인공지능개론 정리노트#2

201904022 김상옥

202284026 안정빈

질문1. ReLu계층과 Sigmoid계층을 Affine계층과 따로 넣는 이유와 이 둘의 차이점

김상옥: Afiine계층 내부에 ReLu, Sigmoid와 같은 활성화 함수를 같이 넣어서 저장하거나 계산하면 더 편리할 거 같은데 왜 굳이 Affine계층과 활성화 함수 계층을 나누었을까?

안정빈: Affine 계층과 활성화 함수 계층을 분리하는 것은 단순히 편리함을 위한 것이 아니라, 신경망 모델의 설계, 학습, 해석, 유연성 등 다양한 측면에서 중요한 의미를 가지고 있다고 생각해. 그래서 분리해서 사용하는 것 같아. 구분하면 각 계층을 이해하기 쉽지 않을까?

김상옥: 생각해보니 따로 클래스로 만들어 두어서 나중에 그냥 불러오기만 하면 되니까 또 신경망 구조와 구조에 맞는 순전파와 역전파 계산을 만들어 두면 굳이 Affine 계층에 데이터를 다 넣지 않더라도 크게 불편한 거는 없겠다. 그리고 Affine계층에 활성화 함수를 넣으면 나중에 활성화 함수만 바꾸기에는 귀찮고 번거로울 거 같다. 편리성 이외에 더 장점이 있을까?

안정빈: 각 계층의 역할이 명확하게 분리되니까 코드를 이해하기 쉬울 것 같아. 그래서 복잡한 신경망을 다룰 때 더 유용할 것 같아. 그리고 코드의 모듈성이 높아지고 재사용하기 쉬울 것 같고, 계산의 효율성도 높아질 것 같아. 계산 과정을 병렬화 하여 처리 속도를 높일 수 있을 것 같아. 그러면 ReLu계층과 Sigmoid계층의 차이점은 어떻게 생각해?

김상옥: 기존에 배웠던 활성화 함수 ReLu와 Sigmoid를 클래스로 만들어 순전파와 역전파의 과정을 작성해둔 거니까 두 계층의 차이점이라고 하면 활성화 함수 ReLu와 Sigmoid의 차이점이라고 생각해. 각 용도에 맞는 활성화 함수 계층을 사용하면 될 거고 미리 순전파와 역전파에 대한 수식들을 함수로 만들어두었으니까 실제 학습할 때 해당 객체에 대해서 함수만 불러오면 편리하게 사용할 수 있을 거야. 시그모이드를 역전파하는 건 역으로 다 계산하는 방식이라고 어느정도 예상은 했었어. 나는 특히 ReLu계층이 특이했었는데 ReLu는 0이하거나 x값 그대로를 리턴하는 활성화 함수로 이 계층을 어떻게 역전파로 적용할지 궁금했는데 True, False로 된 배열을 통해서 역전파를 할 때 같은 행렬 부분의 bool값을 통해 구분하는 점이 특이했었어.

결론: ReLu계층과 Sigmoid계층을 Affine계층과 따로 넣는 이유는 코드의 이해도가 높아지고 코드의 재사용, 계산의 효율성이 높아지고 처리속도가 높아지는 장점이 있어서 사용하는 것 같습니다. 두 계층은 활성화 함수에 대한 순전파와 역전파를 모듈화 해둔 것이기에 차이점이라고 하면 활성화 함수ReLu와 Sigmoid의 차이로 용도에 맞는 함수를 사용하면 좋을 것 같습니다.

질문2. 오차역전파법의 수렴 속도와 신경망 구조의 관계는?

안정빈: 오차역전파법은 신경망 모델 학습에 필수적인 알고리즘으로, 입력값과 출력값 사이의 오차를 계산하여 모델의 가중치를 조정하는 과정을 반복적으로 수행하는데 이 과정에서 수렴 속도와 신경망 구조 관계는 어떻게 될까?

김상옥: 오차역전파법이 수치 미분보다 더 정확하고 빠르다고 배웠고, MNIST를 실습하면서 직접 해보니 확실히 더 빠르다고 느껴지기도 했어. 그런데 간단한 MNIST 학습에서도 나는 생각보다 빠르지는 않다고 느껴졌었는데 이 신경망 구조와 기타 수식들이 더 복잡해지고 그만큼 자원을 많이 필요로 하는 학습을 진행한다면 오차역전파법도 물론 빠른 학습이겠지만 사람이 느끼기에는 빠르다고 느껴지지는 않을 거 같다. 그만큼 불필요한 노드나 수식, 함수를 최대한 줄이면서도 효율적인 구조와 함수를 사용하는 것이 학습 속도를 높이는데 필요하다고 생각해. 그럼 신경망에서 효율적인 구조라는게 있을까?

안정빈: 네트워크의 적절한 깊이와 너비를 선택하여 구성하고, 적절한 활성화 함수와 알고리즘을 선택해서 학습 속도와 성능에 영향을 줄 수 있다고 생각해. Adam, SGD 등 알고리즘 중에 제일 적합 한 것을 선택하는 방법이 있고, 과적합을 방지하고 학습속도를 높이는데 도움을 주기위해 배치정규화를 사용하고 하이퍼파라미터 값을 조정하여 신경망 구조를 효율적으로 만들 수 있다고 생각해.

