будут находить правила, хорошо известные аналитикам или настолько очевидные, что нет никакого смысла проводить такой анализ. С другой стороны, низкое значение поддержки ведет к генерации огромного количества правил, что, конечно, требует существенных вычислительных ресурсов. Тем не менее, большинство интересных правил находится именно при низком значении порога поддержки, хотя слишком низкое значение поддержки ведет к генерации статистически необоснованных правил.

Таким образом, необходимо найти компромисс, обеспечивающий, во-первых, интересность правил и, во-вторых, их статистическую обоснованность. Поэтому значения этих границ напрямую зависят от характера анализируемых данных и подбираются индивидуально.

Еще одним параметром, ограничивающим количество найденных правил, является максимальная **мощность** часто встречающихся **множеств**. Если этот параметр указан, то при поиске правил будут рассматриваться только множества, количество элементов которых будет не больше данного параметра.

Лифт – показатель зависимости элементов друг от друга. Если значение лифта меньше единицы, то правило покупки молока и хлеба в нашем примере на (1 – lift) % “слабее”, чем просто покупки хлеба, если же лифт > 1, то соответственное правило на (lift – 1) % “сильнее”.

Существуют множество алгоритмов составления ассоциативных правил, такие как Apriori, Eclat, FP-роста и другие.

Практическая часть

Будем использовать алгоритм Apriori.

Подключим пакет apriori, реализующий данный алгоритм, с помощью pip install apriori.

Подключим библиотеку pandas и загрузим данные, скачав их с репозитория по ссылке: https://github.com/adivyas99/Market-Basket-Optimization/blob/master/Market_Basket.csv.

```
import pandas as pd
dataset = pd.read_csv('Market_Basket_Optimisation.csv',
header=None)
dataset.head()
```

После этого заменим все значения NaN на последнее значение внутри транзакции для удобства обработки информации.

```
dataset.fillna(method='ffill', axis=1, inplace=True)
dataset.head()
```

Создаем для значений матрицу и обучаем модель.

```
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i, j]) for j in range(0, 20)])
import apriori
result = list(apriori.apriori(transactions, min_support=0.003,
min_confidence=0.2, min_lift=4, min_length=2))
```

Визуализируем полученный результат:

```
import shutil, os
try:
    from StringIO import StringIO
except ImportError:
    from io import StringIO
import json
output = []
for RelationRecord in result:
    o = StringIO()
    apriori.dump_as_json(RelationRecord, o)
    output.append(json.loads(o.getvalue()))
data_df = pd.DataFrame(output)
pd.set_option('display.max_colwidth', -1)
from IPython.display import display, HTML
display(HTML(data_df.to_html()))
```

В результате получим список ассоциативных правил.

Задание

1. Выполнить пример из практической части.
2. Выполнить задания, соответствующие варианту, после каждого шага фиксировать изменения.
3. Сравнить результаты, полученные в примере и в выбранном варианте, и на основе анализа сделать выводы.

Вариант 1

1. Увеличить значение поддержки на 0.002.
2. Уменьшить значение достоверности на 0.1.
3. Увеличить значение лифта на 0.4.

Вариант 2

1. Уменьшить значение поддержки на 0.001.
2. Увеличить значение достоверности на 0.1.

3. Увеличить значение лифта на 2.

Вариант 3

1. Увеличить значение поддержки на 0.004.
2. Уменьшить значение достоверности на 0.1.
3. Уменьшить значение лифта на 2.

Вариант 4

1. Увеличить значение поддержки на 0.002.
2. Уменьшить значение достоверности на 0.15.
3. Увеличить значение лифта на 1.5.

Вариант 5

1. Увеличить значение поддержки на 0.002.
2. Увеличить значение достоверности на 0.2.
3. Уменьшить значение лифта на 1.

Вариант 6

1. Уменьшить значение поддержки на 0.001.
2. Уменьшить значение достоверности на 0.1.
3. Увеличить значение лифта на 1.

Вариант 7

1. Увеличить значение поддержки на 0.003.
2. Уменьшить значение достоверности на 0.05.
3. Увеличить значение лифта на 0.8.

Вариант 8

1. Увеличить значение поддержки на 0.004.
2. Увеличить значение достоверности на 0.1.
3. Уменьшить значение лифта на 1.2.

Вариант 9

1. Увеличить значение поддержки на 0.003.
2. Уменьшить значение достоверности на 0.05.
3. Увеличить значение лифта на 1.

Вариант 10

1. Уменьшить значение поддержки на 0.0005.
2. Уменьшить значение достоверности на 0.1.
3. Увеличить значение лифта на 3.

Контрольные вопросы

1. Назовите известные вам алгоритмы генерации ассоциативных правил.

2. Какие исходные данные нужны для запуска алгоритма поиска ассоциативных правил?

3. Дайте определение понятиям поддержки и достоверности АП.

4. Что такое лифт ассоциативного правила?

5. Всегда ли АП с высокой поддержкой и достоверностью являются значимыми?

Практическое занятие № 6
Проектирование нейронной сети для распознавания рукописных чисел

Цель работы: научиться составлять модели нейронных сетей для работы с изображениями.

Теоретическая часть

Популярной демонстрацией способности методов глубокого обучения является распознавание объектов в данных изображений.

Набор данных MNIST был составлен из нескольких наборов данных отсканированных документов, доступных из Национального института стандартов и технологий(NIST). Отсюда и название набора данных, например модифицированный набор данных NIST или MNIST.

Изображения цифр были взяты из различных отсканированных документов, нормализованы по размеру и центрированы. Это делает его отличным набором данных для оценки моделей, позволяя разработчику сосредоточиться на машинном обучении с минимальной очисткой или подготовкой данных.

Каждое изображение представляет собой квадрат 28 на 28 пикселей (всего 784 пикселя). Стандартный набор данных используется для оценки и сравнения моделей, где 60 000 изображений используются для обучения модели, а отдельный набор из 10 000 изображений используется для ее проверки.

Практическая часть

Сначала необходимо импортировать библиотеки, которые нам понадобятся:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist  # Mnist
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten
```

Далее нам нужно загрузить набор данных MNIST, для этого воспользуемся вспомогательной функцией библиотеки Keras:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Так как значения пикселей – это шкала серого в диапазоне от 0 до 255, нам нужно нормализовать эти значения для их обработки. Для этого поделим значение каждого пикселя на 255.

```
# стандартизация входных данных
x_train = x_train / 255
x_test = x_test / 255
```

Выходной переменной является целое число от 0 до 9. Поэтому эффективнее будет использовать горячее кодирование значений класса, преобразовав вектор целых чисел в двоичную матрицу. Для этого используем встроенную вспомогательную функцию to_categorical () из библиотеки Keras.

```
y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)
Создаем модель нейронной сети.
model = keras.Sequential([
   Flatten(input_shape=(28, 28, 1)),
   Dense(128, activation='relu'),  # Скрытый слой
Dense(10, activation='softmax')  ]) # Выходной слой
# настройка НС
model.compile(optimizer='adam',
loss='categorical_crossentropy',#
            metrics=['accuracy'])
model.fit(x_train, y_train_cat, batch_size=32, epochs=10, validation_split=0.2)  # обучение НС
def digit_definition(res, option_number):
   if res == option_number:
      return 1
   else:
      return 0
count_numb = 25
for n in range(count_numb):
   x = np.expand_dims(x_test[n], axis=0)
   res = model.predict(x)
   print(res)
   print('Результат: ' + str(digit_definition(np.argmax(res), 3)))  # argmax возвращает индекс максимального значения вдоль указанной оси
   plt.imshow(x_test[n], cmap=plt.cm.binary)
   plt.show()
   print()
```

Модель представляет собой простую нейронную сеть с одним скрытым слоем.

Функция активации softmax используется на выходном слое для преобразования выходных значений в вероятностные значения и позволяет выбрать один класс из 10 в качестве выходного прогнозирования модели.

Задание. Изменить нейронную сеть так, чтобы результатом был ответ, есть ли на изображении последняя цифра вашего варианта. Затем вывести результат для 20 случайных изображений.

Контрольные вопросы

1. Что такое горячее кодирование значений класса?
2. Для чего необходима нормализация данных?
3. Дайте определение модели нейронной сети.
4. Как оценить созданную модель нейронной сети?
5. Какая функция потерь используется в работе?

Практическое занятие № 7
Проектирование свёрточной нейронной сети для распознавания рукописных чисел

Цель работы: научиться составлять модели свёрточных нейронных сетей для обработки изображений.

Задачи: реализовать свёрточную нейронную сеть для распознавания рукописных чисел.

Теоретическая часть

Свёрточная нейронная сеть — специальная архитектура искусственных нейронных сетей, нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения. Идея свёрточных нейронных сетей заключается в чередовании свёрточных и субдискретизирующих слоёв (слоёв подвыборки).

Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов — любая, по выбору исследователя.

Название архитектура сети получила из-за наличия **операции свёртки**, суть которой заключается в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а

результат суммируется и записывается в аналогичную позицию выходного изображения.

Основная задача свёрточных нейронных сетей - классификации изображений — это приём начального изображения и вывод его класса или группы вероятных классов, которая лучше всего характеризует изображение.

Когда компьютер видит изображение (принимает данные на вход), он видит массив пикселей. В зависимости от разрешения и размера изображения, например, размер массива может быть 32x32x3 (где 3 — это значения каналов RGB). Каждому из этих чисел присваивается значение от 0 до 255, которое описывает интенсивность пикселя в этой точке. Эти цифры, оставаясь бессмысленными для нас, когда мы определяем, что на изображении, являются единственными вводными данными, доступными компьютеру. Идея в том, что вы даете компьютеру эту матрицу, а он выводит числа, которые описывают вероятность класса изображения (80 для кошки, .15 для собаки, .05 для птицы и т.д.).

Архитектура свёрточной нейронной сети

В обычном перцептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой персональный весовой коэффициент. В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки. Её интерпретируют как графическое кодирование какого-либо признака, например наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков. Естественно, в свёрточной нейронной сети набор весов не один, а целая гамма, кодирующая элементы изображения (например, линии и дуги под разными углами). При этом такие ядра свёртки не закладываются исследователем заранее, а формируются самостоятельно путём обучения сети классическим методом обратного распространения ошибки. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многоканальной

(много независимых карт признаков на одном слое). Также следует отметить, что при переборе слоя матрицей весов её передвигают обычно не на полный шаг (размер этой матрицы), а на небольшое расстояние. Так, например, при размерности матрицы весов 5×5 её сдвигают на один или два нейрона (пикселя) вместо пяти, чтобы не «перешагнуть» искомый признак.

Операция субдискретизации («операция подвыборки» или операция объединения) выполняет уменьшение размерности сформированных карт признаков. В данной архитектуре сети считается, что информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения.

Рассмотрим типовую структуру свёрточной нейронной сети более подробно. Сеть состоит из большого количества слоёв. После начального слоя (входного изображения) сигнал проходит серию свёрточных слоёв, в которых чередуются собственно свёртка и субдискретизация. Чередование слоёв позволяет составлять «карты признаков» из карт признаков, на каждом следующем слое карта уменьшается в размере, но увеличивается количество каналов. На практике это означает способность распознавания сложных иерархий признаков. Обычно после прохождения нескольких слоёв карта признаков вырождается в вектор или даже скаляр, но таких карт признаков становятся сотни. На выходе свёрточных слоёв сети дополнительно устанавливают несколько слоёв полносвязной нейронной сети (перцептрон), на вход которому подаются оконечные карты признаков (рис. 5).

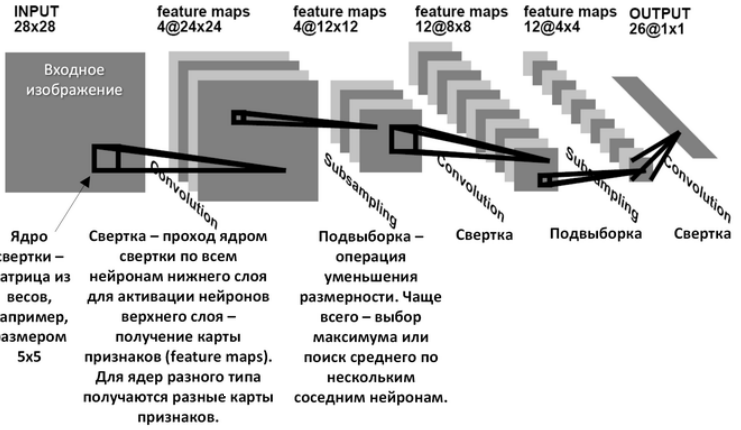


Рис. 5. Слои свёрточной НС

Слой свёртки

Слой свёртки (англ. convolutional layer) — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе обучения.

Особенностью свёрточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении.

Слой активации

Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки (считают, что функция активации встроена в слой свёртки).

Пулинг или слой субдискретизации

Слой пулинга (иначе - подвыборки, субдискретизации) представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размером 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём

изображения. Пулинг интерпретируется так: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой пулинга, как правило, вставляется после слоя свёртки перед слоем следующей свёртки.

Полносвязная нейронная сеть

После нескольких прохождений свёртки изображения и уплотнения с помощью пулинга система перестраивается от конкретной сетки пикселей с высоким разрешением к более абстрактным картам признаков, как правило, на каждом следующем слое увеличивается число каналов и уменьшается размерность изображения в каждом канале. В конце концов, остаётся большой набор каналов, хранящих небольшое число данных (даже один параметр), которые интерпретируются как самые абстрактные понятия, выявленные из исходного изображения.

Эти данные объединяются и передаются на обычную полносвязную нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью (по отношению к количеству пикселей исходного изображения).

Практическая часть

Производим импорт необходимых библиотек:

```
import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
```

Затем нам нужно загрузить набор данных MNIST и изменить его форму так, чтобы он был пригоден для использования при обучении CNN. В Keras слои, используемые для двумерных сверток, ожидают значения в пикселях с размерами [пиксели] [ширина] [высота].

В случае RGB пиксели первого измерения будут равны 3 для красного, зеленого и синего компонентов, и это будет похоже на наличие 3 входных изображений для каждого цветного изображения. В случае MNIST, где значения пикселей представляют собой шкалу серого, размерность пикселя устанавливается равной 1.

```
K.set_image_data_format('channels_first')
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], 1, 28,
28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
```

Так же, как и в предыдущей работе, нужно нормализовать входные данные и так же распределить выходные значения по классам.

```
X_train = X_train / 255
X_test = X_test / 255
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

Далее мы определим модель нейронной сети.

Сверточные нейронные сети являются более сложными, чем стандартные многослойные перцептроны, поэтому мы начнем с использования простой структуры, которая использует все элементы для получения самых современных результатов.

Первый скрытый слой — это сверточный слой, называемый Convolution2D.

Далее определим слой пула, который принимает максимум под названием MaxPooling2D. Он настроен с размером пула 2 × 2.

Следующий уровень — это уровень регуляризации с использованием dropout, называемый Dropout. Он настроен на случайное исключение 20 % нейронов в слое, чтобы уменьшить переоснащение.

Далее идет слой, который преобразует данные 2D-матрицы в вектор с именем Flatten. Это позволяет обрабатывать вывод стандартными полносвязными слоями.

Наконец, выходной слой имеет 10 нейронов для 10 классов и функцию активации softmax для вывода вероятностных прогнозов для каждого класса.

```
def baseline_model():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28),
activation='relu'))
```

```
        model.add(MaxPooling2D(pool_size=(2, 2)))
        # Добавим еще один свёрточный слой
        model.add(Conv2D(32, (5, 5), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(num_classes, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
        return model
    model = baseline_model()
    model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, batch_size=200, verbose=2)
    scores = model.evaluate(X_test, y_test, verbose=0)
    print("CNN Error: %.2f%%" % (100 - scores[1] * 100))
```

Сравните ошибку созданной свёрточной нейронной сети с ошибкой, созданной в прошлой работе нейронной сети с многослойным перцептроном.

Задание. Используйте свёрточную нейронную сеть для выполнения предыдущего задания, затем измените архитектуру так, чтобы процент ошибки был не больше 0.9.

Контрольные вопросы

1. Что такое сверточные нейронные сети? Для чего они используются?
2. Поясните алгоритм градиентного спуска.
3. Какие слои используются в CHC?
4. Как повысить точность результата нейронной сети?

Практическое занятие № 8
Наивный байесовский классификатор

Цель работы: реализовать наивный байесовский классификатор с помощью языка Python.

