**데이터 마이닝 HW#2**

**20141575 임형석**

**Exercise 1**

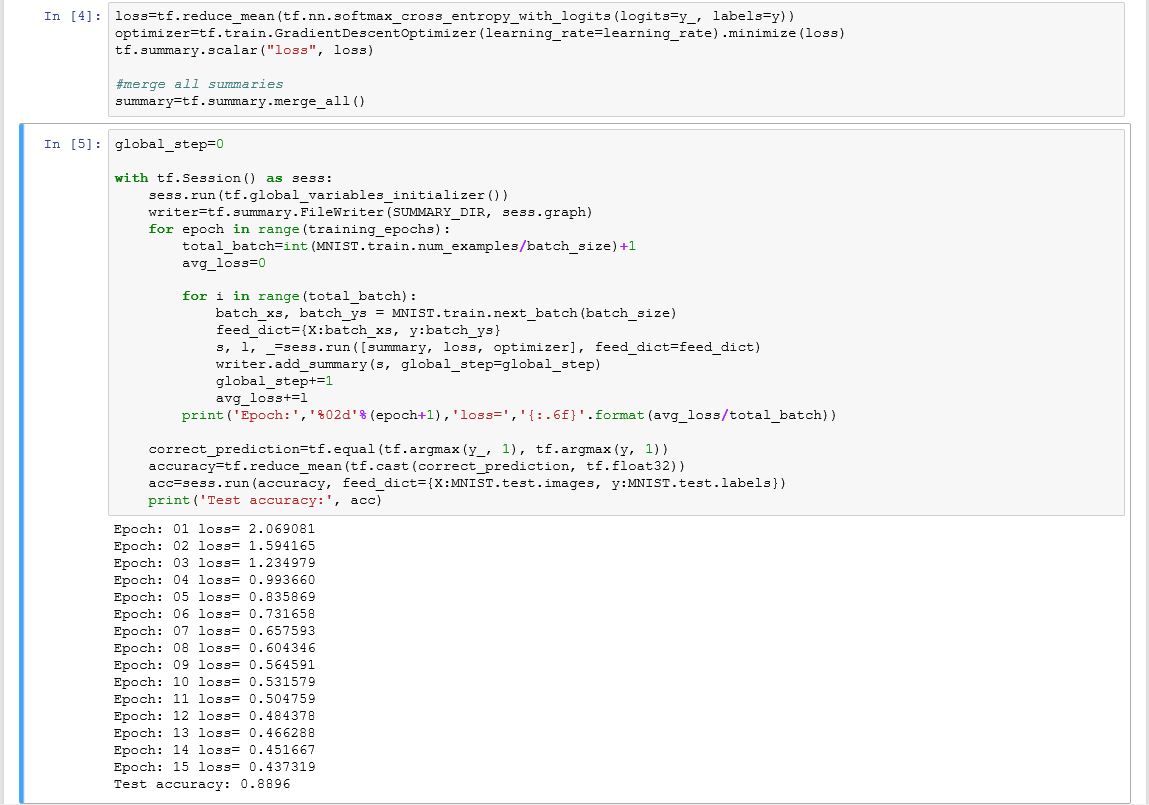
1 . Example의 NN는 한 개의 층으로 구성되어 있다. 이를 수정하여 double layered fully connected neural network with softmax classifier를 구현하고 mnist 데이터로 테스트해 본다.

(parameter, layer 내의 node의 수 자유롭게 수정 가능, 두 개 이상의 층 구축 가능)

기존에 주어진 코드에서 layer를 하나 더 추가해주고 node의 개수를 1024개로 해준 것의 코드와 결과는 다음과 같다.

