Assignment 1

Catalan Word Vectors. Training and analysis

Course: Spoken and Written Language Processing

(POE-GCED)

Jordi Aguilar

Miquel Escobar

March 10, 2020

Contents

1 Introduction

In this homework assignment we have created, analyzed and evaluated word vectors using a database created from the Catalan Wikipedia. To do so, we have modified the given baseline **Kaggle** notebooks with the corresponding contributions. These contributions are specified in section **2. Modified Code**.

We have analyzed the trained models, observing their results and embeddings through samples of pair of words similarity, analogies and clustering techniques, detailed in further sections **3.1**, **3.2** and **3.3** correspondingly.

All the results and visualizations mentioned and included in this report have been generated by the Python scripts contained in the `notebooks` and `analysis` folders. In order to understand their generation, please refer to the scripts if there as there might not be an specific explanation in the report for each case. The names of each of the `analysis` scripts, as well as the code in it, should be self-explanatory, but do not hesitate to contact the authors in order to clarify any doubts or make any inquiries.

2 Modified code

2.1 `CBOW Training` notebook

Given the baseline CBOW Training notebook, we have defined three classes, corresponding to each of the implementations specified in the assignment statement. Note that the different implementations differ essentially on the `position_weight` parameter definition.

a *A fixed scalar weight, e.g, (1,2,2,1) to give more weight to the words that are closer to the predicted central word.* The `position_weight` parameter is defined using the `register_buffer` method as this makes it constant and unalterable by the training.

```
class CBOWa(nn.Module):
    def __init__(self, num_embeddings, embedding_dim):
        super().__init__()
        self.emb = nn.Embedding(num_embeddings, embedding_dim,
padding_idx=0)
        self.lin = nn.Linear(embedding_dim, num_embeddings, bias=
False)
        self.register_buffer('position_weight', torch.tensor
([1,2,2,1], dtype=torch.float32))

    def forward(self, inputs):
        # B * W1
        U = (self.emb(inputs) * self.position_weight.view(1,4,1)).sum
(dim=1)
        # B * E
        V = self.lin(U)
        # B * V
        return V

```

Listing 1: Fixed word weights

b *A trained scalar weight for each position.* The `position_weight` parameter is defined using the `nn.Parameter` class initialized with a 4-element vector, each element corresponding to each position.

```
class CBOWb(nn.Module):
```

```
    def __init__(self, num_embeddings, embedding_dim):
        super().__init__()
        self.emb = nn.Embedding(num_embeddings, embedding_dim,
padding_idx=0)
        self.lin = nn.Linear(embedding_dim, num_embeddings, bias=
False)
        self.position_weight = nn.Parameter(torch.tensor([1,2,2,1],
dtype=torch.float32))

    def forward(self, inputs):
        # B * W1
        U = (self.emb(inputs) * self.position_weight.view(1,4,1)).sum
(dim=1)
        # B * E
        V = self.lin(U)
        # B * V
        return V

```

Listing 2: Trained word weights

c *A trained vector weight for each position. Each word vector is element-wise multiplied by the corresponding position-dependent weight, and then added with the rest of the weighted word vectors.* The `position_weight` parameter is defined using the `nn.Parameter` class initialized with a 4x100 vector to fit the requirements.

```
class CBOWc(nn.Module):
    def __init__(self, num_embeddings, embedding_dim):
        super().__init__()
        self.emb = nn.Embedding(num_embeddings, embedding_dim,
padding_idx=0)
        self.lin = nn.Linear(embedding_dim, num_embeddings, bias=
False)
        self.position_weight = nn.Parameter(torch.ones(4, 100))

    def forward(self, inputs):
        # B * W1
        U = (self.emb(inputs) * self.position_weight.view(1,4,-1)).
```

```
    sum(dim=1)
11          # B * E
12          V = self.lin(U)
13          # B * V
14          return V
15
```

Listing 3: Trained word vector weights

2.2 `Word Vector Analysis` notebook

In this notebook, we had to implement the `WordVectors` class from scratch which, instantiated with a `vectors` and a `vocabulary` parameter, provides the methods `most_similar` to find the most similar words of a given word (or given embedding, due to the flexibility of our implementation), and `analogy` to find the analogy of a word given their sibling analogy.

