Coursera NLP course notes

0. Introduction

1. What is NLP

computers using natural language as input and/or ~~outp~~ output

~~Langage~~ language → [computer] → language

understanding

generation

2. Applications of NLP

① machine translation
② information extraction (unstructured text → structured data)
↳ · complex searches
· statistical queries
③ text summarization
④ dialogue system

3. Basic NLP problems

① tagging (part-of-speech, named entity, etc.)
② parsing

4. Why is ~~[redacted]~~ NLP hard?

① ambiguity
↳ · acoustic level
· syntax level (different parse trees)
· semantic level (word sense ambiguity)
· discourse level (e.g. anaphora)

1. Language modeling

1. Problem definition

Given a finite set of words V and ~~if~~ infinite set of strings V^+, we want to learn a probability distribution p such that

$$\sum_{x \in V^+} p(x) = 1,\ p(x) \geq 0 \text{ for all } x \in V^+$$

2. Application of language modeling
   ① speech recognition (and related OCR, handwriting recognition problems)
   ② the estimation techniques are useful for other problems in NLP

3. Naive method
   Suppose we have N sentences as training data, we may model the language as

$$p(x) = \frac{\text{count}(x)}{N}$$

   Any unseen sentence will have $p(x) = 0$

4. Markov process
   Given a sequence of random variables $X_1, X_2, \ldots, X_n$ $(X \in V)$, we want to model $P(X_1 = x_1, X_2 = x_2, X_3 = x_3, \ldots, X_n = x_n)$

   ① first-order Markov process

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$
$$= P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \leftarrow \text{exact (chain rule)}$$
$$= P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i \mid X_{i-1} = x_{i-1}) \leftarrow \text{Markov assumption}$$

   It assumes that $P(X_i = x_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) = P(X_i = x_i \mid X_{i-1} = x_{i-1})$

   ② second-order Markov process

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$
$$= P(X_1 = x_1) \times P(X_2 = x_2, X_1 = x_1) \times \prod_{i=3}^{n} P(X_i = x_i \mid X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$
$$= \prod_{i=1}^{n} P(X_i = x_i \mid X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

   We assume that $x_0 = x_{-1} = *$ (the "start" symbol) for convenience

   ③ modeling variable length sentences by defining $X_n = \text{STOP}$ $(\text{STOP} \notin V)$

5. Trigram language model
   It consists of ① finite set V
   ② $q(w \mid u, v)$ for each trigram $\{u, v, w\}$ such that $w \in V \cup \{\text{STOP}\}$ and $u, v \in V \cup \{*\}$

For a sentence $x_1 \dots x_n$ where $x_i \in V$ for $i = 1 \dots (n-1)$ and $x_n = STOP$

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-2}, x_{i-1})$$

6. The trigram estimation problem

We want to estimate $q(w_i | w_{i-2}, w_{i-1})$

① a natural estimate (maximum likelihood)

$$q(w_i | w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})}$$

there are in total $|V|^3$ parameters → sparse data problem

7. Evaluating a language model: perplexity

Given test sentences $s_1, s_2, \dots, s_m$, we could measure $\prod_{i=1}^{m} p(s_i)$.

more conveniently, the log probability

$$\log \prod_{i=1}^{m} p(s_i) = \sum_{i=1}^{m} \log p(s_i)$$

The usual ~~perplexity~~ measure is perplexity defined as

$$\text{perplexity} = 2^{-l} \quad \text{where} \quad l = \frac{1}{M} \sum_{i=1}^{m} p(s_i)$$

→ total number of words in S

lower perplexity is better

8. Bias-variance trade-off

Trigram maximum-likelihood estimate

$$q_{ML}(w_i | w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})}$$

low bias
high variance

Bigram maximum-likelihood estimate

$$q_{ML}(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Unigram maximum-likelihood estimate

$$q_{ML}(w_i) = \frac{count(w_i)}{count(\ )}$$

high bias
low variance

→ total word count

9. Linear interpolation

Estimate $q(W_i|W_{i-2}, W_{i-1})$ by

$$q(W_i|W_{i-2}, W_{i-1}) = \lambda_1 \times q_{ML}(W_i|W_{i-2}, W_{i-1}) + \lambda_2 \times q_{ML}(W_i|W_{i-1}) + \lambda_3 \times q_{ML}(W_i)$$

where $\lambda_1+\lambda_2+\lambda_3=1$ and $\lambda_i \geq 0$ for all i

Need to make sure the new estimate defines a distribution ($V' = V \cup \{STOP\}$)

$$\begin{aligned}
&\Sigma_{w \in V'} q(w|u,v) \\
&= \Sigma_{w \in V'} (\lambda_1 q_{ML}(W_i|W_{i-2}, W_{i-1}) + \lambda_2 q_{ML}(W_i|W_{i-1}) + \lambda_3 q_{ML}(W_i)) \\
&= \lambda_1 \Sigma_{w \in V'} q_{ML}(W_i|W_{i-2}, W_{i-1}) + \lambda_2 \Sigma_{w \in V'} q_{ML}(W_i|W_{i-1}) + \lambda_3 \Sigma_{w \in V'} q_{ML}(W_i) \\
&= \lambda_1 + \lambda_2 + \lambda_3 \\
&= 1
\end{aligned}$$

maximum likelihood defines distributions

(can also show that $q(w|u,v) \geq 0$ for all $w \in V'$)

