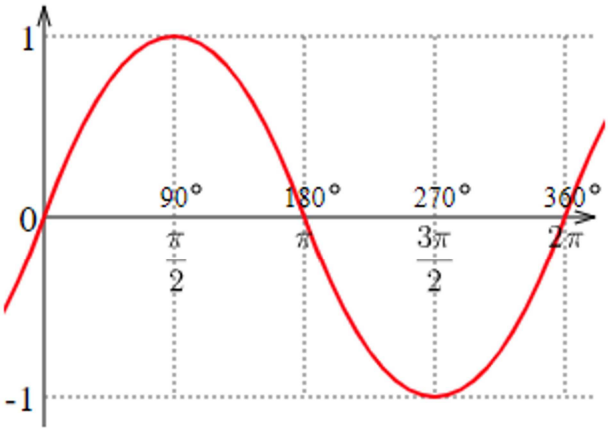
08. 회귀 문제의 역전파 구현

2020년 12월 31일 목요일 오후 3:39

● 신경망 학습 구조를 파악하는 것이 목적이므로 간단한 신경망을 직접 구현해 봄

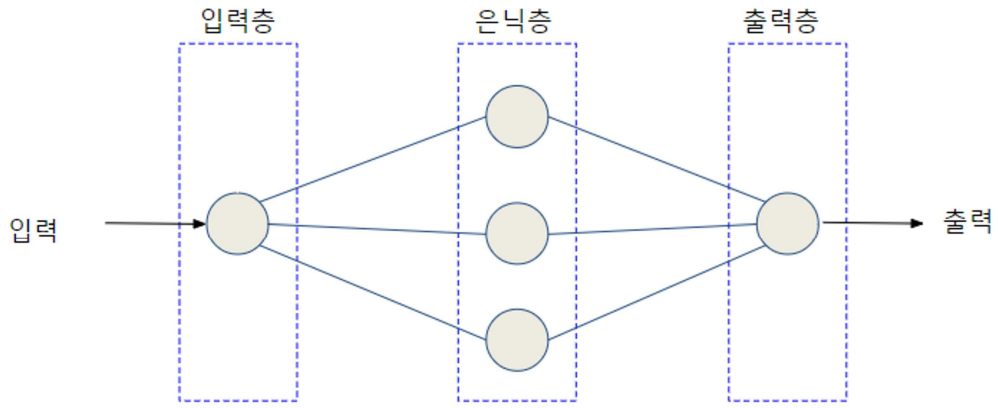
1. sin 함수 학습

1개 이상의 은닉층을 포함하는 신경망
-> deep learning network



● x값을 입력, y값을 출력, sin(x)를 정답으로 함
● sin 함수는 연속 함수이므로 회귀 문제에 해당 feature 1개
● 출력과 정답의 오차를 전파시켜 가중치와 bias를 반복적으로 수정하며 학습

● 입력층 뉴런 1개, 은닉층 뉴런 3개, 출력층 뉴런 1개로 구성



● 은닉층 활성화 함수 : 시그모이드
● 출력층 활성화 함수 : 항등 함수
● 손실 함수 : 오차 제곱합
● 최적화 알고리즘 : SGD
● 배치 사이즈 : 1 one-line 학습

● 학습하는 과정을 살펴보기 위한 목적이므로 모든 데이터를 훈련 데이터로 사용

2. 출력층 구현

● 출력층 클래스

```
class OutputLayer:
    def __init__(self, n_upper, n):  # 초기설정
        self.w = wb_width * np.random.randn(n_upper, n)  # 가중치(행렬)
        self.b = wb_width * np.random.randn(n)  # 편향(벡터)


    def forward(self, x):  # 순전파
        self.x = x
        u = np.dot(x, self.w) + self.b
        self.y = u  # 항등함수


    def backward(self, t):  # 역전파
        delta = self.y - t


        self.grad_w = np.dot(self.x.T, delta)
        self.grad_b = np.sum(delta, axis=0)


        self.grad_x = np.dot(delta, self.w.T)


    def update(self, eta):  # 가중치와 편향 수정
        self.w -= eta * self.grad_w
        self.b -= eta * self.grad_b
```

** np.random.randn(m, n) : 평균 0, 표준편차 1의 표준정규분포 난수 ndarray(m, n) 생성

3. 은닉층 구형

```
class MiddleLayer:
    def __init__(self, n_upper, n):  # 초기설정
        self.w = wb_width * np.random.randn(n_upper, n)  # 가중치(행렬)
        self.b = wb_width * np.random.randn(n)  # 편향(벡터 )


    def forward(self, x):  # 순전파
        self.x = x
        u = np.dot(x, self.w) + self.b
        self.y = 1/(1+np.exp(-u))  # 시그모이드 함수


    def backward(self, grad_y):  # 역전파
        delta = grad_y * (1-self.y)*self.y  # 시그모이드 함수 미분
```

```
        self.grad_w = np.dot(self.x.T, delta)
        self.grad_b = np.sum(delta, axis=0)

        self.grad_x = np.dot(delta, self.w.T)

    def update(self, eta):  # 가중치와 편향 수정
        self.w -= eta * self.grad_w
        self.b -= eta * self.grad_b
```

