# Summary Natural Language Processing 1

Phillip Lippe

November 26, 2019

## Contents

# 1 Morphology and finite state techniques

- Morphology concerns the **structure of words**
- *Morpheme*: minimal information carrying unit in a word. A word consists of morphemes
- *Affix*: Morphemes that only occur in conjunction with other morphemes
- *Stem*: a word is made up of a stem and zero or more affixes. Stems are therefore stand-alone morphemes
- There are different forms of affixes that describe when (prefix, suffix, infix, ...)
- An affix is productive if it applies in general and therefore also probably for new words
- **Inflectional morphology**
  - Fills predefined slots in paradigm, as plural, tense,... (create different grammatical forms, but word stays the same)
  - Fully productive, except irregular forms
  - Inflectional affixes are not combined in English
- **Derivational morphology**
  - Forming a new word through affix (also change of meaning possible)
  - May change POS tag
  - Examples include *anti-*, *re-*, *-ism*, *-ist* ("reset" vs. "set")
  - Generally semi-productive (applies for only subset of words in language)
  - Include *zero-derivation*: word that is both verb and noun, e.g. "text" vs. "(to) text (someone)"
- Ambiguities in terms of morphemes (single stems or affixes are ambiguous like "dog") or structure (combination of affixes/stem like "shorts" vs "short-s")
- **Bracketing**
  - Starting from the stem, find the combination of nearby affixes that still lead to a possible form
  - Example *un-ion-ise-ed*. Putting *un-* and *ion* together not possible as this forms a non-valid word (union would be different stem). $\Rightarrow$ *un-(ion-ise)-ed*
  - Next, adding the *-ed* ending is valid, and finally concatenating it with *un-*: *(un-((ion-ise)-ed))*

## 1.1 Applications of morphological processing in NLP

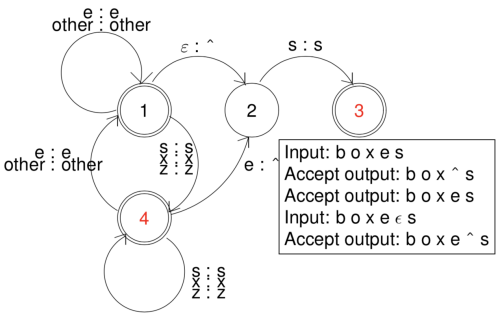
- We can use morphology to create a full-form lexicon (lexicon with each form of every word in it). However, this tends to explode very fast (high redundancy) and is not scalable for new words
- **Stemming**: use rules to get the stem form of a word. This allows us to match words to a small set of base words
- **Lemmatization**: Only finding split of stems and affixes. Is the preprocessing step before parsing (understanding the word!)
- Morphological process can either by analysis or generation
- Possible aspects/steps of morphological processing
  1. Surface/ground-word mapped to stem(s) and affixes. Either by declaring the affixes (*ping-ed*) or by explicitly saying which rule was applied (*ping PAST_VERB*)
  2. After knowing the affixes, analyze internal structure by bracketing
  3. Finally, understand syntactic and semantic effects where parsing can filter results of previous stages
- Overall, we need a lexicon combining three aspects:
  - affixes (with the associated information they carry)
  - irregular forms
  - stems (with syntactic categories)

### 1.2 Spelling rules

- English morphology is essentially concatenative
- English spelling rules can be described independently of the particular stems and affixes involved. It simply looks at the affix boundaries.
- Example spelling rule for e-insertion:

$$\epsilon \rightarrow \text{e} / \left\{ \begin{array}{c} \text{s} \\ \text{x} \\ \text{z} \end{array} \right\} \hat{}\_s$$

  Here, the formula is interpreted as “an empty string maps to e if an s,x or z is followed by an s of the next affix” (where e is inserted in the underscore space)
- Finite state machines (or transducers also creating corresponding output while parsing) can be used to implement spelling rules




- Each transition corresponds to a pair of characters.
- When the transducer is run in analysis mode, the system can detect an affix boundary (where we look up the stem and affix in the corresponding lexicon)
- In generation mode, we can just put in our parsed version and generate the correct spelling
- Morphology systems are usually implemented so that there is one FST per spelling rule and these operate in parallel
- However, FST are not applicable for internal structures as for example no bracketing model is possible
- A system which generates invalid output/accepts invalid derivations is said to **overgenerate**.

## 2 Language models and part-of-speech tagging

### 2.1 Probabilistic language modeling

- The Naive Bayes approach considers words as independent. However, we can also model word sequences using statistical techniques (based on context/semantic and syntax)
- **Corpus**: text collected for some purpose (i.e. movie reviews)
  - A corpus is *balanced* if it represents different genres (types of text/domain)
  - A *tagged* corpus has annotations regarding the POS tags (mostly POS tags are learned unsupervised, but with this data we enable supervised training methods)
- Use of language modeling
  - In speech recognition, it is hard to distinguish between words that sound similar. Thus, language modeling is used to rank the hypothesis of the recognizing system to make an estimation what phrase is most likely being said
  - Language modeling can also be used for word prediction, text entry, spelling correction, ...

### 2.1.1 N-gram models

- Modeling a sequence of $n$ words
- **Bigram** ($n = 2$):
  - use only previous word $\Rightarrow p(w_n|w_{n-1})$
  - Probability of a sequence can be expressed by $p(w_1, ...w_N) = \prod_{n=1}^{N} p(w_n|w_{n-1})$
  - Still, we assume that words with distance of more than 1 are independent
  - Estimating the probabilities by the maximum likelihood solution:

$$p(w_n|w_{n-1}) = \frac{c(w_n, w_{n-1})}{\sum_{w_k} c(w_k, w_{n-1})} = \frac{c(w_n, w_{n-1})}{c(w_{n-1})}$$

    Thus, we normalize the counts over the next word $w_{n-1}$
- **Trigram** ($n = 3$):
  - The probability of a word is based on the two previous words $\Rightarrow p(w_n|w_{n-1}, w_{n-2})$
  - Again, the probability of the sequence is $p(w_1, ...w_N) = \prod_{n=1}^{N} p(w_n|w_{n-1}, w_{n-2})$
- Problems with sparse data
  - **Smoothing**: for smoothing, we add a small extra probability for rare and unseen events to prevent probabilities of zero. E.g. for bigram:

$$p(w_n|w_{n-1}) = \frac{c(w_n, w_{n-1}) + \kappa}{c(w_{n-1}) + |V| \cdot \kappa}$$

    Simple to implement, but only suitable if having few unseen events (high $n$-gram have a lot)
  - **Backoff**: If we have good evidence of a long phrase, we use a high $n$-gram model (for example trigram). Otherwise, if phrase was not seen yet, we go to the next smaller model (here bigram) and checker its probability. If also not known, go deeper until you reach unigram.
  - **Interpolation**: combine the probability estimations of all models. For example we can use linear interpolation where we weight every model with a parameter:

$$p(w_n|w_{n-1}, w_{n-2}) = \lambda_1 p(w_n) + \lambda_2 p(w_n|w_{n-1}) + \lambda_3 p(w_n|w_{n-1}, w_{n-2})$$

    The parameters $\lambda_i$ need to sum up to 1 and are optimized on small held-out training subset.
  - **Unknown word tag**: using a unknown word tag which is also used in the training set. Replace all unknown words in the (test) text by this tag
- Another limitation of $n$-gram models are long-term dependencies as these cannot be captured efficiently
- *Evaluation* of $n$-gram models
  - **Intrinsic evaluation**: evaluate directly on test set designed for the task with a metric
    - A suitable metric is for example *perplexity* which is the inverse probability of the test dataset normalized by number of words $N$:

$$PP(W) = (p(w_1, ..., w_N))^{-1/N}$$

    - For bigram, this would be $PP(W) = \left(\prod_{n=1}^{N} p(w_n|w_{n-1})\right)^{-1/N}$
    - The goal is to minimize perplexity (lower perplexity indicates better model)
    - However, perplexity strongly relies on the similarity of training and test dataset and is therefore not comparable across different datasets
  - **Extrinsic evaluation**: evaluation in the context of external task, i.e. speech recognition or word prediction
    - Better, but very time consuming
    - Hybrid approaches compare own models by perplexity, and apply the best model in extrinsic environment (external task)

