Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Contents

Phrase structure

- Organizes words into nested constituents
- Starting unit: 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

Source: CS224n

Phrase structure

- Organizes words into nested constituents
- Starting unit: words are given a category (part of speech = POS)
  - the, cat, cuddly, by, door
  - Det N Adj P N
- Words combine into phrases with categories
  - the cuddly cat by the door
  - NP → Det Adj N PP → P NP
- Phrases can combine into bigger phrases recursively
  - the cuddly cat by the door
  - NP → NP PP

Source: CS224n

Phrase structure

- Organizes words into nested constituents

the cat

a dog

large in a crate

barking on the table

cuddly by the door

talk to

walked behind

Contents

- NLP application: Dependency parsing
  - Phrase structure, dependency structure, part-of-speech (POS) tagging
  - Before deep neural network: MaltParser
  - Neural approach
- NLP application: Text classification
- Convolutional Neural Network (CNN)
  - CNN for text classification

Another view of linguistic structure: Dependency structure

- Shows which words depend on (modify or are arguments of) which other words

Look in the large crate in the kitchen by the door

Source: CS224n

Why do we need sentence structure?

- We need to understand sentence structure in order to be able to interpret language correctly
- Humans communicate complex ideas by composing words together into bigger units to convey complex meanings
- We need to know what is connected to what

Source: CS224n

Recall

Why is NLP hard?

- Human languages are ambiguous, e.g.
  - I made her duck
    - I cooked waterfowl for her benefit (to eat)
    - I cooked waterfowl belonging to her
    - I created the (ceramic?) duck she owns
    - I caused her to quickly lower her head or body
    - I waved my magic wand and turned her into waterfowl
  - Girl hit by car in hospital
  - Singapore court to try shooting defendant
  - Farmer Bill dies in house

Source: CS224n

Why is NLP hard? – POS tagging
E.g. I made her duck

- I/PRP made/VBD her/PRP duck/NN
- I/PRP made/VBD her/PRP$ duck/NN
- I/PRP made/VBD her/PRP duck/VB

Tag	Description
PRP	Personal pronoun
PRP$	Possessive pronoun
VBD	Verb, past tense
VB	Verb, base form
NN	Noun, singular or mass

- I cooked waterfowl for her benefit (to eat)
- I cooked waterfowl belonging to her
- I created the (ceramic?) duck she owns
- I caused her to quickly lower her head or body
- I waved my magic wand and turned her into waterfowl

Why is NLP hard? – Dependency structure (E.g. I made her duck)

- I/PRP made/VBD her/PRP duck/NN
- I/PRP made/VBD her/PRP$ duck/NN
- I/PRP made/VBD her/PRP duck/VB

Tag	Description
PRP	Personal pronoun
PRP$	Possessive pronoun
VBD	Verb, past tense
VB	Verb, base form
NN	Noun, singular or mass

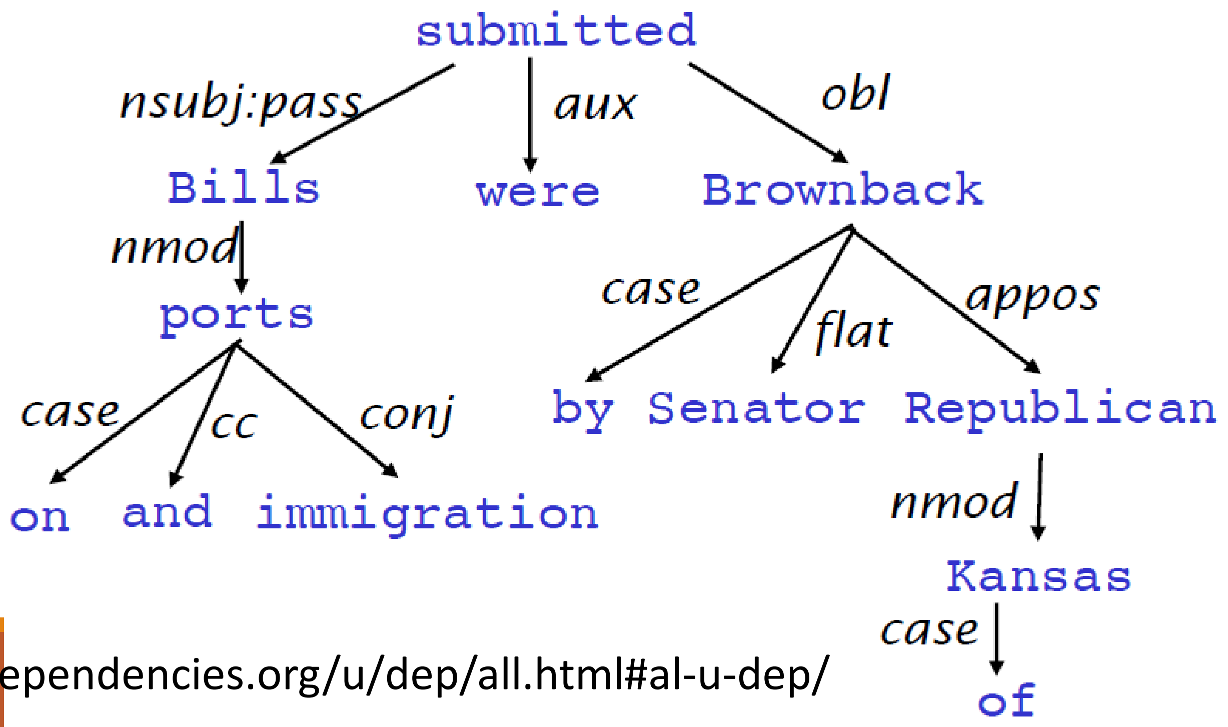
Why is NLP hard?

- Girl hit by car in hospital
- Singapore court to try shooting defendant
- Farmer Bill dies in house

- Girl hit by car in hospital
- Singapore court to try shooting defendant
- Farmer Bill dies in house

Dependencies

- Binary asymmetric relations ("arrows")
- The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)

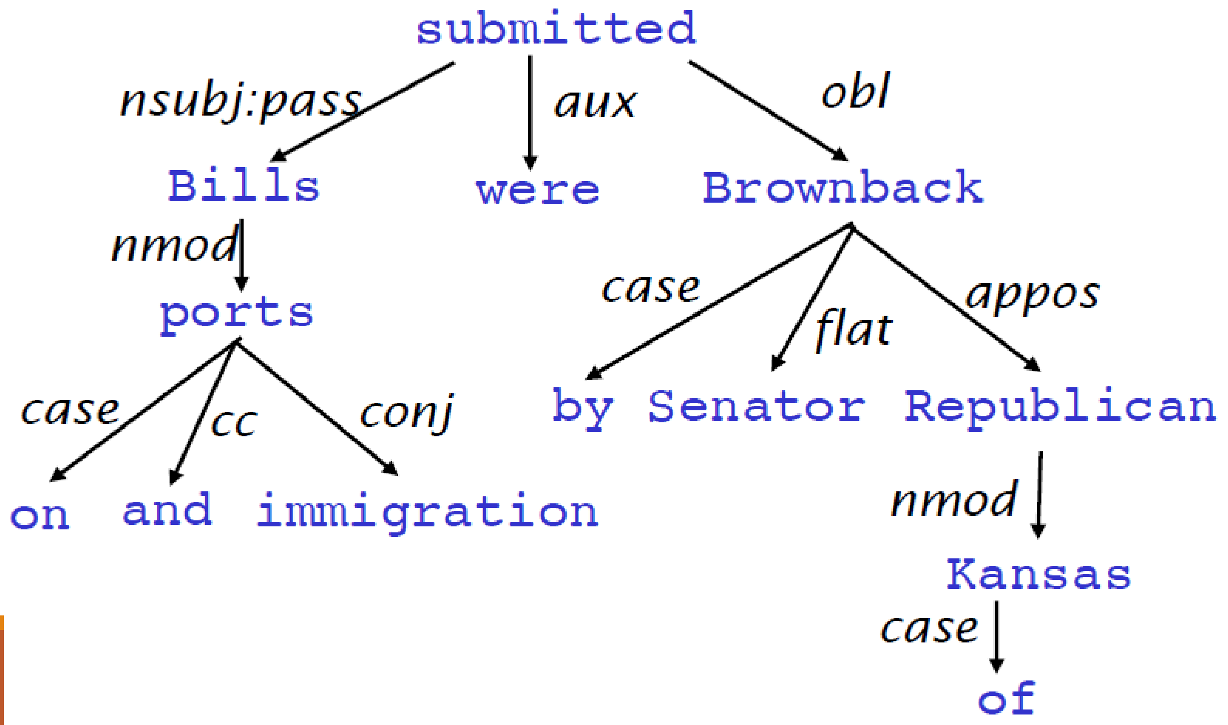


Type	Description
nsubj:pass	Nominal subject, passive voice
nmod	Nominal modifier
case	Case marking
cc	Coordinating conjunction
conj	Conjunct
aux	Auxiliary
obl	Oblique nominal
flat	Flat multi-word expression
appos	Appositional modifier

https://universaldependencies.org/u/dep/all.html#al-u-dep/

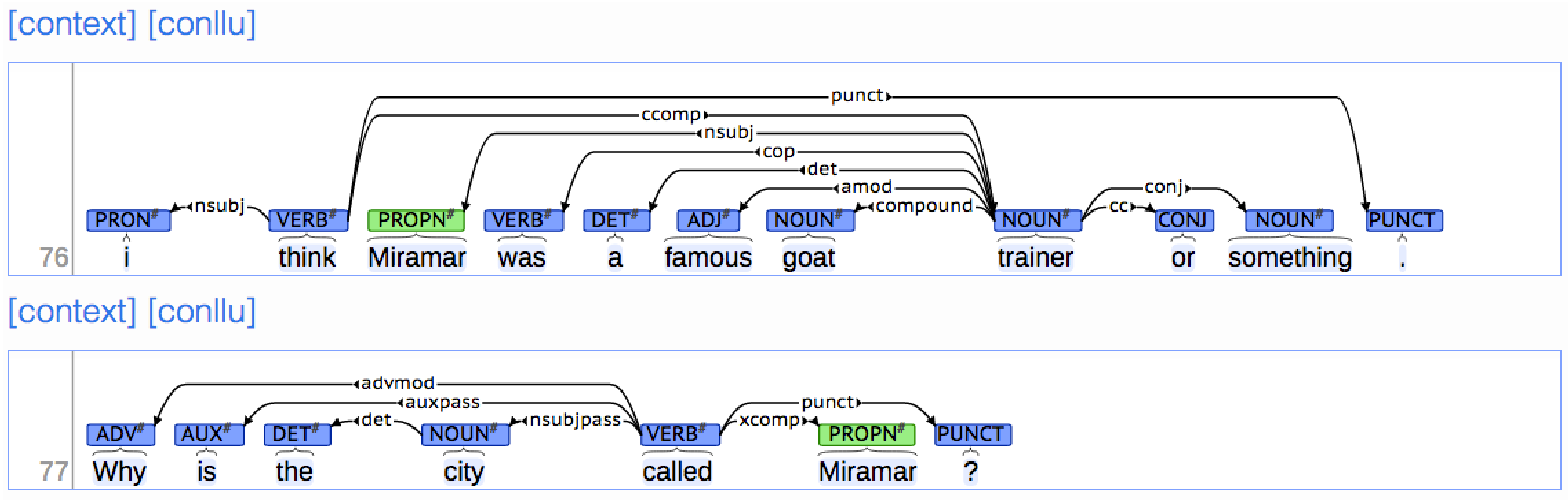
Dependencies

- The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)
- Usually, dependencies form a tree (connected, acyclic, single-head)



