

# Language model, RNN, RNN based model

모바일 플랫폼TF팀 이경임

# References

## 위키독스

- 언어 모델이란?
- 통계적 언어모델
- N-gram 언어 모델
- 순환신경망
- RNN 언어모델

## 실습

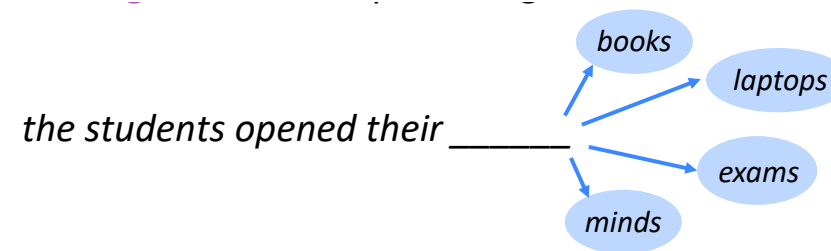
- 글자 단위 RNN

## 기타 자료

- 한국어 임베딩 - 이기창
- CS224n - Recurrent Neural Networks and Language Models

# 언어모델이란?

- 자연어의 통계적 패턴
  - 문장에 어떤 단어가 (많이) 쓰였는가 : 백오브워즈 가정
  - 단어가 어떤 순서로 등장하는가 : 언어모델
  - 어떤 단어가 같이 나타났는가 : 분포 가정



<https://wikidocs.net/21668>

## 1) 관련 주제 기억나니 : DTM, TF-IDF

이전 시간에 저희는 계속해서 자연어를 실제 컴퓨터가 이해할 수 있는 벡터로 변환하기 위한 다양한 기법들을 배웠습니다(임베딩). 임베딩의 핵심은 실제 사람이 사용하는 자연어의 어떠한 정보들을 임베딩 된 결과가 반영할 수 있도록 하는 것입니다.

이때(임베딩을 만들때) 사용하는 통계적 정보는 크게 3가지가 있습니다. 1~3 입니다. 1은 백오브 워즈 가정입니다. 단어의 순서는 무시하고 사용 여부/빈도를 중요시 합니다. 관련하여 TF-IDF라는 주제를 다뤘었죠. 백오브워즈 가정의 대척점에는 단어가 어떤 순서로 등장하는가 인 언어모델이 있습니다.

# 언어모델이란?

- 딥러닝의 발전 이전에도 있었던 개념
- 언어를 모델링하고자 **단어 시퀀스에 확률을 부여**하는 모델이다.
- 잘 학습된 언어모델은 어떤 문장이 더 “자연스러운지”, 또한 주어진 시퀀스 다음에는 무엇이 오는게 자연스러운지를 알 수 있다.
- 단어가 N개 주어진 상황에서 언어모델은 N개 단어가 동시에 나타날 확률, 즉  $P(w_1, w_2, w_3 \dots w_n)$ 을 반환합니다.

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$$

where  $\mathbf{x}^{(t+1)}$  can be any word in the vocabulary  $V = \{w_1, \dots, w_{|V|}\}$

<https://wikidocs.net/21668>

언어모델은 딥러닝의 발전 이전부터 존재했던 개념으로, 언어를 모델링하고자 단어 시퀀스에 확률을 부여하는 모델입니다. 아까 백오브워즈 가정의 대척점에 언어모델이 있다고 했는데, 그 이유를 여기서 알 수 있죠. 즉, 단어의 등장 순서를 무시하는 백오브워즈와 달리 언어모델은 시퀀스 정보를 명시적으로 학습합니다.

???

이때 저번 스터디에서 얘기한 것 처럼 언어모델은 비지도학습인 자연어를 학습하기 위해/확률을 할당하기 위해 이전 단어들이 단어들이 주어졌을 때 다음 단어를 예측하도록 합니다. 또한 각 단어시퀀스에 확률은 어떻게 계산? >> 카운트 기반 접근방식 사용한다.

# 단어 시퀀스에서의 확률 할당

## A. 단어 시퀀스의 확률

하나의 단어를  $w$ , 단어 시퀀스를 대문자  $W$ 라고 한다면,  $n$ 개의 단어가 등장하는 단어 시퀀스  $W$ 의 확률은 다음과 같습니다.

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n)$$

## B. 다음 단어 등장 확률

이제 다음 단어 등장 확률을 식으로 표현해보겠습니다.  $n-1$ 개의 단어가 나열된 상태에서  $n$ 번째 단어의 확률은 다음과 같습니다.

$$P(w_n | w_1, \dots, w_{n-1})$$

|의 기호는 조건부 확률(conditional probability)을 의미합니다.

<https://wikidocs.net/21668>

말했듯이 언어모델은 단어시퀀스에 확률 할당하는 모델. 언어모델이 확률 할당하는 방법은 간단한 조건부 확률.  $W$ 가 전체 단어 시퀀스일때  $W$ 가 등장할 확률은  $W$ 를 구성하는 단어  $N$ 개가 동시에 등장할 확률과 같습니다. => 아나그램은?

$n$ 번째 단어 등장 확률은 앞의  $n-1$ 개 등장 후  $n$ 이 등장할 조건부 확률이지.

