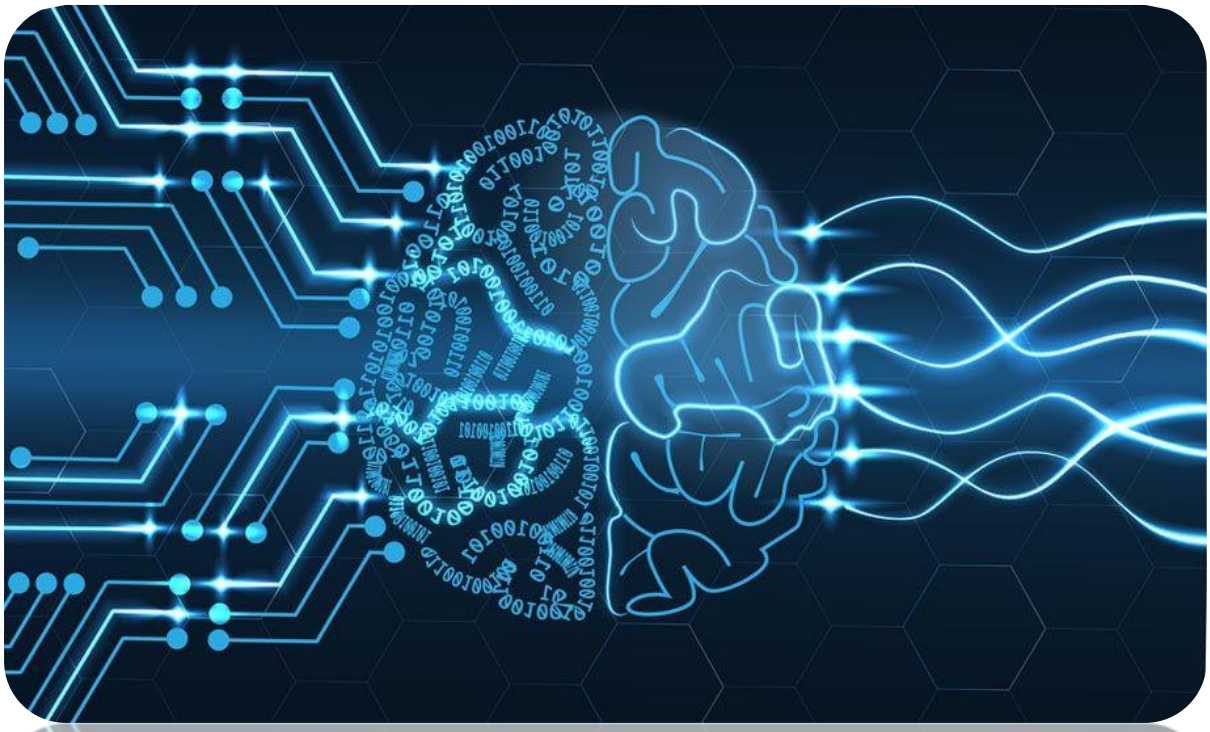


Multi-Layer perceptron

인공지능 과제 #3

컴퓨터과학부 2017920036 양다은

〈 목차 〉

‹ I. 코드구현 ›

1. 과제 설명

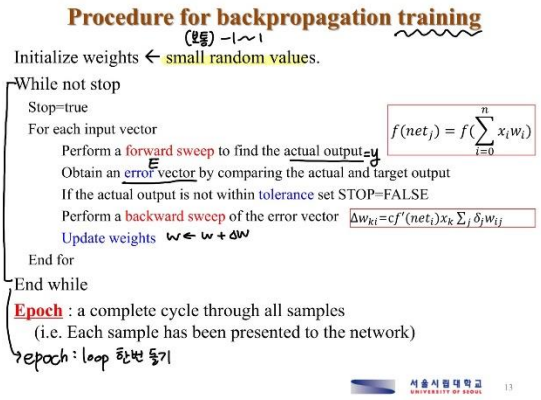
Project #3 Multi-Layer perceptron 구현

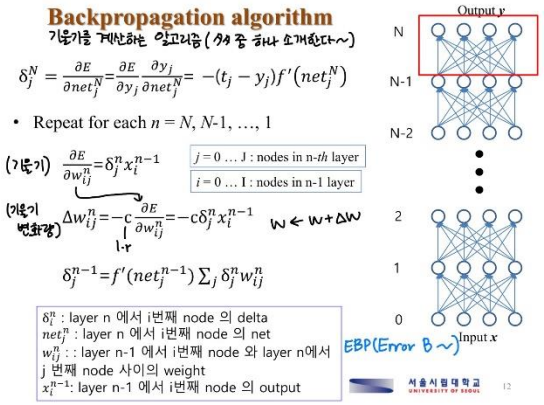
- 실험 과제
  1. AND, OR, XOR 구분 실험
  2. 도우넛 모양 구분 실험 (다음 데이터 이용)
- Layer 수, Layer 당 node 수는 변수로 지정할 것.
- weight는 행렬 형식으로 파일에 저장
- Learning 과정을 그래프로 보여주기 (X1, X2 2차원 직선 그래프).
- 각 노드마다 직선을 그림으로 표시
- 하나의 계산을 하나의 모듈(함수)로 표현
- iteration에 따른 Error 그래프
- 구현언어: C, C++
- 제출물: 프로그램, 결과 보고서
- 실행 10%, 출력 10, 주석 10, 완성도 25, 오류 10, 창의 10 보고서 25%

2. 코드설명

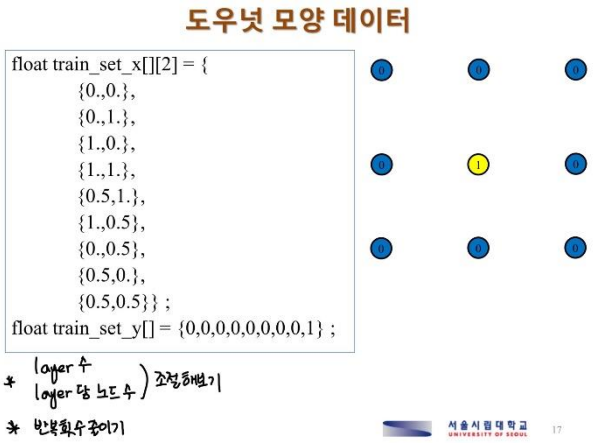
- 첨부파일 [과제#3]2017920036양다은₩소스코드₩ 03.MultiLayerPerceptron 참고한다.
- C++ 언어기반 MLP 신경망 모듈을 구현했다. 클래스 기반 프로그래밍을 위해 구조체와 함수 포인터, 또한 클래스를 활용하여 직접 구현했다. 구체적으로 구조체를 이용해 클래스 변수와 클래스 메서드를 정의하고, 클래스 외부에 생성자 함수(init_)를 생성하여 인스턴스화 했다.
- 두 헤더파일과 클래스를 개발했다. 이전 과제에서 작성한 module.h/ module.cpp 코드는 node와 linear, sigmoid를 적용한 forward, backward 계산을 진행하는 것이다. 이번 과제에서 추가로 작성한 train.h/ train.cpp 코드는 입력받은 layer 수, layer당 node 수, 실행연산을 통해 구체적인 학습을 진행하는 것이다. 마지막으로 main.cpp은 위에서 언급한 입력 값들과 학습환경을 조성할 데이터를 구성하는 메인 코드이다.
- 데이터 보존을 위해서, 파일 입출력 헤더파일인 <fstream>을 헤더에 추가하고, ofstream을 사용했다. 한번의 실행동안 발생한 입력값과 출력값을 각각 구분지어 txt 파일로 '쓰기'를 구현했다. config.txt는 실행 학습환경을 기록하는 점으로 입력받은 layer 수, layer당 node 수, 실행연산을 저장하는 파일이다. loss.txt는 콘솔창과 동일한 데이터를 저장한 것으로 epoch에 따른 loss와 Accuracy를 계산하여 저장한 파일이다. result.txt는 epoch과 input에 따라 계산된 output과 그 값과 target을 비교한 result 값을 기록한 파일이다. 마지막으로 weight.txt는 업데이트되는 weight과 bias 값들을 기록한 파일이다.
- 전체적인 코드의 흐름을 설명하고자 한다. 먼저 main.cpp에서 디버그가 시작한다. 그러면 layer 수와 layer당 node 수를 차례로 입력한다. 내부에서는 입력받은 데이터로 node의 모듈화를 진행한다. module.h와 module.cpp의 인스턴스화로 실행할 신경망 환경이 만들어진다. 그 다음 콘솔창에서 실행할 연산을 선택한다. 이때 제시되지 않은 연산을 선택하면 재선택하도록 하는 에러 차단 코드를 추가적으로 작성했다. 마지막 입력 값인 실행연산을 포함하여 train을 진행한다. 이때, train.h와 train.cpp을 통한 train 클래스에서 학습이 이루어진다. 학습과정은 강의시간에 배운 알고리즘을 구현했다. 정확도(accuracy)가 1.00 이 될 때까지 학습 무한루프에 빠진다. 각 epoch마다 콘솔창으로 loss와 accuracy를 출력하여 진행상황을 파악한다. 학습이 종료하면 main.cpp로 돌아와 총 실행시간을 출력하고, 디버그를 종료한다.
- 구체적인 코드는 자세한 주석을 참고한다. 또한 이전 과제#2에서 언급한 모듈화와 main은 크게 벗

