Advanced Natural Language Processing

Assignment 3: Character-level Language Modelling with LSTM

Credits:

1 Introduction

In this assignment, you will implement an LSTM model and train it to generate text, one character at a time. (So note: We're asking you to create a character-level model; in the lectures, we've so far only seen word-level models. Think about what the difference is, and what its practical consequences are.)

You should use the PyTorch machine learning library to implement this exercise.

- Instructions to install PyTorch can be found here: `http://pytorch.org/`
- Some PyTorch examples for an in depth overview: `https://github.com/jcjohnson/pytorch-examples`
- Another common quickstart tutorial is this: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

(But don't get carried away: For this assignment, you mostly need the very straightforward elements from the `nn` module in PyTorch that implement the layers that you've learned about, such as RNNS, LSTMs, embeddings.)

This assignment is designed to be runnable on a decent CPU. With a 2-layer LSTM and hidden size of 128, it takes 20 minutes to train while with hidden size of 512, it takes 2 hours. Please take this into consideration while doing this assignment.

Alternatively you can also use Google Colab `https://colab.research.google.com/` by uploading your notebook there, which gives you access to a GPU. (Check that you are indeed using the GPU, via `print(torch.cuda.is_available()`.) However, please keep mind as there is limitation for the free edition (i.e. 'maximum lifetime' of 12 hours).

The goal of this assignment is to get you to specify a simple network, and play around with its hyperparameters to explore how they affect the output. This is why we're providing you with a lot of code, to ensure that the basic housekeeping is taken care of.

2 Getting started

You should start by unpacking the archive `assignment3.zip`. This results in a directory `assignment3` with the following structure:

```
- data/
--- dickens_train.txt
--- dickens_test.txt
- model/
--- __init__.py
--- model.py
- assignment3.py
- utils.py
- evaluation.py
```

- `language_model.py`

3 Dataset

For training, we prepared two text files (train and test) containing passages from Charles Dickens' novels (dickens_train.txt, dickens_test.txt). To prepare the data, we turn any potential unicode characters into plain ASCII by using the `unidecode` package (which you can install via `pip` or `conda`). (What do you think is the use of this step?)

To make inputs out of this big string of data, we will first split it into chunks using `random_chunk()`. Each chunk of the training data needs to be turned into a sequence of numbers (of the lookups), specifically a `LongTensor` (used for integer values). This is done by looping through the characters of the string and looking up the index of each character. In our script this function is realised using `char_tensor()`. Finally, with `random_training_set()`, we can assemble a pair of input and target tensors for training, from a random chunk. The inputs will be all characters up to the last, and the targets will be all characters from the first. So if our chunk is "abc" the inputs will correspond to "ab" while the targets are "bc". Play around with these functions to understand what they do.

4 Starter Code

Let's look at the main program in `assignment3.py`. As this assignment is more exploratory in nature, there is not really a fixed set of order on which function to implement first. However, we recommend you to follow the step below:

1. As a first step, you should implement `LSTM` model.
2. Next, implement `tuner()` function in `language_model.py`, which you will use to experiment with the hyperparameters.
3. Implement `compute_bpc()` in `evaluation.py` in order to evaluate your model during hyperparameter tuning.
4. Lastly, fill in `custom_train()`, `plot_loss()` and `diff_temp` in `language_model.py` to experiment with the available hyperparameters and evaluate your model.

Let's dive into the details one by one!

4.1 Build the Model [40 pts]

The model that you are asked to build will take as input characters up to step t-1 and is expected to produce a distribution over characters at step t (which can then be used to sample one character from that distribution).

There are three layers: one layer that maps the input character into its embedding, one LSTM layer (which may itself have multiple layers) that operates on that embedding and a hidden and cell state, and a decoder layer that outputs the probability distribution.

The beauty of frameworks such as PyTorch is that you can express this pretty directly in code, adding (pre-defined) layers to your network. Implement the `LSTM` model in `model/model.py` which takes `nn.Module` as the parent class. You should also initialise

the hidden and cell state at t=0 with zero vectors with the correct shapes. When you are done, run the main script `assignment3.py` with the flag `--default_train` to check if your implementation is working properly.

