의미 분석
(Semantic Analysis)

건국대학교 컴퓨터공학부 /
KAIST 전산학부 (겸직)

김학수

Core Layers of NLU

단계	설명	예제: 나는 그 과자를 먹었다.
형태소 분석	문장을 형태소열로 분리하고 품사를 부착하는 단계	나/대명사+는/조사 그/대명사 과자/명사+를/조사 먹/동사+었/선어말어미+다/어미+./기호
구문 분석	문장의 문법적 적합성과 어절의 구문적 역할(주어, 목적어 등)을 찾는 단계	[SUBJ: 나는 [[MOD: 그 [OBJ: 과자를]] 먹었다]]
의미 분석	문장을 구성하는 술어와 논항들 사이의 의미적 적합성을 분석하는 단계	PREDICATE: 먹다 AGENT: 나/ANIMATE OBJECT: 그 과자/EATABLE
담화 분석	대화 문맥을 파악하여 상호참조를 해결하고 의도를 파악하는 단계	SPEECH ACT: STATEMENT PREDICATE: 먹다 AGENT: 홍길동/ANIMATE OBJECT: 꼬깔콘/EATABLE

Semantics in NLP

- Computational Lexical Semantics
  - WSD (Word Sense Disambiguation)
  - Word Similarity
    - Distance between word vectors
  - SRL (Semantic Role Labeling)

Word Sense Disambiguation

Lexical Ambiguity

- Most words in natural languages have multiple possible meanings.
  - "pen" (noun)
    - The dog is in the pen.
    - The ink is in the pen.
  - "take" (verb)
    - Take one pill every morning.
    - Take the first right past the stoplight.
- Syntax helps distinguish meanings for different parts of speech of an ambiguous word.
  - "conduct" (noun or verb)
    - John's conduct in class is unacceptable.
    - John will conduct the orchestra on Thursday.

Word Sense Disambiguation (WSD)

- Given
  - A word in context,
  - A fixed inventory of potential word sense
- Decide which sense of the word this is.

Supervised Machine Learning Approaches

- Supervised machine learning approach:
  - A training corpus of words tagged in context with their sense
  - Used to train a classifier that can tag words in new text

- Summary of what we need:
  - The **tag set** (“sense inventory”)
  - The **training corpus**
  - A set of **features** extracted from the training corpus
  - A **classifier**

Supervised WSD 1: WSD Tags

Noun

- S: (n) **interest**, involvement (a sense of concern with and curiosity about someone or something) *"an interest in music"*
- S: (n) sake, **interest** (a reason for wanting something done) *"for your sake"; "died for the sake of his country"; "in the interest of safety"; "in the common interest"*
- S: (n) **interest**, interestingness (the power of attracting or holding one's attention (because it is unusual or exciting etc.)) *"they said nothing of great interest"; "primary colors can add interest to a room"*
- S: (n) **interest** (a fixed charge for borrowing money; usually a percentage of the amount borrowed) *"how much interest do you pay on your mortgage?"*
  - *direct hyponym* / *full hyponym*
  - ***direct hypernym*** / *inherited hypernym* / *sister term*
    - S: (n) fixed charge, fixed cost, fixed costs (a periodic charge that does not vary with business volume (as insurance or rent or mortgage payments etc.))
- S: (n) **interest**, stake ((law) a right or legal share of something; a financial involvement with something) *"they have interests all over the world"; "a stake in the company's future"*
- S: (n) **interest**, interest group ((usually plural) a social group whose members control some field of activity and who have common aims) *"the iron interests stepped up production"*
- S: (n) pastime, **interest**, pursuit (a diversion that occupies one's time and thoughts (usually pleasantly)) *"sailing is her favorite pastime"; "his main pastime is gambling"; "he counts reading among his interests"; "they criticized the boy for his limited pursuits"*

Verb

- S: (v) **interest** (excite the curiosity of; engage the interest of)
- S: (v) concern, **interest**, occupy, worry (be on the mind of) *"I worry about the second Germanic consonant shift"*
- S: (v) matter to, **interest** (be of importance or consequence) *"This matters to me!"*

WordNet

배[3] ★★★

[명사] **배**나무의 열매.

배[1] ★★★

[명사]

1. <의학>사람이나 동물의 몸에서 위장, 창자, 콩팥 따위의 내장이 들어 있는 곳으로 가슴...
2. <동물>절족동물, 특히 곤충에서 머리와 가슴이 아닌 부분. 여러 마디로 되어 있으며 숨...
3. 긴 물건 가운데의 볼록한 부분.

유의어 : 복부[1], 중배[1], 앞배[1]

배[2] ★★★

[명사] 사람이나 짐 따위를 싣고 물 위로 떠다니도록 나무나 쇠 따위로 만든 물건. 모양과 쓰임에 따라 보트, 나룻**배**, 기선(汽船), 군함(軍艦), 화물선, 여객선, 유조선 따위로 나눈다.

유의어 : 범택, 선박[2], 주선[3]

배[9] (倍) [배 :] ★

[명사]

1. 어떤 수나 양을 두 번 합한 만큼.
2. 일정한 수나 양이 그 수만큼 거듭됨을 이르는 말.

국어사전

Supervised WSD 2: Get a Corpus

- Lexical sample task:
  - *Line-hard-serve* corpus - 4000 examples of each
  - *Interest* corpus - 2369 sense-tagged examples

- All words:
  - **Semantic concordance**: a corpus in which each open-class word is labeled with a sense from a specific dictionary/thesaurus.
    - SemCor: 234,000 words from Brown Corpus, manually tagged with WordNet senses
    - SENSEVAL-3 competition corpora - 2081 tagged word tokens

Supervised WSD 3: Extract Feature Vectors

- Weaver (1955)

  If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. [...] But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word. [...] The practical question is : "What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?"