Source: CS224n

The rise of annotated data: Universal Dependencies treebanks


[context] [conllu]
76
punct
ccomp
nsubj
cop
det
amod
compound
conj
cc
nsubj
PRON VERB PROPN VERB DET ADJ NOUN NOUN CONJ NOUN PUNCT
i think Miramar was a famous goat trainer or something .
[context] [conllu]
77
advmod
auxpass
det
nsubjpass
xcomp
punct
ADV AUX DET NOUN VERB PROPN PUNCT
Why is the city called Miramar ?

- Earlier: Marcus et al. 1993, The Penn Treebank, Computational Linguistics

The rise of annotated data

- Starting off, building a treebank seems a lot slower and less useful than building a grammar
- But a treebank gives us many things
  - Reusability of the labor
    - Many parsers, part-of-speech taggers, etc. can be built on it
    - Valuable resource for linguistics
  - Broad coverage, not just a few intuitions
  - Frequencies and distributional information
  - A way to evaluate systems

Source: CS224n

Contents

- NLP application: Dependency parsing
  - Phrase structure, dependency structure, part-of-speech (POS) tagging
  - Before deep neural network: MaltParser
  - Neural approach
- NLP application: Text classification
- Convolutional Neural Network (CNN)
  - CNN for text classification

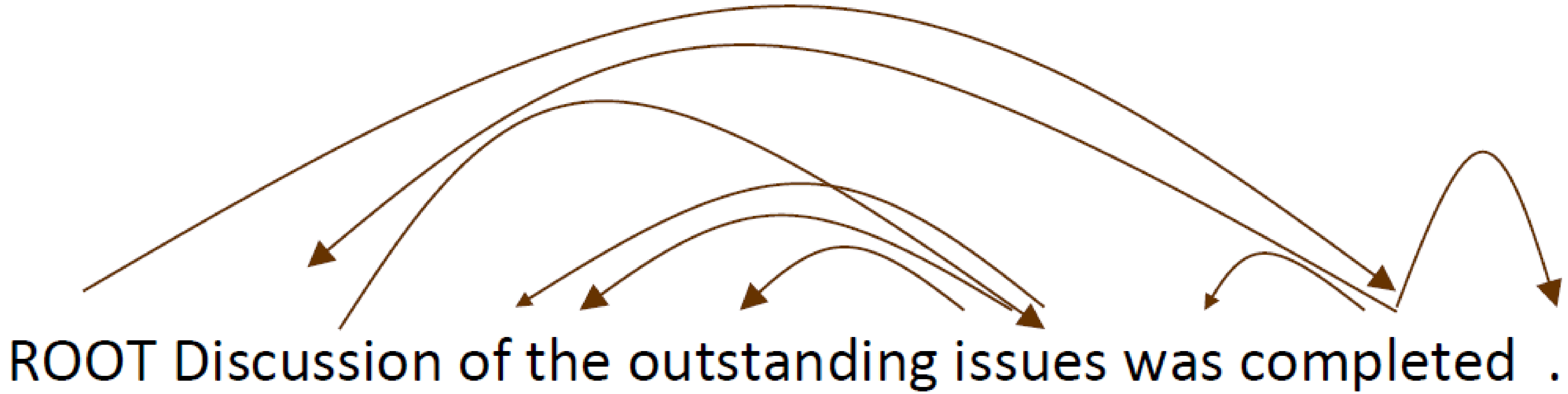
Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
  - Basic approach of 1st millennium Arabic grammarians
- Phrase structure/context-free grammars is a new-fangled invention
  - 20th century invention (R.S. Wells, 1947; then Chomsky)
- Modern dependency work often sourced to L. Tesnière (1959)
  - Was dominant approach in "East" in 20th Century (Russia, China, …)
    - Good for free-er word order languages
- Among the earliest kinds of parsers in NLP, even in the US:
  - David Hays, one of the founders of U.S. computational linguistics, built early (first?) dependency parser (Hays 1962)

Source: CS224n

Dependency Grammar and Dependency Structure

- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent…
- Usually add a fake ROOT so every word is a dependent of precisely 1 other node

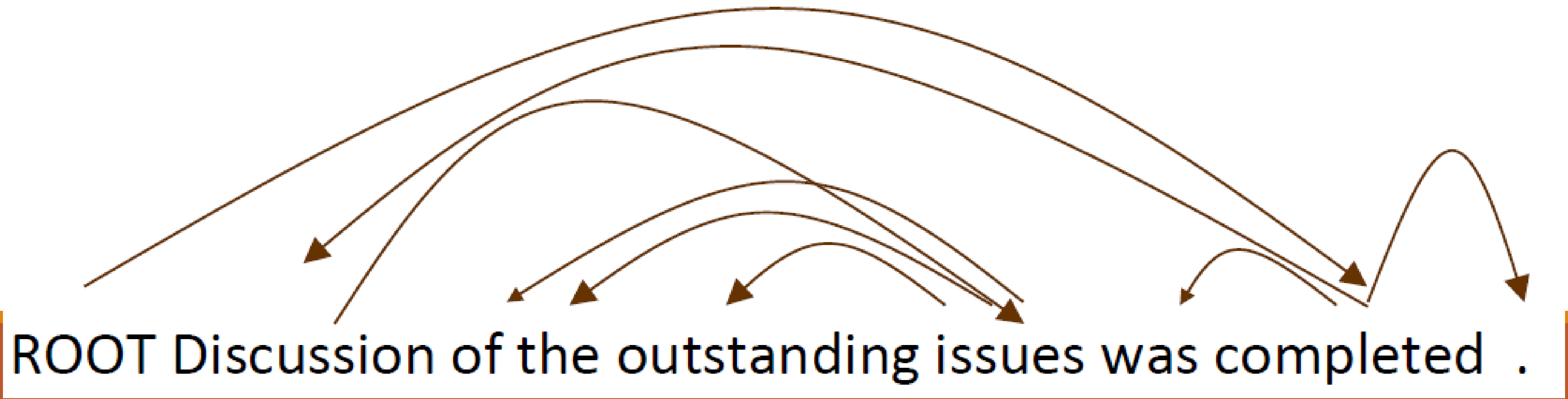


Source: CS224n

Dependency Conditioning Preferences

- What are the sources of information for dependency parsing?

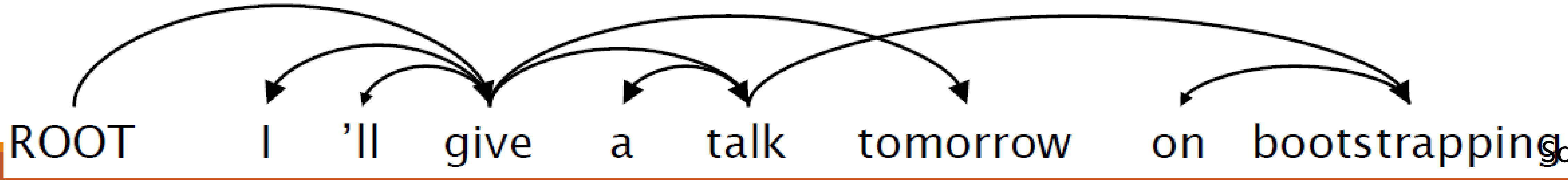
1. Bilexical affinities [discussion → issues] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material Dependencies rarely span intervening verbs or punctuation
4. Valency of heads How many dependents on which side are usual for a head?



Source: CS224n

Dependency Parsing

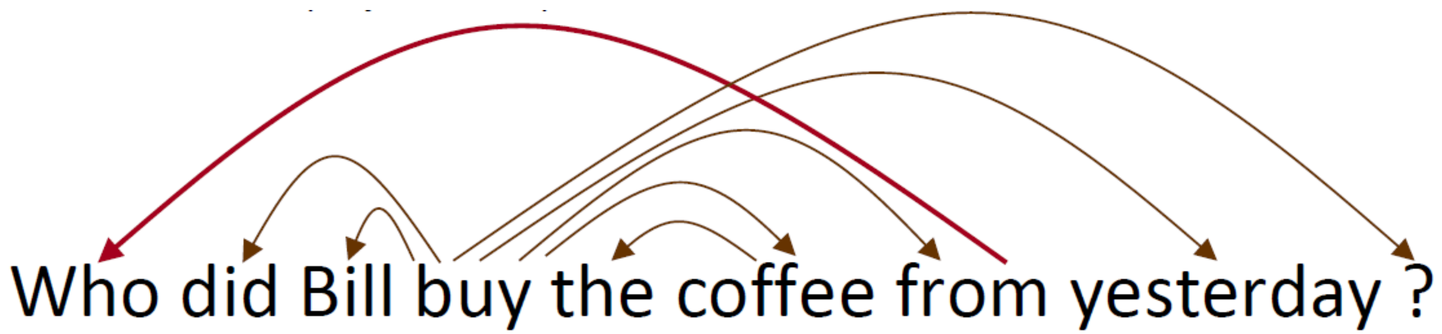
- A sentence is parsed by choosing for each word what other word (including ROOT) it is a dependent of
- Usually some constraints:
  - Only one word is a dependent of ROOT
  - Don't want cycles A → B, B → A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (non-projective) or not



Source: CS224n

Projectivity

- Definition: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- But dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these non-projective dependencies



Source: CS224n

Methods of dependency parsing before deep neural network

- 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
- Graph algorithms
  - You create a Minimum Spanning Tree for a sentence
  - McDonald et al.'s (2005) MSTParser scores dependencies independently using an ML classifier
- Constraint Satisfaction
  - Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
- "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.

