

Chapter 02

인공지능의 딥러닝 알고리즘

01 PyCharm 개발 환경 구성하기

여기서는 파이썬 개발을 좀 더 효율적으로 할 수 있는 파이참을 설치하도록 합니다. 파이참을 이용하면 아래와 같이 intellisense 기능 등을 통해 모듈이나 클래스 내의 함수를 사용할 때 편리하게 개발이 가능합니다. 또 문법에 대한 오류 체크도 자동으로 해 줍니다.



*** 독자 여러분의 선택에 따라 파이참을 설치하지 않고 구글 코랩을 그대로 사용하여 이후의 실습을 진행해도 같은 결과를 얻을 수 있습니다.

01 파이참 설치하기

먼저 파이참 프로그램을 설치합니다.

1. 다음과 같이 검색합니다.



pycharm edu download

2. 다음 사이트로 들어갑니다.

<https://www.jetbrains.com> > ko-kr > pycharm-edu

[PyCharm Edu Python으로 프로그래밍을 배우고 ... - JetBrains](#)

3. 다음 페이지가 열립니다. [무료 다운로드] 버튼을 누릅니다.



4. 다음 페이지가 열립니다. [다운로드] 버튼을 누릅니다.

PyCharm Edu은 프리-오픈 소스이며

Apache 라이선스 버전 2.0에 따라 라이선스가 제공됩니다.

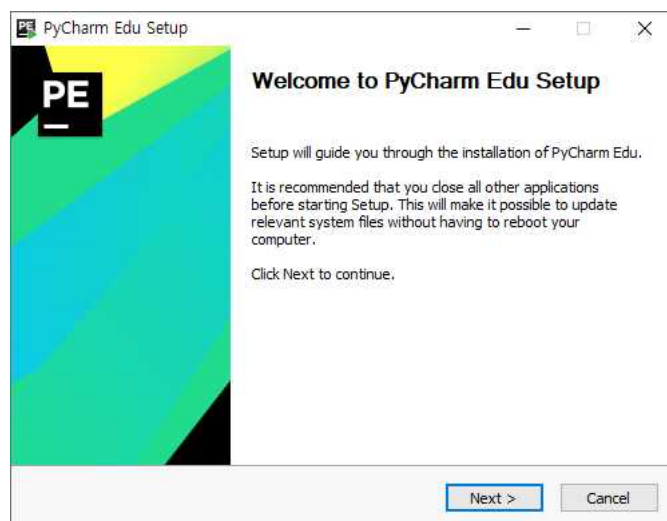


*** 이 책에서는 Windows 64비트 용 프로그램을 사용하여 실습을 진행합니다.

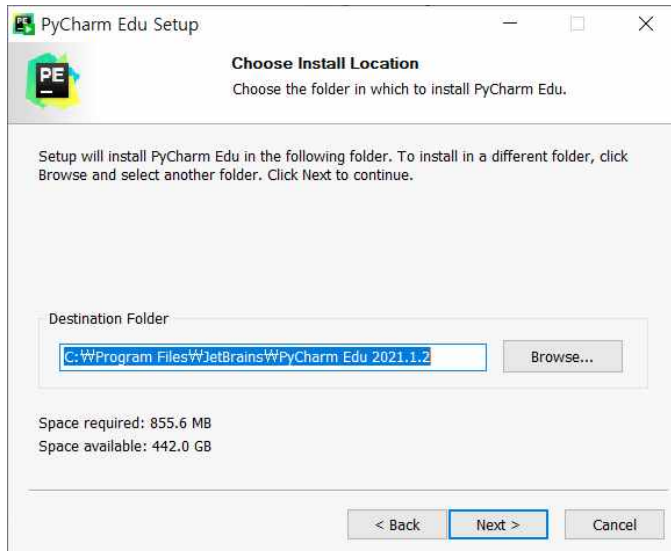
5. 다운로드가 완료되면 프로그램을 실행시켜 설치합니다.



6. 다음과 같이 설치 시작 창이 뜹니다. [Next>] 버튼을 눌러 설치를 진행합니다.

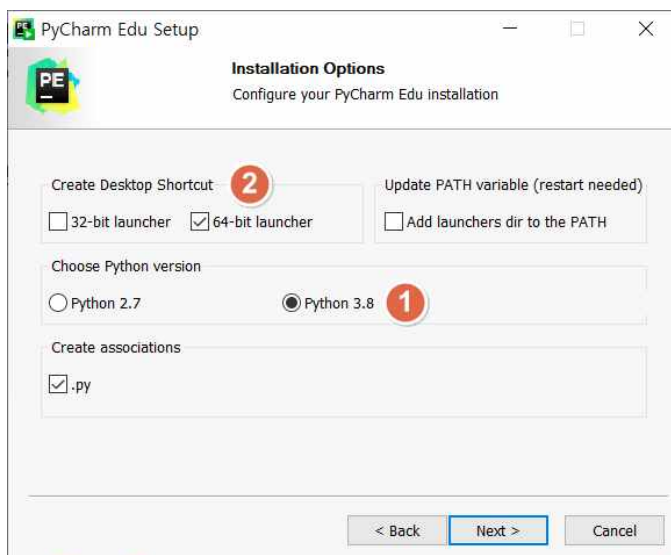


7. [설치 위치 선택] 창입니다. 기본 상태에서 [Next>] 버튼을 누릅니다.



*** 독자 여러분의 필요에 따라 다른 디렉터리에 설치할 수 있습니다.

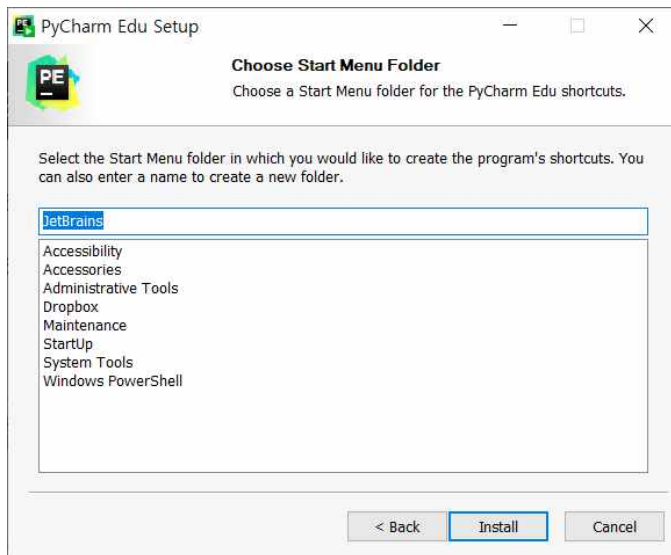
8. [설치 옵션] 창입니다. 다음과 같이 선택한 후, [Next>] 버튼을 누릅니다.



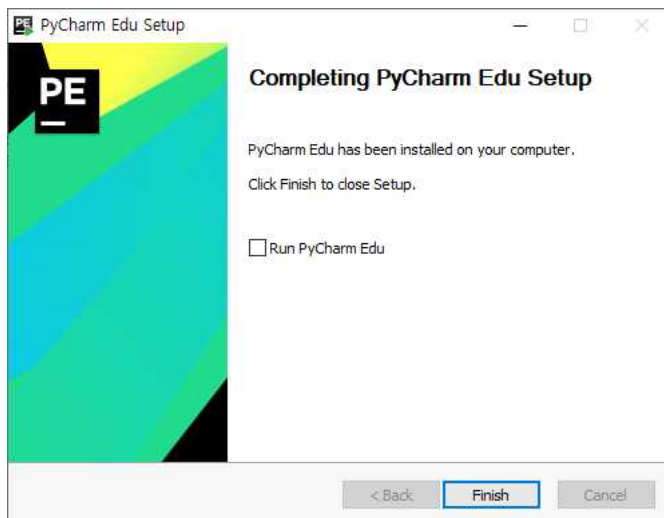
❶ 파이썬 버전 선택을 Python 3.8로 선택합니다.

❷ 바탕화면에 바로가기 아이콘을 64 비트로 선택해 줍니다.

9. [시작 메뉴 폴더 선택] 창입니다. 기본 상태로 [Install] 버튼을 눌러 설치를 진행합니다.



10. 설치가 완료되면 다음과 같은 창이 뜹니다. [Finish] 버튼을 눌러 설치를 완료합니다.

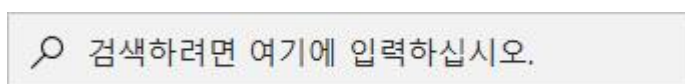


02 파이썬 실습 환경 설정하기

여기서는 파이썬 실습 환경을 설정해 봅니다.

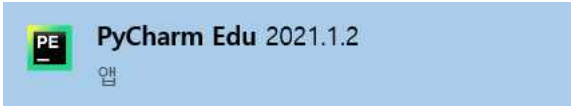
파이썬 실행하기

1. 데스크 탑 좌측 하단에 있는 [검색] 창을 찾아 [pycharm edu]를 입력합니다.

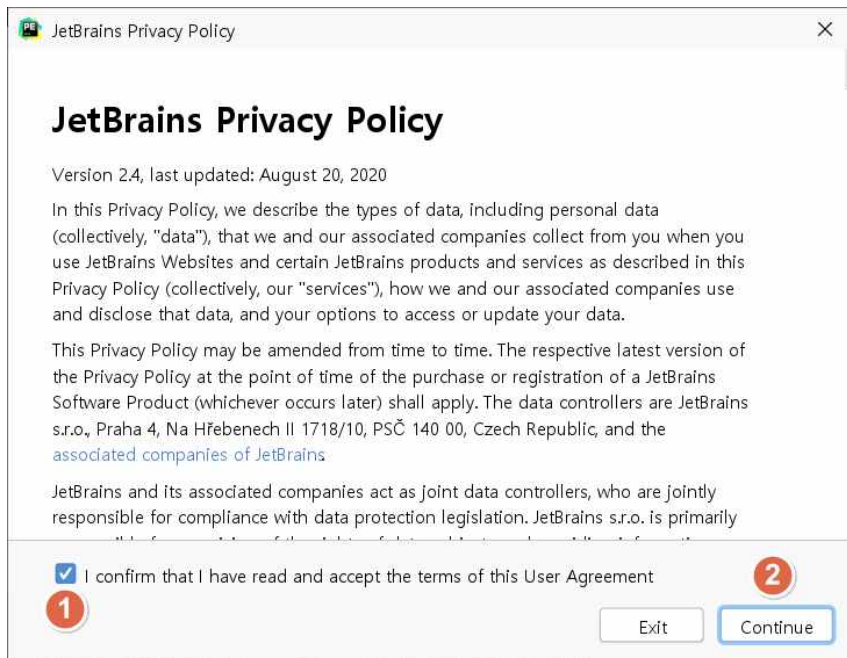




2. 다음 프로그램을 실행합니다.



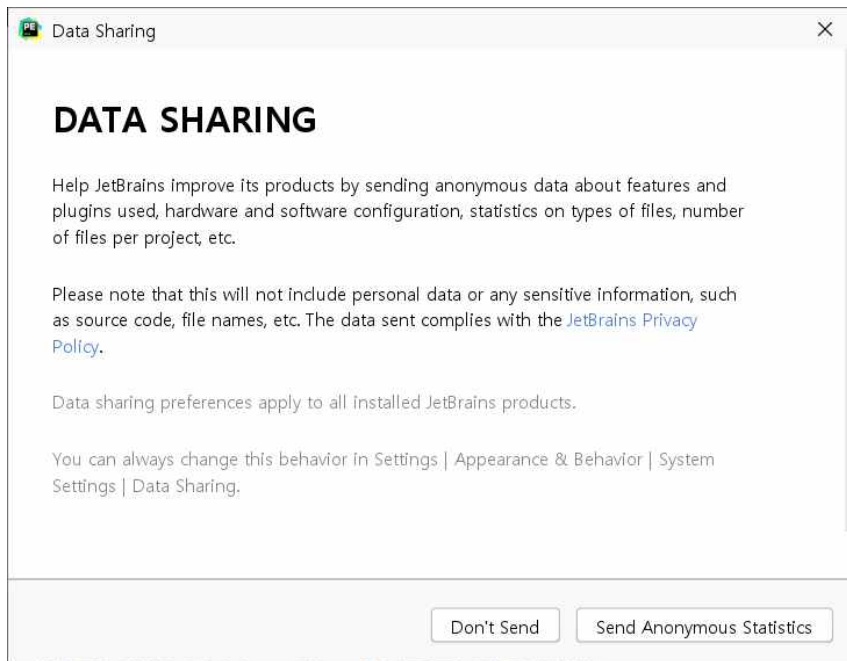
3. 처음엔 다음과 제트브레인 사의 [개인 정보 정책] 창이 뜹니다.



❶ 개인 정보 정책 동의를 체크해 줍니다.

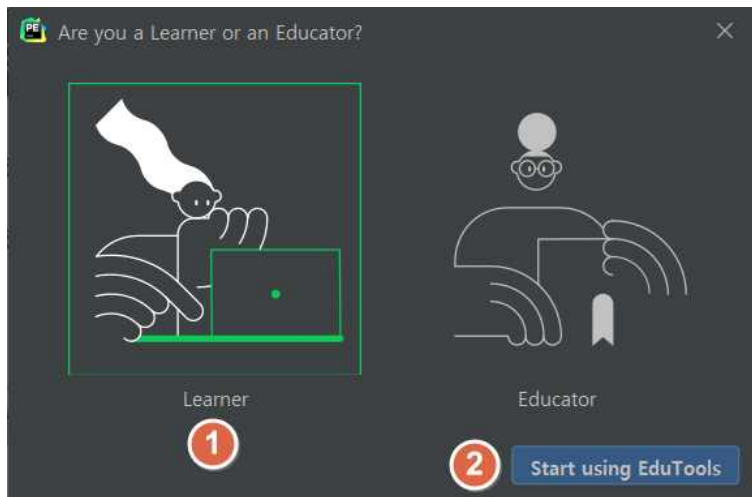
❷ [Continue] 버튼을 눌러 진행합니다.

4. 다음과 같이 제트브레인 사에 툴에 대한 [데이터 공유] 창이 뜹니다.



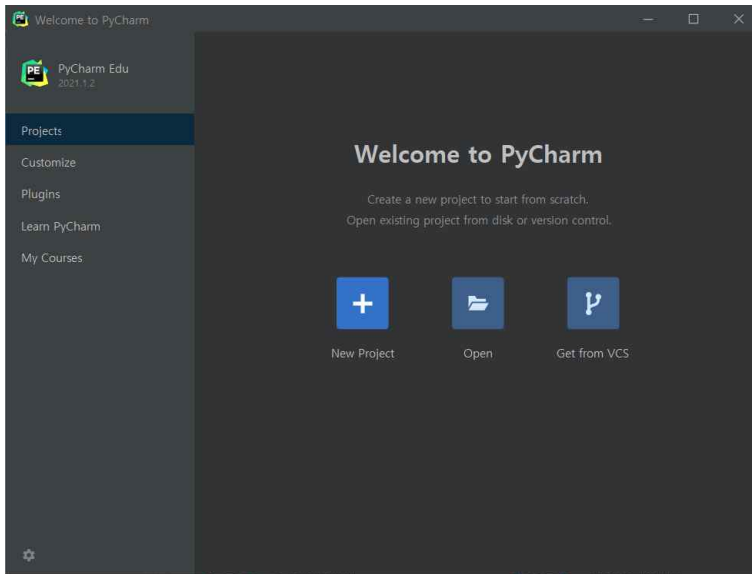
독자 여러분의 선택에 따라 두 버튼 중에 하나를 눌러줍니다. 어떤 버튼을 눌러도 상관없습니다.

5. 다음과 같이 [학습자 또는 교육자 선택] 창이 뜹니다. 여기서는 [학습자]로 선택한 후, [Start using EduTools] 버튼을 누릅니다.

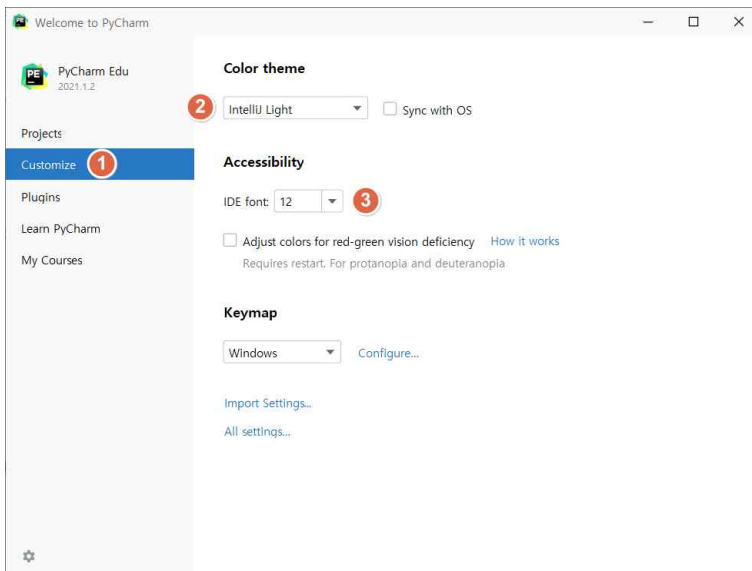


파이썬 프로젝트 생성하기

6. 다음과 같이 [Welcome to PyCharm] 창이 뜹니다.

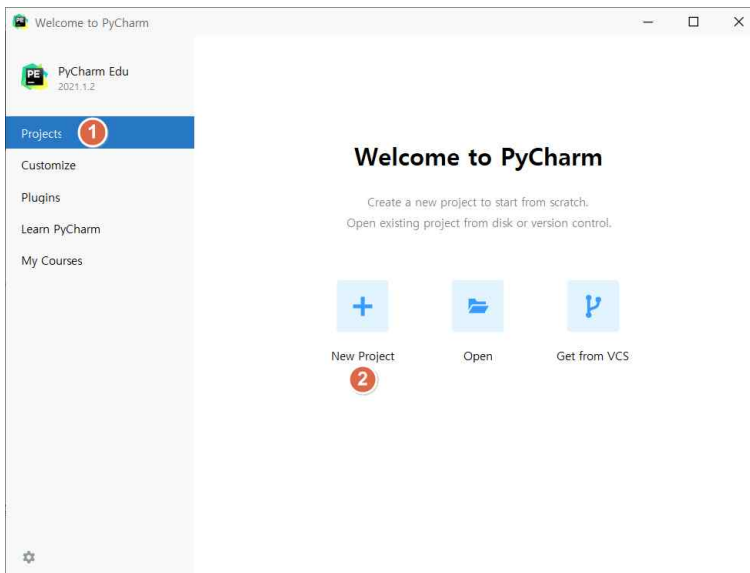


7. 다음과 같이 개발 화면에 대한 [색깔 테마]를 선택해 줍니다.



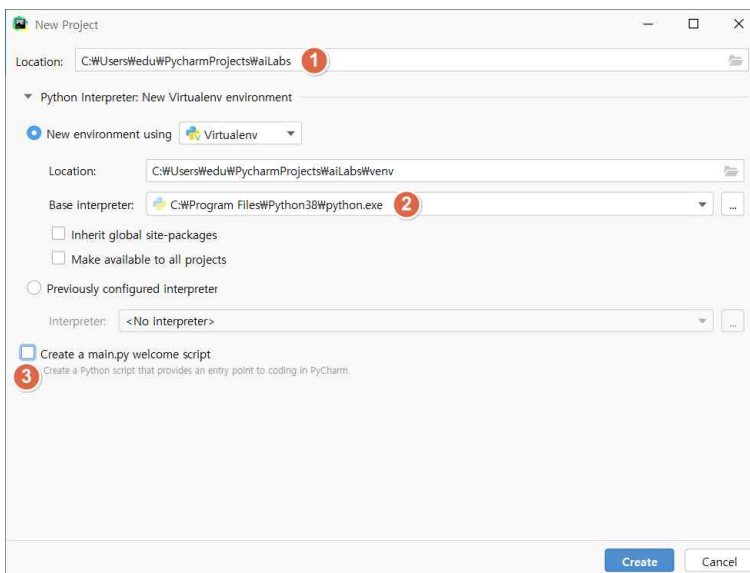
- ❶ [Customize] 메뉴를 선택합니다.
- ❷ [Color theme]를 선택해 줍니다. 필자의 경우엔 [IntelliJ Light]로 선택하였습니다.
- ❸ 폰트의 크기도 적절히 변경해줍니다.

8. 다음과 같이 [Projects] 메뉴를 선택합니다.



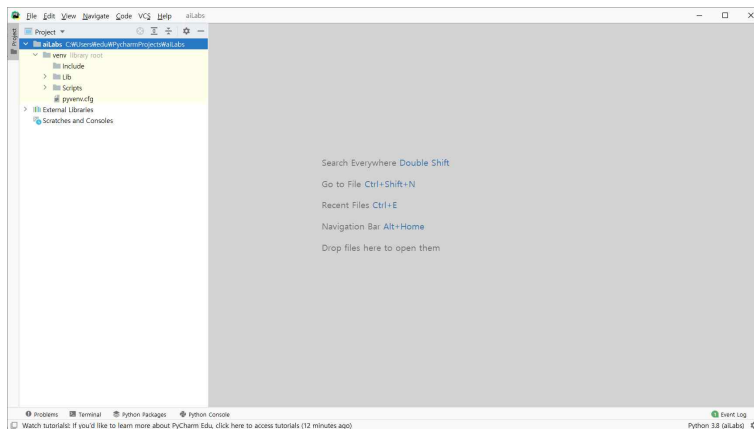
[+ New Project] 메뉴를 선택합니다.

9. 다음은 [New Project] 창입니다. 다음과 같이 설정한 후, [Create] 버튼을 누릅니다.



- ❶ 프로젝트 디렉터리를 입력합니다. 필자는 [aiLabs]로 입력하였습니다.
- ❷ 사용할 파이썬 프로그램을 선택합니다. 이 책에서는 파이썬 3.8을 사용하고 있습니다.
- ❸ main.py 파일 생성을 체크 해제합니다.

10. 다음과 같이 프로젝트가 생성됩니다.



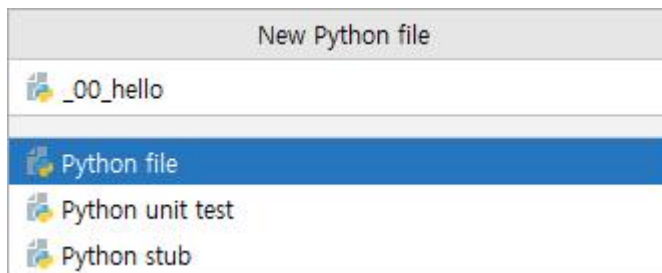
파이썬 파일 생성하기

8. 다음과 같이 파일을 생성합니다.



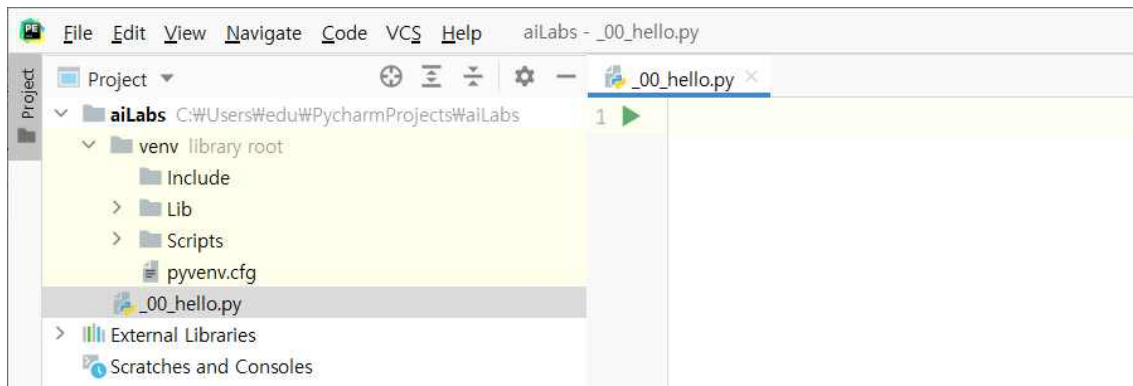
aiLabs 프로젝트 상에서 마우스 오른쪽 버튼을 눌러 팝업 창을 띄운 후, [New]-[Python File] 메뉴를 선택합니다.

9. 파일 이름을 입력한 후, 엔터키를 칩니다. 여기서는 [_00_hello]를 입력합니다.



