Introduction to NLP

Assignment 3

Negative Sampling:

What is Negative Sampling?

Negative Sampling is technique used instead of softmax. So in softmax when there is large units such as vocabulary of the dataset, etc. it becomes computationally very expensive to calculate softmax of millions of outputs. Where as in negative sampling it is very less computation as it applies to a fraction(very small fraction) of the vocabulary and the model trained using this can reach very close to traditional softmax.

How do we approximate the word2vec training computation using the technique?

In Negative Sampling, the goal is to train the model to correctly identify the next word given a current word. To achieve this, the model is trained to assign a higher probability to the correct next word than to any of the negative samples. The probability of a word being the next word is calculated using the sigmoid function instead of the softmax function used in traditional methods. The sigmoid function is a mathematical function that maps any input to a value between 0 and 1. By using the sigmoid function, the model can be trained to assign a probability of 1 to the correct next word and a probability of 0 to the negative samples.

Advantages of Negative Sampling

One of the biggest advantages of Negative Sampling is that it is computationally efficient. Traditional methods like softmax require computing the probability of every

word in the vocabulary, which can be computationally expensive, especially when the vocabulary size is large. Negative Sampling, on the other hand, only considers a small subset of the vocabulary, making it much faster and more efficient.

Another advantage of Negative Sampling is that it can lead to better word embeddings. By focusing on differentiating between the correct next word and negative samples, the model can learn more about the relationship between words and generate more accurate word embeddings.

Semantic Similarity

Semantic similarity is a measure of how closely related two words or phrases are in meaning. Word embeddings can be used to measure semantic similarity by representing words as vectors in a high-dimensional space, where words with similar meanings are located closer together.

Two Techniques

Cosine similarity measures the cosine of the angle between two word vectors and ranges from -1 to 1, where values closer to 1 indicate greater similarity.

Euclidean distance measures the distance between two word vectors and ranges from 0 to infinity, where smaller distances indicate greater similarity.

Word2Vec by Singular Value Decomposition

Window based Co-occurance Matrix

A Co-occurance matrix looks like this

$$
X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array}
\begin{array}{c}
\begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\
\begin{bmatrix}
0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

The sentences are

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

As one can see it is always symmetric unless one tries to contruct it with a direction.

We now perform SVD on X, to get $X = USV^T$

Then we take first k columns of U to get a k dimensional word vectors.

The above represents the amount of variance in first k dimensions.

Applying SVD to X:

$$\underset{|V|}{\underset{|V|}{X}} = \underset{|V|}{\underset{|V|}{\begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix}}} \underset{|V|}{\underset{|V|}{\begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}}} \underset{|V|}{\underset{|V|}{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix}}}$$

Reducing dimensionality by selecting first k singular vectors:

$$\underset{|V|}{\underset{|V|}{\hat{X}}} = \underset{|V|}{\underset{k}{\begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix}}} \underset{k}{\underset{k}{\begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}}} \underset{k}{\underset{|V|}{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix}}}$$

So why are we taking first k and not any other k or random k?

It is because the order of influence or the importance is in decreasing order. But Why decreasing order and what do you mean by importance?

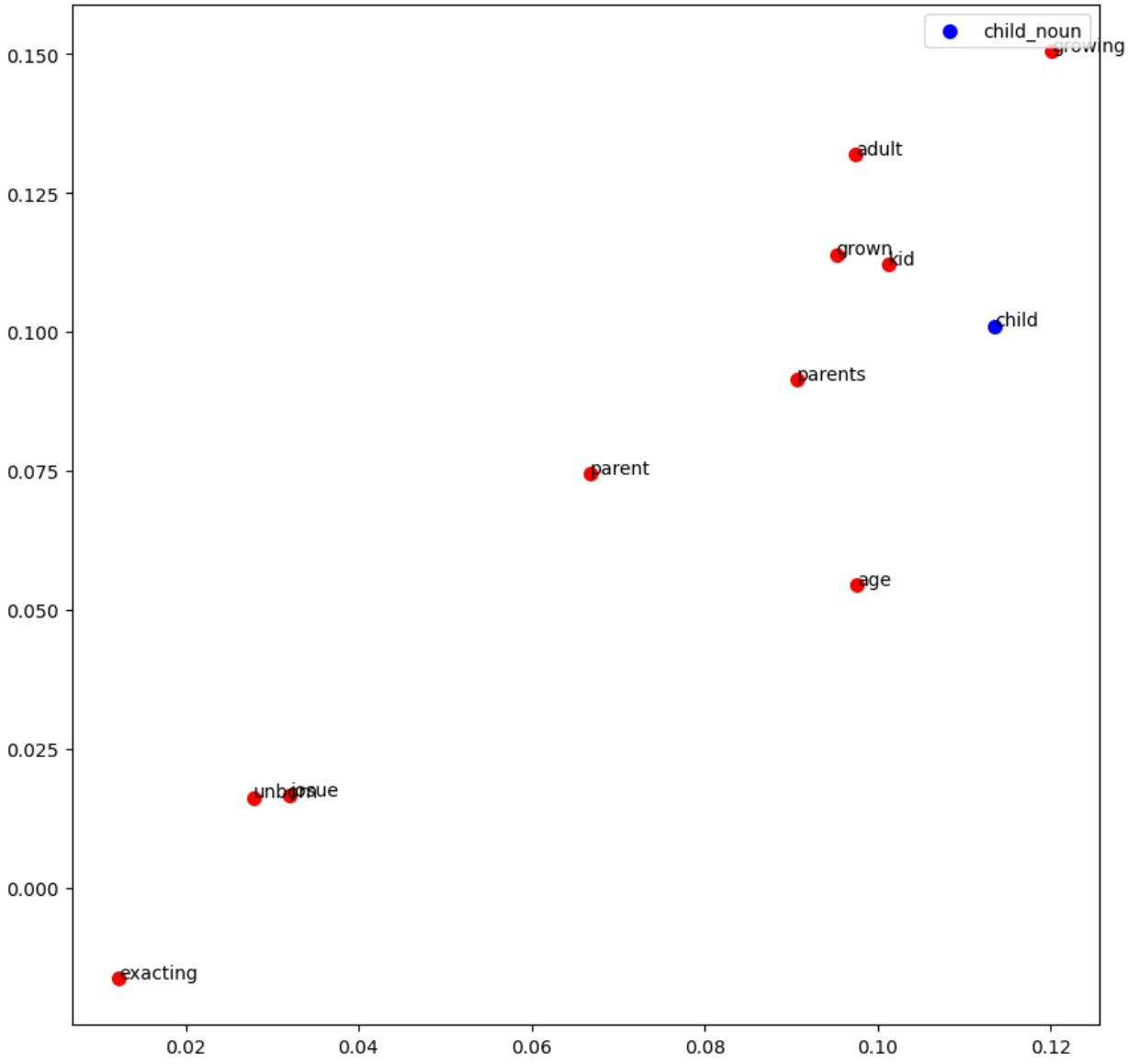
> SVD (Singular Value Decomposition) is a powerful linear algebra tool that can be used to factorize a matrix into three matrices, $A = U\Sigma V^T$, where A is the original matrix, U and V are orthogonal matrices, and Σ is a diagonal matrix of singular values. When we apply SVD to a matrix, the singular values in Σ are typically sorted in decreasing order. This is because the singular values represent the magnitudes of the principal components of the matrix, and the principal components capture the most important features of the data. The first few singular values (and corresponding columns of U and V) capture most of the variability in the data, while the later singular values (and corresponding columns of U and V) capture progressively smaller amounts of variability.

