Wild Experimenting in MT

Aleš Tamchyna, Jan Berka, Ondřej Bojar

February 17, 2012

1 Quick Preview of Addicter

Change to Addicter directory:

```
cd ~/../clara43/addicter
```

Add necessary modules to Perl path:

```
export PERL5LIB="`pwd`/../lib:$PERL5LIB"
```

Run Addicter server with some high port number:

```
./server.pl 9000
```

Copy the outputted link to web browser (Opera or Firefox, not Konqueror) and start using Addicter.

1.1 Warm-up exercises

1. Using Addicter, find the Tamil translation of the English word 'film'.
2. Find the meaning of the Tamil word 'watikai'. (Don't give up too early, try mentally running IBM Model 1.)
3. Can you find the Tamil translation of the English word 'railway'? Why / Why not?
4. We want our system to translate better. On what should we focus first (what is the biggest problem of current translation)?

2 eman

2.1 Getting eman

Check out from UFAL's repository:

```
cd   # go back to your home dir
svn co --username public --depth empty \
https://svn.ms.mff.cuni.cz/svn/statmt/trunk statmt
```

- Choose 'accept permanently' when asked about server certificate validation (i.e. press `p`).
- Password is 'public'.
- Ignore all the KWallet logs printed in your console.

- Press 'Cancel' in KWallet pop-up window when it appears.
- Type `yes` in the console when asked if the password should be stored unencrypted.

Get all the necessary subdirectories:

```
cd statmt
svn update playground scripts addicter perl src
svn update --set-depth empty projects
svn update projects/clara
```

2.2 Setting Up the Environment

Create the file `~/.bash_profile` that sources your `.bashrc`:

```
echo "source ~/.bashrc" >> ~/.bash_profile
```

Add eman to your PATH and set the PATH in your `.bashrc`):

```
export PATH="$PATH:$HOME/statmt/scripts"
echo "export PATH=\"\$PATH:$HOME/statmt/scripts\"" >> ~/.bashrc
```

Add paths to necessary Perl modules:

```
export PERL5LIB="$PERL5LIB:$HOME/../clara44/perl5/lib/perl5/"
echo "export PERL5LIB=\"\$PERL5LIB:$HOME/../clara44/perl5/lib/perl5/\"" >> ~/.bashrc
```

2.3 Seeds

An experiment is composed of steps, such as preparing the data, training a language model etc. Eman uses *seeds* to specify the step. Seeds are special bash scripts (in principle, any executable file would work but the current support is aimed at bash). Bash variables are used to pass arguments to them. By itself, eman does not have any seeds. However, it supposes that it will be run in a *playground* that contains the directory `eman.seeds`. Our playground has seeds that correspond to common tasks in SMT, such as `lm` creation or `mert` optimization.

3 Baseline Experiment

3.1 Required Tools

You will need Moses toolkit, GIZA++ and SRILM for your experiments. Use eman command `add-remote` to include an existing playground where these programs are already prepared.

```
cd ~/statmt/playground
eman add-remote ~/../clara44/statmt/playground binaries
eman reindex
```

In the first command, eman creates a symlink named `binaries` in your playground and registers it as one more playground, i.e. a place where step directories are. The second command scans all registered playgrounds and indexes their steps. You can now query these steps and use them in your experiments. The following command means "list steps of type `srilm` that are done". Full syntax is described in the manpage (use `eman --man` to view it).

```
eman select t srilm d
```

You can see the remote step in the output. The trick with remote playground saved you about 10 minutes of compilation.

3.2 Corpora

The SMT playground includes `corpman`, a tool for managing and deriving corpora. A corpus can contain several languages, each of them can have various factors (e.g. the lemma, morphological tag, or anything you wish to explicitly model). We are going to import English-Tamil parallel data into corpman as corpora called `en-ta-train`, `en-ta-dev` and `en-ta-test`, with languages `en` and `ta` and the single factor called `form` in each language to start with. The seed `corpus` can create a corpus from data you have somewhere on your disk. Try initializing it without any arguments to see its synopsis:

```
eman init corpus
```

eman will print a list of variables recognized by the seed and will complain that some mandatory arguments were not supplied.