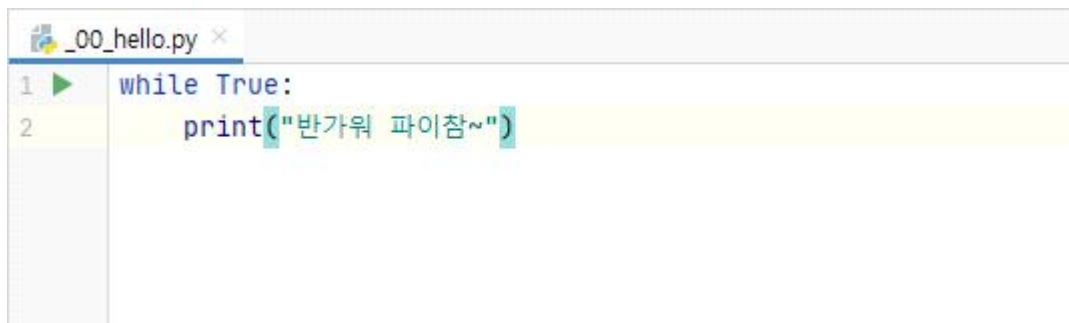
*** [_00_hello.py]로 입력할 수도 있습니다.

10. 다음과 같이 파일이 생성됩니다.



파이썬 프로그램 작성하기

11. 다음과 같이 프로그램을 작성합니다.



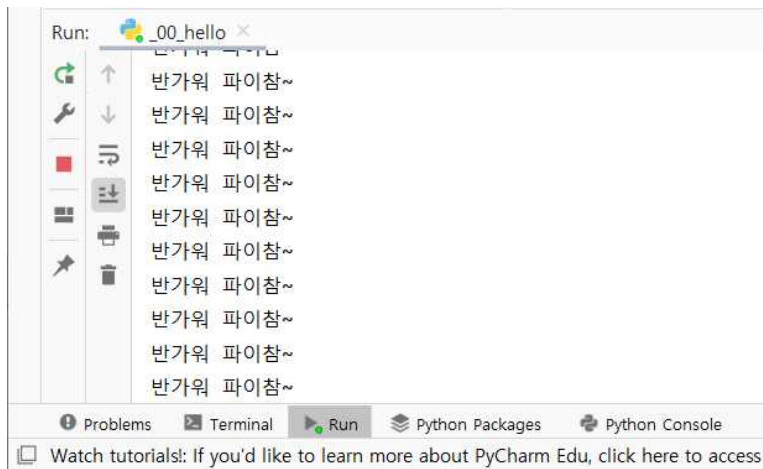
파이썬 프로그램 실행하기

12. 다음과 같이 프로그램을 실행시킵니다.



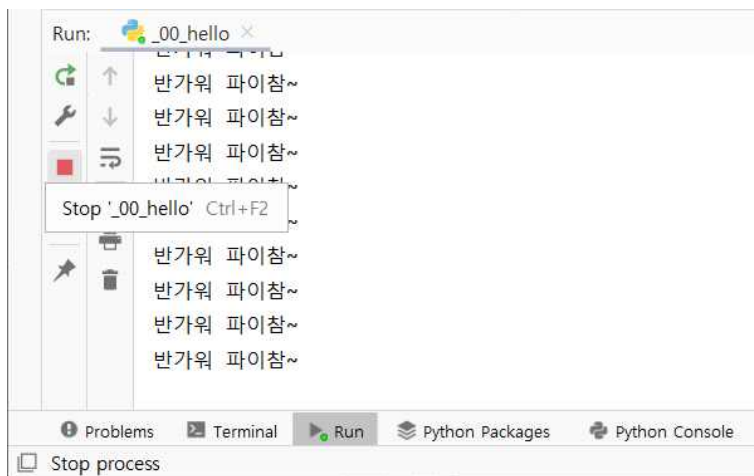
초록색 삼각형 기호를 마우스로 눌러줍니다.

13. 다음은 실행 결과 화면입니다.



파이썬 프로그램 종료하기

14. 다음과 같이 프로그램을 종료합니다.



빨간색 사각형 기호를 마우스로 눌러줍니다.

15. 다음은 프로그램 종료 화면입니다.

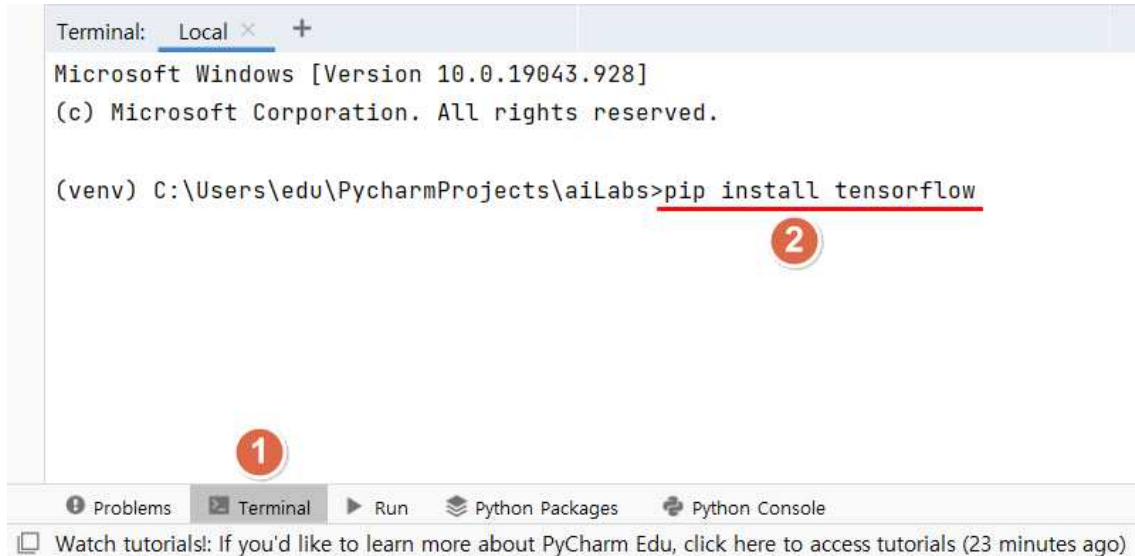


03 딥러닝 실습 환경 설정하기

여기서는 딥러닝 실습 환경을 설정해 봅니다.

tensorflow 라이브러리 설치하기

1. 다음과 같이 tensorflow 라이브러리를 설치합니다.



❶ 화면 하단에서 [Terminal] 탭을 선택합니다.

❷ pip 명령을 이용하여 tensorflow 라이브러리를 설치합니다.

2. 다음은 설치가 정상적으로 완료된 화면입니다.

```
Terminal: Local +
Installing collected packages: urllib3, pyasn1, idna, chardet, certifi, six, rsa, requests, pyasn1-modules, oauthlib, cachetools, requests-oauthlib, google-auth, wheel, werkzeug, tensorbo
ard-plugin-wit, tensorboard-data-server, protobuf, numpy, markdown, grpcio, google-auth-oauthlib, absl-py, wrapt, typing-extensions, termcolor, tensorflow-estimator, tensorboard, opt-eins
um, keras-preprocessing, keras-nightly, h5py, google-pasta, gast, flatbuffers, astunparse, tensorflow
Running setup.py install for wrapt ... done
Running setup.py install for termcolor ... done
Successfully installed absl-py-0.13.0 astunparse-1.6.3 cachetools-4.2.2 certifi-2021.5.30 chardet-4.0.0 flatbuffers-1.12 gast-0.4.0 google-auth-1.32.1 google-auth-oauthlib-0.4.4 google-pa
sta-0.2.0 grpcio-1.34.1 h5py-3.1.0 idna-2.10 keras-nightly-2.5.0.dev2021032900 keras-preprocessing-1.1.2 markdown-3.3.4 numpy-1.19.5 oauthlib-3.1.1 opt-einsum-3.3.0 protobuf-3.17.3 pyasn1
-0.4.8 pyasn1-modules-0.2.8 requests-2.25.1 requests-oauthlib-1.3.0 rsa-4.7.2 six-1.15.0 tensorboard-2.5.0 tensorboard-data-server-0.6.1 tensorboard-plugin-wit-1.8.0 tensorflow-2.5.0 tens
orflow-estimator-2.5.0 termcolor-1.1.0 typing-extensions-3.7.4.3 urllib3-1.26.6 werkzeug-2.0.1 wheel-0.36.2 wrapt-1.12.1
```

*** 설치 시간은 5분 정도 걸립니다.

3. 다음 사이트에 접속합니다.

<https://support.microsoft.com/help/2977003/the-latest-supported-visual-c-downloads>

4. 다음과 같이 [vc_redist.x64.exe] 파일을 마우스 선택하여 다운로드 합니다.

Visual Studio 2015, 2017 및 2019

Visual Studio 2015, 2017 및 2019용 Microsoft Visual C++ 재배포 가능 패키지를 다운로드합니다. 다음 업데이트는 Visual Studio 2015, 2017 및 2019용으로 지원되는 최신 Visual C++ 재배포 가능 패키지입니다. Universal C Runtime의 기본 버전이 포함되어 있습니다. 자세한 내용은 MSDN을 참조하세요.

- x86: [vc_redist.x86.exe](#)
- x64: [vc_redist.x64.exe](#)
- ARM64: [vc_redist.arm64.exe](#)

참고 Visual C++ 2015, 2017 및 2019 모두 동일한 재배포 가능 파일을 공유합니다.

예를 들어, Visual C++ 2019 재배포 가능 패키지를 설치하면 Visual C++ 2015 및 2017로 구축된 프로그램에도 영향을 줍니다. 그러나 Visual C++ 2015 재배포 가능 패키지를 설치하면 Visual C++ 2017 및 2019 재배포 가능 패키지로 인해 설치된 더 최신 버전 파일로 교체되지 않습니다.

각 버전마다 다른 버전과 공유되지 않는 고유한 런타임 파일이 있기 때문에 이전의 모든 Visual C++ 버전과 다릅니다.

*** 이 파일은 윈도우 상에서 텐서플로를 실행하기 위해 필요한 파일입니다.

5. 다운로드가 완료되면 설치를 합니다.



matplotlib 라이브러리 설치하기

6. 다음과 같이 matplotlib 라이브러리를 설치합니다.

```
(venv) C:\Users\edu\PycharmProjects\aiLabs>pip install matplotlib
```

*** matplotlib 라이브러리는 수학 그래픽 라이브러리입니다.

7. 다음은 설치가 정상적으로 완료된 화면입니다.

```
Terminal: Local  +
Collecting python-dateutil>=2.7
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
    |████████████████████| 227 kB 3.3 MB/s
Collecting pyparsing>=2.2.1
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
    |████████████████████| 67 kB 1.8 MB/s
Requirement already satisfied: six in c:\users\edu\pycharmprojects\ailabs\venv\lib\site-packages (from cyclar>=0.10->matplotlib) (1.15.0)
Installing collected packages: python-dateutil, pyparsing, pillow, kiwisolver, cyclar, matplotlib
Successfully installed cyclar-0.10.0 kiwisolver-1.3.1 matplotlib-3.4.2 pillow-8.3.1 pyparsing-2.4.7 python-dateutil-2.8.1

(venv) C:\Users\edu\PycharmProjects\aiLabs>
```

opencv 라이브러리 설치하기

8. 다음과 같이 opencv 라이브러리를 설치합니다.

```
(venv) C:\Users\edu\PycharmProjects\aiLabs>pip install --upgrade opencv-contrib-python
```

9. 다음은 설치가 정상적으로 완료된 화면입니다.

```
(venv) C:\Users\edu\PycharmProjects\aiLabs>pip install --upgrade opencv-contrib-python
Collecting opencv-contrib-python
  Downloading opencv_contrib_python-4.5.3.56-cp38-cp38-win_amd64.whl (41.8 MB)
    |████████████████████| 41.8 MB 6.4 MB/s
Requirement already satisfied: numpy>=1.17.3 in c:\users\edu\pycharmprojects\ailabs\venv\lib\site-packages (from opencv-contrib-python) (1.19.5)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.5.3.56
```

*** opencv 라이브러리는 영상 처리 라이브러리입니다.

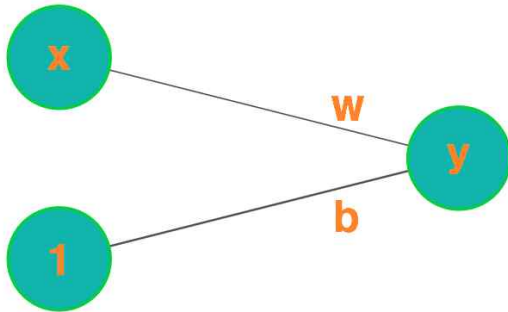
이상에서 파이참 개발 환경을 구성해 보았습니다.

02 딥러닝 동작 원리 이해하기

여기서는 단위 인공 신경(1입력 1출력 인공 신경)의 동작을 상식적인 수준에서 살펴보면서 딥러닝의 동작 원리를 이해해 봅니다. 또 딥러닝과 관련된 중요한 용어들, 예를 들어, 순전파, 목표 값, 역전파 오차, 오차 역전파, 학습률과 같은 용어들을 이해해 보도록 합니다.

01 기본 인공 신경 동작 살펴보기

다음은 앞에서 소개한 단일 인공 신경의 그림입니다. 이 인공 신경은 입력 노드 1개, 출력 노드 1개, 편향으로 구성된 단일 인공 신경입니다.



수식으로는 다음과 같이 표현합니다.

$$y = xw + 1b$$

이 수식에 대해서 구체적으로 생각해 봅니다. 다음과 같이 각 변수에 값을 줍니다.

$$\begin{aligned} x &= 2 \\ w &= 3 \\ b &= 1 \end{aligned}$$

그러면 식은 다음과 같이 됩니다.

$$y = 2 \times 3 + 1 \times 1$$

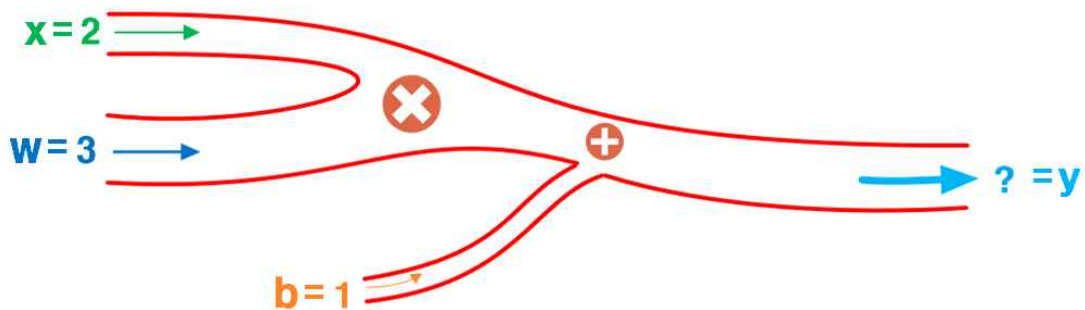
$$y = ?$$

y는 얼마가 될까요? 다음과 같이 계산해서 y는 7이 됩니다.

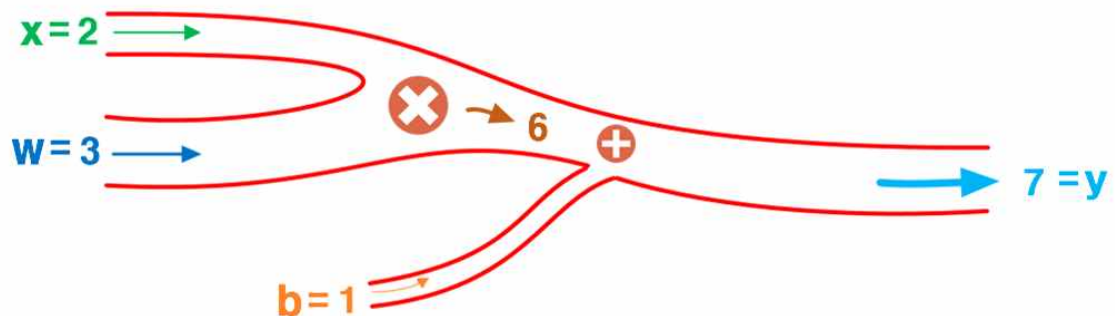
$$2 \times 3 + 1 \times 1 = 7$$

순전파

이 상황을 그림으로 생각해 봅시다. 다음과 같이 x, w, b 값이 y로 흘러가는 인공 신경 파이프가 있습니다. 이 과정을 인공 신경의 순전파라고 합니다.

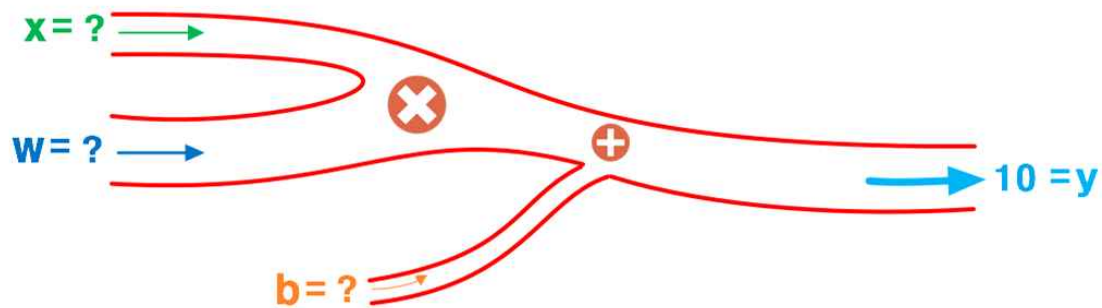


이 경우 y로 얼마가 나올까요? 앞에서 살펴본 대로 다음과 같은 과정을 거쳐 7이 흘러나오게 됩니다.



목표 값과 역전파 오차

그런데 y로 10이 나오게 하려면 들어오는 값들을 어떻게 바꿔야 할까요?



y값이 10이 되려면 3이 모자랍니다. x, w, b값을 적당히 증가시키면 y로 10에 가까운 값이 나오게 할 수 있겠죠? 그러면 x, w, b값을 어떤 기준으로 얼마나 증가시켜야 할까요? 이 과정을 자세히 살펴봅시다.

이전 그림에서 y로 7이 흘러나갔는데 우리는 이 값이 10이 되기를 원합니다. 여기서 10은 목표 값이 됩니다. 다음 수식에서 t는 목표 값 10을 갖습니다.

$$\begin{aligned} t &= 10 \\ y &= 7 \\ y_b &= y - t \\ y_b &= -3 \end{aligned}$$

y값은 현재 값 7인 상태이며, yb는 현재 값에서 목표 값을 뺀 값 -3이 됩니다. 이 때, yb값을 역전파 오차라고 하며, 역전파에 사용할 오차 값입니다.

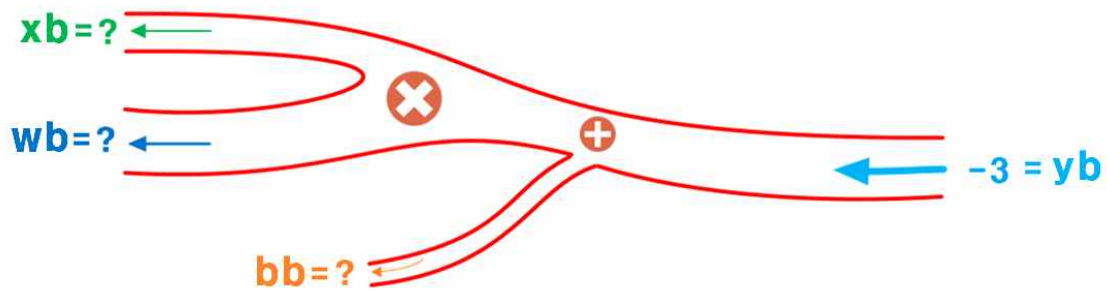
이름표로 정리하면 다음과 같습니다.

목표 값	t	10
현재 값	y	7
역전파 오차	$y_b = y - t$	-3

오차 역전파

이 상황을 그림으로 생각해 봅시다. 이번엔 yb의 값이 xb, wb, bb로 거꾸로 흘러가는 상황이

됩니다.



xb , wb , bb 를 어떤 기준으로 얼마나 값을 할당해야 할까요? 다음과 같은 방법은 어떨까요?

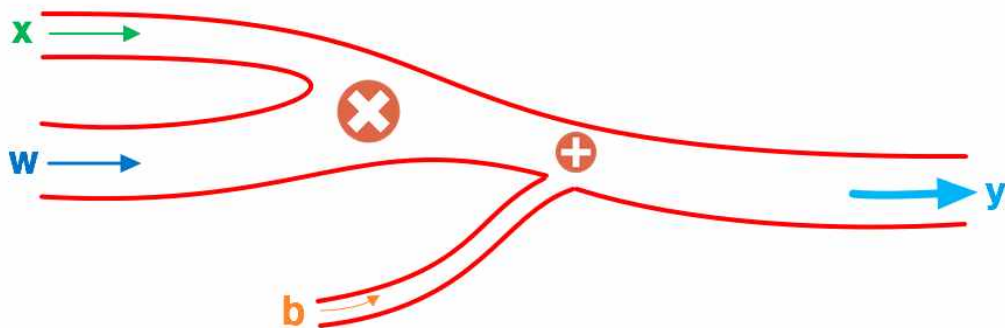
- 2는 3만큼 7로 갔어! 그러니까 -3도 3만큼 2로 돌아가게 하자!
- 3은 2만큼 7로 갔어! 그러니까 -3을 2만큼 3으로 돌아가게 하자!
- 1은 1만큼 7로 갔어! 그러니까 -3을 1만큼 1로 돌아가게 하자!

어떤가요? 설득력 있는 방법인가요? 이 방법이 바로 인공 신경에서 사용하는 오차 역전파입니다. 이 방법은 실제로 조금은 어려울 수 있는 편미분을 적용하여 얻은 방법으로 이 책에서는 직관적인 방법으로 대체하였습니다.

지금까지의 과정을 그림과 수식을 통해서 다시 한 번 정리해 봅니다.

순전파 정리하기

다음은 x , w , b 가 y 로 흘러가는 순전파를 나타냅니다.



다음은 이 그림에 대한 수식입니다.

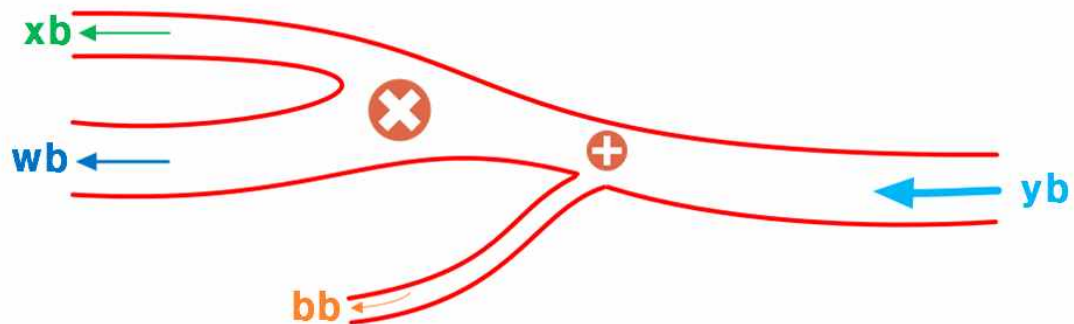
$$xw + 1b = y \quad ①$$

이 수식은 다음과 같은 의미를 갖습니다.

- x 는 w 만큼 y 로 갔어. $x \xrightarrow{w} y$
- w 는 x 만큼 y 로 갔어. $w \xrightarrow{x} y$
- b 는 1 만큼 y 로 갔어. $b \xrightarrow{1} y$

역전파 정리하기

다음은 y_b 가 x_b , w_b , b_b 로 흘러가는 역전파를 나타냅니다.



우리는 다음 사항이 궁금합니다.

- y_b 는 얼마만큼 x_b 로 가야해?
- y_b 는 얼마만큼 w_b 로 가야해?
- y_b 는 얼마만큼 b_b 로 가야해?

이에 대한 답은 다음과 같습니다.

