StatML (Chapter 7): Language Models

Chunqi Shi

Brandeis University

shicq@brandeis.edu

October 15, 2014

Overview

1. Language Models
2. N-Gram Language Models
3. Smoothing
4. Interpolation & Back-off
5. Size of Language Models
6. Concept of Nenural Network Language Model
7. End

Language Models

Why?

Language models answer the question:

How likely is a string of English words good English?

What?

A statistical language model assigns a probability to a sequence of m words $P(w_1, \ldots, w_m)$ by means of a probability distribution.

How?

- Reordering:

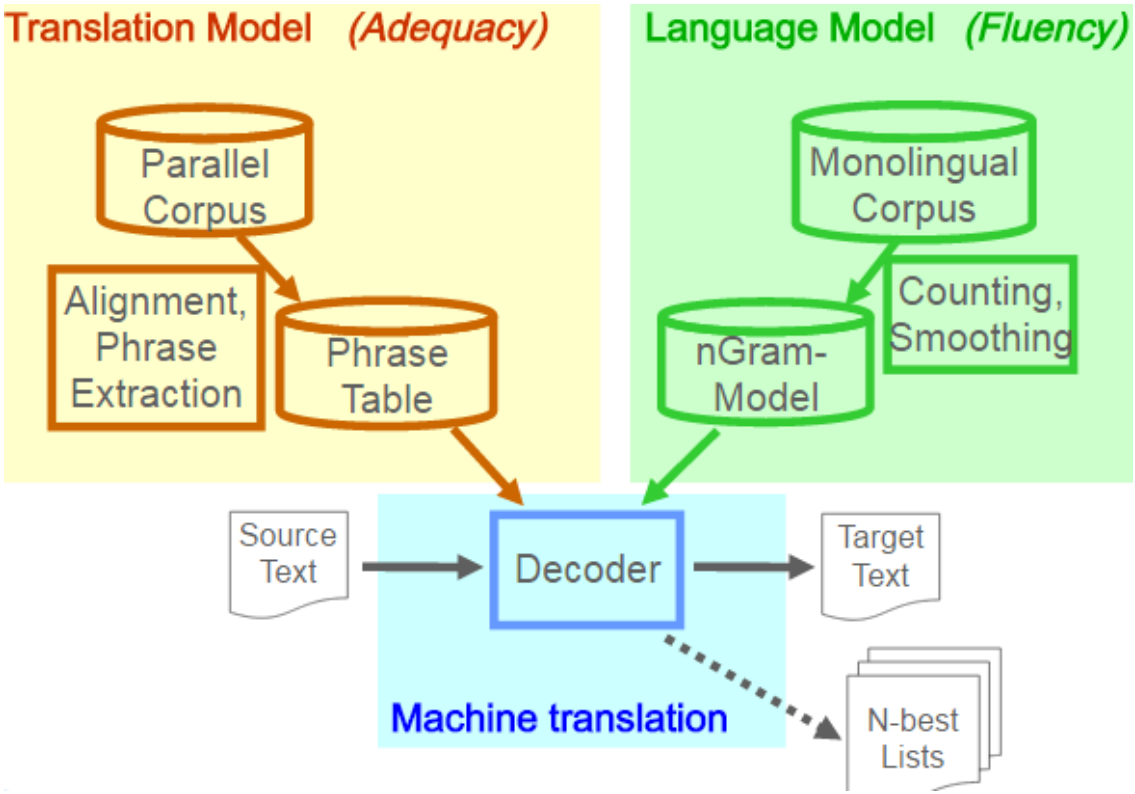
  $P_{LM}(\text{the house is small}) > P_{LM}(\text{small the is house})$

- Word Choice:

  $P_{LM}(\text{I am going home}) > P_{LM}(\text{I am going house})$

Language Models & SMT Architecture

How language models work in a basic SMT architecture[1]?

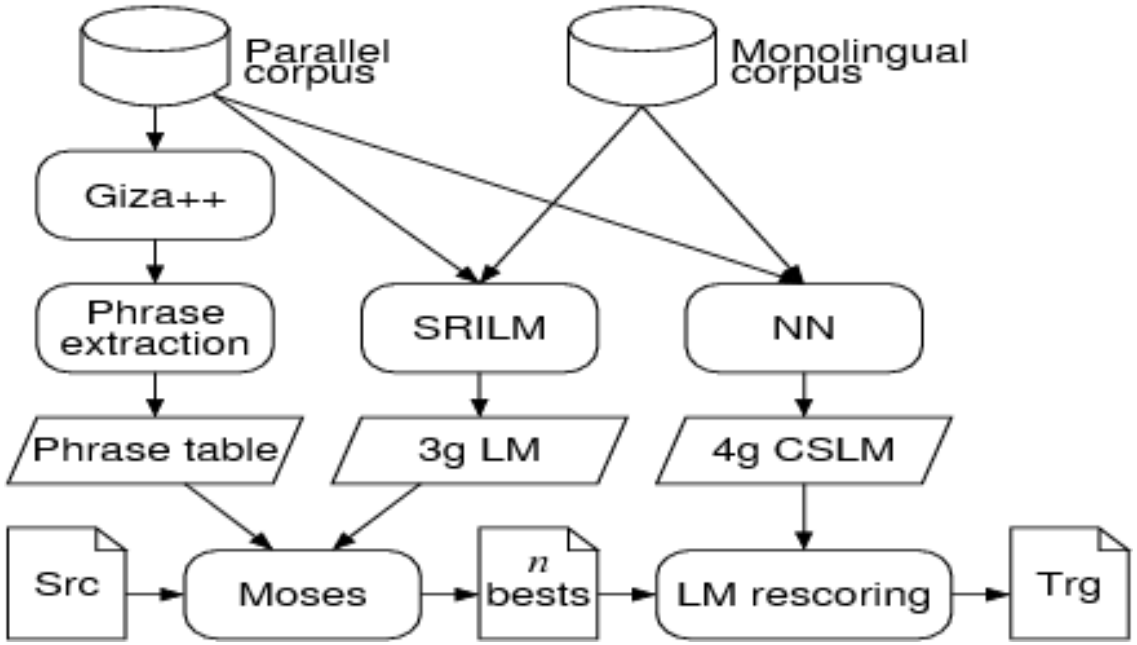


[1] http://slideplayer.us/slide/203403/

Open Source Language Models Example

Architecture of the LIMSI SMT system[2] and open language models:

- SRILM[3] (N-Gram) & NN[Neural Networks] (Continuous Space LM).
- Giza++: Translation Model.
- Moses: Decoder



NNLM

[2] http://www.limsi.fr/tlp/mt/

[3] http://sourceforge.net/projects/irstlm/

Other Language Models Applications

Speech Recognition

$P_{LM}(\text{I saw a van}) > P_{LM}(\text{eyes awe of an})$

Spell Correction

The office is about fifteen minuets from my house.

$P_{LM}(\text{about fifteen minutes from}) > P_{LM}(\text{about fifteen minuets from})$

Information Retrieval

No results found for "University of Brandeis" (Query likelihood model).

$P_{LM}(\text{University of Brandeis}) > P_{LM}(\text{Brandeis University})$

More !!

Part-of-speech Tagging, Parsing, Summarization, Question-Answering, etc.

Probabilistic Language Modeling

How to Compute $P(W)$

$$P(W) = P(w_1, \ldots, w_m)$$

Probability of an upcoming word

$$P(w_k|w_1, w_2, \ldots, w_{k-1})$$

Decomposing using Chain Rule

$$P(w_1, \ldots, w_m) =$$
$$P(w_1)P(w_2|w_1)P(w_2|w_1, w_2)\ldots P(w_m|w_1, w_2, \ldots, w_{m-1})$$

Example

$$P(\text{its water is so transparent}) =$$
$$P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water}) \times P(\text{so}|\text{its water is}) \times$$
$$P(\text{transparent}|\text{its water is so})$$

Chain Rule Estimation

Joint Probability

$$P(w_1 w_2 \dots w_m) = \prod P(w_i | w_1 w_2 \dots w_i - 1)$$

How to estimate?

Maximum likelihood estimation:

$$P(\text{transparent}|\text{its water is so}) =$$

$$\frac{Count(\text{its water is so transparent})}{Count(\text{its water is so})}$$

Problems?

- Sparse data: NO enough data for estimating.
- Large space: HUGE possible sentences.

