Orcun Tasdemir

Skimming Model by word2vec Word Vectors of Google News and GloVE Statistical word vectors of Stanford

The modelling project I took on gave me a lot of freedom in terms of what kind of an approach I could take. Overall, I tried to approximate how humans might be skimming text. I got similar results to how the eye tracking experiment from the paper ***Skim reading by Satisficing: Evidence from Eye Tracking*, 2011** predicted the decline of the reading rate as the paragraph advanced.

The concept of optimal foraging is explained in my introduction, to reiterate the words, my research question is whether skimming is a behaviour that is governed by the general rules of optimal foraging, that is, whether it works by reading the text until the expected information gain for the next sentence falls below a certain expectation that triggers a jump over some number of sentences that were not read as they were deemed discardable and the jump optimal for maximizing the information gain in the limited time the agent has to read through the entire text.

My approach was to focus on 2 different aspects of the reading process:

- The overall interest you have for the next sentence as a function of the word vector average of the sentences you have been reading.
- The frequency occurrences of frequent words throughout the text that might encourage the reader to read more.

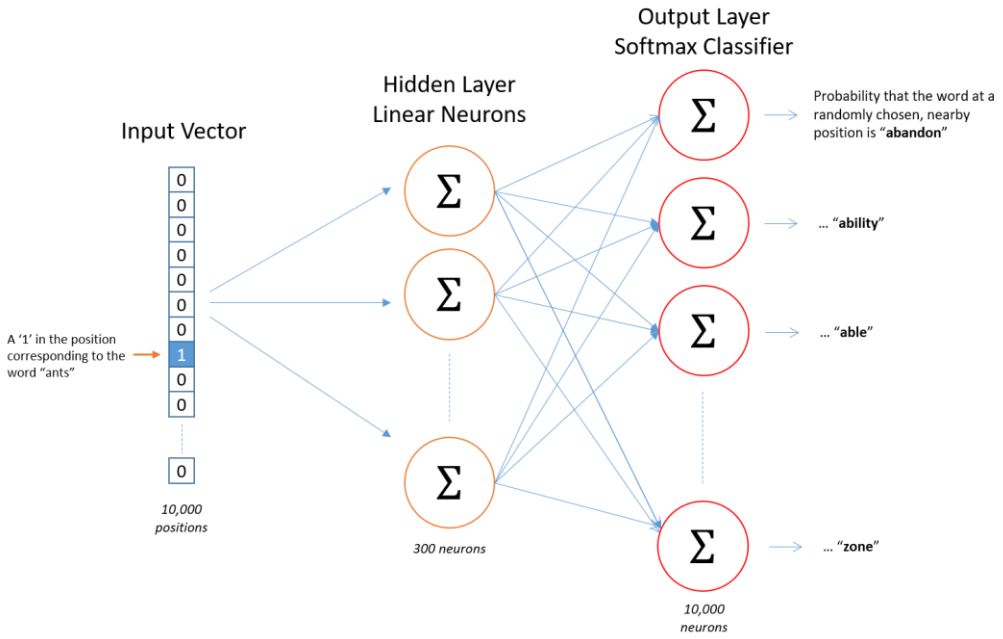
I felt the need to introduce another level of complexity to my model as otherwise it would not be able to capture the full extent of the behaviour I am trying to model: the concepts of long and short term interests and important words. Below I am going to start from scratch and explain the decisions I have made throughout the evolution of the model and the concepts I have used/tackled. First, a quick review of the nature of the word vectors I am using and how they work/ are trained.

Google's Word2Vec word vectors

Word2vec refers to a group of models that are used to create word embeddings. In this project, instead of training my own word vectors, I am using the word vectors I got from google's

pretrained embeddings[1]. The version I am using is the biggest word vector collection google has which was trained on the Google News dataset which is worth about 100 billion words. It contains 300-dimensional vectors both for words and phrases[2]. The embeddings are obtained by training a shallow two layer neural network by a method called skip-gram. Skip-gram is useful for predicting the context of a target word.

