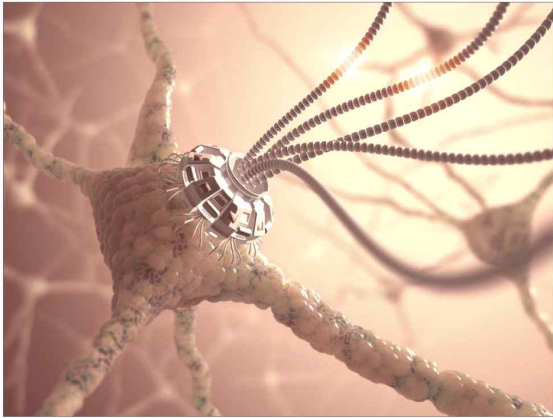


Chapter 01

인공 지능 딥러닝의 이해

이번 장에서는 인공지능의 딥러닝을 이해해 봅니다. 첫 번째, 딥러닝의 핵심인 인공 신경망이 무엇인지 알아보고, 딥러닝에 대해 어떤 학습 방법이 있는지 살펴보고, 생물학적 신경과 비교해 보며 딥러닝의 인공 신경망을 이해해 봅니다. 두 번째 딥러닝에 대한 기본 예제를 구글의 코랩과 Keras 라이브러리를 이용해 수행해 보면서 딥러닝을 접해봅니다. 세 번째 중고등학교 때 배웠던 기본적인 함수를 딥러닝의 인공 신경망으로 구현해 보면서 딥러닝의 인공 신경망과 함수의 관계를 이해해 봅니다. 마지막으로 손 글씨 데이터, 패션 데이터를 이용하여 실제 활용되는 딥러닝을 살펴봅니다.

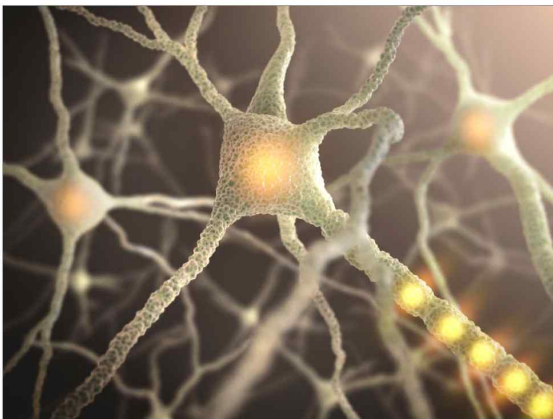
01 인공 신경망의 이해



인공 신경망은 딥러닝의 약진에 의해 최근 몇 년 동안 주목을 받아왔습니다. 그러면 인공 신경망은 무엇이고 어떻게 만들어졌을까요? 여기서는 인공 신경망의 바탕이 되는 실제 생체 신경의 구조와 구성 요소를 살펴보고 그것들이 어떻게 인공 신경의 구조와 구성요소에 대응이 되는지 살펴봅니다.

01 인공 신경망이란?

독자 여러분은 지금까지 왜 사람에게는 아주 간단하지만 컴퓨터에게는 상상할 수 없을 정도로 어려운 일들이 있는지 궁금해 한 적이 있나요? 인공 신경망(ANN's : Artificial neural networks)은 인간의 중앙 신경계로부터 영감을 얻어 만들어졌습니다. 생체 신경망과 같이 인공 신경망은 커다란 망으로 함께 연결되어 있는 인공 신경을 기반으로 구성됩니다. 개개의 인공 신경은 생체 신경과 같이 간단한 신호 처리를 할 수 있도록 구현되어 있습니다.



인공 신경망으로 할 수 있는 일들

그러면 우리는 인공 신경망으로 무엇을 할 수 있을까요? 인공 신경망은 많은 문제 영역에 성공적으로 적용되어 왔습니다. 예를 들어 다음과 같은 문제들에 적용되었습니다.

- 패턴 인식에 의한 데이터 분류
그림에 있는 이것은 나무인가?
- 입력 데이터가 일반적인 패턴과 맞지 않았을 때의 이상 감지
트럭 운전사가 잠들 위험이 있는가?
이 지진은 일반적인 땅의 움직임인가 아니면 커다란 지진인가?
- 신호 처리
신호 거르기
신호 분리하기
신호 압축하기
- 예측과 예보에 유용한 목표 함수 접근
이 폭풍은 태풍으로 변할 것인가?

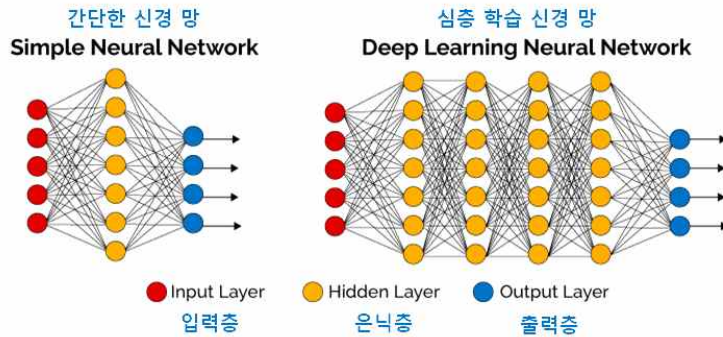
이런 문제들은 조금은 추상적으로 들립니다. 그래서 몇 가지 실제로 적용된 응용 예들을 보도록 합니다. 인공 신경망은 다음과 같은 것들을 할 수 있습니다.

- 얼굴 확인하기
- 음성 인식하기
- 손 글씨 읽기
- 문장 번역하기
- 게임 하기(보드 게임이나 카드 게임)
- 자동차나 로봇 제어하기
- 그리고 더 많은 것들!

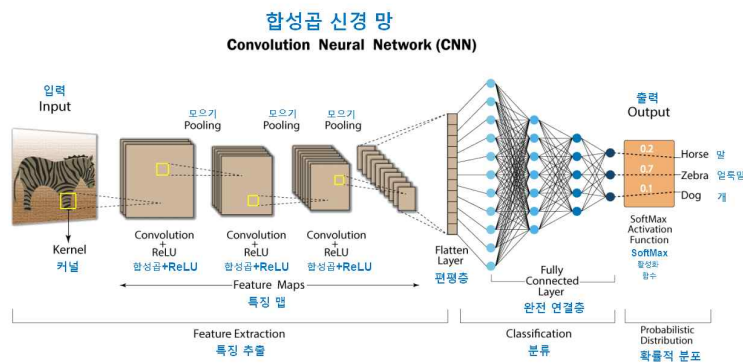
인공 신경망을 이용하면 세상에 있는 많은 문제들을 해결할 수 있습니다. 독자 여러분도 해결하고 싶은 문제가 있다면, 인공 신경망을 이용해 해결할 가능성이 있습니다. 인공 신경망을 통한 문제 해결은 이제 선택이 아닌 필수가 되어가고 있으며, 인공 신경망을 통한 문제 해결 능력은 여러분에게 더 많은 기회를 줄 것입니다.

인공 신경망의 구조

인공 신경망을 구성하는 방법은 다양합니다. 예를 들어 다음과 같은 형태로 인공 신경망을 구성할 수 있습니다. 다음 그림에서 노란색 노드로 표현된 은닉 층이 2층 이상일 때 심층 신경망(DNN)이라고 합니다.

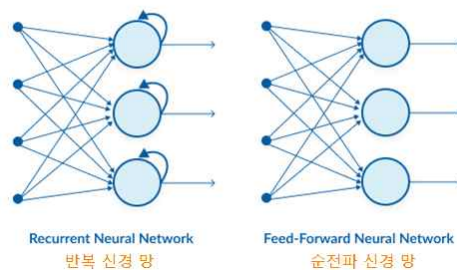


다음은 CNN 형태의 인공 신경망입니다. CNN은 이미지 인식에 뛰어난 인공 신경망으로 이미지의 특징을 뽑아내는 인공 신경망과 분류를 위한 인공 신경망으로 구성됩니다.

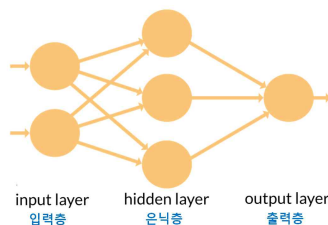


다음은 RNN 형태의 인공 신경망입니다. 아래 그림에서 왼쪽에 있는 그림은 RNN 형태의 신경망으로 노드에서 나온 값이 다시 되먹임 되는 형태로 인공 신경망이 구성됩니다. 오른쪽에 있는 그림은 한 방향으로만 신호가 흐르는 기본적인 인공 신경망입니다. RNN 형태의 인공 신경망은 문장 인식에 뛰어난 인공 신경망입니다.

반복 신경 망 구조
Recurrent Neural Network structure



인공 신경망은 구성 방법에 따라 동작 방식도 달라집니다. 가장 간단한 인공 신경망의 구조는 신호가 한 방향으로 흐르는 인공 신경망으로 다음과 같은 형태입니다.



일반적으로 인공 신경망은 3개의 층으로 구성됩니다. 각각 입력 층, 은닉 층, 출력 층이라고

합니다. 입력 층은 입력 신호를 받아서 다음 층에 있는 은닉 층으로 보냅니다. 은닉 층은 하나 이상 존재할 수 있습니다. 마지막에는 결과를 전달하는 출력 층이 옵니다.

02 인공 신경망의 학습 방법

전통적인 알고리즘들과는 달리 인공 신경망은 프로그래머의 의도대로 작업하도록 ‘프로그래밍되거나’ 또는 ‘구성되거나’ 할 수 없습니다. 인간의 뇌처럼 인공 신경망은 하나의 일을 수행할 방법을 직접 배워야 합니다. 일반적으로 인공 신경망의 학습 방법에는 3가지 전략이 있습니다.

지도 학습





















가장 간단한 학습 방법입니다. 미리 알려진 결과들이 있는 충분히 많은 데이터가 있을 때 사용하는 방법입니다. 지도 학습은 다음처럼 진행됩니다. 하나의 입력 데이터를 처리합니다. 출력값을 미리 알려진 결과와 비교합니다. 인공 신경망을 수정합니다. 이 과정을 반복합니다. 이것이 지도 학습 방법입니다. 예를 들어 엄마가 어린 아이에게 그림판을 이용하여 사물을 학습 시키는 방법은 지도 학습과 같습니다. 한글, 숫자 등에 대한 학습도 지도 학습의 형태입니다. 아래에 있는 그림판에는 동물, 과일, 채소 그림이 있고 해당 그림에 대한 이름이 있습니다. 아이에게 고양이를 가리키면서 ‘고양이’라고 알려주는 과정에서 아이는 학습을 하게 됩니다. 이와 같은 방식으로 인공 신경망도 학습을 시킬 수 있으며, 이런 방법을 지도 학습이라고 합니다.

동물 그림판				과일 그림판			
							
얼룩말	낙타	원숭이	사슴	레몬	포도	딸기	복숭아
							
여우	사자	멧돼지	호랑이	바나나	사과	산딸기	키위
							
표범	코뿔소	늑대	곰	자몽	귤	체리	라임
							
하마	오랑우탄	강아지	고양이	양파	가지	옥수수	토마토

비지도 학습

비지도 학습은 입력 값이 목표 값과 같을 때 사용하는 학습 방법입니다. 예를 들어, 메모리 카드 게임을 하는 방식을 생각해 봅니다. 메모리 카드 게임을 할 때 우리는 그림에 표현된 사

물의 이름을 모르는 상태로 사물의 형태를 통째로 기억해야 합니다. 그리고 같은 그림을 찾아 내며 게임을 진행하게 됩니다. 이와 같이 입력 값과 출력 값이 같은 형태의 데이터를 학습할 때, 즉, 입력 값을 그대로 기억해 내야 하는 형태의 학습 방법을 비지도 학습이라고 합니다.

MEMORY GAME				
				
				
				
				

강화 학습

인공 신경망이 익숙하지 않은 환경에서 시행착오를 통해 이익이 되는 동작을 취할 확률은 높이고 손해가 되는 동작을 취할 확률은 낮추게 하는 학습 방법입니다. 즉, 이익이 되는 동작을 강화해가는 학습 방법입니다. 예를 들어, 우리가 익숙하지 않은 환경에서 어떤 동작을 취해야 하는지 모를 때, 일단 할 수 있는 동작을 취해보고 그 동작이 유리한지 불리한지를 체득하는 형태의 학습 방식과 같습니다. 이 과정에서 유리한 동작은 기억해서 점점 더 하게 되고 불리한 동작도 기억해서 점점 덜 하게 됩니다.



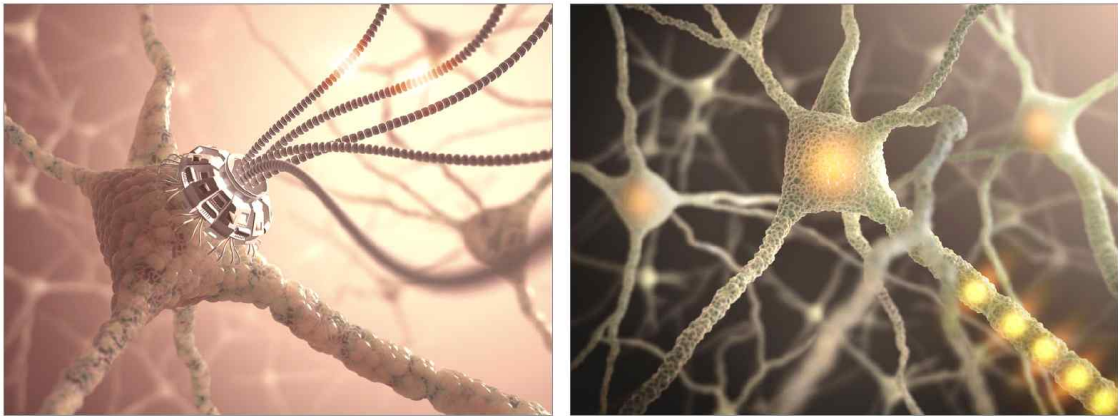
03 인공 신경 살펴보기

앞에서 우리는 인공 신경망에 대해 살펴보았습니다. 그러면 인공 신경망은 무엇으로 구성될까

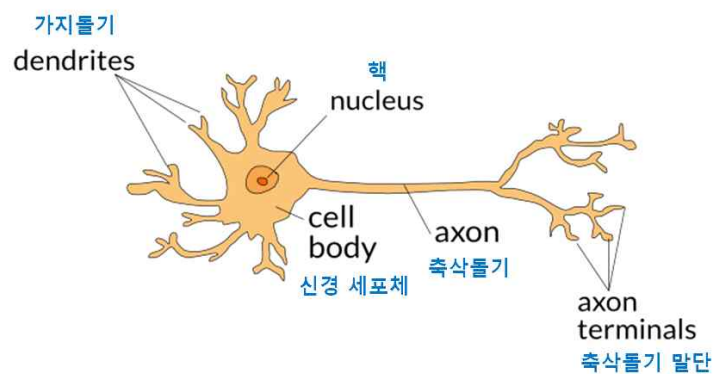
요? 여기서는 인공 신경망을 구성하는 인공 신경에 대해 생물학적 신경과 비교해 보면서 그 내부 구조를 살펴보도록 합니다.

인공 신경과 생물학적 신경

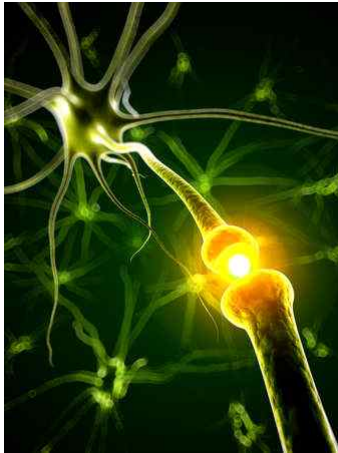
인공 신경망의 구성요소는 인공 신경입니다. 인공 신경이라는 이름은 생물학적 신경으로부터 얻어졌습니다. 인공 신경은 우리 두뇌의 구성 요소 중 하나인 생물학적 신경의 동작을 따라 만들어진 모형(model)입니다. 즉, 인공 신경은 생물학적 신경의 모형입니다.



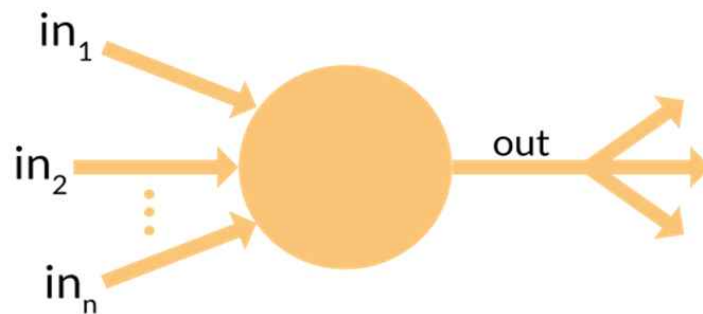
생물학적 신경은 신호를 받기 위한 여러 개의 가지 돌기(dendrites), 입력받은 신호를 처리하기 위한 신경 세포체(cell body), 다른 신경들로 신호를 내보내기 위한 축삭돌기(axon)와 축삭돌기 말단으로 구성됩니다.



특히 축삭돌기 말단과 다음 신경의 가지 돌기 사이의 틈을 시냅스라고 합니다.

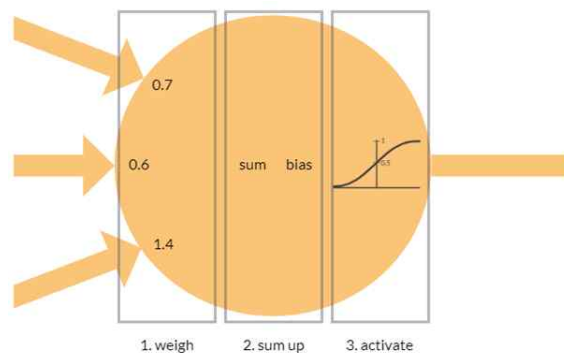


시냅스는 신경결합 부라고도 하며 한 신경에서 다른 신경으로 신호를 전달하는 연결지점을 말합니다. 인공 신경은 데이터를 받기 위한 여러 개의 입력 부, 입력받은 데이터를 처리하는 처리부, 그리고 여러 개의 다른 인공 신경들로 연결될 수 있는 하나의 출력부를 가집니다. 특히 인공 신경의 출력부에는 다음 인공 신경의 입력부에 맞는 형태의 데이터 변환을 위한 활성화 함수가 있습니다.



인공 신경 내부 살펴보기

이제 인공 신경 안으로 들어가 봅시다. 어떻게 인공 신경은 입력을 처리할까요? 독자 여러분은 하나의 인공 신경 안에서 그 계산들이 실제로 얼마나 간단한지 알면 깜짝 놀랄 수도 있습니다. 인공 신경은 세 개의 처리 단계를 수행합니다.



❶ 각각의 입력 값은 가중치에 의해 커지거나 작아집니다.

