# A Language Resonance Model Using Word Embeddings

Ryan Lattanzi

Data Scientist, General Dynamics Information Technology

March 10, 2020

**Abstract**

This paper introduces a language model that quantifies how much an entity or entities resonates with a baseline corpus. Our model aims to eliminate the subjectivity of human input, whose notion of resonance are often biased, and leave it to the computer to uncover patterns of interaction. The idea builds upon those of previous works that use word frequencies as a measure of resonance. However, we saw a big gap in this model in that the *meaning* is not taken into account. With the state-of-the-art word embeddings of Word2Vec, we can capture how words are being used in corpora, which essentially captures word meaning. After training these embeddings over the corpora of interest, we compute a resonance score based on the distance between the embeddings of words found in the intersection of the corpora. Our experiment includes testing this model over corpora with varying topics, and indeed shows that the model yields higher resonance scores for those with similar topics, and lower scores for differing topics.

## 1 Introduction

We are interested in constructing a quantitative measure of resonance between entities to see if they are speaking about the same things in the same way. The boundless boom of unstructured natural language data makes such a model extremely feasible and easy to use after curating the appropriate dataset.

Right now, sentiment analysis is a current practice for analyzing the overall tone in regards to a topic: is this particular comment 'for' or 'against' the topic? Or does this target group hold similar views to a baseline group? How would we use sentiment analysis to answer the latter? Perhaps we would tediously label every comment of the target group as 'similar' or 'not similar' towards the baseline, but this seems to leave too much subjectivity and hence human error in the process. Because the model is only as good as the data, we would like to alleviate as much human error as possible so as to produce more reliable and consistent results. This is what the proposed model intends to do.

A high-level one sentence summary of the process is as follows: take a baseline corpus of text and compare it to a target corpus by analyzing the frequency, meaning, and intersection of (key)words to output a real valued score that indicates how well they 'match up'. The idea was inspired by [1], but we are expanding upon it by introducing word embeddings that capture much more granular information about the words.

The rest of this paper is organized as such: In section 2, we lightly go over [1] for some background and discuss some of the reasons we think our method will yield better results. Section 3 gives an overview of word embeddings only to what is necessary for understanding the model. Section 4 describes the proposed model in detail. Finally, section 5 and 6 discuss the experiment setup and results using real text data.

# 2 Previous Work

## 2.1 Summary of Methods

The setup of the resonance model in [1] requires multiple corpora: a corpus consisting of general language ($\mathcal{G}$), a baseline corpus ($\mathcal{B}$), and $n \geq 2$ target corpora ($\mathcal{T}_i$, $i = 1, ..., n$) of which to compare to the baseline. Constructing the model consists of three main parts:

1. Extract keywords from $\mathcal{B}$ by comparing it to $\mathcal{G}$ to obtain a set of keywords $\mathcal{K}$, where each element maps to a corresponding 'keyness score' $L$.
2. Restrict $\mathcal{K}$ to $\mathcal{K}_{new} \subseteq \mathcal{K}$ to get rid of unimportant words defined by some statistical cutoff rule.
3. Use $\mathcal{K}_{new}$ to calculate a resonance score between $\mathcal{B}$ and $\mathcal{T}_i$.

The first step was carried out via the following equation, where $L_j$ represents the keyness score of the $j^{th}$ word appearing in $\mathcal{B}$:

$$L_j = 2 \cdot \left( f_b \cdot ln\left( \frac{f_b}{N_b \cdot (f_b + f_g) \cdot N_t^{-1}} \right) + f_g \cdot ln\left( \frac{f_g}{N_g \cdot (f_b + f_g) \cdot N_t^{-1}} \right) \right), \tag{1}$$

where $f_b$ represents the frequency of which the $j^{th}$ word appears in $\mathcal{B}$, $f_g$ is the frequency of the word in $\mathcal{G}$, $N_b$ is the total number of words (not distinct words) in $\mathcal{B}$, $N_g$ is the total number of words (not distinct) in $\mathcal{G}$, and finally $N_t = N_b + N_g$. We note that (1) was derived from the likelihood ratio for binomial and multinomial distributions.

