11-761 Language and Statistics Final Project
Recurrent Neural Network and High-Order N-Gram Models for POS Prediction

Yun-Nung (Vivian) Chen, Ting-Hao (Kenneth) Huang, and William Yang Wang

School of Computer Science, Carnegie Mellon University, USA

{yvchen, tinghaoh, yww}@cs.cmu.edu

Abstract

The task of part-of-speech (POS) language modeling typically includes a very small vocabulary, which significantly differs from traditional lexicalized language modeling tasks. In this project, we propose a high-order n-gram model and a state-of-the-art recurrent neural network model, which aims at minimizing the variance in this POS language modeling task. In our experiments, we show that the recurrent neural network model outperforms the n-gram model on various datasets, and the linear interpolation of the two models, which balances the pros and cons of discriminative and generative models, has significantly reduced the perplexities.

1. Recurrent Neural Network

To introduce a simple recurrent neural network language model, we first denote the input layer x, hidden layer s, word w, and output layer y. Then, the input at time t is the concatenation of word w at t and the hidden layer at previous time stamp $t-1$:

$$x(t) = w(t) + s(t-1)$$

The hidden layer s at time t can be represented as the function $f(z)$ taking the sum of product input layer u and each component of the weight vector u:

$$s_j(t) = f(\sum_i x_i(t) u_{ji})$$

where the $f(z)$ is a sigmoid function:

$$f(z) = \frac{1}{1+e^{-z}}$$

The output layer y can be represented as the function $g(z)$ taking the sum of the product of the hidden layer s and the weight vector$v$:

$$y_k(t) = g(\sum_j s_j(t) v_{kj})$$

where the $g(z)$ is a softmax function that makes sure the posterior probabilities sum up to 1.

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

1.1. RNNLM Toolkit

We use the RNNLM toolkit provided by Mikolov et al. (Interspeech 2011). The main program is "./rnnlm", which calls the "rnnlmlib" code to train RNNLM models, and evaluate models. We have modified the rnnlm and rnnlmlib to enable the interactive ouput required by the project, as well as combining SRILM decoding with the RNNLM decoding by performing real-time interpolation of trained models from both toolkits. The detailed FAQ and original README can be found under the rnnlm-0.3e source code directory. We have also included code for converting SRILM output to simple probability distributions and the code for computing perplexity of a given RNNLM model.

2. High-Order N-Gram Model

We use 9-gram language model from SRILM toolkit, and the posterior probability is evaluated as

$$P(w_i \mid w_{i-8}...w_{i-1}) = \frac{C(w_{i-8}...w_i)}{C(w_{i-8}...w_{i-1})}. \tag{1}$$

2.1. SRILM Toolkit

SRILM is a toolkit for building and applying statistical language models (LMs), and it runs under Linux/UNIX system[1].

2.2. Used Components

- Model training: ./ngram-count
  training 9-gram language model with Witten-Bell discounting and Good-Turing smoothing
- Model evaluation: ./ngram
  computing the perplexity of language model given testing data
- Language model reading: ./ngram_test
  implementation of program to dynamically read probability from language model and output the prediction of next POS tag

3. POS Prediction

3.1. Dynamic Viterbi Decoding

Because we need to dynamically output the prediction of POS tags, we compute the posterior probability given current observed history:

$$P(w_n \mid h) = \lambda P_{9gram}(w_n \mid h) + (1-\lambda) P_{RNN}(w_n \mid h), h = w_1, w_2, ..., w_{n-1}, \tag{2}$$

where λ is a interpolation weight, which combines RNNLM and SRILM's posterior probability as final interpolated estimation.

3.2. Domain Adaptation

We trained some out-domain models M_k, and use out-domain training data to compute the cross-entropy (perplexity) between it and each of the out-domain models to get the similarity $SIM(p, M_k)$.

$$SIM(p, M_k) = \frac{1}{H(p, q_k)} = \frac{1}{-\sum_x p(x) \log q_k(x)}, \tag{3}$$

[1] http://www.speech.sri.com/projects/srilm

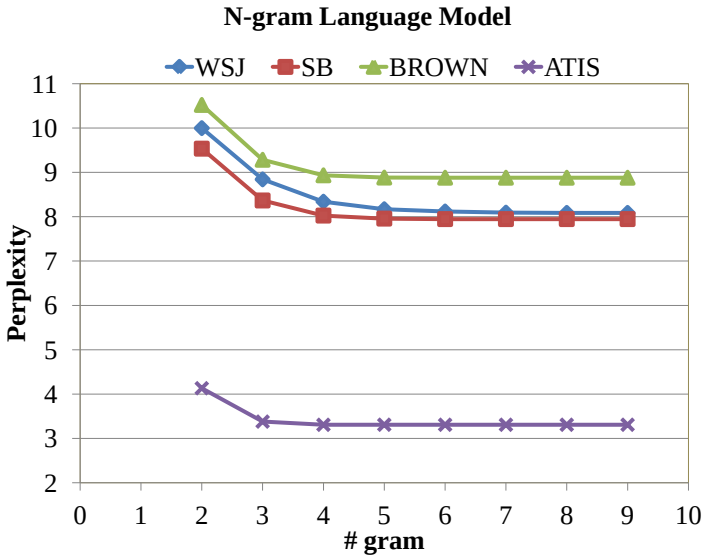

Figure 1: Perplexity of n-gram language model

where p is the testing data's probability and q_k is the probability from model M_k. Then we choose the better model for domain adaptation so that the model can include the distribution more similar to out-domain data.

