Faculty of Engineering Science
Year 2019-20

Natural Language Processing

By Prof. Marie-Francine Moens

Course Summaries

Table of Contents

Lecture 2: Parts-of-speech Tagging 1

Lecture 3: Lexical Semantics: 4

Lecture 4: Language Models 7

Lecture 5: Syntactic Parsing 11

Lecture 6: Named Entity Recognition and Semantic Role Labeling 15

Lecture 7: Semantic Parsing 18

Lecture 8: Machine Translation 1.0 23

Lecture 9: Machine Translation 2.0 25

Lecture 10: Discourse Analysis 27

Lecture 2: Parts-of-speech Tagging

Topics for the exam:

- What PoS tagging is:
- Exercises: Tag providing (the page before finding tagging error)
- Evaluation (for open question) to choose the right technique (Figure 10.2 Jurafsky &...)
- Good things/ bad things to choose the right model
- P19 HMM decoding function (1) combined with exercise
- Computing probabilities (exercise)
- Which type of decoding will you choose *Viterbi, Inter P26
- Maximum of entropy model, combine with conditional random field.
- ~P40
- Objective functions, try to understand what we are computing in those objective functions
- P44
- Theory question
- Evaluation: to choose the proper technique

Task: Given a sentence in English, identify the POS of each Word.
E.g.: "The boy put the keys on the table".
The – Determiner; {boy, keys, table} – Noun; put- Verb; on -preposition.

Categories of POS:

Open Class Words:

These are called **open class** because we keep adding new words each day to the class.
E.g.: Nouns, Verbs, Adjectives, Adverb.

Closed Class Words:

The **closed classes** differ from language to language than do the open classes

Some important closed classes in English: prepositions: *on, under, over, near, by, at, from, to, with*; determiners: *a, an, the*; pronouns: *she, who, I*; conjunctions: *and, but, or, as, if, when*; auxiliary verbs: *can, may, should, are*; particles: *up, down, on, off, in, out, at, by*; numerals: *one, two, three, first, second, third*

Common POS Tags: (PENN TreeBank POS Tags)

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	and, but, or	SYM	symbol	+,%, &
CD	cardinal number	one, two	TO	"to"	to
DT	determiner	a, the	UH	interjection	ah, oops
EX	existential 'there'	there	VB	verb base form	eat
FW	foreign word	mea culpa	VBD	verb past tense	ate
IN	preposition/sub-conj	of, in, by	VBG	verb gerund	eating
JJ	adjective	yellow	VBN	verb past participle	eaten
JJR	adj, comparative	bigger	VBP	verb non-3sg pres	eat
JJS	adj, superlative	wildest	VBZ	verb 3sg pres	eats
LS	list item marker	1, 2, One	WDT	wh-determiner	which, that
MD	modal	can, should	WP	wh-pronoun	what, who
NN	noun, sing. or mass	llama	WPS	possessive wh-	whose
NNS	noun, plural	llamas	WRB	wh-adverb	how, where
NNP	proper noun, sing.	IBM	$	dollar sign	$
NNPS	proper noun, plural	Carolinas	#	pound sign	#
PDT	predeterminer	all, both	"	left quote	' or "
POS	possessive ending	'	"	right quote	' or "
PRP	personal pronoun	I, you, he	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	your, one's	)	right parenthesis	],), }, >
RB	adverb	quickly, never	,	comma	,
RBR	adverb, comparative	faster	.	sentence-final punc	. ! ?
RBS	adverb, superlative	fastest	:	mid-sentence punc	: ; ... – -
RP	particle	up, off			

Figure 10.1 Penn Treebank part of speech tags (including punctuation).

Why POS tagging is hard?

Each word doesn't have a single a specific POS tag. The POS tag for a word changes based on the context in which it was used.
E.g.: words often have more than one POS : BACK

- The back door --> back – JJ
- On my back --> back – NN
- Win the voters back back – RB
- I promise to pay back the bill back – VB

In Brown corpus, 40% of **words** are ambiguous and 12% of **word types** are ambiguous.

How bad is the ambiguity problem?

- One tag is usually more likely than others.
- In Brown Corpus, the word "race" is a noun 98% of time and a verb 2% of the time.
- In such cases of ambiguity to decide the right POS tag, probability can be useful.

Relevant knowledge for POS tagging

The Word Itself:

- Some words may only be nouns. Eg: arrow
- Some words are ambiguous. E.g: like, flies.
- Probability may help, if one tag is more likely than others Local Context:
- Two determiners rarely follow each other
- Two base form verbs rarely follow each other
- Determiners are always followed by an adjective or a noun.

POS Tagging: Two Approaches

1) Rule Based approach
   - Assign each word in the input a list of potential POS tags
   - Window down the list to a single list tag using hand written rules
2) Supervised Machine Learning/ Statistical Tagging:
   - Get a training corpus of tagged text, learn the transformation rules from the most frequent tags.

Probabilistic Tagging:

Find the most likely sequence of tags T for a sequence of words W.

a. This deals with two different types of tagging models
b. Generative VS Discriminative models

Generative Models:

- Generate the observations from hidden class i.e., put a probability over the observations given the class: p(X,C) E.g.: HMM, Naïve Bayes

$$P(t_{1:n}|w_{1:n}) = \frac{P(w_{1:n}|t_{1:n})P(t_{1:n})}{P(w_{1:n})}$$

$$\propto P(w_{1:n}|t_{1:n})\, P(t_{1:n})$$

Bayes rule:Assumption probability of a word depends only on its own tag and is independent of neighboring words and tags

$$\propto \prod_{i=1}^{n} P(w_i|t_i) \times P(t_{1:n})$$

$$\propto \prod_{i=1}^{n} P(w_i|t_i) \times \prod_{i=1}^{n} P(t_i|t_{i-1})$$

First order Markov assumption: the probability of a tag is dependent only on the previous tag (to simplify notation: $p(t_0|t_1)= 1$)

Discriminative models:

- Calculate the probability of the class given
- the data: Logistic regression, Maximum entropy, CRF

POS tagging based on Generative Models:

HMM Tagging

Elements of a HMM Model

a. A set of hidden states (tags)
b. Set of output observations (words)
c. Initial State (Beginning of sentence)
d. State transition probabilities (p(t[n] | t[n-1]))
e. Emission probabilities (p(w[i] | t[i]))

POS Tagging Model

- The goal of HMM decoding is to choose the tag sequence T[1:n] that is most probable given the observation sequence of words W[1:n]
- HMM = directed graphical model (Bayesian network)
- Computation of the joint probability distribution of the

$$P(t_{1:n}) = P\,(t_1)\,P(t_2|t_1)\,P(t_3|t_1,t_2)\,P(t_4|t_1,t_2,t_3)\,\ldots\,P(t_n|t_{1:n-1})$$

sequence of n tags t[1:n]: (Markov chain)

- Assumption of conditional independence (limited horizon):

- Assumption of time invariance (stationary): dependency does

$P(t_{1:n}) = P\,(t_1)\,P(t_2|t_1)\,P(t_3|t_2)\,P(t_4|\,t_3)\,\ldots\,P(t_n|t_{n-1})$ (first order Markov chain)

$P(t_{1:n}) = P\,(t_1)\prod_{t=2}^{n} P(t_i|t_{i-1})$

not change over

time: e.g., if NN has a probability of 0.2 to occur after DT: this probability will not

change if we tag the rest of the sentence or new sentences

- The goal of HMM decoding is to choose the tag sequence that is most probable $t_{1:n}$ given the observation sequence of words

$$w_{1:n}\text{: } \operatorname{argmax} P(t_{1:n}|w_{1:n})$$

$$\hat{t}_{1:n} = \underset{t_{1:n}}{\operatorname{argmax}}\, P(t_{1:n}|w_{1:n}) \propto \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \qquad (1)$$

emission probability

transition probability: here bigram probability, but can be extended to trigram probability => **trigram tagger**

Usually trained with Baum-Welch algorithm for learning the HMM parameters

- HMMs for part-of-speech tagging are trained on a fully labelled dataset, that is, a set of sentences with each word annotated with a part-of-speech tag: => cf. visible Markov model during training
- Parameters are learned by maximum likelihood estimates on this training data

Intermezzo: Decoding

- Decoding = finding the best or good sequence based on the probabilities of the word/tag prediction
- In case of predicting the best sequence of words:
  - Searching through all the possible output sequences and finding the one with highest probability: exponential in the length of the output sequence and is intractable (NP-complete)
  - The size of the vocabulary is often tens or hundreds of thousands of words

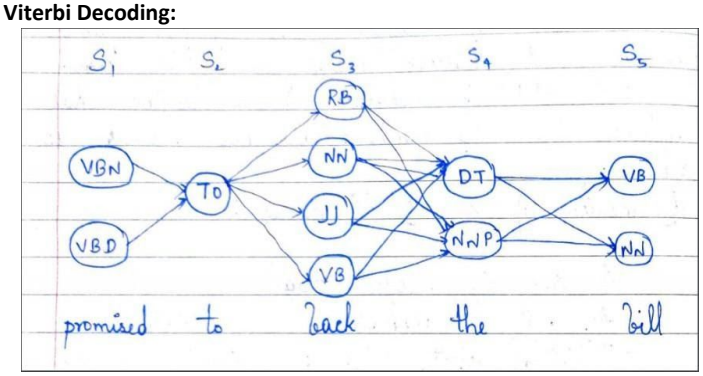
In practice, heuristic search methods are used to return one or more approximate or "good enough" decoded output sequences for a given prediction:

- One popular approximate technique is using greedy prediction = taking the highest scoring item at each stage: often effective, but non-optimal.
- Beam search = considering a beam of k highest scoring items as an approximate search often works better than the greedy approach

Sometimes it is possible to remember partial results rather than recomputing them: dynamic programming or memorization:

- HMM: use of trellis or lattice: Trellis records the probability of all initial subpaths of the
- HMM that end in a certain state at a certain time
- Probability of longer subpaths can then be worked out in terms of one shorter subpath
- Viterbi algorithm for computing the best path

Viterbi Decoding:



Walking through the states to find the best path. E.g.: "promised to back the bill"

Consider the two paths in the above trellis

a. VBD-TO-VB-DT-NN
b. VBD-TO-JJ-DT-NN

Viterbi decoding is all about finding the maximum probability that comes from either VB or JJ to DT.

<u>Intuition:</u>

Optimal path for each state can be recorded. For this we need

- Cheapest cost to state j at steps S.
- Backtrace from that state to best predecessor.

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

For a given state q_j at time t, the value $v_t(j)$ is computed as:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

Learning the parameters:

The parameters can be learned in two scenarios.

1. When a labelled dataset is available with the POS category of individual words in a corpus. In this scenario the parameters can be directly estimated using maximum likelihood estimate from the labelled dataset.
2. When only, the corpus is available, but not the labels with POS category. For this we use Baum-Welch algorithm.

Issues with Markov Model Tagging:

- Unknown Words: We do not have required probabilities for calculating the likelihood of a POS tag in case of an unknown word. A possible solution/work around to this would be using morphological cues to assign a more calculated guess
- Limited Context: Consider the two sentences S1: "is clearly marked" and S2: "he clearly marked". In the sentence S1, marked is verb and defines past participle. In S2, the word marked is again a verb and it defines past tense. Here using HMM alone we cannot predict the tag based on the previous observations.
- The possible solution for this would be to use higher order models or combine various n-gram models to avoid sparseness solution.

Maximum Entropy Classifier

- Maximum entropy classifier (multinomial logistic regression) is a popular classification approach in NLP: It allows to integrate many different features (or feature functions) that act as constraints on the model

Principle of Maximum Entropy:

It states that the probability distribution which best represents the current state of knowledge is the one with the largest entropy.

Feature Functions:

It is common to use feature functions fN that have as arguments the input observation
x and the output class y, and that have binary outcome:

$$f_j(y,x) = \begin{cases} 1 \text{ if } f_j(y,x) \text{ satisfies a certain constraint} \\ 0 \text{ otherwise} \end{cases}$$

$$f_j(y,x) = \begin{cases} 1 \text{ if suffix of } word = \text{"ing" and } y = \text{VBG} \\ 0 \text{ otherwise} \end{cases}$$

While the maximum entropy approach is not in principle limited to binary valued
indicator functions: efficient training procedures such as generalized iterative scaling only works for binary features.
We choose the model p* that preserves as much uncertainty as possible, or
which maximizes the entropy H(p) between all the models p ÎP that satisfy
the constraints enforced by the training examples:

$$p^* = \arg\max_{p \in P} H(p)$$

$$H(p) = -\sum_{(x,y)} P(x,y) \log_2 P(x,y)$$

MaxEnt:

The probability of assigning class y to instance x with the **multinomial logistic regression** classifier given K feature functions is then computed as:

$$P(y|x) = \frac{\exp(\sum_{j=0}^{K} w_j f_j(y,x))}{\sum_{y_c \in C} \exp(\sum_{j=0}^{K} w_j f_j(y_c,x))}$$

extra dummy feature fH with weight wH is added to consider bias term of the
linear function. The multinomial logistic regression classifier is in NLP also known as maximum entropy modelling or MaxEnt
Given a test example x: the probability of class membership is computed for each
class.
When classes are mutually exclusive: the class with highest probability is chosen:

$$\hat{y} = \underset{y_c \in C}{\operatorname{argmax}} P(y_c|x)$$

Training of Logistic regression Classifier

Training is learning the weights $\boldsymbol{w}$ of the feature functions

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)})$$

The objective function that we are maximizing is:

$$L(\boldsymbol{w}) = \sum_i \log P(y^{(i)}|x^{(i)})$$

Finding the weights that

$$L(\boldsymbol{w}) = \sum_i \log P(y^{(i)}|x^{(i)}) = \sum_i \log \frac{\exp(\sum_{j=0}^{K} w_j f_j(y^{(i)},x^{(i)}))}{\sum_{y_c \in C} \exp\left(\sum_{j=0}^{K} w_j f_j(y_c^{(i)},x^{(i)})\right)}$$

$$= \sum_i \log \exp\left(\sum_{j=0}^{K} w_j f_j(y^{(i)},x^{(i)})\right) - \sum_i \log \sum_{y_c \in C} \exp\left(\sum_{j=0}^{K} w_j f_j(y_c^{(i)},x^{(i)})\right)$$

maximizes this objective function: e.g., with gradient ascent method: Initialize the weights (e.g., zero weights) and move in the direction of
the gradient, $L'(\boldsymbol{w})$, the partial derivative of the objective function $L(\boldsymbol{w})$ with respect to the weights
For a given feature fj, this derivative can be shown to be the difference between
the following two counts:

$$L'(w) = \sum_i f_j(y^{(i)},x^{(i)}) - \sum_i \sum_{c=1}^{C} P(y_c^{(i)}|x^{(i)})\, f_j(y_c^{(i)},x^{(i)}) =$$

$$\sum_i \text{Observed count } (f_j) - \text{Expected count } (f_j)$$

Interpretation: The first term is just the count of feature fj in the training data, i.e., the number of times fj = 1.

- The second is the expected count of fj under the probabilities assigned by the current model
- Optimal weights according to this objective: model's expected feature counts match the actual counts in the training data

Maximum Entropy Markov Model:

Local classification of each word/ constituent, but extra features = labels of the previous word/constituent in the sequence y[i-1]:

POS tagging of Unknown Words:

To achieve high accuracy with POS taggers, it is also important to have a good model for dealing with unknown words (new words that appear): Useful features are:

- Word shape
- Morphology: e.g., in English words that end in *-s* are likely to be plural nouns (NNS), words ending with *-ed* tend to be past participles (VBN), words ending with *-able* tend to be adjectives (JJ)

Lecture 3: Lexical Semantics:

Topics for the exam:

- Definitions, dataset do not need to remember by heart, just make a summary
- What is important is what can go wrong is we use Yarowsky algorithm, what is the disadvantage
- Unsupervised models
- Methods to compute words similarity: you should be able to apply that
- Understand what is being computed in the of formulae on Slide 37
- Exercises with core metrics etc.
- Principle of Word vectors: theory question: what are those word vector ; how do you compute the CBOW model?
- Try to understand objective functions of P63. Can you explain the difference between the two models?

Lexicon – the vocabulary of a person, language or a branch of knowledge

Lexical semantics is concerned with the systematic meaning related connections among lexical items and the internal meaning related structures of individual lexical items.

Lexeme – It should be thought as a pairing of a particular orthographic and phonological form with some sort of symbolic representation and meaning.

Homonymy- one word has different unrelated meanings. E.g., "bark")

Polysemy- one word has different related meanings E.g.: "opening"

Metonymy: use of one aspect of a concept or entity to refer to other aspects of the entity or the entity itself E.g., "White House"

