МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

З В І Т

Лабораторна робота №2
з дисципліни
«Сучасні методи та моделі інтелектуальних систем керування»

Виконавець:

аспірант групи АКІТР-23-1а Косей М.П.

Керівник:

викладач Тиханський М. П.

2024

Лабораторна робота №2

Тема: Багатошарова нейронна мережа прямого поширення

Мета: Одержати основні навички побудови моделей штучних нейромереж

ХІД РОБОТИ

1) Ознайомитись з теоретичними відомостями до лабораторної роботи

БАГАТОШАРОВІ НЕЙРОННІ МЕРЕЖІ

Багатошарова нейронна мережа прямого поширення (**MLP — Multilayer Perceptron**) є одним із базових типів штучних нейронних мереж. Вона складається з кількох шарів нейронів, з'єднаних таким чином, що інформація проходить тільки в одному напрямку — від вхідного шару до вихідного, через один або більше прихованих шарів.

MLP складається з:

1. Вхідного шару: отримує дані на вході.
2. Прихованих шарів: виконують нелінійні перетворення вхідних даних.
3. Вихідного шару: генерує остаточний результат (наприклад, класифікація або передбачення).

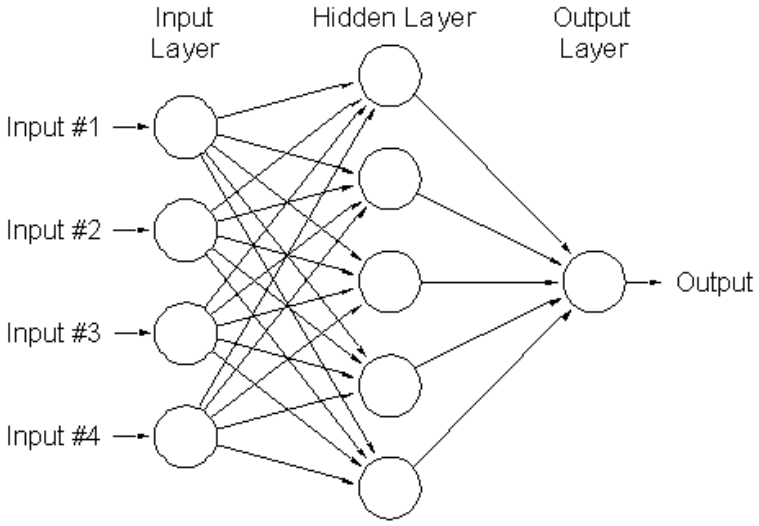


Рисунок 2.1 - Багатошарова нейронна мережа прямого поширення (MLP — Multilayer Perceptron)

Кожен нейрон в шарі з'єднаний з усіма нейронами попереднього шару, що робить мережу повнозв'язаною.

Один з методів навчання нейромереж - зворотнє поширення (**Backpropagation**) - це один з найпоширеніших методів навчання нейронних мереж, який використовується для тренування штучних нейронних мереж з вчителем (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.

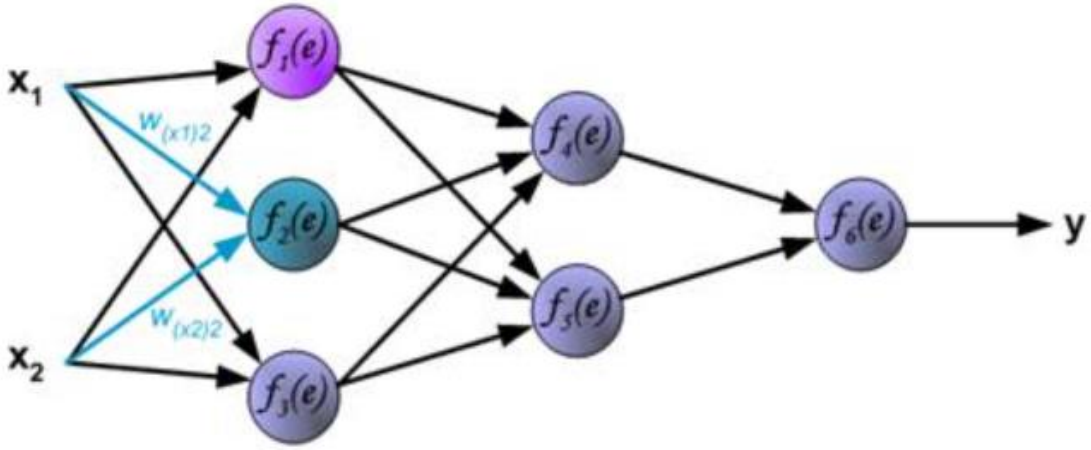
Основна ідея за методом зворотного поширення полягає в пошуку градієнту функції втрати (loss function) відносно ваг мережі, і використання