All our data are in `statmt/projects/clara/english-tamil`. They can be imported as corpora using this simple script:

```
#!/bin/bash

for section in train dev test; do
  for lang in en ta; do
    OUTCORP=en-ta-$section OUTLANG=$lang OUTFACTS=form \
    OUTLINECOUNT=$(wc -l < ../projects/clara/english-tamil/$lang.$section) \
    TAKE_FROM_COMMAND="cat ../../projects/clara/english-tamil/$lang.$section" \
    eman init --start corpus
  done
done
```

Paste its code into a file (e.g. `import_corpora.sh`) in the playground, allow it to be executed and run it:

```
chmod +x import_corpora.sh
./import_corpora.sh
```

The script runs eman 6 times (3 corpora times 2 languages). In each iteration, a new step based on seed `corpus` is initialized by the `init` command and immediately executed (thanks to the `--start` option). All the required variables are provided in the script: the name of the corpus, the output language name, the output factors' names, the expected linecount (for a sanity check) and the Unix command that emits the actual data on its standard output.

Each time a step is initialized, eman creates a directory, e.g. `s.corpus.12a3456b.20120217-1400` with the following files:

- `eman.seed` – the seed `corpus` copied from `playground/eman.seeds/`
- `eman.vars` – the list of variables defined for the step
- `eman.status` – the state of the step
- `eman.command` – the actual code that will run when the step is started, this file is generated by the seed using the variables provided at init
- `eman.deps` – the list of steps that this step depends on
- `eman.tag` – the set of step tags, user defined or automatically inserted by the seed
- `eman.init_env` – the list of all variables defined when the step was initialized, mainly for debugging

Eman recognizes whether it runs on a cluster with Grid Engine (in which case it submits steps using `qsub`, assuring that dependent steps wait for each other), or locally (steps are run directly in the background on your machine and eman ensures subsequent steps will wait in a sleeping loop until the prerequisites finish).

You can use eman to access step data, using commands like `eman vars`, `eman deps` etc.

There are a couple of ways to check progress or status of an eman step. First, go to the step directory (alternatively, stay in `playground` and use step directory instead of '`.`' in the following commands).

```
cd s.corpus.12a3456b.20120217-1400
```

Current step status:

```
eman status .
```

Status of this step and all step it depends on:

```
eman traceback --status .
  # you can also use the abbreviated form: eman tb --stat .
```

Follow step logging output (on cluster, one more log file is created and used, `log.o12345`):

```
tail -f log
```

(Hit CTRL-C to exit `tail -f`.)

Given that we just created six steps, we might like to see the status of all of them:

```
eman select t corpus --status
```

3.3 Experiment Pipeline

Defining all steps by hand or writing a script for each experiment would be tedious. The preferred workflow in eman is to create a pipeline once and then use eman's features to make modifications to it and run multiple scenarios based on it.

The key is the command `eman clone`. It can create copies of individual steps or whole scenarios (somewhat modified, if you wish), assuring that existing steps are re-used wherever possible (based on values of their variables). We are going to clone a scenario that describes the baseline experiment, thus avoiding the need to manually create all the steps (`lm`, `align`, `tm`, `model`, `mert`, `translate`, `eval`). A scenario can be stored as a *traceback* generated by `eman tb --vars LASTSTEP`. Clone the baseline experiment:

```
cd ../   # go back to the playground directory
eman clone < ../projects/clara/en-ta.tb
```

Examine the command output. Notice that:

- The `corpus` steps we created were re-used.
- The remote steps were also re-used (`mosesgiza` and `srilm`).
- The full traceback with step statuses was printed.

You can now run the whole pipeline using:

```
eman start s.eval.... # the last step of the pipeline
                      # (It appears at the top of the traceback.)
```

You can monitor the progress:

```
eman tb --status s.eval....
```

Very soon, all steps before MERT will be finished and MERT will take some time to run. To check how many iterations were completed, you can simply look inside the step:

```
ls s.mert..../mert-tuning
```

You can also instruct eman to wait for the step to finish and send you an e-mail with the full traceback of statuses afterwards:

```
eman wait s.mert... \
; eman tb --stat s.mert... | mail -s "s.mert... finished" yourself@somewhere.edu
```

Once MERT finishes, evaluation data will be translated and translation quality measured using BLEU. You can view the horribly low result by looking in the file `BLEU` in the `s.eval....` step directory.