Source: CS224n

Greedy transition-based parsing

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like "shift" or "reduce" in a shift-reduce parser, but the "reduce" actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack σ, written with top to the right, which starts with the ROOT symbol
  - a buffer β, written with top to the left, which starts with the input sentence
  - a set of dependency arcs A (head -> dependent), which starts off empty
  - a set of actions

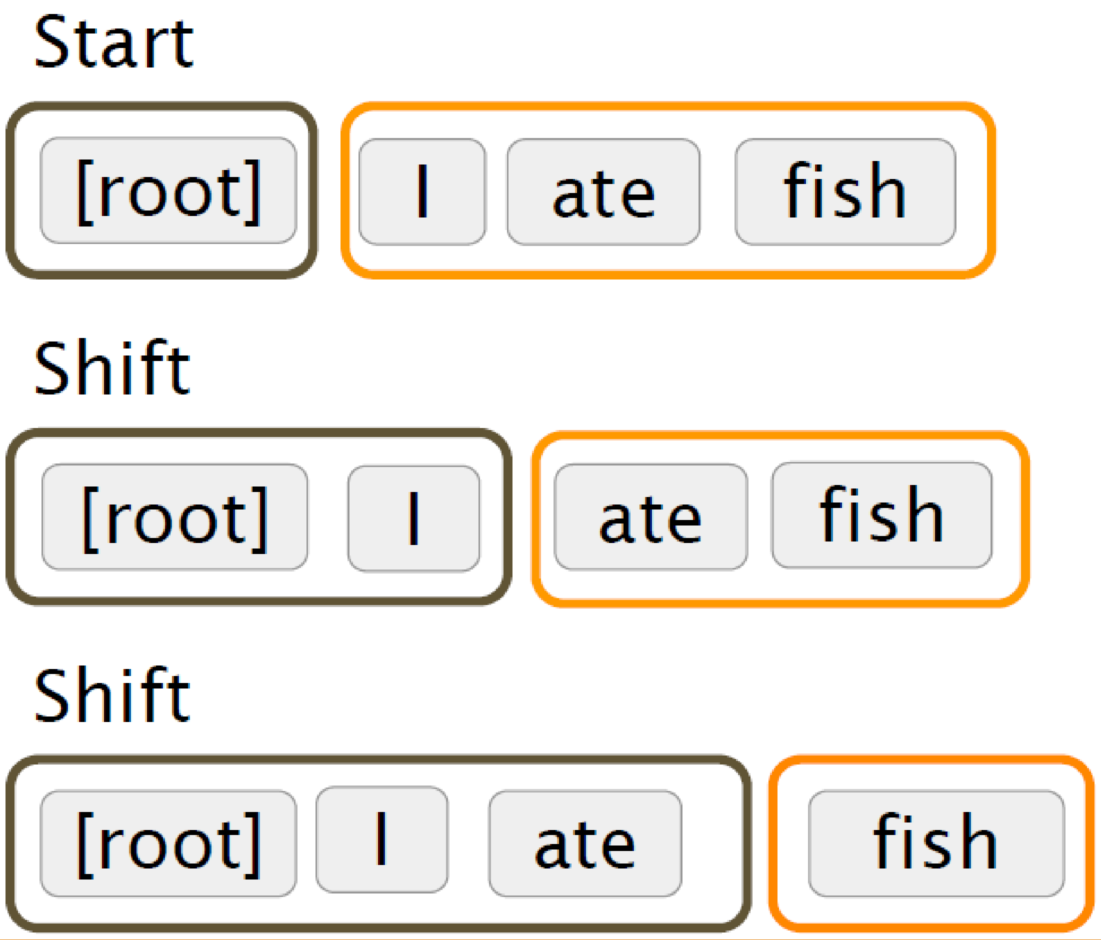
Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT), pages 149–160, Nancy, France.

Basic transition-based dependency parser

- Start: stack σ = [ROOT], buffer $\beta = w_1, \ldots, w_n$, arcs $A = \emptyset$
- Actions
  - Shift $\sigma, \quad w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
  - Left-Arc$_r$ $\sigma|w_i|w_j, \beta, \quad A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j,w_i)\}$
  - Right-Arc$_r$ $\sigma|w_i|w_j, \beta, \quad A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i,w_j)\}$
- Finish: $\sigma = [w]$, $\beta = \emptyset$

Source: CS224n

Arc-standard transition-based parser: e.g. “I ate fish”



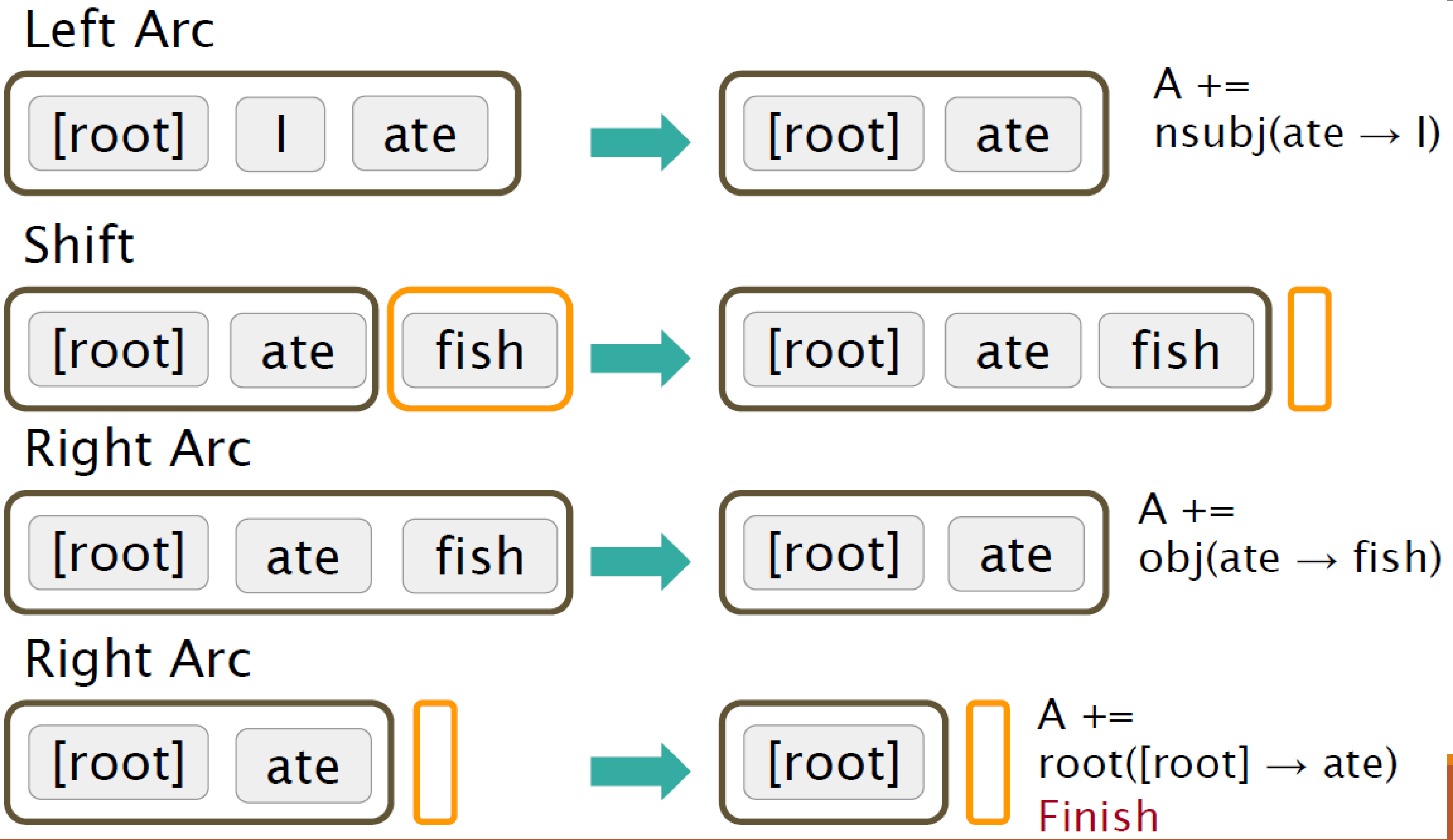
Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = ∅

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: $\sigma = [w]$, β = ∅

Source: CS224n

Arc-standard transition-based parser: e.g. "I ate fish"



Source: CS224n

MaltParser

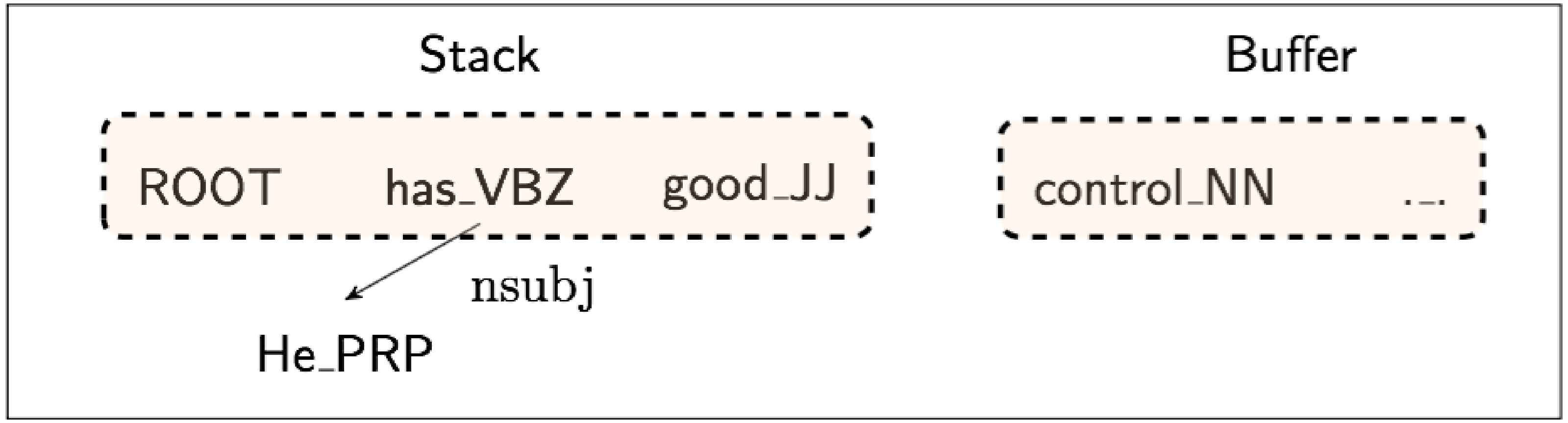
- We have left to explain how we choose the next action
  - Answer: Stand back, I know machine learning!
- Each action is predicted by a discriminative classifier (e.g., softmax classifier) over each legal move
  - Max of 3 untyped choices; max of $|R| \times 2 + 1$ when typed
  - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest form)
  - But you can profitably do a beam search if you wish (slower but better): You keep k good parse prefixes at each time step

Nivre, J. and J. Nilsson (2005) Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99-106.

