**Білет №3**

1 Dataset/DataFrame. Конекціоністський підхід до побудови систем ШІ.

Штучний нейрон. Функція активації. Структура ШНМ.

Dataset/DataFrame — важливі для зберігання і обробки даних у системах штучного інтелекту, особливо за допомогою бібліотеки pandas в Python.

Штучний нейрон і функції активації — основні компоненти нейронних мереж. Штучні нейрони імітують роботу біологічних нейронів, використовуючи ваги для обробки вхідних сигналів і функції активації для передачі результатів.

Структура ІНММ — інтеграція нейронних мереж і мікроелектроніки для підвищення ефективності обчислень, що наслідує принципи роботи людського мозку.

2 Мережі довгої короткочасної пам’яті. Сфери використання. Принцип

роботи функції уваги Луонга.

Мережі довгої короткочасної пам’яті (LSTM):

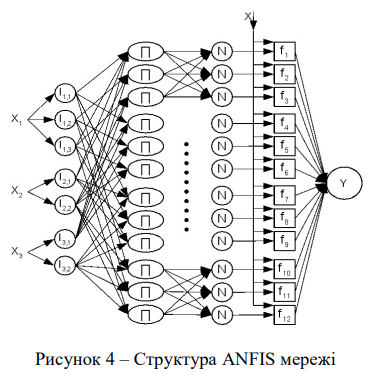
Принцип роботи: LSTM — це спеціалізований тип рекурентної нейронної мережі, здатний вчитися від довготривалих залежностей у даних. Вони містять "ворота" (вхідні, забування, вихідні), які регулюють потік інформації, дозволяючи мережі зберігати або відкидати інформацію.

Сфери використання: LSTM широко використовуються в областях, що вимагають аналізу послідовних даних, таких як мовне розпізнавання, генерація тексту, прогнозування часових рядів, і навіть у музичній композиції.

Функція уваги Луонга:

Принцип роботи: Функція уваги Луонга — це механізм у моделях машинного перекладу, що дозволяє моделі динамічно концентруватися на різних частинах вхідної послідовності під час генерації виходу. Це покращує якість перекладу, оскільки модель може вибірково зосереджуватися на інформації, що є найбільш релевантною для поточного слова чи фрази, яку генерує.

3 Блок обговорення. Тематика згідно з зображеним нижче



На зображенні ми бачимо структуру мережі ANFIS (Adaptive Neuro-Fuzzy Inference System). ANFIS — це форма адаптивної системи, яка поєднує принципи нечіткої логіки з нейронними мережами, забезпечуючи можливість навчання системи на основі даних. Ось як вона працює:

1. Вхідні дані (x1, x2, x3, ...) - це змінні, які вводяться в систему для обробки.
2. Нейрони вхідного шару (позначені як N) - ці нейрони безпосередньо приймають вхідні сигнали та передають їх на наступний шар без змін.
3. Функції приналежності (f1, f2, ... f12) - використовуються для оцінки ступеня, з яким вхідні дані відповідають певним нечітким множинам; вони визначають, наскільки кожен вхідний параметр "підходить" до кожної категорії.
4. Правила (проміжний шар) - кожен вузол в цьому шарі виконує функцію, що враховує вхідні приналежності та об'єднує їх згідно з логікою нечітких правил.
5. Сумаційні нейрони (вихідні N) - агрегують виходи з правил і формують один або декілька виходів, що можуть бути в подальшому нормалізовані.
6. Вихід Y - кінцевий результат обробки, що являє собою виведене значення на основі вхідних даних та налаштувань правил.

Де використовується ANFIS: ANFIS широко застосовується у сферах, де потрібно управління складними системами, такими як робототехніка, прогнозування часових рядів, фінансове моделювання, медицина та екологічне моделювання. Система ефективна в задачах, де необхідно апроксимувати нелінійні функції високої складності, забезпечуючи високу точність та адаптивність до змін у вхідних даних.

1 Використовуючи python бібліотеки tensorflow і keras класифікатор на

основі нейронної мережі. Для навчання використовуйте датасет iris.data

(<https://archive.ics.uci.edu/ml/datasets/iris>)

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'class']

iris\_data = pd.read\_csv(url, names=columns)

X = iris\_data.iloc[:, :-1].values

y = iris\_data.iloc[:, -1].values

encoder = LabelEncoder()

y\_encoded = encoder.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

model = Sequential([

    Dense(10, activation='relu', input\_shape=(4,)),

    Dense(10, activation='relu'),

    Dense(3, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(X\_train\_scaled, y\_train, epochs=100, batch\_size=5)

loss, accuracy = model.evaluate(X\_test\_scaled, y\_test)

print(f'Test accuracy: {accuracy:.3f}')

