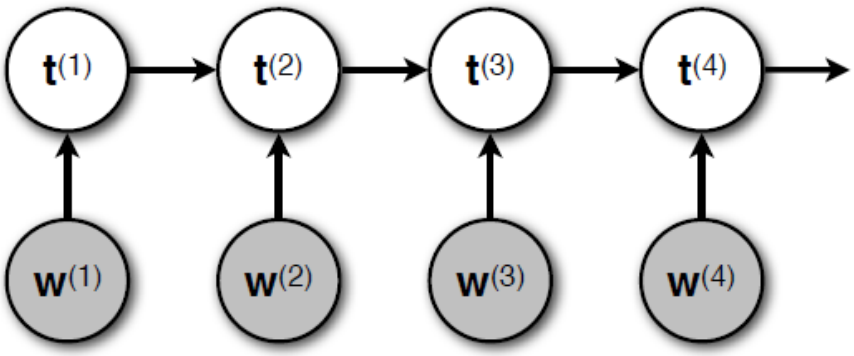


Probabilistic classification: Model the probability P(c—x), where P(c—x) is a probability if $0 \leq P(c_i|x) \leq 1$, and $\sum_i P(c_i|x) = 1$. Return the class $c^* = \arg\max_i P(c_i|x)$ that has the highest probability. One standard way to model P(c—x) is logistic regression (used by MEMMs and CRFs).

Using features: Think of **feature functions** as useful question you can ask about the input x:

- Binary feature functions:
  - $f_{first-letter-capitalized}(Urbana) = 1$
  - $f_{first-letter-capitalized}(computer) = 0$
- Integer (or real-valued) features:
  - $f_{number-of-vowels}(Urbana) = 3$

Which specific feature functions are useful will depend on your task (and your training data).

5.24 From features to probabilities

We associate a $real-valued weight w_{ic}$ with each feature function $f_i(x)$ and output class c. Note, that the feature function $f_i(x)$ does not have to depend on c as long as the weight does (note the double index w_{ic}). This gives us a real-valued score for predicting class c for input x: $score(x,c) = \sum_i w_{ic} f_i(x)$. This score could be negative, so we exponentiate it: $score(x,c) = exp(\sum_i w_{ic} f_i(x))$. To get a probability distribution over all classes c, we re-normalize these scores: $P(c|x) = score(x,c)/\sum_j score(x,c_j)$.

Learning: finding w We use conditional maximum likelihood estimation (and standard convex optimization algorithms). The conditional MLE training objective: Find the w that assigns highest probability to all observed outputs c_i given the inputs x_i.

$$\hat{w} = \arg\max_w \prod_i P(c_i|x_i, w) \tag{5.27}$$

Terminology Models that are of the form:

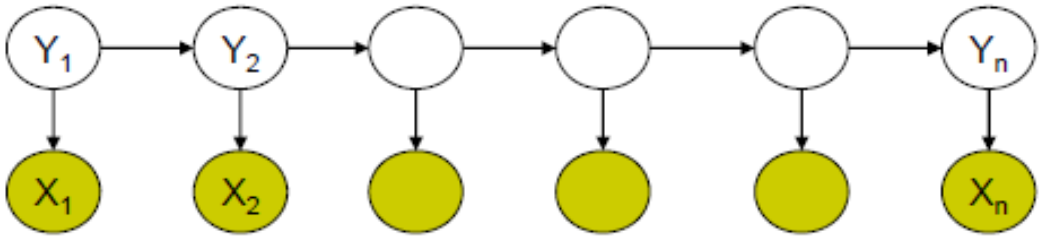
$$P(c|x) = \frac{score(x,c)}{\sum_j score(x,c_j)} = \frac{exp(\sum_i w_{ic} f_i(x))}{\sum_j exp(\sum_i w_{ij} f_i(x))} \tag{5.28}$$

are also called **loglinear** models, Maximum Entropy (**MaxEnt**) models, or **multinomial logistic regression** models. The normalizing term $\sum_j exp(\sum_i w_{ij} f_i(x))$ is also called **partition function** and is often abbreviated as **Z**.

Note, all these terms are the same:

- Softmax
- Softmax regression
- Multinomial (or multiclass) logistic regression
- Multinomial logit
- Monditional maximum entropy model

5.25 Shortcomings of Hidden Markov Model(1): locality of features



HMM models capture dependences between each state and only its corresponding observation. NLP example: in a sentence segmentation task, each segmental state may depend not just on a single word (and adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, identiation, amount of white space, etc.

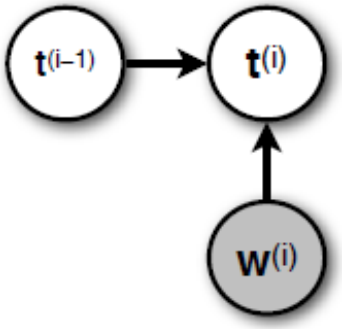
Mismatch between learning objective function and prediction objective function. HMM learns a joint distribution of states and observations $P(Y,X)$, but in a prediction task, we need the conditional probability $P(Y|X)$.

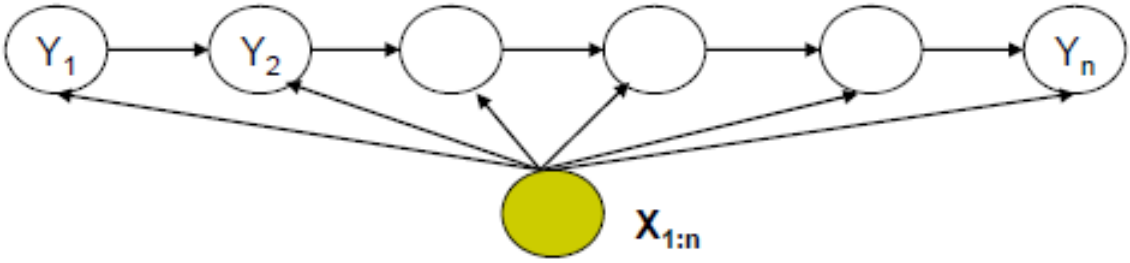
5.26 MEMM

Solution is **Maximum Entropy Markov Model (MEMM)**. It is s sequence version of the logistic regression (also called maximum entropy) classifier. Task is to find the best series of

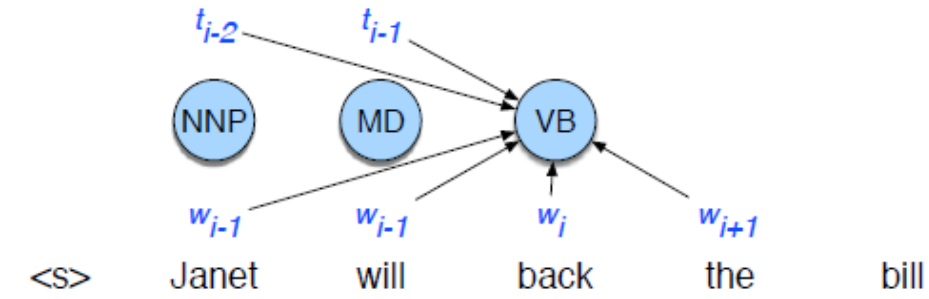
tags:

$$\begin{aligned}\hat{T} &= \arg\max_T P(T|W) \\ &= \arg\max_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned} \tag{5.29}$$

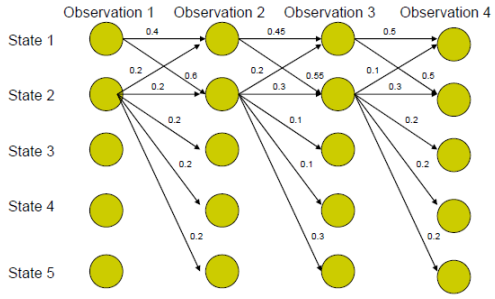




$$P(y_{1:n}|x_{1:n}) = \prod_{i=1}^{n} P(y_i|y_{i-1}, x_{1:n}) = \prod_{i=1}^{n} \frac{exp(w^T f(y_i, y_{i-1}, x_{1:n}))}{Z(y_{i-1}, x_{1:n})} \tag{5.30}$$

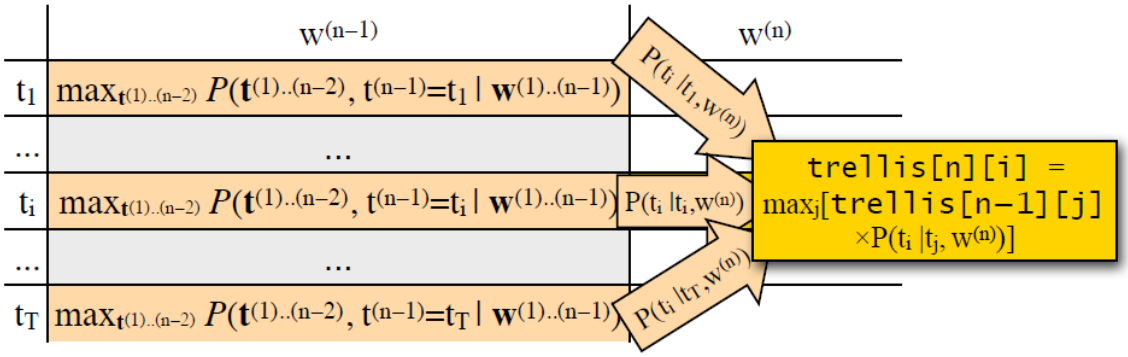


Models dependence between each state and the **full observation** sequence explicitly. It is more expressive than HMMs. MEMM is a discriminative model - it completely ignores modelling P(x): saves modelling effort; and it's learning objective function is consistent with predictive function: $P(Y|X)$.



Viterbi for MEMMs trellis[n][i] stores the probability of the most likely (Viterbi) tag sequence $t^1...t^n$ that ends in tag t_i for the prefix $w^1...w^n$. Remember that we do not generate w in MEMMs. So:

$$\begin{aligned} trellis[n][i] &= \max_{t^1...t^{n-1}} [P(t^1...t^{n-1}, t^n = t_i | w^i...w^n)] \\ &= \max_j [trellis[n-1][j] * P(t_i|t_j, w^n)] \\ &= \max_j [\max_{t^1...t^{n-1}} [P(t^1...t^{n-2}, t^{n-1} = t_j | w^1...w^{n-1})] * P(t_i|t_j, w^n)] \end{aligned} \tag{5.31}$$



Example of features :

- w_i contains a particular prefix (from all prefixes of length ≤ 4)
- w_i contains a particular suffix (from all suffixes of length ≤ 4)
- w_i contains a number
- w_i contains an upper-case letter
- w_i contains a hyphen
- w_i is all upper case
- w_i's words shape

- w_i's short word shape
- w_i is upper case and has a digit and a dash (like CFC-12)
- w_i is upper case and followed within 3 words by Co., Inc., etc.

MEMM decoding Simplest algorithm:

Algorithm 1 function Greedy MEMM Decoding(words, W, model P) returns tag sequence T

for i=1 to length(W): **do**
end for

What we use in practice: The Viterbi algorithm - a version of the same dynamic programming algorithm we used to compute minimum edit distance.

The label bias problem. What the local transition probabilities say:

- State 1 almost always prefer to go to state 2
- State 2 almost prefer to stay in state 2

The probability path $1 \rightarrow 1 \rightarrow 1 \rightarrow 1$ is $0.4 * 0.45 * 0.5 = 0.09$.

The probability path $2 \rightarrow 2 \rightarrow 2 \rightarrow 2$ is $0.2 * 0.3 * 0.3 = 0.018$.

The probability path $1 \rightarrow 2 \rightarrow 1 \rightarrow 2$ is $0.6 * 0.2 * 0.5 = 0.06$.

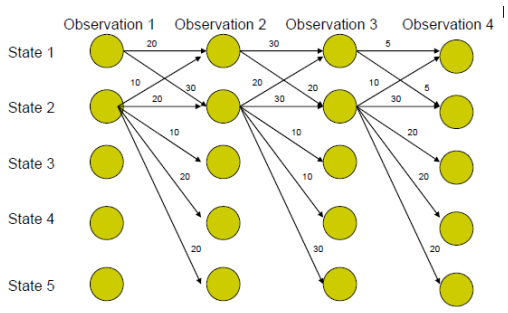
The probability path $1 \rightarrow 1 \rightarrow 2 \rightarrow 2$ is $0.4 * 0.55 * 0.3 = 0.066$.

The probability path $1 \rightarrow 2 \rightarrow 1 \rightarrow 2$ is $0.6 * 0.2 * 0.5 = 0.06$.

