МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

З В І Т

Лабораторна робота №5
з дисципліни
«Компьютерні системи штучного інтелекту»

Виконавець:

студент групи КІ-22м Косей М.П.

Керівник:

викладач Саяпін В.Г.

2023

Лабораторна робота №4

Тема: Ознайомлення з нейроемуляторами та їх застосування для апроксимації функцій.

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.

Нейронні мережі є математичними моделями, які використовуються в машинному навчанні для розв'язання різноманітних завдань, таких як класифікація, регресія, синтез зображень тощо. Вони були названі "нейронними" мережами через своє вдосконалення від біологічної нейронної системи, яка є основою функціонування мозку людей та інших живих організмів.

Нейрони в біологічних нервових системах взаємодіють між собою, передаючи сигнали у вигляді електричних імпульсів, відомих як дії потенціали, через спеціалізовані з'єднання, відомі як синапси. Аналогія між біологічними нейронами і нейронами мережі полягає в тому, що нейронні мережі складаються зі штучних нейронів, або вузлів, які мають вхідні дані (подібно до дендритів біологічних нейронів), ваги, які відповідають силі з'єднань між нейронами (подібно до синапсів), функції активації, які можна розглядати як процес передачі сигналу нейрону, та вихідні дані (подібно до аксонів біологічних нейронів), які передають сигнал до наступних нейронів у мережі.

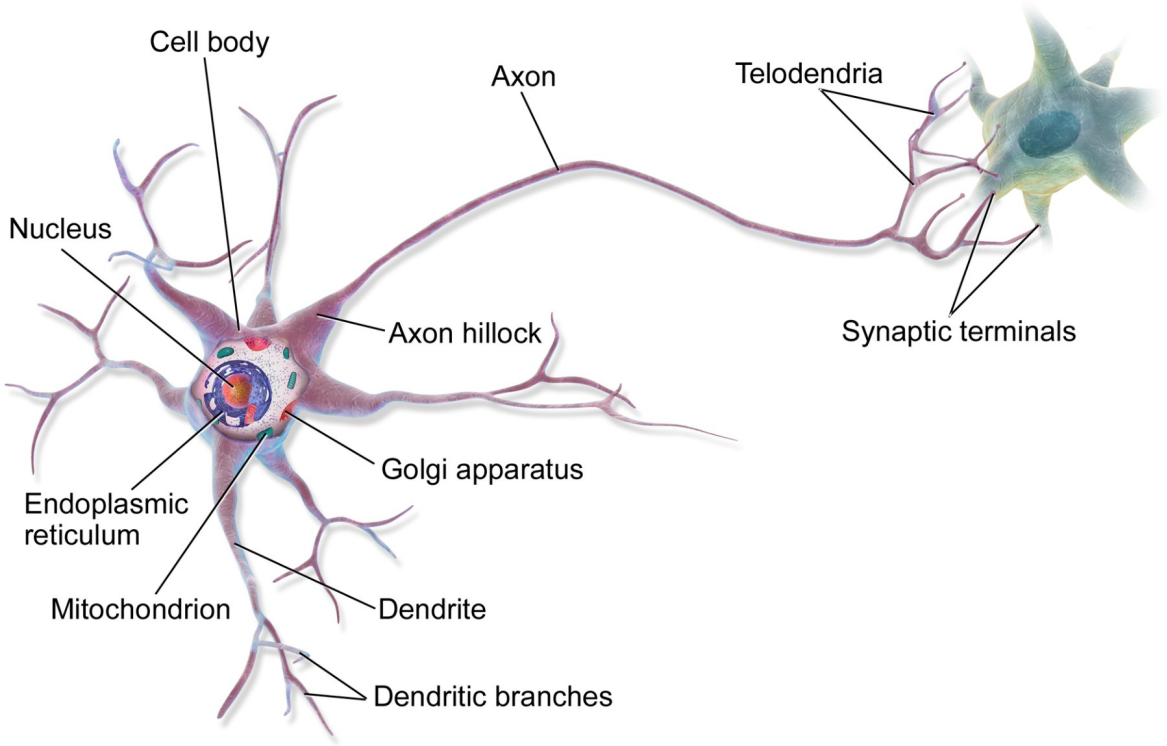
Біологічний нейрон складається з таких основних елементів:

Сома: це тіло клітини, що містить ядро, де знаходиться генетична інформація.

Дендрити: це відростки, які виходять з соми і служать для прийому вхідних сигналів від інших нейронів.

Аксон: це відрізок, який виходить з соми і передає вихідні сигнали від нейрона до інших нейронів або до ефекторних клітин (таких як м'язи або залози).

Синапси: це точки зв'язку між аксоном одного нейрона та дендритами іншого нейрона, де відбувається передача сигналів за допомогою хімічних реакцій.

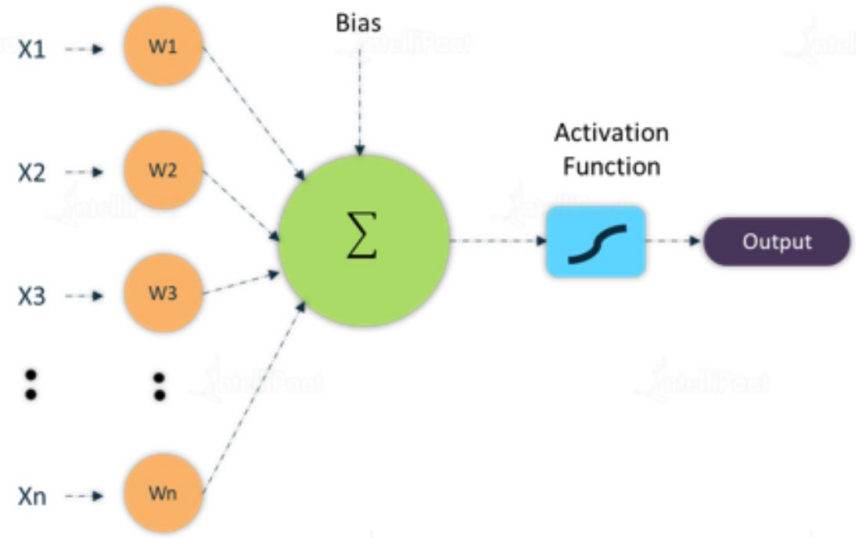


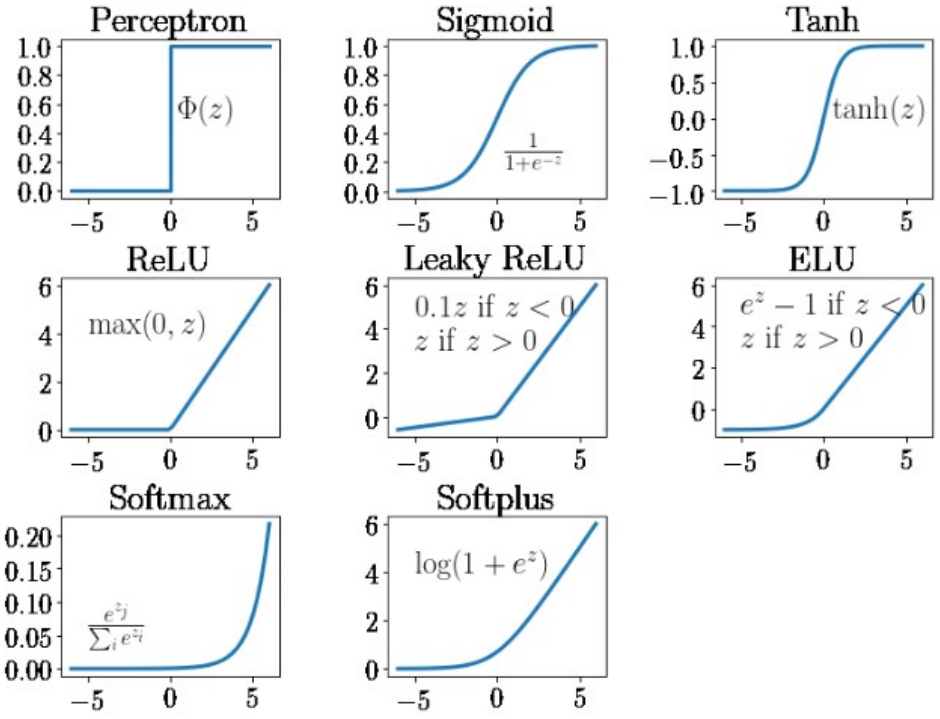
Штучні нейрони в нейронній мережі взаємодіють між собою шляхом передачі даних у вигляді числових значень через з'єднання зі своїми вагами. Задача нейронної мережі полягає в тому, щоб вчитися визначати оптимальні ваги на основі вхідних даних, з метою виконання конкретного завдан.

Штучні нейрони працюють шляхом виконання декількох кроків. Спочатку вони отримують вхідні дані, які можуть бути числовими значеннями або векторами. Далі, ці дані перемножуються на ваги та сумуються в суматорі. Потім до цієї суми додається біас, і отримане значення передається через функцію активації. Вихідний сигнал нейрона визначається значенням функції активації, яке може бути передано наступним нейронам в мережі.

Функції активації можуть бути різними, такими як сигмоїда, ReLU (Rectified Linear Unit), тангенс гіперболічний (tanh) та інші. Вони відповідають за нелінійність штучного нейрона, що дозволяє нейронним мережам виконувати складні не лінійні завдання.

Штучні нейрони та їх ваги можуть бути оптимізовані під час процесу навчання нейронної мережі, де алгоритми оптимізації змінюють значення ваг та біасів, щоб нейронна мережа навчалась краще вирішувати поставлені завдання.





