HW Word2Vec

Alexandre Allauzen & Florian Le Bronnec

October 12, 2023

1 General instruction

1.1 Logistic

Due date. 5 of November 2023

Submission. You are expected to submit **three files**.

1. A **written PDF report**.
2. The **code for the Word2Vec model**, in a **Jupyter notebook** from the supplied template.
3. Another **Jupyter notebook** with the application of the model to classification.

Report guidelines.

- The report should contain the answers to the questions asked in Section 2.
- It should also feature a general summary on your implementation.
- In particular, you are expected to justify each decision you made in the implementation of the model that is not imposed.
- You should also report and analyze the results you obtained (with plots and/or tables).
- It should contain a small conclusion on what you learned by doing this HW.
- The number of pages should be **at most 3** (no minimum).

Coding guidelines. The only requirement is that you submit the notebooks **do not return any error when ran on Google Colab**. Any effort made to make the code readible and easy to follow will be valued. Here are some hints.

- This does not mean to put too much comments, but more likely to write understandable code.

- Choose meaningful variable names.
- Write modular code, with meaningful function names. If necessary put a quick docstring in your functions indicating what it does.
- Print intermediary results, like shapes or a small number of samples in explanative cells.
- Typing is not mandatory, though this is a general good practice and can help for the readibility.
- Use `black` on your final notebook (see `https://black.readthedocs.io/en/stable/getting_started.html`).

1.2 Introduction

You are expected to implement from scratch the Word2Vec model, using PyTorch.

Word2Vec is a contrastive representation learning model that learns word embeddings. You'll find below the exact formulation of the Word2Vec algorithm you are expected to implement.

- Sample a word w from the dataset $\mathcal{D}$, from a random document. Let i be the index of w in this document.
- Take the words surrounding w in a local window of radius R in the document, they'll make the positive context:

  $$C^+ = [w_{i-R} \ \dots \ w_{i-1} \ w_{i+1} \dots \ w_{i+R}].$$

  This yields $2R$ positive examples.
- The usual implementation of Word2Vec uses more negative examples. This is parametrized by the integer factor K. Sample $2KR$ random words in the vocabulary $\mathcal{V}$, that will make the negative examples, C^-.
- Map the word w to a vector $\boldsymbol{w} \in \mathbb{R}^d$ through an embeddings table E^w.
- Map each word in C^+ and C^- to vectors $\boldsymbol{C^+} \in \mathbb{R}^{2R \times d}$, $\boldsymbol{C^-} \in \mathbb{R}^{2KR \times d}$ through the context embeddings table E^C.
- Compute a similarity score between the contexts and w. For a context word $c \in C^+ \bigcup C^-$, the similarity score is given by $\sigma(\boldsymbol{c} \cdot \boldsymbol{w}) \in \mathbb{R}$, where $\sigma(x) = \dfrac{1}{1+e^{-x}}$ is the sigmoid function.
- We therefore have a model M that maps a word w, another word c from a positive (resp. negative) context $c \in C^+$ (resp. $c \in C^-$) to a scalar $\sigma(\boldsymbol{c} \cdot \boldsymbol{w})$. The loss will be the one of binary classification, for a word w, a context word c:

  $$L(M(w,c)) = -\mathbf{1}_{c \in C^+} \log\big(\sigma(\boldsymbol{c} \cdot \boldsymbol{w})\big) - \mathbf{1}_{c \in C^-} \log\big(1 - \sigma(\boldsymbol{c} \cdot \boldsymbol{w})\big), \quad (1)$$

where $\mathbf{1}_{c \in C^+}$ (resp. $\mathbf{1}_{c \in C^-}$) is equal to 1 if c is a positive (resp. negative) context of the word w.

2 Preliminary questions

1. To compute the similarity between a word and a context, we use the similarity $\sigma(\boldsymbol{c} \cdot \boldsymbol{w})$. Since, we want to minimize the loss (1):
   (a) for $c \in C^+$, should we maximize or minimize $\sigma(\boldsymbol{c} \cdot \boldsymbol{w})$?
   (b) Same question for $c \in C^-$.