Estimate λ by validation data, define $c'(w_1, w_2, w_3)$ to be count of trigram w_1, w_2, w_3 in validation set, maximize

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3} c'(w_1, w_2, w_3) \log q(w_3|w_1, w_2)$$

Allowing λ to vary by defining a partitioning function $\Pi(W_{i-2}, W_{i-1})$

$$\Pi = \begin{cases} 1 & \text{if } \text{count}(W_{i-2}, W_{i-1}) = 0 \\ 2 & \text{if } 1 \leq \text{count} \leq 2 \\ 3 & \text{if } 3 \leq \text{count} \leq 5 \\ 4 & \text{otherwise} \end{cases}$$

Then use different sets of λ according to the partition of W_{i-2}, W_{i-1}

10. Discounting method

The maximum-likelihood estimates are high (when count is low)

Define discounted count as

$$\text{count}^*(x) = \text{count}(x) - 0.5 \longrightarrow \text{to be cross-validated}$$

We get the missing probability mass

$$\alpha(W_{i-1}) = 1 - \sum_{w} \frac{count^*(W_{i-1}, w)}{count(W_{i-1})}$$

① Katz Back-off method (bigram)

define $A(W_{i-1}) = \{w: count(W_{i-1}, w) > 0\}$

$B(W_{i-1}) = \{w: count(W_{i-1}, w) = 0\}$

then

$$q_{BO}(W_i|W_{i-1}) = \begin{cases} \frac{count^*(W_{i-1}, W_i)}{count(W_{i-1})} & \text{if } W_i \in A(W_{i-1}) \\ \alpha(W_{i-1}) \frac{q_{ML}(w)}{\sum_{w \in B(W_{i-1})} q_{ML}(w)} & \text{if } W_i \in B(W_{i-1}) \end{cases}$$

where

$$\alpha(W_{i-1}) = 1 - \sum_{w \in A(W_{i-1})} \frac{count^*(W_{i-1}, w)}{count(W_{i-1})}$$

② Katz Back-off method (trigram)

define $A(W_{i-2}, W_{i-1}) = \{w: count(W_{i-2}, W_{i-1}, W_i) > 0\}$

$B(W_{i-2}, W_{i-1}) = \{w: count(W_{i-2}, W_{i-1}, W_i) = 0\}$

then

$$q_{BO}(W_i|W_{i-2}, W_{i-1}) = \begin{cases} \frac{count^*(W_{i-2}, W_{i-1}, W_i)}{count(W_{i-2}, W_{i-1})} & \text{if } W_i \in A(W_{i-2}, W_{i-1}) \\ \alpha(W_{i-2}, W_{i-1}) \frac{q_{BO}(W_i|W_{i-1})}{\sum_{w \in B} q_{BO}(w|W_{i-1})} & \text{if } W_i \in B(W_{i-2}, W_{i-1}) \end{cases}$$

2. Tagging problem and hidden Markov model

1. Tagging problems
   - ① part-of-speech tagging
   - ② named entity recognition

2. Two types of constraints
   - ① local: a word has more likelihood to be a noun
   - ② contextual: usually noun follows verb

   } usually in conflict

3. Supervised learning problems

   Given m training examples $x^{(1)} \ldots x^{(m)}$ and $y^{(1)} \ldots y^{(m)}$, where $x^{(i)}$ is input and $y^{(i)}$ is label

   Task is to learn a function that maps x to y

① Conditional models (~~generer~~ discrimitive model)
learn a distribution $p(y|x)$ from training examples
let $f(x) = \arg\max_y p(y|x)$

② generative models
learn a distribution $p(y, x)$ from training examples
often we have $p(x, y) = p(y)\,p(x|y)$ → noisy channel model
($p(y)$: prior, $p(x|y)$: conditional)

then we have

$$p(y|x) = \frac{p(y)\,p(x|y)}{p(x)} \quad \text{where} \quad p(x) = \sum_y p(y)\,p(x|y)$$

the output should be

$$\begin{aligned} f(x) &= \arg\max_y p(y|x) \\ &= \arg\max_y \frac{p(y)\,p(x|y)}{p(x)} \longrightarrow \text{doesn't change with } y \\ &= \arg\max_y p(y)\,p(x|y) \end{aligned}$$

4. Hidden Markov model
~~Given n wrong training examples $x^1 \dots x^n$, and n tag sequences $y^1 \dots y^n$, we want to compute $p(x^i, y^i)$~~
Given an input sentence $x = x_1, x_2, \dots, x_n$, and a tag sequence $y = y_1, y_2, \dots, y_n$
we use HMM to define $p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$

The most likely sequence is

$$\arg\max_y p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$$

① trigram hidden Markov model
Given a set of words V and a set of tags S, and a sentence $x_1 \dots x_n$
where $x_i \in V$ and a tag sequence $y_1 \dots y_{n+1}$ where $y_i \in S$ and $y_{n+1} = \text{STOP}$
we define

