ECE598: : Representation Learning: Algorithms and Models	Fall 2017
Lecture 13: Language Modeling	
Lecturer: Prof. Pramod Viswanath	*Scribe: Yiren Wang, Oct 17, 2017*

13.1 Probabilistic Langauge Models

13.1.1 Introduction

The probabilistic language model is to compute a probability distribution of a sentence of words sequences of words, i.e. $P(s) = P(w_1, w_2, ..., w_N)$, or to compute the probability of an upcoming word, namely, give a sentence $s = (w_1, w_2, ..., w_N)$, where w_t are discrete words, N is the random-valued sequence length, the goal is to compute the prabability of an upcoming word $P(w_t|w_1, ..., w_{t-1})$.

Estimating the probabilistic language model is very useful in many natural language related applications, in machine translation, language modeling can help with the correct word ording and word choices, e.g. P(the cat is small) > P(small the is cat), and P(walking *home* after school) > P(walking *house* after school).

13.1.2 The N-gram Model

In probabilistic language models, the probability is usually conditioned on the window of N previous words. The joint probability of the entire sequences $P(s) = P(w_1, w_2, ..., w_N)$ is decomposed using the chain rule of probability:

$$\begin{aligned} P(w_1, ..., w_N) &= P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1}) \\ &= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1}) \end{aligned} \tag{13.1}$$

While one straightforward method is to estimate these conditional probabilities directly from relative frequency counts by taking a very large corpus and counting the number of times of sequence $(w_1, ..., w_i)$ and $(w_1, ..., w_{i-1})$, it turns out that it's hard to have enough data for estimation. This is because language is creative; new sentences are created all the time, there are too many possible sentences and we wont always be able to count entire sentences [JM14].

One feasible solution is the **N-gram model**, where the basic intuition is that instead of computing the probability of a word given its entire history, we can approximate the history by just the latest few words.

$$
\begin{aligned}
\textit{Unigram}: \quad & P(w_1, ..., w_N) \approx \prod_{i=1}^{N} P(w_i) \\
\textit{Bigrams}: \quad & P(w_1, ..., w_N) \approx \prod_{i=1}^{N} P(w_i | w_{i-1}) \\
\textit{n-grams}: \quad & P(w_1, ..., w_N) \approx \prod_{i=1}^{N} P(w_i | w_{i-n+1}, ..., w_{i-1})
\end{aligned}
\tag{13.2}
$$

Note: Our N-gram model (Eqn. 13.2) is based on the Markov assumption that the probability of a word depends only on the previous few words, but not too far into the past.

An intuitive way to estimate the probabilities in Eqn. 13.2 is the maximum likelihood estimation (MLE). We can compute the MLE estimations for N-gram model parameters by counting from a large corpus and normalizing the counts to generate probabilities:

$$
\begin{aligned}
\textit{Bigrams}: \quad & P(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_i)} \\
\textit{n-grams}: \quad & P(w_i | w_{i-n+1}, ..., w_{i-1}) = \frac{count(w_{i-n+1}, ..., w_i)}{count(w_{i-n+1}, ..., w_{i-1})}
\end{aligned}
\tag{13.3}
$$

Figure 13.1 is an example of estimating a bigram language model. The first table shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. We can see that the matrix are sparse (majority of word pairs have zero counts). The second table shows the bigram probabilities after normalization, which can be used to compute the probability of sentences by simply multiplying the appropriate bigram probabilities together.

Note: Some practival issues:

- In practice its more common to use higher order n-gram models (i.e., with larger n) when there is sufficient training data.
- For these larger n-grams, it is necessary to assume extra context to the left and right of the sentence end.
  e.g. to compute trigram probabilities at the very beginning of the sentence, we can use two pseudo-words for the first trigram (i.e., $P(I| < s >< s >)$).
- **Log Probabilities:** Since probabilities are in nature less than or equal to 1, the more probabilities we multiply together, the smaller the product becomes. Multiplying a large number of N-grams together would result in numerical underflow. To avoid this problem, we always represent and compute the language model probabilities in log format. Adding in log space is equivalent to multiplying in linear space, so we combine log probabilities by adding them.

