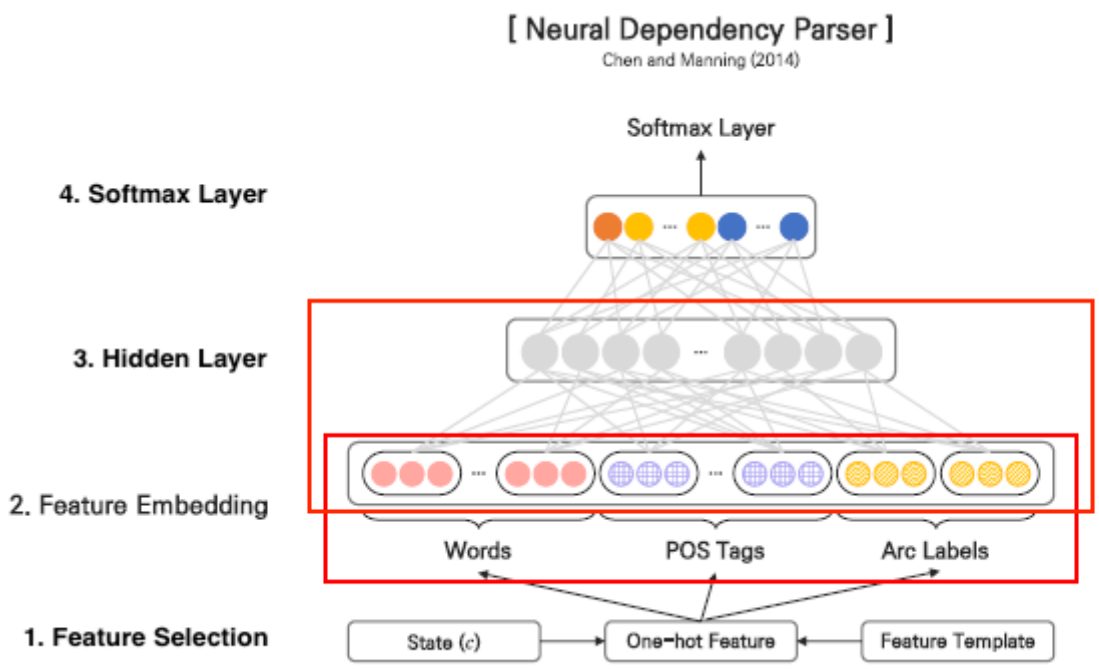
Lecture 5: Recurrent Neural Networks (RNNs)

neural dependency parser

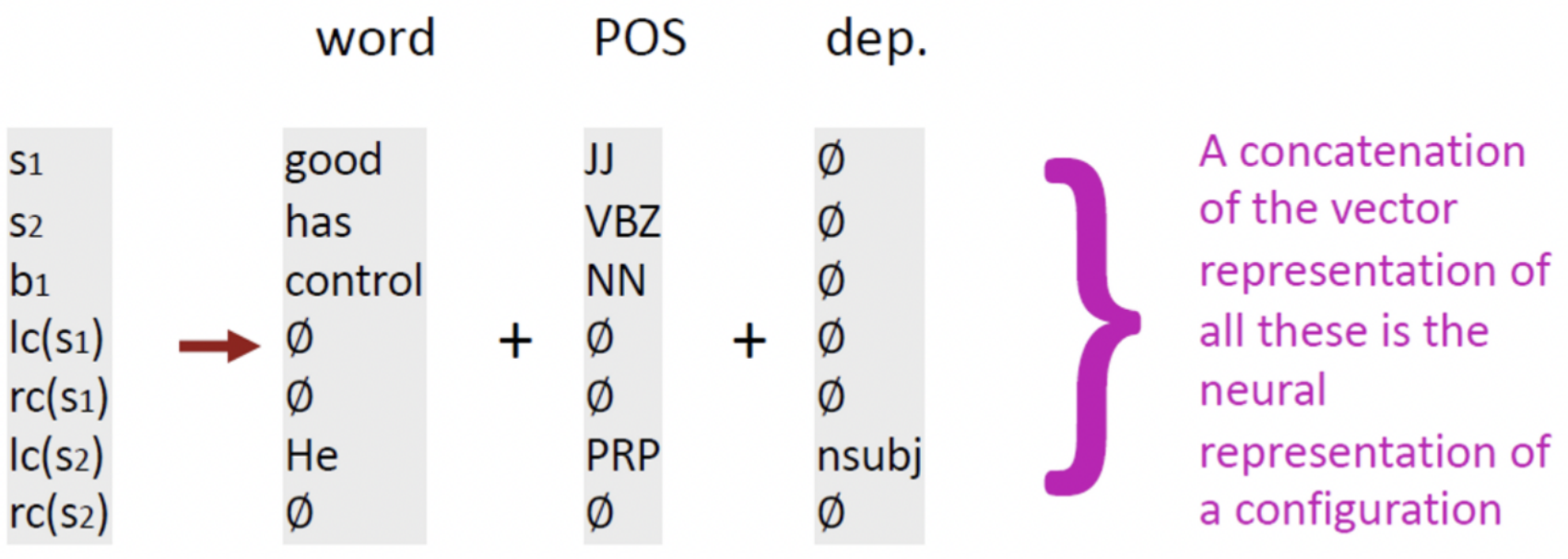
input: words, POS 태그, arc labels (또는 dep)



