Experimenting in MT: Moses Toolkit and Eman

Aleš Tamchyna, Ondřej Bojar
Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University, Prague

Tue Sept 8, 2015
Outline

► Quick overview of Moses.
► Bird’s eye view of (phrase-based) MT.
  ► With pointers to Moses repository.
► Experiment management.
  ► Motivation.
  ► Overview of Eman.
► Run your own experiments.
  ► Introduce Eman’s features through building a baseline Czech→English MT system.
  ► Inspect the pipeline and created models.
  ► Try some techniques to improve over the baseline.
Moses Toolkit

- Comprehensive open-source toolkit for SMT
- Core: phrase-based and syntactic decoder
Moses Toolkit

- Comprehensive open-source toolkit for SMT
- Core: phrase-based and syntactic decoder
- Includes many related tools:
  - Data pre-processing:
    cleaning, sentence splitting, tokenization, ... 
  - Building models for translation:
    create phrase/rule tables from word-aligned data,
    train language models with KenLM
  - Tuning translation systems (MERT and others)
Moses Toolkit

- Comprehensive open-source toolkit for SMT
- Core: phrase-based and syntactic decoder
- Includes many related tools:
  - Data pre-processing: cleaning, sentence splitting, tokenization, ...
  - Building models for translation: create phrase/rule tables from word-aligned data, train language models with KenLM
  - Tuning translation systems (MERT and others)
- You still need a tool for word alignment:
  - GIZA++, fast_align, ...
- Bundled with its own experiment manager EMS
  - We will use a different one.
Bird’s Eye View of Phrase-Based MT

- Monolingual
- Parallel
- Devset
- Input

Preprocessing: tokenization, tagging...
Bird’s Eye View of Phrase-Based MT

Preprocessing: tokenization, tagging...

- Word alignment
- Phrase extraction

Language Model (LM)

Translation M. (TM)

Reordering M. (RM)

Monolingual  Parallel  Devset  Input
Bird’s Eye View of Phrase-Based MT

Monolingual → Parallel → Devset → Input →

Preprocessing: tokenization, tagging...

Language Model (LM) →

Word alignment → Phrase extraction

Translation M. (TM) → Reordering M. (RM)

Basic model
Bird’s Eye View of Phrase-Based MT

- Monolingual
- Parallel
- Devset
- Input

Preprocessing: tokenization, tagging...

- Word alignment
- Phrase extraction

Language Model (LM)

- Translation M. (TM)
- Reordering M. (RM)

Basic model

Parameter optimization (MERT)

Optimized model
Bird’s Eye View of Phrase-Based MT

Monolingual  Parallel  Devset  Input

Preprocessing: tokenization, tagging...

Word alignment

Phrase extraction

Language Model (LM)

Translation M. (TM)

Reordering M. (RM)

Basic model

Parameter optimization (MERT)

Optimized model  Translate
Bird’s Eye View of Phrase-Based MT

Monolingual  Parallel  Devset  Input

Preprocessing: tokenization, tagging...

Word alignment

Phrase extraction

Language Model (LM)

Translation M. (TM)

Reordering M. (RM)

Basic model

Parameter optimization (MERT)

Optimized model

Translate

train-model.perl
mert-moses.pl
moses-parallel.pl
Bird’s Eye View of Phrase-Based MT

Monolingual → Parallel → Devset → Input

Preprocessing: tokenization, tagging...

Word alignment

Phrase extraction

Language Model (LM)

Translation M. (TM)

Reordering M. (RM)

Parameter optimization (MERT)

Translate
Now, This Complex World...

Parameter optimization (MERT)  
Optimized model  
Basic model  
Language Model (LM)  
Word alignment  
Phrase extraction  
Translation M. (TM)  
Reordering M. (RM)  
Preprocessing: tokenization, tagging...  
Parallel  
Monolingual  
Devset  
Input  
Translate
Has to Be Ruled by Someone
...Has to Be Ruled by Someone

Monolingual	Parallel	Devset	Input

Preprocessing: tokenization, tagging...

Parameter optimization (MERT)
Optimized model
Basic model
Language
Model (LM)
Translation M. (TM)
Reordering M. (RM)

Word alignment
Phrase extraction

Parallel
Monolingual
Devset
Input

Ducttape
EMS
M4M

Translate
...Has to Be Ruled by Someone
Motivation for Experiment Mgmt (1/2)

Research needs reproducibility.

