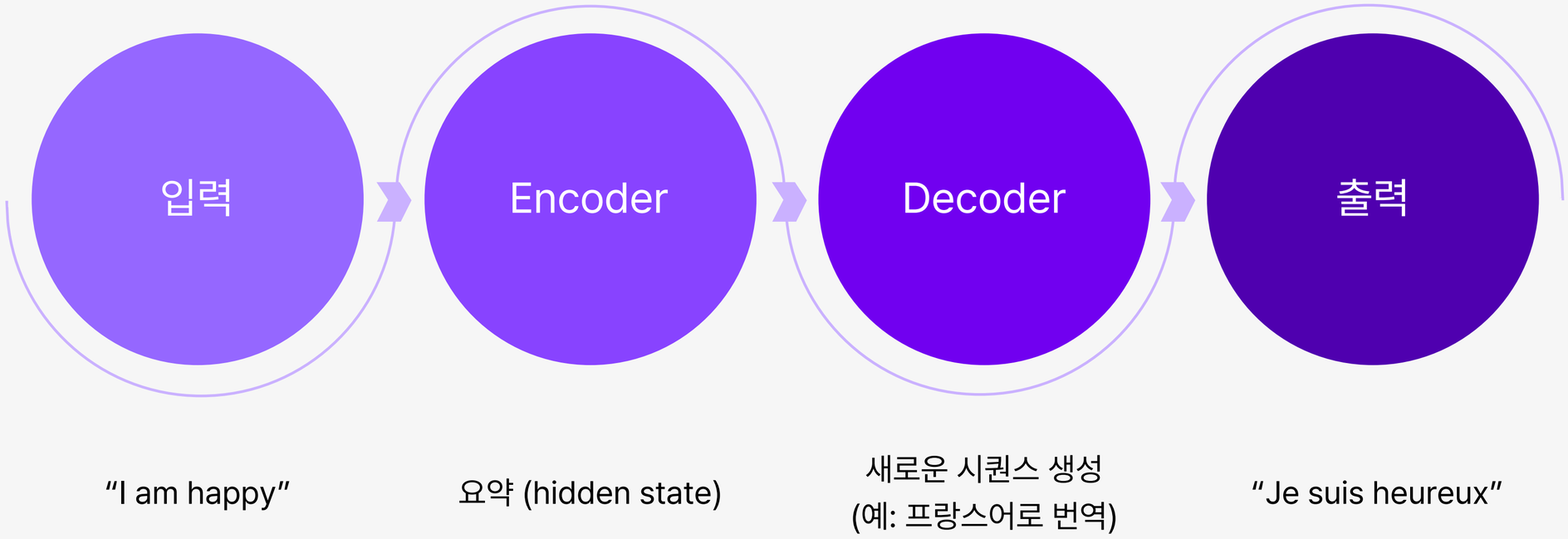
KDT 자율주행트랙

실습에서 사용되는 개념 정리

[참고 자료]

Encoder-Decoder 구조란?

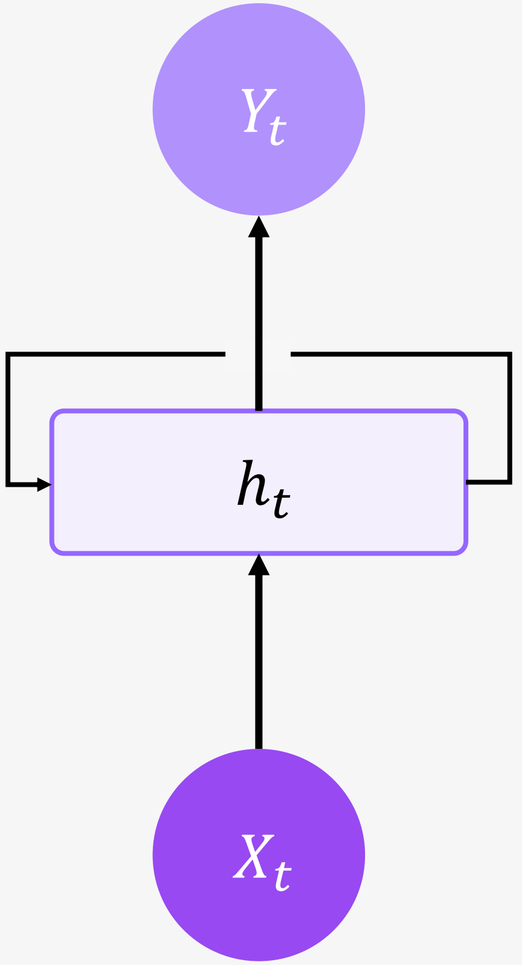
하나의 입력을 받아 요약(인코딩)하고, 그 요약을 기반으로 새로운 출력을 생성(디코딩)하는 구조



RNN 이란?