a `most_similar` method. A heap of maximum size `topn` is updated each time the least similar item in it is less similar than the one of the current iteration, based on the cosine similarity. This heap is what is returned once the similarity to all the `vocabulary` words is computed and compared.

```
1     def most_similar(self, word, topn=10):
2         tokens = list()
3         similarities = list()
4         # If we are given a word, we obtain its embedding
5         if type(word) == str:
6             word_embedding = self.get_word_embedding(word)
7         # If instead we are directly given the embedding no
    transformation is required
8         else:
9             word_embedding = word
10
11        for i, token in enumerate(self.vocabulary.token2idx):
12
13            token_embedding = self.get_word_embedding(token)
14            similarity = ( np.dot(word_embedding, token_embedding) /
15                           (np.linalg.norm(word_embedding)*np.linalg.norm
    (token_embedding)) )
```

```
16
17              if i < topn:
18                  tokens.append(token)
19                  similarities.append(similarity)
20
21              elif similarity > min(similarities):
22                  replace_idx = similarities.index(min(similarities))
23                  tokens[replace_idx] = token
24                  similarities[replace_idx] = similarity
25
26          return sorted(list(zip(tokens, similarities)), key=lambda x: -x
    [1])
27
```

Listing 4: `WordVectors`'s `most_similar` method.

b `analogy` method. The embedding corresponding to adding the difference of the sibling analogy (which represents their "relationship") to the given word is passed as a parameter to the `most_similar` method[1].

```
1       def analogy(self, x1, x2, y1, topn=5, keep_all=False):
2           x1_emb = self.get_word_embedding(x1)
3           x2_emb = self.get_word_embedding(x2)
4           y1_emb = self.get_word_embedding(y1)
5           analogy_emb = y1_emb + (x2_emb - x1_emb)
6
7           analogies = self.most_similar(analogy_emb, topn+3)
8           if not keep_all:
9               analogies = [(k, v) for k,v in analogies if k not in (x1, x2,
     y1)]
10          return analogies[:topn]
11
```

Listing 5: `WordVectors`'s `analogy` method.

[1]Please refer to the `Word Vectors Analysis.ipynb` file for the `get_word_embedding` function source code, used in both `most_similar` and `analogy` methods.

3 Analysis

In order to analyze whether or not our model is a great fit to catalan language, we are going to observe some word similarities, analogies and clusters. These are going to be measured by means of the vector representation of each word, resulting from both the embedding and linear layers of the model **c** (we use this model instead of model **a** or **b** as it is the one with higher precision).

3.1 Close words

An interesting way to measure the quality of our trained neural network and thus, of its predictions and coefficients (pertaining to both the embedding and linear layer), is to observe the similarities between words based on the latter, applying the cosine similarity measure. What are the most similar words embeddings to the given word embedding? Let's dive into some examples.

3.1.1 A couple of good examples

After applying the `most_similar` method to some words, the four most similar words to each of them are the following:

word	embedding similar words	liner layer similar words
'català'	'valencià', 'basc', 'gallec','mallorquí'	'francès','alemany', 'anglès','espanyol'
'jugar'	'disputar','competir','lluitar','triar'	'disputar','guanyar','jugant','cantar'
'Joan'	'Josep','Jaume','Pere','Jordi'	'Pere','Maria','David','ell'

Both vectors give coherent results to each of these words, even though the embeddings seem a little bit better than the linear layer (for example, the most similar words to 'català' are languages in both cases, but the embeddings results return languages from the same country while the linear layer results are languages from other countries).

3.1.2 Not so good examples

Not all cases work well, as some words are surrounded (in terms of similarity) by others which have no linguistic reason to be that close.

word	embedding similar words	liner layer similar words
'pantalla'	'planxa', 'cabina', 'plantilla'	'càmera','maquinari', 'cartutx'
'bitllet'	'cartutx','logotip','proveïdor'	'avaluats','renovats','ascendia'
'banc'	'pavelló', 'mercat', 'vestíbul'	'parlament', 'pal', 'tribunal'

What these three words have in common is that they are homonyms (the same spelling stands for more than one meaning). This obviously hinders the task of generating a good vector representation of the word, which explains the bad similarities obtained.

