

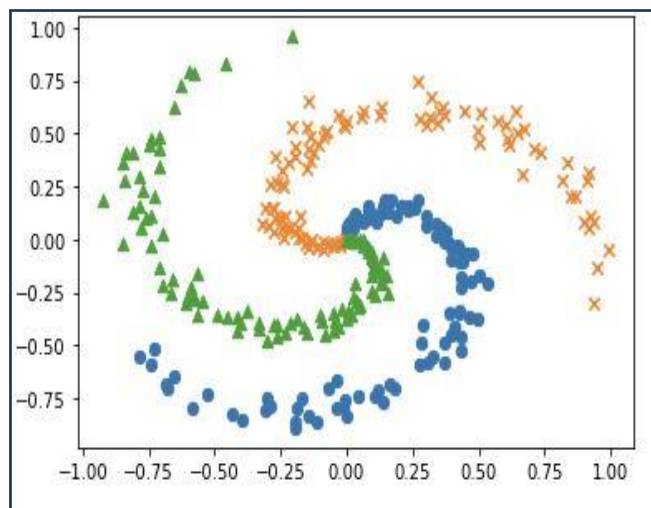
밑바닥부터 시작하는 딥러닝 2

통계학과 구병모

✓ 스파이럴 데이터셋

- 입력: 2차원 데이터 / 분류할 클래스 수: 3개
- 비선형 분리를 학습하기 위한 데이터

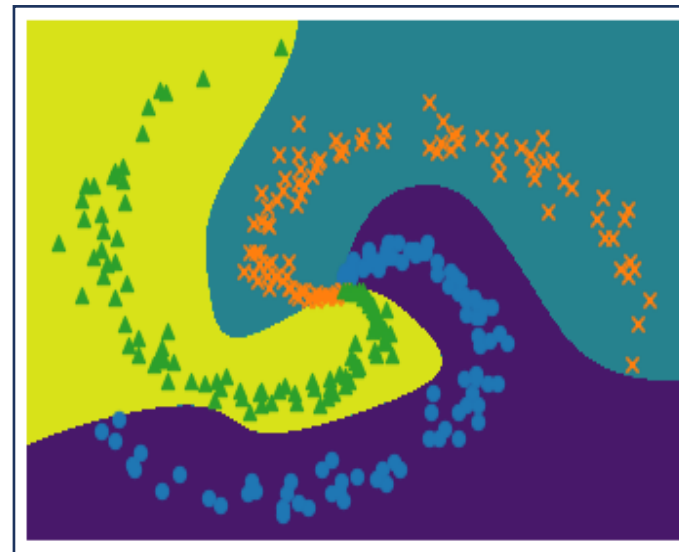
• 입력 데이터



• 학습

에폭	283	반복	10 / 10	손실	0.11
에폭	284	반복	10 / 10	손실	0.11
에폭	285	반복	10 / 10	손실	0.11
에폭	286	반복	10 / 10	손실	0.11
에폭	287	반복	10 / 10	손실	0.11
에폭	288	반복	10 / 10	손실	0.12
에폭	289	반복	10 / 10	손실	0.11
에폭	290	반복	10 / 10	손실	0.11
에폭	291	반복	10 / 10	손실	0.11
에폭	292	반복	10 / 10	손실	0.11
에폭	293	반복	10 / 10	손실	0.11
에폭	294	반복	10 / 10	손실	0.11
에폭	295	반복	10 / 10	손실	0.12
에폭	296	반복	10 / 10	손실	0.11
에폭	297	반복	10 / 10	손실	0.12
에폭	298	반복	10 / 10	손실	0.11
에폭	299	반복	10 / 10	손실	0.11
에폭	300	반복	10 / 10	손실	0.11

• 학습 완료(결정 경계)



✓ Trainer 클래스

- for. 신경망 학습을 수행하는 역할
- fit () 매서드 / plot () 매서드
- trainer = Trainer(model, optimizer)

✓ 계산 고속화

- 비트 정밀도: 32비트 부동소수점 활용(np.float32, 'f') → 64비트 데이터 타입에 비해 메모리 관점에서 유리
→ 학습된 가중치 저장하는 경우: 16비트 부동소수점 활용
- GPU(쿠파이): 병렬 계산에 유리 but 쿠파이는 엔비디아의 GPU만

✓ 자연어 처리(NLP)

- '우리의 말을 컴퓨터에게 이해시키기 위한 기술(분야)'
- '프로그래밍 언어', '마크업 언어': 기계적, 고정적 / 자연어: 부드러움
- 활용 분야: 검색 엔진, 기계 번역, 질의응답 시스템, IME(입력기 전환), 문장 자동요약, 감정분석
- 세 가지 기법: 시소러스 / 통계 기반 / 추론 기반

✓ 시소러스

- 유의어 사전: 뜻이 같은 단어(동의어), 뜻이 비슷한 단어(유의어)
- 단어 사이의 '상위와 하위', '전체와 부분' → 세세한 관계 정의(계층 구조)
- 유의어 집합 → 그래프로 표현 → '단어 네트워크' → 단어 사이의 관계
- 문제점: 시대 변화 대응 hard / 인적 비용 ↑ / 미묘한 차이 표현 x
- Wordnet: 가장 유명한 시소러스 / 유의어, 단어 네트워크, 유사도 도출 가능

```
car.lemma_names()
```

```
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

```
print(car.path_similarity(novel))  
print(car.path_similarity(dog))  
print(car.path_similarity(motorcycle))
```

```
0.055555555555555555  
0.07692307692307693  
0.3333333333333333
```

```
car.hypernym_paths()[0]
```

```
[Synset('entity.n.01'),  
Synset('physical_entity.n.01'),  
Synset('object.n.01'),  
Synset('whole.n.02'),  
Synset('artifact.n.01'),  
Synset('instrumentality.n.03'),  
Synset('container.n.01'),  
Synset('wheeled_vehicle.n.01'),  
Synset('self-propelled_vehicle.n.01'),  
Synset('motor_vehicle.n.01'),  
Synset('car.n.01')]
```

✓ 통계 기반 기법

- 말뭉치: 대량의 텍스트 데이터 for 자연어 처리 연구 → 자연어에 대한 사람의 '지식'
- ex) 위키백과, 구글 뉴스
- 말뭉치 전처리: 소문자 변환 / id_to_word / word_to_id / corpus

```
text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
print(corpus)
print(word_to_id)
print(id_to_word)
```

```
[0 1 2 3 4 1 5 6]
{'you': 0, 'say': 1, 'goodbye': 2, 'and': 3, 'i': 4, 'hello': 5, '.': 6}
{0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}
```

그림 2-3 윈도우 크기가 2인 '맥락'의 예. 단어 "goodbye"에 주목한다면, 그 좌우의 두 단어(총 네 단어)를 맥락으로 이용한다.

you say goodbye and i say hello.

- 단어의 분산 표현: 단어도 벡터로 표현 가능
 - 분포 가설: '단어의 의미'는 주변 단어(맥락)에 의해 형성 / 맥락의 크기 = 윈도우 크기
- 주변 단어 세어보기 = 통계 기반 기법

- 동시발생 행렬: create_co_matrix
- 벡터 간 유사도: 코사인 유사도
- 유사 단어의 랭킹: most_similar(query, word_to_id, id_to_word, word_matrix, top=5)

