ANLP Assignment 2 (Autumn 2021): Building a Part-of-Speech Tagger from Word Vectors

Seraphina Goldfarb-Tarrant, Marcio Fonseca and Shay Cohen (U of Edinburgh)

Overview, goals and motivation

In class, we learned about word embeddings. Static word embeddings, on which we focused, map each word in a vocabulary to a vector of a fixed dimension. As discussed, there are clear limitations to this approach, where *one* vector representation is used for all identical strings, regardless of context, such that the word *bat* will share a representation in all the following sentences:

(a) `She swung her cricket bat`,

(b) `A bat flew through the house`,

(c) `I bat away midges`.

Recent work in NLP has overcome many of these limitations, and used large language models to create embeddings where the context of a word is taken into account, often resulting in significant performance gains. Word embeddings that take into account surrounding context are dubbed as *contextualised word embeddings*.

If you take NLU+ in the next semester, you will learn quite a bit about different methods to derive such embeddings. These embeddings can be thought of as a function $f(w, c) \mapsto \mathbb{R}^d$ – i.e. they map a word w and its context c (for example, the sentence in which it occurs) into a vector.

In this assignment, we will explore some of the differences in static vs. contextual embeddings via analysing the information present in each of them. To determine the information present in the different representations, we will use a technique called *probing*. Probing works via the intuition that if we can build a classifier (aka a *diagnostic classifier* or a *probe*) that, given an embedding representation, can detect some linguistic information, such as part-of-speech, with high accuracy, then the representation encodes that information.

We will build a tagger to predict a part-of-speech tags from a word embedding, whether static or contextualised. It will take an English sentence as input, and output the corresponding tag for each word in the sentence.

Think about what you might expect to find, and come up with your on hypotheses. Contextual embeddings *should* overcome many of the the limitations of static embeddings which will cause errors (as in the above bat example), but they will also contain lots of other information that is not related to word sense disambiguation that will be noise for a part-of-speech classifier. Which do you think will be better, and how would you test this?

Note that this assignment assumes familiarity with the Week 8 Lab!

In addition, you should use the Python environment of Spacy you created for the lab to finish the assignment. (Or you can create a new environment, of course, with all the required packages.)

As is the tradition in the course, the second assignment is more open-ended than the first one, and requires writing a report in the form of a mini-research paper. We suggest that you **start early** working on the assignment and thinking about the arguments you would want to make.

Submitting your assignment

Your assignment needs to be submitted by November 30, 2021 at 16:00. See instructions on Learn under `Assessment` → `Assignment Submission (Gradescope)`. If you followed the video, then please note that the second assignment is traditionally more open-ended than the first one, and therefore you do not need to assign pages to questions.

Good scholarly practice

As with other assignments in this course, this assignment is intended to be done in pairs. You may freely discuss and share work within your pair, but may not share or make available solutions or code outside your group. As usual, please use care when posting questions to ensure you don't give away parts of the solution. In addition, you are responsible for following the University academic misconduct policies regarding all assessed work for credit. Details and advice about this can be found at `http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`.

Preliminaries

Download and unzip the `asgn2.zip` file into your assignment directory and follow the rest of the instructions. The zip file can be found `https://www.inf.ed.ac.uk/teaching/courses/anlp/hw/2021/asgn2/asgn2.zip`.

We highly recommend that you use the same Python environment as you used for the lab, as both the lab and this assignment have the same package requirements.

If you are having an issue with downloading the Spacy models from the github server (you get timeout from the server when running the code), try running the commands (after activating the environment):

> pip install `https://github.com/explosion/spacy-models/releases/download/en_core_web_trf-3.0.0/en_core_web_trf-3.0.0.tar.gz`
> pip install `https://github.com/explosion/spacy-models/releases/download/en_core_web_lg-3.0.0/en_core_web_lg-3.0.0.tar.gz`

Preliminary task

Before implementing anything else, it is important to have a good baseline classifier to compare your final probe against. Think back to Week 8 Lab – it is important to know what the metrics would be of a simple, not very complex solution, so that you know how your probe fairs against such a solution.

Complete the functions that are needed to build a baseline model in the form of a tagger that determines that the tag of a word is the most common tag it appeared with in the training data, and if it did not appear in the training data (meaning, if it is an "unknown" token), the most common tag overall is chosen.

