Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчет по лабораторной работе №2

«Нейронные сети»

по дисциплине «Машинное обучение»

Выполнил
студент гр. 3530904/90102 Афанасьев Е.Д.

Руководитель
старший преподаватель ВШПИ Селин И. А.

Санкт-Петербург 2022

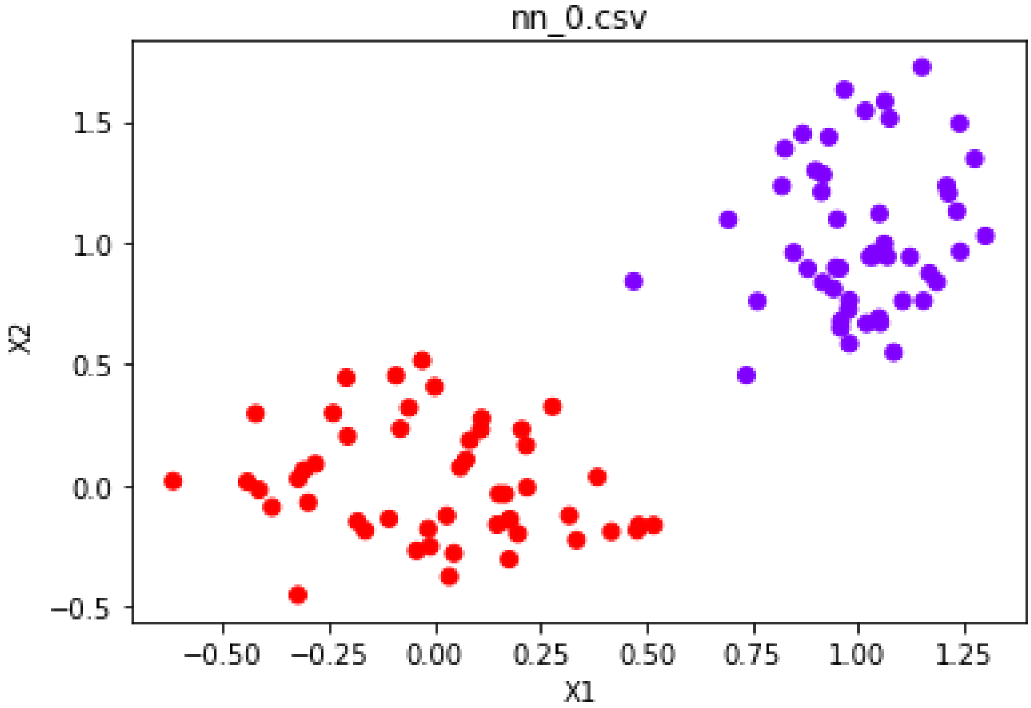
Оглавление

Задачи

1. Постройте нейронную сеть из одного нейрона и обучите её на датасетах nn_0.csv и nn_1.csv. Насколько отличается результат обучения и почему? Сколько потребовалось эпох для обучения? Попробуйте различные функции активации и оптимизаторы.
2. Модифицируйте нейронную сеть из пункта 1, чтобы достичь минимальной ошибки на датасете nn_1.csv. Почему были выбраны именно такие гиперпараметры?
3. Создайте классификатор на базе нейронной сети для набора данных MNIST (так же можно загрузить с помощью torchvision.datasets.MNIST, tensorflow.keras.datasets.mnist.load_data и пр.). Оцените качество классификации.

