Language Modeling 2

CS287

Review: Softmax Issues

Use a softmax to force a distribution,

$$\text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{c \in \mathcal{C}} \exp(z_c)}$$

$$\log \text{softmax}(\mathbf{z}) = \mathbf{z} - \log \sum_{c \in \mathcal{C}} \exp(z_c)$$

- **Issue:** class $\mathcal{C}$ is huge.
- For C&W, 100,000, for word2vec 1,000,000 types
- Note largest dataset is 6 billion words

Quiz

We have trained a depth-two balanced soft-max tree. Give an algorithm and time-complexity for:

- Computing the optimal class decision?
- Greedily approximating the optimal class decision.
- Appropriately sampling a word.
- Computing the full distribution over classes?

Answers I

- Computing the optimal class decision i.e. $\arg\max_y p(\mathbf{y} = y|\mathbf{x})$?
  - Requires full enumeration $O(|\mathcal{V}|)$

$$\arg\max_y p(\mathbf{y} = y|\mathbf{x}) = \arg\max_y p(\mathbf{y}|C, \mathbf{x})p(C|\mathbf{x})$$

- Greedily approximating the optimal class decision.
  - Walk tree $O(\sqrt{|\mathcal{V}|})$

$$\arg\max_y p(\mathbf{y} = y|\mathbf{x})$$

$$c^* = \arg\max_y p(C|\mathbf{x})$$

$$y^* = \arg\max_y p(\mathbf{y}|C = c^*, \mathbf{x})$$

Answers II

- Appropriately sampling a word $y \sim p(\mathbf{y} = \delta(c)|\mathbf{x})$?
  - Walk tree $O(\sqrt{|\mathcal{V}|})$

$$y \sim p(\mathbf{y} = \delta(c)|\mathbf{x})$$

$$\hat{c} \sim p(C|\mathbf{x})$$

$$\hat{y} \sim p(\mathbf{y}|C = \hat{c}, \mathbf{x})$$

- Computing the full distribution over classes?
  - Enumerate $O(|\mathcal{V}|)$

$$p(\mathbf{y} = \delta(c)|\mathbf{x})p(\mathbf{y}|C = \hat{c}, \mathbf{x})$$

Contents

Language Modeling Task

Given a sequence of text give a probability distribution over the next word.

The Shannon game. Estimate the probability of the next letter/word given the previous.

> *THE ROOM WAS NOT VERY LIGHT A SMALL OBLONG READING LAMP ON THE DESK SHED GLOW ON POLISHED* ___

Language Modeling

Shannon (1948) *Mathematical Model of Communication*

> *We may consider a discrete source, therefore, to be represented by a stochastic process. Conversely, any stochastic process which produces a discrete sequence of symbols chosen from a finite set may be considered a discrete source. This will include such cases as:*
> *1. Natural written languages such as English, German, Chinese. ...*

Shannon's Babblers I

4. Third-order approximation (trigram structure as in English). IN NO 1ST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE

5. First-Order Word Approximation. Rather than continue with tetragram, ... , II-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies. REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

Shannon's Babblers II

6. Second-Order Word Approximation. The word transition probabilities are correct but no further structure is included.

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH 'RITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED

The resemblance to ordinary English text increases quite noticeably at each of the above steps.

Smoothness Image/Language

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts. -Sherlock Holmes, A Scandal in Bohemia

Repair Image / Language (Chatterjee et al., 2009)

It is a capital mistake to theorize before one has _____ . . .

Repair Image / Language (Chatterjee et al., 2009)

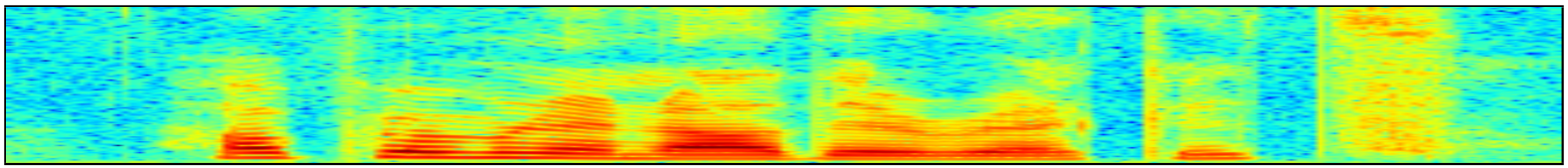
108 938 285 28 184 29 593 219 58 772 _____ . . .

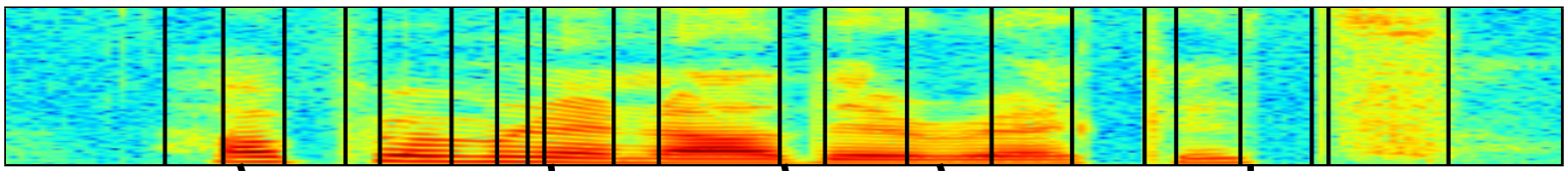
Language Modeling

Crucially important for:

- Speech Recognition
- Machine Translation
- Many deep learning applications
  - Captioning
  - Dialogue
  - Summarization
  - ...

