

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Звіт до лабораторної роботи №2
«Реалізація базових архітектур нейронних мереж»
з дисципліни
«Програмні засоби проектування та реалізації нейромережевих систем»

Виконала:
студентка групи ІМ-13
Мартинюк Марія Павлівна
номер у списку: 63

Перевірів:
Шимкович В. М.

Київ 2024

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

Завдання: Написати програму, що реалізує нейронні мережі для моделювання функції двох змінних. Функцію двох змінних, типу $f(x+y) = x^2 + y^2$, обрати самостійно. Промодельовати на невеликому відрізку, скажімо від 0 до 10. 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:
 - а) 1 внутрішній шар з 10 нейронами;
 - б) 1 внутрішній шар з 20 нейронами;
2. Тип мережі: cascade - forward backprop:
 - а) 1 внутрішній шар з 20 нейронами;
 - б) 2 внутрішніх шари по 10 нейронів у кожному;
3. Тип мережі: elman backprop:
 - а) 1 внутрішній шар з 15 нейронами;
 - б) 3 внутрішніх шари по 5 нейронів у кожному;
4. Зробити висновки на основі отриманих даних.

Варіант 3:

3.	$y = x \cdot \sin(x)$
	$z = x \cdot \cos(y) + \sin(x)$

Хід роботи:

Спершу для наочності було реалізовано додаткову функцію для виведення першочергового графіку функції $z(x, y)$ на проміжку $[0, 5]$. Виведення та збереження результату відбувається у файл з відповідною назвою.

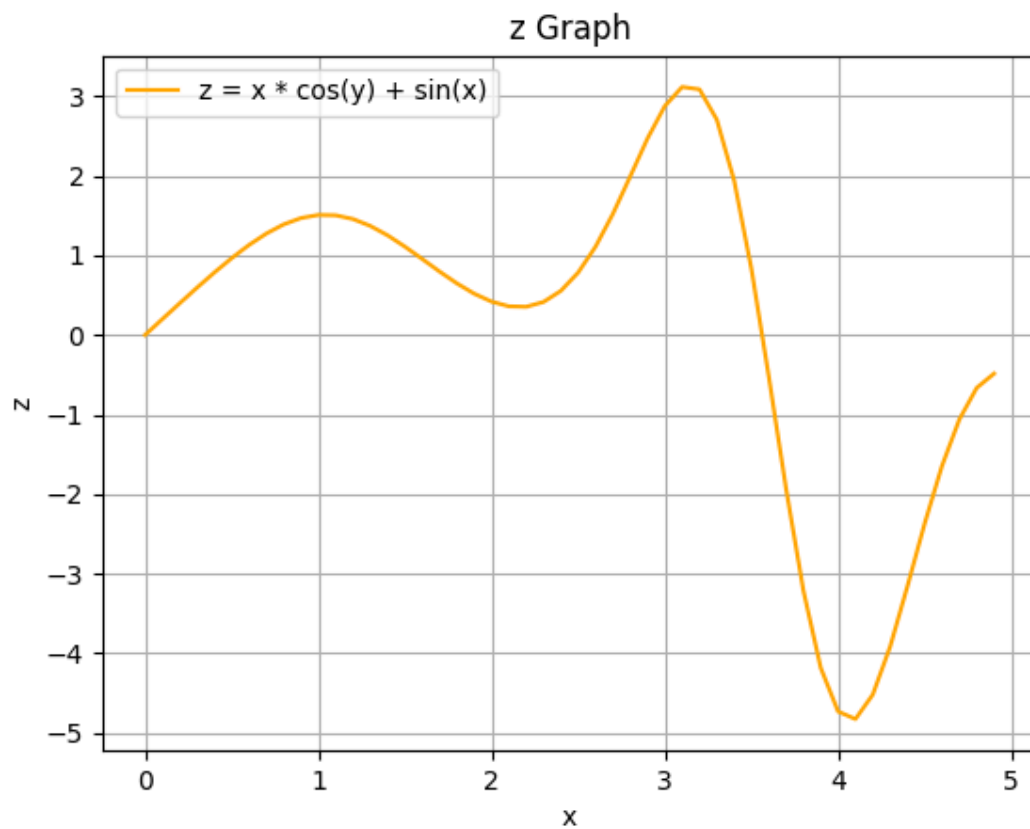
```
def initial_functions_graph():
    x = np.arange(0, 5, 0.1)
    y = x * np.sin(x)
    z = x * np.cos(y) + np.sin(x)

    plt.plot(x, z, label='z = x * cos(y) + sin(x)', color='orange')
    plt.xlabel('x')
    plt.ylabel('z')
    plt.legend()
    plt.grid(True)
    plt.title('z Graph')

    plt.savefig('function_z.png')
    plt.close()

initial_functions_graph()
```

Результат виконання:



Далі було розраховано тренувальні та тестувальні значення, що пізніше використовувались для навчання та перевірки роботи нейромереж. Окрім цього було додатково оголошено розміром вхідних шарів, вихідних шарів, епох, ввідні дані та цільові дані.

```
def y_values(x):
    return x * np.sin(x)

def z_values(x, y):
    return x * np.cos(y) + np.sin(x)

def generate_values():
    x_train = np.linspace(0, 5, 80)
    y_train = y_values(x_train)
    z_train = z_values(x_train, y_train)

    x_test = np.linspace(1, 3, 20)
    y_test = y_values(x_test)
    z_test = z_values(x_test, y_test)

    return (
        np.vstack((x_train, y_train)).T,
        np.vstack((x_test, y_test)).T,
        z_train,
        z_test
    )

# training data
x_train, x_test, y_train, y_test = generate_values()
input_dim = x_train.shape[1]
output_dim = 1
epochs = 5000
inputs = torch.Tensor(x_train)
targets = torch.Tensor(y_train).view(-1, 1)
```

Для розрахунку відносної помилки було реалізовано наступну функцію:

```
def model_error_count(test_inputs, test_targets, model):
    predictions = model(test_inputs)
    abs_relative_error = torch.abs((test_targets - predictions) /
test_targets)
    mean_relative_error = torch.mean(abs_relative_error) * 100
```

```
return mean_relative_error
```

Також було реалізовано дві додаткові функції для створення графіків: для порівняння результатів навчених моделей та першочергової функції та для відображення зміни похибки відносно епох.

```
# functions for graphs display
def result_graph_display(res, target, prediction, title, err):
    plt.plot(res, target, label='Test', color='green')
    plt.plot(res, prediction, label='Prediction', color='red')
    plt.title(title + '\nEvaluation error ' + err + '%')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.savefig(title + '.png')
    plt.close()

def loss_graph_display(epoch_list, error_list, title):
    plt.plot(epoch_list, error_list, label='Error', color='blue')
    plt.title('Error Dependency on Epoch - ' + title)
    plt.xlabel('Epoch')
    plt.ylabel('Error (%)')
    plt.legend()
    plt.grid(True)
    plt.savefig(title + '_error_dependency.png')
    plt.close()
```

1. Мережа feed forward backprop:

- a) 1 внутрішній шар з 10 нейронами;
- b) 1 внутрішній шар з 20 нейронами;

```
# 1.Feed Forward Backprop neural network
class FeedForwardBackprop (nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(FeedForwardBackprop, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, output_dim)
```

```

def forward(self, x):
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    return x

def ffb_network_test(hidden_dim, title, version):
    model = FeedForwardBackprop(input_dim, hidden_dim, output_dim)
    optimizer = optim.RMSprop(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()

    # Lists to store epoch and error values
    epoch_list = []
    error_list = []

    for epoch in range(epochs):
        optimizer.zero_grad()
        predictions = model(inputs)
        loss = criterion(predictions, targets)
        loss.backward()
        optimizer.step()

        # Calculate error for the current epoch
        if epoch % 10 == 0: #frequency of error calculation
            model.eval()
            test_inputs = torch.Tensor(x_test)
            test_targets = torch.Tensor(y_test).view(-1, 1)
            relative_error_value = model_error_count(test_inputs,
test_targets, model).item()
            epoch_list.append(epoch)
            error_list.append(relative_error_value)

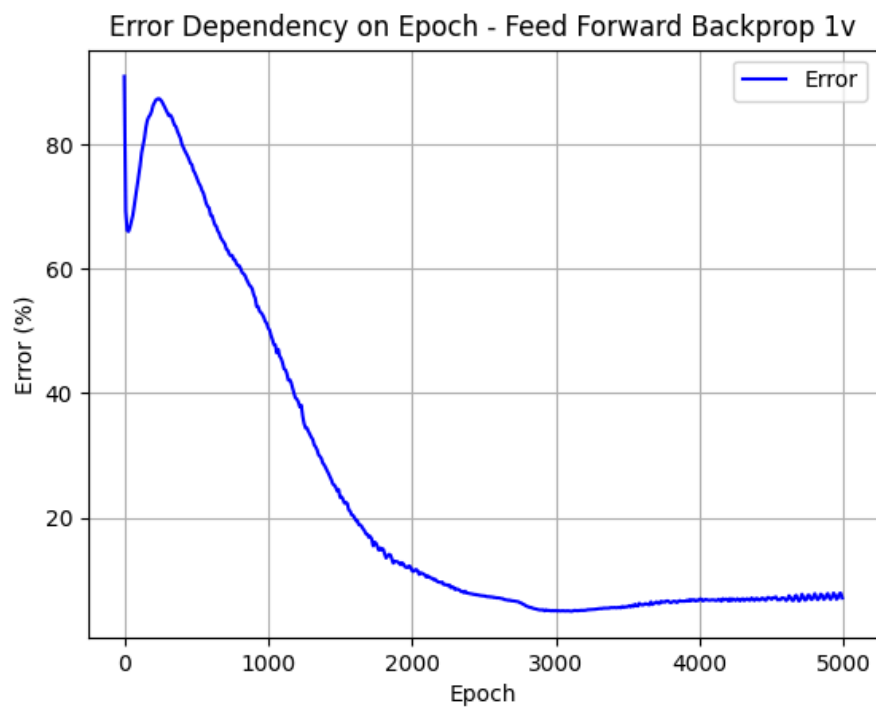
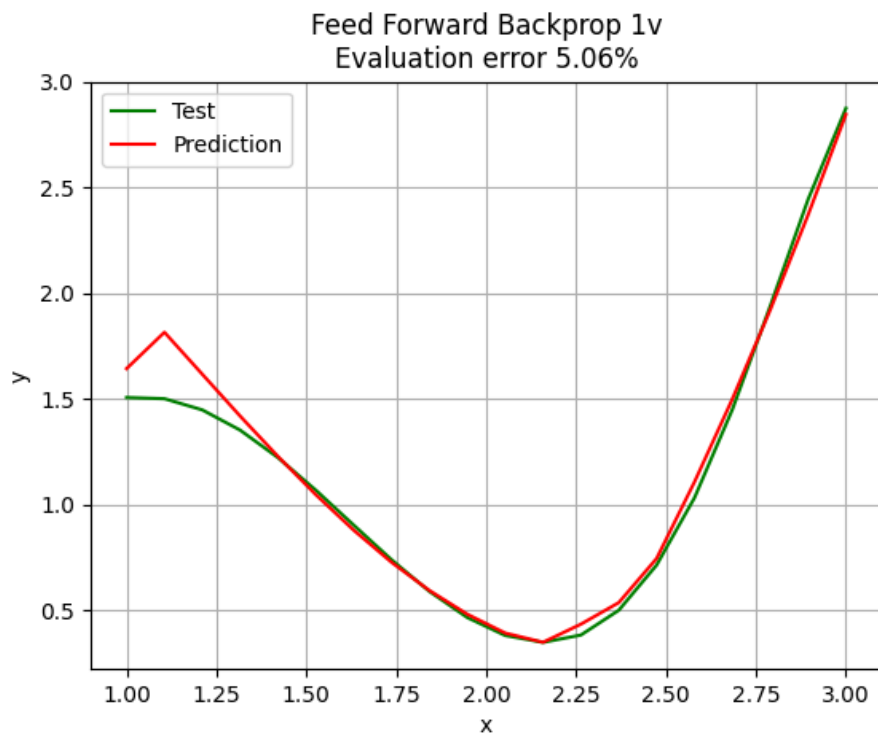
    model.eval()
    test_inputs = torch.Tensor(x_test)
    test_targets = torch.Tensor(y_test).view(-1, 1)
    relative_error_value = model_error_count(test_inputs,
test_targets, model).item()
    err = str(round(relative_error_value, 2))
    print('FFB Evaluation Error ' + version + ' = ' + err + '%')

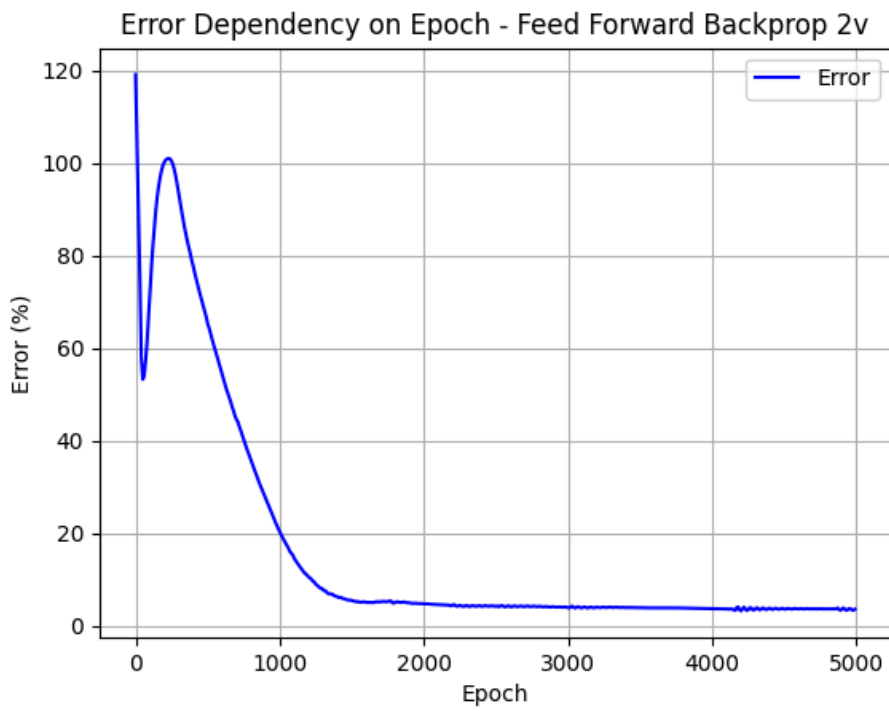
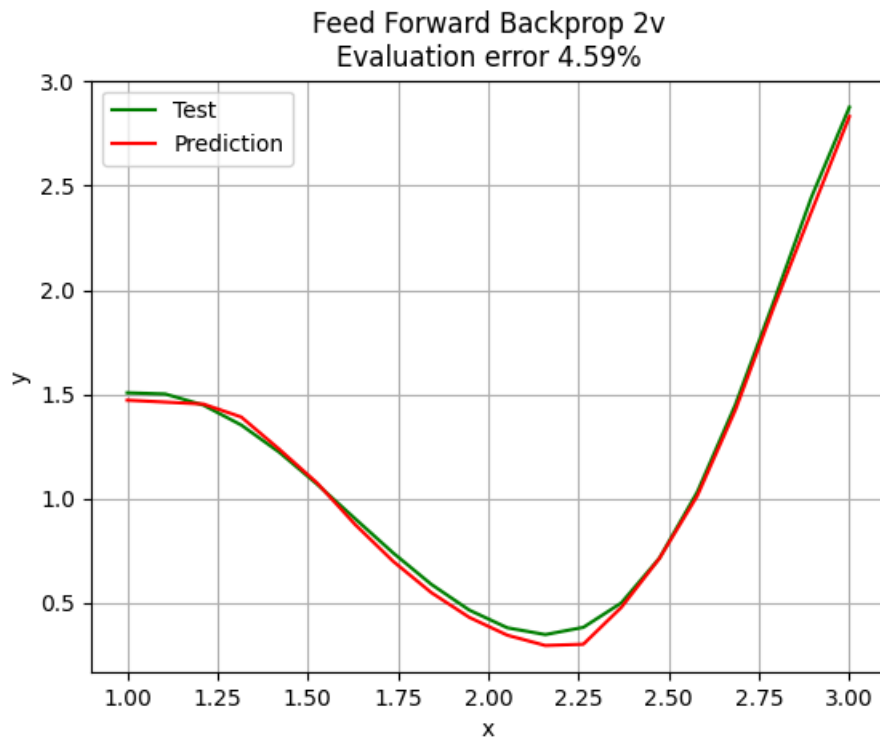
    result_graph_display(x_test[:, 0], y_test,
model(test_inputs).detach().numpy(), title, err)
    loss_graph_display(epoch_list, error_list, title)