1. 각 feature별로 임베딩된 벡터를 input layer에 입력
2. 은닉층에서 임베딩 벡터와 가중치 행렬 곱한 뒤 bias 행렬 더해줌 (feed forward 계산)
3. 활성화 함수로 cube function 사용 (word, POS 태그, arc label간 상호작용 반영)
4. softmax 함수 적용하여 shift, left-arc, right-arc 중 확률이 가장 높은 요소를 output으로 산출

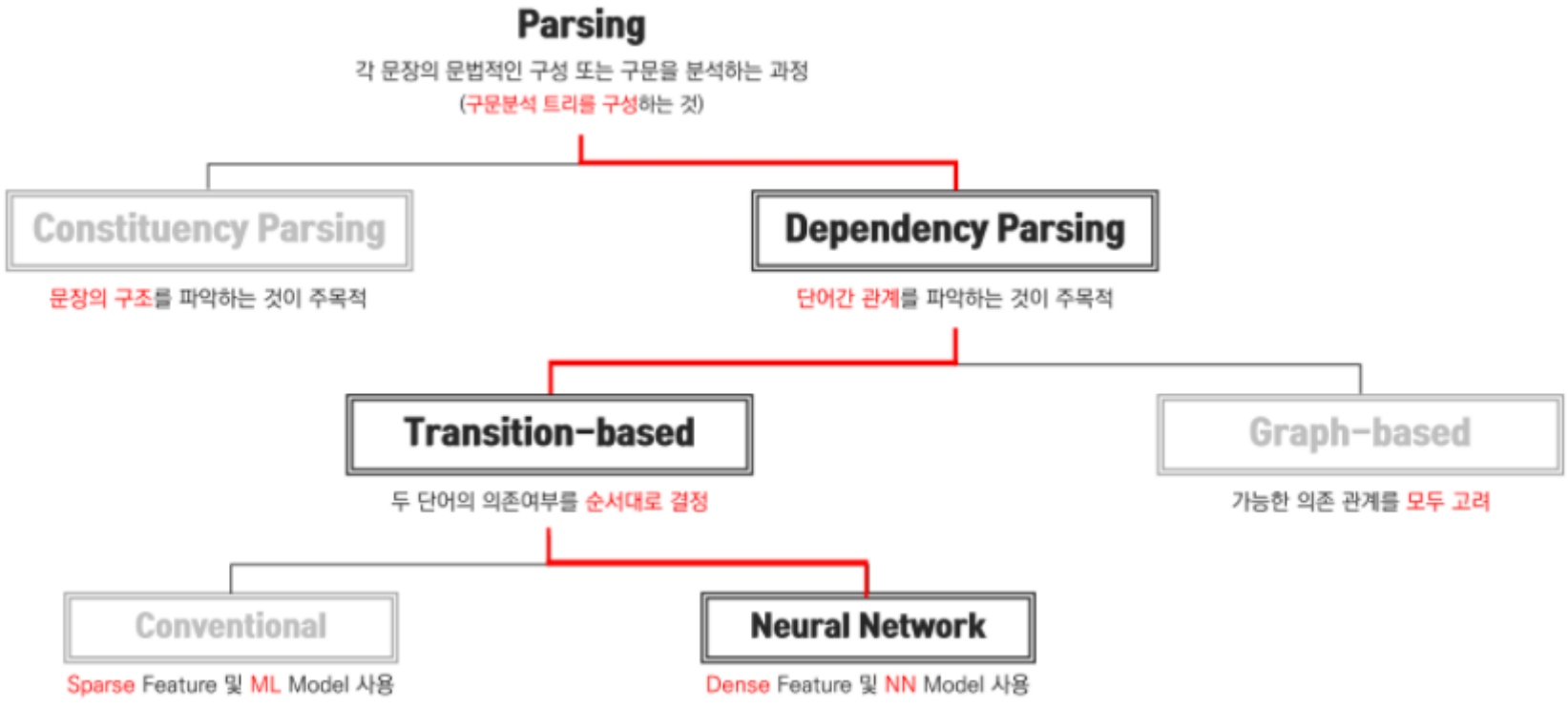
▼ 특징

1. distributed representation
   - 모든 word는 (word, POS, dep) 3가지 특성을 가짐
   - 3가지 특성을 모은 것이 (concatenated vector) input



