

**과제 #2 Perceptron Learning 구현**

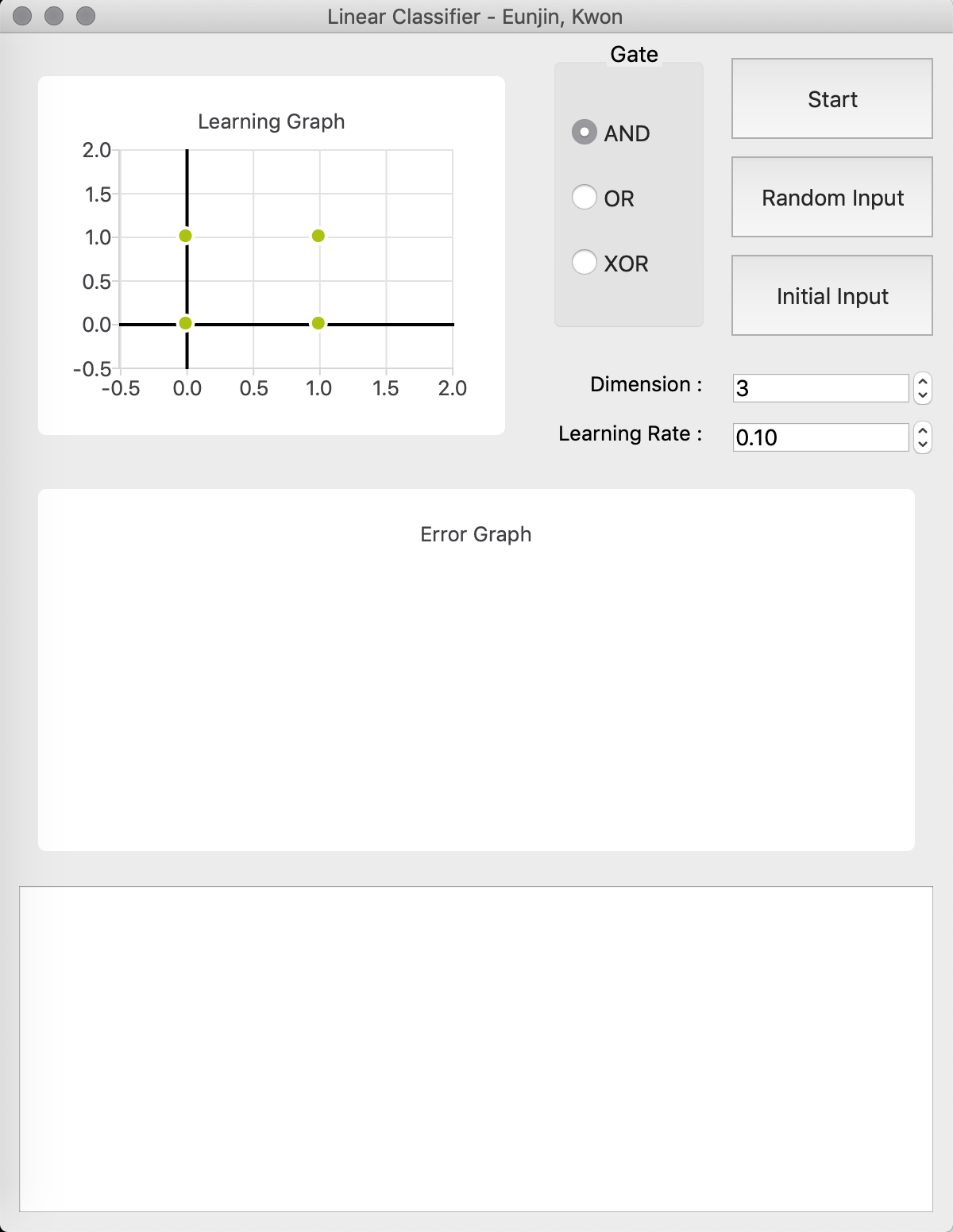
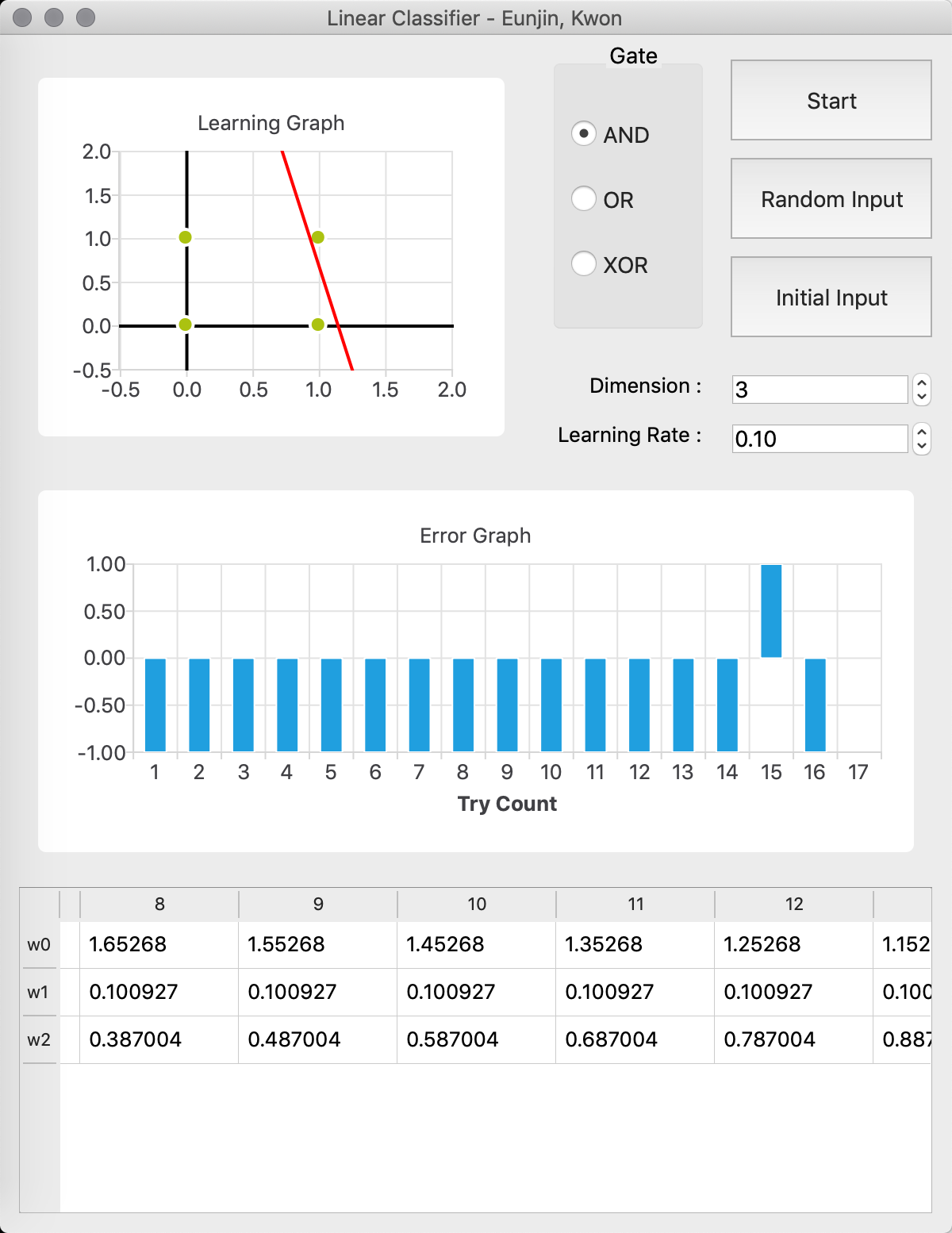
컴퓨터과학부