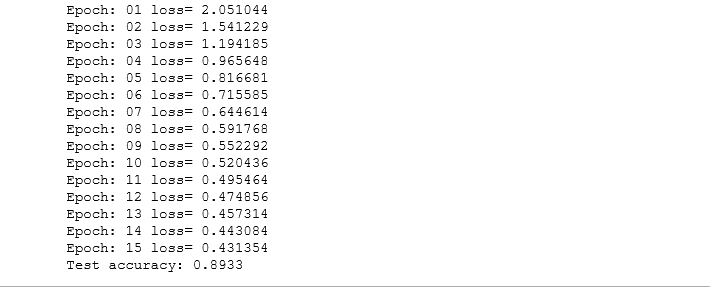
( input layer = [784,512], hidden layer = [512,1024], output layer = [1024,10] )





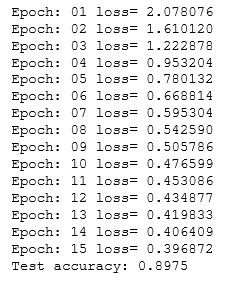
그리고 위와 같이 layer을 하나 더 추가 해 주었지만 node의 개수를 1024개에서 256개로 줄인 결과는 다음 과 같다.

( input layer = [784,512], hidden layer = [512,256], output layer = [256,10] )



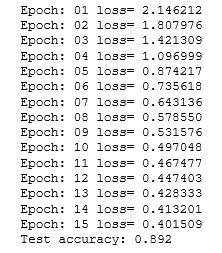
위의 코드에서 층을 하나 더 더해 layer가 총 4개의 일 때의 결과는 다음과 같다

( input layer = [784,512], hidden layer = [512,1024] hidden layer = [1024,256], output layer = [256,10] )