Two Kinds of Features in the Vectors

- **Collocational** features and **bag-of-words** features
  - **Collocational features**
    - Features about words at specific positions near target word: Often limited to just word identity and POS
  - **Bag-of-words features**
    - Features about words that occur anywhere in the window (regardless of position): Typically limited to frequency counts

Examples

- Example text (WSJ)
  - An electric guitar and **bass** player stand off to one side not really part of the scene, just as a sort of nod to gringo expectations perhaps
  - Assume a window of +/- 2 from the target

Examples

- Example text
  - An electric guitar and **bass** player stand off to one side not really part of the scene, just as a sort of nod to gringo expectations perhaps
  - Assume a window of +/- 2 from the target

Collocational Features

- Position-specific information about the words in the window

guitar and bass player stand

  - [guitar, NN, and, CC, player, NN, stand, VB]
    $\text{Word}_{n-2,}$ $\text{POS}_{n-2,}$ $\text{word}_{n-1,}$ $\text{POS}_{n-1,}$ Word_{n+1} POS_{n+1}…
  - In other words, a vector consisting of
    [position n word, position n part-of-speech…]

Bag-Of-Words Features

- Information about the words that occur within the window.
- First derive a set of terms to place in the vector.
- Then note how often each of those terms occurs in a given window.

Co-Occurrence Example

- Assume we've settled on a possible vocabulary of 12 words that includes guitar and player but not and and stand

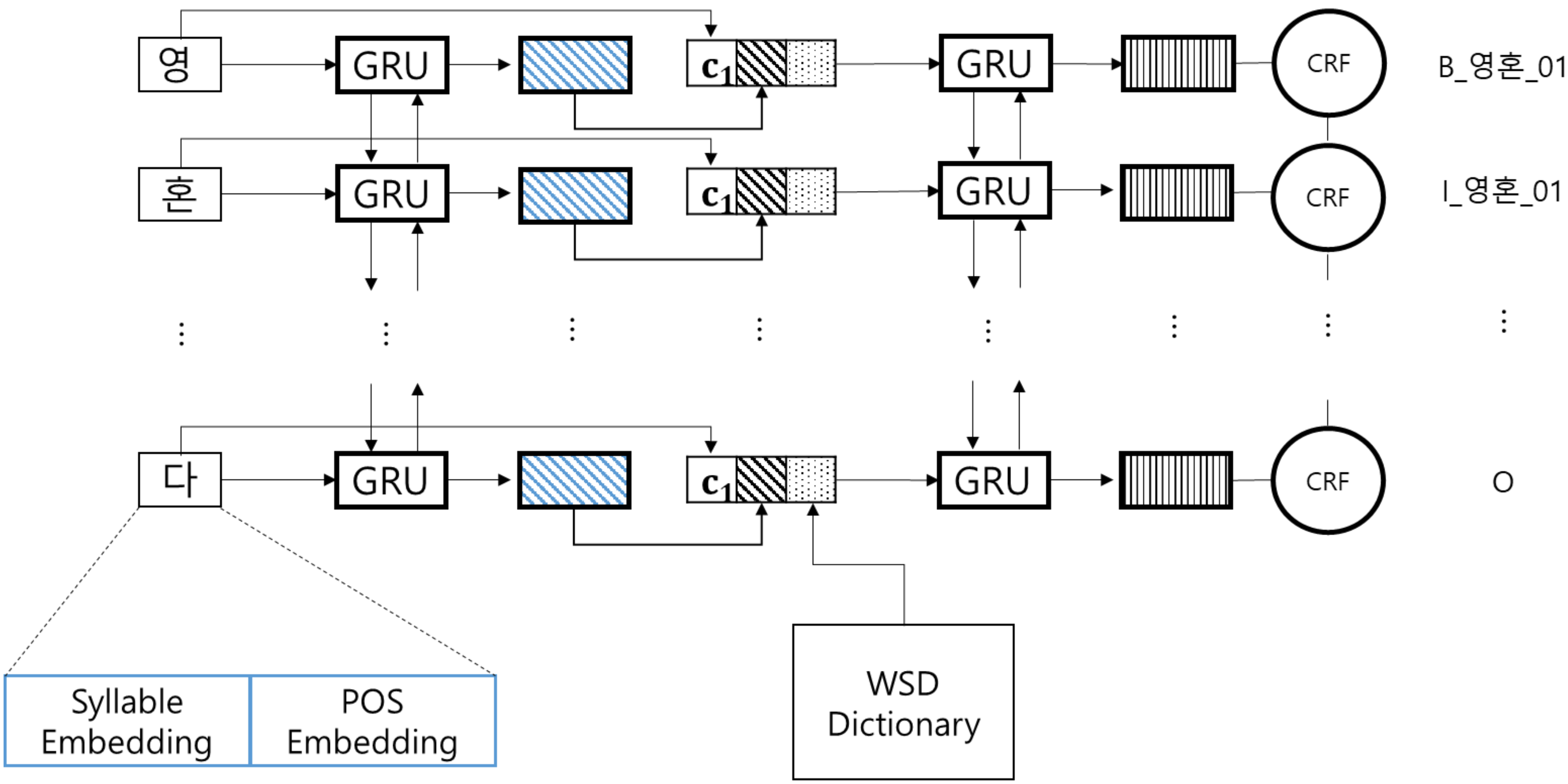
guitar and bass player stand

  - [0,0,0,1,0,0,0,0,0,1,0,0]
  - Which are the counts of words predefined [fish,fishing,viol, guitar, double,cello…]

Classifiers

- Once we cast the WSD problem as a classification problem, then all sorts of techniques are possible
  - Naïve Bayes (the easiest thing to try first)
  - Decision lists
  - Decision trees
  - Neural nets
  - Support vector machines
  - Nearest neighbor methods…