Теоретическая часть

Наивный байесовский классификатор — это модель машинного обучения, используемая для расчета вероятности, основанная на теореме Байеса**.**

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Теорема Байеса — одна из основных теорем теории вероятностей, которая позволяет определить вероятность события при условии, что произошло другое статистически взаимосвязанное с ним событие. По формуле Байеса можно уточнить вероятность какого-либо события, взяв в расчёт как ранее известную информацию, так и данные новых наблюдений.

P(A|B) — это апостериорная вероятность, относящаяся к **целевому классу** с заданными параметрами/атрибутами. Условная вероятность после соответствующих доказательств принимается во внимание.

P(A) — априорная вероятность класса. Безусловная вероятность целевого класса до принятия во внимание соответствующих свидетельств.

P(B|A) — это вероятность, относящаяся к вероятности неизвестных функций для заданного целевого класса. Форма совместного распределения вероятностей для предсказания неизвестных признаков.

P(B) – это априорная вероятность предиктора. Вероятность того, что признак верен без учета гипотезы.

Теорема Байеса использует предположение, что признаки в данных независимы друг от друга, наличие одного конкретного признака не влияет на другой, отсюда и термин «наивный».

Достоинства алгоритма:

- легко и быстро предсказывает класс набора данных, хорошо справляется с многоклассовым прогнозированием;
- производительность наивного байесовского классификатора лучше, чем у других простых алгоритмов;
- классификатор хорошо работает с категориальными признаками. Для числовых признаков предполагается нормальное распределение.

Недостатки алгоритма:

- если переменная имеет категорию (в тестовом наборе данных), которая не наблюдалась в обучающем наборе данных, то модель присвоит нулевую вероятность и не сможет сделать предсказание;
- значения спрогнозированных вероятностей не всегда являются достаточно точными;
- ограничением данного алгоритма является предположение о независимости признаков, однако в реальных задачах полностью независимые признаки встречаются крайне редко.

Алгоритм наивного Байеса – это классификатор, обучение которого идет очень быстро. Данный инструмент идеально подходит

для составления прогнозов в реальном времени. С его помощью можно предсказать вероятность нескольких классов целевой переменной.

Популярные области для применения наивного байесовского классификатора

1. Система рекомендаций. В сочетании алгоритма с методами коллаборативной фильтрации (Collaborative Filtering) можно создать рекомендательную систему, которая использует машинное обучение и методы добычи данных для учета невидимой информации (такой, как поиск фильмов пользователем и длительность просмотра). Цель – предсказание того, понравится ли пользователю данный продукт или нет.

2. Фильтрация спама и классификация текста. Наивный байесовский классификатор в основном используются для классификации текстов (благодаря лучшему результату в многоклассовых проблемах и правилу независимости) и имеет более высокую точность по сравнению с другими алгоритмами. В результате он широко используется в фильтрации спама.

Наивный байесовский классификатор упрощает вычисление вероятностей, предполагая, что вероятность каждого атрибута, принадлежащего данному значению класса, не зависит от всех других атрибутов.

Вероятность значения класса при заданном значении атрибута называется условной вероятностью. Умножая условные вероятности вместе для каждого атрибута для данного значения класса, мы получаем вероятность того, что экземпляр данных принадлежит этому классу. Чтобы сделать прогноз, мы можем вычислить вероятности экземпляра, принадлежащего каждому классу, и выбрать значение класса с наибольшей вероятностью.

Наивные основы часто описываются с использованием категориальных данных, потому что их легко описать и рассчитать с использованием соотношений. Также алгоритм поддерживает числовые атрибуты и предполагает, что значения каждого числового атрибута распределены нормально.

Практическая часть

Реализуем наивный байесовский классификатор.

Для этого подключим библиотеку для работы с файлами формата csv.

```
def add_file(file):   #функция добавления данных
    lines = csv.reader(open(file, "rt"))#считываем данные из строки
    dataset = list(lines)
```

```
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
```

После определим набор строк для обучения и для тестирования. Основываясь на законе Парето, 80 % - данные для обучения, 20 % - для тестирования.

```
def split_set(dataset): #разбиение данных
    trainSize = int(len(dataset) * 0.8)
    trainSet = []
    tmp = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(tmp))
        trainSet.append(tmp.pop(index))
    return [trainSet, tmp]
trainingSet, testSet = split_set(dataset)
```

Сводка собранных обучающих данных включает среднее значение и стандартное отклонение для каждого атрибута по значению класса. Они необходимы при создании прогнозов для расчета вероятности конкретных значений атрибута, принадлежащих каждому значению класса.

Разделим экземпляры обучающего набора данных по значению класса.

```
def divide_by_class(dataset): #разделяем экземпляры по значению класса
    distribut= {}
    for i in range(len(dataset)):
        vec = dataset[i]
        if (vec[-1] not in set):
            distribut[vec[-1]] = []
        distribut[vec[-1]].append(vec)
    #последний атрибут является значением класса
    return set
```

Необходимо рассчитать среднее значение каждого атрибута для значения класса. Будем использовать его как середину гауссовского распределения при расчете вероятностей.

Также следует вычислить стандартное отклонение каждого атрибута для значения класса. Стандартное отклонение описывает изменение разброса данных, оно будет применяться для характеристики ожидаемого разброса каждого атрибута в распределении Гаусса при расчете вероятностей.

```
def get_average_value(values): #получаем среднее значение
```

```
    return sum(values) / float(len(values))
```

Стандартное отклонение рассчитывается как квадратный корень из дисперсии. Дисперсия рассчитывается как среднее значение квадратов разностей для каждого значения атрибута из среднего значения.

```
def standard_deviation(values): #получаем стандартное
отклонение
    avg = getAverageValue(values)
    difference = sum([pow(x - avg, 2) for x in values]) /
float(len(values) - 1)
    return math.sqrt(difference)
```

Для заданного списка экземпляров рассчитаем среднее значение и стандартное отклонение для каждого атрибута. Используем zip с целью вычисления среднего значения и значения стандартного отклонения для атрибута.

```
def summ(set):
    #zip предназначен для группировки значений для каждого
атрибута
    summa = [(getAverageValue(attribute),
standardDeviation(attribute)) for attribute in zip(*set)]
    del summa[-1]
    return summa
```

Разделим обучающий набор данных на экземпляры, сгруппированные по классам. После этого рассчитаем результат для каждого атрибута.

```
def summ_by_class(set):
    divided = divideByClass(set)
    summa = {}
    for classValue, instances in divided.items():
        summa[classValue] = summ(instances)
    return summa
```

Рассчитаем вероятность того, что этот экземпляр данных принадлежит каждому классу, затем выберем классы с наибольшей вероятностью в качестве прогнозирования.

Используем гауссовскую функцию для оценки вероятности данного значения атрибута, учитывая известное среднее значение и стандартное отклонение для атрибута.

С учетом того, что атрибуты суммируются там, где они подготовлены для каждого атрибута и значения класса, результатом является условная вероятность заданного значения атрибута при заданном значении класса.

```
    def calculate_probability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x - mean, 2) / (2 *
math.pow(stdev, 2))))
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent
```

После этого объединим вероятности всех значений атрибута для экземпляра данных и получим вероятность того, что весь экземпляр данных принадлежит классу.

Вероятность экземпляра данных рассчитывается путем умножения вероятностей атрибута для каждого класса. Результатом является отображение значений классов на вероятности.

```
    def calculate_class_probabilities(summaries, inputVector):
        probabilities = {}
        for classValue, classSummaries in summaries.items():
            probabilities[classValue] = 1
            for i in range(len(classSummaries)):
                mean, stdev = classSummaries[i]
                x = inputVector[i]
                probabilities[classValue] *= calculateProbability(x, mean,
stdev)
        return probabilities
```

Найдем наибольшую вероятность.

```
    def max_prob(summ, inputVector):
        probabilities = calculateClassProbabilities(summ, inputVector)
        tmp, bigProb = None, -1
        for classValue, probability in probabilities.items():
            if tmp is None or probability > bigProb:
                bigProb = probability
                tmp = classValue
        return tmp #возвращаем связный класс
```

Оценим точность модели, делая прогнозы для каждого экземпляра данных в тестовом наборе данных.

```
    def get_predictions(summa, testSet):
        predictions = []
        for i in range(len(testSet)):
            result = predict(summa, testSet[i])
            predictions.append(result)
    return predictions #вернет список прогнозов для каждого
экземпляра теста
```

Последним этапом рассчитаем точность и объединим весь программный код.

```
    def get_accuracy(testSet, predictions):
```

```
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(testSet))) * 100.0
```

Задание. Реализуйте наивный байесовский классификатор, предназначенный для проверки на спам-сообщение, с использованием возможностей библиотеки Scikit-learn**.** Точность должна превышать 80 %.

Дополнительные материалы необходимо взять у преподавателя.

Контрольные вопросы

1. Что такое наивный байесовский классификатор?
2. Опишите плюсы и минусы наивного байесовского классификатора.
3. В каких сферах применяется наивный байесовский классификатор?
4. Для чего предназначена теорема Байеса?
5. Опишите основные возможности библиотеки Scikit-learn.

Практическое занятие № 9

Рекомендательные системы на основе сингулярного разложения

Цель работы: создать рекомендательную систему с помощью SVD.

Теоретическая часть

Рекомендательные системы — программы, которые пытаются предсказать, какие объекты будут интересны пользователю, имея определенную информацию о его профиле.

Можно составлять предложения на основе того, что люди уже покупали или просматривали. Это называется фильтрация на основе содержания. Но здесь возникают две проблемы.

1. Cold start — если о пользователе неизвестно ничего (в первый раз в магазине или только что зарегистрировался), то непонятно, на основе каких данных делать рекомендации.
2. Filter Bubble — учитывая только то, что известно о пользователе, нельзя предоставить что-то новое — тем самым мы искусственно ограничиваем свой ассортимент.

Чтобы устранить вышеупомянутые недостатки, используют коллаборативные системы. Они основаны на матрице предпочтений (user-item matrix) (рис. 6).

Предмет рекомендации Пользователь

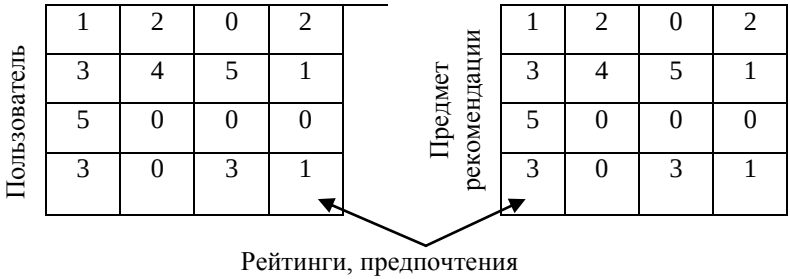


Рис. 6. user-based и item-based представления

Они делятся на два вида: user-based и item-based. Первые строят рекомендации на основе оценок пользователей, вторые основаны на самих продуктах.

Однако в реальных задачах матрицы предпочтений имеют гораздо большие размерности. Использование в работе таких матриц напрямую неудобно — получатся огромные по размерам массивы данных, порядка десятков и сотен терабайт.

Но их объём можно значительно уменьшить за счёт того, что такие матрицы, как правило, являются разреженными — ведь каждый пользователь не покупает все товары, а лишь крайне небольшую часть из них, не говоря уже о том, что оценивать будет ещё меньше. Это приводит к тому, что, хоть матрица предпочтений может иметь большую размерность, её ранг может быть куда более меньшим и можно использовать различные математические методы для оптимизации.

Один из таких приёмов — это метод сингулярного разложения. Он основывается на том, что любая матрица A может быть представлена в следующем виде:

$$A = U\Lambda V^T,$$

где U и V – ортогональные матрицы, а Λ — диагональная, и её элементы расположены по убыванию.

Кроме того, что мы так упрощаем представление матрицы, можно пойти дальше и перейти к усечённым матрицам — убрать последние k чисел в диагональной матрице, что приведёт к снижению размерности, и при этом полученная таким путём матрица будет наилучшим низкоранговым приближением по среднеквадратичному отклонению.

Сущность работы алгоритма

Сущность работы рекомендательной системы на основе SVD можно описать следующим образом.

1. Определиться с подходом: user-based или item-based. Выбор

зависит от предметной области. Так, если сбор данных от пользователей затруднён, то логичнее будет использовать item-based и наоборот.

2. Необходимо составить матрицу предпочтений на основе имеющихся данных — это могут быть как и полученные прямым путём, например непосредственно оценки пользователей, или же косвенными путями, например основываясь на времени просмотра того или иного контента.

3. С помощью алгоритма SVD разбить составленную матрицу на две составляющие: матрица скрытых предпочтений пользователей и скрытых факторов продукта.

4. Используя полученные матрицы, находить неизвестную оценку для пользователя.

Практическая часть

Рассмотрим на практике, как реализуется данный алгоритм. Для упрощения работы будем использовать библиотеку NumPy.

Выбираем матрицу предпочтений.

```
arr = [
   [0.1,0.2,0.6,0.05,0.05],
   [0.1,0.2,0.6,0.05,0.05],
   [0.1,0.25,0.3,0.1,0.25],
   [0.1,0.25,0.3,0.1,0.25],
   [0.8,0.1,0.05,0.05,0.0],
   [0.0,0.05,0.05,0.1,0.8],
   [0.0,0.05,0.05,0.1,0.8],
   [0.2,0.2,0.2,0.2,0.2],
   [0.9,0.1,0.0,0.0,0.0],
   [1,0.0,0.0,0.0,0.0],
   [1,0.0,0.0,0.0,0.0],
]
```

Далее применяем к ней функцию svd (рис. 3). Параметр full_matrices по умолчанию True необходимо изменить на False, иначе размерности матриц не будут совпадать. Сама функция возвращает кортеж из трёх матриц, которые и являются её сингулярным разложением. При этом третья матрица уже транспонированная. Также устанавливаем точность для вывода.

```
np.set_printoptions(precision=10, suppress=True)
U, D, V = np.linalg.svd(ratings, full_matrices=False)
```

Для сокращения размерности можно урезать матрицы. Это не особо сильно повлияет на точность, но позволит более эффективно работать программе при больших по размеру матрицах предпочтений.

```
users = U * D
```

```
value_for_slice = 9 # индекс, по которому будет выполнен срез
users = users[:,:value_for_slice]
products = V[:value_for_slice]
```

Теперь, чтобы получить ожидаемую оценку пользователя, необходимо умножить соответствующую строку матрицы users на соответствующий столбец матрицы products.

```
user_grade = users[4,:] @ products[:,4]
print(user_grade) # 2.9784816066994315
```

Задание. В примере была реализована рекомендательная система user-based. Вам же необходимо реализовать её на основе item-based. Для получения сингулярного разложения использовать библиотеку scikit-learn. При этом нужно рассмотреть случаи урезанной матрицы и проанализировать, насколько и как быстро падает точность при этом (точность оценивать с помощью RMSE из scikit-learn).

Варианты для выполнения задания взять у преподавателя.

Для установки SciKit необходимо установить её с помощью команды pip install scikit-learn и подключить командой import sklearn.

Контрольные вопросы

1. Что такое SVD? Опишите принцип его работы.
2. Для каких задач применяется алгоритм сингулярного разложения?
3. Почему нельзя использовать поиск собственных значений вместо сингулярных?
4. Какие существуют алгоритмы матричного разложения помимо SVD?
5. Опишите основные возможности библиотеки NumPy.
6. В чем отличие подхода user-based от item-based?

Практическое занятие № 10

Цель работы: познакомиться с алгоритмом раскрашивания черно-белых изображений, теорией цвета и построить нейросеть для раскрашивания незнакомых изображений.

Теоретическая часть

Чёрно-белые изображения можно представить в виде сетки из пикселей. У каждого пикселя есть значение яркости, лежащее в диапазоне от 0 до 255, от чёрного до белого (рис. 7).

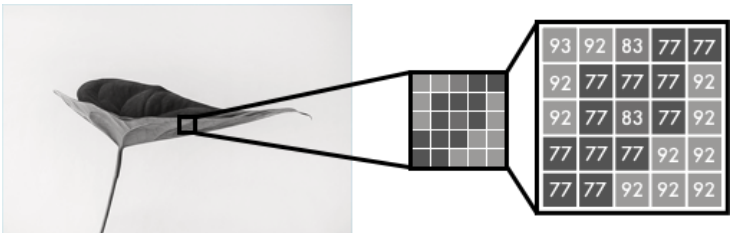


Рис. 7. Пиксели в черно-белом изображении

Цветные изображения состоят из трёх слоёв: красного, зелёного и синего. Слои определяют не только цвет, но и яркость. В цветном изображении с помощью трёх слоёв кодируются цвет и контрастность.

