```
%tensorflow_version 2.x
import pandas as pd
import numpy as np

import tensorflow as tf
from tensorflow import keras

import matplotlib.pyplot as plt

%matplotlib inline


print(tf.__version__)
msg = tf.constant('TensorFlow 2.0 Hello World')
tf.print(msg)
```

```
2.7.0
TensorFlow 2.0 Hello World
```

```
from sklearn.datasets import load_boston
data1 = load_boston()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning

    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)
```

```
df = pd.DataFrame(data1['data'],
                  columns=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD'
                           'TAX', 'PTRATIO', 'B', 'LSTAT'])
```

```
df.isna().sum()
```

```
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
```

```
from sklearn.preprocessing import MinMaxScaler

x = df.values
min_max_scaler = MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled, columns=data1['feature_names'])
```

```
df['target'] = data1['target']
```

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	
0	0.000000	0.18	0.067815	0.0	0.314815	0.577505	0.641607	0.269203	0.000000	0.2(
1	0.000236	0.00	0.242302	0.0	0.172840	0.547998	0.782698	0.348962	0.043478	0.1(
2	0.000236	0.00	0.242302	0.0	0.172840	0.694386	0.599382	0.348962	0.043478	0.1(
3	0.000293	0.00	0.063050	0.0	0.150206	0.658555	0.441813	0.448545	0.086957	0.0(
4	0.000705	0.00	0.063050	0.0	0.150206	0.687105	0.528321	0.448545	0.086957	0.0(

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop('target', 1)
```

```
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.2, random_state=42)
```

Обучение модели

```
result = pd.DataFrame(columns= ['layers','neurons','epochs', 'activation', 'loss_mse', 'me
result
```

layers	neurons	epochs	activation	loss_mse	metrics_mae

```
n = 32
e = 2
activ = 'sigmoid'
epoch = 120
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(X_train.shape[-1],)),
    keras.layers.Dense(n, activation=activ),
    keras.layers.Dense(n, activation=activ),
    keras.layers.Dense(1, activation='linear')
])
model.compile(optimizer='sgd',
              # loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              loss= 'mse',
              metrics=['mae'])
model.fit(X_train, y_train, epochs=epoch, validation_split=0.2)
# model_all.evaluate(X_test,y_test)
result.loc[len(result.index)] = [e,
                                 n,
                                 epoch,
                                 activ,
                                round(model.evaluate(X_test,y_test)[0],4),
                                round(model.evaluate(X_test, y_test)[1],4) ]
```

```
Epoch 1/120
11/11 [==============================] - 0s 18ms/step - loss: 198.5465 - mae: 10.
Epoch 2/120
11/11 [==============================] - 0s 4ms/step - loss: 90.0375 - mae: 7.080
Epoch 3/120
11/11 [==============================] - 0s 4ms/step - loss: 88.4016 - mae: 6.958
Epoch 4/120
11/11 [==============================] - 0s 4ms/step - loss: 88.0472 - mae: 6.873
Epoch 5/120
11/11 [==============================] - 0s 5ms/step - loss: 85.5270 - mae: 6.727
Epoch 6/120
11/11 [==============================] - 0s 4ms/step - loss: 82.2679 - mae: 6.742
Epoch 7/120
11/11 [==============================] - 0s 4ms/step - loss: 79.2469 - mae: 6.686
Epoch 8/120
11/11 [==============================] - 0s 4ms/step - loss: 77.4466 - mae: 6.689
```