So, most likely path is $1 \rightarrow 1 \rightarrow 1 \rightarrow 1$. Although **locally** it seems that state 1 wants to go to state 2 and state 2 wants to remain in state 2. Why? State 1 has only two transitions but state 2 has 5. Plus average transition probability from state 2 is lower.

To sum up the problem of label bias in MEMM is: preference of states with lower number of transitions over others,

Solution: do not normalize probabilities locally. Use local potentials and states with lower transitions do not have an unfair advantage:



5.27 From MEMM to CRF

$$P(y_{1:n}|x_{1:n}) = \frac{1}{Z(x_{1:n})}\prod_{i=1}^{n}\phi(y_i, y_{i-1}, x_{1:n}) = \frac{1}{Z(x_{1:n}, w)}\prod_{i=1}^{n}\exp(w^T f(y_i, y_{i-1}, x_{1:n})) \tag{5.32}$$

Conditional Random Field (CRF) is particularly directed model. It is discriminative model like MEMM, it uses global normalizer Z(x) that overcomes the label bias problem of MEMM. And CRf models the dependence between each state and the entire observation sequence (like MEMM).

$$\begin{aligned} P(y|x) &= \frac{1}{Z(x,\lambda,\mu)}exp(\sum_{i=1}^{n}(\sum_{k}\lambda_k f_k(y_i, y_{i-1}, x) + \sum_{l}\mu_l g_l(y_i, x))) \\ &= \frac{1}{Z(x,\lambda,\mu)}exp(\sum_{i=1}^{n}(\lambda^T f(y_i, y_{i-1}, x) + \mu^T g(y_i, x))) \end{aligned} \tag{5.33}$$

$$\text{where} \quad Z(x,\lambda,\mu) = \sum_{y} exp(\sum_{i=1}^{n}(\lambda^T f(y_i, y_{i-1}, x) + \mu^T g(y_i, x))) \tag{5.34}$$

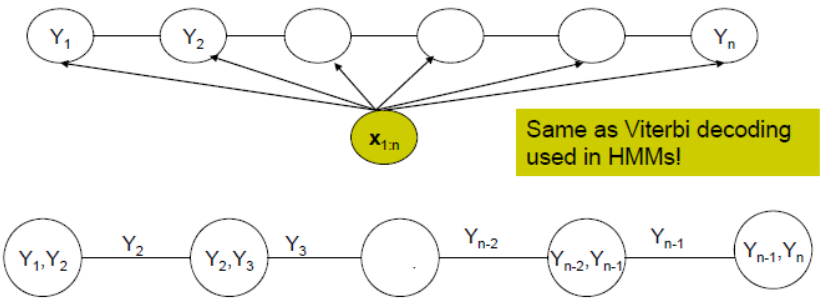
5.28 CRF: Inference

Given CRF parameters λ and μ, find the y^* that maximizes $P(y|x)$:

$$y^* = \arg\max_{y} exp(\sum_{i=1}^{n}(\lambda^T f(y_i, y_{i-1}, x) + \mu^T g(y_i, x))) \tag{5.35}$$

Here we can ignore Z(x) because it is not a function of y.

Run the max-product algorithm on the junction-tree of CRF:



5.29 CRF learning

Given $\{(x_d, y_d)\}_{d=1}^N$, find λ^*, μ^* suh that:

$$\begin{aligned}\lambda^*, \mu^* &= \arg\max_{\lambda,\mu} L(\lambda,\mu) = \arg\max_{\lambda,\mu} \prod_{d=1}^{N} P(y_d|x_d,\lambda,\mu) \\ &= \arg\max_{\lambda,\mu} \prod_{d=1}^{N} \frac{1}{Z(x_d,\lambda,\mu)} exp(\sum_{i=1}^{n} (\lambda^T f(y_{d,i}, y_{d,i-1}, x_d) + \mu^T g(y_{d,i}, x_d))) \\ &= \arg\max_{\lambda,\mu} \sum_{d=1}^{N} (\sum_{i=1}^{n} (\lambda^T f(y_{d,i}, y_{d,i-1}, x_d) + \mu^T g(y_{d,i}, x_d)) - \log Z(x_d,\lambda,\mu))\end{aligned} \tag{5.36}$$

Computing the gradient w.r.t λ:

$$\nabla_\lambda L(\lambda,\mu) = \sum_{d=1}^{N} (\sum_{i=1}^{n} f(y_{d,i}, y_{d,i-1}, x_d) - \sum_{y} (P(y|x_d) \sum_{i=1}^{n} f(y_{d,i}, y_{d,i-1}, x_d))) \tag{5.37}$$

*Gradient of the log-partition function in an exponential family is the expectation of the sufficient statistics.

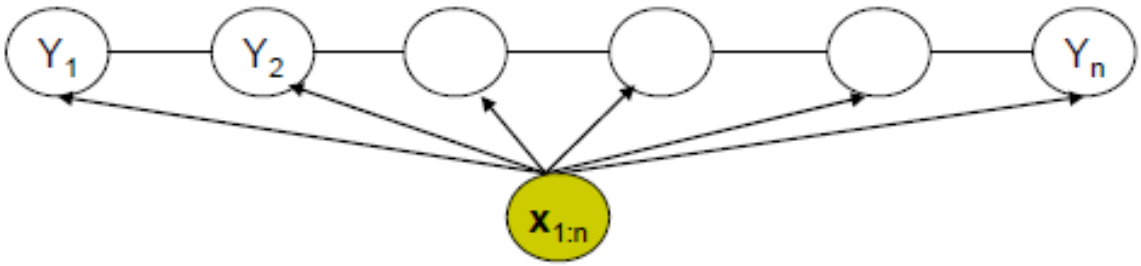
$\sum_y (P(y|x_d) \sum_{i=1}^n f(y_{d,i}, y_{d,i-1}, x_d))$ requires exponentially large number of summations: is it intractable. Is it intractable?

$$\begin{aligned}\sum_{y} (P(y|x_d) \sum_{i=1}^{n} f(y_{d,i}, y_{d,i-1}, x_d)) &= \sum_{i=1}^{n} (\sum_{y} f(y_{d,i}, y_{d,i-1}, x_d) P(y|x_d)) \\ &= \sum_{i=1}^{n} \sum_{y_i, y_{i-1}} f(y_i, y_{i-1}, x_d) P(y_i, y_{i-1}|x_d))\end{aligned} \tag{5.38}$$

* Expectation of **f** in $\sum_{i=1}^n \sum_{y_i,y_{i-1}} f(y_i, y_{i-1}, x_d) P(y_i, y_{i-1}|x_d))$ over the corresponding marginal probability of neighboring nodes!!

So, it is tractable! We can compute marginals using the sum-product algorithm on the chain.

Computing marginals using junction-tree calibration:



Junction tree initialization: $\alpha^0(y_i, y_{i-1}) = exp(\lambda^T f(y_i, y_{i-1}, x_d) + \mu^T g(y_i, x))$:

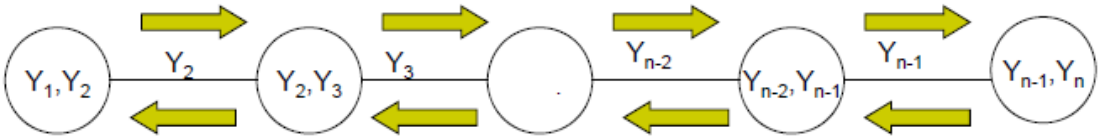


Figure 5.12: *
Also called forward-backward algorithm

After calibration:

$$P(y_i, y_{i-1}|x_d) \propto \alpha(y_i, y_{i-1}) \tag{5.39}$$

$$P(y_i, y_{i-1}|x_d) = \frac{\alpha(y_i, y_{i-1})}{\sum_{y_i, y_{i-1}} \alpha(y_i, y_{i-1})} = \alpha'(y_i, y_{i-1}) \tag{5.40}$$

Computing feature expectations using calibrated potentials:

$$\sum_{y_i, y_{i-1}} f(y_i, y_{i-1}, x_d) P(y_i, y_{i-1}|x_d) = \sum_{y_i, y_{i-1}} f(y_i, y_{i-1}, x_d) \alpha'(y_i, y_{i-1}) \tag{5.41}$$

Now we know how to compute $r_\lambda L(\lambda, \mu)$:

$$\begin{aligned} \nabla_\lambda L(\lambda, \mu) &= \sum_{d=1}^{N} (\sum_{i=1}^{n} f(y_{d,i}, y_{d,i-1}, x_d) - \sum_{y} (P(y|x_d) \sum_{i=1}^{n} f(y_{d,i}, y_{d,i-1}, x_d))) \\ &= \sum_{d=1}^{N} (\sum_{i=1}^{n} (f(y_{d,i}, y_{d,i-1}, x_d) - \sum_{y_i, y_{i-1}} \alpha'(y_i, y_{i-1}) f(y_{d,i}, y_{d,i-1}, x_d))) \end{aligned} \tag{5.42}$$

Learning can now be done using gradient ascent:

$$\lambda^{t+1} = \lambda^t + \eta \nabla_\lambda L(\lambda^{(t)}, \mu^{(t)}) \tag{5.43}$$

$$\mu^{t+1} = \mu^t + \eta \nabla_\mu L(\lambda^{(t)}, \mu^{(t)}) \tag{5.44}$$

In practice, we use a Gaussian Regularizer for the parameter vector to improve generalizability:

$$\lambda_*, \mu_* = \arg\max_{\lambda, \mu} \sum_{d=1}^{N} \log P(y_d|x_d, \lambda, \mu) - \frac{1}{2\sigma^2}(\lambda^T \lambda + \mu^T \mu) \tag{5.45}$$

In practice, gradient ascent has very slow convergence. Alternatives are:

- Conjugate Gradient method
- Limited Memory Quasi-Newton Methods

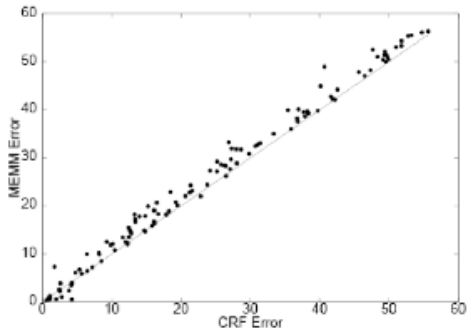


Figure 5.13: *
MEMM error (CRF error)

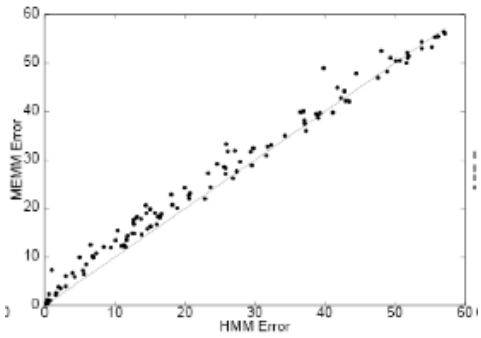
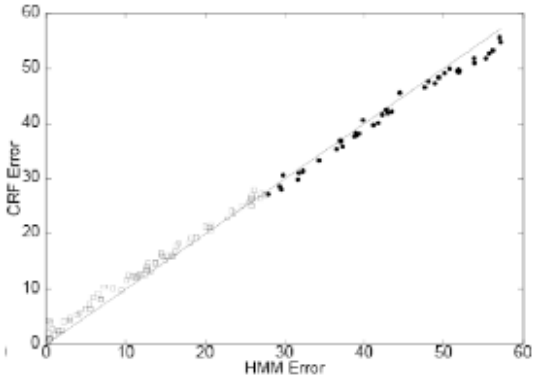


Figure 5.14: *
MEMM error (HMM error)

5.30 CRFs: some empirical results



Comparison of error rates on synthetic data *CRFs achieve the lowest error rate for higher order data. Higher order data here is in domain of higher error rates.

model	*error*	*oov error*
HMM	5.69%	45.99%
MEMM	6.37%	54.61%
CRF	5.55%	48.05%
MEMM$^+$	4.81%	26.99%
CRF$^+$	4.27%	23.76%

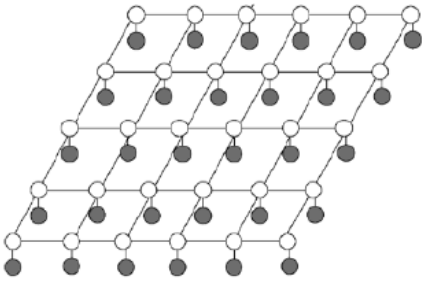
$^+$Using spelling features

Parts of speech tagging

- Using name set of features: $HMM >=< CRF > MEMM$
- Using additional overlapping features: $CRF^+ > MEMM^+ >> HMM$

5.31 Other CRFs

So far we have discussed only 1-dimensional chain CRFs: inference and learning. We could also have CRFs for arbitrary graph structure. E.g. Grid CRFs. Inference and learning are no longer tractable. Approximate techniques used: MCMC Sampling, Variational Inference, Loopy Belief Propagation.