Recent Deep Learning Model



Word Similarity

Word Similarity

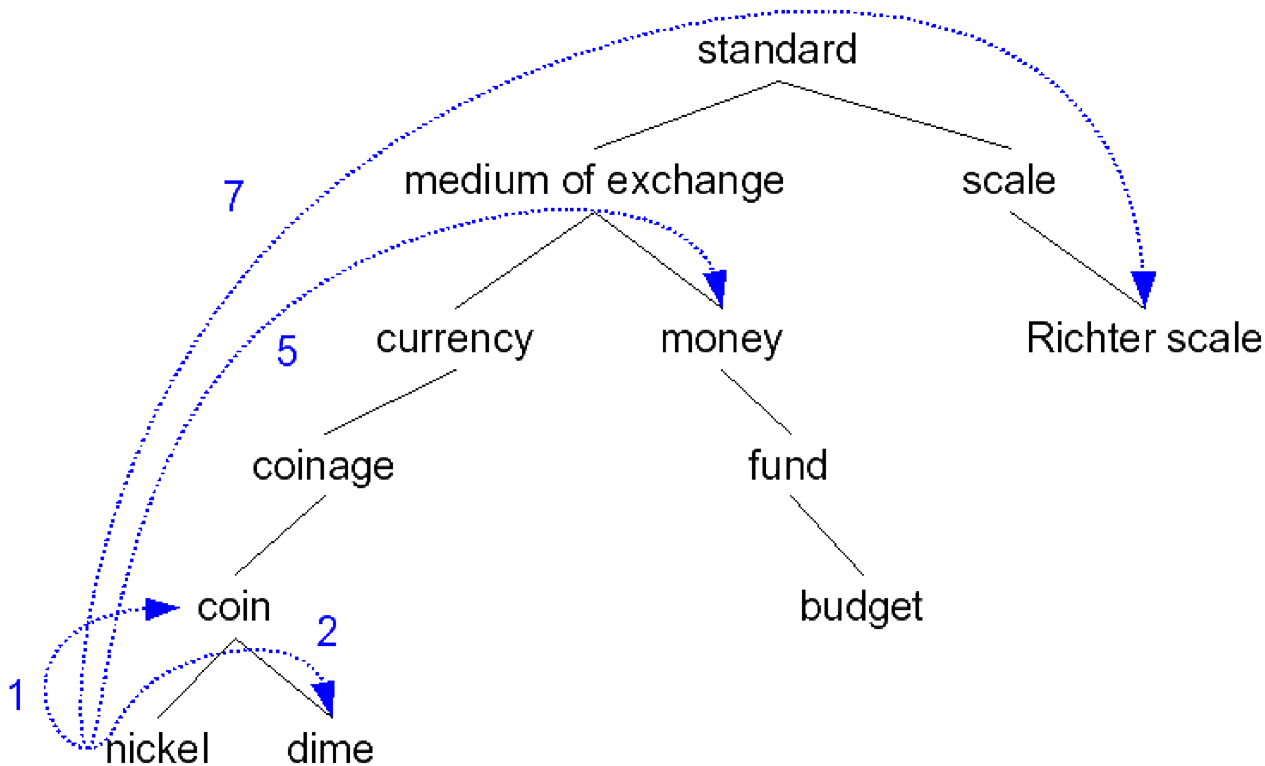
- Synonymy is a binary relation
  - Two words are either synonymous or not
- We want a looser metric
  - Word similarity or Word distance
- Two words are more similar
  - If they share more features of meaning
- Actually these are really relations between **senses**:
  - Instead of saying “bank is like fund”
  - We say
    - Bank1 is similar to fund3
    - Bank2 is similar to slope5
- We’ll compute them over both words and senses

Two Classes of Algorithms

- Thesaurus-based algorithms
  - Based on whether words are "nearby" in Wordnet or MeSH
- Distributional algorithms
  - By comparing words based on their distributional context

Thesaurus-based algorithms: Path based Similarity

- Two words are similar if nearby in thesaurus hierarchy (i.e. short path between them)



Refinements to Path-based Similarity

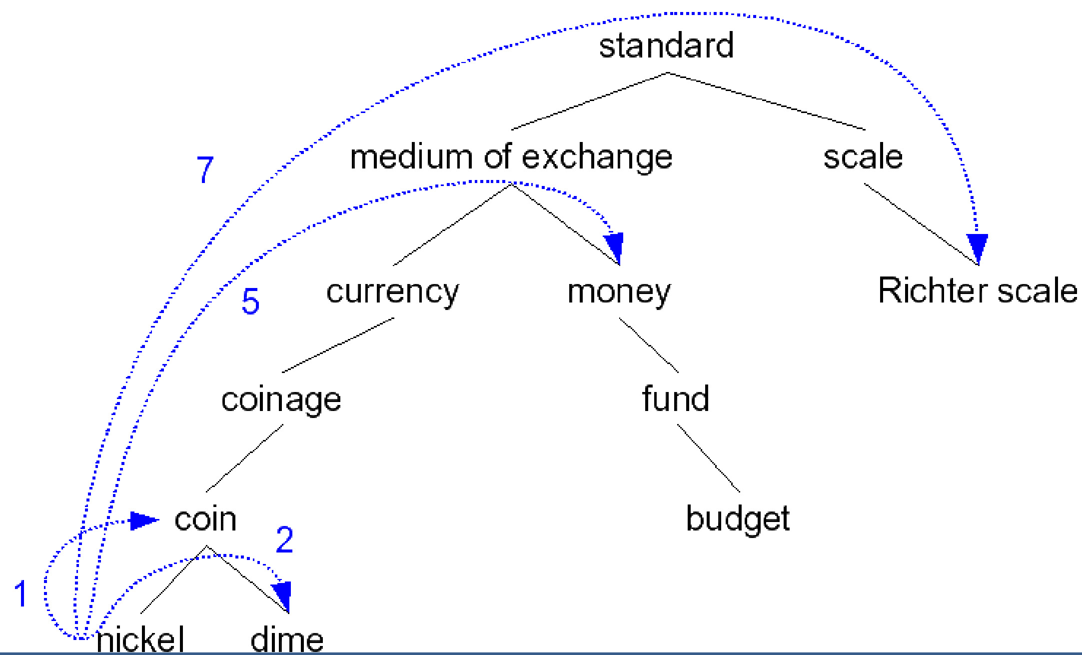
- *pathlen(c1,c2)* = number of edges in the shortest path in the thesaurus graph between the sense nodes *c1* and *c2*

- $simpath(c1,c2) = -\log pathlen(c1,c2)$

- $wordsim(w1,w2) = \max_{c1 \in senses(w1), c2 \in senses(w2)} simpath(c1,c2)$

Problem with Basic Path-based Similarity

- Assumes each link represents a uniform distance
  - *Nickel* to *money* seem closer than *nickel* to *standard*
  - Represent the cost of each edge independently



Problems with Thesaurus-based Methods

- We don't have a thesaurus for every language
- Even if we do, many words are missing
- They rely on hyponym info:
  - Strong for nouns, but lacking for adjectives and even verbs

- Alternative
  - Distributional methods for word similarity

Distributional Methods for Word Similarity

- Firth (1957): "You shall know a word by the company it keeps!"
- Nida example noted by Lin:
  - A bottle of ***tezgüino*** is on the table
  - Everybody likes ***tezgüino***
  - ***Tezgüino*** makes you drunk
  - We make ***tezgüino*** out of corn.
- Intuition:
  - just from these contexts a human could guess meaning of tezguino
  - So we should look at the surrounding contexts, see what other words have similar context.

