N-gram Language Modeling Tutorial

Dustin Hillard and Sarah Petersen

Lecture notes courtesy of Prof. Mari Ostendorf

Outline:

- Statistical Language Model (LM) Basics
- n-gram models
- Class LMs
- Cache LMs
- Mixtures
- Empirical observations (Goodman CSL 2001)
- Factored LMs

Part I: Statistical Language Model (LM) Basics

- What is a statistical LM and why are they interesting?
- Evaluating LMs
- History equivalence classes

What is a statistical language model?

A stochastic process model for word sequences. A mechanism for computing the probability of:

$$p(w_1, \ldots, w_T)$$

Why are LMs interesting?

- Important component of a speech recognition system
  - Helps discriminate between similar sounding words
  - Helps reduce search costs
- In statistical machine translation, a language model characterizes the target language, captures fluency
- For selecting alternatives in summarization, generation
- Text classification (style, reading level, language, topic, . . .)
- Language models can be used for more than just words
  - letter sequences (language identification)
  - speech act sequence modeling
  - case and punctuation restoration

Evaluating LMs

Evaluating LMs in the context of an application can be expensive, so LMs are usually evaluated on the own in terms of perplexity::

$$PP = 2^{\tilde{H}_r} \quad \text{where} \quad \tilde{H}_r = -\frac{1}{T}\log_2 p(w_1, \ldots, w_T)$$

where $\{w_1, \ldots, w_T\}$ is held out test data that provides the empirical distribution $q(\cdot)$ in the cross-entropy formula

$$\tilde{H} = -\sum_x q(x) \log p(x)$$

and $p(\cdot)$ is the LM estimated on a training set.

Interpretations:

- Entropy rate: lower entropy means that it is easier to predict the next symbol and hence easier to rule out alternatives when combined with other models

  $$\text{small } \tilde{H}_r \rightarrow \text{ small } PP$$

- Average branching factor: When a distribution is uniform for a vocabulary of size V, then entropy is $\log_2 V$, and perplexity is V. So perplexity indicates an effective next-word vocabulary size, or branching factor.
- Maximum likelihood criterion: minimizing $\tilde{H}_r$ is equivalent to maximizing log likelihood, and one commonly used model selection criterion (in general, not just for LMs) is maximum likelihood on held out data.
- Min K-L distance to the empirical distribution

Caution: Perplexity is an average per symbol (i.e. per word) figure. Not all contexts have the same branching factor. There are some contexts where it is easier to predict the next word than in others.

History Equivalence Classes

The probability of a sequence of variables can always (without assumptions) be decomposed using the chain rule:

$$p(w_1, \ldots, w_T) = p(w_1) \prod_{i=2}^{T} p(w_i | w_1, \ldots, w_{i-1}) = p(w_1) \prod_{i=2}^{T} p(w_i | h_i)$$

For brevity elsewhere, we will use $h_i = \{w_1, \ldots, w_{i-1}\}$ to denote the history of the i-th word w_i.

It is not practical to use the whole history. Instead, we use equivalence classes,

$$p(w_1, \ldots, w_T) = p(w_1) \prod_{i=2}^{T} p(w_i | \Phi(w_1, \ldots, w_{i-1}))$$

assuming that $p(w|\Phi(h_i))$ is approximately equal for all h_i such that $\Phi(h_i) = j$.

Examples of equivalence classes and resulting number of distributions:

- $\Phi(w_1, \ldots, w_{i-1}) = \{w_{i-2}, w_{i-1}\}$ truncation (V^3)
- $\Phi(w_1, \ldots, w_{i-1}) = \{w_{i-2}, w_{i-1}, t_i\}$ truncation + topic ($V^3 T$)
- $\Phi(w_1, \ldots, w_{i-1}) = \{c_{i-3}, c_{i-2}, c_{i-1}\}$ truncation + word classes (VC^3)

One could use individual word classes (semantic or part-of-speech) or phrase-sized classes.

