인공지능개론 정리노트 #3
ICT융합공학부 202204010 공성택

5장

Q. 신경망 학습에서 수치 미분과 비교했을 때, 오차역전파법을 사용하는 이유는?

A. 신경망 학습을 할 때 가중치 매개변수의 기울기를 구하는데, 이때 수치 미분은, 번거롭고, 오래걸리며, 오차가 있을 수 있다. 하지만 오차역전파법을 사용하면 해석적인 미분을 하기 때문에 수치 미분보다 정확한 결과를 얻을 수 있고, 한 번의 순전파와 역전파로 모든 매개변수에 대한 기울기를 구할 수 있기 때문에 더 빠르고 효율적이다. 때문에 깊은 신경망으로 갈수록 수치 미분보다 오차역전파법을 사용하는 것이 더 효율적이고 바람직하다.

Q. 신경망의 계층을 많이 늘릴 수록 학습 결과의 정확도가 높아지는가?

조사한 결과, 이론적으로 hidden layer가 많을 수록 비선형, 다차원으로 정교하게 학습이 가능하지만, 역전파 알고리즘으로 심층 신경망을 학습시킬 때, 기울기 손실과, 과적합 등이 일어나기 쉬워지기 때문에 무작정 늘리는 것은 오히려 학습에 악영향을 줄 수 있다. 특히, 신경망의 계층을 늘리면 연결 가중치도 많아지기 때문에 더 복잡한 학습 모델이 된다. 그렇게 되면 간단한 문제에서 더 복잡한 모델은 과적합을 유발할 수 있다. 이번에 과제를 하면서 느낀 부분이지만, 계층을 많이 늘릴 수록 하이퍼 파라미터를 조절하여 최적화 하는 데에 어려움을 느꼈다. 그리고 하드웨어의 성능도 더욱 잡아먹는다는 것을 의미하기 때문에 정확도를 높이기 위해 무작정 계층을 늘리는 것보다 적절한 하이퍼 파라미터를 튜닝하는 것이 바람직하다고 느꼈다.

6장

Q. Adam이 AdaGrad와 모멘텀의 이점을 조합한 알고리즘이라고 했는데, 그렇다면 Adam만 사용하면 되는 것 아닌가?

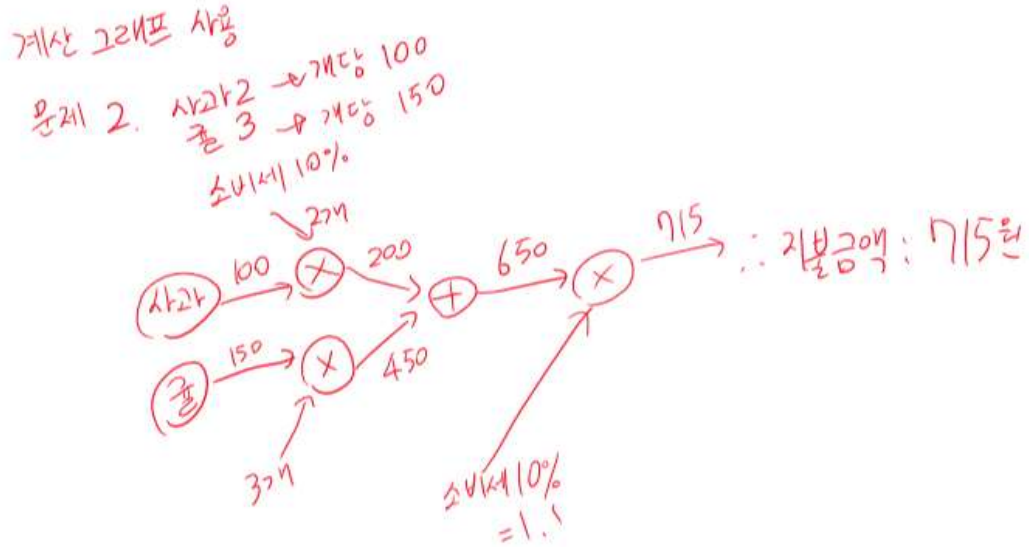
A. Adam은 모멘텀과 유사하게 SGD와 다르게 최솟값에서 탈출할 수 있고, AdaGrad와 유사하게 학습률을 매개변수마다 조정해 희소한 그래디언트를 처리할 수 있다. 그러나 이런 장점을 모두 가지고 있다 해도 모든 상황에서 다른 최적화 알고리즘보다 좋은 것은 아니다. 조사한 결과, Adam은 일부 문제에서 과도한 학습률 갱신으로 인해 학습 과정을 불안정하게 할 수 있다. 또한, Adam은 메모리 사용량이 크다는 단점도 있다. 따라서 간단한 최적화에는 다른 알고리즘을 사용하는 것이 더 적합할 수 있다.