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i)$$

② Viterbi algorithm

Given an input $x_1, \ldots, x_n$, we want to find

$$\arg\max_y p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1})$$

brute force is too inefficient, because search space is $|S|^n$

define S_k for $k = -1 \ldots n$ to be the set of possible tags at position k

$$S_{-1} = S_0 = \{*\}$$
$$S_k = S \text{ for } k = 1 \ldots n$$

define a function r to be

$$r(y_{-1}, y_0, \ldots, y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{k} e(x_i | y_i)$$

and a dynamic programming table π to be

$\pi(k, u, v)$ = max prob. of tag sequence ending in tags u, v at position k
(e.g. max of function r)

the value of π can be calculated recursively
for $k = 1 \ldots n$, we have

$$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v | w, u) \times e(x_k | v))$$

the base case is $\pi(0, *, *) = 1$, thus the algorithm is

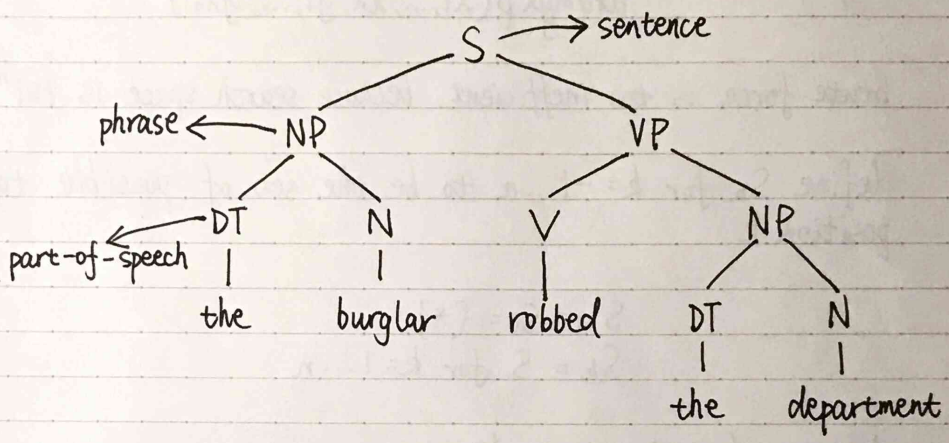
for $k = 1 \ldots n$
  for $u \in S_{k-1}$, $v \in S_k$
    compute $\pi(k, u, v)$ → remember w for book keeping
return sequence $y_1, \ldots, y_n$

asymptotic complexity is $O(n|S|^3)$

3. Parsing and context-free grammars

1. ~~Parsing~~ Parsing

Given an input sentence, we want to generate a parse tree for it.



2. Information in parse trees

① part-of-speech for each word
② phrases (e.g. noun phrase, verb phrase, etc.)
③ useful relationships (e.g. subject & verb, verb & object)

3. Context-free grammars

A context-free grammar $G = (N, \Sigma, R, S)$ where
① N is non-terminal symbols
② Σ is terminal symbols
③ R is a set of rewrite rules
④ S is the start symbol ($S \in N$)

4. Left-most derivation

A left-most derivation is obtained by always replacing the left-most non-terminal symbol with the right-hand side of a rewrite rule.

[S] → [NP VP] → [D N VP] → [the N VP] → [the man VP] → [the man Vi]
↓
[the man sleeps]

5. Properties of CFGs

① A CFG defines a set of possible derivations (can be infinite).

② A string $s \in \Sigma^*$ is in the "language" defined by the CFG if there is at least one derivation that yields s.
③ Each string in the language may have more than one derivation (ambiguity)

6. Probabilistic context-free grammars
Each rule is given a probability, for a tree t with rules

$$\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_n \to \beta_n$$

the probability is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \to \beta_i) \quad \text{where } q(\alpha \to \beta) \text{ is prob. for rule } \alpha \to \beta$$

7. Properties of PCFG
① assigns a probability to each parse tree allowed by CFG
② different parse trees for the same sentence can be ranked

8. Deriving a PCFG from a treebank
Given a treebank, we set the CFG as all rules in the corpus.
The probability of a rule is estimated by

$$q_{ML}(\alpha \to \beta) = \frac{\text{count}(\alpha \to \beta)}{\text{count}(\alpha)} \longrightarrow \text{maximum likelihood}$$

The PCFG will converge to the "true" PCFG given infinite training examples.

9. Parsing with a PCFG
Given a PCFG and a sentence s, define T(s) to be the set of possible parse trees, we want to find

$$\arg\max_{t} p(t)$$

Brute force can be very inefficient for this problem.

① Chomsky Normal Form

A context-free grammar $G = (N, \Sigma, R, S)$ is in Chomsky normal form if

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules which take one of two forms
  - $X \to Y_1 Y_2$ for $X \in N$ and $Y_1, Y_2 \in N$
  - $X \to Y$ for $X \in N$ and $Y \in \Sigma$
- S is the start symbol that $S \in N$

② CKY algorithm

Given a PCFG and a sentence s, we want

$$\max_{t \in T(s)} p(t)$$

define sentence length n, i-th word w_i, generate a dynamic programming table π where

$\pi[i,j,X]$ = max prob. of a constituent with non-terminal X spanning words $i \ldots j$ inclusive

then we have

$$\max_{t \in T(s)} p(t) = \pi[1, n, S]$$

the base case is $\pi[i,i,X] = q(X \to w_i)$, and π can be computed by

$$\pi[i,j,X] = \max_{\substack{X \to YZ \in R \\ s \in \{i \ldots (j-1)\}}} (q(X \to YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z))$$

split point ← $s \in \{i \ldots (j-1)\}$

thus the algorithm is defined by

```
for l = 1 ... (n-1)
  for i = 1 ... (n-l)
    j = i+l
    calculate π[i,j,X] for all X ∈ N → book keeping!
```

asympotatic complexity is $O(n^3|N|^3)$ for CKY algorithm

4. Lexicalized PCFGs

1. Weaknesses of PCFGs

① Lack of sensitivity to lexical information
PCFG makes very strong independent assumption. The probability for a word is only dependent on its part-of-speech.