2. 비선형 분류기
   - input이 은닉층을 거치면서 선형 분류기로 분류될 수 있도록 re-represent
   - softmax 함수를 통해 shift, left-arc, right-arc 3개 클래스로 분류

▼ parsing 요약



Neural Networks 세부사항

▼ regularization

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

목적: (1) 과대적합 방지, (2) 모델의 일반화 성능 향상

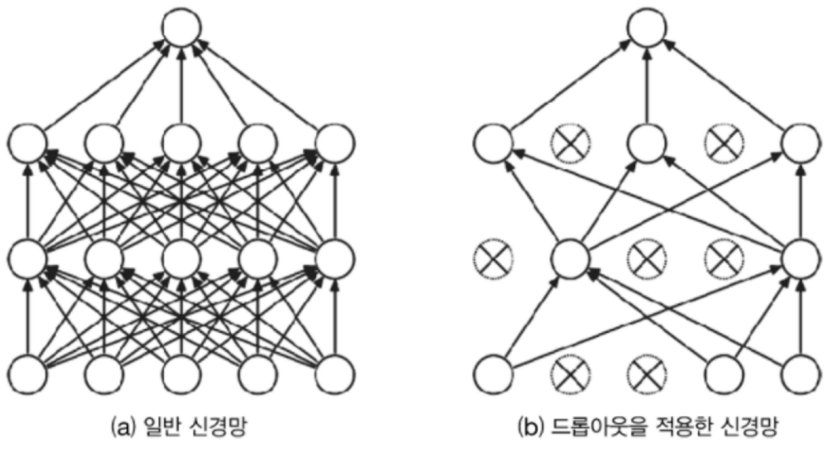
방법: loss function $J(\theta)$ 최소화 → regularization term 최소화

효과: 파라미터가 유의미한 경우에만 non-zero가 되도록 함

예) L2 regularization: 모든 파라미터의 제곱합 * regularization 파라미터 λ

- λ: 규제의 정도

▼ dropout



목적: 특정 feature만 과도하게 학습되는 현상 방지

방법:

1. train 단계에서 해당 층의 뉴런 중 일정 비율만 다음 층으로 전달 (일정 비율 = dropout rate)
2. test 단계에서는 dropout rate만큼 model weight 줄임

효과:

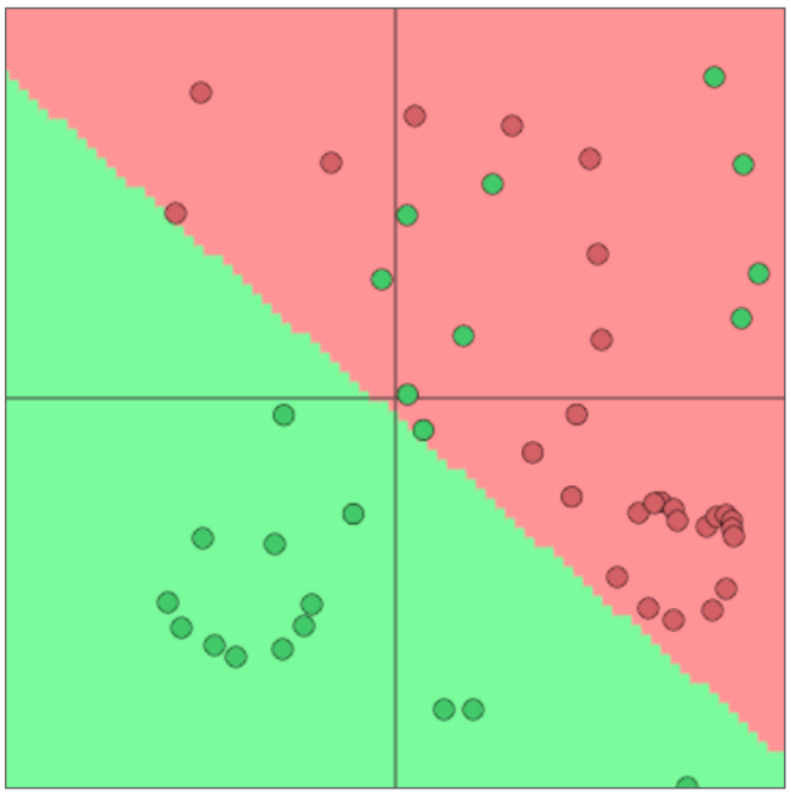
1. feature co-adaption 방지: 특정 feature의 영향이 커지면서 다른 feature의 학습에 영향을 주는 현상 방지
2. feature-dependent regularization: feature마다 규제 정도 상이 (영향력이 적은 feature일수록 규제 정도 ↑)
3. model bagging 효과: ‘무작위’ dropout으로 인한 앙상블 모델 효과

