Getting Started

At Student machines, ÚFAL machines, or your laptop...

"Install" `eman`

- go to `ufal.cz/eman`
- visit the Download page
- get the automatic installation script
- source it to install eman:

```
. install-all.sh mtm15
```

- (This will put everything in the mtm15 directory.)

Setup Corpora

- Czech→English translation
- Training data: roughly 0.1% of CzEng 1.0 (15k sentence pairs)
- Dev set: 10% of WMT 2012 (300 sentence pairs)
- Test set: 10% WMT 2013 (300 sentence pairs)

Download the data:

```
http://bit.ly/mtmarathon15
```

Extract it into a subdirectory your playground, e.g.:

```
mkdir mtm15/ufal-smt-playground/playground/corpora
```

Importing the Corpora

- Every corpus has to "enter the world of eman".
- This can be done using the seed `corpus`.

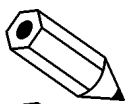
"`eman init corpus`" requires the following variables:

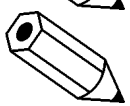
- `TAKE_FROM_COMMAND` command which produces the corpus
- `OUTCORP` corpus name
- `OUTLANG` corpus language
- `OUTFACTS` description of factors
- `OUTLINECOUNT` number of lines that we are expecting to get, used as a sanity check

Importing the Corpora

E.g. for training data, the Czech side:

```
TAKE_FROM_COMMAND="cat ../corpora/train.cs" \
OUTLINECOUNT=15000 \
OUTCORP=train OUTLANG=cs \
OUTFACTS=lc+lemma+tag \
eman init --start corpus
```

 Inspect the step directory. Where is the corpus stored?

 Create a bash script/"one-liner" to import all corpora: train/dev/test, cs/en (loop over sections and languages).

Did it work? Find out:

```
eman ls --stat
```

Frequent mistake: wrong OUTLINECOUNT for dev and test.

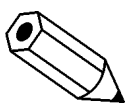
Listing and Printing Corpora

Corpman links symbolic names with `corpus` steps:

```
./corpman ls # show all registered corpora
```

Corpman ensures uniform pre-processing:

```
./corpman train/cs+lemma --dump
 # (Construct and) print the corpus as lemmas.
```



Bonus: Calculate the OOV (out-of-vocabulary) rate of the test data given the training data for:

- English vs. Czech and lowercase forms vs. lemmas

Use `ufal-smt-playground/scripts/count-oov.pl` or `oov.pl` from Moses. (Or write your own.)

Compiling Moses

In eman's philosophy, software is just data.

- Binaries should be compiled in timestamped step dirs.
- ...so we know the exact code that was used.

Compile moses and GIZA++:

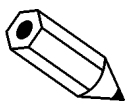
```
MOSESBRANCH=RELEASE-1.0 \
eman init --start mosesgiza
```

Warning: This won't work on local Unix machines. Instead, import an existing Moses step:

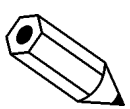
```
eman addremote \
~tamca7am/ufal-smt-playground/playground tamchyna

eman reindex
```

Baseline Experiment

```
cat corpora/baseline.traceback \
| eman clone --start
```

 While the experiment runs:

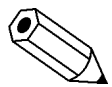
- Copy the traceback into your playground.
- Modify it to train word alignment on **lemmas** instead of **lc**. (But preserve the translation lc→lc!)
  - Note that `ALILABEL` is somewhat arbitrary but has to match between `align` and `tm`.

 Bonus: do the required edits using substitution in `eman`.

Hint: `eman --man`, look for the "`traceback`" command.

Looking Inside the Models

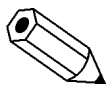
- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.gz`

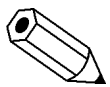


What is more probable: "united kingdom" or "united states"?

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.gz`


What is more probable: "united kingdom" or "united states"?


Why are longer n-grams more probable than short ones?

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.gz`
  - ✎ What is more probable: "united kingdom" or "united states"?
  - ✎ Why are longer *n*-grams more probable than short ones?
- Phrase table: `tm.1/model/phrase-table.0-0.gz`
  - ✎ How do you say "hi" in Czech?

Looking Inside the Models

- Go to one of your baseline `model` steps, look at files:
- Language model: `lm.1.gz`

  ✎ What is more probable: "united kingdom" or "united states"?

  ✎ Why are longer n-grams more probable than short ones?
- Phrase table: `tm.1/model/phrase-table.0-0.gz`

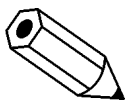
  ✎ How do you say "hi" in Czech?

  ✎ Phrase scores are $P(f|e)$, $lex(f|e)$, $P(e|f)$, $lex(e|f)$.

  Given that, what do the counts in the last column mean?

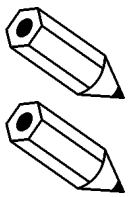
  (Let's look e.g. at the phrase "`ahoj ||| hi`".)

Tuning



How many iterations did MERT take?

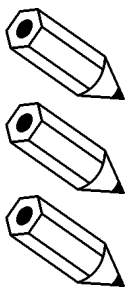
Tuning



How many iterations did MERT take?

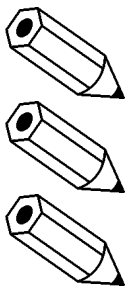
How did the BLEU score on the devset change?

Tuning



- How many iterations did MERT take?
- How did the BLEU score on the devset change?
- How much disk space did your MERTs need?

Tuning



- How many iterations did MERT take?
- How did the BLEU score on the devset change?
- How much disk space did your MERTs need?

- Standard Unix tool:

  `eman du -sh s.mert.*`

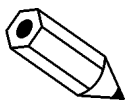
- Eman status:

  `eman eman ls mert --dus --stat`

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas

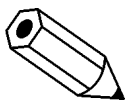


Look at `evaluator` steps. Which one is the baseline?