# 통계적 언어모델(SLM)

- 딥러닝의 발전 이전에도 있었던 개념, 전통적 언어 모델
- 문장의 확률을 구하기 위해 **조건부 확률** 사용
- 이때의 확률값은 **카운트에 기반해 계산**

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

This is what our LM provides

<https://wikidocs.net/21687>

통계적 언어모델은 언어모델의 전통적인 접근 방법입니다. 이러한 통계적 언어모델에서 언어를 모델링하는 방식은 다음과 같습니다.

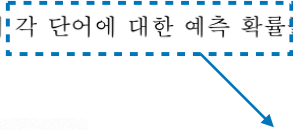
# 통계적 언어모델(SLM)

$$P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{n=1}^n P(w_n | w_1, \dots, w_{n-1})$$

“An adorable little boy is spreading smiles” 문장이 등장할 확률

$$\begin{aligned} P(\text{An adorable little boy is spreading smiles}) = \\ P(\text{An}) \times P(\text{adorable}|\text{An}) \times P(\text{little}|\text{An adorable}) \times P(\text{boy}|\text{An adorable little}) \times P(\text{is}|\text{An adorable little boy}) \\ \times P(\text{spreading}|\text{An adorable little boy is}) \times P(\text{smiles}|\text{An adorable little boy is spreading}) \end{aligned}$$

문장의 확률을 구하기 위해서, 각 단어에 대한 예측 확률들을 곱합니다.


$$P(\text{is}|\text{An adorable little boy}) = \frac{\text{count}(\text{An adorable little boy is})}{\text{count}(\text{An adorable little boy})}$$

<https://wikidocs.net/21687>

예를들어 문장 ~에 대한 통계적 언어모델은 다음과 같이 확률을 구합니다.

이처럼 각 단어에 대한 예측확률들을 곱하는 것. 그럼 이 확률은 어떻게 구할까?? >> 아까 말했듯이 카운트 기반의 접근. 여기 이 수식이 의미하는 바는 무엇이죠? An ~ boy 가 나왔을 때 is가 나올 확률을 의미하는 것.

전체 An adorable little boy 100번 등장했는데 ‘an adorable little boy is’ 가 40번 등장했으면 확률은 0.4

# N-gram 언어 모델

- 통계기반 언어모델의 일종. SLM과 같이 카운트 기반 통계적 접근을 사용한다.
- 전통적 SLM과 달리 이전에 등장한 모든 단어가 아닌 **일부 단어만 고려**하는 방법을 사용한다.
  - n-gram에서의 n은 코퍼스 내 단어들을 n개씩 묶어서 빈도를 학습했음을 의미한다.
- **이전 n-1개의 단어를 보고 n번째 단어를 예측**하는 방식
- 임의의 개수만큼의 이전 단어만 참고하여 확률을 근사
  - 코퍼스에서 해당 단어시퀀스를 카운트할 확률이 높아진다.

<https://wikidocs.net/21692>

2) 일부 단어만 고려 : 장점이 뭘까?

카운트할 확률 높/카운트 못하면 문제 뭘까?

n-gram은 slm의 일종. 통계적 접근 사용한다. 다른점이라면 일부 단어만 고려하는 것. 왜 그럴까? >> 직관적으로 생각했을 때도 긴 전체 문장이 훈련 코퍼스 내에 있을 확률보다 짧은 단어 시퀀스가 코퍼스 안에 있을 확률이 높다. 이렇게 해서 코퍼스에서 해당 단어 시퀀스 카운트할 확률을 높인다.



# N-gram 언어 모델 예시

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ students opened their \_\_\_\_\_  
discard condition on this

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
  - “students opened their books” occurred 400 times
    - $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
  - “students opened their exams” occurred 100 times
    - $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$
- Should we have discarded the “proctor” context?

21

P(명작이다 | 영원히, 기억될, 최고의) (유사) P(명작이다 | 기억될, 최고의)

= Freq(기억될, 최고의, 명작이다) / Freq(명작이다)

= 17 / 298

# N-gram 언어 모델 예시

표현	빈도
영원히	104
기억될	29
최고의	3503
명작이다	298
영원히 기억될	7
기억될 최고의	1
최고의 명작이다	23
기억될 최고의 명작이다	17
영원히 기억될 최고의 명작이다	0

- 영원히 기억될 최고의 시퀀스 뒤에 `명작이다` 라는 단어가 올 확률을 **trigram**으로 근사해보면 얼마일까?

<https://wikidocs.net/21692>

$P(\text{명작이다} \mid \text{영원히, 기억될, 최고의})$  (유사)  $P(\text{명작이다} \mid \text{기억될, 최고의})$

$= \text{Freq}(\text{기억될, 최고의, 명작이다}) / \text{Freq}(\text{명작이다})$

$= 17 / 298$

# N-gram 언어모델의 한계

- 희소문제(Sparsity problem)
  - 카운트 기반 접근방식의 본질적 한계
  - 코퍼스 내에 단어시퀀스가 없을 확률은 여전히 존재
- n의 선택은 trade-off
  - n크기를 키우면 : 예측 정확도 상승 but 희소문제 증가, 모델사이즈 증가
  - n크기를 줄이면 : 희소문제 감소 but 예측 정확도 감소
- long-term dependency : 정해진 개수의 이전 토큰만을 반영하므로 고려할 수 있는 시퀀스 범위 한정됨. >> 모델의 정확도와 연관