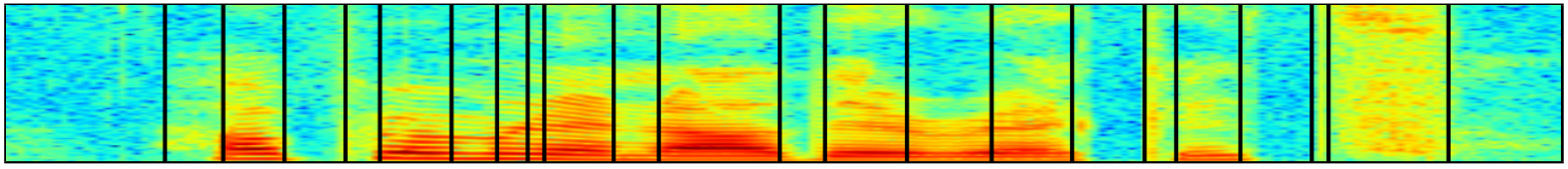
Statistical Model of Speech

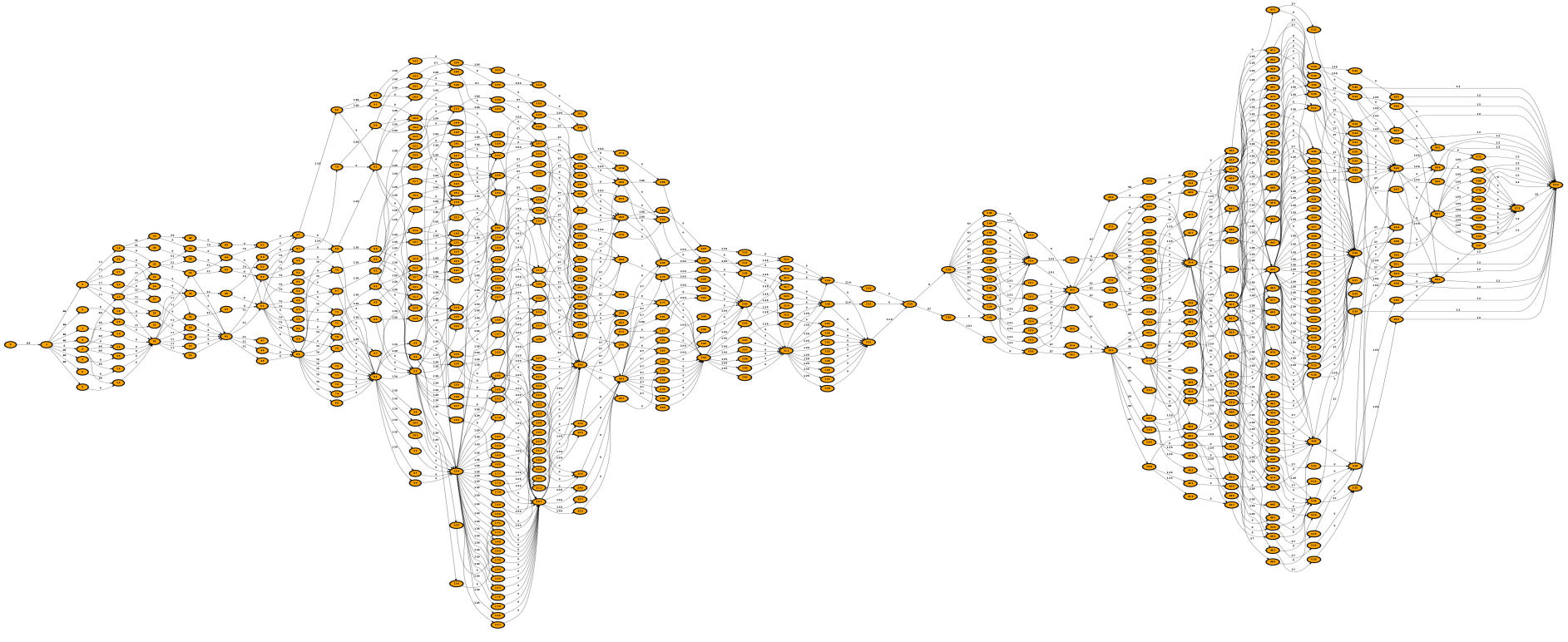




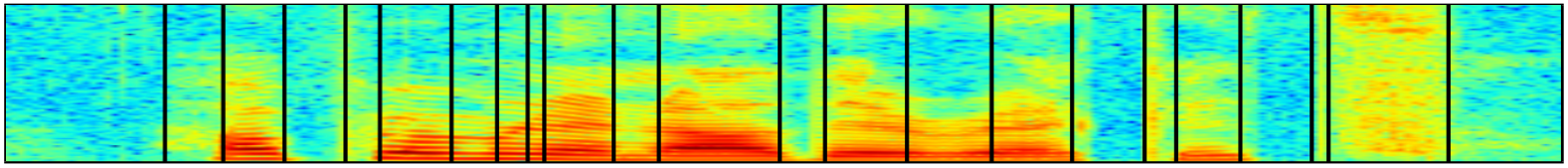
How permanent are their records?