2016920004

권은진

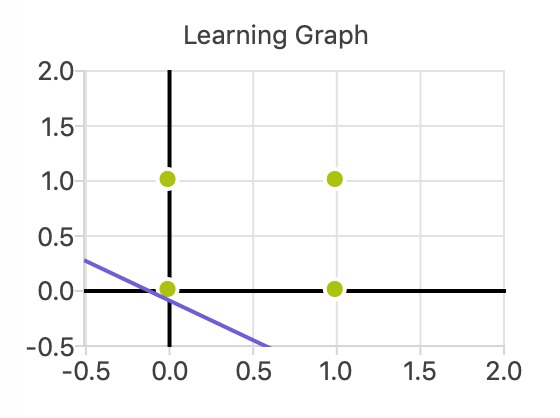
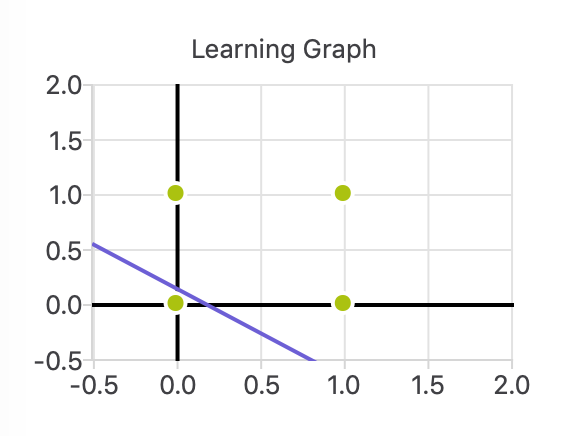
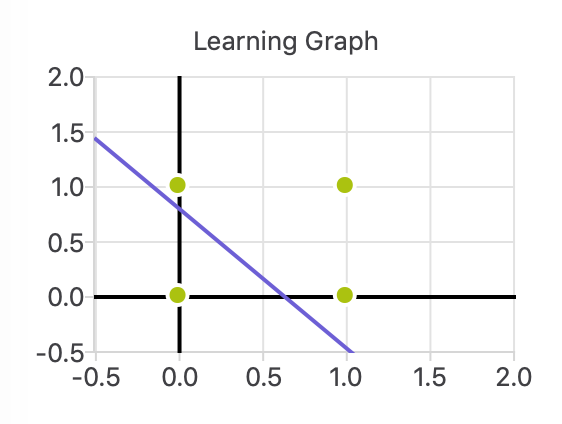
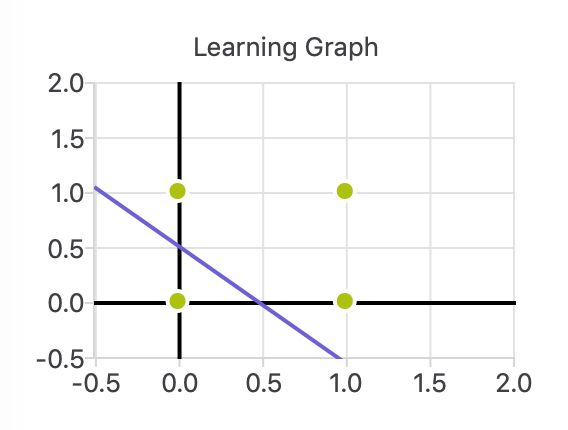
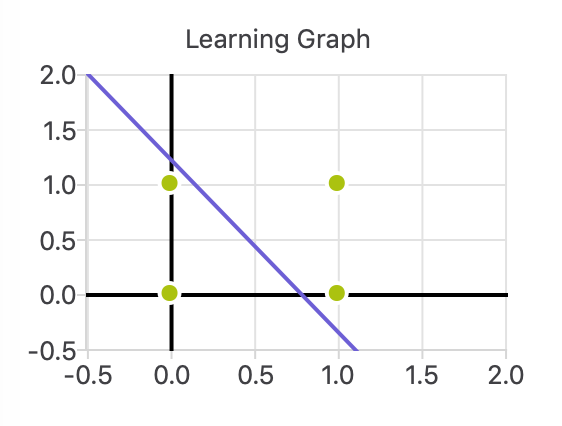
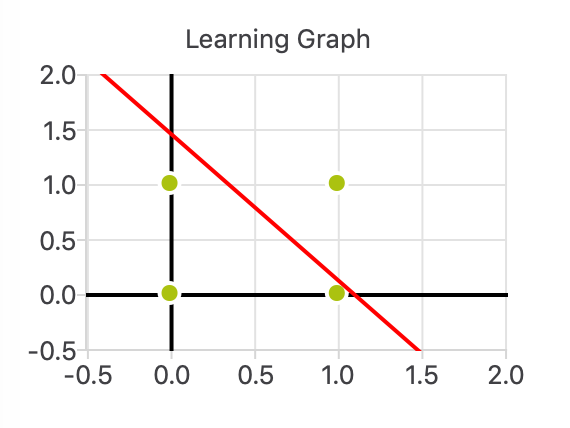
1. 프로그램 개요
   1. 언어: C++
   2. 사용 라이브러리: Qt 5.13 \* 반드시 설치 후 실행 가능 \*
   3. 실행 환경: Cross Platform (Windows, Linux, Mac OS)
   4. 소스코드: <https://github.com/ke2ek/AI_assignment>
   5. 클래스 구조
      1. MainWindow: GUI 용 클래스

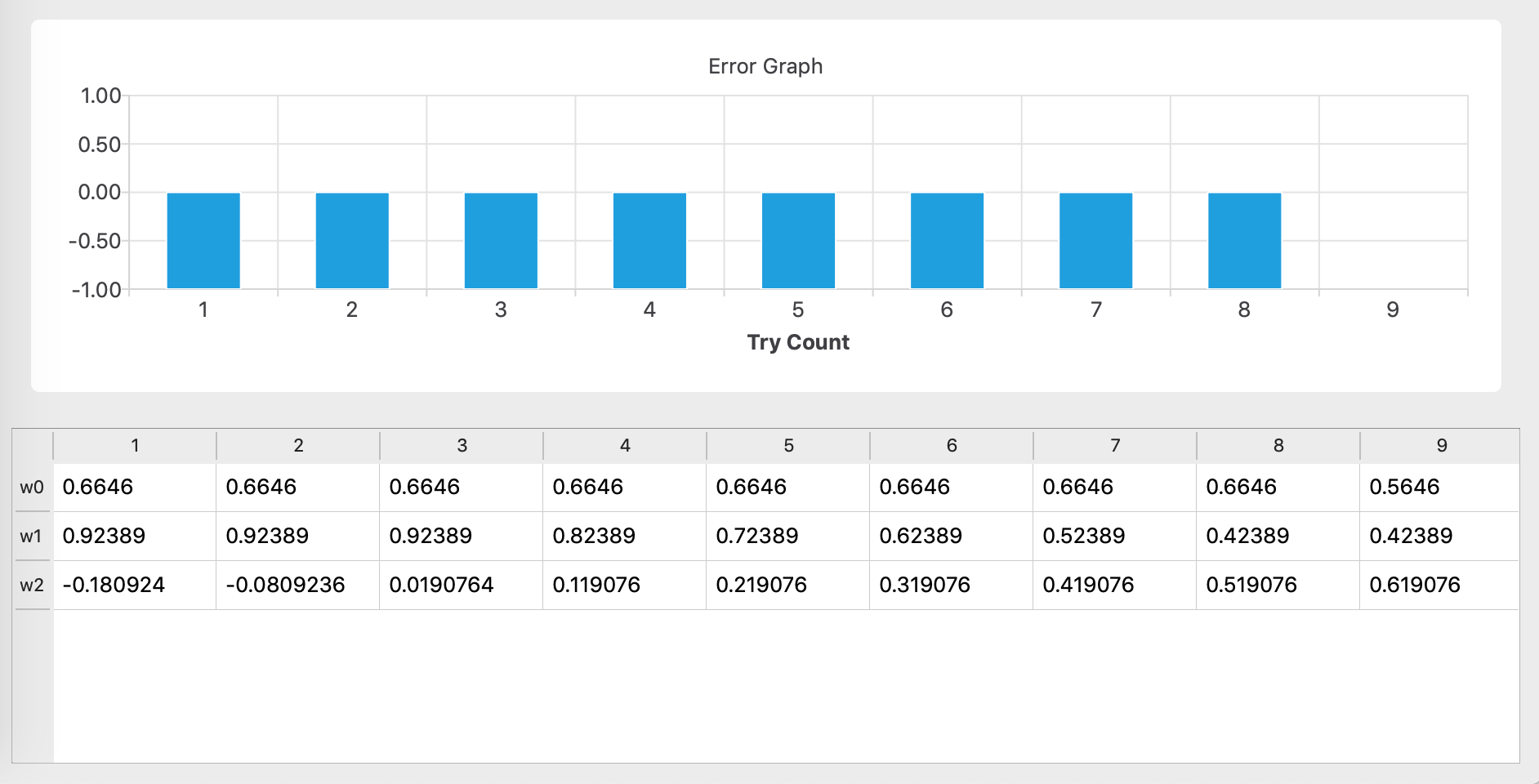
* … (생략)
* **draw\_line()** - 분류 중 가중치 변화마다 보내지는 시그널을 처리하는 함수로, 가중치와 x 좌표들을 이용하여 그래프에 직선을 그림
* **draw\_error()** - 분류가 끝나면 보내지는 시그널을 처리하는 함수로, 모든 오류와 가중치 값들을 그래프 및 표에 표시
  + 1. LinearClassifier: 선형 분류를 하는 클래스
* … (생략)
* **ready()** - 분류 전, 멤버 변수들을 초기화
* **calc()** - 분류과정에서 output 을 계산
* **run()** - thread 함수로, 실제 분류를 수행
* **changed\_weight()** - 매 가중치 변화마다 MainWindow 로 보내지는 시그널(signal)
* **done()** - 분류가 끝나면 MainWindow 로 보내지는 시그널(signal)
  1. 인터페이스
     1. Button
* **Start** - 분류 시작 버튼
* **Random Input** - 랜덤 입력값 설정 버튼, x0와 x1은 고정적인 값을 가지며, 설정한 차원에 따라 x2 부터 랜덤값 저장
* **Initial Input** - 초기값 설정 버튼, 최초의 값은 x0와 x1은 고정적인 값이고 그 외 x2 부터 -1을 저장
  + 1. SpinBox
* **Dimension** - 입력의 차원 설정 (0~1000)
* **Learning Rate** - 학습 스텝 크기 설정 (0.0 ~ 1.0)
  + 1. Graph
* **Learning Graph** - 학습 과정 중 직선의 움직임을 표현하는 그래프
  + 좌표축: x0 (가로), x1 (세로)
* **Error Graph** - 학습 종료 시 각 시도마다 산출된 오류를 보여주는 그래프
  + 그래프 x 축 범위를 드래그할 경우, 확대 및 축소 가능
    1. Table
* **Weight Table** - 학습 종료 시 각 시도마다 계산된 가중치를 보여주는 표

