

**과제 #3 Multi-Layer Perceptron 구현**

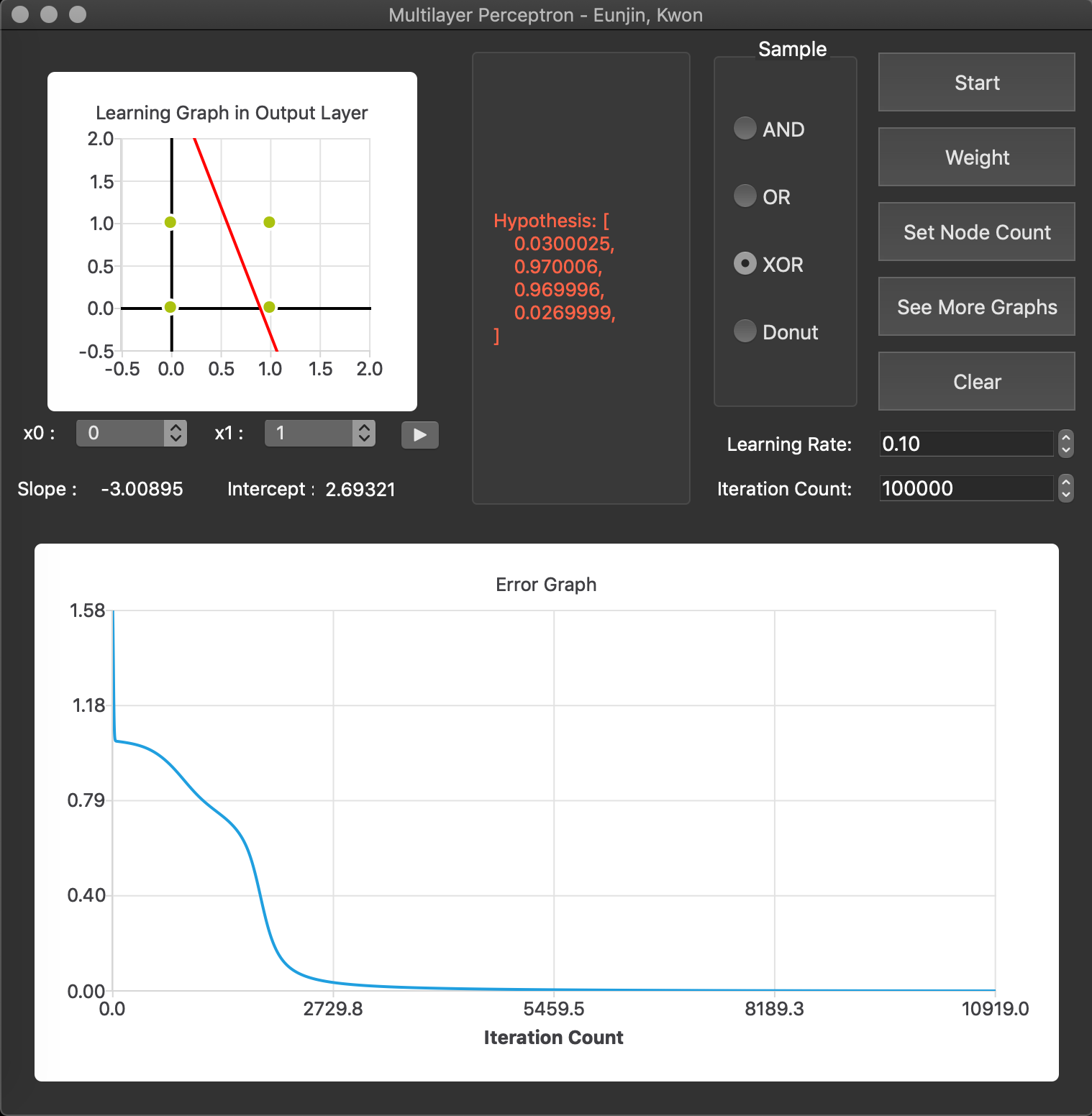
컴퓨터과학부

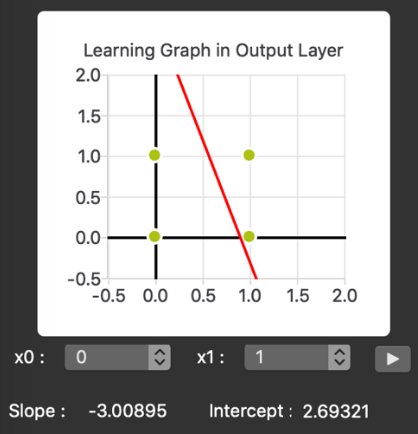
2016920004

권은진

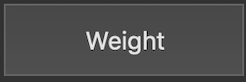
1. 프로그램 개요
   1. 언어: C++
   2. 사용 라이브러리: Qt 5.13 \* 반드시 설치해야 실행가능 \*
   3. 실행 환경: **Linux, Mac OS**
   4. 소스코드: <https://github.com/ke2ek/AI_assignment/tree/master/MultiLayerPerceptron>
   5. 클래스 구조
      1. MainWindow - GUI 용 클래스

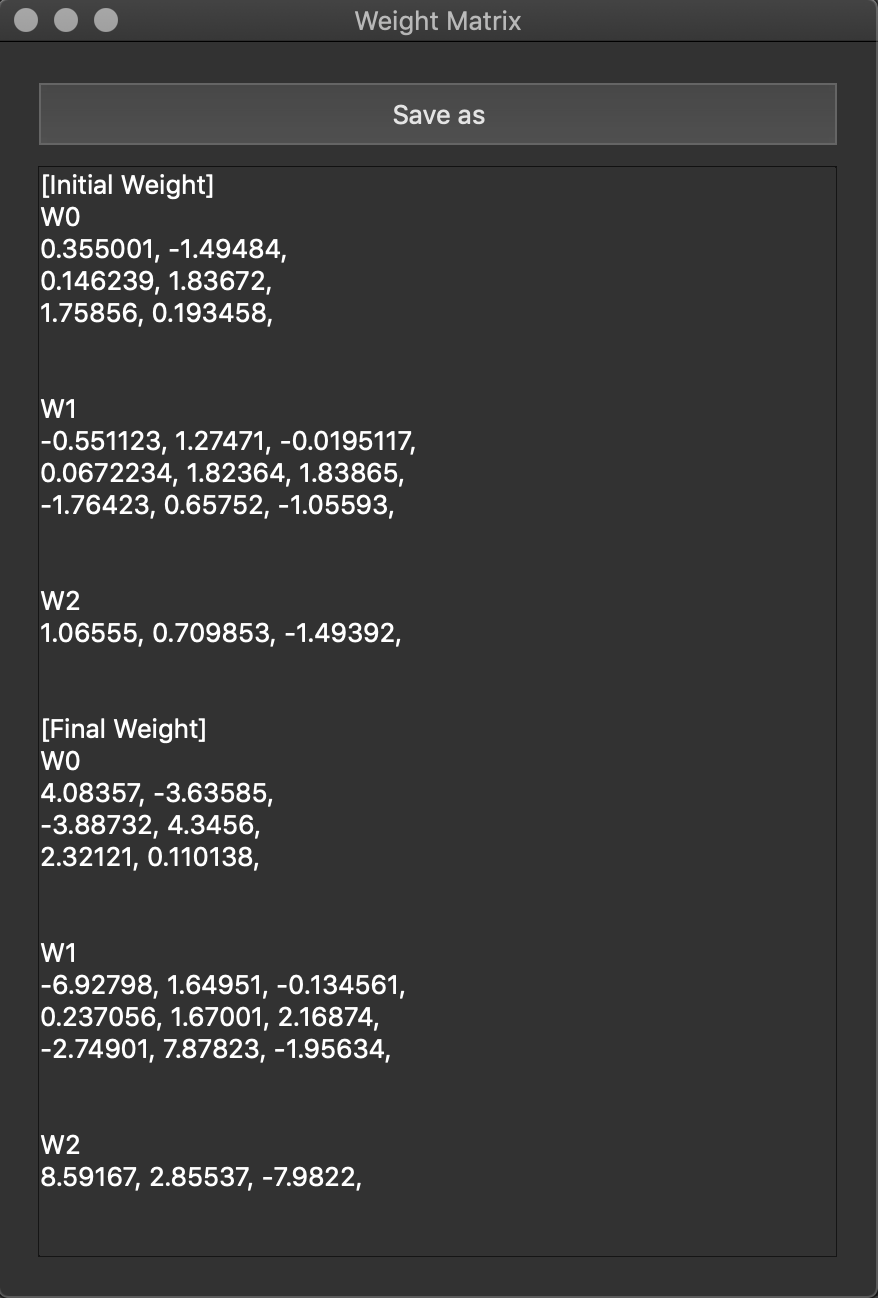
* … (생략)
* iterated\_once() - 매 1회 반복할 때마다 GUI 에 필요한 데이터(weight, input 등)들을 저장하고 UI를 수정하는 함수
* iterated\_all() - 학습이 끝났을 때 GUI에 필요한 데이터(error 등)들을 저장하고 UI 를 수정하는 함수
  + 1. LayerSettingsWindow - layer 개수, layer 당 노드 개수를 설정하는 GUI 용 클래스
* … (생략)
* add\_layer() - 레이어 정보 위젯을 추가하는 함수
* remove\_layer() - 레이어 정보 위젯을 삭제하는 함수
  + 1. ReplayThread - 학습 과정을 다시 그래프에 재생해주는 쓰레드 클래스
* … (생략)
* set() - 현재 레이어와 입력 노드의 개수를 저장하는 함수
* add\_data() - 해당 노드에 필요한 데이터(weight, input)들을 추가하는 함수, 학습 과정에서 매 반복마다 호출된다.
