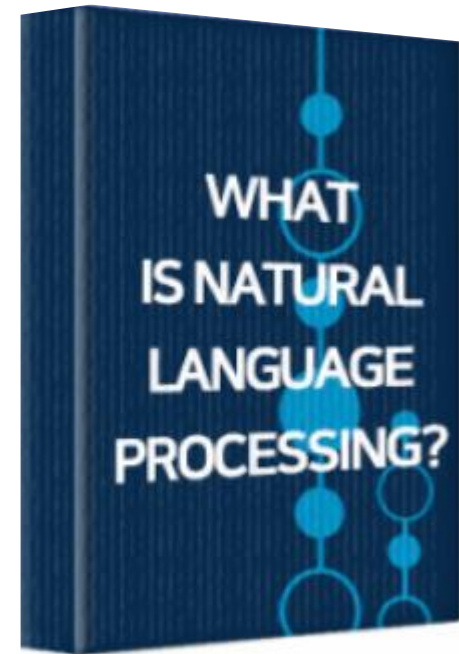


Tensorflow를 활용한 딥러닝 자연어 처리 입문. 2강

앞으로 배우게 될 내용

- Text preprocessing for NLP & Language Model
- **Basic Tensorflow & Vectorization**
- Word Embedding (Word2Vec, FastText, GloVe)
- Text Classification (using RNN & CNN)
- Chatbot with Deep Learning
- Sequence to Sequence
- Attention Mechanism
- Transformer & BERT



참고 자료 : <https://wikidocs.net/book/2155>

자연어 처리를 배우는 순서



자연어 처리를 배우는 순서



오늘 내용 영상을 통해 복습하기

- 오늘 배울 내용을 복습하시고 싶으신 분들은 아래의 링크를 참고.

- 1) <https://www.boostcourse.org/ai212/lecture/41159>

- 2) <https://www.boostcourse.org/ai212/lecture/41844>

Text preprocessing for NLP

Text preprocessing for NLP

지난 시간 내용 복습!

Text Preprocessing (복습)

기계에게는 단어와 문장의 경계를 알려주어야 한다.
이를 위해서 특정 단위로 토큰화 또는 토큰나이징을 해준다.

['His barber kept his word. But keeping
such a huge secret to himself was
driving him crazy.']



Text Preprocessing (복습)

기계에게는 단어와 문장의 경계를 알려주어야 한다.
이를 위해서 특정 단위로 토큰화 또는 토큰나이징을 해준다.

Tokenization

['His barber kept his word. But keeping
such a huge secret to himself was
driving him crazy.']




['His', 'barber', 'kept', 'his', 'word', '.', 'But',
'keeping', 'such', 'a', 'huge', 'secret', 'to',
'himself', 'was', 'driving', 'him', 'crazy', '.']

Text Preprocessing (복습)

기계를 위해 텍스트를 숫자로 수치화해주어야 합니다.
이를 위해 단어들의 집합인 Vocabulary를 생성해준다.

['His', 'barber', 'kept', 'his', 'word', '.', 'But',
'keeping', 'such', 'a', 'huge', 'secret', 'to',
'himself', 'was', 'driving', 'him', 'crazy', '.,']




Vocabulary가 무엇인지 혹시 기억나시나요?

Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But',
'keeping', 'such', 'a', 'huge', 'secret', 'to',
'himself', 'was', 'driving', 'him', 'crazy', '.']



Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary

Vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But',
'keeping', 'such', 'a', 'huge', 'secret', 'to',
'himself', 'was', 'driving', 'him', 'crazy', '.']



his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy

Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary

Vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But',
'keeping', 'such', 'a', 'huge', 'secret', 'to',
'himself', 'was', 'driving', 'him', 'crazy', '.']

his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy

단어 집합을 생성한 후에 할 일은?

Text Preprocessing (복습)

기계가 알고있는 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.
단어 집합이란 훈련 데이터에 있는 단어들의 중복을 제거한 집합을 의미한다.

Build vocabulary

Vocabulary

['His', 'barber', 'kept', 'his', 'word', '.', 'But',
'keeping', 'such', 'a', 'huge', 'secret', 'to',
'himself', 'was', 'driving', 'him', 'crazy', '.']

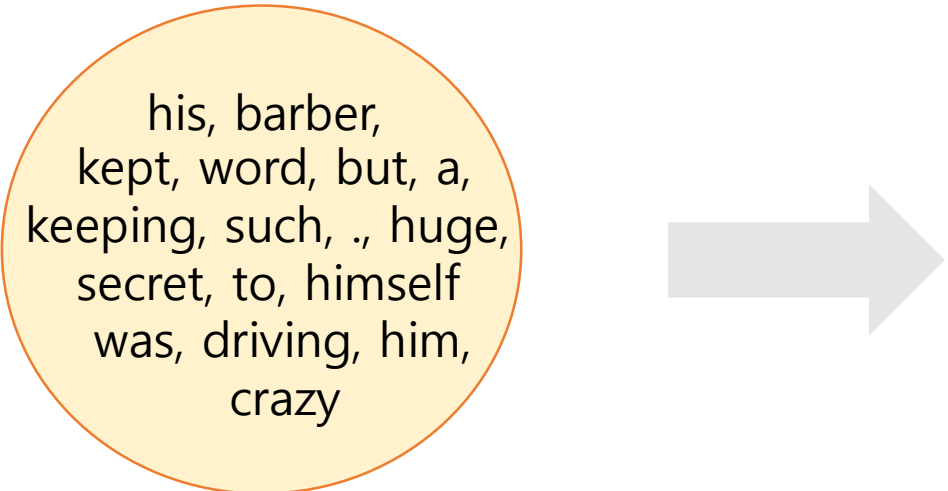
his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy

단어 집합을 생성한 후에 할 일은?
단어에 고유한 정수를 부여!

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.
이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary



his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

각 단어에 정수가 부여됩니다.
단어 집합을 기반으로 하므로
중복은 허용되지 않아요!

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

현재 단어 집합의 크기는?

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Vocabulary

his, barber,
kept, word, but, a,
keeping, such, ., huge,
secret, to, himself
was, driving, him,
crazy



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

현재 단어 집합의 크기는? 17!


Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

새로운 문장이 입력!

['his', 'barber', 'kept', 'a', 'secret', '.']



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

새로운 문장이 입력!

['his', 'barber', 'kept', 'a', 'secret', '.']



{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

새로운 문장이 입력!

['his', 'barber', 'kept', 'a', 'secret', '.']

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17}

각 단어를 고유한 정수로!

['1', '2', '3', '6', '11', '9']

Text Preprocessing (복습)


단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.


Integer Encoding

여러 문장에 대해서는?

[['his', 'barber', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17 }



각 단어를 고유한 정수로!

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

여러 문장에 대해서는?

[['his', 'barber', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17 }

각 단어를 고유한 정수로!


[['1', '2', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해줄기도 한다.

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17 }


Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해줄기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17 }


Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해줄기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17 }

앞으로 모르는 단어가 오면 특별한 토큰 'UNK'로 맵핑하도록 약속!


Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해줄기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

앞으로 모르는 단어가 오면 특별한 토큰 'UNK'로 맵핑하도록 약속!


Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해줄기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

현재 단어 집합의 크기는?


Text Preprocessing (복습)

단어 집합에 없는 단어로 인해 생기는 문제를 OOV 문제라고 한다.
이렇게 생긴 단어들을 일괄적으로 하나의 토큰으로 맵핑해줄기도 한다.

Out-Of-Vocabulary Problem

모르는 단어가 섞여있으면?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

현재 단어 집합의 크기는? 18!

Text Preprocessing (복습)


단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.


Integer Encoding

여러 문장에 대해서는?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }



각 단어를 고유한 정수로!

Text Preprocessing (복습)

단어 집합에 있는 각 단어에는 고유한 정수가 부여된다.

이는 앞으로 입력된 모든 텍스트를 정수 시퀀스로 변환하기 위함이다.

Integer Encoding

여러 문장에 대해서는?

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

각 단어를 고유한 정수로!

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18 }

앞으로 문장의 길이를 동일하게 해주기 위해서는
특별한 토큰 'PAD'를 사용하도록 약속!

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0 }

현재 단어 집합의 크기는?

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]



{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0 }

현재 단어 집합의 크기는? 19!

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0 }

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Text Preprocessing (복습)

여러 문장을 병렬적으로 처리하고 싶은 경우, 이를 하나의 행렬로 인식시켜줄 필요가 있다.
이때, 서로 다른 문장의 길이를 패딩을 통해 동일하게 만들어줄 수 있다.

Padding

문장마다 길이는 다를 수 있다.

[['his', 'teacher', 'kept', 'a', 'secret', '.'],
['a', 'barber', 'was', 'driving', '.']]

{ 'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0 }

패딩 결과

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15', '0', '0']]

[['1', '18', '3', '6', '11', '9'],
['6', '2', '14', '15']]

Vectorization

Vectorization

1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법 : 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

2. 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)

- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

3. 문맥을 고려한 벡터 표현 방법 : ELMo, BERT (2018 - present)

- Pretrained Language Model의 시대.

Vectorization

1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법 : 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF

2. 벡터화에 신경망을 사용하는 경우

- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

3. 문맥을 고려한 벡터 표현 방법 : ELMo, BERT

Vectorization : One-Hot Encoding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩!

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

Vectorization : One-Hot Encoding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩!

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

Vectorization : One-Hot Encoding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩!

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

벡터의 차원이 단어 집합의 크기라는 특징이 있다.

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

Vectorization : One-Hot Encoding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

원-핫 인코딩!

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

단어 벡터 간 유의미한 유사도를 구할 수 없다는 한계가 존재한다.

Vectorization : Word Embedding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

워드 임베딩!

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

Vectorization : Word Embedding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

워드 임베딩!

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

벡터의 차원이 단어 집합의 크기가 아니다.
0과 1의 조합이 아닌 각 원소는 실수값을 가진다.

Vectorization : Word Embedding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

워드 임베딩!

1.2	0.8	0.1	0.2	0.1	0.5	0.1
0.7	0.2	0.5	2.0	0.7	0.11	0.38
5.8	-0.5	3.7	0.11	-1.5	0.8	0.7
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'himself' : 13, 'was' : 14,
'driving' : 15, 'him' : 16, 'crazy' : 17
'unk' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

단어 벡터 간 유의미한 유사도를 구할 수 있다!
'강아지'는 '냉장고'보다 '고양이'와 유사하다.

Vectorization : Word Embedding

- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.

입력된 문장

['his', 'teacher', 'kept', 'a', 'secret', '.']

워드 임베딩!

1.2	0.8	0.1	0.2	0.1	0.5	0.4
0.7	0.2	0.5	2.0	0.1	0.3	0.6
5.8	-0.5	3.7	0.11	-1.1	0.2	-0.3
0.2	0.7	1.2	8.1	0.5	0.1	0.2
1.7	2.1	1.1	0.1	7.8	-0.1	0.8

{'his' : 1, 'barber' : 2, 'kept' : 3,
'word' : 4, 'but' : 5, 'a' : 6, 'keeping' : 7,
'such' : 8, '.' : 9, 'huge' : 10, 'secret' : 11,
'to' : 12, 'myself' : 13, 'was' : 14,
'him' : 16, 'crazy' : 17
'' : 18, 'pad' : 0}

['1', '18', '3', '6', '11', '9']

단어 벡터 간 유의미한 유사도를 구할 수 있다!
'강아지'는 '냉장고'보다 '고양이'와 유사하다.

이건 다음주에 자세히 봅시다!

Vectorization

1. 벡터화에 신경망을 사용하지 않을 경우

- 단어에 대한 벡터 표현 방법 : 원-핫 인코딩
- 문서에 대한 벡터 표현 방법 : **Document Term Matrix, TF-IDF**

2. 벡터화에 신경망을 사용하는 경우

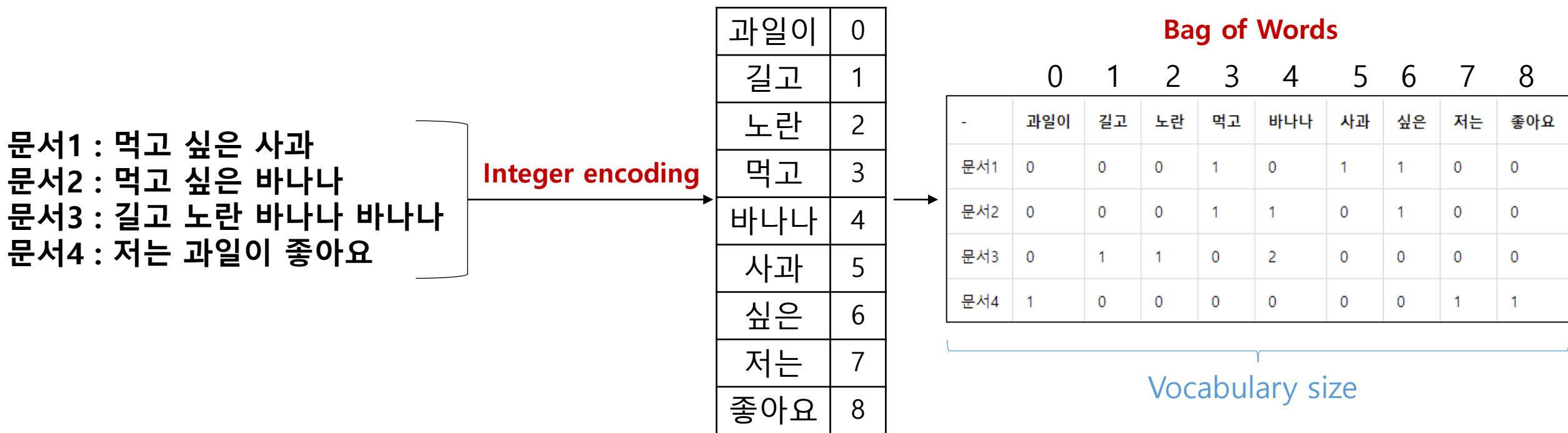
- 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
- 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec

3. 문맥을 고려한 벡터 표현 방법 : ELMo, BERT

Vectorization : Document Term Matrix, DTM

DTM은 마찬가지로 벡터가 단어 집합의 크기를 가지며, 대부분의 값이 0을 가진다.

각 단어는 고유한 정수 인덱스를 가지며, 해당 단어가 등장 횟수를 해당 인덱스의 값으로 가진다.



Vectorization : Document Term Matrix, DTM

DTM은 마찬가지로 벡터가 단어 집합의 크기를 가지며, 대부분의 값이 0을 가진다.

각 단어는 고유한 정수 인덱스를 가지며, 해당 단어가 등장 횟수를 해당 인덱스의 값으로 가진다.

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나
문서4 : 저는 과일이

Bag of Words라는 것이 어떤 의미일까?

과일이	0
길고	1
노란	2
바나나	4
사과	5
싶은	6
저는	7
좋아요	8

	0	1	2	3	4	5	6	7	8
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

Vocabulary size

Bag of Words

Bag of Words를 직역하면 단어들의 가방을 의미한다.
가방에 문장의 단어들을 넣고 흔든다면, 단어의 순서는 무의미해진다.

정리 : 단어의 순서는 무시하고, 오직 단어의 빈도수에만 집중하는 방법

The Bag of Words Representation

I love this movie! It's sweet,
but with satirical humor. The
dialogue is great and the
adventure scenes are fun...
It manages to be whimsical
and romantic while laughing
at the conventions of the
fairy tale genre. I would
recommend it to just about
anyone. I've seen it several
times, and I'm always happy
to see it again whenever I
have a friend who hasn't
seen it yet!

15



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

TF-IDF(Term Frequency-Inverse Document Frequency)

DTM에서 추가적으로 중요한 단어에 가중치를 주는 방식
TF-IDF 기준으로 중요한 단어는 값이 Up.
TF-IDF 기준으로 중요하지 않은 값이 Down.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1



TF-IDF

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'.
- TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- 물론, 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'.
- TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- 물론, 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

이를 통해 텍스트 분류를 수행할 예정!

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 직역하면 '단어 빈도-역 문서 빈도'.
- TF-IDF는 TF와 IDF라는 두 값을 곱한 결과이다.
- 문서의 유사도, 검색 시스템에서 검색 결과의 순위 등을 구하는 일에 쓰인다.
- 물론, 벡터이므로 인공 신경망의 입력으로도 사용할 수 있다.

이를 통해 텍스트 분류를 수행할 예정!

TF-IDF를 처음 접해보는 분들이라면
해보시기를 권장드리는 실습 : <https://wikidocs.net/24603>

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- TF-IDF를 계산해보자.

훈련 데이터

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf (단어 빈도)와 idf (역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d 에서의 특정 단어 t 의 등장 횟수.
- $df(t)$: 특정 단어 t 가 등장한 문서의 수.

**$df(t)$ 로부터 $idf(t)$ 가
무슨 의미인지 유추해볼까요?**

훈련 데이터

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

훈련 데이터

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

각 문서의 각 단어에 대해서 TF를
구하려면 어떻게 해야할까요?

훈련 데이터

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

각 문서의 각 단어에 대해서 TF를
구하려면 어떻게 해야할까요?

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

현재 바나나의 df의 값은?

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

현재 바나나의 df의 값은? 2.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

현재 바나나의 df의 값은? 2.
문서2, 문서 3에서 두 번 등장.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

만약, 문서 2에 바나나가 100번 등장했다고 가정해봅시다. 그렇다면, 바나나의 df의 값은?

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

만약, 문서 2에 바나나가 100번 등장했다고 가정해봅시다. 그렇다면, 바나나의 df의 값은? 2.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

만약, 문서 2에 바나나가 100번 등장했다고 가정해봅시다. 그렇다면, 바나나의 df의 값은? 2.
문서2, 문서 3에서 두 번 등장.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

그렇다면 바나나의 idf는 몇일까?
df에 반비례 하는 수?
df의 역수이니까 $\frac{1}{2}$?

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

그렇다면 바나나의 idf는 몇일까?
df에 반비례 하는 수?
df의 역수이니까 $\frac{1}{2}$?

No!

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 tf(단어 빈도)와 idf(역 문서 빈도)라는 두 값을 곱한 결과이다.
- $tf(d,t)$: 특정 문서 d에서의 특정 단어 t의 등장 횟수.
- $df(t)$: 특정 단어 t가 등장한 문서의 수.
- $idf(d, t)$: $df(t)$ 에 반비례하는 수.

$$idf(d, t) = \log\left(\frac{n}{1 + df(t)}\right)$$

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

df에 반비례하도록 설계하고, log를 씌운다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- $idf(d, t)$ 에는 왜 \log 를 씌울까?
- \log 의 밑은 10을 사용한다고 가정하고, 단어의 df 에 따른 idf 값의 변화를 보자.

$$idf(d, t) = \log(n/df(t))$$

$$n = 1,000,000$$

단어 t	$df(t)$	$idf(d, t)$
word1	1	6
word2	100	4
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

로그 사용

$$idf(d, t) = n/df(t)$$

$$n = 1,000,000$$

단어 t	$df(t)$	$idf(d, t)$
word1	1	1,000,000
word2	100	10,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1

로그 미사용

TF-IDF(Term Frequency-Inverse Document Frequency)

- $idf(d, t)$ 에는 왜 \log 를 씹을까?
- \log 의 밑은 10을 사용한다고 가정하고, 단어의 df 에 따른 idf 값의 변화를 보자.

$$idf(d, t) = \log(n/df(t))$$

$$n = 1,000,000$$

단어 t	$df(t)$	$idf(d, t)$
word1	1	6
word2	100	3
word3	1,000	3
word4	10,000	2
word5	100,000	1
word6	1,000,000	0

로그 사용

$$idf(d, t) = n/df(t)$$

$$n = 1,000,000$$

단어 t	$df(t)$	$idf(d, t)$
word1	1	1,000,000
word3	1,000	1,000
word4	10,000	100
word5	100,000	10
word6	1,000,000	1

로그 미사용

로그를 사용하지 않으면 idf 의 값은 기하급수적으로 커질 수 있다.

IDF에 로그를 씌우는 이유

- TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.

IDF에 로그를 씌우는 이유

- TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.
- 불용어 등과 같이 자주 쓰이는 단어들은 비교적 자주 쓰이지 않는 단어들보다 최소 수십 배 자주 등장한다.
- 비교적 자주 쓰이지 않는 단어들조차 희귀 단어들과 비교하면 또 최소 수백 배는 더 자주 등장하는 편이다.
- log를 씌워주지 않으면, **희귀 단어들에 엄청난 가중치가 부여될 수 있다.**
로그를 씌우면 이런 격차를 줄이는 효과가 있다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- $tf(d,t)$: 특정 문서 d 에서의
특정 단어 t 의 등장 횟수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

TF-IDF(Term Frequency-Inverse Document Frequency)

- $tf(d,t)$: 특정 문서 d 에서의
특정 단어 t 의 등장 횟수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

$$idf(d,t) = \log\left(\frac{n}{1 + df(t)}\right)$$

단어	IDF(역 문서 빈도)
과일이	$\ln(4/(1+1)) = 0.693147$
길고	$\ln(4/(1+1)) = 0.693147$
노란	$\ln(4/(1+1)) = 0.693147$
먹고	$\ln(4/(2+1)) = 0.287682$
바나나	$\ln(4/(2+1)) = 0.287682$
사과	$\ln(4/(1+1)) = 0.693147$
싫은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

TF-IDF(Term Frequency-Inverse Document Frequency)

- $tf(d,t)$: 특정 문서 d 에서의
특정 단어 t 의 등장 횟수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

IDF 값을 보면 2회 등장한 단어들이 값이 더 낮다.

$$idf(d,t) = \log\left(\frac{n}{1 + df(t)}\right)$$

단어	IDF(역 문서 빈도)
과일이	$\ln(4/(1+1)) = 0.693147$
길고	$\ln(4/(1+1)) = 0.693147$
노란	$\ln(4/(1+1)) = 0.693147$
먹고	$\ln(4/(2+1)) = 0.287682$
바나나	$\ln(4/(2+1)) = 0.287682$
사과	$\ln(4/(1+1)) = 0.693147$
싫은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

TF-IDF(Term Frequency-Inverse Document Frequency)

- $tf(d,t)$: 특정 문서 d 에서의
특정 단어 t 의 등장 횟수.

DTM

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1



$$idf(d,t) = \log\left(\frac{n}{1 + df(t)}\right)$$

단어	IDF(역 문서 빈도)
과일이	$\ln(4/(1+1)) = 0.693147$
길고	$\ln(4/(1+1)) = 0.693147$
노란	$\ln(4/(1+1)) = 0.693147$
먹고	$\ln(4/(2+1)) = 0.287682$
바나나	$\ln(4/(2+1)) = 0.287682$
사과	$\ln(4/(1+1)) = 0.693147$
싫은	$\ln(4/(2+1)) = 0.287682$
저는	$\ln(4/(1+1)) = 0.693147$
좋아요	$\ln(4/(1+1)) = 0.693147$

IDF 값을 보면 2회 등장한 단어들이 값이 더 낮다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- $tf(d,t)$: 특정 문서 d 에서의
특정 단어 t 의 등장 횟수.

$$idf(d,t) = \log\left(\frac{n}{1 + df(t)}\right)$$

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

같은 단어라도 TF-IDF값은 다르다.
이는 해당 문서의 TF값에 영향을 받기 때문이다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- $tf(d,t)$: 특정 문서 d 에서의
특정 단어 t 의 등장 횟수.

$$idf(d,t) = \log\left(\frac{n}{1 + df(t)}\right)$$

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

TF-IDF는 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며,
특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단한다.

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 여전히 현업에서도 굉장히 많이 쓰이는 벡터화 방법입니다.
- 문서를 벡터화한다면 각 문서 간의 유사도를 구할 수 있습니다.
- 그리고 문서 간 유사도를 구할 수 있다면 이런 것이 가능합니다.

- 1) 문서 클러스터링
- 2) 유사한 문서 찾기
- 3) 문서 분류 문제

TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF는 여전히 현업에서도 굉장히 많이 쓰이는 벡터화 방법입니다.
- 문서를 벡터화한다면 각 문서 간의 유사도를 구할 수 있습니다.
- 그리고 문서 간 유사도를 구할 수 있다면 이런 것이 가능합니다.

- 1) 문서 클러스터링
- 2) 유사한 문서 찾기
- 3) 문서 분류 문제

인공 신경망으로 단어 임베딩과 유사하게 문서 임베딩 벡터를 얻는 방법도 존재.

Ex) Doc2Vec, Sent2Vec, Universal Sentence Encoder, ELMo, BERT

Summary

- 단어의 표현 방법 : 원-핫 인코딩 Vs. 워드 임베딩
- 문서의 표현 방법 : Document Term Matrix
- 문서의 표현 방법에 가중치를 넣는 방법 : TF-IDF
- 문서가 있을 때, 이를 DTM으로 표현한다면 문서 하나가 벡터가 됩니다.
- 문서가 있을 때, 문서 내의 모든 단어를 워드 임베딩 또는 원-핫 인코딩으로 표현한다면 단어 하나는 벡터가 되고, 문서 하나는 단어의 개수는 행의 크기, 열은 단어 벡터의 차원인 행렬이 됩니다. 즉, 문서는 행렬이 됩니다.
- 문서의 표현 방법을 신경망으로도 얻을 수 있습니다. Ex) Doc2Vec, ELMo 등..

- **DTM과 TF-IDF를 이용한 NLP**

- DTM과 TF-IDF는 사실 일반적으로 머신 러닝 자연어 처리의 입력으로 사용.
- 인공 신경망의 입력으로 사용하는 경우 (최근에는 흔한 경우는 아님.)
- 서비스 관점에서 테스트 속도가 빠른 머신 러닝 모델 Vs. 성능이 약간 더 좋지만 테스트 속도가 더 느린 딥 러닝 모델. 과 같은 경우에 선택 가능.
- TF-IDF 가중치를 Word Embedding 벡터에 곱하여 사용하는 경우.

케라스와 신경망: 훑어보기

Machine Learning

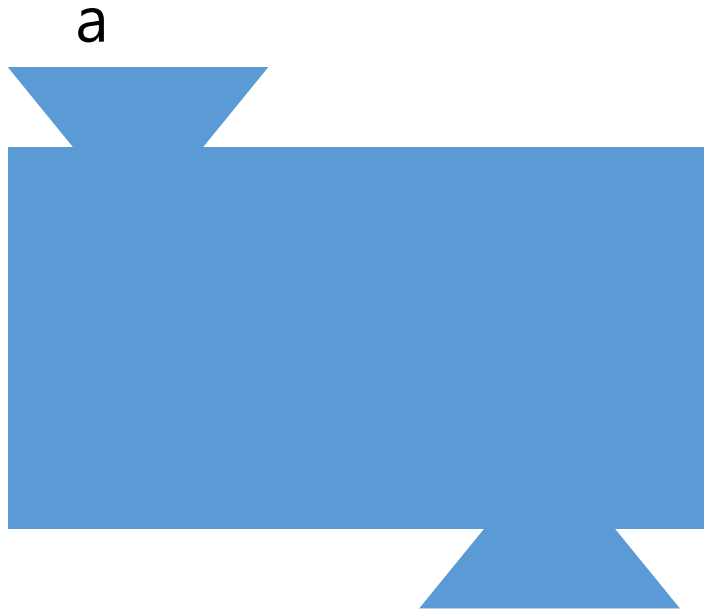
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



예를 들어 더하기를 만드는 프로그램

Machine Learning

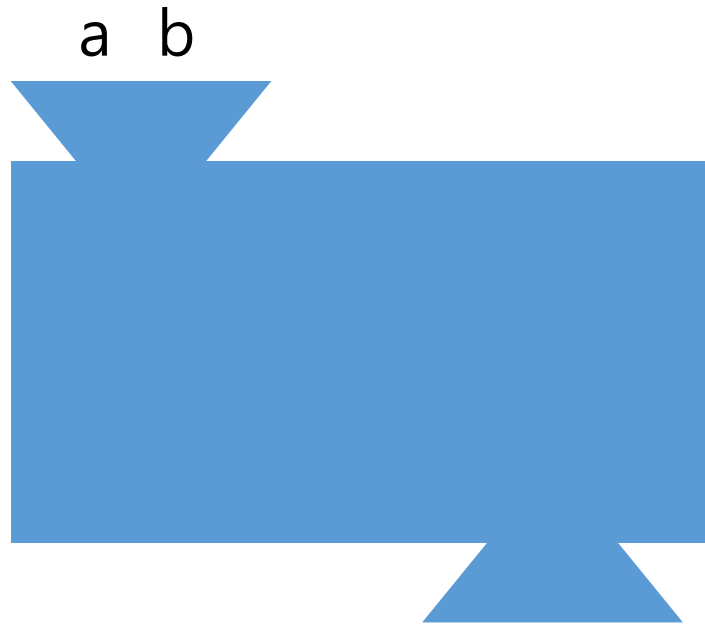
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



예를 들어 더하기를 만드는 프로그램은
a라는 변수와

Machine Learning

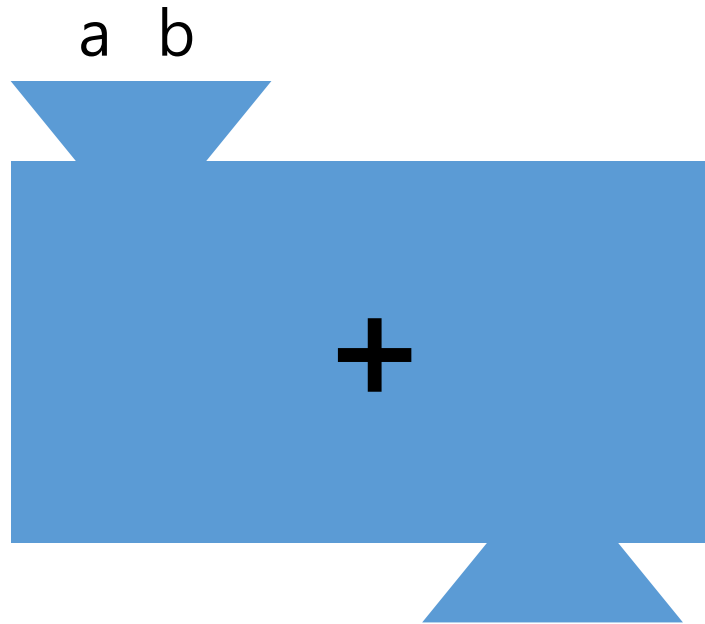
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



예를 들어 더하기를 만드는 프로그램은
a라는 변수와 b라는 변수가 왔을 때

Machine Learning

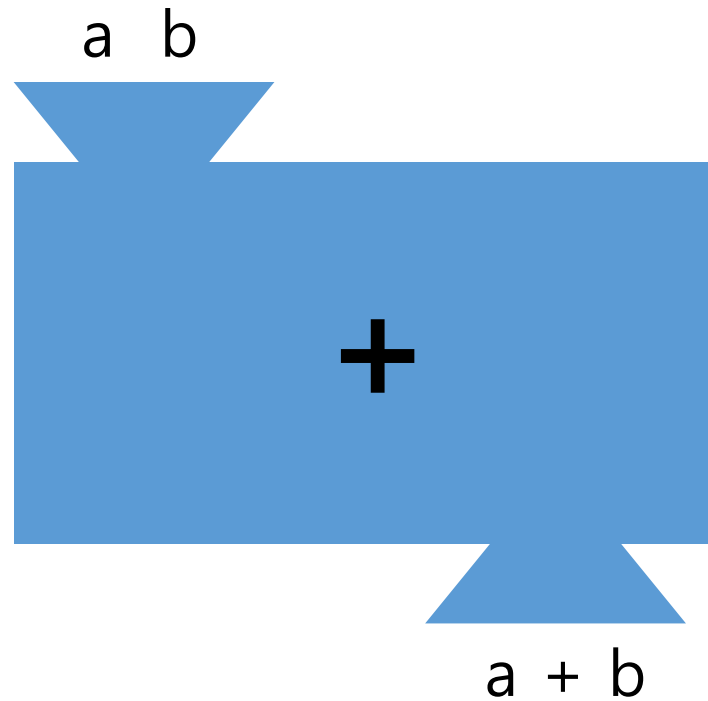
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



예를 들어 더하기를 만드는 프로그램은
a라는 변수와 b라는 변수가 왔을 때
두 개를 더해 결론을 내는 프로그램을 만든다.

