ZOMBIES READING SEGMENTED GRAPHENE ARTICLES ON THE ARXIV

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Alexander Amir Alemi

August 2015

ZOMBIES READING SEGMENTED GRAPHENE ARTICLES ON THE ARXIV

Alexander Amir Alemi, Ph.D.

Cornell University 2015

I present results obtained in four very distinct areas. In Chapter 1, I investigate vector based embedding models for text as applied to a corpus of scientific articles from the arχiv . I report on the utility of the learned word representations, and experiment with several techniques for learning vector representations for articles. I go on to discuss extensions to categories, authors, and readers, and the utility these would provide for enhancing the experience of arχiv users. Chapter 2 reviews work in the field of text segmentation, in particular the segmentation of long sequences of text into coherent topical sections. I introduce a novel segmentation algorithm that achieves state of the art results on a standard test set. Chapter 3 investigates a model of the fictional disease: zombies. I use zombism as an entertaining platform for investigating and exploring techniques in epidemiology modelling and critical phenomena. Chapter 4 summarizes work done towards building a group theoretic framework for creating generalized free energy expansions for elasticity. I found the first 50 terms in the expansion and show that these can be reliably fit to simulated data.

BIOGRAPHICAL SKETCH

Alexander Amir Alemi was born and raised in West Bend, Wisconsin. He moved to Orlando, Florida during his high school years. He attended the California Institute of Technology, earning a Bachelors of Science in Physics in 2009. He then came to Cornell University as a Physics PhD student, studying under James P. Sethna. He earned his Master's Degree in Physics from Cornell in 2013.

To my wife, without whom I would be much less.

ACKNOWLEDGEMENTS

I have had a tremendous time these past years at Cornell. In many ways I consider myself to be extremely priveledged, being exposed to all of the wonderful professors, postdocs, graduate students, undergraduates and staff that the Cornell Physics department houses. I have been allowed to pursue a wide variety of topics and fields, and been able to work with many different groups on a whole slew of different projects.

Thank you to my advisor, Jim Sethna, for continuously challenging me, engaging me in interesting discussions, and allowing me the freedom to pursue a wide array of different topics. Thank you to Paul Ginsparg, for serving as a second advisor to me, and introducing me to many interesting problems and people in computer science.

Thank you to many other professors in the department: Paul McEuen, for serving on my committee and writing an entertaining book. Veit Elser for countless mathematical puzzles. Chris Myers for entertaining many discussions about zombies.

Thank you to the other students in our group that I intersected with: Mark Transtrum, Woosong Choi, Yong Chen, YJ Chen, Ben Matchta, Ashivni Shekhawat, Ricky Chachra, Matthew Bierbaum, Colin Clement, Lorien Hayden, Katherine Quinn, and Archishman Raj, for fantastic discussions and support.

Thank you to my close friends for countless engaging lunch discussions over the years.

Thank you to my family for getting me to this point, and finally, thank you to my wife, Mallory for helping getting me through to the end, and whatever is to come next.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

ARXIV

We use several modern methods for learning vector representations of words to analyze a corpus derived from arχiv articles. We try various methods for extending the word vectors to power vector representations of articles and evaluate their performance on various metrics, both quantitative and qualitative. We discuss schemes for extending these methods to learn vector representations of articles, words, contexts, categories, authors and readers simultaneously, and discuss the possible applications of these representations.

1.1 Statistical Mechanics of Words

In Statistical Mechanics, we are used to studying complicated systems with many degrees of freedom. In its nearly 200 years of development, a single central lesson has emerged. It is to embrace the old maxim: *if you don't have anything nice to say, don't say anything at all*. A surprisingly powerful method for dealing with and describing a system you know very little about is to express only that little, nice, bit you know, and let the rest remain as random, or as unsaid as possible. We're speaking, cryptically, about the *principle of maximum entropy*.

In order to leave things unsaid, we first need to quantity a notion of 'unsaidedness', or uncertainty. As Shannon taught us [92], there is unique mathematical form that measures the uncertainty of a probability distribution subject to three simple constraints. The measure must be positive, monotonic in the number of states, and decompose linearly over independent subsystems. That measure is the *entropy*:

$$H(x) = -\sum_i P(x_i)\log P(x_i)\ . \tag{1.1}$$

What then does this principle look like? Assume we are forced to attempt to explain a system. We know very little about this system. It is large, complicated and there is a lot going on. But, we have the opportunity to measure a small set of expectations, or averages, (ϕ_k) of our system for some set of functions (f_k):

$$\phi_k = \sum_i P(x_i) f_k(x_i)\ . \tag{1.2}$$

What is the probability distribution ($P(x)$) that conforms with these expectations, but is otherwise as noncommittal as possible? We can solve for it. Let's simply maximize the entropy, subject to the constraints that the probability distribution is normalized, and that it conforms to the measured expectations.

$$P(x) = \arg\max\left\{-\sum_i P(x_i)\log P(x_i) - \beta_0\left(1-\sum_x P(x)\right) + \sum_k \beta_k\left(\phi_k - \sum_i P(x_i) f_k(x_i)\right)\right\} \tag{1.3}$$

Taking the variation with respect to $P(x_i)$ we find a condition that must vanish:

$$0 = \sum_i \left[-\log P(x_i) - 1 - \beta_0 - \beta_k f_k\right]\ . \tag{1.4}$$

which implies the maximum entropy distribution takes the form:

$$P(x_i) = \frac{1}{Z}\exp\left(-\sum_k \beta_k f_k(x_i)\right)\ , \tag{1.5}$$

where Z is our normalization constant, or *partition function*:

$$Z = \sum_i \exp\left(-\sum_k \beta_k f_k(x_i)\right) \tag{1.6}$$

where the values of the Lagrange multipliers (β_k) can be determined from the observed expectations:

$$\phi_k = \langle f_k\rangle = \sum_i P(x_i) f_k(x_i) = -\frac{\partial}{\partial\beta_k}\log Z\ . \tag{1.7}$$

Imagine we have a physical system that can take on a tremendous number of individual states s_i, each of which has a particular energy $E(s_i)$. If all we know about the system is its average energy ($\langle E \rangle$), the principle of maximum entropy suggests that we take as the probability distribution over all possible states:

$$P(s_i) = \frac{1}{Z} \exp\left(-\beta E(s_i)\right) \ . \tag{1.8}$$

The fact that this is the Boltzmann distribution of statistical mechanics is no accident. All of statistical mechanics can be formulated in terms of the principle of maximum entropy [41, 42].

