
Natural Language Processing: **N-Gram Language Models**

Sharun Akter Khushbu

Daffodil International University

Language Models

- Formal grammars (e.g. regular, context free) give a hard “binary” model of the legal sentences in a language.
- For NLP, a *probabilistic* model of a language that gives a probability that a string is a member of a language is more useful.
- To specify a correct probability distribution, the probability of all sentences in a language must sum to 1.

Uses of Language Models

- Speech recognition
 - “I ate a cherry” is a more likely sentence than “Eye eight uh Jerry”
- OCR & Handwriting recognition
 - More probable sentences are more likely correct readings.
- Machine translation
 - More likely sentences are probably better translations.
- Generation
 - More likely sentences are probably better NL generations.
- Context sensitive spelling correction
 - “Their are problems wit this sentence.”

Completion Prediction

- A language model also supports predicting the completion of a sentence.
 - Please turn off your cell _____
 - Your program does not _____
- *Predictive text input* systems can guess what you are typing and give choices on how to complete it.

N-Gram Models

- Estimate probability of each word given prior context.
 - $P(\text{phone} \mid \text{Please turn off your cell})$
- Number of parameters required grows exponentially with the number of words of prior context.
- An N-gram model uses only $N-1$ words of prior context.
 - Unigram: $P(\text{phone})$
 - Bigram: $P(\text{phone} \mid \text{cell})$
 - Trigram: $P(\text{phone} \mid \text{your cell})$
- The ***Markov assumption*** is the presumption that the future behavior of a dynamical system only depends on its recent history. In particular, in a ***kth-order Markov model***, the next state only depends on the k most recent states, therefore an N-gram model is a $(N-1)$ -order Markov model.

N-Gram Model Formulas

- Word sequences

$$w_1^n = w_1 \dots w_n$$

- Chain rule of probability

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

- Bigram approximation

$$P(w_1^n) = \prod_{k=1}^n P(w_k | w_{k-1})$$

- N-gram approximation

$$P(w_1^n) = \prod_{k=1}^n P(w_k | w_{k-N+1}^{k-1})$$

Estimating Probabilities

- N-gram conditional probabilities can be estimated from raw text based on the *relative frequency* of word sequences.

$$\textbf{Bigram:} \quad P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

$$\textbf{N-gram:} \quad P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

- To have a consistent probabilistic model, append a unique start (<s>) and end (</s>) symbol to every sentence and treat these as additional words.

Generative Model & MLE

- An N-gram model can be seen as a probabilistic automata for generating sentences.

Initialize sentence with N-1 <s> symbols

Until </s> is generated do:

Stochastically pick the next word based on the conditional probability of each word given the previous N - 1 words.

- Relative frequency estimates can be proven to be *maximum likelihood estimates* (MLE) since they maximize the probability that the model M will generate the training corpus T .

$$\hat{\lambda} = \operatorname{argmax}_{\lambda} P(T \mid M(\lambda))$$

Example from Textbook

- $P(<s> \text{ i want english food } </s>)$
 $= P(i \mid <s>) P(\text{want} \mid i) P(\text{english} \mid \text{want})$
 $\quad P(\text{food} \mid \text{english}) P(</s> \mid \text{food})$
 $= .25 \times .33 \times .0011 \times .5 \times .68 = .000031$
- $P(<s> \text{ i want chinese food } </s>)$
 $= P(i \mid <s>) P(\text{want} \mid i) P(\text{chinese} \mid \text{want})$
 $\quad P(\text{food} \mid \text{chinese}) P(</s> \mid \text{food})$
 $= .25 \times .33 \times .0065 \times .52 \times .68 = .00019$

Train and Test Corpora

- A language model must be trained on a large corpus of text to estimate good parameter values.
- Model can be evaluated based on its ability to predict a high probability for a disjoint (held-out) test corpus (testing on the training corpus would give an optimistically biased estimate).
- Ideally, the training (and test) corpus should be representative of the actual application data.
- May need to *adapt* a general model to a small amount of new (*in-domain*) data by adding highly weighted small corpus to original training data.

Unknown Words

- How to handle words in the test corpus that did not occur in the training data, i.e. *out of vocabulary* (OOV) words?
- Train a model that includes an explicit symbol for an unknown word (<UNK>).
 - Choose a vocabulary in advance and replace other words in the training corpus with <UNK>.
 - Replace the first occurrence of each word in the training data with <UNK>.