* set\_data() - 해당 노드에 필요한 데이터(weight, input)들을 저장하는 함수, 학습이 끝나고 호출된다.
* drawn() - 매 학습마다 그래프 상에 필요한 정보를 LearningGraph 로 보내는 시그널(signal) 함수
* run*()* - thread 함수로 실제 학습 과정을 재생하기 위한 연산을 수행
  + 1. LearningGraph - 각 노드의 학습 과정을 그래프에 보여주는 GUI 클래스
* … (생략)
* draw\_line() - 매 학습마다 직선을 그래프에 그리는 함수, ReplayThread 의 drawn() 시그널을 처리(handle)
  + 1. ReplayWindow - Hidden Layer 의 각 노드별로 학습 그래프를 모두 보여주는 GUI 클래스
* … (생략)
* add\_data() - 매 학습마다 필요한 데이터(weight, input)들을 추가하는 함수, 학습 과정에서 매 반복마다 호출된다.
  + 1. Classifier - MultiLayer Perceptron 를 구현한 쓰레드 클래스
* … (생략)
* ready() - 학습 전, 필요한 데이터(레이어별 노드 개수, 학습률, input, output, 최대 반복 횟수)들을 저장하고 멤버 변수(weight 등)를 초기화하는 함수
* dot\_product() - 학습 시, weight 과 input 의 행렬곱 연산(내적)을 수행하는 함수
* forward() - 학습 시, fowarding pass 를 수행하는 함수
* update\_weights() - 학습 시, 역전파법으로 weight 을 업데이트하는 함수
* changed\_weight() - backpropagation 이후 MainWindow 로 weight, input, bias 등의 데이터를 보내는 시그널(signal) 함수
* done() - 학습이 끝나면 MainWindow 로 error 등의 데이터를 보내는 시그널(signal) 함수
  1. 인터페이스
     1. MainWindow