6. Syntax Parsing

In this chapter we discuss the basics of neural networks. Where they come from, how to build and optimize them. We use text classification as exemplary task.

6.1 Grammars and syntax parsing

Grammar	Languages	Production Rules	Examples
Type 0	Recursively Enumerable	$\alpha A \beta \rightarrow \delta$	
Type 1	Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	$L = \{ a^n b^n c^n \mid n > 0 \}$
Type 2	Context Free	$A \rightarrow \alpha$	$L = \{ a^n b^n \mid n > 0 \}$
Type 3	Regular	$A \rightarrow a$ or $A \rightarrow aB$	$L = \{ a^n \mid n > 0 \}$

6.1.1 Desirable Properties of a Grammar

Chomsky specified two properties that make a grammar "interesting and satisfying":

It should be a finite specification of the strings of the language, rather than a list of its sentences.

It should be revealing, in allowing strings to be associated with meaning (semantics) in a systematic way.

We can add another desirable property:

It should be revealing, in allowing strings to be associated with meaning (semantics) in a systematic way.

Context-free grammars (CFGs) provide a pretty good approximation.

Some features of NLs are more easily captured using mildly context-sensitive grammars, as well see later in the course.

There are also more modern grammar formalisms that better capture structural and distributional properties of human languages. (E.g. combinatory categorial grammar.)

Programming language grammars (such as the ones used with compilers, like LL(1)) aren't enough for NLs.

Grammatical rules	**Lexical rules**
S → *NP VP*	Det → a — the — her **(determiners)**
NP → *Det N*	N → man — park — duck — telescope **(nouns)**
NP → *Det N PP*	Pro → you **(pronoun)**
NP → *Pro*	V → saw — walked — made **(verbs)**
VP → *V NP PP*	Prep → in — with — for **(prepositions)**
VP → *V NP*	
VP → V	
PP→ *Prep NP*	

6.1.2 Grammar for the Tiny Fragment of English

Grammar G1 generates the sentences of the previous sentences:

6.1.3 Recursion

Recursion in a grammar makes it possible to generate an infinite number of sentences.
In **blue**direct recursion, a non-terminal on the LHS of a rule also appears on its RHS. The following rules add direct recursion to G1:

VP → VP Conj VP
Conj → and—or

In **indirect recursion**, some non-terminal can be expanded (via several steps) to a sequence of symbols containing that non-terminal:

NP → Det N PP
PP → Prep NP

6.1.4 Contex-free grammars: formal definition

A **(contex-free grammar)** (CFG) ζ consists of

- a finite set N of **non-terminals**,
- a finite set $\sum$ of **terminals**, disjoint from N,
- a finite set P of **productions** of the form $X \to \alpha$, where $X \in N$, $\alpha \in (N \cup \sum)^*$,
- a choice of **start symbol** $S \in N$.

A sentential form is any sequence of terminals and nonterminals that can appear in a derivation starting from the start symbol.
Formal definition: The set of **sentential forms** derivable from ζ is the smallest set $S(\zeta) \subseteq (N \cup \sum)^*$ such that

- $S \in S(\zeta)$
- if $\alpha X \beta \in X \to \gamma \in P$, then $\alpha\gamma\beta \in S(\zeta)$.

The **language** associated with grammar is the set of sentential forms that contain only terminals.
Formal definition: The **language** associated with ζ is defined by $\mathcal{L}(\zeta) = S(\zeta) \cap \sum^*$

A language $L \subseteq \sum^*$ is defined to be *context-free* if there exists some CFG ζ such that L = $\mathcal{L}(\zeta)$.

6.1.5 Assorted remarks

- $X \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$ is simply an abbreviation for a bunch of productions $X \rightarrow \alpha_1, X \rightarrow \alpha_2, \ldots, X \rightarrow \alpha_n$.
- These grammars are called *context-free* because a rule $X \rightarrow \alpha$ says that an X can always be expanded to α, no matter where the X occurs.
  This contrasts with *context-sensitive* rules, which might allow us to expand X only in certain contexts, e.g. $bXc \rightarrow b\alpha c$.
- Broad intuition: context-free languages allow nesting of structures to arbitrary depth. E.g. brackets, begin-end blocks, if-then-else statements, subordinate clauses in English, …

6.1.6 Parsing: where are we now?

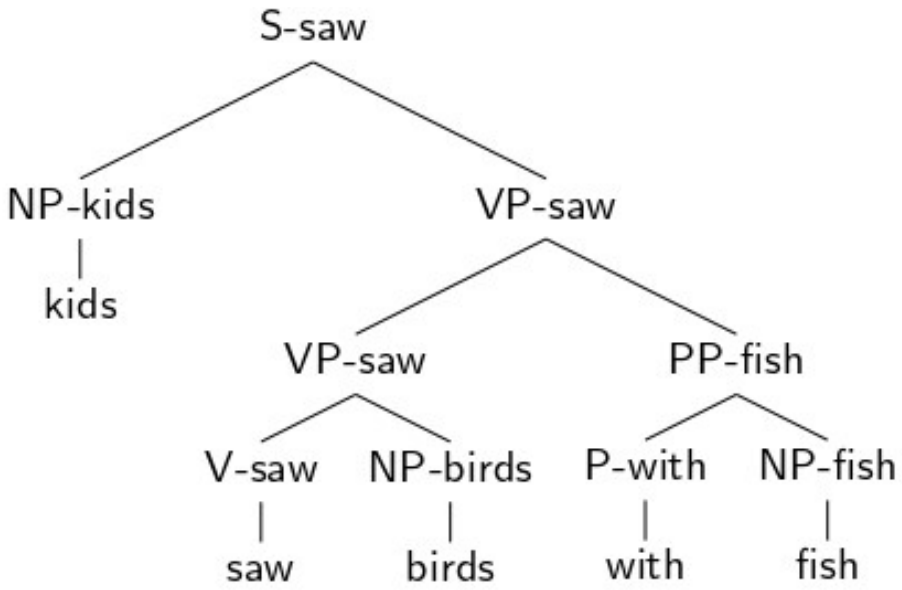
Parsing is not just WSJ. Lots of situations are much harder!

- Other languages, esp with free word order (up next) or little annotated data.
- Other domains, esp with jargon (e.g., biomedical) or non-standard language (e.g., social media text).

In fact, due to increasing focus on multilingual NLP, constituency syntax/parsing (English-centric) is losing ground to **dependency parsing**...

6.1.7 Lexicalization, again

We saw that adding lexical head of the phrase can help choose the right parse:



Dependency syntax focuses on the head-dependent relationships

6.1.8 Dependency syntax

An alternative approach to sentence structure.

- A fully lexicalized formalism: no phrasal categories.

- Assumes binary, asymmetric grammatical relations between words: head-dependent relations, shown as directed edges:
- Here, edges point from heads to their dependents.

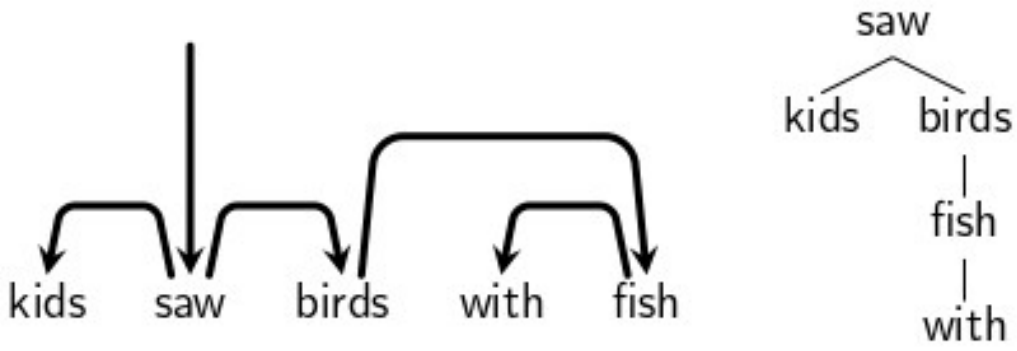
6.1.9 Dependency trees

A valid dependency tree for a sentence requires:

- A single distinguished root word.
- All other words have exactly one incoming edge.
- A unique path from the root to each other word.

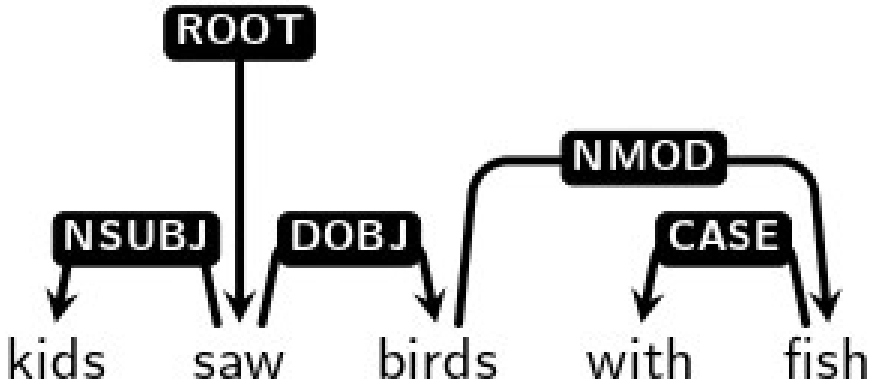
6.1.10 It really is a tree!

- The usual way to show dependency trees is with edges over ordered sentences
- But the edge structure (without word order) can also be shown as a more obvious tree



6.1.11 Labelled dependencies

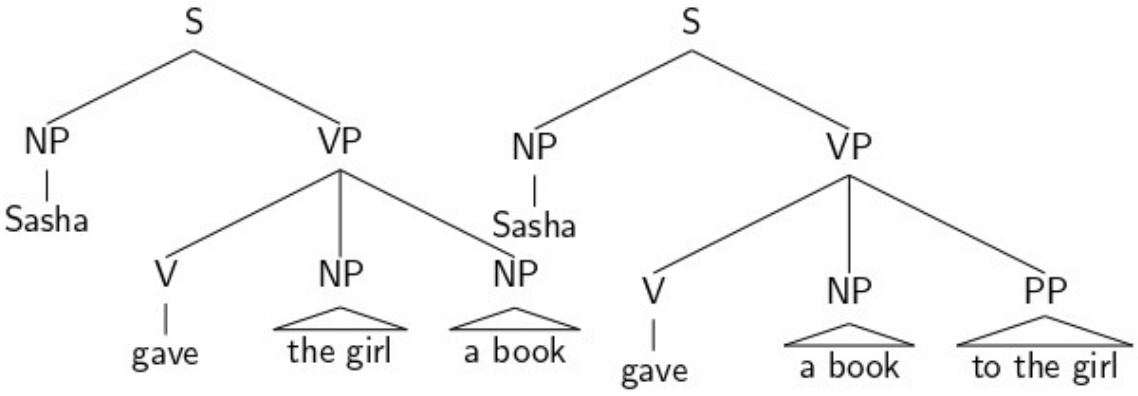
It is often useful to distinguish different kinds of head → modifier relations, by labelling edges:



- Historically, different treebanks/languages used different labels.
- Now, the Universal Dependencies project aims to standardize labels and annotation conventions, bringing together annotated corpora from over 50 languages.
- Labels in this example (and in textbook) are from UD.

6.1.12 Why dependencies??

Consider these sentences. Two ways to say the same thing:



We only need a few phrase structure rules:

- S → NP VP
- V →NP NP
- VP → V NP PP

plus rules for NP and PP.

6.1.13 Equivalent sentences in Russia

Russian uses morphology to mark relations between words:

- knigu means book (kniga) as a direct object.
- devochke means girl (devochka) as indirect object (to the girl).