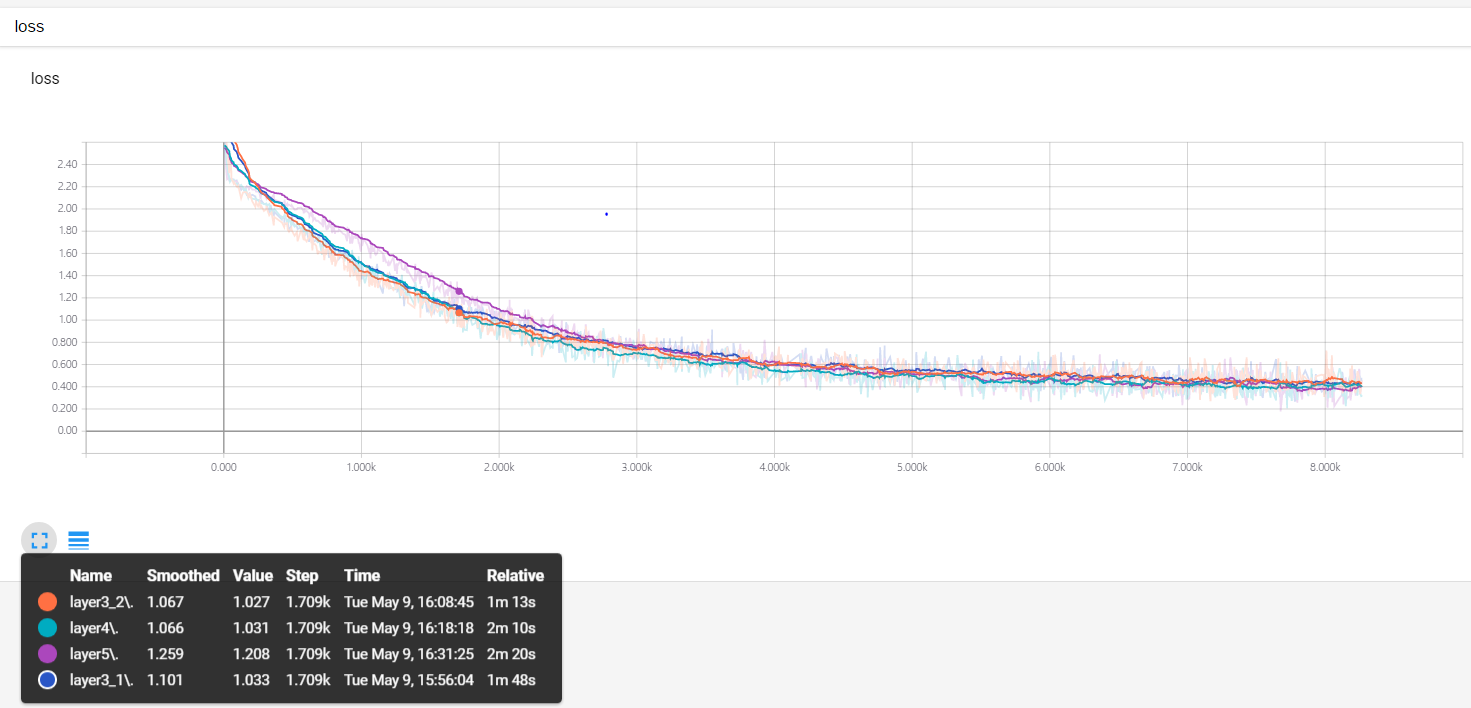


마지막으로 layer가 총 5개의 일 때의 결과는 다음과 같다.

( input layer = [784,512], hidden layer = [512,1024] hidden layer = [1024,256] hidden layer = [256,128], output layer = [128,10] )

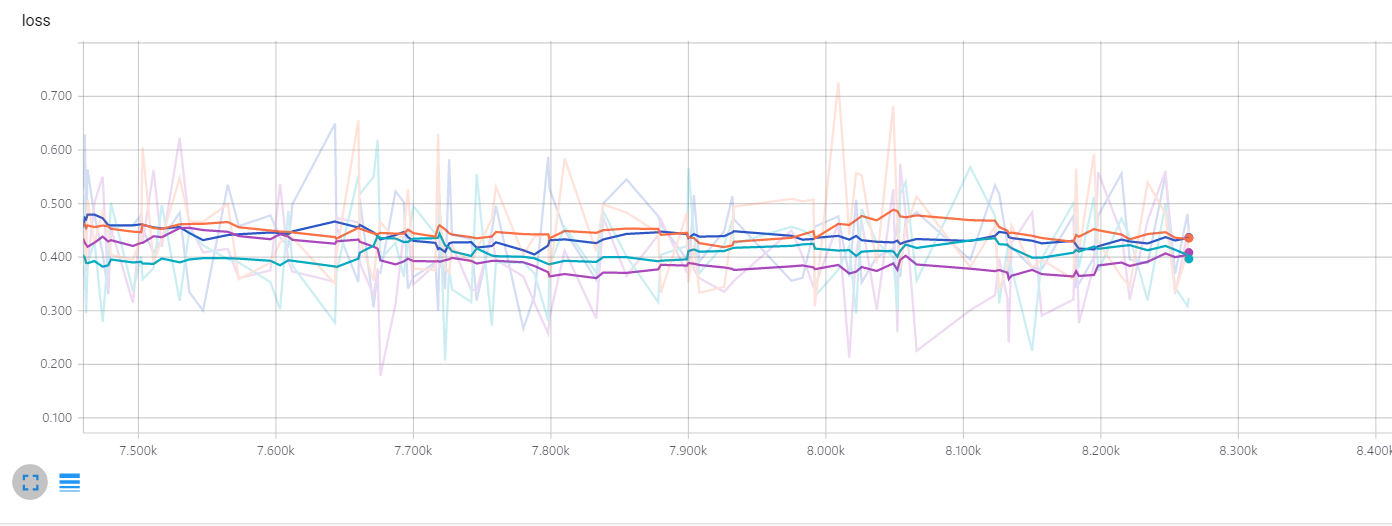


2.Exercise 1-1에서 수행한 실험들을 loss를 텐서보드를 이용해 하나의 plot에 나타낸다.



layer3\_1과 layer3\_2는 layer 층이 3개이면서 각각 노드의 개수를 다르게 해서 나타내었고 layer4와 layer5는 각각 layer층이 4,5개인 그래프를 tensorboard로 나타내었다.

그래프의 마지막 부분을 확대하여 보면 다음과 같은 사진이 나오는데



보라색 선(layer5)가 loss가 제일 작고 그다음 하늘색 선(layer4)이 그 다음으로 loss가 낮고 파란색과 주황색(layer3\_1, layer3\_2)는 거의 비슷한 것을 알 수 있다. 따라서 layer가 많으면 많을 수록 loss가 줄어드는 것을 알 수 있다.

