인공지능 4차 과제

2015871005

컴퓨터과학부

김도현

예제코드 세부 사항.

1. 예제 코드 실행 결과와 실행 세부 사항.

예제코드의 세부 사항은 다음과 같습니다.

1) AdamOptimizer를 사용합니다.
2) loss 함수의 엔트로피가 최소가 되게 학습을 진행합니다.
3) 리턴되는 결과는 소프트 맥스 분류의 최댓값을 따릅니다..
4) 필터에 대한 학습을 진행합니다.
5) 해상도를 ½ 로 낮추는 max 폴링 기법을 사용합니다.
6) 첫번째 convolutional layer에서 32 개의 필터, 두번째convolutional layer에서 64개의 필터를 사용합니다.
7) 마지막 레이어는 입력 1024개의 노드, 출력 10 개의 노드로 이루어진 Dense layer 입니다.
8) 7)의 dense layer 에 서 50%의 드롭 아웃기법으로 오버피팅을 방지합니다.

예제의 코드는 아래와 같은 네트워크 구조입니다.

- 예제코드의 네트워크 구조

종류	필터사이즈	스트라이드	필터 개수
convolutional layer	3 * 3	[1,1,1,1]	32
pooling layer	2 * 2	[1,2,2,1]	
convolutional layer	3 * 3	[1,1,1,1]	64
pooling layer	2 * 2	[1,2,2,1]	

- Dense layer 구조

종류	노드 수	activation function	drop out 비율
Dense layer	3136	relu	0.5
Dense layer	10	softmax	0

아래는 실행결과입니다.

```
Step: 11500, Loss: 286.440033, Accuracy: 0.989600
Step: 12000, Loss: 354.505219, Accuracy: 0.987600
Step: 12500, Loss: 325.713165, Accuracy: 0.989700
Step: 13000, Loss: 328.364166, Accuracy: 0.989800
Step: 13500, Loss: 317.211823, Accuracy: 0.989200
Step: 14000, Loss: 304.178162, Accuracy: 0.988900
Step: 14500, Loss: 325.763489, Accuracy: 0.989300
Step: 15000, Loss: 308.710266, Accuracy: 0.989900
Step: 15500, Loss: 279.710968, Accuracy: 0.990300
Learning complete! : above 99 accuracy in 15500 epoch!)
```

<15500번 학습, 99.03%의 accuracy>

위의 조건만 이용해도, 과제에서 요구하는 1프로 이하의 accuracy를 확인 할 수 있었습니다.
위의 코드를 20 번 반복한 결과, 모든 경우에 대해서 학습이 성공하였고, 소요된 시간과 학습 성공에 필요한 epoch 수는 다음과 같습니다. 위의 코드를 변형하여 20000번 실행 하였을 시 평균 accuracy와 최대 accuracy 를 구하면,

20000번 수행 시 평균 accuracy : 0.9906
20000번 수행 시 최대 accuracy : 0.9914

평균 0.99정도의 accuracy가 나옵니다.

2.층의 개수를 늘렸을 때와 성능 비교.

위의 실험 결과로 convolutional layer 를 두번 거친 mnist 이미지 분류는 99% 의 정확도를 갖는다고 볼 수 있습니다. 하지만 99%의 정확도는 상업적으로 상용화하기 어려운 수치라 생각됩니다. 일반적으로, 모델이 복잡해질수록, 더 자세하게 문제를 풀 수 있습니다.
따라서, 층의 깊이를 깊게 해서, 더 복잡한 모델을 만들어 수행해 보겠습니다.

1) convolutional layer, pooling layer가 하나씩 추가된 네트워크

종류	필터사이즈	스트라이드	필터 개수
convolutional layer	3 * 3	[1,1,1,1]	32
pooling layer	2 * 2	[1,2,2,1]	
convolutional layer	3 * 3	[1,1,1,1]	64
pooling layer	2 * 2	[1,2,2,1]	
convolutional layer	3 * 3	[1,1,1,1]	32
pooling layer	2 * 2	[1,2,2,1]	

- Dense layer 구조

종류	노드 수	activation function	drop out 비율
Dense layer	1024	relu	0.5
Dense layer	10	softmax	0

위의 조건에 각 convolutional layer에서 padding = “SAME”으로 패딩하였을때, 수행결과는

평균 accuracy : 0.9913
최대 accuracy : 0.9938

으로 예상했던 대로, 조금 더 향상된 accuracy를 얻을 수 있었습니다..
앞으로 몇번의 실험에서, 이 모델을 재사용하는 경우가 많으니, 설명을 원활히 하기 위해 이 네트워크 모델을 A모델이라 하겠습니다.

