 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №6**

з дисципліни «Методи та технології обчислювального інтелекту»

на тему: «Нейронна мережа для задачі регресії»

**Виконала:**

студентка гр. БС-01мп

Харченко Наталя

**Перевірив:**

доц. Федорін І.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

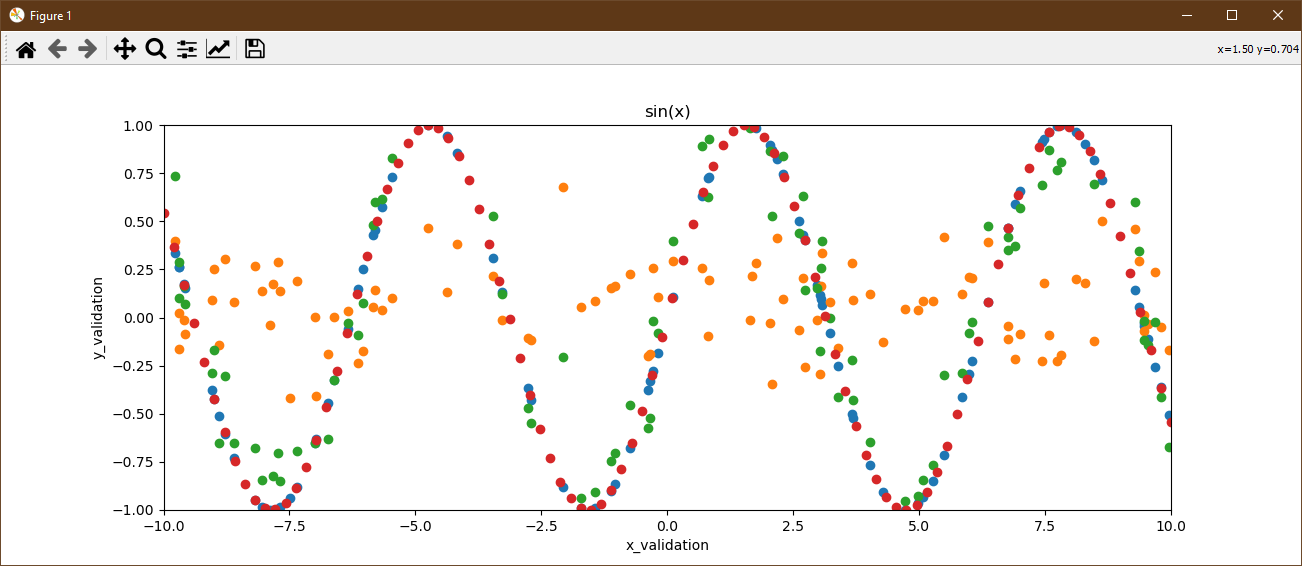
Київ-2020

**Практичне завдання**

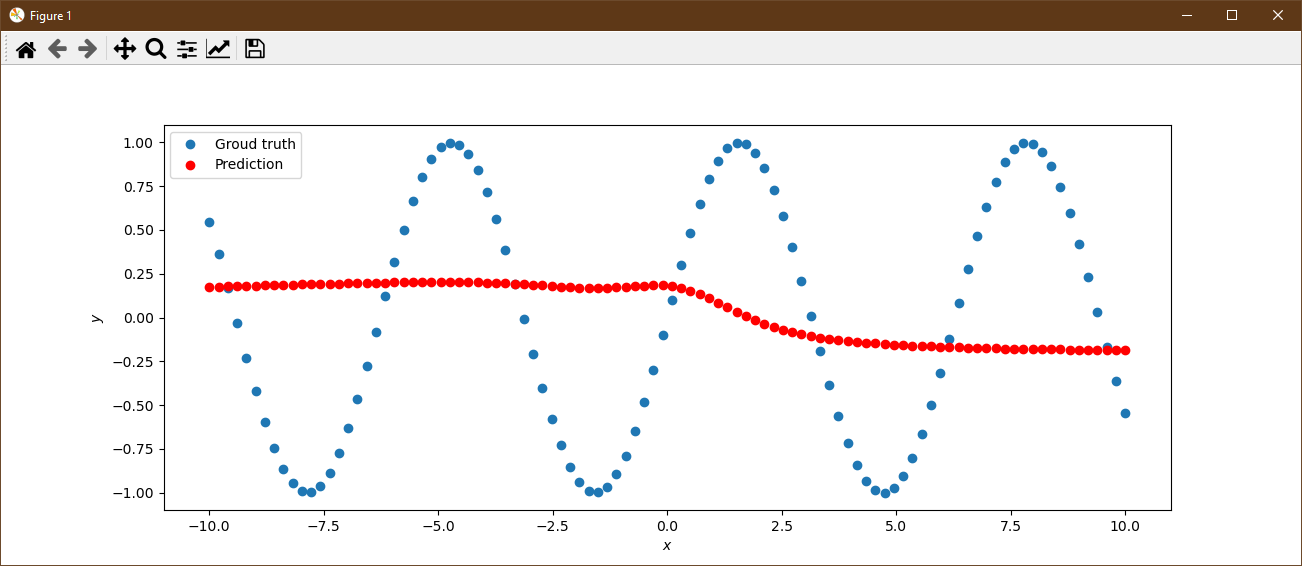
**Лістинг програми:**

import torch  
import matplotlib.pyplot as plt  
import matplotlib  
matplotlib.rcParams['figure.figsize'] = (13.0, 5.0)  
  
x\_train = torch.rand(100)  
x\_train = x\_train \* 20.0 - 10.0  
y\_train = torch.sin(x\_train)  
plt.plot(x\_train.numpy(), y\_train.numpy(), 'o')  
plt.title('$y = sin(x)$')  
  
  
noise = torch.randn(y\_train.shape) / 5.  
plt.plot(x\_train.numpy(), noise.numpy(), 'o')  
plt.axis([-10, 10, -1, 1])  
plt.title('Gaussian noise')  
  
y\_train = y\_train + noise  
plt.plot(x\_train.numpy(), y\_train.numpy(), 'o')  
plt.title('noisy sin(x)')  
plt.xlabel('x\_train')  
plt.ylabel('y\_train')  
  
x\_train.unsqueeze\_(1)  
y\_train.unsqueeze\_(1)  
  
x\_validation = torch.linspace(-10, 10, 100)  
y\_validation = torch.sin(x\_validation.data)  
plt.plot(x\_validation.numpy(), y\_validation.numpy(), 'o')  
plt.title('sin(x)')  