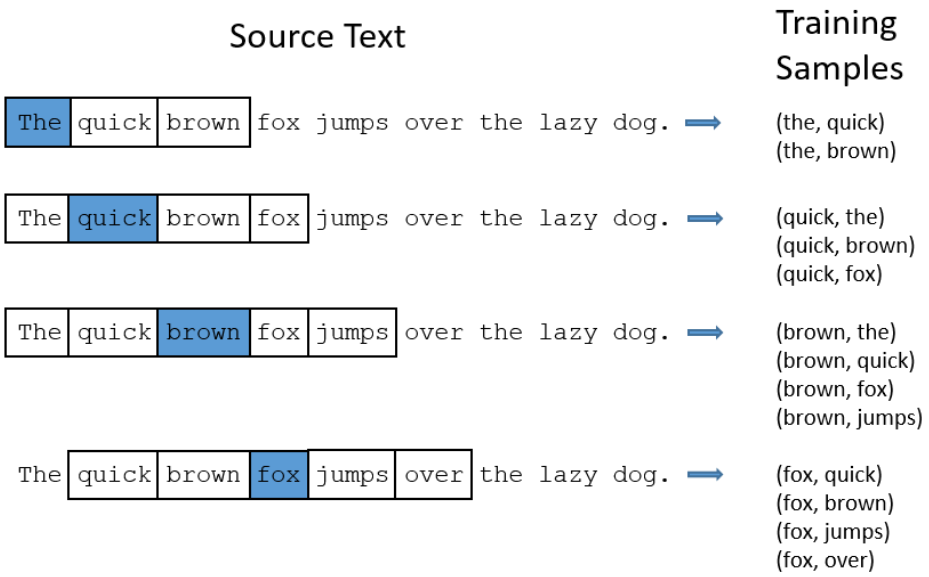
It sets up a fake task for the neural network to learn which in turn will indirectly give us the word vectors we want.[3] Basically, it trains the network by giving it a target word with a set number of words around it (window size). The network learns to predict the probability of 'context words' - words than can be expected to appear inside the window around our target word - given the target word.



1 GoogleNews-vectors-negative300.bin.gz.

2 Phrases such as "New_york", as it means something completely different than the combination of 'new' and 'York' we need to account for phrases like this one.

3 Chris McCormick: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/



For example, after the model is trained with the first sentence, it predicts that given the target word 'the', the word that can appear around it is either 'quick' or 'brown', and sets the probabilities as .5 for those two words. Training on as many as 100 billion words. The probabilities evolve to capture the contextual relationship between words. However, what we are interested in is not the model itself but the hidden layer. Because after the training, the hidden layer now has the ability to generate unique vectors for each target word we input, and since it was trained to find context words, vectors it generates for two words that are closely related in terms of context will be closer to each other than the words that have completely different contexts, and this is how we get the google word vectors.

GloVE vectors from Stanford[4]

GloVE, standing for global vectors, is an unsupervised learning algorithm. Instead of using a predictive approach, Stanford NLP used a count-based approach to get distributed embeddings for words. They first build a large co-occurrence information matrix for each word. It holds the information on the frequency of each word in presence (in context) of other words. They then factorize this matrix so that it yields a lower-dimensional matrix of word embeddings. They use a concept called "reconstruction loss" and try to minimize it for the lower dimensional representations that can still explain the complexity of the high dimensional data. There is more in depth information about how it achieves the word vectors via dimensional reduction which I

4 Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip

cannot get in here. However, basically, it looks for meaning in the statistics of the placement of words with regards to one another.[5] Now we can go back to my model and how it works.

Average Sentence Vectors

This was the biggest choice that I had to make very early into the modelling as to whether use a word based or a sentence based approach. Looking at the research done on skimming, I learned that even though "the majority of studies on semantic processing using language have generally focused on single words" since the level of sentences were "such complex stimuli"[6], the concept of sentence understanding was more relevant to my model as single words would not be enough to communicate what the entire sentence would be saying. An example of this could be the sentence "John was told that her mother died, but he didn't know that he was told a lie." Reading this sentence word by word, being completely blinded to the next word you did not get to read yet, gives a completely different meaning to the sentence until the very last word 'lie'. Even though my average word vectors are nowhere near to actually understanding a sentence like this in its true semantic meaning, reading the word 'lie' would nevertheless helpful as the reader went on the rest of the sentences. Therefore, I concluded that using sentences as my smallest unit of semantics would be my best bet at recreating the skimming behavior.

