

LM model

Y.-B.Hong

Task

- Dataset에서 **Padding**을 붙여서 가져오는 (batch별 **Padding** 추가)
 - 가져올 때 **X**는 첫 **Character**
 - **Y**는 마지막 **character** 제외
 - **Padding** 붙이고 총 길이를 확인할 때 유의해야 한다
- **loss** 떨어지는 그래프 추가하기

Content

- NLP introduction
- Neural network language model
- Language model RNN

Content

- NLP introduction
- Neural network language model
- Language model RNN

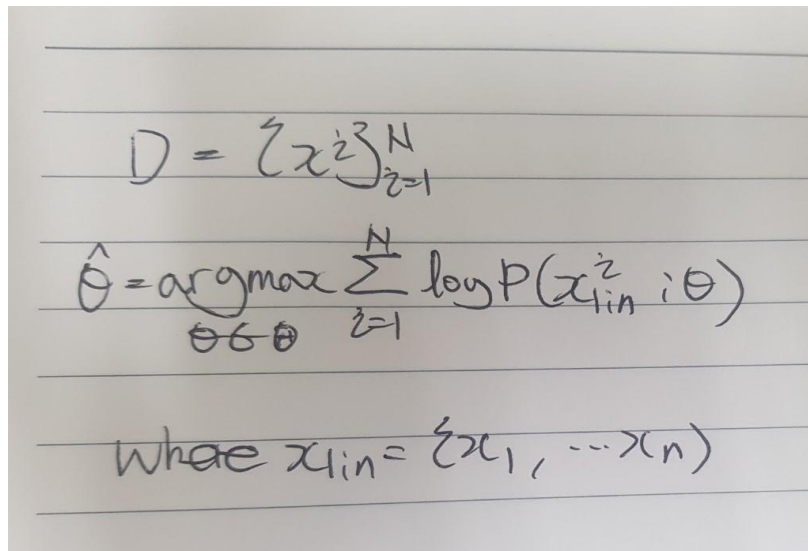
Review: Autoencoders

- 인코더(encoder)와 디코더(decoder)를 통해 압축과 해제를 실행
 - 인코더는 입력(\mathbf{x})의 정보를 최대한 보존하도록 손실 압축을 수행
 - 디코더는 중간 결과물(\mathbf{z})의 정보를 입력(\mathbf{x})와 같아지도록 압축 해제(복원)를 수행
- 복원을 성공적으로 하기 위해,
 - 오토인코더(autoencoder)는 특징(feature)을 추출하는 방법을 자동으로 학습

Introduction

- 우리의 머릿속에는 단어와 단어 사이의 호가률이 우리도 모르게 학습되어 있음
→ 대화를 하다가 정확하게 듣지 못하여도 대화에 지장이 없음
- 많은 문장들을 수집하여, 단어와 단어 사이의 출현 빈도를 세어 확률을 계산
- 궁극적인 목표는 우리가 일상 생활에서 사용하는 언어의 문장 분포를 정확하게 모델링하는 것

Objective of language modeling



Handwritten mathematical expressions on lined paper:

$$D = \{x^i\}_{i=1}^N$$
$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmax}} \sum_{i=1}^N \log P(x_{1:n}^i | \theta)$$

where $x_{1:n} = \{x_1, \dots, x_n\}$

Objective

$$\begin{aligned} P(A, B, C, D) &= P(D|A, B, C)P(A, B, C) \\ &= P(D|A, B, C)P(C|A, B)P(A, B) \\ &= P(D|A, B, C)P(C|A, B)P(B|A)P(A) \end{aligned}$$

Can convert joint probability to conditional probability

$$\begin{aligned} P(x_{1:n}) &= P(x_1, \dots, x_n) \\ &= P(x_n|x_1, \dots, x_{n-1}) \cdots P(x_2|x_1)P(x_1) \\ &= \prod_{i=1}^n P(x_i|x_{<i}) \end{aligned}$$

$$\log P(x_{1:n}) = \sum_{i=1}^n \log P(x_i|x_{<i})$$

Can re-write the equation
→ Chain rule
→ Property of log

Example of chain rule

$$\begin{aligned}P(A, B, C, D) &= P(D|A, B, C)P(A, B, C) \\&= P(D|A, B, C)P(C|A, B)P(A, B) \\&= P(D|A, B, C)P(C|A, B)P(B|A)P(A)\end{aligned}$$

$$\begin{aligned}P(\langle \text{BOS} \rangle, \text{I, love, to, play}, \langle \text{EOS} \rangle) &= P(\langle \text{EOS} \rangle | \langle \text{BOS} \rangle, \text{I, love, to, play})P(\langle \text{BOS} \rangle, \text{I, love, to, play}) \\&= P(\langle \text{EOS} \rangle | \langle \text{BOS} \rangle, \text{I, love, to, play})P(\text{play} | \langle \text{BOS} \rangle, \text{I, love, to})P(\langle \text{BOS} \rangle, \text{I, love, to}) \\&= P(\langle \text{EOS} \rangle | \langle \text{BOS} \rangle, \text{I, love, to, play})P(\text{play} | \langle \text{BOS} \rangle, \text{I, love, to})P(\text{to} | \langle \text{BOS} \rangle, \text{I, love})P(\langle \text{BOS} \rangle, \text{I, love}) \\&= P(\langle \text{EOS} \rangle | \langle \text{BOS} \rangle, \text{I, love, to, play})P(\text{play} | \langle \text{BOS} \rangle, \text{I, love, to})P(\text{to} | \langle \text{BOS} \rangle, \text{I, love})P(\text{love} | \langle \text{BOS} \rangle, \text{I})P(\langle \text{BOS} \rangle, \text{I}) \\&= P(\langle \text{EOS} \rangle | \langle \text{BOS} \rangle, \text{I, love, to, play})P(\text{play} | \langle \text{BOS} \rangle, \text{I, love, to})P(\text{to} | \langle \text{BOS} \rangle, \text{I, love})P(\text{love} | \langle \text{BOS} \rangle, \text{I})P(\text{I} | \langle \text{BOS} \rangle)P(\langle \text{BOS} \rangle)\end{aligned}$$

Example of chain rule

$$\begin{aligned}P(x_{1:n}) &= P(x_1, \dots, x_n) \\&= P(x_n|x_1, \dots, x_{n-1}) \cdots P(x_2|x_1)P(x_1) \\&= \prod_{i=1}^n P(x_i|x_{<i})\end{aligned}$$

$$\log P(x_{1:n}) = \sum_{i=1}^n \log P(x_i|x_{<i})$$

$$\mathcal{D} = \{x^i\}_{i=1}^N$$

$$\hat{\theta} = \operatorname{argmax}_{\theta \in \Theta} \sum_{i=1}^N \log P(x_{1:n}^i; \theta)$$

where $x_{1:n} = \{x_1, \dots, x_n\}$.



$$= \operatorname{argmax}_{\theta \in \Theta} \sum_{i=1}^N \sum_{j=1}^n \log P(x_j^i | x_{<j}^i; \theta)$$

where $x_{1:n} = \{x_1, \dots, x_n\}$.

Using language model

- Pick better(fluent) sentence
- Predict next word given previous words

$$\hat{x}_t = \operatorname{argmax}_{x_t \in \mathcal{X}} \log P(x_t | x_{<t}; \theta)$$

Mid-summary

- 언어모델은 주어진 코퍼스 문장들의 **likelihood**를 최대화 하는 파라미터를 찾아내, 주어진 코퍼스 기반으로 언어의 분포를 학습한다.
→ 즉 코퍼스 기반으로 문장들에 대한 확률 분포 함수를 근사(Approximation)한다.
- 문장 확률은 단어가 주어졌을 때, 다음 단어를 예측하는 확률을 차례대로 곱한 것과 같다.
- 따라서 언어모델은 주어진 단어가 있을 때, 다음 단어의 **likelihood**를 최대화하는 파라미터를 찾는 과정이라고도 볼 수 있다.
→ 주어진 단어들이 있을 때, 다음 단어에 대한 확률 분포 함수를 근사하는 과정

Content

- NLP introduction
- Neural network language model
- Language model RNN

Neural language model

- Resolve sparsity

→ Training set

⇒ 고양이는 좋은 반려동물 입니다.

→ Test set

⇒ 강아지는 훌륭한 애완동물 입니다.

- Because we know

→ 고양이 \approx 강아지

→ 좋은 \approx 훌륭한

→ 반려동물 \approx 애완동물

- But n-gram cannot, because word are discrete symbols

Mid summary

n-gram (previous method)

- 단어를 discrete symbol로 취급
 - ⇒ Extract matching에 대해서만 count
- 따라서 generalization issue 발생
 - Markov assumption 도입 (n-gram)
 - Smoothing & Discounting
 - Interpolation & Back-off
 - Unseen sequence에 대한 대처 미흡
- 빠른 연산 & 쉽고 직관적
 - 단순한 look-up table 방식
 - 문장 fluency 비교 task에서는 괜찮음

Neural network language model

- Word embedding을 통해,
 - unseen sequence에 대해 대처 가능
- Generation task에서 특히 강점
- 연산량 많음 (feed forward 연산)

Neural language model

- Find parameter that maximize likelihood for given training corpus

$$\begin{aligned}\mathcal{D} &= \{x^i\}_{i=1}^N \\ \hat{\theta} &= \operatorname{argmax}_{\theta \in \Theta} \sum_{i=1}^N \log P(x^i; \theta) \\ &= \operatorname{argmax}_{\theta \in \Theta} \sum_{i=1}^N \sum_{j=1}^n \log P(x_j^i | x_{<j}^i; \theta)\end{aligned}$$

Neural language model with entropy

- 불확실성 $\propto 1/\text{확률} \propto \text{정보량}$
- 정보량

→ $-\log$ 때문에, 확률이 0에 가까워질수록 높은 정보량

→ $I(\mathbf{x}) = -\log P(\mathbf{x})$

- 언어모델 관점

→ 흔히 나올 수 없는 문장(확률이 낮은 문장)일수록 더 높은 정보량

Neural language model with perplexity

- 확률값 역수의 기하평균

$$\begin{aligned}\text{PPL}(x_1, \dots, x_n; \theta) &= P(x_1, \dots, x_n; \theta)^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P(x_1, \dots, x_n; \theta)}} \\ &= \sqrt[n]{\frac{1}{\prod_{i=1}^n P(x_i | x_{<i}; \theta)}}$$

Language model with entropy and perplexity

- Cross entropy

$$\begin{aligned} H(P, P_\theta) &= -\mathbb{E}_{x_{1:n} \sim P}[\log P(x_{1:n}; \theta)] \\ &\approx -\frac{1}{n} \sum_{x_{1:n} \in \mathcal{X}} P(x_{1:n}) \log P(x_{1:n}; \theta), \text{ defined as per-word entropy} \\ &\approx -\frac{1}{n \times N} \sum_{i=1}^N \log P(x_{1:n}^i; \theta), \text{ by Monte-carlo} \\ &\approx -\frac{1}{n} \log P(x_{1:n}; \theta), \text{ where } N = 1 \\ &\approx -\frac{1}{n} \sum_{i=1}^n \log P(x_i | x_{<i}; \theta) \\ &= \mathcal{L}(x_{1:n}; \theta) \end{aligned}$$

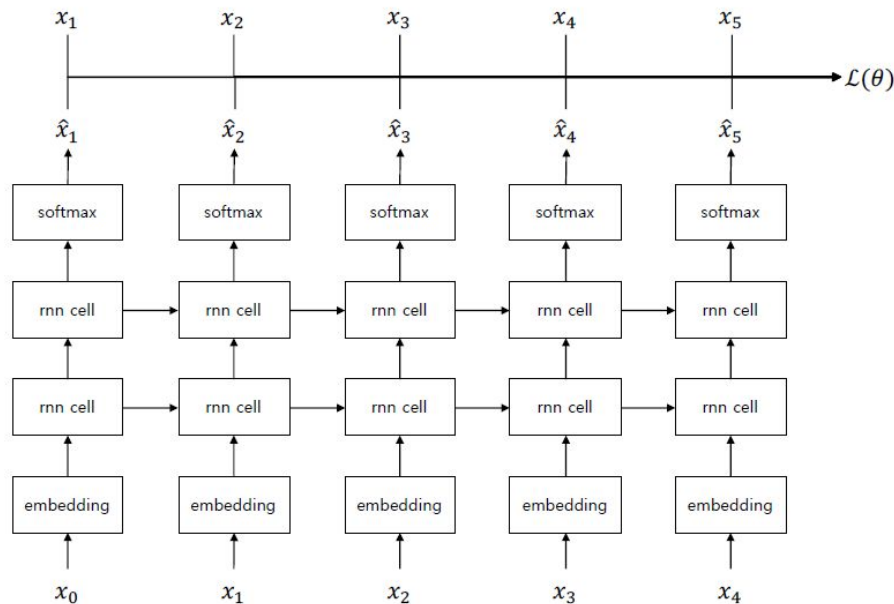
$$\begin{aligned} \mathcal{L}(x_{1:n}; \theta) &\approx -\frac{1}{n} \sum_{i=1}^n \log P(x_i | x_{<i}; \theta) \\ &= -\frac{1}{n} \log \prod_{i=1}^n P(x_i | x_{<i}; \theta) \\ &= \log \sqrt[n]{\frac{1}{\prod_{i=1}^n P(x_i | x_{<i}; \theta)}} \\ &= \log \text{PPL}(x_{1:n}; \theta) \end{aligned}$$

Mid summary

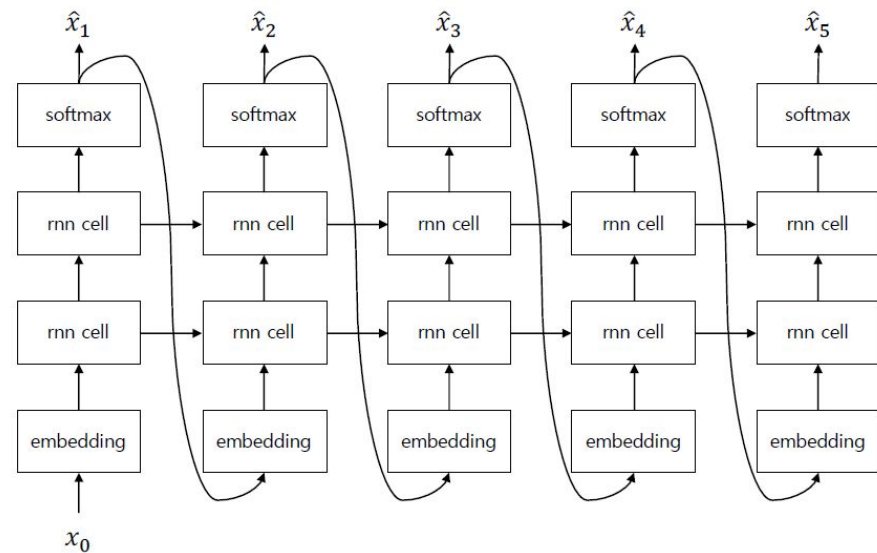
- Objective: minimize perplexity
 - equivalent to minimize cross entropy
 - is also same as minimizing negative log-likelihood
- 문장의 likelihood를 maximize하는 파라미터를 찾고 싶음
 - Ground-truth 확률 분포(실제 사람이 가진 언어 모델)에 언어모델을 근사(approximate)하고 싶음
- GT 분포와 LM 분포사이의 cross entropy를 구하고 minimize
 - 문장의 perplexity를 minimize

Language model with auto-regressive and teacher-forcing

Training mode



Inference mode



Language model with auto-regressive and teacher-forcing

```
teacher_forcing_ratio = 0.5
```

```
def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden
```

Language model with auto-regressive and teacher-forcing

```
if use_teacher_forcing:
    # Teacher forcing 포함: 목표를 다음 입력으로 전달
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_tensor[di])
        decoder_input = target_tensor[di] # Teacher forcing
else:
    # Teacher forcing 미포함: 자신의 예측을 다음 입력으로 사용
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach() # 입력으로 사용할 부분을 히스토리에서 분리

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break
```

```
loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.item() / target_length
```

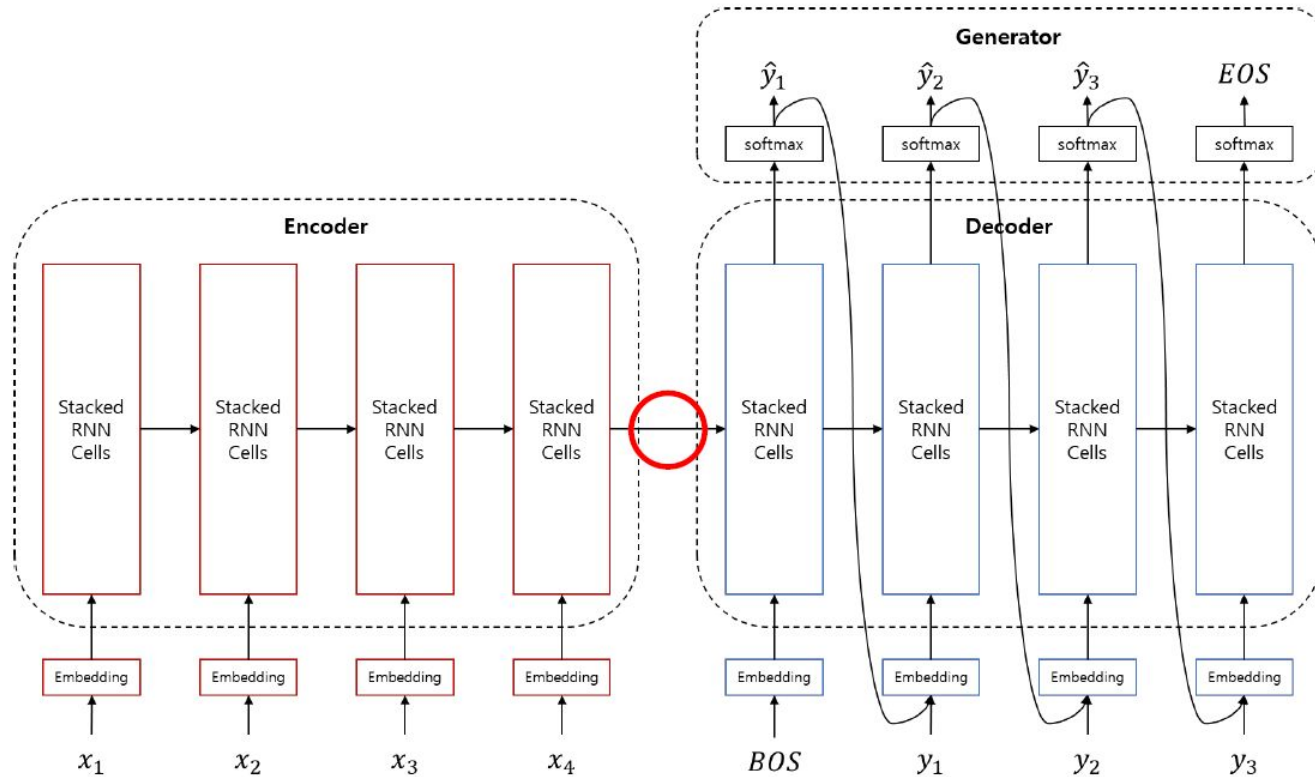
Language model with auto-regressive and teacher-forcing

- Auto-regressive task에서는 보통 이전 time-step의 모델의 출력을 다음 time-step의 입력으로 넣어 줌
→ 이전 time-step의 추력에 따라 현재 모델의 state가 바뀌게 될 것
- 하지만 적절한 학습을 위해서는 학습 시에는 이전 time-step의 출력 값이 아닌, 실제 정답을 넣어줌
- 따라서 학습과 추론을 위한 방법이 다르게 되어 여러가지 문제가 발생
→ 학습을 위한 코드와 추론을 위한 코드를 따로 짜야함
→ 학습과 추론 방법의 괴리(discrepancy)가 발생하여 성능이 저하될 수 있음

Content

- NLP introduction
- Neural network language model
- Language model RNN

Sequence to sequence



Encoder equations

- Given dataset,

$$\mathcal{D} = \{x^i, y^i\}_{i=1}^N$$

$$x^i = \{x_1^i, \dots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \dots, y_n^i\},$$

where $y_0 = \langle \text{BOS} \rangle$ and $y_n = \langle \text{EOS} \rangle$.

$$|X_t| = (\text{batch size}, 1, |V|)$$

embedding layer

(batch size, 1, word embedding vector size)

rnn, lstm layer

(batch size, 1, hidden size)

- Get hidden states of encoder

$$h_t^{\text{enc}} = \text{RNN}_{\text{enc}}(\text{emb}_{\text{enc}}(x_t), h_{t-1}^{\text{enc}}), \text{ where } h_0^{\text{enc}} = 0.$$

if bidirectional:

(batch size, 1, hidden size*2)

$$h_{1:m}^{\text{enc}} = [h_1^{\text{enc}}; \dots; h_m^{\text{enc}}],$$

concatenation

where $h_t^{\text{enc}} \in \mathbb{R}^{\text{batch_size} \times 1 \times \text{hidden_size}}$ and $h_{1:m}^{\text{enc}} \in \mathbb{R}^{\text{batch_size} \times m \times \text{hidden_size}}$. (batch size, n, hidden size)

- If we use bi-directional RNN

$$h_t^{\text{enc}} \in \mathbb{R}^{\text{batch_size} \times 1 \times (2 \times \text{hidden_size})} \text{ and } h_{1:m}^{\text{enc}} \in \mathbb{R}^{\text{batch_size} \times m \times (2 \times \text{hidden_size})}.$$

Encoder equations

- Encoder는 source문장을 압축한 context vector를 decoder에게 넘겨준다.
- Encoder는 train/test 시에 항상 문장 전체를 받음
 - Encoder(자체만 놓고 보면 non-auto-regressive(task.
 - 따라서 bi-directional RNN 사용 가능

Decoder equations

- Given dataset,

$$\mathcal{D} = \{x^i, y^i\}_{i=1}^N$$

$$x^i = \{x_1^i, \dots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \dots, y_n^i\},$$

where $y_0 = \langle \text{BOS} \rangle$ and $y_n = \langle \text{EOS} \rangle$.

- We can get hidden state of decoder

$$h_t^{\text{dec}} = \text{RNN}_{\text{dec}}(\text{emb}_{\text{dec}}(\hat{y}_{t-1}), h_{t-1}^{\text{dec}}),$$

where $h_0^{\text{dec}} = h_m^{\text{enc}}$.

$$h_{1:n}^{\text{dec}} = [h_1^{\text{dec}}; \dots; h_n^{\text{dec}}]$$

$$|Y_t| = (\text{batch size}, 1, |V|)$$

embedding layer

(batch size, 1, word embedding vector size)

rnn, lstm layer

(batch size, 1, hidden size)

concatenation

(batch size, n, hidden size)

Decoder equations

- 디코더는 conditional language model 이라고 볼 수 있음

→ 인코더로부터 문장을 압축한 **context vector**를 바탕으로 문장을 생성

$$\begin{aligned}\hat{\theta} &= \operatorname{argmax}_{\theta \in \Theta} \sum_{i=1}^N \log P(y^i | x^i; \theta) \\ &= \operatorname{argmax}_{\theta \in \Theta} \sum_{i=1}^N \sum_{j=1}^n \log P(y_j^i | x^i, y_{<j}^i; \theta)\end{aligned}$$

- Auto-regressive task에 속하므로, uni-directional RNN을 사용

Generator equations

- Given dataset,

$$\mathcal{D} = \{x^i, y^i\}_{i=1}^N$$

$$x^i = \{x_1^i, \dots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \dots, y_n^i\},$$

where $y_0 = \langle \text{BOS} \rangle$ and $y_n = \langle \text{EOS} \rangle$.

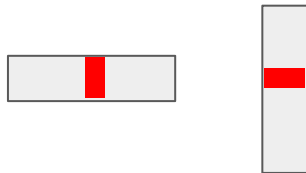
- Get hidden states of encoder

$$h_t^{\text{dec}} = \text{RNN}_{\text{dec}}(\text{emb}_{\text{dec}}(\hat{y}_{t-1}), h_{t-1}^{\text{dec}}),$$

where $h_0^{\text{dec}} = h_m^{\text{enc}}$.

(batch size, 1, hidden size) x (hidden size, |V|)

- If we use bi-directional RNN



$$\hat{y}_t = \text{softmax}(h_t^{\text{dec}} \cdot W_{\text{gen}}),$$

where $h_t^{\text{dec}} \in \mathbb{R}^{\text{batch_size} \times 1 \times \text{hidden_size}}$ and $W_{\text{gen}} \in \mathbb{R}^{\text{hidden_size} \times |V|}$.