Поскольку нейросеть устанавливает взаимосвязь между входным и выходным значениями, то для раскрашивания изображения она должна найти связующие черты между чёрно-белыми и цветными изображениями. То есть необходимо найти свойства, по которым можно сопоставить значения из чёрно-белой сетки со значениями из трёх цветных.

Практическая часть

В первую очередь необходимо воспользоваться алгоритмом изменения цветовых каналов с RGB на Lab.

L означает - светлота (lightness), a и b — декартовы координаты, определяющие положение цвета в диапазоне соответственно от зелёного до красного и от синего до жёлтого.

В качестве входных данных необходимо взять слой с градациями серого и на его основе сгенерировать цветные слои a и b в цветовом пространстве Lab. Его же надо взять и в качестве L-слоя окончательной картинки (рис. 8).

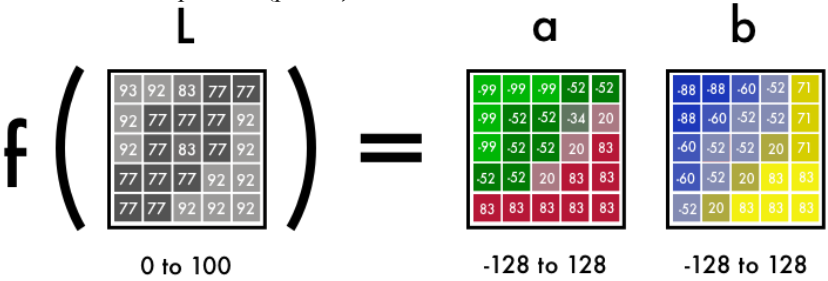


Рис. 8. Визуализация разложения на Lab

1. Для начала в папке проекта необходимо создать три папки: «Faces» (изображений для обучения), «Input» (для черно-белых

изображений на вход) и «Output» (пустая папка для окрашенных изображений).

2. В папку «Input» необходимо поместить изображения размера строго 256*256 пикселей.

```
import os
from os.path import isfile, join
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # 0 = all messages are logged, 3 - INFO, WARNING, and ERROR messages are not printed
from keras.layers import Conv2D, Conv2DTranspose, UpSampling2D
from keras.layers import Activation, Dense, Dropout, Flatten, InputLayer
from keras.layers.normalization import BatchNormalization
from keras.callbacks import TensorBoard
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import array_to_img, img_to_array, load_img
from skimage.color import rgb2lab, lab2rgb, rgb2gray
from skimage.io import imsave
import numpy as np
import random
import tensorflow as tf
import cv2

folder_train = "Faces"
folder_src = "Input"
folder_dst = "Output"
model_file = "faces1.h5"
brightness_corr = 250
do_train = True
# Get train images
X = []
for filename in os.listdir(folder_train):
    X.append(img_to_array(load_img(folder_train + os.sep + filename)))
X = np.array([cv2.resize(i, (256, 256)) for i in X], dtype=float)/255.0
# X = np.array(X, dtype=float)
Xtrain = X
# Model
model = Sequential()
model.add(InputLayer(input_shape=(256, 256, 1)))
```

```
        model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
        model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
strides=2))
        model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
        model.add(Conv2D(128, (3, 3), activation='relu', padding='same',
strides=2))
        model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
        model.add(Conv2D(256, (3, 3), activation='relu', padding='same',
strides=2))
        model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
        model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
        model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
        model.add(UpSampling2D((2, 2)))
        model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
        model.add(UpSampling2D((2, 2)))
        model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
        model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
        model.add(UpSampling2D((2, 2)))
        model.compile(optimizer='rmsprop', loss='mse')
        # Image transformer
        datagen = ImageDataGenerator(shear_range=0.2, zoom_range=0.2,
rotation_range=20, horizontal_flip=True)
        # Generate training data
        batch_size = 10
        def image_a_b_gen(batch_size):
            for batch in datagen.flow(Xtrain, batch_size=batch_size):
                lab_batch = rgb2lab(batch)
                X_batch = lab_batch[:,:,:,0]
                Y_batch = lab_batch[:,:,:,1:] / 128
                yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch)
        if do_train:
            # Train model
            model.fit_generator(image_a_b_gen(batch_size), epochs=1,
steps_per_epoch=400)
            # Save model
            model.save_weights(model_file)
# Load model
        model.load_weights(model_file)
        # Process images
        color_me = []
```

```
    onlyfiles = [f for f in os.listdir(folder_src) if isfile(join(folder_src, f))]
    for filename in onlyfiles:
        color_me.append(img_to_array(load_img(folder_src + os.sep + filename)))
    color_me = np.array([cv2.resize(i, (256, 256)) for i in color_me], dtype=float)
    # color_me = np.array(color_me, dtype=float)
    color_me = rgb2lab(1.0/255*color_me)[:,:,:,0]
    color_me = color_me.reshape(color_me.shape+(1,))
    # Test model
    output = model.predict(color_me)
    # Output colorizations
    for i in range(len(output)):
        cur = np.zeros((256, 256, 3))
        cur[:,:,0] = color_me[i][:,:,0]
        cur[:,:,1:] = output[i] * 128
        img_rgb = lab2rgb(cur)*brightness_corr
        imsave(folder_dst + os.sep + "img_%d.png" % i, img_rgb.astype(np.uint8))
        print("img_%d.png saved" % i)
```

Задание

1. Получите результат для 3 черно-белых изображений с помощью кода из примера (обучите сеть на 1 изображении).

2. Обучите нейронную сеть не на одном изображении, а на 15. Сравните результат на тех же изображениях, что были использованы в п. 1.

3. Опишите положительные и отрицательные стороны получившегося результата и/или алгоритма.

Контрольные вопросы

1. Что является слоем входа и слоем выхода для нейронной сети, которая раскрашивает изображения?

2. На какие 4 слоя можно разложить цветное изображение?

3. Какой алгоритм лежит в основе раскрашивания черно-белых изображений?

Практическое занятие № 11
Введение в обработку естественного языка

Цель работы: получение студентом навыков реализации базовых методов обработки естественного языка, включая

предобработку текста, формирование «мешка слов» («bag-of-words»), выделение стоп-слов и наиболее важных слов в документе, создание тематических моделей.

Теоретическая часть

Задачи обработки естественного языка – одни из самых востребованных в современном машинном обучении. Они включают в себя такие области, как машинный перевод, аннотирование текстов, классификация документов, генерация текста, text-to-image и text-to-video задачи, построение чат-ботов и диалоговых систем и многие другие.

Решение любой задачи в сфере NLP (natural language processing) начинается с предобработки текста, которая обычно включает в себя:

а) удаление всех не относящиеся к естественному языку символов из текста: @, #, & и т.д. (если это не противоречит цели исследования);

б) токенизиция текста, т.е. разделение его на отдельные слова: слово = «токен»;

в) удаление стоп-слов, встречающихся во всех без исключения текстах и не несущих смысловой нагрузки для решения текущей задачи;

г) приведение текста к нижнему регистру, чтобы модель распознавала такие слова, как, например, «привет» и «Привет», одинаково;

д) стэмминг (обрезка слова до его основания) и лемматизация текста [приведение слов к единой форме, например («красивая», «красивый», «красивое») ⇒ «красивый»].

Чтобы использовать методы машинного обучения на текстовых документах, нужно перевести текстовое содержимое (слова на естественном языке) в числовой вектор признаков. Наиболее интуитивно понятный способ сделать описанное выше преобразование — это представить текст в виде набора слов, после чего приписать каждому слову в тексте уникальный целочисленный индекс, соответствующий частоте его появления в документах обучающей выборки. Для этого в каждом документе i следует посчитать количество употреблений каждого слова w и сохранить это число в ячейке X[i,j]. Это будет соответствовать значению признака j, где j — это индекс слова w в словаре.

Такое преобразование текста в матрицу частот употреблений слов носит название «мешка слов» (или «bag of words»). Оно подразумевает, что вероятность появления слова в разных документах одинакова, и строит матрицу объекты-признаки на основании

предположения, что количество признаков соответствует количеству уникальных слов в корпусе документов. «Мешок слов» чаще всего является высокоразмерным разреженным набором данных (с большим количеством нулей).

Однако даже, если документы посвящены одной теме, в относительно длинных документах среднее количество словоупотреблений будет выше, чем в коротких, что может приводить к некорректной настройке моделей машинного обучения. Чтобы избежать потенциальных несоответствий, применяется подход, называемый TF-IDF – «term frequency – inverse document frequency», который позволяет оценить «важность» слова для данного документа. Этот подход позволяет снизить влияние низкоинформативных слов и, наоборот, повысить «вес» важных с точки зрения оценки принадлежности документа к определённой тематике.

Большое значение TF-IDF будут получать слова с высокой частотой внутри конкретного документа и с низкой частотой использования в других документах. Заполняя матрицу объекты-признаки значениями TF-IDF, можно эффективнее выделять важные «признаки» (слова) и проводить более качественную настройку моделей машинного обучения. Более продвинутыми с точки зрения выделения семантики различных слов являются современные решения, полученные в результате обучения на больших корпусах документов. К ним относится, в частности, инструмент word2vec, разработанный в 2013 году компанией Google и получивший широкое распространение. Word2vec вычисляет векторное представление слов, обучаясь на входных текстах. Это векторное представление основывается на контекстной близости: слова, встречающиеся в тексте рядом с одними и теми же словами (а следовательно, имеющие схожий смысл), будут иметь близкие координаты в многомерном пространстве (норма расстояния между их векторными представлениями будет мала).

Сформированные таким образом векторы позволяют вычислять «семантическое расстояние» между словами: например, слова «король» и «королева» будут находиться в этом представлении близко друг к другу, а слова «король» и «яблоко» - далеко.

Практическая часть

«Демонстрационный» сет данных содержит 163830 текстовых сообщений (отзывов об отелях), в сете присутствует сильный дисбаланс в классах (от 1 до 5 в соответствии с выставленными оценками), что ощутимо сказывается на итоговой точности предсказания.

Также тексты весомо отличаются по своей длине, при средней длине отзыва 50 токенов присутствует крупное количество, превышающее значения 100 токенов, – встречаются тексты и более тысячи токенов. Тестовые данные для чистоты эксперимента выделены в отдельный фрейм размером 25000 строк – по 5000 сообщений на каждый класс. В тестовый набор добавлены только сообщения с количеством токенов от 1 до 100.

Представленная ниже нейронная сеть на основе данных «демонстрационного сета» генерирует текст для первого класса (очень плохие отзывы).

Для начала необходимо написать функцию (tokenize_sentences) разбиения текстов корпуса, которая возвращает список токенов, разделённых фиктивным токеном ('END_SENT_START') в конце каждого предложения. ['начнем', 'с', 'того', 'что', 'в', 'этом', 'отеле', 'не', 'берут', 'деньги', 'только', 'за', 'воздух', 'END_SENT_START', 'звонок', 'с', 'телефона', ...]

```
import pandas as pd
import numpy as np
import sklearn
import re
import nltk
from nltk import tokenize
from nltk.tokenize import RegexpTokenizer

def get_text_for_label(df, label): # Получим список с текстами для каждого класса
    label = 'label == ' + str(label)
    df_label = df.query(label).drop(['label'], axis=1)
    return df_label.feedback.values.tolist()
feedback_label_1 = get_text_for_label(df_fin_feedback, 1) # К примеру, список для первого класса
def tokenize_sentences(text_corp): # Функцию разбиения текстов корпуса на токены с разбивкой по предложениям
    token_corp = []
    for text in text_corp:
        text = re.sub(r'\s+', ' ', text, flags=re.M)
        for sent in re.split(r'(?<=[.!?…])\s+', text):
            sent = sent.replace('\n',' ')
            for word in sent.split():
                token = re.search(r'[a-яёA-ЯЁa-zA-Z]+', word, re.I)
                if token is None:
```

```
                continue
        token_corp.append(token.group().lower())
        token_corp.append('END_SENT_START')  # В конце каждого предложения добавляем фиктивный токен
            return token_corp

        token_label_1 = tokenize_sentences(feedback_label_1)  # К примеру, список для первого класса
```

На вход следующей функции поступает получившийся список токенов. Функция возвращает словарь с группированными биграммами: ключи – отдельные токены, в значении – список слов, которые следуют за ними в корпусе с собственной частотой ({ 'отель': [('отличным', 4.116e-06), ('в общем', 2.058e-06), …)

```
        from collections import Counter, defaultdict
        def get_bigramms(token_list):
            bigramm_corp = []
            for i in range(len(token_list) - 1):
                bigramm = token_list[i] + ' ' + token_list[i + 1]

                bigramm_corp.append(bigramm)  # Получим список биграмм

            unique_token_count = len(set(bigramm_corp))  # кол-во уникальных биграмм
            bigramm_proba = {}  # Создаю словарик для результата: Ключ - биграмма. Значение – вероятность

            count_bigramm = Counter(bigramm_corp)  # Создаю словарь для хранения частот биграмм
            count_token = Counter(token_list)  # Создаю словарь для хранения частот токенов

            # Создаю словарь с группированными биграммами: ключи – отдельные токены,
            # в значение – список слов которые следуют за ними в корпусе
            # с собственной частотой ( { 'отель': [('отличным', 4.116e-06), ('вобщем', 2.058e-06), ...)
            grouped_bigramms = defaultdict(list)
            for bigramm in set(bigramm_corp):
```

```
            first_word, second_word = bigramm.split()
        proba = (count_bigramm[bigramm] + 1) / (count_token[first_word] + unique_token_count)  # Формула Лапласа
            grouped_bigramms[first_word].append((second_word, proba))
        return grouped_bigramms
    grouped_bigramm_1 = get_bigramms(token_label_1)  # На примере первого класса
```

Далее код для самой генерации предложений. Скрипт выдаёт 4 предложения по 15 слов каждое.

```
    import random
    def generate_texts(token_label, grouped_bigramm, label, count_text, count_sent, count_word):
    #Создаём словарь для подсчёта биграмм "исключений"
        exceptions_bigramm = defaultdict(int)
    #Создаём список уникальных токенов для старта предложения
        unique_token = list(set(token_label))
        texts = []
        for it_text in range(count_text):  # Цикл с диапазоном кол-ва текстов
            text = ''
            unique_word = set()
    #Цикл с диапазоном кол-ва предложений в тексте
            for it_sent in range(count_sent):
                len_sent = count_word
    #Генерируем случайное слово для начала предложения для обеспечения стохастического процесса генерации предложения
    start_word = random.choice(unique_token)
    # Записываем в строку с финальным предложением первое стартовое слово
                final_sent = start_word
    # Множество уникальных слов, которые уже сгенерировались в предложение (чтобы генерация не зацикливалась)
                unique_word.add(start_word)
    for step in range(count_word):
    next_word = None  # Создаём переменную для нового слова
    frequency = 0  # Переменная-счётчик для частоты каждого нового слова
    # Проходим циклом по словарю с ключом биграммы и значением её частоты
                    for second_word, freq in grouped_bigramm[start_word]:
                        bigramm = start_word + ' ' + second_word
```

```
        # Устанавливаем значение максимального повторения слова в одном тексте
        if exceptions_bigramm[bigramm] > 3:
        Continue
        if freq > frequency and second_word not in unique_word and second_word != 'END_SENT_START':
        next_word = second_word
        frequency = freq  # Если второе слово проходит условие, запоминаем его
        if next_word is None:  # Если подходящего по условиям слова не найдено, перезаписываем стартовое слово и начинаем поиск заново
        start_word = random.choice(unique_token)
                        final_sent += ', ' + start_word
        unique_word.add(start_word)
                    else:
        # Если после цикла нашли подходящее слово (которое запомнили в цикле), записываем его в предложение
        exceptions_bigramm[start_word + ' ' + next_word] += 1
        start_word = next_word
        final_sent += ' ' + next_word
        unique_word.add(start_word)
                final_sent += '. '
                text += final_sent
            texts.append(text)
          generation_text_df = pd.DataFrame(texts, columns=['feedback']) # Формируем фрейм из списка
            generation_text_df['label'] = label
            return generation_text_df[['label', 'feedback']]


        label_1_df = generate_texts(token_label_1, grouped_bigramm_1, label=1, count_text=15000, count_sent=4, count_word=15)
```

Для генерации семантика не имеет принципиального значения, важней частота появления слова в корпусе, и именно на такой результат заточен алгоритм.

