Accelerated Natural Language Processing
Lecture 5
N-gram models, entropy

Sharon Goldwater
(some slides based on those by Alex Lascarides and Philipp Koehn)

24 September 2019

School of informatics

Recap: Language models

- **Language models** tell us $P(\vec{w}) = P(w_1 \dots w_n)$: *How likely to occur is this sequence of words?*

  Roughly: *Is this sequence of words a "good" one in my language?*

Example uses of language models

- Machine translation: reordering, word choice.

$$P_{\text{LM}}(\text{the house is small}) > P_{\text{LM}}(\text{small the is house})$$
$$P_{\text{LM}}(\text{I am going home}) > P_{\text{LM}}(\text{I am going house})$$
$$P_{\text{LM}}(\text{We'll start eating}) > P_{\text{LM}}(\text{We shall commence consuming})$$

- Speech recognition: word choice:

$$P_{\text{LM}}(\text{morphosyntactic analyses}) > P_{\text{LM}}(\text{more faux syntactic analyses})$$
$$P_{\text{LM}}(\text{I put it on today}) > P_{\text{LM}}(\text{I putted onto day})$$

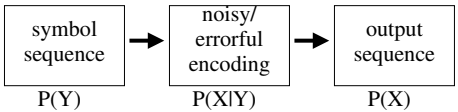
But: How do systems use this information?

Today's lecture:

- What is the Noisy Channel framework and what are some example uses?
- What is a language model?
- What is an n-gram model, what is it for, and what independence assumptions does it make?
- What are entropy and perplexity and what do they tell us?
- What's wrong with using MLE in n-gram models?

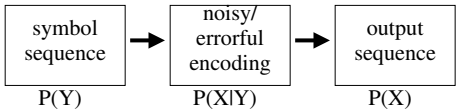
Noisy channel framework

- Concept from Information Theory, used widely in NLP
- We imagine that the observed data (output sequence) was generated as:



Noisy channel framework

- Concept from Information Theory, used widely in NLP
- We imagine that the observed data (output sequence) was generated as:



Application	Y	X
Speech recognition	true words	acoustic signal
Machine translation	words in L_1	words in L_2
Spelling correction	true words	typed words

Example: spelling correction

- $P(Y)$: Distribution over the words (sequences) the user intended to type. A **language model**.
- $P(X|Y)$: Distribution describing what user is **likely** to type, given what they **meant**. Could incorporate information about common spelling errors, key positions, etc. Call it a **noise model**.
- $P(X)$: Resulting distribution over what we actually see.
- Given some particular observation x (say, effert), we want to recover the most probable y that was intended.

Noisy channel as probabilistic inference

- Mathematically, what we want is $\operatorname{argmax}_y P(y|x)$.
  - Read as "the y that maximizes $P(y|x)$"
- Rewrite using Bayes' Rule:

$$\begin{aligned} \operatorname*{argmax}_y P(y|x) &= \operatorname*{argmax}_y \frac{P(x|y)P(y)}{P(x)} \\ &= \operatorname*{argmax}_y P(x|y)P(y) \end{aligned}$$

Noisy channel as probabilistic inference

So to recover the best y, we will need

- a **language model** $P(Y)$: relatively task-independent.
- a **noise model** $P(X|Y)$, which depends on the task.
  - acoustic model, translation model, misspelling model, etc.
  - won't discuss here; see courses on ASR, MT.

Both are normally trained on corpus data.

You may be wondering

If we can train $P(X|Y)$, why can't we just train $P(Y|X)$? Who needs Bayes' Rule?

- Answer 1: sometimes we do train $P(Y|X)$ directly. Stay tuned...
- Answer 2: training $P(X|Y)$ or $P(Y|X)$ requires **input/output pairs**, which are often limited:
  - Misspelled words with their corrections; transcribed speech; translated text

  But LMs can be trained on huge unannotated corpora: a better model. Can help improve overall performance.

Estimating a language model

- Y is really a sequence of words $\vec{w} = w_1 \dots w_n$.
- So we want to know $P(w_1 \dots w_n)$ for big n (e.g., sentence).
- What will not work: try to directly estimate probability of each full sentence.
  - Say, using MLE (relative frequencies): $C(\vec{w})/(\text{tot } \# \text{ sentences})$.
  - For nearly all $\vec{w}$ (grammatical or not), $C(\vec{w}) = 0$.
  - A **sparse data** problem: not enough observations to estimate probabilities well.