13.1.3 Model Evaluation

To evaluate the language model, we train parameters of language models on a training set, and then test the model's performance on data we haven't seen. The most used evaluate metric is called

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 4.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Figure 13.1: Example from Berkeley Restaurant Project [JM14]

perplexity (PP).

The perplexity of a language model is the inverse probability of the test set, normalized by the number of words. For $s = (w_1, ..., w_N)$, the perplexity of s with a n-gram language model is:

$$\begin{aligned} PP(s) &= \sqrt[N]{\frac{1}{P(w_1, ..., w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, ..., w_{i-1})}} \quad \text{(chain rule)} \\ &\approx \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-n+1}, ..., w_{i-1})}} \quad \text{(n-grams)} \end{aligned} \tag{13.4}$$

13.2 Smoothing

Like many statistical models, the N-gram probabilistic language model is dependent on the training corpus. One practical issue with this is that some word sequences and phrases appear in practice (or in the test set), may not also occur in the training set. So it is important to train robust models that generalize well to handel the unseen words and zero probabilities.

To keep a language model from assigning zero probability to these unseen events, well have to shave off a bit of probability mass from some more frequent events and give it to the events weve never seen. This modification is called smoothing or discounting. Popular smoothing methods will be introduced in this section, including:

- Good Turing Smoothing
- Absolute Discounting Smoothing
- Kneser-Ney Smoothing

13.2.1 Good-Turing Smoothing

The Good-Turing estimate [Goo53] states that for any n-gram that occurs r times, we should pretend that it occurs r^* times such that $r^* = (r+1)\frac{N_{r+1}}{N_r}$, where N_r is the number of n-grams that occur r times in the training corpus. Namely, a word with occurrence r should occur with probability:

$$P(w^{(r)}) = \frac{(r+1)N_{r+1}}{N_r \sum_r rN_r} \tag{13.5}$$

It is shown in [OS15] that for distributions over k symbols and n samples, a simple variant of Good-Turing estimator is always within KL divergence of $(3 + o_n(1))/n^{1/3}$ from the best estimator, and that a more involved estimator is within $\tilde{O}_n(\min(k/n, 1/\sqrt{n}))$.

13.2.2 Absolute Discounting Smoothing

The basic idea of the Absolute Discounting Smoothing method [NEK94] is to steal probability mass to unseen samples. The equation for interpolated absolute discounting applied to bigrams:

$$P_{AD}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) - d}{count(w_{i-1})} + \lambda(w_{i-1})P(w_i) \tag{13.6}$$

where first term is the discounted bigram, the second term is the unigram model $P(w_i)$ with interpolation weight $\lambda(w_{i-1})$.

Note: While absolute discounting smoothing method (Eqn. 13.6) solves the zero probability problem with unigram distribution, there are some potential problems for continuation. For example, consider the task of prediciting the next word in sentence "I cant see without my reading ____". The word *glasses* is more likely to follow and thus should have a higher probability than word *Kong*. However, a standard unigram model may assign higher probability to *Kong*, since *Hong Kong* is a very frequent phrase. This issue will be solved with Kneser-Ney Smoothing method in section 13.2.3.

13.2.3 Kneser-Ney Smoothing

Kneser-Ney smoothing algorithm [KN95] roots in absolute discounting method. It solves the issue stressed in section 13.2.2 by augmenting absolute discounting with a more sophisticated way to handle the lower-order unigram distribution. The equation for Interpolated Kneser-Ney smoothing for bigrams is:

$$P_{KN}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) - d}{count(w_{i-1})} + \lambda(w_{i-1})P_{cont}(w_i) \tag{13.7}$$

- ***Continuation probability:*** in KN-smoothing, we estimate P_{cont} based on the number of different contexts words w has appeared in the number of bigram types it completes.

$$P_{cont}(w) = \frac{|v : count(vw) > 0|}{\sum_{w'} |v : count(vw') > 0|} \tag{13.8}$$

  In this way, a frequent unigram occuring in only one context (e.g. word *Kong* in context *Hong Kong*) will have a low continuation probability.

