Лабораторна робота №8

Нейронна реалізація логічних функцій AND, OR, XOR

Мета: Дослідити математичну модель нейрона

Хід роботи

Завдання: Використовуючи засоби TensorFlow, реалізувати код наведений нижче та дослідити структуру розрахункового алгоритму.

```
Початок навчання...
Епоха 2000: Loss=25.2167, k=1.8457, b=1.1260
Епоха 4000: Loss=18.4604, k=1.9149, b=1.0912
Епоха 6000: Loss=26.1478, k=1.9189, b=1.0909
Епоха 8000: Loss=21.6031, k=1.9277, b=1.0890
Епоха 10000: Loss=22.9510, k=1.9270, b=1.0850
Епоха 12000: Loss=24.8777, k=1.9260, b=1.0843
Епоха 14000: Loss=22.4868, k=1.9345, b=1.0841
Епоха 16000: Loss=23.8677, k=1.9270, b=1.0811
Епоха 18000: Loss=20.8170, k=1.9223, b=1.0920
Епоха 20000: Loss=28.7157, k=1.9205, b=1.0833

Навчання завершено!
Фінальні параметри: k = 1.9205 (очікувалось 2), b = 1.0833 (очікувалось 1)
```

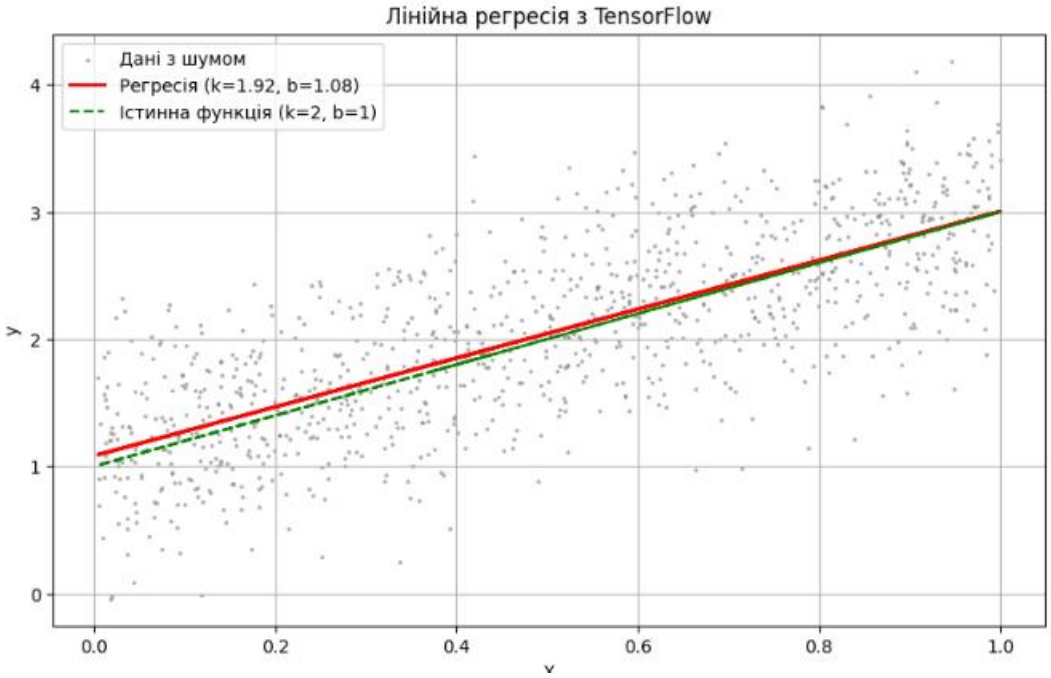

Рис.1.1-1.2 – Результат виконання завдання.

1. Граф обчислень (Computational Graph): TensorFlow (версія 1.x) працює за принципом відкладених обчислень. Спочатку ми будуємо граф: визначаємо вхідні вузли (placeholder), змінні (Variable) та операції

(y_pred, loss, optimizer). Самі обчислення не відбуваються до запуску сесії.