김상옥: 전에 토론했던 신경망 너비와 깊이에 대해서 결국 해결하고자 하는 문제에 맞는 구조를 만들어야 한다고 결론을 냈었어. 이것도 마찬가지로 모든 문제에 대한 완벽한 구조라는 건 없으니까 문제에 대해 적절한 너비와 깊이를 선택하고, 구조를 계속 변경해 나가면서 최적의 신경망 구조와 다양한 함수를 적용해야 할거야. 이번에 과제를 한 것처럼 계층과 하이퍼피라미터를 수정해보면서 확실히 아주 작은 차이라도 학습에 영향을 많이 준다고 느꼈어. 사실 속도부분에서는 간단한 학습 과제라서 그런지 크게 차이는 못 느꼈는데 분명 더 큰 신경망을 구성한다고 하면 분명 속도나 학습에 대해서 아주 작은 부분에서부터 사용하는 함수, 수식, 구조에 많은 영향을 받을 거 같아. 물론 오차역전파법의 학습 속도를 포함하여 전체적인 학습 속도는 효율적인 구조와 함수도 영향이 있겠지만 더 중요한 것은 하드웨어의 성능이 중요하다고 생각해.

안정빈: 맞다고 생각해. 신경망 학습 속도는 모델 구조, 함수, 하이퍼파라미터 뿐만 아니라 하드웨어의 성능도 고려하는 것이 중요한 것 같아. 하드웨어 성능이 지속적으로 향상되고 있고, 다양한 하드웨어 가속기술을 활용하면 신경망 학습속도를 더욱 높일 수 있을 것 같아. 효율적인 알고리즘 개발과 병렬 처리 기술 등을 활용하면 더욱 빠르고 효율적인 신경망 학습이 가능해질 것 같아.

김상옥: 아직 배우지 않은 부분이 너무나도 많고, 이제 수치 미분과 오차역전파법을 배웠기에 얼마나 많은 학습 관련 함수나 수식들이 있을 지는 잘 모르지만 이런 작은 부분부터 이해하고 간다면 나중에 신경망 학습을 할 때가 있으면 분명 도움이 많이 될 거 같다.

결론: 오차역전파법을 이용하거나 다른 학습 방법에 대해서 신경망 학습 속도는 모델 구조, 함수, 하이퍼피라미터를 효율적이고 얼마나 최적화 하는 것이 중요하다. 또한 대규모 신경망 구조라면 하드웨어 성능도 매우 중요한 부분일 것이다.

질문3. 배치 정규화를 적용할 때 고려해야 할 하이퍼파라미터들은 무엇이며, 이들을 조정하는 최적화 전략은 어떻게 될까?

김상옥: 우선 배치 정규화는 순전파를 진행하면서 계층 사이에 배치 정규화를 통해 학습 속도와 성능을 높이는 목적으로 사용한다고 해. 계산된 값들이 너무 양쪽 혹은 한쪽으로 치우쳐져 있는 게 학습에 안좋은 영향을 주기 때문에 이전에는 Xavier, He 초기화를 사용하여 가중치들을 어느정도 분산되게 하여 계산되는 값들이 다양한 값을 가지게 했었어. 배치 정규화는 계층간 계산된 값들을 계층 사이에서 분산시켜 주는 역할을 하여 최적화를 진행하기 때문에 이 배치 정규화를 적용할 때 고려할 하이퍼파라미터가 있다면 가중치를 초기화하는 부분이라고 생각해. 물론 수업에서는 배치 정규화를 하면 가중치 초기화에 대한 문제를 해소할 수 있다고 들었지만 그래도 미리 가중치를 어느정도 최적화를 한 후 배치 정규화를 하면 더 좋아지지 않을까? 또 신경망의 깊이도 매우 중요한 하이퍼파라미터라고 생각해. 이외에는 어떤 것이 고려할 부분일까?

안정빈: 배치 정규화의 모멘텀 파라미터, 학습률, 정규화 강도, 배치 크기등 고려해야할 부분들이 더 있다고 생각해. 예를들어 모멘텀 값을 조절해서 학습 속도와 안정성을 조절하고, 배치 정규화의 스케일 파라미터와 이동 파라미터를 학습하는 데 사용되는 학습률을 결정할 때 너무 큰 학습률은 수렴을 방해할 수있고, 너무 작으면 학습이 느려질 수 있는 등 문제가 있을 수 있어서 적절한 학습률을 고려해야 해. 그리고 정규화 강도는 배치 정규화에서 분모에 더해지는 값으로, 분산이 0에 가까워지는 것을 방지하는데 너무 작거나 큰값은 불안정성 혹은 약화시킬 수있어서 이또한 고려해야해. 이런 다양한 고려해야할 부분들이 있고 이를 통해 모델의 학습에 영향을 주는 것같아. 이들을 조정하는 최적화 전략은 무엇이 있을까?

김상옥: 아직 내가 배치 정규화가 어떤 방식으로 돌아가는 지 완벽하게 이해하지는 못했지만 수식에 적용되는 수치들도 배치 정규화에 영향을 줄 거 같아. 좋은 구조와 함수, 배치 정규화를 적용했다 하더라도 하이퍼파라미터의 값에 영향을 받기 때문에 이 하이퍼파라미터를 최적화하는 것도 중요할거야. 이 하이퍼파라미터를 자동으로 구할 수 있는 방법이나 수식 같은 것이 있지 않을까? 가중치를 적절하게 조절하듯 하이퍼파라미터로 적절하게 자동으로 구할 수 있으면 더 편리하고 그만큼 최적화가 잘 된 값들을 얻을 수 있을 거 같아. 이에 대해 어떻게 생각해?

안정빈: 의견에 대해 동의해. 배치 정규화 하이퍼파리미터를 자동화하는 것은 모델 성능을 향상시키는데 도움이 될 것 같아. 하이퍼파라미터를 자동화하고 최적의 값을 찾는 방법들 중에서는 그리드 탬색, 랜덤탐색 등이 있는 것같아. 그리드 탐색은 모든 하이퍼파라미터의 가능한 조합에 대해 모델을 학습하고 성능을 평가하여 최적의 조합을 찾는 방법이고, 랜덤 탐색은 하이퍼파라미터 공간에서 임의의 포인트를 선택하여 모델을 학습하고 성능을 평가하는 방법이야. 이렇게 다양한 방법들을 사용해서 하이퍼파라미터 최적화를 자동화할 수있어. 그러면 최적화 전략은 자동화 하는 방법일까?

