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.

```
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**2 - 10*t + 3)

def in2(t):
    return np.sin(-2*t**2 + 7*t)

def out(t):
    return np.sin(-2*t**2 + 7*t - np.pi) / 8

h1 = 0.025
h2 = 0.01

range1 = (1, 6)
range2 = (0, 3.5)
```

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
```

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

Убедимся, что датасеты сгенерились правильно

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

```
(array([[ 0.2794155 ,  0.08271696, -0.11603756, -0.30900637, -0.48881774],
        [ 0.08271696, -0.11603756, -0.30900637, -0.48881774, -0.64883879],
        [-0.11603756, -0.30900637, -0.48881774, -0.64883879, -0.78340034]]),
 array([-0.64883879, -0.78340034, -0.88797189]))
```

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
195/195 [==============================] - 2s 4ms/step - loss: 2.8989 - root_mean_squared_error: 1.7026
Epoch 2/50
195/195 [==============================] - 1s 4ms/step - loss: 1.7062 - root_mean_squared_error: 1.3062
Epoch 3/50
195/195 [==============================] - 0s 2ms/step - loss: 0.9485 - root_mean_squared_error: 0.9739
Epoch 4/50
195/195 [==============================] - 1s 3ms/step - loss: 0.5067 - root_mean_squared_error: 0.7118
Epoch 5/50
195/195 [==============================] - 1s 3ms/step - loss: 0.2672 - root_mean_squared_error: 0.5169
Epoch 6/50
195/195 [==============================] - 1s 3ms/step - loss: 0.1489 - root_mean_squared_error: 0.3859
Epoch 7/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0940 - root_mean_squared_error: 0.3066
Epoch 8/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0686 - root_mean_squared_error: 0.2619
Epoch 9/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0560 - root_mean_squared_error: 0.2366
Epoch 10/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0494 - root_mean_squared_error: 0.2222
Epoch 11/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0447 - root_mean_squared_error: 0.2115
Epoch 12/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0411 - root_mean_squared_error: 0.2028
Epoch 13/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0383 - root_mean_squared_error: 0.1957
Epoch 14/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0357 - root_mean_squared_error: 0.1890
Epoch 15/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0332 - root_mean_squared_error: 0.1822
Epoch 16/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0309 - root_mean_squared_error: 0.1757
Epoch 17/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0285 - root_mean_squared_error: 0.1689
Epoch 18/50
195/195 [==============================] - 0s 2ms/step - loss: 0.0261 - root_mean_squared_error: 0.1617
Epoch 19/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0238 - root_mean_squared_error: 0.1543
Epoch 20/50
195/195 [==============================] - 1s 4ms/step - loss: 0.0216 - root_mean_squared_error: 0.1469
Epoch 21/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0193 - root_mean_squared_error: 0.1390
Epoch 22/50
195/195 [==============================] - 1s 3ms/step - loss: 0.0172 - root_mean_squared_error: 0.1313
Epoch 23/50
195/195 [==============================] - 0s 1ms/step - loss: 0.0152 - root_mean_squared_error: 0.1232
Epoch 24/50
195/195 [==============================] - 0s 1ms/step - loss: 0.0132 - root_mean_squared_error: 0.1150
Epoch 25/50
195/195 [==============================] - 0s 1ms/step - loss: 0.0116 - root_mean_squared_error: 0.1075
Epoch 26/50
195/195 [==============================] - 0s 2ms/step - loss: 0.0098 - root_mean_squared_error: 0.0989
Epoch 27/50
195/195 [==============================] - 0s 1ms/step - loss: 0.0083 - root_mean_squared_error: 0.0913
Epoch 28/50
195/195 [==============================] - 0s 2ms/step - loss: 0.0070 - root_mean_squared_error: 0.0834
Epoch 29/50
195/195 [==============================] - 0s 2ms/step - loss: 0.0058 - root_mean_squared_error: 0.0759
```

