Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 3

по курсу «Нейроинформатика»

Студент: Аксенов А. Е.
Группа: М80-408Б-20
Преподаватель: Горохов М. А.
Оценка:

Москва, 2023

Цель работы

Исследование свойств многослойной нейронной сети прямого распространения и алгоритмов ее обучения, применение сети в задачах классификации и аппроксимации функции.

Основные этапы работы

1. Использовать многослойную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.
2. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.
3. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов второго порядка.

Оборудование

Процессор : Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
ОЗУ : 16 ГБ

Программное обеспечение

Python 3.8 + Jupyter Notebook

Сценарий выполнения работы

1. Использовать многослойную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми

1.1 Создадим выборку согласно варианту. Перейдем в декартову систему координат. Отрисуем множества. Классы линейно неразделимы.

1.	Эллипс: $a = 0.3, b = 0.3, \alpha = 0, x_0 = 0, y_0 = 0$ Эллипс: $a = 0.7, b = 0.7, \alpha = 0, x_0 = 0, y_0 = 0$ Эллипс: $a = 1, b = 1, \alpha = 0, x_0 = 0, y_0 = 0$

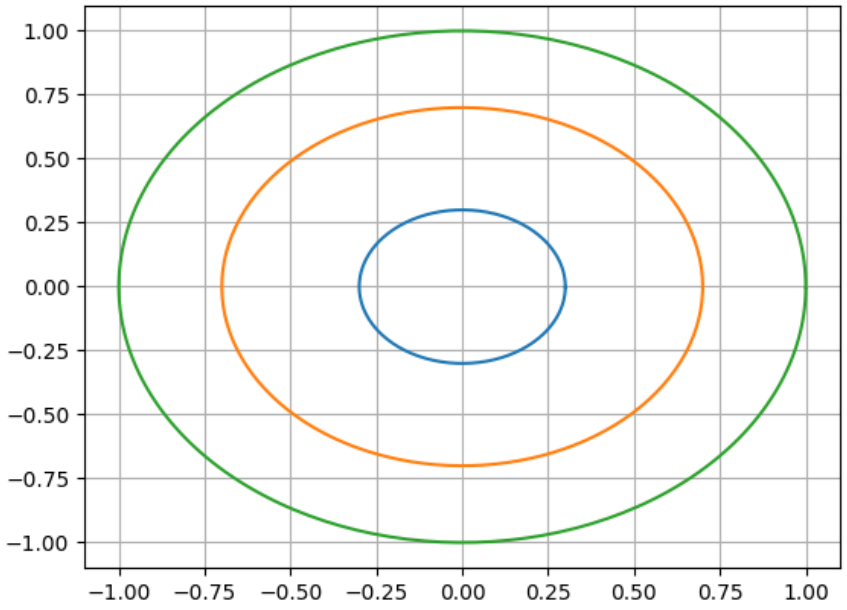

Рис. 1 График с выборкой

1.2 Для первого класса, второго класса и третьего класса возьмем соответственно 60, 100, 120 элементов выборки случайным образом. Отрисуем подвыборку

