МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

З В І Т

Лабораторна робота №6
з дисципліни
«Компьютерні системи штучного інтелекту»

Виконавець:

студент групи КІ-22м Косей М.П.

Керівник:

викладач Саяпін В.Г.

2023

Лабораторна робота №6

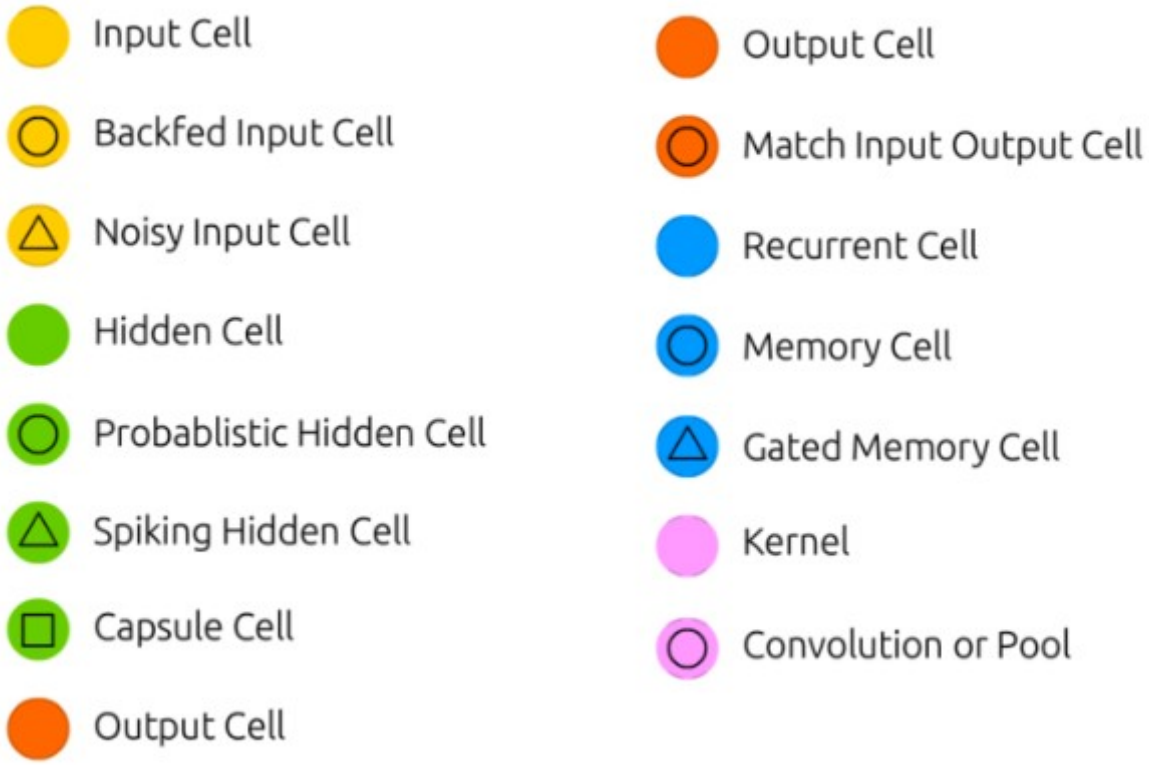
Тема: Програмування моделі штучного нейрону

Мета: Одержати уміння роботи з нейроемулятором, практичні навички з інтелектуальної апроксимації елементарних математичних функцій та програмування нейрону

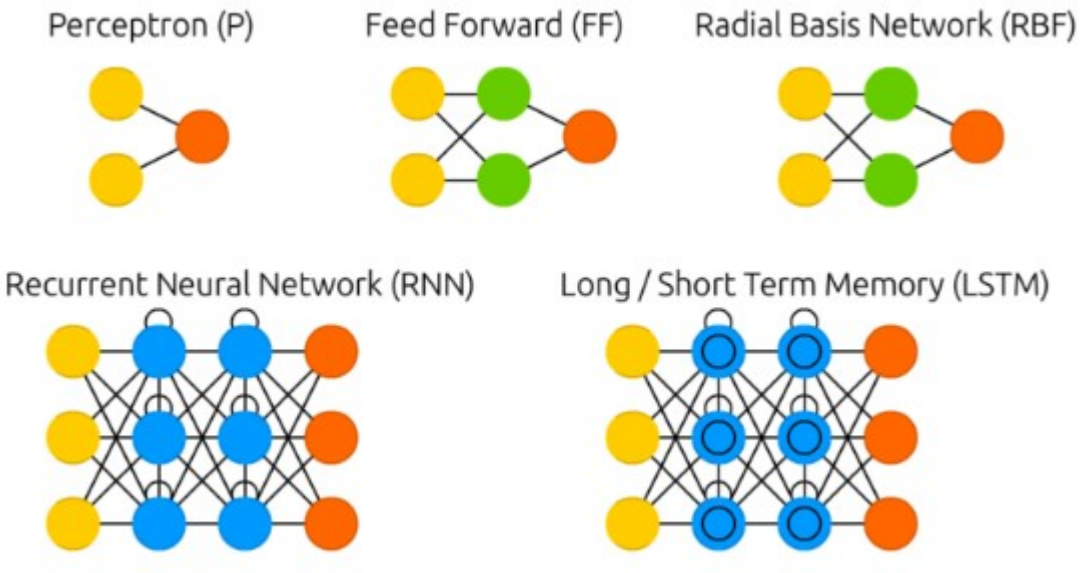
ХІД РОБОТИ

1) Ознайомитись з теоретичними відомостями до лабораторної роботи.

