Language Models: Evaluation & Neural Models

CMSC 470
Marine Carpuat

Slides credit: Jurasky & Martin
Language Models
What you should know

• What is a language model
  • A probability model that assigns probabilities to sequences of words
  • Can be used to score or generate sequences

• N-gram language models
  • How they are defined, and what approximations are made in this definition (the Markov Assumption)
  • How they are estimated from data: count and normalize
  • But we need specific techniques to deal with zeros
    • word sequences unseen in training: add 1 smoothing, backoff
    • word types unseen in training: open vocabulary models with UNK token
Pros and cons of n-gram models

• Really easy to build, can train on billions and billions of words
• Smoothing helps generalize to new data
• Only work well for word prediction if the test corpus looks like the training corpus
• Only capture short distance context
Evaluating Language Models
Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?

- Extrinsic vs intrinsic evaluation
An intrinsic evaluation metric for language models: Perplexity

The best language model is one that best predicts an unseen test set
• Gives the highest $P(\text{sentence})$

**Perplexity** is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1w_2...w_N)^{-\frac{1}{N}}$$

Chain rule:

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1w_2...w_N)}}$$

For bigrams:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1...w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability
Interpreting perplexity as a branching factor

• Let’s suppose a sentence consisting of random digits
• What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

$$PP(W) = P(w_1w_2…w_N)^{-\frac{1}{N}}$$

$$= \left(\frac{1}{10}\right)^{-\frac{1}{N}}$$

$$= \frac{1}{10}^{-1}$$

$$= \frac{10}{10}$$

The Branching factor of a language is the number of possible next words that can follow any word.
We can think of perplexity as the weighted average branching factor of a language.
Lower perplexity = better model

• Comparing models on data from the Wall Street Journal
• Training: 38 million words, test: 1.5 million words

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>
The perils of overfitting

• N-grams only work well for word prediction if the test corpus looks like the training corpus

• In real life, it often doesn’t!

• We need to train robust models that generalize
  • Smoothing is important
  • Choose n carefully
A Neural Network-based Language Model
Toward a Neural Language Model

Figures by Philipp Koehn (JHU)
Representing Words

• “one hot vector”

\[
dog = [0, 0, 0, 0, 1, 0, 0, 0 \ldots] \\
cat = [0, 0, 0, 0, 0, 0, 1, 0 \ldots] \\
eat = [0, 1, 0, 0, 0, 0, 0, 0 \ldots]
\]

• That’s a large vector! practical solutions:
  • limit to most frequent words (e.g., top 20000)
  • cluster words into classes
  • break up rare words into subword units
Language Modeling with Feedforward Neural Networks

Map each word into a lower-dimensional real-valued space using shared weight matrix

Bengio et al. 2003
Example: Prediction with a Feedforward LM

Output layer $P(w|u)$, $1 \times |V|$,

Hidden layer $1 \times d_h$,

Projection layer $1 \times 3d$,

concatenated embeddings for context words

$P(w_t=V_{42}|w_{t-3}, w_{t-2}, w_{t-1})$
Example: Prediction with a Feedforward LM

Note: bias omitted in figure

\[ e = (E_{x1}, E_{x2}, \ldots, E_{x}) \]
\[ h = \sigma(We + b) \]
\[ z = Uh \]
\[ y = \text{softmax}(z) \]
Estimating Model Parameters

• Intuition: a model is good if it gives high probability to existing word sequences

• Training examples:
  • sequences of words in the language of interest

• Error/loss: negative log likelihood
  • At the corpus level \( \text{error}(\lambda) = -\sum_{E \text{ in corpus}} \log P_\lambda(E) \)
  • At the word level \( \text{error}(\lambda) = -\log P_\lambda(e_t|e_1 \ldots e_{t-1}) \)
This is the same loss as the one we saw earlier for Multiclass Logistic Regression

• Loss function for a single example

\[
L_{CE}(\hat{y}, y) = - \sum_{k=1}^{K} 1\{y = k\} \log p(y = k|x)
\]

1\{ \} is an indicator function that evaluates to 1 if the condition in the brackets is true, and to 0 otherwise.
Example: Parameter Estimation

Loss function at each position $t$

$$L = -\log p(w_t|w_{t-1},\ldots,w_{t-n+1})$$

Parameter update rule

$$\theta_{t+1} = \theta_t - \eta \frac{\partial}{\partial \theta} \log p(w_t|w_{t-1},\ldots,w_{t-n+1})$$
Word Embeddings: a useful by-product of neural LMs

- Words that occur in similar contexts tend to have similar embeddings
- Embeddings capture many usage regularities
- Useful features for many NLP tasks
Word Embeddings
Word Embeddings
Word Embeddings Capture Useful Regularities

**Morpho-Syntactic**
- Adjectives: base form vs. comparative
- Nouns: singular vs. plural
- Verbs: present tense vs. past tense

[Mikolov et al. 2013]

**Semantic**
- Word similarity/relatedness
- Semantic relations
- But tends to fail at distinguishing
  - Synonyms vs. antonyms
  - Multiple senses of a word
Language Modeling with Feedforward Neural Networks

Bengio et al. 2003
Count-based n-gram models vs. feedforward neural networks

• Pros of feedforward neural LM
  • Word embeddings capture generalizations across word types

• Cons of feedforward neural LM
  • Closed vocabulary
  • Training/testing is more computationally expensive

• Weaknesses of both types of model
  • Only work well for word prediction if the test corpus looks like the training corpus
  • Only capture short distance context
Language Models
What you should know

• What is a language model
• N-gram language models
• Evaluating language models with perplexity
• Feedforward neural language models
  • Use a neural network as a probabilistic classifier to compute probability of the next word given the previous n words
  • Trained like any neural network by backpropagation
  • Learn word embeddings in the process of language modeling
• Strengths and weaknesses of n-gram and neural language models