2) 층을 2개 늘린 네트워크 구조

종류	필터사이즈	스트라이드	필터 개수
convolutional layer	3 * 3	[1,1,1,1]	32
pooling layer	2 * 2	[1,2,2,1]	
convolutional layer	3 * 3	[1,1,1,1]	64
pooling layer	2 * 2	[1,2,2,1]	
convolutional layer	3 * 3	[1,1,1,1]	64
pooling layer	2 * 2	[1,2,2,1]	
convolutional layer	2 * 2	[1,1,1,1]	64
pooling layer	2 * 2	[1,2,2,1]	

- Dense layer 구조

종류	노드 수	activation function	drop out 비율
Dense layer	256	relu	0.5
Dense layer	10	softmax	0

각 convolutional layer에서 padding = “SAME”으로 패딩하였을때, 수행결과는 아래와 같습니다. 마지막 convolutional filter 는 pooling 계층에서 받는 이미지의 크기를 고려하여 2*2로 수정하였습니다.

평균 accuracy : 0.9022
최대 accuracy : 0.9140

분석

예제코드에서 한개의 층을 늘린 조건보다도, 정확도가 낮게 나왔습니다. 모델이 복잡해졌는데, 정확도가 내려간 이유는 오버피팅이 있기 때문이라고 추측했습니다. 오버 피팅을 줄이기위한 방법은 1) 모델을 간단하게 바꾸기, 2) drop- out을 풀링레이어에 적용하기 등의 방법을 실험해 보기로 결론 내렸습니다.

3. 오버피팅 방지

위의 예제코드를 보며, convolutional filter 가 각각 32개, 64개 인 것은 파라미터가 해결해야 하는 문제에 비해 너무 많은 것은 아닌가 라고 생각했습니다. 슬라이드의 예제코드를 보면, 두개의 가로선, 세로선 필터만으로, 95% 이상의 정확도를 나타내기 때문에 이 문제를 해결하는데 도합 96개의 필터를 적용하는 것은 문제에 비해 너무 어려운 모델을 사용하여, 오버피팅을 야기하는 상황이라 추측하고, 필터의 갯수와 dense layer 의 노드 개수를 줄여 실험해 보았습니다.

실험은 위에서 최고의 accuracy 를 보여준 A모델에서, convolutional layer, pooling layer가 하나씩 추가된 모델에서 적용하였습니다.

▶️convolutional filter의 개수를 줄인 네트워크
위의 A모델에서 convolutional filter의 개수를 낮추어 실험해 보았습니다.

1) 각 convolutional layer 의 필터 개수를 16개로 한 경우.
20회 수행 평균: 0.9885

20회 수행 결과, 오히려 예전보다 못한 성능이 나왔습니다. 너무 노드의 수를 많이 줄인 것 같아, 24개의 필터를 이용해 실험해보았습니다.

2) 각 convolutional layer 의 필터 갯수를 24개로 한 경우.
20회 수행 평균 : 0.9884

실험 결과, 파라미터의 수가 많은 것이 문제는 아니라는 결과가 나왔습니다.

▶️drop out 을 추가로 적용한 네트워크

여태까지 실험 중 최고의 성능을 보인 A 모델과 A모델에 convolutional layer를 하나 더 추가한 모델에 대해, 각 풀링 레이어에 50%의 dropout 을 적용해 보았습니다.