MaltParser

- The model's accuracy is *fractionally* below the state of the art in dependency parsing, but
- It provides very fast linear time parsing, with great performance

Conventional Feature Representation



$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$

$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$

$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$

$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

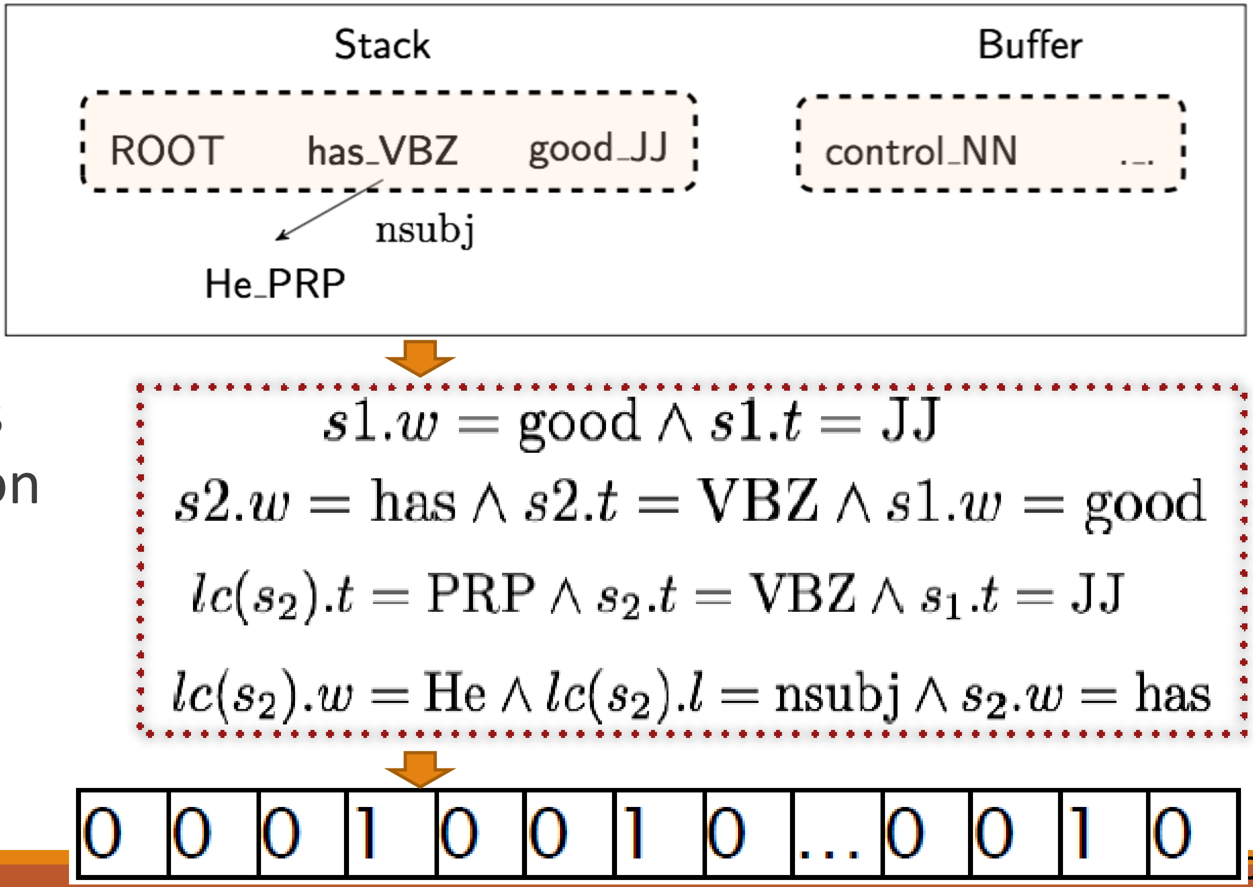
0	0	0	1	0	0	1	0	...	0	0	1	0

- Feature templates: usually a combination of 1~3 elements from the configuration
- Binary, sparse dim. = 10^6~ 10^7

ource: CS224n

Conventional Feature Representation: Issues

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: expensive computation
  - More than 95% of parsing time is consumed by feature computation



$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$

$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$

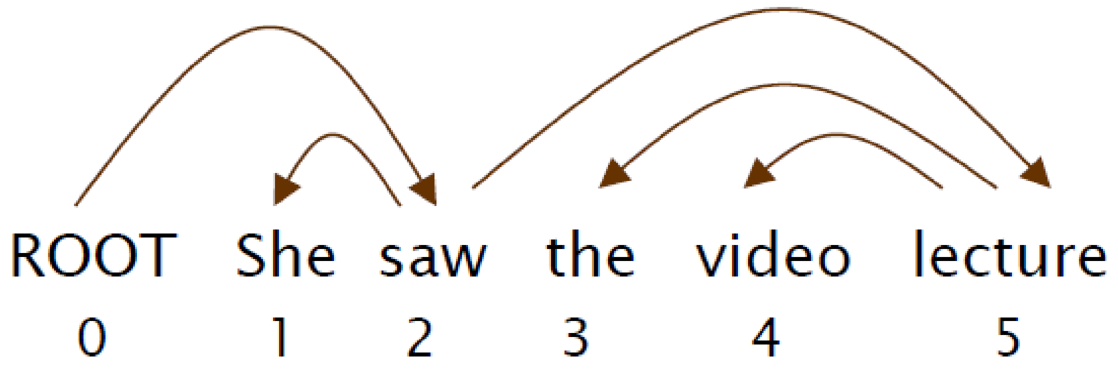
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$

$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

ource: CS224n

Evaluation of Dependency Parsing: (labeled) dependency accuracy

- Accuracy = $\frac{\text{\# of correct deps}}{\text{\# of dependencies}}$
- UAS = 4/5 = 80%
- LAS = 2/5 = 40%
  - Unlabeled/ labeled attachment score



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

Source: CS224n

Handling non-projectivity

- The arc-standard algorithm we presented only builds projective dependency trees
- Possible directions to head:
  - Just declare defeat on nonprojective arcs
  - Use dependency formalism which only has projective representations
  - Use a postprocessor to a projective dependency parsing algorithm to identify and resolve nonprojective links
  - Add extra transitions that can model at least most non-projective structures (e.g., add an extra SWAP transition, cf. bubble sort)
  - Move to a parsing mechanism that does not use or require any constraints on projectivity (e.g., the graph-based MSTParser)

Source: CS224n

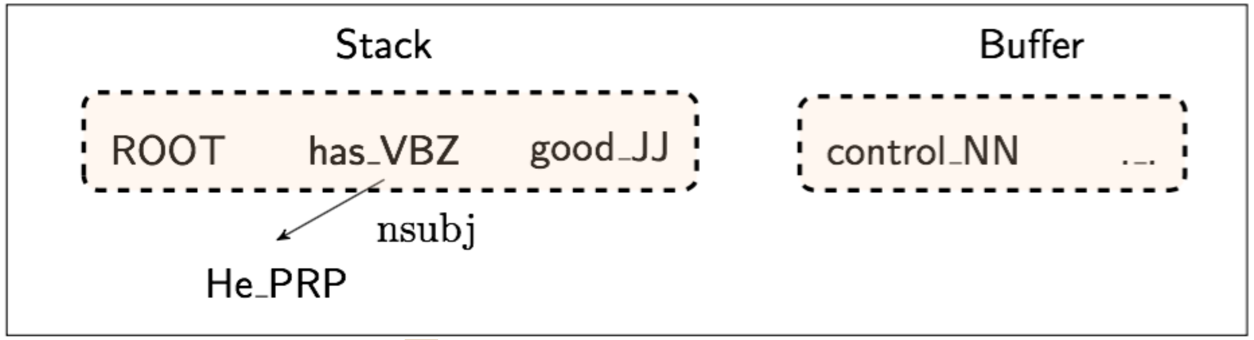
Contents

Recall

Conventional Feature Representation: Issues

- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: expensive computation
  - More than 95% of parsing time is consumed by feature computation

Neural approach: Learn a dense and compact feature representation



$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

0	0	0	1	0	0	1	0	...	0	0	1	0

ource: CS224n

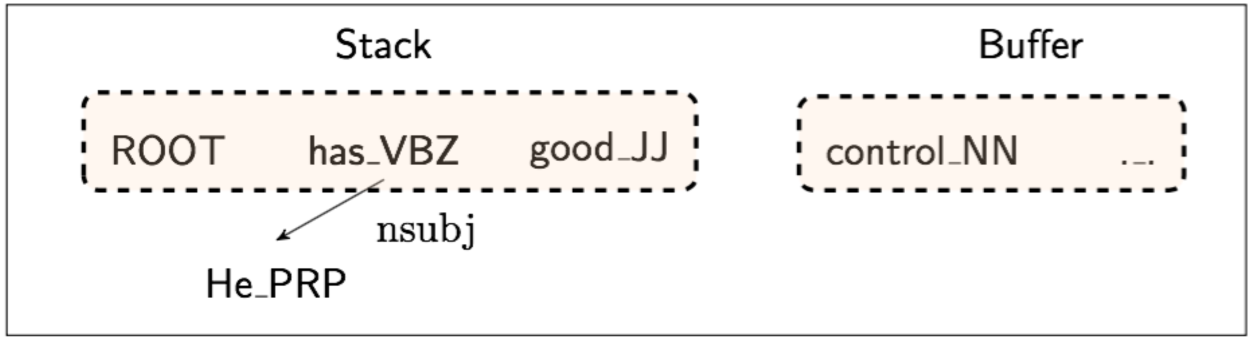
Distributed Representations

- We represent each word as a *d*-dimensional dense vector (i.e., word embedding)
  - Similar words are expected to have close vectors.
- Meanwhile, part-of-speech tags (POS) and dependency labels are also represented as *d*-dimensional vectors
  - The smaller discrete sets also exhibit many semantic similarities.
  - E.g. NNS (plural noun) should be close to NN (singular noun).
  - E.g. num (numerical modifier) should be close to amod (adjective modifier).