The comparison method I used between sentences relied on getting a reliable average vector for both sentences that were 'representative' of the sentences. To enhance the representativeness of an average vector for a sentence I needed to asses the weights with which the individual words contributed to the average meaning of the sentence, depending both on the overall frequency of their usage in the written english language and the part of speech tag each word has.

I ended up using the written & spoken frequency data with POS(part of speech) tags from the American National Corpus's database.[7] I parsed the database to a matrix and wrote functions to access a word's frequency of use and POS tag given the word.

---

5 https://medium.com/sciforce/word-vectors-in-natural-language-processing-global-vectors-glove-51339db89639

6 Tune S., Schlesewsky M., Nagels A., Small S. L., Bornkessel-Schlesewsky I. (2016). Sentence understanding depends on contextual use of semantic and real world knowledge. Neuroimage 136, 10–25. 10.1016/j.neuroimage.2016.05.020

7 http://www.anc.org/data/anc-second-release/frequency-data/

While calculating the average vector for a sentence, I used the frequency information of each word to create a coefficient that I could multiply the related word vector with. I set up a rating function that calculates a coefficient for the word based on its rank divided by the number of words in the frequency list, this way, if a word was less frequent, it would appear with a bigger rating which means that as words get more unique, I will be assigning a greater weight to its vector.

```
def get_frequency_rating(word):
    if is_word_in_matrix(word, pos_freq_matrix):
            for i in range(len(pos_freq_matrix)):
                if word == pos_freq_matrix[i][0]:
                    rating = i / len(pos_freq_matrix)
    return rating
```

pos_freq_matrix - List (98197 elements)

Inde	Type	Size	Value
0	list	3	['like', 'IN', '54490']
1	list	3	['yeah', 'NN', '52301']
2	list	3	['would', 'MD', '51778']
3	list	3	['know', 'VBP', '46581']
4	list	3	['said', 'VBD', '42294']
5	list	3	['well', 'RB', '40404']
6	list	3	['also', 'RB', '32854']
7	list	3	['people', 'NNS', '32524']
8	list	3	['time', 'NN', '32253']

Save and Close Close

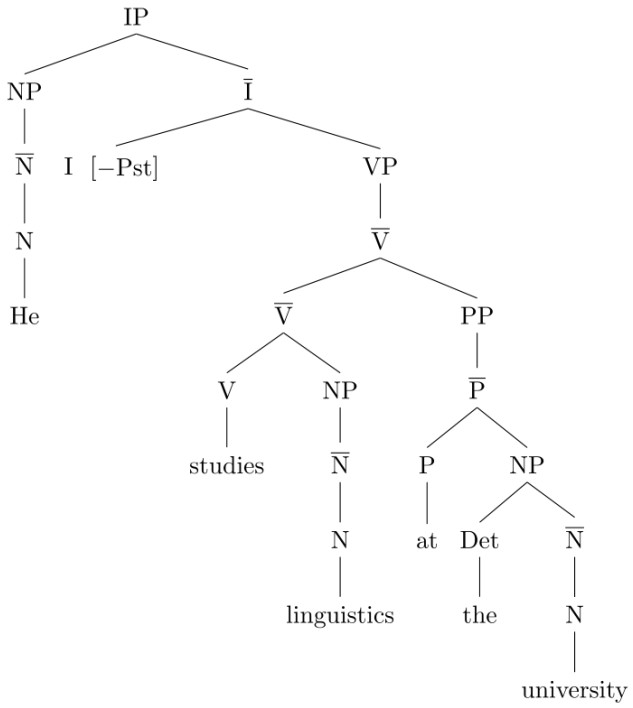
About the code snippet above: In the code I submitted the ratings 0.5 and 0.01 are switched, I was experimenting with it and found no significant differences probably because both the rates are so low anyways, but I forgot to change it back. Here is how it would look like.