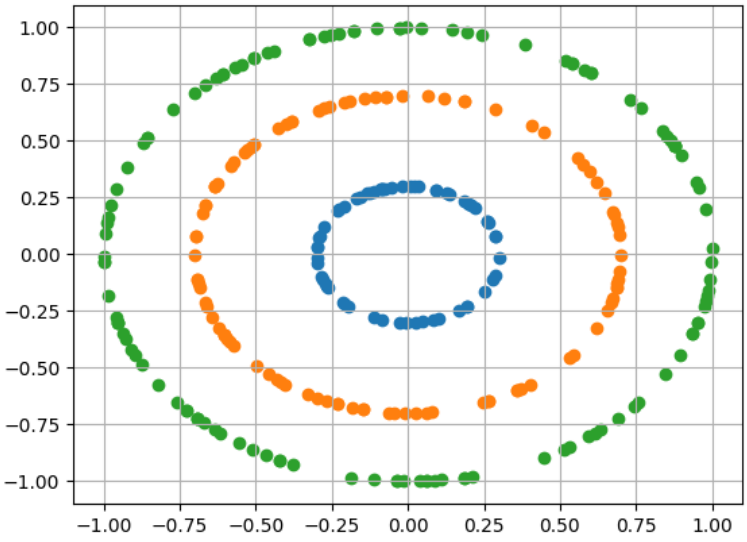

Рис. 2 График подвыборки

1.3 Разделим данные на выборки для обучения, контроля и теста в соотношении 0.7, 0.1, 0.2, чтоб отслеживать переобучение

1. 4 Создадим нейронную сеть с 3 слоями. Первых входной слой с 2 нейронами. Второй слой = 25 нейрона с функцией активации Relu. Третий слой = 3 нейрона с функцией активации softmax. Архитектура выглядит следующем образом

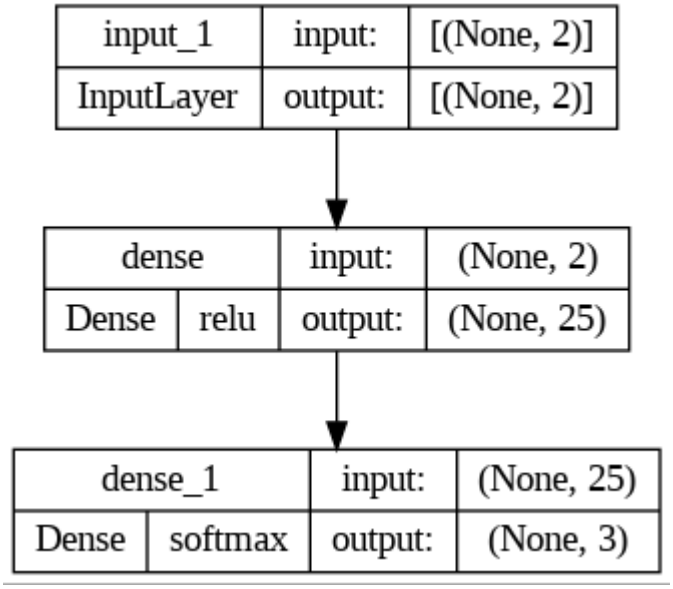

Рис. 3 График архитектуры сети

1.5 Обучим модель. Оптимизатор при обучении будет использоваться Adam. Классифицируем область.

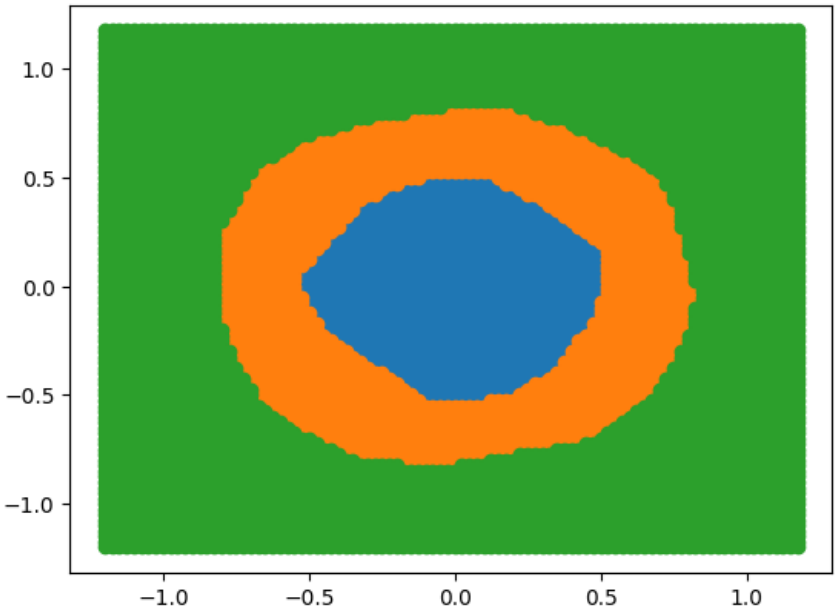

Рис. 4 Классифицированные области

Как видно из графика, модель достаточно хорошо смогла справиться с задачей классификации линейно неразделимых классов.

2. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.

2.1 Инициализируем выборку согласно варианту и отрисуем выборку

$$x = \sin(t^2), \quad t \in [0, 4],\ h = 0.02$$

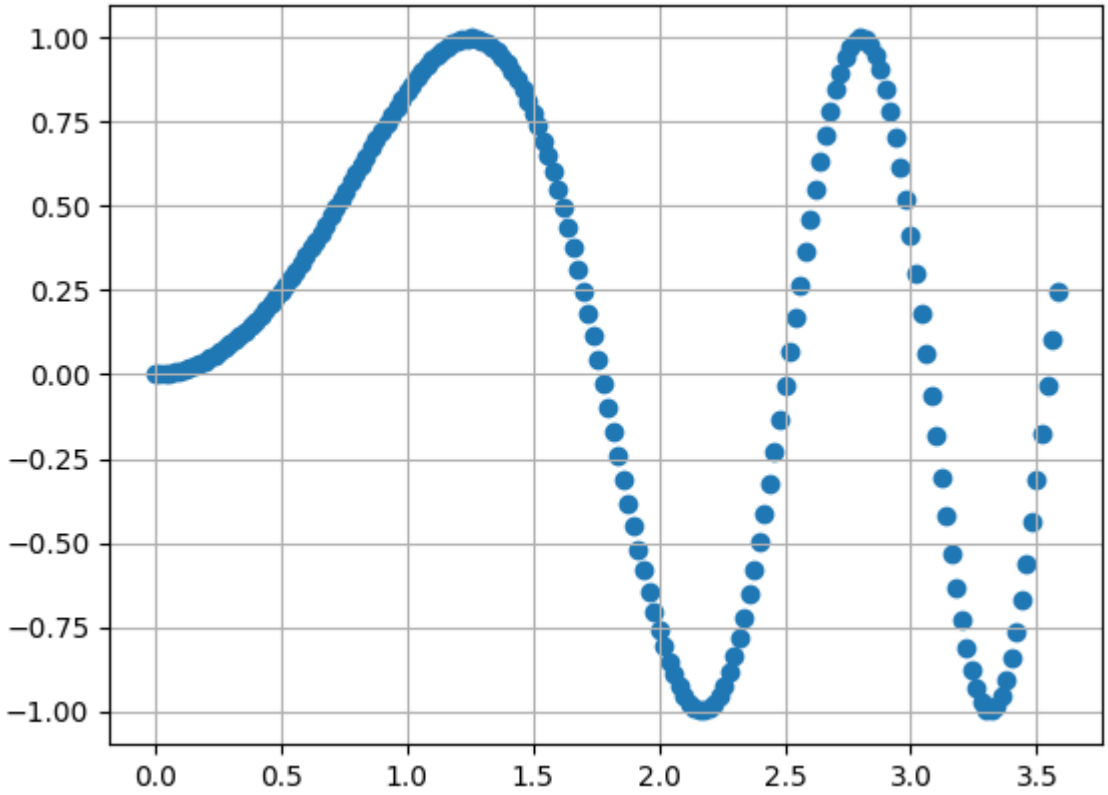

Рис. 5 График эталонной кривой

2.2 Создаем сеть с 4 слоями. Первый слоя является входным и имеет 1 нейрон. Второй слой имеет 10 нейронов и функцию активации Relu, третий слов имеет 30 нейронов и функцию активации Relu, последний слой имеет 1 нейронов и линейную функцию активации. Внешне сеть выглядит так:

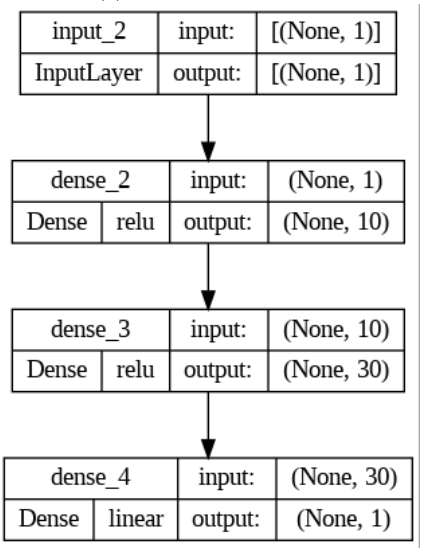

Рис. 6 Структура сети

2.4 Разделим выборку на обучающую и валидационную выборки в соотношении 0.9 и 0.1, при том 0.1 берется с конца выборки. Количество эпох = 1200. Обучим модель методом RMSprop. Построим график истинный график и график предсказаний для тестовой и обучающей выборке.

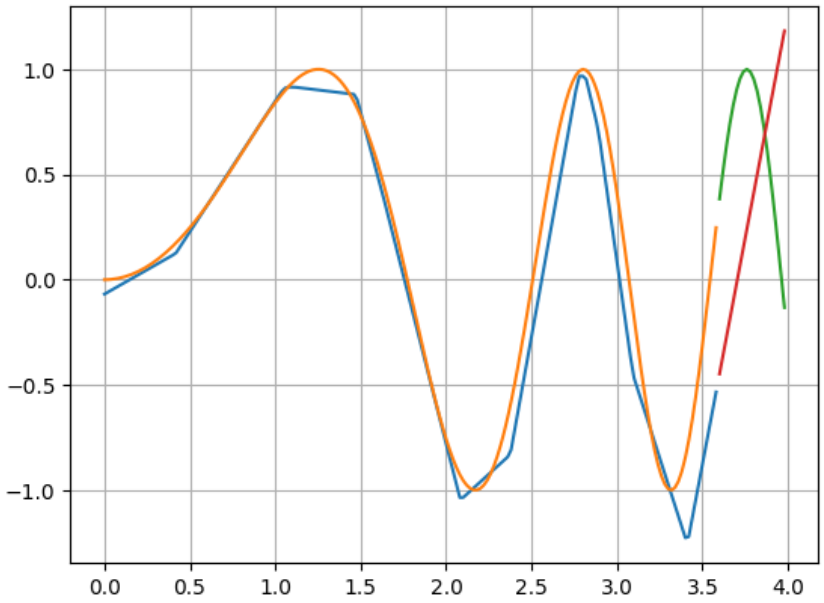

Рис. 7 График предсказанной и истинной кривых

Как видно из графика модель достаточно хорошо справилась с задачей. На тестовой выборке модель смогла предсказать общее направление, но не изгиб кривой.

3. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов второго порядка.

