МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

З В І Т

Лабораторна робота №8
з дисципліни
«Компьютерні системи штучного інтелекту»

Виконавець:

студент групи КІ-22м Косей М.П.

Керівник:

викладач Саяпін В.Г.

2023

Лабораторна робота №8

Тема: Програмування моделі штучних нейромереж

Мета: Одержати основні навички побудови моделей штучних нейромереж

ХІД РОБОТИ

1) Ознайомитись з теоретичними відомостями до лабораторної роботи

Штучні нейронні мережі вирішують ряд різноманітних задач з різних галузей науки, техніки та технології.

Деякі з основних задач, які можуть бути вирішені за допомогою нейронних мереж:

1. Класифікація: Визначення приналежності вхідних даних до однієї з попередньо визначених категорій або класів. Наприклад, класифікація електронного листа на спам та не-спам, розпізнавання об'єктів на зображеннях, визначення медичних діагнозів тощо.
2. Регресія: Прогнозування числових значень на основі вхідних даних. Наприклад, прогнозування ціни на нерухомість на основі ряду факторів, прогнозування погоди на основі метеорологічних даних, прогнозування вартості акцій тощо.
3. Генерація: Створення нових даних на основі вхідних даних. Наприклад, генерація музики, зображень, текстів тощо.
4. Кластеризація: Групування вхідних даних на основі схожості між ними. Наприклад, кластеризація споживачів за їх покупками, кластеризація зірок на зображенні, кластеризація новинних статей за темами тощо.
5. Сегментація: Розбиття зображень на окремі сегменти або області з метою подальшого аналізу. Наприклад, сегментація зображень медичних знімків для виявлення патологій, сегментація зображень автомобільних доріг на карті, сегментація об'єктів на зображеннях відеоспостереження тощо.
6. Оптичне розпізнавання символів: Розпізнавання тексту на зображеннях або в інших формах вхідних даних. Наприклад, розпізнавання номерних знаків автомобілів, розпізнавання рукописного письма, розпізнавання тексту на медичних знімках тощо.
7. Аналіз відео: Обробка та аналіз відеоданих. Наприклад, виявлення руху на відео, визначення об'єктів на відео, аналіз відтоку покупців на основі відеоспостереження в магазині тощо.
8. Рекомендації: Надання рекомендацій користувачам на основі їх взаємодії з системою або на основі аналізу їхніх вхідних даних. Наприклад, рекомендації товарів в онлайн-магазинах, рекомендації фільмів або музики на основі вподобань користувача, рекомендації лікування на основі медичних даних тощо.
9. Оптимізація: Пошук оптимальних рішень на основі вхідних даних

та визначених критеріїв. Наприклад, оптимізація логістичних маршрутів, оптимізація процесів виробництва, оптимізація рекламних кампаній тощо.

Нейронні мережі можуть використовуватись для моделювання логічних функцій, таких як логічні вирази або булеві функції.

Моделювання логічних функцій нейронними мережами на основі таблиці істинності відноситься до задач класифікації.