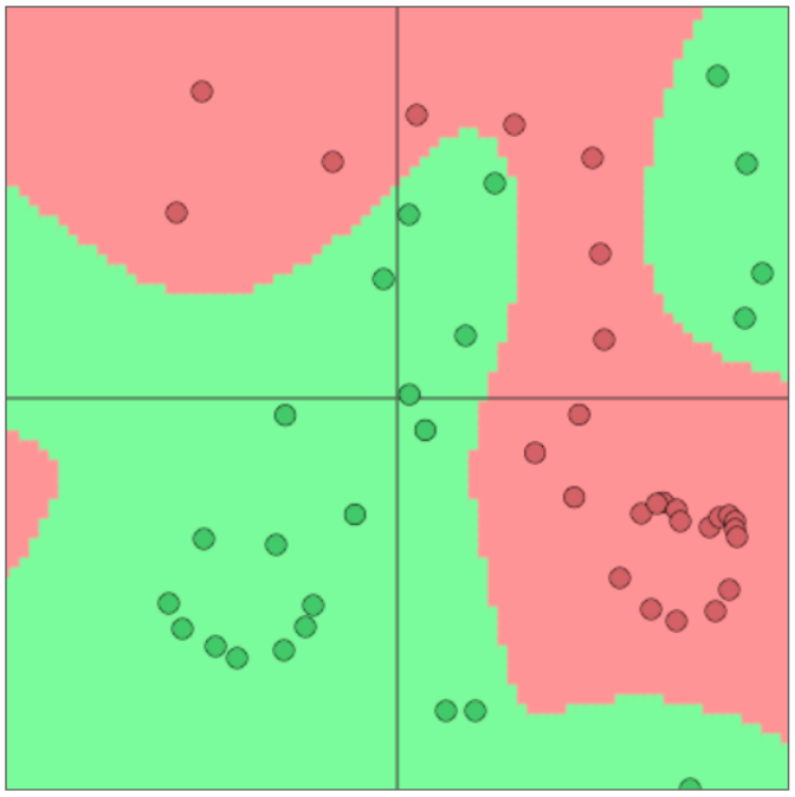
▼ vectorization

계산 속도 향상을 위해 loop 대신 벡터와 행렬 이용!

▼ activation function

목적: 비선형성을 부여하기 위함

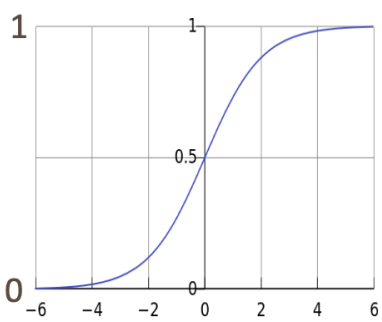




여러 활성함수:

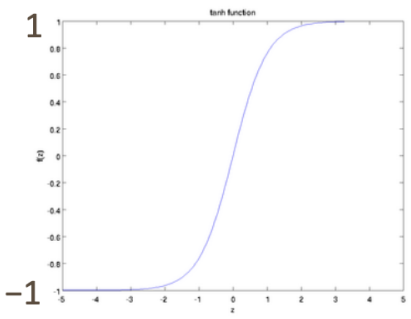
logistic (“sigmoid”)

$$f(z) = \frac{1}{1+\exp(-z)}.$$



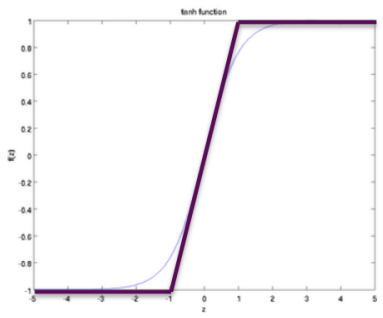
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



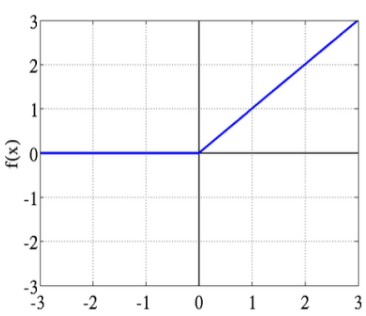
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



