Lab 7 Exercise - Transforming Sequences

- .-. .- -. ... ..-. —— .-. ¯ .. -. ¯. / ... . ¯.- ..- . -. -.-. . ...

Jonathon Hare (jsh2@ecs.soton.ac.uk)

May 6, 2020

This is the exercise that you need to work through **on your own** after completing the seventh lab session. You'll need to write up your results/answers/findings and submit this to ECS handin as a PDF document along with the other lab exercises near the end of the module (1 pdf document per lab).

We expect that you **will use no more than one side** of A4 to cover your responses to *this* exercise. This exercise is worth 5% of your overall module grade.

1 Sequence-to-Sequence Modelling

Sequence-to-Sequence modelling allows one (potentially variable length) input sequence to be translated to another output sequence (potentially of a different length) by transforming the input sequence into a fixed length vector, and using this to drive the generation of the output sequence. There are lots of algorithms and models that can achieve this, but a classic approach is the model proposed by Sutskever et al (`https://arxiv.org/abs/1409.3215`), which uses LSTMs to encode the input to a pair of vectors (the final hidden and cell states), and LSTMs to decode the pair of vectors into the desired output. Sequence-to-sequence models gained popularity as (then) state-of-the-art methods for language translation.

1.1 Complete and train a sequence-to-sequence model (2 marks)

I've implemented a fairly basic sequence-to-sequence model for a translation task and created a dataset from which to train which has the following form:

```
^ .--. .-. . ..-. .- -.-. . / $|^preface $
^ . / ... ..- .--. .--. --- ... $|^e suppos$
```

Each line consists of an input sequence left of the `|` character, and output sequence on the right. The `^` characters represent SOS (Start Of Sequence) tokens, and $ represents EOS (End Of Sequence). I've made the code available to you in a colab notebook here: `https://tinyurl.com/seq-2-seq`.

Unfortunately I didn't quite have time to finish it; it's all complete except for the `forward` method of the `Encoder` class. All the encoder needs to do is pass the input through the embedding and then into the LSTM. The forward method should return the hidden state and the cell state of the LSTM as its output. The output of the LSTM should be ignored as it's not used in this model.

Finish my code and run it to **train** the model. In the report include the small **snippet** of code you had to write and a **plot** of the loss curves duing training.

1.2 now use it! (2 marks)

The following function can be used to make the trained model to translate sequences:

```
def decode(code):
    out = ''
    for chunk in code.split('␣'):
        num = ds.encode_morse('^␣' + chunk + '␣$').unsqueeze(1)
        pred = model(num.cuda(), maxlen=2)
        pred = pred[1:].view(-1, pred_dim).argmax(-1)
        out += ds.decode_alpha(pred.cpu())[::-1]
    return out
```

.- -. ... .-- . .-. / - .... . / ..-. --- .-.. .-.. --- .-- .. -. --.

- .-- .... -.-- / .. ... / - .... . / --- .-. -.. . .-. / --- ..-. / - .... . / --- ..- - .--. ..- - / .-. . ...- . .-. ... . -..
- .-- .... .- - / .. ... / - .... . / .--. --- .. -. - / --- ..-. / - . .- -.-. .... . .-. / ..-. --- .-. -.-. .. -. --.

1.3 Sequence Lengths (1 mark)

The `decode` method in the previous question 'chunked' the input into small blocks before passing to the model. How well does the model work with longer chunks[a]? How does this relate to the training data? (Don't spend long on this; just make one or two brief observations/comments in the report.)

[a] you need to be careful about the position of the chunk boundaries!