Національний технічний університет України

«Київський політехнічний інститут ім. І. Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Методи та технології штучного інтелекту

Лабораторна робота №4

«Моделювання функції двох змінних з двома входами і одним виходом на основі нейронних мереж»

Виконав:

студент групи ІО-21

Безщасний Р. Р.

Номер у списку групи: 2

Перевірив:

Шимкович Володимир Миколайович

Київ 2024р.

**Тема:** «Моделювання функції двох змінних з двома входами і одним виходом на основі нейронних мереж».

**Мета:** Дослідити структуру та принцип роботи нейронної мережі. За допомогою нейронної мережі змоделювати функцію двох змінних.

**Індивідуальне завдання:**

**Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис**

**Код виконання завдання з результатами**

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

def main\_func(x, y):

  return (x - y) \* np.sin(x + y)

x = np.linspace(-8, 8, 10)

y = np.linspace(-4, 4, 10)

x, y = np.meshgrid(x, y)

z = main\_func(x, y)

# Reshape and split

x\_y\_data = np.column\_stack((x.ravel(), y.ravel()))

z\_data = z.ravel()

# Train-test split

x\_y\_train, x\_y\_test, z\_train, z\_test = train\_test\_split(x\_y\_data, z\_data, test\_size=0.4, random\_state=42)

# Normalize inputs

scaler = StandardScaler()

x\_y\_train = scaler.fit\_transform(x\_y\_train)

x\_y\_test = scaler.transform(x\_y\_test)

# Convert to tensors

X\_Y\_train = torch.FloatTensor(x\_y\_train)

X\_Y\_test = torch.FloatTensor(x\_y\_test)

Z\_train = torch.FloatTensor(z\_train) # Unsqueeze for compatibility

Z\_test = torch.FloatTensor(z\_test)

X\_Y\_train.shape, X\_Y\_test.shape, Z\_train.shape, Z\_test.shape

class FeedForwardNetwork(nn.Module):

  def \_\_init\_\_(self, input\_size, hidden\_layers):

    super(FeedForwardNetwork, self).\_\_init\_\_()

    layers = []

    prev\_size = input\_size

    for hidden\_size in hidden\_layers:

      layers.append(nn.Linear(prev\_size, hidden\_size))

      layers.append(nn.ReLU())

      prev\_size = hidden\_size

    layers.append(nn.Linear(prev\_size, 1))

    self.model = nn.Sequential(\*layers)

  def forward(self, x):

    return self.model(x)

class CascadeNetwork(nn.Module):

  def \_\_init\_\_(self, input\_size, hidden\_layers):

    super(CascadeNetwork, self).\_\_init\_\_()

    self.hidden\_layers = nn.ModuleList()

    self.input\_size = input\_size

    prev\_size = input\_size

    for hidden\_size in hidden\_layers:

      self.hidden\_layers.append(nn.Linear(prev\_size + input\_size, hidden\_size))

      prev\_size = hidden\_size

    self.output = nn.Linear(prev\_size + input\_size, 1)

  def forward(self, x):

    out = x

    for layer in self.hidden\_layers:

      out = torch.cat([x, out], dim=1)

      out = torch.relu(layer(out))

    out = torch.cat([x, out], dim=1)

    return self.output(out)

class ElmanNetwork(nn.Module):

  def \_\_init\_\_(self, input\_size, hidden\_sizes, output\_size=1):

    super(ElmanNetwork, self).\_\_init\_\_()

    self.hidden\_sizes = hidden\_sizes

    self.input\_to\_hidden = nn.ModuleList(

    [nn.Linear(input\_size + hidden\_size, hidden\_size) for hidden\_size in hidden\_sizes]

    )

    self.hidden\_to\_output = nn.Linear(hidden\_sizes[-1], output\_size)

    self.tanh = nn.Tanh()

  def forward(self, x, hidden):

    combined = torch.cat((x, hidden[0]), 1)

    hidden\_next = []

    for i, input\_to\_hidden\_layer in enumerate(self.input\_to\_hidden):

      hidden\_i = self.tanh(input\_to\_hidden\_layer(combined))

      combined = torch.cat((x, hidden\_i), 1)

      hidden\_next.append(hidden\_i)

    output = self.hidden\_to\_output(hidden\_next[-1])

    return output, hidden\_next

  def init\_hidden(self, batch\_size):

    return [torch.zeros(batch\_size, hidden\_size) for hidden\_size in self.hidden\_sizes]

while True:

  user\_choice = input("Input: ")

  if user\_choice == '1':

    model = FeedForwardNetwork(input\_size=2, hidden\_layers=[50, 50])

  elif user\_choice == '2':

    model = FeedForwardNetwork(input\_size=2, hidden\_layers=[20])

  elif user\_choice == '3':

    model = CascadeNetwork(input\_size=2, hidden\_layers=[20])

  elif user\_choice == '4':

    model = CascadeNetwork(input\_size=2, hidden\_layers=[10, 10])

  elif user\_choice == '5':

    model = ElmanNetwork(input\_size=2, hidden\_sizes=[15])

  elif user\_choice == '6':

    model = ElmanNetwork(input\_size=2, hidden\_sizes=[5, 5, 5])

  elif user\_choice == '7':

    break

  else:

    continue

  predictions, mse, loss\_curve = train\_model(model, epochs=500, learning\_rate=0.05)

  print(f"\nMSE: {mse:.4f}")

  plt.figure(figsize=(12, 5))

  plt.subplot(2, 2, 1)

  plt.plot(z\_data)

  plt.title('Графік z = 0.5 \* cos(a+b) \* cos(a)')

  plt.xlabel('a')

  plt.ylabel('b')

  plt.grid()

  plt.subplot(2, 2, 3)

  plt.plot(predictions)

  plt.title('Графік нейронної мережі')

  plt.xlabel('a')

  plt.ylabel('b')

  plt.grid()

  plt.subplot(2, 2, 2)

  plt.plot(loss\_curve, color='green')

  plt.xlabel('Епохи')

  plt.ylabel('Втрати')

  plt.title('Графік залежності помилки від епохи')

  plt.ylim(min(loss\_curve) \* 0.9, max(loss\_curve) \* 1.1)

  plt.grid()

  plt.tight\_layout()

  plt.show()

***Результати виконання програми***

Зображення, що містить текст, знімок екрана, Графік, схема

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Графік, ряд

Автоматично згенерований опис

***Висновок***

*На даній лабораторній роботі я дослідив структуру та принцип роботи нейронної мережі та за допомогою нейронної мережі змоделював функцію двох змінних в 6 різних варіантів по 2 з типу мережі (forward back propagation, cascade forward back propagation та elman back propagation)*