2-2) 각 pooling layer 에 50%의 drop out을 주었을 때,

20회 수행 평균 accuracy : 0.9920
20회 수행 중 최대의 accuracy : 0.9934

drop out을 추가로 주어본 결과, 최대의 accuracy 가 나왔다는 것을 확인 할 수 있었습니다. 아래는 위의 시행이 진행되는 과정을 print 함수를 이용해, accuracy, loss, iteration 을 각각 출력한 화면의 일부입니다.

```
acc: 0.992700, loss : 237.481995 , iter : 36500
acc: 0.992400, loss : 234.254639 , iter : 37000
acc: 0.992300, loss : 237.855347 , iter : 37500
acc: 0.992800, loss : 221.774368 , iter : 38000
acc: 0.992200, loss : 233.363251 , iter : 38500
acc: 0.992100, loss : 235.407501 , iter : 39000
acc: 0.993000, loss : 226.088593 , iter : 39500
acc: 0.993400, loss : 218.972336 , iter : 40000
```

<50%의 drop out 을 각 풀링레이어에 준 모델>

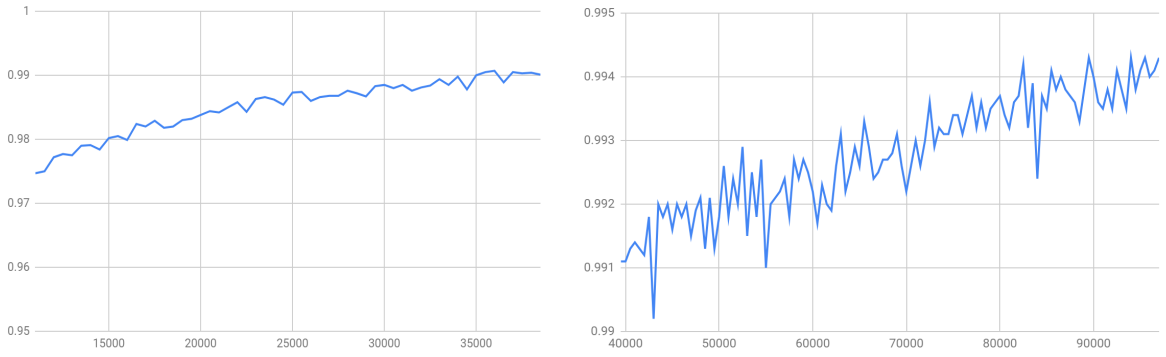
학습이 되는 중간 과정을 살펴 보아도, 평균적으로 높은 accuracy 가 있다는 것을 확인 할 수 있었습니다.

3. 고찰점

실험 결과를 확인하던 중, 의아한점을 발견하였습니다. 예제코드의 모델은 20000번 학습이
후에는 accuracy의 변화가 거의 없던 반면에, 위의 스크린 샷을 보면 A모델에 50%의 drop out 을 적용한 모델에서는 30000번이 넘어도 iteration 이 36000번에서 40000으로 증가하는 중에서도 계속 accuracy가 증가하는 경향을 확인 할 수 있었습니다.

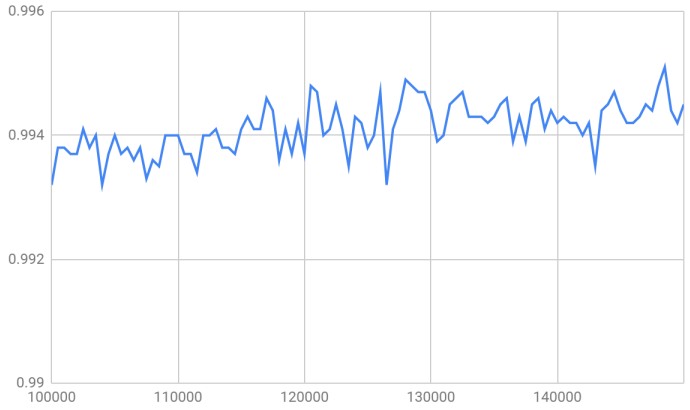
3. 분석

A모델에 convolutional layer를 추가하고 50%의 drop out을 추가한 모델이 학습이 진행되는 과정을, 그래프로 나타낸 것입니다.