어나지 않는다. 새롭게 개발한 부분은 강의시간에 공부한 알고리즘과 DONUT 데이터 추가, train 클래스 구현이다. train 클래스의 동작과정은 propagation과 backpropagation을 순차적으로 진행한 것을 구현한 코드이다.





- 참고한 이미지는 5주차 강의자료임을 출처한다. 다음과 같은 backpropagation algorithm을 프로그래밍했다. 이때 learning rate(=c)는 0.1 실수로 지정하여 계산을 진행했다.



- 이 또한, 5주차 강의자료임을 출처한다. 다른 gate 연산의 입출력과 다른 donut_input 데이터와 donut_output을 main.cpp에 추가적으로 구현했다.

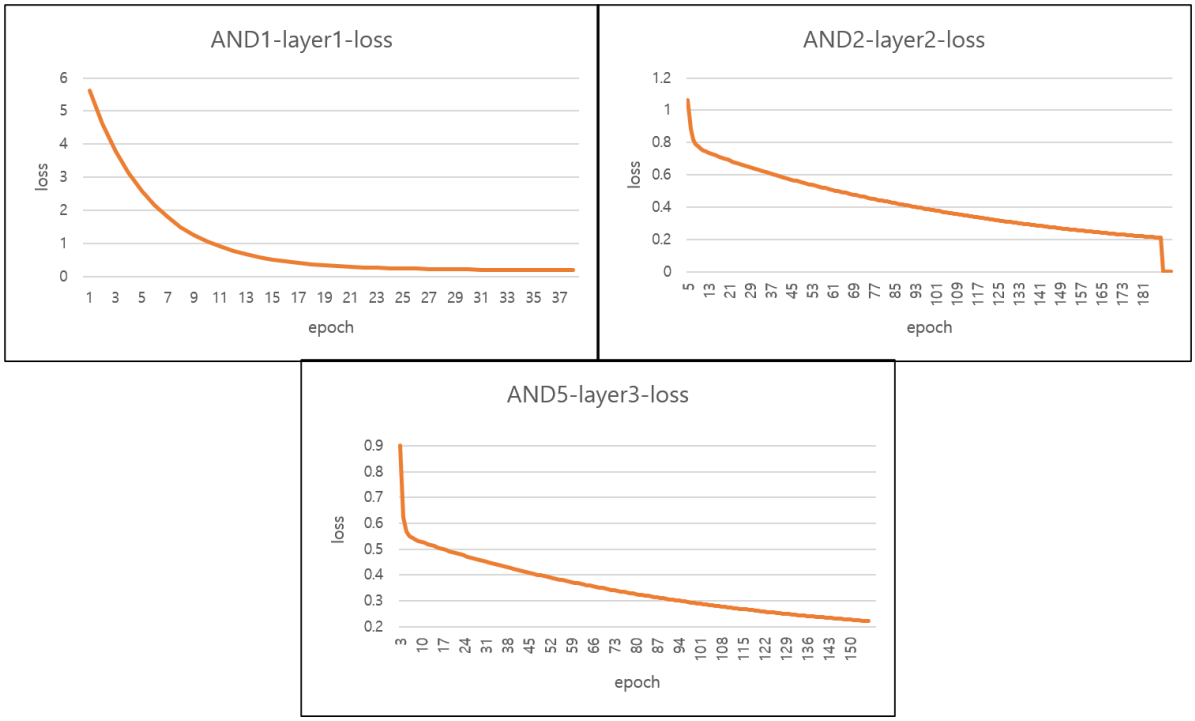
〈 II. 구현결과 〉

지난 과제#2에서는 일일이 프로그래밍 콘솔창을 캡쳐하여 결과를 제시했다. 이번 과제#3에서는 ofstream을 통해 실행 데이터를 txt파일로 저장했고, 프로그래밍 과정이 확정되어 실행 콘솔창을 녹화한 후 저장하는 방식으로 교체했다. 참고자료안에 1.ANDgate 부터 4.DONUT 을 열어보면 구체적인 실행결과를 기록했다. 결과보고서에는 필요한 그래프와 자료만을 활용하여 작성하고자 한다.

layer은 1, 2, 3으로 각각 신경망을 구축하여 학습을 진행했다. 또한 layer당 node 수는 최대 4개까지 입력하여 hidden layer를 생성했다. 이외의 추가적인 신경망 구축을 진행해 보았으나, 학습할 수 없는 환경으로 에러가 발생하거나, 쓰레기 값만 출력하는 문제가 일어났다. 이는 한계점에서 구체적으로 다뤄보고자 한다.