Evaluation of Language Models

- Ideally, evaluate use of model in end application (*extrinsic, in vivo*)
 - Realistic
 - Expensive
- Evaluate on ability to model test corpus (*intrinsic*).
 - Less realistic
 - Cheaper
- Verify at least once that intrinsic evaluation correlates with an extrinsic one.

Perplexity

- Measure of how well a model “fits” the test data.
- Uses the probability that the model assigns to the test corpus.
- Normalizes for the number of words in the test corpus and takes the inverse.

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

- Measures the weighted average branching factor in predicting the next word (lower is better).

Sample Perplexity Evaluation

- Models trained on 38 million words from the Wall Street Journal (WSJ) using a 19,979 word vocabulary.
- Evaluate on a disjoint set of 1.5 million WSJ words.

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Smoothing

- Since there are a combinatorial number of possible word sequences, many rare (but not impossible) combinations never occur in training, so MLE incorrectly assigns zero to many parameters (a.k.a. *sparse data*).
- If a new combination occurs during testing, it is given a probability of zero and the entire sequence gets a probability of zero (i.e. infinite perplexity).
- In practice, parameters are *smoothed* (a.k.a. *regularized*) to reassign some probability mass to unseen events.
 - Adding probability mass to unseen events requires removing it from seen ones (*discounting*) in order to maintain a joint distribution that sums to 1.

Laplace (Add-One) Smoothing

- “Hallucinate” additional training data in which each possible N-gram occurs exactly once and adjust estimates accordingly.

$$\textbf{Bigram:} \quad P(w_n | w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

$$\textbf{N-gram:} \quad P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n) + 1}{C(w_{n-N+1}^{n-1}) + V}$$

where V is the total number of possible $(N-1)$ -grams (i.e. the vocabulary size for a bigram model).

- Tends to reassign too much mass to unseen events, so can be adjusted to add $0 < \delta < 1$ (normalized by δV instead of V).

Advanced Smoothing

- Many advanced techniques have been developed to improve smoothing for language models.
 - Good-Turing
 - Interpolation
 - Backoff
 - Kneser-Ney
 - Class-based (cluster) N-grams

Model Combination

- As N increases, the power (expressiveness) of an N -gram model increases, *but* the ability to estimate accurate parameters from sparse data decreases (i.e. the smoothing problem gets worse).
- A general approach is to combine the results of multiple N -gram models of increasing complexity (i.e. increasing N).

Interpolation

- Linearly combine estimates of N-gram models of increasing order.

Interpolated Trigram Model:

$$\hat{P}(w_n | w_{n-2}, w_{n-1}) = \lambda_1 P(w_n | w_{n-2}, w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$$

$$\text{Where: } \sum_i \lambda_i = 1$$

- Learn proper values for λ_i by training to (approximately) maximize the likelihood of an independent *development* (a.k.a. *tuning*) corpus.

Backoff

- Only use lower-order model when data for higher-order model is unavailable (i.e. count is zero).
- Recursively back-off to weaker models until data is available.

$$P_{katz}(w_n | w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n | w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^n) > 1 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n | w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

Where P^* is a discounted probability estimate to reserve mass for unseen events and α 's are back-off weights (see text for details).

A Problem for N-Grams: Long Distance Dependencies

- Many times local context does not provide the most useful predictive clues, which instead are provided by *long-distance dependencies*.
 - Syntactic dependencies
 - “The *man* next to the large oak tree near the grocery store on the corner *is* tall.”
 - “The *men* next to the large oak tree near the grocery store on the corner *are* tall.”
 - Semantic dependencies
 - “The *bird* next to the large oak tree near the grocery store on the corner *flies* rapidly.”
 - “The *man* next to the large oak tree near the grocery store on the corner *talks* rapidly.”
- More complex models of language are needed to handle such dependencies.

Summary

- Language models assign a probability that a sentence is a legal string in a language.
- They are useful as a component of many NLP systems, such as ASR, OCR, and MT.
- Simple N-gram models are easy to train on unsupervised corpora and can provide useful estimates of sentence likelihood.
- MLE gives inaccurate parameters for models trained on sparse data.
- Smoothing techniques adjust parameter estimates to account for unseen (but not impossible) events.