Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

September 25, 2014

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 1: Probability models of Language

The Language Modeling problem

Setup

- Assume a (finite) vocabulary of words:

$$\mathcal{V} = \{killer, crazy, clown\}$$

- Use $\mathcal{V}$ to construct an infinite set of *sentences*

$$\begin{aligned}\mathcal{V}^+ = \; \{ & \\ & \text{clown, killer clown, crazy clown,} \\ & \text{crazy killer clown, killer crazy clown,} \\ & \ldots \\ \} & \end{aligned}$$

- A *sentence* is **defined** as each $s \in \mathcal{V}^+$

The Language Modeling problem

Data

Given a training data set of example sentences $s \in \mathcal{V}^+$

Language Modeling problem

Estimate a probability model:

$$\sum_{s \in \mathcal{V}^+} p(s) = 1.0$$

- p(clown) = 1e-5
- p(killer) = 1e-6
- p(killer clown) = 1e-12
- p(crazy killer clown) = 1e-21
- p(crazy killer clown killer) = 1e-110
- p(crazy clown killer killer) = 1e-127

Why do we want to do this?

Scoring Hypotheses in Speech Recognition

From acoustic signal to candidate transcriptions

Hypothesis	Score
the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

Scoring Hypotheses in Machine Translation

From source language to target language candidates

Hypothesis	Score
we must also discuss a vision .	-29.63
we must also discuss on a vision .	-31.58
it is also discuss a vision .	-31.96
we must discuss on greater vision .	-36.09
⋮	⋮

Scoring Hypotheses in Decryption

Character substitutions on ciphertext to plaintext candidates

Hypothesis	Score
Heopaj, zk ukq swjp pk gjks w oaynap?	-93
Urbcnw, mx hxd fjwc cx twxf j bnlanc?	-92
Wtdepy, oz jzf hlye ez vyzh l dpncpe?	-91
Mjtufo, ep zpv xbou up lopx b tfdsfu?	-89
Nkuvgp, fq aqw ycpv vq mpqy c ugetgv?	-87
Gdnozi, yj tjp rvio oj fijr v nzxmzo?	-86
Czjkve, uf pfl nrek kf befn r jvtivk?	-85
Yvfgra, qb lbh jnag gb xabj n frperg?	-84
Zwghsb, rc mci kobh hc ybck o gsqfsh?	-83
Byijud, te oek mqdj je adem q iushuj?	-77
Jgqrcl, bm wms uylr rm ilmu y qcapcr?	-76
Listen, do you want to know a secret?	-25

Scoring Hypotheses in Spelling Correction

Substitute spelling variants to generate hypotheses

Hypothesis	Score
… stellar and versatile **acress** whose combination of sass and glamour has defined her …	-18920
… stellar and versatile **acres** whose combination of sass and glamour has defined her …	-10209
… stellar and versatile **actress** whose combination of sass and glamour has defined her …	-9801

Probability models of language

Question

- Given a finite vocabulary set $\mathcal{V}$
- We want to build a probability model $P(s)$ for all $s \in \mathcal{V}^+$
- **But** we want to consider sentences s of each length ℓ separately.
- Write down a new model over $\mathcal{V}^+$ such that $P(s \mid \ell)$ is in the model
- **And** the model should be equal to $\sum_{s \in \mathcal{V}^+} P(s)$.
- Write down the model

$$\sum_{s \in \mathcal{V}^+} P(s) = \ldots$$

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 2: *n*-grams for Language Modeling

