Natural Language Understanding, Generation, and Machine Translation (2018–2019)

Jonathan Mallinson, Denis Emelin, Ida Szubert, and Rico Sennrich
School of Informatics, University of Edinburgh

NLU+ Coursework 2: Neural Machine Translation

This assignment is due March 14 at 4pm.

Deadline for choosing partners: March 7 at 4pm.

Executive Summary. Your task will be to work with a simple baseline NMT model, analyzing its code and neural architecture, exploring the parallel data, and evaluating its performance. Finally, you will implement the lexical model as described in Nguyen and Chiang (2017), and discuss its effects on translation quality.

IMPORTANT: While modifying the baseline code may only take you a few minutes or hours, training the extended model will take you **A LOT OF TIME**. You might implement something in thirty minutes and leave it to train overnight. Imagine that you return the next morning to find it has a bug! If the next morning is the due date, then you'll be in a pickle, but if it's a week before the due date, you have time to recover. So, if you want to complete this coursework on time, start early. I will not take pity on you if you start too late.

Submission Deadline and Pacing. The coursework is due on March 14 at 4pm. If you want to work with a partner, you have an earlier deadline to relate this to us, on March 7 at 4pm, i.e. one week before the coursework due date.

This time, there are eight questions in total, divided into three areas of interest. As the initial four questions will ask you to explore the provided baseline NMT model, they should not be overly time-consuming. However, (re-)training the model will take you several hours, so make sure to plan accordingly. The final four questions, on the other hand, require you to make significant modifications to the baseline. Moreover, training the extended NMT model will take a while. As such, make sure to allocate sufficient time to implement, train, and evaluate your proposed model extensions.

Pair work policy. You are *strongly* encouraged to work with a partner on this assignment. When you work with another person, you learn more, because you need to explain things to each other as you pool your collective expertise to solve problems. Explaining something to another person helps you debug your own thinking, and their questions help you overcome your own blind spots—something you cannot do on your own by staring at maths, code, or data. For this reason, it is best to seek partners with complementary skills to your own. You may not work in teams of three or more. **You may not have the same partner as in coursework 1.**

As a practical matter, students who work in pairs are likely to receive marks earlier, because solo submissions require additional marking hours, and the course enrollment is much higher than projected.

If you work with a partner, only one of you should submit your completed work. But I need to know that the submission represents the work of two people, and I need to know that reliably in advance in order to estimate marking hours. So, **you must do the following to ensure that both of you receive credit:**

1. Create a text file containing two lines. Each line should contain the UUN and name of one partner. So, your file should look something like this:

   ```
   s1234567 Student One
   s7654321 Student Two
   ```

2. Submit the file on DICE using this command:

   ```
   submit nlu+ cw2 partners.txt
   ```

You must do this by March 7 at 4pm. You may not change partners after you have done this, so take this commitment seriously. If I do not receive a submission from your or your partner by the deadline, I will assume that you are working alone. Either way, I will confirm your choice with you shortly after the deadline to avoid later confusion. **I advise you to choose your partner now and get to work.** Piazza offers a feature that allows you to search for a partner, and you are welcome to use this. I will not be in a position to assign partners for you, but by now you should know some of your classmates from ANLP or other courses.

Submission. Your solution should be delivered in two parts and uploaded to Blackboard Learn. **Do not include your name or your partner's name in either the code or the write-up.** The coursework will be marked anonymously since this has been empirically shown to reduce bias. (I anonymize the filenames before marking.)

For your writeup:

- Write up your answers in a file titled `<UUN>.pdf`. For example, if your UUN is `S123456`, your corresponding PDF should be named `S123456.pdf`.
- The answers should be clearly numbered and can contain text, diagrams, graphs, formulas, as appropriate. Do not repeat the question text. If you are not comfortable with writing math on Latex/Word you are allowed to include scanned handwritten answers in your submitted pdf. You will lose marks if your handwritten answers are illegible.
- On Blackboard Learn, select the Turnitin Assignment "Coursework 2 ANSWERS". Upload your `<UUN>.pdf` to this assignment, and use the submission title `<UUN>`. So, for above example, you should enter the submission title `S123456`.
- Please make sure you have submitted the right file. We cannot make concessions for students who turn in incomplete or incorrect files by accident.