김상옥: 하이퍼 피라미터에 대해서도 이미 누가 만들어 둔 방법이 있구나. 가중치를 자동으로 구하는 것 처럼 하이퍼 피라미터도 자동으로 구하는 편이 효율적이라고 생각해. 결국에는 수많은 경우의 수에서 좋은 값들을 찾기 위해서는 그만큼 반복이 필요할 것인데 사람이 수동으로 하기에는 인적 자원과 더불어 매우 비효율적일거야. 자동화를 통한 값을 찾는 방법이 효율면에서나 여러가지 측면에서 이점이 있다고 생각해. 또 과제를 하면서 직접 구하는게 생각보다 귀찮고 힘든 부분이었는데 더 큰 규모의 신경망을 학습한다고 하면 정말 많은 하이퍼피라미터를 고려해야 할 거 같은데 자동화를 통한 최적화 전략이 좋을 거 같다.

결론: 배치 정규화를 적용할 때 고려해야 할 하이퍼파라미터들은 신경망의 깊이, 모멘텀 파라미터, 학습률 등 다양한 부분들이 있고, 이들을 조정할 때 자동화를 통한 최적화 전략이 좋다고 생각합니다.

질문4. AI가 그림에 사람 얼굴은 없지만 사람 얼굴의 형상이 보이는 착시 그림을 구분할 수 있을까?

안정빈: 나는 구분할 수 있다고 생각해. AI 모델의 종류와 학습된 데이터에 따라 결과가 달라질 것 같아. 착시 그림은 일반 사람도 구분하기 힘들어 하는데, AI가 이러한 패턴을 구분하는 것은 매우 어려울 것 같아. 딥러닝 모델과 충분한 학습 데이터를 통하면 가능할 것 같은데 어떻게 생각해?

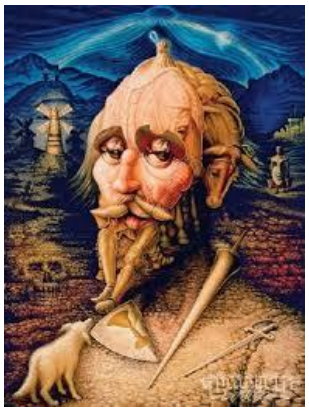
김상옥: 착시를 학습한 AI라면 구분할 수 있다고 생각은 하는데 착시 그림이 매우 불규칙적이고 다양해서 그것도 정확히 구분할 수 있다고는 생각을 하지는 않아. 결국 AI는 이미지 행렬 값을 통해서 사물을 구분할 것인데 착시 그림은 다양한 사물을 적절한 위치와 구도, 색을 이용하여 없지만 있는 것처럼 보이게 하기 때문에 단순히 행렬의 값만 가지고는 구분하기는 힘들 지 않을까? 물론 이 경우에는 일반적인 이미지 구분 AI일 경우이고, 착시 그림을 학습했다면 어느정도는 구분할 수 있을 거 같아. 하지만 불규칙적인 착시 그림을 정확하게 구분하려면 상당한 양의 학습 데이터가 필요하지 않을까? 그렇지 않으면 학습했다고 하더라도 정확도가 떨어질 거라고 생각해. 그럼 단순히 강아지나 고양이, 사람 등 간단한 동물을 구분하는 AI가 착시 그림을 구분할 수 있을까?

안정빈: 착시 그림의 특징이 시각의 한계를 이용해서 실제와 다른 이미지가 보이도록 하는 그림이고, 물리적 착시, 인지적 착시, 심리적 착시 등 다양한 유형이 있고 실제 인간은 착시 그림을 인식하는데 어려움이 있어. 그런데 단순 동물 구분 AI의 특징은 강아지, 고양이, 사람 등 간단한 동물의 이미지를 학습해서 동물을 구분하는 기능을 수행하고, 이미지 색상, 형태, 질감 등으로 특징을 분류하고, 학습된 이미지 특징과 패턴을 기반으로 새로운 이미지를 분석하고 동물을 구분하는데 착시 그림은 실제 물체의 배치나 조명 등으로 시각적 오류를 유발하는데 단순 동물 구분 AI가 이미지 물리적 특징과 단순한 특징에만 집중하면 구분하는데 어려움이 있을 것 같아. 데이터를 아무리 늘려도 좀 어려울 것 같다는 생각이 들어. 해결할 수 있는 방법이 있을까?

김상옥: 지금에서는 힘든 부분이라고 생각하는데, 착시에 대한 특징이 너무 많고, 착시가 되는 원리도 매우 많아서 그 특징들을 모두 학습할 수 있어도 그렇게 구한 신경망의 범용성이 모든 착시 그림을 구분하거나 기대할 정도로 정확도가 나오지는 않을 거 같아. 결국 이미지를 사람처럼 보는 것이 아니라 내부적인 값들을 통해서 학습하는 거라면 어려울 거 같다. 그럼 이미지를 값으로 보는 게 아니라 진짜 사람처럼 보고 판단할 수 있다면 가능하지 않을까? 물론 그 수준까지는 시간이 필요하겠지만 그런 AI가 온다면 그때는 가능할 거 같아.

안정빈: 나도 그렇게 생각해. AI가 이미지를 사람처럼 보고 이해할 수 있는 수준에 도달한다면, 착시 그림을 구분하는 것이 가능성이 있을 것 같아. 지금은 AI가 단순한 형태만 구분하고 있고 인지적인 측면을 이해하고 시뮬레이션 하는 데에는 한계가 있어. 미래에 더 많은 연구와 발전으로 인간을 완벽하게 모방하는 AI모델이 등장한다면 정말 가능할 것 같아.

