

NLP: N-Gram Models

Sargur N. Srihari

srihari@cedar.buffalo.edu

This is part of lecture slides on [Deep Learning](http://www.cedar.buffalo.edu/~srihari/CSE676):
<http://www.cedar.buffalo.edu/~srihari/CSE676>

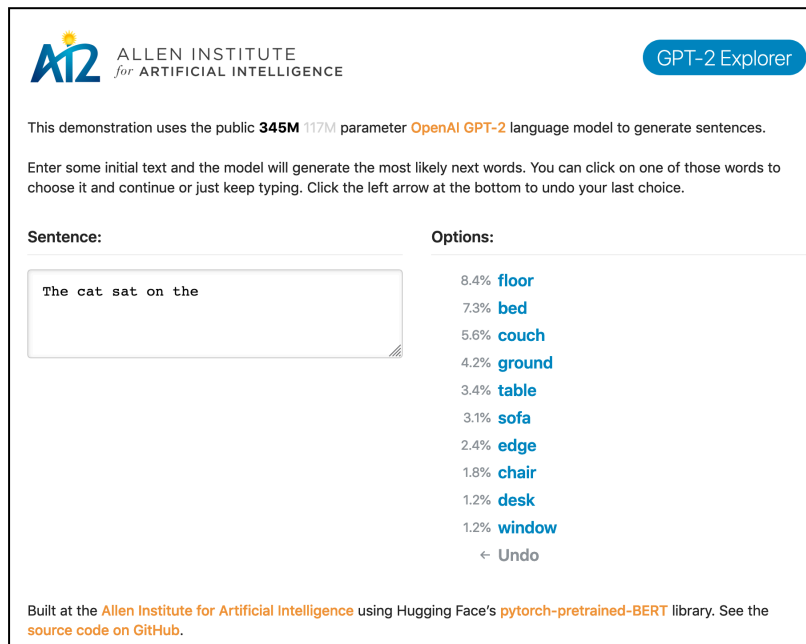
Topics in NLP

1. Overview
2. N-gram Models
3. Neural Language Models
4. High-dimensional Outputs
5. Combining Neural LMs with n-grams
6. Neural Machine Translation
7. Attention Models
8. Historical Perspective

Use of probability in NLP

- Some tasks involving probability

1. Predicting the next word



The screenshot shows the GPT-2 Explorer interface. At the top, it says "ALLEN INSTITUTE for ARTIFICIAL INTELLIGENCE" and "GPT-2 Explorer". Below that, it states: "This demonstration uses the public 345M 117M parameter OpenAI GPT-2 language model to generate sentences." It then instructs: "Enter some initial text and the model will generate the most likely next words. You can click on one of those words to choose it and continue or just keep typing. Click the left arrow at the bottom to undo your last choice." The interface has two main sections: "Sentence:" and "Options:". Under "Sentence:", there is a text input field containing "The cat sat on the". Under "Options:", there is a list of words with their corresponding probabilities: 8.4% floor, 7.3% bed, 5.6% couch, 4.2% ground, 3.4% table, 3.1% sofa, 2.4% edge, 1.8% chair, 1.2% desk, and 1.2% window. At the bottom of the options list is an "Undo" button.

ALLEN INSTITUTE
for ARTIFICIAL INTELLIGENCE

GPT-2 Explorer

This demonstration uses the public 345M 117M parameter OpenAI GPT-2 language model to generate sentences.

Enter some initial text and the model will generate the most likely next words. You can click on one of those words to choose it and continue or just keep typing. Click the left arrow at the bottom to undo your last choice.

Sentence:

The cat sat on the

Options:

- 8.4% floor
- 7.3% bed
- 5.6% couch
- 4.2% ground
- 3.4% table
- 3.1% sofa
- 2.4% edge
- 1.8% chair
- 1.2% desk
- 1.2% window

← Undo

Built at the Allen Institute for Artificial Intelligence using Hugging Face's [pytorch-pretrained-BERT](#) library. See the [source code on GitHub](#).

