N-gram Language Models

COMP90042

Natural Language Processing

Lecture 3

Language Models

- NLP is about *explaining language*
  - Why some sentences are more **fluent** than others
- E.g. in speech recognition:
  - *recognise speech* > *wreck a nice beach*
- We measure ‘goodness’ using **probabilities** estimated by language models
- Language model can also be used for **generation**

Talk to Transformer

See how a modern neural network completes your text. Type a custom snippet or try one of the examples. Learn more below.

Follow @AdamDanielKing for more neat neural networks.

Custom prompt

Today we have a lecture on Natural Language Processing.

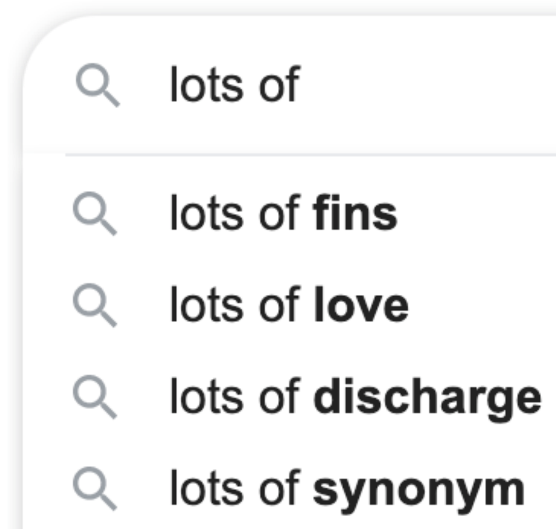
GENERATE ANOTHER

Completion

Today we have a lecture on Natural Language Processing. It's a very unusual subject, but a very interesting subject, and this is the first time that it's been covered in a Livefist podcast. And, of course, we're going to talk about what makes Natural Language Processing a little different from other language processing areas: namely its social nature. We're going to talk about the question of artificial intelligence, which is something that we've spoken about here and a lot of people on the Livefist messageboard talk about a lot. What is the nature of AI? Is it an essentially self-regulating system, an ultimate mind, the best machine of its kind? What is the nature of intelligence, and how does it work?

Language Models

- Useful for
  - Speech recognition
  - Spelling correction
  - Query completion
  - Optical character recog.
- Other generation tasks
  - Machine translation
  - Summarisation
  - Dialogue systems



Outline

- Deriving *n*-gram language models
- Smoothing to deal with sparsity
- Generating Language
- Evaluating language models

Probabilities: Joint to Conditional

Our goal is to get a probability for an arbitrary sequence of *m* words

$$P(w_1, w_2, \ldots, w_m)$$

First step is to apply the chain rule to convert joint probabilities to conditional ones

$$P(w_1, w_2, \ldots, w_m) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2)\ldots P(w_m | w_1 \ldots w_{m-1})$$

The Markov Assumption

Still intractable, so make a simplifying assumption:

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})$$

For some small *n*

When *n* = 1, a unigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i)$$

When *n* = 2, a bigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-1})$$

When *n* = 3, a trigram model

$$P(w_1, w_2, .. w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-2} w_{i-1})$$

Maximum Likelihood Estimation

How do we calculate the probabilities? Estimate based on counts in our corpus:

For unigram models,

$$P(w_i) = \frac{C(w_i)}{M}$$

For bigram models,

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

For n-gram models generally,

$$P(w_i \mid w_{i-n+1} \dots w_{i-1}) = \frac{C(w_{i-n+1} \dots w_i)}{C(w_{i-n+1} \dots w_{i-1})}$$

Book-ending Sequences

- Special tags used to denote start and end of sequence
  - <s> = sentence start (□ in E18)
  - </s> = sentence end (■ in E18)

Trigram example

Corpus:

<s> <s> *yes no no no no yes* </s>
<s> <s> *no no no yes yes yes no* </s>

What is the probability of

<s> <s> *yes no no yes* </s>

under a trigram language model?

