Deep Learning in NLP

Practical Session P3

Note: This is *not* a homework, the homework is theoretical this time (have a look at the website). This document describes the exercise(s) we are going to do/did during the practical session.

Preparation

Download the archive file with the code and the dataset from the course's website. Unpack it and open in VSCode (`File` → `Open Folder`).

Problem

The problem is to classify person names (Downie, Brune, Dubrov, Fontaine, etc.) according to their languages (English, German, Russian, French, etc.).

Dataset

We will use the development dataset (in the `.csv` format), obtained as a result of homework P1, divided in two parts.

The main reason for not using the entire train+dev set is that training with a gradient descent (GD) is pretty slow. We will discuss solutions to the efficiency problem (stochastic GD, batching) next week.

Task

Our goal is to implement a (character-level, continuous-bag-of-words) PyTorch model for the above-mentioned language classification task. Formally, let C be the set of characters. Given a name $(c_i \in C)_{i=1}^{|c|}$:

- We first map each character c_i to a vector $v_i \in \mathbb{R}^n$ (n is a hyper-parameter) using an embedding function $e\colon C \to \mathbb{R}^n$. The vector representations of the individual characters are to be learned (with other parameters of the network) during training.

- The one network architecture we've seen so far is a feed-forward network (FFN). Given a vector representation of a person name, $v \in \mathbb{R}^n$, we can use a FFN to transform v to a score vector $w \in \mathbb{R}^m$, where m is the number of languages. To perform classification, we pick the language with the highest score.
- However, there is a mismatch between the output of the embedding layer (sequence of vectors $(v_i \in \mathbb{R}^n)_{i=1}^{|c|}$) and the input of the FFN (single vector $v \in \mathbb{R}^n$). To overcome this problem, the easiest solution is to use the continuous bag of words (CBOW) representation:

$$v = \sum_{i=1}^{n} v_i$$

Exercise 1

The goal of Ex. 1 is to fill the gaps, marked with `TODO`s, in the provided code. Before you try doing that, go through the code and try to understand the individual pieces and modules. Here are some hints:

- `core.py`: some core types and functions (not much for the moment).
- `module.py`: abstract representation of a parametrized network component.[1]
- `ffn.py`: implementation of FFN in terms of a network module.
- `embedding.py`: simple implementation of an embedding dictionary, which maps symbols from a pre-determined alphabet to vectors.
- `encoding.py`: isomorphic mapping between classes (English, German, etc.) and integers $(0, 1, ...)$.
- `names.py`: dataset-related functionality (loading).
- `main.py`: the main Python module, with the definition of the overal model, training procedure, etc. Most of the missing pieces are in `main.py`.

Exercise 2

The solution to Exercise 1 is a bit weak in that it completely ignores the order of the characters in the given name (e.g., *leon* and *noel* will both get the same scores). This is because:

- It uses *unigram* character-level features, therefore local ordering information is lost.

[1] The choice to call it a *module* is not perfect, but we follow the naming scheme of PyTorch here.

- It uses CBOW to represent words, which also completely ignores both local and non-local ordering information.

We will later see how to improve on the second point. Now, the goal is to improve the first one. The task is to:

- Implement a *bigram* model in which each pair of adjacent characters is embedded separately. Then, to obtain the word (name) representation, CBOW can be used.
- See if the bigram-based input representation improves the accuracy on the dev set (`dev20.csv`)