



Natural Language Processing

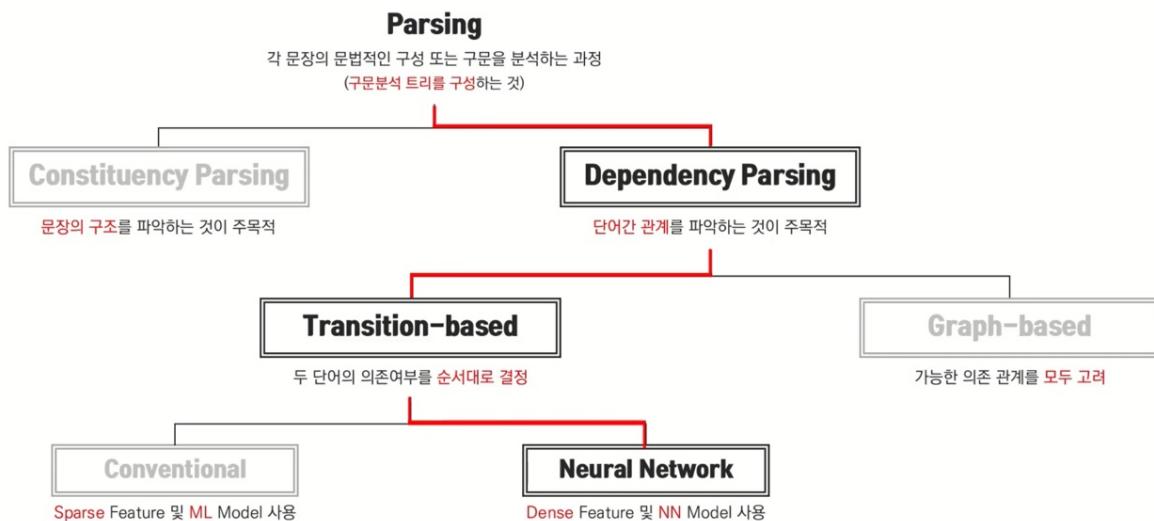
CS224N : lec 5 & 6

index

1. Neural dependency parsing
2. A bit more about neural networks
3. Language modeling + RNNs

Review

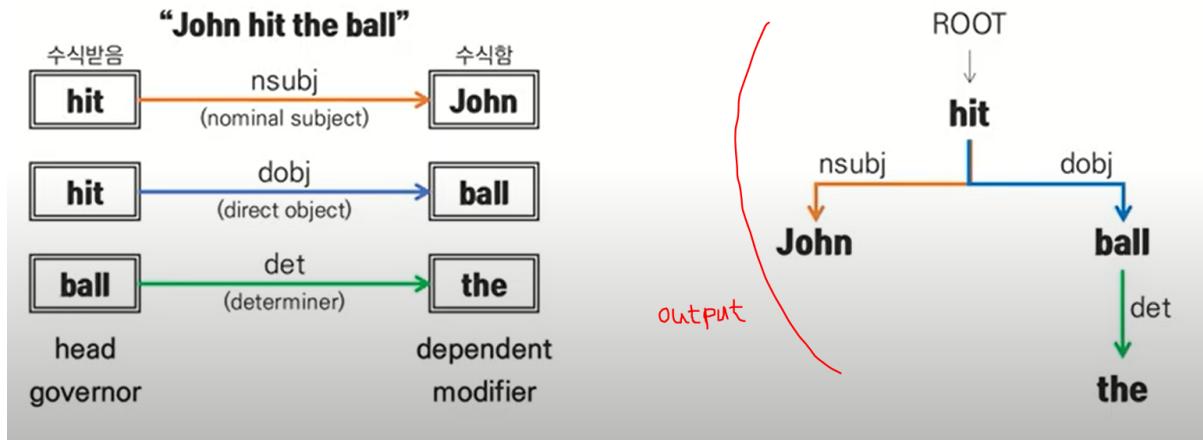
- Parsing의 결과물은 트리 형태.



[Dependency Parsing (의존 구문분석)]

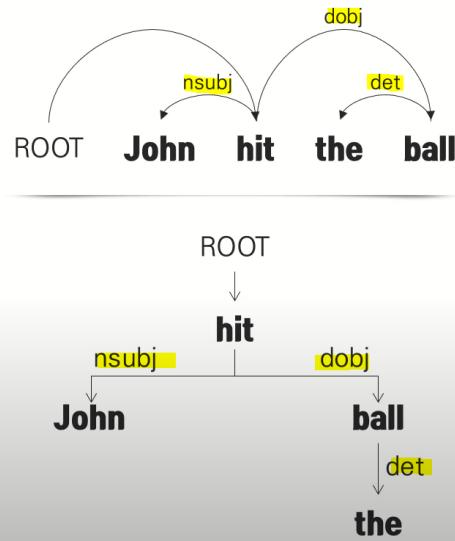
각 단어간 의존 또는 수식 관계를 파악

한국어와 같이 자유 어순을 가지거나 문장성분이 생략 가능한 언어에서 선호
하지만 영어권에서도 점차 Dependency Parsing에 대한 관심이 증가하고 있음



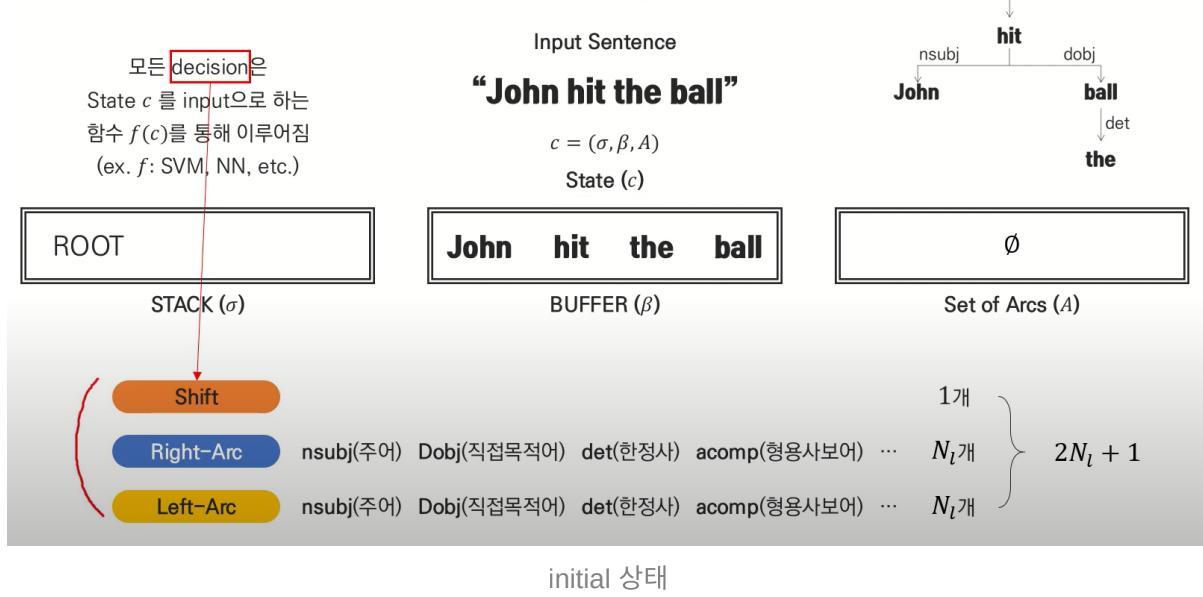
[Grammar and Structure]

- Dependency Structure는 두 가지 형태로 표현 가능
- 화살표는 head에서 dependent로 향함
- 화살표 위 label은 단어간 문법적 관계(dependency)를 의미
- 어떠한 단어의 수식도 받지 않는 단어는 가상의 node인 ROOT의 dependent로 만들어 모든 단어가 최소한 1개 node의 dependent가 되도록 함
- 화살표는 순환하지 않으며(if $A \rightarrow B$, then $B \not\rightarrow A$), 덕분에 parsing 결과물을 tree 형태로 표현할 수 있음



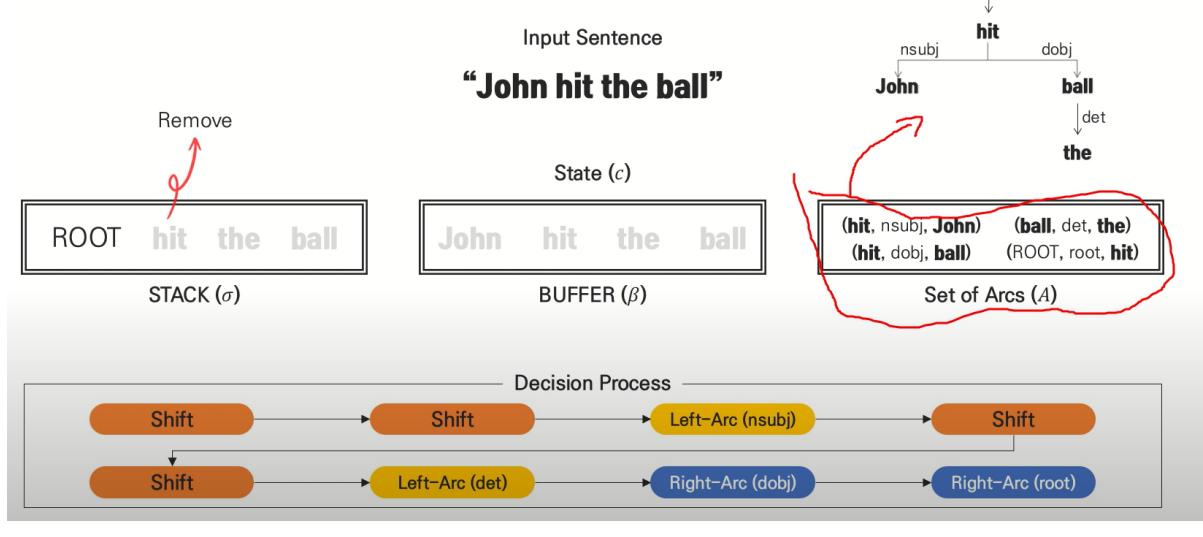
[Transition-based Parsing]

Nivre (2003)



[Transition-based Parsing]

Nivre (2003)



Conventional Feature Representation 방식의 문제점

- sparse
- incomplete
- expensive computation

(parsing 소요시간 중 95% 이상을 feature 연산이 차지.)

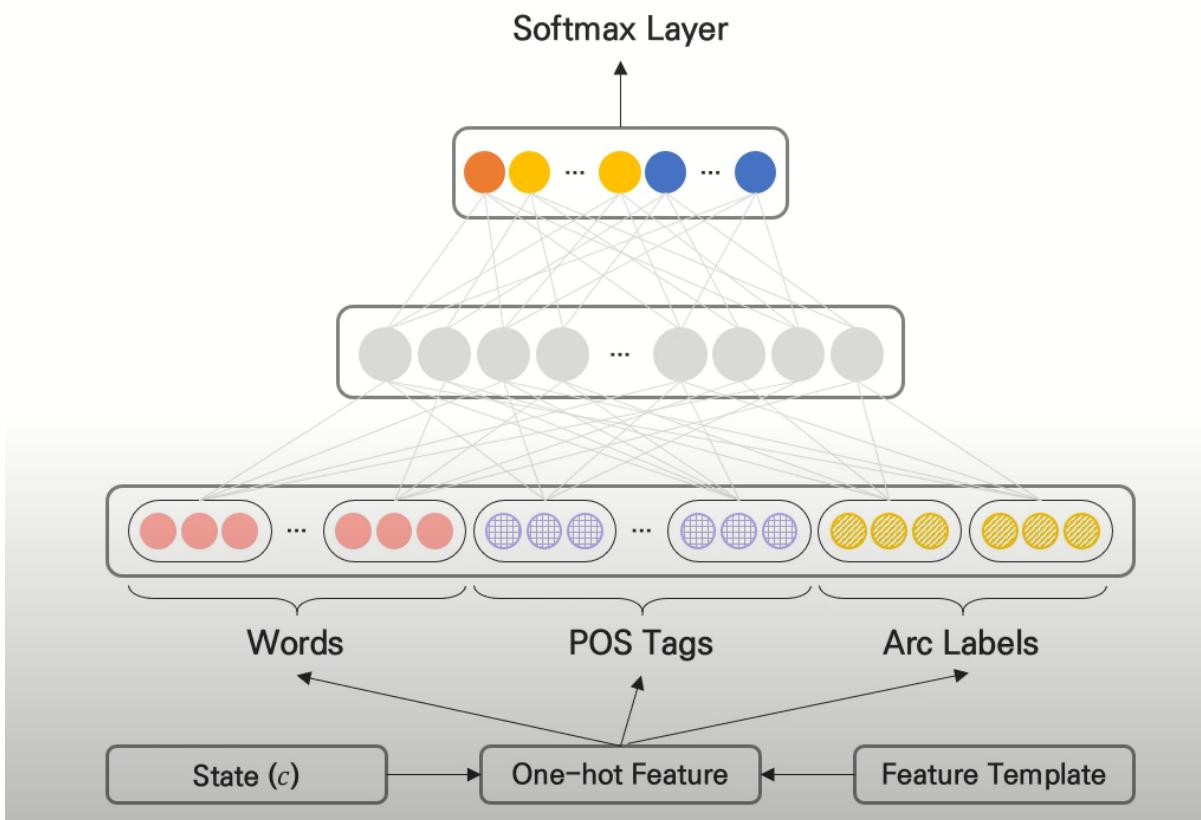
↓ 보완

1. Neural dependency parser

- STACK과 BUFFER configuration에 neural network를 직접 사용한 neural dependency parsing
- 기본적인 feed forward network(순방향 신경망 구조)와 비슷.
- feature representation하는 과정과 hidden layer에 쓰이는 활성화 함수가 다름.

[Neural Dependency Parser]

Chen and Manning (2014)



1-1. 예시

- stack : 후입선출 (나중에 들어온 것이 먼저 나감)
- buffer : 선입선출 (먼저 들어온 것이 먼저 나감)

Example State (c)

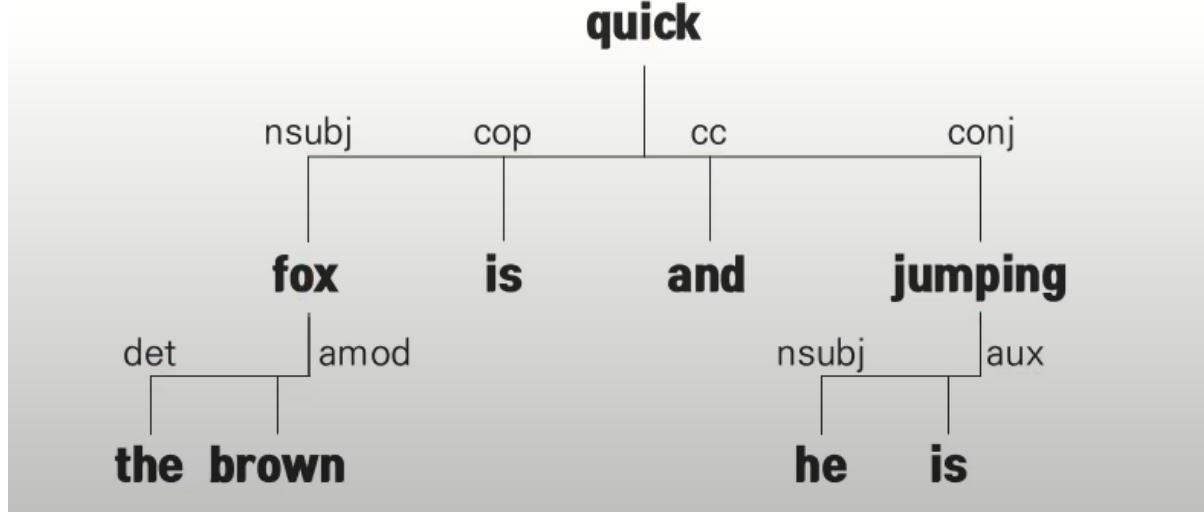
Input Sentence

**“The brown fox is quick and
he is jumping over the lazy dog”**

STACK (σ)

BUFFER (β)