<https://wikidocs.net/21692>

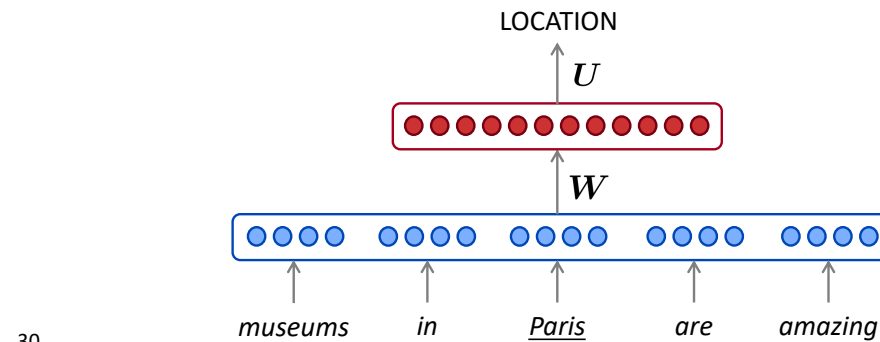
여전히 희소문제는 존재한다. 코퍼스 내에 단어시퀀스가 없을(카운트하지 못할) 확률은 여전히 존재. 하나라도 0 되면 죄다 0됨/아예 시퀀스 카운트하지 못할수도. n의 선택은 trade-off : n의 크기를 키우면 예측의 정확도는 높아지지만, 코퍼스에서 해당 n개의 시퀀스를 카운트할 확률은 낮아짐(희소문제 증가), 모델사이즈 커짐.

카운트 기반 접근은 그 방식상 본질적인 한계를 갖는다.

이를 극복하기 위해 다양한 방법 시도되었지만(백오프, 스무딩) 본질적인 n-gram 언어모델(고정된 개수의 단어만을 입력으로 받아야한다)에 대한 취약점은 해결하지 못함.

# NNLM : 신경망 기반 언어모델

- Recall the Language Modeling task:
  - Input: sequence of words  $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
  - Output: prob dist of the next word  $P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$
- How about a **window-based neural model**?
  - We saw this applied to Named Entity Recognition in Lecture 3:



<https://wikidocs.net/21692>

NNLM(Neural Net Language Model)을 통해 언어 모델 또한 단어의 유사도를 학습 할 수 있도록 설계.

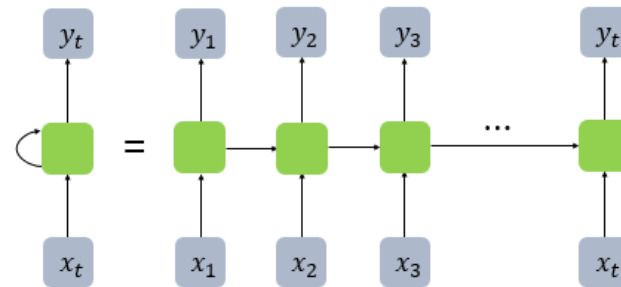
훈련 코퍼스에 없는 단어 시퀀스도 예측을 통해 유사한 단어 시퀀스를 참고하여 확률을 계산해낸다.

워드 임베딩의 아이디어이기도 한다. CBOW형태와 유사 but 앞의 몇개 단어만 반영

4) 단어를 continuous한 밀집벡터의 형태로 표현하여 희소문제를 해결(?)

# RNNLM : RNN 언어모델

- RNN을 이용해 구현한 언어 모델
- n-gram, NNLM의 한계 : 고정된 개수의 단어를 입력으로 받는다
- **timestep**의 개념이 도입된 RNN 언어모델은 입력의 길이를 고정할 필요가 없다.



<https://wikidocs.net/46496>

RNNLM은 RNN을 이용해 구현한 언어모델. Ngram, nnlm의 고정된 개수 한계를 해결할 수 있다.

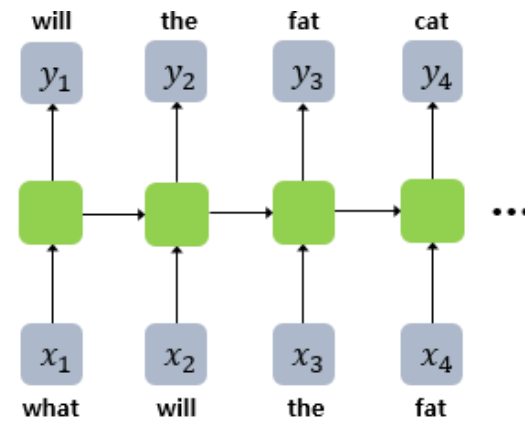
이전에 배웠던 순환신경망 RNN이 이전에 다뤘던 신경망 모델과 가장 달랐던 점은 뭐였죠? 바로 타임스텝의 개념을 갖고 있다는 점. 익숙한 이미지일텐데요. 아래 그림처럼 RNN의 은닉층에서 메모리셀은 활성화 함수를 통해 나온 결과값을 출력층 방향 뿐만 아니라 다시 은닉층 노드 다음 계산의 입력으로 보낸다는 특징을 갖게 됩니다. 이때 다음 시점의 자신에게 보내는 값을 은닉상태라고 부르다고 했었죠.

