협성대학교 소프트웨어공학과

인공지능기초

중간고사 대비 9~14 주차 강의 요약.
(<밑바닥부터 시작하는 딥러닝 2> 요약)

이시헌

2024-6-10

9주차 강의 정리.

2장부터 시작한다. (1장: 1권 복습)

1. 2장
   A. 2장은 '통계 기반 기법'으로 자연어를 처리하는 것을 설명한다.
   B. 3장은 '추론 기반 기법'을 설명한다.

2. 자연어 처리의 시작
   A. 단어의 의미를 파악해야 자연어를 처리할 수 있다.
   B. 즉, 컴퓨터에게 단어의 의미를 이해시켜야 한다.
   C. 대표적인 단어 의미 파악 기법
      ① 시소러스(유의어 사전)를 활용.
      ② 통계 기반 기법.
      ③ 추론 기반 기법.

3. 시소러스
   A. 유의어 사전이다.
   B. (사람이) 뜻이 같거나 비슷한 단어를 찾아내어 유의어로 정의한 사전이다.
      ① 단어 사이의 상위/하위, 전체/부분 등의 관계를 그래프 구조로 정의하는 것도 가능하다.
   C. 문제점
      ① 사람이 직접 레이블링 해야 한다.
         -〉 고비용, 느림 + 오류 존재 + 시대 변화에 대응 어려움 + 단어의 미묘한 차이(용법 등) 표현 불가.
         -〉 위와 같은 문제를 피하기 위해 '통계 기반 기법'과 '추론 기반 기법'을 사용한다.

4. 단어의 분산 표현

= 단어를 다차원 벡터로 표현하는 것.

= 단어를 고정 길이의 밀집 벡터로 표현하는 것.

= 데이터가 가진 속성을 수치로 표현하는 것.

A. 예시

비색: (R, G, B) = (170, 33, 22) <- 3차원 벡터로 '색상'데이터가 표현됨.

B. 밀집 벡터 (dence vector)

= 대부분의 원소가 0이 아닌 실수로 구성된 벡터를 의미.

<- 반대 개념: 희소 벡터.

5. 통계 기반 기법의 의미

= '단어의 의미는 주변 단어에 의해 형성된다'는 분포 가설을 기반으로, 대량의 '말뭉치'에서 단어의 의미를 통계적으로 추출하는 기법이다.

= 단어의 의미를 '분산 표현'의 방법론을 빌려서 벡터로 변환하는 기법.

= 어떤 단어의 주변에 어떤 단어가 몇 번 등장했는지를 세어 집계하는 기법.

A. 통계 기반 기법의 예시

① 말뭉치에 속한 단어의 수 = 단어를 표현할 벡터의 차원 수.

② 단어의 전처리 수행. (연산이 가능하도록 배열에 저장하는 등의 작업)

③ 각 단어의 맥락(윈도우 크기 내부의 인접 단어들)의 등장 빈도 집계.

④ 집계한 등장 빈도를 '동시발생 행렬(co-occurrence matrix)'로 정리.

(말뭉치: "you say goodbye and i say hello ." / 윈도우 크기: 1)

그림 2-7 모든 단어 각각의 맥락에 해당하는 단어의 빈도를 세어 표로 정리한다.

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

이 표의 각 행은 해당 단어를 표현한 벡터.

이 표가 행렬의 형태를 띤다는 뜻에서 동시발생 행렬(co-occurrence matrix)라고 함.

B. 단어의 의미란?

① 위의 동시발생 행렬의 각 '행'은
ID에 해당하는 **단어의 의미를 포함한 '고정 길이 밀집 벡터'**이다.
(말뭉치의 크기가 작아서 밀집벡터로 보이지 않을 수 있지만, 예시임을 감안하길 바란다. 실제로 입력한 문장의 크기는 아주 크다.)

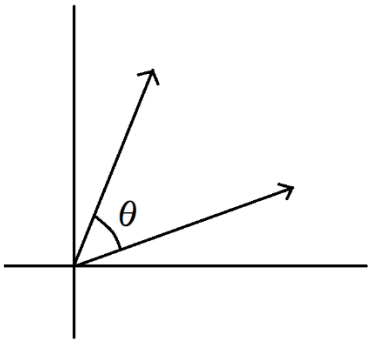
② 위에서 생략되었지만, 각 단어에는 고유한 ID가 부여되어 있다. (전처리 과정의 일부)

6. 단어의 유사도

= 단어 벡터간 유사도

= 단어 벡터간 코사인 유사도(유클리디안 유사도).

= 단어 벡터를 정규화하고 내적한 값.



① 두 벡터가 완전히 동일한 경우 (θ=0) <- 코사인 유사도 = 1

② 두 벡터가 완전히 반대인 경우 ($\theta = \pi$) <- 코사인 유사도 = -1

7. 통계 기반 기법 개선

A. 통계 기반 기법은 고빈도 단어에 대한 연관성 평가가 부정확하다.
(두 단어가 동시에 등장한 횟수를 세었기 때문이다.)

B. 이를 해결하기 위하여 '상호정보량'이란 척도를 사용한다.

8. 점에 대한 상호정보량 (PMI, Pointwize Mutual Information)

A. $PMI(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} = \log_2 \frac{x와\ y의\ 동시발생확률}{x가\ 발생할\ 확률 \cdot y가\ 발생할\ 확률}$

① log는 발산 억제를 위해 씌운 것이다. (스케일 축소)

② $PMI(x,y)$의 값이 높을수록 두 단어의 관련성이 높다.

③ 참고. $P(x) = \frac{C(x)}{N} = \frac{x의\ 동시발생\ 횟수}{말뭉치\ 단어\ 수}$ (C: 동시발생행렬)

B. $PMI(x,y)$를 적용하면, 고빈도 단어에 의한 왜곡을 없앨 수 있다. (많이 등장한 단어의 영향력을, 등장 횟수로 나누었기 때문.)

C. 다만 두 단어의 동시발생 횟수가 0인 경우 $\log_2 0 = -\infty$ 이므로, 실제로 사용할때는 '양의 상호정보량(PPMI)'을 사용한다.

① PPMI = 음수일때는 0으로 취급하는 PMI.

② 수식: $PMI(x,y) = \max(0, PMI(x,y))$ //0 이하를 0으로.

D. 그리하여 우리는 '동시발생행렬'을 '양의 상호정보량 행렬'로 변환하여 사용하게 된다.

```
동시발생 행렬
[[0 1 0 0 0 0 0]
 [1 0 1 0 1 1 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 0]
 [0 1 0 1 0 0 0]
 [0 1 0 0 0 0 1]
 [0 0 0 0 0 1 0]]
------------------------------------------------
PPMI
[[ 0.     1.807  0.     0.     0.     0.     0.   ]
 [ 1.807  0.     0.807  0.     0.807  0.807  0.   ]
 [ 0.     0.807  0.     1.807  0.     0.     0.   ]
 [ 0.     0.     1.807  0.     1.807  0.     0.   ]
 [ 0.     0.807  0.     1,807  0.     0.     0.   ]
 [ 0.     0.807  0.     0.     0.     0.     2.807]
 [ 0.     0.     0.     0.     0.     2.807  0.   ]]
```

9. 차원 감소

= 중요한 정보를 최대한 유지하며, 벡터의 차원 수를 줄이는 방법.

A. 예시의 PPMI행렬은 아주 작지만, 현실에 적용 가능한 수준의 성능이 나오려면, 말뭉치가 아주 커진다.

   -> 우리는 벡터의 정보 일부를 잘 골라내어 버려야 한다.

B. 예시의 PPMI행렬은 대부분의 값이 0이다. 이는 말뭉치의 크기가 커질수록 더 심해진다(고차원의 희소 행렬이 된다.). 그런데 희소 행렬의 연산에는 많은 비용(메모리, 시간)이 든다.

   -> 우리는 이 '고차원 희소 행렬'을 '저차원 밀집 행렬'로 바꾸어야 한다.

C. 대표적인 방법으로 '특이값분해(SVD)'가 있다.

   ① SVD로 행렬을 분해하여 주요한 의미를 담고 있는 요소 추출 가능.

   ② 특이값과 특이 벡터를 통해 데이터의 의미론적 관련성 포착 가능.

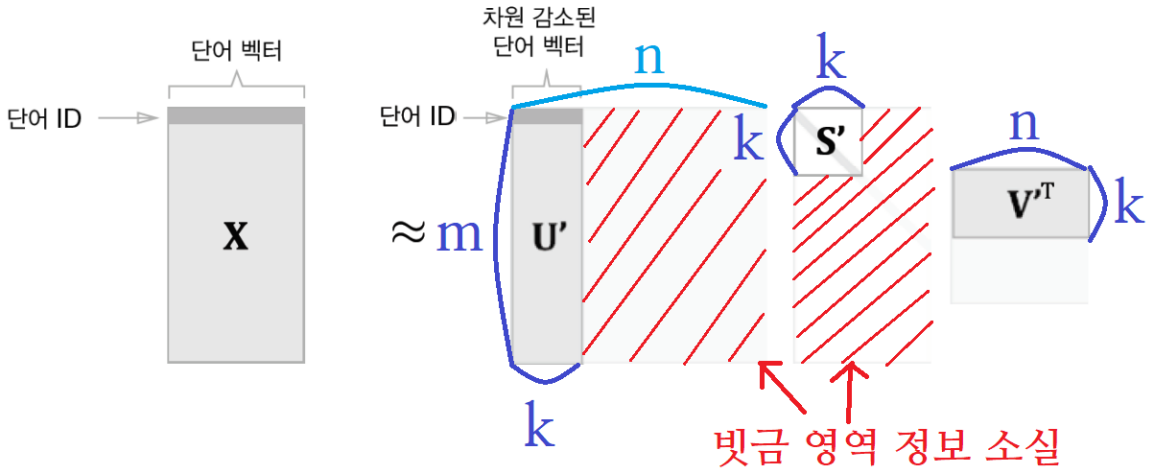
   ③ 단어나 문서간의 유사성을 측정하는데 응용 가능.

10. 특이값분해(SVD)

= 임의의 행렬을 세 행렬의 곱으로 변환하는 것.

A. 수식: $X = USV^T$ -차원축소-> $m \cdot n = (m \cdot k) \cdot (k \cdot k) \cdot (k \cdot n)$

$$X \quad U' \quad S' \quad V'^T$$
$$m \cdot n = (m \cdot k) \cdot (k \cdot k) \cdot (k \cdot n)$$



B. 행렬 V^T는 정방행렬이었다. -> 행렬곱 이후의 형상을 원본인 X와 동일하게 유지할 수 있도록, 하나의 행만 n에서 k로 크기 변경.

C. 행렬 S는 대각행렬이다. -> 행렬곱이 가능하도록 (k, k)로 형상 변환.
(대각선에만 값이 존재하고, 나머지는 모두 0)
대각선 위치에는 '특이값'이 저장되어 있다.
특이값이란?
① 행렬 U의 각 공간에 대한 중요도 정보를 담은 값.
② 행렬의 선형 변환(우리가 하는 것은 정보의 버림)시에 사용되는 기준.

D. 우리는 분해된 행렬 USV^T 각각의 차원을 축소해도, 원본 행렬과 동일한 형상으로 근사할 수 있다. 내용 또한 유사하다. 마치 이미지를 블러 처리하여 압축한 것 처럼, X행렬(U')의 일부 정보를 버릴 수 있게 된다.

E. 즉, 특잇값이 작은 원소를 버린 것이다.

F. 이 결과, 본래의 희소벡터인 'PPMI행렬'을 밀집벡터로 변환할 수 있게 된다. (뭉친 값들을 행렬 공간 전체에 넓게 분포 시킨 것이다. <- 중요한 축을 찾아내어 더 적은 차원으로 다시 표현.)

9주차 정리 끝. (2장 정리 끝.)

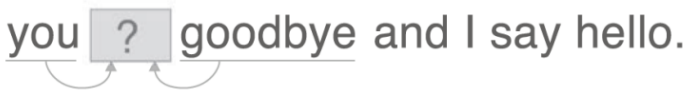
10주차 정리 시작.

1. 서론
   A. 2장은 통계 기반 기법이었다.
   B. 3장은 추론 기반 기법의 일종인 word2vec을 배운다.
   C. 4장에서 word2vec의 단점을 개선해보고,
      5장에서 RNN을 배운다.

2. 통계 기반 기법의 문제점
   A. 대규모 말뭉치를 다룰 때 문제가 발생한다.
      ① 영어의 어휘 수 100만 -〉 100만X100만 크기의 행렬 생성
      ② 그런데 이 큰 행렬을 단 1회의 처리로 얻어내야 함.
         -〉 추론 기반 기법에서는 신경망을 사용한 미니배치 가능!
         -〉 말뭉치의 어휘 수가 많아도, 학습 데이터를 미니배치 단위로 나누어 신경망에 순차적으로 학습 가능.
   B. SVD를 $n \cdot n$행렬에 적용하면 $O(n^3)$시간이 소요된다.
   C. 언어의 어휘가 지니는 의미의 변화를 따라갈 수 없다.
      ① 말뭉치 전체의 통계를 이용해서, 1회의 연산으로 단어의 분산 표현을 얻기 때문에, 추가적인 의미를 나중에 추가할 수 없다. (벡터의 길이부터 변해야 한다. 결과물인 PPMI행렬이 말뭉치에 지나치게 의존적이다.)

3. 추론
   = 맥락(context)이 주어졌을 때, 중앙(target)에 무슨 단어가 들어가는지 추측하는 작업. (신경망이 추측한다.)

   그림 3-2 주변 단어들을 맥락으로 사용해 "?"에 들어갈 단어를 추측한다.

   〈- 윈도우 크기가 1인 경우.

   A. 우리는 신경망에 '맥락'을 투입하고,
      신경망은 '중앙'에 등장할 단어를 예측한다. (확률 출력〈-softmax 사용)
   B. 이전에 배운 신경망과 동일한 구조이다. (신경망의 장점도 계승한다.)
      ① 맥락 -〉 모델 -〉 확률분포 -〉 정답 레이블과 비교하여 오차 계산
         -〉 역전파로 가중치 갱신 -〉 반복….

4. 단어 처리

A. 신경망은 you, say등의 단어를 있는 그대로 처리할 수 없다.

-> 단어를 고정 길이의 벡터로 변환해주어야 한다.