Language models

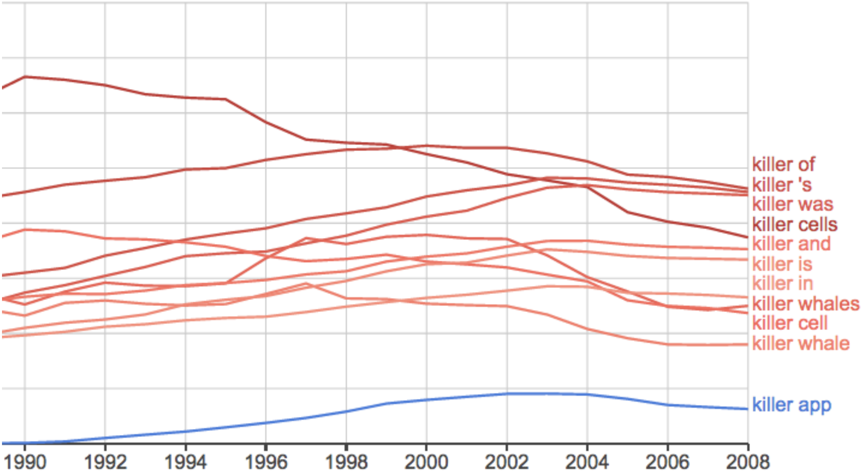
n-grams for Language Modeling

Smoothing *n*-gram Models
- Smoothing Counts
  - Add-one Smoothing
  - Additive Smoothing
  - Good-Turing Smoothing
- Smoothing by Interpolation
  - Interpolation: Jelinek-Mercer Smoothing
- Backoff Smoothing
  - Katz Backoff
  - Backoff Smoothing with Discounting

Evaluating Language Models

n-gram Models

Google n-gram viewer



Learning Language Models

- Directly count using a training data set of sentences: $w_1, \ldots, w_n$:

$$p(w_1, \ldots, w_n) = \frac{n(w_1, \ldots, w_n)}{N}$$

- n is a function that counts how many times each sentence occurs
- N is the sum over all possible $n(\cdot)$ values
- Problem: does not generalize to new sentences unseen in the training data.
- What are the chances you will see a sentence: `crazy killer clown crazy killer`?
- In NLP applications we often need to assign non-zero probability to previously unseen sentences.

Learning Language Models

Apply the Chain Rule: the unigram model

$$
\begin{aligned}
p(w_1, \ldots, w_n) &\approx p(w_1)p(w_2)\ldots p(w_n) \\
&= \prod_i p(w_i)
\end{aligned}
$$

Big problem with a unigram language model

p(the the the the the the the) $>$ p(we must also discuss a vision .)

Learning Language Models

Apply the Chain Rule: the bigram model

$$\begin{aligned} p(w_1, \ldots, w_n) &\approx p(w_1)p(w_2 \mid w_1) \ldots p(w_n \mid w_{n-1}) \\ &= p(w_1) \prod_{i=2}^{n} p(w_i \mid w_{i-1}) \end{aligned}$$

Better than unigram

$p(\text{the the the the the the the}) < p(\text{we must also discuss a vision .})$

Learning Language Models

Apply the Chain Rule: the trigram model

$$
\begin{aligned}
p(w_1, \ldots, w_n) &\approx \\
&p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \ldots p(w_n \mid w_{n-2}, w_{n-1}) \\
&p(w_1)p(w_2 \mid w_1)\prod_{i=3}^{n} p(w_i \mid w_{i-2}, w_{i-1})
\end{aligned}
$$

Better than bigram, but ...

p(we must also discuss a vision .) might be zero because we have not seen p(discuss | must also)

Maximum Likelihood Estimate

Using training data to learn a trigram model

- Let $c(u, v, w)$ be the count of the trigram u, v, w, e.g. $c(crazy, killer, clown)$
- Let $c(u, v)$ be the count of the bigram u, v, e.g. $c(crazy, killer)$
- For any u, v, w we can compute the conditional probability of generating w given u, v:

$$p(w \mid u, v) = \frac{c(u, v, w)}{c(u, v)}$$

- For example:

$$p(clown \mid crazy, killer) = \frac{c(crazy, killer, clown)}{c(crazy, killer)}$$

Number of Parameters

How many probabilities in each *n*-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many unigram probabilities: $P(x)$ for $x \in \mathcal{V}$?

4

Number of Parameters

How many probabilities in each *n*-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many bigram probabilities: $P(y|x)$ for $x, y \in \mathcal{V}$?

$4^2 = 16$

Number of Parameters

How many probabilities in each *n*-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many trigram probabilities: $P(z|x, y)$ for $x, y, z \in \mathcal{V}$?

$4^3 = 64$

Number of Parameters

Question

- Assume $|\mathcal{V}| = 50{,}000$ (a realistic vocabulary size for English)
- What is the minimum size of training data in tokens?
  - If you wanted to observe all unigrams at least once.
  - If you wanted to observe all trigrams at least once.

125,000,000,000,000 (125 Ttokens)

Some trigrams should be zero since they do not occur in the language, $P(\textit{the} \mid \textit{the}, \textit{the})$.
But others are simply unobserved in the training data, $P(\textit{idea} \mid \textit{colourless}, \textit{green})$.