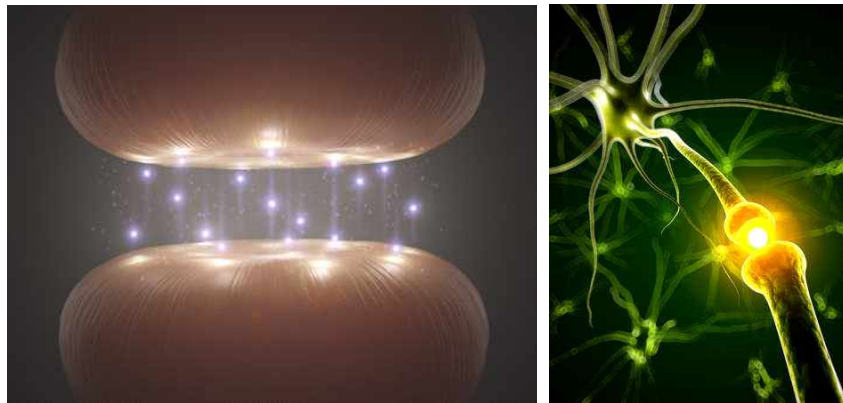
하나의 입력 신호(데이터)가 들어올 때 그 신호는 그 입력에 할당된 하나의 가중치(weight)에 의해 곱해집니다. 예를 들어, 하나의 인공 신경이 그림과 같이 3 개의 입력을 가진다면, 그 인공 신경은 각 입력에 적용될 수 있는 3개의 가중치를 가집니다. 학습 과정에서 인공 신경망은 결과 값과 목표 값의 오차를 기반으로 가중치들을 조정합니다. 생물학적 신경의 가지 돌기가 그 두께에 따라 신호가 더 잘 전달되거나 덜 전달되는 것처럼 인공 신경의 가중치도 그 값에 따라 신호(데이터)가 커지거나 작아집니다. 가중치는 다른 말로 강도(strength)라고도 합니다. 즉, 가중치는 입력 신호가 전달되는 강도를 결정합니다. 입력 신호가 작더라도 가중치가 크면 신호가 커지며, 입력 신호가 크더라도 가중치가 작으면 내부로 전달되는 신호는 작아집니다. 인공 신경의 가중치는 생물학적 신경의 가지 돌기의 두께로 비유할 수 있습니다.

❷ 모든 입력 신호들은 더해집니다.

가중치에 의해 곱해진 입력 신호들은 하나의 값으로 더해집니다. 그리고 추가적으로 보정 값(offset)도 하나 더해집니다. 이 보정 값은 편향(bias)이라고 불립니다. 인공 신경망은 학습 과정에서 편향도 조정합니다. 편향은 하나로 더해진 입력 신호에 더해지는 신호로 신호를 좀 더 크게 하거나 또는 좀 더 작게 하는 역할을 합니다. 즉, 신호를 조금 더 강화하거나 조금 더 약화하는 역할을 합니다.

❸ 신호를 활성화합니다.

앞에서 더해진 입력신호들은 활성화함수를 거쳐 하나의 출력 신호로 바뀝니다. 활성화 함수는 신호 전달 함수라고도 하며 신호의 형태를 다른 인공 신경의 입력에 맞게 변경하여 출력하는 역할을 합니다. 생물학적 신경을 시냅스가 연결하는 것처럼 활성화함수는 인공 신경을 연결하는 역할을 수행합니다.



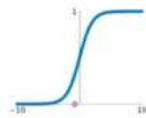
다음은 인공 신경망에 사용되는 활성화함수입니다. 활성화 함수는 인공 신경망의 활용 영역에 따라 달리 사용됩니다.

활성화 함수

Activation Functions

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

$$\tanh(x)$$

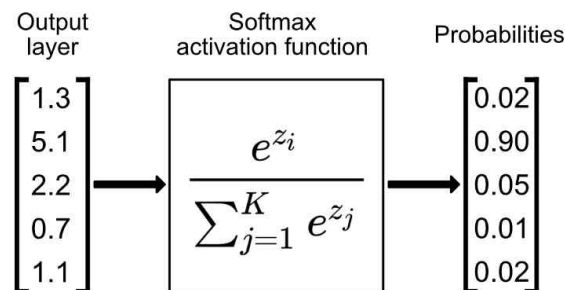


ReLU

$$\max(0, x)$$



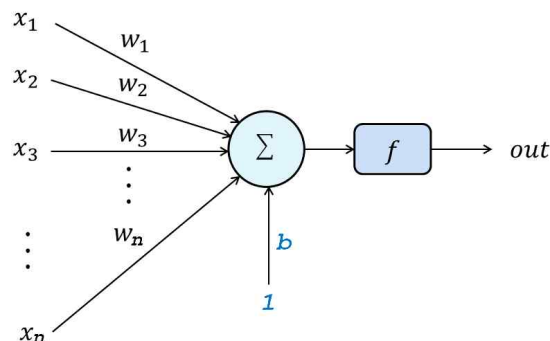
일반적으로 출력 값을 0에서 1사이의 값으로 하고자 할 경우엔 sigmoid 함수, 출력값을 -1에서 1사이의 값으로 하고자 할 경우엔 tanh 함수, 0보다 큰 출력 값만 내보내고자 할 경우엔 relu 함수를 사용합니다. 특히 다음은 분류를 위해 출력 층에 사용할 수 있는 활성화 함수입니다.



활성화 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다. 여기서는 활성화 함수로 이러한 함수들이 사용된다는 정도로 이해하고 넘어갑시다.

인공 신경 함수 수식

다음은 하나의 인공 신경과 그 인공 신경으로 들어가는 입력 값 x 의 집합, 입력 값에 대한 가중치(신호 강도) w 의 집합, 편향 입력 값 1, 편향 b , 가중치와 편향을 통해 들어오는 입력 값들의 합, 그 합을 입력으로 받는 활성화 함수 f , 활성화 함수 f 의 출력 out 을 나타냅니다.



인공 신경의 수식은 일반적으로 다음과 같습니다.

$$out = f(x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3 + \dots + x_n \times w_n + 1 \times b)$$

$$out = f\left(\sum_{i=1}^n x_i \times w_i + 1 \times b\right)$$

예를 들어, 활성화 함수가 sigmoid 함수일 경우 인공 신경의 수식은 다음과 같습니다.

$$out = \frac{1}{1 + e^{x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3 + \dots + x_n \times w_n + 1 \times b}}$$

$$out = \frac{1}{1 + e^{\sum_{i=1}^n x_i \times w_i + 1 \times b}}$$

또, 활성화 함수가 relu 함수일 경우 인공 신경의 수식은 다음과 같습니다.

$$out = \max(0, x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3 + \dots + x_n \times w_n + 1 \times b)$$

$$out = \max(0, \sum_{i=1}^n x_i \times w_i + 1 \times b)$$

이러한 수식들은 뒤에서 자세히 구현해 보면서 그 동작들을 이해합니다. 여기서는 개략적으로 살펴보고 넘어가도록 합니다.

이상에서 인간의 두뇌를 모델로 한 인공 신경망, 인공 신경망으로 할 수 있는 일들, 인공 신경망의 구조, 인공 신경망의 학습 방법, 생물학적 신경과 인공 신경과의 관계, 인공 신경의 구성 요소를 살펴보았습니다. 이 과정에서 인공 신경의 수식은 생물학적 신경으로부터 직관적으로 유도된 것을 알 수 있었습니다. 인공 신경의 수식은 간단한 형태의 수식이지만 이러한 인공 신경으로 망을 구성할 때는 아주 큰 힘을 발휘하게 됩니다.

02 딥러닝 맛보기

이 단원에서는 기본적인 딥러닝 예제를 수행해보고, 머신 러닝이 무엇인지 알아봅니다. 그리고 구글이 제공하는 코랩 개발 환경을 구성한 후, 기존 방식의 함수 정의 방식과 머신 러닝 방식의 신경망 함수를 생성하고 사용해 봅니다.

01 Hello 딥러닝

여기서는 인공 신경망의 기본적인 “Hello, World”를 소개합니다. 기존 프로그래밍에서는 명확한 규칙을 가진 함수를 정의하면서 프로그래밍 합니다. 그러나 인공 신경망을 이용한 프로그래밍에서는 입력 값들과 출력 값들의 관계를 기반으로 인공 신경망 함수를 만듭니다. 인공 신경망 함수를 만드는 과정을 인공 신경망 함수 학습 또는 훈련이라고 합니다. 학습이나 훈련의 과정을 거치면 인공 신경망 함수는 내부적으로 입력 값과 출력 값들의 관계를 기반으로 만들어진 규칙을 가지게 됩니다. 인공 신경망 함수를 만드는 것은 찰흙으로 그릇을 빚는 것과 같습니다. 처음엔 그릇을 만들 찰흙만 준비된 상태에서 조물조물, 주물주물 하면서 그릇을 만들어 가듯이 초기화되지 않은 인공 신경망을 준비한 후, 조물조물, 주물주물 하면서 입력 값과 출력 값을 연결해 주는 인공 신경망 함수를 만들어 가게 됩니다. 이렇게 만들어진 인공 신경망 함수를 이용하여 새로운 입력 값에 대해 출력 값을 예측합니다.



다음 그림을 살펴봅니다. 여러분은 운동 추적을 인식하는 프로그래밍을 하고 있습니다. 여러분은 한 사람이 걷고 있는 속도를 규칙으로 하여 그 사람의 활동을 예측할 수 있습니다.



```
if(speed < 4) {  
    status = WALKING;  
}
```

여러분은 다른 규칙을 추가하여 달리기도 예측할 수 있도록 프로그램을 확장할 수 있습니다.



```
if(speed < 4) {  
    status = WALKING;  
} else {  
    status = RUNNING;  
}
```

마지막 규칙으로 여러분은 비슷하게 자전거 타기 예측을 추가할 수 있습니다.



```
if(speed < 4) {  
    status = WALKING;  
} else if(speed < 12) {  
    status = RUNNING;  
} else {  
    status = BIKING;  
}
```

```
}
```

그러면 다음과 같은 상황은 어떨까요? 여러분은 프로그램에 골프 같은 동작을 포함하고자 합니다. 그런데 골프 같은 동작을 예측하기 위한 규칙을 만들어낼 방법이 명확하지 않습니다.



```
if(speed < 4) {  
    status = WALKING;  
} else if(speed < 12) {  
    status = RUNNING;  
} else {  
    status = BIKING;  
}  
// 어떻게 하지?
```

골프 치는 동작을 인식할 수 있는 프로그램을 작성하는 것은 정말 어렵습니다. 그러면 어떻게 해야 할까요? 여러분은 ML(Machine Learning, 기계학습)을 이용하여 그 문제를 풀 수 있습니다!

02 머신러닝은 무엇일까요?

앞에서 소개된 프로그램을 짜기 위한 기존 방법을 생각해 봅시다.

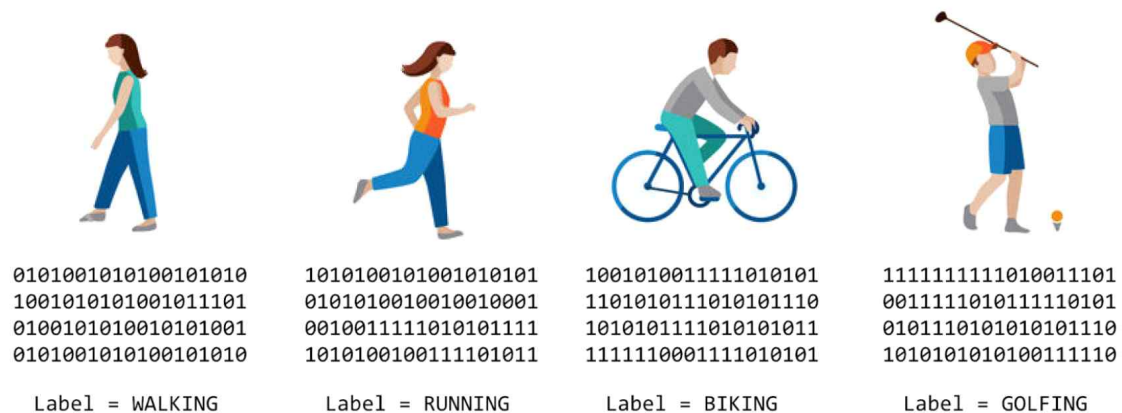


여러분은 프로그래밍 언어로 규칙을 가진 함수(Rules)를 정의합니다. 여러분의 프로그램에서 그 함수는 값(Data)을 받아 내부적으로 처리한 후, 결과 값(Answers)을 내어 놓습니다. 동작 감지의 경우에, 그 규칙들(동작의 형태들을 정의하기 위해 여러분이 작성한 코드)은 값들(사람의 움직임 속도)을 입력으로 받아 답을 생성합니다. 그 답은 사용자의 동작 상태(그들이 걷고 있었는지, 달리고 있었는지, 자전거를 타고 있었는지, 또는 다른 무언가를 하고 있었는지)를 결정하기 위한 함수로부터 나오는 값입니다.

ML을 통한 동작 상태를 감지하기 위한 프로그래밍 과정도 이전 방법과 아주 유사합니다. 단지 입출력 항목들의 위치가 다릅니다.



규칙을 정의하고 그것들을 프로그래밍 언어로 표현하는 대신에, 여러분은 데이터(값)와 함께 답들(값, 일반적으로 라벨이라고 불립니다)을 제공합니다. 그리고 기계(Machine)는 답들과 데이터간의 관계를 결정하는 규칙들을 만들어 냅니다. 그림에서 Data에는 라벨도 포함됩니다. 예를 들어, 여러분의 활동 감지 데이터는 ML 기반 프로그램 안에서 다음과 같이 보일수 있습니다.



여러분은 각각의 동작에 대해 많은 데이터를 모아서 “이것은 걷기처럼 보이는 것이야”, 또는 “이것은 뛰기처럼 보이는 것이야”라고 말하기 위해 모은 데이터에 라벨을 붙입니다. 이 동작 데이터와 라벨을 컴퓨터에 넣어줍니다. 그리고 나면, 컴퓨터는 데이터를 이용하여 특정한 동작을 나타내는 명확한 패턴이 무엇인지 결정할 수 있는 규칙을 만들어낼 수 있습니다.

전통적인 프로그래밍에서, 여러분의 코드는 일반적으로 여러분이 정의한 함수를 위주로 작성됩니다. ML 기반 프로그래밍에서, 여러분은 데이터와 라벨을 이용하여 인공 신경망 함수를 만들어 사용하게 됩니다. 인공 신경망 함수는 일반적으로 모델이라고 불립니다. 인공 신경망 함수는 우리가 원하는 어떤 기능을 유사하게 수행하는 모델 함수라고 생각할 수 있습니다.

여러분이 다음 그림을 통해 원하는 기능을 수행하는 인공 신경망 함수를 만들었다면 이제 여러분은 그 함수를 이용할 수 있습니다.



여러분이 만든 인공 신경망 모델 함수는 다음과 같이 사용됩니다.



여러분은 학습을 통해 만들어진 인공 신경망 모델 함수에 어떤 데이터를 주고 그 인공 신경망 모델 함수는 학습을 통해 얻은 그 규칙들을 사용하여 답을 예측 합니다. 예를 들어, “그 데이터는 걷기처럼 보여요”, 또는 “그 데이터는 골프 치기처럼 보여요”처럼 예측을 합니다.

03 구글 코랩 개발 환경 구성하기

독자 여러분의 첫 번째 인공 신경망 함수를 만들어 보기 위해 먼저 프로그래밍 환경을 구성해 봅니다. 일반적으로 인공 지능 프로그램은 파이썬 기반으로 작성됩니다. 여기서는 손쉽게 파이썬 환경을 구성하여 간단한 인공 신경망을 구성한 후, 학습을 수행해 봅니다. 구글에서 제공하는 코랩을 이용하면 복잡한 환경을 구성하지 않고 인공 신경망 관련 실습을 수행할 수 있습니다. 뒤에서는 파이썬으로 인공 신경망 함수를 직접 구현해 보면서 내부적으로 동작하는 원리를 자세히 살펴봅니다.

1. 다음과 같이 [google colab]을 검색합니다.



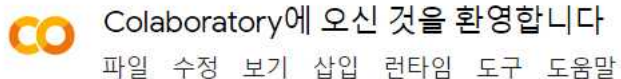
2. 다음 사이트를 찾아 들어갑니다.

<https://colab.research.google.com>

Google Colab

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your ...

3. 다음과 같이 구글 코랩 홈페이지가 열립니다.



4. 다음은 구글 코랩에 대해 소개하고 있습니다. 복잡한 구성이 필요치 않으며, GPU 기능도 제공합니다.

Colaboratory란?

줄여서 'Colab'이라고도 하는 Colaboratory를 사용하면 브라우저에서 Python을 작성하고 실행할 수 있습니다. Colab은 다음과 같은 이점을 자랑합니다.

- 구성이 필요하지 않음
- GPU 무료 액세스
- 간편한 공유

학생이든, 데이터 과학자든, AI 연구원이든 Colab으로 업무를 더욱 간편하게 처리할 수 있습니다. [Colab 소개 영상](#)에서 자세한 내용을 확인하거나 아래에서 시작해 보세요.

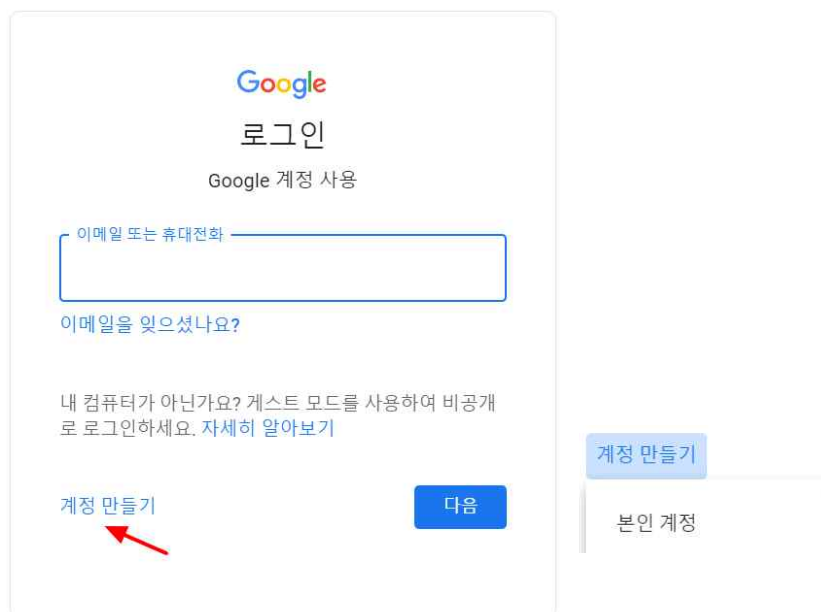
5. 다음과 같이 [파일]--[새 노트] 메뉴를 선택합니다.



6. 구글 코랩을 사용하기 위해서는 구글 계정이 필요합니다. 구글 계정이 있는 독자는 로그인 을 수행합니다.