3.2 Analogies

Analogies are the cognitive process of transferring information from one subject into another. In this case, the approach is that given two words that have some kind of relation (let's call one the parent and the other one the son, just for the purpose of clarity), we want to find the corresponding son of any other given parent (or the other way around, it does not really matter). For example, football (parent) is to footballer (son) what dance (parent) is to dancer (son).

3.2.1 A couple of good examples

The following examples worked pretty well both in the embeddings and linear layer words representation:

sibling analogy	word	emb. top analogies	lin. top analogies
'França'-'francès'	'Polonia'	'polonès','alemany','rus'	'rus','japonès','neerlandès'
'alt'-'baix'	'llarg'	'curt','través','revés'	'curt','creixement','pes'
'calor'-'estiu'	'fred'	'hivern','agost','octubre'	'hivern','primavera','tardor'

The first case works perfectly with the embedding vectors (even though not as well with the linear layer), with a relationship of country-language. In the second case, the relationship is of antonyms, and the result ('curt') is the antonym of the given word 'llarg'. Finally, the third case also returns what is expected ('hivern') given the word 'fred'.

3.2.2 Not so good examples

In other cases we observed a not so good analogy:

sibling analogy	word	emb. top analogies	lin.top analogies
'joc'-'jugar'	'dansa'	'representar','cantar','practicar'	'cantar', 'empatar', 'assistir'
'tecla'-'teclat'	'jugador'	'soldat','boxejador','pilot'	'teatre','partit','militar'
'macarrons'-'pasta'	'poma'	'cera','carbassa','flama'	'memòria','pa','pell'

In the first case, we would expect to obtain 'dansar' or 'ballar', which are the verbs corresponding to the noun 'dansa' (the same as 'jugar' is to 'joc'). Instead, we obtain some other verbs which in some cases are related to the action of 'dansar', but none of them is what we expected, probably due to the fact that 'dansa' is not very oftenly used and as a consequence the embeddings are not precise enough for this analogy. In the second case, we would expect to obtain the word 'equip' or something similar, but the results do not make a lot of sense. Finally, the third case the clear correct result would be 'fruita', and instead we also obtain some bad results.

3.3 Clustering (visualization by means of PCA)

Taking advantage of the word embeddings that have been generated, we will try to plot them by its grammatical category, look for some patterns and check whether they are distributed according to its grammatical category.

To do so, firstly we will reduce the dimensionality of the vectors to 2 by means of PCA, and secondly we will tag each word with all its possible grammatical categories by looking for them in an online dictionary like Word Reference. To do this, we have implemented a simple code that parses the webpage of the definition of the word and retrieves all the tags of the grammatical categories of the word. We must warn that a lot of words do not have any category, as we have imposed that the word defined must match the query. However, we have enough examples to visualize some interesting clusters.

Once we have the two-dimensional vector and the grammatical category for each of the words, we are going to visualize them and find interesting patterns.

3.3.1 Nouns and verbs

First of all we are going to focus on the two main categories of words, that is *nouns* and *verbs*:

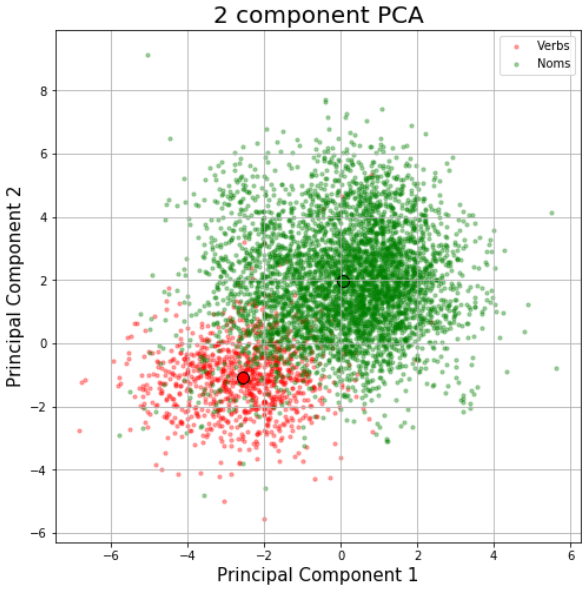

Figure 1: Word embeddings of the verbs (in red) and the nouns (in green).

As we can see in the plot, the categories *nouns* and *verbs* are clearly differentiated. This

could be because they form the two main cores of a sentence, that are the *Subject* and the *Predicate*.