Statistical Model of Speech

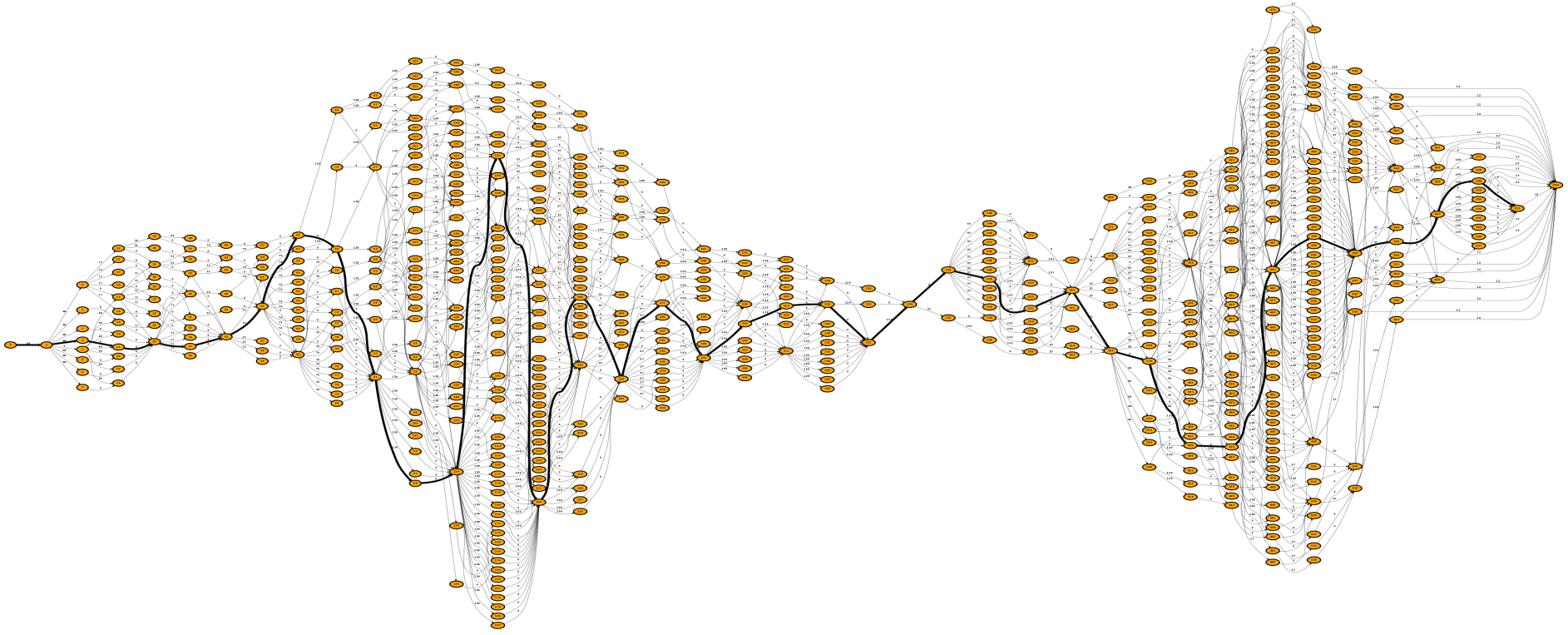




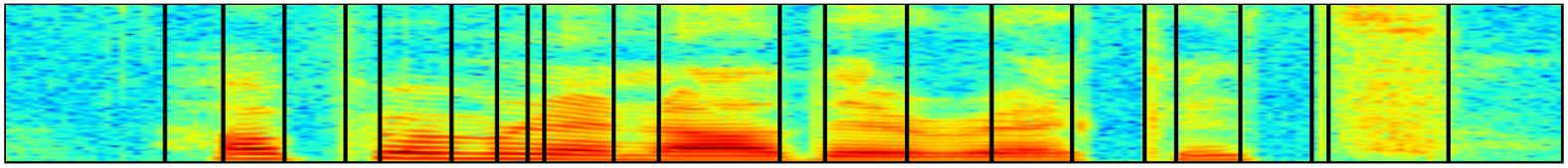
Statistical Model of Speech



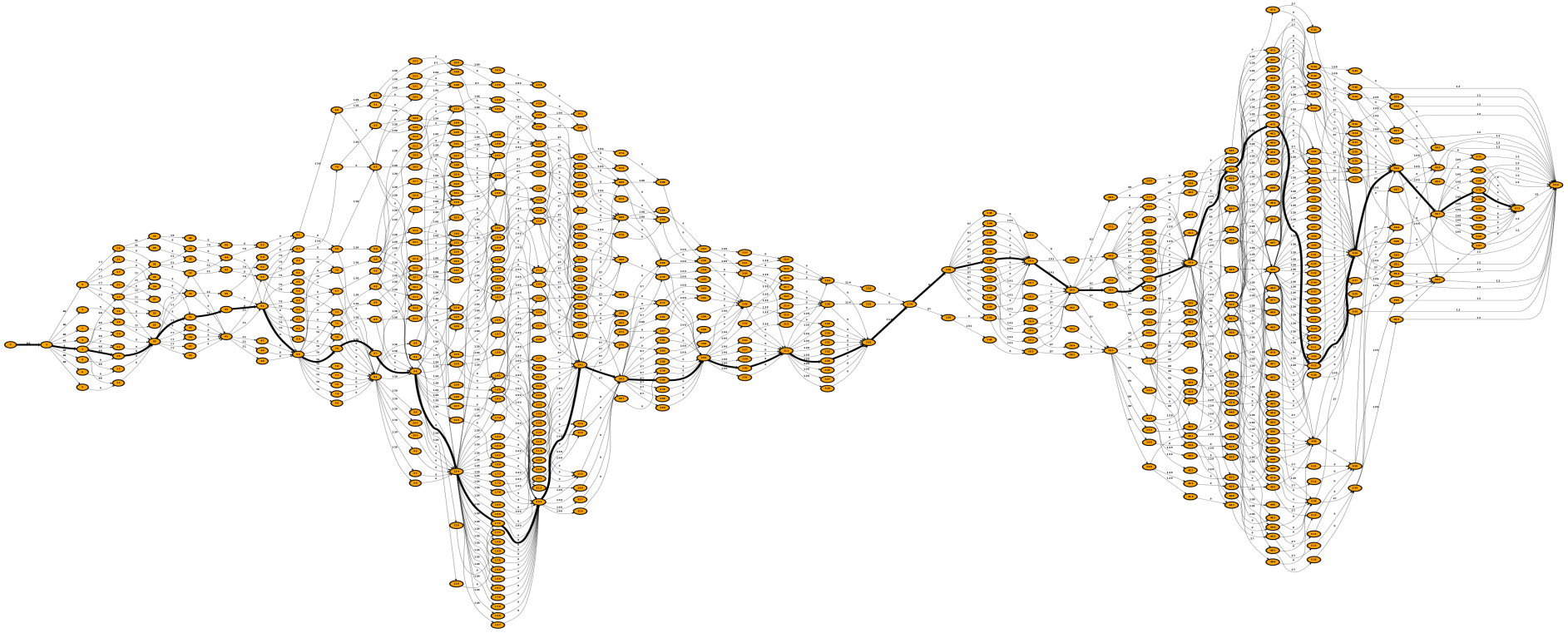
Transcription: Help peppermint on their records