I used around 100 dimensions initially and then reduced it to get the most significant vectors.

The similarity is calculated using Euclidean distance.

The Plotting of vectors:

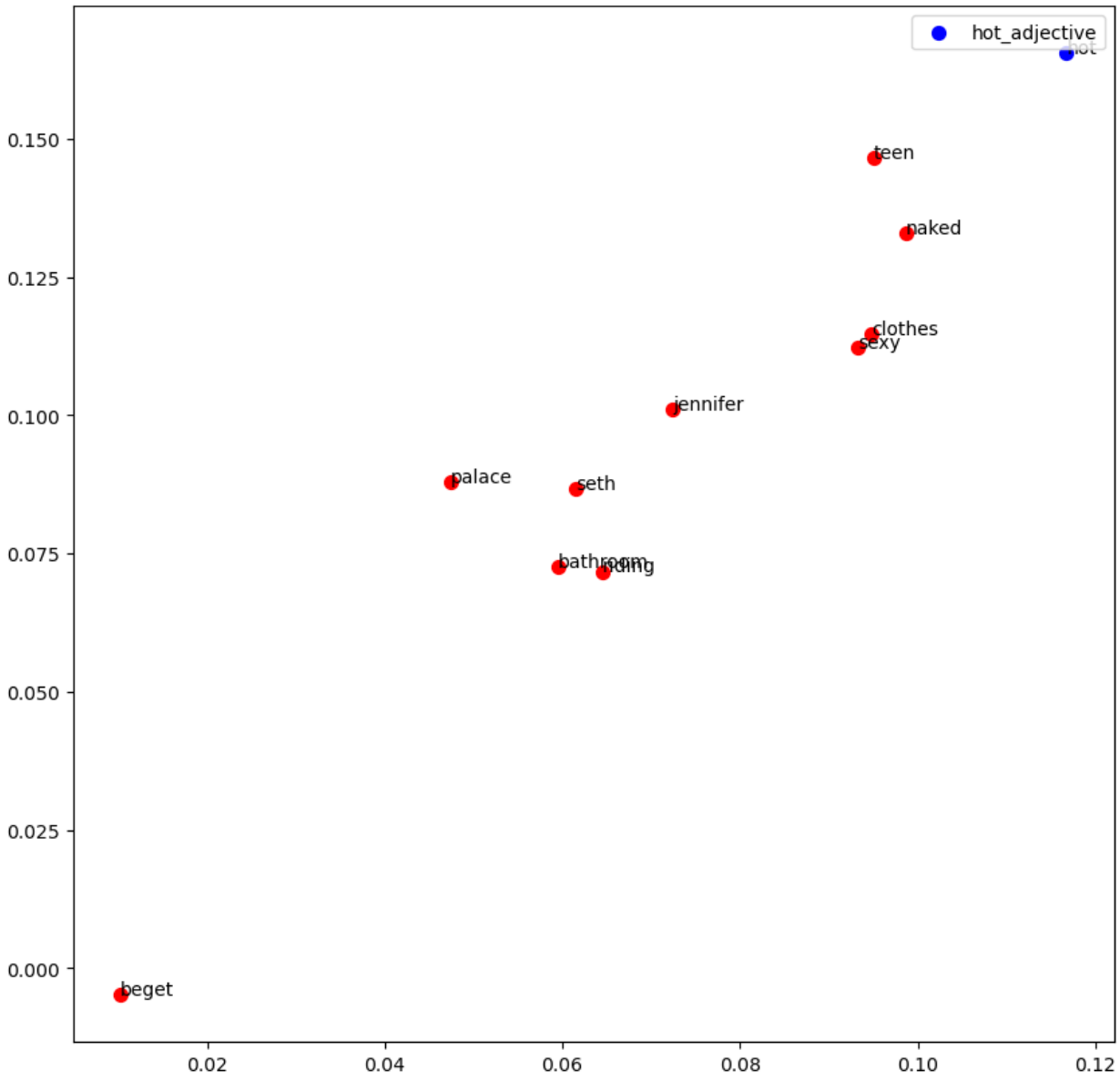
‘CHILD_noun’



	word	similarity
0	parent	0.6206430026
1	kid	0.6016792141
2	adult	0.601561122
3	parents	0.543893937

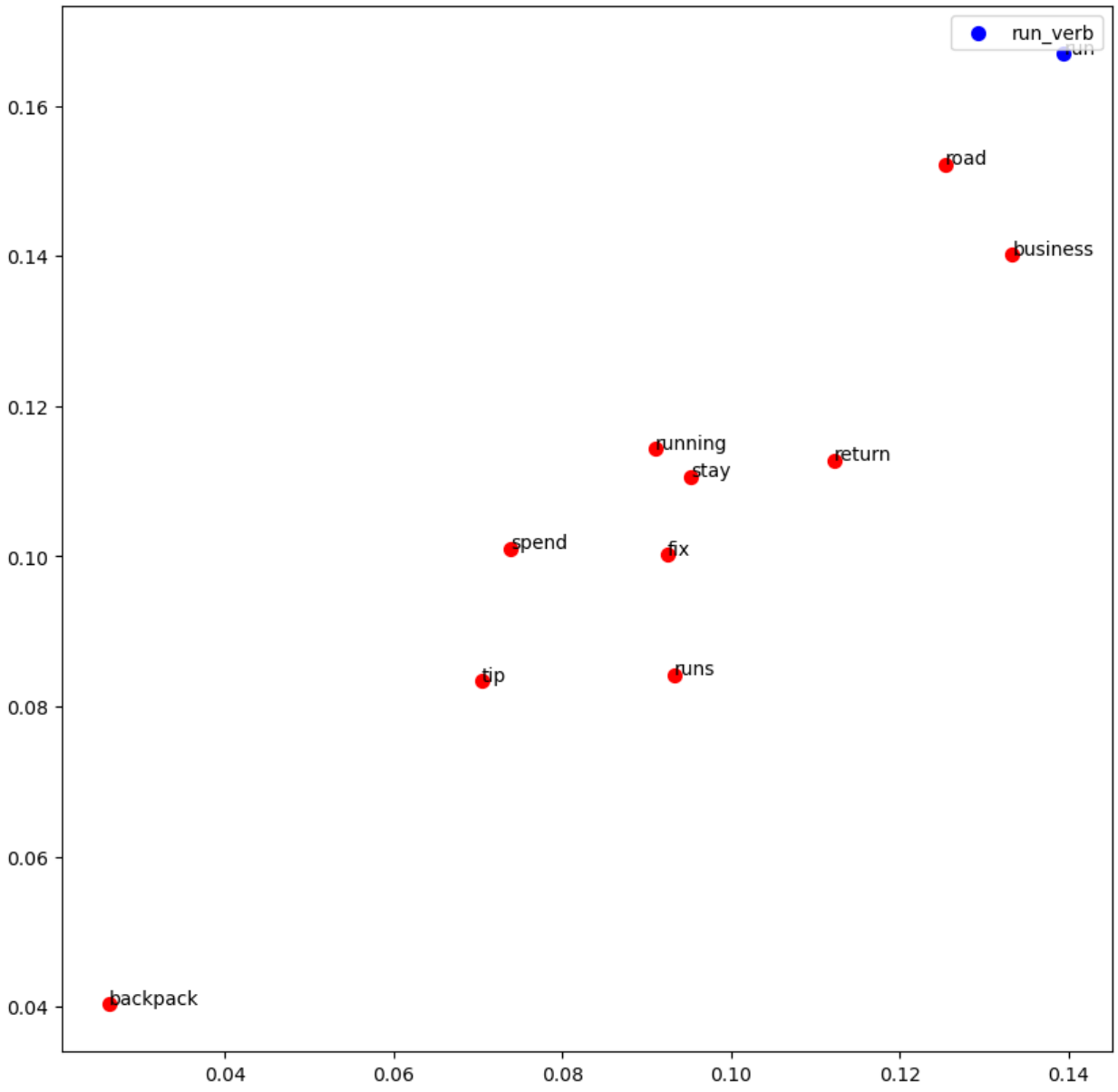
4	growing	0.5320784416
5	grown	0.5078449438
6	josue	0.4776090767
7	age	0.4756988053
8	exacting	0.4734862176
9	unborn	0.4659809907