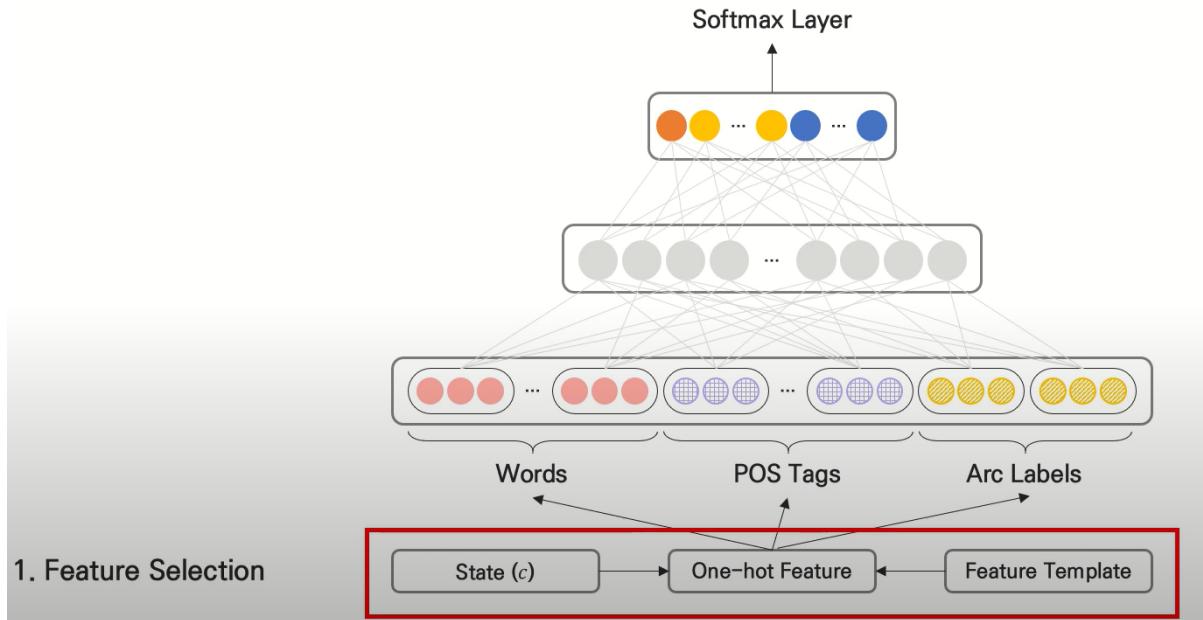
ROOT **quick over** **the lazy dog**



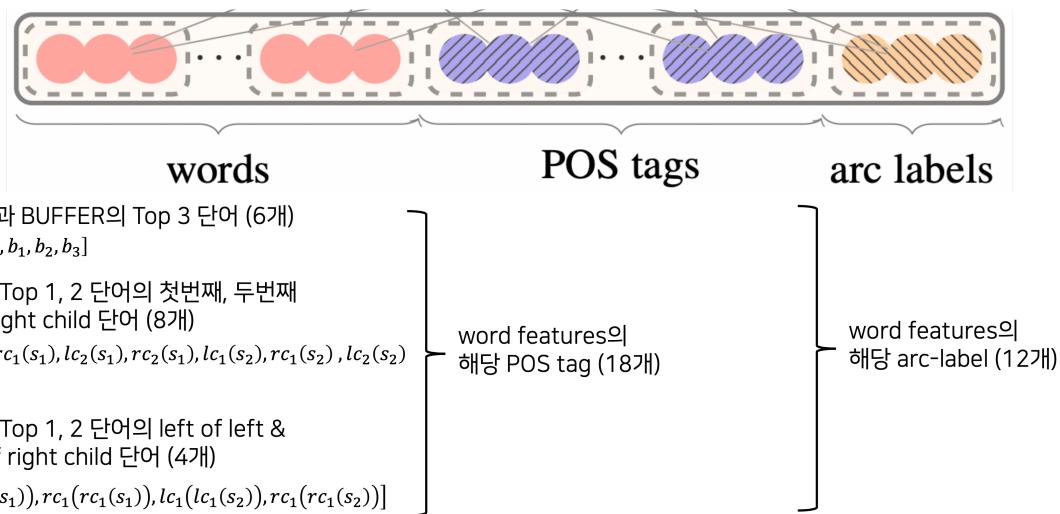
과정1. feature selection

[Neural Dependency Parser]

Chen and Manning (2014)



Part-Of-Speech tagging(POS tagging) : 문장 내 단어들의 품사를 식별하여 태그를 붙여주는 것을 말함



→ 결과

Feature Template

- STACK과 BUFFER의 top 3 단어 (6개) $s_1, s_2, s_3, b_1, b_2, b_3$
 $[\text{over}, \text{quick}, \text{ROOT}, \text{the}, \text{lazy}, \text{dog}]$
- STACK top 1, 2 단어의 1st and 2nd left and right child 단어 (8개)
 $lc_1(s_1), rc_1(s_1), lc_2(s_1), rc_2(s_1) \quad lc_1(s_2), rc_1(s_2), lc_2(s_2), rc_2(s_2)$
 $[\text{Null}, \text{Null}, \text{Null}, \text{Null}] \quad [\text{fox}, \text{jumping}, \text{is}, \text{and}]$
- STACK top 1, 2 단어의 (left of left) and (right of right) child 단어 (4개)
 $lc_1(lc_1(s_1)), rc_1(rc_1(s_1)), lc_1(lc_1(s_2)), rc_1(rc_1(s_2))$
 $[\text{Null}, \text{Null}] \quad [\text{the}, \text{Null}]$
- 선택된 word feature에 해당하는 POS Tag (18개)
 $[\text{IN(전치사)}, \text{JJ(형용사)}, \text{ROOT}, \text{DT(한정사)}, \dots, \text{Null}, \text{DT(한정사)}, \text{Null}]$
- STACK과 BUFFER의 6개 단어를 제외하고 선택된 word에 달린 arc-label (12개)
 $[\text{Null}, \text{Null}, \dots, \text{nsubj(주어)}, \text{conj(접속사)}, \text{cop(연결사)}, \text{cc(등위)}, \dots, \text{Null}]$

one-hot representation

One-hot Representation

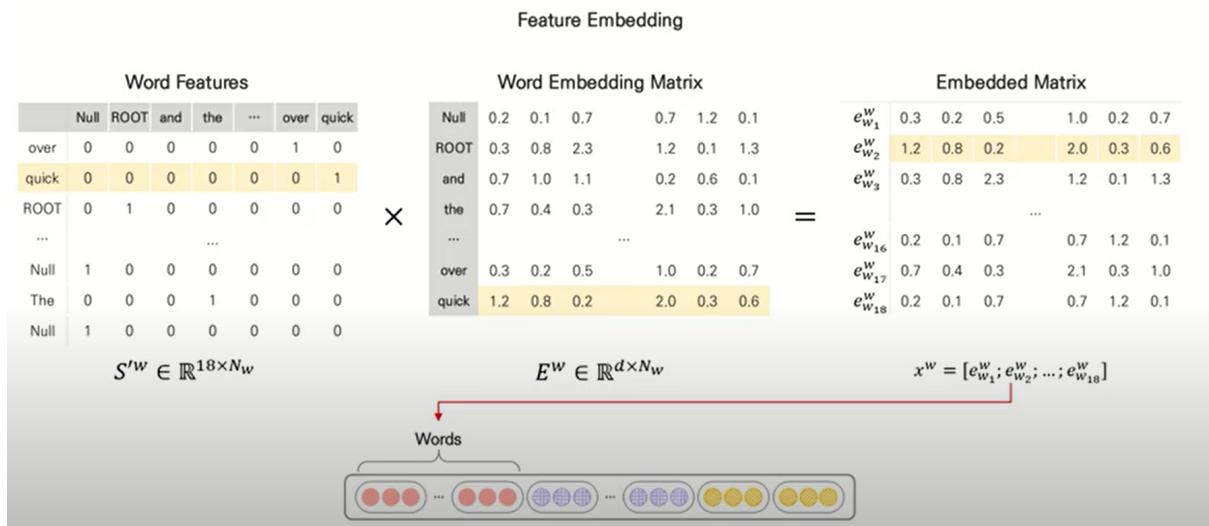
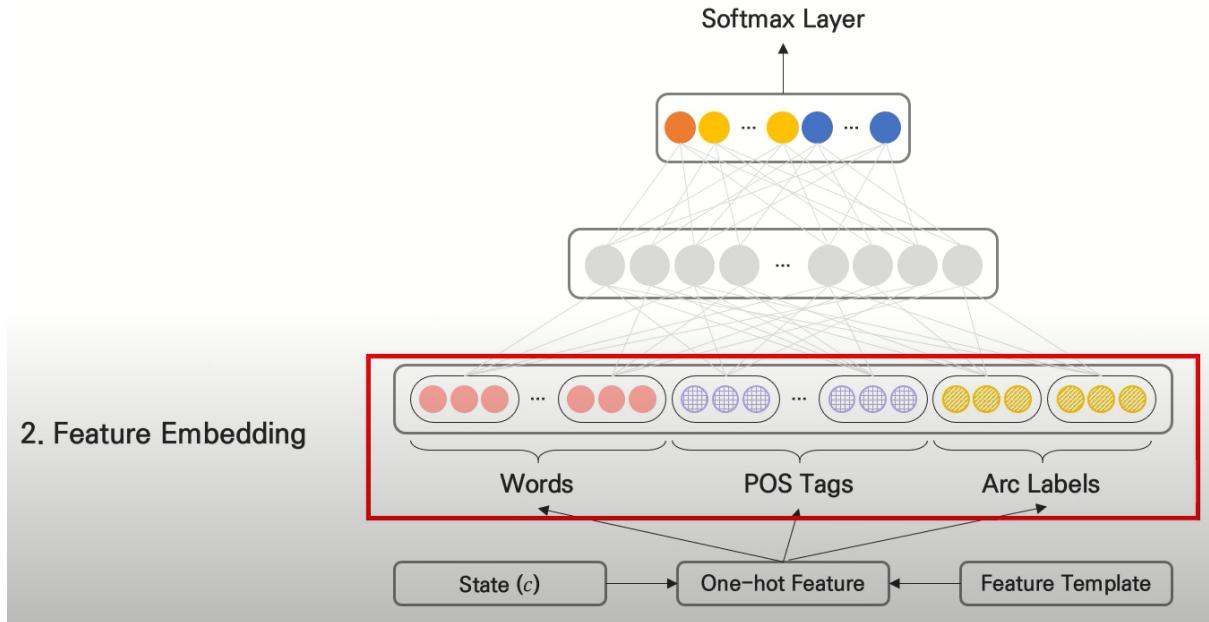
Word Features	POS Tag Features	Arc-label Features																																																																																																																																																																																																																								
$S^w = [\text{over}, \text{quick}, \text{ROOT}, \text{the}, \dots, \text{and}, \text{Null}, \text{Null}, \text{the}, \text{Null}]$	$S^t = [\text{IN}, \text{JJ}, \text{ROOT}, \text{DT}, \text{JJ}, \dots, \text{CC}, \text{Null}, \text{Null}, \text{DT}, \text{Null}]$	$S^l = [\text{Null}, \text{Null}, \text{Null}, \dots, \text{nsubj}, \text{conj}, \text{cop}, \text{cc}, \text{Null}, \dots]$																																																																																																																																																																																																																								
<table border="1"> <thead> <tr> <th></th><th>Null</th><th>ROOT</th><th>and</th><th>the</th><th>...</th><th>over</th><th>quick</th><th></th></tr> </thead> <tbody> <tr> <td>over</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr> <td>quick</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr> <td>ROOT</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>The</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> </tbody> </table>		Null	ROOT	and	the	...	over	quick		over	0	0	0	0	0	1	0		quick	0	0	0	0	0	0	1		ROOT	0	1	0	0	0	0	0		...									Null	1	0	0	0	0	0	0		The	0	0	0	1	0	0	0		Null	1	0	0	0	0	0	0		<table border="1"> <thead> <tr> <th></th><th>Null</th><th>ROOT</th><th>DT</th><th>NN</th><th>...</th><th>JJ</th><th>VBD</th><th></th></tr> </thead> <tbody> <tr> <td>IN</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>JJ</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> <tr> <td>ROOT</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>DT</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> </tbody> </table>		Null	ROOT	DT	NN	...	JJ	VBD		IN	0	0	0	0	0	0	0		JJ	0	0	0	0	0	1	0		ROOT	0	1	0	0	0	0	0		...									Null	1	0	0	0	0	0	0		DT	0	0	1	0	0	0	0		Null	1	0	0	0	0	0	0		<table border="1"> <thead> <tr> <th></th><th>Null</th><th>ROOT</th><th>nsubj</th><th>cc</th><th>...</th><th>cop</th><th>conj</th><th></th></tr> </thead> <tbody> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>Null</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>nsubj</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr> <tr> <td>conj</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td></td></tr> <tr> <td>cop</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td></td></tr> </tbody> </table>		Null	ROOT	nsubj	cc	...	cop	conj		Null	1	0	0	0	0	0	0		Null	1	0	0	0	0	0	0		Null	1	0	0	0	0	0	0		...									nsubj	0	0	1	0	0	0	0		conj	0	0	0	0	0	0	1		cop	0	0	0	0	0	1	0	
	Null	ROOT	and	the	...	over	quick																																																																																																																																																																																																																			
over	0	0	0	0	0	1	0																																																																																																																																																																																																																			
quick	0	0	0	0	0	0	1																																																																																																																																																																																																																			
ROOT	0	1	0	0	0	0	0																																																																																																																																																																																																																			
...																																																																																																																																																																																																																										
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
The	0	0	0	1	0	0	0																																																																																																																																																																																																																			
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
	Null	ROOT	DT	NN	...	JJ	VBD																																																																																																																																																																																																																			
IN	0	0	0	0	0	0	0																																																																																																																																																																																																																			
JJ	0	0	0	0	0	1	0																																																																																																																																																																																																																			
ROOT	0	1	0	0	0	0	0																																																																																																																																																																																																																			
...																																																																																																																																																																																																																										
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
DT	0	0	1	0	0	0	0																																																																																																																																																																																																																			
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
	Null	ROOT	nsubj	cc	...	cop	conj																																																																																																																																																																																																																			
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
Null	1	0	0	0	0	0	0																																																																																																																																																																																																																			
...																																																																																																																																																																																																																										
nsubj	0	0	1	0	0	0	0																																																																																																																																																																																																																			
conj	0	0	0	0	0	0	1																																																																																																																																																																																																																			
cop	0	0	0	0	0	1	0																																																																																																																																																																																																																			

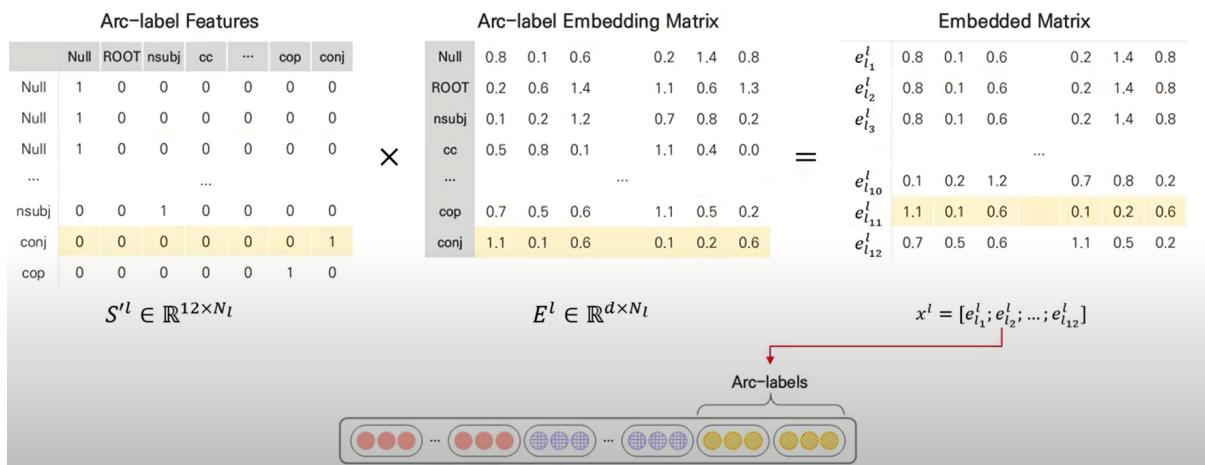
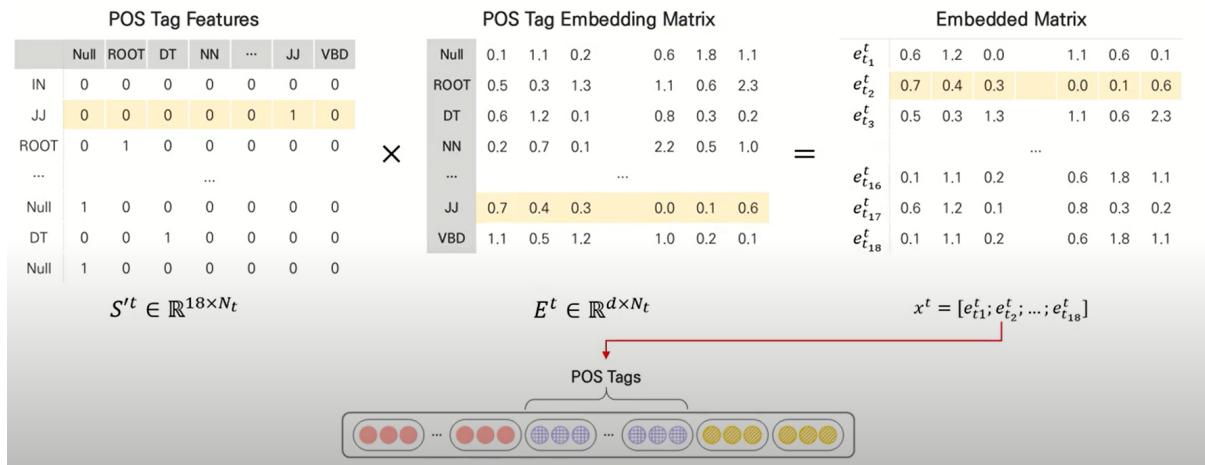
$S'^w \in \mathbb{R}^{18 \times N_w}$ $S'^t \in \mathbb{R}^{18 \times N_t}$ $S'^l \in \mathbb{R}^{12 \times N_l}$