Co-occurrence Vectors

- Define two words by these sparse features vectors
- Apply a vector distance metric
- Say that two words are similar if two vectors are similar

	arts	boil	data	function	large	sugar	summarized	water
apricot	0	1	0	0	1	1	0	1
pineapple	0	1	0	0	1	1	0	1
digital	0	0	1	1	1	0	1	0
information	0	0	1	1	1	0	1	0

Binary vs. Frequency vs. PMI

Object	Count	PMI assoc	Object	Count	PMI assoc
bunch beer	2	12.34	wine	2	9.34
tea	2	11.75	water	7	7.65
Pepsi	2	11.75	anything	3	5.15
champagne	4	11.75	much	3	5.15
liquid	2	10.53	it	3	1.25
beer	5	10.20	<SOME AMOUNT>	2	1.22

- "drink it" is more common than "drink wine"
- But "wine" is a better "drinkable" thing than "it"
- Idea:
  - We need to control for change (expected frequency)
  - We do this by normalizing by the expected frequency we would get assuming independence

Weighting: Mutual Information

- **Mutual information:** between 2 random variables X and Y

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

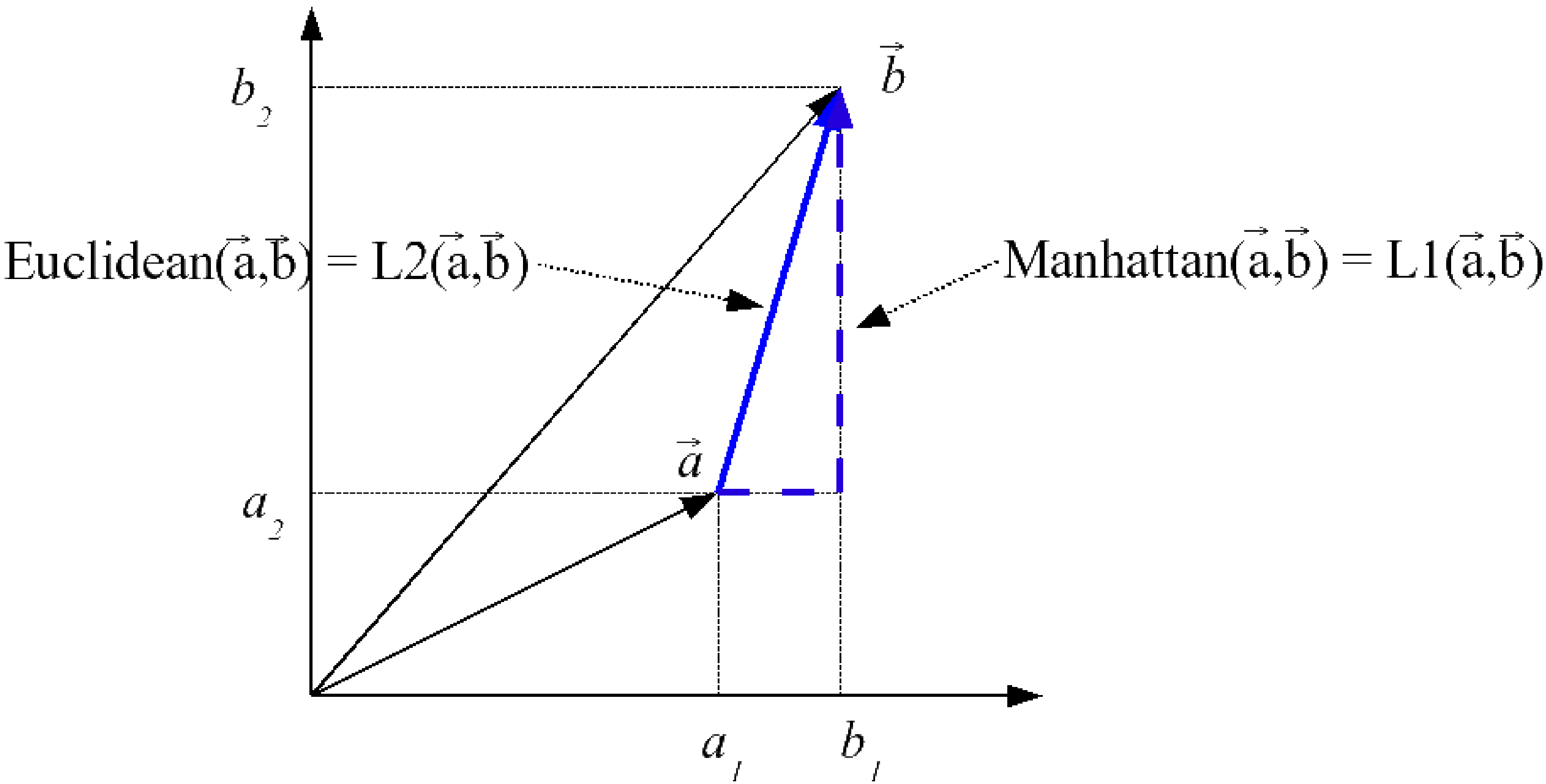
- **Pointwise mutual information**: measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

Now!

- One-hot representation → Distributed representation
  - Word2Vec
  - fastText
  - GloVe
  - CoVe
  - ELMo

Defining Similarity between Vectors



Summary of Similarity Measures

$$\mathrm{sim}_{\mathrm{cosine}}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

$$\mathrm{sim}_{\mathrm{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} \max(v_i, w_i)}$$

$$\mathrm{sim}_{\mathrm{Dice}}(\vec{v},\vec{w}) = \frac{2 \times \sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} (v_i + w_i)}$$

$$\mathrm{sim}_{\mathrm{JS}}(\vec{v}||\vec{w}) = D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2})$$

Semantic Role Labeling

Semantic Role Labeling (SRL)

- For each clause, determine the semantic role played by each noun phrase that is an argument to the verb.
  agent patient source destination instrument
  - John drove Mary from Austin to Dallas in his Toyota Prius.
  - The hammer broke the window.