B. word2vec에서는 단어를 원-핫(one-hot)표현으로 변환한다.

(단 하나의 원소만 1이고, 나머지는 모두 0인 벡터.)

C. 단어를 원-핫 벡터로 변환하기

= 총 어휘 수 만큼의 원소를 갖는 벡터에서, 단어 ID만을 1로 표현.

그림 3-4 단어, 단어 ID, 원핫 표현

단어(텍스트)	단어 ID	원핫 표현
you goodbye	0 2	(1, 0, 0, 0, 0, 0, 0) (0, 0, 1, 0, 0, 0, 0)

(예시의 규칙: 총 어휘 수에서, 몇 번째 어휘인지 == ID)

D. 신경망에서의 단어 처리

① 입력층 <- 완전연결로 구성되어 있다.

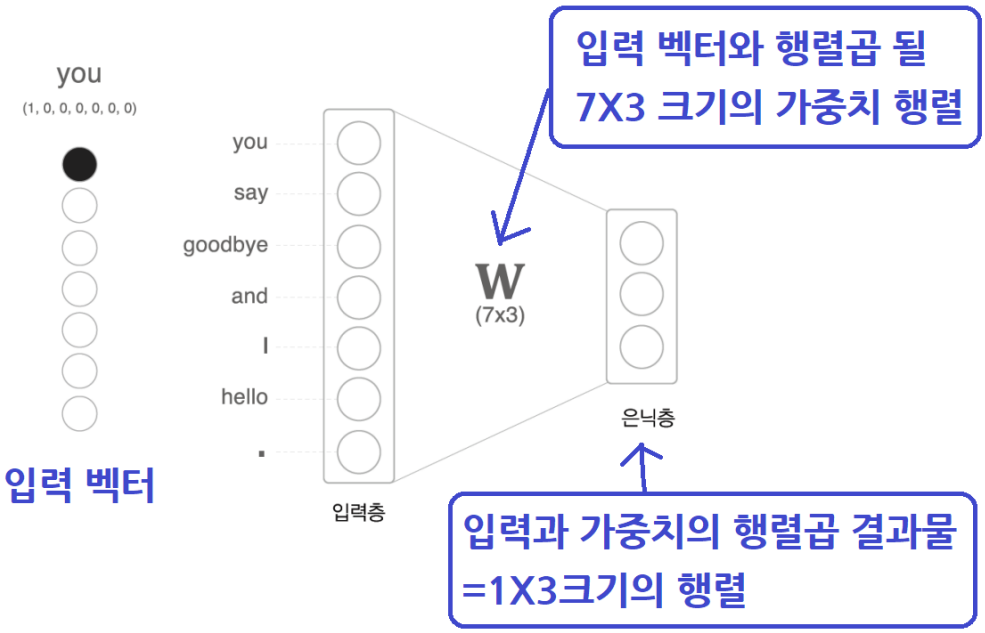
즉, n개의 단어가 n개의 뉴런에 할당된다.

② ID가 0인 단어 [1,0,0,0,0,0,0]이 있다고 하자.

이 단어 벡터는 맥락 중 하나이고, 입력층보다 더 작은 은닉층으로 연결된다.

③ 자세한 것은 조금 뒤에 CBOW모델에서 설명한다.

지금은 입력층이 다음과 같이 단순화될 수 있다고만 알아두라.



④

E. 입력층의 입출력 이해

① 우리는 입력으로 원-핫 벡터를 사용했다. 그렇기에 위의 예시에서 출력되는 1X3크기의 은닉층은 가중치 벡터의 ID번째 행이다.

② c: $[1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$ (*크기*: 1×7)

③ W_{in}: $\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \vdots & \vdots & \vdots \\ a_{71} & a_{72} & a_{73} \end{bmatrix}$ (*크기*: 7×3)

④ $c \cdot W_{in} = h$: $[a_{11} \quad a_{12} \quad a_{13}]$ (*크기*: 1×3)

⑤ 특정 단어의 의미가 내포된 단어의 분산 표현
= 가중치 행렬의 특정 단어 번째 행

5. CBOW 모델

A. 추론 기반 기법이 무엇인지, 신경망에서 단어를 어떻게 처리하는지를 알았으므로, 단어를 벡터로 표현하는 것에 기반을 두는 신경망을 배울 수 있다.

B. CBOW: 맥락으로부터 타겟을 추측하는 신경망.

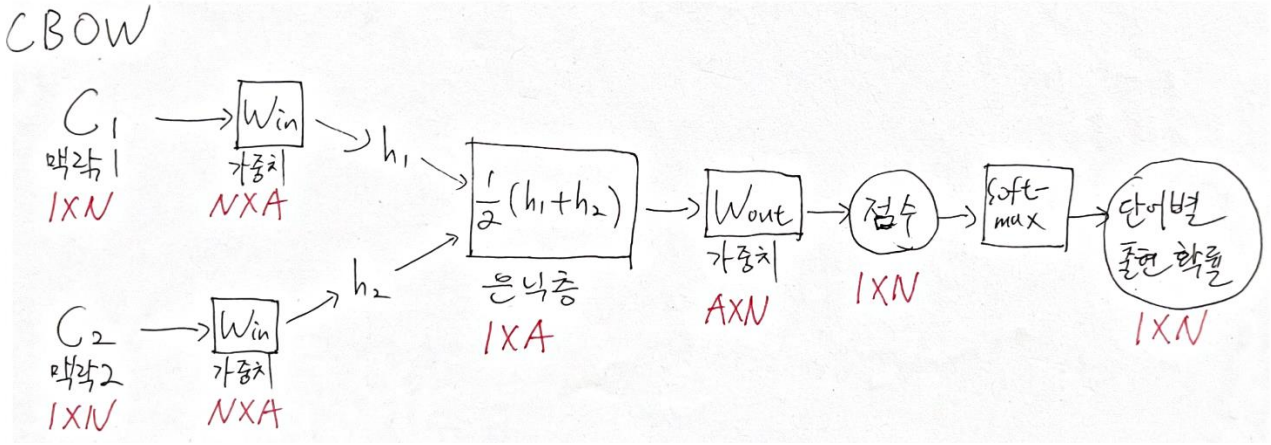
C. CBOW의 특징.

① n개의 입력층(W_{in})을 가진다.
(맥락의 개수만큼 입력을 받는다. 모든 W_{in}은 동일한 값을 가진다.)

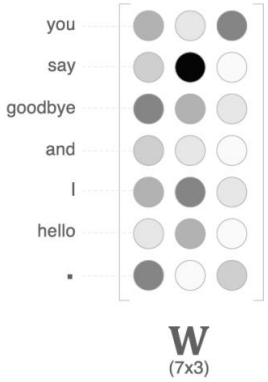
② 하나의 출력층을(W_{out}) 가진다.

③ 입력층의 처리 결과물들은 평균을 내어 은닉층의 값이 된다.
(입력층의 출력 h_1, h_2가 있다고 하자.
-> 입력층의 은닉층 행렬 값 = $\frac{1}{2}(h_1 + h_2)$)

④ 출력층의 각 뉴런들은 각 단어의 점수를 출력한다.
<- 이 점수를 softmax에 투입하여, 각 단어의 출현 확률을 구한다.)



D. 가중치에 대한 이해



① **가중치의 각 행 = 해당 단어의 분산 표현.**

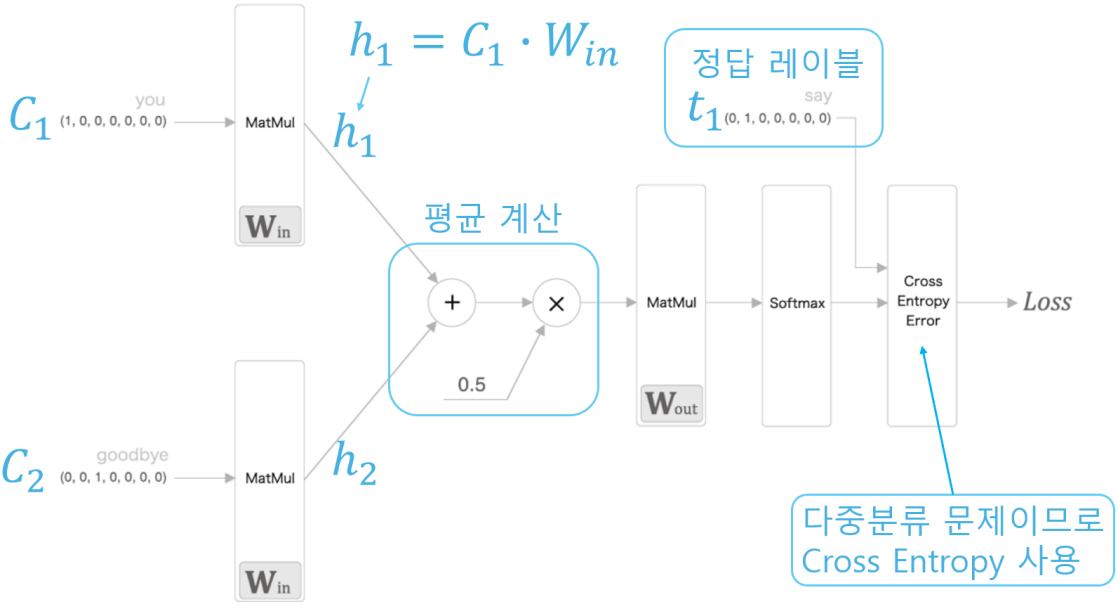
② **'분산 표현'**

= 인간이 이해할 수 없는 방식으로 인코딩 된 '단어의 의미'

〈- 이 분산 표현을 갱신하는 것 = CBOW 모델의 학습.

③ 이 분산 표현은 일반적으로 입력되는 단어 벡터보다 작아야 한다. 단어 예측에 필요한 정보를 간결하게 담기 위함이다. 밀집벡터를 얻고자 은닉층의 크기를 줄이는 것. SVD 분해를 떠올려라. (지금은 크기 7에서 3으로 축소.)

E. CBOW 모델의 계산그래프와 학습



① 계산과정에서의 행렬 형상 변화는 앞서 '항목C'에서 다루었다.

② CBOW로 이진 분류 문제를 다룬다면 출력함수로 Softmax 대신 Sigmoid를 사용해도 된다. 하지만 CBOW에서는 다중 분류 문제를 주로 다루므로, Softmax를 사용하는게 일반적이다.

F. word2vec의 두 가중치에 대하여

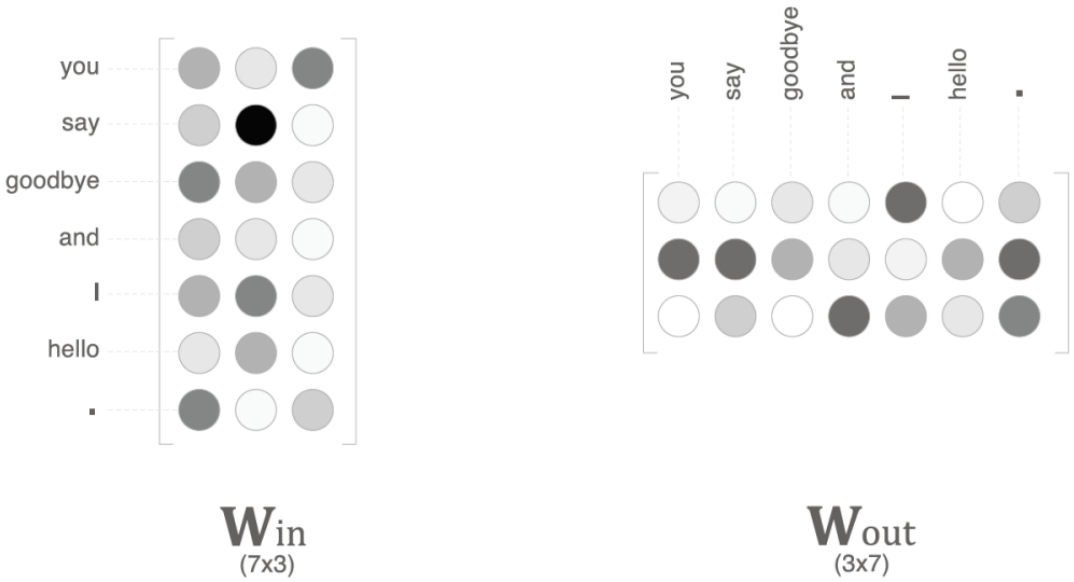
W_{in}과 W_{out} 모두 단어의 의미를 품고 있는 행렬이다(단어의 분산 표현들의 집합체). 일반적으로 입력 측 가중치(W_{in})만을 사용하고, 출력측을 버린다.

① 입력 측 가중치는, 입력 벡터가 원-핫 벡터로 표현된 덕에, 가중치 행렬의 특정 행이 곧 분산 표현이었다.

② 그러나 출력 측 가중치는, 입력 측의 출력인 은닉층의 값($\frac{1}{2}(h_1 + h_2)$)을 W_{out}과 행렬 곱 한다.

-〉 W_{out}의 특정 열 = 단어의 분산 표현.

그림 3-15 각 단어의 분산 표현은 입력 측과 출력 측 모두의 가중치에서 확인할 수 있다.



③

④ 이때 W_{out}은 신경망의 학습 과정에서만 사용되고 버려지는 것이다. W_{out}은 신경망 전체의 출력인 '예측 값'을 얻어내기 위해 존재하며, W_{out}에서 단어의 분산 표현, 즉 '의미'를 추출할 이유는 없다.

⑤ 그러나 꼭 W_{out}을 버려야만 하는 것은 아니다. 필요한 경우 양측의 가중치를 모두 써도 좋고, 출력 측만 사용해도 좋다. 동등하게 단어의 의미가 잘 인코딩 되어 누적된다.