Q. 배치 정규화에 대해서 사용하면 좋은 점만 배웠는데, 배치 정규화의 단점도 있을까?

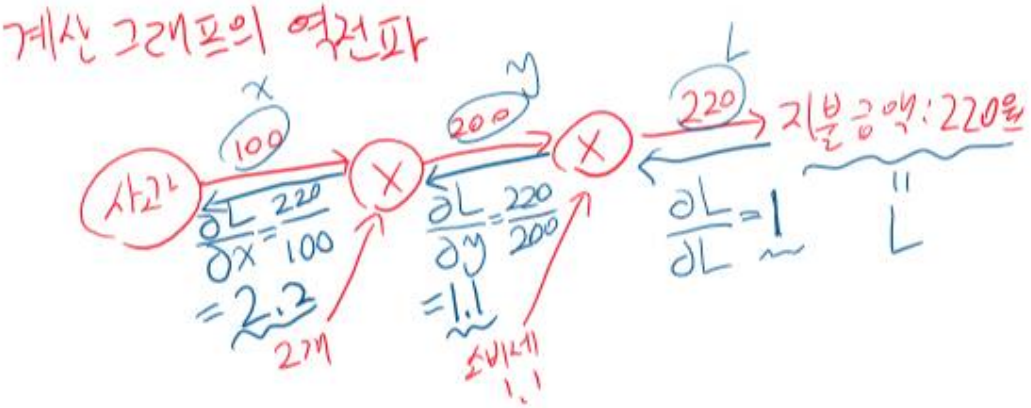
A. 조사한 결과, 배치 정규화는 각 미니배치에 평균과 분산을 정규화해서 추가적으로 계산하므로, 신경망이 클수록 학습 속도가 느려질 수 있다. 때문에 작고 간단한 모델에서는 사용하지 않는 것이 복잡도를 줄일 수 있다. 하지만 이외의 경우에는 일반적으로 심층 신경망에서 학습 속도를 개선할 수 있고, 그래디언트 소실을 완화하고, 가중치 초기화에서 자유롭기 때문에 많은 경우에서 성능 향상을 기대할 수 있다.

모델에 따라서 또 사용하는 것보다 안하는 것이 더 좋은 경우도 있기 때문에 무작정 사용하는 것보다는 경우를 잘 판단해서 적절하게 사용하면 성능 향상에 매우 큰 도움을 줄 것이라고 조사했다.