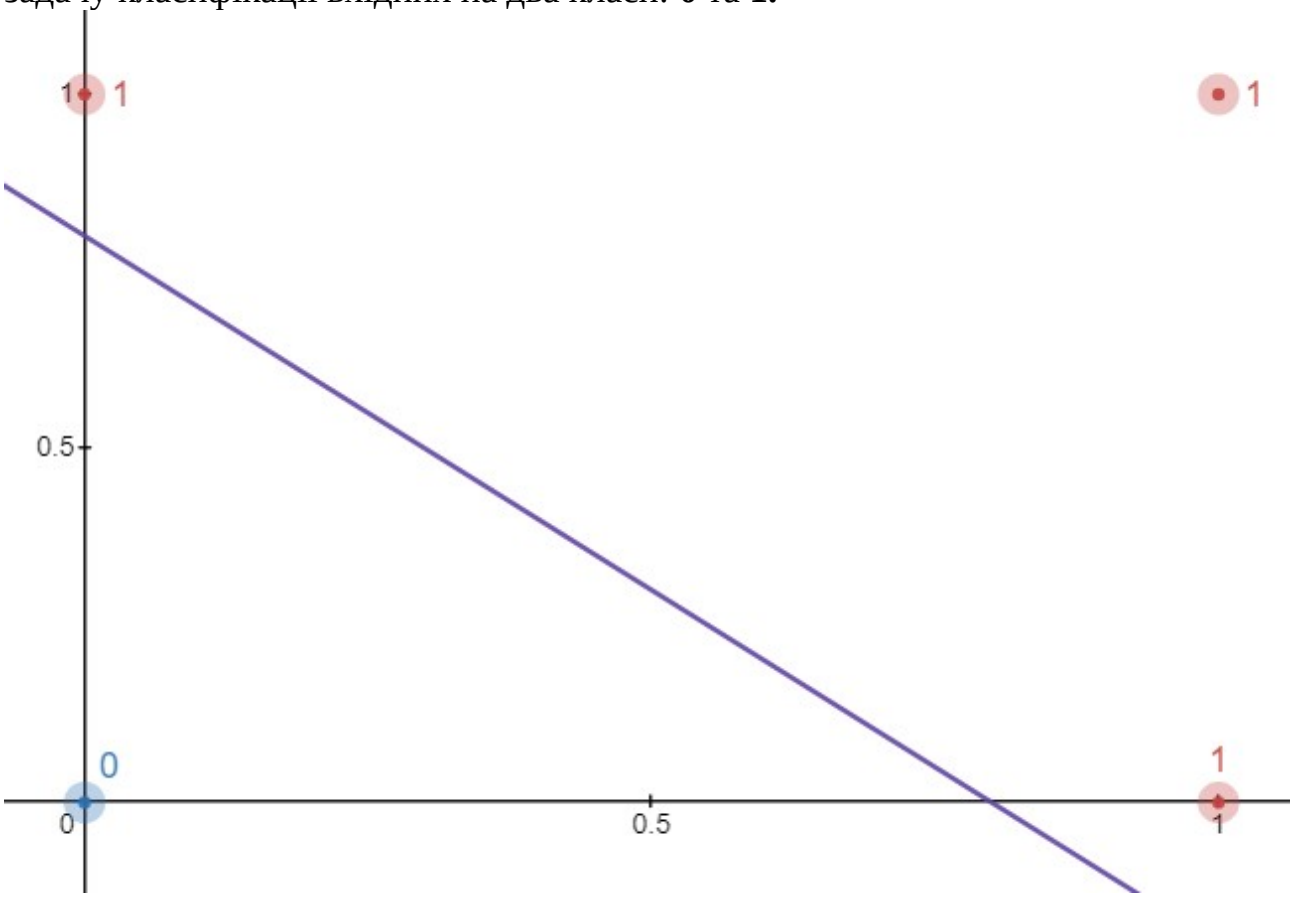
Нейронна мережа навчається класифікувати різні комбінації вхідних значень таблиці істинності на відповідні вихідні значення таблиці істинності.

Розглянемо моделювання функції **“АБО” (OR ∧) , “І” (AND ∨), “ВИКЛЮЧНЕ АБО” (XOR ⊕),** з використання одного нейрона.

Таблиця істинності для логічної функції **"АБО" (OR),** з двома аргументами виглядає наступним чином:

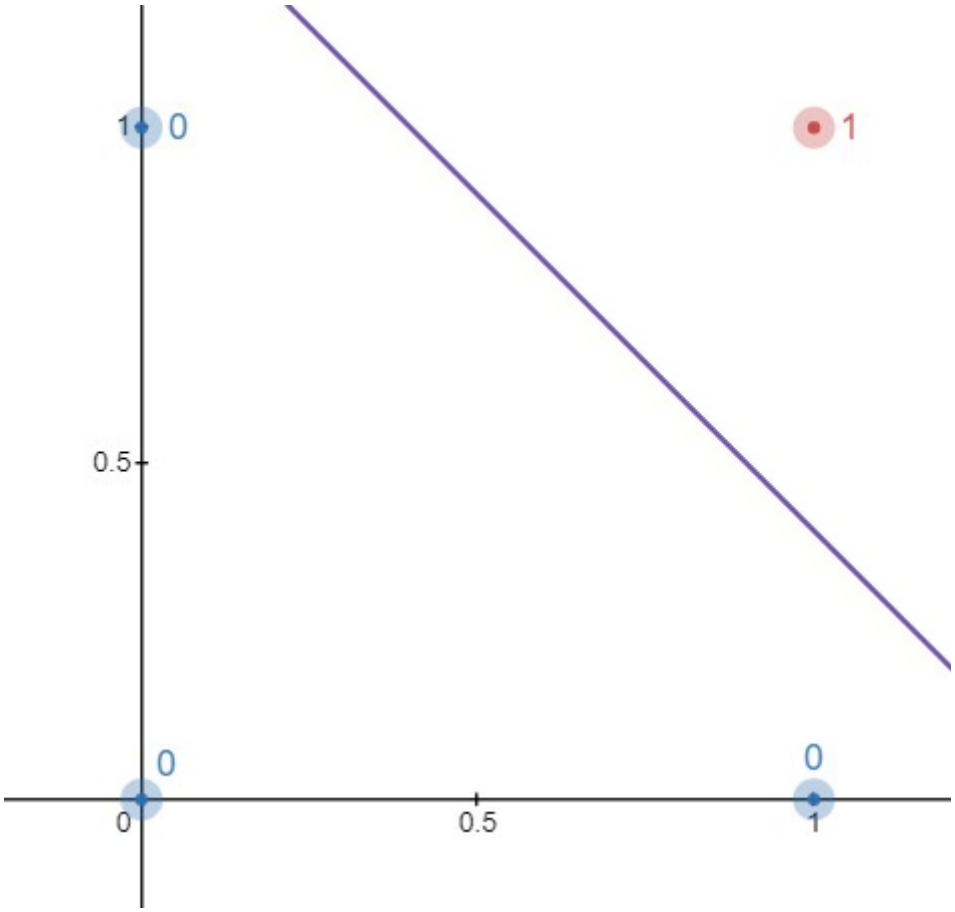
A	**B**	**АБО (OR ∧)**
0	0	0
0	1	1
1	0	1
1	1	1

Нейрон як лінійний елемент за допомогою лінії розділення вирішує задачу класифікації вхідних на два класи: **0** та **1.**



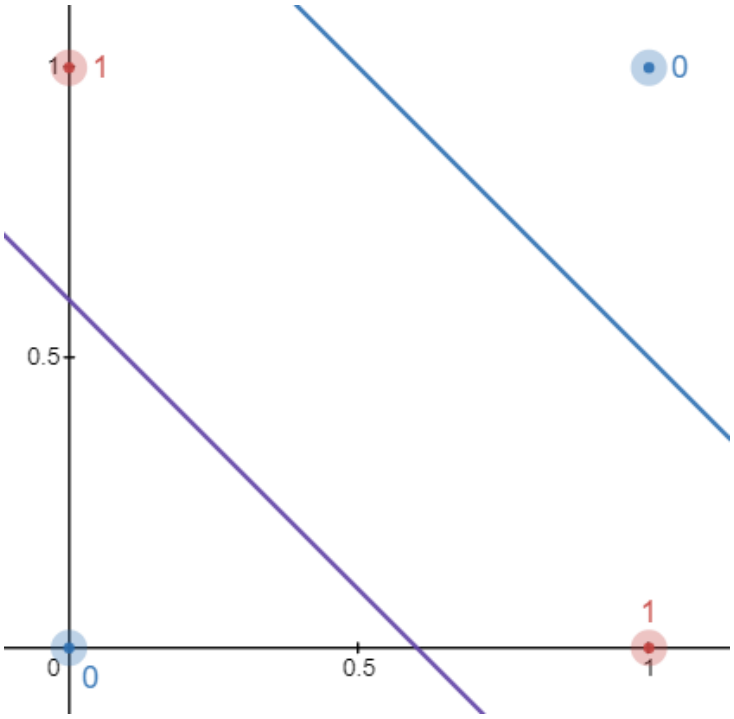
Аналогічно для функції **“I” (AND):**

A	**B**	**I (AND ∨)**
0	0	0
0	1	0
1	0	0
1	1	1

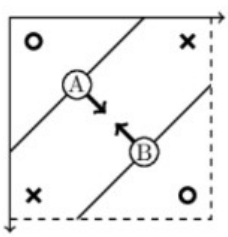


Аналогічно для функції **“ВИКЛЮЧНЕ АБО” (XOR),**

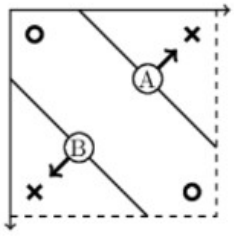
A	**B**	**ВИКЛЮЧНЕ АБО (XOR ⊕)**
0	0	0
0	1	1
1	0	1
1	1	0