For your code and parameter files:

- Compress your code for `lstm.py` and `train.py` into a ZIP file named `<UUN>.zip`. For example, if your UUN is `S123456`, your corresponding ZIP should be named `S123456.zip`.
- On Blackboard Learn, select the Turnitin Assignment "Coursework 2 CODE". Upload your `<UUN>.zip` to this assignment, and use the submission title `<UUN>`. So, for above example, you should enter the submission title `S123456`.

Good Scholarly Practice Please remember the University requirement as regards all assessed work for credit. Details and advice about this can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

and links from there. Note that you are required to take reasonable measures to protect your assessed work from unauthorized access. For example, if you put your work in a public repository then you must restrict access only to yourself and your partner. You are permitted to publish your code solution two weeks after the coursework deadline.

For your write-up, and particularly on the final question, you should pay close attention to the guidance on plagiarism. Your instructor is **very good** at detecting plagiarism that even Turnitin can't spot, and in one year reported 22 students for plagiarism found by hand. In short: the litmus test for plagiarism is not the Turnitin check—that is simply an automated assistant. If you have borrowed or lightly edited someone else's words, you have plagiarized. Write your report in your own words. We will not mark for eloquence—as long as we can clearly understand what you did, that is fine.

Python Virtual Environment For this assignment you will be using Python along with a few open-source packages, such as the pyTorch deep learning library. These packages cannot be installed directly, so you will have to create a virtual environment. We will use virtualenv for this purpose.

The instructions below are for DICE. You are free to use your own machine, but we cannot offer support for non-DICE machines.

Open a terminal on a DICE machine and follow these instructions. We are expecting you to enter these commands in one-by-one. Waiting for each command to complete will help catch any unexpected warnings and errors.

First, make sure that you are using python 3.5 or higher:

```
python3 --version
```

If the version is lower than 3.5, run the following commands:

```
$> wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
$> bash ./Miniconda3-latest-Linux-x86_64.sh
$> rm ./Miniconda3-latest-Linux-x86_64.sh
$> source ~/.bashrc
```

Now your default python version should be 3.7. Confirm with `python3 --version`.

Having ensured you have a right version of python, proceed to create a new environment called mtenv

1. Create a new environment called `mtenv`:
   ```
   $> mkdir mtenv
   $> virtualenv -p python3 mtenv
   ```
2. Activate the `mtenv` virtual environment
   ```
   $> source mtenv/bin/activate
   ```
3. Install pyTorch 0.4.1[1]
   ```
   $> pip install -U pip
   $> pip install https://download.pytorch.org/whl/cpu/torch-0.4.1.post2-cp37-cp37m-linux_x86_64.whl
   ```
4. Install additional packages required by the baseline code
   ```
   $> pip install tqdm seaborn pandas matplotlib numpy
   ```

You should now have all the required packages installed. You only need to create the virtual environment and perform the package installations (step 1-4) **once**. However,

[1]The link provided is compatible with python 3.7. If you're using a different python version, please find a compatible PyTorch wheel at https://pytorch.org/get-started/previous-versions/linux-no-cuda-binaries and run `pip install https://download.pytorch.org/whl/<your wheel>`

make sure you activate your virtual environment (step 2) **every time** you open a new terminal to work on your NMT assignment. Remember to use the `deactivate` command to disable the virtual environment when you don't need it.

Baseline NMT model To get the baseline code, open the terminal and type

```
git clone https://github.com/demelin/nmt_toolkit
```

You'll find several directories, including `raw_data` containing raw English and Japanese parallel data (from this helpful tutorial: `https://github.com/neubig/nmt-tips`), `prepared_data` containing the pre-processed data your models will be trained on, and `seq2seq` containing the code you will be asked to extend. Moreover, you will find several python files of importance to the assignment:

- `train.py` is used to train the translation models. You do not need to modify this script.
- `translate.py` translates the test-set greedily using model parameters restored from the best checkpoint file and saves the output to `model_translations.txt`. You do not need to modify this script.
- `visualize.py` generates heat-maps from the decoder-to-encoder attention weights for the first 10 sentence pairs in the test-set. You do not need to modify this script.

NOTE: You can either train the baseline model yourself or use the pre-trained checkpoints we provide in the `pretrained_checkpoints` directory. If you decide to train the model yourself, follow the instructions below, otherwise skip to Part 1 of the coursework.

To train the baseline model, run the following commands in your terminal from the `nmt_toolkit` directory:

```
source ../mtenv/bin/activate
python train.py
```

You can specify the hyper-parameters for the training using the appropriate argument flags, but we recommend sticking with the default, fine-tuned settings.

