Введение в искусственные нейронные сети ¶

Урок 3. TensorFlow

Практическое задание

1. Постройте нейронную сеть (берём простую линейную сеть, которую разбирали на уроке: меняем число слоёв, число нейронов, типы активации, тип оптимизатора) на датасете from sklern.datasets import load_boston.
2. Измените функцию потерь и метрику для этой задачи. Постройте 10-15 вариантов и сведите результаты их работы в таблицу. Опишите, какого результата вы добились от нейросети? Что помогло вам улучшите её точность?
3. Поработайте с документацией TensorFlow 2. Найти 2-3 полезные команды TensorFlow, не разобранные на уроке (полезные для Вас).

1-2.(*) Попробуйте обучить нейронную сеть на TensorFlow 2 на датасете imb_rewiews. Опишите, какого результата вы добились от нейросети? Что помогло вамулучшить её точность?

Выполнил **Соковнин И.Л.**

Boston House Prices

The Boston Housing Dataset - https://www.kaggle.com/prasadperera/the-boston-housing-dataset/data (https://www.kaggle.com/prasadperera/the-boston-housing-dataset/data)
https://www.machinelearningmastery.ru/linear-regression-on-boston-housing-dataset-f409b7e4a155/ (https://www.machinelearningmastery.ru/linear-regression-on-boston-housing-dataset-f409b7e4a155/)

Data: housing.csv(49.08 kB) - Boston House Price dataset

0.00632 18.00 2.310 0 0.5380 6.5750 65.20 4.0900 1 296.0 15.30 396.90 4.98 24.00

- CRIM: per capita crime rate by town
- ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS: proportion of non-retail business acres per town
- CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX: nitric oxides concentration (parts per 10 million)
- RM: average number of rooms per dwelling
- AGE: proportion of owner-occupied units built prior to 1940
- DIS: weighted distances to five Boston employment centres
- RAD: index of accessibility to radial highways
- TAX: full-value property-tax rate per $10,000
- PTRATIO: pupil-teacher ratio by town
- B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT: % lower status of the population
- MEDV: Median value of owner-occupied homes in $1000's

- CRIM: - уровень преступности на душу населения по городам
- ZN: - доля земли под жилую застройку зонирована на участки площадью более 25 000 кв. футов.
- INDUS: - доля акров, не относящихся к розничной торговле, на город
- CHAS: - фиктивная переменная Charles River (= 1, если участок ограничивает реку; 0 в противном случае)
- NOX: - концентрация оксидов азота (частей на 10 миллионов)
- RM: - среднее количество комнат в доме в PM
- AGE: - доля единиц, занимаемых владельцами, построенных до 1940 г.
- DIS: - взвешенные расстояния до пяти бостонских центров занятости
- RAD: - индекс доступности радиальных автомобильных дорог
- TAX: - ставка налога на имущество в размере полной стоимости из расчета 10 000 долларов США.
- PTRATIO: - cСоотношение учеников и учителей по городам
- B: 1000 (Bk - 0,63) ^ 2, где Bk - доля черных по городам
- LSTAT: - % более низкий статус населения
- MEDV: - средняя стоимость домов, занимаемых владельцами, в 1000 долларов США.

1. Как меняется нейронная сеть под действием задачи?
2. Выход не м.б. ограниченным => либо 'relu' либо линейная.
3. loss должен учитывать характеристики задачи.
4. Метрика учитывает вид выхода, тип задачи (при выборе потерь и метрики):
    - квадратичная ошибка, абсолютная ошибка, коэффициент детерминации

B [33]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

from sklearn.metrics import r2_score
```

B [34]:
```
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adagrad  # Оптимизаторы
```

B [35]:
```
print(tf.__version__)
```

```
2.9.0-dev20211226
```

B [36]:
```
# !conda install -c anaconda scikit-learn
```

B [37]:
```
# !pip3 install -U scikit-learn
```

B [38]:
```
from sklearn.datasets import load_boston
```

B [39]:
```
boston_dataset = load_boston()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function load_boston is depr
ecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

B [40]:
```
print(boston_dataset.keys())

# data: содержит информацию для различных домов
# target: цены на дом
# feature_names: названия функций
# DESCR: описывает набор данных
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_module'])
```

B [41]:
```
print(boston_dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley,
 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Co
nference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

B [42]:
```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
print(boston.shape)
boston.head()
```

```
(506, 13)
```

Out[42]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

B [43]:
```
boston.describe().T
```

Out[43]:

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.9000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700

B [44]:
```
# boston['MEDV'] = boston_dataset.target
target = boston_dataset["target"]
target[:10]
```

Out[44]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9])
```