Part II: The n-gram language model

Key issues:

- Tree representation
- Estimating n-grams with MLE
- Smoothing and back-off
- Pruning and variable n-grams

In an n-gram, we truncate the history to length $n-1$

$$p(w_i|w_1,\ldots,w_{i-1}) = p(w_i|w_{i-n+1},\ldots,w_{i-1})$$

and it is so-named because it characterizes a sequence of n variables.

- unigram: $p(w_i)$ (i.i.d. process)
- bigram: $p(w_i|w_{i-1})$ (Markov process)
- trigram: $p(w_i|w_{i-2},w_{i-1})$

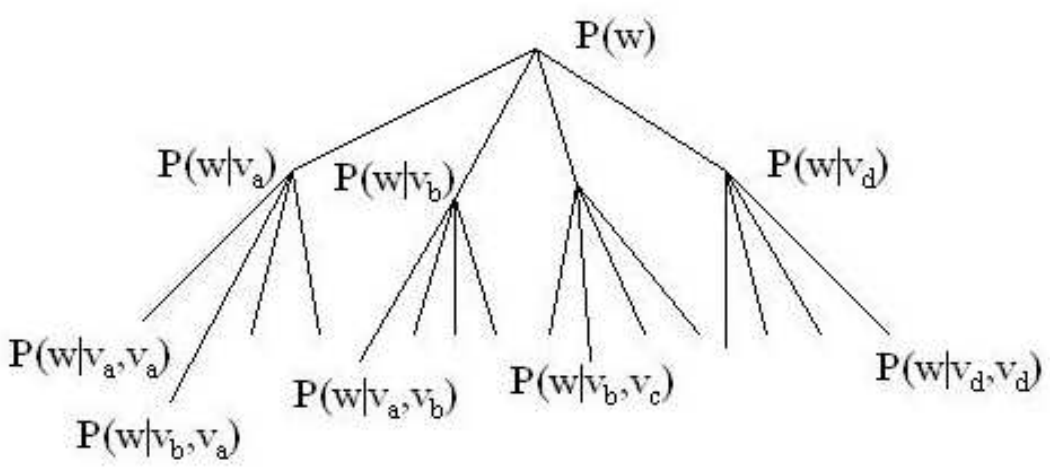
There are many anecdotal examples to show why n-grams are poor models of language. From Manning and Schuetze (p. 193):

> Sue swallowed the large green ....

However, n-grams are very powerful models and difficult to beat (at least for English), since frequently the short-distance context is most important.

Shannon's and Brown et al.'s estimates of the entropy of English demonstrate that n-grams work quite well.

You can represent an n-gram using a V-ary branching tree structure for vocabulary size V, as in the tree below for a 4-word vocabulary.



Each node in the tree is associated with a probability distribution for the V words in the vocabulary. The unigram is at the root node; the V different bigrams are at the next level; and the trigrams are at the next. The tree could be extended further for higher order n-grams. The nodes further down the tree represent longer-distance histories.

This picture should make it clear that there are potentially V^n parameters in an n-gram for vocabulary size V. For moderate n-grams (2-4) and interesting vocabulary sizes (20k-60k), this can get very large. For example, a 20k-word vocabulary would require 8 trillion parameters to fully represent a trigram. This poses storage, memory, and estimation problems.

Estimating n-gram probabilities

We can estimate n-gram probabilities by counting relative frequency on a training corpus.
(This is maximum likelihood estimation.)

$$\hat{p}(w_a) = \frac{c(w_a)}{N}$$
$$\hat{p}(w_b|w_a) = \frac{c(w_a, w_b)}{\Sigma_{w_b} c(w_a, w_b)} \approx \frac{c(w_a, w_b)}{c(w_a)}$$

where N is the total number of words in the training set and $c(\cdot)$ denotes count of the word or word sequence in the training data.

> *(Aside: the approximation is because of edge effects. We will generally not worry about this in writing the equations in the future.)*

PROBLEM: what if $c(w_a) = 0$?

- For unigram case, we get a zero probability, which is generally a bad idea, since you often don't want to rule things out entirely (when you have finite training data).
- For the bigram case, we get an undefined probability, which is even more problematic.