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.

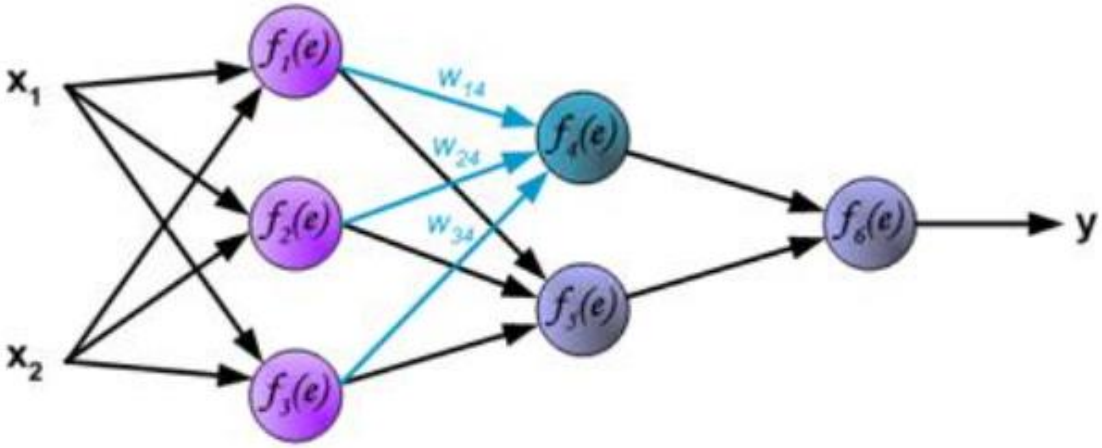
Процес навчання нейромережі зворотнім поширенням можна узагальнити наступними кроками:

Ініціалізація ваг мережі:

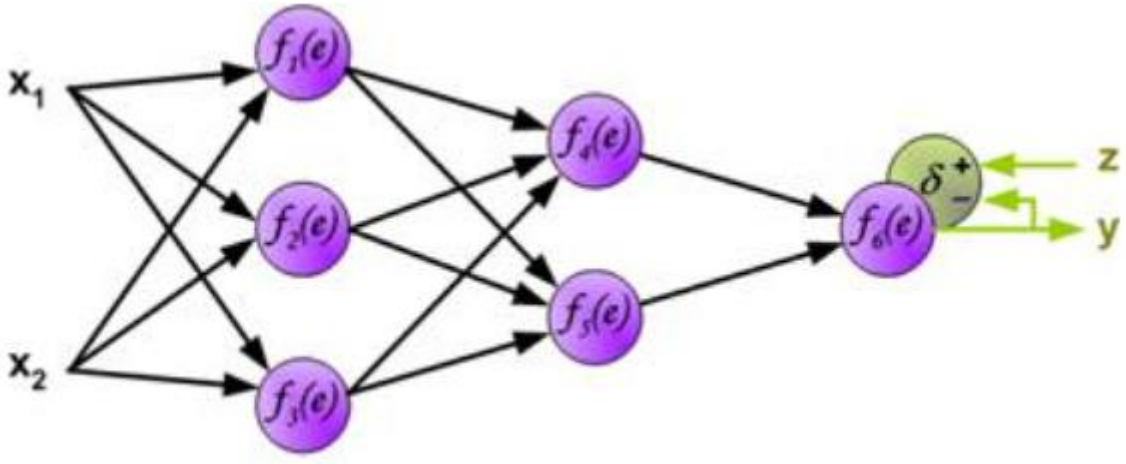
1) Ваги мережі ініціалізуються випадковими значеннями або за допомогою



іншого методу.