6. 학습 데이터 준비

A. 우리는 이미 어떤 데이터를 준비해야 하는 지 다 배웠다.

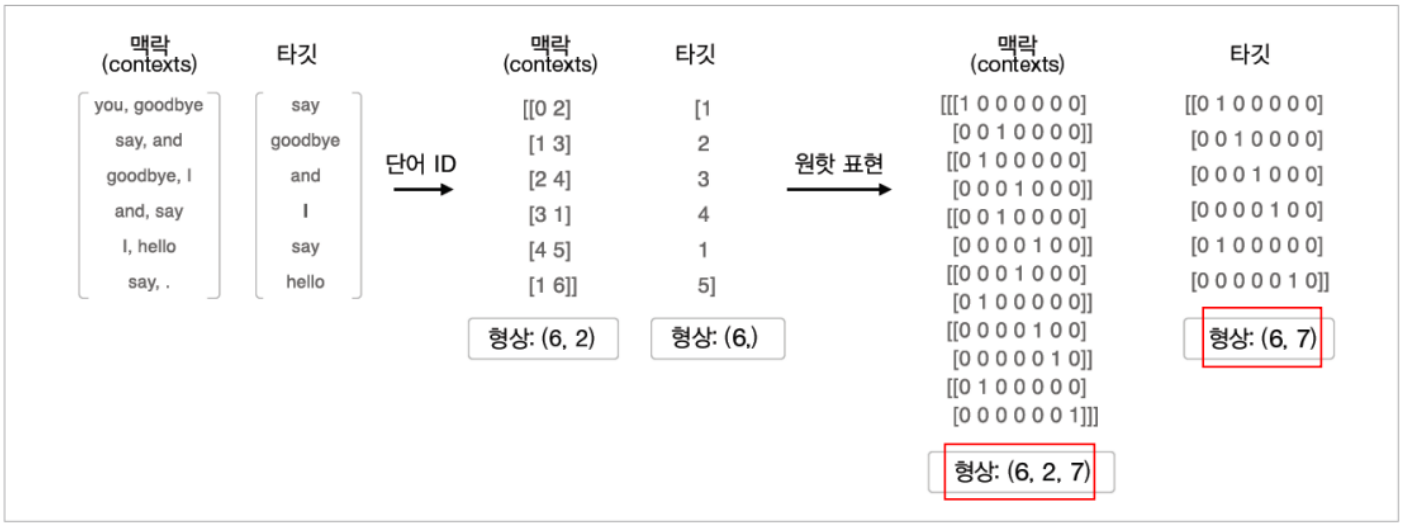
그림 3-16 말뭉치에서 맥락과 타깃을 만드는 예

B.

C. 위의 그림만 보아도 전부 이해 될 것이라 믿는다.

D. 우리는 모든 단어에 고유한 ID를 부여할 것이고,
모든 맥락과 타깃은 'ID번째 원소만 1인 원-핫 벡터'로 표현될 것이다.

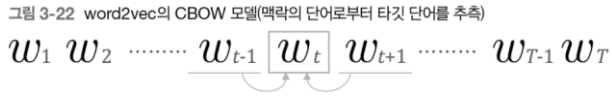
그림 3-18 '맥락'과 '타깃'을 원핫 표현으로 변환하는 예



E. 이 이상 설명할 것이 없다.

7. 확률 관점에서 본 CBOW 모델

그림 3-22 word2vec의 CBOW 모델(맥락의 단어로부터 타깃 단어를 추측)



A.

B. 맥락이 주어졌을 때 타겟이 w_t일 확률

= $P(w_t|w_{t-1}, w_{t+1})$

= 신경망이 추론한 타겟 단어의 정답 확률.

① 이때 우리는 Softmax + CrossEntropy를 사용한다.

② 즉, 손실함수는 $L = -\log P(w_t|w_{t-1}, w_{t+1})$ 이렇게 생겼다.
(정답 레이블이 원-핫 벡터로 되어 있으므로, 예측 대상인 w_t의 정답 확률에 음수와 로그를 씌운 것이 곧 손실값이다.)

③ 그런데 우리는 말뭉치의 모든 단어에 대하여 예측을 해야 하므로, 다음과 같은 손실함수가, 말뭉치 전체에 대한 손실함수가 된다.

④ $$L = -\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_{t-1}, w_{t+1})$$ <- 이것이 CBOW의 수학적 표기.

8. skip-gram <- word2vec에 속하는, CBOW가 아닌 모델.

A. CBOW는 맥락으로부터 타겟 단어를 추론한다.

B. skip-gram은 타겟으로부터 맥락을 추측한다.

그림 3-23 CBOW 모델과 skip-gram 모델이 다루는 문제

C.

D. CBOW를 뒤집으면 skip-gram이 된다.
(하나의 입력층, 맥락의 개수만큼의 출력층 존재.)

E. 그러므로 skip-gram 모델을 수학적으로 표기하면 다음과 같다.

① 맥락 예측: $P(w_{t-1}, w_{t+1}|w_t)$

② 예측값 분해: $P(w_{t-1}, w_{t+1}|w_t) = P(w_{t-1}|w_t)P(w_{t+1}|w_t)$
(맥락의 단어들 사이에 관련성이 없다고 가정한다.)

③ 맥락 둘에 대한 손실함수: $L = -(\log P(w_{t-1}|w_t) + \log P(w_{t+1}|w_t))$

④ 말뭉치 전체에 대한 손실함수:

- $L = -\frac{1}{T}\sum_{t=1}^{T}(\log P(w_{t-1} \mid w_t) + \log P(w_t \mid w_{t+1}))$

9. 통계 기반 기법과 추론 기반 기법의 비교

A. 새로운 단어를 추가해야 하는 경우. -〉 분산 표현의 갱신을 해야 함.

① 통계 - 단어를 추가한 말뭉치로 처음부터 다시 계산.

② 추론 - 기존에 학습된 가중치 매개변수를 초기값으로 하여, 재학습.

- 기존 학습 경험을 유지한 채로, 단어의 분산 표현 갱신 가능.

B. 두 기법에서 얻는 단어의 분산 표현이 갖는 성격과 정밀도.

① 통계 - 단어의 유사성이 주로 인코딩 된다.

② 추론 - 단어의 유사성은 물론, 단어 사이의 패턴도 인코딩 된다.

- “king - man + woman = queen” 같은 유추 문제 해결 가능.

③ 하지만 정량적 평가 결과, 두 기법의 성능은 동등했다.

C. 두 기법의 성능이 동등하다면?

-〉 갱신 가능한 추론기법을 쓰는 것이 좋다.

10주차 강의 정리 끝.

11주차 강의 정리 시작.

1. 서론
   A. 교재의 4장을 공부해 보자.
   B. 4장은 word2vec의 속도를 개선하는 방법에 대하여 배운다.
   C. 주요 개념
      ① Embedding 계층.
      ② 네거티브 샘플링.

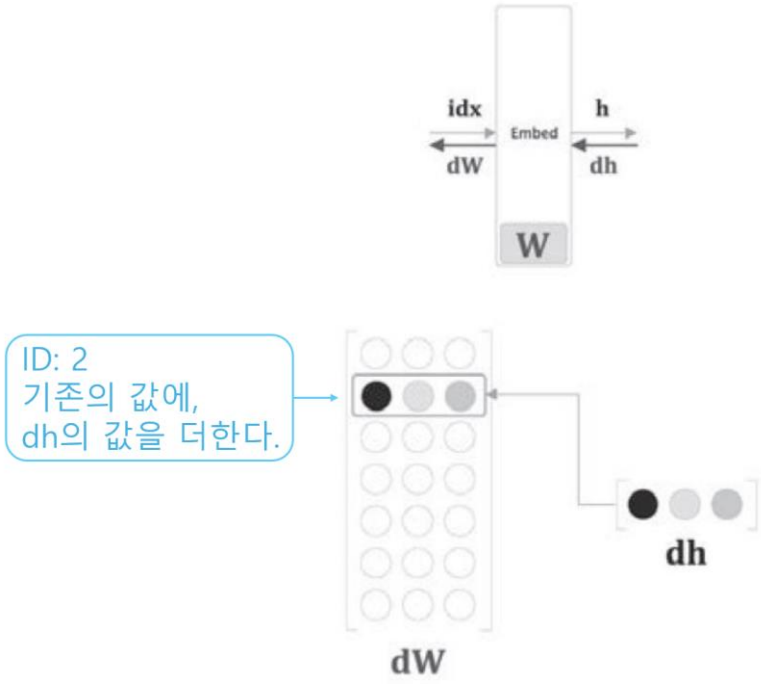
2. CBOW의 문제
   A. 말뭉치의 크기가 매우 큰 경우, 심한 병목이 발생한다.
   B. 병목의 주요 발생처
      ① 입력 벡터와 W_{in}의 행렬 곱. <- Embedding 계층 도입으로 해결.
      ② 은닉층과 W_{out}의 행렬 곱. <- 네거티브 샘플링으로 해결.
      ③ Softmax 계층의 계산. <- 네거티브 샘플링으로 해결.

3. Embedding 계층
   = 단어 ID에 해당하는 행을 추출하여 출력하는 계층.
   A. 교재 3장에서 우리는 행렬W_{in}의 각 행이, 개별 단어의 분산 표현임을 배웠다. 그러나 3장에서 CBOW를 구현 할 때는 '행렬 곱'을 한다고 적어두고 일단 넘어갔다.
   B. 행렬 곱이 필요 없는 단계인데, 행렬 곱을 하라니? 말이 안 되지 않는가?
      -> 그래서 우리는 **특정 ID번째 행을 W_{in}에서 뽑아내어 출력하는 계층**을 만들기로 한 것이다.

4. Embedding 계층의 역전파

A. 순전파와 동일하게 동작한다.

앞 계층에서 넘어온 역전파 값을, 가중치 행렬의 ID번째 행에 더한다.



B.

C. **덮어 쓰면 안된다!**

말뭉치에서 동일 단어가 여러 번 등장할 것이기에, 기존에 학습된 값을 지켜야 한다.

즉, 꼭 기존 값에 dh를 더해야 한다는 말.

5. word2vec의 문제 복습

A. Softmax 대신 네거티브 샘플링을 사용하면,

어휘가 아무리 많아져도 계산량을 일정하게 억제할 수 있다.

(다음 두 문제의 해결책이 네거티브 샘플링이다.)

B. 문제 ②는 W_{out}의 연산 병목이었고, 문제 ③은 Softmax의 연산량이었다.

① W_{out} <- 행렬이 너무 크기 때문 + 원-핫 벡터가 연산에 쓰이지 않아서 Embedding이 안됨.

② Softmax <- 어휘 숫자만큼의 확률 값을 연산하기 때문.

6. ‘다중 분류 문제’를 ‘이진 분류 문제’로 근사할 수 있는가?

A. 이 질문은 ‘네거티브 샘플링’을 가능케 하는 핵심을 포함한다.

B. Softmax가 막대한 연산력을 요구하는 이유는, 주어진 모든 단어에 대하여 각각이 정답일 확률을 계산하기 때문이다. 단지 점수를 변환하는 것 뿐인데, 엄청난 자원을 소모하는 것이다.

-> 그렇다면 우리는 신경망의 입력에 대하여, 모든 분류(클래스)가 Yes인지 No인지를 물어보면 되는 것이다.

① 전: $O(n^2)$ <- 입력을 할 때 마다, 단어 각각의 확률을 모두 계산.

② 후: $O(n)$ <- 모든 분류(클래스)에 Yes/No를 1회씩 계산.

C. 질문에 대한 답: 정답이 아닌 모든 클래스에 대하여 0에 가까운 값이 나오도록 할 수 있다면, 이진분류로의 근사가 가능하다!

7. W_{out}의 연산 개선

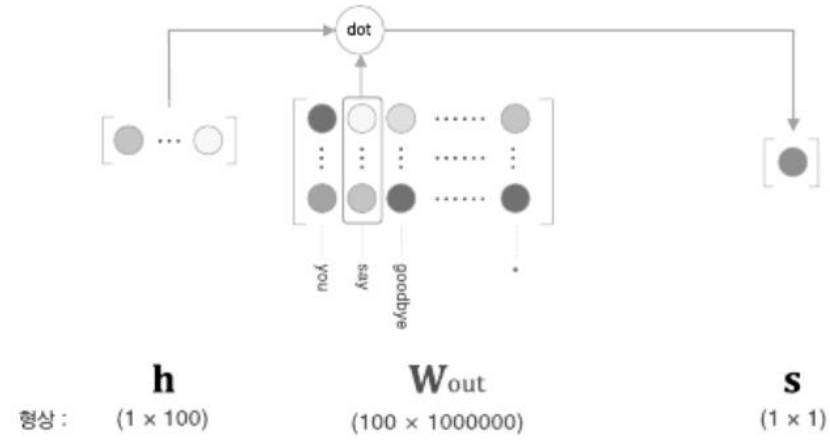
A. 신경망이 이진 분류 문제를 다루게 되었기에,
W_{out}의 연산 방식도 바꿀 수 있다.

B. 이제는 모든 단어에 대한 점수를 출력하지 않아도 된다.

C. 입력된 단어의 의미를 인코딩 한 벡터가 아래의 h이므로,
h와 W_{out}의 모든 열의 유사도를 계산하여, 입력 단어와 가장 유사한 ID가 무엇인지를 구할 수 있다. <- 이렇게 구해진 유사도는 추후 Sigmoid를 통과하여 Yes/No의 답으로 가공된다.

[그림 4-8] “say”에 해당하는 열벡터와 은닉층 뉴런의 내적을 계산한다.

D.



E. [W_{in}의 ID번째 행인 h]와 [W_{out}의 모든 열 벡터]를 순차 비교하여,
[지금 비교중인 ID번째가 타겟인가?]에 대하여 Yes일지 No일지에 대한 점수를 계산하여 출력한다. (단어 개수만큼의 노드 존재)

F. 계산된 점수는 Sigmoid를 적용하여 확률로 변환한다.

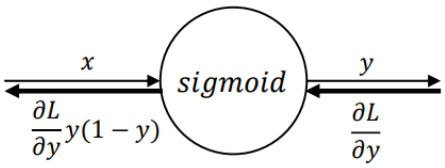
① Sigmoid + CrossEntropy <- 이진 분류 문제에 많이 씀.

② Softmax + CrossEntropy <- 다중 분류 문제에 많이 씀.

G. 역전파되는 값에 대하여.

① Sigmoid 함수: $y = \frac{1}{1+\exp(-x)}$

② 우리가 과거에 배운 대로 CrossEntropy Loss를 미분하고, Sigmoid에 역전파 시키면 다음 그림처럼 표현된다.

③ 

(y: 확률, 시그모이드 함수의 출력값.)

④ Sigmoid + CrossEntropy Loss: $L = -(t\log y + 1 - t\log 1 - y)$

⑤ t는 정답 레이블이므로 0또는 1이다.