Statistical Model of Speech



Transcription: How permanent are their records



Language Modeling Formally

Goal: Compute the probability of a sentence,

- Factorization:

$$p(w_1, \ldots, w_n) = \prod_{t=1}^{n} p(w_t | w_1, \ldots, w_{t-1})$$

k Estimate the probability of the next word, conditioned on prefix,

$$p(w_t | w_1, \ldots, w_{t-1}; \theta)$$

Machine Learning Setup

Multi-class prediction problem,

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$$

- $\mathbf{y}_i$; the one-hot next word
- $\mathbf{x}_i$; representation of the prefix $(w_1, \ldots, w_{t-1})$

Challenges:

- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)

Machine Learning Setup

Multi-class prediction problem,

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$$

- $\mathbf{y}_i$; the one-hot next word
- $\mathbf{x}_i$; representation of the prefix $(w_1, \ldots, w_{t-1})$

Challenges:

- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)

Problem Metric

Previously, used *accuracy* as a metric.

Language modeling uses of version average negative log-likelihood

- For test data $\bar{w}_1, \ldots, \bar{w}_n$
- $$NLL = -\frac{1}{n}\sum_{i=1}^{n} \log p(w_i | w_1, \ldots, w_{i-1})$$

Actually report *perplexity*,

$$perp = \exp\left(-\frac{1}{n}\sum_{i=1}^{n} \log p(w_i | w_1, \ldots, w_{i-1})\right)$$

Requires modeling full distribution as opposed to argmax (hinge-loss)

Perplexity: Intuition

- Effective uniform distribution size.
- If words were uniform: $perp = |\mathcal{V}| = 10000$
- Using unigram : $perp \approx 400$
- $\log_2 perp$ gives average number of bits needed per word.

Contents

n-gram Models

In practice representation doesn't use all $(w_1, \ldots, w_{t-1})$

$$p(w_i | w_1, \ldots, w_{i-1}) \approx p(w_i | w_{i-n+1}, \ldots, w_{i-1}; \boldsymbol{\theta})$$

- We call this an n-gram model.
- $w_{i-n+1}, \ldots, w_{i-1}$; the context

Bigram Models

Back to count-based multinomial estimation,

- Feature set $\mathcal{F}$: previous words
- Input vector $\mathbf{x}$ is sparse
- Count matrix $\mathbf{F} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V}}$
- Counts come from training data:

$$F_{c,w} = \sum_i \mathbf{1}(w_{i-1} = c, w_i = w)$$

$$p_{ML}(w_i | w_{i-1}; \theta) = \frac{F_{c,w}}{F_{c,\bullet}}$$

Trigram Models

- Feature set $\mathcal{F}$: previous two words (conjunction)
- Input vector $\mathbf{x}$ is sparse
- Count matrix $\mathbf{F} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V}, \mathcal{V}}$

$$F_{c,w} = \sum_i \mathbf{1}(w_{i-2:i-1} = c, w_i = w)$$

$$p_{ML}(w_i | w_{i-2:i-1} = c; \theta) = \frac{F_{c,w}}{F_{c,\bullet}}$$

Notation

- $c = w_{i-n+1:i-1}$; context
- $c' = w_{i-n+2:i-1}$; context without first word
- $c'' = w_{i-n+3:i-1}$; context without first two words
- $[x]_+ = \max\{0, x\}$; positive part (ReLU)
- $F_{\bullet,w} = \sum_c F_{c,w}$; sum-over-dimension
- Maximum-likelihood,

$$p_{ML}(w|c) = F_{c,w} / F_{c,\bullet}$$

- Non-zero count words, for all c, w

$$N_{c,w} = \mathbf{1}(F_{c,w} > 0)$$

NGram Models

It is common to go up to 5-grams,

$$F_{c,w} = \sum_i \mathbf{1}(w_{i-5+1} \dots w_{i-1} = c, w_i = w)$$

Matrix becomes very sparse at 5-grams.