```
most_similar('you', word_to_id, id_to_word, C, top=5)
```

```
[query] you
goodbye: 0.7071067691154799
i: 0.7071067691154799
hello: 0.7071067691154799
say: 0.0
and: 0.0
```

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{x_1 y_1 + \dots + x_n y_n}{\sqrt{x_1^2 + \dots + x_n^2} \sqrt{y_1^2 + \dots + y_n^2}}$$

- 과정: 말뭉치 사용 → 맥락에 속한 단어의 등장 횟수(동시발생 행렬) → PPMI 행렬 변환 → SVD 차원 감소 → 좋은 단어 벡터

✓ 통계 기반 기법 개선 1. 상호정보량

- 상호정보량: 동시 발생 행렬 = 두 단어가 동시에 발생한 횟수 but 관련성 판단 어렵
- 점별 상호정보량(PMI): 값 ↑ = 관련성 ↑

$$\text{PMI}(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x,y)}{N}}{\frac{C(x)}{N} \frac{C(y)}{N}} = \log_2 \frac{C(x,y) \cdot N}{C(x)C(y)}$$

- 양의 상호정보량(PPMI): for $\log_2 0 = -\infty$ 피하기 / $\text{PPMI}(x,y) = \max(0, \text{PMI}(x,y))$
- 문제점: 말뭉치 어휘 수 ↑ → 각 단어 벡터의 차원 수 ↑

✓ 통계 기반 기법 개선 2. 차원 감소

- 특잇값분해(SVD): $\mathbf{X} = \mathbf{USV}^T$
- U 행렬: 단어 공간 / S 행렬: 대각성분 '특잇값', 큰 순서대로
- svd 메서드: 넘파이의 linalg 모듈이 제공

✓ PTB 데이터셋(펜 트리뱅크)

- 주어진 기법의 품질 측정하는 벤치마크
- ptb.load_data(): train, test, valid 데이터

✓ 어려웠던 점

✓ 궁금한 점

- 특잇값 분해(SVD) 시 S 대각 행렬의 성분이 특잇값이 큰 순서대로 나열되어 있는데, 이는 모듈 자체에서 따로 처리를 하는 것인지?

✓ 통계 기반 기법의 문제점

- SVD를 $n \times n$ 행렬에 적용하는 비용 $O(n^3)$, 배치 학습(미니 배치 \times) \rightarrow 계산량 \uparrow , 학습 속도 \downarrow

✓ 추론 기반 기법

- 주변 단어(맥락)이 주어졌을 때 타겟에 무슨 단어가 들어가는 지 추측
- 추론 문제 반복 \rightarrow 단어의 출현 패턴 학습
- 고정 길이 벡터 변환(원핫 표현) \rightarrow 완전연결계층

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

you ? goodbye and I say hello.

그림 3-3 추론 기반 기법: 맥락을 입력하면 모델은 각 단어의 출현 확률을 출력한다.

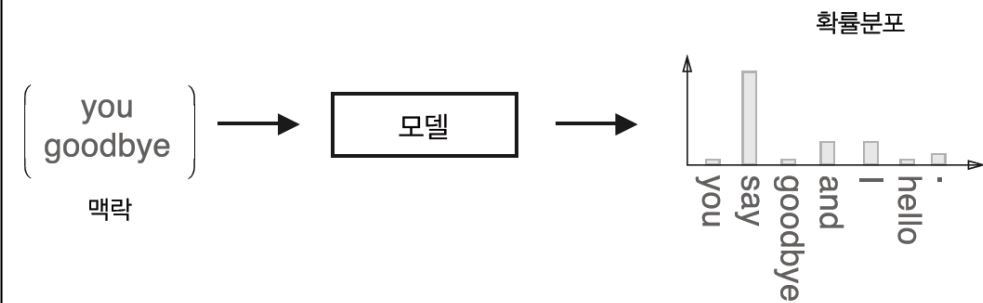
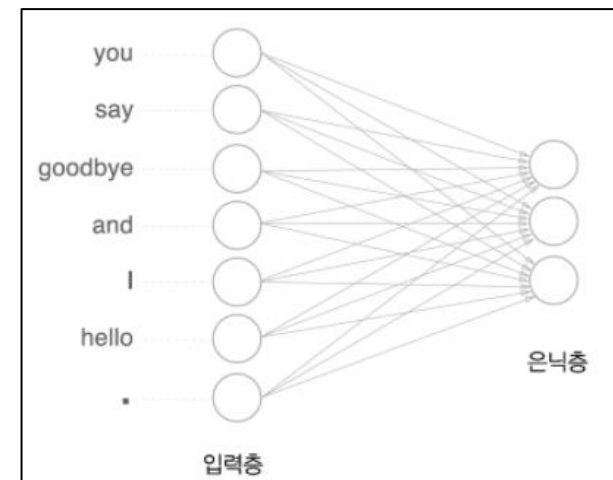
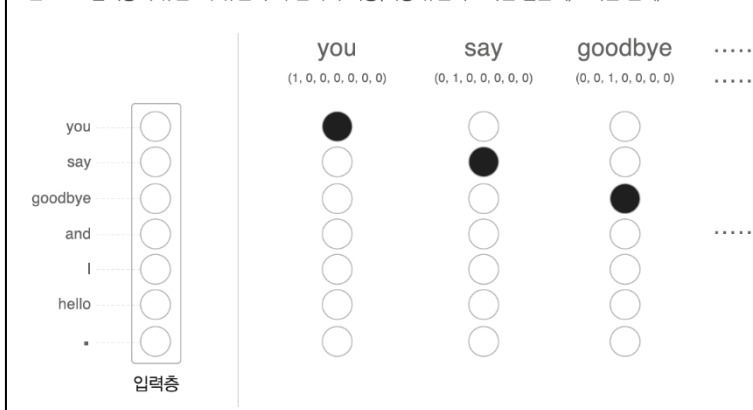


그림 3-5 입력층의 뉴런: 각 뉴런이 각 단어에 대응(해당 뉴런이 1이면 검은색, 0이면 흰색)



✓ CBOW 모델

- 맥락(주변 단어) → 타깃(중앙 단어) 추측하는 용도의 신경망
- 입력층: 맥락으로 고려할 단어 / 출력층: 타깃의 점수(확률 when 소프트맥스)
- 은닉층의 뉴런 수 < 입력층의 뉴런 수 = 단어 예측에 필요한 정보 간결, 밀집벡터 표현
- 가중치 각 행 = 단어의 분산 표현

그림 3-9 CBOW 모델의 신경망 구조

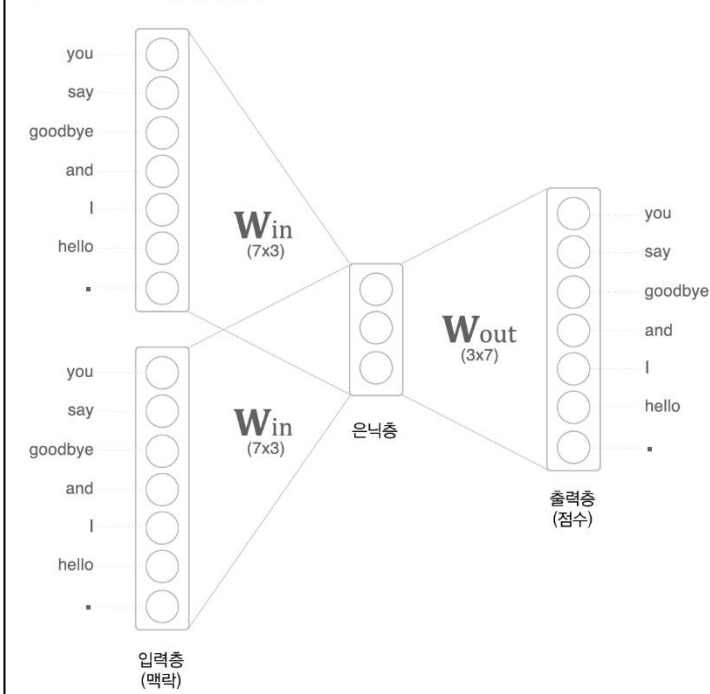
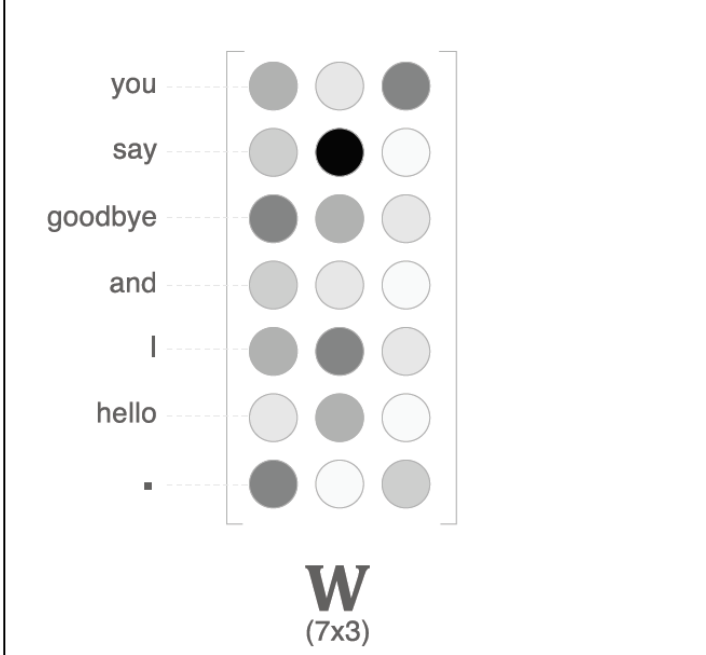
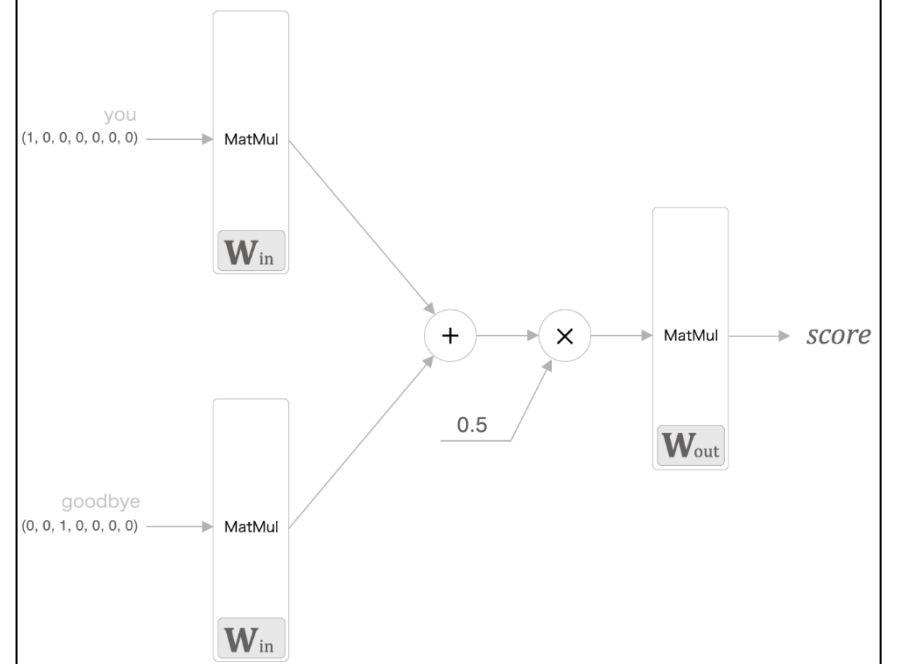


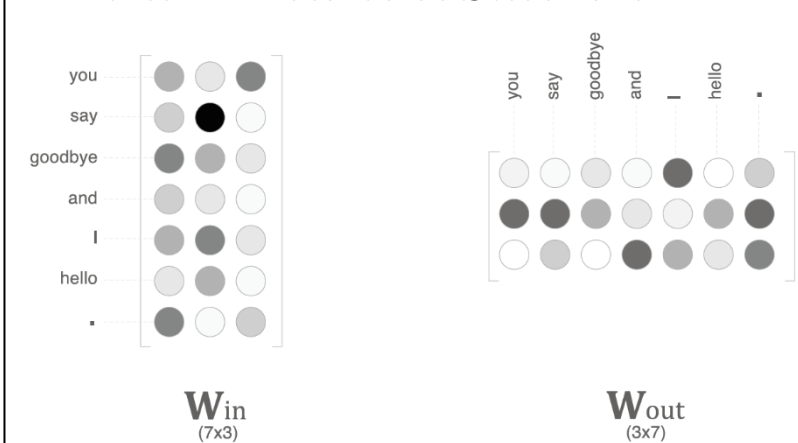
그림 3-10 가중치의 각 행이 해당 단어의 분산 표현이다.

그림 3-11 계층 관점에서 본 CBOW 모델의 신경망 구성: MatMul 계층에서 사용하는 가중치(W_{in} , W_{out})는 해당 계층 안으로 넣었음

✓ CBOW 모델 학습

- 다중 클래스 분류 신경망 → 소프트맥스 함수 & 교차 엔트로피 오차 → 손실

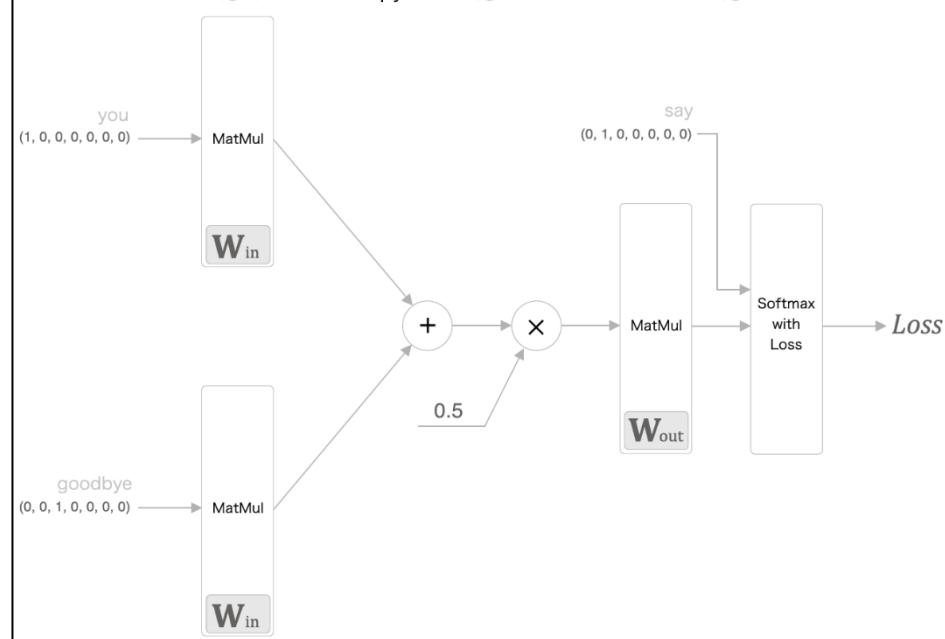
그림 3-15 각 단어의 분산 표현은 입력 층과 출력 층 모두의 가중치에서 확인할 수 있다.



✓ Word2vec의 가중치 & 단어의 분산 표현

- 입력 층 완전연결계층의 가중치 & 출력 층 완전연결계층의 가중치
- 입력 층 가중치만(대중적) / 출력 층 가중치만 / 양쪽 가중치 모두

그림 3-14 Softmax 계층과 Cross Entropy Error 계층을 Softmax with Loss 계층으로 합침



✓ 맥락과 타깃

- 신경망 입력 = 맥락 & 맥락에 둘러싸인 중앙 단어 = 타깃
- Corpus(단어 ID 배열) → 맥락, 타깃 → 원핫 표현
- contexts, target = create_contexts_target(corpus, window_size)

그림 3-18 '맥락'과 '타깃'을 원핫 표현으로 변환하는 예

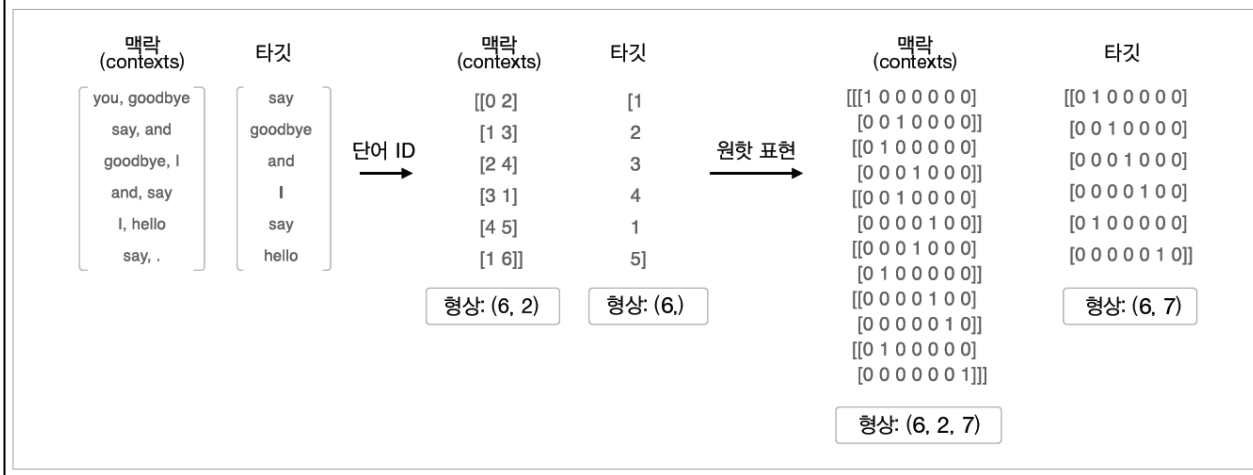
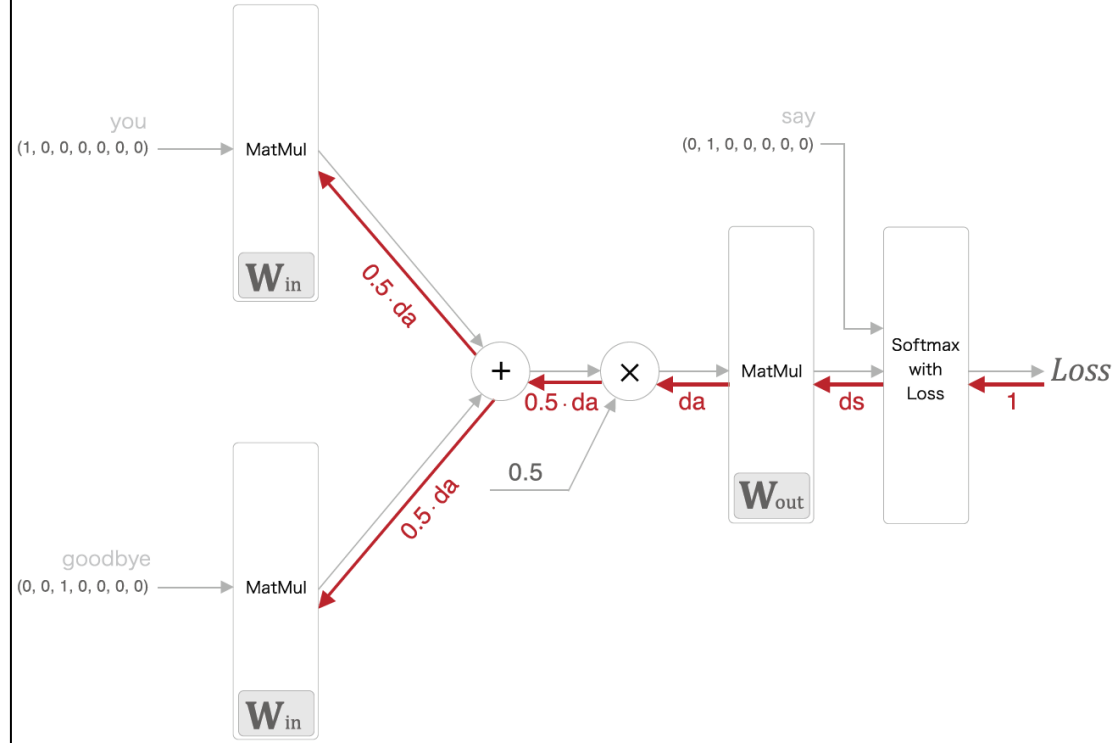
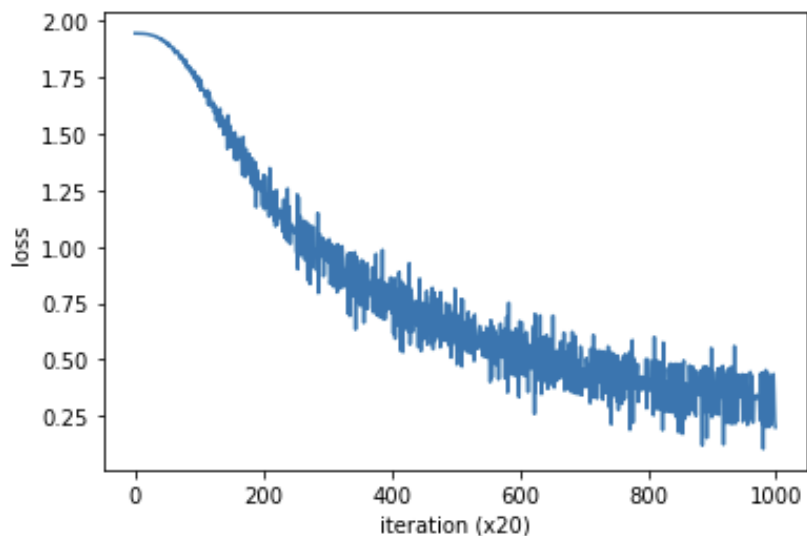


그림 3-20 CBOW 모델의 역전파(역전파의 흐름은 두꺼운(붉은) 화살표로 표시)



✓ SimpleCBOW 학습 결과

- 매개변수 갱신 방법: Adam
- 학습 → 손실 ↓
- 입력 측 MatMul 계층의 가중치 = word_vec → 단어의 분산 표현



