Faculty of Engineering Sciences
Department of Computer Science
Language Intelligence & Information Retrieval Lab (LIIR)

Natural Language Processing

Prof. Marie-Francine Moens

Course Revision (2019/20)

Guidelines:

- Open book - slides/notes allowed
- 4-hour exam (27th/January 2020 @14:00)
- Exercise questions are similar to exercise questions and algorithms in the lectures/exercises
- Theoretical questions (what we understand in the course/avoid copying definitions/focus on what you understood)
- Open questions (applied big application problems) - give high-level design/technologies/why you chose this technology
- Calculator permitted but not necessary

Summarised by: Feiyang Tang

Hope us all a great success in NLP

Chapter 1 - Introduction

Evaluation metrics (very important!!)
Methodology interesting - for open questions - how to solve/evaluate question

Chapter 2 - Part of Speech

What is PoS: word classes or syntactic categories of words
Common 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*	S	dollar sign	$
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	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).

PoS models:

a. Generative model: Hidden Markov model (HMM)
b. Discriminative model:
   i. Maximum entropy classifier (Maxent)
   ii. Maximum entropy Markov model (MEMM)
   iii. Conditional random field (CRF)
   iv. Neural network with CRF layer (NN+CRF)

HMM model: directed graphical model (Bayesian network)

- **Computation of the joint probability distribution of the sequence of n tags t**

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

- **Assumption of conditional independence** (limited horizon):

$$P(t_{1:n}) = P(t_1)\, P(t_2|t_1)\, P(t_3|t_2)\, P(t_4|\, t_3) \ldots P(t_n|t_{n-1}) \text{ (first order Markov chain)}$$
$$P(t_{1:n}) = P(t_1) \prod_{t=2}^{n} P(t_i|t_{i-1})$$

- Goal: choose the tag sequence $t_{1:n}$ that is most probable given the observation sequence of words $w_{1:n}$: $argmax_{t1:n}\, P(t_{1:n}|w_{1:n})$

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

emission probability

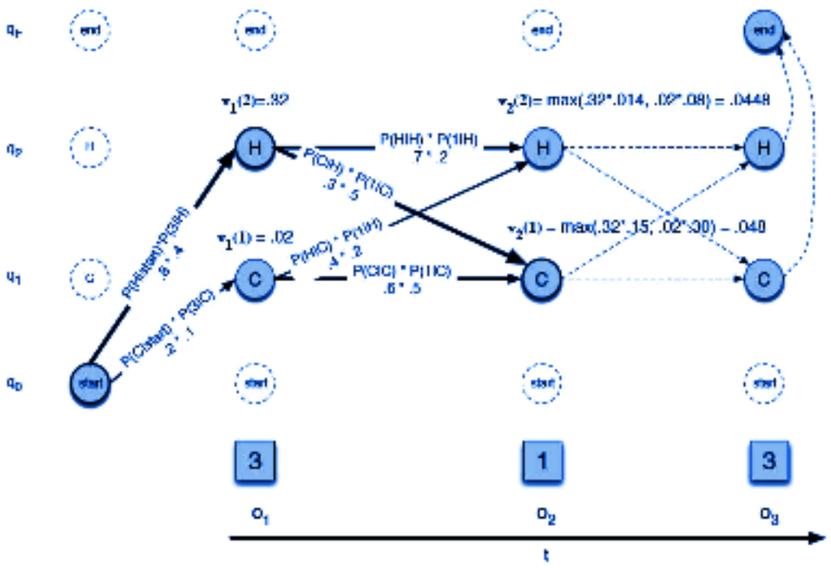
transition probability: here bigram probability, but can be extended to trigram probability => **trigram tagger**

- Decoding: find the best/good sequence based on the prob of the word/tag prediction
  - Parameters are learnt by maximum likelihood estimates on this training data (a fully labelled dataset). In realistic examples, not all possibilities are computed: efficient decoding.
  - Heuristic search: Greedy and beam search
  - Dynamic programming: remember partial info rather than recomputation
    - HMM uses **trellis** or lattice:
      i. Trellis records the probability of all initial subpaths of the HMM that end in a certain state at a certain time
      ii. Probability of longer subpaths can then be worked out in terms of one shorter subpath
      iii. **Viterbi algorithm** for computing the best path (see above)

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



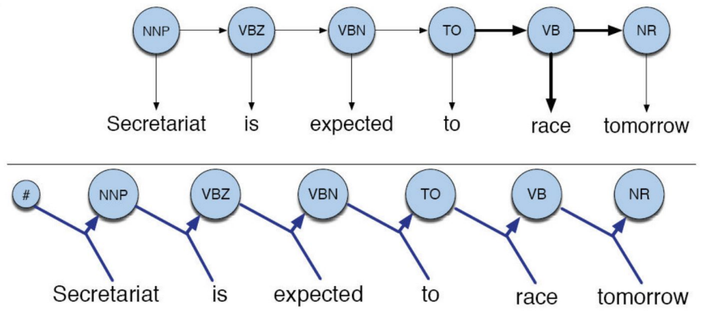
- Maximum Entropy Classifier (MaxEnt): allows to integrate many different features (or feature functions) that act as constraints on the model
  a. Feature function: input observation x and the output class y, and that have a binary outcome
  - The probability of assigning class y to instance x with the **multinomial logistic regression** classifier given K feature functions is then computed as:

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

extra dummy feature f_0 with weight w_0 is added to consider bias term of the linear function

  - The multinomial logistic regression classifier is in NLP also known as **maximum entropy modeling** or **MaxEnt**

- Maximum entropy Markov model: apply MaxEnt classifier to assign a class to each element of a sequence



- **Difference** between MEMM and HMM:
  The HMM (top) and MEMM (bottom) representation of the probability computation for the correct sequence of POS tags. Each arc would be associated with a probability; the HMM computes two separate probabilities for the observation likelihood and the transition probability, while the MEMM computes a single probability function at each state, conditioned on the previous state and current observation.
- Neural Network model: CNN for character representations, add word embedding then feed into BLSTM for training before the last CRF field before output.

Possible Questions:

1. Evaluations on certain techniques are important to help choose/evaluate methods - better to make summaries so good to solve open questions
2. Exercises on calculating probabilities (e.g. HMM/Viterbi)
3. Which kind of decoding do you prefer? (heuristic search/dynamic programming)
4. Algorithms we have seen can be small questions (HMM, Viterbi, MaxEnt, etc.)
5. Feature functions - how we generate it, compute like CRF for PoS tagging (exercises)
6. No need to build a feature function yourself (principles of feature functions)
7. Explain the difference between CRF and Max Markov Model (later in semantic role label)
8. Summarise all techniques by evaluation metrics
   a. Character-based representations or embeddings are very useful:
      i. encode highly inflectional languages (Turkish) more morphology
      ii. encode non-segmented languages (Chinese)

Chapter 3 - Lexical semantics

- **Word similarity computation**: Word similarity or its inverse semantic distance: the similarity between two words is proportional to the number of features of meaning shared
- **Thesaurus based algorithms** (relying on human-built resources such as WordNet)
  - Path-length based similarity

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

where $\mathrm{pathlen}(c_1, c_2)$ = the number of edges in the shortest path in the thesaurus graph between the sense nodes c_1 and c_2 + 1

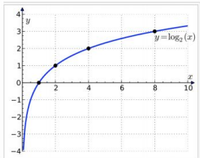
Assumes uniform path length over edges: incorrect assumption for WordNet hypernym hierarchy

  - Information-content word-similarity: still relies on the structure of a thesaurus, but also adds probabilistic information derived from a corpus (e.g., Semcor corpus)

Lowest common subsumer (LCS) of two concepts c_1 and c_2 = lowest node in the hierarchy that subsumes (is hypernym of) both c_1 and c_2 :

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

where the information content (IC) of a concept c, $\mathrm{IC}(c) = -\log P(c)$



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

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

  - **Lin similarity**

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

  - **Jiang-Conrath similarity**

$$\mathrm{sim}_{\mathrm{JC}}(c_1, c_2) = \frac{1}{2 \log P(\mathrm{LCS}(c_1, c_2)) - (\log P(c_1) + \log P(c_2))}$$

- Corpus-based algorithms
  - **Sparse representations**
    - Representing words as discrete symbols: one-hot-encoding
    - Representing words by their context

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 an w_j :

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

$P(w_i)$ 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

- PMI (Pointwise mutual information)
- Collocation = expression consisting of two or more words that correspond to some conventional way of saying things
- PMI is often used as a measure of the strength of a candidate collocations (e.g., where the context window is restricted to neighbouring words in a fixed order)

The above techniques yield a measure of relatedness between words, but have been used to detect typed lexical relations:

- E.g., word co-occurrences are filtered by handcrafted lexico-semantic patterns (e.g., hyponymy: "such as"; meronyms: genitive expressions)

  Difficult task: many false positives, false negatives:
  - Patterns are ambiguous and incomplete
  - Recent results based on deep learning techniques [Schwartz et al. ACL 2016]: dealing with dense representations

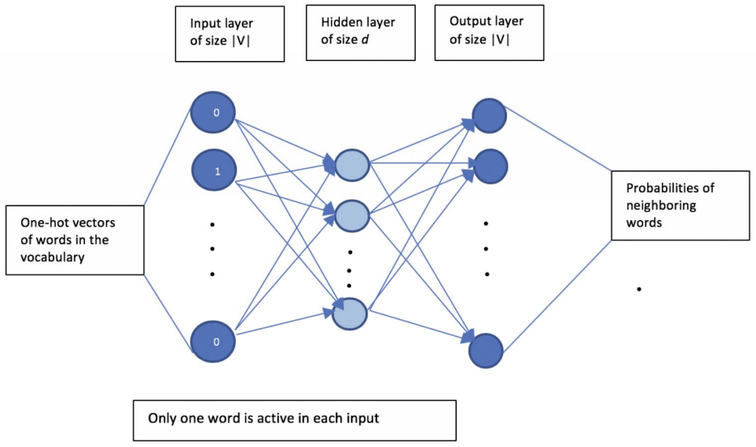
  Evolves to real thesaurus induction

- **Dense representations**
  a. Word embeddings as word vectors: Each word is associated with a real-valued vector in n-dimensional space (usually n = 50 - 1000)
  b. **Skip-gram NNLM**

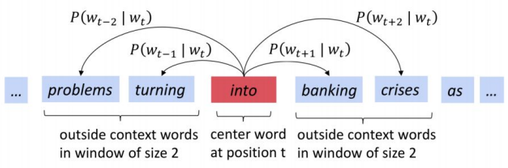
     Input: the current word in the form of a 1-hot vector

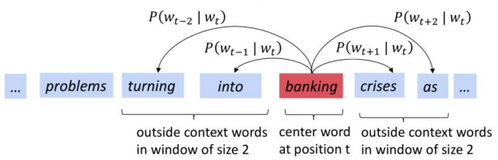
     Output: the words within a short window is predicted, the network predicts the surrounding words, **not their position**

     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: $\boldsymbol{o}_{hidden} = \boldsymbol{o}_{in}\boldsymbol{W}_{in}$

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

$\boldsymbol{W}_{in}, \boldsymbol{W}_{out}$ = interconnection matrices containing the weights, learned during training

Output layer has again the size of the vocabulary: softmax activation function: output: $\hat{\boldsymbol{y}}_{out} = \boldsymbol{o}_{out} = \text{softmax}(\boldsymbol{o}_{hidden}\boldsymbol{W}_{out})$

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

Possible Questions:

1. Not in detail on unsupervised
2. Thesaurus based algorithms should be able to apply (formulas)
3. One hot encoding
4. Computations on similarity or PMI
5. Principle of word vectors (theoretical questions) - why this is a language model, how do build a model
6. What is the difference between the two models?

Chapter 4 - Language modelling

- **Language model**: probabilistic model of word sequences used for word prediction
- **PoS tagging**: predict tags in context
- **Semantic role labelling**: predict semantic roles
- **Simple n-gram models**: use the chain rule to decompose the joint probability into a sequence of conditional probabilities

$$P(w_{1:T}) = P(w_1)\,P(w_2|w_1)\,P(w_3|w_1,w_2)\,P(w_4|\,w_2,w_3) \ldots P(w_T|w_{T-2},w_{T-1})$$
$$P(w_{1:T}) = P(w_1)P(w_2|w_1)\prod_{t=3}^{T} P(w_t|w_{t-2},w_{t-1})$$

MLE:

ML estimation of the *n*-gram models

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

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

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

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

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

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

Some observations:

- The longer the *n*-grams, the more coherent the sentences are when generated
- Language models are pretty useless if training on different dataset then training: the Wallstreet trigrams do not well predict Shakespeare's language

Smoothing method:

Why?

Problem with MLE when training the language model: **sparse data !**

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

How does it work?