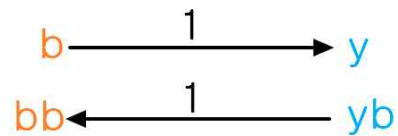
- x가 w만큼 y로 왔으니 yb도 w만큼 xb로 가야하는 거 아냐? ②



- w가 x만큼 y로 왔으니 yb도 x만큼 wb로 가야하는 거 아냐? ③



- b가 1만큼 y로 왔으니 yb도 1만큼 bb로 가야하는 거 아냐? ④



*** 이 방법은 실제로 편미분과 연쇄법칙을 이용하여 유도한 방법으로 이 책에서는 좀 더 직관적인 방법으로 대체하였습니다.

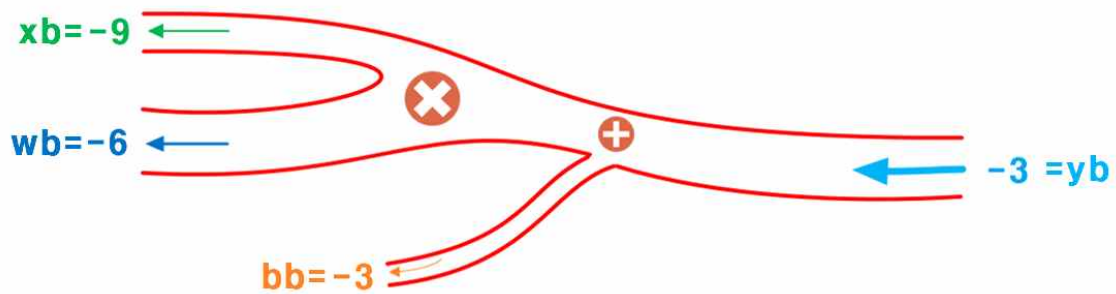
이를 수식으로 정리하면 다음과 같습니다.

$$x_b = y_b w \quad ②$$

$$w_b = y_b x \quad ③$$

$$b_b = y_b 1 \quad ④$$

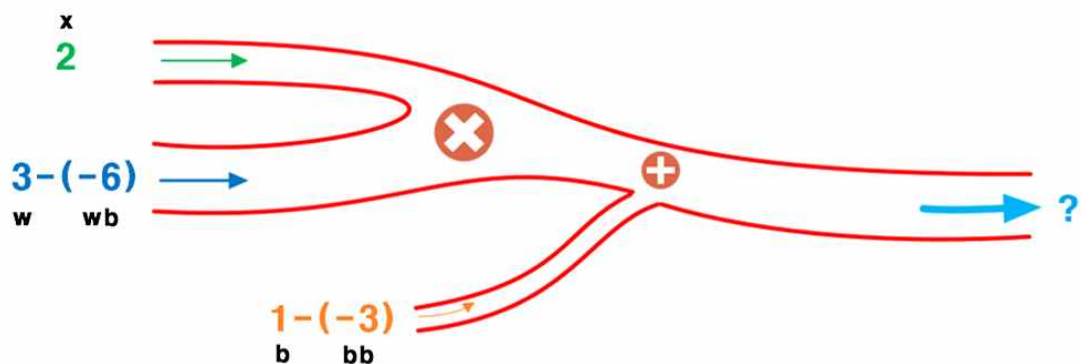
이 수식에 의해 x_b , w_b , y_b 는 다음 그림과 같이 계산됩니다.



*** 여기서 x 값은 앞부분에 또 다른 인공 신경과 연결되어 있을 경우 y 처럼 해당 인공 신경으로 역전파되는 값입니다. 역전파 된 x 값은 해당 인공 신경의 가중치와 편향 학습에 사용됩니다.

최적화하기

이렇게 구한 값을 다시 다음과 같이 밀어 넣으면 될까요? 앞에서 구한 w , b 의 값이 음수가 되기 때문에 일단 빼주어야 합니다. 그래야 원래 값이 증가하기 때문입니다.



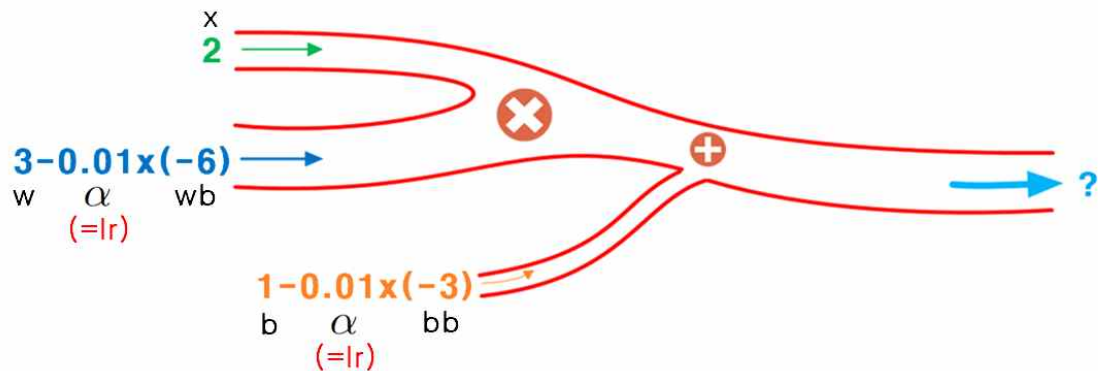
*** 여기서 x 값은 상수라고 가정합니다. x 값이 앞부분에 또 다른 인공 신경과 연결되어 있을 경우엔 해당 인공 신경의 출력 값이 됩니다. 여기서는 최초 입력 값이라고 가정합니다. 이런 경우 입력 층이라고 합니다.

그런데 w , b 값이 너무 큼니다. 이 상태로 계산을 하면 새로운 y 값은 $(2 \times 9 + 4)$ 와 같이 계산되어 22가 되게 되며, 우리가 원하는 10보다 더 큰 값이 나오게 됩니다. 구체적인 계산 과정은 다음과 같습니다.

$ \begin{aligned} x &= 2 \\ w &= 3 - (-6) \\ b &= 1 - (-3) \\ y &= x \cdot w + b = 2 \cdot 9 + 4 = 22 \end{aligned} $
--

학습률

그러면 이런 방법은 어떨까요? w , b 에 적당한 값을 곱해주어 값을 줄이는 겁니다. 여기서 는 0.01을 곱해줍니다. 그러면 다음과 같이 계산할 수 있습니다.



*** 여기서 α 는 학습률이라고 하며 뒤에서는 lr이라는 이름으로 구현합니다. lr은 learning rate의 약자로 학습률을 의미합니다.

이렇게 하면 $2 * (3.06) + 1.03 = 7.15$ 가 나옵니다. 오! 이렇게 조금씩 늘어나가면 10을 만들 수 있겠네요!

여기서 곱해준 0.01은 학습률이라고 하는 값입니다. 일반적으로 학습률 값은 0.01로 시작하여 학습이 진행되는 상황에 따라 조금씩 늘이거나 줄여서 사용합니다.

경사 하강법과 인공 신경망 학습

위 그림에 따라 새로운 w , b 값을 구하는 수식은 다음과 같습니다.

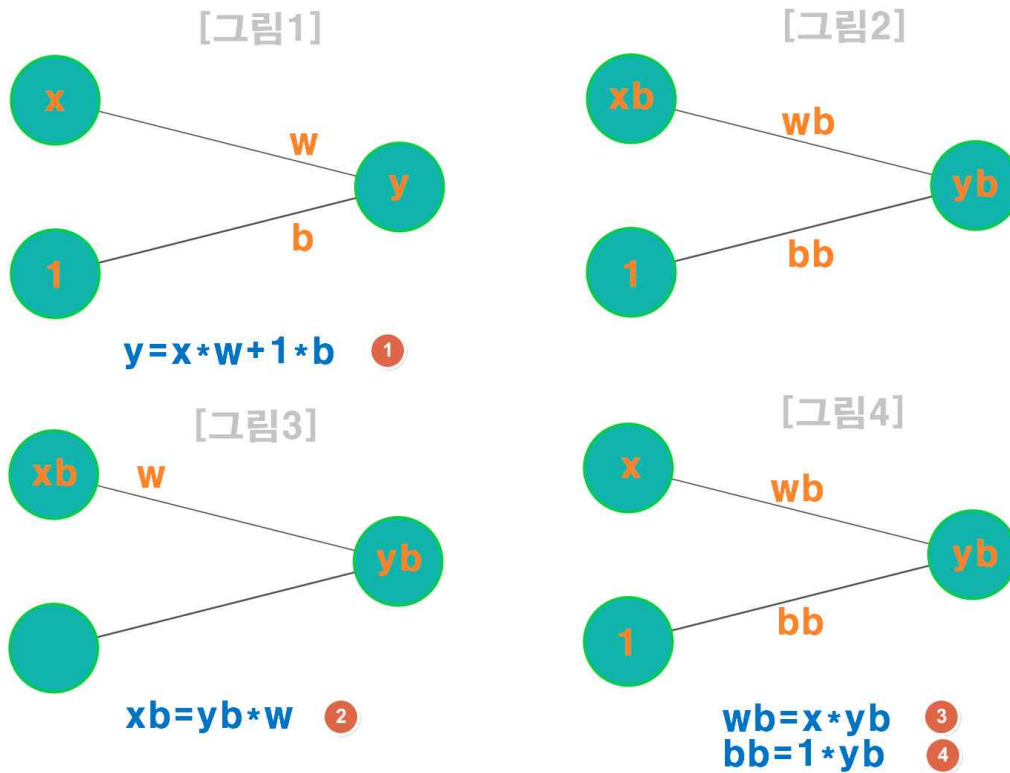
$$\begin{aligned} w &= w - \alpha w_b \quad 5 \\ b &= b - \alpha b_b \quad 6 \end{aligned}$$

이 수식을 경사하강법이라고 합니다. 그리고 이 수식을 적용하여 w , b 값을 갱신하는 과정을 인공 신경망의 학습이라고 합니다. 여러분은 방금 전에 1회의 학습을 수행하는 과정을 보신 겁니다. 이 과정을 컴퓨터를 이용하여 반복하여 수행하면 우리가 원하는 값을 얻게 해 주는 하나짜리 인공 신경망을 만들 수 있습니다.

02 기본 인공 신경 동작 구현해 보기

지금까지의 과정을 그림과 수식을 통해 정리한 후, 구현을 통해 확인해 봅니다.

다음 그림은 지금까지 살펴본 입력1 출력1로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② xb 값은 앞부분에 또 다른 인공 신경과 연결되어 있을 경우 yb 처럼 해당 인공 신경으로 역전파되는 값입니다. 역전파된 xb 값은 해당 인공 신경의 가중치와 편향 학습에 사용됩니다.

*** 편미분과 연쇄법칙을 통해 역전파식을 유도하는 방법은 부록에 소개되어 있으니 궁금한 독자는 참고하시기 바랍니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

<p>순전파</p> $xw + 1b = y$ <p>①</p>

입력 역전파

$$y_b w = x_b \quad \textcircled{2}$$

가중치, 편향 역전파

$$x y_b = w_b \quad \textcircled{3}$$

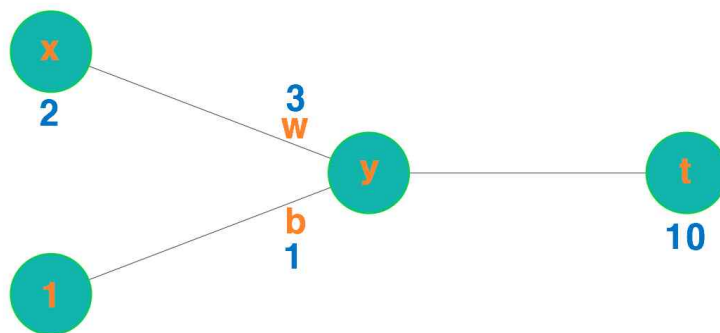
$$1 y_b = b_b \quad \textcircled{4}$$

인공 신경망 학습

$$w = w - \alpha w_b \quad \textcircled{5}$$

$$b = b - \alpha b_b \quad \textcircled{6}$$

지금까지의 과정을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력 값 x , 가중치 w , 편향 b 는 각각 2, 3, 1이고 목표 값 t 는 10입니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.

222_1.py

```
01 x = 2
02 t = 10
03 w = 3
04 b = 1
05
06 y = x*w + 1*b # ①
07 print('y = %6.3f' %y)
08
```

```

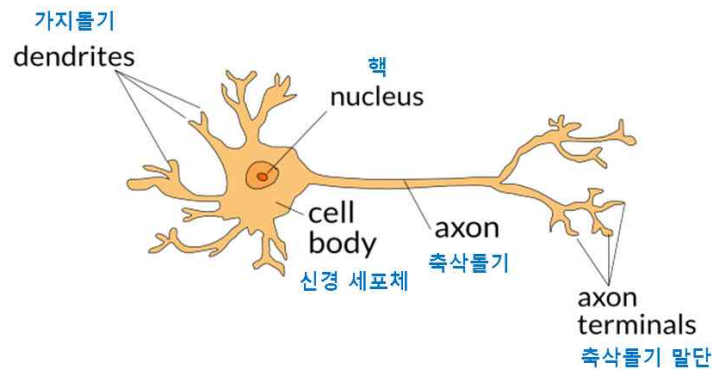
09 yb = y - t
10 xb = yb*w # ②
11 wb = yb*x # ③
12 bb = yb*1 # ④
13 print('xb = %6.3f, wb = %6.3f, bb = %6.3f'%(xb, wb, bb))
14
15 lr = 0.01
16 w = w - lr*wb # ⑤
17 b = b - lr*bb # ⑥
18 print('x = %6.3f, w = %6.3f, b = %6.3f'%(x, w, b))

```

01 : 변수 x를 선언한 후, 2로 초기화합니다.

02 : 변수 t를 선언한 후, 10으로 초기화합니다.

03 : 가중치 변수 w를 선언한 후, 3으로 초기화합니다. 가중치 w는 입력 값의 강도, 세기라고도 하며 입력 값을 증폭 시키거나 감소시키는 역할을 합니다. 인공 신경도 가지 돌기의 두께에 따라 입력 신호가 증폭되거나 감소될 수 있는데, 이런 관점에서 가중치는 가지 돌기의 두께에 해당되는 변수로 생각할 수 있습니다.



04 : 편향 변수 b를 선언한 후, 1로 초기화합니다. 편향은 가중치를 거친 입력 값의 합(=전체 입력 신호)에 더해지는 값으로 입력신호를 좀 더 세게 해주거나 약하게 하는 역할을 합니다.

06 : 순전파 수식을 구현합니다.

07 : print 함수를 호출하여 순전파 결과 값 y를 출력합니다. 소수점 이하 3자리까지 출력합니다.

09 : yb 변수를 선언한 후, 순전파 결과 값에서 목표 값을 빼 오차 값을 넣어줍니다.

10 : xb 변수를 선언한 후, 입력 값에 대한 역전파 값을 받아봅니다. 이 부분은 이 예제에서 필요한 부분은 아니며, 역전파 연습을 위해 추가하였습니다.

11 : wb 변수를 선언한 후, 가중치 값에 대한 역전파 값을 받습니다.

12 : bb 변수를 선언한 후, 편향 값에 대한 역전파 값을 받습니다.

13 : print 함수를 호출하여 역전파 결과 값 wb, bb를 출력합니다. 소수점 이하 3자리까지 출력합니다.


15 : 학습률 변수 lr을 선언한 후, 0.01로 초기화합니다.

16 : wb 역전파 값에 학습률을 곱한 후, w값에서 빼줍니다. 이 과정에서 w 변수에 대한 학습

이 이루어집니다.

17 : bb 역전파 값에 학습률을 곱한 후, b값에서 빼줍니다. 이 과정에서 b 변수에 대한 학습이 이루어집니다.

18 : print 함수를 호출하여 학습된 결과 값 w, b를 출력합니다. 소수점 이하 3자리까지 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
y = 7.000
xb = -9.000, wb = -6.000, bb = -3.000
x = 2.000, w = 3.060, b = 1.030
```

현재 y값은 7입니다. wb, bb 값을 확인합니다. 또, w, b 값을 확인합니다.

반복 학습 2회 수행하기

여기서는 반복 학습 2회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.


222_1.py

```
01 x = 2
02 t = 10
03 w = 3
04 b = 1
05
06 for epoch in range(2):
07
08     print('epoch = %d' %epoch)
09
10     y = x*w + 1*b
11     print(' y = %6.3f' %y)
12
13     yb = y - t
14     xb = yb*w
15     wb = yb*x
16     bb = yb*1
17     print(' xb = %6.3f, wb = %6.3f, bb = %6.3f'%(xb, wb, bb))
18
19     lr = 0.01
20     w = w - lr*wb
```

```
21     b = b - lr*bb
22     print(' x  = %6.3f, w  = %6.3f, b  = %6.3f'%(x, w, b))
```

06 : epoch값을 0에서 2 미만까지 바꾸어가며 8~22줄을 2회 수행합니다.

08 : print 함수를 호출하여 epoch 값을 출력해 줍니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 0
y  =  7.000
xb = -9.000, wb = -6.000, bb = -3.000
x  =  2.000, w  =  3.060, b  =  1.030
epoch = 1
y  =  7.150
xb = -8.721, wb = -5.700, bb = -2.850
x  =  2.000, w  =  3.117, b  =  1.058
```

y 값이 7에서 7.150으로 바뀌는 것을 확인합니다. wb, bb 값을 확인합니다. 또, w, b 값을 확인합니다.


반복 학습 20회 수행하기

여기서는 반복 학습 20회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
06 for epoch in range(20):
```

06 : epoch값을 0에서 20 미만까지 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 18
y = 8.808
xb = -4.437, wb = -2.383, bb = -1.192
x = 2.000, w = 3.747, b = 1.374
epoch = 19
y = 8.868
xb = -4.242, wb = -2.264, bb = -1.132
x = 2.000, w = 3.770, b = 1.385

```

y 값이 8.868까지 접근하는 것을 확인합니다.


반복 학습 200회 수행하기

여기서는 반복 학습 200회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
06 for epoch in range(200):
```

06 : epoch값을 0에서 200 미만까지 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 198
y = 10.000
xb = -0.000, wb = -0.000, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 199
y = 10.000
xb = -0.000, wb = -0.000, bb = -0.000
x = 2.000, w = 4.200, b = 1.600

```

y 값이 10.000에 수렴하는 것을 확인합니다. 이 때, 가중치 w는 4.2, 편향 b는 1.6에 수렴합니다.

오차 값 계산하기

여기서는 인공 신경망을 통해 얻어진 예측 값과 목표 값의 오차를 계산하는 부분을 추가해 보

니다. 오차(error)는 손실(loss) 또는 비용(cost)이라고도 합니다. 오차 값이 작을수록 예측을 잘하는 인공 신경망입니다.

1. 다음과 같이 예제를 수정합니다.

222_1.py


```
01 x = 2
02 t = 10
03 w = 3
04 b = 1
05
06 for epoch in range(200):
07
08     print('epoch = %d' %epoch)
09
10     y = x*w + 1*b
11     print(' y = %6.3f' %y)
12
13     E = (y-t)**2/2
14     print(' E = %.7f' %E)
15     if E < 0.0000001:
16         break
17
18     yb = y - t
19     xb = yb*w
20     wb = yb*x
21     bb = yb*1
22     print(' xb = %6.3f, wb = %6.3f, bb = %6.3f'%(xb, wb, bb))
23
24     lr = 0.01
25     w = w - lr*wb
26     b = b - lr*bb
27     print(' x = %6.3f, w = %6.3f, b = %6.3f'%(x, w, b))
```

13 : 변수 E를 선언한 후, 다음과 같은 형태의 수식을 구현합니다.

$$E = \frac{1}{2}(y - t)^2$$

y의 값이 t에 가까울수록 E의 값은 0에 가까워집니다. 즉, 오차 값이 0에 가까워집니다. 이 수식을 오차함수 또는 손실함수 또는 비용함수라고 합니다.

14 : print 함수를 호출하여 오차 값 E를 출력합니다. 소수점 이하 7자리까지 출력합니다.
15, 16 : 오차 값 E가 0.0000001(1천만분의1)보다 작으면 break문을 수행하여 6줄의 for문을 빠져 나갑니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 171
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 172
y = 10.000
E = 0.0000001
```

epoch 값이 172((173+1)회 째)일 때 for 문을 빠져 나갑니다. y값은 10에 수렴합니다.


학습률 변경하기

여기서는 학습률 값을 변경시켜 보면서 학습의 상태를 살펴봅니다.

1. 다음과 같이 예제를 수정합니다.

24	lr = 0.05
----	-----------

24 : 학습률 값을 0.05로 변경합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 30
y = 9.999
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.001
x = 2.000, w = 4.200, b = 1.600
epoch = 31
y = 10.000
E = 0.0000001
```


32(31+1)회 째 학습이 완료되는 것을 볼 수 있습니다.

3. 다음과 같이 예제를 수정합니다.

222_1.py

```
24 lr = 0.005
```

24 : 학습률 값을 0.005로 변경합니다.

4.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.


```
epoch = 198
y = 9.980
E = 0.0001991
xb = -0.084, wb = -0.040, bb = -0.020
x = 2.000, w = 4.192, b = 1.596
epoch = 199
y = 9.981
E = 0.0001893
xb = -0.082, wb = -0.039, bb = -0.019
x = 2.000, w = 4.192, b = 1.596
```

200(199+1)회 때 학습이 완료되지 않은 상태로 종료되는 것을 볼 수 있습니다.

5. 다음과 같이 예제를 수정합니다.

```
06 for epoch in range(2000):
```

06 : epoch값을 0에서 2000 미만까지 수행합니다.

6.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

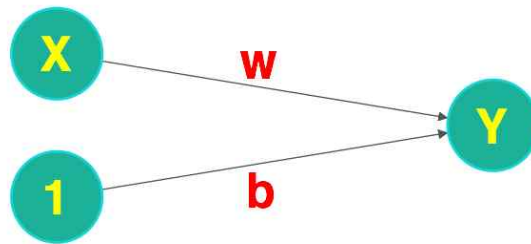
```
epoch = 348
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 349
y = 10.000
E = 0.0000001
```

350(349+1)회 때 학습이 완료되는 것을 볼 수 있습니다.

03 $y=3*x+1$ 학습시켜 보기

여기서는 다음과 같은 숫자들의 집합 X, Y를 이용하여, 단일 인공 신경을 학습시켜 봅니다.

X:	-1	0	1	2	3	4
Y:	-2	1	4	7	10	13



그래서 다음 함수를 근사하는 인공 신경 함수를 만들어 보도록 합니다.

$$y = f(x) = 3*x + 1 \text{ (x는 실수)}$$

인공 신경을 학습시키는 과정은 w, b 값을 X, Y 값에 맞추어 가는 과정입니다. 그래서 학습이 진행됨에 따라 w 값은 3에 가까운 값으로, b 값은 1에 가까운 값으로 이동하게 됩니다.

*** 이 예제는 1장에서 tensorflow를 이용하여 수행했던 예제를 재구현하고 있습니다.

1. 다음과 같이 예제를 작성합니다.

223_1.py

```
01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 y = xs[0]*w + 1*b
07 print("x = %6.3f, y = %6.3f" %(xs[0], y))
08
09 t = ys[0]
10 E = (y-t)**2/2
11 print('E = %.7f' %E)
12
```

```

13 yb = y - t
14 wb = yb*xs[0]
15 bb = yb*1
16 print('wb = %6.3f, bb = %6.3f'%(wb, bb))
17
18 lr = 0.01
19 w = w - lr*wb
20 b = b - lr*bb
21 print('w = %6.3f, b = %6.3f'%(w, b))

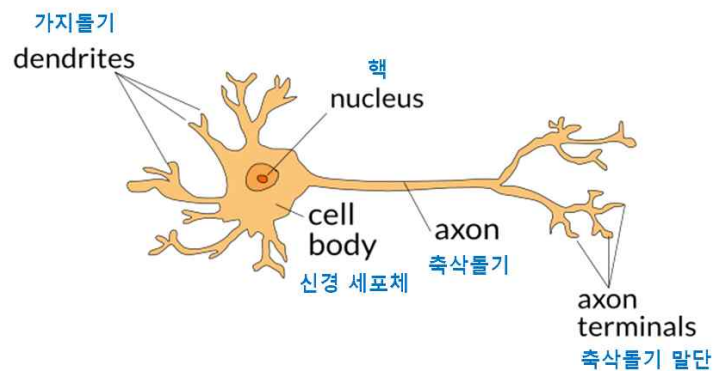
```

01, 02 : 실수 형 리스트 변수 xs, ys를 선언한 후, 다음 X, Y 값으로 초기화합니다.