We also provide a generator function (`generate()`) that shows you how you can sample from your model (and how we expect the interface to work). The argument `decoder` is your model that is passed into the function. To start generating, we pass a priming string to start building up the hidden state, from which we then generate one character at a time. To generate strings with the network, we will feed one character at a time, use the outputs of the network as a probability distribution for the next character, and repeat. The main training function `train()` is also provided in `language_model.py`. **Explain in your own words what does the code do when you use the flag `--default_train`. What do the defined parameters do? And what is happening inside the training loop?**

4.2 Hyperparameter Tuning [30 pts]

Building neural networks is to some extent more an art than a science. As we have seen above, there are several hyperparameters (i.e., parameters that are not optimized during learning, but that determine the shape of the network), and their setting influences the performance. In this problem, you're asked to tune these hyperparameters (that is, optimize heuristically, rather than using for example stochastic gradient descent). You can try to do this systematically (how?), or just in general explore what changing the parameter does to the performance. (Keep in mind the time it takes to train again for each setting.)

To do so, you need a target. We'll compute bits per character (BPC) over the entire the test set dickens_test.txt using `compute_bpc()`. BPC is defined as the empirical estimate of the cross-entropy between the target distribution and the model output in base 2. As here we are dealing with various hyperparameter settings, we should first implement `tuner()` function that wraps over the training process with adjustable hyperparameters. Keep in mind that `tuner()` will also be used for the next part of the assignments. Afterwards, implement `custom_train()` in order to train several models with different set of hyperparameters and compute BPC for each model (Also don't forget to run the main script using the flag `--custom_train`!)

(Hint 1: You can adapt the formula for word-level cross-entropy given in your text book (chapter 9) to character-level as $-\frac{1}{T}\sum_{i=1}^{T}\log_2 m(x_t)$ where T is the length of input string and x_t is the true character in input string at location t.)

(Hint 2: Tune one parameter at a time)

(Hint 3: Keep a log of your experiments for "parameters used –> minimum loss value")

4.3 Plotting the Training Losses [20 pts]

An important aspect of deep network training task is visualization. Visualizing the training loss values would be helpful for debugging the system. For instance, at extremes, a learning rate that is too large will result in weight updates that will be too large and the performance of the model (such as its loss on the training dataset) will oscillate over training epochs. You would set the learning rate which do not cause oscillation with the help of visual charts.

In this exercise, we ask you to add the loss charts of experiments with different learning rates on the same graph and plot the graph using `plot_loss()`. Add an entry for each experiment to the legend of the graph. If there is more than 10 experiments, use more than 1 chart (up to 10 experiments for each chart). You can use `matplotlib` library for plotting. Don't forget to use the flag `--plot_loss` to execute this part of the assignment and **include the graphs in the submission file**.

4.4 Generating at different temperatures [10 pts]

In the `generate()` function, every time a prediction is made, the outputs are divided by the `temperature` argument passed. In `diff_temp()`, generate strings by using different temperature values and evaluate the results qualitatively. Create chunks from the test set (200 character length as above) and take the first 10 characters of a randomly chosen chunk as a priming string. **Discuss what you observe in the output when you increase the temperature values? In your understanding, why does changing the temperature affect the output as the way you observed?**

In addition, **discuss what are the risks of having a language model that generates text automatically? Who is responsible for the output: the person who builds the model, the person who writes the generating script, the person who uses the outputs? What cautions should we take when using language models for this purpose?**

Use the flag `--diff_temp` to execute this part of the assignment.

4.5 Bonus

If you still have energy / time left once you've reached this place here, try using other datasets (other texts; other types of text, like for example the Linux source code), other layers (e.g., a GRU instead of LSTM), etc. etc..

5 Submission

Upload your `assignment3_LASTNAME.zip` file on Moodle.