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: $P_{Laplace}(w_t \mid w_{t-1}) = \frac{C(w_{t-1}, w_t)+1}{C(w_{t-1})+|V|}$

  where $|V|$ = number of words in vocabulary

Good-Turing discounting: 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: $\frac{N_1}{N}$

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

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

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

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

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

Uses information about a word's class or cluster 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

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

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

Advantages:

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

Disadvantages:

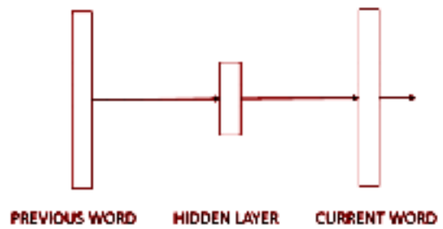
1. n-grams of larger size are very sparse, so cannot be used to capture long-term dependencies
2. Capture symbolic nature of language, but are not good at capturing semantic correlations between words, neither morphological regularities

Feedforward neural network (added to trigram LM):

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

- 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 of simple FF NN models:

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

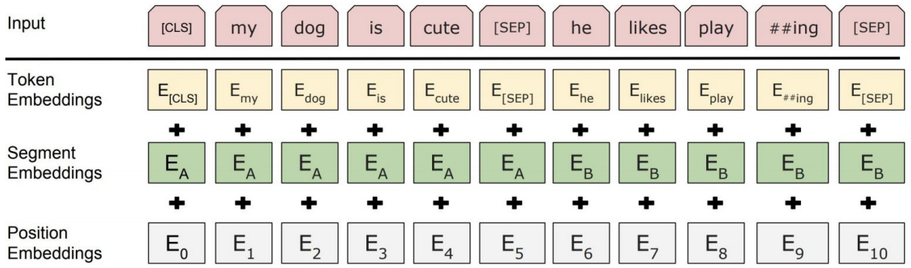
Disadvantages:

1. The number of parameters in the model scales with the n-gram size and thus the length of the history captured
2. Only a fixed number of previous words can be taken into account to predict the next word
3. Has been attempted to solve with sophisticated attention mechanisms (cf. transformer architecture of BERT)

BERT: uses a fully connected feedforward transformer architecture with multi head self attention

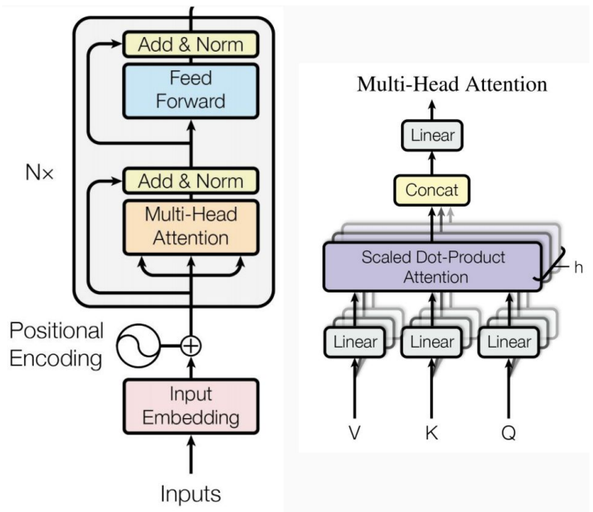
- Input:
  - Summation of token, segment and positional embeddings, representing both a single and a pair of sentences
  - Tokens are represented with word piece embeddings: obtained by breaking words into pieces given a word unit dictionary (automatically learned) and adding word continuation information
    Word tokenizer: He is an unaffable man
    Word piece tokenizer: He is an un ##aff ##able man
  - Total input length of word tokens is restricted to 512

- BERT input representation:



- BERT: training based on two unsupervised prediction tasks
  - <u>Masked language model</u>: 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)
  - <u>Next sentence prediction</u>:
    - 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] [SQP]`
    Label = `IsNext`
    Input = `[CLS] [MASK] live in Zumarraga [SEP] I [MASK] trains [SEP]`
    Label = `NotNext`
- Model Architecture: Transformer encoder
  - Multi-headed self-attention: Models context
  - Feed-forward layers: Computes non-linear hierarchical features
  - Layer norm and residuals: Makes training deep networks healthy
  - Positional embeddings: Allows model to learn relative positioning



Recurrent Neural Network LM:

- Feed forward neural networks (FFNN):
  - Do NOT take into account sequential information
- Recurrent neural networks (RNNs):
  - 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 learnt

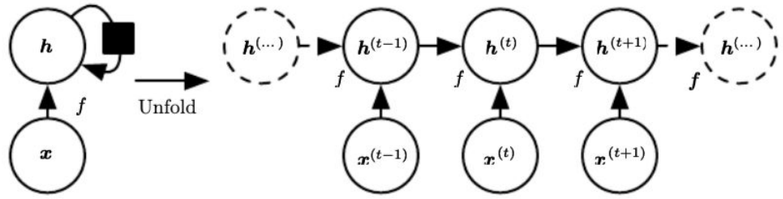


Figure 10.2: A recurrent network with no outputs. This recurrent network just processes information from the input $\boldsymbol{x}$ by incorporating it into the state $\boldsymbol{h}$ that is passed forward through time. *(Left)* Circuit diagram. The black square indicates a delay of a single time step. *(Right)* The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance.

- RNNs are very <u>powerful</u> because:
  - Distributed hidden states that allows them to store a lot of information about the past efficiently
  - Nonlinear dynamics that allows them to update their hidden states in complicated ways

Training RNN:

- During backpropagation through many layers:
  - If the weights are small, the gradients shrink exponentially
  - If the weights are big the gradients grow exponentially
- 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
    - 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 :

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

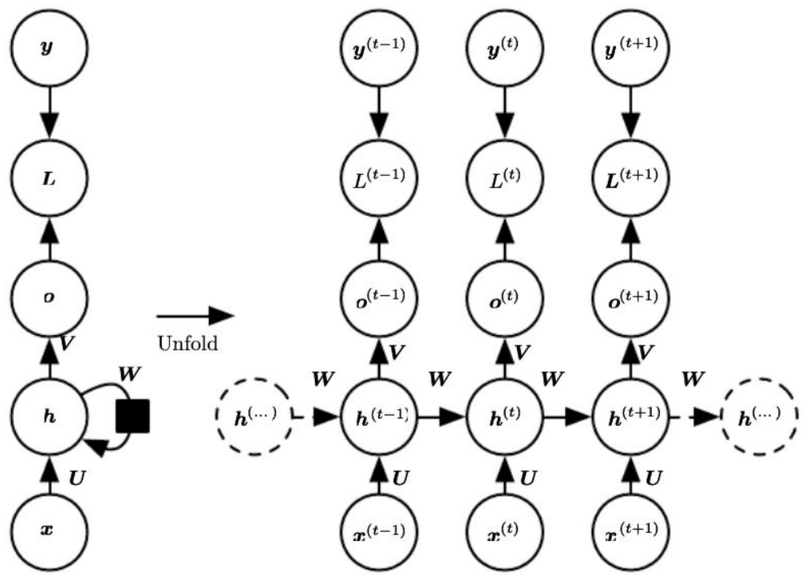


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. 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.

variable. We can then apply the softmax operation as a post-processing step to obtain a vector $\hat{\boldsymbol{y}}$ of normalized probabilities over the output. Forward propagation begins with a specification of the initial state $\boldsymbol{h}^{(0)}$. Then, for each time step from $t = 1$ to $t = \tau$, we apply the following update equations:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \quad (10.8)$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}), \quad (10.9)$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \quad (10.10)$$
$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)}), \quad (10.11)$$

where the parameters are the bias vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ along with the weight matrices $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$, respectively, for input-to-hidden, hidden-to-output and hidden-to-hidden connections. This is an example of a recurrent network that maps an input sequence to an output sequence of the same length. The total loss for a

RNN based Language Models Advantages:
RNNs can represent unbounded dependencies RNNs compress histories of words into a fixed size hidden vector
Disadvantages:

Pure RNNs are hard to learn and will often not discover long-range dependencies Increasing the size of the hidden layer, and thus memory, increases the computation and memory quadratically

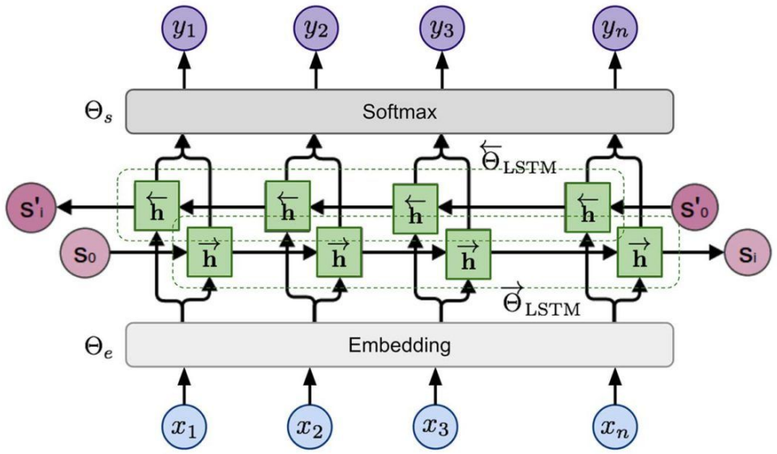
ELMo

- ELMo uses a **bidirectional LSTM** architecture (often several layers)
- Input = sequence of words
- Output = next word given the previous history of words
- Objective = minimise 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 $\boldsymbol{h}_{t,0}$: $\hat{y} = \gamma \sum_{j=0}^{L} \lambda \, softmax(h_{t,j})$

  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



Perplexity

- 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:

$$PP = P(w_{1:T})^{-\frac{1}{T}}, \quad \frac{1}{T} = \text{normalisation by \# of words}$$

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

(chain rule and first-order markov)

- Because it is the inverse of probability, minimising it means maximising the probability of a sentence. Therefore, lowering it is a target.
- Intuition of perplexity: e.g., for n-gram model of words: how good is the learned model to predict the next word when running over the complete test string from left to right
- Perplexity lower is better

- As perplexity is the inverse of the probability of the dataset, we can write perplexity in terms of 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.

Possible Questions:

1. Smoothing method - should be able to apply, e.g. good-turing
2. Class-based n-grams
3. FFNN-language model - different between FF and recursive LM (sentence representation differences)
4. BERT: transformer arch
5. Principle of Recursive NN model: encoding, decoding
6. ELMo - how this is different from other models and also similarities
7. Perplexity

Chapter 5 - Syntactic parsing (Long Chapter Alert!)

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:

- 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 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:

- Use of a context-free grammar for parsing using terminal and non-terminal symbols

Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure 12.1 The $\mathscr{L}_1$ miniature English grammar and lexicon.

Cocke – Kasami - Younger (CKY) algorithm: chart based algorithm based on a (P)CFG

- 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: $A \rightarrow B\ C$; $A \rightarrow w$
- Conversion 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

$\mathscr{L}_1$ 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.

CKY algorithm:

- 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 $(n + 1) \times (n + 1)$ matrix *M*
- The super diagonal of *M* contains the parts-of-speech for each input word in the input
- The subsequent diagonals above that super diagonal contain constituents that cover all the spans of increasing length in the input
- Example of how to perform CKY on Exercise slides

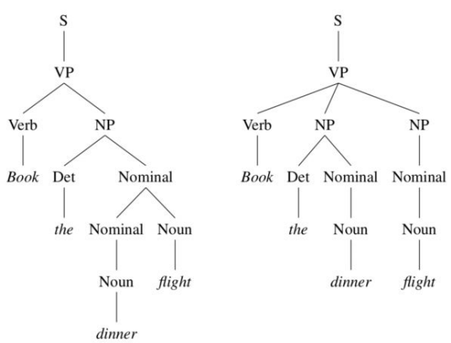
```
function CKY-PARSE(words, grammar) returns table

  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
```

Probabilistic CFG:

Probabilistic context-free grammar (PCFG): a probabilistic augmentation of context-free grammar 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]	| $meal$ [.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]	

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



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

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

- Disambiguation: selected tree $\widehat{T}$, for sentence S: $\widehat{T}(S) = argmax^T P(T)$
- PCFG Algorithm:

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                                 and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar}
       table[j − 1, j, A] ← P(A → words[j])
     for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
              for all { A | A → BC ∈ grammar,
                            and table[i,k,B] > 0 and table[k, j,C] > 0 }
                  if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                      table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                      back[i,j,A] ← {k,B,C}
     return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

<u>Problems</u> with simple PCFG:

- **Maximum likelihood estimation** of the probabilities estimated on a treebank performs poorly:
  - The naive grammar is a poor one because its context-free assumptions are too strong in some places (e.g., it assumes that subject and object NPs share the same distribution) and too weak at other places (e.g., some constituents are not decomposable into smaller steps)
- **Poor independence assumptions**:
  - CFG rules impose an independence assumption on probabilities, resulting in poor modelling of structural dependencies across the parse tree
- **Lack of lexical conditioning**:
  - CFG rules don't model syntactic facts about specific words, leading to problems with sub-categorisation (The same verb might be ambiguous between several subcategories. For instance, "give" is ambiguous between NP PP and NP NP), PP attachment and coordinate structure ambiguities

<u>Solutions for PCFG</u>:

- Enriching the naive grammar:
  - **Lexicalisation 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 = subcategorization the POS categories: at the extreme there is a tag for every single word or word meaning
- Generalising the naive grammar:
  - E.g., by smoothing
  - Among which is also **learning** the tag-splitting

Combinatory Categorial Grammar (CCG)

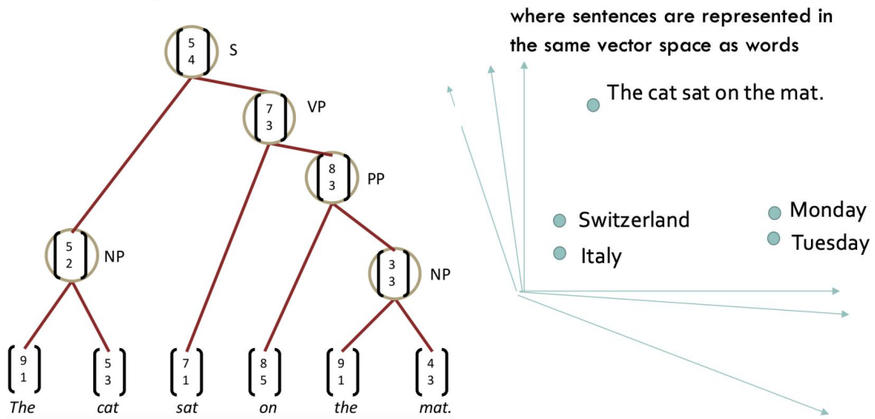
- Lexicalised grammar
- Captures non-local dependencies involved in treebank construction
- Adds categorical and combinatory information:
  - Information about syntactic, phonological and semantic aspects of a given word
  - Information about categories with which it can be combined and the result of the combination

Constituent Parsing

- Lexicalisation
  - Lexicalised parsers: include lexicalized grammar rules

- Lexicalised grammar:
  - Each non-terminal in the tree is annotated with its lexical head:
    `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 the sparseness of the data:
  - Independence assumptions to break down each rule so that we would estimate the probability as the product of the individual probabilities

Recursive neural networks
Principle of 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!
Jointly learning parse trees and compositional vector representations:



Input: two candidate children's representations each in the form of a vector
Output: The parent vector ***p*** representation of their merge

$$p = tanh(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b)$$

Score of how plausible the new node is:

$score(p) = v^T p$ where *v* is a vector of weights that are trained

Score of parse tree *y* of sentence *x*:

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

Principle: Standard recursive neural network⇒same *W* parameters at all nodes of the tree are used
The objective function (Max-margin):

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

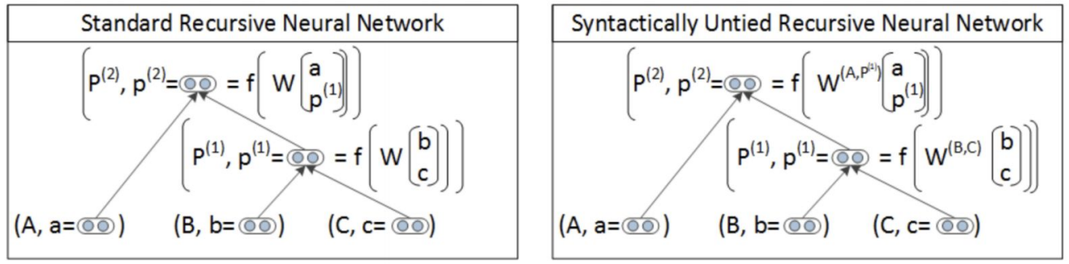
where y_i = correct parse in the training data
$A(x_i)$ = set of candidate parses of x_i
$\Delta(y, y_i)$ = penalty loss, =0 when $y=y_i$, >0 when $y \neq y_i$, thus penalises incorrect parses

- In principle, there are an exponential number of possible parses of x_i:
- In practice:
  - Greedy search (join best nodes each time)
  - Or keep beam of *k* best joins

Recursive neural network for structure prediction

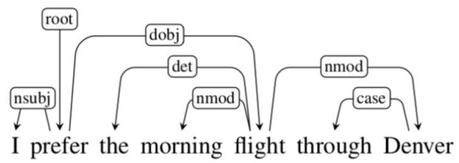
Standard recursive neural network: single weight matrix W:
- The composition function is the same for all syntactic categories, punctuation, etc.
- Decent parse results with single weight matrix, but not adequate for more complex higher order composition and parsing of long sentences

Syntactically united recursive neural network:
- Use of CFG to choose/learn the weight matrix
- Weight matrix depends on the syntactic categories of the children

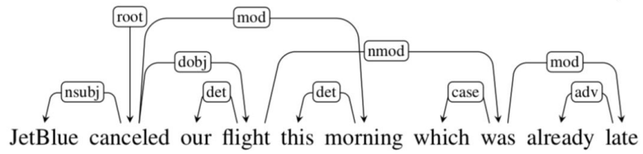


Dependency parsing

- 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



- 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.



- A major advantage of dependency grammar is:
  - their ability to deal with languages
  - that are: Morphologically rich and have a relatively free word order: e.g., when detecting the grammatical function

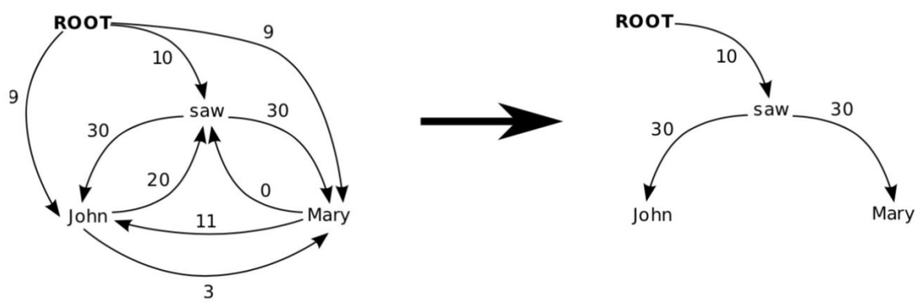
Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	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*.

Idea:



- More specifically, for an input sentence x define a graph $G_x = (V_x, A_x)$ 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 $G_1, \ldots, G_m$:
- $s(T) = \sum_{c=1}^{m} s(G_c)$, Where each G_c does not need to be a subtree.
- 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 (or scores)
- For training, we learn the weights ***w***.
- For Inference or prediction: search for the maximum spanning tree (MST) T* of Gx 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: (example in exercise session)

```
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
```

First-order model

- Scored subgraph G_c is a single arc (*i, j, k*) between head *i* and modifier *j*:

$$S(T) = \sum_{c=1}^{m} s(G_c) = \sum_{(i,j,k)\in T} s(i,j,k) \quad s(i,j,k) = max_k s(i,j,k)$$

- 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 modelled as integer linear program.

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

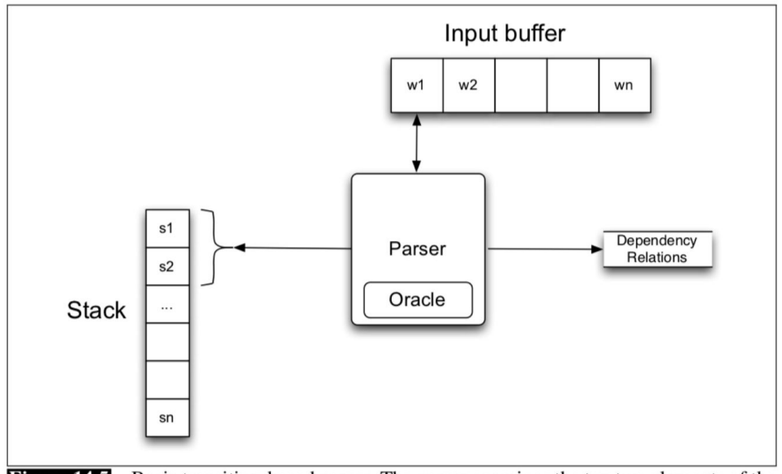


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

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	

Figure 14.7 Trace of a transition-based parse.

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 labelled 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 the entire sequence.
- Use beam search to find highest-scoring sequences

Neural Dependency Parsing:

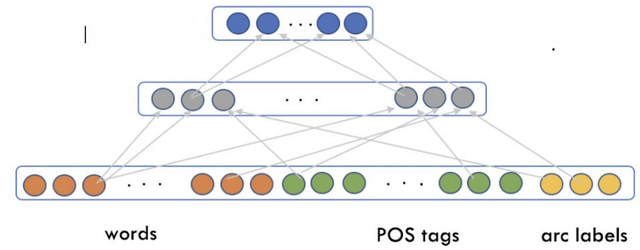
- Illustrated with transition-based dependency parser that predicts transition operators.
- The neural network learns compact dense vector representations of words, POS tags, dependency labels.
- sult is a fast compact classifier, which only uses 200 learned dense features

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 $F_\beta = \frac{(\beta^2+1)PR}{\beta^2P+R}$, where P = labeled precision and R = labeled recall
  - 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
- Labelled attachment score (LAS): recognition of head and label

Possible Questions:

1. CP and DP (two models principles)
2. CKY algorithm - compute the most probable path based on the CKY
3. Advantages and disadvantages of CFG etc.
4. RNN - how you build a constituency parser, know principle and what is happening behind it
5. Graph dependency parsing: how to compute Chu-Liu algorithm using Maximum spanning tree
6. Difference between first or second-order model

7. Transition based dependency parsing (page 66 small exercise)
8. Neural dependency parsing - principle and apply
9. Small exercise on matrix related formulas

Chapter 6 - Named entity recognition semantic role labelling

Named entity recognition:

- Named entity recognition (NER) = to find each mention of a named entity in the text and label
- its type NER types are application specific: 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: two problems:
  - Segmentation
  - Classification

Tasks:

- 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 (**important + major**)
  - 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

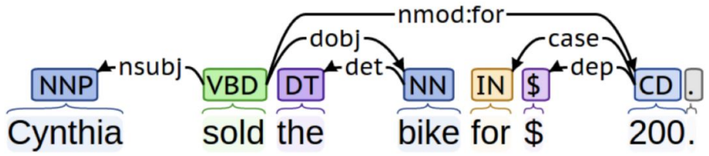
- Use of BIO or IOB tagging:
  - Tag for the beginning (B) and inside (I) of each entity type, and one for tokens outside (O) any entity
  - B= Begin, I = Inside, O = Outside
  - 2*n* + 1 tags, where n is the number of entity types
  - Alternatively: IO tagging using only *n* + 1 tags

Semantic role labeling (SRL)

- 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"...):
- Sometimes called: thematic role labeling, case role assignment or shallow semantic parsing
- 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

  sold nsubj $X \Rightarrow$ SELLER:X

  sold dobj $X \Rightarrow$ GOODS:X

  sold nmod:for $X \Rightarrow$ PRICE:X
- The method is not scalable and generalizable: e.g., many alternatives using syntactic alternations and using different verbs. There are resources with definitions and manually annotated examples:
  - FrameNet (1998)
  - PropBank (2002)

Sequence tagging: Conditional random fields

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., SRL: tagging a sequence of words in a sentence with SR given a predicate
- 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

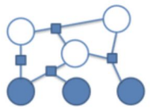
Conditional Random Fields:

- 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
  - 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 $v \in V$ 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 $P(Y|X;\ \theta)$
- $X = (x_1, ..., X_T)$ is a random variable over data sequences to be labelled and
- $Y = (y_1, ..., y_T)$ is a 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

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

Feature functions for CRFs:

- Binary (or real-valued) <u>feature functions</u> model features of local inputs as well as transitions between labels:

$$f_k(y_{t-1}, y_t, X_t) = \begin{cases} 1 \text{ if the observation at position } t \text{ is the word "}run\text{" and } y_t = predicate \\ 0 \text{ otherwise} \end{cases}$$

$$f_k(y_{t-1}, y_t, X_t) = \begin{cases} 1 \text{ if } y_{t-1}\text{B-ORG and } y_t = \text{I-ORG} \\ 0 \text{ otherwise} \end{cases}$$

Representing a Linear CRF:

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

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

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

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 an 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.

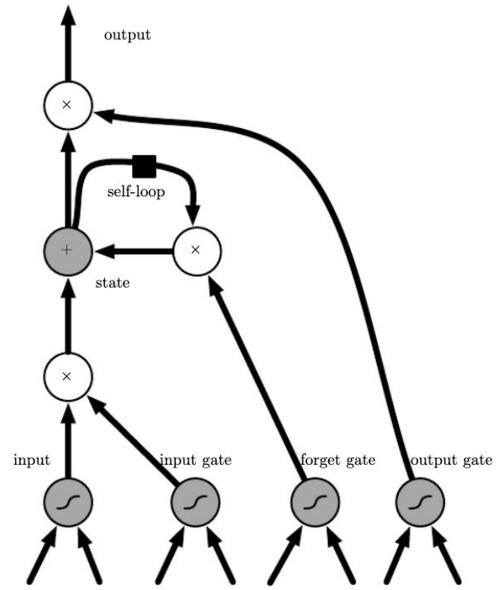


Figure 10.16: Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal 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.

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: $\boldsymbol{o}_t = \text{sigmoid}\left(\boldsymbol{U}^{(o)}\boldsymbol{x}_t + \boldsymbol{W}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o\right)$ — 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 element-wise multiplication

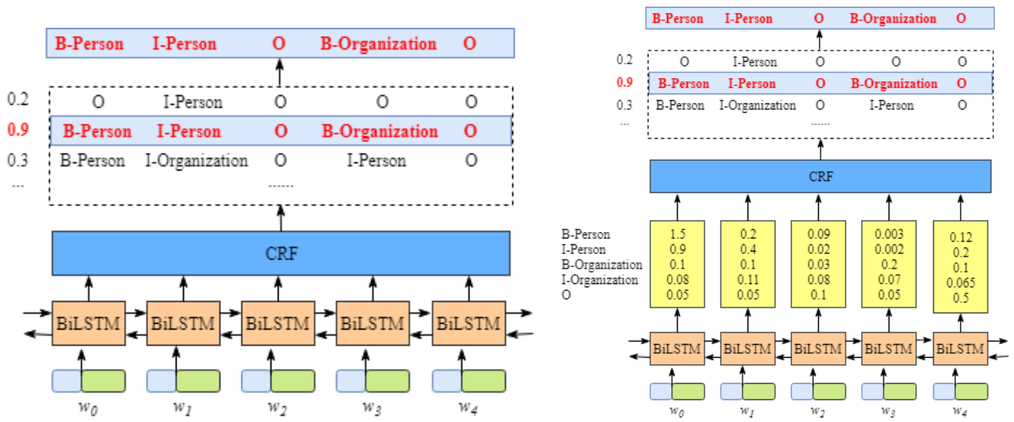
- Very powerful, especially when stacked and made even deeper (each hidden layer is already computed by a deep internal network).
- Most useful if you have lots and lots of data
- Different variations on the LSTM gating are possible, e.g., the Gated Recurrent Unit (GRU)

LSTMs can be the basis for much more sophisticated models:

CRF Layer on the Top of BiLSTM

As shown in the picture below:

- Firstly, every word in sentence x is represented as a vector which includes the word's character embedding and word embedding. The character embedding is initialized randomly. The word embedding usually is from a pre-trained word embedding file. All the embeddings will be fine-tuned during the training process.
- Second, the inputs of BiLSTM-CRF model are those embeddings and the outputs are predicted labels for words in sentence x .

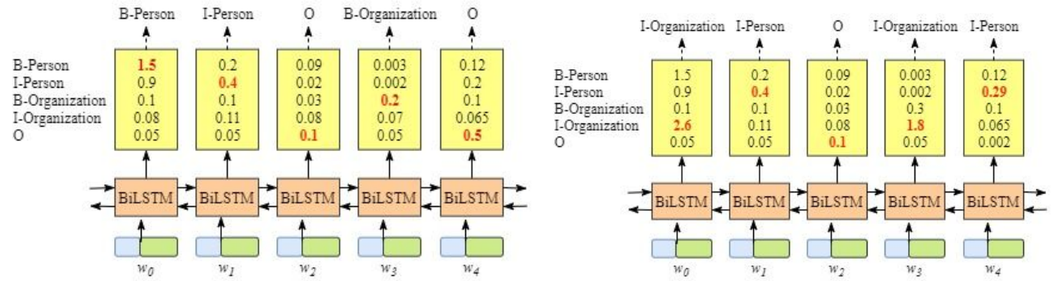


- Although, it is not necessary to know the details of BiLSTM layer, in order to understand the CRF layer more easily, we have to know what is the meaning of the output of BiLSTM Layer.
- The picture above illustrates that the outputs of BiLSTM layer are the scores of each label. For example, for w 0 , the outputs of BiLSTM node are 1.5 (B-Person), 0.9 (I-Person), 0.1 (B-Organization), 0.08 (I-Organization) and 0.05 (O). These scores will be the inputs of the CRF layer.
- Then, all the scores predicted by the BiLSTM blocks are fed into the CRF layer. In the CRF layer, the label sequence which has the highest prediction score would be selected as the best answer.