결론: 단순히 동물, 사물을 구분하는 지금의 AI로는 착시 그림에 있는 것을 구분하는 건 힘들 거 같다. 결국 이미지 행렬을 통해 구분하는 것인데 착시 그림은 다양한 사물과 사물의 각도, 구도, 색 등을 이용하여 없는 것을 있는 것 처럼 보이게 하는 사람의 심리적인 요인도 분명 있기 때문에 지금의 AI로는 구분하기 힘들 것이다. 또한 착시 그림을 학습한 AI라고 한들 착시가 되는 경우와 그림들이 매우 불규칙적이고 다양하기 때문에 학습한다고 해도 정확도를 기대하기는 힘들 것으로 생각한다. 추후 미래의 AI가 "사람처럼" 이미지를 본다면 그때는 가능할 거라고 생각합니다.



안정빈

Affine 계층

```
import numpy as np
```

```
X= np.random.rand(2) #오차
W= np.random.rand(2,3) #가중치
B= np.random.rand(3) #편향
```

```
X.shape
```

```
(2,)
```

```
W.shape
```

```
(2, 3)
```

```
B.shape
```

```
(3,)
```

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

```
Y.shape
```

```
(3,)
```

X,W,B는 각각 형상이 (2,),(2, 3),(3,) 인 다차원 배열입니다. 뉴런의 가중치 합은 Y=np.dot(X,W)+B처럼 계산 Y를 활성화 함수로 변환해 다음 층으로 전파하는 것이 신경망 순전파의 흐름이다.

행렬의 곱에서는 대응하는 차원의 원소 수를 일치시킵니다.

신경망의 순전파 때 수행하는 행렬의 곱은 기하학에서는 어파인 변환 이라고 합니다. 어파인 변환을 수행하는 처리를 Affine 계층 이라는 이름으로 구현합니다.

배치용 Affine 계층

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

```
X_dot_W
```

```
array([[ 0,  0,  0],
       [10, 10, 10]])
```

```
X_dot_W+B
```

```
array([[ 1,  2,  3],
       [11, 12, 13]])
```

오차역전파법을 적용한 신경망 구현하기

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

x_batch = x_train[:3]
t_batch = t_train[:3]

grad_numerical = network.numerical_gradient(x_batch, t_batch)
grad_backprop = network.gradient(x_batch, t_batch)

# 각 가중치의 절대 오차의 평균을 구한다.
for key in grad_numerical.keys():
    diff = np.average( np.abs(grad_backprop[key] - grad_numerical[key]) )
    print(key + ":" + str(diff))
```

```
W1:5.398688478090664e-10
b1:3.2650435077851737e-09
W2:7.627149721649046e-09
b2:1.4077597242229123e-07
```

배치용 Affine 계층에서 편향의 순전파 역전파

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

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

        return out

    #역전파
    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
```

Softmax-with-Loss 계층

```
class SoftmaxWithLoss:
    def __init__(self):
        self.loss =None # 손실
        self.y = None #softmax의 출력
        self.t = None #정답 레이블(원-핫 벡터)

    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]
        dx = (self.y - self.t) / batch_size

        return dx
```

```
class MulLayer:
    def __init__(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
        dy=dout*self.x

        return dx,dy
```

```
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(price)
```

```
220.00000000000003
```

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

print(dapple, dapple_num, dtax)
```

```
2.2 110.00000000000001 200
```

오차역전파법을 사용한 학습 구현하기

```
import sys, os
sys.path.append(os.pardir)
```

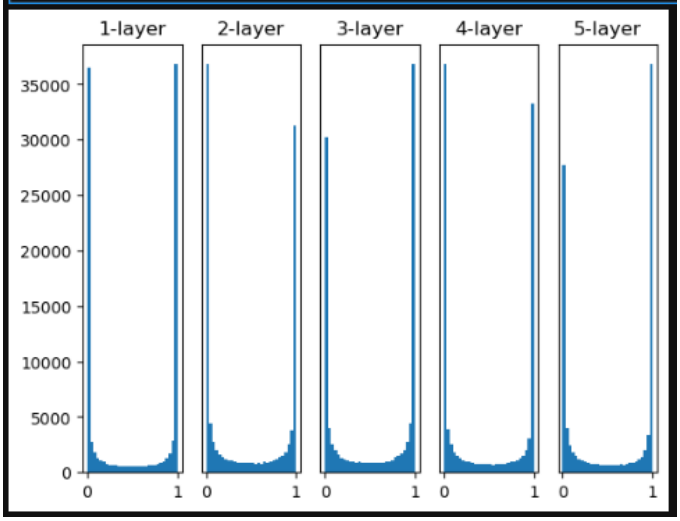
```
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet
```

```
#데이터 읽기
(x_train, t_train), (x_test, t_test) = \
            load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

iters_num = 10000
train_size = x_train.shape[0]
batch_size =100
learning_rate =0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    #오차역전파법 방식
    grad = network.gradient(x_batch, t_batch)

    #갱신
    for key in ('W1','b1','W2','b2'):
        network.params[key] -=learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0 :
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print(train_acc, test_acc)
```

```
0.11238333333333334 0.118
0.9034 0.9075
0.92325 0.9268
0.9363166666666667 0.935
0.94305 0.9405
0.9508666666666666 0.9478
0.9547333333333333 0.951
0.9581333333333333 0.9539
0.9638333333333333 0.9602
0.9661 0.9599
0.9685166666666667 0.963
0.97035 0.9647
0.9729333333333333 0.9657
0.9735833333333334 0.9669
0.97545 0.9669
0.97675 0.9682
0.9781666666666666 0.9691
```

```python
# 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
```

```python
# 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()
```




```python
import os
import sys

sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.util import smooth_curve
from common.multi_layer_net import MultiLayerNet
from common.optimizer import SGD