A first attempt to solve the problem

Perhaps the simplest model of sentence probabilities: a **unigram** model.

- Generative process: choose each word in sentence independently.
- Resulting model: $\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$

A first attempt to solve the problem

Perhaps the simplest model of sentence probabilities: a **unigram** model.

- Generative process: choose each word in sentence independently.
- Resulting model: $\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$
- So, $\mathrm{P}(\text{the cat slept quietly}) = \mathrm{P}(\text{the quietly cat slept})$

A first attempt to solve the problem

Perhaps the simplest model of sentence probabilities: a **unigram** model.

- Generative process: choose each word in sentence independently.
- Resulting model: $\hat{P}(\vec{w}) = \prod_{i=1}^{n} P(w_i)$
- So, $\mathrm{P}(\text{the cat slept quietly}) = \mathrm{P}(\text{the quietly cat slept})$
  - Not a *good* model, but still a model.
- Of course, $P(w_i)$ also needs to be estimated!

MLE for unigrams

- How to estimate $P(w)$, e.g., $P(\text{the})$?
- Remember that MLE is just relative frequencies:

$$P_{\mathrm{ML}}(w) = \frac{C(w)}{W}$$

  - $C(w)$ is the token count of w in a large corpus
  - $W = \sum_{x'} C(x')$ is the total number of word tokens in the corpus.

Unigram models in practice

- Seems like a pretty bad model of language: probability of word obviously *does* depend on context.
- Yet unigram (or **bag-of-words**) models are surprisingly useful for some applications.
  - Can model "aboutness": topic of a document, semantic usage of a word
  - Applications: lexical semantics (disambiguation), information retrieval, text classification. (See later in this course)
  - But, for now we will focus on models that capture at least some syntactic information.

General N-gram language models

Step 1: rewrite using chain rule:

$$P(\vec{w}) = P(w_1 \dots w_n)$$
$$= P(w_n|w_1, w_2, \dots, w_{n-1})P(w_{n-1}|w_1, w_2, \dots, w_{n-2}) \dots P(w_1)$$

- Example: $\vec{w}$ = the cat slept quietly yesterday.

  P(the, cat, slept, quietly, yesterday) =
  P(yesterday|the, cat, slept, quietly) · P(quietly|the, cat, slept)·
  P(slept|the, cat) · P(cat|the) · P(the)

- But for long sequences, many of the conditional probs are also too sparse!

General N-gram language models

Step 2: make an independence assumption:

$$P(\vec{w}) = P(w_1 \dots w_n)$$
$$= P(w_n|w_1, w_2, \dots, w_{n-1})P(w_{n-1}|w_1, w_2, \dots, w_{n-2}) \dots P(w_1)$$
$$\approx P(w_n|w_{n-2}, w_{n-1})P(w_{n-1}|w_{n-3}, w_{n-2}) \dots P(w_1)$$

- **Markov** assumption: only a finite history matters.

- Here, two word history (**trigram** model):
  w_i is cond. indep. of $w_1 \dots w_{i-3}$ given w_{i-1}, w_{i-2}.

  P(the, cat, slept, quietly, yesterday) ≈
  P(yesterday|slept, quietly) · P(quietly|cat, slept)·
  P(slept|the, cat) · P(cat|the) · P(the)

Trigram independence assumption

- Put another way, a trigram model assumes these are all equal:
  - P(slept|the cat)
  - P(slept|after lunch the cat)
  - P(slept|the dog chased the cat)
  - P(slept|except for the cat)

  because all are estimated as P(slept|the cat)

- Not always a good assumption! But it does reduce the sparse data problem.

Another example: bigram model

- Bigram model assumes one word history:

$$P(\vec{w}) = P(w_1) \prod_{i=2}^{n} P(w_i|w_{i-1})$$

- But consider these sentences:

	w_1	w_2	w_3	w_4
(1)	the	cats	slept	quietly
(2)	feeds	cats	slept	quietly
(3)	the	cats	slept	on

- What's wrong with (2) and (3)? Does the model capture these problems?

