Catalan Word Vectors. Training and analysis

Co-authors: Jordi Vaccher[1], Daniel Navarrete[1]

[1]Master in Advanced Telecommunication Technologies
ETSETB - Universitat Politècnica de Catalunya
jordi.vaccher@estudiantat.upc.edu, daniel.navarrete.jimenez@estudiantat.upc.edu

1. Introduction

This report shows the study of transforming word vectors or word embeddings. So we create, analyze, and evaluate these word vectors. They are numerical representations about contextualized words into a learned vocabulary. Specifically we are trying to get the Catalan Word Vectors by training the following model.
To do that task we are using a database created from Wikipedia articles.

2. Continuous Bag-Of-Words model (CBOW)

The standard CBOW model sums all the context word vectors with the same weight. So we are trying to implement and evaluate a weighted sum of the context words with the following ways, divided in four tasks.
The baseline we are improving is the following one:

Table 1: *Train and validation accuracy results*

Accuracy
Train acc = 0.292
Dev acc Wikipedia = 0.270
Dev acc El Periodico = 0.177

2.1. Exercise 1a: A fixed scalar weight

The first exercise consisted in give more contextual weight to the nearest words than the farthest ones of the window. So we are multiplying by 2 the closest and by 1 the farthest.
We have achieved the following results:

Table 2: *Train and validation accuracy results*

Accuracy
Train acc = 0.291
Dev acc Wikipedia = 0.271
Dev acc El Periodico = 0.178

2.2. Exercise 1b: A trained scalar weight for each position

In this exercise it is proposed to train the initial scalar weight values. Then the algorithm will learn the values of every weight position. We have initialized the weights with the values from the exercise 1a. We have achieved the following results:

Table 3: *Train and validation accuracy results*

Accuracy
Train acc = 0.346
Dev acc Wikipedia = 0.324
Dev acc El Periodico = 0.224

2.3. Exercise 1c: A trained vector weight for each position

In this exercise it is proposed to train the initial weight values but instead of training a scalar weight we are using vectors to achieve an appropriate weight representation of every embedding position. In this exercise we have initialize the weights with random values, so it is better for the system giving no information in order to avoid any bias.
We have achieved the following results:

Table 4: *Train and validation accuracy results*

Accuracy
Train acc = 0.423
Dev acc Wikipedia = 0.402
Dev acc El Periodico = 0.305

In order to check the result using a tensor of ones as initial weights, we have trained another model, obtaining the following results:

Table 5: *Train and validation accuracy results*

Accuracy
Train acc = 0.424
Dev acc Wikipedia = 0.404
Dev acc El Periodico = 0.310

2.4. Exercise 1d: Hyperparameter optimization

Finally we are tunning the hyperparameters in order to optimize the learning performance.

- Firstly we have doubled the batch size in order to increase the learning speed and wondering it could improve the accuracy results.
- Noticing the results do not improve so much, we multiply by 2 the embedding size. The results improve as we can see in the following table:

Table 6: *Train and validation accuracy results*

Accuracy
Train acc = 0.441
Dev acc Wikipedia = 0.418
Dev acc El Periodico = 0.317

3. Word Vector analysis

In the third section we are evaluating the performance of our improved system. It is divided into two exercises:

3.1. Exercise 2a: Implement the WordVector class

We implemented the WordVector class so we implement the most similar and analogy methods to find closest vectors and analogies using the cosine similarity measure.
The obtained results show that the first prediction match with the expected one. The other ones varies depending the similarity obtained but the most similar words and analogies are similar though in a different order.

3.2. Exercise 2b: Intrinsic evaluation

In this section we tested our functions with several examples.

Table 7: *Analogies method results*

input	output
ahir - demà - abans	= després
Madrid - Espanya - Milà	= Itàlia
noi - noia - doctor	= infermera
arbre - tronc - flor	= tija

In the Table 7 we can see how the method works properly extracting analogy word. But in the last case we can see how its performance is worse. The system is probably biased by a patriarchal vocabulary.

Table 8: *Most similar method results*

input	output
dimecres	= dijous
pop	= folk
banc	= vehicle

In the Table 8 is showed that it can generate the proper in all cases. But in a more subjective view, in catalan language 'pop' means octopus so it could be biased by an english influence.