Google 1T

Number of token	1,024,908,267,229
Number of sentences	95,119,665,584
Size compressed (counts only)	24 GB
Number of unigrams	13,588,391
Number of bigrams	314,843,401
Number of trigrams	977,069,902
Number of fourgrams	1,313,818,354
Number of fivegrams	1,176,470,663

Contents

NGram Models

- Maximum likelihood models word terribly.
- NGram models are sparse, need to handle unseen cases.
- Presentation follows work of Chen and Goodman (1999)

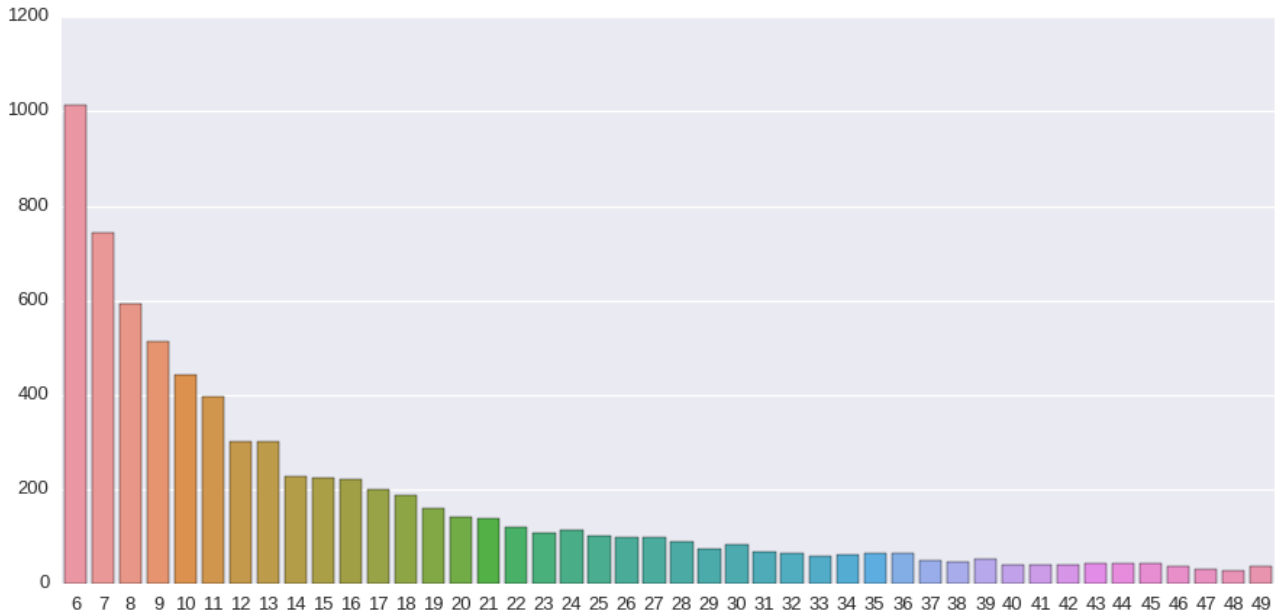
Count Modifications

- Laplace Smoothing

$$\bar{F}_{c,w} = F_{c,w} + \alpha \text{ for all } c, w$$

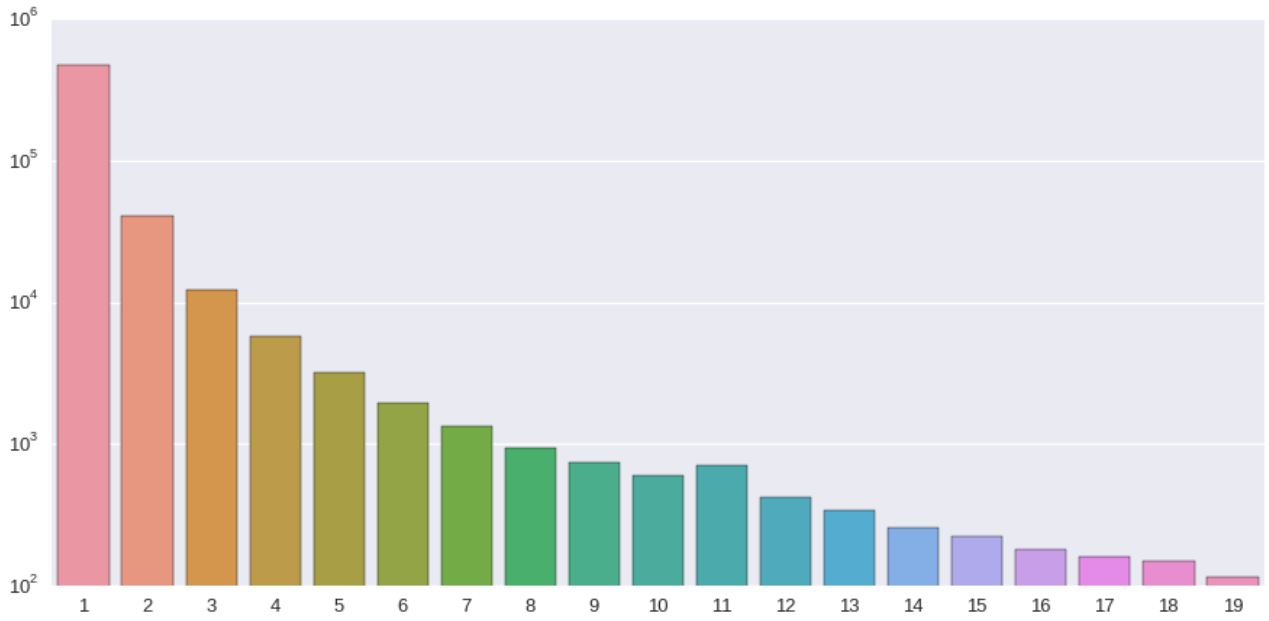
- Good-Turing Smoothing (Good, 1953)

$$\bar{F}_{c,w} = (F_{c,w} + 1)\frac{hist(F_{c,w} + 1)}{hist(F_{c,w})} \text{ for all } c, w$$