ReLU (Rectified Linear Unit)

$$\text{rect}(z) = \max(z, 0)$$



- sigmoid 함수: (1) 출력값이 양수 (→ tanh 함수), (2) 입력값이 양극단으로 갈수록 출력값이 특정 값으로 수렴 (→ ReLU 함수), (3) 중심값이 0이 아님 (→ tanh 함수)
- tanh 함수: (1) 입력값이 양극단으로 갈수록 출력값이 특정 값으로 수렴 (→ ReLU 함수), (2) 계산 속도 느림 (→ hard tanh 함수)
- ReLU 함수가 통상적으로 자주 쓰임 (변형으로 leaky ReLU)
- sigmoid 함수는 확률 구할 때, tanh 함수는 RNN에서 자주 활용

▼ parameter initialization

목적: gradient vanishing, gradient exploding 방지

방법: 파라미터를 작은 랜덤값으로 초기화 (zero initialization, 대칭적 가중치 초기화는 부적절)

▼ zero initialization이 부적절한 이유

- 동일한 feature만 학습하게 됨
- 입력의 가중치가 모두 0이라면 그 다음 층에도 똑같은 값이 전달됨 → 역전파 때 모든 가중치의 값이 같아짐

▼ 대칭적 가중치 초기화가 부적절한 이유

5.2.2 가중치 초기화

- 대칭적 가중치 문제
  - [그림 5-8]의 대칭적 가중치에서는 z_1^{l-1}과 z_2^{l-1}가 같은 값이 됨. $-\delta_j z_i$가 그레이디언트기 때문에 u_{11}^l과 u_{12}^l이 같은 값으로 갱신됨 → 두 노드가 같은 일을 하는 중복성 발생
  - 난수로 초기화함으로써 대칭 파괴

