딥러닝을 이용한 자연어 처리 입문

08. 딥 러닝(Deep Learning) 개요

06) 케라스 훑어보기 ~ 10) 피드 포워드 신경망 언어 모델

집현전 초급 10조
성민석 / 김민정 / 차민기

전처리(Preprocessing) **란**?

- **원시 데이터를 딥/머신 러닝에 사용 할 수 있도록 변환하는 과정**
- **학습 데이터로 사용하기 위해 사람의 언어를 기계가 알아 들을 수 있도록 변환하는 인코딩(encoding)과 데이터들의 길이를 맞추는 패딩(padding) 과정을 수행함**

케라스(Keras) **란**?

- **대표적인 딥러닝 파이썬 라이브러리**
- **딥러닝을 구현 할 수 있도록 도와주는 상위 레벨의 인터페이스**
- https://keras.io/

Tokenizer (1/2)

- **토큰화와 정수 인코딩을 위해 사용**
- **본 장에서는** Keras **라이브러리의** Tokenizer**를 사용함**
- **문장의 공백을 기준으로 토큰화를 수행하며 정수 인코딩을 수행함**

```
[1] from tensorflow.keras.preprocessing.text import Tokenizer
```

- [1] Tokenizer 사용하기 위해 선언

```
[1] t = Tokenizer()
[2] fit_text = "The earth is an awesome place live"
[3] t.fit_on_texts([fit_text])

[4] test_text = "The earth is an great place live"
[5] sequences = t.texts_to_sequences([test_text])[0]
```

- [1] Tokenizer 객체 생성
- [2] 인코딩 학습에 사용되는 텍스트 데이터
- [3] 토큰화 이후 인코딩 학습
- [4] 인코딩을 수행할 테스트 데이터
- [5] 텍스트 토큰화 및 인코딩 수행

Tokenizer (2/2)

```
[1] print("sequences : ", sequences)
[2] # “The earth is an great place live” => [1, 2, 3, 4, 6, 7]
[3] print("word_index : ", t.word_index)
[4] # word_index :  {'the': 1, 'earth': 2, 'is': 3, 'an': 4, 'awesome': 5,
 'place': 6, 'live': 7}
```

- [1]-[2] 토큰화 및 인코딩이 끝난 테스트 데이터 출력
- 학습에 사용된 텍스트 데이터에는 ‘great’ 단어가 없으므로 인코딩 결과 출력되지 않음
- [3]-[4] 학습된 Tokenizer의 단어 집합을 출력함

padding
- 모델 학습에 사용될 각 샘플의 길이는 서로 다를 수 있기에 학습에 사용하기 위해서는 길이를 동일하게 맞추는 과정이 필요함
- 이러한 과정을 패딩(padding)이라 부르며 케라스에서는 pad_sequence()를 사용해서 작업을 수행함
- 사용자가 정한 최대 길이보다 샘플의 길이가 길면 값의 일부를 자르고, 샘플의 길이가 짧으면 특정 값(0)으로 채움

```
[1] from tensorflow.keras.preprocessing.sequence import pad_sequences
[2] padding = pad_sequences([[1, 2, 3], [3, 4, 5, 6], [7, 8]], maxlen=3, padding='pre')
[3] print(padding)
[4] # [[1 2 3] [4 5 6] [0 7 8]]
```

- [1] pad_sequences 사용하기 위해 선언
- [2]-[3] 입력 데이터들의 길이를 동일하게 맞춤
- pad_sequences의 인자 값은 다음과 같음:
  첫번째 : 패딩을 진행할 데이터
  두번째 : 모든 데이터에 대해서 정규화 할 길이
  세번째 : 패딩 값을 채울 위치('pre' = 앞, 'post' = 뒤)
- [3]-[4] 패딩을 수행한 결과를 보여줌. 모두 동일한 길이의 값을 가지게 됐음

워드 임베딩(Word Embedding) 란? (1/2)

- **텍스트 내의 단어들을 밀집 벡터**(dense vector)**로 만드는 것을 의미함**
- **앞서 배운 대표적인 희소 벡터**(sparse vector)**인 원-핫 벡터는** 0, 1 **정수로 이루어진 고차원 벡터로서 간어간 유사도가 모두 동일한 단점이 있음**
- **밀집 벡터**(dense vector)**란 실수로 이루어진 저차원 벡터임**

	원-핫 벡터	임베딩 벡터
차원	고차원	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습
값의 타입	1과 0	실수

- **단어를 원-핫 벡터로 만드는 과정을 원-핫 인코딩이라 하며 벡터의 차원이 매우 크다는 성질을 가짐**(**차원이** 20,000**이상 넘어갈 수 있음**)
- **단어를 밀집 벡터로 만드는 작업을 워드 임베딩이라 하며 원-핫 벡터와 달리 작은 차원을 가짐**(256, 512, 1024**등의 차원을 가짐**)

워드 임베딩(Word Embedding) 란? (2/2)

- 케라스의 Embedding()은 단어를 밀집 벡터로 만드는 역할을 수행함
- 인공 신경망 용어로는 임베딩 층(embedding layer)을 만든다고 표현함
- Embedding()은 정수 인코딩이 된 단어들을 입력 받아 임베딩을 수행함

```
[1] text=[['Hope', 'to', 'see', 'you', 'soon'],['Nice', 'to', 'see', 'you', 'again']]
[2] text=[[0, 1, 2, 3, 4],[5, 1, 2, 3, 6]]
[3] Embedding(7, 2, input_length=5)
[4]‘‘‘
+------------+------------+
| index      | embedding |
+------------+------------+
| 0          | [1.2, 3.1] |
| 1          | [0.1, 4.2] |
| 2          | [1.0, 3.1] |
| 3          | [0.3, 2.1] |
| 4          | [2.2, 1.4] |
| 5          | [0.7, 1.7] |
| 6          | [4.1, 2.0] |
+------------+------------+
‘‘‘
```

- Embedding(단어 집합의 크기, 임베딩 벡터의 출력 차원, 입력 시퀀스의 길이)
- [1]-[2] 입력 문장을 토큰화하고 각 단어에 대한 정수 인코딩 결과
- [3] 인코딩된 각 정수의 데이터가 임베딩 층에 입력되는 과정을 보여줌
- [4] 입력된 각 정수는 임베딩의 인덱스로 사용되며 각 단어에 대해 임베딩 벡터를 리턴함

Sequential (1/3)

- 앞서 인공 신경망 챕터에서 배운 입력층, 은닉층, 출력층을 케라스로 구성하기 위해 Sequential()를 사용함
- Sequential()을 model로 선언한 뒤에 model.add()라는 코드를 통해 층을 단계적으로 추가할 수 있음

```
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(...) # 층 추가
model.add(...) # 층 추가
model.add(...) # 층 추가
```

- Embedding()을 통해 생성하는 임베딩 층(embedding layer) 또한 인공 신경망의 하나의 층이므로 model.add()로 추가해야함

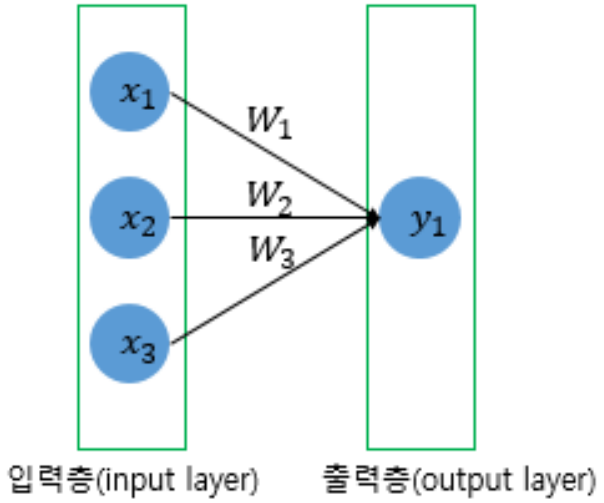
```
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Embedding(vocabulary, output_dim, input_length))
```

Sequential (2/3)

- Dense() : 전결합층(fully-connected layer)을 추가함
- Dense(출력 뉴런의 수, 입력 뉴런의 수, 활성화 함수)
- 활성화 함수(Activation function) : linear, sigmoid, softmax, relu 등

```
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(1, input_dim=3, activation='relu'))
```

- 위 코드를 해석하면 3개의 입력 뉴런을 입력 받아 1개의 뉴런을 출력하라는 의미임
- 이 과정을 시각화 하면 아래의 그림처럼 표현됨

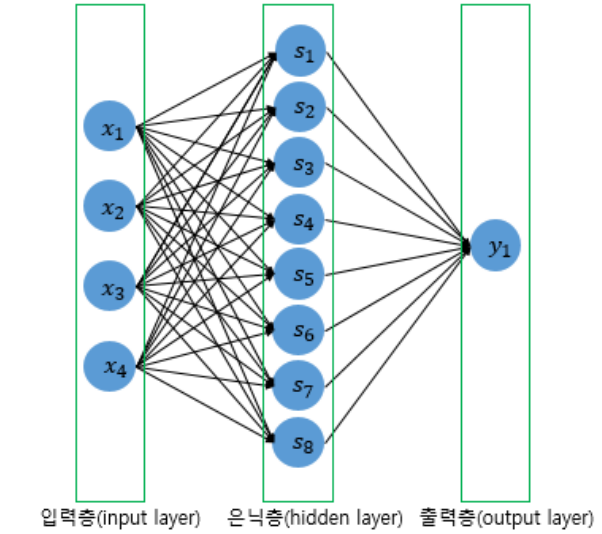


Sequential (3/3)

- Dense()는 여러 번 중첩하여 사용 할 수 있음

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
model.add(Dense(8, input_dim=4, activation='relu'))
model.add(Dense(1, activation='sigmoid')) # 출력층
```

- Dense()를 한번 더 사용하여 전결합층을 하나 더 추가하였음
- 첫번째 Dense()의 출력 뉴런의 수인 8은 은닉층의 뉴런으로 사용됨
- 두번째 Dense()의 input_dim의 값이 없는 이유는 첫번째 Dense()층의 뉴런의 수가 8개란 사실을 알고 있기 때문임
- 이 외에도 다양한 층(LSTM, GRU, Convolution2D, BatchNormalization 등)을 만들 수 있음



Summary
- 모델의 정보를 요약해서 보여줌

model.summary()

```
_________________________________________________________________
Layer(type)                 Output Shape              Param #
=================================================================
dense_1(Dense)              (None, 8)                 40
_________________________________________________________________
dense_2(Dense)              (None, 1)                 9
=================================================================
Total params: 49
Trainable params:
49 Non-trainable params: 0
_________________________________________________________________
```

컴파일(Compile) 란?
- 모델을 기계가 이해할 수 있도록 하는 과정.
- 오차 함수와 최적화 방법, 메트릭 함수를 선택할 수 있음

```
# 이 코드는 뒤의 텍스트 분류 챕터의 스팸 메일 분류하기 실습 코드를 갖고온 것임.
from tensorflow.keras.layers import SimpleRNN, Embedding, Dense
from tensorflow.keras.models import Sequential
max_features = 10000

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32)) #RNN에 대한 설명은 뒤의 챕터에서 합니다.
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

- 위 코드는 임베딩층, 은닉층, 출력층을 추가하여 모델을 설계한 후에, 마지막으로 컴파일 하는 과정을 보여줌

- **optimizer** : 훈련 과정을 설정하는 옵티마이저를 설정합니다. 'adam'이나 'sgd'와 같이 문자열로 지정할 수도 있음
- **loss** : 훈련 과정에서 사용할 손실 함수(loss function)를 설정함
- **metrics** : 훈련을 모니터링하기 위한 지표를 선택함

대표적으로 사용되는 손실 함수와 활성화 함수의 조합

문제 유형	손실 함수명	활성화 함수명	참고 설명
회귀	mean_squared_error	-	
다중 클래스 분류	categorical_crossentropy	softmax	10챕터 참고
다중 클래스 분류	sparse_categorical_crossentropy	softmax	원-핫 인코딩의 상태에서 사용되는 Categorical_crossentropy와 달리 정수 인코딩 상태에서 수행
이진 분류	Binary_crossentropy	sigmoid	10챕터 참고

학습(fitting)이란?
-모델이 오차로부터 매개 변수를 업데이트 시키는 과정

```
# 위의 compile() 코드의 연장선상인 코드
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data(X_val, y_val))
```

- 첫번째 인자 : 훈련 데이터
- 두번째 인자 : 지도 학습에서 레이블 데이터
- Epochs : 총 훈련 횟수
- Batch_size : 배치 크기

```
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_data(X_val, y_val))
```

- validation_data(x_val, y_val) : 검증 데이터, 검증 데이터를 사용하면 각 에포크 마다 정확도를 출력함. 이 정확도를 통해 훈련이 잘 되고 있는지 판단 할 수 있으며 모델 학습 시 검증 데이터를 학습하지는 않음. 검증 데이터의 loss가 낮아지다가 높아지기 시작하면 과적합의 신호임

```
# 훈련 데이터의 20%를 검증 데이터로 사용.
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_split=0.2))
```

- validation_split : validation_data를 대신해서 사용 할 수 있음. 별도로 존재하는 검증 데이터를 주는 것이 아니라 X_train, y_train에서 일정 비율을 분리하여 검증 데이터로 사용함

```
# verbose = 1일 경우.
Epoch 88/100 7/7 [==============================] - 0s 143us/step - loss: 0.1029 - acc: 1.0000

# verbose = 2일 경우.
Epoch 88/100 - 0s - loss: 0.1475 - acc: 1.0000
```

- Vervose : 학습 중 출력되는 문구
  - 0 : 아무 것도 출력하지 않음
  - 1 : 훈련의 진행도를 보여주는 진행 막대를 표기함
  - 2 : 미니 배치마다 손실 정보를 출력함

평가(evaluation)란?
- 테스트 데이터를 통해 학습한 모델에 대한 정확도를 측정하는 과정

```
# 위의 fit() 코드의 연장선상인 코드
model.evaluate(X_test, y_test, batch_size=32)
```

- 첫번째 인자 : 테스트 데이터
- 두번째 인자 : 지도 학습에서 레이블 테스트 데이터
- Batch_size : 배치 크기

예측(predict)이란?
- 임의의 입력에 대한 모델의 출력값을 확인하는 과정

```
# 위의 fit() 코드의 연장선상인 코드
model.predict(X_input, batch_size=32)
```

- 첫번째 인자 : 예측하려는 임의의 데이터
- Batch_size : 배치 크기

학습을 하는 과정 혹은 끝나고나서 모델을 저장하고 불러오는 일은 매우 중요함

모델 저장

```
model.save("model_name.h5")
```

모델 불러오기

```
from tensorflow.keras.models import load_model
model = load_model("model_name.h5")
```

- 앞서 구현한 선형, 로지스틱, 소프트맥스 회귀 모델들과 케라스 훑어보기 챕터에서 배운 케라스의 모델 설계 방식은 Sequential API를 사용한 방식임
- Sequential API는 여러층을 공유하거나 다양한 종류의 입력과 출력을 사용하는 등의 복잡한 모델을 만드는 일에는 한계가 있음
- 더욱 복잡한 모델을 생성하기 위해서는 Functional API(함수형 API)에 대해 알아야함

```
# 이 코드는 소프트맥스 회귀 챕터에서 가져온 코드임.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(3, input_dim=4, activation='softmax'))
```

- 위 코드와 같은 방식(Sequential API)은 직관적이고 편리하지만 단순히 층을 쌓는 것만으로 복잡한 신경망을 구현할 수 없음
- 따라서 초심자에겐 적합한 API지만, 전문가가 되기 위해서는 결과적으로 Functional API를 학습 해야함

- Functional API는 각 층을 일종의 함수(Function)로서 정의함
- 각 함수를 조합하기 위한 연산자들을 제공하는데, 이를 이용하여 신경망을 설계함
- 이 장에서는 Functional API로 FFNN, RNN등 다양한 모델을 만들면서 기존의 Sequential API와 차이를 이해함

전결합 피드 워드 신경망(Fully-connected FFNN) (1/3)

- Sequential API와는 다르게 Functional API에서는 입력 데이터 크기(Shape)를 인자로 입력층에 정의 해야함
- 아래 코드들은 입력의 차원이 1인 FFNN을 만드는 과정임

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

# 텐서를 리턴한다.
inputs = Input(shape=(10,))
```

- 위 코드는 10개의 입력을 받는 입력층을 보여줌
- Input() : 함수에 입력크기를 정의함

전결합 피드 워드 신경망(Fully-connected FFNN) (2/3)

```
inputs = Input(shape=(10,))
hidden1 = Dense(64, activation='relu')(inputs)
hidden2 = Dense(64, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
```

- 위 코드는 앞 페이지의 코드에 은닉층과 출력층을 추가한 코드임
- 이전층을 다음층 함수의 입력으로 사용하고, 변수에 할당함

```
inputs = Input(shape=(10,))
hidden1 = Dense(64, activation='relu')(inputs)
hidden2 = Dense(64, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
model = Model(inputs=inputs, outputs=output)
```

- 마지막으로 Model에 입력 텐서와 출력 텐서를 정의하여 하나의 모델로 구성함
- Model() : 함수에 입력과 출력을 정의함

전결합 피드 워드 신경망(Fully-connected FFNN) (3/3)

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(data, labels)
```

- 이제 이를 model로 저장하면 sequential API를 사용할 때와 마찬가지로 model.compile, model.fit 등을 사용할 수 있음

선형 회귀(Linear Regression)

- https://wikidocs.net/111472 에서 케라스의 Sequential API로 구현했던 선형 회귀를 Functional API로 구현함

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model

X = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적
inputs = Input(shape=(1,))
output = Dense(1, activation='linear')(inputs)
linear_model = Model(inputs, output)

sgd = optimizers.SGD(lr=0.01)

linear_model.compile(optimizer=sgd ,loss='mse',metrics=['mse'])
linear_model.fit(X, y, batch_size=1, epochs=300, shuffle=False)
```

로지스틱 회귀(Logistic Regression)

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

inputs = Input(shape=(3,))
output = Dense(1, activation='sigmoid')(inputs)
logistic_model = Model(inputs, output)
```

다중 입력을 받는 모델(model that accepts multiple inputs) (1/2)

- functional API를 사용하면 아래와 같이 다중 입력과 다중 출력을 가지는 모델도 만들 수 있음

```
# 최종 완성된 다중 입력, 다중 출력 모델의 예
model = Model(inputs=[a1, a2], outputs=[b1, b2, b3])
```

다중 입력을 받는 모델(model that accepts multiple inputs) (2/2)
- 아래 코드는 다중 입력을 받는 모델을 입력층부터 출력층까지 설계한 코드임

```
from tensorflow.keras.layers import Input, Dense, concatenate
from tensorflow.keras.models import Model

# 두 개의 입력층을 정의
inputA = Input(shape=(64,))
inputB = Input(shape=(128,))

# 첫번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의
x = Dense(16, activation="relu")(inputA)
x = Dense(8, activation="relu")(x)
x = Model(inputs=inputA, outputs=x)

# 두번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의
y = Dense(64, activation="relu")(inputB)
y = Dense(32, activation="relu")(y)
y = Dense(8, activation="relu")(y)
y = Model(inputs=inputB, outputs=y)

result = concatenate([x.output, y.output]) # 두개의 인공 신경망의 출력을 연결(concatenate)

z = Dense(2, activation="relu")(result) # 연결된 값을 입력으로 받는 밀집층을 추가(Dense layer)
z = Dense(1, activation="linear")(z) # 선형 회귀를 위해 activation=linear를 설정
model = Model(inputs=[x.input, y.input], outputs=z) # 결과적으로 이 모델은 두 개의 입력층으로부터 분기되어 진행된 후 마지막에는 하나의 출력을 예측하는 모델이 됨.
```

RNN(Recurrence Neural Network) 은닉층 사용하기
- 아래 코드는 RNN 은닉층을 가지는 모델을 설계한 결과임

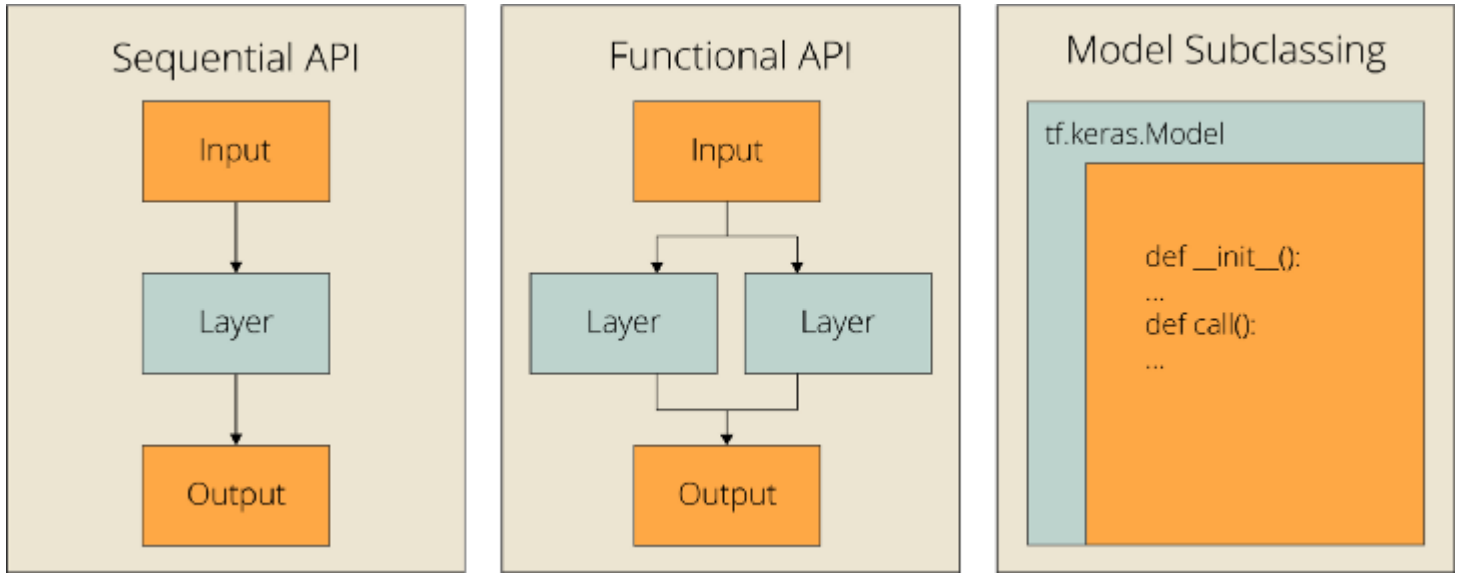
```
from tensorflow.keras.layers import Input, Dense, LSTM
from tensorflow.keras.models import Model

inputs = Input(shape=(50,1))
lstm_layer = LSTM(10)(inputs) # RNN의 일종인 LSTM을 사용
x = Dense(10, activation='relu')(lstm_layer)
output = Dense(1, activation='sigmoid')(x)
model = Model(inputs=inputs, outputs=output)
```

- 하나의 특성(feature)에 50개의 time-step을 입력으로 받는 모델을 설계함

다르게 표현해 보기
- 아래 두 코드는 동일한 동작을 수행하지만 다르게 표현할 수도 있음

```
encoder = Dense(128)(input)
```

```
encoder = Dense(128)
encoder(input)
```



- 케라스의 구현 방식에는 Sequential API, Functional API 외에도 Subclassing API라는 구현 방식이 존재함
- Subclassing API를 통해 선형 회귀를 구현해 보겠음

- https://wikidocs.net/111472 에서 케라스의 Sequential API로 구현했던 선형 회귀를 Subclassing API로 구현함

```
import tensorflow as tf
```

```
class LinearRegression(tf.keras.Model):
  def __init__(self):
    super(LinearRegression, self).__init__()
    self.linear_layer = tf.keras.layers.Dense(1, input_dim=1, activation='linear')
  def call(self, x):
    y_pred = self.linear_layer(x)
    return y_pred
```

```
model = LinearRegression()
```

```
X = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적
```

```
# sgd는 경사 하강법을 의미. 학습률(learning rate, lr)은 0.01.
sgd = tf.keras.optimizers.SGD(lr=0.01)
# 손실 함수(Loss function)은 평균제곱오차 mse를 사용합니다.
model.compile(optimizer=sgd ,loss='mse',metrics=['mse'])
# 주어진 X와 y데이터에 대해서 오차를 최소화하는 작업을 300번 시도합니다.
model.fit(X,y, batch_size=1, epochs=300, shuffle=False)
```

- Class 형태의 모델은 tf.keras.Model을 상속 받음
- `__init__` : 모델의 구조와 동적을 정의하는 생성자를 정의함, 객체가 생성될 때 자동으로 호출됨
- `super()` : 이 함수가 호출되면 상속 받은 `tf.keras.Model` 클래서의 속성들을 가지고 초기화 함
- `call()` : 모델이 데이터를 입력받아 예측값을 리턴하는 `forward` 연산을 수행함
  - ✓ Forward 연산 : H(x) 식에 입력 x로부터 예측된 y를 얻음

- Sequential API는 간단한 모델을 구현하기에 적합함
- Functional API는 Sequential API로 구현할 수 없는 복잡한 모델들을 구현할 수 있음
- Subclassing API
  - ✓ Functional API로 구현할 수 없는 모델들을 구현함
  - ✓ 대표적으로 재귀 네트워크나 트라 RNN은 Functional API로 구현할 수 없음

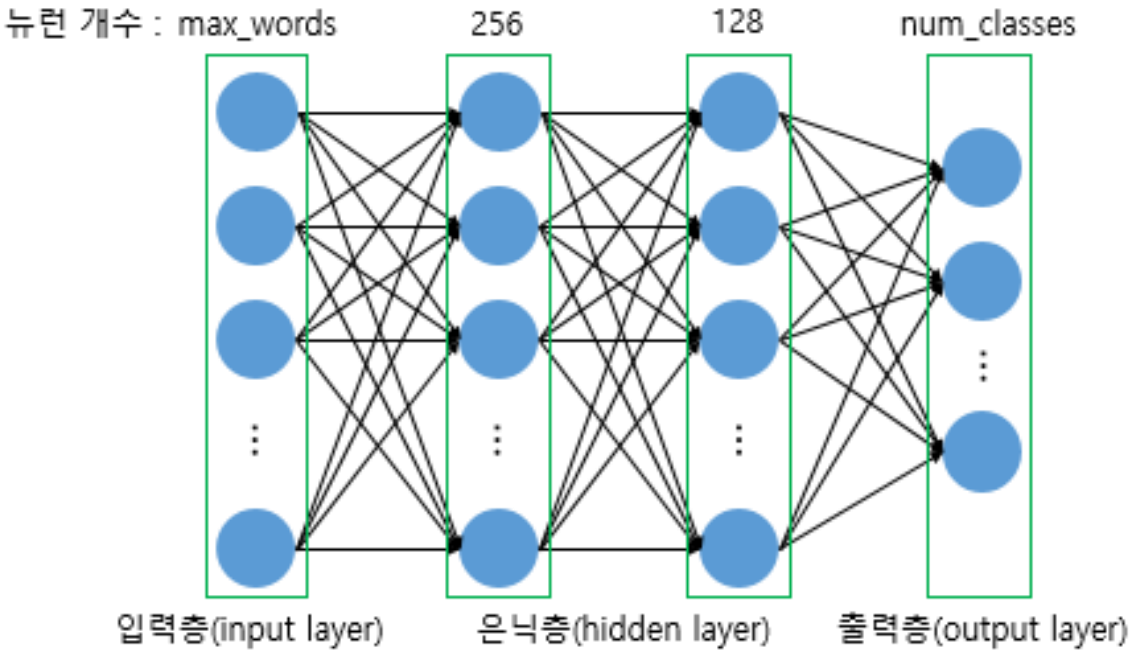
- Sequential API
  - ✓ 장점 : 단순하게 층을 쌓는 방식으로 쉽고 사용하기 간단함
  - ✓ 단점 : 다수의 입력, 다수의 출력을 가진 모델 또는 층 간의 연결이나 덧셈과 같은 연산을 하는 모델을 구현하기에는 적합하지 않음

- Functional API
  - ✓ 장점 : Sequential API로는 구현하기 어려운 복잡한 모델을 구현할 수 있음
  - ✓ 단점 : 입력의 크기(Shape)를 명시한 입력층을 모델의 앞단에 정의 해야함

- Subclassing API
  - ✓ 장점 : Functional API로 구현할 수 없는 모델들조차 구현이 가능함
  - ✓ 단점 : 객체 지향 프로그래밍에 익숙해야 하므로 코드 사용이 가장 까다로움

(복습) 다층 퍼셉트론

- 단층 퍼셉트론의 형태에서 은닉층이 1개 이상 추가된 신경망
- 피드 포워드 신경망(Feed Forward Neural Network, FFNN)의 가장 기본적인 형태
- 입력층에서 출력층으로 오직 한 방향으로만 연산 방향이 정해져 있는 신경망



Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
```

Keras의 texts_to_matrx()

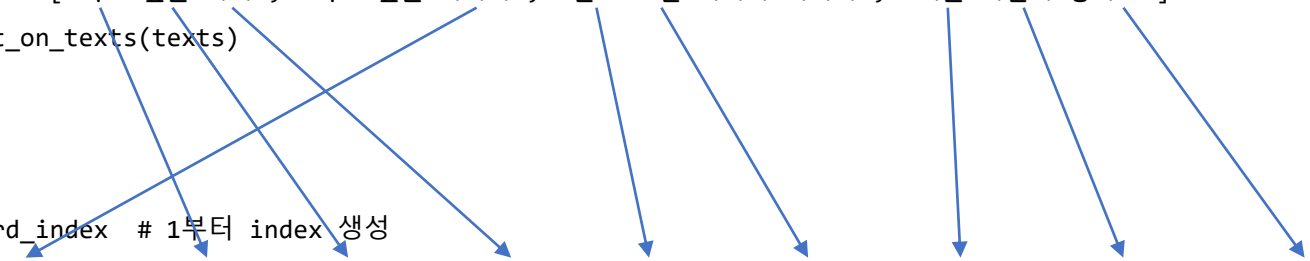
- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)
```

Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)

t.word_index  # 1부터 index 생성
>> {'바나나': 1, '먹고': 2, '싶은': 3, '사과': 4, '길고': 5, '노란': 6, '저는': 7, '과일이': 8, '좋아요': 9}
```



Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)
```

```
t.word_index
>> {'바나나': 1, '먹고': 2, '싶은': 3, '사과': 4, '길고': 5, '노란': 6, '저는': 7, '과일이': 8, '좋아요': 9}
```

'count', 'binary', 'tfidf','freq' 로
총 4개의 모드를 지원

```
t.texts_to_matrix(texts, mode = 'count')
>> [[0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
    [0. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
    [0. 2. 0. 0. 0. 1. 1. 0. 0. 0.]
    [0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]]
```

Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)
```

```
t.word_index
>> {'바나나': 1, '먹고': 2, '싶은': 3, '사과': 4, '길고': 5, '노란': 6, '저는': 7, '과일이': 8, '좋아요': 9}
```

주의할 점은 각 단어에 부여되는 인덱스는 1부터 시작
반면에 완성되는 행렬의 인덱스는 0부터 시작

```
# 1번째 방법: count
t.texts_to_matrix(texts, mode = 'count') # 앞서 배운 문서 단어 행렬(DTM)을 생성, DTM에서의 인덱스는 앞서 확인한 word_index but 단어 순서 정보는 보존X
>> [[0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
    [0. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
    [0. 2. 0. 0. 0. 1. 1. 0. 0. 0.]
    [0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]]
```

Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)
```

```
t.word_index
>> {'바나나': 1, '먹고': 2, '싶은': 3, '사과': 4, '길고': 5, '노란': 6, '저는': 7, '과일이': 8, '좋아요': 9}
```

```
# 2번째 방법: binary
t.texts_to_matrix(texts, mode = 'binary') # 해당 단어가 존재에만 계산 → 해당 단어가 몇개인지 무시
>> [[0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
    [0. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
    [0. 1. 0. 0. 0. 1. 1. 0. 0. 0.]
    [0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]]
```

Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)
```

```
t.word_index
>> {'바나나': 1, '먹고': 2, '싶은': 3, '사과': 4, '길고': 5, '노란': 6, '저는': 7, '과일이': 8, '좋아요': 9}
```

```
# 3번째 방법: tfidf
t.texts_to_matrix(texts, mode = 'tfidf').round(2) # TF-IDF 행렬 생성
>> [[0.   0.   0.85 0.85 1.1  0.   0.   0.   0.   0.  ]
    [0.   0.85 0.85 0.85 0.   0.   0.   0.   0.   0.  ]
    [0.   1.43 0.   0.   0.   1.1  1.1  0.   0.   0.  ]
    [0.   0.   0.   0.   0.   0.   0.   1.1  1.1  1.1 ]]
```

Keras의 texts_to_matrx()

- 입력된 텍스트 데이터로부터 행렬(matrix)를 만드는 도구

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
texts = ['먹고 싶은 사과', '먹고 싶은 바나나', '길고 노란 바나나 바나나', '저는 과일이 좋아요']
t.fit_on_texts(texts)
```

```
t.word_index
>> {'바나나': 1, '먹고': 2, '싶은': 3, '사과': 4, '길고': 5, '노란': 6, '저는': 7, '과일이': 8, '좋아요': 9}
```

```
# 4번째 방법: freq
t.texts_to_matrix(texts, mode = 'freq').round(2) # 각 문서에서의 각 단어의 등장 횟수 / 각 문서의 크기(각 문서에서 등장한 모든 단어의 개수의 총 합)
>> [[0.   0.   0.33 0.33 0.33 0.   0.   0.   0.   0.  ]
    [0.   0.33 0.33 0.33 0.   0.   0.   0.   0.   0.  ]
    [0.   0.5  0.   0.   0.   0.25 0.25 0.   0.   0.  ]
    [0.   0.   0.   0.   0.   0.   0.   0.33 0.33 0.33]]
```

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
%matplotlib inline
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
```

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
%matplotlib inline
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

newsdata = fetch_20newsgroups(subset = 'train') # 'train'을 기재하면 훈련 데이터만 리턴
```

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
%matplotlib inline
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

newsdata = fetch_20newsgroups(subset = 'train') # 'train'을 기재하면 훈련 데이터만 리턴
```

```
print(newsdata.keys())  # 실제로 훈련에 사용할 속성은 이메일 본문인 data와 메일이 어떤 주제인지 기재된 숫자 레이블인 target
>> dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])

print('훈련용 샘플의 개수 : {}'.format(len(newsdata.data)))
>> 훈련용 샘플의 개수 : 11314

print('총 주제의 개수 : {}'.format(len(newsdata.target_names)))
>> 총 주제의 개수 : 20

print(newsdata.target_names)
>> ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x',
'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med',
'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

```
print('첫번째 샘플의 레이블 : {}'.format(newsdata.target[0]))
>> 첫번째 샘플의 레이블 : 7

print('7번 레이블이 의미하는 주제 : {}'.format(newsdata.target_names[7]))
>> 7번 레이블이 의미하는 주제 : rec.autos
```

```
print('첫번째 샘플의 레이블 : {}'.format(newsdata.target[0]))
>> 첫번째 샘플의 레이블 : 7

print('7번 레이블이 의미하는 주제 : {}'.format(newsdata.target_names[7]))
>> 7번 레이블이 의미하는 주제 : rec.autos
```

```
print(newsdata.data[0]) # 첫번째 샘플 출력
>> From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

```
data = pd.DataFrame(newsdata.data, columns = ['email']) # data로부터 데이터프레임 생성
data['target'] = pd.Series(newsdata.target) # target 열 추가 → 메일 본문에 해당하는 email열과 레이블에 해당되는 target 열, 총 2개의 열로 구성
data[:5] # 상위 5개 행을 출력
```

>>

	email	target
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	4
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14

```
data = pd.DataFrame(newsdata.data, columns = ['email']) # data로부터 데이터프레임 생성
data['target'] = pd.Series(newsdata.target) # target 열 추가 → 메일 본문에 해당하는 email열과 레이블에 해당되는 target 열, 총 2개의 열로 구성
data[:5] # 상위 5개 행을 출력
```

>>

	email	target
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	4
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14

```
data.info()
>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 11314 entries, 0 to 11313
Data columns (total 2 columns):
news       11314 non-null object
target     11314 non-null int32
dtypes: int32(1), object(1)
memory usage: 132.7+ KB
```

```
data = pd.DataFrame(newsdata.data, columns = ['email']) # data로부터 데이터프레임 생성
data['target'] = pd.Series(newsdata.target) # target 열 추가 → 메일 본문에 해당하는 email열과 레이블에 해당되는 target 열, 총 2개의 열로 구성
data[:5] # 상위 5개 행을 출력
```

>>

	email	target
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	4
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14

```
data.info()
>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 11314 entries, 0 to 11313
Data columns (total 2 columns):
news       11314 non-null object
target     11314 non-null int32
dtypes: int32(1), object(1)
memory usage: 132.7+ KB
```

```
data.isnull().values.any()  # Null 값을 가진 샘플이 있는지 isnull().values.any()로 확인 가능
>> False
```

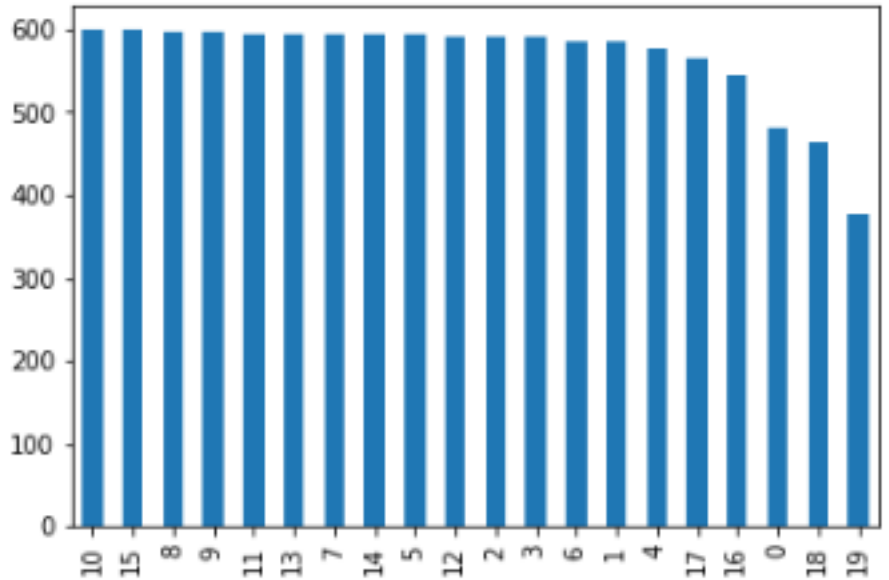
```
data = pd.DataFrame(newsdata.data, columns = ['email']) # data로부터 데이터프레임 생성
data['target'] = pd.Series(newsdata.target) # target 열 추가 → 메일 본문에 해당하는 email열과 레이블에 해당되는 target 열, 총 2개의 열로 구성
data[:5] # 상위 5개 행을 출력
```

>>

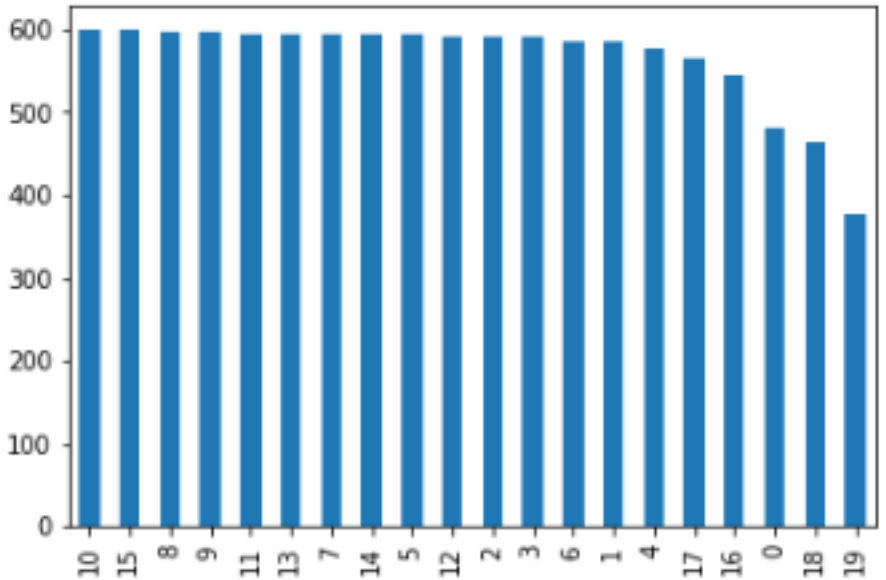
	email	target
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	4
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14

```
data.info()
>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 11314 entries, 0 to 11313
Data columns (total 2 columns):
news       11314 non-null object
target     11314 non-null int32
dtypes: int32(1), object(1)
memory usage: 132.7+ KB

data.isnull().values.any()  # Null 값을 가진 샘플이 있는지 isnull().values.any()로 확인 가능
>> False

print('중복을 제외한 샘플의 수 : {}'.format(data['email'].nunique()))
>> 중복을 제외한 샘플의 수 : 11314
print('중복을 제외한 주제의 수 : {}'.format(data['target'].nunique()))
>> 중복을 제외한 주제의 수 : 20
```

```
data['target'].value_counts().plot(kind='bar');  # 10번 레이블의 수가 가장 많고, 19번 레이블의 수가 가장 적음
```



```
data['target'].value_counts().plot(kind='bar');  # 10번 레이블의 수가 가장 많고, 19번 레이블의 수가 가장 적음
```




```
print(data.groupby('target').size().reset_index(name='count') )  # 대체적으로 400 ~ 600개 사이의 분포
>>  target  count
0        0    480
1        1    584

     ( 중 략 )

18      18    465
19      19    377
```

```
newsdata_test = fetch_20newsgroups(subset='test', shuffle=True) # 'test'를 기재하면 테스트 데이터만 리턴한다.
train_email = data['email'] # 훈련 데이터의 본문 저장
train_label = data['target'] # 훈련 데이터의 레이블 저장
test_email = newsdata_test.data # 테스트 데이터의 본문 저장
test_label = newsdata_test.target # 테스트 데이터의 레이블 저장

max_words = 10000 # 실습에 사용할 단어의 최대 개수 → 토크나이저를 사용하면 빈도수 순으로 인덱스를 부여, 빈도수가 가장 높은 상위 max_words 개수만큼의 단어를 사용
num_classes = 20 # 레이블의 수
```

```
newsdata_test = fetch_20newsgroups(subset='test', shuffle=True) # 'test'를 기재하면 테스트 데이터만 리턴한다.
train_email = data['email'] # 훈련 데이터의 본문 저장
train_label = data['target'] # 훈련 데이터의 레이블 저장
test_email = newsdata_test.data # 테스트 데이터의 본문 저장
test_label = newsdata_test.target # 테스트 데이터의 레이블 저장

max_words = 10000 # 실습에 사용할 단어의 최대 개수 → 토크나이저를 사용하면 빈도수 순으로 인덱스를 부여, 빈도수가 가장 높은 상위 max_words 개수만큼의 단어를 사용
num_classes = 20 # 레이블의 수
```

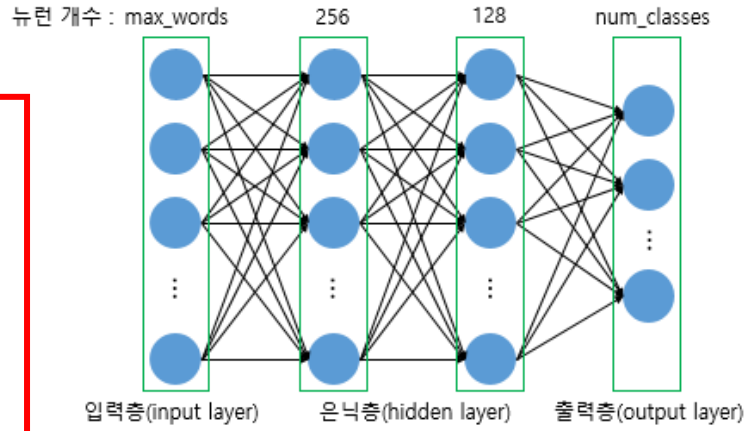
```
def prepare_data(train_data, test_data, mode): # 전처리 함수
    t = Tokenizer(num_words = max_words) # max_words 개수만큼의 단어만 사용한다.
    t.fit_on_texts(train_data)
    X_train = t.texts_to_matrix(train_data, mode=mode) # 샘플 수 × max_words 크기의 행렬 생성
    X_test = t.texts_to_matrix(test_data, mode=mode) # 샘플 수 × max_words 크기의 행렬 생성
    return X_train, X_test, t.index_word
```

```
newsdata_test = fetch_20newsgroups(subset='test', shuffle=True) # 'test'를 기재하면 테스트 데이터만 리턴한다.
train_email = data['email'] # 훈련 데이터의 본문 저장
train_label = data['target'] # 훈련 데이터의 레이블 저장
test_email = newsdata_test.data # 테스트 데이터의 본문 저장
test_label = newsdata_test.target # 테스트 데이터의 레이블 저장

max_words = 10000 # 실습에 사용할 단어의 최대 개수 → 토크나이저를 사용하면 빈도수 순으로 인덱스를 부여, 빈도수가 가장 높은 상위 max_words 개수만큼의 단어를 사용
num_classes = 20 # 레이블의 수

def prepare_data(train_data, test_data, mode): # 전처리 함수
    t = Tokenizer(num_words = max_words) # max_words 개수만큼의 단어만 사용한다.
    t.fit_on_texts(train_data)
    X_train = t.texts_to_matrix(train_data, mode=mode) # 샘플 수 × max_words 크기의 행렬 생성
    X_test = t.texts_to_matrix(test_data, mode=mode) # 샘플 수 × max_words 크기의 행렬 생성
    return X_train, X_test, t.index_word

X_train, X_test, index_to_word = prepare_data(train_email, test_email, 'binary') # binary 모드로 변환
y_train = to_categorical(train_label, num_classes) # 원-핫 인코딩
y_test = to_categorical(test_label, num_classes) # 원-핫 인코딩
```

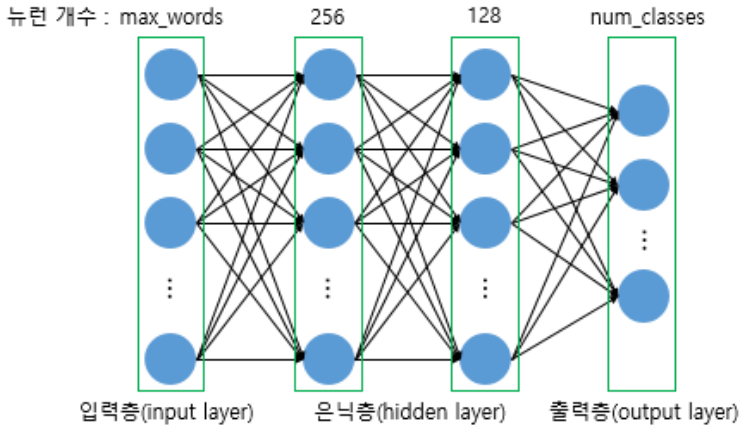
```
newsdata_test = fetch_20newsgroups(subset='test', shuffle=True) # 'test'를 기재하면 테스트 데이터만 리턴한다.
train_email = data['email'] # 훈련 데이터의 본문 저장
train_label = data['target'] # 훈련 데이터의 레이블 저장
test_email = newsdata_test.data # 테스트 데이터의 본문 저장
test_label = newsdata_test.target # 테스트 데이터의 레이블 저장

max_words = 10000 # 실습에 사용할 단어의 최대 개수 → 토크나이저를 사용하면 빈도수 순으로 인덱스를 부여, 빈도수가 가장 높은 상위 max_words 개수만큼의 단어를 사용
num_classes = 20 # 레이블의 수

def prepare_data(train_data, test_data, mode): # 전처리 함수
    t = Tokenizer(num_words = max_words) # max_words 개수만큼의 단어만 사용한다.
    t.fit_on_texts(train_data)
    X_train = t.texts_to_matrix(train_data, mode=mode) # 샘플 수 × max_words 크기의 행렬 생성
    X_test = t.texts_to_matrix(test_data, mode=mode) # 샘플 수 × max_words 크기의 행렬 생성
    return X_train, X_test, t.index_word

X_train, X_test, index_to_word = prepare_data(train_email, test_email, 'binary') # binary 모드로 변환
y_train = to_categorical(train_label, num_classes) # 원-핫 인코딩
y_test = to_categorical(test_label, num_classes) # 원-핫 인코딩

# 메일 본문의 크기가 샘플의 수 × 10,000의 행렬로 변환
# 열의 개수가 10,000인 것은 위의 prepard_data 함수 내부에서 Tokenizer의 num_words의 인자로 max_words를 지정해주었기 때문
print('훈련 샘플 본문의 크기 : {}'.format(X_train.shape))
print('훈련 샘플 레이블의 크기 : {}'.format(y_train.shape))
>> 훈련 샘플 본문의 크기 : (11314, 10000)
>> 훈련 샘플 레이블의 크기 : (11314, 20)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Dense(256, input_shape=(max_words,), activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1, validation_split=0.1)
    score = model.evaluate(X_test, y_test, batch_size=128, verbose=0)
    return score[1]
```



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Dense(256, input_shape=(max_words,), activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1, validation_split=0.1)
    score = model.evaluate(X_test, y_test, batch_size=128, verbose=0)
    return score[1]
```




```
modes = ['binary', 'count', 'tfidf', 'freq'] # 4개의 모드를 리스트에 저장.
for mode in modes: # 4개의 모드에 대해서 각각 아래의 작업을 반복한다.
    X_train, X_test, _ = prepare_data(train_email, test_email, mode) # 모드에 따라서 데이터를 전처리
    score = fit_and_evaluate(X_train, y_train, X_test, y_test) # 모델을 훈련하고 평가.
    print(mode+' 모드의 테스트 정확도:', score)

>> binary 모드의 테스트 정확도: 0.8312533
>> count 모드의 테스트 정확도: 0.8239511
>> tfidf 모드의 테스트 정확도: 0.8381572
>> freq 모드의 테스트 정확도: 0.6902549  → 대체적으로 82% ~ 83%의 비슷한 정확도를 보이는데, 'freq' 모드에서만 정확도가 69%
```

1. 기존 N-gram 언어 모델의 한계

- 언어 모델(Language Model): 문장에 확률을 할당하는 모델
- 언어 모델링(Language Modeling): 주어진 문맥으로부터 아직 모르는 단어를 예측하는 것

[An adorable little boy is spreading] ____

n-gram 언어 모델

- 언어 모델링에 바로 앞의 n-1개의 단어만 참고
- 4-gram 언어 모델이라고 가정

~~An adorable little~~ boy is spreading ____

무시됨 n-1개의 단어

n-gram 언어 모델

- 언어 모델링에 바로 앞의 n-1개의 단어만 참고
- 4-gram 언어 모델이라고 가정

~~An adorable little~~ boy is spreading ____

무시됨 n-1개의 단어

- 훈련 코퍼스에서 (n-1)-gram을 카운트한 것을 분모로, n-gram을 카운트한 것을 분자로 하여 다음 단어가 등장할 확률을 예측

$$P(w|\text{boy is spreading}) = \frac{\text{count(boy is spreading } w)}{\text{count(boy is spreading)}}$$

boy is spreading: 1,000번
boy is spreading insults: 500번
boy is spreading smiles: 200번 등장

$$P(\text{insults}|\text{boy is spreading}) = 0.500$$

$$P(\text{smiles}|\text{boy is spreading}) = 0.200$$

n-gram 언어 모델

- 언어 모델링에 바로 앞의 n-1개의 단어만 참고
- 4-gram 언어 모델이라고 가정

희소 문제(sparsity problem)
n-gram 언어 모델은 충분한 데이터를 관측하지 못하면 언어를 정확히 모델링하지 못함

~~An adorable little~~ boy is spreading ____

무시됨 n-1개의 단어

- 훈련 코퍼스에서 (n-1)-gram을 카운트한 것을 분모로, n-gram을 카운트한 것을 분자로 하여 다음 단어가 등장할 확률을 예측

$$P(w|\text{boy is spreading}) = \frac{\text{count(boy is spreading } w)}{\text{count(boy is spreading)}}$$

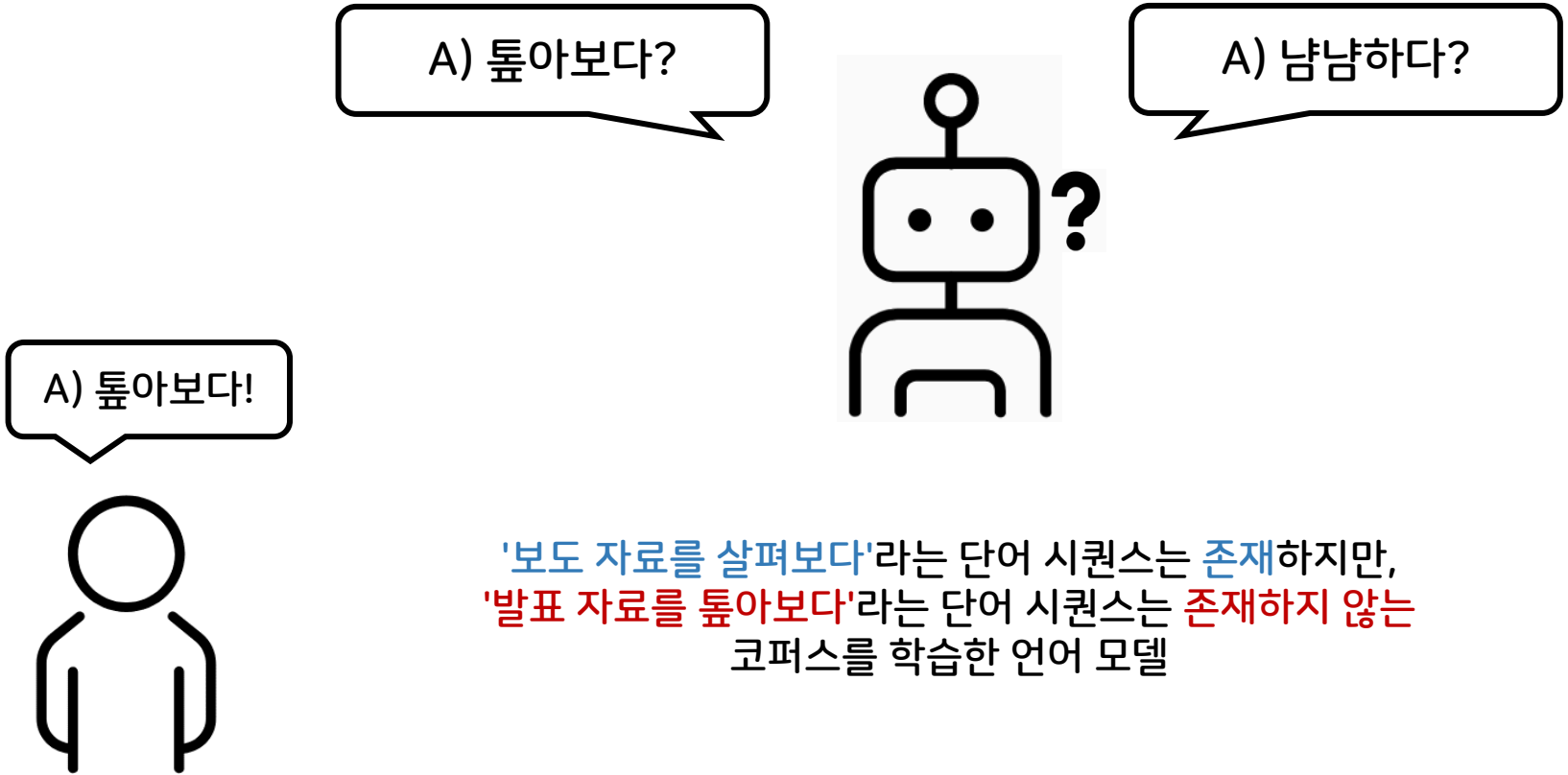
boy is spreading: 1,000번
boy is spreading insults: 500번
boy is spreading smiles: 200번 등장

$$P(\text{insults}|\text{boy is spreading}) = 0.500$$

$$P(\text{smiles}|\text{boy is spreading}) = 0.200$$

- 희소문제는 기계가 단어 간 유사도를 알 수 있다면 해결할 수 있는 문제

Q. 보도 자료를 _____ .

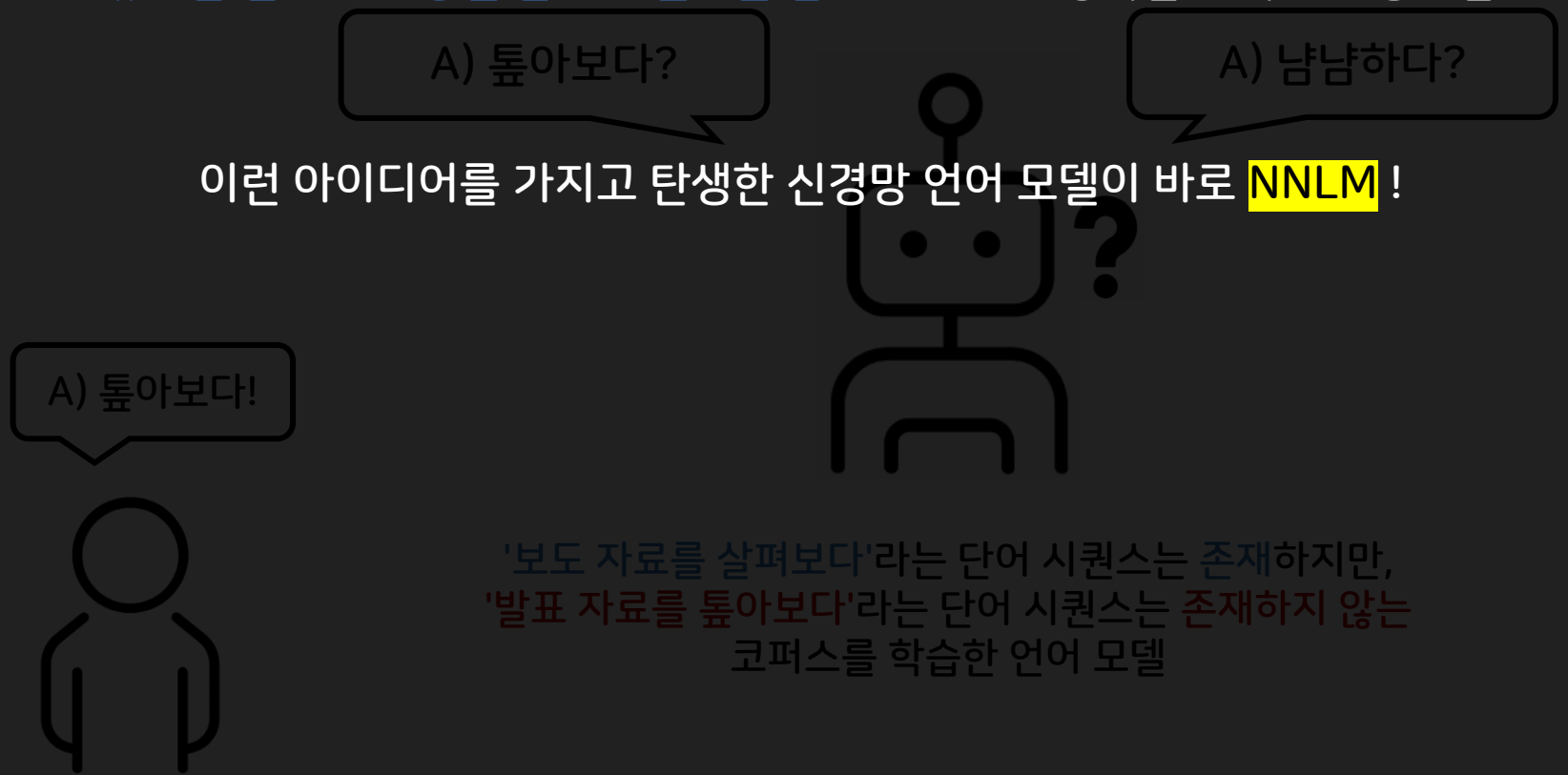


'보도 자료를 살펴보다'라는 단어 시퀀스는 존재하지만,
'발표 자료를 톺아보다'라는 단어 시퀀스는 존재하지 않는
코퍼스를 학습한 언어 모델

- 희소문제는 기계가 단어 간 유사도를 알 수 있다면 해결할 수 있는 문제

Q. 보도 자료를 _____ .

훈련 코퍼스에 없는 단어 시퀀스에 대한 예측이라도
유사한 단어가 사용된 단어 시퀀스를 참고하여 보다 정확한 예측이 가능해짐



이런 아이디어를 가지고 탄생한 신경망 언어 모델이 바로 NNLM !

'보도 자료를 살펴보다'라는 단어 시퀀스는 존재하지만,
'발표 자료를 톺아보다'라는 단어 시퀀스는 존재하지 않는
코퍼스를 학습한 언어 모델

NNLM이 언어 모델링을 학습하는 과정을 살펴보자!

예시 what will the fat cat sit on

1. 원-핫 인코딩

- 가장 먼저 해야 할 일은 기계가 단어를 인식할 수 있도록 모든 단어를 숫자로 인코딩 하는 것
- 훈련 코퍼스에 7개의 단어만 존재한다고 가정

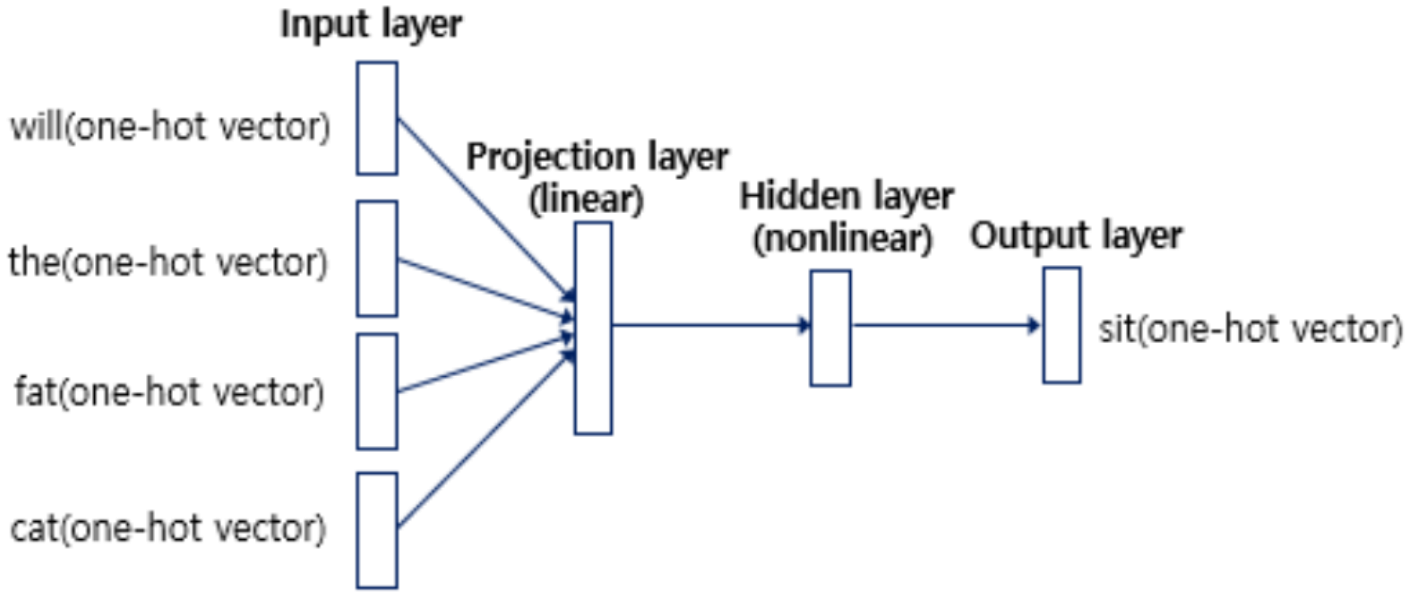
원-핫 인코딩 결과

```
What = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]
```

- 모든 단어가 단어 집합(vocabulary)의 크기인 7의 차원을 가지는 원-핫 벡터로 인코딩 됨
- 원-핫 벡터들은 훈련을 위한 NNLM의 입력이면서 예측을 위한 레이블
- NNLM은 n-gram 언어 모델과 유사하게 다음 단어를 예측할 때 앞의 모든 단어를 참고하는 것이 아니라 정해진 n개의 단어만을 참고

NNLM의 구조

- 4개의 층(layer)으로 이루어진 인공 신경망

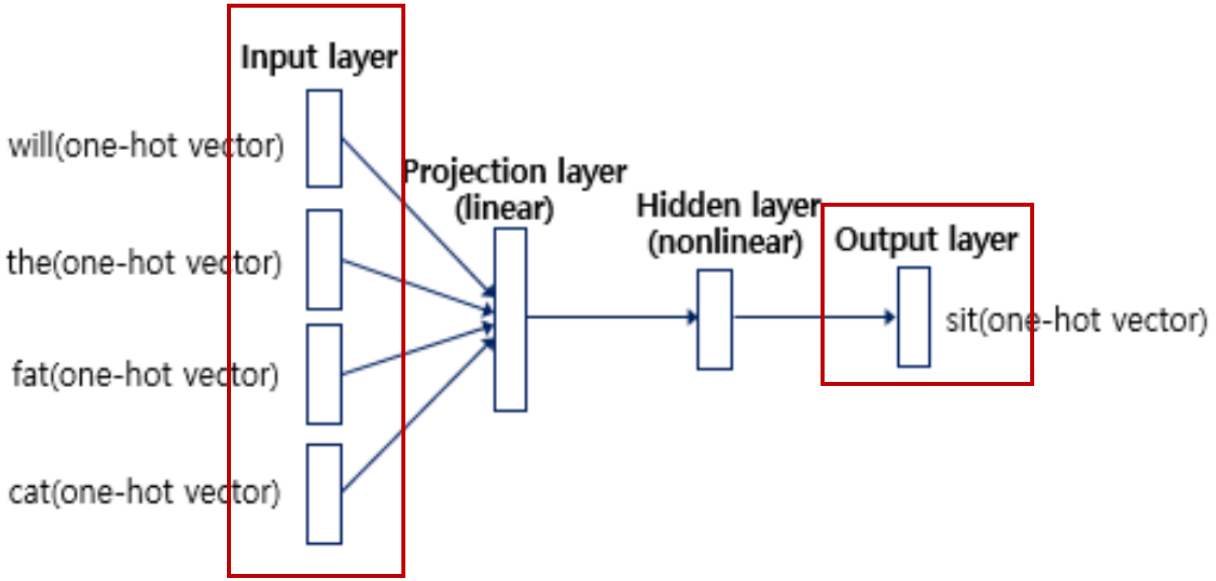


1) 입력층 (Input Layer)

- 윈도우의 크기는 4로 정하였으므로 입력은 4개의 단어 'will, the, fat, cat'의 원-핫 벡터

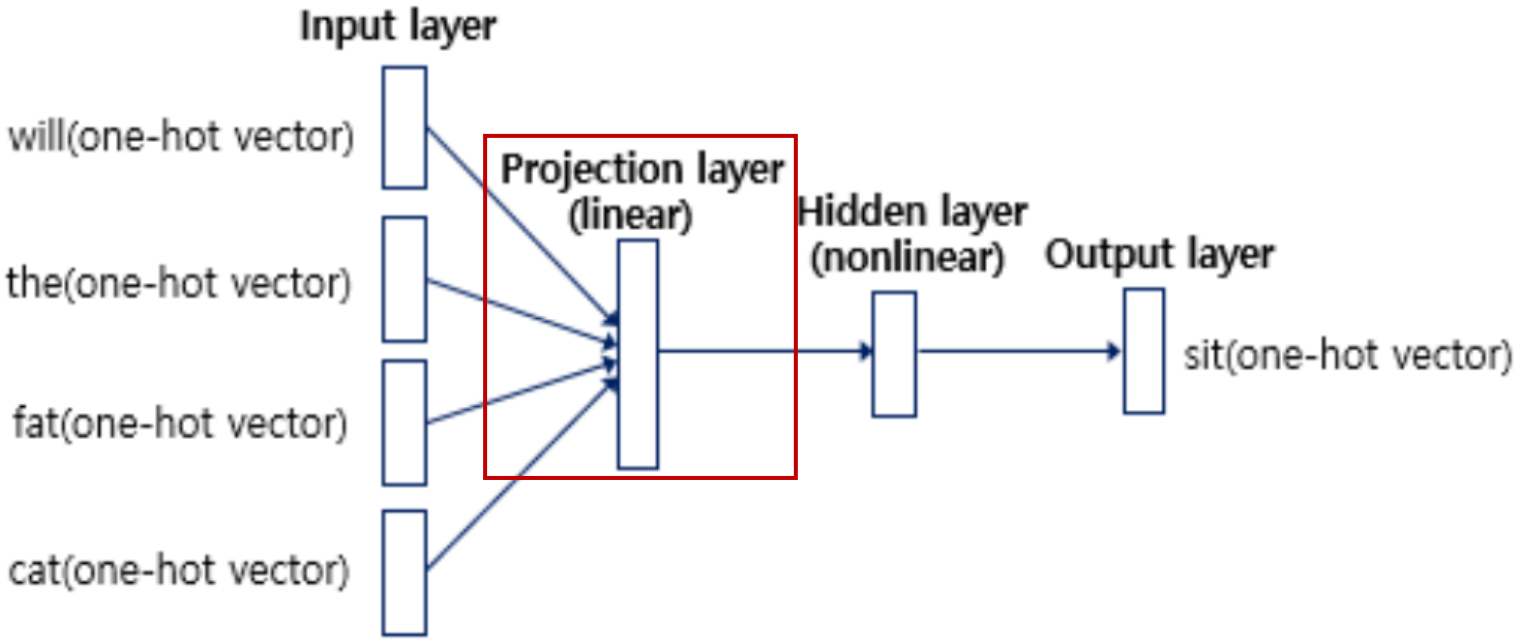
2) 출력층 (Output Layer)

- 모델이 예측해야 하는 정답에 해당되는 단어 sit의 원-핫 벡터는 출력층에서 모델이 예측한 값의 오차를 구하기 위해 사용될 예정
- 이 오차로부터 손실 함수를 사용해 인공 신경망이 학습



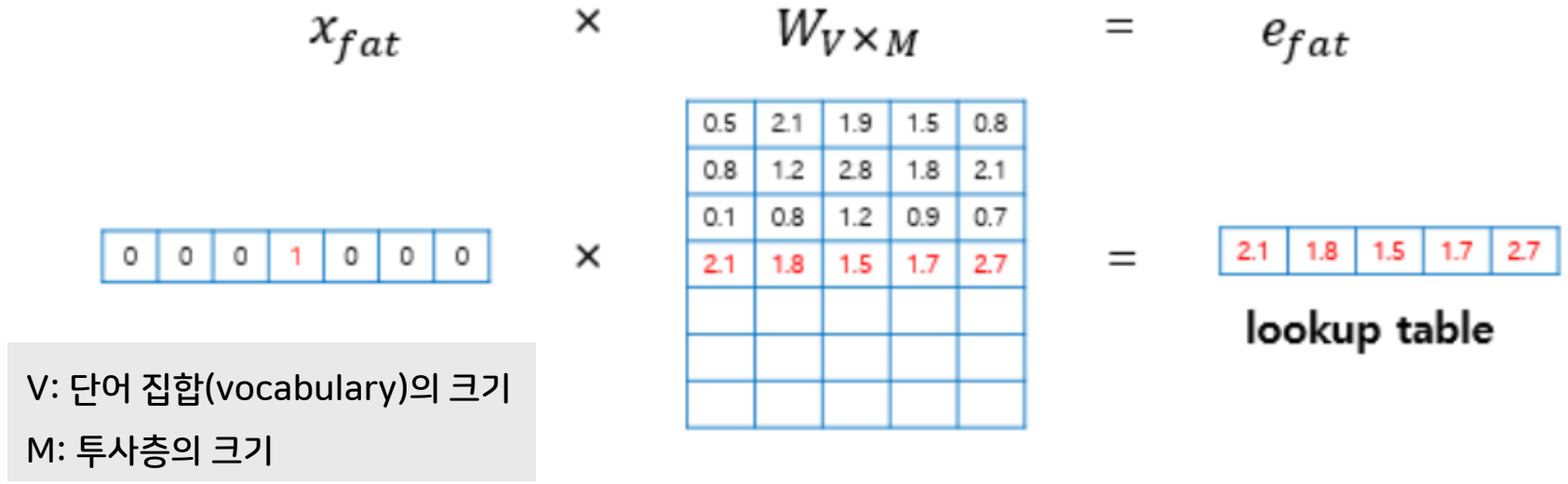
3) 투사층 (Projection Layer)

- 인공 신경망에서 입력층과 출력층 사이의 층을 보통 은닉층이라고 부름
- 투사층이 일반 은닉층과 구별되는 특징은 가중치 행렬과의 연산은 이루어지지만 활성화 함수가 존재하지 않는다는 것



3) 투사층 (Projection Layer)

- 원-핫 벡터와 가중치 W 행렬의 곱은 사실 W행렬의 i번째 행을 그대로 읽어오는 것과(lookup) 동일
- 이 룩업 테이블 작업을 거치면 V의 차원을 가지는 원-핫 벡터는 이보다 더 차원이 작은 M차원의 단어 벡터로 맵핑
- 테이블 룩업 과정을 거친 후의 단어 벡터는 e_fat으로 표현
- 이 벡터들은 초기에는 랜덤한 값을 가지지만 학습 과정에서 값이 계속 변경되는데 이 단어 벡터를 임베딩 벡터라고 함

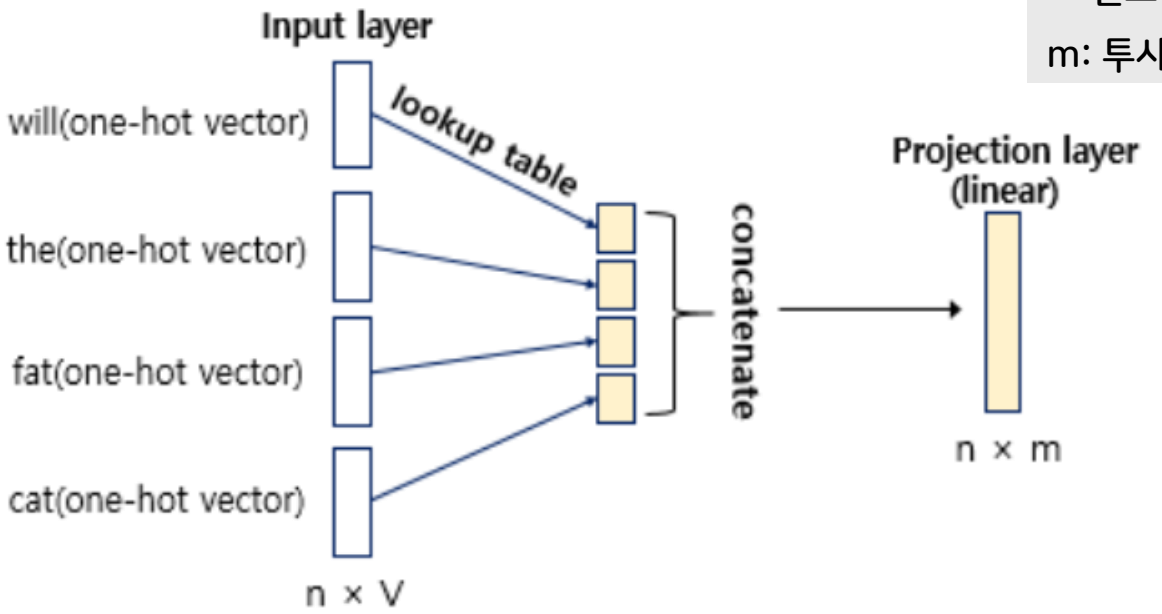


V: 단어 집합(vocabulary)의 크기
M: 투사층의 크기

3) 투사층 (Projection Layer)

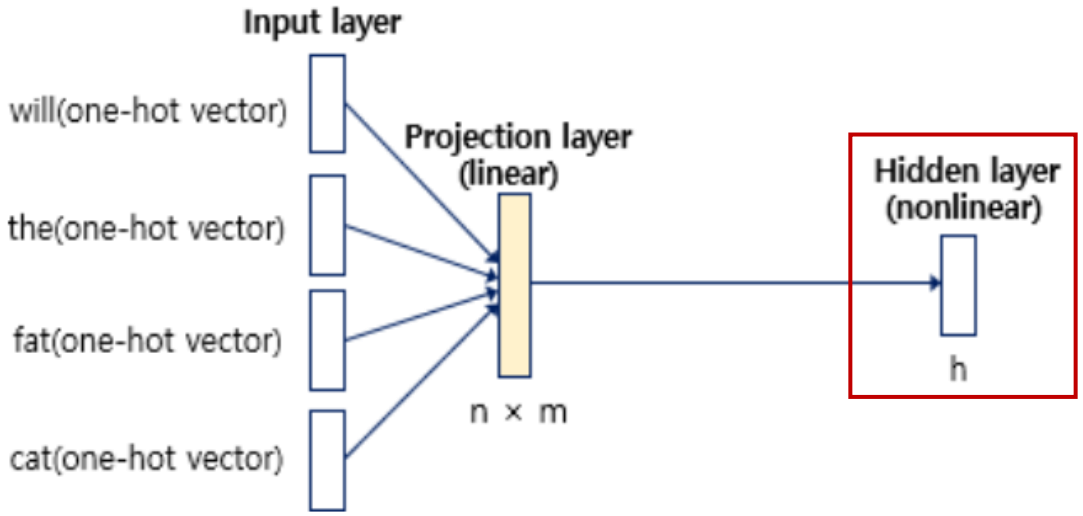
- 각 단어가 테이블 룩업을 통해 임베딩 벡터로 변경되고, 투사층에서 모든 임베딩 벡터들의 값은 연결(concatenation)
- 일반적인 은닉층이 활성화 함수를 사용하는 비선형층(nonlinear layer)
- 투사층은 활성화 함수가 존재하지 않는 선형층(linear layer)

V: 단어 집합(vocabulary)의 크기
n: 윈도우의 크기
m: 투사층의 크기



4) 은닉층 (Hidden Layer)

- 일반적인 피드 포워드 신경망에서 은닉층을 지난다는 것은 은닉층의 입력은 가중치 곱해진 후 편향이 더해져 활성화 함수의 입력이 된다는 의미
- 이때의 가중치와 편향을 W_h와 b_h이라고 하고, 은닉층의 활성화 함수를 하이퍼볼릭탄젠트 함수



n: 윈도우의 크기
m: 투사층의 크기
h: 은닉층의 크기

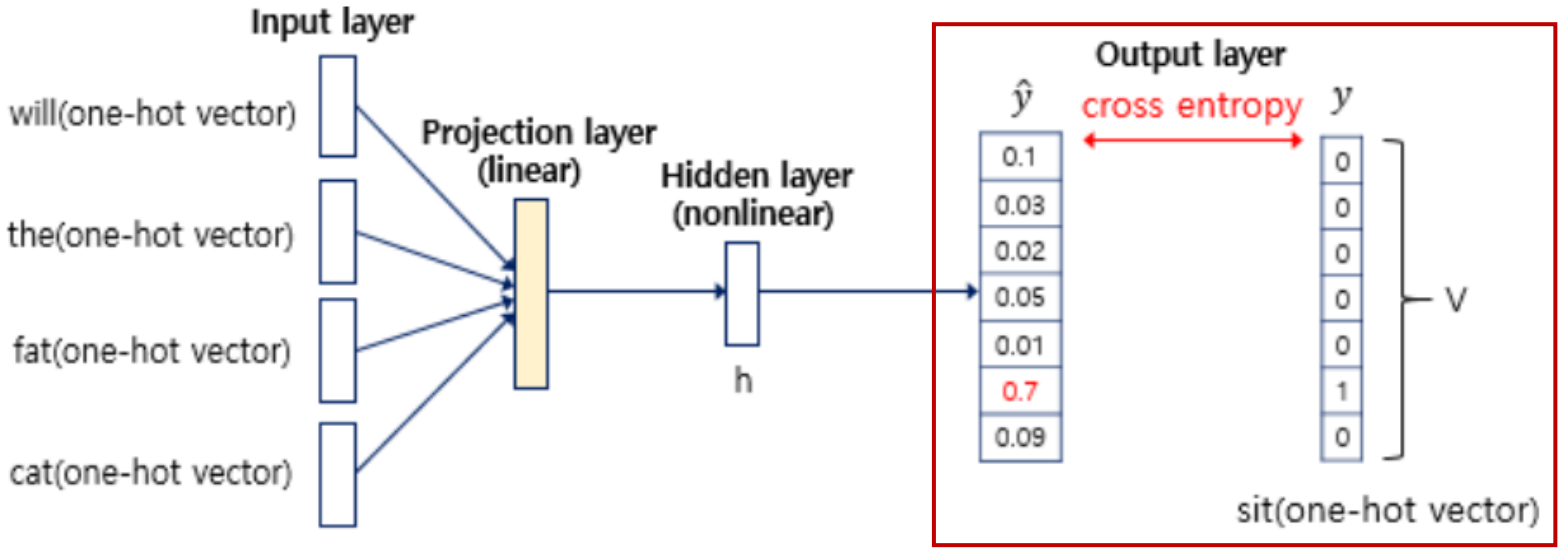
$$\text{은닉층} : h^{layer} = tanh(W_h p^{layer} + b_h)$$

4) 은닉층 (Hidden Layer)

h: 은닉층의 크기
y: 실제값에 해당하는 단어
y^: NNLM의 예측값

- 은닉층의 출력은 V의 크기를 가지는 출력층으로 향함
- 입력 벡터의 차원이 7이었다면 여기서 나오는 벡터도 7차원
- 출력층에서는 활성화 함수로 소프트맥스(softmax) 함수를 사용
- V차원의 벡터는 소프트맥스 함수를 지나면서 각 원소는 0과 1사이의 실수 값을 가지며 총 합은 1이 되는 상태로 바뀜
- 이렇게 나온 벡터를 NNLM의 예측값이라는 의미에서 y^라고 함

출력층 : $\hat{y} = softmax(W_y h^{layer} + b_y)$



만약 충분한 훈련 데이터가 있다는 가정 하에 NNLM이 얻을 수 있는 이점은 무엇일까?

NNLM의 핵심은 충분한 양의 훈련 코퍼스를 위와 같은 과정으로 학습한다면 결과적으로 수많은 문장에서 유사한 목적으로 사용되는 단어들은 결국 유사한 임베딩 벡터 값을 얻게 되는 것에 있습니다.

이렇게 되면 훈련이 끝난 후 다음 단어를 예측 과정에서 훈련 코퍼스에서 없던 단어 시퀀스라고 하더라도 다음 단어를 선택할 수 있습니다.

단어 간 유사도를 구할 수 있는 임베딩 벡터의 아이디어는 Word2Vec, FastText, GloVe 등으로 발전되어서 딥 러닝 모델에서는 필수적으로 사용되는 방법이 되었습니다.

1) 기존 모델에서의 개선점

- NNLM은 단어의 유사도를 단어를 표현하기 위해 밀집 벡터(dense vector)를 사용하여 단어의 유사도를 표현하여 희소 문제(sparsity problem) 해결

> 밀집 벡터(dense vector) : 벡터의 원소들이 실수값을 가지면서, 원-핫 벡터보다 저차원을 가지는 벡터
>
> 희소 벡터(sparse vector): 원-핫 벡터와 같이 벡터의 원소값이 대부분이 0인 벡터

- 더 이상 모든 n-gram을 저장하지 않아도 된다는 점에서 n-gram 언어 모델보다 저장 공간의 이점을 가짐

2) 고정된 길이의 입력

- n-gram 언어 모델과 마찬가지로 다음 단어를 예측하기 위해 모든 이전 단어를 참고하는 것이 아니라, 정해진 n개의 단어만을 참고하기 때문에 버려지는 단어들이 가진 문맥 정보는 참고할 수 없음

감사합니다 ☺