‘HOT_adjective’:



	word	similarity
0	beget	0.6308043808
1	sexy	0.5250260976
2	naked	0.50937656
3	seth	0.5027756888
4	jennifer	0.4971787167
5	palace	0.4768984043
6	teen	0.4660170731
7	riding	0.4631012705
8	clothes	0.4523699647
9	bathroom	0.4483634605

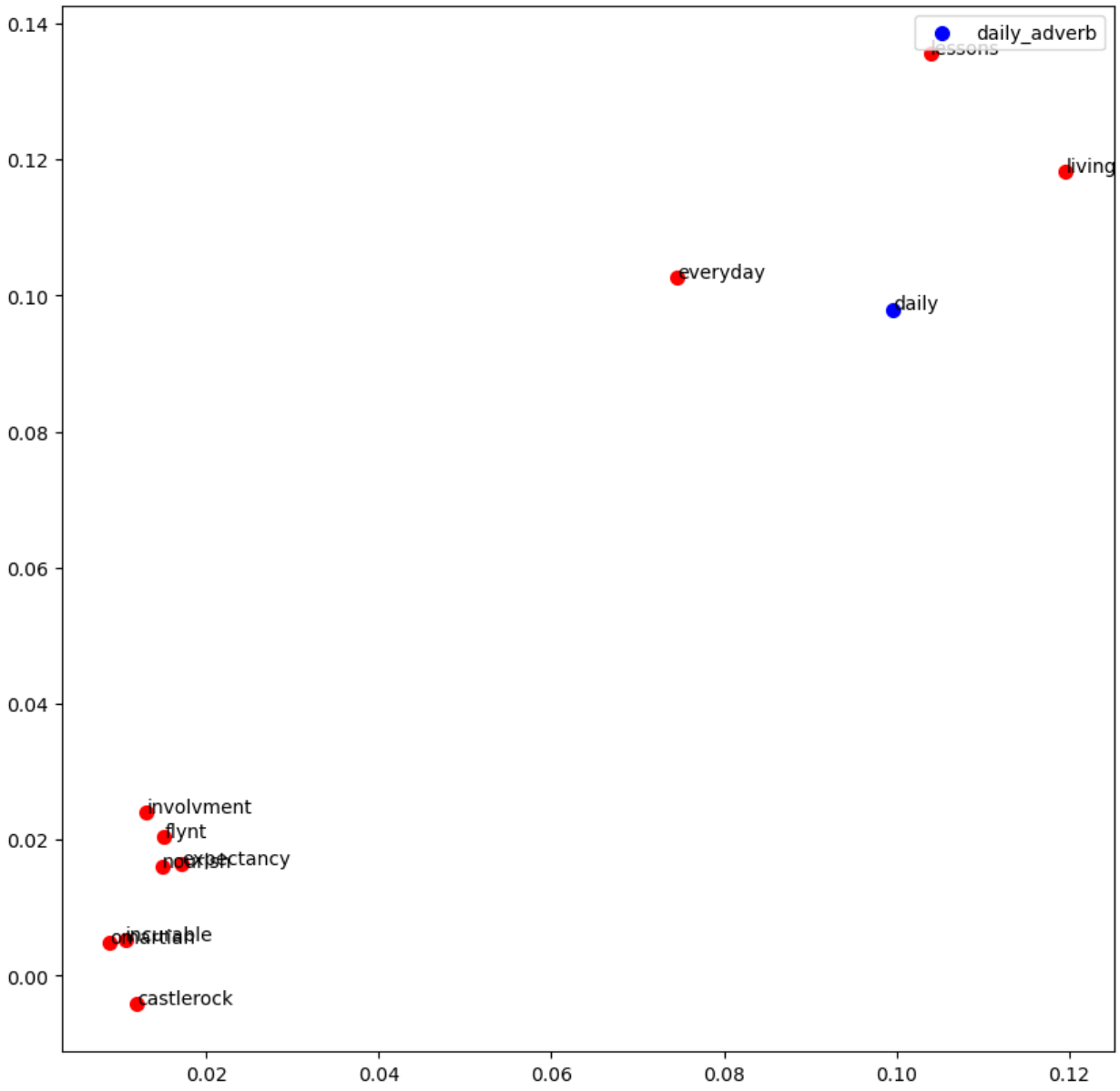
RUN_verb:



	word	similarity
0	running	0.5543230196
1	runs	0.5386585887
2	stay	0.4689356991
3	spend	0.4551744688
4	return	0.4412781134
5	fix	0.4364606934
6	tip	0.4307860464

