F20ML 2020-21: Coursework 3 [20%]

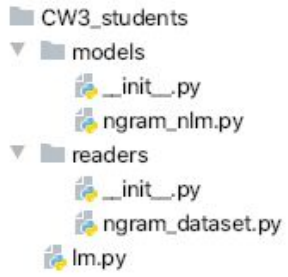
Language Modelling using Deep Learning with Keras

The coursework consists of two parts: i) this **description**, ii) an accompanying **folder with the dataset and several .py files** containing the necessary scaffolding for you to implement the missing parts in order to answer the questions below. **You will NOT have to submit a document report in .pdf or .doc but instead you will enter your responses (upload the completed .py files in a zipped folder and a couple of plot images) in the corresponding Test created on Vision. Read the instructions in the last Section 'Deliverables' below for more details.**

Note: We will use the existing implementations for deep learning from the Python library **Keras**, in order to implement the Language Model and complete all the questions in this coursework. **All the material from Week 9, especially Lab 9, is going to be particularly useful.**

Structure of the Code

The structure of the project should look like this:



a) The main entry point is `lm.py`. There we 1) load the dataset as an instance of the class `NGramDataset`, 2) build and train the model as an instance of the class `LanguageModel` and 3) call various functions that correspond to exercises below, namely, `exercises_sonnet()` and `exercises_toy()`.

b) The dataset reader class `NgramDataset` is in `readers/ngram_dataset.py`. This class loads either of the two datasets: **'Sonnet'**, which is Shakespear's Sonnet 2 represented as a single string, i.e., not split into sentences/verses, or **'Toy'**, which is a list of 5 simple, similar but also crucially different sentences. It contains 5 methods:

i) `__init__()` : the constructor holds a) an instance of `Tokenizer()` **`[self.tokenizer]`** (useful for converting to and from strings and indexes) and b) the dataset as a 2D numpy array of ngrams **`[self.X]`.**

ii) `preprocess()` : takes a sentence of words as a string and prepends/appends special 'start of sentence; <sos>' and 'end of sentence; <eos>' symbols to the beginning and end. This enables better calculations of probabilities of words that typically appear at the beginning and end of sentences.

iii) `vectorize()` : takes a **list of** sentences of words and converts them to a **list of** sequences of indexes using `Tokenizer.texts_to_sequences()`.

iv) `get_ngrams()`: returns a **list of** all ngrams in a sequence of indexes.

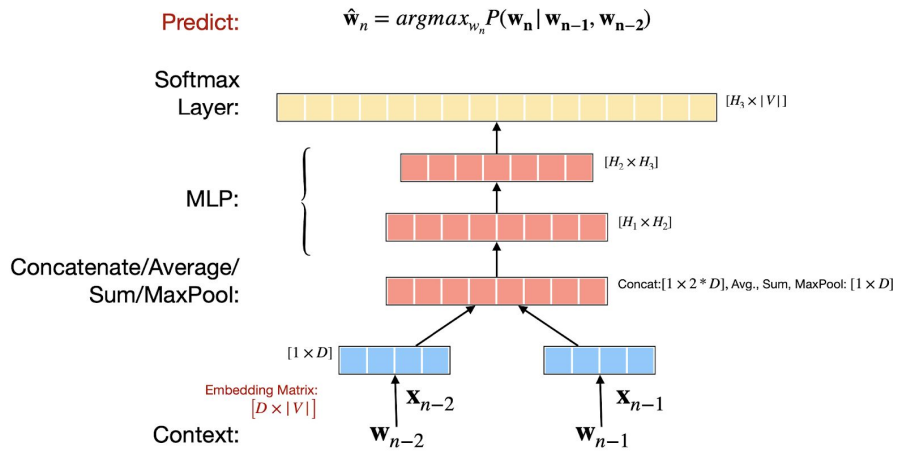
v) `load()`: Loads either of the two datasets by a) *preprocessing* each string/sentence, b) *creating* a vocabulary stored in `self.tokenizer` (accessed via `self.tokenizer.word_index` or

`self.tokenizer.index_word`) and c) *extracting* all the ngrams stored in `self.X`.

c) Our implementation of the language model and various helper methods goes inside the class `LanguageModel` of `models/ngram_nlm.py`. You will need to finish the implementation of `__init__()`, `train()`, `predict()`, `generate()`, `sent_log_likelihood()` and (MSc students only) `fill_in()`.

Exercise 1 - Fit *Sonnet* Language Model

[Implement `__init__()` and `train()` in `models/ngram-nlm.py`]



Implement a 3-gram Neural Language model (as shown in the image above) and fit it on the Sonnet dataset. The hyperparameters (ngram_size and number of epochs) can be found in the beginning of `exercises_sonnet()`. The architecture is very similar to the Neural Text Classifier found in Lab 9; the main difference is that the final Softmax layer needs to predict words from the vocabulary, instead of categorical classes. You can experiment with various projection layers.

In terms of hyperparameters we suggest using a very small model: embedding dimensionality D between 10 and 100, 1 or 2 MLP Layers with about 10 units each.

Perform hyperparameter tuning (1. tune the number of epochs first, then 2. adjust D, leaving the rest fixed): you should try to **overfit** your model.

Plot 2 training curves against the number of epochs using `plot_history()` for 2 different values of D (Just upload two files).

Briefly explain what is going on between these plots.

Exercise 2 - Predict with the *Sonnet* Language Model

[Implement `predict()` in `models/ngram-nlm.py`]

Using the trained model, implement the prediction of the **next word** and its **probability score** given the bigram *'all the'*, i.e., compute **argmax P(w | 'all', 'the')**.

