eppex: Epochal Phrase Table Extraction for Statistical Machine Translation

Česlav Przywara, Ondřej Bojar
Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University in Prague

Outline

- Intro + motivation
- Implementation
  - approximate frequency counting
- Experiments
- Conclusions and future work

Phrase table construction

- Input: parallel corpus + word alignments + phrase extraction algorithm (symmetrisation heuristics)
- Output: phrase table

```
epochal extraction ||| epochální extrakce |||
p(f|e) lex(f|e) p(e|f) lex(e|f) ...
```

  - direct and inverse translation probabilities
    - `p(e|f) = C(e,f) / C(f)`
    - `p(f|e) = C(e,f) / C(e)`
  - lexical weights
    - `lex(f|e), lex(e|f)`
  - ...

Phrase table construction in Moses

- Substeps of steps 5 and 6 of **train-model.perl**
  - **phrase extraction** – produces direct and reverse phrase table halves (with word alignments, no scores yet)
  - **gzipping**, **sorting** and **scoring** of the **direct** table
  - **gzipping**, **sorting** and **scoring** of the **reverse** table
  - **sorting** of the scored reverse table
  - **consolidation** of the scored direct and reverse tables
  - **gzipping** of the consolidated phrase table
- Optional post-processing:
  - **significance filtering**

Motivation

- phrase table construction is time consuming
  - temporary data are read/written to disk
  - phrase tables size ~ usually several GB or even more
- phrase table quality is not strictly determined by its size
  - *significance filtering* – Johnson et al. (2007)
- more and more physical memory is available
  - laptops ~ 4 GB
  - computational clusters ~ 16 GB (and more) per node

From motivation to implementation

- Our inspiration:
  - Goyal et al. (2009) used approximate frequency counting for Language Modeling
- Our current status:
  - extraction of phrase pairs with on the fly filtration implemented via Lossy Counting
- Our ultimate goal:
  - in-memory phrase table construction (with on-the-fly filtration)

Lossy Counting algorithm (1)

- Manku and Motwani (2002)
  - approximate frequency counts over stream of data
- user defines two parameters: *error* ε and *support* s (such that $\varepsilon << s$)
- algorithm guarantees (N = number of instances):
  - all items whose true frequency exceeds sN are output
  - no item whose true frequency is less than $(s-\varepsilon)N$ is output
  - estimated frequencies are less than the true frequencies by at most εN
  - the space used by the algorithm is $O(1/\varepsilon \times log(\varepsilon N))$

Lossy Counting algorithm (2)

- input data ~ *stream* of items conceptually divided into *epochs* of size $w = \lceil 1/\varepsilon \rceil$
  - *T* – current epoch ID
- internally maintains database *D* of triples (*e*, *f*, *Δ*)
  - *e* – element, *f* – est. frequency, *Δ* – max. error
- new item *e* arrives
  - if *e* in *D*: increment *f* by one
  - otherwise: insert new triple (*e*, *1*, *T-1*)
- pruning at the end of each epoch ($N \equiv 0 \bmod w$)
  - remove all triples where $f + \Delta \le T$

Lossy Counting algorithm (3)

- At any time the Lossy Counting algorithm can be asked to produce a list of elements with $f \geq (s - \varepsilon)N$
  - such elements satisfies the aforementioned guarantees
  - in practice an alternative is also to output all items that survived the pruning so far

eppex implementation

- drop-in alternative to *extract* component from *phrase-extract* toolkit
  - fully compatible input/output format
- written in C++
  - strings stored as C-strings in memory pools (*Boost library*)
  - internally all strings represented by **4-byte** integers
  - Lossy Counting implemented as generic template
- comes with *counter* utility

Usage

Syntax:

```
eppex tgt src align extract \
lossy-counter [lossy-counter-2 [lossy-counter-3 […]]] \
[orientation [--model [wbe|phrase|hier]-[msd|mslr|mono]]]
```

Lossy Counter specification:

- *phrase-pair-length:error:support*

```
1:0:0 2-4:2e-7:8e-7
```

- no pruning of phrase pairs of length 1
- phrase pairs of length 2-4 stored by one LC with **$\varepsilon = 2\times10^{-7}$** and **$s = 8\times10^{-7}$**

```
1:0:0 2:2e-7:8e-7 3:2e-7:8e-7 4:2e-7:8e-7
```

- similar as above, but phrase pairs of length 2-4 stored in **separate** counters

Usage (in Moses)

- train-model.perl
  - `--eppex="1:0:0 2-4:2e-7:8e-7"`
- experiment.perl (EMS)
  - config: [TRAINING] > training-options