② Lack of sensitivity to structual frequencies
Two structures may have the same probability while one appears more frequently.

2. Heads in rules of context-free grammar

Head is one of the children in a rule, it represents the important part of the rule.

Rules can be applied to recover heads in CFG rules.

The headwords can be added to a parse tree at various levels of non-terminal symbols. The annotated rewrite rules can be used to construct a new set of rules to form lexicalized CFG

3. Lexicalized CFG in Chomsky normal form

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules which take one of three forms
  - $X(h) \rightarrow_1 Y_1(h)\, Y_2(w)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
  - $X(h) \rightarrow_2 Y_1(w)\, Y_2(h)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
  - $X(h) \rightarrow h$ for $X \in N$, and $h \in \Sigma$
- $S \in N$ is the start symbol

4. Lexicalized PCFG

Similar to PCFG, each rule in the lexicalized PCFG has a probability. However, because of the added headwords in PCFG, lexicalized PCFG has much more parameters compared to normal PCFGs.

5. Parsing with lexicalized PCFGs

For a sentence of length n, this can take $O(n^3|\Sigma|^2|N|^3)$ time using CKY algorithm, because there can be $O(|\Sigma|^2|N|^3)$ rules.

Observe that for a sentence $w_1 \dots w_n$ of length n, at most $O(n^2|N|^3)$ rules are applicable (lexical items not in w can be discarded). Thus, the runtime can be reduced to $O(n^5|N|^3)$

6. Charniak model for parameter estimation

① decompose a parameter into product of two parameters

$$q(S(saw) \rightarrow_2 NP(man)\ VP(saw))$$
$$= q(S \rightarrow_2 NP\ VP \mid S, saw) \times q(man \mid S \rightarrow_2 NP\ VP, saw)$$

② use smoothed estimation for the two parameters

$$q(S \rightarrow_2 NP\ VP \mid S, saw)$$
$$= \lambda_1 \times q_{ML}(S \rightarrow_2 NP\ VP \mid S, saw) + \lambda_2 \times q_{ML}(S \rightarrow_2 NP\ VP \mid S)$$

$$q(man \mid S \rightarrow_2 NP\ VP, saw)$$
$$= \lambda_3 \times q_{ML}(man \mid S \rightarrow_2 NP\ VP, saw) + \lambda_4 \times q_{ML}(man \mid S \rightarrow_2 NP\ VP) +$$
$$\lambda_5 \times q_{ML}(man \mid NP)$$

There are other important details including
- Convert rules to Chomsky normal form
- Incorporate POS in non-terminals (can be useful)
- Encode preferences for close attachment

7. Evaluation of lexicalized PCFG

Tree can be represented by constituents (part-of-speech not included). The label (non-terminal), start point, and end point is recorded.

Let's define

G = number of constituents in gold standard
P = number in parse output
C = number correct

The precision and recall can be calculated as

$$\text{recall} = 100\% \times \frac{C}{G}$$

$$\text{precision} = 100\% \times \frac{C}{P}$$

Another way to evaluate is to define "dependency" as a tuple of headword, modifier word, and the rule. (location of word is recorded) The number of words in a sentence equals to the count of dependencies.

Accuracy can be calculated on dependencies, and precision/recall can be calculated for a particular dependency type.

5. Machine translation
   1. Challenges in MT
      ① lexical ambiguity
      ② differing word orders
      ③ structure not preserved across translation
      ④ syntactic ambiguity causes problems
      ⑤ pronoun resolution

   2. Classical MT
      ① Direct machine translation
         - translate word-by-word
         - little analysis of source
         - rely on bilingual dictionary
         - simple reordering rules are applied

      ② transfer-based approach
         ① Analysis: analyse the source language
         ② Transfer: convert source parse tree to target-language parse tree
         ③ Generation: convert parse tree to output sentence

      ③ interlingual-based approach
         ① analysis: analyze source and generate independent representation of its meaning
         ② generation: convert meaning into output sentence

3. Noisy channel model

Suppose we translate French into English, and we have $p(e|f)$ which estimates the probability of a English sentence e given the French sentence f. A noisy channel model has two components

$p(e)$ the language model
$p(f|e)$ the translation model

then we have

$$p(e|f) = \frac{p(e,f)}{p(f)} = \frac{p(e)\,p(f|e)}{\sum_e p(e)\,p(f|e)}$$

and

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

4. IBM model 1

To model $p(f|e)$ given English sentence e with l words $e_1 \dots e_l$, and french sentence f with m words $f_1 \dots f_m$, define alignment a which English word each French word originated from