RNN 언어모델에서는 이러한 타임스텝의 개념을 도입함으로써 입력의 길이를 고정할 필요가 없게되었습니다.

왜?

# RNNLM의 사용

예문 : What will the fat cat sit on



- 주어진 단어 시퀀스로부터 다음 단어를 예측하는 모델.
- **이전 시점의 출력**(예측값)이 **현재 시점의 입력**이 된다.
- RNNLM은 what을 입력받으면, will을 예측하고 이 will은 다음 시점의 입력이 되어 the를 예측한다.
- 이것이 반복되어 네번째 시점의 **cat**은 앞서 나온 **what, will, the, fat**이라는 시퀀스로 인해 결정된 단어가 된다.

<https://wikidocs.net/46496>

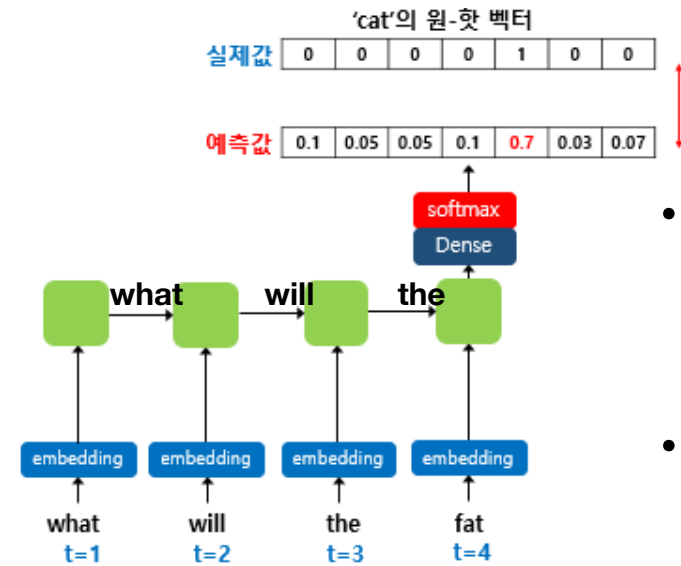
RNNLM이 언어 모델링을 학습하는 과정을 보겠습니다. 이해를 위해 매우 간소화 된 형태로 설명합니다.

예문 : "what will the fat cat sit on"

예를 들어 훈련 코퍼스에 위와 같은 문장이 있다고 해봅시다. 언어 모델은 주어진 단어 시퀀스로부터 다음 단어를 예측하는 모델입니다. 아래의 그림은 RNNLM이 어떻게 이전 시점의 단어들과 현재 시점의 단어로 다음 단어를 예측하는지를 보여줍니다.

RNNLM은 what을 입력받으면, will을 예측하고 이 will은 다음 시점의 입력이 되어 the를 예측합니다. 그리고 the는 또 다시 다음 시점의 입력이 되고 해당 시점에서는 fat을 예측합니다. 그리고 이 또한 다시 다음 시점의 입력이 됩니다. 결과적으로 세번째 시점에서 fat은 앞서 나온 what, will, the라는 시퀀스로 인해 결정된 단어이며, 네번째 시점의 cat은 앞서 나온 what, will, the, fat이라는 시퀀스로 인해 결정된 단어입니다.

# RNNLM의 훈련



- 훈련시에는 예측값이 아닌 실제 알고있는 정답을  $t+1$  시점의 입력으로 사용한다.(교사강요)
- 보다 빠르고 효과적인 모델 훈련을 위한 기법