У таблиці наведено порівняльні характеристики між біологічними мережами, такими як мозок, і нейронними мережами, які використовуються в машинному навчанні та інших технічних застосуваннях. Основні відмінності між ними включають складність, адаптивність, самоорганізацію, взаємодію з організмом, гнучкість, програмованість, швидкість обчислень, масштабованість, архітектуру, процес навчання та пам'ять.

Особливості	**Біологічні мережі**	**Нейронні мережі**
Структура	Складна структура з мільярдами нейронів, різними типами нейронів, різними рівнями організації та спеціалізованими відділами	Спрощена структура з використанням одного або кількох типів штучних нейронів, з'єднаних у шари або шари з різними зв'язками між ними
Розмір	Великий розмір з мільярдами нейронів та трільйонами зв'язків між ними	Відносно менший розмір з кількома сотнями до кількох мільйонів нейронів, залежно від архітектури та розміру мережі
Функції	Складний набір функцій, таких як сприйняття, розуміння, вирішення проблем, прийняття рішень, емоції тощо	Зосереджені на вирішенні конкретних завдань, таких як класифікація, регресія, генерація контенту тощо
Навчання	Природне навчання на основі взаємодії з оточуючим середовищем та взаємодії між нейронами, зміна мережі з часом	Навчання на основі великої кількості даних та процесів навчання, таких як зворотне поширення помилок або навчання з підсиленням
Апаратне забезпечення	Біологічні нейрони, що взаємодіють на основі хімічних та електричних сигналів	Штучні нейрони, які імітують біологічні нейрони, та використовують обчислювальні принципи та алгоритми

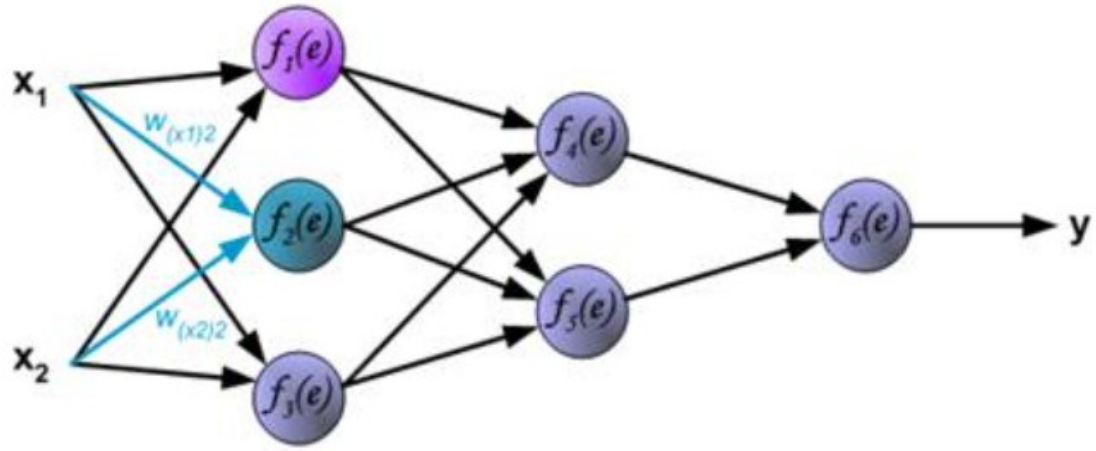
Один з методів навчання нейромереж - зворотнє поширення (**Backpropagation**) - це один з найпоширеніших методів навчання нейронних мереж, який використовується для тренування штучних нейронних мереж з наглядом (supervised learning). Він використовується для покращення ваг (внутрішніх параметрів) мережі на основі помилок між прогнозованими виходами мережі та відповідними цільовими виходами (мітками) у навчальному наборі даних.

Основна ідея за методом зворотного поширення полягає в пошуку градієнту функції втрати (loss function) відносно ваг мережі, і використання цього градієнту для оновлення ваг таким чином, щоб зменшити помилки прогнозів мережі на навчальному наборі даних. Оновлення ваг відбувається за допомогою алгоритму градієнтного спуску (gradient descent), де ваги оновлюються у напрямку, протилежному до градієнту функції втрати, з метою мінімізації функції втрати.

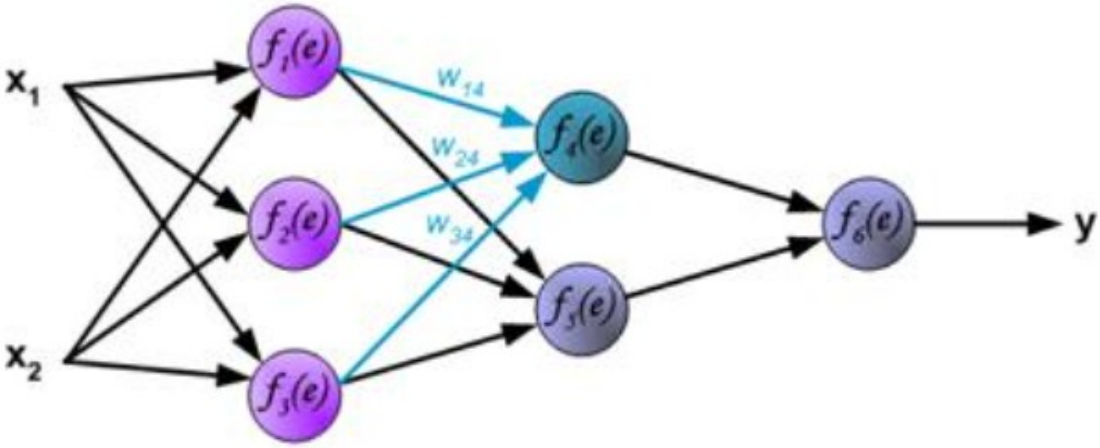
Процес навчання нейромережі зворотнім поширенням можна узагальнити наступними кроками:

Ініціалізація ваг мережі:

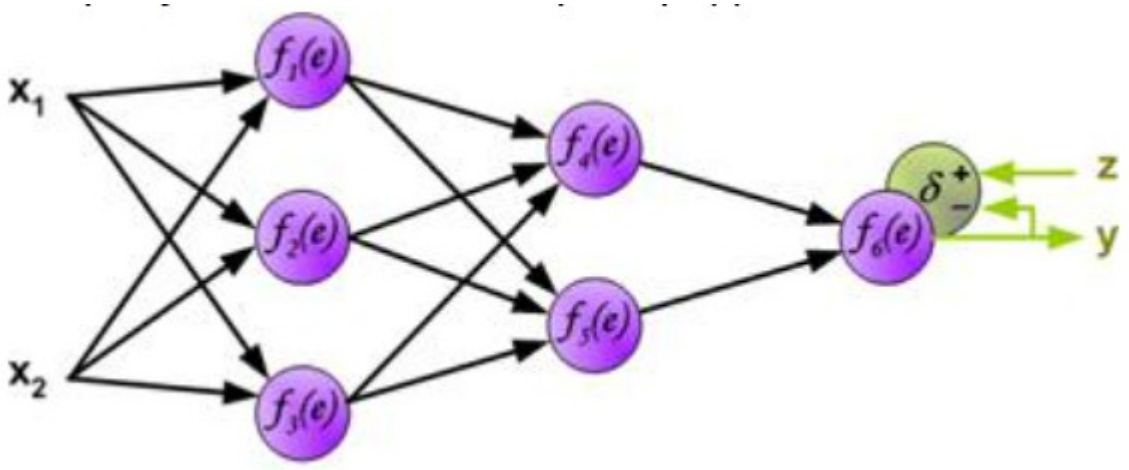
1) Ваги мережі ініціалізуються випадковими значеннями або за допомогою іншого попереднього методу.



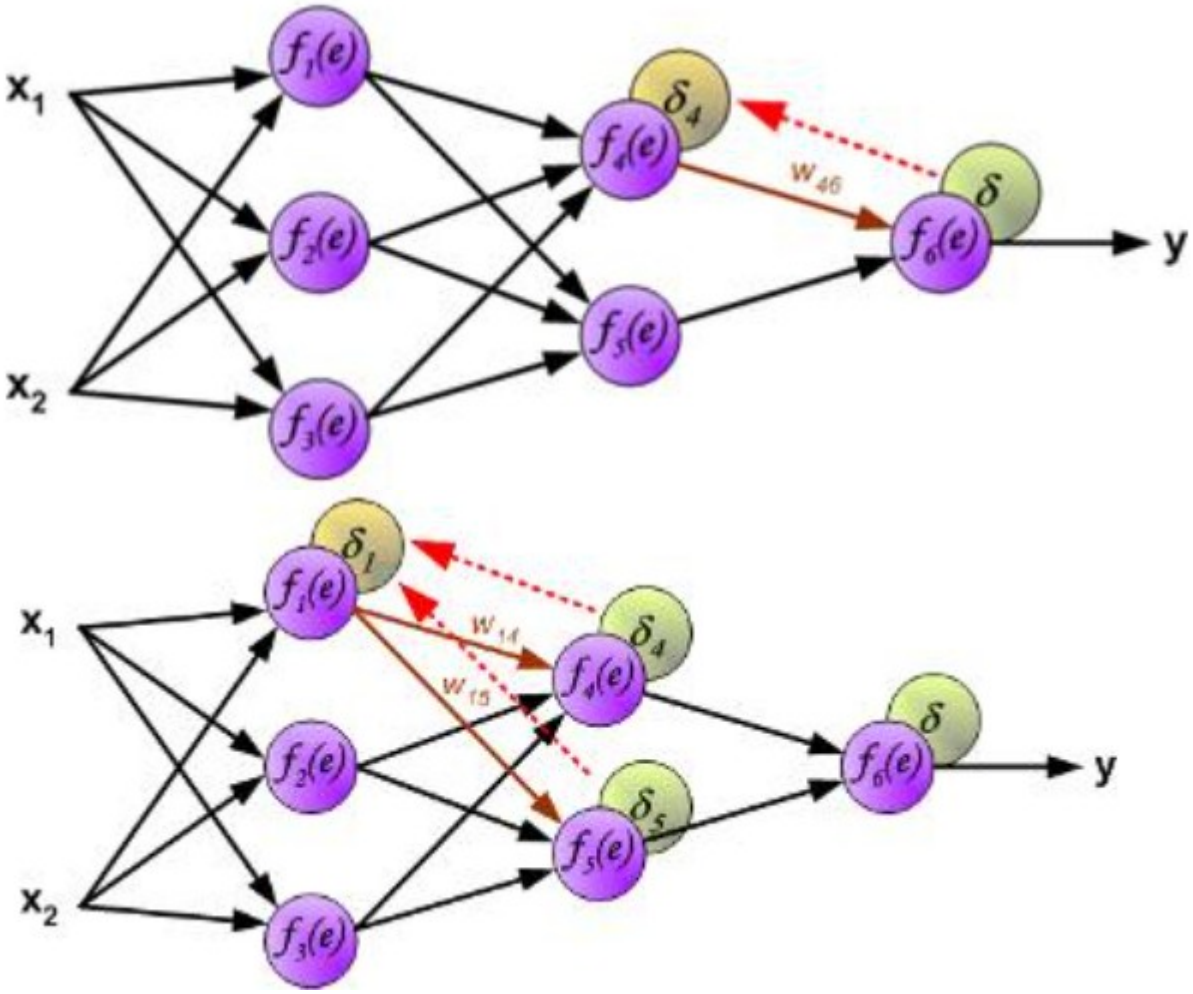
2) Прямий прохід (forward pass): Вхідні дані проходять крізь мережу від вхідного до вихідного шару, розраховуючи вихідні значення мережі.