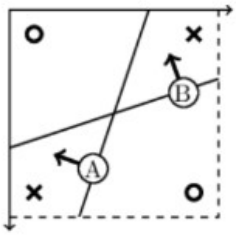
Як видно за допомогою одного нейрона неможливо реалізувати функцію **“ВИКЛЮЧНЕ АБО” (XOR ⊕),** але можливо реалізувати за допомогою більшої кількості нейронів чотирма реалізаціями:



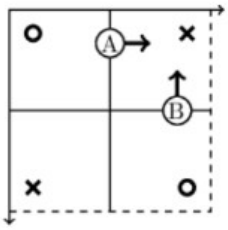
$A \,\&\, B$



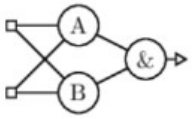
$A \vee B$

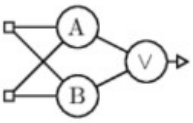


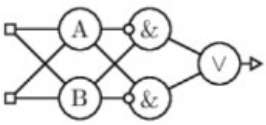
$(A\&\bar{B}) \vee (\bar{A}\&B)$

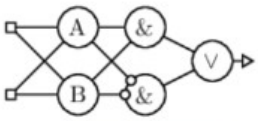


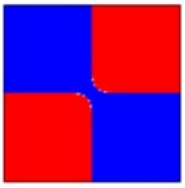
$(A\&B) \vee (\bar{A}\&\bar{B})$











Кожну логічну функцію можна привести до канонічного вигляду, використовуючи логічні закони та алгоритми спрощення виразів.

Канонічний вигляд логічної функції - це такий вираз, в якому використовуються лише базові операції кон'юнкції, диз'юнкції та заперечення, і вираз містить мінімальну кількість операторів та операцій.

Враховуючи раніше згадане, можна зробити висновок, що нейронна мережа здатна змоделювати будь-яку логічну функцію, якщо вона має достатню кількість прихованих шарів та нейронів у цих шарах.

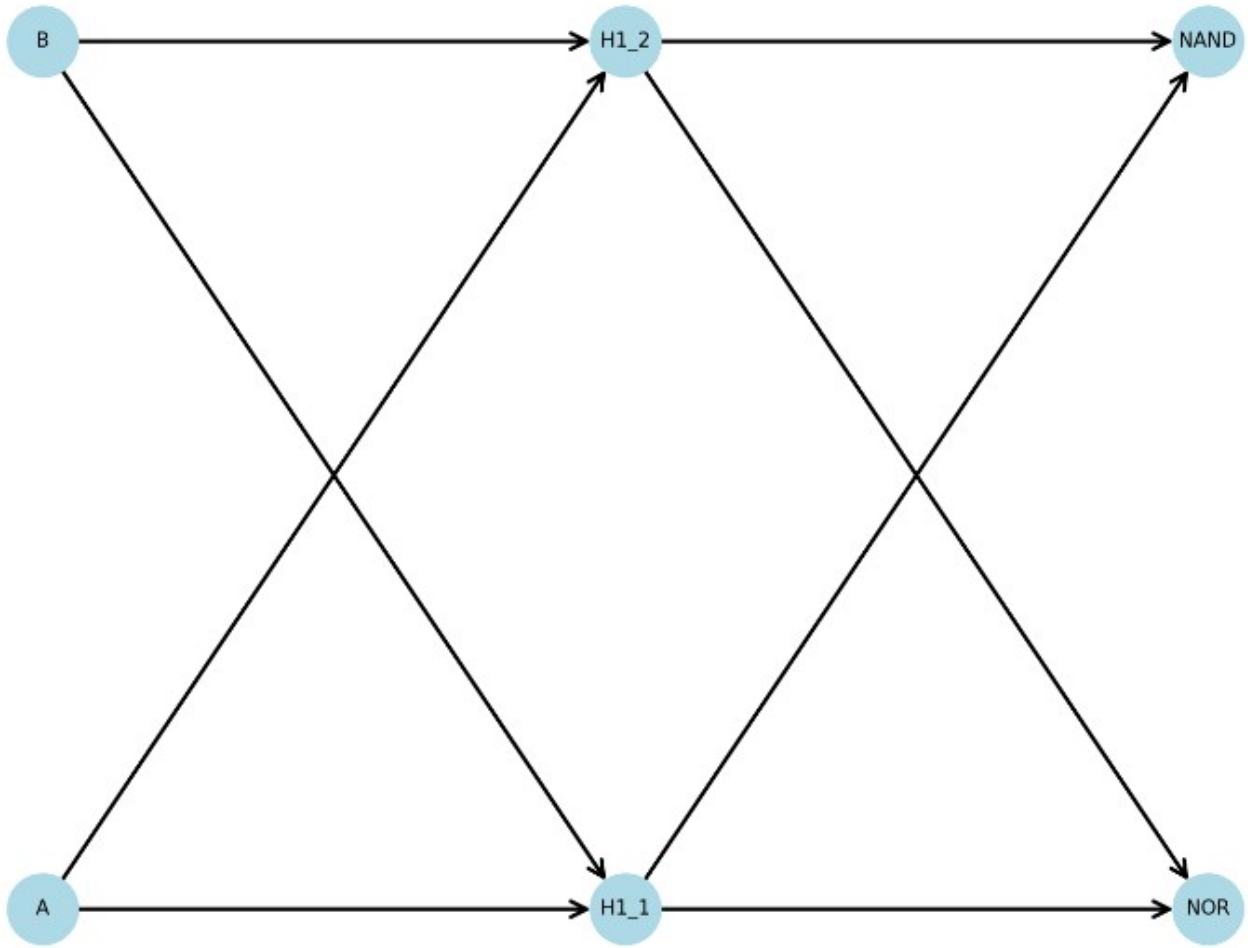
2) Побудувати нейрону мережу для моделювання логічної функції