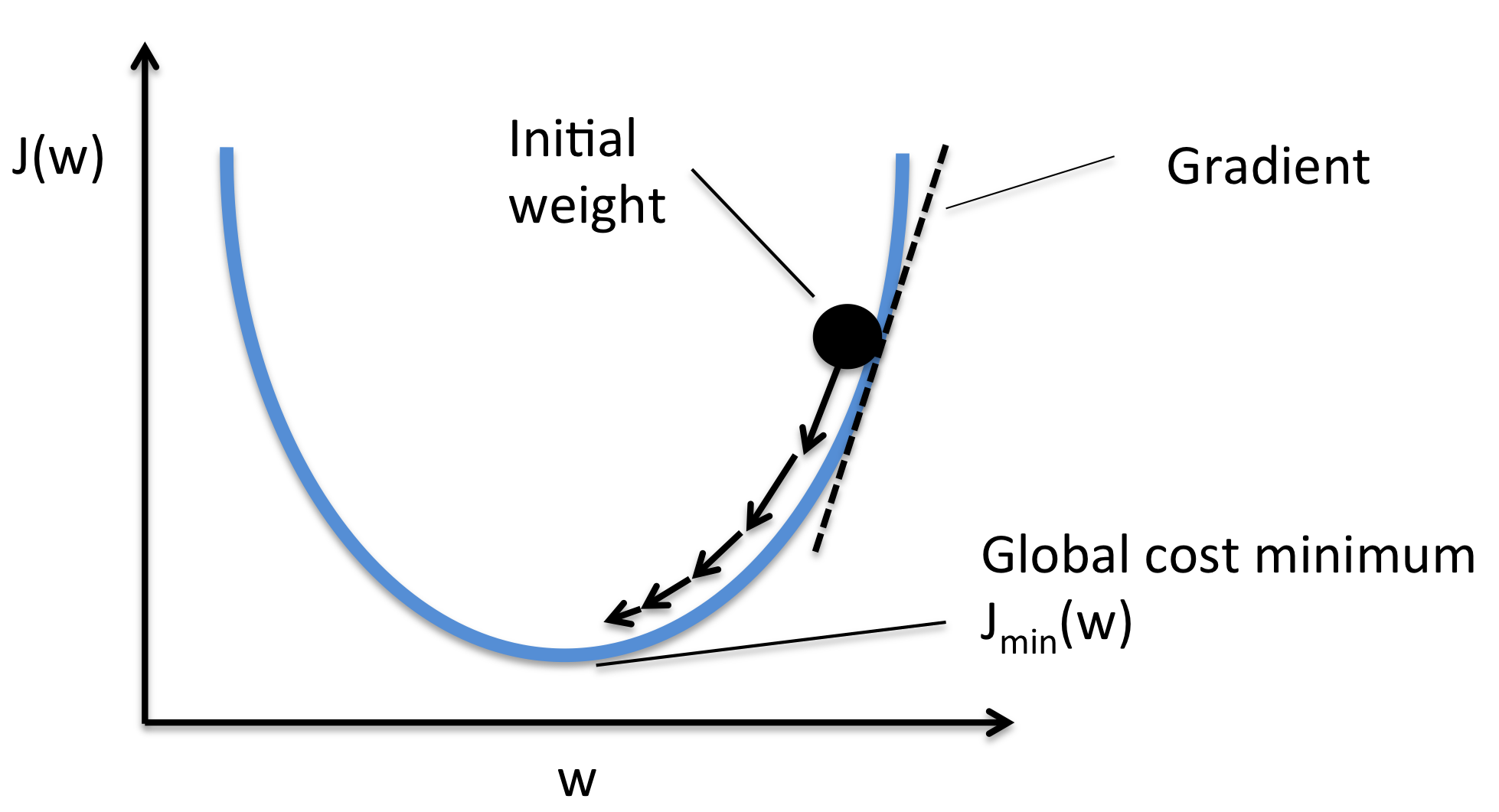
**Exercise 2**

1. Example의 Optimizer는 loss를 minimize하는 방법으로 stochastic gradient descent(SGD)을 사용한다. Exercise 1-2의 예시 그림에서는 다른 방법으로 AdamOptimizer를 사용했는데 SGD와 AdamOptimizer에 대해 간략히 설명한다.