## 2.2 Part-of-speech tagging

- Tag every word by what king of speech it is (verb, noun, ... ⇒ ambiguity)
- The tags are taken from a tagset which uses standardized codes for fine-grained POS
- *Benefits* of POS tagging
  - First step towards syntactic analysis (is very fast, but simpler than full syntax parsing)
  - POS tags can be useful features for application
- Problem of ambiguity: most high-frequency words have more than one POS tag. Language with rich morphology (significant affixes) tend to have less as the distinguish affixes better

### 2.2.1 Tagging strategies

- Simplest strategy: assign to each words its most common tag (also called unigram tagging). Already gives a strong baseline
- **Hidden Marcov models**
  1. Start with untagged text
  2. Assign to the words all their possible POS tags
  3. Find the most probable sequences of tags given sequences of words

$$\hat{t}^n = \arg\max_{t^n} p(t^n|w^n) = \arg\max_{t^n} p(w^n|t^n) \cdot p(t^n)$$

- If we apply for example bigram in this model, we get:

$$p(t^n) \approx \prod_{i=1}^{n} p(t_i|t_{i-1})$$

$$p(w^n|t^n) \approx \prod_{i=1}^{n} p(w_i|t_i)$$

$$\hat{t}^n = \arg\max_{t^n} \prod_{i=1}^{n} p(w_i|t_i)p(t_i|t_{i-1})$$

- Actual systems use trigrams. Smoothing and backoff are important (fewer unknown open class words)
- Evaluation by percentage of correct tags (but using most common tag already gives 90% accuracy. With trigram about 97%)
- Common errors
  - Difference between country “Turkey” and bird “turkey” (it decides based on whether an *a* is in front of turkey or not)
  - Because of smoothing, we can get for the phrase “have hope” that both words are verbs although hope has no past tense which is antigrammatical!

# 3 Formal grammars and syntactic parsing

- Syntax: structure of sentence, parsing syntax to get (long-distance) dependencies of words

## 3.1 Generative grammar

- Formally specified grammar that can generate all and only acceptable sentences of a natural language
- A phrase can be bracketed into its internal structure: *((the (big dog)) slept)*
- Each subpart is a **constituent**: group of words/phrase behaving as a single unit
- Labels can be assigned to the internal structures (for instance, *the big dog* is a noun phrase)

### 3.1.1 Phrases and substitutability

- Words with the same POS tag can be replaced
- Phrasal categories indicate which phrases can be substituted
- Example phrasal categories include noun phrase (NP), verb phrase (VP), propositional phrase (PP), ...
- Goal: capture substitutability at phrase level by phrasal categories

## 3.2 Context-Free Grammars

- Defining a grammar on rules of production, and basic lexicon
- Basic elements of a context-free grammar (CFG):
  1. Set of non-terminal symbols (e.g. S, VP)
  2. Set of terminal symbols (i.e., the words)
  3. Set of rules, where left-hand side is single non-terminal symbol, and right side combination of non-terminal and terminal. Examples:
     `S -> NP VP`
     `V -> fish`
  4. A start symbol (here `S`) which is a non-terminal
- Exclude empty productions, like `NP->`$\epsilon$
- For rules of non-terminal to single word, the non-terminal represents the POS tag of this word
- A context free grammar can be used for either generating sentences (start with `S`, and choose rules), or for analyzing/assigning a structure to a given sentence
- For analyzing, the bracketed notation or a parse tree is often used to represent the structure:
  `(S (NP` *they*`) (VP (V` *fish*`)))` (for parse tree, `S` would be root node and `NP` and `VP` its children a.s.o.)
- However, the grammar is not always unique
  - **Lexical ambiguity**: a word is more than once in the lexicon having different POS tag
  - **Structural ambiguity**: multiple possible analysis because of multiple rules for same non-terminals

## 3.3 Chart parsing with CFGs

- Increase efficiency by recording all possible rules we could apply on a sentence
- The **chart** is a record of all substructures that have ever been built during the parsing / stores partial results of parsing in a vector
- An **edge** is a data structure that represents a rule application ,which includes:
  - An id for referring to it
  - The outer left and right node in the sentence of the phrase on which the rule is applied
  - The *mother* symbol (non-terminal which is on the left side of the rule)
  - The *daughters* which are the symbols on the right side of the rule (words and/or non-terminals produced by previous edges and referred to by the id)
- In conclusion, a full chart for the sentence `they can fish` look like that:

| id | left | right | mother | daugthers |
|---|---|---|---|---|
| 1 | 0 | 1 | NP | (they) |
| 2 | 1 | 2 | V | (can) |
| 3 | 1 | 2 | VP | (2) |
| 4 | 0 | 2 | S | (1 3) |
| 5 | 2 | 3 | V | (fish) |
| 6 | 2 | 3 | VP | (5) |
| 7 | 1 | 3 | VP | (2 6) |
| 8 | 0 | 3 | S | (1 7) |
| 9 | 2 | 3 | NP | (fish) |
| 10 | 1 | 3 | VP | (2 9) |
| 11 | 0 | 3 | S | (1 10) |

where the sentence is structured as $._0$`they`$._1$ `can`$._2$ `fish`$._3$ and rows of the chart are edges. The parsing is visualized in the figure below

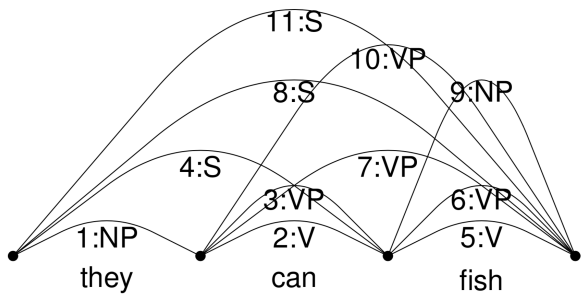



Figure 1: Example for chart parsing. The resulting chart data structure is shown below.

#### 3.3.1 Implementation of bottom-up parser

- A bottom-up parser would start from the left, look at the first two connections points (first word) and check for a rule that can be applied to this word
- If a rule has been found, it is added as edge into the chart, and we start looking for rules that can be applied to the new edge (recursion!). Once no more rule can be applied on this edge and the recursion stops, we go back to the word and continue our search for applicable rules on this word
- After every rule was applied, the parser moves on to the next word on the right and check for rules that can be applied to this word, **and** all other words/edges that have been processed beforehand.
- Only if no more rules can be applied, the parser moves on to the next word, until all words are processed
- The correct parse/grammar structure is this one that end with the start symbol `S` from the first to the last node of the sentence
- Important sub-technique: **Packing**
  - Due to multiple rules with same input, we can have two identical edges that are just based on different daughters
  - Every following rule is then applied on both edges which is very inefficient
  - Thus, with *packing* we change the daughter entries to a list of possible daughter lists
  - For example, the edge 7 and 10 from the previous example can be combined:

| id | left | right | mother | daugthers |
|---|---|---|---|---|
| 7 | 1 | 3 | `VP` | {(2 6), (2 9)} |

  - If a new daughter list is added, no new recursion/rule application needs to be done