⑥ 정답 -> $L = -\log y$

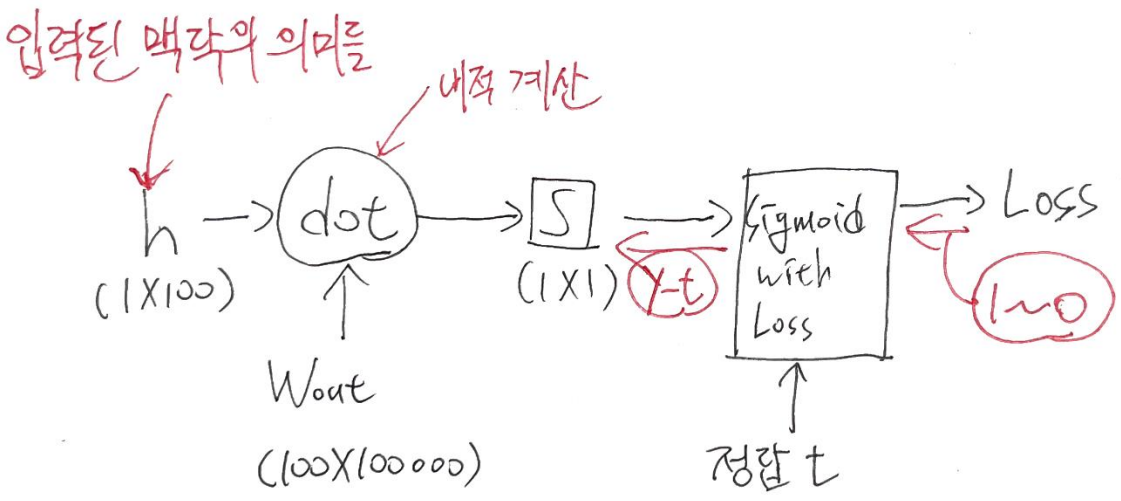
⑦ 오답 -> $L = -\log(1-y)$

⑧ 위의 두 값이, 정답일때와 오답일 때의 손실 값인 것이다.

H. 앞서 계산한 손실 값을 역전파 시키면, (교재 1권에서 배웠던 것 처럼)
역전파되는 값은 "$y - t$"가 된다. (Sigmoid 출력값 - 정답레이블)

① 즉, y가 정답에 가까울수록 오차는 감소한다.

② 오차가 작으면 작게 학습하고, 오차가 크면 크게 학습한다.



8. 네거티브 샘플링

= 부정적인 예(오답)를 학습시키는 것.

A. 모든 오답에 대하여 0이 나오도록 학습시킬 수 없다. (비용, 시간 등….)

B. 그러므로 몇 개의 오답 샘플을 골라서 신경망에 학습시킨다는 것!

C. 앞서 아주 길게 설명한 W_{out} 의 'Embedding dot' 계층과, Softmax를 Sigmoid로 대체하는 작업을 적용하고 나서, 네거티브 샘플링을 수행하면 -> '이진 분류 문제로의 근사'가 성공한다는 말이다.

D. 말뭉치에서 자주 등장하는 단어를 많이 추출하고, 드물게 등장하는 단어를 적게 추출하여 (신경망에 오답으로 투입하여)학습을 수행하면 효과가 좋다. (말뭉치에 등장하는 단어들의 확률분포를 계산하여 샘플링 한다는 말이다.)

9. 분산 표현이 중요한 이유

A. 전이 학습이 가능하다. (기존에 학습된 가중치 데이터를 가지고, 다른 곳에 적용할 수 있다는 말. 추가로 원하는 것을 학습시키기 가능.)

B. 단어를 고정 길이 벡터로 변환해준다는 것에는 장단점이 있다.

① 장점: 머신러닝 기법 적용 가능.

② 단점: 말뭉치가 너무 큰 경우, 벡터 길이가 무한정 늘어날 수 있음.

10. 참고. (공부해서 알아보라고 교수님이 언급)

A. W_{in} 은 임베딩이 가능한데, W_{out}은 임베딩이 안되고 Embedding-dot 계층을 쓰는 이유는?

① 입력 단어가 원-핫 인코딩 되어 있지 않아서 W_{out}은 임베딩이 어려운 것 아닌가?

② 그렇다면 코드를 잘 작성해서 입력된 맥락의 ID를 보전할 수 있는데, 임베딩을 하지 않는 이유는? -> 내 생각엔, W_{out}에서 얻어야 하는 값이 '단어의 의미가 포함된 특정 열 벡터'가 아니기 때문일 것으로 보인다. 우리는 W_{out}을 통해서 타겟과 맥락의 유사도를 얻고 싶지, 임베딩을 해서 뭔가를 해야 하는 것이 아니다.

③ 그리고 교수님 언급이 이상한 점은… Embedding-dot 계층도, 의미적으로 Embedding을 하고 있는 것 아닌가? 음… 이상한 언급이다.

B. 네거티브 샘플링 하는 이유

① 이것은 쉽다. 우리는 Softmax를 Sigmoid로 대체하는 것으로 다중 분류 문제를 이진 분류 문제로 근사하려고 하기 때문에, 필연적으로 오답에 대하여 높은 정확도로 0을 출력할 수 있어야 하기 때문이다.

② 그러므로 우리는 오답을 직접 학습시켜서, 오답에 대한 낮은 점수 출력을 유도하는 것이다.

C. h의 의미는?

① 입력된 맥락 단어들이 지니고 있는 의미들을 인코딩 한 결과물이다.

D. 가중치의 의미는?

① 말뭉치의 모든 단어들에 대하여, 각 단어가 지니고 있는 의미들을 벡터 형태로 인코딩 해 놓은 것.

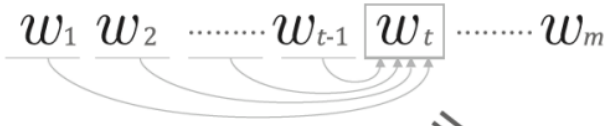
11주차 강의 정리 끝.

12주차 강의 정리 시작.

1. 서론
   A. 교재의 큰 덩어리인 word2vec이 끝났다.
   B. 지금까지 배운 모든 신경망은 피드포워드 유형에 속한다.
      ① 흐름이 단방향인 신경망을 말한다.
      ② 시계열 데이터를 잘 다루지 못한다는 단점이 있다.
   C. 이제부터는 RNN에 대하여 학습을 시작한다.

2. 언어 모델
   A. 단어 나열에 확률을 부여해야, 언어 모델을 만들 수 있다.
   B. 단어 순서의 자연스러움을 확률로 평가 가능해야 한다는 말이다.
   C. 그 말은, 입력된 모든 단어가 시간을 기준으로 선-후 관계가 있다는 것이고, 우리가 구해야 하는 타겟의 확률이 다음과 같이 표현된다는 말이다.

그림 5-3 언어 모델이 다루는 사후 확률: *t*번째 단어를 타깃으로 하여 *t*번째보다 왼쪽 단어 모두를 맥락(조건)으로 고려한다.



$$= P(w_t | w_1, w_2, \ldots, w_{t-1})$$

   D. 즉, 타겟 이전의 모든 단어를 맥락으로 고려해야 한다는 말!

3. CBOW 모델로 언어 모델을 만들 수 있는가?
   A. 전혀 불가능한 것은 아니다. 맥락의 크기를 충분히 키운다면 가능하다.
   B. 하지만 결국 맥락의 크기가 고정된다는 한계가 있다. 맥락 크기를 초과하는 과거의 입력은 망각되는 것이다.
   C. 또한 맥락 안의 단어 순서가 무시된다는 것 또한 큰 한계이다.
      (you, say, [target] == say, you, [target])
      ① 이 문제를 극복하기 위하여 맥락의 단어 벡터를 은닉층에서 별도로 연결해줄 수 있으나, 맥락의 크기에 비례하여 가중치 매개변수가 늘어난다는 문제가 있다.
   D. 결론: 맥락이 아무리 길더라도 기억할 수 있는 순환신경망(RNN)이 필요!

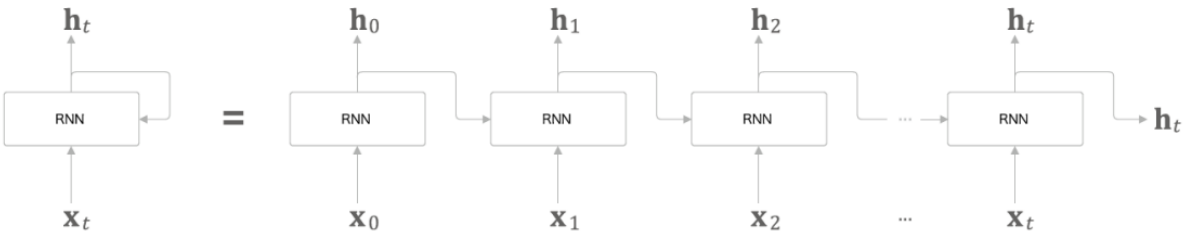
4. RNN

= 순환신경망. (Recurrent neural Network)

A. 순환을 위해서는 닫힌 경로가 필요하다.

B. 그리고 그 닫힌 경로는, 시간을 기준으로 펼칠 수 있다.

그림 5-8 RNN 계층의 순환 구조 펼치기

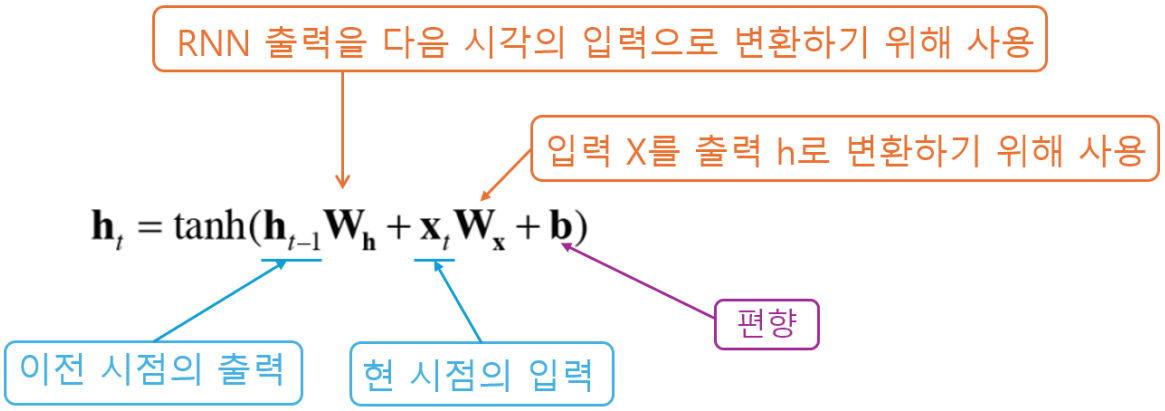


C.

D. 위의 그림은 하나의 계층이다.
(x_0부터 순차적으로 RNN 계층에 투입되는 것이다.)

E. 각 시각의 RNN 계층은 직전 시각의 RNN의 출력과, 새로운 입력을 함께 받는다.

F. 특정 시각의 출력은 다음 수식으로 표현된다.



① 하이퍼볼릭 탄젠트는 활성화 함수처럼 기능한다.

② W는 둘 다 가중치가 맞다.

③ h_{t-1}, x_t: 행 벡터이다.

④ h_t : 은닉 상태 벡터(hidden state vector).

- 특정 시점의 RNN의 출력 값 = 다음 시각의 RNN의 입력 값.
- h_t는 과거의 모든 RNN의 출력을 압축하여 저장하고 있다.
- 그러므로 RNN 계층을 '메모리(기억력, 상태)가 있는 계층'이라 함.
- 지금은 언어를 다루고 있으므로, h_t는 '시퀀스 데이터의 문맥 정보가 압축된 형태'라고 할 수 있을 것이다.

5. BPTT

= 시간 방향으로 펼친 오차역전파법. (Backpropagation Through Time)

A. 이전의 그림에서 RNN 계층을 펼쳐 둔 것은, 오차역전파법의 적용을 위함이다. 우리는 피드포워드 신경망에 했던 것과 동일하게 오차역전파를 할 수 있다.

B. 그러나 역전파를 수행하기 전에 해결해야 할 문제가 남아있다.

① 시간 크기가 너무 클 경우, 컴퓨팅 자원 소모량이 급증한다.

② 시간 크기가 증가할수록 기울기 소실 문제가 심해진다.

6. Truncated BPTT

= 시간축 방향으로 너무 길어진 신경망을, 작은 신경망으로 쪼개어 역전파를 수행하는 것.

A. 너무 긴 계층 -> 계산량과 메모리 사용량 문제 + 기울기 소실 문제 발생.

B. 다만 순전파시의 연결은 유지해야 한다. (쪼개지지 않은 것 처럼 순서대로 처리되어야 함.)

C. 역전파시의 연결만 실제로 쪼개어, 잘라낸 신경망 단위로 학습을 수행한다.

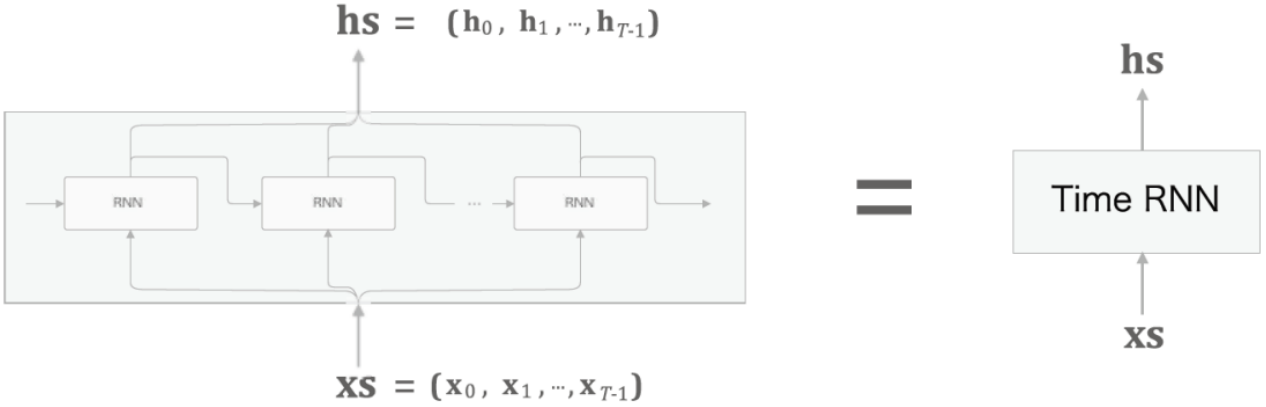
① 이때 이 쪼개진 부분을 '블록'이라 한다.

