Huỳnh Minh Luân

B2106842

Bài tập số 2: Mạng nơ ron đơn tầng, 2 lớp (tensorflow)

Thực hành 1 + 2

1. **Hàm Dự đoán (predict function):**

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

- Hàm này nhận đầu vào X, trọng số W, và độ lệch b, sau đó tính toán đầu ra bằng cách thực hiện phép biến đổi tuyến tính, tiếp theo là áp dụng hàm kích hoạt sigmoid.
- Đây là một mạng nơ-ron đơn lớp với hàm kích hoạt sigmoid.

2. **Hàm Mất mát (L function):**

```python
@tf.function
def L(y, y_hat):
    return tf.reduce_mean((y - y_hat)**2)
```

$$L = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

```python
@tf.function
def L(y, y_hat):
    loss = -tf.reduce_mean(y * tf.math.log(y_hat) + (1 - y) * tf.math.log(1 - y_hat))
    return loss
```

$$L = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(\hat{y}^{(i)}\right)\right)$$

- Hàm này tính toán hàm mất mát trung bình bình phương (MSE) giữa nhãn thực tế y và đầu ra dự đoán y_hat.

3. Quá trình Huấn luyện:

```python
X = tf.constant([[0.0, 0], [0, 1], [1, 0], [1, 1]])
y = tf.constant([[0.0], [0], [0], [1]])
# Khởi tạo W và b ngẫu nhiên
W = tf.Variable(tf.random.normal(shape=[2, 1]))
b = tf.Variable(tf.random.normal(shape=[1]))
```

- Đầu tiên, trọng số W và độ lệch b được khởi tạo ngẫu nhiên.

```
alpha = 0.1

for it in range(500):
    with tf.GradientTape() as t:
        current_loss = L(y, predict(X, W, b))
    print("it", it, current_loss)
    dW, db = t.gradient(current_loss, [W, b])
    W.assign_sub(alpha * dW)
    b.assign_sub(alpha * db)

y_hat = predict(X, W, b)
```

- Trong mỗi vòng lặp, mã tính toán hàm mất mát hiện tại bằng gradient tape, sau đó tính gradient của W và b dựa trên hàm mất mát này.
- Trọng số và độ lệch được cập nhật bằng cách sử dụng phương pháp gradient descent với tốc độ học alpha.

4. **Dự đoán sau khi Huấn luyện:**

```
# So sánh với 0.5 để xác định nhãn dự báo
y_pred = tf.cast(y_hat >= 0.5, tf.int32)
print(y_pred)
```

- Sau 500 lần lặp, mô hình sẽ dự đoán nhãn cho X.
- Đầu ra dự đoán được so sánh với 0.5 để xác định nhãn dự báo cuối cùng (y_pred), kết quả sẽ là 0 hoặc 1.

5. **Kết quả**

```
it 491 tf.Tensor(0.07883472, shape=(), dtype=float32)
it 492 tf.Tensor(0.07875298, shape=(), dtype=float32)
it 493 tf.Tensor(0.078671455, shape=(), dtype=float32)
it 494 tf.Tensor(0.07859011, shape=(), dtype=float32)
it 495 tf.Tensor(0.07850892, shape=(), dtype=float32)
it 496 tf.Tensor(0.078427956, shape=(), dtype=float32)
it 497 tf.Tensor(0.07834715, shape=(), dtype=float32)
it 498 tf.Tensor(0.07826653, shape=(), dtype=float32)
it 499 tf.Tensor(0.07818609, shape=(), dtype=float32)
tf.Tensor(
[[0]
 [0]
 [0]
 [1]], shape=(4, 1), dtype=int32)
```

```
it 491 tf.Tensor(0.23782799, shape=(), dtype=float32)
it 492 tf.Tensor(0.23753068, shape=(), dtype=float32)
it 493 tf.Tensor(0.23723412, shape=(), dtype=float32)
it 494 tf.Tensor(0.23693836, shape=(), dtype=float32)
it 495 tf.Tensor(0.2366434, shape=(), dtype=float32)
it 496 tf.Tensor(0.23634917, shape=(), dtype=float32)
it 497 tf.Tensor(0.23605575, shape=(), dtype=float32)
it 498 tf.Tensor(0.2357631, shape=(), dtype=float32)
it 499 tf.Tensor(0.23547123, shape=(), dtype=float32)
tf.Tensor(
[[0]
 [0]
 [0]
 [1]], shape=(4, 1), dtype=int32)
```

Thực hành 3

1. **Đọc và Chuẩn bị Dữ liệu:**

```
# Đọc file dữ liệu
df = pd.read_csv('iris.data', header=None)

# Giữ lại 100 dòng đầu tiên để chỉ bao gồm hai lớp: Iris-setosa và Iris-versicolor
Data = df.head(100)

# Xác định cột nhãn
label_column = Data.columns[-1]

# Mã hóa nhãn bằng map
label_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1}
Data['label_encoded'] = Data[label_column].map(label_mapping)

# Xóa cột nhãn cũ
Data = Data.drop(label_column, axis=1)

# Tách X và y
X = Data.drop('label_encoded', axis=1)
y = Data['label_encoded']

# Chia dữ liệu thành tập huấn luyện và kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Dữ liệu được đọc từ file iris.data và chỉ giữ lại 100 dòng đầu tiên tương ứng với hai lớp: Iris-setosa và Iris-versicolor.
- Cột nhãn được mã hóa thành các giá trị số (0 cho Iris-setosa và 1 cho Iris-versicolor).
- Dữ liệu sau đó được tách thành các biến đầu vào X và nhãn y.

2. **Chia Tập Huấn Luyện và Tập Kiểm Tra:**

```
# Chia dữ liệu thành tập huấn luyện và kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Dữ liệu được chia thành hai phần: 80% cho tập huấn luyện và 20% cho tập kiểm tra.