```
word_vecs = model.word_vecs
for word_id, word in id_to_word.items():
    print(word, word_vecs[word_id])
```

you [-0.96249646 -0.96506 -0.9238251 1.685398 -0.9509637]
say [1.140355 1.1552193 0.04356381 1.2152562 1.1311442]
goodbye [-1.0647339 -1.0327902 -0.9811917 -0.5848243 -1.0662838]
and [0.80858517 0.75630385 1.8961017 1.2090446 0.9532468]
i [-1.0341116 -1.0040611 -0.956608 -0.5786802 -1.0499307]
hello [-0.9891179 -0.9725827 -0.9140799 1.6740813 -0.9686472]
. [1.1663282 1.1842409 -1.643168 0.7456128 1.026521]

✓ CBOW 모델 = 타깃 단어 출현 확률 출력 given 맥락 / $P(w_t | w_{t-1}, w_{t+1})$ / 손실함수: $L = -\log P(w_t | w_{t-1}, w_{t+1})$

✓ skip-gram 모델: 타깃(중앙의 단어) → 맥락(주변의 여러 단어) 추측 / $P(w_{t-1}, w_{t+1} | w_t)$

- 단어의 정밀도: skip-gram! / 학습 속도: CBOW!

✓ 어려웠던 점

✓ 궁금한 점

- CBOW 모델 & skip-gram 모델에서 입력 측 가중치만 이용하는 것이 대중적인데, 이유가 궁금합니다
- CBOW 모델 구현 시 편향을 제거한다는 가정하에 했는데 실제 연구를 할 때는 편향을 넣어주는지 궁금합니다
- GloVe 기법에 대해서 궁금.

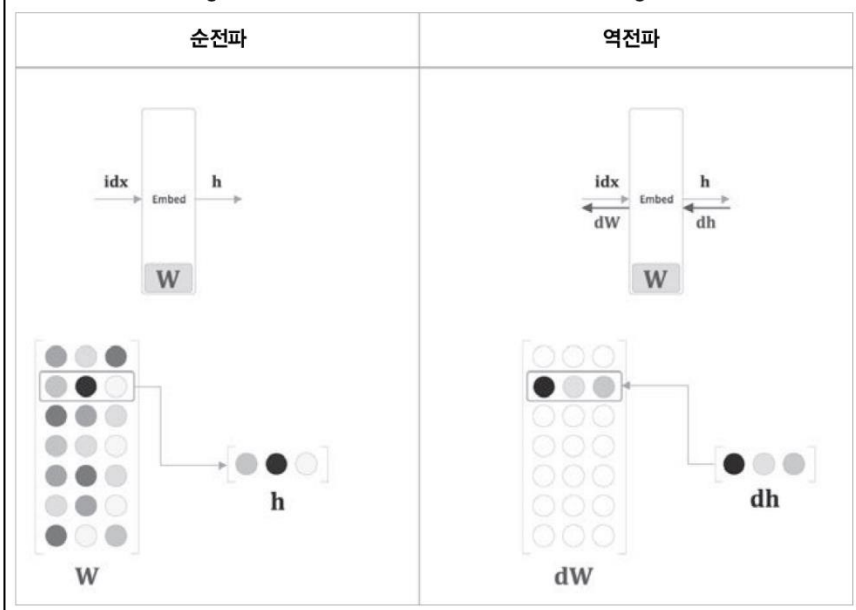
✓ 다루는 어휘가 많다면? 계산량 $\uparrow \rightarrow$ 계산 병목 문제

1. 입력층의 원핫 표현 & 가중치 행렬 W_{in} 의 곱 계산
2. 은닉층 & 가중치 행렬 W_{out} 의 곱 및 소프트맥스 계층의 계산

✓ 1. Embedding 계층

- 1번 계산 병목 문제 해결
- 가중치 매개변수로부터 '단어 ID에 해당하는 행(벡터)'을 추출하는 계층

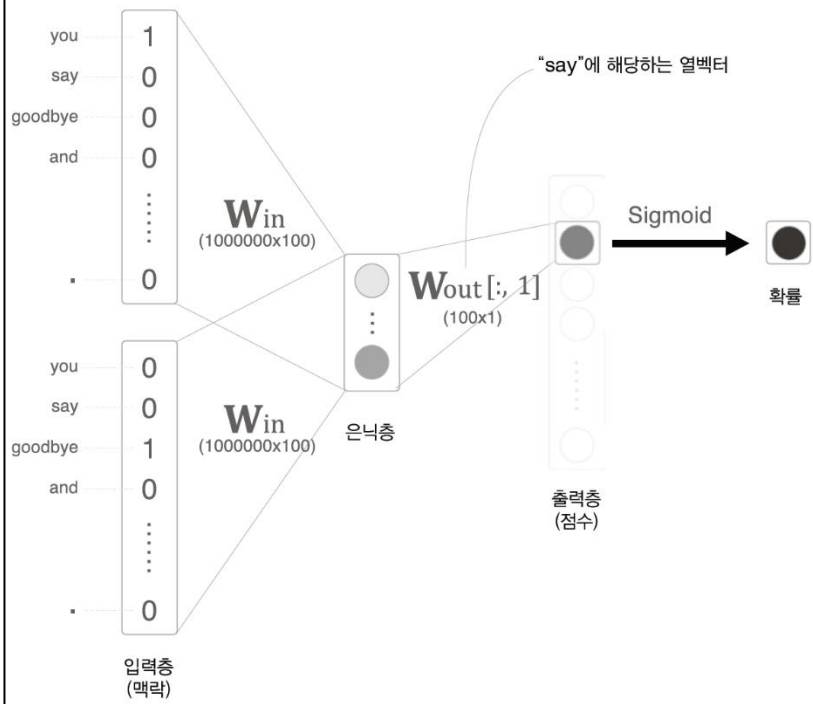
그림 4-4 Embedding 계층의 forward와 backward 처리(Embedding 계층은 Embed로 표기)



✓ 네거티브 샘플링

- 2번 계산 병목 문제 해결
- 핵심: 다중분류 \rightarrow 이중분류(Yes/No) 근사 \rightarrow 출력층 뉴런은 '하나'
- 은닉층 & 가중치 행렬 W_{out} 의 곱 = 타깃에 해당하는 열만 추출한 뒤 은닉층 뉴런과 내적
- 시그모이드 함수 & 교차 엔트로피 오차(손실함수)
- 긍정적 예 타깃으로 한 경우 손실 + 부정적 예 몇 개 샘플링 손실 = 최종 손실

그림 4-7 타깃 단어만의 점수를 구하는 신경망



✓ 부정적인 예 샘플링

- 모든 부정적인 예 이진 분류 학습 \rightarrow 계산량 \uparrow / 좋은 방법 \times
- 무작위로 부정적인 예 적게 샘플링 \rightarrow 좋은 방법 \times
- 말뭉치 \rightarrow 각 단어의 출현 횟수 \rightarrow '확률 분포'(0.75 제공) \rightarrow 단어 샘플링

✓ 유추 문제(비유 문제)

- 단어의 분산 표현 → 유추 문제를 벡터의 덧셈과 뺄셈으로 풀이
- word2vec 단어의 분산 표현 → 단어의 단순한 의미 + 문법적 패턴

✓ word2vec 단어의 분산 표현

- 전이 학습: 먼저 큰 말뭉치로 학습 완료 → 다른 작업에 활용(like. 텍스트 분류, 문서 클러스터링...)
- 단어를 고정 길이 벡터로 변환 → 문장도 고정 길이 벡터로 변환 → bag-of-words
- 자연어를 벡터로 변환 가능하다면? 머신러닝 기법(ex. SVM)에 적용 가능

✓ 단어 벡터 평가 방법

- 실제 애플리케이션과 분리하여 평가
- 평가 척도: (1) 유사성 / (2) 유추 문제
- (1) 유사성 평가: 사람이 작성한 단어 유사도 검증 세트 & word2vec에 의한 코사인 유사도 점수 비교, 상관성 체크
- (2) 유추 문제: 정답률로 측정 / ex) "king : queen = man : ?"
- 모델에 따라 정확도 다름 / 일반적으로 말뭉치 클수록 결과 좋음 / 단어 벡터 차원 수는 적당해야 함

✓ 피드포워드 신경망 = 흐름이 단방향인 신경망

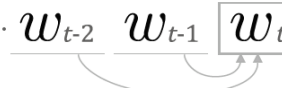
- 장점: 구성이 단순, 많은 문제에 응용 가능
- 단점: 시계열 데이터의 성질(패턴) 충분히 학습 X

✓ CBOW 모델의 학습

- 손실함수(말뭉치 전체의 손실함수의 총합)를 최소화하는 가중치 매개변수 찾기 → 맥락으로부터 타깃을 더 정확하게 추측
- 맥락 안의 단어 순서가 무시된다는 한계