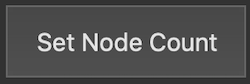


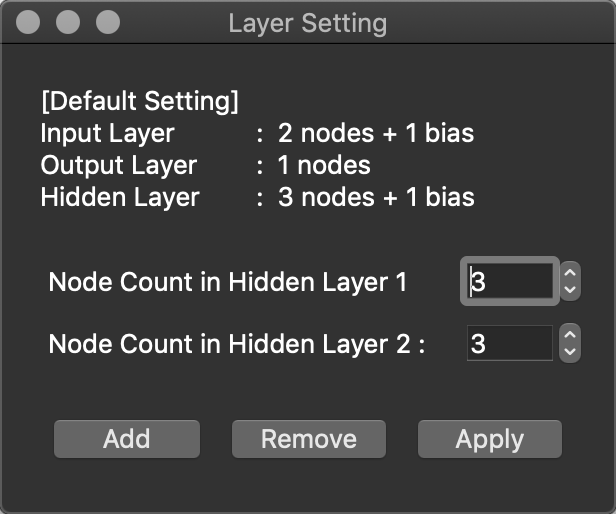


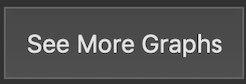
 에서 마지막 hidden layer 의 노드(입력 노드) 번호 선택하고  버튼 클릭 시, Output 노드의 학습 과정의 그래프를 다시 보여준다. 아래의 레이블 2개는 기울기와 y 절편을 의미한다.