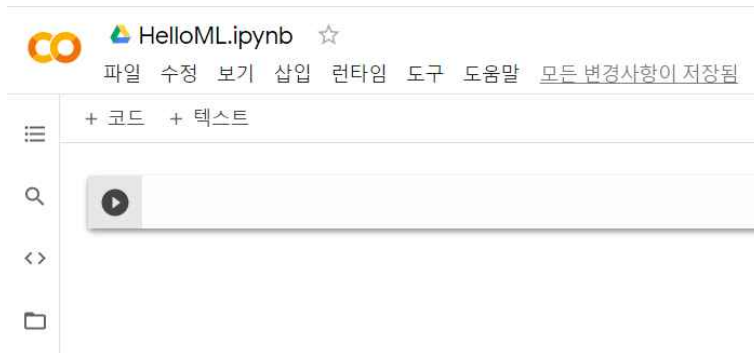
7. 구글 계정이 없는 독자는 계정을 생성합니다. 다음과 같이 [계정 만들기]--[본인 계정] 버튼을 눌러 계정을 생성합니다. 여기서는 구글 계정 생성 과정을 소개하지 않습니다.



8. 다음은 구글 코랩 파이썬 작성 화면입니다.



9. 다음과 같이 파일의 제목을 HelloML로 변경합니다. 제목 부분에 마우스 왼쪽 클릭한 후, 제목을 변경합니다.



04 기존 방식의 함수 정의와 사용

먼저 기존 방식으로 함수를 정의하고 사용하는 과정을 살펴봅니다.

다음은 중학교 때 배운 함수식입니다.

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

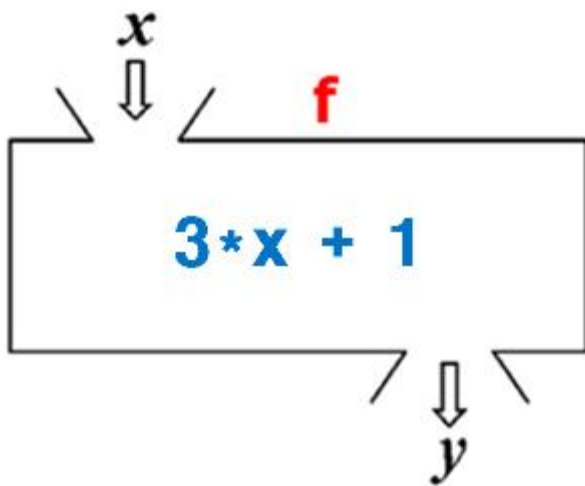
이 식에서

x가 1일 때 $y = f(1) = 3 \cdot 1 + 1$ 이 되어 y는 4가 됩니다.

x가 2일 때 $y = f(2) = 3 \cdot 2 + 1$ 이 되어 y는 7이 됩니다.

x가 -1일 때 $y = f(-1) = 3 \cdot (-1) + 1$ 이 되어 y는 -2가 됩니다.

이 함수를 그림으로 표현하면 다음과 같습니다.



함수 정의하고 사용해 보기

이제 기존 방식으로 함수 f 를 정의하고 사용해 봅니다.

1. 다음과 같이 예제를 작성합니다.

124_1.py

```
1 def f(x):  
2     return 3*x + 1  
3  
4 x = 10  
5 y = f(x)  
6  
7 print('y:', y)
```

1,2 : f 함수를 정의합니다.


4 : x 변수를 생성한 후, 10으로 초기화합니다.


5 : f 함수에 x 를 인자로 주어 호출한 후, 결과 값을 y 변수로 받습니다.

7 : `print` 함수를 호출하여 y 값을 출력합니다.

2. 다음은 구글 코랩에 작성한 화면입니다.

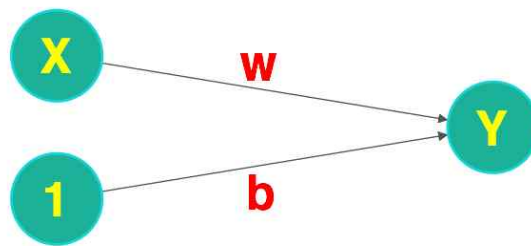
```
1 def f(x):  
2     return 3*x + 1  
3  
4 x = 10  
5 y = f(x)  
6  
7 print('y:', y)
```

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음과 같이 31이 표시되는 것을 확인합니다.

 y: 31

05 머신러닝 방식의 신경망 함수 생성과 사용

이번엔 ML 방식으로 인공 신경망 함수를 학습시키고 학습된 함수를 사용하는 과정을 살펴봅니다. 다음과 같은 모양의 인공 신경망을 구성하고 학습시켜 봅니다.



먼저 다음과 같은 숫자들의 집합 X, Y를 살펴봅니다. 독자 여러분은 숫자들 간의 관계가 보이시나요?

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

독자 여러분은 X, Y 숫자들을 보면서, X 값은 왼쪽에서 오른쪽으로 1씩, Y 값은 3씩 증가하는 것을 눈치 챌 수 있습니다. 그럴 경우 독자 여러분은 아마도 Y는 $3 \times X$ 더하기 또는 빼기 얼마와 같다고 생각했을 겁니다. 그리고 나서, 독자 여러분은 아마도 X가 0일 때, Y가 1인 것을 보았을 겁니다. 그리고 독자 여러분은 마침내 관계식 $Y=3 \times X+1$ 에 도달했을 겁니다.

독자 여러분이 유추해 낸 방식은 인공 신경망이 데이터(X, Y 값)를 이용하여 학습(또는 훈련)을 통해 데이터(X, Y)간의 관계를 발견하는 방식과 같습니다.

이제 이 식을 인공 신경망이 유추해 내는 과정을 살펴봅니다. 그러기 위해 먼저 필요한 것은 무엇일까요? 바로 데이터입니다. 다음으로 필요한 것은 그릇을 만들기 위해 찰흙이 필요한 것과 같이 학습되지 않은 인공 신경망입니다. 인공 신경망에 X 집합과 Y 집합을 주면, 인공 신경망은 학습을 통해 X와 Y간의 관계를 알아낼 수 있어야 합니다. 인공 신경망 함수는 일반적으로 인공 신경망 모델이라고 합니다. 모델은 우리말로 모형을 의미하며, 함수 모형으로 이해할 수 있습니다. 즉, 학습을 통해 만들어진 인공 신경망 함수는 $Y=3 \times X+1$ 함수의 모형 함수입니다. 인공 신경망 함수는 $Y=3 \times X+1$ 함수를 흉내 내는 함수라고 할 수 있습니다. 그래서 인공 신경망 함수를 근사 함수라고도 합니다.

1. 다음과 같이 [+ Code] 버튼을 누릅니다. 실행 창의 경계에 마우스 커서를 대면 버튼이 나타납니다.



2. 다음과 같이 예제를 작성합니다.

125_1.py

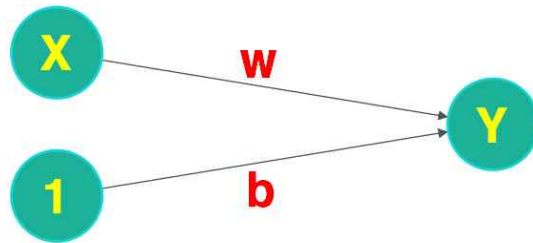
```
01 import tensorflow as tf
02 import numpy as np
03
04 xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0])
05 ys = np.array([-2.0, 1.0, 4.0, 7.0, 10., 13.])
06
07 model = tf.keras.Sequential([
08     tf.keras.layers.InputLayer(input_shape=(1,)),
09     tf.keras.layers.Dense(1)
10 ])
11
12 model.compile(optimizer='sgd', loss='mean_squared_error')
13
14 model.fit(xs, ys, epochs=5)
15
16 p = model.predict([10.0])
17
18 print('p:', p)
```

01 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

02 : import문을 이용하여 numpy 모듈을 np라는 이름으로 불러옵니다. numpy 모듈은 행렬 계산을 편하게 해주는 라이브러리입니다. 인공 신경망은 일반적으로 행렬 계산식으로 구성하게 됩니다.

04, 05 : np.array 함수를 이용하여 앞에서 살펴본 6 개의 X, Y 값을 준비합니다. np.array 함수는 tensorflow 라이브러리에서 데이터를 처리할 때 사용하는 배열을 생성합니다.

07~10 : tf.keras.Sequential 클래스를 이용하여 가장 간단한 인공 신경망을 생성합니다. 여기서 생성한 인공 신경망의 모양은 다음과 같습니다.



이 신경망은 하나의 인공 신경으로 구성됩니다. 인공 신경망의 내부 구조는 뒤에서 자세히 살펴봅니다. 생성된 인공 신경망은 일반적으로 모델이라고 합니다. 모델은 모형을 의미하며, 주어진 데이터에 맞추어진 원래 함수를 흉내 내는 함수인 근사 함수를 의미합니다.

07, 10 : 파이썬의 리스트를 나타냅니다.

8 : `tf.keras.layers.InputLayer` 함수를 이용하여 내부적으로 keras 라이브러리에서 제공하는 tensor를 생성하고, 입력 노드의 개수를 정해줍니다. tensor는 3차원 이상의 행렬을 의미하며, 인공 신경망 구성 시 사용하는 자료 형입니다.

9 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서 Dense는 내부적으로 $Y = X \cdot w + b$ 식을 생성하게 됩니다. 이 식에 대해서는 뒤에서 실제로 구현해 보며 그 원리를 살펴보도록 합니다.

12 : `model.compile` 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망을 구성할 때에는 2개의 함수를 정해야 합니다. loss 함수와 optimizer 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수와 최적화 함수에 대해서는 뒤에서 구현을 통해서 자세히 살펴봅니다. 손실 함수로는 `mean_squared_error` 함수를 사용하고 최적화 함수는 확률적 경사 하강(`sgd : stochastic gradient descent`) 함수를 사용합니다. `mean_squared_error`, `sgd` 함수는 뒤에서 직접 구현해 보도록 합니다.

14 : `model.fit` 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. `fit` 함수에는 X, Y 데이터가 입력이 되는데 인공 신경망을 X, Y 데이터에 맞도록 학습한다는 의미를 갖습니다. 즉, X, Y 데이터에 맞도록 인공 신경망을 조물조물, 주물주물 학습한다는 의미입니다. `fit` 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. `epochs`는 학습 횟수를 의미하며, 여기서는 5회 학습을 수행하도록 합니다. 일반적으로 학습 횟수에 따라 인공 신경망 근사 함수가 정확해 집니다.

16 : `model.predict` 함수를 호출하여 인공 신경망을 사용합니다. 여기서는 학습이 끝난 인공 신경망 함수에 10.0 값을 주어 그 결과를 예측하도록 합니다. 예측한 결과 값은 p 변수로 받습니다.


18 : `print` 함수를 호출하여 예측한 결과 값을 출력합니다.

3. 다음은 구글 코랩에 작성한 화면입니다.


```

1 import tensorflow as tf
2 import numpy as np
3
4 xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0])
5 ys = np.array([-2.0, 1.0, 4.0, 7.0, 10., 13.])
6
7 model = tf.keras.Sequential([
8     tf.keras.layers.InputLayer(input_shape=(1,)),
9     tf.keras.layers.Dense(1)
10 ])
11
12 model.compile(optimizer='sgd', loss='mean_squared_error')
13
14 model.fit(xs, ys, epochs=5)
15
16 p = model.predict([10.0])
17
18 print('p:', p)

```

4.  버튼을 눌러 프로그램을 실행시킵니다. 다음과 같이 학습이 진행되는 것을 확인합니다.

```

1 Epoch 1/5
  1/1 [=====] - 0s 243ms/step - loss: 21.3527
Epoch 2/5
  1/1 [=====] - 0s 7ms/step - loss: 16.8172
Epoch 3/5
  1/1 [=====] - 0s 5ms/step - loss: 13.2486
Epoch 4/5
  1/1 [=====] - 0s 10ms/step - loss: 10.4406
Epoch 5/5
  1/1 [=====] - 0s 14ms/step - loss: 8.2310
WARNING:tensorflow:7 out of the last 14 calls to <function Model.make_r
3 p: [[21.39265]]

```

- ❶ model.fit 함수 내에서 5회 학습이 수행됩니다.
- ❷ loss는 오차를 나타냅니다. 학습이 진행될수록 오차가 줄어드는 것을 확인합니다. 오차에 대해서는 뒤에서 자세히 살펴봅니다.
- ❸ model.predict 함수를 수행한 결과 값입니다. 입력 값 10.0에 대하여 21.39265를 출력합니다. 우리는 31에 가까운 값이 출력되기를 기대하고 있습니다.


5. 예제를 다음과 같이 수정합니다.

```

14 model.fit(xs, ys, epochs=50)

```

14 : 학습을 50회 수행시켜 봅니다.

6.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 마지막 5회 학습의 내용입니다.

```

1/1 [=====] - 0s 6ms/step - loss: 0.0016
Epoch 46/50
1/1 [=====] - 0s 5ms/step - loss: 0.0013
Epoch 47/50
1/1 [=====] - 0s 5ms/step - loss: 9.9620e-04
Epoch 48/50
1/1 [=====] - 0s 6ms/step - loss: 7.9501e-04
Epoch 49/50
1/1 [=====] - 0s 4ms/step - loss: 6.3650e-04
Epoch 50/50
1/1 [=====] - 0s 4ms/step - loss: 5.1155e-04
WARNING:tensorflow:8 out of the last 15 calls to <function Model.make_r
p: [[30.94326]]


```

입력 값 10.0에 대하여 30.94326를 출력합니다. 31에 충분히 가까운 값이 출력되는 것을 볼 수 있습니다. 훈련이 진행되면서 손실은 더 작아집니다.

7. 예제를 다음과 같이 수정합니다.

```
14 model.fit(xs, ys, epochs=500)
```

14 : 학습을 500회 수행시켜 봅니다.

8.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 마지막 5회 학습의 내용입니다.

```

Epoch 496/500
1/1 [=====] - 0s 5ms/step - loss: 8.3604e-07
Epoch 497/500
1/1 [=====] - 0s 7ms/step - loss: 8.1895e-07
Epoch 498/500
1/1 [=====] - 0s 6ms/step - loss: 8.0207e-07
Epoch 499/500
1/1 [=====] - 0s 17ms/step - loss: 7.8544e-07
Epoch 500/500
1/1 [=====] - 0s 5ms/step - loss: 7.6956e-07
WARNING:tensorflow:9 out of the last 16 calls to <function Model.make_r
p: [[31.00256]]

```

입력 값 10.0에 대하여 31.00256을 출력합니다. 31에 더 가까워진 값이 출력되는 것을 볼 수 있습니다. 학습을 500회 수행했을 때, 31에 충분히 가까운 결과 값이 출력되는 것을 볼 수 있습니다. 여기서 학습시킨 인공 신경망 함수는 $Y=3 \times X + 1$ 함수를 흉내 내는 근사함수입니다.

06 축하합니다!

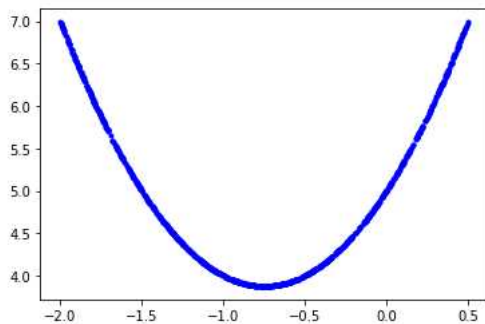
믿거나 말거나, 독자 여러분은 ML에 있는 대부분의 개념을 살펴보았습니다. 앞으로 독자 여러분은 훨씬 더 복잡한 데이터에 대해서도 여기서 배운 ML을 사용할 것입니다. 여기서 독자 여러분은 ❶ `np.array` 함수를 이용하여 X, Y 데이터를 준비해 보고, ❷ `tf.keras.Sequential` 클래스를 이용하여 인공 신경망을 정의해 보았으며, ❸ `model.compile` 함수를 호출하여 인공 신경망을 구성해 보았습니다. 그 과정에서 ❹ 손실 함수 `mean_squared_error`와 ❺ 최적화 함수 `sgd`를 사용해 보았습니다. 그리고 ❻ `model.fit` 함수를 호출하여 X, Y 데이터에 대한 학습을 수행해 보았습니다. 마지막으로 ❼ `model.predict` 함수를 호출하여 새로운 값 10.0 값에 대한 예측을 수행해 보았습니다. 이 과정이 인공 신경망과 관련된 전체적인 과정들입니다. 앞으로 독자 여러분은 이 과정들을 이용하여 복잡한 데이터에 대한 인공 신경망도 다루게 됩니다.

03 인공 신경망과 근사 함수

인공 신경망 함수는 일반적으로 인공 신경망 모델이라고 합니다. 모델은 우리말로 모형을 의미하며, 모형 함수로 이해할 수 있습니다. 모형 함수는 어떤 함수를 흉내 내는 함수를 말하며, 근사 함수라고 합니다. 여기서는 독자 여러분이 중·고등학교 때 배운 여러 가지 수학 함수를 인공 신경망을 이용하여 학습시켜 보며 인공 신경망의 근사 함수 특징을 이해해봅니다.

01 2차 함수 근사해 보기

여기서는 먼저 다음 2차 함수를 근사하는 인공 신경망 함수를 생성해 봅니다.



$$y = 2x^2 + 3x + 5 \quad (-2 \leq x \leq 0.5)$$

x 좌표의 범위는 -2에서 0.5까지입니다.

2차 함수 그리기

1. 다음과 같이 [+ Code] 버튼을 누릅니다. 실행 창의 경계에 마우스 커서를 대면 버튼이 나타납니다.



2. 다음과 같이 예제를 작성합니다.

131_1.py

```
01 import numpy as np
02 import time
03 import matplotlib.pyplot as plt
04
05 NUM_SAMPLES = 1000
```

```

06
07 np.random.seed(int(time.time()))
08
09 xs = np.random.uniform(-2, 0.5, NUM_SAMPLES)
10 np.random.shuffle(xs)
11 print(xs[:5])
12
13 ys = 2*xs**2 + 3*xs + 5
14 print(ys[:5])
15
16 plt.plot(xs, ys, 'b.')
17 plt.show()

```

01 : import문을 이용하여 numpy 모듈을 np라는 이름으로 불러옵니다. 여기서는 numpy 모듈을 이용하여 07, 09, 10, 13 줄에서 x, y 값의 집합을 동시에 처리합니다.

02 : import문을 이용하여 time 모듈을 불러옵니다. 07줄에서 임의 숫자(난수) 생성 초기화에 사용합니다.

03 : import문을 이용하여 matplotlib.pyplot 모듈을 plt라는 이름으로 불러옵니다. 여기서는 matplotlib.pyplot 모듈을 이용하여 16, 17줄에서 그래프를 그립니다.

05 : NUM_SAMPLES 변수를 생성한 후, 1000으로 초기화합니다. NUM_SAMPLES 변수는 생성할 데이터의 개수 값을 가지는 변수입니다.

07 : np.random.seed 함수를 호출하여 임의 숫자 생성을 초기화합니다. time.time 함수를 호출하여 현재 시간을 얻어낸 후, 정수 값으로 변환하여 np.random.seed 함수의 인자로 줍니다. 이렇게 하면 현재 시간에 맞춰 임의 숫자 생성이 초기화됩니다.

09 : np.random.uniform 함수를 호출하여 (-2, 0.5) 범위에서 NUM_SAMPLES 만큼의 임의 값을 차례대로 고르게 추출하여 xs 변수에 저장합니다.

