Spoken and Written Language Processing - POE

Assignment 1. Catalan Word Vectors. Training and analysis

Gerard Martín Pey

Jose Àngel Mola Audí

7th March 2023

Assignment 1. Catalan Word Vectors. Training and analysis

Improve CBOW MODEL

Position-dependent Weighting. The standard CBOW model (CBOW Training notebook) sums all the context word vectors with the same weight. Implement and evaluate a weighted sum of the context words with:

a) **A fixed scalar weight, e.g, (1,2,3,3,2,1) to give more weight to the words that are closer to the predicted central word.**
b) **A trained scalar weight for each position.**
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.**

Prior to the implementation of the three proposed methods, one can already take a good guess about which are the ones that are going to perform better. Therefore, we expect the models to be better the more trainable parameters they add to weight the words. After evaluating the three models, we have obtained the following results:

Method	Training Accuracy (%)	Val Accuracy Wikipedia (%)	Val Accuracy Periódico (%)
Fixed scalar weight	33.6	31.5	21.3
Trained scalar weight	35.4	33.1	22.9
Trained vector weight	41.1	39.1	28.9

The first method is quite simple and does not adapt to the input, but remain invariant. Although giving more weight to central words and less to the ones that are further is a good initial point, fixed scalars have limited a scope.

The second method, as expected, gives a better result than the first one. It is interesting to note that the learned weights tend to give more weight to the central words. This evidences that the first method has a good starting point, given that even when we have initialized the weights randomly, similar distributions are reached.

The last method provides a weight for each position of the embedding. Even though adding parameters can lead to overfit, this is not the case in this model and the accuracy improves considerably for validation and training sets, which makes it the best model.

Both codes can be found in the attached notebook.

d) (Optional) Hyperparameter optimization: study the performance of the model as a function of one of its parameters: the embedding size, batch size, optimizer, learning rate/scheduler, number of epochs, sharing input/output embeddings.

From the previous experiments it has been proved that the model that achieve higher accuracy is the one that uses a vector of trainable weights to rate each embedding. Therefore, we are going to use it as the base for the experiments. In this case, we have selected two hyperparameters to tune, which are the embedding size and the number of context words considered (window size). The values chosen has been 50-100-300 for the former and 3-7-15 for the latter.

Embedding Size	Window Size	Training Accuracy (%)	Val Accuracy Wikipedia (%)	Val Accuracy Periódico (%)
50	**3**	37.4	35.2	25.8
	7	37.3	35.3	25.4
	15	37.3	35.3	25.5
100	**3**	41.4	39.2	28.7
	7	41.1	39.1	28.9
	15	41.1	39.0	28.8
300	**3**	44.8	42.0	31.0
	7	44.7	42.1	30.7
	15	44.7	42.0	30.9
400	**7**	45.2	42.4	30.8

By looking at these results one can figure out that, on the one hand, the window size does not seem to be so relevant when it comes to improving the accuracy metric. Its effect seems to be random, or at least not linear or quadratic, given that in some cases increasing the window size improves the accuracy whereas other times is the opposite. On the other hand, we can observe that an increase in the size of the embeddings contributes to enhance the metric. This result is quite intuitive as the more variables to model words, the more precise the representations are in the embedding space and the better they can process.

Assignment 1. Catalan Word Vectors. Training and analysis

Evaluate the performance of the improved word vectors

a) **Implement the WordVector class (Word Vector Analysis notebook) with the most_similar and analogy methods to find closest vectors and analogies using the cosine similarity measure.**

For the first exercise, we have implemented functions that are responsible for finding the words most similar to a given word. To calculate this distance, we have calculated the cosine similarity between the embedding vectors of both words. Afterwards, we have saved each word with its distance in a dictionary and sorted this dictionary.

To calculate the analogy of a word, we have followed the same procedure, but in this case, we had to find the closest words to the word resulting from the following operation with respect to the embedding vectors y1 + (x2 - x1).