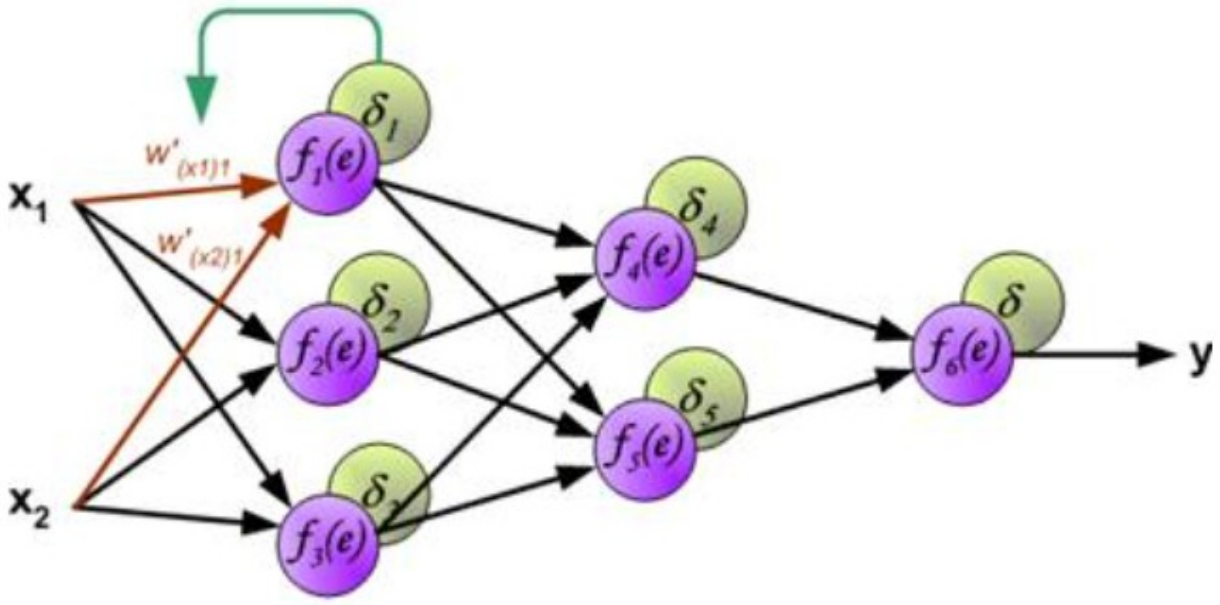
3) Розрахунок помилок (loss calculation): Розраховується значення функції втрати, яка відображає різницю між прогнозованими виходами мережі та цільовими виходами (мітками) у навчальному наборі даних.

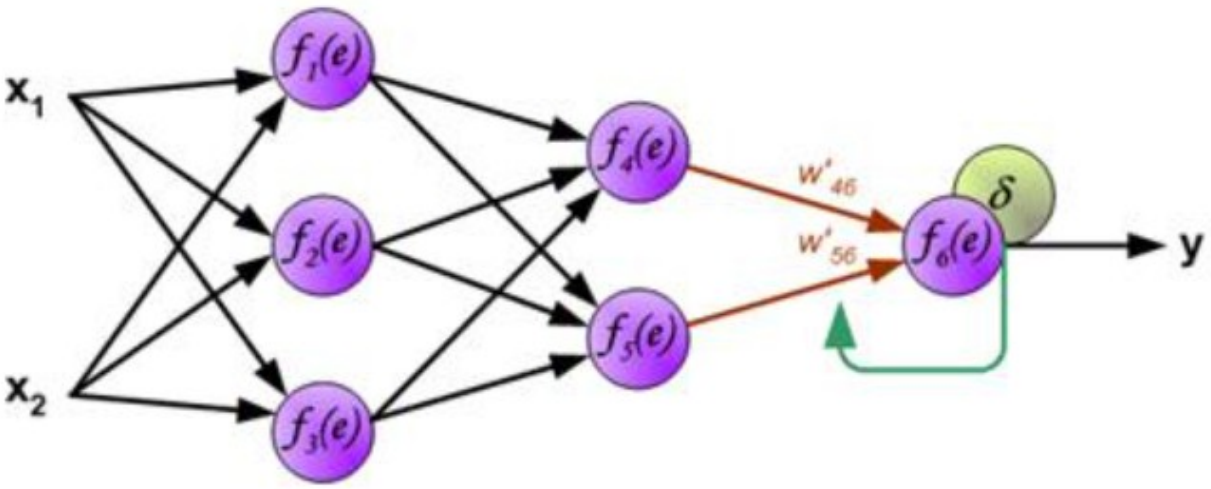


4) Зворотнє поширення градієнту (gradient backpropagation): За допомогою правила ланцюгового правила (chain rule) розраховується градієнт функції втрати відносно вихідних значень мережі. Градієнт використовується для розрахунку градієнтів ваг на всіх рівнях мережі, починаючи з вихідного шару і рухаючись назад до вхідного шару.



5) Оновлення ваг (weight update): Ваги мережі оновлюються на основі розрахованого градієнту. Зазвичай використовується алгоритм градієнтного спуску (gradient descent) або його варіанти, які використовують швидкість навчання (learning rate) для контролю швидкості оновлення ваг.





6) Повторення процесу: Процес прямого проходу, розрахунку помилок, зворотного поширення градієнту та оновлення ваг повторюється для кожного прикладу у навчальному наборі даних протягом кількох епох (повторень) з метою покращення точності мережі.
7) Зупинка навчання: Навчання може бути зупинено за певної умови, такої як досягнення максимальної кількості епох, досягнення заданої точності, або іншої заданої умови.

ПЗ нейросимуляторів дозволяє моделювати роботу нейронних мереж на комп'ютері та надають інструменти та функції для реалізації цього методу навчання та тренування нейронних мереж.

Приклади програмного забезпечення нейросимуляторів:

- **NeuralWorks Pro II/Plus**: Розроблений компанією Aspen Technology, Inc., цей нейросимулятор володіє рядом потужних функцій для моделювання нейронних мереж, включаючи можливість тренування та тестування різних архітектур мереж, використання різних алгоритмів оптимізації та візуалізації результатів.
- **NeuroSolution**: Розроблений компанією NeuroDimension, Inc., NeuroSolution є потужним інструментом для розробки, тренування та тестування нейронних мереж. Він має вбудовані функції для оптимізації гіперпараметрів, візуалізації результатів та інтерфейс, що дозволяє легко створювати складні моделі нейронних мереж.
- **MatLab + Neural Network Toolbox**: Це програмне забезпечення розроблене компанією MathWorks, Inc., включає Neural Network Toolbox - пакет інструментів для моделювання, тренування та тестування нейронних мереж в середовищі MatLab. Він має багато функцій для роботи з нейронними мережами, включаючи підтримку різних архітектур мереж, алгоритмів тренування та візуалізації результатів.
- **BrainMaker**: Розроблений компанією California Scientific Software, Inc., BrainMaker є потужним інструментом для моделювання нейронних мереж та аналізу даних. Він має вбудовані функції для тренування та тестування мереж, оптимізації гіперпараметрів, візуалізації результатів та багато інших функцій.
- **STATISTICA Neural Networks**: Розроблений компанією Statsoft, Inc., STATISTICA Neural Networks є частиною більшої платформи STATISTICA для аналізу даних.