plt.xlabel('x\_validation')  
plt.ylabel('y\_validation')  
x\_validation.unsqueeze\_(1)  
y\_validation.unsqueeze\_(1)  
plt.show()  
  
class SineNet(torch.nn.Module):  
 def \_\_init\_\_(self, n\_hidden\_neurons):  
 super(SineNet, self).\_\_init\_\_()  
 self.fc1 = torch.nn.Linear(1, n\_hidden\_neurons)  
 self.act1 = torch.nn.Tanh()  
 self.fc2 = torch.nn.Linear(n\_hidden\_neurons, 1)  
  
 def forward(self, x):  
 x = self.fc1(x)  
 x = self.act1(x)  
 x = self.fc2(x)  
 return x  
  
  
sine\_net = SineNet(50)  
print(sine\_net)  
  
  
def predict(net, x, y):  
 y\_pred = net.forward(x)  
 plt.plot(x.numpy(), y.numpy(), 'o', label='Groud truth')  
 plt.plot(x.numpy(), y\_pred.data.numpy(), 'o', c='r', label='Prediction')  
 plt.legend(loc='upper left')  
 plt.xlabel('$x$')  
 plt.ylabel('$y$')  
  
 plt.show()  
  
  
predict(sine\_net, x\_validation, y\_validation)  
  
optimizer = torch.optim.Adam(sine\_net.parameters(), lr=0.01)  
  
  
def loss(pred, target):  
 squares = (pred - target) \*\* 2  
 return squares.mean()  
  
  
for epoch\_index in range(2000):  
 optimizer.zero\_grad()  
 y\_pred = sine\_net.forward(x\_train)  
 loss\_val = loss(y\_pred, y\_train)  
 loss\_val.backward()  
 optimizer.step()  
  
  
predict(sine\_net, x\_validation, y\_validation)