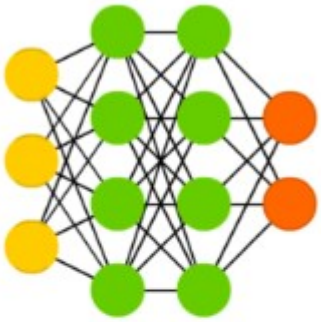
Існує багато різних видів нейронів які використовуються у штучних нейромережах (https://www.asimovinstitute.org/neural-network-zoo/).



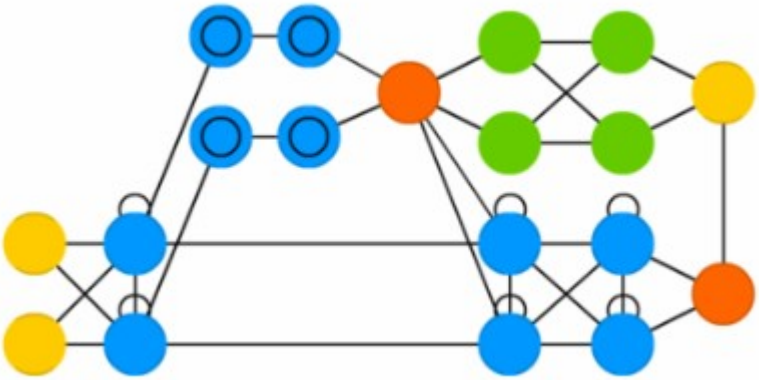
Існує багато різних архітектур нейронних мереж, які використовуються в галузі машинного навчання(https://www.asimovinstitute.org/neural-network-zoo/).



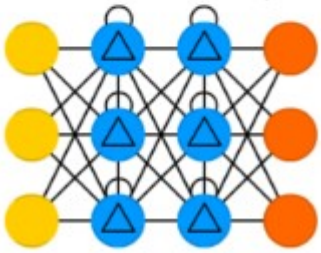
Deep Feed Forward (DFF)



Attention Network (AN)



Gated Recurrent Unit (GRU)



2) Побудувати нейрону мережу для апроксимації функції

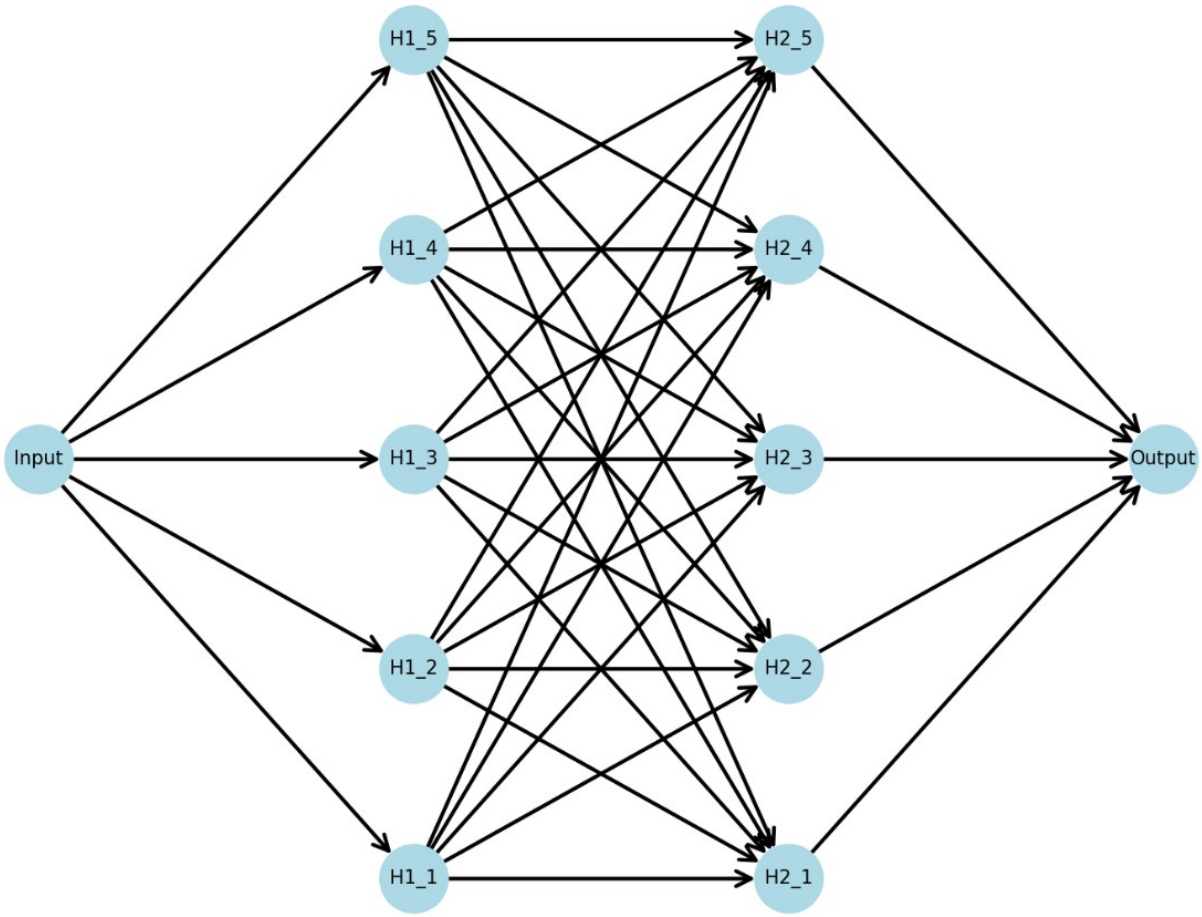
Згідно з варіантом 8 необхідно апроксимувати функцію $\mathbf{Y=2^x}$.