Both codes can be found in the attached notebook.

b) **Intrinsic evaluation: perform an informal evaluation finding good and bad examples of closest words and analogies. You can analyze the behavior of the CBOW word vectors for words with multiple meanings, synonyms, and antonyms, word frequency, different types of analogies, bias (gender, race, sexual orientation, etc.).**

We have searched for different words to see how our model performs. For common or well-known words, the model works quite well. We see some examples:

First, we will see the ones that work correctly:

PILOTA		MATEMÀTIQUES		INFANTESA		CATALÀ	
COPA	0.62	MATEMÀTICA	0.65	INFÀNCIA	0.92	VALENCIÀ	0.83
BARRA	0.60	FILOSOFIA	0.62	JOVENTUT	0.73	BASC	0.73
SAMARRETA	0.59	CIÈNCIES	0.62	ADOLESCÈNCIA	0.60	MALLORQUÍ	0.70
BANQUETA	0.58	ASTROFÍSICA	0.61	VIDA	0.54	GALLEC	0.70
BICICLETA	0.57	TEOLOGIA	0.60	TRAJECTÒRIA	0.53	ESPANYOL	0.67
BOLA	0.57	MEDECINA	0.59	CAPTIVITAT	0.52	ANDALÚS	0.64
RULETA	0.57	ASTRONOMIA	0.59	SOLITUD	0.51	ARAGONÈS	0.62
LLIGA	0.56	MEDICINA	0.59	XICOTA	0.51	FRANCÈS	0.60
BOXA	0.55	SOCIOLOGIA	0.59	MARE	0.50	NAVARRÈS	0.58
PORTERIA	0.53	TEÒRICA	0.58	COMPANYA	0.50	ASTURIÀ	0.58

And then we will see a few examples that show biases in society and therefore are not entirely correct:

TERRORISTA		PEDERÀSTIA		GAI		HOMOSEXUAL	
PARAMILITAR	0.57	CONTRAREFORMA	0.52	HOMOSEXUAL	0.70	GAI	0.70
TERRORISTES	0.56	VERACITAT	0.50	LESBIANA	0.58	SEXUAL	0.62
ISLAMISTA	0.54	MOTIVACIÓ	0.49	TRANSGÈNERE	0.57	LESBIANA	0.56
GUERRILLERS	0.51	PÍTICA	0.49	LGBT	0.55	HOMOSEXUALS	0.55
REBEL	0.50	CÀBALA	0.48	AFROAMERICANA	0.52	RACISTA	0.55
DISSIDENT	0.50	TEMPORALITAT	0.48	TRANSSEXUAL	0.52	HOMOSEXUALITAT	0.53
SEPARATISTA	0.50	PERESTROIKA	0.48	HOMOSEXUALS	0.51	INFELIÇ	0.52
PERPETRAT	0.48	DISCREPÀNCIA	0.48	LGTB	0.50	BISEUXAL	0.52
PARTISÀ	0.48	PSICODÈLIA	0.48	BLOGUER	0.50	CÒNJUGE	0.52
GUERRILLER	0.48	HISTORICITAT	0.47	CORRUPTE	0.47	VIOLENT	0.52

PROSTITUTA		TRANSSEXUAL		LGTB	
NOIA	0.70	GAI	0.52	LGBT	0.74
NENA	0.69	LESBIANA	0.51	LGBTI	0.64
BRUIXA	0.63	REPORTERA	0.49	TRANSGÈNERE	0.57
INSTITUTRIU	0.61	HETEROSEXUAL	0.48	LGTBI	0.55
INFERMERA	0.61	DRAMATURGA	0.48	MARGINATS	0.52
MAINADERA	0.60	PROSTITUTA	0.48	GAI	0.50
CONCUBINA	0.60	PHILO	0.47	DESFAVORITS	0.49
SERVENTA	0.58	AFROAMERICANA	0.47	MAPUTXE	0.48
CAMBRERA	0.57	TRANSGÈNERE	0.47	QUEER	0.48
NINA	0.57	GONZO	0.47	ASSISTENCIAL	0.48

It can be seen how those cells in the table marked in red correspond to inappropriate examples, or that show an unfair bias that we have in society.