7	backpack	0.430078964
8	business	0.4162575076
9	road	0.4155228739

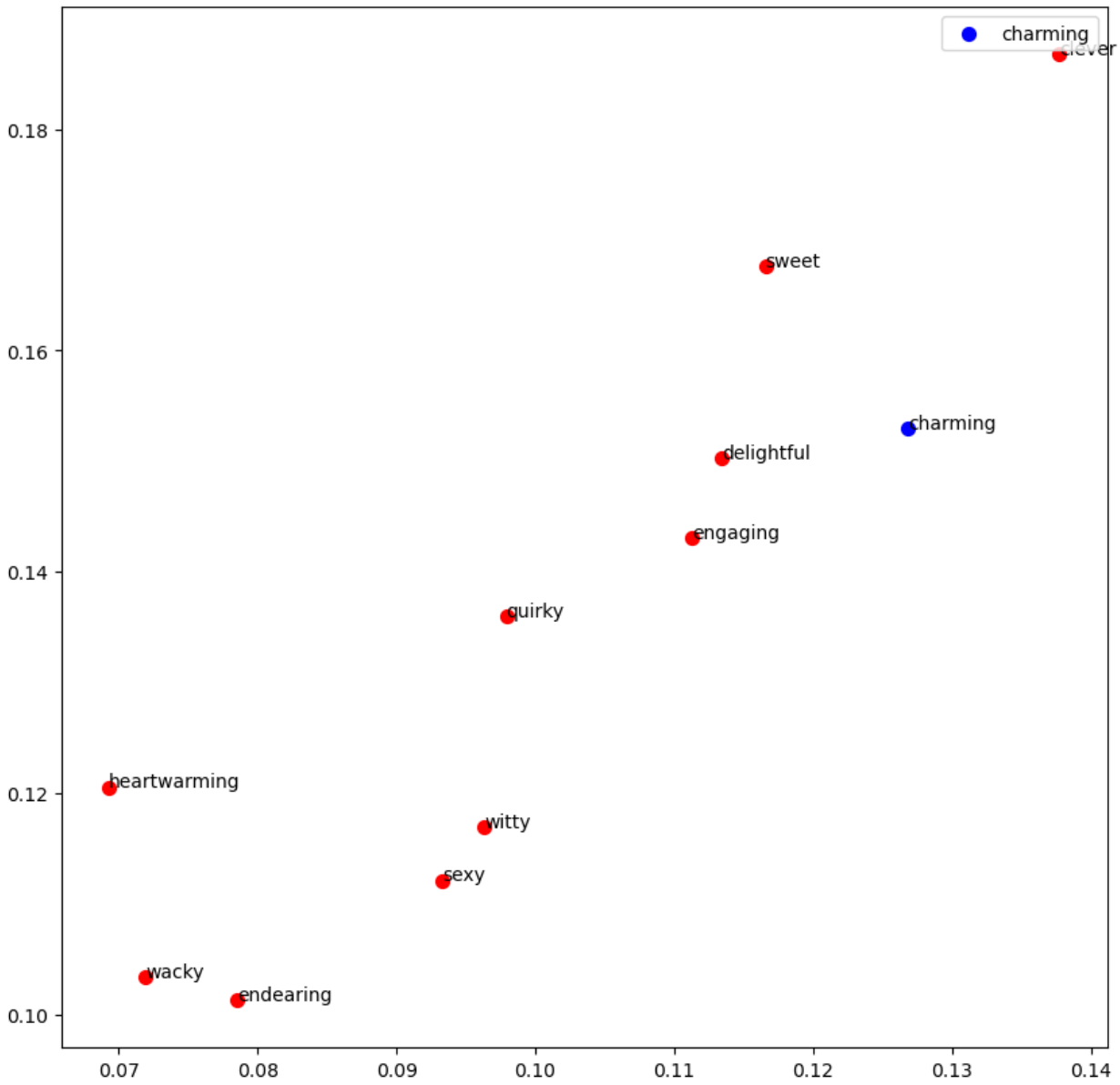
‘DAILY’:



	word	similarity
0	everyday	0.5975563808
1	flynt	0.5203701343

2	omartian	0.5142779243
3	involvment	0.5030758312
4	incurable	0.4972213557
5	living	0.4881605627
6	castlerock	0.4859938368
7	lessons	0.4737673909
8	nourish	0.4716447839
9	expectancy	0.4703301607

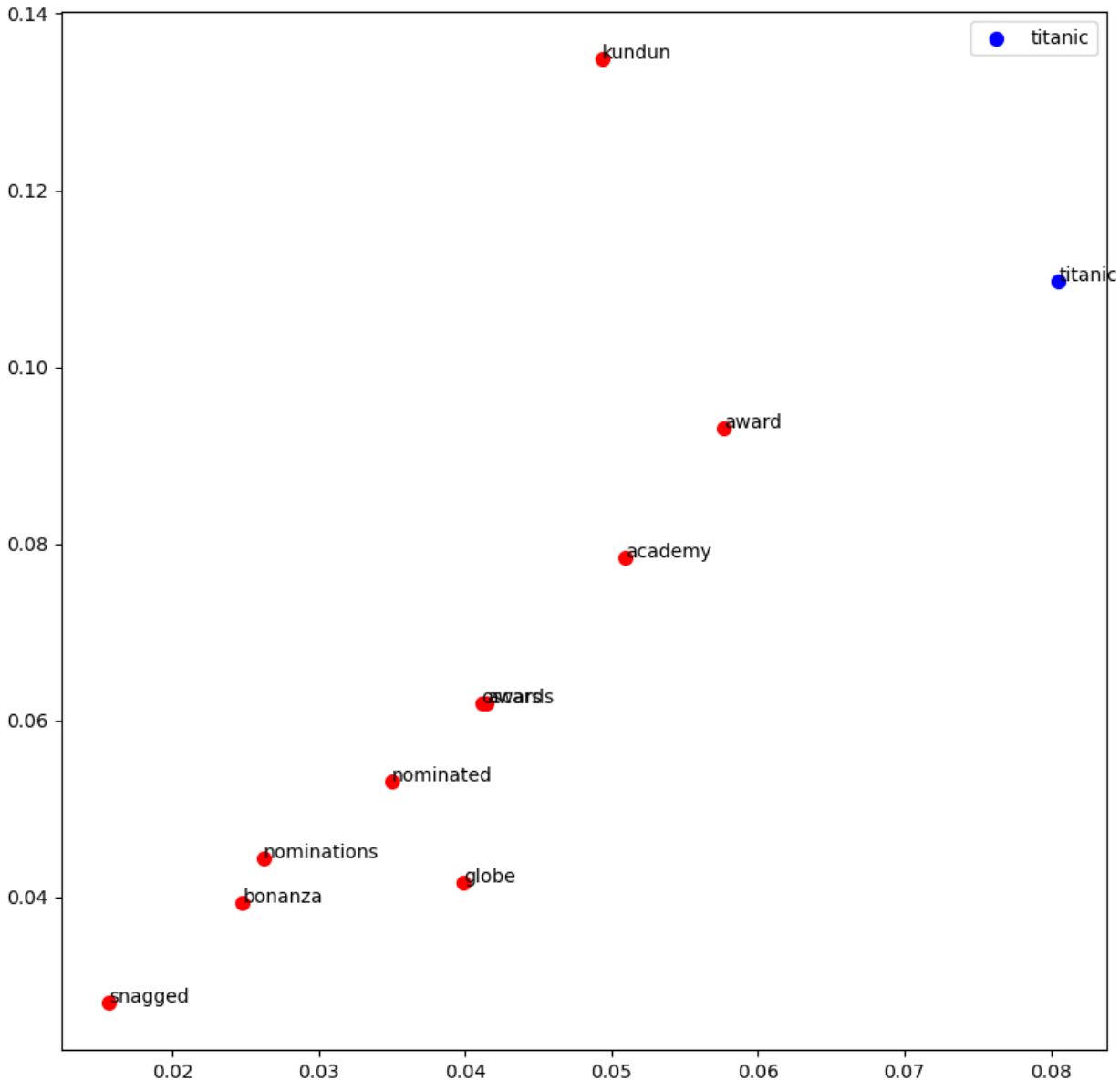
Charming_adj:



	word	similarity
0	witty	0.6257693657
1	delightful	0.6067004654
2	sweet	0.5221672481
3	sexy	0.5211201008
4	quirky	0.5093167354
5	wacky	0.5081041579
6	heartwarming	0.4933603112

7	engaging	0.4867221326
8	clever	0.4788328678
9	endearing	0.4748274145

‘Titanic’:



	word	similarity
0	awards	0.4966860492

1	nominated	0.4917653525
2	oscars	0.4849682835
3	academy	0.4663176735
4	snagged	0.4601805469
5	bonanza	0.4572511462
6	kundun	0.4457426014
7	nominations	0.4451990253
8	globe	0.440692007
9	award	0.4310442281

CBOW with Negative sampling

example ‘The cat jumped over the puddle’

In CBOW given {"The", "cat", ’over", "the’, "puddle"} as a context, we need to predict ‘jumped’.

suppose we have already trained our cbow. Now we have the one hot encodings and 2 matrices V(nx|V’|) and U(|V’|xn) . where V is the vocabulary and n is dimension of our embedding.