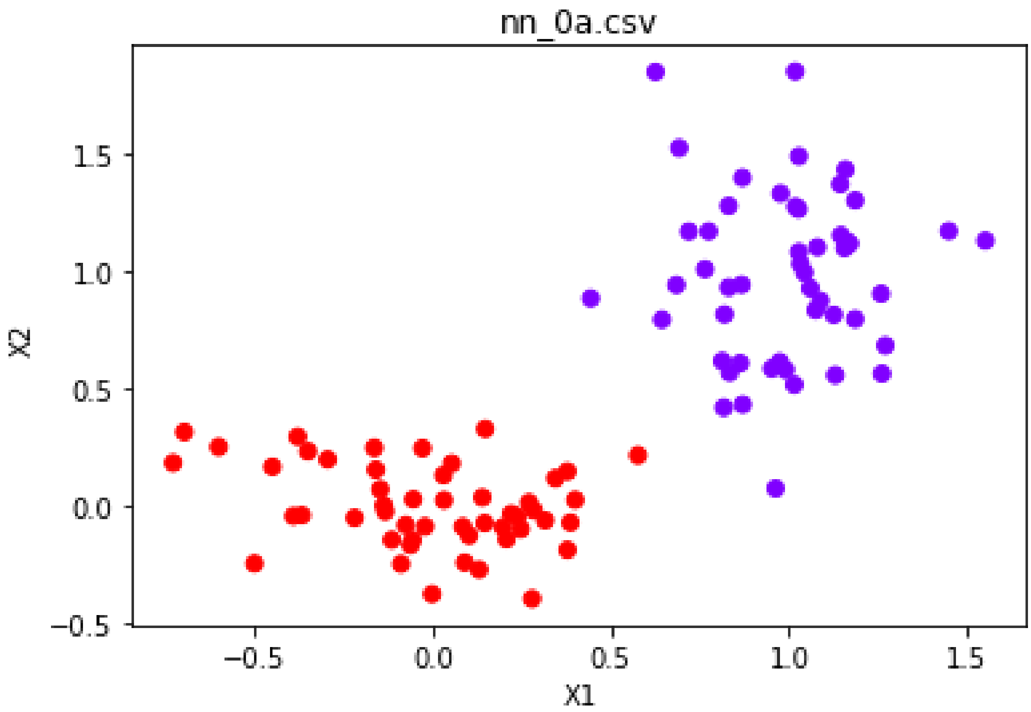
Один нейрон

Для обучения была использована модель с одним нейроном с различными функциями активации relu, logistic, tanh и оптимизаторами sgd, adam, lbfgs на всех датасетах.