Machine Learning

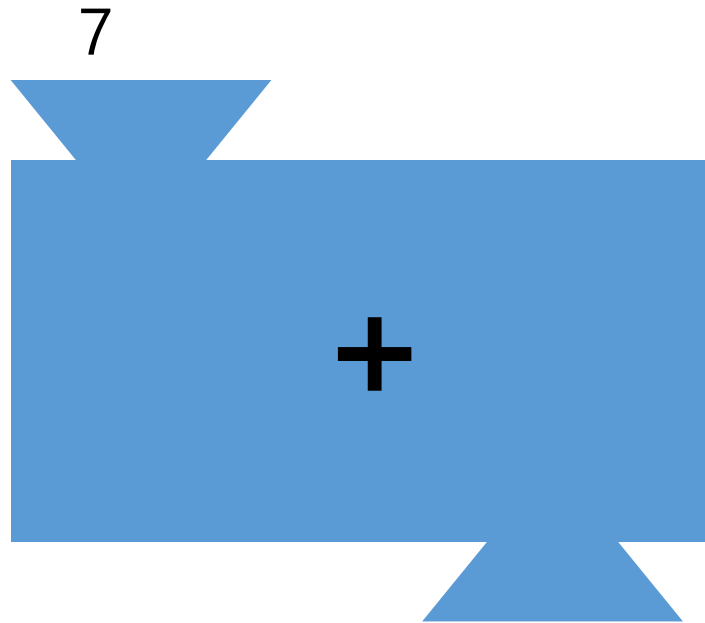
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



예를 들어 더하기를 만드는 프로그램은
a라는 변수와 b라는 변수가 왔을 때
두 개를 더해 결론을 내는 프로그램을 만든다.

Machine Learning

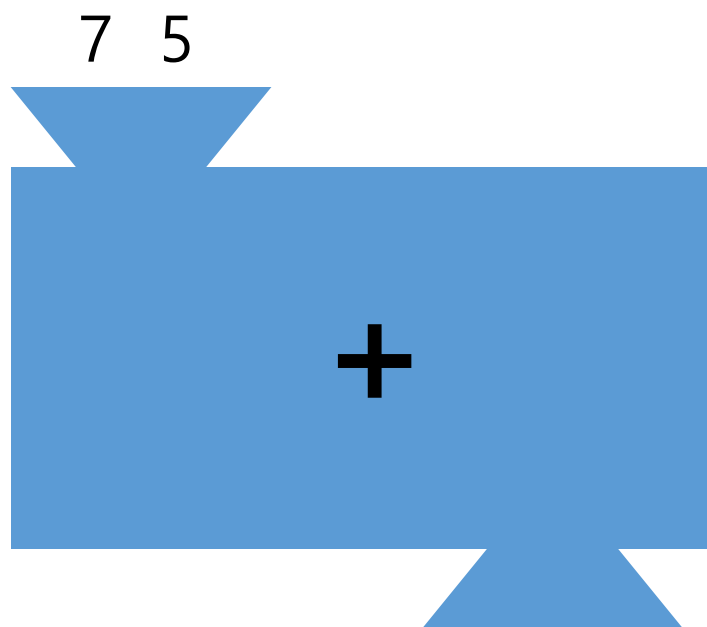
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



7이 들어오고

Machine Learning

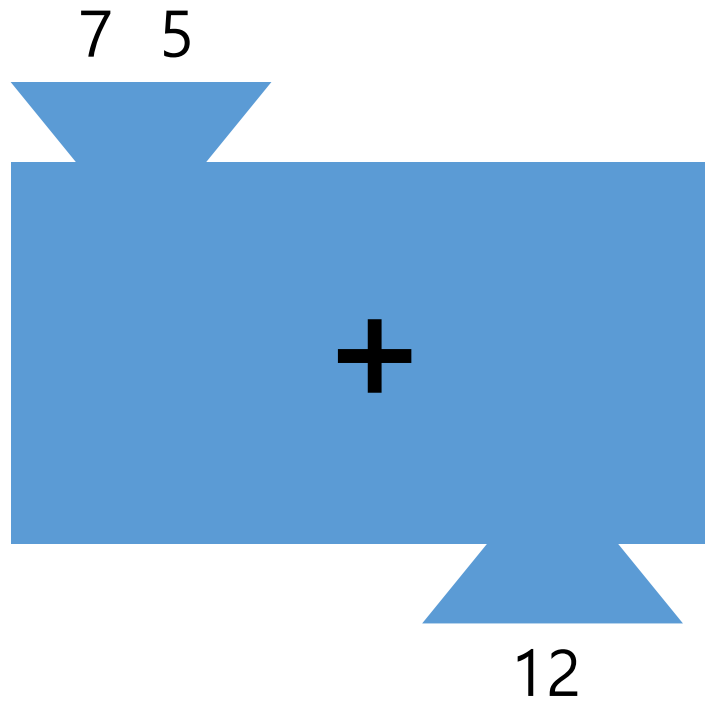
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



7이 들어오고 5가 들어오면

Machine Learning

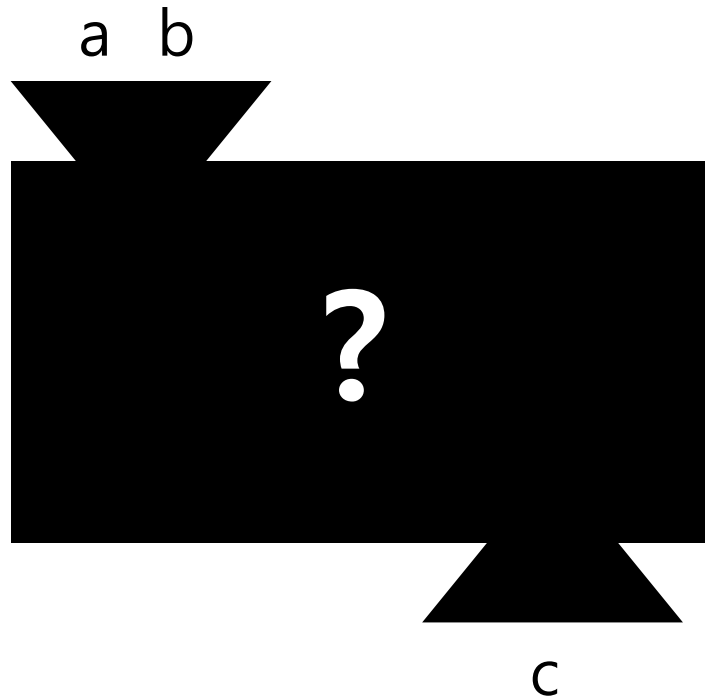
○ 기존의 프로그램 구현 방식(머신 러닝 이전)



7이 들어오고 5가 들어오면 12가 나온다.

Machine Learning

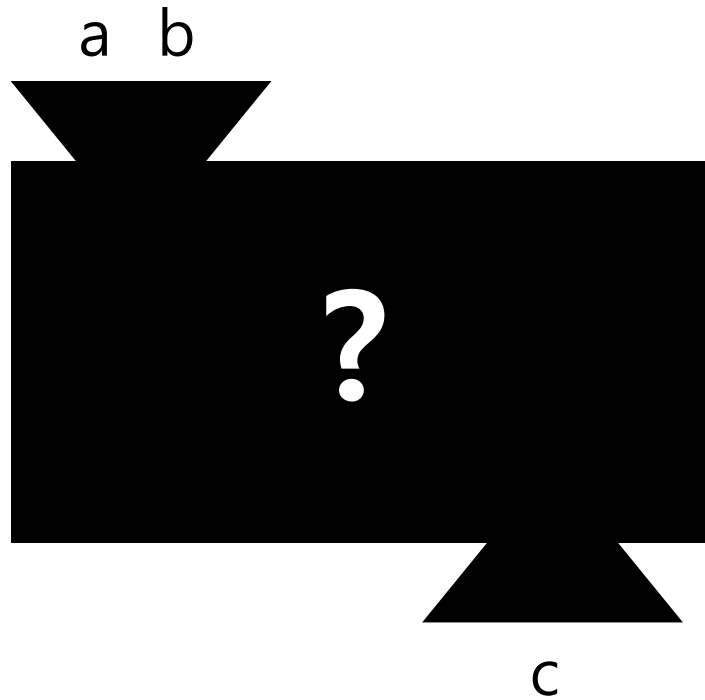
○ 머신 러닝 방식



머신 러닝은 블랙 박스 안에서

Machine Learning

○ 머신 러닝 방식

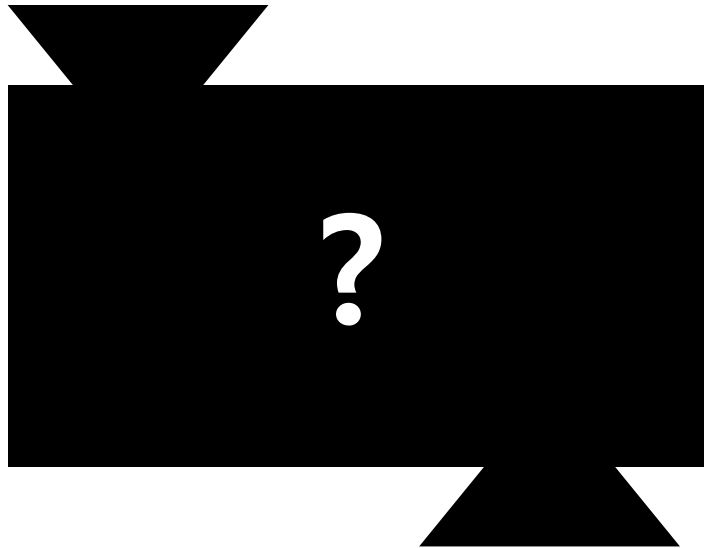


머신 러닝은 블랙 박스 안에서
무슨 일이 일어나는지 정확히
사람이 정의하기 어려운 경우 사용한다.

Machine Learning

☐ 머신 러닝 방식

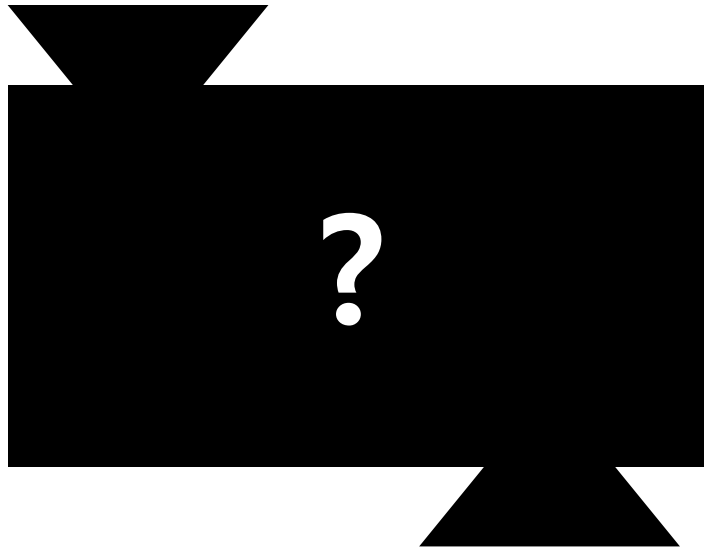
2



Machine Learning

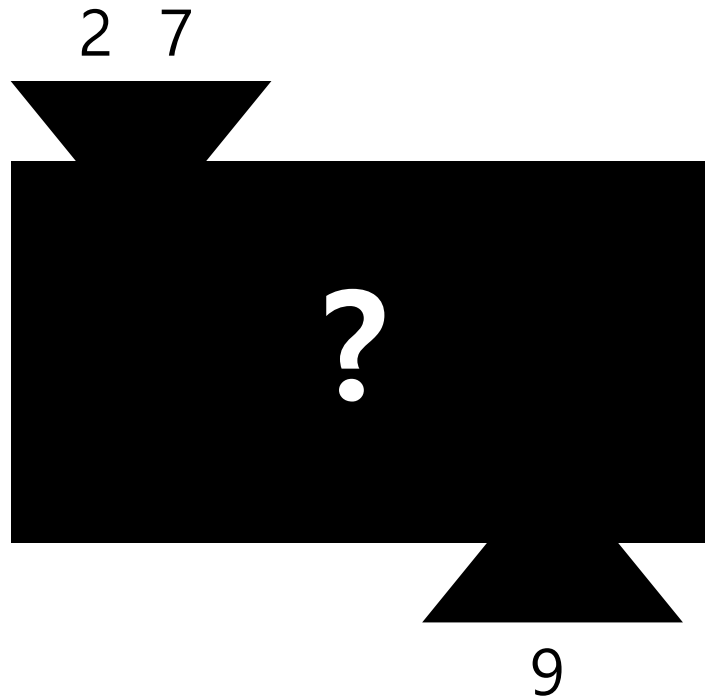
☐ 머신 러닝 방식

2 7



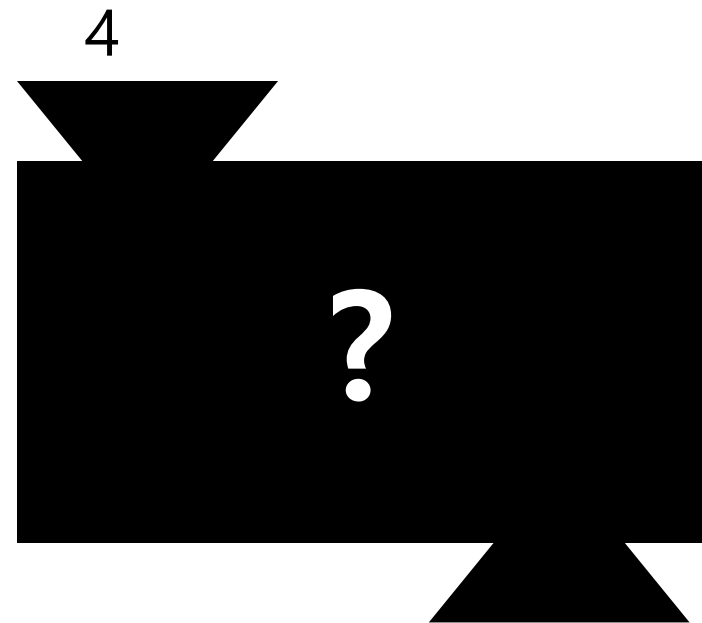
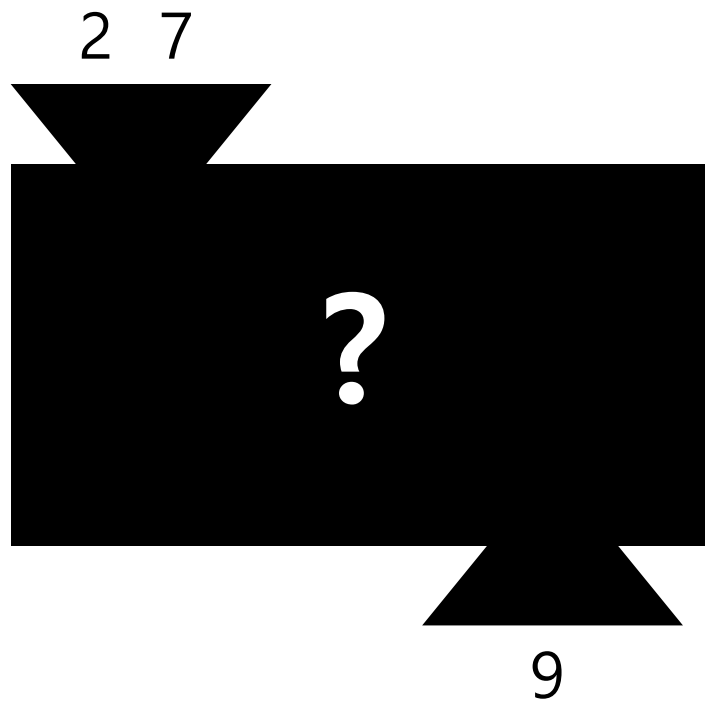
Machine Learning

☐ 머신 러닝 방식



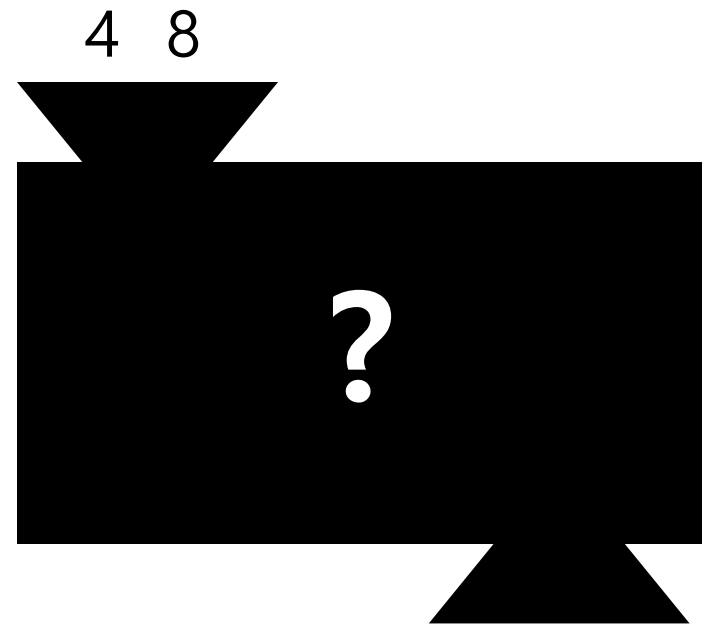
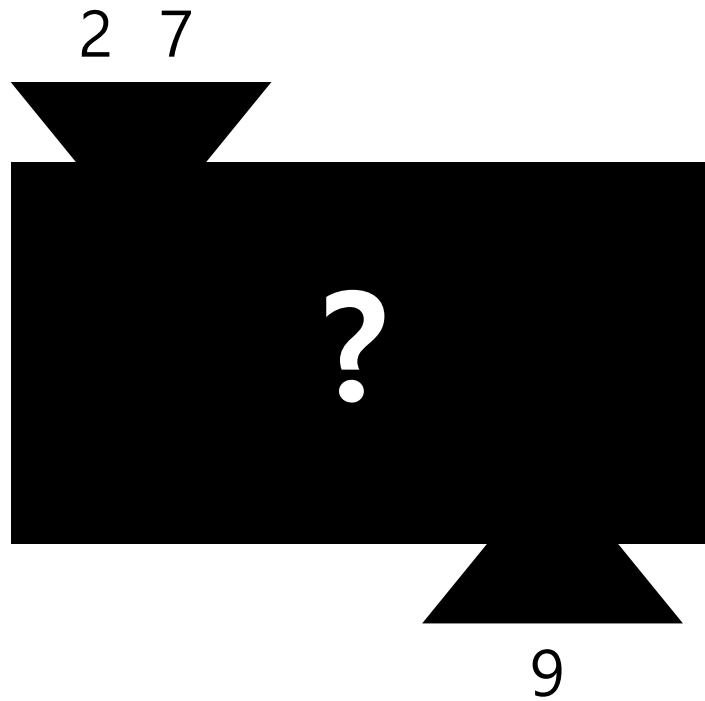
Machine Learning

○ 머신 러닝 방식



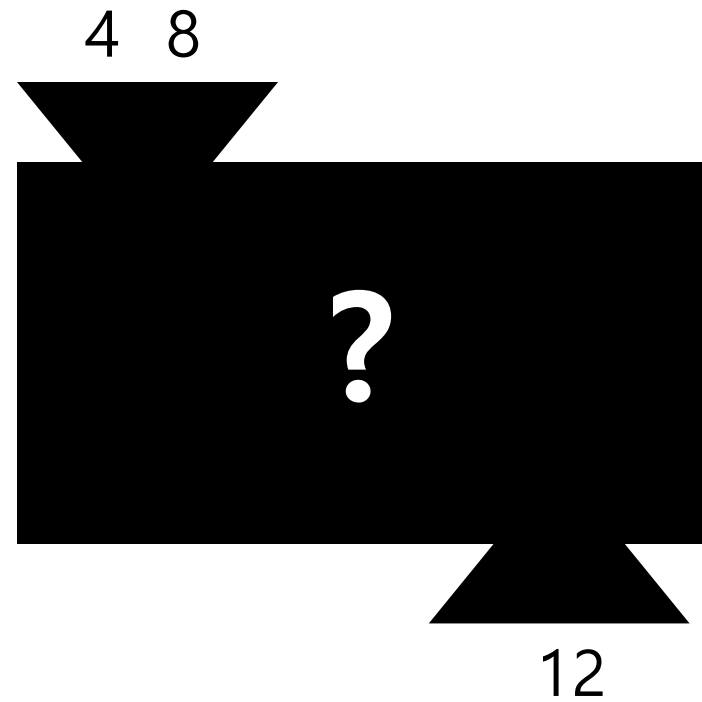
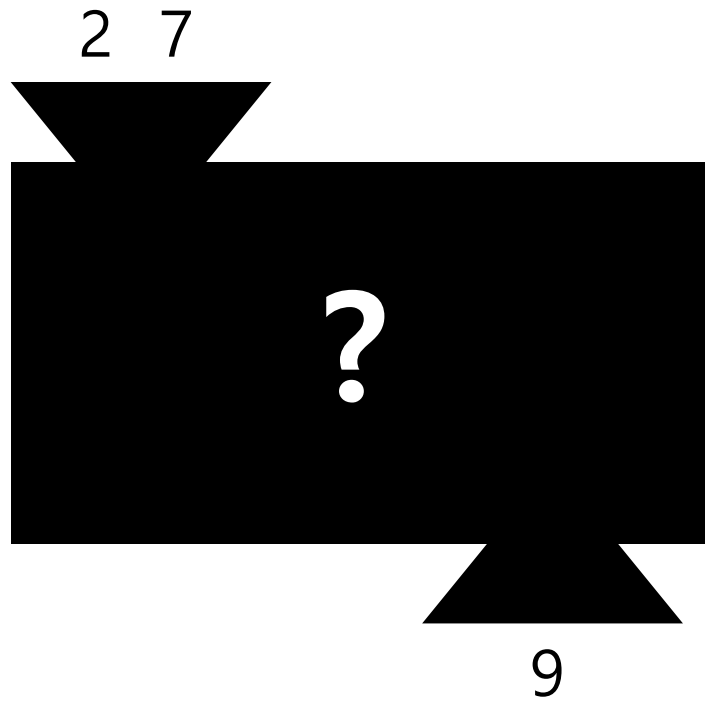
Machine Learning

○ 머신 러닝 방식



Machine Learning

○ 머신 러닝 방식

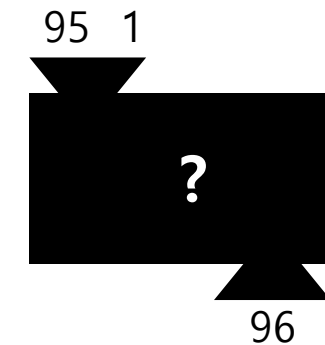
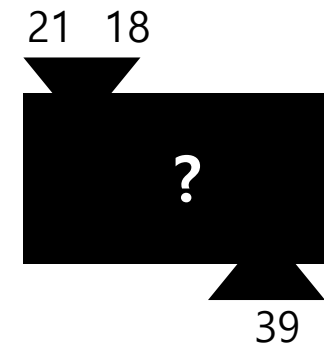
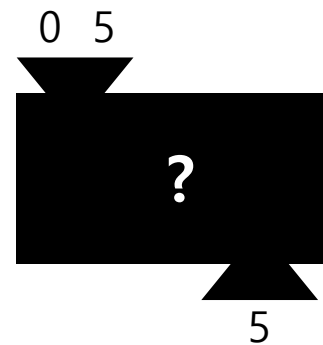
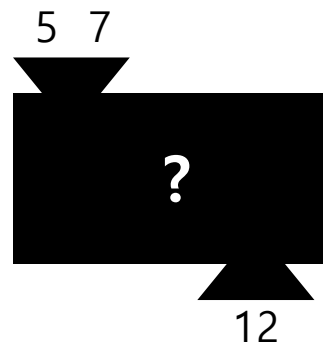
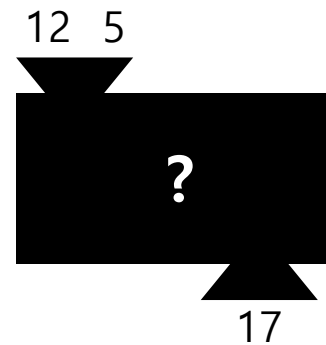
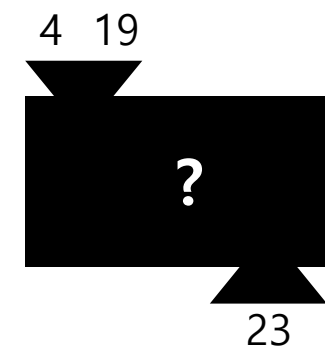
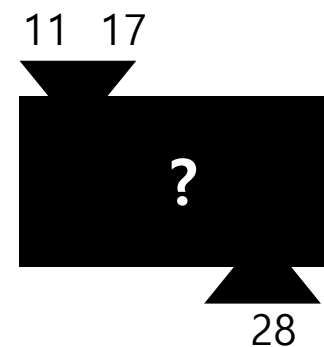
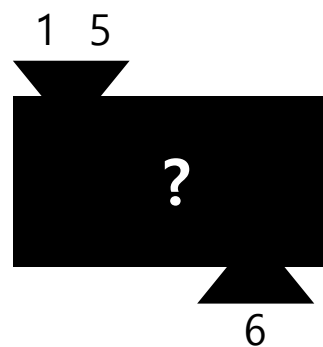
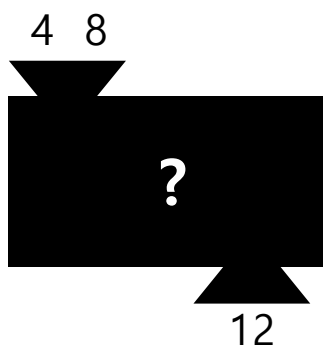
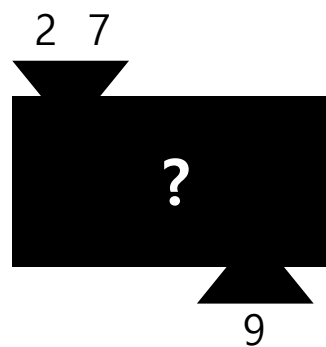


Machine Learning



머신 러닝 방식

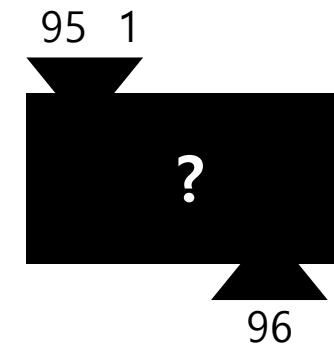
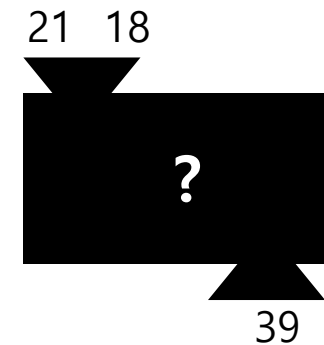
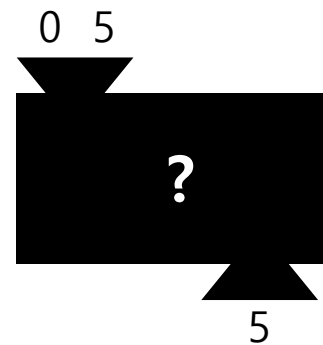
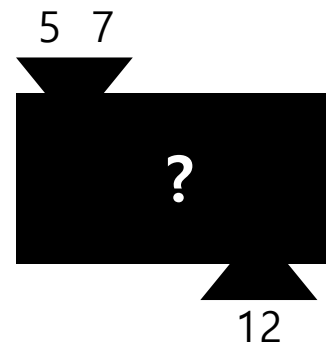
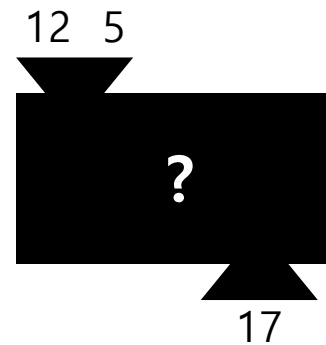
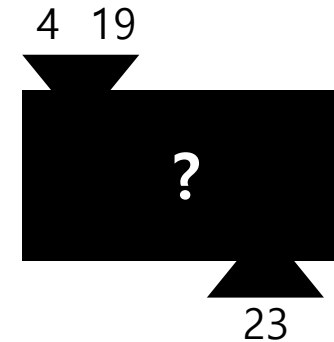
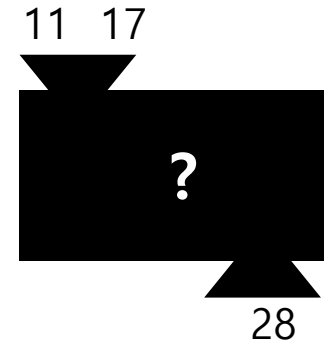
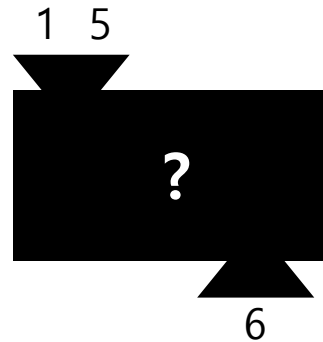
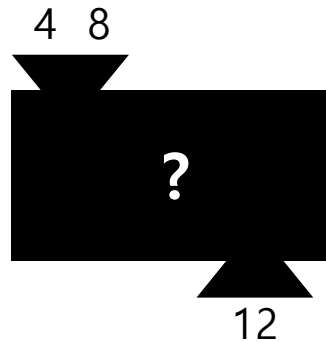
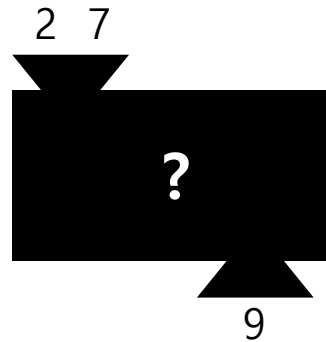
이런 데이터들은 다수 존재하지만



Machine Learning

○ 머신 러닝 방식

이 연산의 정체를 정확히 모르는 상황

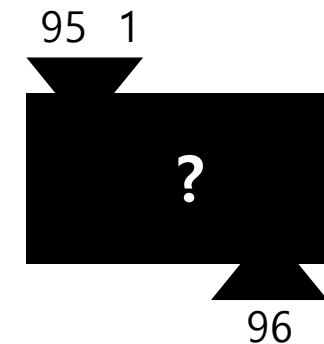
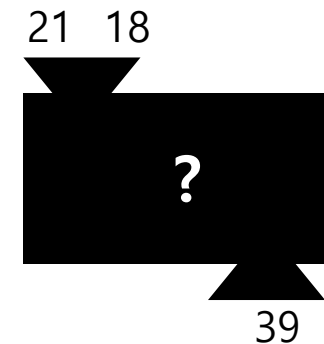
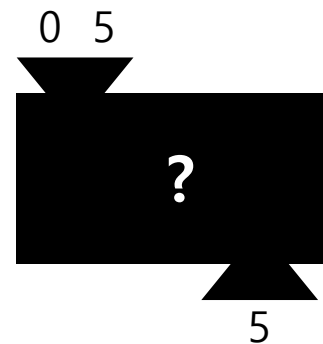
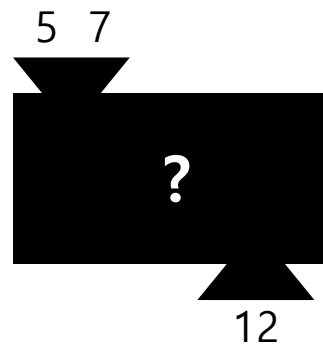
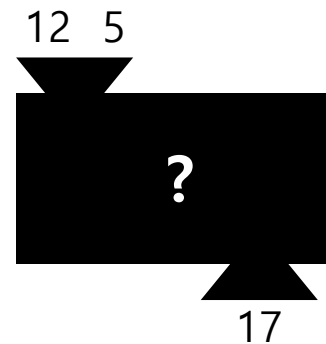
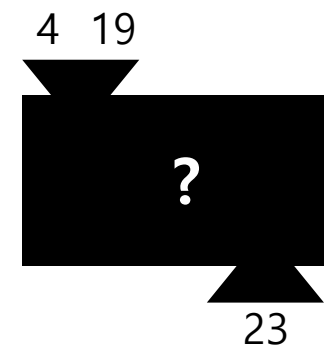
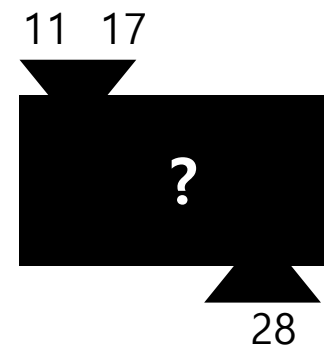
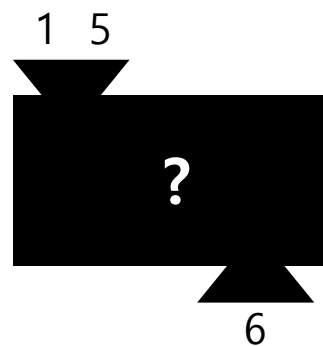
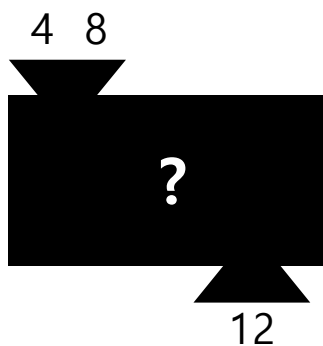
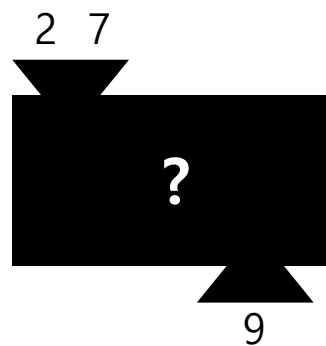


Machine Learning



머신 러닝 방식

?의 정체를 찾아야 한다.