```

```
ffbnetwork_test(10, 'Feed Forward Backprop 1v', 'v1')  
ffbnetwork_test(20, 'Feed Forward Backprop 2v', 'v2')
```

Результати:





Як видно зі графіків вище, навчання мережі є досить результативним. Помилка у розрахунках становить близько 5.06% у першому випадку з 10 нейронами та близько 4.59% у другому з 20 нейронами. Таким чином можна зробити висновок, що збільшення кількості нейронів сприяє кращому навчанню мережі.

2. Мережа cascade - forward backprop:

- a) 1 внутрішній шар з 20 нейронами;
- b) 2 внутрішніх шари по 10 нейронів у кожному;

```
# 2.Cascade-forward neural network
class CascadeForwardBackprop(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        super(CascadeForwardBackprop, self).__init__()
        self.layers = nn.ModuleList()
        in_features = input_dim

        # Cascade adding of layers
        for hidden_dim in hidden_dims:
            # Linear layer
            self.layers.append(nn.Linear(in_features, hidden_dim))
            # Activation layer
            self.layers.append(nn.ReLU())
            in_features = hidden_dim

        # Output layer
        self.layers.append(nn.Linear(in_features, output_dim))

    def forward(self, x):
        for layer in self.layers: x = layer(x)
        return x

def cf_network_test(hidden_dim, title, version):
    model = CascadeForwardBackprop(input_dim, hidden_dim,
output_dim)