2) Побудувати нейрону мережу для апроксимації функції

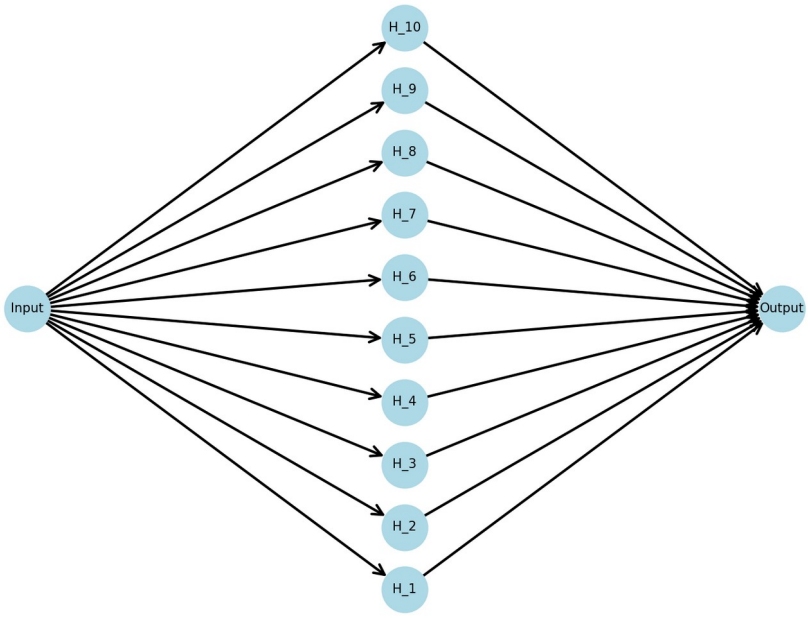
Згідно з варіантом 8 необхідно апроксимувати функцію $\mathbf{Y=2^x}$.

Згідно з універсальною теоремою апроксимації доведеної Джоржем Цибенко у 1989 році, яка стверджує що штучна нейронна мережа прямого зв'язку (англ. feed-forward; у яких зв'язки не утворюють циклів) з одним прихованим шаром і сигмоїдною функцією активацією може апроксимувати будь-яку неперервну функцію багатьох змінних з будь-якою точністю.

Умовами є достатня кількість нейронів прихованого шару, вдалий підбір вагових коефіцієнтів зв’язків нейронів.

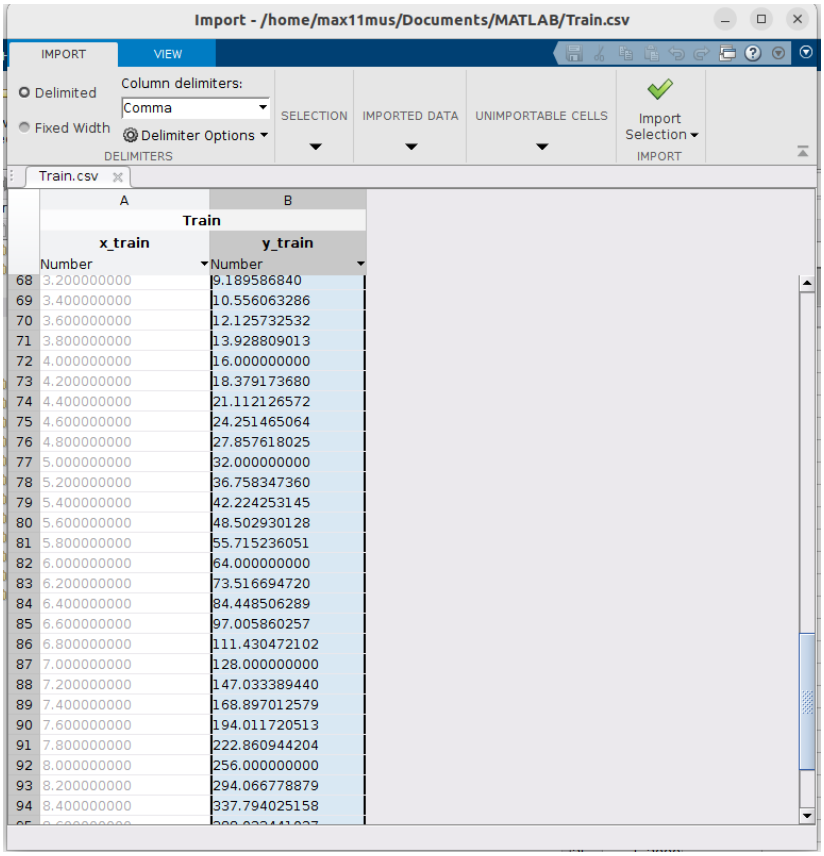
Для апроксимації використовуємо мережу з одним прихованим шаром з 10 нейронів з сигмоїдальною функцією активації, та одним вхідним та вихідним нейроном.

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

# Створюємо граф
G = nx.DiGraph()

# Додаємо вузли
input_nodes = ['Input']
hidden_nodes = []
output_nodes = ['Output']

for i in range(10):
    hidden_nodes.append(f'H_{i+1}')
    G.add_node(f'H_{i+1}')