### 3.4 Probabilistic parsing

- For a single sentence with 20 or more words, we will get over 1000 analysis $\Rightarrow$ how do we determine the best/most probable analysis?
- The traditional approach is it to grammar rules handwritten but they tend to often fail when parsing new sentences
- Current approaches: probabilistic CFG (PCFG) where every rule is augmented with a probability
- The probability of a parse tree is the product of the probabilities of all the grammar rules that are used in the sentence derivation
- Probabilistic CFGs help for *disambiguation* as we can rank all analysis by probability and just pick the best (n) one(s)
- Probabilities can also be used to speed up parsing (drop trees/substructures during parsing that already have a very low probability compared to other current substructures)

| | |
|---|---|
| $S \rightarrow NP\ VP$ | .8 |
| $S \rightarrow VP$ | .2 |
| $NP \rightarrow D\ N$ | .4 |
| $NP \rightarrow NP\ PP$ | .4 |
| $NP \rightarrow PN$ | .2 |
| $VP \rightarrow V\ NP$ | .7 |
| $VP \rightarrow VP\ PP$ | .3 |
| $PP \rightarrow P\ NP$ | 1 |

| | |
|---|---|
| $D \rightarrow the$ | .8 |
| $D \rightarrow a$ | .2 |
| $N \rightarrow flight$ | 1 |
| $PN \rightarrow john$ | .9 |
| $PN \rightarrow schiphol$ | .1 |
| $V \rightarrow booked$ | 1 |
| $P \rightarrow from$ | 1 |

Figure 2: Probabilistic CFGs. The probabilities are normalized over all rules with same *left* side.

#### 3.4.1 Treebank PCFGs

- Instead of specifying/tuning the grammar and its corresponding probabilities by our own, we can use a large dataset of sentences with annotated parse trees
- This way, we implicitly get a grammar and the probabilities of each rule
- A **treebank** is therefore a collection of sentences annotated with constituent trees
- To estimate the rule probabilities, we use the maximum likelihood:

$$p(X \rightarrow \alpha) = \frac{C(X \rightarrow \alpha)}{C(X)}$$

where $C(X \rightarrow \alpha)$ number of times the rule is used in corpus, and $C(X)$ the number of times the non-terminal symbol $X$ appears in treebank

#### 3.4.2 Why CFG and not finite state machines

- Language often has centre-embeddings like $A \rightarrow \alpha A \beta$ which cannot be captures by FSAs
- However, humans limit the application of such centre-embeddings so that we can convert those into finite rules
- The advantage of a FSA would be that we can model hierarchical structures (supported by the fact that we understand the semantic of a sentence, we need good internal structures like the hierarchy)

### 3.5 Dependency structures

- Context free grammars were based on phrase-structures in sentences
- Another possible representation of parsing sentences is using directed/asymmetric binary grammatical relations that hold among the words
- A relation consists of
  - a head `H` (central word)
  - a dependent `D`
  - a label identifying the relation between `H` and `D`

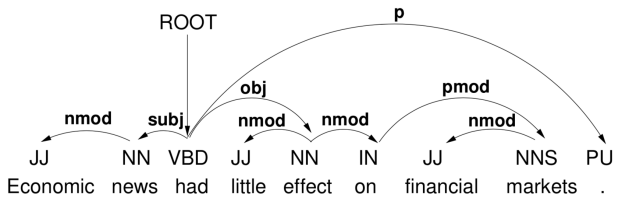



Figure 3: Dependency parsing. All relations are directed form head `H` to dependent `D`.

- It is important to prevent parsing errors as these can significantly change the semantic of a sentence

## 4 Lexical and distributional semantics

- **Compositional semantics**: meaning of phrase/sentence
- **Lexical semantics**: meaning of individual words

## 4.1 Approaches for lexical meaning

- How to represent a meaning of a word. Problem: no representation can fully capture language yet!

### 4.1.1 Formal semantics

- Based on set theory, describing the features of a word
- Meaning postulates: $\forall x \left[\text{bachelor}'(x) \rightarrow \text{man}'(x) \wedge \text{unmarried}'(x)\right]$
  - If a word is in the set *bachelor*, then it also is in *man* and *unmarried*
- Problems:
  - Limited, especially for special cases (i.e. what is the Pope?)
  - Very expensive for large corpus
  - For some words, it is almost impossible to find a good formalization
- Alternative: **Prototype theory**
  - Notion of graded semantic categories with no clear boundaries
  - No requirement that a feature must be shared by all members
  - Certain members are more central or prototypical $\Rightarrow$ **Protoypes**
  - New members are added based on similarity to prototypes
  - Features and category memberships are graded

### 4.1.2 Semantic relations

- **Hyponymy**: IS-A relation, forms a taxonomy (*Example*: dog is a hyponym of animal)
  - Easier to construct for certain nouns, but especially hard for adjectives
- **Meronomy**: PART-OF relation (*Example*: arm is a meronym of body)
- **Synonymy**: Words can be exchanged without changing the meaning of a sentence/phrase
- **Antonymy**: Opposite meanings (*Example*: big vs. little)

### 4.1.3 WordNet

- Large-scale corpus for English resource
- Handconstructed
- Organized in synsets: sets of synonyms
- Synsets are connected by semantic relations
- Similarity of words is the similarity of synsets

## 4.2 Polysemy and word sense disambiguation

- A word can mean different things based on the sentence/context it is used in
- Meaning of words is not fixed, but dynamically adapted by the context
- **Regular polysemy**: mechanisms to apply on words to fit into context
  - *Zero-derivation*: verb $\leftrightarrow$ noun without changing word. Example: "*tango*"
  - *Metaphorical*: using words from a different domain to express similar meaning. Example: "*swallow information*"
  - *Metonymy*: use an entity to actually refer to other aspects of it. Example: "*drinking his glass*"
- **Word sense disambiguation**: derive meaning of word in context
  - *Supervised* (most common) $\rightarrow$ predefined list of senses (i.e. WordNet), and train model on *large* corpus. Problem: we have to learn a new classifier for every word!

- *Semi-supervised* → annotate small dataset, bootstrap from there. Might be helpful as some instances have no single/discrete meaning.
- *Unsupervised* → induce sense by clustering of word occurences. Usually, the result is either too fine-grained or coarse-grained for most word (only work great for frequent words)

#### 4.2.1 Semi-Supervised WSD: Yarowsky algorithm

- Using bootstrapping which needs only a very small hand-labeled training set
- Still, learns a classifier for every word ⇒ no generalization!
- Also, define features (see notion of context) for every word to determine its meaning
- **Algorithm iteration**:
  0. Given a small initial seed set $\Lambda_0$ of labeled instances of each sense, and a much larger unlabeled corpus $V_0$
  1. Train classifier on $\Lambda_0$
  2. Use trained classifier to label $V_0$
  3. Select the examples in $V_0$ that the classifier is most confident on
     (a) Reliability of a prediction defined as $\log\left(\frac{p(a|w)}{p(b|w)}\right)$ for word w and possible senses $a$ and $b$
     (b) Rank reliabilities of all predictions and choose $n$ best
  4. Remove chosen examples from $V_0 \Rightarrow V_1$, and add them to the training set $\Rightarrow \Lambda_1$
  5. Repeat step 2) to 4) until:
     (a) Either $V_i$ is empty
     (b) Or the error rate on the training/validation is sufficient low
- Reported accuracy of 95%, but on easy homonymous examples
- **One sense per discourse**
  - Original algorithm uses *one sense per discourse* as second heuristic
  - If a word appears twice or more often in the same text, they are probably of the same meaning ⇒ Annotate these as well and use them as additional training examples