3.3. Exercise 2c: Visualize word analogies or word clustering properties

In order of seeing the clusters properties we use Principal Component Analysis (PCA). Using the 3 most powerful eigen-vectors we extract the following representation in which we plot 15 words.

You can observe the 3D PCA in video in the following link: https://drive.google.com/file/d/1RrgW3El21XgJMOUMaPxYJ2dKw3Kn33-L/view

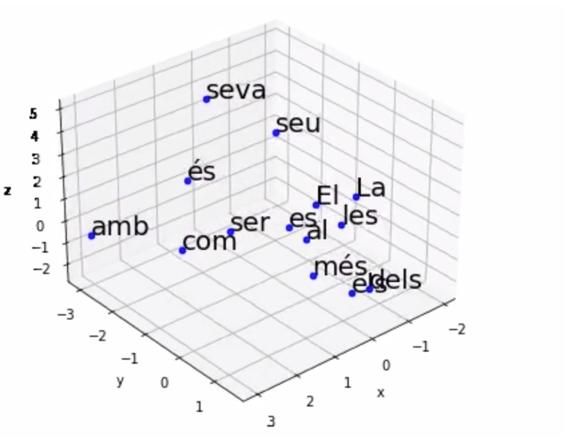


Figure 1: *PCA plot.*

3.4. Exercise 2d: Prediction accuracy

We have compared the accuracy of the implemented CBOW models in the out-of-domain (el Periódico) test set (prediction of the central word given a context of 2 previous and 2 next words).
Our best result (BertNErnie's Team) is showed in the following table:

Table 9: *Kaggle Competition: BertNErnie Team*

Team	Accuracy
20.MarioNLuigi	= 0.30900
21.BertNErnie	= 0.30666
22.Armand N Samuel	= 0.30400

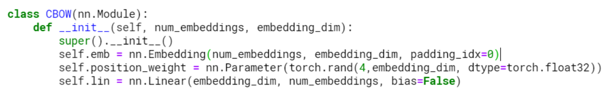
We have achieved a good result . Maybe we could tune the hyperparametres a bit more in order to achieve better results.

4. Conclusions

- In conclusion, we have seen that not only training a weight vector we can improve the system, but also increasing the embedding dimension we have achieved a better accuracy.
- We have reached a very similar accuracy initializing the vector as a tensor of ones and a random vector.
- We could try to increase the embedding dimension to check the optimum value.
- The used vocabulary could be biased.We found few examples of bad performance. Per example when it deals the conversion between male / female jobs.
- The system works properly so we can observe in the PCA representation where are located similar semantic words.Per example the prepositions have a close representation
- We have achieved a good Prediction accuracy in the competition but we are not in the top competitors.

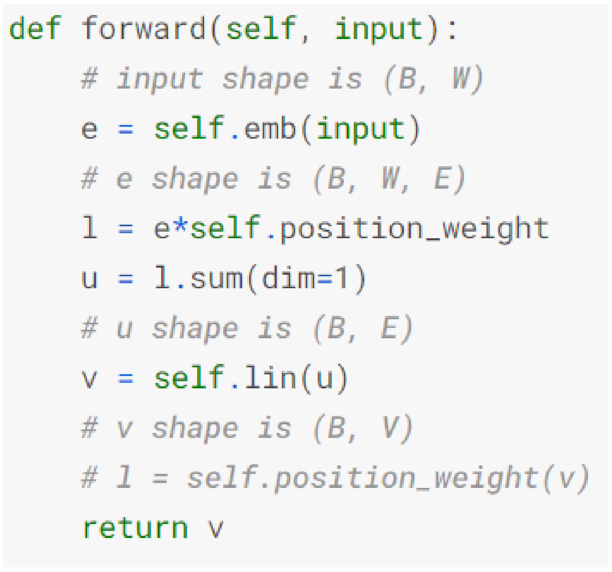
5. ANNEX

In this section we show the code modifications we have done in each exercise.