Markov Chain

Markov Assumption

Only previous history matters:

$P(\text{transparent}|\text{its water is so}) = (\text{transparent}|\text{so})$ or maybe

$P(\text{transparent}|\text{its water is so}) = (\text{transparent}|\text{so})$

k_{th} Order Markov Model

$$P(w_1 w_2 \dots w_m) = \prod P(w_i | w_{i-k} w_2 \dots w_i - 1)$$

Simple Cases

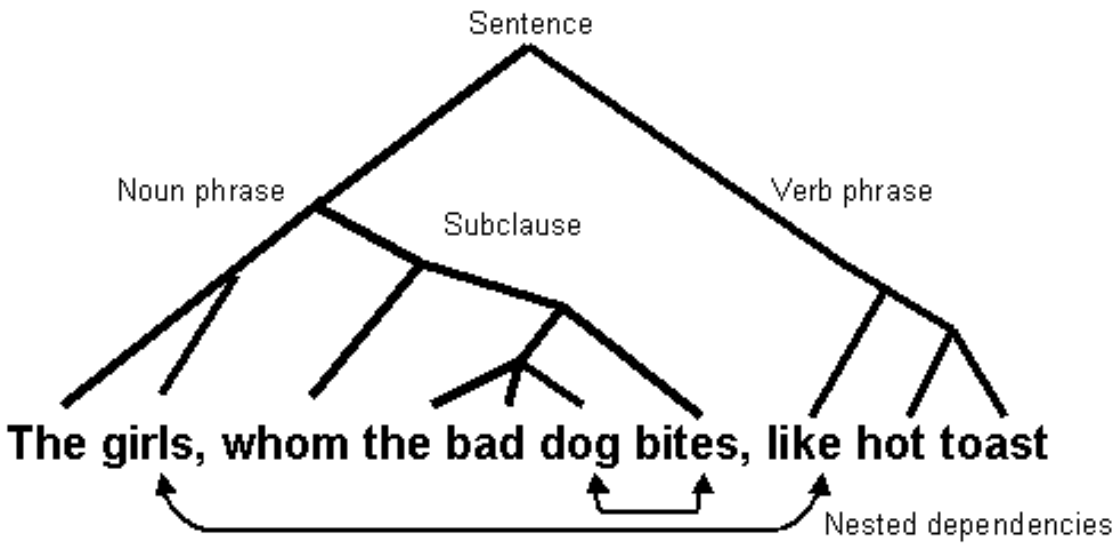
Unigram model: $P(w_1 w_2 \dots w_m) = \prod P(w_i)$

Bigram model: $P(w_1 w_2 \dots w_m) = \prod P(w_i | w_{i-1})$

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

Bigram Example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$$

Trigram Example

- Counts for trigrams and estimated word probabilities

the green (total: 1748)

word	c.	prob.
paper	801	0.458
group	640	0.367
light	110	0.063
party	27	0.015
ecu	21	0.012

the red (total: 225)

word	c.	prob.
cross	123	0.547
tape	31	0.138
army	9	0.040
card	7	0.031
,	5	0.022

the blue (total: 54)

word	c.	prob.
box	16	0.296
.	6	0.111
flag	6	0.111
,	3	0.056
angel	3	0.056

  - 225 trigrams in the Europarl corpus start with the red
  - 123 of them end with cross

$\rightarrow$ maximum likelihood probability is $\frac{123}{225} = 0.547$.

"The red cross" and "The green party" are frequent trigrams in the Europarl corpus.

Evaluation of N-gram Models

How good is our model?

Extrinsic Evaluation: training A & B, testing, comparing accuracy of A & B by evaluation metric.
But it is time-consuming.

Intrinsic Evaluation

Perplexity: How well can we predict the next word?
Intrinsic evaluation is Bad approximation! Unless the test data looks just like the training data.
But is helpful to think about.

Intuition of Perplexity

How hard is the task of recognizing digits "0, 1, 2, 3, 4, 5, 6, 7, 8, 9"?
Perplexity 10.

Perplexity

Cross Entropy

$$H(W) = -\frac{1}{n}\log P(w_1 w_2 \ldots w_n)$$
$$= -\frac{1}{n}\sum_{i}^{n}\log P(w_i|w_1 \ldots, w_{i-1})$$

Perplexity:

$$PP(W) = 2^{H(W)} = P(W)^{-\frac{1}{n}}$$

Perplexity as branching factor

$$PP(W) = P(1,\ 2,\ \ldots,\ 10)^{-\frac{1}{10}}$$
$$= (\frac{1}{10})^{10\times-\frac{1}{10}} = (\frac{1}{10})^{-1} = 10$$

Comparison N-gram Models

Minimizing perplexity is the same as maximizing probability, thus better model.

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

Generalization and Zeros

Unseen N-grams

Things that NOT ever occur in the training set. But occur in the test set.

Training Set:

1. ... denied the allegations
2. ... denied the reports
3. ... denied the claims
4. ... denied the request

Test Set:

1. ... denied the offer
2. ... denied the loan

$$P(\text{“offer”} \mid \text{“denied the”}) = 0$$

Smoothing

Sparse statistics, smoothing to generalize better.

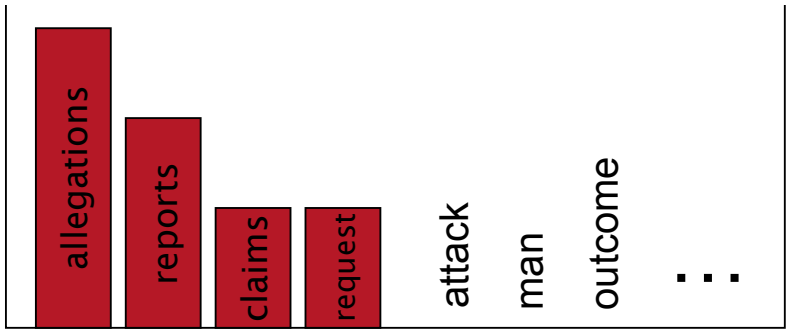
Smoothing

How to smooth all words non-zeros

- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request

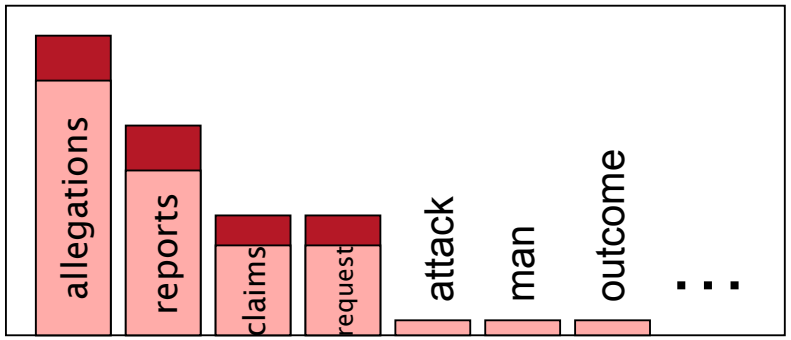
  7 total



- Steal probability mass to generalize better

  P(w | denied the)
  2.5 allegations
  1.5 reports
  0.5 claims
  0.5 request
  2 other

  7 total



Smoothing Methods

1. Additive Smoothing: Laplace (18th century) came up with this smoothing technique when he tried to estimate the chance that the sun will rise tomorrow (sunrise problem).
2. Good-Turing Estimate: Alan Turing and his assistant I. J. Good (1953) as part of their efforts to crack German ciphers for the Enigma machine during World War II.
3. Jelinek-Mercer Smoothing (linear interpolation): Previous IBM workers (1980) Frederick Jelinek and Robert Mercer received ACL Lifetime Achievement Award (The Dawn of Statistical ASR and MT) in June 2014.

Smoothing Methods

1. Katz Smoothing (back-off): Slava M. Katz (1987) proposed non-linear estimation of probabilities from sparse data for the language model component of a speech recogniser.
2. Witten-Bell Smoothing: Ian H. Witten and Timothy C. Bell (1991) proposed estimating the probabilities of novel events in adaptive text compression.
3. Kneser-Ney Smoothing: Kneser and Ney (1995) proposed "improved backing-off for m-gram language modeling", evolved from absolute-discounting interpolation, which is one of the best smoothing methods for n-gram language.

Add-One Smoothing

Laplace smoothing

Pretend we saw each word one more time than we did.