그림 5-2 왼쪽 윈도우만 맥락으로 고려한다.

$w_1 \ w_2 \ \dots \ w_{t-2} \ w_{t-1} \ \boxed{w_t} \ w_{t+1} \ \dots \ w_{T-1} \ w_T$



- if. 맥락 = 왼쪽 두 단어 → CBOW 모델이 출력할 확률 $P(w_t | w_{t-2}, w_{t-1})$ / 손실함수 $L = -\log P(w_t | w_{t-2}, w_{t-1})$

✓ 언어 모델

- 단어 나열에 확률 부여 / 특정한 단어의 시퀀스에 대해, 그 시퀀스가 일어날 가능성의 정도를 확률로 평가
- if. CBOW 모델 → 언어 모델? 맥락의 크기 = 특정 값으로 한정하여 근사 → 특정 길이로 고정
- 한계: 고정된 특정 길이 밖의 단어 정보 무시 / 맥락 안의 단어 순서 무시

$$\prod_{t=1}^m P(w_t | w_1, \dots, w_{t-1})$$

✓ RNN(Recurrent Neural Network)

- for 순환, 닫힌 경로 or 순환하는 경로 필요 → 순환하면서 정보 끊임없이 갱신(동시에 과거 정보 기억)
- t = 시각 / x_t = 각 시각 별 시계열 데이터 / h_t = 은닉 상태 or 은닉 상태 벡터
- W_x = 입력 x 를 출력 h 로 변환하기 위한 가중치 / W_x = 1개의 RNN출력을 다음 시각의 출력으로 변환하기 위한 가중치
- $h_t = \tanh(h_{t-1}W_h + x_tW_x + b)$

그림 5-6 순환 경로를 포함하는 RNN 계층

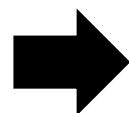
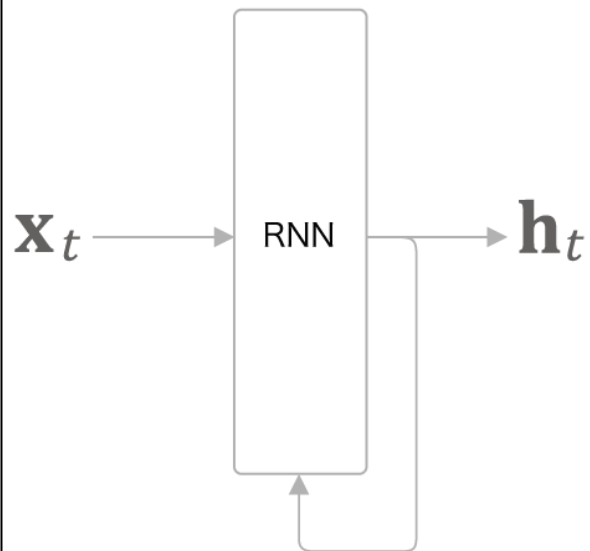
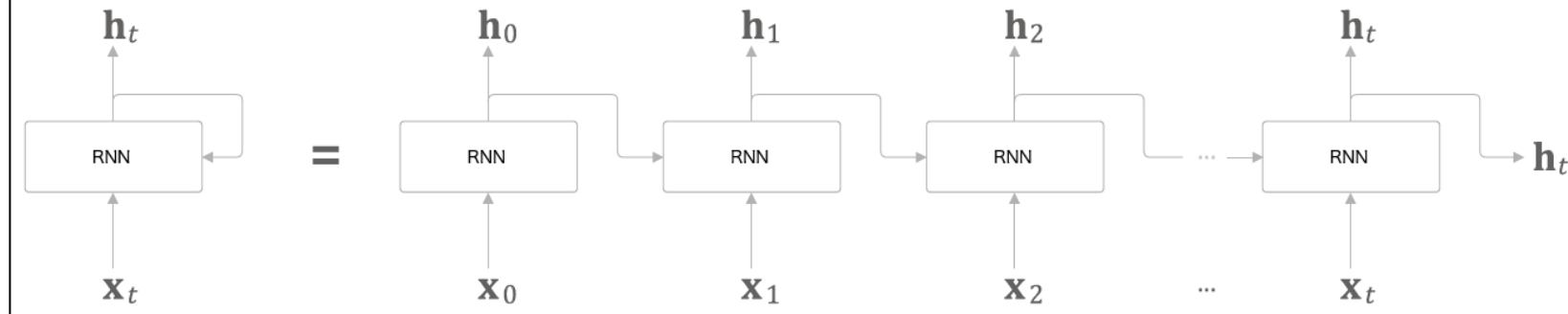


그림 5-8 RNN 계층의 순환 구조 펼쳐기



✓ BPTT

- 시간 방향으로 펼친 신경망의 오차역전파법 → RNN 학습 가능
- 한계: 긴 시계열 데이터 학습 시 → 소비하는 컴퓨팅 자원 ↑ → 기울기 불안정

그림 5-10 순환 구조를 펼친 RNN 계층에서의 오차역전파법

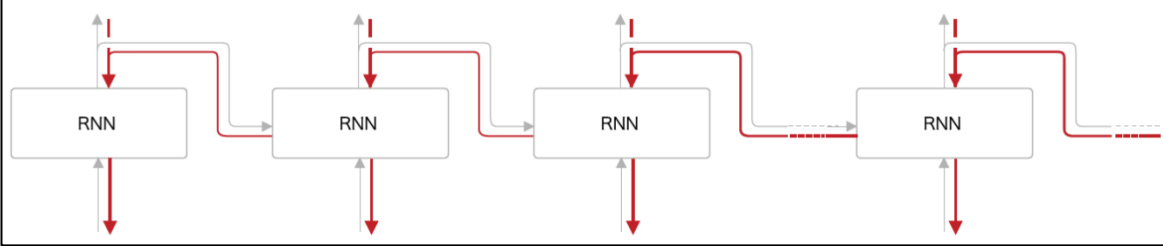
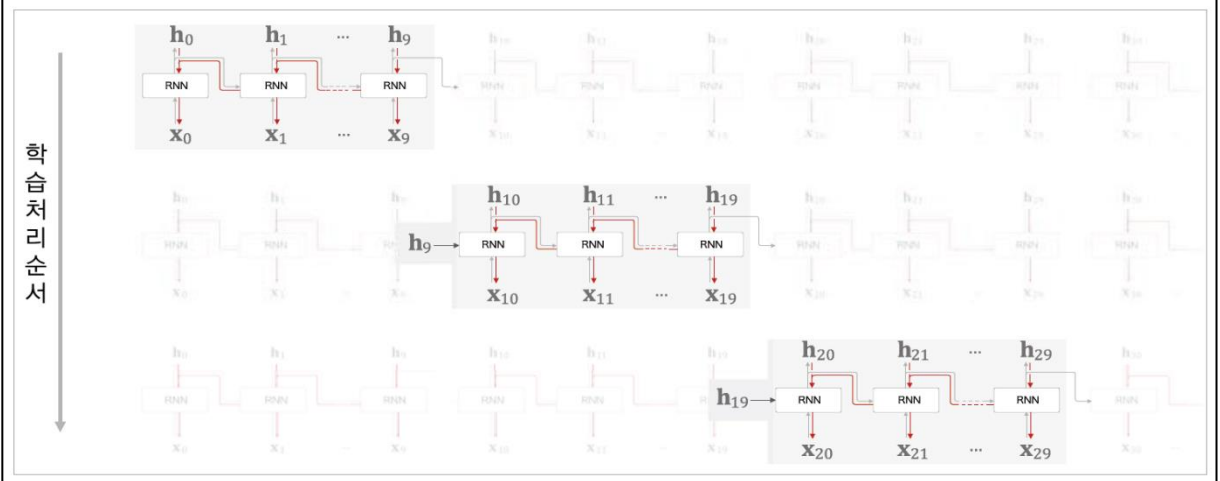


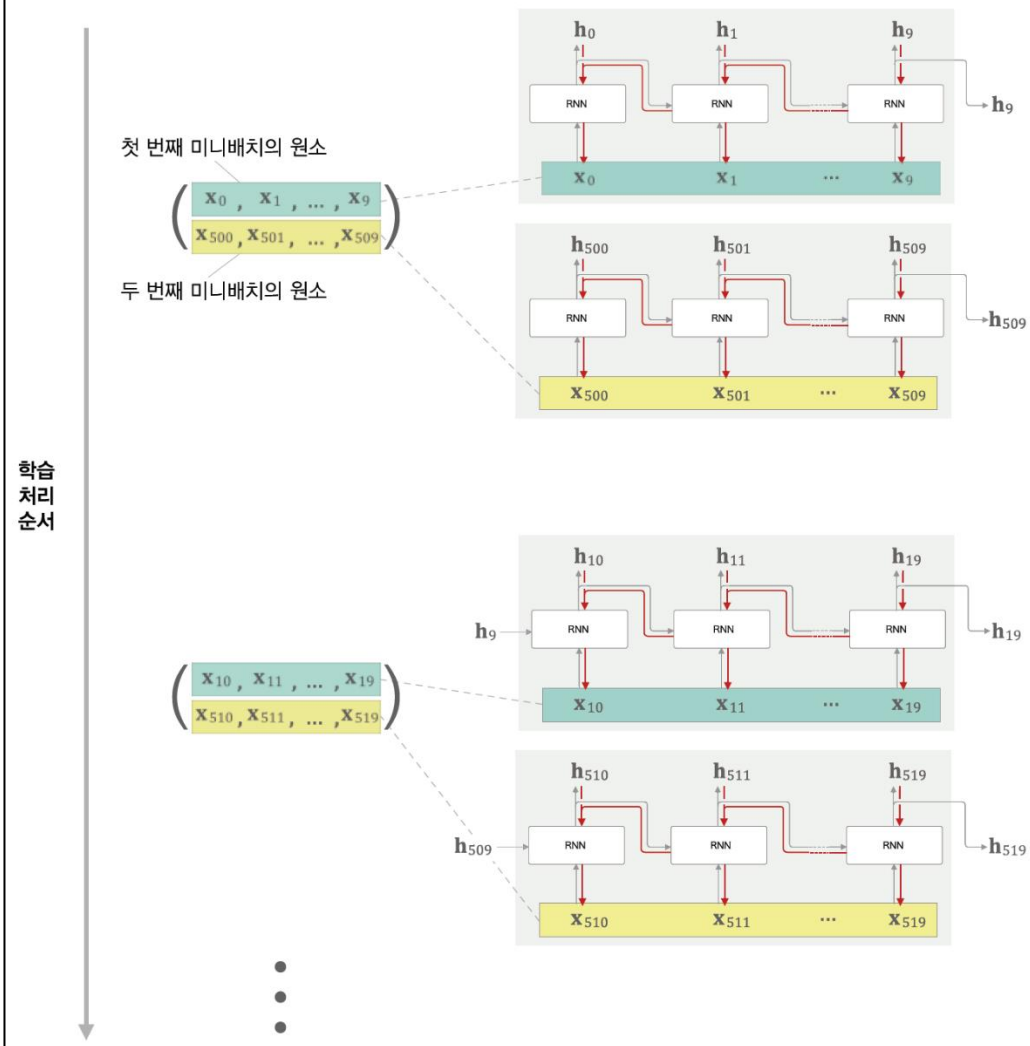
그림 5-14 Truncated BPTT의 데이터 처리 순서



✓ Truncated BPTT

- 시간 방향으로 너무 길어진 신경망 → 적당한 지점 자르기 → 작은 신경망 여러 개 → 잘라낸 작은 신경망(블록 단위)에서 독립적으로 오차역전파법
- 순전파의 연결 그대로 유지(데이터 순서대로 입력) & 역전파의 연결만 끊기
- 순전파 수행 시 마지막 은닉 상태 h_t 필요 BUT 역전파 수행 시 블록 단위로 독립적

그림 5-15 미니배치 학습 시 데이터를 제공하는 시작 위치를 각 미니배치(각 샘플)로 옮긴다.



✓ Truncated BPTT 미니배치 학습

- 각 미니배치의 시작 위치 '오프셋'으로 옮긴 후 → 순서대로 데이터 제공
- ex) 첫 번째 미니 배치: 처음부터 데이터 제공
- ex) 두 번째 미니 배치: 500번째부터 데이터 제공

✓ RNN 구현(Time RNN 계층)

- 길이가 T인 시계열 데이터 / 각 시각의 은닉 상태(h_t) T개 출력 → 하나의 계층으로 구현
- Time RNN 계층 내 한 단계 작업 = RNN 계층 / T개 단계분의 작업 = Time RNN 계층
- RNN 계층의 은닉 상태 h(인스턴스 변수) → '인계'받는 용도 → stateful=True or False

그림 5-17 Time RNN 계층: 순환 구조를 펼친 후의 계층들을 하나의 계층으로 간주한다.

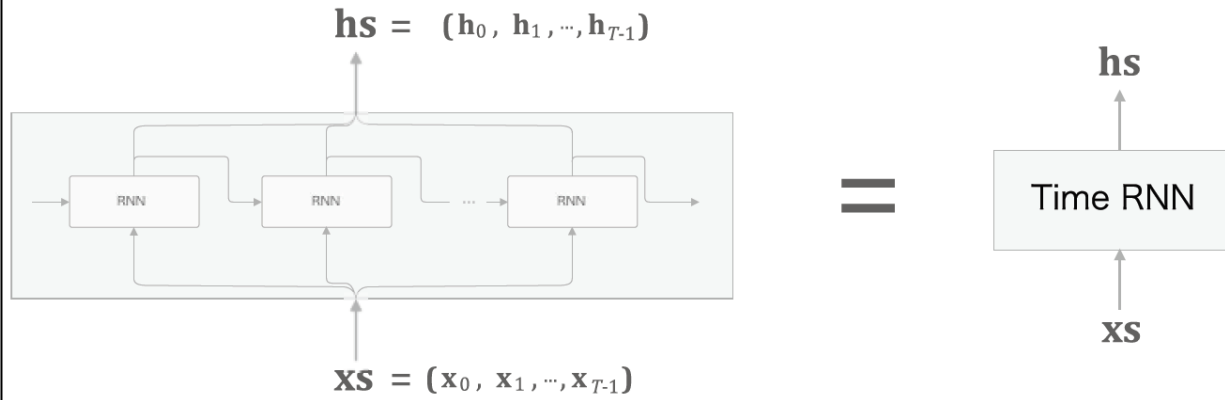


그림 5-22 Time RNN 계층은 은닉 상태를 인스턴스 변수 h로 보관한다. 그러면 은닉 상태를 다음 블록에 인계할 수 있다.

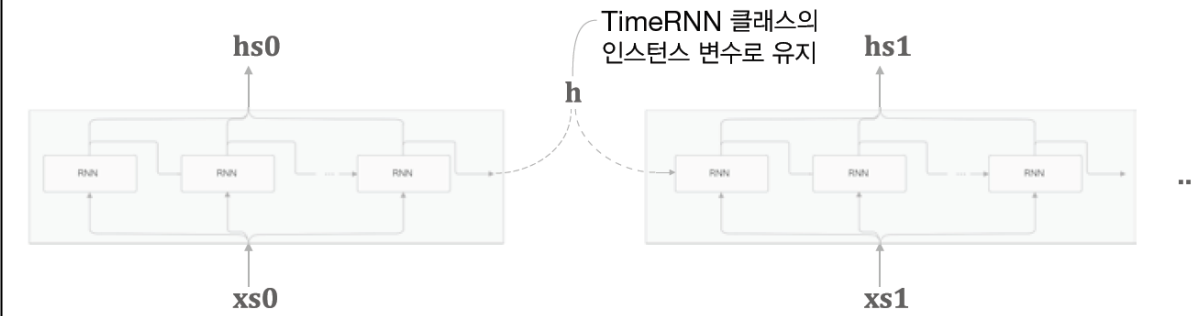
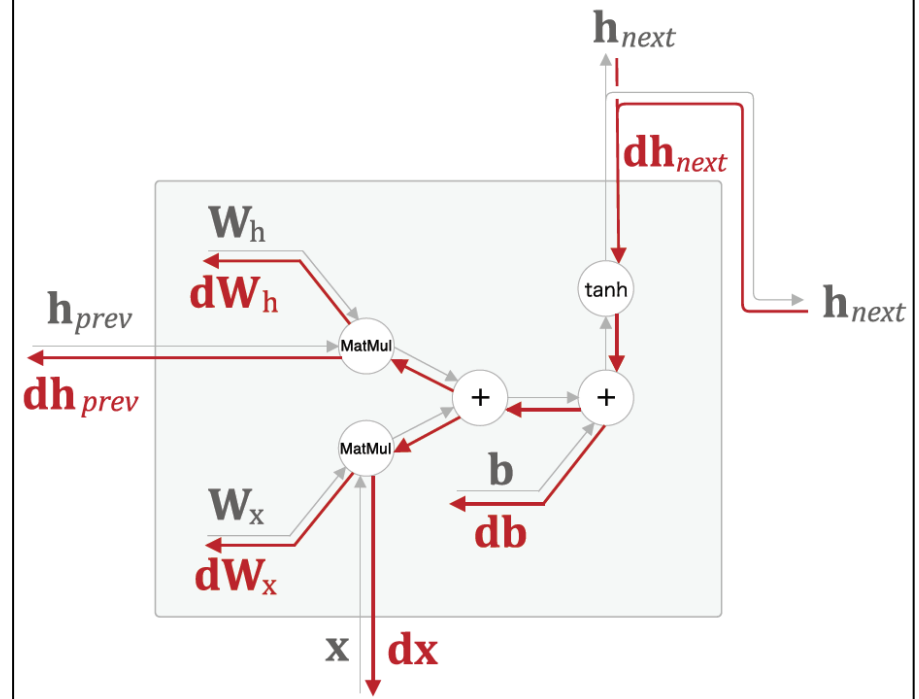


그림 5-20 RNN 계층의 계산 그래프(역전파 포함)



✓ RNNLM(RNN 사용한 언어 모델)

- Embedding → RNN → Affine → Softmax with Loss
- Embedding, Affine, Softmax with Loss 또한 Time 계층
- T개의 Softmax with Loss 계층의 최종 손실

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

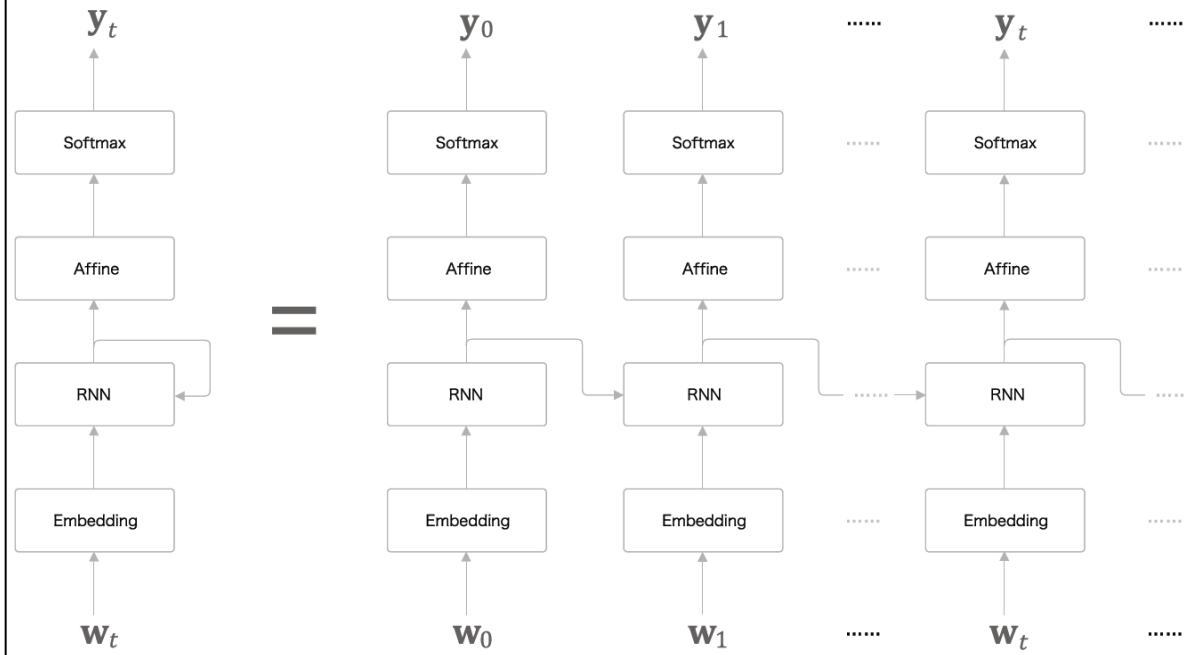
- 구현 시, RNN, Affine 계층에서 Xavier 초깃값 이용

✓ 언어 모델 평가

- 퍼플렉시티: 언어 모델 예측 성능 평가 척도 / 확률의 역수
- 퍼플렉시티 ↓ → GOOD!
- 퍼플렉시티 = 분기 수(다음에 취할 수 있는 선택사항의 수)

