Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №2

з дисципліни «Методології розробки інтелектуальних комп'ютерних програм»

«**Розпізнавання образів за допомогою штучних нейронних мереж**»

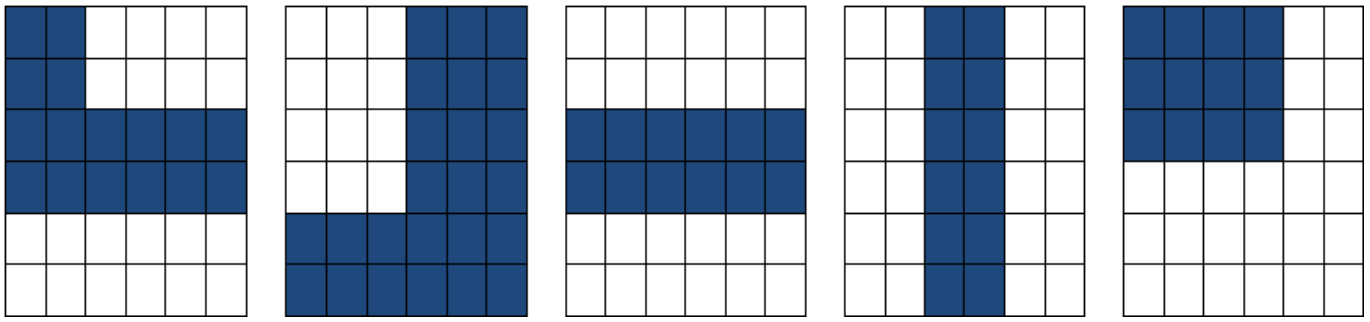
Виконала: студентка 3 курсу
групи ТІ-01
Круть Катерина

Перевірив: д.т.н. Мусієнко А. П.

КИЇВ 2023

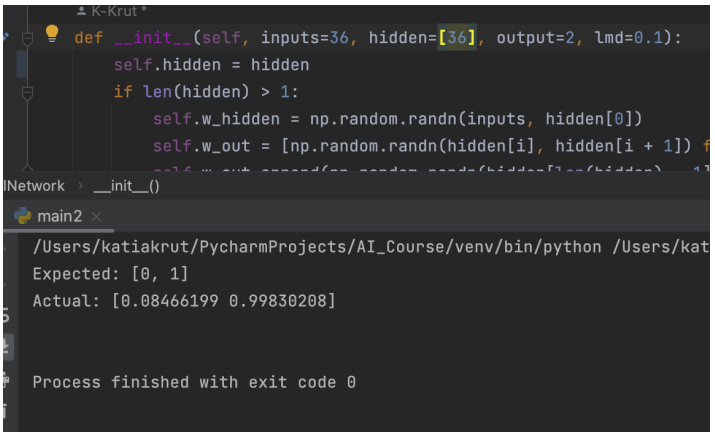
Мета дослідження: Ознайомитися з методами розпізнавання образів за допомогою штучних нейронних мереж, побудувати, навчити та протестувати нейронну мережу для розпізнавання букв.

Варіант 12

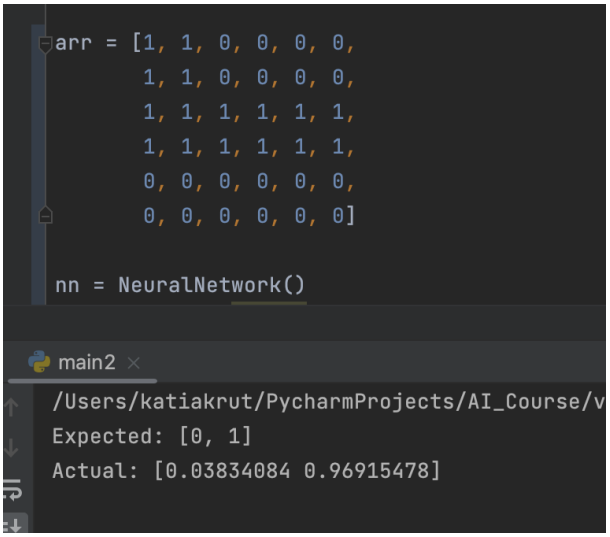


ХІД РОБОТИ

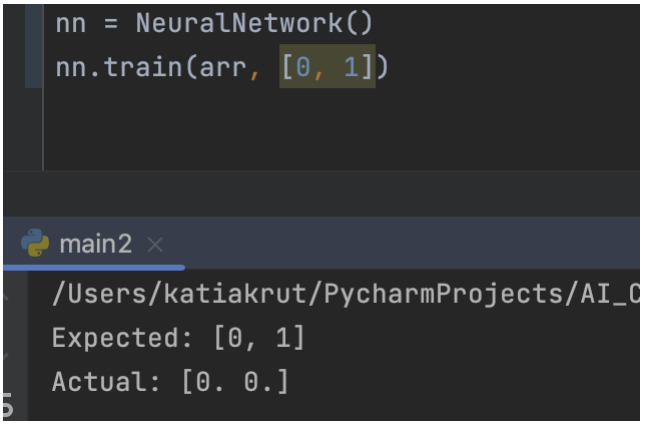
1. Один прихований шар з 36 нейронівз функцією активації SIGMOID, 100 епох:



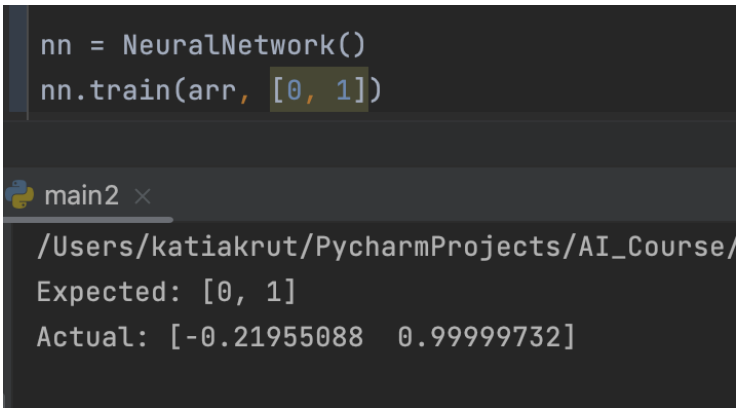
2. Два прихованих шари кожен з який складається з 36 нейронів та активатором SIGMOID, 100 епох:



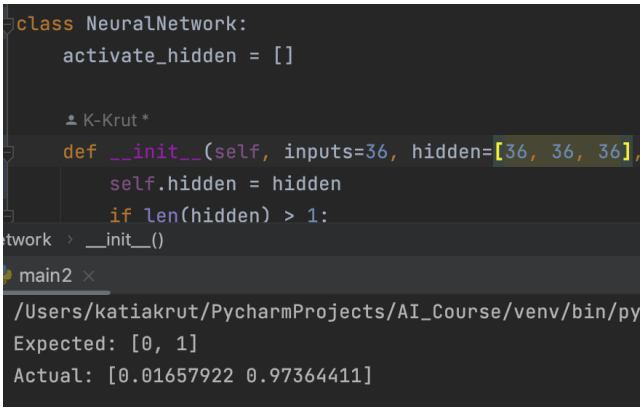
5. Два прихованих шари кожен з який складається з 36 нейронів та активатором RELU, 100 епох:



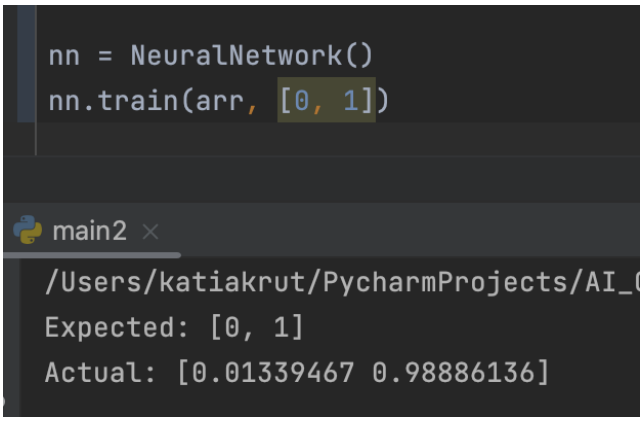
6. Два прихованих шари кожен з який складається з 36 нейронів та активатором TANH, 100 епох:



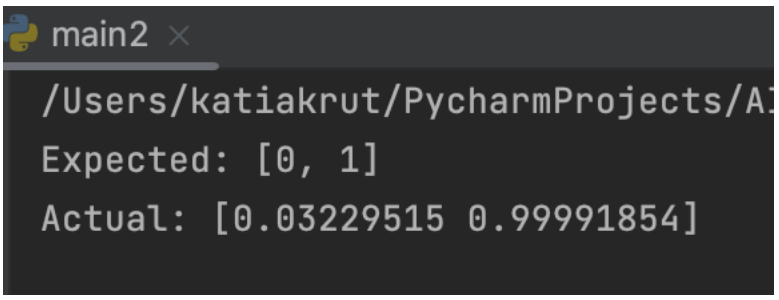
7. Три прихованих шари з 36 нейронів та активаторами SIGMOID, 100 епох:



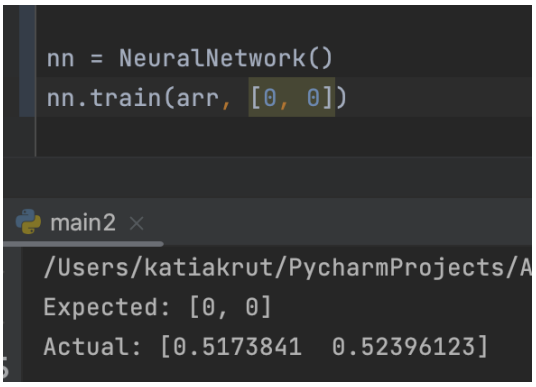
8. Три прихованих шари з 72 нейронів та активатором SIGMOID, 1000 епох:



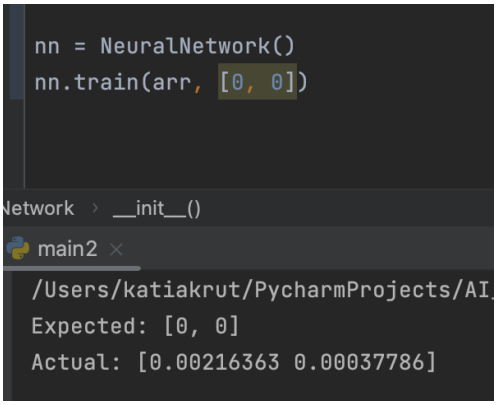
9. Три прихованих шари з 72 нейронів та активатором SIGMOID, 100 епох:



15. Три прихованих шари (36, 36, 36) та активатором TANH, 1000 епох:



17. Два прихованих шари з 72 нейронів з активатором SIGMOID, 5000 епох:



КОПІЯ ВИКОНАНОЇ ПРОГРАМИ

```
import numpy as np


def f(x):
    return 1 / (1 + np.exp(-x))


def df(x):
    return x * (1 - x)


# def f(x):
#     return np.maximum([0.0], x)
#
#
# def df(x):
#     return np.where(x > 0, 1, 0)
#
# def relu(x):
#     return np.maximum([0.0], x)
#
#
# def d_relu(x):
#     return np.where(x > 0, 1, 0)
#
#
# def activate_tanh(x):
#     return np.tanh(x)
#
#
# def deactivate_tanh(x):
#     return 1 - np.tanh(x) ** 2


# def f(x):
#     return np.tanh(x)
#
#
# def df(x):
#     return 1 - np.tanh(x) ** 2


class NeuralNetwork:
    activate_hidden = []

    def __init__(self, inputs=36, hidden=[72, 72], output=2, lmd=0.1):
        self.hidden = hidden
        if len(hidden) > 1:
            self.w_hidden = np.random.randn(inputs, hidden[0])
            self.w_out = [np.random.randn(hidden[i], hidden[i + 1]) for i in
range(len(hidden) - 1)]
            self.w_out.append(np.random.randn(hidden[len(hidden) - 1], output))
        else:
            self.w_hidden = np.random.randn(inputs, hidden[0])
            self.w_out = np.random.randn(hidden[0], output)
        self.out = None
        self.lmd = lmd

    def forward_propagation(self, inputs):
        s_hidden = np.dot(inputs, self.w_hidden)
        if len(self.hidden) > 1:
            self.activate_hidden = []
            self.activate_hidden.append(f(s_hidden))
```

```
            for value in range(0, len(self.w_out)):
                s_hidden = np.dot(self.activate_hidden[-1], self.w_out[value])
                self.activate_hidden.append(f(s_hidden))
            self.out = self.activate_hidden[-1]
        else:
            self.activate_hidden = f(np.dot(inputs, self.w_hidden))
            self.out = f(np.dot(self.activate_hidden, self.w_out))

    def back_propagation(self, inputs, delta):
        if len(self.hidden) == 1:
            d_hidden = np.dot(delta, self.w_out.T) * f(self.activate_hidden)
            wd_after = np.outer(self.activate_hidden, delta) * self.lmd
            wd_before = np.outer(inputs, d_hidden) * self.lmd
            return wd_before, wd_after
        elif len(self.hidden) > 1:
            d_temp = np.dot(delta, self.w_out[-1].T)
            wd_after = np.outer(self.activate_hidden[-2], delta) * self.lmd
            wd_hidden = []
            d_hidden = []
            for i in range(1, len(self.hidden) + 1):
                d_hidden = d_temp * f(self.activate_hidden[-i - 1])
                if i < len(self.hidden):
                    d_temp = np.dot(d_hidden, self.w_out[-i - 1].T)
                    wd_hidden[:0] = np.array([np.outer(self.activate_hidden[-i - 2],
d_hidden) * self.lmd])
            wd_hidden.append(wd_after)
            wd_before = np.outer(inputs, d_hidden) * self.lmd
            return wd_before, wd_hidden

    def train(self, inputs, expected):
        for epoch in range(5000):
            self.forward_propagation(inputs)
            error = expected - self.out
            delta = error * df(self.out)

            w_hidden, w_out = self.back_propagation(inputs, delta)

            self.w_hidden += w_hidden
            for weight, weight_delta in zip(self.w_out, w_out):
                weight += weight_delta

        print(f"Expected: {expected}\nActual: {self.out}\n")


arr = [1, 1, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0]

nn = NeuralNetwork()
nn.train(arr, [0, 0])
```

Висновки: Під час виконання лабораторної роботи було набуто навичок роботи з штучними нейроними мережами, призначених для розпізнавання образів. Для вирішення задачі було використано алгоритм зворотного поширення похибки. В результаті було розроблено нейронну мережу, яка з указаними умовами здатна розпізнавати образи, які частково відрізняються від початкових. Також було проведено порівняння роботи нейронної мережі залежно від різних функцій активацій, кількості прихованих шарів, кількості нейронів у прихованих шарах тощо.