과정2. Feature Embedding

[Neural Dependency Parser]

Chen and Manning (2014)



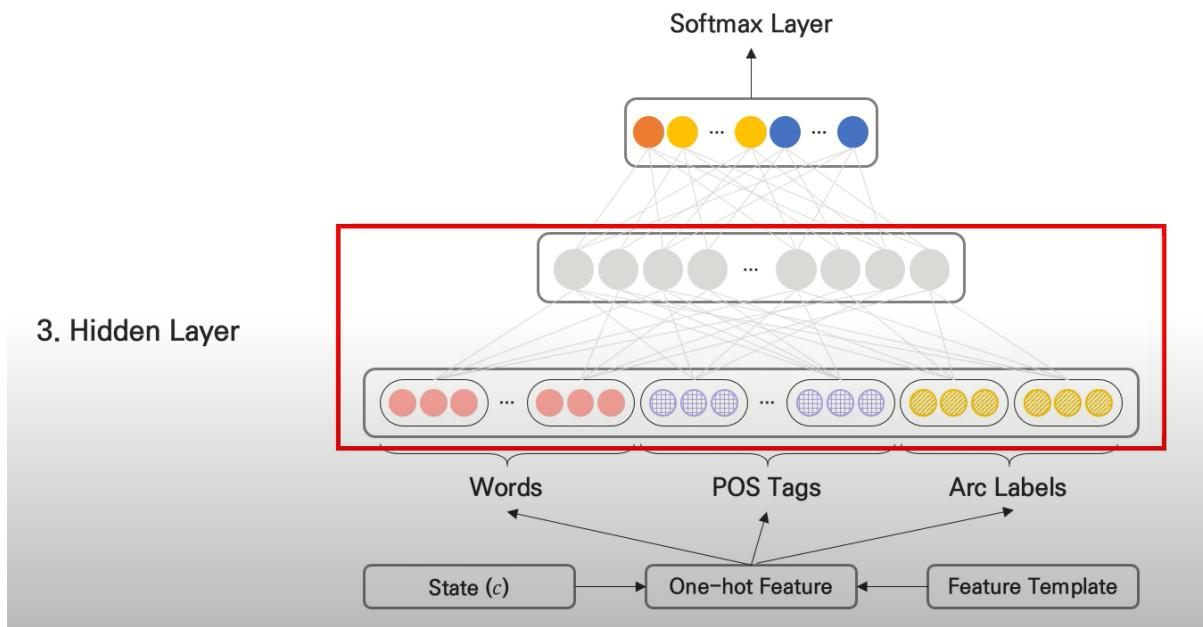


과정 3 : Hidden Layer

embedded된 feature를 hidden layer의 input에 넣는다.

[Neural Dependency Parser]

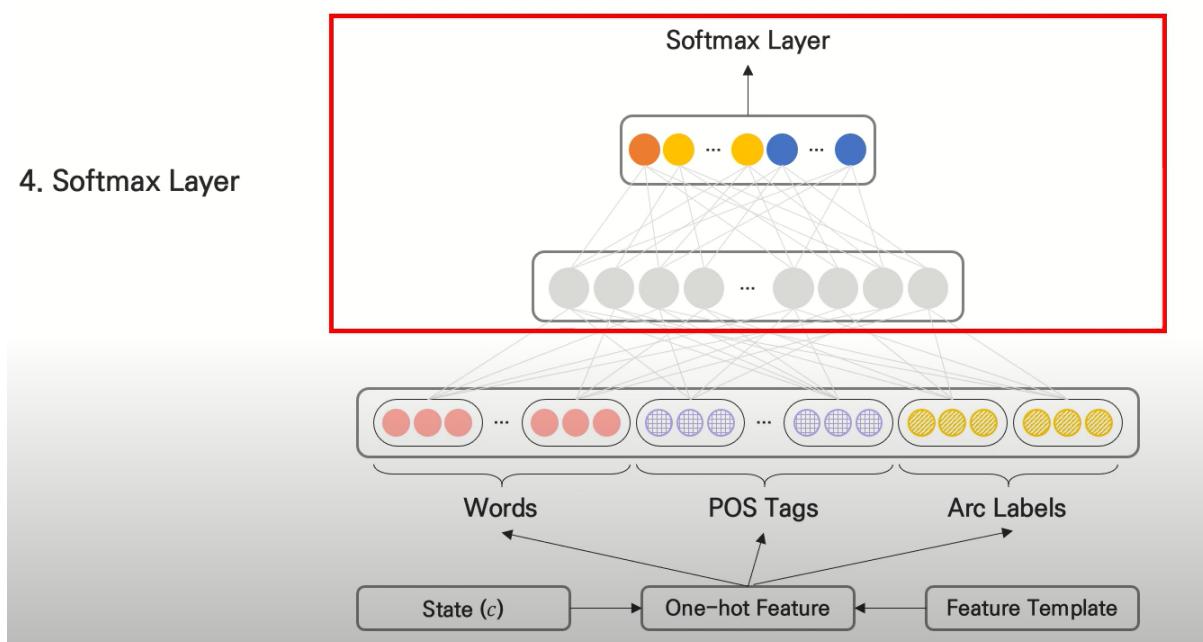
Chen and Manning (2014)

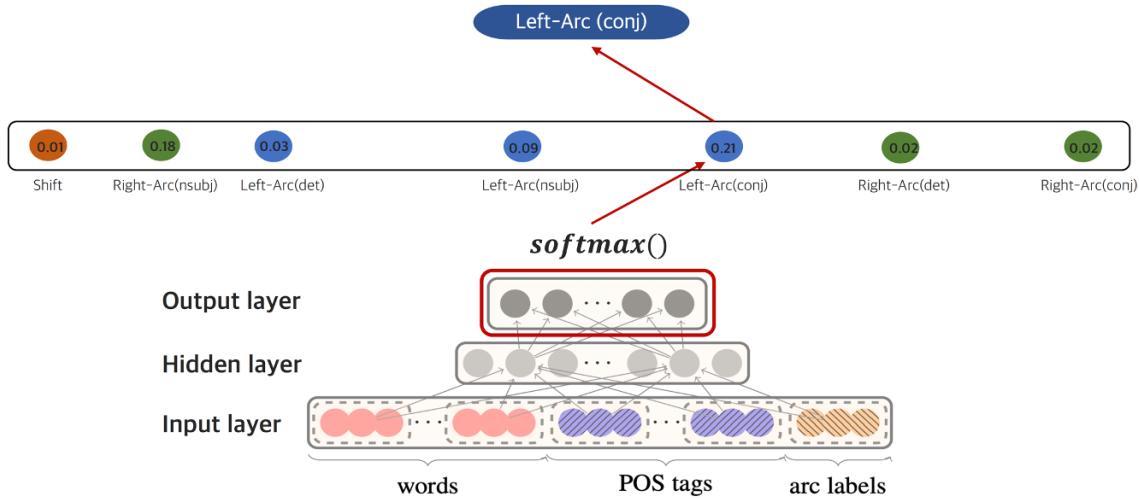


과정4. Output Layer

[Neural Dependency Parser]

Chen and Manning (2014)





hidden layer의 output을 linear projection한 후 softmax layer에 넣어 softmax probability를 구한다.

shift, left-arc, right-arc 중에서 softmax probability가 가장 높은 decision을 선택.

1-2. 성능

평가지표

- UAS (Unlabeled Attachment Score) : Arc 화살표 방향만 예측
- LAS (Labeled Attachment Score) : Arc 화살표 방향과 함께 label 또한 예측

malt parser : conventional한 indicator feature를 사용한 transition-based parser

chen and manning : neural network를 통해 dense feature를 사용한 transition-based parser

Results (English Penn Treebank Datasets)				
Parser	UAS	LAS	Sentence / sec.	비고
MaltParser (2007)	89.8	87.2	469	Transition-based Parser (Indicator Feature)
MSTParser (2007)	91.4	88.1	10	Graph-based Parser
TurboParser (2010)	92.3	89.6	8	Graph-based Parser
Chen and Manning (2014)	92.0	89.7	654	Transition-based Parser (Dense Feature)

neural dependency parser인 Chen and Manning 2014는 높은 성능을 지니면서 속도도 빠르다.

1-3. First win : 분산표현 (Distributed Representations)

- 분산표현 : 단어의 의미를 다차원 공간에 벡터화하는 방법
- 분산 표현을 통해 단어 간 의미적 유사성을 벡터화하는 작업인 word embedding을 하게되고 따라서 유사한 단어는 근접한 벡터를 지님.

1-4. Second win : Non - linear Classifiers

- softmax classifier와 같은 전통적인 ML classifier들은 강력한 분류기가 아니다.
 - why? they only give linear decision boundaries
- 하지만 Neural Networks는 non-linear decision boundary를 통해 복잡한 function를 학습할 수 있다.

1-5. Non-Greedy Algorithms

- Neural Dependency Parser인 Chen and Manning 2014는 greedy한 방식.
 - greedy : 현재 상태에서 가장 적합한 decision을 선택하는 방법
 - —> 따라서 local minima에 빠질 수 있음.
- 이를 보완하고자 Beam search, Conditional Random Field, Global Normalization을 적용한 transition-based dependency parser가 제안됨.
- 성능

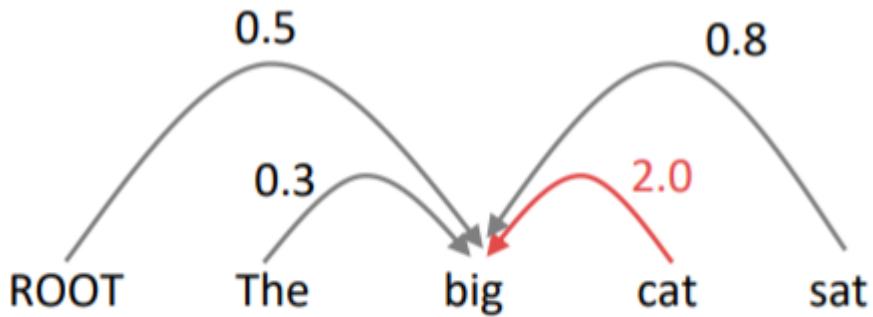
Results (English Penn Treebank Datasets)			
Parser	UAS	LAS	비고
Chen and Manning (2014)	92.0	89.7	Greedy transition-based parser
Weiss et al. (2015)	94.0	92.1	Beam search
Andor et al. (2016)	94.6	92.8	Beam search, Global normalization

Non-greedy 방식이 더 좋은 성능을 보임.

1-6. Graph-based dependency parser

- Compute a score for every possible dependency for each word

각 단어에 대해 가능한 모든 의존관계에 대한 점수를 계산하여
가장 값이 높은 것을 선택.



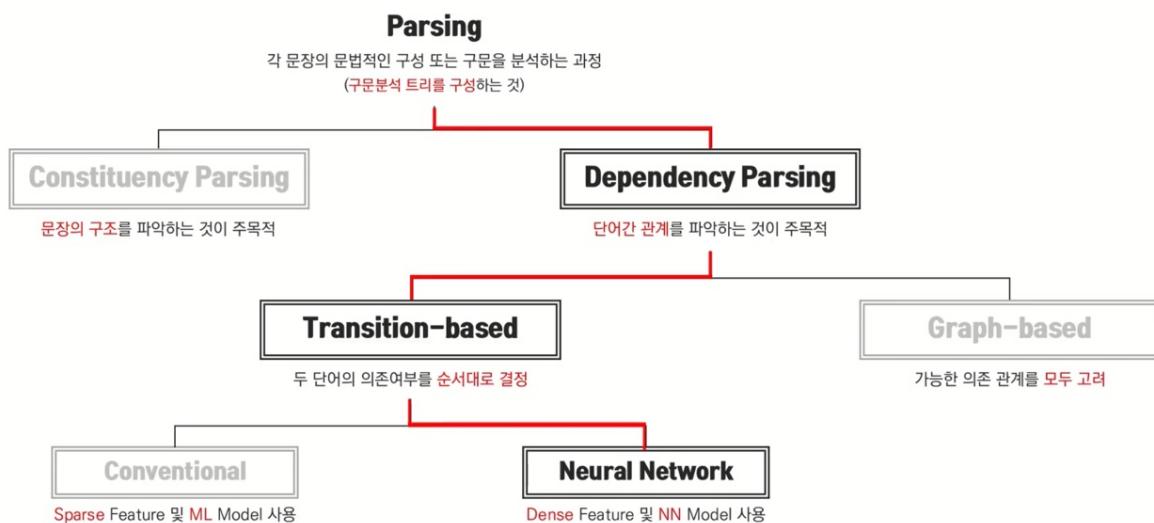
e.g., picking the head for “big”

값이 가장 큰 cat 선택

→ head(수식을 받는 단어) : “cat”

dependent(수식을 하는 단어) : “big”

1-7. Summary



2. A bit more about neural networks

2-1. Regularization

- 손실함수에 θ 를 증가시키는 penalty항을 추가하여 Overfitting 방지.

- A full loss function includes **regularization** over all parameters θ , e.g., L2 regularization:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N -\log \left(\frac{e^{f_{y_i}}}{\sum_{c=1}^C e^{f_c}} \right) + \lambda \sum_k \theta_k^2$$

- classic view : prevent overfitting.
- now : produce models that generalize well.

2-2. Dropout

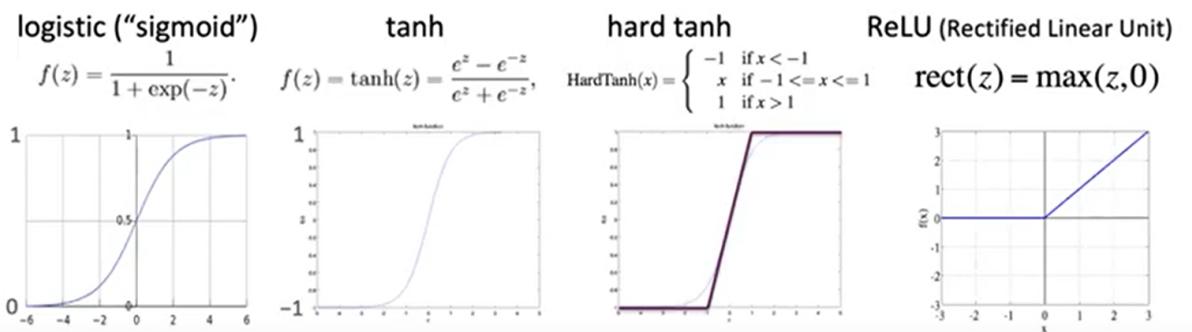
- 매개변수 중 일정량을 학습 중간마다 무작위로 사용하지 않는 방법
- good regularization method

2-3. Vectorization

- word vector를 개별적으로 반복해서 실행하는 것 보다 matrix 연산을 사용하는 것이 학습 시간 감소에 도움이 된다. 이러한 과정을 vectorization이라고 부름.

2-4. Non-linearities

- 활성화 함수로 다양한 비선형 함수를 사용함.



- Both logistic and tanh are still used, but are no longer the defaults for making deep networks.
- First thing you should try is ReLU.
- Leaky ReLU, Parametric ReLU 등 ReLU를 보완하고자 변형된 ReLU 함수들이 등장하는 추세

2-5. Parameter initialization

- Neural Network 학습 전 가중치들을 초기화하는 과정.
- 일반적으로 small random value로 parameter를 초기화함.
 - hidden layer & output layer의 bias term은 0으로 초기화
 - 다른 모든 weight들은 uniform distribution에서 임의로 sampling
- Xavier Initialization 방법도 자주 쓰임.
previous layer size(n_{in})와 next layer size(n_{out})에 맞게 weight의 분산을 조절해주는 방식.

$$\text{Var}(W_i) = \frac{2}{n_{in} + n_{out}}$$

2-6. Optimizer

- parameter를 조정해서 손실함수의 최솟값을 찾는 알고리즘.
- back propagation (역전파)를 수행함으로써 parameter를 최적화한다.
- SGD를 사용해도 최적화는 잘 된다.
→ 더 좋은 성능을 얻으려면 learning rate도 튜닝!
- 복잡한 신경망 구조 : Adaptive Optimizer
 - Adaptive Optimizer : 축적된 gradient값을 바탕으로 step size, parameter를 조절하는 방식.
 - ex) Adagrad, RMSprop, Adam, SparseAdam... :
이 모델들은 parameter 별로 학습률을 차등적으로 제공.

2-7. Learning Rates

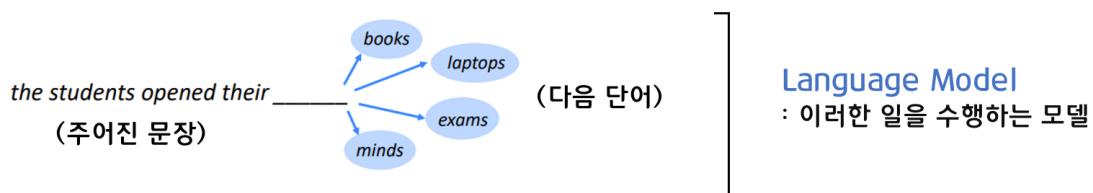
- 주로 0.001 사용
 - 값이 너무 크면 → 모델 발산
 - 값이 너무 작으면 → 업데이트 양이 작아 학습이 느려짐.