What if we DO NOT have the CRF layer

- You may have found that, even without the CRF Layer, in other words, we can train a BiLSTM named entity recognition model as shown in the following picture.
- Because the outputs of BiLSTM of each word are the label scores. We can select the label which has the highest score for each word. For instance, for w_0, "B-Person", has the highest score (1.5), therefore we can select "B-Person" as its best-predicted label. In the same way, we can choose "I-Person" for w_1, "O" for w_2, "B-Organization" for w_3 and "O" for w_4. Although we can get correct labels for sentence x in this example, it is not always like that. Try again for the following example in the picture below.

- Obviously, the outputs are invalid this time, "I-Organization I-Person" and "B-Organization I-Person".



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

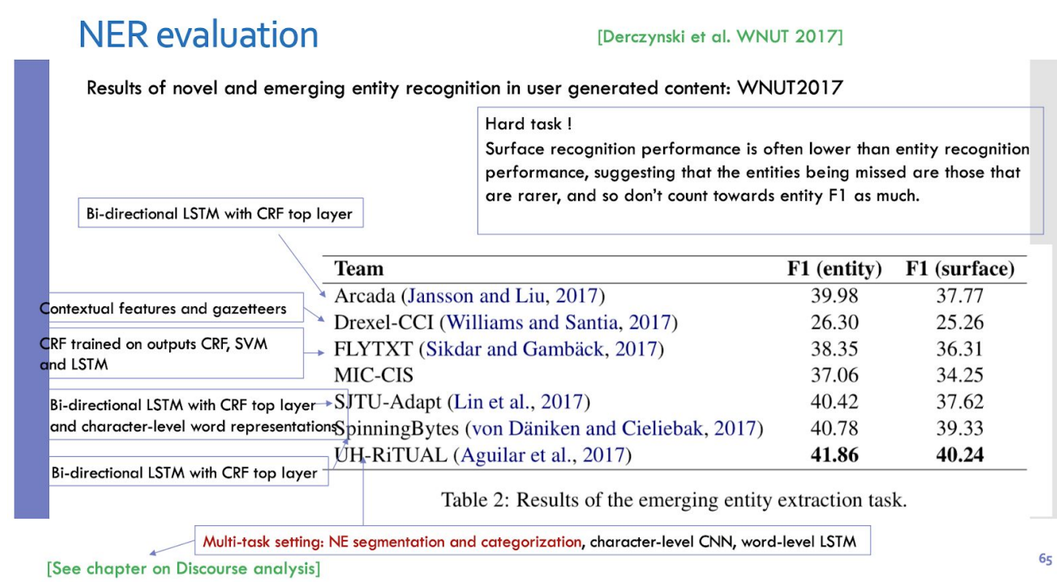
- $\boldsymbol{\lambda} \cdot \boldsymbol{\Phi}(X, Y)$ can be considered as a score of how well the label sequence Y fits the given input sentence
- The idea is to replace the linear scoring function by the score of a non-linear neural network:

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

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

Evaluation on models:



Team	F1 (entity)	F1 (surface)
Arcada (Jansson and Liu, 2017)	39.98	37.77
Drexel-CCI (Williams and Santia, 2017)	26.30	25.26
FLYTXT (Sikdar and Gambäck, 2017)	38.35	36.31
MIC-CIS	37.06	34.25
SJTU-Adapt (Lin et al., 2017)	40.42	37.62
SpinningBytes (von Däniken and Cieliebak, 2017)	40.78	39.33
UH-RiTUAL (Aguilar et al., 2017)	**41.86**	**40.24**

Table 2: Results of the emerging entity extraction task.

Possible Questions:

1. No questions on AMR
2. CRF very important, know models how to scalability, why they are computationally expensive
3. No time to compute calculations for NN should know what actually you want to model in NNs
4. Be able to explain the difference between regular CRF and NN with CRF (LSTM-CRF), know scores

Chapter 7 - Semantic parsing

Basic definitions of SP

- 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: e.g., which yields an answer in case of a question or problem formulated 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, the 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:

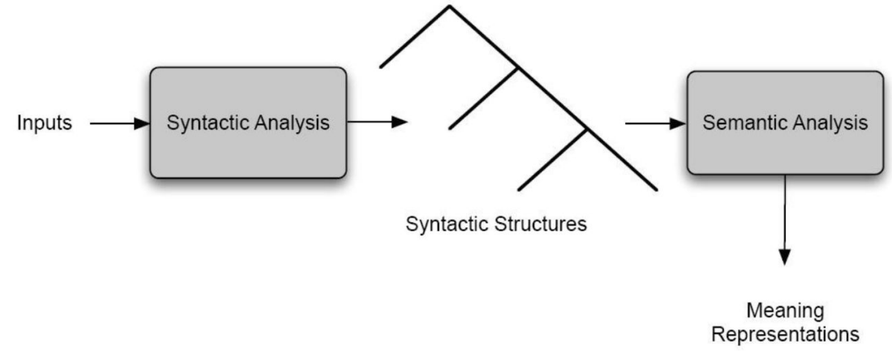
- first-order logic
- Lambda calculi
- natural logics
- diagrammatic languages
- programming languages
- robot controller languages
- database query languages

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 becomes extremely complex and has the 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	$\forall x.\ man(x) \rightarrow walk(x)$
Every man loves a woman	$\forall x.\ man(x) \rightarrow \exists y.\ woman(y) \wedge love(x, y)$

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)
    - Parse sentence to get syntax tree
    - Look up the semantics. Of each word in lexicon
    - 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

  λx.P(x)

  λx.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: λx.P(x)A P(A)
- Eg. λ-application and subsequent reduction to a fully specified logical formula:
- λx. Λy.Near(x,y)(Bacaro) λy.Near(Bacaro,y)
- λy.Near(Bacaro,y)(Centro) 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: $\exists\, e\ closing(e) \wedge Closed(e, Rhumba)$

Feature structure:

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

Associate unification equations with the grammar rules:

```
S → NP VP
<S SEM> = <NP SEM>
<VR ARG0> = <NP INDEXVAR>
<NP SCOPE> = <VP EM>

NP → ProperNoun
<NP SEM> = <ProperNoun SEM>
<NP SCOPE> = <ProperNoun SCOPE>
<NP INDEXVAR> = <ProperNoun INDEXVAR>
```

Given CFG rules: we pair feature structures with the rules to encode constraints and to link semantics to the syntax