Now let's look at a few analogies to evaluate it as well:

(FRANÇA, FRANCESA, RÚSSIA)		(NOI, DIRECTOR, NOIA)		(NOIA, DIRECTORA, NOI)	
RUSSA	0.84	CANTANT	0.91	DIRECTOR	0.95
ALEMANYA	0.75	COMPANYIA	0.91	GENERT	0.95
POLONESA	0.72	PRESIDENT	0.91	EDITORA	0.94
AUSTRÍACA	0.69	MEMBRE	0.90	REALITZADOR	0.94
UCRAÏNESA	0.67	PRODUCCIÓ	0.90	DOCENT	0.94

We were able to play with both models and we saw how model 1, which corresponds to the weights of the embeddings, works better than model 2, which corresponds to the weights of the linear layer. This is because when calculating Word embeddings, embeddings work better than the linear layer, which would be more suitable for making predictions. We have seen how model 1 has placed closer words where they should be and has been more accurate in the obtained examples, although as we have shown above, there is always an unsuitable or erroneous example that comes out.

c) (Optional) Visualize word analogies or word clustering properties

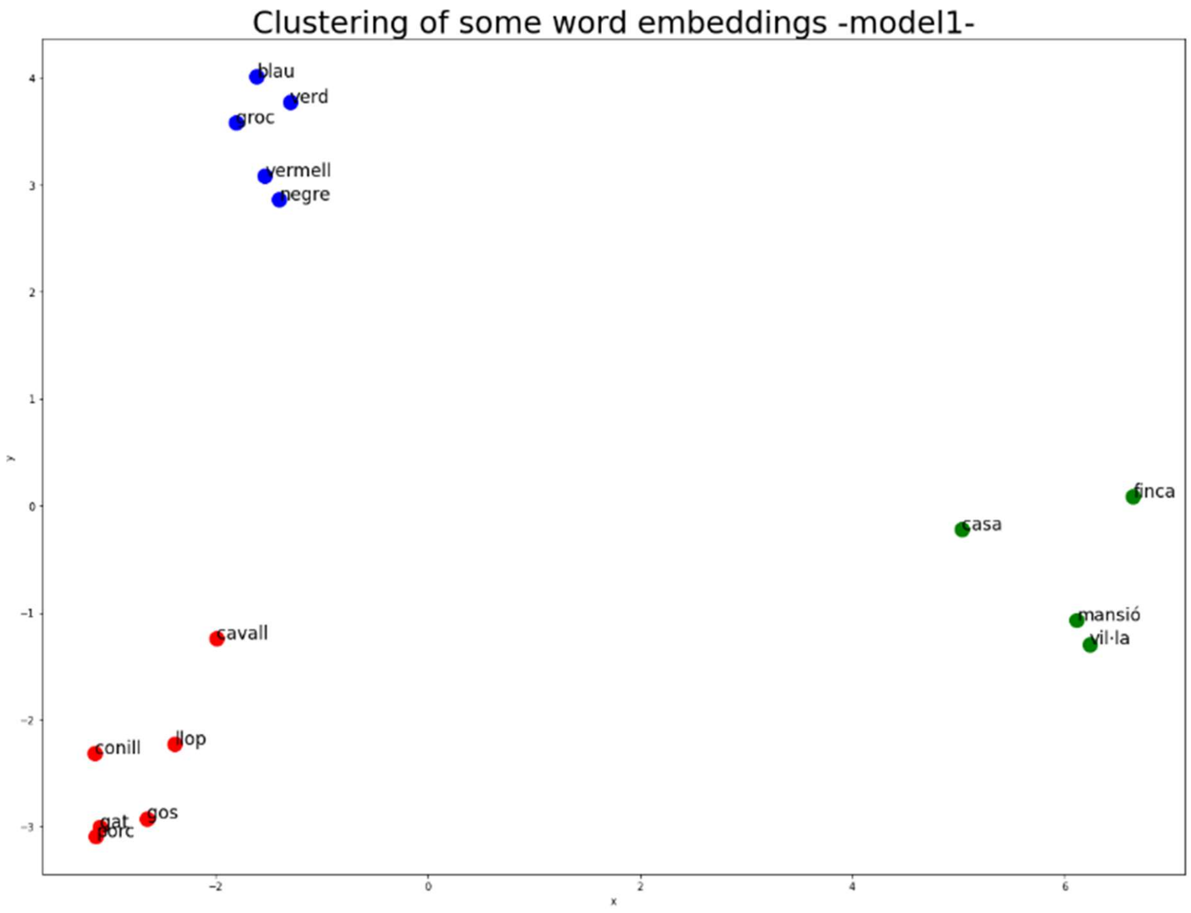
For the next section, we are asked to visualize different words to see in which cluster they are or which nearby words they have. As we have many words, we are aware that visualizing the entire vocabulary would mean not being able to see each word well, so we decided to take 11 words:

words_to_vis = ["gos", "gat", "llop", "cavall", "conill", "porc", "casa", "finca", "mansió", "vil·la", "groc", "blau", "verd", "negre", "vermell"]

To visualize them, we need to use a dimensionality reduction technique since the embedding vectors are of dimension 100, and therefore we cannot visualize them. We have used PCA with 2 principal components to see each word in the Cartesian plane.

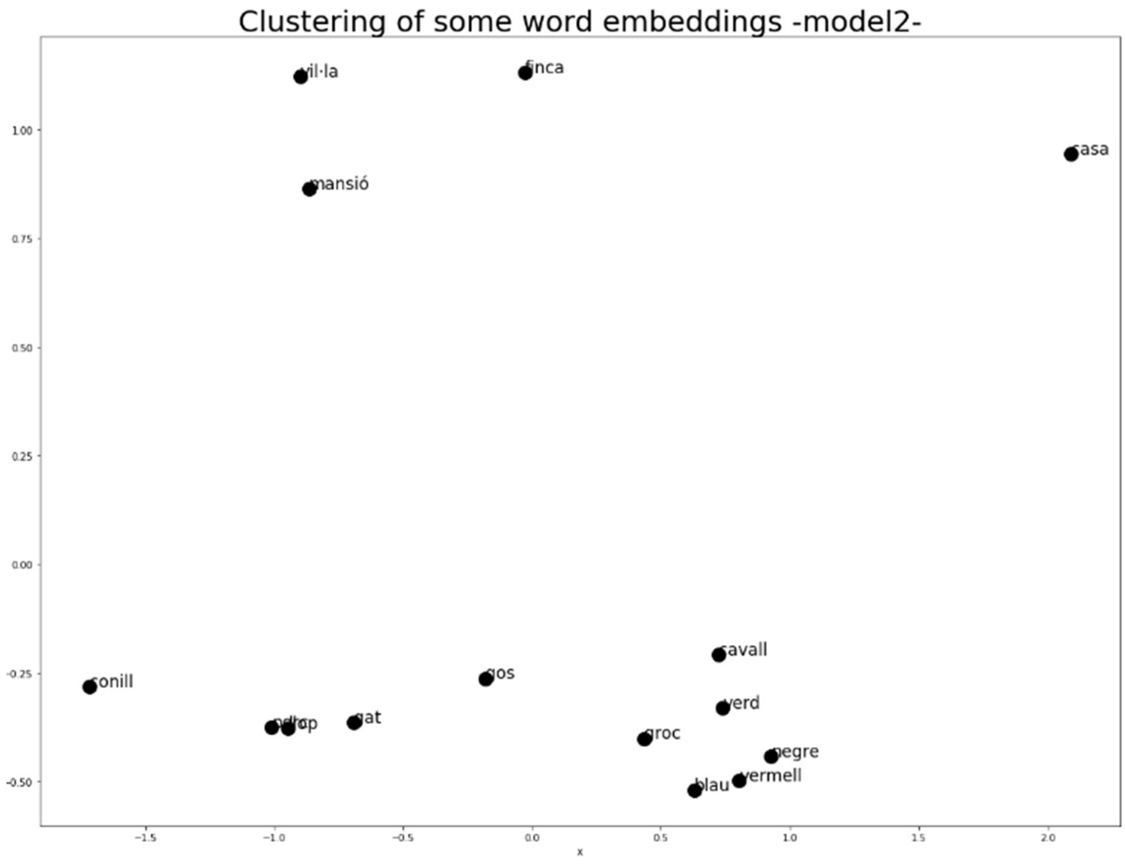
It is expected that in the graph, we can see three well-differentiated clusters, one for animals, one for types of houses, and the other for different colours. The model used is model 1, that corresponds to the embedding weights.