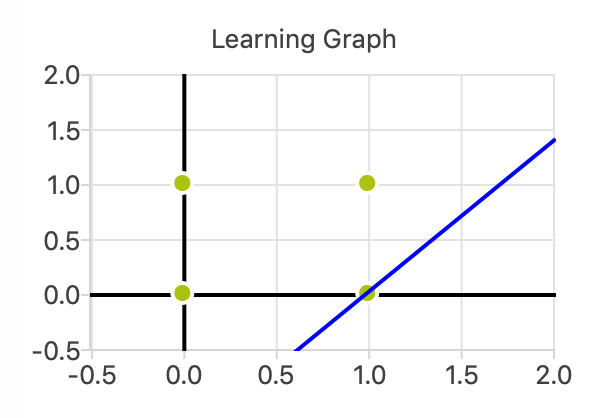
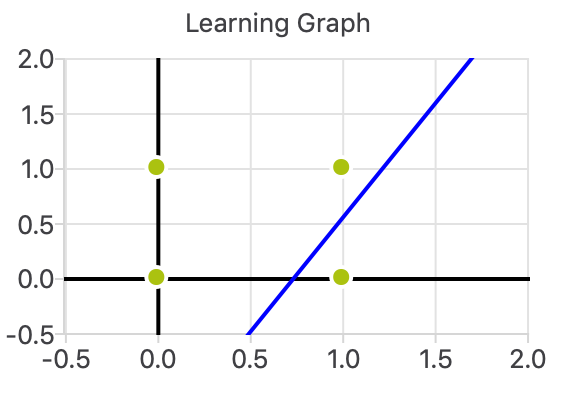
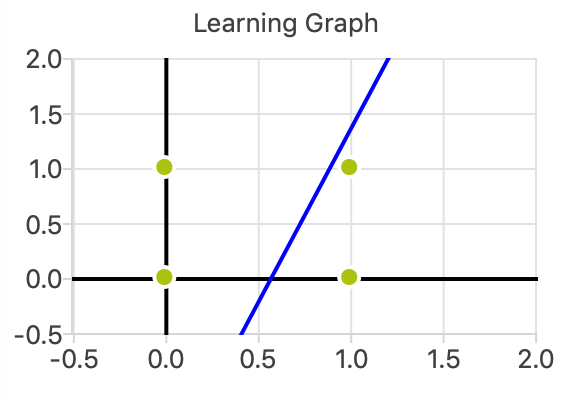
[ 좌 - 학습 전, 우 - 학습 후 ]

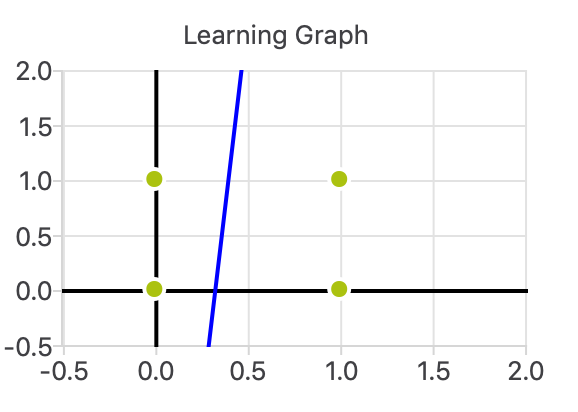
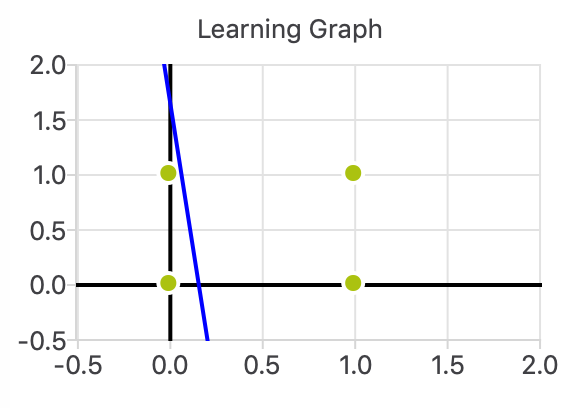
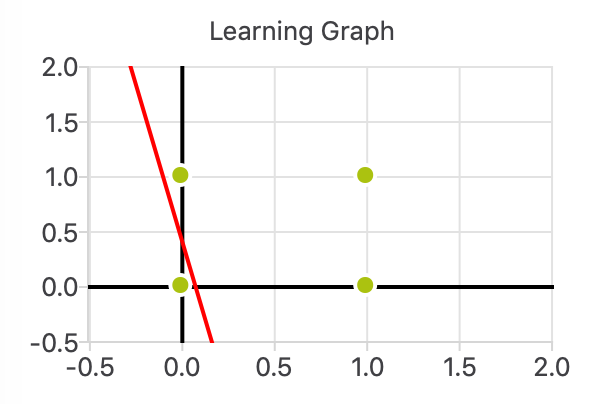
1. 실행 결과
   1. 차원 = 3, 추가 입력값 = -1, Learning Rate = 0.1
      1. [AND Gate]

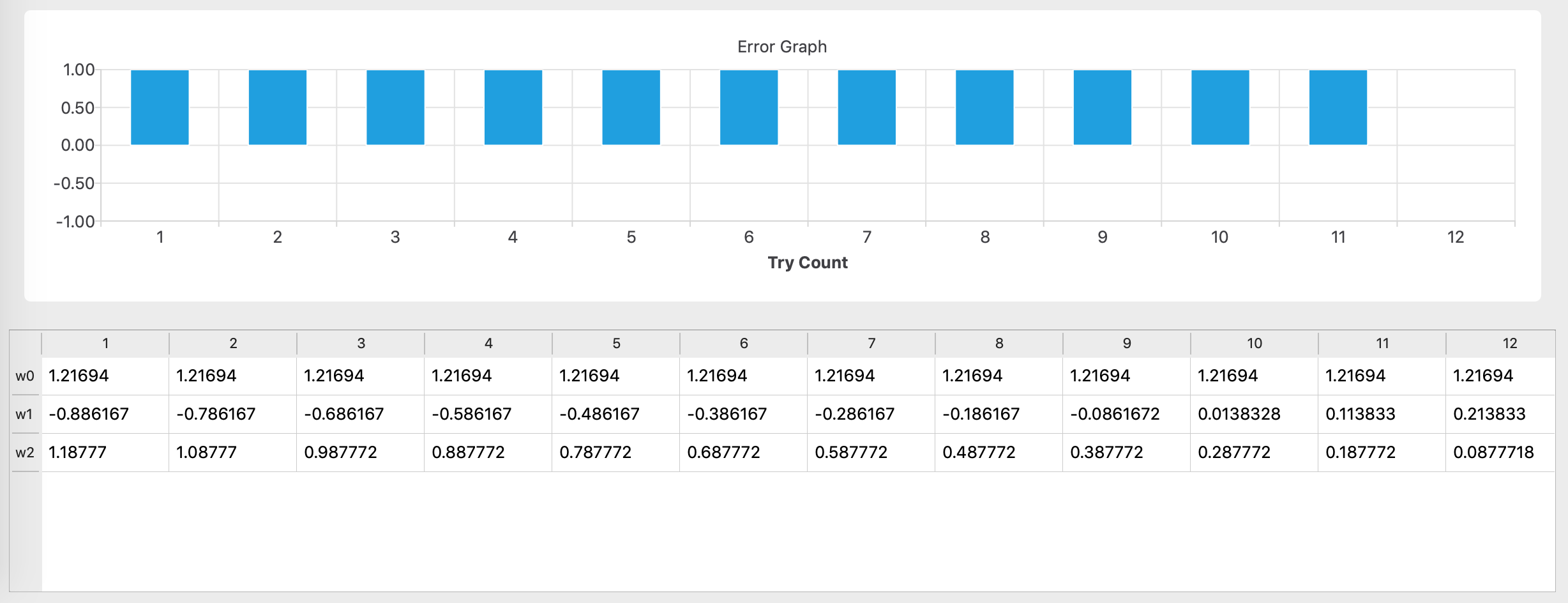
 …  …  … ... 



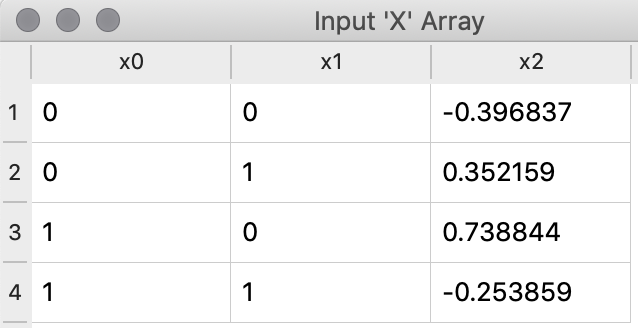
* + 1. [OR Gate]