Experiments enviroment

- All experiments run on the same machine
  - 64-bit Ubuntu 10.04 server edition
  - 2 Core4 AMD Opteron 2.8 GHz processors
  - 32 GB RAM
  - all input and output files read from and written to a locally mounted disk

Experiments - dataset

- Training data: **CzEng** corpus with a few additions
  - 8.4M sentence pairs
  - 107.2M English and 93.2M Czech tokens
  - exact setup: Mareček et al. (2011), system ”cu-bojar”
- Tuning and testing data: WMT 2011 Translation Task

Experiments – scenarios

- baseline (default approach)
- baseline + sigfilter
  - -l a-e → all *1-1-1 phrase pairs* kept in
  - -l a+e → all *1-1-1 phrase pairs* removed
  - -n 30 → top *n* pairs kept (sorted by *forward probability*)
- eppex **1-in**
  - all phrase pairs of length 1–3 kept in
- eppex **1-out**
  - all single-occurring phrase pairs removed

Experiments – BLEU scores

Experiment	Number of phr. pairs	Gzipped file size	BLEU on wmt10	BLEU on wmt11
baseline	153.6 M	3.68 GB	17.36	**18.22**
sigfilter 30	137.0 M	3.36 GB	17.48	18.13
sigfilter a-e	92.4 M	2.39 GB	17.23	17.87
eppex 1-in	57.1 M	1.28 GB	**17.60**	18.10
sigfilter a+e	35.0 M	0.86 GB	17.31	17.99
eppex 1-out	14.4 M	0.33 GB	17.23	17.94

Experiments – wallclock time

Step	baseline	eppex 1-in	eppex 1-out
phr-ext	**1152**	**4360**	**4361**
gzip	1303	502	246
sort	5101	1632	1131
score	20417	7433	712
sort-inv	1569	129	22
cons	1361	269	66
pt-gzip	881	259	65
TOTAL (hh:mm:ss)	31784 **8:49:44**	14584 **4:03:04**	6603 **1:50:03**

Experiments – sigfilter wallclock time

	-l a+e	-l a-e	-n 30
baseline		31784	
sigfiltering	18248	18449	1141
TOTAL (hh:mm:ss)	50032 **13:53:52**	50233 **13:57:13**	32925 **9:08:45**

Experiments – RAM usage

Experiment	VM peak	in step
baseline	1.1 GB	scoring-e2f
sigfilter 30	1.1 GB	scoring-e2f
sigfilter a-e	5.4 GB	sigfilter
eppex 1-in	19.2 GB	phr-ext
sigfilter a+e	5.4 GB	sigfilter
eppex 1-out	16.7 GB	phr-ext

Old vs. new scorer – wallclock time

Step	Baseline (old)	Baseline (new)
phr-ext	1152	1272
gzip	1303	1354
sort	5101	4599
score	**20417**	**7470**
sort-inv	1569	1383
cons	1361	1419
pt-gzip	881	849
TOTAL (hh:mm:ss)	31784 **8:49:44**	18346 **5:05:46**

Conclusions

- bulk of phrase pairs to be scored can be significantly reduced
  - 3.68 GB → 1.28 GB
- translation quality can be preserved (BLEU)
  - wmt10: 17.36 → 17.60
  - wmt11: 18.22 → 18.10
- significant RAM requirements
  - 1.1 GB → 19.2 GB
  - not for laptop use...

Future work

- futher optimization of memory usage
- integration with *memscore* – Hardmeier (2010)
- confrontation with larger corpora (Fr-En)
- (*Ondřej would like me to*)
  - compare eppex and suffix arrays approach used for incremental training

Bibliography

- Goyal, Amit, Hal Daumé, III, and Suresh Venkatasubramanian. *Streaming for large scale NLP: language modeling.* In Proc. of HTL/NAACL, pages 512–520, Boulder, Colorado, 2009.
- Hardmeier, Christian. *Fast and Extensible Phrase Scoring for Statistical Machine Translation.*The Prague Bulletin of Mathematical Linguistics, 93:79–88, 2010.
- Johnson, J Howard, Joel Martin, George Foster, and Roland Kuhn. *Improving Translation Quality by Discarding Most of the Phrasetable.* In Proc. of EMNLP and Computational Natural Language Learning, 2007.
- Manku, Gurmeet Singh and Rajeev Motwani. *Approximate Frequency Counts over Data Streams.* In Proc. of the 28th International Conference on Very Large Data Bases, 2002.
- Mareček, David, Rudolf Rosa, Petra Galuščáková, and Ondřej Bojar. *Two-step translation with grammatical post-processing.* In Proc. of WMT, Edinburgh, UK, July 2011.

Questions?

- Any comments and suggestions are appreciated!

  - ceslav@przywara.cz
  - bojar@ufal.mff.cuni.cz
  - moses-support@mit.edu