### 4.3 Distributional semantics

- Probabilistic models for semantics
- Distributional hypothesis about word meaning: the meaning of a word is determined by its context ⇒ similar meanings have similar contexts
- Thus, distributions are a good conceptual representation if you believe that 'the meaning of a word is given by its usage' ⇒ Corpus-dependent like different culture, domains, ...
- Distributions can encode lexical- and world knowledge, but mostly only partial lexical semantics
- Techniques: Count-based models and prediction models

#### 4.3.1 Count-based models

- Vector spaced models in the semantic space, where every dimension corresponds to a possible context ⇒ features
- Distribution can be seen as point in space
- As a result, we get a feature matrix:

| | $\text{feature}_1$ | $\text{feature}_2$ | $\dots$ | $\text{feature}_n$ |
|---|---|---|---|---|
| $\text{word}_1$ | $f_{1,1}$ | $f_{2,1}$ | $\dots$ | $f_{n,1}$ |
| $\text{word}_2$ | $f_{1,2}$ | $f_{2,2}$ | $\dots$ | $f_{n,2}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $\text{word}_m$ | $f_{1,m}$ | $f_{2,m}$ | $\dots$ | $f_{n,m}$ |

- Possible design choices in count-based models:

1. **Notion of context**: how to define the context of a word
   (a) *Word windows*: n words on either side of the lexical item, and count occurrences of words
   (b) *Filtered word windows*: n words, but remove irrelevant words based on POS-tag or stop-list (don't need to extend window)
   (c) *Lexeme windowing*: word windows (filtered or unfiltered), but with using stemming (mostly lead to more robust models)
   (d) *(Syntactic) dependencies*: context with dependency structure it belongs to (directed link between heads and dependents). Can be used with different extends
   Example: "The prime minister acknowledged the question"
   - [prime_a 1, acknowledge_v 1] (a for adjectives, v for verbs)
   - [prime_a_mod 1, acknowledge_v_subj 1] (mod for modifiers, subj for verb in relation to subject)
   $\Rightarrow$ Problem: complex context lead to sparse vectors

   $\Rightarrow$ Working best: small window sizes or short dependencies

2. **Context weighting**: how to set the weights in the vector
   (a) *Binary model*: if $c$ co-occurs with word $w$, value of entry is 1, else 0
   (b) *Basic frequency model*: number of times $c$ co-occurs (probably normalized)
   (c) *Characteristic model*: weights express how characteristic a given context is for a word $w$
   (d) *Pointwise Mutual Information (PMI)*: example of characteristic model. Comparing probability of both words occur together compared to occurring alone.

$$PMI(w,c) = \log \frac{P(w,c)}{P(w)P(c)} = \log \frac{P(c|w)}{P(c)} \qquad \text{where} \qquad P(c) = \frac{f(c)}{\sum_k f(c_k)}, P(c|w) = \frac{f(w,c)}{f(w)}$$

$$\Rightarrow PMI(w,c) = \log \frac{f(w,c)\sum_k f(c_k)}{f(w)f(c)}$$

   $PPMI \rightarrow$ only use positive values, $PPMI(w,c) = \max(PMI(w,c), 0)$

3. **Semantic space**: what are possible contexts
   (a) *Entire vocabulary*: every word represents a possible context.
   + All info included (also the rare one) - Inefficient (large space & sparse), noisy
   (b) *Top n words with highest frequency*:
   + More efficient, noise is filtered out - May miss out infrequent contexts
   (c) *Singular Value Decomposition*: dimension reduction by exploiting redundancies
   + Very efficient, good generalization - Not interpretable (or very hard)
   (d) *Non-negative matrix factorization*: Similar to SVB, but performs factorization differently

#### 4.3.2 Prediction-based models

- Train a model to predict plausible contexts for a word
- Learn word representations in the training process
- **Short dense** embeddings with **latent** dimensions
  - Easier to use as features with machine learning
  - Better generalization than simple counting $\Rightarrow$ capturing more complex relations like synonym
- One example for prediction-based models is skip-gram, also known as word2vec (see later section)

#### 4.3.3 Similarity

- Definition of similarity very broad. Can include synonym, antonyms, hyponyms, ...
- Measuring similarity with **Cosine** between vectors $v$ and $u$:

$$\cos(\theta) = \frac{\sum_k v_k \cdot u_k}{\sqrt{\sum_k v_k^2} \cdot \sqrt{\sum_k u_k^2}}$$

- Cosine measure calculates the angle between $v$ and $u$, and is length-independent (normalization). Important as frequent words can have longer vectors
- Other measures include Jaccard, Euclidean distance (vectors need to be normalized!), ...
- However, true-synonyms do not always get higher similarity scores than near-synonyms, and also to antonyms
- Identifying antonyms by extra heuristics like checking words with high similarity that frequently appear together (*example*: "we serve hot and cold drinks")

#### 4.3.4 Distributional word clustering

- Cluster words based on the contexts they occur
- Predefine number of clusters and corpus (for instance 2000 nouns in 200 clusters)
- **Features** can represent different kinds of contexts
  - Windows based context, parsed or unparsed, syntactic dependencies
  - Define notion of context, context weighting, and semantic space
  - Feature representation can significantly influence performance
- Clustering algorithm: K-means
  - Given dataset with $N$ points and task of $K$ clusters, minimize sum of squares of distance of each data point to its closest cluster mean $\boldsymbol{\mu}_i$:

$$\arg\min_C \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in C_i} ||\boldsymbol{x} - \boldsymbol{\mu}_i||^2$$

- Small context sizes (small windows, syntactic dependencies) lead to clusters of synonyms (words that can be replaced)
- Large context sizes lead to **topical similarity** (words belonging to the same topic)

#### 4.3.5 Skip-gram

- Sometimes referred to as *word2vec* because it is implemented in this package
- Given a word $w_t$, predict neighbouring words in a context windows of $2L$ words (for $L = 2$: $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$)
- In skip-gram, we learn two representations for every word $w_j \in V$:
  - **word embedding** $v$ in word matrix $W$
  - **context embedding** $c$ in context matrix $C$ (word in the role as context for other words)
- To learn these embeddings, we take every word $w(t)$ in the corpus (index $j$ in vocabulary), and try to predict $w(t+1), ...$ where we denote this word with index $k$ in the vocabulary:

$$p(w_k|w_j)$$

- The idea in skip-gram is that we compute this probability by the similarity between the words $w_k$ and $w_j$ whereas we use the context matrix $C$ for $w_k$ and the word matrix $W$ for $w_j$ (see figure below)
- Similar to the cosine similarity, we use the dot product for calculating this:

$$\text{Similarity}(c_k, v_j) \propto c_k \cdot v_j$$

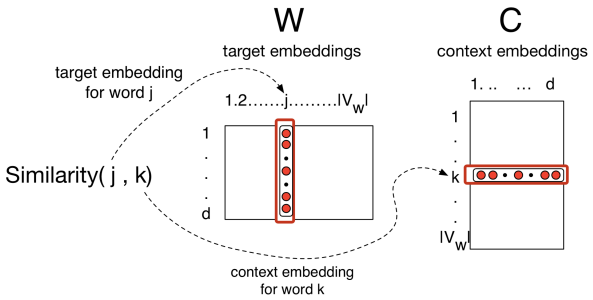



Figure 4: Skip gram overview

- To normalize and get probability distribution over contexts, we use the softmax function:

$$p\left(w_k|w_j\right) = \frac{\exp\left(c_k \cdot v_j\right)}{\sum_{i \in V} \exp\left(c_i \cdot v_j\right)}$$

- For the learning process, we start with randomly initialized vectors, and try to maximize the log-likelihood of the dataset (by performing SGD or similar):

$$\arg\max \sum_{(w_j,w_k)\in D} \log p\left(w_k|w_j\right) = \sum_{(w_j,w_k)\in D} \left(c_k \cdot v_j - \log \sum_{c_i \in V} \exp\left(c_i \cdot v_j\right)\right)$$

- We can also represent skip-gram as a (neural) network:

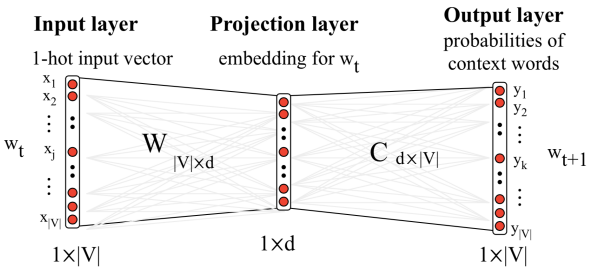



Figure 5: Skip gram as neural network. The weights of the first layer represent the word embedding $W$, whereas the context embedding $C$ is in the second layer.