<convolutional layer 4개, drop out 이 적용된 모델>

지금 까지의 실험은 40000번 학습을 하면 학습을 끝내는 방식으로 측정한 결과 였습니다. 이유는, 처음 코드를 실행한 convolutional layer가 2개 있는 모델은 20000번 학습 이후에는 학습을 진행해도 accuracy가 증가하는 경향을 보이지 않았기 때문입니다. 하지만 더 복잡해진 모델로 실험을 해본 결과, 40000번의 학습으로는 accuracy가 더이상 높아지지 않는 임계점에 도달하지 않는 다는 것을 알았습니다. 위의 오른쪽 그래프는 같은 모델을 10만번까지 학습 시켜 본 결과입니다 여전히 증가 중이라는 것을 알 수 있었습니다. 아래는 accuracy 가 더이상 증가 하지 않을 때까지 학습 횟수를 올려본 결과 입니다.



<위의 모델을 150000번 학습 시킨 모델>

10번 실험 해본 결과, nan이 나오는 결과를 제외하면 모두 위와 비슷한 결과가 나왔습니다. 120000번 학습시키면 더이상 accuracy가 증가하지 않는 다는 것을 확인 할 수 있었습니다.

평균 accuracy : 0.9942
최대 accuracy : 0.9951

복잡한 모델에서 학습 횟수를 늘리고, 모델에 대한 최적의 학습 횟수를 구함으로서, 최대의 accuracy를 구할 수 있었습니다.

2. layer에 따른 학습

1) 히든레이어 1개의 CNN 모델.

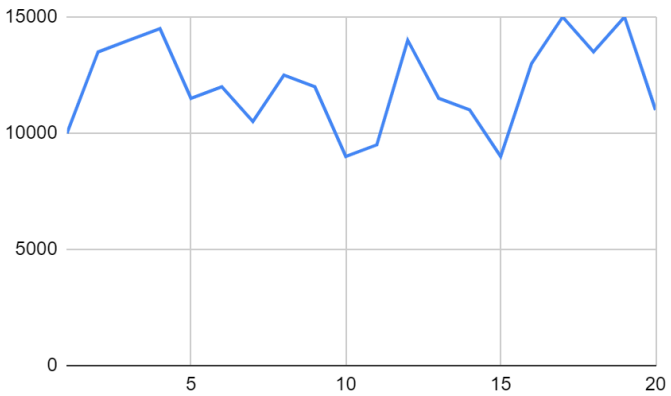
세부사항
- 첫번째 convolutional layer
  필터 크기 : 3*3
  필터의 수 : 32개
- 첫번째 pooling layer
  필터 크기 : 2*2
  stride : [1,2,2,1]

- 두번째 convolutional layer
  필터 크기 : 3*3
  필터의 수 : 64

- 두번째 pooling layer
  필터 크기 : 2*2
  stride : [1,2,2,1]

- 마지막 레이어
  노드의 수 : 1024개
  타입 : fully connected

그래프



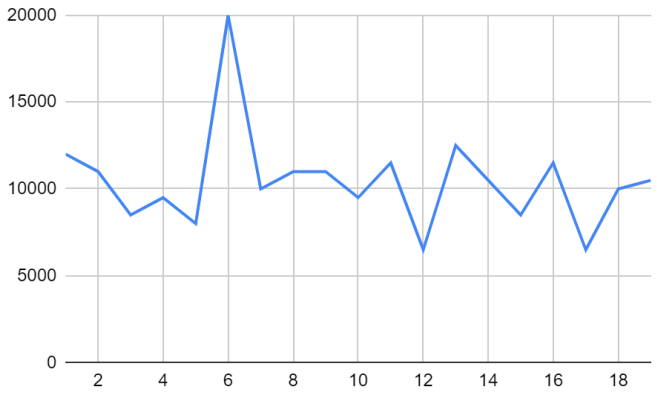
<y축: 학습 성공까지 걸린 학습 횟수>

위의 그래프는 99%에 도달하는 순간을 학습 성공이라 가정했을때, 학습에 공하는데 걸리는 학습 횟수를 그래프로 나타낸 것입니다.
20번 시행을 했을때, 평균 12100 번 학습을 수행해야 학습에 성공하였으며, 20 번 학습을 성공시키는데 소요된 시간은 845초 입니다.

