Московский авиационный институт
(Национальный исследовательский университет)
Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №2
по курсу «Нейроинформатика»

Линейная нейронная сеть. Правило обучения Уидроу-Хоффа

Выполнил: С.А. Красоткин
Группа: 8О-408Б
Вариант: 17
Преподаватели: Тюменцев Ю.В.
Рожлейс И. А.
Оценка:

Москва, 2022

Лабораторная №2

Вариант № 17

Цель работы

Исследование свойств линейной нейронной сети и алгоритмов её обучения, применение сети в задачах аппроксимации и фильтрации.

Основные этапы работы

1. Использовать линейную нейронную сеть с задержками для аппроксимации функции. В качестве метода обучения использовать адаптацию.
2. Использовать линейную нейронную сеть в качестве адаптивного фильтра для подавления помех. Для настройки весовых коэффициентов использовать метод наименьших квадратов.

Код

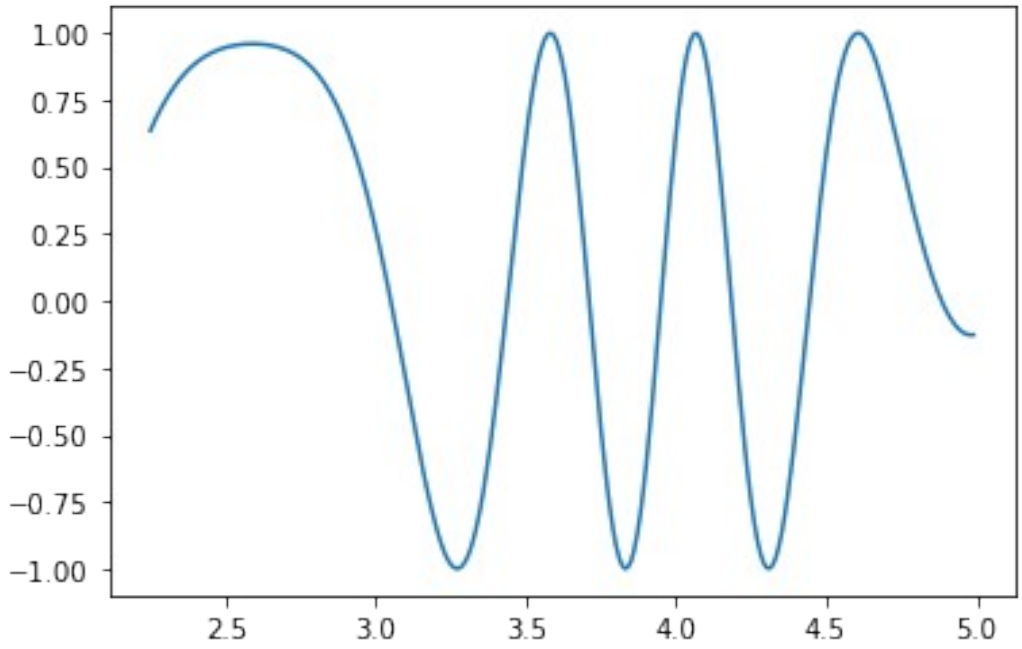
```
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt

def plot_history(h, *metrics):
    for metric in metrics: print(f"{metric}: {h.history[metric][-
1]:.4f}")
    figure = plt.figure(figsize=(5.25 * len(metrics), 3.75))
    for i, metric in enumerate(metrics, 1):
        ax = figure.add_subplot(1, len(metrics), i)
        ax.xaxis.get_major_locator().set_params(integer=True)
        plt.title(metric)
        plt.plot(h.history[metric], '-')
    plt.show()
```

Аппроксимация функции

```
window = 5
EPOCHS = 50

def x1(t): return np.sin(np.sin(t)*(t**2) + 3*t - 10)

t1 = np.arange(2.25, 5, 0.01)
```