For the second step to define $\mathcal{K}_{new}$, the authors established cutoffs based on $p$-values that rely on the scores and frequencies of the words to distinguish $\mathcal{B}$ from $\mathcal{G}$ with a confidence of 99.9%. It is necessary here to define $\mathcal{K}_{i,new} \subseteq \mathcal{K}_{new}$ as the keywords found in $\mathcal{T}_i$. In addition, we define

$$c = |\mathcal{K}_{new}|$$
$$c_i = |\mathcal{K}_{i,new}|.$$

Now, we proceed to the last step to calculate $R_i$, which we denote as the resonance score between our baseline and the $i^{th}$ target corpus:

$$R_i = \frac{\sum_{j=1}^{c_i} f_j L_j}{N_i \sum_{k=1}^{c} \left(f_k L_k\right) N^{-1}}, \tag{2}$$

where $f_j$ is the frequency that the $j^{th}$ keyword in $\mathcal{K}_{i,new}$ appears, with corresponding 'keyness score' $L_j$, $N_i$ is the number of words in $\mathcal{T}_i$, $f_k$ is the frequency that the $k^{th}$ keyword in $\mathcal{K}_{new}$ appears over all $\mathcal{T}_\rangle$, with corresponding score $L_k$, and finally $N$ is the total number of words over all $\mathcal{T}_i$.

## 2.2 Pitfalls

Although the method presented in [1] is a great approach, we felt there were a few issues which we will discuss here. Some of these came about from attempting to implement this model but we got stuck, and others came from taking a deep dive into equations (1) and (2) to see what kind of information we felt is being neglected.

The first issue is simple but huge at the same time: how do we obtain $\mathcal{G}$? This gives rise to a list of questions such as, what defines a general corpus? How do we know our set captures general language? How big does it have to be? Should it be domain specific or should it capture a broad range of domains? For instance, if we were to use this resonance model for tweets, do we go out and capture a random sample of thousands or hundreds of thousands of tweets? How many is enough? How can we be sure we have captured 'general language'? The keywords identified

are completely dependent upon our choice of $\mathcal{G}$, so we must be sure it is chosen appropriately. However, we simply don't know how to do so. Any ideas from the reader are welcome.

Although this model is able to identify keywords via statistically significant frequencies, we feel that this leaves out a lot of information about how the word is used. New state-of-the-art technology in natural language processing have overshadowed frequency-based methods (such as TF-IDF values) to capture information about the meaning of the word. This quantification of meaning is captured via word embeddings, of which we will get a brief overview in section 3. Implementing word embeddings into our model along with an appropriate distance metric will allow us to keep track not only of how often the word appears, but if it is being used in a similar context. We get a more granular picture of resonance. As a simple example to conclude this issue, consider how the word 'pants' is interpreted in American compared to the UK. In the former, it refers to outerwear, whereas the latter refers to underwear. If 'pants' were a keyword, we would like to see this dissonance present in our word embeddings.

Upon careful examination of (2), we observed that resonance scores, $R_i$, are being computed in a relative manner dependent on the sample of target corpora. That is, suppose $i = 10$, so we are comparing $\mathcal{B}$ with $\mathcal{T}_i$, with $i \in \{1, ...10\}$ (we have ten target corpora). Then, the resonance score of $\mathcal{T}_j$ is computed in comparison to $\bigcup_{i=1}^{10} \mathcal{T}_i$. Hence, we need at least two target corpora to be able to compare them in terms of who has a higher resonance than the other. Otherwise, attempting to compute a resonance score for just one target corpus would result in $R_1 = 1$, yielding the fruitless conclusion that the target corpus resonates perfectly with itself. This is great for coming to a conclusion as to which targets are resonating more than another, but it restricts us to this question. Is there a way we can consider one target and establish an objective answer to the question, 'how well is the baseline resonating with a particular target?' This can be useful if the user is interested in the resonance of the target over time. We claim our proposed model can answer both questions presented in this paragraph.

We have constructed the new model with the above considerations in mind. Next, we embark on a little journey of word embeddings.