Задание. Сгенерируйте по 3 примера текстов для отзывов на 5 звезд и на 3 звезды. Сравните полученные результаты. Сделайте выводы. Самостоятельно приведите сгенерированный набор к семантически верному виду.

Контрольные вопросы

1. Опишите сущность метода TF-IDF.
2. Опишите сущность метода Word2vec.

5949

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ В.Ф. УТКИНА

МАШИННОЕ ОБУЧЕНИЕ

методические указания к лабораторным работам и практическим занятиям

Рязань 2020

УДК 004.3
Машинное обучение: методические указания к лабораторным работам и практическим занятиям / Рязан. гос. радиотехн. ун-т; сост.: Е.Р. Муратов, А.С. Тарасов. – Рязань, 2020. – 32 с.
Содержат указания к выполнению лабораторных работ и практических занятий для студентов, обучающихся на направлении 09.03.01 «Информатика и вычислительная техника» и специальности 27.05.01 «Специальные организационно-технические системы». Предназначены для обучающихся очного, очно-заочного и заочного отделений.

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра электронных вычислительных машин Рязанского государственного радиотехнического университета (зав. кафедрой Б.В. Костров)

Машинное обучение

Составители: Муратов Евгений Рашитович
Тарасов Андрей Сергеевич

Рязанский государственный радиотехнический университет.
390005, Рязань, ул. Гагарина, 59/1.
Редакционно-издательский центр РГРТУ.

Машинное обучение – невероятно обширная группа методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме.

В состав машинного обучения входят:

- кластеризация (K-means, C-means);
- классификация (SVM, деревья решений);
- регрессия (линейная, полиномиальная);
- ансамблевые методы (беггинг, бустинг);
- нейросетевые методы.

Самыми быстроразвивающимися методами, безусловно, являются нейросетевые. Поэтому именно им и будут посвящены лабораторные работы. Изначально, без понимания математических основ сетей (которые, к слову, далеко не самые простые), осуществлять работу с ними было невозможно. Сейчас, с появлением новых средств работа с нейросетями вышла на принципиально другой уровень.

В рамках данного курса будет рассмотрено проектирование нейронных сетей с использованием TensorFlow и Keras. Математические основы нейронных сетей будут рассматриваться в меньшей мере, чтобы сделать наибольший акцент на практической части.

Нейронные сети, несомненно, являются мощным инструментом, открывающим доступ к решению различных задач в области обработки изображений, текста и звука. Однако важно понимать, что для получения действительно качественных результатов одной лишь нейронной сети бывает недостаточно. Необходимо уметь подобрать подходящие методы таким образом, чтобы получить максимально хорошие результаты.

Лабораторная работа № 1

Установка и развёртывание библиотеки Google TensorFlow и Keras

Цели работы: научиться самостоятельно производить настройку рабочего места, устанавливать необходимые среды и компоненты

Задача: на локальный компьютер произвести установку платформы Anaconda и компонентов TensorFlow и Keras.

Введение

Данный цикл лабораторных работ предназначен для студентов 3-го курса, обучающихся в бакалавриате по направлению 09.03.01 «**Информатика и вычислительная техника**» по предмету «**Машинное обучение**».

Цикл состоит из 4-х лабораторных занятий в 5-м учебном семестре.

Лабораторные работы ориентированы на использование языка **Python** и компонента для обучения **Keras.**

Историческая справка

В 1943 году У. Маккалок и У. Питтс формируют первые идеи компьютерных нейронных сетей, возникли первые реализации.

В 1949 году предлагается первый алгоритм обучения сетей.

В 1958 году Ф. Розенблатт формирует идею персептрона – наиболее популярной структуры нейронных сетей.

В 1963-1969 годах в СССР формируется список «трудных» задач, решение которых невозможно при помощи персептрона.

В 1974 году на новых моделях обучения формируются структуры многослойных сетей.

В 2015 году анонсирован Google TensorFlow – открытая библиотека для работы с глубокими нейронными сетями.

Тогда же, в 2015 году появился компонент Keras.

В 2019 году официально анонсирован Google TensorFlow 2.0. Теперь многие компоненты, такие как Keras являются частью платформы.

Практическая часть

Общие сведения

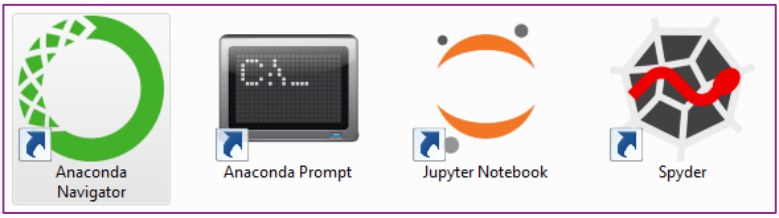
В дальнейшем для выполнения лабораторных работ предлагается использовать платформу **Anaconda** и среду разработки **Spyder**, однако существуют и другие средства для работы с языком (см. Приложение)

> *Anaconda – это платформа для научных исследований, основанная на языке программирования Python.*

Установить данную платформу можно с официального сайта: https://www.anaconda.com/. Дистрибутив изначально поставляется с интегрированной средой Python и набором библиотек. Для скачивания и установки рекомендуется дистрибутив со средой Python версии не ниже 3.7 и разрядностью x64.

Порядок настройки среды

После установки платформы выполните добавление поддержки необходимых для работы библиотек. Для этого запустите программу **Anaconda Navigator**.



Ярлык программ и компонентов, поставляемых в составе платформы Anaconda

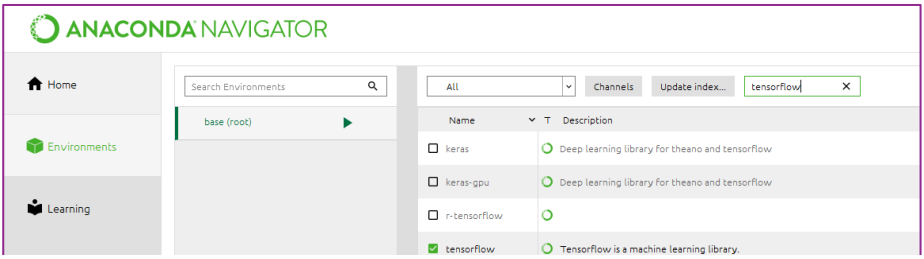
> *Anaconda Navigator – это графический интерфейс пользователя на рабочем столе (GUI), включенный в дистрибутив Anaconda®, который позволяет запускать приложения и легко управлять пакетами, средами и каналами conda без использования команд командной строки. Навигатор*

может искать пакеты в Anaconda Cloud или в локальном репозитории Anaconda. Он доступен для Windows, MacOS и Linux.

В открывшемся окне можно выбрать IDE, на базе которой будет производиться разработка, выбрать компоненты и библиотеки, а также ознакомиться с документацией по языку программирования **Python** и среде **Anaconda**. В рамках данного курса нам необходимо установить компоненты Google TensorFlow.

Для этого:

1. Перейдите в раздел **Environments.**
2. В выпадающем списке укажите значение **All**, чтобы увидеть все библиотеки.
3. Введите в поле поиска название требуемой библиотеки **Keras.**
4. В найденном списке найдите одноимённую библиотеку и установите напротив неё галочку.
5. Нажмите кнопку **Apply** для её установки.
6. В том случае, если среда предложит установить дополнительные компоненты, требуется согласиться с предложенным списком.



Режим настройки среды

Установка TensorFlow и Keras занимает порядка 15-25 минут в зависимости от скорости Интернета и возможностей ПК.

> *В том случае, если ваш ПК оборудован производительной дискретной видеокартой, имеет смысл установить keras-gpu вместо keras.*

Для выполнения лабораторных работ №3-5 потребуется библиотека технического зрения – **OpenCV**. Так как её нет в репозиториях **conda**, установку будем выполнять с помощью специальной утилиты **pip**, которая входит в состав среды **Python**.

Для этого:

1. Откройте программу **Anaconda Prompt.**
2. Введите команду **pip install opencv-python.**

На этом установка **TensorFlow** завершена.

Проверка работоспособности TensorFlow

Для оценки корректности установки выполните следующий код в одной из установленных сред разработки:

```
# Использование библиотеки Tensoflow
import tensorflow as tf

# Создадим объект в среде TF
hello = tf.constant('Hello, TensorFlow!')

# Создаем сессию
sess = tf.Session()

# Результат выполнения сессии
print(sess.run(hello))
```

В результате в отладочное окно выведется надпись: `b'Hello, TensorFlow!'`

Порядок выполнения работы

1. Установить Anaconda Navigator.
2. Выполнить установку TensorFlow и Keras.

3. Проверить корректность данной установки.

> *Подготовка отчёта и защита данной работы не предусмотрены*

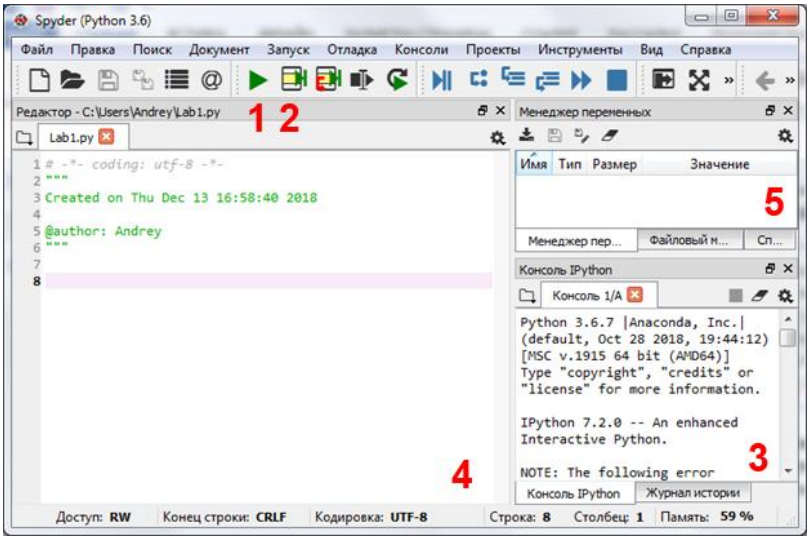
Приложение

Использование различных IDE

Spyder

Spyder — свободная и кроссплатформенная интерактивная IDE для научных расчетов на языке Python, обеспечивающая простоту использования функциональных возможностей и легковесность программной части. Данный программный продукт входит в комплекс платформы Anaconda.

Для запуска среды воспользуйтесь значком данной IDE из меню «Пуск».

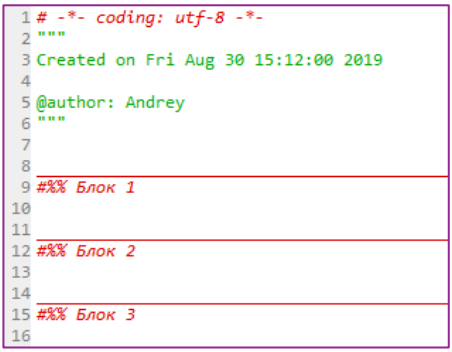


Среда разработки Spyder

Новые файлы по умолчанию сохраняются в папку пользователя (C:\users***username***\). Данная папка также является текущим каталогом.

Среда позволяет выполнять код целиком (кнопка 1 или клавиша «F5») или по блокам (кнопка 2 или клавиша «Ctrl+Enter»).

Блоки отделяются друг от друга при помощи последовательности **#%%**

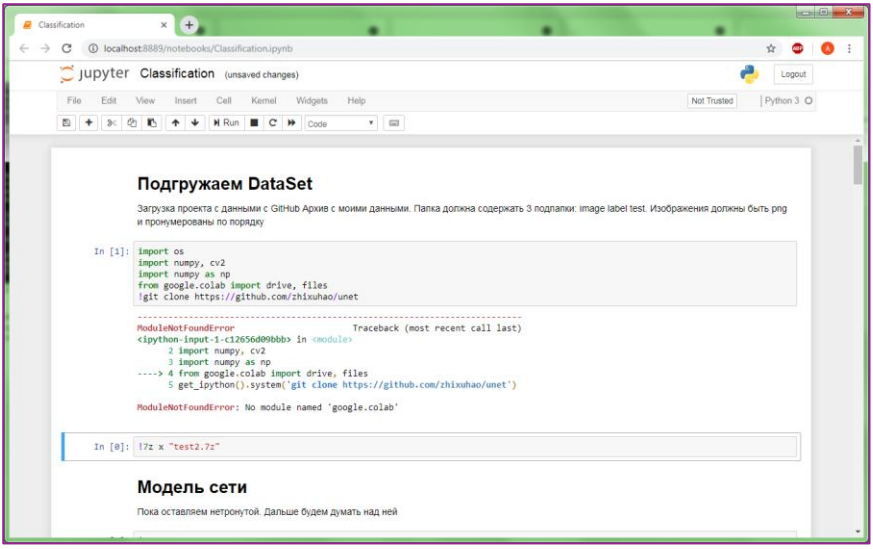


Консоль отладки 3 расположена внизу справа. В ней отображаются вывод сообщений, информация среды и сведения об ошибках.

Все текущие переменные отражены в блоке 5, сверху справа. Переменные будут сброшены только при перезапуске консоли или среды.

JUPYTER NOTEBOOK

Среда разработки, основанная на IPython, отличительная особенность которой – работа над кодом через веб-браузер. В момент запуска среды открываются веб-сервер, который отвечает за выполнение кода, и окно веб-браузера, через которое можно работать над своим кодом.



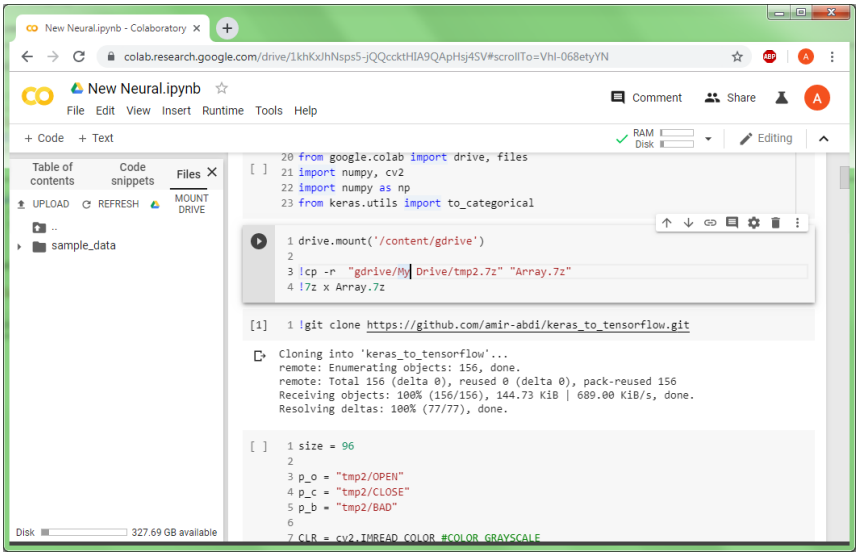
Среда Jupyter Notebook

Среда также может работать с блоками кода, однако выглядят они иначе. Добавление или удаление блоков осуществляется через команды меню. Кроме того, есть возможность добавления html-комментариев для улучшения визуального восприятия проекта.

Рабочая папка по умолчанию располагается также в каталоге пользователя.

Google Colab

Очень похожей на Jupyter Notebook является платформа Google Colab. Главной отличительной особенностью является то, что веб-сервер располагается не на локальном компьютере, а на виртуальных серверах компании Google.



Среда Google Colab

В зависимости от задач можно выбрать одну из трёх архитектур ЭВМ:

CPU – выполнение кода на удалённой виртуальной машине под управлением Intel Xeon E5 (1 ядро). 13 Гб RAM / 50 Гб SSD.

