5. Language Models and Recurrent Neural Networks

1. Language Model

1-1. Dependency Parsing

기존 Parser의 한계

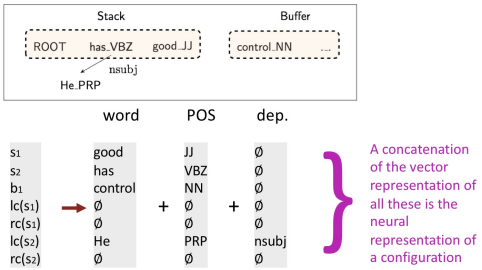
- sparse(희소 행렬이 완성됨)
- incomplete(불완전하다.)
- expensive computation(높은 계산 비용)

Neural Dependency Parser

- 실험 결과

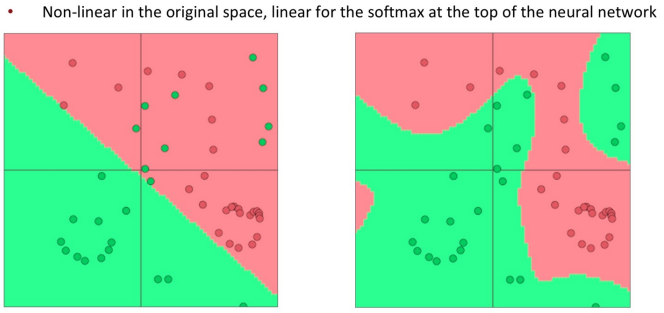
Parser	Unlabeled Attachment Score UAS	Labeled Attachment Score LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3**	89.6	8
C & M 2014	92.0	**89.7**	**654**

- 개선사항
  1. 분산된 표현

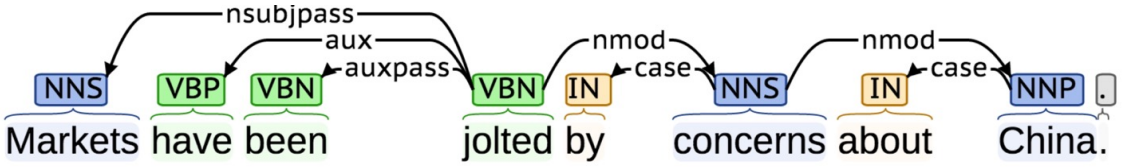


2. Deep Learning에서의 비선형성

- weight matrix W의 neg logloss를 최소화하는 것이 목표
- 기존의 ML 분류기는 오직 선형의 결정 경계만 제공하였음
  - 그러나 복잡한 문제에서는 비선형성이 요구됨

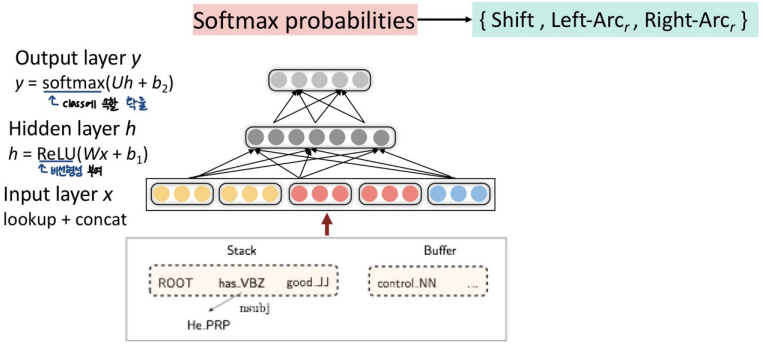


⇒ 이러한 특징들은 정확도와 속도 측면에서 모두 문장 구조를 이해하는 데 도움이 되었음



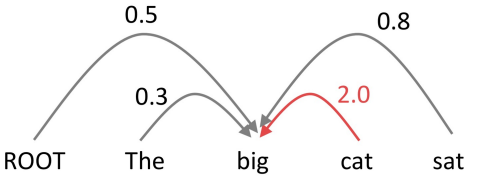
- 구조

Neural Dependency Parser Model Architecture



Graph-based Dependency Parser

- 모든 가능한 의존성에 대해 점수를 계산



e.g., picking the head for “big”

- 굉장히 좋은 성능을 보임

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
Dozat & Manning 2017	**95.74**	**94.08**

- 그러나, 간단한 neural transition-based parser에 비해 속도 측면에서 불리함

  → n^2 개의 가능한 dependencies가 발생하기 때문

  → 대용량 데이터 분석 시 속도의 한계가 있음

1-2. Neural Network에 대한 부가 설명

정규화(Regularization)

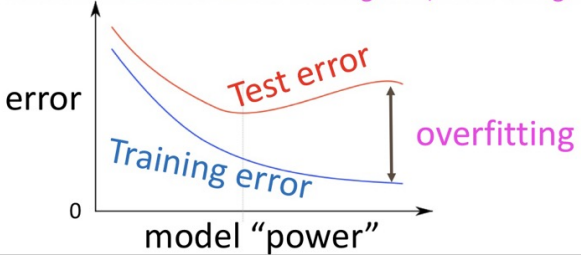
- 과적합(overfitting) 방지
- 모델의 일반화에 도움

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

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

(비용함수가 0이 되는 것 방지; λ: 규제 강도)

