CS 540 Introduction to Artificial Intelligence
Natural Language Processing

Yingyu Liang
University of Wisconsin-Madison
Sept 28, 2021
Based on slides by Fred Sala
What is NLP?

Combining computing with human language. Want to:

- Answer questions
- Summarize or extract information
- Translate between languages
- Generate dialogue/language
- Write stories automatically
Why is it **hard**?

Many reasons:

- Ambiguity: *"We saw her duck". Several meanings.*
- Non-standard use of language
- Segmentation challenges
- Understanding of the world
  - "Bob and Joe are brothers".
  - "Bob and Joe are fathers".
Approaches to NLP

A brief history

- Symbolic NLP: 50’s to 90’s
- Statistical/Probabilistic: 90’s to present
  - Neural: 2010’s to present

Lots of progress!

Lots more to work to do
Outline

• Introduction to language models
  – n-grams, training, improving issues, evaluation
• Classic NLP tasks
  – Part-of-speech tagging, parsing, dependencies
• Word representations
  – One-hot, word embeddings, transformer-based
Language Models

• Basic idea: use probabilistic models to assign a probability to a sentence

\[ P(W) = P(w_1, w_2, \ldots, w_n) \text{ or } P(w_{\text{next}} | w_1, w_2 \ldots) \]

• Goes back to Shannon
  – Information theory: letters

We begin with the simplest question: given a sequence of symbols, determine if it is a natural sentence.

Turn this into a probabilistic question: What’s the probability of a given sequence of symbols from the distribution of natural sentences? Here we assume there is a ground-truth distribution over natural sentences (e.g., imagine putting together all the natural sentences ever spoken/written and think of the uniform distribution over them).

This is called the language modeling problem. The distribution over sentences is called the language model. It is the basis for many (if not all) natural language processing tasks.
Estimate these probabilities (equivalently estimating the probability tables of these distributions): often using statistical methods on data. This is regarded as training.

The probability tables are too large; not enough data to estimate the entries reliably.

Recall the chain rule

\[ P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2 | w_1) \ldots P(w_n | w_{n-1} \ldots w_1) \]

- How do we estimate these probabilities
  - Same thing as “training”
- From data?
  - Yes, but not directly: too many sentences.
  - Can’t estimate reliably.
Training: Make Assumptions

- Markov-type assumptions:
  \[ P(w_i|w_{i-1}w_{i-2} \ldots w_1) = P(w_i|w_{i-1}w_{i-2} \ldots w_{i-k}) \]

- Present doesn’t depend on whole past
  - Just recent past
  - Markov chains have \( k=1 \). (Present only depends on immediate past).
  - What’s \( k=0 \)?

We make conditional independence assumptions (Markov-type), such that we only need to handle smaller tables. Note that practical data may not satisfy these assumptions. We can think of that we are just trying to find a language model satisfying the assumptions that can best approximate the ground-truth.
To see how well the approximation is, we can first use data to estimate the terms on the right hand side to get the language model, and then sample from this language model to see if it generates good natural sentences.

The sampled sentences from the unigram model are bad.
k=1: Bigram Model

• Markov Assumption:
  - (Present depends on immediate past)
  \[ P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w_{n-1}) \]

• Example:
  texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen outside, new, car, parking, lot, of, the, agreement, reached this, would, be, a, record, november

The sampled sentences from the unigram model are better.
How to estimate the terms in the \( k=n-1 \): \( n \)-gram Model

Can do trigrams, 4-grams, and so on
- More expressive as \( n \) goes up
- Harder to estimate

Training: just count? i.e., for bigram:

\[
P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}
\]

How to estimate the terms in the \( n \)-gram model: just counts from data; equivalently, use frequency to estimate the probability.
Smoothing: a technique for estimating probabilities from counts.

Suppose we have V values and we want to estimate their probabilities from their counts (i.e., the histogram over the V values). Then we add 1 to each count, and then compute the frequencies as the estimate of the probabilities.

Example: For $P(w_i | w_{i-1})$, consider a concrete value $w_{i-1} = \text{"the"}$. Then the V values are the possible values of $w_i$ (the vocabulary), and the counts are the counts of different values of $w_i$ appearing after “the”, i.e., the count(“the”, value of $w_i$). We add 1 to all the counts, and normalize them by their sum to get the frequency. Note that the sum of $(\text{count(\"the\", value of } w_i) + 1)$ over all values is exactly $\text{count(\"the\")} + V$. This gives the smoothed estimate for $P(w_i | w_{i-1})$.

We can add some other smoothing factor other than adding 1. And there are other more sophisticated smoothing methods.
Other Solutions: Backoff & Interpolation

For issue 2, back-off methods

- Use n-gram where there is lots of information, r-gram (with r << n) elsewhere. (trigrams / bigrams)

Interpolation

- Mix different models: (tri- + bi- + unigrams)

\[
\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i)
\]
n-gram Training Issues

Issues:

• **1. Multiply tiny numbers?**
  – **Solution:** use logs; add instead of multiply

• **2. Sparse n-grams**
  – **Solution:** smoothing, backoff, interpolation

• **3. Vocabulary: open vs closed**
  – **Solution:** use `<UNK>` unknown word token
Vocabulary: open vs closed

- Possible to estimate size of unknown vocabulary
  - Good-Turing estimator
- Originally developed to crack the Enigma machine
Break & Quiz

Q 1.1: Which of the below are bigrams from the sentence “It is cold outside today”.