My understanding of this strategy was to get rid of words that are so common that they do not contribute to the meaning of the sentence anyways such as ‘like’, ‘yeah’, ‘would’, ‘know’, ‘said’, which are the first 5 most common words according to the american national corpus. At first glance, this might seem like a bad strategy as the rating parameter for the word ‘like’ seems like it is going to end up with a rating of 0 as 0 divided by the length of the word list is 0. However, like ends up with a frequency rating of 0.65. Why? The reason is that my function takes the last entry associated with the word,which in this case is the entry number 63868. It is the word like again but it is a different ‘like’, in that it is not a preposition but a noun. Considering the last entry of the word is important and works in our favor as we have no way of knowing which POS tag that ‘like’ we read off a sentence had within the model, it is better and safer to assume that it is the least common version of the word and pay attention to it accordingly. Besides, I have one more rating system that I use as a coefficient to alter the word vectors that takes care of the greater-than-expected frequency rating for words such as ‘like’.

The second rating function I use takes into consideration the POS tags of the words and assigns them set coefficients depending on the tags.

```python
def get_pos_rating(pos):
    rating = 1
    if pos[0:2] == 'NN':
        rating = 4
    elif pos[0:2] == 'VB':
        rating = 3
    elif pos in ('RB', 'RBR', 'RBS', 'PRP'):
        rating = 0.5
    elif pos in ('CC', 'CD', 'DT', 'EX', 'FW', 'IN', 'JJ', 'JJR', 'JJS', 'LS', 'MD'):
        rating = 0.01
    elif pos in ('PDT', 'POS', 'PRP', 'RP', 'SYM', 'TO', 'UH', 'WDT', 'WP', 'WRB'):
        rating = 0
    return rating
```

The meanings of all the part of speech tags here can be found on the Penn Treebank Project website.[8] My decision to give much higher ratings to words that are nouns and verbs should be rightfully expected as those two categorizes is what makes up the semantic skeleton of a sentence as in linguistics, one of the most common ways of constructing sentence diagrams is through dividing the sentence into its noun phrase (NP) and verb phrases (VB):



I chose to give nouns a slightly higher rating the reason being the fact that sentences are about nouns. Even though verbs are nearly as important as nouns, I think that the aspect of the sentence that continues is mostly the subjects and the objects instead of the actions because actions can be changed but the nouns will be relatively more grounded and stable throughout a text as they actually denote physical

8 https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

objects rather than abstract actions. Though exchanging the ratings between the nouns and the verb did not result in any significant difference either.

I decided to give the personal pronouns and various types of adjectives a rating of 0.5 as I thought that they might still be relevant to the grand scheme of the semantic flow of the text. I gave coordinating conjunctions (CC), cardinal numbers (CD) and the following tags the rating 0.01 which also could just have been zero and the other tags got a rating of 0, some of them are predeterminers, symbols, the word 'to' (it has its own POS tag as 'TO'), particles, and interjections.

In conclusion, these are the two rating systems I use to correct the word vectors for their importance for the sentence. After the word vector are corrected, I take the mean of the word vectors and call that the average sentence vector for the sentence. The similarity between two sentences is calculated by taking the cosine similarity between the two sentence vectors.

