

0 Установка и импорт библиотеки

1. Для установки библиотеки необходимо в терминале ввести следующую команду:

```
pip install widelearning
```

Также необходимо установить дополнительные библиотеки:

```
pip install numpy
```

```
pip install pandas
```

```
pip install requests
```

Для сверточного слоя необходимо установить библиотеку TensorFlow:

```
pip install tensorflow
```

Дальнейшие действия выполнены в Jupyter Notebook. Устанавливается с помощью команды:

```
pip install jupyter
```

Запуск Jupyter Notebook выполняется с помощью команды:

```
jupyter notebook
```

2. Импорт библиотеки:

```
import widelearning as wdl
```

1 Полносвязный слой

1. Подготовка набора данных (датасета).

Для примера использован набор данных User Knowledge Modeling Data Set, который можно скачать по адресу:

<https://archive.ics.uci.edu/ml/datasets/User+Knowledge+Modeling>

Открытый в MS Excel файл выглядит следующим образом:

STG	SCG	STR	LPR	PEG	UNS	Attribute Information:			
0	0	0	0	0	very_low				
0.08	0.08	0.1	0.24	0.9	High				
0.06	0.06	0.05	0.25	0.33	Low				
0.1	0.1	0.15	0.65	0.3	Middle				
0.08	0.08	0.08	0.98	0.24	Low				
0.09	0.15	0.4	0.1	0.66	Middle				
0.1	0.1	0.43	0.29	0.56	Middle				
0.15	0.02	0.34	0.4	0.01	very_low				
0.2	0.14	0.35	0.72	0.25	Low				
0	0	0.5	0.2	0.85	High				
0.18	0.18	0.55	0.3	0.81	High				
0.06	0.06	0.51	0.41	0.3	Low				
0.1	0.1	0.52	0.78	0.34	Middle				
0.1	0.1	0.7	0.15	0.9	High				
0.2	0.2	0.7	0.3	0.6	Middle				
0.12	0.12	0.75	0.35	0.8	High				
0.05	0.07	0.7	0.01	0.05	very_low				
0.1	0.25	0.1	0.08	0.33	Low				
0.15	0.32	0.05	0.27	0.29	Low				

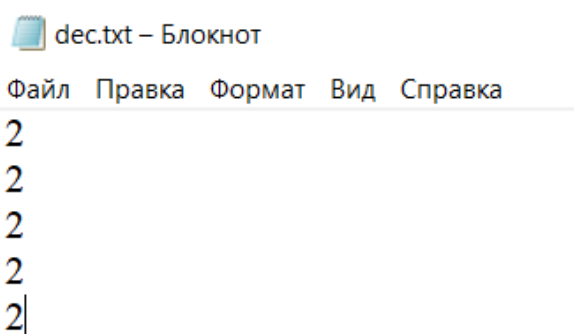
Скопируем часть, содержащую данные, в новый файл формата csv и переместим столбец UNS (состоит из меток принадлежности к классам) в начало. Заменяем разделитель целой и дробной части вместо запятой «,» на точку «.». Также необходимо разделитель столбцов поменять с «;» на «,». Иногда данные в csv файле могут быть после преобразований заключены в двойные кавычки. Необходимо также удалить из итогового файла этот знак.

Получившийся файл, открытый в текстовом редакторе:

```
UNS,STG,SCG,STR,LPR,PEG
very_low,0,0,0,0,0
High,0.08,0.08,0.1,0.24,0.9
Low,0.06,0.06,0.05,0.25,0.33
Middle,0.1,0.1,0.15,0.65,0.3
Low,0.08,0.08,0.08,0.98,0.24
Middle,0.09,0.15,0.4,0.1,0.66
Middle,0.1,0.1,0.43,0.29,0.56
very_low,0.15,0.02,0.34,0.4,0.01
Low,0.2,0.14,0.35,0.72,0.25
High,0,0,0.5,0.2,0.85
High,0.18,0.18,0.55,0.3,0.81
Low,0.06,0.06,0.51,0.41,0.3
Middle,0.1,0.1,0.52,0.78,0.34
High,0.1,0.1,0.7,0.15,0.9
Middle,0.2,0.2,0.7,0.3,0.6
High,0.12,0.12,0.75,0.35,0.8
very_low,0.05,0.07,0.7,0.01,0.05
Low,0.1,0.25,0.1,0.08,0.33
Low,0.15,0.32,0.05,0.27,0.29
```

2. Преобразование в целочисленный вид

Для выполнения вычислений преобразуем с помощью функции `data_int` представленные в вещественнозначном виде данные в целочисленный. Создадим файл `dec.txt`, который содержит количество знаков после запятой для каждого столбца файла `csv`. Значения количества знаков в файле `txt` идут на новой строке для каждого следующего столбца, то есть в нашем случае имеется 5 столбцов, для каждого столбца количество знаков равно 2. В итоге в файле `dec.txt` 5 строк со значением 2:





Первый параметр функции – путь к подготовленной выборке в формате `csv`, второй параметр – файл со значениями количества знаков после запятой для каждого столбца, третий параметр – желаемое название выходных преобразованных в целочисленный вид файлов.

```
wdl.data_int('kahraman.csv', 'dec.txt', 'KAHRAMAN')
```

	UNS	STG	SCG	STR	LPR	PEG	N
0	very_low	-49	-45	-47	-49	-46	0
1	High	-41	-37	-37	-25	44	1
2	Low	-43	-39	-42	-24	-13	2
3	Middle	-39	-35	-32	16	-16	3
4	Low	-41	-37	-39	49	-22	4
...
253	High	12	33	22	43	11	253
254	Middle	29	16	24	-30	14	254
255	High	5	37	24	-20	31	255
256	Middle	1	30	34	12	-20	256
257	High	17	45	29	38	28	257

В результате применения функции `data_int` получим целочисленные обучающую и тестовую выборки.

 KAHRAMAN_test.csv
 KAHRAMAN_train.csv

3. Первоначальное приближение первого нейрона

Используем функцию `select_top` для получения первоначального приближения первого нейрона:

```
wdl.select_top('KAHRAMAN_train.csv', 'UNS')
```

Здесь первый параметр – путь к обучающей выборке в целочисленном виде, второй параметр – название столбца с метками классов.

Вывод функции выглядит следующим образом:

```
Количество отсеженных сверху = 9
Количество НЕотсеженных сверху = 14
Количество отсеженных снизу = 8
Количество НЕотсеженных снизу = 201
===Нижняя категория: High
СУММА отсеженных сверху и снизу = 17
```

```

TOP - ['very_low']
OTHERS - ['High', 'Low', 'Middle']
+++++
Количество отсеченных сверху = 10
Количество НЕотсеченных сверху = 44
Количество отсеченных снизу = 15
Количество НЕотсеченных снизу = 163
===Нижняя категория: very_low
СУММА отсеченных сверху и снизу = 25
TOP - ['High']
OTHERS - ['very_low', 'Low', 'Middle']
+++++
Количество отсеченных сверху = 4
Количество НЕотсеченных сверху = 71
Количество отсеченных снизу = 7
Количество НЕотсеченных снизу = 150
===Нижняя категория: High
СУММА отсеченных сверху и снизу = 11
TOP - ['Low']
OTHERS - ['very_low', 'High', 'Middle']
+++++
Количество отсеченных сверху = 0
Количество НЕотсеченных сверху = 80
Количество отсеченных снизу = 7
Количество НЕотсеченных снизу = 145
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 7
TOP - ['Middle']
OTHERS - ['very_low', 'High', 'Low']
+++++

```

Переменная TOP обозначает целевой класс и соответственно, «Нижняя категория» - это противоположный класс, участвующий в отсечении.

Были также сгенерированы папки:

```

train_['High']['very_low', 'Low', 'Middle']
train_['Low']['very_low', 'High', 'Middle']
train_['Middle']['very_low', 'High', 'Low']
train_['very_low']['High', 'Low', 'Middle']

```


Необходимо выбрать целевой класс. Выбор можно производить по разным критериям:


- по сумме отсеченных сверху и снизу;
- по количеству отсеченных сверху;
- по количеству отсеченных снизу и так далее.


От выбора целевого класса зависит итоговая структура нейронной сети. В нашем случае целевой класс будет выбираться по сумме отсеченных экземпляров сверху и снизу. Таким образом, в качестве целевого на первом нейроне выступает класс «High», поскольку сумма отсеченных равна 25. Если мы хотим оставить полученные результаты без проведения градиентного уточнения вектора весов, то необходимо открыть соответствующую папку и скопировать в отдельный каталог файлы w.txt и train.csv, которые являются вектором весов первого нейрона и сокращенной выборкой для второго нейрона:


```
> KAHARAMAN > train_['High']['very_low', 'Low', 'Middle']
```

Имя

 class_top_['High']['very_low', 'Low', 'Middl...

 other_classes_['High']['very_low', 'Low', 'M...

 train_['High']['very_low', 'Low', 'Middle'].csv

 w_['High']['very_low', 'Low', 'Middle'].txt

4. Градиентное уточнение первого нейрона

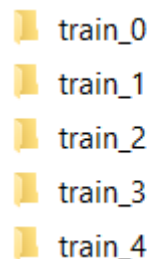
С помощью градиентного уточнения можно увеличить количество отсеченных экземпляров. Функция `grad` изменяет элементы вектора весов после первоначального приближения с некоторым шагом и фиксирует количество отсеченных. В приоритете стоит количество отсеченных сверху (требуется увеличение количества), далее количество отсеченных снизу (требуется увеличение количества), на последнем месте расстояние между границами отсечения. Если количество отсеченных сверху и снизу не изменяется, то элемент вектора изменяется, пока расстояние между границами уменьшается.