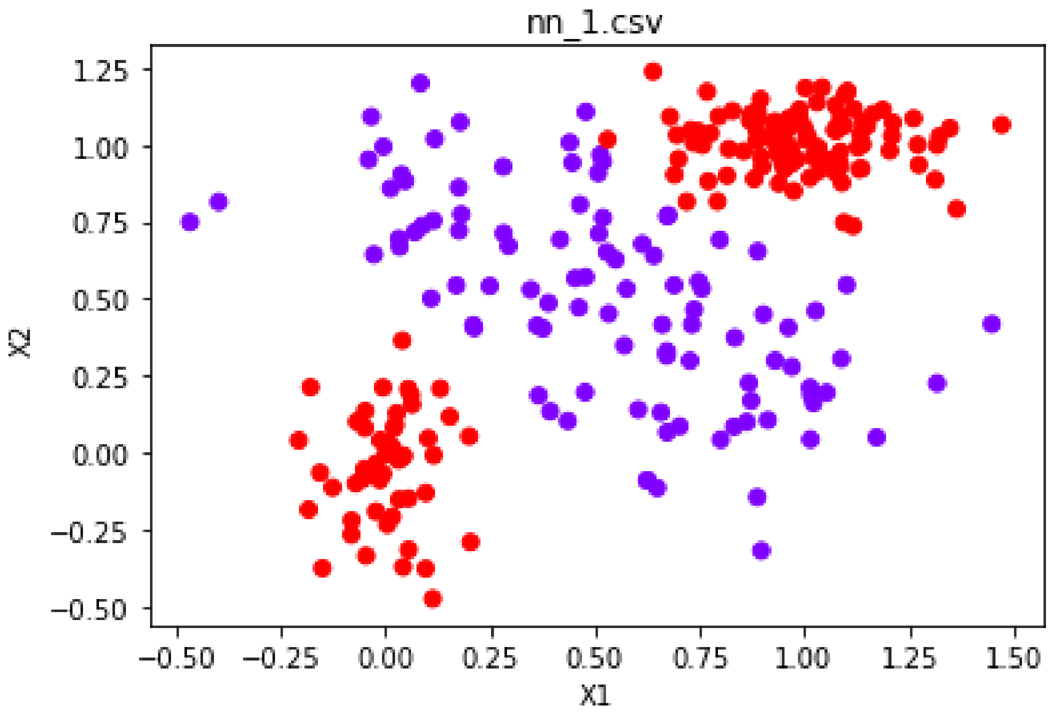

```
Activation function logistic
Optimizer lbfgs
Number of iterations 16
Number of layers 3
Accuracy 1.0
Confusion matrix
 [[ 8  0]
 [ 0 12]]
Activation function logistic
Optimizer sgd
Number of iterations 342
Number of layers 3
Accuracy 0.4
Confusion matrix
 [[ 0 12]
 [ 0  8]]
Activation function logistic
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 0.75
Confusion matrix
 [[8 0]
 [5 7]]
```

```
Activation function tanh
Optimizer lbfgs
Number of iterations 15
Number of layers 3
Accuracy 1.0
Confusion matrix
 [[15  0]
 [ 0  5]]
tanh
Optimizer sgd
Number of iterations 1000
Number of layers 3
Accuracy 1.0
Confusion matrix
 [[10  0]
 [ 0 10]]
Activation function tanh
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 0.55
Confusion matrix
 [[ 0  9]
 [ 0 11]]
Activation function relu
Optimizer lbfgs
Number of iterations 3
Number of layers 3
Accuracy 0.5
Confusion matrix
 [[ 0 10]
 [ 0 10]]
Activation function relu
Optimizer sgd
Number of iterations 962
Number of layers 3
Accuracy 1.0
Confusion matrix
 [[ 8  0]
 [ 0 12]]
Activation function relu
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 0.8
Confusion matrix
 [[8 0]
 [4 8]]
```




```
Activation function logistic
Optimizer lbfgs
Number of iterations 13
Number of layers 3
Accuracy 0.95
Confusion matrix
 [[12  0]
 [ 1  7]]
Activation function logistic
Optimizer sgd
Number of iterations 424
Number of layers 3
Accuracy 0.45
Confusion matrix
 [[ 0 11]
 [ 0  9]]
Activation function logistic
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 1.0
Confusion matrix
 [[13  0]
 [ 0  7]]
Activation function tanh
Optimizer lbfgs
Number of iterations 13
Number of layers 3
Accuracy 1.0
Confusion matrix
```

```
 [[ 7  0]
 [ 0 13]]
Activation function tanh
Optimizer sgd
Number of iterations 1000