- For all possible n-grams, add the count of one.

$$p = \frac{c+1}{n+v}$$

  - c = count of n-gram in corpus
  - n = count of history
  - v = vocabulary size

- But there are many more unseen n-grams than seen n-grams

- Example: Europarl 2-bigrams:
  - $86,700$ distinct words
  - $86,700^2 = 7,516,890,000$ possible bigrams
  - but only about $30,000,000$ words (and bigrams) in corpus

Bigram Add-One Smoothing

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Add-α Smoothing

Will α adjusted count a lot?

- Add $\alpha < 1$ to each count

$$p = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?
- Could be optimized on held-out set

Comparison of Add-α Smoothing

Bigram in Europarl corpus

Count	Adjusted count		Test count
c	$(c+1)\frac{n}{n+v^2}$	$(c+\alpha)\frac{n}{n+\alpha v^2}$	t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

- Add-α smoothing with $\alpha = 0.00017$
- t_c are average counts of n-grams in test set that occurred c times in corpus

Held-out Estimation

Long Tail of N-gram Counting

- 1,266,566 bigrams in this corpus, more than half, 753,777, occur only once.
- Zipf's Law: the frequency of any word is inversely proportional to its rank in the frequency table.

If we observe an n-gram c times in the training corpus, ow often do we expect it to see in the future (Held-out Estimation)?

$p_h(w_1 w_2 \dots w_n) = \frac{E[r]}{N_h} \sim \frac{T_r}{N_r \times N_h}$, $r = count_h(w_1 w_2 \dots w_n)$

Cross-Validation Estimation

$$E(r) = \frac{T_r^1 + T_r^2}{N_r^1 + N_r^2}$$

Deleted Estimation

Deleted Estimation: leave one part of the training corpus out for validation.

Count r	Count of counts N_r	Count in held-out T_r	Exp. count $E[r] = T_r/N_r$
0	7,515,623,434	938,504	0.00012
1	753,777	353,383	0.46900
2	170,913	239,736	1.40322
3	78,614	189,686	2.41381
4	46,769	157,485	3.36860
5	31,413	134,653	4.28820
6	22,520	122,079	5.42301
8	13,586	99,668	7.33892
10	9,106	85,666	9.41129
20	2,797	53,262	19.04992

	Adjusted count		
Count c	$(c+1)\frac{n}{n+v^2}$	$(c+\alpha)\frac{n}{n+\alpha v^2}$	Test count t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

Good-Turing Smoothing Intuition

- You are fishing (a scenario from Josh Goodman), and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?
  - 1/18
- How likely is it that next species is new (i.e. catfish or bass)
  - Let's use our estimate of things-we-saw-once to estimate the new things.
  - 3/18 (because N_1=3)
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18
  - How to estimate?

Good-Turing Smoothing Calculation

$$P^*_{GT}(\text{things with zero frequency}) = \frac{N_1}{N} \qquad c^* = \frac{(c+1)N_{c+1}}{N_c}$$

- Unseen (bass or catfish)
  - c = 0:
  - MLE p = 0/18 = 0
  - P^*_{GT} (unseen) = N_1/N = 3/18
- Seen once (trout)
  - c = 1
  - MLE p = 1/18
  - C*(trout) = 2 * N2/N1
    = 2 * 1/3
    = 2/3
  - P^*_{GT}(trout) = 2/3 / 18 = 1/27

Leave One Out Intuition

- Intuition from leave-one-out validation
  - Take each of the c training words out in turn
  - c training sets of size c–1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c
  - What fraction of held-out words are seen k times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count k
  - There are N_k words with training count k
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
  - ...or expected count:

$$k^* = \frac{(k+1)N_{k+1}}{N_k}$$

Good-Turing for 2-Grams in Europarl

Count	Count of counts	Adjusted count	Test count
r	N_r	r^*	t
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

adjusted count fairly accurate when compared against the test count

approximation.

Derivation of Good-Turing

- A specific n-gram α occurs with (unknown) probability p in the corpus
- Assumption: all occurrences of an n-gram α are independent of each other
- Number of times α occurs in corpus follows binomial distribution

$$p(c(\alpha) = r) = b(r; N, p_i) = \binom{N}{r} p^r (1-p)^{N-r}$$

Derivation of Good-Turing (2)

- Goal of Good-Turing smoothing: compute *expected count* c^*
- Expected count can be computed with help from binomial distribution:

$$\begin{aligned} E(c^*(\alpha)) &= \sum_{r=0}^{N} r\, p(c(\alpha) = r) \\ &= \sum_{r=0}^{N} r \binom{N}{r} p^r (1-p)^{N-r} \end{aligned}$$

- Note again: p is unknown, we cannot actually compute this

Derivation of Good-Turing (3)

- Definition: expected number of n-grams that occur r times: $E_N(N_r)$
- We have s different n-grams in corpus
  - let us call them $\alpha_1, ..., \alpha_s$
  - each occurs with probability $p_1, ..., p_s$, respectively
- Given the previous formulae, we can compute

$$\begin{aligned} E_N(N_r) &= \sum_{i=1}^{s} p(c(\alpha_i) = r) \\ &= \sum_{i=1}^{s} \binom{N}{r} p_i^r (1 - p_i)^{N-r} \end{aligned}$$

- Note again: p_i is unknown, we cannot actually compute this

Derivation of Good-Turing (4)

- Reflection
  - we derived a formula to compute $E_N(N_r)$
  - we have N_r
  - for small r: $E_N(N_r) \simeq N_r$
- Ultimate goal compute expected counts c^*, given actual counts c

$$E(c^*(\alpha)|c(\alpha) = r)$$

Derivation of Good-Turing (5)

- For a particular n-gram α, we know its actual count r
- Any of the n-grams α_i may occur r times
- Probability that α is one specific α_i

$$p(\alpha = \alpha_i | c(\alpha) = r) = \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- Expected count of this n-gram α

$$E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N \; p_i \; p(\alpha = \alpha_i | c(\alpha) = r)$$

Derivation of Good-Turing (6)

- Combining the last two equations:

$$E(c^*(\alpha)|c(\alpha) = r) = \sum_{i=1}^{s} N\ p_i\ \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

$$= \frac{\sum_{i=1}^{s} N\ p_i\ p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- We will now transform this equation to derive Good-Turing smoothing

Derivation of Good-Turing (7)

- Repeat:

$$E(c^*(\alpha)|c(\alpha) = r) = \frac{\sum_{i=1}^{s} N\ p_i\ p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- Denominator is our definition of expected counts $E_N(N_r)$

Derivation of Good-Turing (8)

- Numerator:

$$\sum_{i=1}^{s} N\ p_i\ p(c(\alpha_i) = r) = \sum_{i=1}^{s} N\ p_i \binom{N}{r} p_i^r (1-p_i)^{N-r}$$

$$= N\ \frac{N!}{N-r!r!} p_i^{r+1} (1-p_i)^{N-r}$$

$$= N\ \frac{(r+1)}{N+1} \frac{N+1!}{N-r!r+1!} p_i^{r+1} (1-p_i)^{N-r}$$

$$= (r+1)\ \frac{N}{N+1} E_{N+1}(N_{r+1})$$

$$\simeq (r+1)\ E_{N+1}(N_{r+1})$$

Derivation of Good-Turing (9)

- Using the simplifications of numerator and denominator:

$$\begin{aligned} r^* &= E(c^*(\alpha)|c(\alpha) = r) \\ &= \frac{(r+1)\ E_{N+1}(N_{r+1})}{E_N(N_r)} \\ &\simeq (r+1)\frac{N_{r+1}}{N_r} \end{aligned}$$

- QED

Intuition

Good-Turing allows us to estimate the probability mass assigned to n-grams with lower counts by looking at the number of n-grams with higher counts.