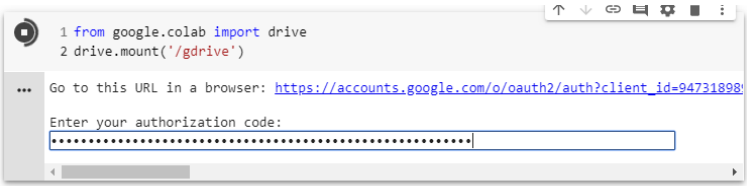
GPU – выполнение кода на удалённой виртуальной машине под управлением Intel Xeon E5 (1 ядро). 13 Гб RAM / 350 Гб SSD + видеокарта NVidia Tesla K80 13 Gb.

TPU – выполнение кода на удалённой виртуальной машине под управлением Intel Xeon E5 (1 ядро). 13 Гб RAM / 350 Гб SSD + тензорный вычислитель компании Google.

Использование режима GPU позволяет ускорить обучение до 100 раз в сравнении с домашним ПК. Выбрать режим можно через команду «Runtime» - «Change runtime type».

> *Среда является полностью бесплатной. Для доступа к ней требуется аккаунт Google. Ограничением данной среды является её доступность по времени: каждые 12 часов*

происходит «форматирование» и перезагрузка. Удаляются все файлы (за исключением набранного кода), и среда устанавливается заново. Для того чтобы не потерять свои результаты, можно подключить Google Диск как папку в рабочем каталоге. Для этого требуется выполнить следующий код:



Выполнение кода возможно только блоками (кнопка запуска находится слева от кода блока). Вывод каждого блока находится под кодом. Для управления файлами предусмотрена панель слева. Через неё осуществляются загрузка и выгрузка локальных данных.

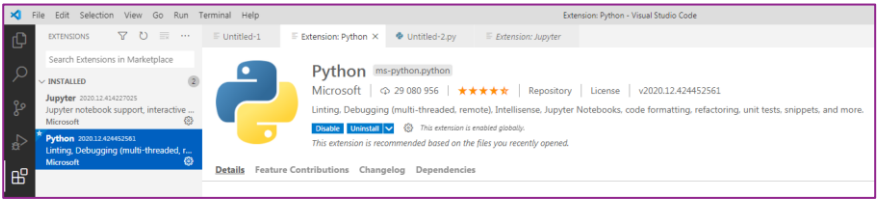
Среда поддерживает команды Linux. Для ввода команд используйте символ **!** перед названием самой команды. Например: **!sudo apt-get install nano.**

Для запуска среды перейдите по ссылке http://colab.research.google.com/ и создайте новый проект приложения Python 3.

VISUAL STUDIO CODE

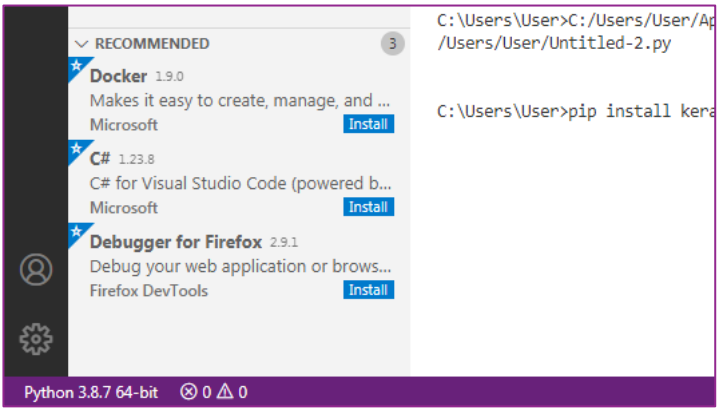
Создание программ на языке Python возможно и без применения платформы Anaconda. Для этого необходимо загрузить и установить пакет интерпретатора Python по ссылке (https://www.python.org/downloads/). При установке интерпретатора обязательно выберите пункт «**Add Python to Environment Variables**». После установки интерпретатора установите среду для разработки, например, Microsoft Visual Studio Code. Скачать её можно, например, с официального сайта (https://code.visualstudio.com/).

Запустите Visual Studio Code. Для работы с языком Python необходимо установить дополнение Python и/или Jupyter (В зависимости от того, в чём удобнее вести разработку).



Среда Microsoft Visual Studio Code

После установки компонента убедитесь, что интерпретатор был подключен к проекту. Если нет, выберите папку, в которую был установлен Python.



Версия интерпретатора указана в левом нижнем углу

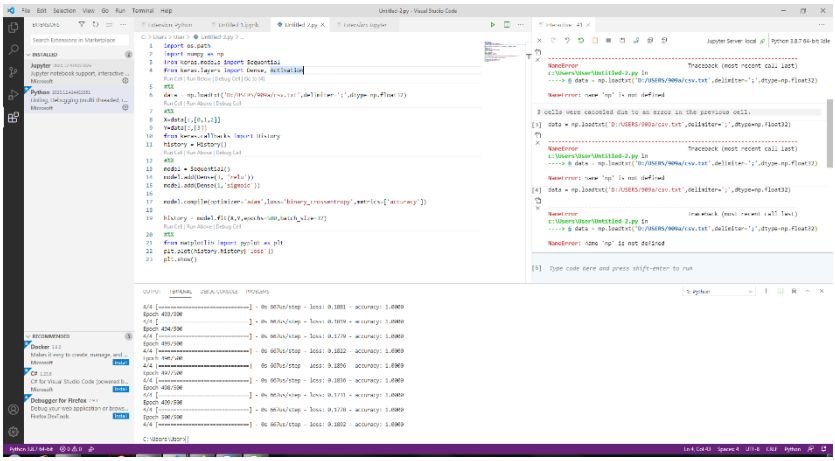
Для установки необходимых библиотек воспользуйтесь окном **TERMINAL** среды VS Code.

```
pip install keras tensorflow opencv-python
```

Если компьютер оборудован дискретной видеокартой компании NVIDIA, имеет смысл установить редакцию, оптимизированную под работу на GPU

```
pip install keras-gpu tensorflow-gpu opencv-python
```

После установки всех необходимых компонентов можно приступить к разработке. Создайте новый файл, сохраните его с расширением .py или .ipynb. После этого появятся панели инструментов для запуска проекта.



Среда Visual Studio Code

Лабораторная работа № 2

Проектирование простейших нейронных сетей с использованием Keras

Цели работы: изучить простейшие модели нейронных сетей и активационных функций, проанализировать зависимости величин и попытаться спрогнозировать результат.

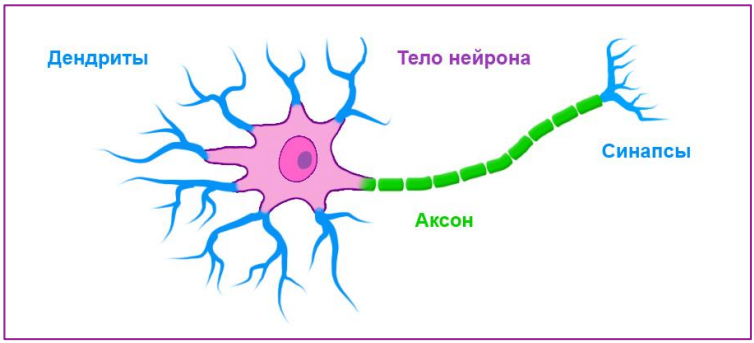
Задача: построить простейшую нейронную сеть, предназначенную для определения вида цветка ириса по его размерам.

Теоретическая часть

Искусственный нейрон как элементарная единица компьютерных нейронных сетей

Прежде чем переходить к нейронам искусственным, рассмотрим то, как устроен нейрон живого организма.

Для обмена сигналами используются электрические импульсы различной интенсивности. Они поступают либо от других нейронов, либо от окружающей среды через рецепторы, которые преобразовывают свет, звуки, прикосновения, вкусы и запахи в электрические сигналы.



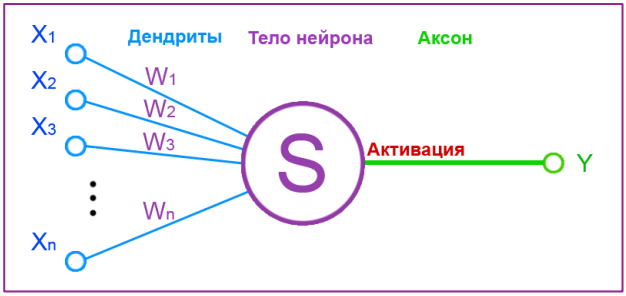
Модель нейрона коры головного мозга.

В процессе жизнедеятельности связи (**синапсы**) между нейронами либо укрепляются, либо разрушаются. Происходит подстройка значений или обучение.

Чем крепче связь, тем больше электронов перенесут заряд в сам **нейрон**.

В зависимости от интенсивности импульсов происходит насыщение или возбуждение нейрона. При наступлении определённой степени насыщения нейрон генерирует импульс, который после **аксона** поступит на другие нейроны.

По аналогии нейрону человека, в 1943 году У. Маккалок и У. Питтс описали искусственный нейрон и его математическую модель.



Схематическая модель компьютерного нейрона

$$S = \sum_{i=0}^{n} X_i w_i \qquad Y = f(S)$$

Математическая модель компьютерного нейрона

Вместо электрических импульсов применяются числовые значения, вместо синапсов – веса при входах, вместо аксона – активационная функция, которая выполняет нелинейное преобразование значений.

В качестве входных значений нейрона могут быть как значения выхода другого нейрона, так и исходные данные, отправленные на вход нейросети. В последнем случае такой нейрон будет называться **нейроном входного слоя**.

Выходное значение нейрона может быть передано другим нейронам либо же являться результатом работы нейронной сети. В последнем случае такой нейрон будет называться **нейроном выходного слоя**.

СТРУКТУРЫ НЕЙРОННЫХ СЕТЕЙ

В ходе данной работы будет рассмотрена наиболее простая модель нейронных сетей, которая называется персептроном.

Персептрон – это последовательная модель нейронной сети, в которой каждый нейрон предыдущего слоя соединён с каждым нейроном текущего слоя. Такая модель сетей получила наиболее широкое распространение из-за удобства построения и быстроты обучения. В данном цикле лабораторных работ будут рассматриваться только сети, построенные на данной модели.

Условно все нейроны персептрона можно разбить на слои.

Первый слой – слой входных данных нейронной сети, набор исходных значений, для которых мы пытаемся спрогнозировать результат.

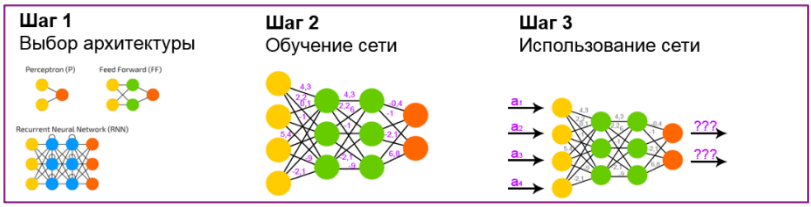
Последний слой – слой выходных данных нейронной сети, набор спрогнозированных значений для текущих входных показателей.

Скрытый слой – слой нейронной сети, который находится внутри самой сети и не является первым и последним.

Нейронная сеть, которая не имеет ни одного скрытого слоя, **сеть Кохонена** – самая простая нейронная сеть. Чем больше скрытых слоёв – тем более сложные зависимости входных величин сможет обнаружить нейронная сеть. Однако большие (глубокие) сети несут и большие проблемы: они занимают много памяти (m слоёв по n нейронов = $m * n^2$ значений весов), долго вычисляют значение и

обучаются, требуют большую обучающую выборку и могут переобучаться. Поэтому важно уметь подобрать оптимальное значение данных параметров.

Важно отличать **три процесса**: формирование модели сети, обучение сети и её эксплуатацию.



Процессы формирования сети

На 1-м шаге определяется вид сети: количество слоёв, нейронов в них, вид активационных функций и т.д.

На 2-м шаге, выполняется только подстройка значений весов сети.

На 3-м шаге, уже в процессе эксплуатации веса и структура уже жёстко зафиксированы и не меняются. То есть для одних и тех же входных значений всегда будет один и тот же результат.

> *Любую нейронную сеть можно представить в виде математической функции. Вопрос только в неудобстве данного представления.*

В общем плане нейронную сеть можно представить себе как функцию, где аргументы функции – вектор входных значений нейронной сети, а выходное значение – вектор выходных значений сети:

$$\bar{Y} = F(\bar{X}).$$

Практическая часть

Порядок выполнения работы

1. В соответствии со своим вариантом задания подготовьте данные обучающей выборки в формате CSV (разделитель символ “;”).

Выборка должна содержать 3 из 4-х параметров из базовой таблицы (по варианту задания) и вероятность того, что данный цветок принадлежит именно данному классу 0 или 1. Заголовки в таблице удалить!

Важно! *ОС Windows использует национальные стандарты при сохранении дробных чисел, даты, времени и т.д., в то время как Python не всегда поддерживает национальные стандарты. Поэтому может быть необходимо заменить разделитель дробной части с символа “,” на символ “.”.*

В результате должна получиться таблица, часть значений которой выглядит примерно следующим образом:

46	5,10	3,80	1,90	1,00
47	4,80	3,00	1,40	1,00
48	5,10	3,80	1,60	1,00
49	4,60	3,20	1,40	1,00
50	5,30	3,70	1,50	1,00
51	5,00	3,30	1,40	1,00
52	7,00	3,20	4,70	0,00
53	6,40	3,20	4,50	0,00
54	6,90	3,10	4,90	0,00

2. Сохраните данные в файл **train.csv**.
3. Перейдите в систему Google Colab или воспользуйтесь одной из IDE, установленной на вашем ПК.
4. Загрузите файл в папку проекта или в облако среды.
5. В первом блоке кода напишите фрагмент кода для загрузки данных из файла в среду:

```
# Базовые операции ввода-вывода данных
import os.path

# Библиотека для работы с числами
import numpy as np

# Импорт библиотеки Keras
from keras.models import Sequential
from keras.layers import Dense, Activation

# Извлечение данных из CSV-файла
data = np.loadtxt('data.csv', delimiter=';',dtype=np.float32)
```

6. Проверьте корректность загруженных данных.
7. В новом блоке выполните разбиение исходного файла на 2 массива: вектор входных данных X и вектор ожидаемых значений Y.

```
# Чтение сведений из массива
# Данная конструкция указывает на номера столбцов таблицы,
# которые будут выбраны из массива в обучающую выборку
X = data[:, [0, 1, 2]]
Y = data[:,[3]]
```

8. После загрузки входных сведений можно приступить к формированию модели нейронной сети.

```
# Инициализация модели сети. Используется структура перцептрон
model = Sequential()

# Входной слой из 3-х нейронов (значений) и Активационная функция
model.add(Dense(3, 'relu'))

# Далее попробуйте самостоятельно описать несколько скрытых слоёв.
# Они поисываются аналогично входому. Исследуйте разное число нейронов и
# разные активационные функции

# Выходной слой из 1 нейрона - вероятность попадания в класс
model.add(Dense(1, 'sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',metrics=['accuracy'])
model.fit(X, Y, epochs=500, batch_size=32)
```

> *Внимание! Данный пример не содержит ни одного скрытого слоя. Такая сеть вряд ли сможет выдать корректный результат в процессе своей работы. Предлагается описать*

свою структуру многослойного персептрона, которая бы давала лучшие результаты, чем у остальных. В качестве опорных значений точности обучения предлагается использовать log-loss и accuracy.

9. Попробуйте выбрать другое число эпох, а также проанализируйте, какой из оптимизаторов даёт лучшую сходимость к требуемым результатам.

```
45  # Вывод графика с результом
46  from matplotlib import pyplot as plt
47  plt.plot(history.history['loss'])
48  plt.show()
```

10. Оцените качество работы построенной сети при помощи нескольких примеров:

```
37  # Для вычисления результата с использованием нейросети необходимо
38  # подготовить входные данные и воспользоваться методом predict
39  # Возьмём 2 тестовых примера
40  test = np.asarray([[1,1,2], [2,2,2]])
41
42  # Получим значение
43  model.predict(test)
```

11. Подведите итоги работы. Оформите результаты в виде отчёта.

Подготовка отчёта

Отчёт о лабораторной работе должен содержать:

1. Титульный лист.
2. Цели и задачи данной работы.
3. Задание, согласно своего варианта (см. Приложение 1).
4. Программный код разных моделей нейронных сетей.
5. Результаты обучения: графики точности работы на разных моделях в зависимости от эпохи.
6. Результат работы программы.