1.1.1 Words

With this principle in mind, we attempt to formulate a theory of word usage. We will proceed similar to the way Stephens and Bialek modelled the distribution of letters in 4-letter English words [98]. The goal here is ultimately some description for the probability of a sequence of words occurring in a stream of text.

$$P(w_1, w_2, w_3, \cdots, w_n, w_{n+1}, \cdots) \tag{1.9}$$

Assume a very long sequences of words, and that this probability distribution is locally homogeneous. Further, we'll assume that the only expectations we are willing to specify are the cooccurrence statistics of pairs of words. This enables us, by the principle of maximum entropy to state that our best guess for the probability distribution for a sequence of words is:

$$P(\{w_i\}) \propto \exp\left(-\sum_i \sum_k J_k(w_i, w_{i+k})\right) \tag{1.10}$$

That is, assuming homogeneity and some observed cooccurrence statistics, streams of words in text behave as if they were governed by pairwise interactions between the words, where in general the interaction potentials would be different for every possible displacement k. Each one of these potentials will have a term involving every possible pair of words. If we assume our vocabulary is on the order of 100,000 words, specifying this language model would would require specifying $(100,000)^2 \times K \sim K \times 10^{10}$ parameters for interactions up to distance K.

To make further progress requires simplifying assumptions. Let's assume that the interaction decays with distance, and that $V_k = V(K - k)$. That is, the pairwise interactions are the same at every separation, but they decay linearly up to some cutoff K. This still leaves a very large set of parameters for specifying the interaction. Next, assume that the interaction between any pair of words is symmetric, so that $V_{ij} \equiv V(w_i, w_j) = V(w_j, w_i) = V_{ji}$, which will cut the number of parameters roughly in half.

At this point we will have $\sim \frac{1}{2} \times 10^{10}$ parameters in our model. To make further progress requires boldness. Assume that this interaction matrix is low rank. That is, represent our interaction V_{ij} as the product of a much smaller matrix W_{ik} with its transpose:

$$V_{ij} = \sum_m W_{im} W_{jm} \ . \tag{1.11}$$

With all of these simplifying assumptions, we obtain a model for the probability of a sequence of words occurring in text of the form:

$$P(\{w_i\}) \propto \prod_{i,k} \exp\left(-(K-k) \sum_m W_{im} W_{(i+k)m} \right) \tag{1.12}$$

This model only has $\sim D \times 10^5$ parameters, where D is the rank chosen for the decomposition. Notice also that the matrix W can be interpreted as defining a D dimensional vector for each word.

This is essentially the model we will study throughout this Chapter, and various small modifications of it. It turns out to be a very powerful model for text, and the vectors that represent each word turn out to encode some very interesting relationships between words if taken seriously.

1.2 Modern Vector Based Representations

Here we give an introduction and overview to various modern methods for learning vector representations for words.

1.3 `word2vec`

Vector based word representations are not new, but have seen a renewed interest following the series of papers by Mikolov and collaborators introducing `word2vec` [59, 62, 63]. At its heart, `word2vec` attempts to learn a vector to represent each word, where the dot products between different words encode their likelihood of cooccurring in the document, and where cooccurrence is measured as appearing together in some small window in the text.

1.3.1 Architectures

The original `word2vec` paper [59] offered two different architectures, the first of which was the *continuous bag of words* or CBOW method.

In CBOW, we attempt to use the vectors for the words surrounding a given

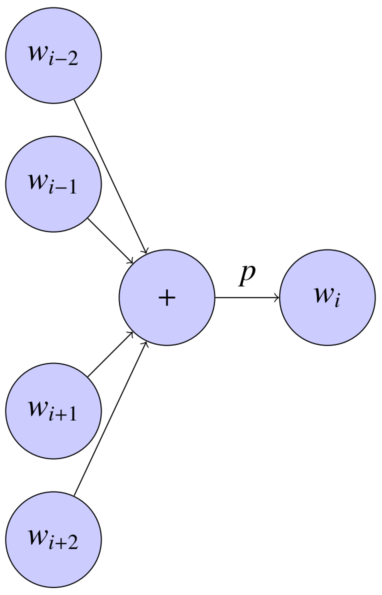


Figure 1.1: The *continuous bag of words* model uses the vectors for the surrounding words to build up a vector for the local context, originally by adding those vectors together. It then uses that feature vector to predict the central word using a classifier.

word to build up a representation of the local context. In the original paper, this was done by adding all of the surrounding word vectors together. This input vector is then used to try to predict the given word, by utilizing a softmax layer. A *softmax* is a differentiable version of a max function, defined as:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \ . \tag{1.13}$$

When any one of the inputs is large compared to the rest, the softmax will set that input near one, and the rest to near zero. A schematic of the CBOW architecture is shown in Figure 1.1.

In the original work, the CBOW model did not perform as well as the skip-gram model, described below. Much of the follow-up work on `word2vec` and vector embeddings more generally have focused on other architectures, but we mention CBOW here given its similarity to some of the models we will explore for article vectors.

The second suggested architecture, and the more successful one in their experiments was the *skip-gram model* [59]. Here we use each word to try to predict each of the words that surround it. A schematic of the architecture is shown below in Figure 1.2

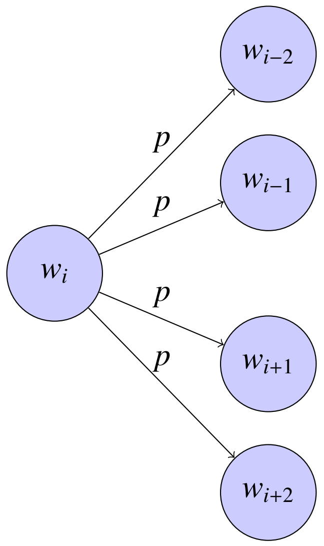


Figure 1.2: The skip-gram model uses the vector for the current word to predict each of the surrounding words using a classifier.