- Console-based environment alone helps a lot:
  - Bash history of past commands.
  - Log files.
- Complications:
  - Experiments carried out in parallel.
    Experiments can take days.
    $\Rightarrow$ Easy to lose track.
  - Should reuse large intermediate files.
  - Different versions of the research software.
    (Both daily updates as well as yearly updates.)
Motivation for Experiment Mgmt (2/2)

Research is search.
(for the best procedure, the best configuration, . . . )

You can think of research in AI/machine-learning terms.

- **Heuristics:**
  - Run quick probes (small data) first, then replicate on full.
- **Beam Search:** Increase your beam size:
  - Run ~10 variations of each experiment.
- **Genetic Algorithms:**
  - Clone and modify most successful experiments.
- ("The best" varies based on the metric chosen.)
  - So look at more metrics at once.
Features of Eman

- Console-based ⇒ easily scriptable (e.g. in bash).
- Versatile: “seeds” are up to the user, any language.
- Support for the manual search through the space of experiment configurations.
- Support for finding and marking (“tagging”) steps or experiments of interest.
- Support for organizing the results in 2D tables.
- Integrated with SGE
  ⇒ easy to run on common academic clusters.

**eman --man** will tell you some details.
**http://ufal.mff.cuni.cz/eman/** has more.
Experiments consist of processing STEPS.

Steps are:

- of a given type, e.g. align, tm, lm, mert,
- defined by immutable variables, e.g. ALISYM=gdfa,
- all located in one directory, the “playground”,
- timestamped unique directories, e.g. s.mert.a123.20120215-1632
- self-contained in the dir as much as reasonable.
- dependent on other steps, e.g. first align, then build tm, then mert.

Lifetime of a step:
Why INITED $\rightarrow$ PREPARED $\rightarrow$ RUNNING?

The call to **eman init seed**:
- Should be quick, it is used interactively.
- Should **only** check and set vars, “turn a blank directory into a valid eman step”.

The call to **eman prepare s.step.123.20120215**:
- May check for various input files.
  - Less useful with heavy experiments where even corpus preparation needs cluster.
- Has to produce **eman.command**.
  $\Rightarrow$ A chance to check it: are all file paths correct etc.?

The call to **eman start s.step.123.20120215**:
- Sends the job to the cluster.
Our Eman Seeds for MT

- Monolingual
- Parallel
- Devset
- Input

Preprocessing: tokenization, tagging...

- Word alignment
- Phrase extraction

Language Model (LM)

- Translation M. (TM)
- Reordering M. (RM)

Basic model

Parameter optimization (MERT)

Optimized model

Translate
Our Eman Seeds for MT

Parameter optimization (MERT)
Optimized model
Basic model
Language Model (LM)
Translation M. (TM)
Reordering M. (RM)

Word alignment
Phrase extraction

Monolingual corpus
Parallel corpus
Devset corpus
Input corpus

Preprocessing: tokenization, tagging...

Corpus
Corpus
Corpus
Corpus

Translate
Our Eman Seeds for MT

Monolingual corpus
Translation M. (TM)
Reordering M. (RM)
Language Model (LM)
Preprocessing: tokenization, tagging...

Phrase extraction
Word alignment

devset
input

parallel

monolingual

Input corpus

Translate
Parameter optimization (MERT)
Optimized model
Basic model
corpman

corpus
corpus
corpus
corpus
corpman
Eman’s Bells and Whistles

Experiment management:

- `ls`, `vars`, `stat` for simple listing,
- `select` for finding steps,
- `traceback` for full info on experiments,
- `redo` failed experiments,
- `clone` individual steps as well as whole experiments.

Meta-information on steps:

- `status`,
- `tags`, autotags,
- `collecting` results,
- `tabulate` for putting results into 2D tables.
Whole Experiment = eman traceback

eman traceback s.evaluator.8102edfc.20120207-1611

|-- s.evaluator.8102edfc.20120207-1611
|  |-- s.mosesgiza.b6073a00.20120202-0037
|  |-- s.translate.b17f203d.20120207-1604
|  |  |-- s.mert.272f2f67.20120207-0013
|  |  |  |-- s.model.3e28def7.20120207-0013
|  |  |  |-- s.lm.608df574.20120207-0004
|  |  |  |  |-- s.srilm.117f0cfe.20120202-0037
|  |  |  |-- s.mosesgiza.b6073a00.20120202-0037
|  |  |  |-- s.tm.527c9342.20120207-0012
|  |  |  |  |-- s.align.dec45f74.20120206-0111
|  |  |  |  |-- s.mosesgiza.b6073a00.20120202-0037
|  |  |  |  |-- s.mosesgiza.b6073a00.20120202-0037
|  |  |  |  |-- s.mosesgiza.b6073a00.20120202-0037
|  |  |  |-- s.mosesgiza.b6073a00.20120202-0037

Options: --vars --stat --log ... --ignore=steptype
Finding Steps: `eman select`

- Step dirs don’t have nice names.
- You need to locate steps of given properties.