`LanguageModel.predict()` takes as an input argument the context, which corresponds to an ngram, represented as a list of indexes. Report on the Vision Test just the next word and its probability score.

Hint: You will need to use `model.predict()` (see the API here).

Hint 2: Use `dataset.tokenizer.index_word` and `dataset.tokenizer.word_index` !

Exercise 3 - Generate with the *Sonnet* Language Model

[Implement `generate()` in `models/ngram-nlm.py`]

Time to have fun and let the machine... speak in prose!

Implement a text generator by using `predict()` from Exercise 2; start with the bigram '<sos> <sos>' and generate 20 words. The idea behind the generator is to keep feeding as input to `predict()` a new ngram that contains words from previous predictions. Let's try to explain this with a fictitious example. Let's assume that you have already generated the sequence:

`'<sos> <sos> The rain in'`

1. Inside `generate()` you should call `predict()` with the argument `context`=(the indexes of) `[rain, in]`.
2. `predict()` should return the most likely word: `Spain` Your sequence becomes:

`'<sos> <sos> The rain in Spain'`

3. Now inside `generate()` call `predict()` with the argument `context`=(the indexes of) `[in, Spain]`.
4. Repeat until you have reached 20 words.

a. Copy-paste your output on the Vision Test.
b. Try with a less overfit model (e.g., adjust the number of epochs). What does your output look like now? Briefly justify your answer.

Exercise 4 - Fit and Predict with the *Toy* Language Model

[Complete the implementation in `exercises_toy()` in `lm.py`]

a. Fit a 3-gram Language Model for the Toy Language Model. You should write the scaffolding on your own. Perform Hyperparameter tuning (similar to Exercise 1) and plot **one** training loss histogram for your best combination of hyperparameters (Just upload the file).
b. Using the trained model (write the scaffolding on your own in order to)
   i) predict the **next word** and its **probability score** given the bigram *'<sos> the'*,
   i.e., compute **argmax P(w | '<sos>, 'the')**.
   ii) Why is the next word *'thief'* and not *'crook'*? Justify briefly your answer by looking at the dataset and the predicted probabilities.

Exercise 5 - Sentence Likelihood with *Toy* Language Model

[Complete the implementation in `sent_log_likelihood()` in models/ngram-nlm.py]

Implement the computation of the full log-likelihood of a sentence $(w_1, w_2, ..., w_N)$,

$P(w) = \prod_{n=3}^{N} P(w_n | w_{n-1}, w_{n-2})$.

Given two sentences:
S1: 'The thief stole the suitcase,'
S2: 'The crook stole the suitcase.'

Compute which is more likely? Briefly justify your answer by looking at the sentences and the computed log-likelihoods.

Exercise 6 - Trained Word Embeddings with *Toy* Language Model

[Complete the implementation in `exercises_toy()` in `lm.py`]

Let's look at the word embeddings we've learned as a by-product of the Language Model. Do they really learn any semantic features?
Implement the computation of the **cosine similarity** of the word embeddings of 'thief' and 'crook' and compare with the **cosine similarity** of the embeddings of 'thief' and 'cop'.

a. Which pairs are closer?
b. By looking at the dataset, why do you think it is that?

Briefly justify your answers.

Hint: Use `LanguageModel.get_word_embedding()` to get the word embedding of a word; it takes a single argument with the word represented as a string.

Deliverables

You will have to **submit the Test in 'Assignments>Coursework 3 - Test' on Vision**. Over there, you should (i) upload the completed source code in a .zip file: PLEASE don't include the dataset files! (**it should be named: Coursework_3_H0XXXXXXX.zip, by replacing the Xs with your Student ID**) , (ii) enter your answers to the questions outlined in the description above, (iii) upload a few of plot images as part of your answers clearly indicated within the test. It is advisable to properly comment your code. **Note**: Any 3rd party source used must be properly cited (see also below).

Important Notes!

- **The assignment counts for 20% of the course assessment.**

- You are permitted to discuss the coursework with your classmates and of course with me and the teaching assistant. However, coursework reports must be written in your own words and the accompanied code must be your own. If some text or code in the coursework has been taken from other sources, these sources must be properly referenced. In particular, for pieces of code you get from the web (e.g., from StackOverflow), minimally you should provide the link where you found it as an inline comment in the jupyter notebook. Failure to reference work that has been obtained from other sources or to copy the words and/or code of another student is **plagiarism** and if detected, this will be reported to the School's Discipline Committee. If a student is found guilty of plagiarism, the penalty could involve voiding the course.
- You should **never give** hard or soft copies of your coursework report or code to another student. You must always refuse any request from another student for a copy of your report and/or code. Sharing a coursework report and/or code with another student is **collusion**, and if detected, this will be reported to the School's Discipline Committee. If found guilty of collusion, the penalty could involve voiding the course. You can find more information about the University's policy on Plagiarism here: https://www.hw.ac.uk/uk/students/studies/examinations/plagiarism.htm
- **Pay special attention** to all the **Labs** as they should provide lots of insight as to how to tackle most of the questions in this coursework. The **Final Lab** is specifically designed for you to ask any questions related to the coursework. Also, **CHECK THE DOCUMENTATION OF scikit-learn and Keras** when you are in doubt.
- The test should be submitted by **15:00 UK time on Friday 4 December**. You will find the coursework under the 'Assessment' section of the course Vision page. No individual extensions are permitted under any circumstances. Students who submit after the deadline but within 5 working days of the deadline will be award 0.7x(awarded coursework mark). Submissions that are more than 5 days late will receive 0 marks.
- You will receive the final mark, answers, sample code solution and cohort-wide feedback no later than 15 working days.