• A. It is
• B. cold today
• C. is cold
• D. A & C
Break & Quiz

Q 1.1: Which of the below are bigrams from the sentence “It is cold outside today”.

• A. It is
• B. cold today
• C. is cold
• D. A & C
Break & Quiz

Q 1.2: Smoothing is increasingly useful for n-grams when

• A. n gets larger
• B. n gets smaller
• C. always the same
• D. n larger than 10
Break & Quiz

Q 1.2: Smoothing is increasingly useful for n-grams when

• A. n gets larger
• B. n gets smaller
• C. always the same
• D. n larger than 10
Evaluating Language Models

How do we know we’ve done a good job?

• Observation
• Train/test on separate data & measure metrics
• **Metrics:**
  – 1. Extrinsic evaluation
  – 2. Perplexity

Evaluation: extrinsic or intrinsic.

Perplexity is the standard intrinsic metric.
Extrinsic Evaluation

How do we know we’ve done a good job?

• **Pick a task** and use the model to do the task

• For two models, $M_1$, $M_2$, compare the accuracy for each task
  
  – Ex: Q/A system: how many questions right. Translation: how many words translated correctly

• Downside: slow; may change relatively
Intrinsic Evaluation: Perplexity

Perplexity is a measure of uncertainty

$$PP(W) = P(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}}$$

Lower is better! Examples:

- WSJ corpus; 40 million words for training:
  - Unigram: 962, Bigram 170, Trigram 109
Further NLP Tasks

Language modeling is not the only task. Two further types:

1. **Auxiliary tasks:**
   - Part-of-speech tagging, parsing, etc.

2. **Direct tasks:**
   - Question-answering, translation, summarization, classification (e.g., sentiment analysis)
Part-of-speech Tagging

Tag words as nouns, verbs, adjectives, etc.

- Tough part: ambiguous, even for people.
- Needs:
  - Getting neighboring word parts right
  - Knowledge of words ("man" is used as a noun, rarely as verb)

<table>
<thead>
<tr>
<th>Model</th>
<th>Features</th>
<th>Token</th>
<th>Unknown</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>56,805</td>
<td>93.69%</td>
<td>82.61%</td>
<td>26.74%</td>
</tr>
<tr>
<td>3Words</td>
<td>239,767</td>
<td>96.57%</td>
<td>86.78%</td>
<td>48.27%</td>
</tr>
</tbody>
</table>

Chris Manning
Parsing

Get the grammatical structure of sentences

The boy put the tortoise on the rug

Chris Manning

• Which words depend on each other? Note: input a sentence, output a tree (dependency parsing)
Break & Quiz

Q 2.1: What is the perplexity for a sequence of $n$ digits 0-9? All occur with equal probability.

- A. 10
- B. 1/10
- C. $10^n$
- D. 0

\[
PP(W) = P(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}}
\]
Break & Quiz

Q 2.1: What is the perplexity for a sequence of $n$ digits 0-9? All occur with equal probability.

- A. 10
- B. $1/10$
- C. $10^n$
- D. 0

$$PP(W) = P(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}}$$
Recent NLP methods use machine learning models on top of the text data. We first need to represent text as numeric numbers.

Basic: represent each word as a numeric vector. Traditional: one-hot encoding

Representing Words

Remember value of random variables (RVs)

- Easier to work with than objects like ‘dog’

Traditional representation: **one-hot vectors**

$$\text{dog} = [0 \ 0 \ 0 \ 1 \ 0]$$

- Dimension: # of words in vocabulary
- Relationships between words?
Recent method: word embeddings, ie, represent each word as a dense vector. More general and more powerful compared to one-hot encoding.

**Smarter Representations**

**Distributional semantics:** account for relationships
- Reps should be close/similar to other words that appear in a similar context

**Dense vectors:**
- **dog** = \([0.13 \quad 0.87 \quad -0.23 \quad 0.46 \quad 0.87 \quad -0.31]^T\)
- **cat** = \([0.07 \quad 1.03 \quad -0.43 \quad -0.21 \quad 1.11 \quad -0.34]^T\)

**AKA word embeddings**
How to get a set of word embeddings, given a set of sentences as training data?

Need to define some scoring of different sets of word embeddings, and then use the scoring to pick the best set of word embeddings.

Word2vec uses a likelihood of the text training data. For simplicity, let $\theta$ denote the set of word embeddings. Also concatenate the whole set of sentences as one big sentence of length $T$. 

$$L(\theta) = \prod_{t=1}^{T} \prod_{-a \leq j \leq a} P(w_{t+j} | w_t, \theta)$$
Training Word Embeddings

Word2vec likelihood

\[ L(\theta) = \prod_{t=1}^{T} \prod_{-a \leq j \leq a} P(w_{t+j}|w_t, \theta) \]

- Maximize this; what’s the probability?
  - Two vectors per word. \( v_w \) \( u_w \) for center/context
    (\( o \) is context word, \( c \) is center)

\[ P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \]

Given a set of word embeddings, we can use the \( P(o|c) \) math expression to compute the probabilities for \( P(w_{(t+j)} \mid w_t) \), and then compute the likelihood. So this gives a definition (also a computation method) for the likelihood of a set of word embeddings.

We then try to find the set of word embeddings that gives the best quality metric (the likelihood).
Recent systems for NLP are using even more sophisticated methods to get the embeddings.

**Beyond “Shallow” Embeddings**

- **Transformers**: special model architectures based on **attention**
  - Sophisticated types of neural networks
- **Pretrained models**
  - Based on transformers: BERT
  - Include context!
- **Fine-tune** for desired task

Vaswani et al. 17