P(sent = yes no no yes)

= P(*yes* | <s> <s>) * P(*no* | <s> *yes*) * P(*no* | *yes no*) * P(*yes* | *no no*) * P(</s> | *no yes*)

$= \frac{1}{2} * 1 * \frac{1}{2} * \frac{2}{5} * \frac{1}{2} = 0.1$

5 "*no no*" bigrams

Several Problems

- Language has long distance effects — need large *n*
  - *The lecture/s that took place last week was/were on preprocessing.*
- Resulting probabilities are often very small
  - Use log probability to avoid numerical underflow
- No probabilities for unseen words
  - Special symbol to represent them (e.g. <UNK>)
- Words in new contexts
  - By default, zero count for any *n*-gram we've never seen before, zero probability for the sentence
  - Need to smooth the LM!

Smoothing

Smoothing

- Basic idea: give events you've never seen before some probability
- Must be the case that *P*(everything) = 1
- Many different kinds of smoothing
  - Laplacian (add-one) smoothing
  - Add-*k* smoothing
  - Jelinek-Mercer interpolation
  - Katz backoff
  - Absolute discounting
  - Kneser-Ney
  - And others…

Laplacian (Add-one) Smoothing

- Simple idea: pretend we've seen each *n*-gram once more than we did.

For unigram models (**V**= the vocabulary),

$$P_{add1}(w_i) = \frac{C(w_i) + 1}{M + |V|}$$

For bigram models,

$$P_{add1}(w_i | w_{i-1}) = \frac{C\left(w_{i-1}w_i\right) + 1}{C\left(w_{i-1}\right) + |V|}$$

Add-one Example

<s> *the rat ate the cheese* </s>

What's the bigram probability *P(ate*|*rat*) under add-one smoothing?

$$= \frac{C(rat\ ate) + 1}{C(rat) + |V|} = \frac{2}{6}$$

V = { the, rat, ate, cheese, </s> }

What's the bigram probability *P(ate*|*cheese*) under add-one smoothing?

$$= \frac{C\left(cheese\ ate\right) + 1}{C\left(cheese\right) + |V|} = \frac{1}{6}$$

Add-*k* Smoothing

- Adding one is often too much
- Instead, add a fraction *k*
- AKA Lidstone Smoothing

$$P_{addk}(w_i \mid w_{i-1}, w_{i-2}) = \frac{C(w_{i-2},\ w_{i-1}, w_i) + k}{C(w_{i-2}, w_{i-1}) + k\,|\boldsymbol{V}|}$$

- Have to choose *k*
  - number of “classes” is huge (n-grams), so can have a big effect

Lidstone Smoothing

Context = *alleged*

- 5 observed bi-grams
- 2 unobserved bi-grams

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$ effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391
offense	5	0.25	4.928	0.246
damage	4	0.2	3.961	0.198
deficiencies	2	0.1	2.029	0.101
outbreak	1	0.05	1.063	0.053
infirmity	0	0	0.097	0.005
cephalopods	0	0	0.097	0.005
	20	1.0	20	1.0

(8 + 0.1) / (20 + 7 x 0.1)

Lidstone Smoothing

Context = *alleged*

- 5 observed n-grams
- 2 unobserved n-grams

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$ effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391
offense	5	0.25	4.928	0.246
damage	4	0.2	3.961	0.198
deficiencies	2	0.1	2.029	0.101
outbreak	1	0.05	1.063	0.053
infirmity	0	0	0.097	0.005
cephalopods	0	0	0.097	0.005
	20	1.0	20	1.0

(8 + 0.1) / (20 + 7 x 0.1)

(0 + 0.1) / (20 + 7 x 0.1)

Lidstone Smoothing

Context = *alleged*

- 5 observed n-grams
- 2 unobserved n-grams

			Lidstone smoothing, $\alpha = 0.1$	
	counts	unsmoothed probability	effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391
offense	5	0.25	4.928	0.246
damage	4	0.2	3.961	0.198
deficiencies	2	0.1	2.029	0.101
outbreak	1	0.05	1.063	0.053
infirmity	0	0	0.097	0.005
cephalopods	0	0	0.097	0.005
	20	1.0	20	1.0

0.391 x 20

(8 + 0.1) / (20 + 7 x 0.1)