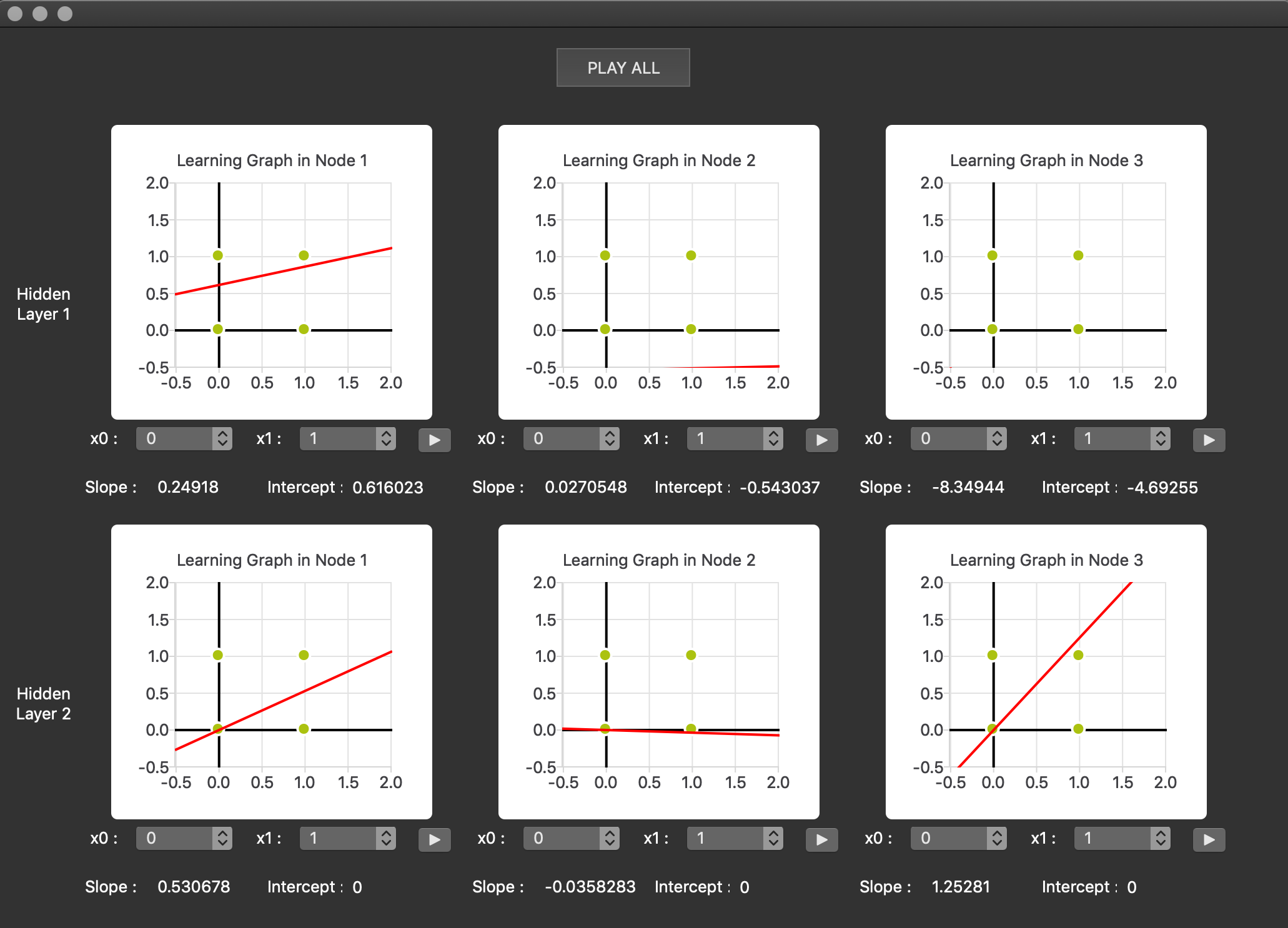
 : 학습 종료 후, 초기/최종 weight 을 보여주는 버튼, 아래의 윈도우에서 별도의 파일로 저장가능하며, weight 은 레이어 간 전달되는 값들의 Matrix 형태로 보여진다.