Interpolation & Back-off

Sparseness in a Trigram Model

1. Back to bigrams, otherwise unigram: Back-off
2. Mix that model with bigram and unigram: Interpolation

Context

1. Back-off: some times it helps to use less context condition on less context for contexts you haven't learned much about.
2. Interpolation: higher and lower order n-gram models have different strengths and weaknesses:
   - high-order n-grams are sensitive to more context, but have sparse counts.
   - low-order n-grams consider only very limited context, but have robust counts

Interpola1on (Jelinek-Mercer Smoothing)

Simple interpolation

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + (1-\lambda_1-\lambda_2)P(w_n)$$

Recursive Interpolation

Lambdas conditional on context:

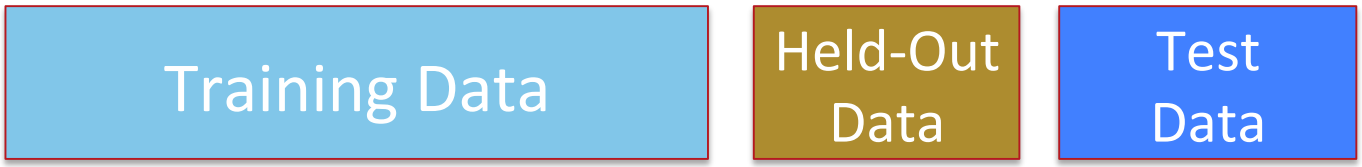
$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-1}w_{n-2}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + (1-\lambda_1(w_{n-2}^{n-1})-\lambda_2(w_{n-2}^{n-1}))P(w_n)$$

$$\sum_{w\in V}\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1(w_{n-2}^{n-1})\sum_{w\in V}P(w_n|w_{n-1}w_{n-2}) + \lambda_2(w_{n-2}^{n-1})\sum_{w\in V}P(w_n|w_{n-1}) + (1-\lambda_1(w_{n-2}^{n-1})-\lambda_2(w_{n-2}^{n-1}))\sum_{w\in V}P(w_n)$$

$$= \lambda_1(w_{n-2}^{n-1}) + \lambda_2(w_{n-2}^{n-1}) + (1-\lambda_1(w_{n-2}^{n-1})-\lambda_2(w_{n-2}^{n-1})) = 1$$

How to Set the Lambdas?

- Use a **held-out** corpus



- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

$$\log P(w_1 ... w_n \mid M(\lambda_1 ... \lambda_k)) = \sum_i \log P_{M(\lambda_1 ... \lambda_k)}(w_i \mid w_{i-1})$$

Intuition

Jelinek and Mercer smoothing uses the lower order n-grams in combination with maximum likelihood estimation.

Back-Off

Prediction Model

Models are defined recursively in terms of lower order models.

$$p_n^{\mathrm{BO}}(w_i|w_{i-n+1},...,w_{i-1})$$
$$= \begin{cases} d_n(w_{i-n+1},...,w_{i-1})\, p_n(w_i|w_{i-n+1},...,w_{i-1}) \\ \qquad \text{if } \mathrm{count}_n(w_{i-n+1},...,w_i) > 0 \\ \alpha_n(w_{i-n+1},...,w_{i-1}))\, p_{n-1}^{\mathrm{BO}}(w_i|w_{i-n+2},...,w_{i-1}) \\ \qquad \text{else} \end{cases}$$

Discounting function: $d_n(w_{i-n+1},\ldots,w_{i-1})$
Back-off Weight: $\alpha_{w_{i-n+1}\cdots w_{i-1}}$

Katz Smoothing (Back-Off with Good-Turing)

Katz Adjusted Probability:

Discounting function

If Good-Turing smoothing adjusts C into C^* and $P(w_i|w_{i-n+1}\cdots w_{i-1}) = \frac{C(w_{i-n+1}\ldots w_{i-1}w_i)}{C(w_{i-n+1}\cdots w_{i-1})}$, then

$$d = \frac{C^*}{C}$$

Back-off Weight

$$\beta_{w_{i-n+1}\cdots w_{i-1}} = 1 - \sum_{\{w_i:C(w_{i-n+1}\cdots w_i)>k\}} d_{w_{i-n+1}\cdots w_i} \frac{C(w_{i-n+1}\ldots w_{i-1}w_i)}{C(w_{i-n+1}\cdots w_{i-1})}$$

$$\alpha_{w_{i-n+1}\cdots w_{i-1}} = \frac{\beta_{w_{i-n+1}\cdots w_{i-1}}}{\sum_{\{w_i:C(w_{i-n+1}\cdots w_i)\leq k\}} P_{bo}(w_i|w_{i-n+2}\cdots w_{i-1})}$$

Katz Adjusted Counts

- Katz adjusted counts:

$$c_{katz}(w_{i-1}^{i}) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1}) p_{ML}(w_i) & \text{if } r = 0 \end{cases}$$

- $\alpha(w_{i-1})$ is chosen so that $\sum_{w_i} c_{katz}(w_{i-1}^{i}) = \sum_{w_i} c(w_{i-1}^{i})$:

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : c(w_{i-1}^{i}) > 0} p_{katz}(w_i | w_{i-1})}{1 - \sum_{w_i : c(w_{i-1}^{i}) > 0} p_{ML}(w_i)}$$

- Compute $p_{katz}(w_i | w_{i-1})$ from corrected count by normalizing:

$$p_{katz}(w_i | w_{i-1}) = \frac{c_{katz}(w_{i-1}^{i})}{\sum_{w_i} c_{katz}(w_{i-1}^{i})}$$

Bigrams Katz Smoothing

- What about d_r? Large counts are taken to be reliable, so $d_r = 1$ for $r > k$, where Katz suggests $k = 5$. For $r \leq k$...
- We want discounts to be proportional to Good-Turing discounts:

$$1 - d_r = \mu(1 - \frac{r^*}{r})$$

- We want the total count mass saved to equal the count mass which Good-Turing assigns to zero counts:

$$\sum_{r=1}^{k} n_r(1 - d_r)r = n_1$$

- The unique solution is:

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$$

Diversity of Predicted Words

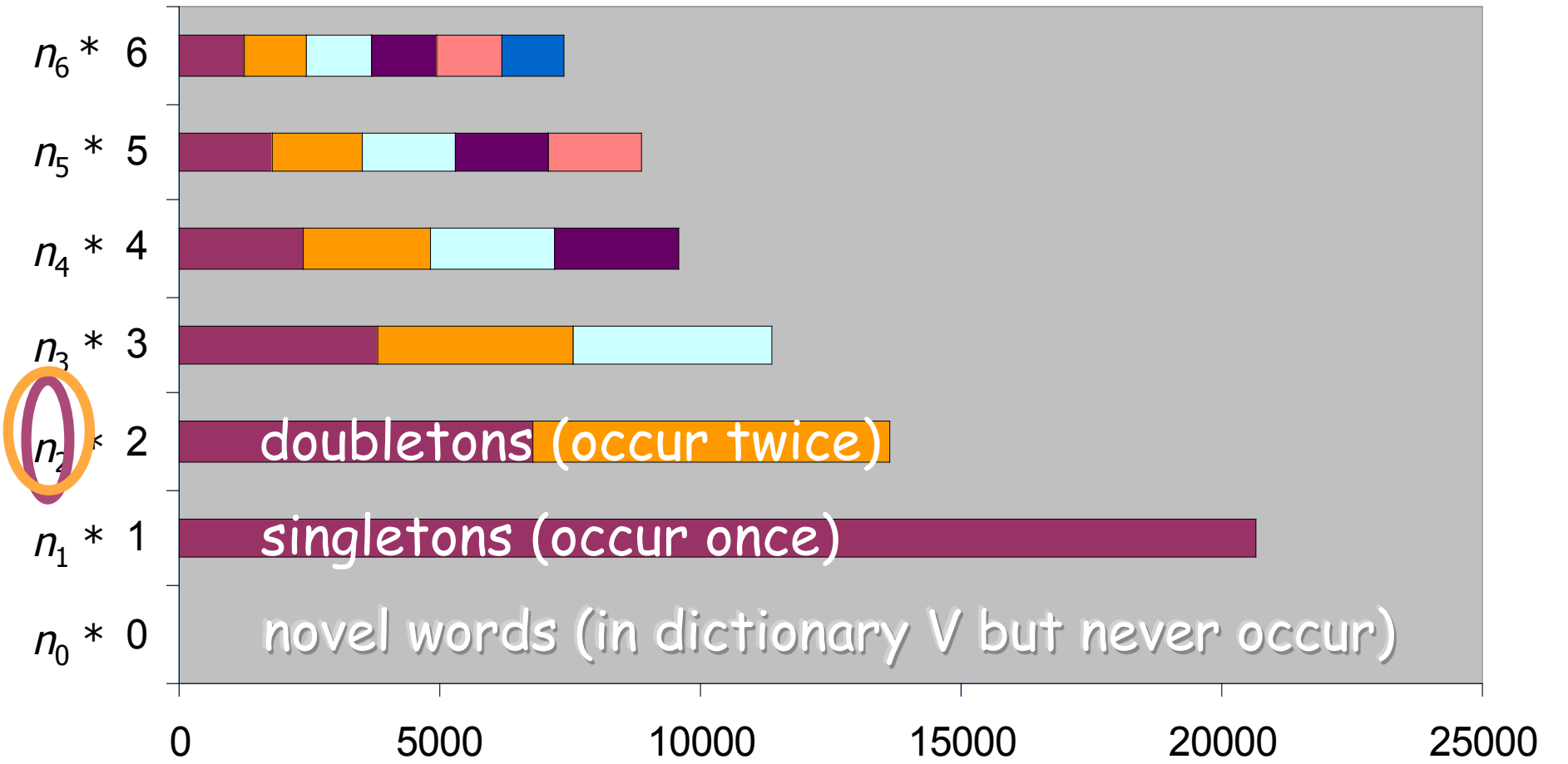
- Consider the bigram histories spite and constant
  - both occur 993 times in Europarl corpus
  - only 9 different words follow spite
    almost always followed by of (979 times), due to expression in spite of
  - 415 different words follow constant
    most frequent: and (42 times), concern (27 times), pressure (26 times),
    but huge tail of singletons: 268 different words