The provided graph is as follows:



We have coloured each cluster differently, and we can see how the three different groups of words are clearly distributed in different spaces, allowing us to correctly differentiate words with a similar meaning.

If we now try to do the same with model 2, we see that it does not group as well:



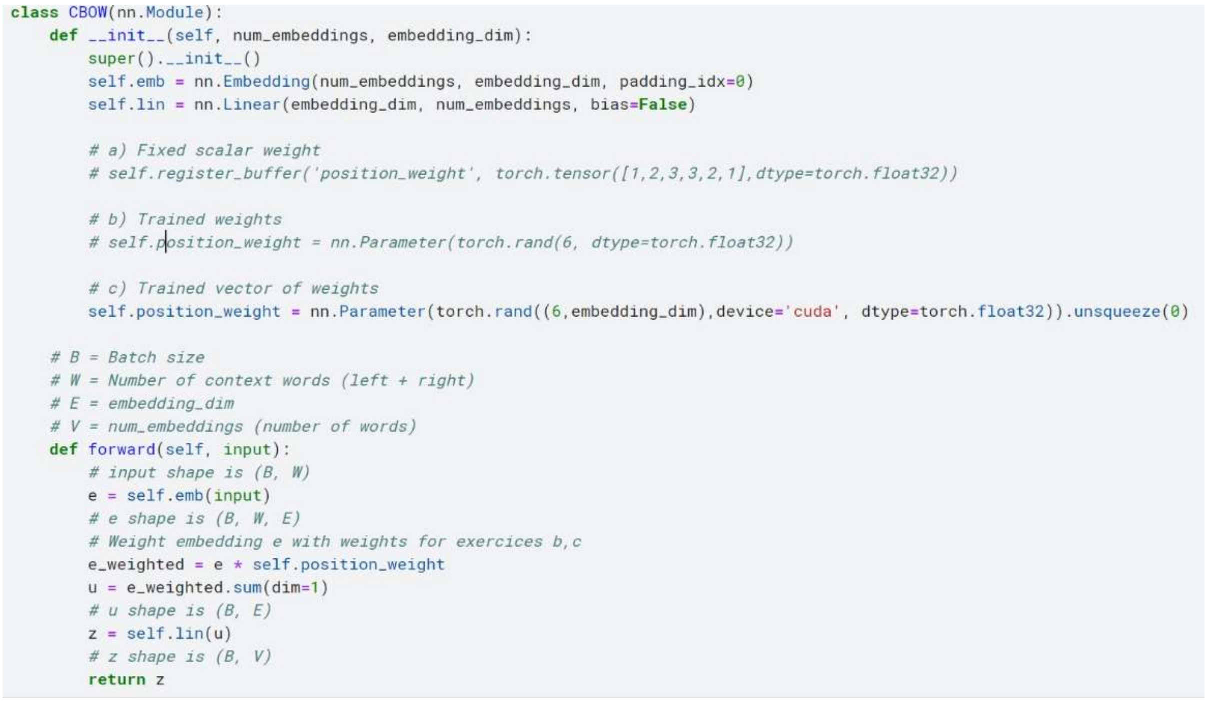
Similar words are not as near as in model 1 and group of colours and animals are in same cluster.

d) **(Optional) Prediction accuracy: compare 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 3 previous and 3 next words). To obtain your score on the competition test set you have to commit your version of the CBOW Training notebook, wait until the training completes, and then "Submit to competition" the obtained file (submission.csv) in the Output section of the notebook.**

We have done the submission and we have reached a 0.30166 of accuracy. The configuration is an embedding dimension of size 300 and a window size of 15.