10 : np.random.shuffle 함수를 호출하여 임의 추출된 x 값을 섞어줍니다. 이렇게 하면 임의로 추출된 x 값의 순서가 뒤섞이게 됩니다. 인공 신경망 학습 시에 데이터는 임의 순서로 입력되는 것이 중요합니다. 데이터가 임의 순서로 입력될 때 모델의 정확도가 높아지기 때문입니다.

11 : print 함수를 호출하여 xs에 저장된 값 중, 앞에서 5개까지 출력합니다. xs[:5]는 xs 리스트의 0번 항목부터 시작해서 5번 항목 미만인 4번 항목까지를 의미합니다.

13 : 다음 식을 이용하여 추출된 x 값에 해당하는 y 값을 얻어내어 ys 변수에 저장합니다. y 값도 NUM_SAMPLES 개수만큼 추출됩니다.


$$y = 2x^2 + 3x + 5$$

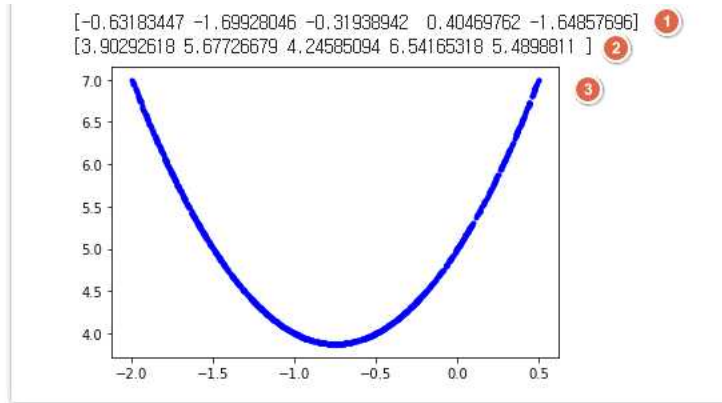
파이썬에서 *는 곱셈기호, **는 거듭제곱기호를 나타냅니다.

14 : print 함수를 호출하여 ys에 저장된 값 중, 앞에서 5개까지 출력합니다.

16 : plt.plot 함수를 호출하여 xs, ys 좌표 값에 맞추어 그래프를 내부적으로 그립니다. 그래프의 색깔은 파란색으로 그립니다. 'b.'은 파란색을 의미합니다.

17 : plt.show 함수를 호출하여 화면에 그래프를 표시합니다.

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



- ❶ xs에 저장된 값 중 앞에서 5개까지 출력 결과입니다. 이 값은 실행할 때마다 달라집니다.
- ❷ ys에 저장된 값 중 앞에서 5개까지 출력 결과입니다. 이 값은 실행할 때마다 달라집니다.
- ❸ $y = 2x^2 + 3x + 5$ 함수의 $(-2, 0.5)$ 범위에서의 그래프입니다.

실제 데이터 생성하기

이번엔 y값을 일정한 범위에서 위아래로 흩뜨려 실제 데이터에 가깝게 만들어 봅시다. 이 과정은 y값에 잡음을 섞어 실제 데이터에 가깝게 만드는 과정입니다.

1. 다음과 같이 예제를 수정합니다.

131_2.py

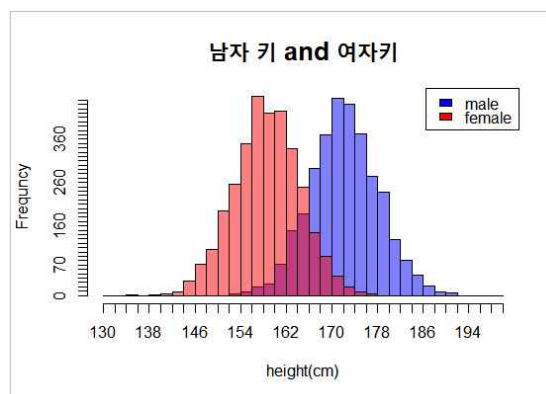
```
01 import numpy as np
02 import time
03 import matplotlib.pyplot as plt
04
05 NUM_SAMPLES = 1000
06
07 np.random.seed(int(time.time()))
08
09 xs = np.random.uniform(-2, 0.5, NUM_SAMPLES)
10 np.random.shuffle(xs)
11 print(xs[:5])
12
13 ys = 2*xs**2 + 3*xs + 5
14 print(ys[:5])
15
```

```

16 plt.plot(xs, ys, 'b.')
17 plt.show()
18
19 ys += 0.1*np.random.randn(NUM_SAMPLES)
20
21 plt.plot(xs, ys, 'g.')
22 plt.show()

```

19 : np.random.randn 함수를 호출하여 정규분포에 맞춰 임의 숫자를 NUM_SAMPLES의 개수만큼 생성합니다. 정규분포는 가우스분포라고도 하며, 종 모양과 같은 형태의 자연적인 분포 곡선입니다. 예를 들어, 키의 분포나 체중의 분포와 같이 자연적인 분포를 의미합니다.




저작권 확인 요망!

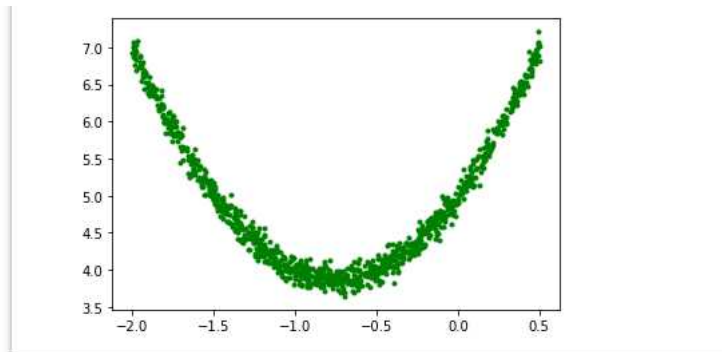
<출처 : <https://hsm-edu.tistory.com/1015>>

생성된 숫자에 0.1을 곱해 ys에 더해줍니다. 이렇게 하면 ys값은 원래 값을 기준으로 상하로 퍼진 형태의 자연스런 값을 갖게 됩니다.

21 : plt.plot 함수를 호출하여 xs, ys 좌표 값에 맞추어 그래프를 내부적으로 그립니다. 그래프의 색깔은 초록색으로 그립니다. 'g.'은 초록색을 의미합니다.

22 : plt.show 함수를 호출하여 화면에 그래프를 표시합니다.

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



마지막에 표시된 그래프의 모양이 원래 모양에서 상하로 퍼진 형태로 나타나게 됩니다. 여기

서 생성된 데이터는 인공 신경망 학습에 사용되며 원래 곡선에 가까운 근사 곡선을 생성하는 인공 신경망 함수를 만들게 됩니다.

훈련, 실험 데이터 분리하기

여기서는 앞에서 생성한 x, y 데이터를 훈련 데이터와 실험 데이터로 분리해 봅니다. 훈련 데이터는 인공 신경망을 학습시키는데 사용하는 데이터이며, 실험 데이터는 학습이 잘 되었는지 확인하는 데이터로 사용합니다.

1. 다음과 같이 예제를 수정합니다.

131_3.py

```
01 import numpy as np
02 import time
03 import matplotlib.pyplot as plt
04
05 NUM_SAMPLES = 1000
06
07 np.random.seed(int(time.time()))
08
09 xs = np.random.uniform(-2, 0.5, NUM_SAMPLES)
10 np.random.shuffle(xs)
11 print(xs[:5])
12
13 ys = 2*xs**2 + 3*xs + 5
14 print(ys[:5])
15
16 plt.plot(xs, ys, 'b.')
17 plt.show()
18
19 ys += 0.1*np.random.randn(NUM_SAMPLES)
20
21 plt.plot(xs, ys, 'g.')
22 plt.show()
23
24 NUM_SPLIT = int(0.8*NUM_SAMPLES)
25
26 x_train, x_test = np.split(xs, [NUM_SPLIT])
27 y_train, y_test = np.split(ys, [NUM_SPLIT])
28
```



```

29 plt.plot(x_train, y_train, 'b.', label='train')
30 plt.plot(x_test, y_test, 'r.', label='test')
31 plt.legend()
32 plt.show()

```

24 : NUM_SAMPLES에 0.8을 곱한 후, 정수로 변경하여 NUM_SPLIT 변수에 할당합니다. 현재 예제의 경우 NUM_SPLIT 변수는 800의 값을 가집니다. 1000개의 x, y 데이터 값 중 800개는 훈련 데이터로, 200개는 실험 데이터로 사용합니다.

26 : np.split 함수를 호출하여 1000개의 값을 가진 xs를 800개, 200개로 나누어 각각 x_train, x_test에 할당합니다. x_train 변수는 1000개의 값 중 앞부분 800개의 값을 할당받고 x_test 변수는 나머지 200개의 값을 할당받습니다.


27 : np.split 함수를 호출하여 1000개의 값을 가진 ys를 800개, 200개로 나누어 각각 y_train, y_test에 할당합니다. y_train 변수는 1000개의 값 중 앞부분 800개의 값을 할당받고 y_test 변수는 나머지 200개의 값을 할당받습니다.

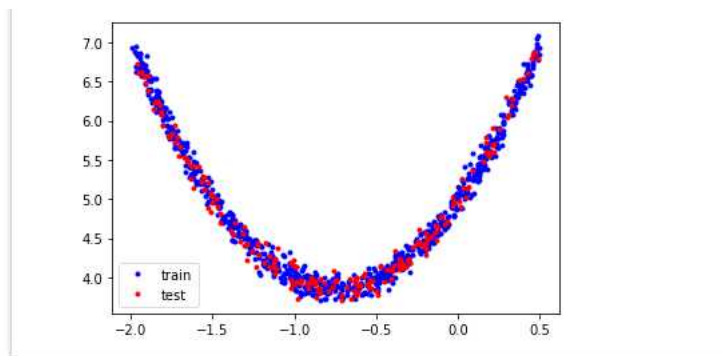
29 : plt.plot 함수를 호출하여 x_train, y_train 좌표 값에 맞추어 그래프를 내부적으로 그립니다. 그래프의 색깔은 파란색으로 그립니다. 'b.'은 파란색을 의미합니다. label 매개변수에는 'train' 문자열을 넘겨줍니다. 이 문자열은 31줄에 있는 plt.legend 함수에 의해 그래프에 표시됩니다.

30 : plt.plot 함수를 호출하여 x_test, y_test 좌표 값에 맞추어 그래프를 내부적으로 그립니다. 그래프의 색깔은 빨간색으로 그립니다. 'r.'은 빨간색을 의미합니다. label 매개변수에는 'test' 문자열을 넘겨줍니다. 이 문자열은 31줄에 있는 plt.legend 함수에 의해 그래프에 표시됩니다.

31 : plt.legend 함수를 호출하여 범례를 표시합니다.

32 : plt.show 함수를 호출하여 화면에 그래프를 표시합니다.

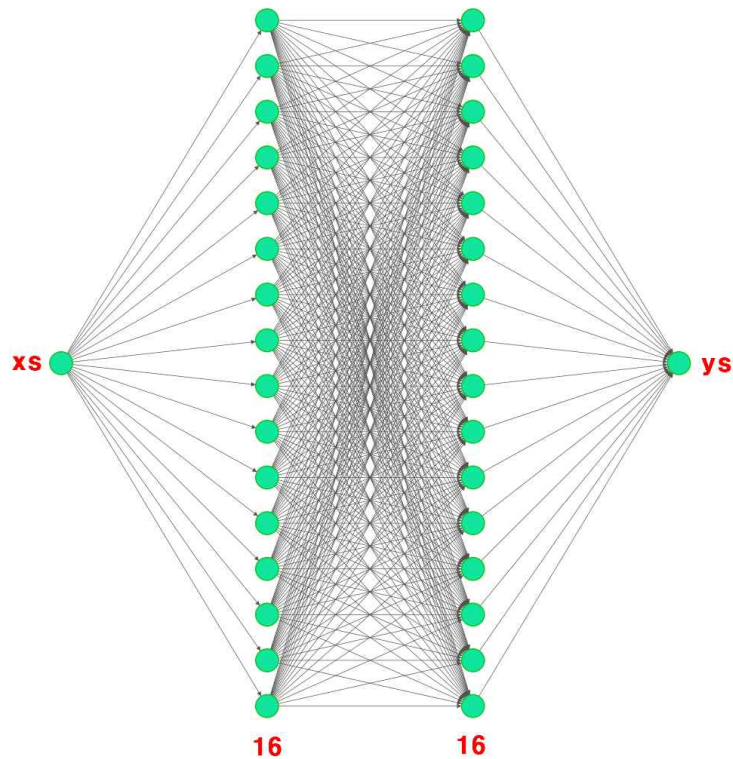
2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



파란색 점은 x_train, y_train의 분포를 나타내며, 빨간색 점은 x_test, y_test의 분포를 나타냅니다. x_train, y_train 데이터는 인공 신경망 학습에 사용되며 원래 곡선에 가까운 근사 곡선을 생성하는 인공 신경망 함수를 만들게 됩니다. x_test, y_test 데이터는 학습이 끝난 인공 신경망 함수를 시험하는데 사용합니다.

인공 신경망 구성하기

이번엔 인공 신경망 함수를 구성한 후, 학습을 수행하지 않은 상태로 시험 데이터를 이용하여 예측을 수행한 후, 그래프를 그려봅니다. 여기서는 다음과 같은 모양의 인공 신경망을 구성합니다. 입력 층 x_s , 출력 층 y_s 사이에 단위 인공 신경 16개로 구성된 은닉 층 2개를 추가하여 인공 신경망을 구성합니다.



1. 다음과 같이 예제를 수정합니다.

131_4.py

```
01 import numpy as np
02 import time
03 import matplotlib.pyplot as plt
04
05 NUM_SAMPLES = 1000
06
07 np.random.seed(int(time.time()))
08
09 xs = np.random.uniform(-2, 0.5, NUM_SAMPLES)
10 np.random.shuffle(xs)
11 print(xs[:5])
12
13 ys = 2*xs**2 + 3*xs + 5
```

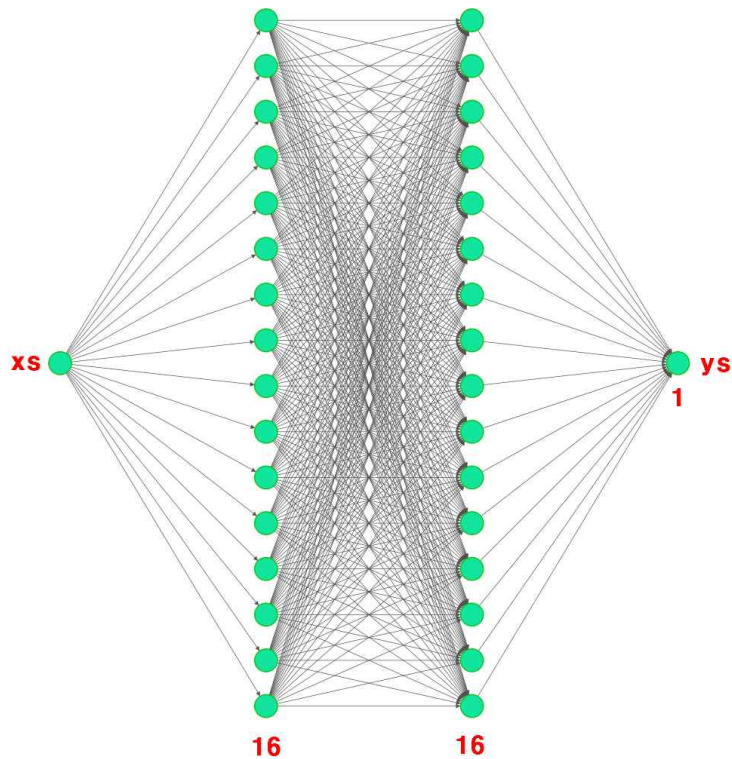
```

14 print(ys[:5])
15
16 plt.plot(xs, ys, 'b.')
17 plt.show()
18
19 ys += 0.1*np.random.randn(NUM_SAMPLES)
20
21 plt.plot(xs, ys, 'g.')
22 plt.show()
23
24 NUM_SPLIT = int(0.8*NUM_SAMPLES)
25
26 x_train, x_test = np.split(xs, [NUM_SPLIT])
27 y_train, y_test = np.split(ys, [NUM_SPLIT])
28
29 plt.plot(x_train, y_train, 'b.', label='train')
30 plt.plot(x_test, y_test, 'r.', label='test')
31 plt.legend()
32 plt.show()
33
34 import tensorflow as tf
35
36 model_f = tf.keras.Sequential([
37     tf.keras.layers.InputLayer(input_shape=(1,)),
38     tf.keras.layers.Dense(16, activation='relu'),
39     tf.keras.layers.Dense(16, activation='relu'),
40     tf.keras.layers.Dense(1)
41 ])
42
43 model_f.compile(optimizer='rmsprop', loss='mse')
44
45 p_test = model_f.predict(x_test)
46
47 plt.plot(x_test, y_test, 'b.', label='actual')
48 plt.plot(x_test, p_test, 'r.', label='predicted')
49 plt.legend()
50 plt.show()

```

34 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

36~41 : tf.keras.Sequential 클래스를 이용하여 인공 신경망을 생성합니다. 여기서 생성한 인공 신경망의 모양은 다음과 같습니다.



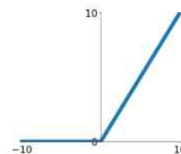
이 신경망은 33(=16+16+1)개의 인공 신경으로 구성됩니다. 입력 층에 표시된 노드는 입력 값의 개수를 표시하며 나머지 층에 있는 노드는 인공 신경을 나타냅니다. 인공 신경망의 내부 구조는 뒤에서 자세히 살펴봅니다. 생성된 인공 신경망은 일반적으로 모델이라고 합니다. 모델은 모형을 의미하며, 주어진 데이터에 맞추어진 원래 함수를 흉내 내는 함수인 근사 함수를 의미합니다. `model_f`는 모델 함수를 의미하는 변수입니다.

36, 41 : 파이썬의 리스트를 나타냅니다.

37 : `tf.keras.layers.InputLayer` 함수를 이용하여 내부적으로 keras 라이브러리에서 제공하는 `tensor`를 생성하고, 입력 노드의 개수를 정해줍니다. `tensor`는 3차원 이상의 행렬을 의미하며, 인공 신경망 구성 시 사용하는 자료 형입니다.

38, 39 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서는 각 층별로 단위 인공 신경 16개를 생성합니다. `activation`은 활성화 함수를 의미하며 여기서는 'relu' 함수를 사용합니다. 다음은 relu 함수를 나타냅니다.

ReLU
 $\max(0, x)$



활성화 함수와 'relu' 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다.

여기서 `Dense`는 내부적으로 $y = \text{activation}(x \cdot w + b)$ 식을 생성하게 됩니다. 이 식에 대해서는 뒤에서 실제로 구현해 보며 그 원리를 살펴보도록 합니다.

40 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서는 단위 인공 신경 1개를 생성합니다. 마지막에 생성한 신경망 층은 출력 신경망이 됩니다.

43 : `model_f.compile` 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망

을 구성할 때에는 적어도 2개의 함수를 정해야 합니다. loss 함수와 optimizer 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수와 최적화 함수에 대해서는 뒤에서 자세히 살펴봅니다. 손실 함수로는 mse 함수를 사용하고 최적화 함수는 rmsprop 함수를 사용합니다. mse, rmsprop 함수는 뒤에서 살펴보도록 합니다.