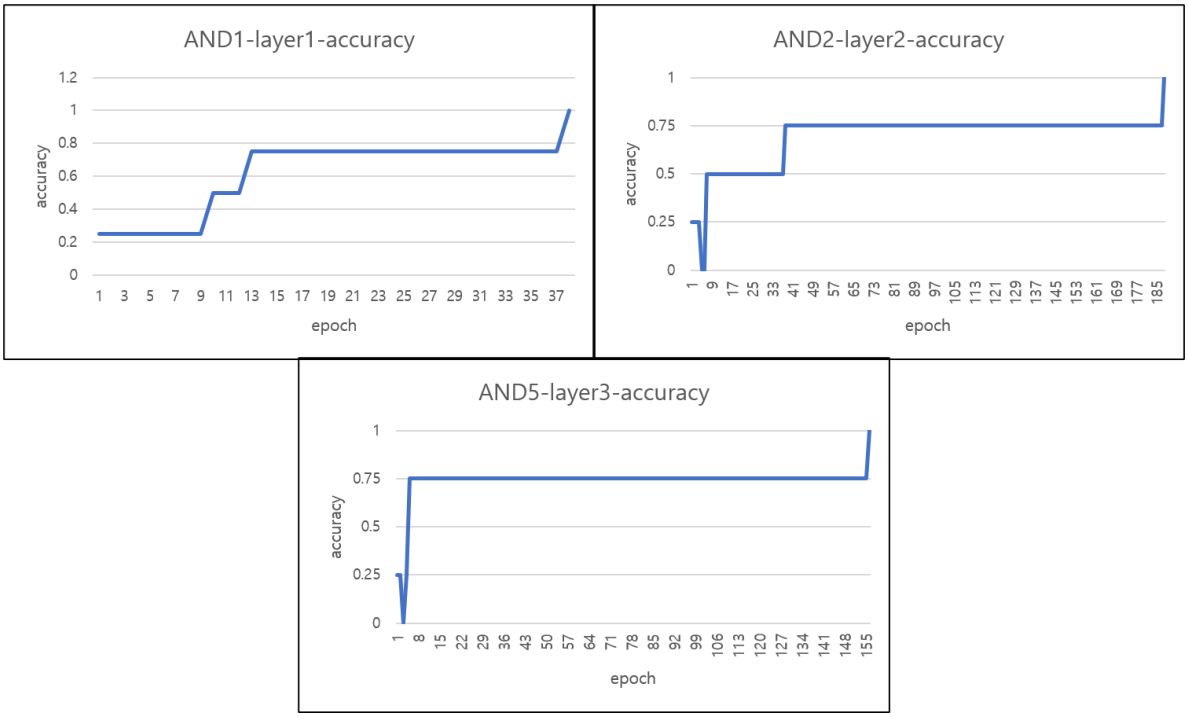
1. AND gate 실행결과

결과적으로 and 연산은 1-layer 신경망, 2-layer 신경망, 3-layer 신경망 모든 신경망에서 정상적으로 동작한다. Epoch는 신경망이 복잡할수록 무수히 증가했으나, 100%의 정확성을 보인다.



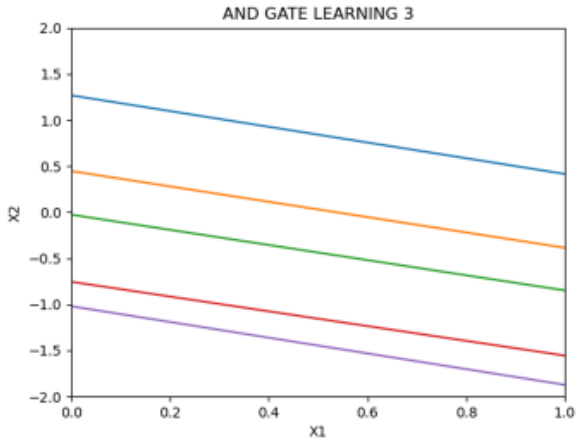
〈AND 게이트 Loss 그래프〉

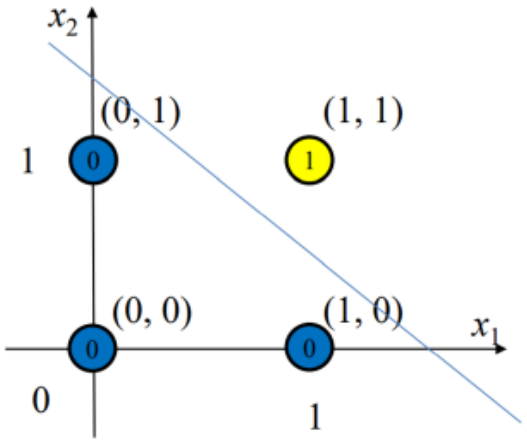
Loss 그래프를 보면 다른 게이트와 비교적으로 완만하게 곡선이 표현되었다. 1-layer 신경망에서 이전 과제#2와 동일한 그래프 곡선이 나타났다. 2-layer 신경망에서 초기에 랜덤한 weight과 bias로 인해 27.816 loss가 제시되었으나 epoch 5번째에서 빠르게 1보다 작은 loss 값을 보임을 알 수 있다. 자세한 데이터는 loss.txt 파일을 참고한다. 이후 완만하게 0.2에 근사하면서 학습을 종료했다. 3-layer 신경망에서 2-layer 신경망과 마찬가지로 초기에 29.859 loss로 시작했으나 epoch 4번째에 0.7보다 작은 loss 값을 보이면서 완만하게 감소하는 자료를 보인다. 0.2에 근사하면서 accuracy가 1이 되면서 학습을 종료했다.



〈AND 게이트 Accuracy 그래프〉

Accuracy 그래프는 loss 그래프와 다르게 증가하는 양상을 보인다. Target과 일치하는 정확도를 측정하여, 결과값이 일치할수록 1에 가까워진다. Loss의 감소율에 따라 accuracy의 증가율도 변화함을 알 수 있다.





〈AND 게이트 Learning 그래프〉

learning 그래프를 살펴보면 weight 값이 예측되는 그래프와 유사하게 학습됨을 알 수 있다. 위의 Loss 그래프와 일치하는 데이터는 아니지만, 같은 코드로 프로그램을 돌려 출력된 W1과 W2 그리고 bias 데이터를 가공하여 python 언어를 활용하여 그린 AND 연산 Learning 과정 그래프이다. 관련코드는 첨부파일 중 그래프코드(Learning)을 참고한다. 인공지능 강의시간에 학습한 그래프와 유사하게 직선이 움직임을 확인할 수 있다. (1 , 1)과 (0, 1), (1, 0) 사이로 직선이 움직이는 동향을 보인다. 대체로 유사한 기울기로 표현된다. 참고한 이미지는 강의자료를 참고하였음을 출처한다.

epoch	input	output	result
1			
	(0,0)	4.73212	1
	(0,1)	4.8276	1
	(1,0)	3.2476	1
	(1,1)	2.86342	1

epoch	input	output	result
78			
	(0,0)	0.210851	0
	(0,1)	0.362086	0
	(1,0)	0.28781	0
	(1,1)	0.366056	0