3.4 (Auto)tags

Tags are labels that describe a step. They are either automatically generated (autotags) or assigned manually.

eman generates autotags based on rules in the file `playground/eman.autotags`. No default rules are defined in our playground, but there is a prepared set of rules in the file "`eman.autotags-.sample-clara2012`". Copying it in `eman.autotags` will "activate" the rules:

```
cp eman.autotags.sample-clara2012 eman.autotags
eman reindex # implies re-tagging, i.e. applying the rules from eman.autotags
```

View the autotags of your `mert` steps:

```
eman sel t mert --tag
```

You can define your own rules in `eman.autotags`.

Assigning tags manually is useful for making notes about a particular step. For example, we label the `eval` step as "baseline":

```
eman add-tag baseline s.eval...
```

The command `eman add-tag` can take a while, because it is also reindexing in order to propagate both manual tags as well as autotags along the traceback. If you would like to check this out, add a tag to one of your `corpus` steps and which all steps will show it when you list all your steps and their tags:

```
eman tag   # this lists all steps and their tags, you can e.g. grep in this
```

4 Collecting Results

Checking results of experiments by manually viewing files inside steps is possible, but eman implements a more convenient way. It supports summarizing results based on rules (`eman collect`) and creating user-defined tables with experiment results (`eman tabulate`).

`eman collect` looks for rules in the file `eman.results.conf`. There is an example configuration file in the SMT playground that we are going to use:

```
cp eman.results.conf.sample eman.results.conf
eman collect
```

eman created a file `eman.results` with a summary of scores of all experiments. For now, the file is concise and readable, but if you create more scenarios, it will become hard to find a particular result. On the other hand, you can `grep` or `sort` your results easily in this format.

Also, `eman.results` serves as an additional "index" for eman – you can now specify steps by their results. If your BLEU score was e.g. 2.65, the following command would identify the step:

```
eman guess 2.65
```

and this command would show you all the details of the experiment:

```
eman tb --vars --log 2.65
```

`eman tabulate` also needs a configuration file. Run these commands to create file `eman.niceresults`:

```
cp eman.tabulate.sample-clara2012 eman.tabulate
eman tabulate
```

The configuration file `playground/eman.tabulate` can define any number of tables, each of them can be configured to give you a different insight into your experiments. In our example, the rows are defined by the order of the language model and the columns by word alignment factors (form, stem). This file will become more interesting as you progress with your experiments.

5 Word Alignment on Stems

Aligning on a different form requires only a slight change of the current experiment. Get the traceback of the existing scenario:

```
eman tb --vars s.eval... > baseline.traceback
```

Examine the traceback file in your favourite text editor and globally change the following variable values:

- `SRCALIAUG=en+form` to `SRCALIAUG=en+stem4`
- `TGTALIAUG=ta+form` to `TGTALIAUG=ta+stem4`
- `ALILABEL=en-form-ta-form` to `ALILABEL=en-stem4-ta-stem4`

Save the file as `align_stems.traceback`.

Examine the file `corpman.rules` to find out how the stems are created. corpman will use the matching rule and generate new `corpus` steps to generate the required factors.

Clone the modified traceback and start the experiment:

```
eman clone --start < align_stems.traceback
```

Running a cloned experiment right away is generally not a good idea – if the result of cloning is different from your expecations, you are left with a number of broken steps that you have to manually remove (after first finding them) and reindex both eman and corpman.

eman supports an option `--dry-run` that causes the cloning command to only simulate step initialization so that you can check the result. So the recommended procedure is:

```
eman clone --dry-run < align_stems.traceback
```

Check if the clone is at all possible, how many steps would be created etc. Only once you are happy, actually init the steps:

```
eman clone < align_stems.traceback
```

Now examine the initialized steps. If everything seems correct, run the cloned scenario:

```
eman start THE-TOPMOST-STEP
```

On the other hand, the modification and cloning can all actually be done on one line (if you are *really* sure or you don't mind cleaning up the rubbish of failed attempts):

```
eman tb s.eval... -s '/(ALIAUG=..)\+form/$1\+stem4/' \
        -s '/en-form-ta-form/en-stem4-ta-stem4/' | eman clone --start
```

You can add a tag to the new `eval` step or rely on the autotags to distinguish between the different experiments.

6 True Tamil Stems for Word Alignment

6.1 Prepare the Data

We are going to use Treex (indirectly) in this section.

