Neural Network Language Modeling with Letter-based Features and Importance Sampling

Hainan Xu[1], Ke Li[1], Yiming Wang[1], Jian Wang[2], Shiyin Kang[3], Xie Chen[4], Daniel Povey[1], Sanjeev Khudanpur[1]

[1]Center for Language and Speech Processing, Johns Hopkins University; [2]Toutiao AI Lab, Beijing, China; [3]Tencent AI Lab, Shenzhen, China; [4]Machine Intelligence Laboratory, Cambridge University

Overview

- Extends Kaldi toolkit to support neural-based language modeling;
- Combining subword features (letter n-grams) and one-hot encoding of frequent words to handle large vocabularies containing infrequent words;
- Propose a new objective function that allows for training of unnormalized probabilities;
- Propose an importance sampling based method to speed up training when the vocabulary is large;
- Kaldi-RNNLM trains faster than a number of other toolkits;
- Kaldi-RNNLM achieves better perplexities and rescoring word-error-rates compared to a number of other toolkits.

Use of Subword Features

- Kaldi-RNNLM combines the use of subword features and one-hot encoding of frequent words to generate word-embeddings;
- Input-embedding and Output-embedding are tied;
- We currently use letter n-grams up to 3-grams, done at script level, and this could be easily expanded.

Cross-entropy Objective Function

We write $\mathbf{z}$ as the layer of the neural network before the final softmax operation, and j as the index for the correct word, then cross-entropy is computed as,

$$z_j - \log \sum_i \exp(z_i) \tag{1}$$

New Objective Function

Note that $\log x \leq x - 1$, we define the following objective function,

$$z_j + 1 - \sum_i \exp(z_i) \tag{2}$$

Analysis of the Objective

- (2) = (1) iff $\sum_i \exp(z_i) = 1$;
- The objective is a lower bound on the cross-entropy objective, with equality when the output is a well-normalized distribution;
- When the new objective is maximized, it is similar to cross-entropy training plus a penalty term that makes the output of the network sum to a value close to 1 ($\sum_i \exp(z_i) \simeq 1$);
- During test time, we use z_j as the computed “probability” as an approximation since we know the expectation of $1 - \sum_i \exp(z_i)$ is 0.

Importance-Sampling

To compute

$$\sum_i f(x_i)$$

We define

$$y_i = \begin{cases} \frac{f(x_i)}{p_i}, & \text{with probability } p_i \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Then

$$\mathbb{E}[y_i] = \mathbb{E}[f(x_i)] \tag{4}$$

$$\mathbb{E}[\sum_i y_i] = \mathbb{E}[\sum_i f(x_i)] \tag{5}$$

$\sum_i y_i$ has a lot of zeros which is easy to compute.

Neural Network Training with Importance Sampling

- If we use importance-sampling to compute cross-entropy loss, then the log operation makes the estimate biased;
- Equation (2) takes the sum out of the log, making it possible to use importance-sampling to compute an unbiased estimate of the objf;
- Any distribution for p_i is OK, but the closer it is to the ground-truth, the faster the convergence;
- We sample words from the averaged n-gram distribution of a minibatch;

Evaluation

We compare Kaldi-RNNLM with CUED-RNNLM and TensorFlow-RNNLM on 5 datasets.
We report perplexities, word-error-rates as well as training speed.

Perplexity

	dataset	CUED	TF	KALDI
AMI	train	52.2	49.1	47.0
	dev	76.5	82.1	**72.2**
HUB4	train	99.7	131.2	73.7
	dev	197.5	192.0	**180.7**
SWBD	train	37.7	46.7	43.4
	dev	52.0	54.3	**47.5**
WSJ	train	39.1	37.4	43.4
	dev	67.2	64.5	**50.9**
TED-LIUM	train	123.3	133.4	96.3
	dev	169.6	154.7	**137.0**

Table 1: Perplexities of Different RNNLMs

Word-error-rate

	dataset	Baseline	CUED	TF	KALDI
AMI	dev	24.2	23.0	23.2	**22.8**
	eval	25.4	**23.9**	24.2	**23.9**
HUB4	test	14.4	12.8	13.1	**12.6**
SWBD	swbd	8.0*	**7.0**	7.1	**7.0**
WSJ	dev93	7.1*	6.1	6.0	**5.8**
	eval92	5.0*	3.9	**3.8**	3.9
TED-LIUM	dev	10.7*	10.3	10.4	**9.9**
	test	9.8*	9.3	9.3	**9.0**

Table 2: WER of Lattice-rescoring of Different RNNLMs

∗: baseline already rescored with a 4-gram arpa model.

Training Speed

RNNLM	Speed (words/second)
CUED-RNNLM, CE	8.30K
CUED-RNNLM, NCE	12.8K
TensorFlow-RNNLM	23.3K
Kaldi-RNNLM, no sampling	18.3K
Kaldi-RNNLM, 512 samples	31.0K

Table 3: Training Speed* of Different RNNLMs

*Evaluation is done on a Tesla K10.G2.8GB GPU.

Acknowledgements

This work was partially supported by DARPA LORELEI award number HR0011-15-2-0024, NSF Grant No CRI-1513128 and IARPA MATERIAL award number FA8650-17-C-9115.

Contact Information

- Web: `http://www.hainanxv.com`
- Email: hxu31@jhu.edu