- Classic view: Regularization works to prevent overfitting when we have a lot of features (or later a very powerful/deep model, etc.)
- Now: Regularization produces models that generalize well when we have a “big” model
  - We do not care that our models overfit on the training data, even though they are **hugely** overfit



17

Dropout

- feature들 간의 과한 의존(co-adaptation)을 방지

Vectorization

- 자연어를 NN model에서 처리하려면 벡터화가 필요함

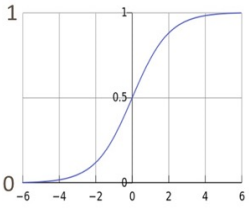
비선형성(Non-linearity)

- 데이터가 복잡한 상황에서 모델이 유연하게 반응할 수 있도록 도와줌

Non-linearities, old and new

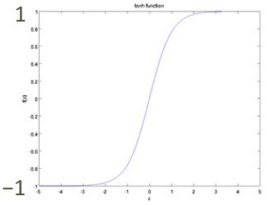
logistic ("sigmoid")

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



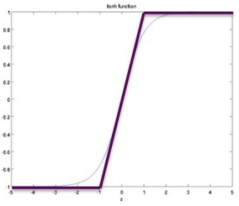
tanh

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



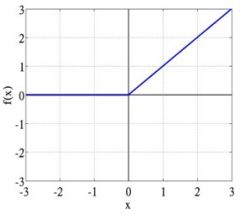
hard tanh

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



ReLU (Rectified Linear Unit)

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



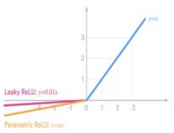
tanh is just a rescaled and shifted sigmoid (2 × as steep, [−1,1]):

$$\tanh(z) = 2\text{logistic}(2z) - 1$$

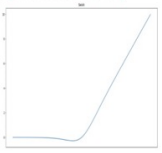
Both logistic and tanh are still used in various places (e.g., to get a probability), but are no longer the defaults for making deep networks

For building a deep network, the first thing you should try is ReLU — it trains quickly and performs well due to good gradient backflow

Leaky ReLU / Parametric ReLU



Swish [Ramachandran, Zoph & Le 2017]



파라미터 초기화

- 중요하다고 한다..그러하다..

옵티마이저(Optimizers)

- 학습률(learning rate) 등을 조절하는 역할
- Adagrad, RMSprop, Adam, SparseAdam 등이 있음

  (무난하게 가려면 `Adam`을 일단 시도해 보는 것도 괜찮다.)

학습률(learning rate)

- 파라미터를 얼만큼 업데이트할 것인가
- 적절한 정도를 찾아야 함
  - 너무 크면 모델이 발산함(최적화 x)
  - 너무 작으면 주어진 실행 시간 동안 충분히 학습되지 못할 가능성이 높음

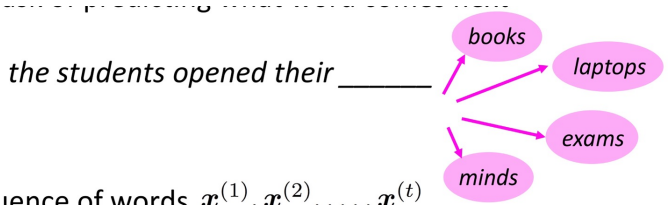
⇒ 일단 조금 크게 시작한 다음 서서히 줄여나가는 것도 방법임

2. RNN Language Model

Language Modeling

- 목표: 어떤 단어가 다음에 올 것인가를 예측

  → 확률 분포를 기반으로 주어진 문맥(sequence) 이후에 위치할 단어를 예측



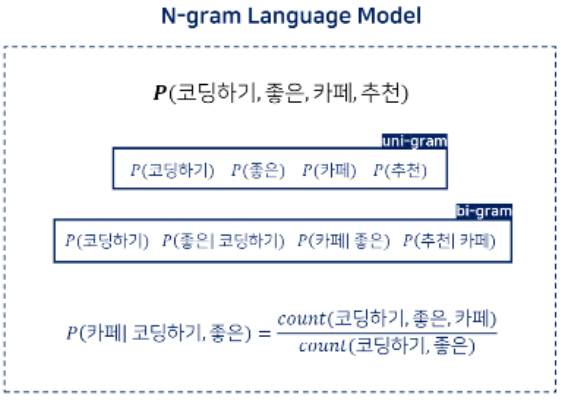
- 시퀀스 결합 확률(by multiplication rule)

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

This is what our LM provides

n-gram LM

- n-gram: n개의 연속적인 단어 묶음
- 이전에 등장한 n개의 단어를 바탕으로 다음에 올 단어를 예측하는 모델



출처: 고려대학교 DSBA 스터디 강의(유튜브)

- 그러나, 입력의 길이가 고정되기에(n, window size) 이전에 등장하는 모든 단어를 고려할 수는 없음
- 단어 예측 과정

- Markov 가정

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

prob of a n-gram

prob of a (n-1)-gram

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

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

n-gram 모델의 한계

1. 희소성 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*.

Note: Increasing *n* makes sparsity problems *worse*. Typically, we can't have *n* bigger than 5.

2. 저장 공간 문제

- 이전의 말뭉치에서 등장한 n-gram에 대한 정보를 계속 가지고 있어야 함

  ⇒ n 증가: 모델의 복잡도 증가

Storage: Need to store count for all *n*-grams you saw in the corpus.

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

Increasing *n* or increasing corpus increases model size!