Language Models

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Dr. Mariana Neves
(adapted from the original slides
of Prof. Philipp Koehn)

January 4th, 2016

Language models

- **Language models** answer the question:

How likely is a string of English words good English?

Language models

- Help with reordering

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

- Help with word choice

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

N-Gram Language Models

- Given: a string of English words $W = w_1, w_2, w_3, ..., w_n$
- Question: what is $p(W)$?

- We collect large amount of text and count how often W occurs to estimate $p(W)$

Sparse data

- Sparse data: Many good English sentences will not have been seen before
- Decomposing $p(W)$ using the chain rule:

  $p(w_1, w_2, w_3, ..., w_n) = p(w_1)\ p(w_2|w_1)\ p(w_3|w_1, w_2)...p(w_n|w_1, ...w_{n-1})$

  (not much gained yet, $p(w_n|w_1, w_2, ...w_{n-1})$ is equally sparse)

Markov Chain

- **Markov assumption**:
  - only previous history matters
  - limited memory: only last k words are included in history (older words less relevant)

  $\rightarrow$ ***k*th order Markov model**

Markov Chain

- For instance 2-gram language model:

$$p(w_1, w_2, w_3, ..., w_n) \simeq p(w_1)\ p(w_2|w_1)\ p(w_3|w_2)...p(w_n|w_{n-1})$$

- What is conditioned on, here w_{i-1} is called the **history**

Model order

- More training data allows for longer histories (higher **kth**).

- Most commonly, trigram (3-grams) models are used.
- But bigrams (2-grams), unigrams (single words) or any other order of n-grams is possible.

Estimating N-Gram Probabilities

- Maximum likelihood estimation

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

- Collect counts over a large text corpus
- Millions to billions of words are easy to get
(trillions of English words available on the web)

Example: 3-Gram

- 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 $\frac{123}{225} = 0.547$.

How good is the LM?

- A good model assigns a text of real English W a high probability
- This can be also measured with cross entropy:

$$H(W) = -\frac{1}{n}\log p(W_1^n)$$
$$-\frac{1}{n}\sum_{i=1}^{n}\log p(w_i|w_1, w_2, \ldots w_{i-1})$$

- Or, **perplexity**

$$\text{perplexity}(W) = 2^{H(W)}$$

Example: trigrams

I would like to commend the rapporteur on his work.

prediction	p_{LM}	$-\log_2 p_{\mathrm{LM}}$
$p_{\mathrm{LM}}(\text{i}\|\text{</s><s>})$	0.109	3.197
$p_{\mathrm{LM}}(\text{would}\|\text{<s>i})$	0.144	2.791
$p_{\mathrm{LM}}(\text{like}\|\text{i would})$	0.489	1.031
$p_{\mathrm{LM}}(\text{to}\|\text{would like})$	0.905	0.144
$p_{\mathrm{LM}}(\text{commend}\|\text{like to})$	0.002	8.794
$p_{\mathrm{LM}}(\text{the}\|\text{to commend})$	0.472	1.084
$p_{\mathrm{LM}}(\text{rapporteur}\|\text{commend the})$	0.147	2.763
$p_{\mathrm{LM}}(\text{on}\|\text{the rapporteur})$	0.056	4.150
$p_{\mathrm{LM}}(\text{his}\|\text{rapporteur on})$	0.194	2.367
$p_{\mathrm{LM}}(\text{work}\|\text{on his})$	0.089	3.498
$p_{\mathrm{LM}}(.\|\text{his work})$	0.290	1.785
$p_{\mathrm{LM}}(\text{</s>}\|\text{work .})$	0.99999	0.000014
	average	2.634

Comparison 1–4-Gram

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

Unseen N-Grams

- We have seen i like to in our corpus
- We have never seen i like to smooth in our corpus

$\rightarrow$ $p(\text{smooth}|\text{i like to}) = 0$

- Any sentence that includes i like to smooth will be assigned probability 0

Add-One Smoothing

- 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 (total number of possible n-grams)

Add-One Smoothing

- 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

Add-α Smoothing

- 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

Example: 2-Grams in Europarl

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

Deleted Estimation

- Estimate true counts in held-out data
  - split corpus in two halves: training and held-out
  - counts in training $C_t(w_1, ..., w_n)$
  - number of n-grams with training count r: N_r
  - total times n-grams of training count r seen in held-out data: T_r

Example: Deleted estimation (bigrams)

Count	Count of counts	Counts in held-out	Exp. count
r	N_r	T_r	$E(r) = T_r/N_r$
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