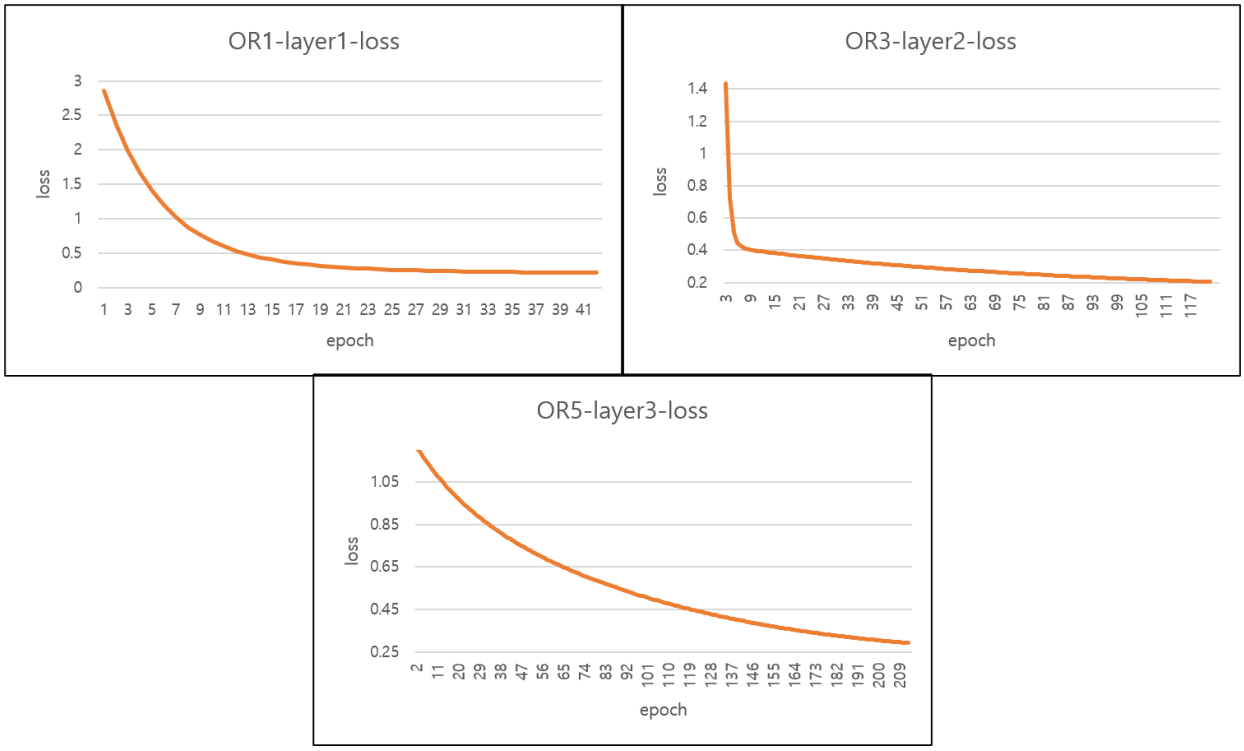
epoch	input	output	result
156			
	(0,0)	0.02767	0
	(0,1)	0.340339	0
	(1,0)	0.277198	0
	(1,1)	0.501022	1

〈AND 3-layer output과 result 데이터〉

자세한 데이터는 result.txt 파일을 참고한다. 초기에는 랜덤한 값들로 인한 계산을 통해 터무니없는 output 값이 도출했다. Multi 신경망 학습을 진행하면서 점점 원하는 result가 출력하는 output 계산값이 나왔다. Sigmoid 방식을 활용했기에 output 값이 0.5 이상이면 1, 그렇지 않으면 0이다. 따라서 epoch 156번째에 target과 동일한 result 도출 후 학습을 종료함을 알 수 있다.

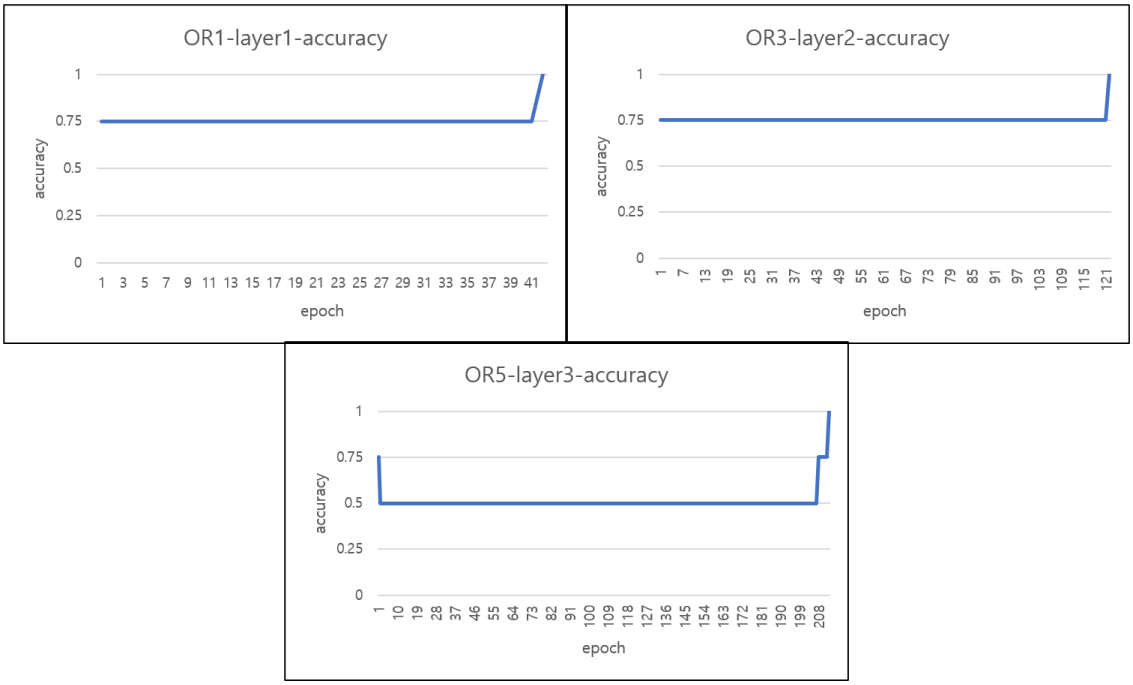
2. OR gate 실행결과

or 연산은 and 연산과 유사하게 1-layer 신경망, 2-layer 신경망, 3-layer 신경망 모든 신경망에서 정상적으로 동작한다. Epoch는 신경망이 복잡할수록 무수히 증가했으나, 100%의 정확성을 보인다.



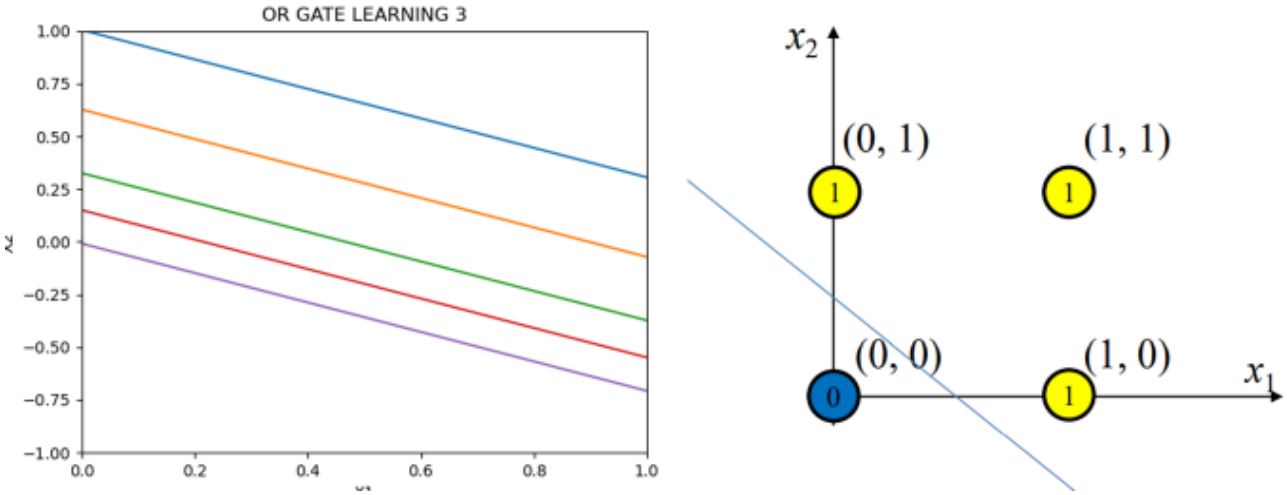
〈OR 게이트 Loss 그래프〉

Loss 그래프를 보면 다른 게이트와 비교적으로 완만하게 곡선이 표현되었다. 1-layer 신경망에서 이전 과제#2와 동일한 그래프 곡선이 나타났다. 2-layer 신경망에서 초기에 랜덤한 weight과 bias로 인해 14.4418 loss가 제시되었으나 epoch 4번째에서 빠르게 1보다 작은 loss 값을 보임을 알 수 있다. 자세한 데이터는 loss.txt 파일을 참고한다. 이후 완만하게 0.2에 근사하면서 학습을 종료했다. 3-layer 신경망에서 2-layer 신경망과 마찬가지로 초기에 45.7695 loss로 시작했으나 epoch 17번째에 1보다 작은 loss 값을 보이면서 완만하게 감소하는 자료를 보인다. 0.29에 근사하고 accuracy가 1이 되면서 학습을 종료했다.