 …  … 

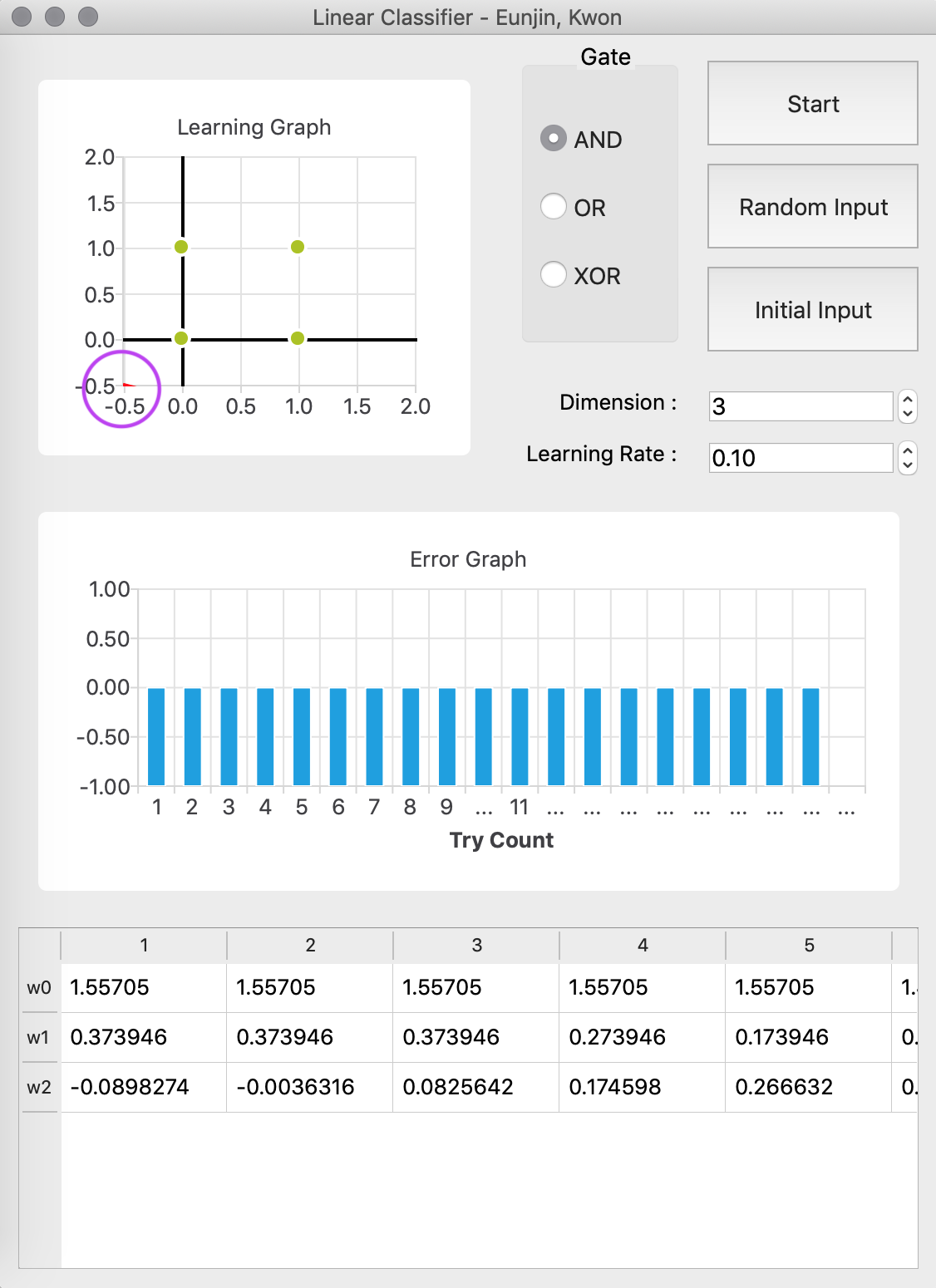
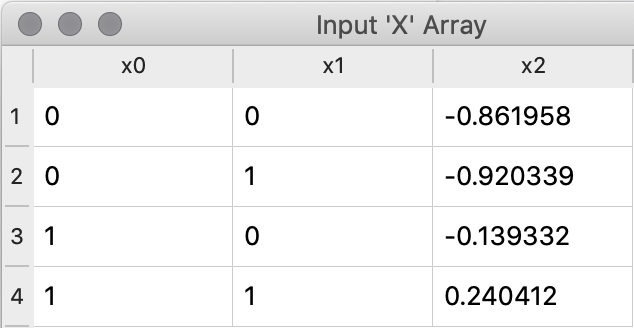
 …  … 



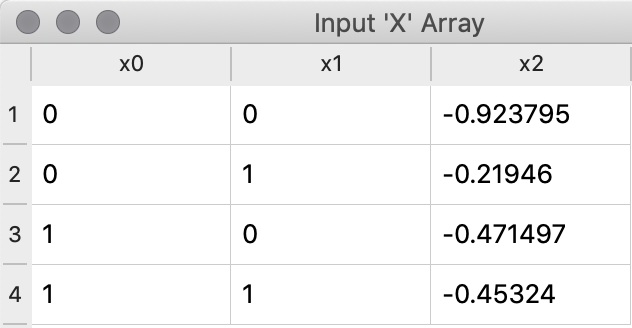
* + 1. [XOR Gate]
* Infinite Loop
  1. 차원 = 3, 랜덤 입력값, Learning Rate = 0.1
     1. [AND Gate]
* 직선을 못찾거나 (Infinite Loop),



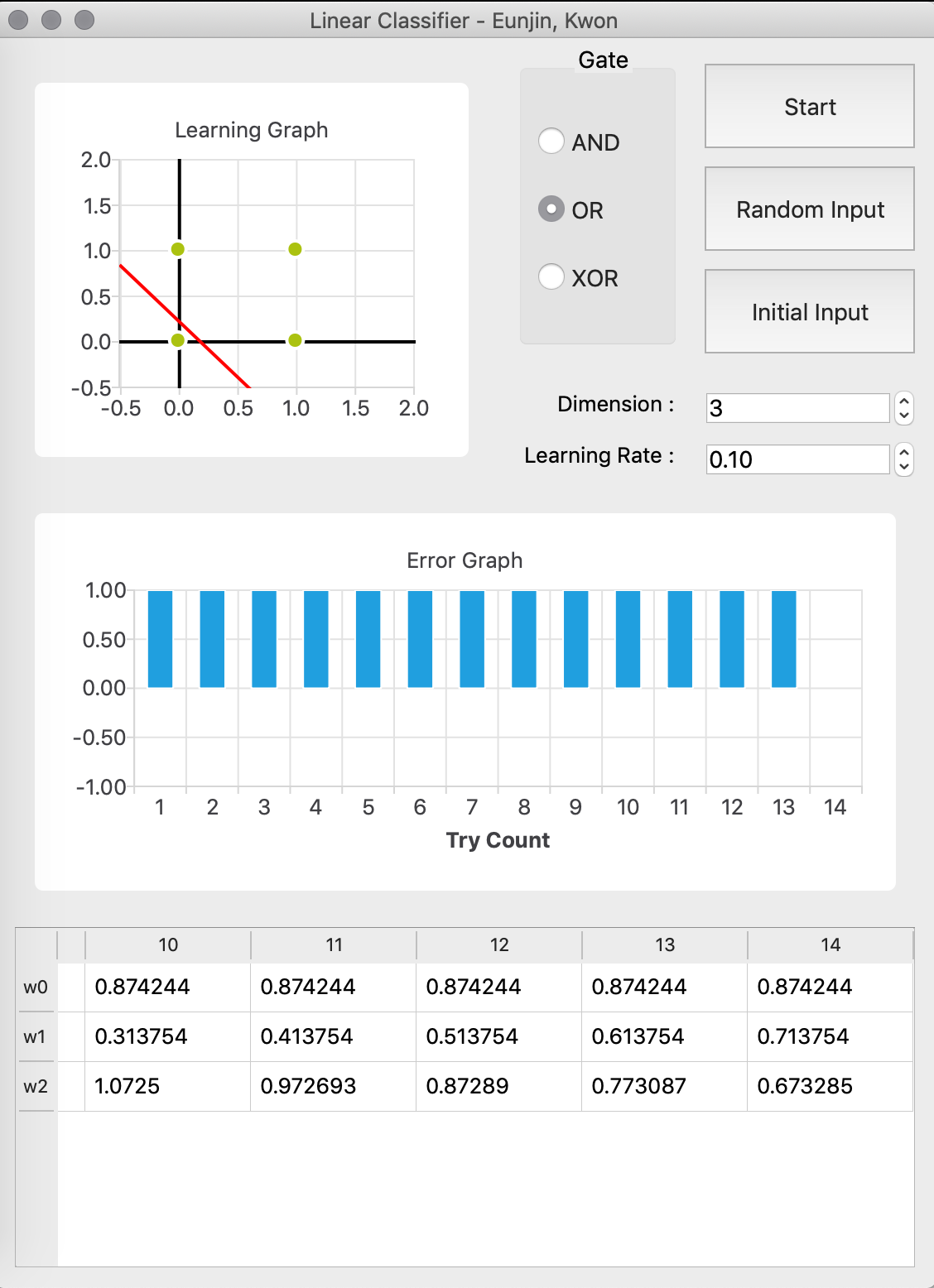
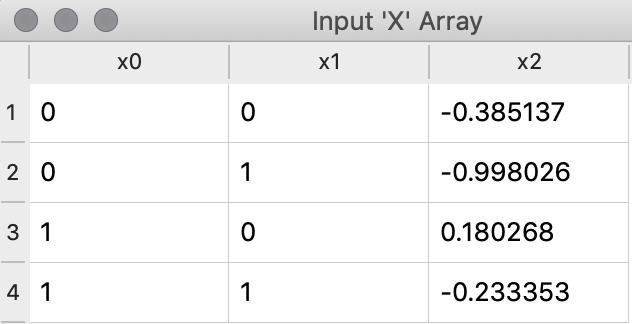
* 잘못된 직선을 찾거나 (정답과 일치하지 않음),



* 입력값에 따라 직선을 찾기도 하였다. (모든 값이 음수인 경우)



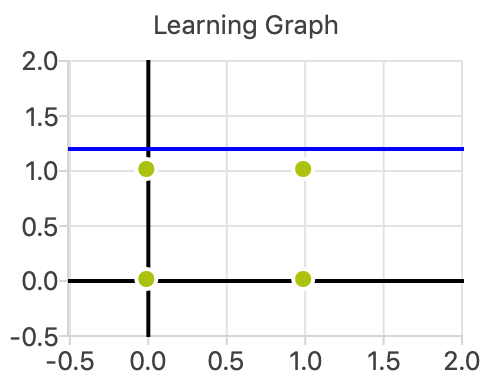
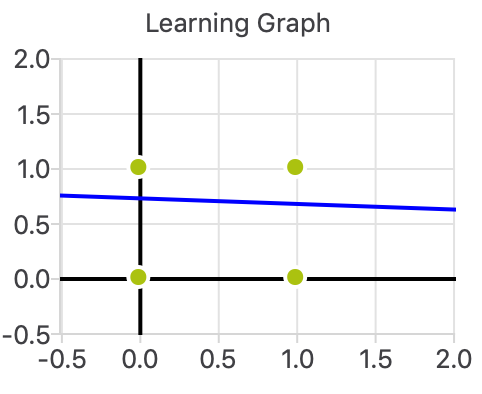
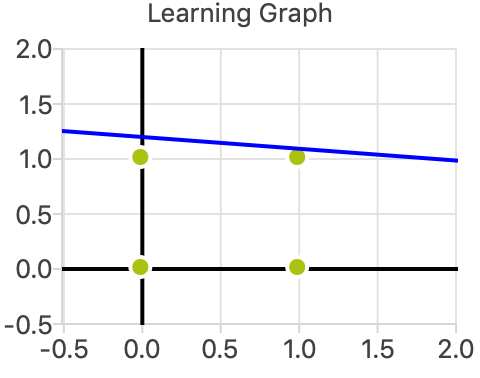
* + 1. [OR Gate] - 목차 b. (i) 와 동일한 결과
* 직선을 찾은 경우는 입력값에 차이가 있었다.

