인공지능개론 5장-6장 정리노트#3

(꼭 PDF로 변환후 제출하기)

ICT 융합공학부 20학번 박준형 / ICT 융합공학부 20학번 김정호

5장

준형: 정호야, 오차역전파법을 이해하는 데 있어서 수치미분과의 차이에 대해 좀 더 자세히 설명해줄 수 있을까?

정호: 물론! 수치미분은 매우 작은 차분을 이용하여 함수의 기울기를 근사하는 방법이야. 이는 간단하고 직관적이지만, 계산량이 많고 오차가 발생할 수 있어. 반면, 오차역전파법은 역전파 알고리즘을 통해 기울기를 효율적으로 계산해. 이를 통해 네트워크 전체의 파라미터에 대한 오차를 효율적으로 업데이트할 수 있어.

정호: 나는 오차역전파법을 이해하는 두 가지 방법에 대해서도 더 자세히 알고 싶어.

준형: 그건 내가 설명할 수 있어! 오차역전파법을 이해하는 두 가지 방법은 수식적 미분과 계산 그래프를 통한 역전파야. 수식적 미분은 파라미터에 대한 미분식을 직접 계산하여 업데이트를 수행하는 방법이야. 반면, 계산 그래프를 통한 역전파는 계산의 흐름을 그래프로 시각화하여 오차를 효율적으로 역전파하는 방법이야. 각각의 방법은 장단점이 있지만, 계산 그래프를 통한 역전파는 네트워크의 구조를 더 잘 이해할 수 있고, 파라미터 업데이트에 있어서도 효율적이지.

준형: 혹시 그러면, 오차역전파법의 역사에 대해서 너는 알고있니 정호야?

정호: 오차역전파법은 1970년대에 등장하여 미분값을 효율적으로 계산하기 위해 개발되었어요 그 후 1986년에 제프리 힌튼이 역전파 알고리즘을 효율적으로 구현하여 딥러닝의 발전에 큰 기여를 했지. 이후 오차역전파법은 딥러닝 학습에서 핵심적인 역할을 하고 있단다.

정호: 준형아 나 또 궁금한게 있어. 덧셈 노드와 곱셈 노드의 역전파는 어떻게 이루어지는 거야?

준형: 덧셈 노드의 역전파는 입력값을 그대로 전달해줘. 즉, 덧셈 노드의 역전파는 입력에 대한 변화량이 그대로 출력에 전달되는 것이지. 반면에 곱셈 노드의 역전파는 입력값에 상대적인 값으로 출력값을 곱해주는 거야. 이것이 바로 덧셈과 곱셈 노드의 역전파 과정이라고해.

정호: 잘 이해했어요. 그럼 Affine 계층과 소프트맥스 함수에 대해서도 더 궁금해!!

준형: 그래 그것도 설명 가능하지! Affine 계층은 입력과 가중치의 내적을 계산하고 편향을 더해주는 역할을 해. 이를 통해 입력 데이터와 가중치를 효과적으로 조합하여 출력값을 계산할 수 있어. 소프트맥스 함수는 출력값을 클래스 확률로 변환해주는 함수로, 각 클래스에 대한 확률을 계산하여 모델의 출력을 해석하기 쉽게 해주는 특징이 있어.

정호: 고마워 덕분에 5장에 대한 부족했던 내용을 더 공부할 수 있었어.

준형: 나도 고마워 정호야 다음에 또 6장으로 공부해보자!

6장

준형: 정호야, 매개변수 갱신시 최적화에 대해서 알고 있어?

정호: 응, 매개변수 갱신 최적화는 모델이 학습을 통해 손실 함수를 최소화하는 과정에서 중요한 역할을 해. 여러 최적화 알고리즘을 사용하여 이 과정을 진행하는데, 그 중에서도 대표적인 것이 확률적 경사 하강법(SGD)이야.

정호: 너가 알고있는 SGD에 대해 좀 더 설명해 줄수 있을까?

준형: 알겠어! SGD는 매개변수를 갱신할 때 전체 데이터가 아니라 랜덤하게 선택한 하나의 데이터만을 사용하여 기울기를 계산하고, 이를 통해 매개변수를 업데이트하는 방식이야. 이러한 방식은 계산이 빠르고 메모리를 적게 사용한다는 장점이 있어.