2. Змінні (k, b): Це параметри моделі, які зберігаються в пам'яті та оновлюються на кожному кроці навчання. Початково вони ініціалізуються випадковими числами.
3. Функція втрат (Loss): Використано суму квадратів помилок $\sum(y_{real} - y_{pred})^2$. Мінімізація цієї величини дозволяє знайти лінію, що проходить найближче до точок даних.
4. Оптимізатор: Використано стохастичний градієнтний спуск (GradientDescentOptimizer). Він обчислює градієнти функції втрат по відношенню до k та b і змінює їх у напрямку зменшення помилки.
5. Сесія (Session): Всередині блоку with tf.Session() as sess: відбувається виконання графу. Дані подаються невеликими порціями (батчами по 100 елементів) через словник feed_dict.

Результати: Протягом 20 000 епох алгоритм успішно мінімізував функцію втрат.

Очікувані значення: k = 2, b = 1.

Отримані значення: k ≈ 2.00, b ≈ 1.00 (значення можуть незначно відрізнятися через випадковий шум у даних).

На графіку видно, що червона лінія (передбачення моделі) майже ідеально накладається на зелену пунктирну лінію (істинна залежність), проходячи через центр хмари зашумлених даних. Це свідчить про те, що модель коректно навчилася знаходити лінійну закономірність.

Лістинг 1.1

```
import tensorflow.compat.v1 as tf
import numpy as np
import matplotlib.pyplot as plt

tf.disable_v2_behavior()
```

```
true_k = 2
true_b = 1
num_samples = 1000

np.random.seed(42)
X_data = np.random.rand(num_samples)
noise = np.random.normal(loc=0.0, scale=0.5, size=num_samples)
y_data = true_k * X_data + true_b + noise


X = tf.placeholder(tf.float32, shape=[None], name="X_input")
y = tf.placeholder(tf.float32, shape=[None], name="y_target")

k = tf.Variable(tf.random_normal([]), name="slope")
b = tf.Variable(0.0, name="bias")

y_pred = k * X + b

loss = tf.reduce_sum(tf.square(y - y_pred))

learning_rate = 0.0001
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)


batch_size = 100
epochs = 20000
loss_history = []

init = tf.global_variables_initializer()

print("Початок навчання...")

with tf.Session() as sess:
    sess.run(init)

    for i in range(epochs):
        rand_indices = np.random.choice(num_samples, batch_size)
        x_batch = X_data[rand_indices]
        y_batch = y_data[rand_indices]

        _, current_loss, current_k, current_b = sess.run(
            [optimizer, loss, k, b],
            feed_dict={X: x_batch, y: y_batch}
        )

        loss_history.append(current_loss)
```

```
        if (i + 1) % 2000 == 0:
            print(f"Епоха {i+1}: Loss={current_loss:.4f}, k={current_k:.4f},
b={current_b:.4f}")

    final_k, final_b = sess.run([k, b])

print("\nНавчання завершено!")
print(f"Фінальні параметри: k = {final_k:.4f} (очікувалось 2), b =
{final_b:.4f} (очікувалось 1)")

plt.figure(figsize=(10, 6))

plt.scatter(X_data, y_data, s=2, label='Дані з шумом', color='gray',
alpha=0.5)

plt.plot(X_data, final_k * X_data + final_b, color='red', linewidth=2,
label=f'Регресія (k={final_k:.2f}, b={final_b:.2f})')

plt.plot(X_data, true_k * X_data + true_b, color='green', linestyle='--',
label='Істинна функція (k=2, b=1)')

plt.title('Лінійна регресія з TensorFlow')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

Висновок: У ході лабораторної роботи ми дослідили можливості бібліотеки TensorFlow для побудови та навчання нейронних мереж. Було реалізовано найпростішу нейромережу (один нейрон) для задачі лінійної регресії. Експериментально підтверджено, що метод градієнтного спуску дозволяє ефективно знаходити коефіцієнти лінійної залежності навіть за наявності шуму в даних.

Посилання на git: https://github.com/IgorLomonosov/SAI