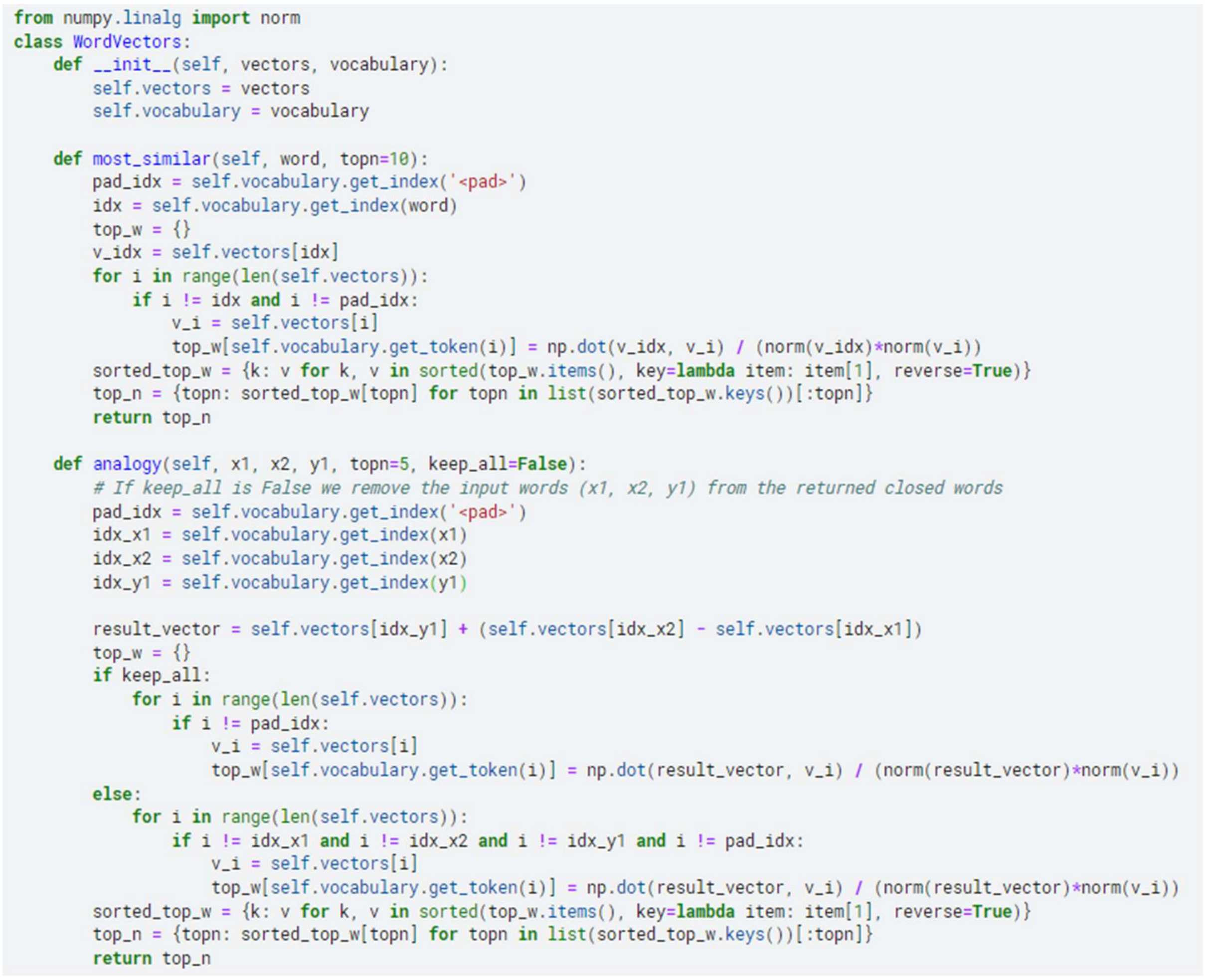
Assignment 1. Catalan Word Vectors. Training and analysis