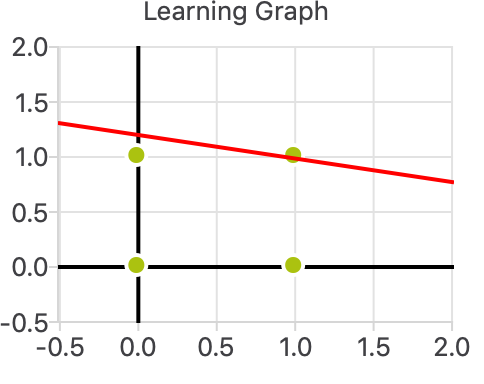


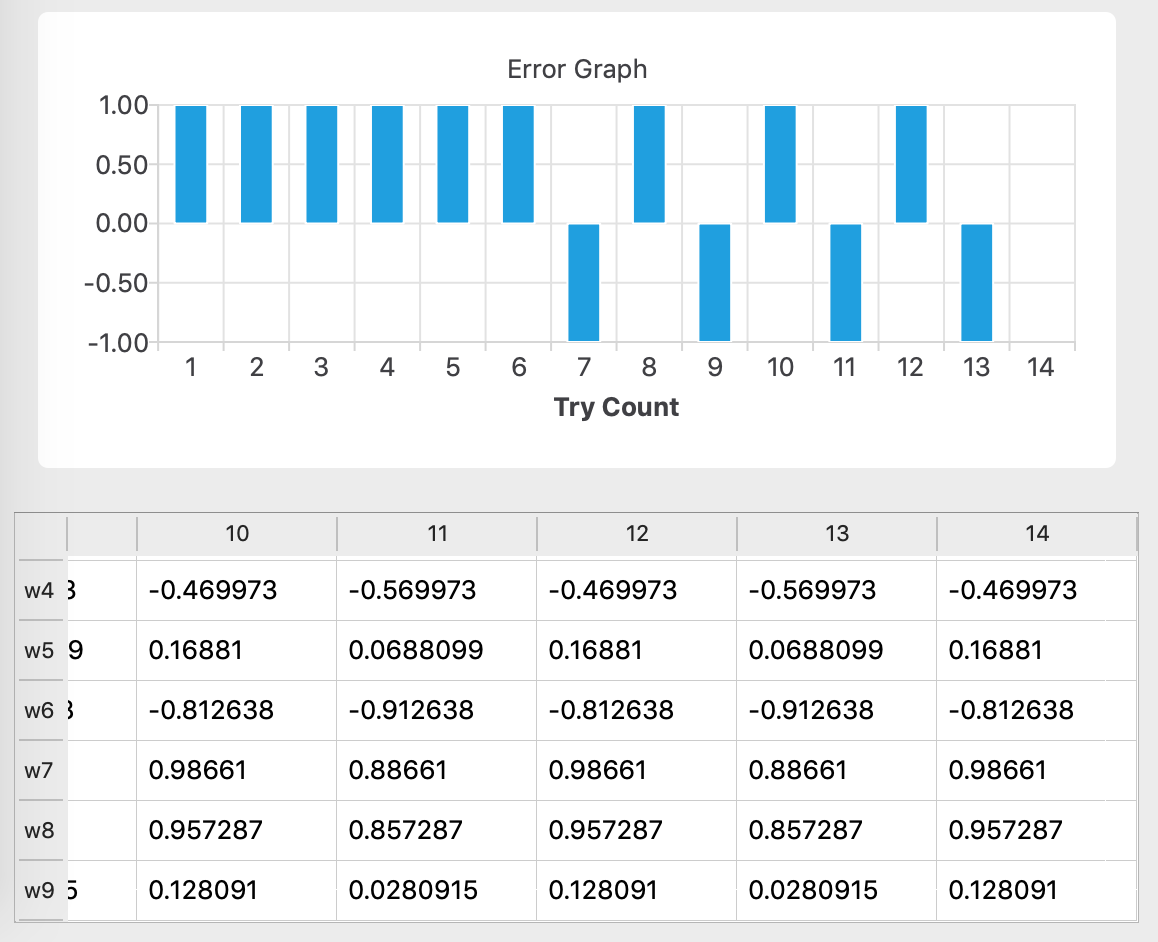
* + 1. [XOR Gate]
* Infinite Loop
  1. 차원 > 3, 추가 입력값 = -1, Learning Rate = 0.1
     1. 차원 = 10



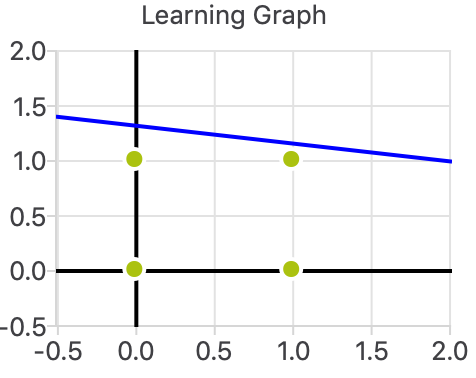
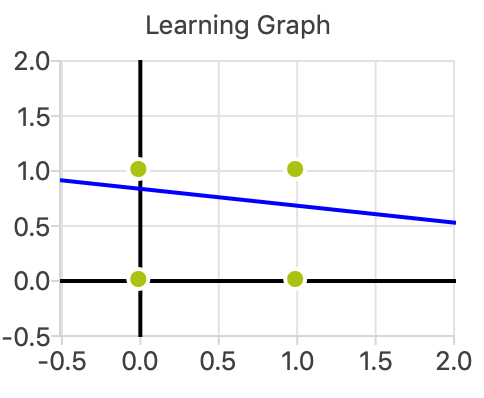
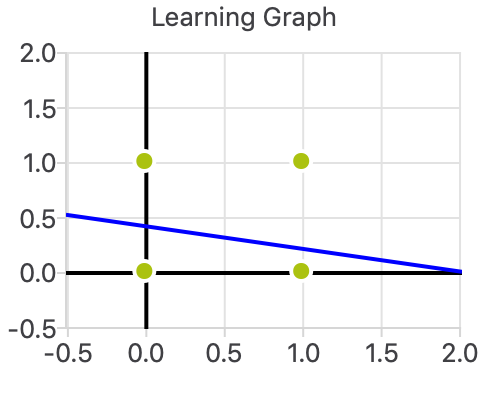
* AND Gate