This really happens! Human language has "lopsided sparsity" – there's a fairly high probability of seeing an event that was not seen in the training corpus, even for large corpora.

Example from Manning and Schuetze p.200 (Table 6.3)

"Probabilities of each successive word for a clause from *Persuasion*. The probability distribution for the following word is calculated by Maximum Likelihood Estimate n-gram models for various values of n. The predicted likelihood rank of different words is shown in the first column. The actual next word is shown at the top of the table in italics, and in the table in bold." (Table is split to fit on two pages.)

In person	*she*		*was*		*inferior*		*to*		*both*		*sisters*	
1-gram	$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$	
1	the	0.034	the	0.034	the	0.034	the	0.034	the	0.034	the	0.034
2	to	0.32	to	0.32	to	0.32	**to**	**0.32**	to	0.32	to	0.32
3	and	0.030	and	0.030	and	0.030			and	0.030	and	0.030
4	of	0.029	of	0.029	of	0.029			of	0.029	of	0.029
...												
8	was	0.015	**was**	**0.015**	was	0.015			was	0.015	was	0.015
...												
13	**she**	**0.011**			she	0.011			she	0.011	she	0.011
...												
254					both	0.0005			**both**	**0.0005**	both	0.0005
...												
435					sisters	0.0003					**sisters**	**0.0003**
...												
1701					**inferior**	**0.00005**						
2-gram	$P(\cdot\|person)$		$P(\cdot\|she)$		$P(\cdot\|was)$		$P(\cdot\|inf.)$		$P(\cdot\|to)$		$P(\cdot\|both)$	
1	and	0.099	had	0.141	not	0.065	**to**	**0.212**	be	0.111	of	0.066
2	who	0.099	**was**	**0.122**	a	0.0522			the	0.057	to	0.041
3	to	0.076			the	0.033			her	0.048	in	0.038
4	in	0.045			to	0.0311			have	0.027	and	0.025
...												
23	**she**	**0.009**							Mrs	0.006	she	0.009
...												
41									what	0.004	**sisters**	**0.006**
...												
293									**both**	**0.004**		
...												
∞					**inferior**	**0**						

In person	*she*	*was*		*inferior*		*to*	*both*		*sisters*	
3-gram	$P(\cdot\|In, p)$	$P(\cdot\|p, she)$		$P(\cdot\|she, was)$		$P(\cdot\|was, i)$	$P(\cdot\|i, to)$		$P(\cdot\|to, both)$	
1	UNSEEN	did	0.5	not	0.057	UNSEEN	the	0.286	to	0.222
2		**was**	**0.5**	very	0.038		Maria	0.143	Chapter	0.111
3				in	0.030		cherries	0.143	Hour	0.111
4				to	0.026		her	0.143	Twice	0.111
...										
∞				**inferior**	**0**		**both**	**0**	**sisters**	**0**
4-gram	$P(\cdot\|u, I, p)$	$P(\cdot\|I, p, s)$		$P(\cdot\|p, s, w)$		$P(\cdot\|s, w, i)$	$P(\cdot\|w, i, t)$		$P(\cdot\|i, t, b)$	
1	UNSEEN	UNSEEN		in	1.0	UNSEEN	UNSEEN		UNSEEN	
...										
∞				**inferior**	**0**					

SOLUTION: smoothing

There are many different alternatives to smoothing, which roughly fall into the following categories:

- Add counts to account for unseen events that may occur in the future. Important historical example: Laplace estimate

$$p(w_a) = \frac{c(w_a) + 1}{N + V}$$

- Interpolation: weighted combination of target and lower-order distributions

$$p(w_i|w_{i-2}, w_{i-1}) = \lambda_3 f(w_i|w_{i-2}, w_{i-1}) + \lambda_2 f(w_i|w_{i-1}) + \lambda_1 f(w_i) + \lambda_0 \frac{1}{V}$$

  where $f(w|\cdot)$ is a relative frequency (ML) estimate and $\Sigma_i \lambda_i = 1$. The weights are typically estimated on a held-out data set.