CODES



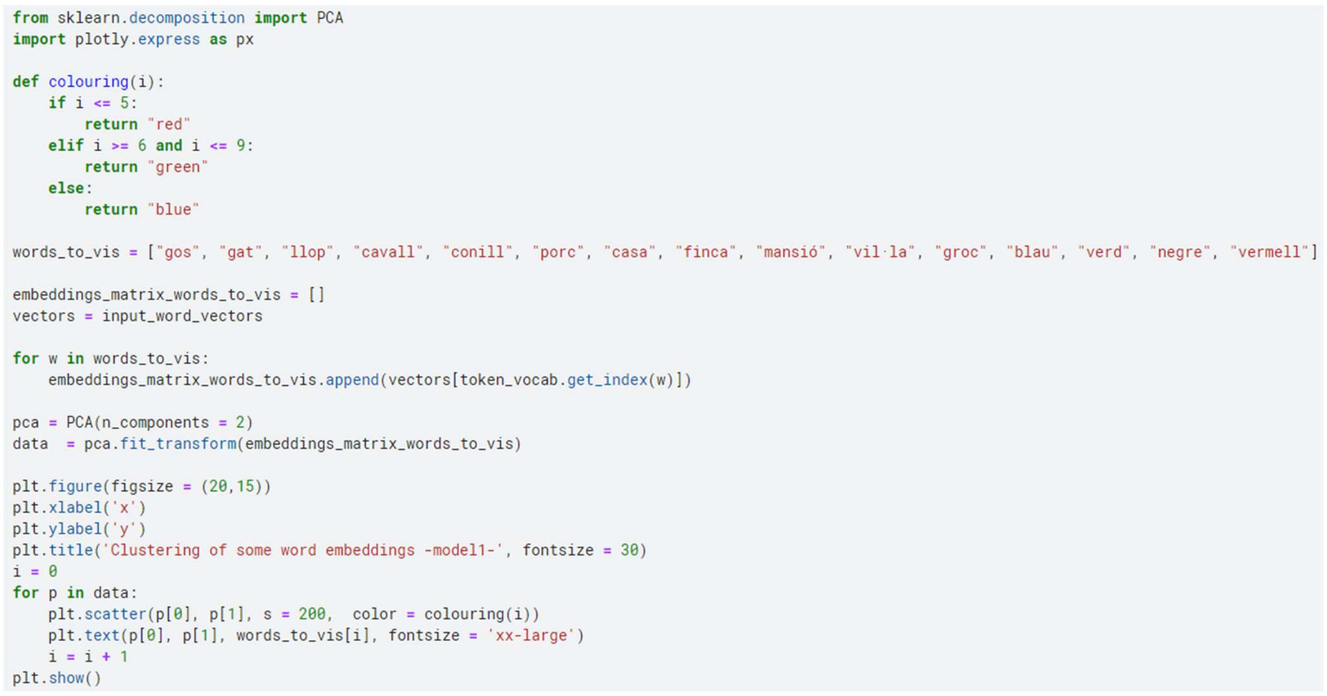
```
class CBOW(nn.Module):
    def __init__(self, num_embeddings, embedding_dim):
        super().__init__()
        self.emb = nn.Embedding(num_embeddings, embedding_dim, padding_idx=0)
        self.lin = nn.Linear(embedding_dim, num_embeddings, bias=False)

        # a) Fixed scalar weight
        # self.register_buffer('position_weight', torch.tensor([1,2,3,3,2,1],dtype=torch.float32))

        # b) Trained weights
        # self.position_weight = nn.Parameter(torch.rand(6, dtype=torch.float32))

        # c) Trained vector of weights
        self.position_weight = nn.Parameter(torch.rand((6,embedding_dim),device='cuda', dtype=torch.float32)).unsqueeze(0)

    # B = Batch size
    # W = Number of context words (left + right)
    # E = embedding_dim
    # V = num_embeddings (number of words)
    def forward(self, input):
        # input shape is (B, W)
        e = self.emb(input)
        # e shape is (B, W, E)
        # Weight embedding e with weights for exercices b,c
        e_weighted = e * self.position_weight
        u = e_weighted.sum(dim=1)
        # u shape is (B, E)
        z = self.lin(u)
        # z shape is (B, V)
        return z
```