Example: bigram model

- To capture behaviour at beginning/end of sentence, we need to augment the input:

	w_0	w_1	w_2	w_3	w_4	w_5
(1)	<s>	the	cats	slept	quietly	</s>
(2)	<s>	feeds	cats	slept	quietly	</s>
(3)	<s>	the	cats	slept	on	</s>

- That is, assume $w_0 =$ <s> and $w_{n+1} =$ </s> so we have:

$$P(\vec{w}) = P(w_0) \prod_{i=1}^{n+1} P(w_i|w_{i-1}) = \prod_{i=1}^{n+1} P(w_i|w_{i-1})$$

Estimating N-Gram Probabilities

- Maximum likelihood (relative frequency) estimation for bigrams:
  - How many times we saw w_2 following w_1,
    out of all the times we saw *anything* following w_1:

$$P_{\text{ML}}(w_2|w_1) = \frac{C(w_1, w_2)}{C(w_1, \cdot)}$$
$$= \frac{C(w_1, w_2)}{C(w_1)}$$

Estimating N-Gram Probabilities

- Similarly for trigrams:

$$P_{\text{ML}}(w_3|w_1, w_2) = \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)}$$

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

Evaluating a language model

- Intuitively, trigram model captures more context than bigram model, so should be a "better" model.
- That is, more accurately predict the probabilities of sentences.
- But how can we measure this?

Entropy

- Definition of the **entropy** of a random variable X:

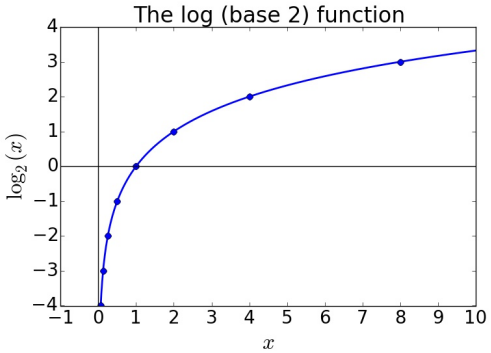
  $H(X) = \sum_x -P(x)\ \log_2 P(x)$

- Intuitively: a measure of uncertainty/disorder

- Also: the expected value of $-\log_2 P(X)$

Reminder: logarithms

$\log_a x = b$
iff
$a^b = x$

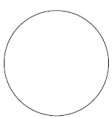


Entropy Example

One event (outcome)

$P(a) = 1$

$H(X) = -1 \log_2 1$
$= 0$

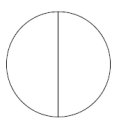


Entropy Example

2 equally likely events:

$P(a) = 0.5$
$P(b) = 0.5$

$H(X) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5$
$= -\log_2 0.5$
$= 1$

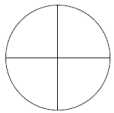


Entropy Example

4 equally likely events:

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

$$\begin{aligned} H(X) = & -0.25\log_2 0.25 - 0.25\log_2 0.25 \\ & -0.25\log_2 0.25 - 0.25\log_2 0.25 \\ = & -\log_2 0.25 \\ = & 2 \end{aligned}$$



Entropy Example

3 equally likely events and one more likely than the others:

$P(a) = 0.7$
$P(b) = 0.1$
$P(c) = 0.1$
$P(d) = 0.1$

$$\begin{aligned} H(X) = & -0.7\log_2 0.7 - 0.1\log_2 0.1 \\ & -0.1\log_2 0.1 - 0.1\log_2 0.1 \\ = & -0.7\log_2 0.7 - 0.3\log_2 0.1 \\ = & -(0.7)(-0.5146) - (0.3)(-3.3219) \\ = & 0.36020 + 0.99658 \\ = & 1.35678 \end{aligned}$$

Entropy Example

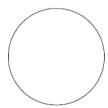
3 equally likely events and one much more likely than the others:

$P(a) = 0.97$
$P(b) = 0.01$
$P(c) = 0.01$
$P(d) = 0.01$

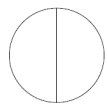
$$\begin{aligned} H(X) = & -0.97\log_2 0.97 - 0.01\log_2 0.01 \\ & -0.01\log_2 0.01 - 0.01\log_2 0.01 \\ = & -0.97\log_2 0.97 - 0.03\log_2 0.01 \\ = & -(0.97)(-0.04394) - (0.03)(-6.64 \\ = & 0.04262 + 0.19932 \\ = & 0.24194 \end{aligned}$$