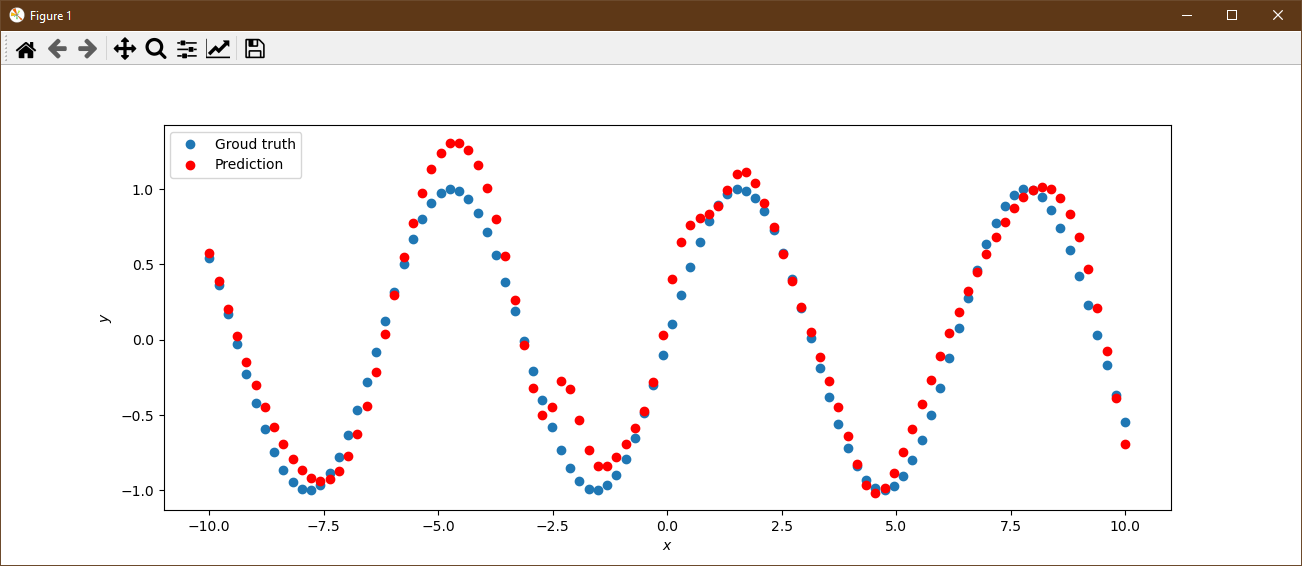
**Результати:**



*До навчання*

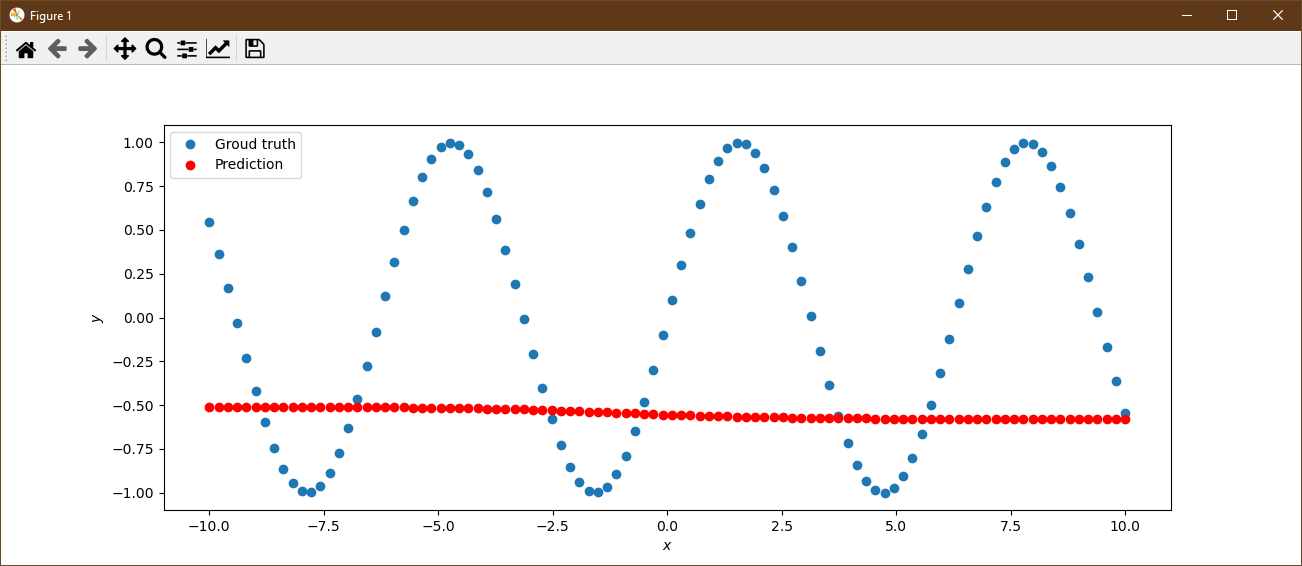


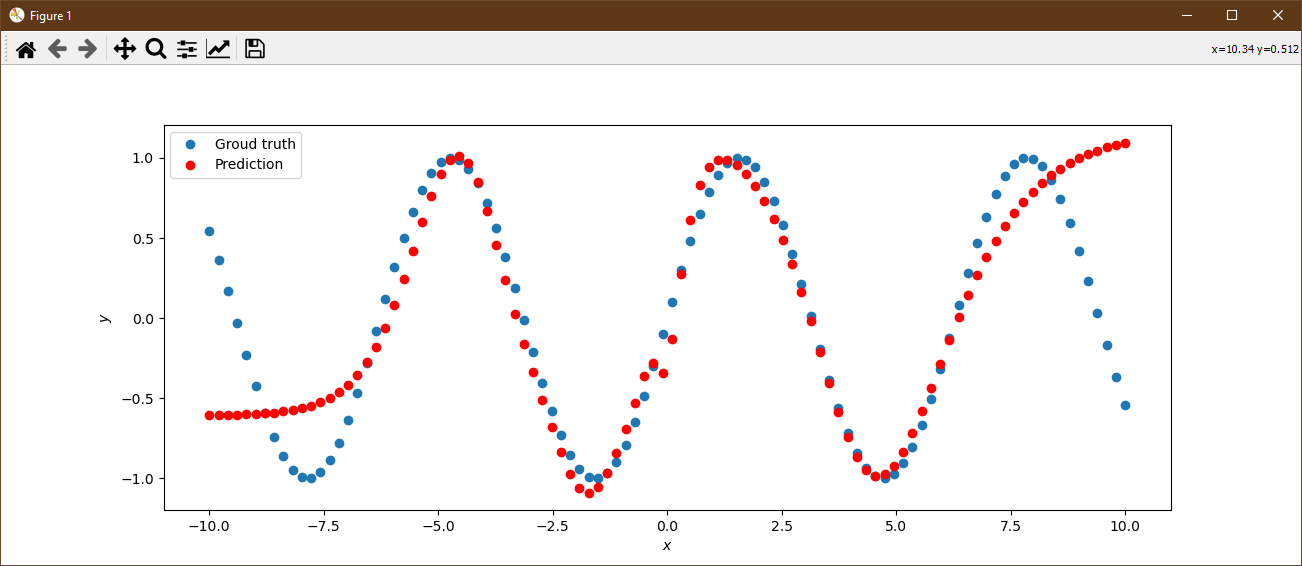
*Після навчання*



1. Додайте ще один fc-шар
2. Замінить активацію між шарами на будь-яку іншу та протестувати відповідь

class SineNet(torch.nn.Module):  
 def \_\_init\_\_(self, n\_hidden\_neurons):  
 super(SineNet, self).\_\_init\_\_()  
 self.fc1 = torch.nn.Linear(1, n\_hidden\_neurons)  
 self.act1 = torch.nn.Sigmoid()  
 self.fc2 = torch.nn.Linear(n\_hidden\_neurons, 25)  
 self.act2 = torch.nn.Sigmoid()  
 self.fc3 = torch.nn.Linear(25, 1)  
  
  
 def forward(self, x):  
 x = self.fc1(x)  
 x = self.act1(x)  
 x = self.fc2(x)  
 x = self.act2(x)  
 x = self.fc3(x)  
 return x





**Проекспериментуйте із отриманою нейромережею**:

1. Спробуйте змінити кількість нейронів (1,2,3,4,5,6,7,8,9,10 – проаналізуйте як змінюються передбачення)

