Нейроинформатика. Лабораторная работа 2

Линейная нейронная сеть. Правило обучения Уидроу-Хоффа

Целью работы является исследование свойств линейной нейронной сети и алгоритмов ее обучения, применение сети в задачах аппроксимации и фильтрации.

Выполнил Лисин Роман, М8О-406Б-20

```
import numpy as np

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

import matplotlib.pyplot as plt
```

Функции сигналов и параметры:

```
def in1(t):
    return np.sin(t * t - 15 * t + 3) - np.sin(t) * np.sin(t)

def in2(t):
    return np.cos(t * t)

def out(t):
    return np.cos(t * t + 2 * np.pi) / 2

h1 = 0.01
h2 = 0.02

range1 = (0.5, 3)
range2 = (0, 4)
```

Задание 1

Попробуем предсказать следующий элемент последовательности

Сгенерируем датасет для обучения

```
t1 = np.linspace(range1[0], range1[1], int((range1[1] - range1[0]) /
h1))
x1 = in1(t1)

def gen_dataset(x, delay=5):
    x_train = np.array([np.hstack([x[i:i+delay]]) for i in
range(len(x) - delay)])
```

```
    y_train = x[delay:]
    assert x_train.shape[0] == y_train.shape[0]
    return x_train, y_train

x_train1, y_train1 = gen_dataset(x1)
x_train1.shape, y_train1.shape
```

```
((245, 5), (245,))
```

Убедимся, что датасеты сгенерились правильно

```
x_train1[:3], y_train1[:3]
```

```
(array([[0.66514051, 0.71027607, 0.73658461, 0.74346538, 0.73070012],
        [0.71027607, 0.73658461, 0.74346538, 0.73070012, 0.69845359],
        [0.73658461, 0.74346538, 0.73070012, 0.69845359,
0.64726658]]),
 array([0.69845359, 0.64726658, 0.57804171]))
```

Все корректно, можем приступать к обучению перцептрона

```
model1 = keras.Sequential()
model1.add(keras.layers.Dense(1))

model1.compile(loss='mse', optimizer='adam',
metrics=tf.keras.metrics.RootMeanSquaredError())

train_info1 = model1.fit(x_train1, y_train1, batch_size=1, epochs=50)
```

```
Epoch 1/50
245/245 [==============================] - 1s 1ms/step - loss: 3.8405
- root_mean_squared_error: 1.9597
Epoch 2/50
245/245 [==============================] - 0s 1ms/step - loss: 1.5399
- root_mean_squared_error: 1.2409
Epoch 3/50
245/245 [==============================] - 0s 993us/step - loss:
0.5419 - root_mean_squared_error: 0.7362
Epoch 4/50
245/245 [==============================] - 0s 1ms/step - loss: 0.1872
- root_mean_squared_error: 0.4327
Epoch 5/50
245/245 [==============================] - 0s 978us/step - loss:
0.0854 - root_mean_squared_error: 0.2922
Epoch 6/50
245/245 [==============================] - 0s 982us/step - loss:
0.0565 - root_mean_squared_error: 0.2376
Epoch 7/50
245/245 [==============================] - 0s 997us/step - loss:
0.0440 - root_mean_squared_error: 0.2098
```

```
Epoch 8/50
245/245 [==============================] - 0s 950us/step - loss:
0.0356 - root_mean_squared_error: 0.1887
Epoch 9/50
245/245 [==============================] - 0s 990us/step - loss:
0.0292 - root_mean_squared_error: 0.1709
Epoch 10/50
245/245 [==============================] - 0s 945us/step - loss:
0.0244 - root_mean_squared_error: 0.1563
Epoch 11/50
245/245 [==============================] - 0s 946us/step - loss:
0.0209 - root_mean_squared_error: 0.1445
Epoch 12/50
245/245 [==============================] - 0s 998us/step - loss:
0.0183 - root_mean_squared_error: 0.1352
Epoch 13/50
245/245 [==============================] - 0s 984us/step - loss:
0.0164 - root_mean_squared_error: 0.1281
Epoch 14/50
245/245 [==============================] - 0s 977us/step - loss:
0.0148 - root_mean_squared_error: 0.1219
Epoch 15/50
245/245 [==============================] - 0s 944us/step - loss:
0.0135 - root_mean_squared_error: 0.1164
Epoch 16/50
245/245 [==============================] - 0s 945us/step - loss:
0.0124 - root_mean_squared_error: 0.1112
Epoch 17/50
245/245 [==============================] - 0s 925us/step - loss:
0.0112 - root_mean_squared_error: 0.1058
Epoch 18/50
245/245 [==============================] - 0s 995us/step - loss:
0.0099 - root_mean_squared_error: 0.0995
Epoch 19/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0088
- root_mean_squared_error: 0.0939
Epoch 20/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0076
- root_mean_squared_error: 0.0872
Epoch 21/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0067
- root_mean_squared_error: 0.0817
Epoch 22/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0056
- root_mean_squared_error: 0.0751
Epoch 23/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0047
- root_mean_squared_error: 0.0686
Epoch 24/50
```

```
245/245 [==============================] - 0s 1ms/step - loss: 0.0039
- root_mean_squared_error: 0.0624
Epoch 25/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0032
- root_mean_squared_error: 0.0564
Epoch 26/50
245/245 [==============================] - 0s 1ms/step - loss: 0.0026
- root_mean_squared_error: 0.0508
Epoch 27/50
245/245 [==============================] - 0s 999us/step - loss:
0.0021 - root_mean_squared_error: 0.0459
Epoch 28/50
245/245 [==============================] - 0s 967us/step - loss:
0.0017 - root_mean_squared_error: 0.0407
Epoch 29/50
245/245 [==============================] - 0s 1000us/step - loss:
0.0013 - root_mean_squared_error: 0.0366
Epoch 30/50
245/245 [==============================] - 0s 961us/step - loss:
0.0011 - root_mean_squared_error: 0.0333
Epoch 31/50
245/245 [==============================] - 0s 951us/step - loss:
9.4333e-04 - root_mean_squared_error: 0.0307
Epoch 32/50
245/245 [==============================] - 0s 924us/step - loss:
8.4196e-04 - root_mean_squared_error: 0.0290
Epoch 33/50
245/245 [==============================] - 0s 950us/step - loss:
7.6273e-04 - root_mean_squared_error: 0.0276
Epoch 34/50
245/245 [==============================] - 0s 968us/step - loss:
7.2365e-04 - root_mean_squared_error: 0.0269
Epoch 35/50
245/245 [==============================] - 0s 991us/step - loss:
6.9109e-04 - root_mean_squared_error: 0.0263
Epoch 36/50
245/245 [==============================] - 0s 1ms/step - loss:
6.8424e-04 - root_mean_squared_error: 0.0262
Epoch 37/50
245/245 [==============================] - 0s 948us/step - loss:
6.7166e-04 - root_mean_squared_error: 0.0259
Epoch 38/50
245/245 [==============================] - 0s 985us/step - loss:
6.8549e-04 - root_mean_squared_error: 0.0262
Epoch 39/50
245/245 [==============================] - 0s 1ms/step - loss:
6.7363e-04 - root_mean_squared_error: 0.0260
Epoch 40/50
245/245 [==============================] - 0s 935us/step - loss:
```

```
6.5737e-04 - root_mean_squared_error: 0.0256
Epoch 41/50
245/245 [==============================] - 0s 978us/step - loss:
7.1418e-04 - root_mean_squared_error: 0.0267
Epoch 42/50
245/245 [==============================] - 0s 1ms/step - loss:
6.6976e-04 - root_mean_squared_error: 0.0259
Epoch 43/50
245/245 [==============================] - 0s 1ms/step - loss:
7.1743e-04 - root_mean_squared_error: 0.0268
Epoch 44/50
245/245 [==============================] - 0s 1ms/step - loss:
6.9878e-04 - root_mean_squared_error: 0.0264
Epoch 45/50
245/245 [==============================] - 0s 963us/step - loss:
6.9120e-04 - root_mean_squared_error: 0.0263
Epoch 46/50
245/245 [==============================] - 0s 1ms/step - loss:
7.1798e-04 - root_mean_squared_error: 0.0268
Epoch 47/50
245/245 [==============================] - 0s 950us/step - loss:
6.7132e-04 - root_mean_squared_error: 0.0259
Epoch 48/50
245/245 [==============================] - 0s 2ms/step - loss:
6.8187e-04 - root_mean_squared_error: 0.0261
Epoch 49/50
245/245 [==============================] - 0s 2ms/step - loss:
6.8167e-04 - root_mean_squared_error: 0.0261
Epoch 50/50
245/245 [==============================] - 0s 2ms/step - loss:
6.7384e-04 - root_mean_squared_error: 0.0260
```