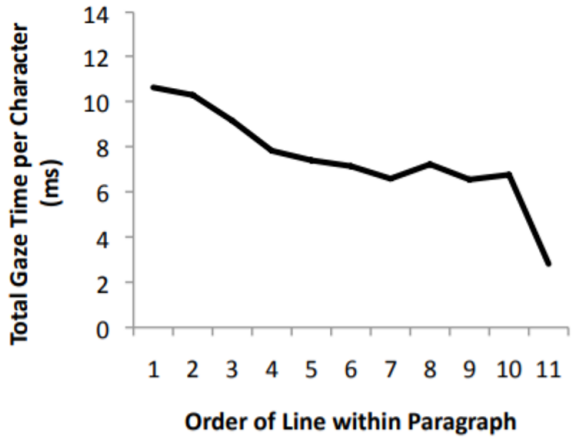
TFI - DF Weightings

Tfi-df stands for "term-frequency - inverse document frequency", it is a statistical metric that try to reflect the importance rates of words in a document, or in a collection of documents. It essentially increases the importance value of a word proportionally to the number of times the word appears in the given document while offsetting the importance value proportionally by the number of documents that contain the word. The reason is to correct for words that are common out of necessity of grammar and syntax. We want to weed out the words that are really important instead of just occurring frequently. They need to occur frequently but in a concentrated area which should show that they are not just overall common but actually specifically important for the context of the document. It is one of the most used term-weighting tools today and text-recommender algorithms also use it.

The way I implement the tfi-df weights is to run the algorithm on the sentences that the model read so far to give me the most important words seen so far. In this context, the sentences are the documents so the inverse document frequency applies in between sentences. I use the calculated most important words to compare it to a threshold I set to decide whether I want to keep reading or not. For example, if the last sentence I read had more important words than the threshold I want to model to keep reading because the next sentence is most likely relevant to the overall theme/plot of the text, which is the reason I am reading the text in the first place, to learn about whatever the text I am reading is about.

Paragraph Correction Values

Another feature I am using is actually inspired by the eye tracking experiment from the paper ***Skim reading by Satisficing: Evidence from Eye Tracking*, 2011.** The experiment they conducted showed that people read the first sentences of the paragraphs more than they read the second one and so on, the trend is a linear decrease with some bumps around sentences 7-8 and a sharp decrease around the sentences 10 -11, however this is highly reliant on average length of the paragraphs in the texts that they used in the skimming experiment.



I have a function that takes in the number of sentences from the beginning of the paragraph and calculates a coefficient that I will later use to alter the interest thresholds and the thresholds for the number of important words both of which guide the decision to skip over sentences or keep reading.

Text Processing

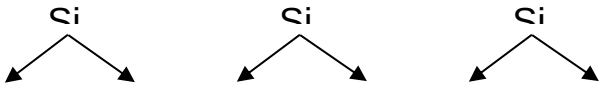
Before going into the model I also want to talk about the text preprocessing I needed to do before I could split the text to sentences for the reading process. I took into consideration the contractions in the language such as ‘can’t’ or ‘he’ll’ since my word vectors did not have the contracted versions in their vocabulary so I converted all the contractions back into their full length form. I also got rid of the punctuations that are not sentence ending such as ‘,’, ‘:’, ‘;’ as they had their own vectors in my word vector library and I did not want them to get involved in the average vector representation for the sentence. I also set up an algorithm that will recognize whether we are in a quotation and will withhold the splitting of the sentence even though there is a period inside the quotation because I did not want to take out the quotations out of the sentences they were used in as separate sentences. I got rid of all the stop words. Those are words that are commonly excluded from any text processing as they just cause an increase in the workload for the algorithms, and are very useless in terms of assessing semantic meaning in a sentence, words such as ‘the’, ‘a’, ‘your’, ‘in’ etc.

The Model

The model starts by setting some global variables that will be used for the decision making. As I talked about how I incorporated short and long term interest rates and important word threshold, we start by defining those:

```
model(text_file,
google_vectors,
pos_freq = pos_freq_matrix,
set_long_term_interest = 0.69,
set_short_term_interest = 0.79,
set_long_term_importance = 5,
set_short_term_importance = 3,
set_long_term_imp_limit = 3,
set_short_term_imp_limit = 1,
how_many_sentences_to_read = len(read_these_sentences))
```

Long term interest refers to the average sentence similarity threshold we set for the model. It is compared with a variable called long term mean similarity which is the mean similarity for the last 3 similarity entries which would correspond to the mean of the similarities between the last 4 sentences read.



Fourth_to_last Thrid_to_last Second_to_last Last sentence

If the mean similarity is below the threshold, this means that the text is still interesting, and fruitful to read. Another chance to keep reading occurs if the mean similarity is over the ceiling threshold which was set at 0.8. The reasoning is that if the two sentences were very close in meaning, it might be a good place to stick reading because there might be valuable information as the topic and the rate of information gan around the topic should be very concentrated above that level of similarity. However, there would still be a random choice with a list of probabilities.

read = 0.5 + pow((long_term_interest - long_term_mean_sim), 2)

“Read” would be the probability for us to read, and it is based on how lower the mean similarity is from the threshold squared plus the .5 probability that already comes with the fact that the mean similarity is lower than the threshold set.