# 3 Word Embeddings

The concept of quantifying the meaning of a word seems like magic. Perhaps when we consider representing a word by a 256 dimensional vector of small floating point numbers, it becomes a tad more mind boggling. However, this is what word embeddings do, and to such a degree that basic vector addition and subtraction yield other word vectors that actually make sense! For example, subtracting the word representation for 'man' from the word representation for 'king', and adding the word representation for 'woman' will yield a vector close to the representation for 'queen'.

How do we obtain such embeddings? There are a couple of ways and they are simply elegant. The first thing to understand is how to build the dataset. For simplicity, we will hone in on one example sentence:

'The cat jumped over the lazy brown dog.'

Arbitrarily supposing we want to examine a window size of two around a target word, we will slide a window encapsulating five words across the sentence to produce our dataset.

The first method is called the 'skip-gram' (SG), which takes one target word as input and uses a fully connected neural network to predict the two words before and after the target. The training set for the sentence above will look as in Table 1.

Once training is complete, it is not the actual predictions that we are after, but the **weights** learned from training. Each word should be inputted as a one-hot vector, and hence extracting its corresponding word vector is as simple as computing a dot product between the input vector and the weight matrix.

The second method is called the 'continuous bag of words' (CBOW). This is the flip of the 'skip gram' in that we take as input the surrounding words, and try to predict the target word. The training set for this approach is shown in Table 2.

Table 1: An example of SG with the example sentence mentioned above

| **Input** $w_t$ | **Output** $w_{t-2}$ | **Output** $w_{t-1}$ | **Output** $w_{t+1}$ | **Output** $w_{t+2}$ |
|---|---|---|---|---|
| The | – | – | cat | jumped |
| cat | – | The | jumped | over |
| jumped | The | cat | over | the |
| over | cat | jumped | the | lazy |
| the | jumped | over | lazy | brown |
| lazy | over | the | brown | dog |
| brown | the | lazy | dog | - |
| dog | lazy | brown | - | - |

Table 2: CBOW with the example sentence mentioned above

| **Input** $w_{t-2}$ | **Input** $w_{t-1}$ | **Input** $w_{t+1}$ | **Input** $w_{t+2}$ | **Output** $w_t$ |
|---|---|---|---|---|
| - | - | cat | jumped | The |
| - | The | jumped | over | cat |
| The | cat | over | the | jumped |
| cat | jumped | the | lazy | over |
| jumped | over | lazy | brown | the |
| over | the | brown | dog | lazy |
| the | lazy | dog | - | brown |
| lazy | brown | - | - | dog |

Similar to SG, we are not necessarily interested in the prediction results as we typically are with training neural nets, but the artifacts of the model that happen to be the word embeddings. Is there a better method to use? As with most machine learning problems, it depends on your situation. [2] makes note that SG 'works well with small corpora and rare terms,' whereas CBOW 'shows higher accuracies for frequent words and is much faster to train.'

Another note that seems intuitive is that word embeddings are *biased* depending on what corpora are used to train them. Alluding back to the example in section 2.2, indeed if we were to train embeddings on a corpus from the UK in which the word 'pants' showed up, we would expect the embedding of 'underwear' trained on an American corpus to be more similar to UK 'pants' than American 'pants'. We will use this to our advantage since we would like our embeddings to be domain specific to $\mathcal{B}$ as our baseline measurement.

A final note on the preference of word embeddings, especially for short documents or statements, is that they consider a small neighborhood around a target word as we saw in SG and CBOW. This allows for our embeddings to provide a more granular and precise meaning to a word - not simply capturing the occurrence of words within the same document as other algorithms such as Latent Semantic Analysis does.

# 4 Resonance Model

## 4.1 High-Level Idea

The new proposed model requires at very least a baseline corpus $\mathcal{B}$ and a target corpus $\mathcal{T}$ of which to measure resonance with $\mathcal{B}$. Note that we have done away with the general corpus. From here, it is as simple as training word embeddings from $\mathcal{B}$, training word embeddings from $\mathcal{T}$, and finally comparing the embeddings of words found in both $\mathcal{B}$ and $\mathcal{T}$ via some distance metric (see section 4.2). Then, the resonance score would simply be an aggregated calculation of the distances over all embeddings. If we wanted relative resonance scores between multiple targets, we simply compare the embeddings of only those words that appear in the intersection of all target corpora.