- Backoff: steal from the rich (seen events) and give to the poor (unseen)

$$p(w_i|w_{i-2}, w_{i-1}) = \begin{cases} f(w_3|w_1, w_2) & \text{if } c(w_1, w_2, w_3) \geq K_2 \\ discount(f(w_3|w_1, w_2)) & \text{if } K_1 \geq c(w_1, w_2, w_3) < K_2 \\ distribute(f(w_3|w_2)) & \text{if } c(w_1, w_2, w_3) < K_1 \end{cases}$$

  (The no-discount case is not always used.) Discounting can take different forms:

  – absolute: subtract counts $(r - \delta)/N$

  – linear: subtract a percentage $(1 - \alpha)r/N$

  where r/N is the relative frequency estimate. Distributing spreads the stolen mass according to lower order distributions.

Language modeling toolkits usually implement several interpolation and backoff options, so you can experiment for yourself.

A paper by Chen and Goodman (CSL, 1999) looks extensively at different alternatives, testing with different amounts of training data and different corpora. The best results under a broad range of conditions are obtained using modified Kneser-Ney smoothing, e.g. for trigrams:

$$p_{KN}(w_i|w_{i-2},w_{i-1}) = \frac{c(w_{i-2},w_{i-1},w_i) - D(c(w_{i-2},w_{i-1},w_i))}{c(w_{i-2},w_{i-1})} + \gamma(w_{i-2},w_{i-1})p_{KN}(w_i|w_{i-1})$$

where $\gamma(w_{i-2},w_{i-1})$ is chosen such that the distributions sum to 1 and where $D(c(\cdot))$ allows you to have smaller discounts for smaller counts.

Practical considerations

Both backoff and interpolation use all orders. You don't want to store all $V^n + V^{n-1} + \cdots + V + 1$ distributions.

Typically, people do not store probabilities unless the count of the n-gram is greater than some threshold $c(\cdot) < K$. There can be different thresholds for different order n-grams and these are often referred to as "cut-offs."

Not storing probabilities is sometimes referred to as pruning, using the tree analogy. One can prune individual probabilities using a count threshold, as described above, or also using an information-theoretic criterion for determining if the higher-order probability is similar to the lower-order pruning. This is often referred to as "Stolcke pruning" and is the entropy pruning option in the SRI LM toolkit.

It is important to understand that pruning does not mean that these n-grams have zero probability, not that you back-off to the lower-order n-gram. Rather, multiple pruned n-grams are all represented with a single parameter depending on the lower-order n-gram ($\gamma(w_{i-2},w_{i-1})$).

Variable N-grams – Distribution Pruning

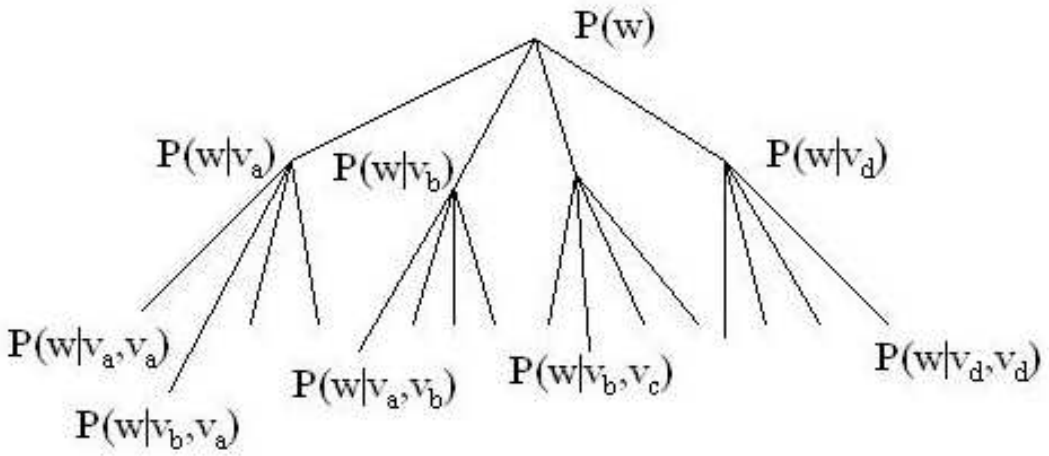
One can also think of pruning whole branches or subtrees out of the tree, i.e. pruning at the distribution level (vs. not storing probabilities for specific words in specific contexts).