2) 예제코드에서 히든레이어 1층을 더 추가시킨 모델

세부사항:
- 두번째 pooling layer까지는 예제 코드와 동일 합니다.
- 추가된 hidden layer
  convolutional layer
    필터 크기 : 3*3
    필터 갯수 : 64개
  pooling layer
    필터 크기 : 2*2
    stride : [1,1,1,1]

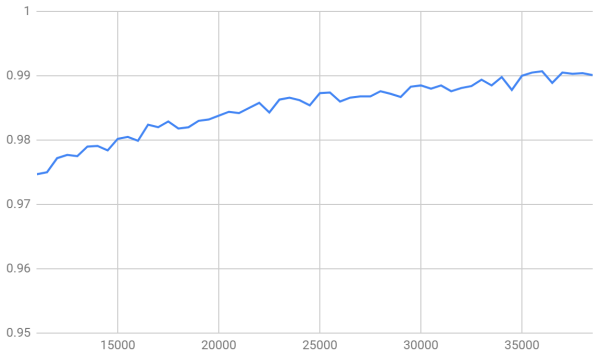
그래프



<y축 : 학습성공까지 걸린 학습 횟수>

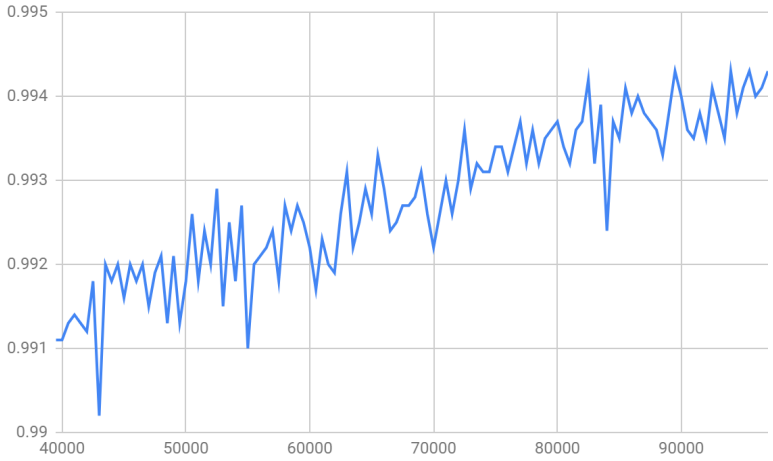
학습 성공에 필요한 학습 횟수 : 평균 12157회
20회 학습성공에 걸린 시간 : 875초

convolutional layer 와 pooling layer 를 하나씩 더 추가한 모델입니다.학습 속도와 학습 횟수에서는 이전 모델과 큰 차이는 보이고 있지 않습니다.



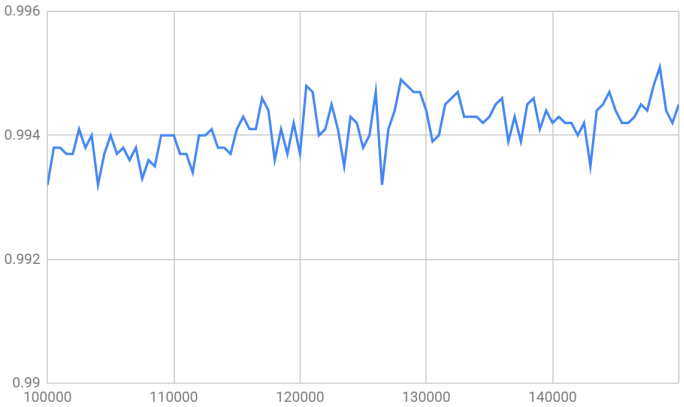
복잡해진 모델에서 수행 하니, 계속 올라감.... 학습 횟수 증가가 필수적임.

다시 실험.



100000번을 학습해도, 여전히 증가하는 추세임. 더해봐야함.

15만번 수행함.



예상대로 멈추긴 함. 이 모델의 포텐셜은 대략 94만번 인 것으로.
5번 실행했는데 3번은 0.994 정도에서 멈췄고, 2번은 중간에 nan이 나오면서 학습에 실패함.

