Lecture 3 -Backprop and Neural Networks <-math of neural network

1. Named Entity Recognition (NER)

The task: find and classify names in text
-tracking mentions of particular entities in documents
-Named Entity Linking/Canonicalization into knowledge base

Named Entity(이름을 가진 개체)를 인식; 유형 인식
[('James', 'NNP'), ('is', 'VBZ'), ('working', 'VBG'), ('at', 'IN'), ('Disney', 'NNP'), ('in', 'IN'), ('London', 'NNP')]

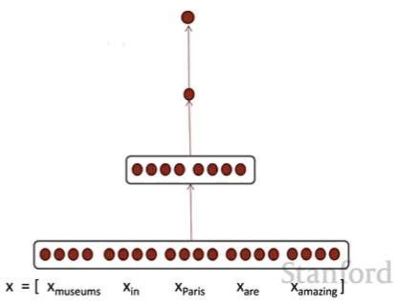
what's NER:
NER is the process of locating and classifying named entities in text into predefined entity categories[1]

1) Simple NER: Window Classification using binary logistic classifier

-IDEA: classify each word in its context window of neighboring words
-hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors in a window

실제론 multi-class softmax 씀

$-x_{window} = x \in R^{5d}$



$J_t(\theta) = \sigma(s) = \frac{1}{1+e^{-s}}$, J:predicted model probability of class label: correct location=1, wrong=0, predict:0~1 R
$s = u^T h$
$h = f(Wx+b)$
$x(input)$

f: non linearity (ex: softmax transformation)
h: hidden vector(smaller dimensionality)

2) Stochastic Gradient Descent (remind!)

-update eq: $\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$;α=step size or learning rate

for each parameter: $\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$

- in deep learning: data representation(e.g. word vectors) too!

1) https://stellarway.tistory.com/29

- How can we compute $\nabla_{\theta} J(\theta)$

by hand/ Algorithmically: the backpropogation algorithm

2. Computing Gradients by Hand

"Matrix Calculus"

1) Gradients

(1) Grad

- Given a function with 1 output and 1 input

  $f(x) = x^3$

- It's gradient (slope) is its derivative

  $\frac{df}{dx} = 3x^2$

  "How much will the output change if we change the input a bit?"

  At $x = 1$ it changes about 3 times as much: $1.01^3 = 1.03$

  At $x = 4$ it changes about 48 times as much: $4.01^3 = 64.48$

1 output, n inputs	
given	$f(x) = f(x_1, x_2, x_3, \ldots, x_n)$
gradient	$\frac{\partial f}{\partial x} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$
Jacobian Matrix: Generalization of the Gradient m outputs, n inputs	
given	$f(x) = [f_1(x_1, x_2, x_3, \ldots, x_n), \ldots, f_m(x_1, x_2, x_3, \ldots, x_n)]$
gradient	$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$

(2) chain rule

$$h = f(z)$$
$$z = Wx + b$$
$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial z} \frac{\partial z}{\partial x} = \ldots$$

(3) Ex for calculation

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \ \text{(Identity matrix)}$$
$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$
$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$
$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

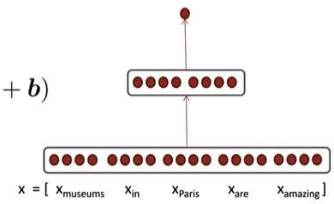
$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

2) In neural Network, Let's compute $\frac{\partial s}{\partial b}$

$s = \boldsymbol{u}^T\boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$

$\boldsymbol{x}$ (input)



$h = f(z)$

$z = Wx + b$

: define variables and keep track of their **dimensionality**

$$\frac{\partial s}{\partial b} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial x}\frac{\partial z}{\partial b} = u^T diag(f'(z)) I = u^T \bigcirc f'(z)$$

*handamard: 같은 크기의 두 행렬의 각 성분을 곱함.

$M, N \in \text{Mat}(m, n; R)$

$$M = \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1n} \\ M_{21} & M_{22} & & M_{2n} \\ \vdots & & \ddots & \vdots \\ M_{m1} & M_{m2} & \cdots & M_{mn} \end{pmatrix}$$
$$N = \begin{pmatrix} N_{11} & N_{12} & \cdots & N_{1n} \\ N_{21} & N_{22} & & N_{2n} \\ \vdots & & \ddots & \vdots \\ N_{m1} & N_{m2} & \cdots & N_{mn} \end{pmatrix}$$

$$M \bigcirc N = \begin{pmatrix} M_{11}N_{11} & M_{12}N_{12} & \cdots & M_{1n}N_{1n} \\ M_{21}N_{21} & M_{22}N_{22} & & M_{2n}N_{2n} \\ \vdots & & \ddots & \vdots \\ M_{m1}N_{m1} & M_{m2}N_{m2} & \cdots & M_{mn}N_{mn} \end{pmatrix} \in \text{Mat}(m, n; R)$$