Source: CS224n

Extracting Tokens and then vector representations from configuration

- Extract a set of tokens based on the stack / buffer positions:
  - S1: last token in stack
  - S2: second last token in stack
  - B1: first token in buffer
  - lc: leftmost child
  - rc: rightmost child



	word	POS	dep.
s_1	good	JJ	∅
s_2	has	VBZ	∅
b_1	control	NN	∅
$lc(s_1)$	∅	∅	∅
$rc(s_1)$	∅	∅	∅
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	∅	∅	∅

$\rightarrow$ word $+$ POS $+$ dep.

A concatenation of the vector representation of all these is the neural representation of a configuration

Model Architecture

Output layer *y*

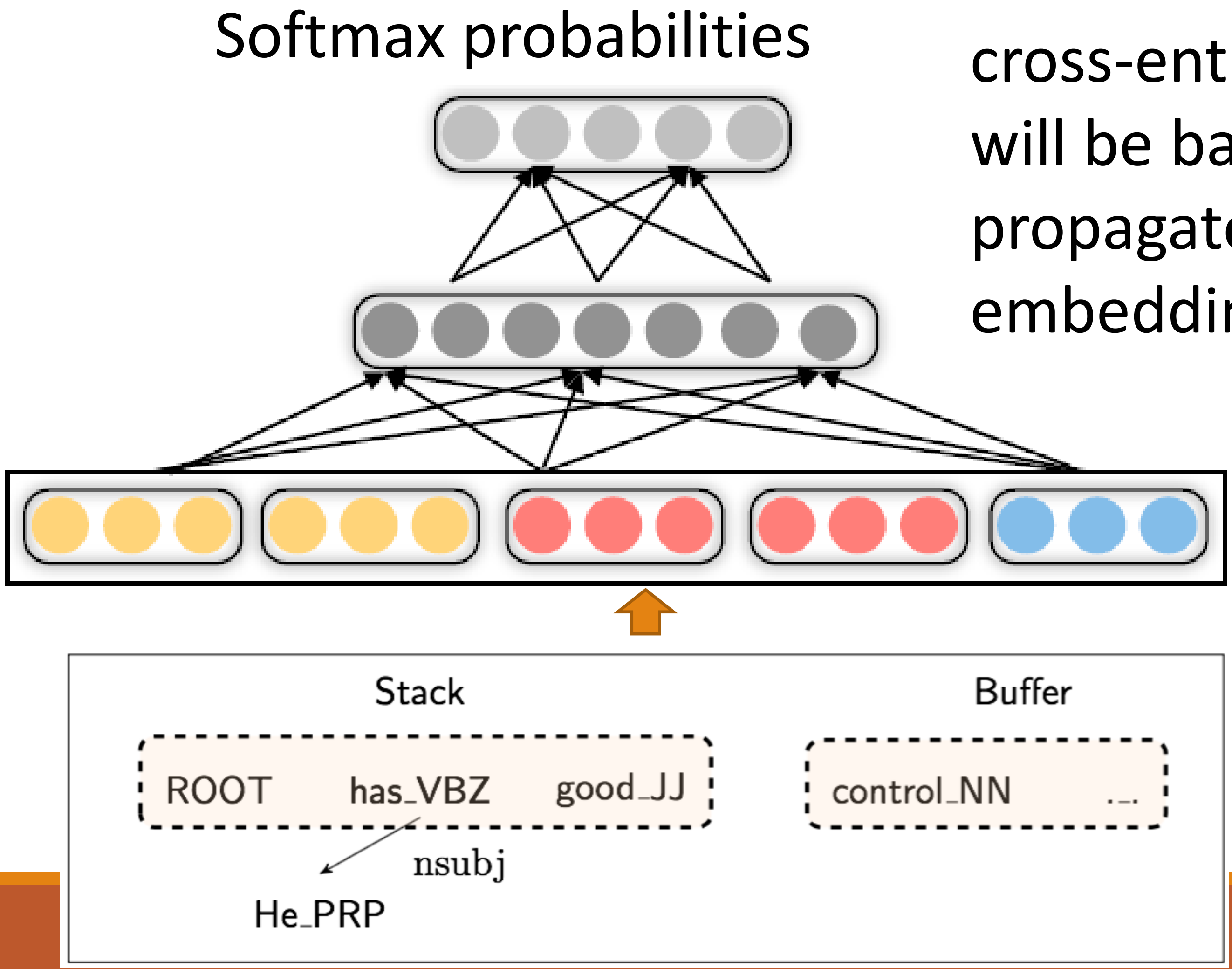
$y = \text{softmax}(Uh + b_2)$

Hidden layer *h*

$h = \text{ReLU}(Wx + b_1)$

Input layer *x*

Lookup + *concat*



cross-entropy error will be back-propagated to the embeddings

ReLU (rectified linear unit): Later in CNN

Source: CS224n

Evaluation results (Chen & Manning, 2014)

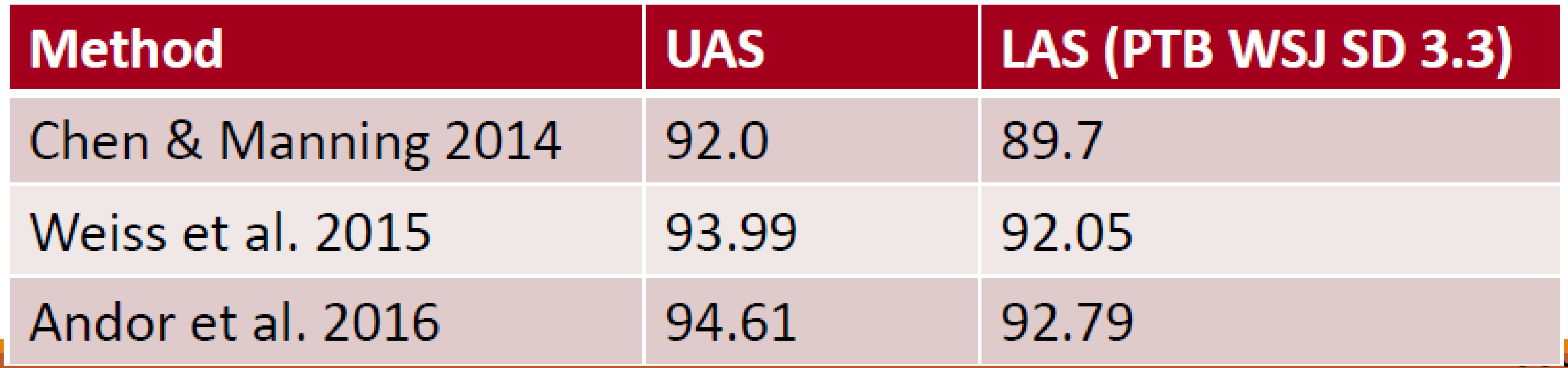
- English parsing to Stanford Dependencies:
  - Unlabeled attachment score (UAS) = head
  - Labeled attachment score (LAS) = head and label

Parser	UAS	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**

Source: CS224n

Further developments in transition-based neural dependency parsing

- This work was further developed and improved by others, including at Google
  - Bigger, deeper networks with better tuned hyper-parameters
  - Beam search
  - Global, conditional random field (CRF)-style inference over the decision sequence
- Leading to SyntaxNet and the Parsey McParseFace model
  - See the link below



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

Contents

- NLP application: Dependency parsing
  - Phrase structure, dependency structure, part-of-speech (POS) tagging
  - Before deep neural network: MaltParser
  - Neural approach
- NLP application: Text classification
- Convolutional Neural Network (CNN)
  - CNN for text classification

Text classification examples

- Routing customer email to corresponding division
- Spam detection
- Analyzing sentiment expressed in customer feedback, social media
- Classifying texts into (international) standard categories
  - E.g. patent, doctor’s reports, financial records
- Classifying problem statements into known problem hierarchy
  - E.g. such a hierarchy reflects institutional knowledge

Recall

tf-idf vector

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\mathrm{w}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Source: IR

Recall

Tutorial 1.2: Sentiment analysis of movie reviews by using binary/tf-idf vectors

- Using NLTK (https://www.nltk.org/) and scikit-learn (https://scikit-learn.org/)
- Download and preprocess a corpus of movie review
- Convert documents into binary vectors
- Run classification methods (e.g. SVM, RF) with binary vectors
- Convert documents into tf-idf vectors
- Run classification methods with tf-idf vectors

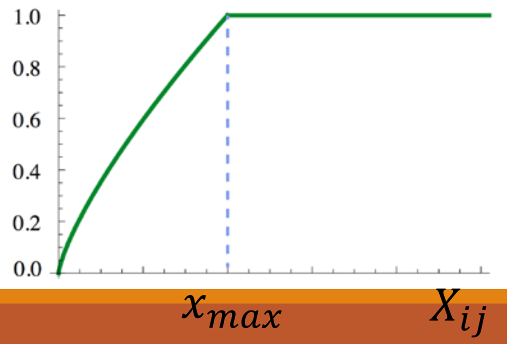
https://drive.google.com/file/d/1DyDKHSg6yTpDNQgv5bMfZKi-VDvCHVkL/view?usp=sharing