3) 예제코드에서 히든 레이어를 2개 추가시킨 모델

세부사항

- 위의 히든 레이어 1층을 더 추가시킨 모델에서 아래의 층을 더 삽입합니다.
- 추가된 hidden layer
  - convolutional layer
    - 필터 크기 : 3*3
    - 필터 갯수 : 64개

pooling layer

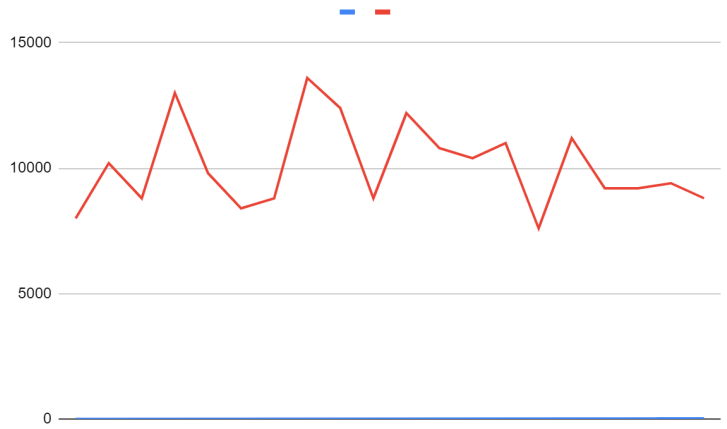
필터 크기 : 1*1

stride : [1,1,1,1]

```
시작시간 :  1576658921.208813
 걸린 시간 :  3652.843896627426
[0.98969996, 0.99, 0.9901, 0.9902, 0.9895, 0.9898, 0.9878, 0.9901,
최대값 : 0.9902
평균:  0.9896199941635132
실패횟수 :  0
```

그래프

```
acc: 0.987900, loss : 353.492493 , iter : 30500
acc: 0.988400, loss : 356.972687 , iter : 31000
acc: 0.987500, loss : 364.726074 , iter : 31500
acc: 0.988100, loss : 364.883118 , iter : 32000
acc: 0.988600, loss : 350.033051 , iter : 32500
acc: 0.988800, loss : 352.437927 , iter : 33000
acc: 0.988200, loss : 366.367981 , iter : 33500
acc: 0.989000, loss : 343.426270 , iter : 34000
acc: 0.988600, loss : 356.157715 , iter : 34500
acc: 0.989900, loss : 329.363831 , iter : 35000
```



<y축 : 학습성공까지 걸린 학습 횟수>

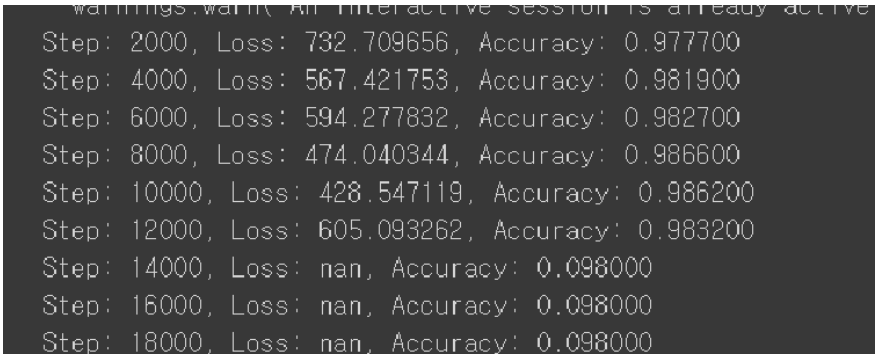
학습 성공에 필요한 학습 횟수 : 평균 10080회

20회 학습성공에 걸린 시간 : 1789초

convolutional layer에서 나온 이미지의 크기와 pooling layer 의 필터 크기가 같기 때문에 pooling layer 의 크기를 1*1 로 조정하였습니다. 즉 마지막 pooling은 하지 않은 것과 같은 상태 입니다.

모델이 필요 이상으로 복잡해지니 평균 학습 횟수는 줄었지만, 학습에 소요된 시간은
2배 이상 증가한 것을 확인 할 수 있었습니다.

4) 예제코드에서 히든 레이어를 2개 추가시킨 모델
위와 같은 방법으로 , 레이어를 하나 더 추가 시킨 히든레이어 4개의 모델은 학습이 불가능했습니다.. 아래 그림을 보시면, loss 함수의 크기가 너무 커져 nan 이라 나오는 것을 확인할 수있습니다. 이유는 gradient exploding 이라 추측됩니다. activation 함수로, sigmoid 함수, tanh 함수를 쓰는 것으로 이 상황을 방지 할 수 있을 것이라 생각됩니다.