45 : model_f.predict 함수를 호출하여 인공 신경망을 사용해 봅니다. 여기서는 학습을 수행하지 않은 상태에서 인공 신경망 함수에 x_test 값을 주어 그 결과를 예측해 봅니다. 예측한 결과 값은 p_test 변수로 받습니다.

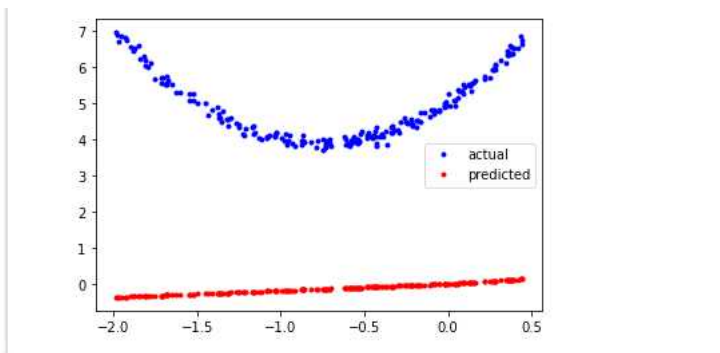
47 : plt.plot 함수를 호출하여 x_test, y_test 좌표 값에 맞추어 그래프를 내부적으로 그립니다. 그래프의 색깔은 파란색으로 그립니다. 'b.'은 파란색을 의미합니다. label 매개변수에는 'actual' 문자열을 넘겨줍니다. 이 문자열은 49줄에 있는 plt.legend 함수에 의해 그래프에 표시됩니다.

48 : plt.plot 함수를 호출하여 x_test, p_test 좌표 값에 맞추어 그래프를 내부적으로 그립니다. 그래프의 색깔은 빨간색으로 그립니다. 'r.'은 빨간색을 의미합니다. label 매개변수에는 'predicted' 문자열을 넘겨줍니다. 이 문자열은 49줄에 있는 plt.legend 함수에 의해 그래프에 표시됩니다.

49 : plt.legend 함수를 호출하여 범례를 표시합니다.

50 : plt.show 함수를 호출하여 화면에 그래프를 표시합니다.

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



파란색 점은 x_test, y_test의 분포를 나타내며, 빨간색 점은 x_test, p_test의 분포를 나타냅니다. 인공 신경망이 학습을 수행하기 전 상태라 x_test 값에 대한 예측 값을 정확히 생성해 내지 못하는 것을 볼 수 있습니다.

인공 신경망 학습시키기

이번엔 인공 신경망 함수를 학습 시킨 후, 시험 데이터를 이용하여 예측을 수행하고 그래프를 그려봅니다.

1. 다음과 같이 예제를 수정합니다.

131_5.py

```
01 import numpy as np
02 import time
03 import matplotlib.pyplot as plt
04
05 NUM_SAMPLES = 1000
06
07 np.random.seed(int(time.time()))
08
09 xs = np.random.uniform(-2, 0.5, NUM_SAMPLES)
10 np.random.shuffle(xs)
11 print(xs[:5])
12
13 ys = 2*xs**2 + 3*xs + 5
14 print(ys[:5])
15
16 plt.plot(xs, ys, 'b.')
17 plt.show()
18
19 ys += 0.1*np.random.randn(NUM_SAMPLES)
20
21 plt.plot(xs, ys, 'g.')
22 plt.show()
23
24 NUM_SPLIT = int(0.8*NUM_SAMPLES)
25
26 x_train, x_test = np.split(xs, [NUM_SPLIT])
27 y_train, y_test = np.split(ys, [NUM_SPLIT])
28
29 plt.plot(x_train, y_train, 'b.', label='train')
30 plt.plot(x_test, y_test, 'r.', label='test')
31 plt.legend()
32 plt.show()
33
34 import tensorflow as tf
35
36 model_f = tf.keras.Sequential([
37     tf.keras.layers.InputLayer(input_shape=(1,)),
38     tf.keras.layers.Dense(16, activation='relu'),
39     tf.keras.layers.Dense(16, activation='relu'),
40     tf.keras.layers.Dense(1)
```

```


41 ])
42
43 model_f.compile(optimizer='rmsprop', loss='mse')
44
45 p_test = model_f.predict(x_test)
46
47 plt.plot(x_test, y_test, 'b.', label='actual')
48 plt.plot(x_test, p_test, 'r.', label='predicted')
49 plt.legend()
50 plt.show()
51
52 model_f.fit(x_train, y_train, epochs=600)
53
54 p_test = model_f.predict(x_test)
55
56 plt.plot(x_test, y_test, 'b.', label='actual')
57 plt.plot(x_test, p_test, 'r.', label='predicted')
58 plt.legend()
59 plt.show()

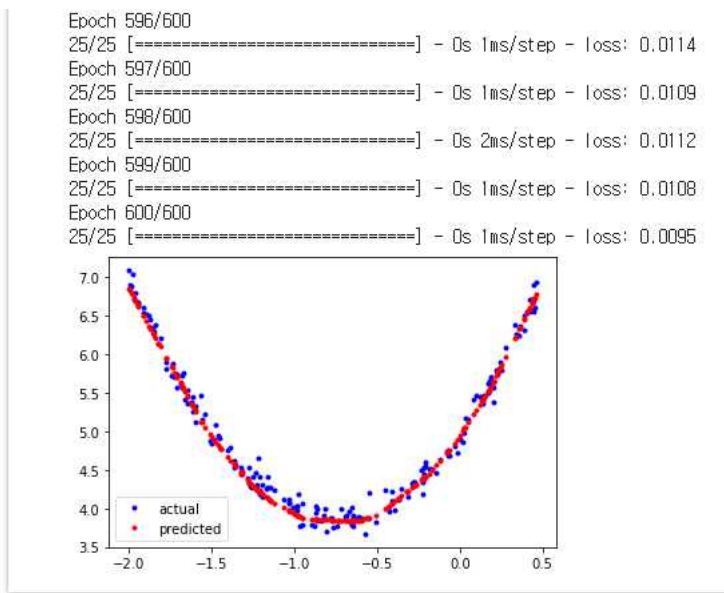
```

52 : model_f.fit 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. fit 함수에는 x_train, y_train 데이터가 입력이 되는데 인공 신경망을 x_train, y_train 데이터에 맞도록 학습한다는 의미를 갖습니다. 즉, x_train, y_train 데이터에 맞도록 인공 신경망을 조물조물, 주물주물 학습한다는 의미입니다. fit 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. epochs는 학습 횟수를 의미하며, 여기서는 600회 학습을 수행하도록 합니다. 일반적으로 학습 횟수에 따라 인공 신경망 근사 함수가 정확해 집니다.

54 : model_f.predict 함수를 호출하여 인공 신경망을 사용합니다. 여기서는 학습이 끝난 인공 신경망 함수에 x_test 값을 주어 그 결과를 예측해 봅니다. 예측한 결과 값은 p_test 변수로 받습니다.

56~59 : 47~50줄에서와 같은 방법으로 그래프를 그립니다.

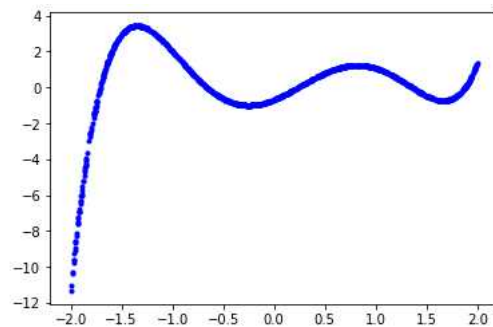
2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



파란색 점은 x_{test} , y_{test} 의 분포를 나타내며, 빨간색 점은 x_{test} , p_{test} 의 분포를 나타냅니다. 인공 신경망이 학습을 수행한 이후에는 x_{test} 값에 대한 예측 값을 실제 함수에 근사해서 생성해 내는 것을 볼 수 있습니다.

02 5차 함수 근사해 보기

이번에는 다음과 같은 5차 함수를 근사하도록 인공 신경망 함수를 학습시켜 봅니다.



$$y = (x + 1.7)(x + 0.7)(x - 0.3)(x - 1.3)(x - 1.9) + 0.2$$

$$(-2 \leq x \leq 2)$$

x 좌표의 범위는 -2에서 2까지입니다.

1. 이전 예제의 09줄을 다음과 같이 수정합니다.

```
09 xs = np.random.uniform(-2, 2, NUM_SAMPLES)
```


09 : `np.random.uniform` 함수를 호출하여 $(-2, 2)$ 범위에서 `NUM_SAMPLES` 만큼의 임의 값을 차례대로 고르게 추출하여 `xs` 변수에 저장합니다.

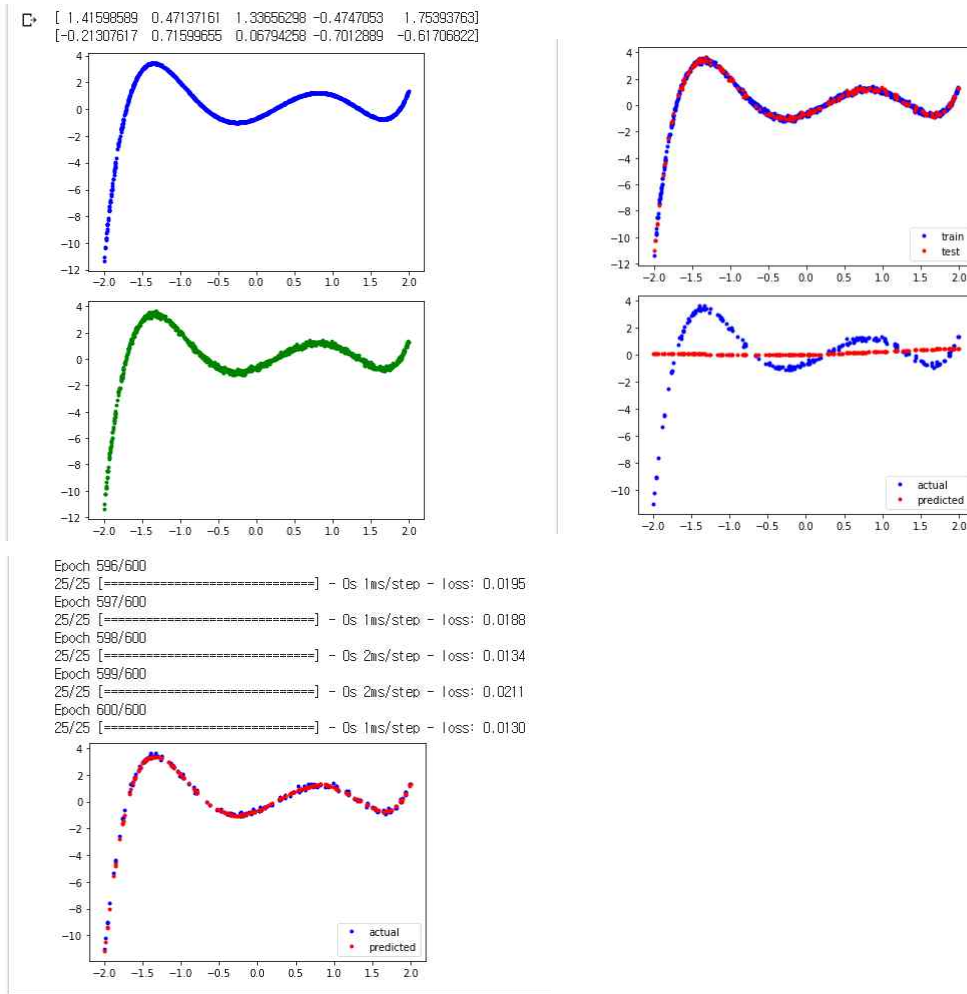
2. 계속해서 13줄을 다음과 같이 수정합니다.

13 ys = (xs+1.7)*(xs+0.7)*(xs-0.3)*(xs-1.3)*(xs-1.9)+0.2

13 : 다음 식을 이용하여 추출된 x 값에 해당하는 y 값을 얻어내어 ys 변수에 저장합니다. y 값도 NUM_SAMPLES 개수만큼 추출됩니다.

$$y = (x + 1.7)(x + 0.7)(x - 0.3)(x - 1.3)(x - 1.9) + 0.2$$

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



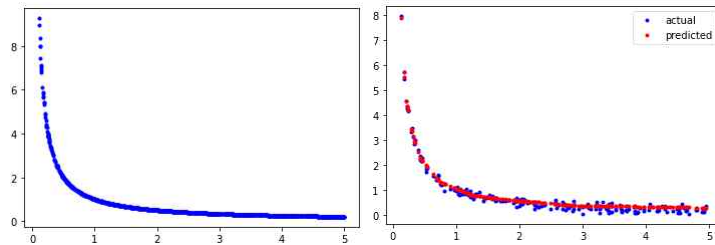
인공 신경망이 학습을 수행한 이후에는 x_test 값에 대한 예측 값을 실제 함수에 근사해서 생성해 내는 것을 볼 수 있습니다.

03 다양한 함수 근사해 보기

여기서는 독자 여러분이 이전과 같이 예제를 수정해 가며, 중·고등학교 때 배운 함수들을 인공 신경망을 학습시켜 근사시켜 봅니다.

분수 함수 근사해 보기

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



$$y = \frac{1}{x} \quad (0.1 \leq x \leq 5)$$

x 좌표의 범위는 0.1에서 5까지입니다. 분수함수의 경우 x 값 0에 대해 정의되지 않습니다.

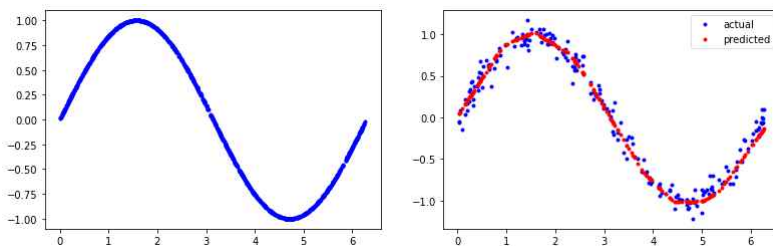
이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(0.1, 5, NUM_SAMPLES)
```

```
13 ys = 1.0/xs
```

sin 함수 근사해 보기

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



$$y = \sin(x) \quad (0 \leq x \leq 2\pi)$$

x 좌표의 범위는 0에서 2π 까지입니다.

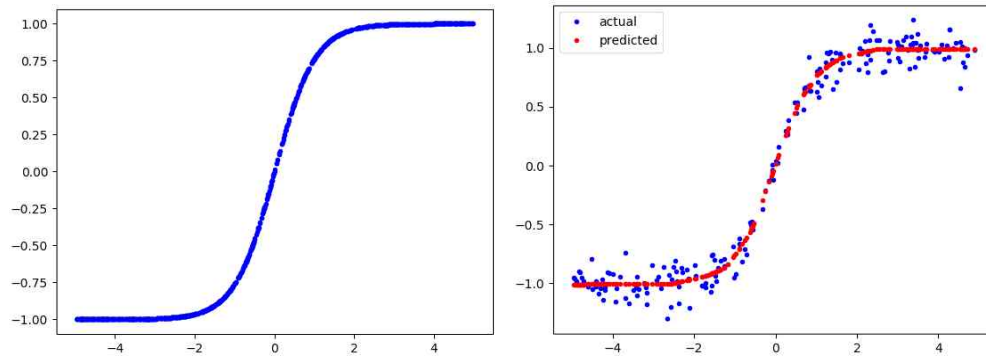
이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(0, 2*np.pi, NUM_SAMPLES)
```

```
13 ys = np.sin(xs)
```

tanh 함수 근사해 보기

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



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

x 좌표의 범위는 -5에서 5까지입니다.

이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

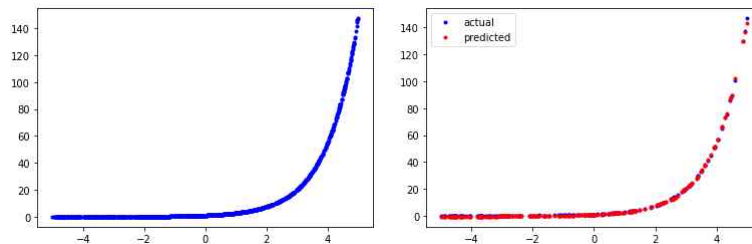
```
09 xs = np.random.uniform(-5, 5, NUM_SAMPLES)
```

```
13 ys = np.tanh(xs)
```

*** tanh 함수는 인공 신경망의 활성화 함수로 사용하는 함수중 하나입니다.

e 지수함수 근사해 보기

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



$$y = e^x \quad (-5 \leq x \leq 5)$$

x 좌표의 범위는 -5에서 5까지입니다.

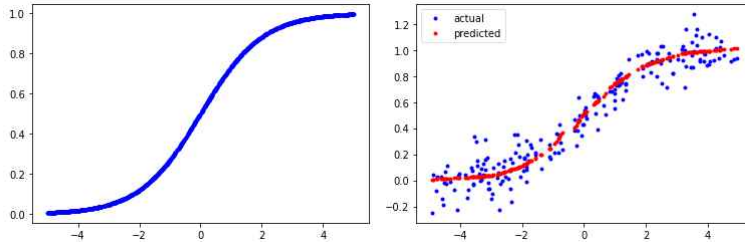
이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(-5, 5, NUM_SAMPLES)
```

```
13 ys = np.exp(xs)
```

sigmoid 함수 근사해 보기

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



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

x 좌표의 범위는 -5에서 5까지입니다.

이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

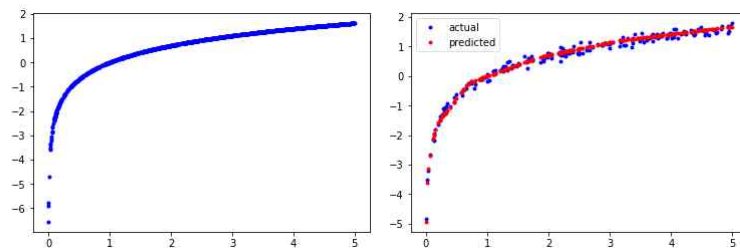
```
09 xs = np.random.uniform(-5, 5, NUM_SAMPLES)
```

```
13 ys = 1.0/(1.0+np.exp(-xs))
```

*** sigmoid 함수는 인공 신경망의 활성화 함수로 사용하는 함수중 하나입니다.

로그함수 근사해 보기

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



$$y = \log(x) \quad (0 \leq x \leq 5)$$

x 좌표의 범위는 0에서 5까지입니다. log함수의 경우 음수 x 값에 대해 정의되지 않습니다.

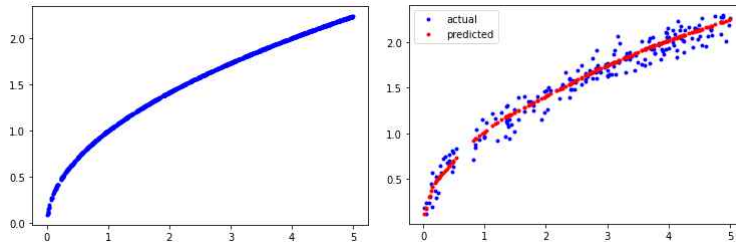
이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(0, 5, NUM_SAMPLES)
```

```
13 ys = np.log(xs)
```

제곱근 함수 근사해 보기

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



$$y = \sqrt{x} \quad (0 \leq x \leq 5)$$

x 좌표의 범위는 0.1에서 5까지입니다. 제곱근함수의 경우 음수 x 값에 대해 정의되지 않습니다.