The functions you need to complete are all in the file `preliminary.py`, and there are clear markers in the form of `YOUR CODE HERE` to complete the necessary code. After completing this task, run the command on the file in which you coded the solution (`preliminary.py` in the case below):

```
python preliminary.py --train_path data/en_ewt-ud-train.json \\
                      --validation_path data/en_ewt-ud-dev.json
```

and include the output table as part of your report (in the extra pages used for figures and tables. See instructions below). The table focuses on the most common part-of-speech tags.

Note about Data: The data used is taken from the Universal Dependecies treebank (see below) and is formatted in JSON format. The JSON format is a standard text-based format to store structured data. You can view the data files in any browser, or alternatively here: `https://jqplay.org/`. Don't forget to put "." in the filters to view the formatted JSON.

Main task

Your main task is to build a part-of-speech tagger as diagnostic classifier, based on the kind of word vectors you learned about in class. This broadly will map a word vector (static), or a representation (in the case of contextualised vectors), to the predicted POS tag.

The task is open-ended, and we would like you to explore at least two ways of building a POS tagger based on word vectors. The default choice would be to build a POS tagger based on static word embeddings and contextualised word embeddings and compare them.

Your POS tagger should be able to accept an English sentence and output the corresponding tag sequence for that sentence. The tagger, for example, can use a linear multiclass classifier with the embedding as input, and can output the corresponding tag.

The tags we will be using are taken from the Universal Dependencies project.[1] We learned about this project in class in the context of dependency parsing, but we will not be using the trees, but just the POS labelling.

After implementing your chosen taggers, you will need to run your methods on the test set provided and analyse your results. You would need to write a brief summary of your analysis. Note that you will not be judged on how *accurate* your POS taggers are (or if they fit within state-of-the-art results), but rather, on the report and analysis of the results in your report.

The three main data files accompanying this assignment are all in the `data` directory: `en_ewt-ud-train.json`, `en_ewt-ud-dev.json` `en_ewt-ud-test.json`.

Important Note: For your evaluation results to be valid, you would want to strictly follow the common practice of training your model on the training set, testing it during development on the development set, and finally reporting the final results on the test set. For more information about this methodological perspective, see `https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets`.

Choosing word embeddings

In the code and data provided, we use Glove as static word embeddings and BERT for the contextualised ones. However, in the recent decade there has been a myriad of other word vector algorithms. If you would like to use a different one you find online, you can add it to your report and analysis. Choosing different word embeddings than Glove or BERT is left as an open-ended non-mandatory option. If you do this, note that tokenization schemes differ for different models, so be sure the whatever you use works with the model you have chosen.

As mentioned briefly in the Week 8 Lab, here are a number of common ways to get one representation from multiple tokens. In the lab you used averaging, but here your choice of combination function will be an experimental variable - these functions are included in *models.py* and you need to choose which one(s) to use in your final experiments and analysis.
The full set of functions implemented is below.

1. **First token**: This is a naive option, but many works do use it. The first token of an entity is taken to represent the full entity, and the rest is ignored. (`FIRST_TOKEN` below.)
2. **Last token**: All tokens save the last one are ignored. (`LAST_TOKEN` below.)
3. **Maxpool**: Elementwise maximum of all vectors. (`REDUCE_MAX` below.)
4. **Average**: Elementwise average of all vectors. (`REDUCE_MEAN` below.)
5. **Sum**: Elementwise sum of all vectors. (`REDUCE_SUM` below.)

[1]`https://universaldependencies.org/`

With the provided files, you can find a version of the embedding class from the Week 8 Lab that allows you to save contextualised embeddings into a file. The command to get the embeddings is:

```
python embedding.py --contextual --combination COMBFUNC
```

where `COMBFUNC` can be one from the list `NONE, REDUCE_MEAN, REDUCE_MAX, REDUCE_SUM, FIRST_TOKEN, LAST_TOKEN`. Please note that running this command takes about 30 minutes on a standard CPU. `NONE` will return the embeddings for all tokens without pooling.

The static embeddings are obtained using the same command without arguments:

```
python embedding.py
```

Running the tagger

While you are more than welcome to implement your own tagger with whatever classification algorithm you choose, we provide as part of the assignment a file called `tagger.py` that has skeleton code for performing training and testing for a tagger based on the embeddings calculated in `embedding.py`.

To train the tagger, use the following command:

```
python tagger.py train --embeddings_path data/en_ewt-ud-embeds.pkl \\
  --model_path models/tagger.pkl
```

or another example for training a different model is:

```
python tagger.py train --embeddings_path \\
  data/en_ewt-ud-embeds-reduce_mean.pkl --model_path \\
  models/tagger-contextual.pkl
```

where the `.pkl` files are generated with the `embedding.py` command as mentioned earlier.

