IN5550 – Spring 2024
Obligatory 1. Document Classification

UiO Language Technology Group

Deadline: 23 February, at 21:59 (Devilry)

Goals

1. Learn how to use the Fox cluster to train deep learning models.
2. Learn how to use *PyTorch* to train and evaluate neural classifiers in NLP tasks.
3. Learn how to integrate pre-trained word embeddings in an arbitrary neural architecture.
4. Apply recurrent neural architectures (RNNs) to a classification task.

Introduction

This is the first obligatory assignment in the IN5550 course, Spring 2024. Your goal is to familiarize yourself with training neural networks on textual data. You will have to address a classification task with different architectures (of different complexities). In *Part 1* you will work with bag-of-words as features, in *Part 2* you will use pre-trained word embeddings and in *Part 3* you will employ recurrent networks.

The recommended machine learning framework in this course is *PyTorch*; we provide a ready-to-use module and Python environment on the Fox cluster,[1] and we strongly encourage you to train your models using Fox. If you would like to use any other libraries, (i.e. *Tensorflow* or *Jax*), this is allowed, provided that you implement your own classifier from scratch, and that your code is available.

Please make sure you read through the entire assignment before you start.

[1] `https://www.uio.no/studier/emner/matnat/ifi/IN5550/v24/setup.html`

Solutions must be submitted through Devilry[2] by **21:59** on **February 23**. Please upload a single PDF file with your answers. Your heavily commented code and data files should be available in a separate private repository (that you will continue to use for your other assignments) on UiO's Git platform.[3] Make your repository private, but allow full access to IN5550 teaching staff.[4] The PDF in Devilry should contain a link to your repository. If you have any questions, please raise an issue in the IN5550 Git repository,[5] email *in5550-help@ifi.uio.no*, or ask on the Mattermost chat. Make sure to take advantage of the group sessions on February 13 and 20.

Recommended reading

1. **Neural network methods for natural language processing.** Goldberg, Y., 2017.[6]
2. **Speech and Language Processing**. Daniel Jurafsky and James Martin. 3rd edition draft of February 3, 2024. Chapter 5, 'Logistic Regression' [7]; Chapter 7 'Neural Networks and Neural Language Models'[8], sections 7.1, 7.2, 7.3, 7.4, 7.6; Chapter 6 'Vector Semantics and Embeddings'[9], Chapter 9 'RNNs and LSTMs'[10].
3. Linear Algebra cheat sheet by the LTG[11]
4. `https://radimrehurek.com/gensim/models/word2vec.html`
5. `https://pytorch.org/`

[2] `https://devilry.ifi.uio.no/`

[3] `https://github.uio.no/`

[4] `https://github.uio.no/orgs/in5550/people`

[5] `https://github.uio.no/in5550/2024/issues`

[6] `https://www.morganclaypool.com/doi/10.2200/S00762ED1V01Y201703HLT037`

[7] `https://web.stanford.edu/~jurafsky/slp3/5.pdf`

[8] `https://web.stanford.edu/~jurafsky/slp3/7.pdf`

[9] `https://web.stanford.edu/~jurafsky/slp3/6.pdf`

[10] `https://web.stanford.edu/~jurafsky/slp3/9.pdf`

[11] `https://www.uio.no/studier/emner/matnat/ifi/IN5550/v24/la.pdf`

1 Dataset and classification task

You will be working with a subset of the *Arxiv-10*[1] dataset. This is a dataset of 80,000 scientific articles in English, containing the article abstracts and the field of the research (10 subcategories of computer science, physics, and math are considered). Your task will be to predict the field of research based on the abstract of the paper. Hence, you will have to address a multiclass classification task.

The training data is available in a subdirectory of `/fp/projects01/ec30/IN5550/` on Fox[12]. It is provided as a gzipped comma-separated text file, with one data instance corresponding to a paper. It has two columns:

- **abstract** (input)
- **label** (output)

We also have a held-out test set of 20,000 papers, which will not be published until we receive all the submissions. Your solutions will then be evaluated on this held-out test set, so that we can rank the systems by their objective performance.

We provide the dataset as is, but before training you have to split it into train and development parts. Make sure to keep the distribution of labels as similar as possible in both subsets. You can use the scikit-learn library, HuggingFace Datasets library, or write your own code, using PyTorch's `torch.utils.data` as a base if necessary. Describe the process of splitting the dataset in your PDF report in detail.

2 Part 1: Bag of words baseline

In the first part you will have to implement a simple baseline that you can later compare with more complex architectures.

You need to train a classifier which will be able to predict the field of a paper based on the abstract. For this, you need to somehow move from abstracts as character sequences to abstracts as feature vectors that can be fed to a machine learning algorithm, such as a feed-forward neural network.

A simple way to do this is to represent all the abstracts as bags-of-words: that is, extract the collection vocabulary (the union of all word types[13] in the texts)

[12] `/fp/projects01/ec30/IN5550/obligatories/1/arxiv_train.csv.gz`

[13] To recall, a 'word type' is the opposite of a 'word token': that is, a unique entry in the vocabulary. For example, the sentence 'the cat is chasing the rat' has 6 word tokens but 5 word types (word type 'the' is used twice).

and count frequency of each word type in each abstract (or mark the binary value of whether the word type appears in the abstract or not). Abstracts are then represented as sparse vectors of the size equal to the size of the vocabulary. Machine learning classifiers can be trained on such data, which is what you are going to do.