Предварительная обработка данных

B [45]:
```
boston.isnull().sum()
```

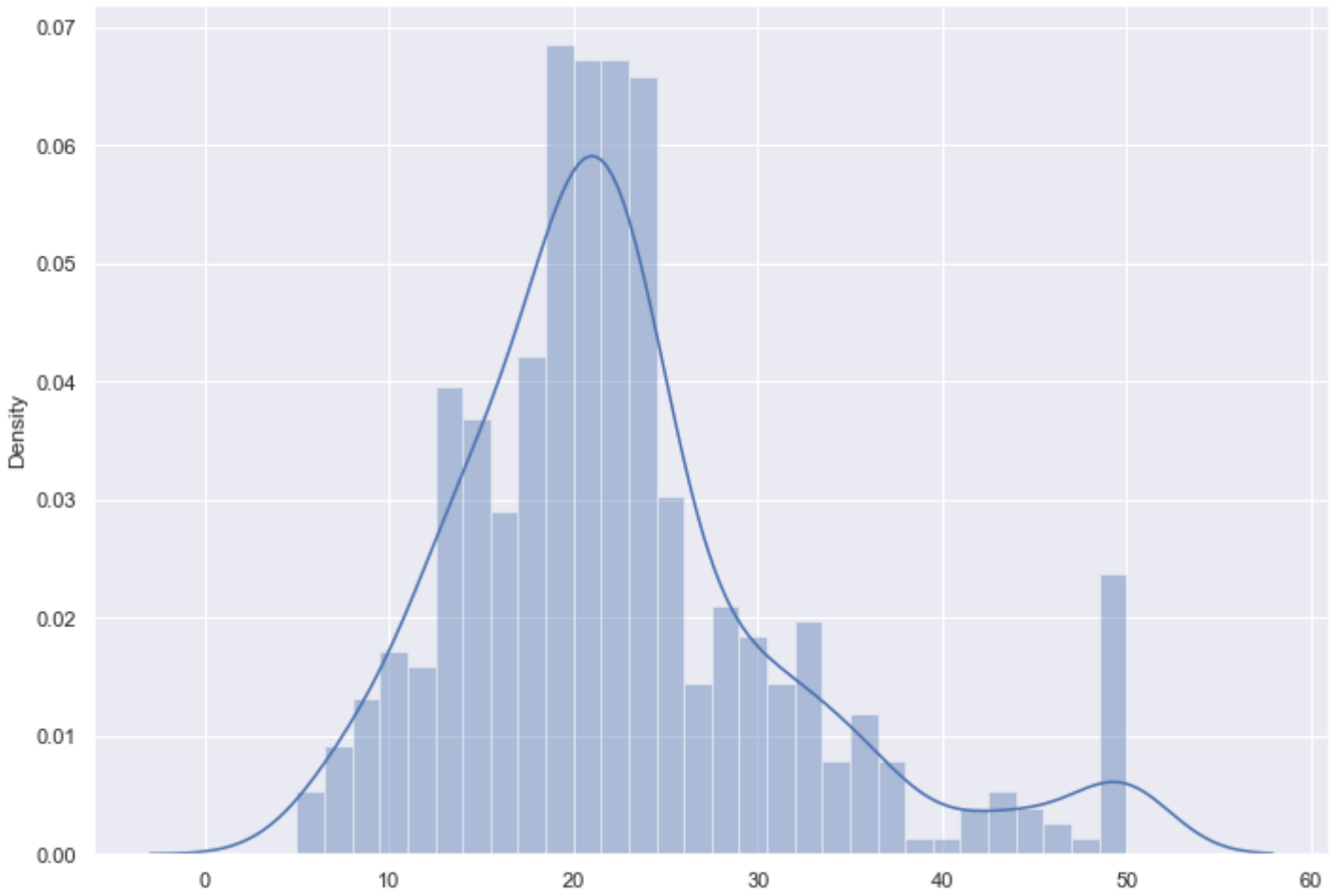
Out[45]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

Исследовательский анализ данных

B [46]:
```
import seaborn as sns
```

B [47]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(target, bins=30)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



B [48]:
```
correlation_matrix = boston.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

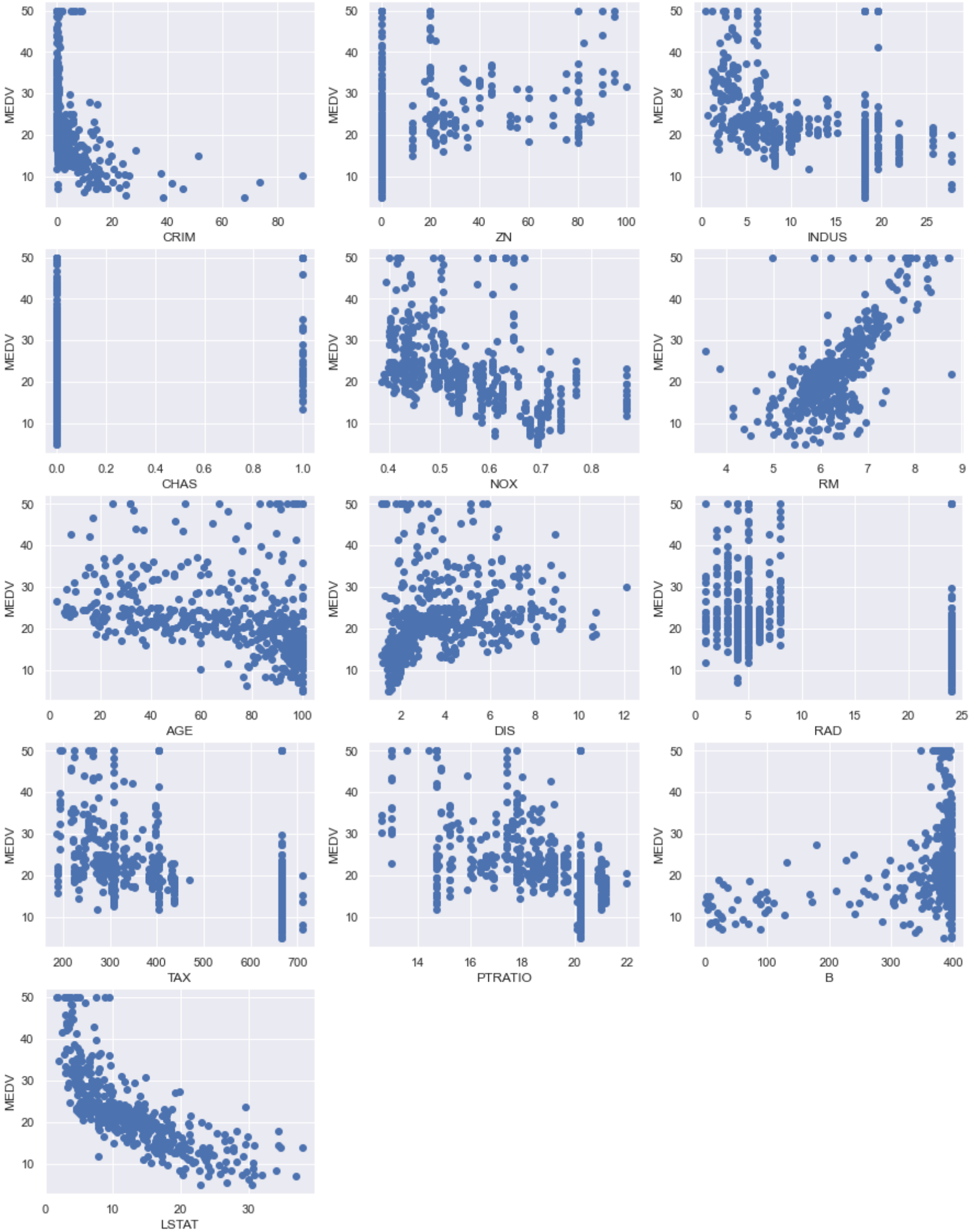
Out[48]: <AxesSubplot:>

Коэффициент корреляции варьируется от -1 до 1. Если значение близко к 1, это означает, что между двумя переменными существует сильная положительная корреляция. Когда оно близко к -1, переменные имеют сильную отрицательную корреляцию.

B [49]:
```
features = boston.columns
features
```

Out[49]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

B [50]:
```
plt.figure(figsize=(15, 20))