Первый параметр функции `grad` – путь к обучающей выборке, которая использовалась в первоначальном приближении, второй – целевой класс, третий – список с остальными классами, четвертый – название столбца с метками классов, пятый – шаг градиентного уточнения, шестой – вектор

весов после первоначального приближения. Вектор весов берется из соответствующей целевому классу папки первоначального приближения (файл w.txt).

```
wdl.grad('KAHRAMAN_train.csv', ['High'], ['very_low', 'Low', 'Middle'], 'UNS', 1,  
[455.02247191011236, 552.8764044943821, 273.38202247191015,  
518.2921348314607, 2276.179775280899]).
```

В результате градиентного уточнения сгенерировались папки train_{i}.csv.







Необходимо проверить каждую из них и выбрать ту папку, в которой в файле txt во второй строке (количество отсеченных сверху) наибольшее значение. Если значения одинаковые, то необходимо проверить третью строку (количество отсеченных снизу) и выбрать ту папку, в которой данное значение наибольшее. В случае одинаковых значений сверху и снизу, необходимо выбирать папку по последней строке файла txt, которое обозначает расстояние между границами, оно должно быть минимальное.

Таким образом, в нашем случае все три значения для всех папок одинаковые, поэтому выбирается любая. Выберем папку под номером 0. Необходимо скопировать отдельно файл train.csv и w.txt. Первый файл является сокращенной обучающей выборкой для второго нейрона, второй файл – является вектором весов первого нейрона.

> KAHRAMAN > train_0

Имя

 class_top_0.csv
 other_classes_0.csv
 train_0.csv
 w_0.txt

После градиентного уточнения количество отсеченных экземпляров увеличилось: сверху до 49, снизу до 10.

5. Первоначальное приближение второго нейрона

Выбирается сокращенная выборка после градиентного уточнения.

```
wdl.select_top('train_2.csv', 'UNS')
```

```
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 4
Количество отсеченных снизу = 3
Количество НЕотсеченных снизу = 165
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 4
TOP - ['High']
OTHERS - ['Middle', 'Low', 'very_low']
+++++
Количество отсеченных сверху = 5
Количество НЕотсеченных сверху = 75
Количество отсеченных снизу = 6
Количество НЕотсеченных снизу = 87
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 11
TOP - ['Middle']
OTHERS - ['High', 'Low', 'very_low']
+++++
Количество отсеченных сверху = 7
Количество НЕотсеченных сверху = 68
Количество отсеченных снизу = 4
Количество НЕотсеченных снизу = 94
===Нижняя категория: Middle
СУММА отсеченных сверху и снизу = 11
TOP - ['Low']
OTHERS - ['High', 'Middle', 'very_low']
+++++
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 12
Количество отсеченных снизу = 1
Количество НЕотсеченных снизу = 159
===Нижняя категория: Middle
СУММА отсеченных сверху и снизу = 2
```



```
TOP - ['very_low']
OTHERS - ['High', 'Middle', 'Low']
+++++
```

Имеется два класса с количеством отсеченных экземпляров сверху и снизу, равным 11. Выберем в качестве целевого класс «Low».

6. Градиентное уточнение второго нейрона

Из соответствующей папки, в названии которой в первом списке «Low», копируем вектор весов.

```
wdl.grad('train_2.csv', ['Low'], ['High', 'Middle', 'very_low'], 'UNS', 1, [-
540.3163265306123, -194.17346938775518, -400.48979591836735,
1046.591836734694, -1941.0408163265306])
```

Выбираем из папок train номер папки, в которой в файле txt наибольшее количество отсеченных сверху, иначе наибольшее количество отсеченных снизу, иначе наименьшее расстояние между границами отсеечения. Во всех папках одинаковые значения, выберем папку 1. Количество отсеченных сверху стало 10, снизу 7. Скопируем в отдельный каталог файл train.csv и w.txt (сокращенная обучающая выборка третьего нейрона, вектор весов второго нейрона). Удалим сгенерированные папки.