*Чим меньше нейронів, ти гірше передбачення після обробки нейромережею.*

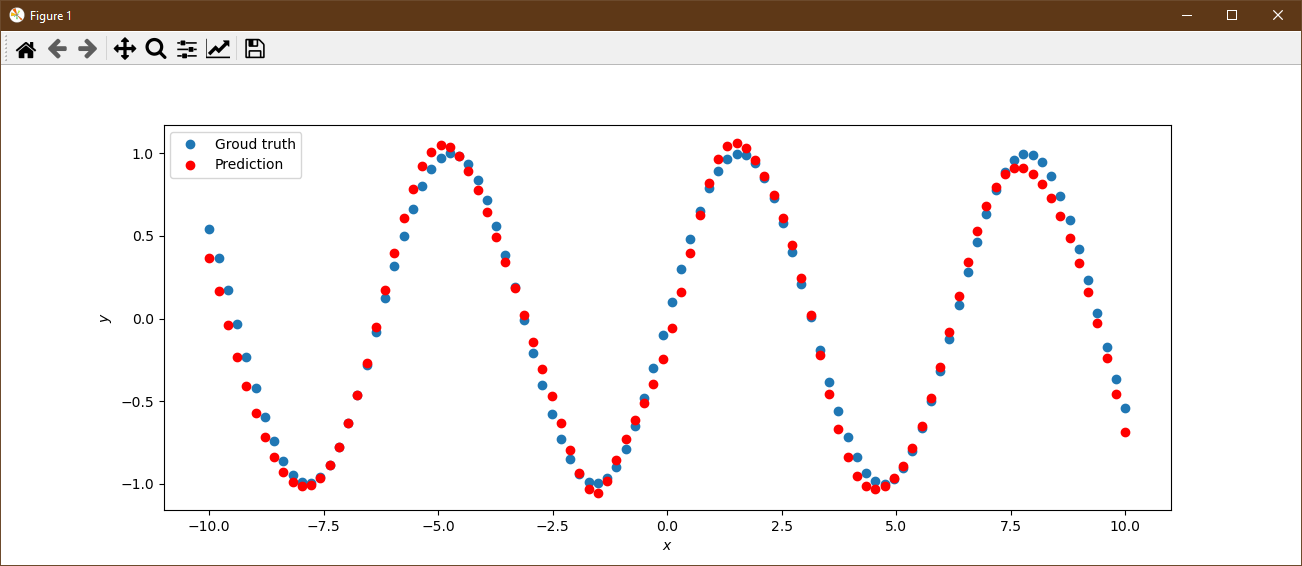
*Якщо буде дуже багато нейронів то ми можеме перенавчити модель, що також є погано, вона не буде відавати реальні результати.*

1. Спробуйте змінити кількість епох навчання

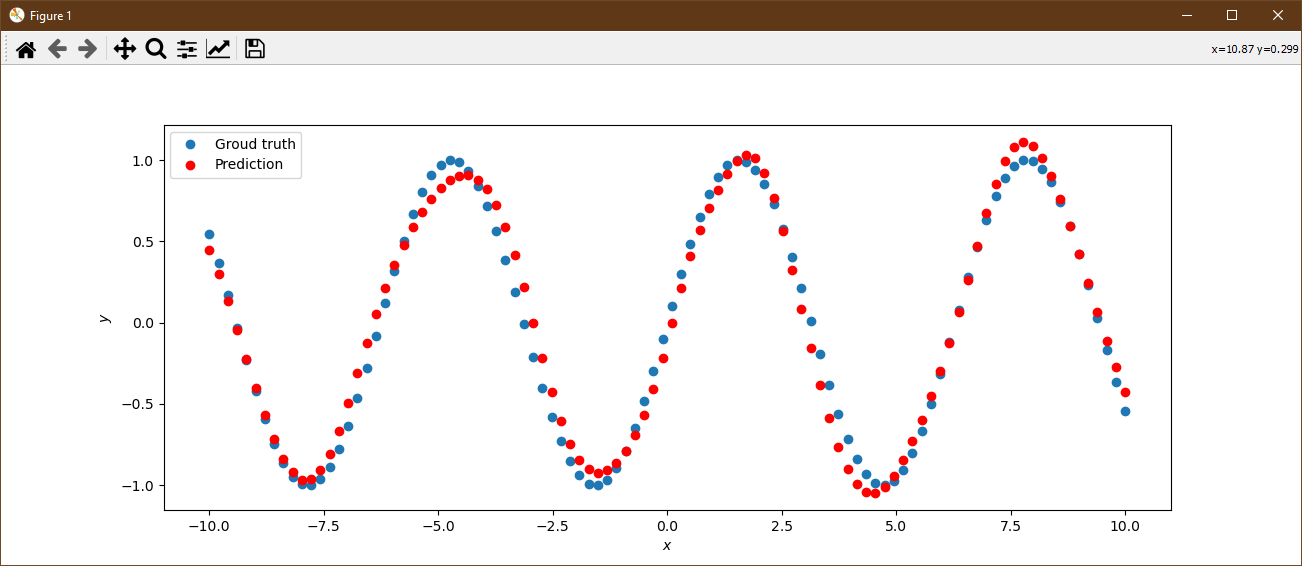
*Теж, саме, що і з нейронами. Необхідно знайти приблизне значення щоб модель навчилася, та не перенавчалася.*

