ЛАБОРАТОРНА РОБОТА № 8

Ресурси Keras. TensorFlow. Навчання лінійної регресії.

Мета: Дослідження ресурсу Keras і TensorFlow. Застосування TensorFlow.

Варіант 7

Хід роботи:

Посилання на GitHub:

Завдання 1

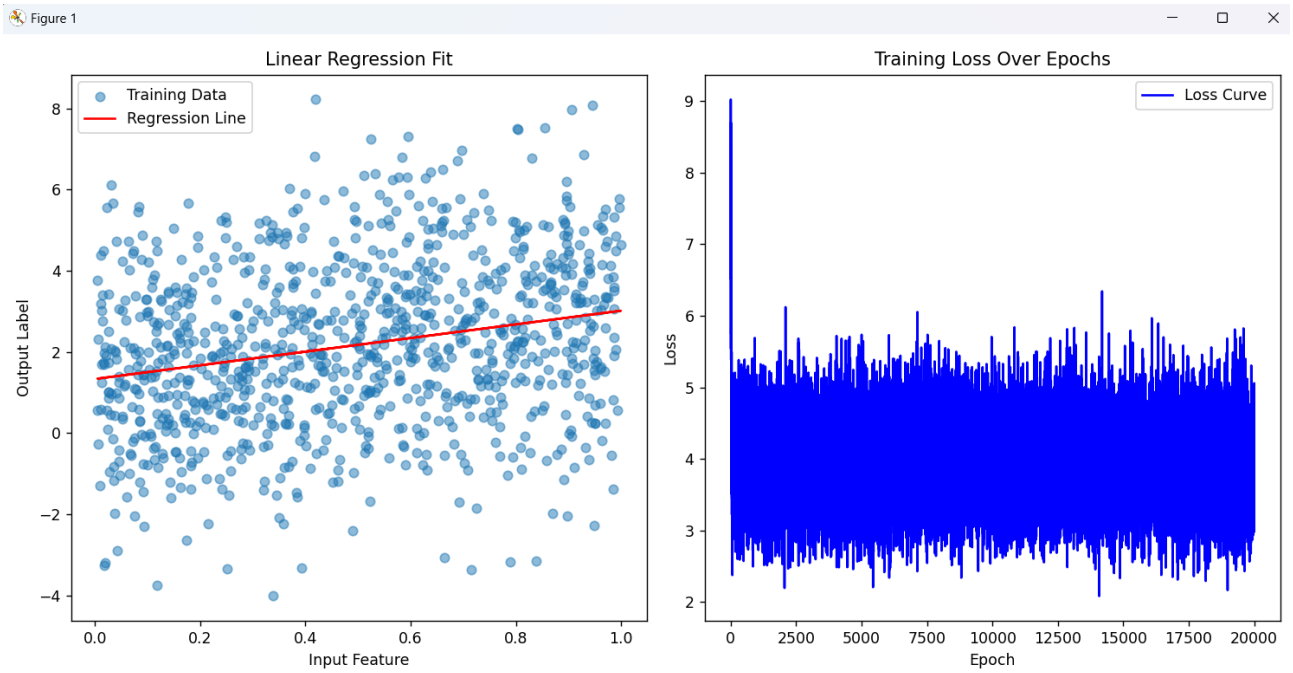
Кластеризація даних за допомогою методу k-середніх

Програмний код:

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

matplotlib.use('TkAgg')

np.random.seed(42)

input_features = np.random.rand(1000, 1).astype(np.float32)
noise_component = np.random.normal(0, 2, size=(1000, 1)).astype(np.float32)
output_labels = 2 * input_features + 1 + noise_component

slope = tf.Variable(tf.random.normal([1]), name='slope')
intercept = tf.Variable(tf.zeros([1]), name='intercept')

learning_rate = 0.01
num_epochs = 20000
batch_size = 100

sgd_optimizer = tf.optimizers.SGD(learning_rate)

def calculate_loss(actual, predicted):
    return tf.reduce_mean(tf.square(actual - predicted))
```

