NLP Monsoon 2019 Course Project

Zubair Abid Akshay Goindani

NLP Monsoon 2019 Course Project

Team: Na Bien Na Mal

Akshay Goindani, Zubair Abid

TODO

- ☒ Visualisation:
  - ☒ Similarity Distribution of Words
- ☐ Evaluation:
  - ☐ Mapping Parallel Corpora to the embedding
  - ☐ Task based:
    - ☐ Versus Facebook Iterative Procrustes

Comments on the existing code

- ~~`code.py`: sorted_MX should be sorting $\sqrt{M_x}$ instead of just M_x I guess.~~
-

Explaining the program

Our end goal is to create a Multilingual Word Embedding, for English and Hindi, We are trying to do it in an Unsupervised manner, albeit we may shift to slightly supervised methods if things do not work out.

Setting up a baseline

dbcjsdb

Attempting Solution 1: Unsupervised Initialisation of A Seed Dictionary by exploiting Isometry of Individual Word Embeddings

The first solution we approached was to implement the Paper given to us as an example. It creates Multilingual Word Embeddings for English-Italian without any supervision, exploiting the idea of isometry of word embeddings.

We tackled the problem, largely, in three major steps:

1. We created word embeddings for both language pairs individually. In our case, we did this using Gensim's implementation of FastText.

   Pre-trained word embeddings are a possible solution, but we trained our own. The paper specifically rules out embeddings trained on Wikipedia data, as the two corpora are then parallel and contain a lot more structural similarity than is generally observed, leading to potentially poor general solutions.

2. We tried to initialise a *seed dictionary* in an unsupervised manner. In the paper, this is done to have a nonrandom initialisation for the self-learning step just after, but this is where the bulk of the problem is to be solved.

   The way this is done is: conceptually, the embeddings are both similar: isometric embeddings, so in theory they can be aligned on top of one another. What we try to do in this solution is to map both to a common vector space with linear operators, where our job is to now figure out this linear operator.

3. With the initialised dictionary, we then implemented the self-learning step. It is pretty simple; for a given dictionary set of values the algorithm tries to optimize the transformation matrices, and then given the matrices the algorithm attempts to update the dictionary.

   This in itself did not give very good results, as in the paper. So we added some optimizations to the algorithm:

   1. Frequency-based vocabulary cutoff
   2. CSLS Retrieval
   3. Bidirectional dictionary induction

   However, even this did not improve our solution by much, with the majority of the word-pairs generated from this turning out to be very wrong.

Now, we wanted to find out whether or not this was a problem unique to our particular language-pair or not; the paper having used 2 closely related languages might have gotten away with a suboptimal initialisation solution for other language pairs.

One way to check this is to download word embeddings for two other languages - say English and Italian - and running these through the program.

Building an evaluation suite

Since eye-estimation is a poor measure of model accuracy we need some sort of evaluation metric to accurately pinpoint the effectiveness/lack of such for our system.

We employ two evaluation metrics: one intrinsic and one extrinsic. We also use multiple embeddings to compare results against: the baseline, the unsupervised solution trained, and iterative Procrustes (Conneau et al., 2018).

Read more about the evaluation methods in the Appendix

Model Name	Metric 1	Metric 2	Metric 2.1
Baseline	oof	oouch	nope
Artetxe	oh no	kk	bleh
MUSE			

Exploring other solutions

Appendix

Architecture

```
graph LR;
    DX[Data in Hindi];
    DY[Data in English];
    EX1[Hindi Embedding]
    EX2[English Embedding]

    FM1((FastText))
    FM2((FastText))


    DX-->FM1
    FM1-->EX1

    DY-->FM2
    FM2-->EX2

    subgraph Normalization;
        EX1-->|length normalization|EX1
        EX1-->|mean-centering|EX1

        EX2-->|length normalization|EX2
        EX2-->|mean-centering|EX2
    end
```

Visualising Embeddings

Using the `sorted_MX` and `sorted_MY`. We take the "smoothed density estimates" of them.

For the ith word in the vocabulary, `sorted_M?[i]` represents the word similarity distribution we want to use.

```
for hi, en in word_pairs:
    plot(sorted_MX[hi])
    plot(sorted_MY[en])
```

Embedding Evaluation

Intrinsic Evaluation: Word Translation Given a set of word pairs that we know translate equivalently in the two languages, we check their similarity to one another. Since both embeddings have been mapped into a common vector space, we can apply the transformation and then check for cosine similarity.

Assuming we already have the calculations done, where the primary bits are

```
embed_X = np.dot(X, WX)
embed_Y = np.dot(Y, WY)
```

Assumption I'm making here: word `i` represented in original embedding space as `X[i]` can be reached in the shared embedding space as `embed_X[i]`

The Task is to measure how close the embeddings for translated words are in the shared common embedding space.

So we find the cosine-similarity of the embeddings (into the same space) of equivalent word-pairs. We take the cosine-similarity for each pair, and then average.

```
cos_sims = []
for hi, en in word_pairs:
    cos_sims.append(cosine_similarity(embed_X[hi], embed_Y[en]))

cos_sims.average()
```

```
gantt
    dateFormat DD-MM-YYY
    axisFormat %m/%y

    title Example
    section example section
    activity :active, 01-02-2019, 03-08-2019
```

- a
- list
- of
- random
- things
- to
- fill

- up
- some
- space
- so
- i
- i
- can
- so
- this
- stuff
- nicer
- can
- so
- this
- stuff
- nicer