Machine Learning



Machine Learning

사람이 찾아서
기계한테 알려준다.

기존의 프로그래밍 방식



Machine Learning

사람이 찾아서
기계한테 알려준다.

기존의 프로그래밍 방식

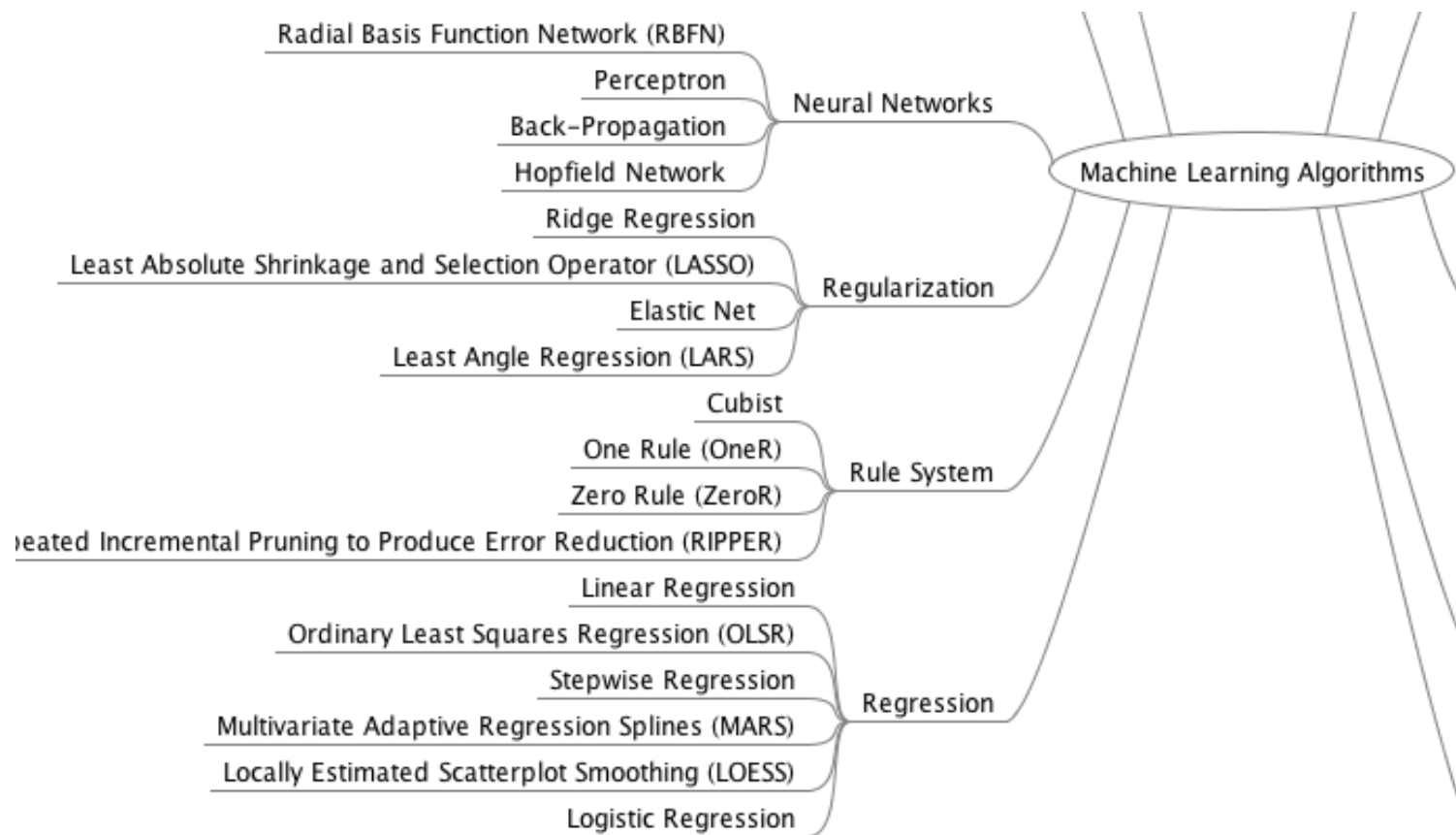


데이터를 많이주고
기계가 찾게한다.

머신 러닝(Machine Learning)

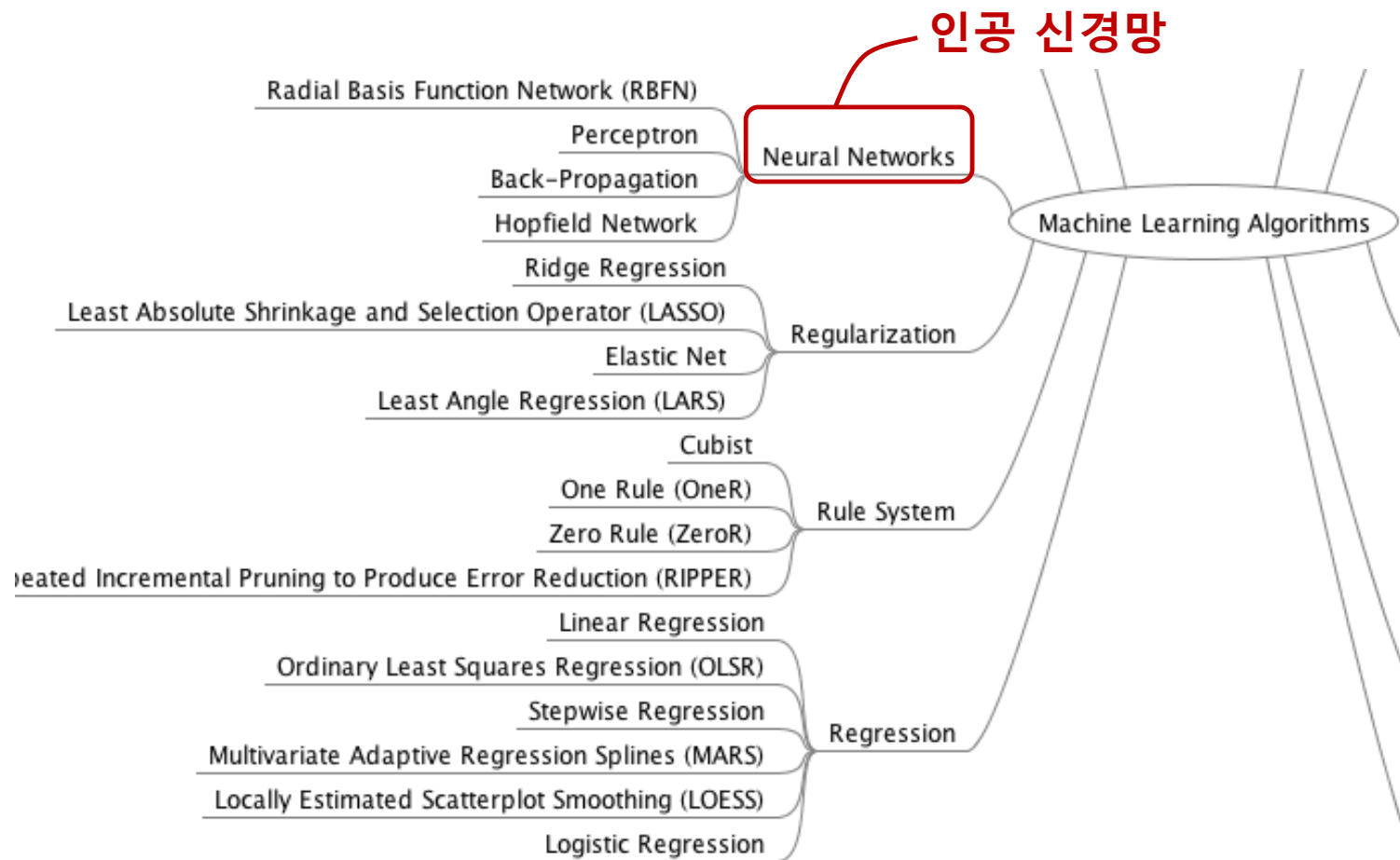
Machine Learning

○ 다양한 머신 러닝 알고리즘



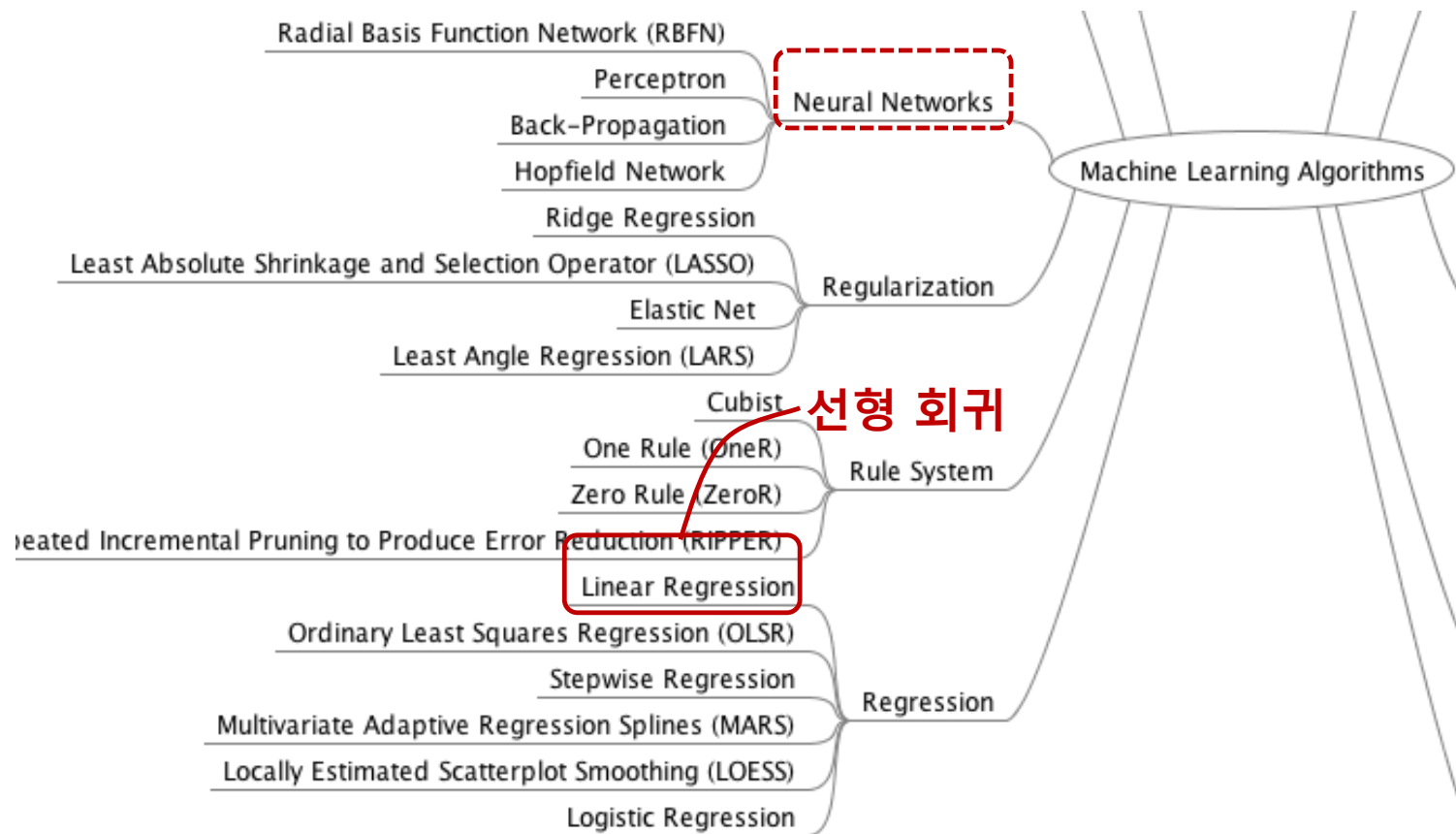
Machine Learning

○ 다양한 머신 러닝 알고리즘



Machine Learning

○ 다양한 머신 러닝 알고리즘



Linear Regression

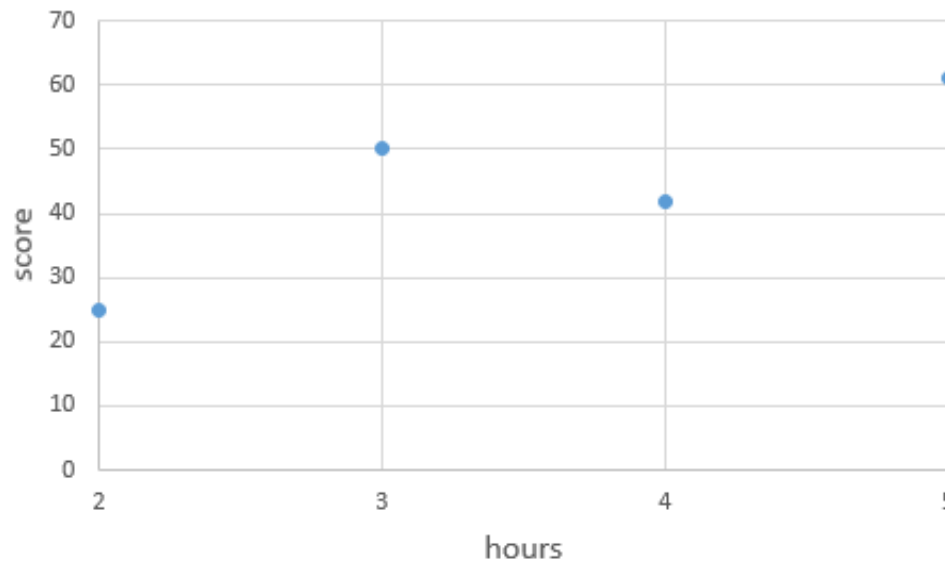
Machine Learning

- 머신 러닝 모델은 기본적으로 다음의 세 가지 과정을 거칩니다.
 - 1) 학습하고자 하는 가설(Hypothesis)을 세운다.
 - 2) 가설의 성능을 측정할 수 있는 손실 함수(Loss Function)를 정의한다.
 - 3) 손실 함수 L 을 최소화 할 수 있는 학습 알고리즘을 설계한다.

Linear Regression

- 데이터를 반영하는 가장 적절한 선을 긋는 작업.

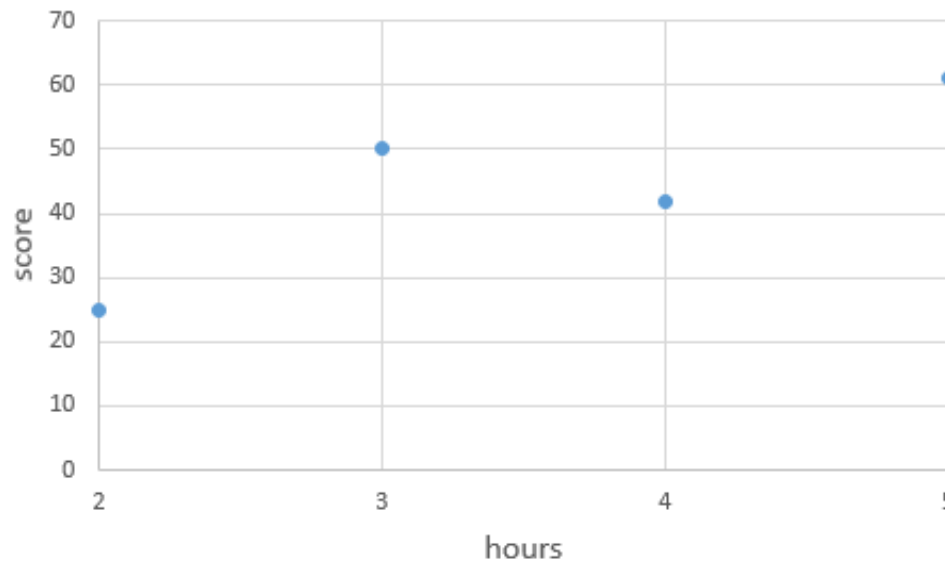
hours	score
2	25
3	50
4	42
5	61



Linear Regression

- 데이터를 반영하는 가장 적절한 선을 긋는 작업.
- 선을 그은 후에는?

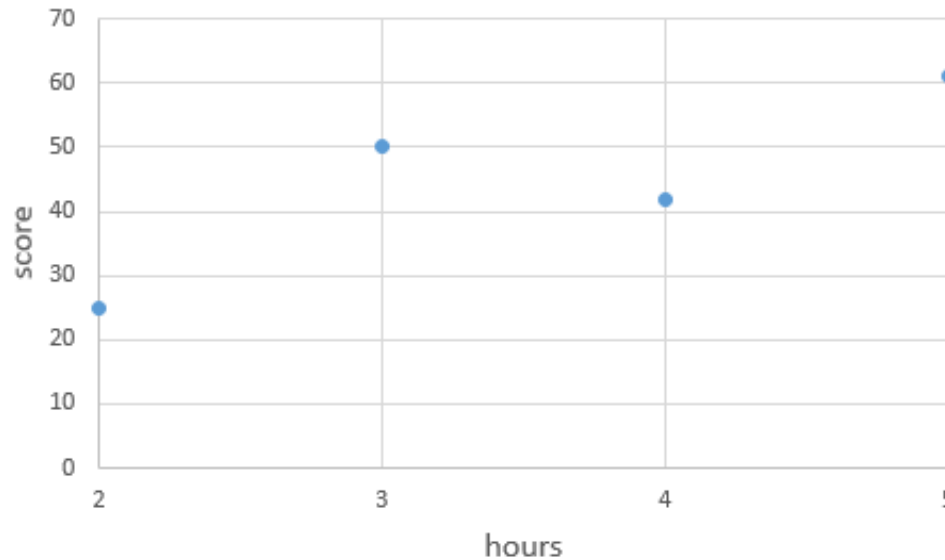
hours	score
2	25
3	50
4	42
5	61



Linear Regression

- 데이터를 반영하는 가장 적절한 선을 긋는 작업.
- 선을 그은 후에는?
- 3시간 22분, 7시간 17분, 12시간 등의 임의의 시간으로부터 점수를 예측 가능!

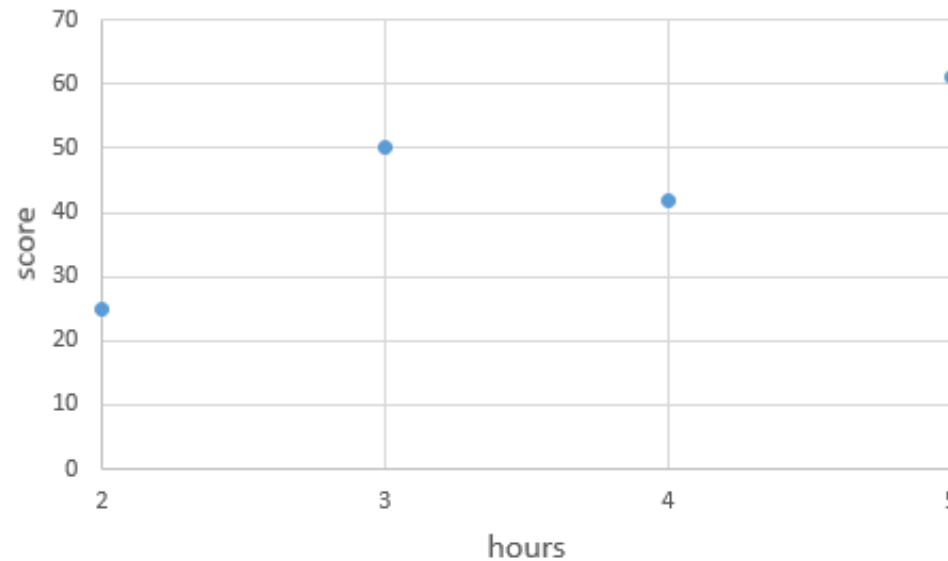
hours	score
2	25
3	50
4	42
5	61



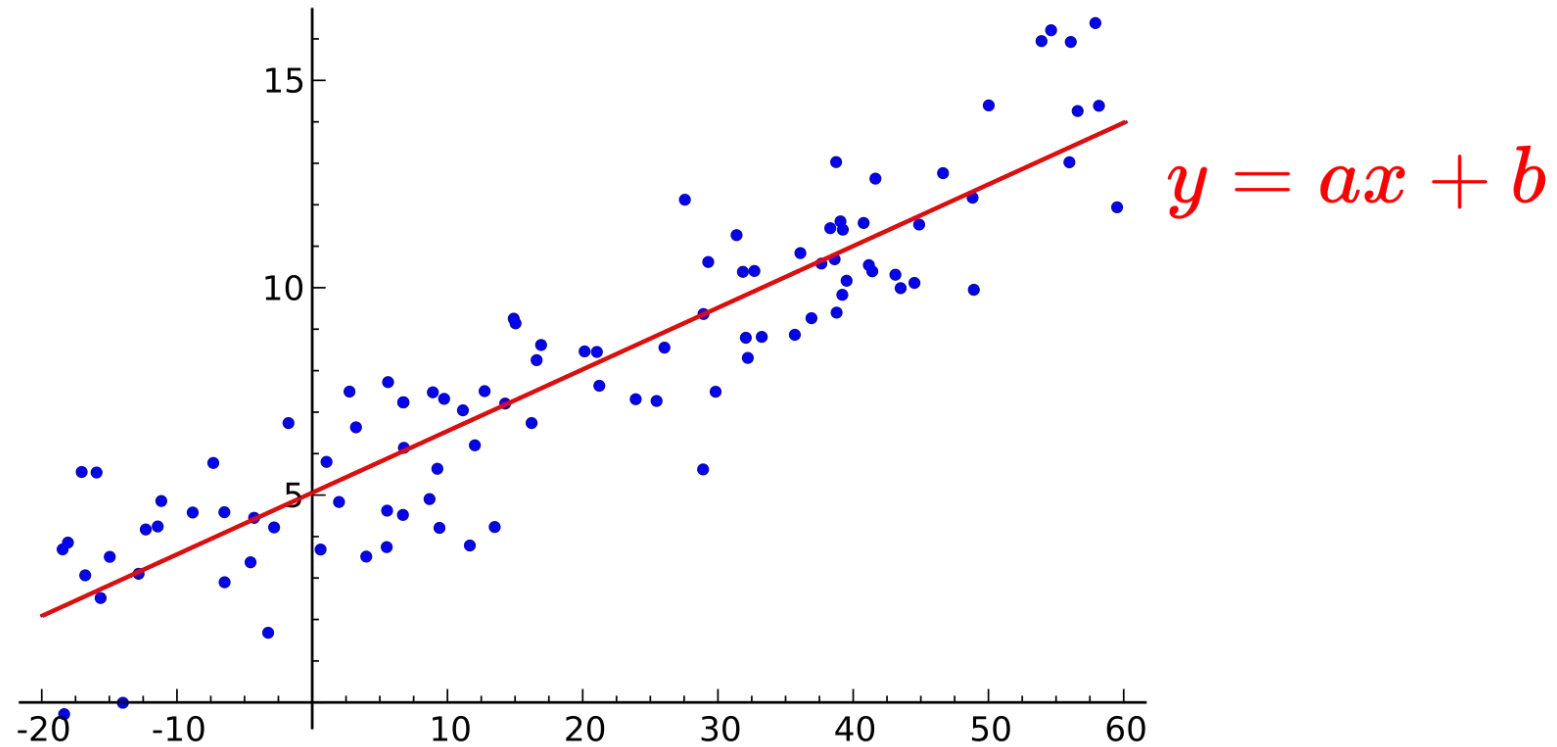
Linear Regression

- 직선을 표현하는 식은 어떤 식이 있을까?

hours	score
2	25
3	50
4	42
5	61



Linear Regression



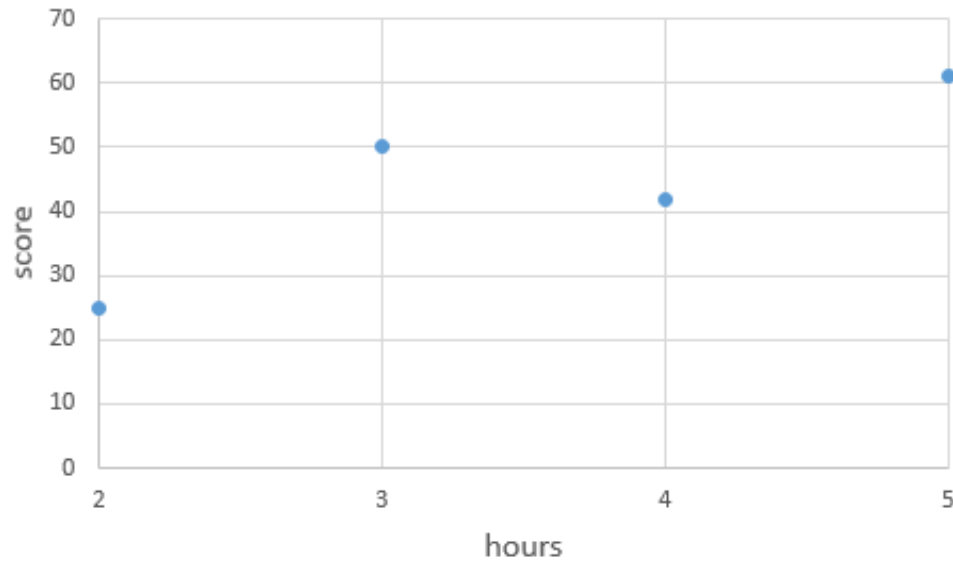
https://en.wikipedia.org/wiki/Linear_regression

Linear Regression

- 직선을 표현하는 식은 어떤 식이 있을까?

$$y = wx + b$$

hours	score
2	25
3	50
4	42
5	61

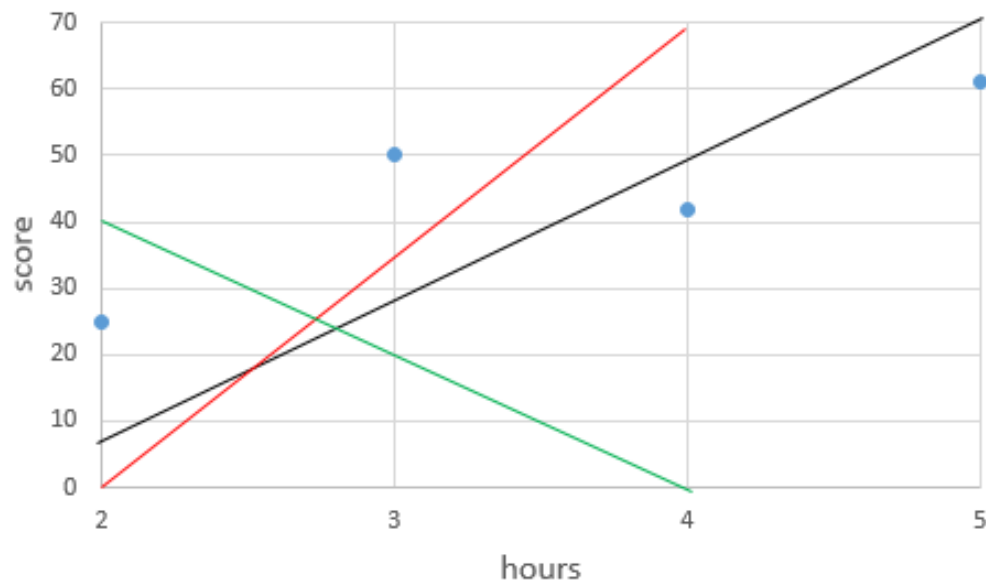


Linear Regression

- w 와 b 의 값에 따라서 표현되는 직선이 달라진다.