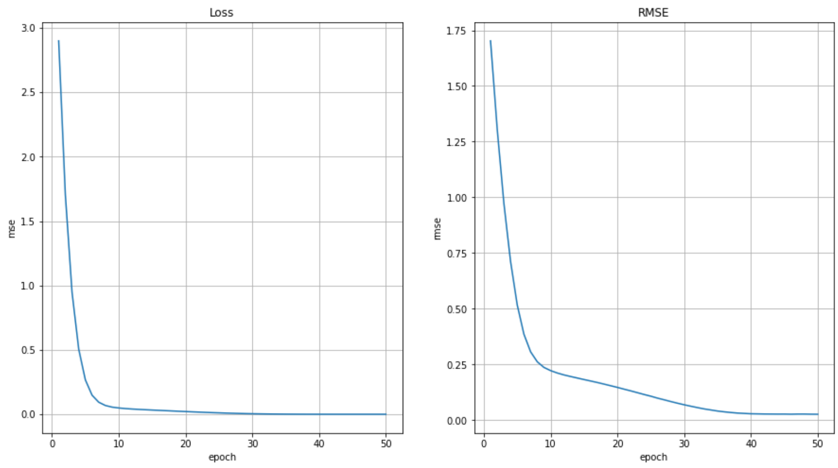
```
model1.layers[0].get_weights()
```

```
[array([[-0.5695189 ],
        [ 0.17684439],
        [ 0.19597706],
        [ 0.33547652],
        [ 0.7909496 ]], dtype=float32), array([-0.00295174], 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')
```

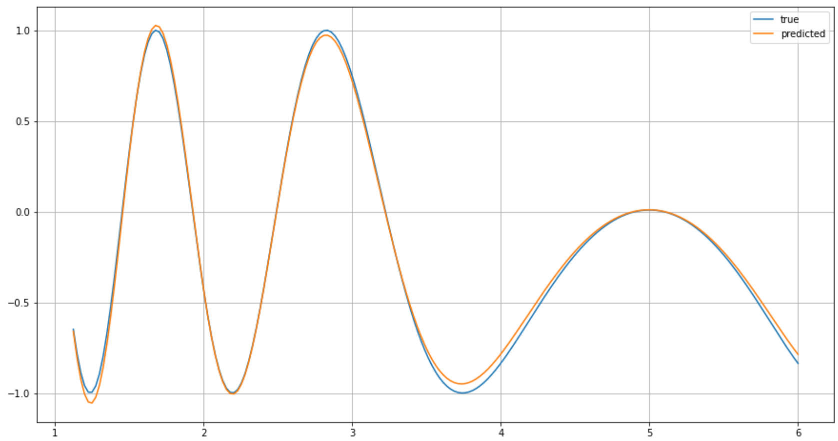
```
    plt.ylabel('rmse')
    plt.plot(range(1, len(loss_history) + 1), loss_history)
    plt.grid()
    plt.title('RMSE')

    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()
```

```
7/7 [==============================] - 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
```

```
((197, 3), (197,))
```

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

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

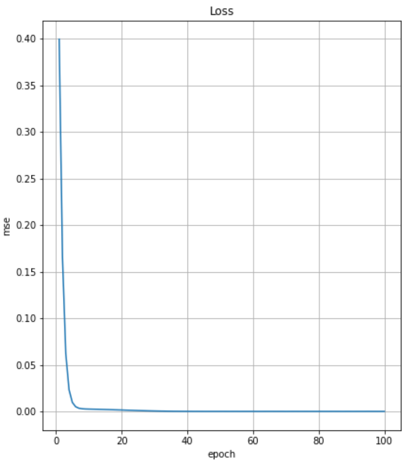
```
train_info2 = model2.fit(x_train2, y_train2, batch_size=1, epochs=100)
```