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 3: Smoothing Probability Models

Language models

n-grams for Language Modeling

Smoothing *n*-gram Models

- Smoothing Counts
  - Add-one Smoothing
  - Additive Smoothing
  - Good-Turing Smoothing
- Smoothing by Interpolation
  - Interpolation: Jelinek-Mercer Smoothing
- Backoff Smoothing
  - Katz Backoff
  - Backoff Smoothing with Discounting

Evaluating Language Models

Bigram Models

- In practice:

$$P(\text{Mork read a book}) = \\ P(\text{Mork} \mid <\text{start}>) \times P(\text{read} \mid \text{Mork}) \times \\ P(\text{a} \mid \text{read}) \times P(\text{book} \mid \text{a}) \times \\ P(<\text{stop}> \mid \text{book})$$

- $P(w_i \mid w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$
  On unseen data, $c(w_{i-1}, w_i)$ or worse $c(w_{i-1})$ could be zero

$$\sum_{w_i} \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} = ?$$

Smoothing

- **Smoothing** deals with events that have been observed zero times
- Smoothing algorithms also tend to improve the accuracy of the model

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

- Not just unobserved events: what about events observed once?

Add-one Smoothing

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

- Add-one Smoothing:

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

- Let V be the number of words in our vocabulary
  Assign count of 1 to unseen bigrams

Add-one Smoothing

$$
\begin{aligned}
& P(\text{Mindy read a book}) = \\
& \quad P(\text{Mindy} \mid \ < \text{start} >) \times P(\text{read} \mid \text{Mindy}) \times \\
& \quad P(\text{a} \mid \text{read}) \times P(\text{book} \mid \text{a}) \times \\
& \quad P(< \text{stop} > \ \mid \text{book})
\end{aligned}
$$

- Without smoothing:

$$P(\text{read} \mid \text{Mindy}) = \frac{c(\text{Mindy, read})}{c(\text{Mindy})} = 0$$

- With add-one smoothing (assuming c(Mindy) = 1 but c(Mindy, read) = 0):

$$P(\text{read} \mid \text{Mindy}) = \frac{1}{V+1}$$

Additive Smoothing: (Lidstone 1920, Jeffreys 1948)

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

- Add-one smoothing works horribly in practice. Seems like 1 is too large a count for unobserved events.
- Additive Smoothing:

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

- $0 < \delta \leq 1$
  Still works horribly in practice, but better than add-one smoothing.

Good-Turing Smoothing: (Good, 1953)

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

- Imagine you're sitting at a sushi bar with a conveyor belt.
- You see going past you 10 plates of tuna, 3 plates of unagi, 2 plates of salmon, 1 plate of shrimp, 1 plate of octopus, and 1 plate of yellowtail
- Chance you will observe a new kind of seafood: $\frac{3}{18}$
- How likely are you to see another plate of salmon: should be $< \frac{2}{18}$

Good-Turing Smoothing

- How many types of seafood (words) were seen once? Use this to predict probabilities for unseen events
  Let n_1 be the number of events that occurred once: $p_0 = \frac{n_1}{N}$
- The Good-Turing estimate states that for any n-gram that occurs r times, we should pretend that it occurs r^* times

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

- n_r: number of different objects seen r times

Good-Turing Smoothing

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- How likely is new data? Let n_1 be the number of items occurring once, which is 3 in this case. N is the total, which is 18.

$$p_0 = \frac{n_1}{N} = \frac{3}{18} = 0.166$$

Good-Turing Smoothing

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- How likely is *octopus*? Since $c(octopus) = 1$ The GT estimate is 1^*.

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

$$p_{GT} = \frac{r^*}{N}$$

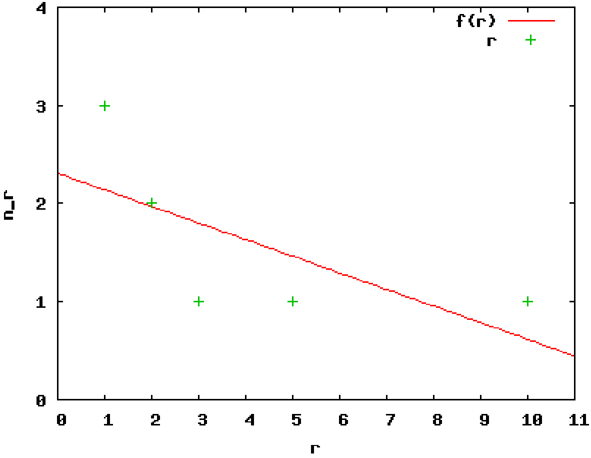
- To compute 1^*, we need $n_1 = 3$ and $n_2 = 1$

$$1^* = 2 \times \frac{1}{3} = \frac{2}{3}$$

$$p_1 = \frac{1^*}{18} = 0.037$$

- What happens when $n_{r+1} = 0$? (smoothing before smoothing)