We want the arrows to denote a sort of classifier that uses the input to produce a probability distribution over the possible output words. A natural choice for this would be to use a Boltzmann distribution over the outputs, in terms of some energy function depending on both the output and input:

$$p(o|i) = \frac{e^{-E(o,i)}}{\sum_{o' \in O} e^{-E(o',i)}} = \frac{1}{Z(i)} e^{-E(o,i)} \, . \tag{1.14}$$

We need to define our energy function, and consequently, our representation. For our representation, we will choose a vector in an n-dimensional space. A natural energy function then is a simple linear function, namely the dot product between our input vector and output vector:

$$E(o,i) = -\boldsymbol{o} \cdot \boldsymbol{i} \tag{1.15}$$

Where $(\boldsymbol{o}, \boldsymbol{i})$ is the vector representing the (output, input) respectfully, and we've chosen our energy to correspond to the ferromagnetic case, wherein our vectors want to point in the same direction. We have just defined an instance of the common log-linear model, used throughout natural language research [7]. This is also the model we motivated by appeals to maximizing the entropy of the distribution in the introduction.

1.3.2 Log Linear Model

The log linear model can be considered an explicit form for the conditional probabilities, where the log probability of a nearby word w_{i+k} occurring, conditioned on the occurrence of the central word w_i, will be proportional to their dot product:

$$\log P(w_{i+k}|w_i) \propto \boldsymbol{w}_j \cdot \boldsymbol{w}_i \; . \tag{1.16}$$

Here $\boldsymbol{w}_i$ represents the vector associated with word w_i. The properly normalized conditional probability takes the form:

$$P(w_j|w_i) = \frac{\exp\left(\boldsymbol{w}_j \cdot \boldsymbol{w}_i\right)}{\sum_k \exp\left(\boldsymbol{w}_k \cdot \boldsymbol{w}_i\right)} = \frac{1}{Z_i} \exp\left(\boldsymbol{w}_j \cdot \boldsymbol{w}_i\right) \; . \tag{1.17}$$

Z_i represents the *partition function* for word w_i. This conditional probability function takes the form of a *softmax* function (1.13) in terms of the dot product between the word vectors.

In other words, at a particular point in our document, we will use the vector $\boldsymbol{w}_i$ to represent word w_i, and we wish to predict a nearby word (w_j). We consider all possible candidates $\{w_j\}$ with associated vectors $\{\boldsymbol{w}_j\}$ and assign probabilities as Boltzmann factors of an energy defined as the dot product between the word vectors. As a result, word vectors that point together are said together. The normalization afforded by the partition function ensures that we don't get too carried away and assign arbitrary weight to a pair of words cooccurring. If we simply increase the lengths of a pair of word vectors, the probability will remain properly normalized, and we will simply assign more probability mass to that pair.

The total log likelihood for our entire corpus is then

$$\mathcal{L} = \sum_i \sum_{k \in [-w,w], k \neq 0} \log P(w_{i+k}|w_i) \, . \tag{1.18}$$

Having defined what should be a suitable objective, it remains to try to train it. Given that the objective is differentiable, one could imagine proceeding by some form of gradient based optimization, some details of which we'll discuss later, but the objective as it is currently defined is particularly hard to train. It is precisely the presence of the partition function term, which helps ensure that the word vectors do not just grow unbounded in magnitude, that presents the largest computational hurdle. If we were to take the objective at face value, for every pair of words we took to be in cooccurrence, we would have to evaluate the sum of the exponentials of the dot products with every word in our vocabulary. But for a large corpus, the vocabulary can be expected to be on the order of millions. The real contribution of the `word2vec` papers and associated code was to contribute practical techniques for mitigating this challenge. Below, we will summarize the two main approaches. More information about the details of the `word2vec` objective can be found in [87, 30, 62].

1.3.3 Hierarchical Softmax

The first way to attack the problem of the partition function is to employ the rather clever trick of *hierarchical softmax* [59, 65, 67]. Hierarchical softmax allows an approximation to the partition function, which itself involves a sum over N terms, with an approximate function that involves a sum over, on average, $\log N$ terms.

At its heart, hierarchical softmax relies on the ability to factor conditional probability distributions over classes. Imagine that we assign each word to a single class, or cluster, $c(w) = k$. Then we can write:

$$P(w|v) = \sum_k P(w, c = k|v) = \sum_k P(w|c = k, v)P(c = k|v) = P(w|c = c(w), v)P(c = c(w)|v) , \tag{1.19}$$

since each word is a assigned to a single cluster. The important thing to note is that in order to normalize $P(w|v)$ we would need to do a sum over N where N is the number of words in our vocabulary. To normalize the right hand side, we would need only sum over the number of words in the associated class, and the number of classes. So if we have 100 words, split into 10 classes of 10 words each, instead of a normalization over 100 terms, we need only normalize two terms each with 10 terms. This is a large saving. Granted, it requires a new sort of model, were we model the word occurrences dependent on the nearby word and class label, as well as an independent model for the expected class given the word we condition on. This original trick was pointed out in [31].

For hierarchical softmax, we take this idea to its extreme and turn the partition function into a sequence of binary decision processes. If we represent the entire vocabulary in a binary tree, with leaves for each word, there will be N

total leaves, and $N - 1$ total internal nodes. In this new tree, each word can be reached by a particular path from the root node down to the corresponding leaf. That path is specified by the Huffman code for the word in question, a binary string of variable length, where we could take 0s in the code to denote moving to the left child and 1s as moving to the right.