- 학습이 진행될수록 learning rate를 감소시키는 방법이 good.
 - by hand : k epoch 마다 learning rate를 반으로 줄이기.
 - by formula
 - cyclic learning rate
- Learning Rate Scheduling을 통해 상황에 따라 변환시키며 학습 진행.

3. Language Modeling + RNNs

Language Modeling

: 현재까지 주어진 문장의 다음 단어를 예측하는 것



- More formally : 단어들의 시퀀스 $x_1 \dots x_t$ 가 주어졌을 때, 다음 단어 x_{t+1} 의 확률분포를 계산하는 것 —> 즉 Language Model은 텍스트에 확률을 할당하는 시스템으로 생각 할 수 있다.

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

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

3-1. n-gram Language Models

(1) Definition

- Deep Learning 이전에 Language Model에 주로 사용된 모델

Definition of N-gram

: 연속된 N개의 단어 덩어리

ex)

- **uni**grams: "the", "students", "opened", "their"
- **bi**grams: "the students", "students opened", "opened their"
- **tri**grams: "the students opened", "students opened their"
- **4-grams**: "the students opened their" **4개의 연속된 단어** -> **4-gram**

- **Idea**: 서로 다른 N-grams의 빈도수를 계산하고 이를 다음에 올 단어를 예측하는 데 사용한다.
- 빈칸에 들어갈 단어의 확률을 구하기 전에 먼저 가정을 하나 한다.
→ Markov assumption : $x^{(t+1)}$ 번째 단어는 앞의 $(n-1)$ 개의 단어에만 영향을 받는다.
- First we make a **Markov assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words

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

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

(definition of
conditional prob)

- **Question**: n-gram과 (n-1)-gram의 확률을 어떻게 얻을 수 있을까?
- **Answer**: 큰 텍스트 둥치(corpus)에서 그들의 빈도를 counting하여 구한다.

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

(2) Example : 4-gram model

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

바로 앞 3개의 단어들만 고려하고 'students' 이전 단어들을 고려하지 않는다.

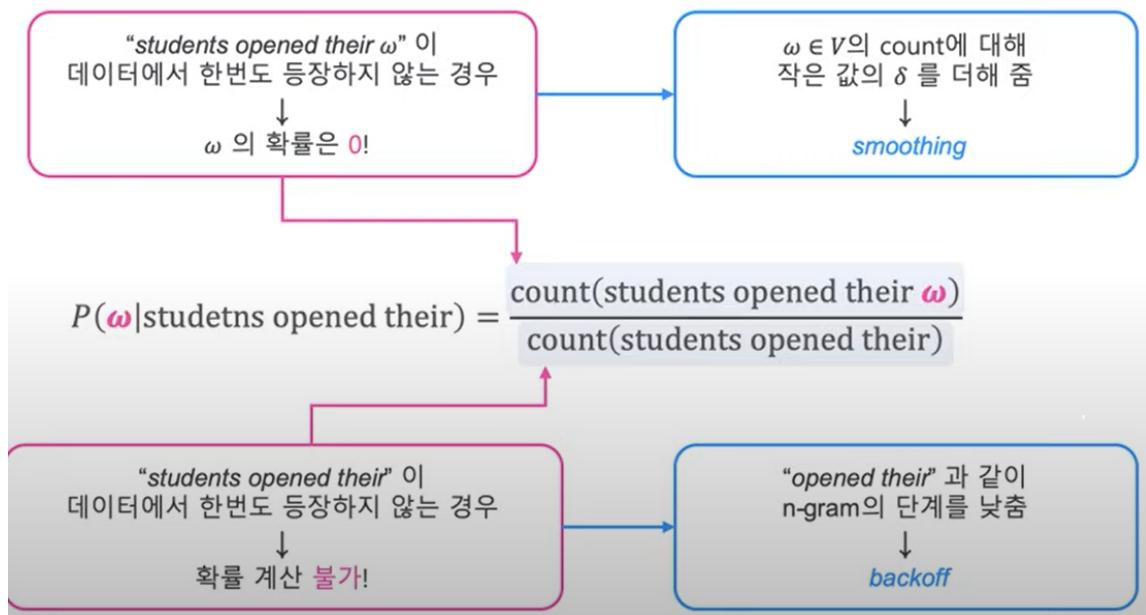
$$P(w| \text{students opened their}) = \frac{\text{count(students opened their } w\text{)}}{\text{count(students opened their)}}$$

- For example, suppose that in the corpus:

- "students opened their" : 1,000번 발생
- "students opened their books" : 400번 발생
 $\Rightarrow P(\text{books} | \text{students opened their})=0.4$
- "students opened their exams" : 100번 발생
 $\Rightarrow P(\text{exams} | \text{students opened their})=0.1$

(3) Problem

- sparsity problems : n이 커질수록 안좋아지며, 일반적으로 $n < 5$ 로 설정.



- storage problems : n이 커지거나 corpus가 증가하면 모델의 크기가 증가함.

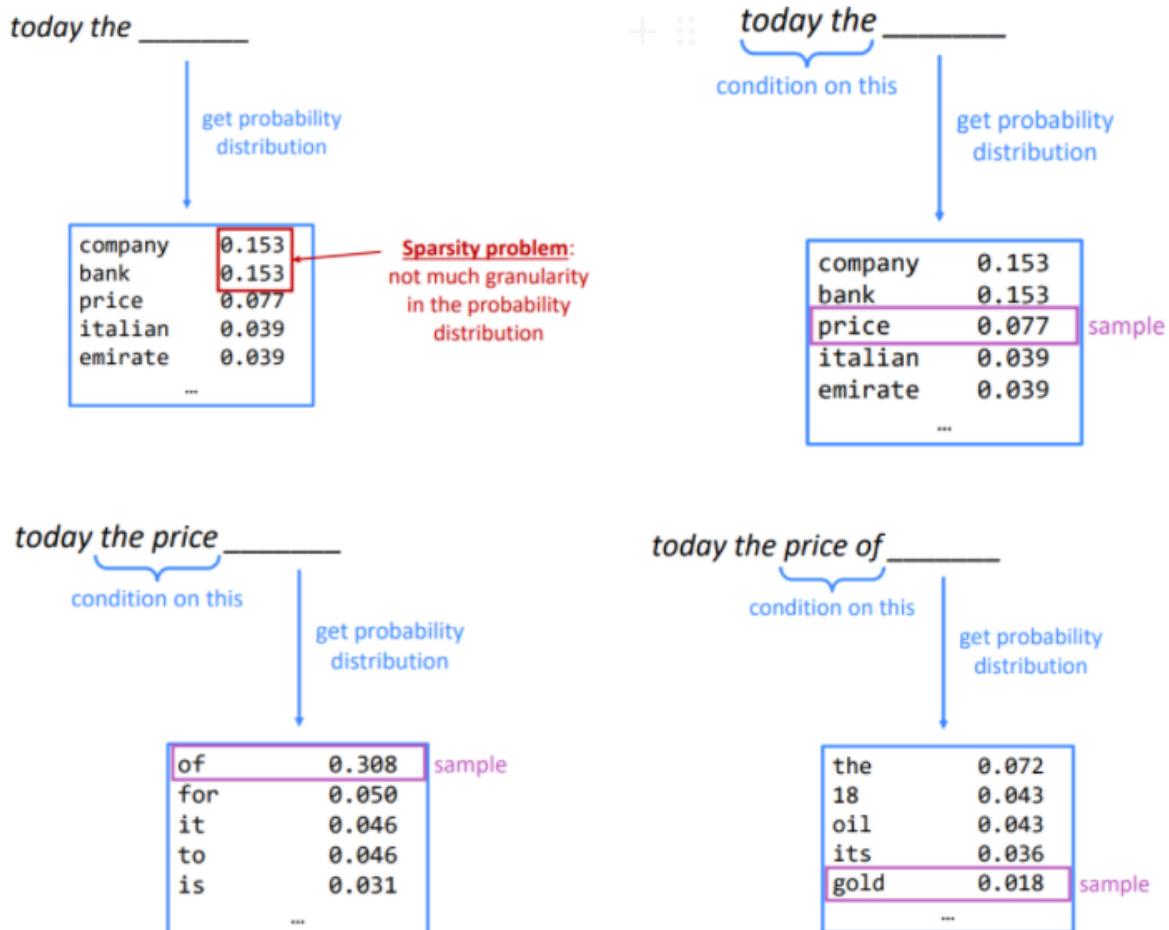
Corpus 내 모든 n-gram에 대한 count를 저장해 줘야함

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

- **n을 작게 선택하면**, 훈련 코퍼스에서 카운트는 잘 되겠지만 근사의 정확도는 현실의 확률분포와 멀어짐 —> 적절한 n을 선택.

(4) Process

- 3-gram으로 정하면 n-1인 2개의 단어(today, the)로 다음 단어를 예측.
- 해당 단어들을 기반으로 확률 분포를 구하면, 가능성 있는 단어들의 확률 분포를 얻을 수 있다.



- 결과

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

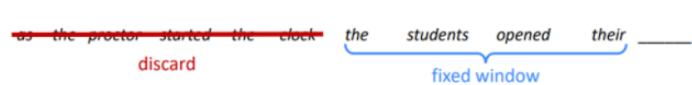
- 생각보다 문법적인 결과.
- 하지만 전체적인 의미에서 일관성이 없다.
'다음 단어는 오직 직전의 $n-1$ 개의 단어에만 영향을 받는다'라는 가정 때문에 이전 문맥을 충분히 반영하지 못한다.
- n 의 크기를 늘리면 이러한 문제를 어느정도 해결할 수 있겠지만 동시에 Sparsity 문제가 심해지게 된다.

3-2. Neural Language Model

Neural Network를 이용한 가장 간단한 Neural Language model

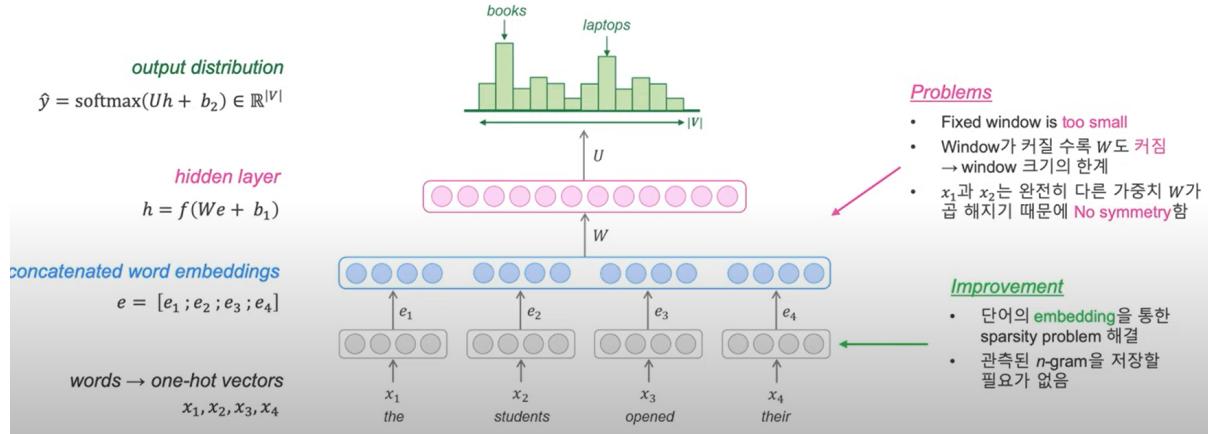


Lecture 3에서 NER에 적용했던
Window-based neural network는
Center를 기준으로 앞뒤의 window를 정했다면



Lecture 6에서는 예측할 단어의
이전에 window를 고정

A fixed-window neural Language Model



<improvements>

- 단어의 embedding을 통한 sparsity problem 해결
- 관측된 n-gram을 저장할 필요가 없음

<Problems>

- 고정된 window size가 너무 작다. → n-gram 모델과 같이 문맥을 반영하지 못한다.
- Window size를 크게 하면 W 도 커진다. → window 크기의 한계
- x_1 와 x_2 에는 완전히 다른 가중치 W 가 곱해지기 때문에 No symmetry하다. → 단어의 위치에 따라 곱해지는 가중치가 다르기 때문에 모델이 비슷한 내용을 여러 번 학습하는 비효율성을 가진다.

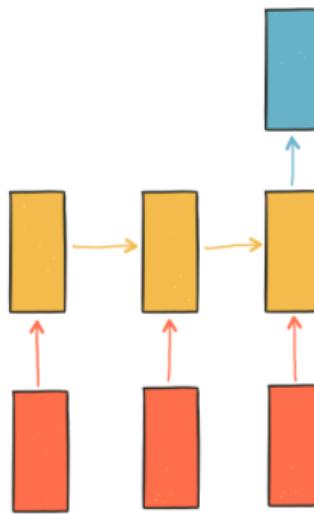
→ 따라서 이를 해결할 수 있는 어떤 길이의 input도 처리할 수 있는 model이 필요하다!

3-3. RNN Language Model

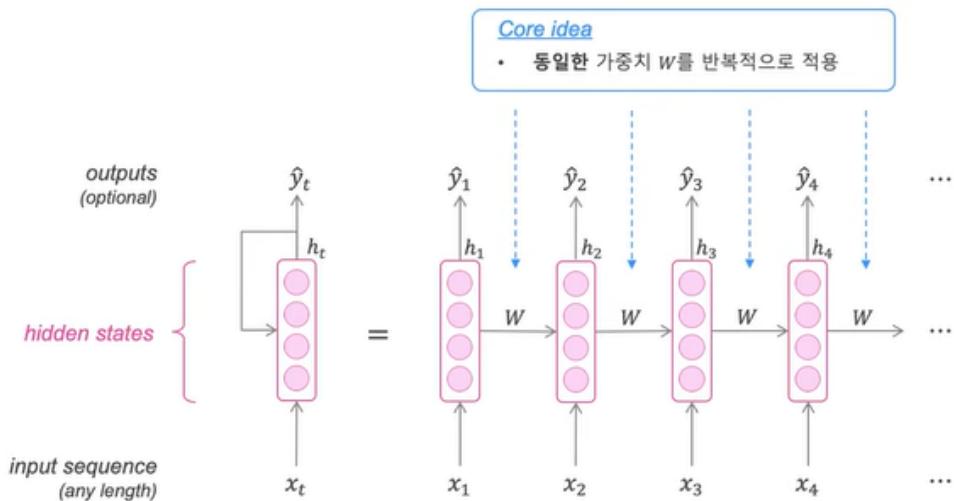
(1) RNN (Recurrent Neural Network, 순환신경망)

시퀀스 데이터를 모델링 하기 위해 등장

기존의 뉴럴 네트워크와 다른 점은 ‘기억’(hidden state)을 하는 것.



- 위 다이어그램에서 빨간색 사각형은 입력, 노란색 사각형은 기억, 파란색 사각형은 출력을 나타냅니다.
- 첫번째 입력이 들어오면 첫번째 기억이 만들어집니다. 두번째 입력이 들어오면 기존의 기억과 새로운 입력을 참고하여 새 기억을 만듭니다.
- 입력의 길이만큼 이 과정을 얼마든지 반복할 수 있으며, RNN은 이 요약된 정보를 바탕으로 출력을 만들어냅니다.
- Core idea:** 동일한 가중치 W 를 반복적으로 적용.
→ neural network의 문제였던 no symmetry의 문제를 해결.