(0 + 0.1) / (20 + 7 x 0.1)

Absolute Discounting

- ‘Borrows’ a **fixed** probability mass from observed n-gram counts
- Redistributes it to unseen n-grams

Absolute Discounting

Context = *alleged*

- 5 observed n-grams
- 2 unobserved n-grams

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$ effective counts	smoothed probability	Discounting, $d = 0.1$ effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391	7.9	0.395
offense	5	0.25	4.928	0.246	4.9	0.245
damage	4	0.2	3.961	0.198	3.9	0.195
deficiencies	2	0.1	2.029	0.101	1.9	0.095
outbreak	1	0.05	1.063	0.053	0.9	0.045
infirmity	0	0	0.097	0.005	0.25	0.013
cephalopods	0	0	0.097	0.005	0.25	0.013

(5 x 0.1) / 2

8 - 0.1

Backoff

- Absolute discounting redistributes the probability mass **equally** for all unseen n-grams
- Katz Backoff: redistributes the mass based on a **lower order** model (e.g. unigram)

$$P_{katz}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1},w_i)-D}{C(w_{i-1})}, & \text{if } C(w_{i-1},w_i) > 0 \\ \alpha(w_{i-1}) \times \frac{P(w_i)}{\sum_{w_j:C(w_{i-1},w_j)=0} P(w_j)}, & \text{otherwise} \end{cases}$$

unigram probability for w_i

sum unigram probabilities for all words that do not co-occur with context w_{i-1}

the amount of probability mass that has been discounted for context w_{i-1} (0.5 in previous slide)

Issues with Katz Backoff

$$P_{katz}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1},w_i)-D}{C(w_{i-1})}, & \text{if } C(w_{i-1},w_i) > 0 \\ \alpha(w_{i-1}) P(w_i), & \text{otherwise} \end{cases}$$

- *I can't see without my reading ___*
- C(*reading*, *glasses*) = C(*reading*, *Francisco*) = 0
- C(*Fransciso*) > C(*glasses*)
- Katz backoff will give higher probability to *Francisco*

Kneser-Ney Smoothing

- Redistribute probability mass based on how many number of **different contexts** word *w* has appeared in
- This measure is called “continuation probability”

Kneser-Ney Smoothing

$$P_{KN}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1},w_i)-D}{C(w_{i-1})}, & \text{if } C(w_{i-1},w_i) > 0 \\ \alpha(w_{i-1}) P_{cont}(w_i), & \text{otherwise} \end{cases}$$

where

$$P_{cont}(w_i) = \frac{|\{w_{i-1} : C(w_{i-1}, w_i) > 0\}|}{\sum_{w_i} |\{w_{i-1} : C(w_{i-1}, w_i) > 0\}|}$$

- High continuation counts for *glasses* (*men's glasses*, *black glasses*, *buy glasses*, *prescription glasses*, *etc)*
- Low continuation counts for *Franciso* (*San Francisco*)

Interpolation

- A better way to combine different order *n*-gram models
- Weighted sum of probabilities across progressively shorter contexts
- Interpolated trigram model:

$$\begin{aligned}\mathrm{p}_{\text{Interpolation}}(w_m \mid w_{m-1}, w_{m-2}) &= \lambda_3 \mathrm{p}_3^*(w_m \mid w_{m-1}, w_{m-2}) \\ &+ \lambda_2 \mathrm{p}_2^*(w_m \mid w_{m-1}) \\ &+ \lambda_1 \mathrm{p}_1^*(w_m).\end{aligned}$$

Learned based on held out data

$$\sum_{n=1}^{n_{\max}} \lambda_n = 1$$

Interpolated Kneser-Ney Smoothing

- Interpolation instead of back-off

$$P_{IKN}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) - D}{C(w_{i-1})} + \beta(w_{i-1})P_{cont}(w_i)$$

- where $\beta(w_{i-1})$ =
  - normalising constant so that $P_{IKN}(w_i|w_{i-1})$ sums to 1.0

In Practice

- Commonly used Kneser-Ney language models use 5-grams as max order
- Has different discount values for each n-gram order.