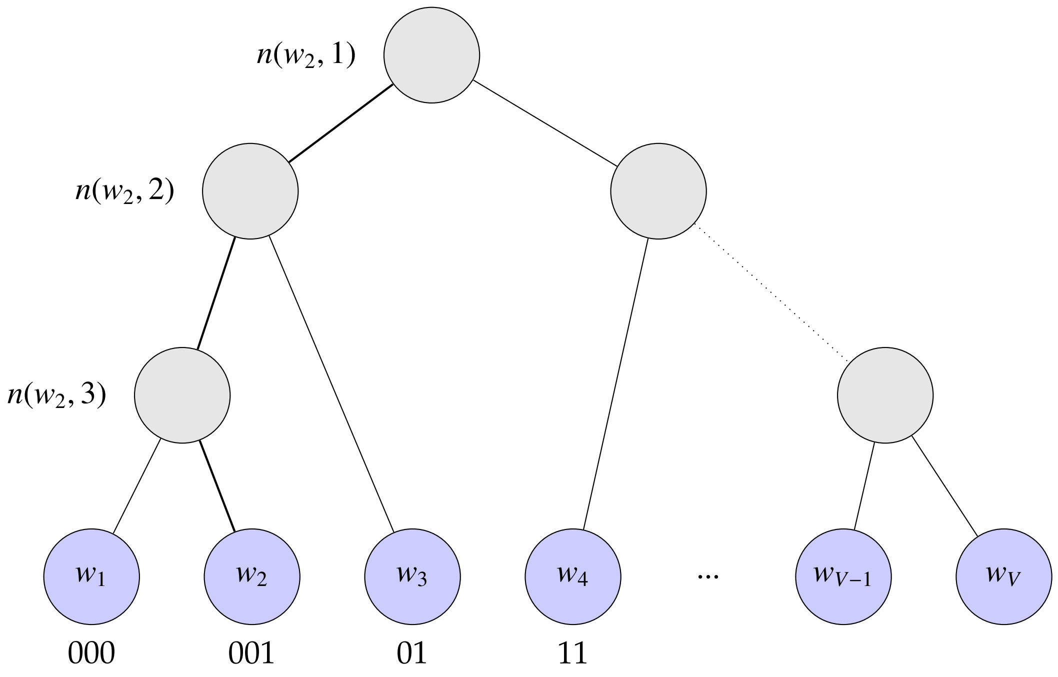


Figure 1.3: An example binary tree and the relationship between leaf nodes for the words, and inner nodes defining a path from the root to the words.

An example of a binary tree is shown in Figure 1.3, which has been recreated from [87]. The new representation of the conditional probability of a word given the input is given by, in equation (3) of [62]:

$$P(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left(\llbracket n(w, j+1) = ch(n(w, j)) \rrbracket \boldsymbol{v}'_{n(w,j)} \cdot \boldsymbol{w}_I\right) . \tag{1.20}$$

Here $\boldsymbol{w}_I$ is our input vector. The $\boldsymbol{v}'$ are the vectors representing each of our inner nodes. $n(w, j)$ is the jth parent of the leaf node w. $ch(n)$ represents the right child of node n. $[\![\cdot]\!]$ represents the binary function, which returns 1 if the argument is true, and -1 if it is false. σ is the *sigmoid function*

$$\sigma(z) = 1/(1 + e^{-z}) \,. \tag{1.21}$$

$L(w)$ is the length of the code for word w.

Trying to dissect this equation, we can think of the process of predicting a word, given some input, as taking a random walk through this binary tree. Each inner node of the tree has an associated vector, and we use the sigmoid of the dot product between the node vector and our input vector to decide whether we should go to the right or left child of that node. That is to say, we train a logistic classifier at each inner node to determine whether we should go left or right. The probability then of reaching a particular word w, is given by the series of choices we need to make to reach that word. For the example encoded in Figure 1.3, if we are trying to reach word w_2, the correct path to follow has the following probability

$$\begin{aligned}
P(w|w_I) &= P(\text{left at } n(w_2, 1))P(\text{left at } n(w_2, 2))P(\text{right at } n(w_2, 3)) && (1.22)\\
&= \sigma(\boldsymbol{v}_{n(w_2,1)} \cdot \boldsymbol{w}_I)\sigma(\boldsymbol{v}_{n(w_2,2)} \cdot \boldsymbol{w}_I)\left(1 - \sigma(\boldsymbol{v}_{n(w_2,3)} \cdot \boldsymbol{w}_I)\right) && (1.23)\\
&= \sigma(\boldsymbol{v}_{n(w_2,1)} \cdot \boldsymbol{w}_I)\sigma(\boldsymbol{v}_{n(w_2,2)} \cdot \boldsymbol{w}_I)\sigma(-\boldsymbol{v}_{n(w_2,3)} \cdot \boldsymbol{w}_I) && (1.24)
\end{aligned}$$

Where in the last line, we have utilized the fact that $1 - \sigma(z) = \sigma(-z)$. You'll notice that this turns determining the conditional probability of a given word into a calculation of the product of $L(w)$ terms, where $L(w)$ is the length of the code for word w. If we use the Huffman tree as our binary tree, the average code length for each word will be the log of the size of our vocabulary: $\log V$.

Notice also that this conditional probability is properly normalized, by its construction. The probabilities of going left or right at each node sum to one, and any word has a well defined probability, so the sum of the probabilities for all of the leaves will sum to 1. In effect, hierarchical softmax allows us to construct an explicit realization of a normalized conditional distribution, in terms of $V - 1$ unknown vectors which we aim to learn.

This also means that when we use hierarchical softmax, during training we will be learning a set of vectors for each word, and a set of vectors for each of these inner nodes. That second set does not correspond to another set of word vectors, just the parameters for this explicit normalized conditional probability model.

This is an approximation to our true skipgram objective (Equation 1.18), but an efficient one, replacing the partition function with a factorized conditional probability model. In practice, this approximation works well. Hierarchical softmax in general seems to depend somewhat sensitively on the particular choice of binary tree (see [65]), but the Huffman encoding works well for word vectors in practice.

1.3.4 Negative Sampling

The second approach to bypassing the partition function is also to switch objectives. Inspired by the idea of Noise Contrastive Estimation [66], we imagine a modified task, namely trying to distinguish the observed data from a model of noise. In the end, we have a new objective:

$$\mathcal{L} = \sum_i \sum_{k \in n(i)} \log \sigma(\boldsymbol{w}_{i+k} \cdot \boldsymbol{w}_i) + \frac{1}{N} \sum_i \sum_{n=1}^{N} \log \sigma(-\boldsymbol{v}_n \cdot \boldsymbol{w}_i) \tag{1.25}$$

In the second term, there is a sum of N draws of random words v_n, drawn from a noise distribution $P_{\text{noise}}(v)$, taken as proportional to the unigram probability distribution raised to the power of α ($P(w)^\alpha$). This can be interpreted as trying to train a logistic classifier to distinguish our data from the noise [30].