3.1 Задаем выборку и отрисуем эталонный график

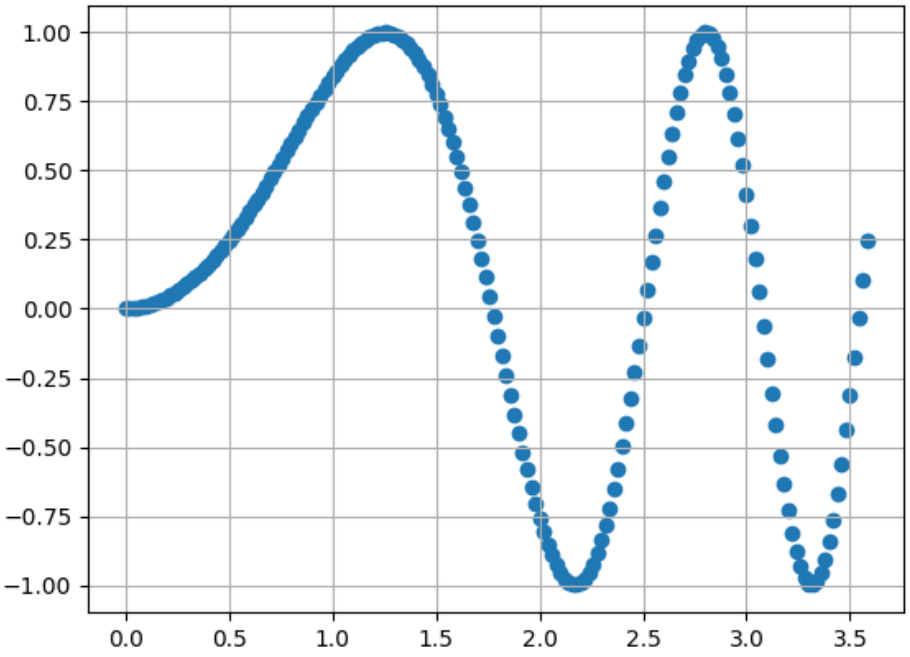

Рис. 8 График эталона

3.2 Создаем сеть с 4 слоями. Первый слоя является входным и имеет 1 нейрон. Второй слой имеет 10 нейронов и функцию активации Relu, третий слов имеет 30 нейронов и функцию активации Relu, последний слой имеет 1 нейронов и линейную функцию активации. Внешне сеть выглядит так:

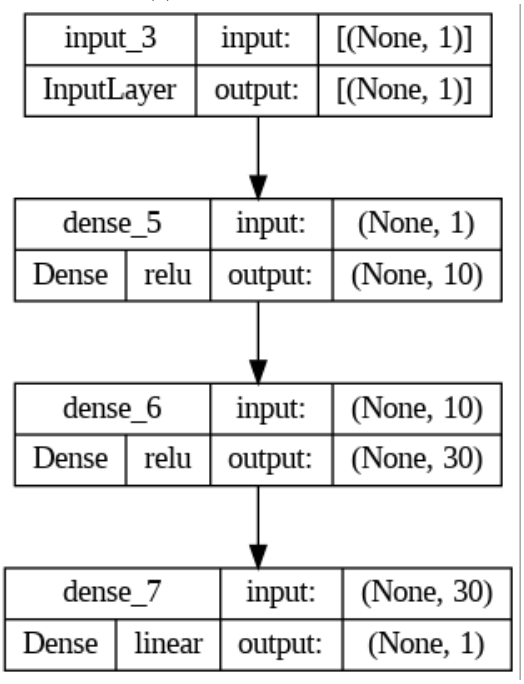

Рис. 9 Архитектура сети

3.4 Разделим выборку на обучающую и валидационную выборки в соотношении 0.9 и 0.1, при том 0.1 берется с конца выборки. Количество эпох = 1200. Обучим модель методом Nadam. Построим график истинный график и график предсказаний для тестовой и обучающей выборке.