- 입력 데이터 여러 개? $L = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$ / perplexity = e^L

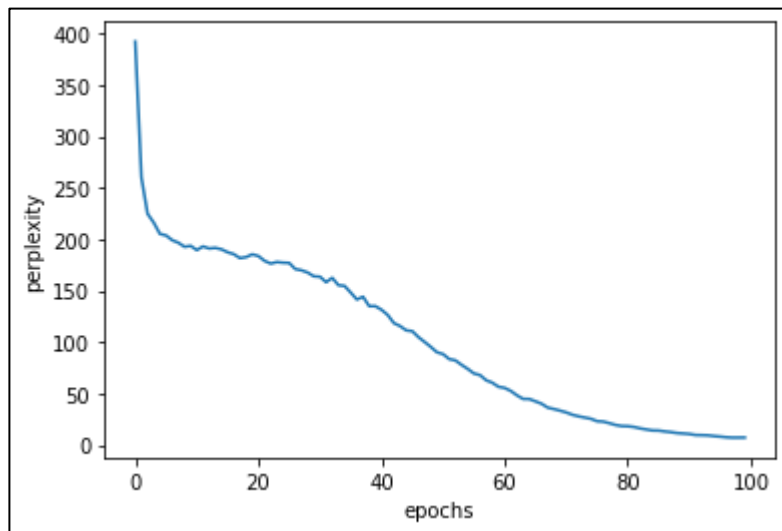
그림 5-25 RNNLM의 신경망(왼쪽이 펼치기 전, 오른쪽은 펼친 후)



✓ RNNLM 학습 결과

- PTB 데이터셋의 처음 1,000개 단어만 이용
- 데이터 순차적으로 제공 / 각각의 미니배치에서 데이터 읽는 시작 위치 조정 using 오프셋
- 에폭마다 손실의 평균 구하기 → 퍼플렉시티 계산
- 300을 넘는 퍼플렉시티 → 7까지 감소
- 한계: 큰 말뭉치에는 전혀 대응 불가

	에폭	87		퍼플렉서티	13.13
	에폭	88		퍼플렉서티	12.48
	에폭	89		퍼플렉서티	11.57
	에폭	90		퍼플렉서티	11.15
	에폭	91		퍼플렉서티	10.67
	에폭	92		퍼플렉서티	9.66
	에폭	93		퍼플렉서티	9.53
	에폭	94		퍼플렉서티	9.33
	에폭	95		퍼플렉서티	8.63
	에폭	96		퍼플렉서티	8.11
	에폭	97		퍼플렉서티	7.46
	에폭	98		퍼플렉서티	7.11
	에폭	99		퍼플렉서티	7.14
	에폭	100		퍼플렉서티	7.18

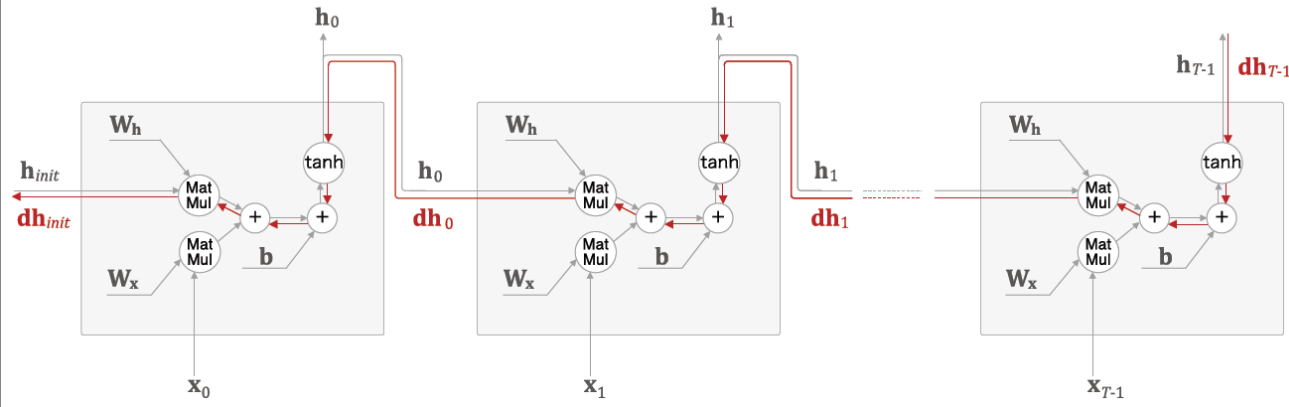


- ✓ 어려웠던 점
- ✓ 궁금한 점
 - RNNLM 모델 구현 시 'Xavier 초기값' 을 사용하는 이유

✓ RNN의 문제점

- 시계열 데이터의 장기 의존 관계 학습 hard ← 이유: 기울기 소실 or 기울기 폭발
- RNN 계층 과거 방향으로 '의미 있는 기울기'를 전달 → 시간 방향의 의존 관계 학습 but 기울기 소실 or 기울기 폭발 발생
- 'tanh', '+', 'MatMul'의 역전파로 기울기 전해짐 → '+' 역전파는 그대로 / 'tanh' 역전파는 지날수록 기울기 감소 → ReLU로 교체

그림 6-5 RNN 계층에서 시간 방향으로의 기울기 전파

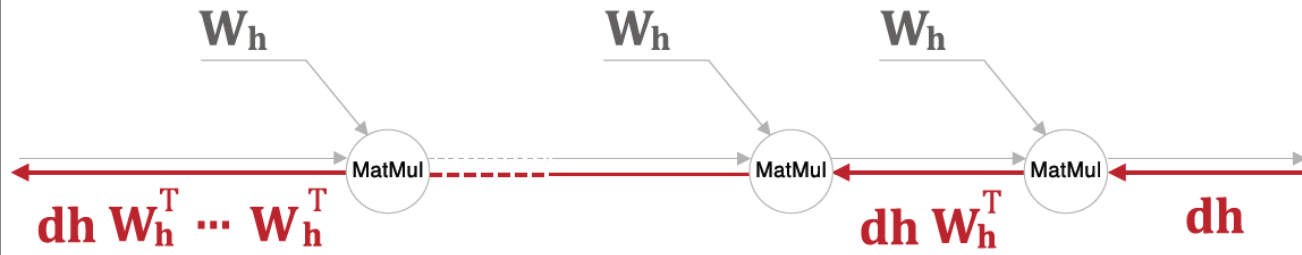


- MatMul의 경우: 가중치 행렬 W_h 의 초깃값에 따라 → 기울기 소실 or 기울기 폭발

✓ 기울기 폭발 대책: 기울기 클리핑

- $if \|\hat{g}\| \geq threshold \rightarrow \hat{g} = \frac{threshold}{\|\hat{g}\|} \hat{g}$
- \hat{g} = 모든 매개변수 기울기 / threshold = 문턱값
- 기울기의 L2노름이 문턱값 초과 → 기울기 수정

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

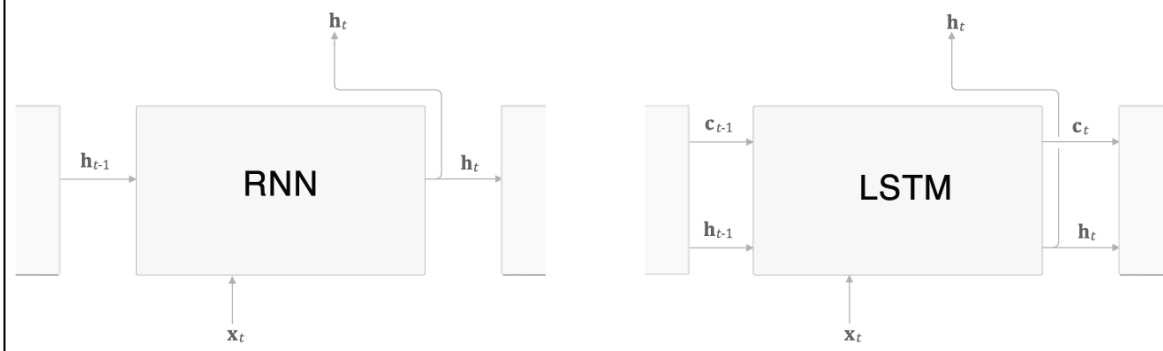
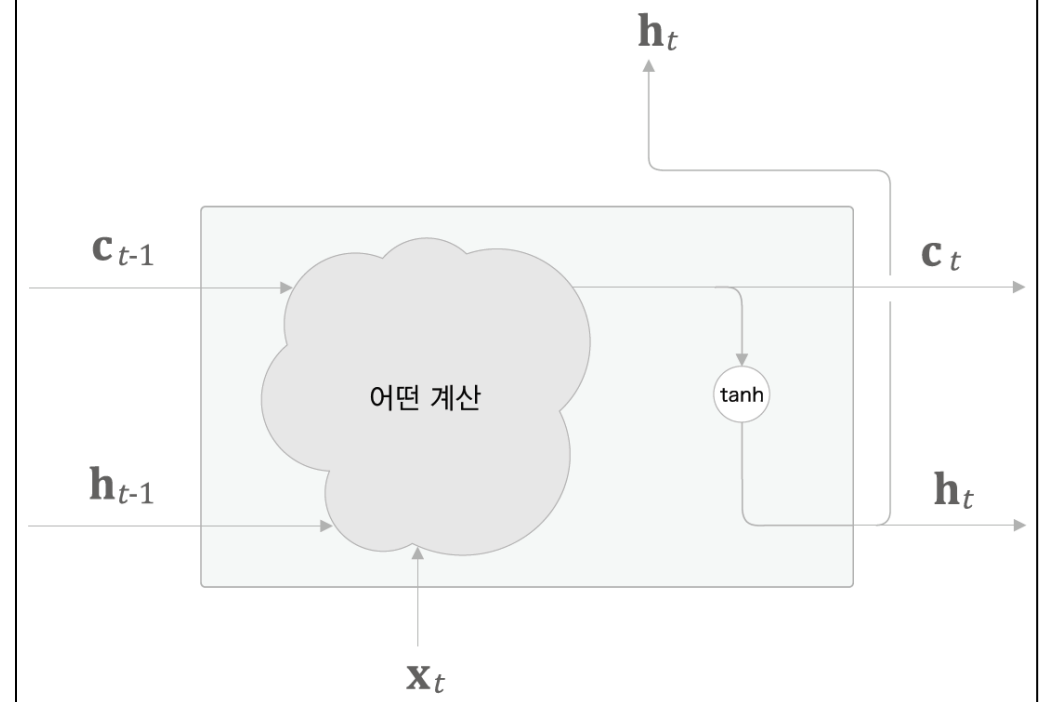


✓ 기울기 소실 대책

- 게이트가 추가된 RNN: LSTM & GRU → 기울기 소실 일으키지 않음

- LSTM 인터페이스: 기억 셀(c) 존재 → LSTM 전용 기억 매커니즘 / 특징: 데이터 자기 자신으로만(LSTM 계층 내) 주고받음
- 현재의 기억셀 $c_t \leftarrow$ 3개의 입력(c_{t-1}, h_{t-1}, x_t) '어떤 계산' 수행하여 구함 / 갱신된 $c_t \rightarrow$ 은닉 상태 h_t 계산 ($h_t = \tanh(c_t)$)

그림 6-11 RNN 계층과 LSTM 계층 비교

그림 6-12 기억 셀 c_t 를 바탕으로 은닉 상태 h_t 를 계산하는 LSTM 계층

✓ 게이트

- 데이터 흐름을 제어 / '열기/닫기' & '열림 상태' 조절 가능
- output 게이트: $\tanh(c_t)$ 의 각 원소가 다음 시각의 은닉 상태에 얼마나 중요한가 / 시그모이드 함수 사용 0.0~1.0 사이 실수 출력

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

- forget 게이트: c_{t-1} 의 기억 중 불필요한 기억 잊게 해줌

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

- 새로운 기억 셀: forget 게이트 거치면서 이전 시각의 기억 셀로부터 잊어야 할 기억 삭제 → 새로 기억해야 할 정보 추가(tanh 노드)

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

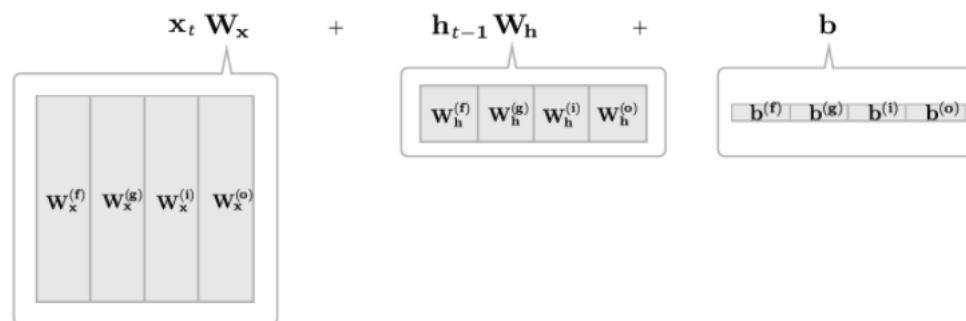
- input 게이트: g의 각 원소가 새로 추가되는 정보로서의 가치가 얼마나 큰지 판단 / 가중된 정보 새로 추가

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

- 위의 네 식을 한 번의 아핀(Affine) 변환으로 계산 가능 →

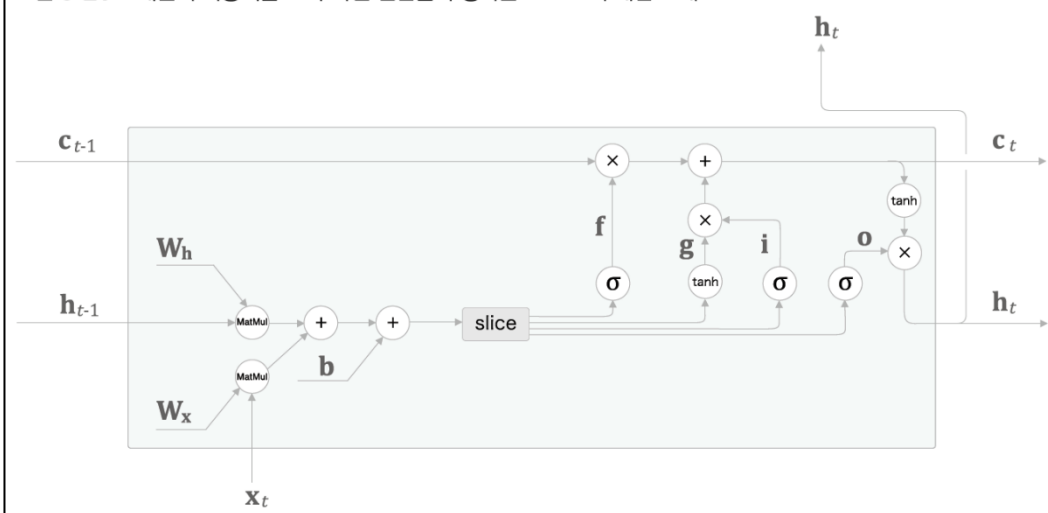
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{g} \odot \mathbf{i}$$

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