For this part you must write the code which takes the provided dataset and:

- infers a common vocabulary for all abstracts in the dataset;
- extracts bags of words for each abstract;
- extracts a list of gold labels/fields for the papers (for example, `field[i]` contains the correct field for the i_{th} paper);
- all this serves to prepare the data to be fed into a classifier.

Briefly describe this code in the report.

Then implement a fully-connected feed-forward neural network (multi-layer perceptron) which trains on these bag-of-words features and evaluate it on the development dataset (see the 'Evaluation' subsection below). For a start, you can look at the classifier code discussed at the group sessions.

You **should experiment with**:

- Different variations of BoW features (at least three)
- Different number of layers

You should document your experiments in the report, providing explanations for what was varied in each experiment and reporting the obtained results.

3 Part 2: Word Embeddings

In this section you will transform the documents into representations that can be processed by neural networks by employing word embeddings. More concretely, you will pass the input text through a pretrained embedding model, leading to a (variable) number of vectors associated to each input. Since in this section you are using a simple feed-forward architecture, this diversity needs to be flattened into 'sentence representations' with a pre-defined shape.

In this part you will have to write code to:

- load pre-trained word embeddings for the input text
- merge the sequences of word embeddings into a fixed-size vector
- apply a classifier over this fixed-size vector

Additionally, you also have to pre-train your own embeddings on a corpus of your choice. You can use any English corpora available on the net or on Fox[14] (not less than 10 million word tokens), but you should justify your choice (why is it relevant for the task and dataset at hand?).

You **should experiment with**:

- Different ways of merging the sequence of embeddings into a fixed-size vector representation: averaging them, summing them, etc.;
- Different ways of dealing with out-of-vocabulary words;
- Different pre-trained embedding models (built using different algorithms: *word2vec, fasttext, GloVe*);
- Existing pre-trained models vs. word embeddings that you pre-trained yourself;
- Frozen or fine-tuned word embeddings.

Please document your choices and experiments in the report.

4 Part 3: Recurrent neural networks

The goal in this section is to implement recurrent neural networks (RNNs) in PyTorch, and apply them to the task of text classification. RNNs take into account the order of elements in a sequence (words in a sentence), and thus are in theory more powerful than, say, simple averaging of word embeddings.

As in the previous section, you will need to compose them into a single fixed-length sentence representation. But since RNNs have some notion of sequential order, the ways to do this are different. These include, for instance:

- Using the last state of the RNN
- Using the first state of the RNN (if your network is bidirectional)
- Using both, concatenated/added/max-pooled
- Adding/max-pooling every state
- etc

[14] `/fp/projects01/ec30/corpora`

You **should experiment with**:

- PyTorch's three different kinds of recurrent network: Simple (Elman's) RNNs, LSTMs and GRUs.
- At least three different ways of combining the sequence of representations into a fixed-size representation (examples were given above).
- Other hyperparameters such as: number of layers, size of representations, whether your model is bidirectional or not, the amount of dropout between layers, frozen or fine-tuned word embeddings, etc. You should construct a grid, altering three such hyperparameters.

Motivate your choices, and describe the effect they had on your results. Report and discuss your results.

Try to also address the following questions:

- How do the number of parameters (weights) in these models compare to the number of parameters in the feed-forward neural classifier?
- How about training time and performance?

Report what you see.

NB: you will need to familiarise yourself with the PyTorch's `pack_padded_sequence` and `pad_packed_sequence` methods before you actually begin running things.

5 Evaluation

You should report the following performance metrics on the development (validation) set for **each** of your models:

1. accuracy;
2. precision, recall and macro-F1 score.

You can calculate these scores using either *PyTorch*, *HuggingFace Evaluate*, *scikit-learn*, or any other mainstream machine learning library.

Recall that machine learning models sometimes can produce different results with the same hyperparameters because of different random initializations. For most of the experiments you can provide one value for each metric, but for your best set of configurations please train the system 3 times and evaluate it 3 times, providing the average and the standard deviation.

You should choose your **best performing model** and publish it in your UiO Github repository as a saved *PyTorch* model. If the model is too large, put it anywhere on Fox and specify the location in your report (do not forget to adjust

the file permissions so that the members of the 'ec30' group could access the model). This model needs to be evaluated on the held-out test set, hence you need to provide a `eval_on_test.py` script which takes as input the path to a saved model and the dataset and logs the obtained results. It should hence be run with:

```
eval_on_test.py --test=[test set] --model=[saved model]
```

If the eval script fails to execute correctly, **we will deduct 1 point from your submission**.

6 Submission details

You are to submit a written report (2-4 pages) to Devilry, which provides details on your experiments and addresses the questions posed in the assignment. Your code should be available in your UiO GitHub repository (private but accessible by IN5550 staff). Irrespective of what framework you decide to use, the programming language must be Python. Your code must be self-sufficient, meaning that it should be possible to run it directly on the data. If you use some additional data sources, these datasets should be included in your repository as well. If you use non-standard Python modules (not included in our NLPL Virtual Laboratory), this should be documented in your report, but in general you should try to stick to the standard NLPL modules[15].

Good luck![1]

References

[1] Farhangi, Ashkan, Ning Sui, Nan Hua, Haiyan Bai, Arthur Huang, and Zhishan Guo. "Protoformer: Embedding Prototypes for Transformers." In PAKDD 2022, Proceedings, Part I, pp. 447-458. 2022.

[15] `http://wiki.nlpl.eu/Eosc/easybuild/modules`