    optimizer = optim.RMSprop(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()

    # Lists to store epoch and error values
    epoch_list = []
```

```

error_list = []

for epoch in range(epochs):
    optimizer.zero_grad()
    predictions = model(inputs)
    loss = criterion(predictions, targets)
    loss.backward()
    optimizer.step()

    # Calculate error for the current epoch
    if epoch % 10 == 0:
        model.eval()
        test_inputs = torch.Tensor(x_test)
        test_targets = torch.Tensor(y_test).view(-1, 1)
        relative_error_value = model_error_count(test_inputs,
test_targets, model).item()
        epoch_list.append(epoch)
        error_list.append(relative_error_value)

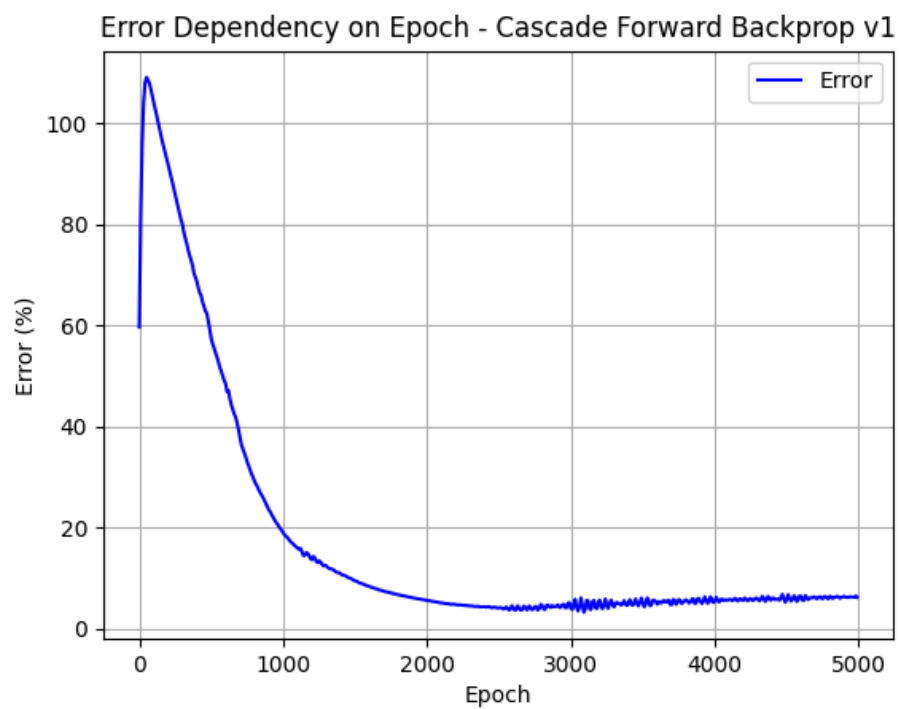
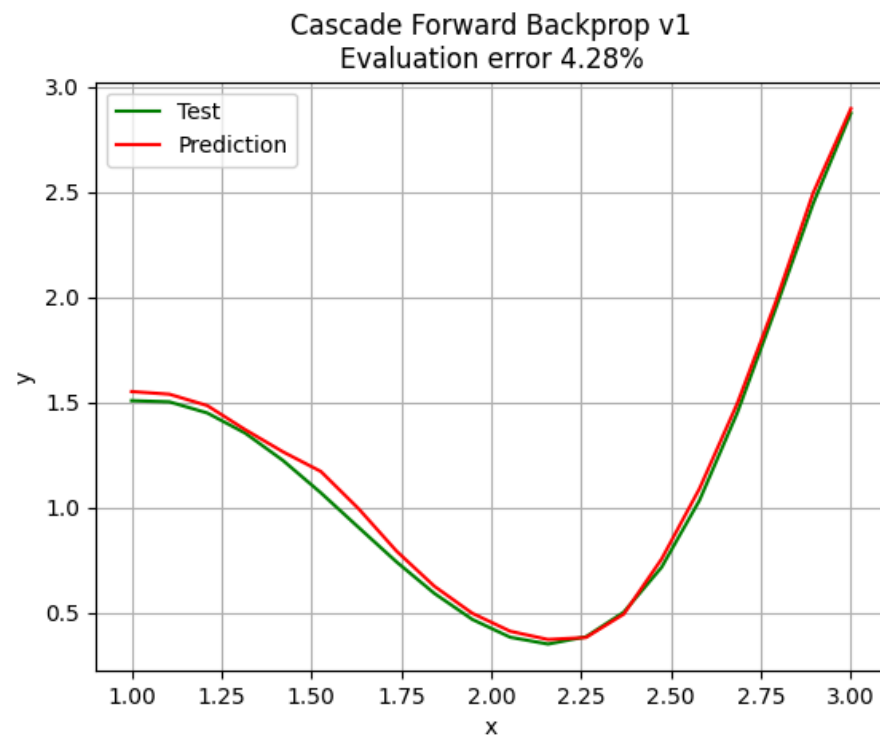
    model.eval()
    test_inputs = torch.Tensor(x_test)
    test_targets = torch.Tensor(y_test).view(-1, 1)
    relative_error_value = model_error_count(test_inputs,
test_targets, model).item()
    err = str(round(relative_error_value, 2))
    print('CF Evaluation Error ' + version + ' = ' + err + '%')

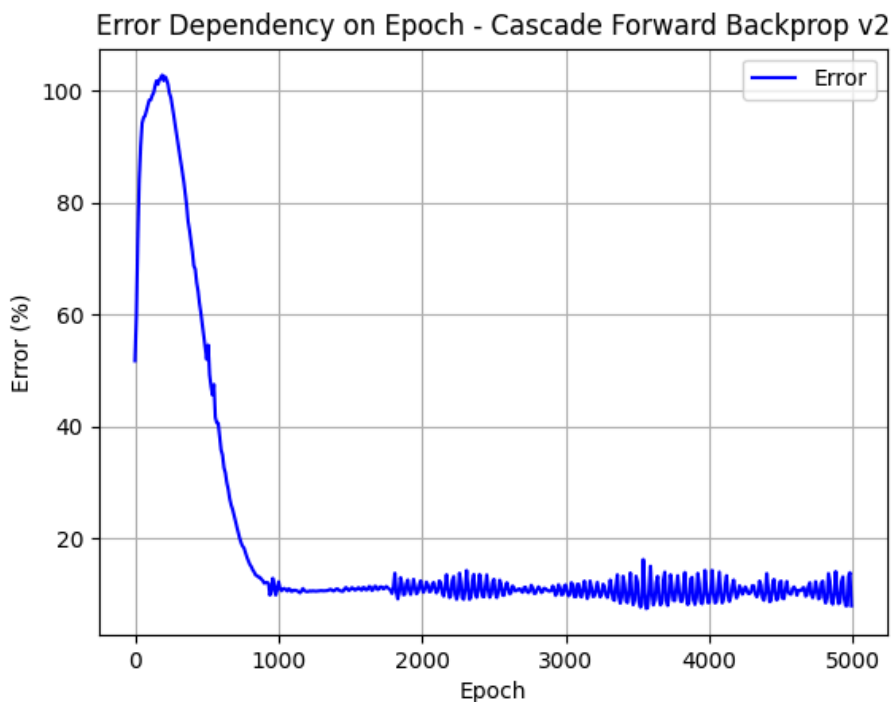
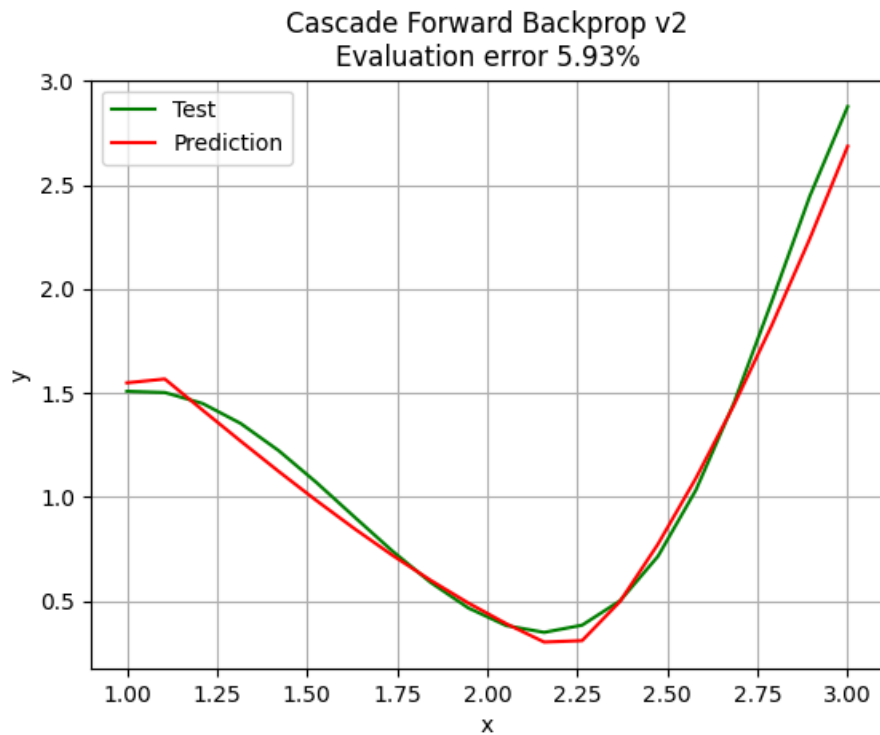
    result_graph_display(x_test[:, 0], y_test,
model(test_inputs).detach().numpy(), title, err)
    loss_graph_display(epoch_list, error_list, title)

cf_network_test([20], 'Cascade Forward Backprop v1', 'v1')
cf_network_test([10, 10], 'Cascade Forward Backprop v2', 'v2')