3.3.2 Nouns, verbs, adjectives and adverbs

Now we are going to add the *adjectives* and the *adverbs* to the plot:

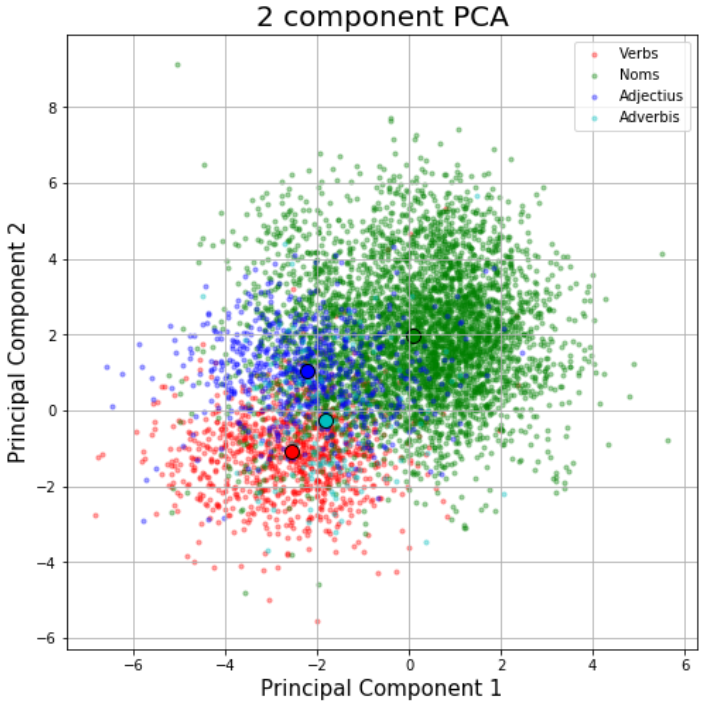

Figure 2: Word embeddings of verbs (red), nouns (green), adjectives (blue) and adverbs (cian).

It is interesting to see how the adjectives and the adverbs fall in the middle of the nouns and the verbs, but surprisingly the adverbs fall much closer from the verbs than the adjectives. We have noticed can make sense as adverbs usually complement verbs.

3.3.3 Femenine and masculine nouns

Now we are going to go deeper and examine different classifications inside the grammatical categories. We will begin by taking a look to a concept that doesn't exist in English, which is *masculine nouns* and *femenine nouns* (keep in mind that the analyzed language in this report is Catalan).

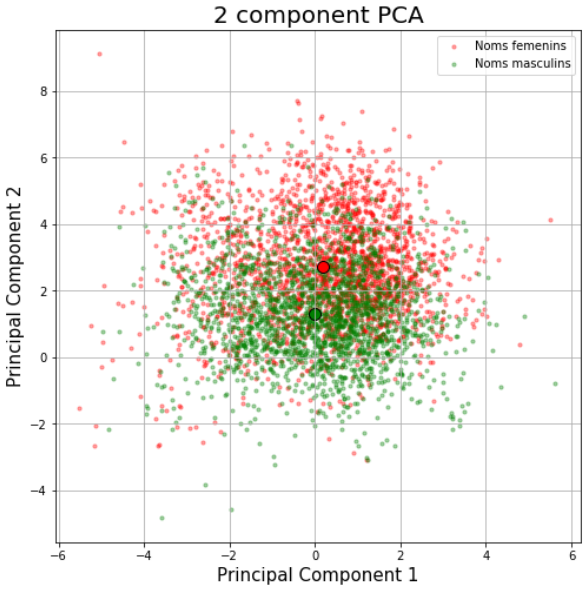

Figure 3: Word embeddings of masculine nouns (green) and femenine nouns (red).

Although the separation of the points is not as obvious as the separation between nouns and verbs, we can appreciate that the femenine nouns reside in the upper part of the plot while the masculine nouns do it in the lower part.

3.3.4 Transitive and intransitive verbs

Another interesting plot is the one between transitive and instransitive verbs.

