LING530F: Deep Learning for Natural Language Processing (DL-NLP)

Muhammad Abdul-Mageed

muhammad.mageed@ubc.ca

Natural Language Processing Lab

The University of British Columbia

Table of Contents

Motivation

What is more probable?

- Brewing a great cup of coffee/tea/espresso . . .
- Brewing a great cup of entropy . . .

Language Models

- **Assign probabilities to sequences of words** (or characters, etc.).
- Play a **very significant role in NLP**.
- Classically, they are motivated by usage in tasks like **handwriting recognition, spelling correction, speech recognition**, and **machine translation**.
- Recently, their applications are exploding as they become an important block in learning word vectors **(referred to as unsupervised pre-training or generative pre-training)** . . .

Statistical Language Models

- A statistical language model (LM): The conditional probability of the next word given all the previous ones.

1: Statistical LM

$$\hat{P}(W_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_1^{t-1})$$

Brewing a great cup of coffee

- coffee (w_t)
- Brewing a great cup of (w_1^{t-1})
- Brewing a great cup of coffee (W_1^T)

- **Markov assumption**: We do not have to look too far in the past

2: n-gram LM

$$\hat{P}(w_t|w_1^{t-1}) \approx \hat{P}(w_t|w_{t-N+1}^{t-1})$$

3: Bigram LM (one word in the past)

$$\hat{P}(w_t|w_{t-1}) = \hat{P}(w_t|w_{t-1})$$

A Simple LM

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Here are the calculations for some of the bigram probabilities from this corpus

$$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$$

Figure: A simple LM on a mini-corpus. [From J&M, 2017, ch. 3]

Computing P(sequence)

$$P(\texttt{i}\,|\,\texttt{<s>}) = 0.25 \qquad P(\texttt{english}\,|\,\texttt{want}) = 0.0011$$
$$P(\texttt{food}\,|\,\texttt{english}) = 0.5 \qquad P(\texttt{</s>}\,|\,\texttt{food}) = 0.68$$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

$$\begin{aligned} P(\texttt{<s> i want english food </s>}) \\ &= P(\texttt{i}\,|\,\texttt{<s>})P(\texttt{want}\,|\,\texttt{i})P(\texttt{english}\,|\,\texttt{want}) \\ &\qquad P(\texttt{food}\,|\,\texttt{english})P(\texttt{</s>}\,|\,\texttt{food}) \\ &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &= .000031 \end{aligned}$$

Figure: Computing P(sequence) with an LM. [See details in J&M, 2017, ch. 3]

Notes on Computing Probs from LM

- We represent LM probabilities as **log probabilities**
- This **avoids numerical underflow** (if we were to use raw format)
- **Adding in log space = multiplying in linear space**
- To convert back, we exponentiate:

4: Log Probabilities

$$p_1 x p_2 x p_3 = exp(log p_1 + log p_2 + log p_3)$$

Evaluating LMs

Extrinsic vs. Intrinsic Evaluation

- Many models in NLP can be evaluated **extrinsically** and **intrinsically**
- **Extrinsic evaluation**: Plugging LMs in an application like speech recognizer or MT system is best method
- **Intrinsic evaluation:** We use **perplexity**
- We need to split our data, possibly into 80% train, 10% dev, and 10% test.

Evaluation Intuition

- A **good LM is one that predicts unseen data** (test data): $P(sequence)$.

Perplexity

- **Perplexity is a branching factor**: how many words can follow a given word
- It is the **weighted average branching factor**
- It is the **inverse probability on unseen data**, normalized by the number of words (for word-level perplexity).
- **Our goal is to maximize probability of unseen data**.
- Minimizing perplexity = maximizing probability.
- Perplexity is closely related to **entropy (recall: entropy is the avg amount of info.)**

Perplexity

5: Perplexity

$$PP(W) = \hat{P}(w_1, w_2 \ldots w_T)^{-\frac{1}{T}}$$

$$= \sqrt[T]{\frac{1}{\hat{P}(w_1, w_2 \ldots w_T)}}$$

6: Chain Rule

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

Perplexity For Bigrams

7: Perplexity for Bigrams

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

Sample Generation From Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Figure 3.3 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

Figure: [From J&M, 2017, ch. 3]

Sample Generation From WSJ

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 3.4 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

Figure: [From J&M, 2017, ch. 3]

Unseen words

Smoothing

- We will see words at test time that we have not seen in **train**
- Can't assign these **zero probability** (otherwise we'll have division by zero!)
- Called **out-of-vocabulary (OOV)**
- We have to do some **smoothing**, e.g.:
  - Laplace smoothing (add-one)
  - Add-K smoothing
  - back-off and interpolation smoothing
  - Kneser-Ney smoothing
  - ...
- For details, see J&M (2017, ch. 03) ...

Problems with n-gram LM

- **Limited to a short sub-sequence** (e.g., 2 words, for n=3).

Data scarcity

LMs for larger n-gram suffer from **data scarcity**.

- **Does not easily capture word "similarity"**.

Challenge with word similarity

Consider the following two sentences where "cat" and "dog" have similar semantic and grammatical roles. [Bengio et al., 2003].

1. "The cat is walking in the bedroom"
2. "A dog was running in a room"

- Bengio et al. (2003, p. 1139. JML paper):

LM Via A Neural Net

- Express each **word as a feature vector** (real-valued).
- Express the **joint probability function** of word sequences in terms of these word vectors.
- **Learn the word vectors and the joint probability function simultaneously** (using a neural network).

Why Does it Work?

- It is possible to naturally generalize (i.e., transfer probability mass) from (1) to (2-4) such that dog and cat end up with similar vectors:

Example

1. `The cat is walking in the bedroom`
2. `A dog was running in a room`
3. `The cat is running in a room`
4. `A dog is walking in a bedroom`
5. `The dog was walking in the room`
6. ...

Bengio et al., 2003 Neural Model

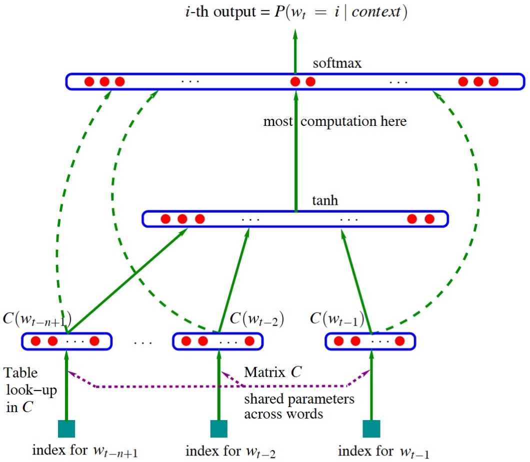

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

It Still Doesn't Scale. . .

It is such a great idea, but there is a **bottleneck**. . .

- **Computationally costly** to obtaining the output probabilities (since **softmax operates on all vocabulary**).
- **Bottleneck** in the computation of **the activations of the output layer**.
- An n-gram model **does not require the computation of the probabilities for all the words in the vocabulary**.
- **Mikolov et al. (2013)** Scale learning word vectors with a large vocabulary (V).