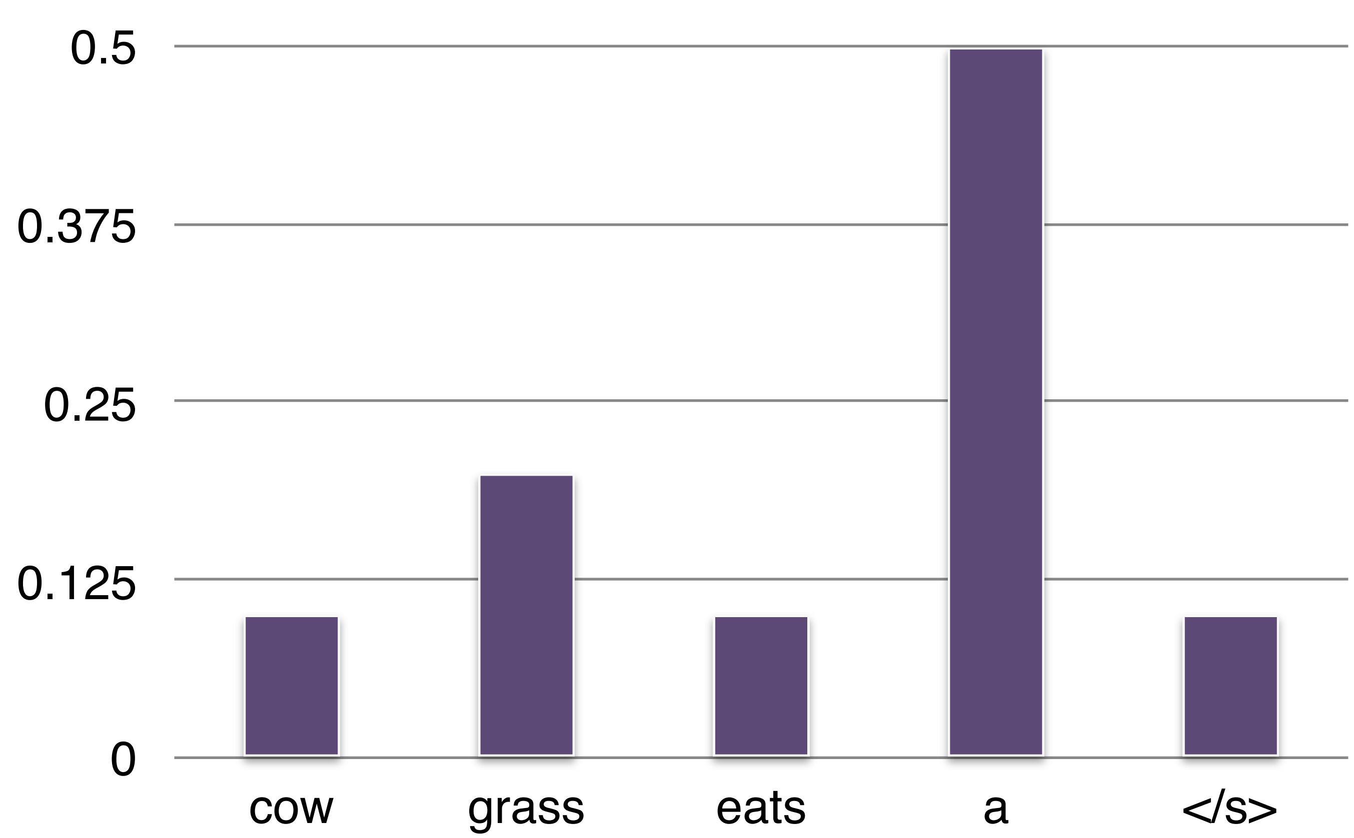
Generating Language

Generation

- Given an initial word, draw the next word according to the probability distribution defined by the language model.
- Include (n-1) start tokens for n-gram model, to provide context to generate first word
  - never generate <s>
  - generating </s> terminates the sequence