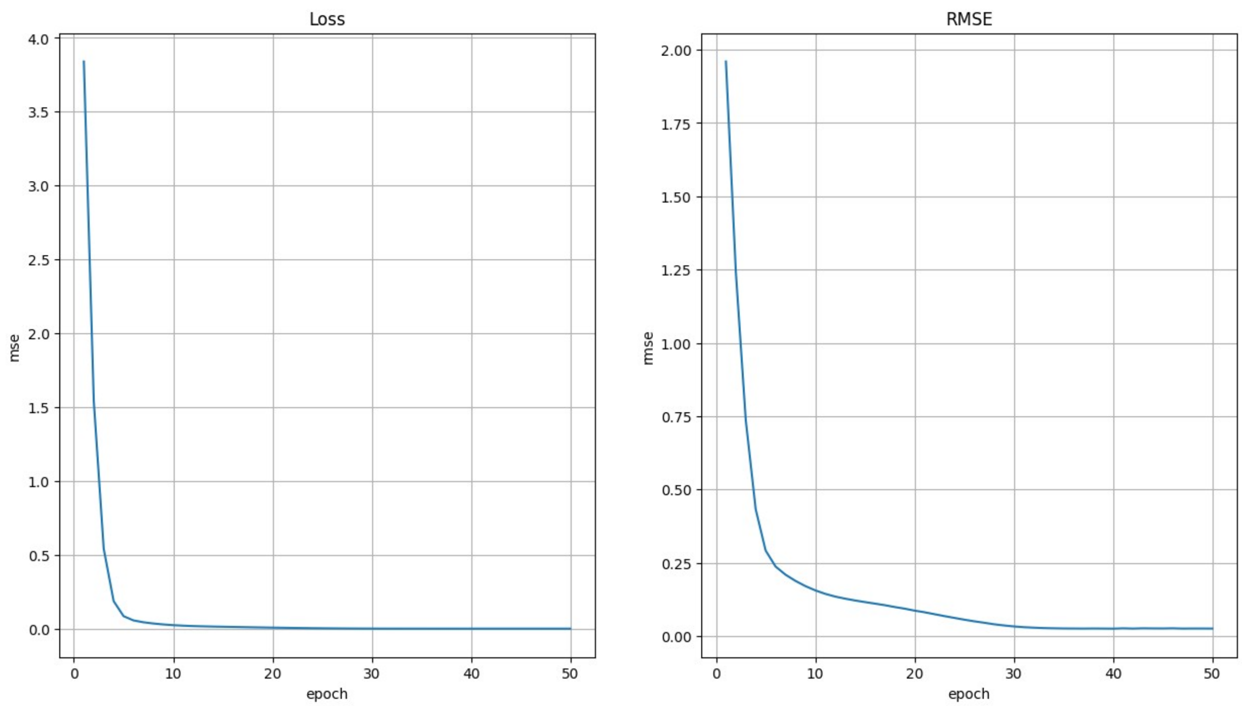
```
model1.layers[0].get_weights()
```

```
[array([[-0.76960367],
        [-0.49731734],
        [ 1.2077867 ],
        [ 1.1024797 ],
        [-0.11325198]], dtype=float32),
 array([-0.03841008], dtype=float32)]
```

