Team GLaDOS - Final Project

Anton Samson, Brian Keung
{ads8, bkeung} @ sfu.ca

December 3, 2014

1 Motivation

As our implementation of a stack decoder performed well on the task of French to English sentence translation, it was decided that the decoder implementation would also be performant in the task of Chinese to English sentence translation. After adapting the decoder to function with the toy dataset, efforts were made to improve the decoder and reduce overall runtime.

An attempt was also made to make use of our previous implementation of a Chinese language sentence segmenter, however, knowing that this approach would require generating new word alignments and a phrase table, this approach was abandoned due to time constraints. Focus was instead shifted on creating a feedback loop between the decoder and reranker.

2 Approach

2.1 Decoder

The decoder was adapted to allow for the use of large amounts of data, better memory usage, and to allow for index offsets when dealing with chunked inputs. Many ease of life improvements were also made to the decoder in the form of various shell scripts and attempts at distributed computing.

2.1.1 Distortion Limit

A distortion limit was applied that allowed phrase re-orderings in a sentence up to a certain distance, d. This limit determined the maximum number of words that could be skipped when generating hypotheses.

$$allowable\,range = abs(index\,of\,last\,translated\,position - start\,position\,of\,current\,phrase) <= d$$

2.1.2 Phrase Retrieval

A new algorithm was written to retrieve untranslated phrases for use by the decoder. The coverage vector is first evaluated and the indices of untranslated slots are returned. These indices are then iterated over and any phrases that do not overlap with the current coverage vector are returned.

Algorithm 1 Phrase Retrieval

```
1: function GET_PHRASES
2:     untranslated ← list of indices in h.coverage that are untranslated
3:     for each value start in untranslated do
4:         for j in start to len(f) do
5:             for k in j+1 to len(f)+1 do
6:                 translated ← list of indices in h.coverage[j : k] that are translated
7:                 if translated == 0 then
8:                     if f[j : k] in tm then
9:                         yield tm[f[j : k]], j, k
```

2.1.3 Top-N

The decoder allows for the pruning of the translation stacks by keeping only the top N results as sorted by best log probability. However, it was observed that keeping more hypotheses resulted in better scores and that this form of pruning may inadvertently prune the best translation instead. Choosing a lower N allowed for greater speed in decoding but generally poorer scores while a greater N allowed for better scores but exponentially slower decoding.

The decoder can also output the top N values for each sentence to generate an n-best list.

2.1.4 Distortion Penalty

No penalty on distortion was applied in this decoder implementation as Chinese to English translations appeared to have many re-orderings at great distances. However, a new penalty formula was introduced but not used.

$$distortion\,penalty = \log(penalty^{index\,of\,last\,translated\,position - start\,position\,of\,current\,phrase})$$

2.1.5 Future Cost Estimation (FCE)

As better translations may appear to be poor translations at the beginning of stack decoding, they are often pruned out when using Top-N and beam width. To better account for this, Future Cost Estimation was used to measure how expensive it would be to translate the rest of a sentence based off of the current hypothesis. By ordering the stacks by their FCE scores, the better hypotheses are ideally maintained.

Spans are created from left to right in a sentence and the highest translation model (TM) and language model (LM) scores are used to estimate the future costs. The FCE of the spans in a sentence are pre-computed and used to calculate the FCE of each hypothesis.

$$adjusted\,score = future\,logprob + distortion\,penalty + local\,logprob - hypothesis_{logprob}$$

The hypothesis adjusted score is also be added to the local adjusted score.

$$adjusted\,score \mathrel{+}= hypothesis_{adjusted\,score} - hypothesis_{future\,logprob}$$

2.1.6 Multiple Features and Weighting

The decoder's model was modified to accept more than one feature. Additionally, feature scores can be weighted by providing a weight file to the decoder.

$$\textbf{features} = \textbf{feature list} \cdot \textbf{weight list}$$

$$model\,score = \sum_{i=1}^{features} feature_i$$

2.2 Reranker

Features are dynamically added to the training data during runtime and include the estimated number of untranslated words in a translation and the difference in model score rank to bleu score rank.

2.2.1 Estimated Number of Untranslated Words

A set is created composed of the words in the source data. The source set is then intersected with the words in the translation candidate sentence and the count of these words is used as the score for the feature.

$$untranslated = -1.0 \times (|Source\,Sentence \cap Translation\,Candidate| + 1)$$

2.2.2 Bleu Rank vs. Model Rank

A model score was calculated by summing over the features of a translation candidate. All of the translation candidates for a sentence were then ordered by their bleu score and the order of model scores compared. This was done in order to penalize candidates that have a large distance in terms of model rank and bleu rank. It was assumed that better candidates have a smaller difference in model and bleu rank.