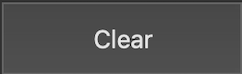


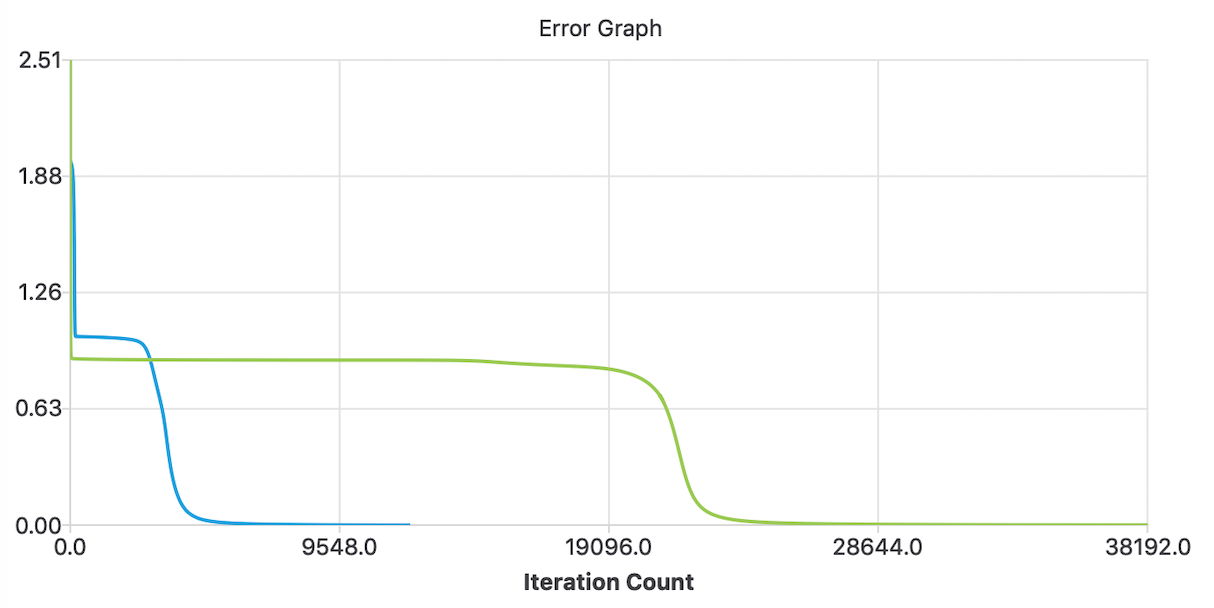
 : 학습 시작 전에 레이어별 노드 개수를 설정하는 버튼

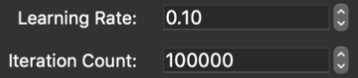


 : 학습이 끝나고, hidden layer 의 다른 노드에 대한 학습 그래프를 보여주는 버튼, 아래 윈도우에서 전체 재생 또는 각 노드별 재생이 가능하다.



 : 모든 UI 를 초기화하는 버튼, 학습이 끝나고 다른 샘플에 대해 학습하면 아래와 같이 오류 그래프가 겹쳐서 보여지므로, 하나만 보고 싶을 때 이 버튼을 누르고 학습을 해야한다.



: 학습률 및 최대 반복 횟수를 설정하는 콤보박스

1. 실행 결과
   1. 고려사항
      1. 모든 실험의 learning rate 은 0.1 이며, 가중치는 -2.0 ~ 2.0 사이의 랜덤값으로 초기화된다. 허용 오차는 0.03 이다.
      2. 모든 실험에서 각 샘플마다 10번의 시도를 하였고, 10번 모두 목표값에 도달하지 못한 경우에 실패한 것으로 간주하였다. 이는 가중치가 랜덤값이기 때문에 적절한 가중치로 초기화되지 못한 경우 학습이 제대로 되지 않는다는 점을 고려한 것이다.
      3. 학습이 제대로 된 경우, 모든 시도 중 하나의 시도에 대해서만 결과로 첨부한다.
   2. Hidden Layer 의 노드 개수(bias 노드는 제외)에 따른 실험
      1. 노드가 2개일 때



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. 노드가 3개일 때



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. 노드가 4개일 때



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. 노드가 8개일 때

(이미지 생략)

|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |
|  |  |

* 1. Hidden Layer 의 개수에 따른 실험 1 (각 층당 노드 개수 2개)
     1. Hidden Layer 2개



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. hidden layer 3개



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. hidden layer 4개



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. hidden layer 8개

(이미지 생략)

|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* 1. Hidden Layer 의 개수에 따른 실험 2 (각 층당 노드 개수 3개)
     1. hidden layer 2개