The same algorithm was applied to short term interest with the only difference being that the similarity threshold was being compared only to the similarity level between the last 2 sentences read.

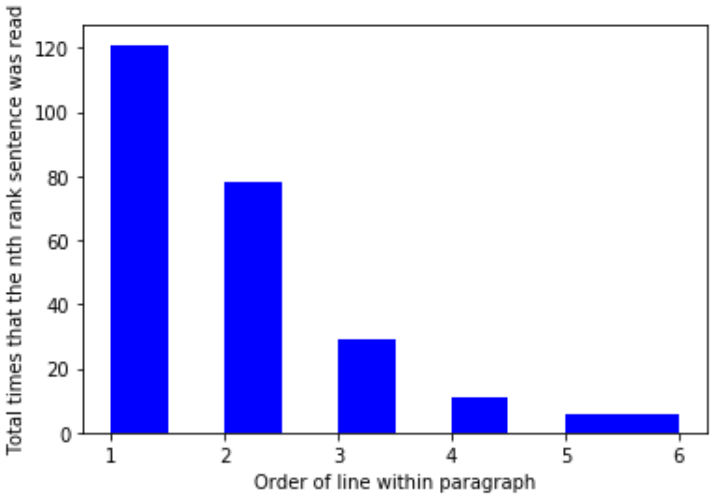
I had 2 more checks to see if we wanted to keep reading. These are regarding the important words list we got from the tfi-df weightings. The important words for the short range would be calculated amongst either the last 4 sentences read or the sentences read since the beginning of the new paragraph - whichever is smaller. If the number of important words in the last sentence were equal to the threshold there would either be another random choice or a straight read decision if the words exceeded the threshold. The same would happen for the long term important words, the difference is that those were selected out of the last 12 sentences read, the reasoning being that the long term important words were supposed to reflect what concepts were important in a greater scope/range.

If none of these statements were successful at passing we would jump, and the jump decision is also tied to how lower the similarities are from the thresholds. A jump value would be set and the upcoming sentences for the amount of jump set, would not be read and omitted. I should not explain the model in detail as the code is available to you, I just wanted to paint an outline of what was going on so that the explanations on this paper would be more grounded.

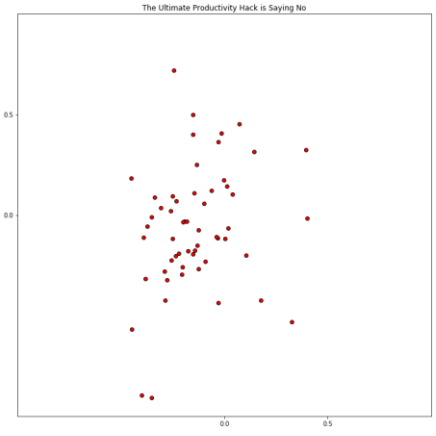
Another last thing to add is that the long/short term interest rates and most important word thresholds would be constantly updated by the constant set by the paragraph correction values. As we advanced through the paragraph the interest rates would go lower which would make it harder to make the lower cut - for the sentence to be not similar enough to cause interest. The most important word threshold would also go higher. The reasoning behind this was to simulate the natural decrease in the readers’ interest in a text as you got deeper into a paragraph. There is a feeling that a paragraph is a coherent block of sentences with a common idea and as people read more into it the easier it becomes for them to feel like they already exhausted the information gain they hoped from the paragraph and this encourages them to just skip on the rest of the paragraph which is -to me, something universal every person feels when reading.