2. Which is more probable?

all of a sudden I notice three guys standing on the sidewalk

Same set of words in a different order is nonsensical:

on guys all I of notice sidewalk three a sudden standing the

Probability essential in tasks with ambiguous input:

- Speech recognition
- Spelling correction, grammatical error correction
- Machine translation

Computing the probability of a word

- Consider task of computing $P(w | h)$
 - the probability of a word w given some history h
 - Suppose the history h is
“its water is so transparent that”
 - We want the probability that the next word is the:
 $P(\text{the} | \text{its water is so transparent that})$
 - Estimate probability from frequency counts (C)
 $P(\text{the} | \text{its water is so transparent that}) =$
 $C(\text{its water is so transparent that the}) / C(\text{its water is so transparent that})$
- Even the web is not big enough to estimate probability from frequency counts

A better method for probabilities

- Chain rule of probability

$$P(X_1 \dots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1^2) \dots P(X_n|X_1^{n-1}) = \prod_{k=1}^n P(X_k|X_1^{k-1})$$

- Applying it to words

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1}) = \prod_{k=1}^n P(w_k|w_1^{k-1})$$

- shows link between computing joint probability of a sequence and computing the conditional probability of a word given previous words.
- The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words

Bigram Probabilities

- Bigram

- instead of computing the probability

- $P(\text{the}|\text{Walden Pond's water is so transparent that})$

- we approximate it with the probability

- $P(\text{the}|\text{that})$

- We are making the assumption that

- $P(w_n | w^{n-1}_1) = P(w_n | w_{n-1})$

N-Gram Models

- An n -gram is a sequence of tokens, e.g., words
 - n -gram models define the conditional probability of the n^{th} token given the previous $n-1$ tokens
- Products of conditional distributions define probability distributions of longer sequences

$$P(x_1, \dots, x_\tau) = P(x_1, \dots, x_{n-1}) \prod_{t=n}^{\tau} P(x_t \mid x_{t-n+1}, \dots, x_{t-1})$$

Sequence of
length n

- Comes from chain rule of probability
 - $P(x_1, \dots, x_n) = P(x_n \mid x_1, \dots, x_{n-1}) P(x_1, \dots, x_{n-1})$
- Distribution of $P(x_1, \dots, x_{n-1})$ may be defined by a different model with a smaller value of n

Common n -grams

- Count how many times each possible n -gram occurs in the training set
 - Models based on n -grams have been core building block of NLP
- For small values of n , we have
 - $n=1$: unigram $P(x_1)$
 - $n=2$: bigram $P(x_1, x_2)$
 - $n=3$: trigram $P(x_1, x_2, x_3)$

Examples of n-grams

1. Frequencies of 4 and 3-grams

4-gram	FirstTerms	LastTerm	Frequency
...
	Your house is	great	19
	Your house looks	dirty	3
	Your house looks	lovely	13
	Your house looks	nice	21
	Your house looks	peaceful	4
	Your house seem	awful	4
...
3-gram	FirstTerms	LastTerm	Frequency
...
	house looks	clean	20
	house looks	dark	6
	house looks	dirty	13
	house looks	haunting	2
	house looks	lovely	14
	house looks	nice	28
	house looks	peaceful	9
	house looks	unclean	3
...

2. Protein Sequencing

3. DNA Sequencing

4. Computational linguistics (character)

5. Computational Linguistics (word)

Field	Unit	Sample sequence	1-gram sequence	2-gram sequence	3-gram sequence
Vernacular name			unigram	bigram	trigram
Order of resulting Markov model			0	1	2
Protein sequencing	amino acid	... Cys-Gly-Leu-Ser-Trp, Cys, Gly, Leu, Ser, Trp,, Cys-Gly, Gly-Leu, Leu-Ser, Ser-Trp,, Cys-Gly-Leu, Gly-Leu-Ser, Leu-Ser-Trp, ...
DNA sequencing	base pair	...AGCTTCGA...	..., A, G, C, T, T, C, G, A,, AG, GC, CT, TT, TC, CG, GA,, AGC, GCT, CTT, TTC, TCG, CGA, ...
Computational linguistics	character	...to_be_or_not_to_be...	..., t, o, _b, e, _o, r, _n, o, t, _t, o, _b, e,, to, o_b, be, e_o, or, r_n, no, ot, t_t, to, o_b, be,, to_o_b, be_e_o, or_r_n, no_ot, t_t, to_o_b, be...
Computational linguistics	word	... to be or not to be, to, be, or, not, to, be,, to be, be or, or not, not to, to be,, to be or, be or not, or not to, not to be, ...