X:	-1	0	1	2	3	4
Y:	-2	1	4	7	10	13

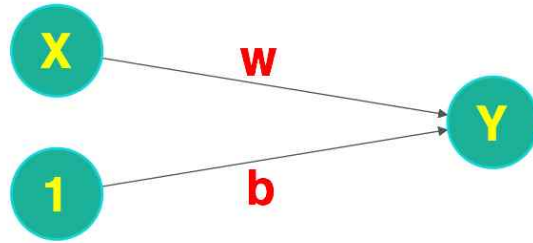
숫자 뒤에 점(.)은 실수를 나타냅니다.

03 : 입력 값의 가중치 값을 저장할 변수 w를 선언한 후, 10.으로 초기화합니다. 10.은 임의로 선택한 값입니다. 입력 값의 가중치는 입력 값의 강도, 세기라고도 하며 입력 값을 증폭시키거나 감소시키는 역할을 합니다. 인공 신경도 가지 돌기의 두께에 따라 입력 신호가 증폭되거나 감소될 수 있는데, 이런 관점에서 가중치는 가지 돌기의 두께에 해당되는 변수로 생각할 수 있습니다.



04 : 인공 신경의 편향 값을 저장할 변수 b를 선언한 후, 10.으로 초기화합니다. 10.은 임의로 선택한 값입니다. 편향 값은 가중치를 거친 입력 값의 합(=전체 입력신호)에 더해지는 값으로 입력신호를 좀 더 세게 해주거나 약하게 하는 역할을 합니다.

06 : 다음과 같이 단일 인공 신경을 수식으로 표현합니다.



$$y = xw + 1b$$

$$= xw + b$$

일단 `xs[0]` 항목을 `w`에 곱한 후, `b`를 더해준 후, 변수 `y`에 대입해 줍니다. 이 과정에서 순전파가 이루어집니다. 즉, `xs[0]` 항목이 `w`에 곱해지고 `b`와 더해져 `y`에 도달하는 과정을 순전파라고 합니다. 순전파 결과 얻어진 `y`값을 인공 신경망에 의한 예측 값이라고 합니다.

07 : `print` 함수를 호출하여 `xs[0]`, `y` 값을 출력합니다.

09 : 변수 `t`를 선언한 후, `ys[0]`값을 받습니다. `ys[0]`은 인공 신경망에 대한 `xs[0]`값의 목표 값입니다.

10 : 변수 `E`를 선언한 후, 다음과 같은 형태의 수식을 구현합니다.

$$E = \frac{1}{2}(y - t)^2$$

`y`의 값이 `t`에 가까울수록 `E`의 값은 0에 가까워집니다. 즉, 오차 값이 0에 가까워집니다. 이 수식을 오차함수 또는 손실함수 또는 비용함수라고 합니다.

11 : `print` 함수를 호출하여 `E` 값을 출력합니다. 소수점 이하 7자리까지 출력합니다.

13 : `yb` 변수를 선언한 후, 순전파 결과 값에서 목표 값을 빼 오차 값을 넣어줍니다.

14 : `wb` 변수를 선언한 후, 가중치 값에 대한 역전파 값을 받습니다.

15 : `bb` 변수를 선언한 후, 편향 값에 대한 역전파 값을 받습니다.

16 : `print` 함수를 호출하여 역전파 결과 값 `wb`, `bb`를 출력합니다. 소수점 이하 3자리까지 출력합니다.


18 : 학습률 변수 `lr`을 선언한 후, 0.01로 초기화합니다.

19 : `wb` 역전파 값에 학습률을 곱한 후, `w`값에서 빼줍니다. 이 과정에서 `w` 변수에 대한 학습이 이루어집니다.

20 : `bb` 역전파 값에 학습률을 곱한 후, `b`값에서 빼줍니다. 이 과정에서 `b` 변수에 대한 학습이 이루어집니다.

21 : `print` 함수를 호출하여 학습이 1회 수행된 `w`, `b` 값을 출력합니다. 소수점 이하 3자리까지

지 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
x = -1.000, y = 0.000
E = 2.0000000
wb = -2.000, bb = 2.000
w = 10.020, b = 9.980
```

w, b 값이 각각 10.020, 9.980으로 표시되는 것을 확인합니다.

전체 입력 데이터 학습 수행하기

이제 다음 좌표값 전체에 대해 1회 학습을 수행해 봅니다.

X:	-1	0	1	2	3	4
Y:	-2	1	4	7	10	13

1. 다음과 같이 예제를 수정합니다.

223_1.py

```
01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 for n in range(6):
07
08     y = xs[n]*w + 1*b
09     print("x = %6.3f, y = %6.3f" %(xs[n], y))
10
11     t = ys[n]
12     E = (y-t)**2/2
13     print('E = %.7f' %E)
14
15     yb = y - t
16     wb = yb*xs[n]
17     bb = yb*1
18     print('wb = %6.3f, bb = %6.3f'%(wb, bb))
19
```

```

20 lr = 0.01
21 w = w - lr*wb
22 b = b - lr*bb
23 print('w = %6.3f, b = %6.3f'%(w, b))
24
25 print("="*25)


```

06 : n값을 0에서 6 미만까지 바꾸어가며 08~25줄을 6회 수행합니다.

09, 09, 16 : xs[0]을 xs[n]으로 변경합니다.

11 : ys[0]을 ys[n]으로 변경합니다.

25 : 실행 경계를 표시하기 위해 "="을 25개 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

x = -1.000, y = 0.000  x = 2.000, y = 29.453
E = 2.00000000        E = 252.0662245
wb = -2.000, bb = 2.000  wb = 44.906, bb = 22.453
w = 10.020, b = 9.980  w = 9.412, b = 9.507
=====
x = 0.000, y = 9.980  x = 3.000, y = 37.742
E = 40.3202000        E = 384.8117627
wb = 0.000, bb = 8.980  wb = 83.226, bb = 27.742
w = 10.020, b = 9.890  w = 8.580, b = 9.229
=====
x = 1.000, y = 19.910  x = 4.000, y = 43.547
E = 126.5672320        E = 466.5735925
wb = 15.910, bb = 15.910  wb = 122.190, bb = 30.547
w = 9.861, b = 9.731  w = 7.358, b = 8.924
=====

```

가중치, 편향 학습과정 살펴보기

가중치와 편향 값만 확인해 봅니다.

1. 다음과 같이 예제를 수정합니다.


223_1.py

```

01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 for n in range(6):
07
08     y = xs[n]*w + 1*b
09
10     t = ys[n]
11     E = (y-t)**2/2
12
13     yb = y - t
14     wb = yb*xs[n]
15     bb = yb*1
16
17     lr = 0.01
18     w = w - lr*wb
19     b = b - lr*bb
20     print('w = %6.3f, b = %6.3f'%(w, b))

```

20 : w, b에 대한 출력만 합니다. 나머지 출력 루틴은 주석처리하거나 지워줍니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 10.020, b = 9.980
w = 10.020, b = 9.890
w = 9.861, b = 9.731
w = 9.412, b = 9.507
w = 8.580, b = 9.229
w = 7.358, b = 8.924

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다.

반복 학습 2회 수행하기

여기서는 반복 학습 2회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.


223_1.py

```

01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 for epoch in range(2):
07
08     for n in range(6):
09
10         y = xs[n]*w + 1*b
11
12         t = ys[n]
13         E = (y-t)**2/2
14
15         yb = y - t
16         wb = yb*xs[n]
17         bb = yb*1
18
19         lr = 0.01
20         w = w - lr*wb
21         b = b - lr*bb
22         print('w = %6.3f, b = %6.3f'%(w, b))

```

06 : epoch값을 0에서 2 미만까지 바꾸어가며 08~22줄을 2회 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 10.020, b = 9.980
w = 10.020, b = 9.890
w = 9.861, b = 9.731
w = 9.412, b = 9.507
w = 8.580, b = 9.229
w = 7.358, b = 8.924
w = 7.393, b = 8.888
w = 7.393, b = 8.809
w = 7.271, b = 8.687
w = 6.947, b = 8.525
w = 6.366, b = 8.331
w = 5.534, b = 8.123

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다.

반복 학습 20회 수행하기

여기서는 반복 학습 20회를 수행해 봅니다.


1. 다음과 같이 예제를 수정합니다.

223_1.py

```
01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 for epoch in range(20):
07
08     for n in range(6):
09
10         y = xs[n]*w + 1*b
11
12         t = ys[n]
13         E = (y-t)**2/2
14
15         yb = y - t
16         wb = yb*xs[n]
17         bb = yb*1
18
19         lr = 0.01
20         w = w - lr*wb
21         b = b - lr*bb
22         if epoch%2==1 and n==0 :
23             print('w = %6.3f, b = %6.3f'%(w, b))
```

06 : epoch값을 0에서 20 미만까지 바꾸어가며 08~23줄을 20회 수행합니다.

22 : epoch값을 2로 나눈 나머지가 1이고 n값이 0일 때 23줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.


```
w = 7.393, b = 8.888
w = 4.332, b = 7.464
w = 2.897, b = 6.614
w = 2.247, b = 6.051
w = 1.974, b = 5.636
w = 1.882, b = 5.303
w = 1.875, b = 5.016
w = 1.907, b = 4.760
w = 1.956, b = 4.526
w = 2.011, b = 4.309
```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다.

반복 학습 200회 수행하기

여기서는 반복 학습 200회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

223_1.py

```
01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 for epoch in range(200):
07
08     for n in range(6):
09
10         y = xs[n]*w + 1*b
11
12         t = ys[n]
13         E = (y-t)**2/2
14
15         yb = y - t
16         wb = yb*xs[n]
17         bb = yb*1
18
```


```

19         lr = 0.01
20         w = w - lr*wb
21         b = b - lr*bb
22         if epoch%20==1 and n==0 :
23             print('w = %6.3f, b = %6.3f'%(w, b))

```

07 : epoch값을 0에서 200 미만까지 바꾸어가며 08~23줄을 200회 수행합니다.

22 : epoch값을 20으로 나눈 나머지가 1이고 n값이 0일 때 23줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 7.393, b = 8.888
w = 2.067, b = 4.107
w = 2.499, b = 2.660
w = 2.732, b = 1.887
w = 2.857, b = 1.474
w = 2.923, b = 1.253
w = 2.959, b = 1.136
w = 2.978, b = 1.072
w = 2.988, b = 1.039
w = 2.994, b = 1.021

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다. w값은 3에, b값은 1에 가까워지는 것을 확인합니다.

반복 학습 2000회 수행하기

여기서는 반복 학습 2000회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

223_1.py

```

01 xs = [-1., 0., 1., 2., 3., 4.]
02 ys = [-2., 1., 4., 7., 10., 13.]
03 w = 10.
04 b = 10.
05
06 for epoch in range(2000):

```


```

07
08     for n in range(6):
09
10         y = xs[n]*w + 1*b
11
12         t = ys[n]
13         E = (y-t)**2/2
14
15         yb = y - t
16         wb = yb*xs[n]
17         bb = yb*1
18
19         lr = 0.01
20         w = w - lr*wb
21         b = b - lr*bb
22         if epoch%200==1 and n==0 :
23             print('w = %6.3f, b = %6.3f'%(w, b))

```

07 : epoch값을 0에서 2000 미만까지 바꾸어가며 08~23줄을 2000회 수행합니다.

22 : epoch값을 200으로 나눈 나머지가 1이고 n값이 0일 때 23줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 7.393, b = 8.888
w = 2.997, b = 1.011
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다. w값은 3에 b값은 1에 수렴하는 것을 확인합니다.

가중치, 편향 바꿔보기 1


여기서는 가중치와 편향 값을 바꾸어 실습을 진행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
03 w = -10.  
04 b = 10.
```

03 : 가중치 w값을 -10.으로 바꿉니다.

04 : 편향 b값은 10.으로 둡니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
w = -6.877, b = 10.358  
w = 2.993, b = 1.022  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000  
w = 3.000, b = 1.000
```

w값은 3에 b값은 1에 수렴하는 것을 확인합니다.


가중치, 편향 바꿔보기 2

1. 다음과 같이 예제를 수정합니다.

```
03 w = -100.  
04 b = 200.
```

03 : 가중치 w값을 -100.으로 바꿉니다.

04 : 편향 b값은 200.으로 바꿉니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
w = -83.789, b = 194.356
w = 2.884, b = 1.385
w = 3.000, b = 1.001
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
```

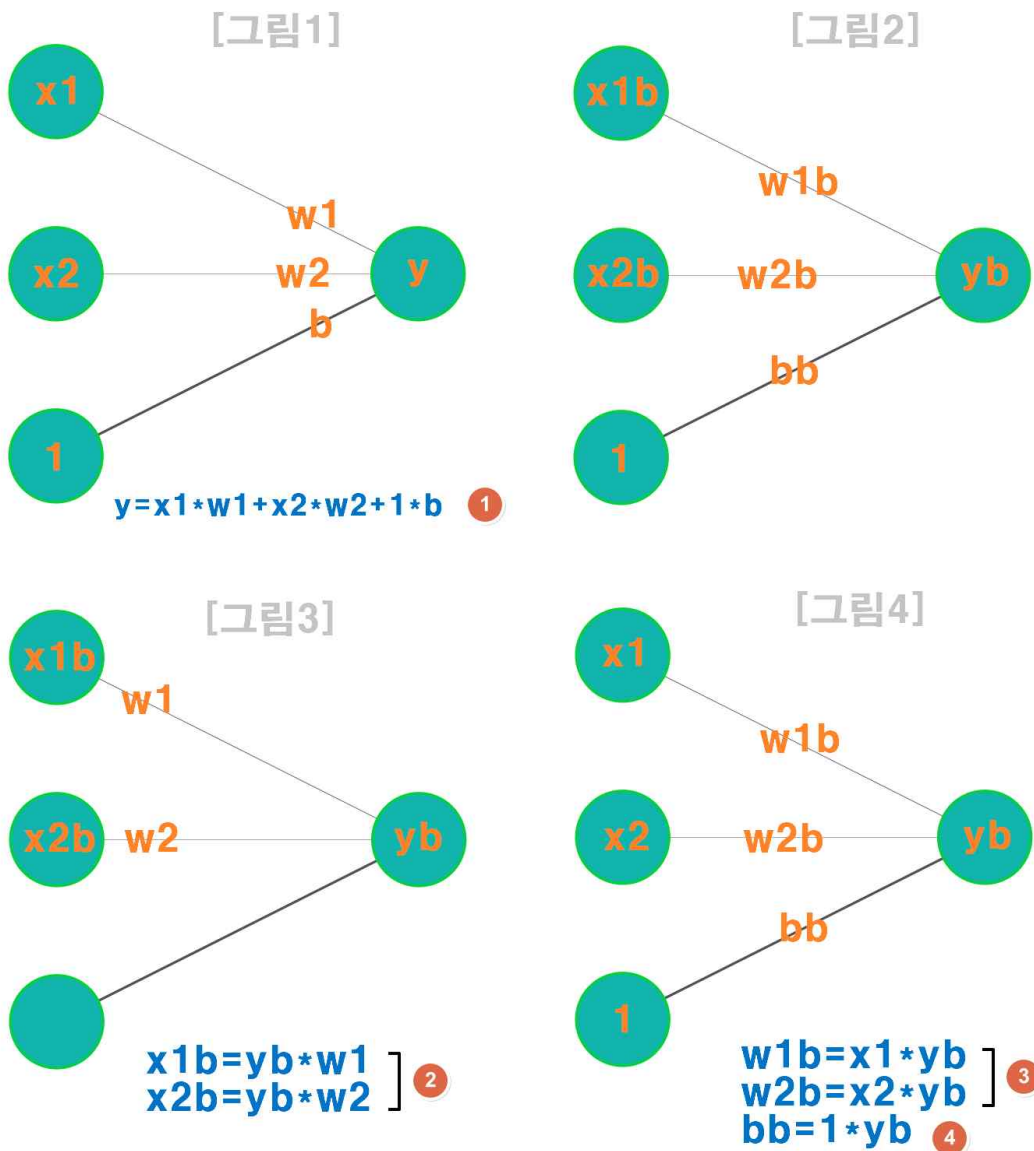
w값은 3에 b값은 1에 수렴하는 것을 확인합니다.

03 다양한 인공 신경망 구현해 보기

우리는 앞에서 입력1 출력1로 구성된 인공 신경의 동작을 살펴보고 구현해 보았습니다. 여기서는 입력2 출력1의 단일 인공 신경과, 입력2 출력2, 입력3 출력3으로 구성된 인공 신경망의 구조를 살펴보고 수식을 세운 후, 해당 수식에 맞는 인공 신경망을 구현해 봅니다. 출력의 개수 2이상은 인공 신경망이 됩니다.

01 2입력 1출력 인공 신경 구현하기

다음 그림은 입력2 출력1로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② $x1b$, $x2b$ 값은 앞부분에 또 다른 인공 신경과 연결되어 있을 경우 yb 처럼 해당 인공 신경으로 역전파되는 값입니다. 역전파된 $x1b$, $x2b$ 값은 해당 인공 신경의 가중치와 편향 학습에 사용됩니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$x_1w_1 + x_2w_2 + 1b = y \quad \textcircled{1}$$

입력 역전파

$$\left. \begin{array}{l} y_b w_1 = x_{1b} \\ y_b w_2 = x_{2b} \end{array} \right\} \quad \textcircled{2}$$

가중치, 편향 역전파

$$\left. \begin{array}{l} x_1 y_b = w_{1b} \\ x_2 y_b = w_{2b} \end{array} \right\} \quad \textcircled{3}$$

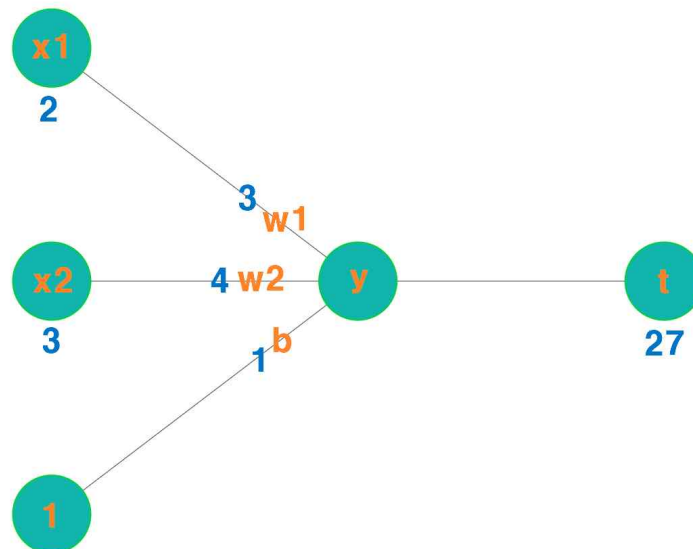
$$1y_b = b_b \quad \textcircled{4}$$

인공 신경망 학습

$$\left. \begin{array}{l} w_1 = w_1 - \alpha w_{1b} \\ w_2 = w_2 - \alpha w_{2b} \end{array} \right\} \quad \textcircled{5}$$

$$b = b - \alpha b_b \quad \textcircled{6}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력 값 x_1 , x_2 는 각각 2, 3, 가중치 w_1 , w_2 는 각각 3, 4, 편향 b 는 1이고 목표 값 t 는 27입니다. x_1 , x_2 를 상수로 고정하고 w_1 , w_2 , b 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.


1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

231_1.py

```
01 x1, x2 = 2, 3
02 t = 27
03 w1 = 3
04 w2 = 4
05 b = 1
06
07 for epoch in range(2000):
08
09     print('epoch = %d' %epoch)
10
11     y = x1*w1 + x2*w2 + 1*b # ❶
12     print(' y = %6.3f' %y)
13
14     E = (y-t)**2/2
15     print(' E = %.7f' %E)
16     if E < 0.0000001:
17         break
18
19     yb = y - t
20     x1b, x2b = yb*w1, yb*w2 # ❷
21     w1b = yb*x1 # ❸
22     w2b = yb*x2 # ❸
23     bb = yb*1 # ❹
24     print(' x1b, x2b = %6.3f, %6.3f'%(x1b, x2b))
25     print(' w1b, w2b, bb = %6.3f, %6.3f, %6.3f'%(w1b, w2b, bb))
26
27     lr = 0.01
28     w1 = w1 - lr*w1b # ❺
29     w2 = w2 - lr*w2b # ❺
30     b = b - lr*bb # ❻
31     print(' w1, w2, b = %6.3f, %6.3f, %6.3f'%(w1, w2, b))
```

20 : x1b, x2b 변수를 선언한 후, 입력 값에 대한 역전파 값을 받아봅니다. 이 부분은 이 예제에서 필요한 부분은 아니며, 역전파 연습을 위해 추가하였습니다.

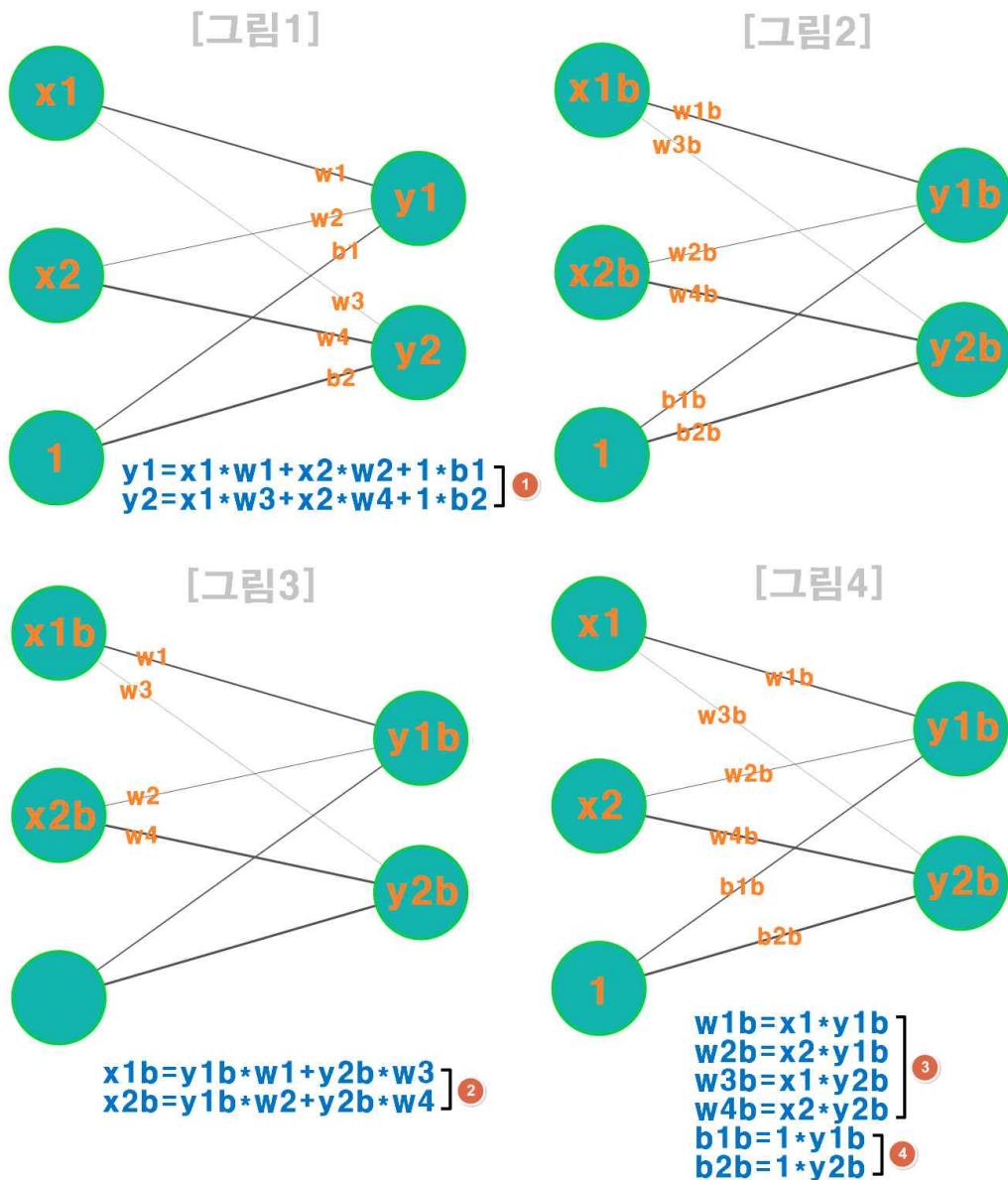
3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 64
y  = 26.999
E  = 0.0000001
x1b, x2b = -0.002, -0.003
w1b, w2b, bb = -0.001, -0.002, -0.001
w1, w2, b = 4.143, 5.714, 1.571
epoch = 65
y  = 27.000
E  = 0.0000001
```

65(64+1)회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 w_1 , w_2 는 각각 4.143, 5.714, 편향 b 는 1.571에 수렴합니다.