Посмотрим на графики лосса и RMSE

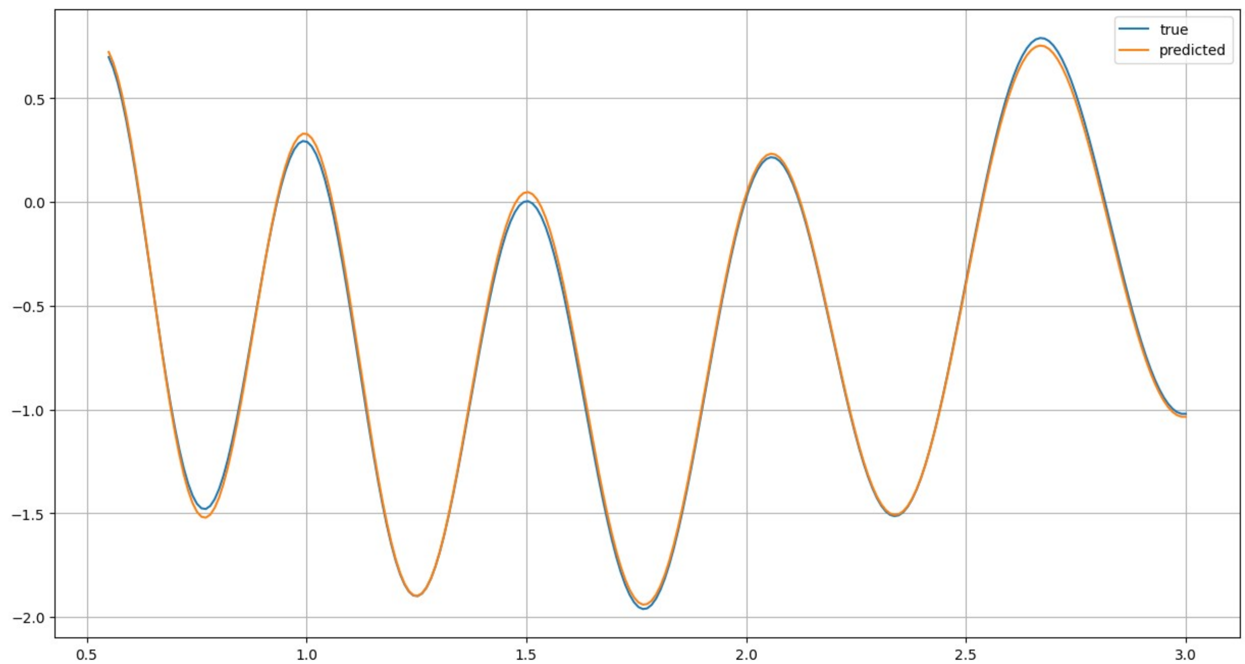
```
def plot_metrics(train_info):
    plt.figure(figsize=(15, 8))

    plt.subplot(1, 2, 1)
    loss_history = train_info.history['loss']
    plt.xlabel('epoch')
```

```
    plt.ylabel('mse')
    plt.plot(range(1, len(loss_history) + 1), loss_history)
    plt.grid()
    plt.title('Loss')

    plt.subplot(1, 2, 2)
    loss_history = train_info.history['root_mean_squared_error']
    plt.xlabel('epoch')
    plt.ylabel('rmse')
    plt.plot(range(1, len(loss_history) + 1), loss_history)
    plt.grid()
    plt.title('RMSE')

    plt.show()

plot_metrics(train_info1)
```



Посмотрим на результат работы модели

```
plt.figure(figsize=(15, 8))

plt.plot(t1[5:], x1[5:], label='true')
plt.plot(t1[5:], model1.predict(x_train1), label='predicted')
plt.legend()
plt.grid()
plt.show()

8/8 [==============================] - 0s 2ms/step
```