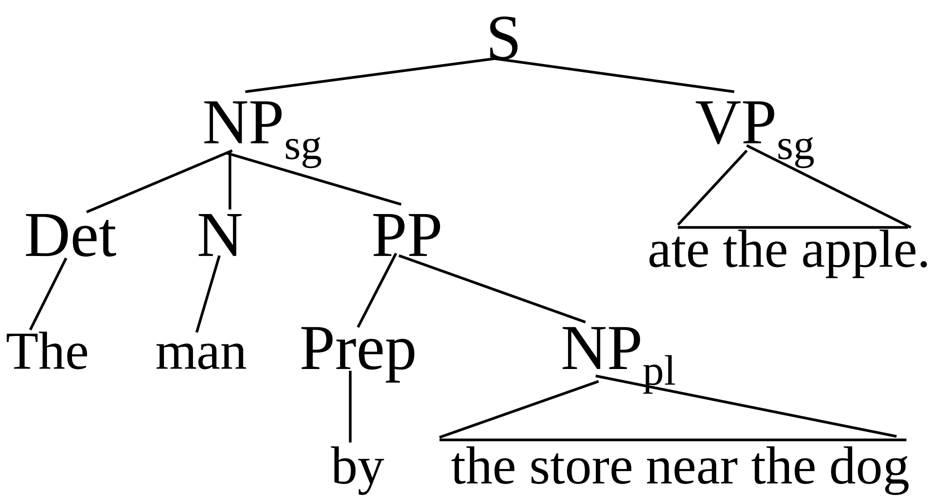
- Also referred to a "case role analysis," "thematic analysis," and "shallow semantic parsing"

Semantic Roles

- Origins in the linguistic notion of case (Fillmore, 1968)
- A variety of semantic role labels have been proposed, common ones are:
  - Agent: Actor of an action
  - Patient: Entity affected by the action
  - Instrument: Tool used in performing action.
  - Beneficiary: Entity for whom action is performed
  - Source: Origin of the affected entity
  - Destination: Destination of the affected entity

SRL with Parse Trees

- Parse trees help identify semantic roles through exploiting syntactic clues like "the agent is usually the subject of the verb".
- Parse tree is needed to identify the true subject.



"The man by the store near the dog ate an apple."

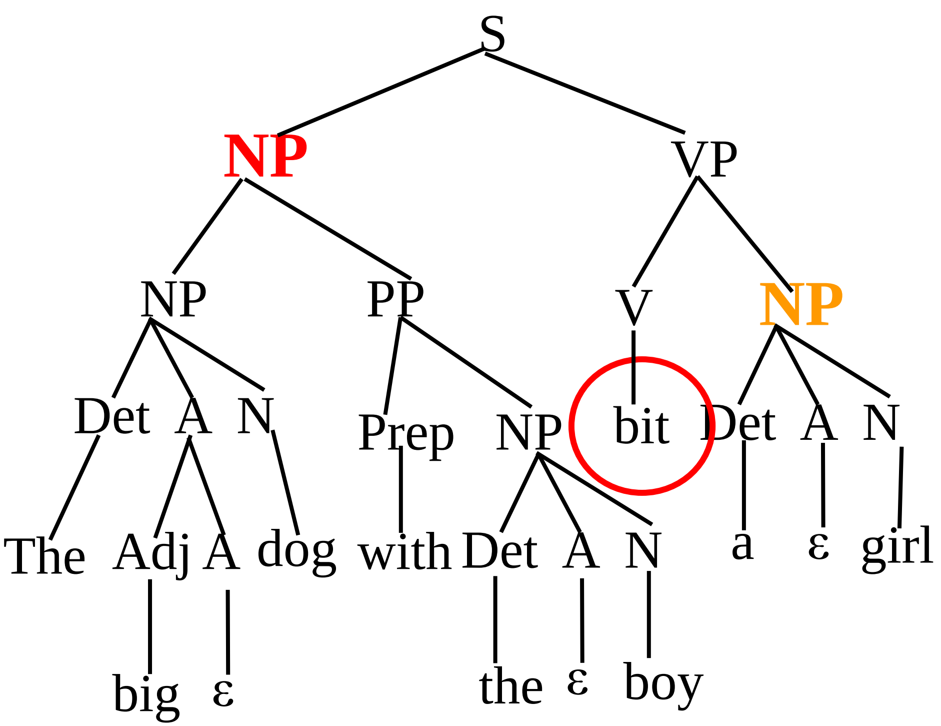
"The man" is the agent of "ate" not "the dog".

SRL with Parse Trees

- Assume that a syntactic parse is available.
- For each predicate (verb), label each node in the parse tree as either not-a-role or one of the possible semantic roles.

Color Code:

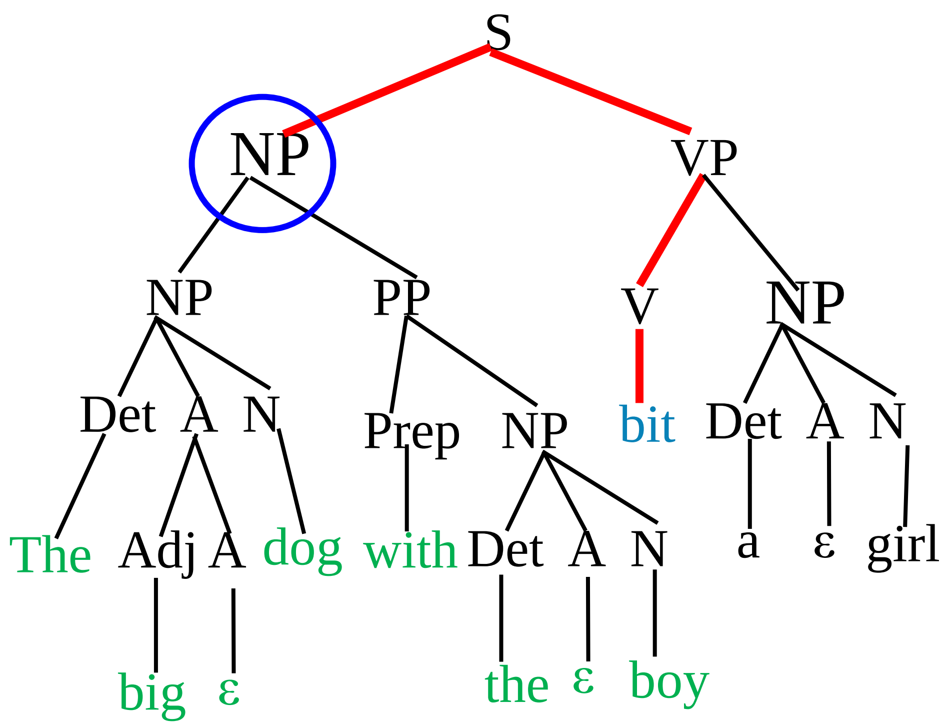
not-a-role
agent
patient
source
destination
instrument
beneficiary



Features for SRL

- **Phrase type**: The syntactic label of the candidate role filler (e.g. NP).
- **Parse tree path**: The path in the parse tree between the predicate and the candidate role filler.
- **Position**: Does candidate role filler ***precede*** or ***follow*** the predicate in the sentence?
- **Voice**: Is the predicate an ***active*** or ***passive*** verb?
- **Head Word**: What is the head word of the candidate role filler?

