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

Лабораторна робота №7
з дисципліни
«Технології Computer Vision»
на тему
«Дослідження технологій ідентифікації об’єктів на цифрових зображеннях для задач Computer Vision»

Виконала:
Студентка групи ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

Мета: дослідити принципи та особливості підготовки даних, синтезу, навчання та застосування штучних нейронних мереж (Artificial Neural Networks) для практичних задач ідентифікації в технологіях Computer Vision.

Завдання:

І рівень складності – максимально 8 балів.
Відповідно до технічних умов, табл.1 додатку.

7	Розробити програмний скрипт, що забезпечує ідентифікацію бінарних зображень 6 літералів, заданих матрицею растра. Для ідентифікації синтезувати, навчити та застосувати штучну нейронну мережу в «сирому» вигляді реалізації матричних операцій. Обґрунтувати вибір архітектури та алгоритму навчання нейромережі. Довести працездатність та ефективність синтезованої нейронної мережі.

Результати виконання лабораторної роботи

Синтезована математична модель:

Прямий прохід (Forward Pass)

- *Прихований шар*

Лінійне перетворення

$z_1 = x * W_1$

Активація

$a_1 = \sigma(z_1)$

$\sigma(x) = 1 / (1 + e^{-x})$

- *Вихідний шар*

Лінійне перетворення

$z_2 = a_1 * W_2$

Активація

$a_2 = \sigma(z_2)$

Результат вихідного шару (a_2) є прогнозованим значенням (ймовірностями) для кожного класу.

Функція втрат

Квадратична похибка

$$L = \frac{1}{n} * \sum_{i=1}^{n} (a_2[i] - y[i])^2$$

де y - це істинні значення (one-hot encoding)

Зворотний прохід (Backward Pass)

Градієнт для вихідного шару: $d_2 = a_2 - y$

Оновлення ваг між прихованим і вихідним шарами: $\Delta W_2 = a_1^T * d_2$

Градієнт для прихованого шару:

$d_1 = (d_2 * W_2^T) \odot \sigma'(z_1)$

$\sigma'(x) = \sigma(x) * (1 - \sigma(x))$

Оновлення ваг між вхідним і прихованим шарами: $\Delta W_1 = x^T \cdot d_1$

Оновлення ваг з урахуванням коефіцієнта навчання (α):

$W_1 \leftarrow W_1 - \alpha \cdot \Delta W_1$,

$W_2 \leftarrow W_2 - \alpha \cdot \Delta W_2$

Точність

Прогнозована категорія визначається як індекс нейрона з найбільшою активацією: predicted_label = arg max(a_2).

Точність обчислюється як:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Samples}} * 100\%$$

Результати архітектурного проектування:

Результати роботи програми:

В якості вхідних даних візьмемо 6 літералів, як і зазначено в завданні: A, B, C, D, E, F. У програмі подамо їх у вигляді матриці растра та виведемо на екран для наочності.

Вивід на екран:

Частина коду, що за це відповідає:

```
# Dataset setup
def data_x():
    # Example symbols defined as binary raster images
    symbols = {
        'A': [
            0, 1, 1, 1, 0,
```

```
            1, 0, 0, 0, 1,
            1, 1, 1, 1, 1,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 1
        ],
        'B': [
            1, 1, 1, 0, 0,
            1, 0, 0, 1, 0,
            1, 1, 1, 0, 0,
            1, 0, 0, 1, 0,
            1, 1, 1, 0, 0
        ],
        'C': [
            0, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 0,
            1, 0, 0, 0, 1,
            0, 1, 1, 1, 0
        ],
        'D': [
            1, 1, 1, 0, 0,
            1, 0, 0, 1, 0,
            1, 0, 0, 0, 1,
            1, 0, 0, 1, 0,
            1, 1, 1, 0, 0
        ],
        'E': [
            1, 1, 1, 1, 0,
            1, 0, 0, 0, 0,
            1, 1, 1, 0, 0,
            1, 0, 0, 0, 0,
            1, 1, 1, 1, 0
        ],
        'F': [
            1, 1, 1, 1, 0,
            1, 0, 0, 0, 0,
            1, 1, 1, 0, 0,
            1, 0, 0, 0, 0,
            1, 0, 0, 0, 0
        ]
    }

    x = [np.array(data).reshape(1, 25) for data in symbols.values()]

    plt.figure(figsize=(10, 2))
    for i, (key, data) in enumerate(symbols.items()):
        plt.subplot(1, 6, i + 1)
        plt.imshow(1 - np.array(data).reshape(5, 5), cmap='gray')  # Inverted
colors
        plt.title(key)
        plt.axis('off')
    plt.show()

    return x
```