У попередній лабораторній роботі функція апроксимувалась нейронною мережею прямого зв'язку з одним прихованим шаром, в цій роботі застосуємо нейрону мережу з двома прихованими зв’язками.

Для апроксимації використовуємо мережу з двома прихованими шарами по 5 нейронів з сигмоїдальною функцією активації, та одним вхідним та вихідним нейроном.

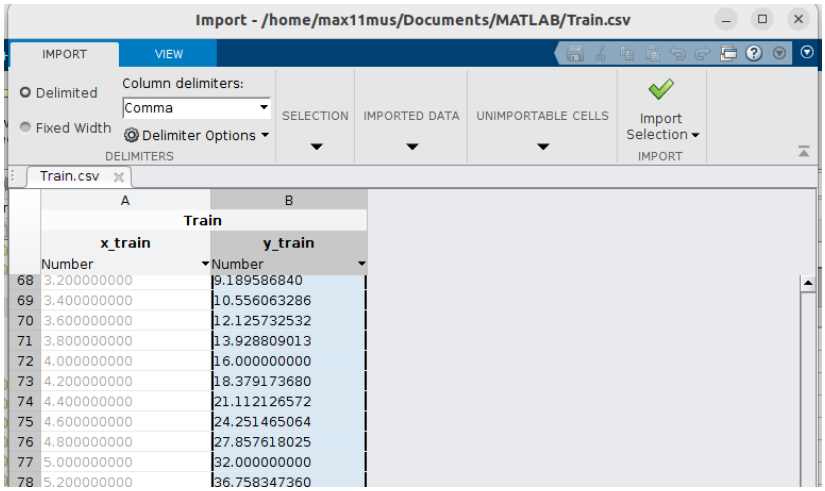
```python
import networkx as nx
import matplotlib.pyplot as plt

# Створюємо граф
G = nx.DiGraph()

# Додаємо вузли
input_nodes = ['Input']
hidden_nodes_layer1 = []
hidden_nodes_layer2 = []
output_nodes = ['Output']

for i in range(5):
    hidden_nodes_layer1.append(f'H1_{i+1}')
    G.add_node(f'H1_{i+1}')

for i in range(5):
    hidden_nodes_layer2.append(f'H2_{i + 1}')
    G.add_node(f'H2_{i + 1}')
```

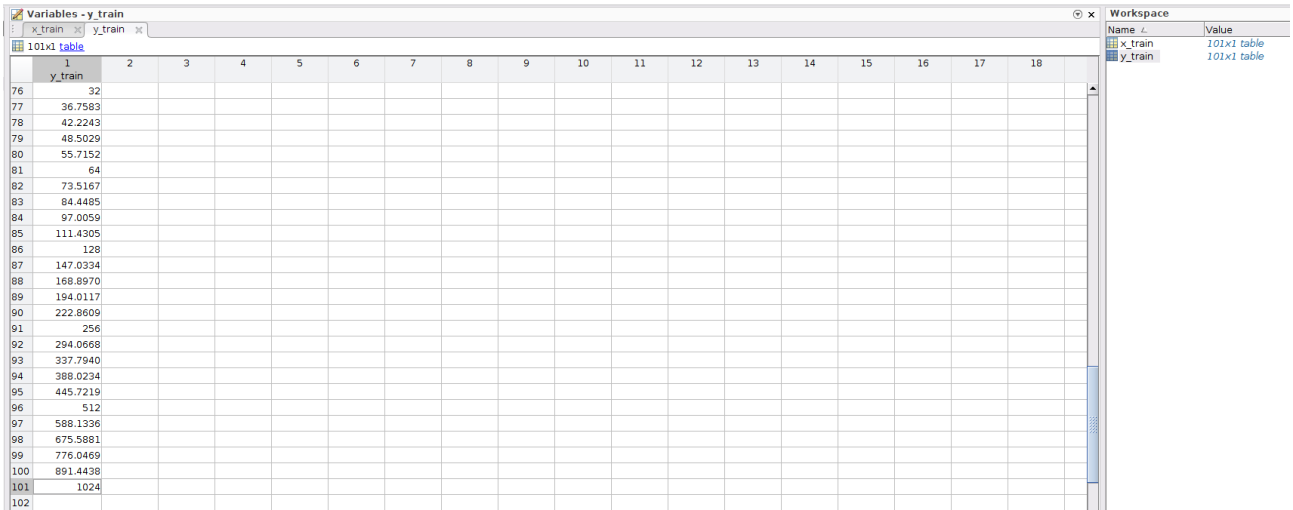
```python
# Додаємо зв'язки між вузлами
for input_node in input_nodes:
    for hidden_node in hidden_nodes_layer1:
        G.add_edge(input_node, hidden_node)

for hidden_node_1 in hidden_nodes_layer1:
    for hidden_node_2 in hidden_nodes_layer2:
        G.add_edge(hidden_node_1, hidden_node_2)

for hidden_node in hidden_nodes_layer2:
    for output_node in output_nodes:
        G.add_edge(hidden_node, output_node)
```