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word

  best-sense ← most frequent sense for word
  max-overlap ← 0
  context ← set of words in sentence
  for each sense in senses of word do
    signature ← set of words in the gloss and examples of sense
    overlap ← COMPUTEOVERLAP(signature, context)
    if overlap > max-overlap then
        max-overlap ← overlap
        best-sense ← sense
  end
  return(best-sense)
```

Synonymy: two words have the same propositional meaning, i.e., they are
substitutable with one another without changing the truth conditions of the
sentence (e.g., "vermin" and "pests")

Antonymy: two words have an opposite meaning (e.g., "cold" and "hot",
"rise" and "fall")

Hypernymy: More general representation of a word is Hypernymy and more specific representation of a word is Hyponymy.

E.g.: "fruit" is hypernym of "apple", "apple" is a hyponym of "fruit"

Meronym and Holonym: E.g.: "wheel" is a meronym of "car" and "car" is a holonym of "wheel"

Word Sense Disambiguation (WSD):

Many words have several meanings or senses.

E.g.: The word "Bass" is a type of music and also a type of fish.

Now, the task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word.

Important clues/features:

- One sense per discourse: the sense of a target word is highly consistent within a document
- One sense per collocation: nearby words provide strong and consistent clues to the sense of a target word, conditional on relative distance, order and syntactic relationship

Knowledge based approaches

i. Overlap based approaches

Requires machine readable dictionary (MRD) like a Wordnet. Suppose consider a word W has senses S1 and S2. Now we construct a sense bag 1 & sense bag 2 along with the context bag. The sense bag which gives max. overlap with context bag is considered appropriate sense.

ii. Minimally supervised / semi-supervised WSD – Yarowsky:

- Annotations are expensive for supervised learning
- "Boot strapping" or co-training.
- Start with a small seed, learn the decision list
- Use the decision list to label the rest of the corpus
- Retain confident labels, treat them as annotated data to learn new decision list.
- Repeat.

Algorithm:

1. Initialize C with the corpus: collect all the contexts of the ambiguous word in the corpus (= training examples)
2. For each sense sk of W do:
   Fk = { fm | fm belongs to initial set of collocations}
   manually identifying a small number of seed collocations ("properly guessed") representative for each sense or manually tagging some examples Ek = {ci belongs to C | for every fm : fm applies to ci and fm belongs to Fk}
   tagging all training examples containing the seed collocations with the seed's sense label
3. Repeat until all Ek don't change anymore. For all senses sk of W do:
   3.1. The classified training examples allow learning new collocations and updating the set of collocations

$$F_k = \{ f_m \mid \forall\ s_l \neq s_k : \log \frac{P(s_k|f_m)}{P(s_l|f_m)} > \alpha \}$$

   Each collocation has a score associated that corresponds to the likelihood that the collocation is specific to sense sk

   The strongest collocations for a particular context are chosen by selecting those collocations that are most characteristic of the just updated Ek

   3.2. *Ek* = {*ci* belongs to C | for every *fm* : *fm* applies to *ci* and *fm* belongs to *Fk*}.

   The newly learned collocations allow tagging (classifying) more training examples with the sense label

   3.3. Optionally apply the constraint one sense per discourse for augmenting and filtering the classification
4. The learned dictionary Fk of collocations can be applied to unseen data

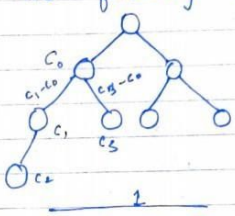
iii. Unsupervised sense disambiguation

Training: clustering of contexts given the number of senses: For each token $w+$ of word w, compute a context vector $\boldsymbol{c}$
Use a clustering algorithm to group the context vectors of $w+$ into a predefined number of clusters: each cluster defines a sense of w Compute the centroid vector $\boldsymbol{sj}$ of each cluster as the average vector of its members:
each $\boldsymbol{sj}$ is a sense vector representing the sense of w

- Prediction / testing : for a token t of w:
- Compute the context vector $\boldsymbol{c}$ for t
- Retrieve all sense vectors $\boldsymbol{s}$ $\boldsymbol{j}$ of w
- Assign t to the sense vector $\boldsymbol{sj}$ that is closest to the context vector $\boldsymbol{c}$ of t

$$Sim_{JC}(c_1, c_2) = \frac{1}{IC(c_1) + IC(c_2) - 2 \times IC(LCS(c_1, c_2))}$$

Consider the following hierarchy



Resnik $Sim(c_1, c_3) = IC(c_0)$

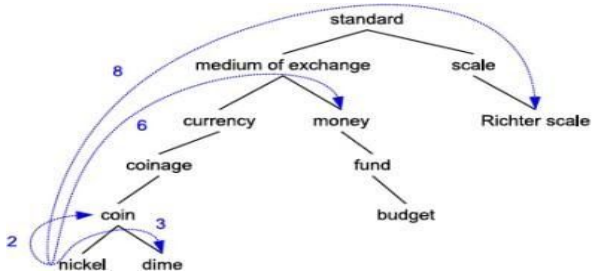
Lin $\frac{2 \cdot IC(c_0)}{IC(c_1) + IC(c_3)}$

$$JC = \frac{1}{c_1 - c_0 + c_3 - c_0}$$

Lin similarity is very popular.

Wordnet is very good if we need to see the similarity b/w word sense, but not for similarity b/w the words.
Eg:
I saw a man who is 98 years old and still can walk and tell fokes
The above sentence has a total of 6758400 Senses.

Word Similarity



Two classes of algorithms:
Distributional algorithms: by comparing words based on their distributional context in the corpora
Thesaurus based algorithms: Based on whether words are "nearby" in Wordnet.
Thesaurus based word similarity:
This method usually uses, the is – a subsumption/ hypernymy hierarchy and sometimes the glosses too.
a gloss = dictionary style definition
a synset = list of (near) synonyms for the sense
E.g.: Car and bicycle are similar in a context.
Car and gasoline are related but not similar.
Two words are said to be similar if they are nearby in the hypernym graph.
Pathlen(c1,c2) = number of edges in the shortest path between c1 and c2.

Path-length based similarity:

$$\text{sim}_{\text{path}}(c_1, c_2) = \frac{1}{\text{pathlen}(c_1, c_2)}$$

where pathlen($c1$, $c2$) = the number of edges in the shortest path in the thesaurus graph between the sense nodes $c1$ and $c2$+1
Assumes uniform path length over edges: incorrect assumption for WordNet
hypernym hierarchy

Leacock – Chodorow(L-C) Similarity:

$$\text{"}dist_{JC}(c, \text{'hypernym}(c)\text{'}) = IC(c) - IC(hypernym(c))$$

$$\begin{aligned} dist_{JC}(c_1, c_2) &= dist_{JC}(c_1, LCS(c_1, c_2)) + dist_{JC}(c_2, LCS(c_1, c_2)) \\ &= IC(c_1) - IC(LCS(c_1, c_2)) + IC(c_2) - IC(LCS(c_1, c_2)) \\ &= IC(c_1) + IC(c_2) - 2\,IC(LCS(c_1, c_2)). \end{aligned}$$

simLC(c1,c2) = -log(pathlen(c1,c2/2d).
d- Maximum depth of the hierarchy.

Information-content word-similarity: still relies on the structure of a thesaurus,
but also adds probabilistic information derived from a corpus IC = -log P(c), where c is the concept.

Resnik Similarity: How similar two words are, depends on how much they have in common. It measures the commonality by the information content of the lowest common subsume (LCS)

$$\mathrm{sim}_{\mathrm{Resnik}}(c_1, c_2) = -\log\ P(\mathrm{LCS}(\,c_1, c_2))$$

where the information content (IC) of a concept c, IC (c) = - log P(c)

$$P(c_1) = \frac{C(c_1)}{\sum_{i=1}^{N} C(c_i)}$$

where C(ci) = counts of the concept's synset in a reference corpus (e.g., Semcor corpus), N = number of synsets in WordNet

Lin Similarity: It's not just about similarities – it's also about differences! (BAM)
Resnik – The more information content they share, the more similar they are.
Lin: The more information content they don't share, the less similar they are.
Not the absolute quantity of shared information but the proportion of shared information.

$$\mathrm{sim}_{\mathrm{Lin}}(c_1, c_2) = \frac{2\ \log P(\mathrm{LCS}(\,c_1, c_2))}{\log P(\,c_1) +\ \log P(\,c_2)}$$

Jiang-Conrath Similarity (JC):
It is based on JC distance. It is calculated using the IC(information content) which assigns the lengths to graph edges:

Word representations in the form of vectors can be computed based on term co-occurrence e.g., within a document, sentence or window of words

- **Sparse word vectors** with dimensions corresponding to the words in the vocabulary and which are sparse (many zero values)
- **Dense word vectors** = word embeddings (e.g., computed with neural networks) with lower dimensions (e.g., 50-1000) and which are dense (most values are nonzero)

- Vectors are trained on large corpus Term

Co-occurrence Matrix

	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

Co-occurrence vectors for 4 words, computed based on the Brown corpus, showing only eight of the (binary) dimensions (handpicked).
Note that the vectors are often extremely sparse.

Co-occurrence is computed based on the occurrence of the words in the same context Context can vary: document, paragraph, sentence, window of words, ...

- In the above term co-occurrence matrices: entrees have a binary value indicating cooccurrence or not, or are the frequency of co-occurrence of two terms
- Often the probability that two terms co-occur is used as alternative: e.g., as computed in pointwise mutual information statistic

PointWise Mutual Information (MI):

The **mutual information** between two random variables *X* and *Y* measures common information in the two variables:

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

Pointwise mutual information (PMI) statistic measures common information between two events here between two words *w*i and *w*j:

$$PMI(w_i, w_j) = I(w_i, w_j) = \log_2 \frac{P(w_i, w_j)}{P(w_i)P(w_j)}$$

$P(w_i)$ and $P(w_j)$ = probabilities of occurrence of respective word

$P(w_i, w_j)$ = their probability of co-occurrence

if $PMI(w_i, w_j)$ >>1: w_i and w_j have a strong correlation

if $PMI(w_i, w_j)$. » 0: w_i and w_j have no correlation

Dense Word Vectors

The goal is to reduce the dimensionality of the vectors so that similar words (hopefully
synonyms) are projected to the same points in the vector space Each word is associated with a real valued vector in *n*-dimensional space (usually n = 50 - 1000)

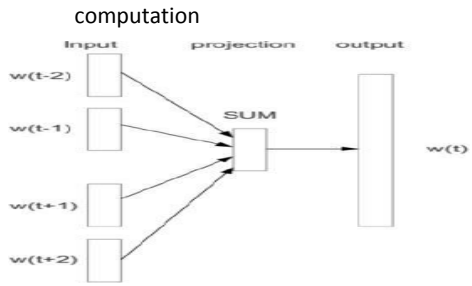
There are different architectures to train the vectors, the most famous ones are the below:

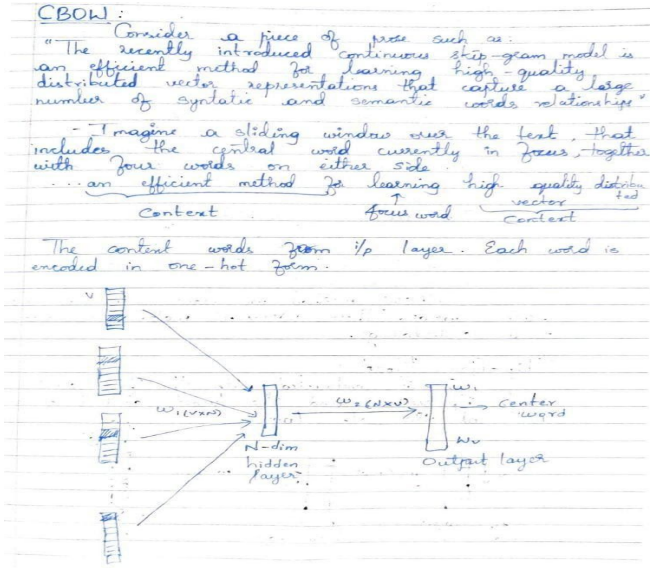
- CBOW
- Skip-gram NN-LM
- GloVe

CBOW:

CBOW model (in word2vec package):

- Input: words in the form of 1-hot word vectors within short window **without their position**: the vectors are summed
- Output: the current word is predicted
- The hidden layer implements a linear function: efficient computation

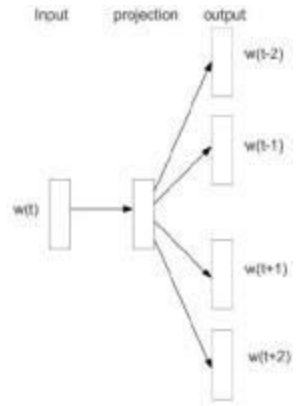


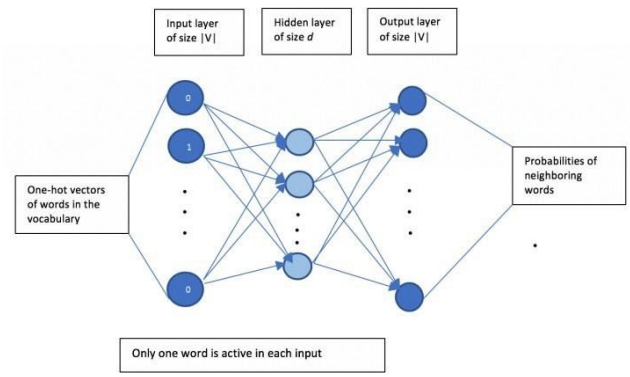


Skip Gram Model:

Skip-gram NNLM (in word2vec package):

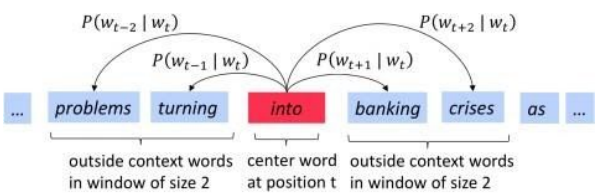
- Input: the current word in the form of a 1-hot vector
- Output: the words within a short window is predicted, the network predicts the surrounding words, **not their position**
- The hidden layer implements a linear function: efficient computation

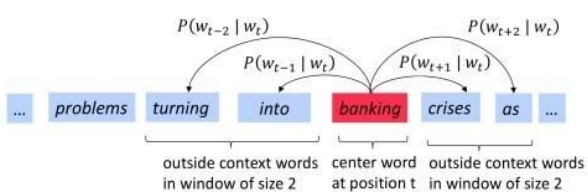




Idea: a sliding window goes over all texts of the large training corpus:

- Input the word at position t
- Output: words in context with as goal training a model that predicts $P(w_{t+j}|w_t)$





Input: one-hot coding: i.e., number of inputs = size of the vocabulary: every neuron represents a word

- Input is fed to the hidden layer without activation function: projection layer
- Hidden layer typically contains around 300 neurons (= dimension d of the word embedding vector): output: o(hidden) = $\boldsymbol{o}$(in) $\boldsymbol{W}$(in)

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Output layer has again the size of the vocabulary: softmax activation function:

output: $\hat{\boldsymbol{y}}_{out} = \boldsymbol{o}_{out} = \text{softmax}(\boldsymbol{o}_{hidden}\boldsymbol{W}_{out})$

$$\boxed{\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{l=1}^{L} e^{x_l}}}$$

<u>After training:</u>

- Rows of form the vector representations of each word of the vocabulary = embedding of the word
- Trained model can also be used as language/prediction mode

Training the skip-gram NNLM

- Training the CBOW or skip-gram NNLM model supposes a training set of correct word-context pairs D and incorrect word-context pairs D~
- The positive examples D are generated from the corpus; the negative examples D~ can be generated in many ways (negative sampling): In the word2vec tool they are generated as follows:
- For each good pair (w, c) sample k words w(i), each forming w(i), c as a negative example to D~ where k is a parameter in the model
- The corpus-wide objective of the algorithm is to maximize the log-likelihood of the data $D \cup D$~ :

$$\mathcal{L}(\theta;\ D,\ \bar{D}) = \sum_{(w,c)\in D} \log P(CorrectPair = 1|w,c) + \sum_{(w_i,c)\in \bar{D}} \log P(CorrectPair = 0|w_i,c)$$

GloVe

GloVe = Global Vectors for Word Representation: the best of two worlds

Construct matrix $\mathbf{A}$ |v| × |v| with word co-occurrence probabilities based on a large, unlabeled corpus and retain only co-occurrence frequencies A(ij) > 0 of word pairs(wi,wj)

Train word vectors $\boldsymbol{w}$(i) such that their pairwise dot product equals the logarithm of the words' probability of co-occurrence

This yields the following weighted least square loss:

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{|V|} f(A_{ij})(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_j + b_i + b_j - \log A_{ij})^2$$

subject to A(ij) > 0, where f = weighting function that assigns lower weights to infrequent and very frequent word pairs and bi and bj are word biases (which are also trained).

Similarity between word vectors

The word vectors (sparse or dense) are compared and the distance or similarity **between two vectors** $\boldsymbol{w}$(i) and $\boldsymbol{w}$(j) with dimension n is computed as relatedness score of the words

<u>Inner Product Similarity:</u>

$$sim(\boldsymbol{w}_i, \boldsymbol{w}_j) = \boldsymbol{w}_i^{\mathrm{T}} \cdot \boldsymbol{w}_j = \sum_{k=1}^{n} w_{ik} w_{jk}$$

<u>Cosine Similarity:</u>

$$sim(\boldsymbol{w}_i, \boldsymbol{w}_j) = \frac{\boldsymbol{w}_i^T \cdot \boldsymbol{w}_j}{\|\boldsymbol{w}_i\|\|\boldsymbol{w}_j\|} = \frac{\sum_{k=1}^{n} w_{ik} w_{jk}}{\sqrt{\sum_{k=1}^{n} w_{ik}^2}\sqrt{\sum_{k=1}^{n} w_{jk}^2}}$$

<u>Evaluation:</u>

Refer to the slides for evaluation results between CBOW and Skip- Gram.

Lecture 4: Language Models

Topics for the exam:

- N-gram model, how to train them
- How to do the estimation
- Smoothing methods; Claused; understand those smoothing methods for the small exercises

- Principles of neural networks based LMs (attention, **RNN: encoding, decoding**)
- Bert
- P63 Elmo how is it different from those models
- Perplexity

Overview:

- Language modeling: what, why and how?
- Count based n-grams language models and smoothing methods
- Neural network based LM
- Feedforward NN LM: e.g., BERT
- Recurrent NN LM: e.g., ELMo
- Evaluation

What is a Language Model?

- Language model is a probabilistic model of word sequences used for word prediction:
- E.g., n-gram models which predict the next word from the previous n - 1 words (e.g., bigram, trigram models)
- Predicting a word given left and right context

Uses:

- LMs are obviously some of the most important NLP tools.
- They are often used as the features for most tasks (and these days in all tasks).
- Eg. Predicting words in context (their primary usage), speech recognition, machine translation, spelling correction, language generation, etc.
- They also find usage in predicting part-of-speech tags and semantic roles in context.

Corpora (Unimportant for exam, so just summed up)

- Old: Brown Corpus, Switchboard Corpus, and British National Corpus
- New: Wik-text datasets and Google corpus.

N-grams and their training (Important)

- Most language models use the chain rule to decompose the joint probability into a sequence of conditional probabilities:

$$P(w_{1:T}) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1,w_2)\, P(w_4|w_1,w_2,w_3) \dots P(w_T|w_{1:T-1})$$

- A first order Markov model or chain: dependence is limited within 2 successive states:
  $P(w_{1:T}) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_2)\, P(w_4|\, w_3) \dots P(w_T|w_{T-1})$
  $P(w_{1:T}) = P(w_1) \prod_{t=2}^{T} P(w_t|w_{t-1})$
- A second Markov model or chain: dependence is limited within 3 successive states:
  $P(w_{1:T}) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1,w_2)\, P(w_4|\, w_2,w_3) \dots P(w_T|w_{T-2},w_{T-1})$
  $P(w_{1:T}) = P(w_1) P(w_2|w_1) \prod_{t=3}^{T} P(w_t|w_{t-2},w_{t-1})$

Unsmoothed N-grams:

- Problem: estimating the probability of a word w given its history $hist$: $P(w|hist)$: e.g.,

$$P(w = \text{meeting}|hist = \text{the ongoing})$$

Maximum likelihood estimation (MLE) from large corpus: computing the relative frequency:

$$\frac{c(\text{the ongoing meeting})}{c(\text{the ongoing})}$$

where c = frequency of the sequence in the corpus

Estimating the N-grams:

- n-gram language models look n -1 words in the past:
  - Bigram model: $P(w_T|w_1, w_2, \dots, w_T) \approx P(w_t|w_{t-1}) = \frac{c(w_{t-1},w_t)}{\sum_w c(w_{t-1},w)}$

$$= \frac{c(w_{t-1},w_t)}{c(w_{t-1})}$$

  - Trigram model: $P(w_t|w_1, w_2, \dots, w_T) \approx P(w_t|w_{t-2},w_{t-1})$

$$= \frac{c(w_{t-2},w_{t-1},w_t)}{c(w_{t-2},w_{t-1})}$$

  - …
  - n-gram model: $P(w_t|w_1, w_2, \dots, w_{t-1}) \approx P(w_t|w_{t-n+1}, \dots, w_{t-1})$

$$= \frac{c(w_{t-n+1}, \dots, w_{t-1}, w_t)}{c(w_{t-n+1}, \dots, w_{t-1})}$$

Points to keep in mind:

- The longer the n-grams, the more coherent, the sentences are when generated. **But** after a point, training them becomes unreasonably expensive.
- Language models are pretty useless if training on different dataset then training: the Wallstreet trigrams do not well predict Shakespeare's language.
- Closed vocabulary: Lexicon of known words
- Open vocabulary:
  - Unknown words are tagged as <UNK>
  - OOV rate: percentage of unknown words

Smoothing (Important):

Why?

- Some n-gram patterns have a zero probability based on the training set: inner product => 0
- Some very small estimates might be rounded to 0
- Rare n-gram patterns in the training set might mislead the estimates.

What does it do?

- Smoothing assigns some-non zero probability to any n-gram, even if it is not seen in the training data.
- Smoothing can be viewed as discounting (lowering) non-zero counts to get the probability mass that will be assigned to zero counts.

Laplace Smoothing:

- **Laplace smoothing**: add small value to all counts before normalizing => useful baseline:
  - e.g., bigram model: $P_{Laplace}(w_t|w_{t-1}) = \frac{C(w_{t-1},w_t)+1}{C(w_{t-1})+|V|}$
    where $|V|$ = number of words in vocabulary

Good-Turing discounting:

Good -Turing discounting: the count of the things you have seen help estimate the count of things you have never seen.

N_c = the number of n-grams that occurs c times = frequency of frequency c

- E.g., bigrams: N_0 = number of bigrams with count 0, N_1 = number of bigrams with count 1, etc.
- Compute revised counts: $c^* = (c+1)\frac{N_{c+1}}{N_c}$
- Estimate probability by maximum likelihood using the revised counts
- Estimate probability for events that have zero count N_0: use counts of n-gram that occurs only once N_1: $\frac{N_1}{N}$

Example:

- Suppose we are fishing in a lake with 8 species (bass, carp, catfish, eel, perch, salmon, trout, whitefish) and we have caught 6 species with the following counts: 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon and 1 eel. What is the probability that the next fish we catch is a new species (catfish or bass), and what is the revised probability for a known species such as trout?

	Unseen (Bass or Catfish)	Trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT P^*_{GT}	$P^*_{GT}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$P^*_{GT}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

- N_c can still be zero, so N_c is usually smoothed a priori
- In practice:

$$c^* = c \text{ for } c > k \ (k = \text{threshold})$$

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \text{ for } 1 \le c \le k$$

- Good-Turing discounting used in combination with backoff and interpolation algorithms

Interpolation:

Interpolation with lower order n-grams.

- E.g., **simple linear interpolation**: e.g., when estimating the trigram probability:

$$P(w_t \mid w_{t-2}, w_{t-1}) = \lambda_1 P(w_t \mid w_{t-2}, w_{t-1}) + \lambda_2 P(w_t \mid w_{t-1}) + \lambda_3 P(w_t)$$

with $\sum_i \lambda_i = 1$

- **Estimation of the λ_i :** e.g.,
  - Set empirically based on good results in validation set (held-out corpus)
  - Learned automatically from a held-out corpus: e.g., with an Expectation-Maximization (EM) algorithm: maximizes the likelihood on the held-out corpus

Backoff = only using interpolation with a lower order n-gram if we have zero evidence for a higher order n-gram.

$$P_{Katz}(w_t \mid w_{t-n+1}, \dots, w_{t-1}) =$$

BER $\begin{cases} P_{Good-Turing}(w_t \mid w_{t-n+1}, \dots, w_{t-1}), \text{ if } c(w_{t-n+1}, \dots, w_t) > 0 \\ \alpha P_{Katz}(w_t \mid w_{t-n+2}, \dots, w_{t-1}), \text{otherwise.} \end{cases}$

α ensures that the probability mass from all the lower order *n*-grams sums up to exactly the amount that we saved by discounting the higher-order *n*-grams

Practical: use of log probabilities and summation to avoid numerical underflow:

Class-based N-grams:

Class-based n-grams or cluster n-grams (Brown clustering):

- Uses information about a word's class or cluster cl_i :

$$P(w_i \mid w_{i-1}) \approx P(cl_i \mid cl_{i-1}) P(w_i \mid cl_i)$$

$$P(cl_i \mid cl_{i-1}) = \frac{c(cl_{i-1}, cl_i)}{\sum_k c(cl_{i-1}, cl_k)}$$

$$P(w_i \mid cl_i) = \frac{c(w_i)}{c(cl_i)}$$

- Assumes availability of word clusters: for example, bigram "to Shanghai" never occurs in the training set, but bigrams "to London", "to Beijing", etc. do: predict the likelihood of a city following "to"
- Often mixed with regular word-based *n*-grams

Advantages and Disadvantages of N-grams:

Advantages:

- Very scalable: can be easily trained on trillions of words
- Advanced smoothing techniques can simulate empirical distributions of language
- Fast constant time evaluation of probabilities at test time

Disadvantages:

- n-grams of larger size are very sparse, so cannot be used to capture long-term dependencies