# 0. MNIST 데이터 읽기==========
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

train_size = x_train.shape[0]
batch_size = 128
max_iterations = 2000


# 1. 실험용 설정==========
weight_init_types = {'std=0.01': 0.01, 'Xavier': 'sigmoid', 'He': 'relu'}
optimizer = SGD(lr=0.01)

networks = {}
train_loss = {}
for key, weight_type in weight_init_types.items():
    networks[key] = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100],
                                  output_size=10, weight_init_std=weight_type)
    train_loss[key] = []
```

```python
# 2. 훈련 시작==========
for i in range(max_iterations):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in weight_init_types.keys():
        grads = networks[key].gradient(x_batch, t_batch)
        optimizer.update(networks[key].params, grads)

        loss = networks[key].loss(x_batch, t_batch)
        train_loss[key].append(loss)

    if i % 100 == 0:
        print("===========" + "iteration:" + str(i) + "===========")
        for key in weight_init_types.keys():
            loss = networks[key].loss(x_batch, t_batch)
            print(key + ":" + str(loss))


# 3. 그래프 그리기==========
markers = {'std=0.01': 'o', 'Xavier': 's', 'He': 'D'}
x = np.arange(max_iterations)
for key in weight_init_types.keys():
    plt.plot(x, smooth_curve(train_loss[key]), marker=markers[key], markevery=100, label=key)
plt.xlabel("iterations")
plt.ylabel("loss")
plt.ylim(0, 2.5)
plt.legend()
plt.show()
```

```
===========iteration:0===========
std=0.01:2.302544767455796
Xavier:2.3117967395778862
He:2.3191493719051977
===========iteration:100===========
std=0.01:2.3024172483891268
Xavier:2.2400323717243653
He:1.3790426576164923
===========iteration:200===========
std=0.01:2.299290101339929
Xavier:2.0599169677630824
He:0.7262553260848184
===========iteration:300===========
std=0.01:2.3014737397159477
Xavier:1.7305601921031397
He:0.5459644302920093
===========iteration:400===========
```

```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.multi_layer_net_extend import MultiLayerNetExtend
from common.optimizer import SGD, Adam

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

# 학습 데이터를 줄임
x_train = x_train[:1000]
t_train = t_train[:1000]

max_epochs = 20
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01


def __train(weight_init_std):
    bn_network = MultiLayerNetExtend(input_size=784, hidden_size_list=[100, 100, 100, 100, 100], output_size=10,
                                     weight_init_std=weight_init_std, use_batchnorm=True)
    network = MultiLayerNetExtend(input_size=784, hidden_size_list=[100, 100, 100, 100, 100], output_size=10,
                                  weight_init_std=weight_init_std)
    optimizer = SGD(lr=learning_rate)

    train_acc_list = []
    bn_train_acc_list = []

    iter_per_epoch = max(train_size / batch_size, 1)
    epoch_cnt = 0

    for i in range(1000000000):
        batch_mask = np.random.choice(train_size, batch_size)
        x_batch = x_train[batch_mask]
        t_batch = t_train[batch_mask]
```

```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from common.multi_layer_net_extend import MultiLayerNetExtend

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = MultiLayerNetExtend(input_size=784, hidden_size_list=[100, 100], output_size=10,
                              use_batchnorm=True)

x_batch = x_train[:1]
t_batch = t_train[:1]

grad_backprop = network.gradient(x_batch, t_batch)
grad_numerical = network.numerical_gradient(x_batch, t_batch)


for key in grad_numerical.keys():
    diff = np.average( np.abs(grad_backprop[key] - grad_numerical[key]) )
    print(key + ":" + str(diff))
```

```python
        for _network in (bn_network, network):
            grads = _network.gradient(x_batch, t_batch)
            optimizer.update(_network.params, grads)

        if i % iter_per_epoch == 0:
            train_acc = network.accuracy(x_train, t_train)
            bn_train_acc = bn_network.accuracy(x_train, t_train)
            train_acc_list.append(train_acc)
            bn_train_acc_list.append(bn_train_acc)

            print("epoch:" + str(epoch_cnt) + " | " + str(train_acc) + " - " + str(bn_train_acc))

            epoch_cnt += 1
            if epoch_cnt >= max_epochs:
                break

    return train_acc_list, bn_train_acc_list
```

```python
# 그래프 그리기==========
weight_scale_list = np.logspace(0, -4, num=16)
x = np.arange(max_epochs)

for i, w in enumerate(weight_scale_list):
    print( "============== " + str(i+1) + "/16" + " ==============")
    train_acc_list, bn_train_acc_list = __train(w)

    plt.subplot(4,4,i+1)
    plt.title("W:" + str(w))
    if i == 15:
        plt.plot(x, bn_train_acc_list, label='Batch Normalization', markevery=2)
        plt.plot(x, train_acc_list, linestyle = "--", label='Normal(without BatchNorm)', markevery=2)
    else:
        plt.plot(x, bn_train_acc_list, markevery=2)
        plt.plot(x, train_acc_list, linestyle="--", markevery=2)

    plt.ylim(0, 1.0)
    if i % 4:
        plt.yticks([])
    else:
        plt.ylabel("accuracy")
    if i < 12:
        plt.xticks([])
    else:
        plt.xlabel("epochs")
    plt.legend(loc='lower right')

plt.show()
```

```
============== 1/16 ==============
epoch:0 | 0.097 - 0.06
epoch:1 | 0.097 - 0.074
C:\Users\komi1\202284026_인공지능\10
ountered in square
  weight_decay += 0.5 * self.weight
C:\Users\komi1\202284026_인공지능\10
e encountered in scalar multiply
  weight_decay += 0.5 * self.weight
epoch:2 | 0.097 - 0.102
epoch:3 | 0.097 - 0.128
epoch:4 | 0.097 - 0.159
epoch:5 | 0.097 - 0.181
epoch:6 | 0.097 - 0.199
epoch:7 | 0.097 - 0.219
epoch:8 | 0.097 - 0.24
```