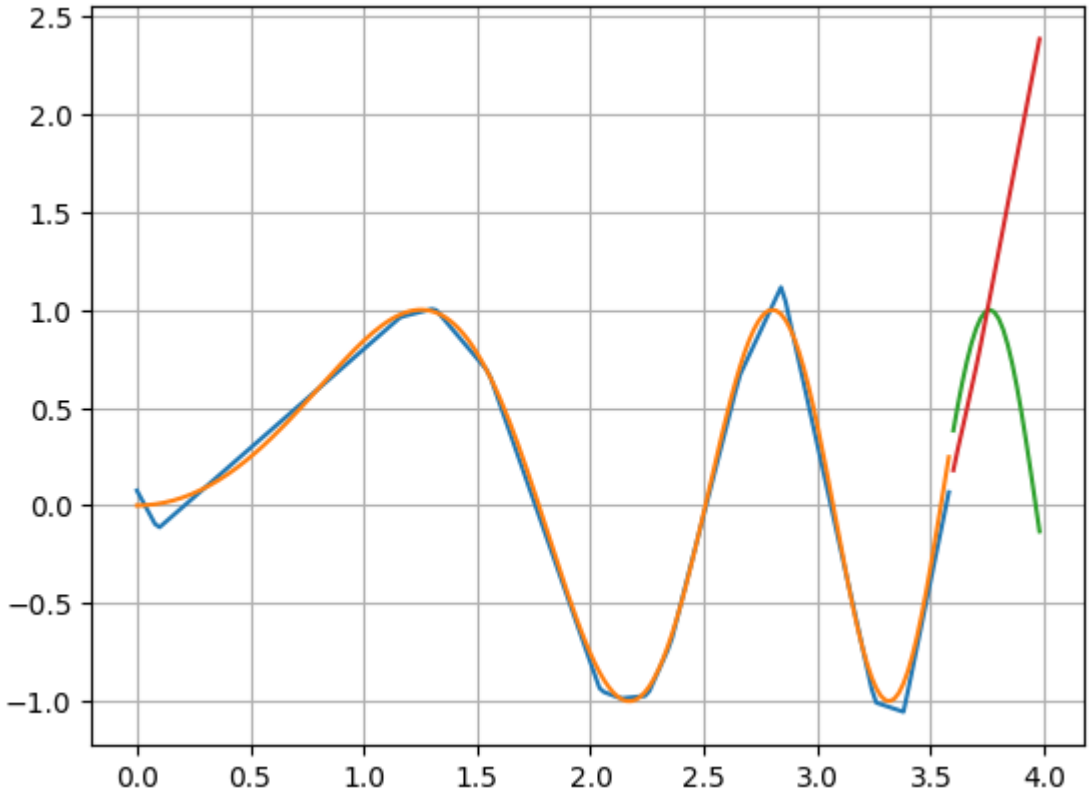

Рис. 8 График предсказанной и истинной кривых

Как видно из графика модель достаточно хорошо справилась с задачей. На тестовой выборке модель смогла предсказать общее направление, но не изгиб кривой.

Код программы

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split


# In[2]:


# Создание точек
def generate_points(a: float, b: float, x0: float, y0: float):
    t = np.arange(0, 2*np.pi, 0.025)
    points = np.zeros((t.shape[0], 2))
    points[:, 0] = a * np.cos(t) + x0
    points[:, 1] = b * np.sin(t) + y0
    return points


# In[3]:


# Создание выборки
points1 = generate_points(0.3, 0.3, 0, 0)
points2 = generate_points(0.7, 0.7, 0, 0)
points3 = generate_points(1, 1, 0, 0)
plt.plot(points1[:, 0], points1[:, 1])
plt.plot(points2[:, 0], points2[:, 1])
plt.plot(points3[:, 0], points3[:, 1])
plt.grid(True)
plt.show()


# In[4]:


# Разбиение выборки
points1 = points1[np.random.choice(points1.shape[0], 60),:]
points2 = points2[np.random.choice(points2.shape[0], 100),:]
points3 = points3[np.random.choice(points3.shape[0], 120),:]

plt.scatter(points1[:, 0], points1[:, 1])
plt.scatter(points2[:, 0], points2[:, 1])
plt.scatter(points3[:, 0], points3[:, 1])
plt.grid(True)
plt.show()


# In[5]:


# Подготовка выборки
features = np.concatenate((points1, points2, points3))
labels = np.concatenate((np.full(60, 0), np.full(100, 1), np.full(120, 2)))
```

```
# In[6]:


x_train, x_test, y_train, y_test = train_test_split(features, labels,
test_size=0.3, random_state=23)
x_valid, x_test, y_valid, y_test = train_test_split(x_test, y_test,
test_size=0.33, random_state=26)
print(x_train.shape[0] / 280, x_valid.shape[0] / 280, x_test.shape[0] / 280)


