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:

Індивідуальне завдання

За допомогою програмних засобів моделювання або мови програмування високого рівня створити та дослідити вплив кількості внутрішніх шарів та кількості нейронів на середню відносну помилку моделювання для різних типів мереж (feed forward backprop, cascade - forward backprop, elman backprop):

1. Тип мережі: feed forward backprop:
   a) 1 внутрішній шар з 10 нейронами;
   b) 1 внутрішній шар з 20 нейронами;
2. Тип мережі: cascade - forward backprop:
   a) 1 внутрішній шар з 20 нейронами;
   b) 2 внутрішніх шари по 10 нейронів у кожному; 3.Тип мережі: elman backprop:
   a) 1 внутрішній шар з 15 нейронами;
   b) 3 внутрішніх шари по 5 нейронів у кожному;
4. Зробити висновки на основі отриманих даних.

Код виконання завдання з результатами

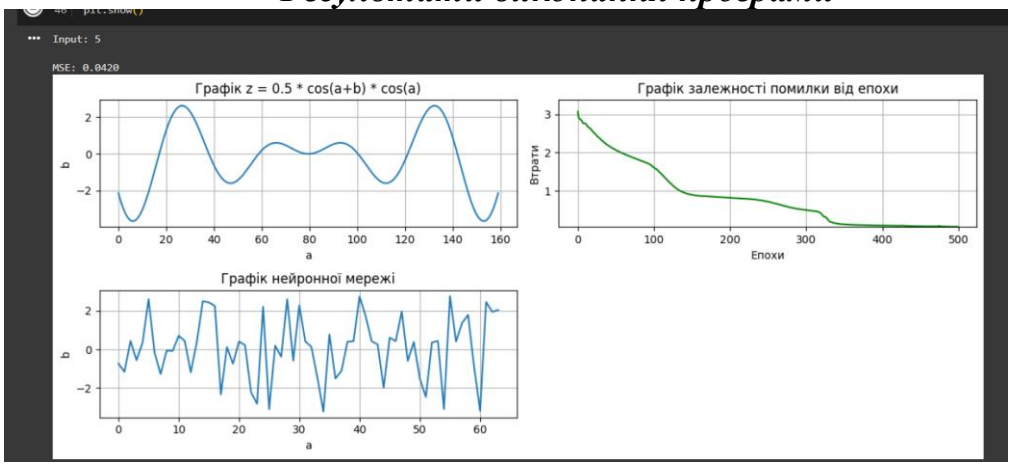
```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

def main_func(x, y):
  return (x - y) * np.sin(x + y)

x = np.linspace(-8, 8, 10)
y = np.linspace(-4, 4, 10)
```

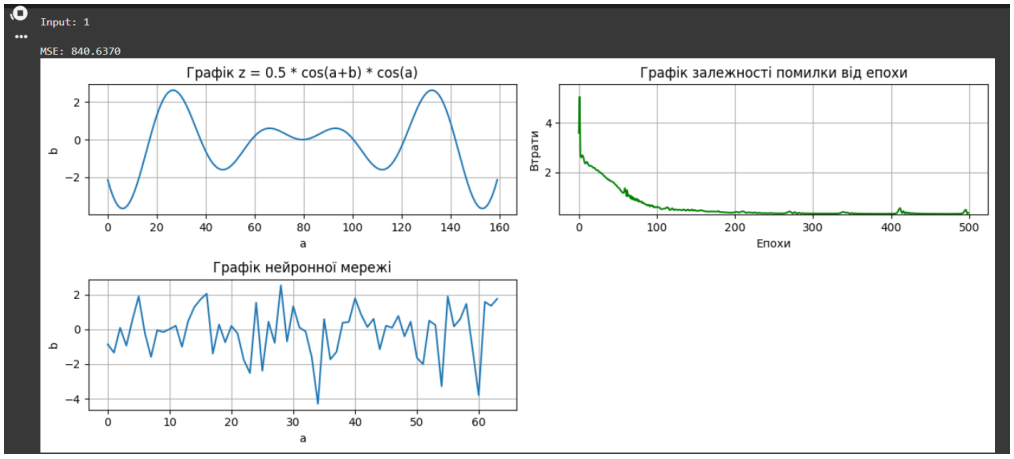
```
x, y = np.meshgrid(x, y)
z = main_func(x, y)

# 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





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)