- ***Normalizing discount:*** $\lambda(w_{i-1})$ is a normalizing constant that is used to distribute the probability mass we've discounted:

$$\lambda(w_{i-1}) = \frac{d}{count(w_{i-1})}|w : count(w_{i-1}w) > 0| \tag{13.9}$$

  where the first term is the normalized discount, and the second term $|w : count(w_{i-1}w) > 0|$ is the number of word types that can follow word w_{i-1}.

Finally, we can generalize Eqn. 13.7 to n-grams with recursive formulation:

$$\begin{aligned} P_{KN}(w_i|w_{i-n+1}, ..., w_{i-1}) = &\frac{\max(C_{KN}(w_{i-n+1}, ..., w_i) - d, 0)}{C_{KN}(w_{i-n+1}, ..., w_{i-1})} \\ &+ \lambda(w_{i-n+1}, ..., w_{i-1})P_{KN}(w_i|w_{i-n+2}, ..., w_{i=1}) \end{aligned} \tag{13.10}$$

where count C_{KN} depends on the order of n-grams being counted:

$$C_{KN}(\cdot) = \begin{cases} \#\text{occurrence of } (\cdot) & \text{for the highest order} \\ \#\text{unique word types for } (\cdot) & \text{for lower orders} \end{cases} \tag{13.11}$$

13.3 Neural Language Models

As is introduced in section 13.1.2, the n-grams probabilistic language model is based on the Markov assumption that the probability of a word depends only on the previous few words. This assumption is necessary for feasible estimation, but in fact incorrect for accurate computing. Empirical studies show that performance of n-grams models improves with keeping around higher order n-grams counts and doing smoothing (section 13.2).

Neural network based language models (NNLM) outperform standard n-gram language models. In neural language models, words are projected into low-dimentional space, and similar words are automatically clustered together. Also, smoothing is solved implicitly in the neural models.

13.3.1 Feedforward NNLM

Feedforward neural network based language model [BDVJ03] is similar to N-grams models in that the history is also represented by a limited context of n previous words.

Figure 13.2 shows the neural architecture, which consists of a projection layer, a hidden layer and an output layer. The objective is to estimate probabilities for word w_j given the previous n words $w_{j-n+1}, ..., w_{j-1}$

- *Projection Layer:* The projection layer maps each word from a word index to a dense vector with a word feature matrix $C \in \mathbb{R}^{|V| \times d}$, a shared parameter across all words, where each row i is the word vector $C(i)$ for word i.
- *Hidden Layer:* The hidden layer takes in all the input word vectors and performs linear and non-linear transform: $H = tanh(Ux + b)$, where $x = \Big(C(w_{j-n+1}), ..., C(w_{j-1})\Big)$.
- *Output Layer:* The output layer is a softmax layer that generates probability for each word in the vocabulary that is the j-th word: $P(w_j = k|h_j) = softmax(v_{w_t})$

The objective is to look for the optimal parameter sets Θ that maximizes the log-likelihood:

$$\mathcal{L} = \frac{1}{T} \sum_t f(w_j, w_{j-1}, ..., w_{j-n+1}; \Theta) + R(\Theta) \tag{13.12}$$

where f is the mapping function that describe the model architecture, $R(\Theta)$ is a regularization term.