- Capture symbolic nature of language, but are not good at capturing semantic correlations between words, neither morphological regularities.

NNLM (NN based Language Models)

- It's bigram based.
- Previous word is used to predict the current word by going through hidden layer (classifier with as many outputs as there are words in the vocabulary)
- This model formed the basis to learn word embeddings and to learn current NN based language models.
- The model can be learned with trigrams, or any n-grams.

$$\boldsymbol{h} = g(\boldsymbol{W}[\boldsymbol{w}_{t-2}; \boldsymbol{w}_{t-1}] + \boldsymbol{b})$$
$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{V}\boldsymbol{h} + \boldsymbol{c})$$

- Inputs: $\boldsymbol{w}_i$ is one-hot vector (; = concatenation)
- Output:
  - Ground truth $\boldsymbol{y}$ is one-hot vector of the predicted word
  - $\hat{\boldsymbol{y}}$ is a probability distribution over all words of the vocabulary $|V|$
- g = e.g., non-linear activation function
- Loss: usual cross-entropy between the predicted probability distribution $\hat{\boldsymbol{y}}$ and the one regarding the true next word $\boldsymbol{y} = \boldsymbol{w}_t$:

$$J(\theta) = -\sum_{j=1}^{|V|} y_j \log \hat{y}_j$$

Advantages:

- Better generalization on unseen n-grams, but with regard to seen n-grams difficult to beat frequency based n-gram models.

Disadvantages:

- The number of parameters in the model scales with the n- gram size and thus the length of the history captured.
- Only a fixed number of previous words can be taken into account to predict the next word (not a problem with Recurrent or Attention-based LMs)

We arrive at one of the most important LMs of the current era.

- BERT uses a fully connected feedforward transformer architecture with multihead self-attention (covered in Ch 9).

Input:

- Summation of token, segment and positional embeddings, representing both a single and a pair of sentences.
- Tokens are represented with word piece embeddings: obtained by breaking words into pieces given a word unit dictionary (automatically learned) and adding word continuation information
- Total input length of word tokens is restricted to 512

Word tokenizer = `He is an unaffable man`
Word piece tokenizer = `He is an un ##aff ##able man`

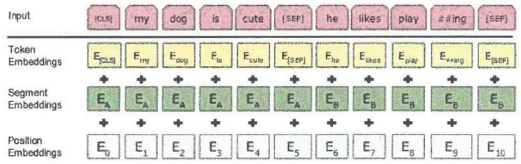
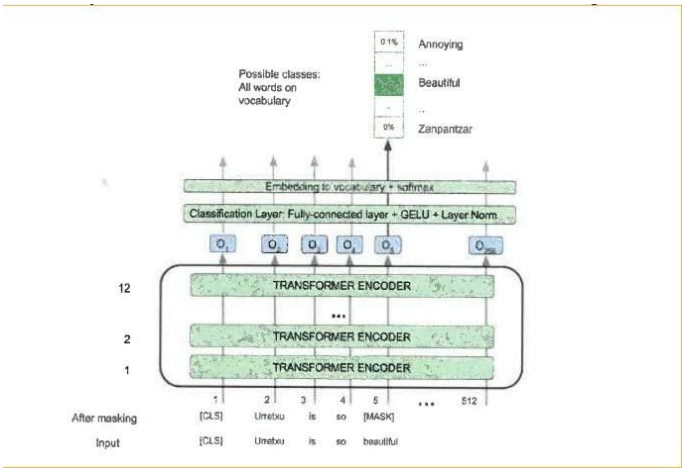


FIGURE 3.3: Input representation of the BERT model. The input embeddings are the sum of the token, segmentation and positional embeddings. Source: (Devlin et al., 2018).

BERT: training based on two unsupervised prediction tasks

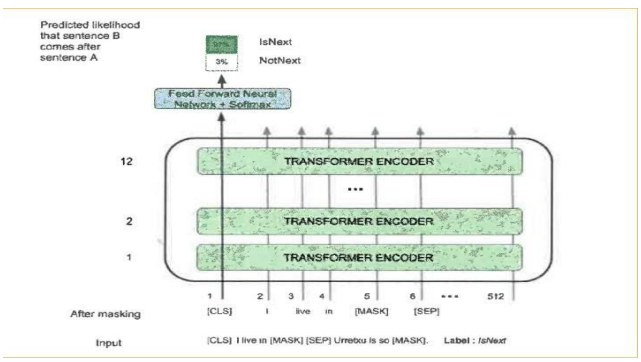
- **Masked language model**: 15% of random input WordPiece tokens in each sequence are masked and the system has to predict the masked token (during training a small amount of them predicts a noise token to avoid overfitting of the model)



- **Next sentence prediction:** Choosing two sentences A and B from the training corpus:
  - 50% of the cases: B is the actual next sentence of A
  - 50% of the cases: B is just a random sentence from the corpus

Input = `[CLS] I live in [MASK] [SEP] Urretxu is so [MASK] [SEP]`
Label = `IsNext`
Input = `[CLS] [MASK] live in Zumarraga [SEP] I [MASK] trains [SEP]`
Label = `NotNext`

The Model:



RNN-based LMs

RNNs are one of the most important NN advances of our era. They learn representations of text across positions in a sequence.

Variant: Long short-term memory networks (LSTMs):

- Store an internal cell state even if that cell does not activate, allowing for syntactic and semantic relations that span across words or sentences to be learned.

RNNs are very powerful, because they have:

- Distributed hidden states that allows them to store a lot of information about the past efficiently
- Non-linear dynamics that allows them to update their hidden states in complicated ways.

Perfectly understanding the image below is heavily recommended:

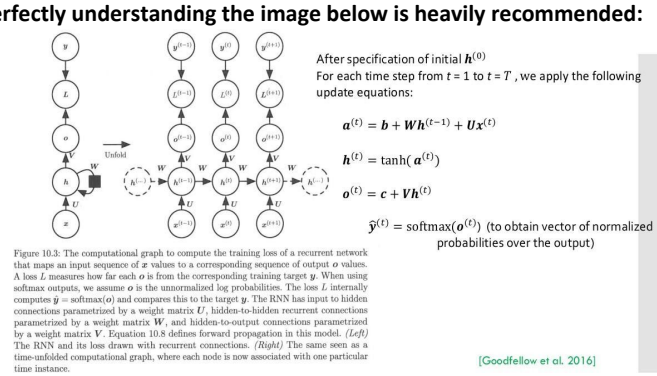


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)* The RNN and its loss drawn with recurrent connections. *(Right)* The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

After specification of initial $\boldsymbol{h}^{(0)}$
For each time step from $t = 1$ to $t = T$, we apply the following update equations:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$

$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)})$ (to obtain vector of normalized probabilities over the output)

[Goodfellow et al. 2016]

RNN training:

Tricky because of the temporal component:
In an RNN trained on long sequences (e.g., 100 time steps) the gradients can easily vanish or explode:

- We can avoid this by initializing the weights very carefully, but long-term dependencies still difficult to model.
- Long short-term memory (LSTM) network offers a solution to this problem of vanishing or exploding gradients with gated recurrent units.

How to train:

- Get a big corpus of text which is a sequence of words $\boldsymbol{w}^{(1)}, \dots, \boldsymbol{w}^{(T)}$; the one-hot vectors of the words are used as input and ground-truth output
- Feed into the RNN-LM; compute the output distribution $\hat{\boldsymbol{y}}^{(t)}$ for every step t = predict the probability distribution of every word, given words so far
- Loss function that operates on step t is usual the the cross-entropy between the predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)} = \boldsymbol{w}^{(t+1)}$:

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

- The overall loss for the entire training set is the average of the losses at each step t:

$$J(\theta) = \frac{1}{T}\sum_{j=1}^{T} J^{(t)}(\theta)$$

ELMo (important):

- ELMo uses a **bidirectional** LSTM architecture (often several layers)
- Input = sequence of words
- Output = next word given the previous history of words
- Objective = minimize the negative log-likelihood of forward and backward pass:

$$J(\theta) = -\sum_{t=1}^{T}(\log P(w_t|w_1, \dots, w_{t-1}) + \log P(w_t|w_{t+1}, \dots, w_T))$$

- ELMo outputs a representation that is a linear combination of all the L inner layer hidden states including the token layer $\boldsymbol{h}_{t,0}$:

$$\hat{\boldsymbol{y}} = \gamma \sum_{j=0}^{L} \lambda \, \text{softmax}(\boldsymbol{h}_{t,j})$$

where γ and λ are task specific weight parameters trained on the task

- ELMo is trained for a specific task and learns tasks specific word representations

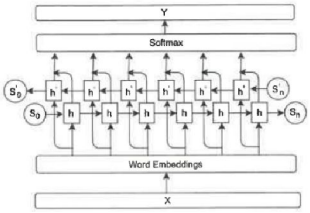


FIGURE 2.7: The architecture of a single layer bidirectional LSTM. The model receives a sequence of words as input and outputs the probability of the next word in the both forward and backward directions.(h: the hidden states of the forward pass; h': the hidden states of the backward pass; x: the input sequence; y: the output probabilities.)

Perplexity (important):

Possibly, the most well-known evaluation metric for language models.

- **Perplexity** is the inverse of the probability of the sequence of test corpus, normalized by the number of words T:

$$PP = P(w_{1:T})^{-\frac{1}{T}}$$

$\frac{1}{T}$ = normalization by number of words

$$= \sqrt[T]{\frac{1}{P(w_{1:T})}}$$

chain rule and first order Markov

$$= \sqrt[T]{\frac{1}{\prod_{t=1}^{T} P(w_t|w_{t-1})}}$$

$$= \sqrt[T]{\prod_{t=1}^{T} \frac{1}{P(w_t|w_{t-1})}}$$

So, because it is the inverse of prob, minimising it means maximising the probability of a sentence. Therefore, lowering it is a target.

- As perplexity is the inverse of the probability of the dataset, we can write perplexity in terms of the cross-entropy loss:

$$PP = \prod_{t=1}^{T} \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

This makes it an excellent loss function.

Not covering the results and future research, because there is nothing to summarize there. Refer to the slides for them.

Lecture 5: Syntactic Parsing:

Topics for the exam:

- Dataset: no need to know
- CKY algorithm, how to compute the most probabilic part
- Advantages/ disadvantages (for the open question)
- CCG for information, not for question
- Lexicalization (principles, not go into depth)
- Recursive neural model (how to build the sentence parser, what is the principle behind it?)
- GBDP graph based dependency parsing: P56 how to compute the best dependency parse by maximum spanning tree P58
- The difference between first order model and second order model?
- P66 transation based parsing, understand
- Neural dependency parsing: principles (for open question to apply it)
- Equations in P77 and P78

Overview:

- What is syntactic parsing?
- Constituent parsing:
- Probabilistic version of the CKY algorithm
- Recursive neural networks
- Dependency parsing:
- Graph based dependency parsing
- Transition based dependency parsing
- Neural dependency parsers

What are we dealing with?

Syntactic Parsing is the task of recognising the syntactic structure of a sentence. Probabilistic parsers recognise the most likely parse of a sentence.

Problems:

Structural Ambiguity: Two types.

Attachment Ambiguity

- Classic PP (prepositional phrase attachment) example:
  - I saw the man with the binoculars.
- Adverbial phrase attachment:
  - We saw the Eiffel Tower flying to Paris

Coordination Ambiguity:

E.g.,

- Old men and women attend the school meeting.

So what does this mean for the parsers?

Often demands for softer approaches: e.g.,

- Parsing based on a probabilistic CFG
- Learning of a model that outputs a probability distribution over the different parses (and choose the parse with the highest probability)

Constituent parsing:

The Cocke – Kasami - Younger (CKY) algorithm = chart based algorithm based on a (P)CFG (It is best to practice the exercises for it.)

- To find possible parses
- To find the highest scoring parse: probabilistic version
- It's a dynamic-programming based approach to parsing
- It needs conversion of the grammar to Chomsky normal form

Chomsky Normal Form:

- Grammars in CNF are restricted to rules of the form:

  $A \longrightarrow B\ C$

  $A \longrightarrow w$

The Process:

- Copy all conforming rules to the new grammar unchanged.
- Convert terminals within rules to dummy non-terminals
- Convert unit-productions (= rules with single non-terminal at the right side)
- Make all rules binary and add them to new grammar

The L1 grammar in the CNF:

$\mathscr{L}_1$ Grammar	$\mathscr{L}_1$ in CNF
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

Figure 12.3 $\mathscr{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here, all the original lexical entries from $\mathscr{L}_1$ carry over unchanged as well.

The Algorithm:

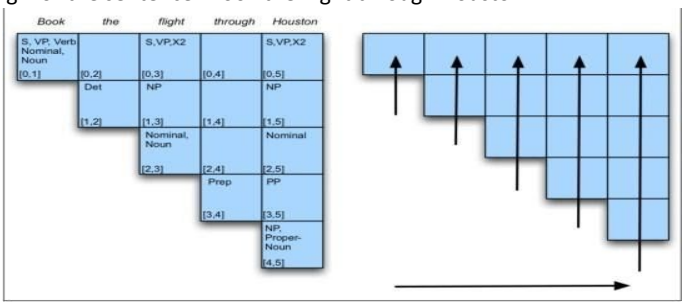
```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      for all {A | A → words[j] ∈ grammar}
            table[j − 1, j] ← table[j − 1, j] ∪ A
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              for all {A | A → BC ∈ grammar and B ∈ table[i, k] and C ∈ table[k, j]}
                    table[i,j] ← table[i,j] ∪ A
```

What does it require?

- A two-dimensional matrix to encode the structure of an entire tree: for a sentence of length n, we will work with the upper-triangular portion of an (n + 1) × (n + 1) matrix M
- The superdiagonal of M contains the parts-of-speech for each input word in the input
- The subsequent diagonals above that superdiagonal contain constituents that cover all the spans of increasing length in the input

Eg: For the sentence "Book the flight through Houston".



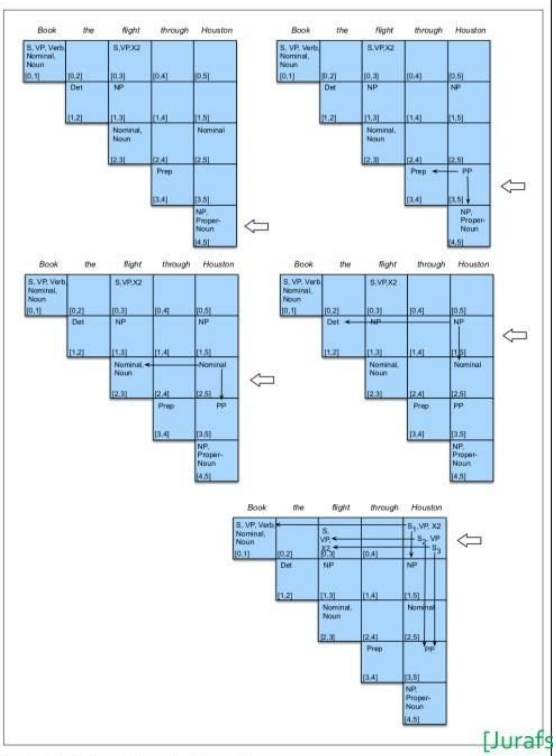


Figure 12.7 Filling the cells of column 5 after reading the word *Houston*.

[Jurafs

Probabilistic CFG:

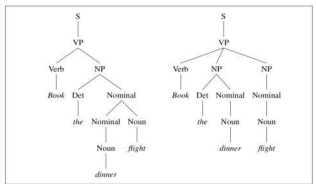
Probabilistic context-free grammar (PCFG): a probabilistic augmentation of context-free grammars in which each rule is associated with a probability: e.g., toy example:

Grammar		Lexicon
$S \rightarrow NP\ VP$	.80	$Det \rightarrow that$ [.10] | a [.30] | the [.60]
$S \rightarrow Aux\ NP\ VP$	.15	$Noun \rightarrow book$ [.10] | $flight$ [.30]
$S \rightarrow VP$	.05	| $meat$ [.015] | $money$ [.05]
$NP \rightarrow Pronoun$	.35	| $flight$ [.40] | $dinner$ [.10]
$NP \rightarrow Proper\text{-}Noun$	.30	$Verb \rightarrow book$ [.30] | $include$ [.30]
$NP \rightarrow Det\ Nominal$	.20	| $prefer$ [.40]
$NP \rightarrow Nominal$	.15	$Pronoun \rightarrow I$ [.40] | she [.05]
$Nominal \rightarrow Noun$	.75	| me [.15] | you [.40]
$Nominal \rightarrow Nominal\ Noun$	.20	$Proper\text{-}Noun \rightarrow Houston$ [.60]
$Nominal \rightarrow Nominal\ PP$	.05	| NWA [.40]
$VP \rightarrow Verb$	.35	$Aux \rightarrow does$ [.60] | can [40]
$VP \rightarrow Verb\ NP$	.20	$Preposition \rightarrow from$ [.30] | to [.30]
$VP \rightarrow Verb\ NP\ PP$	.10	| on [.20] | $near$ [.15]
$VP \rightarrow Verb\ PP$	.15	| $through$ [.05]
$VP \rightarrow Verb\ NP\ NP$	.05	
$VP \rightarrow VP\ PP$	.15	
$PP \rightarrow Preposition\ NP$	1.0	

$$\sum_{LHS} P(RHS|LHS) = 1$$

Example:

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$
$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$



Rules		P	Rules		P
S	→ VP	.05	S	→ VP	.05
VP	→ Verb NP	.20	VP	→ Verb NP NP	.10
NP	→ Det Nominal	.20	NP	→ Det Nominal	.20
Nominal	→ Nominal Noun	.20	NP	→ Nominal	.15
Nominal	→ Noun	.75	Nominal	→ Noun	.75
			Nominal	→ Noun	.75
Verb	→ book	.30	Verb	→ book	.30
Det	→ the	.60	Det	→ the	.60
Noun	→ dinner	.10	Noun	→ dinner	.10
Noun	→ flight	.40	Noun	→ flight	.40

Figure 13.2 Two parse trees for an ambiguous sentence. The transitive parse on the left corresponds to the sensible meaning "Book a flight that serves dinner", while the ditransitive parse on the right corresponds to the nonsensical meaning "Book a flight on behalf of 'the dinner' ".

Disambiguation: selected tree $\hat{T}$ for sentence S:
$\hat{T}(S) = \underset{T}{\operatorname{argmax}}\, P(T)$

[Jurafsky & Martin 2008]

The Algorithm:

function PROBABILISTIC-CKY(*words,grammar*) **returns** most probable parse and its probability

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  for all $\{A \mid A \rightarrow words[j] \in grammar\}$
    $table[j-1, j, A] \leftarrow P(A \rightarrow words[j])$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{A \mid A \rightarrow BC \in grammar,$
        and $table[i,k,B] > 0$ **and** $table[k,j,C] > 0\}$
        if $(table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C])$ **then**
          $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
          $back[i,j,A] \leftarrow \{k, B, C\}$
  return BUILD_TREE($back[1, \text{LENGTH}(words), S]$), $table[1, \text{LENGTH}(words), S]$

Figure 13.3 The probabilistic CKY algorithm for finding the maximum probability parse of a string of *num_words* words given a PCFG grammar with *num_rules* rules in Chomsky normal form. *back* is an array of backpointers used to recover the best parse. The *build_tree*

MLE from a treebank is the preferred method for learning probabilities

Problems and potential solutions:

MLE of the probabilities estimated on a treebank performs poorly.
The naive grammar is a poor one because its CFG assumptions are too strong in some places and too weak at other places.
CFG rules impose an independence assumption on probabilities, resulting in poor modeling of structural dependencies across the parse tree
CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, PP attachment, and coordinate structure ambiguities.

Solutions:

- Enriching the naive grammar: Lexicalization of rules is possible, but leads to increasing complexity and manual effort to craft the rules: e.g., Combinatory Categorial Grammar (CCG)
- Tag splitting = node splitting = subcategorizing the POS categories: at the extreme there is a tag for each single word or word meaning.
- Generalizing the naive grammar: E.g., by smoothing

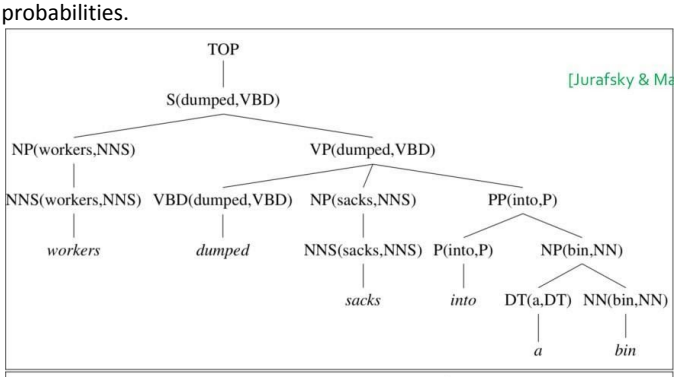
Combinatory Categorial Grammar (CCG) (skipped)

Lexicalization and tag splitting

- These grammars include lexicalised rules.
- Lexicalized grammar: Each non-terminal in the tree is annotated with its lexical head: e.g., *VP → VBD NP PP* would be extended as:
- *VP(dumped) → VBD(dumped) NP(sacks) PP(into)*
- Associate the head tag, i.e., the part-of-speech tag of the headword, with the non-terminal symbols as well

VP(dumped,VBD) → VBD(dumped,VBD) NP(sacks,NNS) PP(into,P)

Maximum likelihood estimation of the lexicalized rules is not very accurate due to sparseness of the data.
We use Independence assumptions to break down each rule so that we would estimate the probability as the product of the individual probabilities.



Internal Rules				Lexical Rules		
TOP	→	S(dumped,VBD)		NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)	VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)		NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS) PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)	DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)	NN(bin,NN)	→	bin

Figure 13.10 A lexicalized tree, including head tags, for a WSJ sentence, adapted from Collins (1999). Below we show the PCFG rules that would be needed for this parse tree, internal rules on the left, and lexical rules on the right.

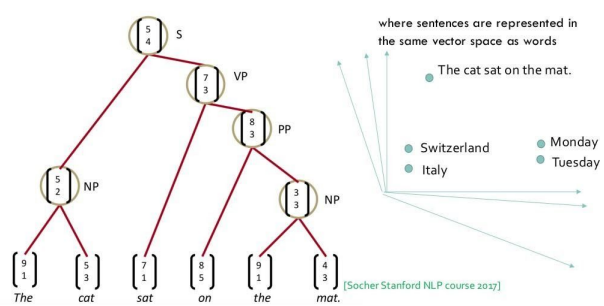
Tag Splitting (Skipped, because unimportant for exam)

Recursive Neural Networks:

- **Compositionality:** Language understanding requires to understand bigger things from knowing about smaller parts
- The meaning of a sentence is determined:
  - by the meaning of its words
  - the rules that combine them
- Languages are recursive. The

Idea:

- Jointly learning parse trees and compositional vector representations:

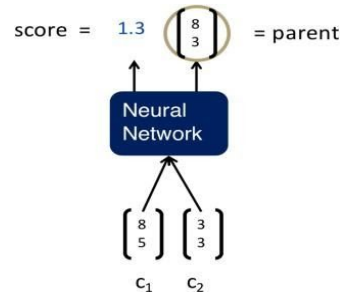


The RNN:

Input: Two candidate children's representations each in the form of a vector.

Output:

- The parent vector **p** representation of their merge:
  - $\boldsymbol{p} = \tanh\left(\boldsymbol{W}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \boldsymbol{b}\right)$
- Score of how plausible the new node is:
  - score $(\boldsymbol{p}) = \boldsymbol{v}^{\mathrm{T}}\mathbf{p}$



The Principle: Standard recursive neural network: same **W** parameters at all nodes of the tree are used.

The objective function (Max-margin):

$$\text{score}_{\text{tree}}(x, y) = \sum_{j \in nodes(y)} score_j$$

$$J(\theta) = \sum_i score\,(x_i, y_i) - \max_{y \in A(x_i)} (score(x_i, y) + \Delta(y, y_i))$$

where y_i = correct parse in the training data

$A(x_i)$ = set of candidate parses of x_i

$\Delta(y, y_i)$ = penalty loss, $= 0$ when $y = y_i$, > 0 when $y \neq y_i$, thus penalizes incorrect parses

The number of parses can be exponential in the sentence length.
So in practice, beam or greedy search is used.

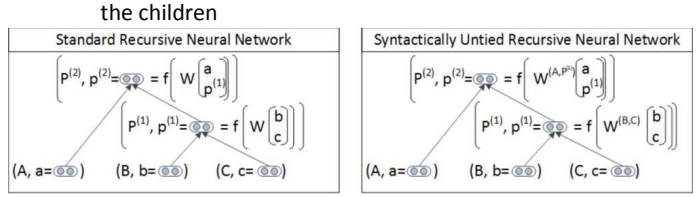
Recursive neural network for structure prediction:

For a standard ReNN:

- The composition function is the same for all syntactic categories, punctuation, etc.
- Decent parse results with single weight matrix, but not adequate for more complex higher order composition and parsing of long sentences

For a syntactically untied ReNN:

- Use of CFG to choose/learn the weight matrix
- Weight matrix depends on the syntactic categories of the children

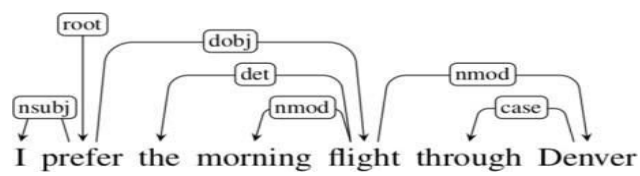


Dependency Parsing:

The syntactic structure or typed dependency structure of a sentence is described:

- Solely in terms of the words (or lemmas) in a sentence and an associated set of directed binary grammatical relations that hold among the words.
- Phrasal constituents and phrase-structure rules do not play a direct role.

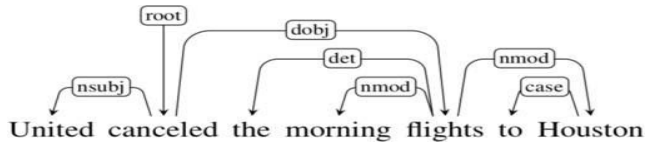
Example:



It uses an explicit root node i.e. the Head of the entire structure, and binary relations that comprise a head and a dependent.

Beyond this, it allows for recognition of types of grammatical relations, or grammatical functions like subject, direct

object and indirect object.



An **arc** from a head to a dependent is said to be **projective** if there is a path from the head to every word that lies between the head and the dependent in the sentence.

A dependency tree is projective if it can be drawn with no crossing edges.
Advantages:

- The ability to deal with languages
- that are: morphologically rich and have a relatively free word order: e.g., when detecting the grammatical function.

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

Graph based dependency parsing:

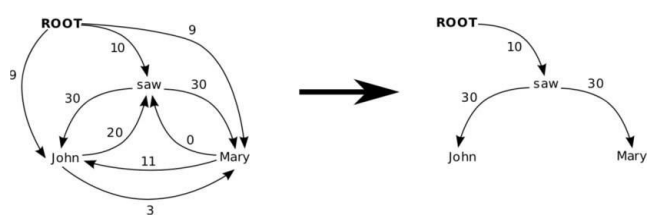
It makes intuitive sense to represent dependent structures as directed graphs.

- Graph G = (V, A) consisting of a set of vertices (nodes) V, and a set of ordered pairs of vertices (arcs) A, where V captures the words of the sentence (possibly including punctuation, stems, (affixes), and A captures the head-dependent and grammatical function relationships between the elements in V

The following constraints are applicable here:

- There is a single designated root node that has no incoming arc/.
- With the exception of the root node, each vertex has exactly one incoming arc
- There is a unique path from the root node to each vertex in V.

The Idea:



More specifically, for an input sentence x define a graph G a = (V_x, A_x) where

$V_x = \{0, 1, \ldots, n\}$

$A_x = \{(i, j, k) \mid i, j \in V \text{ and } j \neq i \text{ and } l_k \in L\}$

e.g., (i, j, k) represents a dependency from head word w_i to modifier word w_j with label l_k

The goal is to score candidate dependencies trees based on the scores of subgraphs where each subgraph doesn't need to be a tree.

$$s(T) = \sum_{c=1}^{m} s(\, G_c)$$

The Scoring function for each subgraph is learned. The learner can be anything, but we assume here that the scoring function is a linear classifier.

$$s(T) = \sum_{c=1}^{m} s(\, G_c) = \sum_{c=1}^{m} \boldsymbol{w}\, \boldsymbol{f}(\, G_c)$$

where $\boldsymbol{w} \in \mathbb{R}^K$ is the learned weight vector

$\boldsymbol{f} \in \mathbb{R}^K$ is the representation vector of subgraph G_c

The classifier labels an edge, or all edges of the subgraph, and outputs a corresponding score.

For training, we learn the weights ***w.***
For Inference or prediction: search for the maximum spanning tree (MST) T* of G_x given the constraints of being a dependency tree: For parsing:

- If it's a projective graph, use the Eisner algorithm (a version of CKY).
- If it's a non-projective graph, use the Chu-Liu Edmonds algorithm.

Chu-Liu Edmonds algorithm:

```
function MAXSPANNINGTREE(G=(V,E), root, score) returns spanning tree

  F ← []
  T' ← []
  score' ← []
  for each v ∈ V do
    bestInEdge ← argmax_{e=(u,v)∈E} score[e]
    F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
      score'[e] ← score[e] − score[bestInEdge]

    if T=(V,F) is a spanning tree then return it
    else
      C ← a cycle in F
      G' ← CONTRACT(G, C)
      T' ← MAXSPANNINGTREE(G', root, score')
      T ← EXPAND(T', C)
      return T

function CONTRACT(G, C) returns contracted graph

function EXPAND(T, C) returns expanded graph
```

Figure 14.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

The idea of the algorithm is to have each vertex in the graph greedily select the incoming edge with highest weight.

First Order Model:

Scored subgraph G_c is a single arc i, j, k between head i and modifier j: As a result features are restricted to the local scope (properties of the arc, its nodes, and local context)

We we want a global decision, i.e. the best tree in a scalable way, e.g., by using constraints modeled as integer linear program.

Transition-based Dependency Parsing:

Transition operators for transition based parsing:

- LEFTARC: Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack

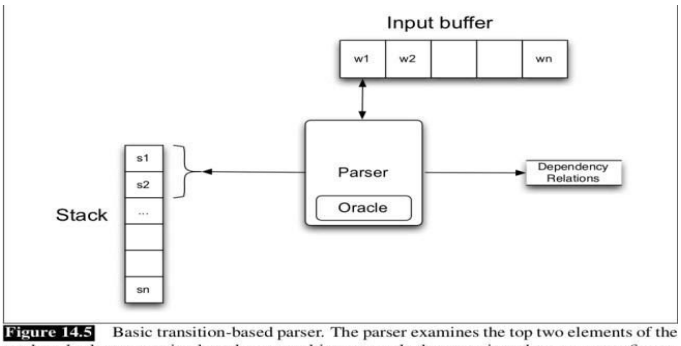


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

The Algorithm:

- Benefits: Linear complexity: O(n): oracle gives one decision (no backtracking) and a single parse is returned

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
   tr  ← ORACLE(state)          ; choose a transition operator to apply
     state ← APPLY(tr,state)    ; apply it, creating a new state
  return state
```

Example:

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

The Oracle:

The oracle has been trained on annotated texts (with transition operations): the learning algorithm can be anything, but we assume here that the oracle o is a linear classifier that predicts the next transition operation tr given the context stack c.

To produce labeled trees, we can parameterize the LEFTARC and RIGHTARC operators with dependency labels: e.g., LEFTARC(NSUBJ) or RIGHTARC(DOBJ)

Problems and Potential Solutions:

Standard transition based parsing makes a greedy pass over the sequence error prone (due to greedy inference and local learning).

How to improve:

- Structured prediction algorithms (e.g., conditional random field).
- During training: minimize loss over entire sequence.
- Use beam search to find highest-scoring sequences

Neural Dependency Parsing:

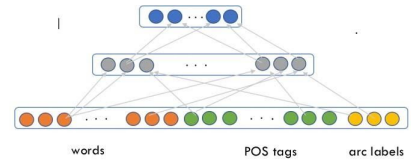
- Illustrated with transition based dependency parser that predicts transition operators.
- The neural network learns compact dense vector representations of words, POS tags, dependency labels.
- sult is a fast compact classifier, which only uses 200 learned dense features

Softmax layer:
$\boldsymbol{p} = softmax(\boldsymbol{V}\boldsymbol{h})$

Hidden layer:
$\boldsymbol{h} = (\boldsymbol{W}^w\boldsymbol{x}^w + \boldsymbol{W}^t\boldsymbol{x}^t + \boldsymbol{W}^l\boldsymbol{x}^l)^3$

Input layer:
$[\boldsymbol{x}^w; \boldsymbol{x}^t; \boldsymbol{x}^l]$

hidden layer uses a cube activation function



[Chen & Manning 2014]

Objective function: cross-entropy loss plus a L^2 regularization term:

$$J(\theta) = -\sum_i \log P(tr_i) + \frac{\lambda}{2}\|\theta\|^2$$

Evaluation Metrics:

- Stringent metric: a parse is correct when it is exactly the same tree as in the treebank:
  - Accuracy $= \frac{\#\ of\ correctly\ parsed\ sentences}{\#\ of\ sentences}$
  - Using this metric, the best existing parser today has accuracy of ~40%
- Instead we usually use:
  - Labeled recall $= \frac{\#\ of\ correct\ constituents\ in\ system\ parse\ of\ sentence}{\#of\ constituents\ in\ reference\ parse\ of\ sentence}$
  - Labeled precision $= \frac{\#\ of\ correct\ constituents\ in\ system\ parse\ of\ sentence}{\#of\ constituents\ in\ system\ parse\ of\ sentence}$
  - Averaged over all sentences and computation of $F_\beta = \frac{(\beta^2+1)PR}{\beta^2P+R}$
  - Cross-brackets (error rate): the number of constituents for which the reference parse has a bracketing such as ((A B) C) but the hypothesis parse has a bracketing such as (A (B C))

Unlabeled attachment score (UAS): recognition of head
Labeled attachment score (LAS): recognition of head and label

This lecture has the longest summary in the course.

Lecture 6: Named Entity Recognition and Semantic Role Labeling
Topics for the exam:

- Sequence model (conditional random field, rnn)
- You do not need to learn the labeling before P32
- AMR (no question on this)
- Conditional random field (exercise) computational expensive, why is it computationally expensive
- P50 sequence tagging
- LSTM no need to know formulae by heart, no time for those computations, you should know what actually we model in this
- CRF is important tech. what is the difference between normal CRF and the NN model with a CRF layer.
- P59 score(LSTM-CRF)

Overview:

- What is named entity recognition?
- What is semantic role labeling?
- Towards abstract meaning representation
- Sequence tagging:
- Conditional random fields
- Long short-term memory networks
- Neural networks with CRF top layer
- Evaluation

Primary Usage of the topics: Language understanding and question answering

What is NER?

- To find each mention of a named entity in the text and label its type.
- This task is usually application-specific. Examples:

Examples:

-
  - Commonly include people, places, and organizations
  - Biomedical domain: names of genes and proteins
  - Education: codes of college courses

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	The **Mt. Sanitas** loop is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states, provinces	**Palo Alto** is raising the fees for parking.
Facility	FAC	bridges, buildings, airports	Consider the **Tappan Zee Bridge**.
Vehicles	VEH	planes, trains, automobiles	It was a classic **Ford Falcon**.

Figure 21.1 A list of generic named entity types with the kinds of entities they refer to.

Name	Possible Categories
Washington	Person, Location, Political Entity, Organization, Vehicle
Downing St.	Location, Organization
IRA	Person, Organization, Monetary Instrument
Louis Vuitton	Person, Organization, Commercial Product

Figure 21.2 Common categorical ambiguities associated with various proper names.

As in many "information extraction" tasks, NER has two problems:

- Segmentation.
- Classification.

The task:

It can be handled in the following two ways:

- Sequence labeling:
  - Given an input sentence *X* composed of a sequence of words x, assign a sequence of labels Y where each word receives a label y. (Primary focus)
  - Alternatively, given an input sentence *X* composed of chunks x (obtained with a chunker), assign a sequence of labels Y where each chunk receives a label y.
- Tree labeling:
  - Given a constituent parse tree *X*, assign a set of labels Y where each constituent x, receives a label y
- NER classes are usually mutually exclusive (but not necessarily), so the task is often seen as a multiclass problem.

Sequence Labeling:

- This uses BIO or IOB tags.
  - Tag for the beginning (B) and inside (I) of each entity type, and one for tokens outside (O) any entity
  - B= Begin, I = Inside, O = Outside
  - 2 n + 1 tags, where n is the number of
  - entity types
  - Alternatively: IO tagging using only n + 1
  - Tags
- Example:

Datasets(skipped)

Semantic Role Labeling (SRL):

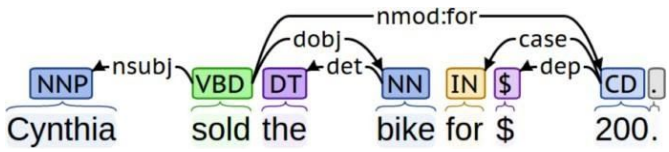
SRL involves:

- Determining which constituents in a sentence act as predicate or as semantic arguments to a predicate: segmentation.
- Determining the role for each of those arguments in the sentence: classification or recognition.
- recognizing the basic event structure of a sentence ("who" "does what" "to whom/what" "when" "where" 'how"...).
- Semantic frame labeling: cf. SRL, but frame filling might entail the analysis of several sentences.
- Key task for natural language understanding.

The task (extremely similar to NER):

- In the sequence labeling task: SRL classes are usually mutually exclusive (but not necessarily), so the task is often seen as a multiclass problem.

SRL: classification or recognition:



Simple solution: extraction rules based on output of a dependency parse:

$$\textit{sold}\ \text{nsubj}\ X \Rightarrow \text{SELLER}{:}X$$

$$\textit{sold}\ \text{dobj}\ X \Rightarrow \text{GOODS}{:}X$$

$$\textit{sold}\ \text{nmod:for}\ X \Rightarrow \text{PRICE}{:}X$$

The method is not scalable and generalizable: e.g., many alternations using syntactic alternations and using different verbs. There are resources with definitions and manually annotated examples:

- FrameNet (1998)
- PropBank (2002)
- **The details are unimportant for the exam & best understood from the slides so they are not written here.**

SRL Datasets (Skipped)

SRL pipeline:

○

Dependencies between the input variables X do not need to be explicitly represented, affording the use of rich global features of the input

- ○ We define G = (V, E) to be an undirected graph such that there is a node v ∈ V corresponding to each of the random variables representing an element y_v of Y
- ○ If each y_v obeys the Markov property (e.g., first order dependence) with respect to G, then the model(Y, X) is a conditional random field
- ○ A CRF is a discriminative model: we predict P(Y|X; θ)

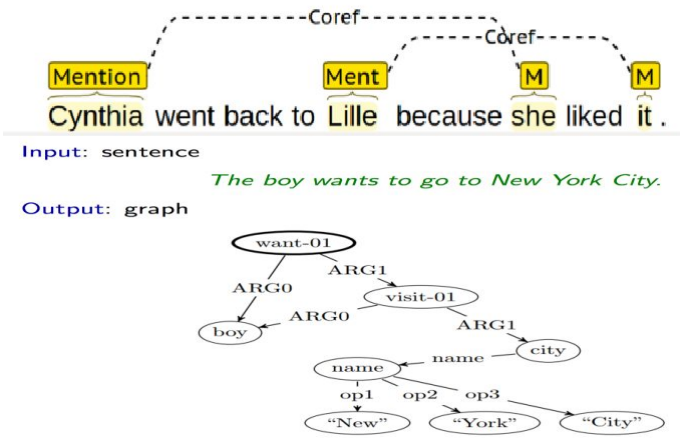
Linear chain CRF:

- $X = (x_1, \dots, x_T)$ is a random variable over data sequences to be labeled and $Y = (y_1, \dots, y_T)$ is random variable over the corresponding label/class sequences.
- X might range over the words or constituents of a sentence, while Y ranges over the semantic classes to be recognized in these sentences

Towards Abstract Meaning Representation (AMR):

Coreference resolution (CR) (covered in Ch 10-11)

AMR = SRL + NER + CR using unifying semantic annotations



Structured Predictions:

- Structured prediction refers to machine learning models that predict multiple interrelated and dependent quantities.
- E.g., NER: tagging a sequence of words in a sentence with NE information

Input:	John	is traveling	to the US.	
Output:	PERSON	NONE	|	LOCATION

- E.g., SFL: tagging semantic frames in a sentence composed of a predicate and its arguments (roles)

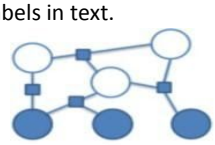
Input:	*Cynthia*	*sold*	*the bike*	*to Bob*	*for $200*
Output:	SELLER	**PREDICATE**	GOODS	BUYER	PRICE

Conditional Random Fields:

- CRFs are undirected graphical model of the family of Markov random fields or Markov networks:
- Undirected graph: There is a dependency between output labels modeled, but the dependency has no directions

General CRF:

In theory the structure of graph G may be arbitrary: e.g., template based or general CRF, where you can define the dependencies between labels in the Markov network or graph. It can capture long-distance dependencies between labels in text.