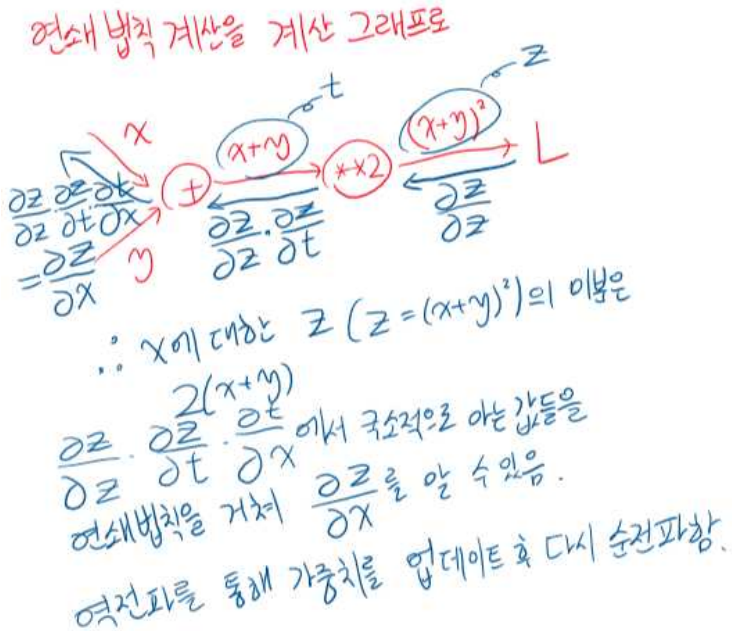
계산 그래프 사용(문제 2)