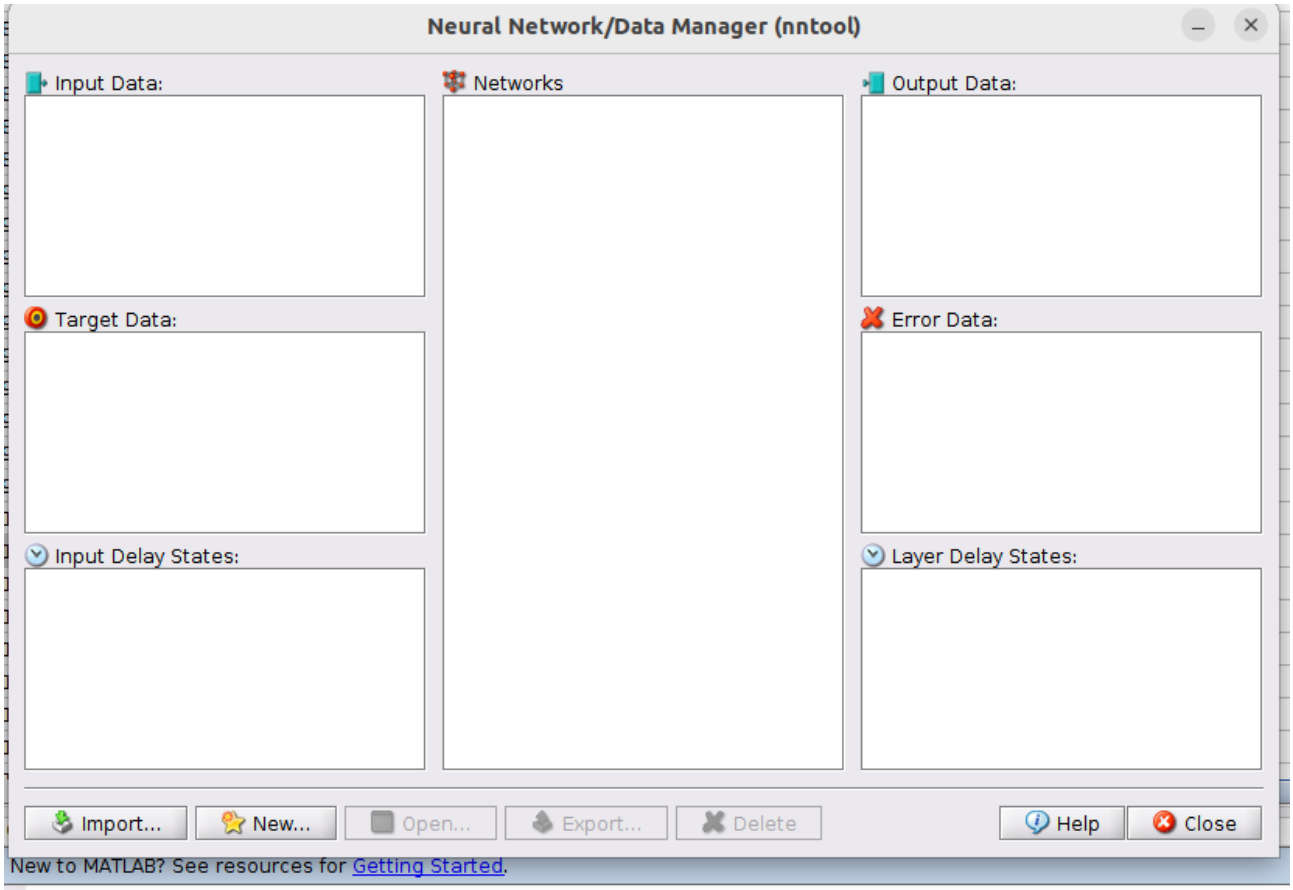
Для моделювання використовуємо MATLAB з використанням вбудованого інструменту NNTool (Neural Network Toolbox).

Завантажуємо датасет x_train та y_train, що містять вхідні дані для навчання моделі для **x** від **-10 до 10 з кроком 0.2**.





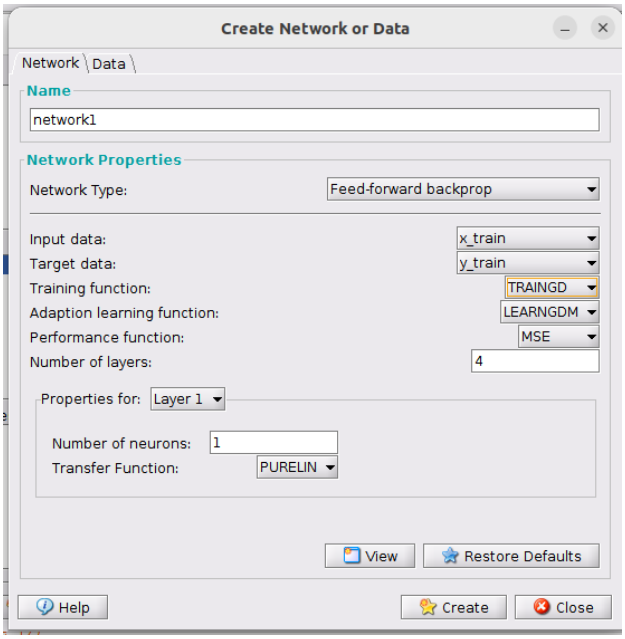
Завантажуємо NNTool, виконавши команду nntool в командному вікні MATLAB.