For example, if $p_b = p(w|v_b)$ and $p_{ab} = p(w|v_a, v_b)$, then prune the p_{ab} branch if $D(p_{ab}||p_b) < \epsilon$.

This results in a *variable n-gram*, or model that uses a variable-length history depending on the context, e.g.

$$\begin{aligned} p(w_1, \ldots, w_T) &= p(w_1)p(w_2|w_1)p(w_3)p(w_4|w_2, w_3)p(w_5|w_2, w_3, w_4) \\ &\quad \cdots p(w_T|w_{T-1}) \end{aligned}$$

Note: as in any tree growing problem, it is better to grow big and prune back, rather than prune as you go.



Part III: Class LMs

Grouping words in classes can help improve probability estimates for less frequent words.

Generally, people assume that words are deterministically associated with classes (for simplicity). Consider the trigram case, for example:

$$\begin{aligned} p(w_1, w_2, \ldots, w_T) &= p(w_1, c_1, w_2, c_2, \ldots, w_T, c_T) \\ &= \prod_{i=1}^{T} p(w_i, c_i | w_{i-2}, c_{i-2}, w_{i-1}, c_{i-1}) \\ &= \prod_{i=1}^{T} p(w_i | w_{i-2}, c_{i-2}, w_{i-1}, c_{i-1}, c_i) p(c_i | w_{i-2}, c_{i-2}, w_{i-1}, c_{i-1}) \\ &\approx \prod_{i=1}^{T} p_t(w_i | c_i) p_s(c_i | c_{i-2}, c_{i-1}) \end{aligned}$$

The last line corresponds to the simplest version, which represents sequence behavior ($p_s(\cdot)$, C^3 parameters) in terms of classes only and content/topic behavior ($p_t(\cdot)$, VC parameters) depending on only the current class. Compare $C^3 + VC$ to V^3 parameters for the trigram.

There are other options. Using the Goodman notation of lower case for words and upper case for classes, some options are:

$p(z\|Z)p(Z\|X,Y)$	above model, simplest case
$p(z\|y,Z)p(Z\|X,y)$	intermediate model, words only at bigram level
$p(z\|x,y,Z)p(Z\|x,y)$	no simplification, impacts back-off only

Note: the word prediction class (Z) and the sequence prediction classes (X and Y, changed to $\mathcal{X}$ and $\mathcal{Y}$ for clarity) need not be the same partition of the word space, e.g.

$$p(z|x,y) \approx p(z|Z)p(Z|\mathcal{X},\mathcal{Y})$$

Example: “a” and “an” can follow the same words, but very few words can follow both “a” and “an”. Following Goodman ’01, we’ll call Z a predictive class, and $\mathcal{X}$ a conditional class.

Why use classes?

- They provide the opportunity for another dimension of smoothing: rather than dropping x from the history because it was not observed one can use the word class which may have been observed.
- They require fewer parameters; smaller LM takes up less memory, i.e. $C < V$ generally implies $C^3 << V^3$.

How do you get the classes?

- By knowledge (part-of-speech or hand-specified semantic classes)
- Automatic clustering to maximize likelihood (min perplexity)
  - hierarchical (e.g. agglomerative: iterative merging of words and clusters, Brown et al. '92, in standard toolkits)
  - partitioning (iterative reassignment)

(Aside: hierarchical clustering gives a representation that can be interpreted as a bit encoding of words, which can be used as a numeric or feature representation of words (Jelinek 1997).)

To do clustering, you need an objective: min perplexity $\Longrightarrow$ max likelihood of:

$$\prod_i P(w_i|W_{i-1}) \quad \text{and} \quad \prod_i P(w_{i-1}|\mathcal{W}_i)$$

Empirical observations:

- With sufficient data, automatic clusters work better than POS (since you can easily increase the number), but smoothing is needed.
- It is often useful to put the most frequent words in their own class, which can reduce the difference between automatic and POS classes.
- In constrained domains, hand-defined semantic classes are useful.