Algorithm 2 Bleu Rank vs. Model Rank

```
function ASSIGN_RANK_SCORES(nbests)
    for nbest in nbests do
        bleu_sorted ← sort by descending smoothed bleu score in nbest
        lm_sorted ← sort by descending model score in nbest
        for each index i in bleu_sorted do
            if bleu_sorted[i] == lm_sorted[i] then
                bleu_sorted[i].features ← bleu_sorted[i].features + [−1.0]
            else
                adjusted ← −1.0 × (abs((i + 1) − (index of lm_sorted[i] in bleu_sorted + 1)) + 1)
                bleu_sorted[i].features ← bleu_sorted[i].features + [adjusted]
    return nbests
```

3 Data

3.1 Toy Data

The toy dataset was used to adapt the decoder to work with multiple features and weighting.

	File
TM	/toy/phrase-table/phrase_table.out
LM	/lm/en.tiny.3g.arpa
Input	/toy/train.cn
Reference	/toy/train.en

3.2 Dev Data

The dev dataset was used to generate an n-best list and for training the reranker perceptron. The reference files were used to calculate bleu scores. The reference file was used to calculate bleu scores. The input file was split evenly, by line count, into four files for use by the decoder. These files are stored in the inputs folder.

	File
TM	/large/phrase-table/dev-filtered/rules_cnt.final.out
LM	/lm/en.gigaword.3g.filtered.train_dev_test.arpa.gz
Input	/dev/all.cn-en.cn
Reference	/dev/all.cn-en.en0, /dev/all.cn-en.en1, /dev/all.cn-en.en2, /dev/all.cn-en.en3

3.3 Test Data

The test dataset was used to generate a decoding from Chinese to English. The reference files were used to calculate bleu scores. The input file was split evenly, by line count, into four files for use by the decoder. These files are stored in the inputs folder.

	File
TM	/large/phrase-table/dev-filtered/rules_cnt.final.out
LM	/lm/en.gigaword.3g.filtered.train_dev_test.arpa.gz
Input	/test/all.cn-en.cn
Reference	/test/all.cn-en.en0, /test/all.cn-en.en1, /test/all.cn-en.en2, /test/all.cn-en.en3

4 Code

All of the code used in the final project was from homework 4 and 5.

Homework 4	Homework 5
models.py	bleu.py
decoder.py	score-reranker.py
	get-weights.py

Several shell scripts were written to generate decodings, n-best lists, weights, and scores. These include and should be run in this order: *nbest.sh*, *rerank.sh*, and *score.sh*.

5 Experimental Setup

5.1 Setup: Feedback Loop

1. Split the input file into $2^n, n \geq 0$, files to reduce decoder runtime.
2. Generate an n-best list using the input files. Join the n-best lists into a single file.
3. Generate weights using the joined n-best list.
4. Run the decoder using the weights and split input files. Join the outputs.
5. Run the previous output through the score program.

5.2 Setup: Decoder Only

1. Split the input file into $2^n, n \geq 0$, files to reduce decoder runtime.
2. Run the decoder using the default weights and split input files. Join the outputs.
3. Run the previous output through the score program.

5.3 Evaluation

1. The bleu score was used to evaluate the machine translation system.
2. The occurrences of n-grams of size 1 to 4 were counted and sentences that were shorter than the reference sentence penalized.
3. Four references were used instead of one to improve the n-gram precision.
4. Different parameters such as distortion limit and number of translations per phrase were used and their scores compared.

6 Results

6.1 Dev Data

When evaluating over a single reference, */dev/all.cn-en.en0* was used.