Real Issues N-Gram Sparsity

- Histogram of trigrams



Idea 1: Interpolation

For trigrams:

$$p_{interp}(w|c) = \lambda_1 p_{ML}(w|c) + \lambda_2 p_{ML}(w|c') + \lambda_3 p_{ML}(w|c'')$$

Ensure that λs form convex combination

$$\sum_i \lambda_i = 1$$

$$\lambda_i \geq 0 \text{ for all i}$$

Idea 1: Interpolation (Jelinek-Mercer Smoothing)

Can write recursively,

$$p_{interp}(w|c) = \lambda p_{ML}(w|c) + (1 - \lambda) p_{interp}(w|c')$$

Ensure that λ form convex combination

$$0 \leq \lambda \leq 1$$

How to set parameters: Validation Tuning

- Treat λ as hyper-parameters
- Can estimate λ using EM (or gradients directly)
- Alternatively: Grid-search to held-out data.
- Can add more parameters $\lambda(c, w)$
- Language models are often stored with prob and backoff value (λ)

How to set parameters: Witten-Bell

Define $\lambda(c, w)$ as a function of c, w

$$(1 - \lambda) = \frac{N_{c,\bullet}}{N_{c,\bullet} + F_{c,w}}$$

- $N_{c,\bullet}$; Number of unique word types seen with context

$$p_{wb}(w|c) = \frac{F_{c,w} + N_{c,\bullet} \times p_{wb}(w|c')}{F_{c,\bullet} + N_{c,\bullet}}$$

Interpolation counts are a new events estimated as proportional to seeing a new word.

Idea 2: Absolute Discounting

Similar form to interpolation

$$p_{abs}(w|c) = \lambda(w,c)p_{ML}(w|c) + \eta(c)p_{abs}(w|c')$$

Delete counts and redistribute:

$$p_{abs} = \frac{[F_{c,w} - D]_+}{F_{c,\bullet}} + \eta(c)p_{abs}(w|c')$$

Need for a given c

$$\sum_w \lambda(w,c)p_1(w) + \eta(c)p_2(w) = \sum_w \lambda(w)p_1(w) + \eta(c) = 1$$

In-class: To ensure a valid distribution, what are λ and η with $D = 1$?

Answer

$$\lambda(w) = \frac{[F_{c,w} - D]_+}{F_{c,w}}$$

$$\eta(w) = 1 - \sum_w \frac{[F_{c,w} - D]_+}{F_{c,\bullet}} = \sum_w \frac{F_{c,\bullet} - [F_{c,w} - D]_+}{F_{c,\bullet}}$$

Total counts minus counts discounted.

$$\eta(w) = \frac{D \times N_{c,\bullet}}{F_{c,\bullet}}$$

Lower-Order Smoothing Issue

Consider the example: `San Francisco` and `Los Franciso`

Would like:

- $p(w_{i-1} = \text{San}, w_i = \text{Francisco})$ to be high
- $p(w_{i-1} = \text{Los}, w_i = \text{Francisco})$ to be very low

However, interpolation alone doesn't ensure this. Why?

$$(1 - \lambda) p(w_i = \text{Francisco})$$

Could be quite high, from seeing `San Francisco` many times.

Lower-Order Smoothing Issue

Consider the example: `San Francisco` and `Los Franciso`

Would like:

- $p(w_{i-1} = \text{San}, w_i = \text{Francisco})$ to be high
- $p(w_{i-1} = \text{Los}, w_i = \text{Francisco})$ to be very low

However, interpolation alone doesn't ensure this. Why?

$$(1 - \lambda)p(w_i = \text{Francisco})$$

Could be quite high, from seeing `San Francisco` many times.

Kneser-Ney Smoothing

- Uses Absolute Discounting instead of ML
- However want distribution to match ML on lower-order terms.
- Ensure this by marginalizing out and matching lower-order distribution.
- Uses a different function for the lower-order term KN'.
- Most commonly used smoothing technique (details follow).

Review: Chain-Rule and Marginalization

Chain rule:

$$p(X, Y) = P(X|Y)P(Y)$$

Marginalization:

$$p(X, Y) = \sum_{z \in \mathcal{Z}} P(X, Y, Z = z)$$

Marginal matching constraint.

$$p_A(X, Y) \neq p_B(X, Y)$$

$$\sum_y p_A(X, Y = y) = p_B(X)$$

Kneser-Ney Smoothing

Main Idea: match ML marginals for all c'

$$\sum_{c_1} p_{KN}(c_1, c', w) = \sum_{c_1} p_{KN}(w|c) p_{ML}(c) = p_{ML}(c', w)$$

$$\sum_{c_1} p_{KN}(w|c) \frac{F_{c,\bullet}}{F_{\bullet,\bullet}} = \frac{F_{c'w,\bullet}}{F_{\bullet,\bullet}}$$

$$[\ c_1 \qquad\qquad c' \qquad\qquad w\ ]$$

Kneser-Ney Smoothing

$$p_{KN} = \frac{[F_{c,w} - D]_+}{F_{c,\bullet}} + \eta(c) p_{KN'}(w|c')$$

$$\begin{aligned}
F_{c'w,\bullet} &= \sum_{c_1} F_{c,\bullet}[p_{KN}(w|c)] \\
&= \sum_{c_1} F_{c,\bullet}[\frac{[F_{c,w} - D]_+}{F_{c,\bullet}} + \frac{D}{F_{c,\bullet}} N_{c,\bullet} \times p_{KN'}(w|c')] \\
&= \sum_{c_1 : N_{c,w} > 0} F_{c,\bullet} \frac{F_{c,w} - D}{F_{c,\bullet}} + \sum_{c_1} DN_{c,\bullet} \times p_{KN'}(w|c') \\
&= F_{c',w} - N_{\bullet c',w} D + DN_{\bullet c',\bullet} \times p_{KN'}(w|c')
\end{aligned}$$