# features = ['LSTAT', 'RM']
# target = boston['MEDV']

features = boston.keys()

for i, col in enumerate(features):
    plt.subplot(5, 3, i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker='o')
    # plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



Замечания:

- Цены растут по мере линейного увеличения стоимости RM. Существует несколько выбросов, и данные, похоже, ограничены 50.
- Цены имеют тенденцию к снижению с ростом LSTAT (не линейно).

B [19]:
```
# # репрезентация данных в виде графиков
# g = sns.pairplot(boston)
# plt.show() # расскоментируйте, чтобы посмотреть
```

Практическое задание

1. Постройте нейронную сеть (берём простую линейную сеть, которую разбирали на уроке: меняем число слоёв, число нейронов, типы активации, тип оптимизатора) на датасете from sklern.datasets import load_boston.
2. Измените функцию потерь и метрику для этой задачи. Постройте 10-15 вариантов и сведите результаты их работы в таблицу. Опишите, какого результата вы добились от нейросети? Что помогло вам улучшите её точность?

Разделение данных на обучающие и тестовые наборы

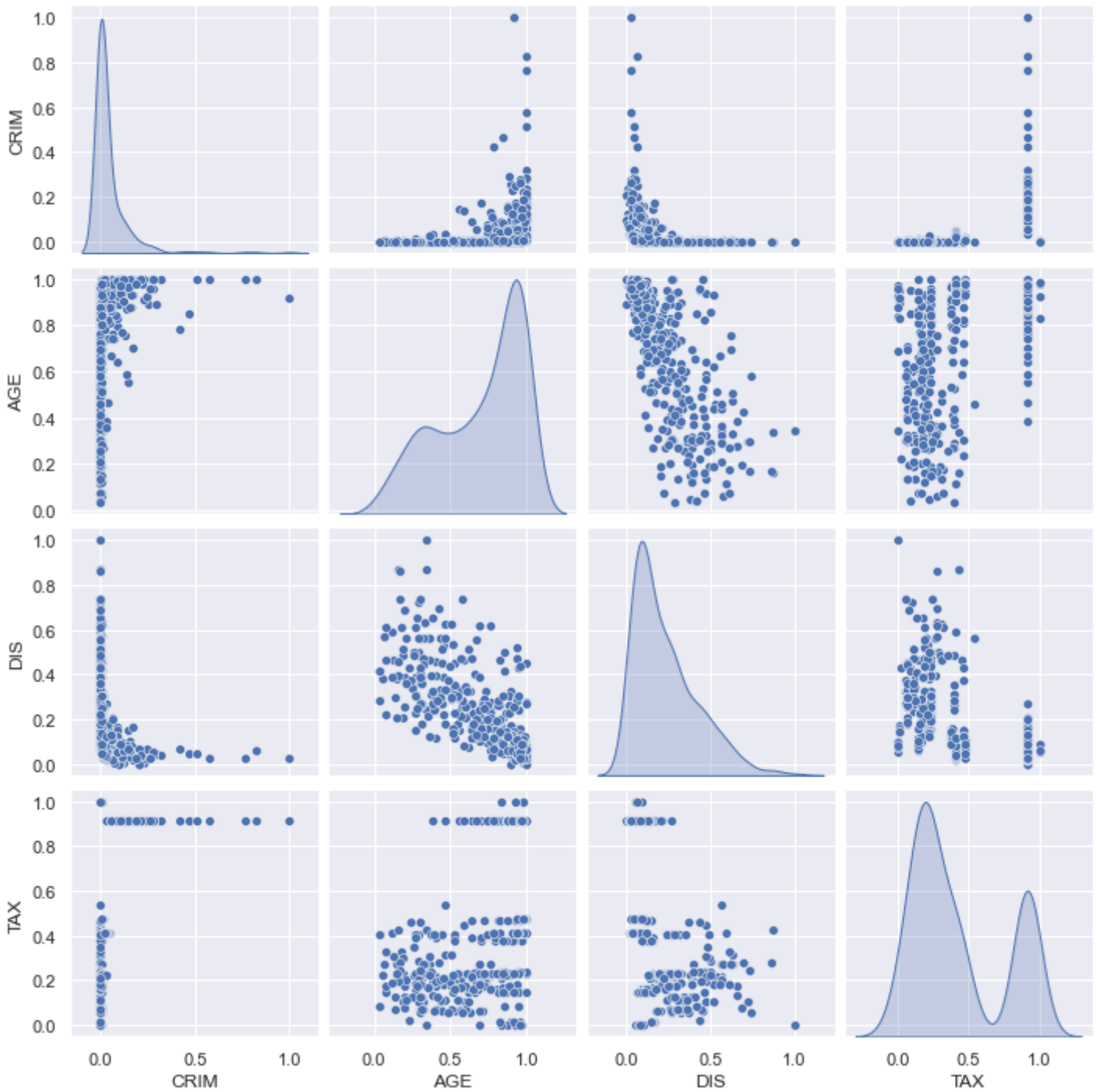
B [54]:
```
from sklearn.preprocessing import MinMaxScaler

scaler_data = MinMaxScaler()
train_data = scaler_data.fit_transform(boston)
```