Following the discussion in [30], we can make sense of this objective. Instead of trying to learn the normalized conditional probabilities for the neighboring words given their center, we could instead try to learn to distinguish true data for pairs of nearby words from false data. In particular, let's say we had a model for $P(\text{true data}|w, c)$, the probability that the word and context pair came from our actual dataset. This coupled with a process to generate random data, suggests the complement:

$$P(\text{noise}|w, c) = 1 - P(\text{true data}|w, c)\ . \tag{1.26}$$

To model these binary decisions (is the pair true or false?) we could use a logistic classifier, powered by vector representations for the words and contexts:

$$P(\text{true}|w, c) = \sigma(\boldsymbol{w} \cdot \boldsymbol{c})\ . \tag{1.27}$$

Here $\sigma(z)$ is the sigmoid function of Equation 1.21.

It is clear to see what the negative sampling objective is doing. The first term tries to make words that occur together point together, but since we've dropped our partition function, we could arbitrarily increase this likelihood by just having all of the word vectors point in the same direction and have an arbitrary length. It is the second term that tries to do the job of the original partition function at restricting our vectors to be nontrivial, by attempting to make random pairs of words point in opposite directions. This can be thought of as a type of regularization, which prevents our model from overfitting.

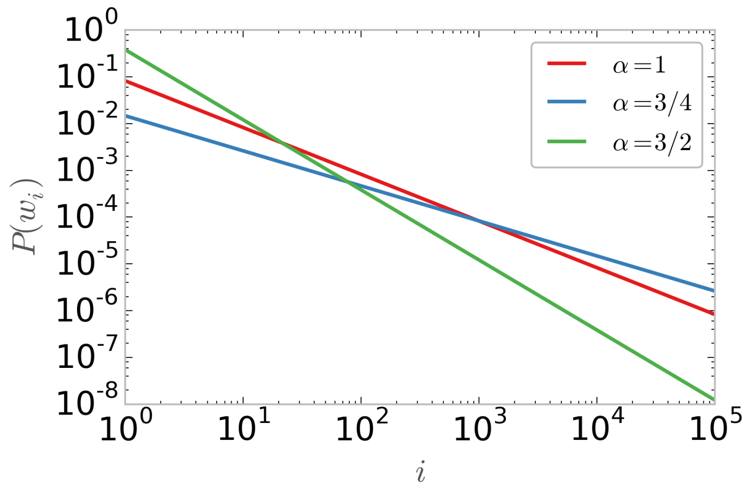


Figure 1.4: Using a noise distribution that is a power law scaling of the unigram distribution has the effect of moving probability mass towards the rarer words, provided the exponent is less than 1.

In [62], the noise was constructed explicitly at each word. That is, every time we considered a cooccurring pair of words in the corpus, we would take a single step of gradient descent to align the two word vectors, and then generate a fixed number of false target words, drawn from the unigram distribution to the 3/4 power:

$$P_{\text{noise}}(w_i) \propto (\text{count}_i)^{3/4} \quad . \tag{1.28}$$

The choice of power is somewhat experimental. Overall, you can think of the power law scaling of the unigram distribution as a form of smoothing, which moves probability mass from the most common words towards the less common words, provided the exponent is less than 1. Below in Figure 1.4, We've shown the probability distribution for a collection of words that follow a perfect Zipf law distribution, and the effect that the power law scaling has on the probability distribution, for two choices of power law, one less than 1 and one greater than 1.

This smoothing turns out to be very important. In a paper by Levy and Goldberg [55], the smoothing was one of the most important tweaks to the `word2vec` method that gave improved performance. The same smoothing can be easily generalized to other types of models and effect similar improvements. In fact, `word2vec` employs several tricks that improve its performance relative to other methods [55], including subsampling frequently occurring words before generating the windows, and dynamic window sizes that place more weight towards the closer words.

Another important distinction between negative sampling and hierarchical softmax is that negative sampling builds two different sets of word vectors, those for 'words' and those for 'contexts'. Normally, as in the original `word2vec` paper, just the word representations are extracted and used, but we could just as well consider the vector representations for the 'contexts', where there will similarly be a single vector for each word in our corpus. We could go yet further and combine these two representations, representing each word by the vector which is the sum of these two different representations. This was originally done in [78] for a competing method, and improved performance generally [55].

1.4 `GloVe`

In practice, the `word2vec` objective is trained as a streaming objective. The objective is minimized by reading the corpus a single word at a time in a single pass. Alternatively, one could turn the local streaming objective into a global one that operates on the total cooccurrence statistics for each pair of words. This is the idea behind a competing framework: `GloVe` [78].

The `GloVe` objective is:

$$\mathcal{U} = \sum_{ij} f(X_{ij}) \left(\boldsymbol{w}_i \cdot \boldsymbol{w}'_j + b_i + b'_j - \log X_{ij} \right)^2 \ . \tag{1.29}$$

Here X_{ij} is the total number of times w_i and w_j cooccur the corpus, $\boldsymbol{w}_i$ and $\boldsymbol{w}'_i$ are two separate sets of vectors representing words: one called the word vectors ($\boldsymbol{w}_i$) and the other called the *context* vectors ($\boldsymbol{w}'_i$). The b_i, b'_i represent bias terms for each word and associated context word, and f is a weighting function, taken in the work to be:

$$f(x) = \begin{cases} \left(\frac{x}{x_0}\right)^{\alpha} & x < x_0 \\ 1 & x \geq x_0 \end{cases} \ . \tag{1.30}$$

This weight function is sublinear with exponent $\alpha = 3/4$ for counts less than some threshold $x_0 = 100$ and 1 for all counts above the threshold. This has the effect of putting more weight on our loss terms as the counts get higher, up to some threshold x_0. Given that we approximate the cooccurrences with empirical counts from a fixed size corpus, this helps to lessen the contribution of low counts, where the noise is expected to be largest.

You should consider this objective as a weighted matrix factorization. It attempts to represent the matrix $\log X_{ij}$ as the product of two matrices with bias terms. The fidelity of the representation is measured by taking the square of the difference between the reconstruction and the original, similar to a Frobenius norm, with the caveat that it is weighted to reduce the influence of matrix elements that are small. Smaller elements contribute less to the overall loss.