One key thing to note here is that *we must be consistent with the initial random weights set by the neural network that trains the embeddings.* That is, due to the stochastic nature of neural

nets, every time we begin training, initial weights are typically set to small, random values which in turn produce different final weights. Hence, we must set a seed for the initial weights in order to ensure an identical starting point for training on $\mathcal{B}$ and $\mathcal{T}$ to ensure our comparison is authentic. We address how to handle this in our experiment in section 5.1

Why would this work? Due to the way the embeddings are learned, we see that they capture *how words are being used in a corpus.* This, in essence, captures the *meaning* of the word. Hence, if two corpora are using words in the same way (rather than simply containing the same words), we can say that they are resonating. Now, some may see this as just another type of sentiment analysis, where two entities are sharing a similar sentiment. So be it if it is; however this would be a different kind of sentiment analysis, one that is not dependent on human coding text as 'positive' or 'negative', but rather an unsupervised *relative* sentiment analyzer between two groups that alleviates human subjectivity and lets the algorithm learn objective features.

## 4.2 Distance Metrics

We decided to experiment with a few different metrics to calculate the distance between two word embeddings. We chose the three that we have come across most often when dealing with vectors and we list them below, followed with a brief discussion of them:

- Manhattan (or $L_1$ norm of the difference): $||x - y||_1 = \sum_{i=1}^{n} |x_i - y_i|$
- Euclidean (or $L_2$ norm of the difference): $||x - y||_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
- Cosine Similarity: $cos(\theta) = \frac{\langle x, y \rangle}{||x|| \cdot ||y||}$

**Manhattan vs. Euclidean:** We expect these two to yield extremely similar results, since they both are computed element-wise. The difference is in scaling, and since we expect values of the embeddings to be small ($\ll 1$), squaring the difference in the Euclidean distance will make them even smaller, but this will be somewhat counteracted by the root. Hence, we probably will prefer the Manhattan because of its relative simplicity and interpretability.

**Cosine Similarity:** This metric is quite different compared to the other two since it takes away the notion of magnitude of the vectors by dividing by the norms. If the two vectors have vastly different scales (which we presume they will not given they are neural network weights), the cosine similarity depends solely on the *direction* from the origin of the vectors. If the vectors lie in the same direction, we get a similarity value of 1, meaning they are the same; we get 0 if they are completely unrelated, and -1 if they are exactly opposite. What does this mean? Suppose in one corpus, the word 'bad' is used in the traditional sense that something is not good. However, some slang uses 'bad' to actually mean 'good'. In this case, we would like to see a negative cosine similarity. Although in high dimensions, two random vectors have a high probability of being orthogonal, we discount that by the assumption that the final embeddings are not random, but learned.

## 4.3 Resonance Score Calculation

The metrics in the previous section serve to calculate the distance between two embeddings. To achieve an overall resonance score between corpora, we need to aggregate all of the distances between the embeddings. This is where the art comes in.

Arbitrarily, we would like the resonance score to lie in $(0, 100]$, where 0 represents no resonance, and 100 represents perfect resonance. We define $\mathbf{d} = (d_1, d_2, ..., d_k) \in \mathbb{R}^k$ to be the vector of distances computed between $k$ words that lie in the intersection of $\mathcal{B}$ and $\mathcal{T}$. More precisely,

$$d_j = ||x - y||_{(\cdot),j}$$

where $j \in \{1, 2, ..., k\}$ and $(\cdot) \in \{1, 2, \cos\}$, representing the distance metric used. We acknowledge the abuse of notation since cosine similarity is not a true metric. We have come up with two

methods of calculation depending on if the Euclidean/Manhattan distance or Cosine Similarity is chosen.