Recall

GloVe: Encoding meaning in vector differences

- Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?
- A: Log-bilinear model $w_i \cdot w_j = w_i^T w_j = \log p(i|j)$

  with vector differences $w_x \cdot (w_a - w_b) = \log \frac{p(x|a)}{p(x|b)}$
- Add a bias term for each word to capture the fact that some words occur more often than others
- For $w_i \cdot w_j$ $$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$ $f(X_{ij})$



https://www.aclweb.org/anthology/D14-1162/

Hands-on: Sentiment analysis of movie reviews by using word vectors

- Using GloVe word vectors (https://nlp.stanford.edu/projects/glove/)
- Download and preprocess movie review corpus
- Convert each document into average vector of all word vectors
- Train SoftmaxClassifier
- Convert each document into concatenation of all word vectors
- Train SoftmaxClassifier

https://drive.google.com/file/d/1fbVUPXcBQiuKnumQpmNKAY6Yhfe4iPBP/view?usp=sharing

Contents

- NLP application: Dependency parsing
  - Phrase structure, dependency structure, part-of-speech (POS) tagging
  - Before deep neural network: MaltParser
  - Neural approach
- NLP application: Text classification
- Convolutional Neural Network (CNN)
  - CNN for text classification

Convolutional Neural Nets

- Convolutional neural network (CNN, or ConvNet) is a type of feed-forward network where the individual neurons are tiled in such a way that they respond to overlapping regions (context windows) in the input.
- In other words, CNN is a type of feed-forward neural network with
  - Local connectivity (with the input layer)
  - Shared parameters across spatial/temporal positions
- Advantage over Feed-forward Neural Net (FNN)
  - Requires less parameters (sharing)
- Advantage over Recurrent Neural Net (next topic): Parallelizable

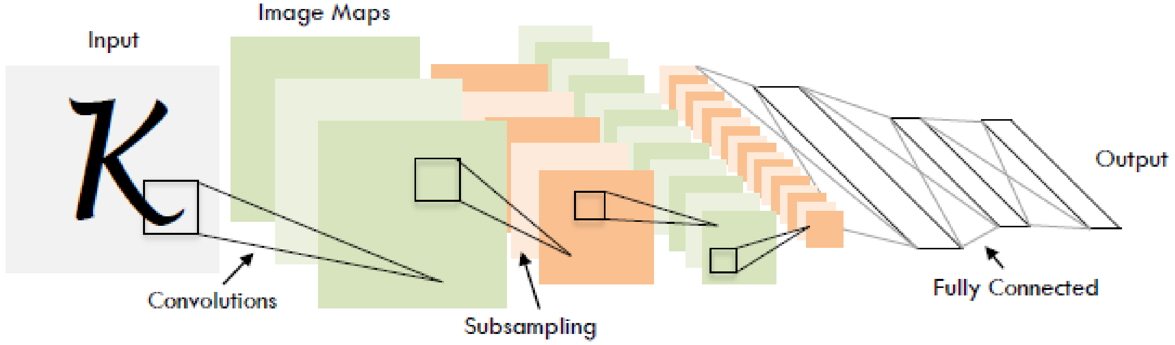
Source: NTUNLP

CNNs in Computer Vision

- CNNs were responsible for major breakthroughs in Image Classification (e.g., face recognition) and are the core of most Computer Vision systems today
  - This is also changing (see Transformers)

1998

LeCun et al.



# of transistors

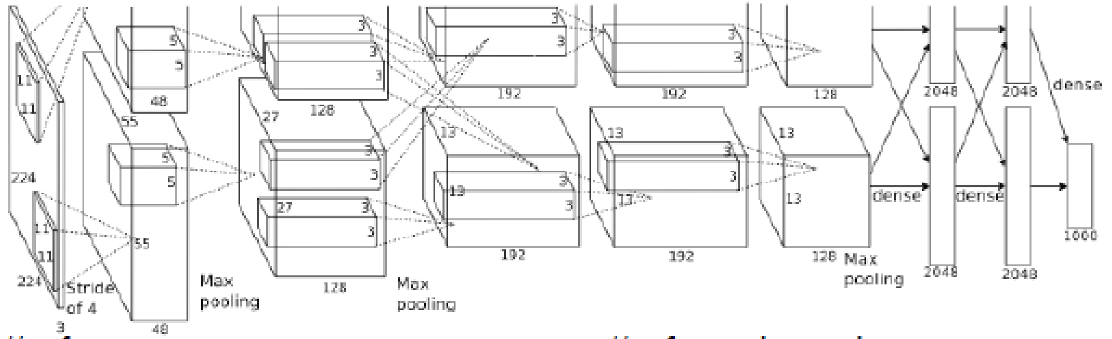
10^6

# of pixels used in training

10^7 NIST

2012

Krizhevsky et al.



# of transistors

10^9

GPUs

# of pixels used in training

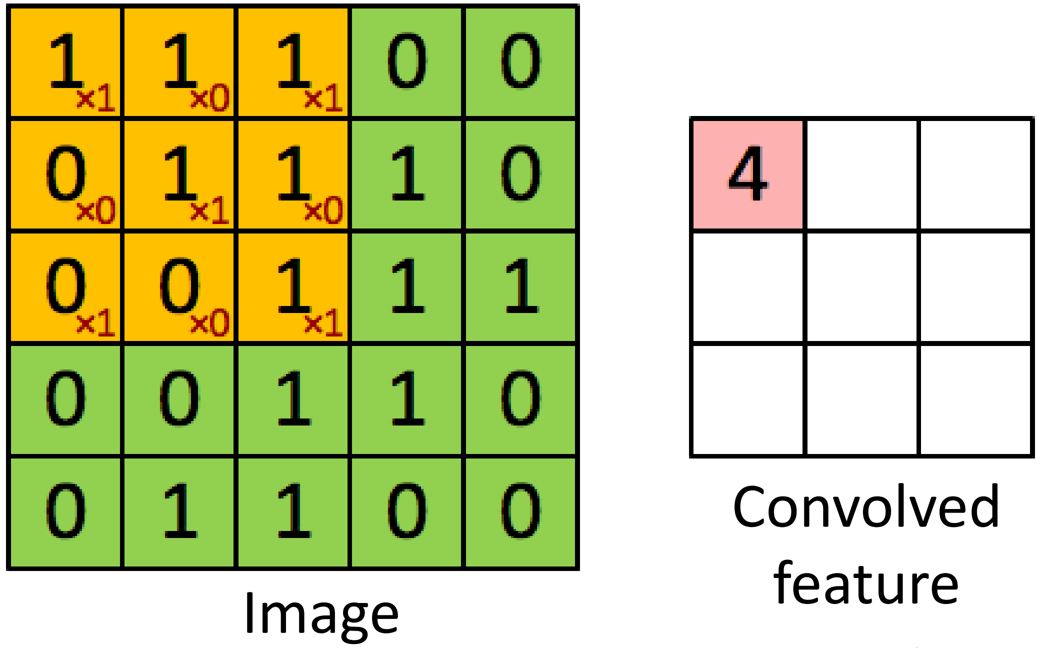
10^{14} IMAGENET

Source: CS231n

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

What is convolution?

- 1d discrete convolution generally: $$(f * g)[n] = \sum_{m=-M}^{M} f[n+m]g[m]$$
- Convolution is classically used to extract features from images
  - Models position-invariant identification
- 2d example →
  - Yellow color and red numbers show filter (=kernel) weights
  - Green shows input
  - Pink shows output



Source: CS224n

A 1D convolution for text

tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3

t,d,r	-1.1
d,r,t	-0.4
r,t,k	-3.9
t,k,g	-0.2
k,g,o	0.3

Apply a **filter** (or **kernel**) of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Source: CS224n

A 1D convolution for text with padding = 1

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	-0.6
t,d,r	-1.1
d,r,t	-0.4
r,t,k	-3.9
t,k,g	-0.2
k,g,o	0.3
g,o, Ø	-0.5

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Apply a **filter** of size 3

Source: CS224n

3 channel 1D convolution for text with padding = 1

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	-0.6	0.2	1.4
t,d,r	-1.1	1.6	-1.0
d,r,t	-0.4	-0.2	0.7
r,t,k	-3.9	0.2	0.2
t,k,g	-0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o, Ø	-0.5	-0.9	0.1

Apply 3 **filters** of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

1	0	0	1
1	0	-1	-1
0	1	0	1

1	-1	2	-1
1	0	-1	3
0	2	2	1

rce: CS224n

Conv1d, padded with max pooling over time

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

3	1	2	-3
-1	2	1	-3
1	1	-1	1

1	0	0	1
1	0	-1	-1
0	1	0	1

1	-1	2	-1
1	0	-1	3
0	2	2	1

∅,t,d	-0.6	0.2	1.4
t,d,r	-1.1	1.6	-1.0
d,r,t	-0.4	-0.2	0.7
r,t,k	-3.9	0.2	0.2
t,k,g	-0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o, ∅	-0.5	-0.9	0.1

Max p	0.3	1.6	1.4

Source: CS224n

Conv1d, padded with average pooling over time

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

3	1	2	-3
-1	2	1	-3
1	1	-1	1

1	0	0	1
1	0	-1	-1
0	1	0	1

1	-1	2	-1
1	0	-1	3
0	2	2	1

Ø,t,d	-0.6	0.2	1.4
t,d,r	-1.1	1.6	-1.0
d,r,t	-0.4	-0.2	0.7
r,t,k	-3.9	0.2	0.2
t,k,g	-0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o, Ø	-0.5	-0.9	0.1

Ave p	-0.91	0.23	0.50

Source: CS224n

Other notions: stride = 2

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

∅,t,d	-0.6	0.2	1.4
d,r,t	-0.4	-0.2	0.7
t,k,g	-0.2	0.1	1.2
g,o, ∅	-0.5	-0.9	0.1

Apply 3 **filters** of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
1	1	-1	1	0	1	0	1	0	2	2	1

rce: CS224n

Other notions: dilation = 2

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	-0.6	0.2	1.4
t,d,r	-1.1	1.6	-1.0
d,r,t	-0.4	-0.2	0.7
r,t,k	-3.9	0.2	0.2
t,k,g	-0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o, Ø	-0.5	-0.9	0.1