02 2입력 2출력 인공 신경망 구현하기

다음 그림은 입력2 출력2로 구성된 인공 신경망을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② x_{1b}, x_{2b} 값은 앞부분에 또 다른 인공 신경과 연결되어 있을 경우 y_{1b}, y_{2b} 처럼 해당 인공 신경으로 역전파되는 값입니다. 역전파된 x_{1b}, x_{2b} 값은 해당 인공 신경의 가중치와 편향 학습에 사용됩니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$\begin{bmatrix} x_1w_1 + x_2w_2 + 1b_1 = y_1 \\ x_1w_3 + x_2w_4 + 1b_2 = y_2 \end{bmatrix} \quad 1$$

입력 역전파

$$\begin{bmatrix} y_{1b}w_1 + y_{2b}w_3 = x_{1b} \\ y_{1b}w_2 + y_{2b}w_4 = x_{2b} \end{bmatrix} \quad 2$$

가중치, 편향 역전파

$$\begin{bmatrix} x_1y_{1b} = w_{1b} \\ x_2y_{1b} = w_{2b} \\ x_1y_{2b} = w_{3b} \\ x_2y_{2b} = w_{4b} \end{bmatrix} \quad 3$$

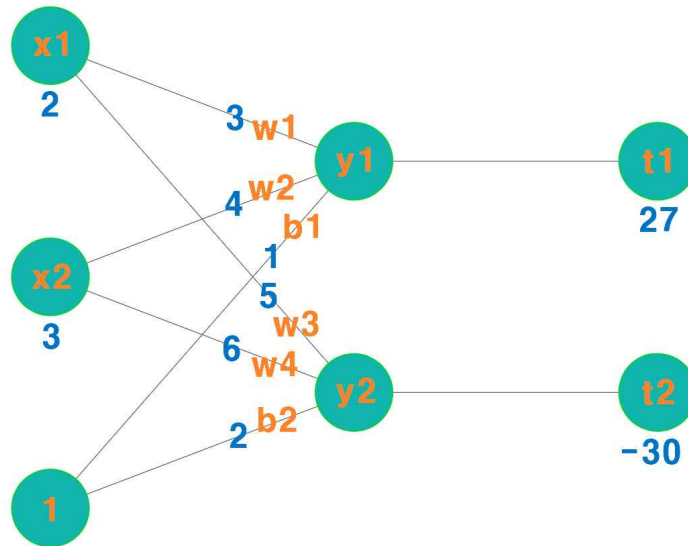
$$\begin{bmatrix} 1y_{1b} = b_{1b} \\ 1y_{2b} = b_{2b} \end{bmatrix} \quad 4$$

인공 신경망 학습

$$\begin{bmatrix} w_1 = w_1 - \alpha w_{1b} \\ w_2 = w_2 - \alpha w_{2b} \\ w_3 = w_3 - \alpha w_{3b} \\ w_4 = w_4 - \alpha w_{4b} \end{bmatrix} \quad 5$$

$$\begin{bmatrix} b_1 = b_1 - \alpha b_{1b} \\ b_2 = b_2 - \alpha b_{2b} \end{bmatrix} \quad 6$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력 값 x_1 , x_2 는 각각 2, 3, 가중치 w_1 , w_2 , 편향 b_1 은 각각 3, 4, 1, 가중치 w_3 , w_4 , 편향 b_2 는 각각 5, 6, 2이고 목표 값 t_1 , t_2 는 각각 27, -30입니다. x_1 , x_2 를 상수로 고정한 채 w_1 , w_2 , w_3 , w_4 , b_1 , b_2 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 이전 예제를 복사합니다.
 2. 다음과 같이 예제를 수정합니다.
- 232_1.py

```

01 x1, x2 = 2, 3
02 t1, t2 = 27, -30
03 w1, w3 = 3, 5
04 w2, w4 = 4, 6
05 b1, b2 = 1, 2
06
07 for epoch in range(2000):
08
09     print('epoch = %d' %epoch)
10
11     y1 = x1*w1 + x2*w2 + 1*b1 # ❶
12     y2 = x1*w3 + x2*w4 + 1*b2 # ❶
13     print(' y1, y2 = %6.3f, %6.3f' %(y1, y2))
14
15     E = (y1-t1)**2/2 + (y2-t2)**2/2


```

```

16     print(' E = %.7f' %E)
17     if E < 0.0000001:
18         break
19
20     y1b, y2b = y1 - t1, y2 - t2
21     x1b, x2b = y1b*w1+y2b*w3, y1b*w2+y2b*w4 # ②
22     w1b, w3b = x1*y1b, x1*y2b # ③
23     w2b, w4b = x2*y1b, x2*y2b # ③
24     b1b, b2b = 1*y1b, 1*y2b # ④
25     print(' x1b, x2b = %6.3f, %6.3f'%(x1b, x2b))
26     print(' w1b, w3b = %6.3f, %6.3f'%(w1b, w3b))
27     print(' w2b, w4b = %6.3f, %6.3f'%(w2b, w4b))
28     print(' b1b, b2b = %6.3f, %6.3f'%(b1b, b2b))
29
30     lr = 0.01
31     w1, w3 = w1 - lr*w1b, w3 - lr*w3b # ⑤
32     w2, w4 = w2 - lr*w2b, w4 - lr*w4b # ⑤
33     b1, b2 = b1 - lr*b1b, b2 - lr*b2b # ⑥
34     print(' w1, w3 = %6.3f, %6.3f'%(w1, w3))
35     print(' w2, w4 = %6.3f, %6.3f'%(w2, w4))
36     print(' b1, b2 = %6.3f, %6.3f'%(b1, b2))

```

21 : x1b, x2b 변수를 선언한 후, 입력 값에 대한 역전파 값을 받아옵니다. 이 부분은 이 예제에서 필요한 부분은 아니며, 역전파 연습을 위해 추가하였습니다.

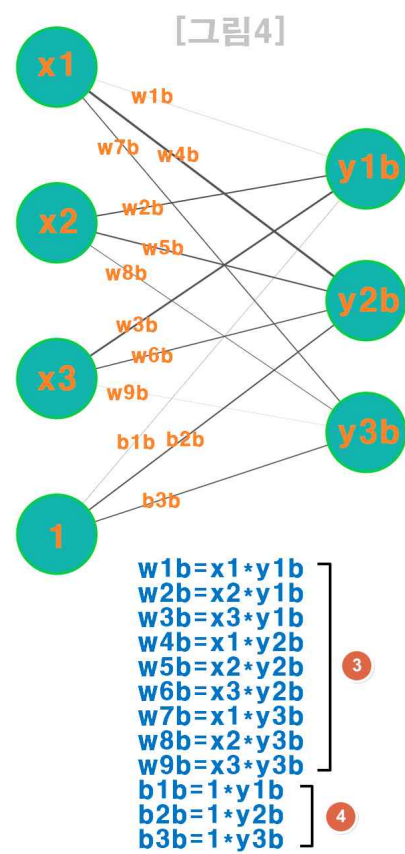
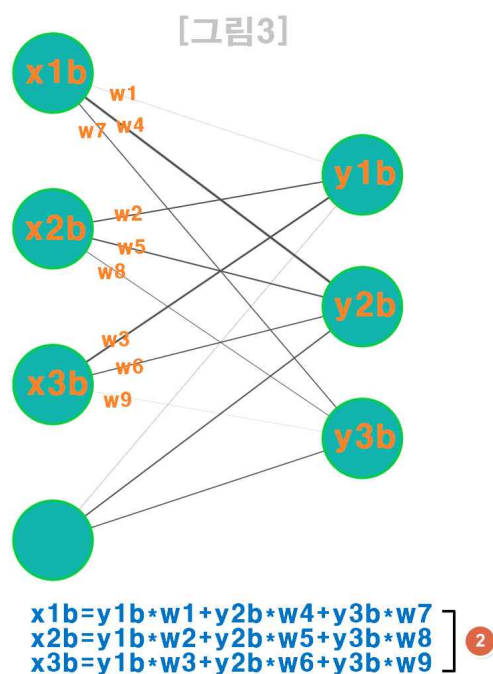
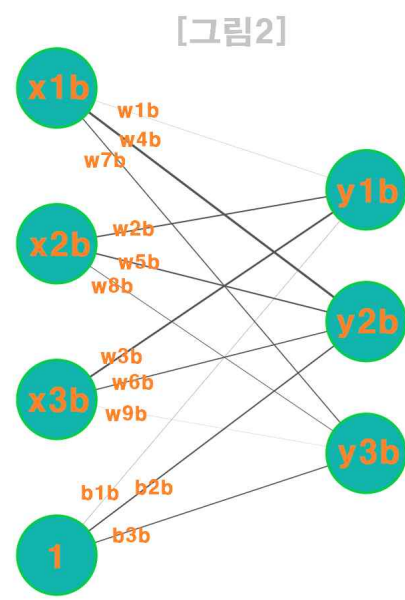
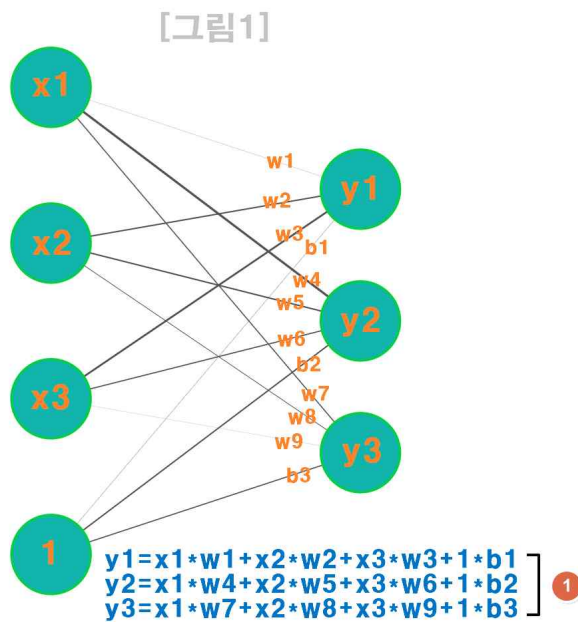
3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 78
y1, y2 = 27.000, -30.000
E = 0.0000001
x1b, x2b = -0.002, -0.004
w1b, w3b = -0.000, 0.001
w2b, w4b = -0.000, 0.001
b1b, b2b = -0.000, 0.000
w1, w3 = 4.143, -3.571
w2, w4 = 5.714, -6.857
b1, b2 = 1.571, -2.286
epoch = 79
y1, y2 = 27.000, -30.000
E = 0.0000001
```

79회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 $w1$, $w2$ 는 각각 4.143, 5.714, 편향 $b1$ 은 1.571, 가중치 $w3$, $w4$ 는 각각 -3.571, -6.857 편향 $b2$ 는 -2.286에 수렴합니다.

03 3입력 3출력 인공 신경망 구현하기

다음 그림은 입력3 출력3으로 구성된 인공 신경망을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② x_{1b} , x_{2b} , x_{3b} 값은 앞부분에 또 다른 인공 신경과 연결되어 있을 경우 y_{1b} , y_{2b} , y_{3b} 처럼 해당 인공 신경으로 역전파되는 값입니다. 역전파된 x_{1b} , x_{2b} , x_{3b} 값은 해당 인공 신경의 가중치와 편향 학습에 사용됩니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$\left. \begin{aligned} x_1w_1 + x_2w_2 + x_3w_3 + 1b_1 &= y_1 \\ x_1w_4 + x_2w_5 + x_3w_6 + 1b_2 &= y_2 \\ x_1w_7 + x_2w_8 + x_3w_9 + 1b_3 &= y_3 \end{aligned} \right\} \textcircled{1}$$

입력 역전파

$$\left. \begin{aligned} y_{1b}w_1 + y_{2b}w_4 + y_{3b}w_7 &= x_{1b} \\ y_{1b}w_2 + y_{2b}w_5 + y_{3b}w_8 &= x_{2b} \\ y_{1b}w_3 + y_{2b}w_6 + y_{3b}w_9 &= x_{3b} \end{aligned} \right\} \textcircled{2}$$

가중치, 편향 역전파

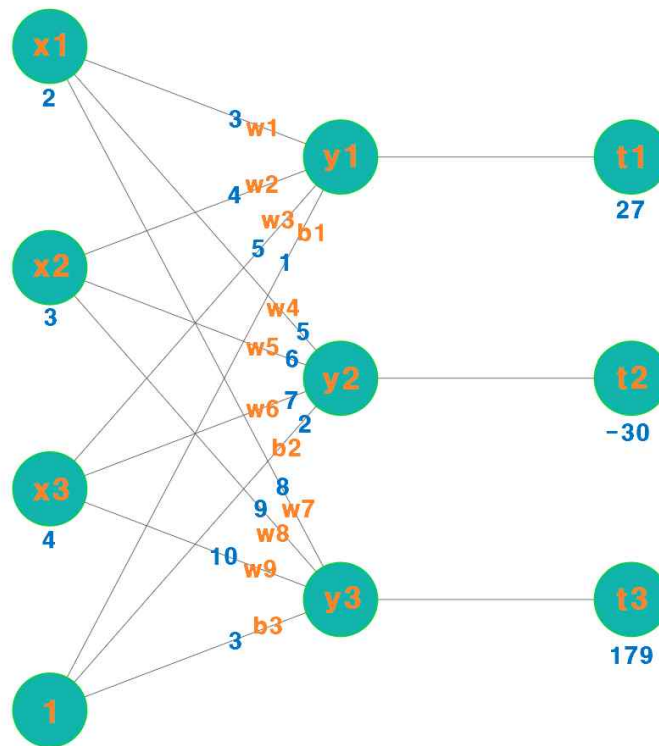
$$\left. \begin{aligned} x_1y_{1b} &= w_{1b} \\ x_2y_{1b} &= w_{2b} \\ x_3y_{1b} &= w_{3b} \\ x_1y_{2b} &= w_{4b} \\ x_2y_{2b} &= w_{5b} \\ x_3y_{2b} &= w_{6b} \\ x_1y_{3b} &= w_{7b} \\ x_2y_{3b} &= w_{8b} \\ x_3y_{3b} &= w_{9b} \end{aligned} \right\} \textcircled{3}$$

$$\left. \begin{aligned} 1y_{1b} &= b_{1b} \\ 1y_{2b} &= b_{2b} \\ 1y_{3b} &= b_{3b} \end{aligned} \right\} \textcircled{4}$$

인공 신경망 학습

$$\begin{aligned}
 w_1 &= w_1 - \alpha w_{1b} \\
 w_2 &= w_2 - \alpha w_{2b} \\
 w_3 &= w_3 - \alpha w_{3b} \\
 w_4 &= w_4 - \alpha w_{4b} \\
 w_5 &= w_5 - \alpha w_{5b} \\
 w_6 &= w_6 - \alpha w_{6b} \\
 w_7 &= w_7 - \alpha w_{7b} \\
 w_8 &= w_8 - \alpha w_{8b} \\
 w_9 &= w_9 - \alpha w_{9b} \\
 b_1 &= b_1 - \alpha b_{1b} \\
 b_2 &= b_2 - \alpha b_{2b} \\
 b_3 &= b_3 - \alpha b_{3b}
 \end{aligned}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력 값 x1, x2, x3은 각각 2, 3, 4, 가중치 w1, w2, w3, 편향 b1은 각각 3, 4, 5, 1, 가중치 w4, w5, w6, 편향 b2는 각각 5, 6, 7, 2, 가중치 w7, w8, w9, 편향 b3은 각각 8, 9, 10, 3이고 목표 값 t1, t2, t3은 각각 27, -30, 179입니다. x1, x2, x3를 상수로 고정한 채 w1~w9, b1~b3에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

233_1.py


```
01 x1, x2, x3 = 2, 3, 4
02 t1, t2, t3 = 27, -30, 179
03 w1, w4, w7 = 3, 5, 8
04 w2, w5, w8 = 4, 6, 9
05 w3, w6, w9 = 5, 7, 10
06 b1, b2, b3 = 1, 2, 3
07
08 for epoch in range(2000):
09
10     print('epoch = %d' %epoch)
11
12     y1 = x1*w1 + x2*w2 + x3*w3 + 1*b1 # ❶
13     y2 = x1*w4 + x2*w5 + x3*w6 + 1*b2 # ❶
14     y3 = x1*w7 + x2*w8 + x3*w9 + 1*b3 # ❶
15     print(' y1, y2, y3 = %6.3f, %6.3f, %6.3f' %(y1, y2, y3))
16
17     E = (y1-t1)**2/2 + (y2-t2)**2/2+ (y3-t3)**2/2
18     print(' E = %.7f' %E)
19     if E < 0.0000001:
20         break
21
22     y1b, y2b, y3b = y1 - t1, y2 - t2, y3 - t3
23     x1b = y1b*w1+y2b*w4+y3b*w7 # ❷
24     x2b = y1b*w2+y2b*w5+y3b*w8 # ❷
25     x3b = y1b*w3+y2b*w6+y3b*w9 # ❷
26     w1b, w4b, w7b = x1*y1b, x1*y2b, x1*y3b # ❸
27     w2b, w5b, w8b = x2*y1b, x2*y2b, x2*y3b # ❸
28     w3b, w6b, w9b = x3*y1b, x3*y2b, x3*y3b # ❸
29     b1b, b2b, b3b = 1*y1b, 1*y2b, 1*y3b # ❹
30     print(' x1b, x2b, x3b = %6.3f, %6.3f, %6.3f'%(x1b, x2b, x3b))
31     print(' w1b, w4b, w7b = %6.3f, %6.3f, %6.3f'%(w1b, w4b, w7b))
32     print(' w2b, w5b, w8b = %6.3f, %6.3f, %6.3f'%(w2b, w5b, w8b))
33     print(' w3b, w6b, w9b = %6.3f, %6.3f, %6.3f'%(w3b, w6b, w9b))
34     print(' b1b, b2b, b3b = %6.3f, %6.3f, %6.3f'%(b1b, b2b, b3b))
```

```

35
36     lr = 0.01
37     w1, w4, w7 = w1 - lr*w1b, w4 - lr*w4b, w7 - lr*w7b # ⑤
38     w2, w5, w8 = w2 - lr*w2b, w5 - lr*w5b, w8 - lr*w8b # ⑤
39     w3, w6, w9 = w3 - lr*w3b, w6 - lr*w6b, w9 - lr*w9b # ⑤
40     b1, b2, b3 = b1 - lr*b1b, b2 - lr*b2b, b3 - lr*b3b # ⑥
41     print(' w1, w4, w7 = %6.3f, %6.3f, %6.3f'%(w1, w4, w7))
42     print(' w2, w5, w8 = %6.3f, %6.3f, %6.3f'%(w2, w5, w8))
43     print(' w3, w6, w9 = %6.3f, %6.3f, %6.3f'%(w3, w6, w9))
44     print(' b1, b2, b3 = %6.3f, %6.3f, %6.3f'%(b1, b2, b3))

```

23~25 : x1b, x2b, x3b 변수를 선언한 후, 입력 값에 대한 역전파 값을 받아옵니다. 이 부분은 이 예제에서 필요한 부분은 아니며, 역전파 연습을 위해 추가하였습니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 35
y1, y2, y3 = 27.000, -30.000, 179.000
E = 0.00000001
x1b, x2b, x3b = -0.005, -0.007, -0.009
w1b, w4b, w7b = 0.000, 0.001, -0.001
w2b, w5b, w8b = 0.000, 0.001, -0.001
w3b, w6b, w9b = 0.000, 0.001, -0.001
b1b, b2b, b3b = 0.000, 0.000, -0.000
w1, w4, w7 = 2.200, -0.867, 14.200
w2, w5, w8 = 2.800, -2.800, 18.300
w3, w6, w9 = 3.400, -4.733, 22.400
b1, b2, b3 = 0.600, -0.933, 6.100
epoch = 36
y1, y2, y3 = 27.000, -30.000, 179.000
E = 0.00000001

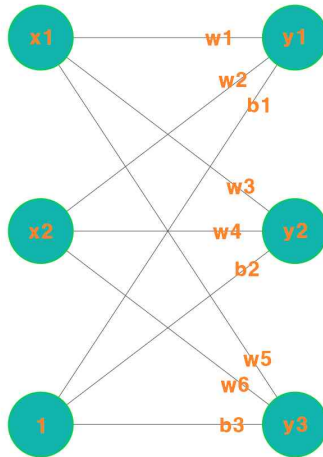
```

37(36+1)회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 w1~w9, 편향 b1~b3이 수렴하는 값도 살펴봅니다.

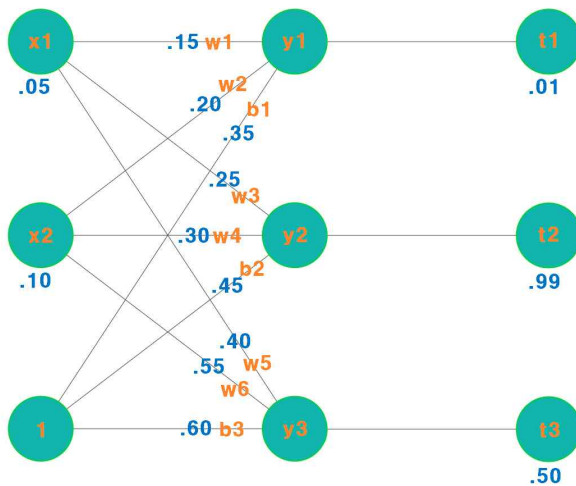
연습문제

❶ 2입력 3출력

1. 다음은 입력2 출력3의 인공 신경망입니다. 이 인공 신경망의 순전파, 역전파 수식을 구합니다.

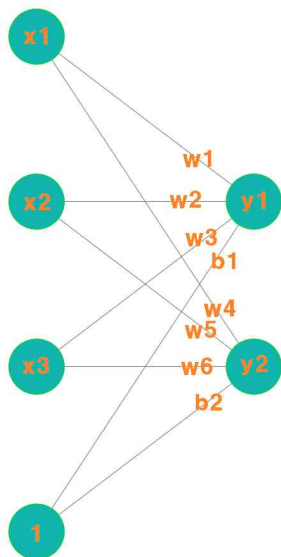


2. 앞에서 구한 수식을 이용하여 다음과 같이 초기화된 인공 신경망을 구현하고 학습시켜 봅니다. x_1 , x_2 는 입력 층으로 상수로 처리합니다.

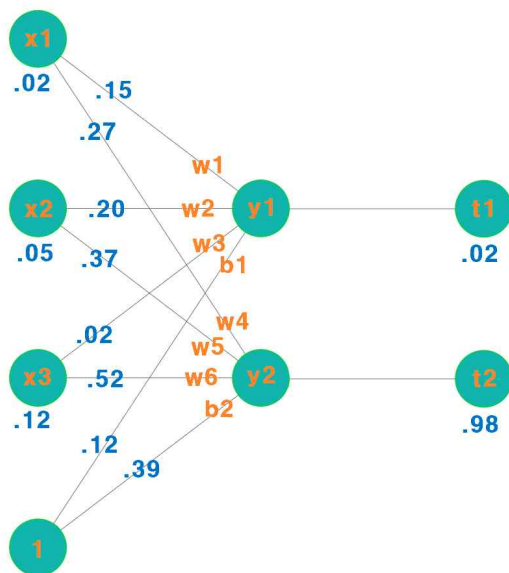


❷ 3입력 2출력

1. 다음은 입력3 출력2의 인공 신경망입니다. 이 인공 신경망의 순전파, 역전파 수식을 구합니다.



2. 앞에서 구한 수식을 이용하여 다음과 같이 초기화된 인공 신경망을 구현하고 학습시켜 봅니다. x1, x2, x3은 입력 층으로 상수로 처리합니다.