This is a very long sentence explaining about a long sentence.

rnn은 다 본다!

trigram bigram

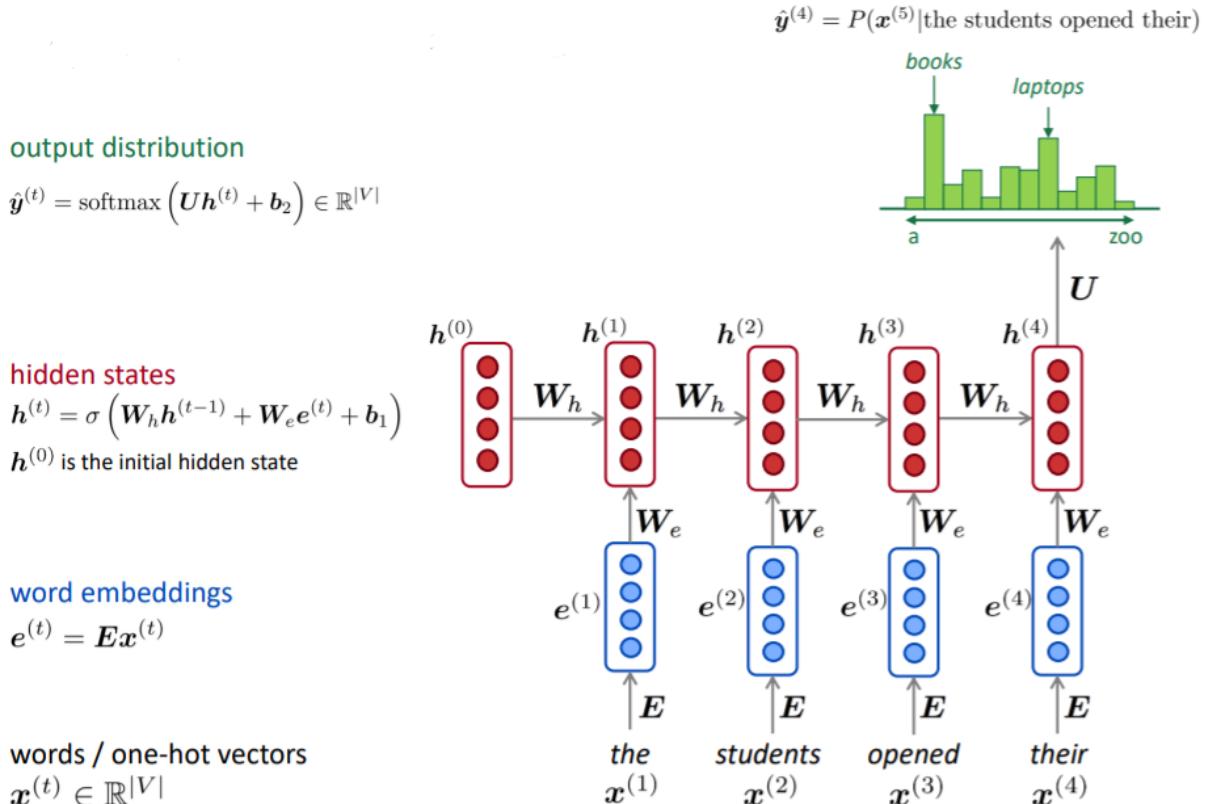
Target

기존의 신경망 구조 -> 모든 입력이 각각 독립적이라고 가정

RNN -> 이전의 계산 결과에 영향을 받음

(현재까지 계산된 결과에 대한 메모리 정보를 갖고 있다고 볼 수도 있음)

- 구조



- RNN 계층은 그 계층으로의 입력과 1개 전의 RNN 계층으로부터의 출력을 받는다. 그리고 이 두 정보를 바탕으로 현 시각의 출력을 계산한다.
- $x^{(t)}$: t 시간 스텝에서의 입력 벡터, one-hot vector 형태
- $e^{(t)}$: 입력 벡터 $x(t)$ 에 대한 word embedding
- 은닉층 : $h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$
 - $h^{(t)}$: t 시간 스텝에서 RNN의 기억을 담당하는 hidden state
 - 입력 $e^{(t)}$ 와 과거의 기억 $h^{(t-1)}$ 의 조합으로 만들어짐.

- W_e : 새로운 입력($e^{(t)}$)이 새로운 기억에 영향을 미치는 정도
- W_h : 과거의 기억($h^{(t-1)}$)이 새로운 기억에 영향을 미치는 정도
- 비선형 함수 σ 는 주로 tanh나 ReLU 사용
- 출력층 : $\hat{y}_t = \text{softmax}(Uh^{(t)} + b_2)$
 - 출력값 \hat{y}_t 는 $h^{(t)}$ 에 의해 계산된다.
 - U : hidden state와 출력을 연결시켜주며 출력 벡터의 크기를 맞춰주는 역할.
 - 출력을 확률값으로 변환하기 위해 softmax 함수를 적용.
(softmax 함수는 모든 출력값을 0 ~ 1 사이로 변환하고, 출력값들의 합이 1이 되도록 함.)

(2) Improvements & Disadvantages

< Improvements >

- 입력(input)의 길이에 제한이 없음.
- 길이가 긴 time step t에 대해 처리가 가능함.
- 아무리 입력이 길어도 모델의 크기가 증가하지 않는다. (모델의 크기는 Wh와 We로 고정되어 있다.)
- 모든 time step에 대해 동일한 가중치를 적용한다. (symmetry)

< Disadvantages >

- 이전 hidden state를 기반으로 다음 hidden state를 계산해야 하기 때문에 순차적인 계산이 필요하다.
→ 반복하는 과정에서 계산이 매우 느리다.
- 큰 단점은 **입력과 출력 단계 사이의 거리가 멀어질 수록 그 관계를 학습하기 어려워진다.**
- 신경망이 깊어질수록 **Vanishing gradient**로 인해, 문장 초반부의 단어가 결과에 미치는 영향이 적어진다.

(3) Training a RNN Language Model

1. 단어 $x^{(1)} \dots x^{(t)}$ 들로 이루어진 시퀀스의 corpus를 준비한다.

2. $x^{(1)} \dots x^{(t)}$ 를 순서대로 RNN language model에 입력하고, 매 step t 에 대한 출력분포 \hat{y}_t 를 계산한다.

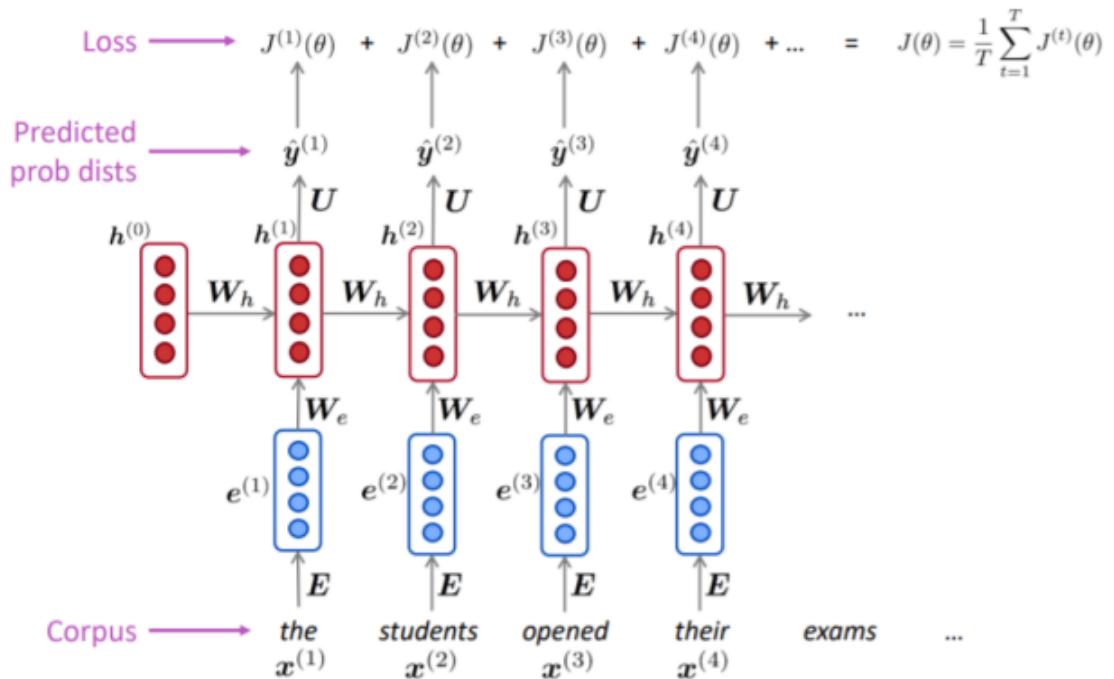
- 주어진 단어에서부터 시작하여 그 다음에 나올 수 있는 모든 단어들에 대한 확률을 예측

3. step t 에 대한 손실함수 Cross-Entropy를 계산한다.

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

4. 모든 step t 에 대한 loss값을 구한 후 이 값들의 평균을 계산한다.

이 값이 바로 training set에 대한 loss값이 된다.



5. loss값을 최소로 하자 gradient descent를 통해 weight를 optimization한다.

$$\theta^{new} = \theta^{old} - \underbrace{\alpha \nabla_{\theta} J(\theta)}_{\text{gradient}}$$

learning rate

- 다만 전체 corpus $x^{(1)} \dots x^{(t)}$ 에 대한 loss와 gradients를 계산하는 데는 많은 시간이 걸리기 때문에, 실제로는 문장이나 문서 단위로 입력을 주기도 한다.
- SGD를 통해서 Optimize하는 것도 하나의 방법이다.

(4) Backpropagation for RNNs

Q : 반복되는 가중치 행렬 W_h 에 대한 $J^{(t)}(\theta)$ 의 gradient는 어떻게 구할까?
 $(J^{(t)}(\theta)$ 는 step t 에서의 손실함수 값)

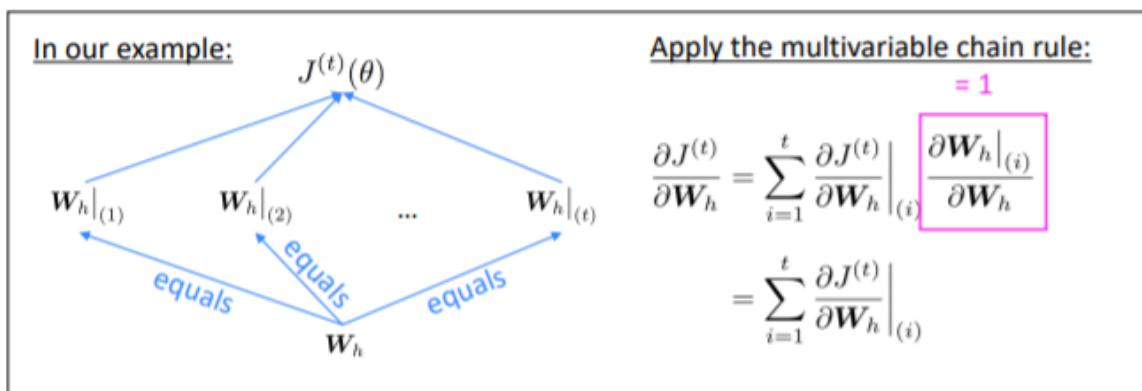
A : 각 시점에서 발생한 hidden state까지의 gradient 값을 모두 합친 값

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}$$

Why? using multivariable chain rule.

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

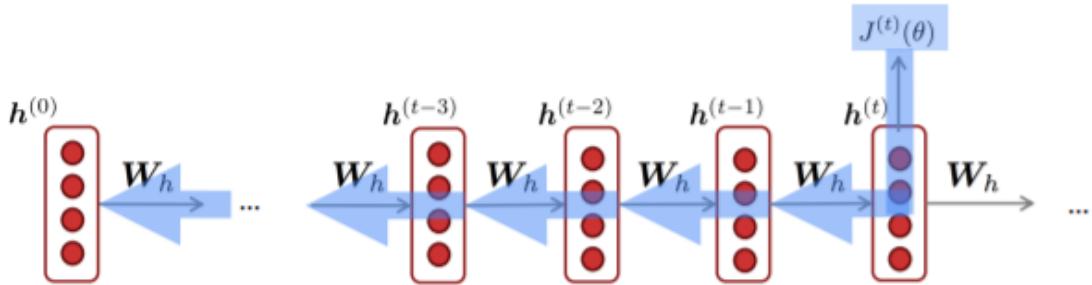


Q : How can we calculate this?

A : Using BPTT(Backpropagation Through Time)

- RNN을 학습하는 것은 기존의 신경망 모델을 학습하는 것과 매우 유사함.

- 그러나 기존의 backpropagation과 다르게 RNN은 시간, 시점의 수가 영향을 주어 Backpropagation Through Time (BPTT) 즉 '시간에 따른 역전파'라는 약간 변형된 알고리즘을 사용함.
(각 출력 부분에서의 gradient가 현재 시간 스텝에만 의존하지 않고 이전 시간 스텝들에도 의존하기 때문)
- timesteps $i=t, \dots, 0$ 에 따라 gradients를 더해가면서 backpropagate한다.



$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}$$

Question: How do we calculate this?

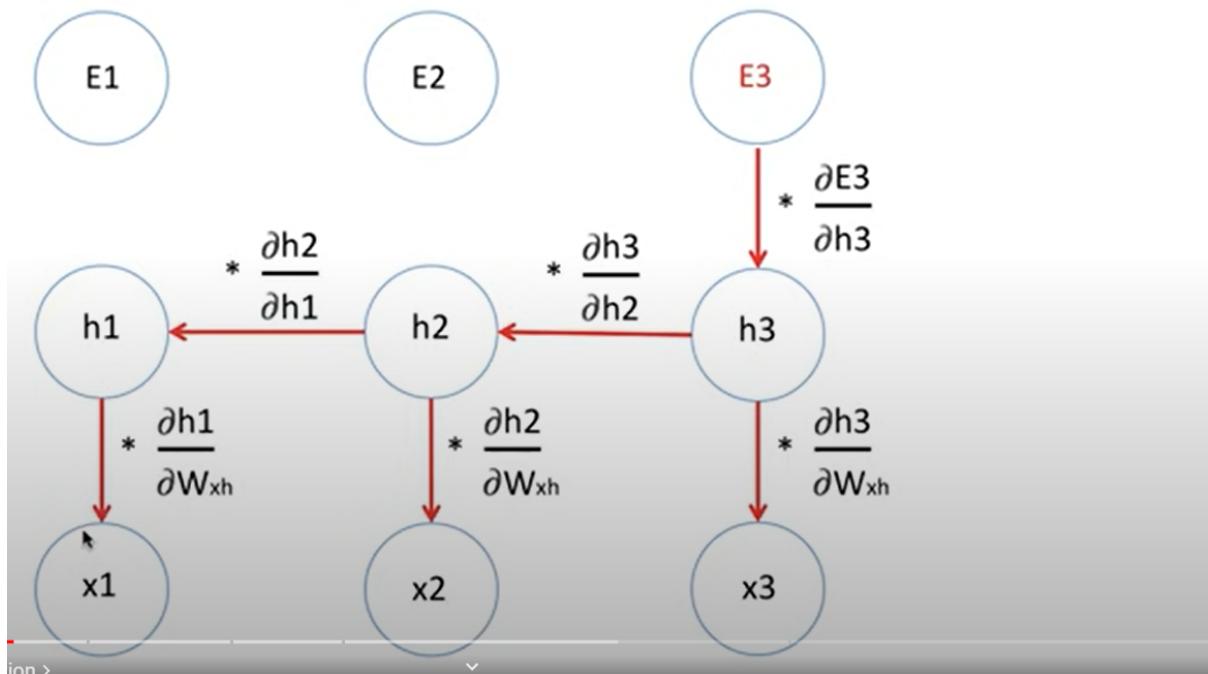
Answer: Backpropagate over timesteps $i=t, \dots, 0$, summing gradients as you go.
This algorithm is called "backpropagation through time"
[Werbos, P.G., 1988, *Neural Networks 1*, and others]

예시 : 3번째 step에서의 error값(E3)를 가중치(W)에 대해 미분하는 경우

→ 각 시점에서 발생한 hidden state까지의 gradient 값을 모두 합친다

E3 derivative calculation

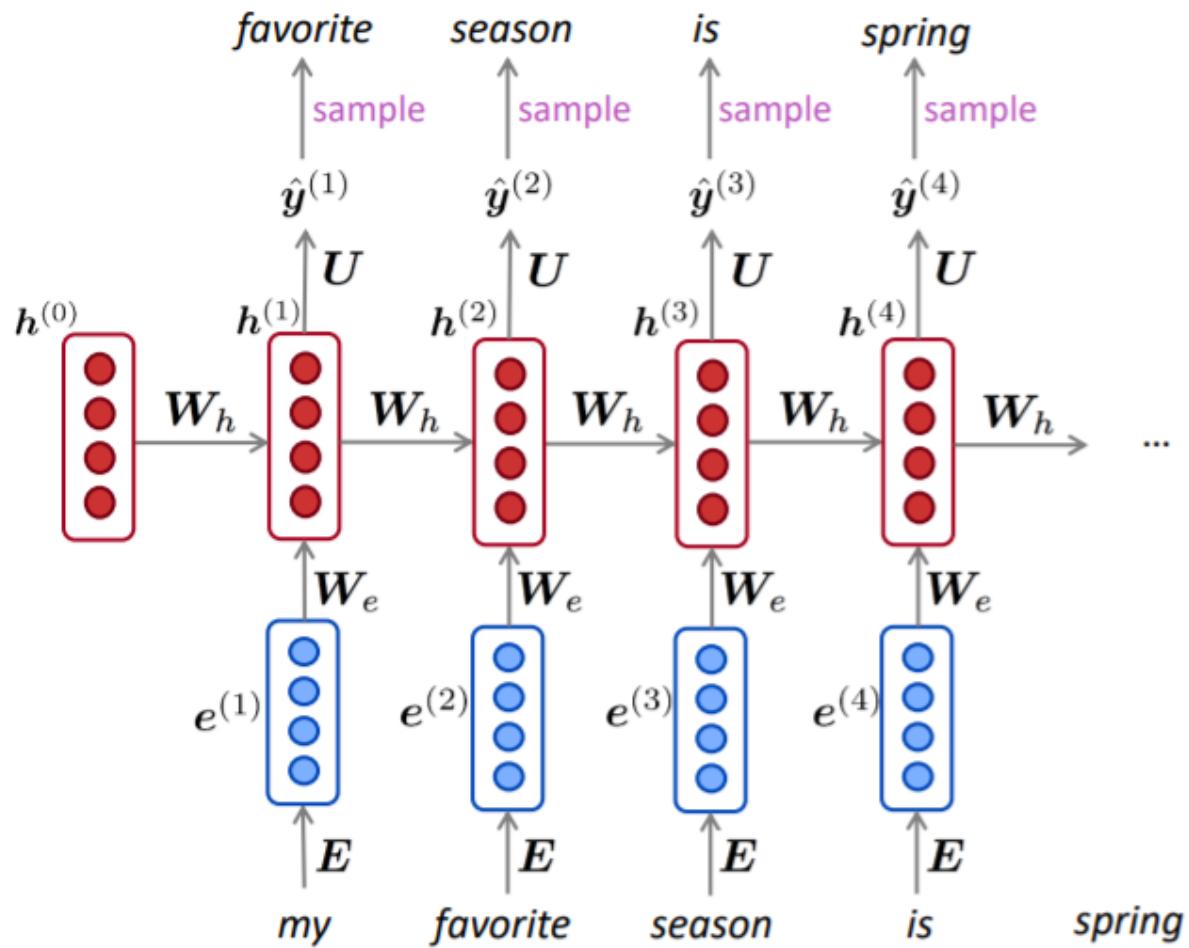
$$\frac{\partial E3}{\partial W} = \frac{\partial E3}{\partial h3} * \frac{\partial h3}{\partial W_{xh}} + \frac{\partial E3}{\partial h3} * \frac{\partial h3}{\partial h2} * \frac{\partial h2}{\partial W_{xh}} + \frac{\partial E3}{\partial h3} * \frac{\partial h3}{\partial h2} * \frac{\partial h2}{\partial h1} * \frac{\partial h1}{\partial W_{xh}}$$



= 3번째 hidden state에 대한 미분값 + 2번째 hidden state에 대한 미분값 + 1번째 hidden state에 대한 미분값

(5) Generating text with a RNN Language Model

- n-gram Language Model과 같이 RNN Language Model을 사용해 반복된 샘플링으로 text를 생성할 수 있다.
- 샘플링된 출력은 다음 단계의 입력이 된다.