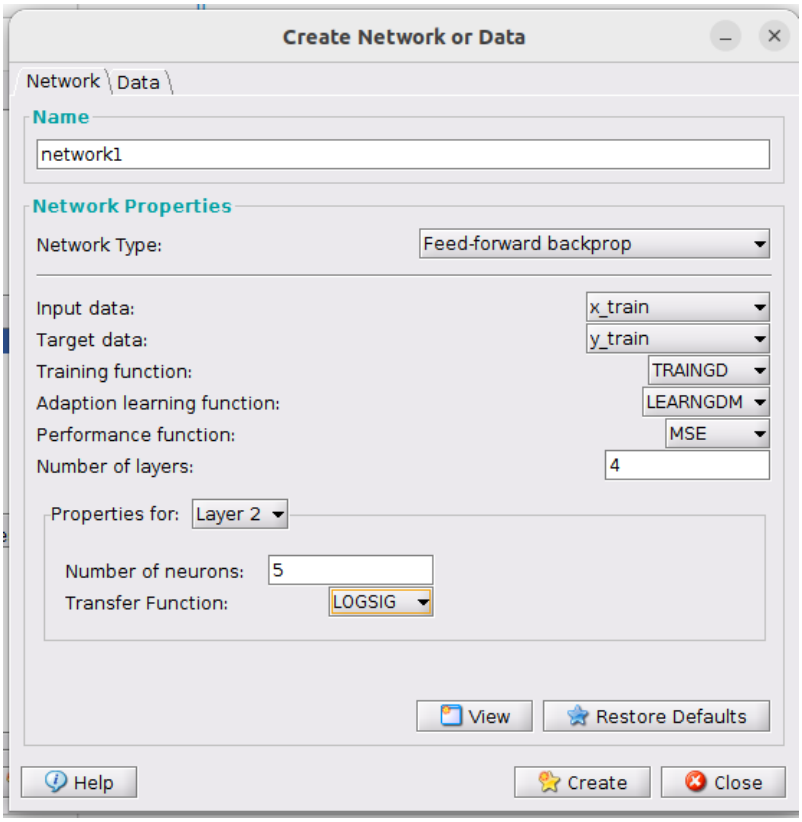


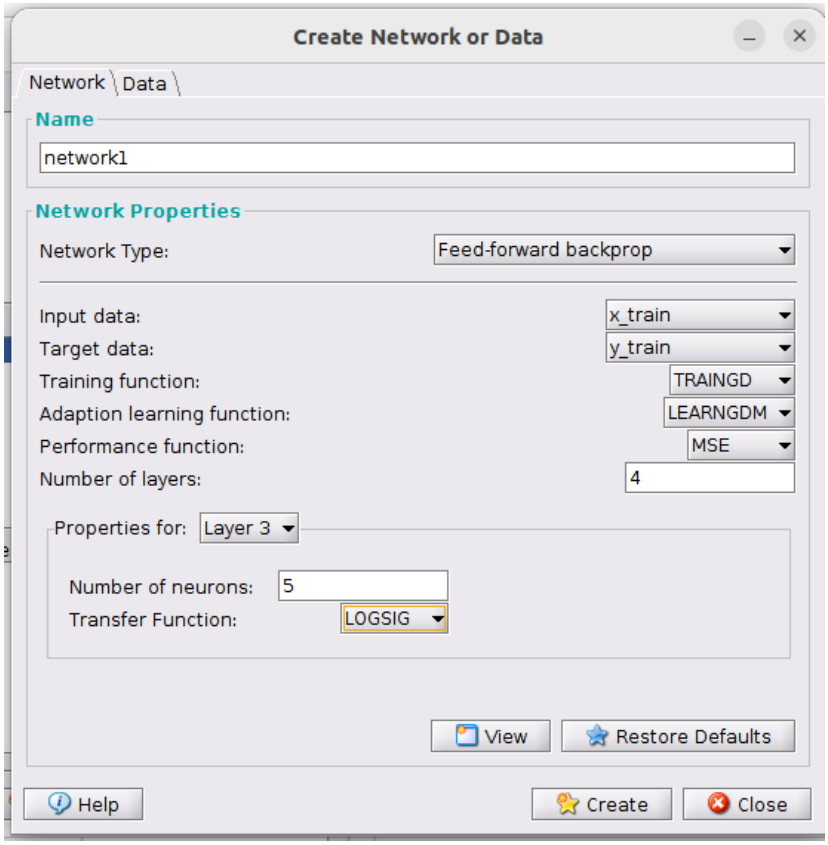

```
>> x_train.x_train(1) = 0;
>> x_train(1,:) = [];
>> y_train(1,:) = [];
>> nntool
Warning: nntool will be removed in a future release. Use nnstart instead.
> In nntool (line 17)
```

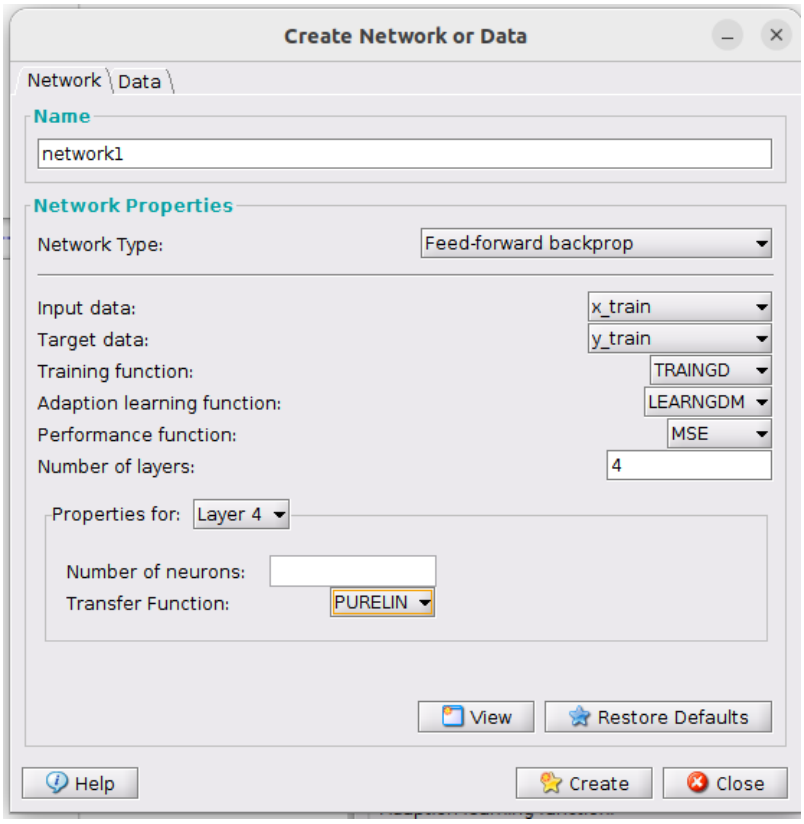
У вікні NNTool обираємо "New" для створення нової нейромережі.