```
Epoch 9/120
11/11 [==============================] - 0s 6ms/step - loss: 72.9070 - mae: 6.250
Epoch 10/120
11/11 [==============================] - 0s 4ms/step - loss: 70.4854 - mae: 5.942
Epoch 11/120
11/11 [==============================] - 0s 6ms/step - loss: 63.8434 - mae: 5.792
Epoch 12/120
11/11 [==============================] - 0s 4ms/step - loss: 58.8991 - mae: 5.481
Epoch 13/120
11/11 [==============================] - 0s 6ms/step - loss: 56.5954 - mae: 5.624
Epoch 14/120
11/11 [==============================] - 0s 6ms/step - loss: 55.0706 - mae: 5.310
Epoch 15/120
11/11 [==============================] - 0s 4ms/step - loss: 50.5891 - mae: 5.086
Epoch 16/120
11/11 [==============================] - 0s 6ms/step - loss: 49.6049 - mae: 5.143
Epoch 17/120
11/11 [==============================] - 0s 4ms/step - loss: 46.2782 - mae: 4.716
Epoch 18/120
11/11 [==============================] - 0s 5ms/step - loss: 42.8228 - mae: 4.561
Epoch 19/120
11/11 [==============================] - 0s 4ms/step - loss: 42.7720 - mae: 4.668
Epoch 20/120
11/11 [==============================] - 0s 4ms/step - loss: 40.0417 - mae: 4.395
Epoch 21/120
11/11 [==============================] - 0s 5ms/step - loss: 38.3741 - mae: 4.345
Epoch 22/120
11/11 [==============================] - 0s 4ms/step - loss: 34.1951 - mae: 4.234
Epoch 23/120
11/11 [==============================] - 0s 5ms/step - loss: 37.7836 - mae: 4.328
Epoch 24/120
11/11 [==============================] - 0s 4ms/step - loss: 30.7809 - mae: 3.948
Epoch 25/120
11/11 [==============================] - 0s 4ms/step - loss: 32.6147 - mae: 4.107
Epoch 26/120
11/11 [==============================] - 0s 4ms/step - loss: 32.0770 - mae: 4.119
Epoch 27/120
11/11 [==============================] - 0s 6ms/step - loss: 40.7201 - mae: 4.606
Epoch 28/120
11/11 [==============================] - 0s 6ms/step - loss: 25.9227 - mae: 3.613
Epoch 29/120
```

```
result.head(15)
```

	layers	neurons	epochs	activation	loss_mse	metrics_mae
0	1	64	120	tanh	20.9509	3.0427
1	2	64	120	tanh	13.2469	2.3069
2	2	512	120	tanh	NaN	NaN

Вывод

Лучший результат получился с двумя слоями по 64 нейрона с активационной ф-ей "tanh" с увеличением числа нейронов или слоев ошибка сильно растет и в итоге принимают значение "NAN"

```
result.to_csv('result.csv')
```

9	2	32	120	sigmoid	16.2974	2.8040

Обучение Моделей и обьеденение по среднему значению

```
# заполняем модель списком
model1 = keras.Sequential([
    keras.layers.Flatten(input_shape=(X_train.shape[-1],)),
    keras.layers.Dense(512, activation='sigmoid'),
    keras.layers.Dense(1, activation='linear')
])

# заполняем модель , добавляя слои последовательно
model2 = keras.Sequential()
model2.add(keras.layers.Flatten(input_shape=(X_train.shape[-1],)))
model2.add(keras.layers.Dense(512, activation='linear'))
model2.add(keras.layers.Dense(1, activation='linear'))


# Заполняем модель, формируя граф передачи тензоров
x_input = keras.layers.Input(shape=(X_train.shape[-1],))
x3 = keras.layers.Flatten()(x_input)
x3 = keras.layers.Dense(512, activation='relu')(x3)
x_output = keras.layers.Dense(1, activation='linear')(x3)
model3=keras.models.Model(x_input,x_output)

model1.compile(optimizer='sgd',
               # loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               loss= 'mse',
               metrics=['mae'])
model2.compile(optimizer='rmsprop',
               # loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                loss= 'mse',
               metrics=['mae'])
model3.compile(optimizer='adam',
```

```
                # loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 loss= 'mse',
                metrics=['mae'])

hist1=model1.fit(X_train, y_train, epochs=25, batch_size=20, validation_split=0.2)
hist2=model2.fit(X_train, y_train, epochs=25, batch_size=20, validation_split=0.2)
hist3=model3.fit(X_train, y_train, epochs=25, batch_size=20, validation_split=0.2)
```

```
    ====================] - 0s 5ms/step - loss: 31.5495 - mae: 4.1210 - val_loss: 28.1

    ====================] - 0s 5ms/step - loss: 31.0712 - mae: 4.0389 - val_loss: 29.0

    ====================] - 0s 4ms/step - loss: 29.7420 - mae: 4.0623 - val_loss: 28.5

    ====================] - 0s 5ms/step - loss: 28.8336 - mae: 4.0076 - val_loss: 26.0