$$a = \{a_1, \dots, a_m\} \text{ where } a_i \in \{0 \dots l\}$$

there are $(l+1)^m$ possible alignments

define model for $p(a|e,m)$ and $p(f|a,e,m)$ then

$$p(f,a|e,m) = p(a|e,m)\,p(f|a,e,m)$$

$$p(f|e,m) = \sum_{a \in A} p(a|e,m)\,p(f|a,e,m)$$

where A is the set of all possible alignments, we also have

$$p(a|f,e,m) = \frac{p(f,a|e,m)}{\sum_{a \in A} p(f,a|e,m)}$$

which gives the most likely alignment a^*

$$a^* = \arg\max_a p(a|f,e,m)$$

the model is mostly used for recovering alignments nowadays

we define $p(a|e,m)$ as

$$p(a|e,m) = \frac{1}{(l+1)^m} \longrightarrow \text{simplifying assumption}$$

and $p(f|a,e,m)$ as

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

therefore the final result is

$$p(f,a|e,m) = \frac{1}{(l+1)^m} \prod_{j=1}^{m} t(f_j|e_{a_j})$$

5. IBM model 2

It differs from IBM model 1 only on alignment, define

$q(i|j,l,m)$ = prob. that j-th French word is connected with i-th English word given sentence length l and m

then we have

$$p(a|e,m) = \prod_{j=1}^{m} q(a_j|j,l,m)$$

and that

$$p(f,a|e,m) = \prod_{j=1}^{m} q(a_j|j,l,m)\, t(f_j|e_{a_j})$$

we can recover the most likely alignment a^* where

$$a_j^* = \arg\max_{a_i} q(a_i|j,l,m)\, t(f_j|a)$$

6. Parameter estimation

Given sentence pairs $(e^{(k)}, f^{(k)})$ for $k = 1 \dots n$, each $e^{(k)}$ is a English sentence and each $f^{(k)}$ is a French sentence. We want to estimate $t(f|e)$ and $q(i|j,l,m)$. The key challenge is that there is no alignment for any pairs.

① Estimation with alignment

If we have the alignment, we have

$$t_{ML}(f|e) = \frac{count(e,f)}{count(e)}$$

$$q_{ML}(j|i,l,m) = \frac{count(j,i,l,m)}{count(i,l,m)}$$

② Estimation with EM algorithm

The EM algorithm calculates q and t in an iterative manner. First a random guess for q and t is used. Then the count is calculated. And it's used to re-estimate q and t.

We define a function $\delta(k,i,j)$ as

$$\delta(k,i,j) = \frac{q(j|i,l_k,m_k) * t(f_i^{(k)}|e_j^{(k)})}{\sum_{j=0}^{l_k} q(j|i,l_k,m_k) * t(f_i^{(k)}|e_j^{(k)})}$$

the algorithm is defined as

initialization: set q and t to random value

for $s = 1 \dots S$ → number of iterations
  clear all counts
  for $k = 1 \dots n$
    for $i = 1 \dots m_k$, for $j = 0 \dots l_k$
      update the counts with δ
  recalculate q and t with max likelihood

If we define a log-likelihood function

$$L(t,q) = \sum_{k=1}^{n} \log(f^{(k)} | e^{(k)}) = \sum_{k=1}^{n} \log \sum_{a} p(f^{(k)}, a | e^{(k)})$$

the EM algorithm will converge to a local maximum

6. Phrase-based translation models

1. Phrase-based ~~model~~ lexicon

A phrase-based lexicon is used and probability is learned for translating one phrase to another phrase (instead of words). Given training sentence pairs, phrase pairs can be extracted using alignments from the IBM model.

2. Alignment matrix representation

Alignments can be represented using matrices.

	Le	chien	rit
The	•		
dog		•	
Laughs			•

3. Finding alignment matrices

For the alignments in IBM models, there exist a many-to-one relationship. Also, the alignment could be noisy. A better way to get the alignment is to train the model in both directions. And take the intersect of the two alignments as the starting point to generate the final alignment.

4. Phrase pair extraction

A phrase-pair is consistent if

① there is at least one word in e aligned to a word in f
② there are no words in f aligned to word outside e
③ there are no words in e aligned to word outside f

All consistent phrase pairs will be extracted.

5. Probabilities for phrase pairs

We can estimate the probability for a phrase pair e,f by

$$t(e|f) = \frac{count(f,e)}{count(f)}$$

6. Phrase-based model definition

A phrase-based model consists of

① A phrase-based lexicon with each entry given a score

$$g(f,e) = \log\left(\frac{count(f,e)}{count(e)}\right)$$

② A trigram language model

③ A distortion parameter η (typically negative)

7. Phrase-based translation

Given a foreign sentence $x_1, \ldots, x_n$, tuple (s,t,e) is defined for any phrase in the sentence. In (s,t,e), s is the start point, t is the end point, and e is the corresponding English phrase. P is the set of all phrases in a sentence.

Define derivation y to be a finite sequence $p_1, \ldots p_L$ where p_j for $j \in \{1 \ldots L\}$ is in P. L can be any positive integer. The derivation defines the translation $e(y)$, for example

$y = (1,3,$ we must also$), (7,7,$ take$), (4,5,$ this criticism$), (6,6,$ seriously$)$

$\Downarrow$

$e(y) =$ we must also take this criticism seriously

A derivation $p_1 \ldots p_L$ is valid if

① p_k for $k \in \{1 \ldots L\}$ is in P

② each word is translated exactly once

③ for $k \in \{1 \ldots L-1\}$, $|t(p_k) + 1 - s(p_{k+1})| \leq d$ where $d \geq 0$ is a parameter. also, we have $|1 - s(p_1)| \leq d$.

→ distortion limit

To score any derivation, we define a function f where

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=0}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

$h(e(y))$: log of the trigram model; $g(p_k)$: phrase score; $\eta \times |t(p_k) + 1 - s(p_{k+1})|$: distortion penalty

The derivation with highest score is selected as the translation.