# In[7]:


# Создание модели
model = keras.Sequential([
    keras.Input(shape=(2,)),
    layers.Dense(25, activation='relu'),
    layers.Dense(3, activation='softmax')
])
model.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=["accuracy"]
)


# In[8]:


keras.utils.plot_model(model,show_shapes=True, show_layer_activations=True)


# In[9]:


model.fit(x_train, y_train, batch_size=32, epochs=1000)


# In[10]:


a = np.mgrid[-1.2:1.2:0.025, -1.2:1.2:0.025].reshape(2, -1).T


# In[11]:


labels = model.predict(a)
labels = labels.argmax(1)


# In[12]:


set(labels)


# In[13]:


# Разметка пространства сетью
plt.scatter(a[labels==0,0],a[labels==0,1])
plt.scatter(a[labels==1,0],a[labels==1,1])
```

```
plt.scatter(a[labels==2,0],a[labels==2,1])


# ##### Модель достаточно хорошо решила задачу. А также верно
классифицировала области.

# In[14]:


# Создание выборки
xt = lambda t: np.sin(t*t)
features = np.arange(0, 4, 0.02)
targets = xt(features)
n = int(0.9 * features.shape[0])
x_train, y_train = features[:n], targets[:n]
x_test, y_test = features[n:], targets[n:]
plt.scatter(x_train, y_train)
plt.grid(True)
plt.show()


# In[15]:


# Создание модели
approx_model = keras.Sequential([
    keras.Input(shape=(1,)),
    layers.Dense(10, activation='relu', use_bias=True),
    layers.Dense(30, activation='relu', use_bias=True),
    layers.Dense(1)
])
approx_model.compile(
    loss='mse',
    optimizer=keras.optimizers.RMSprop(learning_rate=0.01),
    metrics=['mse', 'mae', keras.metrics.RootMeanSquaredError()]
)


# In[16]:


keras.utils.plot_model(approx_model,show_shapes=True,
show_layer_activations=True)


# In[17]:


# Обучаем
approx_model.fit(x_train, y_train, epochs=1200, batch_size=20)


# In[18]:


train_predictions = approx_model.predict(x_train)
test_predictions = approx_model.predict(x_test)


# In[19]:


# Отображаем обучающую и тестовую выборки, предсказания на них
plt.plot(x_train, train_predictions)
```

```
plt.plot(x_train, y_train)
plt.plot(x_test, y_test)
plt.plot(x_test, test_predictions)

plt.grid(True)
plt.show()


# In[20]:


del approx_model
# Создаем модель
approx_model = keras.Sequential([
    keras.Input(shape=(1,)),
    layers.Dense(10, activation='relu', use_bias=True),
    layers.Dense(30, activation='relu', use_bias=True),
    layers.Dense(1)
])
approx_model.compile(
    loss='mse',
    optimizer=keras.optimizers.Nadam(learning_rate=0.01),
    metrics=['mse', 'mae', keras.metrics.RootMeanSquaredError()]
)


# In[21]:


keras.utils.plot_model(approx_model,show_shapes=True,
show_layer_activations=True)


# In[22]:


# Обучаем модел
approx_model.fit(x_train, y_train, batch_size=20, epochs=1200)


# In[23]:


train_predictions = approx_model.predict(x_train)
test_predictions = approx_model.predict(x_test)


# In[24]:


# Отображаем обучающую и тестовую выборки, предсказания на них
plt.plot(x_train, train_predictions)
plt.plot(x_train, y_train)
plt.plot(x_test, y_test)
plt.plot(x_test, test_predictions)

plt.grid(True)
plt.show()


# ##### Nadam показала себя лучше чем RMSprop.
# #### На тестовой выборке оба метода показала верно общее начальное
направление. Но не смогли верно пресказать наличие изгиба
```

Выводы

В лабораторной работе было проведено исследование свойств многослойной нейронной сети прямого распространения и алгоритмов её обучения. Было продемонстрировано применение сети в задачах классификации и аппроксимации функции.