7. Первоначальное приближение третьего нейрона

Загрузим соответствующую сокращенную выборку.

```
wdl.select_top('train_3.csv', 'UNS')
```

```
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 64
Количество отсеченных снизу = 1
Количество НЕотсеченных снизу = 90
===Нижняя категория: High
СУММА отсеченных сверху и снизу = 2
TOP - ['Low']
OTHERS - ['very_low', 'Middle', 'High']
+++++
Количество отсеченных сверху = 4
Количество НЕотсеченных сверху = 9
Количество отсеченных снизу = 1
Количество НЕотсеченных снизу = 142
===Нижняя категория: Middle
СУММА отсеченных сверху и снизу = 5
```

```

TOP - ['very_low']
OTHERS - ['Low', 'Middle', 'High']
+++++
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 72
Количество отсеченных снизу = 2
Количество НЕотсеченных снизу = 81
===Нижняя категория: very_low
СУММА отсеченных сверху и снизу = 3
TOP - ['Middle']
OTHERS - ['Low', 'very_low', 'High']
+++++
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 4
Количество отсеченных снизу = 2
Количество НЕотсеченных снизу = 149
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 3
TOP - ['High']
OTHERS - ['Low', 'very_low', 'Middle']
+++++

```

Наибольшая сумма отсеченных равна 5 для целевого класса «very_low».

8. Градиентное уточнение третьего нейрона

```

wdl.grad('train_3.csv', ['very_low'], ['Low', 'Middle', 'High'], 'UNS', 1,
[47.81818181818181, -182.63636363636363, -141.54545454545456,
91.72727272727275, -387.27272727272725])

```

В папке под номером 0 расстояние минимальное и равно 7.34. Из данной папки копируются в отдельный каталог файлы train.csv и w.txt (обучающая выборка для четвертого нейрона и вектор весов третьего нейрона). Сгенерированные папки необходимо удалить.

9. Первоначальное приближение четвертого нейрона

В качестве целевого выбирается класс «High», поскольку сумма отсеченных наибольшая и равна 4.

```

wdl.select_top('train_4.csv', 'UNS')

```

```

Количество отсеченных сверху = 0
Количество НЕотсеченных сверху = 7
Количество отсеченных снизу = 1

```

```

Количество НЕотсеченных снизу = 141
===Нижняя категория: High
СУММА отсеченных сверху и снизу = 1
TOP - ['very_low']
OTHERS - ['Low', 'Middle', 'High']
+++++
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 64
Количество отсеченных снизу = 2
Количество НЕотсеченных снизу = 82
===Нижняя категория: High
СУММА отсеченных сверху и снизу = 3
TOP - ['Low']
OTHERS - ['very_low', 'Middle', 'High']
+++++
Количество отсеченных сверху = 0
Количество НЕотсеченных сверху = 72
Количество отсеченных снизу = 1
Количество НЕотсеченных снизу = 76
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 1
TOP - ['Middle']
OTHERS - ['very_low', 'Low', 'High']
+++++
Количество отсеченных сверху = 2
Количество НЕотсеченных сверху = 3
Количество отсеченных снизу = 2
Количество НЕотсеченных снизу = 142
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 4
TOP - ['High']
OTHERS - ['very_low', 'Low', 'Middle']
+++++

```

10. Градиентное уточнение четвертого нейрона

```

wdl.grad('train_4.csv', ['High'], ['very_low', 'Low', 'Middle'], 'UNS', 1, [-63.5625,
75.15972222222223, -44.30555555555556, -83.38888888888889,
196.91666666666666])

```

Вектор весов и сокращенная обучающая выборка копируются из папки №3. Количество отсеченных сверху 5, снизу – 2, расстояние – 751.9. Класс «High» полностью отсекается.

Удаляются сгенерированные папки.

11. Первоначальное приближение пятого нейрона

```

wdl.select_top('train_5.csv', 'UNS')

```

```

Количество отсеченных сверху = 52
Количество НЕотсеченных сверху = 20
Количество отсеченных снизу = 1
Количество НЕотсеченных снизу = 69
===Нижняя категория: Low
СУММА отсеченных сверху и снизу = 53
TOP - ['Middle']
OTHERS - ['Low', 'very_low']
+++++
Количество отсеченных сверху = 1
Количество НЕотсеченных сверху = 62
Количество отсеченных снизу = 60
Количество НЕотсеченных снизу = 19
===Нижняя категория: Middle
СУММА отсеченных сверху и снизу = 61
TOP - ['Low']
OTHERS - ['Middle', 'very_low']
+++++
Количество отсеченных сверху = 0
Количество НЕотсеченных сверху = 7
Количество отсеченных снизу = 54
Количество НЕотсеченных снизу = 81
===Нижняя категория: Middle
СУММА отсеченных сверху и снизу = 54
TOP - ['very_low']
OTHERS - ['Middle', 'Low']
+++++

```

Целевым выбирается класс «Middle», поскольку несмотря на сумму отсеченных сверху отсекается только один экземпляр. Предпочтительнее выбрать вариант, в котором сверху отсекается большее количество экземпляров.