Згідно з варіантом 8 необхідно запрограмувати штучну нейрону мережу з двома виходами, щоб отримати значення:

- першої функції **стрілка Пірса (NOR ↓)** на першому виході;
- другої функції **штрих Шеффера (NAND ↑)** на другому виході.

Структура мережі буде мати таку структуру:

1. Вхідний шар с 2-ма нейронами, кількість нейронів дорівнює кількості вхідних змінних;
2. Прихований шар с 2-ма нейронами, кожний нейрон буде обчислювати свою функцію **NOR** або **NAND;**
3. Вихідний шар с 2-ма нейронами, кількість нейронів дорівнює кількості вихідних змінних.



```python
import networkx as nx
import matplotlib.pyplot as plt

# Створюємо граф
G = nx.DiGraph()

# Додаємо вузли
input_nodes = ['A', 'B']
hidden_nodes_layer = []
output_nodes = ['NOR', 'NAND']

for i in range(2):
    hidden_nodes_layer.append(f'H1_{i + 1}')
    G.add_node(f'H1_{i+1}')

# Додаємо зв'язки між вузлами
for input_node in input_nodes:
    for hidden_node in hidden_nodes_layer:
        G.add_edge(input_node, hidden_node)

for hidden_node in hidden_nodes_layer:
    for output_node in output_nodes:
        G.add_edge(hidden_node, output_node)

# Визначаємо позиції вузлів для відображення на графіку
pos = {}
pos.update((node, (0, i*100)) for i, node in enumerate(input_nodes))  # Позиції вхідних вузлів
pos.update((node, (25, i*100)) for i, node in enumerate(hidden_nodes_layer))  # Позиції вузлів прихованого шару
pos.update((node, (50, i*100)) for i, node in enumerate(output_nodes))  # Позиції вихідних вузлів

# Відображаємо граф
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=300)
nx.draw_networkx_edges(G, pos, arrows=True, arrowstyle='->', arrowsize=10)
nx.draw_networkx_labels(G, pos, font_size=5, font_color='black', verticalalignment='center')

# Зберігаємо графік
plt.savefig('neuron_network.png', dpi=300)
```

Таблиця істинності для логічної функції.