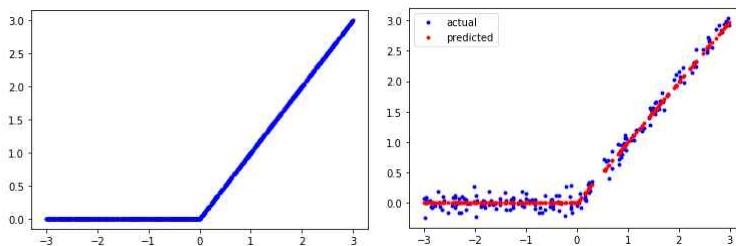
이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(0, 5, NUM_SAMPLES)
```

```
13 ys = np.sqrt(xs)
```

relu 함수 근사해 보기

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



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

x 좌표의 범위는 -3에서 3까지입니다.

이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(-3, 3, NUM_SAMPLES)
```

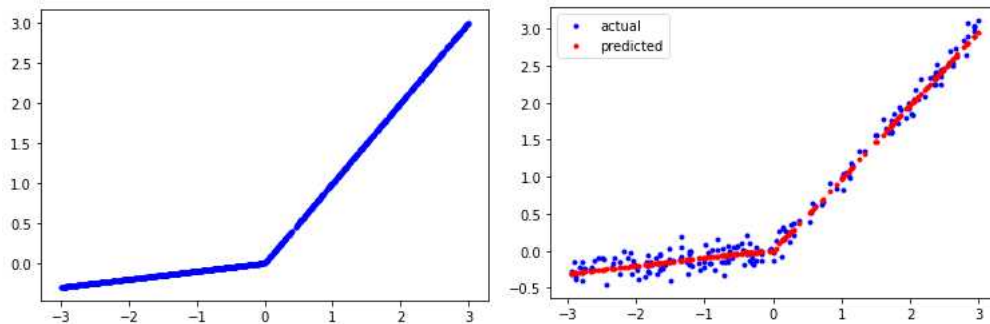
```
13 ys = (xs>0)*xs
```

13 : xs>0 연산의 경우 xs가 0보다 큰 항목은 1이 되고, 그렇지 않은 경우는 0이 됩니다. 그래서 xs가 0보다 큰 항목은 ys = xs가 되며, 그렇지 않은 경우는 ys = 0이 됩니다.

*** relu 함수는 인공 신경망의 활성화 함수로 사용하는 함수중 하나입니다.

leaky relu 함수 근사해 보기

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



$$y = \begin{cases} x & (x > 0) \\ \alpha x & (x \leq 0) \end{cases} \quad (-3 \leq x \leq 3, \alpha = 0.1)$$

x 좌표의 범위는 -3에서 3까지입니다. 위 그래프에서 α 는 0.1입니다. 일반적으로 0.01을 사용합니다.

이전 예제를 다음과 같이 수정한 후, 테스트를 수행합니다.

```
09 xs = np.random.uniform(-3, 3, NUM_SAMPLES)
```

```
13 ys = (xs>0)*xs + (xs<=0)*0.1*xs
```

*** leaky relu 함수는 인공 신경망의 활성화 함수로 사용하는 함수중 하나입니다.

이상에서 독자 여러분이 중·고등학교 때 배운 함수들에 대해 인공 신경망을 학습시켜 근사 함수를 만들어 보았습니다. 또, 몇 가지 활성화 함수들에 대해서도 인공 신경망을 학습시켜 근사 함수를 만들어 보았습니다. 실제로 인공 신경망 함수는 앞에서 살펴본 함수로 표현하기 어려운 복잡한 형태의 입출력 데이터에 대한 근사 함수를 만들 때 사용합니다. 예를 들어, 자동차 번호판을 인식하는 함수라든지 사람이나 자동차를 인식하는 함수를 만들 때 사용합니다.

04 인공 신경망 소스 살펴보기

다음은 지금까지 실습한 인공 신경망 관련 루틴을 정리한 내용입니다.

134_1.py

```

01 import numpy as np
02 import time
03 import matplotlib.pyplot as plt
04
05 NUM_SAMPLES = 1000
06
07 np.random.seed(int(time.time()))
08
09 xs = np.random.uniform(0.1, 5, NUM_SAMPLES)
10 np.random.shuffle(xs)
11
12 ys = 1.0/xs
13
14 ys += 0.1*np.random.randn(NUM_SAMPLES)
15
16 NUM_SPLIT = int(0.8*NUM_SAMPLES)
17
18 x_train, x_test = np.split(xs, [NUM_SPLIT])
19 y_train, y_test = np.split(ys, [NUM_SPLIT])
20
21 import tensorflow as tf
22
23 model_f = tf.keras.Sequential([
24     tf.keras.layers.InputLayer(input_shape=(1,)),
25     tf.keras.layers.Dense(16, activation='relu'),
26     tf.keras.layers.Dense(16, activation='relu'),
27     tf.keras.layers.Dense(1)
28 ])
29
30 model_f.compile(optimizer='sgd', loss='mean_squared_error')
31
32 model_f.fit(x_train, y_train, epochs=600)
33
34 p_test = model_f.predict(x_test)
35
36 plt.plot(x_test, y_test, 'b.', label='actual')
37 plt.plot(x_test, p_test, 'r.', label='predicted')
38 plt.legend()
39 plt.show()

```

05~14 : 인공 신경망 학습에 사용할 데이터를 생성합니다.

16~19 : 데이터를 훈련 데이터와 실험 데이터로 나눕니다.

23~28 : 인공 신경망 구성에 필요한 입력 층, 은닉 층, 출력 층을 구성합니다.

30 : 인공 신경망 내부 망을 구성하고, 학습에 필요한 오차함수, 최적화함수를 설정합니다.

32 : 인공 신경망을 학습시킵니다.

34 : 학습시킨 인공 신경망을 이용하여 새로 들어온 데이터에 대한 예측을 수행합니다.

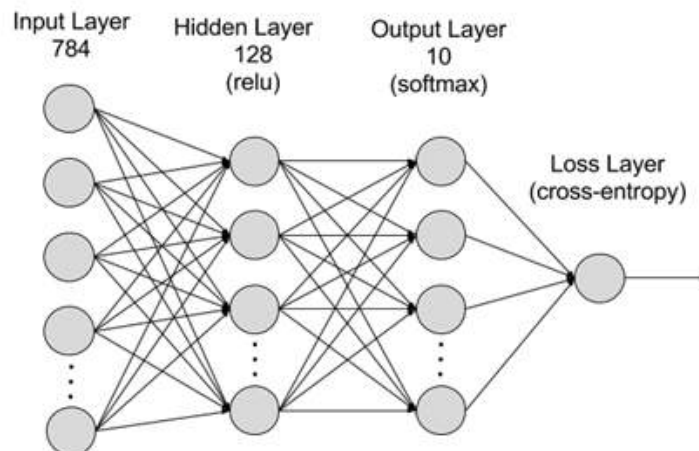
04 딥러닝 활용 맛보기

앞에서 우리는 중·고등학교 때 배운 함수들에 대해 인공 신경망을 학습시켜 근사 함수를 만들어 보았습니다. 실제로 인공 신경망 함수는 이러한 함수들로 표현하기 어려운 복잡한 형태의 입출력 데이터에 대한 근사 함수를 만들 때 사용합니다. 여기서는 인공 신경망을 학습시켜 숫자와 그림을 인식해 보도록 합니다. 일반적으로 인공 신경망 관련된 책에서 소개되는 예제를 수행해 보며, 인공 신경망이 어떻게 활용되는 지 살펴봅니다.

01 딥러닝 활용 예제 살펴보기



여기서는 MNIST라고 하는 손 글씨 숫자 데이터를 입력받아 학습을 수행하는 예제를 살펴봅니다. 여기서 소개하는 예제의 경우 독자 여러분은 구체적으로 어떤 데이터가 사용되는지 알기 어렵습니다. 데이터에 대해서는 다음 단원에서 자세히 살펴보도록 합니다. 여기서는 딥러닝 예제의 기본적인 구조를 살펴보도록 합니다. 다음과 같은 모양의 인공 신경망을 구성하고 학습시켜 봅니다.



1. 다음과 같이 예제를 작성합니다.

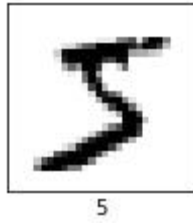
141_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train), (x_test, y_test) = mnist.load_data()
06 x_train, x_test = x_train / 255.0, x_test / 255.0
07 x_train, x_test = x_train.reshape((60000, 784)), x_test.reshape((10000, 784))
08
09 model = tf.keras.Sequential([
10     tf.keras.layers.InputLayer(input_shape=(784,)),
11     tf.keras.layers.Dense(128, activation='relu'),
12     tf.keras.layers.Dense(10, activation='softmax')
13 ])
14
15 model.compile(optimizer='adam',
16               loss='sparse_categorical_crossentropy',
17               metrics=['accuracy'])
18
19 model.fit(x_train, y_train, epochs=5)
20
21 model.evaluate(x_test, y_test)
```

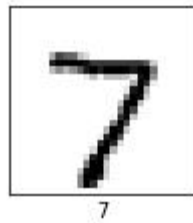
01 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

03 : mnist 변수를 생성한 후, tf.keras.datasets.mnist 모듈을 가리키게 합니다. mnist 모듈은 손 글씨 숫자 데이터를 가진 모듈입니다. mnist 모듈에는 6만개의 학습용 손 글씨 숫자 데이터와 1만개의 시험용 손 글씨 숫자 데이터가 있습니다. 이 데이터들에 대해서는 다음 단원에서 자세히 살펴봅니다.

05 : mnist.load_data 함수를 호출하여 손 글씨 숫자 데이터를 읽어와 x_train, y_train, x_test, y_test 변수가 가리키게 합니다. x_train, x_test 변수는 각각 6만개의 학습용 손 글씨 숫자 데이터와 1만개의 시험용 손 글씨 숫자 데이터를 가리킵니다. y_train, y_test 변수는 각각 6만개의 학습용 손 글씨 숫자 라벨과 1만개의 시험용 손 글씨 숫자 라벨을 가리킵니다. 예를 들어 x_train[0], y_train[0] 항목은 각각 다음과 같은 손 글씨 숫자 5에 대한 그림과 라벨 5를 가리킵니다.



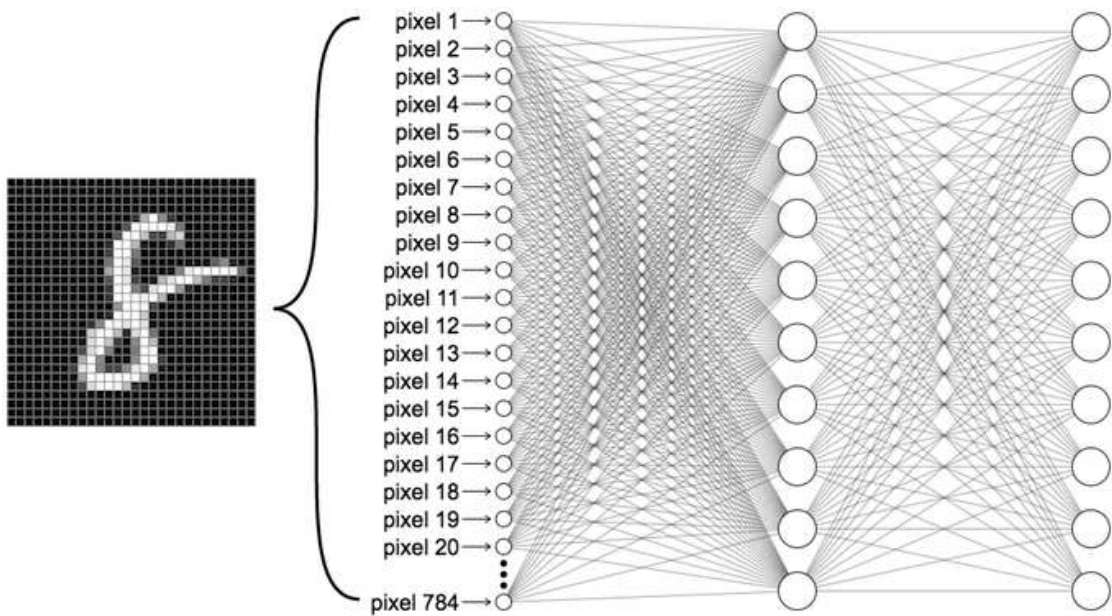
또, `x_test[0]`, `y_test[0]` 항목은 각각 다음과 같은 손 글씨 숫자 7에 대한 그림과 라벨 7을 가리킵니다.



06 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀로 구성된 그림이며, 1픽셀의 크기는 8비트로 0에서 255사이의 숫자를 가집니다. 모든 픽셀의 숫자를 255.0으로 나누어 각 픽셀을 0.0에서 1.0사이의 실수로 바꾸어 인공 신경망에 입력하게 됩니다. 다음은 x_train[0] 그림의 픽셀 값을 출력한 그림입니다.

[illegible]

07 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀, 28x28 픽셀로 구성되어 있습니다. 이 예제에서 소개하는 인공 신경망의 경우 그림 데이터를 입력할 때 28x28 픽셀을 784(=28x28) 픽셀로 일렬로 세워서 입력하게 됩니다. 그래서 10줄에 있는 InputLayer 클래스는 일렬로 세워진 784 픽셀을 입력으로 받도록 구성됩니다.



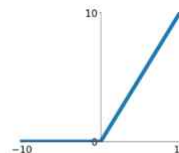
<출처 : <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>>

09~13 : `tf.keras.Sequential` 클래스를 이용하여 인공 신경망을 생성합니다. 여기서 생성한 인공 신경망은 138(=128+10)개의 인공 신경으로 구성됩니다. 입력 층에 표시된 노드는 입력 값의 개수를 표시하며 나머지 층에 있는 노드는 인공 신경을 나타냅니다. 인공 신경망의 내부 구조는 뒤에서 자세히 살펴봅니다. 생성된 인공 신경망은 일반적으로 모델이라고 합니다. 모델은 모형을 의미하며, 주어진 데이터에 맞추어진 원래 함수를 흉내 내는 함수인 근사 함수를 의미합니다. 여기서는 손 글씨 숫자를 구분하는 근사함수입니다.

10 : `tf.keras.layers.InputLayer` 함수를 이용하여 내부적으로 keras 라이브러리에서 제공하는 tensor를 생성하고, 입력 노드의 개수를 정해줍니다. tensor는 3차원 이상의 행렬을 의미하며, 인공 신경망 구성 시 사용하는 자료 형입니다. 여기서는 784개의 입력 노드를 생성합니다.

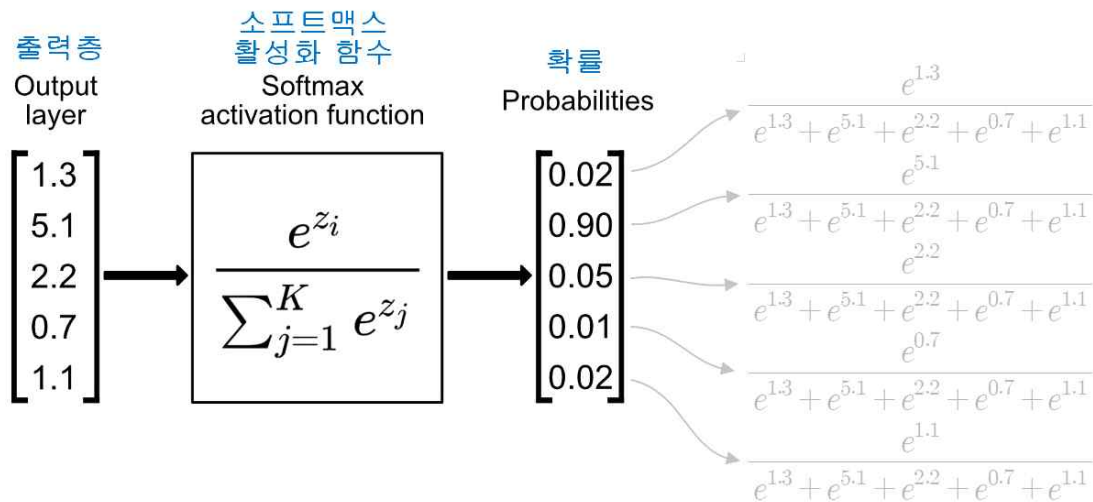
11 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서는 단위 인공 신경 128개를 생성합니다. activation은 활성화 함수를 의미하며 여기서는 'relu' 함수를 사용합니다. 다음은 relu 함수를 나타냅니다.

ReLU
 $\max(0, x)$



활성화 함수와 'relu' 함수에 대해서는 뒤에서 구현해 보면서 자세히 살펴보도록 합니다. 여기서 Dense는 내부적으로 $y = \text{activation}(x \cdot w + b)$ 식을 생성하게 됩니다. 이 식에 대해서는 뒤에서 실제로 구현해 보며 그 원리를 살펴보도록 합니다.

12 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서는 단위 인공 신경 10개를 생성합니다. activation은 활성화 함수를 의미하며 여기서는 'softmax' 함수를 사용합니다. 다음은 softmax 함수를 나타냅니다.



참고로 'softmax' 함수는 출력 층에서만 사용할 수 있는 활성화 함수입니다. 활성화 함수와 'softmax' 함수에 대해서는 뒤에서 구현해 보면서 자세히 살펴보도록 합니다.

15~17 : model.compile 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망을 구성할 때에는 적어도 2개의 함수를 정해야 합니다. loss 함수와 optimizer 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수와 최적화 함수에 대해서는 뒤에서 자세히 살펴보겠습니다. 손실 함수로는 sparse_categorical_crossentropy 함수를 사용하고 최적화 함수는 adam 함수를 사용합니다. sparse_categorical_crossentropy, adam 함수는 뒤에서 살펴보도록 합니다. fit 함수 로그에는 기본적으로 손실 값만 표시됩니다. metrics 매개 변수는 학습 측정 항목 함수를 전달할 때 사용합니다. 'accuracy'는 학습의 정확도를 출력해 줍니다.

19 : model.fit 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. fit 함수에는 x_train, y_train 데이터가 입력이 되는데 인공 신경망을 x_train, y_train 데이터에 맞도록 학습한다는 의미를 갖습니다. 즉, x_train, y_train 데이터에 맞도록 인공 신경망을 조물조물, 주물주물 학습한다는 의미입니다. fit 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. epochs는 학습 횟수를 의미하며, 여기서는 5회 학습을 수행하도록 합니다. 일반적으로 학습 횟수에 따라 인공 신경망 근사 함수가 정확해 집니다.

21 : model.evaluate 함수를 호출하여 인공 신경망의 학습 결과를 평가합니다. 여기서는 학습이 끝난 인공 신경망 함수에 x_test 값을 주어 학습 결과를 평가해 봅니다.

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

```
Epoch 1/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.4329 - accuracy: 0.8786
Epoch 2/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.1204 - accuracy: 0.9545
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0763 - accuracy: 0.9779
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0558 - accuracy: 0.9834
Epoch 5/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.0438 - accuracy: 0.9871
313/313 [=====] - 1s 1ms/step - loss: 0.0776 - accuracy: 0.9767
[0.0776418074965477, 0.9767000079154968]
```

❶ 손실 함수에 의해 측정된 오차 값을 나타냅니다. 학습 횟수가 늘어남에 따라 오차 값이 줄

어듭니다.

❷ 학습 진행에 따른 정확도가 표시됩니다. 처음에 87.86%에서 시작해서 마지막엔 98.71%의 정확도로 학습이 끝납니다. 즉, 100개의 손 글씨가 있다면 98.71개를 맞춘다는 의미입니다.

❸ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 학습 데이터의 예측 결과에 비해 시험 데이터의 예측 결과에서는 손실 값이 늘어났고, 정확도가 97.67%로 약간 떨어진 상태입니다.

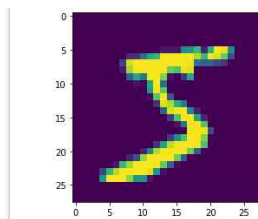
❹ 평가한 결과가 표시됩니다.

연습문제

학습 회수를 다음과 같이 50회로 늘려 정확도를 개선해 봅니다.