Complete SRL Example



Phrase type	Parse Path	Position	Voice	Head word
NP	**V↑VP↑S↓NP**	precede	active	dog

SRL Datasets

- FrameNet:
  - Developed at Univ. of California at Berkeley
  - Based on notion of Frames
- PropBank:
  - Developed at Univ. of Pennsylvania
  - Based on elaborating their Treebank

FrameNet

- Project at UC Berkeley led by Chuck Fillmore for developing a database of frames, general semantic concepts with an associated set of roles.
- Roles are specific to frames, which are "invoked" by multiple words, both verbs and nouns.
  - JUDGEMENT frame
    - Invoked by V: blame, praise, admire; N: fault, admiration
    - Roles: JUDGE, EVALUEE, and REASON
- Specific frames chosen, and then sentences that employed these frames selected from the British National Corpus and annotated by linguists for semantic roles.
- Initial version: 67 frames, 1,462 target words, 49,013 sentences, 99,232 role fillers

PropBank

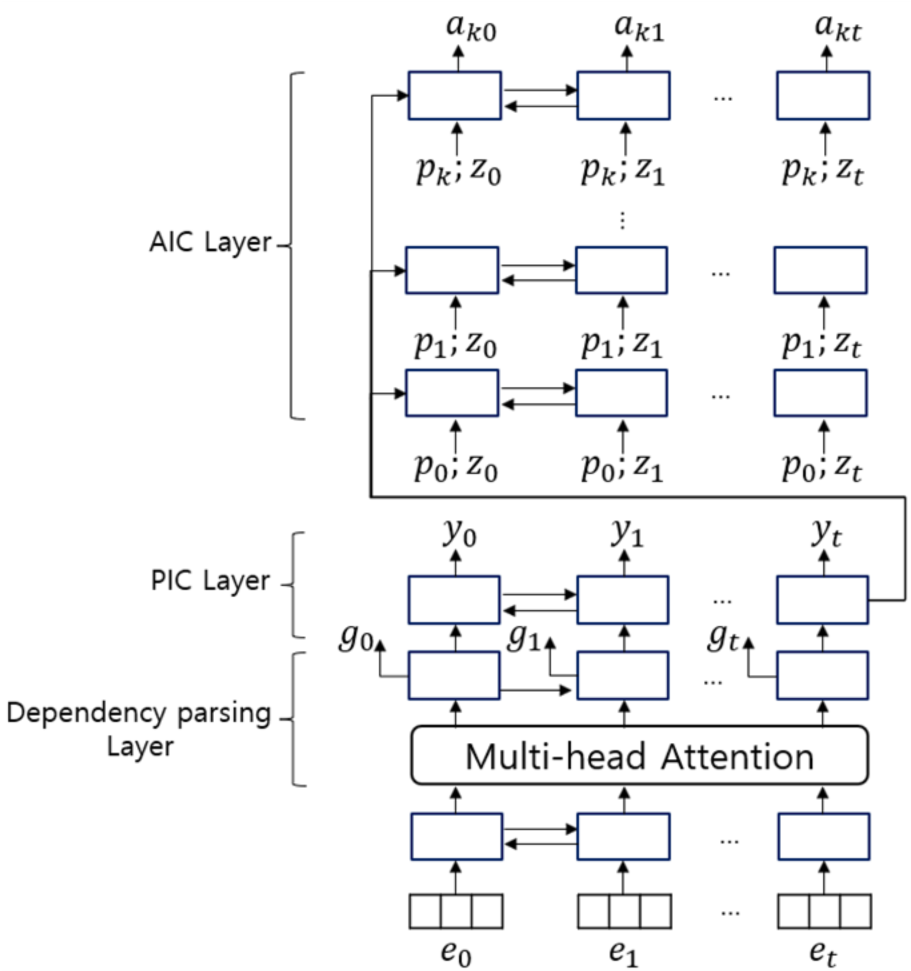
- Project at U Penn lead by Martha Palmer to add semantic roles to the Penn treebank.
- Roles (Arg0 to ArgN) specific to each individual verb to avoid having to agree on a universal set.
  - Arg0 basically "agent"
  - Arg1 basically "patient"
- Annotated over 1M words of Wall Street Journal text with existing gold-standard parse trees.
- Statistics:
  - 43,594 sentences 99,265 propositions (verbs + roles)
  - 3,324 unique verbs 262,281 role assignments

구문-의미 통합 분석 모델

- 의존 구문 분석과 의미역 결정 사이에는 상관 관계 존재
  - 서술어와 논항사이에 직간접적 의존 관계가 존재하는 경우가 빈번
  - 어절이 갖는 의존 관계와 의미역이 비슷한 역할을 가짐

	나도	미소를	띄었다.
의존 관계	NP_SBJ	NP_OBJ	VP
	주어	목적어	용언구
서술어, 논항	ARG0	ARG1	Predicate
	주체	대상	서술어

구문-의미 통합 분석 모델 구조도



의미역 부착 계층

- 서술어 예측값 y_t, 서술어 어절 임베딩 p_k, 각 계층(입력층~구문분석층) 출력 연결 벡터 z_t
- 순환 신경망 병렬화를 통한 다중 라벨링으로 논항 예측 a_{kt}

의존구조 분석 계층

- 어절 임베딩 e_t, 지배소 위치 g_t, 의존 관계명 l_t
- 멀티헤드 어텐션과 포인터 네트워크 기반 의존 구문 분석 (박성식 외, 2018)

Experimental Results

- 실험 말뭉치
  - 학습 데이터 : UPropBank 11만 문장
  - 평가 데이터 : UPropBank 2만 문장

학습 방법	F1-Score (PIC)	F1-Score (AIC)
SRL Only	0.9949	**0.7312**
DP + SRL	**0.9952**	0.7255