- More likely to see new bigram that starts with constant than spite

Witten-Bell

Think of unseen events as ones not having happened yet; The probability of this event, when it happens, can be modeled by the probability of seeing it for the first time.

First Time Words



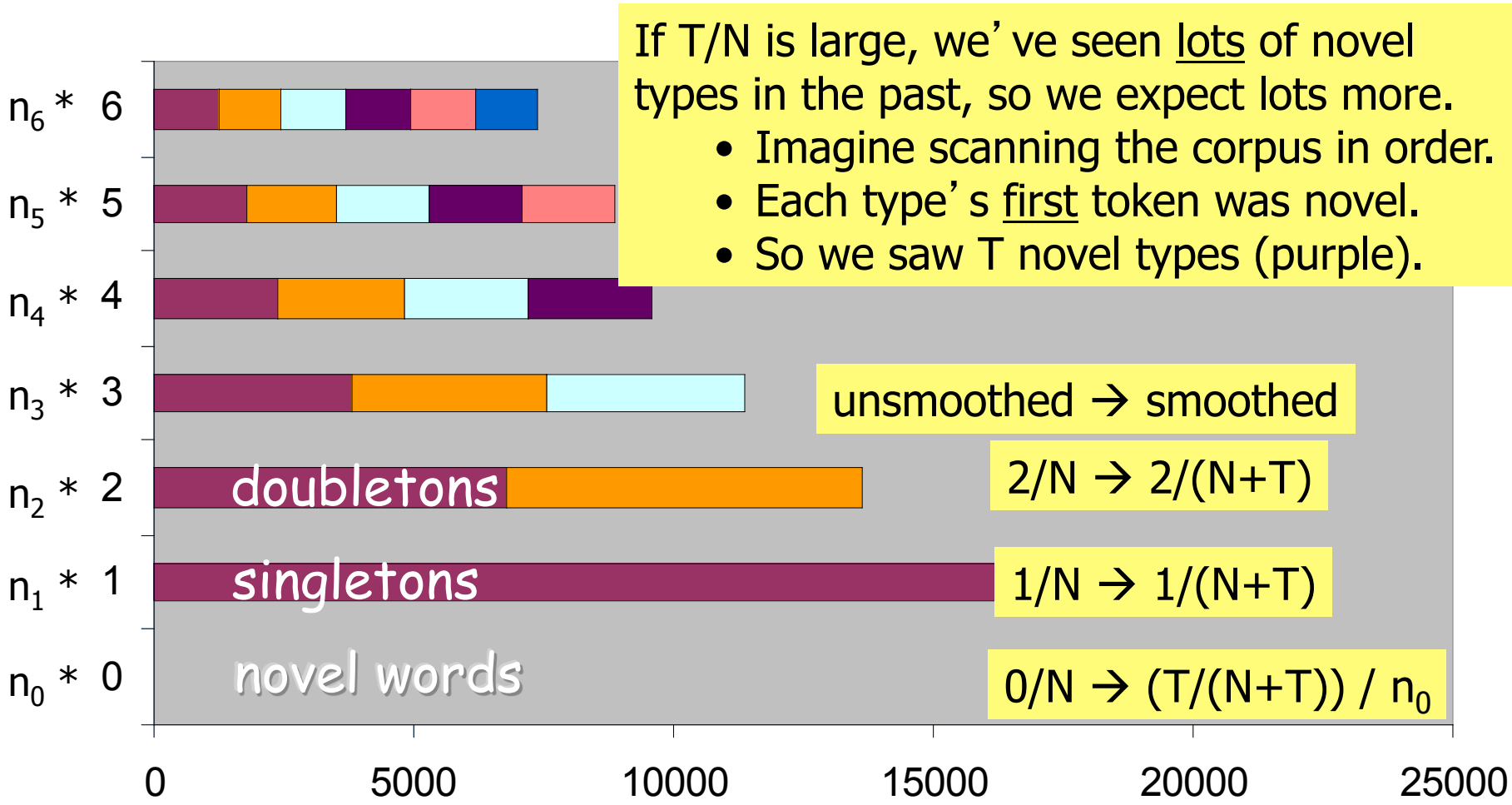
n2 = # doubleton types

n2 * 2 = # doubleton tokens

$\sum_r n_r$ = total # types = T (purple bars)

$\sum_r (n_r * r)$ = total # tokens = N (all bars)

First Time Estimation



Intuition: When we see a new *type* w in training, ++count(w); ++count(novel)
So *p*(novel) is estimated as T/(N+T), divided among n_0=*Z* specific novel types

Witten-Bell Smoothing

An Instance of Jelinek-Mercer Smoothing

Recursive interpolation method of using higher-order model.

$$P^{WB}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) P^{WB}(w_i|w_{i-n+2}^{i-1})$$

- Number of possible extensions of a history $w_1, ..., w_{n-1}$ in training data

$$N_{1+}(w_1, ..., w_{n-1}, \bullet) = |\{w_n : c(w_1, ..., w_{n-1}, w_n) > 0\}|$$

- Lambda parameters

$$1 - \lambda_{w_1,...,w_{n-1}} = \frac{N_{1+}(w_1, ..., w_{n-1}, \bullet)}{N_{1+}(w_1, ..., w_{n-1}, \bullet) + \sum_{w_n} c(w_1, ..., w_{n-1}, w_n)}$$