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas



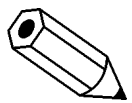
Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:
  `eman tb --vars s.evaluator.xyz | grep ALIAUG`

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas



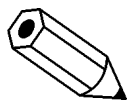
Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:
  `eman tb --vars s.evaluator.xyz | grep ALIAUG`
- Trace forward from the alignment step:
  `eman tf $(eman sel t align vre 'SRC.*lc')`

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas



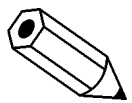
Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:
  `eman tb --vars s.evaluator.xyz | grep ALIAUG`
- Trace forward from the alignment step:
  `eman tf $(eman sel t align vre 'SRC.*lc')`
- Or just one `select` query:
  `eman sel t evaluator br t align vre 'SRC.*lc'`

Results

Let's compare MT quality (BLEU) of 2 systems:

- alignment on lowercase forms
- alignment on lemmas



Look at `evaluator` steps. Which one is the baseline?

- Trace back + grep:
  `eman tb --vars s.evaluator.xyz | grep ALIAUG`
- Trace forward from the alignment step:
  `eman tf $(eman sel t align vre 'SRC.*lc')`
- Or just one `select` query:
  `eman sel t evaluator br t align vre 'SRC.*lc'`

BLEU is in the "`s.evaluator.../scores`" file.

Team Work

- MERT is unstable $\Rightarrow$ multiple runs needed for a better estimate of "true" system performance.
- We do have multiple runs! ...among us.
- We will use `eman addremote` to share experiments.

Caveat: Your `home` directory is not accessible to other users. Let's fix that first:

```
cd ~ ; fs setacl -dir . -acl system:authuser rl ;
find ufal-smt-playground -type d \
-exec fs setacl -dir {} -acl system:authuser rl \;
```

Team Work

- Import your colleague's experiments, e.g.:

```
eman addremote \
~mtm999/ufal-smt-playground/playground fred
```

- Also add Aleš's playground for pre-compiled multeval:

```
~mtm003/multeval/playground
```

- Reindex (your playground):

```
eman reindex && ./corpman reindex
```

... from now on, `eman ls` is better than plain `ls`.

Team Work

✎ Use `eman select --remote` to find `evaluator` steps.

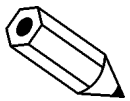
✎ Bonus: import `evaluator` steps from more playgrounds to get more reliable statistics (2 runs is too few).

✎ Run `multeval` (Clark et al. 2011):

```
MEVALBIN=s.mevalbin.f6750437.20130906-1727 \
BASELINE_EVALSTEPS="s.evaluator.XYZ,s.evaluator.WXY" \
IMPROVED_EVALSTEPS="s.evaluator.ABC,s.evaluator.DEF" \
eman init --start multeval
```

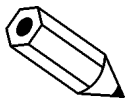
Results are written to `scores` file.

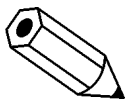
Wild Experimenting



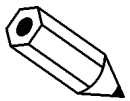
Run word alignment on `lcstem4`, `lcstem5`.

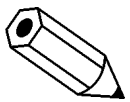
Wild Experimenting

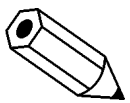

Run word alignment on `lcstem4`, `lcstem5`.


Try different orders of the language model (3, 4, 6).

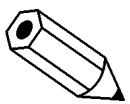
Wild Experimenting

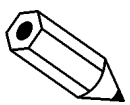

Run word alignment on `lcstem4`, `lcstem5`.

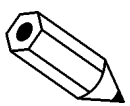

Try different orders of the language model (3, 4, 6).

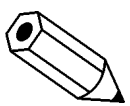

Translate from Czech lemmas into English forms (`lc`).

Wild Experimenting

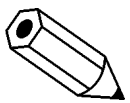

Run word alignment on `lcstem4`, `lcstem5`.

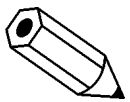

Try different orders of the language model (3, 4, 6).

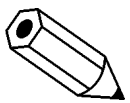

Translate from Czech lemmas into English forms (`lc`).

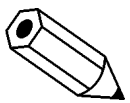

Try the opposite translation direction: English→Czech.

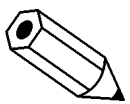
Wild Experimenting


Run word alignment on `lcstem4`, `lcstem5`.


Try different orders of the language model (3, 4, 6).


Translate from Czech lemmas into English forms (`lc`).


Try the opposite translation direction: English→Czech.


Set up a factored system:

- lc→lc (baseline path), and
- lemma→lc (alternative path).

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.

If you want to help:

- use eman,
- contribute to the "Commonspector" project.

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**

eman clone

CLONING is initing 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                        BLEU\s*=\s*([^\s,]+)
Snts  s.eval*/corpus.translation        CMD: wc -l
```

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

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

- 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**:

```
                    PER    CDER   TER    BLEU
5gr towards-CDER    44.61  33.97  31.04  12.28
5gr                 44.19  33.76  31.02  12.18
5gr PRO             43.91  33.87  31.49  12.09
5gr towards-PER     44.44  33.52  30.74  11.95
```

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

Jonathan H. Clark, Jonathan Weese, Byung Gyu Ahn, Andreas Zollmann, Qin Gao, Kenneth Heafield, and Alon Lavie. 2010. The Machine Translation Toolpack for LoonyBin: Automated Management of Experimental Machine Translation HyperWorkflows. Prague Bulletin of Mathematical Linguistics, 93:117–126.

Philipp Koehn. 2010. An Experimental Management System. Prague Bulletin of Mathematical Linguistics, 94:87–96, September.