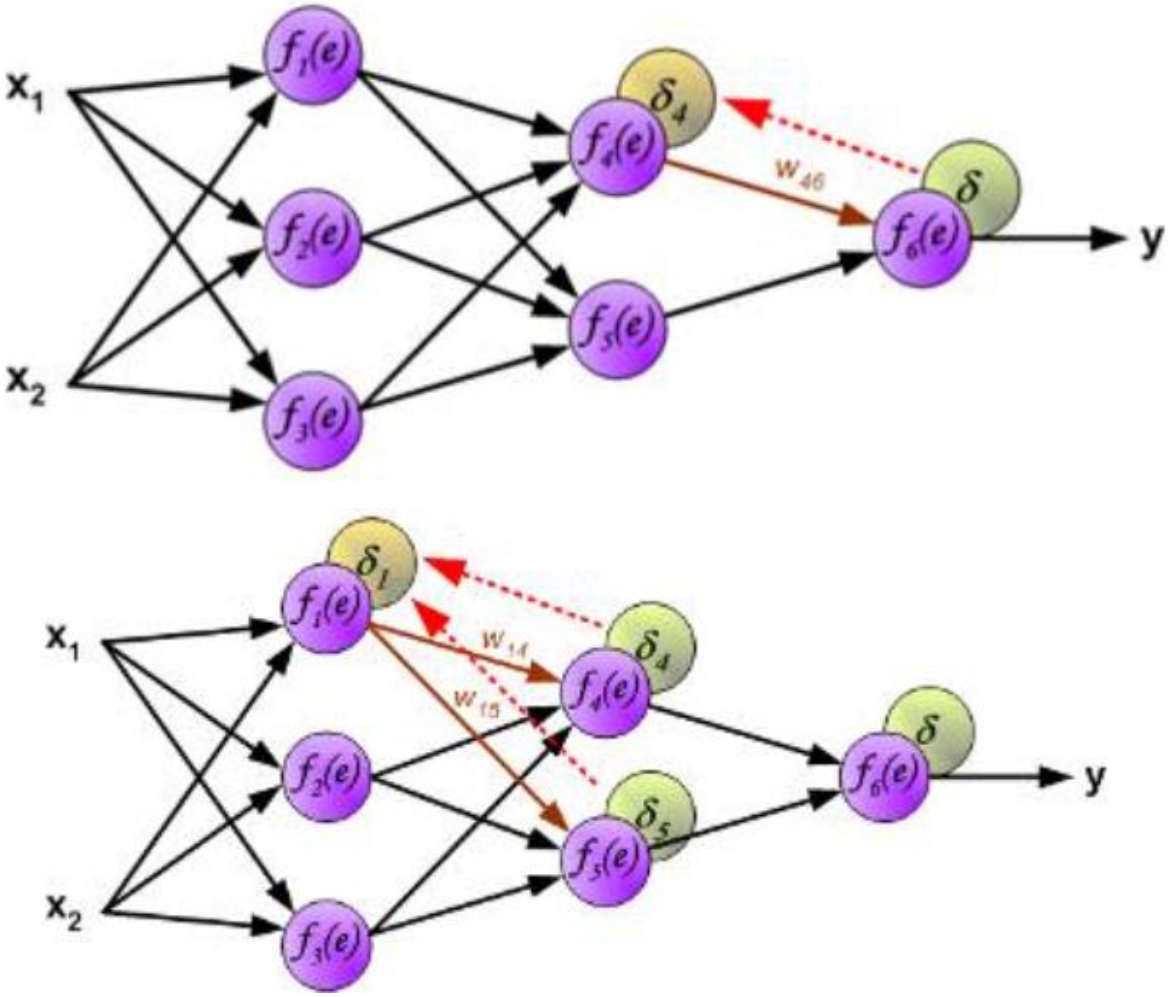


2) Прямий прохід (forward pass): Вхідні дані проходять крізь мережу від вхідного до вихідного шару, розраховуючи вихідні значення мережі.
3) Розрахунок помилок (loss calculation): Розраховується значення функції