= : unification
SEM : meaning representation
INDEXVAR: path leading to SEM representation
SCOPE: scope of variable

```
ProperNoun → 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

Statistical machine learning approaches

Supervised Learning:

- Training: the training examples take the form of pairs (x, y) with $x \in X$ the system's input and $y \in Y$ 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 $\varphi(x, y) \in R^K$ 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 $_j$ 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:

  $$score_w(x, y) = \sum_{j=1}^{K} w_j f(x, y)_j = w \bullet \varphi(x, y)$$

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

  $$\hat{y}(x) = argmax_{y \in Y}\ score_w(x, y)$$

Training:

1. Given a set of examples $\boldsymbol{D}$ consisting of input-output pairs (x, y): optimise the weights $\boldsymbol{w}$ so that the score of the correct output y is generally larger that the score of the incorrect outputs $y' \in Y \setminus \{y\}$.
2. There are many suitable objective functions for the problem, eg. multiclass hinge loss objective

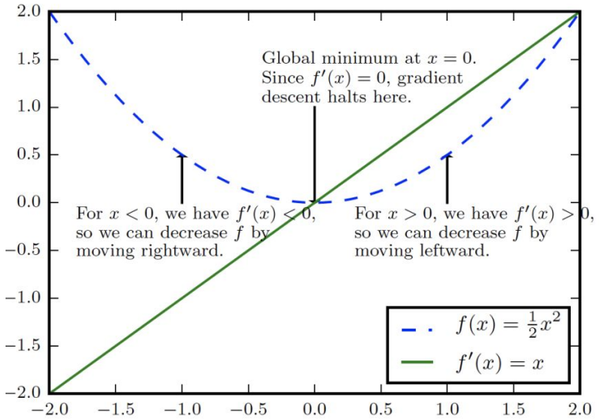
$$J(\boldsymbol{w}) = \min_{\boldsymbol{w}} \sum_{(x,y)\in\mathcal{D}} \max_{y'\in Y} [\text{Score}_{\boldsymbol{w}}(x,y') + c(y,y')] - \text{Score}_{\boldsymbol{w}}(x,y)$$

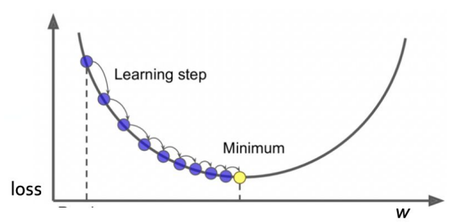
Where $c(y,y')$ = cost for predicting y' when the correct output is y in example below = 0 when $y = y'$, 1 otherwise.

3. In words, we are trying to find the best weight vector $\boldsymbol{w}$ that minimizes the cumulative loss over training examples $\boldsymbol{D}$.

Gradient-based optimisation

- Many supervised learning and most deep learning algorithms involve optimisation:
  - 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 optimisation, 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 $= \frac{\partial y}{\partial x} = f'(x)$
    - $f'(x)$ gives the slope of $f(x)$ at point x and tells how much to change x to make an improvement in y:
      - E.g., **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 negative slope means the function goes down towards the right, so we want to move right to find the minimum

How big of 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 $\boldsymbol{w}$ of the network are adapted with the gradient of the objective function XZ X[scaled by the learning rate $\frac{\vartheta J}{\vartheta w}$
- 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 on a minibatch of examples: "stochastic" approximation of the true gradient
- Guaranteed convergence (under some technical conditions)
- Scales to large datasets
- Semantic parsing task = to learn an accurate mapping from utterances $\boldsymbol{u}$ to logical forms $\boldsymbol{\sigma}$
- Usually a finite set of possible logical predicates and arguments, e.g., weather information
- Interpretation task = to learn an accurate mapping from utterances $\boldsymbol{u}$ to denotations $\boldsymbol{\delta}$ via latent semantic representations, in effect combining semantic parsing and interpretation
- Often involves structure prediction

$\textsc{StochasticGradientDescent}(\mathcal{D}, T, \eta)$

$\mathcal{D}$: 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 $\mathbf{w}$

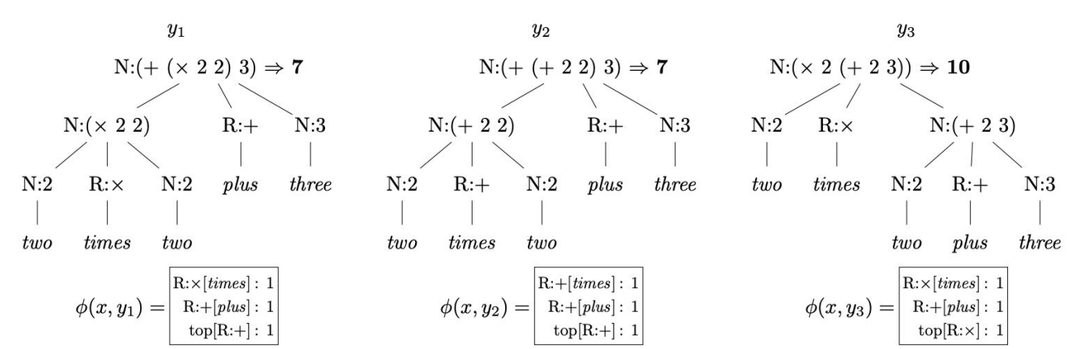
Optimisation by Stochastic Gradient descent (SGD):

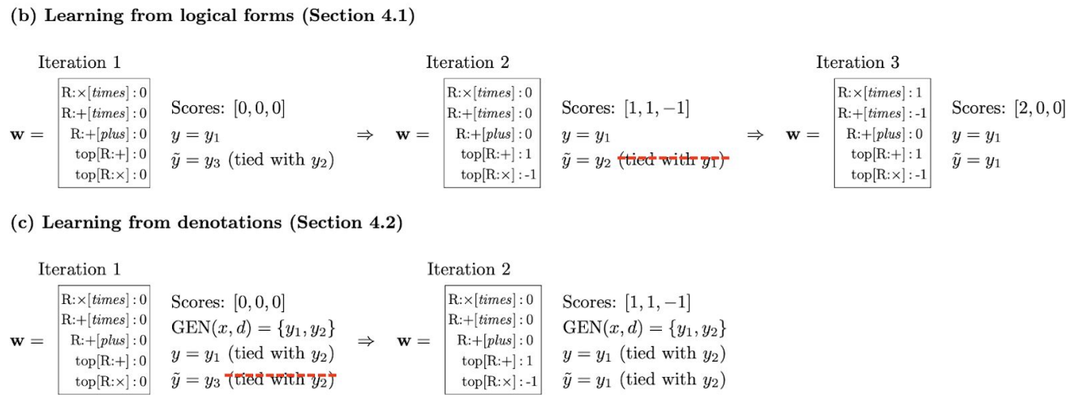
- Iteratively computes the (sub)gradient of a single term of the optimization problem
- For each training example $(x, y) \in D$, SGD make a prediction for it (line 4)
- When the prediction is correct $(y = \hat{y})$, the update does nothing because the difference in line 5 is 0.
- Where the prediction 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 $\boldsymbol{\eta}$ 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

(a) Candidates GEN(x) for utterance x = *two times two plus three*





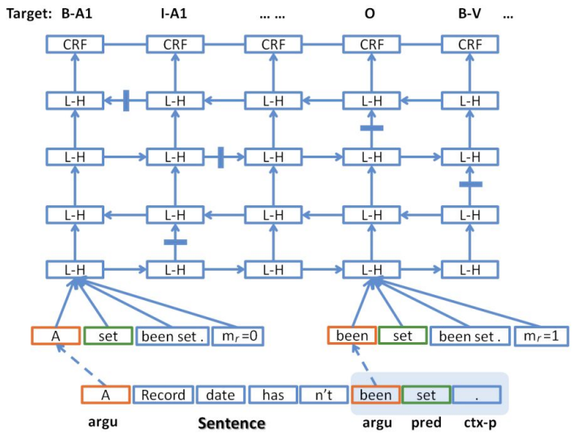
Learning to map directly to a denotation. More difficult?

- Learn from denotations (meaning of word or phrase) directly, using logical forms only as intermediate, hidden representations along the way. This is a 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 in sequence-to-sequence models: e.g., Using deep bi-directional long short-term memory (DB-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 as the input features without any intermediate tags such as syntactic information

Problem:

Needs lots of training data, hence the interest in semi-supervised models

Except for sequence relations: lack relational and nested properties which are often important in semantic parsing

Semi-supervised model:

- SEQ4: Two sequence-to-sequence models with attention chained together: consists of 4 LSTMs
    - The first, a bidirectional LSTM, encodes the sequence y; next, an LSTM with stochastic output, draws a sequence of latent distributions $\bar{x}$ over words in vocabulary x ; the third bidirectional LSTM encodes these distributions for the last one to attend over and reconstruct y as $\hat{y}$
- S2S: supervised sequence-to-sequence model: when the decoder is trained without the encoder in a fully supervised manner

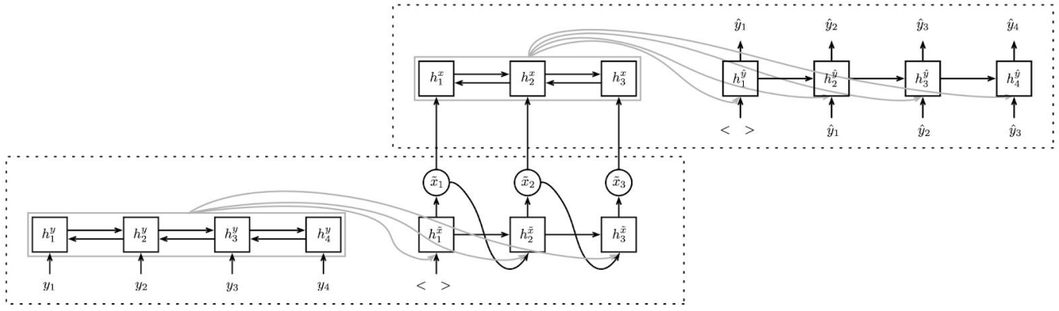


Figure 1: SEQ4 model with attention-sequence-to-sequence encoder and decoder. Circle nodes represent random variables.

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 $x = x_1, \ldots x_T$ denote a natural language expression and $y = y_1, \ldots, y_L$ its meaning representation

The conditional probability of meaning representation $P(y|x)$ as:

$$P(y|x) = P(y|x,a)P(a|x)$$

where $a = a_1 \ldots a_F$ 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 $y_{<t} = y_1 \ldots y_{t-1}$

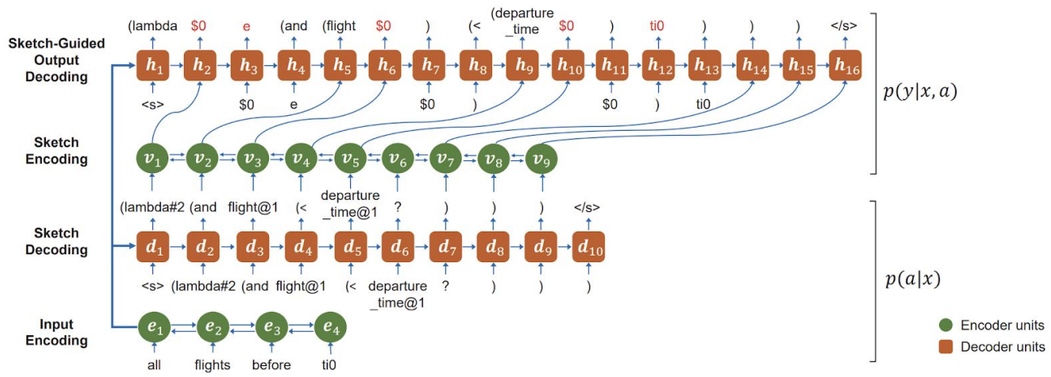


Figure 1: We first generate the meaning sketch a for natural language input x. Then, a fine meaning decoder fills in the missing details (shown in red) of meaning representation y. The coarse structure a is used to guide and constrain the output decoding.

Coarse-to-fine 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 $\{\boldsymbol{v}_k\}_{k=1}^{F}$ similarly with what is done with the input encoder, where $\boldsymbol{v}_k$ 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 states $\{\boldsymbol{h}_t\}_{t=1}^{L}$ 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, i.e., there is a one-to-one alignment between y_{t-1} and a_k the corresponding token's vector $\boldsymbol{v}_k$ 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 $\mathcal{D}$ represents the training pairs

- Prediction: At test time, the prediction for input x is obtained via $\hat{a} = \arg\max_{a'} P(a'|x)$ and $\hat{y} = \arg\max_{y'} P(y'|x,\hat{a})$ 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 (i.e., basic meaning), even though their actual meaning representations are different (e.g., 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 a global context to improve the prediction of the final details

Evaluation

- S2S is a fully supervised NN-based model; SEQ4 is semi-supervised NN-based model:
- The semi-supervised model has a small improvement (<1%) compared to the supervised learning model in the GEOQUERY dataset. It's hard to learn from unlabelled data. Better performance of SEQ4 on SAIL 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

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.

Method	GEO	ATIS
ZC07 (Zettlemoyer and Collins, 2007)	86.1	84.6
UBL (Kwiatkowksi et al., 2010)	87.9	71.4
FUBL (Kwiatkowski et al., 2011)	88.6	82.8
GUSP++ (Poon, 2013)	—	83.5
KCAZ13 (Kwiatkowski et al., 2013)	89.0	—
DCS+L (Liang et al., 2013)	87.9	—
TISP (Zhao and Huang, 2015)	88.9	84.2
SEQ2SEQ (Dong and Lapata, 2016)	84.6	84.2
SEQ2TREE (Dong and Lapata, 2016)	87.1	84.6
ASN (Rabinovich et al., 2017)	85.7	85.3
ASN+SUPATT (Rabinovich et al., 2017)	87.1	85.9
ONESTAGE	85.0	85.3
COARSE2FINE	88.2	87.7
− sketch encoder	87.1	86.9
+ oracle sketch	93.9	95.1

Table 2: Accuracies on GEO and ATIS.

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.

Semantic parsing in general:

- Successful semantic parsing approaches in limited domains eg. mapping questions to structured databases in domains of US geography, job postings, air travel 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.

Possible Questions:

1. No lambda calculation for this course
2. Semantic parsing examples - translate denotation
3. Apply SGD algorithm, produce translation, translate to denotation (no update of weights with gradient)
4. General idea what works what does not work - helpful for open questions
5. NN model is good, how and why? - which model for your semantic parsing task

Chapter 8 - Machine Translation

Why machine translation is difficult?

1. Typology of language
2. Structural divergences
3. Lexical divergences

- Language typology: Systematic cross-linguistic similarities and differences
- Universals: true for every language
- Languages have sentences that consist of words with a pronunciation and 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

<u>Word alignment</u>

- 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 Maximisation 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:
- $P(y|x) = P(x|y)P(y)$
- 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, the probability of particular

words having a particular fertility (=distribution indicating how many words a word usually translates to) etc.

IBM Model 1

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

- Translation probability:
  - for a foreign sentence $f = (f_1, \dots, f_{l_f})$ of length l_f
  - to an English sentence $e = (e_1, \dots, e_{l_e})$ of length l_e
  - with an alignment of each English word e_j to a foreign word f_i according to the alignment function a

Result 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)

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

Example of IBM 1 model here:

das

e	$t(e\|f)$
the	0.7
that	0.15
which	0.075
who	0.05
this	0.025

Haus

e	$t(e\|f)$
house	0.8
building	0.16
home	0.02
household	0.015
shell	0.005

ist

e	$t(e\|f)$
is	0.8
's	0.16
exists	0.02
has	0.015
are	0.005

klein

e	$t(e\|f)$
small	0.4
little	0.4
short	0.1
minor	0.06
petty	0.04

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

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

Learning the lexical translation probabilities

To learn the lexical translation probabilities e.g., t (e.g., Español_word|English_word):
IBM model 1 uses an Expectation Maximization (EM) algorithm:

- 1. Initialize model parameters (e.g., uniform)
- 2. Assign probabilities to latent data: probability of the alignments, compute fractional counts for the complete data (expected counts) -> Expectation step
- 3. Re-estimate the model parameters -> Maximization step
- 4. Iterate steps 2-3 until convergence

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

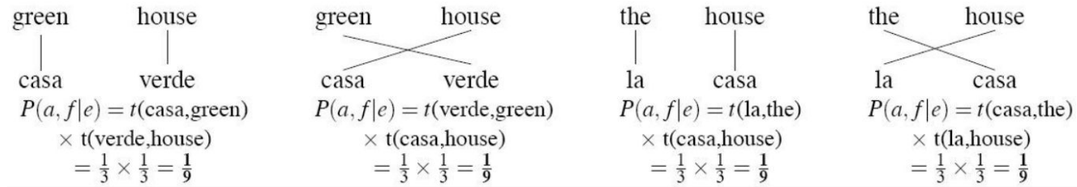
Toy example

Corpus
green house casa verde
the house la casa
Vocabulary of English language= {green, house, the}
Vocabulary of the Spanish language = {casa, verde, la}
E.g., start with uniform word pair probabilities:

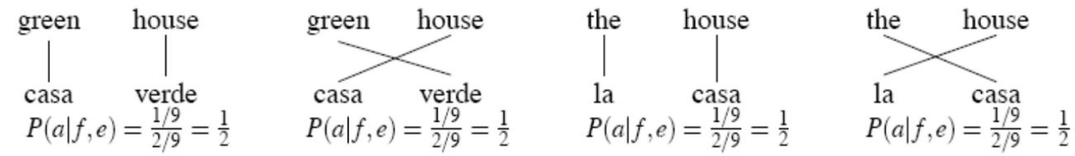
$t(casa\|green) = \frac{1}{3}$	$t(verde\|green) = \frac{1}{3}$	$t(la\|green) = \frac{1}{3}$
$t(casa\|house) = \frac{1}{3}$	$t(verde\|house) = \frac{1}{3}$	$t(la\|house) = \frac{1}{3}$
$t(casa\|the) = \frac{1}{3}$	$t(verde\|the) = \frac{1}{3}$	$t(la\|the) = \frac{1}{3}$

Expectation step:

Compute the sentence probability by multiplying the word pair probabilities



and then normalize to obtain the alignment probability



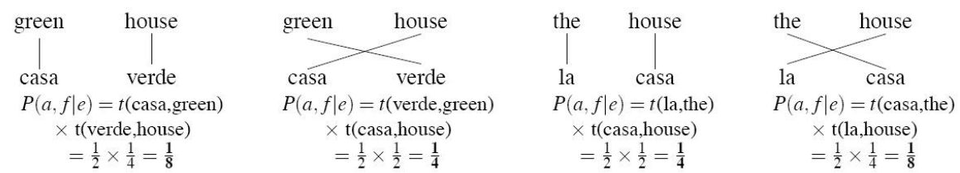
Compute fractional counts overall sentence pairs by weighting each count by $P(a|e,f)$: count x alignment probability

Maximization step

Compute maximum likelihood estimation: normalize (sum to 1)

$t(\text{casa}\|\text{green}) = \frac{1/2}{1} = \frac{1}{2}$	$t(\text{verde}\|\text{green}) = \frac{1/2}{1} = \frac{1}{2}$	$t(\text{la}\|\text{green}) = \frac{0}{1} = 0$
$t(\text{casa}\|\text{house}) = \frac{1}{2} = \frac{1}{2}$	$t(\text{verde}\|\text{house}) = \frac{1/2}{2} = \frac{1}{4}$	$t(\text{la}\|\text{house}) = \frac{1/2}{2} = \frac{1}{4}$
$t(\text{casa}\|\text{the}) = \frac{1/2}{1} = \frac{1}{2}$	$t(\text{verde}\|\text{the}) = \frac{0}{1} = 0$	$t(\text{la}\|\text{the}) = \frac{1/2}{1} = \frac{1}{2}$

Iterate using improved estimators



Higher-order IBM models

- 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 modelling of a null insertion