Unigram and Bigram Probabilities

Unigram Counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram Probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

$$\begin{aligned}
 &P(<s> \text{ i want english food } </s>) \\
 &= P(i|<s>)P(\text{want}|i)P(\text{english}|\text{want}) \\
 &\quad P(\text{food}|\text{english})P(</s>|\text{food}) \\
 &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
 &= .000031
 \end{aligned}$$

Training n -Gram Models

- Usually train both an n -gram model and an $n-1$ gram model making it easy to compute

$$P(x_t | x_{t-n+1}, \dots, x_{t-1}) = \frac{P_n(x_{t-n+1}, \dots, x_t)}{P_{n-1}(x_{t-n+1}, \dots, x_{t-1})}$$

n -gram probability

$n-1$ gram probability

- Simply by looking up two stored probabilities
- For this to exactly reproduce inference in P_n we must omit the final character from each sequence when we train P_{n-1}

Example of Trigram Model Computation

- How to compute probability of “THE DOG RAN AWAY”
 - The first words of the sentence cannot be handled by the default formula on conditional probability because there is no context at the beginning of the sentence
 - Instead we must use the marginal probability over words at the start of the sentence. We thus evaluate $P_3(\text{THE DOG RAN})$
 - The last word may be predicted using the typical case, of using conditional distribution $P(\text{AWAY} | \text{DOG RAN})$
 - Putting this together

$$P(\text{THE DOG RAN AWAY}) = \frac{P_3(\text{THE DOG RAN})P_3(\text{DOG RAN AWAY})}{P_2(\text{DOG RAN})}$$

Limitation of Maximum Likelihood for n -gram models

- P_n estimated from training samples is very likely to be zero in many cases even though the tuple x_{t-n+1}, \dots, x_t may appear in test set
 - When P_{n-1} is zero the ratio is undefined
 - When P_{n-1} is non-zero but P_n is zero the log-likelihood is $-\infty$
- To avoid such catastrophic outcomes, n -gram models employ smoothing
 - Shift probability mass of observed tuples to unobserved similar ones

Smoothing techniques

1. Add non-zero mass to next symbol values

- Justified as Bayesian inference with a uniform or Dirichlet prior over count parameters

2. Mixture of higher/lower-order n -gram models

- with higher-order models providing more capacity and lower-order models more likely to avoid counts of zero

3. Back-off methods look-up lower-order n -grams if frequency of context $x_{t-1}, \dots, x_{t-n+1}$ is too small to use higher-order model

- More formally, they estimate distribution over x_t by using contexts $x_{t-n+k}, \dots, x_{t-1}$, for increasing k , until a sufficiently reliable estimate is found

N -gram with Backoff

- For high order models, e.g, $N=5$, only a small fraction of N -grams appear in training corpus
 - a problem of data sparsity
 - with 0 probability for almost all sentences
- To counteract this, several back-off techniques have been suggested, the most popular being:

$$P_N^{NG}(w_t) = \begin{cases} p_{w_t, \mathbf{c}_{N-1}} & \text{if not zero} \\ P_{N-1}^{NG}(w_t) \alpha_{\mathbf{c}_{N-1}} & \text{otherwise} \end{cases}$$

- where α and p are called back-off coefficients and discounted probabilities, respectively

Shortcomings of n -gram models

- Vulnerable to curse of dimensionality
- There are $|V|^n$ possible n -grams and $|V|$ is large
- Even with a massive training set most n -grams will not occur
- One way to view a classical n -gram model is that it is performing nearest-neighbor lookup
 - In other words, it can be viewed as a local non-parametric predictor, similar to k -nearest neighbors
 - Any two words are at same distance from each other

Class-based language models

- To improve statistical efficiency of n -gram models
 - Introduce notion of word categories
 - Share statistics of words in same categories
- Idea: use a clustering algorithm to partition words into clusters based on their co-occurrence frequencies with other words
 - Model can then use word-class IDs rather than individual word-IDs to represent context
- Still much information is lost in this process