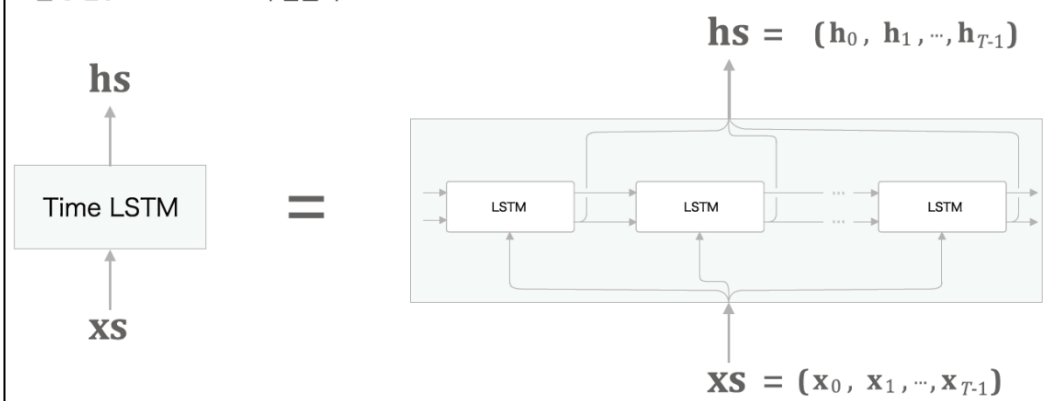
- LSTM의 계산 그래프

그림 6-21 4개분의 가중치를 모아 아핀 변환을 수행하는 LSTM의 계산 그래프



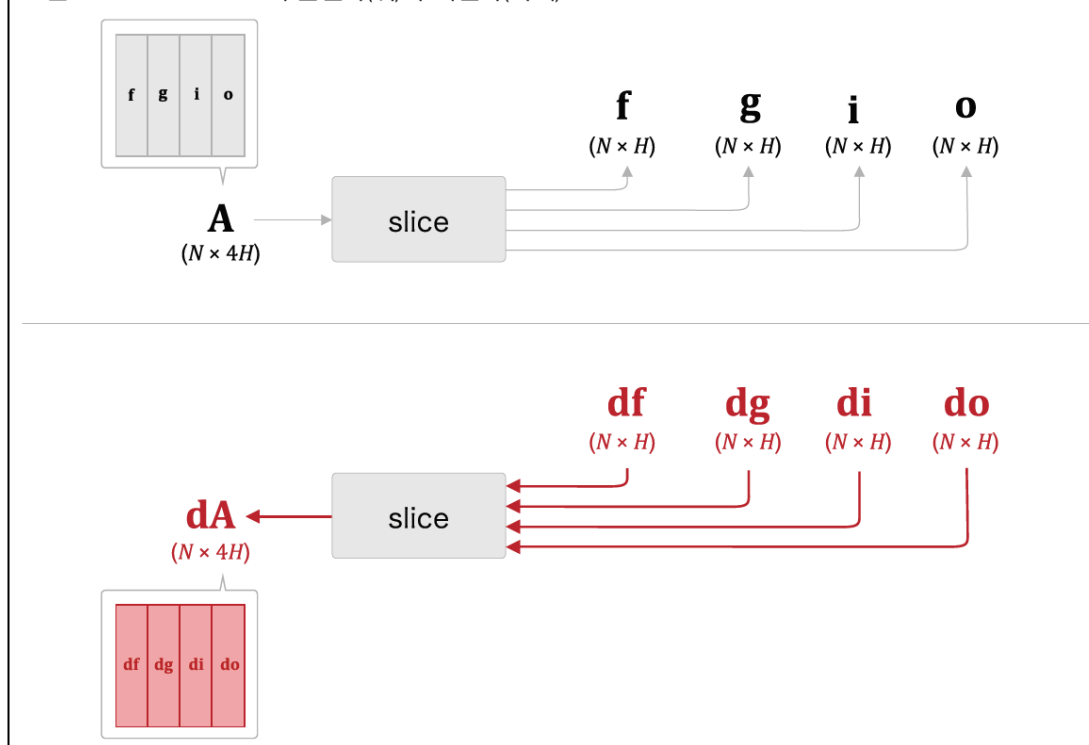
- Time LSTM 구현 \rightarrow T개분 시계열 데이터 처리

그림 6-24 Time LSTM의 입출력



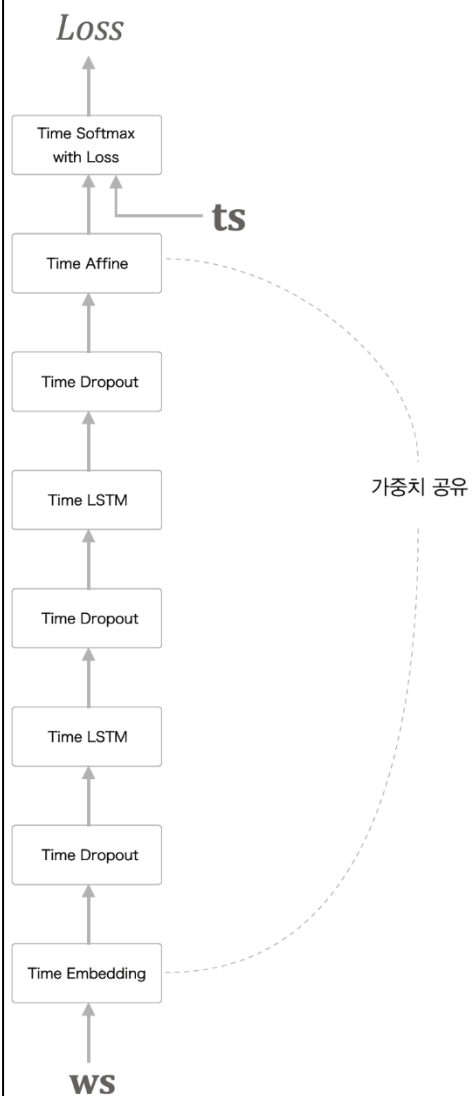
- slice 노드의 순전파&역전파

그림 6-23 slice 노드의 순전파(위)와 역전파(아래)



✓ RNNLM 개선 포인트 3가지

그림 6-36 BetterRnnlm 클래스의 신경망 구성



- LSTM 계층 다층화: 모델의 정확도 향상 / 2~4층 정도가 적당 / 복잡한 의존 관계(패턴) 학습
- 드롭아웃: 과적합 억제 / 깊이 방향(상하 방향)으로 삽입
(변형 드롭아웃: 시간 방향 정규화 → 일반적으로 시계열 방향 드롭아웃은 시간이 흐름에 따라 정보 사라짐)
- 가중치 공유: Embedding 계층의 가중치 & Affine 계층의 가중치 공유 → 학습할 매개변수 ↓, 정확도 ↑

✓ 어려웠던 점

- 특잇값과 기울기 소실/기울기 폭발과의 관계
- 변형 드롭아웃에 대해 (특히 마스크 개념)

✓ 궁금한 점

✓ 언어 모델 → 문장생성

- 언어 모델: 주어진 단어들 → 다음에 출현하는 단어의 확률분포 출력
- (1) 확률이 가장 높은 단어 / (2) 확률적으로 선택(샘플링 단어)
- (2)번 선택(확률적 알고리즘_매번 다른 문장)
- <eos> 같은 종결 기호 나올 때까지 샘플링 반복

• LSTM 언어 모델

그림 7-1 앞 장에서 구현한 언어 모델: 오른쪽은 시계열 데이터를 한꺼번에 처리하는 Time 계층을 사용했고, 왼쪽은 같은 구성을 펼친 모습

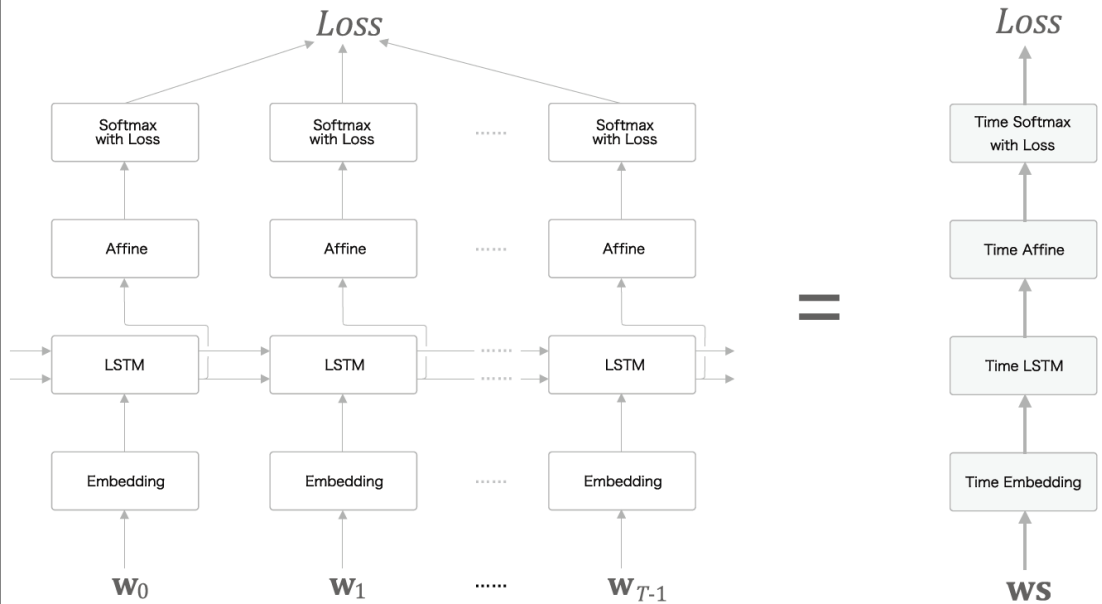
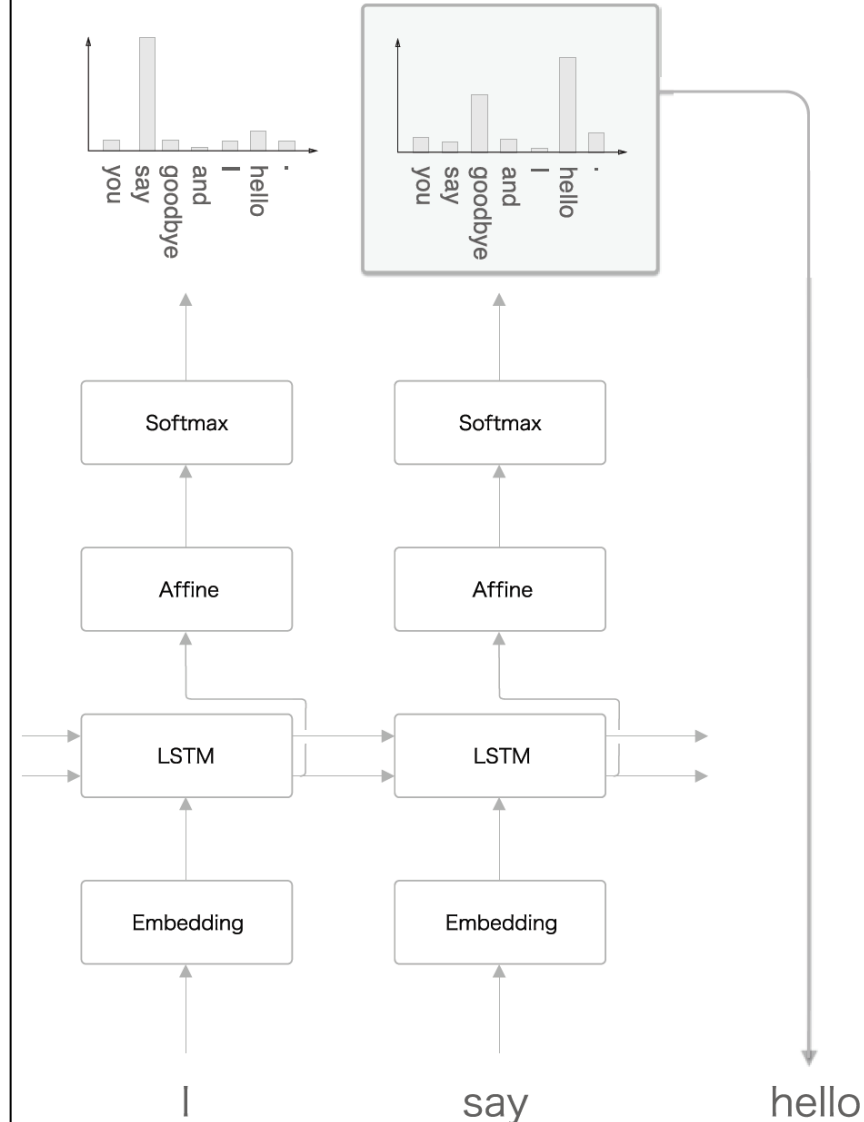


그림 7-4 확률분포 출력과 샘플링을 반복한다.



✓ seq2seq

- 시계열 데이터 → 다른 시계열 데이터로 변환 (2개의 RNN 이용)
- Encoder-Decoder 모델

✓ Encoder 계층

- RNN 이용: 시계열 데이터 → 은닉 상태 벡터(h) 변환
- h : 입력 문장 번역 시 필요한 정보 인코딩 / 고정 길이 벡터 / Encoder, Decoder 가교 역할
- 순전파: 인코딩 정보 → h → Decoder / 역전파: 기울기 → h → Encoder

✓ Decoder 계층: LSTM 계층이 벡터 h 를 입력 받음

그림 7-5 Encoder와 Decoder가 번역을 수행하는 예

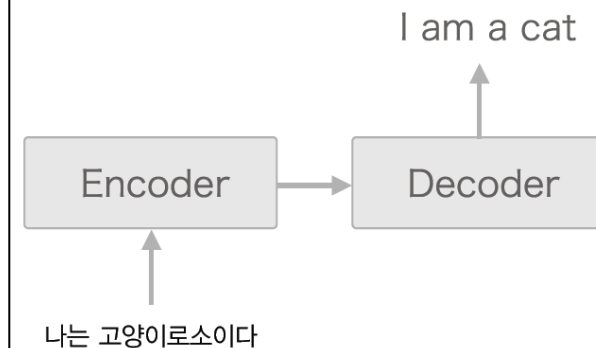
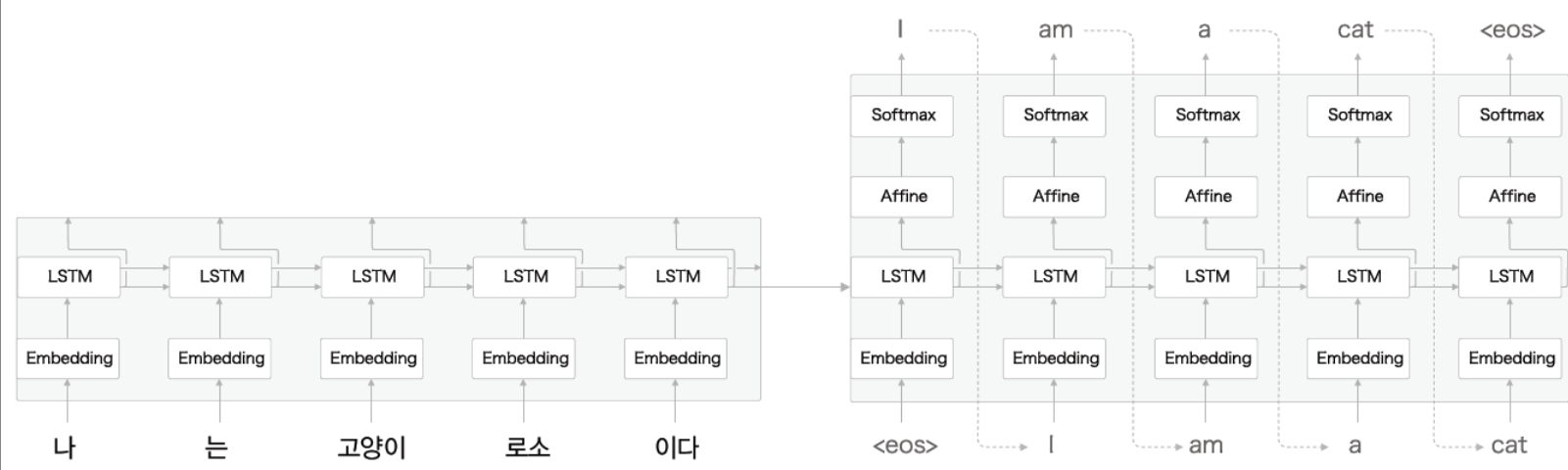


그림 7-9 seq2seq의 전체 계층 구성



✓ 장난감 문제(for. 머신러닝 평가 문제)

- 더하기 문제 ← 단어가 아닌 문자 단위로 분할
- 가변 길이 시계열 데이터 → for. 미니배치 학습 → 패딩(padding) 사용(공백문자)
- 0~999 사이 숫자 2개만 더하기 / 입력 최대 문자 수: 7 / 출력 최대 문자 수: 4
- for. 질문과 정답 구분 → 출력 앞 구분자(_) 추가 → 출력 데이터: 5 문자

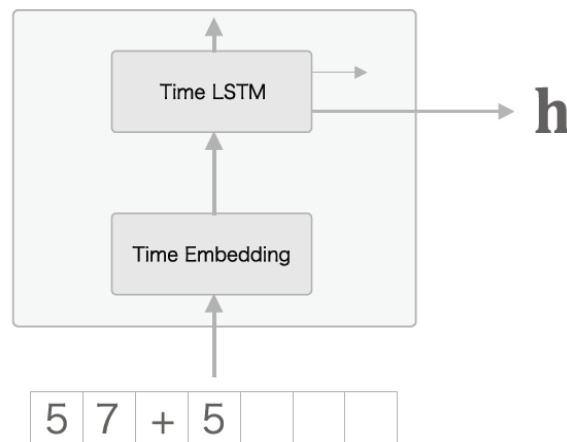
그림 7-11 미니배치 학습을 위해 '공백 문자'로 패딩을 수행하여 입력·출력 데이터의 크기를 통일한다.

입력							출력				
5	7	+	5				-	6	2		
6	2	8	+	5	2	1	-	1	1	4	9
2	2	0	+	8			-	2	2	8	

- Encoder: LSTM 위쪽 출력 폐기
- Decoder: for. 결정적 답 → argmax 노드 활용 / Time Softmax with Loss 계층 앞까지만
- Seq2seq: Encoder&Decoder 연결 / 손실 계산

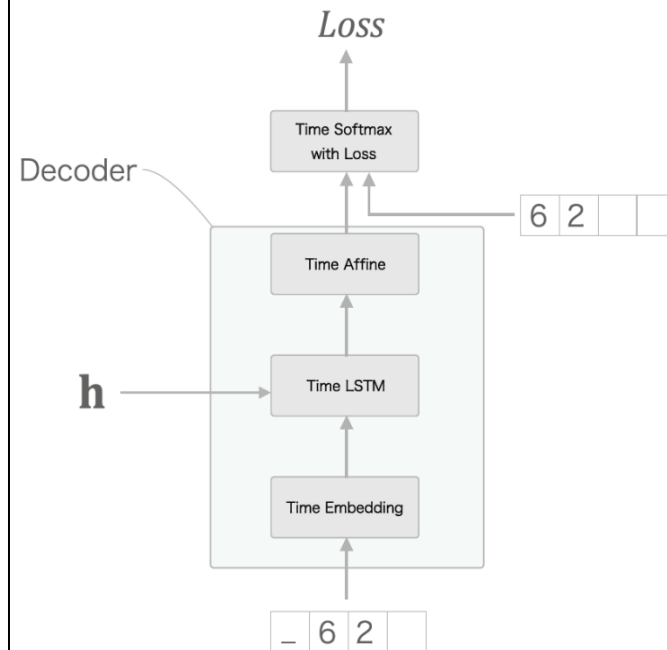
• Encoder 계층

그림 7-15 Encoder를 Time 계층으로 구현한다.



• Decoder 계층

그림 7-19 Decoder 클래스의 구성



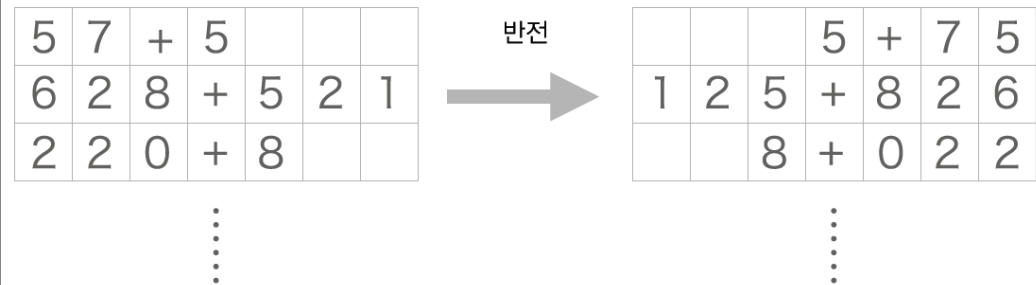
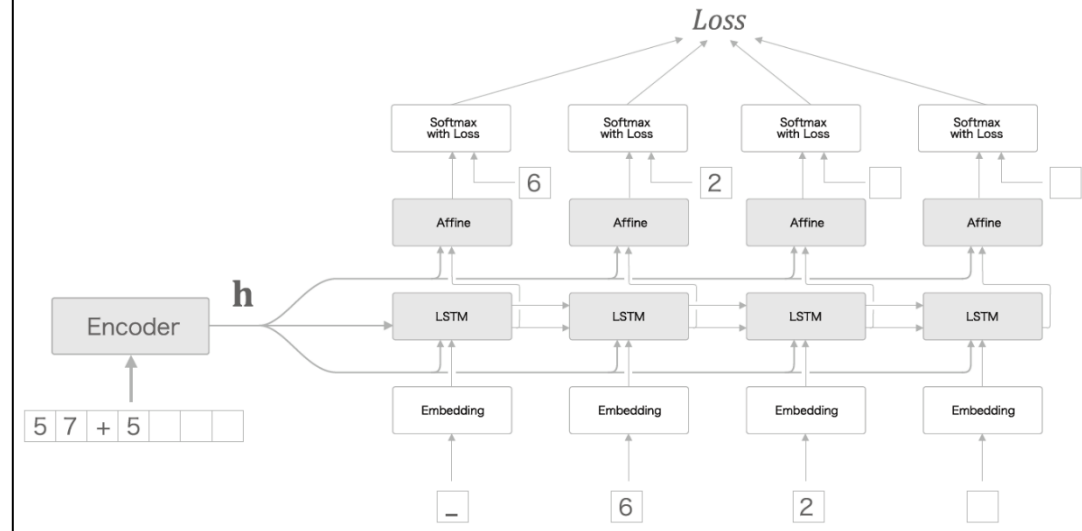
✓ Seq2seq 학습 흐름

- (1) 학습 데이터에서 미니배치 선택 / (2) 미니배치로부터 기울기 계산 / (3) 기울기 사용하여 매개변수 갱신
- 평가 척도: 정답률(에폭마다 테스트 데이터의 문제 중 몇 개 풀고 올바르게 답했는지 채점) / eval_seq2seq
- 아직까지는 정답률 높지 않음

✓ Seq2seq 개선

- (1) 입력 데이터 반전(Reverse) → 학습 진행 빨라짐, 최종 정확도 ↑ (:: maybe. 기울기 전파 원활)