Приложения

Приложение 1. Варианты заданий

Вариант задания	Описание	Параметр 1	Параметр 2	Параметр 3
1	Определить, относится ли указанный ирис к виду Iris setosa	Длина чашелистика	Ширина чашелистика	Длина лепестка
2	Определить, относится ли указанный ирис к виду Iris setosa	Длина чашелистика	Ширина чашелистика	Ширина лепестка
3	Определить, относится ли указанный ирис к виду Iris setosa	Длина чашелистика	Длина лепестка	Ширина лепестка
4	Определить, относится ли указанный ирис к виду Iris setosa	Ширина чашелистика	Длина лепестка	Ширина лепестка
5	Определить, относится ли указанный ирис к виду Iris versicolor	Длина чашелистика	Ширина чашелистика	Длина лепестка
6	Определить, относится ли указанный ирис к виду Iris versicolor	Длина чашелистика	Ширина чашелистика	Ширина лепестка
7	Определить, относится ли указанный ирис к виду Iris versicolor	Длина чашелистика	Длина лепестка	Ширина лепестка

8	Определить, относится ли указанный ирис к виду Iris versicolor	Ширина чашелистика	Длина лепестка	Ширина лепестка
9	Определить, относится ли указанный ирис к виду Iris virginica	Длина чашелистика	Ширина чашелистика	Длина лепестка
10	Определить, относится ли указанный ирис к виду Iris virginica	Длина чашелистика	Ширина чашелистика	Ширина лепестка
11	Определить, относится ли указанный ирис к виду Iris virginica	Длина чашелистика	Длина лепестка	Ширина лепестка
12	Определить, относится ли указанный ирис к виду Iris virginica	Ширина чашелистика	Длина лепестка	Ширина лепестка

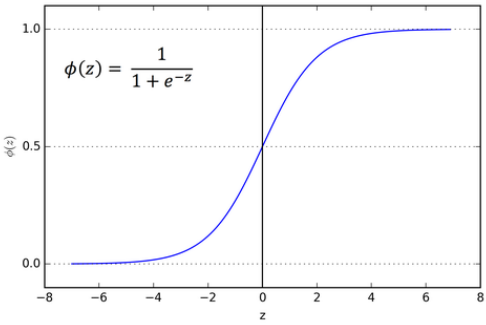
Приложение 2. Виды активационных функций

Sigmoid

Область определения: (-inf; +inf)

Область значений: (0; 1)

Одна из старейших функций активации. Аналогична работе нейрона человека. Отличается плавным, но резким насыщением в области перехода через 0. Главным недостатком считается низкая скорость вычисления.



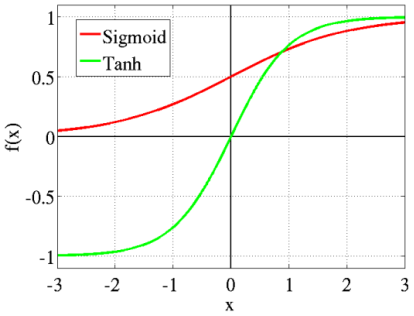
Функция сигмоиды

TANH

Область определения: (-inf; +inf)

Область значений: (-1; 1)

Одна из популярных активационных функций. Отличается от sigmoid более быстрым насыщением и наличием перехода через центр координат. Имеет тот же недостаток – низкую скорость вычисления.



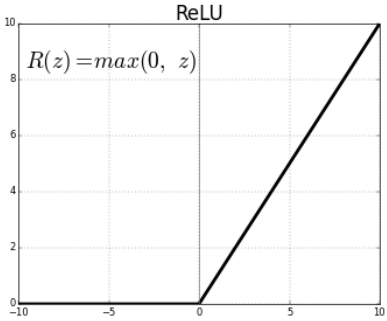
Функция гиперболического тангенса

RELU

Область определения: (-inf; +inf)

Область значений: [0; +inf)

В последние годы большую популярность приобрела функция активации ReLU. Она имеет следующую формулу: **f(x) = max(0, x)** и реализует простой пороговый переход в нуле. Главным недостатком считается быстрое «отмирание» нейронов при высокой скорости обучения. К концу обучения может оказаться так, что более 40 % нейронов никогда не активируются (не набирают на входе более 0).



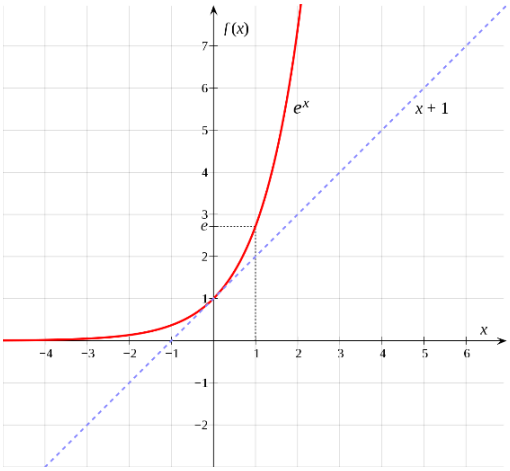
Функция линейного выпрямителя (ReLU)

EXPONENTIAL

Область определения: (-inf; +inf)

Область значений: [0; +inf)

Ещё одним вариантом активации является экспоненциальная активация. Она основана на применении классической функции exp(x).



Функция экспоненты

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое многослойный персептрон?
2. Что такое активационная функция, зачем она нужна?
3. Что такое оптимизатор нейронной сети?
4. Что такое обучающая выборка?
5. Почему оценка точности работы сети на обучающей выборке не является корректной?
6. Что такое переобучение? В каких случаях оно возникает?
7. Что такое входной и выходной слой нейронной сети?
8. Изобразите схематично график функции ReLU, Sigmoid.
9. Изобразите в виде графа примеры многослойного персептрона.
10. Что такое Keras и TensorFlow?

Лабораторная работа № 3

Использование нейронных сетей в задачах классификации изображений

Цели работы: изучить свёрточные нейронные сети, научиться их проектировать; ознакомиться с наиболее известными архитектурами сверточных сетей; научиться собирать данные для обучения.

Задачи: подготовить данные для обучения нейронной сети; сформировать модель, обучить сеть на размеченной выборке и протестировать её с использованием OpenCV в одном из языков программирования.

Теоретическая часть

Свёрточные нейронные сети

В ходе предыдущей работы были рассмотрены полносвязные нейронные сети, в которых каждый нейрон предыдущего слоя связан с каждым нейроном последующего слоя. Такая организация удобна в том случае, если во входном слое имеются 2,3, 10 или даже 20 значений. Но как быть в случае, если на вход нейронной сети подаётся цветное фото размером 100x100 пикселей? Такой входной слой будет представлен 30.000 нейронов, и, если предполагаются полные связи со следующим слоем, который содержит в себе столько же нейронов, количество связей будет составлять $9*10^8$ весовых коэффициентов. 400 Мб потребуется лишь для хранения коэффициентов на первом слое. Из этого следует, что применение такой стратегии явно не будет корректным для обработки изображений.

Впервые термин «Свёрточная нейронная сеть» был введен Яном Лекуном в 1988 году как эффективный подход при

распознавании образов. Метод основан на особенности строения зрительной системы человека.

Входными рецепторами сетчатки глаза человека считаются палочки и колбочки, которые определяют яркость света и его цветовой тон. Их количество составляет более $1{,}4*10^7$. Однако до коры головного мозга доходят около 10^6 нейронов. Следовательно, ещё до коры головного мозга происходит снижение количества информации, с использованием специализированных нейронов, которые реагируют уже не на изменение освещенности в пределах заданной точки, а на элементарные структуры – линии, простые фигуры, вспышки, пятна и т.д. Такие клетки получают данные от определённой группы соседних палочек/колбочек. Такой подход позволяет живому организму мгновенно отреагировать на быстро изменяющиеся условия среды.

Аналогичный подход был использован и в искусственных свёрточных сетях. Данные, поступаемые через входные рецепторы (в данном случае пиксели матрицы), сжимаются в несколько раз при помощи специальных операций, а уже после отправляются на полносвязные слои. Такие операции принято называть свёрткой (Conv) и подвыборкой (Pool).

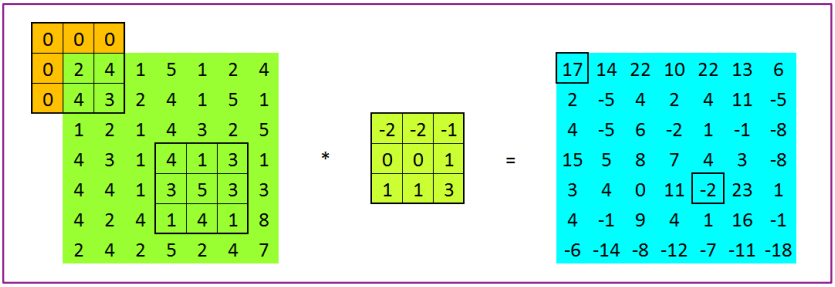
> *В разной литературе операция подвыборки может также называться субдискретизацией или прореживанием.*

Свёртка и подвыборка

Свёртка

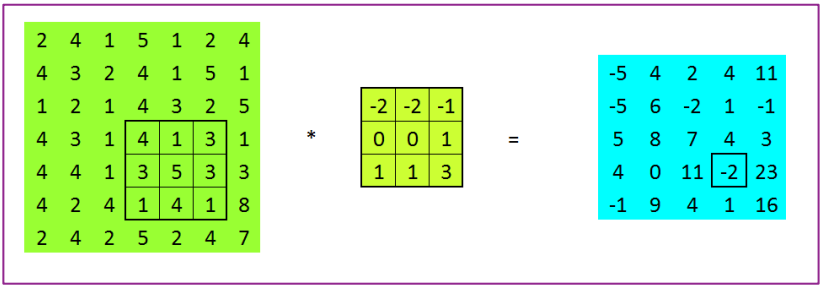
Свёртка – это операция выделения требуемых характеристик изображения. Она основана на прохождении по текущему слою матрицей NxN (обычно 3x3). Таких матриц используется несколько. Каждая из них нацелена на выделение каких-то своих признаков на изображении (линии, яркие точки, пятна и т.д.). Коэффициенты в этих матрицах и есть те самые весовые коэффициенты, которые

определяются во время обучения нейронной сети. Пример свёртки изображен на рисунке.



Свёртка с расширением пространства признаков

Выполнение свёртки окном 3x3. Обычно таких окон несколько, и в результате получается несколько новых матриц. В данном примере свёртка выполняется с дополнением пространства признаков. Дополнение выполнено нулями. Если бы дополнение не выполнялось, то пространство приказов бы уменьшилось на 2 единицы аналогично примеру на рисунке.

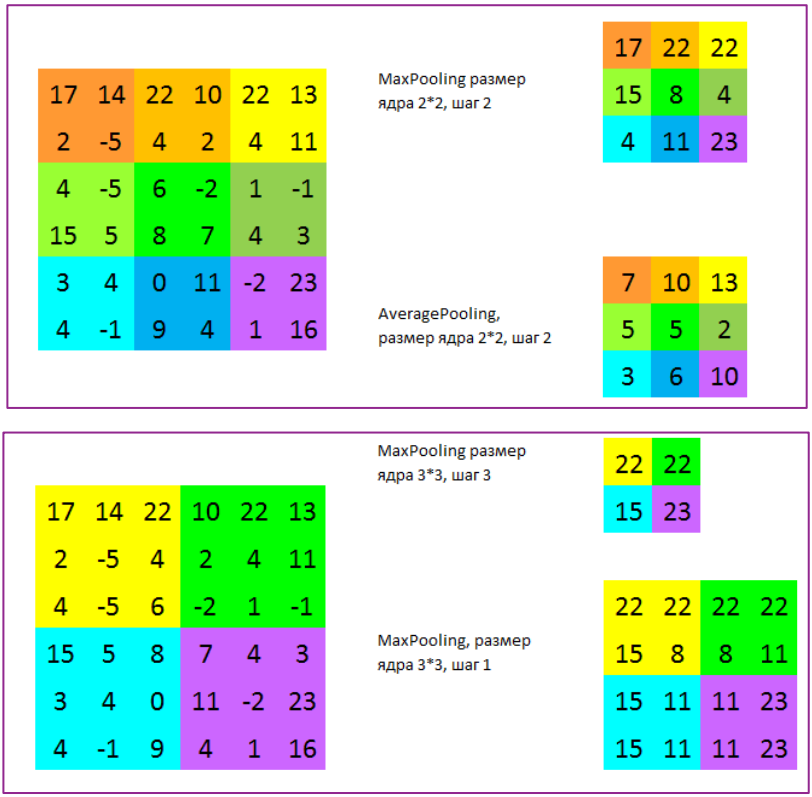


Свёртка без расширения пространства признаков

Подвыборка

После выполнения свёртки выполняется подвыборка значений. Она призвана сократить размерность карты признаков.

Подвыборка обычно выполняется при помощи операции MaxPooling, которая призвана значительно сократить пространство признаков (рис. 3).



Различные подходы при выполнении подвыборки

После выполнения операции подвыборки все имеющиеся значения активируются, обычно при помощи функции ReLU, и вновь поступают на вход следующего свёрточного слоя.

Практическая часть

Работа с библиотекой Keras

Объявление параметров

Ниже приведен фрагмент кода с инициализацией констант, которые потребуются для дальнейшей работы программы.

```
# Размер входного изображения
size = 96

# Папка с обучающей выборкой
train_dir = "training"

# Цвет изображения
CLR = cv2.IMREAD_COLOR #COLOR GRAYSCALE

# Количество классов
classes = 2
```

Подключение необходимых библиотек

Следующий фрагмент кода предназначен для подключения библиотек Keras и OpenCV к программе.

```
import numpy, cv2
import os
import numpy as np
import keras
from keras.optimizers import Adam
from keras.models import Model, Sequential
from keras.layers import Dense, Input, Drop
out, Flatten, Conv2D, MaxPooling2D, GlobalA
veragePooling2D
from keras.models import Sequential
```

```
from keras.layers.normalization import Batc
hNormalization
from keras.layers.convolutional import Conv
2D
from keras.layers.convolutional import MaxP
ooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras import backend as K
from keras.utils import to_categorical
```

ФУНКЦИЯ ЗАГРУЗКИ ОБУЧАЮЩЕЙ ВЫБОРКИ

Следующий фрагмент кода предназначен для чтения изображений с указанным размером и цветовым пространством из требуемого каталога. Если данный код выполнится успешно, то в поле output будет выведена таблица Ключ-Значение для ваших классов. Сохраните её, она потребуется позже.

```
def create_train_data():
  # Массивы X и Y
  train_data = []
  train_labels = []

  #Индекс класса
  pp = 0

  # По всем классам(папкам) и по всем файла
м в них
  for p1 in sorted(os.listdir(train_dir)):
```

```
    for img in os.listdir(os.path.join(trai
n_dir, p1)):
            # Формируем путь к файлу
            path = os.path.join(os.path.join(
train_dir, p1), img)
            # Изменяем размер изображения и ц
ветовое пространство
            img = cv2.resize(cv2.imread(path,
 CLR), (size, size))
            # Добавляем картинку в обучающую
выборку
            train_data.append(list(np.array(i
mg)))
            # А также её индекс
            train_labels.append([pp])
    print(pp,p1)
    pp=pp+1

  # Формируем и выгружаем массивы
  train_labels = np.array(train_labels)
  train_data = np.array(train_data)
  return (train_data,train_labels)
```

Загрузка выборки и её преобразование

Так как нейронная сеть должна возвратить не номер класса, а вероятности, то превратим одномерный массив индексов классов в двумерный по следующей схеме:

[0] => [1, 0, ..., 0]
[1] => [0, 1, ..., 0]
...
[n] => [0, 0, ..., 1]

```
(X_train, Y_train) = create_train_data()
Y_train_en = to_categorical(Y_train, num_cl
asses=classes, dtype='float32')
```

Объявление модели сети

Выполняем построение модели будущей сети. Модель последовательная, состоит из 6 свёрточных слоёв и 2 полносвязных в самом конце.

```
# Конфигурируем модель сети                как
последовательную
def build_model():
    model = Sequential()

    # размером SIZE*SIZE*3. В первом блоке
свёртка окном 3*3
    model.add(Conv2D(32, (3, 3), padding="s
ame", input_shape=(size, size, 3)))
    model.add(Activation("relu"))   # Актива
ция ReLU
    # Подвыборка квадратом 3*3 максимума из
 квадрата
    model.add(MaxPooling2D(pool_size=(3, 3)
))

    # [32, size/3, size/3]
    model.add(Conv2D(32, (3, 3), padding="s
ame"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)
))
```

```
    # [32, size/6, size/6]
    model.add(Conv2D(64, (3, 3), padding="s
ame"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)
))

    # [64, size/12, size/12]
    model.add(Conv2D(128, (3, 3), padding="
same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)
))

    # [128, size/24, size/24]
    model.add(Conv2D(64, (3, 3), padding="s
ame"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)
))

    # [64, size/48, size/48]
    model.add(Conv2D(32, (3, 3), padding="s
ame"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)
))

    # [32, size/96, size/96]
    model.add(GlobalAveragePooling2D())
    model.add(Dense(1024))
    model.add(Activation("relu"))
```

```
    model.add(Dense(classes))
    model.add(Activation("softmax"))
    return model
```

Компиляция сети и обучение

После того как были выполнены все подготовительные этапы, можно приступить к обучению сети. Оценивая результаты работы на тестовой выборке, попробуйте определить, при каком количестве эпох нейронная сеть не переходит в переобучение.

```
model = build_model()
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])

model.fit(X_train,Y_train_en,epochs=500, batch_size=32)
```

Проверка работоспособности в Python

Для проверки нейронной сети воспользуемся следующим программным кодом. Он выведет вероятности того, что изображение принадлежит тому или иному классу.

```
raw = cv2.resize(cv2.imread("SomeFile.png",
 CLR), (size, size))
raw = np.expand_dims(raw, 0)
model.predict(raw)
```

Экспорт нейронной сети

Для того чтобы получить возможность работы с обученной нейронной сетью при помощи OpenCV, необходимо выгрузить модель специальным образом. Для этого выполните следующий скрипт и загрузите к себе на ПК файл с получившейся моделью (Final_model.pb)

```
# Скачиваем скрипт для конвертации сети
!git clone https://github.com/amir-
abdi/keras_to_tensorflow.git

# Сохраняем сеть в формате Keras
model.save("Keras_model.h5")

# Сохраняем описание узлов в описании JSON
model_json = model.to_json()
with open("JSON_model.json", "w") as json_f
ile:
    json_file.write(model_json)

# Сохраняем описание узлов в "замороженном"
 виде в формате protobuf
!python keras_to_tensorflow/keras_to_tensor
flow.py --input_model="Keras_model.h5" --
input_model_json="JSON_model.json" --
output_model="Final_model.pb"
```

Работа с моделью сети в OpenCV

Реализации библиотеки OpenCV существуют для большинства современных языков программирования. В ходе данной работы предлагается реализация решения на языке C#.