〈OR 게이트 Accuracy 그래프〉

Accuracy 그래프는 loss 그래프와 다르게 증가하는 양상을 보인다. Target과 일치하는 정확도를 측정하여, 결과값이 일치할수록 1에 가까워진다. Loss의 감소율에 따라 accuracy의 증가율도 변화함을 알 수 있다. Output 계산값이 1보다크면 result 데이터가 1이 되고, target 자체에 1이 많아서 or 게이트는 정확도가 학습과정에 우수하게 나타난다. 하지만 target 0 인 (0,0)을 학습하는 과정에서 신경망이 local minimun에 빠져나오는데 수많은 epoch가 걸렸음을 추측한다.



〈OR 게이트 Learning 그래프〉

위의 Loss 그래프와 일치하는 데이터는 아니지만, 같은 코드로 프로그램을 돌려 출력된 W1과 W2 그리고 theta 데이터를 가공하여 python 언어를 활용하여 그린 OR 연산 Learning 과정 그래프이다. 관련 코드는 첨부파일 중 그래프코드(Learning)을 참고한다. 인공지능 강의시간에 학습한 그래프와 유사하게 직선이 움직임을 확인할 수 있다. 각각의 데이터와 초기 weight, bias 값에 따라 기울기가 다른 직선이 보여진다. 그러나 통일적으로 (0 , 0)과 (0, 1), (1, 0) 사이로 직선이 움직이는 동향을 보인다. 참고한 이미지는 강의자료를 참고하였음을 출처한다.

epoch	input	output	result	epoch	input	output	result
1				71			
	(0,0)	9.21886	1		(0,0)	0.936385	1
	(0,1)	3.52594	1		(0,1)	0.423354	0
	(1,0)	1.39009	1		(1,0)	0.814204	1
	(1,1)	0.861824	1		(1,1)	0.988546	1
epoch	input	output	result	epoch	input	output	result
142				213			
	(0,0)	0.684247	1		(0,0)	0.499725	0
	(0,1)	0.466127	0		(0,1)	0.502714	1
	(1,0)	0.81454	1		(1,0)	0.804627	1
	(1,1)	1.11428	1		(1,1)	1.22035	1

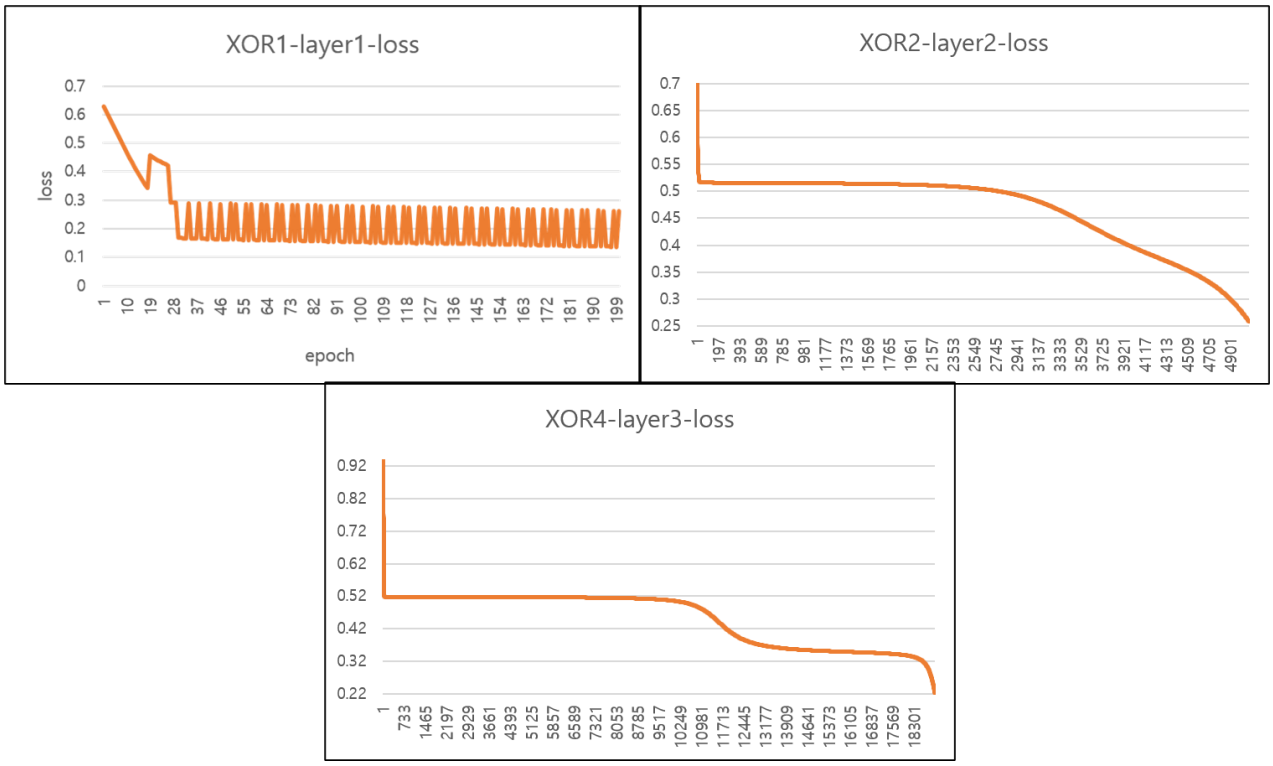
〈OR 3-layer output과 result 데이터〉

자세한 데이터는 result.txt 파일을 참고한다. 초기에는 랜덤한 값들로 인한 계산을 통해 input에 따라 터무니없는 output 값들이 도출했다. Multi 신경망 학습을 진행하면서 점점 원하는 result가 출력하는 output 계산 값이 나왔다. Sigmoid 방식을 활용했기에 output 값이 0.5 이상이면 1, 그렇지 않으면 0이다. (0,0) input 과 (0,1) input을 학습하면서 epoch가 오래 진행했고, epoch 213번째에 target과 동일한 result 도출 후 학습을 종료함을 알 수 있다.