구문-의미 통합 분석 모델 시연 영상


```
keep prob:  1    dropout prob:  0.0
Reading model parameters from ./model/analyzer/59_model.ckpt
Success Load!
문장 입력:산업 발전을 위한 인력 정책의 하나로 인력의 고급화와 교육의 기회균등 방안을 마련하고 고급 기술 개발을 위한 기술 교육 정책을 강화할 계획이다.
산업 발전을 위한 인력 정책의 하나로 인력의 고급화와 교육의 기회균등 방안을 마련하고 고급 기술 개발을 위한 기술 교육 정책을 강화할 계획이다.
산업 발전을 위한 인력 정책의 하나로 인력의 고급화와 교육의 기회균등 방안을 마련하고 고급 기술 개발을 위한 기술 교육 정책을 강화할 계획이다.
1   산업     -   0    0    0    0
2   발전을   -   ARG1 0    0    0
3   위한     위한 0   0    0    0
4   인력     -   0    0    0    0
5   정책의   -   ARG0 0    0    0
6   하나로   -   ARG0 ARGM-PRD  0  ARGM-PRD
7   인력의   -   0    0    0    0
8   고급화와 -   0    0    0    0
9   교육의   -   0    0    0    0
10  기회균등 -   0    0    0    0
11  방안을   -   0    ARG1 0    0
12  마련하고 마련하고 0 0  0    0
13  고급     -   0    0    0    0
14  기술     -   0    0    0    0
15  개발을   -   0    0    ARG1 0
16  위한     위한 0   0    0    0
17  기술     -   0    0    0    0
18  교육     -   0    0    0    0
19  정책을   -   0    0    ARG0 ARG1
20  강화할   강화할 0  0    0    0
21  계획이다.  -  0    0    0    0


문장 입력:
```

의미 분석
(Semantic Analysis)

Practical Exercise

실습: WSD

예제 코드 다운로드:
https://github.com/KUNLP/Lecture

- 문맥 정보가 반영된 벡터 표현 비교해보기
  - 보기만 해도 배가 부르다.
  - 점심을 먹지 못해 배가 많이 고팠다.
  - 배 한 척이 바다 한가운데 떠 있다.
  - 그 섬에는 하루에 두 번씩 배가 들어온다.
  - 나는 과일 중에서 배를 가장 좋아한다.
  - 사각사각 씹히는 배의 맛이 달고 시원하다.

- BERT를 사용하여 서로 다른 문장에서 사용된 "배"에 대한 벡터 표현을 구하고 유사도 비교

실습: WSD

모델 설계

```python
from transformers import BertPreTrainedModel, BertModel


class WSD(BertPreTrainedModel):

    def __init__(self, config):
        super(WSD, self).__init__(config)

        # BERT 모델
        self.bert = BertModel(config)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)

        # BERT 출력 (batch_size, max_length, hidden_size)
        bert_output = outputs[0]

        return bert_output
```

BERT 언어모델 사용

BERT 사전학습 모델
생성자 오버라이딩

Self-attention 범위
지정을 위한 마스크

last_hidden_state (`torch.FloatTensor` of shape `(batch_size, sequence_length, hidden_size)`):

Sequence of hidden-states at the output of the last layer of the model.

실습: WSD

데이터 읽기

```python
def read_data(file_path):
    with open(file_path, "r", encoding="utf8") as inFile:
        lines = inFile.readlines()

    datas = []
    for line in lines:
        # 입력 데이터를 \t을 기준으로 분리
        pieces = line.strip().split("\t")

        # 입력 토큰 시퀀스, 목표 토큰 인덱스
        token_sequence, target_token_index = pieces[0].split(" "), int(pieces[1])

        datas.append((token_sequence, target_token_index))
    return datas
```

WSD 대상 토큰의 인덱스
문자열을 숫자로 변환

BERT 토크나이저 사용
(sentecepiece)

```
_보기 만 _해도 _배 가 _부르 다 .	3
_점 심을 _먹 지 _못해 _배 가 _많이 _고 팠 다 .	5
_배 _한 _ 척 이 _바다 _한 가 운 데 _떠 _있다 .	0
_그 _섬 에는 _하루 에 _두 _번 씩 _배 가 _들어 온 다 .	8
_나는 _ 과 일 _중 에서 _배 를 _가장 _좋아 한다 .	6
_사 각 사 각 _ 씹 히 는 _배 의 _맛 이 _달 고 _시 원 하다 .	8
```

WSD 대상 토큰의 인덱스

실습: WSD

데이터 전처리

```
def convert_data2feature(datas, max_length, tokenizer):
    input_ids_features, attention_mask_features = [], []
```

_보기 만 _해도 _배 가 _부르 다 .

3

```
for token_sequence target_token_index in datas:

    # CLS, SEP 토큰 추가
    tokens = [tokenizer.cls_token]
    tokens += token_sequence
    tokens = tokens[:max_length - 1]
    tokens += [tokenizer.sep_token]

    # word piece들을 대응하는 index로 치환
    input_ids = tokenizer.convert_tokens_to_ids(tokens)

    # padding을 제외한 실제 데이터 정보를 반영해주기 위한 attention mask
    attention_mask = [1] * len(input_ids)

    # padding 생성
    padding = [tokenizer._convert_token_to_id(tokenizer.pad_token)] * (max_length - len(input_ids))
    padding_for_mask = [0] * (max_length - len(input_ids))

    # padding 추가
    input_ids += padding
    attention_mask += padding_for_mask

    # 변환한 데이터를 각 리스트에 저장
    input_ids_features.append(input_ids)
    attention_mask_features.append(attention_mask)

return input_ids_features, attention_mask_features
```

정해진 max length 보다 길면 삭제 (보통 512)

```
tokens : ['[CLS]', '_보기', '만', '_해도', '_배', '가', '_부르', '다', '.', '[SEP]']
input_ids : [2, 2362, 6150, 5002, 2287, 5330, 2432, 5782, 54, 3, ...]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, ...]
```