So we can have the same word orders as English:

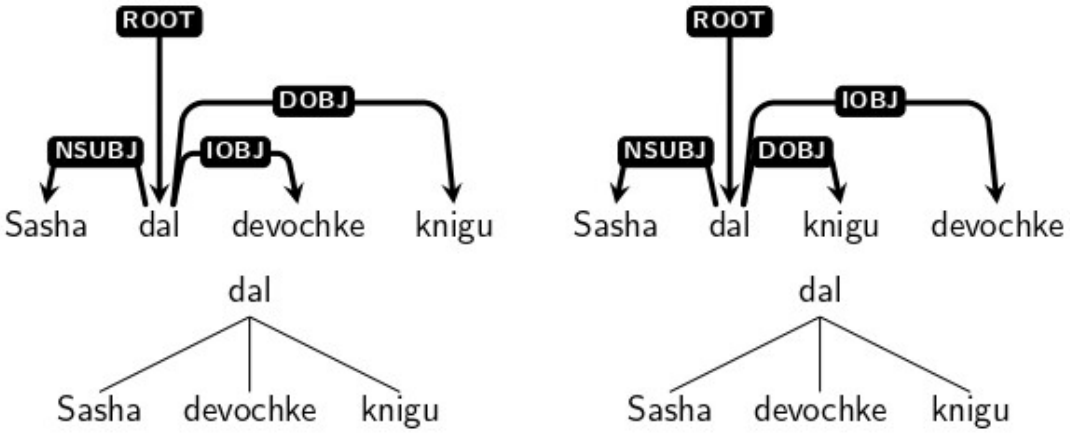
- Sasha dal devochke knigu
- Sasha dal knigu devochk

But also many others!

- Sasha devochke dal knigu
- Devochke dal Sasha knigu
- Knigu dal Sasha devochke

6.1.14 Phrase structure vs dependencies

- In languages with free word order, phrase structure (constituency) grammars don't make as much sense
  - E.g., we would need both S → NP VP and S → VP NP, etc. Not very informative about what's really going on.
- In contrast, the dependency relations stay constant:



6.1.15 Pros and cons

- Sensible framework for free word order languages
- Identifies syntactic relations directly. (using CFG, how would you identify the subject of a sentence?)
- Dependency pairs/chains can make good features in classifiers, for information extraction, etc.
- Parsers can be very fast

But

- The assumption of asymmetric binary relations isn't always right... e.g., how to parse dogs and cats?

6.1.16 How do we annotate dependencies?

Two options:

1. Annotate dependencies directly.
2. Convert phrase structure annotations to dependencies. (Convenient if we already have a phrase structure treebank.)

6.1.17 Head Rules

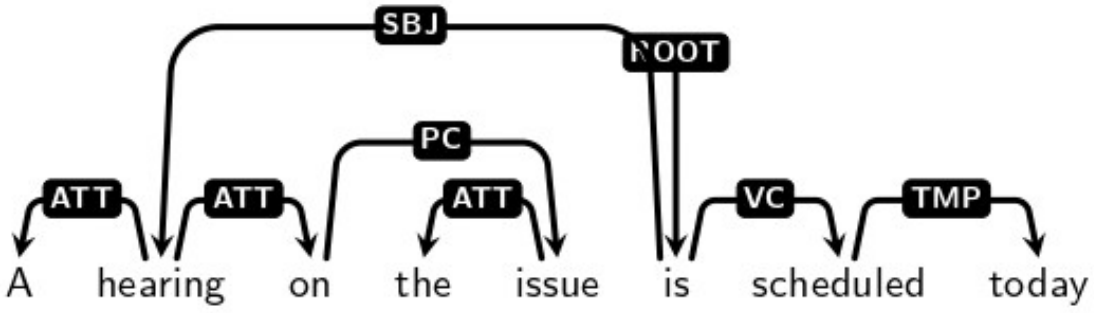
We saw how the lexical head of the phrase can be used to collapse down to a dependency tree.But how can we find each phrase's head in the first place? The standard solution is to use head rules: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. **S** → **NP** **VP**,**VP** → **VP** **PP**, **PP** → **P** **NP** (content head), etc.

Heuristics to scale this to large grammars: e.g., within an **NP**, last immediate **N** child is the head.

6.1.18 Projectivity

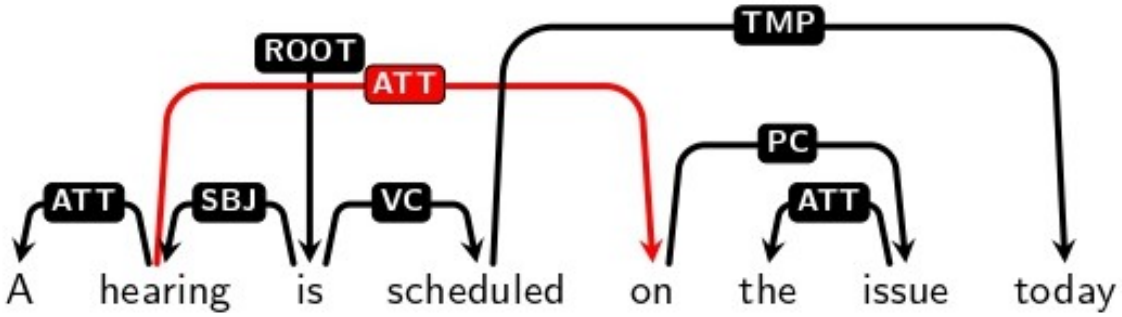
If we convert constituency parses to dependencies, all the resulting trees will be projective.

- Every subtree (node and all its descendants) occupies a contiguous span of the sentence.
- = the parse can be drawn over the sentence w/ no crossing edges.



6.1.19 Nonprojectivity

But some sentences are nonprojective.



- We'll only get these annotations right if we directly annotate the sentences (or correct the converted parses).
- Nonprojectivity is rare in English, but common in many languages.
- Nonprojectivity presents problems for parsing algorithms.

6.1.20 Dependency Parsing

Some of the algorithms you have seen for PCFGs can be adapted to dependency parsing

CKY can be adapted, though efficiency is a concern: obvious approach is O(Gn^5); Eisner algorithm brings it down to O(Gn^3)

- N. Smith's slides explaining the Eisner algorithm: http://courses.cs.washington.edu/courses/cse517/ 16wi/slides/an-dep-slides.pdf

Shift-reduce:more efficient, doesn't even require a grammar!

6.1.21 Recall: shift-reduce parser with CFG

- Same example grammar and sentence.
- Operations:
  - Reduce(R)
  - Shift (S)
  - Backtrack to step n (Bn)
- Note that at 9 and 11 we skipped over backtracking to 7 and 5 respectively as there were actually no choices to be made at those points.

6.1.22 Transition-based Dependency Parsing

The arc-standard approach parses input sentence $w_1 \dots w_N$ using two types of reduce actions (three actions altogether):

- **Shift**: Read next word wi from input and push onto the stack.

- **LeftArc**:Assign head-dependent relation s2 ← s1; pop s2
- **RightArc**: Assign head-dependent relation s2 → s1; pop s1

where s1 and s2 are the top and second item on the stack, respectively. (So, s2 preceded s1 in the input sentence.)

6.1.23 Labelled dependency parsing

These parsing actions produce unlabelled dependencies (left).

For labelled dependencies (right), just use more actions: LeftArc(NSUBJ), RightArc(NSUBJ), LeftArc(DOBJ), ...

6.1.24 Differences to constituency parsing

Shift-reduce parser for CFG: not all sequences of actions lead to valid parses. Choose incorrect action → may need to backtrack.

Here, all valid action sequences lead to valid parses.

- Invalid actions: can't apply LeftArc with root as dependent; can't apply RightArc with root as head unless input is empty.
- Other actions may lead to incorrect parses, but still valid.

So, parser doesn't backtrack. Instead, tries to greedily predict the correct action at each step.

- Therefore, dependency parsers can be very fast (linear time).
- But need a good way to predict correct actions.

6.1.25 Notions of validity

In constituency parsing, valid parse = grammatical parse.

- That is, we first define a grammar, then use it for parsing.

In dependency parsing, we don't normally define a grammar.

6.1.26 Summary: Transition-based Parsing

- arc-standard approach is based on simple shift-reduce idea.
- Can do labelled or unlabelled parsing, but need to train a classifier to predict next action, as we'll see next time.
- Greedy algorithm means time complexity is linear in sentence length.
- Only finds projective trees (without special extensions)
- Pioneering system: Nivre's MaltParser.

6.1.27 Alternative: Graph-based Parsing

- Global algorithm: From the fully connected directed graph of all possible edges, choose the best ones that form a tree.

- Edge-factored models: Classifier assigns a nonnegative score to each possible edge; maximum spanning tree algorithm finds the spanning tree with highest total score in $O(n^2)$ time.
- Pioneering work: McDonald's MSTParser
- Can be formulated as constraint-satisfaction with integer linear programming (Martins's TurboParser)

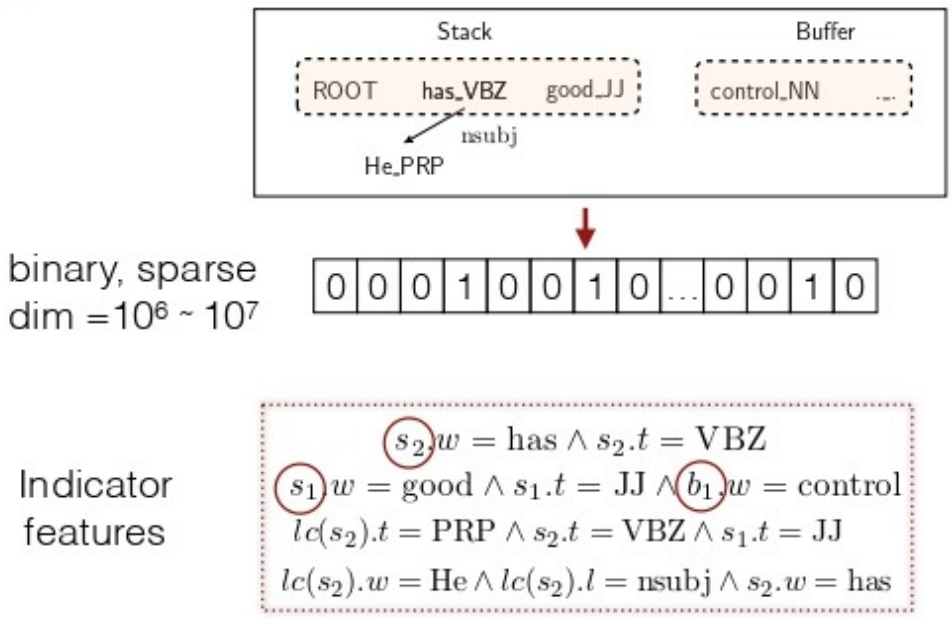
6.1.28 Graph-based vs. Transition-based vs. Conversion-based

TB: Features in scoring function can look at any part of the stack; no optimality guarantees for search; linear-time; (classically) projective only
GB: Features in scoring function limited by factorization; optimal search within that model; quadratic-time; no projectivity constraint
CB: In terms of accuracy, sometimes best to first constituency-parse, then convert to dependencies (e.g., Stanford Parser). Slower than direct methods.

6.1.29 Summary

- Constituency syntax: hierarchically nested phrases with categories like NP.
- Dependency syntax: trees whose edges connect words in the sentence. Edges often labeled with relations like nsubj.
- Can convert constituency to dependency parse using head rules.
- For projective trees, transition-based parsing is very fast and can be very accurate.
- Google "online dependency parser"

6.1.30 Traditional Features



w = word, t = part-of-speech tag, l = dep.label, lc = leftmost child

6.1.31 Indicator Features Revisited

- Problem 1: sparse
  - lexicalized features
  - high-order interaction features
- Problem 2: incomplete
  - Unavoidable in hand-crafted feature templates.

- Problem 3: computationally expensive
  - More than 95% of parsing time is consumed by feature computation.

Solution: Neural Networks!