Исходный сигнал

```
plt.plot(t1, x1(t1))
plt.show()
```



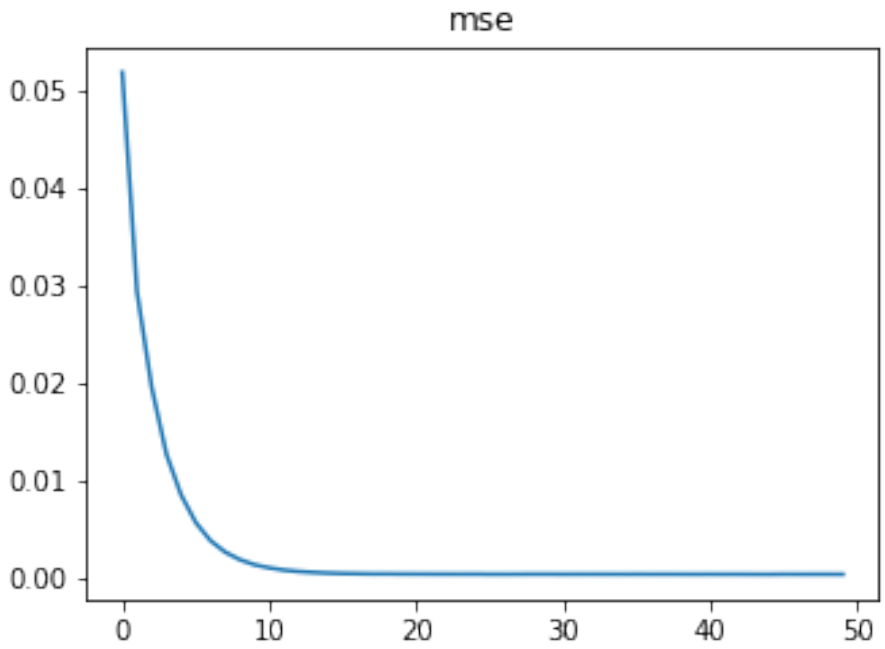
Подготовка данных

```
data1 = x1(t1)
target1 = data1[window:]
data1 = np.array([data1[i:i+window] for i in range(0, len(data1) -
window)])
```

Построение и обучение модели

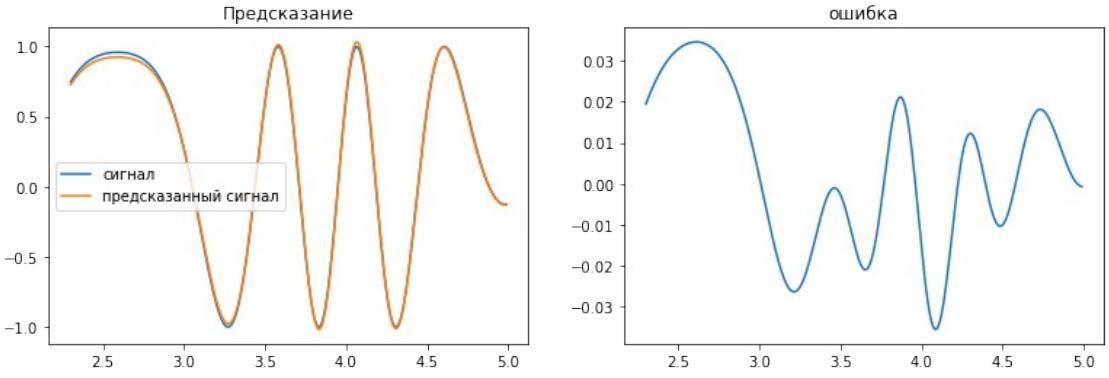
```
model1 = keras.models.Sequential([ keras.layers.Dense(1,
input_dim=window, activation='linear')])

model1.compile(keras.optimizers.SGD(0.01), 'mse', ['mse'])

hist1 = model1.fit(data1, target1, batch_size=1, epochs = EPOCHS,
verbose=0, shuffle=True)

plot_history(hist1, 'mse')
```

```
mse: 0.0003
```




```
pred1 = model1.predict(data1)
figure = plt.figure(figsize=(13, 4))
figure.add_subplot(1, 2, 1)
plt.title('Предсказание')
plt.plot(t1[window:], target1, label='сигнал')
plt.plot(t1[window:], pred1, label='предсказанный сигнал')
plt.legend()
figure.add_subplot(1, 2, 2)
plt.title('ошибка')
plt.plot(t1[window:], target1 - pred1.flat)
plt.show()
```