After calling the training script, you should see a progress bar denoting the training progress for the current epoch. Training will continue until no improvement can be observed on the development set for 5 consecutive epochs. After your model has finished training, use it to translate the test set:

```
python translate.py
```

Next, use the `multi-bleu.perl` script to calculate the test-BLEU score of the baseline model:

```
perl multi-bleu.perl -lc raw_data/test.en < model_translations.txt
```

Note the BLEU score and back up the checkpoints directory (e.g. by renaming it to `checkpoints_baseline`). This model is still quite basic and trained on a small dataset, so the quality of translations will be (very) poor. Your goal will be to see if you can improve it.

The current translation model implementation in `sec2sec/models/lstm.py` encodes the sentence using a bidirectional LSTM: one LSTM passing over the input sentence from left-to-right, the other from right-to-left. The final states of these LSTMs are concatenated and attended over by the decoder, using global attention with the general scoring function as described in Luong *et al.* (2015). While the encoder is implemented as a single-layer bidirectional RNN equipped with the LSTM cell, the decoder is a single-layer unidirectional RNN, also equipped with the LSTM cell.

Part 1: Getting Started

Before we go deeply into modifications to the translation model, it is important to understand the baseline implementation, the data we run it on, and some of the techniques that are used to make the model run on this data.

Question 1: Understanding the Baseline Model [10 marks]

The file `seq2seq/models/lstm.py` contains explanatory comments to step you through the code. Five of these comments (A-E) are missing, but they are easy to find: search for the string `__QUESTION` in the file. A sixth comment (F) is missing from `train.py`. For each of these cases:

1. Add explanatory comments to the code
2. Copy your comments to your answer file (we will mark the comments in your answer file, not the code, so it is vital that they appear there)

If you aren't certain what a particular function does, refer to the pyTorch documentation: `https://pytorch.org/docs/0.4.1/`. (However, explain the code in terms of its effect on the MT model; don't simply copy and paste function descriptions from the documentation).

Question 2: Understanding the Data [10 marks]

In preparing the training data, word types that appear only once are replaced by a special token, `<UNK>`. This prevents the vocabulary from growing out of hand, and enables the model to handle unknown words in new test sentences (which may be addressed by post-processing).

Examine the parallel training data located in the `raw_data` directory (`train.en` and `train.jp`) and answer the following questions.

1. Plot (choose sensible graphs) the distribution of sentence lengths in the English and Japanese and their correlation. What do you infer from this about translating between these languages?
2. How many word tokens are in the English data? In the Japanese data?
3. How many word types are in the English data? In the Japanese data?
4. How many word tokens will be replaced by `<UNK>` in English? In Japanese?
5. Given the observations above, how do you think the NMT system will be affected by differences in sentence length, type/token ratios, and unknown word handling?

Part 2: Exploring the Model

Let's first explore the decoder. It makes predictions one word at a time from left-to-right, as you can see by examining the decoder module in the file `sec2sec/models/lstm.py` and the greedy decoding script in `translate.py`. Prediction works by first computing a distribution over all possible words conditioned on the input sentence. We then choose the most probable word, output it, add it to the conditioning context, and repeat until the predicted word is an end-of-sentence token (`<EOS>`).

Question 3: Improved Decoding [10 marks]

1. Currently, the model implements greedy decoding, which simply chooses the maximum-probability word at each time step. Can you explain why this might be problematic? Give language specific examples as part of your answer.
2. How would you modify this decoder to do beam search - that is, to consider multiple possible translations at each time step. **NOTE**: You don't need to implement

beam search. The purpose of this question is simply for you to think through and clearly explain how you would do it.

3. Often with beam search (and greedy decoding), the decoder will output translations which are shorter than one would expect. As such, length normalization is often used to fix this. Why does the decoder favour short sentences? What is a problem that length normalization can introduce?

Question 4: MOAR Layers! [10 marks]

1. Change the number of layers in the encoder = 2, decoder = 3. You dont need to modify the code to train a deeper model - this is already supported by the provided code. Inspect the source code to find out how you can control the number of encoder and decoder layers via command line arguments. Train a system with this deeper architecture, and report the command that you used.

2. Draw a diagram showing this new architecture (you may ignore the memory cell of the LSTM).

3. What effect does this change have on dev-set perplexity, test BLEU score and the training loss (all in comparison to the baseline)? Can you explain why it does worse/better on the training, dev, and test sets than the baseline single layer model? Is there a difference between the training set, dev set, and test set performance? Why is this the case?