<히든 레이어 4층의 모델, 14000회 부터 nan>

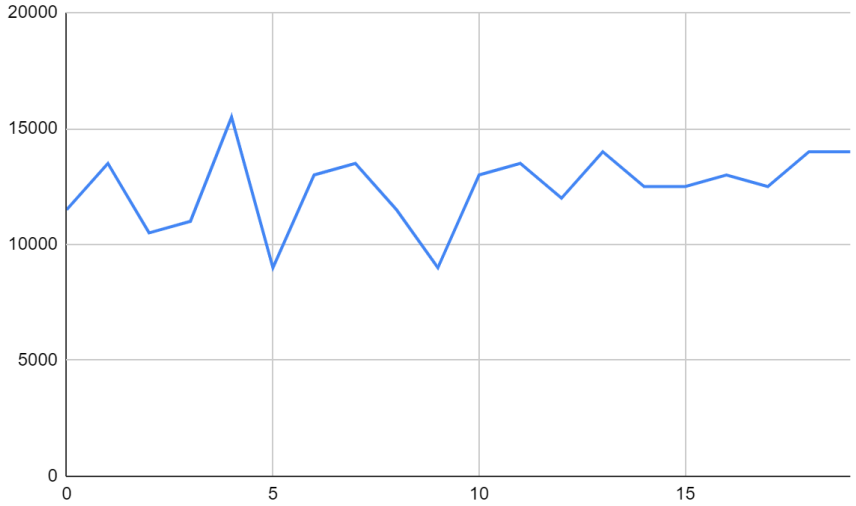
5) 예제 코드보다 간략한 모델.
세부사항
- 첫번째 convolutional layer
  필터 크기 : 5*5
  필터의 수 : 32개
- 첫번째 pooling layer
  필터 크기 : 2*2
  stride : [1,2,2,1]

- 두번째 convolutional layer
  필터 크기 : 5*5
  필터의 수 : 32
- 두번째 pooling layer
  필터 크기 : 2*2
  stride : [1,2,2,1]

- 마지막 레이어
  노드의 수 : 1024개

타입 : fully connected

위의 결과는 책에서 제시된 예제 코드에서 더욱 복잡해 질수록, 학습하는데 걸리는 시간은 늘어남을 확인 할 수 있었습니다. 예제 모델이 애초에 풀어야 할 문제보다 복잡하게 설계되었을 가능 성을 고려하여 히든 레이어의 convolutional filter 의 개수를 절반으로 줄여 학습해 보았습니다.



<두번째 convolutional filter의 개수가 32개인 예제 코드 20회 실행 결과>

20회 실행에 걸린 시간 :1796초
학습 성공에 걸린 학습 횟수 : 12450회

예제코드보다 간결한 구조이고, 같은 런타임 옵션(GPU)를 사용하였음에도 불구하고, 수행 시간이 늘었다는 것을 확인할 수 있었습니다.

3. 결론

과제에서 소개된 문제를 CNN을 이용하여 어떻게하면 문제를 빨리, 학습을 적게해서 학습에 성공할 수 있을지에 대해 중점적으로 생각하면서 실험해 보았습니다.
위의 실험 결과를 토대로 얻은 결론은 다음과 같습니다.

1) mnist를 99%이상의 정확도를 맞추는데 convolutional layer를 이용하고, 각 레이어의 필터의 수 를 64개라고 한다면, 히든 레이어의 수는 1개 or 2개가 적당하다.

2) 모델을 필요 이상으로 복잡하게 설계할 경우 vanishing gradient, exploding gradient 의 문제를 야기할 가능성이 높다.

3) 모델을 간략하게 바꾼다 해서, 학습시간이 그에 비례하여 적어지는 것은 아니다.

4) 학습성공에 필요한 반복횟수가 적다고 해서, 학습에 소요된 시간이 적어지는 것은 아니다. 즉, 필요한 것은 학습 시간을 측정하는 일이다.