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. hidden layer 3개



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. hidden layer 4개



|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* + 1. hidden layer 8개

(이미지 생략)

|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* 1. Hidden Layer 의 노드 개수가 다른 경우
     1. 뉴럴 네트워크 구조

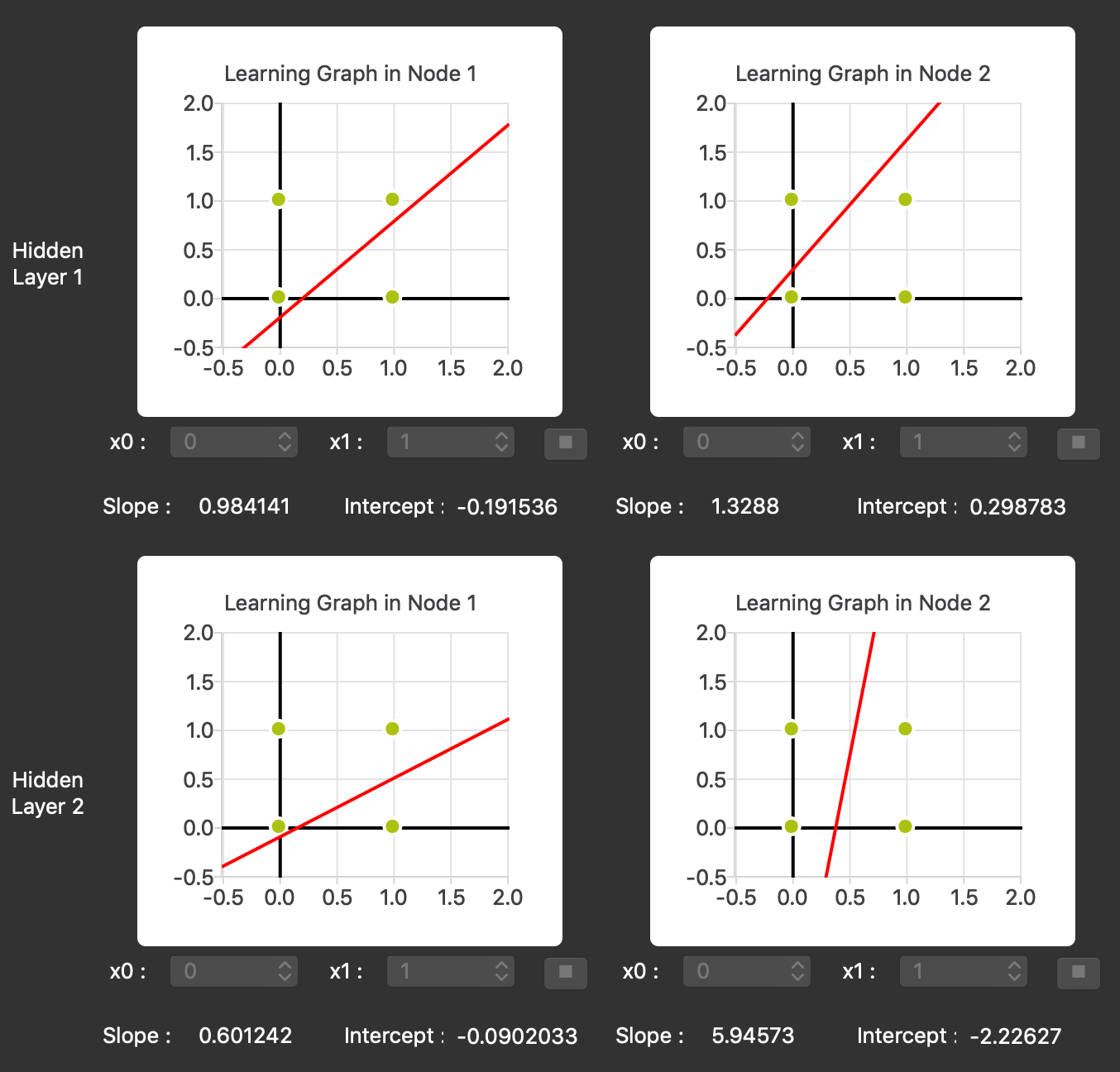


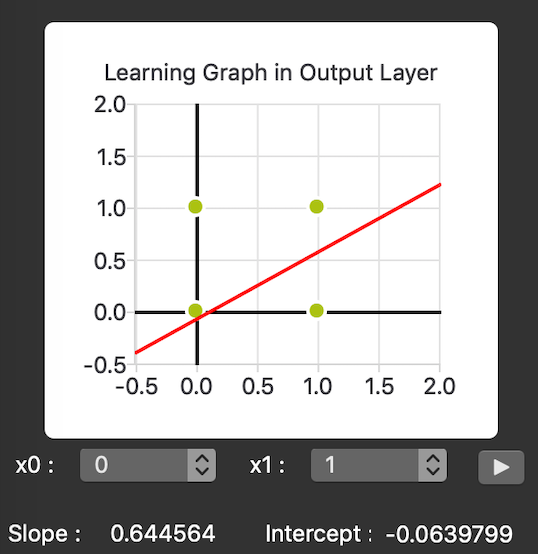
|  |  |  |
| --- | --- | --- |
| Sample | Hypothesis | Error Graph |
| AND  Gate |  |  |
| OR  Gate |  |  |
| XOR  Gate |  |  |
| Donut |  |  |

* 1. 2차원 직선 그래프
     1. 2차원에서 노드의 직선 그래프를 표현하기 위해, 모든 실험이 아닌 노드가 2개인 hidden layer 2개의 실험에 대해서만 결과를 첨부하였다.
     2. 아래의 실험에서 learning rate 은 0.5이다.



|  |  |  |
| --- | --- | --- |
| XOR  Gate |  |  |





1. 결론
   1. Hidden Layer 의 노드 개수(bias 노드는 제외)에 따른 실험
      1. 노드가 2개일 때 : XOR, Donut 학습되지 않음

XOR Gate 는 점차 정답에 가까워지는 듯 했으나, Donut 실험에서는 반복 횟수가 7만을 넘어서부터는 hypothesis 값에 거의 변화가 없었다. 오류 그래프를 보면 알 수 있듯이 0.89 밑으로는 오류가 감소하지 않고 있다.