1,3,5	-0.6	-1.2
2,4,6	-1.2	-1.0
3,5,7	-4.6	-4.3

2	3	1
1	-1	-1
3	1	0

1	3	1
1	-1	-1
3	1	-1

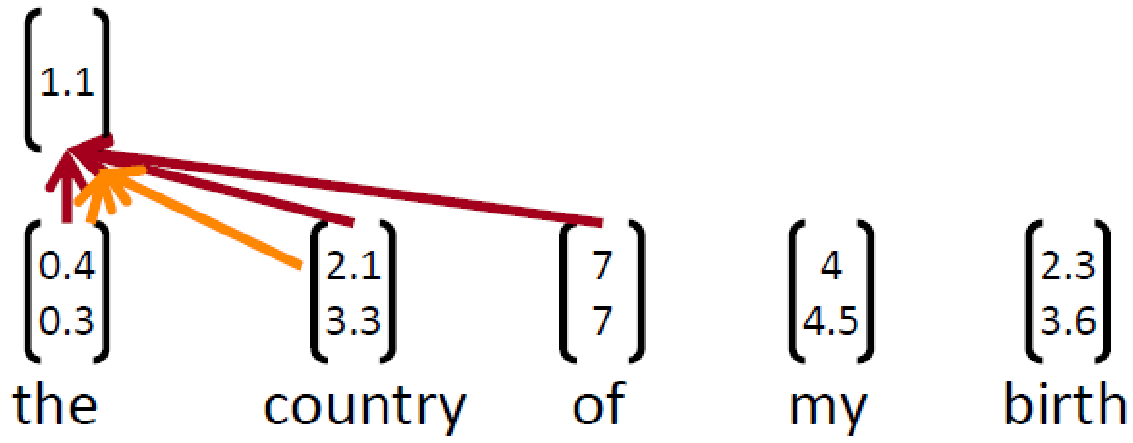
Source: CS224n

Contents

A CNN layer for text classification

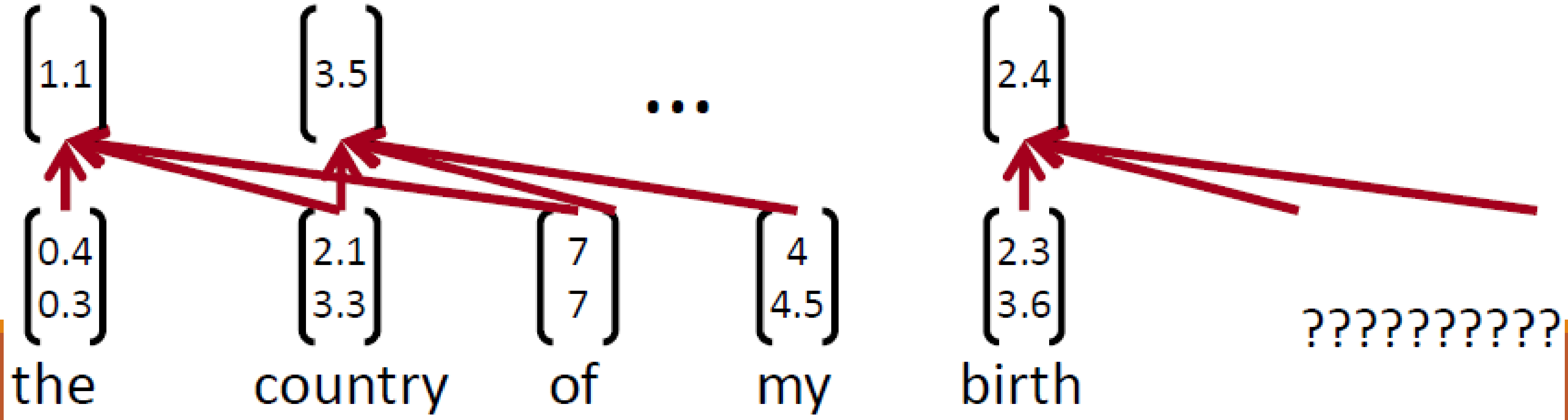
- A simple use of one convolutional layer and pooling
- Word vectors: $\boldsymbol{x}_i \in \mathbb{R}^k$
- Sentence: $\boldsymbol{x}_{1:n} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus \cdots \oplus \boldsymbol{x}_n$
- Concatenation of words in range: $\boldsymbol{x}_{i:i+j}$
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$

vectors concatenated over window of *h* words



A CNN layer for text classification

- Filter **w** is applied to all possible windows (concatenated vectors)
- All possible windows of length *h* in sentence $\boldsymbol{x}_{1:n}$: $\{\boldsymbol{x}_{1:h}, \boldsymbol{x}_{2:h+1}, \cdots, \boldsymbol{x}_{n-h+1:n}\}$
- To compute feature (one *channel*) for CNN layer: $c_i = f(\boldsymbol{w}^T \boldsymbol{x}_{i:i+h-1} + b)$
- Result is a feature map: $\boldsymbol{c} = [c_1, c_2, \cdots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Pooling and channels

- Pooling: max-over-time pooling layer
  - Idea: capture most important activation (maximum over time)
  - From feature map: $\boldsymbol{c} = [c_1, c_2, \cdots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
  - Pooled single number: $\hat{c} = \max\{\boldsymbol{c}\}$
- Use multiple filter weights **w** (i.e. multiple channels)
  - Useful to have different window sizes *h*
  - Because of max pooling, length of **c** irrelevant
  - So we could have some filters that look at unigrams, bigrams, tri-grams, etc
    - See definition of n-gram later in language modeling

Source: CS224n

Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification

- Window filter sizes h = 3, 4, 5
- Each filter size has 100 feature maps. In total, 300 feature maps
- Mini batch size for SGD training: 50
- Word vectors: pre-trained with word2vec, k = 300
- Nonlinearity: ReLU
- Dropout p = 0.5. Kim (2014) reports 2–4% accuracy improvement from dropout
- L_2 constraint s for rows of softmax, s = 3
- During training, keep checking performance on dev set and pick highest accuracy weights for final evaluation

A 1D convolution with bias and sigmoid

tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.2	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3

Apply a **filter** (or **kernel**) of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

t,d,r	-1.1
d,r,t	-0.4
r,t,k	-3.9
t,k,g	-0.2
k,g,o	0.3

⇨

-0.1	0.52
0.6	0.35
-2.9	0.95
0.8	0.31
1.3	0.21

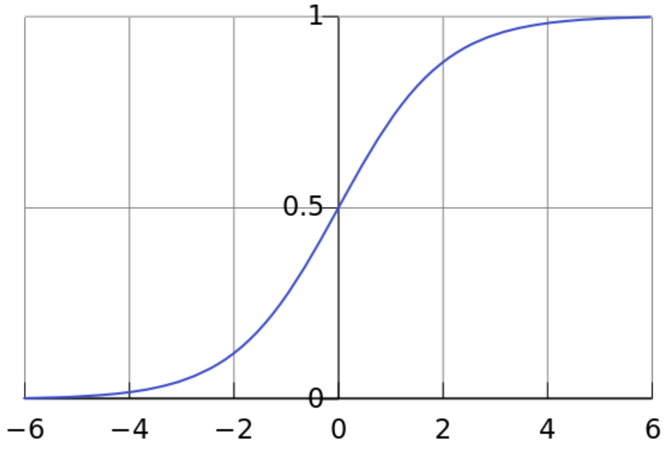
- bias (+1)
- non-linearity (sigmoid(-x))

Source: CS224n

Non-linearities: Starting points

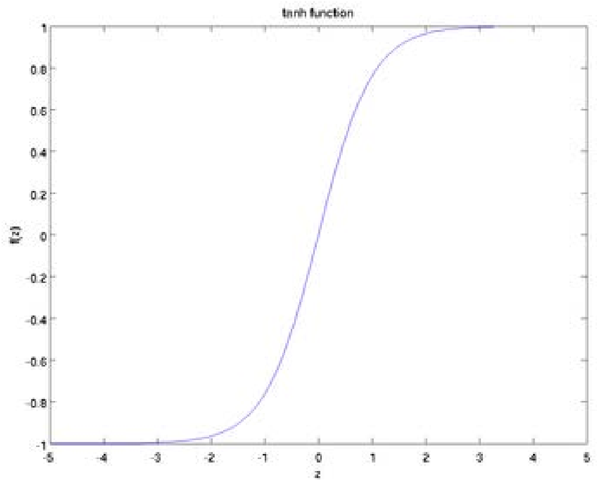
logistic ("sigmoid")

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



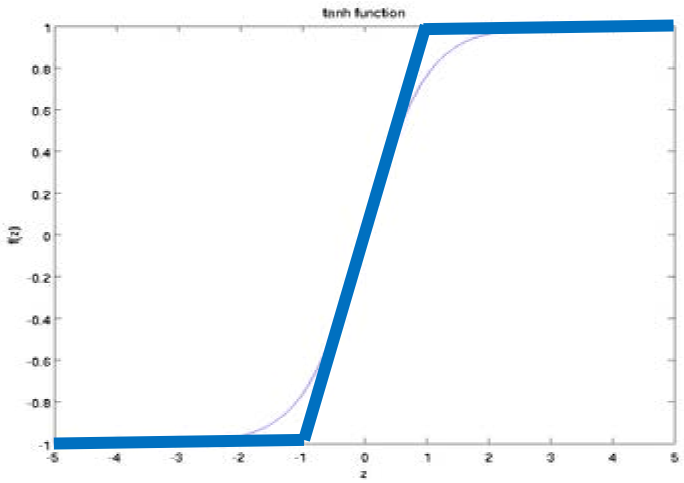
tanh

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



hard tanh

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

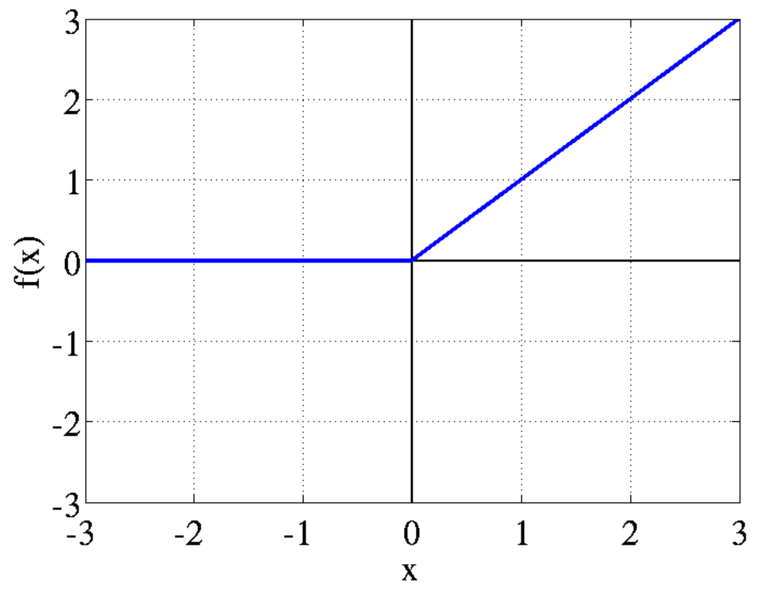


tanh is just a rescaled and shifted sigmoid (2x as steep, [-1, 1])