3. **Chuyển Đổi Dữ Liệu Thành Tensor:**

```
# Chuyển đổi dữ liệu sang dạng tensor
X_train = tf.convert_to_tensor(X_train, dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train.values.reshape(-1, 1), dtype=tf.float32)
X_test = tf.convert_to_tensor(X_test, dtype=tf.float32)
y_test = tf.convert_to_tensor(y_test.values.reshape(-1, 1), dtype=tf.float32)
```

- Các biến X_train, y_train, X_test, và y_test được chuyển đổi thành các tensor để có thể sử dụng trong TensorFlow.

4. **Khởi Tạo Trọng Số và Bias:**

```
# Khởi tạo trọng số và bias
W = tf.Variable(tf.random.normal(shape=(4, 1)))
b = tf.Variable(tf.random.normal(shape=(1,)))
```

- Trọng số W và bias b được khởi tạo ngẫu nhiên với kích thước phù hợp cho mô hình hồi quy logistic.

5. **Định Nghĩa Hàm Sigmoid và Hàm Mất Mát:**

```
# Định nghĩa hàm sigmoid cho mô hình
def sigmoid(x):
    return 1 / (1 + tf.exp(-x))


# Định nghĩa hàm dự đoán
def predict(X):
    return sigmoid(tf.matmul(X, W) + b)


# Định nghĩa hàm mất mát binary cross-entropy
def binary_cross_entropy(y_true, y_pred):
    # epsilon = 1e-7
    # y_pred = tf.clip_by_value(y_pred, epsilon, 1.0 - epsilon)
    return -tf.reduce_mean(y_true * tf.math.log(y_pred) + (1 - y_true) * tf.math.log(1-y_pred))
```

- Hàm sigmoid được sử dụng làm hàm kích hoạt cho mô hình.
- Hàm mất mát binary cross-entropy được định nghĩa để đo lường sự khác biệt giữa giá trị dự đoán và giá trị thực tế.

6. **Huấn Luyện Mô Hình:**

```python
# Huấn luyện mô hình
learning_rate = 0.1
epochs = 100

for epoch in range(epochs):
    with tf.GradientTape() as tape:
        y_pred = predict(X_train)
        loss = binary_cross_entropy(y_train, y_pred)

    # Tính toán gradient
    dW, db = tape.gradient(loss, [W, b])

    # Cập nhật trọng số và bias
    W.assign_sub(learning_rate * dW)
    b.assign_sub(learning_rate * db)

    # if epoch % 10 == 0:
    print(f"Epoch {epoch}, Loss: {loss.numpy()}")
```

- Mô hình được huấn luyện qua 100 epochs sử dụng gradient descent để cập nhật trọng số và bias.
- Tại mỗi epoch, hàm mất mát được tính toán và trọng số được cập nhật dựa trên gradient.

7. **Dự Đoán và Đánh Giá Mô Hình:**

```python
# Dự đoán trên tập kiểm tra
y_hat_test = predict(X_test)

# Phân ngưỡng để lấy nhãn dự báo
y_pred_test = tf.cast(y_hat_test >= 0.5, tf.float32)

# Tính độ chính xác phân lớp
accuracy = tf.reduce_mean(tf.cast(tf.equal(y_test, y_pred_test), tf.float32))

print("Test Accuracy:", accuracy.numpy())
```

- Sau khi huấn luyện, mô hình dự đoán nhãn cho tập kiểm tra X_test.

- Kết quả dự đoán được phân ngưỡng tại giá trị 0.5 để xác định nhãn dự báo cuối cùng.
- Độ chính xác của mô hình trên tập kiểm tra được tính toán và in ra.

8. Kết quả

```
Epoch 87, Loss: 0.0827808827161789
Epoch 88, Loss: 0.08187656104564667
Epoch 89, Loss: 0.08099200576543808
Epoch 90, Loss: 0.08012659102678299
Epoch 91, Loss: 0.07927972078323364
Epoch 92, Loss: 0.0784507766366005
Epoch 93, Loss: 0.07763922214508057
Epoch 94, Loss: 0.07684449851512909
Epoch 95, Loss: 0.07606613636016846
Epoch 96, Loss: 0.0753035843372345
Epoch 97, Loss: 0.07455640286207199
Epoch 98, Loss: 0.07382414489984512
Epoch 99, Loss: 0.07310633361339569
Test Accuracy: 1.0
```