Модель довольно неплохо научилась предсказывать следующую точку

Задание 2

Попробуем сделать многошаговый прогноз

Сначала обучим модель с задержкой = 3

```
x_train2, y_train2 = gen_dataset(x1, delay=3)
x_train2.shape, y_train2.shape
```

```
((247, 3), (247,))
```

```
model2 = keras.Sequential()
model2.add(keras.layers.Dense(1))
```

```
model2.compile(loss='mse', optimizer='adam',
metrics=tf.keras.metrics.RootMeanSquaredError())
```

```
train_info2 = model2.fit(x_train2, y_train2, batch_size=1, epochs=100)
```

```
Epoch 1/100
247/247 [==============================] - 1s 1ms/step - loss: 0.2856
- root_mean_squared_error: 0.5344
Epoch 2/100
247/247 [==============================] - 0s 990us/step - loss:
0.0963 - root_mean_squared_error: 0.3102
Epoch 3/100
247/247 [==============================] - 0s 961us/step - loss:
0.0738 - root_mean_squared_error: 0.2717
Epoch 4/100
```

```
247/247 [==============================] - 0s 1ms/step - loss: 0.0706
- root_mean_squared_error: 0.2657
Epoch 5/100
247/247 [==============================] - 0s 987us/step - loss:
0.0681 - root_mean_squared_error: 0.2610
Epoch 6/100
247/247 [==============================] - 0s 990us/step - loss:
0.0667 - root_mean_squared_error: 0.2583
Epoch 7/100
247/247 [==============================] - 0s 943us/step - loss:
0.0649 - root_mean_squared_error: 0.2548
Epoch 8/100
247/247 [==============================] - 0s 967us/step - loss:
0.0636 - root_mean_squared_error: 0.2522
Epoch 9/100
247/247 [==============================] - 0s 985us/step - loss:
0.0619 - root_mean_squared_error: 0.2488
Epoch 10/100
247/247 [==============================] - 0s 987us/step - loss:
0.0601 - root_mean_squared_error: 0.2451
Epoch 11/100
247/247 [==============================] - 0s 984us/step - loss:
0.0588 - root_mean_squared_error: 0.2424
Epoch 12/100
247/247 [==============================] - 0s 969us/step - loss:
0.0571 - root_mean_squared_error: 0.2390
Epoch 13/100
247/247 [==============================] - 0s 959us/step - loss:
0.0557 - root_mean_squared_error: 0.2360
Epoch 14/100
247/247 [==============================] - 0s 969us/step - loss:
0.0541 - root_mean_squared_error: 0.2325
Epoch 15/100
247/247 [==============================] - 0s 958us/step - loss:
0.0524 - root_mean_squared_error: 0.2289
Epoch 16/100
247/247 [==============================] - 0s 918us/step - loss:
0.0506 - root_mean_squared_error: 0.2249
Epoch 17/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0494
- root_mean_squared_error: 0.2222
Epoch 18/100
247/247 [==============================] - 0s 934us/step - loss:
0.0478 - root_mean_squared_error: 0.2186
Epoch 19/100
247/247 [==============================] - 0s 927us/step - loss:
0.0462 - root_mean_squared_error: 0.2151
Epoch 20/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0448
```

```
- root_mean_squared_error: 0.2117
Epoch 21/100
247/247 [==============================] - 0s 991us/step - loss:
0.0433 - root_mean_squared_error: 0.2081
Epoch 22/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0418
- root_mean_squared_error: 0.2044
Epoch 23/100
247/247 [==============================] - 0s 1000us/step - loss:
0.0408 - root_mean_squared_error: 0.2019
Epoch 24/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0391
- root_mean_squared_error: 0.1979
Epoch 25/100
247/247 [==============================] - 0s 934us/step - loss:
0.0379 - root_mean_squared_error: 0.1947
Epoch 26/100
247/247 [==============================] - 0s 942us/step - loss:
0.0370 - root_mean_squared_error: 0.1922
Epoch 27/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0348
- root_mean_squared_error: 0.1864
Epoch 28/100
247/247 [==============================] - 0s 916us/step - loss:
0.0336 - root_mean_squared_error: 0.1834
Epoch 29/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0326
- root_mean_squared_error: 0.1806
Epoch 30/100
247/247 [==============================] - 0s 965us/step - loss:
0.0313 - root_mean_squared_error: 0.1770
Epoch 31/100
247/247 [==============================] - 0s 962us/step - loss:
0.0301 - root_mean_squared_error: 0.1734
Epoch 32/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0288
- root_mean_squared_error: 0.1696
Epoch 33/100
247/247 [==============================] - 0s 913us/step - loss:
0.0279 - root_mean_squared_error: 0.1669
Epoch 34/100
247/247 [==============================] - 0s 928us/step - loss:
0.0265 - root_mean_squared_error: 0.1629
Epoch 35/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0257
- root_mean_squared_error: 0.1604
Epoch 36/100
247/247 [==============================] - 0s 916us/step - loss:
0.0244 - root_mean_squared_error: 0.1561
```

```
Epoch 37/100
247/247 [==============================] - 0s 986us/step - loss:
0.0234 - root_mean_squared_error: 0.1529
Epoch 38/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0224
- root_mean_squared_error: 0.1496
Epoch 39/100
247/247 [==============================] - 0s 939us/step - loss:
0.0215 - root_mean_squared_error: 0.1465
Epoch 40/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0202
- root_mean_squared_error: 0.1422
Epoch 41/100
247/247 [==============================] - 0s 928us/step - loss:
0.0193 - root_mean_squared_error: 0.1391
Epoch 42/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0184
- root_mean_squared_error: 0.1357
Epoch 43/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0176
- root_mean_squared_error: 0.1327
Epoch 44/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0166
- root_mean_squared_error: 0.1288
Epoch 45/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0159
- root_mean_squared_error: 0.1261
Epoch 46/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0150
- root_mean_squared_error: 0.1227
Epoch 47/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0143
- root_mean_squared_error: 0.1195
Epoch 48/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0134
- root_mean_squared_error: 0.1157
Epoch 49/100
247/247 [==============================] - 0s 991us/step - loss:
0.0127 - root_mean_squared_error: 0.1126
Epoch 50/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0120
- root_mean_squared_error: 0.1097
Epoch 51/100
247/247 [==============================] - 0s 998us/step - loss:
0.0113 - root_mean_squared_error: 0.1061
Epoch 52/100
247/247 [==============================] - 0s 996us/step - loss:
0.0106 - root_mean_squared_error: 0.1029
Epoch 53/100
```

```
247/247 [==============================] - 0s 966us/step - loss:
0.0098 - root_mean_squared_error: 0.0990
Epoch 54/100
247/247 [==============================] - 0s 969us/step - loss:
0.0094 - root_mean_squared_error: 0.0967
Epoch 55/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0086
- root_mean_squared_error: 0.0929
Epoch 56/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0080
- root_mean_squared_error: 0.0896
Epoch 57/100
247/247 [==============================] - 0s 930us/step - loss:
0.0075 - root_mean_squared_error: 0.0864
Epoch 58/100
247/247 [==============================] - 0s 977us/step - loss:
0.0070 - root_mean_squared_error: 0.0838
Epoch 59/100
247/247 [==============================] - 0s 962us/step - loss:
0.0064 - root_mean_squared_error: 0.0801
Epoch 60/100
247/247 [==============================] - 0s 943us/step - loss:
0.0060 - root_mean_squared_error: 0.0776
Epoch 61/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0055
- root_mean_squared_error: 0.0743
Epoch 62/100
247/247 [==============================] - 0s 910us/step - loss:
0.0051 - root_mean_squared_error: 0.0711
Epoch 63/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0047
- root_mean_squared_error: 0.0682
Epoch 64/100
247/247 [==============================] - 0s 967us/step - loss:
0.0042 - root_mean_squared_error: 0.0649
Epoch 65/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0039
- root_mean_squared_error: 0.0623
Epoch 66/100
247/247 [==============================] - 0s 973us/step - loss:
0.0035 - root_mean_squared_error: 0.0588
Epoch 67/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0031
- root_mean_squared_error: 0.0557
Epoch 68/100
247/247 [==============================] - 0s 939us/step - loss:
0.0028 - root_mean_squared_error: 0.0531
Epoch 69/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0025
```