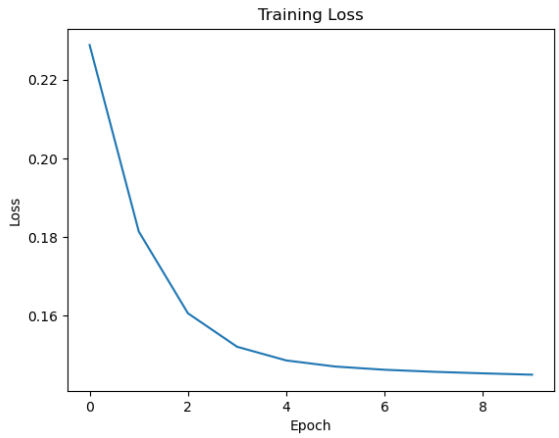
Після цього створимо нейромережу, де на самому початку вагові коефіцієнти будуть визначатись випадковим чином, а впродовж декількох епох будуть коригуватись.

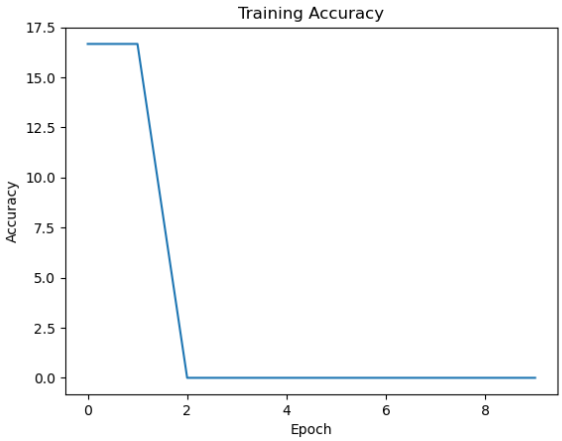
Для початку виконаємо тренування нейромережі на невеликій кількості епох – наприклад, 10. Бачимо такий результат роботи програми:

```
Evaluating the network...
Expected    Predicted   Correct
A           F           False
B           F           False
C           E           False
D           F           False
E           F           False
F           F           True
```

Бачимо, що лише один літерал був ідентифікований правильно.

Тепер подивимось на графік втрат та точності. Точність - відсоток правильних відповідей з усіх, тому враховуючи зовсім невеликий датасет – 6 літер, цей графік є більш різким і має чіткі кути. При збільшенні даних вони мали б згладитись.





Із результатів бачимо, що модель тільки почала виходити на більш-менш стабільні втрати, але це не є достатнім для правильного визначення літералів, бо з другого графіка чітко видно, що частка правильних відповідей не зростає.

Спробуємо взяти 50 епох і знов запустити програму:

```
Evaluating the network...
Expected    Predicted   Correct
A           A           True
B           D           False
C           C           True
D           D           True
E           E           True
F           F           True
```

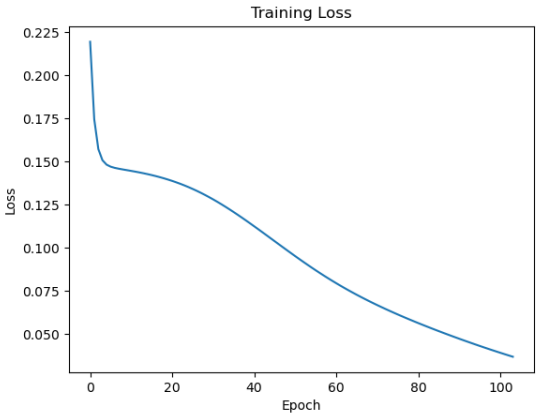
Бачимо, що кількість правильних відповідей збільшилась, але модель все ще недостатньо точна.