Обираємо тип нейромережі "Feedforward", що є найпоширенішим типом нейромереж. Обираємо кількість прихованих шарів (у нашому випадку два приховані шари) та кількість нейронів у кожному прихованому шарі (у нашому випадку 5 нейронів).

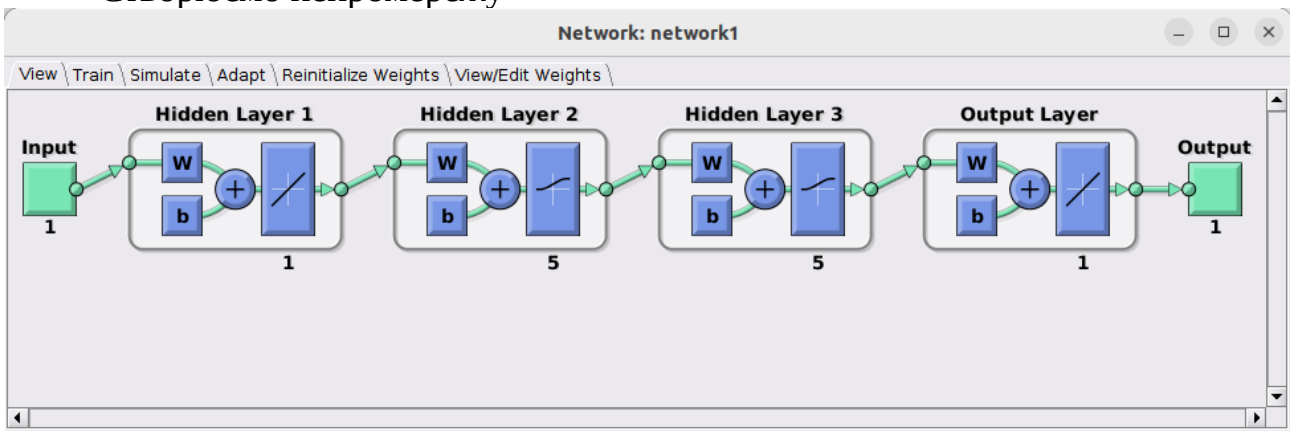



Create Network or Data
Network
Data
Name
network1
Network Properties
Network Type:
Feed-forward backprop
Input data:
x_train
Target data:
y_train
Training function:
TRAINGD
Adaption learning function:
LEARNGDM
Performance function:
MSE
Number of layers:
4
Properties for:
Layer 2
Number of neurons:
5
Transfer Function:
LOGSIG
View
Restore Defaults
Help
Create
Close

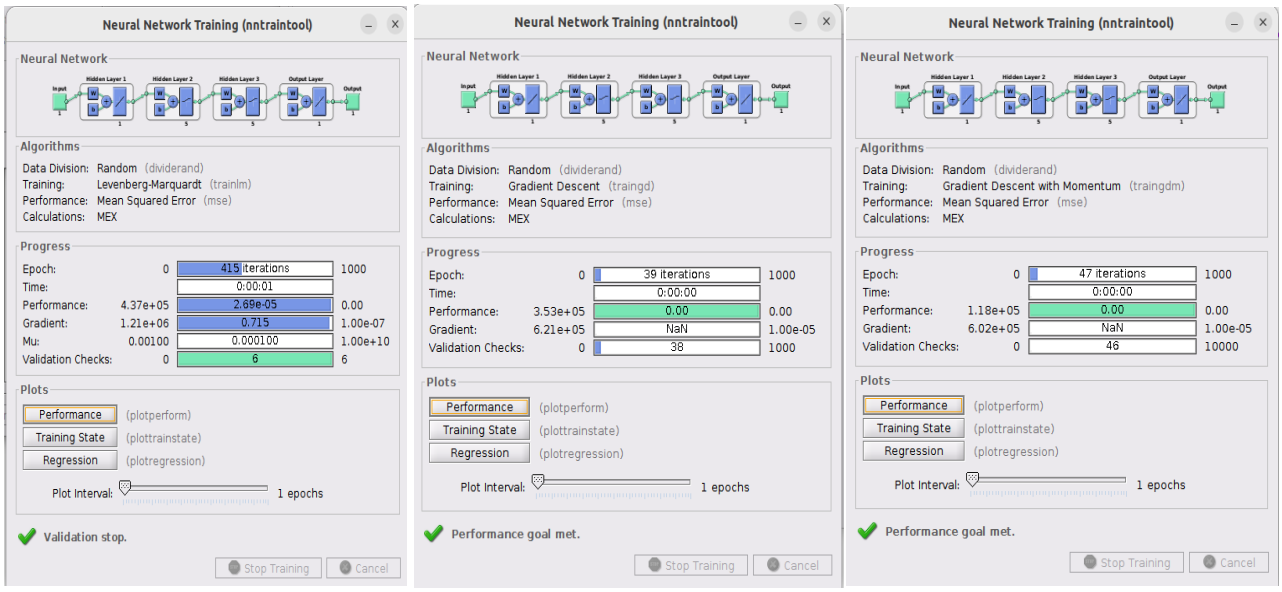

Create Network or Data
Network
Data
Name
network1
Network Properties
Network Type:
Feed-forward backprop
Input data:
x_train
Target data:
y_train
Training function:
TRAINGD
Adaption learning function:
LEARNGDM
Performance function:
MSE
Number of layers:
4
Properties for:
Layer 3
Number of neurons:
5
Transfer Function:
LOGSIG
View
Restore Defaults
Help
Create
Close