3. XOR gate 실행결과

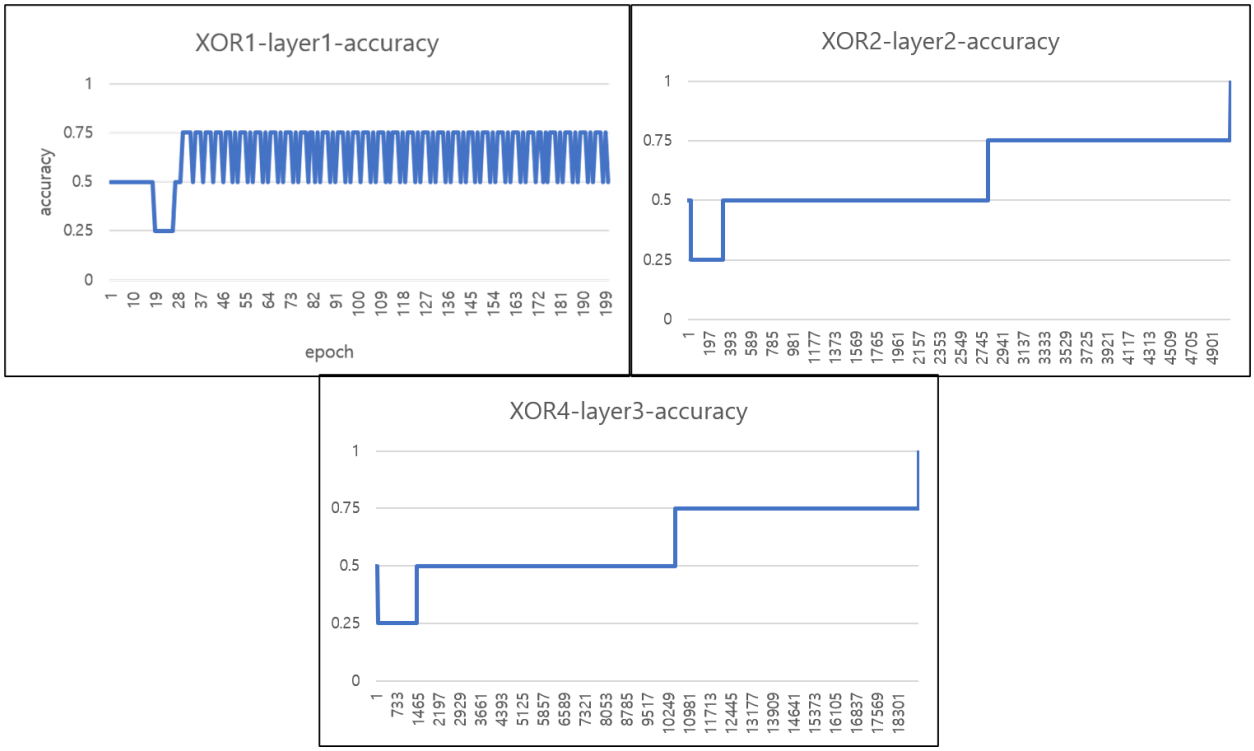
XOR 연산을 실행하는 Multi 신경망 과정에서 아직 이해하지 못한 에러가 발생했다. 이는 파일입출력을 담당하는 ofstream에서 프로그램이 동작하지 않았다. 그래서 2-layer 신경망과 3-layer 신경망 실험에 대한 result.txt와 weight.txt 파일 데이터는 추출하지 못했다. 추후에 부족한 에러와 오류를 방지하는 코드를 설계하여 추가할 예정이다. (코드의 한계점) 또한 이와 관련있는 learning 그래프를 그리지 못했다. (데이터 전처리의 한계점)

XOR 게이트의 경우, AND 게이트와 OR 게이트와 다른 양상의 실험을 보였다. 1-layer 신경망은 지난 과제#2와 동일하게 강제종료 했다. 초반에는 다소 급격하게 Loss 값을 감소해 나갔으나 진동이 보이는 과정에는 약 50%~ 75%의 정확성이 나타났다. 이를 통하여 1-layer 신경망에서 XOR 게이트는 적절한 weight, bias 값을 구할 수 없을 알 수 있다. 이를 수행하기 위해서는 비선형 함수 즉, 단층 신경망으로 분류하기에 불가능하다는 의미이다. 2-layer 신경망과 3-layer 신경망에서 정상적으로 동작했다. Epoch가 무수히 증가했으나, multi 신경망에서 100%의 정확성을 보인다.



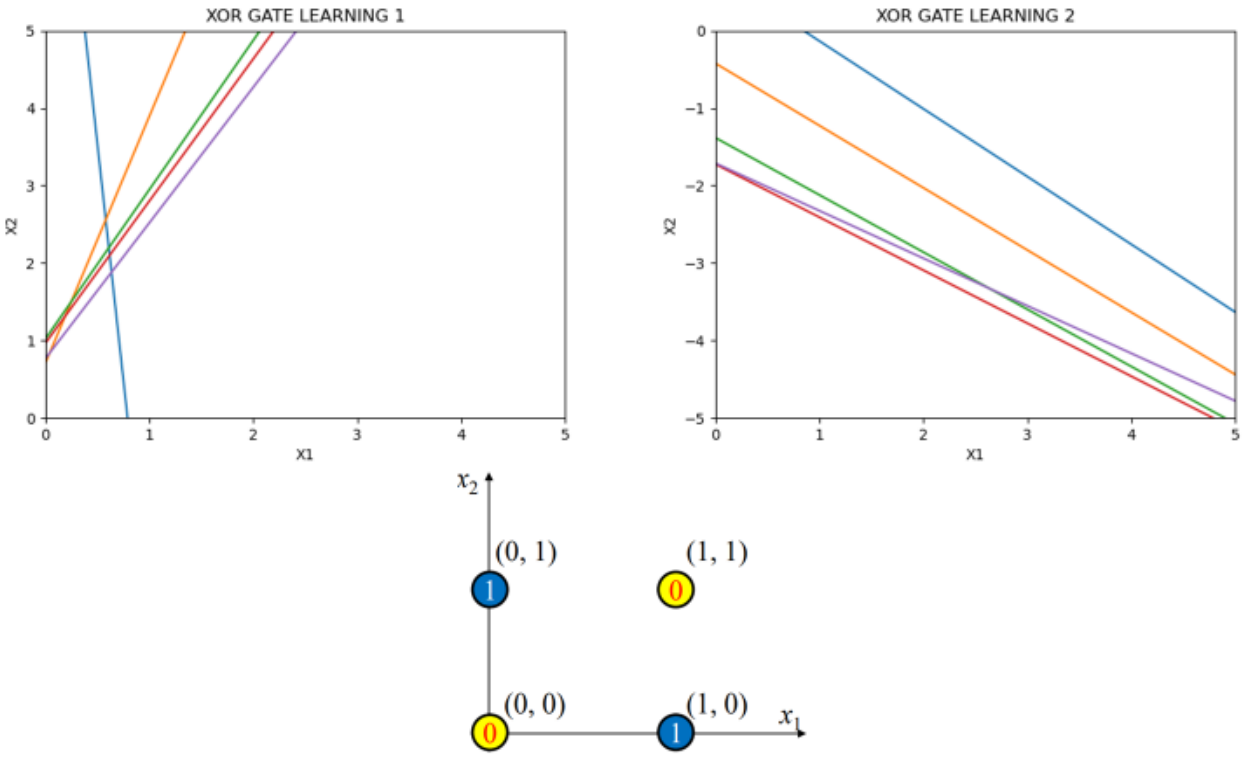
〈XOR 게이트 Loss 그래프〉

Loss 그래프를 양상은 단일과 멀티 신경망의 차이로 구분된다. 1-layer 신경망에서 이전 과제#2와 동일한 그래프 곡선이 나타났다. 초반에는 loss 값이 줄어지는 양상을 보였으나, 약 0.1과 0.3 사이를 진동하는 그래프가 그려진다. 2-layer 신경망에서 초기에 랜덤한 weight과 bias로 인해 터무니없는 loss값으로 시작했으나 빠르게 0.6보다 작은 loss 값이 도출하고 그 이후 완만한 곡선을 그리며 감소하다가 정확도가 1이 되어 학습을 종료했다. 자세한 데이터는 loss.txt 파일을 참고한다. 3-layer 신경망에서 2-layer 신경망과 마찬가지로 epoch 50번째부터 0.52로 loss 값이 급격하게 감소하고 이후 완만에서 동작하였다. Epoch 19027번째에 loss가 0.22에 근사하여 학습을 종료했다. Multi 신경망은 loss가 특정 시점을 지나 급격히 loss를 줄이는 동향이 있다. 이는 local minimum에서 빠져나오지 못하고 횡보하다가 특정 시점에 탈출하여 global minimum(혹은 더 우수한 local minimum)을 향해 학습했을 것으로 추측한다.