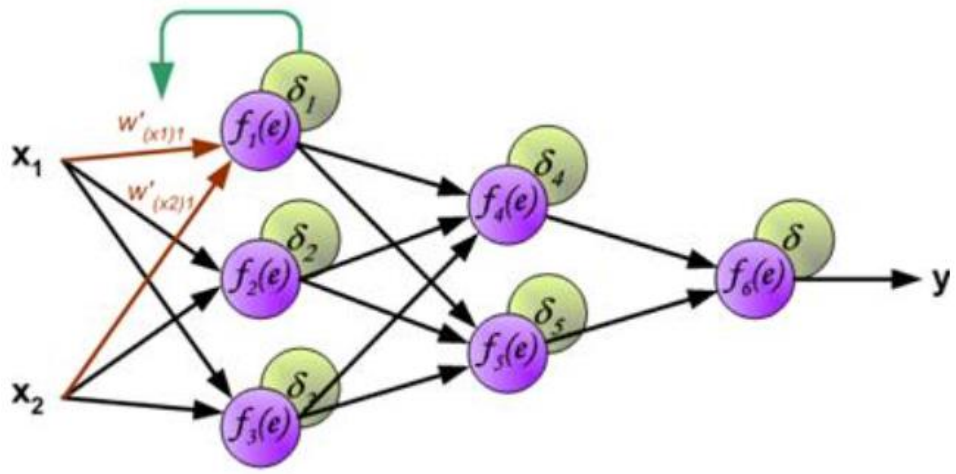


втрати, яка відображає різницю між прогнозованими виходами мережі та цільовими виходами (мітками) у навчальному наборі даних.

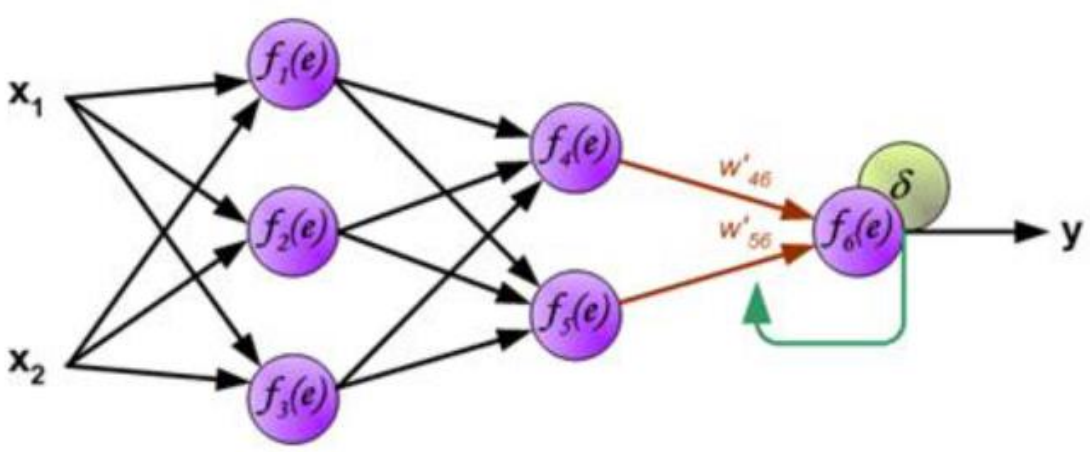
4) Зворотнє поширення градієнту (gradient backpropagation): За допомогою ланцюгового правила (chain rule) розраховується градієнт функції втрати відносно вихідних значень мережі. Градієнт використовується для розрахунку градієнтів ваг на всіх рівнях мережі, починаючи з вихідного шару і рухаючись назад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

2) Практична частина – моделювання багатошарових нейронних мереж прямого поширення (MLP — Multilayer Perceptron).