B [56]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score


train_data = pd.DataFrame(train_data, columns=boston_dataset["feature_names"])

# X_train, X_test, y_train, y_test = train_test_split(boston, target, test_size = 0.2, random_state=5)
X_train, X_test, y_train, y_test = train_test_split(train_data, target, test_size=0.25)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(379, 13)
(127, 13)
(379,)
(127,)
```

B [59]:
```
sns.pairplot(X_train[["CRIM","AGE","DIS","TAX"]], diag_kind="kde")
```

Out[59]:
```
<seaborn.axisgrid.PairGrid at 0xf9a2015490>
```



B [25]:
```
train_data.columns
```

Out[25]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

В [ ]:
```
# определим число нейронов скрытого слоя (ширина скрытого слоя)
neurons = [50, 100, 200]

# определим скорость обучения (learning rate)
# n = [0.005, 0.05, 0.5, 1.0]

# определим количество эпох
# epoch_numb = [100, 1000, 5000]

# определим оптимизаторы
optimizers =  ['SGD','RMSProp','adam','NAdam']

# определим метрики
metrics = ['MSE', 'MAE']
```

В [27]:
```
# input6 = keras.layers.Input( shape=(28, 28) )
# x6 = keras.layers.Flatten()(input6)
# x6 = keras.layers.Dense(256, activation='relu')(x6)
# x6 = keras.layers.Dense(1, activation='sigmoid')(x6)


# model6 =Model(inputs=input6,outputs=x6)
# model6.compile(optimizer=tf.keras.optimizers.Adagrad(lr=0.1, epsilon=1e-08, decay=0.0),
#                loss=tf.keras.losses.BinaryCrossentropy(),
#                metrics=['accuracy'])

# model6.fit(train_images,y_train_labels[:,6], epochs = 10, validation_split = 0.2)
```

B [28]:
```
from sklearn.metrics import mean_squared_error

colNames = ["neurons", "epochs", "optimizer", "metric",
            "loss (train)", "metric_val (train)", "rmse (train)", "r2 (train)",
            "loss (test)", "metric_val (test)", "rmse (test)", "r2 (test)" ]

model_3_layers = pd.DataFrame(columns=colNames)

k = 1000  # количество эпох
for i in neurons:  # ширина скрытого слоя
#     for j, i_optim in enumerate([keras.optimizers.RMSprop(),keras.optimizers.SGD(),keras.optimizers.Adam()]):
    for i_optim in optimizers:  # оптимизаторы
        for k_metric in metrics:  # метрики
            x_input = keras.layers.Input( shape=(13))
            x = keras.layers.Dense(i, activation='relu')(x_input)
            x_output = keras.layers.Dense(1)(x)
            model = keras.models.Model(inputs=x_input, outputs=x_output)

            model.compile(
                optimizer=i_optim,
                loss='mae',
                metrics=[k_metric])

            model.fit(
                X_train,
                y_train,
                epochs=k,
                batch_size=100,
                # validation_split=0.2,  # использовать 20 % данных для валидации
                verbose=0
            )

            # model evaluation for training set
            loss_metric_train = model.evaluate(X_train, y_train)

            y_train_predict = model.predict(X_train)
            rmse_train = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
            r2_train = r2_score(y_train, y_train_predict)

            # model evaluation for testing set
            loss_metric_test = model.evaluate(X_test, y_test)

            y_test_predict = model.predict(X_test)
            rmse_test = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
            r2_test = r2_score(y_test, y_test_predict)

#             print(k_metric)
            model_3_layers = model_3_layers.append(pd.DataFrame({
                'neurons': [i],
#                 'скорость обучения': [n],
                'epochs': [k],
                'optimizer': i_optim,
                'metric': k_metric,

                'loss (train)': [loss_metric_train[0]],
                'metric_val (train)': [loss_metric_train[1]],
                'rmse (train)': [rmse_train],
                'r2 (train)': [r2_train],

                'loss (test)': [loss_metric_test[0]],
                'metric_val (test)': [loss_metric_test[1]],
                'rmse (test)': [rmse_test],
                'r2 (test)': [r2_test]}),
                ignore_index=True)
```