   Give a simple geometrical interpretation.
2. One of the first application to contrastive learning has been presented by Chopra, Hadsell, and LeCun [1].
   (a) By reading the introduction of this article, vulgarize (i.e., explain very simply) what is contrastive learning.

   Page 3 of this article, you'll find the expression:

$$L\big(W, (Y, X_1, X_2)^i\big) = (1 - Y) L_G\big(E_W(X_1, X_2)^i\big) + Y L_I\big(E_W(X_1, X_2)^i\big). \tag{2}$$

   This is very similar to what we are doing. In our setup described above, explain here:
   (b) What is the analog of Y?
   (c) What is the analog of E_W? Why isn't it the same?
   (d) What are the analogs of L_G and L_I?

3 Implementation

This section aims at guiding you through the construction of your own Word2Vec model. You are required to follow these instructions. For readibility purposes, when possible, comment which part of the code refers to which question.

Make sure you understand the experimental setup described in section 1.2.

3.1 Data preprocessing

1. The first cells of the notebook are the same as in the TP on text convolution. Apply the same preprocessing to get a dataset (with the same tokenizer) with a `train` and a `validation` split, with two columns `review_ids` (list of int) and `label` (int).

2. Write a function `extract_words_contexts`. It should retrieve all pairs of valid (w, C^+) from a list of ids representing a text document. It takes the radius R as an argument. Its output is therefore two lists:
   - The first one contains the ids of w.
   - The second one contains the list of ids of C^+, within the local window, corresponding to w.

   Make sure that every C^+ has the same size (i.e., contains the same number of ids). Explain clearly how you handle the borders (every strategy can be valid, as as long as it is justified).
3. Write a function `flatten_dataset_to_list` that applies the function `extract_words_contexts` on a whole dataset.
4. Apply the function to your initial `document_train_set` and `document_valid_set`, and get the corresponding flattened lists.
5. Embed these lists in two valid PyTorch `Dataset`, like in HW 1, call them `train_set` and `valid_set`.
6. Write a `collate_fn` function that adds the negative context to the batch. It should be parametrized by the scaling factor K. The output of `collate_fn` should be a Python dictionary, with three keys:
   - `word_id`,
   - `positive_context_ids`,
   - `negative_context_ids`.

   Make sure that each value of the dictionary is a valid `torch.Tensor`. *Hint: for the negative context, simply randomly sample from the whole vocabulary set.*
7. Wraps everything in a `DataLoader`, like in HW 1.
8. Make 2 or 3 three iterations in the `DataLoader` and print R, K and the shapes of all the tensors in the batches (let the output be visible).

3.2 Model

9. Write a model named `Word2Vec` which is a valid `torch.nn.Module` (i.e., write a class that inherits from the `torch.nn.Module`), and implement the Word2Vec model. It should be parametrized by the vocabulary size and the embeddings dimension. Use the module `torch.nn.Embedding`.
10. Train the model. The training should be parametrized by the batch size B, and the number of epochs E.
11. Validates its accuracy on the test set.

12. Write a function `save_model` that saves the model's embeddings in a file. The file name should be formated like:

    `"model_dim-<d>_radius-<R>_ratio-<K>-batch-<B>-epoch-<E>.ckpt"`.

13. Once you have a working code, you can launch a bigger training, using more documents, if it does not take too much time.

3.3 Classification task

In this section you will experiment with the classification task of the lab, augmented with your Word2Vec model.

Use the notebook from the lab, with the dataset and the training script.

1. Write a function `load_model` that takes a path to a saved Word2Vec embeddings (with the previous formatting) and loads the checkpoint the embeddings directly to the `ConvolutionModel` (you can use either the state-of-the art model or the first small model).
2. Train the model, initialized with these emebeddings.
3. Compare the results with the model without this initialization.
4. Make a small ablation study on the influence of some parameters of the Word2Vec model on the classification task. Analyze the results.

References

[1] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 539–546 vol. 1. DOI: 10.1109/CVPR.2005.202. URL: http://yann.lecun.com/exdb/publis/pdf/chopra-05.pdf.

[2] HuggingFace. *HuggingFace Hub*. URL: https://huggingface.co/datasets.