

Lab 7: Neural Machine Translation

Instructions to follow the lab n.7 GIAN course about MT

Status: 15/12/2016 BP V0.01

Preliminaries

1. Your host computer contains a “SMT” virtual machine provided
2. Open virtual box (on Ubuntu, click on “search”, type virtual, open it)
3. The virtual machine will appear, select the machine, click “start”

The username is “smt” and password is the same

Use of a simple perceptron

Open a terminal, go into “lab” directory

```
cd ~/Icon2016/labs
```

You can see that there is a “languess-min.py” python script

A python script is a text file, so you can look at it already.

This is the perceptron we saw together during the lecture, it reads sentences from a text file (‘test.txt’, which you should look at), then from the 100 first lines, it learns a classifier. Then apply it to the last 10 lines.

You can edit the file using e.g. gedit, and please delete the two lines

```
if sys.argv...  
    fileName=argv[]
```

(this was a wrong version of the script that was published on your virtual machine)

Then you can execute it using

```
python langguess-min.py
```

- ⇒ Usually it displays nothing, because all the tests were correct
- ⇒ But if you launch it several times, you will see sometimes appearing:

```
Test:192 +A mixture or composition to form a variable chemical solution or an aqueous  
substance and process of application, which permits the control of environmental pollution  
caused mainly by smoke-stack industries and by the increase of internal combustion  
vehicles, characterized because the mixture or composition consists of: sodium bicarbonate  
between 0.1% and 98%, preferably between 2% and 30%; a urea between 1% and 98%, preferably  
between 1% and 63%; and distilled or carbonated water between 20% and 95%, preferably  
between 10% and 70%. error=0 Prediction:False wasTrue
```

this is displayed because this specific example was classified as being non-English while it should have been English

It shows that the random initialization of the weights in the network makes it non-deterministic

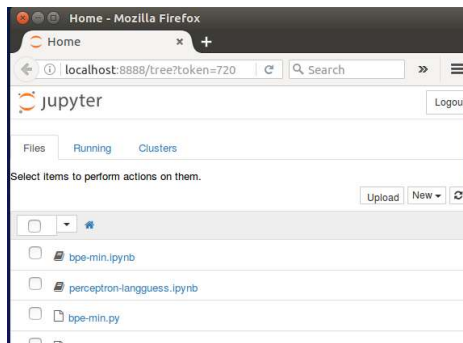
Use jupyter notebook

The jupyter notebook is a nice environment for developing in Python,

Open it typing:

`jupyter notebook`

The following appears:




(this is a webinterface allowing you to edit and run python).

The “language guesser” script is available in that environment, you can open it clicking on “perceptron-langguess.ipynb)

Note that, on our smt virtual machine, we have numpy library installed only for python2, therefore install pyhton2 for jupyter.

```
python2 -m ipykernel install -user
```

then, on jupyter noteboook go to „Kernel“, change kernel to python2

Then you an easily edit the script and execute it on the environment (execute it clicking on )

Try to use this environment to launch a training/testing and then display the content of the “weigth” matrix, it contains the weights that it puts for each single char, look at which chars are more representative for English/Others

BPE (Byte Pair Encoding)

Note that BPE is not a Neural Network learning process, it is usefull as a preliminary step, but you can skip this paragraph if you want to stick to Neural Machine Translation

On “jupyter notebook”, you can open the bpe-min.ipynb script and explore the algorithm.

Character level language model

Minimal:

This python script, taken from Andrej Karpathy, is a RNN trained on characters only, if you run it, you will see that, at the beginning, it displays random characters, then, after few iterations, it will begin to produce characters, then it will begin to produce some English words (e.g. “pats the bloom wray in the romntir”).

Remark: do not expect much from this simple tool: it does not use LSTMs nor GRUs, so even if you let it run one full day, the output will never be good English

```
cd ~/min-char-rnn
python min-char-rnn.py
```

→ it runs all the time, to stop you will have to type “CTRL-C”

Improved:

An improved version of this script exists (implementing LSTMs), using the NN library Torch.