```
12/12 [==============================] - 0s 1ms/step - loss: 2.6603 - MSE: 18.6085
4/4 [==============================] - 0s 3ms/step - loss: 2.8441 - MSE: 22.5243
12/12 [==============================] - 0s 2ms/step - loss: 2.7214 - MAE: 2.7214
4/4 [==============================] - 0s 4ms/step - loss: 2.9402 - MAE: 2.9402
12/12 [==============================] - 0s 2ms/step - loss: 2.5203 - MSE: 17.3340
4/4 [==============================] - 0s 3ms/step - loss: 2.7231 - MSE: 20.5860
12/12 [==============================] - 0s 2ms/step - loss: 2.5915 - MAE: 2.5915
4/4 [==============================] - 0s 3ms/step - loss: 2.7658 - MAE: 2.7658
12/12 [==============================] - 0s 2ms/step - loss: 2.7688 - MSE: 19.5373
4/4 [==============================] - 0s 3ms/step - loss: 2.9402 - MSE: 22.8572
12/12 [==============================] - 0s 2ms/step - loss: 2.9428 - MAE: 2.9428
4/4 [==============================] - 0s 4ms/step - loss: 2.8902 - MAE: 2.8902
12/12 [==============================] - 0s 2ms/step - loss: 2.7989 - MSE: 20.0604
4/4 [==============================] - 0s 3ms/step - loss: 2.9519 - MSE: 23.9031
12/12 [==============================] - 0s 2ms/step - loss: 2.8562 - MAE: 2.8562
4/4 [==============================] - 0s 3ms/step - loss: 2.9315 - MAE: 2.9315
12/12 [==============================] - 0s 2ms/step - loss: 2.6991 - MSE: 19.1236
4/4 [==============================] - 0s 3ms/step - loss: 2.8587 - MSE: 22.9158
12/12 [==============================] - 0s 2ms/step - loss: 2.6479 - MAE: 2.6479
4/4 [==============================] - 0s 3ms/step - loss: 2.8495 - MAE: 2.8495
12/12 [==============================] - 0s 2ms/step - loss: 2.4269 - MSE: 16.6393
4/4 [==============================] - 0s 3ms/step - loss: 2.6356 - MSE: 20.2487
```

```
12/12 [==============================] - 0s 2ms/step - loss: 2.4208 - MAE: 2.4208
4/4 [==============================] - 0s 3ms/step - loss: 2.6830 - MAE: 2.6830
12/12 [==============================] - 0s 2ms/step - loss: 2.2588 - MSE: 15.6234
4/4 [==============================] - 0s 3ms/step - loss: 2.6255 - MSE: 20.7431
12/12 [==============================] - 0s 2ms/step - loss: 2.3641 - MAE: 2.3641
4/4 [==============================] - 0s 3ms/step - loss: 2.7010 - MAE: 2.7010
12/12 [==============================] - 0s 2ms/step - loss: 2.7931 - MSE: 20.8813
4/4 [==============================] - 0s 3ms/step - loss: 2.8975 - MSE: 24.0148
12/12 [==============================] - 0s 2ms/step - loss: 2.6672 - MAE: 2.6672
4/4 [==============================] - 0s 2ms/step - loss: 2.7853 - MAE: 2.7853
12/12 [==============================] - 0s 2ms/step - loss: 2.6690 - MSE: 18.9900
4/4 [==============================] - 0s 3ms/step - loss: 2.8415 - MSE: 22.7509
12/12 [==============================] - 0s 2ms/step - loss: 2.6735 - MAE: 2.6735
4/4 [==============================] - 0s 3ms/step - loss: 2.8683 - MAE: 2.8683
12/12 [==============================] - 0s 2ms/step - loss: 2.1753 - MSE: 14.3301
4/4 [==============================] - 0s 4ms/step - loss: 2.6135 - MSE: 19.0411
12/12 [==============================] - 0s 3ms/step - loss: 2.2836 - MAE: 2.2836
4/4 [==============================] - 0s 3ms/step - loss: 2.6458 - MAE: 2.6458
12/12 [==============================] - 0s 2ms/step - loss: 2.0336 - MSE: 13.0888
4/4 [==============================] - 0s 3ms/step - loss: 2.5328 - MSE: 18.2020
12/12 [==============================] - 0s 3ms/step - loss: 2.1548 - MAE: 2.1548
4/4 [==============================] - 0s 4ms/step - loss: 2.6097 - MAE: 2.6097
12/12 [==============================] - 0s 2ms/step - loss: 2.3524 - MSE: 16.7657
4/4 [==============================] - 0s 4ms/step - loss: 2.7222 - MSE: 21.7673
12/12 [==============================] - 0s 3ms/step - loss: 2.2835 - MAE: 2.2835
4/4 [==============================] - 0s 3ms/step - loss: 2.6949 - MAE: 2.6949
```

B [29]:
```
# pd.options.display.max_columns = None
# pd.options.display.max_rows = None
model_3_layers.sort_values(by='r2 (test)', ascending=False)
```

Out[29]:

	neurons	epochs	optimizer	metric	loss (train)	metric_val (train)	rmse (train)	r2 (train)	loss (test)	metric_val (test)	rmse (test)	r2 (test)
20	200	1000	adam	MSE	2.033568	13.088802	3.617845	0.854712	2.532759	18.202017	4.266382	0.726369
18	200	1000	RMSProp	MSE	2.175313	14.330061	3.785507	0.840934	2.613506	19.041100	4.363611	0.713755
21	200	1000	adam	MAE	2.154835	2.154835	3.787960	0.840728	2.609661	2.609661	4.395317	0.709580
19	200	1000	RMSProp	MAE	2.283551	2.283551	3.950742	0.826745	2.645793	2.645793	4.484584	0.697664
10	100	1000	RMSProp	MSE	2.426916	16.639341	4.079135	0.815301	2.635586	20.248686	4.499854	0.695601
11	100	1000	RMSProp	MAE	2.420751	2.420751	4.046276	0.818264	2.683014	2.683014	4.507568	0.694557
2	50	1000	RMSProp	MSE	2.520309	17.334049	4.163418	0.807589	2.723081	20.585991	4.537179	0.690531
12	100	1000	adam	MSE	2.258771	15.623443	3.952650	0.826577	2.625455	20.743137	4.554463	0.688168
13	100	1000	adam	MAE	2.364052	2.364052	4.111797	0.812331	2.700979	2.700979	4.644477	0.675720
22	200	1000	NAdam	MSE	2.352403	16.765678	4.094592	0.813898	2.722205	21.767317	4.665545	0.672772
23	200	1000	NAdam	MAE	2.283510	2.283510	4.020187	0.820600	2.694912	2.694912	4.672455	0.671802
15	100	1000	NAdam	MAE	2.667193	2.667193	4.368858	0.788132	2.785326	2.785326	4.672460	0.671801
9	100	1000	SGD	MAE	2.647913	2.647913	4.330896	0.791798	2.849529	2.849529	4.729398	0.663753
0	50	1000	SGD	MSE	2.660324	18.608490	4.313756	0.793443	2.844080	22.524300	4.745977	0.661392
3	50	1000	RMSProp	MAE	2.591540	2.591540	4.269142	0.797693	2.765797	2.765797	4.762915	0.658971
17	200	1000	SGD	MAE	2.673540	2.673540	4.332473	0.791646	2.868333	2.868333	4.766646	0.658436
16	200	1000	SGD	MSE	2.669021	18.989975	4.357749	0.789208	2.841514	22.750919	4.769792	0.657985
4	50	1000	adam	MSE	2.768833	19.537346	4.420107	0.783132	2.940224	22.857212	4.780922	0.656387
8	100	1000	SGD	MSE	2.699124	19.123608	4.373054	0.787725	2.858680	22.915777	4.787043	0.655507
7	50	1000	NAdam	MAE	2.856231	2.856231	4.607445	0.764360	2.931466	2.931466	4.836031	0.648420
1	50	1000	SGD	MAE	2.721354	2.721354	4.414626	0.783670	2.940151	2.940151	4.846568	0.646886
6	50	1000	NAdam	MSE	2.798899	20.060415	4.478885	0.777326	2.951890	23.903143	4.889084	0.640664
5	50	1000	adam	MAE	2.942799	2.942799	4.697543	0.755054	2.890187	2.890187	4.899943	0.639066
14	100	1000	NAdam	MSE	2.793050	20.881323	4.569609	0.768214	2.897529	24.014818	4.900491	0.638985

Лучшее значения на test-е при **количестве нейронов**=200, **количество слоёв**=3, **optimizer**='RMSProp', **тип активации**='relu', **metric**='MSE' и составил **r2** = 0.726369.

- **train**: r2 = 0.854712
- **test**: r2 = 0.726369

Для train:

- loss = 2.033568
- metrics values=13.088802
- rmse = 3.617845
- r2 = 0.854712

Для test:

- loss=2.532759

- metrics values=18.202017
- rmse = 4.266382
- r2 = 0.726369

Увеличим количество слоёв и проверим как изменился результат.

B [31]:
```
model_n_layers = pd.DataFrame(columns=colNames)