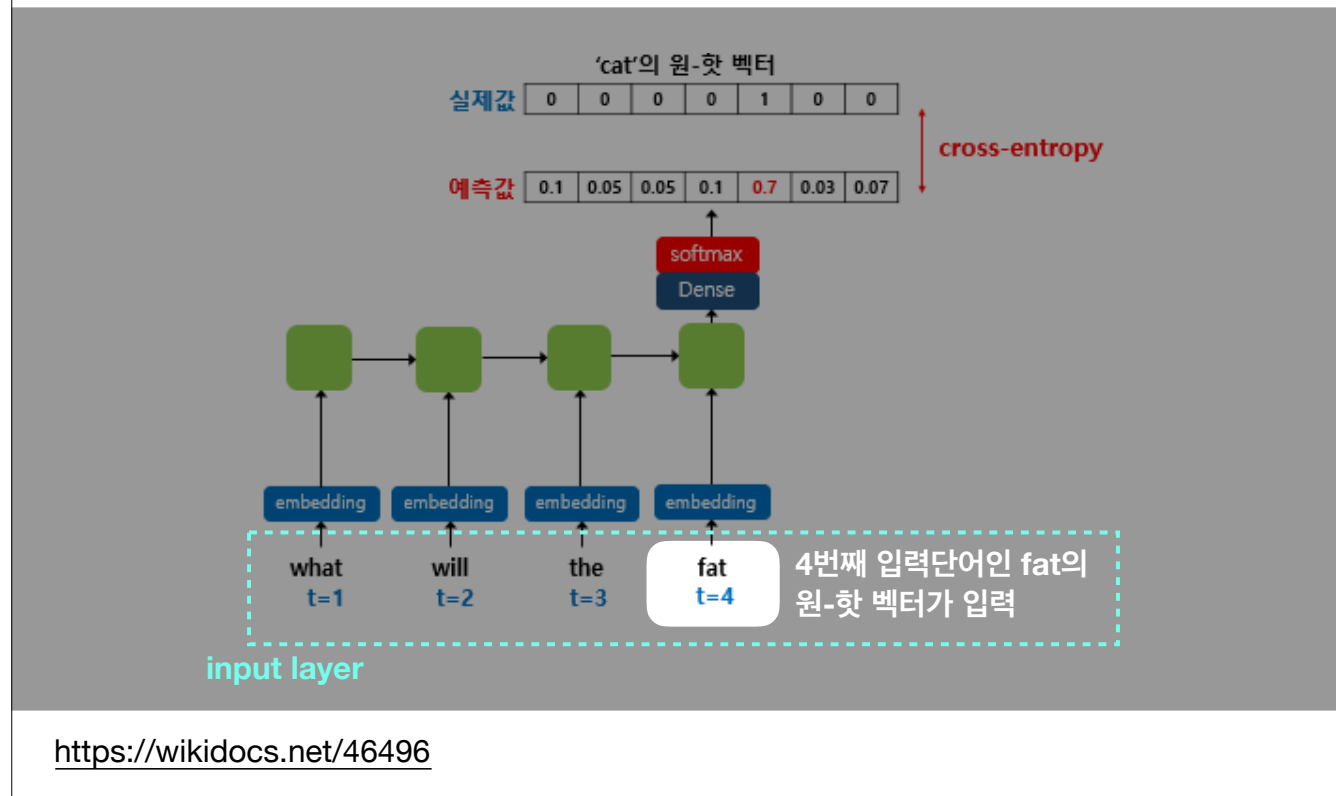
<https://wikidocs.net/46496>

이러한 RNN 훈련 기법을 교사 강요(teacher forcing)라고 합니다.

훈련할 때 교사 강요를 사용할 경우, 모델이  $t$  시점에서 예측한 값을  $t+1$  시점에 입력으로 사용하지 않고,  $t$  시점의 레이블, 즉, 실제 알고있는 정답을  $t+1$  시점의 입력으로 사용합니다.

훈련 과정에서도 이전 시점의 출력을 다음 시점의 입력으로 사용하면서 훈련 시킬 수도 있지만 이는 한 번 잘못 예측하면 뒤에서의 예측까지 영향을 미쳐 훈련 시간이 느려지게 되므로 교사 강요를 사용하여 RNN을 좀 더 빠르고 효과적으로 훈련시킬 수 있습니다.