It is assumed that Treex is present in your Perl modules path.

Go to the directory with the original data:

```
cd ~/statmt/projects/clara/english-tamil
```

Examine the script `../split.pl`. It is a simple wrapper script that calls a Treex tool for splitting Tamil affixes for each line in its input. Optionally, it can restore split Tamil to its original form.

Concatenate the Tamil files, pass them as input into the script and separate them again:

```
cat ta.train ta.dev ta.test | ../split.pl > ta_split
cat ta_split | sed "s/ +[^ ]*//g" > ta_stems
head -5000 ta_split > ta_split.train
tail -600 ta_split | head -300 > ta_split.dev
tail -300 ta_split > ta_split.test
head -5000 ta_stems > ta_stems.train
tail -600 ta_stems | head -300 > ta_stems.dev
tail -300 ta_stems > ta_stems.test
```

Notice that we created two sets of corpora:

- `ta_split` containing roots and suffixes (marked by '+').
- `ta_stems` containing only word roots.

We are going to use the first set later on.

Go back to the playground and import a new language `ta_split` and a new factor of language `ta truestem` into the existing corpora. Create another simple bash script to do it:

```
#!/bin/bash

for section in train dev test; do
  OUTCORP=en-ta-$section OUTLANG=ta OUTFACTS=truestem \
  OUTLINECOUNT=$(wc -l < ../projects/clara/english-tamil/ta_stems.$section) \
  TAKE_FROM_COMMAND="cat ../../projects/clara/english-tamil/ta_stems.$section" \
  eman init --start corpus

  OUTCORP=en-ta-$section OUTLANG=ta_split OUTFACTS=form \
  OUTLINECOUNT=$(wc -l < ../projects/clara/english-tamil/ta_split.$section) \
  TAKE_FROM_COMMAND="cat ../../projects/clara/english-tamil/ta_split.$section" \
  eman init --start corpus
done
```

Store it in `import_split_tamil.sh`. Use `chmod` to allow its execution and run it.

```
chmod +x import_split_tamil.sh
./import_split_tamil.sh
```

6.2 Run the Experiment

Open the file `baseline.traceback` and change the variables as follows:

- `SRCALIAUG=en+form` to `SRCALIAUG=en+stem4`
- `TGTALIAUG=ta+form` to `TGTALIAUG=ta+truestem`
- `ALILABEL=en-form-ta-form` to `ALILABEL=en-stem4-ta-truestem`

Save the file as `align_truestems.traceback`, clone it and run the experiment:

```
eman clone --start < align_truestems.traceback
```

It might now be interesting to view the new results:

```
eman retag && eman collect && eman tabulate && cat eman.niceresults
```

7 Translating into ta_split

Edit the baseline traceback once more. Specific occurrences of `ta` need to be changed, i.e. the traceback needs to be hacked a bit this time. The following `sed` pipeline does the job for you but make sure you understand it:

```
cat baseline.traceback \
| sed -e "s/AUG=ta+/AUG=ta_split+/" \
        -e "s/LANG=ta/LANG=ta_split/" \
        -e "s/ta\./ta_split./" \
        -e "s/ta-form/ta_split-form/" \
> ta_split.traceback
```

Clone the traceback and start the experiment:

```
eman clone --start < ta_split.traceback
```

7.1 Evaluation

Go to the `eval` step of this experiment. The following command will compare your joined output with the original Tamil reference:

```
cat corpus.translation | ../../projects/clara/split.pl -r \
| ./testbleu ../../projects/clara/english-tamil/ta.test
```

In order to see the results correctly in `eman.niceresults`, back up the existing file `BLEU` and redirect the output of `testbleu` into it.

8 Reordered English

8.1 Create the Data

In this step, we are going to take advantage of Treex again. Download the analyzed English data in Treex format:

```
cd ~/statmt/project/clara/english-tamil
wget http://ufallab.ms.mff.cuni.cz/~tamchyna/clara/english-tamil-en.analyzed
```

Run a Treex scenario that applies your reordering block (or the reference solution) on this data:

```
cat english-tamil-en.analyzed \
| treex Read::Treex language=en from=- \
| Util::SetGlobal language=en \
| Util::Eval anode='$anode->set_no_space_after(0);' \
| Your::Reordering::Block \
| A2W::ConcatenateTokens \
| Write::Sentences > en_reord
```

You can also use `A2A::ReorderHeadFinal` instead of your implementation.