**Euclidean/Manhattan Resonance:**
In this case, $d_j \geq 0$ for all $j \in \{1, 2, ..., k\}$ and thus

$$||\mathbf{d}||_1 = \sum_{i=1}^{k} |d_i| = \sum_{i=1}^{k} d_i.$$

Recall that $||\mathbf{d}||_1$ is a measure of error; a growing norm indicates a lack of resonance. Hence, we want our resonance score to be monotonically *decreasing* as a function of $||\mathbf{d}||_1$. With this in mind, we define the resonance score to be:

$$R(\mathbf{d}; \lambda) = 100 - 100 \cdot \tanh\left(\frac{1}{\lambda}||\mathbf{d}||_1\right), \tag{3}$$

where $\lambda \geq 1$ is an appropriately chosen stretching parameter to alleviate scores being scrunched along the asymptote of $R(\mathbf{d}; \lambda)$. For a geometric intuition, we recommend to the reader a visit to Desmos [3] to play around with some values, but reference Figure 1 below for a glimpse.

We see that (3) is monotonically decreasing on $[0, \infty)$, with $R = 100$ if $||\mathbf{d}||_1 = 0$, and $R \to 0$ as $||\mathbf{d}||_1 \to \infty$. Since $||\mathbf{d}||_1 \geq 0$, the domain of $R(\mathbf{d}; \lambda)$ is restricted to $[0, \infty)$, guaranteeing our desired resonance range of $(0, 100]$.

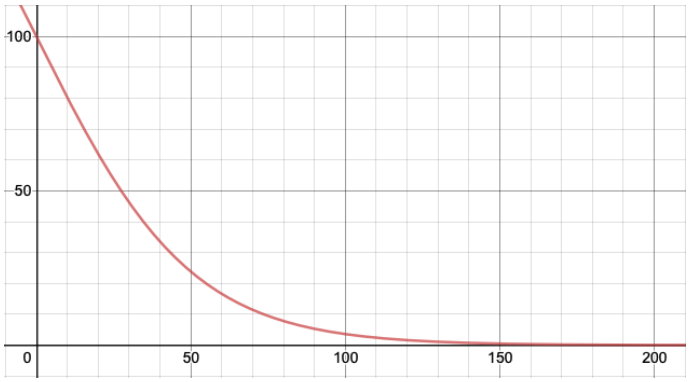


Figure 1: A snapshot of equation (3) with $\lambda = 50$.

**Cosine Similarity Resonance:**
We change our calculation of resonance to account for the possibility of $d_i \in \mathbf{d}$, $i \in \{1, ..., k\}$ being negative, as well as the interpretation of this negativity. With this distance measure, a higher value of $\sum_i d_i$ is good because that means more word embeddings are pointing in similar directions, indicating a similarity of meaning. Hence, we want our resonance score to be monotonically *increasing* as a function of $\sum_i d_i$.

Some may want to discard the negative values because their interpretation may be superfluous or confusing. Hence, we describe two possible resonance measures that include and discard the negative values, respectively. The first is defined as:

$$R(\mathbf{d}; \lambda) = 100 \cdot \sigma\left(\frac{1}{\lambda}\sum_{i=1}^{k} d_i\right), \tag{4}$$

where $\sigma(x) = \frac{e^x}{1+e^x}$ and $\lambda \geq 1$ is a chosen stretching parameter playing a similar role as in (3). Since cosine similarity can return negative values that can be interpreted as *opposing* usage of

the word, (4) must be ingested with a different air than (3). We see that $R \to 0$ when we have an abundance of negative similarities, indicating *opposing* narratives; $R = 50$ when there is no correlation (or resonance); $R = 100$ as more word embeddings have similar direction. See Figure 2 for an example.

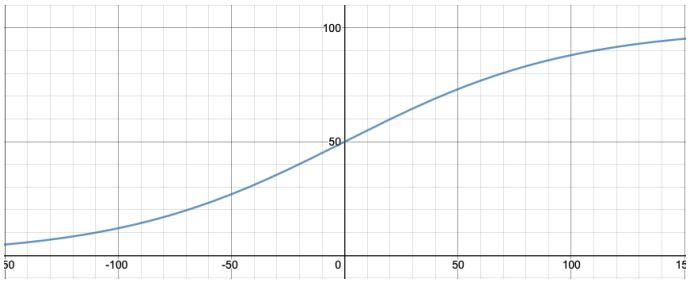


Figure 2: A snapshot of equation (4) with $\lambda = 50$.