▼ 특정 종류의 text에서 RNN-LM을 학습시킨 다음 해당 style의 text를 생성할 수 있다.

- RNN-LM trained on **Harry Potter**:



“Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

- RNN-LM trained on **recipes**:



Title: CHOCOLATE RANCH BARBECUE

Categories: Game, Casseroles, Cookies, Cookies

Yield: 6 Servings

2 tb Parmesan cheese -- chopped

1 c Coconut milk

3 Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients and stir in the chocolate and pepper.

(6) Evaluating Language Models

- Language Model : 주어진 과거 단어(정보)로부터 다음에 출현할 단어의 확률분포를 출력하는 모델
- Language Model을 평가하는 대표적인 척도는 **Perplexity**이다.

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

Inverse probability of corpus, according to Language Model

Normalized by number of words

⇒ Language Model을 통해 예측한 corpus의 inverse를 corpus의 길이로 normalize해준다.

- 이는 cross-entropy에다가 로그를 취한값의 exponential 값과 같다.

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

- Perplexity 값이 작을수록 좋은 Language Model.

The diagram illustrates the progression of Language Models. It starts with an *n-gram model* and moves through various stages of increasing complexity, including RNNs (RNN-1024, RNN-2048), matrix factorization, and different LSTM architectures (LSTM-2048, 2-layer LSTM-8192). The final models listed are *Ours small* (LSTM-2048) and *Ours large* (2-layer LSTM-2048). A vertical double-headed arrow on the right indicates that Perplexity improves as it goes down, with lower values being better.

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

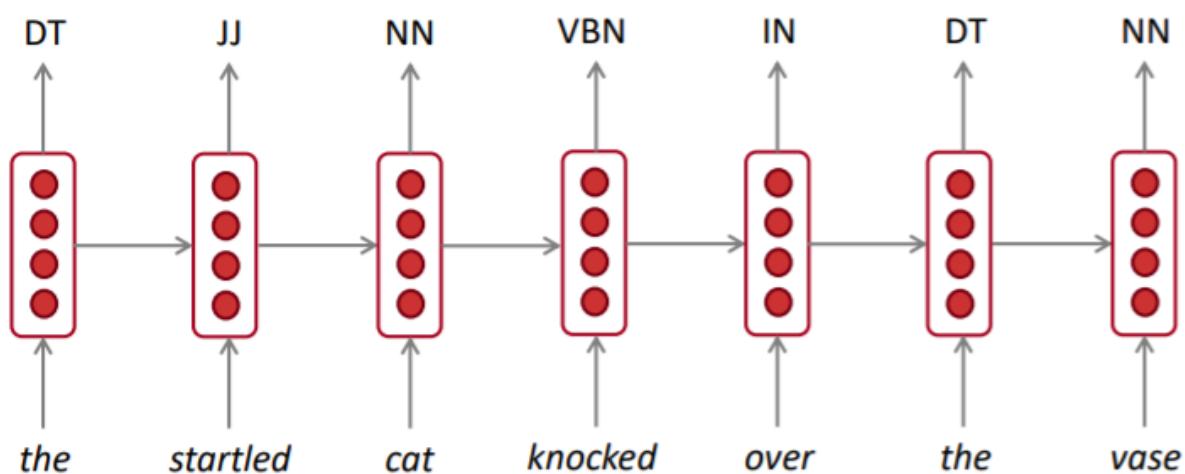
최근 Perplexity 값이 감소하고 있음을 확인할 수 있다.

(7) Recap

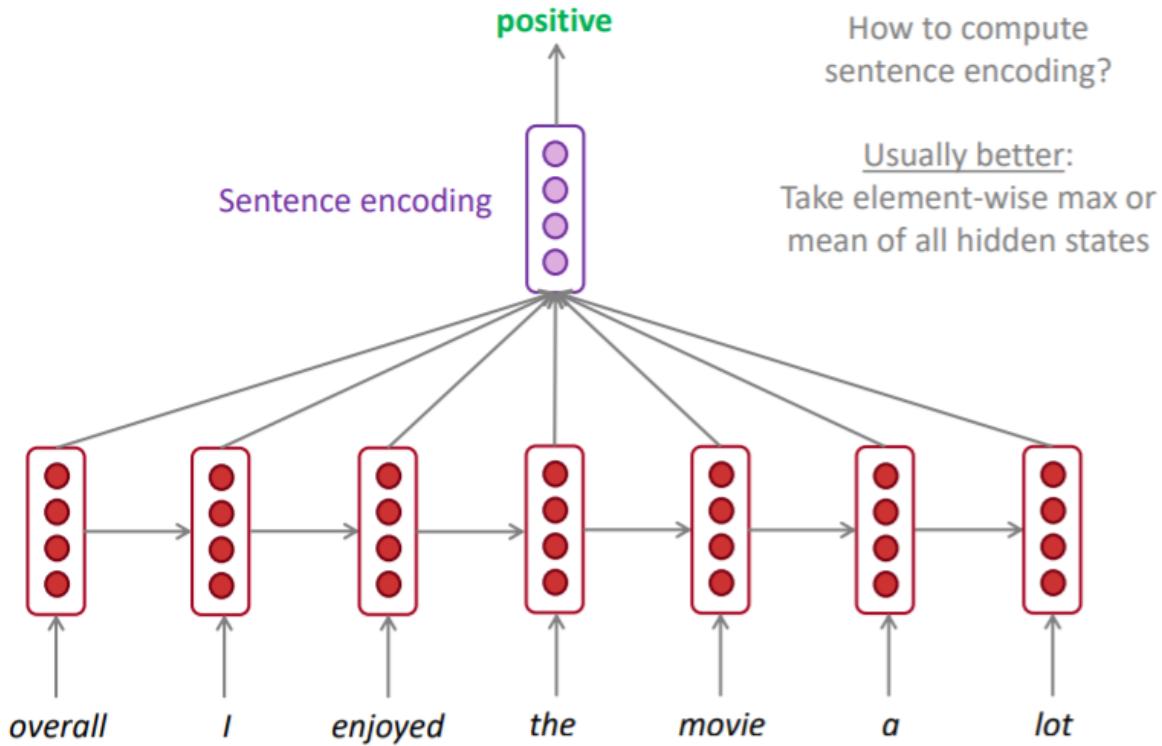
- **Language Model**: A system that predicts the next word
- **Recurrent Neural Network**: A family of neural networks that:
 - Take sequential input of any length
 - Apply the same weights on each step
 - Can optionally produce output on each step
- Recurrent Neural Network ≠ Language Model
- We've shown that RNNs are a great way to build a LM.
- But RNNs are useful for much more!

(8) Other RNN uses

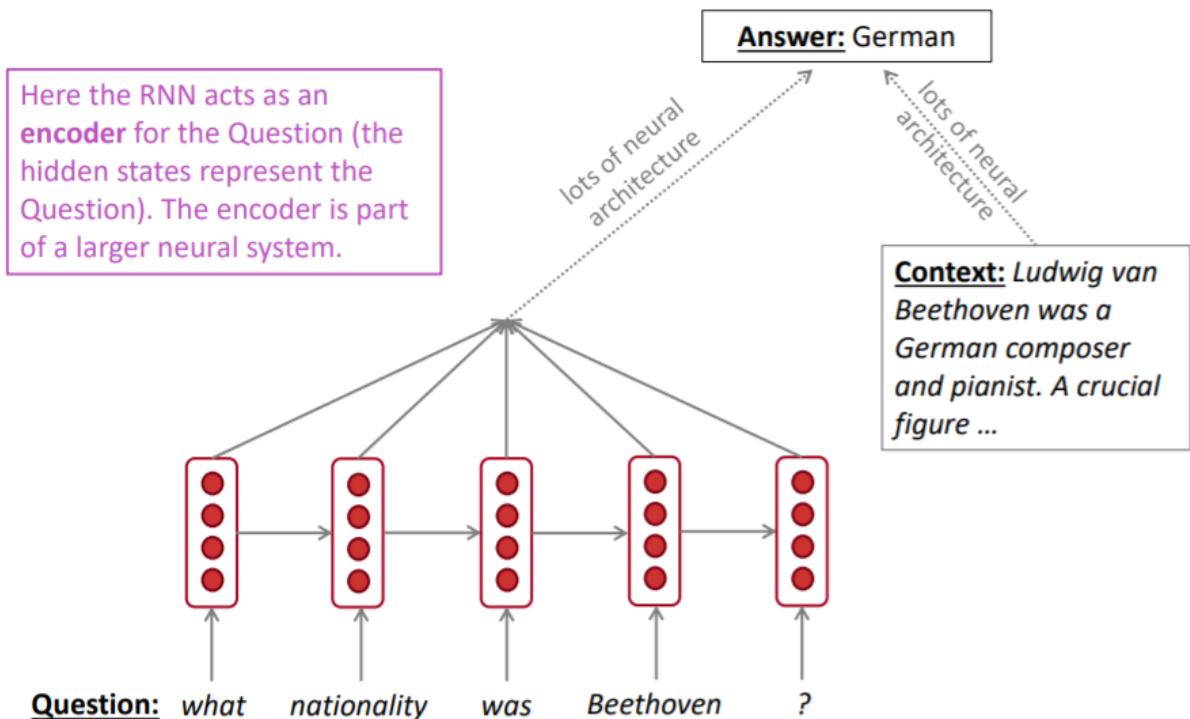
- RNNs can be used for tagging
 - ex) part-of-speech tagging (POS 태그), named entity recognition



- RNNs can be used for sentence classification
 - ex) sentiment classification (감성 분석)

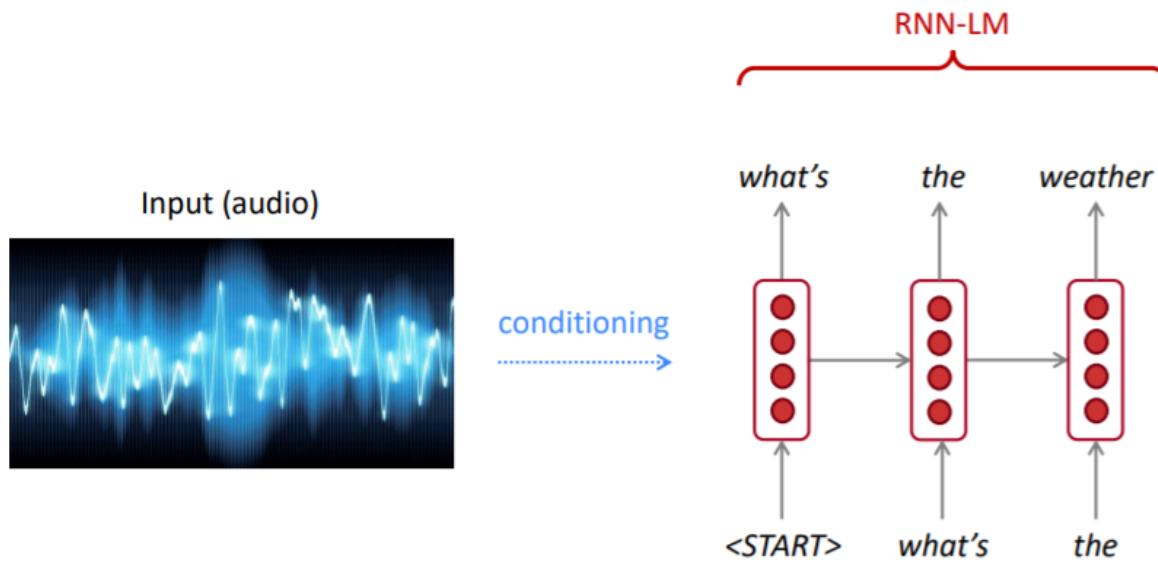


- **RNNs can be used as an encoder module**
ex) question answering, machine translation, many other task



- **RNN-LMs can be used to generate text**
ex) speech recognition(음성 인식), machine translation, summarization

input : audio signal

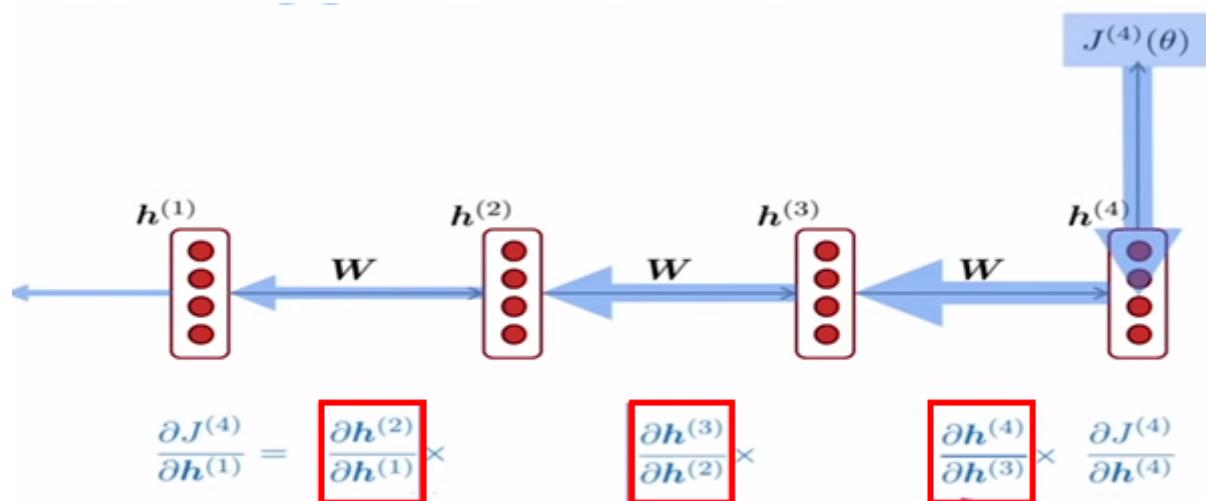


This is an example of a *conditional language model*.
We'll see Machine Translation in much more detail later.

(9) Problems with Vanishing and Exploding Gradients

Backpropagation for RNNs

ex) 네번째 지점의 손실 $J^{(4)}$ 에 대한 $h^{(1)}$ 의 gradient는 어떻게 계산될까?



Chain rule에 의해 발생

>

$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

→ i번째 시점에서의 손실 $J^{(i)}$ 에 대한 $h^{(j)}$ 의 gradient는?

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \boxed{\prod_{j < t \leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}}}$$



(9)-1. Vanishing Gradient

t번째 시점에서의 hidden state의 정의를 이용해서

t번째 hidden state에 대한 t-1번째 hidden state의 gradient를 정의하면 다음과 같다.

- Recall: $h^{(t)} = \sigma(W_h h^{(t-1)} + W_x x^{(t)} + b_1)$
- Therefore: $\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag}(\sigma'(W_h h^{(t-1)} + W_x x^{(t)} + b_1)) W_h$ (chain rule)

위 식을 대입하면, 아래와 같이 그래디언트를 일반화할 수 있습니다.

$$\begin{aligned} \frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \boxed{\prod_{j < t \leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}}} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \boxed{W_h^{(i-j)}} \prod_{j < t \leq i} \text{diag}(\sigma' (W_h h^{(t-1)} + W_x x^{(t)} + b_1)) \end{aligned}$$

- Norm의 성질에 의해, 다음 부등식 성립! $\Rightarrow W_h$ 의 L2 norm은 W_h 의 가장 큰 고유값(eigenvalue)이다!