Для моделювання нейронних мереж також можна використовувати наступні інструменти:

1) **MATLAB** – популярне середовище для математичних обчислень, яке має вбудовані інструменти для побудови та навчання нейронних мереж. MATLAB зручний для наукових досліджень, а його модулі для нейронних мереж полегшують процес моделювання без необхідності глибокого занурення в програмування.

2) **Python** – універсальна мова програмування з багатим набором бібліотек для роботи з нейронними мережами (наприклад, TensorFlow, PyTorch, Keras). Python є найпоширенішою мовою для машинного навчання завдяки простоті та багатофункціональності.

3) **Kaggle** – платформа для змагань з машинного навчання, де можна знайти готові набори даних, різноманітні приклади використання нейронних мереж, а також навчальні матеріали. Kaggle також має вбудоване середовище для запуску Jupyter Notebook, що спрощує експерименти з моделями.

4) **Jupyter Lab** – інтегроване середовище для обчислень, яке дозволяє створювати та виконувати Python-код у вигляді нотаток. Ідеально підходить для моделювання нейронних мереж, тестування моделей, аналізу результатів та візуалізації.

Використовуємо мову програмування **Python** та проект **JupyterLab**.

а) Створити двошарову нейронну мережу прямого поширення з сигмоїдальною активаційною функцією 'logsig' з двома входами. Навчити штучну нейронну мережу обчислювати значення функції на основі методу зворотного поширення помилки.

Моделювання багатошарових нейронних мереж прямого поширення (MLP — Multilayer Perceptron)

[36]:
```
# Імпортуємо необхідні бібліотеки
import os
import tensorflow as tf
# Вимкнути використання GPU
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
```

[37]:
```
# Вхідні дані
X = np.array([[1, -2],
              [1, -1],
              [3, -2],
              [3, -1]])

# Вихідні дані
y = np.array([2, 1, 0, -3])
```

[38]:
```
# Створення моделі
model = Sequential()

# Прихований шар: 5 нейронів, sigmoid, вхід з 2 ознак
model.add(Dense(5, input_dim=2, activation='sigmoid'))

# Вихідний шар: 1 нейрон з лінійною активацією
model.add(Dense(1, activation='linear'))

# Компіляція моделі для регресії
optimizer = Adam(learning_rate=0.3)
model.compile(optimizer=optimizer,
              loss='mean_squared_error',
              metrics=['mae'])
```

[48]:
```
# Навчання моделі
model.fit(X, y, epochs=100, verbose=2)
```

```
Epoch 1/100
1/1 - 1s - 706ms/step - loss: 5.4269 - mae: 2.1936
Epoch 2/100
1/1 - 0s - 25ms/step - loss: 3.8075 - mae: 1.5634
Epoch 3/100
1/1 - 0s - 28ms/step - loss: 3.8056 - mae: 1.4290
Epoch 4/100
1/1 - 0s - 27ms/step - loss: 3.5200 - mae: 1.3396
Epoch 5/100
1/1 - 0s - 29ms/step - loss: 3.1779 - mae: 1.2933
Epoch 6/100
1/1 - 0s - 30ms/step - loss: 2.7979 - mae: 1.2330
Epoch 7/100
1/1 - 0s - 29ms/step - loss: 2.2873 - mae: 1.1339
Epoch 8/100
1/1 - 0s - 29ms/step - loss: 1.6863 - mae: 1.1072
Epoch 9/100
1/1 - 0s - 30ms/step - loss: 1.2347 - mae: 1.0638
Epoch 10/100
1/1 - 0s - 29ms/step - loss: 0.9682 - mae: 0.9520
Epoch 11/100
1/1 - 0s - 29ms/step - loss: 0.5749 - mae: 0.6799
```

```
1/1 - 0s - 29ms/step - loss: 1.5084e-05 - mae: 0.0037
Epoch 98/100
1/1 - 0s - 33ms/step - loss: 2.3148e-05 - mae: 0.0041
Epoch 99/100
1/1 - 0s - 32ms/step - loss: 4.8790e-05 - mae: 0.0063
Epoch 100/100
1/1 - 0s - 27ms/step - loss: 8.4788e-06 - mae: 0.0022
```