    ====================] - 1s 16ms/step - loss: 590.4573 - mae: 22.3356 - val_loss: 4

    ====================] - 0s 6ms/step - loss: 502.5388 - mae: 20.1180 - val_loss: 38

    ====================] - 0s 6ms/step - loss: 389.6221 - mae: 16.9611 - val_loss: 27

    ====================] - 0s 6ms/step - loss: 269.4721 - mae: 13.2975 - val_loss: 16

    ====================] - 0s 6ms/step - loss: 180.0653 - mae: 10.5198 - val_loss: 11

    ====================] - 0s 7ms/step - loss: 145.7865 - mae: 9.4028 - val_loss: 100

    ====================] - 0s 4ms/step - loss: 131.0734 - mae: 8.8618 - val_loss: 91.

    ====================] - 0s 4ms/step - loss: 119.4486 - mae: 8.3976 - val_loss: 82.

    ====================] - 0s 5ms/step - loss: 108.4436 - mae: 7.8922 - val_loss: 75.

    ====================] - 0s 5ms/step - loss: 98.2123 - mae: 7.4117 - val_loss: 68.1

    ====================] - 0s 4ms/step - loss: 89.1501 - mae: 7.0333 - val_loss: 62.2

    ====================] - 0s 5ms/step - loss: 81.2326 - mae: 6.6914 - val_loss: 56.6

    ====================] - 0s 6ms/step - loss: 74.1657 - mae: 6.2428 - val_loss: 52.9

    ====================] - 0s 5ms/step - loss: 68.8585 - mae: 6.0110 - val_loss: 49.7

    ====================] - 0s 5ms/step - loss: 64.6758 - mae: 5.8987 - val_loss: 47.5

    ====================] - 0s 5ms/step - loss: 60.5561 - mae: 5.6157 - val_loss: 45.9

    ====================] - 0s 5ms/step - loss: 58.4106 - mae: 5.4483 - val_loss: 44.5

    ====================] - 0s 5ms/step - loss: 55.9230 - mae: 5.4467 - val_loss: 43.4

    ====================] - 0s 5ms/step - loss: 54.4081 - mae: 5.3288 - val_loss: 42.3

    ====================] - 0s 4ms/step - loss: 52.5693 - mae: 5.2328 - val_loss: 41.3

    ====================] - 0s 5ms/step - loss: 51.0509 - mae: 5.2227 - val_loss: 40.3