실습: WSD

Test | 모델 선언 및 학습 데이터 로드

```
def test(config):

    # BERT config 객체 생성
    bert_config = BertConfig.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],
                                             cache_dir=config["cache_dir_path"])

    # BERT tokenizer 객체 생성
    bert_tokenizer = KoBertTokenizer.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],
                                                     cache_dir=config["cache_dir_path"])

    # 데이터 읽기
    datas = read_data(file_path=config["data_path"])

    # 데이터 전처리
    input_ids_features, attention_mask_features = convert_data2feature(datas=datas,
                                                                       max_length=config["max_length"],
                                                                       tokenizer=bert_tokenizer)

    # 사전 학습된 BERT 모델 파일로부터 가중치 불러옴
    model = WSD.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],
                                cache_dir=config["cache_dir_path"], config=bert_config).cuda()
```

실습: WSD

Test 벡터 추출 및 데이터 후처리

```python
# 모델 예측 결과
bert_outputs = model(input_ids_features, attention_mask_features)

input_ids_features = input_ids_features.cpu().detach().numpy().tolist()
bert_outputs = bert_outputs.cpu().detach().numpy().tolist()
```

```python
# batch 단위로 구성되어 있어 반복문을 통해 하나씩 확인
word2vec = {}
for batch_index in range(len(bert_outputs)):
    input_tokens = bert_tokenizer.convert_ids_to_tokens(input_ids_features[batch_index])
    bert_output = bert_outputs[batch_index]

    token_sequence, target_token_index = datas[batch_index]

    # 입력 토큰 시퀀스 문장으로 변환
    # ['_보기', '만', '_해도', '_배', '가', '_부르', '다', '.'] -> 보기만 해도 배가 부르다.
    sentence = bert_tokenizer.convert_tokens_to_string(token_sequence)

    # token_sequence 앞에 [CLS] 토큰이 추가되었기 때문에 1 추가
    target_token_index += 1

    target_token = input_tokens[target_token_index]
    # 토큰을 단어로 변경 (_배 -> 배)
    target_word = bert_tokenizer.convert_tokens_to_string([target_token])
    target_vector = bert_output[target_token_index]
    # 단어, 단어가 사용된 batch_index, 단어가 사용된 문장
    # 배_1 (보기만 해도 배가 부르다.)
    word2vec["{}_{} ({})".format(target_word, batch_index+1, sentence)] = target_vector
```

배_1 (보기만 해도 배가 부르다.)

실습: WSD

Test 유사도 계산 및 비교

코싸인 유사도

```
def get_cos_sim(vector_1, vector_2):
  return np.dot(vector_1, vector_2)/(np.linalg.norm(vector_1)*np.linalg.norm(vector_2))
```

```
# "배"에 대응하는 각 벡터 표현들 사이의 유사도 계산
word_similarity = {}
for word_1, vector_1 in word2vec.items():

    # word_1과 나머지 단어들 사이의 유사도를 담을 리스트 생성
    word_similarity[word_1] = []
    for word_2, vector_2 in word2vec.items():

        # 같은 토큰인 경우 건너뜀
        if word_1 == word_2:
            continue

        # word_1과 word_2 사이의 코사인 유사도 계산
        cos_sim = get_cos_sim(vector_1=vector_1, vector_2=vector_2)
        # word_2와 대응하는 유사도를 리스트에 추가
        word_similarity[word_1].append((word_2, cos_sim))

print("\n단어1 (단어1이 사용된 문장) vs 단어2 (단어2가 사용된 문장) -> 유사도\n")
for word in word_similarity.keys():

    # 유사도를 기준으로 정렬, reverse=True를 통해 내림차순으로 정렬
    word_similarity[word] = sorted(word_similarity[word], key=operator.itemgetter(1), reverse=True)

    for index in range(len(word_similarity[word])):
        print("{} vs {} -> {}".format(word, word_similarity[word][index][0], round(word_similarity[word][index][1], 4)))
    print()
```

[(w1,s1),(w2,s2),...]

실습: WSD

Main

```python
if(__name__=="__main__"):
    cache_dir = os.path.join(root_dir, "cache")
    if not os.path.exists(cache_dir):
        os.makedirs(cache_dir)

    config = {
        "data_path": os.path.join(root_dir, "datas.txt"),
        "cache_dir_path": cache_dir,
        "pretrained_model_name_or_path": "monologg/kobert",
        "max_length": 30
    }

    test(config=config)
```

```
단어1 (단어1이 사용된 문장) vs 단어2 (단어2가 사용된 문장) -> 유사도

배_1 (보기만 해도 배가 부르다.) vs 배_2 (점심을 먹지 못해 배가 많이 고팠다.) -> 0.7238
배_1 (보기만 해도 배가 부르다.) vs 배_4 (그 섬에는 하루에 두 번씩 배가 들어온다.) -> 0.6015
배_1 (보기만 해도 배가 부르다.) vs 배_6 (사각사각 씹히는 배의 맛이 달고 시원하다.) -> 0.5815
배_1 (보기만 해도 배가 부르다.) vs 배_5 (나는 과일 중에서 배를 가장 좋아한다.) -> 0.5376
배_1 (보기만 해도 배가 부르다.) vs 배_3 (배 한 척이 바다 한가운데 떠 있다.) -> 0.4839

배_2 (점심을 먹지 못해 배가 많이 고팠다.) vs 배_1 (보기만 해도 배가 부르다.) -> 0.7238
배_2 (점심을 먹지 못해 배가 많이 고팠다.) vs 배_5 (나는 과일 중에서 배를 가장 좋아한다.) -> 0.5074
배_2 (점심을 먹지 못해 배가 많이 고팠다.) vs 배_4 (그 섬에는 하루에 두 번씩 배가 들어온다.) -> 0.5071
배_2 (점심을 먹지 못해 배가 많이 고팠다.) vs 배_6 (사각사각 씹히는 배의 맛이 달고 시원하다.) -> 0.4337
배_2 (점심을 먹지 못해 배가 많이 고팠다.) vs 배_3 (배 한 척이 바다 한가운데 떠 있다.) -> 0.4213

배_3 (배 한 척이 바다 한가운데 떠 있다.) vs 배_4 (그 섬에는 하루에 두 번씩 배가 들어온다.) -> 0.7059
배_3 (배 한 척이 바다 한가운데 떠 있다.) vs 배_6 (사각사각 씹히는 배의 맛이 달고 시원하다.) -> 0.6008
```

질의응답

Q&A

Homepage: http://nlp.konkuk.ac.kr
E-mail: nlpdrkim@konkuk.ac.kr