Kneser-Ney Smoothing

Final equation for unigram

$$p_{KN'}(w) = N_{\bullet,w} / N_{\bullet,\bullet}$$

- Intuition: Prob of a unique bigrams ending in w.

$$p_{KN'}(w|c') = N_{\bullet c',w} / N_{\bullet c',\bullet}$$

- Intuition: Prob of a unique ngram (with middle c') ending in w.

Modified Kneser-Ney

Set D based on $F_{c,w}$

$$D = \begin{cases} 0 & F_{c,w} = 0 \\ D_1 & F_{c,w} = 1 \\ D_2 & F_{c,w} = 2 \\ D_{3+} & F_{c,w} \geq 3 \end{cases}$$

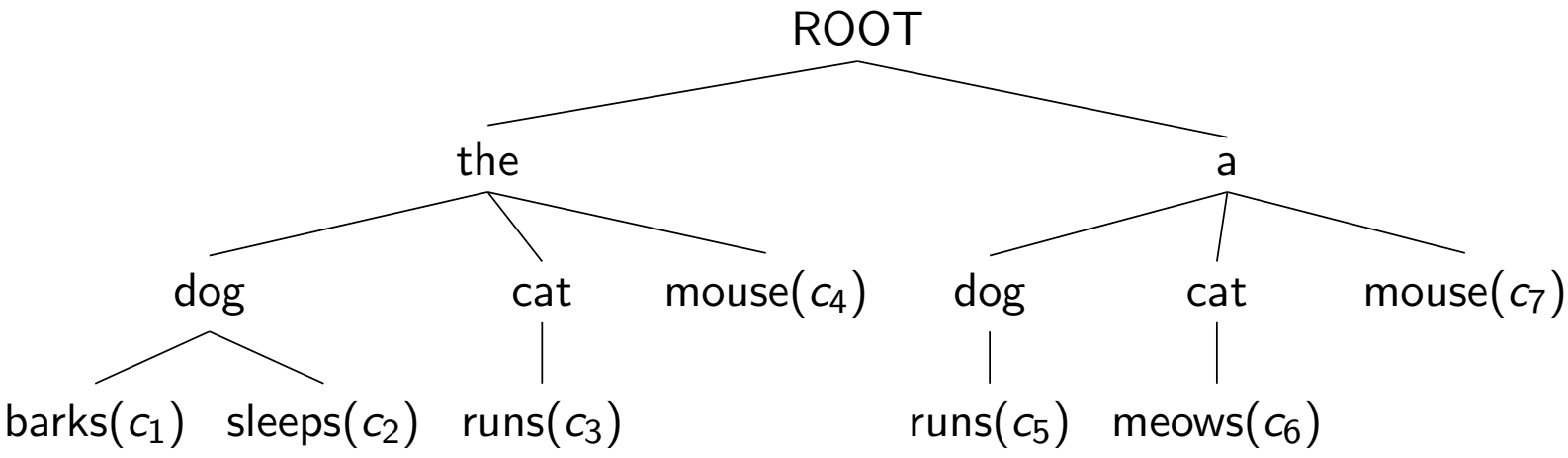
- Modifications to the formula are in the paper

Contents

Language Model Issues

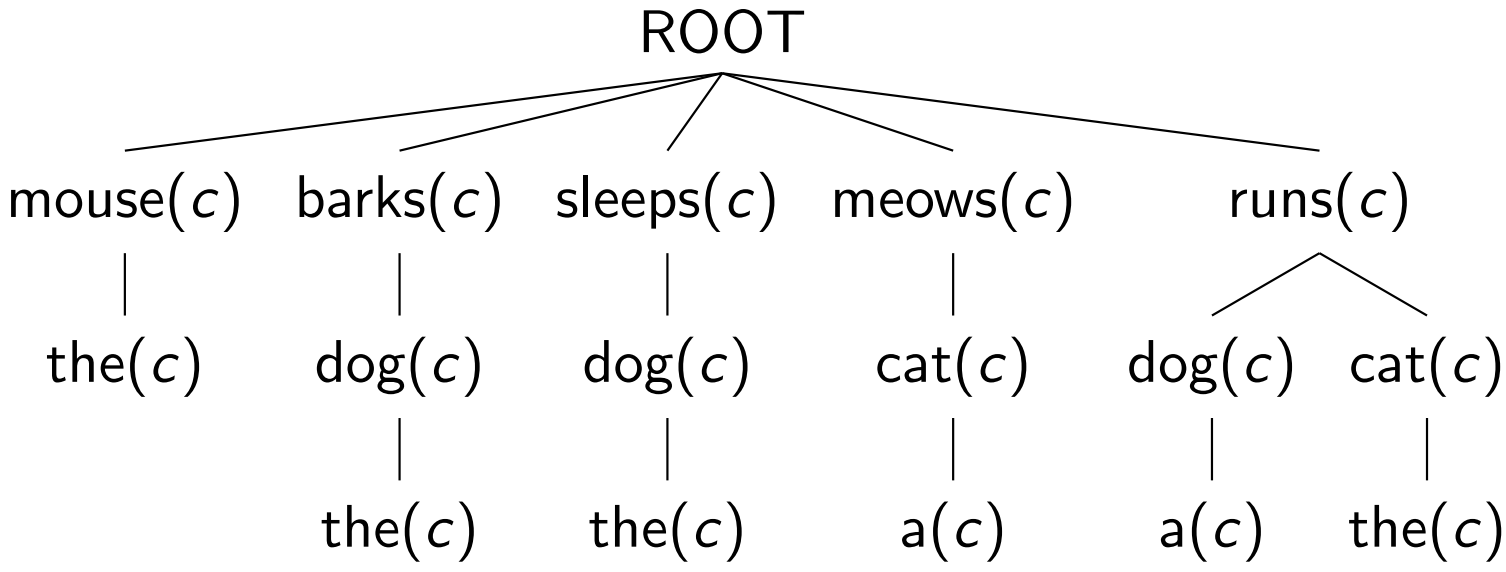
1. LMs become very sparse.
   - Obviously cannot use matrices ($|\mathcal{V}| \times |\mathcal{V}| \times |\mathcal{V}| > 10^{12}$)
2. LMs are very big
3. Lookup speed is crucial