k = 1000  # количество эпох
for i in neurons:  # ширина скрытого слоя
    for i_optim in optimizers:  # оптимизаторы
        for k_metric in metrics:  # метрики

            x_input = keras.layers.Input( shape=(13))
            x = keras.layers.Dense(i, activation='relu')(x_input)
            x_1 = keras.layers.Dense(i, activation='relu')(x)
            x_2 = keras.layers.Dense(i, activation='relu')(x_1)
            x_output = keras.layers.Dense(1)(x_2)
            model = keras.models.Model(inputs=x_input, outputs=x_output)

            model.compile(
                optimizer=i_optim,
                loss='mae',
                metrics=[k_metric])

            model.fit(
                X_train,
                y_train,
                epochs=k,
                batch_size=100,
                # validation_split=0.2,  # использовать 20 % данных для валидации
                verbose=0
            )

            # model evaluation for training set
            loss_metric_train = model.evaluate(X_train, y_train)

            y_train_predict = model.predict(X_train)
            rmse_train = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
            r2_train = r2_score(y_train, y_train_predict)

            # model evaluation for testing set
            loss_metric_test = model.evaluate(X_test, y_test)

            y_test_predict = model.predict(X_test)
            rmse_test = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
            r2_test = r2_score(y_test, y_test_predict)

#             print(k_metric)
            model_n_layers = model_3_layers.append(pd.DataFrame({
                'neurons': [i],
#                 'скорость обучения': [n],
                'epochs': [k],
                'optimizer': i_optim,
                'metric': k_metric,

                'loss (train)': [loss_metric_train[0]],
                'metric_val (train)': [loss_metric_train[1]],
                'rmse (train)': [rmse_train],
                'r2 (train)': [r2_train],

                'loss (test)': [loss_metric_test[0]],
                'metric_val (test)': [loss_metric_test[1]],
                'rmse (test)': [rmse_test],
                'r2 (test)': [r2_test]}),
                ignore_index=True)
```

