NLP Programming Tutorial 2 - Bigram Language Models

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Review: Calculating Sentence Probabilities

- We want the probability of
  
  \[ W = \text{speech recognition system} \]

- Represent this mathematically as:

  \[
  P(|W| = 3, w_1 = \text{"speech"}, w_2 = \text{"recognition"}, w_3 = \text{"system"}) = \\
  P(w_1 = \text{"speech"} \mid w_0 = \"<s>\") \\
  \quad \times P(w_2 = \text{"recognition"} \mid w_0 = \"<s>\", w_1 = \"speech\") \\
  \quad \times P(w_3 = \text{"system"} \mid w_0 = \"<s>\", w_1 = \"speech\", w_2 = \"recognition\") \\
  \quad \times P(w_4 = \"</s>\" \mid w_0 = \"<s>\", w_1 = \"speech\", w_2 = \"recognition\", w_3 = \"system\")
  \]

**NOTE:** sentence start <s> and end </s> symbol

**NOTE:**
\[ P(w_0 = \text{\"<s>\")} = 1 \]
Incremental Computation

• Previous equation can be written:

\[ P(W) = \prod_{i=1}^{\lvert W \rvert + 1} P(w_i \mid w_0 \ldots w_{i-1}) \]

• Unigram model ignored context:

\[ P(w_i \mid w_0 \ldots w_{i-1}) \approx P(w_i) \]
Unigram Models Ignore Word Order!

- Ignoring context, probabilities are the same:

\[
P_{\text{uni}}(w=\text{speech recognition system}) = P(w=\text{speech}) \times P(w=\text{recognition}) \times P(w=\text{system}) \times P(w=</s>)
\]

= 

\[
P_{\text{uni}}(w=\text{system recognition speech}) = P(w=\text{speech}) \times P(w=\text{recognition}) \times P(w=\text{system}) \times P(w=</s>)
\]
Unigram Models Ignore Agreement!

• Good sentences (words agree):

\[
P_{\text{uni}}(w=\text{i am}) = \ P(w=\text{i}) * P(w=\text{am}) * P(w=</s>)
\]

\[
P_{\text{uni}}(w=\text{we are}) = \ P(w=\text{we}) * P(w=\text{are}) * P(w=</s>)
\]

• Bad sentences (words don't agree)

\[
P_{\text{uni}}(w=\text{we am}) = \ P(w=\text{we}) * P(w=\text{am}) * P(w=</s>)
\]

\[
P_{\text{uni}}(w=\text{i are}) = \ P(w=\text{i}) * P(w=\text{are}) * P(w=</s>)
\]

But no penalty because probabilities are independent!
Solution: Add More Context!

- **Unigram** model ignored context:

  \[ P(w_i|w_0 \ldots w_{i-1}) \approx P(w_i) \]

- **Bigram** model adds one word of context

  \[ P(w_i|w_0 \ldots w_{i-1}) \approx P(w_i|w_{i-1}) \]

- **Trigram** model adds two words of context

  \[ P(w_i|w_0 \ldots w_{i-1}) \approx P(w_i|w_{i-2}w_{i-1}) \]

- Four-gram, five-gram, six-gram, etc...
Maximum Likelihood Estimation of n-gram Probabilities

- Calculate counts of n word and n-1 word strings

\[ P(w_i | w_{i-n+1} \ldots w_{i-1}) = \frac{c(w_{i-n+1} \ldots w_i)}{c(w_{i-n+1} \ldots w_{i-1})} \]

i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

n=2 →

P(osaka | in) = \frac{c(in osaka)}{c(in)} = \frac{1}{2} = 0.5
P(nara | in) = \frac{c(in nara)}{c(in)} = \frac{1}{2} = 0.5
Still Problems of Sparsity

- When n-gram frequency is 0, probability is 0

\[
P(\text{osaka} \mid \text{in}) = \frac{c(\text{i osaka})}{c(\text{in})} = 1 / 2 = 0.5
\]
\[
P(\text{nara} \mid \text{in}) = \frac{c(\text{i nara})}{c(\text{in})} = 1 / 2 = 0.5
\]
\[
P(\text{school} \mid \text{in}) = \frac{c(\text{in school})}{c(\text{in})} = 0 / 2 = 0!!
\]

- Like unigram model, we can use linear interpolation

**Bigram:**

\[
P(w_i \mid w_{i-1}) = \lambda_2 P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda_2) P(w_i)
\]

**Unigram:**

\[
P(w_i) = \lambda_1 P_{ML}(w_i) + (1 - \lambda_1) \frac{1}{N}
\]
Choosing Values of $\lambda$: Grid Search

- One method to choose $\lambda_2, \lambda_1$: try many values

$$
\begin{align*}
\lambda_2 &= 0.95, \lambda_1 = 0.95 \\
\lambda_2 &= 0.95, \lambda_1 = 0.90 \\
\lambda_2 &= 0.95, \lambda_1 = 0.85 \\
\lambda_2 &= 0.95, \lambda_1 = 0.05 \\
\lambda_2 &= 0.90, \lambda_1 = 0.95 \\
\lambda_2 &= 0.90, \lambda_1 = 0.90 \\
\lambda_2 &= 0.90, \lambda_1 = 0.05 \\
\lambda_2 &= 0.05, \lambda_1 = 0.10 \\
\lambda_2 &= 0.05, \lambda_1 = 0.05 \\
\end{align*}
$$

Problems:

- Too many options → Choosing takes time!
- Using same $\lambda$ for all n-grams → There is a smarter way!
## Context Dependent Smoothing

### High frequency word: “Tokyo”
- \( c(\text{Tokyo city}) = 40 \)
- \( c(\text{Tokyo is}) = 35 \)
- \( c(\text{Tokyo was}) = 24 \)
- \( c(\text{Tokyo tower}) = 15 \)
- \( c(\text{Tokyo port}) = 10 \)
  ...