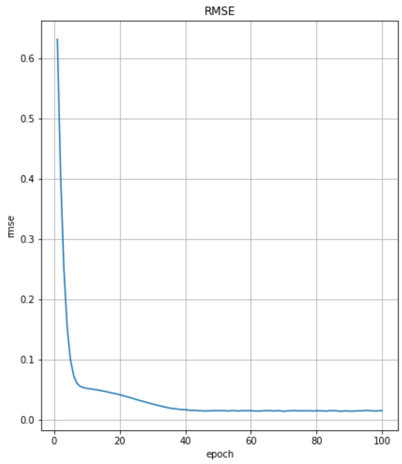
```
Epoch 1/100
197/197 [==============================] - 1s 1ms/step - loss: 0.3995 - root_mean_squared_error: 0.6320
Epoch 2/100
197/197 [==============================] - 0s 2ms/step - loss: 0.1652 - root_mean_squared_error: 0.4065
Epoch 3/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0624 - root_mean_squared_error: 0.2499
Epoch 4/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0235 - root_mean_squared_error: 0.1532
Epoch 5/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0099 - root_mean_squared_error: 0.0993
Epoch 6/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0053 - root_mean_squared_error: 0.0726
Epoch 7/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0036 - root_mean_squared_error: 0.0602
Epoch 8/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0030 - root_mean_squared_error: 0.0551
Epoch 9/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0028 - root_mean_squared_error: 0.0531
Epoch 10/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0027 - root_mean_squared_error: 0.0520
Epoch 11/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0026 - root_mean_squared_error: 0.0512
Epoch 12/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0025 - root_mean_squared_error: 0.0504
Epoch 13/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0024 - root_mean_squared_error: 0.0495
Epoch 14/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0024 - root_mean_squared_error: 0.0485
Epoch 15/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0023 - root_mean_squared_error: 0.0475
Epoch 16/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0021 - root_mean_squared_error: 0.0463
Epoch 17/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0020 - root_mean_squared_error: 0.0452
Epoch 18/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0019 - root_mean_squared_error: 0.0439
Epoch 19/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0018 - root_mean_squared_error: 0.0428
Epoch 20/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0017 - root_mean_squared_error: 0.0415
Epoch 21/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0016 - root_mean_squared_error: 0.0399
Epoch 22/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0015 - root_mean_squared_error: 0.0384
Epoch 23/100
197/197 [==============================] - 0s 2ms/step - loss: 0.0014 - root_mean_squared_error: 0.0369
Epoch 24/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0013 - root_mean_squared_error: 0.0354
Epoch 25/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0011 - root_mean_squared_error: 0.0336
Epoch 26/100
197/197 [==============================] - 0s 1ms/step - loss: 0.0010 - root_mean_squared_error: 0.0320
Epoch 27/100
197/197 [==============================] - 0s 1ms/step - loss: 9.3697e-04 - root_mean_squared_error: 0.0306
Epoch 28/100
197/197 [==============================] - 0s 1ms/step - loss: 8.4275e-04 - root_mean_squared_error: 0.0290
Epoch 29/100
197/197 [==============================] - 0s 1ms/step - loss: 7.6042e-04 - root_mean_squared_error: 0.0276
```

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

```
[array([[-0.94829816],
        [ 0.92858356],
        [ 0.99817014]], dtype=float32), array([-0.00529915], dtype=float32)]
```

```
plot_metrics(train_info2)
```