XOR, Donut 실험에서는 어느정도 오류가 감소하다가 최대반복횟수가 될때까지 더이상 오류가 감소하지 않는다는 것을 알 수 있다. 물론 적절하지 않은 가중치로 초기화되어 시간이 좀 더 걸린 걸 수도 있겠지만, AND, OR 의 경우 어떤 랜덤값을 가지던지 부드러운 오류곡선을 그리며, 정답을 찾아갔기 때문에 XOR, Donut 의 경우 학습이 제대로 되지 않는다고 볼 수 있다.

* + 1. 노드가 3개일 때 : Donut 학습되지 않음

XOR 은 hidden layer 에 노드가 3개는 있어야 학습이 가능하다는 것을 알 수 있다. Donut 은 여전히 학습이 되지 않고 있으며, 심지어는 이전 현상과 동일하게 특정 시점부터는 오류가 감소하지 않거나 거의 0에 가까운 값만큼 감소한 것을 알 수 있다.

* + 1. 노드가 4개일 때 : 모두 학습 완료

XOR, Donut 모두 계단형의 그래프가 많이 나왔다. 그래프 상에 가로로 긴 부분은 학습 중 적절한 가중치로 맞추는데 AND와 OR 에 비해 시간이 걸렸음을 의미한다. 초기 가중치가 적절하지 못할수록 계단형 그래프가 나올 확률이 높다고 볼 수 있는데, 그 이유는 오류값이 확 떨어질 때까지 계속 가중치를 갱신해줘야 하며 그 순간만큼은 오류가 매우 조금씩 감소하기 때문이다. 적절한 가중치로 맞춘 뒤로는 오류가 급격하게 떨어지며 정답에 가까워진다.

* + 1. 노드가 8개일 때 : 모두 학습 완료

XOR, Donut 모두 AND 와 OR 만큼 부드러운 오류 곡선이 나왔다. Donut 은 계단형도 종종 나왔는데 이는 오류가 클수록 가중치를 적절히 맞추는 시간이 걸리기 때문이었다.

이번 실험에서는 적절하지 못한 가중치라 하더라도, hidden layer 의 뉴런의 개수가 3개 이상이면 10만번의 반복횟수 안에 가중치를 적절하게 맞춰서 학습을 할 수 있었다.

즉, 뉴런이 2개(bias 제외)라면 복잡한 학습에 대해 가중치를 조정하는데 시간이 상당히 많이 걸리거나 너무 많은 학습으로 가중치가 0에 수렴할 수 있다.