Number of layers 3
Accuracy 1.0
Confusion matrix
 [[ 8  0]
 [ 0 12]]
Activation function tanh
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 0.95
Confusion matrix
 [[11  0]
 [ 1  8]]
Activation function relu
Optimizer lbfgs
Number of iterations 12
Number of layers 3
Accuracy 0.95
Confusion matrix
 [[10  1]
 [ 0  9]]
Activation function relu
Optimizer sgd
Number of iterations 1000
Number of layers 3
Accuracy 0.95
Confusion matrix
 [[15  0]
 [ 1  4]]
Activation function relu
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 0.5
Confusion matrix
 [[10  0]
 [10  0]]
```




```
Activation function logistic
Optimizer lbfgs
Number of iterations 20
Number of layers 3
Accuracy 0.66
Confusion matrix
 [[ 5 17]
 [ 0 28]]
Activation function logistic
Optimizer sgd
Number of iterations 12
Number of layers 3
Accuracy 0.58
Confusion matrix
 [[ 0 21]
 [ 0 29]]
Activation function logistic
Optimizer adam
Number of iterations 770
Number of layers 3
Accuracy 0.36
Confusion matrix
 [[ 0 23]
 [ 9 18]]
Activation function tanh
Optimizer lbfgs
Number of iterations 30
Number of layers 3
Accuracy 0.54
Confusion matrix
```

```
 [[18  0]
 [23  9]]
Activation function tanh
Optimizer sgd
Number of iterations 12
Number of layers 3
Accuracy 0.58
Confusion matrix
 [[ 0 21]
 [ 0 29]]
Activation function tanh
Optimizer adam
Number of iterations 461
Number of layers 3
Accuracy 0.56
Confusion matrix
 [[ 0 22]
 [ 0 28]]
Activation function relu
Optimizer lbfgs
Number of iterations 31
Number of layers 3
Accuracy 0.78
Confusion matrix
 [[19  0]
 [11 20]]
Activation function relu
Optimizer sgd
Number of iterations 683
Number of layers 3
Accuracy 0.56
Confusion matrix
 [[ 0 22]
 [ 0 28]]
Activation function relu
Optimizer adam
Number of iterations 1000
Number of layers 3
Accuracy 0.42
Confusion matrix
 [[21  0]
 [29  0]]
```