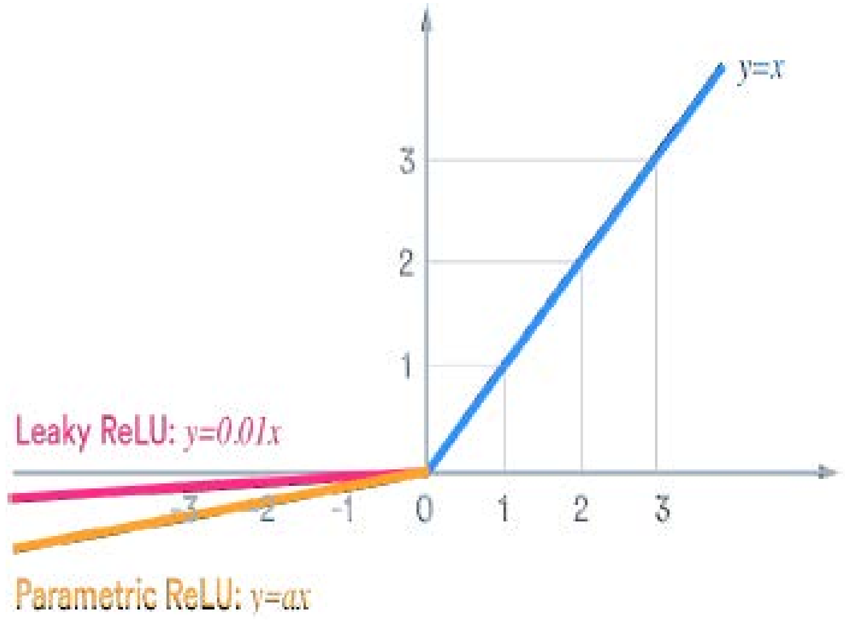
Source: CS224n

Non-linearities: Recent developments

ReLU (rectified linear unit)

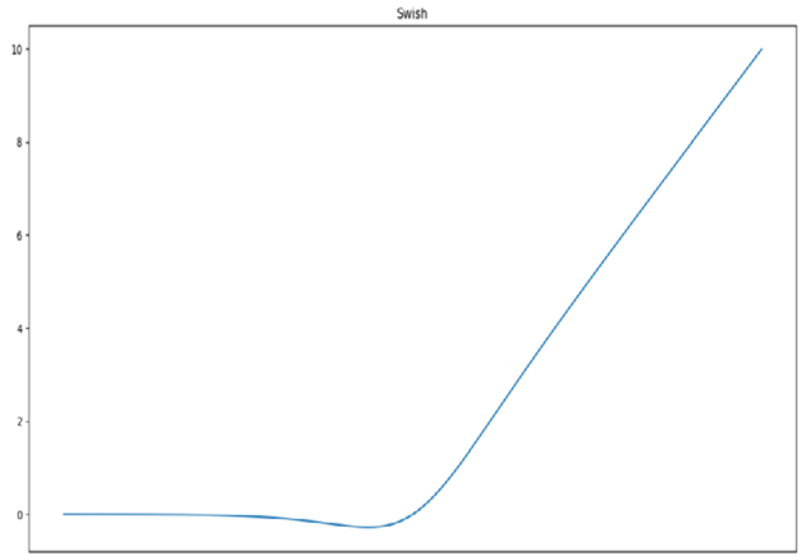


Leaky/parametric ReLU



Swish

https://arxiv.org/abs/1710.05941



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

Source: CS224n

Regularization: dropout

- Create masking vector *r* of Bernoulli random variables with probability *p* (a hyperparameter) of being 1
- Delete features during training: $y = \mathrm{softmax}\left(W^{(S)}(r \circ z) + b\right)$
  - E.g. randomly set 50% of the inputs to each neuron to 0, at each instance
- Reasoning: Prevents co-adaptation (overfitting to seeing specific feature constellations)
  - See the link below for details
- At test time, no dropout, scale down final vector by probability *p*: $\widehat{W}^{(S)} = pW^{(S)}$

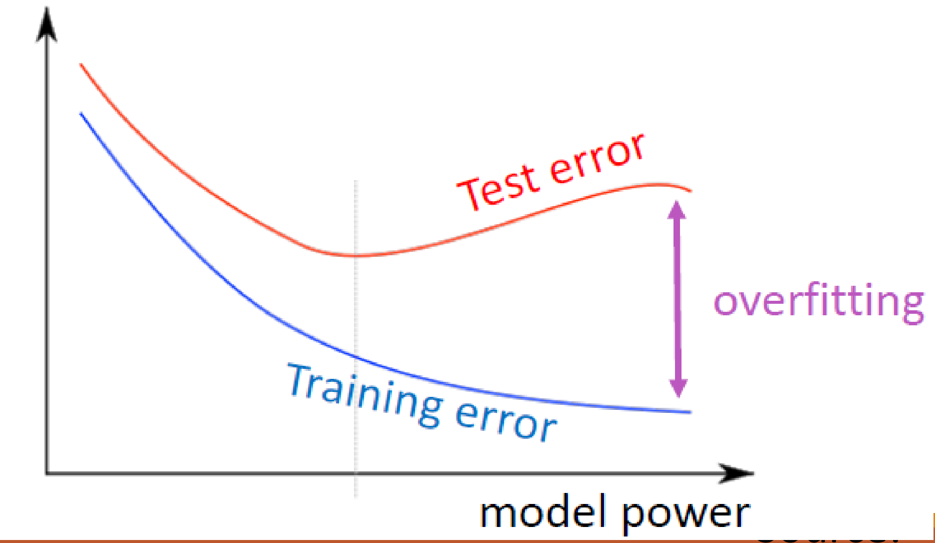
https://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

Source: CS224n

Regularization: l_2 norm

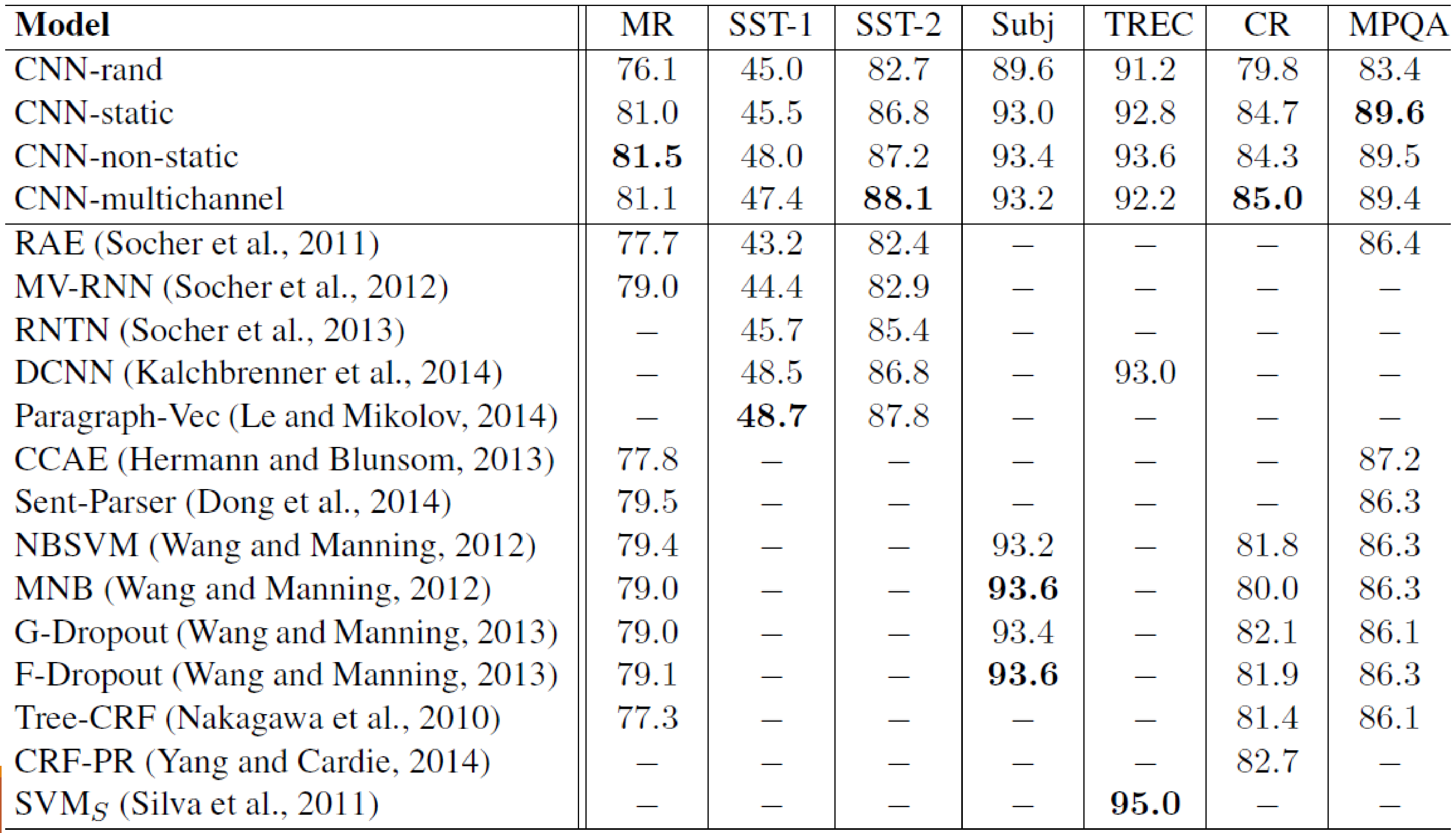
- Recall: $\|v\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2 + \cdots}$
- Constrain l_2 norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to be upper-bounded by a fixed number *s* (also a hyperparameter)
  - If $\left\|W_c^{(S)}\right\|_2 > s$, then rescale it so that: $\left\|W_c^{(S)}\right\|_2 = s$
- Or, add regularization to loss function
  - Helps to prevent overfitting when we have a lot of features

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



CS224n

Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification



Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM$_S$ (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Source: CS224n