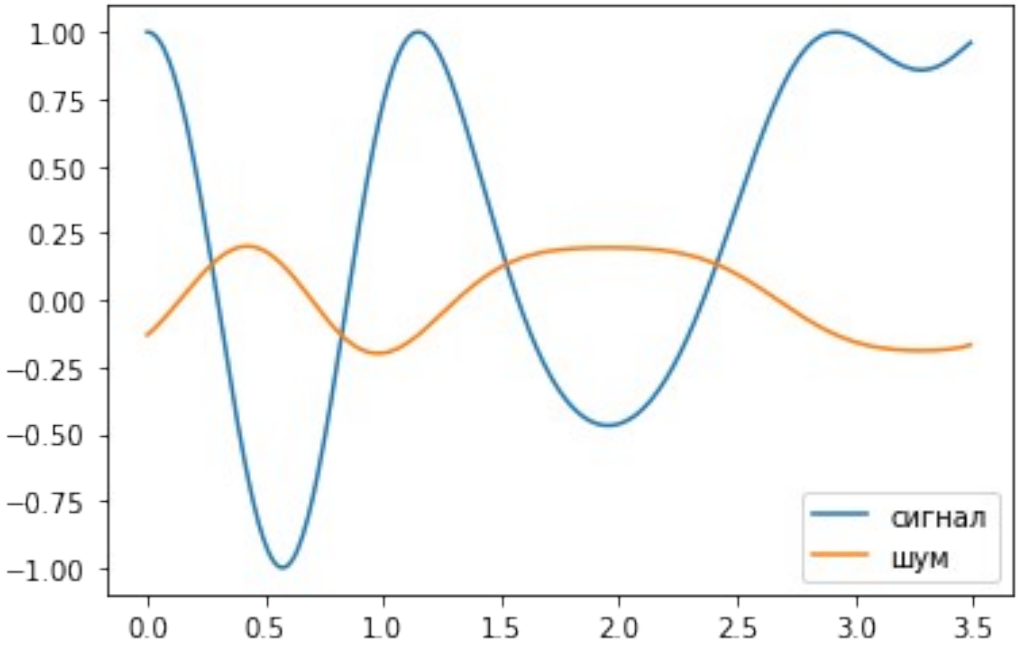
```
9/9 [==============================] - 0s 2ms/step
```



Фильтрализация

```
window = 4
EPOCHS = 600
```

```
def x2(t): return np.cos(np.cos(t)*(t**2) + 5*t)

def y(t): return 0.2 * np.cos(np.cos(t)*(t**2) + 5*t + 4)

t2 = np.arange(0, 3.5, 0.01)

plt.plot(t2, x2(t2), label='сигнал')
plt.plot(t2, y(t2), label='шум')
plt.legend()
plt.show()
```