6.1.1 First 100 Sentences, No Reranking

Max. Stack Size	Distortion Limit	Max. Translations	Bleu Score (4 refs)
100	1	10	0.16673775
100	3	10	0.148131408563

6.1.2 First 100 Sentences, No Reranking

Max. Stack Size	Distortion Limit	Max. Translations	Bleu Score (4 refs)	Bleu Score (1 ref)
100	1	5	0.16751394	0.110099834726
100	2	5	0.15719862	0.10400971324
100	3	5	0.15798428	0.104612493422
100	4	5	0.15312168	0.10046669526
100	5	5	0.15390518	0.0991379823101
100	6	5	0.14549784	0.0935341263432

6.1.3 Over All Sentences, No Reranking

Max. Stack Size	Distortion Limit	Max. Translations	Bleu Score (4 refs)	Bleu Score (1 ref)
100	1	5	0.13619778	0.0800299414496
100	3	5	0.12116937	0.0713563447896
100	4	5	0.11855773	0.0693950240411
100	5	5	0.11844958	0.0690408722579

6.2 Test Data

The stack size used for all tests was 100. When evaluating over a single reference, */test/all.cn-en.en0* was used.

6.2.1 First 100 Sentences, Reranking

Reranking generated bleu scores in the range of 0.08326205 to 0.12005043 over 4 references when distortion limit, d, is 1 and translations per sentence, k, is 5.

6.2.2 Over All Sentences, No Reranking

Max. Stack Size	Distortion Limit	Max. Translations	Bleu Score (4 refs)	Bleu Score (1 ref)
100	1	5	0.12870545	0.0762740411297
100	3	5	0.12077389	0.070930367351
100	5	5	0.12009749	0.0714391721834

7 Analysis of the Results

7.1 Observations

1. Scoring over a greater number of sentences gives a lower score. This may be due to longer sentences being penalized or the possibility of more unknown words.

2. When d, distortion limit, is greater than 1, scores received seem to be trending downwards. This is unexpected behaviour as if we allow words to swap at a bigger distance, it should improve Chinese to English accuracy. Knowing that the sentence structure for Chinese and English are generally reversed, we tried to remove the penalty for distortion but ended up still getting a lower score compared to when d is set to 1. This may be due to a bug in our code or the way we are evaluating the test data.

3. Reranking appears to lower bleu scores using our current implementation and data.

7.2 Improvements over Baseline

As the baseline score was around 0.06 bleu, the use of FCE, stack decoding, and distortion limits increased the bleu score to 0.1287 when d is set to 1.

8 Future Work

1. Investigate issues in the distortion limit code or related.

2. Tweak the ratio of TM and LM scores.

3. Use our own Chinese word segmenter.

4. Use of other weighting heuristics would likely help to improve the overall runtime and translation scores of this decoder.

5. Use of additional features, perhaps involving the number of words or IBM Model 1 alignments, may help to improve the reranking score. These along with the process of Minimum Bayes Risk and Ordinal Regression should be explored.

9 Usage

NOTE: Some of the paths used in the shell scripts may need to be changed to function correctly.

9.1 Distributed

To generate an n-best list using input chunk file 01 and an index offset of 482:

```
./nbest.sh 01 482
```

To generate translations using input chunk file 02:

```
./fast.sh 02
```

To generate 10 weights from an n-best file and store them in weights folder 3:

```
./rerank.sh 3 /path/to/some.nbest 10
```

To decode using the weights from weight folder 3:

```
./score.sh 3
```

9.2 Single Computer

To view all of the available options:

```
python decoder.py -h
python get-weights.py -h
python score-reranker.py -h
```

To generate a n-best list, n is 100 in this example, in one pass:

```
python decoder.py -s 100 -v -a
```

To decode with with a maximum of 100 hypotheses per stack, a distortion limit of 10, and limit the number of translations to 20:

```
python decoder.py -s 100 -d 10 -k 20 -v
```

To specify your own input file, translation model, and language model:

```
python decoder.py -i /path/to/input.file -t /path/to/tm -l /path/to/lm
```

To specify your own weights file:

```
python decoder.py -w /path/to/weights.file
```

To generate weights using the default values and an n-best file *data/train.nbest*, target language file *data/train.en*, and source language file *data/train.fr*:

```
python get-weights.py -n data/train.nbest -e data/train.en -f data/train.fr
```

To generate weights using the default training data, generate 5,000 random samples while using only 100 of them, set the cutoff for the sampler to 0.21, over 5 perceptron iterations with a learning rate of 0.1:

```
python get-weights.py -t 5000 -a 0.21 -x 100 -r 0.1 -i 5
```

To score output file of the first 100 sentences, not an n-best list, from the decoder:

```
python score-reranker.py -r /path/to/reference.file -n 100 < /path/to/output.file
```

To score the an output file, not an n-best list, from the decoder:

```
python score-reranker.py -r /path/to/reference.file < /path/to/output.file
```