CRF

[Lafferty et al. ICML 2001]

Feature functions for CRFs:

Binary (or real valued) feature functions model features of local inputs as well as transitions between labels:

Representing a Linear CRF:

Let X, Y represent sequences, $\boldsymbol{\lambda} = \{\lambda_k\} \in \mathbb{R}^K$ be a parameter vector, and $\{f_k(y_{t-1}, y_t, X_t)\}_{k=1}^K$ be a set of real-valued feature functions, a linear CRF is the distribution $P(Y|X)$ that takes the form:

$$P(Y|X) = \frac{1}{Z(X)} \exp\left\{\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, X_t)\right\}$$

$$Z(X) = \sum_Y \exp\{\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, X_t)\}$$

Learning model parameters:

It requires usage of numerical optimisation methods:

- Quasi-Newton methods, SGD, etc.
- In general CRFs we use approximate inference (e.g., Markov Chain Monte Carlo sampler, such as Gibbs sampling)
- Training, however, is rather computationally expensive.

Prediction:

Find the most likely labeling the sequence:

$$\hat{Y} = \arg\max_{Y_l} P(Y_l|X)$$

Decoding during prediction: Viterbi, beam search.

Sequence Tagging: Long short-term memory networks

RNNs were covered in Ch 4 so all that material has been skipped.

Long short-term memory networks (LSTMs): Store an internal cell state even if that cell does not activate, allowing for semantic relations that span across words or sentences to be learned.

RNNs accumulate information but LSTMs manage to forget information and learn when to forget with a gated memory cell that replaces a standard neuron of a RNN :

- The LSTM cell consists of input, forget, and output gates that manage the information flow of the cell's internal state.
- Allows the network to remember things for a long time.
- Realized through the connection weights which are trained via backpropagation through the memory cell.

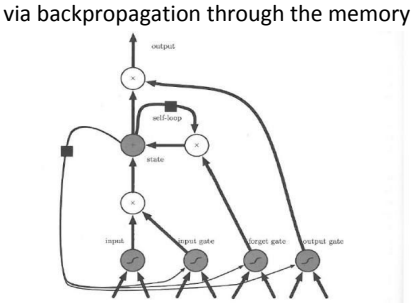


Figure 10.16: Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

[Goodfellow et al. 2016]

The equations: (Best way to understand an LSTM imo):

- LSTM cell replaces a RNN cell, so at each time step we modify:

Input gate: $\boldsymbol{i}_t = \text{sigmoid}\left(\boldsymbol{U}^{(i)}\boldsymbol{x}_t + \boldsymbol{W}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i\right)$ — to control what information is extracted from the new input $\boldsymbol{x}_t$

Forget gate: $\boldsymbol{f}_t = \text{sigmoid}\left(\boldsymbol{U}^{(f)}\boldsymbol{x}_t + \boldsymbol{W}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f\right)$ — to control what information is propagated from the previous to the next state

Output gate: $\boldsymbol{o}_t = \text{sigmoid}\left(\boldsymbol{U}^{(o)}x_t + \boldsymbol{W}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o\right)$ — to control what information of the state is exposed to the output neurons

State (memory) cell: $\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tanh\left(\boldsymbol{U}^{(c)}\boldsymbol{x}_t + \boldsymbol{W}^{(c)}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c\right)$

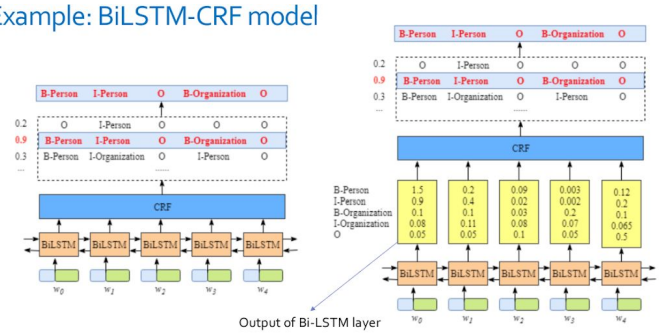
Final hidden state: $\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)$

$\circ$ denotes elementwise multiplication

- Very powerful, especially when stacked and made even deeper (each hidden layer is already computed by a deep internal network).
- Most useful if you have lots and lots of data
- Different variations on the LSTM gating are possible, e.g., the Gated Recurrent Unit (GRU)

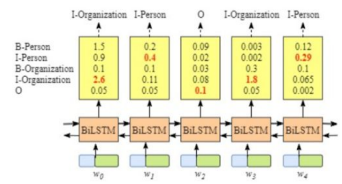
LSTMs can be the basis for much more sophisticated models:

Example: BiLSTM-CRF model



https://createmomo.github.io/2017/09/12/CRF_Layer_on_the_Top_of_BiLSTM_1/

- Without CRF layer: we would choose the sequence labeling that for each output of the Bi-LSTM chooses the label with maximum score (or probability if softmax is applied)



- The CRF layer will impose additional constraints learned from the training data: e.g., first-order Markov dependencies between labels

https://createmomo.github.io/2017/09/12/CRF_Layer_on_the_Top_of_BiLSTM_1/

$$P(Y|X) = P(Y|X) = \frac{1}{Z(X)} \exp\left\{\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, X_t)\right\} = \frac{1}{Z(X)} \exp\left\{\sum_{t=1}^{T} \boldsymbol{\lambda} \cdot \boldsymbol{\Phi}(X, Y)\right\}$$

- $\boldsymbol{\lambda} \cdot \boldsymbol{\Phi}(X, Y)$ can be considered as a score of how well the label sequence Y fits the given input sentence
- The idea is to replace the linear scoring function by the score of a non-linear neural network:

$$score_{LSTM-CRF} = \sum_{t=1}^{T} \boldsymbol{w}_{y_{t-1},y_t} \cdot LSTM(\boldsymbol{x})_t + b_{y_{t-1},y_t}$$

where $\boldsymbol{w}_{y_{t-1},y_t}$ and b_{y_{t-1},y_t} are weight vector and bias corresponding to the transition from y_{t-1} to y_t

- Then during training the conditional probability $P(Y|X; \boldsymbol{w}, b)$ is optimized like in the usual CRF and propagated back in the network: cf. loss function slide 45

Congratulations, you're halfway through the syllabus.

Here, look at a tiny hamster eating tiny burritos:

https://www.youtube.com/watch?v=JOCtdw9FG-s

Lecture 7: Semantic Parsing Overview:

- Introductory concepts
- Logical approaches based on manually drafted rule based systems
- Machine learning approaches
  - Learning to map a semantic knowledge representation
  - Learning to map directly a denotation
  - Neural network models
  - Coarse-to-fine decoding for neural semantic parsing
- Evaluation
- Problem of lack of training data and lack of commonsense world knowledge
- Link to statistical models that learn compositional semantics

<u>Introductory concepts</u>

Semantic parsing = translation of a language utterance into:

- A structured representation that captures the semantic meaning of the utterance and which can be used by a machine for further processing
- A denotation: eg. that yields an answer in case of a question or problem formulates in natural language

The technologies of this chapter primarily aim at methods that have the long-term vision of achieving deep natural language understanding.
Have applications in:
- Question answering
- Robotics
- Speech interfaces
- Machine translation
- Interfaces to databases

Given triples < υ , σ, δ > where υ is an utterance, σ is a semantic representation and δ is the denotation of σ, task is to map or translate υ to σ, or to directly compute δ from υ .

Semantic representation = a formal structure that captures the meaning of a linguistic expression
- Bridges the gap between linguistic input and the non-linguistic knowledge of the world
- Consists of structures composed from a set of symbols or representational vocabulary: often in the form of predicates and their argument, eg. entities (eg object, persons), their properties and relations
- unambiguous

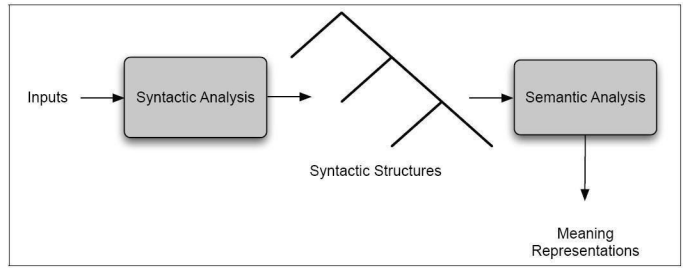
Denotation = meaning of a phrase or sentence

Traditional semantic representations are generally logical forms: expressions in a fully specified, unambiguous artificial language.
Examples:
- first order logic
- lambda calculi
- natural logics
- diagrammatic languages
- programming languages
- robot controller languages
- database query languages

but note that the artificial language is usually less expressive than natural; language and practically operates only in a limited subject domain. When expanded to open domains and high expressiveness ¾ becomes extremely complex and has same problems of natural language

Data sets:
- Geo: 880 language queries to a dataset of US geography, meaning representation based on λ -calculus
- Atis: 5410 queries to a flight book system, meaning representation based on λ -calculus
- Django: database built upon python code and Django library
- WikiSQL: contains 80654 examples of questions and SQL queries distributed across 24241 tables from Wikipedia
- Sail data: hand designed maze, action instructions for path navigation containing 3236 sentences

Logical approaches based on manually drafted rule based systems
Semantic parsing: oldest approaches: a simple pipeline approach to semantic analysis



First order logic (FOL)

John walks	$walk(john)$
John loves Mary	$love(john, mary)$
A man walks	$\exists x.\, man(x) \wedge walk(x)$
A man loves Mary	$\exists x.\, man(x) \wedge love(x, mary)$
John and Mary walk	$walk(john) \wedge walk(mary)$
Every man walks	$\forall x.\, man(x) \longrightarrow walk(x)$
Every man loves a woman	$\forall x.\, man(x) \longrightarrow \exists y.\, woman(y) \wedge love(x, y)$

Similarity with machine translation task: translation of language to logical expression

Compositionality: linked to human's ability to produce and interpret novel utterances: once a human has acquired the syntax of the language, memorized all the lexical meanings and mastered the few modes of composition, one can interpret novel combinations of words and phrases.
The meaning of the sentence is constructed from:
- The meaning of the words
- Paralleling the syntactic construction (ie. the semantic rules, manually constructed)
  1. Parse sentence to get syntax tree
  2. Look up the semantics. Of each word in lexicon
  3. Build the semantics for each constituent bottom-up, by combining the semantics of the children. Using a priori defined rules

This approach is not scalable.

Lambda abstraction: Lambda notation: operator λ to bind free variables, it marks missing information in the object language λ -FOL: exact value not predefined
- Consists of λ , one or more variables x and FOL formula in the form of predicate P that makes use of the variables
- Provides a way to abstract from fully specified FOL formula in a way that will be particularly useful for semantic analysis

  λ x.P(x)

  λ x.love(x.mary) % we abstract over x

Lambda reduction = simple textual replacement of the λ variables with the specified FOL terms, accompanied by the subsequent removal of λ .
- Eg. application of lambda reduction to constant A: λ x.P(x)A ¾ P(A)
- Eg. λ -application and subsequent reduction to a fully specified logical formula:
- λ x. $\wedge$ y.Near(x,y)(Bacaro) ¾ λ y.Near(Bacaro,y)
- λ y.Near(Bacaro,y)(Centro) ¾ Near(Bacaro,Centro)

Implementation: unification-based approach:
- FOL-representations and λ -based semantic attachments are replaced by complex featured structures and unification equations
- Feature structure and unification operators provide effective way to implement syntax-driven semantic analysis
- How to go from syntax to semantics?

Example: sentence "Rhumba closed"
Meaning representation where e=event variable: $\exists e\ closing(e) \wedge Closed(e, Rhumba)$

$$\begin{bmatrix} \text{QUANT} & \exists \\ \text{VAR} & \boxed{1} \\ \text{FORMULA} & \begin{bmatrix} \text{OP} & \text{AND} \\ \text{FORMULA1} & \begin{bmatrix} \text{PRED} & \text{CLOSING} \\ \text{ARG0} & \boxed{1} \end{bmatrix} \\ \text{FORMULA2} & \begin{bmatrix} \text{PRED} & \text{CLOSED} \\ \text{ARG0} & \boxed{1} \\ \text{ARG1} & \text{RHUMBA} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Feature structure:

Associate unification equations with the grammar rules:

S ¾ NP VP

<S SEM> = <NP SEM>

<VR ARG0> = <NP INDEXVAR>

<NP SCOPE> = <VP EM>

NP ¾ ProperNoun

<NP SEM> = <ProperNoun SEM>

<NP SCOPE> = <ProperNoun SCOPE>

<NP INDEXVAR> = <ProperNoun INDEXVAR>

ProperNoun ¾ Rhumba

<ProperNoun SEM PRED> = RHUMBA

<ProperNoun INDEXVAR> = <ProperNoun SEM PRED>

Manual drafting of rules: not scalable!

Lambda calculus abstraction: popular for small applications

Semantic parsing

First example in this chapter:

- Define possible syntactic structures using a context-free grammar
- Construct semantics bottom-up, following syntactic structure and manually drafted rules
- Possibly, use grammar induction to avoid manual grammar engineering (not covered here)

Machine learning approaches:

- Statistical algorithms and optimization
- Score possible parses with eg. a (log-)linear model learned from training data
- Neural network approaches

<u>Statistical machine learning approaches</u>

Supervised Learning:

- Training: the training examples take the form of pairs (x,y) with x ϵ X the system's input and y ϵ Y the desired output
- Prediction/interface: given a new (unseen) test example (x, ?), the trained system makes a prediction

A machine learning system has 3 components:

- A feature representation of the data
- An object function (which usually corresponds to minimizing error on training examples)
- An algorithm for optimizing the objective function during training

A linear classification to map each input-output pair (x,y) to a K-dimensional feature vector ϕ (x,y) ϵR^K where each coordinate (feature function) $f(x,y)_j$ represents something about y and its relationship to x. We associate each feature function $f(x,y)_j$ with a real-values weight wj intuitively representing how informative the feature is. The scoring function of a linear classifier then assigns a value to any (x,y) pair computing a weighted sum over the features:

$$Scorew(x, y) = \text{S}_{\text{UVW}}^{\text{T}} \; wj\, f(x, y)j = w * \varphi(x, y)$$

Given any new input x, we simply predict the output with the highest score

$$\hat{y}(x) = \underset{y \in Y}{\operatorname{argmax}}\, \text{Score}_w(x, y)$$

-

Training: given a set of examples D consisting of input-output pairs(x,y): optimize the weights W so that the score of the correct output y is generally larger that the score of the incorrect outputs y' ϵ Y \ {y}. there are many suitable objective functions for the problem, eg. multiclass hinge loss objective

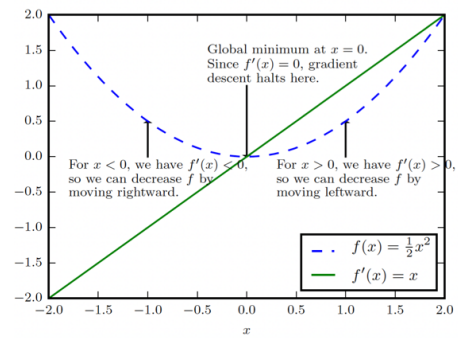
$$J(\boldsymbol{w}) = \min_{\boldsymbol{w}} \sum_{(x,y)\in\mathcal{D}} \max_{y'\in Y} \left[\text{Score}_w(x, y') + c(y, y')\right] - \text{Score}_w(x, y)$$

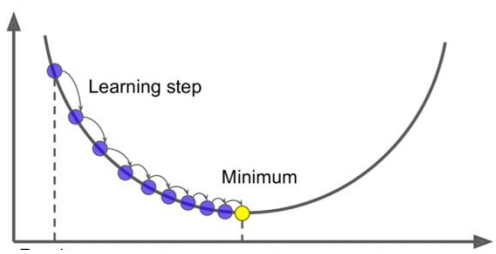
Where c(y, y') = cost for predicting y' when the correct output is y in example below = 0 when y = y', 1 otherwise. In words, we are trying to find the best weight vector w that minimized the cumulative loss over training examples D.

Gradient based optimization

Many supervised learning and most deep learning algorithms involve optimization: the function we want to minimize or maximize is called the objective function. In case of minimizing we call it cost function, loss function or error function. In gradient based optimization the derivative of the objective function is used. Suppose we have a function y=f(x) where both x and y are real numbers. The derivative of this function $^{\text{Z}}\text{L} = f^{]}(x)$. $f^{]}(x)$ gives the slope of $f(x)$ at point x and tells how much

to change x to make an improvement in y. Eg. gradient descent as minimization technique: we can thus reduce $f(x)$ by moving x in small steps with the opposite sign of the derivative.





Slope:

- A positive slope means the function goes upward as you move right, so we want to move left in order to find the minimum
- A genitive slope means the function goes downwards towards the right, so we want to move right to find a minimum

How big a step to take?

- If the slope is large we want to take a large step because we're far from the minimum
- If the slope is small we want to take a smaller step

Stochastic gradient descent

- The weights w of the network are adapted with the gradient of the objective function $^{\text{Z}^\wedge}$ scales by the learning rate η

Stochastic gradient descent is an extension and simplification of the gradient descent algorithm extensively used in machine learning and especially deep learning

- Instead of computing the gradient on the full training set, each iteration estimates this gradient on the basis of a single randomly picked example or a minibatch of examples: "stochastic" approximation of the true gradient

- Guaranteed convergence, under some conditions
- Scales to large datasets

Semantic parsing task: to learn an accurate mapping from utterances u to logical forms σ. Usually finite set of possible logical predicates and arguments, eg weather information.
Interpretation task = to learn an accurate mapping from utterances u to denotations δ via latent semantic representations, in effect combining semantic parsing and interpretation. Often involves structure prediction.

STOCHASTICGRADIENTDESCENT($\mathcal{D}, T, \eta$)
$\mathcal{D}$: a set of training examples $(x, y) \in (\mathcal{X} \times \mathcal{Y})$
T: the number of passes to make through the data
$\eta > 0$: learning rate (e.g., $\frac{1}{\sqrt{T}}$)
1 Initialize $\mathbf{w} \leftarrow \mathbf{0}$
2 Repeat T times
3 for each $(x, y) \in \mathcal{D}$ (in random order)
4 $\tilde{y} \leftarrow \arg\max_{y' \in \mathcal{Y}} \text{Score}_{\mathbf{w}}(x, y') + c(y, y')$
5 $\mathbf{w} \leftarrow \mathbf{w} + \eta(\phi(x, y) - \phi(x, \tilde{y}))$
6 Return $\mathbf{w}$

Figure 1: The stochastic gradient descent (SGD) optimization algorithm.

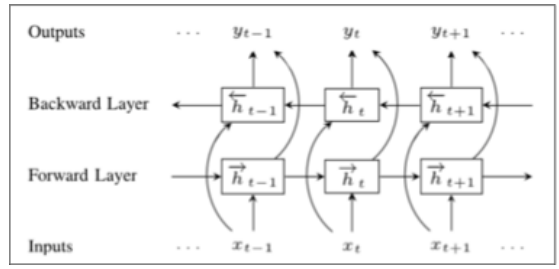
Optimization, by eg. stochastic. Gradient descent (SGD):
- Iteratively computes the (sub)gradient of a single term of the optimization problem
- For each training example (x,y) ϵ D , SGD make a prediction for it (line 4)
- When the prediction is correct (y = ŷ), the update does nothing because the difference in live 5 is 0.
- Where the prediction is incorrect, the update adjusts the weights in the direction of the correct answer (x,y) and away from the incorrectly predicted answer (x,y')
- The parameter η controls the learning rate (size of these adjustments)

Complexity: significant computational challenges:
- The number of candidate logical forms GEN(x) is in general exponential in the length of the sentence, especially when structure and modes of combination would not be given

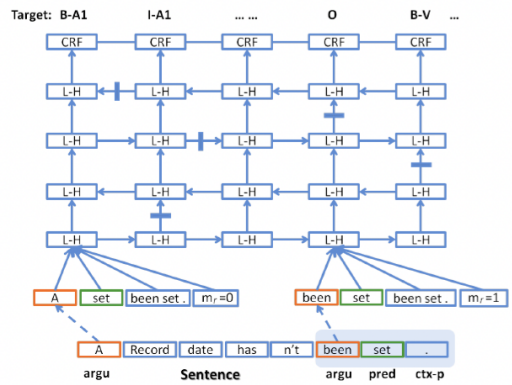
Learning to map directly to a denotation. More difficult?
Learn from denotations (meaning of word or phrase) directly, using logical forms only as intermediate, hidden representations along the way. This is more realistic portrayal of human language acquisition and is also advantageous from the point of view of developing artificial systems. Significant computational challenges, stemming from the fact that the number of candidate (logical) forms GEN(x) is in general exponential in the length of the sentence.
- Applications include querying databases
- Interpreting natural language for performing programming tasks
- Playing. Computer games
- Following navigational instructions
- Implementing dialogue systems
- Interacting in the real world via perception

Neural network models

Recent interest is sequence-to-sequence models, using bi-directional LSTM model



At the top of the network a conditional random field (CRF) model for tag sequence prediction.