Recurrent Neural Network

- 시퀀스 데이터를 처리하는 신경망
- SimpleRNN은 그 중 가장 기본적인 RNN 구조로, 연속된 데이터를 처리함
  - 예: 단어 시퀀스, 시간 데이터 등



Embedding

단어(또는 정수)를 고정된 크기의 dense vector로 변환해주는 레이어

구성 요소	설명
`input_dim`	단어 집합 크기 사용할 단어의 개수
`output_dim`	임베딩 차원 출력 벡터의 차원
`input_length`	입력 시퀀스의 길이

Embedding Layer 예시 코드

- **원-핫 인코딩**은 희소(sparse)하고 차원이 커져 계산 비효율적임
- **Embedding**은 각 단어를 **의미 기반 벡터로 압축**해서 표현
- 학습 과정에서 벡터의 의미가 **자동으로 학습됨**

- **모델 구성 예시**
  - input_dim : 1000 (0부터 999까지 총 1000개의 인덱스 사용)
  - output_dim : 64 (각 단어를 64차원 벡터로 변환)
  - input_length : 10 (각 문장은 10개의 정수로 구성됨)

```
# 라이브러리 import
import tensorflow as tf
from tensorflow.keras.layers import Embedding


# 모델 구성
embedding_layer = Embedding(
    input_dim = 1000,
    output_dim = 64
    input_length = 10
)
```

SimpleRNN

TensorFlow(Keras)에서 제공하는 Vanilla RNN의 구현 클래스 이름

구성 요소	설명
`units`	RNN 셀의 은닉 상태(hidden state) 크기 출력 차원
`activation`	RNN 내부에서 사용하는 상태 벡터의 크기(기본 : tahn)
`use_bias`	bias 사용 여부 (기본 : True)
`dropout`	입력에 적용할 드롭아웃 비율
`input_shape`	(timesteps, features) 형태의 전체 입력 구조
`return_state`	마지막 hidden state 반환 여부 True이면 (출력, 상태) 형태로 반환 Encoder에서 주로 사용
`return_sequences`	전체 시퀀스 출력 여부 True면 모든 시점 출력, False면 마지막 시점만 Decoder에서 주로 사용

SimpleRNN을 활용한 모델 구성

- `model=Sequential()`을 통해
  내부적으로 **빈 레이어 리스트를 가진 모델**을 생성
- 그 다음 `model.add(...)`를 통해 순서대로 Layer 하나씩 추가
- 모델 구성
  - units (hidden state의 크기) : 16
  - input_shape : (10, 1)

```
# 라이브러리 import
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import SimpleRNN


# 모델 구성
model = Sequential()
model.add(SimpleRNN(16, input_shape = (10, 1)))
```

Dense

모든 입력 노드가 모든 출력 노드와 연결되는 신경망의 기본 구조

구성 요소	설명
`units`	출력 뉴런(노드)의 수
`activation`	출력에 적용할 활성화 함수
`use_bias`	bias 사용 여부 (기본 : True)
`kernel_initializer`	가중치 초기화 방식
`bias_initializer`	bias 초기화 방식
`input_shape`	입력 형태 (Sequential 모델의 첫 층일 때 필요)

Dense Layer 활용 예시

- `model=Sequential()`을 통해
  내부적으로 **빈 레이어 리스트를 가진 모델**을 생성
- 그 다음 `model.add(...)`를 통해 순서대로 Layer 하나씩 추가
- SimpleRNN 구성
  - units (hidden state의 크기) : 16
  - input_shape : (10, 1)
- Dense 구성
  - 출력 노드 개수 : 1
  - activation (활성화 함수) : softmax

```
# 라이브러리 import
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense


# 모델 구성
model = Sequential()
model.add(SimpleRNN(16, input_shape = (10, 1)))
model.add(Dense(1, activation = “softmax”))
```