준형: 하지만 SGD의 단점이 있다는거 알고있니?

정호: 그건 나도 알고있어! SGD는 각 데이터에 대한 기울기를 계산하여 매개변수를 갱신하기 때문에 불안정한 움직임을 보일 수 있고, 지역 최솟값에 갇힐 수도 있지. 또한 수렴이 느리다는 단점도 있어.

준형: 대단해 정호야! 그러면 모멘텀은 어떻게 도와주는 건지 설명 가능하니?

정호: 응 한번 설명해볼게. 모멘텀은 이전 기울기의 영향을 고려하여 갱신할 방향과 거리를 조절해주는 방식이야. 이전에 이동했던 방향과 일정 비율만큼 현재의 기울기 방향을 고려하여 새로운 방향으로 이동하게 돼. 이를 통해 SGD보다 안정적으로 수렴하고 지역 최솟값에서 벗어나는 데 도움이 되고있어.

정호: 준형아! 모멘텀을 코드로 구현할 수 있을까?

준형: 응 당연하지! 모멘텀을 코드로 구현할 수 있어.

```
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
```

```
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum * self.v[key] - self.lr * grads[key]
            params[key] += self.v[key]
```

정호: 아하, 모멘텀을 이렇게 구현할 수 있구나. 그럼 AdaGrad에 대해서 알아볼까?

정호: 계속 내가 설명해볼게. AdaGrad는 학습률을 각 매개변수에 맞게 조정하여 학습을 진행하는 방식이야. 기존의 경사 하강법과 달리 학습률이 각 매개변수에 따라 동적으로 조절되기 때문에 학습이 더욱 효율적으로 이루어져.

준형: 그럼 이번에도 내가 AdaGrad를 코드로 구현해볼게?

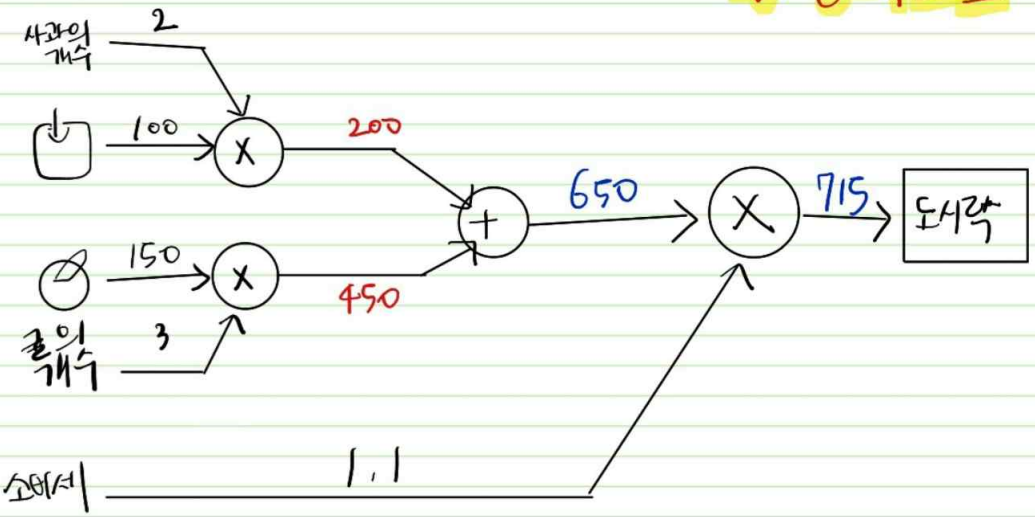
```
class AdaGrad:
    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] += grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)  # 1e-7은 0으로 나누는 것을 방지하기 위한 아주 작은 값
```

정호: 코드로 이해하니 더 쉽고 배움이 빠른거같아!