Why should we be so interested in faithfully representing $\log X_{ij}$? As the `GloVe` article notes [78], the ratios of conditional probabilities for words can be very discerning. Consider:

$$\frac{P(w|\text{ice})}{P(w|\text{steam})} , \tag{1.31}$$

the ratio of a cooccurrence of a word with both ‘ice’ and ‘steam’. For a word like ‘solid’, this ratio will be large. For ‘gas’ it should be small. For unrelated words, such as ‘hat’, we expect this ratio should be near one. For words that apply to both, such as ‘water’, it should also be near one.

The conditional probabilities can be estimated from the observed word cooccurrence counts:

$$P(i|j) = \frac{X_{ij}}{X_j} . \tag{1.32}$$

Here X_j is the number of times word w_j appears in the corpus. This suggests meaningful relationships between words are reflected in the ratios of cooccurrence counts

$$\frac{P(k|i)}{P(k|j)} = \frac{X_{ki}}{X_{kj}} \frac{X_j}{X_i} \tag{1.33}$$

We want to encode these meanings, which are reflected in ratios, in terms of a linear structure on word vectors, which suggests we encode the logarithm of this relationship:

$$w_k \cdot (w'_i - w'_j) \sim (\log X_{ki} - \log X_i - \log X_k) - (\log X_{kj} - \log X_j - \log X_k) , \tag{1.34}$$

where we have added and subtracted $\log X_k$ on the right hand side. This brings us to and objective of the form:

$$w_i \cdot w'_j \sim \log X_{ij} - \log X_i - \log X_j . \tag{1.35}$$

In the `GloVe` method proper, we allow a general bias term that we additionally learn:

$$w_i \cdot w'_j \sim \log X_{ij} - b_i - b'_j . \tag{1.36}$$

To enforce this structure, a very natural loss is the quadratic loss:

$$\left(w_i \cdot w'_j + b_i + b_j - \log X_{ij}\right)^2 . \tag{1.37}$$

The `GloVe` objective attempts to enforce this structure for all pairs of words (i, j), with a weighting term that both de-emphasizes the contribution of the low count cooccurrences, and avoids the singularity present for $X_{ij} = 0$.

1.5 arχiv Dataset

This work focusses on applying vector based methods to a dataset corpus derived from the arχiv [2]. The arχiv is a preprint repository of scientific articles. In an effort to encourage access to scientific work, and to help speed the flow of collaboration and access, researchers are encouraged to upload a preprint version of their work to an online repository, viewable by all. Currently run by the Cornell University Library system with help from donations contributed by other major university libraries, as of the time of this writing, the arχiv has 1,034,387 e-prints available to read, in various subject areas spanning Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance, and Statistics.

For most of our analysis here, we will focus on a subset of the arχiv articles, namely those uploaded in January, February and March of 2015. This consists of 25,142 articles after hand pruning a few.

1.5.1 Preprocessing

Authors are encouraged to upload the raw TEX or LATEX for their article, and the arχiv makes the raw source as well as pdfs available. However, some authors choose only to upload a compiled pdf of their work. For uniformity, our dataset then consists of text extracted from these pdfs, using the `pdf2txt.py` python utility.

Unfortunately, pdfs are not the most amenable to text extraction. In particular, a large amount of mathematical equations and notation tends to be garbled by the translation process. We preprocess the raw `utf-8` encoded text to remove all non-ASCII characters, replacing each with a space. The text is then lowercased. Some characters fail to be converted by `pdf2txt.py`, because of encoding or font issues. These are represented by 'UNK' in our corpus. We then remove any pure number, so that 'word2vec' would be allowed but 1230 would be removed. We also make each article a single line. This leaves us with a nearly 1 gigabyte ASCII corpus with 25,142 lines, one for each article.

1.5.2 Mutual Information of Nearby Words

Since the previous methods are powered by word cooccurrence, we could ask the simple question: How much does a word influence its neighboring words? In an effort to answer that, we can attempt to calculate the mutual information $I(w_i, w_{i+k})$ between a word and a word some finite distance away from it. In order to facilitate computation, we will utilize the fact that the mutual information is the sum of the entropies of each random variable alone, minus their joint entropy:

$$I(w_i, w_{i+k}) = H(w_i) + H(w_{i+k}) - H(w_i, w_{i+k}) \, . \tag{1.38}$$

By observing the frequencies of pairs of words occurring a fixed distance apart, we can obtain an estimate of the mutual information between those words. Notice first that for our raw vocabulary of 85,515 words that occur at least 30 times, the mutual information of a word with itself is just its entropy, which we observe to be 10.2 bits. Below in Figure 1.5 We've plotted the mutual information as a function of word separation.

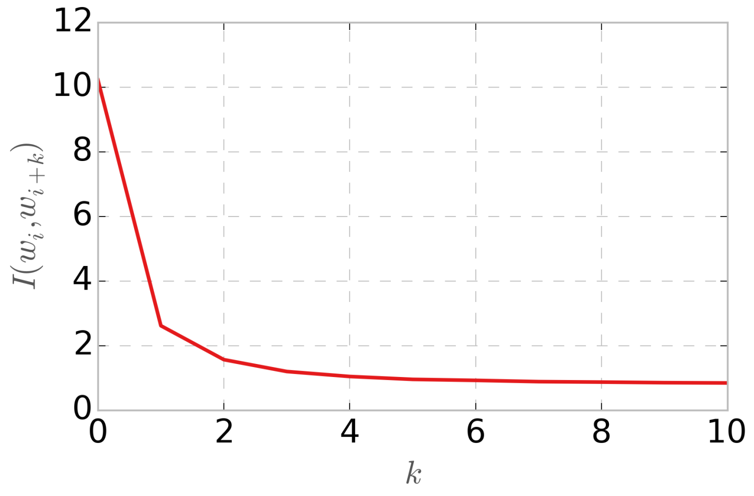


Figure 1.5: The observed empirical mutual informations for words a fixed distance apart, for the raw vocabulary of terms appearing at least 30 times in the arχiv corpus. Notice that this mutual information decays rapidly initially, but has a long tail as we move to larger separations. The mutual information drops below 1 bit at a 5 word separation, suggesting that we need not consider windows that are too large when we try to learn our vector representations.