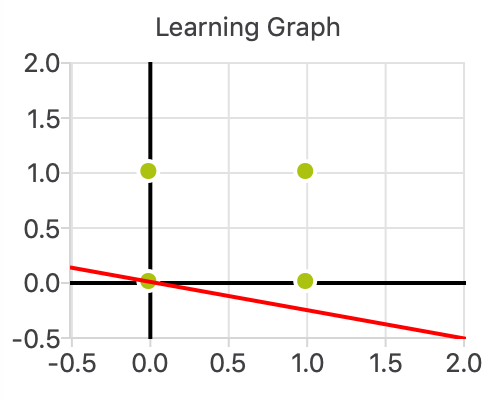
 …  … 

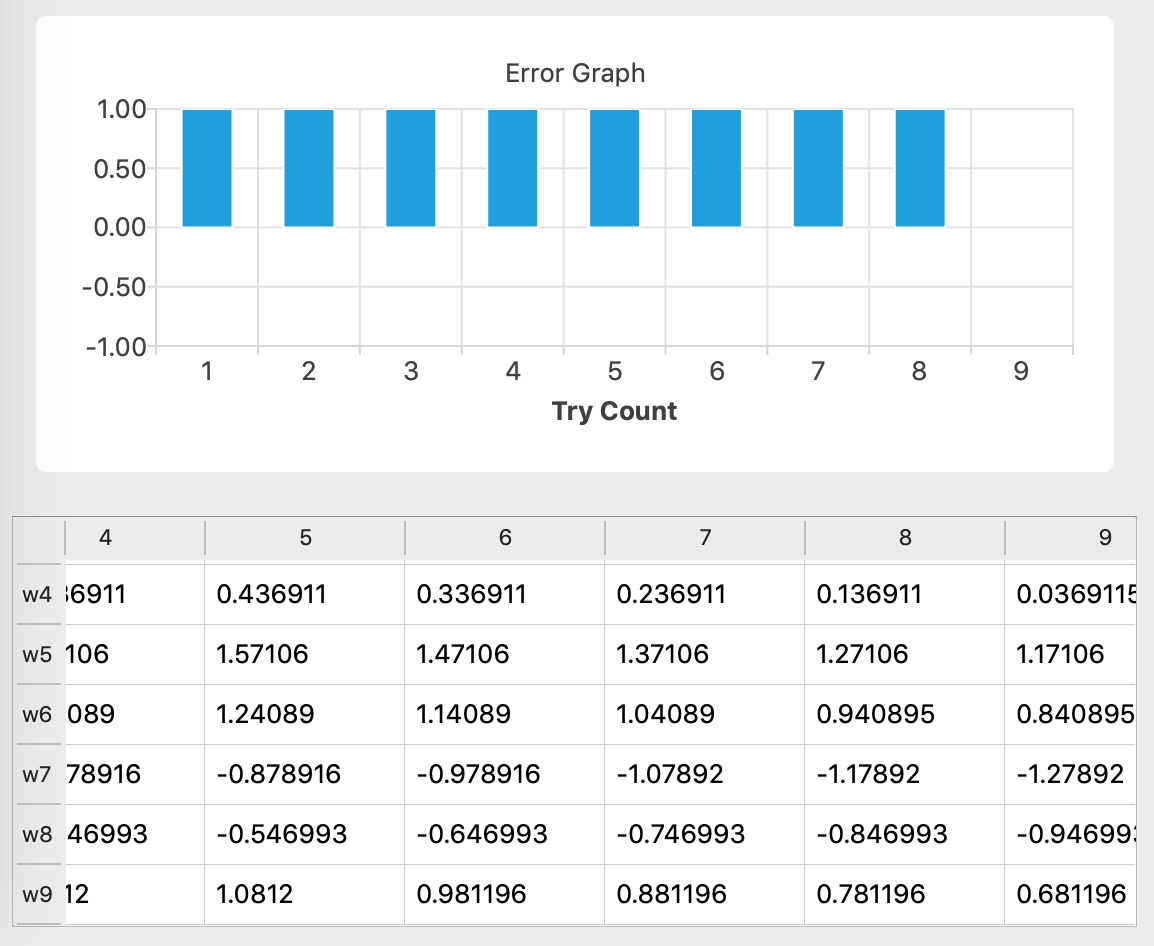
… 



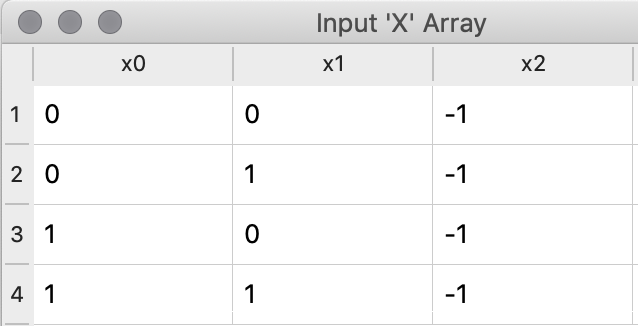
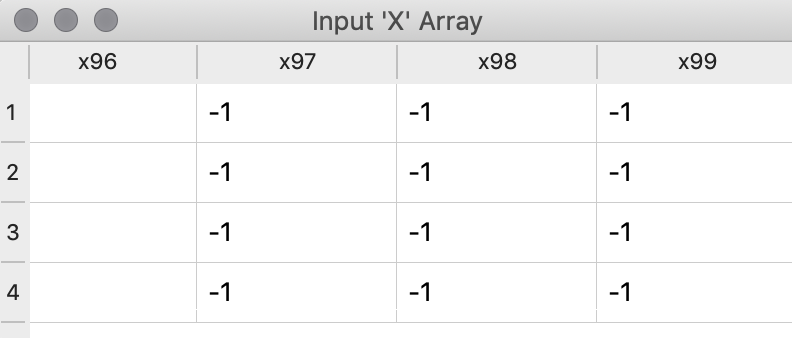
* OR Gate

 …  … 

… 



* XOR Gate - Infinite Loop
  + 1. 차원 = 100

 … 

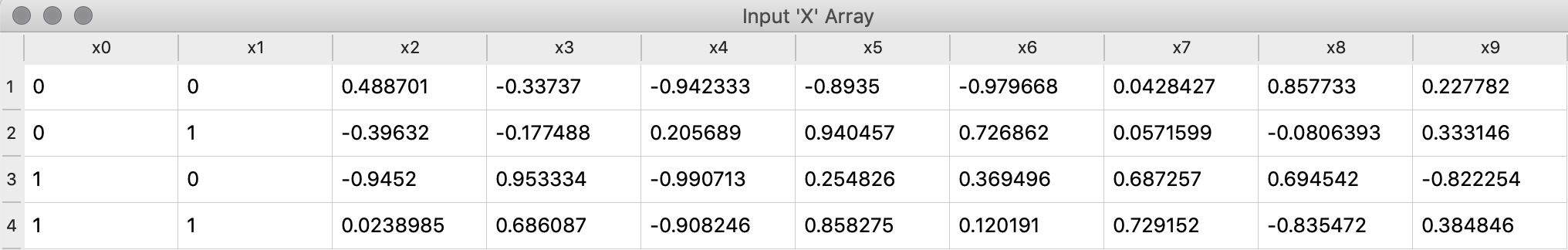
* AND Gate



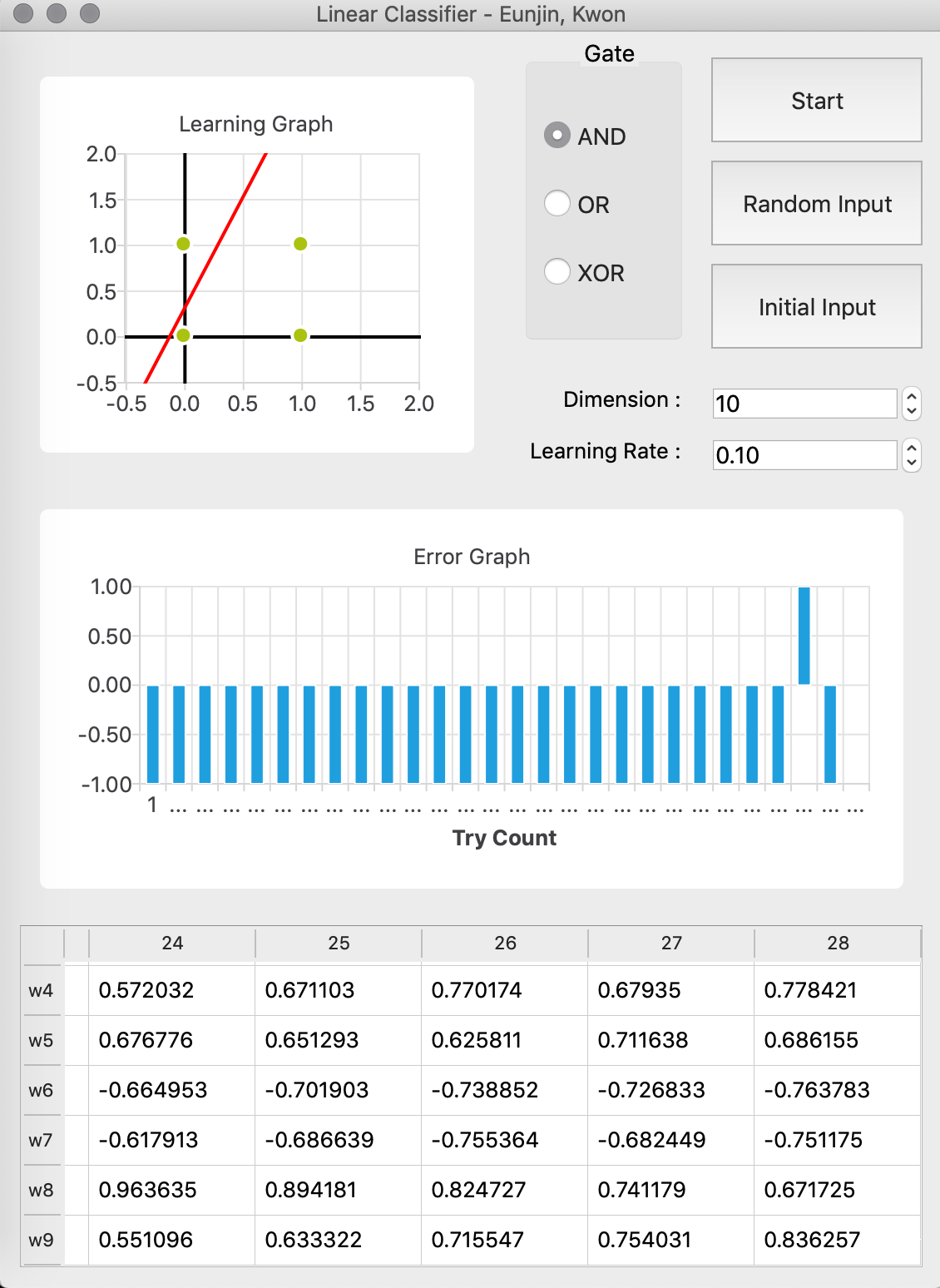
* OR Gate



* XOR Gate - Infinite Loop
  1. 차원 > 3, 랜덤 입력값, Learning Rate = 0.1
     1. 차원 = 10

