Bài tập số 4: Mạng nơ ron đa tầng, đơn lớp, đa lớp

Thực hành 3

1. **Lớp 1 (Lớp ẩn):**

```python
@tf.function
def layer1(X, W, B):
    return tf.nn.relu(tf.matmul(X, W) + B)
```

- Sử dụng hàm kích hoạt relu sau khi thực hiện phép nhân ma trận giữa đầu vào `X` và trọng số `W1`, sau đó cộng thêm độ lệch `B1`.

2. **Lớp 2 (Lớp đầu ra):**

```python
@tf.function
def layer2(X, W, B):
    return tf.nn.sigmoid(tf.matmul(X, W) + B)
```

- Sử dụng hàm kích hoạt sigmoid sau khi thực hiện phép nhân ma trận giữa đầu ra của lớp đầu tiên và trọng số `W2`, sau đó cộng thêm độ lệch `B2`.

3. **Hàm dự đoán:**

```python
@tf.function
def predict(X, W1, B1, W2, B2):
    return layer2(layer1(X, W1, B1), W2, B2)
```

- Chuyển đầu vào `X` qua cả hai lớp để lấy đầu ra cuối cùng, `y_hat`.

4. **Hàm mất mát (Binary Cross-Entropy):**

```python
@tf.function
def L(y, y_hat):
    epsilon = 1e-7
    y_hat = tf.clip_by_value(y_hat, epsilon, 1.0 - epsilon)
    return -tf.reduce_mean(y * tf.math.log(y_hat) + (1 - y) * tf.math.log(1-y_hat))
```

- Tính toán mất mát giữa nhãn thực `y` và nhãn dự đoán `y_hat` sử dụng binary cross-entropy.

5. **Vòng lặp huấn luyện:**

```
X = tf.constant([[0.0, 0],[0, 1],[1, 0],[1, 1]])
y = tf.constant([[0.0],[1],[1],[0]])

n = 2
W1 = tf.Variable(tf.random.normal((n, 2)))
B1 = tf.Variable([0.0, 0.0])

W2 = tf.Variable(tf.random.normal((2, 1)))
B2 = tf.Variable([0.0])
alpha = 0.1

for it in range(150):
    with tf.GradientTape() as t:
        current_loss = L(y, predict(X, W1, B1, W2, B2))
    print("it", it, current_loss)
    dW1, dB1, dW2, dB2 = t.gradient(current_loss, [W1, B1, W2, B2])
    W1.assign_sub(alpha * dW1)
    B1.assign_sub(alpha * dB1)
    W2.assign_sub(alpha * dW2)
    B2.assign_sub(alpha * dB2)

y_hat = predict(X, W1, B1, W2, B2)
```

- Trong 150 vòng lặp, tính toán gradient của mất mát đối với các trọng số và độ lệch, sau đó cập nhật chúng bằng phương pháp gradient descent.

6. **Dự đoán:**

```
y_pred = tf.cast(y_hat >= 0.5, tf.int32)
print(y_pred)
```

- Sau khi huấn luyện, mô hình sẽ dự đoán đầu ra cho mỗi đầu vào trong `X` và chuyển đổi đầu ra liên tục sang dự đoán nhị phân dựa trên việc nó lớn hơn hay nhỏ hơn 0.5.

```
it 145 tf.Tensor(0.27417344, shape=(), dtype=float32)
it 146 tf.Tensor(0.27075142, shape=(), dtype=float32)
it 147 tf.Tensor(0.2685514, shape=(), dtype=float32)
it 148 tf.Tensor(0.2670531, shape=(), dtype=float32)
it 149 tf.Tensor(0.26655546, shape=(), dtype=float32)
tf.Tensor(
[[0]
 [1]
 [1]
 [0]], shape=(4, 1), dtype=int32)

Process finished with exit code 0
```

Thực hành 4