What language models do I have?

- `eman ls lm`
- `eman select t lm`

If we need just the finished ones:

- `eman stat lm | grep DONE`
- `eman select t lm d`

And just 5-gram ones for English:

- `eman select t lm d vre ORDER=5 vre CORPAUG=en`
Deriving Experiments using `clone`

The text form of traceback allows to tweak the experiment:

- `eman tb step | sed 's/cs/de/' | eman clone`
  replicates our experiment on German instead of Czech.

The regex substitution is available in `eman` itself:

- `eman tb step -s '/cs/de/' -s '/form/lc/'`
  shows the traceback with the substitutions highlighted.
  - A good chance to check if the derivation does the intended.

- `eman tb step -s '/cs/de/' -s '/form/lc/' \`
  | eman clone --dry-run`
  - Last chance to check if existing steps get reused and what vars will new steps be based on.
  - Drop `--dry-run` to actually init the new steps.
  - Add `--start` if you’re feeling lucky.
Hacking Welcome

Eman is designed to be hacking-friendly:

▶ Self-contained steps are easy to inspect:
  ▶ all logs are there,
  ▶ all (or most of) input files are there,
  ▶ the main code (**eman.command**) is there,
  ▶ often, even the binaries are there, or at least clearly identifiable.

▶ Step halfway failed?
  ⇒ Hack its **eman.command** and use **eman continue**.

▶ Seed not quite fit for your current needs?
  ⇒ Just init the step and hack **eman.seed**.
  ⇒ Or also prepare and hack **eman.command**.

Always mark manually tweaked steps, e.g. using eman’s tags.
Fit for Cell-Phone SSH 😊

- Experiments run long but fail often.
- You don’t want to be chained to a computer.

Most eman commands have a short nickname.

- How are my last 10 merts?
  
  ```bash
  eman sel t mert l 10 --stat
  ```

Specify steps using any part of their name/hash or result:

- s.foobar.a0f3b123.20120215-1011 failed, retry it:
  
  ```bash
  eman redo a0f3 --start
  ```

- How did I achieve this great BLEU score of 25.10?
  
  ```bash
  eman tb 25.10 --vars | less
  ```
Fit for Team Work

Playgrounds can be effectively merged:

- `eman add-remote /home/fred/playground freds-exps`
- You can re-interpret Fred’s results.
- You can clone Fred’s experiments.
- You can make your steps depend on Fred’s steps.
  - Only a shared file system is needed.

Caveat: we don’t bother checking for conflicts yet.
Summary

Hopefully, you now understand:

▶ within (PB)MT:
  ▶ the structure of a (PB)MT experiment,
  ▶ what is the language model and the translation model,
▶ meta-level:
  ▶ eman’s organization of the experimentation playground,
  ▶ the idea of cloning of experiments.

Now the exercise comes…
Extra Slides
Eman is Versatile

What types of steps should I have?
   ▶ Any, depending on your application.

What language do I write steps in?
   ▶ Any, e.g. bash.

What are the input and output files of the steps?
   ▶ Any, just make depending steps understand each other.
   ▶ Steps can have many output files and serve as prerequisites to different types of other steps.

What are measured values of my experiments?
   ▶ Anything from any of the files any step produces.
What the User Implements: Just Seeds

Technically, a seed is any program that:

- responds to arbitrary environment variables,
- runs `eman defvar` to register step variables with `eman`,
- produces another program, `.eman.command` that does the real job.

The seed is actually run twice:

- At “init”: to check validity of input variables and register them with `eman`.
- At “prepare”: to produce `eman.command`.

The user puts all seeds in `playground/eman.seeds`.

- Eman runs a local copy of the seed in a fresh step dir.
eman redo

On cluster, jobs can fail nondeterminically.

- Bad luck when scheduled to a swamped machine.
- Bad estimate of hard resource limits (RAM exceeds the limit $\Rightarrow$ job killed).

Eman to the rescue:

- **eman redo step** creates a new instance of each failed step, preserving the experiment structure.
- **eman redo step --start** starts the steps right away.

To make sure eman will do what you expect, first try:

- **eman redo step --dry-run**
Cloning is initiating a new step using vars of an existing one. Cloning of individual steps is useful:

- when a step failed (used in `eman redo`),
- when the seed has changed,
- when we want to redefine some vars:
  ```
  ORDER=4 eman clone s.lm.1d6f791c...
  ```

Cloning of whole tracebacks:

- The text of a traceback gets instantiated as steps.
- Existing steps are reused if OK and with identical vars.