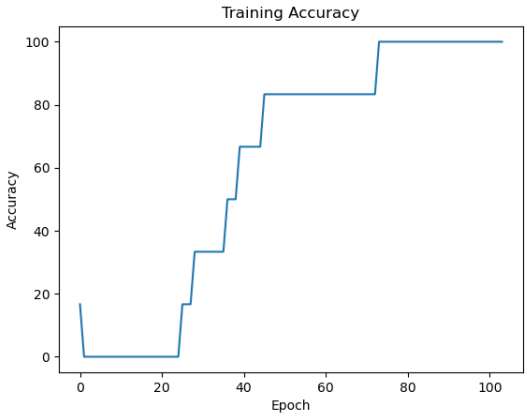
Тепер спробуємо обрати дуже велику кількість епох, наприклад, 400. При цьому поставимо параметр patience=30. Це означатиме, що якщо протягом 30 епох відсоток правильних відповідей залишатиметься тим самим, то модель вже достатньо навчилась.

```
Early stopping at epoch 104
Evaluating the network...
Expected    Predicted   Correct
A           A           True
B           B           True
C           C           True
D           D           True
E           E           True
F           F           True
```

Бачимо, що завдяки patience=30 відбувся завчасний вихід із тренування моделі на 104 епосі. При цьому всі літери були ідентифіковані правильно.

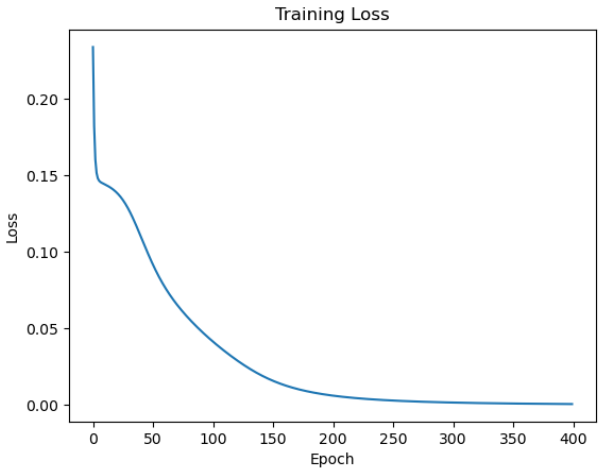
Графіки цього разу виглядають так:





Бачимо дещо дивний вигин на графіку втрат, але враховуючи те, що ми ідентифікуємо всього 6 літер і тренувальних даних небагато, то така трохи неконсистентна поведінка буде вважатись нормою.

Спробуємо виставити дуже велику patience, щоб подивитись на графік втрат саме при 400 епохах:



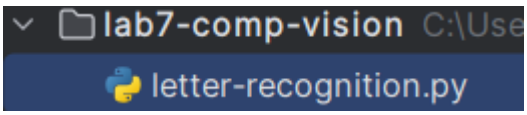

```
Evaluating the network...
Expected    Predicted   Correct
A           A           True
B           B           True
C           C           True
D           D           True
E           E           True
F           F           True
```

Бачимо, що втрати стабілізувались і вийшли на один постійний рівень близький до нуля. Такого рівня втрати почали досягати десь близько із 250 епохи. Отже, незважаючи на те, що кількість правильних відповідей досягла максимума ще на ~104 епохах, втрати стали стабільно малими лише після ~250 епохи, і це видно на графіку втрат для 400 епох.

Отже, оптимальнішою кількістю епох буде ~250 і більше. При такому невеликому датасеті графік втрат є більш надійним ідентифікатором ефективності навчання, ніж частка правильних відповідей.

Опис структури проекту програми:

В проєкті є лише один файл, що повністю забезпечує виконання поставленого завдання