$$y = wx + b$$

hours	score
2	25
3	50
4	42
5	61

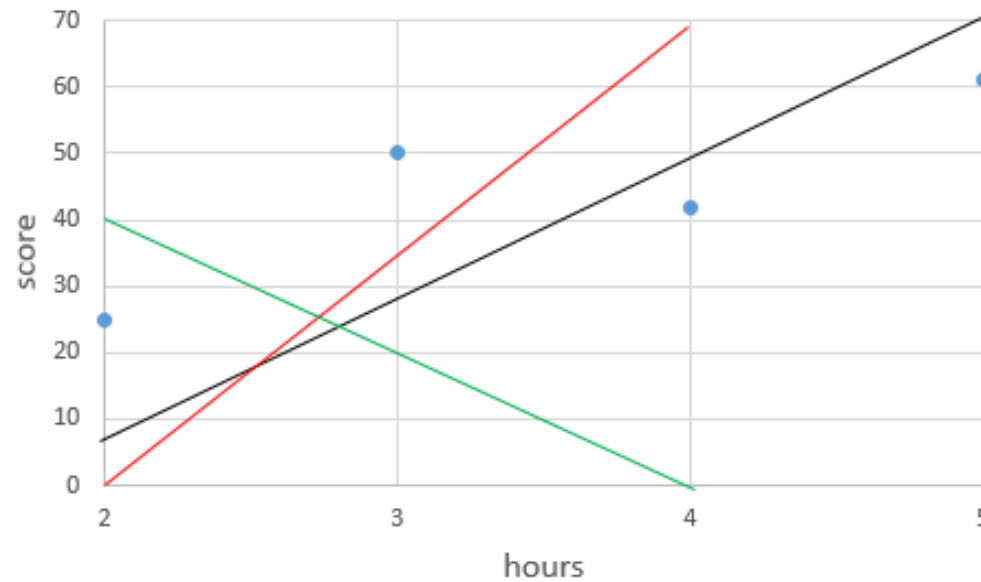


Linear Regression

- w 와 b 의 값에 따라서 표현되는 직선이 달라진다.
- 선형 회귀는 결국 데이터를 적절히 반영하는 w 와 b 를 찾아내는 작업.

$$y = wx + b$$

hours	score
2	25
3	50
4	42
5	61



Linear Regression

- 아래의 3개의 직선 중 데이터를 가장 잘 반영하는 직선은?

$$y = wx + b$$

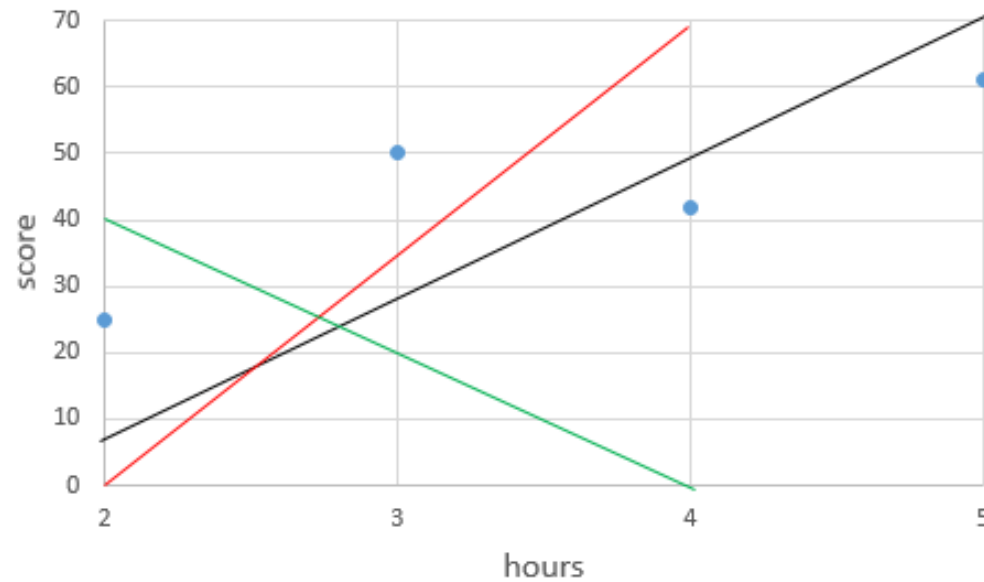
hours	score
2	25
3	50
4	42
5	61

Linear Regression

- 아래의 3개의 직선 중 데이터를 가장 잘 반영하는 직선은?
- 이를 수치적으로 표현하여 판단할 수 있어야 한다.

$$y = wx + b$$

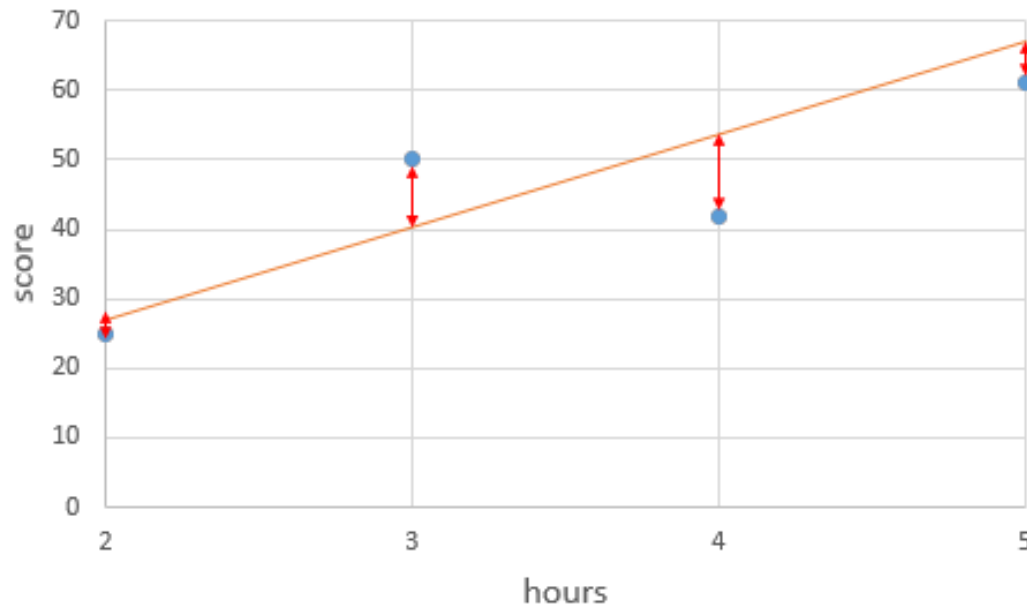
hours	score
2	25
3	50
4	42
5	61



Linear Regression

- 수치적으로 표현하여 판단할 수 있어야 한다.
- 이를 판단하기 위해서 오차(Error)라는 개념을 도입하자.

$$y = wx + b$$

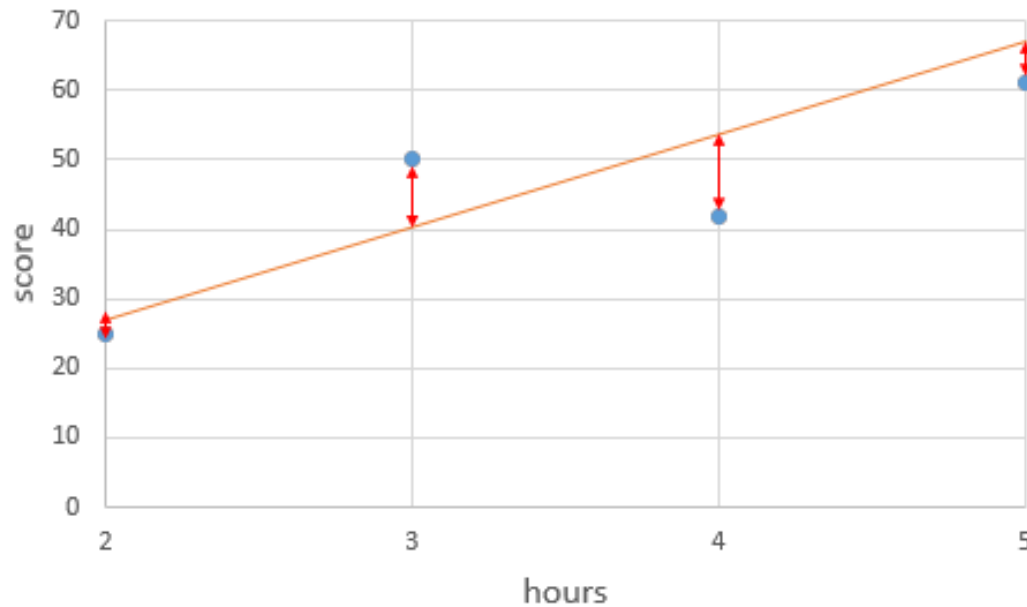


위 직선은 $w = 13$, $b = 1$ 인 임의의 직선

Linear Regression

- 이를 판단하기 위해서 오차(Error)라는 개념을 도입하자.
- 아래의 그림에서 \updownarrow 는 각 점에서의 오차의 크기를 보여준다.

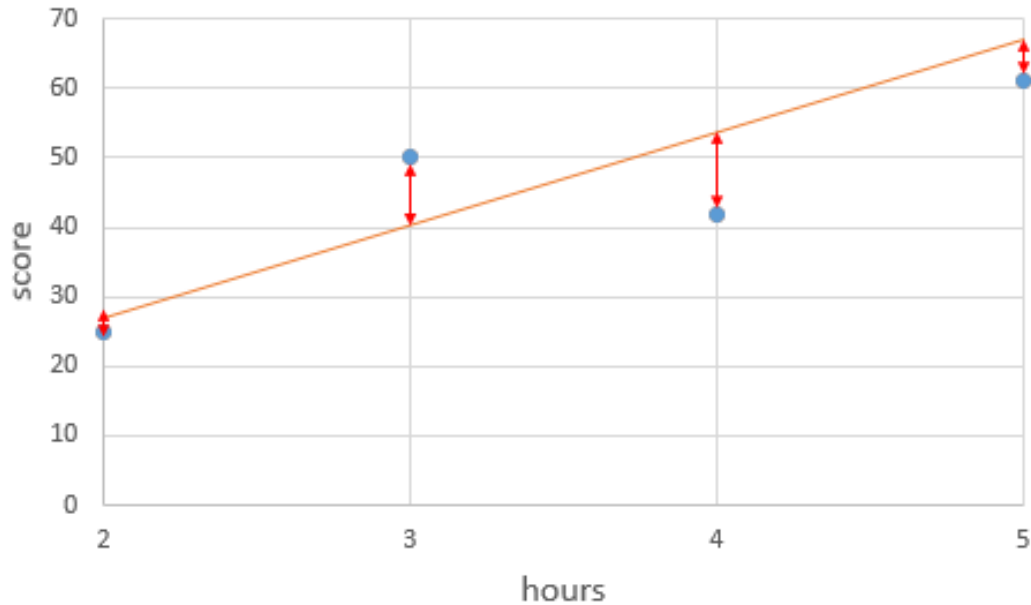
$$y = wx + b$$



Linear Regression

- 우리의 목표는 오차를 최소로 만드는 w 와 b 를 찾아내는 것.
- 오차의 크기를 측정하는 가장 기본적인 방법은 모든 오차를 더하는 것.

$$y = wx + b$$

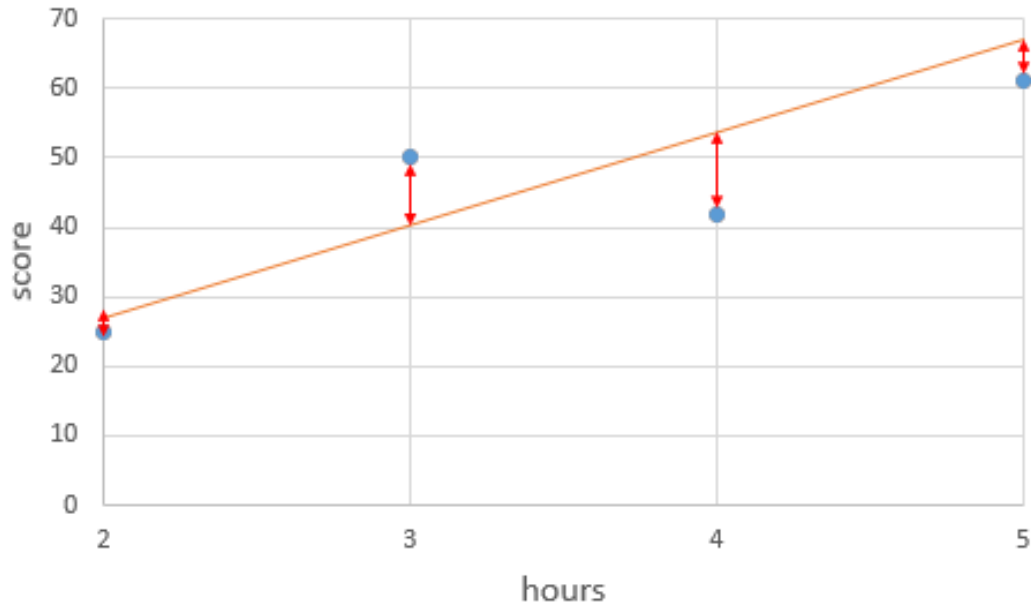


hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

Linear Regression

- 우리의 목표는 오차를 최소로 만드는 w 와 b 를 찾아내는 것.
- 오차의 크기를 측정하는 가장 기본적인 방법은 모든 오차를 더하는 것.

$$y = wx + b$$

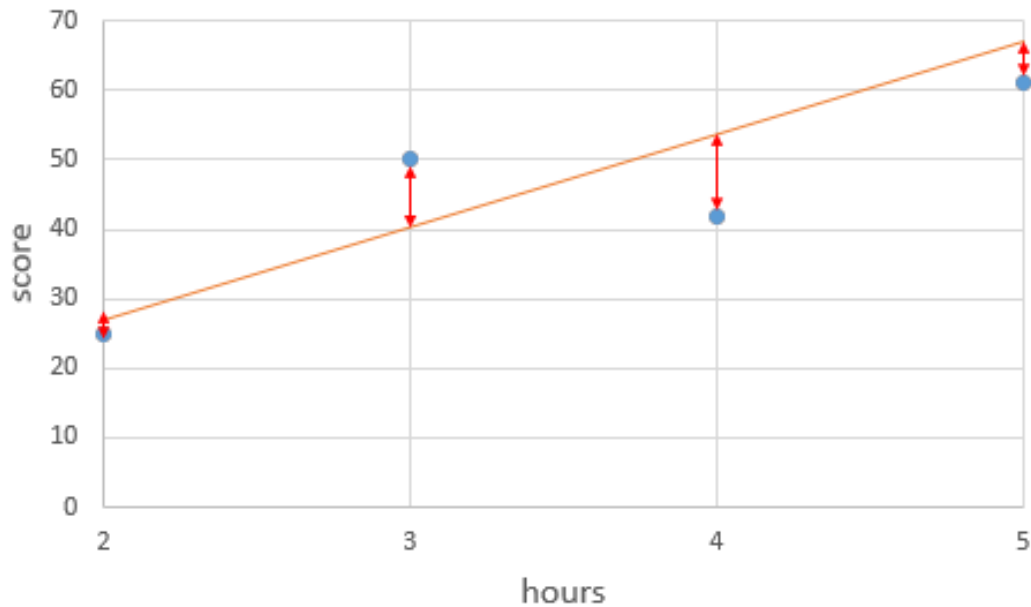


hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

Linear Regression

- 우리의 목표는 오차를 최소로 만드는 w 와 b 를 찾아내는 것.
- 오차의 크기를 측정하는 가장 기본적인 방법은 모든 오차를 더하는 것.

$$y = wx + b$$



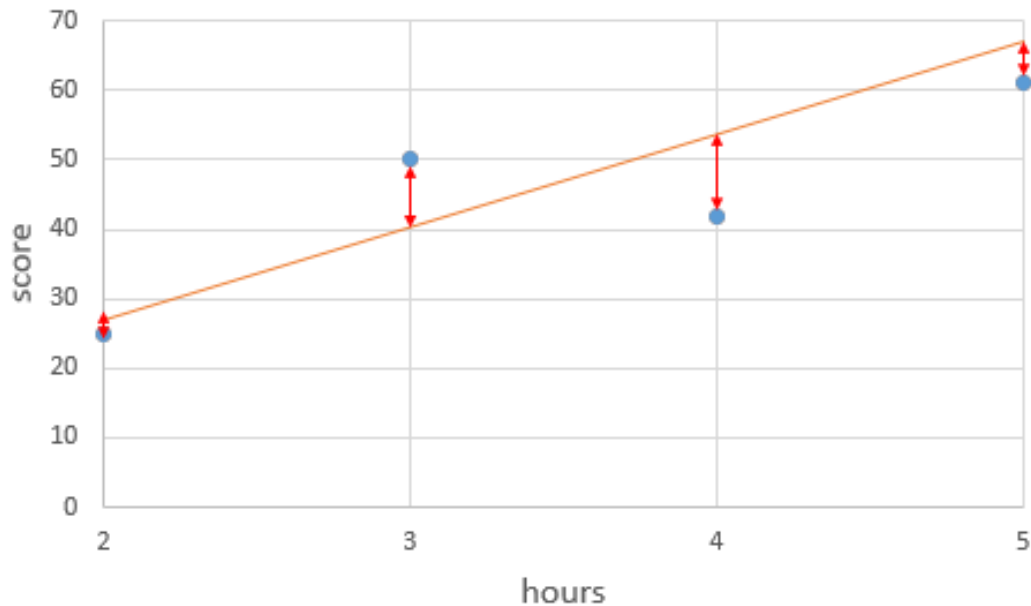
hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$(-2) + 10 + (-7) + (-5) = ?$$

Linear Regression

- 우리의 목표는 오차를 최소로 만드는 w 와 b 를 찾아내는 것.
- 오차의 크기를 측정하는 가장 기본적인 방법은 모든 오차를 더하는 것.

$$y = wx + b$$



hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

음수 오차와 양수 오차가 섞여있어
제대로 된 오차의 양을 구할 수 없다.

Mean Square Error

- 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 이를 '제곱 오차'라 하자.

hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2 = (-2)^2 + 10^2 + (-7)^2 + (-5)^2 = 178$$

Mean Square Error

- 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 데이터의 개수가 n 이라고 할 때, 데이터의 개수인 n 으로 나누면 모든 데이터의 오차 평균을 얻는다. 이를 '평균 제곱 오차'라 한다.

hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2 = (-2)^2 + 10^2 + (-7)^2 + (-5)^2 = 178$$

$$\frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2 = 178/4 = 44.5$$

Mean Square Error

- 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 데이터의 개수가 n 이라고 할 때, 데이터의 개수인 n 으로 나누면 모든 데이터의 오차 평균을 얻는다. 이를 '평균 제곱 오차'라 한다.

hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = (-2)^2 + 10^2 + (-7)^2 + (-5)^2 = 178$$

$$\frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = 178/4 = 44.5$$

이를 평균 제곱 오차라고 한다.
이 평균 제곱 오차가 최소가 되는
 w 와 b 를 찾아내는 것이 목표다.

$$\underset{W, b}{\text{minimize}} \text{cost}(W, b)$$

Cost Function

머신 러닝에서는 실제값과 예측값에 대한 오차에 대한 식을 Cost function이라 한다.
적어도 이 강의 내에서는 아래는 같은 용어로 사용한다.

목적 함수(Objective function) = 비용 함수(Cost function) = 손실 함수(Loss function)

Cost Function

머신 러닝에서는 실제값과 예측값에 대한 오차에 대한 식을 Cost function이라 한다. 적어도 이 강의 내에서는 아래는 같은 용어로 사용한다.

목적 함수(Objective function) = 비용 함수(Cost function) = 손실 함수(Loss function)

인공 신경망은 기본적으로 Cost function을 최소화하는 w 와 b 를 찾는 것이 목적!

Cost Function

머신 러닝에서는 실제값과 예측값에 대한 오차에 대한 식을 Cost function이라 한다.
적어도 이 강의 내에서는 아래는 같은 용어로 사용한다.

목적 함수(Objective function) = 비용 함수(Cost function) = 손실 함수(Loss function)

인공 신경망은 기본적으로 Cost function을 최소화하는 w 와 b 를 찾는 것이 목적!
다시 선형 회귀 이야기로 돌아가보자.

Mean Square Error

- 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 데이터의 개수가 n 이라고 할 때, 데이터의 개수인 n 으로 나누면 오차의 제곱합에 대한 평균을 얻는다.

hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = (-2)^2 + 10^2 + (-7)^2 + (-5)^2 = 178$$

$$\frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = 178/4 = 44.5$$

$$cost(w, b) = \frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2$$

 Linear Regression의 cost function

Mean Square Error

- 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 데이터의 개수가 n 이라고 할 때, 데이터의 개수인 n 으로 나누면 오차의 제곱합에 대한 평균을 얻는다.

hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = (-2)^2 + 10^2 + (-7)^2 + (-5)^2 = 178$$

$$\frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = 178/4 = 44.5$$

$$cost(w, b) = \frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2$$

$$w, b \rightarrow \text{minimize } cost(w, b)$$

Mean Square Error

- 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 데이터의 개수가 n 이라고 할 때, 데이터의 개수인 n 으로 나누면 오차의 제곱합에 대한 평균을 얻는다.

hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = (-2)^2 + 10^2 + (-7)^2 + (-5)^2 = 178$$

$$\frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2 = 178/4 = 44.5$$

$$cost(w, b) = \frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2$$

$$w, b \rightarrow \text{minimize } cost(w, b)$$

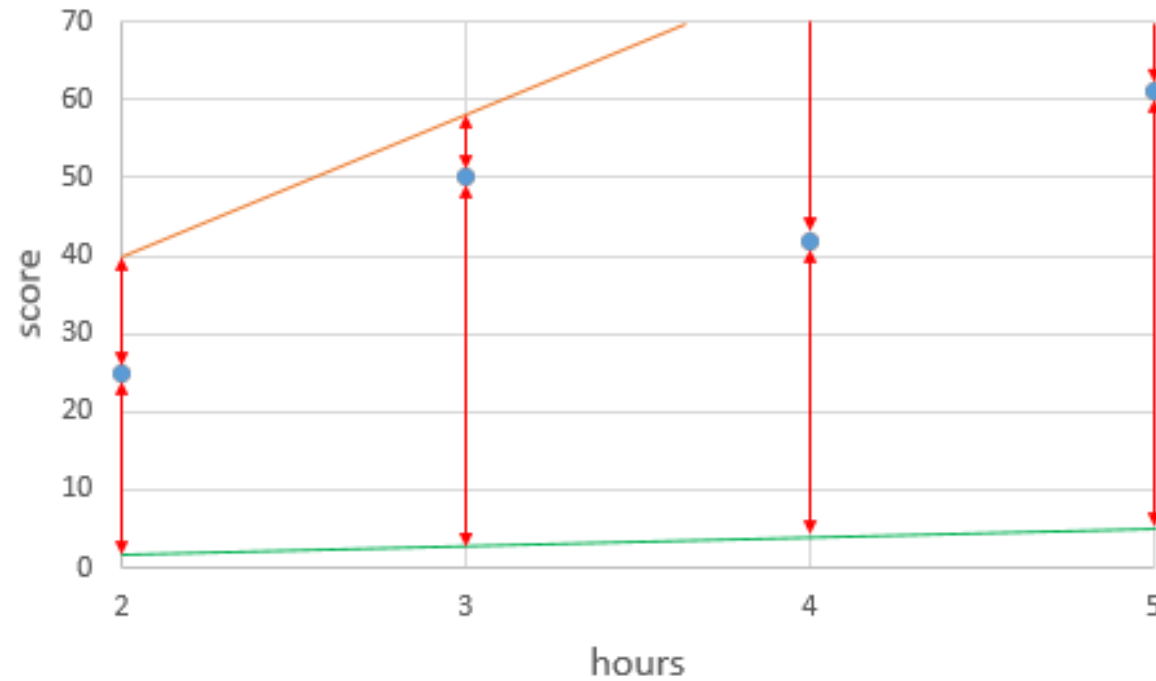
Cost function을 최소화하는 w 와 b 를 찾는 것이 목적!

Gradient Descent

- 선형 회귀를 포함한 인공 신경망에서는 비용 함수를 최소화하는 **매개 변수**를 찾는 것이 목적이다.
- 이때 사용되는 알고리즘을 **옵티마이저(Optimizer)**라고 한다.
- 경사 하강법은 대표적인 옵티마이저 알고리즘이다.

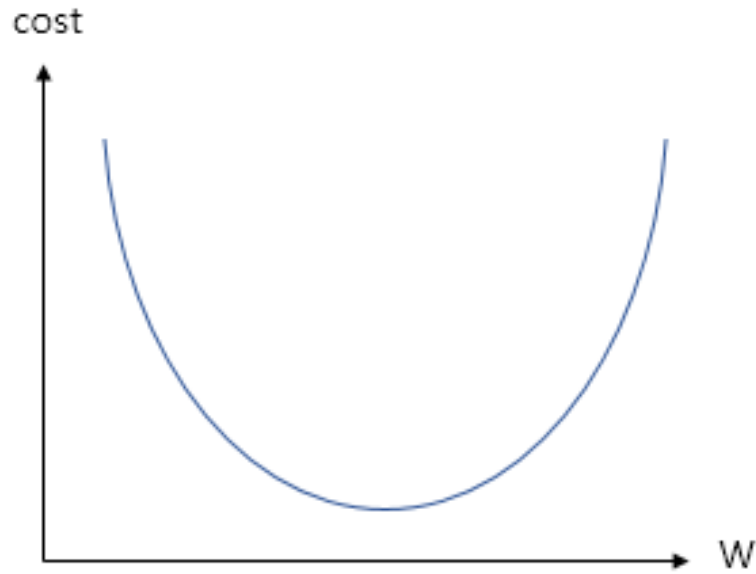
Gradient Descent

- 아래의 그림은 w 가 지나치게 높거나, 낮을 때 오차가 증가함을 보여준다.
- 사실 b 또한 마찬가지로 지나치게 높거나 낮으면 오차가 증가하게 된다.



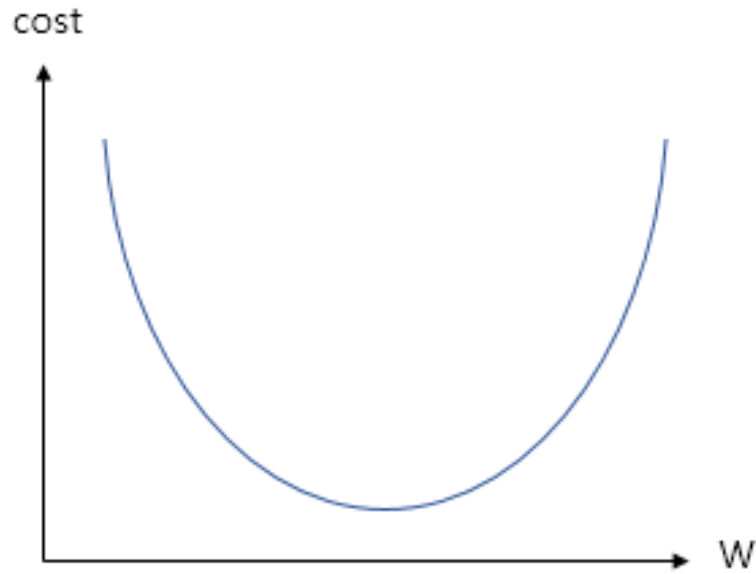
Gradient Descent

- 아래의 그림은 w 가 지나치게 높거나, 낮을 때 오차가 증가함을 보여준다.
- 사실 b 또한 마찬가지로 지나치게 높거나 낮으면 오차가 증가하게 된다.
- 이 관계를 그래프로 표현하면 아래와 같다.



Gradient Descent

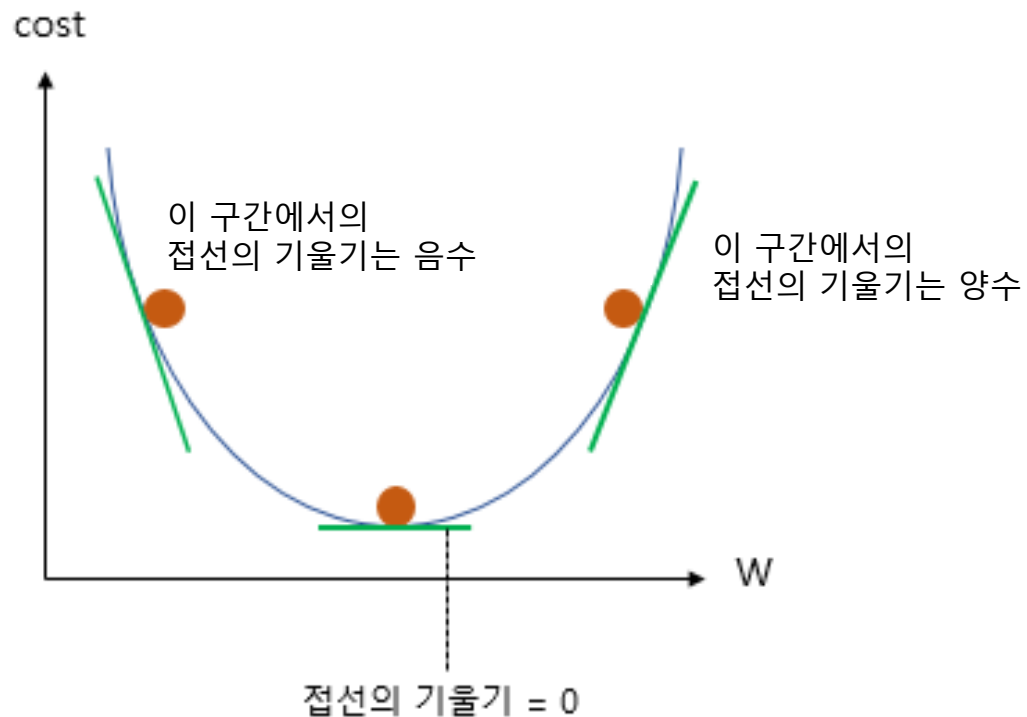
- 아래의 그림은 w 가 지나치게 높거나, 낮을 때 오차가 증가함을 보여준다.
- 사실 b 또한 마찬가지로 지나치게 높거나 낮으면 오차가 증가하게 된다.
- 이 관계를 그래프로 표현하면 아래와 같다.



경사하강법은
한 점에서의 순간 변화율 또는
접선에서의 기울기 개념을 사용한다.

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.
- 비용 함수를 W 에 대해 미분하면 접선의 기울기를 얻는다.

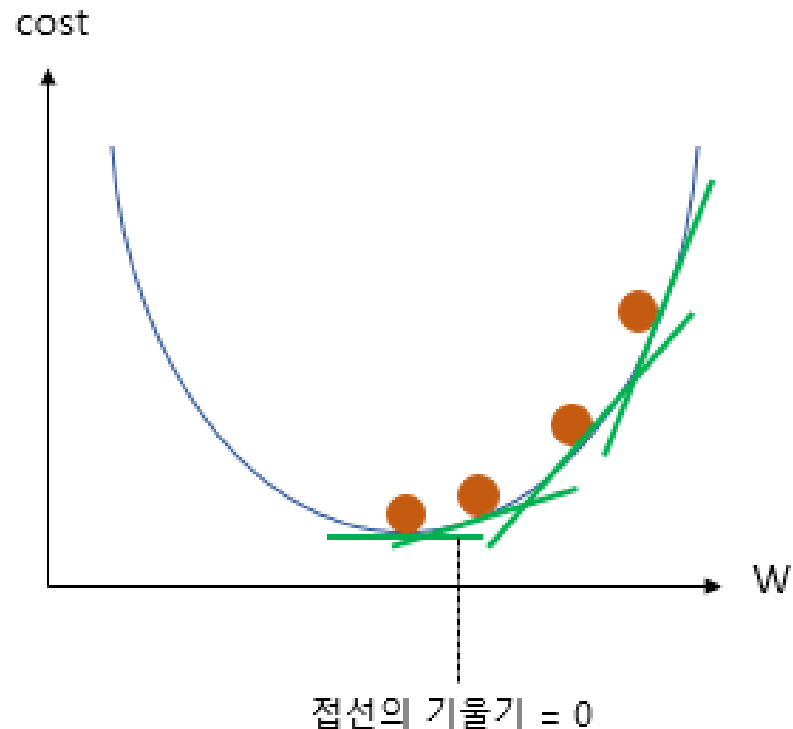


$$\frac{\partial}{\partial W} cost(W)$$

비용 함수를 W 에 대해 미분
= 접선에서의 기울기

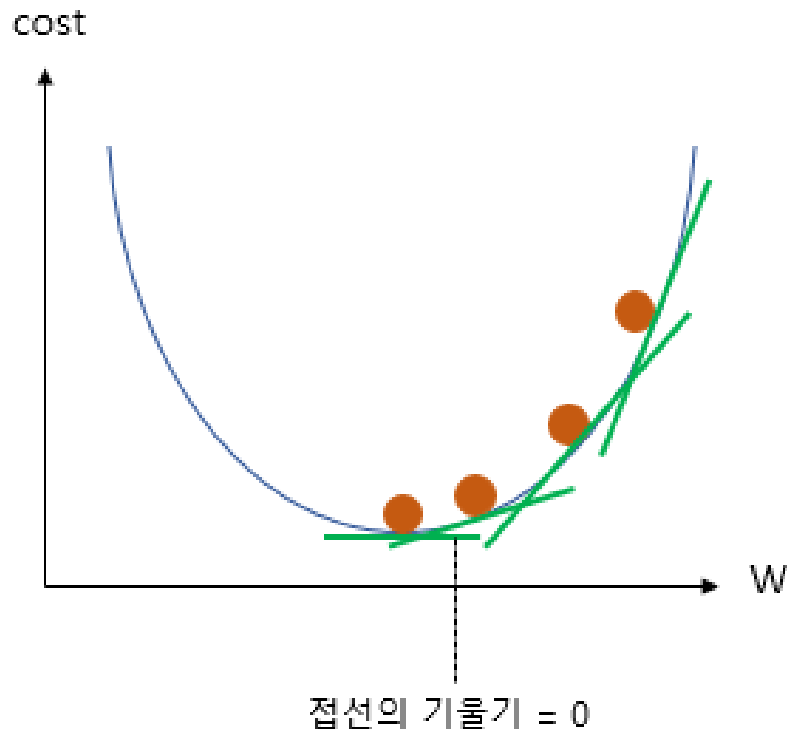
Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



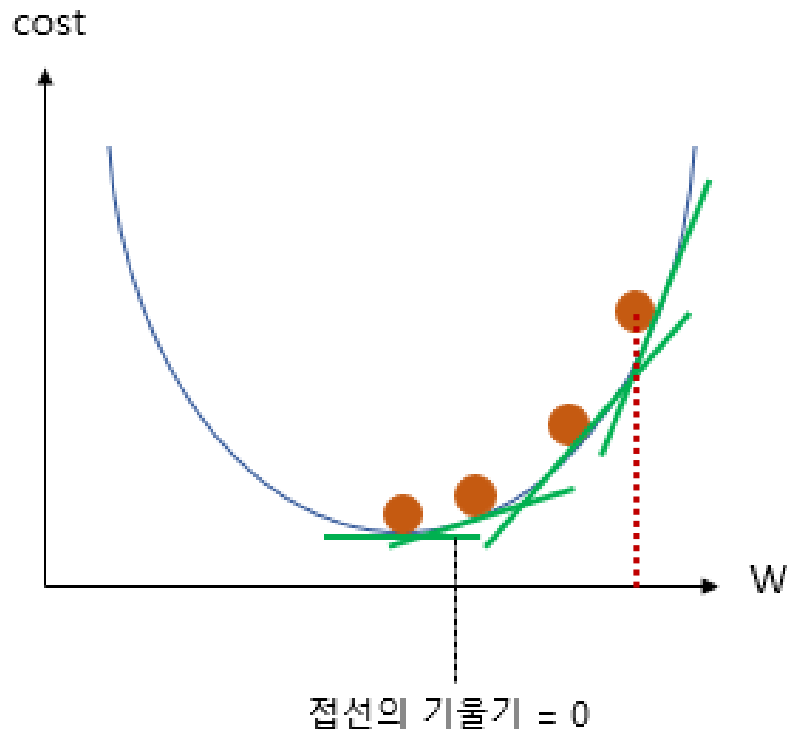
경사 하강법의 아이디어는 비용 함수(Cost function)를 미분하여 현재 w 에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 w 의 값을 변경하고 다시 미분하고 이 과정을 충분히 반복.

$$W := W - \frac{\partial}{\partial W} \text{cost}(W)$$

다음 W값 = 현재의 W값 - 접선에서의 기울기

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



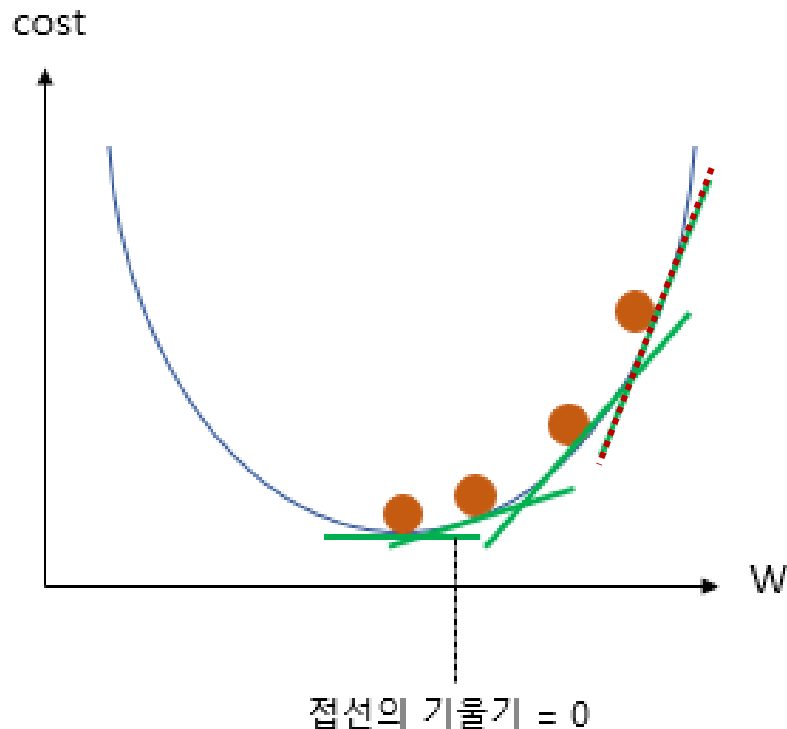
경사 하강법의 아이디어는 비용 함수(Cost function)를 미분하여 현재 w 에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 w 의 값을 변경하고 다시 미분하고 이 과정을 충분히 반복.

$$W := W - \frac{\partial}{\partial W} \text{cost}(W)$$

다음 W 값 = 현재의 W 값 - 접선에서의 기울기

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



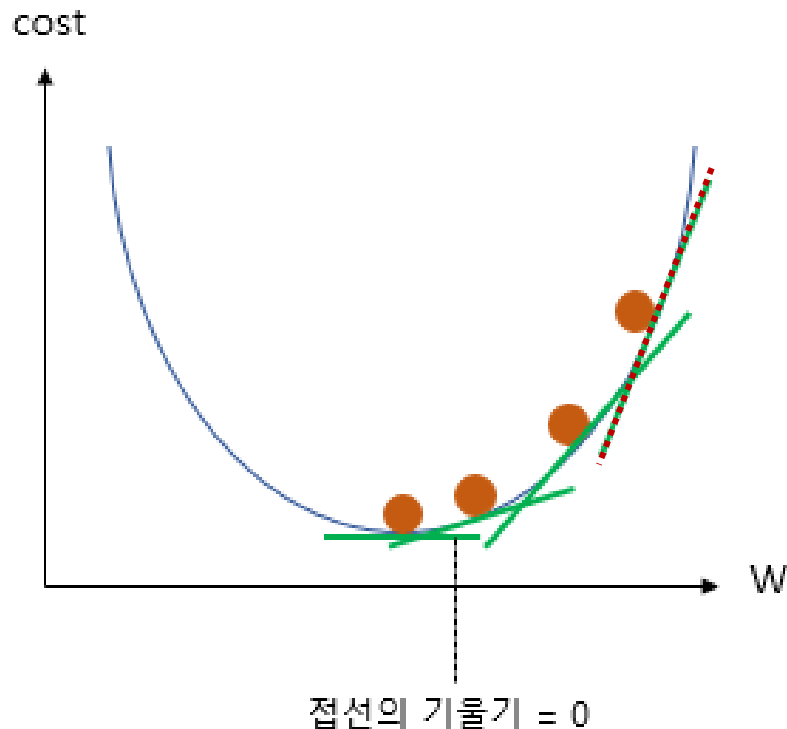
경사 하강법의 아이디어는 비용 함수(Cost function)를 미분하여 현재 w 에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 w 의 값을 변경하고 다시 미분하고 이 과정을 충분히 반복.

$$W := W - \frac{\partial}{\partial W} \text{cost}(W)$$

다음 W 값 = 현재의 W 값 - 접선에서의 기울기(양수)

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



경사 하강법의 아이디어는 비용 함수(Cost function)를 미분하여 현재 w 에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 w 의 값을 변경하고 다시 미분하고 이 과정을 충분히 반복.

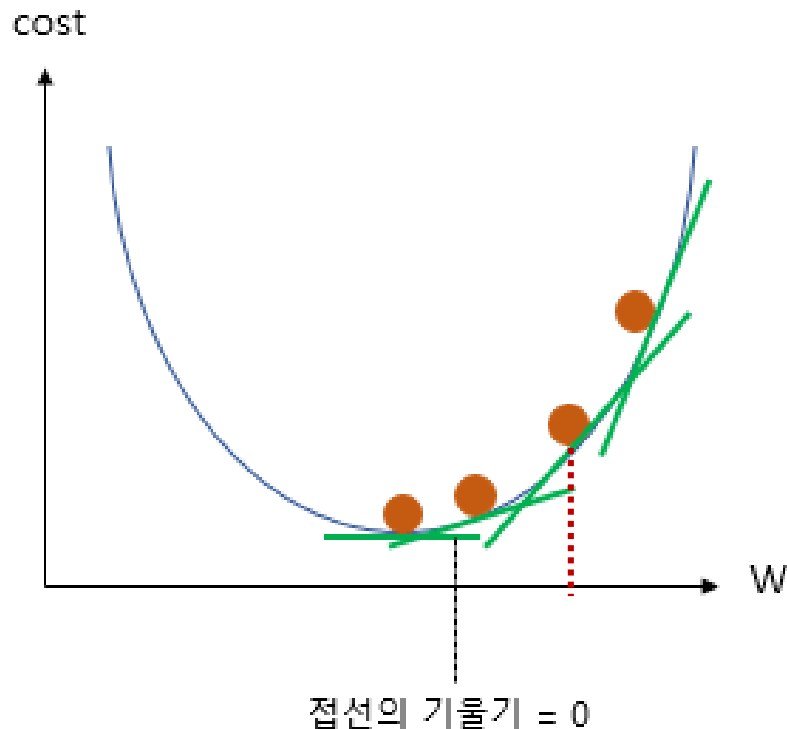
$$W := W - \frac{\partial}{\partial W} \text{cost}(W)$$

다음 W 값 = 현재의 W 값 - 접선에서의 기울기(양수)

양수를 뺌하므로 W 값은 작아진다.

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



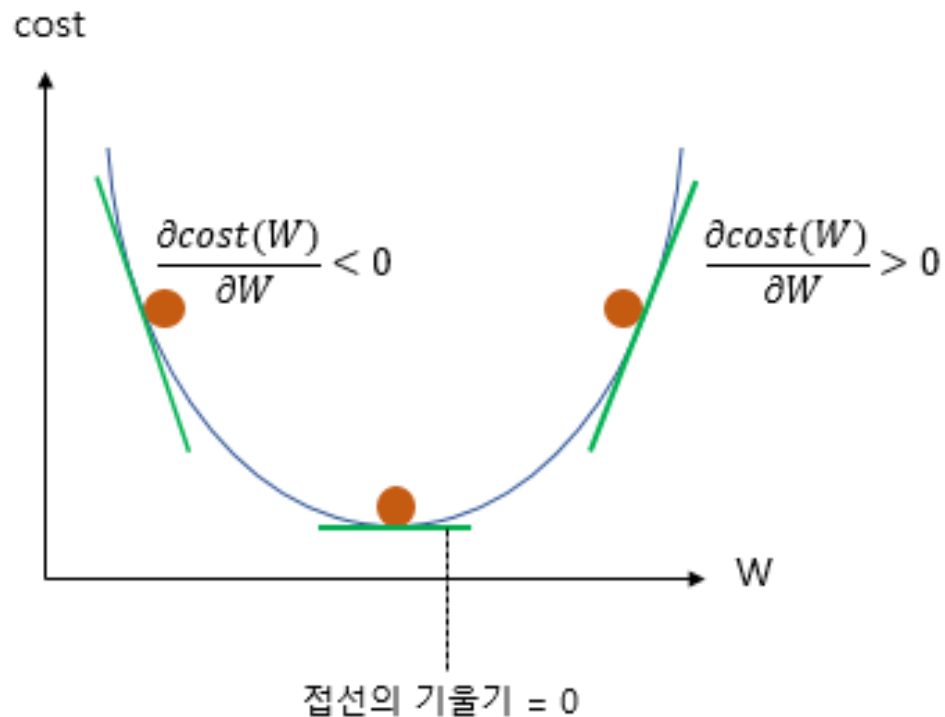
경사 하강법의 아이디어는 비용 함수(Cost function)를 미분하여 현재 w 에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 w 의 값을 변경하고 다시 미분하고 이 과정을 충분히 반복.

$$W := W - \frac{\partial}{\partial W} \text{cost}(W)$$

다음 W값 = 현재의 W값 - 접선에서의 기울기

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.

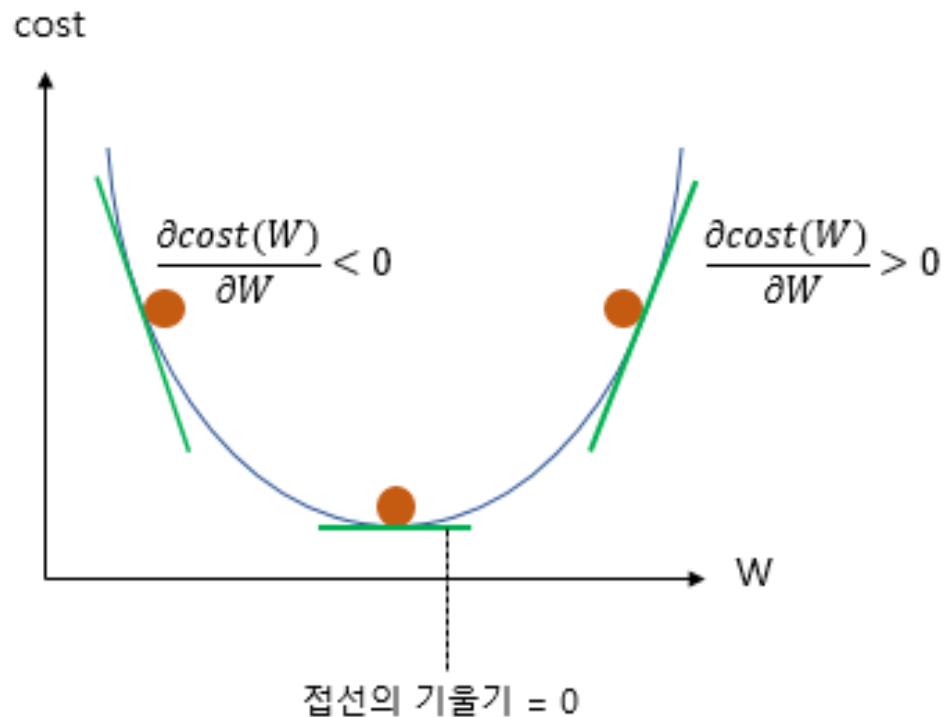


$$\text{cost}(W, b) = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$

$$W := W - \frac{\partial}{\partial W} \text{cost}(W)$$

Gradient Descent

- 경사 하강법은 접선의 기울기 개념을 사용한다.
- 경사 하강법은 최적의 값을 향해서 매개 변수를 반복 업데이트한다.



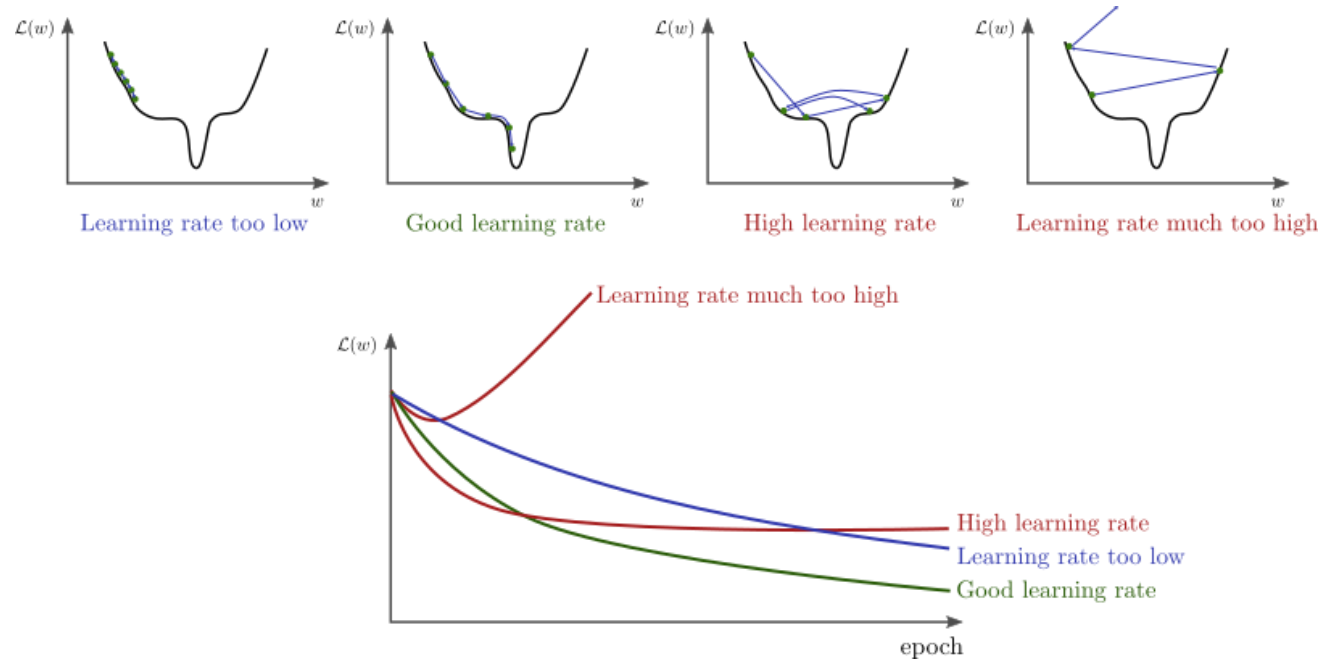
$$cost(W, b) = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Learning rate (0~1 사이값)

Gradient Descent

- Learning rate를 너무 높게 준다고해서 최적의 값을 빨리 찾지는 않는다.
- 지나친 Learning rate는 오히려 이를 방해한다.
- 만약 loss가 NaN이 나온다면, learning rate를 낮추는 걸 시도해볼 수 있다.



Summary : Machine Learning

- 머신 러닝 모델은 기본적으로 다음의 세 가지 과정을 거칩니다.
 - 1) 학습하고자 하는 가설(Hypothesis)을 세운다.
 - 2) 가설의 성능을 측정할 수 있는 손실 함수(Loss Function)를 정의한다.
 - 3) 손실 함수 L 을 최소화 할 수 있는 학습 알고리즘을 설계한다.

Summary : Linear Regression

- 앞서 세운 세 가지 과정을 선형 회귀에 적용해보면 다음과 같습니다.

1) 학습하고자 하는 가설(Hypothesis)을 세운다.

$$y = wx + b$$

이때 x 와 y 는 주어지는 입력 데이터와 타겟 데이터이고,
 w 와 b 는 파라미터라고 부르며 훈련 데이터로부터 학습을 통해 적절한 값을 찾아내야 한다.

Summary : Linear Regression

- 앞서 세운 세 가지 과정을 선형 회귀에 적용해보면 다음과 같습니다.

2) 가설의 성능을 측정할 수 있는 손실 함수(Loss Function)를 정의한다.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$

손실 함수는 여러가지 형태로 정의될 수 있는데, 그 중 가장 대표적인 손실 함수 중 하나는 평균 제곱 오차(Mean Squared Error, MSE)이다.

Summary : Linear Regression

- 앞서 세운 세 가지 과정을 선형 회귀에 적용해보면 다음과 같습니다.

2) 가설의 성능을 측정할 수 있는 손실 함수(Loss Function)를 정의한다.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$

손실 함수는 여러가지 형태로 정의될 수 있는데, 그 중 가장 대표적인 손실 함수 중 하나는 평균 제곱 오차(Mean Squared Error, MSE)이다.

손실 함수는 출처에 따라서 비용 함수(Cost Function)이라고 부르기도 한다.

Summary : Linear Regression

- 앞서 세운 세 가지 과정을 선형 회귀에 적용해보면 다음과 같습니다.

3) 손실 함수 L 을 최소화 할 수 있는 학습 알고리즘을 설계한다.

머신 러닝 모델은 처음에는 랜덤한 값으로 파라미터(w 와 b)를 초기화한 후에 파라미터를 적절한 값으로 계속 업데이트 한다. 이때 파라미터를 적절하게 업데이트 하는 방법을 최적화(optimization) 기법이라고 하며, 대표적인 것이 경사 하강법(Gradient Descent)이다.

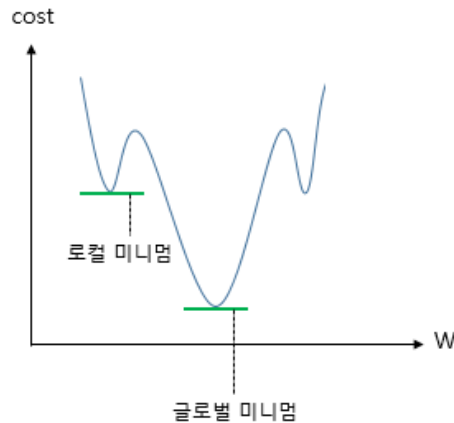
경사 하강법은 현재 스텝의 파라미터에서 손실 함수의 미분값에 러닝 레이트 α 를 곱한 만큼을 빼서 다음 스텝의 파라미터 값으로 지정한다. 따라서, 손실 함수의 미분값이 크거나 α 가 크면, 하나의 스텝에서 파라미터가 많이 업데이트 되고, 반대의 경우에는 적게 업데이트 된다.

Q & A

- **Learning Rate는 사람이 정해주는 값인지? 정해주는 규칙이나 방법이 있는지?**
 - 원론적으로 답변드리면 정답은 없습니다. 한 가지 팁을 드리자면 일반적으로는 조금 큰 값에 시작해서 1/2씩 줄이는 방법이 있습니다. 가령, 처음에는 0.001로 하였는데 발산을 한다거나 성능이 좋지 않은 경우 0.0005를 하고, 그 후에는 0.00025를 해볼 수 있겠습니다. 하지만 가장 좋은 방법은 하고자하는 Task에 대해서 높은 성능을 얻었던 모델에 대한 논문이 공개가 되었을 것이고, 해당 논문의 저자가 선택한 Learning Rate 값을 사용하는 것이 좋습니다

Q & A

- 신경망도 경사 하강법을 사용한다던데, 그러면 신경망도 Loss가 최소가 되는 지점에 도달할 때 학습을 멈추게 하는 것인지?
- 실제로는 학습 중 모델의 Loss가 현재가 정말 최소값인지 알기 어렵습니다. 방금 배운 선형 회귀의 경우 식이 간단해서 글로벌 미니멈을 찾기 쉬웠을지 모르지만, 실제 신경망의 경우 가설이 복잡해나가 파라미터가 굉장히 많은 관계로 로컬 미니멈이 존재하기 때문입니다. 그래서 사람이 충분한 횟수라고 판단되는 횟수만큼 학습 횟수를 지정하여, 해당 횟수를 학습하였을 때 모델이 그나마 Loss가 낮은 로컬 미니멈에 도달했기를 기대합니다.



Keras 이해하기

Keras를 임포트하는 방법.

Keras는 텐서플로우의 하이레벨 API입니다.

조금 더 쉽게 설명하면 텐서플로우 안에 Keras가 존재합니다.

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam
```

Keras를 임포트하는 방법.

Keras는 텐서플로우의 하이레벨 API입니다.

조금 더 쉽게 설명하면 텐서플로우 안에 Keras가 존재합니다.

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam
```

```
import tensorflow as tf
tf.keras.layers.Dense(10)
```

OR

```
from tensorflow.keras.layers import Dense
Dense(10)
```

Keras를 이용한 모델링의 이해

인공 신경망 모델을 설계하는 방법은 다음과 같습니다.

- 1. 전처리: 학습에 필요한 데이터 전처리를 수행합니다.
- 2. 모델링(model): 모델을 정의합니다.
- 3. 컴파일(compile): 모델을 생성합니다.
- 4. 학습 (fit): 모델을 학습시킵니다.

Keras를 이용한 모델링의 이해

```
# 필요한 패키지 import
import numpy as np
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

# 데이터 전처리
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([5.0, 6.0, 7.0, 8.0, 9.0, 10.0], dtype=float)

# 모델의 정의 (modeling)
model = Sequential()

# Dense의 첫번째 인자는 항상 출력의 차원을 의미한다.
model.add(Dense(1, input_dim=1, activation='linear'))

# 모델의 생성 (compile)
model.compile(optimizer='sgd', loss='mse')

# 학습 (fit)
model.fit(xs, ys, epochs=1200, verbose=0)

# 검증
# 16.000046
model.predict([10.0])

array([[16.000046]], dtype=float32)
```

Dense를 통해서 입, 출력의 차원을 기재.

'sgd'는 경사 하강법을 의미.

'mse'는 평균 제곱 오차를 의미.

'fit'은 데이터에 fitting되어 간다.라는 의미로 '학습'을 의미.

'epochs'는 전체 샘플에 대한 Weight와 Bias 업데이트 횟수.

Logistic Regression

Logistic Regression

두 개의 선택지 중에서 정답을 고르는 문제에 사용하는 모델.

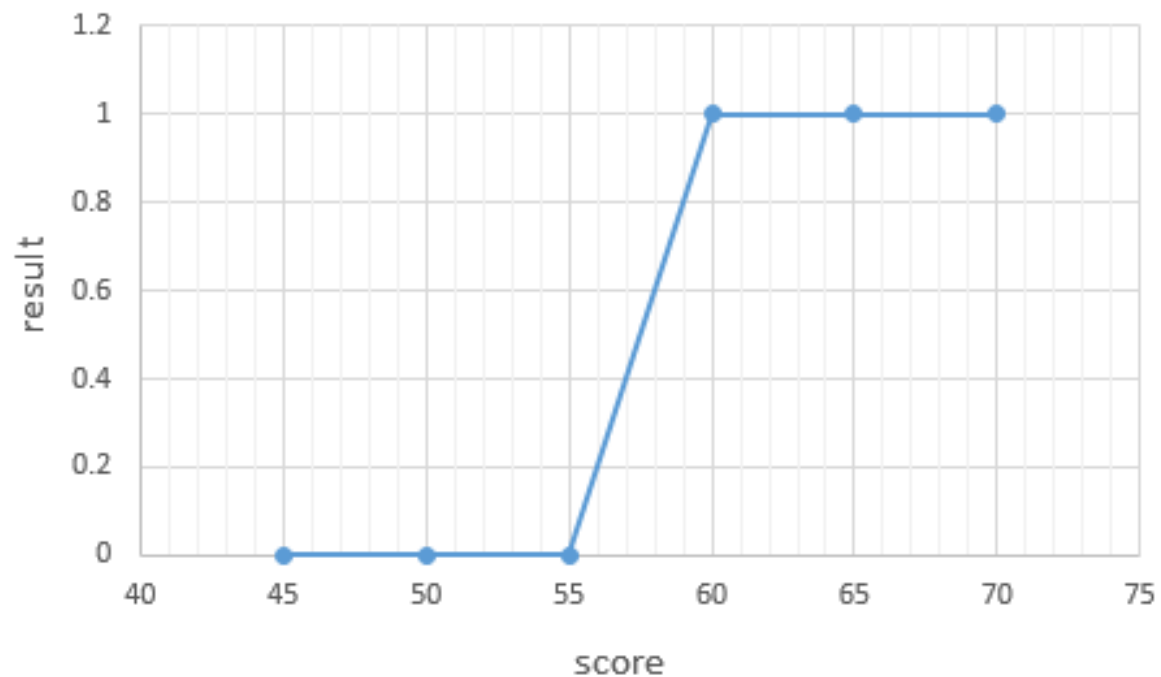
Logistic Regression

두 개의 선택지 중에서 정답을 고르는 문제에 사용하는 모델.
스팸 메일 분류 or 합격, 불합격을 분류.

Logistic Regression

두 개의 선택지 중에서 정답을 고르는 문제에 사용하는 모델.
스팸 메일 분류 or 합격, 불합격을 분류.
합격을 1, 불합격을 0이라 하고 그래프로 표현하면 다음과 같다.

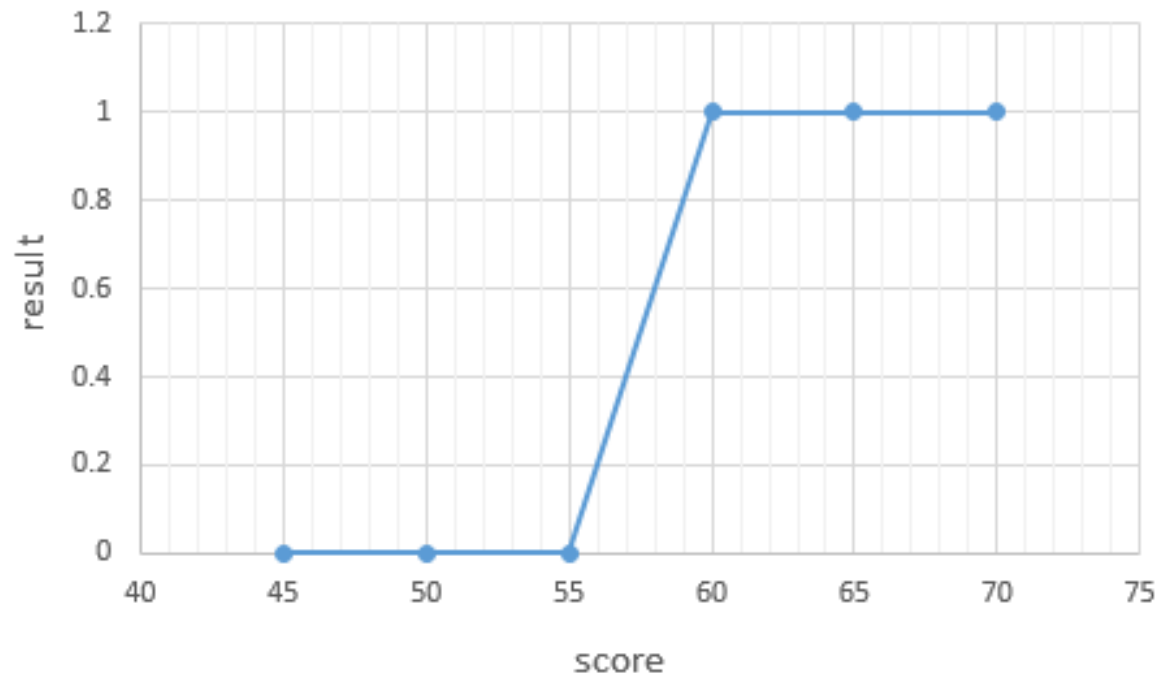
score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격



Logistic Regression

앞에서 배운대로 선을 그어서 이 점의 특성을 나타내는 직선을 그을 수는 없을까?

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격



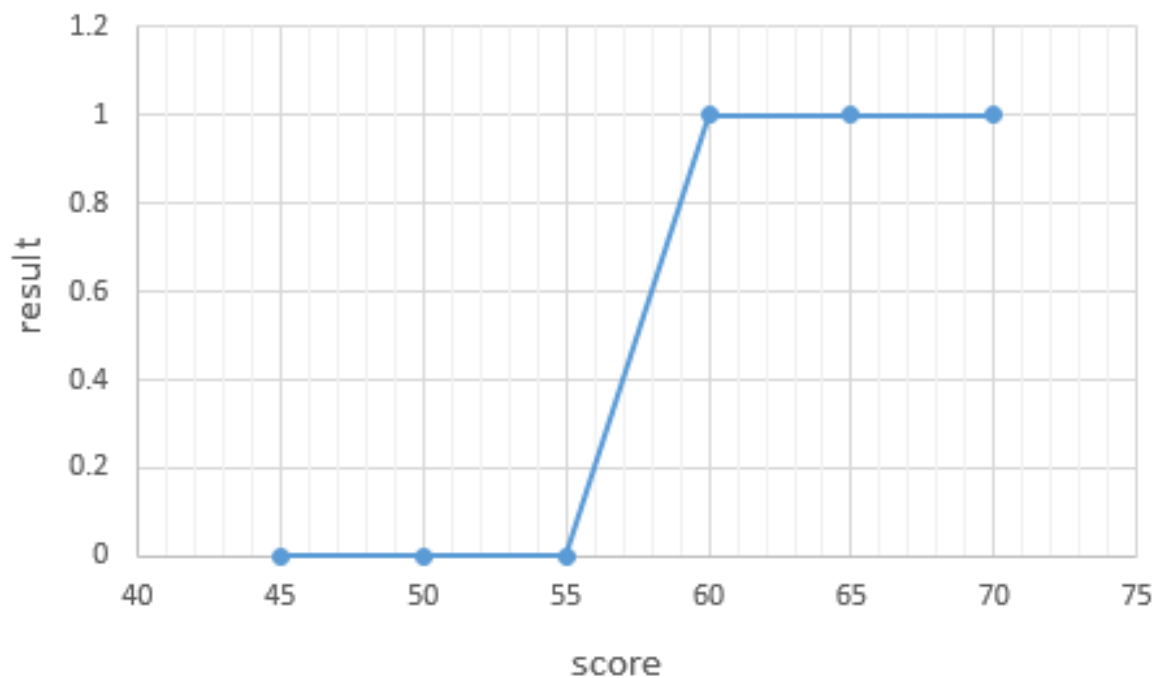
Logistic Regression

앞에서 배운대로 선을 그어서 이 점의 특성을 나타내는 직선을 그을 수는 없을까?

이 점들은 1과 0 사이의 값이 없으므로 직선으로 그리기 어렵다.

이를 반영한 S자 형태의 그래프가 필요하다.

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격



Logistic Regression : Sigmoid function

앞에서 배운대로 선을 그어서 이 점의 특성을 나타내는 직선을 그을 수는 없을까?

이 점들은 1과 0 사이의 값이 없으므로 직선으로 그리기 어렵다.

이를 반영한 S자 형태의 그래프가 필요하다.

$$\frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

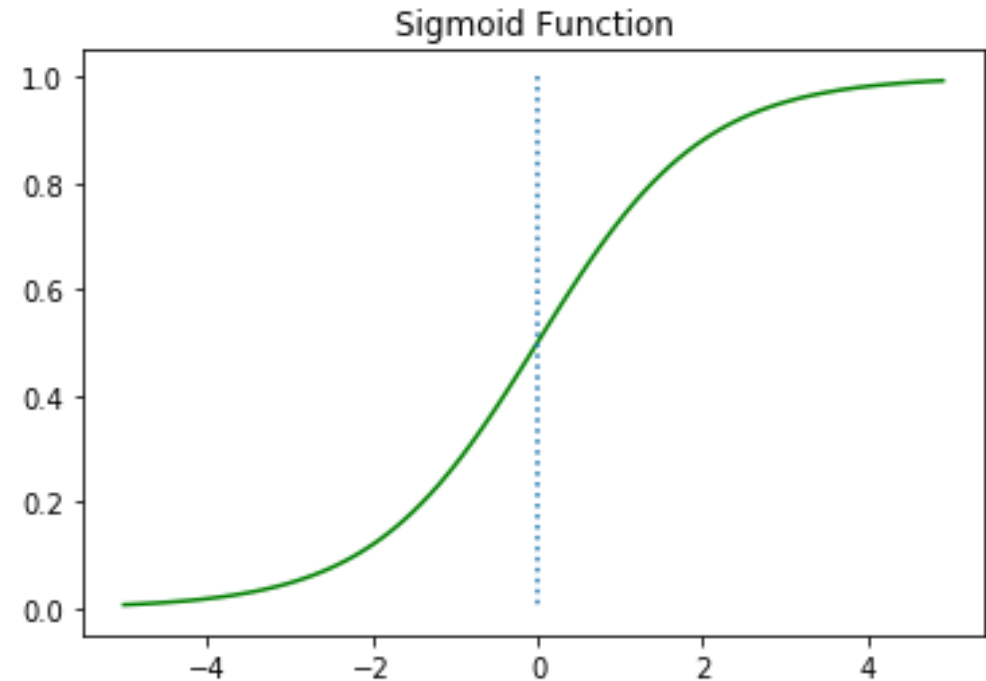
Logistic Regression : Sigmoid function

앞에서 배운대로 선을 그어서 이 점의 특성을 나타내는 직선을 그을 수는 없을까?

이 점들은 1과 0 사이의 값이 없으므로 직선으로 그리기 어렵다.

이를 반영한 S자 형태의 그래프가 필요하다.

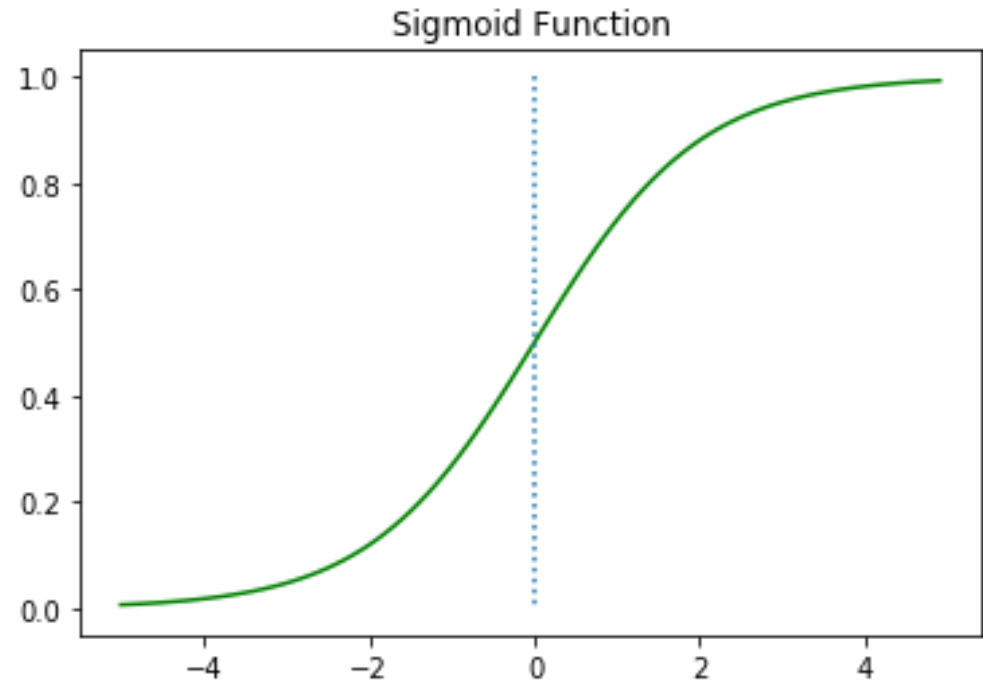
$$\frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$



Logistic Regression : Sigmoid function

시그모이드 함수는 입력값이 커지면 1에 수렴하고, 작으면 0에 수렴한다.
0부터의 1까지의 값을 가지는데 출력값이 0.5 이상이면 1(True),
0.5미만이면 0(False)로 만들면 이진 분류 문제로 사용할 수 있다.

$$\frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

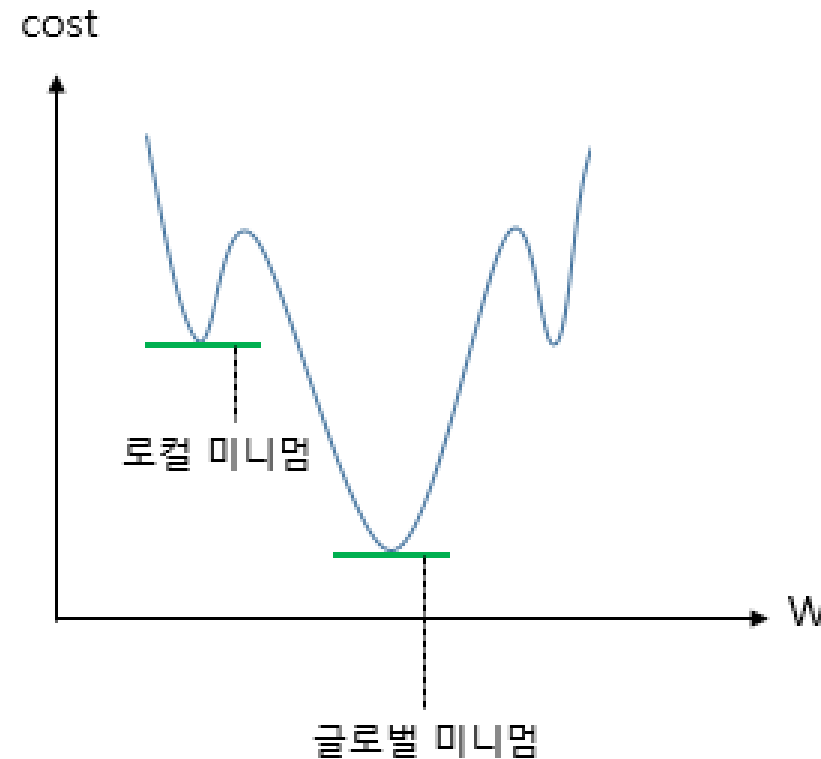


Logistic Regression : Cost function

로지스틱 회귀는 비용 함수로 평균 제곱 오차를 사용하지 않는다.
비용 함수로 평균 제곱 오차를 사용할 경우, 아래와 유사 그래프를 얻는다.

$$\text{cost}(W, b) = \frac{1}{n} \sum_{i=1}^n \left[y^{(i)} - H(x^{(i)}) \right]^2$$

$$\frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

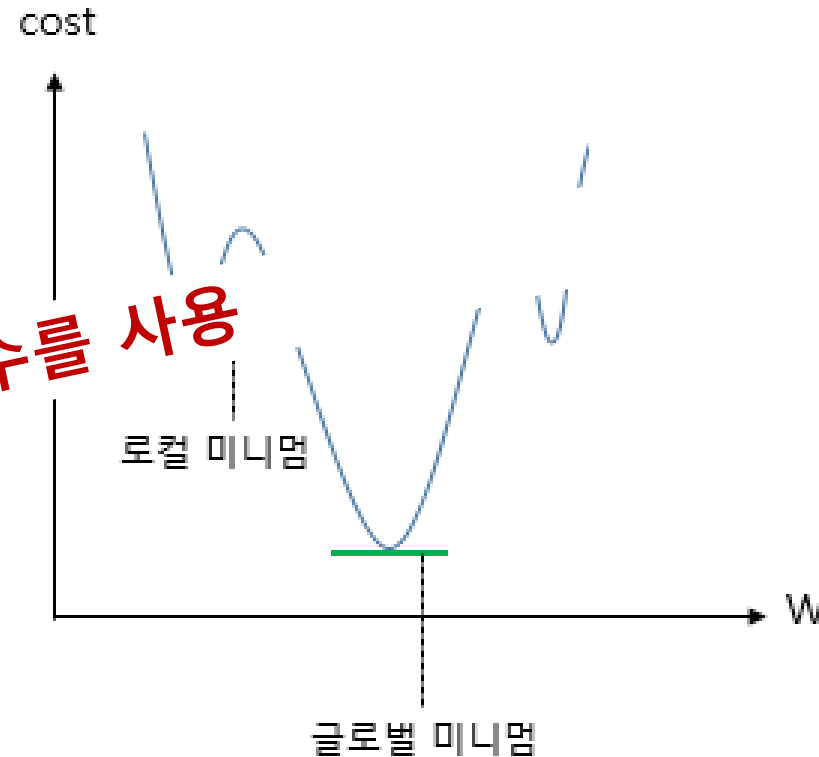


Logistic Regression : Cost function

로지스틱 회귀는 비용 함수로 평균 제곱 오차를 사용하지 않는다.
비용 함수로 평균 제곱 오차를 사용할 경우, 아래와 유사 그래프를 얻는다.

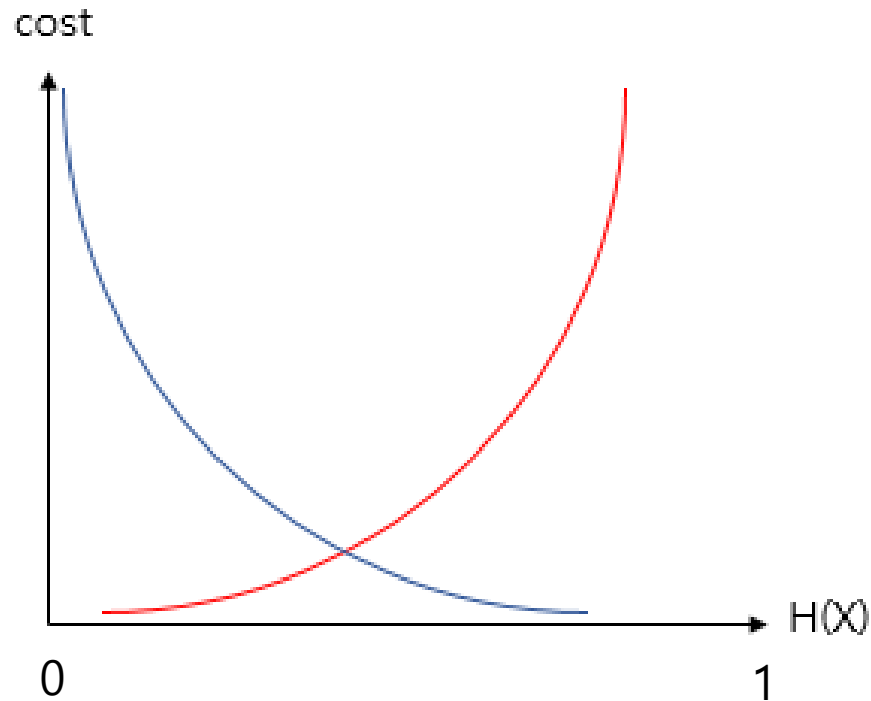
$$cost(W, b) = \frac{1}{n} \sum_{i=1}^n [y^{(i)} - H(x^{(i)})]^2$$
$$\frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) =$$

다른 비용 함수를 사용



Cross Entropy Function

로지스틱 회귀는 비용 함수로 **크로스 엔트로피 함수**를 사용한다.



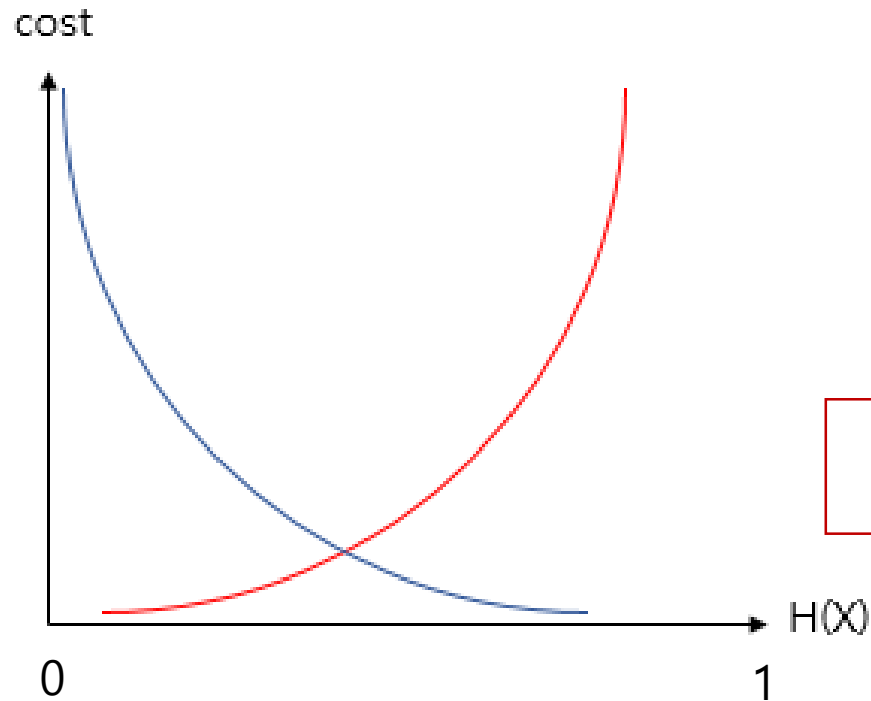
$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

if $y = 1 \rightarrow \text{cost}(H(x), y) = -\log(H(x))$ 파란선 그래프

if $y = 0 \rightarrow \text{cost}(H(x), y) = -\log(1 - H(x))$ 빨간선 그래프

Cross Entropy Function

로지스틱 회귀는 비용 함수로 **크로스 엔트로피 함수**를 사용한다.



$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

if $y = 1 \rightarrow \text{cost}(H(x), y) = -\log(H(x))$ 파란선 그래프

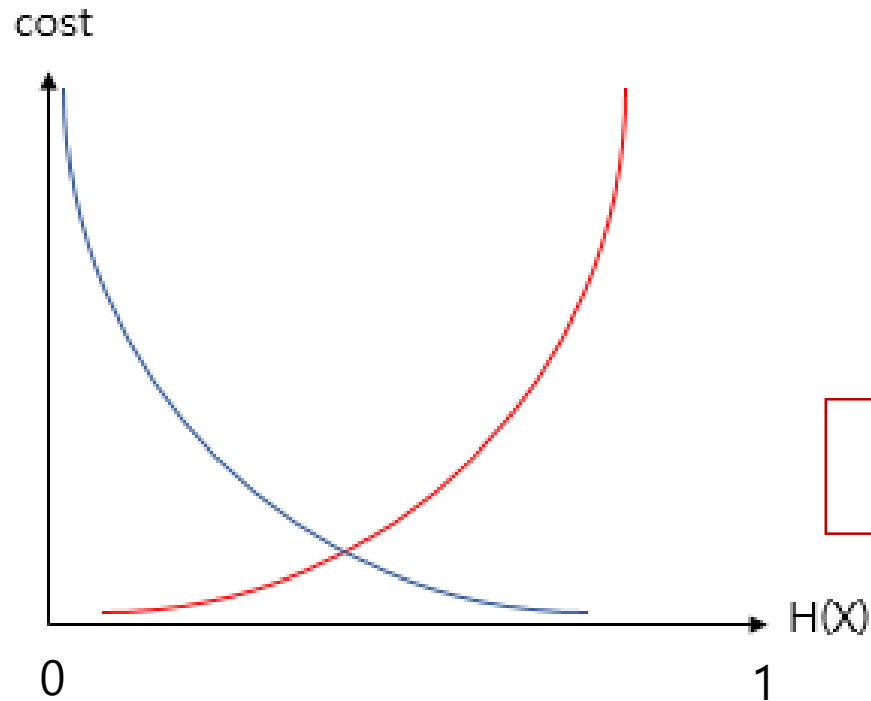
if $y = 0 \rightarrow \text{cost}(H(x), y) = -\log(1 - H(x))$ 빨간선 그래프

$$\text{cost}(H(x), y) = -[y \log H(x) + (1 - y) \log(1 - H(x))]$$

y 가 0이면 $y \log H(x)$ 가 없어지고, y 가 1이면 $(1-y) \log(1-H(x))$ 가 없어지는데 이는 각각 y 가 1일 때와 y 가 0일 때의 앞서 본 식과 동일.

Cross Entropy Function

로지스틱 회귀는 비용 함수로 **크로스 엔트로피 함수**를 사용한다.



$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

$$\text{if } y = 1 \rightarrow \text{cost}(H(x), y) = -\log(H(x))$$

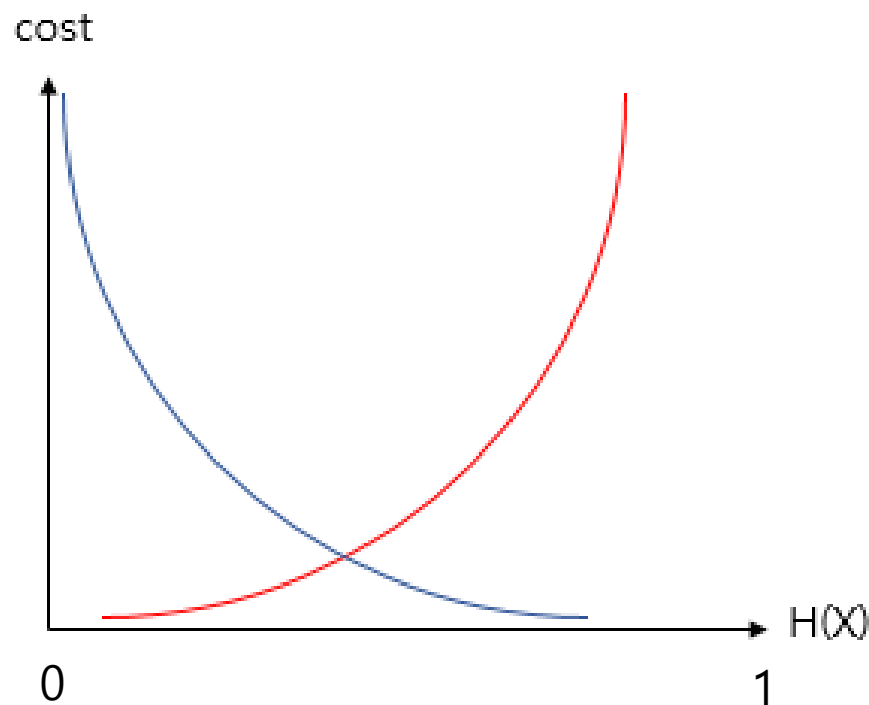
$$\text{if } y = 0 \rightarrow \text{cost}(H(x), y) = -\log(1 - H(x))$$

$$\text{cost}(H(x), y) = -[y \log H(x) + (1 - y) \log(1 - H(x))]$$

마찬가지로 모든 데이터에 대해서 평균을 내야하므로

Cross Entropy Function

로지스틱 회귀는 비용 함수로 크로스 엔트로피 함수를 사용한다.



$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = \text{sigmoid}(Wx + b) = \sigma(Wx + b)$$

$$\text{if } y = 1 \rightarrow \text{cost}(H(x), y) = -\log(H(x))$$

$$\text{if } y = 0 \rightarrow \text{cost}(H(x), y) = -\log(1 - H(x))$$

$$\text{cost}(H(x), y) = -[y \log H(x) + (1 - y) \log(1 - H(x))]$$

$$\begin{aligned} \text{cost}(H(x), y) = & \\ & -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log H(x^{(i)}) + (1 - y^{(i)}) \log(1 - H(x^{(i)}))] \end{aligned}$$

Linear Regression Vs. Logistic Regression

- 선형 회귀와 로지스틱 회귀의 비교
 - 선형 회귀의 가설은 $y = wx + b$
 - 로지스틱 회귀의 가설은 $y = \text{sigmoid}(wx + b)$
 - 선형 회귀의 손실 함수는 Mean Squared Error
 - 로지스틱 회귀의 손실 함수는 Cross-Entropy
 - 선형 회귀는 회귀 문제에 푸는 알고리즘으로 예측값이 연속적인 값이다.
 - 로지스틱 회귀는 이진 분류 문제에 푸는 알고리즘으로 예측값은 1 또는 0이다.

Linear Regression Vs. Logistic Regression

- 선형 회귀와 로지스틱 회귀의 비교
 - 선형 회귀의 가설은 $y = wx + b$
 - 로지스틱 회귀의 가설은 $y = \text{sigmoid}(wx + b)$
 - 선형 회귀의 손실 함수는 Mean Squared Error
 - 로지스틱 회귀의 손실 함수는 Cross-Entropy
 - 선형 회귀는 회귀 문제에 푸는 알고리즘으로 예측값이 연속적인 값이다.
 - 로지스틱 회귀는 이진 분류 문제에 푸는 알고리즘으로 예측값은 1 또는 0이다.

저 두 가지는 이해도 중요하지만, 사실 그냥 암기한듯이 무조건 알고 있어야함.

Keras를 이용한 모델링의 이해 (선형 회귀)

```
# 필요한 패키지 import
import numpy as np
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

# 데이터 전처리
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([5.0, 6.0, 7.0, 8.0, 9.0, 10.0], dtype=float)

# 모델의 정의 (modeling)
model = Sequential()

# Dense의 첫번째 인자는 항상 출력의 차원을 의미한다.
model.add(Dense(1, input_dim=1, activation='linear'))

# 모델의 생성 (compile)
model.compile(optimizer='sgd', loss='mse')

# 학습 (fit)
model.fit(xs, ys, epochs=1200, verbose=0)

# 검증
# 16.000046
model.predict([10.0])

array([[16.000046]], dtype=float32)
```

Dense를 통해서 입, 출력의 차원을 기재.

'sgd'는 경사 하강법을 의미.

'mse'는 평균 제곱 오차를 의미.

'fit'은 데이터에 fitting되어 간다.라는 의미로 '학습'을 의미.

'epochs'는 전체 샘플에 대한 Weight와 Bias 업데이트 횟수.

Keras를 이용한 모델링의 이해 (로지스틱 회귀)

```
import numpy as np
from tensorflow.keras.models import Sequential # 케라스의 Sequential()을 임포트
from tensorflow.keras.layers import Dense # 케라스의 Dense()를 임포트
from tensorflow.keras import optimizers # 케라스의 옵티마이저를 임포트

# X의 입력이 10부터 y의 출력이 1이 되도록 설계된 데이터
X = np.array([-50, -40, -30, -20, -10, -5, 0, 5, 10, 20, 30, 40, 50])
y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])

model = Sequential()

# Dense의 첫번째 인자는 항상 출력의 차원을 의미.
# activation function은 출력값에 적용할 함수를 의미.
model.add(Dense(1, input_dim=1, activation='sigmoid'))

# sgs는 경사하강법을 의미. lr은 learning rate의 값.
sgd = optimizers.SGD(lr=0.01)

# 이진 분류를 위해 activation function으로 sigmoid 함수를 사용한다면
# loss는 binary_crossentropy를 사용. (암기하듯이 기억해도 무방함.)
model.compile(optimizer=sgd, loss='binary_crossentropy', metrics=['binary_accuracy'])

model.fit(X,y, batch_size=1, epochs=200, shuffle=False)
```

activation function은 출력에 적용할 함수를 의미.

Dense를 통해서 입, 출력의 차원을 기재.

'sgd'는 경사 하강법을 의미.

'mse'는 평균 제곱 오차를 의미.

'fit'은 데이터에 fitting되어 간다.라는 의미로 '학습'을 의미.

'epochs'는 전체 샘플에 대한 Weight와 Bias 업데이트 횟수.

이진 분류에서

activation='sigmoid'

loss는 'binary_crossentropy'

다중 선형 회귀와 다중 로지스틱 회귀

만약, 입력이 스칼라가 아니라 3차원 벡터라면?

Ex) 중간, 기말, 가산점 점수로부터 최종 성적 예측하기

```
import numpy as np
from tensorflow.keras.models import Sequential # 케라스의 Sequential()을 임포트
from tensorflow.keras.layers import Dense # 케라스의 Dense()를 임포트
from tensorflow.keras import optimizers # 케라스의 옵티마이저를 임포트
```

```
# 입력 벡터의 차원은 3입니다. 즉, input_dim은 3입니다.
X = np.array([[70,85,11],[71,89,18],[50,80,20],[99,20,10],[50,10,10]]) # 중간, 기말, 가산점
# 출력 벡터의 차원은 1입니다. 즉, output_dim은 1입니다.
y = np.array([73,82,72,57,34]) # 최종 성적

model = Sequential()
model.add(Dense(1, input_dim=3, activation='linear'))
sgd = optimizers.SGD(lr=0.00001)

model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
model.fit(X,y, batch_size=1, epochs=500, shuffle=False)
```

만약, 입력이 스칼라가 아니라 2차원 벡터라면?

```
import numpy as np
from tensorflow.keras.models import Sequential # 케라스의 Sequential()을 임포트
from tensorflow.keras.layers import Dense # 케라스의 Dense()를 임포트
from tensorflow.keras import optimizers # 케라스의 옵티마이저를 임포트
```

```
# 입력 벡터의 차원은 2입니다. 즉, input_dim은 2입니다.
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
# 출력 벡터의 차원은 1입니다. 즉, output_dim은 1입니다.
y = np.array([0, 1, 1, 1])

model=Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binary_accuracy'])
model.fit(X,y, batch_size=1, epochs=800, shuffle=False)
```


이진 분류와 다중 클래스 분류

두 개의 선택지 중에서 정답을 고르는 문제를
이진 분류 문제(Binary Classification)라고 한다.

세 개 이상의 선택지 중에서 정답을 고르는 문제에 사용하는 모델.
세 개 이상의 선택지 중에 정답을 고르는 문제를
다중 클래스 분류 문제(MultiClass Classification)라고 한다.

Softmax Regression

Softmax Regression

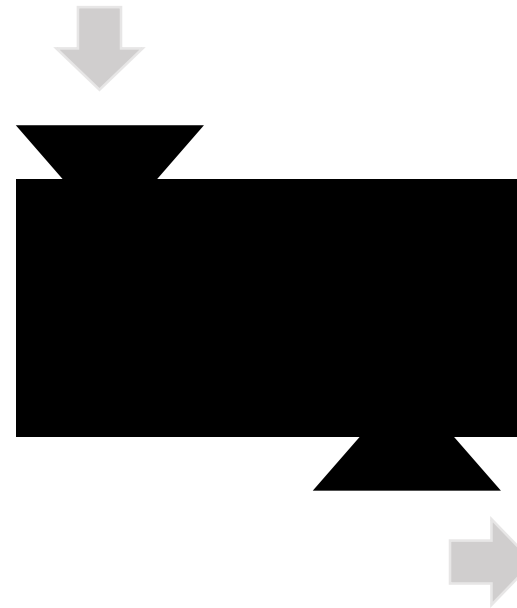
꽃받침 길이, 꽃받침 넓이, 꽃잎 길이, 꽃잎 넓이로부터
setosa, versicolor, virginica라는 3개의 품종 중 어떤 품종인지를 예측하는 문제

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)	Species(y)
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
5.8	2.6	4.0	1.2	versicolor
6.7	3.0	5.2	2.3	virginica
5.6	2.8	4.9	2.0	virginica

Softmax Regression

꽃받침 길이, 꽃받침 넓이, 꽃잎 길이, 꽃잎 넓이로부터
setosa, versicolor, virginica라는 3개의 품종 중 어떤 품종인지를 예측하는 문제

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)
5.1	3.5	1.4	0.2



**Softmax Regression은
각 클래스일 확률을 반환한다.**

0.25	0.1	0.65
------	-----	------

Softmax Regression

소프트맥스 회귀는 k 차원의 벡터에서 i 번째 원소를 z_i
 i 번째 클래스가 정답일 확률을 p_i 라고 하였을 때, 소프트맥스 함수는
 p_i 를 다음과 같이 정의한다.

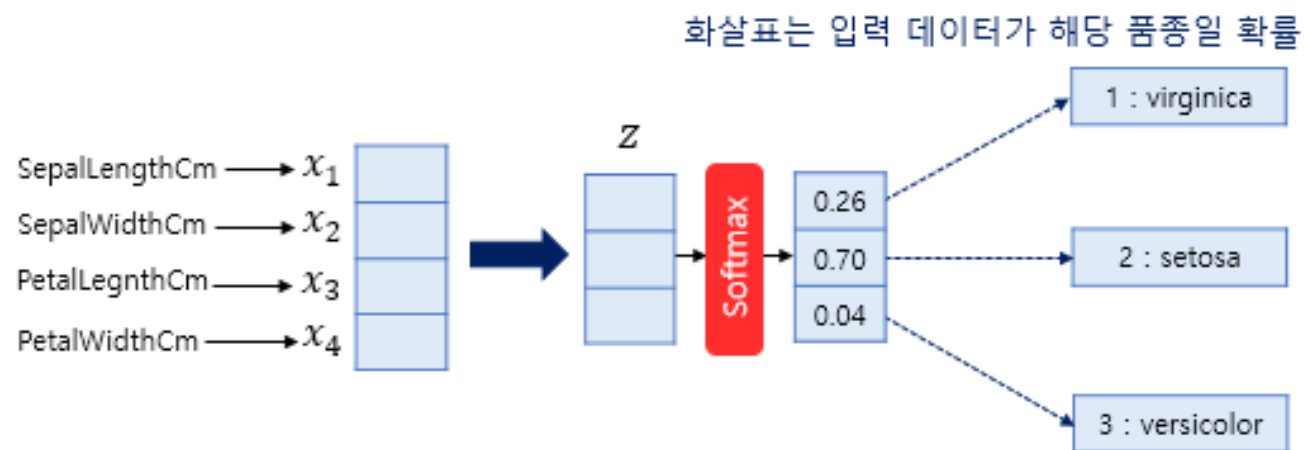
$$p_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \text{ for } i = 1, 2, \dots, k$$

이 경우 $k=3$ 이므로 $z = [z_1 \ z_2 \ z_3]$

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{j=1}^3 e^{z_j}} \quad \frac{e^{z_2}}{\sum_{j=1}^3 e^{z_j}} \quad \frac{e^{z_3}}{\sum_{j=1}^3 e^{z_j}} \right] = [p_1, p_2, p_3] = \hat{y} = \text{예측값}$$

Softmax Regression

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)
5.1	3.5	1.4	0.2

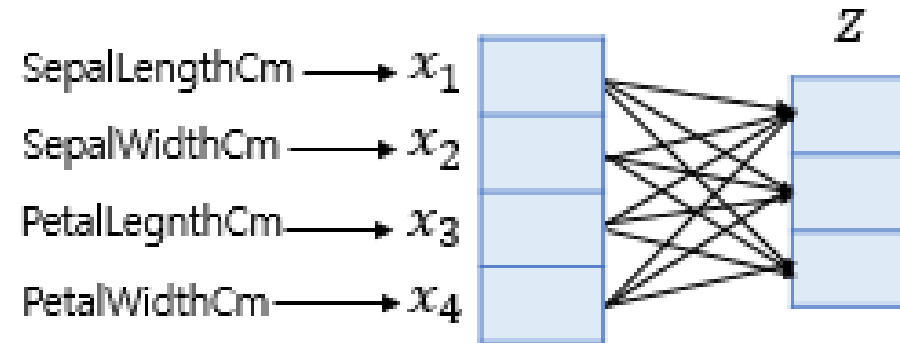


소프트맥스 함수의 입력으로 어떻게 바꿀까?

오차를 어떻게 구할까?

Softmax Regression

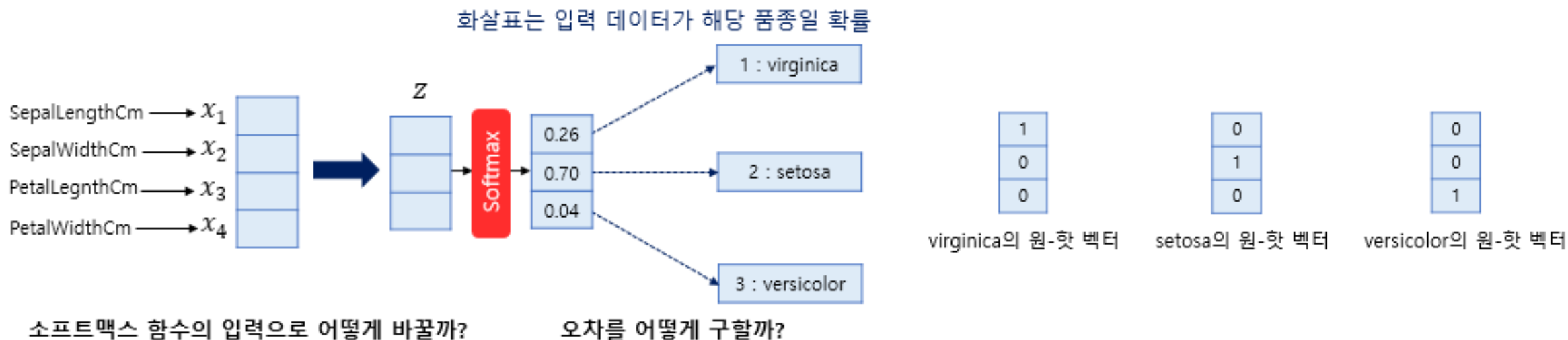
SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)
5.1	3.5	1.4	0.2



입력의 4차원, 변경하고자 하는 벡터는 3차원이므로 가중치는 총 12개를 가진다.

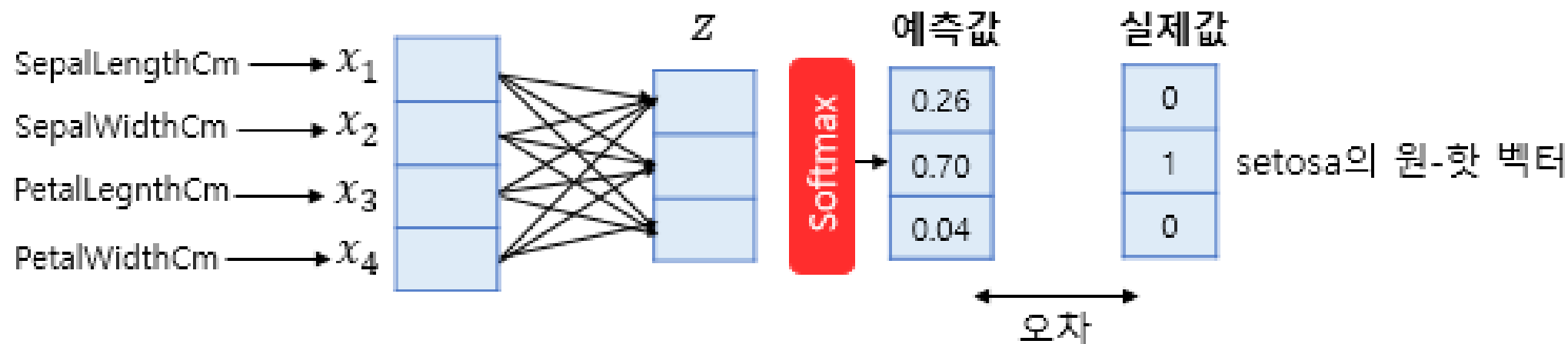
Softmax Regression

0번을 virginica, 1번을 setosa, 2번을 versicolor라고 하고,
이를 원-핫 인코딩하여 실제값으로 사용한다.



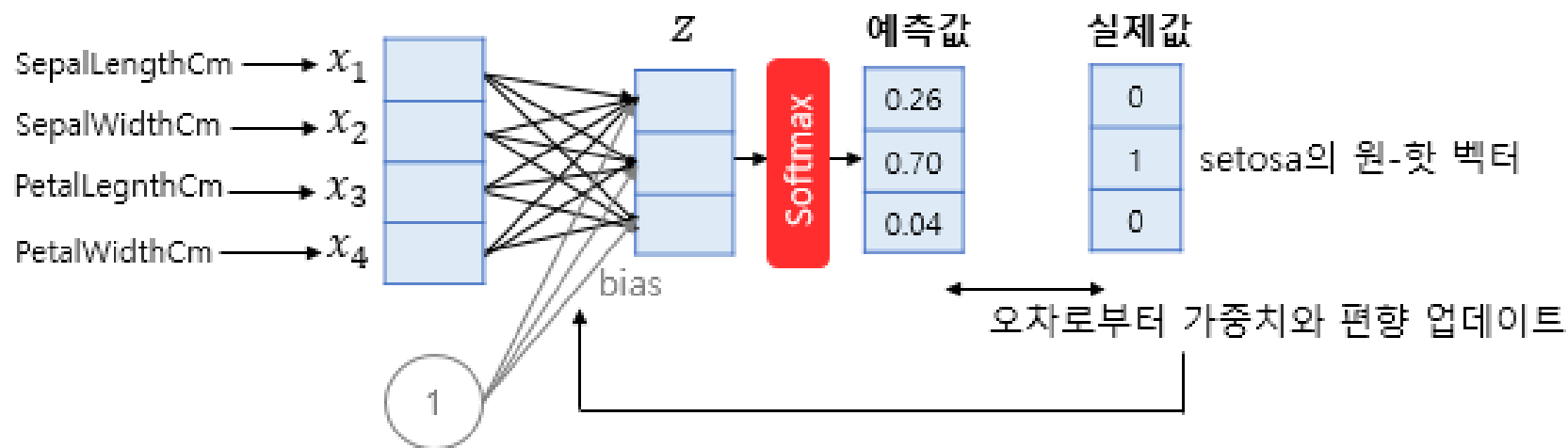
Softmax Regression

0번을 virginica, 1번을 setosa, 2번을 versicolor라고 하고,
이를 원-핫 인코딩하여 실제값으로 사용한다.



Softmax Regression

선형 회귀나 로지스틱 회귀와 마찬가지로
편향 또한 업데이트의 대상이 되는 매개 변수이다.



Softmax Regression

소프트맥스 회귀의 forward 연산을 다음과 같이 벡터와 행렬 연산으로 이해할 수 있다.

$$\text{softmax} \left(\begin{matrix} W \\ 3 \times 4 \end{matrix} \times \begin{matrix} x \\ 4 \times 1 \end{matrix} + \begin{matrix} b \\ 3 \times 1 \end{matrix} \right) = \begin{matrix} \text{예측값} \\ 3 \times 1 \end{matrix}$$

Cross Entropy Function

소프트맥스 회귀의 비용 함수 또한 크로스 엔트로피 함수이다.

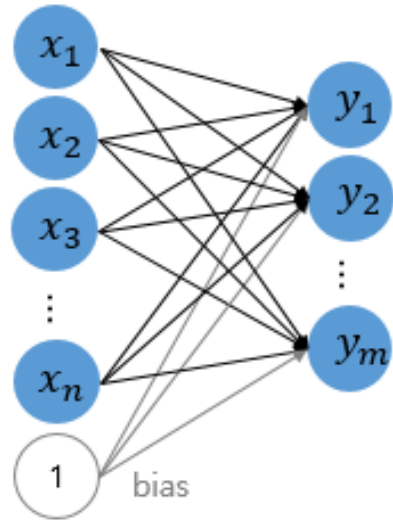
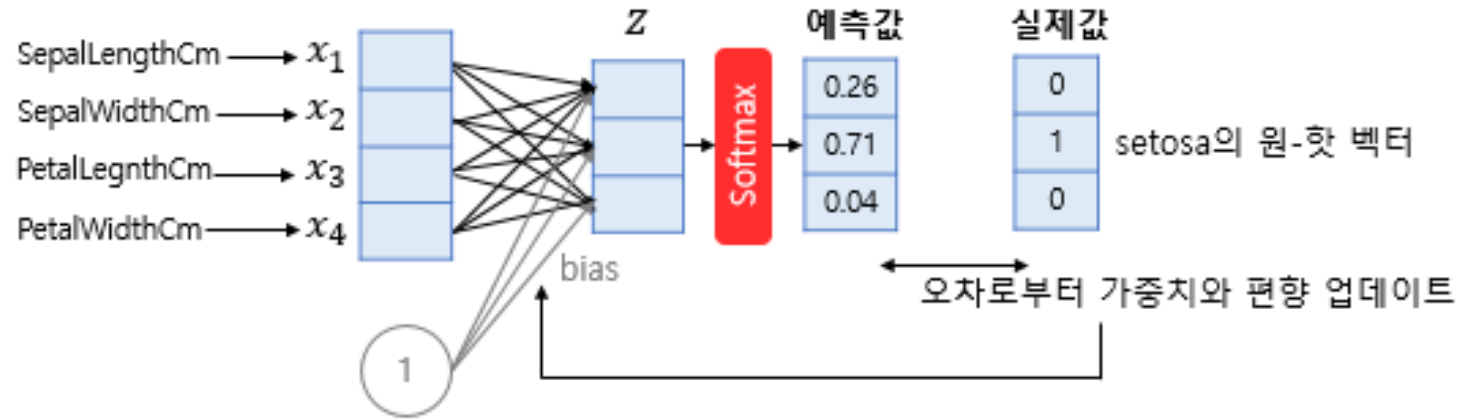
$$cost(W) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k y_j^{(i)} \log(p_j^{(i)})$$

사실 로지스틱 회귀의 크로스 엔트로피 함수는 $k=2$ 인 경우이다.

$$cost(W) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k y_j^{(i)} \log(p_j^{(i)}) = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(p^{(i)}) + (1 - y^{(i)}) \log(1 - p^{(i)})]$$

식 도출 : <https://wikidocs.net/35476>

Artificial Neural Network



이 둘은 사실 동일한 그림.

Logistic Regression Vs. Softmax Regression

- 로지스틱 회귀와 소프트맥스 회귀의 비교
 - 로지스틱 회귀는 이진 분류 문제에 사용한다.
 - 소프트맥스 회귀는 다중 클래스 분류 문제에 사용한다.
 - 로지스틱 회귀는 가설에 sigmoid 함수를 사용한다.
 - 소프트맥스 회귀는 가설에 softmax 함수를 사용한다.
 - 소프트맥스 회귀는 각 카테고리를 원-핫 인코딩하여야 한다.
 - 두 함수 모두 손실 함수는 크로스 엔트로피 함수를 사용한다.

Keras를 이용한 모델링의 이해 (소프트맥스 회귀)

```
from tensorflow.keras.models import Sequential # 케라스의 Sequential()을 импорт
from tensorflow.keras.layers import Dense # 케라스의 Dense()를 импорт
from tensorflow.keras import optimizers # 케라스의 옵티마이저를 импорт

model = Sequential()

# 입력의 차원은 4, 출력의 차원은 3, activation function은 softmax
model.add(Dense(3, input_dim=4, activation='softmax'))

# 학습률(learning rate, lr)은 0.01로 합니다.
sgd = optimizers.SGD(lr=0.01)

# 옵티마이저는 경사하강법의 일종인 adam을 사용합니다.
# 손실 함수(Loss function)는 크로스 엔트로피 함수를 사용합니다.
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 200번 시도합니다.
history = model.fit(x_data, y_one_hot, batch_size=1, epochs=200)
```

activation function은 출력에 적용할 함수를 의미.

Dense를 통해서 입, 출력의 차원을 기재.

이번에는 입력은 4차원 벡터, 출력은 3차원 벡터.

'adam'은 sgd의 변형.

다중 클래스 분류에서

activation='softmax'

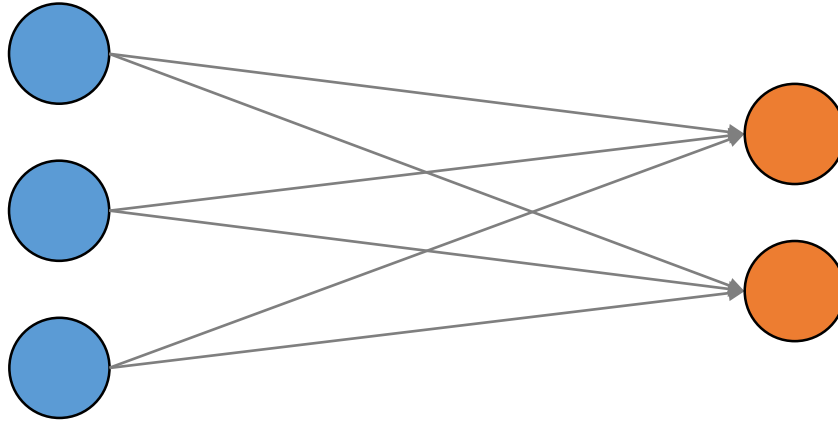
loss는 'categorical_crossentropy'

그리고 loss를 categorical_crossentropy를 사용할 경우 y에 해당하는 레이블은 반드시 원-핫 벡터여야 한다.

신경망 이해하기

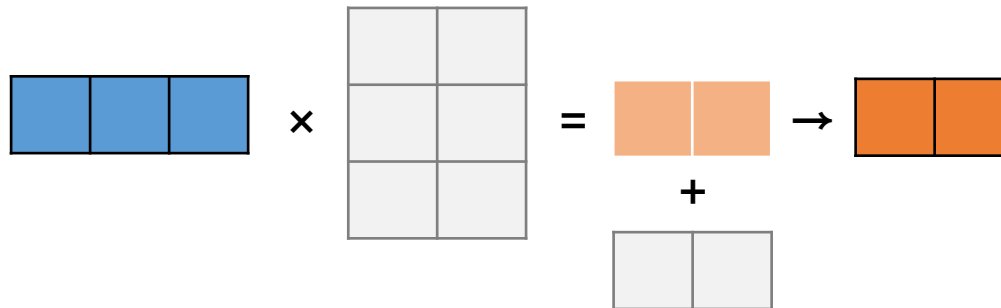
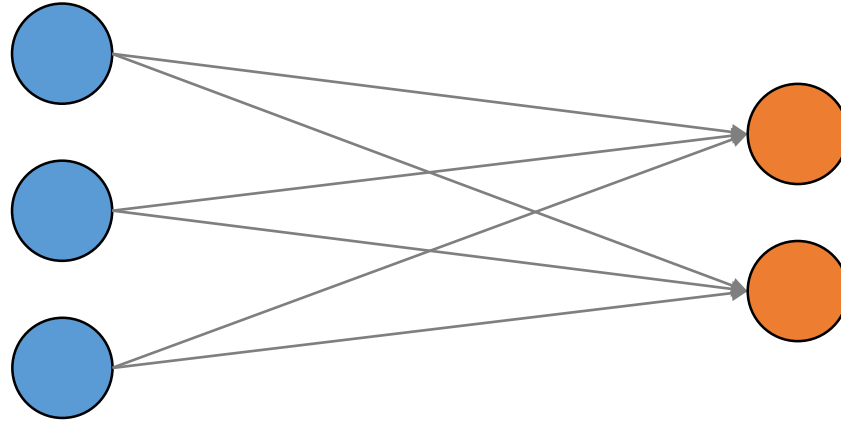
Neural Network 이해하기

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



Neural Network 이해하기

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



Logistic Regression

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀져 있는 데이터가 있을 때, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측하고자 한다.

SepalLengthCm(x_1)	PetalLengthCm(x_2)	Species(y)
5.1	3.5	A
4.7	3.2	A
5.2	1.8	B
7	4.1	A
5.1	2.1	B

Logistic Regression

꽃받침(Sepal)의 길이와 꽃잎(Petal)의 길이와 해당 꽃이 A인지 B인지가 적혀져 있는 데이터가 있을 때, 새로 조사한 꽃받침의 길이와 꽃잎의 길이로부터 무슨 꽃인지 예측하고자 한다.

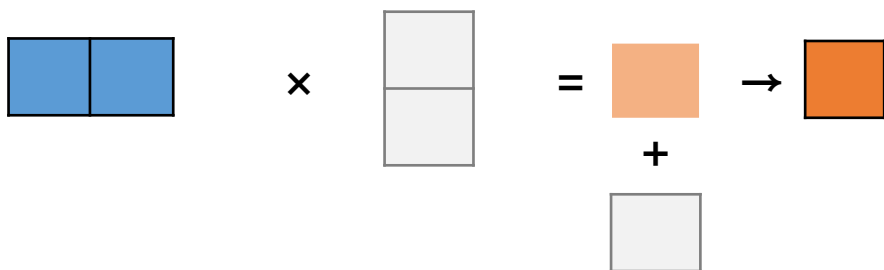
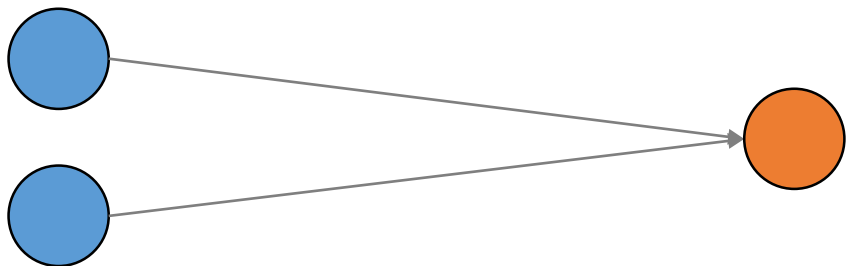
SepalLengthCm(x_1)	PetalLengthCm(x_2)	Species(y)
5.1	3.5	A
4.7	3.2	A
5.2	1.8	B
7	4.1	A
5.1	2.1	B

- 1) 입력이 2차원, 출력이 1차원
- 2) 이진 분류 문제

$$H(X) = \text{sigmoid}(W_1x_1 + W_2x_2 + b)$$

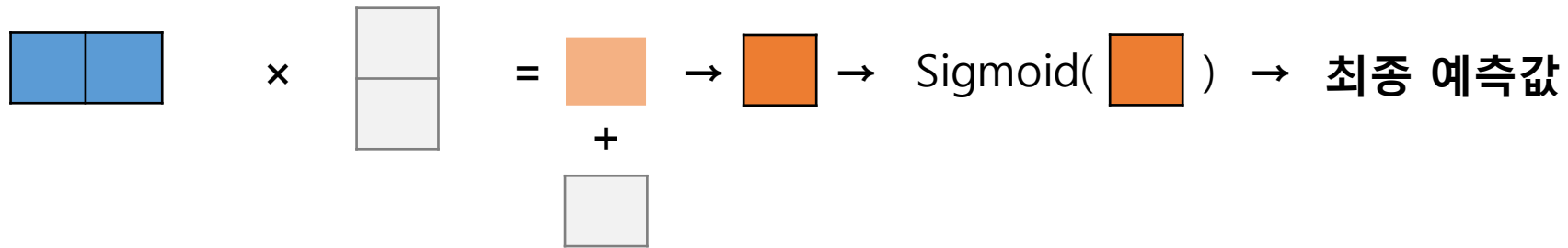
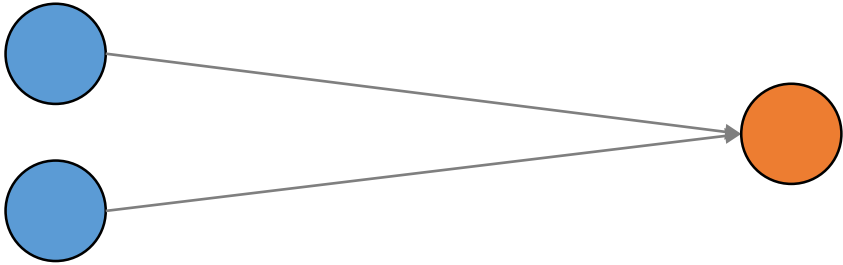
Logistic Regression

로지스틱 회귀는 다음과 같이 신경망으로 표현할 수 있다.



Logistic Regression

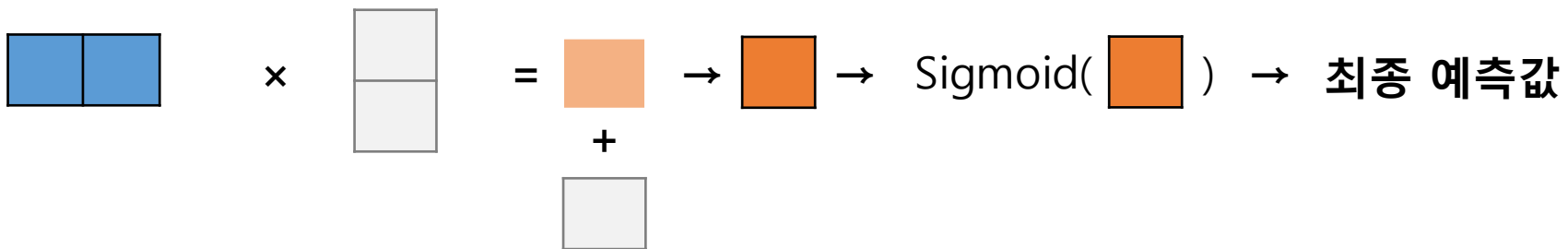
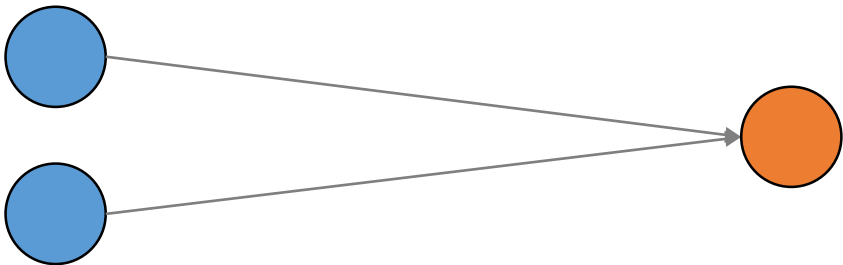
로지스틱 회귀는 다음과 같이 신경망으로 표현할 수 있다.



Logistic Regression

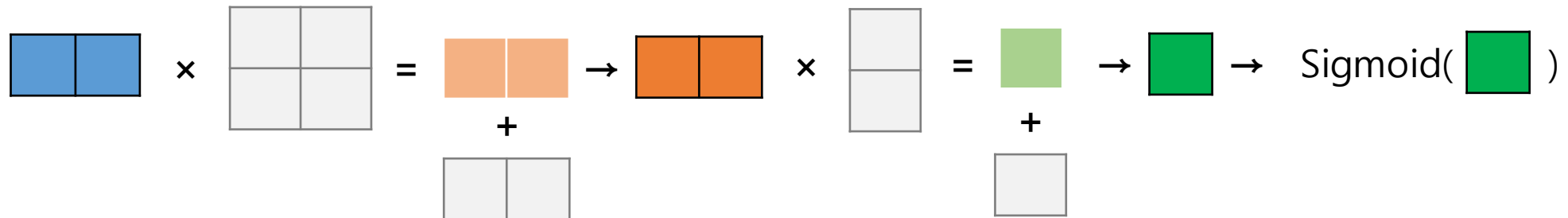
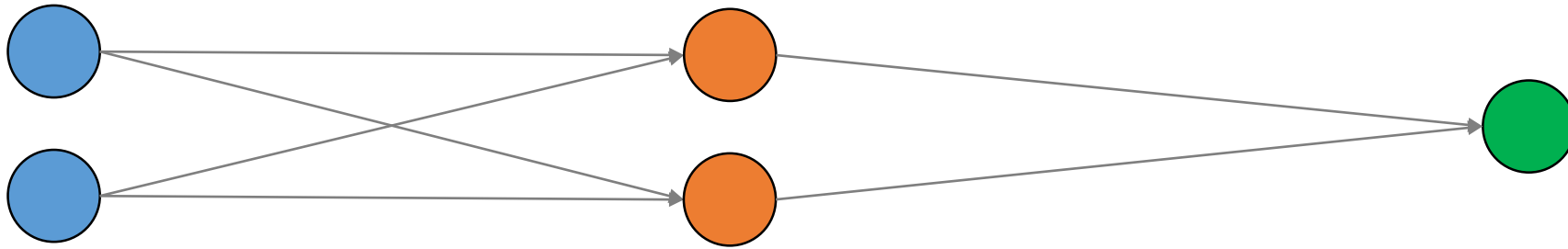
로지스틱 회귀는 다음과 같이 신경망으로 표현할 수 있다.

조금 더 복잡하게 층을 추가한다면?