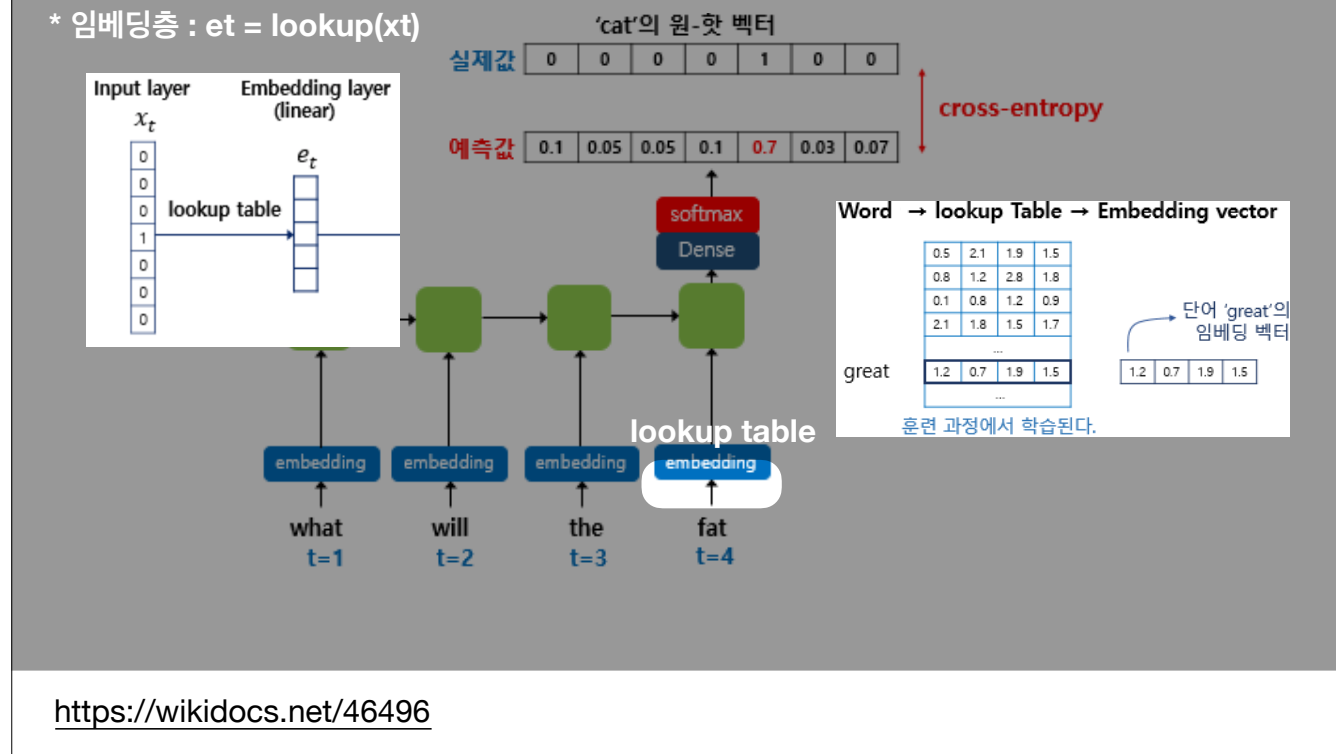
# RNNLM의 구조



RNNLM의 구조를 보겠습니다. RNNLM은 위의 그림과 같이 총 4개의 층(layer)으로 이루어진 인공 신경망입니다. 우선 입력층(input layer)을 봅시다. RNNLM의 현 시점(timestep)은 4로 가정합니다. 그래서 4번째 입력 단어인 fat의 원-핫 벡터가 입력이 됩니다.



# RNNLM의 구조



현 시점의 입력 단어의 원-핫 벡터  $x_t$   
 5) 단어집합 크기인 이유는?

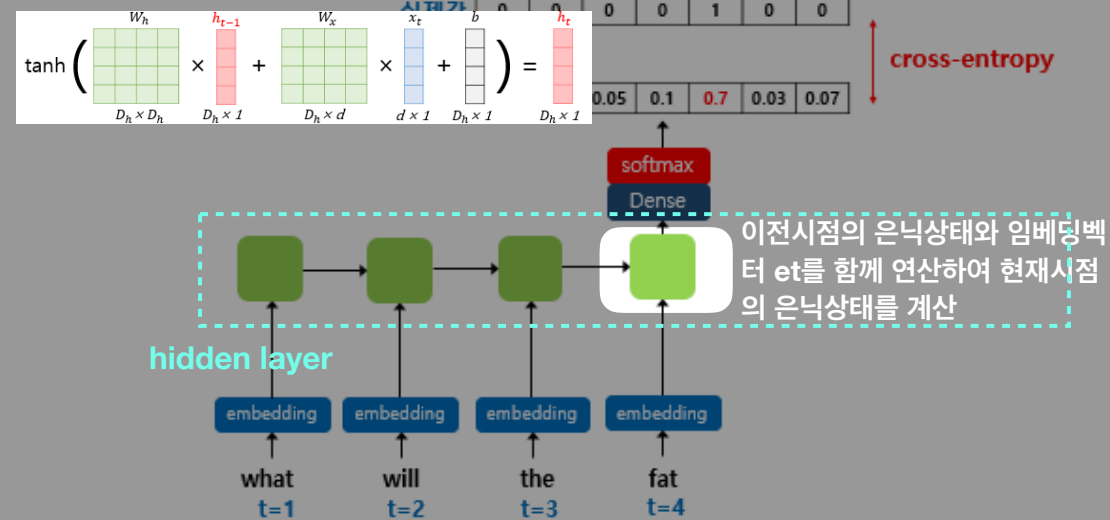
를 입력 받은 RNNLM은 우선 임베딩층(embedding layer)을 지납니다. 이 임베딩층은 기본적으로 NNLM 챕터에서 배운 투사층(projection layer)입니다. NNLM 챕터에서는 룩업 테이블을 수행하는 층을 투사층이라고 표현했지만, 이미 투사층의 결과로 얻는 벡터를 임베딩 벡터라고 부른다고 NNLM 챕터에서 학습하였으므로, 앞으로는 임베딩 벡터를 얻는 투사층을 임베딩층(embedding layer)이라는 표현을 사용할 겁니다.

단어 집합의 크기가  $V$ 일 때, 임베딩 벡터의 크기를  $M$ 으로 설정하면, 각 입력 단어들은 임베딩층에서  $V \times M$  크기의 임베딩 행렬과 곱해집니다. 여기서  $V$ 는 단어 집합의 크기를 의미합니다. 만약 원-핫 벡터의 차원이 7이고,  $M$ 이 5라면 임베딩 행렬은  $7 \times 5$  행렬이 됩니다. 그리고 이 임베딩 행렬은 역전파 과정에서 다른 가중치들과 함께 학습됩니다. 이는 NNLM 챕터에서 이미 배운 개념입니다.