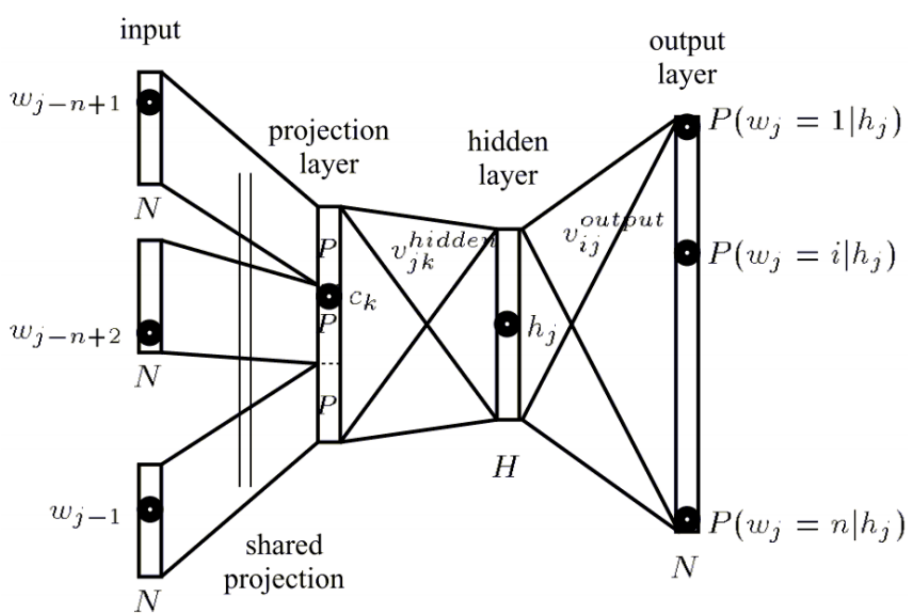

Figure 13.2: Feedforward neural network based language model [BDVJ03, MKB+10]

13.3.2 Recurrent NNLM

Recurrent neural network based language model (RNNLM) [MKB+10] solves the problem of limited word history. In recurrent networks, history is represented by neurons with recurrent connections. Compared with feedforward NN that project single word into a dense vector, RNN learns to encode whole history into a low dimensional space. So intuitively, RNNLM is better for handling long-term dependencies in natual languages.

Figure 13.3 shows the architecture for RNNLM, which consists of an input layer x, a hidden (context) layer s and an output layer y.

- *Input Layer:* The input vector $x(t)$ is the concatenation of current input word vector $w(t)$ and the output from neurons in context layer from the last time stamp $s(t-1)$. Namely, $x(t) = \Big[w(t); s(t-1)\Big]$.
- *Context Layer:* The context layer maps input vectors into a low dimensional space with sigmoid activation function $f(\cdot)$:

$$s(t) = f\Big(Wx(t)\Big) = f\Big(W\Big[w(t); s(t-1)\Big]\Big) \tag{13.13}$$

- *Output Layer:* The output layer use the same softmax function as in feedforward architecture (section 13.3.1)

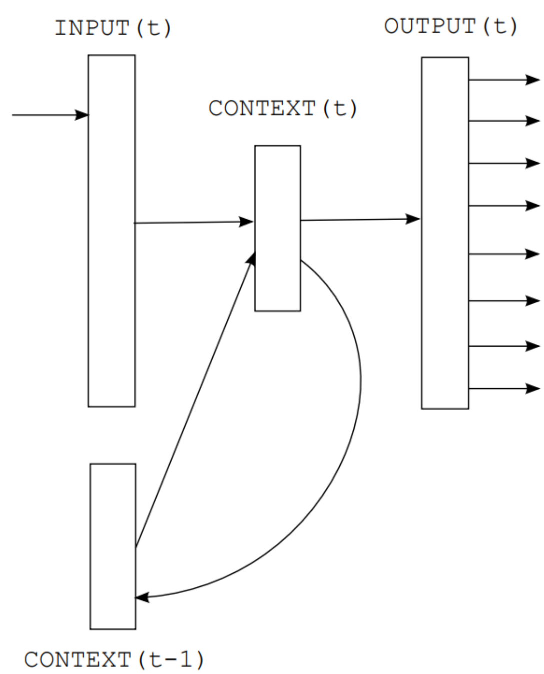

Figure 13.3: Recurrent neural network based language model [MKB+10]

Bibliography

[BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[Goo53] Irving J Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

[JM14] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[KN95] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[MKB+10] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[NEK94] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

[OS15] Alon Orlitsky and Ananda Theertha Suresh. Competitive distribution estimation: Why is good-turing good. In *Advances in Neural Information Processing Systems*, pages 2143–2151, 2015.