These are often end-to-end systems. Original text can as the input features without any intermediate tags as syntactic information.
Problem: needs lots of training data, hence the interest in semi-supervised models, except for sequence relations, last relational and nested properties which are often important in semantic parsing.

Semi supervised model:

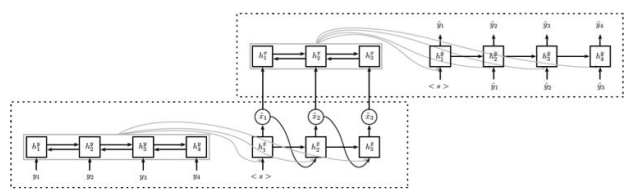


Figure 1: SEQ4 model with attention-sequence-to-sequence encoder and decoder. Circle nodes represent random variables.

SEQ4: two sequence-to-sequence models with attention chained together, consist of 4 LSTMs. The first a bidirectional LSTM, encodes the sequence y; next, an LSTM with stochastic outputs, draws a sequence of latent distributions x` over words in vocabulary x. The third bi-LSTM encodes these distributions for the last one to attend over and reconstruct y as ya .
S2S: supervised sequence-to-sequence model, when the decoder is trained without the encoder in a fully supervised manner

Coarse-to-fine decoding for neural semantic parsing

A structure-aware neural architecture which decomposes the semantic parsing process into two stages trained end-to-end. Given an input utterance, we first generate a rough sketch of its meaning, where low- level information, such as variable names and arguments, is glossed over. We fill in missing details by taking into account the natural language input and the sketch itself.
Let x = x_1 … x_T denote a natural language expression and y = y_1 … y_L its meaning representation
The conditional probability of meaning representation P(y|x) = P(y|x,a)P(a|x), where a= a_1 … a_F. Is an abstract sketch representing the meaning of y.

$$P(a|x) = \prod_{t=1}^{F} P(a_t|a_{<t}, x)$$

$$P(y|x, a) = \prod_{t=1}^{L} P(y_t|y_{<t}, x, a)$$

Where $a_{<t} = a_1 \ldots a_{t-1}$ and $y_{<t} = y_1 \ldots y_{t-1}$

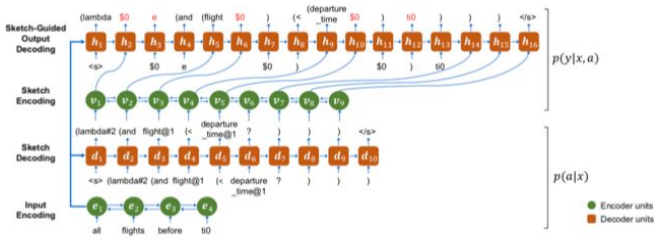


Figure 1: We first generate the meaning sketch a for natural language input x. Then, a fine meaning decoder fills in the missing details (shown in red) of meaning representation y. The coarse structure a is used to guide and constrain the output decoding.

Input encoder: Bi-directional LSTM
Coarse meaning decoder: LSTM with attention mechanism between encoder and decoder
Sketch encoder Bi-directional LSTM encoder maps the sketch sequence a into vectors $\{v_c\}$e similarity with what is done with the input encoder, where v_c denotes the vector of the k-th time step
Fine-meaning decoder: LSTM with attention mechanism, and shares the input encoder: the decoder's hidden state $\{h\}^g$ are computed via:

$$\boldsymbol{i}_t = \begin{cases} \boldsymbol{v}_k & y_{t-1} \text{ is determined by } a_k \\ \boldsymbol{y}_{t-1} & \text{otherwise} \end{cases}$$

$$\boldsymbol{h}_t = f_{\text{LSTM}}(\boldsymbol{h}_{t-1}, \boldsymbol{i}_t)$$

If y_{t-1} is determined by a_k in the sketch, ie, there is a one-to-one alignments between y_{t-1} and a_k the corresponding token's vector v_c is used as input to the next time step.

Training: objective function = maximization of the log likelihood of the generated meaning. representations given natural language expressions:

$$\max \sum_{(x,a,y)\in\mathcal{D}} \log P(y|x, a) + \log P(a|x)$$

where D represents the training pairs
Prediction: at test time, the prediction for input x is obtained via $aaP(a^{]}|x)$ and $yaP(y^{]}|x, aa)$, where a' and y' represent coarse and fine-grained meaning candidates.
Greedy search is used to generate tokens one by one, rather than iterating over all candidates.

Advantages:

1. The decomposition disentangles high-level from low-level semantic information, which enables the decoders to model meaning at different levels of granularity, sketches are more compact and as a result easier to generate compared to decoding the entire meaning structure in one go.
2. The model can explicitly share knowledge of coarse structures for the examples that have the same sketch (ie basic meaning), even though their actual meaning representations are different, due to different details
3. After generating the sketch, the decoder knows what the basic meaning of the utterance looks like, and the model can use it as global context to improve the prediction of the final details

Evaluation

-

The semi-supervised model has a small improvement (<1%) compared to the supervised learning model in the GEOQUERY dataset. It's hard talk to learn from unlabeled data. Better performance of SEQ4 on SAIL corpus

Model	Accuracy
Zettlemoyer and Collins (2005)	79.3
Zettlemoyer and Collins (2007)	86.1
Liang et al. (2013)	87.9
Kwiatkowski et al. (2011)	88.6
Zhao and Huang (2014)	88.9
Kwiatkowski et al. (2013)	89.0
Dong and Lapata (2016)	84.6
Jia and Liang (2016)[6]	89.3
S2S	86.5
SEQ4	87.3

Table 2: Non-neural and neural model results on GEOQUERY using the train/test split from (Zettlemoyer and Collins, 2005).

Model	Accuracy
Chen and Mooney (2011)	54.40
Kim and Mooney (2012)	57.22
Andreas and Klein (2015)	59.60
Kim and Mooney (2013)	62.81
Artzi et al. (2014)	64.36
Artzi and Zettlemoyer (2013)	65.28
Mei et al. (2016)	69.98
S2S	58.60
SEQ4	63.25

Table 6: Results on the SAIL corpus.

Model	Accuracy
Haas and Riezler (2016)	68.30
S2S	78.03

Table 4: Results on the NLMAPS corpus.

- Geo and Atis: 88.2 % and if we add oracle sketch we climb to 93,9%
- If we do it with Python (DJANGO) , the accuracy is lower

Semantic parsing in general:
Successful semantic parsing

Method	Accuracy
Retrieval System	14.7
Phrasal SMT	31.5
Hierarchical SMT	9.5
SEQ2SEQ+UNK replacement	45.1
SEQ2TREE+UNK replacement	39.4
LPN+COPY (Ling et al., 2016)	62.3
SNM+COPY (Yin and Neubig, 2017)	71.6
ONESTAGE	69.5
COARSE2FINE	74.1
− sketch encoder	72.1
+ oracle sketch	83.0

Table 3: DJANGO results. Accuracies in the first and second block are taken from Ling et al. (2016) and Yin and Neubig (2017).

Method	Accuracy	Execution Accuracy
SEQ2SEQ	23.4	35.9
Aug Ptr Network	43.3	53.3
SEQ2SQL (Zhong et al., 2017)	48.3	59.4
SQLNET (Xu et al., 2017)	61.3	68.0
ONESTAGE	68.8	75.9
COARSE2FINE	71.7	78.5
− sketch encoder	70.8	77.7
− table-aware input encoder	68.6	75.6
+ oracle sketch	73.0	79.6

Table 4: Evaluation results on WIKISQL. Accuracies in the first block are taken from Zhong et al. (2017) and Xu et al. (2017).

Method	GEO	ATIS	DJANGO	WIKISQL
ONESTAGE	85.4	85.9	73.2	95.4
COARSE2FINE	89.3	88.0	77.4	95.9

Table 5: Sketch accuracy. For ONESTAGE, sketches are extracted from the meaning representations it generates.

approaches in limited domains eg,

- mapping questions to structured databases in domains of US geography, job postings, airtravel planning
- map instructions to robot programming language or to a robot's programming
- models for question-answering on Freebase across 81 different domains

MUSE project = bringing text to life

- Translate language to knowledge representation and then to 3D images in VR with Unity
- Machine understands the stories and animates them

What do we recognize in the text?
At the sentence level: actions/events and their semantic roles (actor, patient, instrument. …)
At the discourse level: co-referent noun phrases, temporal relations between actions, spatial relations between objects
No end-to-end system because of lack of training data for the mapping to the knowledge representation
Machine understands the stories and render them as 3D virtual worlds. To truthfully render the content of a story in a virtual world and to contextually adapt to this world: a large amount of commonsense world knowledge is needed because content is not made explicit in language and communicated in a more abstract way.

Today, distributional semantics: learning conventional aspects of natural language meaning from corpora, databases and images

- Novel ways of representing rich linguistic models
- Novel ways for computational semantics and compositionality
- Novel ways of treating ambiguity, inference and grounding (eg. inference in question answering)

Novel paradigm semantic representations are vectors, matrices and tensors. No need for. Feature engineering. Compositionality is obtained by mathematical operations for obtaining compositionality and training neural networks where denotations are outputs. One of the most exciting open questions in computational semantics is what kind. Of data and models will be needed to operate in richer denotation domains.

Conclusions

Semantic parsing shifts from logical to statistical approaches. Statistical and neural approaches are promised, but when they avoid linguistic features and grammar rules, they need many training examples to learn a valuable model, especially in open domains. Interest in learning models that better reflect how humans learn to understand language.

Lecture 8: Machine Translation 1.0

Overview:

Intro to machine translation
Why machine translation is difficult?
Statistical machine translation:

- IBM model 1 and expectation maximization algorithm (alignment probabilities of a word and the target language)
- Phrase based machine translation

Old task, goes back to the 50s. started in the 2nd WW when people were coding and decoding information. Then we got the idea to use similar techniques for machine translation.

- Machine translation initially was rule based
- In the 80s, companies (eg. IBM) came up with statistical solutions, using data
- Recently, the big companies are leading neural machine translation eg. Google
- Currently, it's not perfect, but relatively good

Today there are more than 7000 languages spoken in the world. The rule based systems are not scalable

Machine translation = task of translating a sentence x from one language (source language) to a sentence y in another language (target language)

- Focus on data driven methods: usually trained on parallel data
- Traditionally rule based systems were used, but data driven methods have best performance
- Current research: using comparable data, with similar content, but not exact translation and monolingual data eg. Wikipedia

Datasets:

- LDC, Linguistic data Consortium
- Europarl corpus
- OPUS: open source documentation
- Acquis Communautaire: legal documents signed by EU member countries

Why machine translation is difficult?

- Typology of language
- Structural divergences
- Lexical divergences

Language typology: Systematic cross-linguistic similarities and differences
Universals: true for every language
Languages have sentences that consists of words with a pronunciation and a meaning
All languages have vowels and consonants All languages have nouns and verbs

Morphology

- Number of morphemes per word
  - Isolating eg Vietnamese, Cantonese: words generally have one morpheme
  - Polysynthetic eg Siberian: words may have. many morphemes
- Segmentable morphemes
  - Agglutinative: morphemes have relatively clean boundaries
  - Fusion: a single morpheme may conflate multiple morphemes

Syntax: word order in simple declarative clauses

- SVO (subject-verb-object): English, French, German
- SOV: Hindi, Japanese
- VSO: Hebrew,

Arabic Word order in noun phrase

- NP: ADJ N (eg. EN, GE, NL)
- NP: N ADJ (eg. SP, FR)

Typology: argument structure

- Linking of predicates with arguments
  - Headmarking: mark relation between head and dependents on the head
  - Dependent marking: mark relation on the non-head Typology:
- verb-framed: mark the direction of motion on the verb while manner is framed on the satellite
- satellite-framed: mark the direction of motion on the satellite, eg use of particles in English

Typology: referential density: amount of pronouns used

- Pro-drop languages
- Referentially sparse languages
- Referentially rich languages

Other differences:

- Structural differences: presentation of dates, idiosyncratic syntactic constructions
- Lexical divergences: different kinds of polysemy, translation to different POS, morphological differences, lexical gap

Word alignment
Spurious words: addition of NULL as a dummy English word, so not all foreign words have to map onto one single English word
One-to-many alignment Many-to-one alignments
Many-to-many alignments

<u>Statistical Machine Translation</u>

IBM model 1 and Expectation Maximization algorithm

Core idea: learn a probabilistic model from data. Eg. translate from French to English: we want to find the best English sentence y, given the French sentence x: $P(y|x)$

Use Bayes rule to break down p(y|x) into two components that are learned separately:

$$P(x) \propto P(y)P(y)$$

Translation model: models how words and phrases should be translated, learned from parallel data

We use a decoder to find the translation y with highest probability given x

Alignment = word or phrase level correspondence between two languages

Task is to align a target word (or phrase) in the target sentence to a word (or phrase) in the source sentence, ie. Learning P(x,a|y), as a combination of many factors, such as: probability of alignment of a particular word at a position in a sentence, probability of particular words having a particular fertility (=distribution indicating how many words a word usually translates to) etc

IBM Model 1: when we find these alignment, we can use the probabilities to translate full sentences. It uses only lexical translation probabilities.

- For a foreign sentence $f = (f_{W}, \dots, f_{kl})$ of length lf
- To an English sentence $e = (e_{W}, \dots, e_{km})$ of length le
- With an alignment if each English word e_{U} to a foreign word f_{n} according to the alignment function a, is computed as:

$$P(e, a|f) = \frac{\epsilon}{(l_f + 1)^{l_e}} \prod_{j=1}^{l_e} t(e_j | f_{a(j)})$$

Where parameter ε is a normalization constant and t refers to the translation probability, here of a foreign word to an English word.

$(l_f + 1)^{le}$ = number of possible alignments. Between foreign and English sentence taking into account l_f foreign words, + the null word as input.

das		Haus		ist		klein	
e	$t(e\|f)$	e	$t(e\|f)$	e	$t(e\|f)$	e	$t(e\|f)$
the	0.7	house	0.8	is	0.8	small	0.4
that	0.15	building	0.16	's	0.16	little	0.4
which	0.075	home	0.02	exists	0.02	short	0.1
who	0.05	household	0.015	has	0.015	minor	0.06
this	0.025	shell	0.005	are	0.005	petty	0.04

$$P(e, a|f) = \frac{\epsilon}{5^4} \times t(\text{ the } | \text{ das }) \times t(\text{ house } | \text{ Haus }) \times t(\text{ is list }) \times t\text{ (small)lklein)}$$

$$= \frac{\epsilon}{5^4} \times 0.7 \times 0.8 \times 0.8 \times 0.4 = 0.0029\epsilon$$

Learning the lexical translation probabilities: use an Expectation Maximization (EM) algorithm

1. Initialize model parameters
2. Assign probabilities to latent data: probability of the alignments, compute fractional counts for the complete data (expected counts) ¾ expectation step
3. Re-estimate the model parameters ¾ maximization step
4. Iterate steps 2-3 until convergence

Training is done with a set of sentence pairs S where S_{ef} = {e,f}

Toy example:

Green house ¾ casa verde The house ¾ la casa

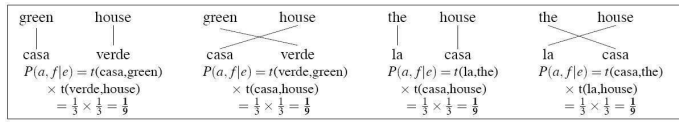
Vocabulary of English language = {green, house, the}

Vocabulary of Spanish language = {casa, verde, la} Start with uniform word pair probabilities

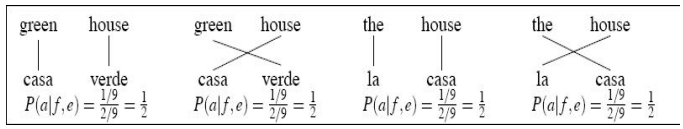
t(casa|green) = $^{W}_{o}$ –	t(verde|green) = $^{W}_{o}$ –	t(la|green) = $^{W}_{o}$ –
t(casa|house) = $^{W}_{o}$ –	t(verde|house) = $^{W}_{o}$ –	t(la|house) = $^{W}_{o}$ –
t(casa|the) = $^{W}_{o}$ –	t(verde|the) = $^{W}_{o}$ –	t(la|the) = $^{W}_{o}$ –

Expectation step: for each sentence there are 2 possible alignments

- Compute the sentence probability by multiplying the word pair probabilities



- And then normalize to obtain the alignment probability



- Compute fractional counts over all sentence pairs by weighting each count by P(a|e,f): Counts x alignment probability

Maximization step: compute max likelihood estimation: normalize (sum to 1)

- Iterate using improved estimators
-

Higher order IBM models: become more complex

- IBM model 1: lexical translation
- IBM model 2: adds absolute reordering model: alignment model based on the positions of input and output words: translation of a foreign input word at position i to an English word at position j: alignment probability distribution
- IBM model 3: adds fertility model, probability distribution of how many words a word translates to, including the modeling of a null insertion
- IBM model 4: uses relative reordering model/relative distortion: the placement of an input work is typically based on the placement of the translation of the proceeding input word
- IBM model 5: fixes deficiencies

Symmetrize alignment models: from source to target and from target to source, we

- Train separate word-alignment models
- Take intersection of those 2: higher precision, might get lower recall
- Take the union: higher recall, might lower precision
- Refinement methods: start from intersection and add some points from the union

Growing heuristics:

- Good alignment points are neighbors of other alignment points
- Starting from alignment points in the intersection, neighboring candidate alignment points from the union are progressively added
- Popular method: grow-diag-final: use of diagonal neighboring points

grow-diag-final(e2f,f2e)

```
neighboring = {(-1,0),(0,-1),(1,0),(0,1),(-1,-1),(-1,1),(1,-1),(1,1)}
alignment A = intersect(e2f,f2e); grow-diag(); final(e2f); final(f2e);
```

grow-diag()

```
while new points added do
    for all English word e ∈ [1...e_n], foreign word f ∈ [1...f_n], (e, f) ∈ A do
        for all neighboring alignment points (e_new, f_new) do
            if (e_new unaligned OR f_new unaligned) AND (e_new, f_new) ∈ union(e2f,f2e) then
                add (e_new, f_new) to A
            end if
        end for
    end for
end while
```

final()

```
for all English word e_new ∈ [1...e_n], foreign word f_new ∈ [1...f_n] do
    if (e_new unaligned OR f_new unaligned) AND (e_new, f_new) ∈ union(e2f,f2e) then
        add (e_new, f_new) to A
    end if
end for
```

The algorithm starts with the intersection of the alignments.

Growing step: neighboring points that are the union but not in the intersection are added.
Final step: alignment points for words that are still unaligned are added.

Phrase based machine translation

Translate phrases as atomic units: when translating foreign language sentence into English sentence:

- Foreign input is segmented into phrases
- Each phrase is translated into English
- Phrases are reordered

So we need to learn probability of the translation of a phrase

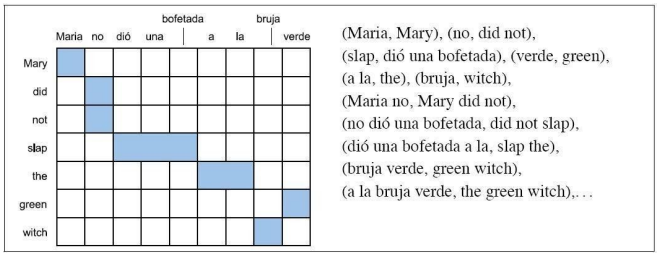
1. Compute word alignments, eg. using an IBM model
2. Extract phrase pairs
3. EM training of the phrases model

¾ From the diagonal matrix, extract translation pairs that can be phrases and then reorder. Apply again expectation Maximization algorithm to obtain probabilities

Start from word alignments in intersection and incrementally add alignments from union (eg. with grow-diag-final).

Harvest phrase pairs that are consistent with this word alignment:

- All the words are aligned with each other
- None of the words are aligned externally



Toolkits: those IBM models are open source:

- Gisa++: implementation of IBM word based models
- Moses: implementation of phrase based decoder

In practice:

- SMT has been successful for many years as the result of a huge amount of research
- Many subcomponents are language dependents, so for different languages, they need to be rebuilt
- Neural models: we do not depend on premade language features, they are learned during training by the model itself

The best systems are very complex:

- They have many separately designed subcomponents
- Lots of feature engineering and machine learning used in subcomponent models
- Require compiling and maintaining extra resources (eg. tables of equivalent phrases)
- Lots of human effort to maintain, repeated effort for each language pair

Lecture 9: Machine Translation 2.0

Overview:

Neural Machine Translation

- RNN based architectures
- Attention models
- Self-attention and transformer architectures
- Improvements and alternative architectures

Evaluation

Neural Machine Translation (NMT)

- Goal is to learn/estimate P(y|x), where
- $x = x_1, x_2, \ldots, x_N$: source sentence
- $Y = y_1, y_2, \ldots, y_N$: target sentence = translation

$P(y|x) = P(y_1|x)\ P(y_2|x,y_1) \ldots P(y_T|x,y_1,\ldots, y_{t-1})$: probability of the next target word, given the target words for far and source sentence x, Conditioned on the source sentence (x) and previous words on target sentence / translated sentence $(y_1, y_2, \ldots, y_N)$

$P(y_T|x,y_1,\ldots, y_{t-1})$ is parametrized by a **neural network with a deep learning end-to-end model** that computes the probability of the next

target word given the source sentence and the previous target words. Unlike traditional phrase based translation which consists of many small sub-components that are tuned separately, in NMT a single, large neural network is trained that reads a sentence and outputs a correct translation.