Deleted Estimation

- We can adjust the real counts to these expected counts
  - better estimates for both seen and unseen events

- Both halves can be switched and results combined

$$r_{del} = \frac{T_r^1 + T_r^2}{N_r^1 + N_r^2} \text{ where } r = count(w_1, ..., w_n)$$

Good-Turing Smoothing

- Adjust actual counts r to expected counts r^* with formula

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

  - N_r number of n-grams that occur exactly r times in corpus

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

Back-Off

- In given corpus, we may never observe
  - Scottish beer drinkers
  - Scottish beer eaters
- Both have count 0

  → our smoothing methods will assign them the same probability

Back-Off

- Better: backoff to bigrams:
  - beer drinkers
  - beer eaters

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

Interpolation

- Combine them

$$p_I(w_3|w_1,w_2) = \lambda_1\ p_1(w_3) + \lambda_2\ p_2(w_3|w_2) + \lambda_3\ p_3(w_3|w_1,w_2)$$

$\forall\lambda_n : 0 \leq \lambda_n \leq 1$
$\sum_n \lambda_n = 1$

Recursive Interpolation

- We can trust some histories $w_{i-n+1}, \ldots, w_{i-1}$ more than others
- Condition interpolation weights on history: $\lambda_{w_{i-n+1},\ldots,w_{i-1}}$
- Recursive definition of interpolation

$$p_n^I(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \lambda_{w_{i-n+1},\ldots,w_{i-1}} \; p_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) + \\ + \; (1 - \lambda_{w_{i-n+1},\ldots,w_{i-1}}) \; p_{n-1}^I(w_i|w_{i-n+2}, \ldots, w_{i-1})$$

Back-Off

- Trust the highest order language model that contains n-gram

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

- Requires
  - adjusted prediction model $\alpha_n(w_i|w_{i-n+1},...,w_{i-1})$
  - discounting function $d_n(w_1,...,w_{n-1})$

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 smoothing considers diversity of predicted words

Witten-Bell Smoothing

- Recursive interpolation method
- 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: Examples

Let us apply this to our two examples:

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

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

Diversity of Histories

- 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

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

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

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 $w_1, ..., w_{n-1}$

Estimation on Disk

- 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

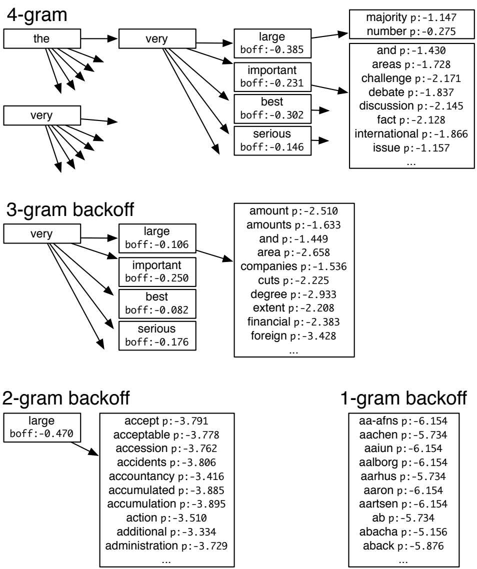
Efficient Data Structures

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

→ no need to store history twice

→ Trie

Efficient Data Structures



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

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

  - not possible with number token

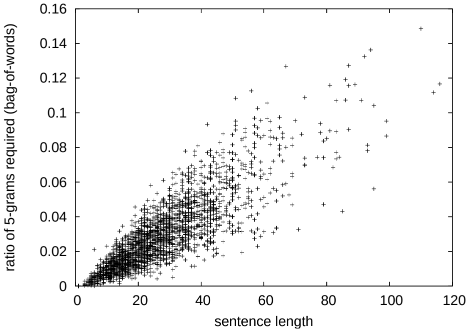
$$p_{\mathrm{LM}}(\text{I pay NUM in May NUM}) = p_{\mathrm{LM}}(\text{I pay NUM in May NUM})$$

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

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

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):



Summary

- Language models: *How likely is a string of English words good English?*
- N-gram models (Markov assumption)
- Perplexity
- Count smoothing
  - add-one, add-α
  - deleted estimation
  - Good Turing
- Interpolation and backoff
  - Good Turing
  - Witten-Bell
  - Kneser-Ney
- Managing the size of the model

Suggested reading

- Statistical Machine Translation, Philipp Koehn (chapter 7).