The disadvantage of class LMs is that they are less powerful, so class LMs almost always give higher perplexity than word LMs, except in cases where training resources are small. However, even if the class LM is less powerful, it can be useful in combination with the trigram:

$$\hat{p}(w_i|w_{i-2}, w_{i-1}) = (1 - \lambda)p_{tri}(w_i|w_{i-2}, w_{i-1}) + \lambda[p(w_i|c_i)p(c_i|c_{i-2}, c_{i-1})]$$

Part IV: Cache LMs

The basic idea of a cache LM is to interpolate a static LM estimated from a large amount of data with a dynamic LM estimated from recently observed words in the document/speech being processed:

$$p(w_i|h_i) = \lambda p_{static}(w_i|w_{i-2}, w_{i-1}) + (1-\lambda)p_{cache}(w_i|w_{i-1})$$

Usually the cache does not contain much data, so it does not make sense to use a high-order n-gram, but bigrams are often useful.

Practical issues:

- There are various options for restarting or windowing text for building the cache, and for temporally weighting data (higher weights for recent words)
- λ is usually tuned by heuristically adjusting it to minimize error rate.
- Sometimes function words are not updated, but this leads to trickier normalization.

Early work (in speech recognition) was done by Kupiec, Kuhn & DeMori and then popularized by IBM. Later work on trigger language modeling, using information-theoretic measures to find "trigger" words that indicate need for increasing probabilities of topically related words, showed that the most important triggers are "self-triggers", i.e. an instance of the word itself. This finding supports the use of cache LMs.

Part V: Mixture LMs

Mixtures are used for interpolating:

- different types of LMs (e.g. word LM and class LM),
- cache LMs,
- topic LMs,
- LMs trained on different sources,

and more. The type of dependence of the mixture weights on the history depends on the application.

n-gram mixtures

$$p(w_i|h_i) = \sum_{j=1}^{m} \lambda_j(h_i) p_j(w_i|h_i)$$

$$p(w_1, \ldots, w_T) = \prod_{i=1}^{T} p(w_i|h_i) = \prod_{i=1}^{T} \left(\sum_j \lambda_j(h_i) p_j(w_i|h_i) \right)$$

sentence-level mixtures

$$p(w_1, \ldots, w_T) = \sum_{j=1}^{m} p_j(w_1, \ldots, w_T) = \sum_j \left[\prod_{i=1}^{T} \lambda_j(h_i) p_j(w_i|h_i) \right]$$

Interpretation:
Let a mixture component p_j correspond to a topic. In an n-gram-level mixture, the topic can change within a sentence, while it cannot in a sentence-level mixture. For some phenomena, one approach may be more realistic than the other, but the sentence-level mixture has a higher computational cost because the Markov assumption no longer holds.

Mixture LMs are usually designed by estimating:

- the component language models from different training sets or via unsupervised clustering, and
- the mixture weights to max likelihood of a held-out data set.

Unsupervised (or semi-supervised) clustering via partitioning:

Given an initial representation of each cluster (e.g. n-gram LM)

1. Assign each sentence (or document) to the cluster with the highest probability
2. Re-estimate cluster model

Part VI: Empirical Observations (Goodman, 2001)

Some findings based on perplexity

- Of all the above LM variations, the cache LM is by far the most useful
- Modified Kneser-Ney is always the best back-off alternative (see also Chen and Goodman, 1999)
- Sentence-level mixtures have a lot of potential (depending on amount of training)

(see the paper for more)

Bad News

- In ASR, cache LM often breaks due to word errors
- Specific smoothing method doesn't matter with lots of data and big cut-offs (to reduce size of LM)
- Classes, sentence-level mixtures, etc. complicate applications such as ASR

Good News

- ASR error rates are going down and you can use confidence to improve caching
- Computer memory is increasing so bigger LMs can be used, and data is not always large
- N-best rescoring allows for more complex LMs

Lesson: The application and implementation details matter!