8. The decoding algorithm

Define a tuple (e_1, e_2, b, r, α), where e_1 and e_2 are English words, b is a bit string of length n, r is the end point of the last phrase, and α is the score for the state.

The initial state is $q_0 = (*, *, 0^n, 0, 0)$

0^n: bit string with n zeros

The search space can be formulated as a directed state transition graph. Define $ph(q)$ for a state to be the set of allowed phrases. A phrase is allowed if

① p doesn't overlap with bit-string b

② distortion limit is not violated

Also define $next(q, p)$ to be the state formed by combining state q and phrase p, and $eq(q, q')$ which returns true if e_1, e_2, b, and r are all equal.

Given a sentence $x_1 \ldots x_n$, the algorithm is defined as

Initialization: set $Q_0 = \{q_0\}$, $Q_i = \emptyset$ for $i = 1 \ldots n$

Q_i: all states with i words translated

for $i = 0 \ldots n-1$

  for each $q \in beam(Q_i)$, for each phrase $p \in ph(q)$

    $q' = next(q, p)$

    add (Q_i, q', q, p) where $i = len(q')$

return highest scoring state in Q_n, sequence can be found with back pointers

the function add is defined as

if $\exists q'' \in Q$ and $eq(q', q'') = \text{True}$:
  if $\alpha(q'') < \alpha(q')$:
    $Q = \{q'\} \cup Q \setminus \{q''\}$
else
  $Q = Q \cup \{q'\}$

and function beam is defined as

$$beam(Q) = \{q \in Q : \alpha(q) \geq \alpha^* - \beta\} \quad \text{where } \alpha^* = \max_q \alpha(q)$$

→ beam width $\beta \geq 0$

7. Log-linear models

1. General problem

Given input domain X and a finite label set Y, we want to compute the conditional probability $p(y|x)$ for any x, y where $x \in X$, $y \in Y$

2. Feature vector representation

A feature is a function $f_k(x,y) \in \mathbb{R}$ (often binary feature $f_k(x,y) \in \{0,1\}$)

Given m features f_k for $k = 1 \dots m$, it can be represented as a vector $f(x,y) \in \mathbb{R}^m$ for any x, y

3. Feature vector in practice

For language modeling, a feature could be defined as

$$f(x,y) = \begin{cases} 1 & \text{if } y = \text{model and } w_{i-1} = \text{statistical} \\ 0 & \text{otherwise} \end{cases}$$

→ bigram feature

For POS tagging

$$f(x,y) = \begin{cases} 1 & \text{if } w_i = \text{base and } y = Vt \\ 0 & \text{otherwise} \end{cases}$$

→ $p(\text{base}|Vt)$

Usually, the label y is taken into account. If all features are

binary features, the output is a bit-string. It's often sparse with relatively few 1's than 0's.

4. Parameter vector

Given m features, also define a parameter vector $v \in \mathbb{R}^m$. Then for each (x, y) pair, the score could be defined as

$$v \cdot f(x, y) = \sum_{k=1}^{m} v_k f_k(x, y)$$

5. Log-linear model

Given feature vector $f(x, y)$ and parameter vector v, the probability could be defined as

$$p(y|x; v) = \frac{e^{v \cdot f(x,y)}}{\sum_{y' \in Y} e^{v \cdot f(x,y')}}$$

It can be proven that

$$\sum_{y \in Y} p(y|x; v) = 1$$

so that $p(y|x; v)$ defines a distribution

Taking log on $p(y|x; v)$ gives

$$\log p(y|x; v) = v \cdot f(x, y) - \log \sum_{y' \in Y} e^{v \cdot f(x,y')}$$

linear term — normalization term

6. Parameter estimation

Given training examples $(x^{(i)}, y^{(i)})$ for $i = 1 \ldots n$, estimate v to be

$$v_{ML} = \underset{v \in \mathbb{R}^m}{\operatorname{argmax}} L(v) \quad \text{where}$$

$$\text{concave} \longleftarrow L(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v) = \sum_{i=1}^{n} v \cdot f(y^{(i)}|x^{(i)}; v) - \sum_{i=1}^{n} \log \sum_{y' \in Y} e^{v \cdot f(x,y')}$$

Gradient descent could be used to optimize $L(v)$, taking the derivative of $L(v)$ gives

$$\frac{dL(v)}{dv_k} = \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y' \in Y} f_k(x^{(i)}, y') p(y' | x^{(i)}; v)$$

empirical count — expected count

and $dL(v)/dv$ is a vector of derivatives, the full algorithm is defined as

initialization: $v = 0$

iterate until convergence:
- calculate $\Delta = dL(v)/dv$
- calculate $\beta^* = \text{argmax}_\beta L(v + \beta\Delta)$
- set $v = v + \beta^* \Delta$

since vanilla gradient ascent is slow, conjugate gradient is usually adopted

7. Smoothing in log-linear model

For a given feature that is satisfied every time in training examples, the max likelihood solution satisfies

$$\sum_i f_k(x^{(i)}, y^{(i)}) = \sum_i \sum_y p(y | x^{(i)}; v) f_k(x^{(i)}, y)$$

indicating that
- $p(y | x^{(i)}; v) = 1$ for any example
- $v_k \to \infty$ (most likely)

this causes overfitting, to regularize, define $L(v)$ to be

$$L(v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in Y} e^{v \cdot f(x^{(i)}, y')} - \frac{\lambda}{2} \sum_{k=1}^{m} v_k^2$$