Код програми:

```
import numpy as np
import matplotlib.pyplot as plt

# Dataset setup
def data_x():
    # Example symbols defined as binary raster images
    symbols = {
        'A': [
            0, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            1, 1, 1, 1, 1,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 1
        ],
        'B': [
            1, 1, 1, 0, 0,
            1, 0, 0, 1, 0,
            1, 1, 1, 0, 0,
            1, 0, 0, 1, 0,
            1, 1, 1, 0, 0
        ],
        'C': [
            0, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            1, 0, 0, 0, 0,
            1, 0, 0, 0, 1,
            0, 1, 1, 1, 0
        ],
        'D': [
            1, 1, 1, 0, 0,
            1, 0, 0, 1, 0,
            1, 0, 0, 0, 1,
            1, 0, 0, 1, 0,
            1, 1, 1, 0, 0
        ],
        'E': [
            1, 1, 1, 1, 0,
            1, 0, 0, 0, 0,
            1, 1, 1, 0, 0,
            1, 0, 0, 0, 0,
            1, 1, 1, 1, 0
        ],
        'F': [
            1, 1, 1, 1, 0,
            1, 0, 0, 0, 0,
            1, 1, 1, 0, 0,
            1, 0, 0, 0, 0,
            1, 0, 0, 0, 0
        ]
```

```
    }

    x = [np.array(data).reshape(1, 25) for data in symbols.values()]

    plt.figure(figsize=(10, 2))
    for i, (key, data) in enumerate(symbols.items()):
        plt.subplot(1, 6, i + 1)
        plt.imshow(1 - np.array(data).reshape(5, 5), cmap='gray')  # Inverted
colors
        plt.title(key)
        plt.axis('off')
    plt.show()

    return x

# Labels
def data_y():
    y = np.eye(6)  # One-hot encoding for 6 classes
    return y

# Weight initialization
def generate_weights(input_size, output_size):
    return np.random.randn(input_size, output_size) * 0.1

# Sigmoid activation
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of sigmoid
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Forward pass
def forward(x, w1, w2):
    z1 = x.dot(w1)
    a1 = sigmoid(z1)
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)
    return z1, a1, z2, a2

# Backward pass
def backprop(x, y, z1, a1, z2, a2, w1, w2, alpha):
    d2 = a2 - y
    d1 = np.multiply(d2.dot(w2.T), sigmoid_derivative(z1))

    w2_update = a1.T.dot(d2)
    w1_update = x.T.dot(d1)

    w2 -= alpha * w2_update
    w1 -= alpha * w1_update

    return w1, w2

# Loss function
def loss(a2, y):
    return np.mean(np.square(a2 - y))

# Training the network
def train(x, y, w1, w2, alpha, epochs, patience):
```

```
    accuracy_history = []
    loss_history = []
    best_accuracy = 0
    patience_counter = 0

    for epoch in range(epochs):
        epoch_loss = 0
        correct_predictions = 0
        for i in range(len(x)):
            z1, a1, z2, a2 = forward(x[i], w1, w2)
            epoch_loss += loss(a2, y[i])
            w1, w2 = backprop(x[i], y[i], z1, a1, z2, a2, w1, w2, alpha)
            if np.argmax(a2) == np.argmax(y[i]):
                correct_predictions += 1

        loss_history.append(epoch_loss / len(x))
        accuracy = correct_predictions / len(x) * 100
        accuracy_history.append(accuracy)

        # Early Stopping Logic
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"Early stopping at epoch {epoch + 1}")
                break

        if (epoch + 1) % 10 == 0:
            print(f"Epoch {epoch + 1}/{epochs}, Loss: {epoch_loss /
len(x):.4f}")
    return w1, w2, loss_history, accuracy_history

# Prediction
def predict(x, w1, w2):
    _, _, _, a2 = forward(x, w1, w2)
    return np.argmax(a2)

# Visualizing results
def evaluate(x, y, w1, w2):
    labels = ['A', 'B', 'C', 'D', 'E', 'F']
    print("Expected\tPredicted\tCorrect")
    for i in range(len(x)):
        expected = labels[np.argmax(y[i])]
        predicted = labels[predict(x[i], w1, w2)]
        correct = expected == predicted
        print(f"{expected}\t\t\t{predicted}\t\t\t{correct}")

# Main execution
if __name__ == "__main__":
    x = data_x()
    y = data_y()

    print("Dataset loaded.")

    w1 = generate_weights(25, 10)  # Hidden layer with 10 neurons
    w2 = generate_weights(10, 6)   # Output layer with 6 neurons
```

```
    print("Training the network...")
    w1, w2, loss_history, accuracy_history = train(x, y, w1, w2, alpha=0.1,
epochs=400, patience=1000)

    plt.plot(loss_history)
    plt.title("Training Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

    plt.plot(accuracy_history)
    plt.title("Training Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.show()

    print("Evaluating the network...")
    evaluate(x, y, w1, w2)
```

Висновки: виконавши лабораторну роботу №7, я ознайомилась із особливостями підготовки даних, синтезу, навчання та застосування простих штучних нейронних мереж. В рамках лабораторної роботи було натреновано нейронну мережу, що вміє розрізняти 6 літералів – A, B, C, D, E, F.