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

for hidden_node in hidden_nodes:
    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, 900)) for node in input_nodes)  # Позиції вхідних вузлів
pos.update((node, (50, i*200)) for i, node in enumerate(hidden_nodes))  # Позиції вузлів прихованого шару
pos.update((node, (100, 900)) for node in 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)
```

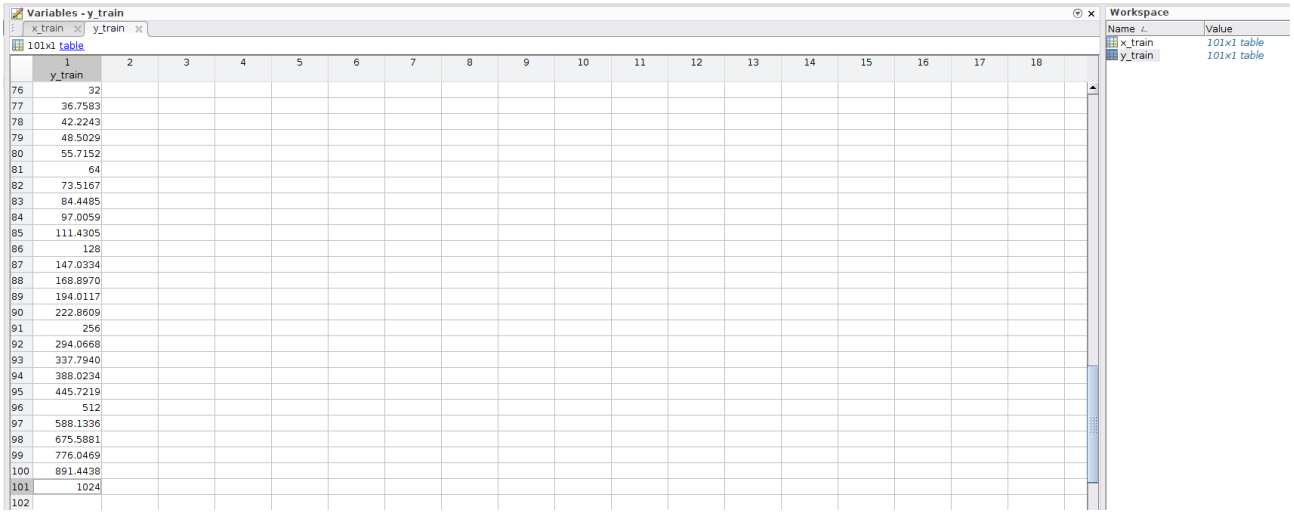


Для моделювання використовуємо MATLAB з використанням вбудованого інструменту NNTool (Neural Network Toolbox).

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

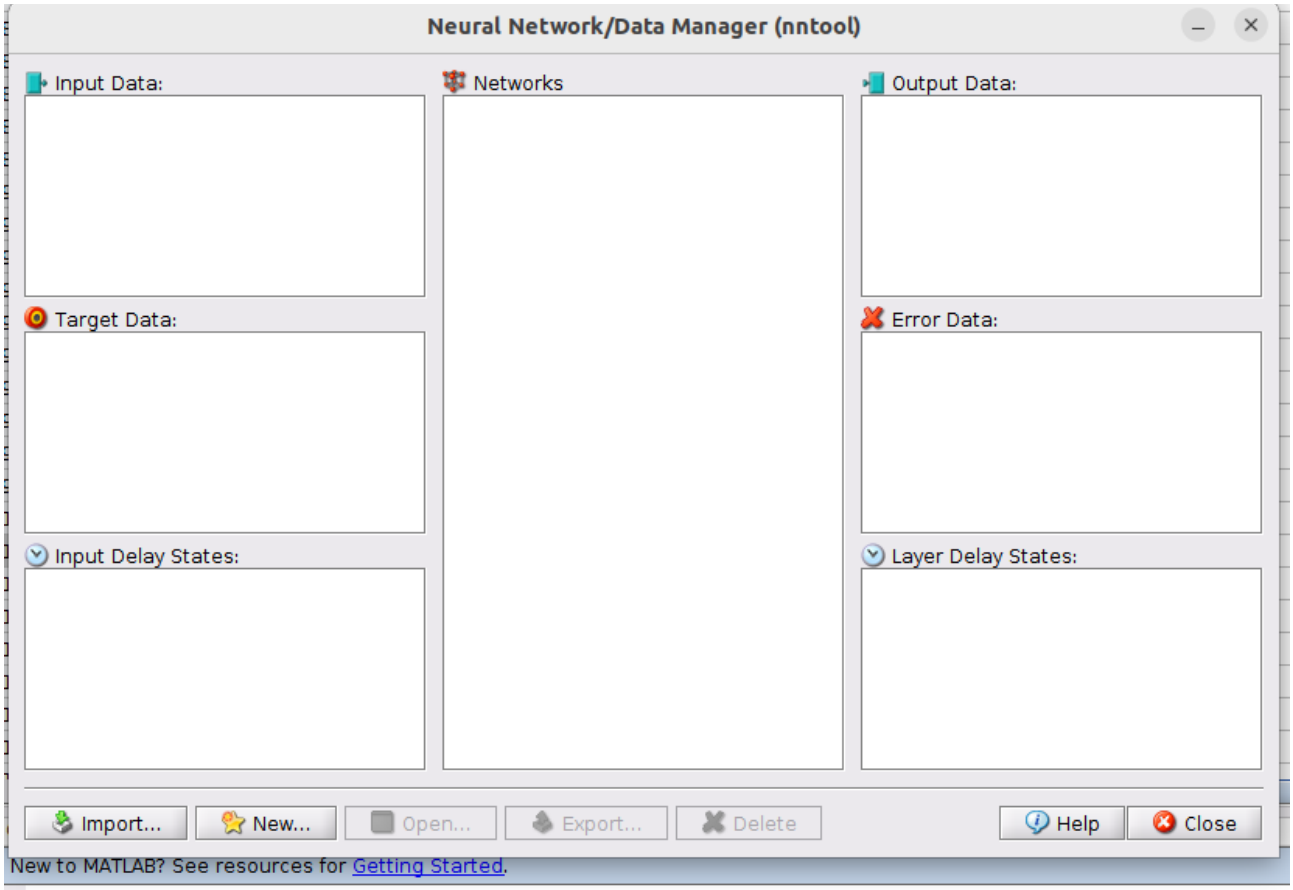


	x_train	y_train
68	3.200000000	9.189586840
69	3.400000000	10.556063286
70	3.600000000	12.125732532
71	3.800000000	13.928809013
72	4.000000000	16.000000000
73	4.200000000	18.379173680
74	4.400000000	21.112126572
75	4.600000000	24.251465064
76	4.800000000	27.857618025
77	5.000000000	32.000000000
78	5.200000000	36.758347360
79	5.400000000	42.224253145
80	5.600000000	48.502930128
81	5.800000000	55.715236051
82	6.000000000	64.000000000
83	6.200000000	73.516694720
84	6.400000000	84.448506289
85	6.600000000	97.005860257
86	6.800000000	111.430472102
87	7.000000000	128.000000000
88	7.200000000	147.033389440
89	7.400000000	168.897012579
90	7.600000000	194.011720513
91	7.800000000	222.860944204
92	8.000000000	256.000000000
93	8.200000000	294.066778879
94	8.400000000	337.794025158



	1 y_train
76	32
77	36.7583
78	42.2243
79	48.5029
80	55.7152
81	64
82	73.5167
83	84.4485
84	97.0059
85	111.4305
86	128
87	147.0334
88	168.8970
89	194.0117
90	222.8609
91	256
92	294.0668
93	337.7940
94	388.0234
95	445.7219
96	512
97	588.1336
98	675.5881
99	776.0469
100	891.4438
101	1024
102	

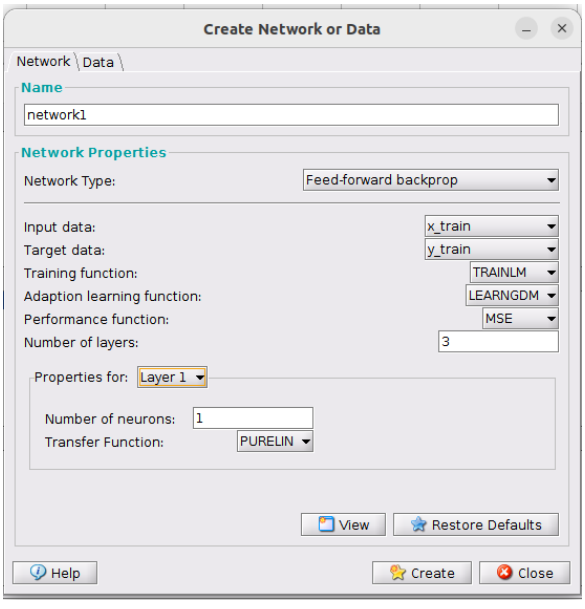
Завантажуємо NNTool, виконавши команду nntool в командному вікні MATLAB.



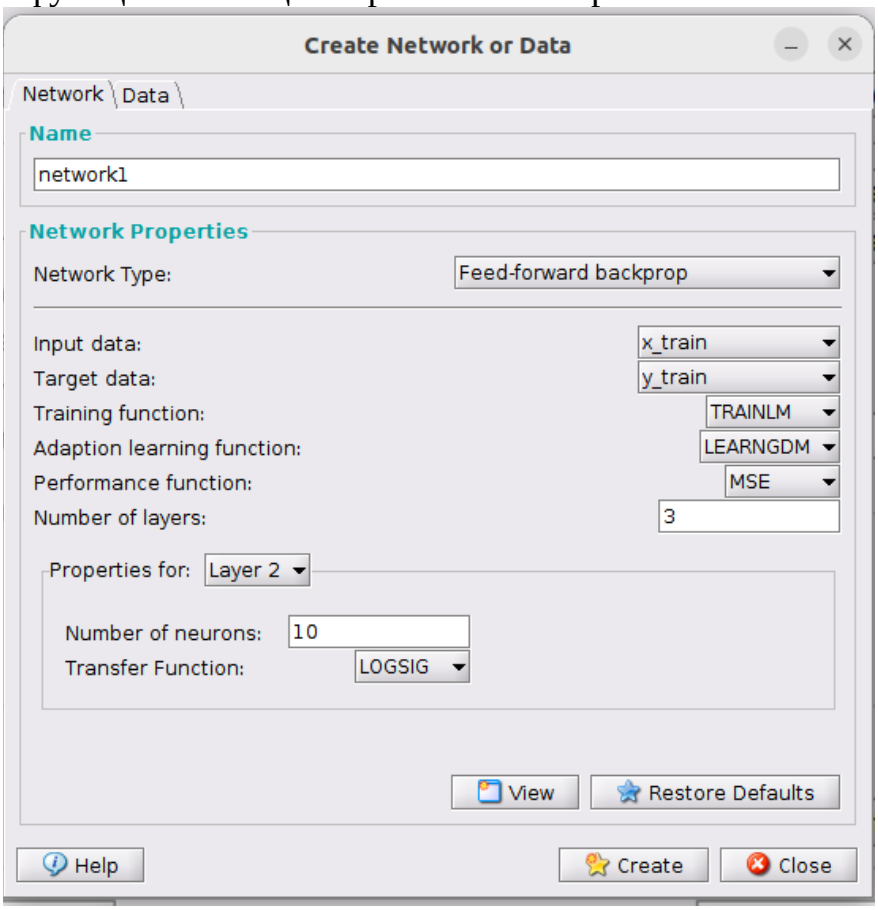