② 각각의 블록은 미래의 블록과 상관 없이, 독립적으로 역전파를 통한 학습을 수행할 수 있다.

③ 미니배치도 가능하다.

7. Time RNN

= 시계열 데이터를 입력받아서, 한번에 작업을 처리하고 출력하는 RNN 계층.



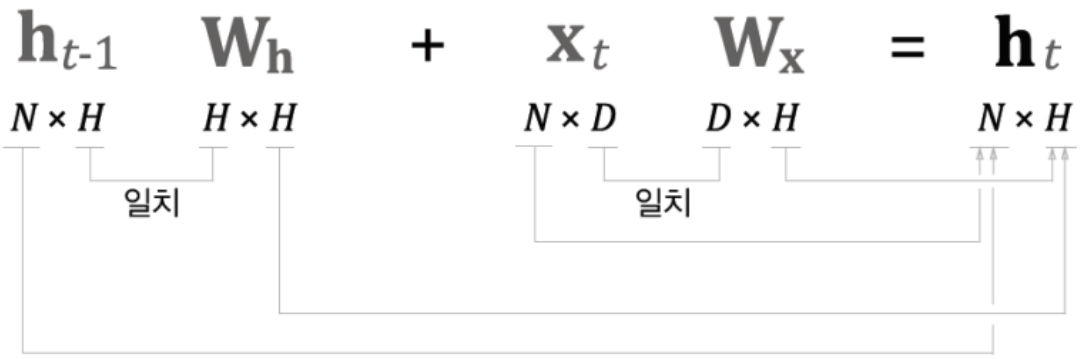
A. 내부적으로 RNN을 순환시키면서, 복수의 벡터 입력을 처리한다.

8. RNN의 미니배치 학습 이해

A. RNN의 순전파: $h_t = \tanh(h_{t-1}W_h + x_tW_x + b)$

B. 여기서 미니배치 크기를 N이라 하면, 다음과 같은 형상으로 구성 가능.

그림 5-18 형상 확인: 행렬 곱에서는 대응하는 차원의 원소 수를 일치시킨다(편향은 생략).



C.

D. x_t, h_t의 형상에 대한 이해.

① 각 행은 하나의 시퀀스 데이터이다.

- N개의 데이터를 1배치로 설정했음을 기억하라.
- 즉, 하나의 행은 특정 샘플의 시각별 입력 값들이 순서대로 저장된 것이다.

② 각 열은 시퀀스의 시간 단계들이다.

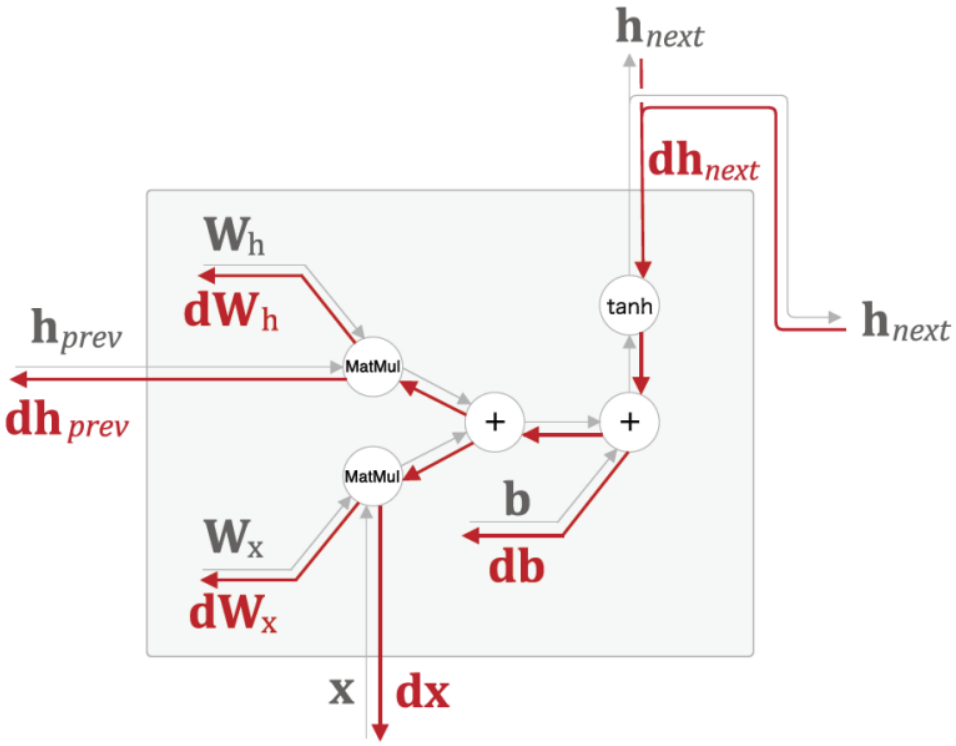
- 앞서 하나의 행이, 하나의 샘플임을 이야기 한 것을 기억하라.
- 열 번호를 시각으로 여기어도 좋다.

③ 구체적 예시 (N=3, H=5 / 미니배치3, 시각 5단계)

- t=1 t=2 t=3 t=4 t=5
- x1 x2 x3 x4 x5
- y1 y2 y3 y4 y5
- z1 z2 z3 z4 z5

9. RNN계층의 계산 그래프

그림 5-20 RNN 계층의 계산 그래프(역전파 포함)



A. 이전 시각의 출력 h와, 현 시각의 입력 x에
가중치 행렬을 곱하여 하이퍼볼릭탄젠트로 집어넣는 것을 보라.

10. Time RNN 계층 구현.

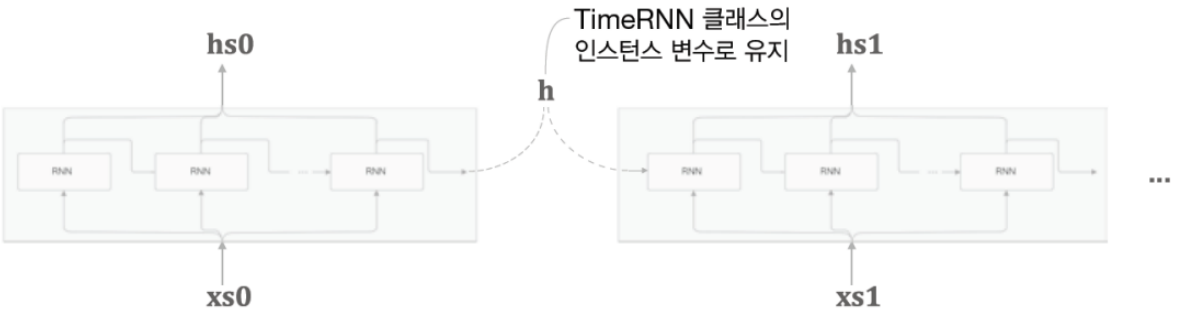
(Time RNN이 무엇인지는 항목7번 에서 설명하였다.
내부적으로 RNN을 순환시키면서, 복수의 입력을 한번에 처리하는 RNN이다.
T개의 RNN계층으로 구성된 하나의 계층으로 보아도 좋다.)

A. 구현시에 알아야 할 것은, 다른 블록으로 어떻게 정보를 전달하는지이다.

B. [마지막 RNN의 출력]을 [다음 블록의 첫 RNN]에 투입하면 된다.

그림 5-22 Time RNN 계층은 은닉 상태를 인스턴스 변수 h로 보관한다. 그러면 은닉 상태를 다음 블록에 인계할 수 있다.

C.



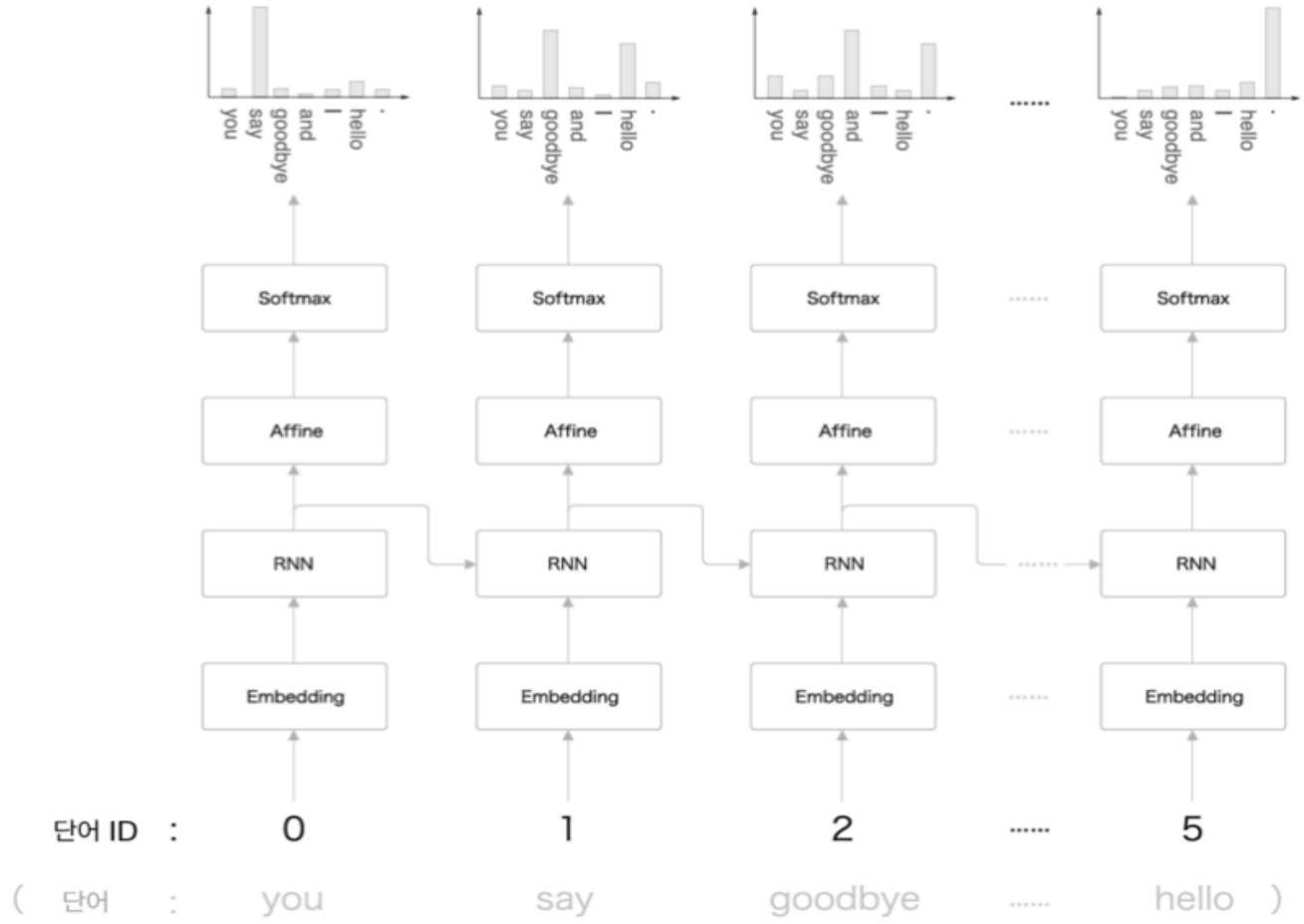
11. RNNLM

= RNN Language Model

= RNN을 사용한 언어 모델

A. 이제 드디어 언어 모델을 만들어 본다.

그림 5-26 샘플 말뭉치로 "you say goodbye and I say hello ."를 처리하는 RNNLM의 예



B. 이것이 전부다. 그림만 보아도 이해가 되지 않는가? 그래도 먼 미래에 망각하여 이해하기 어려울 수 있으니 설명을 남긴다.

① Embedding 〈- 입력된 [단어 ID]를, [단어 벡터(분산 표현)]로 변환.

② RNN 〈- [단어 벡터]를 입력 받고, [다음 출력]을 예측하여, [Affine계층]과 [다음 시각 RNN]에게 예측값을 넘긴다.

③ Affine 〈- Softmax로 확률을 구하기 위해 존재하는 완전연결 계층.

④ Softmax 〈- Affine계층으로부터 넘어온 [단어별 점수]를, [단어별 정답 확률]로 변환.

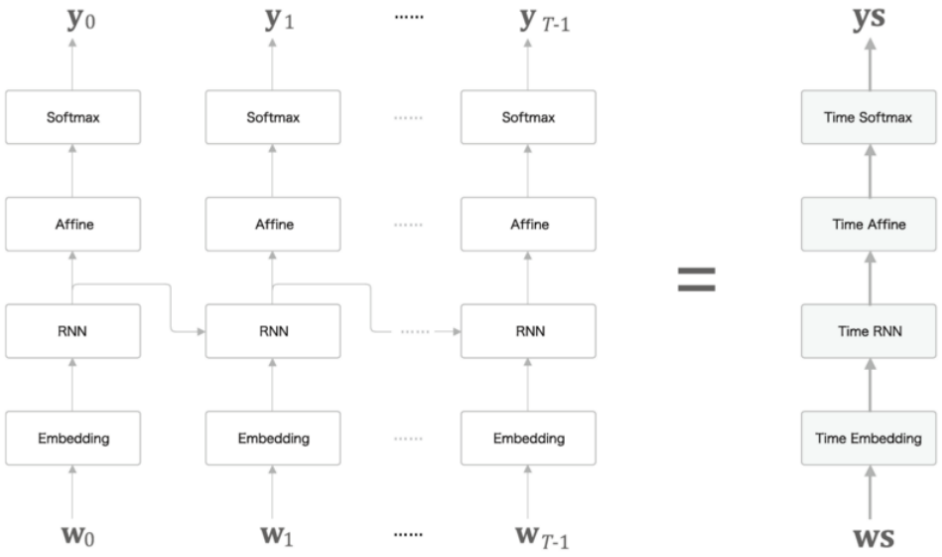
C. 주의! 첫 번째 RNN계층의 입력: 0또는 가비지값 입력.

