

Lecture 06. Language models and RNNs

Css224n Natural language Processing with Deep Learning

UOS STAT NLP Study
Hyehyeon Moon

2020.11.04

Language Model

단어 시퀀스에 대해 확률을 부여하는 모델

=확률을 통해 다음 단어가 무엇이 올지 예측하는 모델

= $P(x(t)|x(t-1), \dots, x(1))$ 를 통해 다음 단어가 무엇이 올지 예측하는 모델

Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.
- For example, if we have some text $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) &= P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)}) \\ &= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)}) \end{aligned}$$

This is what our LM provides

But, count를 통해 확률을 구하는데, 엄청 큰 corpus에 대해서 위와 같이 한 단어 앞서 나온 모든 단어들을 고려하는 것은 불가능!

따라서 n-gram 모델이 나오게 되는데, 아래와 같이 한 단어의 앞서 나온 (n-1) 단어들만 고려해 주어 $P(x^{(t+1)}|x^{(t)}, \dots, x^{(t-n+2)})$ 계산!

n-gram Language Models

- First we make a **simplifying assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words.

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) = P(x^{(t+1)} | \overbrace{x^{(t)}, \dots, x^{(t-n+2)}}^{n-1 \text{ words}}) \quad (\text{assumption})$$

prob of a n-gram \rightarrow $P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})$

prob of a (n-1)-gram \rightarrow $P(x^{(t)}, \dots, x^{(t-n+2)})$

$\quad \quad \quad =$

$\quad \quad \quad$ (definition of conditional prob)

- Question:** How do we get these n -gram and $(n-1)$ -gram probabilities?
- Answer:** By **counting** them in some large corpus of text!

$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \quad (\text{statistical approximation})$$

Markov Assumption

"오직 직전 기록만이 현재에 영향을 끼친다는 가정"

N-gram model에서 확률을 계산할 때 마르코프 가정 하에 확률을 계산하는 것임

Unigram : 마르코프 가정

Bigram, N-gram : 마르코프 가정의 완화 버전

확률 언어 모델 (Probabilistic language model)

- Markov Assumption
 - 계산 모델의 메모리에 대한 제한
 - "오직 직전 기록만이 의미를 가짐"
 - Markov property
 - 미래 상태에 대한 조건부 확률 분포가 현재의 상태에만 종속적이라면 해당 모델은 Markov Property 를 가진다
 - $P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-1})$

$$P(\text{the} \mid \text{its water is so transparent } \underline{\text{that}}) \approx P(\text{the} \mid \underline{\text{that}})$$

확률 언어



Andrei Markov

확률 언어 모델 (Probabilistic language model)

- Markov Assumption 의 완화
 - "일부 직전 기록이 의미를 가짐"
 - $P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$
 - $P(w_i | w_1 w_2 \dots w_{i-1}) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$



Andrei Markov

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

- Memoryless 모델의 고려
 - 언어에 내재한 단어 관계성을 제한하는 것
 - "모든 단어의 등장은 독립적이다." → Binary independence model in IR
 - 실용적인 분야 (검색엔진 등) 에서 대규모 데이터의 계산 가능성 확보

N-gram model의 종류

Unigram model

식 : $P(w_1, w_2, \dots, w_n) = P(w_i | w_1, w_2, \dots, w_{i-1}) = \pi_i P(w_i)$

가정 : memoryless model

의미 없는 단어의 나열이 됨

←

Bigram model

식 : $P(w_1, w_2, \dots, w_n) = P(w_i | w_1, w_2, \dots, w_{i-1}) = \pi_i P(w_i | w_{i-1})$

가정 : Markov Assumption

실용적으로 성공을 거둠

←

N-gram model

식 : $P(w_1, w_2, \dots, w_n) = P(w_i | w_1, w_2, \dots, w_{i-1}) = \pi_i P(w_i | w_{i-N+1}, \dots, w_{i-2}, w_{i-1})$

가정 : 완화된 Markov Assumption

어떠한 context에서 의미를 반영할 수 있는 모델로 평가받음

N-gram model은 언어가 가진 long-distance dependencies를 적절히 무시함(sara=she이지만 알지 못함)

Count based N-gram language Model : MLE를 통해 확률 추정

강의에서 앞서서 $P(x(t+1)|x(t), \dots, x(t-n+2)) = \text{count}$ 기반으로 확률추정이 된다고 했는데, 왜 count 기반으로 확률추정이 되는지에 대한 증명(?)

바로 MLE를 통해 count 기반으로 확률이 추정된다!

(자세한 계산과정은 ppt 참고자료 안에 있음)

Bigram 확률 추정

- 주어진 corpus 에서 Bigram의 등장 횟수를 셈 (count)
 - MLE 관점으로 확률 추정
- 이전 단어 x 가 주어졌을 때, 단어 y 가 등장할 확률을 추정하기 위해
 - Observed frequency
 - bigram 모델의 경우 두 단어 x, y 의 인접 등장 횟수, $C(xy)$, 를 계산하여 모든 bigram의 등장 횟수로 나누어 줌

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

- w_{i-1} 로 시작하는 bigram 의 개수는 w_{i-1} unigram 의 개수와 동일

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Count based Language Model의 문제 : Unknown Words, Sparsity and Storage

- Unknown Words

LM의 문제: 일반화 (Generalization)

- Unknown words 문제
 - 우리가 한번도 본적이 없는 단어란 무엇인가?
 - out of vocabulary (OOV) 단어라고도 함
 - OOV rate
 - test set 에서 OOV 가 차지하는 비율
- Closed vocabulary assumption
 - (주어진) lexicon 으로부터 추출한 단어만을 포함하여 test set 을 구축함
 - OOV 발생 없음
- Open vocabulary system
 - <UNK> 라는 pseudo-word 를 추가하여 잠재적 OOV에 대한 모델링을 수행
 - vocab 을 사전에 고정하여 closed vocab assumption 을 가정하고, text normalization 과정에 OOV 에 대해서는 <UNK> 를 배정
 - 혹은, 사전에 vocab 을 결정하지 않고, 학습 데이터에서 발생하는 단어 등장 빈도에 따라, 빈도가 적은 단어를 <UNK> 로 결정

pp를 비교할 때 vocab이 같은 것이 사용되었는지, 가정이 어떠한지 유의해야 함

Count based Language Model의 문제 : Unknown Words, Sparsity and Storage

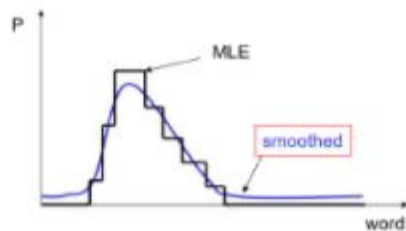
- Sparsity의 해결책 : Smoothing and Backoff

(여러 Smoothing의 정의 및 수식과 Backoff는 참고 ppt 안에 자세히 나와 있음)

- Storage의 해결책 : 없고, n을 너무 크게 하면 안 됨

Smoothing

- OOV는 아니지만, **unseen context** 에 대한 모델링 방법
 - 추정 확률에 대한 변경
 - 빈번한 사건으로부터 일부 떼어낸 확률값을 **unseen event** 에 배정
 - Laplace smoothing
 - add-k smoothing
 - Backoff and Interpolation
 - Kneser-Ney smoothing
 - Stupid backoff
- Smoothing 의 직관

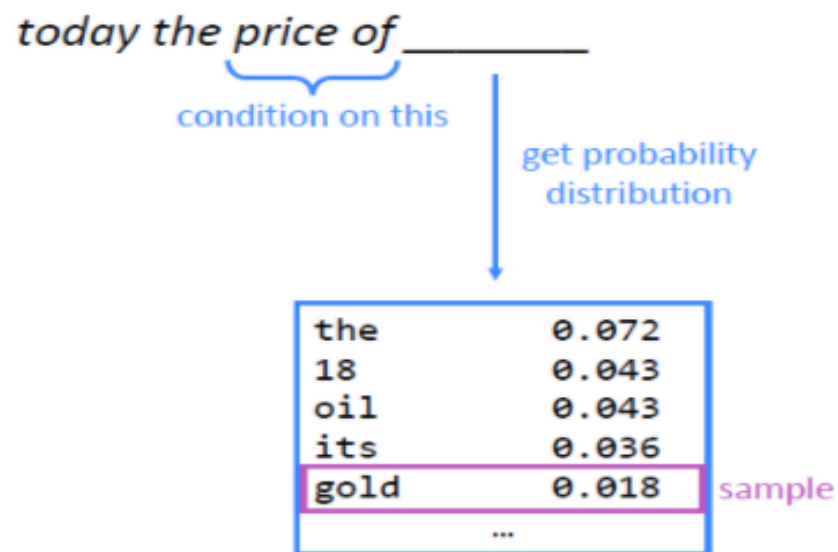


Generating text with a n-gram Language Model

Training data로부터 학습한 확률들을 이용해서 가장 적절한 확률을 선택하여 문장을 만들어나감

Generating text with a n-gram Language Model

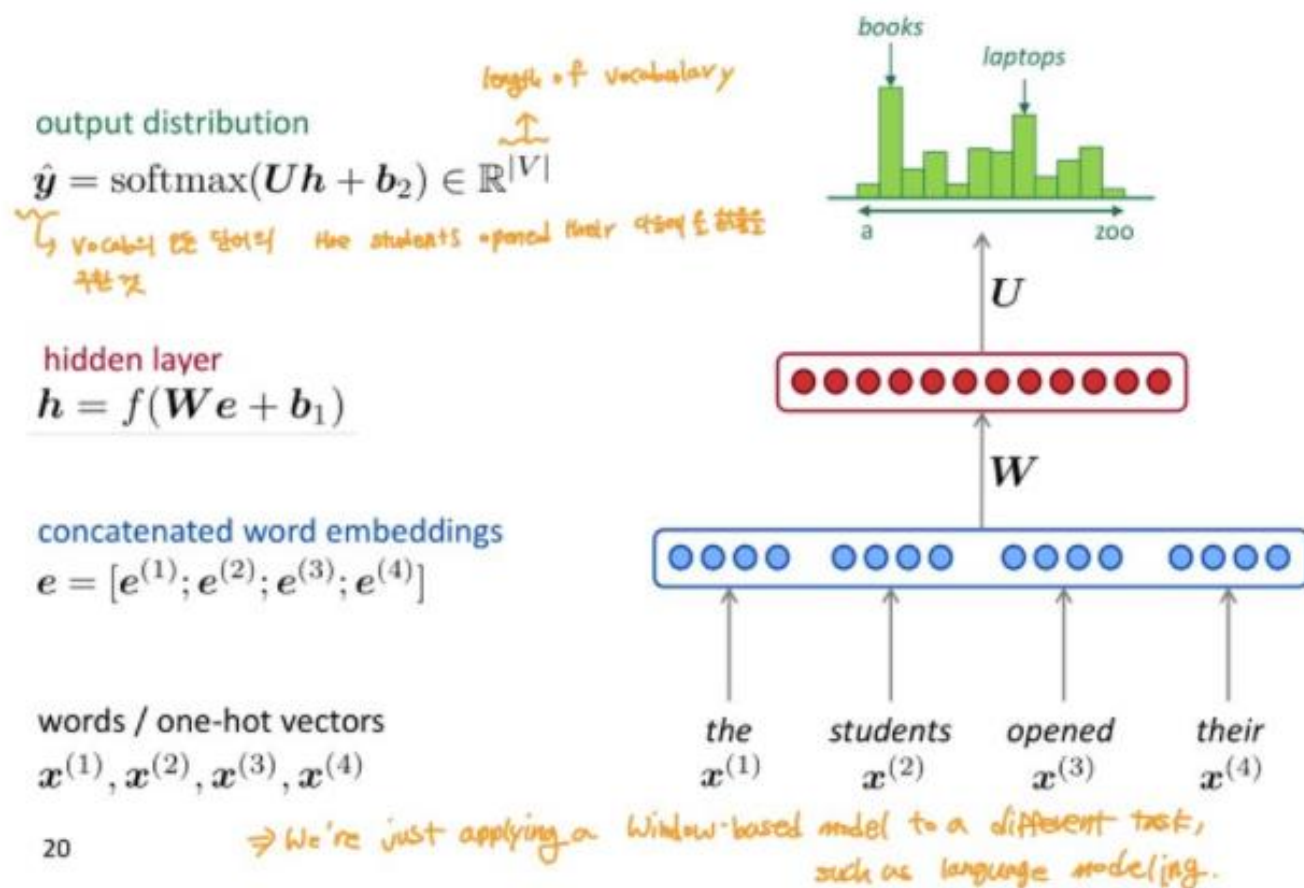
- You can also use a Language Model to generate text.



Neural Language Model for N-gram

window-based neural model 방식을 language model에 사용한 형태

A fixed-window neural Language Model



Neural Language Model for N-gram 장점 및 문제점

A fixed-window neural Language Model

Improvements over n -gram LM:

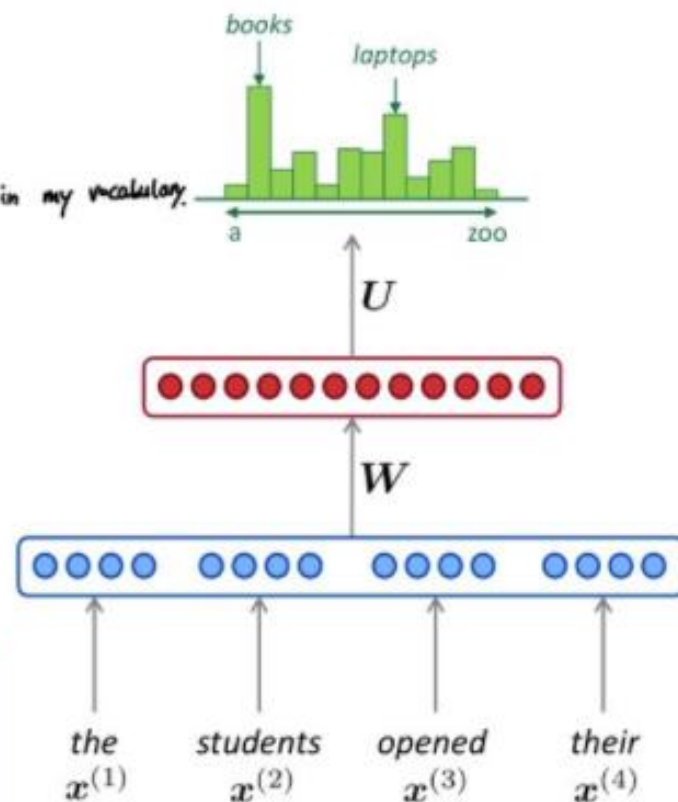
- No sparsity problem
- Don't need to store all observed n -grams

→ just store all of the word vectors for all the words in my vocabulary.

Remaining problems:

- Fixed window is too small
 - Enlarging window enlarges W
 - Window can never be large enough!
 - $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W .
- No symmetry in how the inputs are processed.

We need a neural architecture that can process any length input

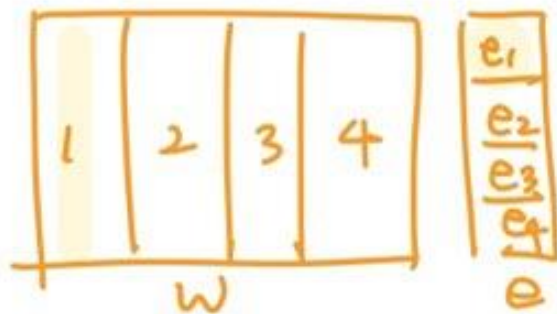


ppt 문제점 중 4번째에 대해 추가적 설명

- (1) 만약 단어들이 비슷한 경우, weight이 서로 공유되지 않아서 비슷한 processing을 반복해 주어야 하는 비효율성 발생한다는 것
- (2) 특정 단어에 맞추어서 weight의 특정 부분만 업데이트가 되므로
만약 test 문장에서 input에 train set과는 다른 word가 오면 성능이 나빠질 것임

[추가설명]

- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W .
No symmetry in how the inputs are processed.



⇒ e_1 원들은 1번 section의
값들만 공유해서

learn in the weight matrix in one section is
not shared with the others.

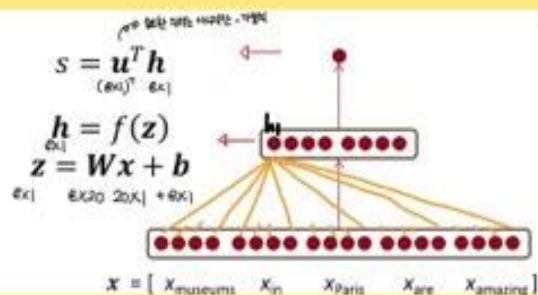
⇒ 만약 단어들이 비슷할 경우, section 1~4 등 비슷한 weights
가져가는데 share되지 않으므로 훈련을 각각 총 4번씩이나 해야됨.

⇒ It's kind of inefficient that we're learning,
all of these separate weights for these different words
when there's a lot of commonalities between them.

⇒ 비효율적

Backpropagation

①



$$\begin{aligned} \frac{\partial s}{\partial b} &= \frac{\partial s}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial b} \\ &= u^T \cdot \text{diag}(f'(z)) \cdot I \\ &= u^T \circ f'(z) \\ (6 \times 1)^T \quad 6 \times 20 \end{aligned}$$

② δ : local error signal

$$\begin{aligned} \frac{\partial s}{\partial w} &= \frac{\partial s}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial w} = \delta \cdot \frac{\partial z}{\partial w} \\ \frac{\partial s}{\partial b} &= \frac{\partial s}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial b} = \delta \cdot \frac{\partial z}{\partial b} \end{aligned}$$

계승

$$\delta = \frac{\partial s}{\partial h} \cdot \frac{\partial h}{\partial z} = u^T \circ f'(z)$$

1x6

③ $\frac{\partial s}{\partial w}$ 이 어떤 결과?

$\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_{\theta} J(\theta)$ 로 업데이트 하기 때문에

$w_{6 \times 20}$ 라 같은 형태 $\nabla_{\theta} J(\theta) = \frac{\partial s}{\partial w}$ 형태 이어야 함

$\Rightarrow \frac{\partial s}{\partial w} = \delta x$ 인데 6×20 형태를 만들어주기 위해

$$\frac{\partial s}{\partial w} = \delta^T x^T$$

$[1 \times 6] \quad [6 \times 1] \quad [1 \times 20]$
 $6 \times 20 \quad 6 \times 1 \quad 1 \times 20$

$$\begin{aligned} \left(\frac{\partial h}{\partial z} \right)_{ij} &= \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \\ &= \begin{cases} f'(z_i) & i=j \\ 0 & i \neq j \end{cases} \\ \Rightarrow \frac{\partial h}{\partial z} &= \begin{pmatrix} f'(z_1) & 0 & \dots & 0 \\ 0 & f'(z_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f'(z_n) \end{pmatrix} \\ &= \text{diag}(f'(z)) \end{aligned}$$

* Jacobian matrix (제2)

$f(x) = [f_1(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)]$: m output and n input

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad \left(\frac{\partial f}{\partial x} \right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

* W 구체적으로 살펴보기

$$\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_6 \\ 6 \times 1 \end{bmatrix} = \begin{bmatrix} \text{h와 다른 x들의 편미} \\ \vdots \\ 6 \times 20 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{20} \\ 20 \times 1 \end{bmatrix} + \begin{bmatrix} b \\ 6 \times 1 \end{bmatrix}$$

(참고 : symmetry : 같은 단어가 다른 위치에 나타나면 다르게 처리하는 것을 말함,

<https://jeongukjae.github.io/posts/cs224n-lecture-6-language-model-and-rnn/>)

RNN

- Neural language model은 fixed-window로 input의 길이가 정해져 있어 가변 input에 대해서는 처리해 주지 못한다는 단점이 있음. 이를 해결하는 것이 바로 RNN
- symmetry(weight들이 서로 공유가 안 된다는 단점)을 RNN은 weight 공유로서 해결함

(=a network where you apply the exact same weights on every step)

RNN Forward & Backpropagation

Forward of RNN

A RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(U h^{(t)} + b_2) \in \mathbb{R}^{|\mathcal{V}|}$$

$\mathcal{V} = \{a, n, o, s, t, u, z\}$

hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = E x^{(t)}$$

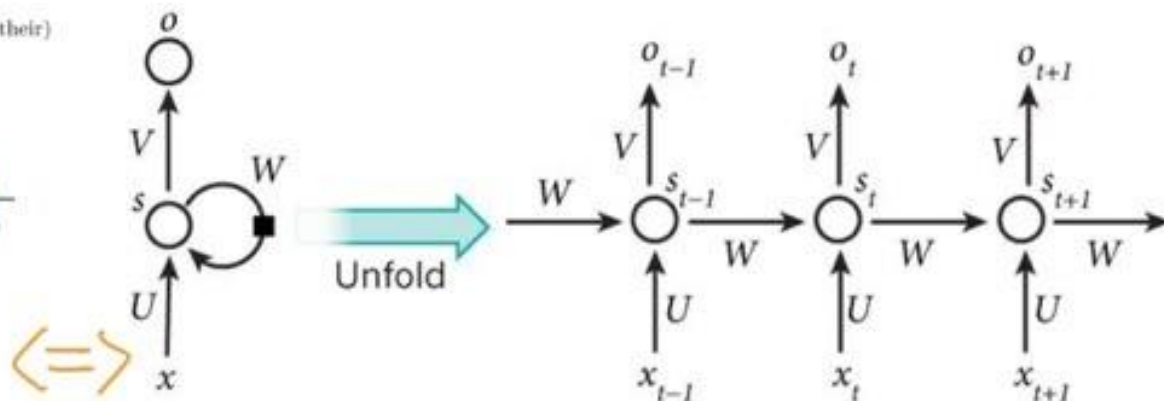
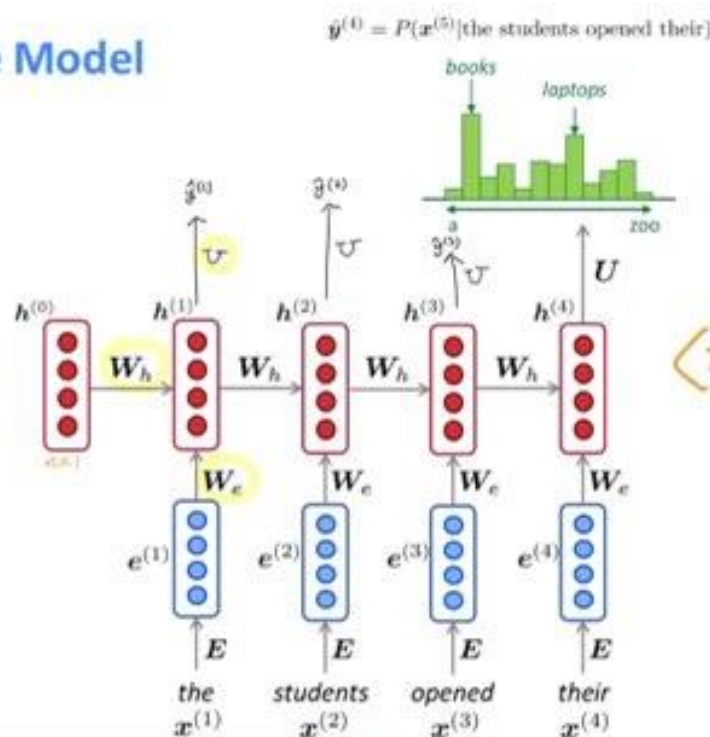
$\mathcal{X} = \{the, students, opened, their\}$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|\mathcal{V}|}$$

--

Note: this input sequence could be much



$$s_t = \tanh(U x_t + W s_{t-1})$$

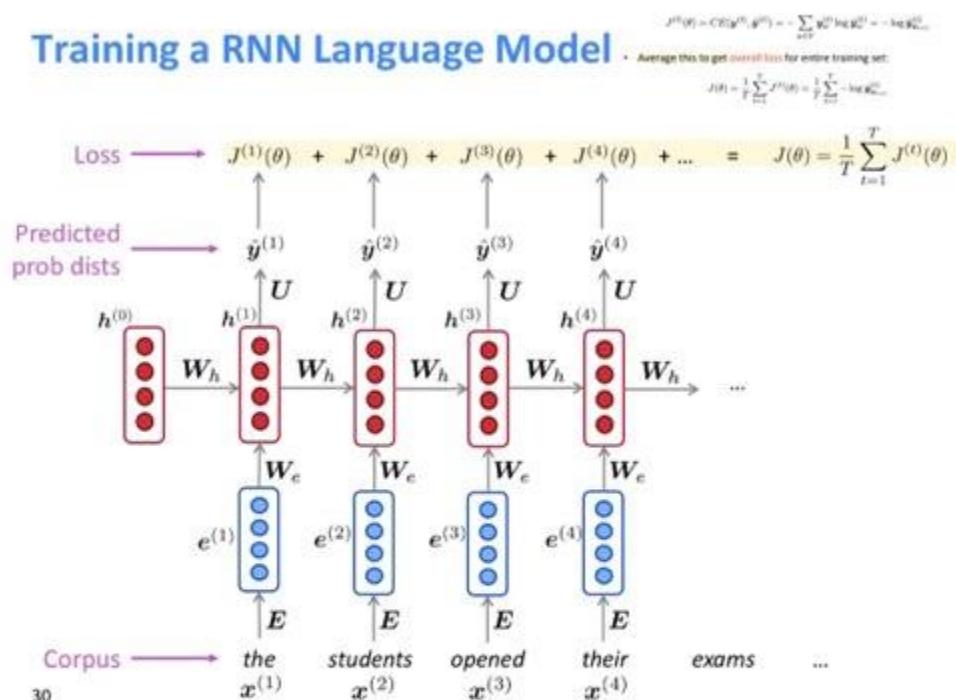
$$\hat{y}_t = \text{softmax}(V s_t)$$

$$\therefore \hat{y}_t = \text{softmax}(z_t) \text{ 도 가능}$$

RNN Forward & Backpropagation

Backpropagation Through Time (BPTT)

Training a RNN Language Model



$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)}$$

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)}$$

||

$$E(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = -\sum_t E_t(y_t, \hat{y}_t)$$

$$= -\sum_t -y_t \log \hat{y}_t$$

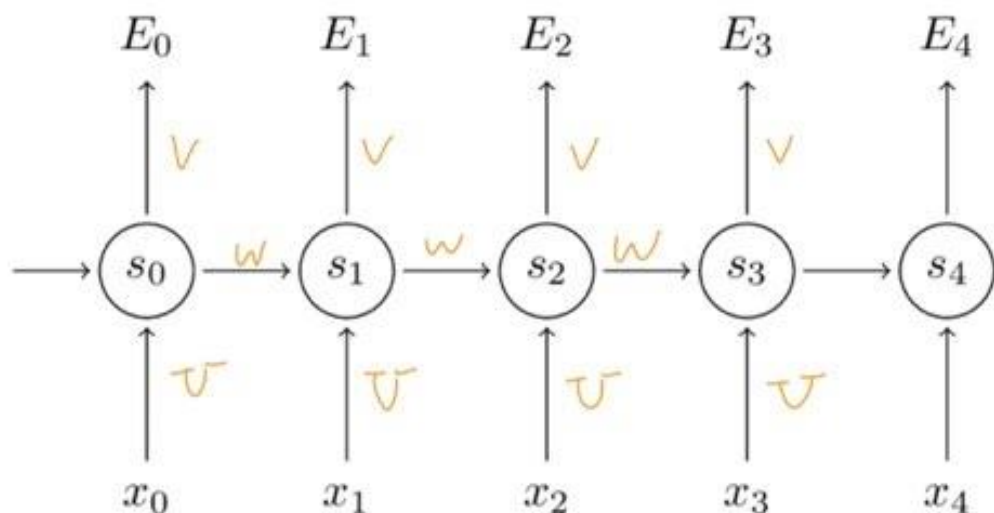
$$\Rightarrow \frac{\partial E}{\partial w} = \sum_t \frac{\partial E_t}{\partial w}$$

(multivariate chain rule)

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(i)}}{\partial W_h} \bigg|_{(i)} \frac{\partial W_h|_{(i)}}{\partial W_h} = 1$$

$$= \sum_{i=1}^t \frac{\partial J^{(i)}}{\partial W_h} \bigg|_{(i)}$$

RNN Forward & Backpropagation



목표: V, W, U 를 업데이트 시키는 것

$$\begin{aligned} \textcircled{1} \quad \frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V} \\ &= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial V} \\ &= (\hat{y}_3 - y_3) \otimes s_3 \end{aligned}$$

$\Rightarrow \frac{\partial E_3}{\partial V}$ 가 현재 시간 step의 \hat{y}_3, y_3, s_3 에만 의존한다는 정의로 단순한 행렬 곱

$$\textcircled{2} \quad \frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \rightarrow s_3(w, s_2(w), s_1(w), s_0(w))$$

↑
multivariable

$s_t = \tanh(Ux_t + Ws_{t-1})$ 로 s_2 에 의존, s_2 는 w 과 s_1 에 의존해서 chain rule이 계속 이어짐.

w 는 우리가 현재 처리 중인 출력 부분까지의 모든 시간 step에 적용되기 때문에 $t=3$ 부터 $t=0$ 까지의 gradient들을 전부 backpropagation 함

RNN 장점과 단점

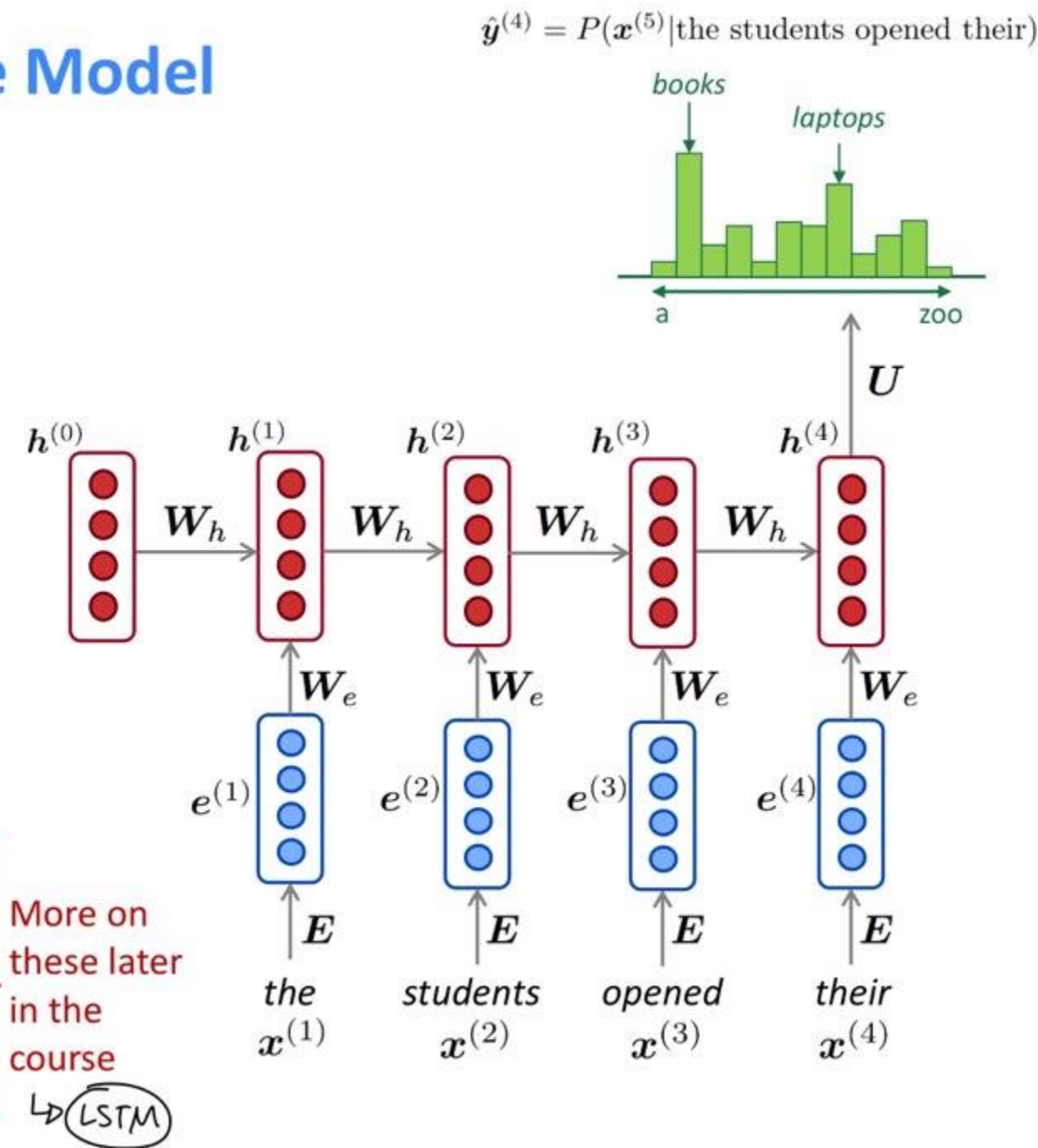
A RNN Language Model

RNN Advantages:

- Can process **any length** input
- Computation for step t can (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is **slow**
- In practice, difficult to access information from **many steps back**



More on these later in the course

↳ LSTM

Perplexity(PPL)

- 정의

the standard evaluation metric for Language Models

(=Language Model을 평가하기위한 표준 기준)

- 식

단어의 수로 정규화(normalization)된 테스트 데이터에 대한 확률의 역수

(정규화를 하는 이유는 corpus가 커질수록 PPL이 작아지는 것을 막기 위해)

- 의미

PPL은 낮을수록 좋음

⇒문장의 확률이 최대화될수록

- 주의점

PPL의 값이 낮다는 것은 테스트 데이터 상에서 높은 정확도를 보인다는 것이지, 사람이 직접 느끼기에 좋은 언어 모델이라는 것을 반드시 의미하진 않는다는 점

언어 모델의 ppl은 테스트 데이터에 의존하므로 두 개 이상의 언어 모델을 비교할 때는 정량적으로 양이 많고, 또한 도메인에 알맞은 동일한 테스트 데이터를 사용해야 신뢰도가 높다는 것

Perplexity(PPL)

PPL은 단어의 수로 정규화(normalization) 된 테스트 데이터에 대한 확률의 역수입니다. PPL을 최소화한다는 것은 문장의 확률을 최대화하는 것과 같습니다. 문장 W 의 길이가 N 이라고 하였을 때의 PPL은 다음과 같습니다.

$$PPL(W) = P(w_1, w_2, w_3, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \dots, w_N)}}$$

문장의 확률에 체인룰(chain rule)을 적용하면 아래와 같습니다.

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \dots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_1, w_2, \dots, w_{i-1})}}$$

여기에 n-gram을 적용해볼 수도 있습니다. 예를 들어 bigram 언어 모델의 경우에는 식이 아래와 같습니다.

$$PPL(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_{i-1})}}$$

⇒J(seta)(각 step t에서의 자코비안들을 모두 더해 평균낸 값)를 최소화될수록

Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^T \left(\frac{1}{P_{\text{LM}}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

하루이유는
perplexity would just
get smaller and smaller
as your corpus get bigger.

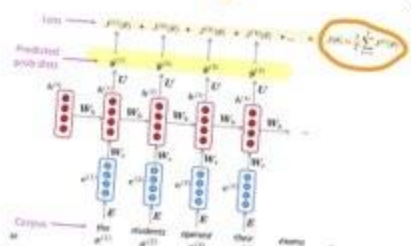
- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$\text{perplexity} = \prod_{t=1}^T \left(\frac{1}{\hat{y}_{x_{t+1}}^{(t)}} \right)^{1/T} = \exp \left(\frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

가분함고

\Leftrightarrow Jacobian $J(\theta)$ 를 최소화하기 위해 학습시키는 것 \Leftrightarrow pp를 최소화하는 것
 \hookrightarrow inverse prob. of the corpus이므로

Lower perplexity is better!



Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^T \left(\frac{1}{P_{LM}(x^{(t+1)} | x^{(1)}, \dots, x^{(t)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^T \left(\frac{1}{\hat{y}_{x_{t+1}}^{(t)}} \right)^{1/T} = \exp \left(\frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

Lower perplexity is better!

41

⇒ 분기계수로서 각 step마다 선택할 수 있는 평균적으로 가능한 경우의 수가 적을수록

PPL은 선택할 수 있는 가능한 경우의 수를 의미하는 분기계수(branching factor)입니다. PPL은 이 언어 모델이 특정 시점에서 평균적으로 몇 개의 선택지를 가지고 고민하고 있는지를 의미합니다. 가령, 언어 모델에 어떤 테스트 데이터를 주고 측정했더니 PPL이 10이 나왔다고 해봅시다. 그렇다면 해당 언어 모델은 테스트 데이터에 대해서 다음 단어를 예측하는 모든 시점(time-step)마다 평균적으로 10개의 단어를 가지고 어떤 것이 정답인지 고민하고 있다고 볼 수 있습니다. 같은 테스트 데이터에 대해서 두 언어 모델의 PPL을 각각 계산 후에 PPL의 값을 비교하면, 두 언어 모델 중 어떤 것이 성능이 좋은지도 판단이 가능합니다. 당연히 PPL이 더 낮은 언어 모델의 성능이 더 좋다고 볼 수 있습니다.

$$PPL(W) = P(w_1, w_2, w_3, \dots, w_N)^{-\frac{1}{N}} = \left(\frac{1}{10} \right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

RNN 다양한 활용

- tagging(part-of-speech tagging, named entity recognition)
- sentence classification(sentiment classification)
- encoder module(question answering, machine translation, etc)
- generate text(conditional language model_speech recognition, machine translation, summarization)

수업 중 나온 좋은 질문들 정리

1.RNN에서 같은 weight을 주는 것은 설계할 때 내가 주는 가정인가요?

not a assumption, it's more a deliberate decision in the design of an RNN

2.RNN에서 같은 weight을 주는 것은 왜 좋은가요?

fixed-window language model에서 단점으로 언급되었듯이 일부 weight이 특정 단어에만 특화되는 문제가 있었는데 weight을 똑같이 하면 모든 단어에 대해서 학습 가증해서 더 generalization,

즉 test에 어떤 문장이 와도 성능이 좋아질 것임

3.단어들은 다 다른데 weight을 똑같이 주면 다양한 단어의 특성이 반영이 안 되는 것 아닌가요?

weight이 아주 큰 matrix라는 점을 감안할 때 많은 단어의 특성을 store 할 수 있고, 단어의 다양성은 weight matrix가 얼마나 그 정보양을 store 할 수 있느냐가 관건

4.What length is the input during training?

Efficiency concern or data based

수업 중 나온 좋은 질문들 정리

5.Does W (one of weights) depend on the length you used?

no. the model size doesn't increase for longer input. 길어지더라도 새로운 weight 이 더 추가되는 게 아니기 때문에

6.How do we choose the dimension of the lowercase Es?

자유선택 or download word2vec의 사이즈에 따라서

7.How you decide to break up your batches affects how you learn?

shuffling it differently each epoch

SGD consider : it should be a good enough approximation that over many steps you will minimize your loss

8.Is it ever practical to combine RNNs with a list of hand-written rules?

ex)hand written rules : don't let your sentence be longer than this many words⇒hacky rules

Beam search가 대표적인 예시로 이후 lecture 에서 배움

[Reference]

서울시립대학교 "비정형자료분석" 박재휘 교수님 강의(2020)

<https://wikidocs.net/21697>