CBOW CODE



```
from numpy.linalg import norm
class WordVectors:
    def __init__(self, vectors, vocabulary):
        self.vectors = vectors
        self.vocabulary = vocabulary

    def most_similar(self, word, topn=10):
        pad_idx = self.vocabulary.get_index('<pad>')
        idx = self.vocabulary.get_index(word)
        top_w = {}
        v_idx = self.vectors[idx]
        for i in range(len(self.vectors)):
            if i != idx and i != pad_idx:
                v_i = self.vectors[i]
                top_w[self.vocabulary.get_token(i)] = np.dot(v_idx, v_i) / (norm(v_idx)*norm(v_i))
        sorted_top_w = {k: v for k, v in sorted(top_w.items(), key=lambda item: item[1], reverse=True)}
        top_n = {topn: sorted_top_w[topn] for topn in list(sorted_top_w.keys())[:topn]}
        return top_n

    def analogy(self, x1, x2, y1, topn=5, keep_all=False):
        # If keep_all is False we remove the input words (x1, x2, y1) from the returned closed words
        pad_idx = self.vocabulary.get_index('<pad>')
        idx_x1 = self.vocabulary.get_index(x1)
        idx_x2 = self.vocabulary.get_index(x2)
        idx_y1 = self.vocabulary.get_index(y1)

        result_vector = self.vectors[idx_y1] + (self.vectors[idx_x2] - self.vectors[idx_x1])
        top_w = {}
        if keep_all:
            for i in range(len(self.vectors)):
                if i != pad_idx:
                    v_i = self.vectors[i]
                    top_w[self.vocabulary.get_token(i)] = np.dot(result_vector, v_i) / (norm(result_vector)*norm(v_i))
        else:
            for i in range(len(self.vectors)):
                if i != idx_x1 and i != idx_x2 and i != idx_y1 and i != pad_idx:
                    v_i = self.vectors[i]
                    top_w[self.vocabulary.get_token(i)] = np.dot(result_vector, v_i) / (norm(result_vector)*norm(v_i))
        sorted_top_w = {k: v for k, v in sorted(top_w.items(), key=lambda item: item[1], reverse=True)}
        top_n = {topn: sorted_top_w[topn] for topn in list(sorted_top_w.keys())[:topn]}
        return top_n
```

EVALUATION CODE



```
from sklearn.decomposition import PCA
import plotly.express as px

def colouring(i):
    if i <= 5:
        return "red"
    elif i >= 6 and i <= 9:
        return "green"
    else:
        return "blue"

words_to_vis = ["gos", "gat", "llop", "cavall", "conill", "porc", "casa", "finca", "mansió", "vil·la", "groc", "blau", "verd", "negre", "vermell"]

embeddings_matrix_words_to_vis = []
vectors = input_word_vectors

for w in words_to_vis:
    embeddings_matrix_words_to_vis.append(vectors[token_vocab.get_index(w)])

pca = PCA(n_components = 2)
data  = pca.fit_transform(embeddings_matrix_words_to_vis)

plt.figure(figsize = (20,15))
plt.xlabel('x')
plt.ylabel('y')
plt.title('Clustering of some word embeddings -model1-', fontsize = 30)
i = 0
for p in data:
    plt.scatter(p[0], p[1], s = 200,  color = colouring(i))
    plt.text(p[0], p[1], words_to_vis[i], fontsize = 'xx-large')
    i = i + 1
plt.show()
```

PCA CODE