D. 역전파시 정답 레이블과 비교하여 $y - t$를 역전파. (예측 확률 - 정답)

12. Time RNNLM

A. RNNLM을 T개 묶어서, 시계열 데이터를 한 번에 처리하도록 만들 수 있다. (T개 분의 시계열 데이터를 한번에 처리.)

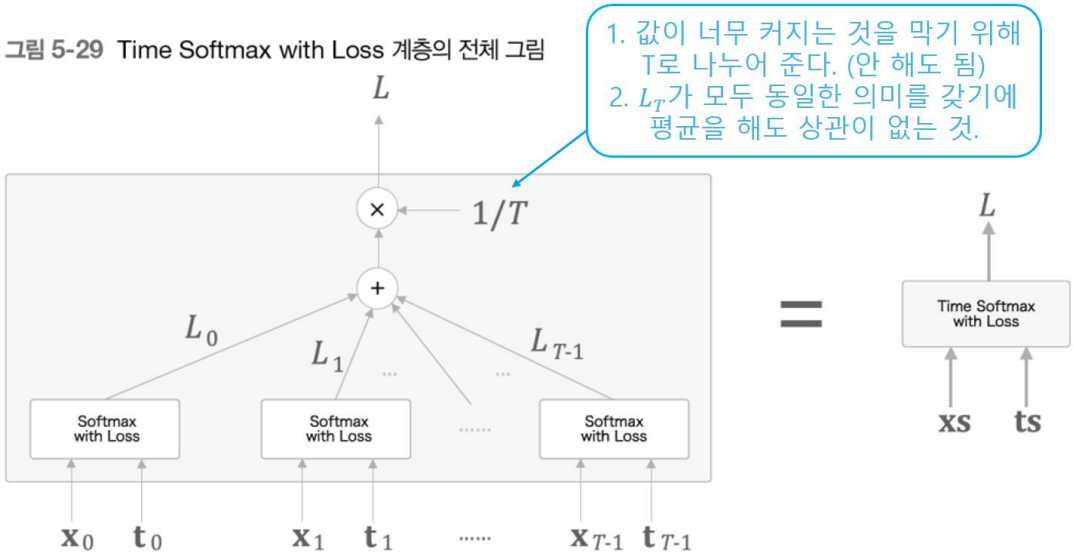
그림 5-27 시계열 데이터를 한꺼번에 처리하는 계층을 Time XX 계층으로 구현



B.

C. 이것이 펼쳐진 형태임을 기억하라. 여럿 처럼 보이지만, 실제로는 하나의 RNNLM이 있고, 그것이 반복적으로 순환하며 실행되고 있는 것이다. 그러므로 Time 계층을 만들어도 연산 시간이 빨라지는 것이 아니다. (모든 RNN 계층은, 이전 RNN계층의 연산을 기다려야 하므로.)

그림 5-29 Time Softmax with Loss 계층의 전체 그림



D.

$$L = \frac{1}{T}(L_0 + L_1 + \cdots + L_{T-1})$$

E. 그림 내부의 2번을 보라. '동일한 의미'라는 말은, 실제로 같은 신경망이 순환하며 재실행되기 때문이다. 만일 여러 신경망을 조합하여 Time RNNLM을 구현했다면 그림5-29처럼 평균을 낼 수 없다. (그 경우, 값을 합산해야 할 것.)

13. 퍼플렉서티 (Perplexity)

= 확률의 역수.

= 분기수, 혹은 혼란도.

= 주어진 선택지의 폭을 수치화 한 것.

= 다음에 취할 수 있는 선택사항의 수.

= 다음에 출현 가능한 단어의 후보 수.

A. 언어 모델의 예측 성능 평가 척도로 잘 쓰인다.

① 확률 0.2 예측 -> 퍼플렉서티=5

② 확률 0.8 예측 -> 퍼플렉서티=1.25

③ 확률 1 예측 -> 퍼플렉서티=1

④ 이제 '분기수'라는 표현이 이해가 가는가? 말 그대로 선택지의 폭을 수치화 한 것이다. 실제 선택지 개수와는 다르지만, 상식적인 선에서 느끼는 '가능한 선택지가 몇 개 있는지?' 와 닮았다.

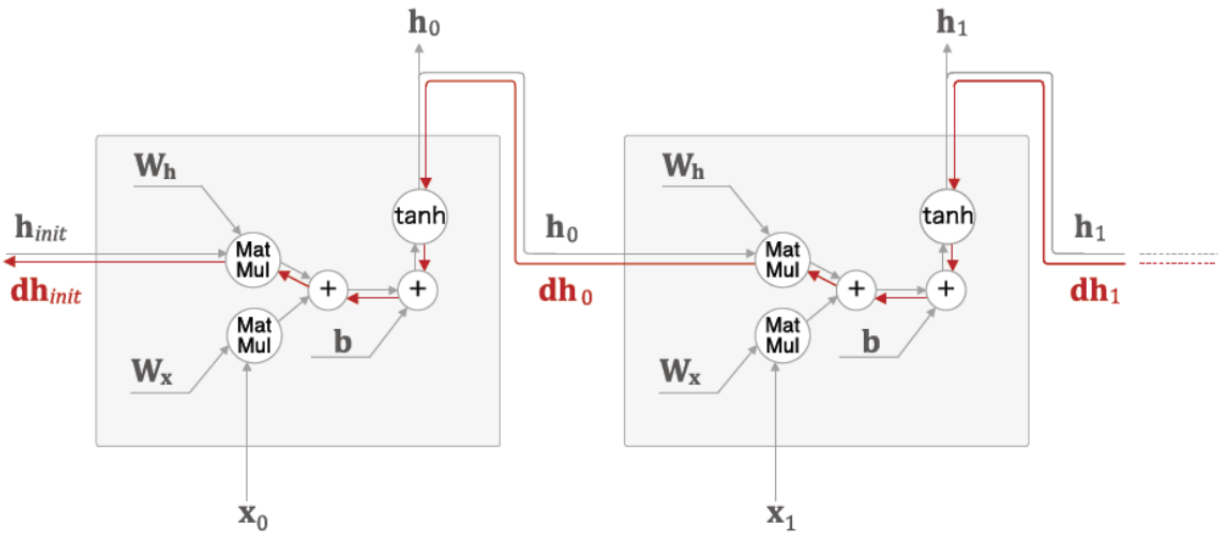
12주차 강의 정리 끝.

(기본적인 RNN 학습 끝.)

13주차 강의 정리 시작.

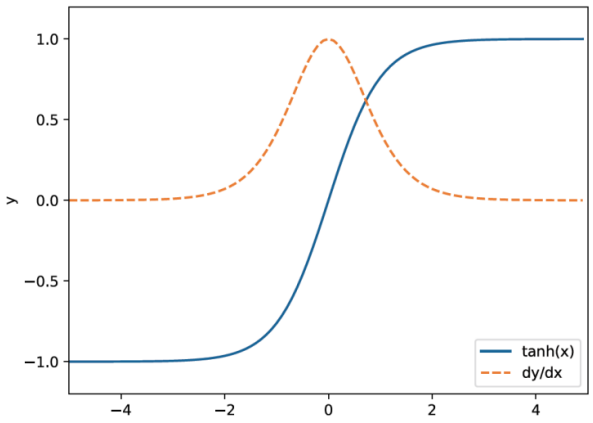
1. 서론
   A. 교재 6장의 내용이다.
   B. 5장에서 배운 RNN은 성능이 좋지 못하다. 이유는 다음과 같다.
      ① 먼 과거의 정보를 잘 기억하지 못한다.
      (장기 의존 관계를 잘 학습하지 못한다.)
      ② 왜 잘 기억하지 못하는가? '기울기 소실 및 폭발' 문제 때문.
      ③ RNNLM은 RNN 계층 여럿이 순차적으로 자신의 정보를 넘기므로, 시계열의 크기가 커질수록 기울기에 문제가 발생할 가능성이 높아진다.
      (이전에 배운 다양한 신경망에서 우리를 괴롭혔던 문제가 똑같이 반복되는 것이다. 순차적으로 값을 넘기다 보면, 문제가 생길 수 밖에 없다.)
2. RNN의 기울기 소실, 폭발
   A. 근본적으로 어느 지점에서 기울기에 문제가 생기는가?
      -> tanh 연산이다.

그림 6-5 RNN 계층에서 시간 방향으로의 기울기 전파



   B.
   C. 역전파에 관여하는 연산은 셋이다.
      ① tanh
      ② +
      ③ MatMul
   D. 여기서 +연산은 정보를 그대로 넘기므로 범인이 아니다.
   E. 첫 번째 문제는 tanh 함수의 도함수이다.

그림 6-6 $y = \tanh(x)$의 그래프(점선은 미분)



F.

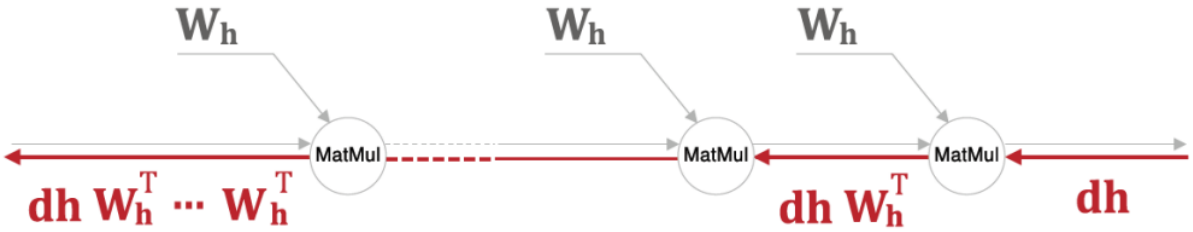
G. 도함수가 0 근처에서만 의미 있는 값을 갖고, 0에서 멀어질수록 급격히 0에 수렴하는 모습이 보이는가? 이것이 문제다!

H. 그러므로 tanh(x)를 T번 통과하면, 기울기가 T번 작아진다.
(tanh(x)의 값은 1을 넘지 않으므로, 1이하의 값을 계속 곱하면 작아질 수 밖에 없다. + tanh(x)의 도함수 형상은 RNN의 순환마다 역전파 값을 급격하게 줄일 수 있다.(0에 가까운 영역 넓음))

I. 그러므로 이전에 Sigmoid 대신 ReLU를 사용했던 것 처럼,
문제 해결을 위한 적절한 대안이 필요하다.

J. 두 번째 문제는 행렬 곱(MatMul)이다.

그림 6-7 RNN 계층의 행렬 곱에만 주목했을 때의 역전파의 기울기



tanh(x)를 잠시 무시하자. 역전파시에, 시계열의 크기만큼 행렬 곱을 반복하게 된다. 만일 W_h의 값이 1이 넘어간다면, 역전파 과정에서 지속적으로 값이 커지고, 기울기 폭발(지수적인 역전파 값의 증가)이 발생할 수 있다.

① W_h는 스칼라 값일 수도 있고 행렬일 수도 있다.

② W_h가 행렬이라면, 행렬의 '특이값'이 발산과 소실의 척도로 평가된다.

③ 특이값: 행렬의 데이터가 얼마나 퍼져 있는지를 나타내는 값.
(혹은, '튀는 값'. 매우 큰 값이나, 다른 값들에서 동떨어진 값이다.)

④ 행렬이 지닌 특이값들 중 최대값이 1보다 크다면 발산이 일어날 수 있다.

3. 기울기 폭발 대책

A. 기울기 클리핑(gradients clipping)

① 기울기의 L2 norm이 임계값을 초과하면 기울기를 공식에 맞게 수정하는 것. <- 이정도만 알고 있으라.

4. 기울기 소실 문제는 해결이 안된다.

A. 기울기 소실 문제는 RNN의 구조에서 기인한다.

B. 즉, RNN에다가 무슨 기법을 적용해도 소실 문제를 피하기 어렵다.

C. 그리하여 근본적으로 RNN을 뜯어 고치게 되는데….
그 결과물이 LSTM이다.

5. LSTM

= 게이트가 추가된 RNN 중 하나. (다양한 아키텍처가 제안되어 있다.)

A. LSTM에서는 RNN에 다양한 요소를 추가한다.

① 기억 셀

② output 게이트

③ forget 게이트

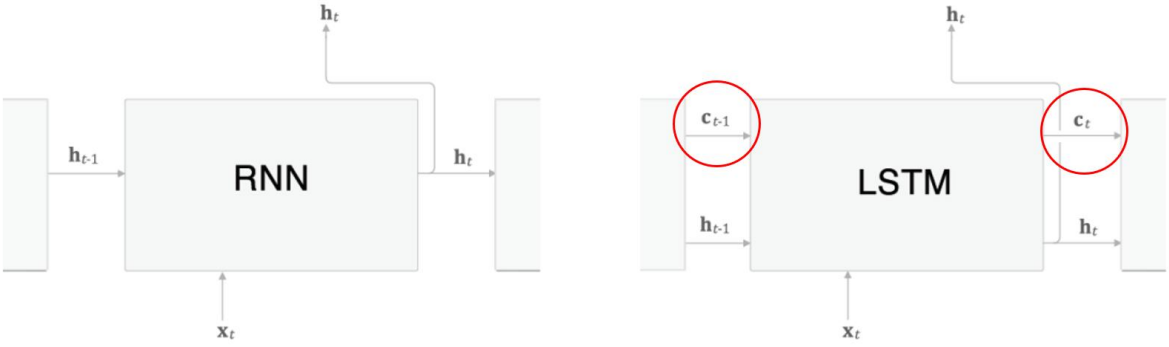
④ input 게이트

6. 기억 셀(C_t)

= 시각t 까지의 모든 중요한 정보가 저장되어 있는 셀.

=

그림 6-11 RNN 계층과 LSTM 계층 비교



A. LSTM은 기억 셀을 바탕으로, 은닉 상태h_t를 만들어서 출력한다.

① C_{t-1}: 직전 시각까지의 단기기억.

② h_{t-1}: 직전 시각까지의 단기기억. //그러므로 C_{t-1}와 h_{t-1}은 같다.

③ X_t: 입력 값.

④ C_t: 장기기억. <- 시각t 에 필요한 모든 정보가 저장되어 있으므로!