- However, the problem here is that for a large vocabulary size, the denominator of the softmax is very expensive to calculate $\Rightarrow$ **negative sampling**
  - Approximate denominator by sampling $k$ random words from vocabulary (probability of sampling for a word is mostly connected to its unigram probability/frequency in the corpus like $\text{count}^{\alpha}(w)$ with for example $\alpha = 0.75$)
  - Dataset consists therefore out of words pair which are either positive or negative examples for context + word (note that we do not distinguish between the probability calculation of $w_{t-2}$ and $w_{t-1}$ for example)
  - We convert the classification task into predicting whether a context pair is a positive or negative example from the corpus

$$p\left(+|w_j, w_k\right) = \sigma(c_k \cdot v_j) = \frac{1}{1 + \exp(-c_k \cdot v_j)}$$

$$p\left(-|w_j, w_k\right) = 1 - p\left(+|w_j, w_k\right) = \frac{1}{1 + \exp(c_k \cdot v_j)}$$

$$\Rightarrow \arg\max \sum_{(w_j,w_k)\in D_+} \log p\left(+|w_k, w_j\right) + \sum_{(w_j,w_k)\in D_-} \log p\left(-|w_k, w_j\right)$$

- Embeddings capture **analogies**: *a* is to *b* as *c* is to *d*

- Due to similarity, we can use the offsets to find the appropriate word $d$:

$$a - b \approx c - d \Rightarrow d' = \arg\max_{d'_w \in V} \cos(a - b, c - d')$$

- Word2vec is often used as initialization/pretraining for other tasks. Reasons:
  - Will help the model to start from an informed position
  - Only needs a plain text corpus without any annotation
  - Is very fast and pretrained versions are also available on the internet
  - Best performance can be achieved by fine-tuning the weights afterwards

# 5 Compositional semantics and discourse processing

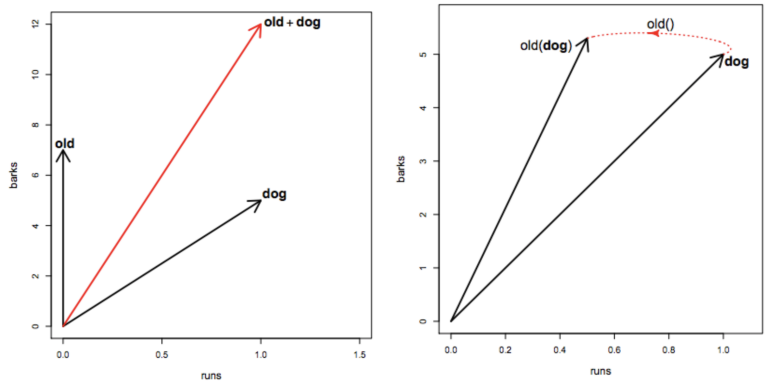
- **Principle of Compositionality**: meaning of whole phrase derivable from meaning of its parts
- Sentence structure conveys some meaning as well
  - Different syntactic structures may have the same meaning, but similar syntactic structures can also have different meanings
- Not all phrases are interpreted compositionally (e.g. *kick the bucket*) but can be grouped together and viewed as one element
- Meaning of a single word can depend on the composition (*fast* programmer vs. *fast* plane, metaphors,...)

## 5.1 Compositional distributional semantics

- Extending distributional semantics to phrases/sentences
- Unsupervised model ⇒ general-purpose representations
- Model composition in vector space. However, if we would model every sentence as independent, we would get an infinite dimensional space

### 5.1.1 Vector mixture model

- Combining the vectors of all words in the sentence
- Mostly done either additive (adding all vector) or multiplicative (elementwise product of vectors)
- Problem: does not consider word order and is therefore suitable for modelling content words (nouns, verbs, adjectives,...), but not for function words that require syntactic dependencies (pronouns, ...)
- Is often used as baseline




(a) Vector mixture model (b) Lexical function model

Figure 6: Compositional distributional semantics

### 5.1.2 Lexical function model

- Discriminate between words that meaning is determined by its context/distribution (e.g. nouns), and function words that are applied on the represented words as **lexical functions**
- Example: $\underbrace{old}_{\text{functional}}$ $\underbrace{dog}_{\text{distributional}}$ $\Rightarrow$ apply function of *old* on *dog*
- Lexical functions are parameter matrices (i.e. $\boldsymbol{A}_{old}$) which are multiplied with the vector representation of nouns

| **OLD** | runs | barks |
|---|---|---|
| runs | 0.5 | 0 |
| barks | 0.3 | 1 |

$\times$

| | **dog** |
|---|---|
| runs | 1 |
| barks | 5 |

$=$

| | **OLD(dog)** |
|---|---|
| runs | $(0.5 \times 1) + (0 \times 5) = 0.5$ |
| barks | $(0.3 \times 1) + (5 \times 1) = 5.3$ |

- Adjectives that does not change the meaning of a word are diagonal up to identity matrix $\Rightarrow$ element captures how features interact with each other given the adjective
- The matrices are learned by comparing the representation of plain nouns to combination of noun and adjective. Pseudo-code algorithm:
  1. Obtain a distributional vector $n_j$ for each noun $n_j$ in the lexicon.
  2. Collect adjective noun pairs $(a_i, n_j)$ from the corpus.
  3. Obtain a distributional vector $p_{ij}$ of each pair $(a_i, n_j)$ from the same corpus using a conventional DSM.
  4. The set of tuples $\{(n_j, p_{ij})\}_j$ represents a dataset $\mathcal{D}(a_i)$ for the adjective $a_i$.
  5. Learn matrix $\boldsymbol{A}_i$ from $\mathcal{D}(a_i)$ using linear regression by minimizing:

$$L(\boldsymbol{A}_i) = \sum_{j \in \mathcal{D}(a_i)} ||p_{ij} - \boldsymbol{A}_i n_j||^2$$

- Verbs can be represented as high-order tensors. If only subject is taken into consideration, it is a two-dimensional matrix. When also considering object, then it is three dimensional (or even higher)
- *Polysemy* (different forms of a word) are mostly handled by a single representation. We assume that ambiguity can be handled as long as the context is given.
- To identify *metaphors*, two separate senses of every adjective can be learned (literal and metaphorical). We then map from literal to metaphorical by a linear transformation.

### 5.1.3 Compositional semantics in neural networks

- Supervised learning framework $\Rightarrow$ compositional representations are fine-tuned for specific application/task
- Word representations are taken as input and processed within the network
- Example tasks include sentiment classification, paraphrasing, machine translation, ...
- Using recurrent and/or recursive networks (LSTMs, Tree-LSTMs, ...)