    ====================] - 0s 4ms/step - loss: 49.7852 - mae: 5.1054 - val_loss: 39.5
```

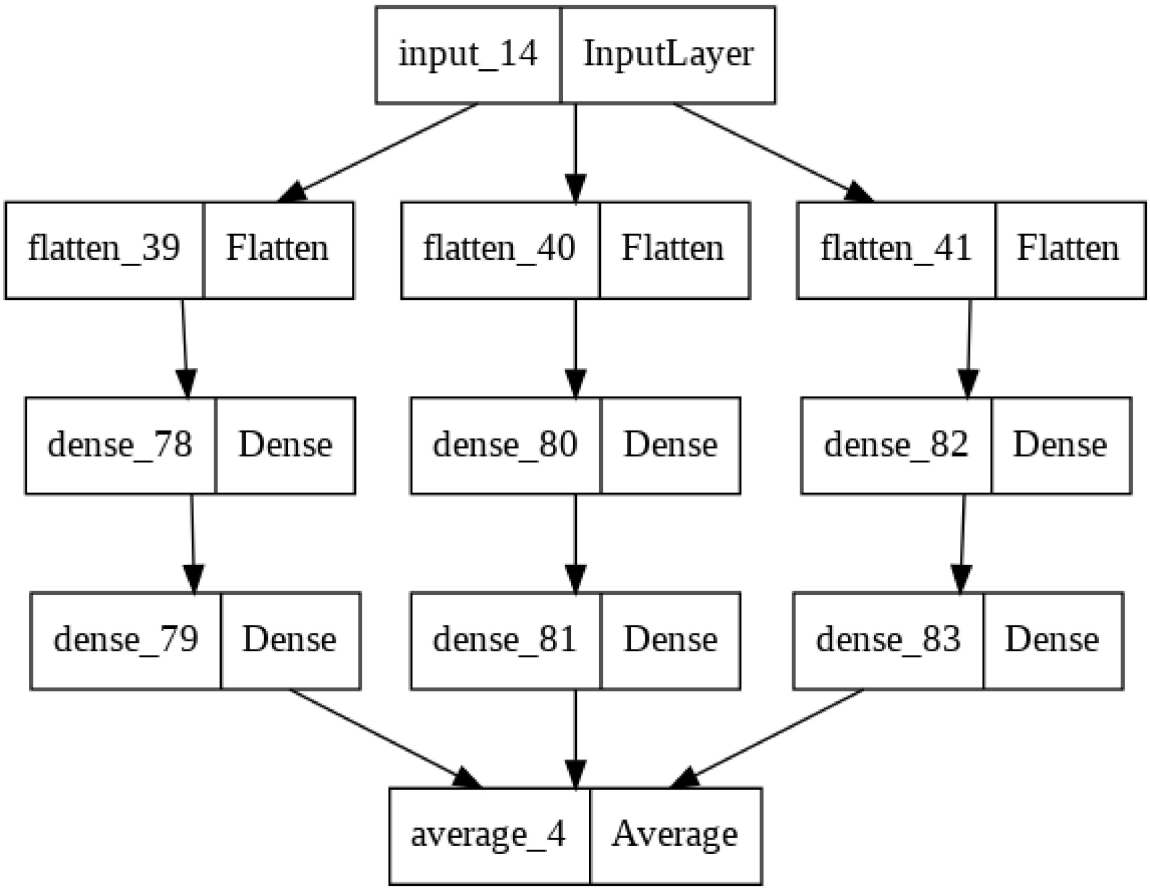
```
====================] - 0s 5ms/step - loss: 48.2005 - mae: 5.0168 - val_loss: 38.6

====================] - 0s 5ms/step - loss: 46.8253 - mae: 4.9980 - val_loss: 37.7

====================] - 0s 4ms/step - loss: 45.6065 - mae: 5.0078 - val_loss: 36.7
```

```
# сохранить модели на диск.
model1.save_weights('model1.h5')
model2.save_weights('model2.h5')
model3.save_weights('model3.h5')
# прочитать модели с диска (перед этим, нужно положить их в каталог для работы виртуальной
# model.load_weights('model.h5')


input1 = keras.layers.Input(shape=(X_train.shape[-1],))
x1= keras.layers.Flatten()(input1)
x1 = keras.layers.Dense(512, activation='relu')(x1)
x1 =keras.layers.Dense(1, activation='linear')(x1)

model11 =keras.models.Model(inputs=input1,outputs=x1)


x2= keras.layers.Flatten()(input1)
x2 = keras.layers.Dense(512, activation='relu')(x2)
x2=keras.layers.Dense(1, activation='linear')(x2)

model22 =keras.models.Model(inputs=input1,outputs=x2)

x3= keras.layers.Flatten()(input1)
x3 = keras.layers.Dense(512, activation='relu')(x3)
x3=keras.layers.Dense(1, activation='linear')(x3)


model33 =keras.models.Model(inputs=input1,outputs=x3)

# усредняем выходы сетей
out_all = keras.layers.Average()([model11.output,model22.output,model33.output])
# out_all = keras.layers.ReLU()(out_all)

model_all=keras.models.Model(inputs =[input1],outputs =out_all)
model_all.compile(optimizer=tf.keras.optimizers.Adagrad(lr=0.1, epsilon=1e-08, decay=0.0),
              # loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              loss= 'mse',
              metrics=['mae'])


model11.load_weights('model1.h5')
model22.load_weights('model2.h5')
model33.load_weights('model3.h5')
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adagrad.py:74: UserWarning
  super(Adagrad, self).__init__(name, **kwargs)
```

```
from tensorflow.keras.utils import plot_model
```

```
plot_model(model_all,to_file='new_model-all.png')
```




```
model_all.evaluate(X_test,y_test)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 155.5349 - mae: 10.6417
[155.53494262695312, 10.641709327697754]
```

```
y_pred = model_all.predict(X_test)
```

```
y_pred[0]
```

```
array([12.664565], dtype=float32)
```

✓ 0 сек. выполнено в 23:47