$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \right\| \|W_h\|^{(i-j)} \prod_{j < t \leq i} \left\| \text{diag}(\sigma' (W_h h^{(t-1)} + W_x x^{(t)} + b_1)) \right\|$$

$$\Rightarrow \|W_h\|_2 = \sqrt{\lambda_{max}}$$

- RNN 역전파시 chain rule에 의해 Hidden state의 Gradient를 지속적으로 곱해주어야 한다.
- 하지만 부등식을 통해, 손실에 대한 hidden state의 gradient의 L2 norm(크기)는 절대적으로 W_h 의 L2 norm(크기)에 달려있다는 것을 확인할 수 있다.
- 따라서 W_h 의 L2 norm값 (= W_h 의 가장 큰 고유값)이 1보다 작다면, 1보다 작은 값이 계속해서 곱해지는 것이기 때문에 gradient가 빠르게 사라져버리는 문제가 발생! 이것이 바로 **Vanishing gradient problem**이다.
- 반대로 W_h 의 L2 norm값 (= W_h 의 가장 큰 고유값)이 1보다 크다면, 1보다 큰 값이 계속해서 곱해지는 것이기 때문에 gradient가 폭발적으로 증가해버리는 문제가 발생! 이것이 바로 **Exploding gradient problem**이다.

？ 그렇다면 Vanishing gradient가 발생하는 이유는 이해했는데, 그게 왜 문제가 되는 걸까? ?

(9)-2. Why is vanishing gradient a problem?

모델이 가까이 위치한 dependency에 맞게 학습을 하고, 멀리 떨어진 dependency에 대해서는 학습을 하지 못하게 됨.

- Near-effects만 반영되고, Long-term effects는 무시하게 되는것!
- ex) 만약 연속해서 곱해지는 모든 미분값들이 1보다 작은 값이라면, 현재 셀과 거리가 먼 셀들은 1보다 작은 값이 연속해서 곱해져서 결국 0과 가까운 값이 됨 —> 거리가 먼 셀들은 현재 셀에 거의 영향을 주지 못함.

(9)-3. Effect of vanishing gradient on RNN-LM

Vanishing Gradient가 실제 Language Model에서 발생시키는 문제들을 확인해보겠습니다.

ex1)

LM task: When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her tickets

- 빈칸에 들어가야할 단어는 첫 줄에 나왔던 tickets 이다.

- 하지만 첫줄에 나온 tickets과 빈칸 사이에는 많은 Sequence들이 존재하기 때문에 Gradient가 소실될 가능성이 큼.
→ LM Model은 멀리 있는 단어 간의 의존성을 학습하지 못하기 때문에 ticket 예측에 실패하게 됨.

(9)-4. Why is exploding gradient a problem?

- 연속해서 곱해지는 모든 미분값들이 1보다 큰 경우 gradient가 폭발적으로 증가해버리는 문제가 발생!
- → 손실값은 커지고 이상한 값으로 weight 값이 업데이트 된다.

(9)-5. Gradient clipping: solution for exploding gradient

- exploding gradient problem은 Gradient clipping을 통해 해결.
- gradient가 일정 threshold를 넘어가면 gradient값의 L2 norm값으로 나눠주는 방식
- 쉽게 말해, 파라미터를 update할 때, gradient가 정상적인 값보다 너무 크다고 판단되었을 때, scale down을 해주는 방법

Algorithm 1 Pseudo-code for norm clipping

```

 $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq \text{threshold}$  then
     $\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$ 
end if

```

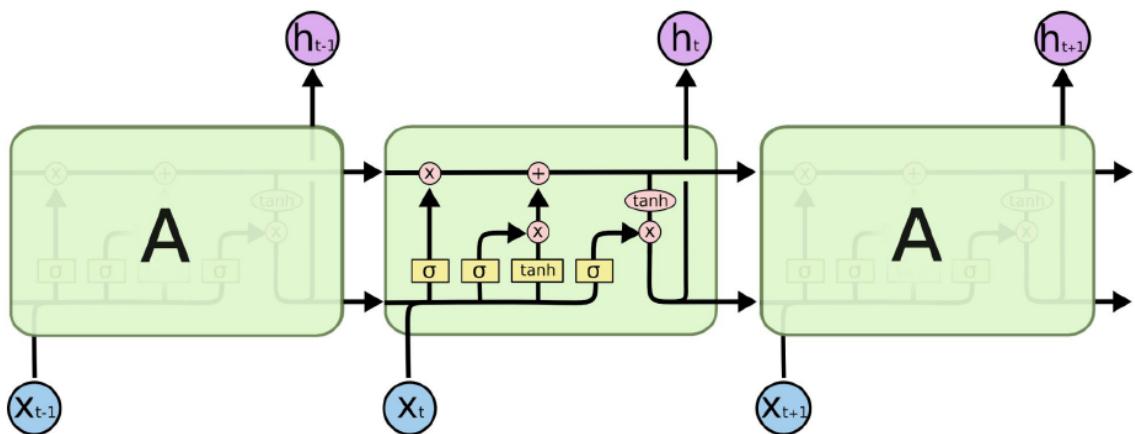
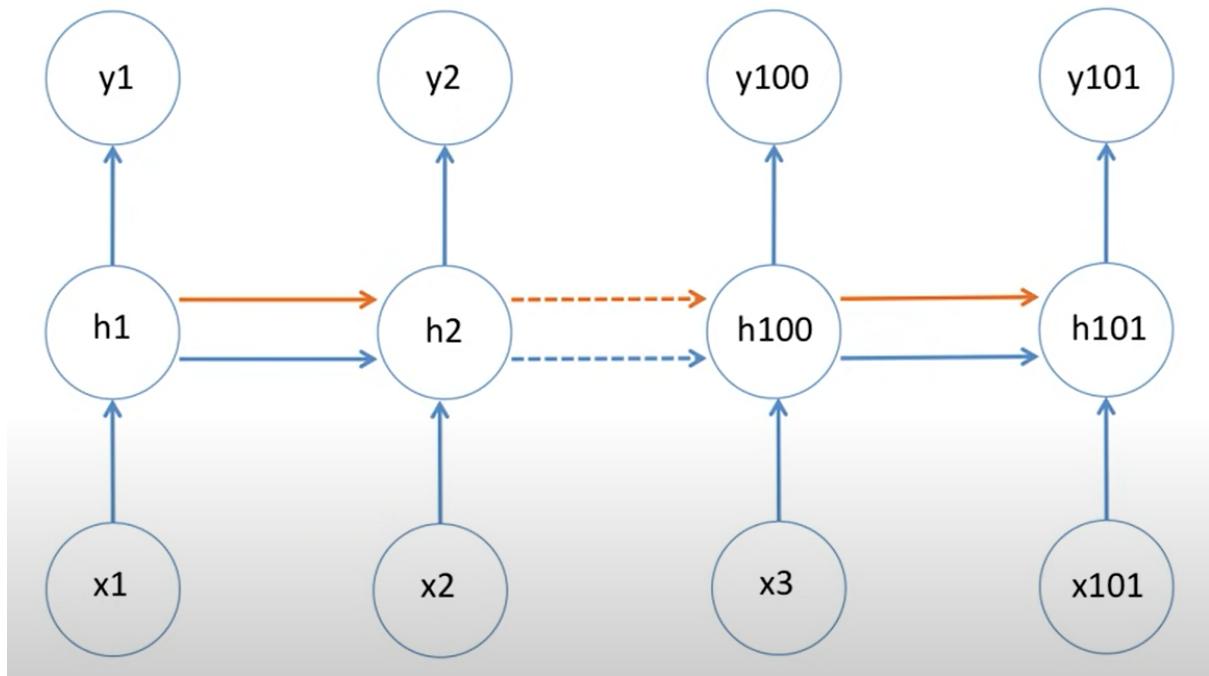
? 그렇다면 Vanishing gradient 문제를 해결할 수 있는 방법은? ?

3-4. Long Short-Term Memory RNNs (LSTMs)

LSTM의 Main idea

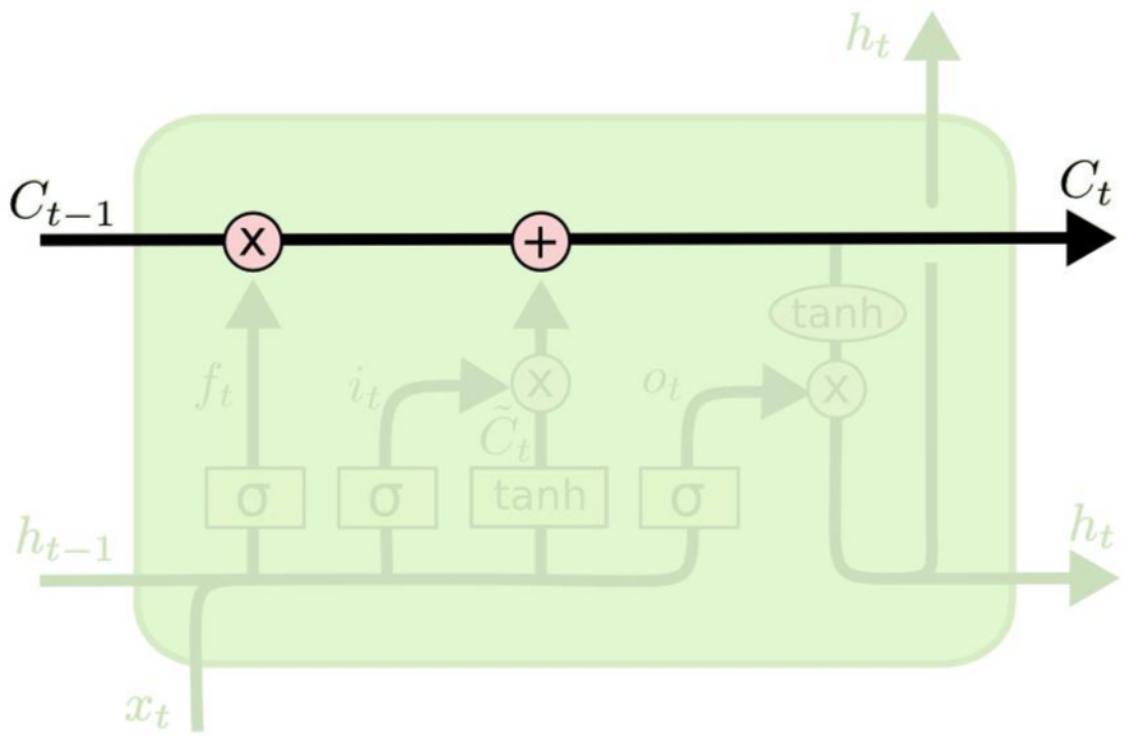
" Gradient가 레이어를 지날수록 0에 가까워지는게 문제라면(vanishing gradient problem), 정보를 Save하는 메모리를 따로둬서 Gradient를 살릴 수 있지 않을까? "

→ 메모리셀 추가(주황색 선)



- 현재 시점의 정보를 바탕으로 **과거의 내용을 얼마나 잊을 지 곱해주고, 그 결과에 현재 정보를 더해서** 다음 시점으로 정보를 전달하는 것

(1) Cell state

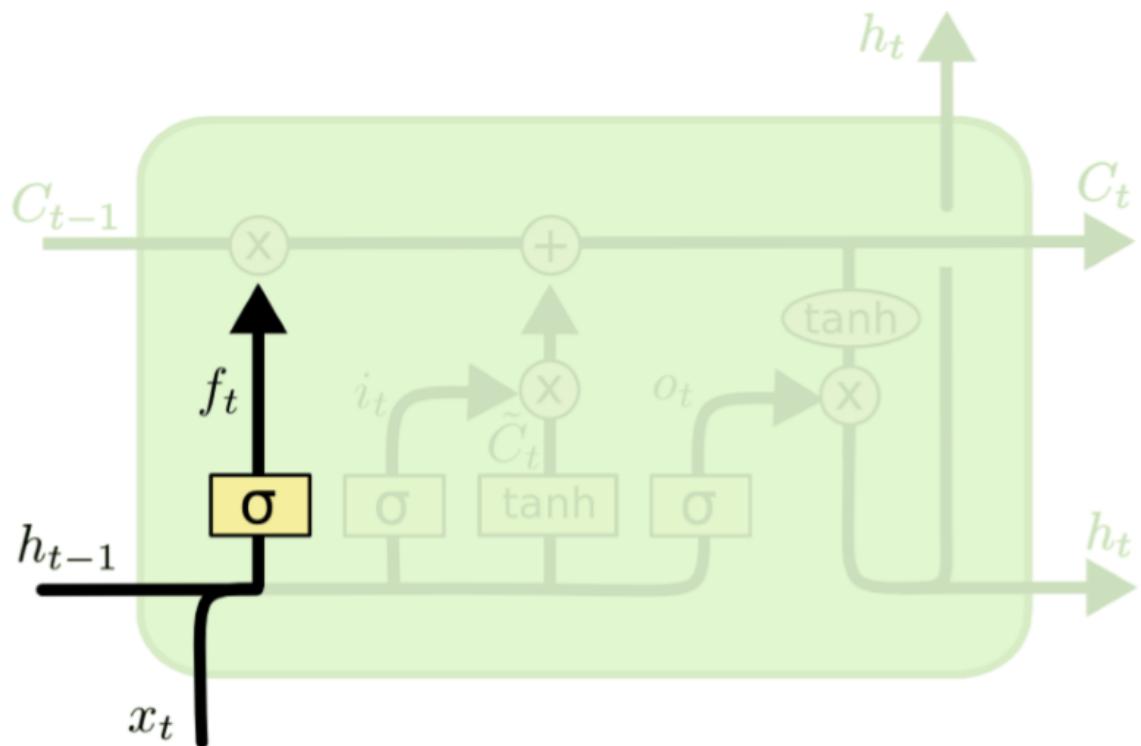


- LSTM의 핵심적인 부분
- 이전 단계의 정보를 memory cell에 저장해서 흘려보내는 것
- cell state는 input, forget, output 세 개의 gate들을 이용하여 정보의 반영여부를 결정

각각 세개의 gate는 어느 정보를 쓰고, 읽고, 잊을 것인지를 결정하는데, 이를 단계별로 살펴보겠습니다.

(2) LSTM 과정

1) Forget gate layer

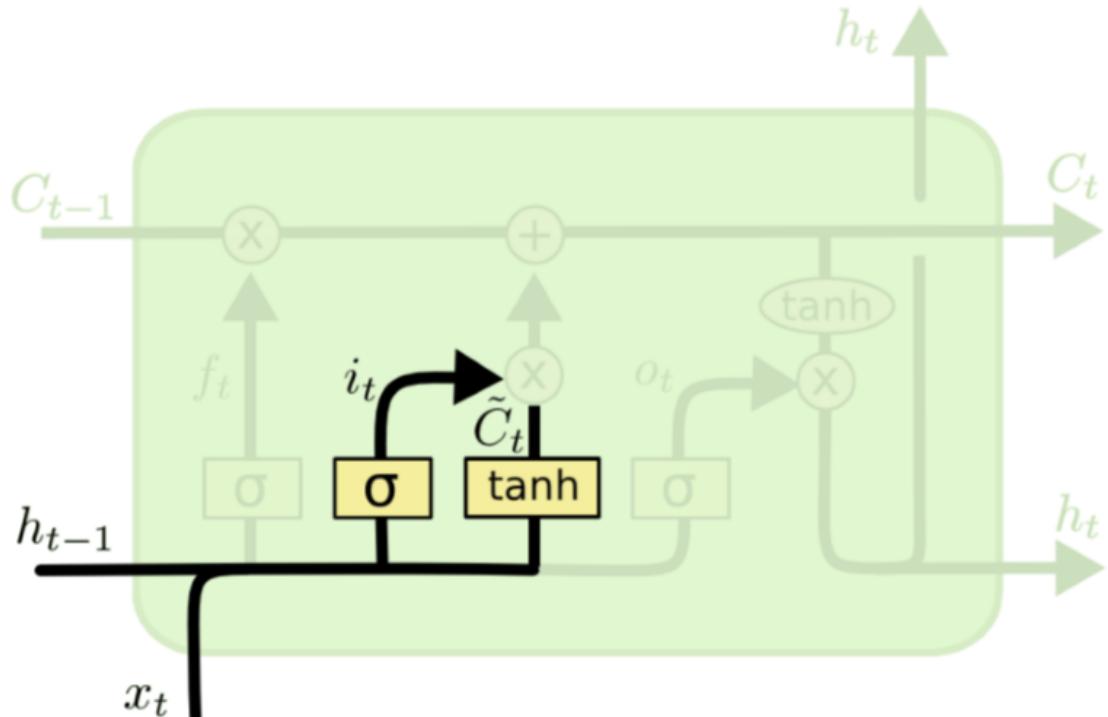


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

forget gate : 어떤 정보를 잊고 어떤 정보를 기억할지에 대한 결정을 하는 gate

- t번째 시점에서의 x 값과 t-1시점에서의 hidden state를 입력값으로 받아 sigmoid activation function을 통해 0에서 1사이의 값을 출력합니다.
- 출력값 = 1 → 과거의 메모리셀 기억 그대로 유지
출력값 = 0.8 → 과거 메모리셀 기억 80%만 기억, 나머지 20%의 기억은 소멸.
출력값 = 0 → 과거의 메모리셀 기억 모두 소멸.

