Deep Learning in NLP

Homework P6

Solution to be sent (pdf, zip) to `waszczuk@phil.hhu.de` and `cwurm@phil.hhu.de` by 26.11.2019 (UPDATE: **before the session**!).

Introduction

The overall goal today is to improve our implementation of the language prediction model with *batching*. By batching, we mean an adaptation of the code which allows to apply our model to batches of inputs. In our particular case, this will allow to predict language scores for several names in parallel.[1]

Batching makes the code more difficult to understand, but it also allows to significantly speed it up. In research papers, only the versions of neural models with batching are often formalized and explained, leaving it to the reader to infer/guess how the model works on a single input...

Preparation

Download the code from the course webpage, unpack it, and open in VSCode (or your other favorite editor). The code contains a working solution (together with the dataset), so you can enter `ipython` and run grid search straight away:

```
In [1]: run main
In [2]: main()
Train size: 16059
Dev size: 2008
# emb_size=10, hid_size=10
...
```

Some parts of the code are already „batching-enabled". In particular, in the `main.py` module, the `forward` method of the `LangRec` class, as well as the `batch_loss` function already work with batches.

[1] http://dynet.io/ provides *auto-batching*: it performs batching automatically for the user. But this feature is not available in more popular frameworks.

Git

If you want to use `git` to keep track of changes in the `hhu-dl-materials` repository, see the appendix at the end of this document.

The code for the today's session is in the `lang-rec-batch/code` directory of the repository. Note that the github repository only contains code, you will have to download the datasets separately.

Exercise 1

The goal of the first exercise is to write a batching-enabled version of a feed-forward network with a single hidden layer (we call it FFN in the rest of the document). We will implement it in the `fnn.py` module.

Regular FFN (reminder)

A linear layer L with input size n and output size m has two parameters:

- $\mathbf{M} \in \mathbb{R}^{m \times n}$: matrix corresponding to the linear transformation
- $\mathbf{b} \in \mathbb{R}^{m}$: the corresponding bias vector

Given the input vector $\mathbf{x} \in \mathbb{R}^{n}$, the forward calculation formula is:

$$L(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{b} \tag{1}$$

FFN F can be defined as a combination of two linear transformation layers L_1, L_2, with a non-linear element-wise activation function σ applied in between. Formally:

$$\begin{aligned} F(\mathbf{x}) &= L_2(\mathbf{h}), \text{ where} \\ \mathbf{h} &= \sigma(L_1(\mathbf{x})) \end{aligned} \tag{2}$$

Alternatively, using function composition:

$$F(\mathbf{x}) = (L_2 \circ \sigma \circ L_1)(\mathbf{x}) \tag{3}$$

Batching-enabled FFN

The goal of the first exercise is to implement a batching-enabled FFN. Our point of departure is Eq. 3 (or the equivalent Eq. 2). To make it work in batches, we basically need to adapt a linear layer to work in batches, too.

Let b be the batch size and $\mathbf{X}$ the matrix with input vectors, one vector per row. Hence, $\mathbf{X}_{[1]}$ is the first input vector, $\mathbf{X}_{[2]}$ the second input vector, etc.[2]

Let L be a linear layer. Its batching-enabled version L^* should satisfy the following equation:

$$\forall_{i=1}^{b} L(\mathbf{X}_{[i]}) = L^*(\mathbf{X})_{[i]} \tag{4}$$

[2]Don't forget that, in comp. sci., indexing starts with 0...

Given a FFN F, the formula for a batching-enabled FFN F^* is then:

$$F^*(\mathbf{X}) = (L_2^* \circ \sigma \circ L_1^*)(\mathbf{X}) \tag{5}$$

You can check that, provided that Eq. 4 holds, the following equation is also satisfied. In the code, this is checked using doctests.

$$\forall_{i=1}^{b} F^*(\mathbf{X})_{[i]} = F(\mathbf{X}_{[i]}) \tag{6}$$

Your job is to implement a batching-enabled linear layer in the `ffn.py` module. A batching-enabled FFN is already implemented. The missing parts are marked with `TODO`s.

Hints:

- Try replacing the matrix-vector product (`torch.mv`) with the matrix-matrix product (`torch.mm`). Be careful with the order of the arguments!
- Try implementing a linear layer **without** the bias vector first. Once it works, you can include the bias vector by adding it iteratively to each row in the resulting batch matrix. Is this solution optimal?

1b (optional)

Once you implement the batching-enabled linear layer in the `ffn.py` module, shortly explain how your code works and try to briefly show that it satisfies Eq. 4.

Exercise 2 (optional)

Using a batching-enabled variant of FFN is not enough to make our code really fast, because another part of our language recognition model – namely, embedding n-gram features with vectors – still works in a sequential manner.

2a

An optimized variant of Embedding, `EmbeddingSum`, which is combined with CBOW and which works on feature groups is implemented in the `embedding.py` module. Your job is to incorporate it in the rest of the code. To this end, you will need to modify the `LangRec` class in `main.py`. The placeholders are marked with `TODO EX2`.

2b

`EmbeddingSum` works over feature groups, but it does not work over batches of feature groups yet. Can you think of (or implement) a way to adapt it to work with batches?

Warning: exercise 2b may require more changes in the code than other exercises.

A. git

Instead of downloading the code manually, you can use `git` to keep track of changes in the `hhu-dl-materials` repository.

On linux, you will first have to clone the repository to a local directory:

```
git clone https://github.com/kawu/hhu-dl-materials
cd hhu-dl-materials
```

Then, each time you want to sync with the github repository:

```
git pull
```

The command above has to be run in the local `hhu-dl-materials` directory.