```
' '
============== 2/16 ==============
epoch:0 | 0.087 - 0.088
epoch:1 | 0.097 - 0.102
epoch:2 | 0.097 - 0.119
epoch:3 | 0.097 - 0.136
epoch:4 | 0.097 - 0.163
epoch:5 | 0.097 - 0.187
epoch:6 | 0.097 - 0.23
epoch:7 | 0.097 - 0.241
epoch:8 | 0.097 - 0.267
epoch:9 | 0.097 - 0.305
epoch:10 | 0.097 - 0.325
epoch:11 | 0.097 - 0.348
epoch:12 | 0.097 - 0.367
epoch:13 | 0.097 - 0.397
epoch:14 | 0.097 - 0.411
epoch:15 | 0.097 - 0.428
```

김상옥

```
[26]: import numpy as np
      import matplotlib.pylab as plt
```

오차역전파법

수치미분의 단점

- 계산 속도가 느리다
- 오차를 가지고 있다

위 단점을 해결하기 위해 오차역전파법 사용

계산 그래프

- 자료구조 그래프를 사용하여 오차역전파법을 이해하기 쉬워짐

계산 그래프 문제2번

- (사과, 100) -> (2개, *2) -> (200원) ——> (사과+귤, 650원) -> (소비세, 10%) --> **715원**
- (귤 , 150) -> (3개, *3) -> (450원) ——/

계산 그래프 역전파

- (사과, 100) <- (2개, *2) <- (?, 200원) <- (소비세, *1.1) <-- **220원**

연쇄 법칙

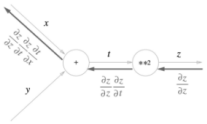
- 합성 함수 : 함수가 여러 개로 이루어진 함수

  x와 1의 미분을 구하기 위해서는
  (x or y)와 t의 미분 -> t와 z의 미분을 곱하여 알 수 있다.

  x->1의 미분을 통해서 x의 증감에 따라 1의 변화를 알 수 있다.
  --> w1의 변화를 통해 1의 변화를 알 수 있다.
  --->

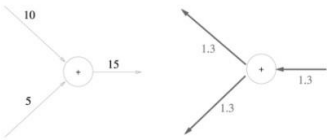
  반대로 z->t 미분, t->x의 미분

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x+y)$$



덧셈 노드의 역전파

- 1.3이라는 값이 오른쪽에서 왔을 때 x에 대한 z 미분을 구하면 1이 나온다
- ■ 1이라는 건 변화가 x값 그 자체라는 의미다




```
[ ]:
```

```
[27]: class MulLayer:
          def __init__(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
              dy = dout * self.x
              return dx, dy
```

순전파 ▽

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

```
[29]: mul_apple_layer = MulLayer()
      mul_tax_layer = MulLayer()
```

```
[30]: apple_price = mul_apple_layer.forward(apple, apple_num)
```

```
[31]: price = mul_tax_layer.forward(apple_price, tax)
```

```
[32]: price
```

```
[32]: 220.00000000000003
```

역전파 ▽

```
[33]: dprice = 1
      dapple_price, dtax = mul_tax_layer.backward(dprice)
```

```
[34]: dapple, dapple_num = mul_apple_layer.backward(dapple_price)
```

```
[35]: dapple_price, dtax, dapple_num, " apple에 대한 L 결과 ---> ", dapple
```

```
[35]: (1.1, 200, 110.00000000000001, ' apple에 대한 L 결과 ---> ', 2.2)
```

```
[36]: # 사과 가격이 1000이 됐을 때 최종 가격
      apple = 1000
      apple * dapple
```

```
[36]: 2200.0
```

```
[37]: # 사과 가격이 2323원이 됐을 때 최종 가격
      apple = 2323
      apple * dapple
```

```
[37]: 5110.6
```

```
[ ]:
```

활성화 함수(렐루, 시그모이드 등)도 위 곱셈과 마찬가지로 구할 수 있다.

강의 9주차 인공지능개론

```
[538]: import numpy as np
       import matplotlib.pylab as plt
```

```
[539]: #시그모이드 활성화 함수
       def sigmoid(x):
           return 1 / 1 + np.exp(-x)
```

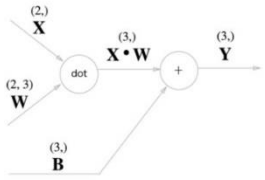
순전파 흐름 ▽

1. 입력 x값
2. 가중치 w값
3. 편향 b값
4. y = np.dot(x, w) + b

```
[540]: x = np.random.rand(2)
       print(x)

       w = np.random.rand(2, 3)
       print(w)

       b = np.random.rand(3)
       print(b)

       y = np.dot(x, w) + b
       print(y)