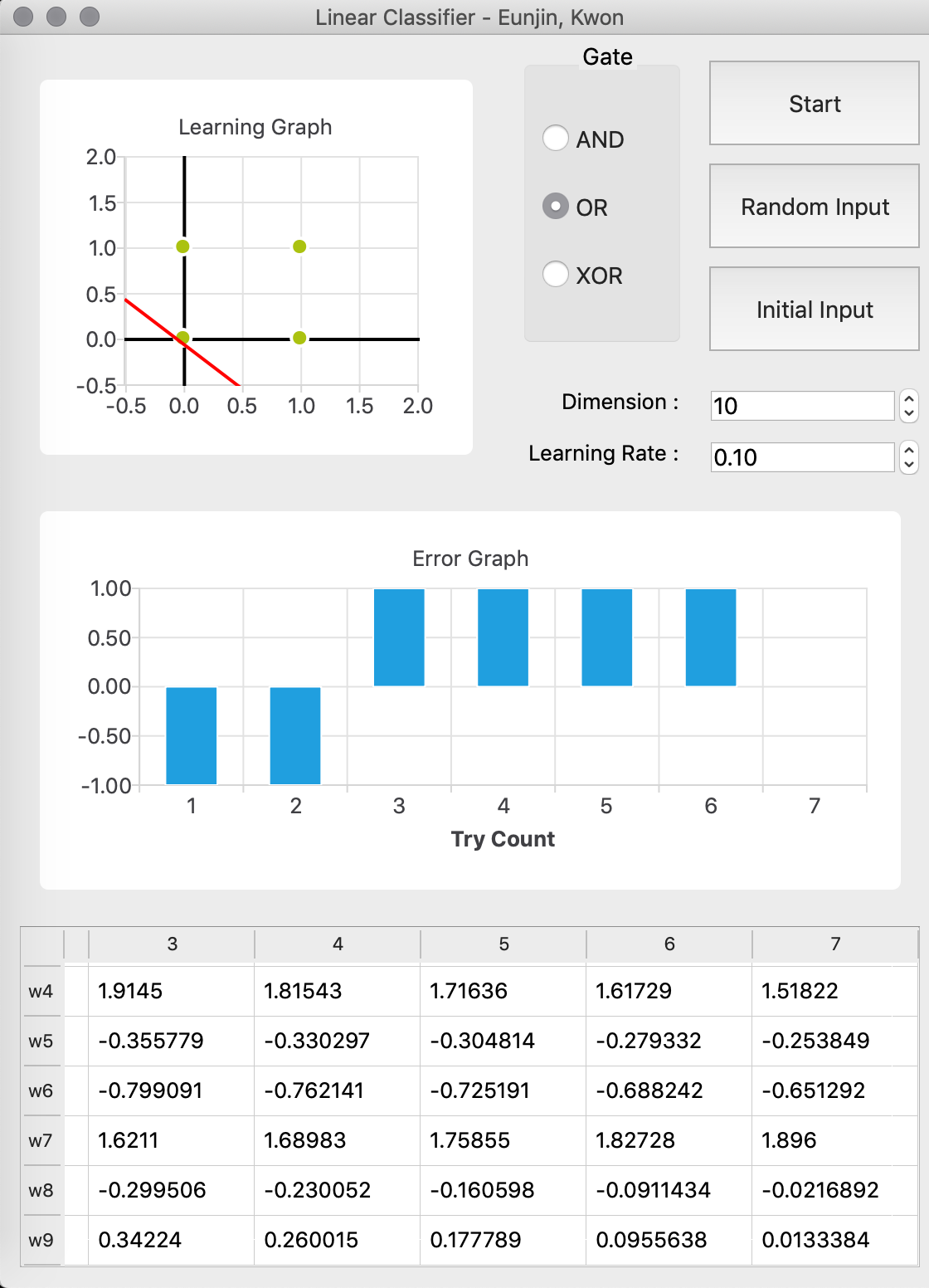


* AND Gate - 목차 b. (i) 와 동일한 결과

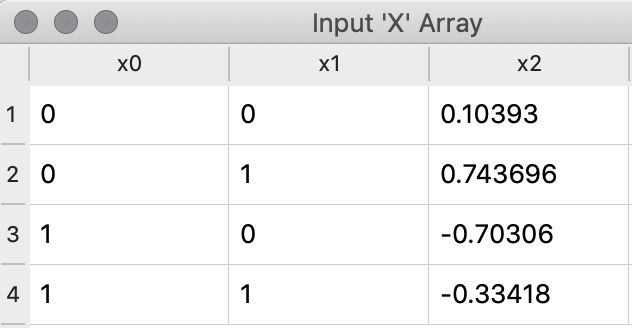
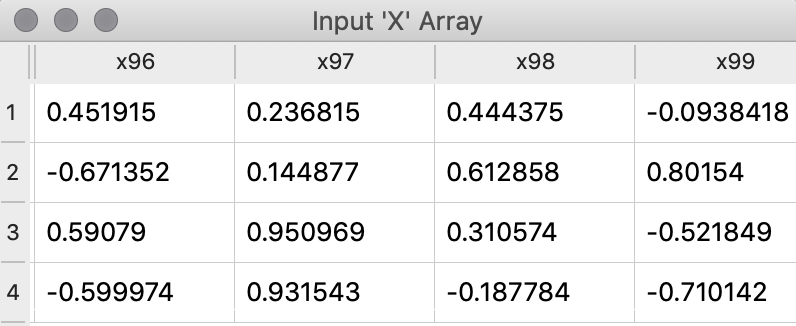
[ 잘못된 직선을 찾은 경우의 예시 ]

* OR Gate - 목차 b. (i) 와 동일한 결과

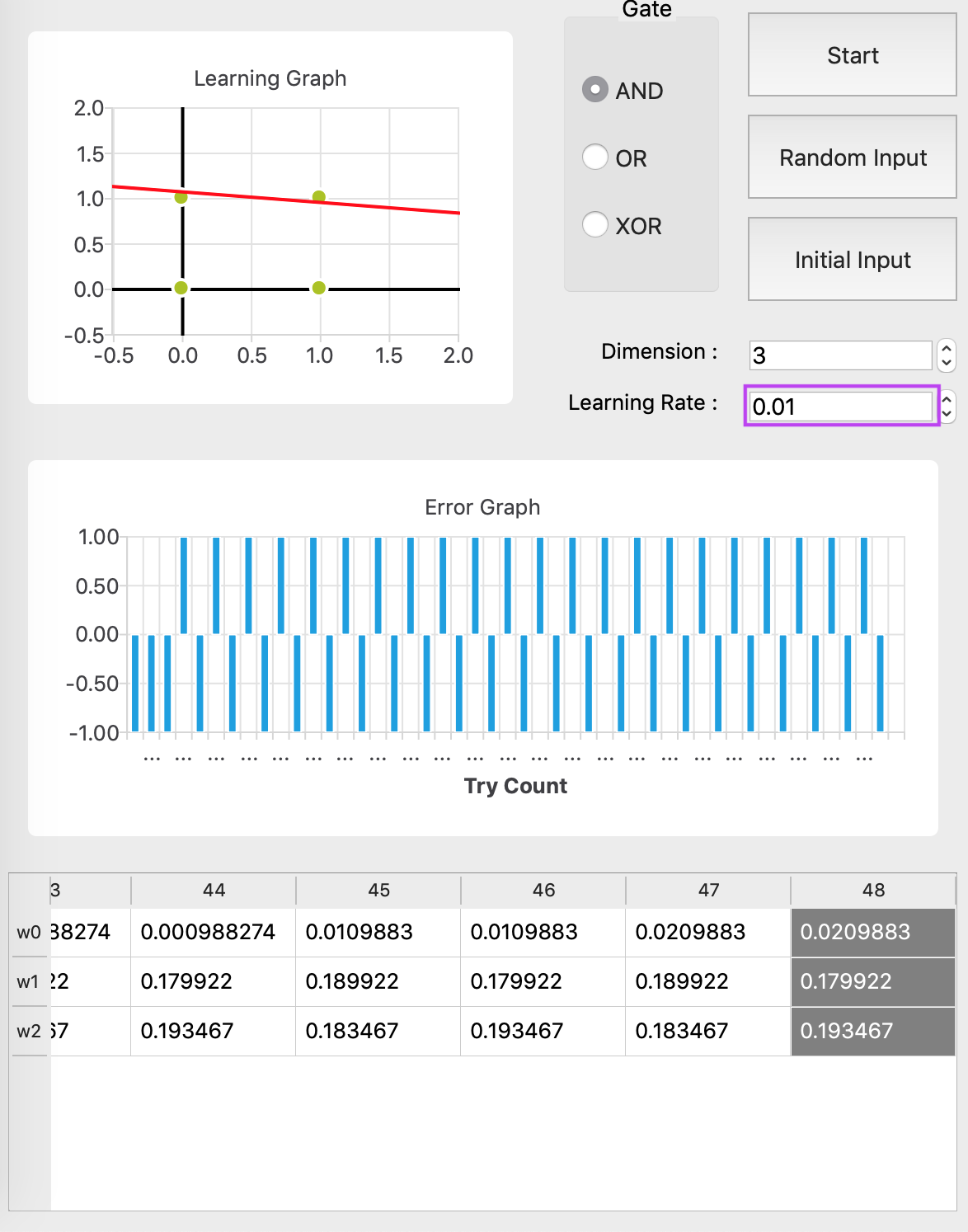
 

[ 잘못된 직선을 찾은 경우의 예시 ]

* XOR Gate - Infinite Loop
  + 1. 차원 = 100

 ... 