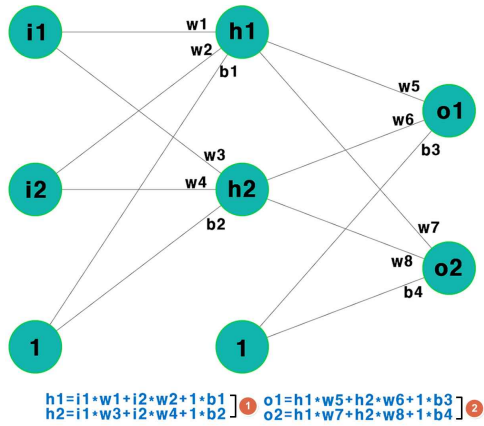


04 2입력 2은닉 2출력 인공 신경망 구현하기

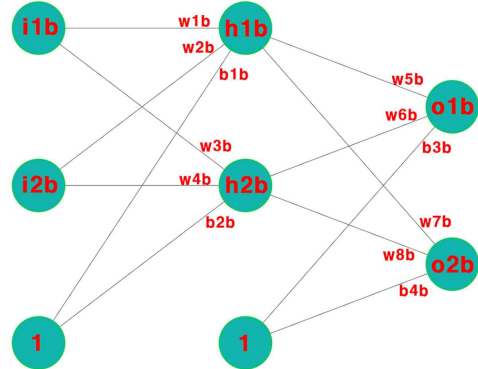
여기서는 은닉 신경을 추가한 인공 신경망을 구현해 봅니다. 은닉 신경이 추가된 경우에도 순전파, 역전파 수식을 구하는 방식은 이전과 같습니다.

다음 그림은 입력2 은닉2 출력2로 구성된 인공 신경망을 나타냅니다.

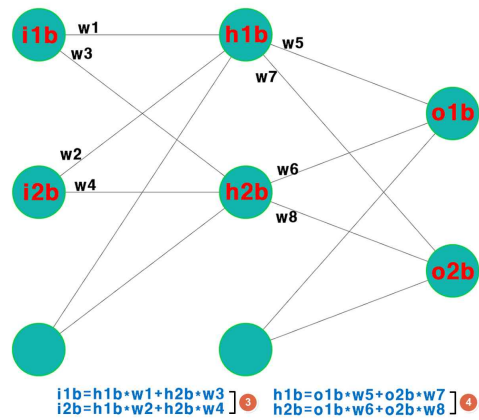
[그림1]



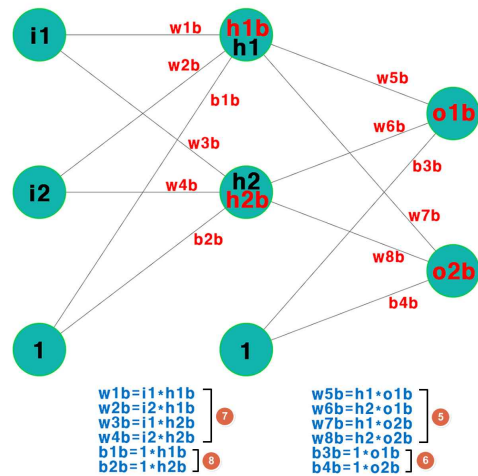
[그림2]



[그림3]



[그림4]



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ③ i1b, i2b값은 앞부분에 또 다른 인공 신경과 연결되어 있을 경우 h1b, h2b처럼 해당 인공 신경으로 역전파되는 값입니다. 역전파된 i1b, i2b값은 해당 인공 신경의 가중치와 편향 학습에 사용됩니다. 여기서 i1, i2는 은닉 층에 연결된 입력 층이므로 i1b, i2b의 수식은 필요치 않습니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$\left. \begin{aligned} i_1 w_1 + i_2 w_2 + 1b_1 &= h_1 \\ i_1 w_3 + i_2 w_4 + 1b_2 &= h_2 \end{aligned} \right] \textcircled{1}$$

$$\left. \begin{aligned} h_1 w_5 + h_2 w_6 + 1b_3 &= o_1 \\ h_1 w_7 + h_2 w_8 + 1b_4 &= o_2 \end{aligned} \right] \textcircled{2}$$

입력 역전파

$$\left. \begin{aligned} o_{1b} w_5 + o_{2b} w_7 &= h_{1b} \\ o_{1b} w_6 + o_{2b} w_8 &= h_{2b} \end{aligned} \right] \textcircled{4}$$

가중치, 편향 역전파

$$\left. \begin{aligned} i_1 h_{1b} &= w_{1b} \\ i_2 h_{1b} &= w_{2b} \\ i_1 h_{2b} &= w_{3b} \\ i_2 h_{2b} &= w_{4b} \end{aligned} \right] \textcircled{7}$$

$$\left. \begin{aligned} 1h_{1b} &= b_{1b} \\ 1h_{2b} &= b_{2b} \end{aligned} \right] \textcircled{8}$$

$$\left. \begin{aligned} h_1 o_{1b} &= w_{5b} \\ h_2 o_{1b} &= w_{6b} \\ h_1 o_{2b} &= w_{7b} \\ h_2 o_{2b} &= w_{8b} \end{aligned} \right] \textcircled{5}$$

$$\left. \begin{aligned} 1o_{1b} &= b_{3b} \\ 1o_{2b} &= b_{4b} \end{aligned} \right] \textcircled{6}$$

인공 신경망 학습

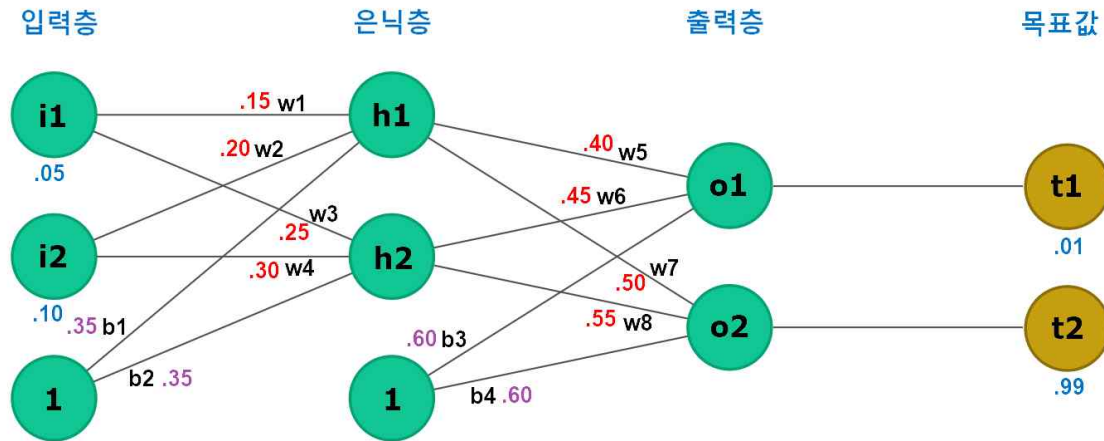
$$\left. \begin{aligned} w_1 &= w_1 - \alpha w_{1b} \\ w_2 &= w_2 - \alpha w_{2b} \\ w_3 &= w_3 - \alpha w_{3b} \\ w_4 &= w_4 - \alpha w_{4b} \end{aligned} \right] \textcircled{11}$$

$$\left. \begin{aligned} b_1 &= b_1 - \alpha b_{1b} \\ b_2 &= b_2 - \alpha b_{2b} \end{aligned} \right] \textcircled{12}$$

$$\left. \begin{aligned} w_5 &= w_5 - \alpha w_{5b} \\ w_6 &= w_6 - \alpha w_{6b} \\ w_7 &= w_7 - \alpha w_{7b} \\ w_8 &= w_8 - \alpha w_{8b} \end{aligned} \right] \textcircled{9}$$

$$\left. \begin{aligned} b_3 &= b_3 - \alpha b_{3b} \\ b_4 &= b_4 - \alpha b_{4b} \end{aligned} \right] \textcircled{10}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



입력 값, 가중치 값, 편향 값은 그림을 참조합니다. i1, i2를 상수로 고정한 채 w1~w8, b1~b4에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.

234_1.py

```

01 i1, i2 = .05, .10
02 t1, t2 = .01, .99
03
04 w1, w3 = .15, .25
05 w2, w4 = .20, .30
06 b1, b2 = .35, .35
07
08 w5, w7 = .40, .50
09 w6, w8 = .45, .55
10 b3, b4 = .60, .60
11
12 for epoch in range(2000):
13
14     print('epoch = %d' %epoch)
15
16     h1 = i1*w1 + i2*w2 + 1*b1 # ❶
17     h2 = i1*w3 + i2*w4 + 1*b2 # ❶
18     o1 = h1*w5 + h2*w6 + 1*b3 # ❷
19     o2 = h1*w7 + h2*w8 + 1*b4 # ❷
20     print(' h1, h2 = %6.3f, %6.3f' %(h1, h2))


```



```

21     print(' o1, o2 = %6.3f, %6.3f' %(o1, o2))
22
23     E = (o1-t1)**2/2 + (o2-t2)**2/2
24     print(' E = %.7f' %E)
25     if E < 0.0000001:
26         break
27
28     o1b, o2b = o1 - t1, o2 - t2
29     h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8 # ④
30     w1b, w3b = i1*h1b, i1*h2b # ⑦
31     w2b, w4b = i2*h1b, i2*h2b # ⑦
32     b1b, b2b = 1*h1b, 1*h2b # ⑧
33     w5b, w7b = h1*o1b, h1*o2b # ⑤
34     w6b, w8b = h2*o1b, h2*o2b # ⑤
35     b3b, b4b = 1*o1b, 1*o2b # ⑥
36     print(' w1b, w3b = %6.3f, %6.3f'%(w1b, w3b))
37     print(' w2b, w4b = %6.3f, %6.3f'%(w2b, w4b))
38     print(' b1b, b2b = %6.3f, %6.3f'%(b1b, b2b))
39     print(' w5b, w7b = %6.3f, %6.3f'%(w5b, w7b))
40     print(' w6b, w8b = %6.3f, %6.3f'%(w6b, w8b))
41     print(' b3b, b4b = %6.3f, %6.3f'%(b3b, b4b))
42
43     lr = 0.01
44     w1, w3 = w1 - lr*w1b, w3 - lr*w3b # ⑪
45     w2, w4 = w2 - lr*w2b, w4 - lr*w4b # ⑪
46     b1, b2 = b1 - lr*b1b, b2 - lr*b2b # ⑫
47     w5, w7 = w5 - lr*w5b, w7 - lr*w7b # ⑨
48     w6, w8 = w6 - lr*w6b, w8 - lr*w8b # ⑨
49     b3, b4 = b3 - lr*b3b, b4 - lr*b4b # ⑩
50     print(' w1, w3 = %6.3f, %6.3f'%(w1, w3))
51     print(' w2, w4 = %6.3f, %6.3f'%(w2, w4))
52     print(' b1, b2 = %6.3f, %6.3f'%(b1, b2))
53     print(' w5, w7 = %6.3f, %6.3f'%(w5, w7))
54     print(' w6, w8 = %6.3f, %6.3f'%(w6, w8))
55     print(' b3, b4 = %6.3f, %6.3f'%(b3, b4))

```

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

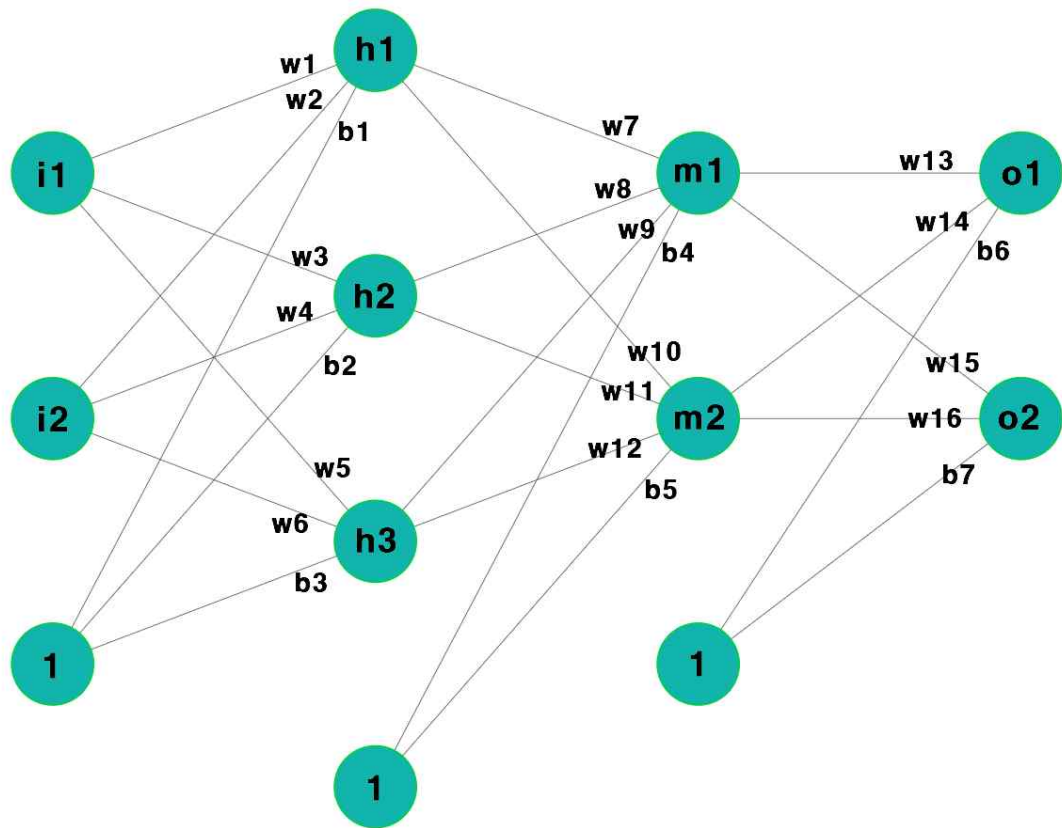
epoch = 664
h1, h2 = 0.239, 0.226
o1, o2 = 0.010, 0.990
E = 0.0000001
w1b, w3b = -0.000, 0.000
w2b, w4b = -0.000, 0.000
b1b, b2b = -0.000, 0.000
w5b, w7b = 0.000, -0.000
w6b, w8b = 0.000, -0.000
b3b, b4b = 0.000, -0.000
w1, w3 = 0.143, 0.242
w2, w4 = 0.186, 0.284
b1, b2 = 0.213, 0.186
w5, w7 = 0.203, 0.533
w6, w8 = 0.253, 0.583
b3, b4 = -0.095, 0.730
epoch = 665
h1, h2 = 0.239, 0.226
o1, o2 = 0.010, 0.990
E = 0.0000001

```

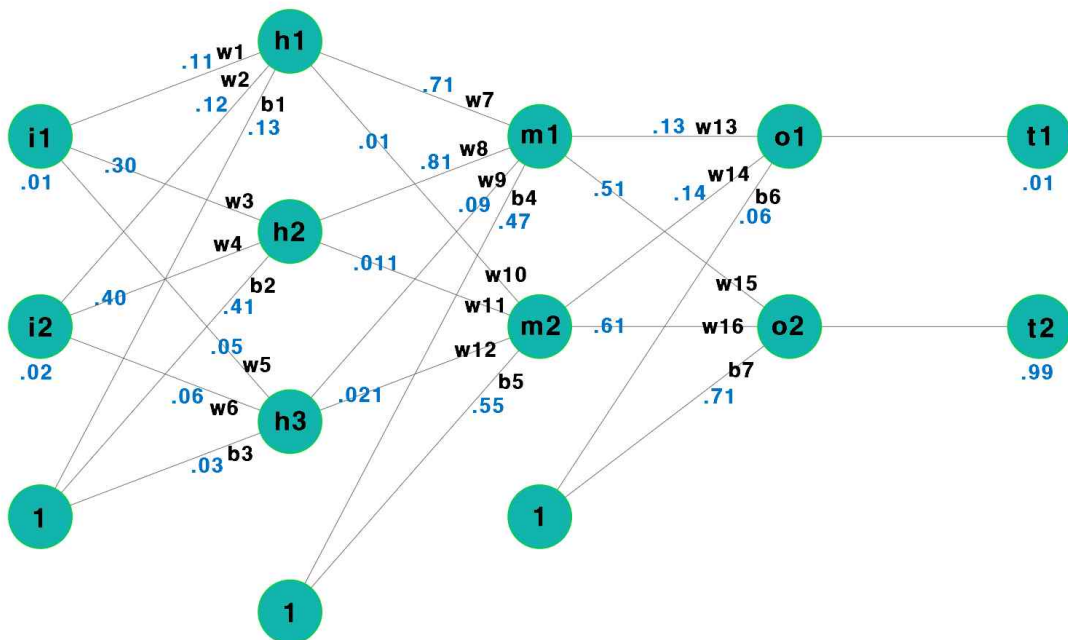
연습문제

2입력 2은닉 3은닉 2출력

1. 다음은 입력2 은닉3 은닉2 출력3의 심층 인공 신경망입니다. 이 신경망에는 2개의 은닉 층이 포함되어 있습니다. 일반적으로 은닉 층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 순전파, 역전파 수식을 구합니다.



2. 앞에서 구한 수식을 이용하여 다음과 같이 초기화된 인공 신경망을 구현하고 학습시켜 봅니다. $i1$, $i2$ 는 입력 층으로 상수로 처리합니다.



04 활성화 함수 추가하기

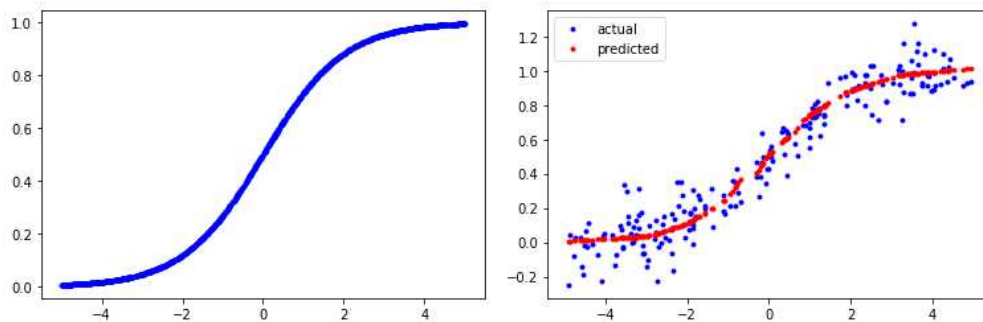
일반적으로 인공 신경의 출력단에는 활성화 함수가 추가됩니다. 활성화 함수를 추가하면 입력된 데이터에 대해 복잡한 패턴의 학습이 가능해집니다. 또 인공 신경의 출력 값을 어떤 범위로 제한할 수도 있습니다. 여기서는 주로 사용되는 몇 가지 활성화 함수를 살펴보고, 활성화 함수가 필요한 이유에 대해서도 살펴봅니다. 그리고 활성화 함수를 추가한 인공 신경망을 구현해 봅니다.

01 활성화 함수 살펴보기

우리는 앞에서 다음과 같은 활성화 함수를 직접 그려보았습니다.

sigmoid 함수

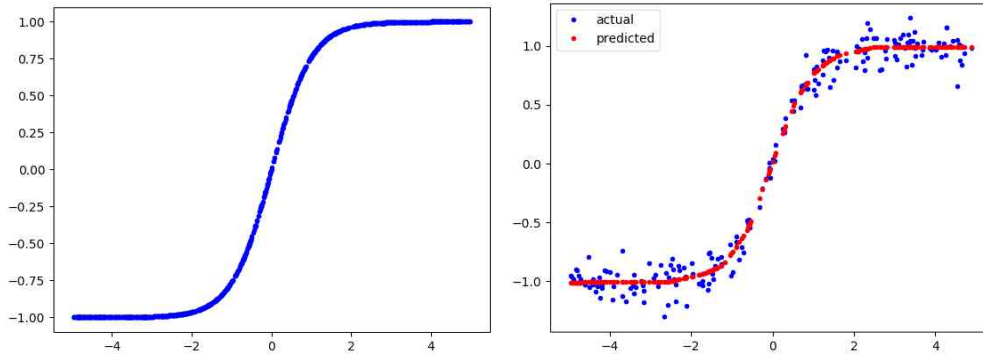
다음은 sigmoid 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



$$y = \frac{1}{1 + e^{-x}} \quad (-5 \leq x \leq 5)$$

tanh 함수

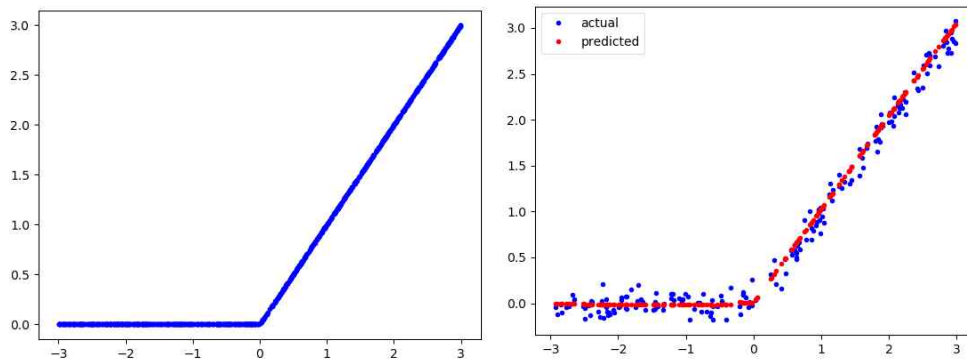
다음은 tanh 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



$$y = \tanh(x) \quad (-5 \leq x \leq 5)$$

ReLU 함수

다음은 ReLU 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \quad (-3 \leq x \leq 3)$$

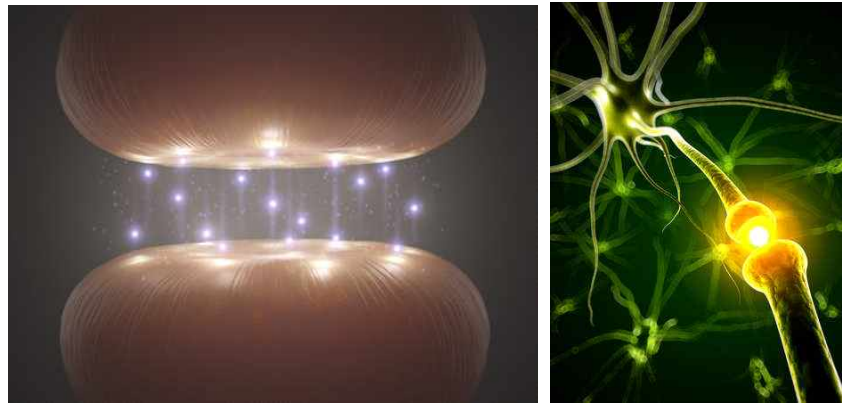
02 활성화 함수의 필요성

여기서는 활성화 함수가 무엇인지, 활성화 함수는 왜 필요한지, 어떤 활성화 함수가 있는지 살펴봅니다.

활성화 함수는 무엇인가요?

활성화 함수는 인공 신경망에 더해져 복잡한 패턴을 학습하게 해줍니다. 즉, 다양한 형태의 입력 값에 대해 신경망을 거쳐 나온 출력 값을 우리가 원하는 목표 값에 가깝게 해 주기가 더 쉬워집니다. 우리 두뇌에 있는 생체 신경과 비교할 때, 활성화 함수는 신경 말단에서 다음 신경으로 전달될 신호를 결정하는 시냅스와 같은 역할을 합니다. 시냅스는 이전 신경 세포가 내

보내는 출력 신호를 받아 다음 신경 세포가 받아들일 수 있는 입력 신호로 형태를 변경합니다. 마찬가지로 활성화 함수는 이전 인공 신경이 내보내는 출력 신호를 받아 다음 인공 신경이 받아들일 수 있는 입력 신호로 형태를 변경해 주는 역할을 합니다.



[시냅스]

활성화 함수는 왜 필요한가요?

앞에서 언급했던 생물학적 유사성과는 별도로 인공 신경의 출력 값을 우리가 원하는 어떤 범위로 제한해 줍니다. 이것은 활성화 함수로의 입력이 $w \cdot x + b$ 이기 때문입니다. 여기서 w 는 인공 신경의 가중치, x 는 입력, b 는 그것에 더해지는 편향입니다. 이 값은 어떤 범위로 제한되지 않으면 신경망을 거치며 순식간에 아주 커지게 됩니다. 특히 수백만 개의 매개변수(가중치와 편향)으로 구성된 아주 깊은 신경망의 경우에는 더욱 그렇습니다. 인공 신경을 거치며 반복적으로 계산되는 $w \cdot x + b$ 는 factorial 연산과 같은 효과를 내며 이것은 순식간에 컴퓨터의 계산 범위를 넘어서게 됩니다. 인공 신경망을 학습시키다보면 Nan이라고 표시되는 경우가 있는데 이 경우가 그런 경우에 해당합니다.

$n!$	$= n$
0	$= 1$
1	$= 1$
2	$= 2$
3	$= 6$
4	$= 24$
5	$= 120$
6	$= 720$
7	$= 5.040$
8	$= 40.320$
9	$= 362.880$
10	$= 3.628.800$
11	$= 39.916.800$
12	$= 479.001.600$

어떤 활성화 함수가 있나요?

❶ 시그모이드

시그모이드 활성화 함수는 단지 역사적인 이유로 여기에 소개되며 일반적으로 딥러닝에서 많이 사용되지 않습니다. 시그모이드 함수는 3층 정도로 구성된 인공 신경망에 적용될 때는 학습이 잘 되지만 깊은 신경망에 적용될 때는 학습이 잘 되지 않습니다. 시그모이드 함수는 계산에 시간이 걸리고, 입력 값이 아무리 크더라도 출력 값의 범위가 0에서 1사이로 매우 작아 신경망을 거칠수록 출력 값은 점점 더 작아져 0에 수렴하게 됩니다. 이것은 신경을 거치면서 신호가 점점 작아져 출력에 도달하는 신호가 아주 작거나 없어지는 것과 같습니다. 출력에 미치는 신호가 아주 작거나 없다는 것은 역으로 전달될 신호도 아주 작거나 없다는 것을 의미합니다. 시그모이드 함수는 일반적으로 0이나 1로 분류하는 이진 분류 문제에 사용됩니다. 심층 신경망에서 시그모이드 함수를 사용해야 할 경우엔 출력 층에서만 사용하도록 합니다.

❷ 소프트맥스

소프트맥스 활성화 함수는 시그모이드 활성화 함수가 더욱 일반화된 형태입니다. 이것은 다중 클래스 분류 문제에 사용됩니다. 시그모이드 함수와 비슷하게 이것은 0에서 1사이의 값들을 생성합니다. 소프트맥스 함수는 은닉 층에서는 사용하지 않으며, 다중 분류 모델에서 출력 층에서만 사용됩니다.

❸ tanh

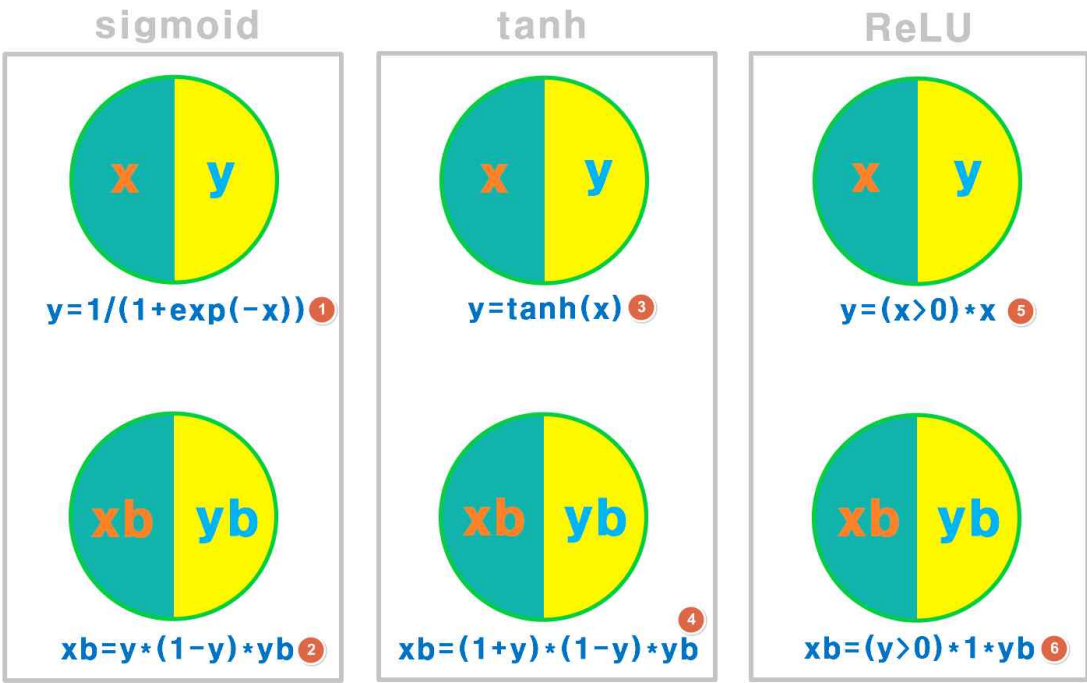
tanh 함수는 출력 값의 범위가 -1에서 1사이라는 것을 빼고는 시그모이드 함수와 유사합니다. 시그모이드 함수처럼 많이 사용되지 않습니다.

❹ ReLU

ReLU 함수는 딥러닝에서 가장 인기 있는 활성화 함수입니다. 특히 합성곱 신경망(CNN)에서 많이 사용됩니다. ReLU 함수는 계산이 빠르고 심층 신경망에서도 신호 전달이 잘 됩니다. ReLU 함수의 경우 입력 값이 음수가 될 경우 출력이 0이 되기 때문에 이런 경우에는 어떤 노드를 완전히 죽게 하여 어떤 것도 학습하지 않게 합니다. 이러한 노드가 많으면 많을수록 신경망 전체적으로 학습이 되지 않는 단점이 있습니다. ReLU의 다른 문제는 활성화 값의 극대화입니다. 왜냐하면 ReLU의 상한 값은 무한이기 때문입니다. 이것은 가끔 사용할 수 없는 노드를 만들어 학습을 방해하게 됩니다. 이러한 문제들은 초기 가중치 값을 고르게 할당하여 해결할 수 있습니다. 일반적으로 은닉 층에는 ReLU 함수를 적용하고, 출력 층은 시그모이드 함수나 소프트맥스 함수를 적용합니다.

03 활성화 함수의 순전파와 역전파

여기서는 sigmoid, tanh, ReLU 활성화 함수의 순전파와 역전파 수식을 살펴보고, 앞에서 구현한 인공 신경망에 활성화 함수를 적용하여 봅니다. 다음 그림은 활성화 함수의 순전파와 역전파 수식을 나타냅니다.



- sigmoid 함수의 경우 순전파 출력 y 값이 0이나 1에 가까울수록 역전파 x_b 값은 0에 가까워집니다. 순전파 출력 y 값이 0이나 1에 가깝다는 것은 순전파 입력 값 x 의 크기가 양 또는 음의 방향으로 어느 정도 크다는 의미입니다.
- tanh 함수의 경우 순전파 출력 y 값이 -1이나 1에 가까울수록 역전파 x_b 값은 0에 가까워집니다. 순전파 출력 y 값이 -1이나 1에 가깝다는 것은 순전파 입력 값 x 의 크기가 양 또는 음의 방향으로 어느 정도 크다는 의미입니다.
- ReLU 함수의 경우 순전파 입력 값 x 값이 0보다 크면 x 값이 y 로 전달되며, 0보다 작거나 같으면 0값이 y 로 전달됩니다. 역전파의 경우 순전파 출력 값 y 가 0보다 크면 y_b 값이 x_b 로 전달되며, 출력 값 y 가 0보다 작거나 같으면 x_b 로 0이 전달됩니다. 이 경우 x_b 에서 전 단계의 모든 노드로 전달되는 역전파 값은 0이 됩니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

시그모이드 순전파와 역전파

$$y = \frac{1}{1 + e^{-x}} \quad (1) \quad x_b = y(1 - y)y_b \quad (2)$$

tanh 순전파와 역전파

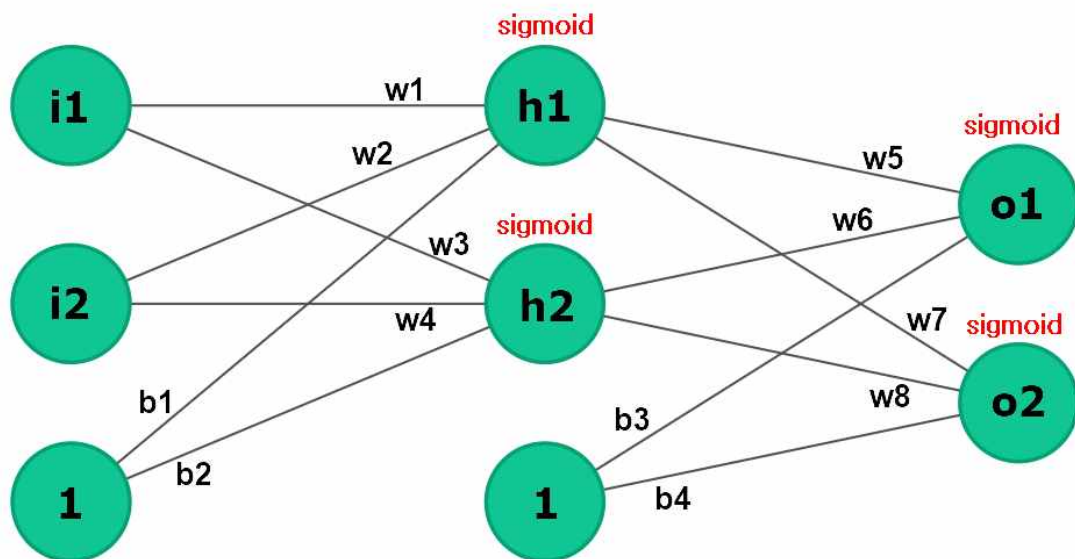
$$y = \tanh(x) \text{ ③ } \quad x_b = (1+y)(1-y)y_b \text{ ④}$$

ReLU 순전파와 역전파

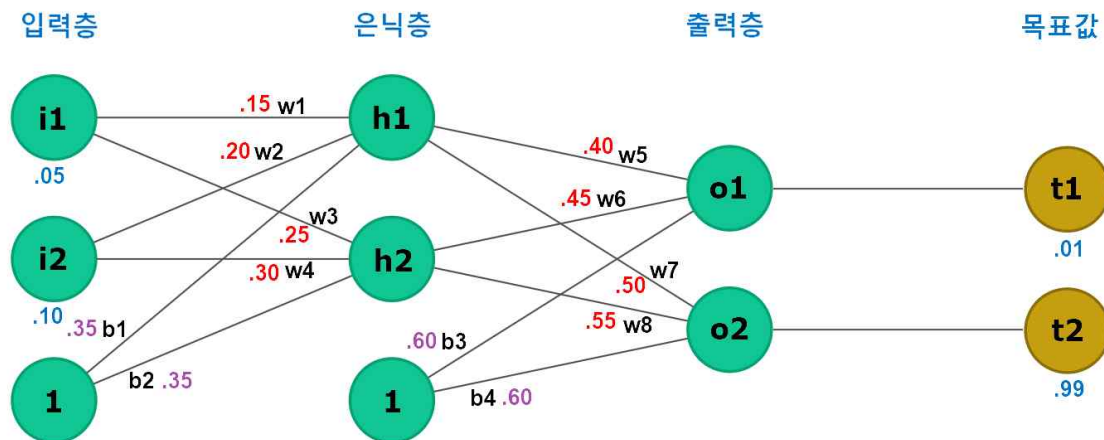
$$y = (x > 0)x \text{ ⑤ } \quad x_b = (y > 0)1y_b \text{ ⑥}$$

sigmoid 함수 적용해 보기

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림은 앞에서 구현한 2개의 입력, 2개의 은닉 신경, 2개의 출력 신경으로 구성된 인공 신경망입니다. 여기서는 은닉 신경과 출력 신경에 시그모이드(sigmoid) 활성화 함수를 추가해 봅니다. 이전 예제와 같이 목표 값은 각각 0.01, 0.99입니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

243_1.py