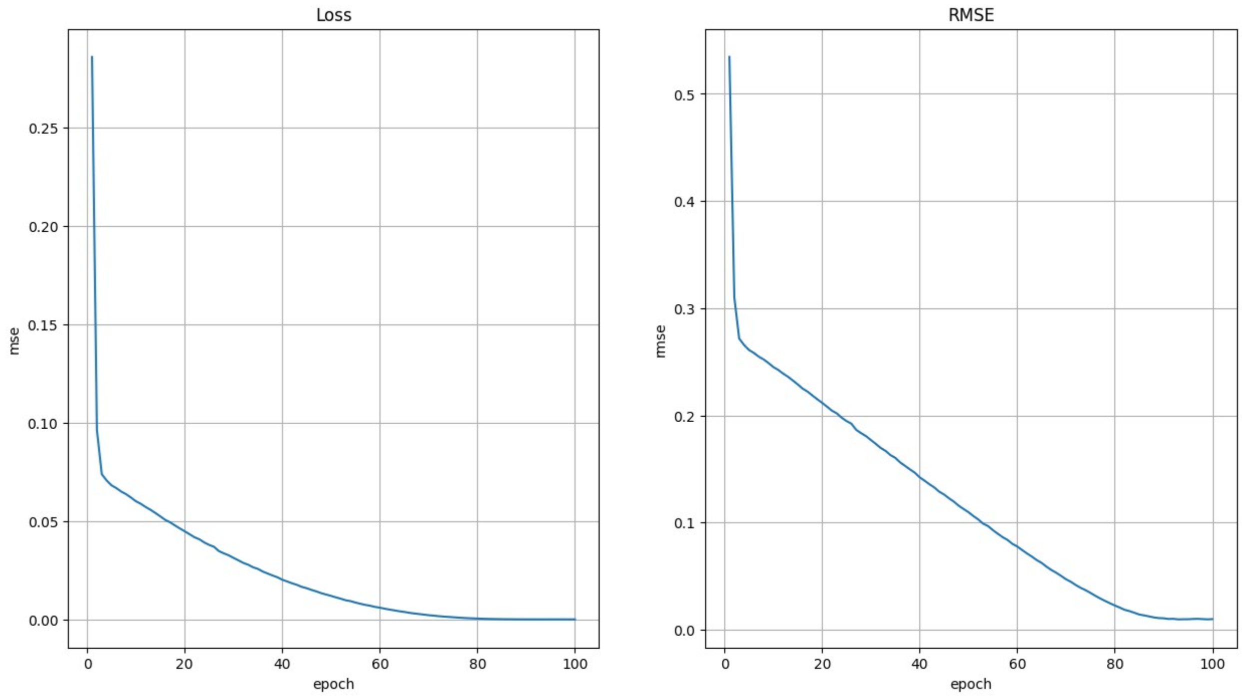
```
- root_mean_squared_error: 0.0502
Epoch 70/100
247/247 [==============================] - 0s 960us/step - loss:
0.0022 - root_mean_squared_error: 0.0471
Epoch 71/100
247/247 [==============================] - 0s 960us/step - loss:
0.0020 - root_mean_squared_error: 0.0447
Epoch 72/100
247/247 [==============================] - 0s 967us/step - loss:
0.0017 - root_mean_squared_error: 0.0417
Epoch 73/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0015
- root_mean_squared_error: 0.0391
Epoch 74/100
247/247 [==============================] - 0s 2ms/step - loss: 0.0014
- root_mean_squared_error: 0.0369
Epoch 75/100
247/247 [==============================] - 0s 1ms/step - loss: 0.0012
- root_mean_squared_error: 0.0344
Epoch 76/100
247/247 [==============================] - 0s 962us/step - loss:
0.0010 - root_mean_squared_error: 0.0317
Epoch 77/100
247/247 [==============================] - 0s 989us/step - loss:
8.5060e-04 - root_mean_squared_error: 0.0292
Epoch 78/100
247/247 [==============================] - 0s 997us/step - loss:
7.2212e-04 - root_mean_squared_error: 0.0269
Epoch 79/100
247/247 [==============================] - 0s 905us/step - loss:
6.0863e-04 - root_mean_squared_error: 0.0247
Epoch 80/100
247/247 [==============================] - 0s 1ms/step - loss:
5.1106e-04 - root_mean_squared_error: 0.0226
Epoch 81/100
247/247 [==============================] - 0s 988us/step - loss:
4.2827e-04 - root_mean_squared_error: 0.0207
Epoch 82/100
247/247 [==============================] - 0s 982us/step - loss:
3.4479e-04 - root_mean_squared_error: 0.0186
Epoch 83/100
247/247 [==============================] - 0s 961us/step - loss:
3.0174e-04 - root_mean_squared_error: 0.0174
Epoch 84/100
247/247 [==============================] - 0s 987us/step - loss:
2.5025e-04 - root_mean_squared_error: 0.0158
Epoch 85/100
247/247 [==============================] - 0s 994us/step - loss:
2.0170e-04 - root_mean_squared_error: 0.0142
```

```
Epoch 86/100
247/247 [==============================] - 0s 1ms/step - loss:
1.7792e-04 - root_mean_squared_error: 0.0133
Epoch 87/100
247/247 [==============================] - 0s 1ms/step - loss:
1.5446e-04 - root_mean_squared_error: 0.0124
Epoch 88/100
247/247 [==============================] - 0s 1ms/step - loss:
1.3152e-04 - root_mean_squared_error: 0.0115
Epoch 89/100
247/247 [==============================] - 0s 1ms/step - loss:
1.1756e-04 - root_mean_squared_error: 0.0108
Epoch 90/100
247/247 [==============================] - 0s 1ms/step - loss:
1.1349e-04 - root_mean_squared_error: 0.0107
Epoch 91/100
247/247 [==============================] - 1s 3ms/step - loss:
1.0044e-04 - root_mean_squared_error: 0.0100
Epoch 92/100
247/247 [==============================] - 0s 1ms/step - loss:
1.0304e-04 - root_mean_squared_error: 0.0102
Epoch 93/100
247/247 [==============================] - 1s 2ms/step - loss:
9.2654e-05 - root_mean_squared_error: 0.0096
Epoch 94/100
247/247 [==============================] - 0s 1ms/step - loss:
9.4798e-05 - root_mean_squared_error: 0.0097
Epoch 95/100
247/247 [==============================] - 0s 941us/step - loss:
9.4881e-05 - root_mean_squared_error: 0.0097
Epoch 96/100
247/247 [==============================] - 0s 955us/step - loss:
9.9928e-05 - root_mean_squared_error: 0.0100
Epoch 97/100
247/247 [==============================] - 0s 924us/step - loss:
1.0251e-04 - root_mean_squared_error: 0.0101
Epoch 98/100
247/247 [==============================] - 0s 996us/step - loss:
9.7744e-05 - root_mean_squared_error: 0.0099
Epoch 99/100
247/247 [==============================] - 0s 932us/step - loss:
9.3148e-05 - root_mean_squared_error: 0.0097
Epoch 100/100
247/247 [==============================] - 0s 994us/step - loss:
9.7447e-05 - root_mean_squared_error: 0.0099
```

```
model2.layers[0].get_weights()
```

```
[array([[-1.1373785 ],
        [ 1.2760266 ],
```