3) In neural Network, Let's compute $\frac{\partial s}{\partial W}$: Re-using Computation!!

$$\frac{\partial s}{\partial W} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial x}\frac{\partial z}{\partial W}$$

re-use! $\frac{\partial s}{\partial b} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial b} = u^T diag(f'(z))I = u^T \bigcirc f'(z)$

$$\frac{\partial s}{\partial W} = \delta\frac{\partial z}{\partial W}$$

$\frac{\partial s}{\partial b} = \delta\frac{\partial z}{\partial b} = \delta$ δ: local error signal[2)]

$$\delta = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z} = u^T \bigcirc f'(z)$$

-> Jacobian:

$\frac{\partial s}{\partial W}$는 1 by nm Jacobian

incoveninet to subtract (when grad is hug row vector) $\theta^{new} = \theta^{old} - \alpha\nabla_\theta J(\theta)$

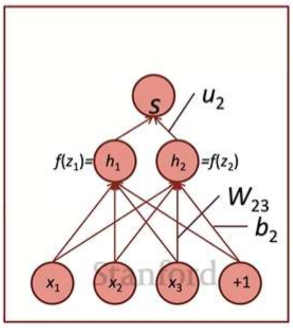
4) Let Shape Convention: shape of the grad= shape of param

$$\frac{\partial s}{\partial W} = \begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

<<<<<<<<Go back to calculation,

$\frac{\partial s}{\partial W} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial x}\frac{\partial z}{\partial W}$ note that $z = Wx + b$

-> $\frac{\partial s}{\partial W} = \delta^T x^T$ δ: local error signal at z/ x: local input signal



single wieght W_{ij}

W_{ij} contricutes only on z_i

2) *And so we do that by defining delta, which is delta is the partials composed that are above the linear transform. And that's referred to as the local error ???

$$\frac{\delta z_i}{\delta W_{ij}} = \frac{\delta}{\delta W_{ij}}(W_i.x + b_i) = \frac{\delta}{\delta W_{ij}} \sum_{k=1}^{d} W_{ik}.x_k = x_j$$

$$\begin{array}{ccc} \frac{\partial s}{\partial \boldsymbol{W}} & = & \boldsymbol{\delta}^T \quad \boldsymbol{x}^T \\ [n \times m] & & [n \times 1][1 \times m] \end{array}$$

$$= \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} [x_1, ..., x_m] = \begin{bmatrix} \delta_1 x_1 & \dots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \dots & \delta_n x_m \end{bmatrix}$$

-What shape should derivatives?

$\frac{\partial s}{\partial b} = h^T \circ f'(z)$: row vector (b is column vector..,, contradiction!)

Jacobian form(chain rule easy) VS shape convention(SGD easy) ->Disagreement

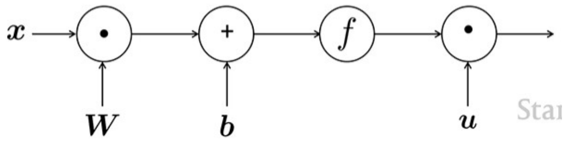
sol1: Use Jacobian form as much as possible, reshape to follow the shape convention at the end