⑤ h_t: 예측 값. (다음 노드에게, 이전 노드의 예측 값으로 투입됨.)
구체적으로는, $h_t = \tanh(C_t)$ 이다.
(C_t와 h_t의 원소 개수가 같다.)

7. 게이트
= 입력 값을 얼마나 통과시킬 지 결정하는 노드.
= 입력을 0~100% 사이의 비율로 축소하는 문.

8. output 게이트
= h_t(다음 은닉 상태)의 출력을 조절하는 게이트.

A. 이후, o게이트라고 칭하겠다.

B. o게이트는 자신만의 가중치 행렬을 갖는다.

C. o게이트는 [현 시각의 입력 X_t]와 [이전 시각의 예측 h_{t-1}]을 받아서, [LSTM 계층이 출력할 h_t]를 조작한다.

✓ output 게이트의 열림 상태는 입력 x_t와 이전 상태 h_{t-1}로부터 구함.
✓ 이때의 식은 다음과 같음.

$$\mathbf{o} = \sigma(\mathbf{x}_t \mathbf{W}_{\mathbf{x}}^{(\mathbf{o})} + \mathbf{h}_{t-1} \mathbf{W}_{\mathbf{h}}^{(\mathbf{o})} + \mathbf{b}^{(\mathbf{o})})$$

✓ 가중치와 편향의 첨자 o는 output의 첫 글자를 의미하며, 시그모이드 함수를 $\sigma(\,)$로 표기.
✓ 마지막으로 출력 **o**와 $\tanh(c_t)$의 원소별 곱(아다마르 곱)을 h_t로 출력.

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

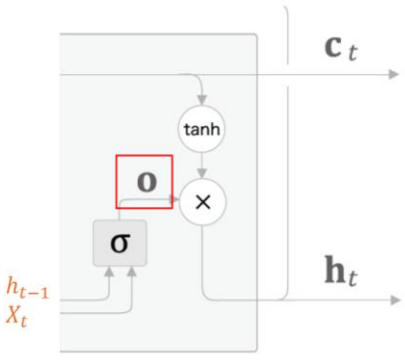
D.

① o게이트로의 두 입력을 각각 가중치에 곱하고, 시그모이드로 감싼다.

② 시그모이드의 출력과, 원본 h_t를 원소별로 곱셈시킨다.

③ 원소별 곱의 결과가 새로운 h_t이고, 다음 LSTM으로 넘어간다.

E.



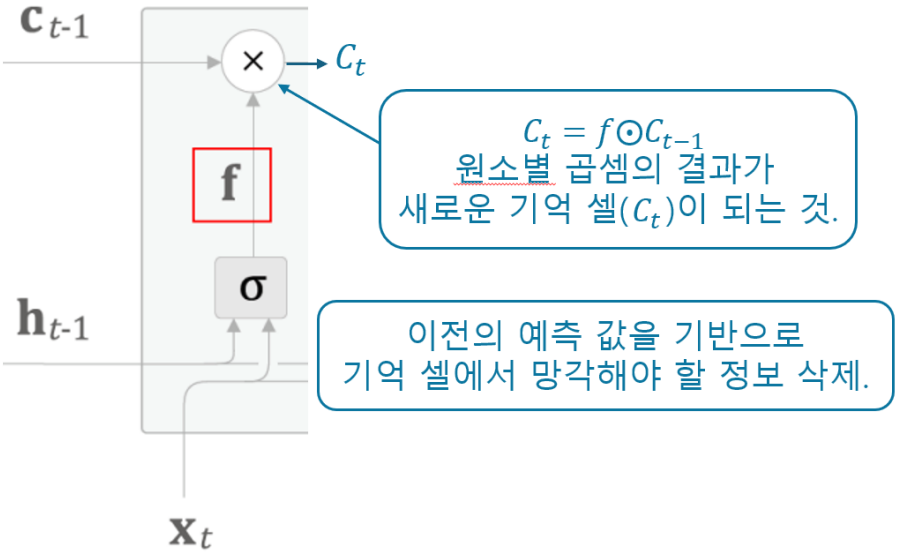
9. forget 게이트

= 기억 셀에서 불필요한 기억을 삭제하는 게이트.

$$\mathbf{f} = \sigma(\mathbf{x}_t \mathbf{W}_x^{(\mathbf{f})} + \mathbf{h}_{t-1} \mathbf{W}_\mathbf{h}^{(\mathbf{f})} + \mathbf{b}^{(\mathbf{f})})$$

A. f게이트도 내부적으로 자신만의 가중치 매개변수를 갖는다.

B. f게이트도 시그모이드로 감싸여 있으며,
출력 f를 이전 기억 셀C_{t-1}에 원소별 곱셈시킨다.



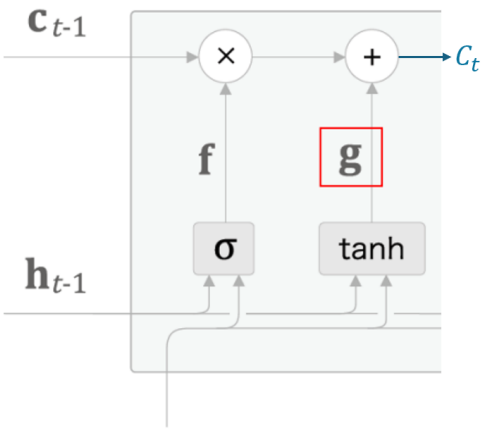
10. 기억 셀에 새로운 정보 추가

A. 특수한 연산의 결과(g = *추가할 기억*)를 기존 기억 셀에 덧셈할 것이다.

B. $$\mathbf{g} = \tanh(\mathbf{x}_t \mathbf{W}_x^{(\mathbf{g})} + \mathbf{h}_{t-1} \mathbf{W}_\mathbf{h}^{(\mathbf{g})} + \mathbf{b}^{(\mathbf{g})})$$

C. g는 게이트가 아니다. 하지만 자신만의 가중치 매개변수를 갖는다.

D. [[자신만의 가중치]와 [입력값]을 행렬 곱 시킨 값]들을 하이퍼볼릭 탄젠트로 감싸서, 기존의 C_{t-1}에 더한다.



E. 이전의 예측 값을 기반으로 기억 변화!

11. input 게이트

= 앞서 g 연산의 결과에 새로운 정보를 추가하는 게이트이다.

$$\mathbf{i} = \sigma(\mathbf{x}_t \mathbf{W}_x^{(\mathbf{i})} + \mathbf{h}_{t-1} \mathbf{W}_\mathbf{h}^{(\mathbf{i})} + \mathbf{b}^{(\mathbf{i})})$$

A. i게이트도 자신만의 가중치 매개변수를 갖는다.

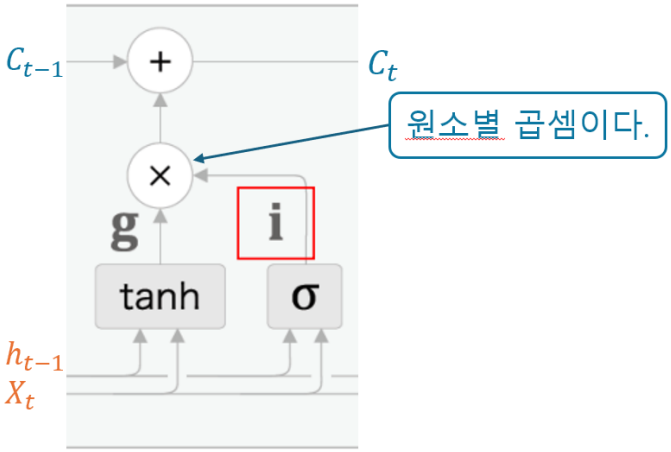
B. 기존의 g연산은 새로운 정보를 무비판적으로 수용시킨다.

-> 우리는 input 게이트를 통해, 어느 정보가 더 중요한지를 정해준다.

(새로 추가되는 정보 각각의 가치를 판단.)

C. f게이트, o게이트와 마찬가지로, [h_{t-1}과 X_t]를 입력받아서 [자신만의 가중치]에 행렬곱 하여 시그모이드로 감싼다. 그리고 [그 값]을 [g연산의 결과]와 원소별 곱셈하여 C_{t-1}에 덧셈한다.

D.



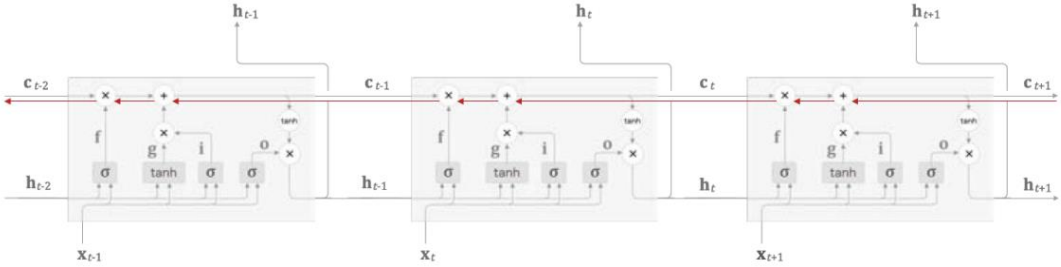
12. 기울기 소실이 해결된 이유

A. 지금껏 다양한 장치를 RNN에 추가하는 것을 보았다면,

기억 셀에 세심하게 정보를 추가하고 삭제했음을 잘 이해했을 것이다.

B. 기억 셀이 역전파 될 때, 덧셈과 곱셈 노드만이 영향을 미치도록 설계되어 있다. 이때 곱셈 노드가 행렬 곱이 아닌, 원소별 곱(아다미르 곱)이므로, 지수적인 기울기 폭발은 일어날 수 없다. (곱셈 효과 누적 안됨.)

C.



D. 한편, 게이트의 출력값이 매번 다르다는 것도 좋은 영향을 준다. (곱셈 효과 누적 억제. RNN에서는 동일한 가중치를 반복적으로 행렬곱 했다.)

13. Time LSTM

= T개의 LSTM을 묶어서 한번에 처리한다.

A. 다만 시계열의 크기를 무한히 크게 할 수 없다.

B. 일정 단위로 나누어 계산한다. <- 일반적으로 100회.

14. RLSTM

= 다층 LSTM.

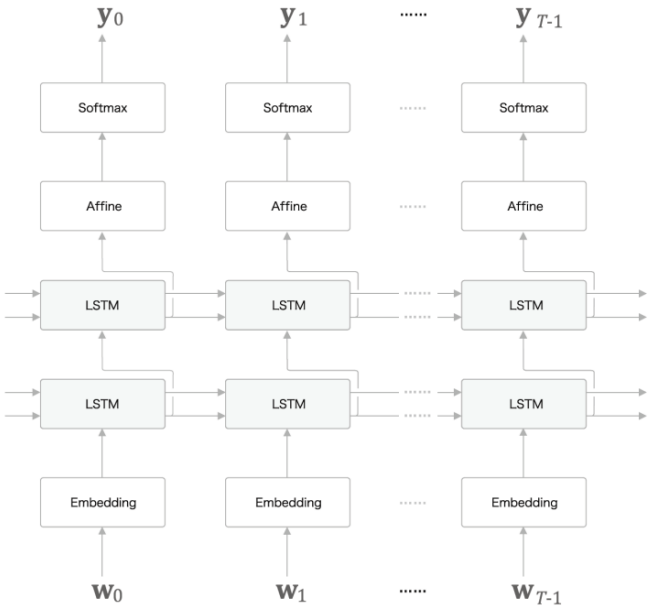
A. 일반적으로 2~4층을 쌓았을 때 성능 향상이 좋았다고 한다.

B. 하지만 꼭 몇 층을 쌓아야 한다는 것이 정해져 있는 것은 아니다.

C. 적층은 내부적으로 이루어지는 것이다. Embedding과 Affine 사이!

D.

그림 6-29 LSTM 계층을 2층으로 쌓은 RNNLM



15. 드롭아웃(dropout)

= 훈련 시, 계층 내의 일부 뉴런을 무작위로 무시하고 학습하는 것.

A. RNN은 쉽게 과적합을 일으키는 경향이 있다. -> 과적합 대책이 중요!

B. 전통적인 과적합 대책

① 훈련 데이터의 양 늘리기

② 모델 복잡도 줄이기

③ 정규화

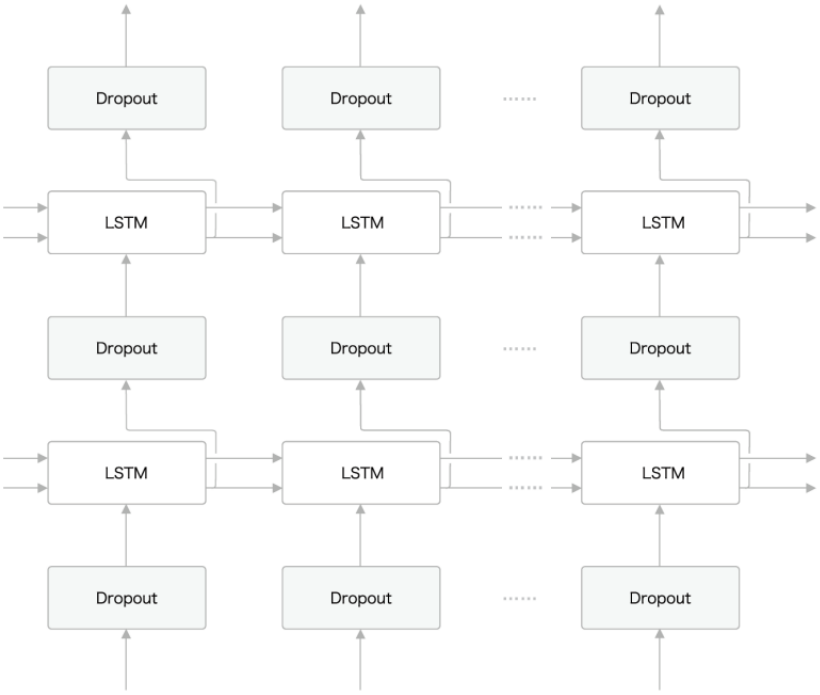
C. 그런데 우리는 LSTM을 적층하고 싶지 않은가?
이때 적용할 수 있는 방법으로 드롭아웃이 있다.