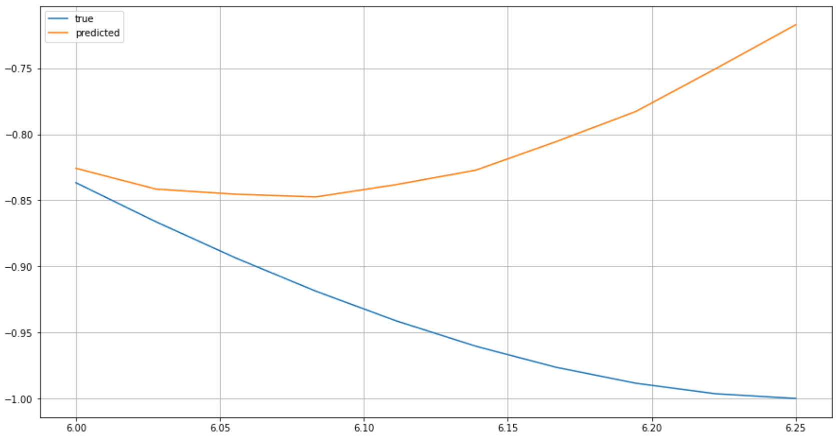


Теперь сделаем прогноз на 10 шагов вперед

```
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 44ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/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
```

```
((345, 5), (345,))
```

Обучаем модель

```
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)
```

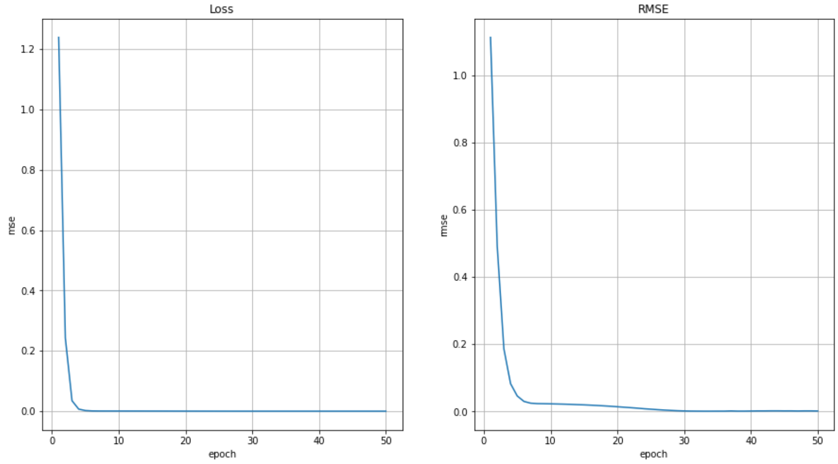
```
Epoch 1/50
345/345 [==============================] - 1s 2ms/step - loss: 1.2385 - root_mean_squared_error: 1.1129
Epoch 2/50
345/345 [==============================] - 1s 2ms/step - loss: 0.2425 - root_mean_squared_error: 0.4925
Epoch 3/50
345/345 [==============================] - 0s 1ms/step - loss: 0.0347 - root_mean_squared_error: 0.1863
Epoch 4/50
345/345 [==============================] - 0s 1ms/step - loss: 0.0068 - root_mean_squared_error: 0.0825
Epoch 5/50
345/345 [==============================] - 0s 1ms/step - loss: 0.0021 - root_mean_squared_error: 0.0460
Epoch 6/50
345/345 [==============================] - 0s 1ms/step - loss: 8.9095e-04 - root_mean_squared_error: 0.0298
Epoch 7/50
345/345 [==============================] - 0s 1ms/step - loss: 6.0463e-04 - root_mean_squared_error: 0.0246
Epoch 8/50
345/345 [==============================] - 0s 1ms/step - loss: 5.4232e-04 - root_mean_squared_error: 0.0233
Epoch 9/50
345/345 [==============================] - 0s 1ms/step - loss: 5.2782e-04 - root_mean_squared_error: 0.0230
Epoch 10/50
345/345 [==============================] - 0s 1ms/step - loss: 5.1099e-04 - root_mean_squared_error: 0.0226
Epoch 11/50
345/345 [==============================] - 0s 1ms/step - loss: 4.9075e-04 - root_mean_squared_error: 0.0222
Epoch 12/50
345/345 [==============================] - 0s 1ms/step - loss: 4.6870e-04 - root_mean_squared_error: 0.0216
Epoch 13/50
345/345 [==============================] - 0s 1ms/step - loss: 4.4229e-04 - root_mean_squared_error: 0.0210
Epoch 14/50
345/345 [==============================] - 0s 1ms/step - loss: 4.0814e-04 - root_mean_squared_error: 0.0202
Epoch 15/50
345/345 [==============================] - 0s 1ms/step - loss: 3.8239e-04 - root_mean_squared_error: 0.0196
Epoch 16/50
345/345 [==============================] - 0s 1ms/step - loss: 3.4817e-04 - root_mean_squared_error: 0.0187
Epoch 17/50
345/345 [==============================] - 0s 1ms/step - loss: 3.1194e-04 - root_mean_squared_error: 0.0177
Epoch 18/50
345/345 [==============================] - 0s 1ms/step - loss: 2.7329e-04 - root_mean_squared_error: 0.0165
Epoch 19/50
345/345 [==============================] - 1s 2ms/step - loss: 2.3596e-04 - root_mean_squared_error: 0.0154
Epoch 20/50
345/345 [==============================] - 1s 1ms/step - loss: 1.9340e-04 - root_mean_squared_error: 0.0139
Epoch 21/50
345/345 [==============================] - 0s 1ms/step - loss: 1.5695e-04 - root_mean_squared_error: 0.0125
Epoch 22/50
345/345 [==============================] - 0s 1ms/step - loss: 1.2404e-04 - root_mean_squared_error: 0.0111
Epoch 23/50
345/345 [==============================] - 0s 1ms/step - loss: 9.3846e-05 - root_mean_squared_error: 0.0097
Epoch 24/50
345/345 [==============================] - 0s 1ms/step - loss: 6.3681e-05 - root_mean_squared_error: 0.0080
Epoch 25/50
345/345 [==============================] - 0s 1ms/step - loss: 4.4042e-05 - root_mean_squared_error: 0.0066
Epoch 26/50
345/345 [==============================] - 1s 1ms/step - loss: 2.7322e-05 - root_mean_squared_error: 0.0052
Epoch 27/50
```

```
345/345 [==============================] - 0s 1ms/step - loss: 1.6333e-05 - root_mean_squared_error: 0.0040
Epoch 28/50
345/345 [==============================] - 1s 1ms/step - loss: 9.0061e-06 - root_mean_squared_error: 0.0030
Epoch 29/50
345/345 [==============================] - 1s 1ms/step - loss: 4.2728e-06 - root_mean_squared_error: 0.0021
```