12. Градиентное уточнение пятого нейрона

```

wdl.grad('train_5.csv', ['Middle'], ['Low', 'very_low'], 'UNS', 1,
[277.4000000000001, 201.71428571428555, 150.99999999999994, -
709.5714285714286, 2071.085714285714])

```

В результате класс «very_low» полностью отсекается.

Выберем папку под номером 0. Скопируем в отдельный каталог выборку для шестого нейрона train.csv и вектор весов пятого нейрона w.txt. Удалим сгенерированные папки.

13. Первоначальное приближение шестого нейрона

В обучающей выборке осталось только два класса. Количество экземпляров выборки равно 64. Необходимо использовать бинарные варианты функций с меткой «binary».

```
wdl.select_top_binary('train_6.csv', 'UNS')
```

```
Количество отсеченных сверху = 18
Количество отсеченных снизу = 45
===Верхняя категория: Low
===Нижняя категория: Low
TOP - ['Middle']
OTHER - ['Low']
СУММА отсеченных сверху и снизу = 63
[-929.5079365079366, -696.952380952381, -46.03174603174604, -
108.99999999999997, -1315.1269841269843]
DISTANCE = 342.9047619047633
+++++

Количество отсеченных сверху = 57
Количество отсеченных снизу = 6
===Верхняя категория: Low
===Нижняя категория: Low
TOP - ['Low']
OTHER - ['Middle']
СУММА отсеченных сверху и снизу = 63
[1009.0, 48.0, -929.0, -1056.0, -13.0]
DISTANCE = 205.0
+++++
```

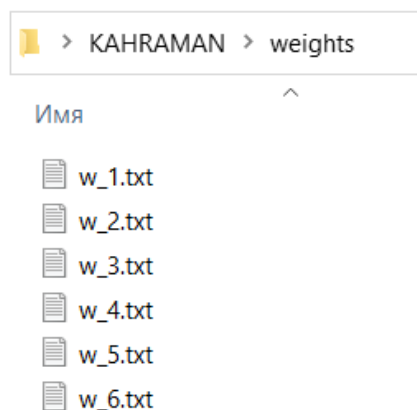
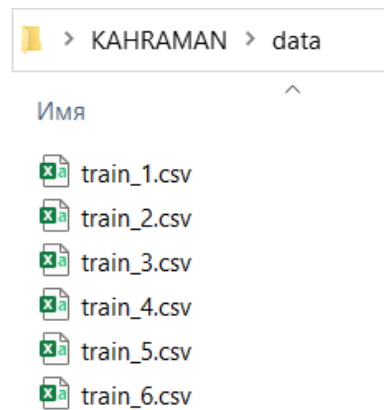
Выберем в качестве целевого класс «Low». Он полностью отсекается. В выборке остался только один экземпляр класса «Middle». Градиентное уточнение в данном случае уже не нужно.

Необходимо скопировать вектор весов из соответствующего файла:

```
w_['Low']['Middle'].txt
w_['Middle']['Low'].txt
```

14. Проверка результатов

В папку data переместим обучающие выборки для каждого нейрона. В папку weights соответствующие вектора весов. Принцип наименования должен соответствовать следующему виду:



Полученные веса сначала необходимо масштабировать в диапазон от -1 до 1 с помощью функции `scale_weights`, в которой первый параметр – это название столбца с метками классов, второй параметр – путь к папке, содержащей обучающие выборки, третий параметр – путь к папке, содержащей веса для каждого нейрона.

```
wdl.scale_weights('UNS', 'data/', 'weights/')
```

В результате дополнительно сгенерировались папки «scale», «result» и «all_weights». Папка «all_weights» содержит специальный текстовый файл, содержащий отмасштабированные веса всех нейронов. Данный файл необходим для функции `check_test`. Функция `check_test` позволяет проверить точность вычислений полученной структуры нейронной сети.

Первый параметр – путь к обучающей выборке, второй параметр – путь к текстовому файлу `weights.txt`, третий параметр – путь к тестовой выборке, четвертый параметр – название столбца с метками классов.

```
wdl.check_test('KAHRAMAN_train.csv', 'all_weights/weights.txt', 'KAHRAMAN_test.csv', 'UNS')
```

Таким образом имеется две ошибки на тестовой выборке (экземпляры № 3 и 160).

```
Ошибок = 2 из 26  
Экземпляры №: ['3', '160']
```

Если третьим параметром передать путь к обучающей выборке, то будет определена точность по обучающей выборке:

```
wdl.check_test('KAHRAMAN_train.csv', 'all_weights/weights.txt', 'KAHRAMAN_train.csv', 'UNS')
```

```
Ошибок = 0 из 232  
Экземпляры №: []
```