       print(x.shape, ' x ',w.shape, ' + ',b.shape, ' = ',y.shape)
```

```
[0.9531788  0.25671633]
[[0.42492413 0.66099088 0.33818395]
 [0.49112389 0.30588661 0.50163431]]
[0.91613382 0.39779301 0.48729946]
[1.44724203 1.10636159 0.93842696]
(2,)  x  (2, 3)  +  (3,)  =  (3,)
```

순전파 흐름 그래프

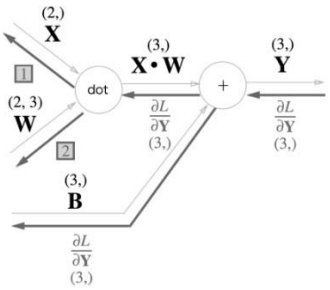


역전파 흐름 그래프

- 덧셈 노드의 역전파는 미분이 1이기에 그대로 보냄
- 곱셈 노드의 역전파는 미분이 대응하는 원소

$$\boxed{1}\ \underset{(2,)}{\frac{\partial L}{\partial X}} = \underset{(3,)}{\frac{\partial L}{\partial Y}}\ \underset{(3,\ 2)}{W^T}$$

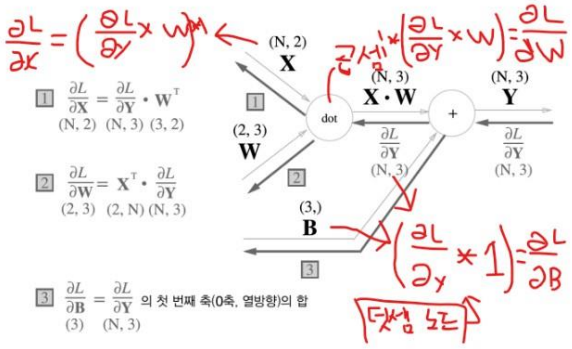
$$\boxed{2}\ \underset{(2,\ 3)}{\frac{\partial L}{\partial W}} = \underset{(2,\ 1)}{X^T}\ \underset{(1,\ 3)}{\frac{\partial L}{\partial Y}}$$




```
[541]: x = np.random.rand(2)
       print("x : ", x)

       w = np.random.rand(2, 3)
       print("w : ", w)

       b = np.random.rand(3)
       print("b : ", b)

       xw = np.dot(x, w)
       y = xw + b
       print("y : ", y)

       print(x.shape, ' x ',w.shape, ' + ',b.shape, ' = ',y.shape)
```

```
x :  [0.32527602 0.14962062]
w :  [[0.29273888 0.40286714 0.57837329]
 [0.12640808 0.02430014 0.39727107]]
b :  [0.78201089 0.84157826 0.93866212]
y :  [0.89614508 0.97625708 1.18623302]
(2,)  x  (2, 3)  +  (3,)  =  (3,)
```

역전파 해보기



[542]:
```
# 가상의 손실 함수값
cross = np.random.rand(3)
cross
```

[542]:
```
array([0.44728219, 0.30406049, 0.54252038])
```

[543]:
```
# dL / dY
back = cross / y
back
```

[543]:
```
array([0.49911806, 0.31145535, 0.45734722])
```

x * w 부분과 편향은 덧셈 노드라 그대로 L / b가 됨

[544]:
```
# 덧셈 노드는 미분이 1이라 그대로 흐름
back_b = back * 1
back_xw = back * 1
back_b
```

[544]:
```
array([0.49911806, 0.31145535, 0.45734722])
```

np.dot은 곱셈 노드로 대응하는 원소의 곱이 된다

곱셈 노드는 행렬의 차원 수를 일치시켜야함

- 아래는 행렬 차원 수를 일치하지 않아 오류가 발생

[545]:
```
print("x에 대한 역전파 " ,back_xw.shape, ' x ', w.shape)
print("w에 대한 역전파 " ,back_xw.shape, ' x ', x.shape)
```

```
x에 대한 역전파  (3,)  x  (2, 3)
w에 대한 역전파  (3,)  x  (2,)
```

[546]:
```
back_xw_x = back_xw.copy()
print("x에 대한 역전파 XW * W" ,back_xw_x.shape, ' x ', w.shape)

back_x = w * back_xw_x
print(back_x)
```

```
x에 대한 역전파 XW * W (3,)  x  (2, 3)
[[0.14611126 0.12547512 0.26451742]
 [0.06309256 0.00756841 0.18169082]]
```

[547]:
```
back_xw_w = back_xw.copy()
back_xw_w = np.reshape(back_xw_w, (3, 1))
x = np.reshape(x, (1, 2))
print("w에 대한 역전파 XW * X" ,back_xw_w.shape, ' x ', x.shape)

back_w = np.dot(back_xw_w, x)
print(back_w)
```

```
w에 대한 역전파 XW * X (3, 1)  x  (1, 2)
[[0.16235113 0.07467835]
 [0.10130896 0.04660014]
 [0.14876408 0.06842858]]
```

[548]:
```
print(back_xw_x.shape, ' x ', x.shape)
print(back_xw_w.shape, ' x ', w.shape)
```

```
(3,)  x  (1, 2)
(3, 1)  x  (2, 3)
```

[549]:
```
back_x = back_xw_x * w *  1
back_w = back_xw_w * x *  1
```

[550]:
```
print("aL / aX")
print(back_x)
```

```
aL / aX
[[0.14611126 0.12547512 0.26451742]
 [0.06309256 0.00756841 0.18169082]]
```

[551]:
```
print("aL / aW")
print(back_w)
```

```
aL / aW
[[0.16235113 0.07467835]
 [0.10130896 0.04660014]
 [0.14876408 0.06842858]]
```

[ ]:

실습 내용

덧셈에 대한 순전파

[552]:
```
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]]
```

위 덧셈에 대한 역전파

덧셈 노드에서 기존 편향 행렬 차원의 수를 맞추기 위해 axis를 이용하여 차원을 조절함

[553]:
```
dY = Y
print(dY, '\n\n')
dB = np.sum(dY, axis = 0)
print(dB)
```

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


[12 14 16]
```

역전파의 구조를 만들어두면 출력값만 넣었을 때 바로 입력에 대해서 역전파를 구할 수 있음

- 즉 순전파의 구조에 맞게 역전파 구조를 만들고, 오차역전파법을 이용하여 가중치를 조절한다

[599]:
```
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)

        dout = dout.reshape(-1, dout.shape[0])
        x = self.x.T.reshape(self.x.T.shape[0], -1)
        print("np.dot(x, dout) = ",  x.shape," * ", dout.shape)
        print("np.sum(dout, axis=0) = ", dout.shape)

        self.dW = np.dot(x, dout)
        self.db = np.sum(dout, axis=0)
        return dx
```