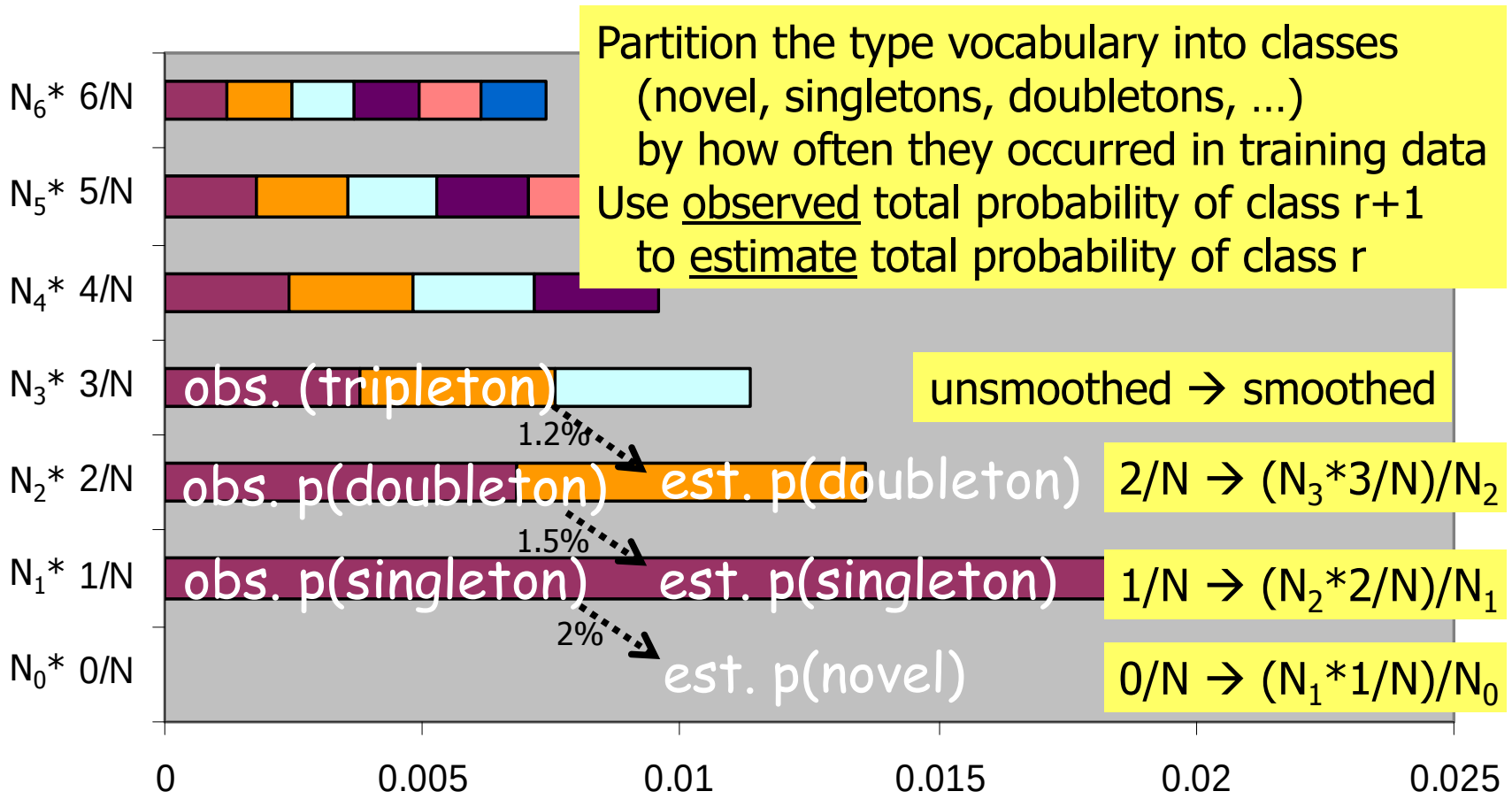
Witten-Bell Smoothing Example

Let us apply this to our two examples:

$$1 - \lambda_{spite} = \frac{N_{1+}(\text{spite}, \bullet)}{N_{1+}(\text{spite}, \bullet) + \sum_{w_n} c(\text{spite}, w_n)}$$
$$= \frac{9}{9 + 993} = 0.00898$$

$$1 - \lambda_{constant} = \frac{N_{1+}(\text{constant}, \bullet)}{N_{1+}(\text{constant}, \bullet) + \sum_{w_n} c(\text{constant}, w_n)}$$
$$= \frac{415}{415 + 993} = 0.29474$$

Comparing to Good-Turing Smoothing



$r/N = (N_r * r/N)/N_r \quad \rightarrow \quad (N_{r+1} * (r+1)/N)/N_r$

Diversity of History

- Consider the word York
  - fairly frequent word in Europarl corpus, occurs 477 times
  - as frequent as foods, indicates and providers

  → in unigram language model: a respectable probability

- However, it almost always directly follows New (473 times)

- Recall: unigram model only used, if the bigram model inconclusive
  - York unlikely second word in unseen bigram
  - in back-off unigram model, York should have low probability

Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account
- Count of histories for a word

$$N_{1+}(\bullet w) = |\{w_i : c(w_i, w) > 0\}|$$

- Recall: maximum likelihood estimation of unigram language model

$$p_{ML}(w) = \frac{c(w)}{\sum_i c(w_i)}$$

- In Kneser-Ney smoothing, replace raw counts with count of histories

$$p_{KN}(w) = \frac{N_{1+}(\bullet w)}{\sum_{w_i} N_{1+}(w_i w)}$$

Absolute Discounting Interpolation

Good Turing Smoothing is used in Katz smoothing. If we look into it, might save some time by calculating. ▸ Good Turing
Chen and Goodman suggested absolute discounting.

- Absolute discounting: subtract a fixed D from all non-zero counts

$$\alpha(w_n|w_1,...,w_{n-1}) = \frac{c(w_1,...,w_n) - D}{\sum_w c(w_1,...,w_{n-1},w)}$$

- Refinement: three different discount values

$$D(c) = \begin{cases} D_1 & \text{if } c = 1 \\ D_2 & \text{if } c = 2 \\ D_{3+} & \text{if } c \geq 3 \end{cases}$$

Parameters of Absolute Discounting

- Optimal discounting parameters D_1, D_2, D_{3+} can be computed quite easily

$$
\begin{aligned}
Y &= \frac{N_1}{N_1 + 2N_2} \\
D_1 &= 1 - 2Y\frac{N_2}{N_1} \\
D_2 &= 2 - 3Y\frac{N_3}{N_2} \\
D_{3+} &= 3 - 4Y\frac{N_4}{N_3}
\end{aligned}
$$

- Values N_c are the counts of n-grams with exactly count c

Kneser-Ney Smoothing

$$p_I(w_1 n|w_1, ..., w_{n-1})$$

$$= \begin{cases} \alpha(w_n|w_1, ..., w_{n-1}) & \text{if } c(w_1, ..., w_n) > 0 \\ \gamma(w_1, ..., w_{n-1})\, p_I(w_n|w_2, ..., w_{n-1}) & \text{otherwise} \end{cases}$$

- Recall: base on count of histories $N_{1+}(\bullet w)$ in which word may appear, not raw counts.

$$\alpha(w_n|w_1, ..., w_{n-1}) = \frac{N_{1+}(\bullet w_1, ..., w_n) - D}{\sum_w N_{1+}(\bullet w_1, ..., w_{n-1}, w)}$$

- Again, three different values for D (D_1, D_2, D_{3+}), based on the count of the history $w_1, ..., w_{n-1}$

Lower Order Formula for $\gamma(d)$

- Probability mass set aside from seen events

$$d(w_1, ..., w_{n-1}) = \frac{\sum_{i \in \{1,2,3+\}} D_i N_i(w_1, ..., w_{n-1}\bullet)}{\sum_{w_n} c(w_1, ..., w_n)}$$

- N_i for $i \in \{1, 2, 3+\}$ are computed based on the count of extensions of a history $w_1, ..., w_{n-1}$ with count 1, 2, and 3 or more, respectively.

- Similar to Witten-Bell smoothing

Interpolated Back-off

- Back-off models use only highest order n-gram
  - if sparse, not very reliable.
  - two different n-grams with same history occur once $\rightarrow$ same probability
  - one may be an outlier, the other under-represented in training
- To remedy this, always consider the lower-order back-off models
- Adapting the α function into interpolated α_I function by adding back-off

$$\alpha_I(w_n|w_1,...,w_{n-1}) = \alpha(w_n|w_1,...,w_{n-1}) + d(w_1,...,w_{n-1})\, p_I(w_n|w_2,...,w_{n-1})$$

- Note that d function needs to be adapted as well

Interpolation & Back-off

- Both interpolation (Jelinek-Mercer) and backoff (Katz) involve combining information from higher- and lower-order models.
- Key difference: in determining the probability of n-grams with nonzero counts, interpolated models use information from lower-order models while backoff models do not.
- In both backoff and interpolated models, lower-order models are used in determining the probability of n-grams with zero counts (sparsity).
- It turns out that it's not hard to create a backoff version of an interpolated algorithm, and vice-versa. (Kneser-Ney was originally backoff; Chen & Goodman made interpolated version.)

Summary of Smoothing

By Chen & Goodman (1998)

- The factor with the largest influence is the use of a modified backoff distribution as in Kneser-Ney smoothing.
- Jelinek-Mercer performs better on small training sets; Katz performs better on large training sets.
- Katz smoothing performs well on n-grams with large counts; Kneser-Ney is best for small counts.
- Absolute discounting is superior to linear discounting.
- Interpolated models are superior to backoff models for low (nonzero) counts.
- Adding free parameters to an algorithm and optimizing these parameters on held-out data can improve performance.

