Лабораторная работа № 3

Многослойные сети. Алгоритм обратного распространения ошибки

Цель работы: исследование свойств многослойной нейронной сети прямого распространения и алгоритмов ее обучения, применение сети в задачах классификации и аппроксимации функции

Студент	Попов И.П.
Группа	М8О-406Б-20
Вариант	17

```
import numpy as np

import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

Применение многослойной нейронной сети в задачах классификации

Функция, которая генерирует координаты точек эллипса по параметрическим уравнениям

```
def ellipse(t, a, b, x0, y0):
    x = x0 + a * np.cos(t)
    y = y0 + b * np.sin(t)
    return x, y
```

Функция, которая генерирует координаты точек параболы по параметрическим уравнениям

```
def parabola(t, p, x0, y0):
    x = x0 + p * t**2
    y = y0 + t
    return x, y
```

Функция, выполняющая поворот точек вокруг начала координат

```
def rotate(x, y, alpha):
    xr = x * np.cos(alpha) - y * np.sin(alpha)
    yr = x * np.sin(alpha) + y * np.cos(alpha)
    return xr, yr
```

Генерирация равномерно распределенных значений от 0 до 2π для эллипсов и от -5 до 5 для параболы с общим количеством точек равным 200

```
t_for_ellipse = np.linspace(0, 2 * np.pi, 200)
t_for_parabola = np.linspace(-5, 5, 200)
```

Классы, выраженные эллипсами и параболой

```
# Класс 1
x1, y1 = ellipse(t_for_ellipse, a=0.4, b=0.15, x0=0.1, y0=-0.15)
x1, y1 = rotate(x1, y1, np.pi / 6.)

# Класс 2
x2, y2 = ellipse(t_for_ellipse, a=0.7, b=0.5, x0=0., y0=0.)
x2, y2 = rotate(x2, y2, -np.pi / 3.)

# Класс 3
x3, y3 = parabola(t_for_parabola, p=1, x0=-0.8, y0=0.)
x3, y3 = rotate(x3, y3, np.pi / 2.)
```

Создание списков points для каждого класса. Далее, создаются списки классов classes1, classes2, classes3 в формате one-hot encoding для каждого класса. Наконец, переменные X и Y объединяют все точки и соответствующие им классы для последующего использования в обучении многослойной нейронной сети.

```
points1 = [[x, y] for x, y in zip(x1, y1)]
points2 = [[x, y] for x, y in zip(x2, y2)]
points3 = [[x, y] for x, y in zip(x3, y3)]

classes1 = [[1., 0., 0.] for _ in range(len(points1))]
classes2 = [[0., 1., 0.] for _ in range(len(points2))]
classes3 = [[0., 0., 1.] for _ in range(len(points3))]

X = points1 + points2 + points3
Y = classes1 + classes2 + classes3
```

Разделение данных на обучающий (x_train, y_train) и тестовый (x_test, y_test) наборы. 20% данных будут использоваться для тестирования, а 80% - для обучения. Параметр random_state=42 фиксирует случайное разбиение, обеспечивая воспроизводимость результатов.

```
x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
```

Преобразование в тензоры

```
x_train = torch.FloatTensor(np.array(x_train))
y_train = torch.FloatTensor(np.array(y_train))

x_test = torch.FloatTensor(np.array(x_test))
y_test = torch.FloatTensor(np.array(y_test))
```

Через train_loader будем загружать на обучение батчи размером 10% от обучающего датасета.

```
train_dataset = TensorDataset(x_train, y_train)
test_dataset = TensorDataset(x_test, y_test)

batch_size = int(len(train_dataset) * 0.1)
train_loader = DataLoader(train_dataset, batch_size)
```

Создание двухслойной сети. Сеть состоит из двух полносвязных слоев, применяющих нелинейные функции активации tanh и sigmoid. tanh в скрытом слое помогает избежать проблемы затухающих градиентов, а sigmoid в выходном слое сжимает значения в нужный диапазон для предсказания цветов.

```
class TwoLayerNetwork(nn.Module):
    def __init__(self, in_features: int, hidden_layer: int,
out_features: int):
        super().__init__()
        self.fc1 = nn.Linear(in_features, hidden_layer)
        self.fc2 = nn.Linear(hidden_layer, out_features)

    def forward(self, x):
        x = self.fc1(x)
        x = torch.tanh(x)
        x = self.fc2(x)
        x = torch.sigmoid(x)
        return x
```

Сеть настроена на прогнозирование трех значений в диапазоне [0, 1], что может быть интерпретировано как цветовая компонента RGB.

