NMT with Minimal Parallel Resources Lab

June 28, 2017, 2:45 - 5:00

Welcome to the lab for the Neural Machine Translation with Minimal Parallel Resources workshop. In Monday's neural MT lab, you looked at the low level details involved in building a neural MT system from scratch in DyNet. In today's lab, we'll bring you to the other end of the spectrum (and another toolkit) as we experiment with and modify a complete, industrial-class NMT system, sockeye, which is built over the mxnet neural network toolkit. This will allow you to get a feel for how one engineers NMT to scale, and how to use the toolkits developed by larger labs to your advantage. Also, having done both labs, you will have been exposed to the two toolkits being used in our workshop.

For this afternoon's lab, you can work on two activities:

1. Training some NMT models on small data
2. Extending sockeye with new functionality

But first, we'll step you through getting up and running with mxnet and Sockeye on the Pittsburgh Supercomputing Center (PSC).

Enabling Python3 and mxnet on the PSC

To facilitate getting up and running quickly with mxnet, we have already made available an installation of Anaconda3 with mxnet and all of its dependencies installed. To enable this, add

```
>> export PATH="/pylon2/ci560op/fosterg/tools/anaconda3/bin:$PATH"
```

to your .bashrc, then use the following commands to get an interactive node

```
>> interact           # book a CPU node for an hour and ssh there
```

To make sure everything worked, you can run:

```
>> python /pylon2/ci560op/cherryc/min_mxnet_cpu.py
```

Note that this process will change your default python to python3, to support sockeye. Watch out, print() now requires brackets! If you want to duplicate this setup on your laptop (which we don't really recommend, as it'll take a fair amount of time), you can download Anaconda and then pip install mxnet and its other dependencies. Note that you'll want to ignore the GPU-specific instructions below.

Downloading the scripts from:

The interact nodes do not have internet. Exit to a head node and clone the repo.

```
>> exit
>> git clone https://github.com/duyvuleo/JSALT17-NMT-Lab
```

Downloading and Preprocessing the data

```
>> cd <your_current_folder>
(e.g., /home/<your_username>/JSALT17-NMT-Lab)
```

Source the required locations of tools into your environment by running:

```
>> source ENV.sh
```

(We'll do this again later, after installing Sockeye)

We are using a small parallel data from the Multi30K[1] collection (http://www.statmt.org/wmt16/multimodal-task.html), translating from English to German or German to English.

To download the data, simply run the script:

```
>> sh data.sh
```

The downloaded data include: training, validation and testing.

To preprocess the data, run:

```
>> sh preprocess.sh
```

Downloading and running Sockeye

To download Sockeye:

```
>> git clone https://github.com/awslabs/sockeye.git
```

To run Sockeye, first set up the CPUs:

```
>> interact -t 05:00:00
```

Update the location of your sockeye installation in ENV.sh

```
>> vim ENV.sh # Update the location of SOCKEYE here
>> source ENV.sh # Important
```

If you see the following, then everything is set correctly!

```
>> python $SOCKEYE/sockeye/train.py
```

---

[1] https://arxiv.org/abs/1605.00459

```
usage: train.py [-h] --source SOURCE --target TARGET
--validation-source VALIDATION_SOURCE --validation-target
VALIDATION_TARGET --output OUTPUT [--source-vocab SOURCE_VOCAB]
...
```

Build some baseline models

Here, we will build an attention-based encoder-decoder model. The neural network configuration is as follows:

- Word frequency cut-off: 2
- No. of layers of both encoder and decoder networks: 1
- Recurrent unit: LSTM
- Hidden dimension: 64
- Embedding dimension: 64
- Embedding source dimension: 64
- Embedding target dimension: 64
- Attention: MLP with hidden dimension 64
- Mini-batch size: 16
- Dropout: 0.1
- SGD: Adam
- SGD learning rate: 0.001 (Adam is an adaptive SGD optimization method that requires a very small initial learning rate. Please refer to Adam paper[2] for more details.)

This a very small model, we do it because we need to make sure it runs on the CPU in a reasonable time. If you have to access to GPUs feel free to change these.

Translating from English to German

As we are training on CPU only we will use a toy subset of the full dataset. You can train for as long as you want but you should have reasonable results after 3-4 checkpoints (15-20min). Just press Ctrl-C to halt training.