- 각 x, W, b로 순전파를 진행

- 각 x, w, b로 순전파를 진행

```
[600]: x = np.random.rand(2)
       w = np.random.rand(2, 3)
       b = np.random.rand(3)

       af = Affine(w, b)
       out = af.forward(x)

       print(af.x)
       print()
       print(af.W)
       print()
       print(af.b)
```

```
[0.99638685 0.05120904]

[[0.76170311 0.97803158 0.85787354]
 [0.51178584 0.15059227 0.90979617]]

[0.420763   0.30779774 0.76664588]
```

- 출력값으로 역전파를 진행

```
[601]: af.backward(out)
```

```
np.dot(x, dout) =  (2, 1)  *  (1, 3)
np.sum(dout, axis=0) =  (1, 3)
[601]: array([3.61116364, 2.32898767])
```

```
[602]: print(af.dW)
       print()
       print(af.db)
```

```
[[1.20156485 1.28534625 1.66198282]
 [0.06175411 0.06606003 0.08541717]]

[1.20592202 1.29000723 1.66800959]
```

```
[ ]:
```

```
[604]: class SoftmaxWithLoss:
           def __init__(self):
               self.y = None # softmax
               self.t = None # 정답
               self.loss = None #손실

           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]
               dx = (self.y - self.t) / batch_size
               return dx
```

```
[ ]:
```

미니배치 학습

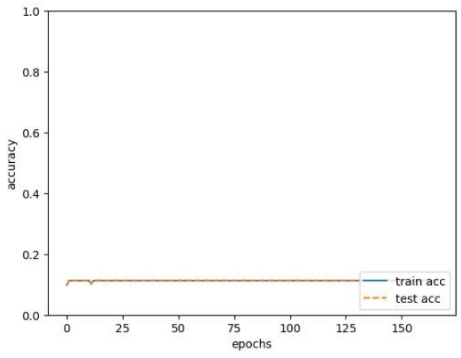
```
*[14]: hidden_size = [15, 15, 15, 15, 15]
       iters_num = 10000
       learning_rate = 0.001

       train_size = x_train.shape[0]
       batch_size = 1000

       # 은닉층 수 = Len(hidden_size)
       network = TwoLayerNet(input_size=784, hidden_size=[15, 15, 15, 15, 15], output_size=10)

       train_loss_list = []
       train_acc_list = []
       test_acc_list = []

       iter_per_epoch = max(train_size // batch_size, 1)
       total = 0

       sgd = SGD()
       momentum = Momentum()
       ada = AdaGrad()
```

```
W1
b1
W2
b2
W3
b3
W4
b4
W5
b5
SGD 생성
Momentum 생성
AdaGrad 생성
```

```
[15]: for i in range(iters_num):
          # 미니배치 획득
          batch_mask = np.random.choice(train_size, batch_size)
          x_batch = x_train[batch_mask]
          t_batch = t_train[batch_mask]

          # 기울기 계산
          grad = network.gradient(x_batch, t_batch)

          # SGD 학습, 모멘텀 학습, AdaGrad 학습
          #sgd.update(network.params, grad)
          momentum.update(network.params, grad)
          #ada.update(network.params, grad)


          # 학습 경과 기록
          loss = network.loss(x_batch, t_batch)
          train_loss_list.append(loss)

          # 1에폭당 정확도 계산
          if i % iter_per_epoch == 0:
              total += 1
              train_acc = network.accuracy(x_train, t_train)
              test_acc = network.accuracy(x_test, t_test)
              train_acc_list.append(train_acc)
              test_acc_list.append(test_acc)
              print(str(total) + "번" + " ---- " + "train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

```
150번 ---- train acc, test acc | 0.11236666666666667, 0.1135
151번 ---- train acc, test acc | 0.11236666666666667, 0.1135
152번 ---- train acc, test acc | 0.11236666666666667, 0.1135
153번 ---- train acc, test acc | 0.11236666666666667, 0.1135
154번 ---- train acc, test acc | 0.11236666666666667, 0.1135
155번 ---- train acc, test acc | 0.11236666666666667, 0.1135
156번 ---- train acc, test acc | 0.11236666666666667, 0.1135
157번 ---- train acc, test acc | 0.11236666666666667, 0.1135
158번 ---- train acc, test acc | 0.11236666666666667, 0.1135
159번 ---- train acc, test acc | 0.11236666666666667, 0.1135
160번 ---- train acc, test acc | 0.11236666666666667, 0.1135
161번 ---- train acc, test acc | 0.11236666666666667, 0.1135
162번 ---- train acc, test acc | 0.11236666666666667, 0.1135
163번 ---- train acc, test acc | 0.11236666666666667, 0.1135
164번 ---- train acc, test acc | 0.11236666666666667, 0.1135
165번 ---- train acc, test acc | 0.11236666666666667, 0.1135
166번 ---- train acc, test acc | 0.11236666666666667, 0.1135
167번 ---- train acc, test acc | 0.11236666666666667, 0.1135
```

```
[16]: markers = {'train': 'o', 'test': 's'}
      x = np.arange(len(train_acc_list))
      plt.plot(x, train_acc_list, label='train acc')
      plt.plot(x, test_acc_list, label='test acc', linestyle='--')
      plt.xlabel("epochs")
      plt.ylabel("accuracy")
      plt.ylim(0, 1.0)
      plt.legend(loc='lower right')
      plt.show()
```




```
[ ]:
```

기울기 소실이 발생하는 것으로 추측

혹은 기울기 소실이 아닌 가중치 초기값 설정 문제

-> 이러한 문제를 해결하기 위해 Xavier He초기값, 배치 정규화가 필요하다