- (2) 엿보기(Peeky): Encoder의 출력 h 를 Decoder의 다른 계층에게도 전달 (Affine 계층)

그림 7-23 입력 데이터를 반전시키는 예

그림 7-26 개선 후: Encoder의 출력 h 를 모든 시각의 LSTM 계층과 Affine 계층에 전해준다.

• 개선 전

Q 761+292
T 1053
X 1049

Q 830+597
T 1427
X 1418

Q 26+838
T 864
X 869

Q 143+93
T 236
X 242

검증 정확도 8.620%

• Reverse

Q 292+167
T 1053
X 1052

Q 795+038
T 1427
X 1426

Q 838+62
T 864
O 864

Q 39+341
T 236
O 236

검증 정확도 54.280%

• Peeky

Q 761+292
T 1053
O 1053

Q 830+597
T 1427
X 1430

Q 26+838
T 864
O 864

Q 143+93
T 236
O 236

검증 정확도 81.620%

• Reverse & Peeky

Q 292+167
T 1053
O 1053

Q 795+038
T 1427
O 1427

Q 838+62
T 864
O 864

Q 39+341
T 236
O 236

검증 정확도 99.300%

✓ Seq2seq 이용 어플리케이션 (한 시계열 데이터 → 다른 시계열 데이터)

- 기계 번역 / 자동 요약 / 질의응답 / 메일 자동 응답
- 챗봇: 사람과 컴퓨터가 텍스트로 대화 나누는 프로그램 / 상대의 말 → 자신의 말 변환
- 알고리즘 학습: 소스 코드 또한 시계열 데이터 / NTM(Neural Turing Machine)
- 이미지 캡셔닝: 이미지 → 문장 변환 / Encoder가 합성곱 신경망(CNN) / im2txt

✓ 어려웠던 점

- Seq2seq의 패딩 전용 처리(마스크 기능) p.305

✓ 궁금한 점

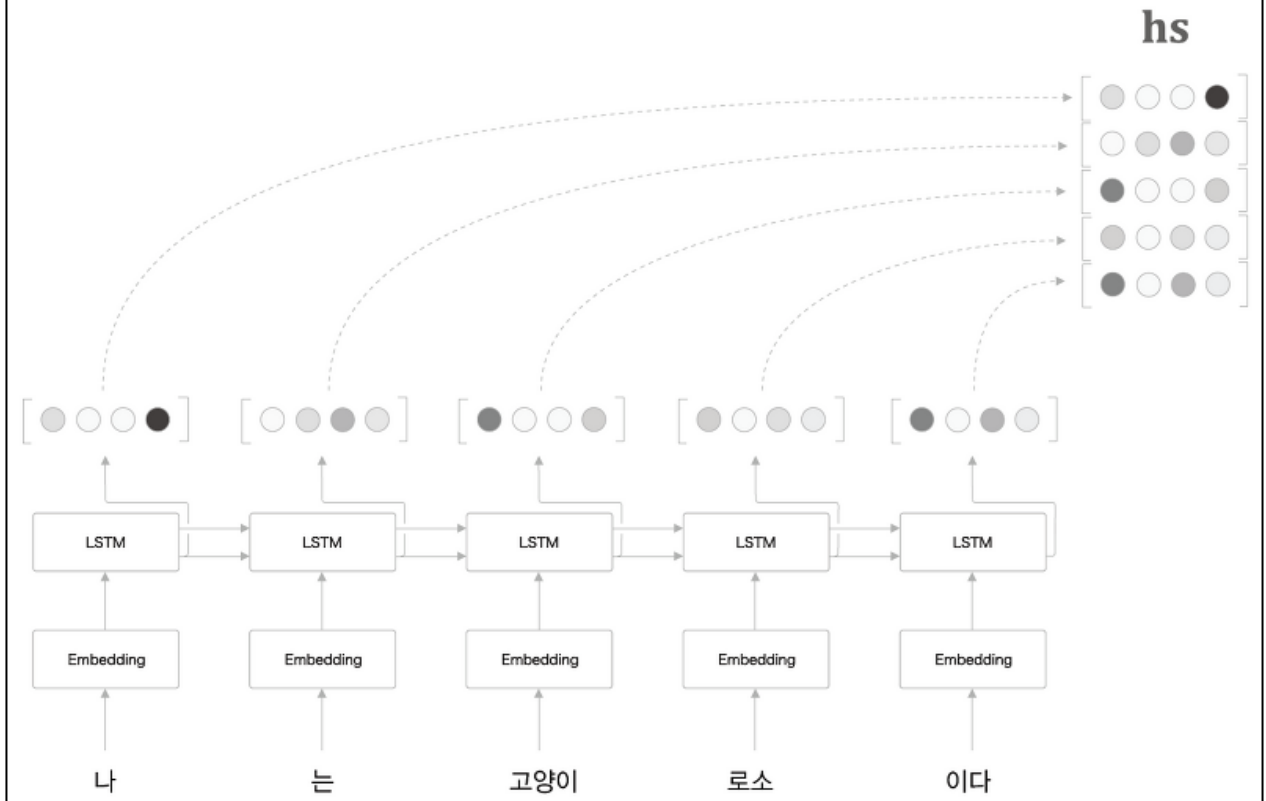
- 입력 데이터 반전 만으로도 성능이 향상되는 이유가 아직도 밝혀지지 않는지
- lm2txt에 대해 (아직도 계속 발전되고 있는지)

- 어텐션 매커니즘 → seq2seq 인간처럼 필요한 정보에만 주목 / 기존의 seq2seq의 문제점 해결
- seq2seq의 문제점: Encoder의 출력 = '고정 길이의 벡터' → 필요한 정보가 벡터에 다 담기지 못함

✓ Encoder 개선

- Encoder 출력의 길이 ~ 입력 문장의 길이에 따라 바꾸기
- 시각별 LSTM 계층의 은닉 상태 벡터를 모두 이용
- hs 행렬 = 각 단어에 해당하는 벡터들의 집합

그림 8-29 LSTM 계층에 의한 hs 출력



✓ Decoder 개선

- 개선 Point! Decoder가 encoder의 LSTM 계층의 마지막 은닉상태만을 이용 X, hs 를 전부 활용할 수 있도록!
- Using 어텐션 구조 = 입력과 출력의 여러 단어 중 어떤 단어끼리 서로 관련되어 있는지 주목
- 각 시각에서 Decoder에 입력된 단어와 대응관계인 단어의 벡터 hs 에서 '선택 작업'(단어들의 얼라인먼트 추출) → 미분 불가
→ 하나를 선택 X → 모든 것을 선택 / 대신. 가중치(각 단어의 중요도)를 별도로 계산
- a (가중치): 0.0~1.0 사이의 스칼라, 모든 원소의 총합 = 1 → 가중치 큰 단어 성분 많이 포함 → 그 단어 선택
- c (맥락 벡터): 가중치 a & 각 단어의 벡터 hs 가중합

- Attention Weight 계층: for. 가중치 a 계산 / 은닉 상태 벡터 h 와 각 단어 벡터 hs 의 내적(유사도) = s (점수)
→ $a = \text{softmax}(s)$
- Weight Sum 계층: for. 맥락 벡터 c 계산 / a 와 hs 의 가중합
- Attention Weight + Weight Sum = Attention 계층 → LSTM & Affine 계층 사이에 삽입

그림 8-16 맥락 벡터를 계산하는 계산 그래프

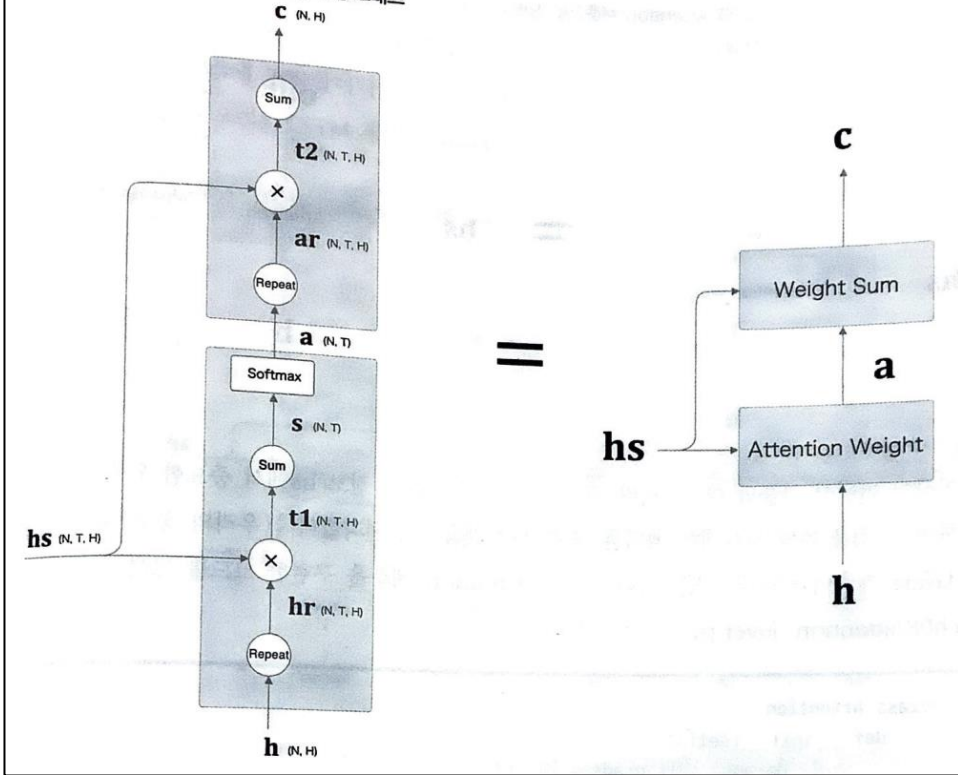
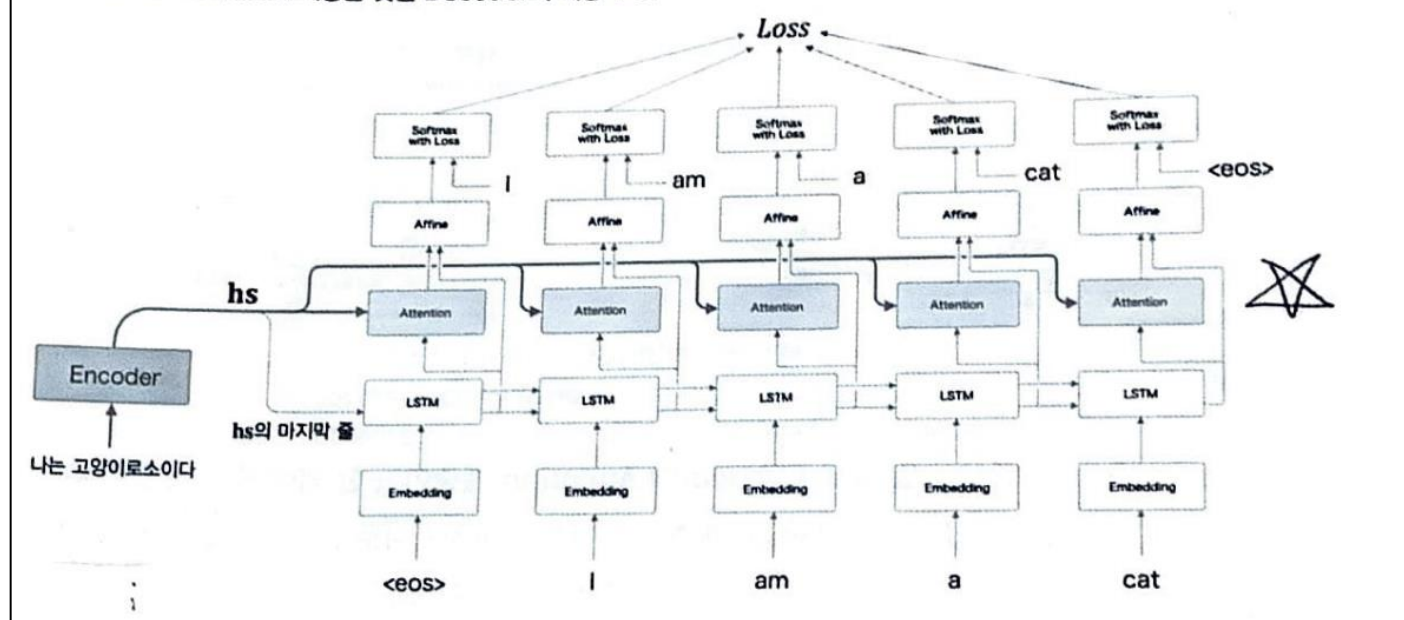


그림 8-18 Attention 계층을 갖춘 Decoder의 계층 구성



- 시계열 방향으로 펼쳐진 다수의 Attention 계층 → Time Attention 계층으로 모아서 구현

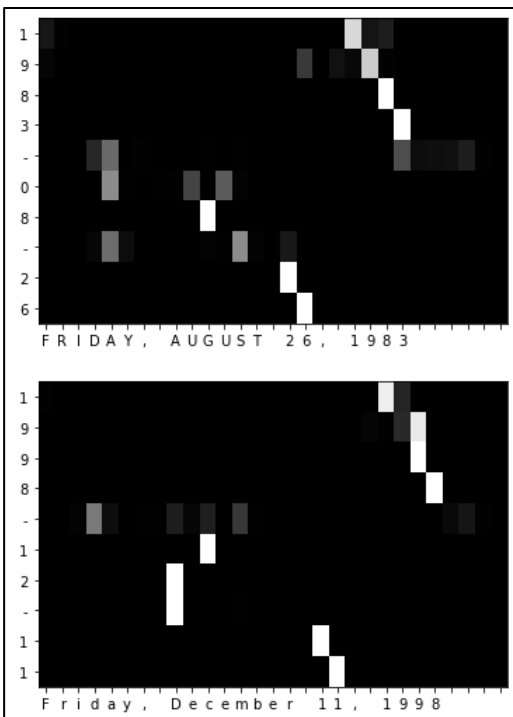
✓ 어텐션 평가

- '날짜 형식 변환' 문제
- 다양한 변형 존재(복잡) / 알기 쉬운 대응 관계(년·월·일)

september 27, 1994	-->	1994-09-27
JUN 17, 2013	-->	2013-06-17
2/10/93	-->	1993-02-10

- 모델: AttentionSeq2seq / Reverse 적용 / 에폭마다 테스트 데이터 사용 → 정답률 측정
- 3 에폭 정답률이 99.9 / 8 에폭부터 정답률 100 → 성능 Good!
- 결론: 최종 정확도 면 – 어텐션 & Peeky 비슷 / BUT. 현실의 시계열 데이터 길고 복잡 → 어텐션 – 학습 속도, 정확도 모두 유리