6.1.32 Distributed Representations

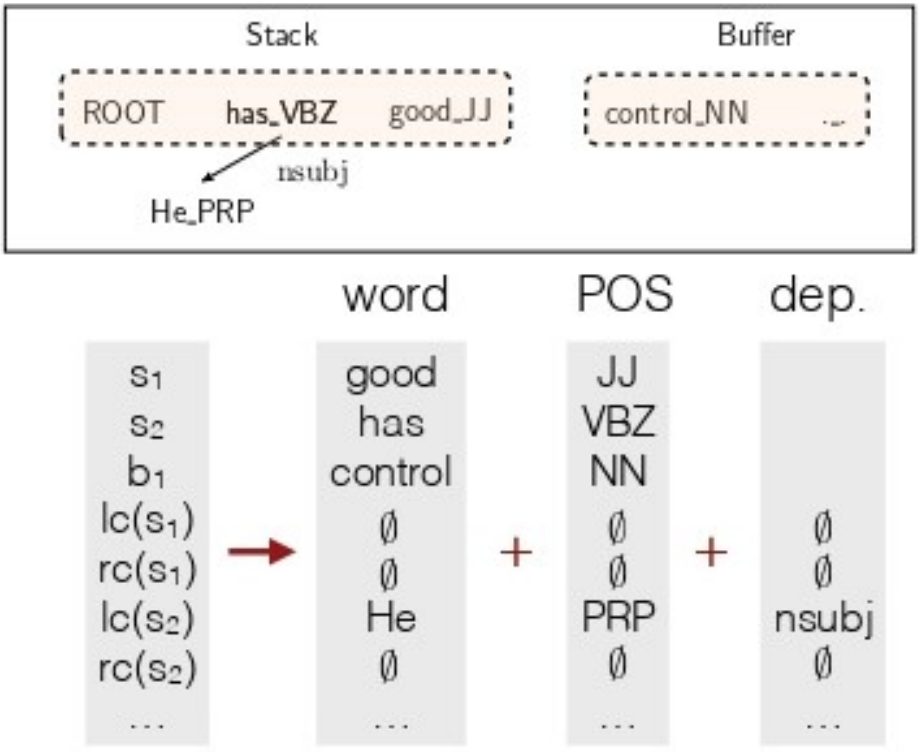
How to encode all the available information?
How to model high-order features?

- Represent each word as a d-dimensional dense vector(i.e., word embeddings)
  - Similar words expect to have close vectors.
- Meanwhile,part-of-speech tags and dependency labels are also represented as d-dimensional vectors
  - POS and dependency embeddings.
  - The smaller discrete sets also exhibit many sematical similarities.

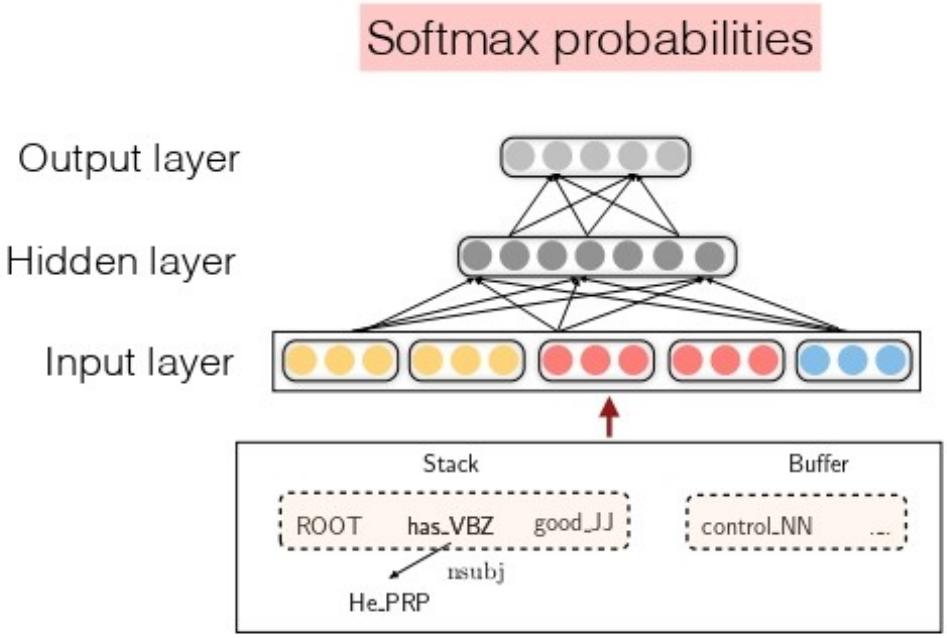
NNS (plural noun) should be close to NN (singular noun). num (numerical modifier) should be close to amod (adjective modifier).

6.1.33 Exctracting Tokens from Configuration

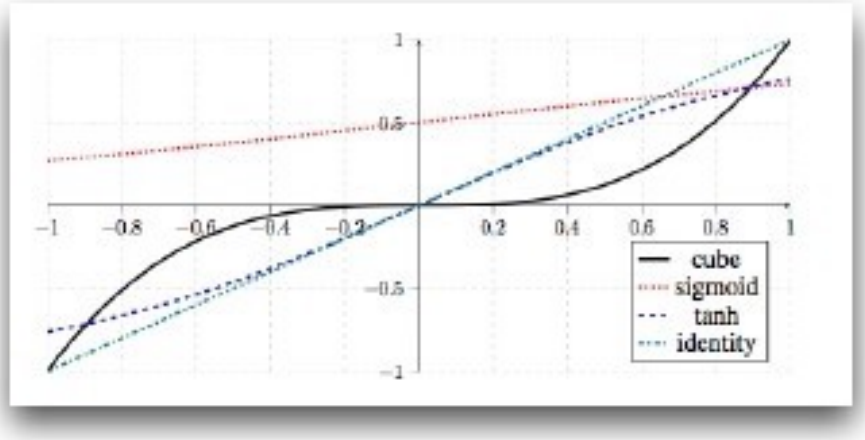
Exctract a set of tokens based on the positions.



6.1.34 Model Architecture



6.1.35 Cube Activation Function



6.1.36 Training

- Generating training examples using a fixed oracle.
- Training objective: cross entropy loss.
- Back-propagation to all embeddings.
- Initialization:
  - Word embeddings from pre-trained word vectors.
  - Random initialization for others.

6.1.37 Indicator vs. Dense Features

6.1.38 Results

- **Problem #1: sparse**

  Distributed representations can capture similarities.

- **Problem #2: incomplete**

  We don't need to enumerate the combinations.
  Cube non-linearity can learn combinations automatically.

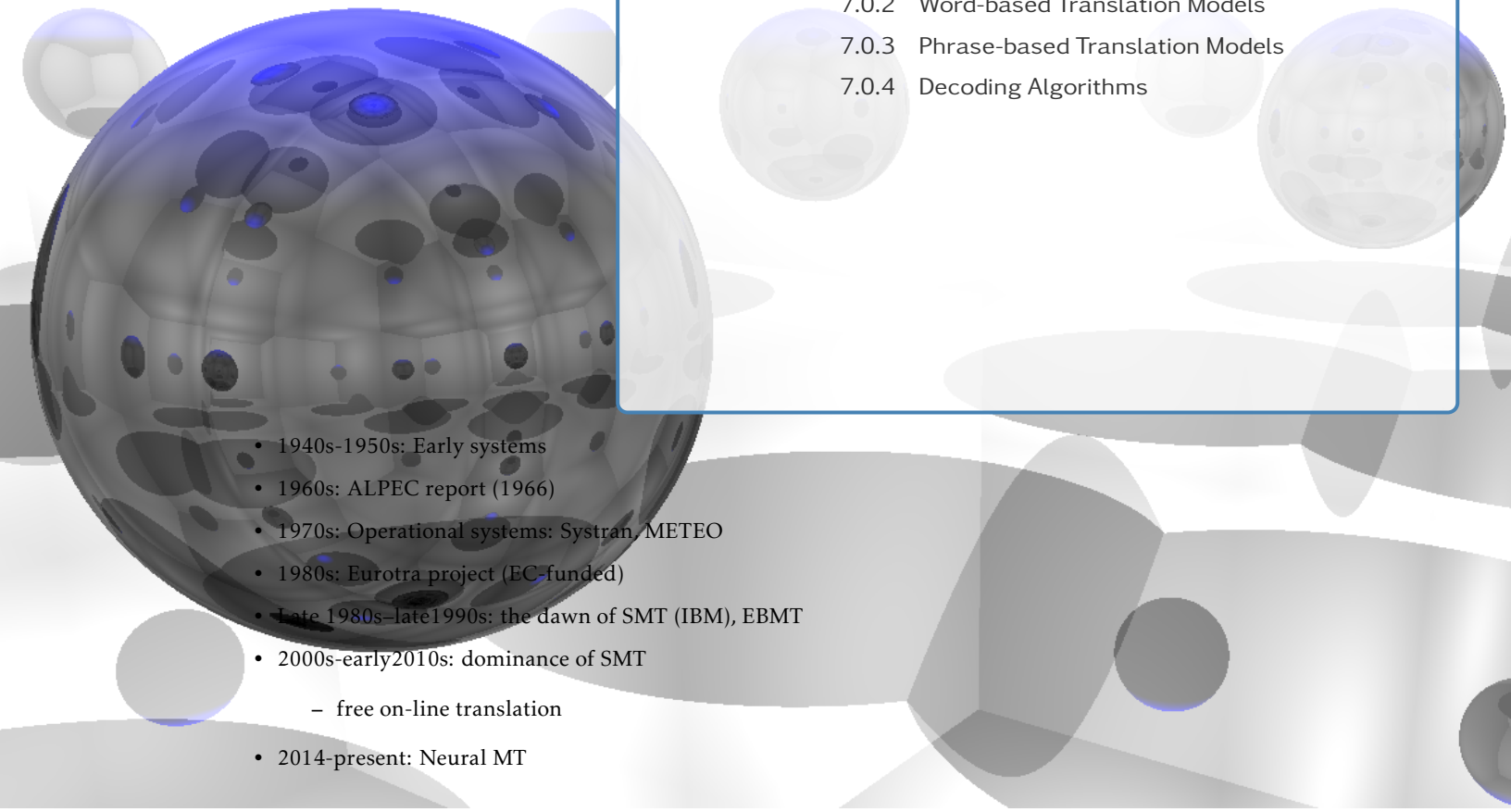
- **Problem #3: computationally expensive**

  String concatenation + look-up in a big table $\Longrightarrow$ matrix operations. Pre-computation trick can speed up.

		Unlabeled attachment score (UAS)	sent / s
Transition -based	MaltParser (greedy)	89.9	560
		+2.1	×1.8
	Our Parser (greedy)	92.0	1013
	Zpar: beam = 64	92.9*	29*
Graph -based	MSTParser	92.0	12
	TurboParser	93.1*	31*

7. Machine Translation

Accurate machine translation between human languages Free conversation between humans and computers



- 1940s-1950s: Early systems
- 1960s: ALPEC report (1966)
- 1970s: Operational systems: Systran, METEO
- 1980s: Eurotra project (EC-funded)
- Late 1980s–late1990s: the dawn of SMT (IBM), EBMT
- 2000s-early2010s: dominance of SMT
  - free on-line translation
- 2014-present: Neural MT

- Direct Translation
- Rule-based Machine Translation (RBMT)
- Memory-based Translation
- Example-based Machine Translation (EBMT)
- Statistical Machine Translation (SMT)
- Neural Machine Translation (NMT)

Theoretically, interlingua-based approach is ideal because it requires much less components compared with transfer-based approach. However, using a human-defined interlingua (i.e. a specific knowledge expression) is practically too difficult. In business, a feasible way is to use a natural language (mostly English) as the interlingua, which is usually called a pivot language in this case.