Entropy take-aways

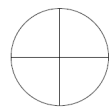
Entropy of a **uniform distribution** over N outcomes is $\log_2 N$:



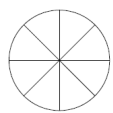
$H(X) = 0$



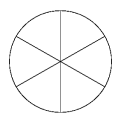
$H(X) = 1$



$H(X) = 2$



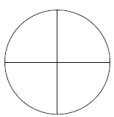
$H(X) = 3$



$H(X) = 2.585$

Entropy take-aways

Any **non-uniform** distribution over N outcomes has **lower** entropy than the corresponding uniform distribution:



$H(X) = 2$

$H(X) = 1.35678$

$H(X) = 0.24194$

Entropy as y/n questions

How many yes-no questions (bits) do we need to find out the outcome?

- Uniform distribution with 2^n outcomes: n q's.
- Other cases: entropy is the average number of questions per outcome in a (very) long sequence of outcomes, where questions can consider multiple outcomes at once.

Entropy as encoding sequences

- Assume that we want to encode a sequence of events X
- Each event is encoded by a sequence of bits
- For example
  - Coin flip: heads = 0, tails = 1
  - 4 equally likely events: a = 00, b = 01, c = 10, d = 11
  - 3 events, one more likely than others: a = 0, b = 10, c = 11
  - Morse code: *e* has shorter code than *q*
- Average number of bits needed to encode $X \geq$ entropy of X

The Entropy of English

- Given the start of a text, can we guess the next word?
- Humans do pretty well: the entropy is only about 1.3.
- But what about N-gram models?
  - Ideal language model would match the true entropy of English.
  - The closer we get to that, the better the model.
  - Put another way, a good model assigns high prob to real sentences (and low prob to everything else).

How good is the LM?

- **Cross entropy** measures how well model M predicts the data.
- For data $w_1 \dots w_n$ with large n, well approximated by:

$$H_M(w_1 \dots w_n) = -\frac{1}{n} \log_2 P_M(w_1 \dots w_n)$$

  – Avg neg log prob our model assigns to each word we saw

- Or, **perplexity**:

$$\text{PP}_M(\vec{w}) = 2^{H_M(\vec{w})}$$

Perplexity

- On paper, there's a simpler expression for perplexity:

$$\begin{aligned} \text{PP}_M(\vec{w}) &= 2^{H_M(\vec{w})} \\ &= 2^{-\frac{1}{n} \log_2 P_M(w_1 \dots w_n)} \\ &= 2^{\log_2 P_M(w_1 \dots w_n)^{-\frac{1}{n}}} \\ &= P_M(w_1 \dots w_n)^{-\frac{1}{n}} \end{aligned}$$

  – 1 over the geometric average of the probabilities of each w_i.

- But in practice, when computing perplexity for long sequences, we use the version with logs (see week 3 lab for reasons...)

Example: trigram (Europarl)

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

- What happens when I try to compute $P(\text{consuming}|\text{shall commence})$?
  - Assume we have seen shall commence in our corpus
  - But we have never seen shall commence consuming in our corpus

Unseen N-Grams

- What happens when I try to compute $P(\text{consuming}|\text{shall commence})$?
  - Assume we have seen shall commence in our corpus
  - But we have never seen shall commence consuming in our corpus

$\rightarrow P(\text{consuming}|\text{shall commence}) = 0$

- Any sentence with shall commence consuming will be assigned probability 0

The guests shall commence consuming supper
Green inked shall commence consuming garden the

The problem with MLE

- MLE estimates probabilities that make the observed data maximally probable
- by assuming anything unseen cannot happen (and also assigning too *much* probability to low-frequency observed events).
- It **over-fits** the training data.
- We tried to avoid zero-probability sentences by modelling with smaller chunks (n-grams), but even these will sometimes have zero prob under MLE.

Next time: **smoothing** methods, which reassign probability mass from observed to unobserved events, to avoid overfitting/zero probs.

Questions for review:

- What is the Noisy Channel framework and what are some example uses?
- What is a language model?
- What is an n-gram model, what is it for, and what independence assumptions does it make?
- What are entropy and perplexity and what do they tell us?
- What's wrong with using MLE in n-gram models?