```
>> x_train.x_train(1) = 0;
>> x_train(1,:) = [];
>> y_train(1,:) = [];
>> nntool
Warning: nntool will be removed in a future release. Use nnstart instead.
> In nntool (line 17)
```

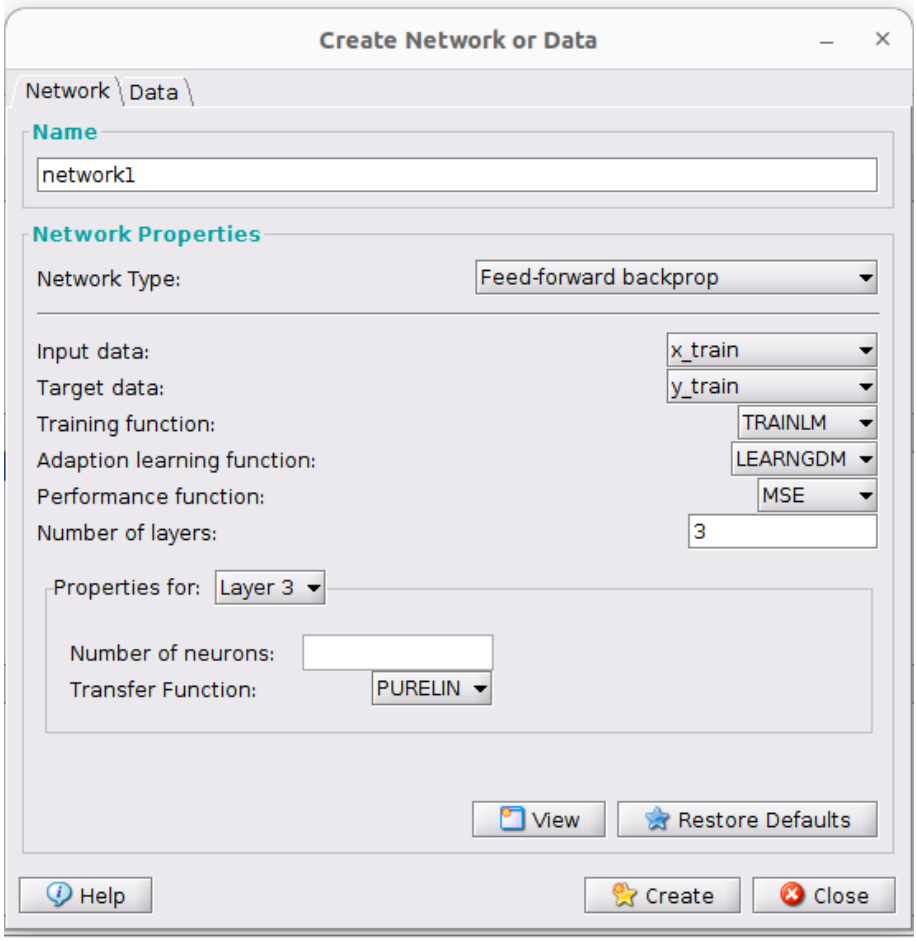
У вікні NNTool обираємо "New" для створення нової нейромережі.

Виберіть тип нейромережі "Feedforward", що є найпоширенішим типом нейромереж. Ви можете вибрати кількість прихованих шарів (у вашому випадку один прихований шар) та кількість нейронів у кожному прихованому шарі (у вашому випадку 10 нейронів).

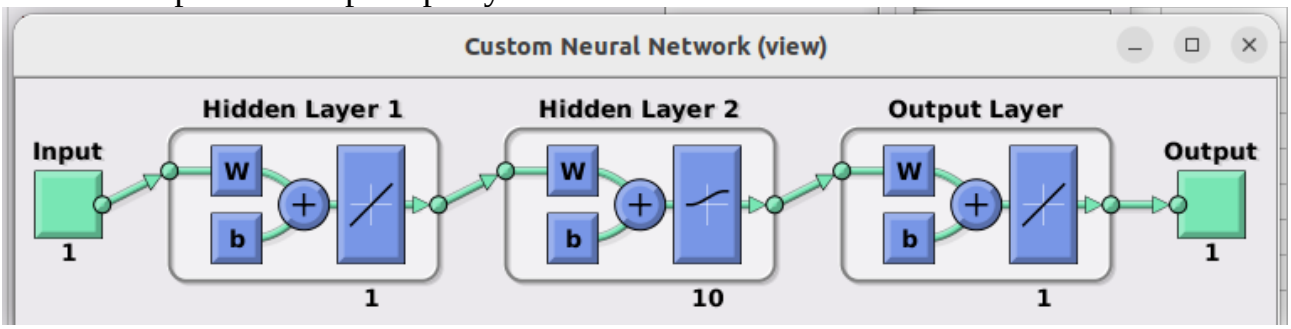


Обираємо функцію активації для кожного прихованого шару "logsig" (сігмоїда) для функцій активації в прихованих шарах.

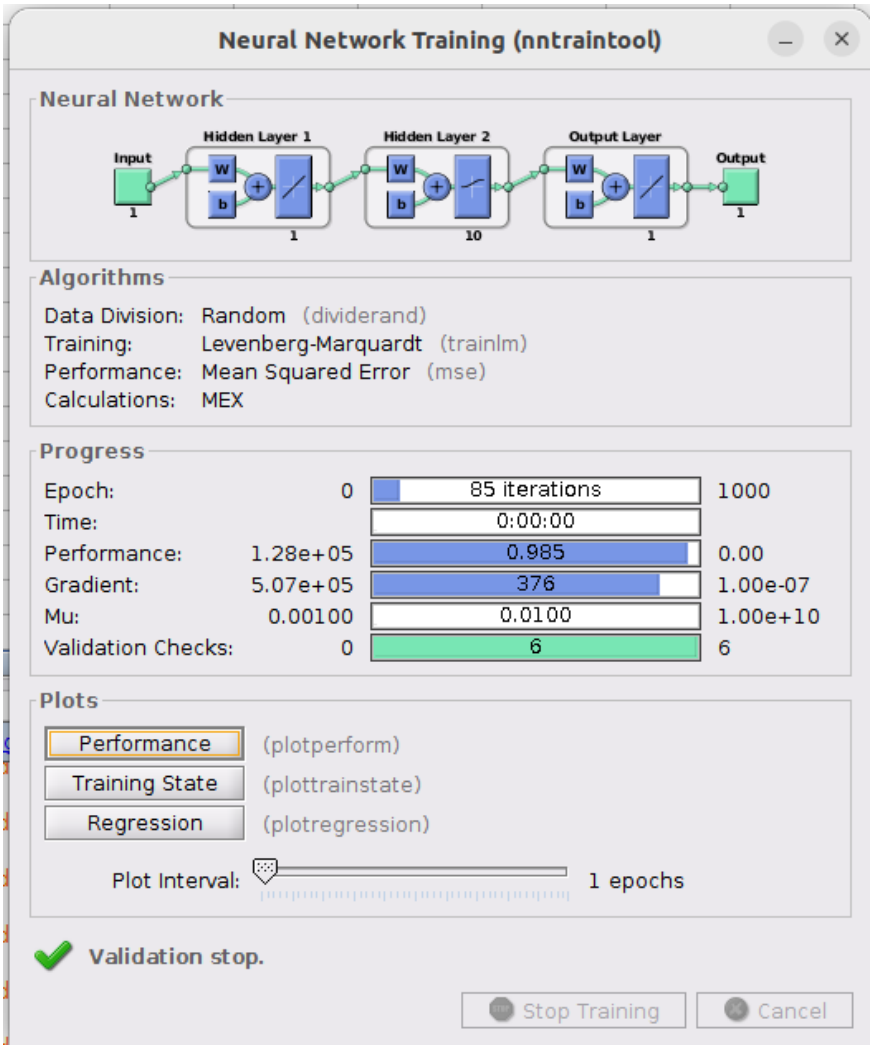




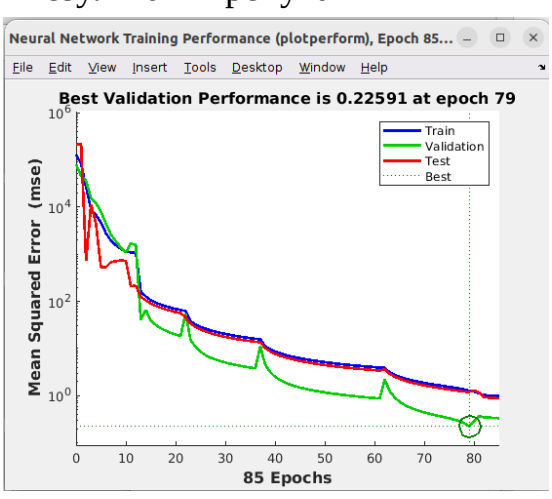
Створюємо нейромережу