## 5.2 Discourse structure

- Most documents have a implicit (in news paper articles, first sentence is a summary) or explicit structure like sections and paragraphs
- There are also relationships between sentences that need to be modeled as follow

### 5.2.1 Rethorical relations

- There are implicit relations between sentences. For example:
  `Max fell. John pushed him.`
  can be interpreted as *explanation* (Max fell because John pushed him), or as *narration* (Max fell and then John pushed him).
- This relation is called **discourse relation** or **rhetorical relation**
- **Cue phrases** indicate what kind of relation it is. In the previous examples, the cue phrases were `because` and `and then`.
- Analyzing a text for rhetorical relations mostly gives a binary structure: the main sentence is called **nucleus**, and subsidiary phrase (explanation, justification, ...) is called **satellite**
- In a *narration* (cue phrase `and`) both sentences have equal weight instead of nucleus vs satellite.

### 5.2.2 Coherence

- Discourses need to have connectivity/context to be coherent.
- Otherwise, a sentence/small discourse might not make sense
- However, this information is mostly missing (background/world knowledge)!
- Assuming discourse coherence can affect interpretation. Especially when dealing with pronouns, th

### 5.2.3 Overview of factors influencing discourse interpretation

1. *Cue phrases* (`because`, `and`, ...)
2. *Punctuation and text structure* (`Max fell (John pushed him), and Kim laughed.`)
3. *Real world context* (`Max was falling. John pushed him as he lay on the ground.`)
4. *Tense and aspects* (`Max was falling. John pushed him.`)

- Discourse parsing (understanding discourse structure) is a hard task
- Mostly done by supervision (annotated data of about 8-10 discourses)
- However, *surface techniques* (primitive algorithms that look at characteristic phrases, punctuation, ...) seem to work to some extent

## 5.3 Referring expressions and anaphora

- To fully process a discourse, co-references/referring expressions like pronouns need to be resolved
- We can define the following entities for a referring expression:
  - *referent* - a real world entity to which is referred
  - *referring expression* - part of speech that refers to an entity
  - *antecedent* - the text initially evoking a referent (where referent is named)
  - *anaphora* - the phenomenon of referring to an antecedent
  - *cataphora* - pronouns that appear *before* the pronoun (rare)
- **Pronoun resolution**
  - Identifying the referents of pronouns
  - *Anaphora resolution*: in most cases, the task is limited to identifying referents that are mentioned before the actual pronoun/reference

### 5.3.1 Algorithms for anaphora resolution

- For anaphora resolution, we mostly apply a supervised training algorithm
- The instances in the corpus are possible pairs of pronoun and antecedent (possible antecedent include all noun phrases in the current and last 5 sentences)
- The classification is binary (true if pronoun refers to this specific antecedent, otherwise false)
- Training data is annotated by humans
- Beware that there are also pronouns in the text that might have no referent at all (*pleonastic pronouns*)
- Distinguishing between *hard* and *soft* constraints that must be fulfilled between pronouns and antecedent
- **Hard constraints** : Pronoun must match in terms of tense, singular/plural, gender, ...
- **Soft constraints/Salience**:
  - *recency* - more recent antecedents are preferred
  - *grammatical role* - subjects might be referred to more often than objects. Also, it is preferred that entity and pronoun has same role in sentence (subject, object, ...)
  - *repeated mention* - entities that have been mentioned more often are preferred
  - *coherence effect* - pronoun resolution might depend on discourse relation/semantic within the sentences
- Based on the hard and soft constraints, we can define features for every pronoun-antecedent pair
- Simple classification model takes these features as input and classifies the link as valid or not
- Simplest evaluation matrix is link accuracy (number of correct links). However, it does not take into account pleonastic pronouns or a chain of references so that multiple metrics exist

# 6 Textual Entailment and Paraphrasing

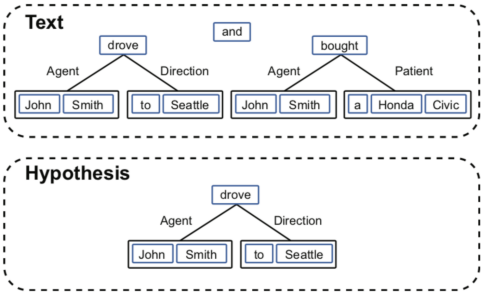
- Textual entailment is defined as a directional relationship of text $T$ to hypothesis $H$
- We say $T$ entails $H$ if the meaning of $H$ can be inferred from the meaning of $T$
- Task of recognizing textual entailment aims for classifying a pair of sentences as whether they are an entailment or not (binary classifier). Can be used in different settings:
  - *Question-Answering*: A question answering system generates $n$ candidate solutions. The textual entailment recognizer must now decide which candidate solution is correct
  - *Summarization*: A summarization system sequentially generates new sentences to add to the summary in progress. The textual entailment recognizer should now identify whether a new sentence contains information that is already in the summary or not (redundancy checker).

## 6.1 Levels of Representation

- Determining the equivalence of the meaning of $T$ and $H$
- The representation of the $T$-$H$ pair is used to train a supervised model
- There are different levels of representation that can be used (all having their own benefits and drawbacks)
- **Lexical level**
  - Solely looking on the words used in $T$ and $H$ (basically BoW of both sentences)
  - Comparing the used words for similarity (are words of $H$ in $T$)
  - Problem: structure of $H$ and $T$ cannot be fully captured by BoW
- **Structural level**
  - Build up syntactic structure (like parse tree from context-free grammar or dependency graph) for $T$ and $H$

- If $T$ contains same structures as $H$ (i.e. certain dependency edges, subtrees, ...), we predict the texts to be entailed
- However, it is hard to distinguish which edges should contribute to similarity and which not

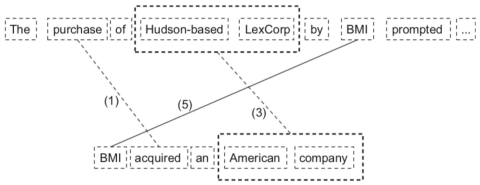
- **Semantic level**
  - The idea is to label words/phrases with semantic role in sentence
  - Words are group into *arguments* (such as person or place) and connected to *predicates* (mostly verbs)
  - Now we check whether semantic connections of $H$ are in $T$ or not




- **Knowledge Acquisition for RTE**
  - To answer some text entailments, background/world knowledge is required (which words are synonyms, what is connected to a certain noun as i.e. a person/place...)
  - Knowledge is mostly constrained to lexical-semantic between two words (synonym, hypnonymy, ...)
  - But we can also model more complex relations like $X$ causes $Y \implies Y$ is a *symptom* of $X$
  - Such connections/knowledge can be retrieved from WordNet, Wikipedia, ...
  - This leads to the **Extended Distributional Hypothesis**: if two paths occur in similar contexts, the meaning of the paths tend to be similar ($X$ *solves* $Y$ compared to $X$ *is a solution of* $Y$)

## 6.2 Recognizing Text Entailment Methods

- RTE depend on the representation which is used for $T$ and $H$
- Different approaches to model the classifier
- **Similarity-based approach**
  - Pair with strong similarity score gets high entailment relation
  - Similarity is measured by for example WordNet (how many edges to traverse to get to other word) and string similarity (length or even single letters)
- **Alignment-based approaches**
  - Use heuristics to align junk of words from $T$ to $H$
  - For example, match phrase "*purchase of* $X$ *to* $Y$" with "$Y$ *acquired* $X$"
  - However, we need a knowledge base to infer these relations




- **Formal Logic approaches**
  - Finding proof by theorem prover that $H$ can be proofed by $T$
  - Convert statements in $T$ and $H$ into formal logic

  - Problem: mostly the lack of background knowledge is the bottleneck, as the simplest mistakes/missing statements can stop this approach to get the correct result

- **Edit distance-based approaches**
  - Sequence of transformations that need to be applied on $T$ to get to $H$
  - If the number of transformations is higher than specified threshold, classify relation as `false`
  - Alternative for *expensive* theorem prover
- Evaluation done on dataset with 1,600 $T$-$H$ pairs with accuracy as metric. Lexical baseline is at about 58%