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

```
[array([[-0.35170996],
        [ 0.25001252],
        [ 0.3431131 ],
        [ 0.09131388],
        [-0.45964116]], dtype=float32), array([-0.00181413], dtype=float32)]
```

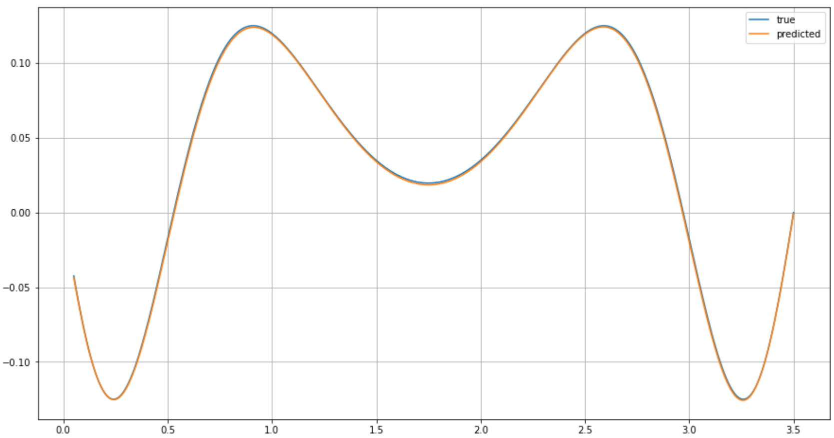
```
plot_metrics(train_info3)
```



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()
```

```
11/11 [==============================] - 0s 1ms/step
```



Модель хорошо справилась с предсказанием значения выходного сигнала

Вывод

В данной работе я еще 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