: stochastic gradient descent algorithm을 설명하기에 앞서 gradient descent algorithm을 먼저 설명하자면 cost function을 최소화하여 linear regression 학습을 최종적으로 완성하고 이때 cost function이 최소화 되는 W와 b를 구하는 것이다.( 가설 H(x) = Wx + b )

cost가 가장 작은 점을 찾는 방법은 아무 점에서 시작하여 경사가 있는 곳으로 계속 내려가여 바닥에 도착할 때 까지 경사를 따라 한 번씩 움직이는 것이다.

즉 아이디어는 함수의 기울기를 구하여 계속 해서 기울기가 낮은 쪽으로 이동시켜 기울기가 0인 즉 극값에 이를 때 까지 반복시키는 것이 Gradient descent algorithm이다. 또한 learning rate를 어떻게 정해주느냐에 따라 학습의 정도 즉 움직이는 정도가 달라지게 된다. 적당한 learning rate를 정해주지 않으면 변동이 너무 크거나, 내려가는 속도가 너무 느려 제대로 극값에 도달되지 않는 경우도 있다. 또한 아래와 같은 함수의 경우 대부분 같은 결과가 나오 겠지만 주어진 함수에서 곡률에 따라서 거의 같은 위치에서 시작했음에도 불구하고 완전히 다른 결과로 이어질 수도 있다.



Gradient descent 방식에서의 문제점은 machine learning과 같이 데이터의 규모가 큰 case에서 Cost function의 계산이 너무 오래 걸린다는 것이다. 즉 Cost=1 /n \* ∑Cost(x)에서 ∇C 를 구하려면 모든 학습 데이터 x 각각에 대해 ∇Cost(x) 를 계산해야 하고,

평균 ∇Cost=1 / n \* ∑ ∇Cost(x) 을 구해야 하므로 학습 데이터가 많은 경우 계산이 오래 걸리게 된다. 이를 해결하기 위해 등장한 방법이 stochastic gradient descent이다.

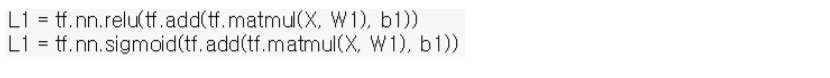
Stochastic gradient descent(SGD)와 gradient descent는 모두 cost function이 작아지도록 수정하는 것인데, 큰 차이점으로 Gradient descent는 전체 샘플들을 반복적으로 업데이트하고, 반면에 SGD는 training set에서 랜덤으로 하나의 sample만을 반복적으로 업데이트한다.

즉 SGD는 모든 ∇Cost(x)로 부터 정확한 ∇Cost를 구해주는 것이 아니라 일부 표본인 ∇Cost(x)로 부터 ∇Cost를 추정해내는 방법이다. 학습 데이터 중 m개 만을 무작위로 선출하였을 때 각각의 데이터를 X1,X2,X3, … Xm이라고 하고 이들을 mini batch라고 부르는데 이 학습 데이터들로 학습을 해나간다. 그런 다음 또 다른 mini batch를 선택하고 다시 그 데이터로 학습을 해가는 식으로 학습을 진행한다.