〈XOR 게이트 Accuracy 그래프〉

Accuracy 그래프는 loss 그래프와 다르게 증가하는 양상을 보인다. Target과 일치하는 정확도를 측정하여, 결과값이 일치할수록 1에 가까워진다. 2-layer 신경망과 3-layer 신경망에서 정확도가 0.5에서 0.25로 잠시 떨어졌다가 다시 회복 후, 점차 우수한 accuracy 양상을 보인다. 이는 우수한 local minimun에 빠져 수많은 학습을 통해 global minimun이 아니라는 점을 알아내는데 걸린 시점이라 추측한다.



〈XOR 게이트 Learning 그래프〉

위의 Loss 그래프와 일치하는 데이터는 아니지만, 같은 코드로 프로그램을 돌려 출력된 W1과 W2 그리고 bias 데이터를 가공하여 python 언어를 활용하여 그린 XOR 연산 Learning 과정 그래프이다. 관련 코드는 첨부파일 중 그래프코드(Learning)을 참고한다. 각각의 데이터와 초기 weight, bias 값에 따라 기울기가 다른 직선의 동향이 보여진다. 가장 아래 XOR 연산표를 보면 1-layer 신경망은 input 값을 적절히 분류할 수 없는 것이다. 신경망이 분류가능한 요소는 최대 3개이다. 학습과정에서 정확히 분류하지 못한 나머지 1개를 분류하려고 직선이 이동하고, 나머지 1개가 올바르게 분류되면 이미 분류된 데이터 중에서 분류범위를 벗어나게 된다. 이러한 반복과정이 Loss 그래프의 진동으로 표현되며 Loss 가 0에 도달할 수 없을 보여준다. 이미지는 강의자료를 참고하였음을 출처한다. 2-layer 신경망과 3-layer 신경망에서는 앞서 언급한 코드의 한계점과 데이터전처리의 한계점으로 인해 result.txt와 weight.txt 데이터를 추출하지 못했다. 그래서 멀티 신경망에서 learning 그래프를 자세히 다루지 못한 점이 아쉽다.

epoch	input	output	result	epoch	input	output	result	epoch	input	output	result
1				500				1000			
	(0,0)	1.52	1		(0,0)	0.498125	0		(0,0)	0.507435	1
	(0,1)	2.3031	1		(0,1)	0.491271	0		(0,1)	0.492368	0
	(1,0)	2.33654	1		(1,0)	0.491704	0		(1,0)	0.49237	0
	(1,1)	3.09691	1		(1,1)	0.515148	1		(1,1)	0.507756	1

<OR 1-layer output과 result 데이터>

자세한 데이터는 result.txt 파일을 참고한다. 초기에는 랜덤한 값들로 인한 계산을 통해 input에 따라 터무니없는 output 값들이 도출했다. Sigmoid 방식을 활용했기에 output 값이 0.5 이상이면 1, 그렇지 않으면 0이다. 단일 신경망에서는 다음과 같이 target과 일치하지 않는 result의 값이 반복적으로 도출하면서 학습을 스스로 멈추지 못한다. 따라서 강제종료를 진행한다.

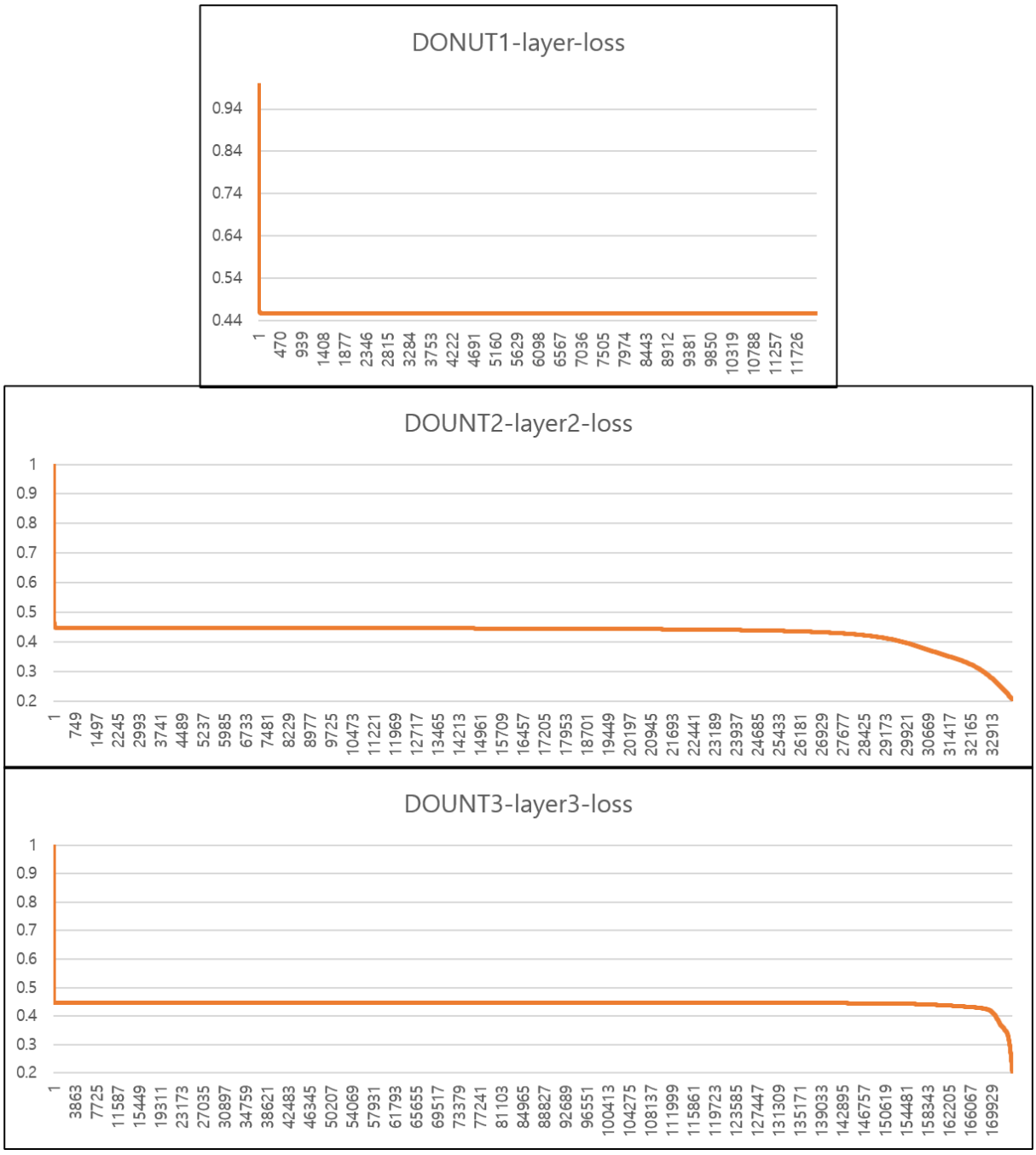
4. DONUT 실행결과

Donut 모양의 input 데이터에 대하여 실행하는 Multi 신경망 과정에서 아직 이해하지 못한 에러가 발생했다. 이는 파일입출력을 담당하는 ofstream에서 프로그램이 동작하지 않았다. 그래서 2-layer 신경망과 3-layer 신경망실험에 대한 result.txt와 weight.txt 파일 데이터는 추출하지 못했다. 추후에 부족한 에러와 오류를 방지하는 코드를 설계하여 추가할 예정이다. (코드의 한계점) 또한 이와 관련있는 learning 그래프를 그리지 못했다. (데이터 전처리의 한계점)

도넛모양의 input 값과 output 값은 신선한 실험이었다. 비록 우수한 멀티 신경망 프로그램인지 확신할 수 없으나, 새로운 도전이 과제를 더 흥미롭게 했다.