Simple Good-Turing: linear interpolation for missing n_{r+1}



$$f(r) = a + b * r$$
$$a = 2.3$$
$$b = -0.17$$

r	$n_r = f(r)$
1	2.14
2	1.97
3	1.80
4	1.63
5	1.46
6	1.29
7	1.12
8	0.95
9	0.78
10	0.61
11	0.44

Comparison between Add-one and Good-Turing

freq	num with freq r	NS	Add1	SGT
r	n_r	p_r	p_r	p_r
0	0	0	0.0294	0.12
1	3	0.04	0.0588	0.03079
2	2	0.08	0.0882	0.06719
3	1	0.12	0.1176	0.1045
5	1	0.2	0.1764	0.1797
10	1	0.4	0.3235	0.3691

- $N = (1 * 3) + (2 * 2) + 3 + 5 + 10 = 25$
- $V = 1 + 3 + 2 + 1 + 1 + 1 = 9$
- Important: we added a new word type for unseen words. Let's call it UNK, the unknown word.
- Check that: $1.0 == \sum_r n_r \times p_r$
  $0.12 + (3 * 0.03079) + (2 * 0.06719) + 0.1045 + 0.1797 + 0.3691 = 1.0$

Comparison between Add-one and Good-Turing

freq	num with freq r	NS	Add1	SGT
r	n_r	p_r	p_r	p_r
0	0	0	0.0294	0.12
1	3	0.04	0.0588	0.03079
2	2	0.08	0.0882	0.06719
3	1	0.12	0.1176	0.1045
5	1	0.2	0.1764	0.1797
10	1	0.4	0.3235	0.3691

- NS = No smoothing: $p_r = \frac{r}{N}$
- Add1 = Add-one smoothing: $p_r = \frac{1+r}{V+N}$
- SGT = Simple Good-Turing: $p_0 = \frac{n_1}{N}$, $p_r = \frac{(r+1)\frac{n_{r+1}}{n_r}}{N}$ with linear interpolation for missing values where $n_{r+1} = 0$ (Gale and Sampson, 1995) *http://www.grsampson.net/AGtf1.html*

Using unigrams to smooth bigrams: incorrect version

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

- In add-one or Good-Turing:
  $P(\text{the} \mid \text{string}) = P(\text{Fonz} \mid \text{string})$
- If $c(w_{i-1}, w_i) = 0$, then use $P(w_i)$ (back off)
- Works for trigrams too: back off to bigrams and then unigrams
- Problem: probabilities get mixed up (unseen bigrams, for example will get higher probabilities than seen bigrams)

Interpolation: Jelinek-Mercer Smoothing

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

- $P_{JM}(w_i \mid w_{i-1}) = \lambda P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda) P_{ML}(w_i)$ where, $0 \leq \lambda \leq 1$
- Notice that $P_{JM}(\text{the} \mid \text{string}) > P_{JM}(\text{Fonz} \mid \text{string})$ as we wanted
- Jelinek-Mercer (1980) describe an elegant form of this **interpolation**:

$$P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1 - \lambda) P_{JM}(n - 1\text{gram})$$

- What about $P_{JM}(w_i)$?
  For missing unigrams: $P_{JM}(w_i) = \lambda P_{ML}(w_i) + (1 - \lambda) \frac{\delta}{V}$

Interpolation: Finding λ

$$P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1 - \lambda) P_{JM}(n - 1\text{gram})$$

- Deleted Interpolation (Jelinek, Mercer)
  compute λ values to minimize cross-entropy on **held-out** data which is deleted from the initial set of training data
- Improved JM smoothing, a separate λ for each w_{i-1}:

$$P_{JM}(w_i \mid w_{i-1}) = \lambda(w_{i-1}) P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda(w_{i-1})) P_{ML}(w_i)$$