Makoto Nagao (Kyoto University): When the pivot language (i.e. interlingua) is used, the results of the analytic stage must be in a form which can be utilized by all of the different languages into which translation is to take place. This level of subtlety is a practical impossibility. (Machine Translation, Oxford, 1989)
Patel-Schneider (METAL system): METAL employs a modified transfer approach rather than an interlingua. If a meta-language (an interlingua) were to be used for translation purposes, it would need to incorporate all possible features of many languages. That would not only be an endless task but probably a fruitless one as well. Such a system would soon become unmanageable and perhaps collapse under its own weight. (A four-valued semantics for terminological reasoning, Artificial Intelligence, 38, 1989)

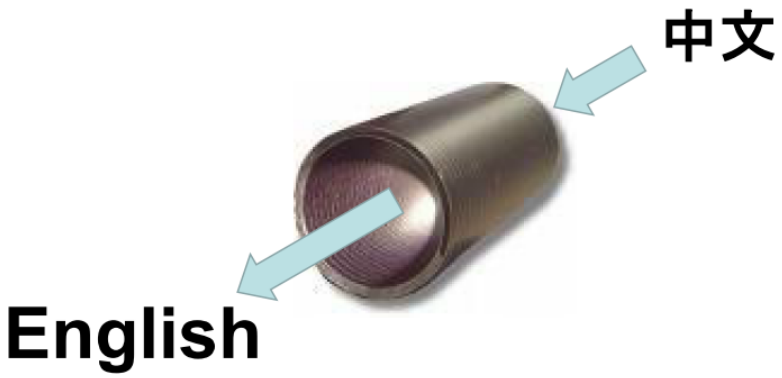


Figure 7.1: Accurate machine translation (Recap)

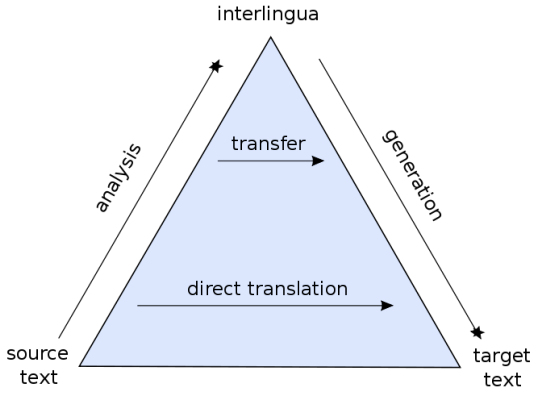


Figure 7.2: The Vauquois Pyramid

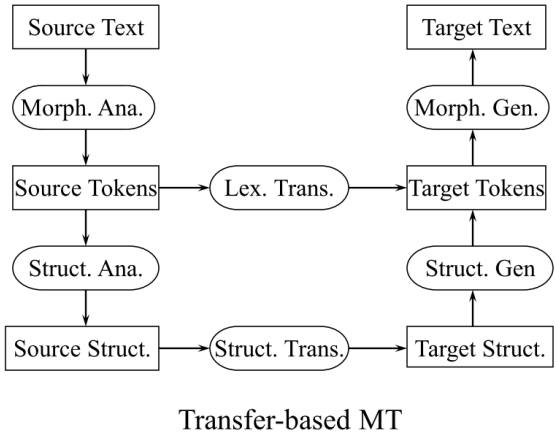


Figure 7.3: Transfer-based MT

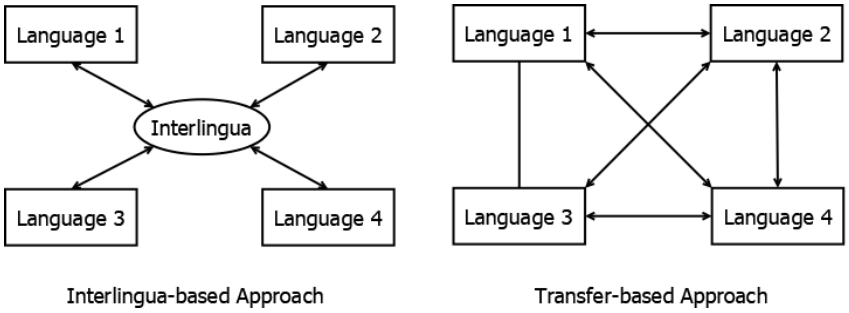


Figure 7.4: Interlingua approach vs. transfer approach

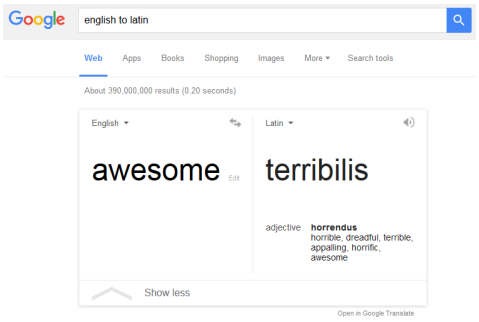


Figure 7.5: Web Translator and Photo Translator

Figure 7.6: Voice Translator and Real-time Conference Translator

Since neural machine translation become popular, the quality of MT systems is greatly improved, and MT is used in more and more scenarios.

Machine translation evaluation is not easy because corrent translations for the same text may differ a lot literally.

- Human evaluation:
  - Reliable
  - Inconsistent
  - Expensive
  - Slow

- Automatic evaluation:
  - Unreliable
  - Consistent
  - Cheap
  - Fast

Goals for Automatic MT Evaluation

- Meaningful: score should give intuitive interpretation of translation quality.
- Consistent: repeated use of metric should give same results.
- Correct: metric must rank better systems higher.
- Low cost: reduce time and money spent on carrying out evaluation.
- Tunable: automatically optimise system performance towards metric.

Automatic MT Evaluation Metrics

Input:

- output of machine translation systems.
- reference translations give by human translators.
- source text (optional).

Output:

- a score which represents the similarity between the MT output and the human reference given the source sentence.

Plenty of automatic MT evaluation metrics are proposed by researchers, among which the most popular metrics include:

- WER (Word Error Rate)
- BLEU (BiLingual Evaluation Understudy)
- TER (Translation Error Rate)
- METEOR (Metric for Evaluation of Translation with Explicit ORdering)

Word Error Rate (WER): the ratio of the Levenstain Distance between the MT result and the reference, to the number of words in the reference. Levenstain Distance (or Edit Distance): the minimal number of edit operations (insertions, deletions or substitutions) required to change the reference to the MT result:

Reference:

- Israeli officials are responsible for airport security

MT:

- Israeli official is responsible airport security

WER: calculation

- Label all matched words
- Search a shortest path from top-left corner to bottom-right corner

Insertions	are, for	2
Deletions	is	1
substitutions	official →officials	1

Figure 7.7: WER: an example

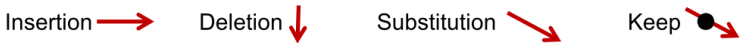


Figure 7.8: WER: calculation

- Calculate the number of different operations

Good translation but in opposite order to the reference translation will have a **low WER score**.

Reference translation:

- Israeli officials are responsible for airport security

System output:

- This airport's security is the responsibility of the Israeli security officials

BLEU: Motivation

- N-gram precision between the machine translation and the reference
- Compute n-gram precisions for 1 to n (typically $n = 4$)
- A harmony average of the n-gram precisions is calculated over different n's
- Compute on the entire test set, rather than single sentences, to avoid 0 value for higher n-gram precisions

BLEU: Definition

$$\text{BLEU} = \underbrace{\min(1, \frac{length\_of\_MT}{length\_of\_reference})}_{\text{Brevity Penalty}} \times \underbrace{(\prod_{i=1}^{n} \text{Precision}_i)^{\frac{1}{n}}}_{\text{N-gram Precision}}$$

$$\text{Precision}_i = \frac{clipped\_number\_of\_matched\_\{i\}grams\_in\_MT}{number\_of\_total\_\{i\}grams\_in\_MT}$$

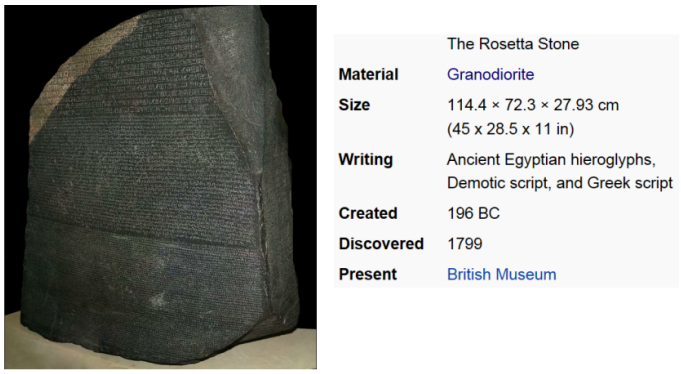
Brevity Penalty is used to avoid that a very short MT get a high BLEU score. The number of matched $\{i\}$grams is *clipped* to avoid that a single word in the reference is matched multiple times in MT.

To avoid translation variability, multiple references can by used:

- n-grams may match in any of the references
- closest reference length is used for brevity penalty

7.0.1 SMT: basic ideas

In the time of rule-based MT, the translation knowledge was explicitly encoded by linguists, which is labor intensive, time consuming and inefficient. The basic idea of statistical machine translation is to learn a machine translation system directly from parallel translation data, without involving linguistics to encode translation knowledge.



The Rosetta Stone	
Material	Granodiorite
Size	114.4 × 72.3 × 27.93 cm (45 x 28.5 x 11 in)
Writing	Ancient Egyptian hieroglyphs, Demotic script, and Greek script
Created	196 BC
Discovered	1799
Present	British Museum

Figure 7.9: Rosette Stone

The first statistical machine translation was conducted by IBM researchers in late 1980s to early 1990s. The research of SMT was not very active in 1990s because the computing resources used by IBM researchers were higher than most researchers could have. In 1999, a group of researchers gathered in the JHU summer workshop and repeated IBM's early work successfully and release GIZA++, the implementation of the training algorithm of IBM models. 2000 SMT became a dominate MT paradigm until it was replaced by NMT in recent years.

A parallel corpus, or a bitext, is a collection of texts in one language and its correnting translation in another language. Parallel corpara used for MT research are normally aligned at sentence level. Document-level MT needs parallel corpara aligned in both word level and document level. Many parallel corpara are aligned only in document level in their original form, so a technique called sentence alignement is developed to align the sentences in a document aligned corpus.

Rosette stone is a good example that it is feasible to learn translation knowledge from a parallel corpus.

A statistical translation model is the probability that a target text e is the translation of a given source text f:

$$p(e|f), \text{where} \sum_{e} p(e|f) = 1$$

Given a statistical translation $p(e|f)$, the machine translation problem can be transfered to a search problem: to search a target sentence e which has the highist translation probability given f:

$$\hat{e} = \underset{e}{\operatorname{argmax}}\, p(e|f)$$

Statistical machine translation with a single translation model does not work well. IBM researchers proposed a Noisy Channel Model for statistical machine translation, which includes an additional language model.

> Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, Paul S. Roossin, A Statistical Approach to Machine Translation, Computational Linguistics,1990

- The *translation model* models how likely it is that f is a translations of e – **adequacy**.
- The *language model* models how likely it is that e is an acceptable sentence – **fluency**.
- The *decoder* searches for the most likely e.
- We have introduced language models in previouse lectures, here we will mainly focus on translation models and decoding algorithms

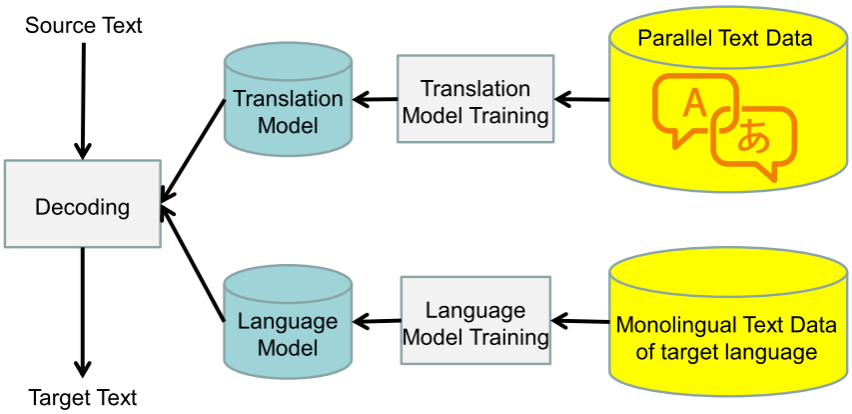


Figure 7.10: SMT Workflow

7.0.2 Word-based Translation Models

Various translation models have been proposed, which belong to different categories, according to the language units on which they are built up:

- Word-based models
  - IBM models 1-5
  - HMM models
- Phrase-based models
- Syntax-based models
  - Tree-to-string models
  - String-to-tree models
  - Tree-to-tree models
  - Dependency-based models

IBM researchers proposed 5 models with increased complexity:

- IBM Model 1: only consider lexical translation probabilities
- IBM Model 2: add a absolute reordering model
- IBM Model 3: add a fertility model
- IBM Model 4: add a relative reordering model
- IBM Model 5:

To estimate the word translation probabilities, we need alignment between words in the parallel sentences

Can you image a word alignment pattern like this?

We would like to estimate the lexical translation probabilities from a parallel corpus...but we do not have the alignments:

- If we had the alignments, we could estimate the lexical translation probabilities.
- If we had the probabilities, we could estimate the alignments.

Incomplete data

English	Chinese	Prob.
a	一	0.2
a	一个	0.4
a	个	0.2
a	一只	0.1
a	一本	0.05
a	...	...

English	Chinese	Prob.
book	书	0.7
book	预定	0.2
book	...	...
take	拿	0.4
take	带走	0.3
take	...	...

