Dream Machine

ap-VTK
Tetsuhiro Tamada, Valentin Deyringer
Center for Information and Language Processing, University of Munich, Germany
Tetsuhiro.Tamada@campus.lmu.de, v.deyringer@campus.lmu.de

1. Introduction

- The task was to build a **Dream Machine** which automatically generates Text.
- Our system generated **sentences one by one**.
- As data basis we used the **Wall Street Journal Corpus**.
- To measure the quality we used the measure of **perplexity**.
- We chose **Python 2.7** as our programming language because it offers convenient ways to implement the underlying procedures and fast enough computation.

2. Preprocessing

- Not much preprocessing was necessary, since the provided corpus was already edited very well following the **Penn Treebank Tokenization guidelines**[1].
- **Sentence segmentation** was not an issue as sentences were separated by line breaks.
- The step of **tokenization** also presented no problems and could be implemented easily by splitting a sentence at white spaces.
- A few **tags** resembling different kinds of brackets were **replaced**.
  (`-LRB-`, `-RRB-`, `-LSB-`, `-RSB-`, `-LCB-`, `-RCB-`)
- To model the **start and end of a sentence** we added the tags `<s>` and `<\s>`.
- As optional step we implemented a switch to covert the whole text into **lower case**.

3. Text Generation

- A **language model** describes word probabilities in a given language.
- Optimally language models are trained on **large and diverse** corpora.
- The **probability of a word sequence** is given by:

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{n-1}) \tag{1}$$

- In an **ngram model** word probabilities rely only on the $n-1$ preceding words:

$$P(w_i | w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{count(w_{i-(n-1)}, \ldots, w_i)}{count(w_{i-(n-1)}, \ldots, w_{i-1})} \tag{2}$$

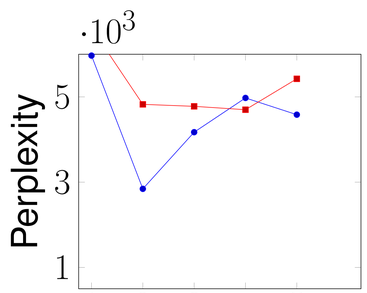
- To generate a sentence we **start with the sentence start tag** `<s>`.
- The probabilities of **all possible following words** are then calculated with equation 2.
- One of the possible words is then selected by a **weighted random choice** and appended to the existing word sequence.
- This procedure is continued **until the sentence end tag** `<\s>` is reached which means that the sentence is generated.

4. Evaluation

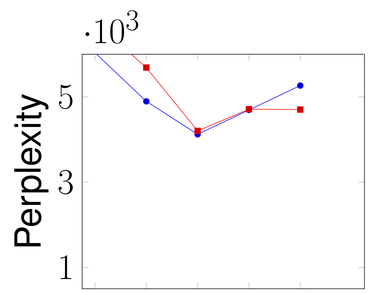
- How likely does the generated sentence in test corpus occur? Which model generates the best sentence?
- Generated sentences through bigram and trigram model reached especially a better score
- Nomalization of train and test corpus to lower case was useful
- Perplexity calculation in Log Space:

$$PP(S) = 2^{-1/N \sum_{n=1}^{N} \log_2 P(w_i | w_1^{i-1})} \tag{3}$$

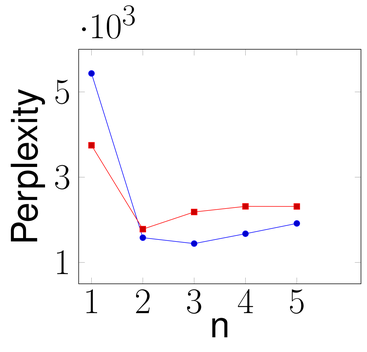
- The smaller perplexity a sentence has, the more likely it appears in test corpus
- Result:
  y-axis: perplexity
  x-axis: n



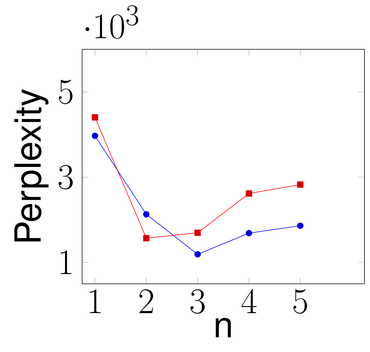
using **trigram** model for the calculation, with **50**% of test data vocab as unknown, averaged over 10 sents
blue: normalized to lowercase
red: not normalized to lowercase



using **trigram** model for the calculation, with **25**% of test data vocab as unknown, averaged over 10 sents
blue: normalized to lowercase
red: not normalized to lowercase



using **bigram** model for the calculation, with **50**% of test data vocab as unknown, averaged over 10 sents
blue: normalized to lowercase
red: not normalized to lowercase



using **bigram** model for the calculation, with **25**% of test data vocab as unknown, averaged over 10 sents
blue: normalized to lowercase
red: not normalized to lowercase

5. Program / Code

- Python 2.7
- Apart from numpy[2] which offers a better "random choice" no third party libraries were used
- Execution:
  - Options:

`-h`	shows help
`-n N`	order of ngram model to use
`-l` / `--lower`	normalize text to lower case
`-s`	number of sentences to generate

  - example program call:

```
./src/dream_machine.py -n 3 -l -s 5 res/wsj/train-wsj-00-20.sent > gene
```

- evaluation
  - Options:

`-h`	shows help
`-n N`	order of ngram model to use
`-l` / `--lower`	normalize text to lower case
`-u`	fraction of words to treat as unknown

  - example program call:

```
./src/dream_machine_test.py -n 3 -l -u 0.5 generated_sents.txt res/wsj/
```

- Options:

`-h`	shows help
`-n N`	order of ngram model to use
`-l` / `--lower`	normalize text to lower case
`-s`	number of sentences to generate

- example program call to generate text:

```
./src/dream_machine.py -n 3 -l -s 5 res/wsj/train-wsj-00-20.sent
```

6. Future Work

- **Interpolation** could be used by incorporating lower order ngram models to adjust the probabilities of possibly selected words.
- By adding more **linguistic knowledge** the generation models could be improved further.
  - Tuning probabilities by taking into account the probabilities of **part of speech tag sequences**.
- Check output for **grammaticality** with grammar rules.
- To evaluate the output **other evaluation metrics** like could be used
  (taking into account some specifics)
  - BLEU score [1]
  - Word Error Rate

References

[1] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[1] https://www.cis.upenn.edu/ treebank/tokenization.html
[2] http://www.numpy.org/