Between a given word and its neighbor, there are 2.6 bits of mutual information. Seen another way, this means that if we observe a word, the residual entropy for the next word ($H(w_{i+1}|w_i) = H(w_{i+1}) - I(w_i, w_{i+1})$) is only 7.6 bits. If words occurred with equal frequency, this would correspond to a drop in the size of the vocabulary from $2^{10.2} \sim 1,200$ words to only $2^{7.6} \sim 200$ words. This is a large reduction in complexity. The mutual information decreases as we get to larger word separations, as can be expected, but has a long tail, with 0.85 bits of mutual information remaining even 10 words apart.

1.5.3 Variable *n*-grams

Not all word pairs are created equal. In an effort to reflect the fact that "black hole" represents a single semantic entity, distinct from "black" and "hole" considered together, we preprocess the corpus using a statistical approach to join frequently occurring *n*-grams, the same approach used in the `word2vec` work [62].

This consists of four passes of joining commonly occurring bigrams in the corpus, where two words are joined with an underscore if they pass a certain threshold:

$$\frac{\mathrm{count}(w_i, w_j) - \delta}{\mathrm{count}(w_i)\mathrm{count}(w_j)} \geq \tau \; . \tag{1.39}$$

Here δ ensures that the bigram has to occur a certain number of times. For this work, we set $\delta = 40$ and do four passes with decreasing thresholds $\tau = [400, 300, 200, 100]$ in order to create relatively long *n*-grams.

To facilitate computation, we then threshold the vocabulary to consist of *n*-grams that occur at least 30 times in the corpus, leaving us with a vocabulary of 133,419 'words'.

For instance, after this process `quantum_mechanics`, `black_hole`, and `power_spectrum` are all recognized as single 'words'. Some of the longest *n*-

grams correspond to `pdf2txt.py` renderings of common references:

- `phys_rev_lett_url_http_link_aps_org_doi_physrevlett`
- `royden_real_analysis_3rd_ed_upper_saddle_river_prentice_hall`
- `issn_url_http_www_sciencedirect_com_science_article_pii`

or common phrases:

- `there_exist_positive_constants_c1_c2`
- `authors_declare_no_competing_financial_interests`
- `multiplicative_contractive_completely_positive_linear`
- `root_mean_square_error_rmse`

The observed 'word' counts show an approximately Zipfian or power law distribution, wherein the occurrence of a word is proportional to the inverse of its rank.

$$P(w) \propto \frac{1}{\text{count}(w)} \tag{1.40}$$

We plot the observed frequency distribution of the words in Figure 1.6 below.

1.5.4 Multiple Domains

The arχiv dataset contains more than a simple corpus of word usage. There is a natural division of the corpus into independent articles (25,142 articles for this corpus). Associated with each article is a set of metadata, consisting of the categories the article is assigned, its authors, even its readers. This is in many ways the central question for this portion of the thesis, namely, can we extend

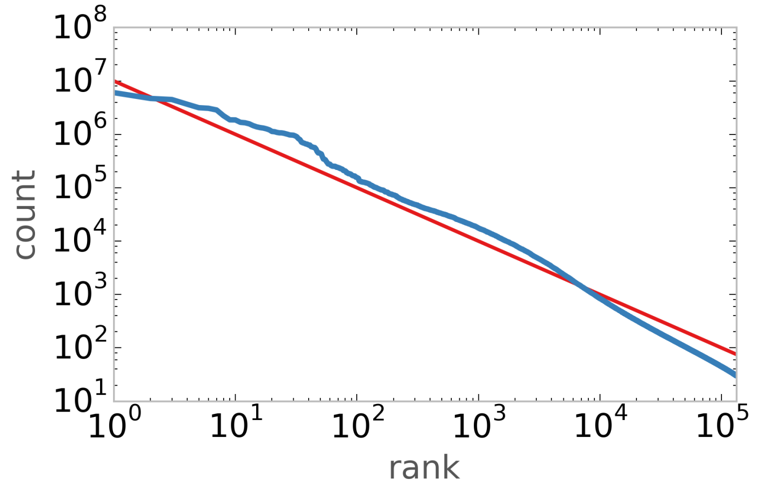


Figure 1.6: The observed counts as a function of rank for the words in the arχiv corpus. The word counts show an approximately Zipfian distribution, imposed here in red.

the idea of vector based embeddings to additionally represent articles, authors, readers, and categories with vectors living in a similar, or the same space as the words themselves.

To be precise, each article on the arχiv has a single primary category to which it has been assigned, and optionally, can be cross listed in other categories as well. Each article naturally has a list of authors. Each article also has a summary, usually taken to be the abstract, which is separately available and shown to potential readers before they download the pdf, which we could consider separately.

We also had access to some anonymized access logs, not publicly available, at the cookie or IP level. Every time someone views an article's abstract page, or downloads its pdf, it leaves a trace on the server, associated with a cookie, which should persist across different sessions with the same browser for that user (assuming they do not clear their cookies). The user's IP address is also logged, which could potentially link different browsers on the same machine to the same cookie or login, but IP addresses are often rather fleeting these days, as many users' ISPs will assign different IP addresses every time their router reconnects.

1.5.5 Minor and Major Categories

For the corpus in question, there are 147 different categories represented, of which only 143 appear as the primary category. Some categories are duplicates of one another, such as math-ph (Physics: Mathematical Physics) and math.MP (Mathematics: Mathematical Physics). These (minor) categories are at a fairly fine level, and can be coarsened into a larger scale separation of the articles, at the highest level into just 7 major categories: Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance, and Statistics. The occurrence of each of the primary minor and major categories is shown below in Figure 1.7. Notice that there is a large discrepancy between the most popular category: `quant-ph`: Quantum Physics with 1119 articles in this period, and `cs.OS`: Operating Systems, with only 4. The same discrepancy holds even at the major category level, where physics saw 13,645 articles, while quantitative finance saw only 156. In Figure (1.7), we've given each category a color, where the major categories were each assigned a hue, and then the minor categories were given random variations on that hue. The physics categories are green, math are blue, cs are red, stat are yellow, q-bio are orange, and q-fin are purple. Later, we will use these colors in detailed visualizations, and in this way, Figure 1.7 will serve as a key to the color scheme.