```
19 model.fit(x_train, y_train, epochs=50)
```

02 손 글씨 숫자 인식 예제 살펴보기



여기서는 앞에서 전체적으로 실행해 본 예제를 단계별로 살펴봅니다. 즉, 입력 데이터의 모양도 살펴보고 학습을 수행한 후, 학습에 사용하지 않은 손 글씨 숫자를 얼마나 잘 인식하는지 살펴봅니다. 또 잘못 인식한 숫자를 독자 여러분이 직접 화면에 그려보며, 인공 신경망의 인식 성능을 확인해 봅니다.

데이터 모양 살펴보기

여기서는 먼저 MNIST 손 글씨 숫자 데이터의 개수와 형식을 살펴보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

142_1.py

```
1 import tensorflow as tf
2
3 mnist = tf.keras.datasets.mnist
4
5 (x_train, y_train),(x_test, y_test) = mnist.load_data()
```


```
6 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
7       x_train.shape, y_train.shape, x_test.shape, y_test.shape))
```

01 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

03 : mnist 변수를 생성한 후, tf.keras.datasets.mnist 모듈을 가리키게 합니다. mnist 모듈은 손 글씨 숫자 데이터를 가진 모듈입니다. mnist 모듈에는 6만개의 학습용 손 글씨 숫자 데이터와 1만개의 시험용 손 글씨 숫자 데이터가 있습니다.

05 : mnist.load_data 함수를 호출하여 손 글씨 숫자 데이터를 읽어와 x_train, y_train, x_test, y_test 변수가 가리키게 합니다.

06, 07 : print 함수를 호출하여 x_train, y_train, x_test, y_test의 데이터 모양을 출력합니다.

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

```
x_train:(60000, 28, 28) y_train:(60000,) x_test:(10000, 28, 28) y_test:(10000,)
```

x_train, y_train 변수는 각각 6만개의 학습용 손 글씨 숫자 데이터와 숫자 라벨을 가리킵니다. x_test, y_test 변수는 각각 1만개의 시험용 손 글씨 숫자 데이터와 숫자 라벨을 가리킵니다. x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀, 28x28 픽셀로 구성되어 있습니다.

학습 데이터 그림 그려보기

여기서는 학습용 데이터의 그림을 화면에 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py


```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train),(x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07       x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
```

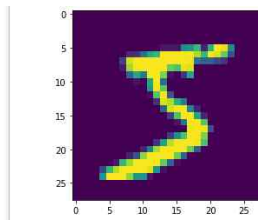

09 : import문을 이용하여 matplotlib.pyplot 모듈을 plt라는 이름으로 불러옵니다. 여기서는 matplotlib.pyplot 모듈을 이용하여 11~13줄에서 그래프를 그립니다.

11 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다.

12 : plt.imshow 함수를 호출하여 x_train[0] 항목의 그림을 내부적으로 그립니다.

13 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

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



x_train[0] 항목의 손 글씨 숫자 그림은 5입니다.

그림 픽셀 값 출력해 보기

여기서는 앞에서 출력한 그림의 픽셀 값을 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train),(x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
```



```

13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(x_train[i], cmap=plt.cm.binary)
26     plt.xlabel(y_train[i])
27 plt.show()

```

20 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다. figsize는 그림의 인치 단위의 크기를 나타냅니다. 여기서는 가로 10인치, 세로 10인치의 그림을 그린다는 의미입니다.

21 : 0에서 24에 대해


22 : plt.subplot 함수를 호출하여 그림 창을 분할하여 하위 그림을 그립니다. 5,5는 각각 행의 개수와 열의 개수를 의미합니다. i+1은 하위 그림의 위치를 나타냅니다.

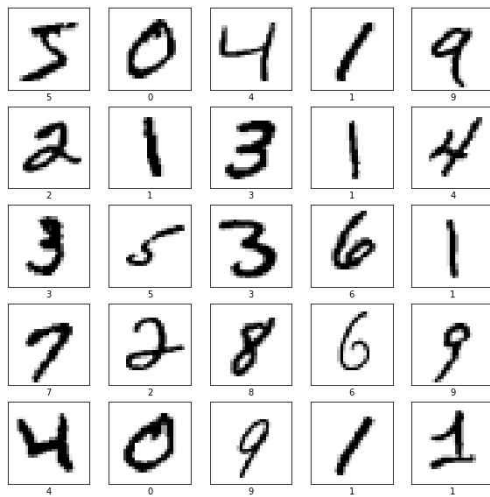
23, 24 : plt.xticks, plt.yticks 함수를 호출하여 x, y 축 눈금을 설정합니다. 여기서는 빈 리스트를 주어 눈금 표시를 하지 않습니다.

25 : plt.imshow 함수를 호출하여 x_train[i] 항목의 그림을 내부적으로 그립니다. cmap는 color map의 약자로 binary는 그림을 이진화해서 표현해 줍니다.

26 : plt.xlabel 함수를 호출하여 x 축에 라벨을 붙여줍니다. 라벨의 값은 y_train[i]입니다.

27 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

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



x_train 변수가 가리키는 손 글씨 숫자 그림 25개를 볼 수 있습니다. x_train 변수는 이런 그림을 6만개를 가리키고 있습니다.

인공 신경망 학습시키기

여기서는 이전 예제에서 살펴본 손 글씨 숫자 데이터를 이용하여 인공 신경망을 학습시켜 봅니다. 인공 신경망은 앞에서 구성했던 신경망을 그대로 사용합니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train), (x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
```

```

18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(x_train[i], cmap=plt.cm.binary)
26     plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)

```

29 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀로 구성된 그림이며, 1픽셀의 크기는 8비트로 0에서 255사이의 숫자를 가집니다. 모든 픽셀의 숫자를 255.0으로 나누어 각 픽셀을 0.0에서 1.0사이의 실수로 바꾸어 인공 신경망에 입력하게 됩니다. 다음은 x_train[0] 그림의 픽셀 값을 출력한 그림입니다.

30 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀, 28x28 픽셀로 구성되어 있습니다. 이 예제에서 소개하는 인공 신경망의 경우 그림 데이터를 입력할 때 28x28 픽셀을 784(=28x28) 픽셀로 일렬로 세워서 입력하게 됩니다. 그래서 33줄에 있는 InputLayer 클래스는 일렬로 세워진 784 픽셀을 입력으로 받도록 구성됩니다.

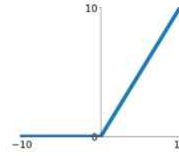


33 : `tf.keras.layers.InputLayer` 함수를 이용하여 내부적으로 keras 라이브러리에서 제공하는 `tensor`를 생성하고, 입력 노드의 개수를 정해줍니다. `tensor`는 3차원 이상의 행렬을 의미하며, 인공 신경망 구성 시 사용하는 자료 형입니다. 여기서는 784개의 입력 노드를 생성합니다.

다.

34 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서는 단위 인공 신경 128개를 생성합니다. `activation`은 활성화 함수를 의미하며 여기서는 'relu' 함수를 사용합니다. 다음은 relu 함수를 나타냅니다.

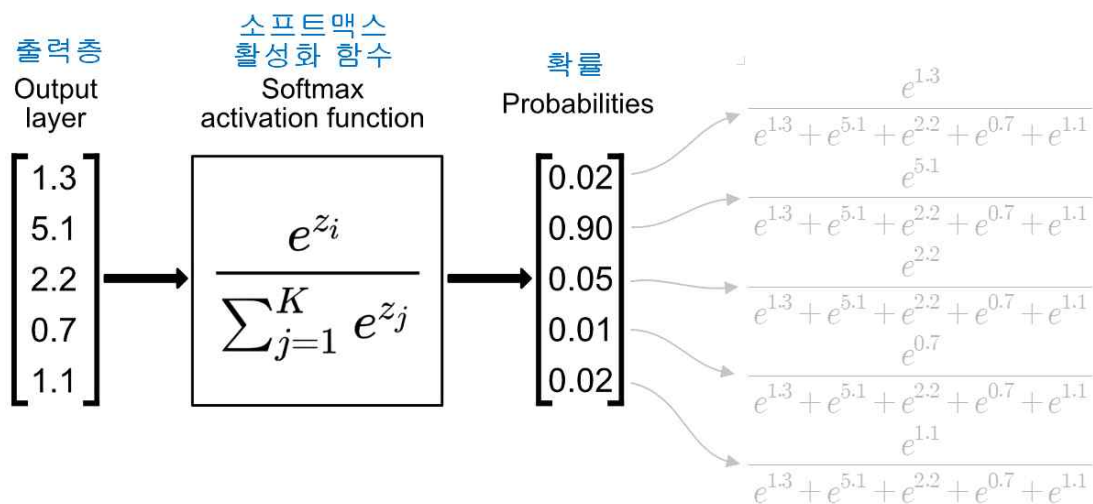
$$\text{ReLU} \\ \max(0, x)$$



활성화 함수와 'relu' 함수에 대해서는 뒤에서 구현해 보면서 자세히 살펴보도록 합니다.

여기서 `Dense`는 내부적으로 $y = \text{activation}(x*w + b)$ 식을 생성하게 됩니다. 이 식에 대해서는 뒤에서 실제로 구현해 보며 그 원리를 살펴보도록 합니다.

35 : `tf.keras.layers.Dense` 클래스를 이용하여 신경망 층을 생성합니다. 여기서는 단위 인공 신경 10개를 생성합니다. `activation`은 활성화 함수를 의미하며 여기서는 'softmax' 함수를 사용합니다. 다음은 softmax 함수를 나타냅니다.




참고로 'softmax' 함수는 출력 층에서만 사용할 수 있는 활성화 함수입니다. 활성화 함수와 'softmax' 함수에 대해서는 뒤에서 구현해 보면서 자세히 살펴보도록 합니다.

38~40 : `model.compile` 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망을 구성할 때에는 적어도 2개의 함수를 정해야 합니다. `loss` 함수와 `optimizer` 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수와 최적화 함수에 대해서는 뒤에서 자세히 살펴보십시오. 손실 함수로는 `sparse_categorical_crossentropy` 함수를 사용하고 최적화 함수는 `adam` 함수를 사용합니다. `sparse_categorical_crossentropy`, `adam` 함수는 뒤에서 살펴보도록 합니다. `fit` 함수 로그에는 기본적으로 손실 값만 표시됩니다. `metrics` 매개 변수는 학습 측정 항목 함수를 전달할 때 사용합니다. 'accuracy'는 학습의 정확도를 출력해 줍니다.

42 : `model.fit` 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. `fit` 함수에는

x_train, y_train 데이터가 입력이 되는데 인공 신경망을 x_train, y_train 데이터에 맞도록 학습한다는 의미를 갖습니다. 즉, x_train, y_train 데이터에 맞도록 인공 신경망을 조물조물, 주물주물 학습한다는 의미입니다. fit 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. epochs는 학습 횟수를 의미하며, 여기서는 5회 학습을 수행하도록 합니다. 일반적으로 학습 횟수에 따라 인공 신경망 근사 함수가 정확해 집니다.

44 : model.evaluate 함수를 호출하여 인공 신경망의 학습 결과를 평가합니다. 여기서는 학습이 끝난 인공 신경망 함수에 x_test 값을 주어 학습 결과를 평가해 봅니다.

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

```
Epoch 1/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.4268 - accuracy: 0.8779
Epoch 2/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.1233 - accuracy: 0.9637
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0780 - accuracy: 0.9762
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0586 - accuracy: 0.9827
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0447 - accuracy: 0.9860
313/313 [=====] - 1s 1ms/step - loss: 0.0705 - accuracy: 0.9776
```

- ❶ 손실 함수에 의해 측정된 오차 값을 나타냅니다. 학습 횟수가 늘어남에 따라 오차 값이 줄어듭니다.
- ❷ 학습 진행에 따른 정확도가 표시됩니다. 처음에 87.79%에서 시작해서 마지막엔 98.60%의 정확도로 학습이 끝납니다. 즉, 100개의 손 글씨가 있다면 98.60개를 맞춘다는 의미입니다.
- ❸ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 손실 값이 늘어났고, 정확도가 97.76%로 약간 떨어진 상태입니다.

학습된 인공 신경망 시험하기

여기서는 학습된 신경망에 시험 데이터를 입력하여 예측해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train), (x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
```




```

12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(x_train[i], cmap=plt.cm.binary)
26     plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)
45
46 p_test = model.predict(x_test)
47 print('p_test[0] :', p_test[0])

```

46 : model.predict 함수를 호출하여 인공 신경망을 시험합니다. 여기서는 학습이 끝난 인공 신경망 함수에 x_test 값을 주어 그 결과를 예측해 봅니다. 예측한 결과 값은 p_test 변수로 받습니다. x_test는 1만개의 손 글씨 숫자를 가리고 있으며, 따라서 1만개에 대한 예측을 수행합니다.

47 : print 함수를 호출하여 x_test[0] 손 글씨 숫자의 예측 값을 출력합니다.

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

```
p_test[0] : [1.07918666e-07 4.64393946e-09 3.81268319e-06 5.01001021e-04
2.92917357e-10 3.51789765e-07 5.53046931e-12 9.99484420e-01
9.99521490e-07 9.21584615e-06]
```

p_test[0]은 x_test[0]이 가리키는 손 글씨 숫자에 대해 0~9 각각의 숫자에 대한 확률값 리스트를 출력합니다. x_test[0]은 실제로 숫자 7을 가리키고 있습니다. 그래서 p_test[0]의 8번째 값의 확률이 가장 높게 나타납니다. 8번째 값은 9.99484420e-01이며 99.9%로 7이라고 예측합니다. p_test[0]의 1번째 값은 숫자 0일 확률을 나타냅니다.

예측 값과 실제 값 출력해 보기

여기서는 예측 값과 실제 값을 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train),(x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
```


```

24 plt.yticks([])
25 plt.imshow(x_train[i], cmap=plt.cm.binary)
26 plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)
45
46 p_test = model.predict(x_test)
47 print('p_test[0] :', p_test[0])
48
49 import numpy as np
50
51 print('p_test[0] :', np.argmax(p_test[0]), 'y_test[0] :', y_test[0])

```

49 : import문을 이용하여 numpy 모듈을 np라는 이름으로 불러옵니다. 여기서는 numpy 모듈의 argmax 함수를 이용하여 51 줄에서 p_test[0] 항목의 가장 큰 값의 항목 번호를 출력합니다.

51 : print 함수를 호출하여 p_test[0] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨 값을 출력합니다.

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

```
p_test[0] : 7 y_test[0] : 7
```

p_test[0] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨 값이 같습니다. x_test[0] 항목의 경우 예측 값과 실제 값이 같아 인공 신경망이 옳게 예측합니다.

시험 데이터 그림 그려보기

여기서는 시험용 데이터의 그림을 화면에 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train), (x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(x_train[i], cmap=plt.cm.binary)
26     plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
```

```

34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)
45
46 p_test = model.predict(x_test)
47 print('p_test[0] :', p_test[0])
48
49 import numpy as np
50
51 print('p_test[0] :', np.argmax(p_test[0]), 'y_test[0] :', y_test[0])
52
53 x_test = x_test.reshape(10000, 28, 28)
54
55 plt.figure()
56 plt.imshow(x_test[0])
57 plt.show()


```

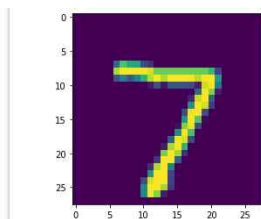
53 : reshape 함수를 호출하여 x_test가 가리키는 그림을 원래 모양으로 돌려놓습니다. 그래야 pyplot 모듈을 이용하여 그림을 화면에 표시할 수 있습니다.

55 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다.

56 : plt.imshow 함수를 호출하여 x_test[0] 항목의 그림을 내부적으로 그립니다.

57 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

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



x_test[0] 항목의 손 글씨 숫자 그림은 7입니다.

