1. **Training 과정 코드분석**

매개변수로 받은 배치 입력, 출력데이터를 mini\_batch\_inputs, mini\_batch\_outputs 변수에 저장

**#forward**

* 입력 받은 데이터를 deque모델을 이용해 스택 자료구조형으로 z\_stack 변수에 저장
* 입력 받은 데이터를 activation 초기값으로 지정 (forward와 loss 연산에 사용)
* 반복문을 사용해서 Layers에 저장 된 각각의 레이어에 대해 forward 연산을 한다. 반복문 내 변수 z에는 input값에 대한 레이어의 forward 연산 한 결과를 저장하고, 변수 activation에는 forward연산 결과인 z를 레이어의 activation 연산한 결과를 저장한다. 여기서 activation은 레이어들의 forward연산을 진행하기 위해 사용하고 z는 backward연산을 진행하기 위해 사용된다.

**#calculate loss**

* forward연산 이후 네트워크를 통과한 결과값인 activation과 레이블 outputs와의 loss값을 Network 객체를 선언할 때 지정한 loss연산을 통해 변수 loss\_err에 저장한다.

**#backward pass**

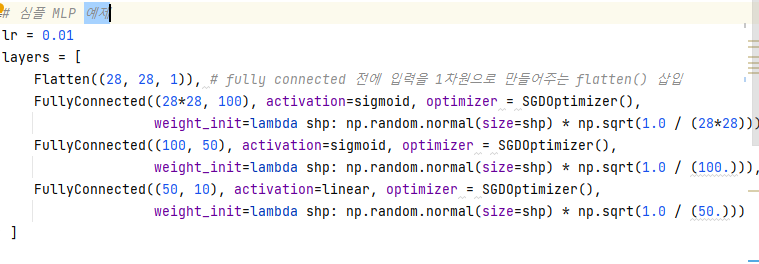
* backward연산을 위해서 z\_stack에 레이어의 역순으로 저장해 놓은 각 레이어의 forward연산 결과들 중 맨 처음 값을 변수 lz의 초기값으로 지정
* 변수 upstream\_gradient에 초기값으로 전체 loss 값을 지정
* grads를 빈 스택으로 지정
* 반복문을 사용해서 Layer의 역순으로 변화율을 구한다. 변수 layer\_err에는 경사를 수정하기 위해 local gradient와 연산할 값(=상위 레이어에서 내려오는 upstream gradient가 미분된 activation을 통과한 값)을 저장한다.
* 해당 레이어의 결과값을 z\_stack에서 바깥쪽부터 하나 추출해서 변수 lz를 최신화하고 layer의 가중치를 수정할 변화율을 구하는 get\_grad연산으로 가중치 경사를 grads에 추가해 저장한다. 그리고 layer의 backward 연산으로 입력방향으로의 Downstrem gradient를 구해 Upstream gradient에 저장하고 하위 레이어로 내려 반복한다.

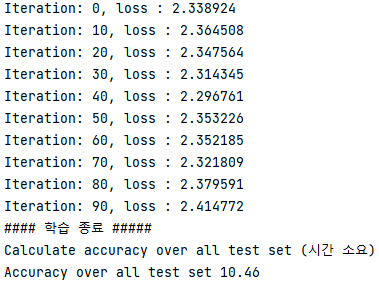
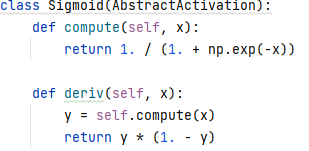
**#update**

* grads에 레이어 역순으로 값(=각 레이어 가중치의 변화율)을 optimizer의 update함수를 이용해 Regularization한 값을 레이어의 기존 w에 반영한다.
* assert 문을 통해 grads에 남아 있는 데이터가 있는지 점검한다. 남아있는 경우, 예외를 발생시킨다.

1. **ReLU 구현**

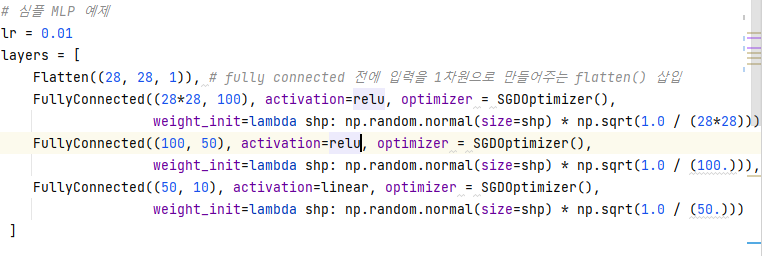
-변경 전

****

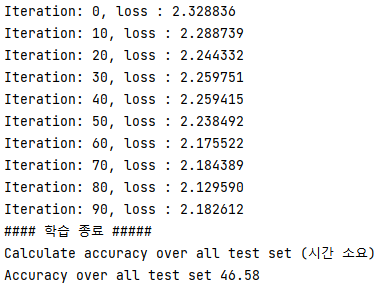


위 세 그림은 기존에 주어진 심플 MLP 예제의 코드 그대로 Sigmoid activation을 이용하여 학습을 진행한 코드와 결과이다. 테스트를 하기 위해 iteration을 100회로 설정하고 진행하였다.