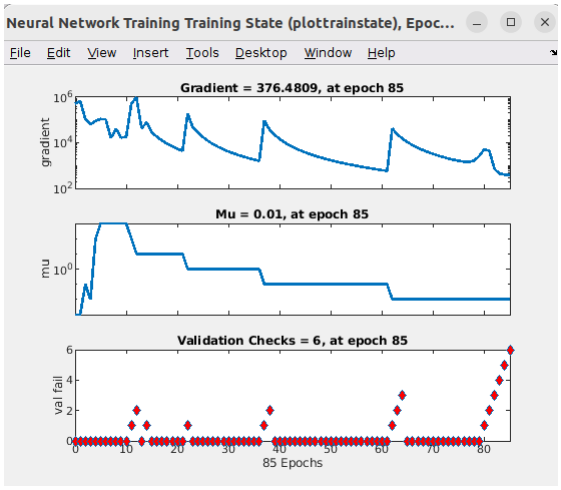


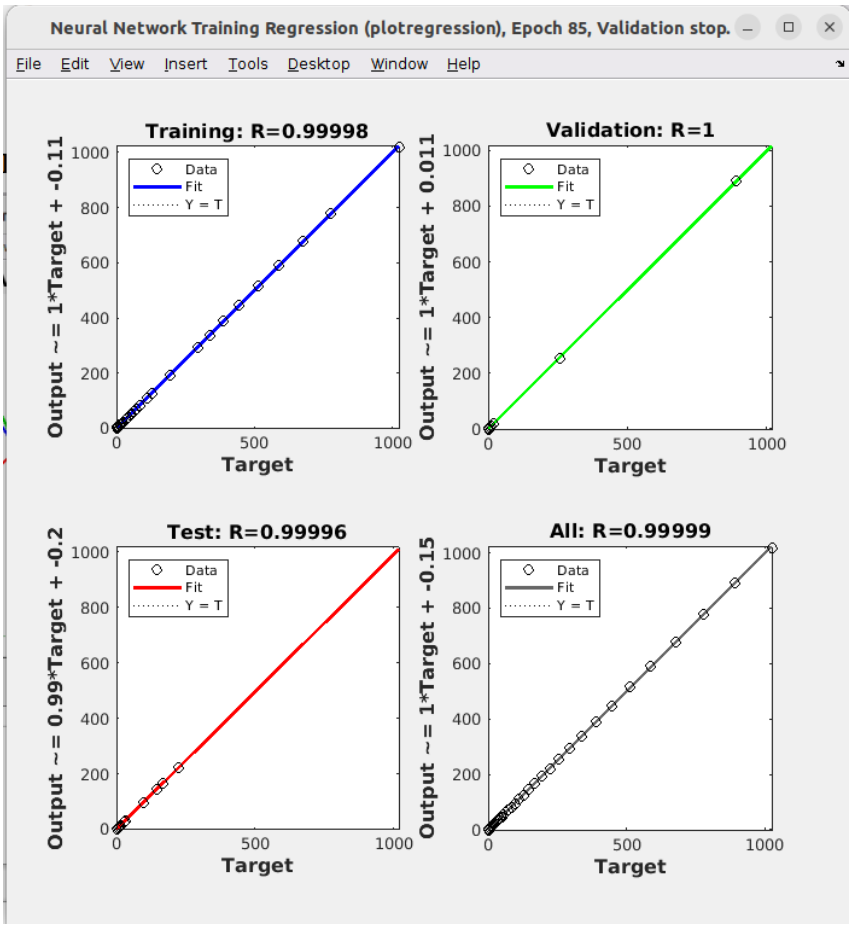
Запускаємо процес тренування



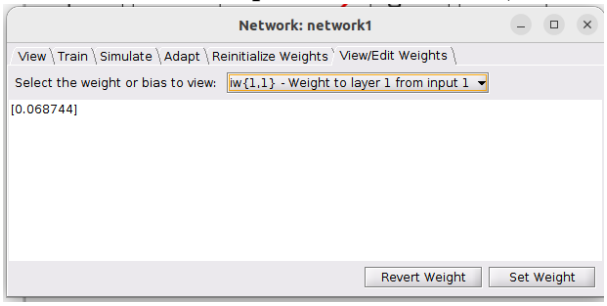
Результати тренування

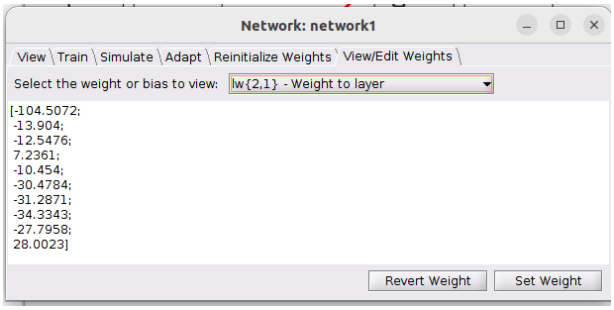


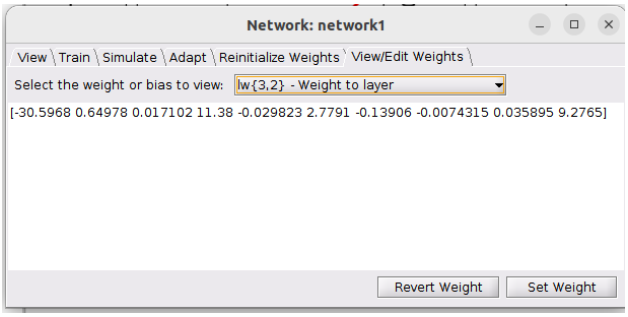


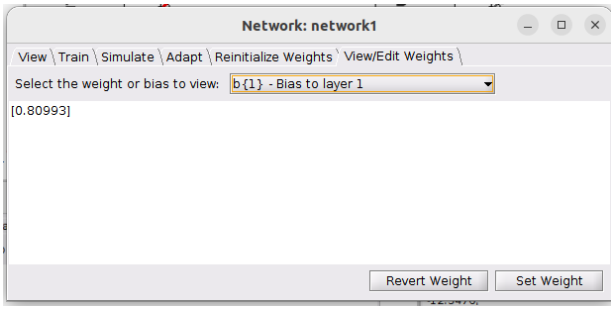


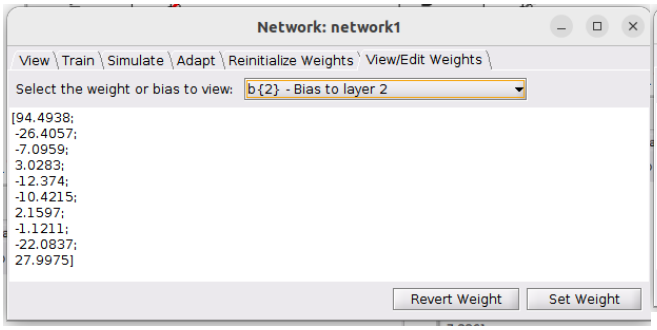
Вагові коефіцієнти та зміщення

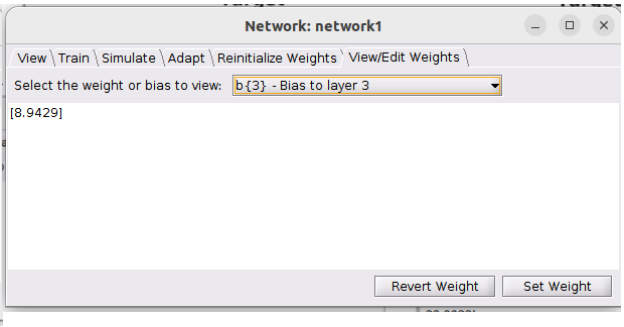




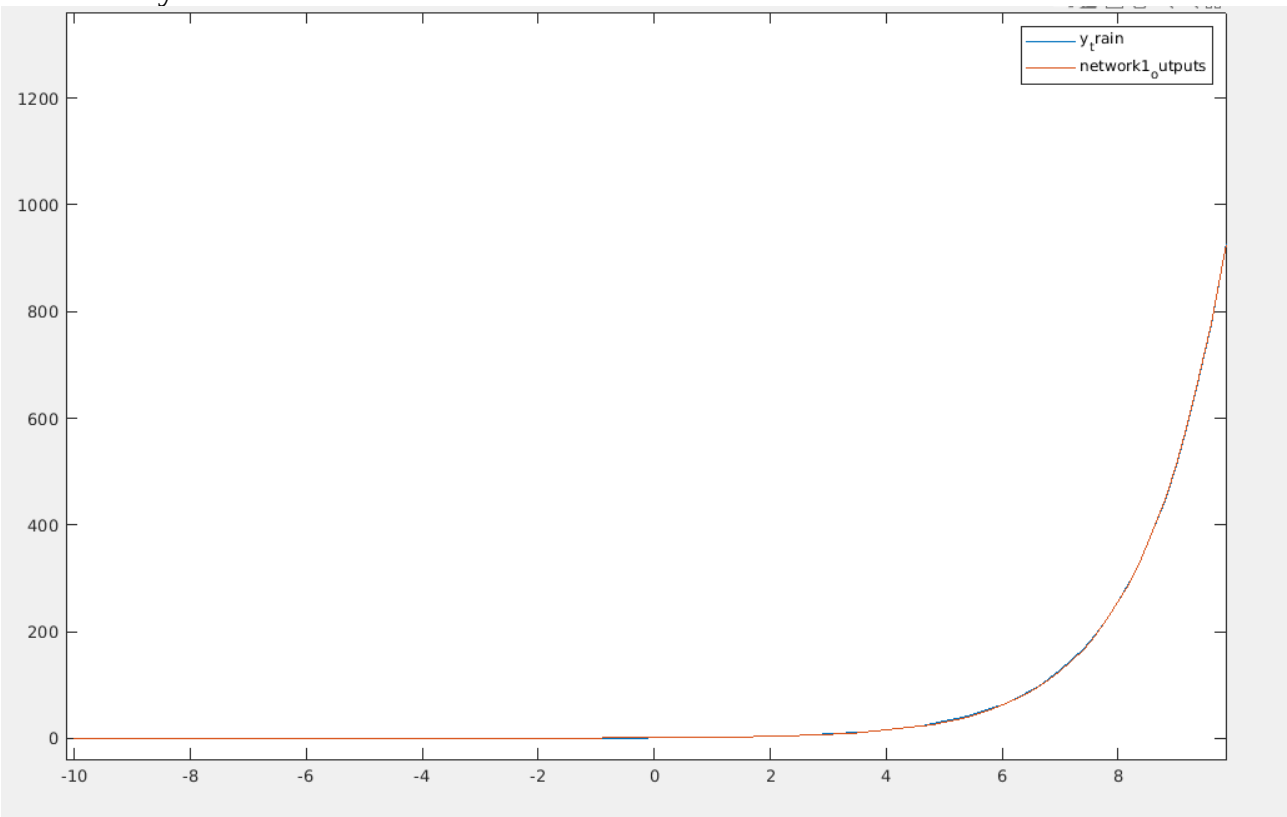


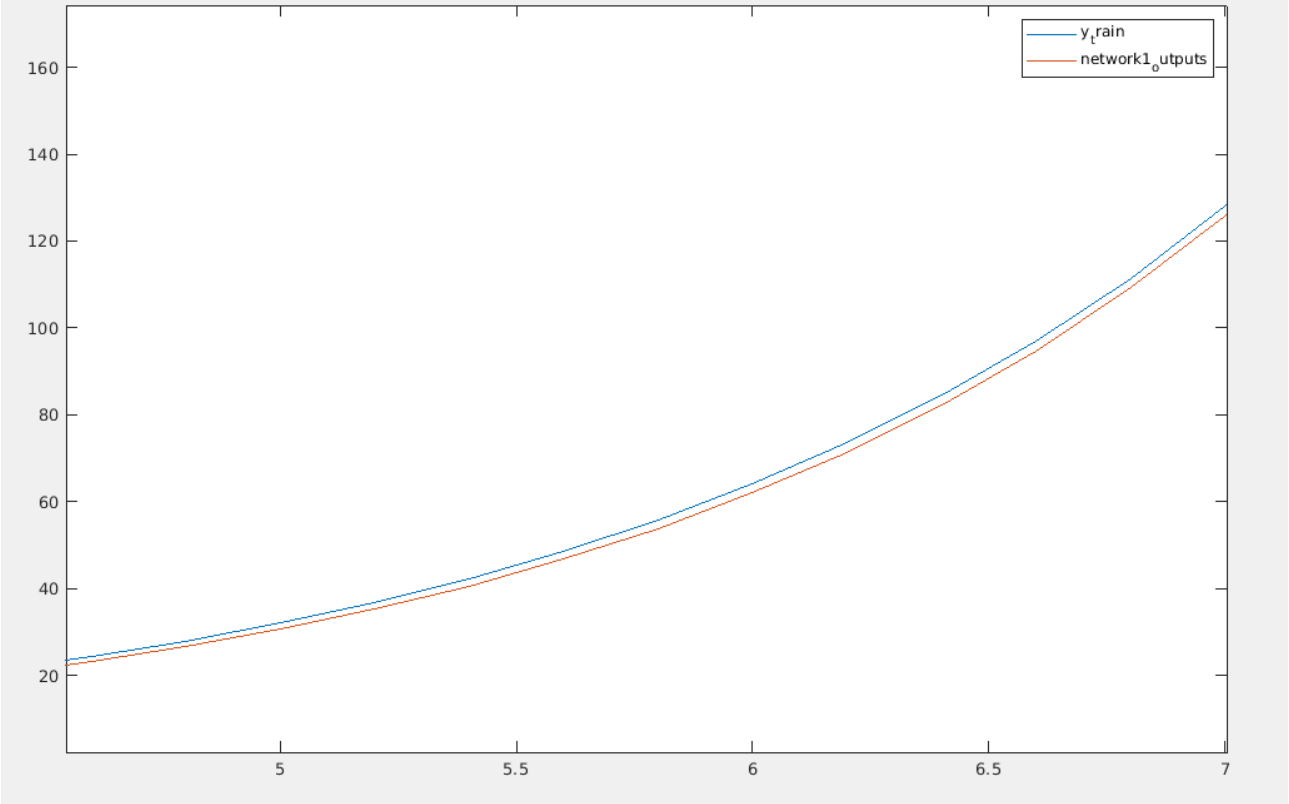


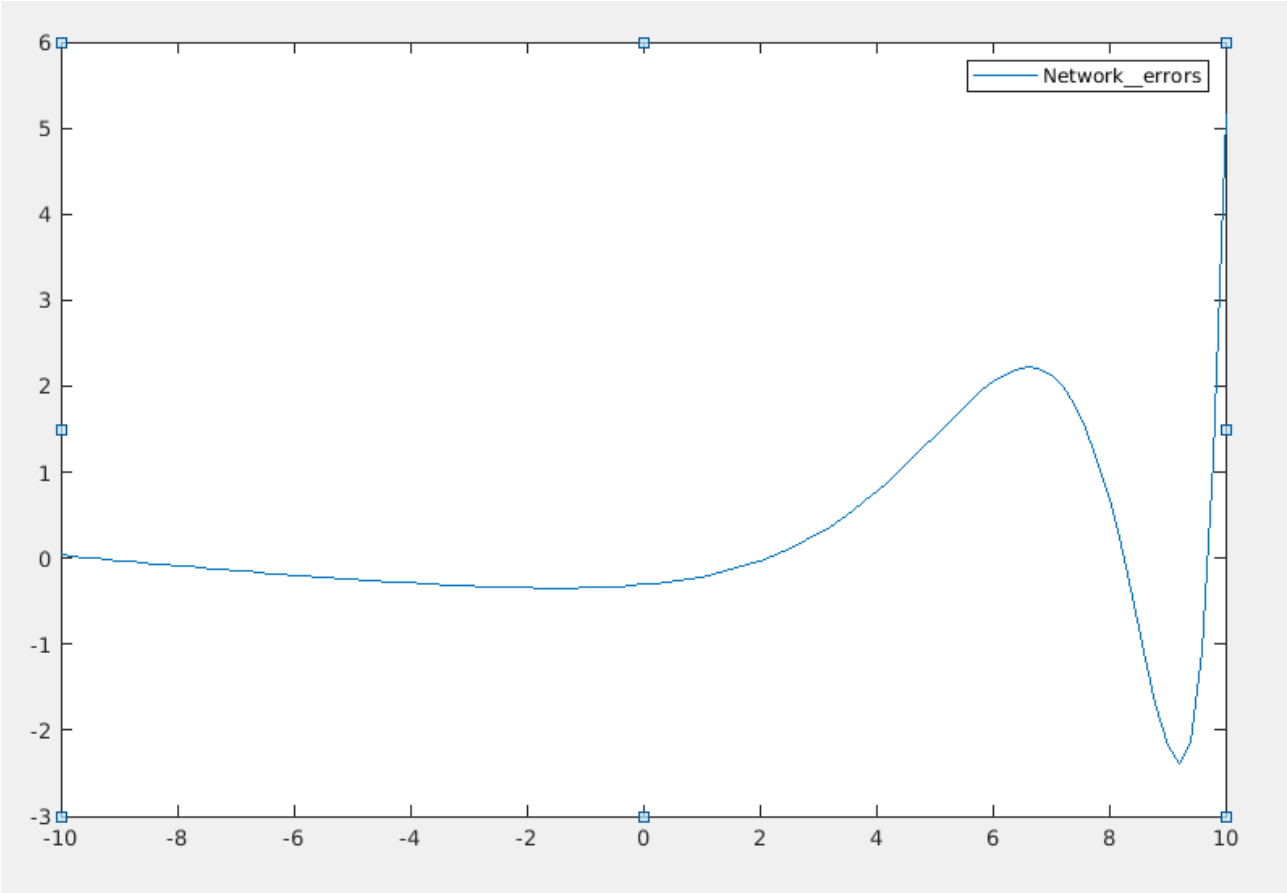




Запускаємо симуляцію мережі, потім експортуємо дані навчання і будуємо графіки функції $\mathbf{Y=2^x}$ результати апроксимації функції нейромережею та похибку.






Network__errors
6
5
4
3
2
1
0
-1
-2
-3
-10
-8
-6
-4
-2
0
2
4
6
8
10

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