Оптимизатор lbfqs работает в среднем лучше остальных, поскольку он лучше подходит для небольших датасетов. Результаты в последнем наборе данных хуже, поскольку сами точки расположены сложнее, чем в других случаях. Одному нейрону сложно справится с таким разбросом точек.

Оптимизация для датасета nn_1.csv

Для подбора параметров будем пользоваться GridSearchCV. Для одного нейрона набор данных слишком сложный, поэтому будем пробовать большее количество слоев и нейронов в целом. Датасет небольшой, поэтому будем использовать оптимизатор lbfgs.

Получились следующие результаты:

```
Parameters {'activation': 'logistic', 'hidden_layer_sizes': (2, 3), 'max_i
ter': 1000, 'solver': 'lbfgs'}
Accuracy 0.98
Confusion matrix
 [[20  1]
 [ 0 29]]
```

Достигнута хорошая точность с сигмоидой, 2 нейронами в первом и 3 во втором слое.

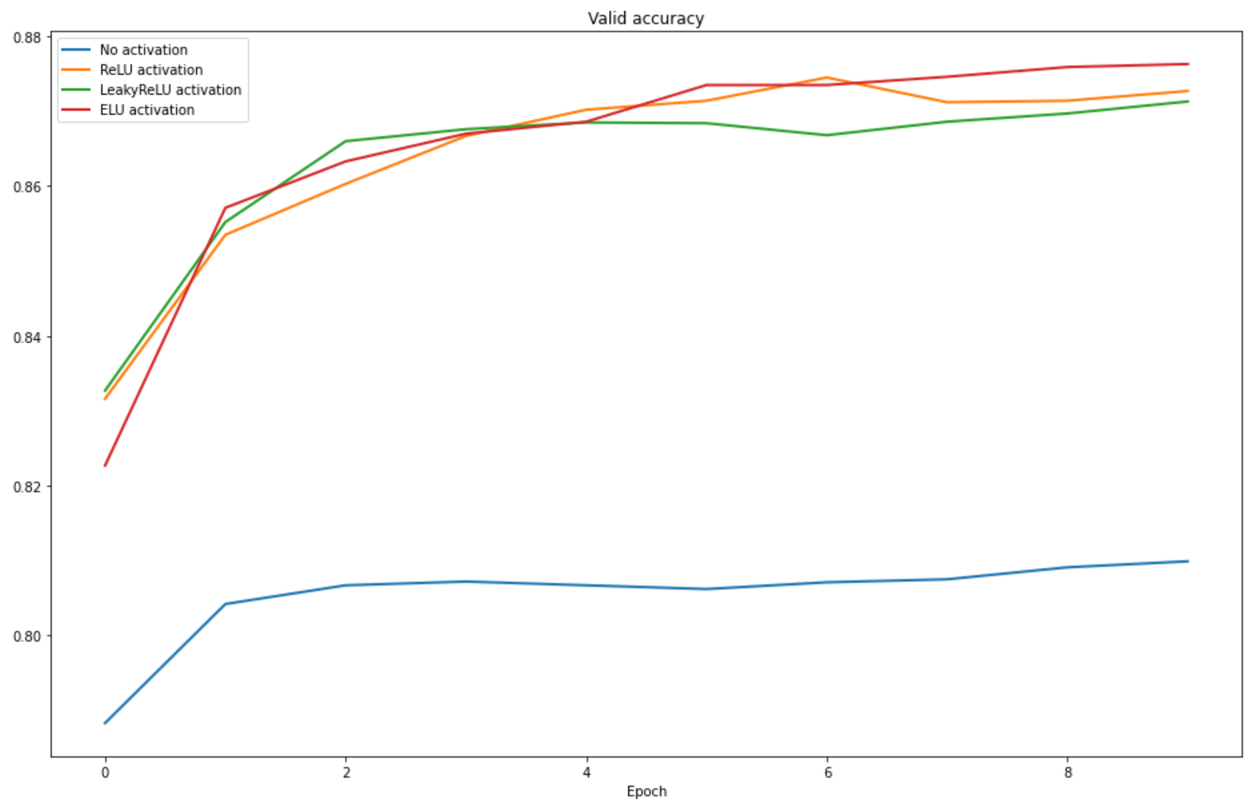
MNIST

Для решения этой задачи буду использовать библиотеку PyTorch. Я реализовал следующую нейронную сеть, в которой буду использовать различные функции активации, чтобы их сравнить.

```
activation = Identical()
d_in, dim, d_out = 28*28, 128, 10

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(d_in, dim),
    activation,
    nn.Linear(dim, dim),
    activation,
    nn.Linear(dim, d_out),
)
```

В качестве функции потерь я выбрал кросс-энтропию, в качестве оптимизатора – Адам. Сравним сеть без активации, с ReLU, LeakyReLU, ELU.



Из графика видно, что ELU показала лучший результат.

Приложения

1 и 2. 1_neuron_and_optimization.py

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

# In[86]:


import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.neural_network import MLPClassifier
from matplotlib import pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[83]:


def solve(X, y, activation='relu', optimizer='adam'):
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)

    nn = MLPClassifier(hidden_layer_sizes=1, activation=activation, solver=optimizer, max_iter=1000)
    nn.fit(X_train, y_train)

    y_pred = nn.predict(X_test)
```

```
    print('Activation function', activation)
    print('Optimizer', optimizer)
    print('Number of iterations', nn.n_iter_)
    print('Number of layers', nn.n_layers_)
    print('Accuracy', accuracy_score(y_test, y_pred))
    print('Confusion matrix \n', confusion_matrix(y_test, y_pred))


# In[84]:


activations = ['logistic', 'tanh', 'relu']
optimizers = ['lbfgs', 'sgd', 'adam']
datasets = ['nn_0.csv', 'nn_0a.csv', 'nn_1.csv']


# In[85]:


for ds in datasets:
    data = pd.read_csv(ds)
    X = data.drop('class', axis=1)
    y = data['class']

    plt.scatter(X.to_numpy()[:, 0], X.to_numpy()[:, 1], c=y, cmap=plt.cm.rainbow)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title(ds)
    plt.show()
```

```
    for activation in activations:
        for optim in optimizers:
            solve(X, y, activation, optim)


# In[88]:


data = pd.read_csv(ds)
X = data.drop('class', axis=1)
y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)


# In[91]:


params = {
    'hidden_layer_sizes': [(1, 1), (2, 2), (3, 3), (3, 2), (2, 3)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['lbfgs'],
    'max_iter': [1000]
}

nn_grid = GridSearchCV(MLPClassifier(), params, cv=3)
nn_grid.fit(X_train, y_train)
grid_pred = nn_grid.predict(X_test)
```

In[94]:

```
print('Parameters', nn_grid.best_params_)
print('Accuracy', accuracy_score(y_test, grid_pred))
print('Confusion matrix\n', confusion_matrix(y_test, grid_pred))
```

3. 3_mnist.py

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

# In[8]:



import numpy as np
from sklearn.model_selection import train_test_split

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

import os
from torchvision.datasets import MNIST
from torchvision import transforms as tfs

from matplotlib import pyplot as plt
```

```
get_ipython().run_line_magic('matplotlib', 'inline')


# In[9]:


data_tfs = tfs.Compose([
    tfs.ToTensor(),
    tfs.Normalize((0.5), (0.5))
])

# install for train and test
root = './'
train_dataset = MNIST(root, train=True,  transform=data_tfs, download=True)
val_dataset  = MNIST(root, train=False, transform=data_tfs, download=True)

train_dataloader =  DataLoader(train_dataset, batch_size=128)
valid_dataloader =  DataLoader(val_dataset, batch_size=128)


# In[10]:


class Identical(nn.Module):
    def forward(self, x):
        return x


# In[11]:
```

```
activation = Identical()
d_in, dim, d_out = 28*28, 128, 10

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(d_in, dim),
    activation,
    nn.Linear(dim, dim),
    activation,
    nn.Linear(dim, d_out),
)


# In[12]:


criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters())

loaders = {"train": train_dataloader, "valid": valid_dataloader}


# In[13]:


max_epochs = 10
```

```
accuracy = {"train": [], "valid": []}
for epoch in range(max_epochs):
    for k, dataloader in loaders.items():
        epoch_correct = 0
        epoch_all = 0
        for x_batch, y_batch in dataloader:
            if k == "train":
                model.train()
                optimizer.zero_grad()
                outp = model(x_batch)
                loss = criterion(outp, y_batch)
                loss.backward()
                optimizer.step()
            else:
                model.eval()
                with torch.no_grad():
                    outp = model(x_batch)
            preds = outp.argmax(-1)
            correct =  preds[preds==y_batch]
            all = len(y_batch)
            epoch_correct += correct.count_nonzero()
            epoch_all += all

        if k == "train":
            print(f"Epoch: {epoch+1}")
        print(f"Loader: {k}. Accuracy: {epoch_correct/epoch_all}")
        accuracy[k].append(epoch_correct/epoch_all)
```

```python
# In[14]:


elu_accuracy = accuracy["valid"]


# In[15]:



def test_activation_function(activation):
    d_in, dim, d_out = 28*28, 128, 10

    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(d_in, dim),
        activation,
        nn.Linear(dim, dim),
        activation,
        nn.Linear(dim, d_out)
    )

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters())
    loaders = {"train": train_dataloader, "valid": valid_dataloader}

    max_epochs = 10
    accuracy = {"train": [], "valid": []}
    for epoch in range(max_epochs):
        for k, dataloader in loaders.items():
```

```
        epoch_correct = 0
        epoch_all = 0
        for x_batch, y_batch in dataloader:
            if k == "train":
                model.train()
                optimizer.zero_grad()
                outp = model(x_batch)
                loss = criterion(outp, y_batch)
                loss.backward()
                optimizer.step()
            else:
                model.eval()
                with torch.no_grad():
                    outp = model(x_batch)
            preds = outp.argmax(-1)
            correct =  preds[preds==y_batch]
            all = len(y_batch)
            epoch_correct += correct.count_nonzero()
            epoch_all += all
        if k == "train":
            print(f"Epoch: {epoch+1}")
        print(f"Loader: {k}. Accuracy: {epoch_correct/epoch_all}")
        accuracy[k].append(epoch_correct/epoch_all)
    return accuracy["valid"]


# In[ ]:
```

```
plain_accuracy = test_activation_function(Identical())
relu_accuracy = test_activation_function(nn.ReLU())
leaky_relu_accuracy = test_activation_function(nn.LeakyReLU())



# In[ ]:



plt.figure(figsize=(16, 10))
plt.title("Valid accuracy")
plt.plot(range(max_epochs), plain_accuracy, label="No activation", linewidth=2)
plt.plot(range(max_epochs), relu_accuracy, label="ReLU activation", linewidth=2)
plt.plot(range(max_epochs), leaky_relu_accuracy, label="LeakyReLU activation",
linewidth=2)
plt.plot(range(max_epochs), elu_accuracy, label="ELU activation", linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```