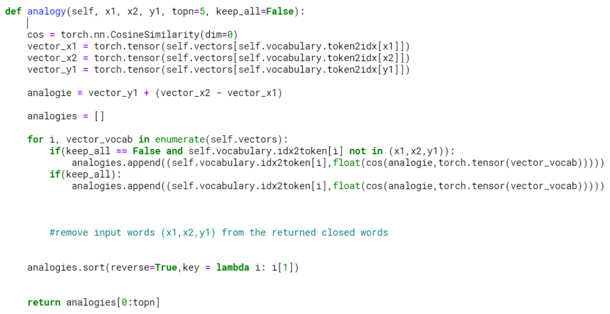
```python
class CBOW(nn.Module):
    def __init__(self, num_embeddings, embedding_dim):
        super().__init__()
        self.emb = nn.Embedding(num_embeddings, embedding_dim, padding_idx=0)
        self.position_weight = nn.Parameter(torch.rand(4,embedding_dim, dtype=torch.float32))
        self.lin = nn.Linear(embedding_dim, num_embeddings, bias=False)
```

Figure 2: *init Code modifications exercise 1c.*



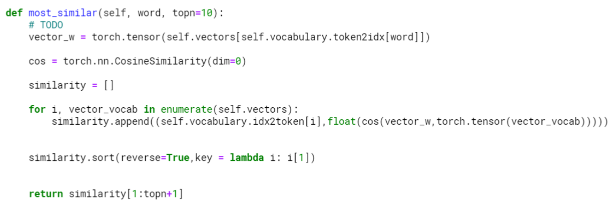
```python
def forward(self, input):
    # input shape is (B, W)
    e = self.emb(input)
    # e shape is (B, W, E)
    l = e*self.position_weight
    u = l.sum(dim=1)
    # u shape is (B, E)
    v = self.lin(u)
    # v shape is (B, V)
    # l = self.position_weight(v)
    return v
```

Figure 3: *forward Code modifications exercise 1c.*



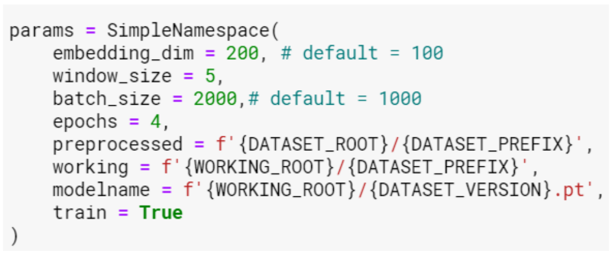
```python
def analogy(self, x1, x2, y1, topn=5, keep_all=False):

    cos = torch.nn.CosineSimilarity(dim=0)
    vector_x1 = torch.tensor(self.vectors[self.vocabulary.token2idx[x1]])
    vector_x2 = torch.tensor(self.vectors[self.vocabulary.token2idx[x2]])
    vector_y1 = torch.tensor(self.vectors[self.vocabulary.token2idx[y1]])

    analogie = vector_y1 + (vector_x2 - vector_x1)

    analogies = []

    for i, vector_vocab in enumerate(self.vectors):
        if(keep_all == False and self.vocabulary.idx2token[i] not in (x1,x2,y1)):
            analogies.append((self.vocabulary.idx2token[i],float(cos(analogie,torch.tensor(vector_vocab)))))
        if(keep_all):
            analogies.append((self.vocabulary.idx2token[i],float(cos(analogie,torch.tensor(vector_vocab)))))



        #remove input words (x1,x2,y1) from the returned closed words


    analogies.sort(reverse=True,key = lambda i: i[1])


    return analogies[0:topn]
```

Figure 4: *Analogies Code modifications exercise 2a.*



```python
def most_similar(self, word, topn=10):
    # TODO
    vector_w = torch.tensor(self.vectors[self.vocabulary.token2idx[word]])

    cos = torch.nn.CosineSimilarity(dim=0)

    similarity = []

    for i, vector_vocab in enumerate(self.vectors):
        similarity.append((self.vocabulary.idx2token[i],float(cos(vector_w,torch.tensor(vector_vocab)))))


    similarity.sort(reverse=True,key = lambda i: i[1])


    return similarity[1:topn+1]
```

Figure 5: *most similar Code modifications exercise 2a.*



```python
params = SimpleNamespace(
    embedding_dim = 200, # default = 100
    window_size = 5,
    batch_size = 2000,# default = 1000
    epochs = 4,
    preprocessed = f'{DATASET_ROOT}/{DATASET_PREFIX}',
    working = f'{WORKING_ROOT}/{DATASET_PREFIX}',
    modelname = f'{WORKING_ROOT}/{DATASET_VERSION}.pt',
    train = True
)
```

Figure 6: *Hyperparam. code.*