You can use it on the virtual machine:

I advise you to open 2 terminals: one for launching the training, one other to look at the results

```
cd ~/char-rnn
```

Training:

```
th train.lua -gpuid -1 -eval_val_every 1
```

Look at the Readme file to understand the parameters: here we ask it to run without GPU (gpu=-1) and we want the system to write a model at every single step.

You can see, in the other terminal, that the directory “cv” has a file that appears at every epoch

You can now try to generate a text using any of the saved model:

```
th sample.lua -gpuid -1 cv/lm_lstm_epoch0.04_3.3237.t7
```

Note that 0.04 means: the 4th epoch on the data, and 3.3237 the value of the loss function (the lower the better)

Nematus / AmuNMT

(if we find a way to copy the models on your virtual machine)

```
cd /home/smt/nematusTrainedModelHiEn
```

Look at the two models that exists, there were trained at various “epochs”

Each one can be decoded using AmunMT using the following “long” command:

```
cat ~/parallel-corpora/hi-en/trainMoses/test.hi |~/amunmt/build/bin/amun -m
model.npz.dev.npz.best_bleu -s trainset.bpe.hi.json -t trainset.bpe.en.json -n -b 12 |perl -pe
's/\@\\@ //g' > output.en
```

Launch a BLEU evaluation on this, comparing it with the “old” model

How-to generate a NMT model from Hindi to English

Here is the script that was used to generate this model:

Note that you will need a GPU – here gpu0 – to be able to run it:

created out of hi-en corpus (hindic corpus)

take devset.0 as devset, testset0 as testset

```
# First learn the BPEs out of the trainsets
cat trainset.en | subword-nmt/learn_bpe.py -s 5000 > bpe-codes.en
cat trainset.hi | subword-nmt/learn_bpe.py -s 5000 > bpe-codes.hi
# Tehn apply those BPEs on trainset, devset and testset
cat trainset.hi | subword-nmt/apply_bpe.py -c bpe-codes.hi > trainset.bpe.hi
cat trainset.en | subword-nmt/apply_bpe.py -c bpe-codes.en > trainset.bpe.en
cat dev.hi | subword-nmt/apply_bpe.py -c bpe-codes.hi > devset.bpe.hi
cat dev.en.0 | subword-nmt/apply_bpe.py -c bpe-codes.en > devset.bpe.en
cat test.hi | subword-nmt/apply_bpe.py -c bpe-codes.hi > testset.bpe.hi
cat test.en.0 | subword-nmt/apply_bpe.py -c bpe-codes.en > testset.bpe.en

# Create Nematus' dictionaries (it will create trainset.bpe.hi.json)
/data/smt/neural/install/nematus/data/build_dictionary.py trainset.bpe.hi
/data/smt/neural/install/nematus/data/build_dictionary.py trainset.bpe.en

# Now launch the Nematus training
THEANO_FLAGS=device=gpu0,floatX=float32 python /data/smt/neural/install/nematus/nematus/nmt.py --
factors 1 --datasets trainset.bpe.hi trainset.bpe.en --dictionaries trainset.bpe.hi.json
trainset.bpe.en.json --objective CE --valid_datasets devset.bpe.hi devset.bpe.en --max_epoch 600 -
-dim_word 256 --dim 250 --validFreq 5000 --sampleFreq 2000 --saveFreq 2000 --dispFreq 2000
# It will run for a long time, and will create some "model.npz", you then choose the best one
# let's say we call it model.npz

# Then you can test it using AmuNMT (here we ask for the translation of devset.bpe.hi):
cat devset.bpe.hi | /data/smt/amunmt/build/bin/amun -m model.npz -s trainset.bpe.hi.json -t
trainset.bpe.en.json -n -b 12 -d 0 | perl -pe 's/\@\\@ //g' > devset.en.outputNmt
```