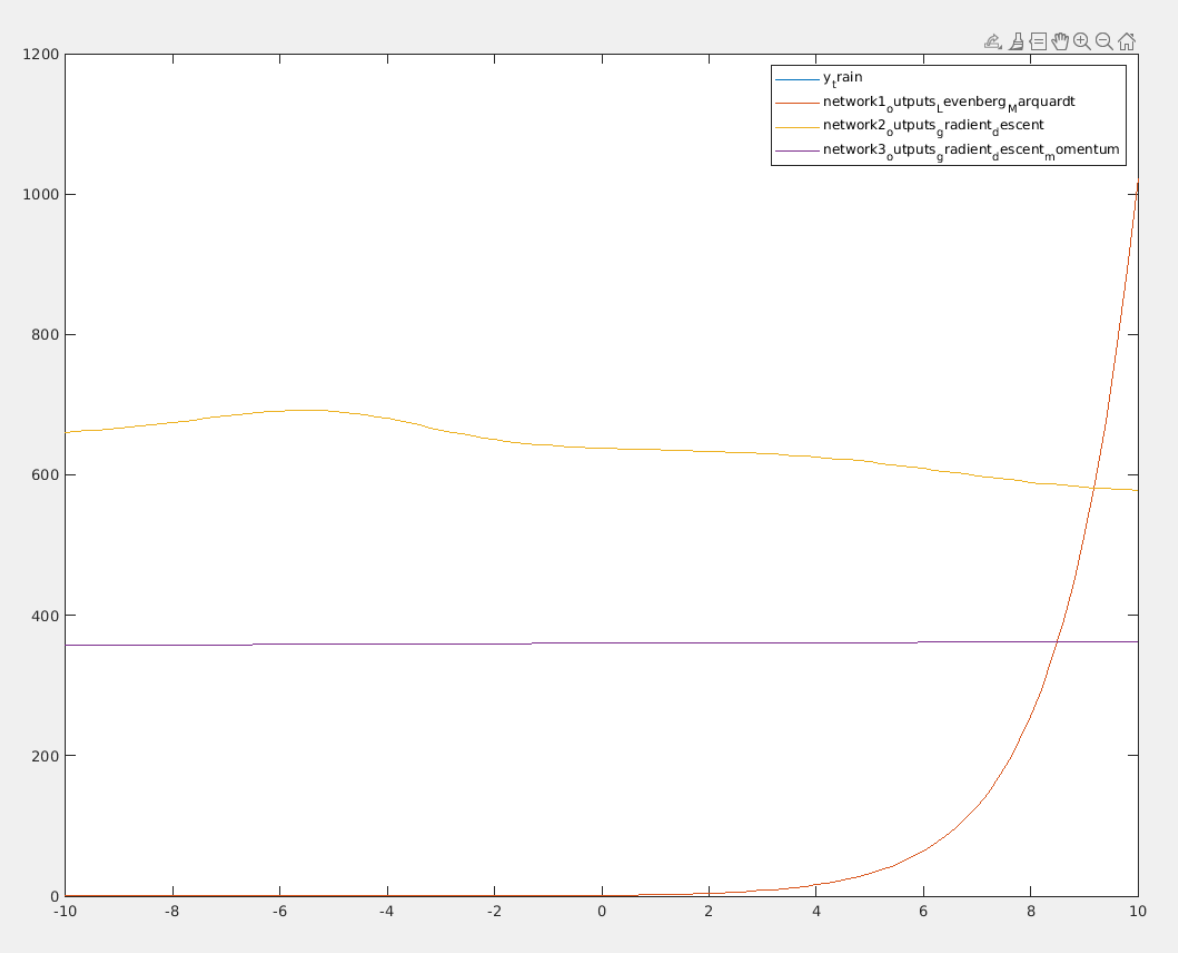
Створюємо нейромережу



Запускаємо процес тренування з різними методами навчання



Запускаємо симуляцію мережі, потім експортуємо дані навчання і будуємо графіки функції **Y=2^x** результати апроксимації функції нейромережею



3) Побудувати нейрону мережу для апроксимації функції (Python)
Використовуємо IDE **PyCharm** та мову програмування **Python**