```

Результати:





У порівнянні із попередньою моделлю, ця мережа навчилася приблизно з таким самим успіхом. Помилка при розрахунках становить близько 4.28% в першому випадку та 5.93% в другому випадку, що тим не менш є досить непоганим результатом.

3. Мережа elman backprop:

- а) 1 внутрішній шар з 15 нейронами;
- б) 3 внутрішніх шари по 5 нейронів у кожному;

```
# 3.Elman Backprop neural network
class ElmanBackprop(nn.Module):
    def __init__(self, input_dim, hidden_dim, layers, output_dim):
        super(ElmanBackprop, self).__init__()
        self.hidden_dim = hidden_dim
        self.layers = layers
        self.elman_layers = nn.ModuleList()
        self.elman_layers.append(nn.RNN(input_dim, hidden_dim,
batch_first=True))

        # Add hidden layers
        for _ in range(layers - 1):
            self.elman_layers.append(nn.RNN(hidden_dim, hidden_dim,
batch_first=True))

        # Output layer
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        batch_size = x.size(0)
        h = [torch.zeros(1, batch_size, self.hidden_dim).to(x.device)
for _ in range(self.layers)]
        out = x

        for i in range(self.layers):
            out, h[i] = self.elman_layers[i](out, h[i])

        out = self.fc(out[:, -1, :])
        return out

def elman_network_test(layers, hidden_dim, title, version):
    model = ElmanBackprop(input_dim, hidden_dim, layers,
output_dim)