8.2 Import the Corpora

Split the data into sections:

```
head -5000 en_reord > en_reord.train
tail -600 en_reord | head -300 > en_reord.dev
tail -300 en_reord > en_reord.test
```

Go back to playground and create a script `import_reordered.sh`:

```
#!/bin/bash

for section in train dev test; do
  OUTCORP=en-ta-$section OUTLANG=en_reord OUTFACTS=form \
  OUTLINECOUNT=$(wc -l < ../projects/clara/english-tamil/en_reord.$section) \
  TAKE_FROM_COMMAND="cat ../../projects/clara/english-tamil/en_reord.$section" \
  eman init --start corpus
done
```

Import the reordered English as an additional language by running the script:

```
chmod +x import_reordered.sh
./import_reordered.sh
```

8.3 Run the Experiment

Edit the file `baseline.traceback` and replace occurrences of `en` with `en_reord` or use the following `sed` pipeline:

```
cat baseline.traceback \
| sed -e 's/AUG=en+/AUG=en_reord+/' \
        -e 's/LANG=en/LANG=en_reord/' \
        -e 's/en\./en_reord./' \
        -e 's/en-form/en_reord-form/' \
> en_reord.traceback
```

Clone the traceback and start the experiment:

```
eman clone --start < en_reord.traceback
```

9 Excercises

9.1 Other Metrics

Re-run all your `eval` steps and replace them with seed `evaluator`. This seed runs Moses evaluator that is slower, but supports a number of MT metrics.

9.2 Higher Order of Language Model

Run a series of experiments (cloned, based on the baseline) that evaluate translation performance with different order of target language model. You can try values 4, 5 and further as long as you get improvement.

9.3 Varying Stem Lengths

We used stem length 4 in the first attempt to improve MT quality. How would the results look for different lengths? Run a series of experiments to find out. Note that you are going to have to add rules to `playground/corpman.rules` for length other than 4 and 5.

9.4 Inverse Translation Direction

So far we only experimented with English→Tamil. What is the simplest way to invert the translation direction? Apply the necessary modifications and try a baseline experiment. (Manual hacking of traceback will be needed.)

9.5 Hindi

Reordering on the source side (English) has been shown to improve translation quality for Hindi. All data are available in the directory:

```
~/statmt/projects/clara/english-hindi/
```

Pre-analyzed English data can be downloaded here:

```
http://ufallab.ms.mff.cuni.cz/~tamchyna/clara/english-hindi-en.analyzed
```

Create two experiments:

- Baseline with stem4-stem4 alignment.
- Reordered English in the source, stem4-stem4 alignment.

Note that you will need to import the corpora similarly as we did in the previous experiments.

9.6 Hack eman.tabulate

Look into the file `playground/eman.tabulate`. Try to swap columns and rows in the first table. Try to define more tables that would provide meaningful insight into your results.

10 Addicter

Addicter was installed along with SMT tools during your checkout of `statmt` repository.

10.1 Experiment Preparation

Go to the cgi folder:

```
cd addicter/cgi
```

Create the experiment folder (folder name will be then understood by Addicter as the experiment name):

```
mkdir en_ta_stemalign
```

10.1.1 Data

For each experiment, you need to supply the following data to Addicter:

- Training data:
  - Source: `s.tm.../corpus/corpus.src.gz`
  - Target: `s.tm.../corpus/corpus.tgt.gz`
  - Word alignment: `s.tm.../alignment.custom`

- Test data:
  - Source: `s.translate.../corpus.src`
  - Reference: `s.eval.../corpus.reference`
  - MT output: `s.translate.../translated.gz`

Test data are mandatory for automatic error detection and classification, while training data are needed for Word Explorer.