[48]:
```
<keras.src.callbacks.history.History at 0x789e93fd03e0>
```

[49]:
```
# Оцінка моделі
loss, mae = model.evaluate(X, y, verbose=0)
print(f'Втрати (MSE): {loss}, MAE: {mae}')

# Прогнозування
predictions = model.predict(X)

# Виведення навчальних значень та прогнозів
print(f"{'X1':>4} {'X2':>4} {'Actual':>10} {'Prediction':>10}")
print("-" * 34)
for x, actual, pred in zip(X, y, predictions):
    print(f"{x[0].item():>4} {x[1].item():>4} {actual:>10} {pred.item():>10.4f}")
```

```
Втрати (MSE): 2.7240774215897545e-05, MAE: 0.0037167370319366455
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
  X1   X2     Actual Prediction
----------------------------------
   1   -2          2     1.9973
   1   -1          1     0.9995
   3   -2          0    -0.0017
   3   -1         -3    -3.0099
```

б) Створити двошарову нейронну мережу прямого поширення з лінійною активаційною функцією 'purelin' з двома входами. Навчити штучну нейронну мережу обчислювати значення функції на основі методу зворотного поширення помилки.

:
```
# Створення моделі
model = Sequential()

# Прихований шар: 5 нейронів,  вхід з 2 ознак
model.add(Dense(5, input_dim=2, activation='linear'))

# Вихідний шар: 1 нейрон з лінійною активацією
model.add(Dense(1, activation='linear'))

# Компіляція моделі для регресії
optimizer = Adam(learning_rate=0.3)
model.compile(optimizer=optimizer,
              loss='mean_squared_error',
              metrics=['mae'])
```

:
```
# Навчання моделі
model.fit(X, y, epochs=100, verbose=2)
```

```
Epoch 1/100
1/1 - 1s - 823ms/step - loss: 14.3538 - mae: 3.2282
Epoch 2/100
1/1 - 0s - 25ms/step - loss: 12.1547 - mae: 3.2047
Epoch 3/100
1/1 - 0s - 29ms/step - loss: 10.9433 - mae: 2.9648
Epoch 4/100
1/1 - 0s - 26ms/step - loss: 4.5123 - mae: 2.0318
Epoch 5/100
```

```
Epoch 98/100
1/1 - 0s - 28ms/step - loss: 0.2500 - mae: 0.5000
Epoch 99/100
1/1 - 0s - 30ms/step - loss: 0.2501 - mae: 0.5000
Epoch 100/100
1/1 - 0s - 29ms/step - loss: 0.2502 - mae: 0.5000
<keras.src.callbacks.history.History at 0x73bb675b6ba0>
```

```python
# Оцінка моделі
loss, mae = model.evaluate(X, y, verbose=0)
print(f'Втрати (MSE): {loss}, MAE: {mae}')

# Прогнозування
predictions = model.predict(X)

# Виведення навчальних значень та прогнозів
print(f"{'X1':>4} {'X2':>4} {'Actual':>10} {'Prediction':>10}")
print("-" * 34)
for x, actual, pred in zip(X, y, predictions):
    print(f"{x[0].item():>4} {x[1].item():>4} {actual:>10} {pred.item():>10.4f}")
```

```
Втрати (MSE): 0.2502131462097168, MAE: 0.4999999701976776
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
  X1   X2     Actual Prediction
-----------------------------------
   1   -2          2     2.4818
   1   -1          1     0.4935
   3   -2          0    -0.4915
   3   -1         -3    -2.4798
```

```
Якщо всі шари у  нейронній мережі використовують лінійні активаційні функції,
то вся модель стає еквівалентною звичайному лінійному виразу, незалежно від кількості шарів.
Це означає, що така мережа не зможе моделювати нелінійні залежності між вхідними та вихідними даними.
Вся багатошарова мережа еквівалентна одному лінійному нейрону (тобто лінійній регресії).
Тому додавання додаткових лінійних шарів у такій конфігурації не приносить жодної нової користі
– всю роботу можна виконати одним лінійним нейроном.
```