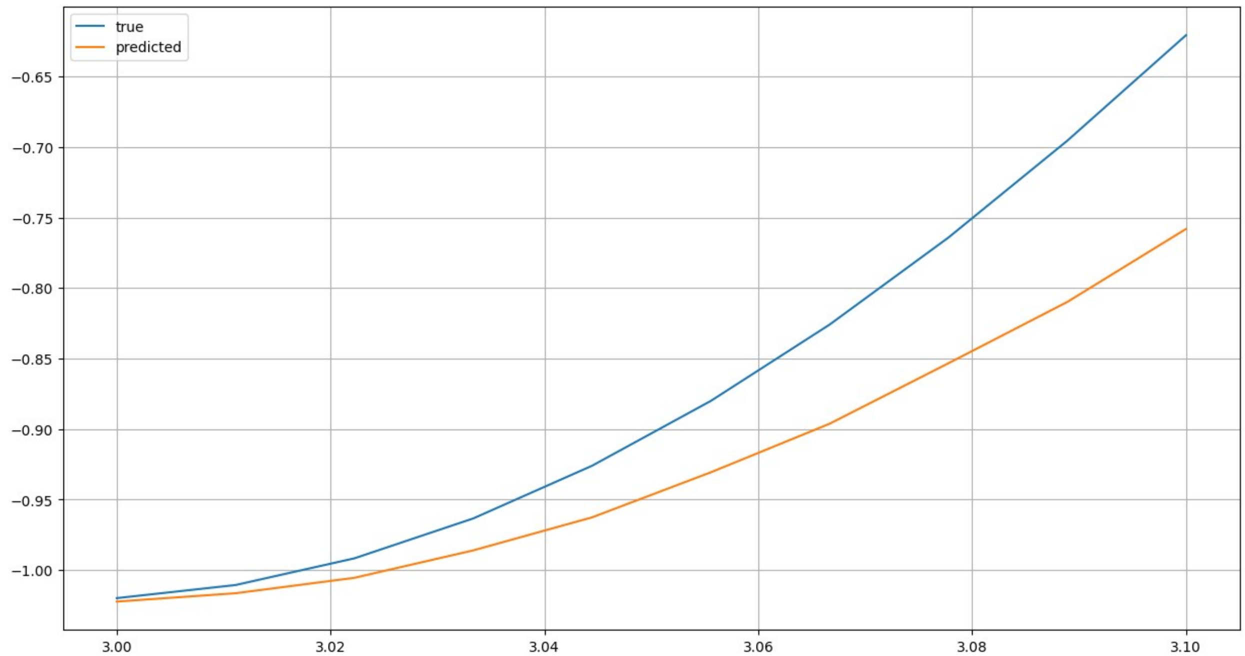
```
        [ 0.83491665]], dtype=float32),
 array([-0.01236548], dtype=float32)]

plot_metrics(train_info2)
```



Теперь сделаем прогноз на 10 шагов вперед

```
t_test = np.linspace(range1[1], range1[1] + 10 * h1, 10)
x_test = in1(t_test)

x_pred = x_train2[-1]
for i in range(10):
    x_pred = np.append(x_pred, model2.predict(np.expand_dims(x_pred[-
3:], axis=0)))

1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 16ms/step

plt.figure(figsize=(15, 8))
```

```
plt.plot(t_test, x_test, label='true')
plt.plot(t_test, x_pred[3:], label='predicted')
plt.legend()
plt.grid()
plt.show()
```



Здесь модель уже справилась похуже. На несколько шагов вперед она смотрит плохо

Задание 3

Попробуем обучить адаптивный линейный фильтр

```
t3 = np.linspace(range2[0], range2[1], int((range2[1] - range2[0]) /
h2))
x3 = in2(t3)
y3 = out(t3)

def gen_dataset_filter(x, y, delay=5):
    x_train = np.array([np.hstack([x[i:i+delay]]) for i in
range(len(x) - delay)])
    y_train = y[delay:]
    assert x_train.shape[0] == y_train.shape[0]
    return x_train, y_train

x_train3, y_train3 = gen_dataset_filter(x3, y3)
x_train3.shape, y_train3.shape
```

```
((195, 5), (195,))
```