To test the classifier, you can use, for example:

```
python tagger.py test --embeddings_path \\
  data/en_ewt-ud-embeds-reduce_mean.pkl --model_path \\
  models/tagger-contextual.pkl
```

Inspect the code in `tagger.py` to see more options on how to run the code and how it is written.

What to look for

Below are some suggestions for questions you might want to consider if you follow the default task as above:

- How does prediction of parts of speech from static word embeddings compares to the most common tag baseline? Does it have higher accuracy of unknown words?
- Does the use of different combination functions affect the results? Or are contextualised embeddings quite stable with respect to the choice of a combination function?
- Are there tags that are identified more often correctly than other tags? Are they more frequent or less frequent? What metric would you use to analyse a specific POS tag?
- What are some mistakes that either the contextualised or static embedding tagger makes? What is the cause of these mistakes? Are there mistakes that one tagger makes and the other does not?
- Do you expect the pattern of mistakes to differ in different languages? Why or why not?

- In which cases would we need to evaluate more than just *overall* accuracy?

Or think of your own question! You do not need to answer all of these questions in your report. Rather, you should choose one or two questions to focus on, where you feel you have something interesting to say. Identify which question(s) you are addressing, how you addressed them, and what results you got

How to approach the question

Ideally, you should use a combination of *quantitative* and *qualitative* analysis. In NLP, the most common form of quantitative analysis is comparing the accuracy of a result against some gold standard. However, we also recommend to accompany such results by a qualitative analysis.

There are many more options for qualitative analysis. You could decide to visualize some results, discuss the behaviour of some particular examples, or analyse the tagging of a sentence or two in more detail. You may also choose to explain some new conclusion that you learned about in the process of following this assignment, or some hypothesis that you confirmed.

As you work on the assignment, you may indeed do several of these, but again, please do *not* try to describe everything you did in your report. Focus on one or two analyses that seem the most interesting or illuminating, and give a clear and concise explanation of those, with evidence to support any claims you make.

What to write up

This assignment is more open-ended than the previous one, but we are strictly limiting the length of your report (see below). So you must limit the scope of your investigation—we are not looking for you to try every possible thing you can think of or provide completely definitive answers to all questions. We are basically looking for you to implement some fairly simple methods, spend a bit of time thinking about what is in the dataset, and see what you can discover about the data or the methods.

As mentioned above, your report should focus on just one or two questions that you investigated. You are more likely to do well by providing a thoughtful analysis of a small number of methods than by attempting to compare a very large number of methods (because you probably won't have time or space to do a good job of that). Your report should describe:

- what question(s) you investigated.
- what methods you compared. You do not need to describe all the technicalities, but you *should* include a brief description of whatever other methods you chose.
- how you analysed the results and what you concluded.

Typically, a good report will very briefly summarize at least the first three of these points in the introduction, then expand upon them as needed in the remainder, along with reporting results and conclusion.

In addition, your report should include:

- the output of the program after you have completed the Preliminary Task. This can be included as the first part of your "additional two pages" (see below). You do not need to comment on it or relate it to the rest of your report, it's just for us to check the numbers.

Note well: Your report is limited to **two pages of text** (no smaller than 11 point font; if in doubt about format, try not to exceed 1,000 words in your report) plus **two pages of additional material** which can include figures, tables, example sentences and tagging, etc. We may reduce marks for reports that go over these limits. Your bibliography (reference list) may extend beyond the 4-page limit as we do not wish to limit your use of references. The two extra pages should not be embedded in the two pages of content—they should be on separate pages.

These limits are very short! Writing a short document well is often harder than writing a long one. Don't wait until the last minute to start writing. You will need time to decide what is the most important information to include and how to edit your text to the right length. All figures must be properly labelled and large enough to be readable without magnifying.

Think of this exercise as writing a (very) short research paper about POS tagging. The content would be open-ended, and you would need to structure it with some preamble, the method description and the results.

Marking information

Marking will be based on your report. A high-quality report will have the following properties:

- Clear statement of the question/hypothesis
- Methods are appropriate and clearly explained/justified as needed (including the overall design of experiments/analyses)
- Reported results suggest that methods are correctly implemented. (To help us check this, you *must* include the output of the preliminary task in your report. See above under 'What to write up'.)
- Figures/tables are clear and appropriate
- Good discussion/interpretation of results
- Appropriate use of references/citations
- Concise; does not exceed required length. (See above under 'What to write up')