AI 공모전 간단한 구현

https://youtu.be/uE_nrgEcUXs

AI 공모전 아이디어 및 시스템 구성

Attention

Transformer

구현 및 결과

Meta Police AI - 데이터

사이버 스토킹 판단을 위해 반영할 데이터 예시

Action Code	Context Code	World Code
월드 이동	즐겨찾기 된 월드 접속	한성대학교 학술정보관
	타 사용자 따라가기로 접속	한성대학교 연구관
타사용자 피드 방문	선물하기	노원 문화의 거리
	메시지 전송	한강 공원
	프로필 확인	
	포토부스 같이 찍기	…
음성채팅 및 채팅	음성 채팅 가능 거리 내에서 마이크 사용하여 상호작용	
	음성 채팅 가능 거리 내에 마이크 사용 없음	
카메라 기능	월드 내 사진 찍기	

User = [“월드 이동_ 즐겨찾기 된 월드_ 한성대학교 학술정보관”, “음성채팅_마이크사용하여 채팅_한성대학교 학술정보관”, …]

Stalker = [“월드 이동_ 타 사용자 따라가기로 접속_ 한성대학교 학술정보관”, “카메라 기능_월드 내 사진 찍기_한성대학교 학술정보관”, …]

1. 이와 같이 3가지 feature (AC, CC, WC)를 사용하여 데이터 생성 및 사전 구축

2. 사용자마다 시간 흐름에 따른 순차적인 행동 로그들이 벡터 형태로 모델에 입력 (어떤 지역에서 어떤 행동을 하였는지에 관한 행동 정보들)

Meta Police AI – 신경망 구조 및 학습

Transformer (Encoder-Decoder) + Autoencoder + **Binary classification**

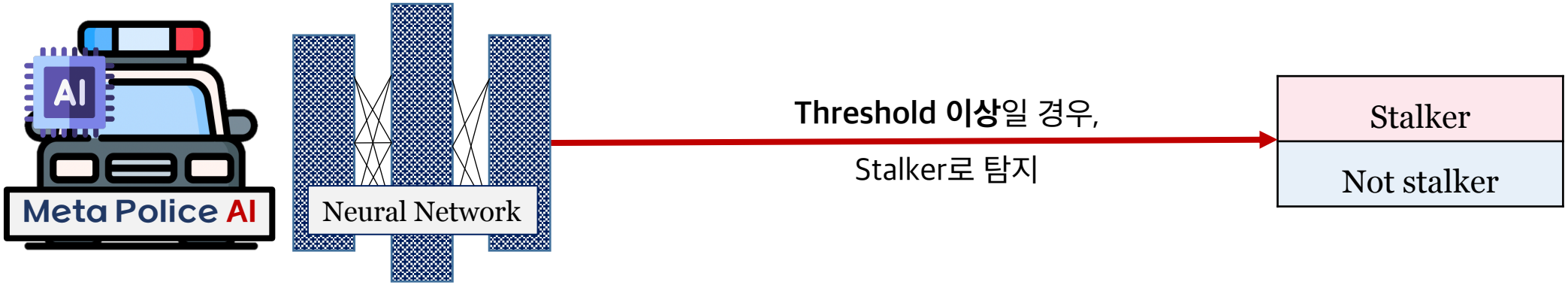
시계열 데이터 학습 모델 (+ 인코더를 통한 특징 추출) 분류 모델

데이터로 구성한 행동 패턴들은 **일반사용자는 지속적, 반복적으로 하지 않을 행동**이므로

행동 로그 데이터의 특징을 추출하면 그 차이를 알 수 있음

따라서 추출된 특징은 이진 분류기에 입력되어, **스토커와 일반 유저로 분류 가능**

Meta Police AI – 신경망 구조 및 학습

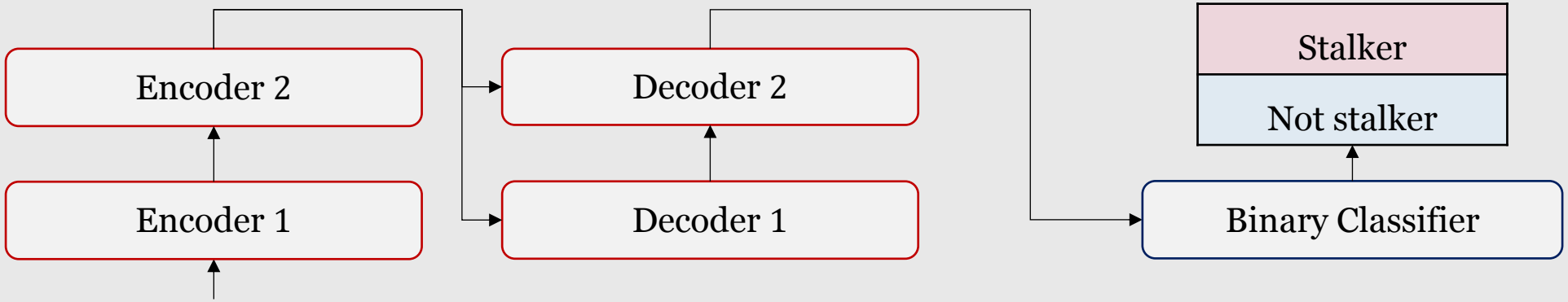


Transformer (Encoder-Decoder) + Autoencoder + **Binary classification**

행동 로그를 임베딩하여 **특징 벡터 추출** (input과 가깝게 output 생성)