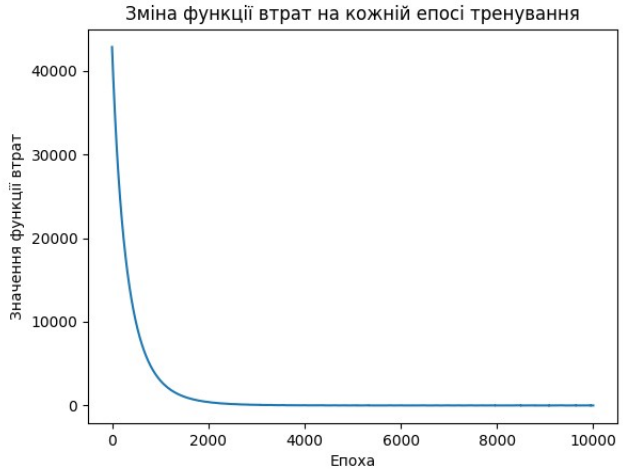

```python
# Імпортуємо необхідні бібліотеки
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense
import numpy as np
import matplotlib.pyplot as plt


# Генеруємо дані для навчання моделі
# Вхідні дані x - значення від -10 до 10 з кроком 0.2
x_train = np.arange(-10,10.2,0.2)
# Вихідні дані y - значення 2^x
y_train = 2**x_train


# Створюємо модель нейромережі
# Вхідний шар з одним нейроном
input_layer = Input(shape=(1,))
```

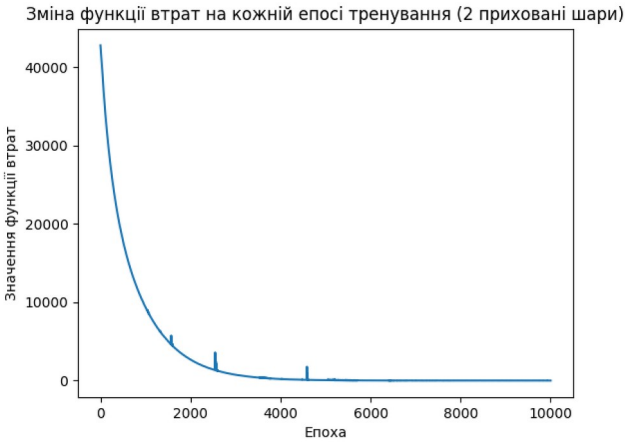
```python
# Два приховані шари з 5 нейронами та функцією активації сігмоїдального типу
hidden_layer_1 = Dense(5, activation='sigmoid')(input_layer)
hidden_layer_2 = Dense(5, activation='sigmoid')(hidden_layer_1)
# Вихідний шар з одним нейроном
output_layer = Dense(1)(hidden_layer_2)

# Створюємо модель нейромережі
model = tf.keras.models.Model(inputs=input_layer, outputs=output_layer)

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

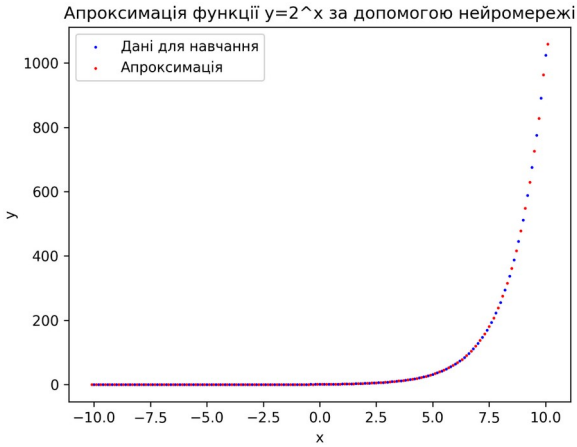
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1)]               0

 dense (Dense)               (None, 5)                 10

 dense_1 (Dense)             (None, 5)                 30

 dense_2 (Dense)             (None, 1)                 6

=================================================================
Total params: 46
Trainable params: 46
Non-trainable params: 0
```

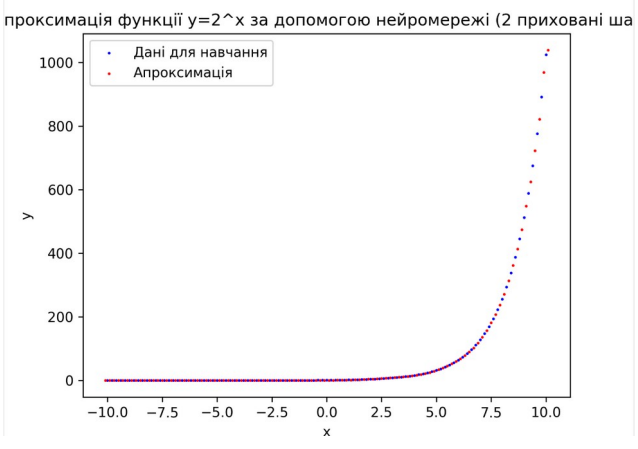
```python
# Компілюємо модель, встановлюючи функцію втрат та оптимізатор
model.compile(loss='mean_squared_error', optimizer=keras.optimizers.Adam(0.1))

# Навчаємо модель на вхідних даних
history = model.fit(x = x_train, y = y_train, epochs=10000, batch_size=101)

# Отримуємо значення функції втрат на кожній епосі тренування
loss = history.history['loss']

# Виводимо значення функції втрат
plt.plot(loss)
plt.xlabel('Епоха')
plt.ylabel('Значення функції втрат')
plt.title('Зміна функції втрат на кожній епосі тренування')
plt.show()
```

```
34      # Проводимо апроксимацію на тестових даних навчання
35      # Виконуємо передбачення за допомогою моделі
36      x_test = np.arange(-10.1,10.2,0.2)
37      y_test = model.predict(x_test)
38
39      # Виводимо результати
40      plt.scatter(x_train, y_train, color='blue', label='Дані для навчання')
41      plt.plot(x_test, y_test, color='red', label='Апроксимація')
42      plt.legend()
43      plt.xlabel('x')
44      plt.ylabel('y')
45      plt.title('Апроксимація функції y=2^x за допомогою нейромережі')
46      plt.savefig(fname='aproximation.png', dpi=300)
47
```









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-Lab6**.**