Backoff Smoothing: Katz Backoff

- Use smoothing over counts for backoff smoothing.
- Also called discounting since we remove some probability from observed events.
- Katz Backoff (include Good-Turing with Backoff Smoothing)

$$P_{katz}(y \mid x) = \begin{cases} \frac{c^*(xy)}{c(x)} & \text{if } c(xy) > 0 \\ \alpha(x) P_{katz}(y) & \text{otherwise} \end{cases}$$

- where $\alpha(x)$ is chosen to make sure that $P_{katz}(y \mid x)$ is a proper probability

$$\alpha(x) = 1 - \sum_y \frac{c^*(xy)}{c(x)}$$

Backoff Smoothing: Katz Backoff

x	$c(x)$	$c^*(x)$	$\frac{c^*(x)}{c(\mathit{the})}$
the	48		
the,dog	15	14.5	$14.5/48$
the,woman	11	10.5	$10.4/48$
the,man	10	9.5	$9.5/48$
the,park	5	4.5	$4.5/48$
the,job	2	1.5	$4.5/48$
the,telescope	1	0.5	$0.5/48$
the,manual	1	0.5	$0.5/48$
the,afternoon	1	0.5	$0.5/48$
the,country	1	0.5	$0.5/48$
the,street	1	0.5	$0.5/48$
TOTAL			0.9479
the,UNK	0		0.052

Backoff Smoothing with Discounting

- Witten-Bell discounting
  use the $n-1$ gram model when the n gram model has too few unique words in the n gram context
- Absolute discounting (Ney, Essen, Kneser)

$$P_{abs}(y \mid x) = \begin{cases} \frac{c(xy)-D}{c(x)} & \text{if } c(xy) > 0 \\ \alpha(x) P_{abs}(y) & \text{otherwise} \end{cases}$$

compute $\alpha(x)$ as was done in Katz smoothing

Language models

n-grams for Language Modeling

Smoothing *n*-gram Models

- Smoothing Counts
  - Add-one Smoothing
  - Additive Smoothing
  - Good-Turing Smoothing
- Smoothing by Interpolation
  - Interpolation: Jelinek-Mercer Smoothing
- Backoff Smoothing
  - Katz Backoff
  - Backoff Smoothing with Discounting

Evaluating Language Models

Evaluating Language Models

- So far we've seen the probability of a sentence: $P(w_0, \ldots, w_n)$
- What is the probability of a collection of sentences, that is what is the probability of an unseen test corpus T
- Let $T = s_0, \ldots, s_m$ be a test corpus with sentences s_i
- T is assumed to be separate from the training data used to train our language model $P(s)$
- What is $P(T)$?

Evaluating Language Models: Independence assumption

- $T = s_0, \ldots, s_m$ is the text corpus with sentences s_0 through s_m
- $P(T) = P(s_0, s_1, s_2, \ldots, s_m)$ – but each sentence is independent from the other sentences
- $P(T) = P(s_0) \cdot P(s_1) \cdot P(s_2) \cdot \ldots \cdot P(s_m) = \prod_{i=0}^{m} P(s_i)$
- $P(s_i) = P(w_0^i, \ldots, w_n^i)$ – which can be any n-gram language model
- A language model is better if the value of $P(T)$ is higher for unseen sentences T, we want to maximize:

$$P(T) = \prod_{i=0}^{m} P(s_i)$$

Evaluating Language Models: Computing the Average

- However, T can be any arbitrary size
- $P(T)$ will be lower if T is larger.
- Instead of the probability for a given T we can compute the *average* probability.
- M is the total number of tokens in the test corpus T:

$$M = \sum_{i=1}^{m} \text{length}(s_i)$$

- The average *log* probability of the test corpus T is:

$$\frac{1}{M} \log_2 \prod_{i=1}^{m} P(s_i) = \frac{1}{M} \sum_{i=1}^{m} \log_2 P(s_i)$$

Evaluating Language Models: Perplexity

- The average *log* probability of the test corpus T is:

$$\ell = \frac{1}{M} \sum_{i=1}^{m} \log_2 P(s_i)$$

- Note that ℓ is a negative number
- We evaluate a language model using *Perplexity* which is $2^{-\ell}$

Evaluating Language Models

Question

Show that:

$$2^{-\frac{1}{M}\log_2 \prod_{i=1}^{m} P(s_i)} = \frac{1}{\sqrt[M]{\prod_{i=1}^{m} P(s_i)}}$$

Evaluating Language Models

Question

What happens to $2^{-\ell}$ if any n-gram probability for computing $P(T)$ is zero?