임베딩층 :  $e_t = \text{lookup}(x_t)$

# RNNLM의 구조

\* 은닉층 :  $\tanh(W_x * e_t + W_h * h_{t-1} + b)$  cat'의 원-핫 벡터



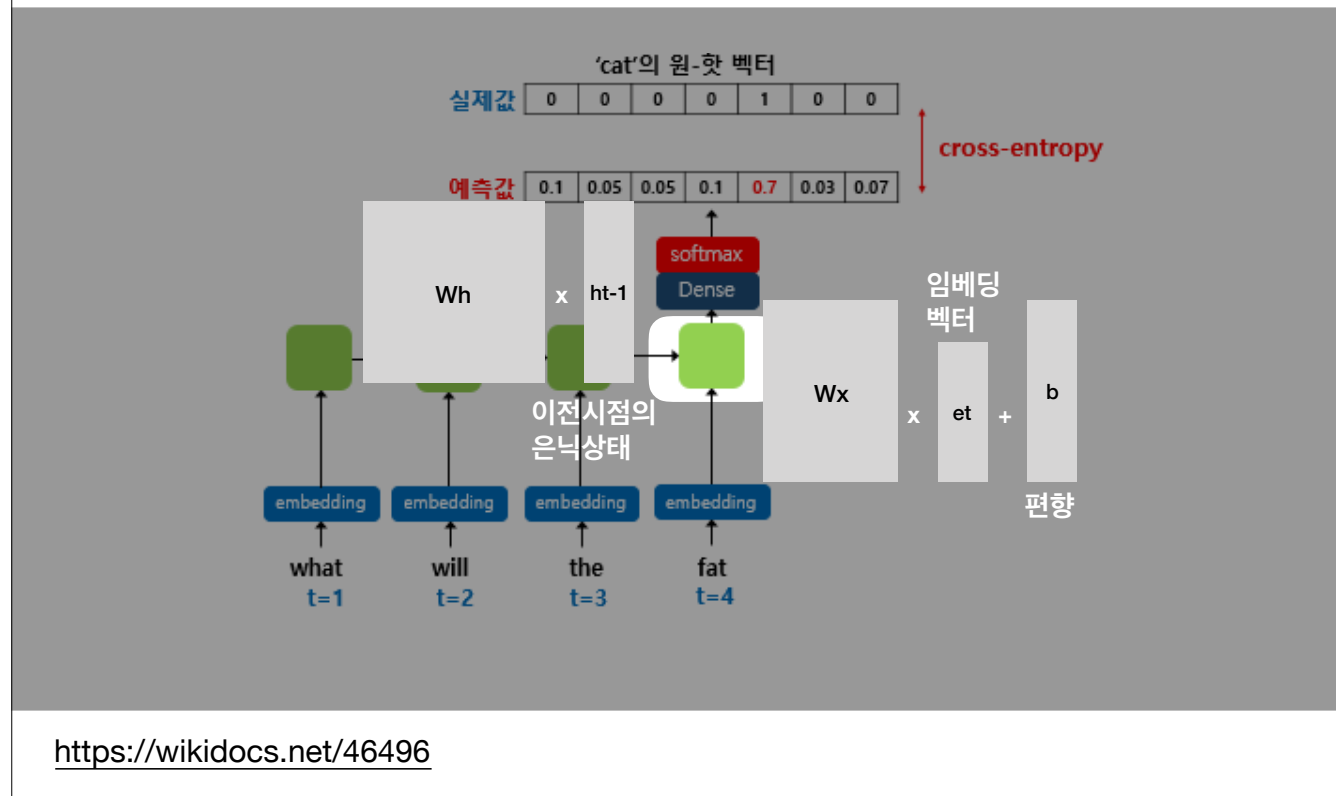
<https://wikidocs.net/46496>

여기서부터는 다시 RNN을 복습하는 것과 같습니다.

이 임베딩 벡터는 은닉층에서 이전 시점의 은닉 상태인  $h_{t-1}$ 과 함께 다음의 연산을 하여 현재 시점의 은닉 상태  $h_t$ 를 계산하게 됩니다.

은닉층 :  $h_t = \tanh(W_x e_t + W_h h_{t-1} + b)$

# RNNLM의 구조



여기서부터는 다시 RNN을 복습하는 것과 같습니다.

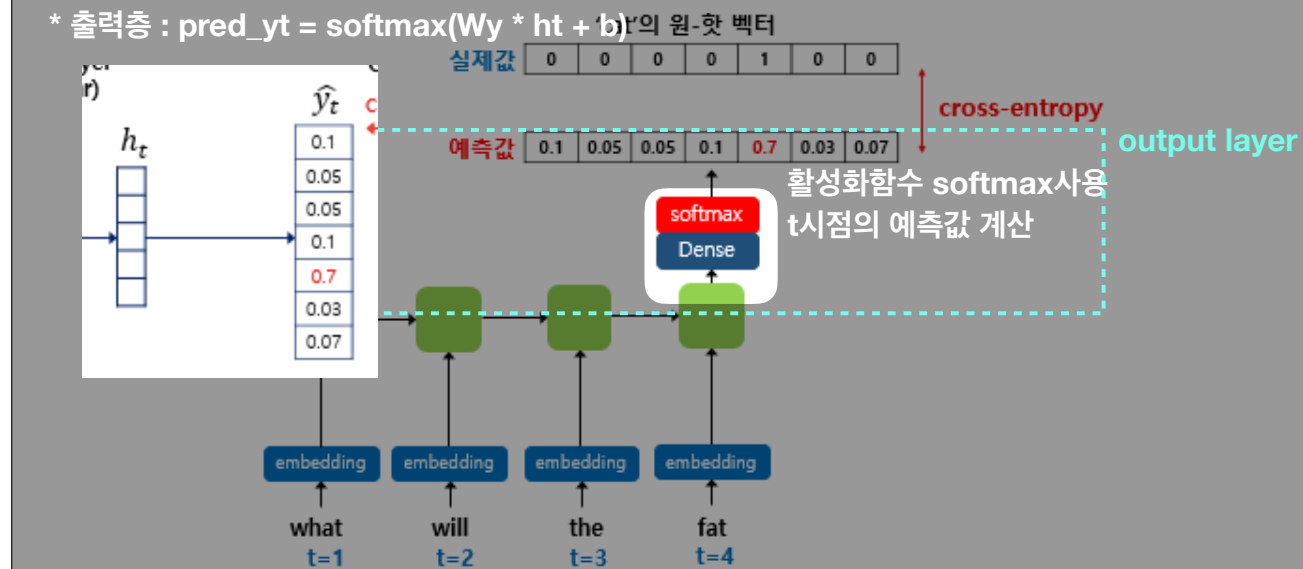
이 임베딩 벡터는 은닉층에서 이전 시점의 은닉 상태인  $h_{t-1}$ 과 함께 다음의 연산을 하여 현재 시점의 은닉 상태  $h_t$ 를 계산하게 됩니다.