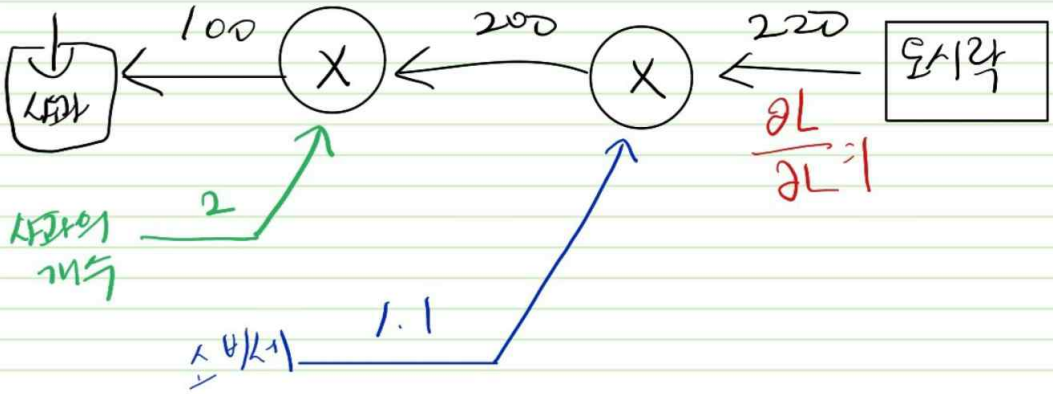
준형: 나도 설명하면서 이해할수 있어 더 좋았어. 다음에 또 해보자 정호야~

*계산그래프 문제2

*정리노트



*계산그래프 역전파

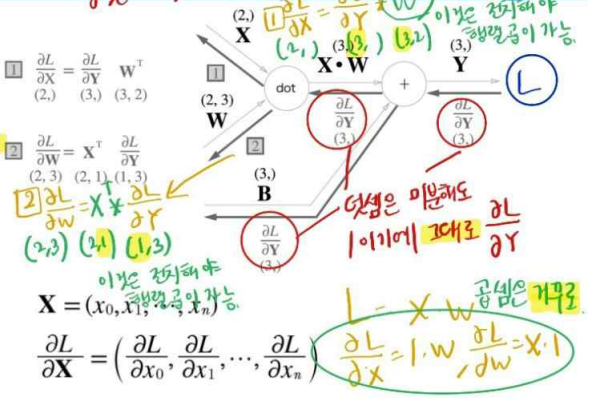


5.6.1 Affine 계층 (2)

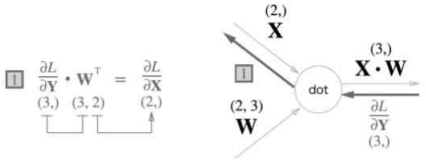
- **Affine 계층의 역전파**

*정리노트

- 각 변수의 형상에 주의해서 봄
- X와 $\frac{\partial L}{\partial X}$은 같은 형상(shape)
- W와 $\frac{\partial L}{\partial W}$도 같은 형상임



- 행렬 곱의 역전파는 행렬의 대응하는 차원의 원소 수가 일치하도록 곱을 조립하여 구함



jupyter week7_0417 Last Checkpoint: 25 days ago

File Edit View Run Kernel Settings Help

Code

[1]:
```
class MulLayer:
    def _int_(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out
    def backward(self, dout):
        dx = dout * self.y # x와 y를 바꾼다.
        dy = dout * self.x

        return dx, dy
```

[4]:
```
apple = 100
apple_num = 2
tax = 1.1

# 계층들
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

print(apple_price) #200
print(price) #220

# 역전파
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

print(dapple, dapple_num, 'dtax : ' , dtax)
```

```
200
220.00000000000003
2.2 110.00000000000001 dtax :  200
```

jupyter week9_0501 Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help

Code

[9]:
```
import numpy as np

X = np.random.rand(2)
W = np.random.rand(2,3)
B = np.random.rand(3)
print(X.shape)
print(W.shape)
print(B.shape)

Y = np.dot(X, W) + B
print(Y)
```

```
(2,)
(2, 3)
(3,)
[1.21381343 1.63866725 1.68176321]
```

[11]:
```
import numpy as np

X_dot_W = np.array([[0, 0, 0], [10, 10, 10]])
B = np.array([1, 2, 3])
Y = X_dot_W + B
print(Y)
```

```
[[ 1  2  3]
 [11 12 13]]
[12 14 16]
```

jupyter week9_0501 Last Checkpoint: 11 days ago

File Edit View Run Kernel Settings Help

Code

[17]:
```
dY = np.array([[1, 2, 3], [4, 5, 6]])
dB = np.sum(dY, axis=0)
print(dB)
dB1 = np.sum(dY, axis=1)
print(dB1)
```

```
[5 7 9]
[ 6 15]
```