Figure 7.11: Lexical translation probabilities

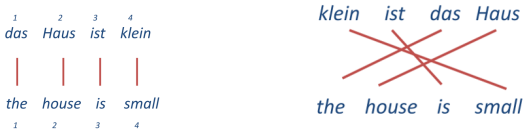


Figure 7.12: Word alignment

Figure 7.13: Word alignment

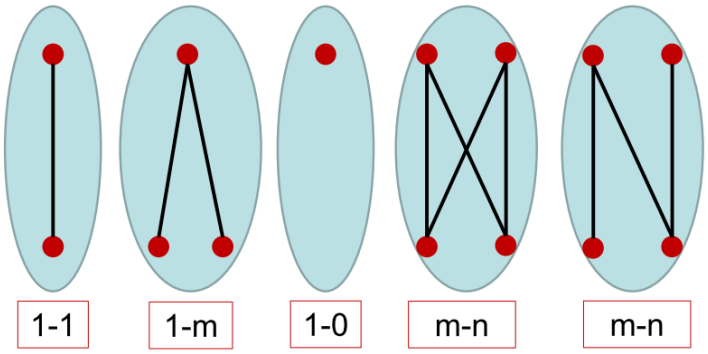


Figure 7.14: Word alignment patterns

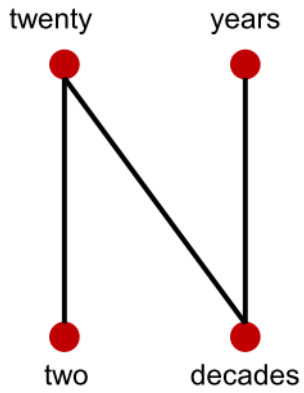


Figure 7.15: Word alignment patterns

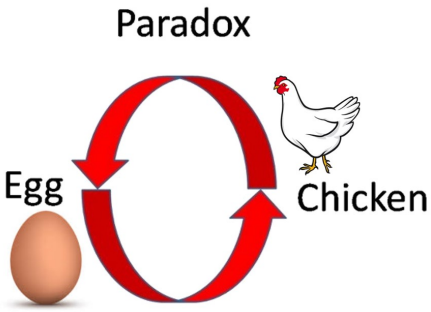


Figure 7.16: A Paradox

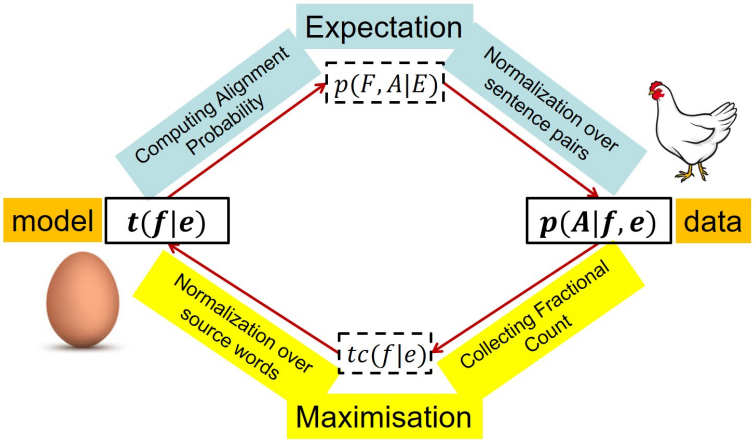


Figure 7.17: EM Algorithm

- If we had complete data, we could estimate model.
- If we had the model, we could fill in the gaps in the data.

Solution: **Expectation Maximization (EM)** Algorithm

- Initialize model parameters. (e.g. uniform)
- Assign probabilities to the missing data. (E-step)
- Astimate model parameters from completed data. (M-step)
- Iterate E-step and M-step until the model converges.

EM Algorithm consists of two steps:

- **Expectation-Step**: Apply model to the data
  - parts of the data are hidden (here: alignments)
  - using the model, assign probabilities of the hidden data to possible values (alignments)
- **Maximization-Step**: Estimate new model from data
  - take assigned values as fact
  - collect counts (weighted by probabilities)
  - estimate new model from counts
- Iterate the E-step and the M-step until convergence

7.0.3 Phrase-based Translation Models

Word-based translation models do not take into account contextual information for translation decisions. They are not good at dealing with 1-to-many, many-to-1 and many-to-many translations.

Phrase-based translate models are proposed to solve the problems for word-based models.

A monolingual phrase can be any contiguous sequence of words in a sentence.

- A phrase is not necessarily syntactically well-formed

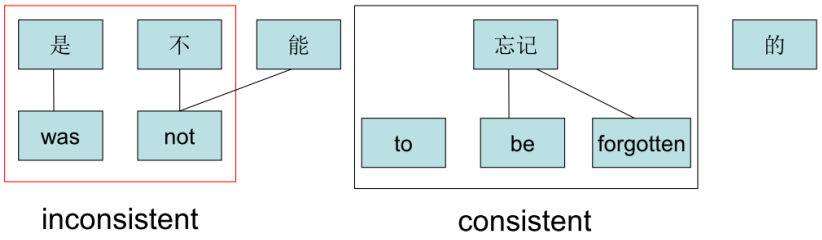


Figure 7.18: Bilingual Phrase Pairs

English	Probability	English	Probability
the proposal	0.6277	the suggestions	0.0114
's proposal	0.1068	the proposed	0.0114
a proposal	0.0341	the motion	0.0091
the idea	0.025	the idea of	0.0091
this proposal	0.0227	the proposal ,	0.0068
proposal	0.0205	its proposal	0.0068
of the proposals	0.0159	it	0.0068
the proposals	0.0159	.....	

Figure 7.19: Bilingual Phrase Pairs

- A phrase is not necessarily semantically meaningful

A bilingual phrase pair should be consistent with word alignment.

A bilingual phrase pair should be constistent with word alignment:

A real example taken from Europarl for the German phrase **den Vorschlag:**

Task: learn the model from a parallel corpus

Three stages:

1. Word alignment: using IBM models or other method
2. Extraction of phrase pairs
3. Scoring phrase pairs

With IBM models, each target word can be aligned to at most one source word (patterns supported: 1-0,0-1,1-1,m-1). Therefore, it's not possible to end up with an alignment of one target word to many source words (patterns not supported: 1-m, m-m). To obtain a word alignment with all possible patterns, a symmetric word alignment algorithm should be adopted.

- A typical symmetric word alignment algorithm:
  - Word alignment using IBM Models in one direction.
  - Word alignment using IBM Models in the other direction.
  - Merge the above two alignment results with a certain criterion.

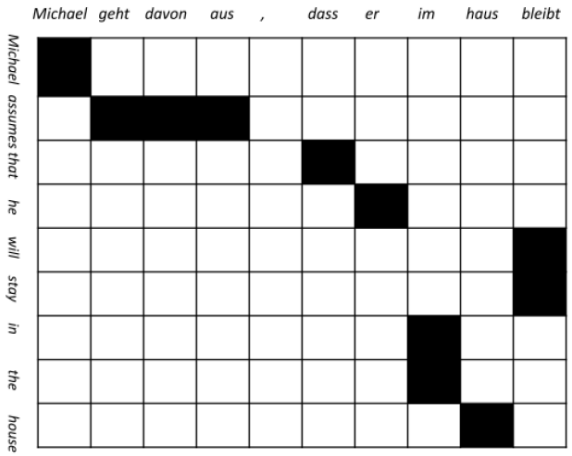


Figure 7.20: A matrix view of word alignment

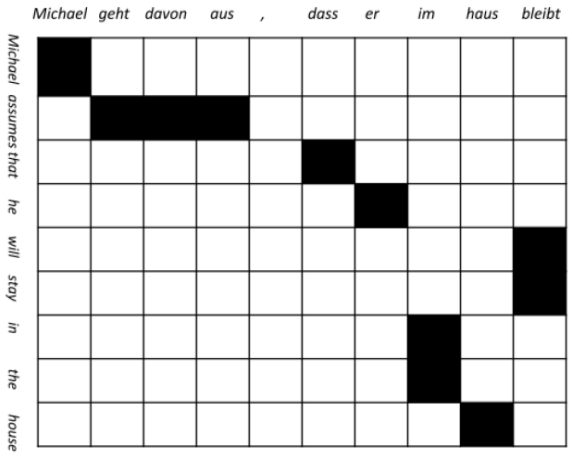


Figure 7.21: A matrix view of word alignment

a

A phrase pair (e, f) is consistent with a bidirectional word alignment A if and only if:

- For all words $e_i \in e$, if exists an f_j: $(e_i, f_j) \in A$, then $f_j \in f$.
- For all word $f_j \in f$, if exists an e_i: $(e_i, f_j) \in A$, then $e_i \in e$.
- There exists an $e_i \in e$, and an $f_j \in f$: $(e_i, f_j) \in A$

A consistent phrase pair defined by the red area should meet the following requirement:

- There should be one or more filled blocks in the red area.
- The blue areas should be all clear.

Consistent phrases in the matrix view

A consistent phrase pair defined by the red area should meet the following requirement:

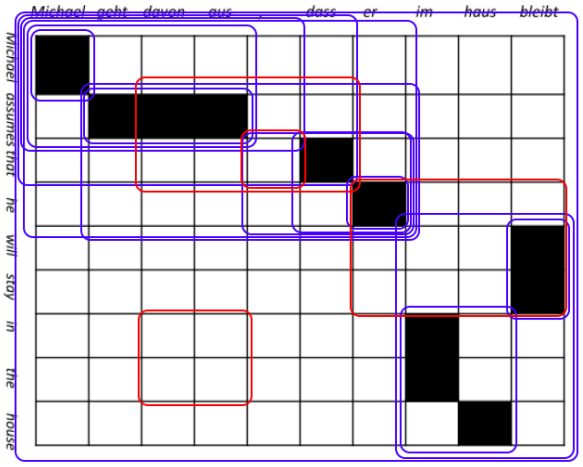
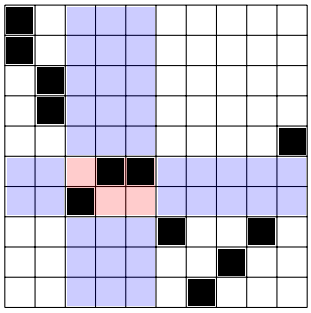


Figure 7.22: Phrase pair extraction

- There should be one or more filled blocks in the red area.
- The blue areas should be all clear.

1.03/0.03/2.47/1.47, 0.03/1.53/0.97/2.47, 2.53/1.53/4.97/2.47, 1.03/2.53/2.47/4.97 0.03/4.53/0.47/4.97, 0.03/4.03/0.47/4.47, 0.53/3.53/0.97/3.97, 0.53/3.03/0.97/3.47, 1.03/1.53/1.47/1.97, 1.53/2.03/1.97/2.47, 2.03/2.03/2.47/2.47, 2.53/1.03/2.97/1.47, 3.03/0.03/3.47/0.47, 3.53/0.53/3.97/0.97, 4.03/1.03/4.47/1.47, 4.53/2.53/4.97/2.97



consistent phrase pairs, inconsistent phrase pairs

- Phrase pairs extracted from the above example:
  - michael assumes — michael geht davon aus ,
  - michael assumes — michael geht davon aus
  - assumes that — geht davon aus , dass
  - assumes that he — geht davon aus , dass er
  - that he — , dass er
  - that he — dass er
  - in the house — im haus
  - michael assumes that — michael geht davon aus , dass

- michael assumes that he — michael geht davon aus , dass er
- michael assumes that he will stay in the house — michael geht davon aus , dass er im haus bleibt
- assumes that he will stay in the house — geht davon aus , dass er im haus bleibt
- that he will stay in the house — dass er im haus bleibt ,
- that he will stay in the house — dass er im haus bleibt
- he will stay in the house — er im haus bleibt
- will stay in the house — im haus bleibt

Phrase pair scoring: assign probabilities to all the phrase pairs extracted from the aligned corpus.

Scoring by relative probability:

$$\phi(f|e) = \frac{count(e,f)}{count(e)} = \frac{count(e,f)}{\sum_{f_i} count(e,f_i)}$$

7.0.4 Decoding Algorithms

Decoding is to searche for the most likely target sentence e for a given source sentence f:

$$\hat{e} = \underset{e}{\operatorname{argmax}}\, p(e)p(f|e)$$

- Build the translation from left to right:
  1. Match the untranslated source words against the phrase table, and locate a matched source phrase
  2. Append the target phrase to the end of the partial translation
  3. Mark the source words translated
  4. Iterate (1) to (3) until all source words are marked translated.
- One to many translation
- Many to one translation
- Reordering: skip forward
- Reordering: skip backward

When multiple untranslated source phrases are matched against the phrase table, which one should we select? Should we translate a source phrase immediately in the right side, or we can skip forward or backward to select another source phrase? If multiple bilingual phrases are matched and have the same source phrase but different target phrases, which target phrase should we select?

All the possible decisions during search form a huge search space. A Heuristic Beam Search algorithm is designed to search the optimal translation:

- Hypotheses are organized in different beams.
- All the hypotheses in the same beam share the same number of translated source words.
- A score is assigned to each hyposis.
- Hypotheses with scores lower than a certain threshold are pruned.
- A group of hypotheses can be recombined if one of them can represent the others in the future steps.