- IBM model 4: Uses relative reordering model: relative distortion: the placement of an input word is typically based on the placement of the translation of the proceeding input word

- IBM model 5: Fixes deficiencies

Symmetrize alignment models

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

Growing heuristics Heuristic:

- good alignment points are neighbours of other alignment points
- Starting from alignment points in the intersection, neighbouring candidate alignment points from the union are progressively added
- Popular method: **grow-diag-final**: use of diagonal neighbouring points

Phrase-based machine translation

1. Phrase based models translate phrases as atomic units:
   - When translating a foreign language sentence into an English sentence:
     - Foreign input is segmented into phrases
     - Each phrase is translated into English
     - Phrases are reordered
   - So we need to learn the probability of the translation of a phrase:
     - Compute word alignments, e.g., using an IBM model
     - Extract phrase pairs (see next slide)
     - EM training of the phrase mode

Phrase to phrase alignments

- Start from word alignments in intersection and incrementally add alignments from union (e.g., 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

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 ($y_1, y_2, \ldots, y_N$)

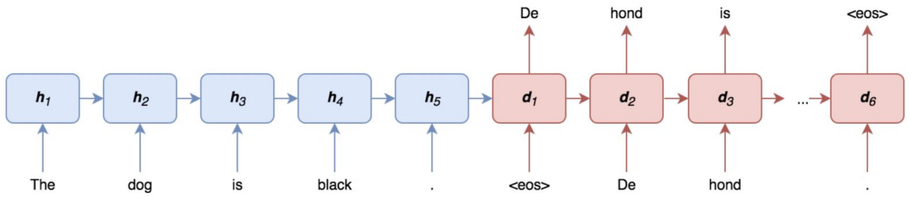
$P(y_T|x,y_1,\ldots, y_{t-1})$ is parametrized by a neural network with a deep learning end-to-end model that computes the probability of the next target word given the source sentence and the previous target words. Unlike traditional phrase based translation which consists of many small sub-components 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 o_t parametrized by a neural network with a deep learning end-to-end model
- Encoder maps a source language sequence x to one or more vectors
- Decoder predicts target language sequence y symbol by symbol using the source sequence vector representations and previously predicted symbol



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

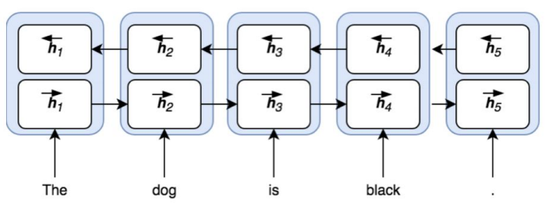
Bidirectional RNN Encoder

- Bidirectional RNN (e.g., with LSTM or GRU cells) is the most common encoder architecture
- The input is a sequence of words x, each word vector $\boldsymbol{x}$ of x is mapped into the embedding space by $\boldsymbol{Wx}$
- Bidirectional RNNs:

$$\overrightarrow{\boldsymbol{h}_i} = RNN(\overrightarrow{\boldsymbol{h}_{i-1}}, \boldsymbol{Wx})$$

$$\overleftarrow{\boldsymbol{h}_i} = RNN(\overleftarrow{\boldsymbol{h}_{i+1}}, \boldsymbol{Wx})$$

$$\boldsymbol{h}_i = (\overrightarrow{\boldsymbol{h}_i}; \overleftarrow{\boldsymbol{h}_i})$$

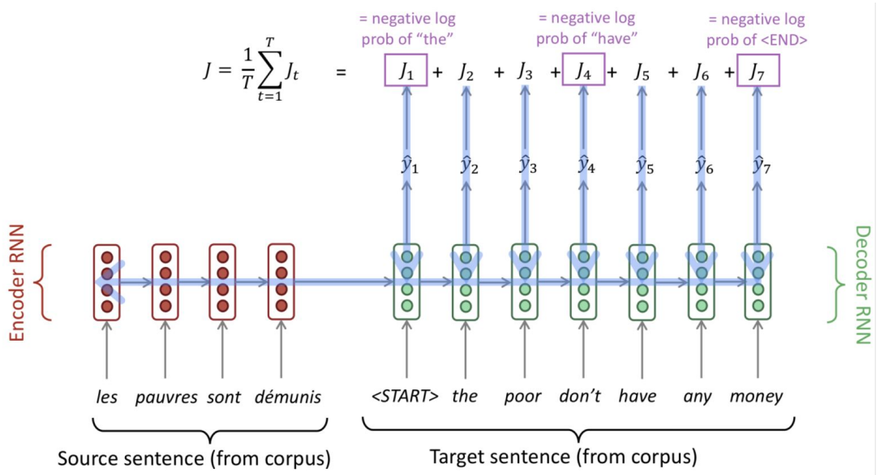


RNN Decoder (without attention)

- An RNN (eg. LSTM or GRU cells) is the most common decoder architecture
- Hidden state dt at time step t is given by: dt = RNN(dt-1, Vyt-1), where Vyt-1 = embedding of previous target word

- From hidden state dt, Score vector ot with as dimension the size of the target vocabulary is computed: ot = Udt + b
- ot is then normalized to a probability distribution with the softmax
- activation function: softmax(ot) => P(yt|x,y1, …, yt-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

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



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 $P(y_t|x,y_1,\dots,y_{t-1})$ 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?
- Strategy 1: 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)
- Strategy 2: 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
- Strategy 3: 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)

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

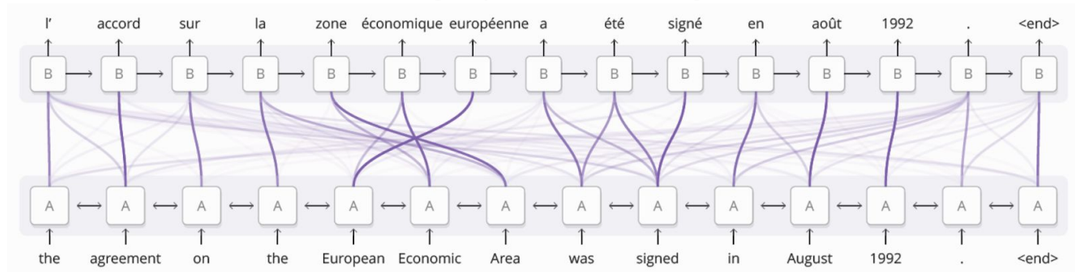
Attention is used with many different architectures. Problem of RNN encoder-decoder architecture: represents a variable length sequence with a fixed length vector

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

When humans 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

- E.g.: 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 extents ("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: $h_i = (\overrightarrow{h_i}; \overleftarrow{h_j})$
- 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 $c_t = \Sigma a_{ti} h_i$** : 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

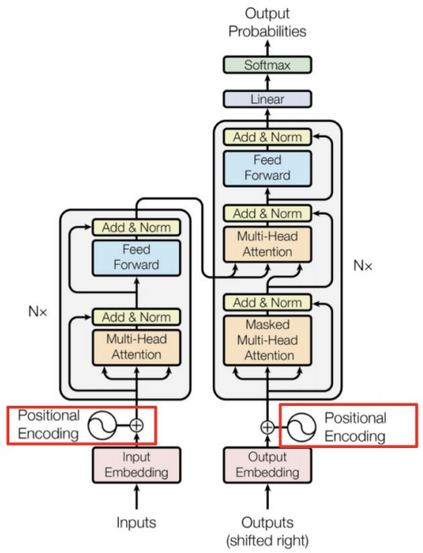
- Change RNN to something that allows you to do self-attention
  - attention to the source language in the encoder model, and the target language in the decoder model
- Transformer model not relying on RNN but only used attention to model dependencies between input and output
- RNN are replaced with:
  - 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-attention $c_t^1, c_t^2, \ldots, c_t^H$, 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
- Paper Vaswani et al. 2017: multi-head attention in the encoder, 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 e.g. using different random initializations, using dumps of earlier iterations, and combine the predictionsCNNs: 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 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 a 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 e.g. "pulling someone's leg"
- Unsupervised machine translation
- Combining SMT and NMT.

Possible Questions:
1. IBM 1 model and EM model (try one iteration)
2. Compute translation probabilities from IBM 1 model
3. Know the theory about SMT

4. Principles of how to compute sentence representation: encoding and decoding
5. Be able to explain the models different between NNs
6. Principle of attention/self-attention/multi-attention
7. No exam on CNN
8. Important on evaluations to pick the best model for open question

Chapter 9 - 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

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, e.g.
    - 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 co-referring 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.

>

coherence

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

Noun phrase coreference resolution

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.

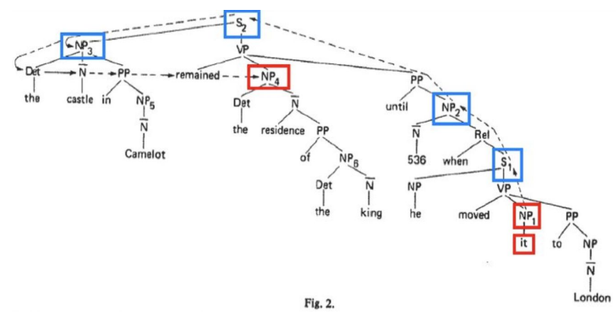


Fig. 2.

Hobbs Algorithm:
- Begin at the noun phrase (NP) node immediately dominating the pronoun.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- Go to step 4

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: e.g., 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
- $C_b(U)$ = backward looking center = entity currently being focused on in the discourse after U_n is interpreted
- The algorithm defines 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
$C_b(U_{n+1}) \neq C_p(U_{n+1})$	Retain	Rough-Shift

Algorithm:

- Generate possible C_b - C_f combinations for each possible set of reference assignments
- Filter by constraints, eg syntactic coreference constraints, selectional restrictions, centering rules and constraints
- 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 :

- John saw a beautiful 1961 Ford Falcon at the used car dealership (U_1)
- He showed it to Bob. (U_2) He bought it. (U_3)
- Bob opened a new dealership last week. (U_1) John took a look at the Fords in his lot. (U_2) He ended up buying one. (U_3)

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

- Supervised classifier if trained that locally classifies a candidate coreference relation between two entity mentions (= pairwise relation classifier) relying on number of features
- 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

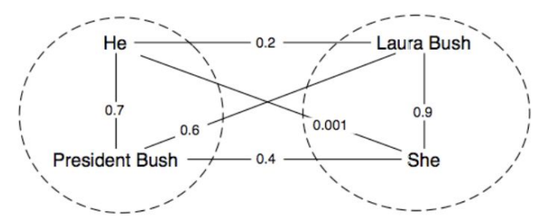
- **Input:** X is the set of pairs of noun phrases where $x_{ij} = \{m_i, m_j\}$
- **Output:** Y is the set of variables representing each pairwise coreference decision:
  - Binary random variable $y_{ij} = 1$ if m_i and m_j are coreferent, $y_{ij} = 0$ otherwise
- $P(y_{ij}|x_{ij})$: computed with a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

$$P(y_{ij}|x_{ij}) = \frac{\exp(\sum_{l=1}^{K} w_l f_l(x_{ij}, y_{ij}))}{\sum_{y_{ij} \in \{0,1\}} \exp(\sum_{l=1}^{K} w_l f_l(x_{ij}, y_{ij}))}$$

$f_l(x_{ij}, y_{ij})$ = one of the K binary-valued feature functions
w_l = real-valued weight parameter estimated from the training data

- **Training:**
  - Positive examples: pair of noun phrases that exhibit the coreference relation
  - Negative examples: pairing the noun phrases with intervening noun phrases
- Until recently hand-engineered features based on automatically produced parse trees
- [Durrett & Klein EMNLP 2013] show that 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 mj 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 post-processing 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 method e.g. 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 $y_{ij} \in \{0,1\}$
- Global optimization of the pairwise classification decisions by integer linear programming

- Add binary constraints on each of the variables: $y_{ij} \in \{0,1\}$
- 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:

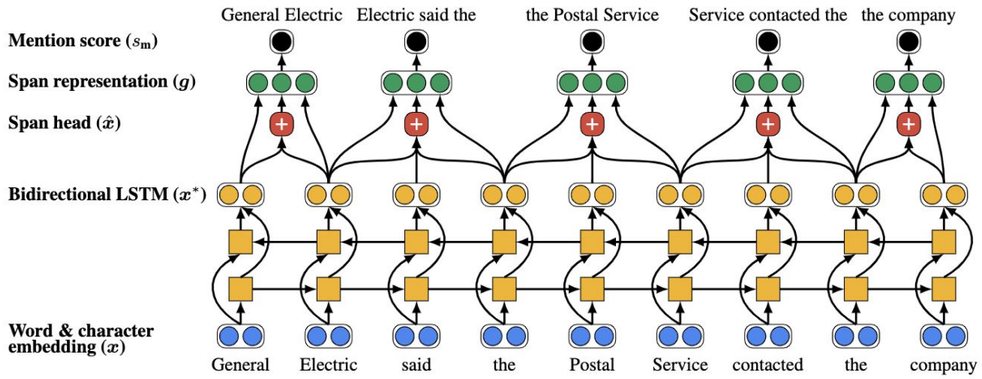
$$(1 - y_{ij}) + (1 - y_{jk}) \geq (1 - y_{ik})$$

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 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.

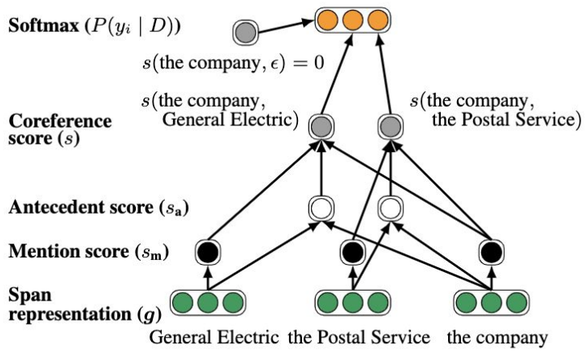
$$s_m(i) = \boldsymbol{w}_m \cdot \mathrm{FFNN}(\boldsymbol{g}_i)$$
$$s_a(i) = \boldsymbol{w}_a \cdot \mathrm{FFNN}(\boldsymbol{g}_i, \boldsymbol{g}_j, \boldsymbol{g}_i \circ \boldsymbol{g}_j, \phi(i,j))$$

where $\phi(i,j)$ = feature vector that may contain additional information (e.g., speaker and genre information, distance between two spans)

Training:
Objective: optimize the marginal log-likelihood of all correct antecedents implied by the gold clustering:

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

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}

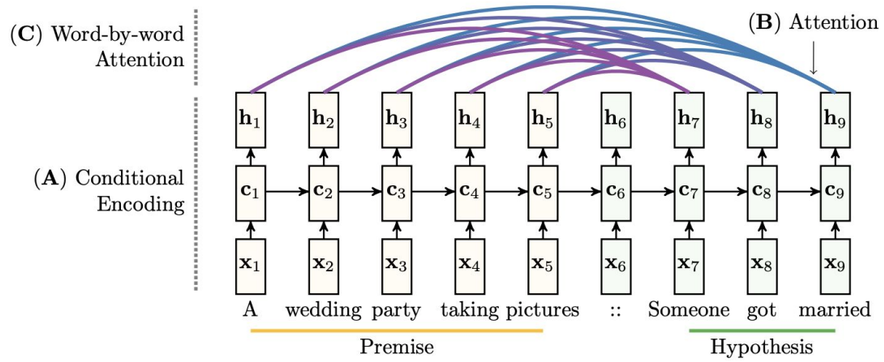


Reasoning about entailment

- Entailment : detecting the asymmetric relationship between 2 sentences: e.g.,
  - 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 language processing pipelines
- Simple bag-of-word-pairs classifiers using only lexical similarity
- Today neural networks with attention mechanisms



Recognition of rhetorical discourse structure: Evaluation

Method	Matrix Form	+Features	K	Span	Nuclearity	Relation
Prior work						
1. HILDA (Hernault *et al.*, 2010)				**83.0**	68.4	54.8
2. TSP 1-1 (Joty *et al.*, 2013)				82.47	68.43	55.73
3. TSP SW (Joty *et al.*, 2013)				82.74	68.40	55.71
Our work						
4. Basic features	$\mathbf{A} = \mathbf{0}$	Yes		79.43	67.98	52.96
5. Word embeddings	Concatenation	No	75	75.28	67.14	53.79
6. NMF	Concatenation	No	150	78.57	67.66	54.80
7. Bag-of-words	$\mathbf{A} = \mathbf{I}$	Yes		79.85	69.01	60.21
8. DPLP	Concatenation	No	60	80.91	69.39	58.96
9. DPLP	Difference	No	60	80.47	68.61	58.27
10. DPLP	Concatenation	Yes	60	82.08	**71.13**	**61.63**
11. DPLP	General	Yes	30	81.60	**70.95**	**61.75**
Human annotation				88.70	77.72	65.75

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, e.g., that “snow” is related to “outside”

But, training data are needed

Possible Questions:

1. Hobbs algorithm - important may be a question
2. Local classifier
3. Linear programming
4. Neural end to end model - principle
5. Evaluation metrics here are not examinable
6. Reasoning about entailment, which model to choose

Chapter 10 - Temporal and spatial

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:

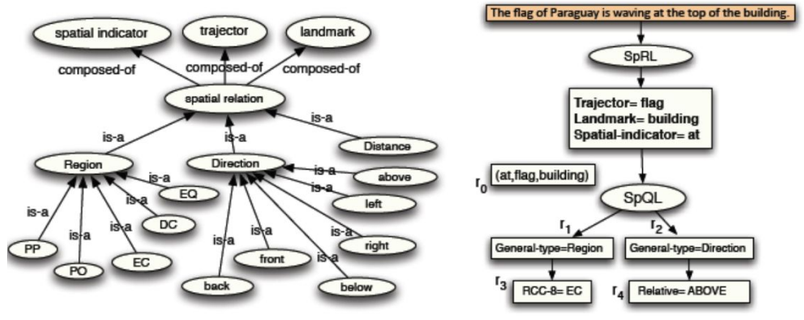


Figure 1. (a) The spatial ontology. (b) Example sentence and the recognized spatial concepts.

Annotation scheme: examples

A woman$_{TR}$ and a child$_{TR}$ are walking$_{MOTION}$ over$_{SP}$ the square$_{LM}$.

General-type: region
Specific type: RCC
Spatial value: PP (proper part)
Dynamic
Path: middle
Frame of reference: –

SemEval 2012

- *In Brazil coming from the North-East I ...* $\Longrightarrow$ ⟨Direction, [$_{Sp\ indicator}$ *In*], [$_{Trajector}$ *I*], [$_{Landmark}$ *Brazil*], [$_{Motion}$ *coming*],[$_{Path}$ *from the North-East*]⟩

SemEval 2013

- *... I stepped into the small forest and ...* $\Longrightarrow$ ⟨Direction, [$_{Trajector}$ *I*], [$_{Motion}$ *stepped into*],[$_{Path}$ *the small forest*]⟩
- *... I [...] and followed down a dried creek.* $\Longrightarrow$ ⟨Direction, [$_{Trajector}$ *I*], [$_{Motion}$ *followed down*],[$_{Path}$ *a dried creek*]⟩

SpaceEval 2015

[**Arriving**$_{m1}$] [**in**$_{ms1}$] the [**town of Juanjui**$_{pl1}$], near the [**park**$_{pl2}$], [**I**$_{se1}$] learned that my map had lied to me.