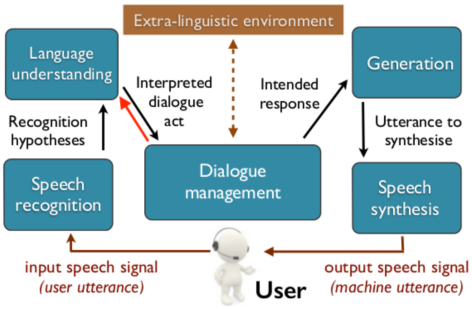
### 6.3 Current methods

- RTE datasets are mostly very small which limits the application of complex systems
- However, there are large Natural Language Inference datasets, where also neural networks can be trained on (different domains, for example image to text)
- We need datasets over multiple domains as otherwise the algorithms generalize poorly
- **Neural networks**
  - Specifying features by hand for the input
  - Using both hypothesis and text as input. Mostly, we classify then into classes *entailment*, *contradiction* and *neutral* (not enough info to decide)
  - Using various LSTM models with attention modules
  - Generative models create a hypothesis given the text and the class for which the hypothesis should be generated
  - However, networks show to overfit on noise in the data (contradiction mostly contains negative words, entailments biased on animals and so on)

## 7 Computational Dialog Modeling

### 7.1 Modular dialog systems

- There are two main tasks in dialog modeling: either understand a conversation from outsider's view (summarizing), or the capability to take part in conversation $\Rightarrow$ make a dialog agent
- First approaches of modeling dialog agent were based on hand-specified patterns/transformation rules based on keywords to find an appropriate answer
- Recently, the focus shifted towards data-driven methods by either retrieving existing information or generating new sentences by i.e. Encoder-Decoder architectures
- Problems: hard to evaluate, such systems often show to just copy patterns in training dataset but don't generalize well.
- Different approach: in dialogs, there is a tendency to ascribe goal and **intentions**
- However, intentions are not easy to recognize. That's why such methods are often used for **task-oriented** dialog systems where the end-goal makes intentions tractable
- The modular dialog system architecture:
- *Language understanding*: NLP1 course. Morphology, POS tagging, lexical semantics and syntactic parsing, compositional semantics, ...
- *Dialog management*: consists of two modules:
  - *Dialog state tracker*: handle linguistic context (what has been said) and how relevant it is to the task. We can convert messages into slots with parameters (like `request(name)`) to simplify the task.
  - *Dialog policy*: select what action to take next/model the next answer. Estimate probabilities for possible actions, and choose best ones. Training mostly done in a reinforcement learning way with simulator
- *Extra-linguistic environment*: taking information into account which is not coming from this dialog itself (images, databases, ...)




## 7.2 Visually grounded, task-oriented dialog

- **Visual dialog**: given an image and a history of human-dialog, answer a follow-up question
- We can evaluate this task with the same metrics as for summarization and translation (BLEU, ROGUE)
- However, in this task the agent is thrown at random into a conversation without being able to interact
- *Image guessing game*: one agents ($Q$) sees the original image, and the other agent ($A$) sees the image with the object highlighted that $Q$ needs to guess
- Current implementation is based on LSTMs with CNN encoders. Still, the models perform poorly compared to humans

# 8 Language generation and summarization

- Most tasks/methods until now have concentrated on language analysis. Next coming: tasks where we have to generate text
- Generation mostly has the starting point at semantic representation like distributional semantics or hidden representation for neural networks
- We can also concentrate on *regeneration* where we convert input to another representation. Examples include summarization, translation, ...
- For generation, there are various subtasks (e.g. content selection, discourse structuring, ...)
- Approaches for generation include:
  - *Templates*: fixed text that has slots that can be filled
  - *Statistical*: using machine learning
  - *Deep Learning*: using deep embeddings, especially for regeneration task

## 8.1 Text Summarization

- Task: generate short version of input text with important points
- We distinguish between **single-document summarization** (given a single document, produce summary with important points) and **multi-document summarization** (given a set of documents, produce brief summary of combination)
- Also, we differentiate between **generic summarization** (identifying important parts by itself and present these) and **query-focused summarization** (regarding a query/question from the user, find relevant parts in the document/s)
- There are mostly main approaches:
  - **Extractive summarization**: extract important info from document by copying sentences and combine them into a summary
  - **Abstractive summarization**: interpret content of document and generate completely new sentences (much harder task!)
- Most approaches deal with extractive summarization as it is much easier to realize and achieves better results till now

### 8.1.1 Extractive summarization

- For extractive summarization, there are three main steps:
  1. **Content selection**: identify important parts/sentences from the document
  2. **Information ordering**: order the sentence within the summary
  3. **Sentence realization**: optimizing the text by e.g. sentence simplification
- Approaches for *content selection*
  - *Unsupervised*:
    - Take those words that are significantly more often used than in other documents in average ⇒ these are the "informative" words and mostly biased towards names/cities (pronoun resolution important to find these references as well)
    - Measured by metrics like `tf-idf`
  - *Supervised*:
    - Large training corpus with human summary needed
    - Sentences of summary are aligned with those in the original document, and features are extracted (position in document, sentence length, informative words, ...)
    - Based on these features, we train a binary classifier whether a sentence should be included in the summary or not
    - Problem: expensive to generate all this data, and the supervised approaches did not significantly outperform the unsupervised ones
- Approaches for *information ordering*:
  - For a single document, the sentences are mostly structured in the order they occur in the original document

### 8.1.2 Query-focused multi-document summarization

- For query-focused multi-document summarization, we need to extend the extractive summarization by two pre-processing steps:
  1. Find a set of relevant documents
  2. (Optionally) simplify sentences in the documents (to make the task of content selection easier)
  3. *Content selection*: identify informative sentences in the documents (much harder than for the single-document task)
  4. *Information ordering*: order the sentences in the summary
  5. *Sentence realization*: modify sentences to get consistent summary
- Approaches for *sentence simplification*
  - Parse sentences and apply hand-rules what parts of a sentence we might drop (initial adverbials as "for example", irrelevant attribute clauses, ...)
  - Also possible to train a classifier to identify satellites (non-informative parts on a nucleon phrase)
- Approaches for *content selection* for multiple documents
  - We can either combine all documents into one, or retrieve information from all documents separately and weight these documents
  - Estimate informativeness similarly to single-document
  - Then, start by adding the most informative sentences in summary (one by one) until the maximum length of the summary is reached
  - When adding new sentences, we need to make sure that not the same/very similar sentences from different documents are added ⇒ **Maximum marginal relevance**
    - Iterative method to determine best sentence to add to summary. Relies on two counter-part measures:
    - *Relevance to query*: high cosine similarity between a sentence and the query indicates a high relevance for the summary

    - ∗ *Novelty regarding the summary so far*: low cosine similarity between sentences and summary
    - ∗ Estimated score is calculated as follows (for query $Q$, summary $S$, documents $D$):

$$\hat{s} = \arg\max_{s_i \in D} \left[ \lambda \text{sim}\left(s_i, Q\right) - (1-\lambda) \max_{s_j \in S} \text{sim}\left(s_i, s_j\right) \right]$$

- Approaches for *sentence ordering*
    - *Chronologically*: for example by date of document
    - *Coherence*: sentences that are similar/discuss same entity should be grouped together in the summary
    - *Topical ordering*: learns set of topics present in documents (by e.g. topic modeling), and then order the sentences by topic

### 8.1.3 Summarization using neural networks

- We can apply neural networks for the task of summarization
- For extractive summarization, we train a RNN on word level creating a representation of words, and a RNN on sentence/document level that combines sentence embeddings
- Apply classifier on output of all document-level RNN to decide whether to include sentence in summary or not (problem: still captures coarse-grained features)