> *Вы можете выбрать любой другой язык программирования, если вашего опыта будет достаточно, чтобы реализовать на нём данное решение.*

Создание проекта

1. Создайте новый проект Windows Forms Application в среде Microsoft Visual Studio (потребуется версия 2013 и старше).
2. Сохраните решение.
3. Используя диспетчер пакетов Nuget найдите и установите пакет OpenCvSharp-AnyCPU, author: schimat.

Добавление компонентов на форму

Для работы программы добавьте на форму следующие элементы: **PictureBox, OpenFileDialog, Button.** Расположите элементы по своему усмотрению. Для кнопки укажите надпись: "Открыть..." и добавьте подписку на событие Click.

Написание программного кода

Перейдите к программному коду. Добавьте директивы

```
using OpenCvSharp;
using OpenCvSharp.Dnn;
```

Используя список «Ключ-Значение», полученный ранее опишите перечислитель по своим классам:

```
enum Labels
{
```

```
    Собака = 0,
    Кот = 1,
    Крокодил = 2,
    Человек = 3
}
```

Объявите модель нейронной сети как глобальную переменную и укажите модель из файла

```
  Net NeuralNetwork = CvDnn.ReadNetFromTensorflow(@"Final_model.pb");
```

Для нажатия кнопки опишите следующий код:

```
// Диалоговое окно выбора файла
if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
{
    listBox1.Items.Clear();
    // Выводим картинку
    pictureBox1.Image = new Bitmap(openFileDialog1.FileName);

    // Читаем изображение
    Mat m = new Mat(openFileDialog1.FileName);

    // Нормируем его
    Mat blob = CvDnn.BlobFromImage(m, 1, new OpenCvSharp.Size(96, 96), new Scalar(0, 0, 0), false, false);

    // Отправляем на вход сети
    NeuralNetwork.SetInput(blob);

    // Получаем результат
    Mat result = NeuralNetwork.Forward();
```

```
    // Определяем наиболее вероятный индекс и выводим
 его на экран
    OpenCvSharp.Point p1, p2;
    Cv2.MinMaxLoc(result, out p1, out p2);
    MessageBox.Show("На рисунке представлен класс: "
+ ((Labels)p2.X).ToString());
}
```

Порядок выполнения работы

1. Подготовить выборку по своему варианту задания. Выборка должна содержать не менее 200 изображений каждого класса.

> *Можете воспользоваться утилитой в файлах к лабораторной работе, однако не забудьте проверить полученную выборку на адекватность*

2. Из выборки выделите в отдельную папку 10-15 картинок для тестирования полученной нейронной сети. Из основной выборки их требуется удалить.
3. Полученную обучающую выборку отправьте в рабочий каталог среды.

> *На Google Colab это можно сделать, запаковав данные в архив. Распаковать их можно при помощи команды:* **!7z x Array.7z**

4. Загрузите обучающую выборку как массив средствами OpenCV.
5. Сконфигурируйте модель нейронной сети для классификации изображений.
6. Обучите данную сеть. Проанализируйте точность работы на обучающей выборке на 1, 10, 25, 50-и эпохах обучения.
7. Сохраните граф нейронной сети с весами.
8. Создайте новый проект в C#.
9. Добавьте поддержку библиотеки OpenCV.

10. Напишите программу, которая для указанного графического файла определяла бы класс.
11. Подведите итоги работы. Оформите результаты в виде отчёта.

Подготовка отчёта

Отчёт о лабораторной работе должен содержать:

1. Титульный лист.
2. Цели и задачи данной работы.
3. Задание согласно своего варианта.
4. Содержание обучающей выборки (несколько картинок).
5. Программный код.
6. Результаты обучения: графики точности на тестовой и обучающей выборках.
7. Результат работы программы.

Приложение

Варианты заданий

Вариант задания	Описание	Классы
1	Определить, какой кустарник изображен на фотографии	Черника, брусника, клюква, малина
2	Определить, фасад какого из университетов изображён на фото	РГРТУ, РГУ, РВВДКУ, РГАТУ, Другие
3	Определить, какое животное изображено на фото	3-4 класса на выбор студента
4	Определить страну, которой принадлежит автомобильный номер	Россия, Беларусь, Казахстан, Другой номер, номер отсутствует на фото

5	Определить, в каком городе расположен данный Кремль	Москва, Рязань, Коломна, Тула, Н. Новгород, Владимир

Контрольные вопросы

1. Что такое классификация?
2. Что такое свёрточная нейронная сеть?
3. Что такое Conv2D и MaxPooling?
4. Что такое обучающая и тестовая выборка?
5. Почему оценка точности работы сети на обучающей выборке не является корректной?
6. Зачем изображения приводятся к одному размеру перед подачей на вход нейросети?

Лабораторная работа № 4

Использование нейронных сетей в задачах сегментации данных

Цели работы: изучить архитектуру сегментационной свёрточной нейронной сети UNet, понять принципы её работы и проанализировать основные сферы её применения.

Задачи: подготовить данные для обучения нейронной сети, сформировать модель, обучить сеть на размеченной выборке и протестировать её на тестовой выборке.

Теоретическая часть

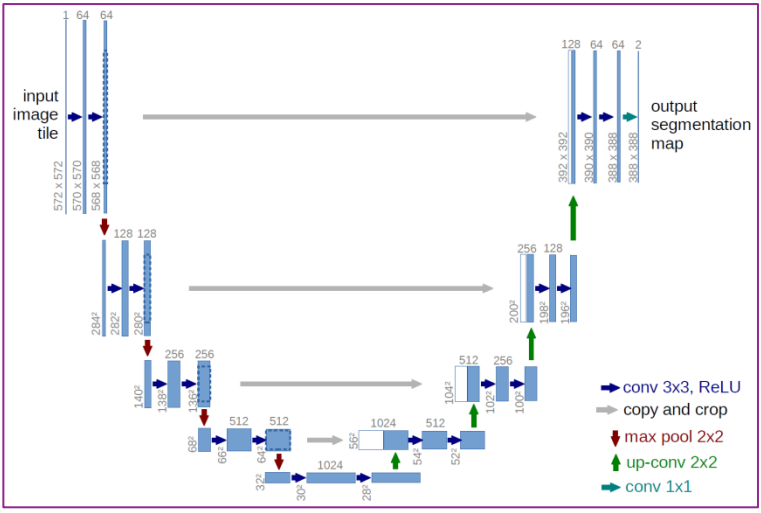
Общие сведения о принципах работы сегментационных сетей

Сегментация — это процесс разбиения исходного изображения на несколько фрагментов. В ходе данной работы выполняется разбиение всего на 2 фрагмента: объект и фон .

Существует большое разнообразие нейронных сетей, предназначенных для разбиения изображения на фрагменты: Unet, SegNet, Enet, FCN, Deeplab и другие.

В данной работе речь пойдет об архитектуре Unet (рис. 2). Своё название данная архитектура берет из-за формы, которую образуют слои нейросети. Изначально она применялась в медицинских задачах, однако в настоящее время нейросеть показала свою эффективность и в других задачах. Например, анализ дорожной обстановки.

Сегментация изображения дорожной обстановки
при помощи сети Unet



Архитектура сети Unet

Принцип работы архитектуры основывается на последовательной свёртке исходного изображения (слои Conv2D и MaxPooling), а затем на обратном расширении пространства признаков (слои Conv2D и UpSampling2D). В итоге нейросеть анализирует изображения как и на самом низком (попиксельном) уровне, так и на самом высоком, при котором анализируется целое изображение. Чем больше совпадений с обучающим набором, тем с большей вероятностью нейросеть корректно выполнит поставленную перед ней задачу.

Практическая часть

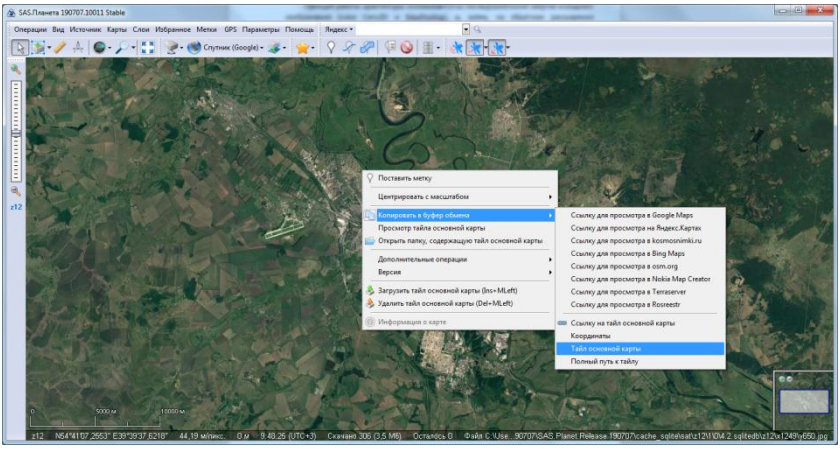
Разметка данных

Загрузка изображений

В данной работе предлагается выполнить сегментацию исходного спутникового снимка на 2 класса (см. варианты в Приложении). Для загрузки исходных изображений предлагается воспользоваться программой SAS.Planet (http://www.sasgis.org/download/).

Наиболее часто для хранения и отображения спутниковых снимков применяется разбиение карты на тайлы. Тайлы — небольшие квадратные изображения (обычно 256*256), отражающие определённые фрагменты на карте. Из таких фрагментов собирается единое полотно.

Выбрав необходимый масштаб, источник карт и необходимую область, нажмите на область карты правой кнопкой мыши и выберите пункт «Копировать в буфер обмена» – «Тайл основной карты». Сохраните полученное изображение. Для корректной работы нейронной сети таких изображений потребуется около 50.



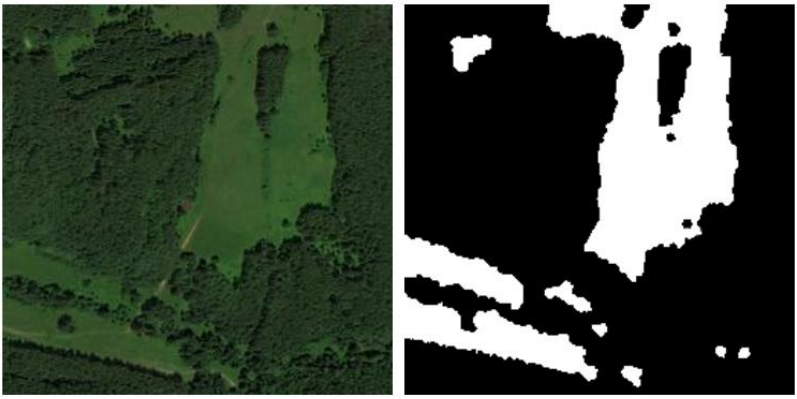
Загрузка тайлов спутникового снимка

Обработка изображений

После того как необходимые изображения получены, можно приступить к их разметке (созданию файлов масок). Для этого рекомендуется использовать графический редактор, например GIMP или Photoshop.

Исходное изображение необходимо представить только двумя цветами: чёрным — области с интересующим классом, белым — все остальные области в которых нет искомого класса. Результат должен быть аналогичен рисунку «Пример размеченного изображения (целевой класс — лесной массив)».

Полученные изображения и их маски сохраните в папки Dataset\IMG и Dataset\MASK. Обе полученные папки запакуйте в архив Dataset.7z и отправьте в среду разработки. Обратите внимание на то, что изображение и его маска должны иметь одно расширение и называться одинаково!



Пример размеченного изображения (целевой класс — лесной массив)

Построение и обучение модели

Подключение необходимых библиотек

Следующий фрагмент кода выполнит подключение всех необходимых для работы библиотек.

```
import os
import cv2
import sys
import random
import warnings

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from tqdm import tqdm
from itertools import chain
from skimage.io import imread, imshow, imre
ad_collection, concatenate_images
```

```
from skimage.transform import resize
from skimage.morphology import label

from keras.models import Model, load_model
from keras.layers import Input
from keras.layers.core import Dropout, Lamb
da
from keras.layers.convolutional import Conv
2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling
2D
from keras.layers.merge import concatenate
from keras.callbacks import EarlyStopping,
ModelCheckpoint
from keras import backend as K
import numpy as np
import os
import skimage.io as io
import skimage.transform as trans
import numpy as np
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint
, LearningRateScheduler
from keras import backend as keras
import tensorflow as tf
```

ФУНКЦИЯ ЗАГРУЗКИ ОБУЧАЮЩЕЙ ВЫБОРКИ

Следующий фрагмент кода выполнит чтение картинок и их масок. Входной размер: 128x128 пикс. На вход подаётся цветная картинка, на выходе ожидается одноканальное изображение – уверенность в нахождении требуемого класса в данной точке.

```
IMG_FOLDER = "/content/IMG"
MASK_FOLDER = "/content/MASK"
SIZE = 128
CLR = cv2.IMREAD_COLOR
GR = cv2.IMREAD_GRAYSCALE

def create_train_data():
  xx = []
  yy = []

  for img in os.listdir(IMG_FOLDER):
        path = os.path.join(IMG_FOLDER, img
)
        img = cv2.resize(cv2.imread(path, C
LR), (SIZE, SIZE))
        xx.append(list(np.array(img)))

  for img in os.listdir(MASK_FOLDER):
        path = os.path.join(MASK_FOLDER, im
g)
        img = cv2.resize(cv2.imread(path, G
R), (SIZE, SIZE))
        img = img/255.0
        yy.append(list(np.array(img)))

  yy = np.expand_dims(yy,3)
  xx=np.array(xx)
```

```
    yy=np.array(yy)
    return (xx,yy)

  (X_train2, Y_train2) = create_train_data()
```