[18]:
```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.x = x
        out = np.dot(x, self.W) + self.b
        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)
        return dx
```

[19]:
```
import numpy as np

class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.x = x
        out = np.dot(x, self.W) + self.b
        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)
        return dx

# 예제 입력 데이터
x = np.array([[1, 2]])
# 예제 가중치와 편향
W = np.array([[1, 2], [3, 4]])
b = np.array([1, 1])

# Affine 계층 생성
affine_layer = Affine(W, b)
# Forward pass
out = affine_layer.forward(x)
print("Forward pass 출력:", out)

# 역전파를 위한 미분값 설정
dout = np.array([[1, 1]])
# Backward pass
dx = affine_layer.backward(dout)
print("Backward pass 입력에 대한 미분값:", dx)
print("가중치에 대한 미분값:", affine_layer.dW)
```

```python
# Affine 계층 생성
affine_layer = Affine(W, b)
# Forward pass
out = affine_layer.forward(x)
print("Forward pass 출력:", out)

# 역전파를 위한 미분값 설정
dout = np.array([[1, 1]])
# Backward pass
dx = affine_layer.backward(dout)
print("Backward pass 입력에 대한 미분값:", dx)
print("가중치에 대한 미분값:", affine_layer.dW)
print("편향에 대한 미분값:", affine_layer.db)
```

```
Forward pass 출력: [[ 8 11]]
Backward pass 입력에 대한 미분값: [[3 7]]
가중치에 대한 미분값: [[1 1]
 [2 2]]
편향에 대한 미분값: [1 1]
```

22]:
```python
import numpy as np

class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None
        self.t = None

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shpae[0]
        dx = (self.y - self.t) / batch_size
        return dx
```

jupyter week10_0508 Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help

Code

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

# 시그모이드 함수 정의
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# ReLU 함수 정의
def ReLU(x):
    return np.maximum(0, x)

# 하이퍼볼릭 탄젠트 함수 정의
def tanh(x):
    return np.tanh(x)

# 1000개의 데이터를 가진 100차원의 입력 데이터 생성
input_data = np.random.randn(1000, 100)

# 각 은닉층의 노드(뉴런) 수
node_num = 100

# 은닉층이 5개
hidden_layer_size = 5

# 각 은닉층의 활성화값(활성화 결과)을 저장할 딕셔너리
activations = {}

# 초기 입력값 설정
x = input_data

# 은닉층을 순회하며 활성화 함수 적용
for i in range(hidden_layer_size):
    if i != 0:
        # 첫 번째 은닉층이 아니라면, 이전 층의 활성화값(x)을 현재 입력값으로 사용
        x = activations[i-1]

    # 현재 층의 가중치(w)를 초기화
    node_num = 100
```

jupyter week10_0508 Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help

Code

```
        x = activations[i-1]

    # 현재 층의 가중치(w)를 초기화
    node_num = 100

    # w = np.random.randn(node_num, node_num) * 0.01
    # w = np.random.randn(node_num, node_num)/ np.sqrt(node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)

    # 선형 변환(a) 후 활성화 함수(sigmoid)를 적용하여 활성화값(z) 계산
    a = np.dot(x, w)
    #z = sigmoid(a)
    z = ReLU(a)
    # z = tanh(a)

    # 활성화값(z)을 activations 딕셔너리에 저장
    activations[i] = z

# 각 은닉층의 활성화값 분포를 히스토그램으로 시각화
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)  # 여러 개의 서브플롯 중 하나 선택
    plt.title(str(i+1) + "-layer")  # 서브플롯의 제목 설정
    plt.hist(a.flatten(), 30, range=(0,1))  # 히스토그램 그리기
plt.show()  # 그래프 출력
```

jupyter week10_0508 Last Checkpoint: 4 days ago

File Edit View Run Kernel Settings Help

Code

```
    # 활성화값(z)을 activations 딕셔너리에 저장
    activations[i] = z

# 각 은닉층의 활성화값 분포를 히스토그램으로 시각화
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)  # 여러 개의 서브플롯 중 하나 선택
    plt.title(str(i+1) + "-layer")  # 서브플롯의 제목 설정
    plt.hist(a.flatten(), 30, range=(0,1))  # 히스토그램 그리기
plt.show()  # 그래프 출력
```