Lab3

February 20, 2019

Problem

In this lab, we'll try to perform a sequential classification task using RNN. The task is to predict the country of origin of an input lastname. We'll write a character-level RNN - an RNN which takes one character at one time step - and output the category once it has seen all the input letters of a name. The dataset can be found in `./data/` directory.

The implementation tasks for the assignment are divided into two parts:

1. Writing a dataloader to load the temporal data in an orderly fashion.
2. Designing an RNN architecture using PyTorch's *nn* module.
3. Training the RNN

Below you will find the details of tasks required for this assignment.

1. **DataLoader:** Like last time, we provide a skeleton code for the dataloader. the `__init__()` function reads the data and creates two lists: `self.inputs` and `self.labels`. The `self.inputs` list contains all the names in the dataset shuffled randomly. `self.labels` contains the corresponding country names. Your task is to implement the `__getitem__()` function. It's input is a random index in the range between 0 and number of training images. Note that both the input and labels are currently lists of strings. You need to do the following pre-processing:

   (a) **inputs:** Consider the input three lettered name 'abc'. Assuming there are `n` letters in your alphabet, the preprocessed name should be a numpy array of dimension $3 \times n\_characters$. For every character, create a one-hot vector input of dimension $1 \times n$. Now, since different names can be of different length, you need to appropriately pad the inputs to enable batching of your inputs. **Yes, the dataloader expects every element of the batch to be of the same size.** Find the max number of

letters that a name has in the dataset (18), and create a numpy array of the same size. The padding should be prepended, i.e. if your name has 15 characters and the max length is 18, the first three inputs should be zero and the remaining 15 should be the actual one-hot-encoded vectors.

(b) **labels:** For granting labels, associate every country name with a number between 0 to `n_countries`. This is similar to what we did in Lab1.

At the end of preprocessing, your dataloader should output a $n\_batch \times max\_length \times n$ dimensional input and a n_batch dimensional target tensor.

2. **Designing an RNN:** Please find the skeleton code in model.py file. We have designed a basic RNN network for you. It contains just one hidden layer. You may try playing with the the layer size or number of hidden layers. You may also wish to implement a custom LSTM of your own.

3. **Training:** RNNs need a small tweak in the way the gradients are backpropagated. The inputs need to be fed to the model in a loop, one time step at a time, and, the backprop happens only after the loop is complete. Note that this is where the underlying graph structure plays its role and automatically unrolls the network during backprop. Another detail that needs to be kept in mind is that the hidden state should be initialized before every iteration.

   In this part, fill-in the missing lines in train() function which runs the loop for one epoch. If trained well, the network should reach a training accuracy of 75%.