в) Обчислити 10 значень функції sin(x1) + cos(x2) - cos(x3) по заданих значення аргументів x1=1/2, x2=1/5, x3=1/4 та їх варіаціях з кроком ±1.

```python
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
from tabulate import tabulate

class CustomEarlyStopping(Callback):
    """Кастомний колбек для зупинки, коли MSE < 1e-5."""
    def on_epoch_end(self, epoch, logs=None):
        current_loss = logs.get("loss")
        if current_loss is not None and current_loss < 1e-5:
            print(f"\nЗупинка на {epoch+1}-й епосі: MSE < 1e-5")
            self.model.stop_training = True


# Функція для обчислення цільових значень
def target_function(x1, x2, x3):
    return np.sin(x1) + np.cos(x2) - np.cos(x3)

# Генеруємо дані
x1_values = np.linspace(0.5 - 5, 0.5 + 4, 10)
x2_values = np.linspace(0.2 - 5, 0.2 + 4, 10)
x3_values = np.linspace(0.25 - 5, 0.25 + 4, 10)
X = np.column_stack((x1_values, x2_values, x3_values))
Y = np.array([target_function(x1, x2, x3) for x1, x2, x3 in X])
```

```python
# Функція для створення та навчання моделі
def build_and_train_model(activation, optimizer_class, lr=0.03):
    optimizer = optimizer_class(learning_rate=lr)

    # Створення моделі
    model = Sequential()
    model.add(Dense(5, input_dim=3, activation=activation))  # Прихований шар
    model.add(Dense(1, activation='linear'))  # Вихідний шар

    early_stopping = CustomEarlyStopping()
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    history = model.fit(X, Y, epochs=2000, verbose=0, callbacks=[early_stopping])

    # Отримання втрати та кількості епох
    loss = history.history['loss'][-1]
    epochs = len(history.history['loss'])
    predictions = model.predict(X)

    return loss, epochs, predictions
```

```python
# Списки активаційних функцій та оптимізаторів
activations = ['tanh', 'relu', 'sigmoid']
optimizer_classes = [Adam, SGD, RMSprop]

# Збереження результатів
results = []

# Збір даних у циклі
for activation in activations:
    for optimizer_class in optimizer_classes:
        loss, epochs, _ = build_and_train_model(activation, optimizer_class)
        optimizer_name = optimizer_class.__name__
        results.append((activation, optimizer_name, f"{loss:.6f}", epochs))

# Вивід таблиці
headers = ["Activation", "Optimizer", "Loss", "Epochs"]
print(tabulate(results, headers=headers, tablefmt="pretty"))

# Пошук найкращої комбінації
best_result = min(results, key=lambda x: float(x[2]))
print("\nНайкраща комбінація за похибкою:")
print(f"Активація: {best_result[0]}, Оптимізатор: {best_result[1]}, "
      f"Втрата: {best_result[2]}, Епохи: {best_result[3]}")
```

Activation	Optimizer	Loss	Epochs
tanh	Adam	0.000010	363
tanh	SGD	0.156832	2000
tanh	RMSprop	0.005312	2000
relu	Adam	0.007204	2000
relu	SGD	0.431553	2000
relu	RMSprop	0.070482	2000
sigmoid	Adam	0.000010	744
sigmoid	SGD	0.091660	2000
sigmoid	RMSprop	0.011379	2000

Найкраща комбінація за похибкою:
Активація: tanh, Оптимізатор: Adam, Втрата: 0.000010, Епохи: 363

ВИСНОВКИ

В результаті виконаної лабораторної роботи розроблені моделі багатошарових нейронних мереж прямого поширення.

Усі матеріали викладенні у репозіторії GitHub, за посиланням https://github.com/Max11mus/-LAB2-Modern-Methods-and-Models-of-Intelligent-Control-Systems**.**