RNN Exercise

2024

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

1 Coding Homework – Generating Star Trek Dialogues

You can take as starting point the tutorial:
`https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html`,
then use
`star_trek_transcripts_all_episodes_f.csv` as input.

What do you have to modify in the tutorial code?

- Number of categories = 1 for this exercise as you only have one category. So, there will be no category index as input
- Use an LSTM, replace the custom RNN (what works well: 2 or 3 layers of LSTM with 100 or 200 hidden dimensions)

reading the csv:

```
    all_letters = string.ascii_letters + "0123456789 .,;:!?'[]()/+-="

def get_data():
  category_lines = {}
  all_categories = ['st']
  category_lines['st']=[]
```

```
filterwords=['NEXTEPISODE']

with open('./star_trek_transcripts_all_episodes.csv', newline='') as csvfile:
  reader = csv.reader(csvfile, delimiter=',', quotechar='"')
  for row in reader:
    for el in row:
      if (el not in filterwords) and  (len(el)>1):
        print(el)
        v=el.strip().replace(';','').replace('\"','')
        category_lines['st'].append(v)

n_categories = len(all_categories)
print(len(all_categories), len(category_lines['st']))
print('done')

return category_lines,all_categories
```

This gives you a dictionary with only one key and that key contains a huge list with many star trek TOS sentences.

You can use also the raw data `star_trek_transcripts_all_episodes.csv` but there you may need to filter more special characters. You are allowed to manually clean the csv.

1.1 What else to program

- implement a character level RNN for the star trek monologues. Model suggestion:
  - LSTM 2 or 3 layers, 100 or 200 or 300 units
  - dropout with 10% probability
  - fully connected (hidden units → number of output tokens)
    - number of output tokens = alphabet plus EOS
  - if you use NLLLoss, then a logsoftmax layer
- implement a temperatured sample
- after every 2k to 5k samples in training and after every full epoch sample 10 to 20 different samples. capture that output (e.g. in a .txt-file), for me a temperature of 0.5 worked okay
- measure accuracy on the test set every epoch
- copy off that .txt file and the model before you shutdown the GPU session.
- sampling: what works well for me is a temperature of 0.5. You can use a subset of all letters as starting letters. I suggest only CAPITAL letters,

no Q or XY, as all the start trek dialogues start with the name of the speaker in capitals.

- you are **not** required to use a batchsize > 1, it makes things faster though

1.2 What to plot

- your code, your trained model (can save the model dictionary of parameters rather than the model as a whole).
- the .txt file of outputs of the sampling from the moodel (maybe 5 samples every epoch) for each of your epochs
- plot a graph of train and test loss over epochs
- report accuracy per epoch – accuracy here is the prediction of the next character in a sentence.
- **Report the best quote you have seen !** We will vote your quotes with a prize in class.

```
best_model_wts = net.state_dict()
torch.save(best_model_wts,'./model.pt')
```

2 Some help

Similar approach as with the exercise on classifying names.

some helper to read if you want to use it with batch size larger than 1:
`https://towardsdatascience.com/taming-lstms-variable-sized-mini-batches-and-why-pytorch-is-good-for-your-health-61d35642972e`

- `category_lines[category]` is a list of names for the particular `category`, in this code you have only one category, and it would be a list of words or sentences instead here.
- `def randomTrainingPair():` returns a random pair of category and word/sentence, here it needs to return only a word/sentence.

3 Going beyond

THIS IS NOT PART OF THE HOMEWORK.

You can use your saved model to play with the sampling temperature.

You can even implement an approach which assigns a higher sampling temperature whenever the input was a space or at the start of a sentence.

The character level rnn gets words right, but not really their ordering. Since star trek uses 20th century english, you can judge that. You can try on your own to do a word-level modelling with word embeddings, but that will be much harder to achieve (one needs usually larger corpora). Modern approaches combine both ideas. One can learn hierarchical rnns (tree-LSTMs for example, where folding of batches is more complicated). You can use your code to play with.

Babbling words is not so bad and can make one achieve a lot, compare with successes of certain politicians on other continents.

For fun: you can model an RNN for philosophy texts if they are of sufficient length `http://www.philosophy-index.com/texts.php` and the like. Problem is quality judgments ...

You can improve this modeling by doing this with character embeddings, or with word embeddings as a RNN on a word level.