Most 2-grams already exist
→ Large \( \lambda \) is better!

### Low frequency word: “Tottori”
- \( c(\text{Tottori is}) = 2 \)
- \( c(\text{Tottori city}) = 1 \)
- \( c(\text{Tottori was}) = 0 \)

Many 2-grams will be missing
→ Small \( \lambda \) is better!

- Make the interpolation depend on the context

\[
P(w_i | w_{i-1}) = \lambda_{w_{i-1}} P_{ML}(w_i | w_{i-1}) + (1 - \lambda_{w_{i-1}}) P(w_i)
\]
Witten-Bell Smoothing

- One of the many ways to choose $\lambda_{w_{i-1}}$

$$\lambda_{w_{i-1}} = 1 - \frac{u(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

$u(w_{i-1})$ = number of unique words after $w_{i-1}$

- For example:

<table>
<thead>
<tr>
<th>Word String</th>
<th>Count $c$</th>
<th>Unique Count $u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tottori is</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Tottori city</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Tottori</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Tokyo city</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>Tokyo is</td>
<td>35</td>
<td>30</td>
</tr>
<tr>
<td>Tokyo</td>
<td>270</td>
<td>30</td>
</tr>
</tbody>
</table>

$\lambda_{Tottori} = 1 - \frac{2}{2 + 3} = 0.6$

$\lambda_{Tokyo} = 1 - \frac{30}{30 + 270} = 0.9$
Programming Techniques
Inserting into Arrays

• To calculate n-grams easily, you may want to:

```python
my_words = ["this", "is", "a", "pen"]
my_words = ["<s>", "this", "is", "a", "pen", "</s>"]
```

• This can be done with:

```python
my_words.append("</s>")  # Add to the end
my_words.insert(0, "<s>")  # Add to the beginning
```
Removing from Arrays

- Given an n-gram with $w_{i-n+1} \ldots w_i$, we may want the context $w_{i-n+1} \ldots w_{i-1}$.
- This can be done with:

```python
my_ngram = "tokyo tower"
my_words = my_ngram.split(" ")  # Change into ["tokyo", "tower"]
my_words.pop()  # Remove the last element ("tower")
my_context = " " + .join(my_words)  # Join the array back together
print my_context
```
Exercise
Exercise

• Write two programs
  • train-bigram: Creates a bigram model
  • test-bigram: Reads a bigram model and calculates entropy on the test set
• Test train-bigram on test/02-train-input.txt
• Train the model on data/wiki-en-train.word
• Calculate entropy on data/wiki-en-test.word (if linear interpolation, test different values of $\lambda^2$)

• Challenge:
  • Use Witten-Bell smoothing (Linear interpolation is easier)
  • Create a program that works with any n (not just bi-gram)
train-bigram (Linear Interpolation)

create map counts, context_counts

for each line in the training_file
    split line into an array of words
    append “</s>” to the end and “<s>” to the beginning of words
for each i in 1 to length(words)-1  # Note: starting at 1, after <s>
    counts[“w_{i-1} w_i”] += 1  # Add bigram and bigram context
    context_counts[“w_{i-1}”] += 1
    counts[“w_i”] += 1  # Add unigram and unigram context
    context_counts[“”] += 1

open the model_file for writing
for each ngram, count in counts
    split ngram into an array of words  # “w_{i-1} w_i” → {“w_{i-1}”, “w_i”}
    remove the last element of words  # {“w_{i-1}”, “w_i”} → {“w_{i-1}”}
    join words into context  # {“w_{i-1}”} → “w_{i-1}”
    probability = counts[ngram]/context_counts[context]
print ngram, probability to model_file
test-bigram (Linear Interpolation)

$\lambda_1 = ???$, $\lambda_2 = ???$, $V = 1000000$, $W = 0$, $H = 0$

load model into `probs`

for each `line` in `test_file`
    split `line` into an array of `words`
    append “</s>” to the end and “<s>” to the beginning of `words`
for each `i` in `1` to `length(words)-1`
    # Note: starting at 1, after <s>
    P1 = $\lambda_1$ `probs[“w_i”] + (1 - \lambda_1) / V$
    # Smoothed unigram probability
    P2 = $\lambda_2$ `probs[“w_{i-1} w_i”] + (1 - \lambda_2) * P1$
    # Smoothed bigram probability
    H += $-\log_2(P2)$
    W += 1

print “entropy = ”+$H/W$
Thank You!