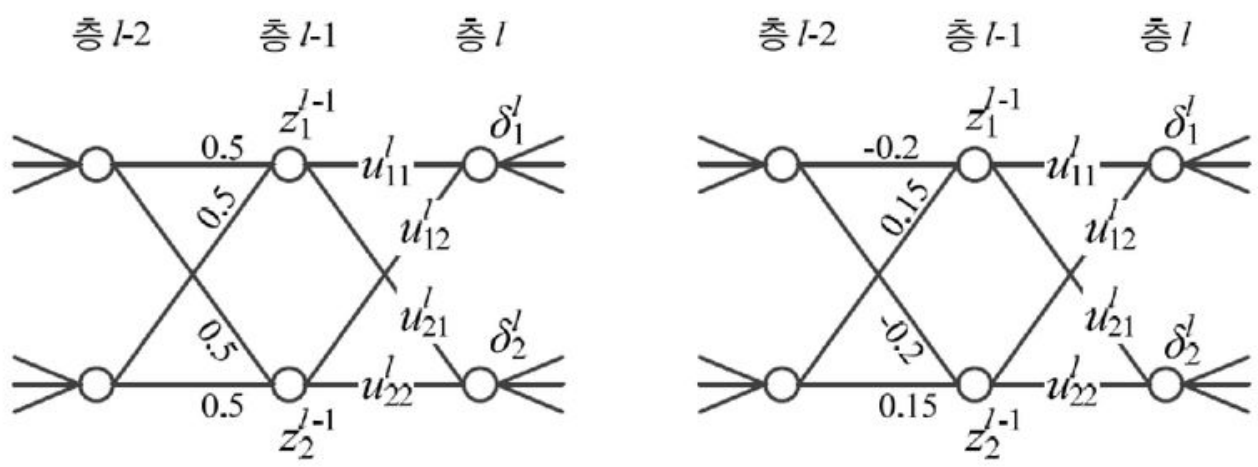


그림 5-8 대칭적 가중치로 초기화된 경우의 중복성 문제

예) Xavier initialization: 노드가 n개일 때 표준편차가 $\frac{1}{\sqrt{n}}$인 정규분포로 초기화

▼ Xavier initialization 보충 설명

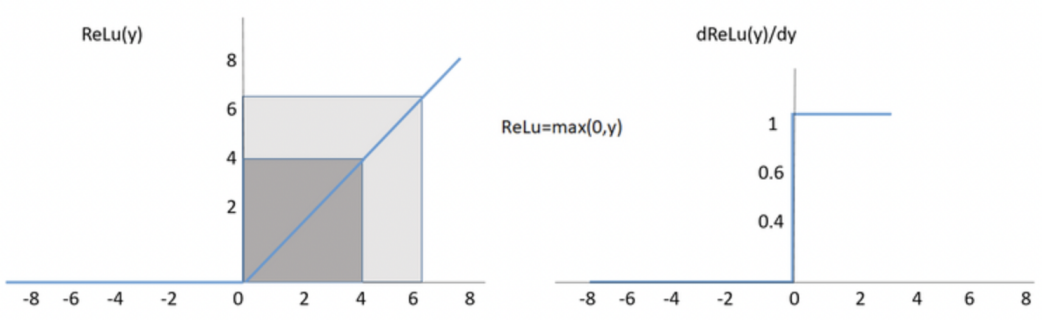
- 노드의 개수가 많을수록 가중치가 좁은 범위 내에서 초기화

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

(n_{in}과 n_{out}가 같을 때 위 식에 루트를 씌우면 앞서 설명한 것과 같은 결과 도출)

- sigmoid 또는 tanh 함수로 활성화된다고 가정하고 초기화

- ReLU 함수 부적절



- 극복 방안: He initialization (활성화 함수로 ReLU 사용 시 적절)

  (노드가 n개일 때 표준편차가 $\frac{2}{\sqrt{n}}$인 정규분포로 초기화)

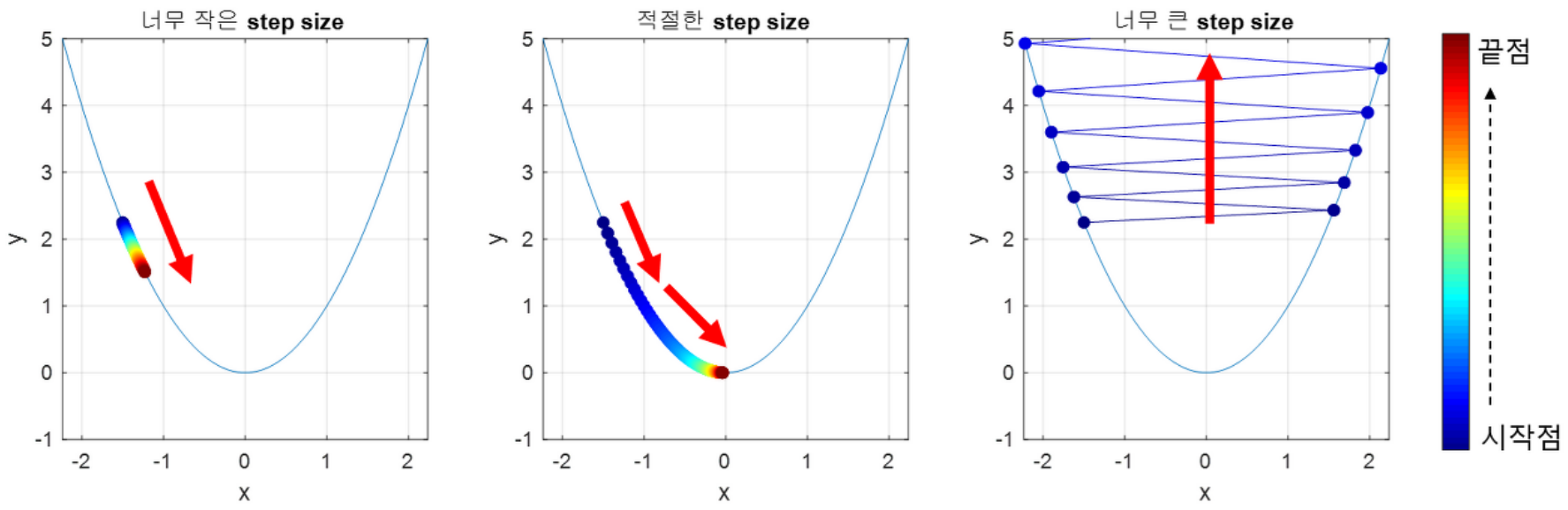
▼ optimizers

adaptive optimizers: accumulated gradient를 통해 각 파라미터별로 최적의 학습률 제공

예) AdaGrad, RMSprop, Adam, SparseAdam 등

▼ learning rates

적절한 learning rate 지정해줘야 함!