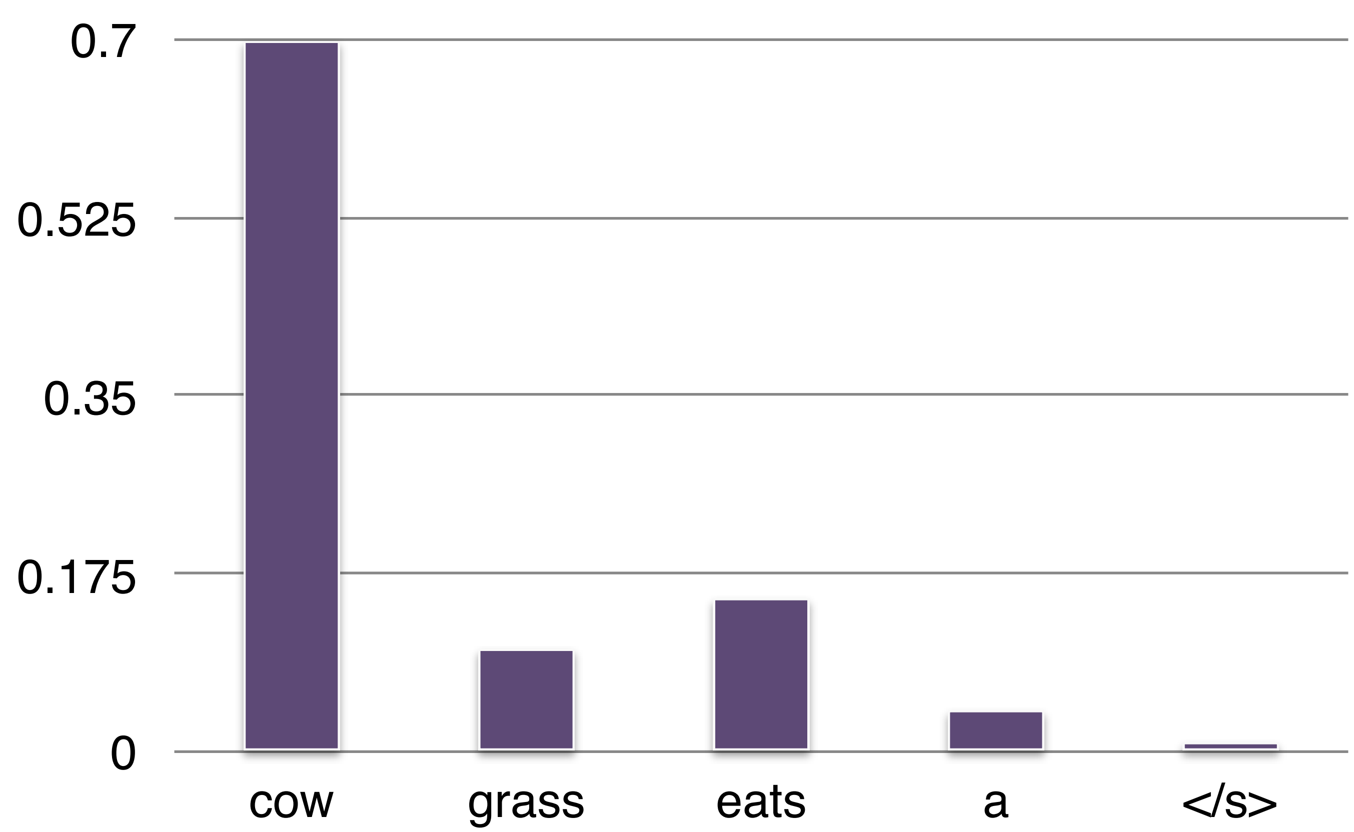
Generation (Bigram LM)

- Sentence = <s>
- P(? | <s>) = "*a*"