Вхідні змінні		**Вихідні змінні**	
A	**B**	**NOR ↓**	**NAND ↑**
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

Використовуємо IDE **PyCharm** та мову програмування **Python**


```python
# Імпортуємо необхідні бібліотеки
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense

# Вхідні дані A,B
a_b_train = [[0, 0], [0, 1], [1, 0], [1, 1]]
# Вихідні дані NOR, NAND
nor_nand_train = [[1, 1], [0, 1], [0, 1], [0, 0]]
```

```python
# Створюємо модель нейромережі
# Вхідний шар з двома нейронами
input_layer = Input(shape=(2,))
# Прихований шар з 2 нейронами
hidden_layer = Dense(2,)(input_layer)
# Вихідний шар з двома нейронами і функцією активації hard_sigmoid
output_layer = Dense(2, activation='hard_sigmoid')(hidden_layer)
model = tf.keras.models.Model(inputs=input_layer, outputs=output_layer)
```

Використовуємо для вихідного шару функцію активації виду hard sigmoid.

Returns

The hard sigmoid activation, defined as:

- `if x < -2.5: return 0`
- `if x > 2.5: return 1`
- `if -2.5 <= x <= 2.5: return 0.2 * x + 0.5`

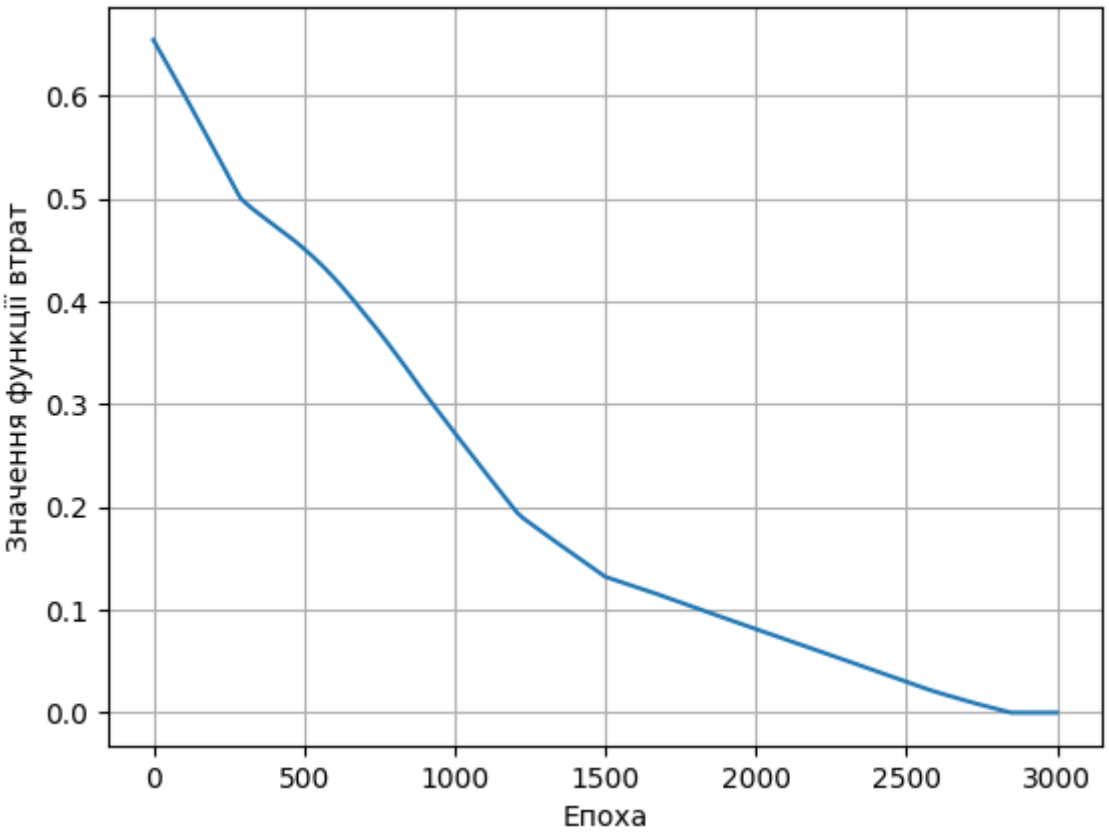
```
# Виводимо опис моделі
model.summary()

# Компіляція моделі з використанням бінарної кросс-ентропії як функції втрати та алгоритму оптимізації Adam
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Навчання мережі
history = model.fit(x = a_b_train, y = nor_nand_train, epochs=3000, batch_size=4)
```