Evaluating Language Models: Typical Perplexity Values

From 'A Bit of Progress in Language Modeling' by Chen and Goodman

Model	Perplexity
unigram	955
bigram	137
trigram	74

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 4: Event space in Language Models

Trigram Models

- The trigram model:
  $P(w_1, w_2, \ldots, w_n) =$
  $P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times P(w_4 \mid w_2, w_3) \times$
  $\ldots P(w_i \mid w_{i-2}, w_{i-1}) \ldots \times P(w_n \mid w_{n-2}, \ldots, w_{n-1})$
- Notice that the length of the sentence n is variable
- What is the event space?

The stop symbol

- Let $\mathcal{V} = \{a, b\}$ and the language L be $\mathcal{V}^*$
- Consider a unigram model: $P(a) = P(b) = 0.5$
- So strings in this language L are:

$$\begin{array}{rc} a \text{ stop} & 0.5 \\ b \text{ stop} & 0.5 \\ aa \text{ stop} & 0.5^2 \\ bb \text{ stop} & 0.5^2 \\ & \vdots \end{array}$$

- The sum over all strings in L should be equal to 1:

$$\sum_{w \in L} P(w) = 1$$

- But $P(a) + P(b) + P(aa) + P(bb) = 1.5$!!

The `stop` symbol

- What went wrong?
  We need to model variable length sequences
- Add an explicit probability for the stopsymbol:

$$P(a) = P(b) = 0.25$$

$$P(\text{stop}) = 0.5$$

- $P(\text{stop}) = 0.5$, $P(a \text{ stop}) = P(b \text{ stop}) = 0.25 \times 0.5 = 0.125$, $P(aa \text{ stop}) = 0.25^2 \times 0.5 = 0.03125$ (now the sum is no longer greater than one)

The `stop` symbol

- With this new `stop` symbol we can show that $\sum_w P(w) = 1$
  Notice that the probability of any sequence of length n is $0.25^n \times 0.5$
  Also there are 2^n sequences of length n

$$\sum_w P(w) =$$

$$\sum_{n=0}^{\infty} 2^n \times 0.25^n \times 0.5$$

$$\sum_{n=0}^{\infty} 0.5^n \times 0.5 = \sum_{n=0}^{\infty} 0.5^{n+1}$$

$$\sum_{n=1}^{\infty} 0.5^n = 1$$

The $\texttt{stop}$ symbol

- With this new $\texttt{stop}$ symbol we can show that $\sum_w P(w) = 1$
  Using $p_s = P(\texttt{stop})$ the probability of any sequence of length n is $p(n) = p(w_1, \ldots, w_{n-1}) \times p_s(w_n)$

$$
\begin{aligned}
\sum_w P(w) &= \sum_{n=0}^{\infty} p(n) \sum_{w_1, \ldots, w_n} p(w_1, \ldots, w_n) \\
&= \sum_{n=0}^{\infty} p(n) \sum_{w_1, \ldots, w_n} \prod_{i=1}^{n} p(w_i)
\end{aligned}
$$

$$
\sum_{w_1, \ldots, w_n} \prod_i p(w_i) =
$$

$$
\sum_{w_1} \sum_{w_2} \ldots \sum_{w_n} p(w_1) p(w_2) \ldots p(w_n) = 1
$$

The stop symbol

$$\sum_{w_1}\sum_{w_2}\dots\sum_{w_n} p(w_1)p(w_2)\dots p(w_n) \quad = \quad 1$$

$$\begin{aligned}
\sum_{n=0}^{\infty} p(n) &= \sum_{n=0}^{\infty} p_s(1-p_s)^n \\
&= p_s \sum_{n=0}^{\infty}(1-p_s)^n \\
&= p_s \frac{1}{1-(1-p_s)} = p_s \frac{1}{p_s} = 1
\end{aligned}$$

Acknowledgements

Many slides borrowed or inspired from lecture notes by Michael Collins, Chris Dyer, Adam Lopez, and Luke Zettlemoyer from their NLP course materials. All mistakes are my own.