DONUT의 경우, AND 게이트와 OR 게이트와 다른 양상의 실험을 보였다. 모든 신경망 실험에서 초반에는 다소 급격하게 Loss 값을 감소했다. 1-laye 신경망에서 빠르게 loss가 0.4에 도달 후 더 이상 학습이 진행되지 않아 강제종료했다. 이를 통하여 1-layer 신경망에서 적절한 weight, bias 값을 구할 수 없음 알 수 있다. 이를 수행하기 위해서는 비선형 함수 즉, 단층 신경망으로 분류하기에 불가능하다는 의미이다. 2-layer 신경망과 3-layer 신경망에서도 빠르게 loss가 약 0.44에 도달 후 무수히 많은 epoch에 걸쳐 있다가 후반에 빠르게 loss값을 감소하여 스스로 학습을 종료했다. 도넛의 데이터는 multi 신경망에서 100%의 정확성을 보인다. 1-layer를 분류하는데 실패한 이유는 XOR 연산과 동일한 이유음을 추측한다. 또한 멀티 신경망에서 loss 값이 감소하지 않는 정체기는 local minimum을 빠져

나오지 못하다가 특정 시점이 이를 탈출하고 global minimum (혹은 더 우수한 local minimum)으로 다가갔던 것을 추측한다.



〈DONUT Loss 그래프〉

위에 언급한 내용과 동일하다. 자세한 데이터는 loss.txt 파일을 참고한다.

〈DONUT Accuracy 그래프〉

Accuracy 그래프는 loss 그래프와 다르게 증가하는 양상을 보인다. Target과 일치하는 정확도를 측정하여, 결과값이 일치할수록 1에 가까워진다. 모든 신경망에서 빠르게 0.88이라는 정확도를 보인다. 그러나 이후 무수히 많은 epoch 동안 이를 해결하지 못한다. 1-layer 신경망에서는 강제종료했다. 이는 우수한 local minimun에 빠져 수많은 학습을 통해 global minimun이 아니라는 점을 알아내는데 걸린 시점이라 추측한다.

도넛데이터의 경우, 앞서 언급한 코드의 한계점과 데이터전처리의 한계점으로 인해 result.txt와 weight.txt 데이터를 추출하지 못했다. 그래서 신경망에 대한 learning 그래프를 자세히 다루지 못한 점이 아쉽다.

epoch	input	output	result	epoch	input	output	result	epoch	input	output	result
1				1000				2000			
	(0,0)	0.9266	1		(0,0)	0.120031	0		(0,0)	0.120031	0
	(0,1)	1.23087	1		(0,1)	0.118375	0		(0,1)	0.118375	0
	(1,0)	1.07125	1		(1,0)	0.119384	0		(1,0)	0.119384	0
	(1,1)	1.3496	1		(1,1)	0.115363	0		(1,1)	0.115363	0
	(0.5,1)	1.21382	1		(0.5,1)	0.109862	0		(0.5,1)	0.109862	0
	(1,0.5)	1.11071	1		(1,0.5)	0.108249	0		(1,0.5)	0.108249	0
	(0,0.5)	0.928195	1		(0,0.5)	0.104589	0		(0,0.5)	0.104589	0
	(0.5,0)	0.843187	1		(0.5,0)	0.104184	0		(0.5,0)	0.104185	0
	(0.5,0.5)	0.961546	1		(0.5,0.5)	0.099963	0		(0.5,0.5)	0.099963	0

		epoch	input	output	result	epoch	input	output	result		
		5000				10000					
			(0,0)	0.120031	0		(0,0)	0.120031	0		
			(0,1)	0.118375	0		(0,1)	0.118375	0		
			(1,0)	0.119384	0		(1,0)	0.119384	0		
			(1,1)	0.115363	0		(1,1)	0.115363	0		
			(0.5,1)	0.109862	0		(0.5,1)	0.109862	0		
			(1,0.5)	0.108249	0		(1,0.5)	0.108249	0		
			(0,0.5)	0.104589	0		(0,0.5)	0.104589	0		
			(0.5,0)	0.104185	0		(0.5,0)	0.104185	0		
			(0.5,0.5)	0.099963	0		(0.5,0.5)	0.099963	0		

<DONUT 1-layer output과 result 데이터>

자세한 데이터는 result.txt 파일을 참고한다. 초기에는 랜덤한 값들로 인한 계산을 통해 input에 따라 터무니없는 output 값들이 도출했다. Sigmoid 방식을 활용했기에 output 값이 0.5 이상이면 1, 그렇지 않으면 0이다. 단일 신경망에서는 다음과 같이 target과 일치하지 않는 result의 값이 반복적으로 도출하면서 학습을 스스로 멈추지 못한다. (1,1)에서 result=1을 도출해야 한다. 그러나 epoch 10000번째동안에도 이를 구하지 못함을 알수 있다. 따라서 강제종료를 진행한다.

5. 한계점

Donut 모양과 XOR연산에서 input 데이터에 대하여 실행하는 Multi 신경망 과정 중 아직 이해하지 못한 에러가 발생했다. 이는 파일입출력을 담당하는 ofstream에서 프로그램이 동작하지 않았다. 그래서 2-layer 신경망과 3-layer 신경망실험에 대한 result.txt와 weight.txt 파일 데이터는 추출하지 못했다. 추후에 부족한 에러와 오류를 방지하는 코드를 설계하여 추가할 예정이다. (코드의 한계점)

Donut 모양과 XOR연산에서 몇몇의 result.txt와 weight.txt 파일 데이터를 추출하지 못했다. 따라서 이에 대한 learning 그래프를 그리지 못했다. (데이터 전처리의 한계점)

c 즉, learning rate에 따라 학습 속도나 loss, accuracy의 변화율에 영향을 미치는 것으로 추측한다. 이는 프로그램을 돌리면서 직접적으로 연관이 있음을 알게 되었다. 그러나 좀 더 공부해보니, 이를 계산하는 다양한 알고리즘이 있음을 깨달았다. [https://forensics.tistory.com/28], [https://sacko.tistory.com/38](학습과정에서 learning rate의 영향)

〈 III. 출처 및 참고자료〉

원리 참고

- https://eungbean.github.io/2018/08/26/udacity-107-MLP-Reminder/
- https://ynebula.tistory.com/22
- https://ang-love-chang.tistory.com/26
- https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- https://wikidocs.net/37406

한계점 참고

- https://sacko.tistory.com/38

- https://forensics.tistory.com/28

코딩 참고

- https://psychoria.tistory.com/774
- https://jhnyang.tistory.com/363

데이터 처리 참고

- https://m.blog.naver.com/top-dream/221054393897

이미지 참고

- https://www.pinterest.co.kr/pin/441986150938119758/
- https://eungbean.github.io/2018/08/26/udacity-107-MLP-Reminder/
- 강의자료 : AI-2021-YU-02.pdf
- 강의자료 : AI-2021-YU-05-MLP-Training.pdf