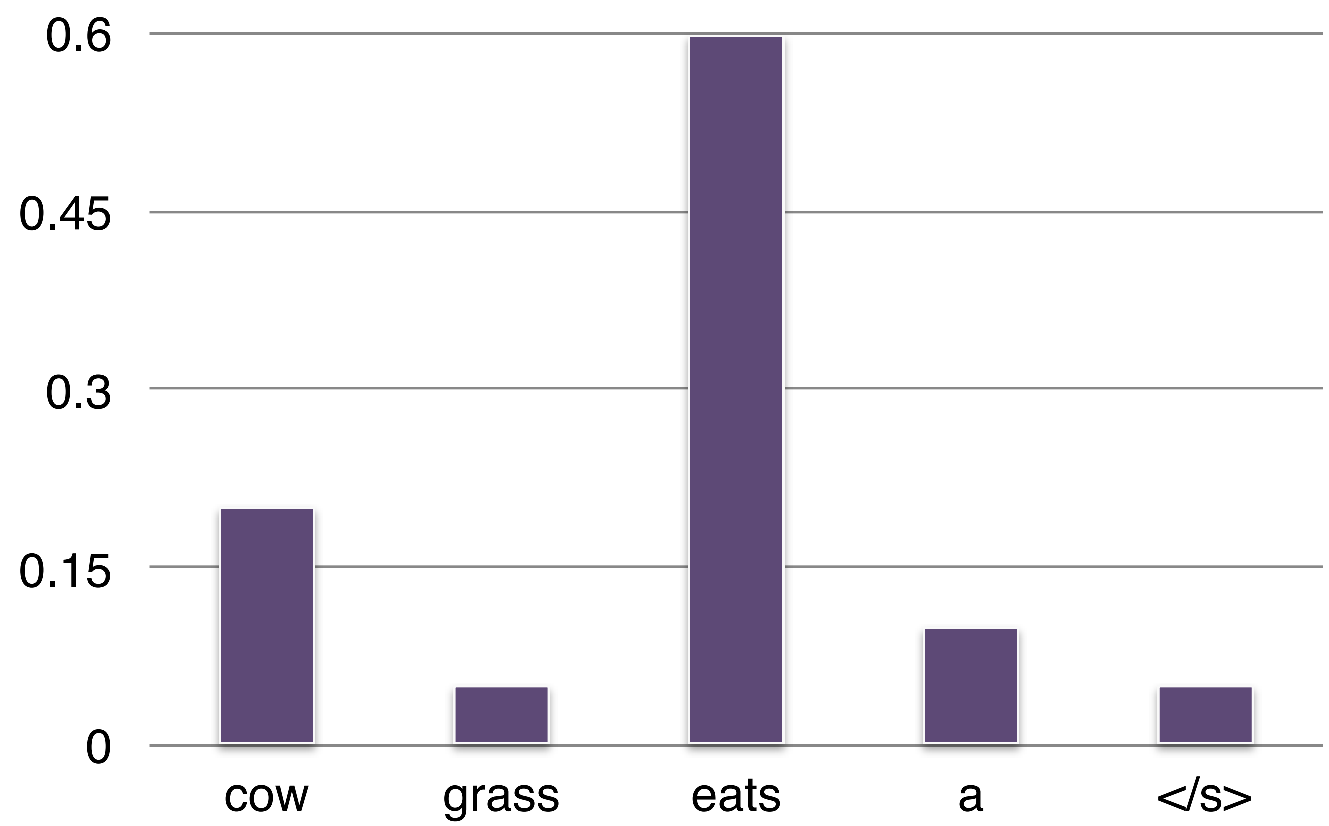
Generation (Bigram LM)

- Sentence = <s> *a*
- P(? | a) = "*cow*"