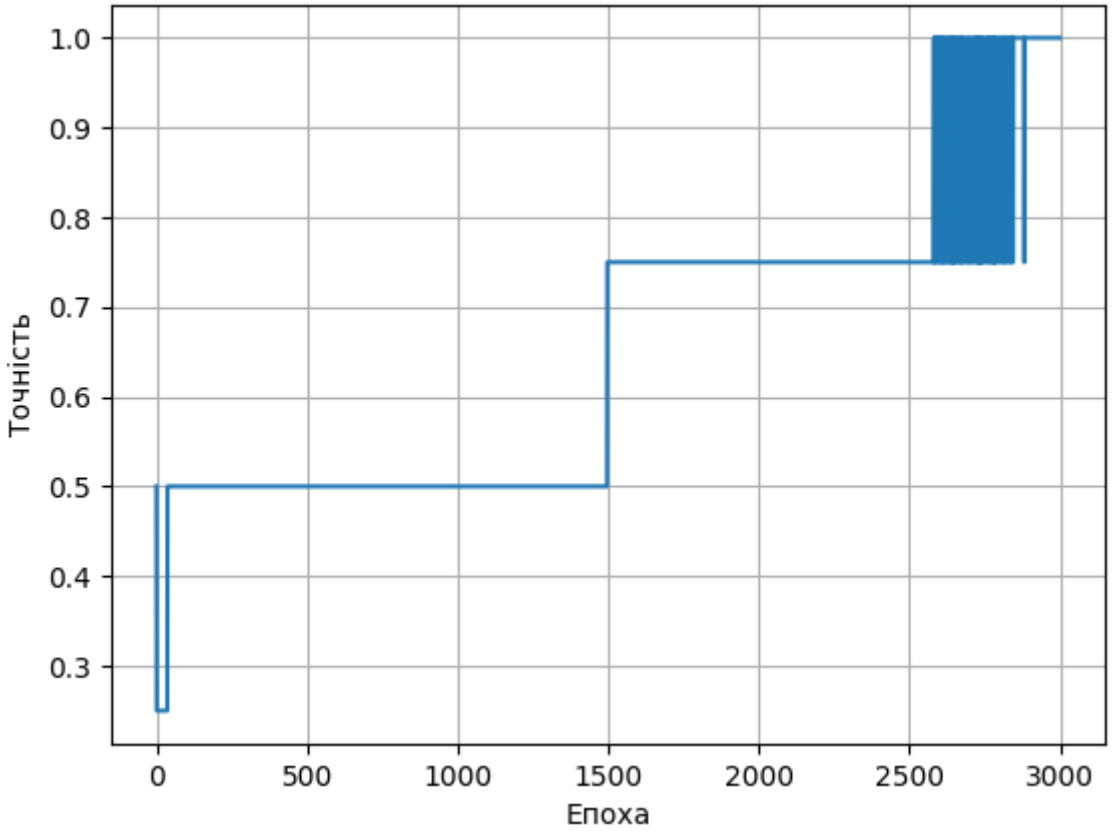
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 2)                 6

=================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
```

```
# Використання навченої моделі для класифікації
predictions = model.predict(a_b_train)

# Виведення результатів класифікації
for i in range(len(a_b_train)):
    print("Вхідні дані: {}, Вихідні дані: {}".format(a_b_train[i], predictions[i]))
```

```
Вхідні дані: [0, 0], Вихідні дані: [1. 1.]
Вхідні дані: [0, 1], Вихідні дані: [0. 1.]
Вхідні дані: [1, 0], Вихідні дані: [0. 1.]
Вхідні дані: [1, 1], Вихідні дані: [0. 0.]
```

```python
# Отримуємо значення функції втрат на кожній епосі тренування
loss = history.history['loss']
accuracy = history.history['accuracy']
plt.plot(loss)
plt.xlabel('Епоха')
plt.ylabel('Значення функції втрат')
plt.grid(True)
plt.show()

# Виводимо значення точності
plt.plot(accuracy)
plt.xlabel('Епоха')
plt.ylabel('Точність')
plt.grid(True)
plt.show()
```


Значення функції втрат
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
500
1000
1500
2000
2500
3000
Епоха


Точність
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
500
1000
1500
2000
2500
3000
Епоха

ВИСНОВКИ

В результаті виконаної лабораторної роботи розроблені моделі нейронних мереж для апроксимації функцій.

Усі матеріали викладенні у репозиторії GitHub, за посиланням https://github.com/Max11mus/-Artifition-Intelect-Lab8.git**.**