Объявление модели сети

Выполняем построение модели будущей сети. Перед самой моделью объявим метрику, на основе которой будет выполнятся анализ скорости работы. Слои прореживания связей (Dropout) отключены. В случае, если нейросеть не сходится к требуемому результату, их активация иногда может помочь.

```
def mean_iou(y_true, y_pred):
    prec = []
    for t in np.arange(0.5, 1.0, 0.05):
        y_pred_ = tf.to_int32(y_pred > t)
        score, up_opt = tf.metrics.mean_iou(y_
true, y_pred_, 2)
        K.get_session().run(tf.local_variables
_initializer())
        with tf.control_dependencies([up_opt])
:
            score = tf.identity(score)
        prec.append(score)
    return K.mean(K.stack(prec), axis=0)

inputs = Input((128, 128, 3))
s = Lambda(lambda x: x / 255) (inputs)

c1 = Conv2D(16, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (s
)
```

```
c1 = Conv2D(16, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (c
1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(32, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (p
1)
c2 = Conv2D(32, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (c
2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(64, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (p
2)
c3 = Conv2D(64, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (c
3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(128, (3, 3), activation='elu', ker
nel_initializer='he_normal', padding='same') (
p3)
c4 = Conv2D(128, (3, 3), activation='elu', ker
nel_initializer='he_normal', padding='same') (
c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(256, (3, 3), activation='elu', ker
nel_initializer='he_normal', padding='same') (
p4)
```

```
c5 = Conv2D(256, (3, 3), activation='elu', ker
nel_initializer='he_normal', padding='same') (
c5)

u6 = Conv2DTranspose(128, (2, 2), strides=(2,
2), padding='same') (c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(128, (3, 3), activation='elu', ker
nel_initializer='he_normal', padding='same') (
u6)
c6 = Conv2D(128, (3, 3), activation='elu', ker
nel_initializer='he_normal', padding='same') (
c6)

u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2
), padding='same') (c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(64, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (u
7)
c7 = Conv2D(64, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (c
7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2
), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (u
8)
c8 = Conv2D(32, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (c
8)
```

```
u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2
), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(16, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (u
9)
c9 = Conv2D(16, (3, 3), activation='elu', kern
el_initializer='he_normal', padding='same') (c
9)

outputs = Conv2D(1, (1, 1), activation='sigmoi
d') (c9)

model = Model(inputs=[inputs], outputs=[output
s])
model.compile(optimizer='adam', loss='binary_c
rossentropy', metrics=[mean_iou])
model.summary()
```

Обучение сети

После того как были выполнены все подготовительные этапы, можно приступить к обучению сети. В случае, если данные были размечены корректно, точность работы на обучающей выборке должна составлять более 90 %. Если этого не происходит, попробуйте изменить значения batch_size, активировать слои Dropout и ещё раз проверить обучающую выборку на корректность разметки.

```
model.fit(x=X_train2, y=Y_train2,batch_size
=3,epochs=1000)
```

Тестирование качества работы сети

Проверим, как работает нейронная сеть на изображениях, которые не представлены в обучающей выборке.

```
raw = cv2.resize(cv2.imread("SomeFile.png
", CLR), (SIZE, SIZE))
raw = np.expand_dims(raw, 0)
pred = model.predict(raw)*255
pred = pred.squeeze().reshape(128,128)
cv2.imwrite("SomeResults.png", pred)
plt.imshow(pred)
```

Порядок выполнения работы

1. Подготовьте набор изображений по своему варианту задания. Выборка должна содержать не менее 50 изображений. Поместите найденные изображения в папку IMG.
2. Полученный набор изображений необходимо разметить на 2 класса согласно варианту задания. Области, относящиеся к классу 1 залейте белым цветом, области другого класса – чёрным.
3. Размеченные изображения сохраняйте под теми же именами, но в папку MASK.
4. Полученную обучающую выборку отправьте в рабочий каталог среды.
5. Выполните указанный выше программный код.
6. Протестируйте работу сети на тестовой выборке.
7. Подведите итоги работы. Оформите результаты в виде отчёта.

Подготовка отчёта

Отчёт о лабораторной работе должен содержать:

1. Титульный лист.

2. Цели и задачи данной работы.
3. Задание, согласно своему варианту.
4. Содержание обучающей выборки (несколько картинок).
5. Программный код.
6. Результаты обучения: графики точности на тестовой и обучающей выборках.

Приложение

Варианты заданий

Вариант задания	Описание	Масштаб снимка	Размер изображения
1	На фрагменте спутникового снимка определить расположение лесного массива	z16	128*128 RGB
2	На фрагменте спутникового снимка определить расположение аэропортов	z13	128*128 RGB
3	На фрагменте спутникового снимка определить расположение грунтовых и асфальтовых дорог	z16	128*128 RGB
4	На фрагменте спутникового снимка определить расположение зелёных зон в городе	z16	128*128 RGB
5	На фрагменте спутникового снимка	z14	128*128 RGB

	определить наличие построек (дома, дачи, садовые постройки и др.)		

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое сегментационные нейронные сети?
2. В чём особенность работы архитектуры UNet?
3. Какие ещё методы сегментации изображений (за исключением нейронных сетей) вам известны?

Лабораторная работа № 5

Использование нейронных сетей в задачах поиска объекта на изображении

Цели работы: изучить особенности детектирования объектов с использованием нейронных сетей. Научиться работать с нейронными сетями типа SSD(Single Shot MultiBox)

Задачи: подготовить данные для обучения нейронной сети; сформировать модель, обучить сеть на размеченной выборке и протестировать её с использованием OpenCV в одном из языков программирования.

Теоретическая часть

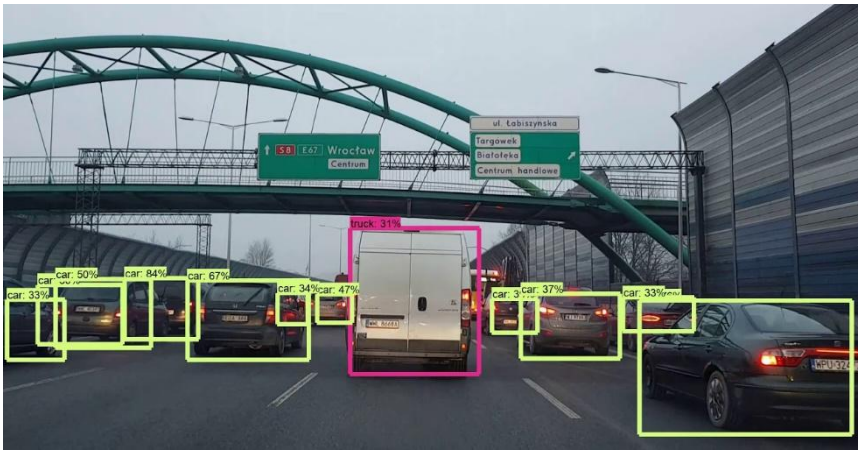
Особенности архитектуры сетей Single Shot MultiBox Detector

В данной работе пойдет речь о нейронных сетях типа SSD (Single Shot MultiBox Detector). Такие сети позволяют обнаружить и локализовать объекты на изображениях. Детектор SSD основан на идее детектора MultiBox, которая заключается в предсказании ограничивающих прямоугольников в разных масштабах. Якоря для SSD выбираются вручную, что позволяет избежать этапа предобучения якорей и позволяет алгоритму работать при любых входных данных. Многомасштабные карты признаков позволяют существенно увеличить точность обнаружения и распознавания объектов, что является большим преимуществом по сравнению с другими подходами.

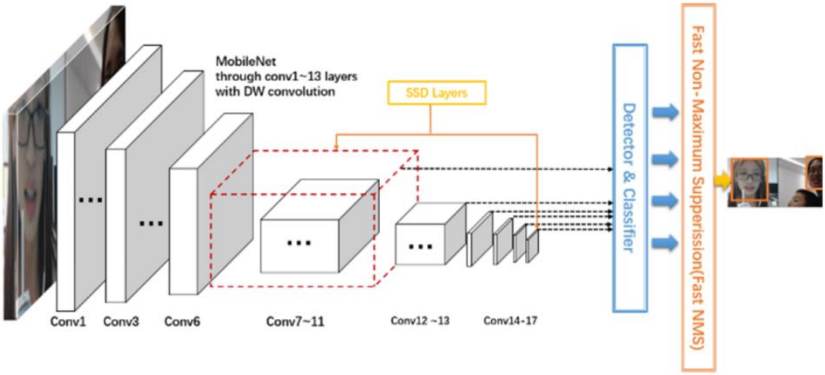
В ходе данной лабораторной работы будет использоваться архитектура MobileNet v2 SSD.

Даная архитектура основана на MobileNet v2, однако в случае SSD, из каждого свёрточного блока идут «нити», называемые Extra

Feature Layers, в которых и содержатся полезные сведения о расположении объектов в кадре.



Пример работы MobileNet SSD. Детектирование автомобилей



Архитектура MobileNet v2 SSD

Практическая часть

Подготовка данных

Так как работа в дальнейшем пойдёт с TensorFlow, то входные данные должны быть представлены одним файлом в формате **record**. Получить такой файл можно несколькими способами. Самый простой – на основе изображений и файла CSV, который будет содержать сведения о разметке.

Для получения такой разметки предлагается использовать программные средства, входящие в состав материалов в лабораторным работам (каталог Lab5)

- **Lab5.1** – программа для разметки изображений, получаемых с камеры компьютера или ноутбука.
- **Lab5.2** – программа для разметки произвольных изображений, полученных из сети Интернет.

Для работы желательно разметить не менее 100 изображений одного класса.

> *MobileNet SSD позволяет одновременно определять несколько объектов различных классов на одном изображении. В этом случае, обучающая выборка должна содержать разметку как для одного класса, так и для другого. Переключить ID размечаемого класса в программе можно используя клавиши 1..9.*

Размеченные данные представляют собой каталог **images** и файл **dataset.csv**. Эти файлы потребуются для создания архива **train.record** – обучающей выборки нейросети.

Для создания архива train.record в материалах к лабораторной работе представлен скрипт generate_tfrecord.py. В него передаётся 3 параметра: **csv_input** – путь к файлу CSV, **output_path** – имя сохраняемого файла, **image_dir** – путь к папке с изображениями.

```
python generate_tfrecord.py --csv_input=data/dataset.csv --output_path=train.record –image_dir=data/images
```

Также подготовьте 10-15 размеченных изображений под тестовую выборку. Из них сформируйте архив **test.record**.

Таким образом должны быть сформированы 2 файла: **train.record** и **test.record** с которыми в дальнейшем будет идти работа.

Обучение нейронной сети

Поскольку обучение Mobile Net SSD с использованием TensorFlow 2.0 и старше невозможно, потребуется установить более старую версию TensorFlow (например, 1.14.0), а также ряд библиотек: tf_slim и lvis.

> *Данную часть работы рекомендуется выполнять на платформе Google Colab. Это позволит избежать манипуляций с версиями платформы на рабочем ПК.*

```
!pip install tensorflow==1.14.0
!pip install tf_slim lvis
```

Кроме того, для работы потребуются дополнительные компоненты из состава репозитория TensorFlow и OpenCV. Загрузить их можно при помощи следующих команд (Далее программный код применим только для Google Colab)

```
!git clone https://github.com/tensorflow/models.git
!git clone https://github.com/opencv/opencv.git
```

Также потребуется получить предобученную нейронную сеть. Далее работа пойдёт с MobileNet SSD v2:

```
%cd /content
!pip install wget
import wget, os
wget.download('http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz', '/content/neural.tar.gz')
!tar -xvzf "neural.tar.gz"
```

```
!mv /content/ssd_mobilenet_v2_coco_2018_03_29 /cont
ent/pretrained_model
```

Добавим в переменные среды ряд каталогов:

```
%cd /content/models/research/
!protoc object_detection/protos/*.proto --
python_out=.
os.environ['PYTHONPATH'] += ':/content/models/resea
rch/:/content/models/research/slim/:/content/models
/research/object_detection/utils/:/content/models/r
esearch/object_detection'
```

Загрузите в каталог /content все необходимые файлы:

Файл	**Описание**
train.record	Архив с обучающей выборкой
test.record	Архив с тестовой выборкой
object_detection.pbtxt	Файл с описанием детектируемых классов
ssd_mobilenet_v2_coco.config	Конфигурационный файл с гиперпараметрами нейронной сети

Запустите обучение. Оно может потребовать значительное время (порядка 30 минут).

```
%cd /content/
!python /content/models/research/object_detection/m
odel_main.py \
    --
pipeline_config_path=/content/ssd_mobilenet_v2_coco
.config --model_dir=/content/trainer \
    --alsologtostderr \
    --num_train_steps=4000 \
    --num_eval_steps=300
```

После окончания обучения в каталоге trainer будет сформирована обученная нейронная сеть. Для того, чтобы с ней в дальнейшем можно было производить работу, её необходимо экспортировать. Для этого, воспользуйтесь следующей командой:

```
%cd /content

lst = os.listdir('trainer')
lf = filter(lambda k: 'model.ckpt-' in k, lst)
last_model = sorted(lf)[-1].replace('.meta', '')
print(last_model)
!python /content/models/research/object_detection/e
xport_inference_graph.py \
    --input_type=image_tensor \
    --
pipeline_config_path=/content/ssd_mobilenet_v2_coco
.config \
    --output_directory=fine_tuned_model \
    --trained_checkpoint_prefix=trainer/$last_model

%cd /content/opencv/samples/dnn
from tf_text_graph_ssd import *

createSSDGraph("/content/fine_tuned_model/frozen_in
ference_graph.pb", "/content/ssd_mobilenet_v2_coco.
config", "/content/result.pbtxt",)
```

В результате выполнения данного программного кода будет сформировано 2 файла: **frozen_inference_graph.pb** и **result.pbtxt**.

Использование в OpenCV

Для того, чтобы проверить работоспособность сети воспользуетесь утилитой из материалов к лабораторной работе. Для

проверки корректности работы не обязательно собирать проект, достаточно будет положить файлы **`frozen_inference_graph.pb`** и **`result.pbtxt`** в каталог с программой.

Порядок выполнения работы

1. Подготовить выборку по своему варианту задания. Выборка должна содержать не менее 50 размеченных изображений;
2. Загрузить выборку в среду Google Colab и распаковать её;
3. Загрузить предобученные модели сетей;
4. Выполнить обучение на полученной выборке. Оценить результаты точности работы;
5. Разработать приложение на любом языке программирования для оценки точности работы алгоритма;
6. Подвести итоги работы. Оформить результаты в виде отчёта.

Подготовка отчёта

Отчёт по лабораторной работе должен содержать:

1. Титульный лист
2. Цели и задачи данной работы
3. Задание, согласно своего варианта
4. Результаты обучения: графики точности на тестовой и обучающей выборках
5. Результат работы программы

Приложения

Приложение 1. Варианты заданий

Вариант задания	Описание
1	Поиск автомобильных номеров в кадре
2	Поиск дорожных знаков в кадре

3	Наличие в кадре дисконтной карты любого (на выбор) магазина
4	Поиск любого (на выбор) животного в кадре
5	Поиск человека в кадре

Контрольные вопросы

1. В чём особенность архитектуры MobileNet SSD?
2. Для каких задач может быть применена данная архитектура?

Информационные ресурсы

Список литературы

1. Николенко С., Кадурин Е., Архангельская А. Глубокое обучение. Погружение в мир нейронных сетей. СПб.:Питер, 2018.
   ISBN: 978-5-496-02536-2

2. Бринк Хенрик, Ричардс Джозеф. Машинное обучение. СПб.:Питер, 2017.
   ISBN: 978-5-496-02989-6

3. Шакла Н. Машинное обучение и TensorFlow. СПб.:Питер. 2019.
   ISBN: 978-5-4461-0826-8

Список Интернет-ресурсов

1. Дополнительную информацию об активационных функциях вы сможете получить на странице: https://keras.io/activations/

2. Подробная документация по библиотеке Keras: http://faroit.com/keras-docs/1.2.2/getting-started/sequential-model-guide/
3. Описание того, как работает операция свёртки в сетях: https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/
4. Подробное описание того, как работают свёрточные сети: https://habr.com/ru/post/309508/
5. Подробная документация по библиотеке Keras: http://faroit.com/keras-docs/1.2.2/getting-started/sequential-model-guide/

МАТЕРИАЛЫ К ЛАБОРАТОРНЫМ РАБОТАМ

Ссылка на ресурсы к лабораторной работе:
https://yadi.sk/d/ntJ74G_KogpjHg

Лабораторные работы предназначены для студентов, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника» по предмету «Машинное обучение»

Авторы курса: Муратов Е.Р., Тарасов А.С.,
Рязань, 2020