```

```python

# 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

```

```python
# Створюємо модель нейромережі
# Вхідний шар з одним нейроном
input_layer = Input(shape=(1,))
# Прихований шар з 10 нейронами та функцією активації сігмоїдального типу
hidden_layer = Dense(10, activation='sigmoid')(input_layer)
# Вихідний шар з одним нейроном
output_layer = Dense(1)(hidden_layer)
```

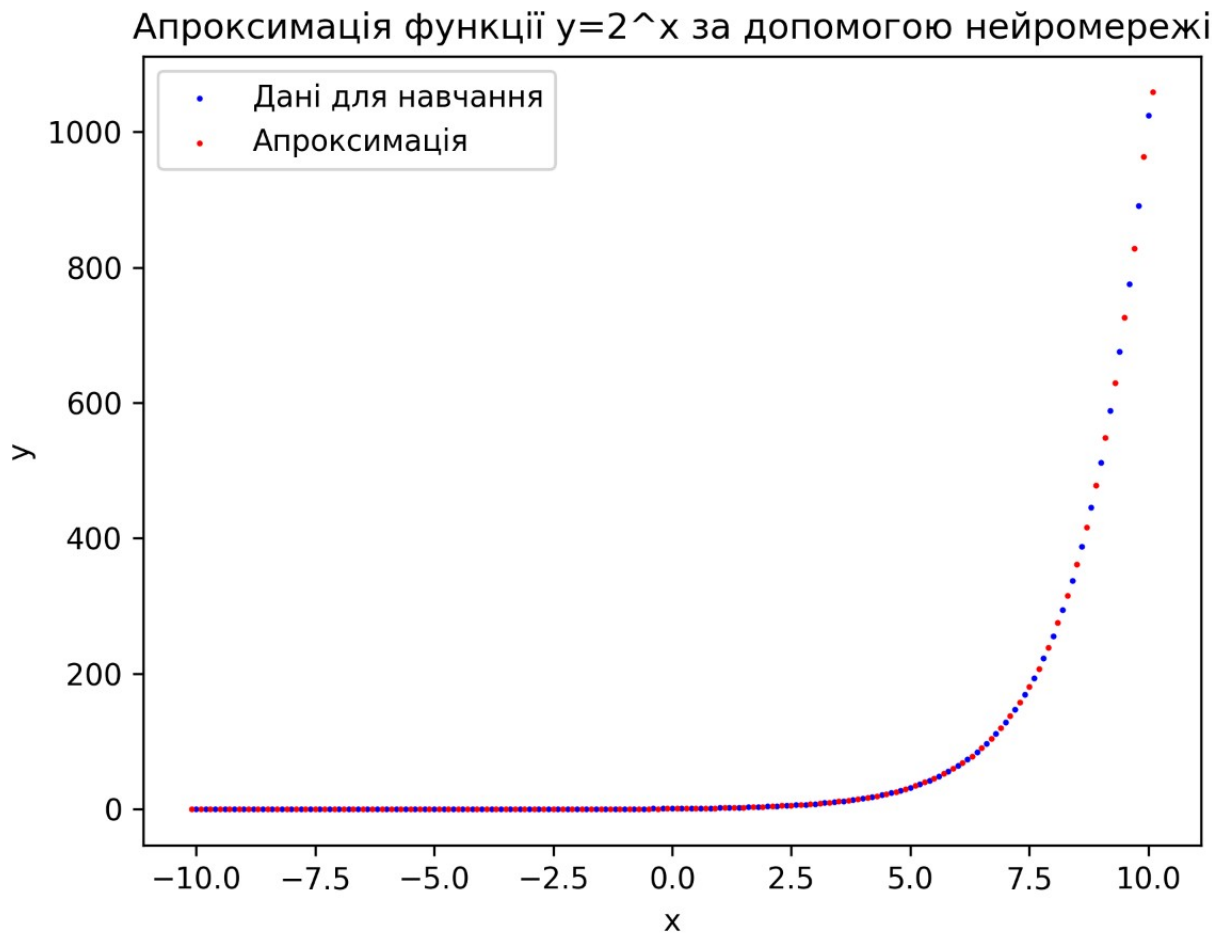
```python
# Створюємо модель нейромережі
model = tf.keras.models.Model(inputs=input_layer, outputs=output_layer)

# Виводимо опис моделі
model.summary()

```

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

 dense (Dense)               (None, 10)                20

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 31
Trainable params: 31
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
model.fit(x = x_train, y = y_train, epochs=10000, batch_size=101)

```

```
Epoch 9998/10000
1/1 [==============================] - 0s 3ms/step - loss: 0.0511
Epoch 9999/10000
1/1 [==============================] - 0s 4ms/step - loss: 0.0511
Epoch 10000/10000
1/1 [==============================] - 0s 5ms/step - loss: 0.0510
4/4 [==============================] - 0s 1ms/step
```

```
# Проводимо апроксимацію на тестових даних навчання
# Виконуємо передбачення за допомогою моделі
x_test = np.arange(-10.1,10.2,0.2)
y_test = model.predict(x_test)

# Виводимо результати
plt.scatter(x_train, y_train, color='blue', label='Дані для навчання')
plt.plot(x_test, y_test, color='red', label='Апроксимація')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Апроксимація функції y=2^x за допомогою нейромережі')
plt.savefig(fname='aproximation.png', dpi=300)

```



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-Lab5**.**