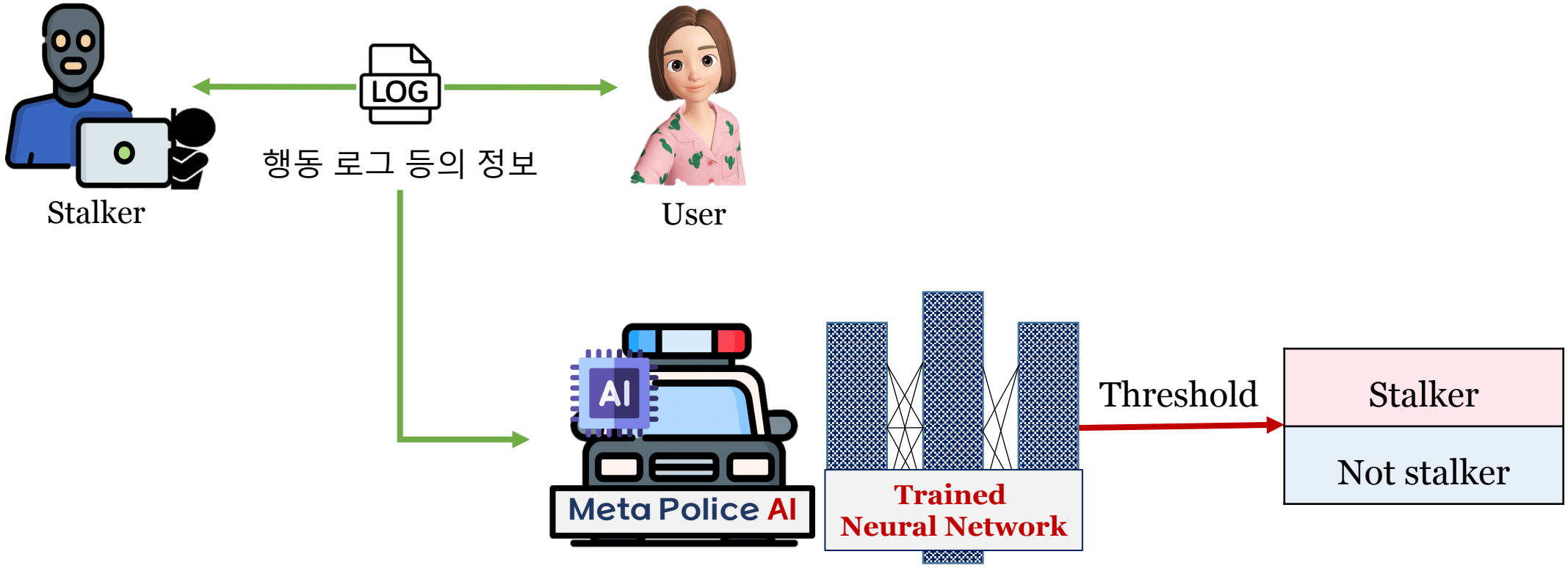
훈련 시, 스토커와 일반사용자의 **임베딩 벡터 간의 유사도가 적을 것**

해당 벡터는 **이진 분류 신경망**을 통해 **스토커와 일반 유저로 분류**



Stalker = [“월드 이동_ 타 사용자 따라가기로 접속_ 한성대학교 학술정보관”, …]

Meta Police AI – 추론



학습된 신경망에 학습 시와 동일한 형태의 데이터를 입력

이상행동 유저를 탐지해내도록 **잘 학습된 모델을 사용하므로 탐지 가능**

Transformer

- **정교한 어텐션 메커니즘으로 전체 시퀀스를 처리할 수 있는 대형 인코더-디코더 모델**
- **기존의 시계열 신경망들**

  RNN → LSTM → Seq2Seq → Attention → Transformer → GPT/BERT …

Seq2Seq

- **Seq2seq** (번역 등과 같이 시계열 데이터 입력받아 시계열 데이터를 출력)**의 문제**

  1. 시퀀스의 처음부터 **순차적으로 처리**하며, 해당 정보가 다음 요소로 감

     → 마지막 요소가 해당 시퀀스 전체의 **정보를 압축**하여 담고 있음

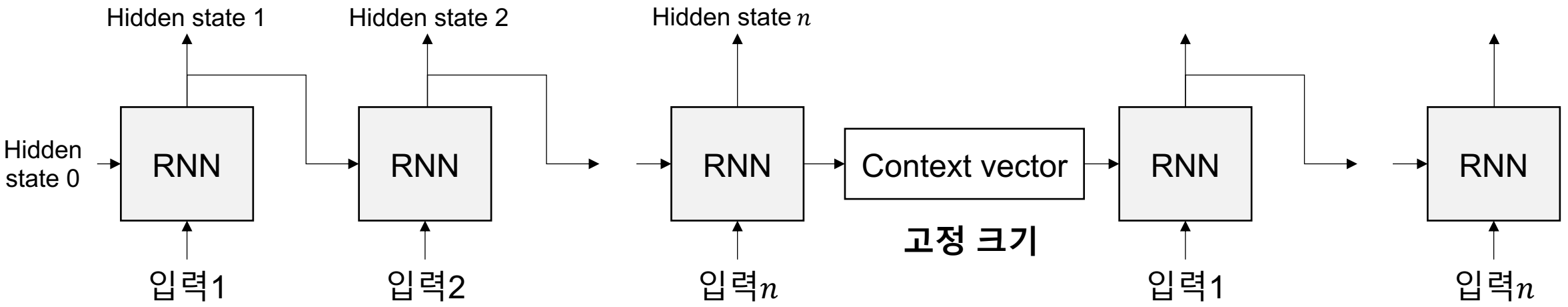
  2. 압축된 벡터는 고정된 크기

     → 시퀀스 길이가 아무리 길어도 **고정된 크기의 context vector**로 표현

  3. 해당 context vector로부터 시퀀스를 다시 구성

- 즉, Seq2seq는 하나의 **고정된 크기의 벡터가 입력 시퀀스 전체의 정보를 담아야 하므로 성능 저하**

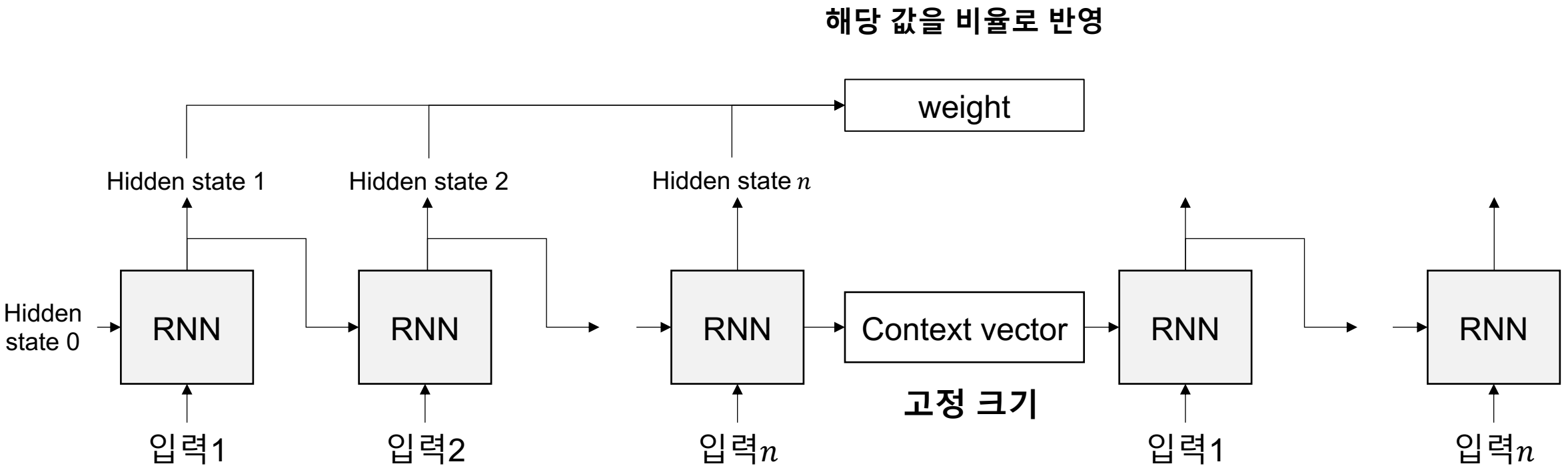
  → 그럼 고정 크기 벡터에 압축하지 않는다면?



Attention

- 시퀀스를 다시 생성해낼 때 입력 시퀀스의 어떤 부분에 초점을 둘 필요가 있는지 점수를 계산

  → 시퀀스의 각 요소(단어)에 대한 결과값 (hidden state)에 가중치를 곱하여 각각의 입력에 대한 점수 매김



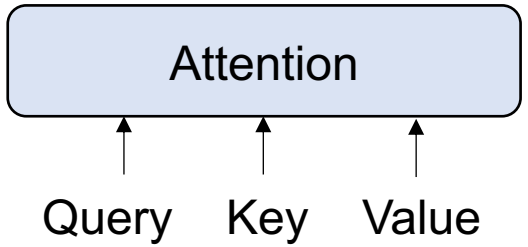
Attention

- **Query (주체), Key (다른 대상), Value 로 구성**

  → Query가 Key에 대해서 각각의 Attention score를 계산 (Query 기준으로 각 Key들이 얼마나 영향을 미치는지)

  → Attention score 계산 후, Value 값과 곱하여 최종 Attention value 구함 → 조정된 가중치 얻음

- **I am hungry 예시**
  - Query : I
  - Key : I, am, hungry
  - Value 값과 해당 Score를 곱하여 최종 Score 구함



- **행렬곱 → 스케일링 → 소프트맥스** (Scaled-dot Attention)

  → 각각의 Key중에서 어떤 Key가 가장 높은 연관성을 갖는지 비율로 확인 가능

  → 해당 결과를 Value와 곱하여 조정된 가중치 얻어냄

  → 영향을 더 크게 받는 경우 더 큰 attention score 얻으므로 더 큰 가중치 얻게 됨 **(연관 있는 단어에 더 집중)**

- Dot, Scaled-dot, Concat 등 다양한 기법 존재

Multi-head Attention

- **Multi-head attention**
  - **여러 개의 attention을 수행**

    → 입력은 Value, Key, Query로 나뉘고, 각 헤드마다 다르게 존재

    → 각 헤드마다 서로 다른 결과 나옴

    → 즉, 각 단어에 대한 **여러 관계와 뉘앙스를 인코딩**할 수 있는 더 큰 능력을 제공 **(더 다양한 특징을 학습 가능)**
  - 각 헤드를 나누어 병렬적으로 계산한 후, Concat → 입출력 차원 동일

Transformer

- **RNN을 사용하지 않고 Attention을 사용**하여 시퀀스 데이터 학습 가능

  → 대신 위치 정보를 반영하는 Positional Encoding 사용

- **Seq2Seq과 달리 문장 전체가 한번에 입력된 후 Attention** (순차적 X)

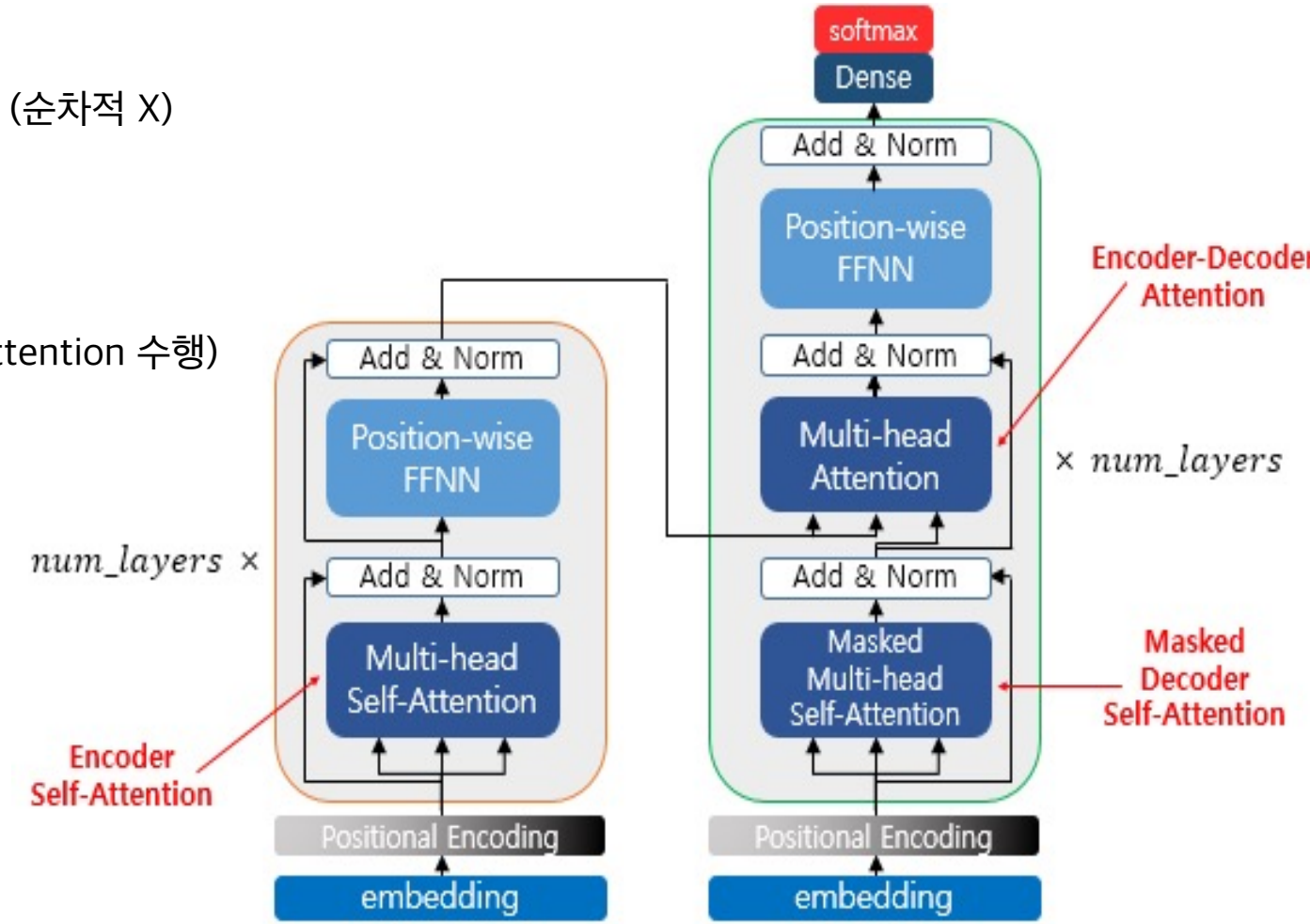
- **Encoder + Decoder 모델**

  → Encoder 및 Decoder를 여러 개 쌓아서 구성

  → En/Decoder 각각에서 Attention 수행 (즉, 여러 번의 Attention 수행)

- **전체 흐름**

  1. word embedding 후 위치 정보 더해서 인코딩
  2. 해당 값에 Self attention 적용
  3. Self attention 적용하지 않은 값과 더하고 정규화
  4. Feed Forward layer + Add & Norm
  5. Encoder의 최종 출력은 Decoder의 각 레이어에 입력
  6. Attention - Add & Norm … 출력



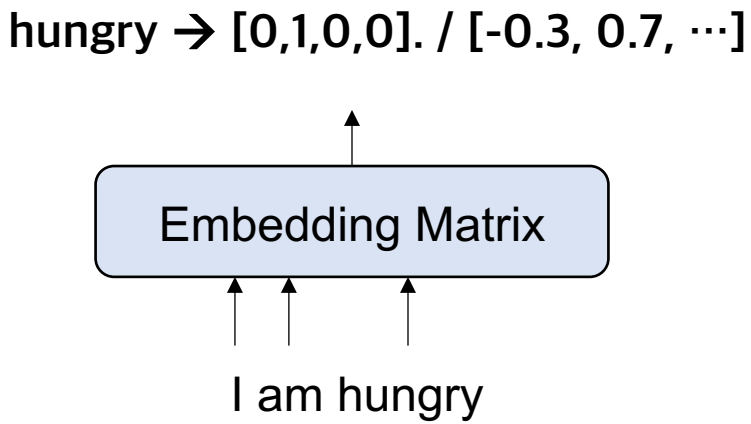
Transformer - Embedding

- **Embedding**

  인간이 쓰는 단어의 의미를 컴퓨터가 이해할 수 있도록 특정 차원의 벡터로 표현

- Word2Vec, CBOW, Skip-Gram 등 다양한 embedding 관련 기법 존재

- One-hot encoding, 실수 벡터 등으로 표현 가능



Transformer – Positional encoding

- Transformer는 입력 시퀀스를 순차적으로 처리하는 **RNN을 사용하지 않음**
- **위치 정보를 반영**하는 Positional Encoding 필요

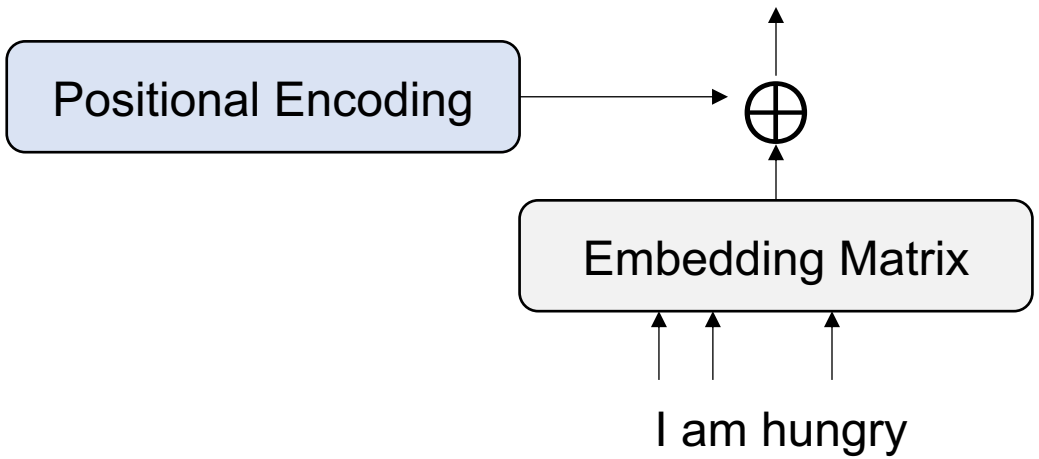
  → sin, cos 함수를 통해 인코딩 (-1 ~ 1 사이 값 가짐)

  (위치 정보 표현을 위한 조건을 모두 만족 ; 토큰 간 거리, 유일한 값, 긴 문장 표현, 위치 값 예측 가능)

  → **전체 차원이 d 차원 → 0~d 차원 중, 짝수는 sin, 홀수는 cos 적용** (pos는 각 토큰의 위치 정보 값이며, 정수)

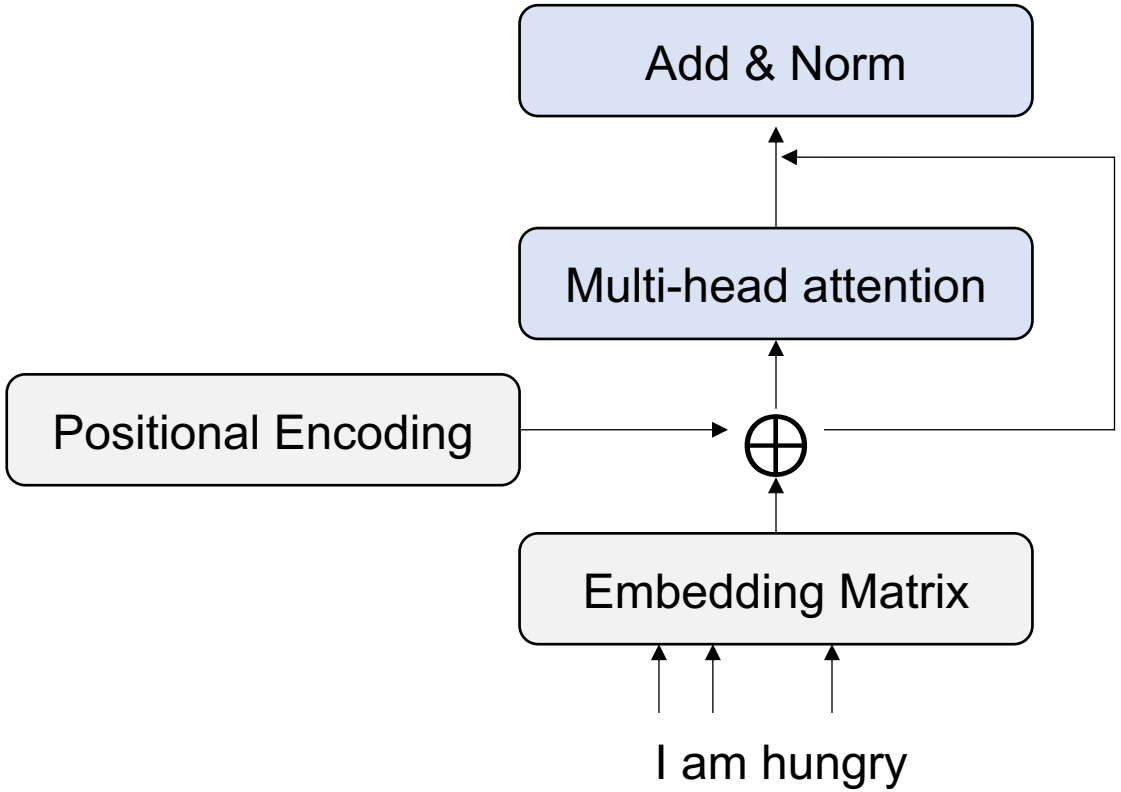
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- Embedding 행렬과 같은 차원을 가지는 위치 정보를 담은 해당 값을 각 **요소마다 더함**



Transformer – Multi-head attention

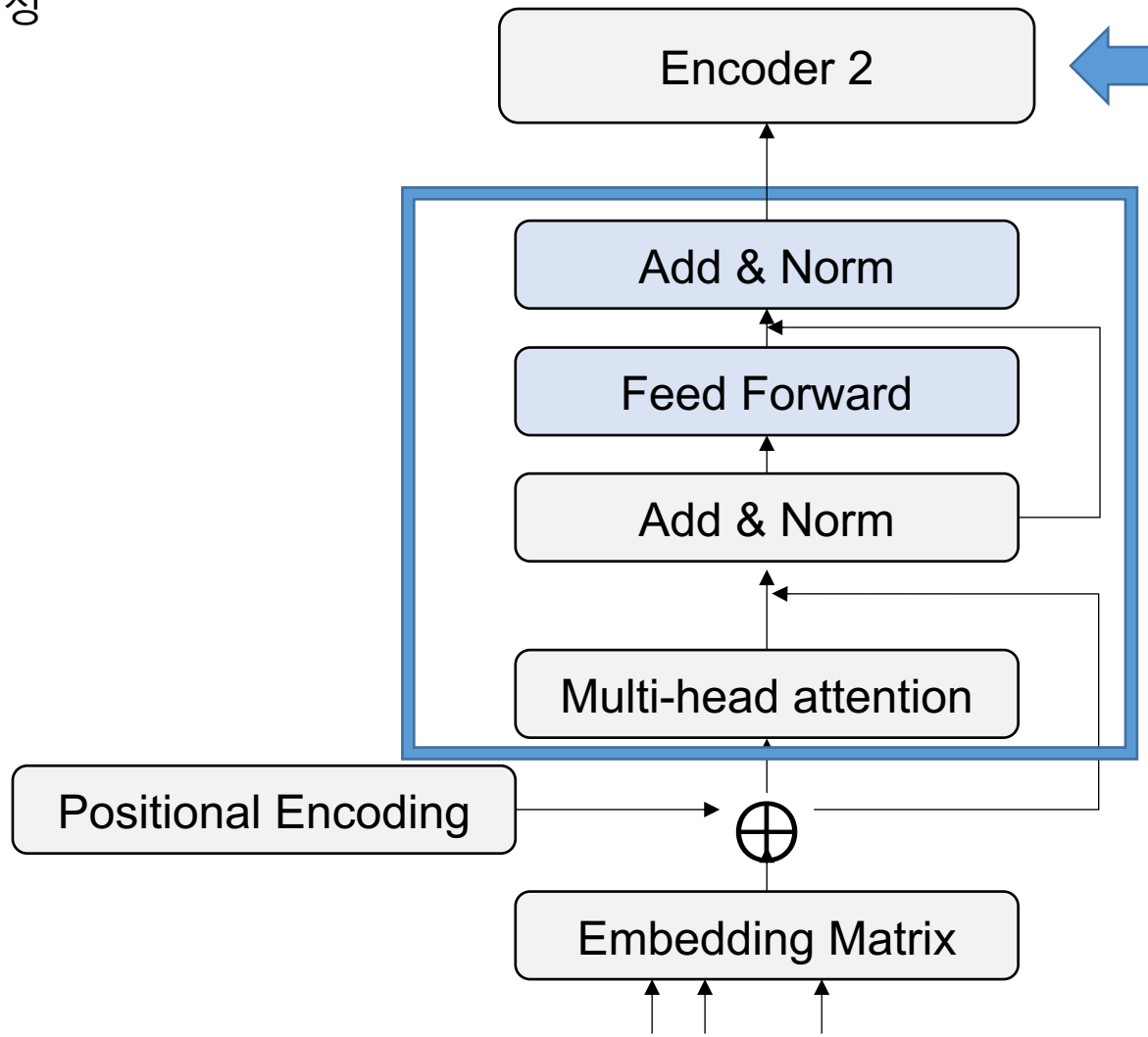
- 입력 문장의 각각의 단어가 서로 어떤 연관성을 갖는지 계산하는 **Self-Attention 여러 개 수행**
  → 각각의 attention score를 구해서 **각 단어가 어떤 단어와 가장 많은 연관이 있는지 파악** 가능
- 입력 시퀀스의 **문맥에 대한 전반적인 정보**를 학습
- Residual connection으로 받은 값과 Attention한 결과를 함께 받아서 더한 후, 정규화 수행



잔여(Residual)학습
→ Attention 정보 + 그냥 정보
→ 초기 수렴 속도 상승

Transformer - Encoder

- Encoder 전체 구성



동일 구조

하나의 encoder 출력이 다음 encoder의 입력
(즉, 차원 동일)

Encoder 1

Transformer - Decoder

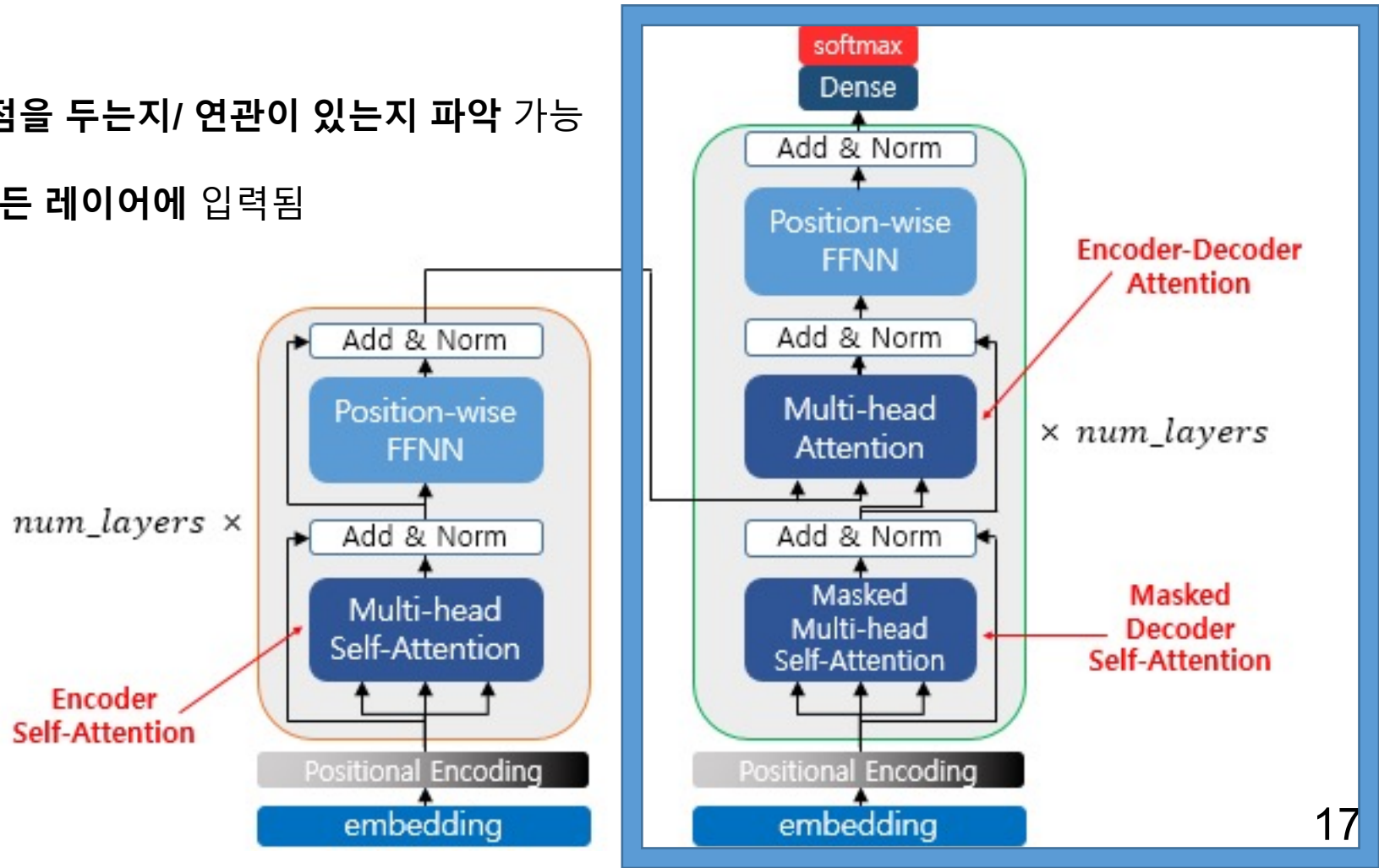
- Decoder 또한 Encoder와 동일한 구조

  → Decoder의 output 또한 Sequence

- Encoder의 출력을 받아서

  입력 시퀀스 중에서 어떤 단어에 가장 초점을 두는지/ 연관이 있는지 파악 가능

- **마지막 Encoder**의 출력이 **Decoder의 모든 레이어에** 입력됨



구현 - 데이터

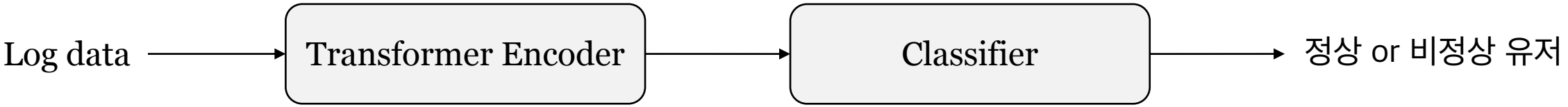
- 데이터는 앞에 나온 것과 같이 직접 구성 (10개만 생성해 봄)

Log → [“월드 이동_ 즐겨찾기 된 월드_ 한성대학교 학술정보관”, “음성채팅_마이크사용하여 채팅_한성대학교 학술정보관”, …]

1	id	log	label
2	1	Move the world. Favorite World. Academic Information Center. Move the world. Favorite World. I	0
3	2	Move the world. Favorite World. Academic Information Center. Voice chat. Chat using micropho	0
4	3	Move the world. World. Cultural street. Chat. Chat without microphone. Cultural street. Move the	0
5	4	Visit follower's feed. Give a present. Follower's feed. Move the world. World. Research Center.	0
6	5	Use a camera. Take pictures in favorite world. Hangang.	0
7	6	Move the world. Follow other users. Academic Information Center. Move the world. Follow other	1
8	7	Move the world. Follow other users. Hangang. Use a camera. Take pictures in world. Hangang.	1
9	8	Visit other user's feed. Send a message. Other user's feed. Visit other user's feed. Send a message	1
10	9	Move the world. Follow other users. Park. Voice chat. Chat using microphone. Park. Chat. Chat v	1
11	10	Move the world. Follow other users. City hall. Visit other user's feed. Check a profile. Other user's	1

구현 - 네트워크

- **행동로그 데이터 임베딩**을 위한 네트워크 구조는 **Transformer 내부의 인코더만** 쓰는 것으로 대체
  - 지금 생각으로는 Transformer의 encoder만 사용하여도 될 것 같아서 디코더 생략
  - 데이터의 수가 많아지거나 성능 향상을 원할 경우 디코더와 오토인코더 더해서 써도 될 것 같음 (레이어 추가만 하면 됨)
- **임베딩 된 벡터를 분류**하는 네트워크는 **Transformer encoder 뒤에 붙여서 하나의 네트워크로 구성**
  - Fully-connected neural network 사용



구현

```
tokenizer = Tokenizer(num_words=max_num_words, oov_token='<unk>')       # 문장으로부터 단어를 토큰화하고 숫자에 대응시키는 딕셔너리를 사용
tokenizer.fit_on_texts(texts_train)                                     # 문자 데이터를 입력받아서 리스트의 형태로 변환
```

```
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = keras.Sequential(
            [layers.Dense(ff_dim, activation="relu"), layers.Dense(embed_dim),]
        )
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

```
inputs = layers.Input(shape=(max_sequence_len,))
embedding_layer = TokenAndPositionEmbedding(max_sequence_len, max_num_words, embed_dim)
x = embedding_layer(inputs)
transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
x = transformer_block(x)
x = layers.GlobalAveragePooling1D()(x)
x = layers.Dropout(0.1)(x)

#ae = layers.Dense(16, activation="relu")(x)
#x = layers.Dense(32, activation="relu")(ae)

# classifier
x = layers.Dense(20, activation="relu")(x)
x = layers.Dropout(0.1)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

실험 결과

- 분류는 성공

```
Prediction : 1 || Real label : 1.0
Prediction : 0 || Real label : 0.0
Prediction : 1 || Real label : 1.0
```

- 데이터가 매우 적어서 그런 걸 수도 있지만, 부정사용자 탐지 관련된 연구가 많으므로 다양한 방법으로 가능할 것으로 예상

감사합니다.