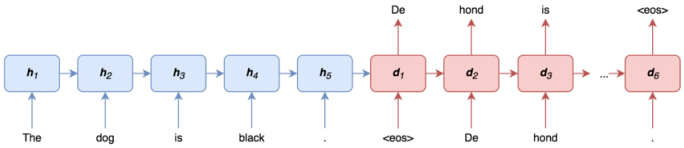
In ML we learn alignments between phrases. The translation model is built upon phrases.

Many proposed NMT models use an **encoder-decoder architecture**:

- The encoder neural network reads and encodes a source sentence into a fixed-length vector
- A decoder then outputs a translation from the encoder vector
- This model is also a sequence-to-sequence model: both the input and output form a sequence of words
- The encoder-decoder network is jointly trained to maximize the probability of a correct translation given a source sentence

Learn MT (o_t) parameterized by a neural network with a deep learning end-to-end model

- Encoder maps a source language sequence x to one or more fixed- length vectors
- Decoder predicts target language sequence y symbol by symbol using the source sequence vector representation and previously predicted symbol
- During training, the model will learn the weights of the encoder and decoder.
- Sequence-to-sequence model: both the input and output form a sequence of words.
- We learn which word to output at each timestep/position
- Output: vector with scores about each word, we use a softmax function to get the probabilities of the output words



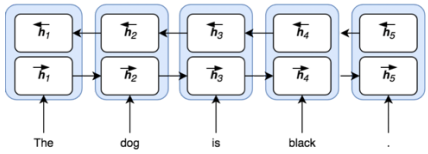
RNN based architecture

We use an RNN for the encoder and the decoder

- Encoder: RNN that encodes a source sentence x into a fixed-length vector
- Decoder: RNN language model conditioned on the fixed-length encoder vector that generates y
- The encoder-decoder network is jointly trained to maximize the probability of a correct translation given a source sentence
- Very often, output words are conditioned on the encoding vector and an embedding of the output words from the decoder
- Bidirectional RNN (eg. LSTM or GRU cells) is the most common encoder architecture
- The input is a sequence of words x, each word vector x of x is mapped into the embedding space by Wx
- It helps to better remember and

forget Bi-directional RNNs

$$\vec{h}_i = RNN\left(\vec{h}_{i-1}, Wx\right)$$
$$\dot{h}_i = RNN\left(\overleftarrow{h}_{i+1}, Wx\right)$$
$$h_i = \left(\vec{h}_i; \hbar_i\right)$$



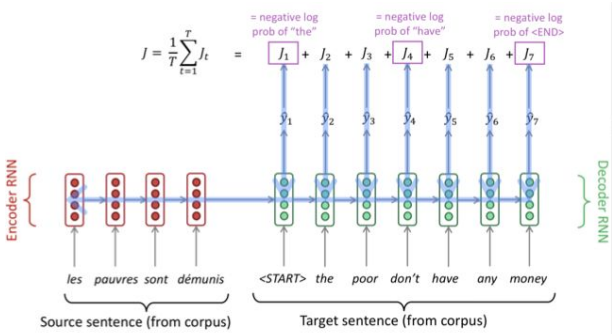
RNN Decoder (without attention)

- An RNN (eg. LSTM or GRU cells) is the most common decoder architecture
- Hidden state d_t at time step t is given by: d_t = RNN(d_{t-1}, Vy_{t-1}), where Vy_{t-1} = embedding of previous target word
- From hidden state d_t, Score vector $\mathbf{o}_t$ with as dimension the size of the target vocabulary is computed: $\mathbf{o}_t = Ud_t + b$
- $\mathbf{o}_t$ is then normalized to a probability distribution with the softmax activation function: softmax($\mathbf{o}_t$) => $P(y_t|x,y_1, \ldots, y_{t-1})$

Training:

- Training based on large parallel corpus
- The NMT is trained end-to-end with backpropagation
- Loss function for a sentence pair x-y in the parallel Corpus: Minimize cross-entropy = maximize log-likelihood

$$J(\theta) = -\log P(y|x;\theta) = -\frac{1}{T}\sum_{t=1}^{T}\log P(y_t|x, y_1, \ldots, y_{t-1};\theta)$$



Overall organization of the training

- Shuffle corpus of parallel sentence pairs
- Break into mini-batches
- Process mini-batch, update parameters
- Once done, repeat
- Typically 1-15 epochs needed (= passes through entire training corpus)

Decoding / Inference

- How do we use $P(y_t|x,y_1,\ldots,y_{t-1})$ to generate a translation? We want to find the whole target sentence with max probability, but what happens computationally when we have a huge corpus?
- <u>Strategy 1</u>: **greedy decoding**: we generate the target sentence word by word by emitting the word with maximum probability (we never backtrack, taken decisions cannot be undone)
- <u>Strategy 2</u>: at the other extreme we can enumerate all possible target sentences y and choose the most likely based on P(y|x): optimal solution, but complexity of $O(|V|^T)$ where V = vocabulary size and T the sequence length
- <u>Strategy 3</u>: **beam search** at every step t the decoder keeps track of the k most likely continuations of the k partial translations of the previous step where k is the beam size (eg. between 5 and 10)

Current research focuses on:

- What is the best way to present the source sentence to the encoder
- Giving the decoder the ability to attend to different parts of the input during translation
- Interesting work on improving how these systems handle idiomatic language

NMT: Attention models (important in NLP, CV, Visual Question Answering)

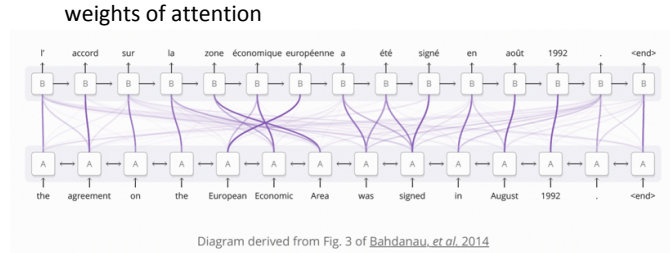
Attention is used with many different architectures. Problem of RNN encoder-decoder architecture: represents a variable length sequence with a fixed length vector

- needs to compress all the necessary information of a source sentence into a fixed-length vector
- does not work well for long sentences

When human translate a sentence, they pay special attention to the word they are presently translating

Attention mechanism: in NNs focuses on a subset of the information they are given, it learns at which parts of the encoder and decoder needs to focus

- Eg an RNN can attend over the output of another RNN: at every time step, it focuses on different positions in the other RNN
- During training the network learns where to focus: the system focuses everywhere but to different extends ("attention distribution")
- When decoding: it will look at all vector of encoder but with different weights.
- We want to learn at each decoding step, which are the weights of attention



Task: given what is generated so far, on which representations of the input (encoder hidden states) should the decoder hidden state pay attention to?

- Given the previous hidden state of the decoder d_{t-1}
- Given the representation of an input word at time t: $h_n = (h\mathbf{r}\vec{r}\iota;\ hj)^{\leftarrow}$:
- Compute an **attention score $a(d_{t-1}, h_i)$**: modelled with a feed- forward NN
- **Normalize** the attention weights with a softmax function

$$\alpha_{ti} = \frac{\exp(a(\boldsymbol{d}_{t-1}, \boldsymbol{h}_i))}{\sum_{k=1}^{N}\exp(a(\boldsymbol{d}_{t-1}, \boldsymbol{h}_k))}$$

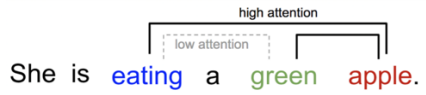
- Now the input context is weighted with the attention to form a **context vector $c_t = \sum a_{ti} h_i$** : Sum of all hidden states of encoder weighted with attention score
- **Decoder hidden layer** at time 1: $d_1 = RNN(d_0, (c_1, y_0))$
- **Decoder hidden layer** at time 2: $d_2 = RNN(d_1, (c_2, y_1))$
- The attention weights are different of each decoding vector
- The probability of the output is conditioned on a context vector which contains the weighted sum of a sequence of annotations where the weights simulate alignment scores that reflect how well the inputs around a word and the output at a certain position match.

NMT: Self-attention and transformer architecture

- Change RNN to something that allows you to do self-attention ¾ attention to the source language in the encoder model, and the target language in the decoder model
- Transformer model not relying on RNN but only used attention to model dependencies between input and output
- RNN are replaced with:
  - Feed-forward NNs

- Self-attention
- Multi-head attention

- Deep model
- Some optimization tricks are regularization tricks
- Faster training
- We have an encoder and decoder: the encoder feeds the decoder
- Refine the representation of a word by considering the representation of related words in the sentence



- Can be easily parallelized faster in training
- Could yield more interpretable models
- We build a context vector: weighted combination of hidden nodes

Self attention / intra attention = within a language

- Attention model is similar to the one seen before, but now we attend on representations of words in the same language
- Is useful when decoding long sentences to keep track of what you already translated

Multi-head attention

- Instead of one context vector c_t, people have design more context vectors to learn different self-attentions $c^{W_1}, c^{V_1}, \dots, c^{W_n}$, each with a different set of attention weights
- Intuition: different heads can focus on a different type of information, they will capture different linguistic phenomena eg:
  - Which word to focus when generating an output
  - Which context word to focus

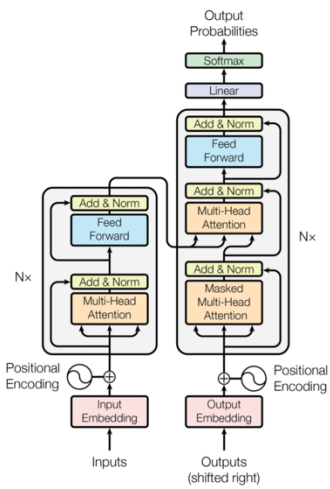


Figure 1: The Transformer - model architecture.

- Paper Vaswani et al. 2017: m r, decoder and between encoder and decoder
- Lot's of parameters to be learned during training, need for lots of data of parallel sentences. This model captures a lot of features in the data, they perform very well
- In RNN we have positional information important in language, it can switch the meaning of a sentence
- Attention mechanisms do not inherently take into account sequence order. Not to lose this info, it is added to the input vectors themselves. In RNN-based architectures this is not necessarily because the RNN learn to add information about the sequence order automatically
- Same for feedforward: positional information can be encoded in the network, and it's taken into account in the encoder and decoder

Improvements and alternative architectures

- Combine Transformer (multi-head attention, regularization tricks) with RNN based architectures
- Ensembles of models: train multiple models eg uding different random initializations, using dumps of earlier iterations, and combine the predictions
- CNNs: the kernel has a bigger receptive field, when going to deeper layers
- CNNs are often used for classification of texts, machine translation and other NLP tasks
- We can have a CNN as an encoder or decoder, define self-attention between the 2 models
- CNN with attention: Facebook paper 2017, A novel. Approach to neural machine translation
- Other: subword level modelling, character level modelling, improvements of loss function
- Using Extra inputs: a priori assigned linguistic annotations eg POS tags, morphological features, at encoder and decoder, concatenate with other input vectors
- Using extra outputs: a priori assigned linguistic annotation eg POS tags, morphological features, puts more constraints on the output to learn to better predict the outputs words

Evaluation

- BLEU scores: to evaluate translation, not optimal measure for machine translation ¾ we need better measures
- The bigger/deeper model the better
- Quite high different in training cost
- Influence of model depth, no of layers in encoder/decoder ¾ we can stack the layers
  - There is a gain when we have 4 layers in encoder and decoder, it better captures the features in the data
- Models with attention perform better vs models without attention
- Unidirectional / bidirectional: bi-d are better models, with 2 layers
- however not spectacular gain
- Humans do better job at translation of Documents. Machines are better in sentence level

Unsolved problems:

- Biases in training data are modeled
- Domain mismatch between train and test data
- Maintaining context over longer text
- Low-resource language pairs
- Sparse data: words occur only once of twice
- Unusual meanings of words

Current research:

- What is the best way to present the source sentence to the encoder
- Giving the decoder the ability to attend to different parts of the input during translation
- How to handle idiomatic language eg "pulling someone's leg"
- Unsupervised machine translation
- Combining SMT and NMT.

Lecture 10: Discourse Analysis

Discourse = capturing meaning involving multiple sentences In this chapter:

- Resolution of coreferring expressions in a sentence and discourse
- Recognition of rhetorical relations and argumentation mining

The technologies of this chapter primarily have applications in language understanding and question answering, machine translation

Overview:

- Discourse coherence
- Noun phrase coreference resolution:
  - What is coreference resolution and noun phrase coreference resolution?
  - Seminal rule based algorithms
  - Feature based learning of coreference relationships
  - Inference with integer linear programming as postprocessing step
  - Neural end-to-end coreference resolution
  - Evaluation
- Recognition of rhetorical discourse structure
  - Introduction to rhetorical structure theory and its recognition
  - Argumentation mining
  - Evaluation

Discourse coherence

Language normally does not consist of isolated, unrelated sentences, but of collocated, structure coherent groups of sentences = discourse

- Monologue (treated in this chapter)
- Human-human dialogue
- Human-computer dialogue (see chatbots)

Coherence refers to the meaning relation between textual units, ie, how they can combine to build a discourse meaning for the larger unit, eg

- Entity based coherence
- rhetorical coherence

Cohesion refers to the way textual units are linked together.

Entity based coherence = lexical coherence
Repetition of terms, synonyms, hypernyms and other coreferring expressions lexical chains

John went to his favorite music store to buy a piano. He had frequented the store for many years. He was excited that he could finally buy a piano. He arrived just as the store was closing for the day.	John went to his favorite music store to buy a piano. It was a store John had frequented for many years. He was excited that he could finally buy a piano. It was closing just as John arrived.

Rhetorical coherence example:
"because no deal is in place, spending cuts will take effect Friday.":
"because" acts as a clear indicator of causality
"with no deal in place spending cuts will take effect Friday." :
expresses the same causality with a highly ambiguous preposition "with".
"No deal is in place, the spending cuts will take place effect Friday." : no lexical indicators are present

Entity based coherence:

- because we avoid repeating the same words in our discourse or we denote certain referents in the discourse by shorter or other words coreference resolution and ana(cata)phora resolution is important
- applicable to noun phrase references, temporal references, spatial references, events

rhetorical coherence: use of connecting rhetorical relationships

<u>Noun phrase coreference resolution</u>

What are coreference and noun phrase coreference resolution?

Example: Clinton was the third-youngest president, behind Theodore Roosevelt (the youngest president) and John F. Kennedy (the youngest to be elected president). He was the first baby boomer president.
The husband of Hillary Clinton ...

Usually a priori segmentation step that generates candidate mentions: mention is here a noun phrase, including a pronoun "phrase"

- Coreference resolution = determining which NPs in a text or dialogue refer to the same entity in the discourse
- Anaphor resolution = to identify an antecedent for an anaphoric NP (often anaphoric pronouns)

A document is often processed as one would read it, at the encounter of a phrase to be resolved, the text is processed backwards in the search for potential antecedents (ignoring cataphor resolution) Result: NP mentions form an entity cluster or entity chain

Seminal rule based algorithms

Rule based algorithms for anaphora resolution

- Hobbs algorithm
- Centering algorithm

Both algorithms require syntactic parse tree and gender, number and person information being available

- Number information returned by the parse tree
- Person information of the pronouns can be easily encoded by rules
- Gender: eg.
  - Use of ancestor information in WordNet
  - List of personal names with gender

The Hobbs algorithm for pronoun resolution

Input syntactic parse of the discourse up to and including the current sentence
Given pronoun to be resolved: search for candidate referents:

- Walk up the parse tree to the node S
- For each NP or S node, breadth-first, left-to-right search of the node's children to the left of the target
- If no candidates found, breadth first, left to right search or preceding sentences
- Search strategy approximates binding theory, recency and grammatical role preferences
- Always check for gender, person and number constraints

Example: the castle in Camelot remained the residence of the king until 536 when he moved it to London.

Hobbs Algorithm:

1. Begin at the noun phrase (NP) node immediately dominating the pronoun.
2. Go up the tree to the first NP or sentence (S) node encountered. Call this node X, and call the path used to reach it p.
3. Traverse all branches below node X to the left of path p in a left-to-right, breadth-first fashion. Propose as the antecedent any NP node that is encountered which has an NP or S node between it and X.
4. If node X is the highest S node in the sentence, traverse the surface parse trees of previous sentences in the text in order of recency, the most recent first; each tree is traversed in a left-to- right, breadth-first manner, and when an NP node is encountered, it is proposed as antecedent. If X is not the highest S node in the sentence, continue to step 5.
5. From node X, go up the tree to the first NP or S node encountered. Call this new node X, and call the path traversed to reach it p.
6. If X is an NP node and if the path p to X did not pass through the Nominal node that X immediately dominates, propose X as the antecedent.
7. Traverse all branches below node X to the left of path p in a left-to-right, breadth- first manner. Propose any NP node encountered as the antecedent.
8. If X is an S node, traverse all branches of node X to the right of path p in a left-to- right, breadth-first manner, but do not go below any NP or S node encountered. Propose any NP node encountered as the antecedent.
9. Go to Step4

Centering Theory

Single entity is centered on at a given point of the discourse (cf. entity based coherence)

Centers are semantic entities (= discourse referents) that are part of the discourse model for each utterance in a discourse segment.

U_n and U_{n+1} are two adjacent utterances.
$C_f(U_n)$ = the set of discourse entities (forward looking centers) evoked by an utterance U_n
The entities in $C_f(U_n)$ are ordered according to discourse salience: eg. using grammatical role hierarchy:
Subject > existential predicate nominal > object > indirect object or oblique > demarcated adverbial PP,
Where C_p = preferred center = highest ranked forward looking center
$C_p(U)$ = backward looking center = entity currently being focused on in the discourse after U_n is interpreted

The algorithm defined four intersentential relations between a pair of utterances U_n and U_{n+1}
Two factors:

- Whether the backward looking center C_b stays the same in U_n and U_{n+1}
- Whether this discourse entity of U_{n+1} is the same as its preferred forward looking center C_p

Table with possible transitions:

	$C_b(U_{n+1}) = C_b(U_n)$ or Undefined $C_b(U_n)$	$C_b(U_{n+1}) \neq C_b(U_n)$
$C_b(U_{n+1}) = C_p(U_{n+1})$	Continue	Smooth-Shift

Algorithm:

1. Generate possible C_b - C_f combinations for each possible set of reference assignments
2. Filter by constraints, eg syntactic coreference constraints, selectional restrictions, centering rules and constraints
3. Rank by transition orderings

The following rules are used by the algorithm:
Rule 1: if any element of $C_f(U_n)$ is realized by a pronoun in U_{n+1}, then $C_b(U_{n+1})$ must be realized. As a pronoun also = continuation of pronouns
Rule 2: transition states are preferentially ordered: Continue > Retain > Smooth-Shift > Rough-Shift

Examples, used in pronoun resolution with centering algorithm :

1) John saw a beautiful 1961 Ford Falcon at the used car dealership (U1)
   He showed it to Bob.
   (U2) He bought it. (U3)
2) Bob opened a new dealership last week.
   (U1) John took a look at the Fords in his lot. (U2) He ended up buying one. (U3)

Feature based learning of coreference relationships

Machine learning models for noun phrase coreference resolution A traditional approach is composed of two-steps:

1. *Supervised classifier* if trained that locally classifies a candidate coreference relation between two entity mentions (= pairwise relation classifier) relying on number of features
2. Followed by an *optimization step* that *globally* optimizes the recognition of coreferences in the discourse: eg
   - Clustering of the mentions with as similarity the probability of the coreference relationship
   - Global optimization via integer linear programming

Pairwise coreference relation classifier

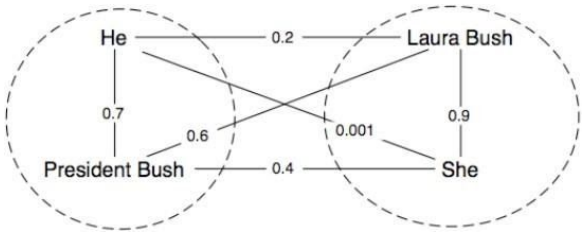
- Input: X is the set of pairs of noun phrases where $x_{ij} = \{m_i, m_j\}$
- Output: Y is the set of variables representing each pairwise coreference decision: binary random variable $y_{ij} = 1$ is mi and m_j are coreferent, $y_{ij} = 0$ otherwise
- $P(y_{ij} | x_{ij})$: computed with a classification model: eg.

  logistic regression classifie
- Training:
  - Positive examples: pair of noun phrases that exhibit the coreference relation
  - Negative examples: pairing the noun phrases with intervening noun phrases

Features for noun phrase coreference resolution
Until recently hand-engineered features based on automatically produced parse trees.
Simple features yield state-of-the-art performance for English:

- Mention type (nominal, proper, or pronominal)
- Complete string of a mention
- Semantic head of a mention
- First word and last word of each mention
- Word immediately preceding and word immediately following a mention
- Mention length, in words
- Two distance measures between mentions (based on number of sentences and number of mentions)

Constructing the clusters



Vertices = noun phrases
Edge weights = proportional to probability that two nouns are coreferent
Partitioning such a graph into disjoint clusters corresponds to performing coreference resolution on the noun phrases

How to incorporate information over sets of noun phrases?
Goal: partition graph into clusters with high intra-cluster weights and low inter-cluster edge weights:
- Based on strength or probability of detected relations
- Possibly augmented with heuristic constraints of forbidden merge of mentions

Often greedy clustering: each noun phrase mj is assigned to the same cluster as the closest preceding noun phrase m_j for which $P(y_{ij}|x_{ij}) > \varepsilon$ (threshold) (eg. $\varepsilon = 0.5$)
Best clustering: intractable problem, but in this case: column generation

Inference with integer linear programming as postprocessing step
Linear programming (LP) = collection of techniques and methods for finding an assignment for a set of variables so that the linear objective function composed of these variables is optimal, ie, by maximizing or minimizing the linear function, while subject to linear equality or inequality constraints
Integer linear programming (ILP) = variant of LP, variables are restricted to only have integer values

LP problem example: maximizing the profit of manufacturing the number of men's pants (x1) and women's pants (x2). Polytope is 2 dimensional polygon, this area gives all valid combinations .