```
loss_history = []

for epoch in range(num_epochs):
    random_indices = np.random.choice(len(input_features), batch_size)
    x_batch = input_features[random_indices]
    y_batch = output_labels[random_indices]

    with tf.GradientTape() as tape:
        predictions = slope * x_batch + intercept
        current_loss = calculate_loss(y_batch, predictions)
    gradients = tape.gradient(current_loss, [slope, intercept])
    sgd_optimizer.apply_gradients(zip(gradients, [slope, intercept]))

    loss_history.append(current_loss.numpy())

    if (epoch + 1) % 1000 == 0:
        print(f"Epoch {epoch + 1}: Loss={current_loss.numpy():.4f}, "
              f"Slope={slope.numpy()[0]:.4f}, Intercept={intercept.numpy()[0]:.4f}")

print(f"Final model parameters: Slope={slope.numpy()[0]:.4f},
Intercept={intercept.numpy()[0]:.4f}")

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(input_features, output_labels, label='Training Data', alpha=0.5)
plt.plot(input_features, slope.numpy() * input_features + intercept.numpy(),
         color='red', label='Regression Line')
plt.title('Linear Regression Fit')
plt.xlabel('Input Feature')
plt.ylabel('Output Label')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(num_epochs), loss_history, color='blue', label='Loss Curve')
plt.title('Training Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Результат виконання:




```
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1000: Loss=2.9137, Slope=1.5185, Intercept=1.4380
Epoch 2000: Loss=4.0864, Slope=1.6677, Intercept=1.3556
Epoch 3000: Loss=3.4294, Slope=1.7159, Intercept=1.3493
Epoch 4000: Loss=2.9558, Slope=1.6785, Intercept=1.3550
Epoch 5000: Loss=4.0940, Slope=1.6410, Intercept=1.3770
Epoch 6000: Loss=4.1837, Slope=1.6770, Intercept=1.3631
Epoch 7000: Loss=3.8920, Slope=1.6564, Intercept=1.3662
Epoch 8000: Loss=3.4565, Slope=1.7110, Intercept=1.3558
Epoch 9000: Loss=4.2998, Slope=1.7114, Intercept=1.3500
Epoch 10000: Loss=3.6722, Slope=1.7081, Intercept=1.3399
Epoch 11000: Loss=4.1471, Slope=1.6605, Intercept=1.3793
Epoch 12000: Loss=3.9804, Slope=1.7041, Intercept=1.3371
Epoch 13000: Loss=3.6825, Slope=1.6727, Intercept=1.3543
Epoch 14000: Loss=3.5979, Slope=1.7379, Intercept=1.3364
Epoch 15000: Loss=3.1446, Slope=1.7003, Intercept=1.3299
Epoch 16000: Loss=3.8188, Slope=1.7080, Intercept=1.3244
Epoch 17000: Loss=5.2164, Slope=1.6989, Intercept=1.3453
Epoch 18000: Loss=3.3307, Slope=1.6893, Intercept=1.3679
Epoch 19000: Loss=4.3900, Slope=1.6774, Intercept=1.3288
Epoch 20000: Loss=4.5945, Slope=1.6822, Intercept=1.3330
Final model parameters: Slope=1.6822, Intercept=1.3330
```

Дослідження розрахункового алгоритму

У даній роботі досліджується алгоритм лінійної регресії з використанням стохастичного градієнтного спуску (SGD) для визначення параметрів лінійної моделі. Алгоритм реалізований з використанням бібліотеки TensorFlow, а дані генеруються штучно для імітації задачі регресії.

1. Створення початкових даних

Алгоритм починається з генерації випадкових даних, які слідують лінійній залежності y=2x+1+шуму = 2x + 1 + \text{шум}. Генерація даних проводиться у два кроки: створення випадкових значень для xx та додавання шуму, щоб моделювати реальні дані.

Код генерації даних:

```
np.random.seed(42)

input_features = np.random.rand(1000, 1).astype(np.float32)
noise_component = np.random.normal( loc: 0,  scale: 2, size=(1000, 1)).astype(np.float32)
output_labels = 2 * input_features + 1 + noise_component
```

Це дозволяє згенерувати 1000 точок, де кожна точка xx має відповідний вихід yy з доданим випадковим шумом. Шум забезпечує, що дані не є ідеальними, як це буває в реальних задачах.