```
twoLayerNetwork = TwoLayerNetwork(2, 100, 3)
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(twoLayerNetwork.parameters())
```

Внутри цикла по эпохам, для каждого батча, модель обновляется, вычисляется значение функции потерь и производится оптимизация параметров.

```
def fit(model, train_loader, criterion, optimizer, epochs):
    losses = []
```

```
    running_loss = 0.0
    processed_data = 0

    log_template = "\nEpoch {ep:03d} train_loss: {t_loss:0.4f}"
    for epoch in range(epochs):
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outp = model(inputs)

            loss = criterion(outp, labels)

            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            processed_data += inputs.size(0)

        train_loss = running_loss / processed_data
        losses.append(train_loss)
        print(log_template.format(ep=epoch+1, t_loss=loss))
    return losses
```

Выполняет предсказание с использованием модели, обученной на тестовых данных.

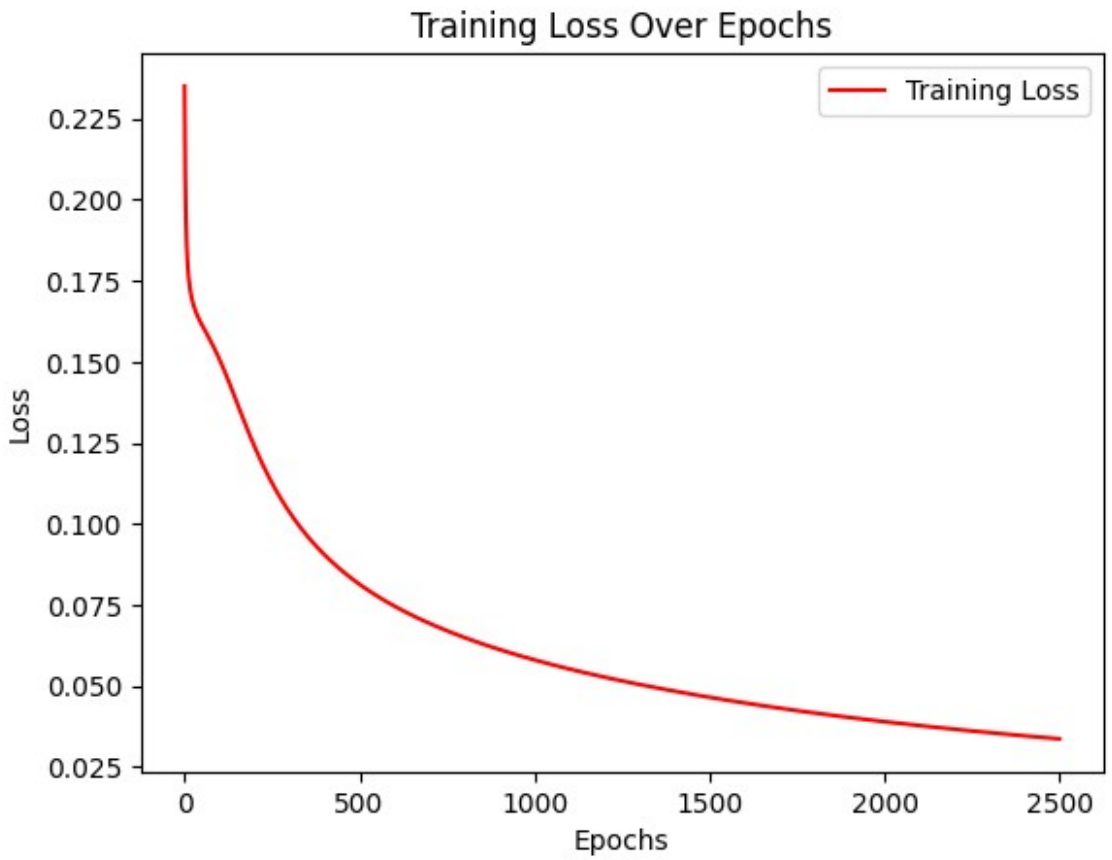
```
def predict(model, x_test):
    with torch.no_grad():
        model.eval()
        outp = model(x_test)
    return outp
```

Процесс обучения, который включает 2500 эпох.