sol2: Always shpae convention

3. Backpropogation

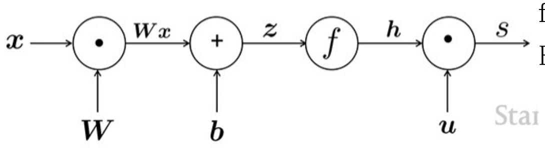
1) Basic Idea

Re-use Derivatives compute higher layers -> lower layers



source nodes: input

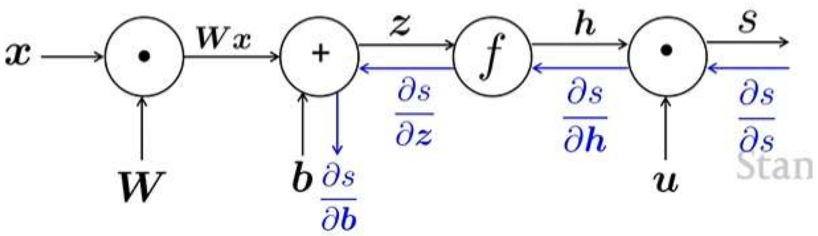
interior nodes:operations



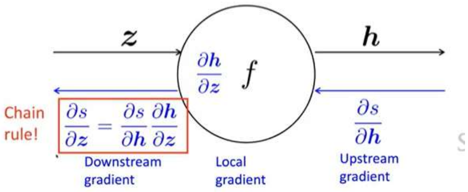
forward propogation

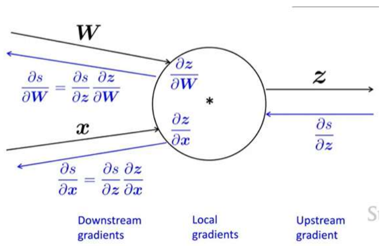
Edge: Results

2) Back Propogation

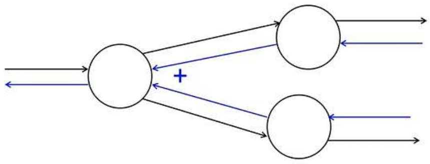


each node has a local gradient





get upstream gradient -> chain rule with local gradient -> give downstream grad



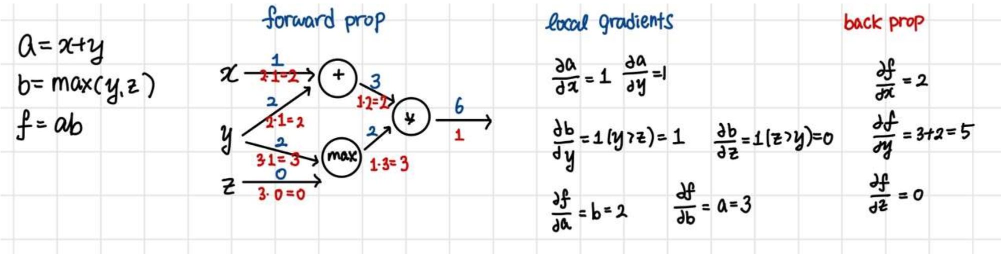
Gradients sum at outward branches

$$a = x + y$$
$$b = \max(y, z) \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$
$$f = ab$$

4. Example

1)

$f(x,y,z) = (x+y)\max(y,z)$
$x = 1,\ y = 2,\ z = 0$

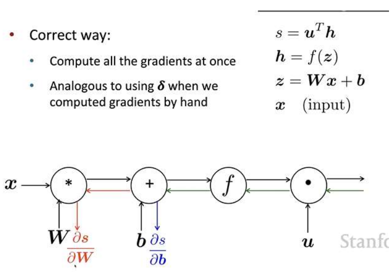


node intuitions: +: “distributes” the upstream gradient to each summand

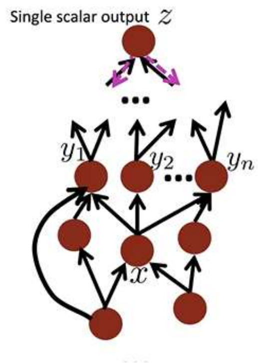
max: “routes” the upstream gradient

∗ : “switches” the upstream gradient

2) Efficiency: compute all gradients at once



3) Back-Prop in General Computation Graph



Fprop: visit nodes in topological sort order

comput value of node given predecessors

Bprop:

initialize output gradient = 1

visit nodes in reverse order

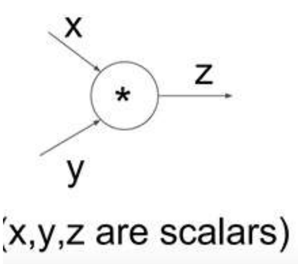
compute each node using successors

successors of x={$y_1, y_2, \cdots, y_n$}

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

big O()complexity in fprop = bprop

4) Code




```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

5. Manual Gradient Checking

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Summary

We've mastered the core technology of neural nets! 🎉

- Backpropagation: recursively (and hence efficiently) apply the chain rule along computation graph
  - [downstream gradient] = [upstream gradient] x [local gradient]
- Forward pass: compute results of operations and save intermediate values
- Backward pass: apply chain rule to compute gradients

자막(c)

Stanford

##Why do we hand-calculate?
-modern DL frameworks don't give local derivative
-Back Propogation fails easily
-Check Layers are correctly implemented

Lecture 4 -Synthetic Structure and Dependency Parsing

1. Content

- Synthetic Sturcture: Consistency and Dependency
- Dependency Grammar and Treebanks
- Transition-based dependency parsing
- Neural dependency parsing

2. Syntatic Sturcture: Consistency and Dependency

1) Constituency= phrase sturcture grammar = context-free grammars(CFGs)

(1) Phrase structure: organize words into nested constituents

EX) starting units: words(the, cat, cuddly, by, door)

words combine into phrases(the cuddly cat, by the door)

phrases can combine into bigger phrases (the cuddly cat by the door)

(2) How does it work?

see: talk to, walk behind, in a crate, on the table

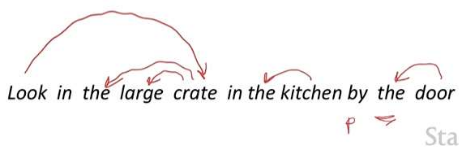
Grammer:S -> NP VP / NP->Det (Adj)*[3)]N(pp) / pp->P NP / VP -> V pp

Lexicon {N : dog, N: cat, Det: a, Det: the, pp: in}

=> got Grammer and Lexicon

2) Dependency Structure (*more frequently used)

shows which words depend on (modify, attach to, or are arguments of) which other words



3) Why do we need sentence structure

Complex ideas <- compsing words together into bigger units to convey complex meanings

Listeners need to work out what [modifies/ attaches to] what

A model needs to understand sentence sturcture in ordder to be able to interpre language correctly

(1) PP attachment ambiguities multiply: non-solved

---

3) ()*: 0 or other numbers, (): 0 or 1

- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with $n+2$ sides
    - Turns up in triangulation of probabilistic graphical models (CS228)....

(2) Coordination scope ambiguity

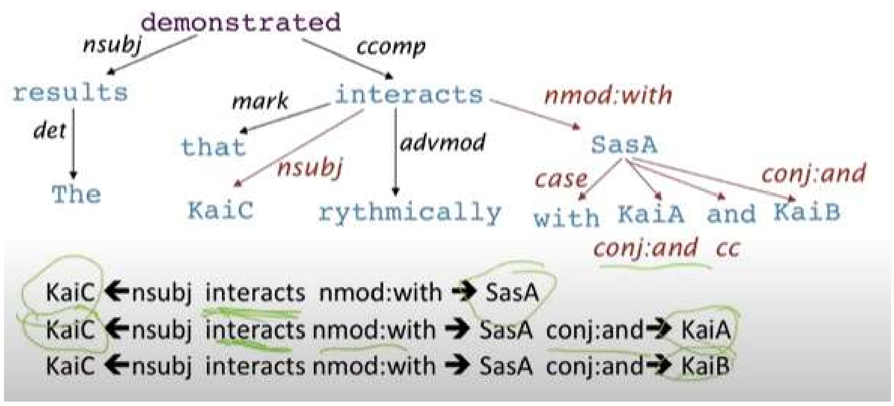
[Shuttle veteran] and [longtime NASA executive Fred Gregory] appointed to board   2 people

[Shuttle veteran and longtime NASA executive] Fred Gregory appointed to board   1 person

(3) Adjectival/ Adverbial Modifier Ambiguity

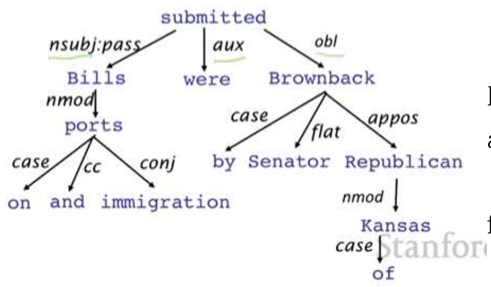
3. Dependency Grammar and Treebanks

-Dependency paths help extract semantic interpretation ex. extracting protein-protein interpretion



1) Dependency Grammar

: postulates that syntatic structure consists of realtion between lexical items, binary asymmetric relations("arrows") called dependencies

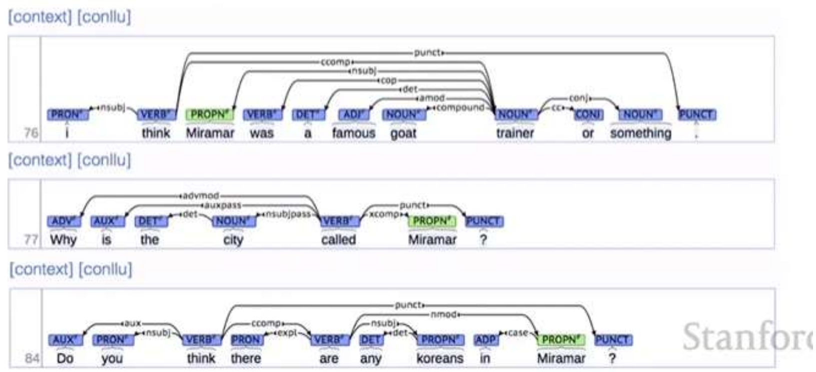


tree analysis (acyclic, single-rooted graph)
head-> dependent
arrow: typed with grammatical relationships

fake root: 모든 word가 dependent 가지게

2) Tree Banks: Annotated Data & Universal Dependencies Trees(for lots of human language)

-hand parsing



why annotated data rather than writing a grammar
- Reusability of the labor