Для генерации функции логического вывода можно применить функцию `generate_fa`:

```
wdl.generate_fa('UNS', 'result', 'scale')
```

```
for n in range(len(data)):
    if(data["SC0"][n]>=1):
        data["FA"][n]="High"
    elif(data["SC0"][n]<=-1):
        data["FA"][n]="very_low"
    elif(data["SC1"][n]>=1):
        data["FA"][n]="Low"
    elif(data["SC1"][n]<=-1):
        data["FA"][n]="Middle"
    elif(data["SC2"][n]>=1):
        data["FA"][n]="very_low"
    elif(data["SC2"][n]<=-1):
        data["FA"][n]="Middle"
    elif(data["SC3"][n]>=1):
        data["FA"][n]="High"
    elif(data["SC3"][n]<=-1):
        data["FA"][n]="Low"
    elif(data["SC4"][n]>=1):
        data["FA"][n]="Middle"
    elif(data["SC4"][n]<=-1):
        data["FA"][n]="very_low"
    elif(data["SC5"][n]>=1):
        data["FA"][n]="Low"
    elif(data["SC5"][n]<=-1):
        data["FA"][n]="Low"
```

Здесь первый параметр – название столбца с метками классов, второй параметр – путь к папке `result`, содержащей вычисления скалярного

произведения для каждой из обучающих выборок, третий параметр – путь к папке `scale` – в которой содержатся в отдельных файлах отмасштабированные веса каждого нейрона.

2 Сверточный слой

Для примера работы использован набор данных MNIST и структура нейронной сети LeNet-5:

https://github.com/SlinkoIgor/Neural_Networks_and_CV/blob/master/module05_mnist_conv.ipynb

По ссылке используется библиотека Pytorch, код был преобразован для использования в библиотеке TensorFlow. Структура была сокращена до одного сверточного слоя с 4 ядрами и одного полносвязного слоя с 10 нейронами. Для работы необходимо установить библиотеки TensorFlow, matplotlib:

```
pip install tensorflow
```

```
pip install matplotlib
```

В LeNet-5 в первом сверточном слое используется 6 сверточных ядер 3×3 с шагом 1. Число вычислительных операций для слоя равно:

```
wdl.count_conv_operations(28, 28, 1, 3, 1, 6)
```

```
Кол-во вычислительных операций = 73008
```

```
Кол-во вычислительных операций (no_bias) = 72954
```

Первые два параметра функции — это высота и ширина исходного изображения, третий параметр — количество цветовых каналов, четвертый параметр — размерность сверточного ядра, пятый — шаг сдвига сверточного ядра по изображению, шестой — количество сверточных ядер.

Теперь сравним ресурсоемкость вычислений при использовании четырех сверточных ядер размерности 14×14 с шагом 7:

```
wdl.count_conv_operations(28, 28, 1, 14, 7, 4)
```

```
Кол-во вычислительных операций = 14112
```

```
Кол-во вычислительных операций (no_bias) = 13328
```

Таким образом, это снизит количество вычислительных операций.

Зададим структуру модели следующим образом, отключим обучение сверточного слоя:

```
from keras import models
from keras import layers

model = models.Sequential([
    layers.Conv2D(4, (14,14), activation='relu', strides=(7, 7), input_shape=(28,28,1), use_bias=False, trainable=False),
    layers.Flatten(),
    layers.Dense(10, 'softmax')
])
```

Сгенерируем горизонтальное ядро размера 14*14, состоящее из значений диапазона от -7 до 7 с шагом в 2:

```
wdl.horizontal(14, [-7,-5,-3,-1,1,3,5,7])
```

```
[[-7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7],  
 [-5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5],  
 [-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3],  
 [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
 [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],  
 [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5],  
 [7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7],  
 [-7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7],  
 [-5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5],  
 [-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3],  
 [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
 [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]]
```

Аналогичным образом сгенерируем вертикальное ядро:

```
wdl.vertical(14, [-7,-5,-3,-1,1,3,5,7])
```

[illegible]

В случае генерации диагональных ядер сразу генерируются два ядра: главная и побочная диагональ:

```
wdl.diagonal([-7, -5, -3, -1], [1, 3, 5, 7], 1, 14)

[[1, -7, -5, -3, -1, -7, -5, -3, -1, -7, -5, -3, -1, -7],
 [1, 1, -7, -5, -3, -1, -7, -5, -3, -1, -7, -5, -3, -1],
 [3, 1, 1, -7, -5, -3, -1, -7, -5, -3, -1, -7, -5, -3],
 [5, 3, 1, 1, -7, -5, -3, -1, -7, -5, -3, -1, -7, -5],
 [7, 5, 3, 1, 1, -7, -5, -3, -1, -7, -5, -3, -1, -7],
 [1, 7, 5, 3, 1, 1, -7, -5, -3, -1, -7, -5, -3, -1],
 [3, 1, 7, 5, 3, 1, 1, -7, -5, -3, -1, -7, -5, -3],
 [5, 3, 1, 7, 5, 3, 1, 1, -7, -5, -3, -1, -7, -5],
 [7, 5, 3, 1, 7, 5, 3, 1, 1, -7, -5, -3, -1, -7],
 [1, 7, 5, 3, 1, 7, 5, 3, 1, 1, -7, -5, -3, -1],
 [3, 1, 7, 5, 3, 1, 7, 5, 3, 1, 1, -7, -5, -3],
 [5, 3, 1, 7, 5, 3, 1, 7, 5, 3, 1, 1, -7, -5],
 [7, 5, 3, 1, 7, 5, 3, 1, 7, 5, 3, 1, 1, -7],
 [1, 7, 5, 3, 1, 7, 5, 3, 1, 7, 5, 3, 1, 1]]
```