```

01 from math import exp
02
03 i1, i2 = .05, .10
04 t1, t2 = .01, .99
05
06 w1, w3 = .15, .25
07 w2, w4 = .20, .30
08 b1, b2 = .35, .35
09
10 w5, w7 = .40, .50
11 w6, w8 = .45, .55
12 b3, b4 = .60, .60
13
14 for epoch in range(2000):
15
16     print('epoch = %d' %epoch)
17
18     h1 = i1*w1 + i2*w2 + 1*b1
19     h2 = i1*w3 + i2*w4 + 1*b2
20     h1 = 1/(1+exp(-h1)) # ❶
21     h2 = 1/(1+exp(-h2)) # ❶
22
23     o1 = h1*w5 + h2*w6 + 1*b3
24     o2 = h1*w7 + h2*w8 + 1*b4

```

```

25 o1 = 1/(1+exp(-o1)) # ❶
26 o2 = 1/(1+exp(-o2)) # ❶
27
28 print(' o1, o2 = %6.3f, %6.3f' %(o1, o2))
29
30 E = (o1-t1)**2/2 + (o2-t2)**2/2
31 if E < 0.0000001:
32     break
33
34 o1b, o2b = o1 - t1, o2 - t2
35 o1b, o2b = o1b*o1*(1-o1), o2b*o2*(1-o2) # ❷
36
37 h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
38 h1b, h2b = h1b*h1*(1-h1), h2b*h2*(1-h2) # ❷
39
40 w1b, w3b = i1*h1b, i1*h2b
41 w2b, w4b = i2*h1b, i2*h2b
42 b1b, b2b = 1*h1b, 1*h2b
43 w5b, w7b = h1*o1b, h1*o2b
44 w6b, w8b = h2*o1b, h2*o2b
45 b3b, b4b = 1*o1b, 1*o2b
46
47 lr = 0.01
48 w1, w3 = w1 - lr*w1b, w3 - lr*w3b
49 w2, w4 = w2 - lr*w2b, w4 - lr*w4b
50 b1, b2 = b1 - lr*b1b, b2 - lr*b2b
51 w5, w7 = w5 - lr*w5b, w7 - lr*w7b
52 w6, w8 = w6 - lr*w6b, w8 - lr*w8b
53 b3, b4 = b3 - lr*b3b, b4 - lr*b4b

```


01 : math 모듈에서 exp 함수를 불러옵니다.

20, 21 : h1, h2 노드에 순전파 시그모이드 활성화 함수를 적용합니다.

25, 26 : o1, o2 노드에 순전파 시그모이드 활성화 함수를 적용합니다.

35 : o1b, o2b 노드에 역전파 시그모이드 활성화 함수를 적용합니다.

38 : h1b, h2b 노드에 역전파 시그모이드 활성화 함수를 적용합니다.


3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 1998
o1, o2 = 0.180, 0.878
epoch = 1999
o1, o2 = 0.180, 0.878
```

(1999+1)번째에 o1, o2가 각각 0.180, 0.878이 됩니다.

4. 다음과 같이 예제를 수정합니다.

```
14 for epoch in range(20000):
```


5.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 19998
o1, o2 = 0.042, 0.959
epoch = 19999
o1, o2 = 0.042, 0.959
```

(19999+1)번째에 o1, o2가 각각 0.042, 0.959가 됩니다.

6. 다음과 같이 예제를 수정합니다.

```
14 for epoch in range(200000):
```


7.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 199998
o1, o2 = 0.015, 0.985
epoch = 199999
o1, o2 = 0.015, 0.985
```

(199999+1)번째에 o1, o2가 각각 0.015, 0.985가 됩니다.

8. 다음과 같이 예제를 수정합니다.

```
14 for epoch in range(2000000):
```

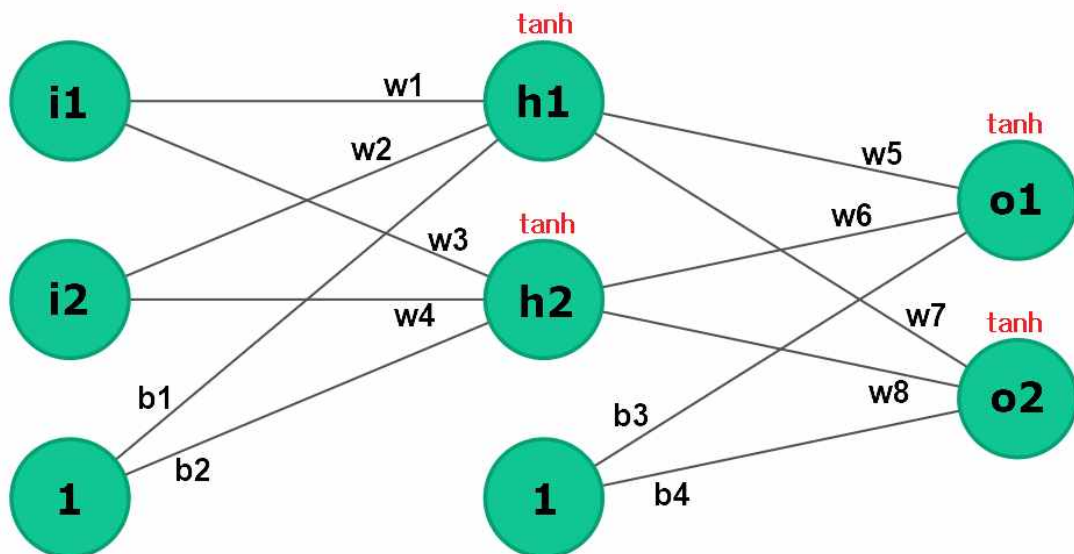
9.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 1078010
o1, o2 = 0.010, 0.990
epoch = 1078011
o1, o2 = 0.010, 0.990
```

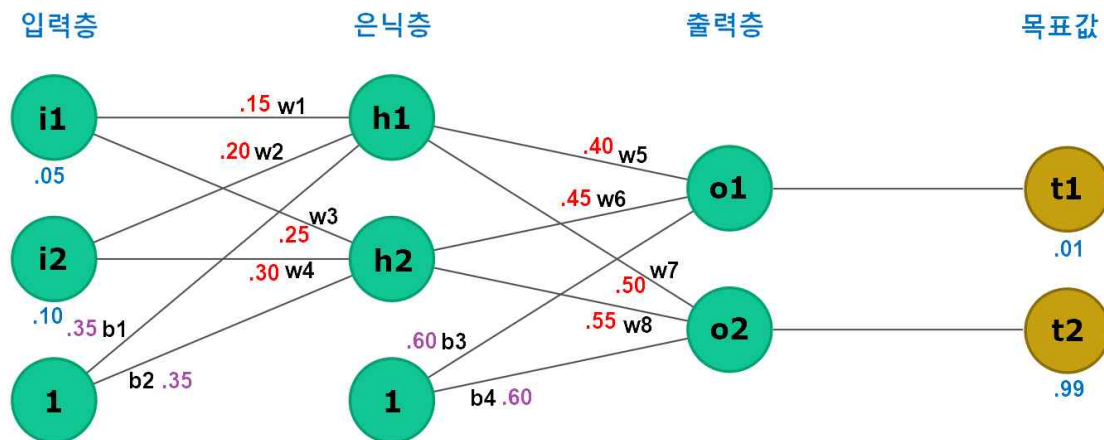
(1078011+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다.

tanh 함수 적용해 보기

이번에는 이전 예제에 적용했던 sigmoid 함수를 tanh 함수로 변경해 봅니다. 다음 그림을 살펴봅니다.



여기서는 은닉 신경과 출력 신경에 tanh 활성화 함수를 적용해 봅니다. 이전 예제와 같이 목표 값은 각각 0.01, 0.99입니다.



1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

243_2.py

```
01 from math import exp, tanh
```

01 : math 모듈에서 tanh 함수를 추가로 불러옵니다.

```
18 h1 = i1*w1 + i2*w2 + 1*b1
19 h2 = i1*w3 + i2*w4 + 1*b2
20 h1 = tanh(h1) # ③
21 h2 = tanh(h2) # ③
22
23 o1 = h1*w5 + h2*w6 + 1*b3
24 o2 = h1*w7 + h2*w8 + 1*b4
25 o1 = tanh(o1) # ③
26 o2 = tanh(o2) # ③
```


20, 21 : h1, h2 노드에 순전파 tanh 활성화 함수를 적용합니다.

25, 26 : o1, o2 노드에 순전파 tanh 활성화 함수를 적용합니다.

```
34 o1b, o2b = o1 - t1, o2 - t2
35 o1b, o2b = o1b*(1+o1)*(1-o1), o2b*(1+o2)*(1-o2) # ④
36
37 h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
38 h1b, h2b = h1b*(1+h1)*(1-h1), h2b*(1+h2)*(1-h2) # ④
```

35 : o1b, o2b 노드에 역전파 tanh 활성화 함수를 적용합니다.

38 : h1b, h2b 노드에 역전파 tanh 활성화 함수를 적용합니다.

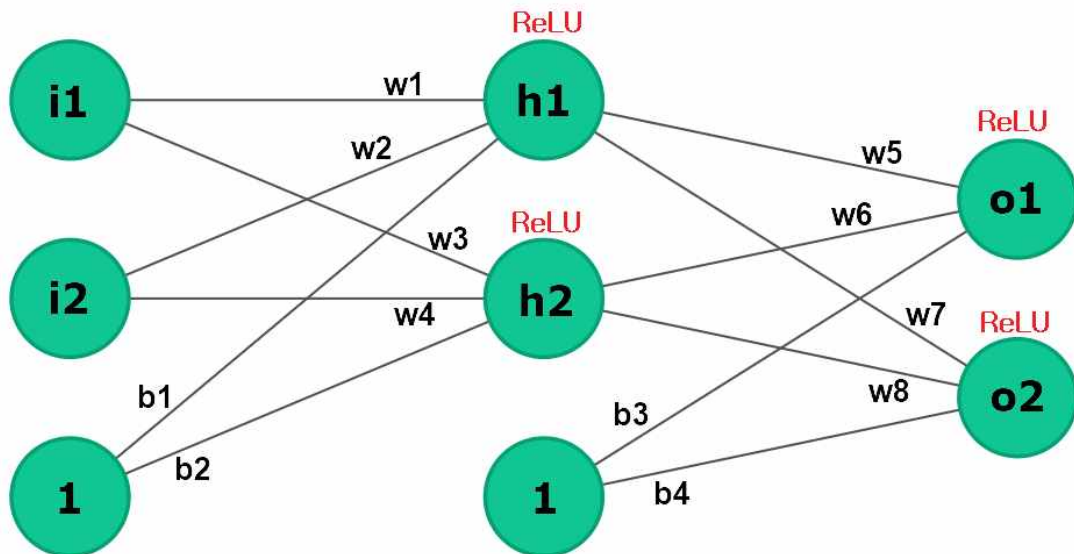
3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 236409
o1, o2 = 0.010, 0.990
epoch = 236410
o1, o2 = 0.010, 0.990
```

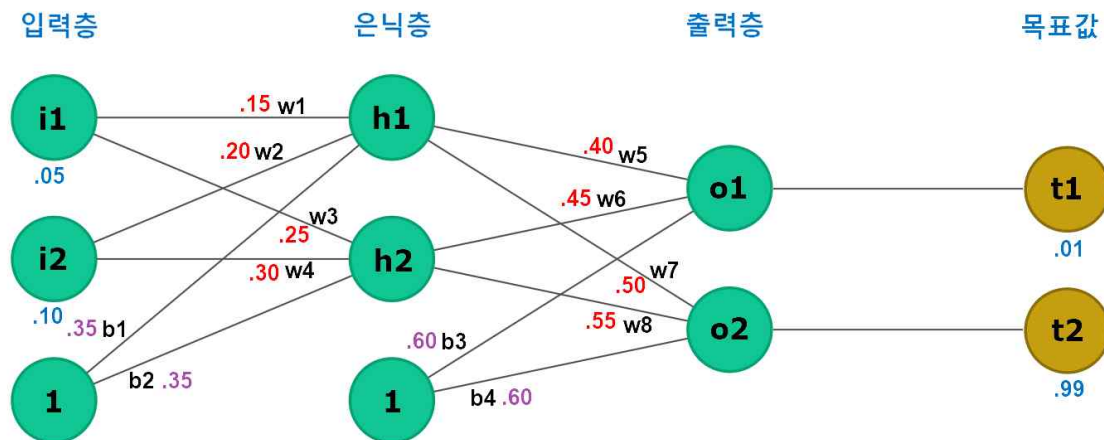
(236410+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다. sigmoid 함수보다 결과가 더 빨리 나오는 것을 볼 수 있습니다.

ReLU 함수 적용해 보기

이번에는 이전 예제에 적용했던 tanh 함수를 ReLU 함수로 변경해 봅니다. 다음 그림을 살펴 봅니다.



여기서는 은닉 신경과 출력 신경에 ReLU 활성화 함수를 적용해 봅니다. 이전 예제와 같이 목표 값은 각각 0.01, 0.99입니다.