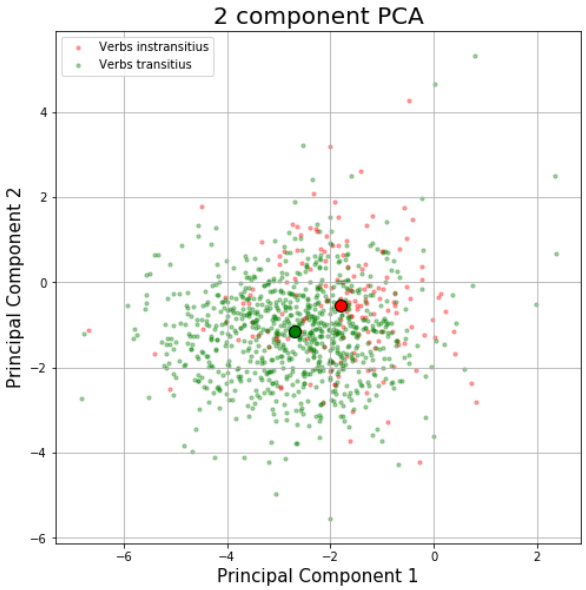

Figure 4: Word embeddings of transitive verbs (green) and intransitive verbs (red).

Recalling on what we learned at highschool, transitive verbs are those which require a direct complement right afterwards. For example "dir" (which means "say"): "Jo dic *alguna cosa*" (in English: "I say something"). In the plot they are slightly to the right.

This case is interesting as it shows us the great amount of information that is stored in the word embeddings: the transitivity of the verbs is not incredibly relevant information about them, but even so the neural network has been able to capture it sufficiently to observe the differences in a two-dimensional PCA representation.

3.3.5 Tagged vs. non tagged words

Finally, we have decided to plot all the words that we were able to classify into a grammatical category versus those who we could not, with the following result:

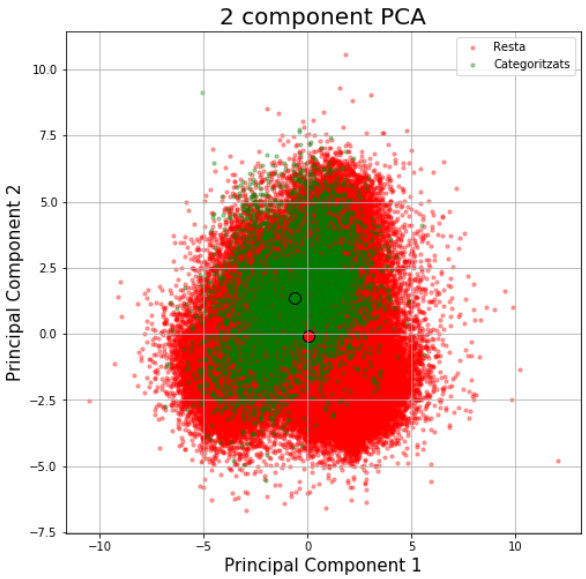

Figure 5: Word embeddings of the classified words (green) and unclassified (red).

We observe that our tagged words occupy the majority of all the words, nonetheless, there is a region at the lower right part of the plot where there are no words with any grammatical category. Carried by the curiosity of knowing what kind of words were, we tried to take this sub-region of the points (precisely the points under the line $y = x - 3$), switched the index by the token and displayed a sample. This was the result:

`'Bàrcino','Canomals','Mantinea','Servera','Austerlitz','411','Rambert','Canadell'`

It clearly seems that these are proper nouns. We have added a few lines in our code corroborate this hypothesis, and from the subset that we extracted with a size of 20.642 words, 89% start with a capital letter and 8.5% are numbers. So we can state that the proper nouns and the numbers form another cluster apart from the tagged words.

Before ending this section, we must point out that we have done this clustering with the word embeddings of the *model c* because is the model that visually differentiated the groups in a more clear manner.

4 Conclusions

To wrap up this report, we can say that we now understand more deeply the power of word embeddings. In these vectors there is stored everything the machine needs to know to understand or construct a sentence. From similarity, to grammatical categories or word analogies, the neural network has been able to learn all the required features with no initial clue, and instead simply by taking 5-word sentences and predicting the middle word.

Another interesting aspect about word embeddings that we would have liked to go deeper into is to understand what kind of information is stored in each dimension of the vector and what is its purpose, understanding if it is possible to extract word properties from each of the vector dimensions.

In general, it has been a great assignment to familiarize ourselves with word embeddings and as a useful and practical introduction to the world of language processing.