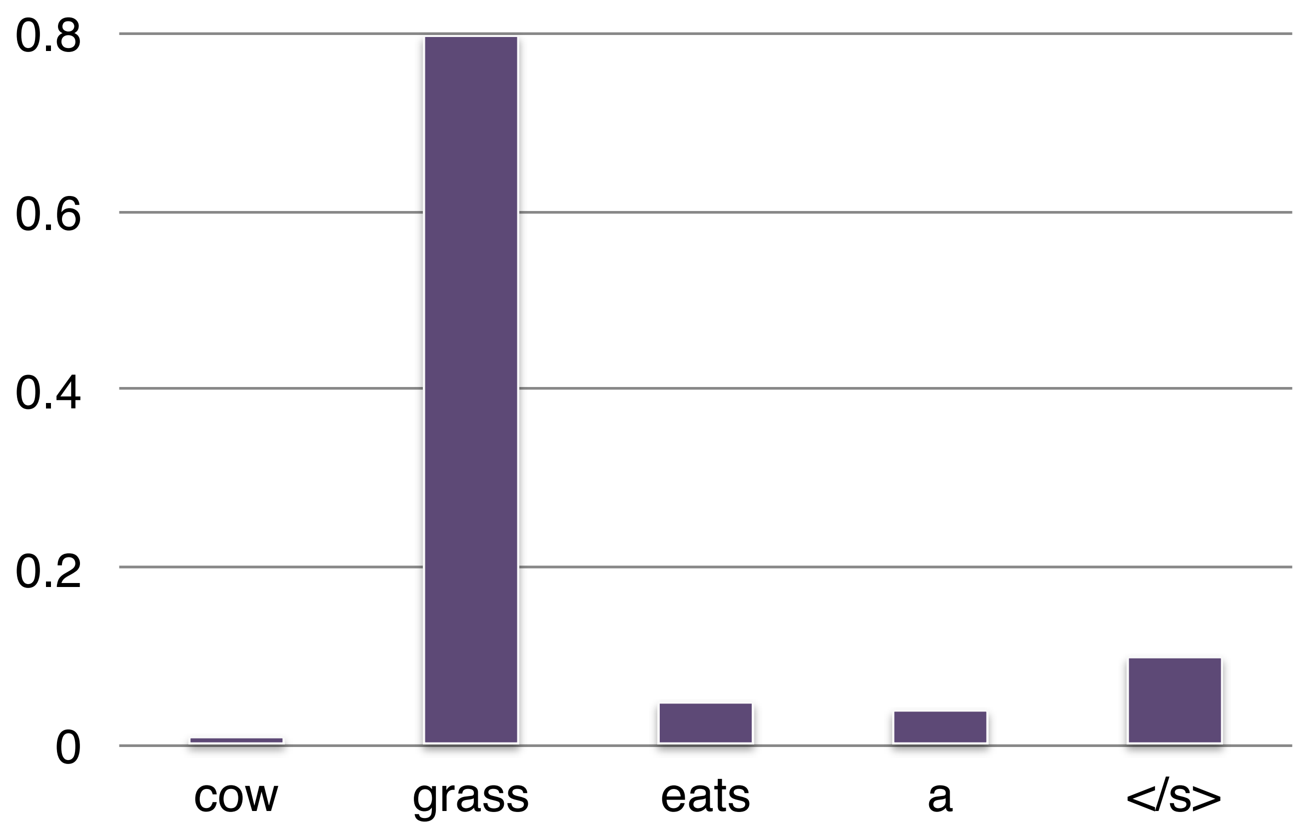
Generation (Bigram LM)

- Sentence = <s> *a cow*
- P(? | cow) = "*eats*"