```
<MOTION id=m1 extent=``Arriving''
motion_type=PATH motion_class=REACH
motion_sense=LITERAL>
<MOTION_SIGNAL id=ms1 extent=``in''
motion_signal_type=PATH>
<PLACE id=pl1 extent=``town of
Juanjui'' form=NAM countable=TRUE
dimensionality=AREA>
<PLACE id=pl2 extent=``park'' form=NAM
countable=TRUE dimensionality=AREA>
<SPATIAL_ENTITY id=se1 extent=``I''
form=NOM countable=TRUE
dimensionality=VOLUME>
<MOVELINK id=mv11 trigger=m1
goal=pl1 mover=se1 goal_reached=TRUE
motion_signalID=ms1>
```

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

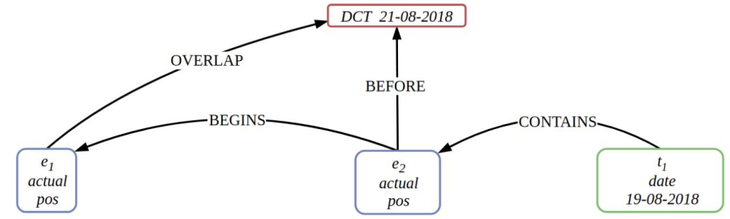
Temporal Information Extraction:
Goal: Extract the events from a text, and position them on a time-line.

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



Pain with swallowing started after insertion of Port-A-Cath two days ago.

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:

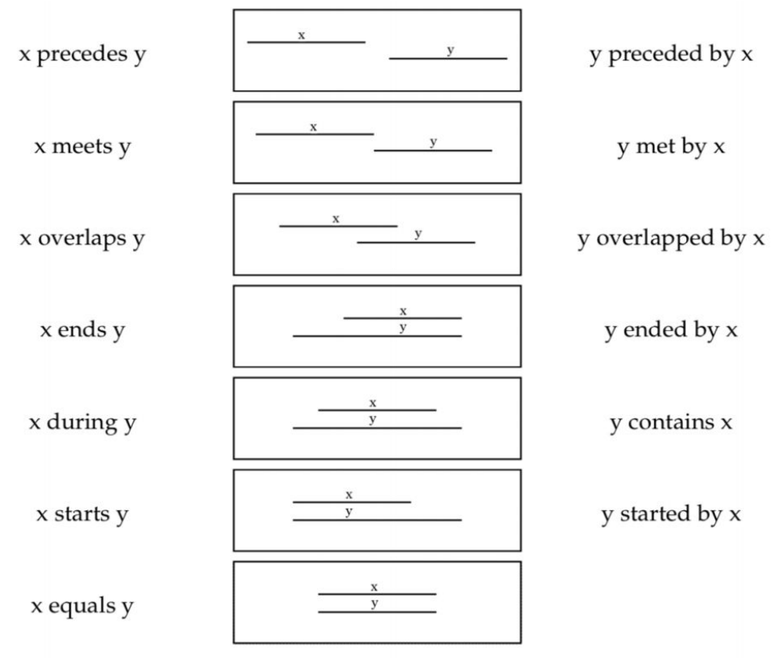


Figure 2
Allen's Thirteen Basic Interval Relations.

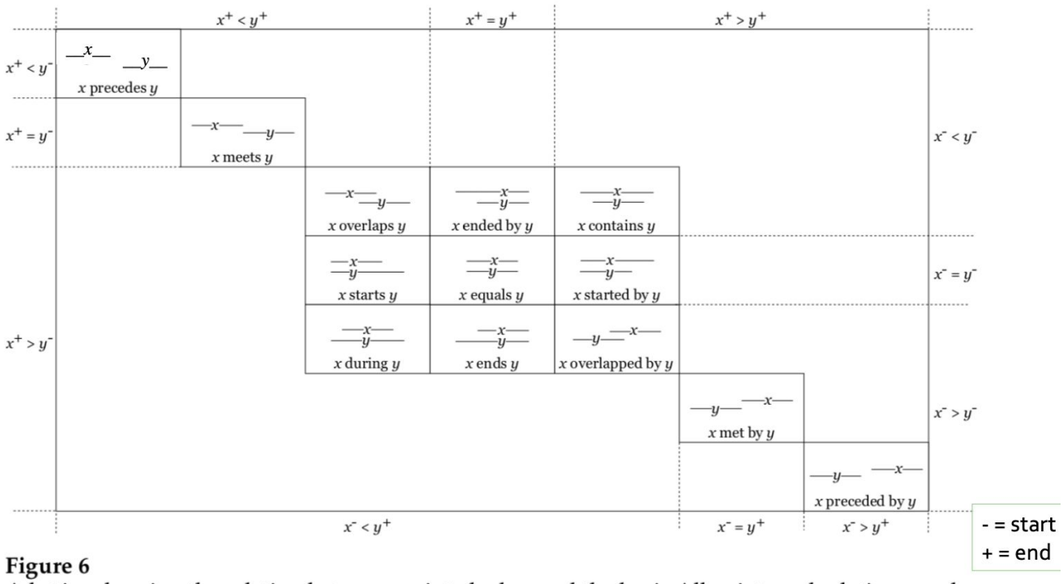


Figure 6
A lattice showing the relation between point algebra and the basic Allen interval relations, and the conceptual distance between interval relations.

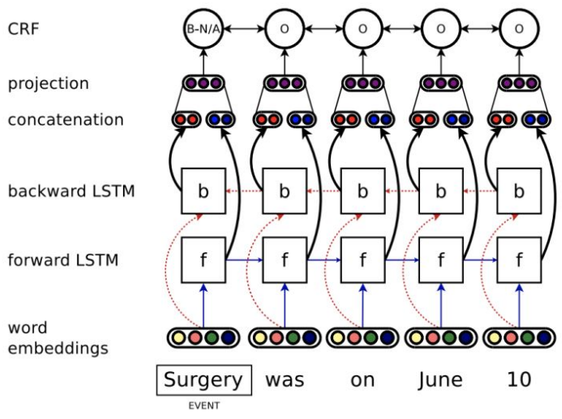
[PhD Tuur Leeuwenberg]

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:

Word	*POS*	*Label*
The	DT	O
company	NN	O
's	POS	O
sales	NN	O
force	NN	???
applauded	VB	???
the	DT	???
shake	NN	
up	PRT	I_OCCUR
.	O	O



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

Example:

Absolute	Relative	Durations
April 24, 1916	yesterday	four hours
the summer of ' 77	next semester	three weeks
10:15 AM	two weeks from yesterday	six days
the third quarter of 2006	last quarter	the last three quarters

Normalized Timex Expressions:

```
<TIMEX3
  type="DATE"
  value="2003-06-07">
June 7, 2003
</TIMEX3>
```

```
<TIMEX3
  type="DURATION"
  value="P60D"
  mod="EQUAL_OR_LESS">
no more than 60 days
</TIMEX3>
```

```
<TIMEX3
  type="TIME"
  value="1998-02-12T12:00">
noon Thursday
</TIMEX3>
```

```
<TIMEX3
  type="DATE"
  value="2000"
  mod="START">
the dawn of 2000
</TIMEX3>
```

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

Spatial relations as pairwise classification

- Features:
  - Object words, e.g. nouns
  - POS, e.g. CD and VB
  - Dependency relation between the words
  - Syntactic path, e.g. NP↑NP↓PP↓NP
  - Distance
    - # of words, number of nodes in parse tree
  - Event/Time attributes

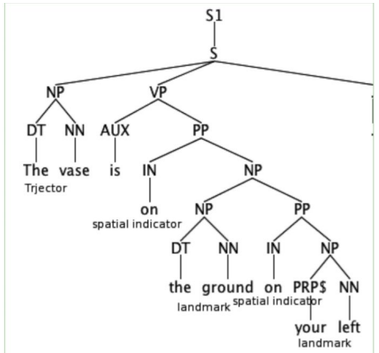


Fig. 1. Parse tree labeled with spatial roles.

Output

- Output variables = labels in the structure

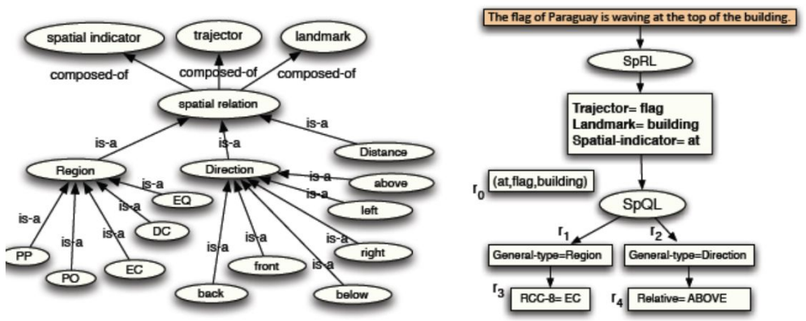


Figure 1. (a) The spatial ontology. (b) Example sentence and the recognized spatial concepts.

Structured learning

We have seen **probabilistic graphical models** (e.g., template based conditional random fields in the chapter on NER and SRL) : assignment of label structure $\hat{Y}$ given input X with maximum probability

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

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

- 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:

$$g(X, Y; \boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{f}(x, y)$$

$$\hat{Y} = \arg\max_{Y_l} g(X, Y; \boldsymbol{w})$$

$g(X, Y; \boldsymbol{w})$ 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 labelling

- Certain inputs to be labelled regarding individual words (e.g., a noun phrase headword that plays the role of trajector or landmark)
- Other inputs to be labelled regard subphrases of a sentence (e.g., a pair of noun phrase headwords connected by preposition or verb that can be labelled as spatial relation, or that can be labelled with a spatial relation attribute)
- 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 $\boldsymbol{w}_p$ resulting in the following scoring function (**L** = set van labels, C_{y_p} = set of candidate inputs/features for a particular label y_p)

$$g(X, Y; \boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{f}(x, y)$$
$$= \sum_{y_p \in \mathbf{L}} \sum_{x_k \in C_{y_p}} \boldsymbol{w}_p \cdot \boldsymbol{f}_p(x, y)$$

Training of the model

- Given N training examples: $\{(X^{(i)}, Y^{(i)})\}_{i=1}^{N}$
- 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 $Y \in \mathcal{Y}$ + some loss/cost :

$$g(X^{(i)}, Y^{(i)}; \boldsymbol{w}) \geq g(X^{(i)}, Y; \boldsymbol{w}) + \Delta(Y^{(i)}, Y), \quad \forall Y \in \mathcal{Y}$$

- A violation :

$$g(X^{(i)}, Y; \boldsymbol{w}) - g(X^{(i)}, Y^{(i)}; \boldsymbol{w}) + \Delta(Y^{(i)}, Y) > 0$$

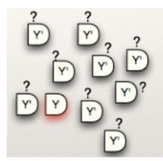
- Training = finding the weights $\boldsymbol{w}$ that minimize these violations:

$$\text{minimize}\, J(\boldsymbol{w}) = \sum_{i=1}^{N} \max_{Y \in \mathcal{Y}} \left(g(X^{(i)}, Y; \boldsymbol{w}) - g(X^{(i)}, Y^{(i)}; \boldsymbol{w}) + \Delta(Y^{(i)}, Y) \right)$$

e.g., Hamming distance between ground truth labeling and wrong labeling

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

$$\arg\max_{Y \in \mathcal{Y}} \left(g(X^{(i)}, Y; \boldsymbol{w}) - g(X^{(i)}, Y^{(i)}; \boldsymbol{w}) + \Delta(Y^{(i)}, Y) \right)$$



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 efficient solver
  - These constraints can additionally be applied during prediction

As an illustration: constraints could be formulated as ILP problem

$$sp_i + nsp_i = 1$$
$$sp_i tr_j + sp_i lm_j + sp_i nrol_j = 1$$
$$sp_i tr_j - sp_i \le 0, \quad sp_i lm_j - sp_i \le 0$$
$$sp_i - \sum_j (sp_i tr_j) \le 0, \quad sp_i - \sum_j (sp_i lm_j) \le 0$$
$$sp_i tr_j + sp_i lm_j \le 1$$
$$\sum_i (sp_i tr_j) \le 1, \quad \sum_i (sp_i lm_j) \le 1$$
$$sp_i tr_j lm_k r_{\gamma'} - sp_i tr_j lm_k r_\gamma \le 0, \quad \forall \gamma \prec \gamma' \quad \gamma, \gamma' \in \mathcal{H}$$
$$\sum_{\gamma \in \mathcal{H}_{leafs}} r_\gamma(x_i, x_j, x_k) \ge r_0(x_i, x_j, x_k)$$
$$\sum_{\gamma \in QSR_h} sp_i tr_j lm_k r_\gamma \le 1, \quad \forall h, \quad \forall QSR_h \subset \mathcal{H}_{leafs}$$

Relation extraction in the biomedical domain

- [GE] Genia Event Extraction for NFkB knowledge base
- [CG] Cancer Genetics
- [PC] Pathway Curation
- [GRO] Corpus Annotation with Gene Regulation Ontology
- [GRN] Gene Regulation Network in Bacteria
- [BB] Bacteria Biotopes (semantic annotation by an ontology)

- Challenging !

Participant	Rank	Recall	Precision	F1
	1	0.28	0.82	0.42
	2	0.36	0.46	0.40
	3	0.21	0.38	0.27
	4	0.04	0.19	0.06

Extraction of localization relations between bacteria and habitats.

Participant	Rank	Recall	Precision	F1
	1	0.12	0.18	0.14
	2	0.04	0.12	0.06
Structured SVM		0.31	0.17	0.22

Bacteria habitat categorization through the MBTO-Habitat ontology.

http://2013.bionlp-st.org/BioNLP
[Kordjamshidi et al. BMC Bioinformatics 2015]

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 hereby inference during prediction:
  - Local classifier for each label
  - Infer P(e_1, e_2, 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 labelled training data (Markov logic model)
  - Training: e.g., via Gibbs sampling

Temporal logic as feature functions

rel(t1, DCT, before)∧ rel(e1, DCT, after) ⇒ rel(e1, t1, after) [w_i=1.0]

rel(t1, DCT, before)∧ rel(e1, DCT, overlap) ⇒ rel(e1, t1, after) [w_i=0.7]

tense(e1, future) ⇒ rel(e1, DCT, after) [w_i=0.9]

tense(e1, past) ∧ tense(e2, future) ⇒ rel(e1, e2, before) [w_i=0.8]

- 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

- Local relation classification models:
  - SVM [Bethard SemEval 2013]
  - LSTM [Tourille et al. EACL 2017]
  - CNN [Dligach et al. EACL 2017]
  - ...

Each classification is independent of the others

- Structured relation classification models:
  - Structured prediction:
    - Best first greedy prediction [Bramsen et al. EMNLP 2006]
    - Integer Linear Programming (ILP) [Denis et al. IJCAI 2011]
    - ...
  - Structured training + prediction:
    - Markov logic networks [Yoshikawa et al. ACL 2009]
    - Structured perceptron [Ning et al. EMNLP 2017; Leeuwenberg et al. EACL 2017]
    - Stacked inference [Meng et al. ACL 2018]
    - ...

Classification of one pair may depend on the classifications assigned to other pairs

See survey of [Leeuwenberg & Moens JAIR 2019]

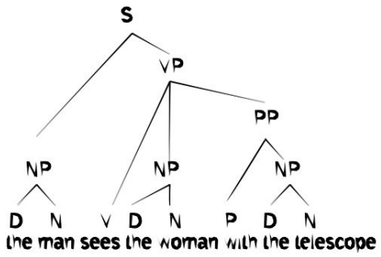
Possible Questions:

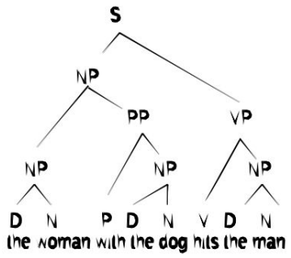
1. The different models we can compute from the output label structure
2. 9sequence label also structured labels, also tree-labels)
3. Use feature functions like CRF structure, the constraint between labels in your classifications
4. TimeML example

August 2019

1. **Context-free grammars and parsing**

Suppose you have the following toy treebank: a. From this treebank, derive a probabilistic context-free grammar (PCFG). b. Convert this PCFG to a Chomsky Normal Form (CNF). c. Parse the following sentence with this PCFG in CNF using the probabilistic CKY algorithm:





- a. From this treebank, derive a probabilistic context-free grammar (PCFG).
- b. Convert this PCFG to a Chomsky Normal Form (CNF).
- c. Parse the following sentence with this PCFG in CNF using the probabilistic CKY algorithm: "The woman sees the man"
  Give a completely filled in probabilistic CKY matrix. What is the most likely parse of the sentence?
- d. Discuss how to augment a parser to deal with input that may be incorrect, for example, containing spelling errors or mistakes arising from automatic speech recognition.

2. **Lexical semantics**

What are the possible causes of error when training the Yarowsky algorithm? Will this algorithm guarantee that an ambiguous word will be sense tagged in all examples of the training corpus? Why or why not? The algorithm needs to know the number of senses of the ambiguous word in advance, 2/3 how can you change the algorithm so that it also learns the number of senses of an ambiguous word? This bootstrapping algorithm is usually applied for word sense disambiguation, can you propose and explain another application of the algorithm?

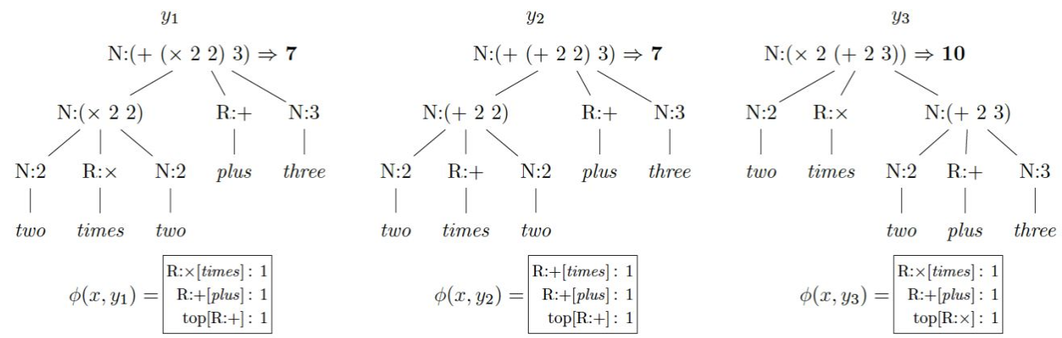
3. **Alignment in machine translation**

Given the following aligned (parallel) toy sentences in English and Dutch: the red book het rode boek the wall de muur We have some prior evidence that the translation probability of "rode" given "red" and of "boek" given "book" is each 0.6. We use the EM algorithm to compute the most probable alignments between the words of the sentences. Show one iteration of this EM algorithm.

4. **Semantic parsing**

a. We consider the case of semantic parsing where we train a classifier that maps a language utterance directly to a denotation by using the stochastic gradient descent algorithm seen in the course. The figure below shows two iterations of training this classifier. Give the details of the iterations by showing the values of the variables when applying the stochastic gradient descent algorithm (show all computations). Use the features as defined in the figure below, and the value of 1 for the learning parameter.

Candidates GEN(x) for utterance $x = \textit{two times two plus three}$



Learning from denotations

Iteration 1

$\mathbf{w} =$ R:×[times] : 0, R:+[times] : 0, R:+[plus] : 0, top[R:+] : 0, top[R:×] : 0

Scores: $[0, 0, 0]$
$\text{GEN}(x, d) = \{y_1, y_2\}$
$y = y_1$ (tied with y_2)
$\tilde{y} = y_3$ ~~(tied with y_2)~~

$\Rightarrow$

Iteration 2

$\mathbf{w} =$ R:×[times] : 0, R:+[times] : 0, R:+[plus] : 0, top[R:+] : 1, top[R:×] : -1

Scores: $[1, 1, -1]$
$\text{GEN}(x, d) = \{y_1, y_2\}$
$y = y_1$ (tied with y_2)
$\tilde{y} = y_1$ (tied with y_2)

b. What are the advantages of mapping to denotations? List two applications of mapping to denotations, other than the semantic parsing of arithmetic expressions formulated in natural language.

5. **Design**: Chatbot Design a small air-travel help system based on text input. Your system should get constraints from users about a particular flight that they want to take, expressed in natural language, and display possible flights on a screen. You may use in a simple flight database or you may use a flight information system on the Web as your backend. How would you evaluate the system?

January 2018

1. Discuss RBMT, PFT and NMT. Compare the advantages and disadvantages of these three methods.

2. Given two syntax trees
    a. Write down the PCFG for these trees.
    b. Parse the sentence "Time flies like a banana" using Earley Parsing.
    c. Show the resulting parse tree(s) and their probability.
    d. Discuss the results.
    e. How could we improve this parsing-method?

3. Give two phrases which were tagged using B-I-O tags combined with argument tags (as seen in the semantic role labelling presentation)
    a. Explain what these tags mean.
    b. If you're using MaxEnt, design and explain 3 feature functions.
    c. If you were to model this using integer linear programming, design 2 constraints and explain.
    d. Assume you're using a conditional random field. Design one extra feature function that ensures the transitions between labels.

4. Theory questions
    a. I forgot the first one.
    b. Compare classic N-grams with a neural network approach. How do both deal with unseen words?

5. Design question: you have to design a query answering system that, given a question, will query a relational database and return the results. You had to explain this in sufficient detail and explain all the methods that you would use.

January 2015

1. Given three phrases.

a) Tag all the phrases using the Penn treebank POS tags.
b) Compute the HMM transition probability matrix (do not apply smoothing).
c) Compute the HMM emission probability matrix (no smoothing).
d) Given a new phrase: compute the bigram probabilities for this phrase.
e) Discuss the results of d). What would happen if we applied smoothing?

2. Given two phrases:

Fruit flies like a ripe banana. Time flies like an arrow.

a) Write down the PCFG for this corpus.
b) Convert it to CNF.
c) Apply CKY parsing.
d) Show the resulting parse trees and their probabilities.
e) Discuss the results.

3. Given a hidden Markov model and a series of observations, compute the most likely joint state of the HMM.

4. Given three text snippets, compute the association of two pairs of words using pointwise mutual information association measure and the Lin association measure.

5. Question 5, January 2015.

6. You want to build a system which finds companies, brands, names, etc. which are mentioned on web pages. How would you do this? Go step by step, in enough detail, using the techniques mentioned in the course. Mention the bottlenecks involved in the process.

January 2015

1) Given an FSA, give the matching regular expression and give 10 strings that can be generated by the FSA. The FSA was like this:

```
q_0 -> q_1 <-> f <- q_2
       |            ^
       |            |
        - - - - - -

with q_0 -> q_1 = a
   q_1 -> f = b
   q_1 -> q_2 = b
   q_2 -> f = a
   f -> q_1 = a
```

2) You get 2 parse-trees of the following (simple) sentences: "time flies like an arrow" and "time flies like a banana".

- a) create the corresponding PCFG
- b) convert the PCFG to CNF (Chomsky normal form)
- c) perform CKY algorithm on the sentence "Time flies like a banana"
- d) draw all possible parse trees and give their probability

3) Given 5 sentences (sentence 1 is the same as sentence 3 and sentence 2 is the same as sentence

4). The word "anybody" didn't appear in the 5 sentences, the word "nobody" appeared 3 times.

- a) Making use of bigrams, compute the probability of the word "nobody" and the word "anybody", using maximum likelihood estimation MLE, Laplace smoothing and Good Turing smoothing.
- b) Compare every method and give their advantages and disadvantages.

4) Theoretical question: the question handled "pairwise coreference classifier" (discourse analysis) and had 2 subquestions:

- a) how are "coreference chains" formed, based on the result of the classifier? (not completely sure about the exact question but it was something like this)
- b) give 5 features (seen during the lectures) which can be used for this classifier

5) Theoretical question: given the slide of the last lecture about semantic parsing. On these slides, there is an example execution of "stochastic gradient descent", an optimizing algorithm. Explain the different iterations of the examples and explain how the algorithm computes, derives every value of every parameter.

June 2009

1) Same as June 2015 but without the part "give 10 strings that can be generated by the FSA".

2) Same as June 2015.

3) What is "rhetorical structure theory"? Give relevant examples, explanation and applications.

4) Given 3 English sentences.

a) compute the similarity between 'flight' & 'debris' and 'flight' & 'Airbus' using the lin assoc method

b) in which application can these similarities be used?

5) Theoretical question...can’t quite remember

6) Suppose you have an e-mail management program that consists of two parts:

- Task manager: gets all the tasks from the incoming emails

- Time manager: saves the overview of the appointments and reminds you of deadlines of tasks

How would you do this?