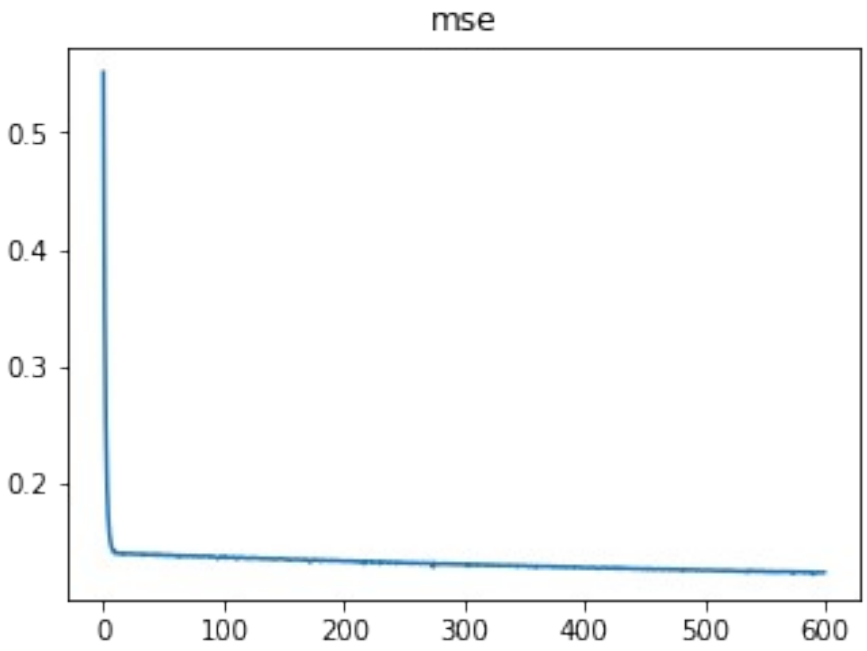
Подготовка обучающих данных

```
data2 = y(t2)
data2 = np.array([data2[i:i+window] for i in range(0, len(data2) -
window)])
target2 = x2(t2)[window:]
```

Построение и обучение модели

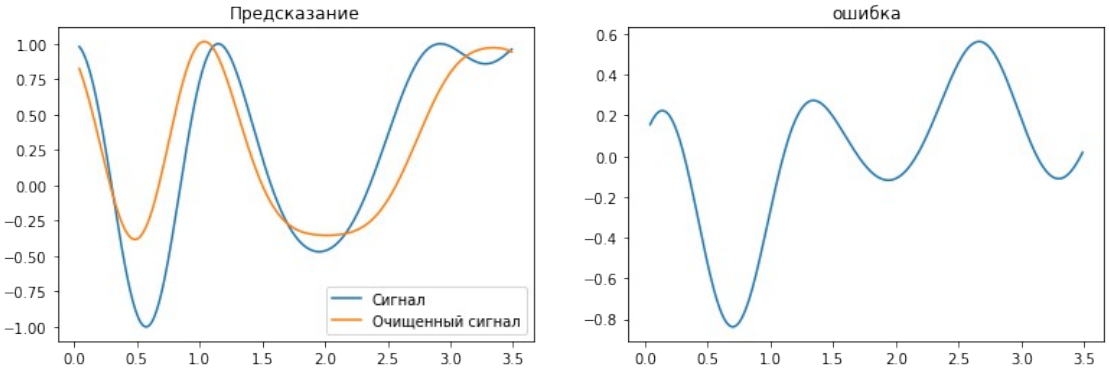
```
model2 = keras.models.Sequential([ keras.layers.Dense(1,
input_dim=window, activation='linear')])

model2.compile(keras.optimizers.SGD(0.0055), 'mse', metrics = ["mse"])

hist2 = model2.fit(data2, target2, batch_size=1, epochs = EPOCHS,
verbose=0, shuffle=True)

plot_history(hist2, 'mse')
```

```
mse: 0.1244
```




```
pred2 = model2.predict(data2)
figure = plt.figure(figsize=(13, 4))
figure.add_subplot(1, 2, 1)
plt.title('Предсказание')
plt.plot(t2[window:], target2, label='Сигнал')
plt.plot(t2[window:], pred2, label='Очищенный сигнал')
plt.legend()
figure.add_subplot(1, 2, 2)
plt.title('ошибка')
plt.plot(t2[window:], target2 - pred2.flat)
plt.show()
```

```
11/11 [==============================] - 0s 2ms/step
```



Выводы

В ходе выполнения лабораторной работы отработал апроксимацию и фильтрализацию с помощью однослойной нейросети.

Последнее вышло не точно, но чуть улучшилось после увеличения эпох и learning rate.