As a basis for your comparison, here are results for the baseline model (your baseline performance may vary slightly due to the random nature of model parameter initialization):

- `test set BLEU: 7.98`
- `dev set perplexity during last epoch:  22.4`

After each training epoch, the latest model file is saved to disk as `checkpoint_last.pt`. If the model achieved a lower dev-set perplexity in the concluded epoch than in the previous epochs, a best model file is saved to disk, as well, as `checkpoint_best.pt`. You can find the checkpoint files in the appropriately named checkpoints directory.

Part 3: Lexical Model

In this final part of the assignment, we ask you to augment the encoder-decoder with the lexical model defined in section 4 of Nguyen and Chiang (2017). For this task, your primary guidance should be the descriptions provided in the paper. Moreover, we have marked the different points in the encoder-decoder implementation where you are strongly encouraged to insert your code.

Implementing the lexical model can be roughly subdivided into three steps:

1. Compute the weighted sum of source embeddings using weights extracted from the decoder-to-encoder attention mechanism.
2. Define the feed-forward layers used to project the weighted sum of source language embeddings.
3. Incorporate the so obtained lexical context into the calculation of the predictive distribution over output words.

To accomplish this, you only need to modify `sec2sec/models/lstm.py` and nothing else. Implementing the modifications should not take you very long, but retraining the model will.

NOTE: We recommend that test your modifications by retraining on a small subset of the data (e.g. a thousand sentences). To do that, you should add the flag `--train-on-tiny` to the set of arguments when executing `train.py`, i.e.:

```
python train.py --train-on-tiny
```

The results will not be very good; your goal is simply to confirm that the change does not break the code and that it appears to behave sensibly. This is simply a sanity check, and a useful time-saving engineering test when youre working with computationally expensive models like neural MT. For your final models, you should train on the entire training set.

Question 5: Implementing the Lexical Model [25 marks]

Implement lexical translation as described above. All changes to the baseline implementation must be done in the decoder. You should be able to easily access both source embeddings (assigned to the `src_embeddings` variable) as well as attention weights specific to each decoding step (assigned to the `step_attn_weights` variable). Adding your code to the specified positions within the decoder architecture will help ensure that everything works correctly.

Question 6: Effects on Model Quality [5 marks]

Retrain your translation model after augmenting it with the lexical model by running the following command:

```
python train.py --decoder-use-lexical-model True
```

Again, explain how the change affects results compared to the baseline in terms training set loss, dev perplexity, and test BLEU scores.

Question 7: Effects on Attention [10 marks]

One claim presented in the paper that you have just re-implemented is that attending over source embeddings allows the attention mechanism to learn better alignments between the source and target sentence (see section 6.3 and figure 2). To evaluate whether this is indeed the case, we ask you to visualize the attention weights learned by the baseline model and by your modified model (i.e. with the addition of the lexical model) for the first 10 sentences in the test set. To do so, you should run the following command, specifying the appropriate checkpoint directory:

```
python visualize.py --checkpoint-path checkpoints/checkpoint_best.pt
```

The corresponding attention heat-maps will be saved to the `visualizations` directory of the NMT toolkit.

Do you notice a difference between the alignments learned by the extended model as compared to the baseline model? Why do you think this is the case? Base your argument on evidence from the data. You'll need to understand the Japanese words to do this effectively (use Google Translate). If the attention does not look reasonable, is it still a better translation than the baseline model? If it is, what does this say about attention?

Question 8: Analyse Effects on Translation Quality [20 marks]

Where does the lexical model help? You should have noticed that your test-BLEU and dev-perplexity scores improved after adding the lexical model to your baseline translation model. Can you find any patterns in which type of source-target pairs this improvement is found?

1. Come up with hypotheses of what type of source or/and target sentence benefit from lexical information. Give reasons for your hypothesis.

2. Find evidence which tests this hypothesis. You may want to consider manually comparing the test-set translations generated by your extended model with baseline translations and the gold-standard references found in `raw_data/test.en` (however you may gather evidence any way you like).

3. Evaluate your evidence, does it support your hypothesis? What does your evidence mean?

References

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Toan Q Nguyen and David Chiang. Improving lexical choice in neural machine translation. *arXiv preprint arXiv:1710.01329*, 2017.

Acknowledgements

The baseline NMT implementation is based on the following codebase: `https://github.com/tangbinh/machine-translation`