Encoder-Decoder 구현

- **Encoder 구성**
  - units (hidden state의 크기) : 16
  - return_state : True
  - input_shape : (10, 1)
- **Decoder 구성**
  - units (hidden state의 크기) : 16
  - return_sequences : True
  - input_shape : (30, 1)
- **Dense 구성**
  - 출력 노드 개수 : 5
  - activation (활성화 함수) : softmax

```
# 라이브러리 import
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

class myModel(Model) :
      def __init__(self) :
            super.__init__()
            # Encoder 구성
            self.encoder = SimpleRNN(
                  16, return_state = True, input_shape=(10, 1))

            # Decoder 구성
            self.decoder = SimpleRNN(
                  16, return_sequences = True, input_shape=(30, 1)))

            # Dense 구성
            self.dense = Dense(5, activation = “softmax”)
```

Encoder-Decoder 구현

call 메서드 구현

- encoder_input : 인코더에 들어가는 입력
- decoder_input : 디코더에 들어가는 입력
- Encoder 실행
  - 반환값 : 전체 출력 시퀀스, 마지막 hidden state

    → encoder의 return_state=True로 설정했기 때문
- Decoder 실행
  - initial_state : hidden_state

    → encoder의 마지막 hidden state를 사용

    → initial_state는 **list 형식**으로 전달되어야 함

```
# call 메서드 구현
def call(self, encoder_input, decoder_input):
     # Encoder 실행
     _, hidden_state = self.encoder(encoder_input)

     # Decoder 실행
     decoder_output = self.decoder(
          decoder_input, initial_state=[hidden_state])

     # 단어 예측, 분류, 회귀 등 최종 출력이 필요한 경우
     # Dense Layer 필요
     outputs = self.dense(decoder_output)

     return outputs
```

모델 학습 세팅

compile()

모델을 학습시키기 위해 최적화 방법, 손실 함수, 평가지표 등을 설정하는 함수

구성 요소	설명
`optimizer`	최적화 방법 가중치를 업데이트하는 방법 (학습 방식)
`loss`	손실 함수 모델이 얼마나 틀렸는지 계산하는 함수
`metrics`	평가 지표 입력 형태 (Sequential 모델의 첫 층일 때 필요)
`(optimizer 설정)` `learning_rate`	학습률 가중치를 얼마나 빠르게 업데이트할지를 결정

모델 학습 세팅 예시

- **optimizer** : Adam
  - **learning_rate** : 0.005
- **loss** : Mean Squared Error (MSE)
- **metrics** : accuracy (정확도)

```
# 라이브러리 import
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

# 모델 학습 세팅
model.compile(
    optimizer = Adam(0.005),
    loss = 'mse',
    metrics = [‘accuracy']
)
```

모델 학습 실행

fit()

입력 데이터와 정답을 주고, 설정한 epoch 수만큼 모델을 반복 학습시킴

구성 요소	설명
x	입력 데이터 (features)
y	정답 데이터 (labels)
batch_size	몇 개씩 데이터를 묶어 학습할지
epochs	전체 데이터셋을 몇 번 반복 학습할지
shuffle	매 epoch마다 데이터를 섞을지
validation_data	검증 데이터 (x_val, y_val)
verbose	출력 로그 형태 (0 : 출력 X, 1 : 진행 바 형태로 출력, 2 : epoch별 간략한 로그 출력)

모델 학습 실행 예시

- **학습 데이터** : x_train, y_train
- **batch_size** : 32 (데이터를 32개씩 묶어서 학습)
- **epochs** : 20 (총 20번 학습)
- **validation_data**
  - (x_val, y_val) : 검증 데이터로 x_val과 y_val을 사용
- **shuffle :** True (매 epoch마다 데이터를 섞음)
- **verbose :** 1 (출력을 진행 바 형태로 출력)

```
# 모델 학습 실행
model.fit(
    x_train, y_train,
    batch_size = 32,
    epochs = 20,
    validation_data = (x_val, y_val),
    shuffle = True,
    verbose = 1
)
```