Marian toolkit for NMT

Josef Jon

April 1, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Outline

Introduction to Marian

Lab: Practical NMT

Lab: Efficient NMT

Introduction to Marian

Outline

Introduction to Marian

Lab: Practical NMT

Lab: Efficient NMT

Marian

- Pure C++ toolkit for training and decoding NMT models
- Minimal depedencies (CUDA for GPU, MKL for CPU decoding)
- `https://www.aclweb.org/anthology/P18-4020/`

Marian

- Very fast decoding on CPU
- Implements many features useful in practice
- Simple to deploy and use
- Not so comfortable to debug and extend (but it might get better soon: `https://marian-project.eu/`)

Features

- Transformer and RNN models
- Dynamically sized mini-batches for efficient GPU memory utilization
- Guided attention (to make the model produce usable word alignments)
- Factors – annotation of the input tokens with additional information (e.g. case)
- Delayed updates (allows for large batches on smaller GPUs)
- Exponential smoothing of the model parameters (works similarly to checkpoint averaging in other frameworks)
- built-in SentencePiece subword segmentation – allows for training on raw text

Lab: Practical NMT

Outline

Steps for using NMT model in practice

1. Get the training data
2. Filtering and preprocessing
3. Train the model
4. Evaluation
5. Deployment (fast, efficient and cheap)

Getting the data

- Where to obtain the training corpora?
  - ENCS – CzEng
  - `http://statmt.org/wmt20/` if the language pair was used in some WMT tasks (you can replace wmt20 with wmt19,18,... and have a look)
  - others `http://opus.nlpl.eu/` – but be careful, you usually don't want e.g. localization files for software like GNOME
  - Europarl for most EU languages, high quality and quite large for most languages, but very specific domain
  - ParaCrawl – large, most EU languages, but not very high quality
  - CCAligned, MultiParacrawl - huge but not very useful for high quality translation
- Test sets
  - WMT Newstest
  - parts of high-quality datasets, like Global Voices

Preprocessing

- deduplication
- (tokenization, truecasing)
- Filtering
  - removing identical sentence pairs
  - language – fastText language id
  - length – clean-corpus.pl from moses
  - heuristics like length ratios, ratio of "real" words
  - adequacy - dual cross-entropy filtering Junczys-Dowmunt (2018)
- normalization (Romanian, Arabic) ?
- transliteration (Serbian, Multi-slavic models) ?
- subword segmentation – SentencePiece

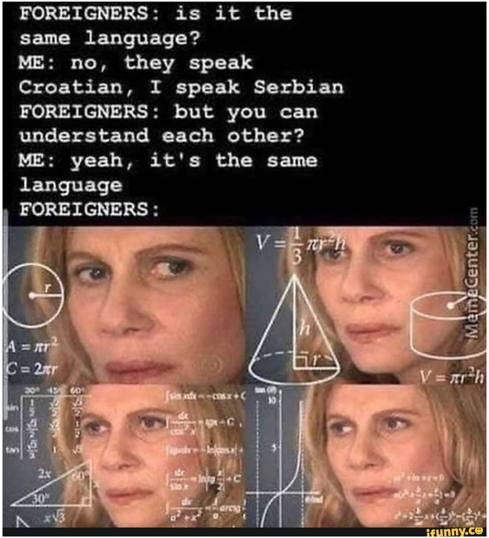
SentencePiece

- We need finite vocabulary – solution is to split rare words into parts (subwords) which occur more frequently
- https://github.com/google/sentencepiece
- if you want some nice practical features, like case factors, there's this almost secret wrapper around SP: https://github.com/microsoft/factored-segmenter

```
$ echo "This is a rare word: scrumptious" | spm_encode --model en.spm
▁This ▁is ▁a ▁rare ▁word : ▁scrum pt ious
$ echo "▁This ▁is ▁a ▁rare ▁word : ▁scrum pt ious" | spm_decode --model
    en.spm
This is a rare word: scrumptious
```

Language specific preprocessing

- normalization of Romanian
  - Ţţ Şş
  - Țț Șș
- (de)diacritization/vocalization of Arabic
  - بيت → بَيْتٌ
- transliteration of Serbian
  - пиво → pivo
  - then it can be concatenated with Croatian and Bosnian corpus if we translate from Serbian to another language
- Special word or morpheme segmentation (e.g. Japanese, Chinese)



Training

- Many different frameworks, we will use Marian
- Usually you want to train transformer model with default parameters
- Marian has internal SentencePiece segmenter, so we will work with raw text in our examples

Training

- Very basing training example, expects that you have two parallel files corp.en and corp.cs and trains transformer translation model using GPU 0, with as large batches as it can fit in 9000MB (this is setting for 11GB GPU, since the model also takes up some memory which is not taken into account here)
- After uncommenting the last option, dumps the config into a file config.yml, where you can have a look at the default values of all the options and change something (e.g. add validation files)

```
marian --task transformer -d 0 -t corp.en corp.cs -v encs.spm encs.spm
    -m model.npz --dim-vocabs 32000 32000 -w 9000 --mini-batch-fit #
    --dump-config > config.yml
```