2) input gate layer



$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

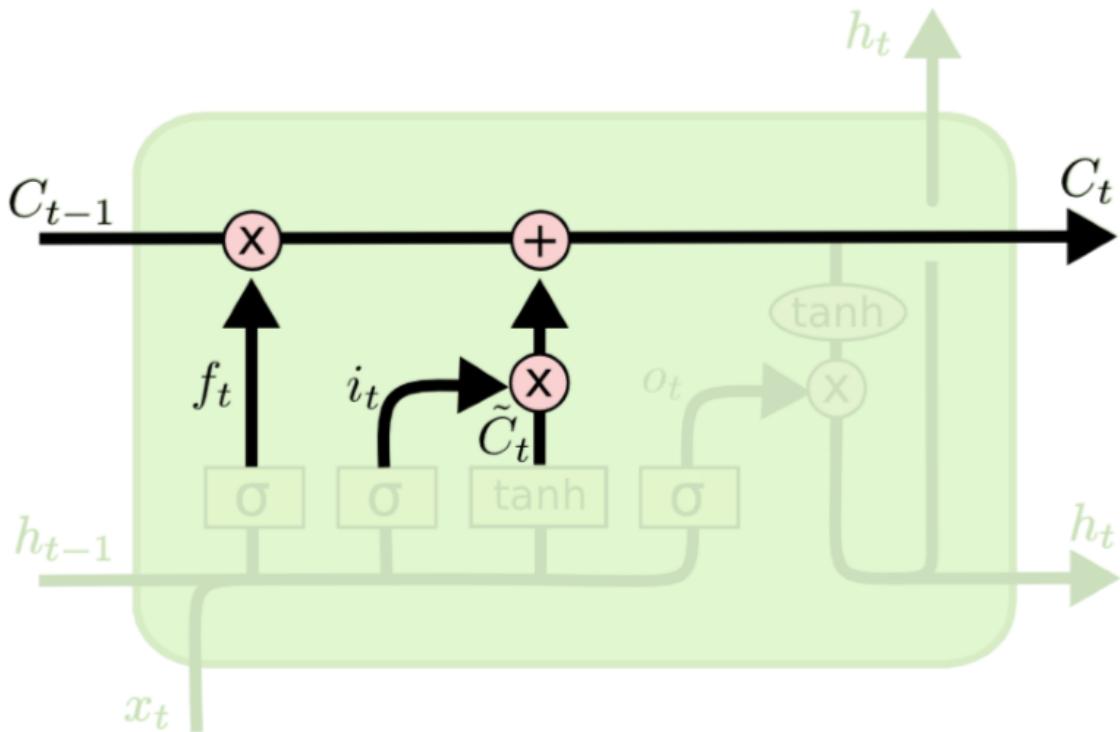
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

input gate: 새로운 정보가 cell state에 얼마나 저장될지를 결정하는 gate

- forget gate와 마찬가지로, input으로 $h_{(t-1)}$ 과 x_t 를 받습니다.
- C_t : \tanh 함수를 사용하여 새로운 정보 생성
- i_t : sigmoid 함수에 의해 0에서 1사이의 값으로 출력하는 부분
--> 현재의 정보를 어느정도 반영할지를 나타내는 값.

C_t 와 i_t 의 output값이 곱해짐으로써 새로운 정보를 반영할 것인지를 결정

3) Update Cell state



$$C_t = \boxed{f_t * C_{t-1}} + \boxed{i_t * \tilde{C}_t}$$

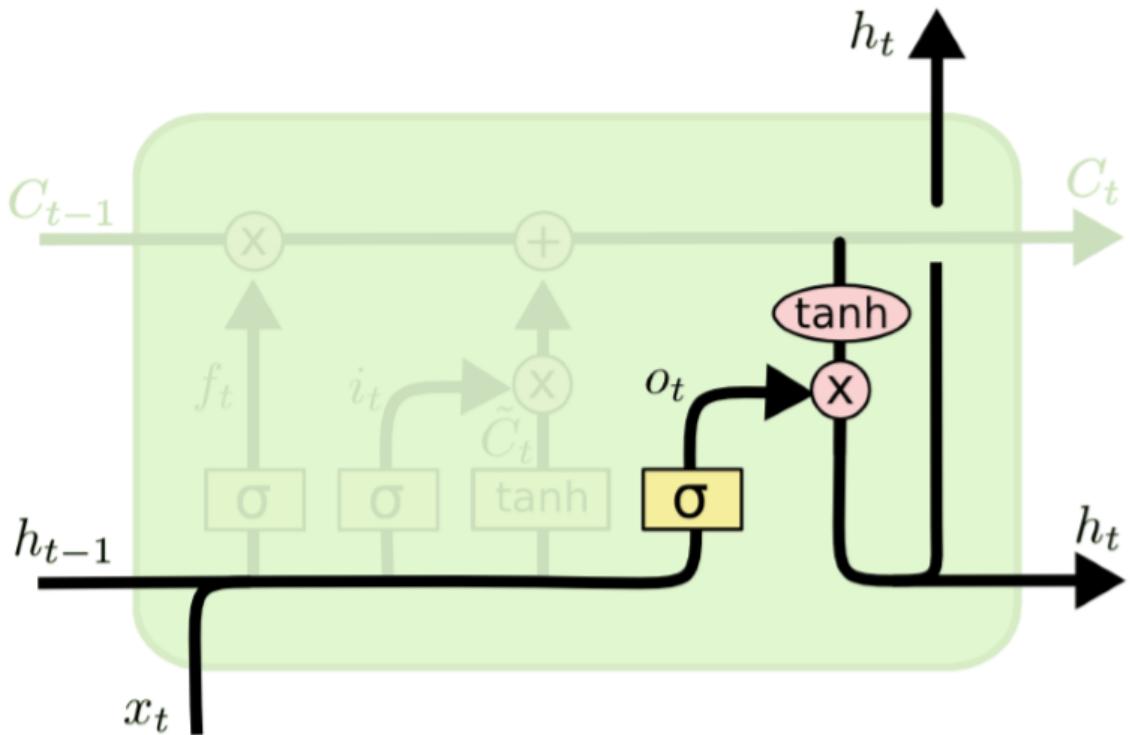
과거 현재

최종적으로

- $f_t * C_{t-1}$: 과거의 기억을 얼마나 반영할지, **forget gate**를 통해 결정됨.
- $i_t * \tilde{C}_t$: 현재의 값을 얼마나 반영할지, **input gate**를 통해 결정됨.

이 두 값이 더해져서 다음 cell state의 입력값으로 들어가게 된다.(Update)

4) Output Gate Layer



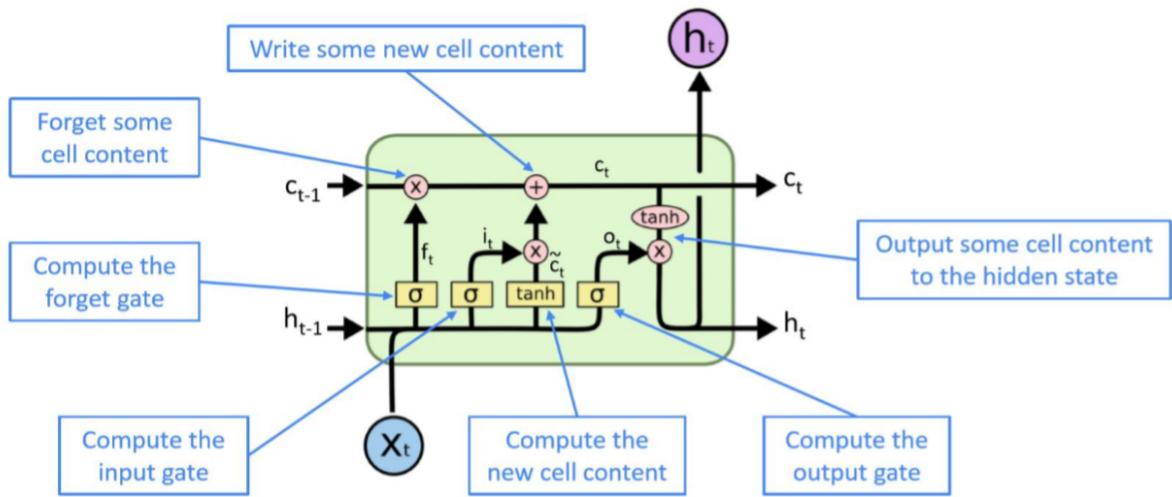
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

마지막으로, 출력값을 반환하는 Output gate가 존재하고, 최종 output은 cell state를 바탕으로 가공된 값이 됩니다.

1. o_t : sigmoid 함수에 input들이 들어가 0에서 1사이의 값을 출력
--> cell state의 어느 부분을 output으로 내보낼지를 결정
2. $\tanh(C_t)$: Cell State 정보를 tanh를 통해 가공
3. h_t : o_t 와 $\tanh(C_t)$ 를 곱함으로써 가공된 cell state 정보를 다음 hidden State에 얼마나 반영할지 결정. --> 최종 output 값 & 다음 state의 input 값

5) 과정 요약



- 모든 state와 gate는 길이가 n인 벡터이고,
- 모든 gate는 sigmoid를 통해서 0과 1사이의 숫자로 나오고,
- 전 step의 hidden state와 현재 input context를 기반으로 계산되므로 dynamic한 모델입니다.

(3) How does LSTM solve vanishing gradients?

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)} \quad h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

- 만약 forget gate가 1이고 input gate가 0이라면, t시점에서의 cell state는 이전의 정보가 완전히 보존되는 채로 hidden state를 update하기 때문에 cell의 정보가 완전하게 보존됨.
→ 장기 의존성 문제 해결!
- vanishing/exploding gradient 문제가 아예 없다고 보장할 수는 없다.

(4) Is vanishing/exploding gradient just a RNN problem?

Q : RNN에서의 gradient 문제는 RNN만의 문제일까?

A : No! 이 문제는 feed-forward, convolutional을 포함한 모든 NN에서의 문제!

- chain rule과 비선형 함수의 선택으로 인해, gradient 값은 backpropagation을 할 때 점점 작아지게 된다.
- → lower layer에서는 update가 잘 되지 않아 학습하기 어려운 문제가 발생

gradient vanishing 문제를 해결하는 몇몇 방법을 살펴보자

1) Residual connections "ResNet" (= skip-connections)

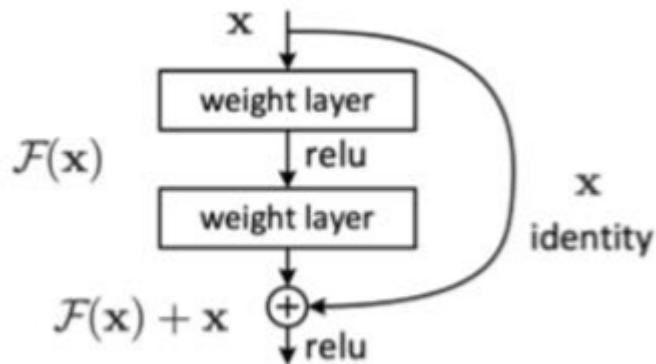


Figure 2. Residual learning: a building block.

- input x에 convolutional layer를 지나고 나온 결과를 더해줌으로써, 과거의 내용을 기억할 수 있도록 함.
- 과거의 학습 내용 보존 + 추가적으로 학습하는 정보 => gradient 사라지는 문제 해결

2) Dense connections "DenseNet"

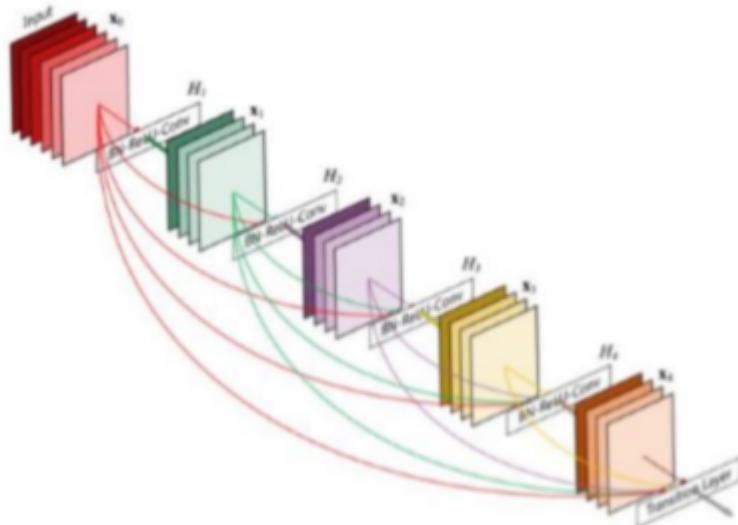


Figure 1: A 5-layer dense block with a growth rate of $k = 4$.
Each layer takes all preceding feature-maps as input.

- 이전 layer들의 feature map을 계속해서 다음 layer의 입력과 연결하는 방식

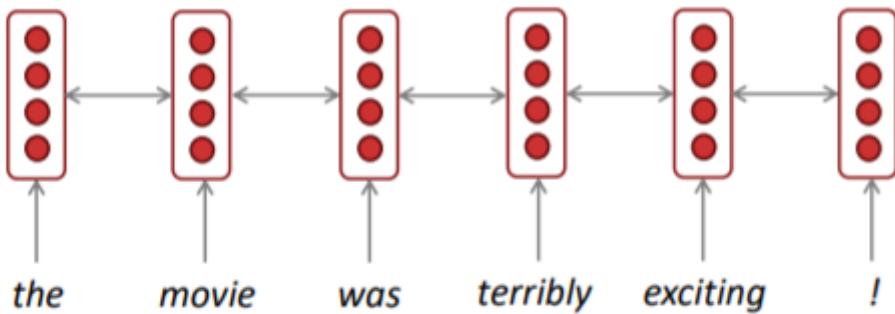
3) Highway connections "HighwayNet"

- Residual connections과 비슷
- LSTM에서 영감을 받은 모델

결론 : Vanishing gradient 문제는 여러 분야에서 매우 general한 문제이지만, RNN과 같이 동일한 weight matrix를 반복적으로 곱하는 모델은 특히 더 불안정하므로, 더욱 심각한 문제이다.

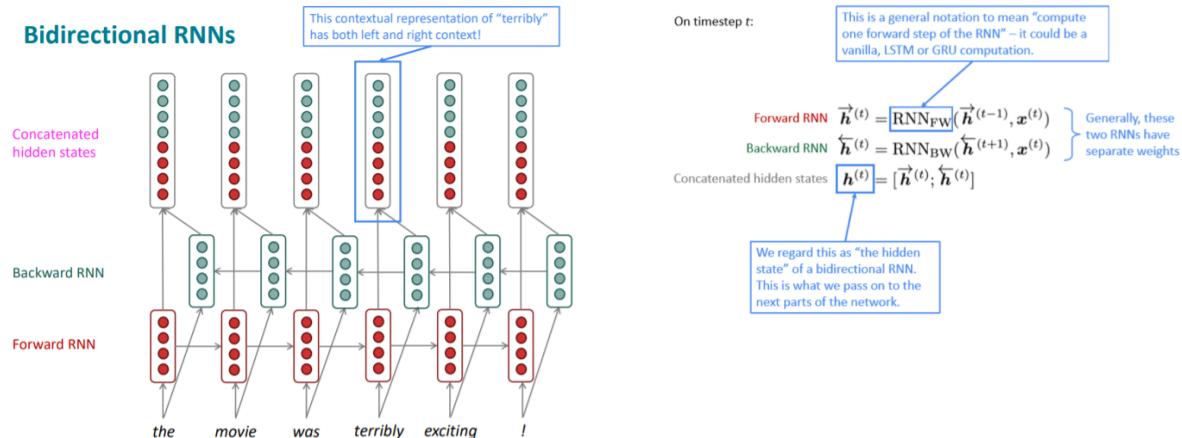
(5) Bidirectional and Multi-layer RNNs : motivation

- **Bidirectional RNNs :** 양 방향 정보를 모두 이용한 RNN 구조
- 기존의 RNN은 forward 방향으로만 학습을 진행.
→ but 이는 과거의 상태에만 의존하여 미래를 예측하기에 다소 오류 발생.
→ 이를 해결하고자 backward 방향의 학습도 진행하여 미래의 상태까지 고려하는 모델을 만듦. 이것이 바로 Bidirectional RNNs



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

- **Bidirectional RNNs 과정**



- 1) forward RNN : 정방향으로 입력받아 hidden state 생성
- 2) backward RNN: 역방향으로 입력받아 hidden state 생성
- 3) 두 개의 hidden state를 연결해서 전체 모델의 hidden state로 사용

- ex) BERT(Bidirectional Encoder Representations from Transformers)는 bidirectionality를 기반으로 만들어진 강력한 contextual representation이다.