How do we Predict the word?

We breakdown the way this model works in these steps:

1. We generate our one hot word vectors $(x^{(c-m)}, ..., x^{(c-1)}, x^{(c+1)}, ..., x^{(c+m)})$ for the input context of size m.

2. We get our embedded word vectors for the context ($v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{(c-m+1)}, \ldots, v_{c+m} = \mathcal{V}x^{(c+m)}$)

3. Average them to get

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + ... + v_{c+m}}{2m}$$

4. Generate a score vector $z = U\hat{v}$
5. turn these score to probabilites by softmax $\hat{y} = softmax(z)$
6. These probabilties are nothing but one hot encoding ideally.

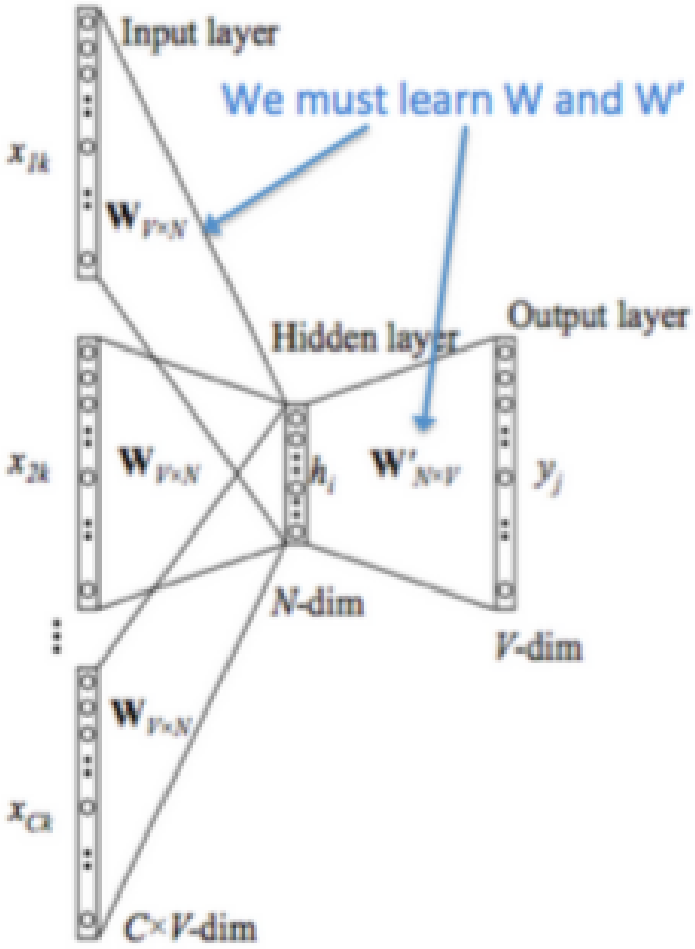


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

But how do you train these matrices ?

We can use cross entropy and backpropagation.

The loss function looks like

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log(\hat{y}_j)$$

But since y is one hot vector we can eliminate all except when its one.

Therefore our new equation becomes

$$H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

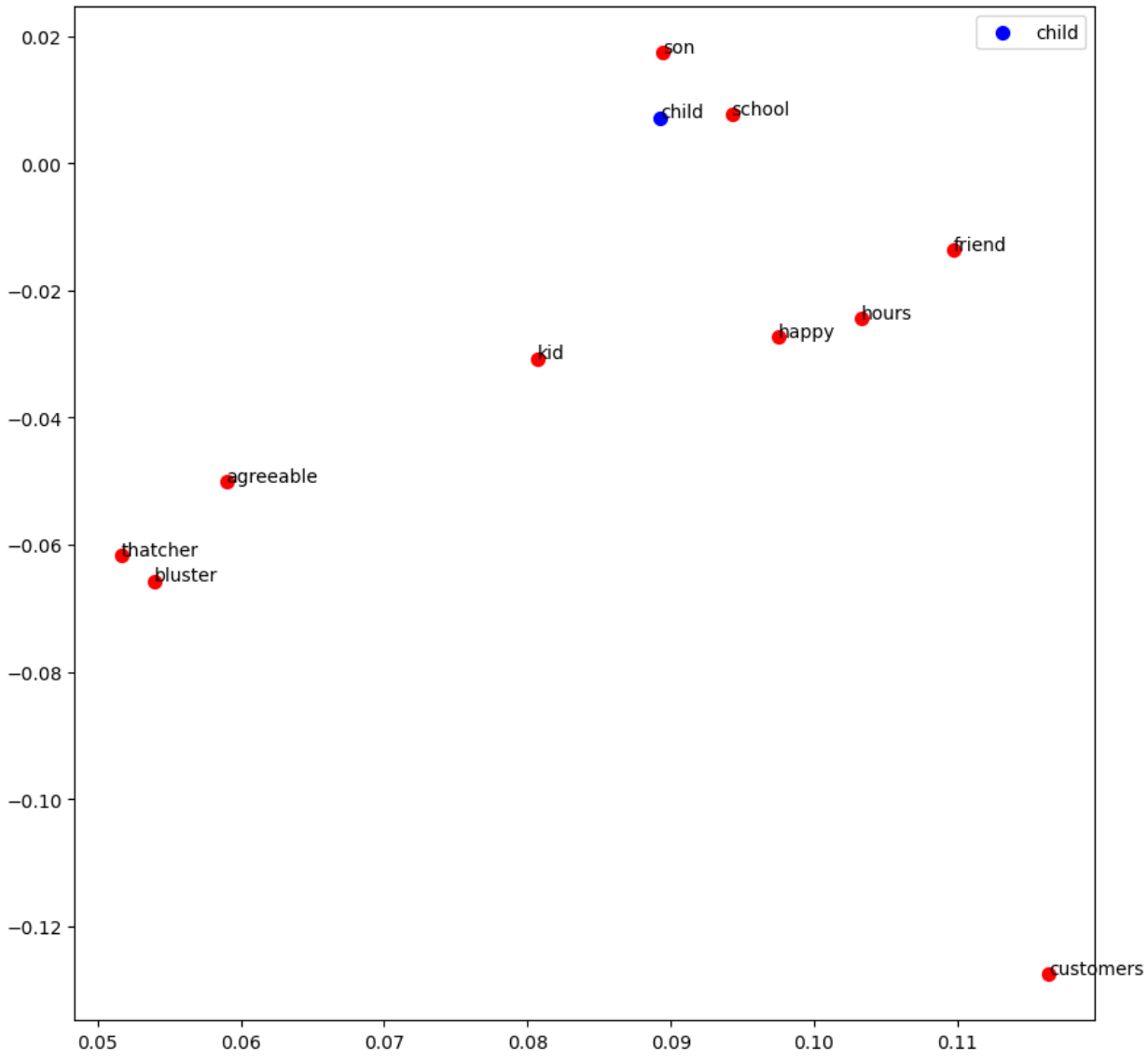
We calculate our loss and try to minimize it.

$$\begin{aligned}
\text{minimize } J &= -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= -\log P(u_c | \hat{v}) \\
&= -\log \frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})
\end{aligned}$$