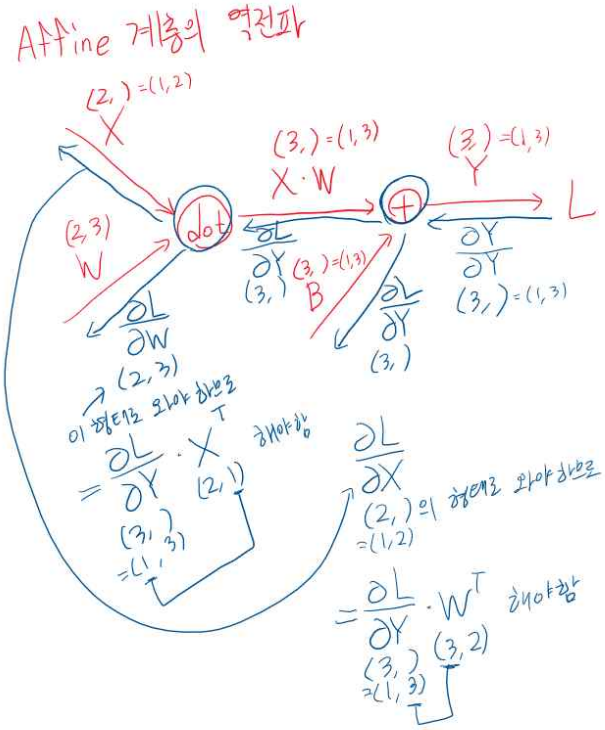
계산 그래프의 역전파



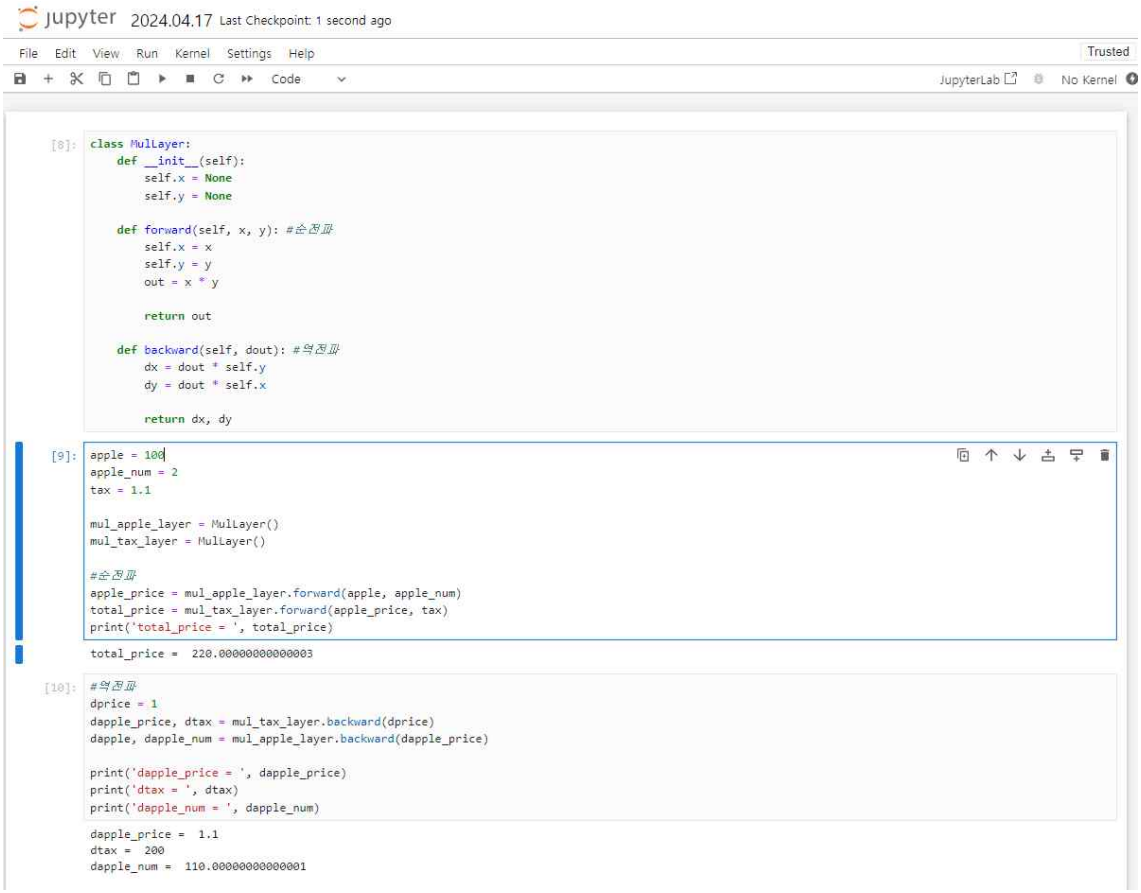
연쇄 법칙 계싼을 계산 그래프로 해보기



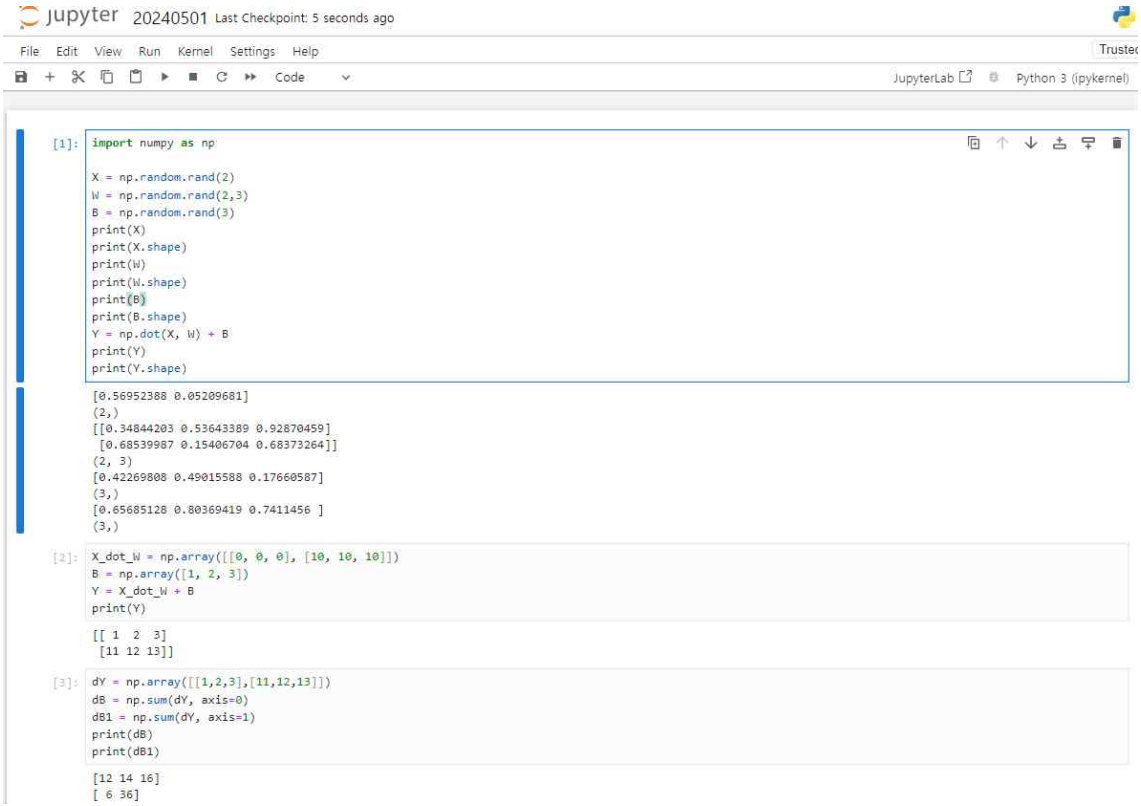
Affine 계층의 역전파 해보기



5장 수업시간 실습(1)


jupyter 2024.04.17 Last Checkpoint: 1 second ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
No Kernel
[8]:
class MulLayer:
def __init__(self):
self.x = None
self.y = None
def forward(self, x, y): #순전파
self.x = x
self.y = y
out = x * y
return out
def backward(self, dout): #역전파
dx = dout * self.y
dy = dout * self.x
return dx, dy
[9]:
apple = 100
apple_num = 2
tax = 1.1
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()
#순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
total_price = mul_tax_layer.forward(apple_price, tax)
print('total_price = ', total_price)
total_price = 220.00000000000003
[10]:
#역전파
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)
print('dapple_price = ', dapple_price)
print('dtax = ', dtax)
print('dapple_num = ', dapple_num)
dapple_price = 1.1
dtax = 200
dapple_num = 110.00000000000001

5장 수업시간 실습(2)



[1]:
```
import numpy as np

X = np.random.rand(2)
W = np.random.rand(2,3)
B = np.random.rand(3)
print(X)
print(X.shape)
print(W)
print(W.shape)
print(B)
print(B.shape)
Y = np.dot(X, W) + B
print(Y)
print(Y.shape)
```

```
[0.56952388 0.05209681]
(2,)
[[0.34844203 0.53643389 0.92870459]
 [0.68539987 0.15406704 0.68373264]]
(2, 3)
[0.42269808 0.49015588 0.17660587]
(3,)
[0.65685128 0.80369419 0.7411456 ]
(3,)
```

[2]:
```
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]]
```

[3]:
```