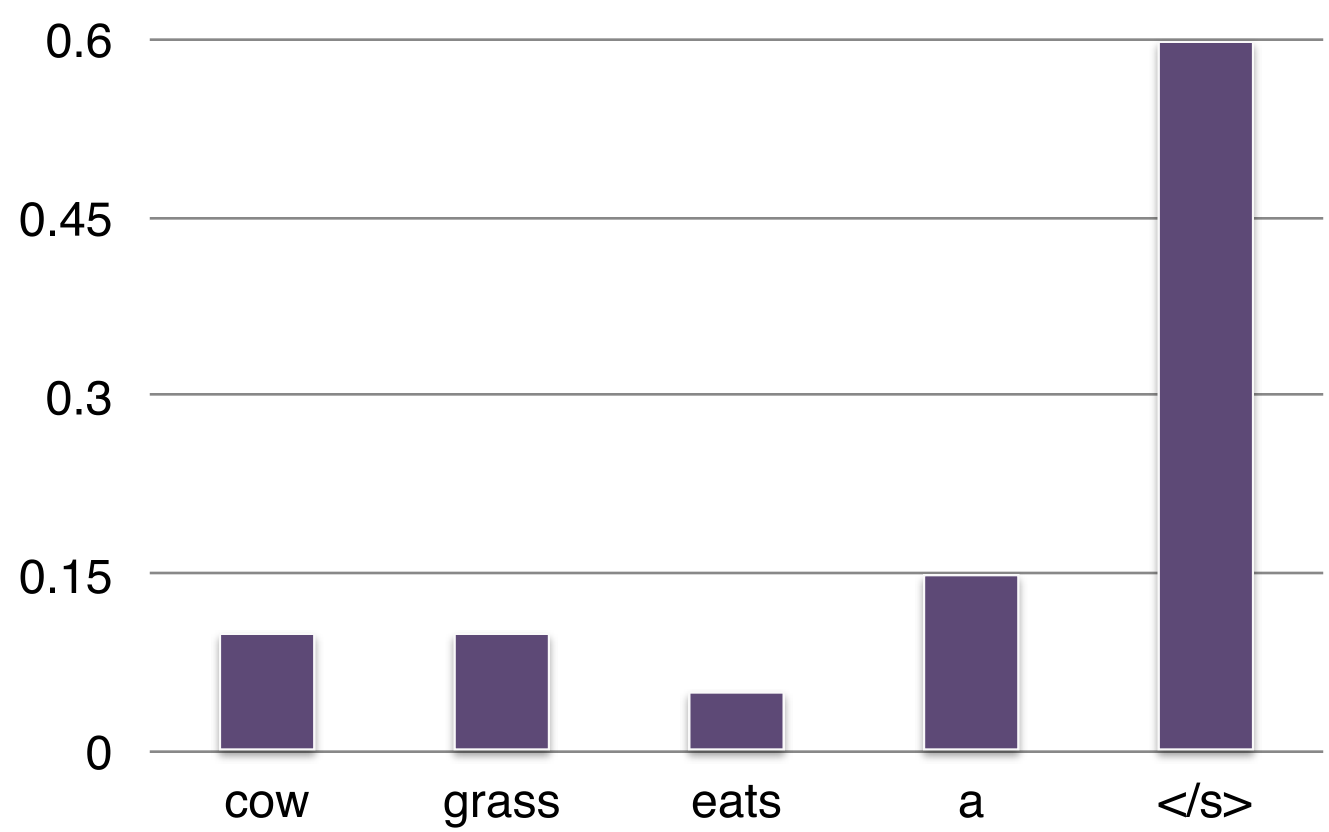
Generation (Bigram LM)

- Sentence = <s> *a cow eats*
- P(? | eats) = "*grass*"



Generation (Bigram LM)

- Sentence = <s> *a cow eats grass*
- P(? | grass) = “*</s>*”



Generation (Bigram LM)

- Sentence = <s> *a cow eats grass* </s>
- Done!

How to Select Next Word?

- Argmax: takes highest probability word each turn
  - Greedy search
- Beam search decoding:
  - Keeps track of top-N highest probability words each turn
  - Produces sentences with near-optimal sentence probability
- Randomly samples from the distribution (e.g. temperature sampling)

Evaluating Language Models

Evaluation

- Extrinsic
  - E.g. spelling correction, machine translation
- Intrinsic
  - Perplexity on held-out test set

Perplexity

- Inverse probability of entire test set
  - Normalized by number of word tokens (including </s>)
- The lower the better

$$PP(w_1, w_2, .. w_m) = \sqrt[m]{\frac{1}{P(w_1, w_2, .. w_m)}}$$

equivalently

$$PP(w_1, w_2, .. w_m) = 2^{-\frac{\log_2 P(w_1,\ w_2,\ ..\ w_m)}{m}}$$

- Unknown (OOV) words a problem (omit)

Example Perplexity Scores

	Unigram	Bigram	Trigram
Perplexity	962	170	109

- Corpus: Wall Street Journal
- Train partition: 38 million word tokens, almost 20K word types (vocabulary)
- Test partition: 1.5 million word tokens

A Final Word

- *N*-gram language models are a simple but effective way to capture the predictability of language
- Information can be derived in an unsupervised fashion, scalable to large corpora
- Require smoothing to be effective, due to sparsity

Reading

- E18 Chapter 6 (skip 6.3)