→ penalty for large weight

and

$$dL(v)/dv_k = \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y' \in Y} f_k(x^{(i)}, y') p(y' | x^{(i)}; v) - \lambda v_k$$

8. Log-linear model for tagging (Max-entropy Markov model)

1. Model definition

Given input sentence $W_{[1:n]}$ and tag sequence$_{[1:n]}$, we want to use log-linear model to define

$$p(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n) \longrightarrow \text{conditional probability}$$

and then the best tag sequence is

$$t^*_{[1:n]} = \arg\max_{t_{[1:n]}} p(t_{[1:n]} | W_{[1:n]})$$

A trigram log-linear model could be defined as

$$p(t_{[1:n]} | W_{[1:n]}) = \prod_{j=1}^{n} p(t_j | w_1 \ldots w_n, t_{j-1}, \ldots, t_1) \longrightarrow \text{chain rule}$$

$$= \prod_{j=1}^{n} p(t_j | w_1 \ldots w_n, t_{j-2}, t_{j-1}) \longrightarrow \text{independence assumption}$$

2. Representation: histories

A history is a 4-tuple $\langle t_{-2}, t_{-1}, W_{[1:n]}, i \rangle$

- t_{-2}, t_{-1} are the last two tags
- $W_{[1:n]}$ is the sentence
- i is the current position

define X to be the set of all tuples (input domain for features)

3. Features for tagging problem

Given the input format as 4-tuple, and all possible tags as Y, we could define features like

① word/tag feature for all word/tag pairs
② spelling feature for prefix/suffix of length ≤4
③ contextual feature (e.g. bigram)

4. The Viterbi algorithm

The Viterbi algorithm could be used to generate the most likely sequence given the input sentence

define the function r

$$r(t_1 \dots t_k) = \prod_{i=1}^{k} q(t_i | t_{i-2}, t_{i-1}, w_{[1-n]}, i)$$

define a dynamic programming table

$\pi(k, u, v)$ = max prob. of a tag sequence ending in tags u, v at position k

$$= \max_{\langle t_1 \dots t_{k-2} \rangle} r(t_1 \dots t_{k-2}, u, v)$$

the base case is $\pi(0, *, *) = 1$, and for $k \in \{1 \dots n\}$, $u \in S_{k-1}$, $v \in S_k$

$$\pi(k, u, v) = \max_{t \in S_{k-2}} (\pi(k-1, t, u) \times q(v | t, u, w_{[1-n]}, k))$$

use backpointers to recover the sequence with max probability

9. (Conditional) History-based model

1. Model definition

History-based model could be used to calculate $p(T|S)$ for a parse tree T (or any other structure)

The steps are

① represent a tree as a sequence of decisions $d_1 \dots d_m$

$$T = \langle d_1, d_2, \dots, d_m \rangle$$

② the probability $p(T|S)$ is

$$p(T|S) = \prod_{i=1}^{m} p(d_i | d_1, \dots, d_{i-1}, S)$$

→ no Markov assumption

③ train a log-linear model to estimate $p(d_i | d_1 \dots d_{i-1}, S)$

④ use beam or heuristic search to recover the sequence

2. Representing tree as decision sequence

A parse tree could be represented as a three-layer structure

① part-of-speech tags
the first n decisions are the POS tag for the sentence

② ~~chunks~~ chunks
any phrase where all children are POS tags are chunks

next n decisions are chunk tag for each word (e.g. Start(NP))

③ remaining structure
alternate between two classes of actions
- Join(X) and Start(X)
- check=YES or check=NO

these form the remaining decisions in the decision sequence

3. Feature selection
The set of features are different depending on whether the next decision is
- POS tag
- chunk tag
- Start or Join tag
- check=YES or check=NO

4. Searching
Because no Markov assumption is used, dynamic programming can't be applied to search for the best decision sequence. Instead, beam search could be used for this purpose.

10. Global linear model
1. Three components in GLM
① a function f that maps (x, y) to a feature vector $f(x,y) \in \mathbb{R}^d$
② a function GEN that maps x to candidates GEN(x)
③ a parameter vector $v \in \mathbb{R}^d$

2. Features
The function f is defined by a set of functions $h_1 \ldots h_d$

$$f(x) = \langle h_1(x), h_2(x), \ldots, h_d(x) \rangle$$

and f outputs a feature vector

3. GEN

GEN enumerates a set of candidates for a sentence, depending on the application, GEN(x) could be very different. GEN(x) can be a very large set sometimes.

4. Parameter vector

The parametor vector v has the same dimension as feature vector $f(x,y)$. A score can be calculated with f and v by taking the dot product $f \cdot v$ which maps a candidate solution to a real value

5. GLM

Given input sentence x, function f and GEN, and parameter vector v, we want to compute the best output

$$F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}} f(x,y) \cdot v$$

6. GLM for parsing

The reranking approach could be applied for parsing.

First, we select a base model, for example lexicalized PCFG. Then, define GEN(x) to be the top N (e.g. 25, 50, 100) results from the base model for input x.

The ground truth is set as y for any sentence x, or the closest parse tree could also be used as y.

As for feature, we could select features related to a CFG rule, bigram, grandparent rule, and two-level rules.