Trie Data structure



Issues?

Reverse Trie Data structure



Used in several standard language modeling toolkits.

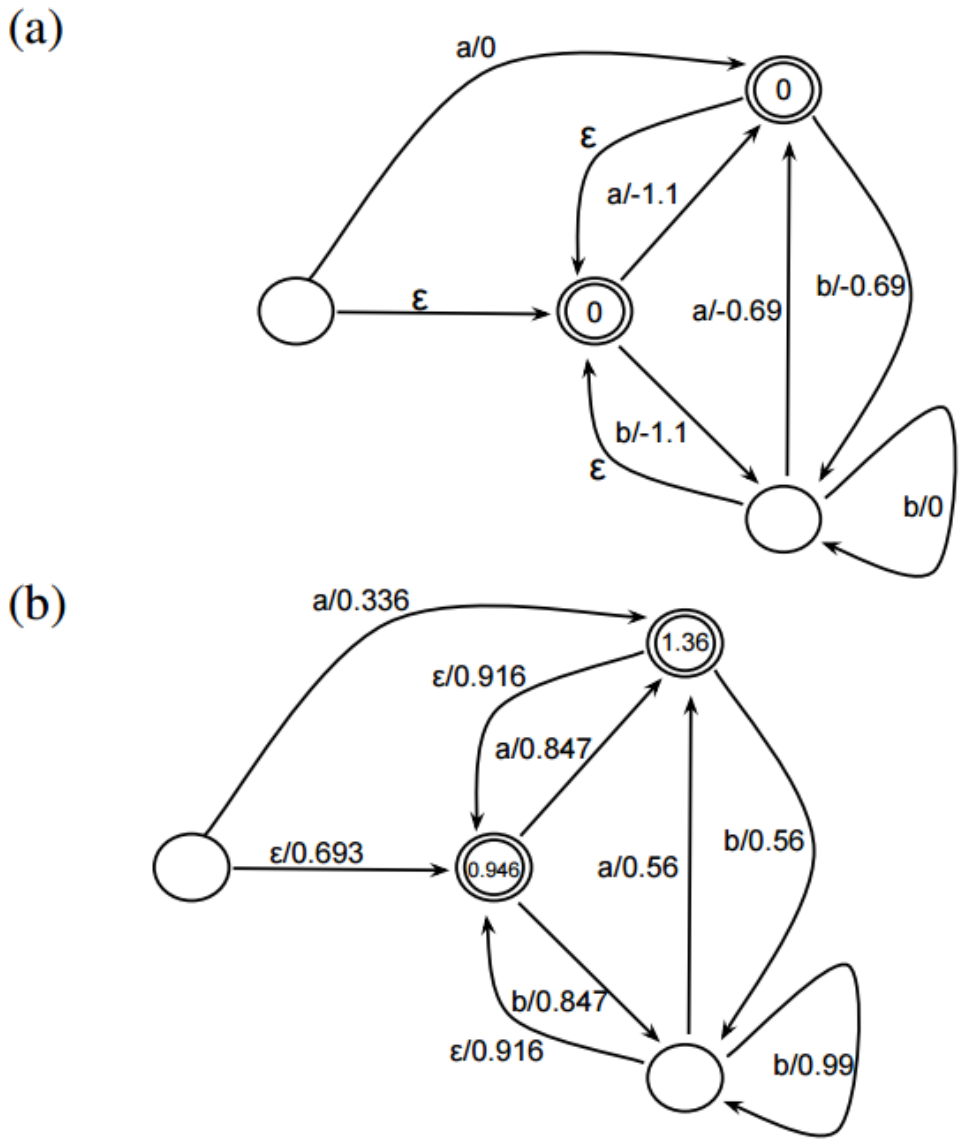
Efficiency: Hash Tables

- KenLM finds it more efficient to directly hash ngrams
- All ngrams of a given weight are kept in a hash table.
- Fast linear-probing hash tables make this work well.

Quantization

- Memory issues are often a concern
- Probabilities are kept in log space
- KenLM quantizes from 32 bits to smaller (others use 8 bits)
- Quatization is done by sorting, binning, and averaging.

Finite State Automata



Conclusion

Today,

- Language Modeling
- Various ideas in smoothing
- Tricks for efficient implementation.

Next time,

- Neural Network Language models