Copy all data into the experiment folder:

```
cp s.translate.../corpus.src        ../addicter/cgi/en_ta_stemalign/test.src
cp s.eval.../corpus.reference       ../addicter/cgi/en_ta_stemalign/test.tgt
zcat s.translate.../translated.gz > ../addicter/cgi/en_ta_stemalign/test.system.tgt
zcat s.tm.../corpus/corpus.src.gz > ../addicter/cgi/en_ta_stemalign/train.src
zcat s.tm.../corpus/corpus.tgt.gz > ../addicter/cgi/en_ta_stemalign/train.tgt
cp s.tm.../alignment.custom         ../addicter/cgi/en_ta_stemalign/train.ali
```

10.1.2 Reference to Hypothesis Alignments

Go to the experiment folder:

```
cd ../addicter/cgi/en_ta_stemalign
```

Each alignment must be in its own subfolder. The name of the subfolder is understood by Addicter as the name of the alignment. So, for each alignment create a folder. Each reference to hypothesis alignment must be named as test.refhyp.ali in the given folder.

Addicter implements several monolingual alignment algorithms: LCS, HMM, Injective Greedy alignment. All are implemented as Perl scripts in `testchamber` folder. All take the reference and hypothesis translations as the input.

LCS usage:

```
mkdir ali_lcs
../../testchamber/align-lcs.pl test.tgt test.system.tgt > ali_lcs/test.refhyp.ali
```

HMM usage:

```
mkdir ali_hmm
../../testchamber/align-hmm.pl test.tgt test.system.tgt > ali_hmm/test.refhyp.ali
```

In order to use the Greedy alignment, you need morphologically analyzed data.

Addicter also implements creating reference-hypothesis alignment via source-to-reference and source-to-hypothesis alignments. The source-to-reference alignment must be named test.ali, source-to-hypothesis alignment must be named test.system.ali. Both files must be in experiment folder. Unfortunately, we do not have source-to-reference neither source-to-hypothesis alignments, so we can't use the "via source" alignment.

10.1.3 Indexing of Training Data for Word Explorer

This step is mandatory if you want to use Word Explorer. Go to the `prepare` folder:

```
cd ../../addicter/prepare
```

Make an auxiliary directory for indexes:

```
mkdir indexes
```

Run Addicter's indexer on your data:

```
./addictindex.pl -trs ../cgi/en_ta_stemalign/train.src \
  -trt ../cgi/en_ta_stemalign/train.tgt \
  -tra ../cgi/en_ta_stemalign/train.ali -oprf s -o indexes
```

Reverse the alignment:

```
../../../clara43/scripts/reverse_alignment.pl ../cgi/en_ta_stemalign/train.ali \
  > ../cgi/en_ta_stemalign/train_reversed.ali
```

Run Addicter's indexer in reversed direction:

```
./addictindex.pl -trs ../cgi/en_ta_stemalign/train.src \
  -trt ../cgi/en_ta_stemalign/train.tgt \
  -tra ../cgi/en_ta_stemalign/train_reversed.ali -oprf t -target -o indexes
```

You could also use test data and source-to-reference and source-to-hypothesis alignments as options -s, -r, -h, -ra, -ha, if you had them.

Move the index files to experiment folder:

```
mv indexes/*index* ../cgi/en_ta_stemalign
```

Remove the auxiliary folder if you need to save space:

```
rm -r indexes
```

10.2 Error Detection

Go to the `prepare` folder:

```
cd addicter/prepare
```

For each reference-to-hypothesis alignment run the detecter.pl script:

```
./detecter.pl -s ../cgi/en_ta_stemalign/test.src \
  -r ../cgi/en_ta_stemalign/test.tgt \
  -h ../cgi/en_ta_stemalign/test.system.tgt \
        -a ../cgi/en_ta_stemalign/ali_lcs/test.refhyp.ali        \
  -w ../cgi/en_ta_stemalign/ali_lcs
```

```
./detecter.pl -s ../cgi/en_ta_stemalign/test.src \
  -r ../cgi/en_ta_stemalign/test.tgt \
  -h ../cgi/en_ta_stemalign/test.system.tgt \
        -a ../cgi/en_ta_stemalign/ali_hmm/test.refhyp.ali        \
  -w ../cgi/en_ta_stemalign/ali_hmm
```

If you don't specify the alignment file (use the script without the -a option), Addicter will use the greedy alignment.

10.3 Running the Server

You are now ready to view your results in web browser. Go to Addicter main folder:

```
cd ..
```

Run the server.pl script:

```
./server.pl 8080
```

Copy the link from script output into web browser (Opera or Firefox, not Konqueror). The number is the port number. You can write a different one or completely omit it – in that case, Addicter will choose the port randomly.