방법: 학습이 진행될수록 learning rate 줄이기 등 (k번째 epoch에서 $lr = lr_0 e^{-kt}$)

Language Model

💡 단어 sequence $x^{(1)}, x^{(2)}, ..., x^{(t)}$가 주어졌을 때 다음 단어 $x^{(t+1)}$의 확률 분포

문장 내에서 단어 단위 예측:

$$P(x^{(t+1)}|x^{(t)}, ..., x^{(1)})$$

확장하여 문장 단위 예측:

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

▼ n-gram Language Model

▼ n-gram이란? n개의 연속적인 단어들의 chunk

- unigrams: “the”, “students”, “opened”, “their”
- bigrams: “the students”, “students opened”, “opened their”
- trigrams: “the students opened”, “students opened their”

- 4-grams: “the students opened their”

▼ 가정: $x^{(t+1)}$**은 선행하는** $n-1$**개의 단어에만 의존한다** (Markov assumption)

$$P(x^{(t+1)}|x^{(t)},...,x^{(t-n+2)})$$

조건부 확률의 정의에 따라 다음과 같이 표현:

$$P(x^{(t+1)}|x^{(t)},...,x^{(t-n+2)}) = \frac{P(x^{(t+1)},...,x^{(t-n+2)})}{P(x^{(t)},...,x^{(t-n+2)})}$$

▼ 개념: **n-gram의 빈도**를 기반으로 다음 단어를 예측해보자!

따라서 다음과 같이 근사:

$$\frac{P(x^{(t+1)},...,x^{(t-n+2)})}{P(x^{(t)},...,x^{(t-n+2)})} \approx \frac{count(x^{(t+1)},...,x^{(t-n+2)})}{count(x^{(t)},...,x^{(t-n+2)})}$$

▼ 예) 4-gram language model로 students opened their ( ) 예측

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ students opened their _____

discard

condition on this

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

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
  - → P(books | students opened their) = 0.4
- “students opened their exams” occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the “proctor” context?

31

▼ 문제점

1. sparsity 문제

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then *w* has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

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

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any w*!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

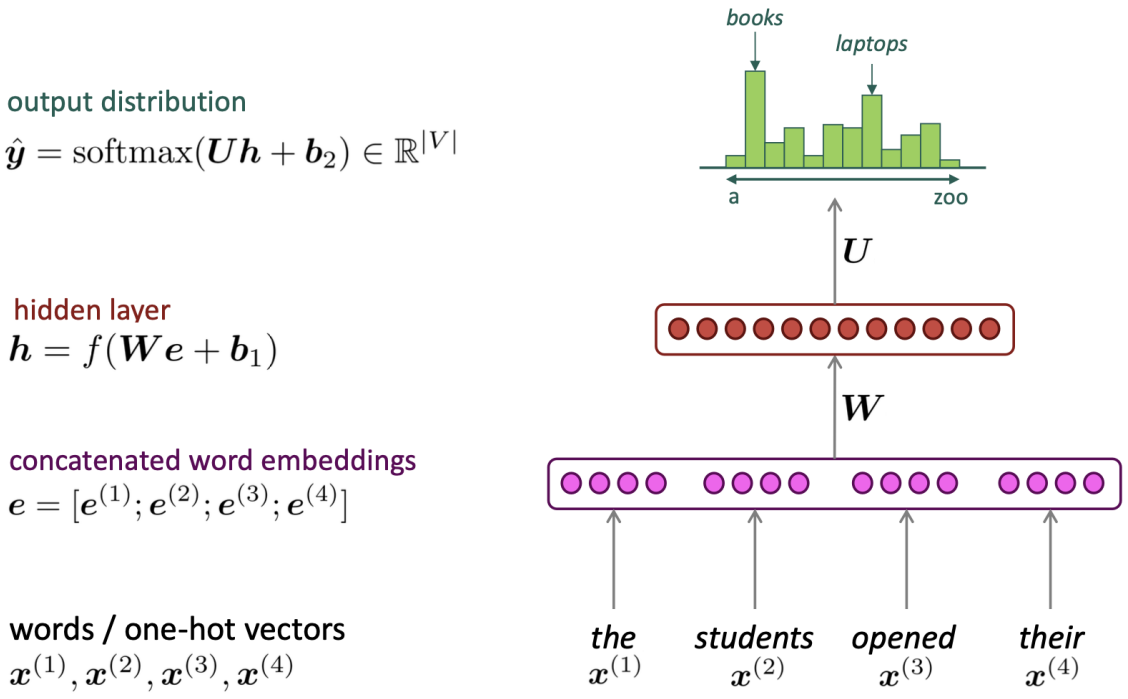
(대체로 n이 커질수록 sparsity 문제 ↑)

2. storage 문제: corpus내 모든 n-gram 저장해야 함
3. 맥락 반영 어려움 (sparsity, storage 문제로 인해 n을 무작정 키울 수도 없음)

💡 fixed-window neural Language Model로 발전!