The Testing

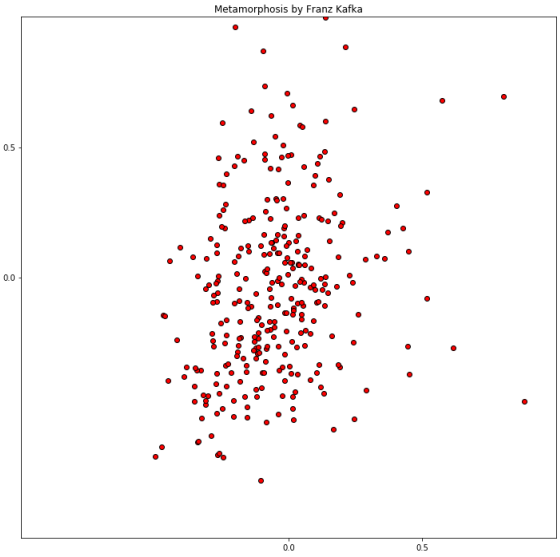
Since the nature of my model does not allow for much of an objective statistical evaluation, the tests I ran on the model were pretty limited. The most important thing to me was to see the same decay in the graph for the number of sentences read versus the distance of the sentences from a paragraph beginning:

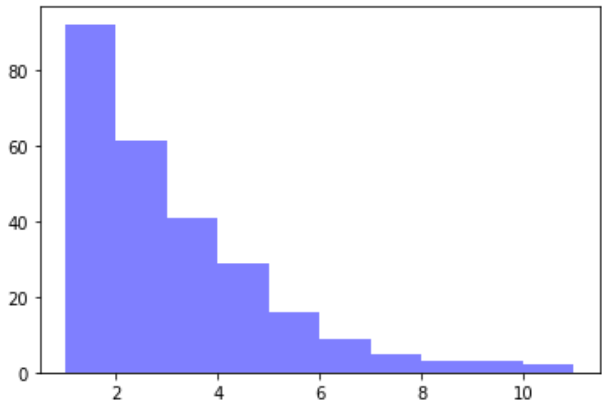


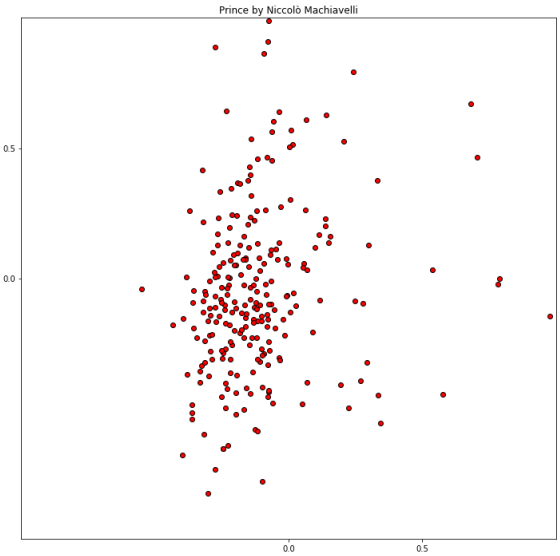
I ran my model on several small articles I found online such as this article called “The Ultimate Productivity Hack is Saying No”. I ran the model 10 times on the same text and got this graph on the left. We see that the first sentences are always read much more than the others and there is a gradual decrease as was shown in the eye tracking experiment.

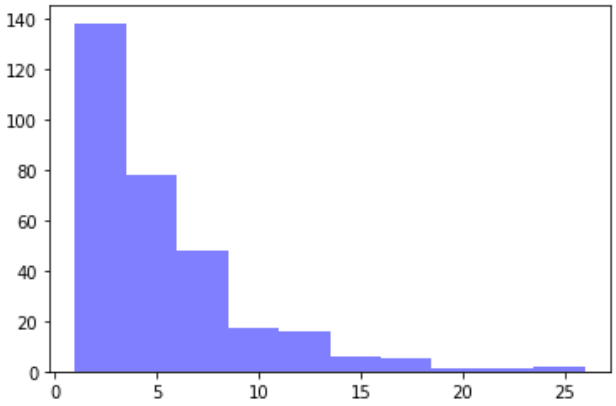


Here on the left is the average sentence vectors from the same article. It does not show any evidence for whether the model is working fine or not but you can see some sentences on the vectorial map that are pretty close to each other, which shows that model was capable in keeping reading for coherent parts of the article.