If this interpretation does not suit the reader, we can reconstruct our calculation by altering our definition of cosine similarity. If we decide we do not want to account for *negative resonance*, we can simply change our measure from

$$d_1(x, y) = \cos(\theta) = \frac{\langle x, y \rangle}{||x|| \cdot ||y||} \tag{5}$$

as was used for equation (4), to

$$d_2(x, y) = \max\{0, \cos(\theta)\}. \tag{6}$$

Then, we use (6) as our distance measure to feed our second option of a resonance calculation:

$$R(\mathbf{d}; \lambda) = 100 \cdot \tanh\left(\frac{1}{\lambda}||\mathbf{d}||_1\right), \tag{7}$$

where we again see our stretching parameter $\lambda \geq 1$. Equation (5) rids our distance measure of computing negative values, such that we regard the notion of negative resonance as no resonance at all. Then, our domain is restricted to $[0, \infty)$ since $d_i \geq 0$ for all $i = 1, ..., k$. We see that $R = 0$ when $d_2(x, y)_j \leq 0 \ \forall \ j = 1, ..., k$ (all word embeddings have either negative or no resonance), and $R \to \infty$ as $||\mathbf{d}||_1 \to \infty$.

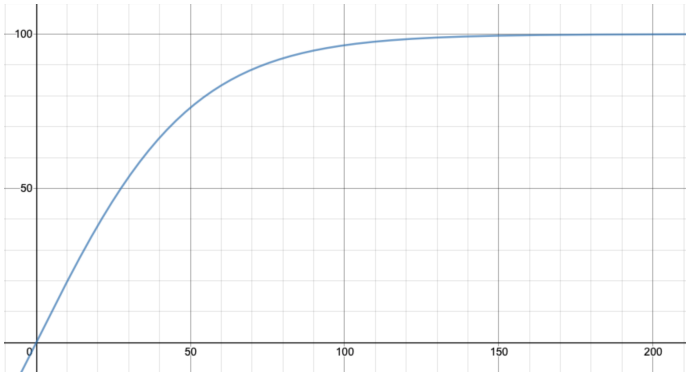


Figure 3: A snapshot of equation (7) with $\lambda = 50$.

Next, we describe the experiment used that use equations (3), (4), and (7).

# 5 Experiment

**All of the code and a more in-depth explanation of testing the model with real data can be found in the Jupyter Notebook found at [4]. Here, we give just enough detail to paint the picture.**

Recall section 2.2 in which we claimed that our model would be able to answer both of the following questions:

1. How well is a particular target resonating with a baseline?
2. Given a group of targets, which are resonating more than others?

To answer the first question, it is trivial to only consider a baseline $\mathcal{B}$ and a target $\mathcal{T}$, and compare the word embeddings of words found in both corpora. Then, we could track these scores over time to see if the target is evolving.

In this experiment, however, we aim to answer the second question by comparing embeddings of words found in the intersection of $\mathcal{B}$, $\mathcal{T}_1$, and $\mathcal{T}_2$ which we define in section 5.1.

## 5.1 Data and Setup

We pulled data from the Big Bad NLP Database [5]. We chose the AG News set, which has about 120k articles for topic classification - the topics being world, sports, business, and sci/tech. We also chose the Amazon Fine Food Reviews dataset to introduce text that we expect would be different than any of the text found in AG, in particular those articles classified as sports articles.

From AG, we segmented out the sports articles and split it with a 60/40 ratio, keeping the first as $\mathcal{B}$, and the second as $\mathcal{T}_1$. This gave us $17,154$ articles in $\mathcal{B}$ and $11,470$ in $\mathcal{T}_1$. Then, scaling down the massive Amazon Fine Food Reviews set to match the size of $\mathcal{T}_1$, we randomly extracted $11,470$ reviews and set them as $\mathcal{T}_2$. After some minor text preprocessing, we were ready to test the model.

We had to set numerous hyperparameters of the model, and did so based on intuition of our context. At a later time, it would probably prove beneficial to perform some type of search to optimize these values. Nevertheless, we set the following:

- *num_features*: Length of the word vectors; set to 128 since our corpora are not huge.
- *min_word_count*: Minimum word frequency in the corpus to be considered for a word vector; set to 3.
- *num_workers*: Number of cores to use on the computer; set to 1 to ensure same initial weights.
- *window_size*: Max distance between the current and predicted word in a sentence; set to 2 since we have relatively short sentences.
- *sample_rate*: Frequency threshold above which to apply subsampling; set it to the recommended value of $10^{-5}$.
- *skipgram*: Training algorithm for Word2Vec 0 for CBOW, 1 for SG; set to the default value of 0.
- *epochs*: Number of epochs for training the embeddings; set to 15 to ensure quality meaningful embeddings.