Обучаем модель

```python
model3 = keras.Sequential()
model3.add(keras.layers.Dense(1))

model3.compile(loss='mse', optimizer='adam',
metrics=tf.keras.metrics.RootMeanSquaredError())

train_info3 = model3.fit(x_train3, y_train3, batch_size=1, epochs=50)
```

```
Epoch 1/50
195/195 [==============================] - 0s 919us/step - loss:
0.0049 - root_mean_squared_error: 0.0699
Epoch 2/50
195/195 [==============================] - 0s 924us/step - loss:
0.0015 - root_mean_squared_error: 0.0384
Epoch 3/50
195/195 [==============================] - 0s 943us/step - loss:
0.0011 - root_mean_squared_error: 0.0334
Epoch 4/50
195/195 [==============================] - 0s 1ms/step - loss:
8.5888e-04 - root_mean_squared_error: 0.0293
Epoch 5/50
195/195 [==============================] - 0s 1ms/step - loss:
6.8774e-04 - root_mean_squared_error: 0.0262
Epoch 6/50
195/195 [==============================] - 0s 1ms/step - loss:
6.0835e-04 - root_mean_squared_error: 0.0247
Epoch 7/50
195/195 [==============================] - 0s 934us/step - loss:
5.3744e-04 - root_mean_squared_error: 0.0232
Epoch 8/50
195/195 [==============================] - 0s 960us/step - loss:
4.7857e-04 - root_mean_squared_error: 0.0219
Epoch 9/50
195/195 [==============================] - 0s 891us/step - loss:
4.5739e-04 - root_mean_squared_error: 0.0214
Epoch 10/50
195/195 [==============================] - 0s 987us/step - loss:
4.5638e-04 - root_mean_squared_error: 0.0214
Epoch 11/50
195/195 [==============================] - 0s 1ms/step - loss:
4.3687e-04 - root_mean_squared_error: 0.0209
Epoch 12/50
195/195 [==============================] - 0s 885us/step - loss:
4.5637e-04 - root_mean_squared_error: 0.0214
Epoch 13/50
195/195 [==============================] - 0s 991us/step - loss:
4.7867e-04 - root_mean_squared_error: 0.0219
Epoch 14/50
195/195 [==============================] - 0s 972us/step - loss:
4.4918e-04 - root_mean_squared_error: 0.0212
```

```
Epoch 15/50
195/195 [==============================] - 0s 924us/step - loss:
4.8454e-04 - root_mean_squared_error: 0.0220
Epoch 16/50
195/195 [==============================] - 0s 1ms/step - loss:
4.5565e-04 - root_mean_squared_error: 0.0213
Epoch 17/50
195/195 [==============================] - 0s 897us/step - loss:
4.7485e-04 - root_mean_squared_error: 0.0218
Epoch 18/50
195/195 [==============================] - 0s 951us/step - loss:
4.7176e-04 - root_mean_squared_error: 0.0217
Epoch 19/50
195/195 [==============================] - 0s 989us/step - loss:
4.7075e-04 - root_mean_squared_error: 0.0217
Epoch 20/50
195/195 [==============================] - 0s 970us/step - loss:
4.1801e-04 - root_mean_squared_error: 0.0204
Epoch 21/50
195/195 [==============================] - 0s 995us/step - loss:
4.4212e-04 - root_mean_squared_error: 0.0210
Epoch 22/50
195/195 [==============================] - 0s 970us/step - loss:
4.5328e-04 - root_mean_squared_error: 0.0213
Epoch 23/50
195/195 [==============================] - 0s 895us/step - loss:
4.6490e-04 - root_mean_squared_error: 0.0216
Epoch 24/50
195/195 [==============================] - 0s 953us/step - loss:
4.5058e-04 - root_mean_squared_error: 0.0212
Epoch 25/50
195/195 [==============================] - 0s 960us/step - loss:
4.2800e-04 - root_mean_squared_error: 0.0207
Epoch 26/50
195/195 [==============================] - 0s 968us/step - loss:
4.6079e-04 - root_mean_squared_error: 0.0215
Epoch 27/50
195/195 [==============================] - 0s 1ms/step - loss:
4.2614e-04 - root_mean_squared_error: 0.0206
Epoch 28/50
195/195 [==============================] - 0s 989us/step - loss:
4.4599e-04 - root_mean_squared_error: 0.0211
Epoch 29/50
195/195 [==============================] - 0s 966us/step - loss:
4.3687e-04 - root_mean_squared_error: 0.0209
Epoch 30/50
195/195 [==============================] - 0s 994us/step - loss:
4.3791e-04 - root_mean_squared_error: 0.0209
Epoch 31/50
```

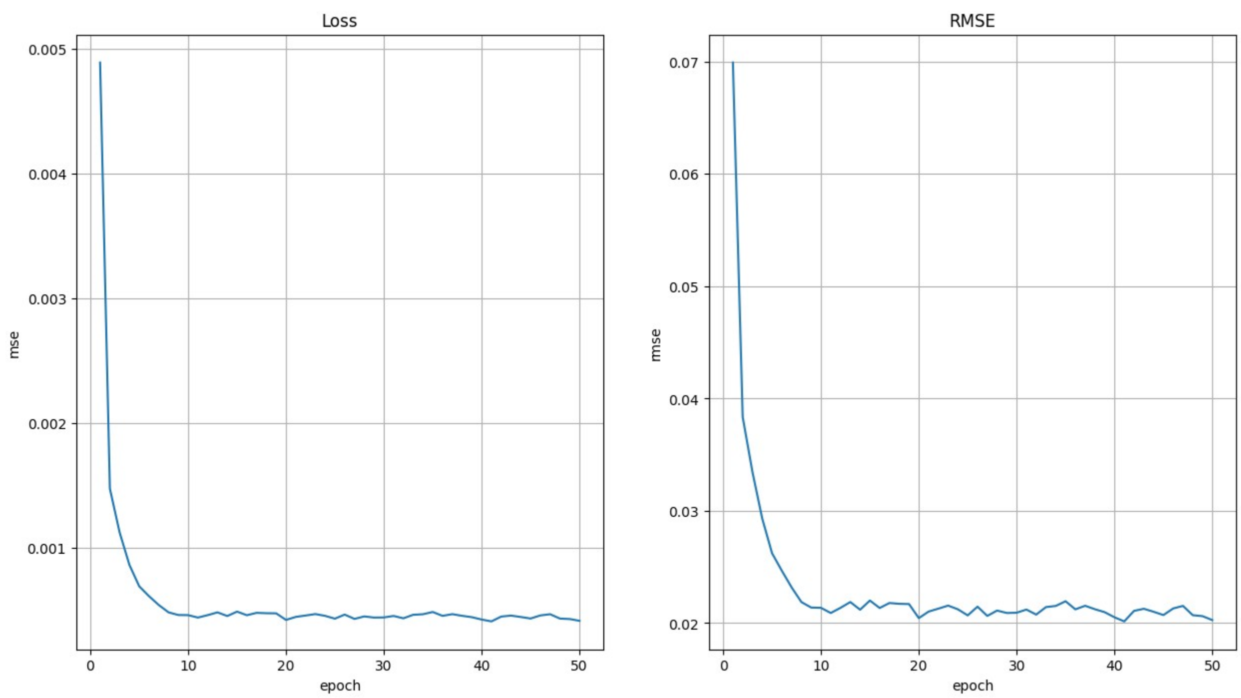
```
195/195 [==============================] - 0s 954us/step - loss:
4.4977e-04 - root_mean_squared_error: 0.0212
Epoch 32/50
195/195 [==============================] - 0s 1ms/step - loss:
4.3095e-04 - root_mean_squared_error: 0.0208
Epoch 33/50
195/195 [==============================] - 0s 961us/step - loss:
4.5919e-04 - root_mean_squared_error: 0.0214
Epoch 34/50
195/195 [==============================] - 0s 939us/step - loss:
4.6352e-04 - root_mean_squared_error: 0.0215
Epoch 35/50
195/195 [==============================] - 0s 1ms/step - loss:
4.8175e-04 - root_mean_squared_error: 0.0219
Epoch 36/50
195/195 [==============================] - 0s 947us/step - loss:
4.5058e-04 - root_mean_squared_error: 0.0212
Epoch 37/50
195/195 [==============================] - 0s 961us/step - loss:
4.6418e-04 - root_mean_squared_error: 0.0215
Epoch 38/50
195/195 [==============================] - 0s 980us/step - loss:
4.5100e-04 - root_mean_squared_error: 0.0212
Epoch 39/50
195/195 [==============================] - 0s 969us/step - loss:
4.3992e-04 - root_mean_squared_error: 0.0210
Epoch 40/50
195/195 [==============================] - 0s 912us/step - loss:
4.2102e-04 - root_mean_squared_error: 0.0205
Epoch 41/50
195/195 [==============================] - 0s 1ms/step - loss:
4.0618e-04 - root_mean_squared_error: 0.0202
Epoch 42/50
195/195 [==============================] - 0s 1ms/step - loss:
4.4482e-04 - root_mean_squared_error: 0.0211
Epoch 43/50
195/195 [==============================] - 0s 1ms/step - loss:
4.5272e-04 - root_mean_squared_error: 0.0213
Epoch 44/50
195/195 [==============================] - 0s 1ms/step - loss:
4.4147e-04 - root_mean_squared_error: 0.0210
Epoch 45/50
195/195 [==============================] - 0s 1ms/step - loss:
4.2921e-04 - root_mean_squared_error: 0.0207
Epoch 46/50
195/195 [==============================] - 0s 1ms/step - loss:
4.5393e-04 - root_mean_squared_error: 0.0213
Epoch 47/50
195/195 [==============================] - 0s 1ms/step - loss:
```

```
4.6373e-04 - root_mean_squared_error: 0.0215
Epoch 48/50
195/195 [==============================] - 0s 1ms/step - loss:
4.2873e-04 - root_mean_squared_error: 0.0207
Epoch 49/50
195/195 [==============================] - 0s 1ms/step - loss:
4.2519e-04 - root_mean_squared_error: 0.0206
Epoch 50/50
195/195 [==============================] - 0s 964us/step - loss:
4.1085e-04 - root_mean_squared_error: 0.0203
```

```
model3.layers[0].get_weights()
```

```
[array([[-0.6930382 ],
        [ 0.15167661],
        [ 0.62448573],
        [ 0.50529087],
        [-0.12660067]], dtype=float32),
 array([0.00452936], dtype=float32)]
```

```
plot_metrics(train_info3)
```