-변경 후

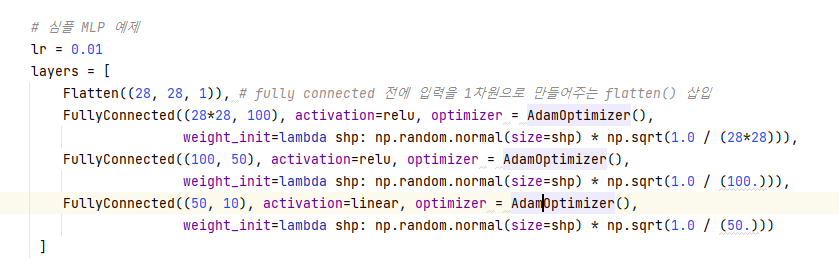


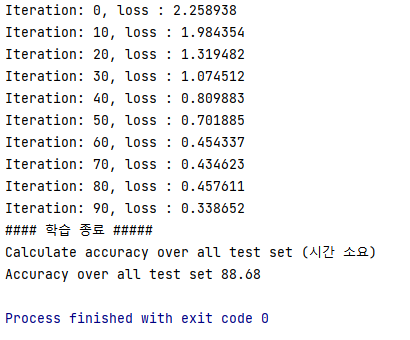
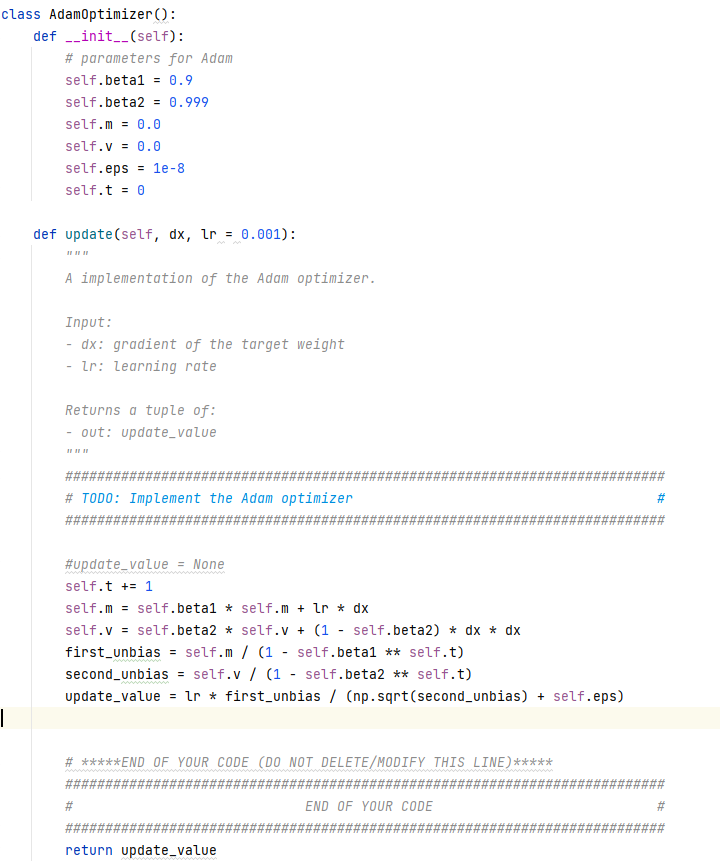


.

기존에 주어진 FC Layer들중 Sigmoid activation을 사용한 Layer들을 ReLU activation으로 교체하여학습을 진행, 다른 변화없이 activation만의 변화로 정확도 20정도 상승한 모습으로 보아 ReLU 구현이 잘 동작하는 것으로 볼 수 있다.

1. **Adam Optimizer 구현**





위 2번(ReLU 구현) 문제에서 ReLU activation을 사용하여 학습한 것과 비교해 보았을 때 optimizer가 SGDOptimizer인 것과 AdamOptimizer인 것의 차이만으로 정확도가 40정도 오른 것을 볼 수 있다. Adam Optimizer 구현이 잘 동작하는 것으로 볼 수 있다.

1. **Weight initialization**