dY = np.array([[1,2,3],[11,12,13]])
dB = np.sum(dY, axis=0)
dB1 = np.sum(dY, axis=1)
print(dB)
print(dB1)
```

```
[12 14 16]
[ 6 36]
```

[4]:
```
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, out):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)
        return dx
```

[7]:
```
import numpy as np
from common.functions import softmax, cross_entropy_error

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.shape[0]
        bx = (self.y - self.t)/batch_size
        return dx
```

6장 수업시간 실습

jupyter 20240508 Last Checkpoint: 3 seconds ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab No Kernel

[12]:
```
# coding: utf-8
import numpy as np
import matplotlib.pyplot as plt


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


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


def tanh(x):
    return np.tanh(x)

input_data = np.random.randn(1000, 100)  # 1000개의 데이터
node_num = 100  # 각 은닉층의 노드(뉴런) 수
hidden_layer_size = 5  # 은닉층이 5개
activations = {}  # 이곳에 활성화 결과를 저장

x = input_data

for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    # w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    # 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)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    # z = sigmoid(a)
    z = ReLU(a)
    # z = tanh(a)

    activations[i] = z
```

```
# 히스토그램 그리기
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)
    plt.title(str(i+1) + "-layer")
    if i != 0: plt.yticks([], [])
    # plt.xlim(0.1, 1)
    # plt.ylim(0, 7000)
    plt.hist(a.flatten(), 30, range=(0,1))
plt.show()
```