```
12/12 [==============================] - 0s 2ms/step - loss: 1.9990 - MSE: 11.0744
4/4 [==============================] - 0s 3ms/step - loss: 2.4897 - MSE: 16.3536
12/12 [==============================] - 0s 1ms/step - loss: 1.8143 - MAE: 1.8143
4/4 [==============================] - 0s 3ms/step - loss: 2.4118 - MAE: 2.4118
12/12 [==============================] - 0s 2ms/step - loss: 1.5422 - MSE: 7.3078
4/4 [==============================] - 0s 3ms/step - loss: 2.3649 - MSE: 14.8665
12/12 [==============================] - 0s 2ms/step - loss: 1.6359 - MAE: 1.6359
4/4 [==============================] - 0s 3ms/step - loss: 2.2707 - MAE: 2.2707
12/12 [==============================] - 0s 2ms/step - loss: 1.2817 - MSE: 4.8690
4/4 [==============================] - 0s 3ms/step - loss: 2.1308 - MSE: 9.3186
12/12 [==============================] - 0s 3ms/step - loss: 1.5695 - MAE: 1.5695
4/4 [==============================] - 0s 3ms/step - loss: 2.4520 - MAE: 2.4520
12/12 [==============================] - 0s 2ms/step - loss: 1.6294 - MSE: 8.2444
4/4 [==============================] - 0s 4ms/step - loss: 2.2014 - MSE: 14.1310
12/12 [==============================] - 0s 2ms/step - loss: 1.3483 - MAE: 1.3483
4/4 [==============================] - 0s 3ms/step - loss: 2.2379 - MAE: 2.2379
12/12 [==============================] - 0s 2ms/step - loss: 1.8207 - MSE: 8.3312
4/4 [==============================] - 0s 3ms/step - loss: 2.4338 - MSE: 13.9453
12/12 [==============================] - 0s 2ms/step - loss: 2.1404 - MAE: 2.1404
4/4 [==============================] - 0s 3ms/step - loss: 2.7499 - MAE: 2.7499
12/12 [==============================] - 0s 2ms/step - loss: 1.5531 - MSE: 5.5301
4/4 [==============================] - 0s 4ms/step - loss: 2.1662 - MSE: 10.4698
12/12 [==============================] - 0s 2ms/step - loss: 1.3336 - MAE: 1.3336
4/4 [==============================] - 0s 4ms/step - loss: 2.0773 - MAE: 2.0773
12/12 [==============================] - 0s 2ms/step - loss: 0.9760 - MSE: 2.8693
4/4 [==============================] - 0s 4ms/step - loss: 2.2389 - MSE: 9.9273
12/12 [==============================] - 0s 2ms/step - loss: 0.8929 - MAE: 0.8929
```

```
4/4 [==============================] - 0s 3ms/step - loss: 2.1425 - MAE: 2.1425
12/12 [==============================] - 0s 2ms/step - loss: 1.0863 - MSE: 2.6692
4/4 [==============================] - 0s 3ms/step - loss: 1.8594 - MSE: 7.8601
12/12 [==============================] - 0s 1ms/step - loss: 0.9953 - MAE: 0.9953
4/4 [==============================] - 0s 4ms/step - loss: 1.9099 - MAE: 1.9099
12/12 [==============================] - 0s 3ms/step - loss: 2.4374 - MSE: 11.3094
4/4 [==============================] - 0s 3ms/step - loss: 2.9939 - MSE: 17.6482
12/12 [==============================] - 0s 2ms/step - loss: 1.9496 - MAE: 1.9496
4/4 [==============================] - 0s 4ms/step - loss: 2.4067 - MAE: 2.4067
12/12 [==============================] - 0s 2ms/step - loss: 1.2737 - MSE: 2.8795
4/4 [==============================] - 0s 4ms/step - loss: 1.9864 - MSE: 9.3964
12/12 [==============================] - 0s 2ms/step - loss: 1.0952 - MAE: 1.0952
4/4 [==============================] - 0s 3ms/step - loss: 2.4584 - MAE: 2.4584
12/12 [==============================] - 0s 2ms/step - loss: 0.6733 - MSE: 1.3943
4/4 [==============================] - 0s 3ms/step - loss: 1.9664 - MSE: 9.4012
12/12 [==============================] - 0s 3ms/step - loss: 0.6361 - MAE: 0.6361
4/4 [==============================] - 0s 4ms/step - loss: 1.9908 - MAE: 1.9908
12/12 [==============================] - 0s 2ms/step - loss: 0.8721 - MSE: 1.5871
4/4 [==============================] - 0s 4ms/step - loss: 2.0307 - MSE: 9.0119
12/12 [==============================] - 0s 2ms/step - loss: 0.8756 - MAE: 0.8756
4/4 [==============================] - 0s 4ms/step - loss: 2.0603 - MAE: 2.0603
```

B [32]:

```
model_n_layers.sort_values(by='r2 (test)', ascending=False)
```

Out[32]:

	neurons	epochs	optimizer	metric	loss (train)	metric_val (train)	rmse (train)	r2 (train)	loss (test)	metric_val (test)	rmse (test)	r2 (test)
24	200	1000	NAdam	MAE	0.875556	0.875556	1.325170	0.980507	2.060283	2.060283	3.186189	0.847388
20	200	1000	adam	MSE	2.033568	13.088802	3.617845	0.854712	2.532759	18.202017	4.266382	0.726369
18	200	1000	RMSProp	MSE	2.175313	14.330061	3.785507	0.840934	2.613506	19.041100	4.363611	0.713755
21	200	1000	adam	MAE	2.154835	2.154835	3.787960	0.840728	2.609661	2.609661	4.395317	0.709580
19	200	1000	RMSProp	MAE	2.283551	2.283551	3.950742	0.826745	2.645793	2.645793	4.484584	0.697664
10	100	1000	RMSProp	MSE	2.426916	16.639341	4.079135	0.815301	2.635586	20.248686	4.499854	0.695601
11	100	1000	RMSProp	MAE	2.420751	2.420751	4.046276	0.818264	2.683014	2.683014	4.507568	0.694557
2	50	1000	RMSProp	MSE	2.520309	17.334049	4.163418	0.807589	2.723081	20.585991	4.537179	0.690531
12	100	1000	adam	MSE	2.258771	15.623443	3.952650	0.826577	2.625455	20.743137	4.554463	0.688168
13	100	1000	adam	MAE	2.364052	2.364052	4.111797	0.812331	2.700979	2.700979	4.644477	0.675720
22	200	1000	NAdam	MSE	2.352403	16.765678	4.094592	0.813898	2.722205	21.767317	4.665545	0.672772
23	200	1000	NAdam	MAE	2.283510	2.283510	4.020187	0.820600	2.694912	2.694912	4.672455	0.671802
15	100	1000	NAdam	MAE	2.667193	2.667193	4.368858	0.788132	2.785326	2.785326	4.672460	0.671801
9	100	1000	SGD	MAE	2.647913	2.647913	4.330896	0.791798	2.849529	2.849529	4.729398	0.663753
0	50	1000	SGD	MSE	2.660324	18.608490	4.313756	0.793443	2.844080	22.524300	4.745977	0.661392
3	50	1000	RMSProp	MAE	2.591540	2.591540	4.269142	0.797693	2.765797	2.765797	4.762915	0.658971
17	200	1000	SGD	MAE	2.673540	2.673540	4.332473	0.791646	2.868333	2.868333	4.766646	0.658436
16	200	1000	SGD	MSE	2.669021	18.989975	4.357749	0.789208	2.841514	22.750919	4.769792	0.657985
4	50	1000	adam	MSE	2.768833	19.537346	4.420107	0.783132	2.940224	22.857212	4.780922	0.656387
8	100	1000	SGD	MSE	2.699124	19.123608	4.373054	0.787725	2.858680	22.915777	4.787043	0.655507
7	50	1000	NAdam	MAE	2.856231	2.856231	4.607445	0.764360	2.931466	2.931466	4.836031	0.648420
1	50	1000	SGD	MAE	2.721354	2.721354	4.414626	0.783670	2.940151	2.940151	4.846568	0.646886
6	50	1000	NAdam	MSE	2.798899	20.060415	4.478885	0.777326	2.951890	23.903143	4.889084	0.640664
5	50	1000	adam	MAE	2.942799	2.942799	4.697543	0.755054	2.890187	2.890187	4.899943	0.639066
14	100	1000	NAdam	MSE	2.793050	20.881323	4.569609	0.768214	2.897529	24.014818	4.900491	0.638985

Лучшее значения на test-е при **количестве нейронов**=200, **количество слоёв**=5, **optimizer**='RMSProp', **тип активации**='relu', **metric**='MSE' и составил **r2** = 0.847388.

- **train**: **r2** = 0.980507
- **test**: **r2** = 0.847388

Для **train**:

- loss = 0.875556
- metrics values=0.875556
- rmse = 1.325170
- **r2** = 0.980507

Для **test**:

- loss=2.060283
- metrics values=2.060283
- rmse = 3.186189
- **r2** = 0.847388

Результат улучшился:

- **test**: **r2** = 0.847388 для **5 слоёв**, против **r2** = 0.726369 для **3 слоёв**.
- **train**: **r2** = 0.980507 для **5 слоёв**, против **r2** = 0.854712 для **3 слоёв**.

Улучшить точность нейросети помогли следующие факторы:

- выбор подходящего оптимизатора,
- подбор оптимального количества нейронов,
- увеличение количества слоёв,
- r2 колебался от 0,64 до 0,86 (при неизменных параметрах).

Практическое задание

3. Поработайте с документацией TensorFlow 2. Найти 2-3 полезные команды TensorFlow, не разобранные на уроке (полезные для Вас).

https://www.tensorflow.org/api_docs/python/tf (https://www.tensorflow.org/api_docs/python/tf)

- tf.Graph(): A TensorFlow computation, represented as a dataflow graph.
- tf.size(): Returns the size of a tensor.
- tf.image - module contains various functions for image processing and decoding-encoding Ops.

  - resize(): Resize images to size using the specified method.

  - resize_with_pad(): Crops and/or pads an image to a target width and height.

  - convert_image_dtype(): Convert image to dtype, scaling its values if needed.
- tf.summary - Operations for writing summary data, for use in analysis and visualization.

  - audio(): Write an audio summary.

  - text(): Write a text summary.

  - image(): Write an image summary.

В [ ]:

В [ ]:

В [ ]:

1. Как меняется нейронная сеть под действием задачи?
2. Выход не м.б. ограниченным => либо 'relu' либо линейная.
3. loss должен учитывать характеристики задачи.
4. Метрика учитывает вид выхода, тип задачи (при выборе потерь и метрики):
   - квадратичная ошибка, абсолютная ошибка, коэффициент детерминации

OpenCV (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков[2]. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD.

TensorFlow — Введение

https://coderlessons.com/tutorials/mashinnoe-obuchenie/vyuchit-tensorflow/tensorflow-kratkoe-rukovodstvo (https://coderlessons.com/tutorials/mashinnoe-obuchenie/vyuchit-tensorflow/tensorflow-kratkoe-rukovodstvo)

Искусственный интеллект включает в себя процесс моделирования человеческого интеллекта с помощью машин и специальных компьютерных систем. Примеры искусственного интеллекта включают обучение, рассуждение и самокоррекцию. Приложения искусственного интеллекта включают распознавание речи, экспертные системы, распознавание изображений и машинное зрение.

Машинное обучение — это отрасль искусственного интеллекта, которая занимается системами и алгоритмами, которые могут изучать любые новые данные и шаблоны данных.

В [ ]: