Lecture06. Language Models and Recurrent Neural Networks

Ⅰ. Language Modeling

-이전 단어들이 주어졌을 경우, 그 다음으로 올 단어를 예측하는 문제.

$$P(\boldsymbol{x}^{(t+1)}|\ \boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)})$$

-우리가 예측하려는 단어 x^{t+1}는 현재 존재하는 vocabulary 리스트 중에서 선택하게 되므로 classification task라고도 볼 수 있다.

$$P(\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)}|\ \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)}|\ \boldsymbol{x}^{(T-1)},\ldots,\boldsymbol{x}^{(1)})$$
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)}|\ \boldsymbol{x}^{(t-1)},\ldots,\boldsymbol{x}^{(1)})$$

<전체 text에 대한 probability 계산>

-전체 텍스트를 구성하는 x^1x^T가 나타날 확률은 각각 x^1만 나타날 확률, x^1이 주어졌을 때 x^2가 나타날 확률,..., x^1x^{T-1}이 주어졌을 때 x^T가 나타날 확률의 전체 곱으로 나타낼 수 있고, 해당 값들은 language modeling을 통해서 구할 수 있다.

Ⅱ. n-gram Language Modeling

-Deep learning 이전에 LM을 해결하기 위해서 n-gram 방법을 사용.

-n-gram은 연속적인 단어들의 모음.

the students opened their _____

unigrams: "the", "students", "opened", "their"

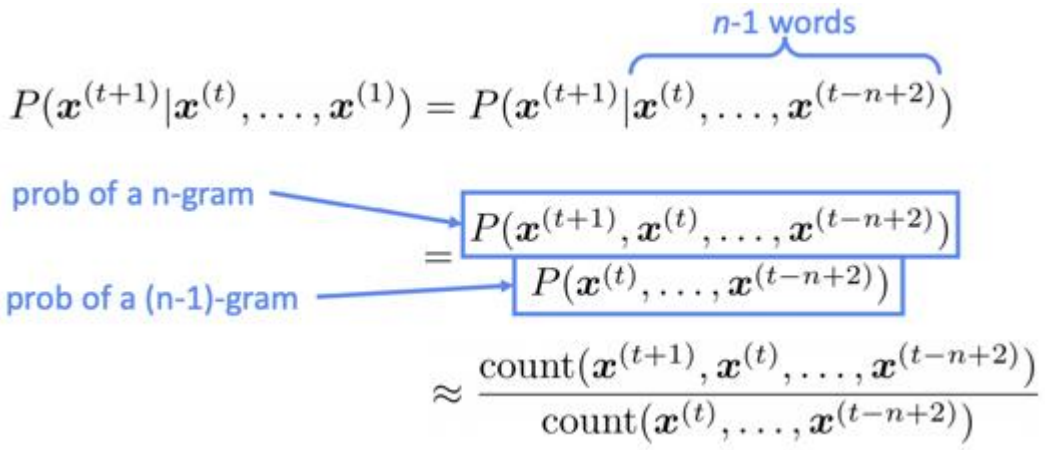
bigrams: "the students", "students opened", "opened their"

trigrams: "the students opened", "students opened their"

4-grams: "the students opened their"

-n-gram LM에서는 큰 corpus에서 해당 n-gram의 통계를 모두 구해서 해당 통계를 기반으로 다음에 나올 단어를 예측한다.

-n-gram LM에서는 Markov assumption이라는 전제를 기본으로 한다. Markov assumption은 t+1번 째 오는 단어는 전체 텍스트에 의존적이지 않고, 이전에 오는 n-1개의 단어에만 의존적이라는 것이다.



1. Sparsity problem

   -분모가 0이 되는 경우, 즉 어떤 단어를 고려해도 'students opened their w'가 단 한 번도 corpus에서 나타나지 않는 경우이다.

   ➔ Smoothing으로 해결 : 모든 단어에 대해 작은 count δ를 추가하여 모든 단어에 대해서 확률이 0이 발생하는 것을 막는다.

   -분모가 0인 경우, 즉 w를 제외하고 'students opened their'라는 문구 자체가 한 번도 나타나지 않는 경우이다.

   ➔ Back-off라는 방식으로 해결 : 앞서 오는 n-1개의 단어를 고려하는 것이 아니라 n-2개의 단어를 사용하는 방식.

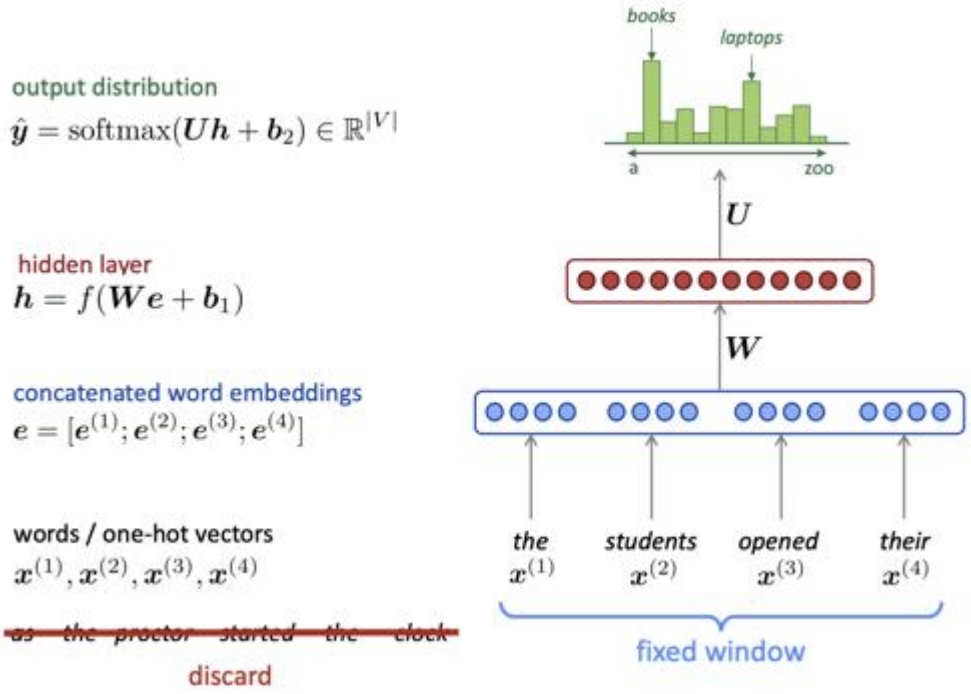
2. Storage problem

   -모든 n-gram에 대한 통계정보를 사용하기 때문에 모든 n-gram에 대한 정보를 저장하고 있어야 한다. corpus가 커지고 n이 커질수록 저장해야 하는 크기가 커지게 된다.

Ⅲ. Fixed-window neural Language Modeling

-n-gram이 위와 같은 문제를 가지고 있어서 neural network를 이용해서 LM을 시도한다. 그 중 하나가 Fixed-window neural Language Modeling.

-Fixed-window이기 때문에 fixed-window 밖의 단어들은 모두 고려하지 않는다. Fixed-window 내에 존재하는 단어들의 word embedding을 구하고 해당 word embedding을 연결한 벡터 e를 생성한다. 벡터 e를 hidden layer에 입력하고 결과로 벡터 h를 제공한다. H는 최종적으로 softmax 함수의 입력으로 주어져서 결과에 대한 확률분포를 제공하게 된다 .확률분포는 가장 높은 확률로 다음에 올 단어를 예측한다.



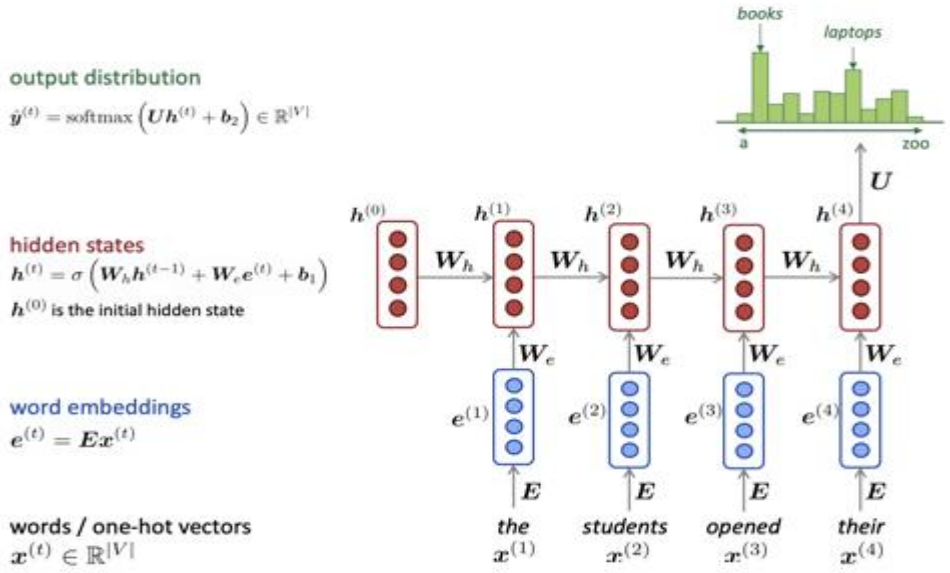
-fixed-window neural LM은 n-gram LM의 문제점들을 모두 해결한다.

-단점 : 전체 문장에 대한 이해를 하기 위해서는 window size가 아무리 커도 충분하지 않다.

더 넓은 범위를 이해하기 위해서 window 크기를 늘리는 경우, hidden layer W의 크기가 함께 커진다.

IV. Recurrent Neural Network(RNN)

-입력으로 들어오는 값의 길이에 구애받지 않고 처리할 수 있어서 fixed-window neural LM의 문제점을 해결할 수 있는 모델이다.

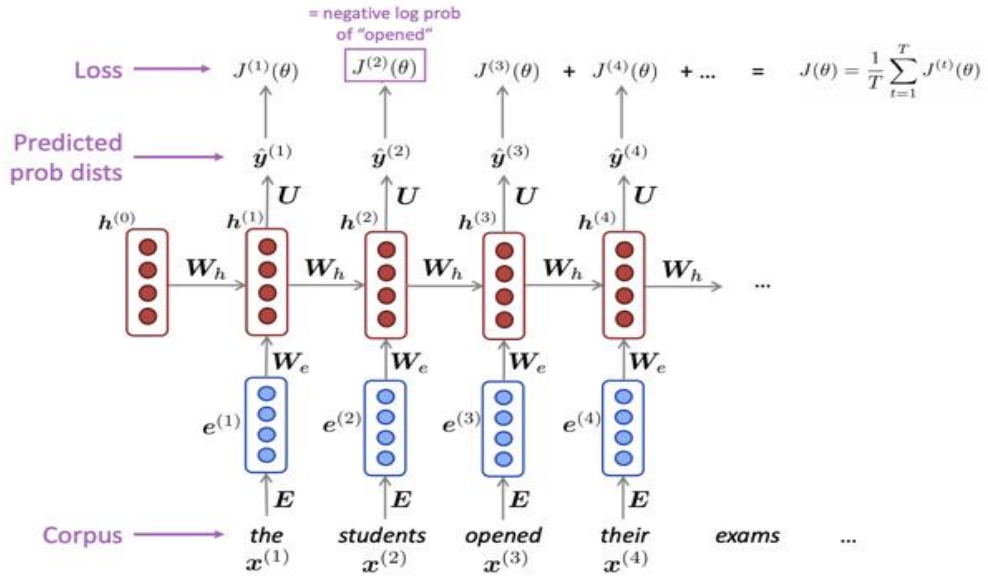


-입력으로 들어오는 값은 x^1, x^2, x^3,...으로 길이에 상관없이 제공할 수 있다. 각 단어는 하나의 timestep에서 입력으로 제공된다. 입력으로 들어온 단어는 해당 단어에 대한 word embedding으로 변환되고 W_e와 연산을 하게 된다. RNN에는 hidden state가 존재하게 되는데 h^0는 initial Hidden state로 zero-vector 또는 모델과 함께 학습되는 벡터를 사용한다. Initial hidden state는 W_h와 연산을 하게 된다.

-두 연산의 결과를 sigmoid 함수를 통해서 다음 timestep의 hidden state인 h^1이 생성된다. 다음 timestep에서는 두 번째 단어 벡터가 제공되고 동일한 연상을 통해서 두 번째 hidden state인 h^2가 생성된다. 마지막 단어에 대해서도 동일하게 연산이 진행되고 최종 hidden state는 softmax 함수의 입력으로 제공되어 다음으로 나올 단어의 확률분포를 제공한다.

-전체 과정에서 동일한 W_h, W_e를 사용하고 해당 matrix는 모든 timestep에 동일하게 적용된다.

-RNN 학습은 cross-entropy loss를 최소화하는 방향으로 진행된다. 각 timestep마다 모델이 예측한 값과 실제 정답을 이용해서 손실값 J를 구한다. 모든 timestep에 대한 손실 값을 구하고, 전체 모델의 손실 값은 각 timestep의 손실 값의 평균을 이용한다.



-RNN을 이용해서 text generation을 할 수 있다. LM과 동일한 방식으로 진행되는데, 한가지 다른 점은 한 timestep의 결과로 sampling된 값을 다음 timestep의 입력으로 제공한다는 점이다. 이전에 생성된 단어를 입력으로 제공해서 다음 단어를 생성하는 방식으로 text generation을 진행.

-장점 : 입력 값의 길이에 상관 없이 처리 가능. 이론적으로는 몇 단계 이전의 정보를 기억해서 결과에 반영할 수 있다. 입력의 길이가 길어져도 모델의 크기(W_h, W_e)는 변하지 않는다.

-단점 : 연산 속도가 느리다. 이전 timestep의 연산을 완료해야 다음 timestep의 연산이 가능하기 때문에, 병렬 처리가 불가능하다. 실제로 사용했을 때, 과거의 내용을 그렇게 잘 기억하지 못한다.

V. Evaluating Language Model

-LM은 perplexity를 이용해서 성능을 평가한다

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

-perplexity는 전체 corpus의 probability의 역이다. 1/T를 통해서 normalization을 진행하는데, 단어가 많아지는 경우 perplexity가 작아지기 때문이다.

-LM의 성능이 좋으면 perplexity는 낮다.