Negative sampling is explained above

results:

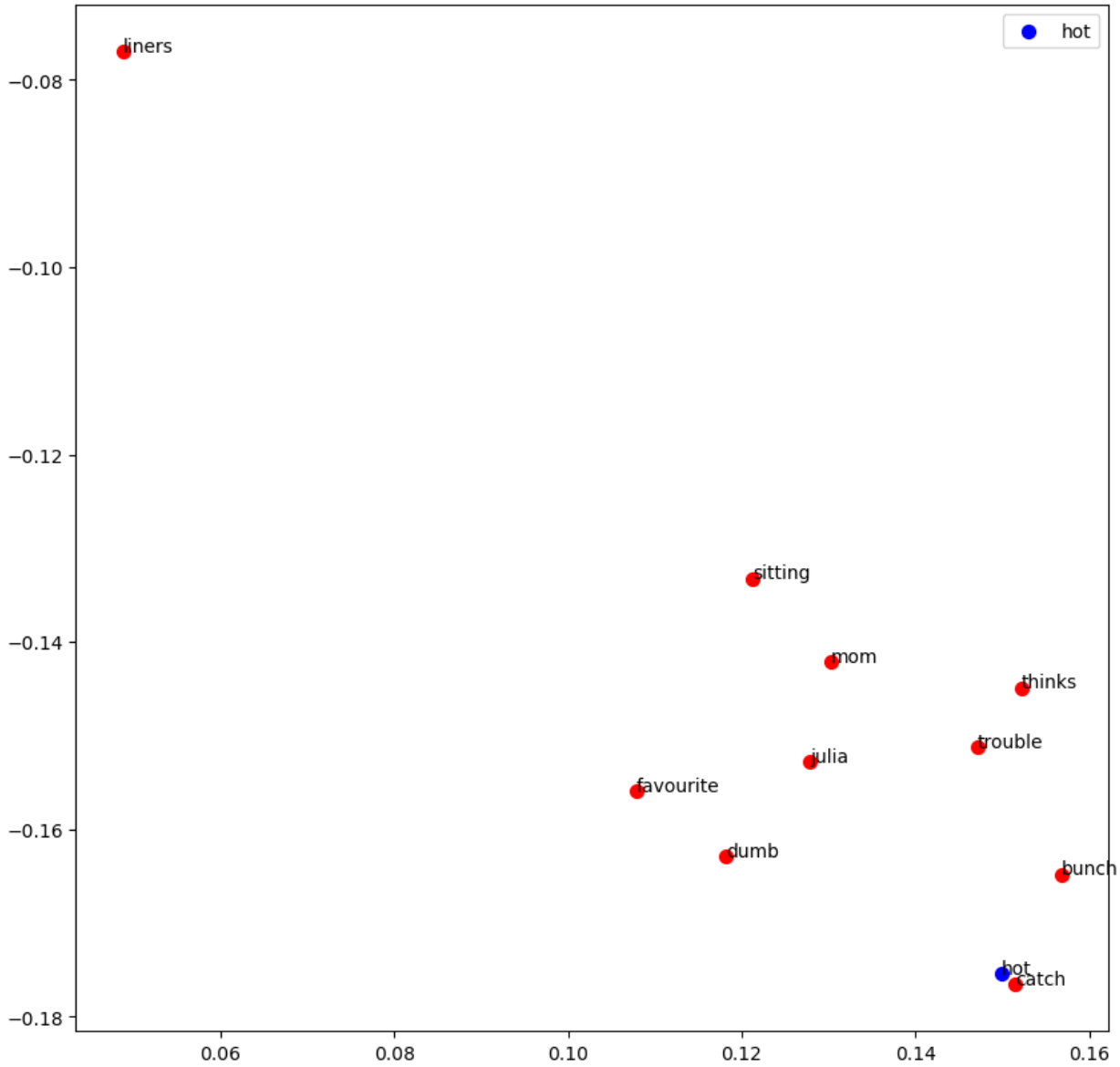
‘CHILD_noun’



	word	similarity
0	son	0.4725493398
1	friend	0.4524393767
2	kid	0.3772909226
3	school	0.3738606973
4	bluster	0.3684980568
5	happy	0.3625563691
6	thatcher	0.3563424597
7	agreeable	0.35377703

8	hours	0.3462536021
9	customers	0.3417468175

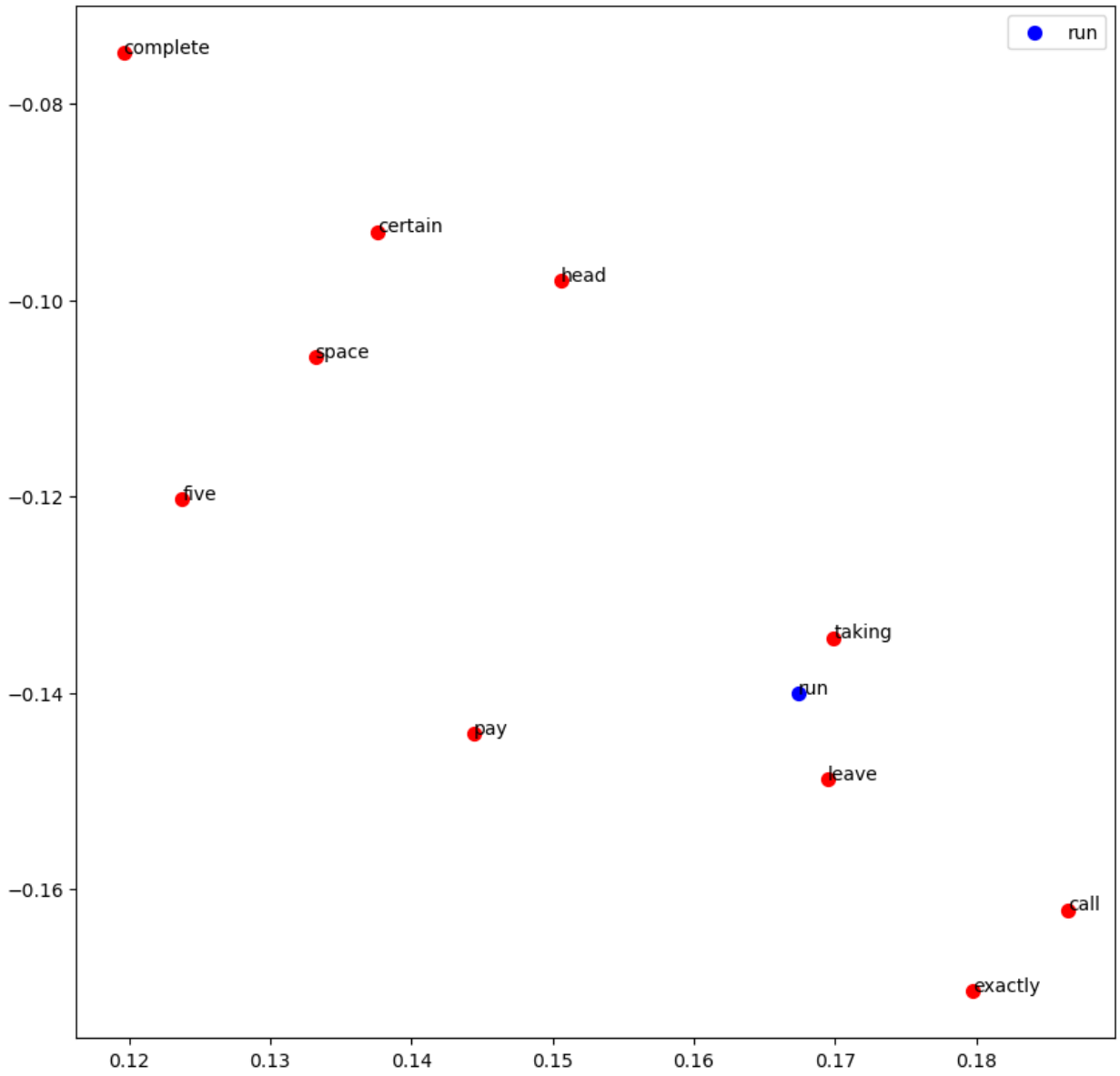
HOT_adjective:



	word	similarity
0	dumb	0.6484736611
1	liners	0.6416018578
2	catch	0.6277194125

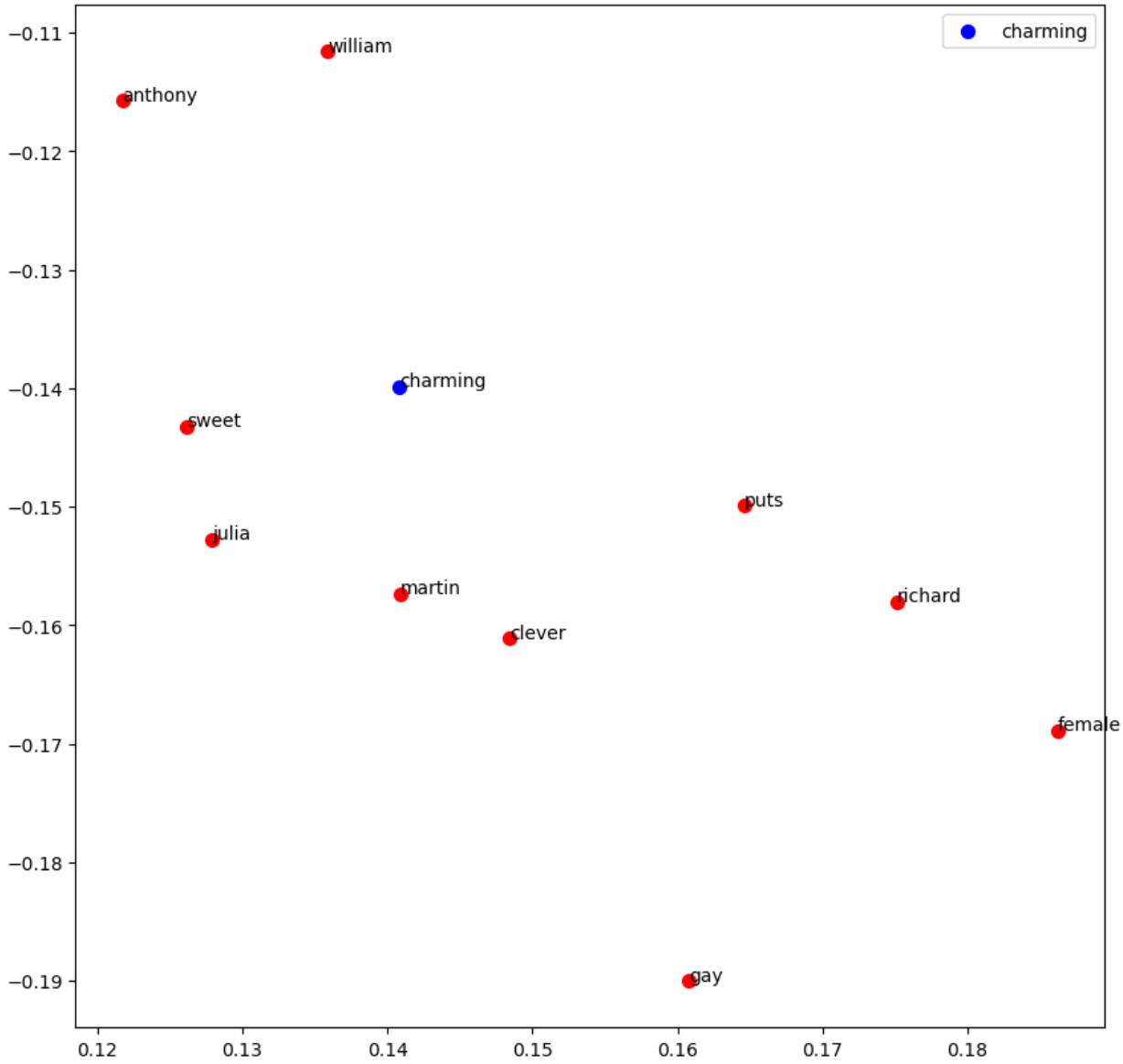
3	julia	0.6014143208
4	thinks	0.5761366604
5	trouble	0.5544796642
6	sitting	0.5538290067
7	mom	0.5508607096
8	bunch	0.5305404358
9	favourite	0.5301985505

RUN_verb



	word	similarity
0	leave	0.6181560859
1	exactly	0.5915992793
2	certain	0.568882395
3	complete	0.5622476942
4	five	0.5611725979
5	call	0.5408351468
6	space	0.5389340782
7	taking	0.5242762411
8	pay	0.5137389512
9	head	0.5105223626