    optimizer = optim.RMSprop(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()

    # Lists to store epoch and error values
    epoch_list = []
    error_list = []
```

```

inputs = torch.Tensor(x_train).unsqueeze(1)
for epoch in range(epochs):
    optimizer.zero_grad()
    predictions = model(inputs)
    loss = criterion(predictions, targets)
    loss.backward()
    optimizer.step()

    # Calculate error for the current epoch
    if epoch % 10 == 0:
        model.eval()
        test_inputs = torch.Tensor(x_test).unsqueeze(1)
        test_targets = torch.Tensor(y_test).view(-1, 1)
        relative_error_value = model_error_count(test_inputs,
test_targets, model).item()
        epoch_list.append(epoch)
        error_list.append(relative_error_value)

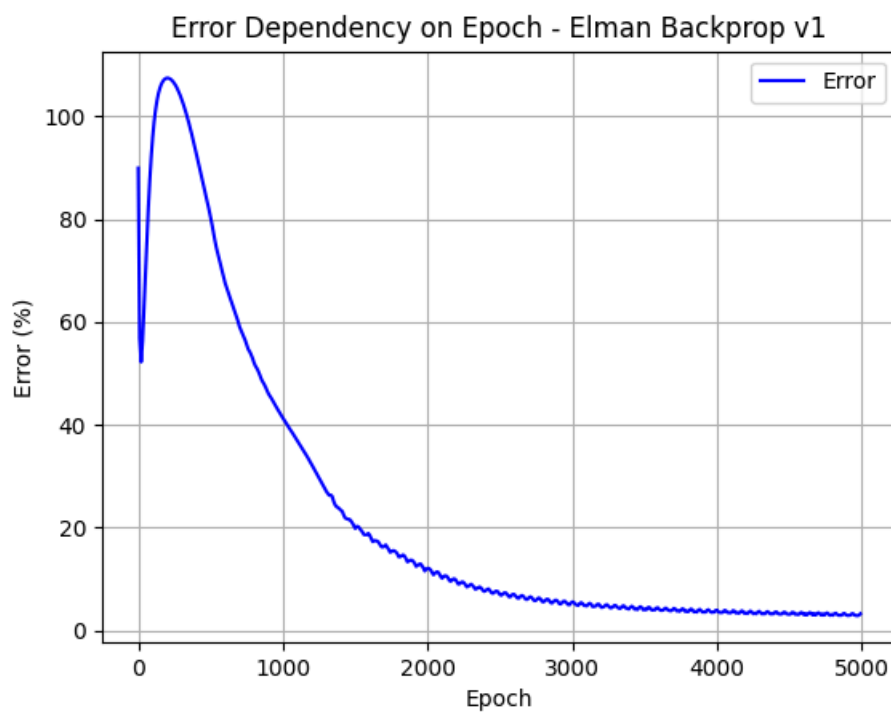
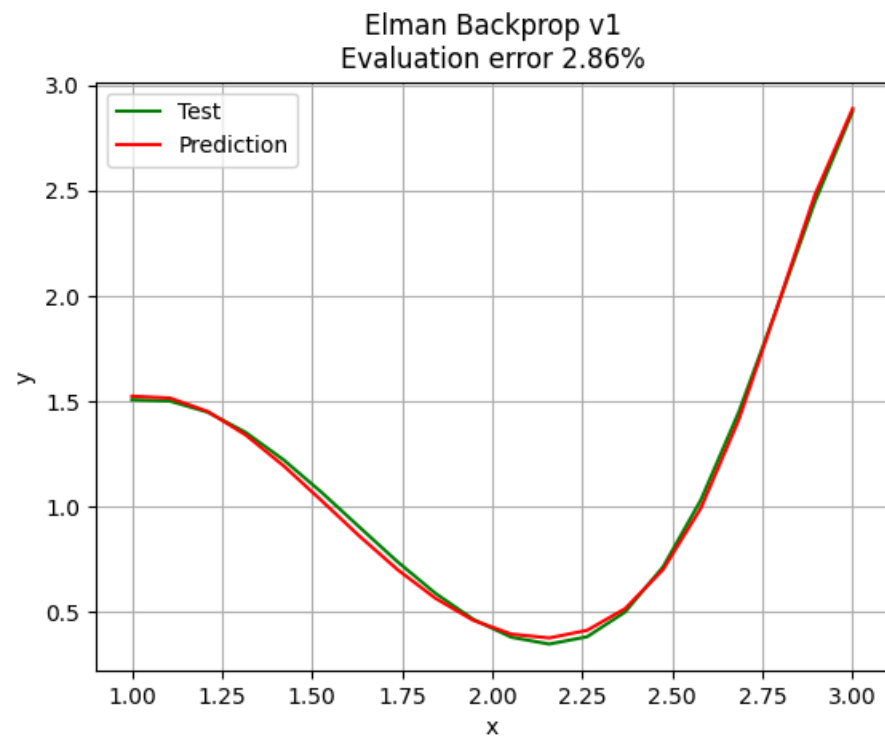
model.eval()
test_inputs = torch.Tensor(x_test).unsqueeze(1)
test_targets = torch.Tensor(y_test).view(-1, 1)
relative_error_value = model_error_count(test_inputs,
test_targets, model).item()
err = str(round(relative_error_value, 2))
print('EB Evaluation Error ' + version + ' = ' + err + '%')

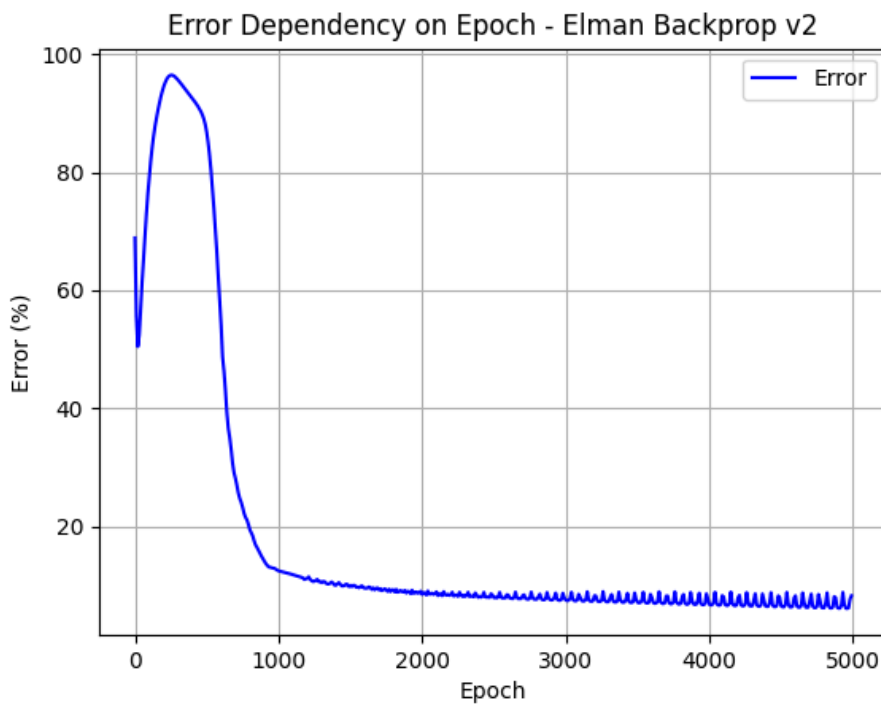
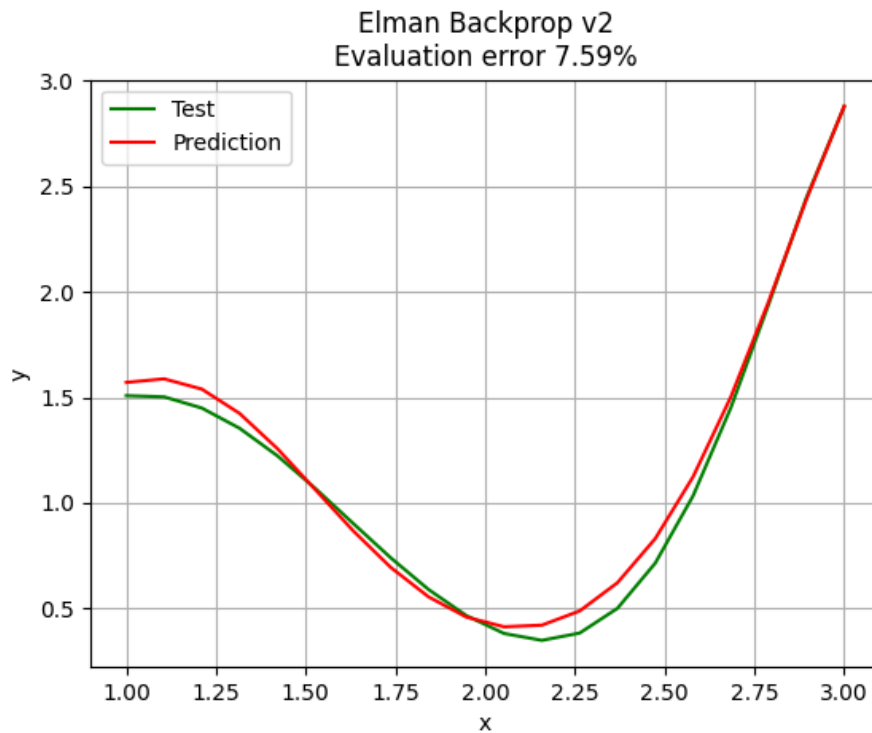
result_graph_display(x_test[:, 0], y_test,
model(test_inputs).detach().numpy(), title, err)
loss_graph_display(epoch_list, error_list, title)

elman_network_test(1, 15, 'Elman Backprop v1', 'v1')
elman_network_test(3, 5, 'Elman Backprop v2', 'v2')