4. 역전파 구현

```
# -- 각 층의 초기화 --
middle_layer = MiddleLayer(n_in, n_mid)
output_layer = OutputLayer(n_mid, n_out)

# -- 학습 --
for i in range(epoch):

    # 인덱스 임의 섞기
    index_random = np.arange(n_data)
    np.random.shuffle(index_random)

    for idx in index_random:

        x = input_data[idx:idx+1]  # 입력
        t = correct_data[idx:idx+1]  # 정답

        # 순전파
        middle_layer.forward(x.reshape(1, 1))  # 입력을 행렬로 변환
        output_layer.forward(middle_layer.y)

        # 역전파
        output_layer.backward(t.reshape(1, 1))  # 정답을 행렬로 변환
        middle_layer.backward(output_layer.grad_x)

        # 가중치와 편향 수정
        middle_layer.update(eta)
        output_layer.update(eta)
```

5. 전체 코드

```
%matplotlib inline

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

```
# -- 입력과 정답 준비 --
input_data = np.arange(0, np.pi*2, 0.1)  # 입력
correct_data = np.sin(input_data)  # 정답
input_data = (input_data-np.pi)/np.pi  # 입력을 – 1.0 ~ 1.0 범위안으로
n_data = len(correct_data)  # 데이터 수

# -- 각 설정 값 --
n_in = 1  # 입력층의 뉴런 수
n_mid = 3  # 은닉층의 뉴런 수
n_out = 1  # 출력층의 뉴런 수

wb_width = 0.01  # 가중치와 편향 설정을 위한 정규분포의 표준편차
eta = 0.1  # 학습률
epoch = 2001
interval = 200  # 경과 표시간격

# -- 은닉층 --
class MiddleLayer:
    def __init__(self, n_upper, n):  # 초기설정
        self.w = wb_width * np.random.randn(n_upper, n)  # 가중치(행렬)
        self.b = wb_width * np.random.randn(n)  # 편향(벡터 )

    def forward(self, x):  # 순전파
        self.x = x
        u = np.dot(x, self.w) + self.b
        self.y = 1/(1+np.exp(-u))  # 시그모이드 함수


    def backward(self, grad_y):  # 역전파
        delta = grad_y * (1-self.y)*self.y  # 시그모이드 함수 미분


        self.grad_w = np.dot(self.x.T, delta)
        self.grad_b = np.sum(delta, axis=0)


        self.grad_x = np.dot(delta, self.w.T)


    def update(self, eta):  # 가중치와 편향 수정
        self.w -= eta * self.grad_w
        self.b -= eta * self.grad_b

# -- 출력층 --
class OutputLayer:
    def __init__(self, n_upper, n):  # 초기설정
        self.w = wb_width * np.random.randn(n_upper, n)  # 가중치(행렬)
        self.b = wb_width * np.random.randn(n)  # 편향(벡터)
```

```
    def forward(self, x):  # 순전파
        self.x = x
        u = np.dot(x, self.w) + self.b
        self.y = u  # 항등함수

    def backward(self, t):  # 역전파
        delta = self.y - t

        self.grad_w = np.dot(self.x.T, delta)
        self.grad_b = np.sum(delta, axis=0)

        self.grad_x = np.dot(delta, self.w.T)

    def update(self, eta):  # 가중치와 편향 수정
        self.w -= eta * self.grad_w
        self.b -= eta * self.grad_b


# -- 각 층의 초기화 --
middle_layer = MiddleLayer(n_in, n_mid)
output_layer = OutputLayer(n_mid, n_out)

# -- 학습 --bb
for i in range(epoch):

    # 인덱스 임의 섞기
    index_random = np.arange(n_data)
    np.random.shuffle(index_random)

    # 결과 표시
    total_error = 0
    plot_x = []
    plot_y = []

    for idx in index_random:

        x = input_data[idx:idx+1]  # 입력
        t = correct_data[idx:idx+1]  # 정답

        # 순전파
        middle_layer.forward(x.reshape(1, 1))  # 입력을 행렬로 변환
        output_layer.forward(middle_layer.y)

        # 역전파
        output_layer.backward(t.reshape(1, 1))  # 정답을 행렬로 변환
        middle_layer.backward(output_layer.grad_x)
```

```
    # 가중치와 편향 수정
    middle_layer.update(eta)
    output_layer.update(eta)

    if i%interval == 0:

        y = output_layer.y.reshape(-1)  # 행렬을 벡터로 되돌림

        # 오차계산
        total_error += 1.0/2.0*np.sum(np.square(y - t))  # 오차제곱합


        # 출력 기록
        plot_x.append(x)
        plot_y.append(y)

if i%interval == 0:

    # 출력 그래프 표시
    plt.plot(input_data, correct_data, linestyle="dashed")
    plt.scatter(plot_x, plot_y, marker="+")
    plt.show()

    # 에포크 수와 오차 표시
    print("Epoch:" + str(i) + "/" + str(epoch), "Error:" + str(total_error/n_data))
```

6. 은닉층의 뉴런 수가 학습에 미치는 영향

- 은닉층의 뉴런수를 1, 2, 3, 4, 5 차례로 증가시키며 결과를 확인
- 3 또는 4개의 뉴런 이후 에는 결과가 더 좋아지지 않고 계산량만 증가
- 이 문제에서는 은닉층에서 3 또는 4개의 뉴런이 최적임
- 은닉층에서 필요 이상의 뉴런 수는 과적합(Overfitting)을 일으킴

이하의 뉴런 수는 Underfitting

layer의 노드 수에 따라서도 결과 복잡도에 영향을 미침