7. Parameter estimation

A variant of the perceptron algorithm is used

initialization: set $v=0$

$\rightarrow$ iteration count

for $t=1 \ldots T$, $i=1 \ldots n$

$z_i = F(x_i)$

if $(z_i \neq y_i)$

$v = v + f(x_i, y_i) - f(x_i, z_i)$

return v

II. GLM for tagging

1. Model definition

Given input sentence $W_{[1:n]} = \{w_1, \ldots, w_n\}$, define T to be the set of all possible tags. Let $GEN(W_{[1:n]})$ to be T^n, which is exponential in the sentence length.

To define the feature vector f, first define local feature vector g with the 4-tuple history representation. There will be n histories in a sentence of length n. Therefore, there will be n local feature vector return for a sentence. Now define the global vector f to be

$$f(W_{[1:n]}, t_{[1:n]}) = \sum_{i=1}^{n} g(h_i, t_i)$$

then we have

$$F(W_{[1:n]}) = \arg \max_{t_{[1:n]} \in GEN(W_{[1:n]})} v \cdot f(W_{[1:n]}, t_{[1:n]})$$

$$= \arg \max_{t_{[1:n]} \in GEN(W_{[1:n]})} v \cdot \sum_{i=1}^{n} g(h_i, t_i)$$

$$= \arg \max_{t_{[1:n]} \in GEN(W_{[1:n]})} \sum_{i=1}^{n} v \cdot g(h_i, t_i)$$

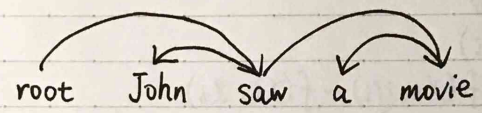
$\rightarrow$ local score

which can be computed using the Viterbi algorithm

12. GLM for dependency parsing

1. Dependency parsing

Dependency parse is an alternative to syntactic parse. Each dependency is a pair (h, m) where h is the index of the head word, and m is the index of the modifier word. In the figure, h points to m. root is a special symbol.



In the above figure, there is no label for dependencies. Dependencies can also be assigned labels (e.g. SBJ, OBJ, etc.).

2. Conditions on dependency structure

- The dependency arcs form a directed tree, with the root symbol at the root of the tree
- There are no "crossing dependencies" (projective structure)

3. Dependency parsing resource

- Hand-annotated corpus is available for certain languages
- A treebank can be used to construct dependency bank through lexicalization

4. Model definition

Given a sentence x, let GEN(x) return all possible dependency parses for sentence x (exponential in size).

Define feature vector f to be the sum of local feature vector g.

$$f(x,y) = \sum_{(h,m) \in y} g(x,h,m)$$

For local feature vector g, we could define features based on the words, the POS tags, and the distance between x_h and x_m.

Similar to GLM for tagging, $F(x)$ could be calculated efficiently using dynamic programming ($O(n^3)$).

13. Brown clustering algorithm

1. Algorithm description

Given a large corpus of words, the algorithm partitions the words into word clusters, or a hierarchichal word clusters.

The intuition is that similar words appear in similar context (distribution of words to their immediate left and right).

2. Algorithm formulation

Define V to be the set of words seen in the corpus. A function $C: V \to \{1 \dots k\}$ maps each word to a cluster (in total k clusters). Then we model the corpus as

$$p(w_1, \dots w_T) = \prod_{i=1}^{n} e(w_i \mid C(w_i)) q(C(w_i) \mid C(w_{i-1}))$$

more conveniently

$$\log p(w_1, \dots, w_T) = \sum_{i=1}^{n} \log e(w_i \mid C(w_i)) q(C(w_i) \mid C(w_{i-1}))$$

we want to maximize this probability.

3. Measuring the quality of C

Define the quality of C to be

$$\text{Quality}(C) = \sum_{i=1}^{n} \log e(w_i \mid C(w_i)) q(C(w_i) \mid C(w_{i-1}))$$

$$= \sum_{c=1}^{k} \sum_{c'=1}^{k} p(c,c') \log \frac{p(c,c')}{p(c)p(c')} + G$$

$\log \frac{p(c,c')}{p(c)p(c')}$ → mutual information

G → constant

where

$$p(c,c') = \frac{n(c,c')}{\sum_{c,c'} n(c,c')} \qquad p(c) = \frac{n(c)}{\sum_{c} n(c)}$$

4. A first algorithm

Start with $|V|$ clusters, and find k final clusters by merging in $|V| - k$ steps. We pick two clusters to merge each time which

maximizes quality of C after merging.

Naive implementation runs in $O(|V|^5)$, improved implementation runs in $O(|V|^3)$. These are too slow.

5. A second algorithm
Define a parameter m (e.g. m=1000). Take the top m most frequent words into its own clusters $c_1 \dots c_m$.

For $i = (m+1) \dots |V|$
create a new cluster c_{m+1} for the i-th most frequent word
merge two clusters from $c_1 \dots c_{m+1}$ which maximizes Quality(C)
Carry out (m-1) final merges to create a full hierarchy

The asymptotic complexity is $O(|V|m^2+n)$ where n is corpus length

6. Clusters in named entity recognition
In Miller et al, NAACL 2004, a log-linear model is used for named entity recognition. Besides standard features in the model, features related to the brown corpus is also used. For example, it looks at the first 8, 16, 20 bits for current, previous and next words.

Active learning also improves the performance by a lot.