```
Q 8/23/08
T 2008-08-23
O 2008-08-23
---
Q 8/30/07
T 2007-08-30
O 2007-08-30
---
Q 10/28/13
T 2013-10-28
O 2013-10-28
---
Q sunday, november 6, 2016
T 2016-11-06
O 2016-11-06
---
val acc 100.000%
```



- Attention 가중치 시각화: seq2seq이 필요한 정보에 주의를 기울임
- Ex) '년·월·일' & 입력데이터 가중치: 비슷하게 대응 → August & 08

✓ 양방향 RNN

- 타깃 단어의 주변 정보를 균형있게 담고싶을 때 활용 → 양방향 LSTM
- 기존의 LSTM 계층에, '역방향'으로 처리하는 LSTM 추가
- 각 시각에서 두 LSTM 계층의 은닉 상태를 '연결'시키는 벡터 = 최종 은닉 벡터
- 구현 방법: 2개의 LSTM 계층 사용 → 하나는 그대로, 하나는 입력문을 '오른쪽에서 왼쪽으로' → 연결

✓ Attention 계층 사용법: Attention 계층 이용하는 위치 정해져 있지 않음 / 여러 변형 존재

- ✓ seq2seq 심층화: RNN 층 깊게 쌓기 / Encoder & Decoder 같은 층 수 일반적
→ Decoder의 LSTM 계층의 은닉 상태 → Attention 계층 입력 → 맥락 벡터(Attention 계층 출력) → Decoder의 여러 계층 전파
- ✓ skip 연결: 층을 깊게 할 때 중요한 기법 / 계층을 넘어 선을 연결(계층을 건너뛰는 연결) → 층이 깊어져도 기울기 소실 or 폭발 X

✓ 어텐션 응용

- 구글 신경망 기계 번역(GNMT): 신경망 기계 번역 / LSTM 계층의 다층화, 양방향 LSTM, skip 연결 / 다수의 GPU
- 트랜스포머: RNN의 단점인 병렬 처리 불가능을 막기 위해 RNN을 없애고 어텐션만을 사용 / 셀프어텐션 / 계산량 ↓, 병렬 계산 혜택
- 뉴럴 튜링 머신(NTM): 외부 메모리를 통한 확장 ← 어텐션 ~ 메모리 조작 유사 / 콘텐츠 기반 어텐션, 위치 기반 어텐션

✓ 어려웠던 점

- Skip 연결의 개념
- 셀프 어텐션이 하나의 시계열 내에서 원소 간 대응 관계를 구해낸다는 점이 어색
- 뉴럴 튜링 머신 기술 전반적으로 어려움

✓ 궁금한 점

- 선택하는 작업이 왜 미분할 수 없는지? 모든 것을 선택하면 왜 미분 가능한지
- 자연어 처리(NLP) 분야에서는 가장 핫한 기술이 어텐션인가?

감사합니다