Evaluation

Evaluation of smoothing methods:

Perplexity for language models trained on the Europarl corpus

Smoothing method	**bigram**	**trigram**	**4-gram**
Good-Turing	96.2	62.9	59.9
Witten-Bell	97.1	63.8	60.4
Modified Kneser-Ney	95.4	61.6	58.6
Interpolated Modified Kneser-Ney	94.5	59.3	54.0

Managing the Size of the Model

- Millions to billions of words are easy to get
  (trillions of English words available on the web)

- But: huge language models do not fit into RAM

Number of Unique N-Grams

Number of unique n-grams in Europarl corpus

29,501,088 tokens (words and punctuation)

Order	**Unique n-grams**	**Singletons**
unigram	86,700	33,447 (38.6%)
bigram	1,948,935	1,132,844 (58.1%)
trigram	8,092,798	6,022,286 (74.4%)
4-gram	15,303,847	13,081,621 (85.5%)
5-gram	19,882,175	18,324,577 (92.2%)

$\rightarrow$ remove singletons of higher order n-grams

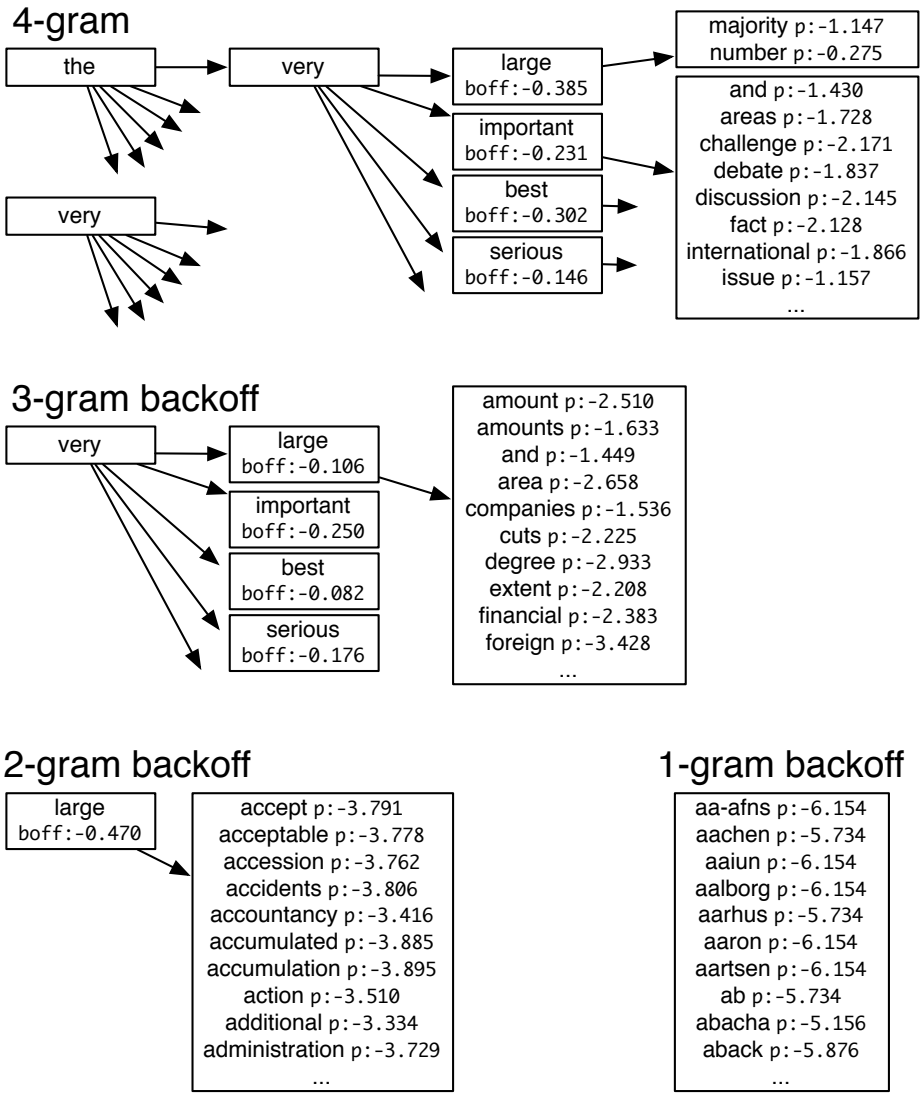
Estimation on Disk

- Language models too large to *build*
- What needs to be stored in RAM?
  - maximum likelihood estimation

$$p(w_n|w_1,...,w_{n-1}) = \frac{\text{count}(w_1,...,w_n)}{\text{count}(w_1,...,w_{n-1})}$$

  - can be done separately for each history $w_1,...,w_{n-1}$
- Keep data on disk
  - extract all n-grams into files on-disk
  - sort by history on disk
  - only keep n-grams with shared history in RAM
- Smoothing techniques may require additional statistics

Efficient Data Structures



- Need to store probabilities for
  - the very large majority
  - the very language number
- Both share history the very large

$\rightarrow$ no need to store history twice

$\rightarrow$ Trie

Fewer Bits to Store Probabilities

- Index for words
  - two bytes allow a vocabulary of $2^{16} = 65,536$ words, typically more needed
  - Huffman coding to use fewer bits for frequent words.
- Probabilities
  - typically stored in log format as floats (4 or 8 bytes)
  - quantization of probabilities to use even less memory, maybe just 4-8 bits

Reducing Vocabulary Size

- For instance: each number is treated as a separate token
- Replace them with a number token NUM
  - but: we want our language model to prefer

$$p_{\text{LM}}(\text{I pay 950.00 in May 2007}) > p_{\text{LM}}(\text{I pay 2007 in May 950.00})$$

  - not possible with number token

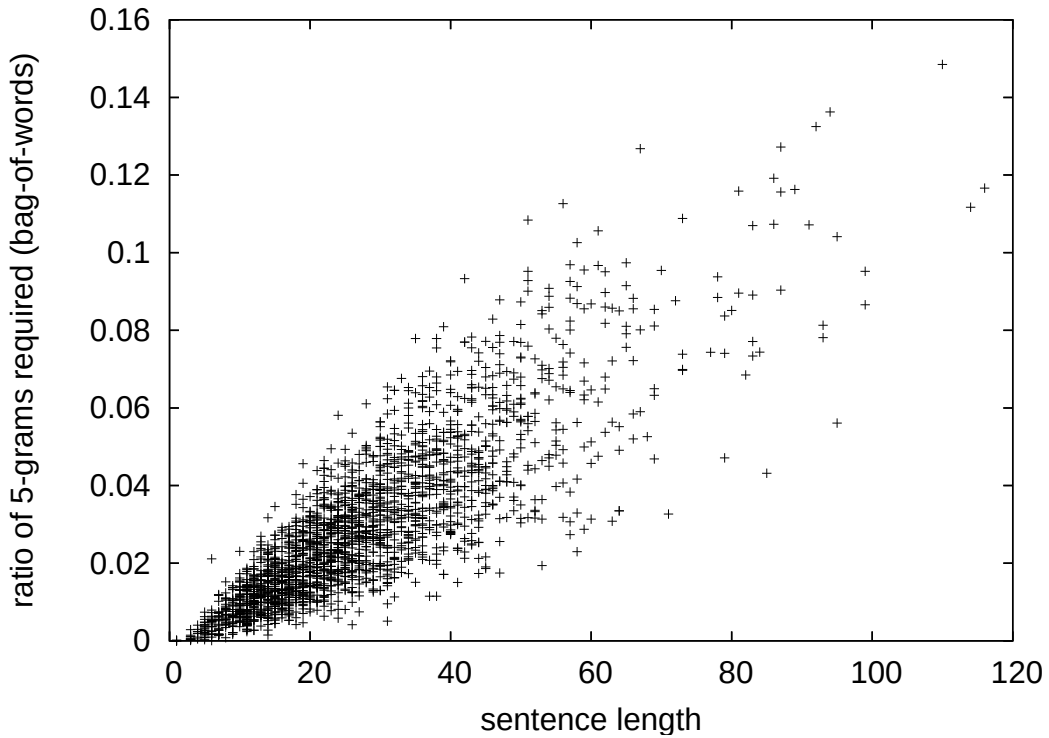
$$p_{\text{LM}}(\text{I pay NUM in May NUM}) = p_{\text{LM}}(\text{I pay NUM in May NUM})$$

- Replace each digit (with unique symbol, e.g., @ or 5), retain some distinctions

$$p_{\text{LM}}(\text{I pay 555.55 in May 5555}) > p_{\text{LM}}(\text{I pay 5555 in May 555.55})$$

Filtering Irrelevant N-Grams

- We use language model in decoding
  - we only produce English words in translation options
  - filter language model down to n-grams containing only those words
- Ratio of 5-grams needed to all 5-grams (by sentence length):



Statistical Language Models Based on Neural Networks

By Tomas Mikolov, Google (2012)

Motivation

A good model of language, meaningful sentences should be more likely than the ambiguous ones.