- `eman traceback step | eman clone`
- `eman traceback step | mail bojar@ufal`
  followed by `eman clone < the-received-mail`. 
eman tag or eman ls --tag shows tags

**Tags and autotags are:**
- arbitrary keywords assigned to individual steps,
- inherited from dependencies.

**Tags are:**
- added using `eman add-tag the-tag steps`,
- stored in `s.stepdir.123/eman.tag`.
  ⇒ Use them to manually mark exceptions.

**Autotags are:**
- specified in `playground/eman.autotags` as regexes over step vars, e.g.: `/ORDER=(.*)/$1gr/` for LM,
- (re-)observed at `eman retag`.
  ⇒ Use them to systematically mark experiment branches.
eman collect

Based on rules in **eman.results.conf**, e.g.:

```
BLEU */BLEU.opt
Snts s.eval*/corpus.translation
```

**eman collects results from all steps into** **eman.results**:

<table>
<thead>
<tr>
<th>#</th>
<th>Step Name</th>
<th>Status</th>
<th>Score</th>
<th>Value</th>
<th>Tags and Autotags</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.evaluator.11ccf590.20120208-1554</td>
<td>DONE</td>
<td>TER</td>
<td>31.04</td>
<td>5gr</td>
<td>DEVwmt10 LMc-news towards-</td>
</tr>
<tr>
<td>s.evaluator.11ccf590.20120208-1554</td>
<td>DONE</td>
<td>PER</td>
<td>44.61</td>
<td>5gr</td>
<td>DEVwmt10 LMc-news towards-</td>
</tr>
<tr>
<td>s.evaluator.11ccf590.20120208-1554</td>
<td>DONE</td>
<td>CDER</td>
<td>33.97</td>
<td>5gr</td>
<td>DEVwmt10 LMc-news towards-</td>
</tr>
<tr>
<td>s.evaluator.11ccf590.20120208-1554</td>
<td>DONE</td>
<td>BLEU</td>
<td>12.28</td>
<td>5gr</td>
<td>DEVwmt10 LMc-news towards-</td>
</tr>
<tr>
<td>s.evaluator.11ccf590.20120208-1554</td>
<td>DONE</td>
<td>Snts</td>
<td>3003</td>
<td>5gr</td>
<td>DEVwmt10 LMc-news towards-</td>
</tr>
<tr>
<td>s.evaluator.29fa5679.20120207-1357</td>
<td>OUTDATED</td>
<td>TER</td>
<td>17.66</td>
<td>5gr</td>
<td>DEVwmt10 LMc-news</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>s.evaluator.473687bb.20120214-1509</td>
<td>FAILED</td>
<td>Snts</td>
<td>3003</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Perhaps hard to read.
- Easy to grep, sort, whatever, or **tabulate**.
eman tabulate to Organize Results

The user specifies in the file `eman.tabulate`:

- which results to ignore, which to select,
- which tags contribute to col labels, e.g. **TER, BLEU,**
- which tags contribute to row labels, e.g. `[0-9]gr, towards-[A-Z]+, PRO**.

Eman tabulates the results, output in `eman.niceresults`:

<table>
<thead>
<tr>
<th></th>
<th>PER</th>
<th>CDER</th>
<th>TER</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>5gr</code></td>
<td>towards-CDER</td>
<td>44.61</td>
<td>33.97</td>
<td>31.04</td>
</tr>
<tr>
<td><code>5gr</code></td>
<td></td>
<td>44.19</td>
<td>33.76</td>
<td>31.02</td>
</tr>
<tr>
<td><code>5gr</code></td>
<td>PRO</td>
<td></td>
<td>43.91</td>
<td>33.87</td>
</tr>
<tr>
<td><code>5gr</code></td>
<td>towards-PER</td>
<td>44.44</td>
<td>33.52</td>
<td>30.74</td>
</tr>
</tbody>
</table>
Related Experiment Mgmt Systems

Eman is just one of many, consider also:

- **LoonyBin (Clark et al., 2010)**
  - Clickable Java tool.
  - Support for multiple clusters and scheduler types.

- **Moses EMS (Koehn, 2010)**
  - Experiment Management System primarily for Moses.
  - Centered around a single experiment which consists of steps.

- **Pure Makefiles**
  - Yes, you can easily live with fancy Makefiles.
    - You will use commands like `make init.mert`
    - or `cp -r exp.mert.1 exp.mert.1b`
    - You need to learn to use `$*`, `$@` etc.
    - You are likely to implement your own eman soon.

There are also the following workflow management systems: DAGMan, Pegasus, Dryad.
References