D. 드롭아웃은 일종의 정규화로 볼 수 있다.

E. 중요! 드롭아웃을 시계열 방향으로 하는 것은 의미가 없다.
(시간의 흐름에 따라 노이즈가 축적된다.)

F. 그러므로 드롭아웃은 계층 사이에 투입하는 것이 좋다.

그림 6-33 좋은 예: 드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입



G.

H. 하지만 꼭 계층 사이에만 넣어야 하는 것은 아니다. 다양한 방법으로 넣을 수 있다.

16. 가중치 공유

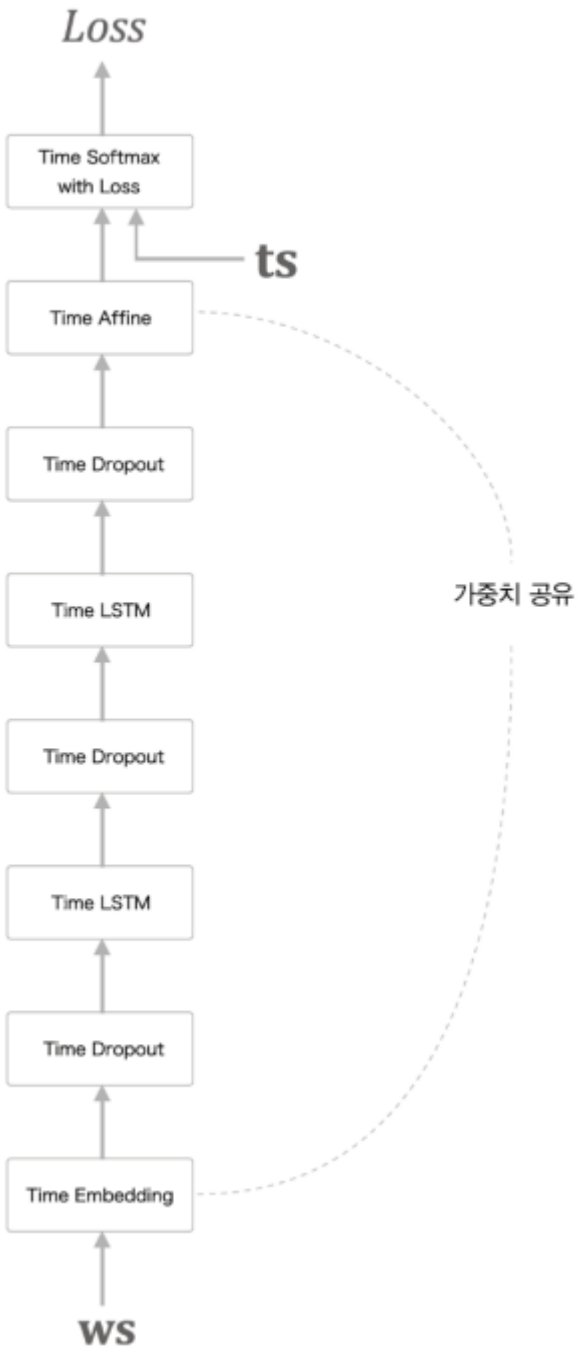
= 신경망 내부의 일부 가중치를 서로 공유하도록 만드는 것.

A. 일반적으로 Embedding 계층과 Affine 계층의 가중치를 공유한다.

B. 이렇게 되면, 실제로 학습해야 하는 매개변수의 숫자가 줄어든 효과가 발생한다. 그러므로 과적합이 억제된다.

17. 개선된 RNN

그림 6-36 BetterRnnlm 클래스의 신경망 구성



13주차 강의 정리 끝.

14주차 강의 정리 시작.

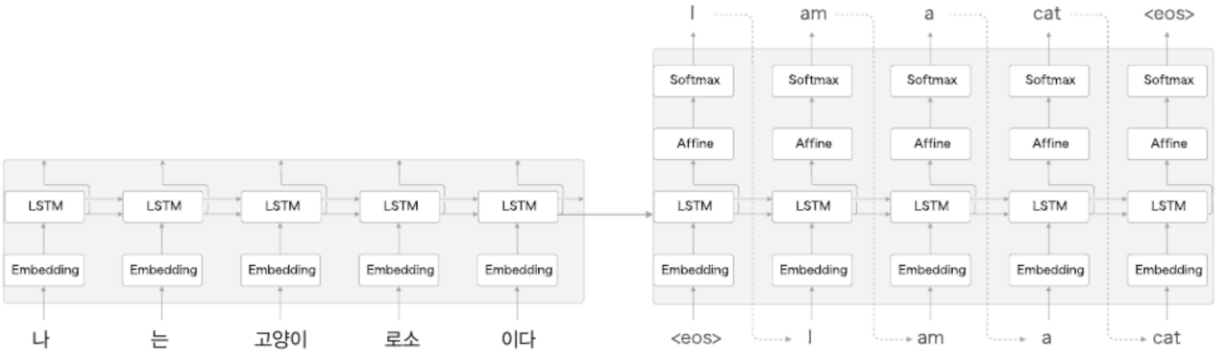
1. 서론
   A. 14주차 강의에서는 교재의 7장과 8장을 학습한다.
   B. 시간이 없어서 간결하게 정리해 보겠다.
   C. 7장: seq2seq 학습.
   D. 8장: 어텐션 학습.

2. 언어 모델의 이해
   A. 언어 모델은 다음에 출력할 단어의 확률 분포를 출력한다.
   B. 이 확률 분포에서 단어를 고르는 방법은 두 가지가 있다.
      ① 확률적 샘플링: 확률이 높으면 선택될 가능성이 높지만 확정은 아님.
      ② 결정적 샘플링: 확률이 가장 높은 단어를 선택하는 등의 방법들.

3. 문장 생성
   = 학습된 모델에 입력을 넣고, 예측한 단어를 뽑아내는 것.
   (우리 교재에서는 RNN, 단어 뽑기는 확률적이든 결정적이든 개발자 마음)
   A. 그리하여 뽑아낸 단어들을 누적하여 기록하면 '문장'이 된다.

4. seq2seq
   = 두 개의 RNN으로 시계열 데이터를 입출력하는 모델.
   = Encoder-Decoder 모델.
   A. 문장을 Encoder에 집어넣으면, 문장 전체를 고정 길이 벡터로 인코딩한다.
      ① 인코딩: 문장을 임의의 길이를 갖는 고정 길이 벡터로 변환.
   B. Encoder의 출력 h는?
      ① Encoder의 마지막 RNN의 출력.
      ② Decoder의 입력.
      ③ 벡터화된 입력 문장. (은닉 벡터)
      ④ 문장의 컨셉(의미)가 압축된 벡터.
   C. Decoder는 고정 길이 벡터 h를 받아서, 출력해야 할 값을 계산한다.
   D. 우리는 Encoder가 내놓은 h와, Decoder가 내놓은 출력이 동일 의미이기를 바란다. (번역 문제를 생각하라.)

그림 7-9 seq2seq의 전체 계층 구성



E.

F. 이때 Encoder의 상단 출력 값들은 모두 버려진다.

5. seq2seq의 개선 방법

A. 입력 데이터 반전.

① 학습 속도 개선으로, 신경망의 성능을 향상시키는데 도움 준다.

B. 엿보기(peeky)

① Enc의 출력 h가 굉장히 중요한데,
Dec의 첫 LSTM만 h를 입력받고 있다….
즉, 이후의 LSTM들은 원본 h를 볼 수 없다.

② 해결법: 벡터를 [[원본h] + [이전 LSTM의 출력]]으로 바꾼다.
(벡터가 다차원이 되었다고 생각해도 좋다.
핵심은, 모든 시각의 LSTM계층이 [원본 h]를 입력 받아야 한다는 것.)

6. 7장 요약

A. 시계열 데이터 -> Enc -> 은닉벡터h -> Dec -> 시계열 데이터

B. 개선법: reverse input, peeky

7. 어텐션 메커니즘 (8장)

= seq2seq를 더 강력하게 개선하는 방법.

A. seq2seq의 문제

① Enc가 고정길이 벡터를 출력함.
-> Enc에 입력되는 문장의 길이가 무시된다.

B. 해결법

① Enc에서 버려졌던 상부 방향으로의 출력(h)들을 모아서 집합(hs)으로 만들고 출력시킨다. (h를 hs로 대체)

② 그리고 hs를 처리 가능한 Dec를 새로 설계한다.

C. 아이디어

= 단어들의 대응관계를 사람이 아닌 기계가 계산하도록 하자!

(대응관계(얼라인먼트)를 기계가 직접 알아내도록 한다.)

① 즉, hs에서 어떤 단어에 '주목'해야 하는지를 알아내어, 시계열 변환에 사용해야 한다. (이 정보는 후술할 맥락 벡터에 포함된다.)

D. 맥락 벡터 C 구하기

맥락 벡터 = 지금 당장의 번역에 필요한, 입력 단어를 보았을 때 집중해야 할 속성들이 주로 포함되어 있는 벡터.

E. 맥락 벡터 계산법.

① hs에 가중치 a를 곱한다. -> 중요한 단어의 값이 강조됨.

② 강조된 hs를 열방향으로 모두 합한다. (a는 모두 1 이하의 확률값이므로, hs를 열방향으로 모두 더해도 괜찮다.)

③ 합해진 hs는 **맥락 벡터인 C이다! <- 가중치 a에서 중요하다고 지정한 단어의 의미들이 더 진하게 남아 있는 벡터이다.**

F. 얼라인먼트 제작을 위해, 선택 문제를 '모두 선택'으로 변환해야 한다.

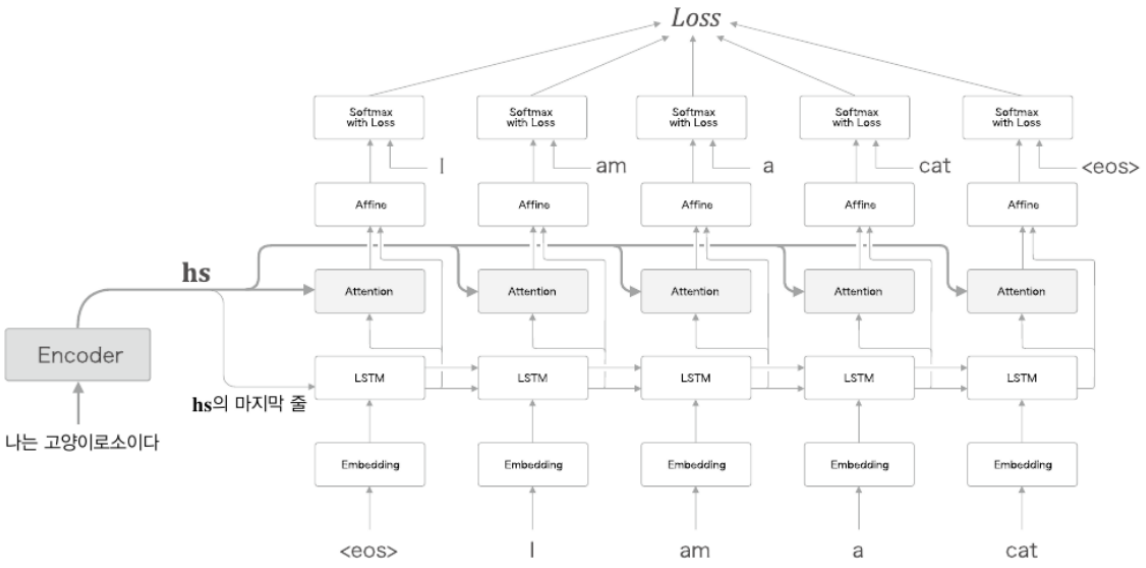
(미분 가능하도록)

① 이것을 맥락 벡터가 가능하게 한다.

G. Dec 개선

① 우리는 맥락 벡터를 Dec 내부에서 계속 계산하도록 할 것이다.

그림 8-18 Attention 계층을 갖춘 Decoder의 계층 구성

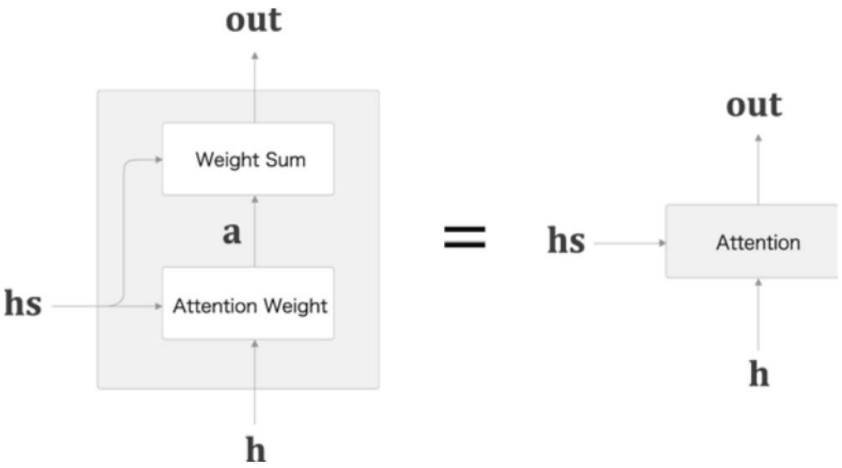


②

③ Enc는 hs를 출력하여 Dec의 Attention 계층에 계속 투입한다.

④ Attenction 계층에서는, 현재의 예측값 h와, hs를 내적하여, 가중치 a를 구하고, 맥락 벡터를 만들어서 출력한다. 이 출력값이 Affine과 Softmax를 통과하므로, [맥락 벡터 = h와 얼라인먼트 된 짝]인 것이다.

그림 8-17 왼쪽 계산 그래프를 Attention 계층으로 정리



⑤

⑥ hs의 모든 행과 h를 내적하여 유사도를 구하고 -> 유사도 스코어를 Softmax에 넣어서 나오는 확률이 a인 것이다.

⑦ 이때 이 a값을 hs에 스칼라 곱하여 hs를 변화시키고 -> hs를 열 방향으로 모두 더하여, 맥락 벡터를 구한다.

⑧ 이 맥락 벡터는 h와 짝이 되는 의미를 포함한다!

8. 트랜스포머

A. RNN은 병렬 처리가 어렵다. (이전 결과를 넘겨 받아야 함.)

B. 그러므로 RNN을 없애고 어텐션만 남기자는 주장도 활발하다.

14주차 강의 요약 끝.