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- way to evaluate NLP systems

3) How to build parser by dependancy?

Sources of information

1. Bilexical affinities — The dependency [discussion → issues] is plausible
2. Dependency distance — Most dependencies are between nearby words
3. Intervening material — Dependencies rarely span intervening verbs or punctuation
4. Valency of heads — How many dependents on which side are usual for a head?

4) Dependency Parsing

sentence is parse by choosing for each word what other word it is a dependent of
constraints (usually)
: only one word is a dependent of ROOT
: Don't want cycles A->B, B->A

final issues: wheter arrows can cross(be non-projective) or not

5) Projectivity

projective parse: no crossing dependency arcs

dependencies corresponding to a CFG(Context Free Grammar): must be projectivity

4. Methods of Dependency Parsing

1. Dynamic programming
   Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle
2. Graph algorithms
   You create a Minimum Spanning Tree for a sentence
   McDonald et al.'s (2005) MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)
   Neural graph-based parser: Dozat and Manning (2017) et seq. – very successful!
3. Constraint Satisfaction
   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
4. "Transition-based parsing" or "deterministic dependency parsing"
   Greedy choice of attachments guided by good machine learning classifiers
   E.g., MaltParser (Nivre et al. 2008). Has proven highly effective.

<Transition-based parsing>

0) shift reduce parsing[4][5]

$$\underbrace{\alpha\beta}_{\text{seen and possibly reduced, contains terminals and non-terminals}} | \underbrace{\omega}_{\text{not yet seen, contains only terminals}}$$

대체할만한 reduction이 나올 때까지 separator(|)을 한 칸 씩 이동. 대체 가능하면 대체.

-Shift: This involves moving symbols from the input buffer onto the stack.

-Reduce: If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of a production rule is popped out of a stack and LHS of a production rule is pushed onto the stack

1) Greedy Transition-based parsing

-simple form of greedy discriminative dependency parser

---

4) https://talkingaboutme.tistory.com/entry/Compiler-Shift-Reduce-Parsing

5) https://www.geeksforgeeks.org/shift-reduce-parser-compiler/

-parser has:

stack σ: written with top to the right (starts with ROOT symbol)

buffer β: written with top to the left (starts with the input sentence)

a set of dependency arcs A (starts off empty)

a set of actions

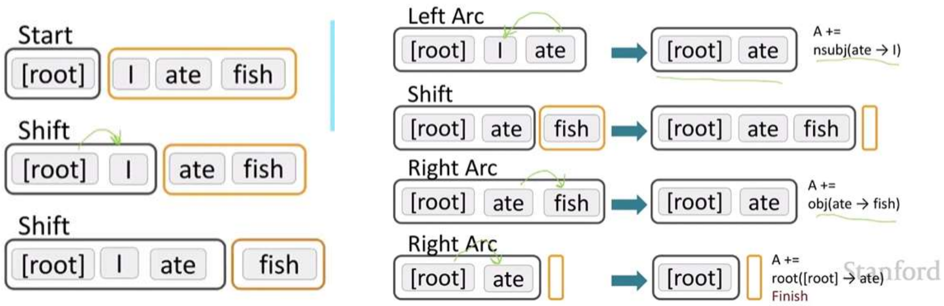
- how

Start: $\sigma = [\mathrm{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset$

1. Shift $\quad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j, w_i)\}$
3. Right-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i, w_j)\}$

Finish: $\sigma = [w], \beta = \emptyset$

- ex



2) MaltParser

-how choose next action(classify)

-Discriminative Classifier over each legal move

R*2+1 choices

Features: top of stcak words, POS; first in buffer word, POS(태그임); etc.

-NO search

-very fast linear time parsing

->back of words: dimension 증가

->sol: neural network

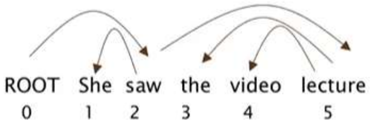
5. Evaluation of Dependency Parsing: (labeled) dependency accuracy

Unlabeled

$$Acc = \frac{\#\, correct\, deps}{\#\, of\, deps}$$

Labeled

Label & Dependancy



Gold			
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed			
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

https://gnoej671.tistory.com/5