These are my results for running the model on the books Prince and the Metamorphosis. The same trend for the early sentences in the paragraph being read more shows here.[9]

Looking at the percentages of the text read also revealed that it was steadily between around 45 to 60 percent of the entire text.

9 The code I submitted was altered to give me the distance graphs, if you want to recreate the sentence vector graphs, you need to go to the model file and change the return value from the dist_para_bin to the avg_vector_bin, so that the “skim” function can plot the average vectors.

Below is the sentences read from the article *The Ultimate Productivity Hack is Saying No* (can be accessed here: https://jamesclear.com/saying-no).
The percentage read in this case was: 55.00000000000001

The dollar "$" sign denotes the beginnings of the sentences:

The Ultimate Productivity Hack is Saying No

The ultimate productivity hack is saying no. $ Not doing something will always be faster than doing it. This statement reminds me of the old computer programming saying , " Remember that there is no code faster than no code. " $ The same philosophy applies in other areas of life. For example , there is no meeting that goes faster than not having a meeting at all. $ This is not to say you should never attend another meeting , but the truth is that we say yes to many things we do n't actually want to do. There are many meetings held that do n't need to be held.

Many of them are not , and a simple " no " will be more productive than whatever work the most efficient person can muster. $ Why We Say Yes We agree to many requests not because we want to do them , but because we do n't want to be seen as rude , arrogant , or unhelpful. Often , you have to consider saying no to someone you will interact with again in the future—your co-worker , your spouse , your family and friends. $ Saying no to these people can be particularly difficult because we like them and want to support them. (Not to mention , we often need their help too.) Collaborating with others is an important element of life.

In reality , they are not just opposite in meaning , but of entirely different magnitudes in commitment. When you say yes , you are saying no to every other option. $ I like how the economist Tim Harford put it , " Every time we say yes to a request , we are also saying no to anything else we might accomplish with the time. " Once you have committed to something , you have already decided how that future block of time will be spent. $ In other words , saying no saves you time in the future. Saying yes costs you time in the future. No is a form of time credit. You retain the ability to spend your future time however you want.

It is also a strategy that can help you become successful. You need to say no to distractions. As one reader told me , " If you broaden the definition as to how you apply no , it actually is the

only productivity hack (as you ultimately say no to any distraction in order to be productive). ” $ Nobody embodied this idea better than Steve Jobs , who said , “ People think focus means saying yes to the thing you ’ ve got to focus on. But that ’ s not what it means at all. It means saying no to the hundred other good ideas that there are. You have to pick carefully. ” $ There is an important balance to strike here.

This period of exploration can be particularly important at the beginning of a project , job , or career. $ The opportunity cost of your time increases as you become more successful. At first , you just eliminate the obvious distractions and explore the rest. As your skills improve and you learn to separate what works from what does n't , you have to continually increase your threshold for saying yes. $ You still need to say no to distractions , but you also need to learn to say no to opportunities that were previously good uses of time , so you can make space for great uses of time. It 's a good problem to have , but it can be a tough skill to master. $ In other words , you have to upgrade your “ no 's ” over time.

It just means you default to saying no and only say yes when it really makes sense. $ How to Say No Most of us are probably too quick to say yes and too slow to say no. It 's worth asking yourself where you fall on that spectrum. $ If you have trouble saying no , you may find the following strategy proposed by Tim Harford , the British economist I mentioned earlier , to be helpful. He writes , “ One trick is to ask , “ If I had to do this today , would I agree to it. ” It ’ s not a bad rule of thumb , since any future commitment , no matter how far away it might be , will eventually become an imminent problem. ” $ If an opportunity is exciting enough to drop whatever you 're doing right now , then it 's a yes. If it 's not , then perhaps you should think twice.

$ It 's impossible to remember to ask yourself these questions each time you face a decision , but it 's still a useful exercise to revisit from time to time. Saying no can be difficult , but it is often easier than the alternative. As writer Mike Dariano has pointed out , “ It ’ s easier to avoid commitments than get out of commitments.