**ЛАБОРАТОРНА РОБОТА № 8**

***Тема*:** «Ресурси Keras. TensorFlow. Навчання лінійної регресії».

***Мета*:** Дослідження ресурсу Keras і TensorFlow. Застосування TensorFlow.

Без налаштування та встановлення так як використовуеться ***Google Collab*** – в якому влаштоване все для опрацювання завдання.

**Код:**

import tensorflow as tf

import numpy as np

# 1. Data generation

np.random.seed(42)  # For reproducibility

x\_data = np.random.uniform(0, 1, 1000).reshape(-1, 1)

y\_data = 2 \* x\_data + 1 + np.random.normal(0, 2, size=(1000, 1))

# 2. Creating data as tensors

X = tf.convert\_to\_tensor(x\_data, dtype=tf.float32)

y = tf.convert\_to\_tensor(y\_data, dtype=tf.float32)

# 3. Initializing model variables

k = tf.Variable(tf.random.normal([1], stddev=0.1), name='k')

b = tf.Variable(tf.zeros([1]), name='b')

# 4. Building the linear regression model

def model(X):

    return k \* X + b

# 5. Defining the loss function and optimizer

def compute\_loss(y\_true, y\_pred):

    return tf.reduce\_mean(tf.square(y\_true - y\_pred))

optimizer = tf.optimizers.SGD(learning\_rate=0.01)

# 6. Training the model

num\_epochs = 20000

batch\_size = 100

num\_batches = len(x\_data) // batch\_size

for epoch in range(num\_epochs):

    for i in range(num\_batches):

        # Selecting a random mini-batch

        batch\_indices = np.random.choice(len(x\_data), batch\_size)

        x\_batch = tf.convert\_to\_tensor(x\_data[batch\_indices], dtype=tf.float32)

        y\_batch = tf.convert\_to\_tensor(y\_data[batch\_indices], dtype=tf.float32)

        # Optimization within GradientTape

        with tf.GradientTape() as tape:

            y\_pred = model(x\_batch)

            loss = compute\_loss(y\_batch, y\_pred)

        gradients = tape.gradient(loss, [k, b])

        optimizer.apply\_gradients(zip(gradients, [k, b]))

    # Printing results every 100 epochs

    if (epoch + 1) % 100 == 0:

        y\_pred\_full = model(X)

        full\_loss = compute\_loss(y, y\_pred\_full).numpy()

        print(f"Epoch {epoch + 1}: Loss={full\_loss:.4f}, k={k.numpy()[0]:.4f}, b={b.numpy()[0]:.4f}")

print("\nTraining completed!")

print(f"Final parameters: k={k.numpy()[0]:.4f}, b={b.numpy()[0]:.4f}")

**Результат:**

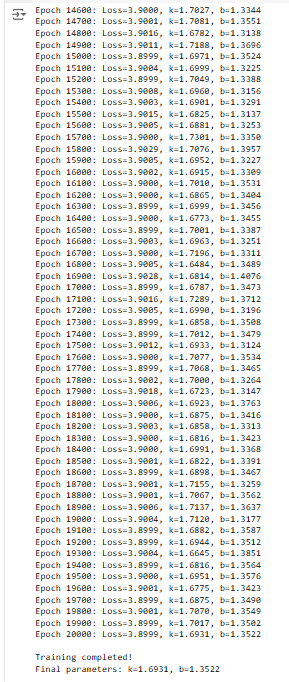
****

Рис. 1. Результат ітерацій.

Висновок результату роботи коду:

Модель лінійної регресії навчилася на даних, і підсумкові параметри kk та bb після 20000 епох складають:

• k = 1.6931k = 1.6931 (очікувалося k = 2k = 2),

• b = 1.3522b = 1.3522 (очікувалося b = 1b = 1).

Відхілення можут значити таке -

1. Випадковий шум у даних: При генерації даних додавався шум, який ускладнив точне відновлення параметрів.

2. Темп навчання: Хоча процес мінімізації втрат стабілізувався, kk і bb не досягли ідеальних значень. Можливо, варто переглянути швидкість навчання чи кількість епох.

3. Початкові параметри: Випадкові початкові значення kk і bb могли вплинути на підсумкові результати.

Підсумок:

Модель успішно наблизилася до справжніх значень коефіцієнтів, мінімізувавши помилку, але через шум і гіперпараметри результат має невеликі відхилення. Для покращення точності можна:

• Збільшити кількість епох,

• Зменшити рівень шуму даних,

• Підібрати оптимальні гіперпараметри (швидкість навчання, початкові значення).

***Висновок:*** Під час лабораторної роботи було досліджено ресурси Keras і TensorFlow, зокрема використано TensorFlow для реалізації та навчання моделі лінійної регресії.

**Github** - https://github.com/TAMOTO24/-Intelligen-Systems