```
losses = fit(twoLayerNetwork, train_loader, loss_function, optimizer,
2500)
```

График функции потерь (ошибка - MSE)

```
plt.plot(losses, color='red', label='Training Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Over Epochs')
plt.show()
```



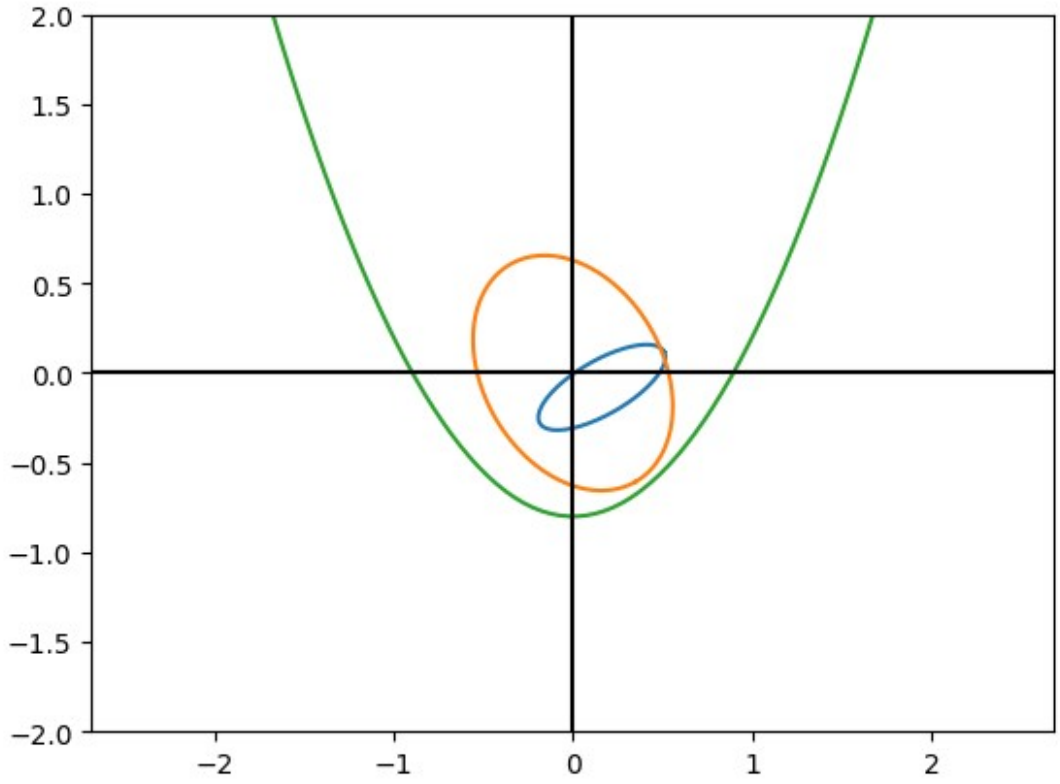
Формируется сетка точек для области [-1, 1] x [-1, 1], и каждая точка представляется парой координат x и y. Затем эти точки преобразуются в тензор модель используется для предсказания цветовых компонент RGB для каждой точки. Результаты предсказаний сохраняются в переменной predicted_RGB, которая изменяется в форму (200, 200, 3) для удобства последующего использования.

```
x_test = [[x, y] for x in np.linspace(-1, 1, 200) for y in
np.linspace(-1, 1, 200)]

x_test = torch.FloatTensor(np.array(x_test))

predicted_RGB = predict(twoLayerNetwork, x_test)
predicted_RGB = predicted_RGB.reshape((200, 200, 3))
```

Визуализация приведенных выше классов.

```
plt.plot(x1, y1)
plt.plot(x2, y2)
plt.plot(x3, y3)

plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
```

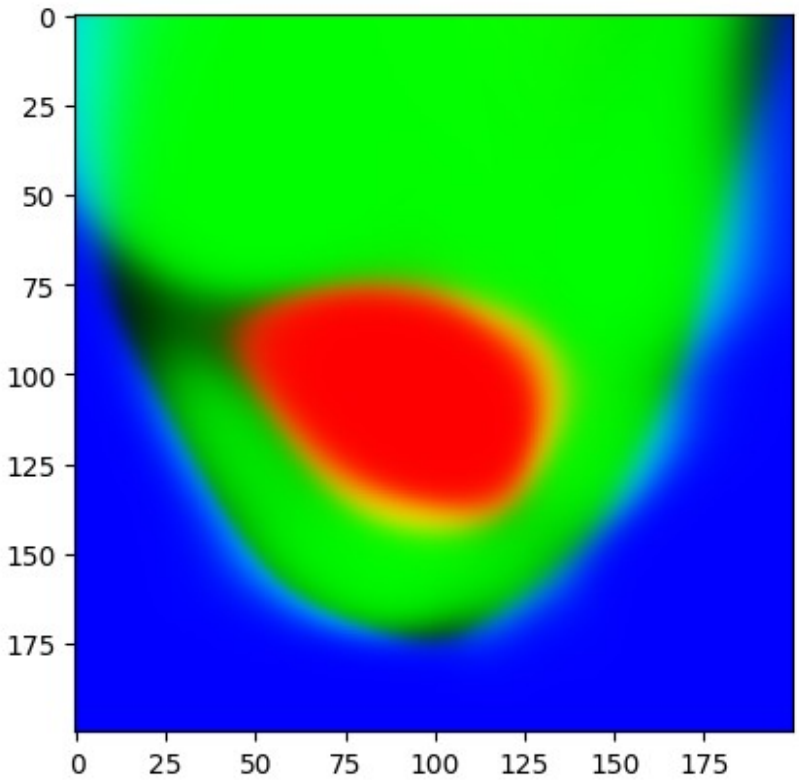
```
plt.axis('equal')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.show()
```



Разделение области на классы, которые линейно неразделимы

```
rotated_image = np.rot90(predicted_RGB)
flipped_image = np.fliplr(rotated_image)
plt.imshow(flipped_image)
```

```
<matplotlib.image.AxesImage at 0x1c2db1562f0>
```



Применение многослойной нейронной сети в задачах аппроксимации функций

Функция для аппроксимации

```
def function(t):
    return np.cos(-5 * t**2 + 10 * t - 5)
```

Создадим два набора точек - один разряженный, другой - эталонный

```
t1 = np.linspace(0, 2.5, 150)
f1 = function(t1)

t2 = np.linspace(0, 2.5, 2000)
f2 = function(t2)
```

Преобразуем данные для обучения и тестированияв тензоры. Изменяем форму данных, чтобы соответствовать ожидаемому формату для использования в многослойной нейронной сети. Создается датасет для обучения, который будет использоваться при обучении сети.

```
x_train2 = torch.FloatTensor(t1)
y_train2 = torch.FloatTensor(f1)
```

```
x_train2 = x_train2.view(-1, 1)
y_train2 = y_train2.view(-1, 1)
train_dataset2 = TensorDataset(x_train2, y_train2)

x_test2 = torch.FloatTensor(t2)
x_test2 = x_test2.view(-1, 1)
```

В данном классе трехслойной нейросети использование tanh в скрытых слоях предоставляет сети возможность работать со сложными, нелинейными зависимостями, что важно для успешной аппроксимации разряженной функции.

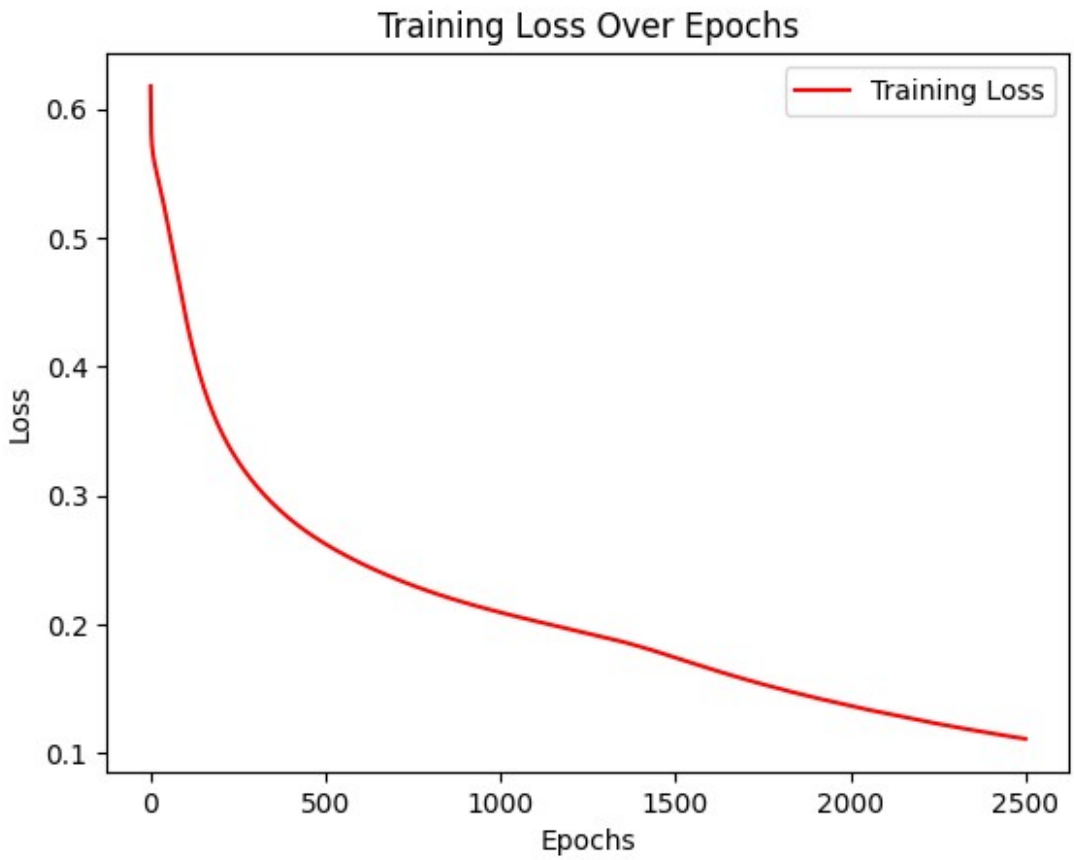
```
class TripleLayerNetwork (nn.Module):
    def __init__(self, in_features: int, hidden_layer: list,
out_features: int):
        super().__init__()
        assert len(hidden_layer) == 2
        self.fc1 = nn.Linear(in_features, hidden_layer[0])
        self.fc2 = nn.Linear(hidden_layer[0], hidden_layer[1])
        self.fc3 = nn.Linear(hidden_layer[1], out_features)