2. Ініціалізація моделі

Модель лінійної регресії представлена у вигляді параметрів: коефіцієнт нахилу (slope) та вільний член (intercept). Параметри ініціалізуються випадковими значеннями.

Код ініціалізації:

```
slope = tf.Variable(tf.random.normal([1]), name='slope')
intercept = tf.Variable(tf.zeros([1]), name='intercept')
```

• slope – параметр, що відповідає за нахил лінії регресії.

• intercept – зсув лінії відносно осі y.

3. Процес навчання (Стохастичний градієнтний спуск)

Для навчання моделі використовується стояхастичний градієнтний спуск (SGD), який оновлює параметри на основі випадково обраної підмножини даних (batch). Оновлення параметрів відбувається шляхом мінімізації функції втрат, що обчислює середньоквадратичну помилку (MSE)

Код процесу навчання:

```
learning_rate = 0.01
num_epochs = 20000
batch_size = 100

sgd_optimizer = tf.optimizers.SGD(learning_rate)

1 usage
def calculate_loss(actual, predicted):
    return tf.reduce_mean(tf.square(actual - predicted))

loss_history = []

for epoch in range(num_epochs):
    random_indices = np.random.choice(len(input_features), batch_size)
    x_batch = input_features[random_indices]
    y_batch = output_labels[random_indices]

    with tf.GradientTape() as tape:
        predictions = slope * x_batch + intercept
        current_loss = calculate_loss(y_batch, predictions)
    gradients = tape.gradient(current_loss, sources: [slope, intercept])
    sgd_optimizer.apply_gradients(zip(gradients, [slope, intercept]))

    loss_history.append(current_loss.numpy())

    if (epoch + 1) % 1000 == 0:
        print(f"Epoch {epoch + 1}: Loss={current_loss.numpy():.4f}, "
              f"Slope={slope.numpy()[0]:.4f}, Intercept={intercept.numpy()[0]:.4f}")

print(f"Final model parameters: Slope={slope.numpy()[0]:.4f}, Intercept={intercept.numpy()[0]:.4f}")
```

На кожному кроці:

1. Випадково обирається міні-батч розміром batch_size.
2. Обчислюються градієнти функції втрат відносно параметрів моделі.
3. Параметри оновлюються за допомогою SGD.

4. Аналіз результатів

Після завершення навчання модель повертає знайдені параметри лінії регресії: коефіцієнт нахилу kk та вільний член bb.

Виведення результатів:

```
print(f"Final model parameters: Slope={slope.numpy()[0]:.4f}, Intercept={intercept.numpy()[0]:.4f}")
```

Також будується графік, який порівнює вихідні дані та знайдену лінію регресії, а також графік втрат під час навчання.

Візуалізація результатів:

```
plt.figure(figsize=(12, 6))

plt.subplot( *args: 1, 2, 1)
plt.scatter(input_features, output_labels, label='Training Data', alpha=0.5)
plt.plot( *args: input_features, slope.numpy() * input_features + intercept.numpy(),
         color='red', label='Regression Line')
plt.title('Linear Regression Fit')
plt.xlabel('Input Feature')
plt.ylabel('Output Label')
plt.legend()

plt.subplot( *args: 1, 2, 2)
plt.plot( *args: range(num_epochs), loss_history, color='blue', label='Loss Curve')
plt.title('Training Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

5. Висновок

У результаті роботи алгоритму:

1. Параметри лінійної моделі (kk і bb) знаходяться методом стохастичного градієнтного спуску.

2. Алгоритм поступово знижує втрати, що відображається на графіку зміни функції втрат.

3. Знайдена лінія регресії адекватно апроксимує вихідні дані, незважаючи на доданий шум.

Приклад результатів:

Це значення близькі до заданих k=2k=2 та b=1b=1, що підтверджує ефективність алгоритму.

Висновки:

в ході виконання лабораторної роботи було досліджено ресурси Keras і TensorFlow. Застосовано TensorFlow.