Первый параметр – список значений, которые будут находиться выше диагонали, второй параметр – список значений, которые будут находиться ниже диагонали, третий параметр – значение, из которого будет состоять диагональ, последний параметр – размерность сверточного ядра.

У нас имеется четыре сгенерированных файла txt с ядрами. Процесс установки в структуру модели следующий:

- инициализировать структуру ядер и преобразовать первое ядро с помощью функции `txt_kernel`. Параметр функции – путь к сгенерированному ядру. В данном случае `w1` обозначает весовые коэффициенты первого ядра.

```
w1 = wdl.txt_kernel('horizontal.txt')
```

- установить дополнительное ядро с помощью функции `add_kernel`. Первый параметр – путь к добавляемому ядру. Второй параметр – переменная, которой присвоено значение матрицы предыдущего ядра/ядер, к которому производится добавление.

```
w2 = wdl.add_kernel('vertical.txt', w1)
```

- продолжать добавление нужного количества ядер.

```
w3 = wdl.add_kernel('diagonal1.txt', w2)
```

```
w4 = wdl.add_kernel('diagonal2.txt', w3)
```

- В переменной w4 представлено 4 подготовленных сверточных ядра: горизонтальное, вертикальное, два диагональных. Необходимо преобразовать полученную матрицу в массив np.array:

```
w4 = np.array(w4)
```

Для вставки в модель полученных ядер необходимо выполнить команду:

```
model.layers[0].set_weights([w4])
```

В первых квадратных скобках установлен номер сверточного слоя, в который необходимо вставить ядра, во вторых скобках — название переменной, которой присвоена матрица весов.

Чтобы посмотреть в визуальном удобном виде установленные коэффициенты весов сверточных матриц слоя, можно использовать функцию show_kernel:

```
wdl.show_kernel(0, model)
```

Указывается номер слоя, для которого необходимо вывести значения ядер, а также название переменной, которой присвоена структура модели.

Вывод функции:

Номер сверточного ядра: 0

-----ЦВЕТОВОЙ КАНАЛ №0-----

-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,
-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,
-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,
-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,
5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,5.0,
7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,7.0,
-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,-7.0,
-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,-5.0,
-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,-3.0,
-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,3.0,

Номер сверточного ядра: 1

-----ЦВЕТОВОЙ КАНАЛ №0-----

-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,
-7.0,-5.0,-3.0,-1.0,1.0,3.0,5.0,7.0,-7.0,-5.0,-3.0,-1.0,1.0,3.0,

Номер сверточного ядра: 2

-----ЦВЕТОВОЙ КАНАЛ №0-----

1.0,-7.0,-5.0,-3.0,-1.0,-7.0,-5.0,-3.0,-1.0,-7.0,-5.0,-3.0,-1.0,-7.0,
1.0,1.0,-7.0,-5.0,-3.0,-1.0,-7.0,-5.0,-3.0,-1.0,-7.0,-5.0,-3.0,-1.0,

Как видим, ядра установлены правильно. Теперь скомпилируем модель и запустим обучение:

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

```
model.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [=====] - 1s 615us/step - loss: 7.8812 - accuracy: 0.5989
Epoch 2/10
1875/1875 [=====] - 1s 599us/step - loss: 0.8603 - accuracy: 0.8072
Epoch 3/10
1875/1875 [=====] - 1s 610us/step - loss: 0.6113 - accuracy: 0.8247
Epoch 4/10
1875/1875 [=====] - 1s 601us/step - loss: 0.5770 - accuracy: 0.8261
Epoch 5/10
1875/1875 [=====] - 1s 602us/step - loss: 0.5661 - accuracy: 0.8281
Epoch 6/10
1875/1875 [=====] - 1s 603us/step - loss: 0.5654 - accuracy: 0.8276
Epoch 7/10
1875/1875 [=====] - 1s 602us/step - loss: 0.5641 - accuracy: 0.8278
Epoch 8/10
1875/1875 [=====] - 1s 602us/step - loss: 0.5639 - accuracy: 0.8266
Epoch 9/10
1875/1875 [=====] - 1s 603us/step - loss: 0.5597 - accuracy: 0.8272
Epoch 10/10
1875/1875 [=====] - 1s 612us/step - loss: 0.5599 - accuracy: 0.8288
```