At this time, we bring up an important consideration mentioned in section 4.1 - we want to be sure that the initial weights of the neural nets that train the word embeddings are the same throughout the corpora. Otherwise, each corpora will have a different starting point which can lead to different local optima (in this case different embeddings). Setting $num\_workers = 1$ seems to do the trick, and we verify that training over the same corpus with two distinct models will yield the same embeddings (See [4]).

**Our hypothesis is that our proposed model will compute a higher resonance score for $\mathcal{T}_1$ than $\mathcal{T}_2$ regardless of which distance measure in section 4.2 is chosen.**

# 6 Results and Discussion

Main results are shown in Table 3. **The results fit our hypothesis.**

Table 3: Results of the experiment that show resonance scores for each distance metric with a specified value of $\lambda$. We chose these particular values to put them on a similar interpretable scale.

| **Corpus** | **R(d;** 10000) **Euclidean** | **R(d;** 100000) **Manhattan** | **R(d,**2500) **Cosine Neg** | **R(d,**2700) **Cosine Pos** |
|---|---|---|---|---|
| $\mathcal{T}_1$ | 73.58 | 76.40 | 74.51 | 75.87 |
| $\mathcal{T}_2$ | 57.82 | 62.03 | 66.39 | 55.83 |

Obviously, raw resonance scores are completely dependent on $\lambda$. But, this is discounted if we only care about resonance *ranking* between multiple target corpora - although the raw values will change, the rank will not since $\lambda$ is simply manipulating the spread of the values over monotonic functions. However, if we were interested in keeping track of raw scores over time, we must hold $\lambda$ constant to keep each measurement throughout time on the same scale; otherwise, we lose interpretation of the scores.

Finally, we address once more the practice of training using the same initial weights that the reader is probably sick of hearing about by now. As it turns out, when training on the same Jupyter Notebook Kernel and setting $num\_workers = 1$, every training instance will use the same initial weights. It is only after restarting the Kernel and re-training do we see differing values. What is extremely interesting, though, is that even after restarting the Kernel and re-training with all the same hyperparameters as before, the final resonance scores are within a small neighborhood of the previous scores. Perhaps this is an intuitive and expected finding to other more experienced neural network practitioners. We presume it is due to the initial weights not deviating *too* much, and that with enough epochs our network will learn consistently. Clearly, this calls for a simulation after which we can provide confidence bounds for the scores. We leave it that to further work.

# 7 Conclusion

This paper introduced a new language model to quantify *resonance* between a baseline, $\mathcal{B}$ and either one other target $\mathcal{T}$, or multiple targets $\mathcal{T}_i$, $i = 1, ...n$. We designed this model to address some pitfalls of previous work, one of the biggest being leaving information out of the model by only basing resonance on word frequencies between corpora. Our model uses word embeddings, which are able to capture how words are being used in each corpus, essentially capturing the *meaning* of the word. After constructing new resonance calculations based on different distance metrics between word embeddings, we put them to the test in an experiment that used real text data. The results of the experiment matched our hypothesis that the corpus relating more to the baseline would have a higher resonance score than the corpus not related. This was true across all vector distance metrics. For future work, we plan on running a simulation to gain confidence bounds of the resonance scores, and experimenting with optimization techniques for tuning the various hyperparameters.

# 8 References

[1] Marcellino, W., Cragin, K., Mendelsohn, J., Cady, A., and Magnuson, M. (2017) *Measuring the Popular Resonance of Daesh's Propaganda.* Journal of Strategic Security 10, no.1: 32-52.

[2] Lane, H., Howard, C., and Hapke, H. (2019) *Natural Language Processing In Action.* Manning Publication Co.

[3] desmos.com

[4] https://github.com/ryanlattanzi/resonance_model

[5] https://datasets.quantumstat.com