시험 데이터 그림 그려보기 2

여기서는 시험 데이터의 그림 25개를 화면에 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train),(x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(x_train[i], cmap=plt.cm.binary)
26     plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
```

```

34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)
45
46 p_test = model.predict(x_test)
47 print('p_test[0] :', p_test[0])
48
49 import numpy as np
50
51 print('p_test[0] :', np.argmax(p_test[0]), 'y_test[0] :', y_test[0])
52
53 x_test = x_test.reshape(10000,28,28)
54
55 plt.figure()
56 plt.imshow(x_test[0])
57 plt.show()
58
59 plt.figure(figsize=(10,10))
60 for i in range(25):
61     plt.subplot(5,5,i+1)
62     plt.xticks([])
63     plt.yticks([])
64     plt.imshow(x_test[i], cmap=plt.cm.binary)
65     plt.xlabel(np.argmax(p_test[i]))
66 plt.show()

```

59 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다. figsize는 그림의 인치 단위의 크기를 나타냅니다. 여기서는 가로 10인치, 세로 10인치의 그림을 그린다는 의미입니다.

60 : 0에서 24에 대해


61 : plt.subplot 함수를 호출하여 그림 창을 분할하여 하위 그림을 그립니다. 5,5는 각각 행의 개수와 열의 개수를 의미합니다. i+1은 하위 그림의 위치를 나타냅니다.

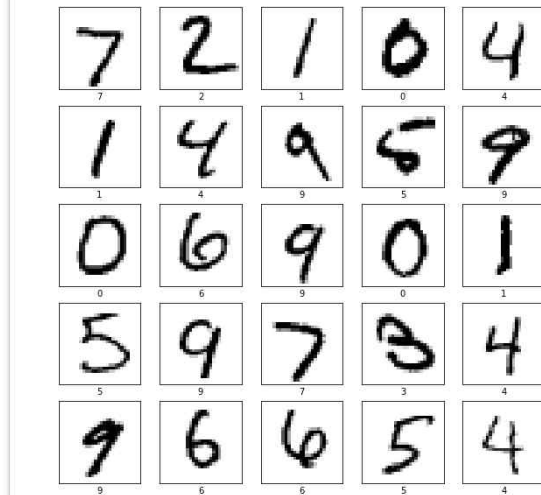
62, 63 : plt.xticks, plt.yticks 함수를 호출하여 x, y 축 눈금을 설정합니다. 여기서는 빈 리스트를 주어 눈금 표시를 하지 않습니다.

64 : plt.imshow 함수를 호출하여 x_test[i] 항목의 그림을 내부적으로 그립니다. cmap은 color map의 약자로 binary는 그림을 이진화해서 표현해 줍니다.

65 : plt.xlabel 함수를 호출하여 x 축에 라벨을 붙여줍니다. 라벨의 값은 y_train[i]입니다.

66 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

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



x_test 변수가 가리키는 손 글씨 숫자 그림 25개를 볼 수 있습니다. x_test 변수는 이런 그림을 1만개를 가리키고 있습니다.

잘못된 예측 살펴보기

여기서는 시험 데이터 중 잘못된 예측이 몇 개나 되는지 또 몇 번째 그림이 잘 못 예측되었는지 살펴보도록 합니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train),(x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
```



```
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(x_train[i], cmap=plt.cm.binary)
26     plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)
45
46 p_test = model.predict(x_test)
47 print('p_test[0] :', p_test[0])
48
49 import numpy as np
50
```

```

51 print('p_test[0] :', np.argmax(p_test[0]), 'y_test[0] :',y_test[0])
52
53 x_test = x_test.reshape(10000,28,28)
54
55 plt.figure()
56 plt.imshow(x_test[0])
57 plt.show()
58
59 plt.figure(figsize=(10,10))
60 for i in range(25):
61     plt.subplot(5,5,i+1)
62     plt.xticks([])
63     plt.yticks([])
64     plt.imshow(x_test[i], cmap=plt.cm.binary)
65     plt.xlabel(np.argmax(p_test[i]))
66 plt.show()
67
68 cnt_wrong = 0
69 p_wrong = []
70 for i in range(10000):
71     if np.argmax(p_test[i]) != y_test[i]:
72         p_wrong.append(i)
73         cnt_wrong +=1
74
75 print('cnt_wrong :', cnt_wrong)
76 print('predicted wrong 10 :', p_wrong[:10])

```

68 : cnt_wrong 변수를 선언한 후, 0으로 초기화합니다. cnt_wrong 변수는 잘못 예측된 그림의 개수를 저장할 변수입니다.

69 : p_wrong 변수를 선언한 후, 빈 리스트로 초기화합니다. p_wrong 변수는 잘못 예측된 그림의 번호를 저장할 변수입니다.

70 : 0부터 10000미만까지


71 : p_test[i] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨 값이 다르면

72 : p_wrong 리스트에 해당 그림 번호를 추가하고

73 : cnt_wrong 값을 하나 증가시킵니다.

75 : print 함수를 호출하여 cnt_wrong 값을 출력합니다.

76 : print 함수를 호출하여 p_wrong에 저장된 값 중, 앞에서 10개까지 출력합니다. p_wrong[:10]은 p_wrong 리스트의 0번 항목부터 시작해서 10번 항목 미만인 9번 항목까지를 의미합니다.

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

```
cnt_wrong : 224
predicted wrong 10 : [115, 247, 320, 321, 340, 381, 445, 582, 610, 619]
```

학습이 끝난 인공 신경망은 시험 데이터에 대해 10000개 중 224개에 대해 잘못된 예측을 하였습니다. 즉, 10000개 중 9776(=10000-224)개는 바르게 예측을 했으며, 나머지 224개에 대해서는 잘못된 예측을 하였습니다. 예측 정확도는 97.76%, 예측 오류 도는 2.24%입니다. 잘못 예측한 데이터 번호 10개에 대해서도 확인해 봅니다. 115번 데이터로 시작해서 619번 데이터까지 10개의 데이터 번호를 출력하고 있습니다.

잘못 예측한 그림 살펴보기

여기서는 시험 데이터 중 잘못 예측한 그림을 출력해봅니다.

1. 다음과 같이 예제를 수정합니다.

142_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (x_train, y_train), (x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
18     print()
19
20 plt.figure(figsize=(10,10))
21 for i in range(25):
22     plt.subplot(5,5,i+1)
23     plt.xticks([])
24     plt.yticks([])
```

```
25 plt.imshow(x_train[i], cmap=plt.cm.binary)
26 plt.xlabel(y_train[i])
27 plt.show()
28
29 x_train, x_test = x_train/255.0, x_test/255.0
30 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
31
32 model = tf.keras.models.Sequential([
33     tf.keras.layers.InputLayer(input_shape=(784,)),
34     tf.keras.layers.Dense(128, activation='relu'),
35     tf.keras.layers.Dense(10, activation='softmax')
36 ])
37
38 model.compile(optimizer='adam',
39               loss='sparse_categorical_crossentropy',
40               metrics=['accuracy'])
41
42 model.fit(x_train, y_train, epochs=5)
43
44 model.evaluate(x_test, y_test)
45
46 p_test = model.predict(x_test)
47 print('p_test[0] :', p_test[0])
48
49 import numpy as np
50
51 print('p_test[0] :', np.argmax(p_test[0]), 'y_test[0] :', y_test[0])
52
53 x_test = x_test.reshape(10000,28,28)
54
55 plt.figure()
56 plt.imshow(x_test[0])
57 plt.show()
58
59 plt.figure(figsize=(10,10))
60 for i in range(25):
61     plt.subplot(5,5,i+1)
62     plt.xticks([])
63     plt.yticks([])
64     plt.imshow(x_test[i], cmap=plt.cm.binary)
```

```

65     plt.xlabel(np.argmax(p_test[i]))
66 plt.show()
67
68 cnt_wrong = 0
69 p_wrong = []
70 for i in range(10000):
71     if np.argmax(p_test[i]) != y_test[i]:
72         p_wrong.append(i)
73         cnt_wrong += 1
74
75 print('cnt_wrong :', cnt_wrong)
76 print('predicted wrong 10 :', p_wrong[:10])
77
78 plt.figure(figsize=(10,10))
79 for i in range(25):
80     plt.subplot(5,5,i+1)
81     plt.xticks([])
82     plt.yticks([])
83     plt.imshow(x_test[p_wrong[i]], cmap=plt.cm.binary)
84     plt.xlabel("%s : p%s y%s"%(
85         p_wrong[i], np.argmax(p_test[p_wrong[i]]), y_test[p_wrong[i]]))
86 plt.show()

```

78 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다. figsize는 그림의 인치 단위의 크기를 나타냅니다. 여기서는 가로 10인치, 세로 10인치의 그림을 그린다는 의미입니다.

79 : 0에서 24에 대해


80 : plt.subplot 함수를 호출하여 그림 창을 분할하여 하위 그림을 그립니다. 5,5는 각각 행의 개수와 열의 개수를 의미합니다. i+1은 하위 그림의 위치를 나타냅니다.

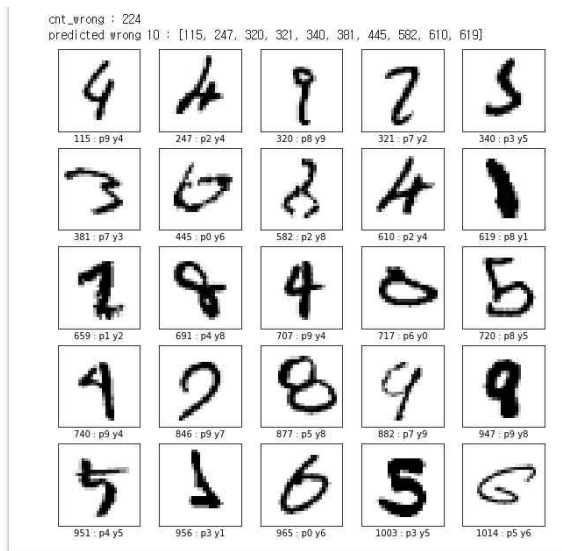
81, 82 : plt.xticks, plt.yticks 함수를 호출하여 x, y 축 눈금을 설정합니다. 여기서는 빈 리스트를 주어 눈금 표시를 하지 않습니다.

83 : plt.imshow 함수를 호출하여 x_test[p_wrong[i]] 항목의 그림을 내부적으로 그립니다. cmap는 color map의 약자로 binary는 그림을 이진화해서 표현해 줍니다.

84, 85 : plt.xlabel 함수를 호출하여 x 축에 라벨을 붙여줍니다. 라벨의 값은 잘못 예측한 그림 번호, 인공 신경망이 예측한 숫자 값, 라벨에 표시된 숫자 값으로 구성됩니다.

86 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

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



필자의 경우 첫 번째 잘못 예측한 숫자는 115번째의 숫자이며, 인공 신경망은 9로 예측하였으며, 실제 값은 4입니다. 언뜻 보면 사람이 보기에 혼동될 수 있는 형태의 숫자들입니다.

연습문제

학습 회수를 다음과 같이 50회로 늘려 정확도를 개선해 봅니다.

```
42 model.fit(x_train, y_train, epochs=50)
```

이상에서 손 글씨 숫자를 이용하여 인공 신경망을 학습시키고, 학습시킨 결과를 예측하는 과정을 자세히 살펴보았습니다. 일반적으로 인공 신경망을 이용할 때, 달라지는 부분은 데이터의 종류와 인공 신경망의 구성 형태입니다. 나머지 부분은 프레임워크처럼 비슷한 형태로 작성될 가능성이 높습니다.

03 패션 MNIST 데이터 셋 인식시켜보기

여기서는 이전에 작성했던 예제를 부분적으로 수정해 가며 또 다른 데이터 셋인 패션 MNIST 데이터 셋을 사용해서 인공 신경망을 학습시켜봅니다. 패션 MNIST 데이터 셋은 10개의 범주(category)와 70,000개의 흑백 이미지로 구성됩니다. 패션 MNIST 데이터 셋의 그림은 손 글씨 데이터 셋과 마찬가지로 28x28 픽셀의 해상도를 가지며 다음처럼 신발, 옷, 가방 등의 품목을 나타냅니다. 패션 MNIST 데이터 셋은 손 글씨 MNIST보다 좀 더 복잡한 형태의 이미지를 제공하기 위해 만들어졌습니다.



1. 다음과 같이 예제를 수정합니다.

143_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.fashion_mnist
04
05 (x_train, y_train), (x_test, y_test) = mnist.load_data()
06 print("x_train:%s y_train:%s x_test:%s y_test:%s"%(
07     x_train.shape, y_train.shape, x_test.shape, y_test.shape))
08
09 import matplotlib.pyplot as plt
10
11 plt.figure()
12 plt.imshow(x_train[0])
13 plt.show()
14
15 for y in range(28):
16     for x in range(28):
17         print("%4s"%x_train[0][y][x],end='')
```

```
18     print()
19
20 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
21               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
22
23 plt.figure(figsize=(10,10))
24 for i in range(25):
25     plt.subplot(5,5,i+1)
26     plt.xticks([])
27     plt.yticks([])
28     plt.imshow(x_train[i], cmap=plt.cm.binary)
29     plt.xlabel(class_names[y_train[i]])
30 plt.show()
31
32 x_train, x_test = x_train/255.0, x_test/255.0
33 x_train, x_test = x_train.reshape(60000,784), x_test.reshape(10000,784)
34
35 model = tf.keras.models.Sequential([
36     tf.keras.layers.InputLayer(input_shape=(784,)),
37     tf.keras.layers.Dense(128, activation='relu'),
38     tf.keras.layers.Dense(10, activation='softmax')
39 ])
40
41 model.compile(optimizer='adam',
42               loss='sparse_categorical_crossentropy',
43               metrics=['accuracy'])
44
45 model.fit(x_train, y_train, epochs=5)
46
47 model.evaluate(x_test, y_test)
48
49 p_test = model.predict(x_test)
50 print('p_test[0] :', p_test[0])
51
52 import numpy as np
53
54 print('p_test[0] :', np.argmax(p_test[0]), class_names[np.argmax(p_test[0])],
55       'y_test[0] :', y_test[0], class_names[y_test[0]])
56
57 x_test = x_test.reshape(10000,28,28)
```



```

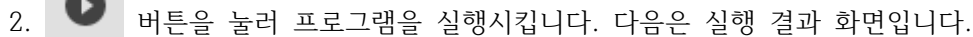
58
59 plt.figure()
60 plt.imshow(x_test[0])
61 plt.show()
62
63 plt.figure(figsize=(10,10))
64 for i in range(25):
65     plt.subplot(5,5,i+1)
66     plt.xticks([])
67     plt.yticks([])
68     plt.imshow(x_test[i], cmap=plt.cm.binary)
69     plt.xlabel(class_names[np.argmax(p_test[i])])
70 plt.show()
71
72 cnt_wrong = 0
73 p_wrong = []
74 for i in range(10000):
75     if np.argmax(p_test[i]) != y_test[i]:
76         p_wrong.append(i)
77         cnt_wrong += 1
78
79 print('cnt_wrong :', cnt_wrong)
80 print('predicted wrong 10 :', p_wrong[:10])
81
82 plt.figure(figsize=(10,10))
83 for i in range(25):
84     plt.subplot(5,5,i+1)
85     plt.xticks([])
86     plt.yticks([])
87     plt.imshow(x_test[p_wrong[i]], cmap=plt.cm.binary)
88     plt.xlabel("%s : p%s y%s"%(
89         p_wrong[i], class_names[np.argmax(p_test[p_wrong[i]]),
90         class_names[y_test[p_wrong[i]])])
91 plt.show()

```

03 : mnist 변수를 생성한 후, `tf.keras.datasets.fashion_mnist` 모듈을 가리키게 합니다. `fashion_mnist` 모듈은 신발, 옷, 가방 등의 데이터를 가진 모듈입니다. `fashion_mnist` 모듈에는 6만개의 학습용 데이터와 1만개의 시험용 데이터가 있습니다.

20, 21 : 패션 MNIST 데이터 셋을 이루는 품목의 종류는 손 글씨 MNIST 데이터 셋과 마찬가지로 10가지로 구성되며, 품목의 라벨은 숫자 0~9로 구성됩니다. 여기서는 품목의 해당 라벨 값에 품목의 이름을 대응시킵니다. `class_names` 변수를 선언한 후, 품목의 이름 10가지로 초기화합니다. 예를 들어, 품목의 라벨 값 0은 'T-shirt/top'을 의미하며, 9는 'Ankle boot'

90 : y_test[p_wrong[i]]을 class_names[y_test[p_wrong[i]]]으로 변경해 줍니다.



각 픽셀의 값이 0~255 사이의 값에서 출력되는 것을 확인합니다.



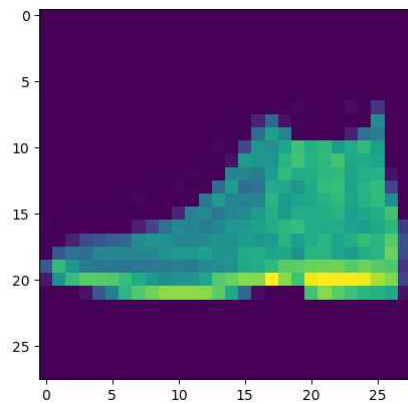
x_train 변수가 가리키는 패션 MNIST 그림 25개를 볼 수 있습니다. x_train 변수는 이런 그림을 6만개를 가리키고 있습니다.

```
Epoch 1/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.5019 - accuracy: 0.8240
Epoch 2/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.3741 - accuracy: 0.8645
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.3359 - accuracy: 0.8782
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.3124 - accuracy: 0.8847
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.2963 - accuracy: 0.8913
313/313 [=====] - 1s 1ms/step - loss: 0.3546 - accuracy: 0.8687
```

- ❶ 손실 함수에 의해 측정된 오차 값을 나타냅니다. 학습 횟수가 늘어남에 따라 오차 값이 줄어듭니다.
- ❷ 학습 진행에 따른 정확도가 표시됩니다. 처음에 82.40%에서 시작해서 마지막엔 89.13%의 정확도로 학습이 끝납니다. 즉, 100개의 패션 MNIST 그림이 있다면 89.13개를 맞춘다는 의미입니다.
- ❸ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 손실 값이 늘어났고, 정확도가 86.87%로 약간 떨어진 상태입니다.

```
p_test[0] : [2.0026380e-05 1.8734058e-06 8.2557347e-07 6.7981420e-09 1.1301979e-06
5.2375207e-03 4.0497648e-06 1.2388683e-02 1.8332830e-05 9.8232746e-01]
p_test[0] : 9 Ankle boot y_test[0] : 9 Ankle boot
```

p_test[0]은 x_test[0]이 가리키는 패션 MNIST 그림에 대해 0~9 각각의 숫자에 대한 확률값 리스트를 출력합니다. x_test[0]은 실제로 숫자 9를 가리키고 있습니다. 그래서 p_test[0]의 9번째 값의 확률이 가장 높게 나타납니다. 9번째 값은 9.8232746e-01이며 98.2%로 9라고 예측합니다. p_test[0]의 1번째 값은 숫자 0일 확률을 나타냅니다. p_test[0] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨 값이 같습니다. x_test[0] 항목의 경우 예측 값과 실제 값이 같아 인공 신경망이 옳게 예측합니다.



`x_test[0]` 항목의 그림은 Ankle boot입니다.



`x_test` 변수가 가리키는 패션 MNIST 그림 25개를 볼 수 있습니다. `x_test` 변수는 이런 그림을 1만개를 가리키고 있습니다.

```
cnt_wrong : 1313
```

```
predicted wrong 10 : [12, 17, 23, 25, 29, 40, 42, 48, 49, 50]
```



학습이 끝난 인공 신경망은 시험 데이터에 대해 10000개 중 1313개에 대해 잘못된 예측을 하였습니다. 즉, 10000개 중 8687(=10000-1313)개는 바르게 예측을 했으며, 나머지 1313개에 대해서는 잘못된 예측을 하였습니다. 예측 정확도는 86.87%, 예측 오류 도는 13.13%입니다. 잘못 예측한 데이터 번호 10개에 대해서도 확인해 봅니다. 12번 데이터로 시작해서 50번 데이터까지 10개의 데이터 번호를 출력하고 있습니다. 첫 번째 잘못 예측한 패션 MNIST 그림은 12번째 그림이며, 인공 신경망은 Sandal로 예측하였으며, 실제 값은 Sneaker입니다. 두 번째 잘못 예측한 패션 MNIST 그림은 17번째의 그림이며, 인공 신경망은 Pullover로 예측하였으며, 실제 값은 Coat입니다. 언뜻 보면 사람이 보기에 혼동될 수 있는 형태의 품목들입니다.

연습문제

학습 회수를 다음과 같이 50회로 늘려 정확도를 개선해 봅니다.

```
45 model.fit(x_train, y_train, epochs=50)
```

이상에서 이전 예제를 수정한 후, 패션 MNIST 그림을 이용하여 인공 신경망을 학습시키고, 학습시킨 결과를 예측하는 과정을 살펴보았습니다. 패션 MNIST 그림의 경우 숫자 MNIST 그림보다 인식률이 낮습니다. 패션 MNIST 그림에 대한 낮은 인식률은 인공 신경망의 구성을 바꾸어서 높일 수 있습니다.