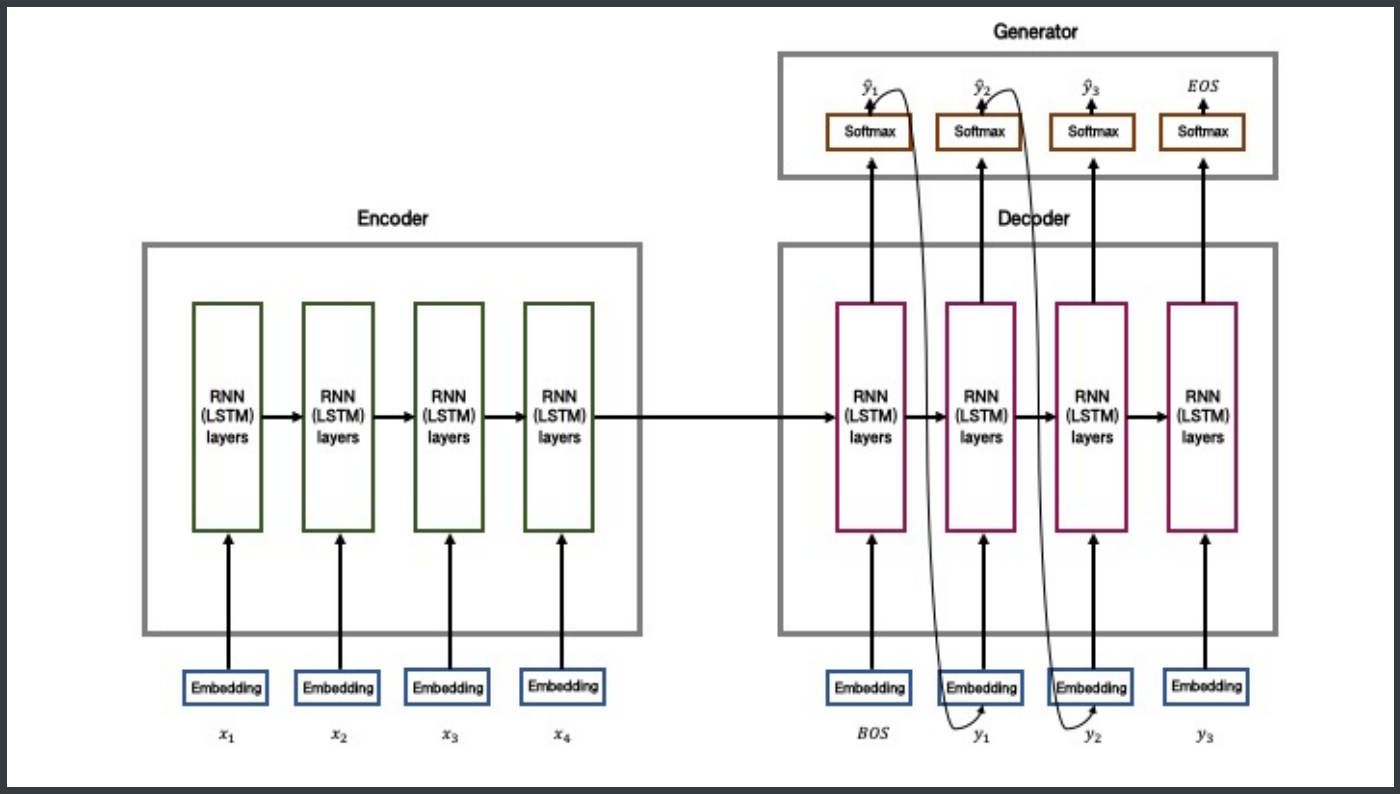
시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)

시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)는 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 모델이다.
기계번역, 챗봇, text summarization 등 다양한 분야에서 사용되지만, 모델 자체의 이해를 돕기 위해 아래의 설명에서는 번역 task를 기반으로 설명해보겠다.



sequence to sequence는 크게 세 가지 부분으로 나뉜다.

- Encoder
- Decoder
- Generator

각자의 요소들이 어떤 기능을 하는지 차근차근 알아보자

Encoder

encoder는 입력되는 sequence의 정보를 최대한 보존하도록 압축을 진행하는 역할을 한다.
즉, 문장의 의미가 잘 담겨있는 상태로 높은 차원에 있던 data를 낮은 차원의 latent space(잠재 공간)에 투영시키는 것이다.
seq2seq에서의 encoder는 입력을 받은 뒤, 정보를 보존하도록 압축을 진행하고, 이렇게 압축한 정보인 context vector를 decoder에 넘겨주는 역할을 한다.
더 자세한 이해를 위해 Encoder를 수식과 함께 살펴보자

$$
\begin{gathered}
D = \{x^i, y^i\}_{i=1}^N \\
x^i = \{x_1^i, \ldots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \ldots, y_n^i\} \\
\text{where } y_0 = <\text{BOS}> \; and \; y_n = <\text{EOS}>
\end{gathered} \tag{1}
$$

Dataset은 x와 y의 문장 pair로 이루어져 있고, (번역 task에서는 x가 한국어 문장, y가 영어 문장이라고 가정하자)
문장 x는 m개의 단어, 문장 y는 n개의 단어로 이루어져있다고 한다.
단, 이때 문장 y의 시작과 끝은 각각 , 토큰으로 이루어져 있다.
이때, 각 문장들의 shape은 다음과 같다

$$
\begin{gathered}
|x^i| = (\text{batch\_size}, \text{m}, |V_s|) \\
|y^i| = (\text{batch\_size}, \text{n}, |V_t|) \\
\text{where } |V_s| \; and \; |V_t| = \text{vocab size}
\end{gathered} \tag{2}
$$

일단, encoder에는 x만 입력으로 들어가기 때문에, x가 encoder에 들어가는 과정을 수식으로 살펴보겠다.

$$
\begin{gathered}
h_t^{enc} = \text{RNN}_{enc}(\text{emb}_{enc}(x_t), h_{t-1}^{enc}), \text{ where } h_0^{enc} = 0 \\
h_{1:m}^{enc} = [h_1^{enc}, \ldots, h_m^{enc}], \\
\text{where } h_t^{enc} \in \mathbb{R}^{\text{batch\_size x 1 x hidden\_size}} \text{ and } h_{1:m}^{enc} \in \mathbb{R}^{\text{batch\_size x m x hidden\_size}}
\end{gathered} \tag{3}
$$

먼저, encoder의 input인 문장 x_t가 encoder의 embedding layer를 통과한다 (이 때의 shape은 $(batch\_size, 1, embedding\_size)$)
이후, embedding layer를 통과한 input은 이전 timestep($t-1$)의 hidden state와 함께 RNN layer의 input으로 들어가게 된다.

이렇게 현재 timestep(t)의 hidden state를 구하게 된다. (이 때의 shape은 $(batch\_size, 1, hidden\_size)$)
결론적으로, $h_{1:m}^{enc}$은 $(batch_size, m, hidden_size)$의 shape을 가지게 된다
그런데, 만약 encoder가 bidirectional RNN을 사용하게 된다면 다음과 같은 shape을 가지게 된다.

$$h_t^{enc} \in \mathbb{R}^{\text{batch\_size x 1 x (2 x hidden\_size)}} \text{ and } h_{1:m}^{enc} \in \mathbb{R}^{\text{batch\_size x m x (2 x hidden\_size)}} \quad (4)$$

Decoder

decoder는 encoder로 압축된 정보를 입력되는 sequence와 같아지도록 압축 해제하는 역할이다.
즉, encoder가 압축한 정보를 받아서 단어를 하나씩 뱉어내는 역할이다.
seq2seq에서의 decoder는 encoder로부터 정보를 받아온 뒤, encoder의 마지막 hidden state를 decoder의 initial state로 넣어준다.
수식과 함께 자세히 살펴보도록 하겠다.

$$\begin{gathered} D = \{x^i, y^i\}_{i=1}^N \\ x^i = \{x_1^i, \ldots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \ldots, y_n^i\} \\ \text{where } y_0 = <\text{BOS}> \; and \; y_n = <\text{EOS}> \end{gathered} \quad (5)$$

우선, encoder 설명때와 마찬가지로 Dataset은 x와 y의 문장 pair로 이루어져 있고, (번역 task에서는 x가 한국어 문장, y가 영어 문장이라고 가정하자) 문장 x는 m개의 단어, 문장 y는 n개의 단어로 이루어져있다고 한다.

$$\begin{gathered} h_t^{dec} = \text{RNN}_{dec}(\text{emb}_{dec}(\hat{y}_{t-1}), h_{t-1}^{dec}), \\ \text{where } h_0^{dec} = h_m^{dec} \\ h_{1:n}^{dec} = [h_1^{dec}, \ldots, h_n^{dec}] \end{gathered} \quad (6)$$

이전 timestep($t-1$)의 decoder의 output인 $\hat{y}_{t-1}$이 embedding layer를 통과한다 (이 때의 shape은 $(batch\_size, 1, embedding\_size)$))
이후, embedding layer를 통과한 input은 이전 timestep($t-1$)의 hidden state와 함께 RNN layer의 input으로 들어가게 된다.
이렇게 현재 timestep(t)의 hidden state를 구하게 된다. (이 때의 shape은 $(batch\_size, 1, hidden\_size)$)
결론적으로, $h_{1:n}^{dec}$은 $(batch_size, n, hidden_size)$의 shape을 가지게 된다.

이러한 decoder는 encoder로부터 문장을 압축한 context vector를 바탕으로 문장을 생성하며, auto-regressive task이기 때문에 bi-directional RNN을 사용하지 못한다는 특징이 있다.

Generator

Generator는 decoder의 hidden state를 받아 현재 timestep의 출력 token에 대한 확률 분포를 반환하는 역할을 한다.

수식과 함께 자세히 알아보도록 하겠다.

$$
\begin{gathered}
D = \{x^i, y^i\}_{i=1}^N \\
x^i = \{x_1^i, \ldots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \ldots, y_n^i\} \\
\text{where } y_0 = <\text{BOS}> \; and \; y_n = <\text{EOS}>
\end{gathered} \tag{7}
$$

encoder와 decoder때와 마찬가지로 dataset은 동일하다.
Dataset은 x와 y의 문장 pair로 이루어져 있고, (번역 task에서는 x가 한국어 문장, y가 영어 문장이라고 가정하자) 문장 x는 m개의 단어, 문장 y는 n개의 단어로 이루어져있다.
와 는 decoder에만 존재하는 token으로, 의 경우 decoding(문장 생성)을 시작하는 신호이며, 이후 순차적으로 decoding(문장 생성)을 진행하다 가 출력되게 되면 decoding이 끝났다는 뜻으로, decoding이 종료되게 된다.

$$
\begin{gathered}
h_t^{dec} = \text{RNN}_{dec}(\text{emb}_{dec}(\hat{y}_{t-1}), h_{t-1}^{dec}), \\
\text{where } h_0^{dec} = h_m^{dec}
\end{gathered} \tag{8}
$$

generator의 경우, decoder의 각 timestep별 output인 h_t^{dec}를 입력으로 받는다. 위에서 언급한것처럼, 이전 timestep($t-1$)의 decoder의 output인 $\hat{y}_{t-1}$이 embedding layer를 통과한다 (이 때의 shape은 $(batch\_size, 1, embedding\_size)$)) 이후, embedding layer를 통과한 input은 이전 timestep($t-1$)의 hidden state와 함께 RNN layer의 input으로 들어가게 된다. 이렇게 현재 timestep(t)의 hidden state를 구하게 된다. (이 때의 shape은 $(batch\_size, 1, hidden\_size)$)

$$
\begin{gathered}
\hat{y}_t = \text{softmax}(h_t^{dec} \cdot W_{gen}) \\
\text{where } h_t^{dec} \in \mathbb{R}^{\text{batch\_size x 1 x hidden\_size}} \text{ and } W_{gen} \in \mathbb{R}^{\text{hidden\_size x } |\text{V}|}
\end{gathered} \tag{9}
$$

이후 decoder의 output을 받아와서 linear layer를 통과시킨 이후 softmax를 적용시켜 단어의 확률 분포를 반환한다.
즉, 현재 timestep의 단어를 예측하기 위해, 현재 timestep의 결과물을 vocab(V) 안의 단어 별 확률값로 변환해주는 것이다.

따라서, linear layer는 decoder의 output($1, \text{hidden size}$)을 vocab의 size($|V|$)로 변환해주기 때문에 $(\text{hidden size}, |\text{V}|)$의 shape을 가지게 된다.