1. Спробуйте змінити функцію активації

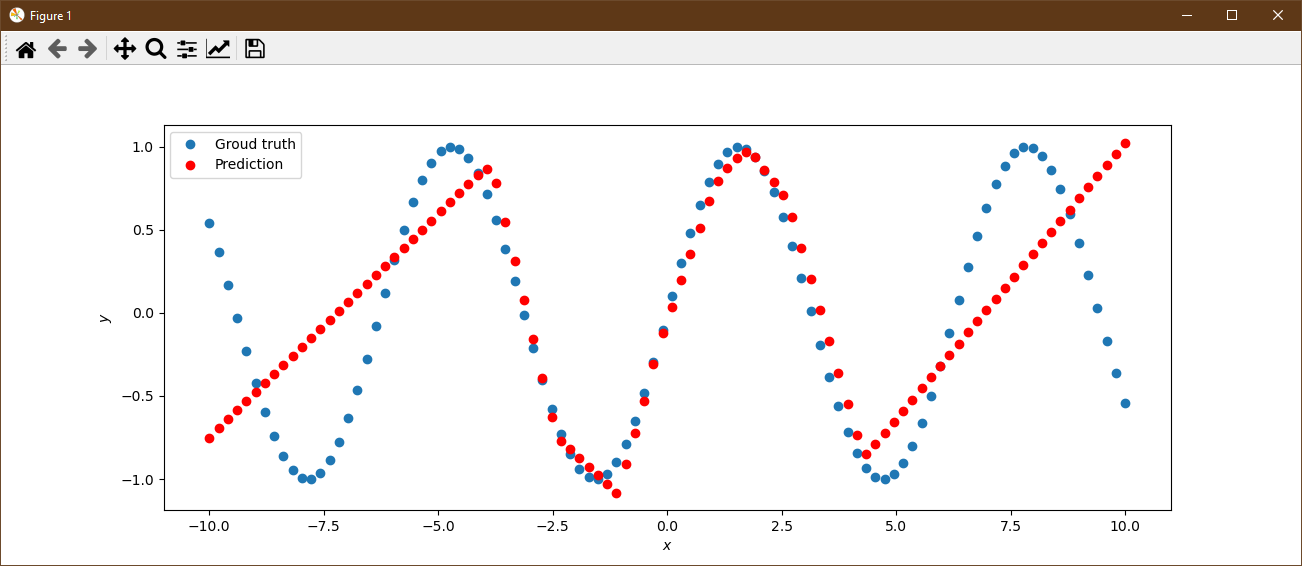
Tanh



Sigmoid



ReLU



*Мы бачимо що функція ReLU підходить найменше для нашого варінату.*

1. Спробуйте змінити (зменшити або додати додаткові шари)