4. Experiments

4.1. Experiment Setup

Both RNNLM and SRILM models are evaluated on four different datasets respectively: The Wall Street Journal(WSJ), Switchboard(SB), Brown, and ATIS. For each dataset, we randomly select 90% of data as training set, 5% as testing set, and 5% as develope set. Models are trained on training set, tuned on develope set, and test on testing set.

4.2. Effect of N

We first compare the perplexities of SRILM n-gram models with different n. Without applying any smoothing technologies, the best model for WSJ is 5-gram, for SW and Brown is 4-gram, and for ATIS is trigram. Higher order language models are not beneficial even harmful if no smoothing technologies are applied.

Then we evaluate the smoothed SRILM models. Based on the experiments done by [Carpenter, 2005], the Witten-Bell discounting is better than Dirichlet and absolute discounting for 12-gram character language model. So we adopted the Witten-Bell and Good Turing discounting, and tune the discounting parameter carefully for each n-gram model. As a result, the perplexity of smoothed n-gram language model is apparently lower than non-smoothed n-gram model with all n in all data set. And the perplexity reduced when the order of language model is getting higher. In other words, we can get benefit from using high order language models when smoothing technologies applied. The changes of perplexity are shown in Figure 4.2.

4.3. Final Results

Finally, we compare the performance of SRILM, RNNLM, and the interpolated model of them. The results are shown in Table 1. The average perplexity of SRILM over four datasets is 7.0575, and the average perplexity of RNNLM is 6.4975. The average perplexity of interpolation model is 4.09.

Table 1: Performance comparison between recurrent neural network and high-order n-gram language model

Corpus	SRILM	RNNLM	Interpolation
WSJ	8.09	7.43	**5.48**
Switchboard	7.95	7.09	**3.67**
Brown	8.88	8.15	**5.75**
ATIS	3.31	3.32	**1.46**

4.4. Space Comparison

Table 2 shows the space of two different language models, and we can find that high-order n-gram language models have larger space but recurrent neural network requires relatively small space.

Table 2: Space comparison between recurrent neural network and high-order n-gram language model

Model	Size	Description
N-gram LM	138 M	9-gram w/ smoothing
	32 M	7-gram w smoothing
RNN	7.9 M	1000 hidden layers, bptt = 10
	2.2 M	500 hidden layers, bptt = 10
	131 K	100 hidden layers, bptt = 10

5. Conclusion

High-order n-grams reduce variance for POS language modeling Recurrent neural network outperforms high-order n-gram Recurrent neural network requires longer training time but less space Interpolation balances pros and cons of generative and discriminative models

6. Responsibilites and Contributions

- Vivian Chen has contributed to the data pre-processing, the design and the implementation of high-order n-gram language models for POS prediction, the creation of submission scripts, the dynamic viterbi decoding for model integration, the combination of two toolkits, the presentation slides and part of this writeup.
- Kenneth Huang has contributed to the investigation of related toolkits, the design of overall experiments and evaluations, tuning of high-order n-gram model parameters, the smoothing techniques for high-order n-gram LMs, SRILM model generation, the compilation script of all toolkits, the writeup and the slides.
- William Wang has contributed to the task analysis, the design and implementation of recurrent neural network language models, the literature reviews, the theoretical background of recurrent neural network LM, the parameter estimation for RNNLM, the linear interpolation of both models, the writeup and the slides.

7. Comments and Suggestions

- Vivian Chen
  This project can train my programming skills because I modified the code in the toolkit a lot. It make me understand language modeling more and consider the tradeoff between accuracy and perplexity. The best thing I think is that

this is a group project and that I have two excellent partners. I learned a lot about team work, which makes us work efficiently and full of energy. I also have some suggestions. Dynamic outputting the POS prediction takes us a lot of time, which shouldn't be the part we focus on. The submission script has some complicate requirements, which takes us some time to see. It should allows us to submit complete report for documentation later so that we have more time to write a good report.

- Kenneth Huang
  It's an interesting project. The implementations really drove us to learn the details of various models and toolkits. However, for those groups using language models rather than simple POS taggers or classifiers, it seems inevitable to spend huge amount of time to deal with the final decoding and dynamic output. If we are allowed to output the prediction results in some easier ways, we can save a lot of time to work on models. I strongly recommend the instructor and TAs of this class re-think about the evaluation part – especially the output requirements – of the final project.

  Another thing I learned from this project is that it could be very tricky, not only theoretically but also realistically, to select good parameters. In this project, we spend lots effort to design the experiments, and tune and test our models. When building models, many real-world issues may occur and make some guaranteed situations become difficult to control. Building a robust model is not only about mathematics, but also involve in data collecting, programming skills, hacking skills, even management skills. That's the most precious lesson I learned from this project.

- William Wang
  This project is absolutely of significant amount of interests to me, not only because of the content, but also the competition among peer students, most of whom might not have chance to participate formal evaluations (e.g. Darpa, NIST, IARPA). The fun part also includes the research aspect of the project, which requires individuals to conduct literature reviews about the topic, design experiments, implement the detailed algorithms, and analyzing the results. The only suggestion I have is that may be next year we can simplify the submission requirements and format, which take up a lot of time, but it's not directly related to the goal of this project.

8. References

[1] B. Carpenter, "Scaling High-Order Character Language Models to Gigabytes," in *ACL Workshop on Software*, 2005.

[2] A. Stolcke, "SRILM - An Extensible Language Modeling Toolkit," in *ICSLP InterSpeech*, 2002.

[3] M. Tomás, "Empirical Evaluation and Combination of Advanced Language Modeling Techniques," in *InterSpeech*, 2011.