Training example

```
$ qrsh -q 'cpu*' -l mem_free=16G,act_mem_free=16G,h_data=16G -pe smp 8
$ git clone https://github.com/cepin19/marian-lab.git/
$ cd marian-lab
$ export marian=/lnet/express/work/people/jon/marian-dist #for ufal network/grid
$ # OR export marian=/lnet/aic/personal/jon/marian-dist #for aic
$ export marian=TODO # for metacentrum?
$ ./download-models.sh
$ echo "this is a test" | $marian/guessmarian-decoder -v model/encs.spm
   model/encs.spm --mini-batch 1 --cpu-threads 1 -m model/model_base.npz
$ ./train.sh
```

Evaluation

- Automatic reference based metrics
  - BLEU
  - chrF
  - use sacrebleu

```
python -m pip install sacrebleu
sacrebleu -t wmt19 -l en-cs --echo src > news19.en.snt
cat news19.en.snt | bash translate.sh > news19.translated
cat news19.translated | sacrebleu -t wmt19 -l en-cs
```

- check validation results during the training to see if everything is ok or when to stop
  - `--valid-metrics`
    - builtin: perplexity, ce, bleu, chrf
    - custom script (see `val.sh`)
- Also look at the translation manually

Evaluation

- Any reference based metric is just a number saying how much is our input similar to a reference (usually single)
- But more similar words don't always mean better translation (what if the translation is even better than the reference?)
- Also some errors are worse than others

	BLEU
Source: It was a beautiful trip, I am glad I came along. Reference: Byl to nádherný výlet, je dobře, že jsem jel s vámi.	
Translation 1: Je dobře to nádherný výlet, to že jsem jel s vámi byl, je dobře, že jsem.	50.76
Translation 2: Krásná vyjížďka, rád jsem se k vám přidal.	3.7

Deployment

- marian-server – simple websocket server for Marian, needs to be wrapped in preprocessing
- Transformers Python library – supports Marian format models, but much slower inference
- Bergamot – inside browser

Modifying Marian

- We'll have a look at marian source code, but before that, run compare_speed.sh (just leave it running, it will take few minutes)
- If you are not a C++ expert, I would recommend to first implement a new feature in some other framework that is easier to debug and then port the implementation into marian
- Use CLion debugger (but you won't see whats inside the tensors)
- Pro tip: use debug() macro to print tensor values

Marian source code

- Real documentation started to appear recently:
  `https://marian-nmt.github.io/docs/api/`
- Few very old notes about the code:
  `http://www.statmt.org/jhu/?n=NMTWinterSchool.Marian`
- Marian is a differentiation engine based on dynamic computation graphs (basically the same thing as pytorch or tensorflow)
- basic unit of the computation graph is `Chainable`, usually referenced by its pointer of type `Expr`
- if the `Chainable` represents operation, it usually implements `forward` and `backward methods`
- also all the weight matrices and inputs are `Chainables`
- debug() macro prints out the content of a tensor in `Chainable` or values for forward and backward pass if the Chainable represents operation

Marian source code

- two files must be edited to add a new model type:
  - actual model file, for example models/transformer.h
  - and the new model must be registered in models/model_factory.cpp
  - model_factory links `type` config parameter to a specific constructor
  - so if you want to do some modifications to a transformer model, easy way to do that is to copy transformer.h, rename the classes, make the modifications, register the new classes in model_factory as `mytransformer` and run marian with `--task mytransformer`
- to change something in decoding/beam search/output generation, have a look at files

Lab: Efficient NMT

Efficient NMT decoding

- The main advantage of Marian over other frameworks is the inference speed on CPU
- Most of the the companies providing MT do not say what framework they use
- But when they do, it's usually Marian (Microsoft, ModernMT, Unbabel, Tilde)

Teacher-student training

- First, we train strong, but slow model (teacher)
- We translate the whole training corpus by the model
- This results in sentence pairs with lower cross-entropy for the model (less diverse and less surprising translations)
- We train smaller model (student) on the translated data
- Capacity of the small model is not enough to model the original training data (it is harder to "remember" all the "exceptions"), but it is good enough to learn from translations "normalized" by the teacher model
- Nice bonus: student can learn beam search, i.e. greedy search on the probability distributions generated by the student works almost as well as beam search, resulting in big speedup

Another sources of speedup

- Efficient implementation of low precision operations using special CPU instruction sets
- Smaller decoders - usually, encoder-decoder transformer has 6 encoder layers and 6 decoder layers
- Encoder outputs are computed only once, while decoder has to be recomputed after every output token
- It is much faster to have 10 layers in encoder and 2 in decoder
- Special model architectures – SSRU instead of self-attention in the decoder
- Lexical shortlists – softmax is expensive on CPU

References I

Marcin Junczys-Dowmunt. Dual conditional cross-entropy filtering of noisy parallel corpora. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 888–895, Belgium, Brussels, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6478. URL https://www.aclweb.org/anthology/W18-6478.