*Збільшення кількості шарів погано впліває.*

1. Спробуйте змінити функцію втрат

*При зміні функції втрат у нас буде по іншому рахуватися похибка.*

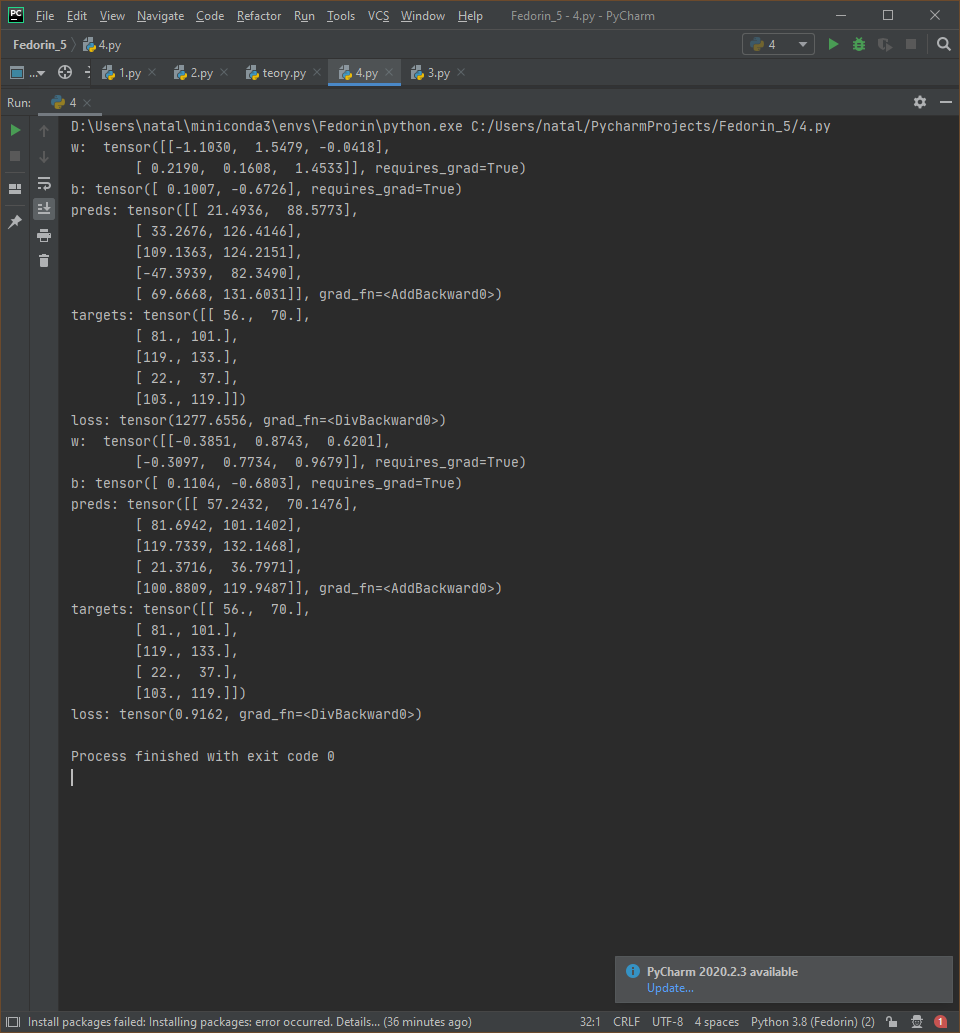
1. Запишіть значення функції втрат після кожної ітерації та виведіть графік залежності функції втрат від номера епоха.

**Лістинг програми:**

import torch  
import numpy as np  
  
# Input (temp, rainfall, humidity)  
inputs = np.array([[73, 67, 43],  
 [91, 88, 64],  
 [87, 134, 58],  
 [102, 43, 37],  
 [69, 96, 70]], dtype='float32')  
# Targets (apples, oranges)  
targets = np.array([[56, 70],  
 [81, 101],  
 [119, 133],  
 [22, 37],  
 [103, 119]], dtype='float32')  
  
# Convert inputs and targets to tensors  
inputs = torch.from\_numpy(inputs)  
targets = torch.from\_numpy(targets)  
  
w = torch.randn(2, 3, requires\_grad=True)  
b = torch.randn(2, requires\_grad=True)  
  
  
# MSE loss  
def mse(t1, t2):  
 diff = t1 - t2  
 return torch.sum(diff \* diff) / diff.numel()  
  
  
def model(x):  
 return x @ w.t() + b  
  
print('w: ', w, '\nb:', b)  
# Generate predictions  
preds = model(inputs)  
# Calculate the loss  
loss = mse(preds, targets)  
# Compute gradients  
loss.backward()  
print('preds:', preds)  
print('targets:', targets)  
print('loss:', loss)  
  
for \_ in range(2500):  
 preds = model(inputs)  
 loss = mse(preds, targets)  
 loss.backward()  
 # Adjust weights & reset gradients  
 with torch.no\_grad():  
 w -= w.grad \* 1e-5  
 b -= b.grad \* 1e-5  
 w.grad.zero\_()  
 b.grad.zero\_()  
  
print('w: ', w, '\nb:', b)  
preds = model(inputs)  
loss = mse(preds, targets)  
print('preds:', preds)  
print('targets:', targets)  
print('loss:', loss)

**Результат:**

*До тренування*



*Після тренування*