Things to look for:

1. Speed (samples/sec)
2. Reduction in (training/validation) perplexity: This should decrease after every epoch/checkpoint
3. When does this converge?

❖ Training

```
>> sh train-toy.sh en de
```

❖ Decoding

[2] https://arxiv.org/pdf/1412.6980.pdf

```
>> sh translate.sh models/multi30k-en-de/baseline
data/multi30k/test.en.atok
models/multi30k-en-de/baseline/test.de.atok.translated-beam5
```

- ❖ Evaluating

```
>> sh evaluate.sh data/multi30k/test.de.atok
models/multi30k-en-de/baseline/test.de.atok.translated-beam5
```

Translating from German to English

You can do the same now but in the reverse direction.

- ❖ Training

```
>> sh train-toy.sh de en
```

- ❖ Decoding

```
>> sh translate.sh models/multi30k-de-en/baseline
data/multi30k/test.de.atok
models/multi30k-de-en/baseline/test.en.atok.translated-beam5
```

- ❖ Evaluating

```
>> sh evaluate.sh data/multi30k/test.en.atok
models/multi30k-de-en/baseline/test.en.atok.translated-beam5
```

Inspecting the outputs

Take a look at the output translations. What kind of errors do you see? Are there any evidences of over or underfitting? If you know German you can do the same for the English-German outputs.

You can also try to translate with previous checkpoints by changing the translate.sh script. Try to translate using the parameters from the first checkpoint (params.0001). See any changes in the BLEU scores? Any changes in the translations?

Training with the full dataset

If you have access to GPUs you can also try to train on the full dataset. You'll probably want to change the hyperparameters for that, for instance, increase the hidden layer size. Training on the full dataset in a CPU is possible but might take a few hours so it is up to you.

Playing with hyper-parameters in Sockeye

For example:

- Different attention type: MLP, dot, … (Luong et al., 2015) by changing `"-attention-type {mlp,dot,...}"`.

- Different recurrent structure: LSTM, GRU by changing "`--rnn-cell-type {lstm,gru}`".
- Coverage Model (Tu et al., 2016) by adding "`--attention-coverage-type {tanh,sigmoid,relu,softrelu,gru,count}`" and "`--attention-coverage-num-hidden ATTENTION_COVERAGE_NUM_HIDDEN`"
- Different dropout values by changing "`--dropout DROPOUT`".
- Ensembling the models (e.g. executing different runs of training the models and ensembling them) by setting up "`--models MODEL_1 MODEL_ 2`" and "`--ensemble-mode {linear,log_linear}`".

Note that you need to modify the respective train.sh and translate.sh to add the above-mentioned hyper-parameters.

Extend Sockeye with a new feature

The goal in this section is to integrate external linguistic knowledge into the model. Here we assume that we have the means to annotate the source sentences with some sort of linguistic preprocessing such as word morphology, syntactic dependencies, semantics, etc. In this lab we will focus on adding Part-of-Speech (POS) information to the source sentences.

English POS Tagging

First, we need to annotate the text data with POS tags. We will use the NLTK toolkit for that, which comes with pre-trained POS tagging models for English.

```
>> pip install nltk --user
>> python
>> import nltk
>> nltk.download()
```

then follow the instructions from NLTK. You need to download the packages:

- averaged_perceptron_tagger
- punkt

To tag the source data, run the scripts:

```
>> cat data/multi30k/train-toy.en.atok | python tag_pos.py > data/multi30k/train-toy.en.atok.pos
>> cat data/multi30k/val.en.atok | python tag_pos.py > data/multi30k/val.en
```

Integration into the translation model

A simple way to integrate POS tag information is to augment the source words. This can be done by concatenating each word with its corresponding POS tag, as in this example:

- **Source:** I saw her duck
- **Augmented source 1:** I_PRP saw_VBD her_PRP duck_NN

- **Augmented source 2:** I PRP saw VBD her PRP duck NN

Using this format we can simply reuse the same code as before, now treating each source sentence as sequence of “augmented words”.

Your task here is to preprocess the data in the augmented form described above and then train and evaluate your model. Is it better or worse than the baseline?