1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

243_3.py

```

18  h1 = i1*w1 + i2*w2 + 1*b1
19  h2 = i1*w3 + i2*w4 + 1*b2
20  h1 = (h1>0)*h1 # 5
21  h2 = (h2>0)*h2 # 5
22
23  o1 = h1*w5 + h2*w6 + 1*b3
24  o2 = h1*w7 + h2*w8 + 1*b4
25  o1 = (o1>0)*o1 # 5
26  o2 = (o2>0)*o2 # 5

```

20, 21 : h1, h2 노드에 순전파 ReLU 활성화 함수를 적용합니다.

25, 26 : o1, o2 노드에 순전파 ReLU 활성화 함수를 적용합니다.


```

34  o1b, o2b = o1 - t1, o2 - t2
35  o1b, o2b = o1b*(o1>0)*1, o2b*(o2>0)*1 # 6
36
37  h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
38  h1b, h2b = h1b*(h1>0)*1, h2b*(h2>0)*1 # 6

```

35 : o1b, o2b 노드에 역전파 ReLU 활성화 함수를 적용합니다.

38 : h1b, h2b 노드에 역전파 ReLU 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.


```
epoch = 664
o1, o2 = 0.010, 0.990
epoch = 665
o1, o2 = 0.010, 0.990
```

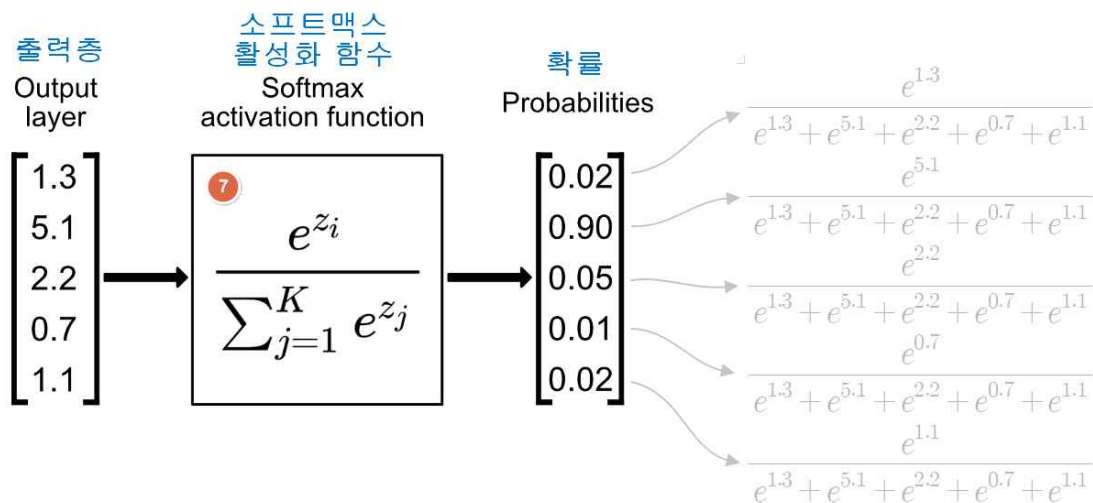
(665+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다. sigmoid, tanh 함수보다 결과가 훨씬 더 빨리 나오는 것을 볼 수 있습니다.

04 출력 층에 softmax 함수 적용해 보기

이전 단원에서 우리는 은닉 신경과 출력 신경에 시그모이드(sigmoid), tanh, ReLU 활성화 함수를 차례대로 적용해 보았습니다. 이 단원에서는 출력 신경의 활성화 함수를 소프트맥스(softmax)로 변경해 봅니다. softmax 활성화 함수는 크로스 엔트로피 오차(cross entropy error) 함수와 같이 사용되며, 분류(classification)에 사용됩니다.

softmax와 cross entropy

다음은 출력 층에서 활성화 함수로 사용되는 소프트맥스(softmax) 함수를 나타냅니다.



소프트맥스 함수는 출력 층에서 사용되는 활성화함수로 다중 분류를 위해 주로 사용됩니다. 소프트맥스 함수는 확률의 총합이 1이 되도록 만든 함수이며 아래에 나타난 크로스 엔트로피 오차 함수와 같이 사용됩니다.

$$E = - \sum_k t_k \log o_k \quad 8$$

우리는 앞에서 다음과 같은 평균 제곱 오차 함수를 살펴보았습니다.

$$E = \sum_k \frac{1}{2} (o_k - t_k)^2$$

평균 제곱 오차 함수의 경우 역전파 시 전파되는 오차가 다음과 같이 예측 값과 목표 값의 차인 것을 우리는 이미 앞에서 살펴보았습니다.

$$o_{kb} = o_k - t_k$$

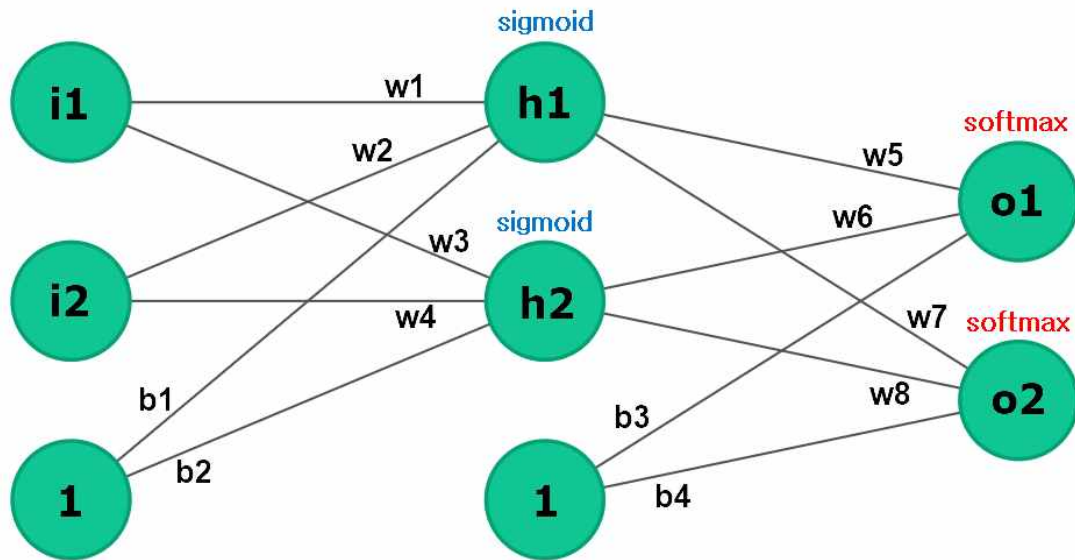
소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측 값과 목표 값의 차가 됩니다.

$$o_{kb} = o_k - t_k \quad 9$$

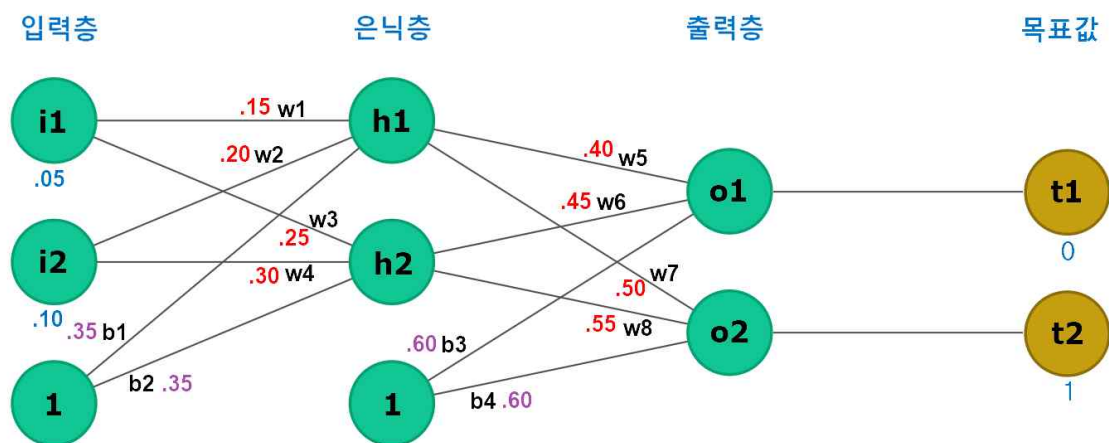
그래서 소프트맥스 함수를 활성화 함수로 사용할 경우 오차 함수는 크로스 엔트로피 오차 함수가 됩니다.

출력 층에 softmax 함수 적용해 보기

다음은 우리가 사용할 인공 신경망의 구조입니다.



여기서는 은닉 신경에는 sigmoid, 출력 신경에는 softmax 활성화 함수를 적용합니다. 목표 값은 0, 1를 사용합니다. 소프트맥스 활성화 함수와 크로스 엔트로피 오차 함수를 같이 사용할 때 일반적으로 목표 값은 0 또는 1의 값만 가지며, 총 합은 1이 됩니다. 다음 그림의 맨 오른쪽에 추가된 2개의 노드는 목표 값을 나타내며, 출력 층으로 나오는 예측 값을 목표 값에 가깝도록 가중치를 조정하게 됩니다.



1. 이전 예제를 복사합니다.
 2. 다음과 같이 예제를 수정합니다.
- 244_1.py

```
01 from math import exp, tanh, log
02
03 i1, i2 = .05, .10
```

```

04 t1, t2 = 0, 1
05
06 w1, w3 = .15, .25
07 w2, w4 = .20, .30
08 b1, b2 = .35, .35
09
10 w5, w7 = .40, .50
11 w6, w8 = .45, .55
12 b3, b4 = .60, .60
13
14 for epoch in range(2000000):
15
16     print('epoch = %d' %epoch)
17
18     h1 = i1*w1 + i2*w2 + 1*b1
19     h2 = i1*w3 + i2*w4 + 1*b2
20     h1 = 1/(1+exp(-h1)) # ❶
21     h2 = 1/(1+exp(-h2)) # ❶
22
23     o1 = h1*w5 + h2*w6 + 1*b3
24     o2 = h1*w7 + h2*w8 + 1*b4
25     o1m = o1 - max(o1, o2) # ❷
26     o2m = o2 - max(o1, o2) # ❷
27     o1 = exp(o1m)/(exp(o1m)+exp(o2m)) # ❷
28     o2 = exp(o2m)/(exp(o1m)+exp(o2m)) # ❷
29
30     print(' o1, o2 = %6.3f, %6.3f' %(o1, o2))
31
32     E = -t1*log(o1) + -t2*log(o2) # ❸
33     if E < 0.0001:
34         break
35
36     o1b, o2b = o1 - t1, o2 - t2 # ❹
37     # nothing for softmax + cross entropy error
38
39     h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
40     h1b, h2b = h1b*h1*(1-h1), h2b*h2*(1-h2) # ❺
41
42     w1b, w3b = i1*h1b, i1*h2b
43     w2b, w4b = i2*h1b, i2*h2b

```

```

44 b1b, b2b = 1*h1b, 1*h2b
45 w5b, w7b = h1*o1b, h1*o2b
46 w6b, w8b = h2*o1b, h2*o2b
47 b3b, b4b = 1*o1b, 1*o2b
48
49 lr = 0.01
50 w1, w3 = w1 - lr*w1b, w3 - lr*w3b
51 w2, w4 = w2 - lr*w2b, w4 - lr*w4b
52 b1, b2 = b1 - lr*b1b, b2 - lr*b2b
53 w5, w7 = w5 - lr*w5b, w7 - lr*w7b
54 w6, w8 = w6 - lr*w6b, w8 - lr*w8b
55 b3, b4 = b3 - lr*b3b, b4 - lr*b4b

```

01 : math 라이브러리에서 log 함수를 추가로 불러옵니다. log 함수는 자연로그 함수입니다.

04 : 목표 값을 각각 0과 1로 변경합니다.

25~28 : 출력 층의 활성화 함수를 소프트맥스로 변경합니다.

25, 26 : o1, o2에 대해 둘 중 더 큰 값을 빼줍니다. 이렇게 하면 26, 27 줄에서 오버플로우를 막을 수 있습니다. o1, o2에 대한 최종 결과는 같습니다. 자세한 내용은 [소프트맥스 오버플로우]를 검색해 봅니다.

32 : 오차 계산을 크로스 엔트로피 오차 형태의 수식으로 변경합니다. 소프트맥스 활성화 함수는 크로스 엔트로피 오차와 같이 사용합니다.


$$E = - \sum_k t_k \log o_k$$

33 : for 문을 빠져 나가는 오차 값을 0.0001로 변경합니다. 여기서 사용하는 값의 크기에 따라 학습의 정확도와 학습 시간이 결정됩니다.

36 : 소프트맥스 함수의 역전파 오차 계산 부분은 다음과 같습니다. 소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측 값과 목표 값의 차가 됩니다.

$$o_{kb} = o_k - t_k$$

그래서 소프트맥스 함수를 활성화 함수로 사용할 경우 오차 함수는 크로스 엔트로피 오차 함수가 됩니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

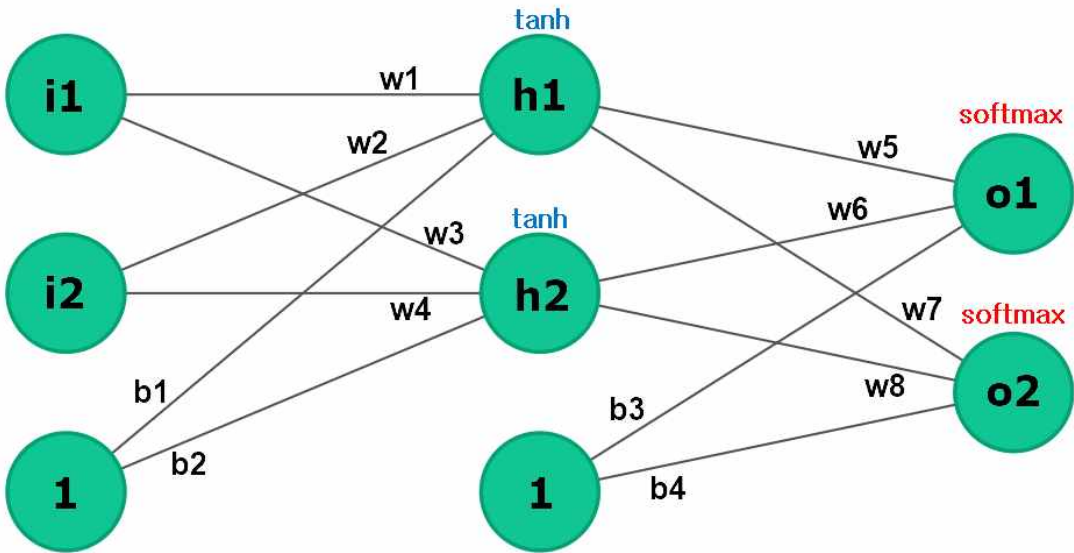
epoch = 211287
  o1,  o2  =  0.000,  1.000
epoch = 211288
  o1,  o2  =  0.000,  1.000

```

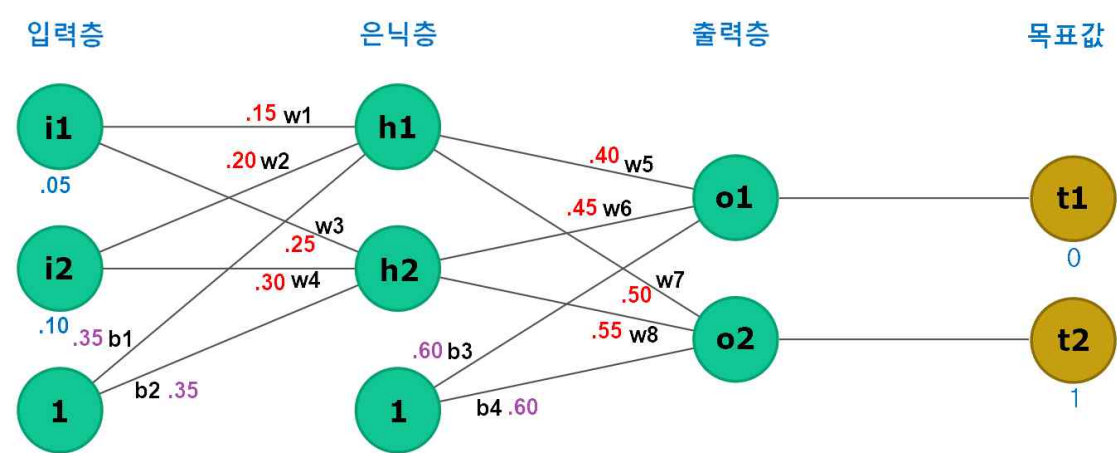
(211288+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

tanh와 softmax

여기서는 은닉 층 활성화 함수를 tanh로 변경해 봅니다. 다음 그림을 살펴봅니다.



여기서는 은닉 신경에는 tanh, 출력 신경에는 softmax 활성화 함수를 적용합니다. 목표 값은 0, 1를 사용합니다. 소프트맥스 활성화 함수와 크로스 엔트로피 오차 함수를 같이 사용할 때 일반적으로 목표 값은 0 또는 1의 값만 가지며, 총 합은 1이 됩니다. 다음 그림의 맨 오른쪽에 추가된 2개의 노드는 목표 값을 나타내며, 출력 층으로 나오는 예측 값을 목표 값에 가깝도록 가중치를 조정하게 됩니다.



1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

244_2.py

```
18 h1 = i1*w1 + i2*w2 + 1*b1
19 h2 = i1*w3 + i2*w4 + 1*b2
20 h1 = tanh(h1) # ③
21 h2 = tanh(h2) # ③
22
23 o1 = h1*w5 + h2*w6 + 1*b3
24 o2 = h1*w7 + h2*w8 + 1*b4
25 o1m = o1 - max(o1, o2) # ⑦
26 o2m = o2 - max(o1, o2) # ⑦
27 o1 = exp(o1m)/(exp(o1m)+exp(o2m)) # ⑦
28 o2 = exp(o2m)/(exp(o1m)+exp(o2m)) # ⑦
```


20, 21 : h1, h2 노드에 순전파 tanh 활성화 함수를 적용합니다.

25~28 : 출력 층의 활성화 함수는 softmax입니다.

```
36 o1b, o2b = o1 - t1, o2 - t2 # ⑨
37 # nothing for softmax + cross entropy error
38
39 h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
40 h1b, h2b = h1b*(1+h1)*(1-h1), h2b*(1+h2)*(1-h2) # ④
```

36 : softmax 함수의 역전파 오차 계산 부분입니다.

40 : h1b, h2b 노드에 역전파 tanh 활성화 함수를 적용합니다.

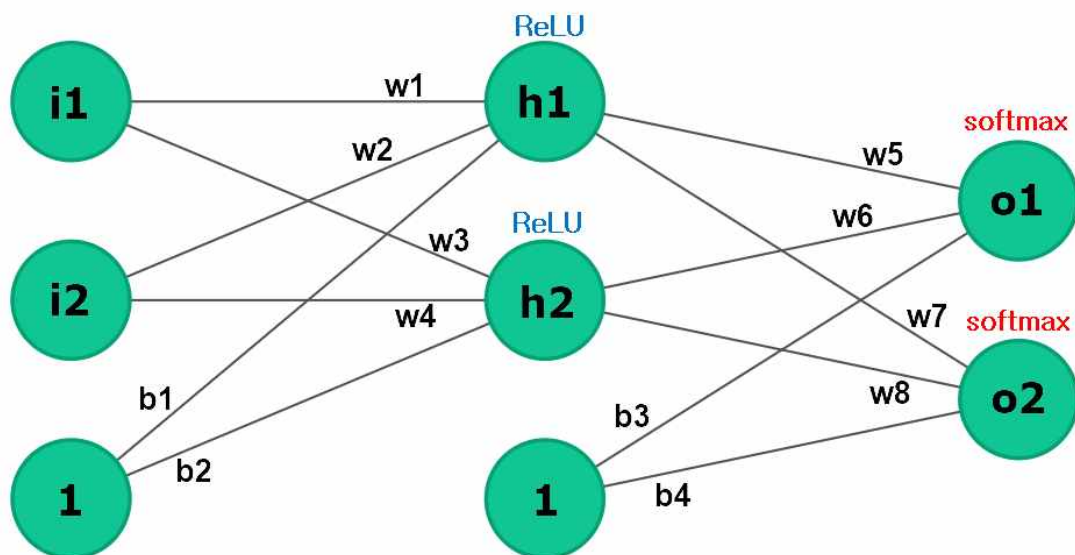
3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 174992
o1, o2 = 0.000, 1.000
epoch = 174993
o1, o2 = 0.000, 1.000
```

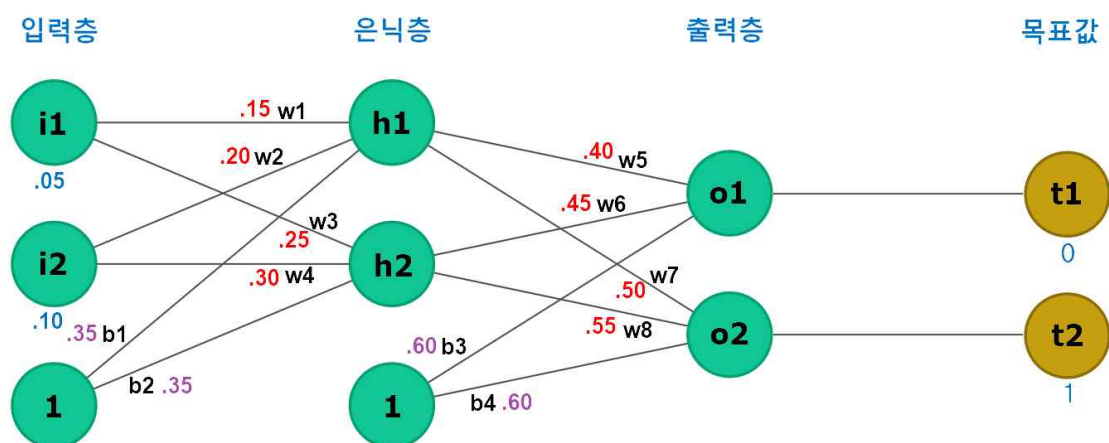
(174993+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

ReLU와 softmax

여기서는 은닉 층 활성화 함수를 ReLU로 변경해 봅니다. 다음 그림을 살펴봅니다.



여기서는 은닉 신경에는 ReLU, 출력 신경에는 softmax 활성화 함수를 적용합니다. 목표 값은 0, 1를 사용합니다. 소프트맥스 활성화 함수와 크로스 엔트로피 오차 함수를 같이 사용할 때 일반적으로 목표 값은 0 또는 1의 값만 가지며, 총 합은 1이 됩니다. 다음 그림의 맨 오른쪽에 추가된 2개의 노드는 목표 값을 나타내며, 출력 층으로 나오는 예측 값을 목표 값에 가깝도록 가중치를 조정하게 됩니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.
244_3.py

```
18 h1 = i1*w1 + i2*w2 + 1*b1
19 h2 = i1*w3 + i2*w4 + 1*b2
20 h1 = (h1>0)*h1 # 5
```



```

21     h2 = (h2>0)*h2 # ⑤
22
23     o1 = h1*w5 + h2*w6 + 1*b3
24     o2 = h1*w7 + h2*w8 + 1*b4
25     o1m = o1 - max(o1, o2) # ⑦
26     o2m = o2 - max(o1, o2) # ⑦
27     o1 = exp(o1m)/(exp(o1m)+exp(o2m)) # ⑦
28     o2 = exp(o2m)/(exp(o1m)+exp(o2m)) # ⑦

```

20, 21 : h1, h2 노드에 순전파 ReLU 활성화 함수를 적용합니다.

25~28 : 출력 층의 활성화 함수는 softmax입니다.


```

36     o1b, o2b = o1 - t1, o2 - t2 # ⑨
37     # nothing for softmax + cross entropy error
38
39     h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
40     h1b, h2b = h1b*(h1>0)*1, h2b*(h2>0)*1 # ⑥

```

36 : softmax 함수의 역전파 오차 계산 부분입니다.

40 : h1b, h2b 노드에 역전파 ReLU 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 56960
  o1,  o2  =  0.000,  1.000
epoch = 56961
  o1,  o2  =  0.000,  1.000

```

(56961+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

이상에서 출력 층의 활성화 함수는 소프트맥스, 오차 계산 함수는 크로스 엔트로피 오차 함수인 인공 신경망을 구현해 보았습니다.