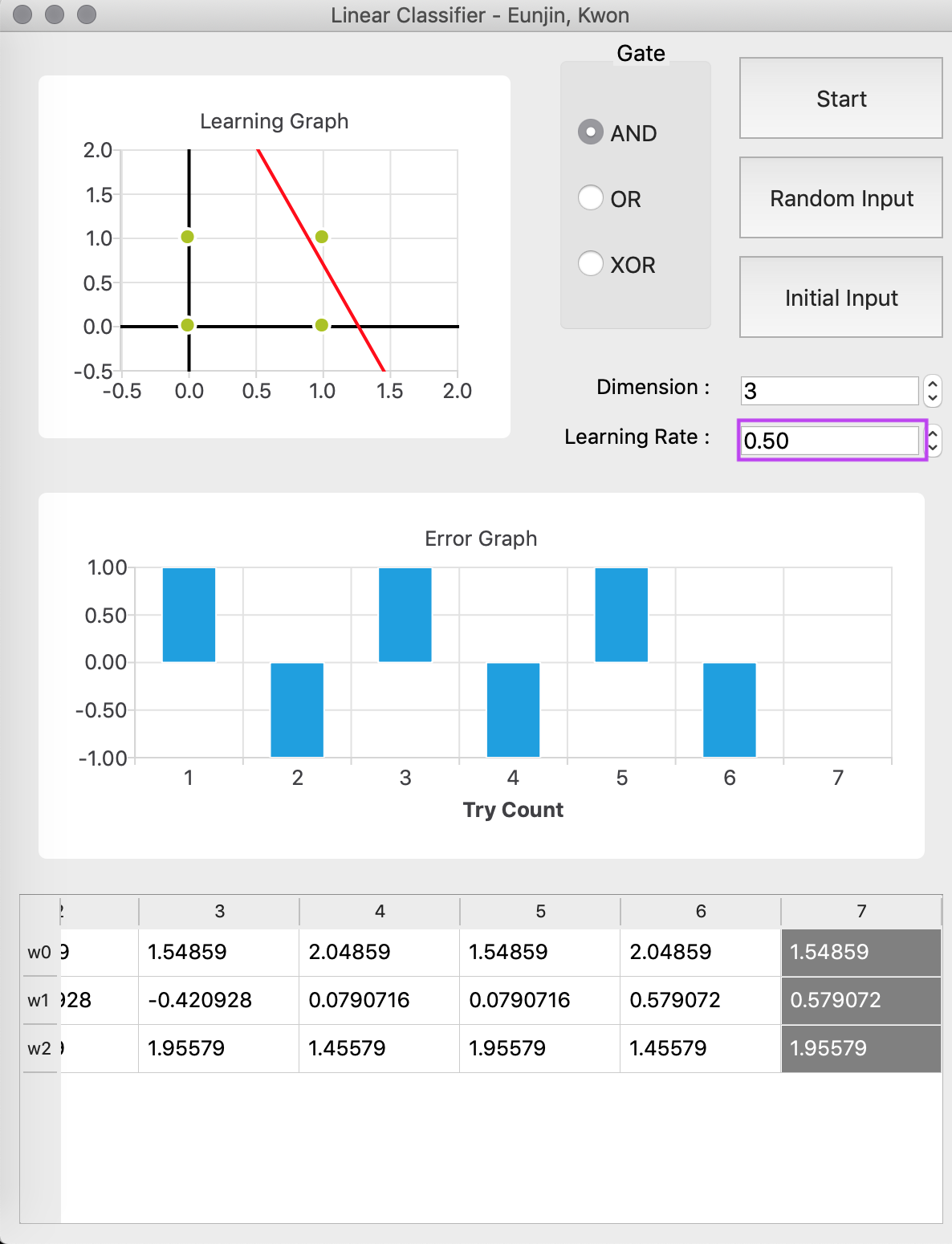
* AND Gate - 항상 잘못된 직선을 찾음
* OR Gate - 항상 잘못된 직선을 찾음
* XOR Gate - Infinite Loop
  1. learning rate 에 따른 변화
     1. c = 0.01



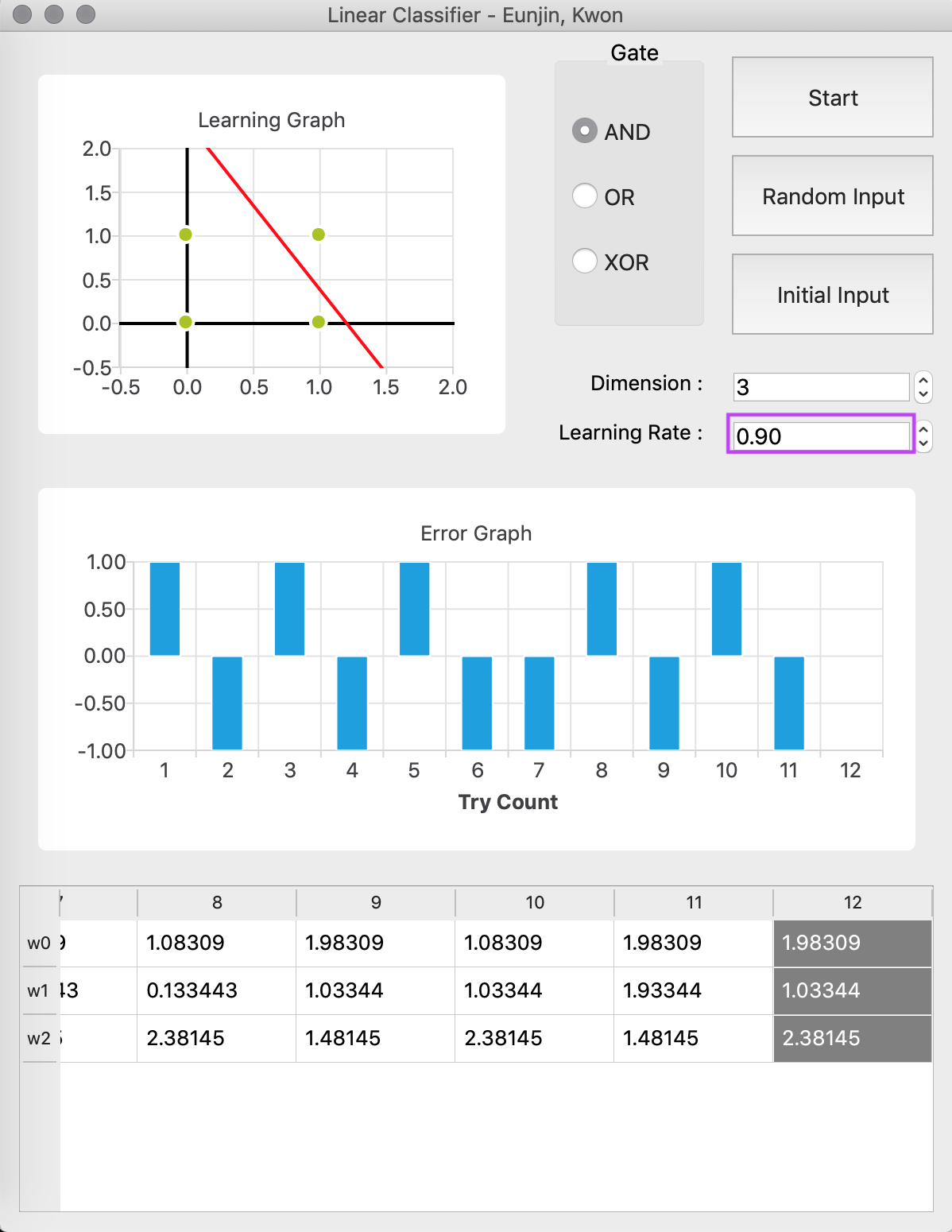
* + 1. c = 0.1



* + 1. c = 0.5



* + 1. c = 0.9



1. 결론
   1. x2 부터 입력값이 -1 인 경우 직선을 빠르게 찾았다. 단, 입력 차원의 크기가 커질수록 x0 와 x1 에 대한 직선에서 x0 절편값이 커지게 되어 시간이 좀 더 걸리는 것을 확인하였다.
   2. x2 부터 입력값이 랜덤인 경우 대부분 직선을 못찾았다. 그러나 AND Gate 에서 입력값이 모두 음수일 때 시도횟수가 많았으나 직선을 찾을 수 있었고, OR Gate 에서는 x2의 입력값 중 하나만 양수여도 직선을 찾을 수 있었다.
   3. learning rate 이 작은 경우 (0.01 이하) 직선의 움직임이 느린 결과가 나왔다. 이는 학습 과정에서 가중치의 step size 가 작기 때문에 나온 현상으로 보이며, 시도횟수가 상대적으로 많은 것으로 확인되었다.
   4. learning rate 은 0.1 ~ 0.5 사이의 값이 적절하다고 생각한다. 0.01 이하인 경우 위에 언급했듯이 가중치 변화가 너무 작아 시간이 오래 걸리며 0.5 를 넘어서는 경우 step size 가 커지기 때문에 직선의 움직임이 커져 오히려 그래프 상에서 제대로 된 직선을 찾는데 방해가 되었다.
   5. 추가 입력값이 -1일 때, error graph 에서는 정답 직선의 왼쪽 방향에서 시작하는 경우 지속적으로 오른쪽으로 움직여야 하기 때문에 error 값이 +1 이 나왔다. 반대로 정답 직선의 오른쪽 방향에서 시작하는 경우 왼쪽으로 움직여야 하기 때문에 error 값이 -1 이 계속 나왔다.
   6. 반면 추가 입력값이 랜덤인 경우 또는 learning rate 이 0.5 보다 큰 경우 직선의 움직임이 크기 때문에 error 값은 1과 -1을 교차하며 나오는게 대부분이었다.
   7. xor gate 의 경우 어떤 직선도 찾지못하고 계속해서 무한루프를 돌았다. 하나의 직선으로 분리할 수 없기 때문에 이런 결과가 나온 것 같다.