На таком варианте выбранных параметров сверточного слоя имеются следующие значения точности по обучающей и тестовой выборкам:

```
train_loss, train_acc = model.evaluate(x_train, y_train)
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
1875/1875 [=====] - 1s 577us/step - loss: 0.5363 - accuracy: 0.8386
313/313 [=====] - 0s 593us/step - loss: 0.5191 - accuracy: 0.8463
```

MNIST содержит черно-белые изображения. Для цветных изображений (трехканальных RGB) процесс установки ядер в структуру немного отличается. В качестве примера рассмотрим датасет Breast Histopathology Images:

<https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images>

Использовалась следующая структура нейронной сети:

<https://www.kaggle.com/code/abeerelmorshedy/breast-cancer-using-cnn>

Сократим количество сверточных слоев до одного. Также установим 4 больших сверточных ядра с большим шагом вместо нескольких ядер малой размерности.

Размер входных изображений 50*50. Установим ядра размерности 25*25 с шагом 12. Генерация ядер аналогична примеру с черно-белыми изображениями.

Ядра устанавливаются с помощью функций с пометкой rgb:

```
w1 = txt_kernel_rgb('/content/vertical.txt', '/content/horizontal.txt', '/content/diagonal1.txt')
w2 = add_kernel_rgb('/content/horizontal.txt', '/content/diagonal2.txt', '/content/vertical.txt', w1)
w3 = add_kernel_rgb('/content/diagonal1.txt', '/content/vertical.txt', '/content/horizontal.txt', w2)
w4 = add_kernel_rgb('/content/horizontal.txt', '/content/diagonal1.txt', '/content/diagonal2.txt', w3)
w4 = np.array(w4)
model.layers[0].set_weights([w4])
```

В функцию `txt_kernel_rgb` передается три параметра, которые соответствуют трем цветовым каналам, то есть в канал R будет установлено ядро вертикальное, в канал G – горизонтальное, в канал B – главная диагональ.

Четвертый параметр функции `add_kernel_rgb` соответствует ядру/ядрам, к которому происходит добавление, а первые три параметра – это добавляемые ядра в три цветовых канала.

Компилируем модель и обучаем:

```
[61] model.compile(Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

```
[62] history = model.fit(X_train, Y_train, validation_data = (X_test, Y_test), epochs = 11, batch_size = 35)
```

```
Epoch 1/11
1429/1429 [=====] - 15s 9ms/step - loss: 0.4519 - accuracy: 0.8019 - val_loss: 1.3223 - val_accuracy: 0.8829
Epoch 2/11
1429/1429 [=====] - 11s 7ms/step - loss: 0.3119 - accuracy: 0.8795 - val_loss: 0.2948 - val_accuracy: 0.8875
Epoch 3/11
1429/1429 [=====] - 10s 7ms/step - loss: 0.3009 - accuracy: 0.8826 - val_loss: 0.2954 - val_accuracy: 0.8894
Epoch 4/11
1429/1429 [=====] - 11s 8ms/step - loss: 0.2947 - accuracy: 0.8850 - val_loss: 0.2841 - val_accuracy: 0.8910
Epoch 5/11
1429/1429 [=====] - 11s 8ms/step - loss: 0.2894 - accuracy: 0.8864 - val_loss: 0.2828 - val_accuracy: 0.8922
Epoch 6/11
1429/1429 [=====] - 10s 7ms/step - loss: 0.2865 - accuracy: 0.8876 - val_loss: 0.2750 - val_accuracy: 0.8914
Epoch 7/11
1429/1429 [=====] - 10s 7ms/step - loss: 0.2829 - accuracy: 0.8880 - val_loss: 0.2804 - val_accuracy: 0.8914
Epoch 8/11
1429/1429 [=====] - 11s 8ms/step - loss: 0.2821 - accuracy: 0.8881 - val_loss: 0.2762 - val_accuracy: 0.8908
Epoch 9/11
1429/1429 [=====] - 13s 9ms/step - loss: 0.2790 - accuracy: 0.8900 - val_loss: 0.2727 - val_accuracy: 0.8923
Epoch 10/11
1429/1429 [=====] - 13s 9ms/step - loss: 0.2779 - accuracy: 0.8896 - val_loss: 0.2931 - val_accuracy: 0.8933
Epoch 11/11
1429/1429 [=====] - 10s 7ms/step - loss: 0.2771 - accuracy: 0.8904 - val_loss: 0.2704 - val_accuracy: 0.8939
```