Chris Callison-Burch and Philipp Koehn. Introduction to statistical machine translation. Slides for ESSLL 2005.

- According to the noisy channel model, the score of a hypothesis includes two components:

```
initialize hypothesisStack[0 .. nf];
create initial hypothesis hyp_init;
add to stack hypothesisStack[0];
for i=0 to nf-1:
  for each hyp in hypothesisStack[i]:
    for each new_hyp that can be derived from hyp:
      nf[new_hyp] = number of foreign words covered by new_hyp;
      add new_hyp to hypothesisStack[nf[new_hyp]];
      prune hypothesisStack[nf[new_hyp]];
find best hypothesis best_hyp in hypothesisStack[nf];
output best path that leads to best_hyp;
```

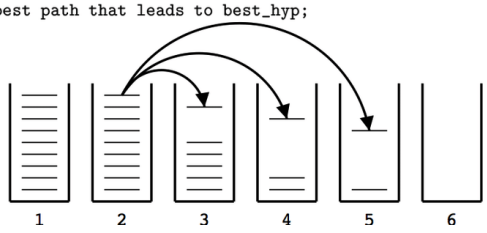


Figure 7.23: Beam Search Algorithm

  - Language model score
  - Phrase-based translation model score

- Shortcomings of the above scoring mechanism:
  - The importance of language model and translation model may be different
  - More factors should be considered in the decisions, for example, word reordering (to select a next source phrase to translate, or skip forwards or backwards), etc.

A log-linear framework was proposed to replace the noisy channel framework to overcome the above shortcomings:

$$p(e|f) = \frac{\exp(\sum_{i=1}^{n} \lambda_i h_i(e,f))}{\sum_{e'} \exp(\sum_{i=1}^{n} \lambda_i h_i(e',f))}$$

$$\hat{e} = \underset{e}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i h_i(e,f)$$

- Advantages of the log-linear framework:
  - Abitrary number of user-defined features (h_i) can be added in the model.
  - Weights (λ_i) can be tuned to balance the importance among the features.

- A held-out data set (usually called development set) is used to train the parameters (λ_i) for the log-linear model;
- The training process of the log-linear model is called fune-tuning in SMT;
- Unlike in neural network training, the gradient descend algorithm is not applicable here, because the input data is discrete symbols and the function is not differetiable.
- Fune-tuning algorithms, e.g. MERT, or MIRA, were developed to fine-tune the log-linear model for SMT.

- Under the noisy channel framework, only two features are considered:

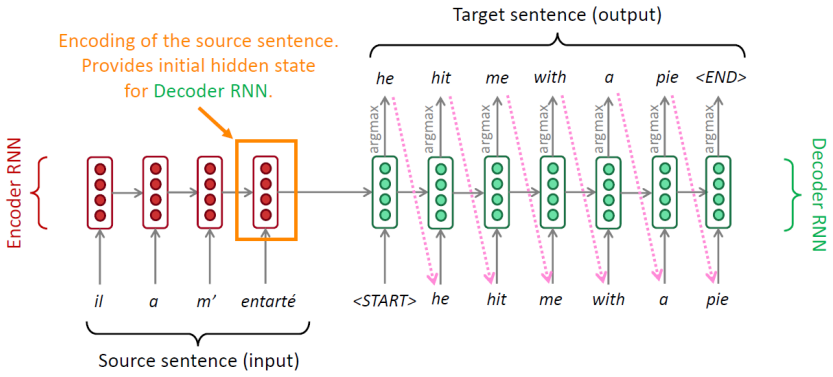


Figure 7.24: The sequence-to-sequence model

  - Target language model.
  - Backward translation model.

- Under the log-linear framework, more features can be considered, typically including:
  - Forward translation model.
  - Forward and backward lexicalized translation models.
  - Additional language models. (e.g. with different ngram orders)
  - Word reordering models.
  - Length of the target sentence.
  - User-defined dictionaries.

Neural Machine Translation (NMT) is a way to do Machine Translation with a single neural network. The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves two RNNs.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding. Encoder RNN produces an encoding of the source sentence. Note: This diagram shows test time behavior: decoder output is fed in → as next step's input.

Sequence-to-sequence is versatile! Sequence-to-sequence is useful for more than just MT.

Many NLP tasks can be phrased as sequence-to-sequence:

- Summarization (long text → short text)
- Dialogue (previous utterances → next utterance)
- Parsing (input text → output parse as sequence)
- Code generation (natural language → Python code)

The sequence-to-sequence model is an example of a Conditional Language Model.

- **Language Model** because the decoder is predicting the next word of the target sentence y
- **Conditional** because its predictions are also conditioned on the source sentence x

NMT directly calculates $P(y|x)$:

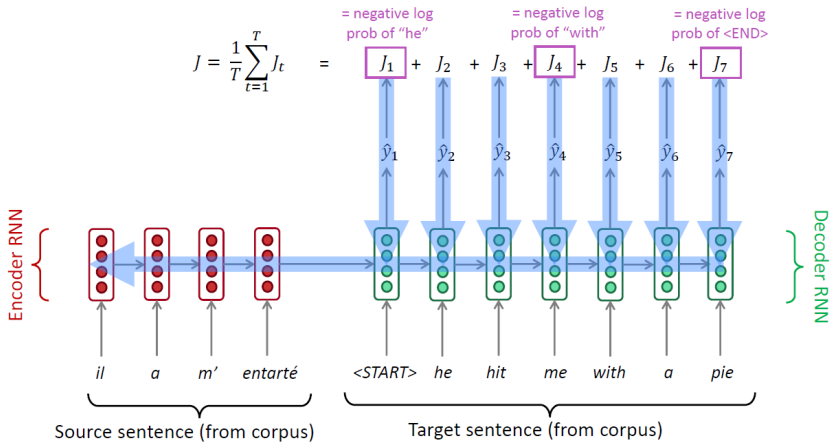


Figure 7.25: Training a Neural Machine Translation system

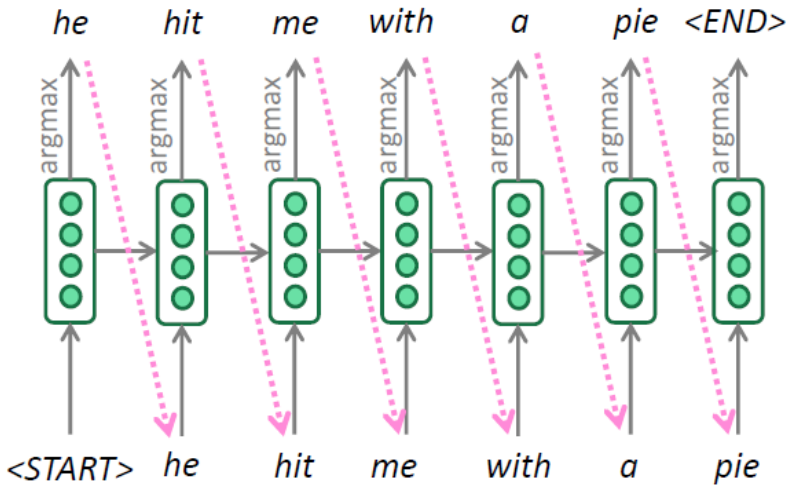


Figure 7.26: Greedy decoding

$$P(y|x) = P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)... \underbrace{P(y_T|y_1,...,y_{T-1},x)}_{\text{Probability of next target word, given target words so far and source sentence x}}$$

Question: How to train a NMT system? **Answer:** Get a big parallel corpus...

Seq2seq is optimized as a single system. Backpropagation operates "end-to-end". We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder.

This is greedy decoding (take most probable word on each step).

Problems with this method? Greedy decoding has no way to undo decisions!

- Input: il a m'entarté (he hit me with a pie)

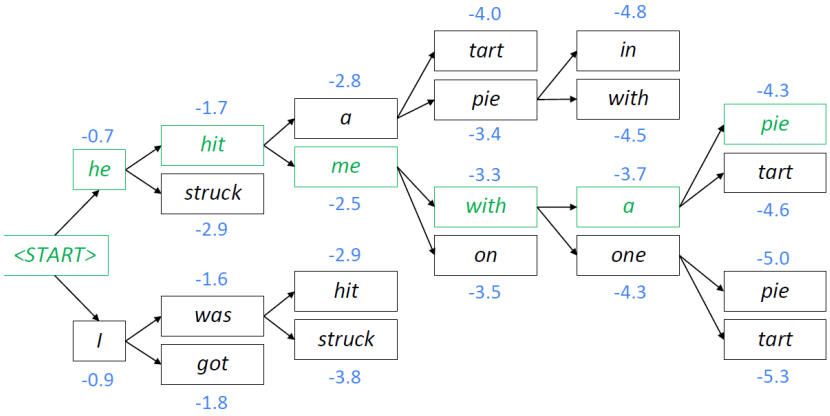


Figure 7.27: Beam search decoding

- → he _____
- → he hit _____
- → he hit a _____ (whoops! no going back now...)

How to fix this? Ideally we want to find a (length T) translation y that maximizes.

$$P(y|x) = P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_T|y_1,...,y_{T-1},x) = \prod_{t=1}^{T} P(y_t|y_1,...,y_{t-1},x)$$

We could try computing all possible sequences y. This means that on each step t of the decoder, we're tracking Vt possible partial translations, where V is vocab size. This $O(V^T)$ complexity is far too expensive!

Beam search decoding

Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)

- k is the beam size (in practice around 5 to 10)

A hypothesis $y_1,...,y_t$ has a score which is its log probability:

$$\text{score}(y_1,...,y_t) = \log P_{LM}(y_1,...,y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1,...,y_{i-1},x)$$

- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top k on each step

Beam search is not guaranteed to find optimal solution. But much more efficient than exhaustive search!

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\sum_{i=1}^{t} \log P_{LM}(y_i|y_1,...,y_{i-1},x)$

- Take top k words and compute scores
- For each of the k hypotheses, find top k next words and calculate scores

- Of these k^2 hypotheses, just keep k with highest scores
- This is the top-scoring hypothesis!
- Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces a ¡END¿ token.
  - For example: ¡START¿ he hit me with a pie ¡END¿.
- In beam search decoding, different hypotheses may produce ¡END¿ tokens on different timesteps
  - When a hypothesis produces ¡END¿, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

We have our list of completed hypotheses. How to select top one with highest score?

Each hypothesis $y_1, ..., y_t$ on our list has a score

$$\text{score}(y_1, ..., y_t) = \log P_{LM}(y_1, ..., y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, ..., y_{i-1}, x)$$

Problem with this: longer hypotheses have lower scores. Fix: Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, ..., y_{i-1}, x)$$

Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities
- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized
  - Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

Disadvantages of NMT?

Compared to SMT:

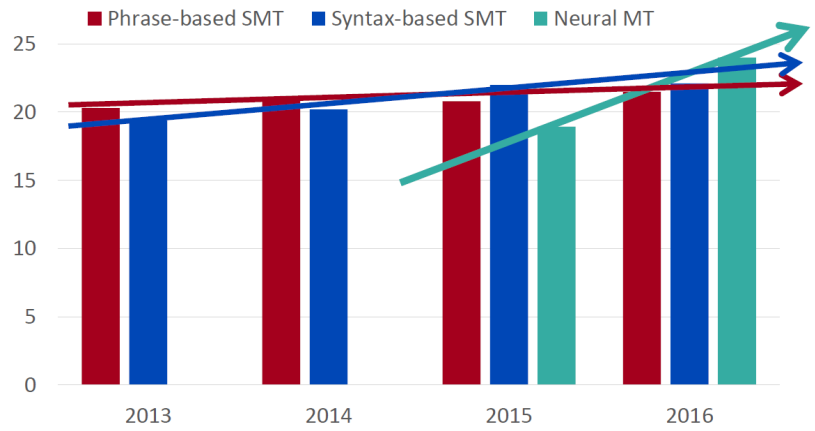


Figure 7.28: MT progress over time [Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]

- NMT is less interpretable
  - Hard to debug
- NMT is difficult to control
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

Neural Machine Translation went from a fringe research activity in 2014 to the leading standard method in 2016

- 2014: First seq2seq paper published
- 2016: Google Translate switches from SMT to NMT

This is amazing! SMT systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months.

So is Machine Translation solved?

Nope!

Many difficulties remain:

- Out-of-vocabulary words
- Domain mismatch between train and test data
- Maintaining context over longer text
- Low-resource language pairs