세로길이 모두  $h_t$ 의 길이 = 은닉층 차원 = 단어집합의 크기

은닉층 :  $h_t = \tanh(Wx_t + Whh_{t-1} + b)$

=>  $\tanh()$  : 은닉층의 활성화함수. 이전에는 렐루썼었지?

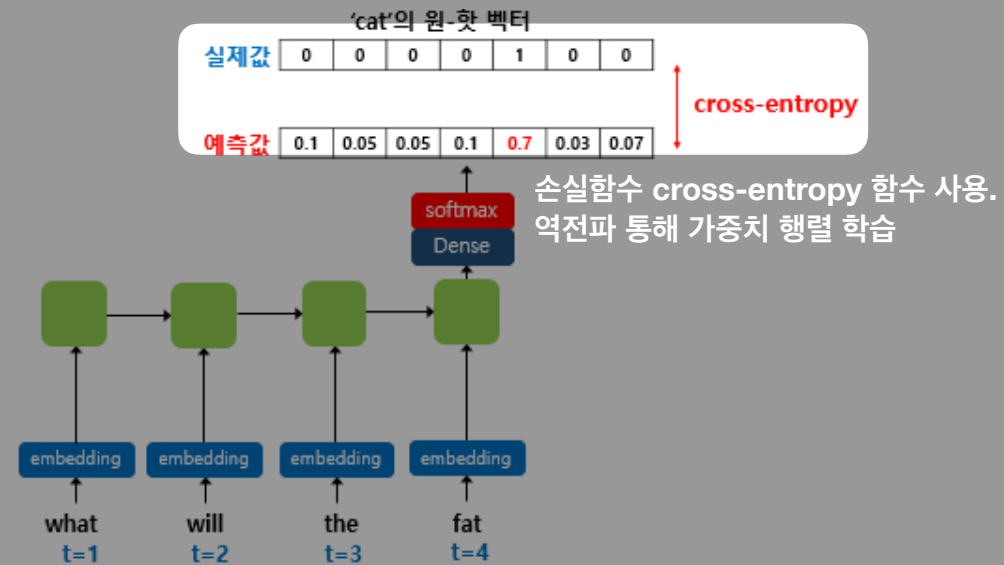
# RNNLM의 구조



<https://wikidocs.net/46496>

출력층에서는 활성화 함수로 소프트맥스(softmax) 함수를 사용하는데, V차원의 벡터는 소프트맥스 함수를 지나면서 각 원소는 0과 1사이의 실수값을 가지며 총 합은 1이 되는 상태로 바뀝니다. 이렇게 나온 벡터를 RNNLM의 t시점의 예측값이라는 의미에서  $\hat{y}_t$  라고 합니다. 이를 식으로 표현하면 아래와 같습니다.

# RNNLM의 구조



<https://wikidocs.net/46496>

그리고  $y_t^*$

는 실제값. 즉, 실제 정답에 해당되는 단어인 원-핫 벡터의 값에 가까워져야 합니다. 실제값에 해당되는 다음 단어를  $y$

라고 했을 때, 이 두 벡터가 가까워지게 하기위해서 RNNLM는 손실 함수로 cross-entropy 함수를 사용합니다. 그리고 역전파가 이루어지면서 가중치 행렬들이 학습되는데, 이 과정에서 임베딩 벡터값들도 학습이 됩니다.

6) 그럼 최종적으로 RNNLM에서 학습되는 행렬들의 개수는?

# RNNLM의 실습

순환 신경망을 활용한 문자열 생성

[https://www.tensorflow.org/tutorials/text/text\\_generation](https://www.tensorflow.org/tutorials/text/text_generation)

이러한 RNN 훈련 기법을 교사 강요(teacher forcing)라고 합니다.

훈련할 때 교사 강요를 사용할 경우, 모델이  $t$  시점에서 예측한 값을  $t+1$  시점에 입력으로 사용하지 않고,  $t$  시점의 레이블. 즉, 실제 알고있는 정답을  $t+1$  시점의 입력으로 사용합니다.

훈련 과정에서도 이전 시점의 출력을 다음 시점의 입력으로 사용하면서 훈련 시킬 수도 있지만 이는 한 번 잘못 예측하면 뒤에서의 예측까지 영향을 미쳐 훈련 시간이 느려지게 되므로 교사 강요를 사용하여 RNN을 좀 더 빠르고 효과적으로 훈련시킬 수 있습니다.