3-3. 체인 룰(Chain Rule) 딥러닝에서 역전파 알고리즘과 연결

체인 룰(Chain Rule)은 딥러닝에서 역전파 알고리즘(Backpropagation)과 매우 깊은 연관이 있습니다. 신경망의 학습에서 체인 룰은 가중치에 대한 오차의 변화율(기울기)을 계산하는데 사용되며, 이 과정이 역전파의 핵심입니다.

1. 체인 룰의 기본 개념

   체인 룰은 합성 함수의 미분을 구하는 방법입니다. 예를 들어, 함수 $y = f(g)(x))$가 주어 졌다면, 체인 룰을 사용해 이를 다음과 같이 미분할 수 있습니다.

   $$\frac{dy}{dx} = \frac{df}{dg} * \frac{dg}{dx}$$

   이것은 복합적인 함수에서 각각의 내부 함수의 변화율을 곱하는 방식입니다.

2. 역전파 알고리즘(Backpropagation)과 체인 룰

   신경망에서의 연쇄 법칙

   신경망의 각 층은 함수의 합성 형태로 표현될 수 있습니다. 예를 들어, 3층 신경망을 다음과 같이 정의한다고 가정해봅시다.

   - $z_1 = W_1 * x + b_1$
   - $a_1 = f(z_1)$
   - $z_2 = W_2 * a_1 + b_2$
   - $a_2 = f(z_2)$
   - $z_3 = W_3 * a_2 + b_3$
   - $a_3 = f(z_3)$
   - $\hat{y} = a_3$

   여기서 W는 가중치, b는 편향, f는 활성화 함수입니다. 신경망의 목표는 입력 x에 대해 출력 $\hat{y}$와 실제 목표 값 y 사이의 오차를 최소화 하는 것입니다.

역전파의 개념

역전파는 각 층에서 오차의 변화율, 즉 기울기를 계산하여 가중치를 업데이트하는 방법입니다. 각 층의 가중치가 미치는 영향은 체인 룰을 사용해 계산됩니다.

예를 들어, 신경망의 마지막 출력층에서 손실 함수 $L(\hat{y}, y)$를 미분할 때, 출력층의 가중치에 대한 미분은 체인 룰을 적용하여 계산됩니다.

$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial a_3} * \frac{\partial a_3}{\partial z_3} * \frac{\partial z_3}{\partial W_3}$$

이 과정을 각 층에 대해 반복하면서, 손실 함수가 가중치에 얼마나 영향을 미치는지 계산하는 것이 역전파의 핵심입니다.

3. 체인 룰의 단계별 적용

(1) 출력층에서 오차 계산

우선 출력층에서 손실함수 L에 대한 기울기를 계산합니다. 예를 들어, 손실 함수가 평균 제곱 오차(MSE)라면

$$L = \frac{1}{2}(\hat{y} - y)^2$$

출력층에서의 오차는

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

(2) 활성화 함수의 미분

각 층에서 사용하는 활성화 함수의 기울기를 계산합니다. 예를 들어, 활성화 함수로 시그모이드 함수$\partial(z) = \frac{1}{1+e^{-1}}$를 사용하면, 시그모이드 함수의 미분은 $\frac{\partial \sigma(z)}{\partial z} = \sigma(z) * (1 - \sigma(z))$

(3) 가중치에 대한 기울기 계산

체인 룰을 이용해 가중치에 대한 기울기를 계산합니다. 각 층에서 오차의 변화율을 체인 룰로 연결하여 역전파 합니다. 위의 예에서 W_3에 대한 기울기는 다음과 같이 구할 수 있습니다.

$$\frac{\partial L}{\partial W_3} = \delta_3 \cdot a_2$$

여기서 δ_3는 z_3에 대한 오차 기울기이며, 이는 체인 룰을 통해 계산된 값입니다.

(4) 가중치 업데이트

기울기가 계산된 후에는 경사하강법을 사용해 가중치를 업데이트합니다. 가중치 업데이트는 다음과 같은 방식으로 이루어 집니다.

$$W = W - \eta \cdot \frac{\partial L}{\partial W}$$

여기서 η는 학습률입니다.

```
import numpy as np

# 시그모이드 함수와 그 미분
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# 경사하강법 학습 함수
def train(X, y, epochs, lr):
    np.random.seed(1)

    # 가중치 초기화 (2층 신경망)
    weights_0 = np.random.rand(X.shape[1], 4)  # 입력층 -> 은닉
    weights_1 = np.random.rand(4, 1)           # 은닉층 -> 출력

    # 학습 과정
    for epoch in range(epochs):
        # 순전파 (Forward Propagation)
        layer_0 = X
        z_1 = np.dot(layer_0, weights_0)
        layer_1 = sigmoid(z_1)
        z_2 = np.dot(layer_1, weights_1)
        layer_2 = sigmoid(z_2)

        # 손실 계산 (Mean Squared Error)
```

```
        loss = np.mean((layer_2 - y) ** 2)
        if epoch % 1000 == 0:
            print(f"Epoch {epoch}, Loss: {loss}")

        # 역전파 (Backpropagation)
        layer_2_error = layer_2 - y
        layer_2_delta = layer_2_error * sigmoid_derivative(z_

        layer_1_error = layer_2_delta.dot(weights_1.T)
        layer_1_delta = layer_1_error * sigmoid_derivative(z_

        # 가중치 업데이트
        weights_1 -= layer_1.T.dot(layer_2_delta) * lr
        weights_0 -= layer_0.T.dot(layer_1_delta) * lr

    return weights_0, weights_1

# 학습 데이터
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # XOR 문제
y = np.array([[0], [1], [1], [0]])

# 학습 실행
weights_0, weights_1 = train(X, y, epochs=10000, lr=0.1)

# 결과 출력
def predict(X, weights_0, weights_1):
    layer_0 = X
    layer_1 = sigmoid(np.dot(layer_0, weights_0))
    layer_2 = sigmoid(np.dot(layer_1, weights_1))
    return layer_2

predictions = predict(X, weights_0, weights_1)
print(f"Predictions:\n{predictions}")
```

코드 설명

1. 순전파 (Forward Propagation): 입력 데이터를 통해 출력값을 계산합니다. 각 층의 결과는 활성화 함수(시그모이드)를 통해 얻습니다.

2. 역전파 (Back Propagation): 체인 룰을 사용해 오차를 각 층으로 전달하며, 각 층의 가중치에 대한 기울기를 계산합니다.
3. 가중치 업데이트: 경사하강법을 사용해 가중치를 업데이트합니다.
4. 예측: 학습된 가중치를 사용해 입력 데이터에 대한 예측값을 출력합니다.