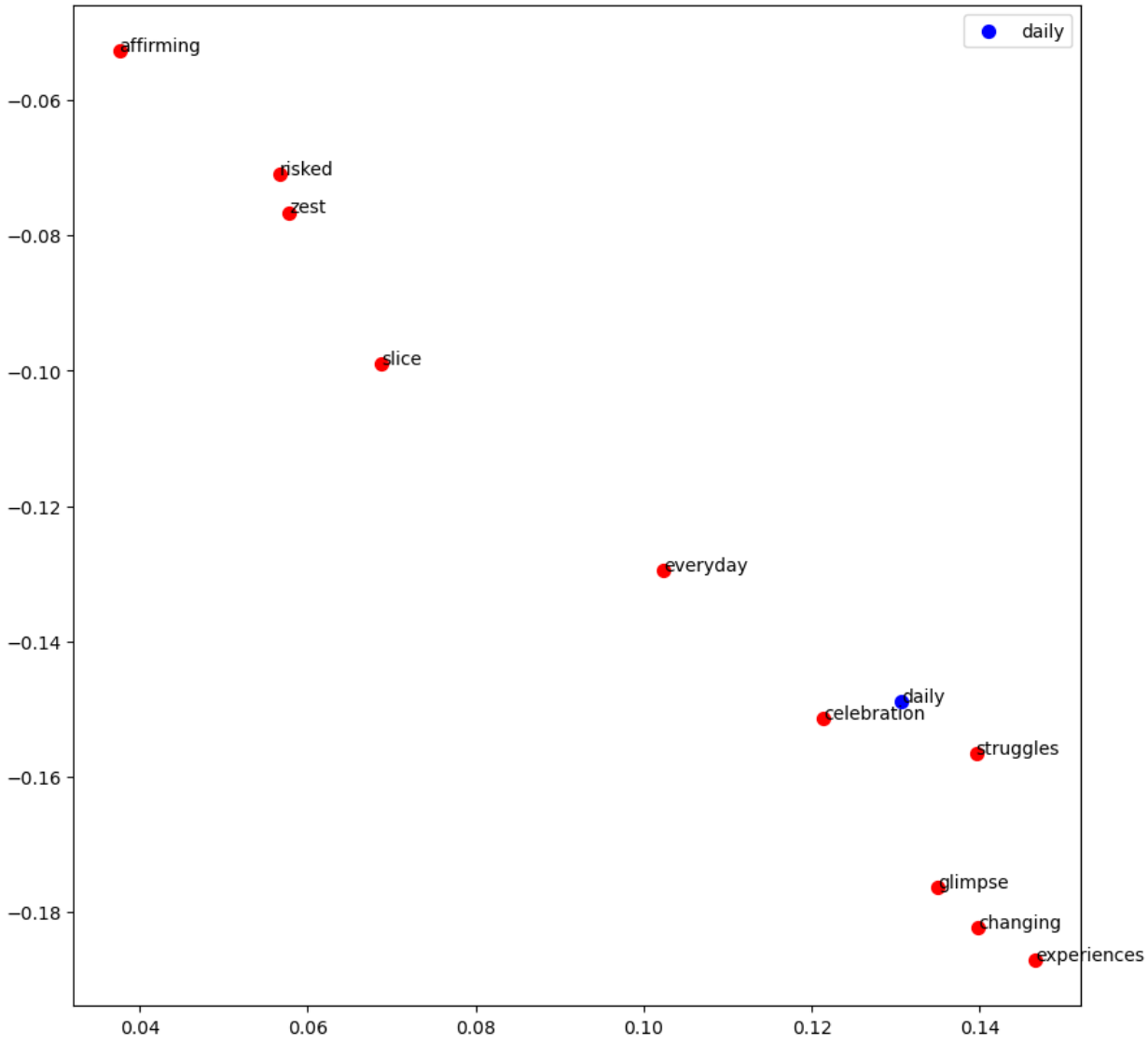
charming_adj:



	word	similarity
0	julia	0.6438550261
1	clever	0.6278842784
2	richard	0.6244180199
3	sweet	0.6221558965
4	female	0.6131739519
5	william	0.6072760962
6	anthony	0.6047867794

7	gay	0.6016315922
8	puts	0.5913923023
9	martin	0.5730436281

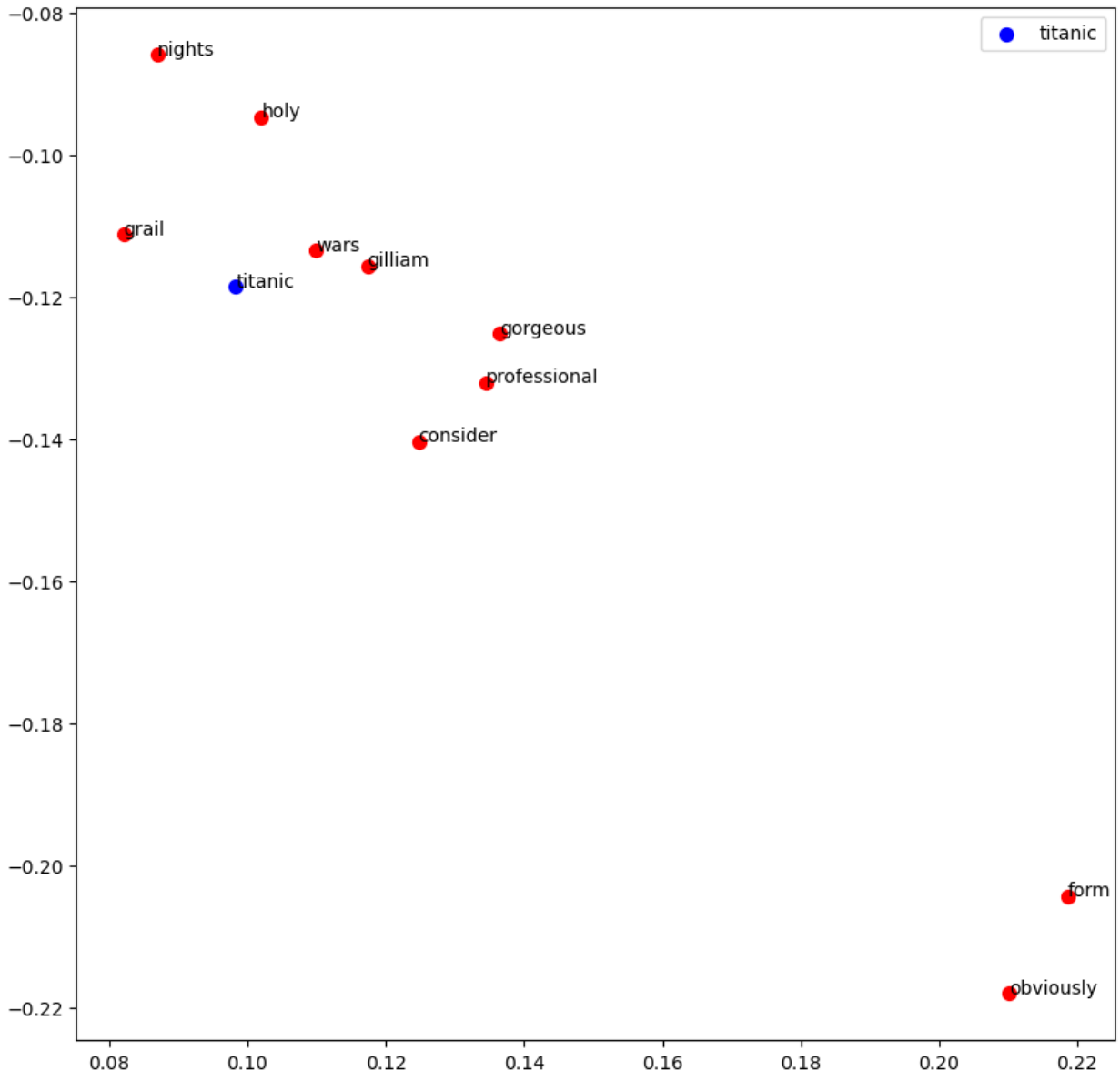
Daily_adv:



	word	similarity
0	changing	0.6841401749
1	experiences	0.6836525407

2	struggles	0.6719649905
3	celebration	0.6660904607
4	affirming	0.6491539448
5	zest	0.647268257
6	glimpse	0.641158779
7	slice	0.6352428244
8	risked	0.6321361196
9	everyday	0.6276777919

Titanic:



	word	similarity
0	wars	0.5706684245
1	professional	0.5493369026
2	gorgeous	0.5321150835
3	obviously	0.503804223
4	form	0.4953277189
5	nights	0.4944103542
6	grail	0.4935490195

7	holy	0.4770198186
8	gilliam	0.4763756554
9	consider	0.4763724756

I chose the gensim's glove vectors trained on the `glove-wiki-gigaword-50` corpus. The top 10 closest words for the gensim model were :

'odyssey', 0.65303
'phantom', 0.65100
'doomed', 0.64136
'r.m.s.', 0.63026
'cinderella', 0.62629
'voyager', 0.62269
'wreck', 0.60453
'ghost', 0.59909
'horror', 0.59604
'tragedy', 0.59548