ILP: the optimization problem is computed by using a solver:
- The simplex algorithm and its variants find an optimal solution by moving along the edges of the n-dimensional polytope defined by the constraints (ie the convex hull) until no better value is found, because the objective function is linear, a local optimum is also a global optimum
- Interior point algorithms, which move inside the polytope
- Off-the-shelf solver eg. Gurobi.com
-

Popular methode eg for computing the joint prediction of a global model given the probabilistic outcomes of local models and possibly imposing additional constraints = form of reasoning/inference with the outputs of local classifiers
- Here illustrated with finding a global solution for the coreferents in a discourse as a post-processing step, when finding entity clusters
- In exercises: additional constraints for this problem are studied
- ILP can be used for any inference based on constraint rules in NLP

Example:
- Add binary constraints on each of the variable yij ∈ {0,1}
- Global optimization of the pairwise classification decisions by integer linear programming

$$\max_{y_{ij}} \sum_{x_{ij}} \log P(y_{ij}|x_{ij}) \cdot y_{ij} - \log\left(1 - P(y_{ij}|x_{ij})\right) \cdot (1 - y_{ij})$$

$P(y_{ij}|x_{ij})$: computed with a classification model (see above)

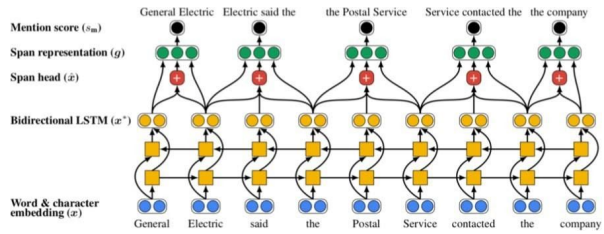
- Add constraints over each triplet of mentions to enforce transitivity:

$$(1 - y_{ij}) + (1 - y_{jk}) \geq (1 - y_{ik})$$

ensures that whenever $y_{ij} = y_{jk} = 1$ also $y_{ik} = 1$

- Solution for short text because of computational complexity of ILP.

Neural end-to-end coreference resolution
Jointly learns which spans are entity mentions and how to best cluster them
Does not use external resources such as a syntactic parser or hand- engineered mention detector, but uses a neural head-finding attention mechanism
To reduce computational complexity the model allows aggressive pruning of potential mentions



STEP 1: compute embedding representations of spans for scoring potential entity mentions. Prune low scoring spans, so that only a manageable number of spans is considered for coreference decision. In general the, the model considers all possible spans up to a maximum width, but we depict here only a small subset.
STEP 2: antecedent scores are computed from pairs of span representations, the final coreference score of a pair of spans is computed by summing the mention scores of both spans and their pairwise antecedent score.

$$s_y(i) = w_y \cdot FFNN(g_n)$$

$$s_j(i) = w_j \cdot FFNN \sim g_n, g_U, g_n{}^\circ g_U, \varphi(i, j)\ddot{A}$$

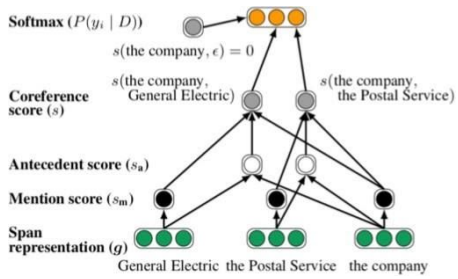
Where $\varphi(i, j)$ = feature vector that may contain additional information (eg. speaker and genre information, distance between two spans)

Objective: optimize the marginal log-likelihood of all.

Correct antecedents implied by the gold clustering:

$$\log \prod_{i=1}^{N} \sum_{\hat{y} \in y(i) \cap \mathrm{GOLD}(i)} P(\hat{y})$$

GOLD(i) = set of spans in the gold cluster containing span i. If span i does not belong to a gold cluster of all gold antecedents have been pruned, GOLD(i) = { ε }



Evaluation

MUC = link based metric

Recall = Çl ÉÇÑÑmÉf Ö[Öfmy ÜmámÑ}fmà knácÖ # / Çl ÉÇÑÑmÉf knácÖ

Precision = " Çl ÉÇÑÑmÉf Ö[Öfmy ÜmámÑ}fmà knácÖ / # Çl Ö[Öfmy ÜmámÑ}fmà knácÖ

Links cannot represent singleton entities, ie. Entities, which are mentioned only once

B³ : Instead of looking at the links, B^3 computes precision and recall for each mention in the document:

$$B^3 - \text{precision}(m_i) = \frac{|S_{m_i} \cap G_{m_i}|}{|S_{m_i}|} \qquad B^3 - \text{recall}(m_i) = \frac{|S_{m_i} \cap G_{m_i}|}{|G_{m_i}|}$$

S_{mi} = system generated cluster to which m_i belongs
G_{mi} = gold/ground truth cluster to which m_i belongs
B^3 – recision/recall values are averaged over all mentions

Constrained Entity-Aligned F-Measure (CEAF):
The first step in the CEAF computation is getting the best scoring alignment called $\Phi(r^*)$ between the key (ground-truth) and restored (system generated) entities, where alignment for an entity pair $(K_{\tilde{a}}, R_U)$ can be computed as:

$$\phi_4(K_i, R_j) = \frac{2|K_i \cap R_j|}{|K_i| + |R_j|}$$

$\varphi_{\tilde{a}}(K_{\tilde{a}}, R_U)$: summed over the involved aligned entities to obtain $\Phi(r^*)$

Computed at mention level: based on $\Phi(r^*)$

- $CEAF_m$-precision = number of aligned mentions divided by the number of response mentions
- $CEAF_m$-recall = number of aligned mentions divided by the number of key mentions

Computed at the entity level: using the best mapping $\Phi(r^*)$:

- CEAF -precision = ç(Ñ
- CEAF -recall = ç(Ñ

N_r and N_k are the number of response and key entities, respectively

F-measure based on the above precision and recall values
Average F-measure of MUC-F, B^3-F, CEAF-F to obtain a more complete evaluation

Coreference resolution is governed by syntactic, semantic, and discourse constraints:

- Constraints resolve ambiguity, but not completely
- Highly contingent on lexical meaning and world knowledge: eg. a "president" can be a "leader" but cannot be an "increase"
- Related: lack of knowledge of semantic compatibility: present a generative approach that jointly learns to resolve coreference and to recognize coarse-grained entity types (names entity recognition) for each mention. Word embeddings could be improved: use word embeddings helps, but also hurts
- Correct segmentation in mentions could largely improve the result (use of gold mentions improved F1 score by 17,5% in [Lee et al, EMNLP 2017])

<u>Recognition of rhetorical discourse structure</u>

Introduction to rhetorical structure theory (RST) and its recognition

RST assumes:

- Text to have a hierarchical organization based on asymmetrical nucleus-satellite relationships
- Pairs of elementary textual units combine into parent units – text spans – which are. Again recursively merged until at a certain point a unit spanning the entire text – the root – is reached
- Constituent halves of any text span are linked together by a text structuring. Relation, which typically holds between a more central unit – the nucleus – and a more peripheral one – the satellite (although there also is a small set of multinuclear relations)

RST relations:

- Elaboration: the satellite gives further information about the content of the nucleus
  [$_S$ In order to have some more space $_S$] [$_N$ we will have to rent another building $_N$]
  Attribution: the satellite gives the source of attribution for an instance of reported speech in the nucleus, eg.

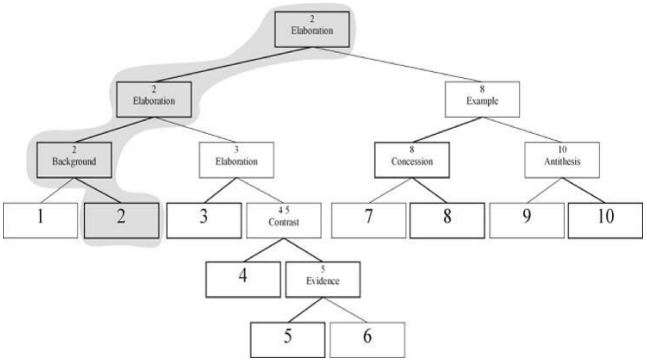
[$_S$ Analysts estimated $_S$] [$_N$ that sales at US stores declines in the quarter, too]

- Contrast: multinuclear. Relation, in which to or more nuclei contrast along some important dimension
  [$_N$ I don't want a truck], [$_N$ I'd prefer a convertible.]
- List: multinuclear relation, a series of nuclei is given, without contrast of explicit comparison
  [$_N$ John is a quiz master;] [$_N$ Ellen is the contact point.]

Set of discourse relations: the following relations have been agreed upon for most West European languages

Nucleus-Satellite Relations		
Evidence	Justify	Antithesis
Concession	Circumstance	Solutionhood
Elaboration	Background	Enablement
Motivation	Volitional Cause	Non-volitional Cause
Volitional Result	Non-volitional Result	Purpose
Condition	Otherwise	Interpretation
Evaluation	Restatement	Summary
Multi-nuclear relations		
Sequence	Contrast	Joint

A discourse tree for the scientific American text



Discourse parsing:

- Segmentation of discourse fragments, eg on a clause or sentence level: use of syntactic parser and additional segmentation rules
- Classification of segments or their relationships: often entails structured learning: often feature rich supervised classification
- Best results with dependency parse model
- Many unsolved research problems

Argumentation Mining

Argumentation mining or argument mining deals with the identification of argumentative structures within language. It involves among others the automatic detection of:

- Arguments = argumentative language
- Argument components
- Classification of relations between arguments

Argument(ation) mining has been defined as "the general task of analyzing discourse on the pragmatics level and applying a certain argumentation theory to model and automatically analyze the data at hand". Two stages are crucial in argument mining:

1) Arguments' extraction: identification of arguments within the input natural language text. This step may be further split in two different stages such as the detection of argument components (eg. claim, evidence) and the identification of their textual boundaries. Many approached have recently been proposed to address this task, that adopt different methods like SVMs, Naïve Bayes classifiers, Logistic Regression
2) Relations' prediction: what are the relations (eg. attack and support) holding between the arguments identified in the first stage. Methods: standard SVMs, Textual Entailment. This stage is also in charge of predicting, in structured argumentation, the internal relations between the argument components, ie, the connection between the evidence and the claim.

There are many different argumentation schemes and their annotations eg:

- Recognition of textual entailment and contradictions
- Stance detection
- Recognition of claims and their premises
- Different types of propositions and their

structure Examples:

Reasoning about entailment

Entailment: detecting the asymmetric relationship between 2 sentences, eg:

Premise: A group of people standing in the snow with a. mountain in the background

Hypothesis: People are outside.

Entailment = classification into: entailment, contradiction or neutral relationship

Classical approaches:

- Classifiers employing hand engineered features derived from complex natural languages processing pipelines
- Simple bag-of-word-pairs classifiers using only lexical similarities

Today neural networks with attention mechanisms

Recognition of rhetorical structure evaluation

Deep learning models with word embeddings as input:

- Perform better than bag-of-word-pairs representations, or such representations enhanced with linguistic features
- Up to 83,5% accuracy on the test set of the Stanford Natural Language Inference corpus
- Some world knowledge is captured with the embeddings and by the attention mechanism that models the alignments eg that "snow" is related to "outside"

But training data are needed.

Conclusion

- Discourse analysis is an. Important topic of research in NLP
- Both coreference resolution and rhetoric structure recognition can still be largely improved

Lecture 11: Extraction of Spatial and Temporal Information

Topics for the exam (might not be complete):

- Different models we can build to ouput a label structure
- Sequential labels in previous units
- Spacial labels in this unit (structured, linear classifier)
- Special; Temporal P21 nice example
- Use feature function (CRF)
- Constraints in the labels (classification)
- Give a very good look to the exercises. They are not easy.

Overview:

- Introductory concepts and annotation schemes
- Timex recognition and normalization
-

Temporal and spatial reasoning
- Spatial relation extraction
- Temporal relation extraction
- Direct prediction of spatial and temporal coordinates and reasoning in the geometric spaces

Introductory concepts and annotation schemes:

First of all, why are we doing this?

Spatial and temporal extractions or recognitions are key descriptors of content in:

- Language understanding
- Search
- Visualization
- Mining

Spatial Relation Labeling:

- This involves including argument objects, such as the trajectory landmark, and attributes of the spatial relation.

The ultimate goal of SRL is to predict the position of objects in a 2D or 3D space.

As we did in the chapter on Named Entity Recognition and Semantic Role Labeling, this task can be performed in the following two ways:

Sequence labeling:

- Given an input sentence X composed of a sequence of phrases x, assign a set of labels Y to the phrases and composing words, where both the phrase and the words receive labels y.
- Phrases are obtained by considering triplets composed of pairs of nouns and a preposition or verb that could act as a spatial indicator

Tree labeling:

- Given a constituent parse tree X, assign a set of labels Y where each constituent x receives labels y.

Annotation Schemes: spatial semantics

- SemEval 2012
- SemEval 2013
- SpaceEval 2015 (as part of SemEval 2015): ISOSpace in SemEval 2015: the spatial language specification language
- CLEF 2017: use of ISOSpace, also images as context
- Several small annotated corpora available.

Finally, back to SRL:

The goal is to jointly assign the labels of the ontology to a text item:

How do the annotation schemes look like?

SpaceEval 2015:

The following annotation categories are included in SpaceEval:

- Locations: regions, spatial objects, geographic and geopolitical places
- Entities participating in spatial relations
- Paths: routes, lines, turns, arcs
- Topological relations: in, connected.
- Direction and orientation: North, down
- Time and space measurements: 20 miles away, for two hours
- Object properties: intrinsic orientation, dimensionality.
- Frames of reference: absolute, intrinsic, relative
- Motion: tracking objects over time

Temporal Information Extraction:

Goal: Extract the events from a text, and position them on a time-line.

What sort of temporal relationships exist in language and are therefore of interest?

- Descriptions of: event position, start, end, duration
- Absolute or relative
- Definite or indefinite
- Explicit or implicit
- Background knowledge required
- They are usually underspecified

TimeML annotation:

- Event and timex spans:
- Event and timex attributes
- Timex normalization
- Temporal relations: between events and the document-creation time
- Temporal relations: among events and timex

Example:

-

TimeML annotation scheme:

Time and event expressions (The what)

- Segmenting and recognizing times and events
- Recognizing attributes:
  - Actual/hedged event? and is it negated (polarity)?
  - Time/date/duration/set time expression?

Timex normalization and Temporal relations (The When)

- Normalize time expressions (timex) to calendar values.
  - Temporally relate each event to the document-creation-time (DCT)
- Temporally relate events and times.

Temporal relations: Allen's interval algebra:

Time and event expression extraction:

This task can be viewed as a word tagging or sequence labeling task. This means that the concepts from Ch 2 and Ch 5/6 carry right over.

- Words are labeled as:
  - B(eginning)
  - I(nside)
  - O(utside)
- Words are restricted to POS classes of noun, adjective, adverb and number
- Use word classifier:
  - Classify each word independently, e.g., with logistic regression classifier, character-based neural network
  - Use context window to extract features
- Or sequence classifier
  - E.g., Conditional Random Fields (CRFs)

Example:

LSTM + CRF for event detection

Attribute classification:

Event tense, aspect, class, polarity, etc.:

- they had not yet <EVENT tense="PAST" aspect="PERFECTIVE" class="PERCEPTION" polarity="NEG" >seen</EVENT> a suit

Time type: date, duration or time:

- <TIMEX3 type="DATE>October</TIMEX3>
- <TIMEX3 type="DURATION">20 years</TIMEX3>
- <TIMEX3 type="TIME">noon Thursday</TIMEX3>

Typical approach:

- Multiclass or multitask classification given event or time.
- Similar features to timex recognition tasks

Time expression normalization

- Absolute temporal expressions
  - Map to calendar dates or times of the day
- Relative temporal expressions
  - Map to particular dates or times via a reference point
- Durations
  - Denote spans of time of varying granularity

Examples:

Normalized Timex Expressions:

Methods for normalisation:
The current normalization systems are rule based.
Eg. HeidelTime (https://heideltime.ifi.uni-heidelberg.de/heideltime/)

Temporal and Spatial Reasoning:

Spatial Relationship Extraction:
Potential Approaches:

- Pipeline of local classifiers
- Structured learning: global inference during prediction: e.g., spatial logic coded as constraints during prediction, e.g., with integer linear program
- Structured learning: global inference during training and prediction: e.g., spatial logic coded as feature functions, or as constraints during training (here illustrated), e.g., with integer linear program

And combinations of these.

Structured learning: global inference during training and Prediction:

- Joint training:
  - 1 classification model for the global structure
  - Output is = structure (here spatial ontology)

Model Input:

- Object to which the classification model is applied: e.g., sentence (in our case), paragraph, full document,
- Is usually composed of different input components: single words, phrases, ... depending on the type of text snippet to which a label will be assigned

Different input types have type related features and in the feature functions labels are restricted to certain types

- Hence we use input-output templates and their related blocks of weights $w_{p,}$ resulting in the following scoring function (L = set van labels, C_{yp} = set of candidate inputs/features for a particular label y_p)

Training:

Given N training examples:

The score of a training example with a correct labeling $Y^{(i)}$, should be larger or equal than the score of a training example of an incorrect labeling Y ∈ Y + some loss/cost :

A violation :

Training = finding the weights w that minimize these violations. Use hamming distances between the true and false labels.

To improve the efficiency of the training: training is done only with the most violated Y for each X.

- The target is to minimize the loss jointly: structured loss minimization.
- Training is done e.g., with a structured support vector machines (SSVM) or structured perceptron
- Training with the most violated constraints/outputs (Y) per training example
- Because spatial ontology defines a number of constraints when computing we can also
- add these during training when computing g(X, Y; w) with maximum score:
    - Using of efficient solver
    - These constraints can additionally be applied during prediction

Spatial relations as pairwise classification:

Output:

Structured learning:

PGMs, similar to the CRFs used for NER and SRL: assignment of label structure Ycap given input X with maximum probability:

Here we illustrate spatial relation learning with a structured learning with linear structured classifier: as is the case with a probabilistic graphical model, we learn the weights of each feature function, but use a linear scoring function g, with assignment of a label structure with maximum score:

g(X, Y; w) is a linear function in terms of the combined feature representations associated with each candidate input component x and an output label y

Spatial relation and role labeling:

-

Evidence that Relation extraction is challenging:

Example from a biomed task:

Temporal relation extraction:

The approach is similar to spatial info extraction, so it isn't detailed again.

Structured learning: inference during prediction:

- Structured learning: several labels are assigned jointly here by inference during prediction:
  - Local classifier for each label
  - Infer P(e1, e2, before) , etc. from rules given the (probabilistic) output of the classifiers, e.g., coded as integer linear program, where e refers to an event

Structured learning: inference during training and Prediction:

- One classifier assigns label structure
- Integrate (soft) rules during training:
  - E.g., rules as feature functions in a Markov random field classifier: iteratively update rule weights to match the labeled training data (Markov logic model)
  - Training: e.g., via Gibbs sampling

Temporal logic as feature functions:

Possibility to incorporate both local features and global transition rules between temporal relations

As any feature function: the weights w_i of the rules are learned during training based on their fit on the training data

Temporal relation extraction models:

And now my watch has ended.