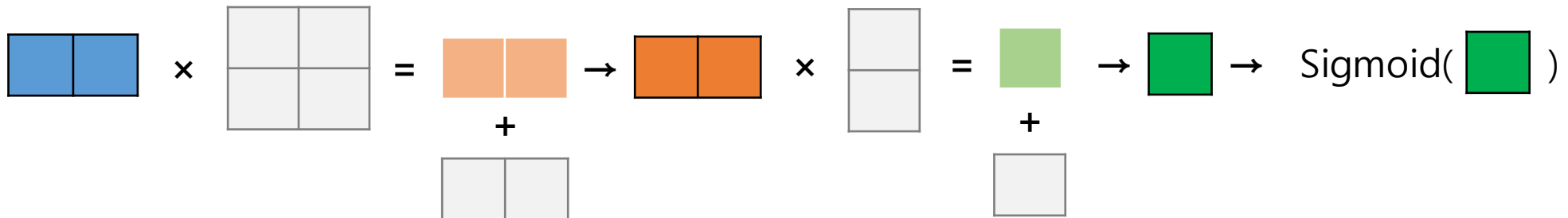
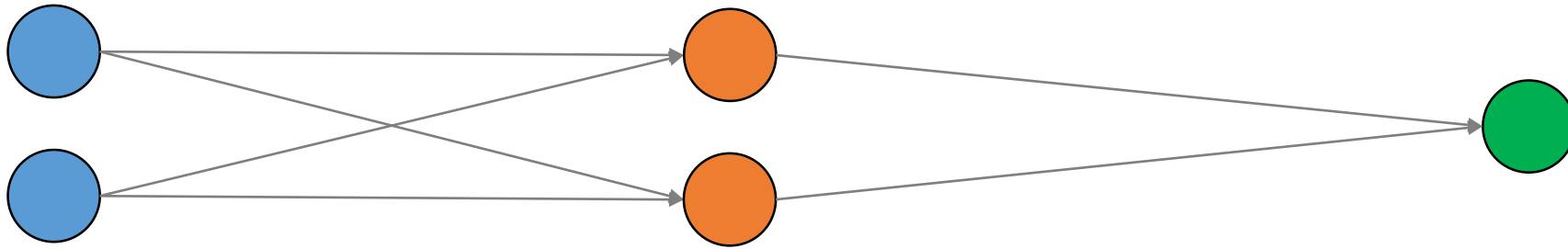
이진 분류를 수행하는 신경망

입력이 들어가는 '입력층'과 출력을 내뱉는 '출력층' 사이에 새로운 층을 추가.



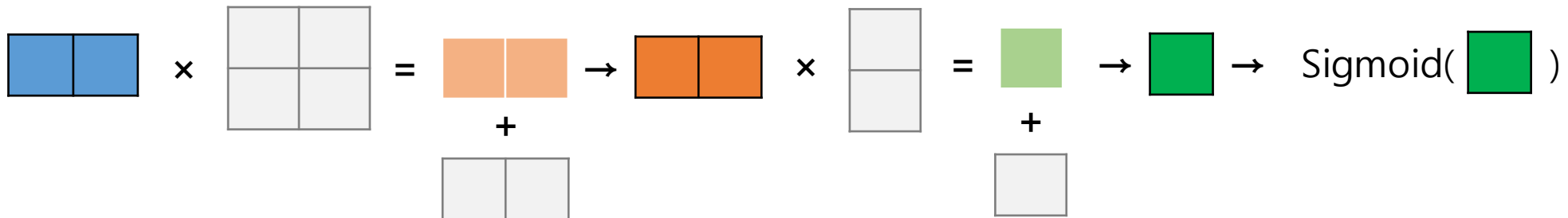
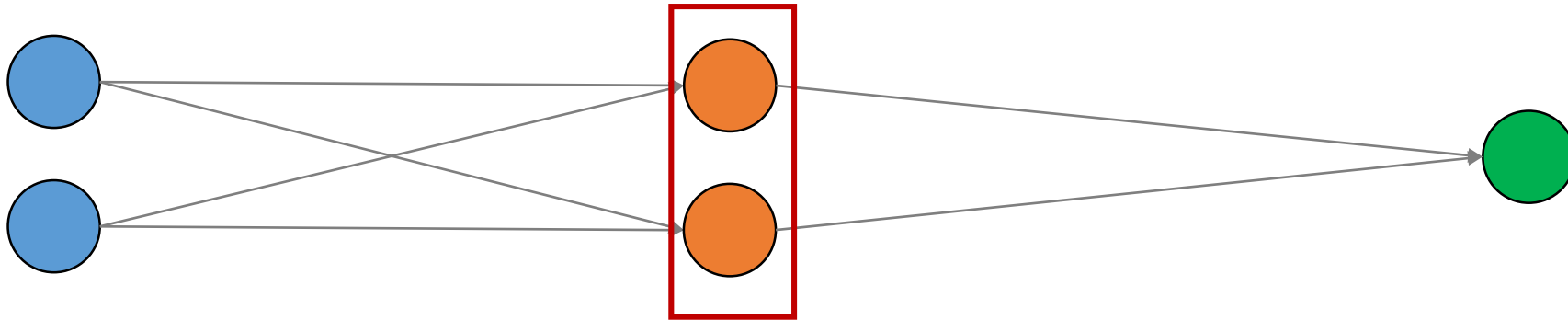
이진 분류를 수행하는 신경망

입력이 들어가는 '입력층'과 출력을 내뱉는 '출력층' 사이에 새로운 층을 추가.
이와 같이 '입력층'과 '출력층' 사이에 층을 추가하면 '**은닉층**'이라고 한다.



이진 분류를 수행하는 신경망

입력이 들어가는 '입력층'과 출력을 내뱉는 '출력층' 사이에 새로운 층을 추가.
이와 같이 '입력층'과 '출력층' 사이에 층을 추가하면 '**은닉층**'이라고 한다.



Neural Network 이해하기

입력의 특성이 3개이고, 세 개 중 1개를 선택해야 하는 다중 클래스 분류 문제.

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	Species(y)
5.1	3.5	1.4	setosa
4.9	3.0	1.4	setosa
5.8	2.6	4.0	versicolor
6.7	3.0	5.2	virginica
5.6	2.8	4.9	virginica

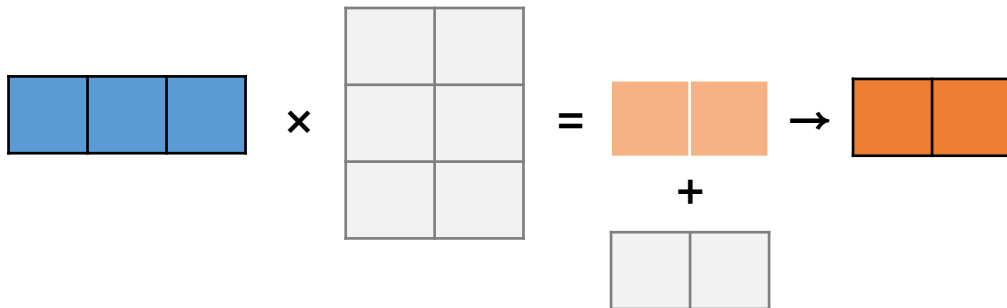
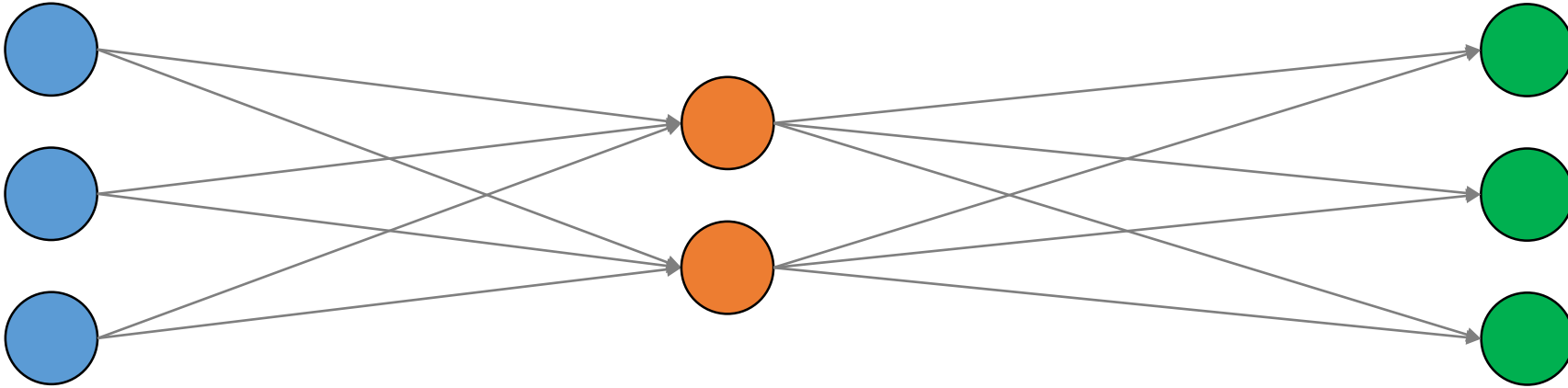
Neural Network 이해하기

입력의 특성이 3개이고, 세 개 중 1개를 선택해야 하는 다중 클래스 분류를 수행하는 신경망이 있다. 은닉층은 1개이며 뉴런은 2개이다. 이 신경망의 구조는?

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	Species(y)
5.1	3.5	1.4	setosa
4.9	3.0	1.4	setosa
5.8	2.6	4.0	versicolor
6.7	3.0	5.2	virginica
5.6	2.8	4.9	virginica

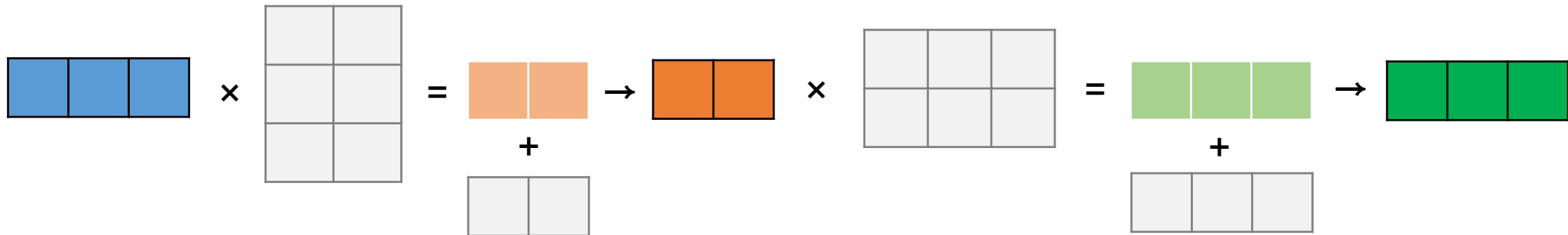
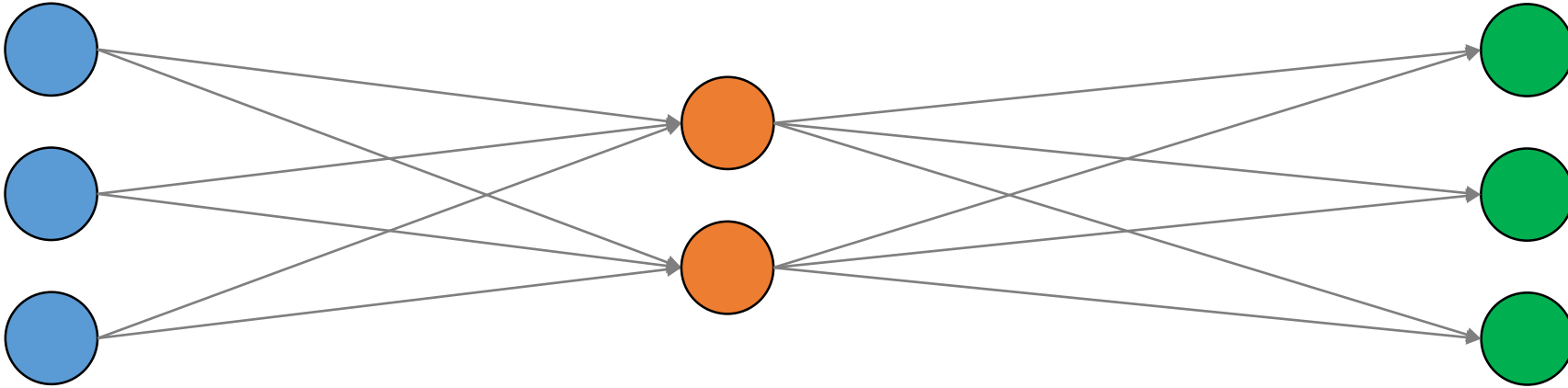
Neural Network 이해하기

입력의 특성이 3개이고, 세 개 중 1개를 선택해야 하는 다중 클래스 분류를 수행하는 신경망이 있다. 은닉층은 1개이며 뉴런은 2개이다. 이 신경망의 구조는?



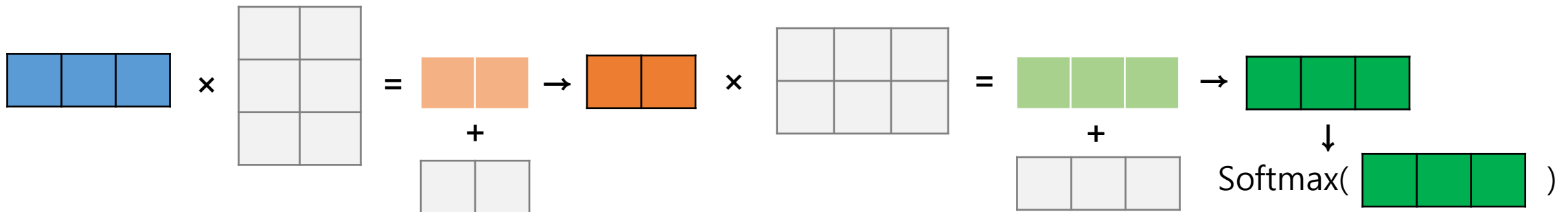
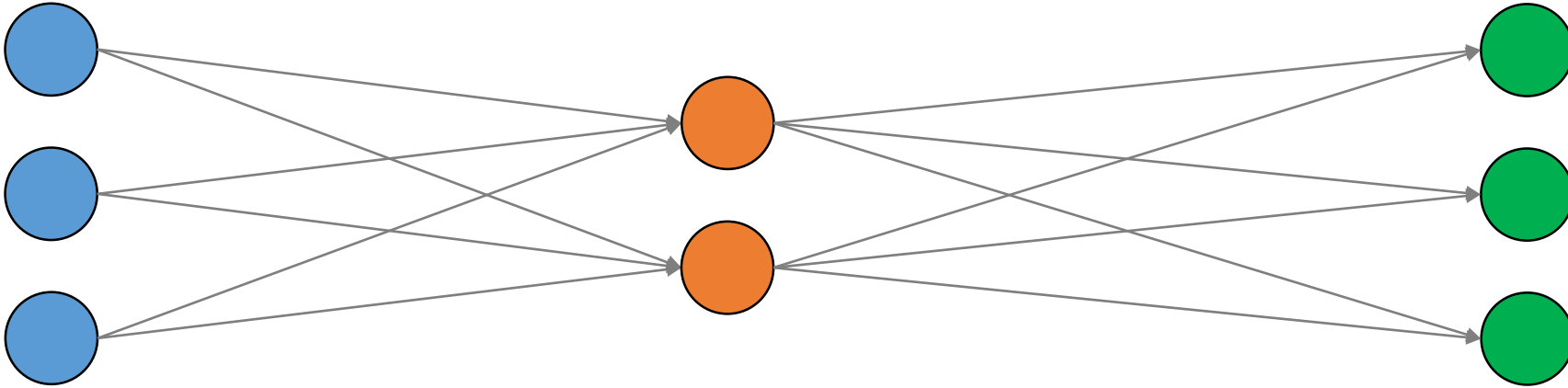
Neural Network 이해하기

입력의 특성이 3개이고, 세 개 중 1개를 선택해야 하는 다중 클래스 분류를 수행하는 신경망이 있다. 은닉층은 1개이며 뉴런은 2개이다. 이 신경망의 구조는?



Neural Network 이해하기

입력의 특성이 3개이고, 세 개 중 1개를 선택해야 하는 다중 클래스 분류를 수행하는 신경망이 있다. 은닉층은 1개이며 뉴런은 2개이다. 이 신경망의 구조는?



Text Classification

Text Classification

텍스트 분류란 주어진 텍스트가 이미 정의된 카테고리 중 어디로 분류되어야 하는지를 예측하는 작업
자연어 처리 작업 중 가장 수요가 높으면서도 기본적인 작업

실제로 쓰이는 예제

- 언론사의 뉴스들을 자동으로 IT, 정치, 문화 등의 카테고리로 분류하는 뉴스 분류기
- 스팸 메일이나 문자를 자동으로 스팸으로 판단하는 스팸 분류기
- 뉴스 데이터로부터 어떤 종목 뉴스인지 판별하여 해당 종목 게시판으로 분류
- 사용자가 남긴 리뷰로부터 긍정인지 부정인지 감성 분류
- 챗봇은 사용자가 입력한 질의로부터 '환불', '주문', '건의' 등인지 분류하고 이에 맞는 대응을 한다.

Text Classification

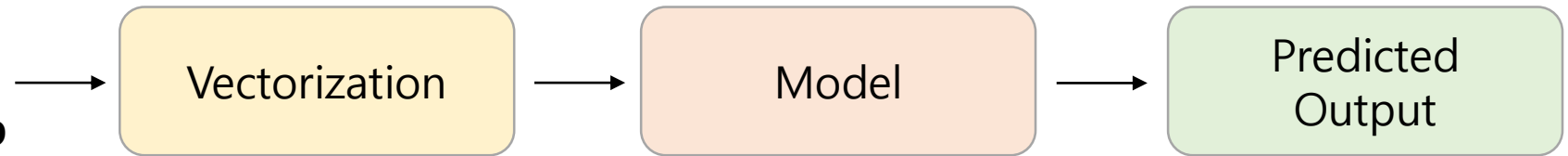
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.

문서1 : me free lottery

문서2 : free get free you

문서3 : you free scholarship

문서4 : free to contact me



Text Classification

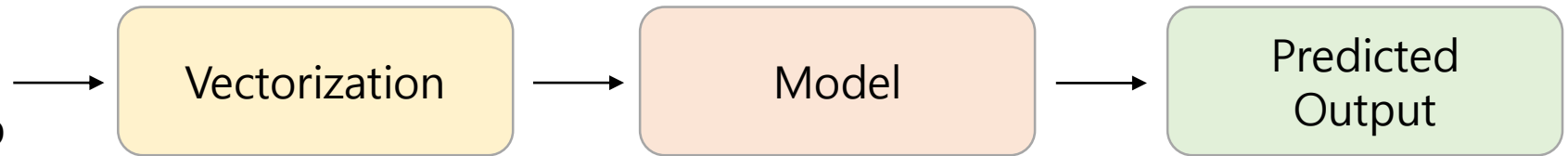
- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 이진 분류(Binary Classification)과 다중 클래스 분류(multiclass classification)가 있다.

문서1 : me free lottery

문서2 : free get free you

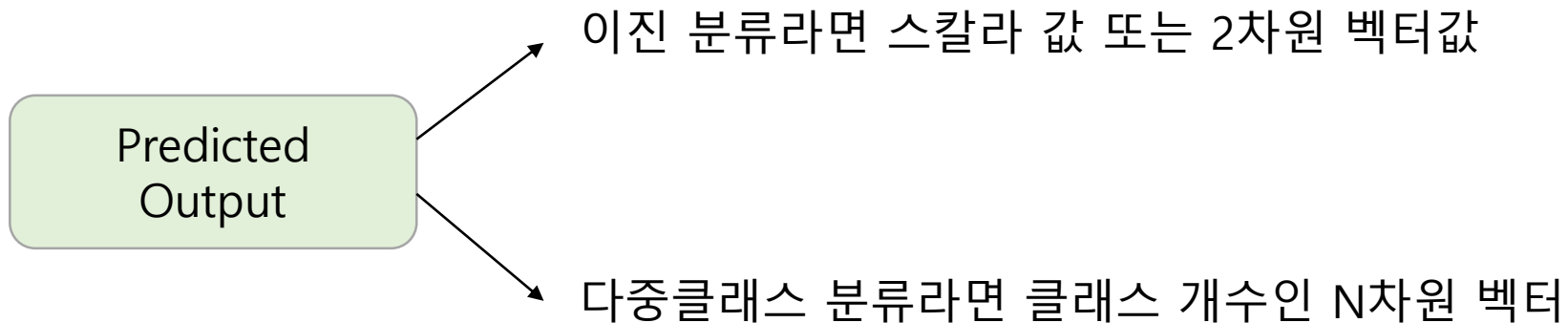
문서3 : you free scholarship

문서4 : free to contact me



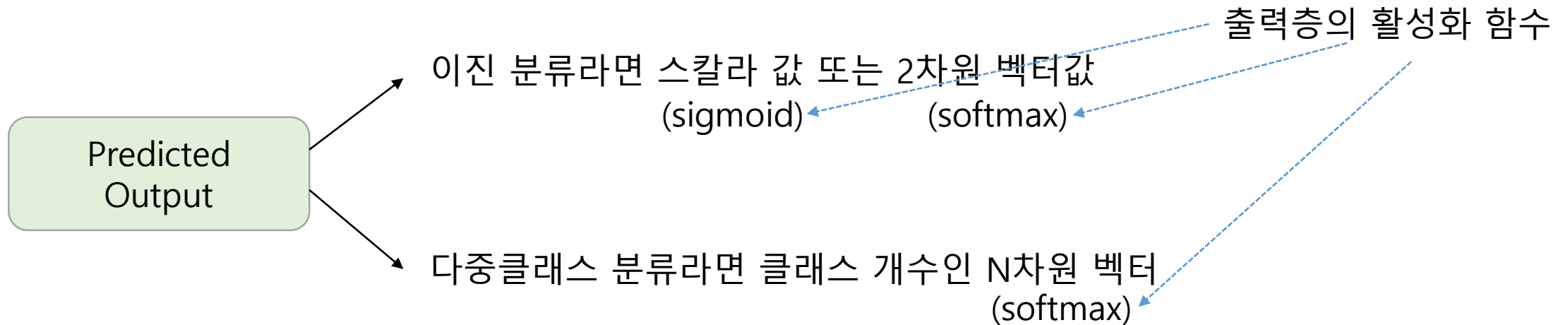
Text Classification

- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 이진 분류(Binary Classification)과 다중 클래스 분류(multiclass classification)가 있다.
- 클래스 수가 **2개**면 이진 분류라고 하며, **3개 이상**이면 다중클래스 분류



Text Classification

- 인공 신경망을 이용한 텍스트 분류 과정을 그림으로 도식화하면 아래와 같다.
- 분류는 크게 이진 분류(Binary Classification)과 다중 클래스 분류(multiclass classification)가 있다.
- 클래스 수가 **2개**면 이진 분류라고 하며, **3개 이상**이면 다중클래스 분류



Text Classification using MLP

각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환한다.

문서1 : me free lottery

문서2 : free get free you

문서3 : you free scholarship

문서4 : free to contact me

Bag of Words

Document Term Matrix

	contact	free	get	lottery	me	scholarship	to	you
0	0	1	0		1	1	0	0
1	0	2	1		0	0	0	1
2	0	1	0		0	0	1	0
3	1	1	0		0	1	0	1

문서 1의 벡터

Vocabulary size

Text Classification using MLP

각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환한다.

문서1 : me free lottery

문서2 : free get free you

문서3 : you free scholarship

문서4 : free to contact me

Bag of Words

Document Term Matrix

	contact	free	get	lottery	me	scholarship	to	you
0	0	1	0		1	1	0	0
1		0	2	1	0	0	0	1
2		0	1	0	0	0	1	0
3	1	1	0		0	1	0	1

Vocabulary size

문서 1의 벡터

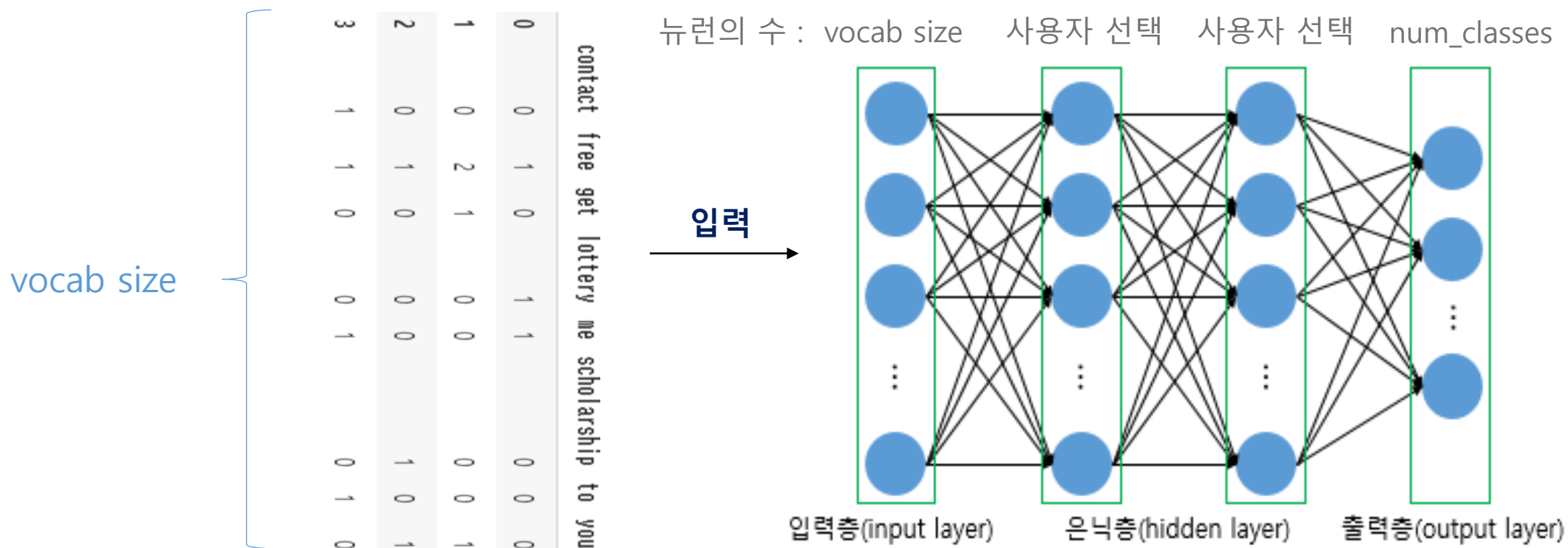
label
1
1
0
0

레이블

Text Classification using MLP

각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환한다.

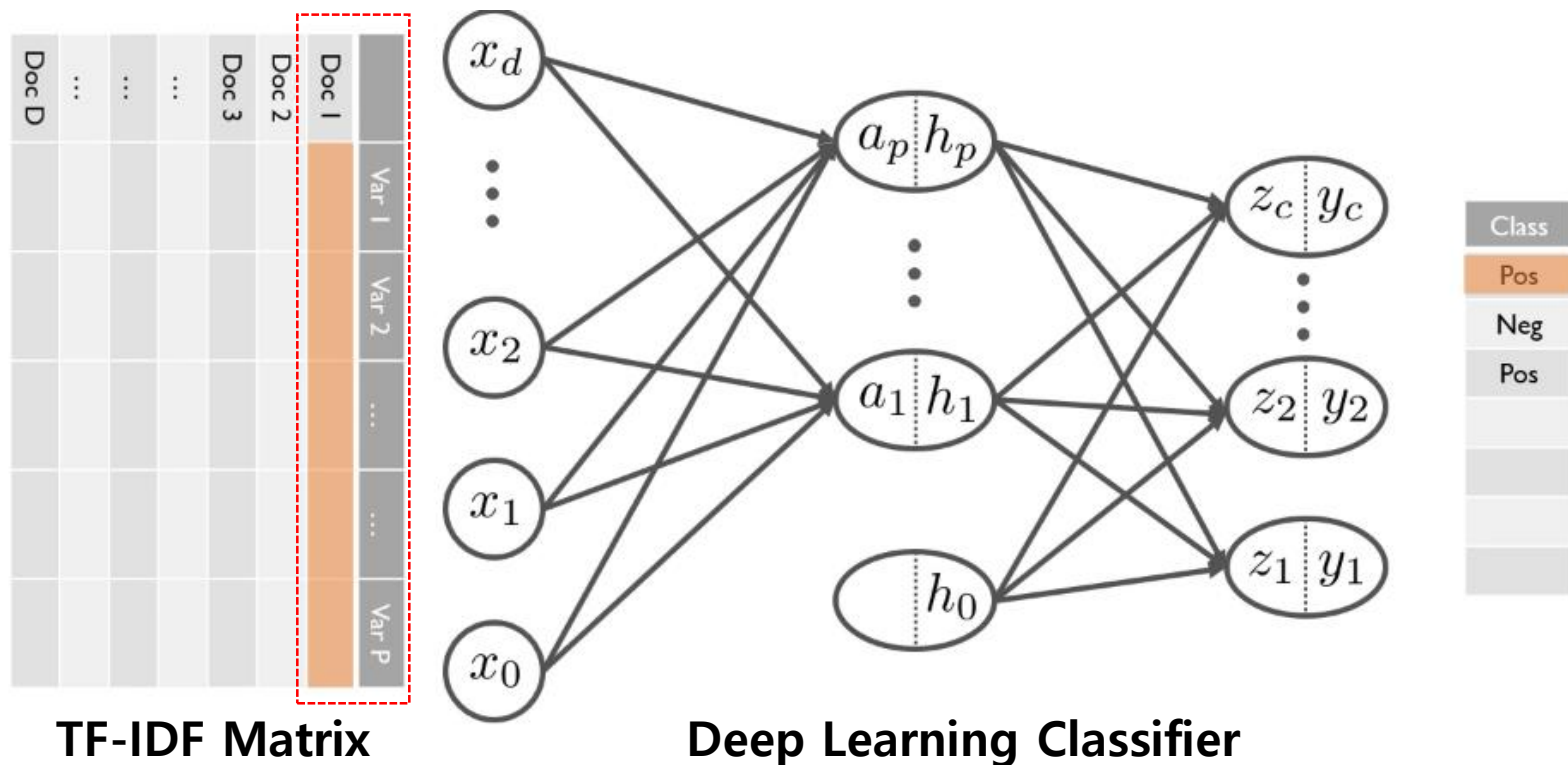
Vocabulary size를 입력층의 뉴런 수로 가지는 다층 퍼셉트론에 문서 벡터를 입력으로 사용.



Text Classification using MLP

각각의 문서를 다음과 같이 고정 길이를 가지는 벡터로 변환한다.

Vocabulary size를 입력층의 뉴런 수로 가지는 다층 퍼셉트론에 문서 벡터를 입력으로 사용.



Subword Tokenization

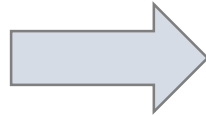
Tokenization

- 기계에게 어느 구간까지가 문장이고, 단어인지를 알려주어야 합니다.
- 문장 토큰화, 단어 토큰화, subword 토큰화 등. 다양한 단위의 토큰화가 존재.

왜 필요할까요?

His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. He dug a hole in the midst of some reeds. He looked about, to mae sure no one was near.

word
tokenization



['His', 'barber', 'kept', 'his', 'word', '.', 'But', 'keeping', 'such', 'a', 'huge', 'secret', 'to', 'himself', 'was', 'driving', 'him', 'crazy', '.', 'Finally', ',', 'the', 'barber', 'went', 'up', 'a', 'mountain', 'and', 'almost', 'to', 'the', 'edge', 'of', 'a', 'cliff', '.', 'He', 'dug', 'a', 'hole', 'in', 'the', 'midst', 'of', 'some', 'reeds', '.', 'He', 'looked', 'about', ',', 'to', 'mae', 'sure', 'no', 'one', 'was', 'near', '.']

Subword Tokenization

- 하나의 단어 안에는 의미를 가진 더 작은 토큰들이 있을 것이다.

영어

- Retry → Re(다시) + try(시도하다)
- Birthplace → Birth(출생) + place(장소)

한국어

- 학교(學校) → 배울 학 + 학교 교
- 집중력(集中力) → 모을 집 + 가운데 중 + 힘 력