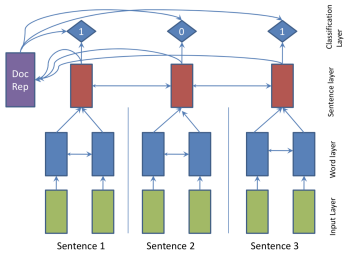



Figure 7: Summarization by RNNs

- Abstractive summarization can be realized by large newspaper datasets where for a small article, a headline must be predicted
- We use an Encoder-Decoder architecture (seq2seq models) where the encoder generates fixed-size embedding, and the decoder generates word-by-word output given this representation (decoder is autoregressive as it takes own output back as input for next time step)

### 8.1.4 Evaluating summarization models

- Human judgments of quality is too expensive
- Better, automatic method: **ROUGE** (recall oriented understudy for gisting evaluation)
- We compare a few human-generated summaries $R_1, ..., R_N$ with the system generated summary $S$ by computing the percentage of $n$-grams from the reference summaries $R_1, ..., R_N$ that occur in $S$. Example: ROUGE-2 (using bigram):

$$\frac{\sum_{R_i} \sum_{bigram_j \in R_i} \text{count}_{\text{match}}(j, S)}{\sum_{R_i} \sum_{bigram_j \in R_i} \text{count}(j, R_i)}$$

- Note that summary length is not considered here
- Example for calculating ROUGE metric:

Question: "What is dadaism?"

Human 1: Dadaism was an art movement formed during the First World War in Zurich in negative reaction to the horrors of the war.

Human 2: Dada or Dadaism was a form of artistic anarchy born out of disgust for the social, political and cultural values of the time.

Human 3: **Dadaism was** a short-lived but highly influential **art movement** from **the early 20th century**.

System: Dada or **Dadaism was** an **art movement** of the European avant-garde in **the early 20th century**.

$$\text{ROUGE-2} = \frac{5+4+5}{21+22+13} = \frac{14}{56} = 0.25$$

# 9 Machine Translation

## 9.1 Statistical Machine Translation

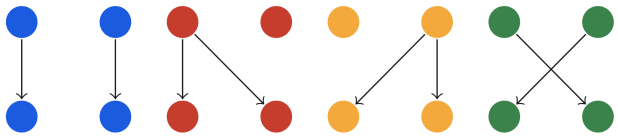
- Given a sentence $f$ in foreign language, find most probable translation $\hat{e}$:

$$\hat{e} = \arg\max_e P(e|f) = \arg\max_e \underbrace{P(f|e)}_{\text{channel}} \underbrace{P(e)}_{\text{source}}$$

- The source is the **language model** which makes sure that the grammatical structure in the text is correct
  - It is also helpful for disambiguate the word decision in the translating language
  - This is very important if a word in the foreign language is ambiguous
- The (noisy) channel is the **translation model** which is responsible to translate the text (makes sure that $f$ are translations of $e$)
- IBM-3 model:
  - For every word:
    - Choose a fertility $\phi_i$ (number of words in goal language should be translated into in foreign language. E.g. “did” has fertility of 0, “slap” 3 in French)
    - Generate $\phi_i$ foreign words
    - Generate spurious/default words that might be needed
  - Permute translated words based on the position a word was before, and language it was in before

### 9.1.1 Learning parameters of models

- For learning the parameters of the language and translation model, we would need the word alignments in the translation which however require the parameters
- Thus, we apply the Expectation-Maximization algorithm
- Assume we have alignments, but for every sentences multiple ones. For example, we can have the following possible alignments for a two-word sentence:



- Every alignment is assigned to a probability/fractional count it occurs. Initially, we set the probability of every alignment/word to a uniform distribution
- We try to maximize the probability that a foreign word/phrase $f_j$ in our corpus is a translation of $e_{a_j}$ where $a_j$ is the alignment of the foreign to translated language. When using Bayes rule (and looking at only 1to1 alignments), we maximize:

$$P(a, f|e) = \prod_{j=1}^{M} t(f_j|e_{a_j})$$

where $t$ are the fractional counts

- EM algorithm:

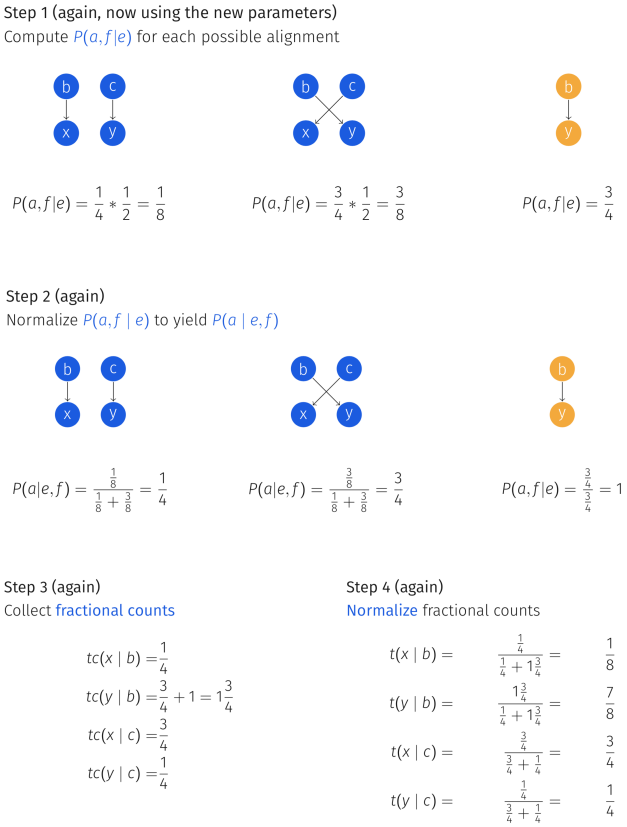
Step 1: Compute $P(a, f|e)$ for every possible alignment and sentence

Step 2: Normalize the alignments for the same foreign sentence. $P(a, f|e) \rightarrow P(a|f, e)$

Step 3: Collect the fractional counts $tc(x|b)$ by summing up the probabilities of all $P(a|f, e)$ where $b$ is aligned to $x$.

Step 4: Normalize fractional counts by $b \Rightarrow$ revised parameters for next iteration

- Example: Given $t(x|b) = 1/4$, $t(x|c) = 3/4$, $t(y|b) = 1/2$, $t(y|c) = 1/2$.




- Note on EM: Optimization function is non-convex so that we might find a local minimum

## 9.2 Phrase-based Statistical Machine Translation

- Previously, translation was based on single words as atomic unit. However, we can also use phrases (few consecutive words) as unit
- The advantage is that context can be taken into account for translation, and no more fertility, insertion and deletion of words are necessary to translate
- We now have a phrase table where we have probabilities to translate a certain phrase into another
- The translation model uses phrases instead of words, but also needs to consider to reorder the phrases:

$$P(f|e) = \prod_{i=1} \phi(\overline{f}_i|\overline{e}_i) \underbrace{d(\text{start}_i - \text{end}_i - 1)}_{\text{distance-based reordering}}$$

  Note that start and end are the positions in the foreign language, but $i$ is the index of the translated language!

- Extract all phrases that are consistent with a word alignment $A$. A phrase is consistent if all words of $\overline{f}'$ are only aligned to words in $\overline{e}'$ and not any other words outside this phrase (and the other way round).
- The *phrase translation probability* $\phi$ is estimated by the relative frequency:

$$\phi(\overline{f}, \overline{e}) = \frac{\text{count}(\overline{f}, \overline{e})}{\sum_i \text{count}(\overline{f}_i, \overline{e})}$$