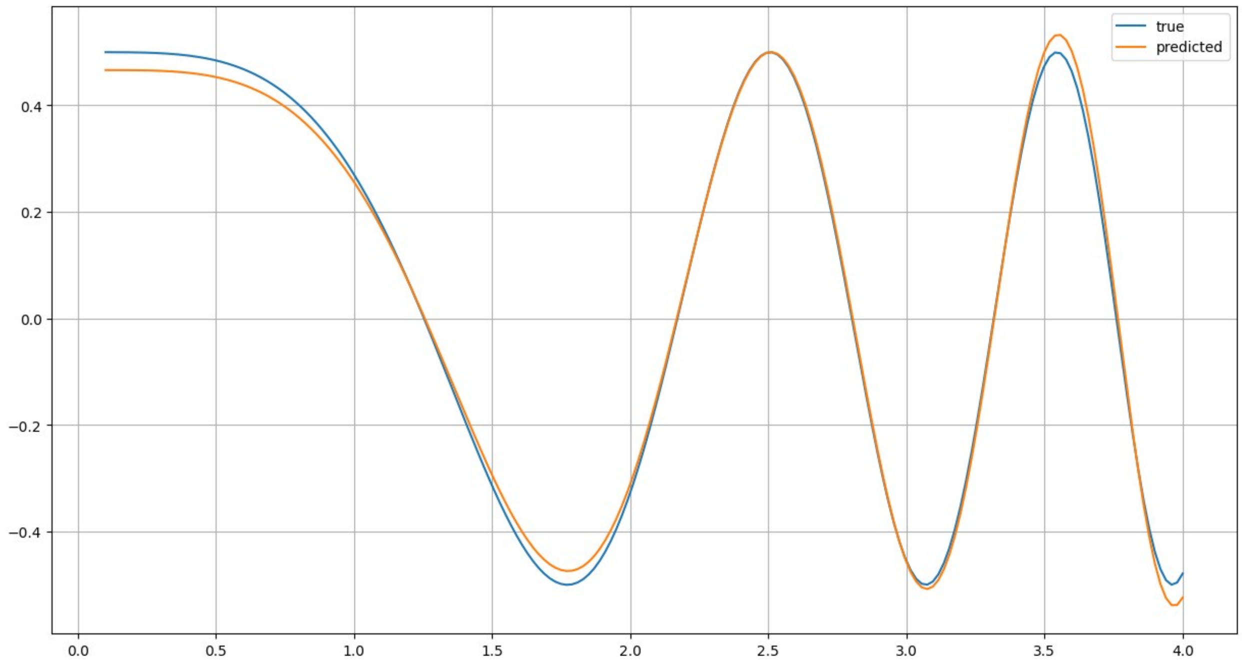


Посмотрим на результат модели

```
plt.figure(figsize=(15, 8))

plt.plot(t3[5:], out(t3[5:]), label='true')
plt.plot(t3[5:], model3.predict(x_train3), label='predicted')
plt.legend()
```

```
plt.grid()
plt.show()
```

```
7/7 [==============================] - 0s 1ms/step
```



Модель хорошо справилась с предсказанием значения выходного сигнала

Вывод

В данной работе я снова потренировался в обучении перцептронов. В этот раз я учил модель предсказывать следующее значение последовательности. Выяснил, что перцептрон хорошо учится предсказывать вперед на 1 шаг, но предсказывать на 10 шагов вперед получается плохо (из-за накапливаемой ошибки).

Также я попробовал реализовать свой адаптивный линейный фильтр. Результаты получились хорошие - перцептрон достаточно точно предсказывает значение выходного сигнала.