```

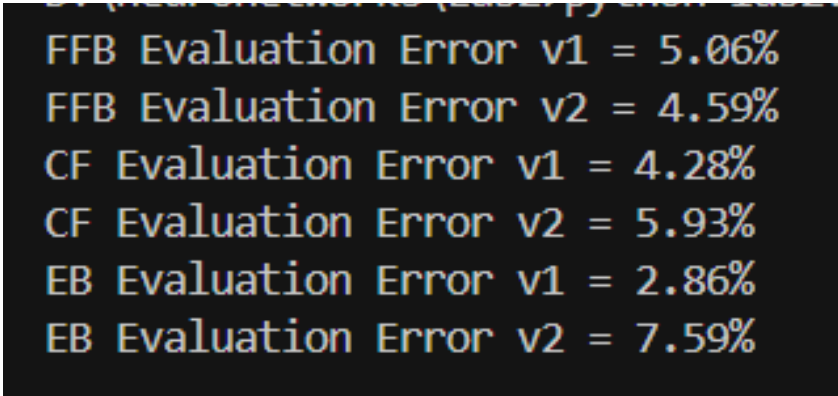
Результат:





Навчання цієї мережі було найскладнішим у реалізації, втім показало досить непоганий результат. Помилка в першому варіанті становить 2.86%, а в другому - 7.59%.

Висновок:



FFB Evaluation Error v1 = 5.06%
FFB Evaluation Error v2 = 4.59%
CF Evaluation Error v1 = 4.28%
CF Evaluation Error v2 = 5.93%
EB Evaluation Error v1 = 2.86%
EB Evaluation Error v2 = 7.59%

Під час виконання лабораторної роботи було досліджено декілька різновидів нейронних мереж з різною кількістю внутрішніх шарів та нейронів. За результатами роботи усіх трьох варіантів моделей мереж можна зробити висновок, що найкраще відпрацювала мережа Elman Backprop з 1 внутрішнім шаром з 15 нейронами. Найгірший результат отримала мережа Elman Backprop з 3 внутрішніми шарами по 5 нейронів. Інші дві мережі навчилися з приблизно однаковою успішністю.