* 1. Hidden Layer 의 개수에 따른 실험 1 (각 층당 노드 개수 2개)
     1. hidden layer 2개: Donut 학습되지 않음

Donut 을 제외하고 학습이 되긴 했으나 오류곡선이 계단형에 가까운 형태인 것으로 보아 학습에 상당한 시간이 필요했음을 알 수 있다. 오히려 layer 가 1개인 이전 실험보다 안 좋은 결과가 나타났다.

* + 1. hidden layer 3개: Donut 학습되지 않음

마찬가지로, Donut 을 제외하고 계단형이 많이 나왔다.

* + 1. hidden layer 4개: Donut 학습되지 않음

마찬가지로, Donut 을 제외하고 계단형이 많이 나왔는데 오류그래프를 보면 이전 실험보다 더 나빠졌음을 알 수 있다. 계단형에서 가로로 길수록 적절한 가중치를 찾는데 오래 걸렸기 때문이다. 또한 학습이 완료되기까지 반복 횟수가 이전보다 많이 늘었다

* + 1. hidden layer 8개: 모두 학습되지 않음

레이어당 노드 개수가 2개인 경우, 레이어 개수가 많을수록 오히려 학습이 제대로 되지 않았다.

오류가 감소하지 않은 것으로 보아 가중치 갱신이 거의 이루어지지 않은 것인데, 층이 깊을수록 모든 가중치가 0에 가깝다면 미분이 0에 수렴하기 때문에 가중치 갱신이 이루지지 않았다고 볼 수 있다. 가중치가 0에 가깝지 않더라도, XOR 의 오류 그래프처럼 계단보다는 언덕형이 나타나지만 미분이 소실되는 문제가 나타나 학습이 제대로 되지 않음을 알 수 있다.

* 1. Hidden Layer 의 개수에 따른 실험 2 (각 층당 노드 개수 3개)
     1. hidden layer 2개: 모두 학습 완료

노드 3개인 layer 1개인 실험과 노드 개수가 2개인 layer 2개에 대한 실험 보다 더 좋은 결과가 나왔다.

* + 1. hidden layer 3개: 모두 학습 완료

노드 3개인 layer 2개인 위 실험보다 더 나은 결과가 나왔다. 가중치를 맞추는데 (오류가 감소하지않아 그래프 상에 가로로 길게 나온 부분) 시간이 걸렸지만, 학습이 완료될 때의 횟수가 이전보다 적게 나오는 경우가 빈번한것으로 보아 적절한 값으로 맞추고나면 학습이 빠르게 이루어짐을 알 수 있다.

* + 1. hidden layer 4개: 모두 학습 완료

노드 3개인 layer 3개인 위 실험보다는 좋지 못한 결과가 나왔다. 학습 완료에 걸린 반복 횟수와 오류 그래프의 모양을 봤을 때 층이 깊어서 학습에 시간이 많이 소요되었음을 알 수 있다.

* + 1. hidden layer 8개: 모두 학습되지 않음

모든 실험에서 학습이 되지 않았다.

* 1. Hidden Layer 의 노드 개수가 다른 경우
     1. hidden layer 의 각 노드가 4,3,2 개: 모두 학습 완료

초기 가중치에 따라 결과가 많이 바뀌었다. 마지막 hidden layer 에서의 가중치가 0에 가까울수록 학습이 되지 않는 경우가 많았다.

* 1. 2차원 직선 그래프
     1. 재생을 해보면, 첫번째 레이어가 분주하게 움직이는 반면, 다음 레이어에서는 천천히 움직이는데 이전 레이어에서 입력값을 선형분리하도록 옮김에 따라 다음 레이어의 직선이 결정되기 때문이다. 예를 들어, 기울기가 5감소했어도 다음 레이어에서는 기울기가 5가 아닌 1만큼 감소하는 것이다.
     2. output layer 를 보면, (0,1)의 점이 두번째 hidden layer 에서 아래로 옮겨지면서 마지막에 xor 값들로 분리된 것을 볼 수 있다. 즉, 이 경우 (0, 1) 은 (1, 0)으로 옮겨진 것이다.
     3. 그리고 마지막 hidden layer 의 2개의 직선의 모습과 output layer 의 직선의 모습을 비교해보면 항상 동일한 방향을 갖는 것을 알 수 있다.