Limitations of N-gram Models

- Many histories h are similar, but n-grams assume exact match of h.
- Practically, n-grams have problems with representing patterns over more than a few words
- With increasing order of the n-gram model, the number of possible parameters increases exponentially
- There will be never enough of the training data to estimate parameters of high-order N-gram models

Architecture

NN Based LM

- The sparse history h is projected into some continuous low-dimensional space, where similar histories get clustered
- Thanks to parameter sharing among similar histories, the model is more robust: less parameters have to be estimated from the training data

Model Description - Feedforward NNLM

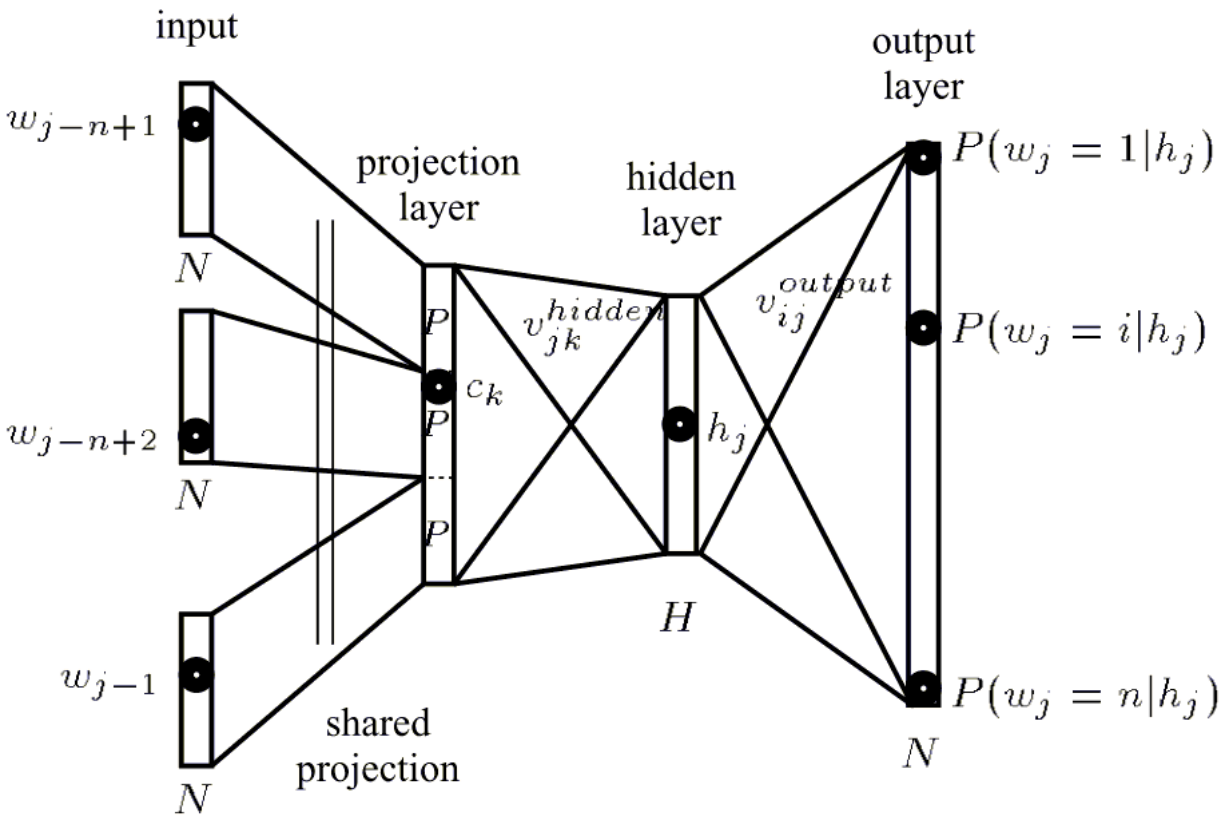
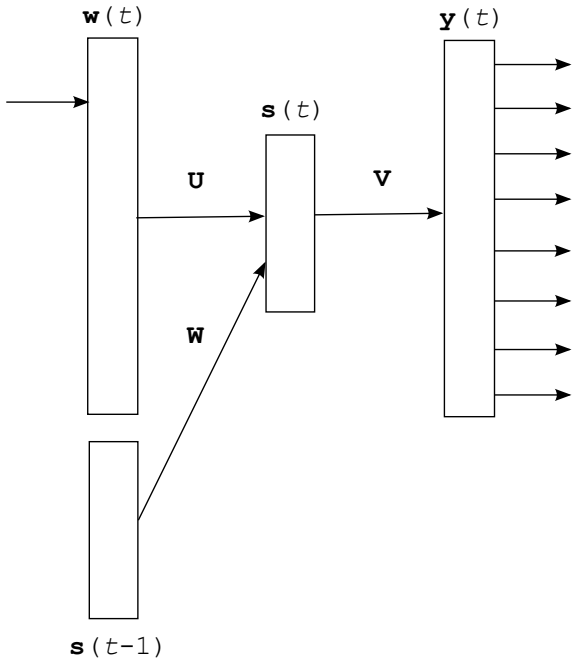


Figure: *Feedforward neural network based LM used by Y. Bengio and H. Schwenk*

Model description - recurrent NNLM



Input layer $\mathbf{w}$ and output layer $\mathbf{y}$ have the same dimensionality as the vocabulary (10K - 200K)
Hidden layer $\mathbf{s}$ is orders of magnitude smaller (50 - 1000 neurons)
$\mathbf{U}$ is the matrix of weights between input and hidden layer, $\mathbf{V}$ is the matrix of weights between hidden and output layer
Without the recurrent weights $\mathbf{W}$, this model would be a bigram neural network language model

Model Description - Recurrent NNLM

The output values from neurons in the hidden and output layers are computed as follows:

$$\mathbf{s}(t) = f\left(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1)\right) \tag{1}$$

$$\mathbf{y}(t) = g\left(\mathbf{V}\mathbf{s}(t)\right), \tag{2}$$

where $f(z)$ and $g(z)$ are sigmoid and softmax activation functions (the softmax function in the output layer is used to ensure that the outputs form a valid probability distribution, i.e. all outputs are greater than 0 and their sum is 1):

$$f(z) = \frac{1}{1+e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{3}$$

Part of Results

Table: *BLEU on IWSLT 2005 Machine Translation task, Chinese to English.*

Model	BLEU
baseline (n-gram)	48.7
300-best rescoring with RNNs	51.2

- About 400K training tokens, small task

Table: *BLEU and NIST score on NIST MT 05 Machine Translation task, Chinese to English.*

Model	BLEU	NIST
baseline (n-gram)	33.0	9.03
1000-best rescoring with RNNs	34.7	9.19

- RNNs were trained on subset of the training data (about 17.5M training tokens), with limited vocabulary

RNNLM Conclusion

Beyond N-grams?

- RNN LMs can generate much more meaningful text than n-gram models trained on the same data
- Many novel but meaningful sequences of words were generated
- RNN LMs are clearly better at modeling the language than n-grams
- However, many simple patterns in the language cannot be efficiently described even by RNNs...

Summary

- Language models: How likely is a string of English words good English?
- N-gram models: Markov assumption (useful but not sufficient)
- Perplexity (intrinsic evaluation)
- Count smoothing (sparsity & large space)
  1. Lapalace (add-one, add-α) [unseen words same probability]
  2. Deleted (held-out) estimation [things-we-saw-once]
  3. Good Turing [things new, assuming leave-one-out]
  4. Back-off (Katz) smoothing [non-linear zero-count]
  5. Interpolation (Jelinek-Mercer) smoothing [linear zero-count]
  6. Witten-Bell [interpolation, diversity of predicted word]
  7. Absolute discounting [approximating Good-Turing]
  8. Kneser-Ney [back-off, diversity of history]
  9. Modified Kneser-Ney [interpolated back-off]
- Managing the size of the model

References

Many slides are from:

- The StatML book's Web site &
- Dan Jurafsky's "Language Modeling: Introduction to N-grams". &
- Bill MacCartney's "NLP Lunch Tutorial: Smoothing"

Thank You!

The End