    def forward(self, x):
        x = self.fc1(x)
        x = torch.tanh(x)
        x = self.fc2(x)
        x = torch.tanh(x)
        x = self.fc3(x)
        return x
```

Экземпляр модели будет с одним входным признаком, двумя скрытыми слоями (40 и 12 нейронов соответственно) и одним выходным признаком.

```
three_layer_net = TripleLayerNetwork (1, [40, 12], 1)
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(three_layer_net.parameters())

batch_size = 10
train_loader2 = DataLoader(train_dataset2, batch_size)

losses2 = fit(three_layer_net, train_loader2, loss_function,
optimizer, 2500)
```

График функции потерь (ошибка - MSE)

```
plt.plot(losses2, color='red', label='Training Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Over Epochs')
plt.show()
```

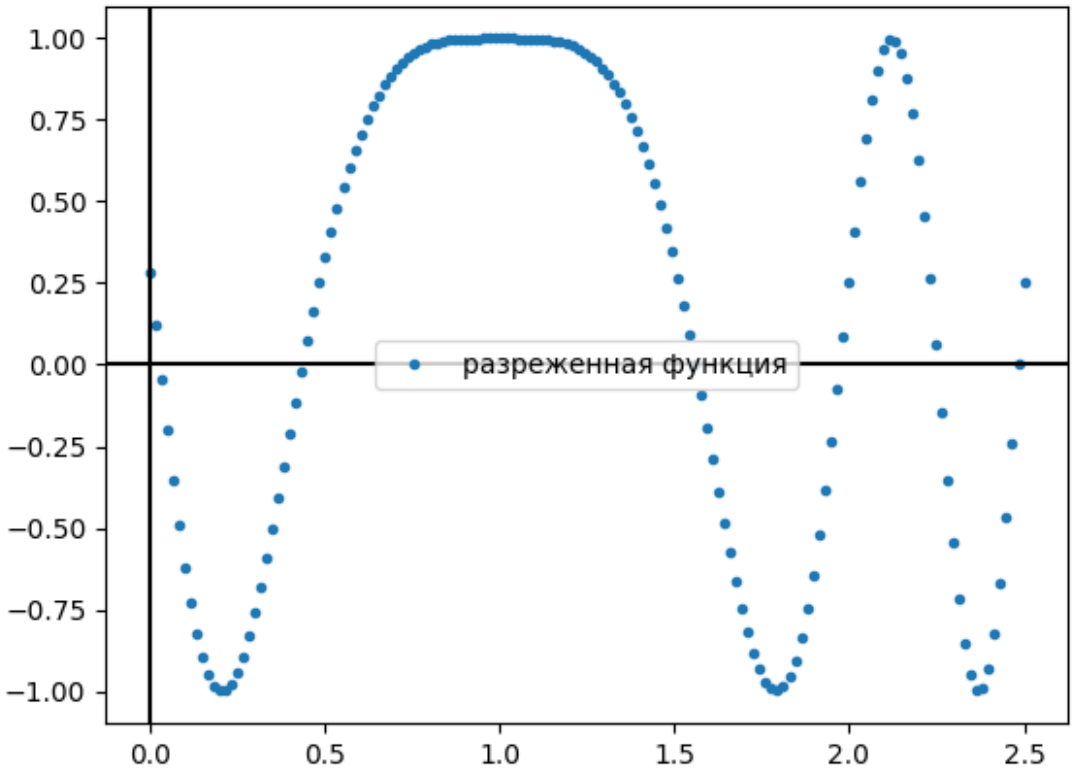


Получение предсказанных значений для тестового набора данных

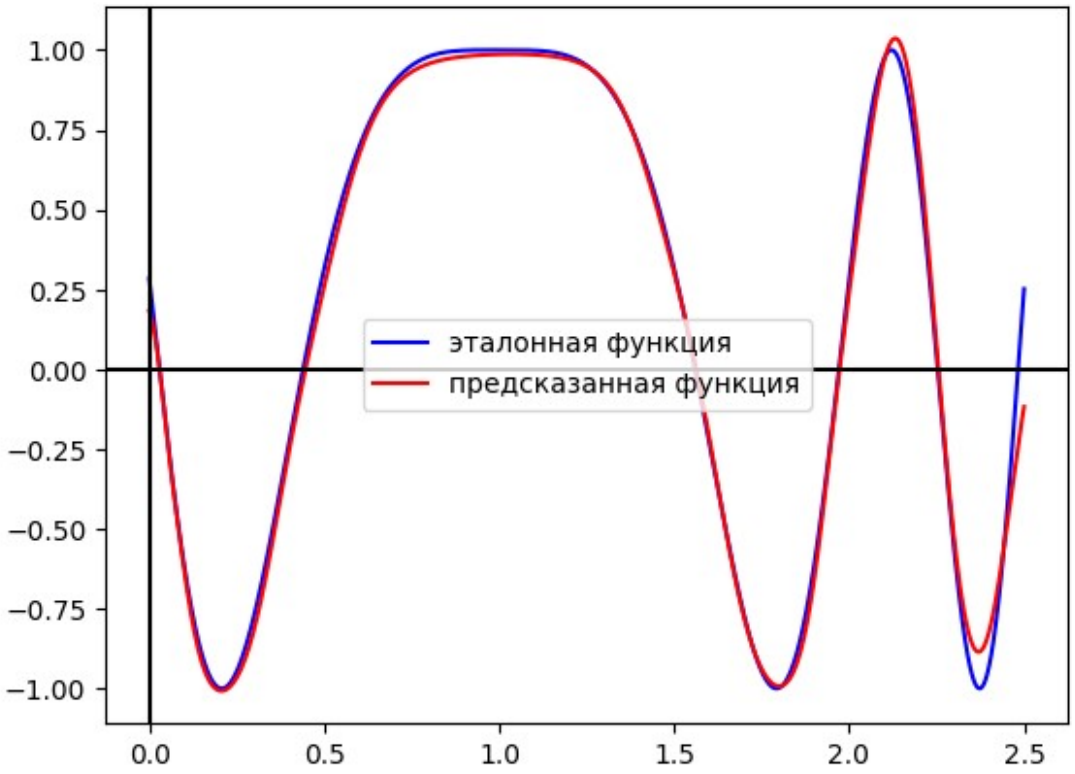
```
f2_pred = predict(three_layer_net, x_test2)
```

Сравнение разреженного варианта, приближения и истинной функции

```
plt.plot(t1, f1, '.', label='разреженная функция')

plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')

plt.legend()
plt.show()
```




```
plt.plot(t2, f2, color="blue", label='эталонная функция')
plt.plot(t2, f2_pred, color="red", label='предсказанная функция')

plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')

plt.legend()
plt.show()
```



Выводы: В ходе выполнения лабораторной работы были разработаны и обучены две многослойные нейронные сети с использованием алгоритма обратного распространения ошибки. Первая сеть применялась в задаче классификации для разделения линейно неразделимых данных, вторая же использовалась для аппроксимации функции.

Обе сети продемонстрировали хорошие результаты после обучения, хотя требовалось до 2500 эпох для достижения стабильных результатов. Важно отметить, что многослойные нейронные сети обладают способностью изучать сложные зависимости в данных, что делает их мощным инструментом для разнообразных задач машинного обучения.