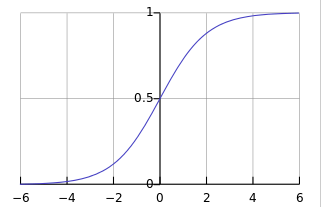
Adamoptimizer(Adaptive Moment Estimation)은 gradient descent optimization algorithm중 하나인데 각각의 parameter에 learning rate를 적용시켜 계산(학습)해주는 방법 중 하나이다. 전의 제곱된 gradients vt의 기하급수적으로 감소하는 평균을 저장하는 것 외에도 Adadelta와 RMSprop와 같이 Adam은 전의 gradients mt을 기하급수적으로 감소시키는 평균을 유지시킨다.

mt = β1mt-1 + ( 1 - β1)gt

vt = β2vt-1 + (1-β2)gt^2

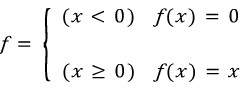
2. 또한 example의 layer1에서 activation function으로 sigmoid를 사용하였는데 Exercise 1-2의 예시 그림에서는 다른 방법으로 ReLU를 사용했다. Sigmoid와 ReLU에 대해 간략하게 설명한다.

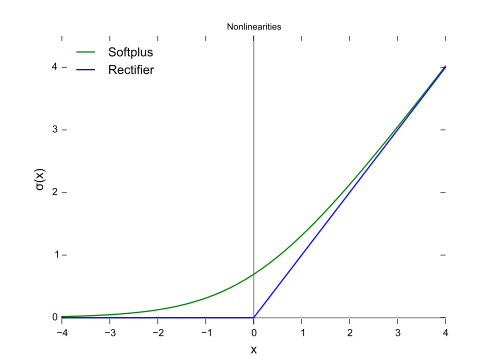
함수가 계단 함수와 같은 경우 연속이 되지 않으므로 미분이 불가능 할때 gradient descent가 사용 될 수 없는데 이럴 때 미분이 되지 않는 지점에서 사용되는 것이 sigmoid 함수이다. 즉 계단 형식의 함수를 미분이 가능하도록 곡선화를 해주는 함수인데 표현과 함수는 다음과 같다.



기울기에 따라 계단 함수와 비슷한 것을 볼 수 있다.

neural network에서 문제점 중 하나인 gradient vanishing은 hidden layer가 1,2개일 때는 발생하지 않지만 네트워크가 깊어지면 깊어 질수록 생긴다. gradient vanishing은 gradient가 0으로 수렴하는 문제이다. sigmoid에서 gradient는 0에서 1사이의 값이 나오게 되는데 backpropagation을 하면서 layer를 거듭 하면 거듭할수록 계속해서 gradient를 곱하게 되는데 0에 가까운 수를 계속 곱하다 보니 값이 점점 작아져 gradient가 0에 수렴하게 되는 것이다. 그래서 gradient vanishing의 문제의 해결책으로 나온 것이 ReLU function인데 표현과 그래프는 다음과 같다.





ReLu가 가지는 이점은 다음과 같은데 첫 째로 0이하의 입력(음수)에 대해 0을 출력함으로서 부분적으로 활성화 시킬 수 있고, 두번째로 gradient vanishing이 없으며 gradient가 exploding되지 않고, 선형함수이므로 미분계산이 매우 간단하다는 점이 있다.

3. SGD를 적용하기 위해 한 training epoch에서 mnist 학습 데이터를 128개씩 N ÷ 128개의 그룹으로 나누어 매 학습 당 128개 (마지막은 N % 128개)의 데이터를 학습시킬 수 있는 get\_train\_batch를 구현한다.



매 학습당 128개의 데이터를 학습시켜야 하므로 batch\_size는 128개로 바꾸어주었다.

get\_train\_batch를 다음과 같이 구현하였는데, 처음에 0부터 총 examples수 만큼의 list를 생성한다. 그리고 시작 index를 정해주고, 첫번 째 mini batch일 경우에는 example list를 shuffle 시켜주어 임의의 데이터를 학습할 수 있게 해준다. batch\_cnt는 total\_batch-1 에 해당하는 값으로 마지막에 남는 mini batch가 아닐 경우 시작 index + batch size만큼을 end index로 해주고 만약 마지막 mini batch일 경우 end index를 128개로 나눈 나머지만큼을 end index로 설정해주었다. 그리고 list에서 start부터 end까지의 리스트내용들을 local\_id에 저장하였다.

위의 SGD를 적용하고 layer 층이 3개인 결과는 다음과 같다.