▼ fixed-window neural Language Model

A fixed-window neural Language Model



sparsity 문제와 storage 문제는 해결되었지만 여전히 문제점 존재

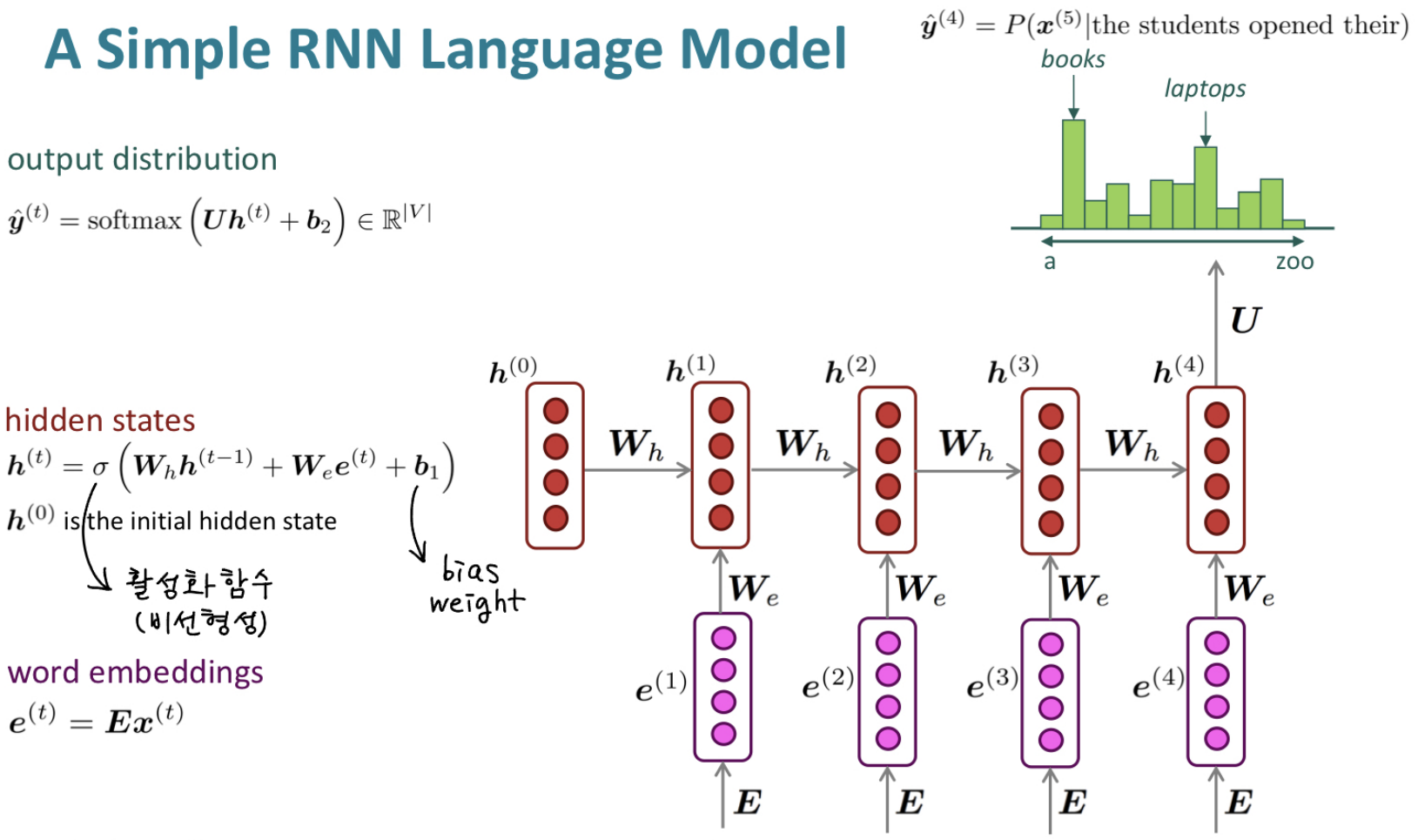
1. fixed window의 크기가 너무 작음
2. fixed window의 크기가 바뀌면 W가 완전히 바뀌기 때문에 새로 학습해야 함

💡 RNN으로 발전!

▼ Recurrent Neural Networks (RNNs)

개념: 같은 **가중치 W를 반복적으로 적용**해보자!

A Simple RNN Language Model



output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

$h^{(0)}$ is the initial hidden state

활성화함수 (비선형성)

bias weight

word embeddings

$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$

***Note**: this input sequence could be much longer now!*

장점:

1. input의 길이와 상관없이 처리 가능 (길이가 길어져도 모델 사이즈가 증가하지 않음)
2. 매번 같은 가중치가 적용되기 때문에 (이론상으로는) 선행하는 여러 단어 고려 용이

단점:

1. 계산 속도 느림
2. 실제로는 선행하는 여러 단어를 고려를 하기 힘듦