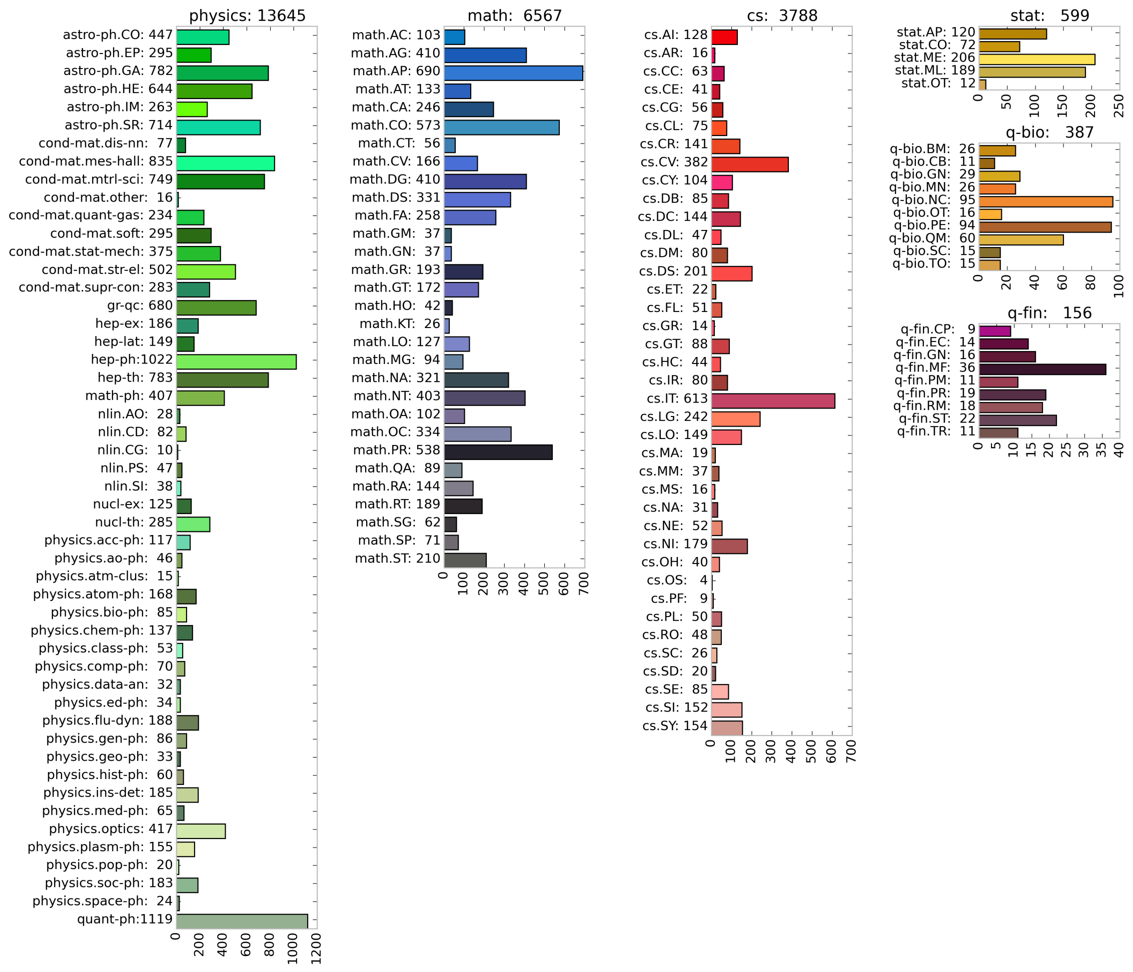


Figure 1.7: The observed category counts for the arχiv corpus. The 142 minor primary categories are grouped into the 6 major categories. The counts for each major and minor category are shown next to the name. Notice that the *x*-scaling is not uniform across major categories in order to emphasize the differences in population for the less populated major categories.

1.5.6 Authors

For the given 3 month corpus, there are 73,497 distinct authors listed across all of the papers. In Figure 1.8, we've shown the distribution for the number of papers associated with each author.

Due to discrepancies between the way author names appear across different

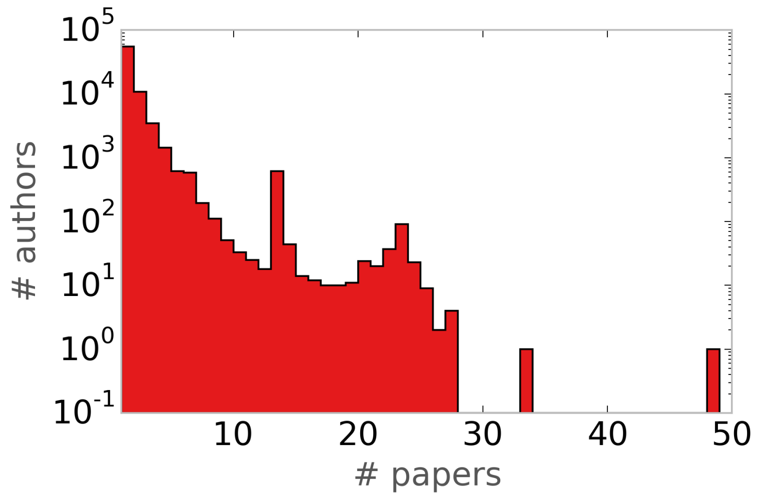


Figure 1.8: The observed article counts for each author for the 3 month arχiv corpus.

papers, each author is reduced to Last Name, Initial form. This helps prevent a single author from appearing under different names, but can obviously cause some ambiguity problems for common names. The most prolific author for this window was "de Oliveira, H. M.", with 48 papers, who it appears took the time to upload most of his previous publications earlier this year. The second most prolific author was "Zhang, Y.", with 33 papers. This demonstrates the downside of collapsing names in this way.

1.5.7 Readership

For readership data, we decided to look at the cookie level. We utilized all of the cookies from the start of the year until the present day, for both the viewing of abstract pages, as well as pdfs. We applied a cut, keeping only those cookies associated with at least 20 accesses. This left 96,470 distinct cookies. The distribution for the visits associated with each cookie is shown in Figure 1.9 below.

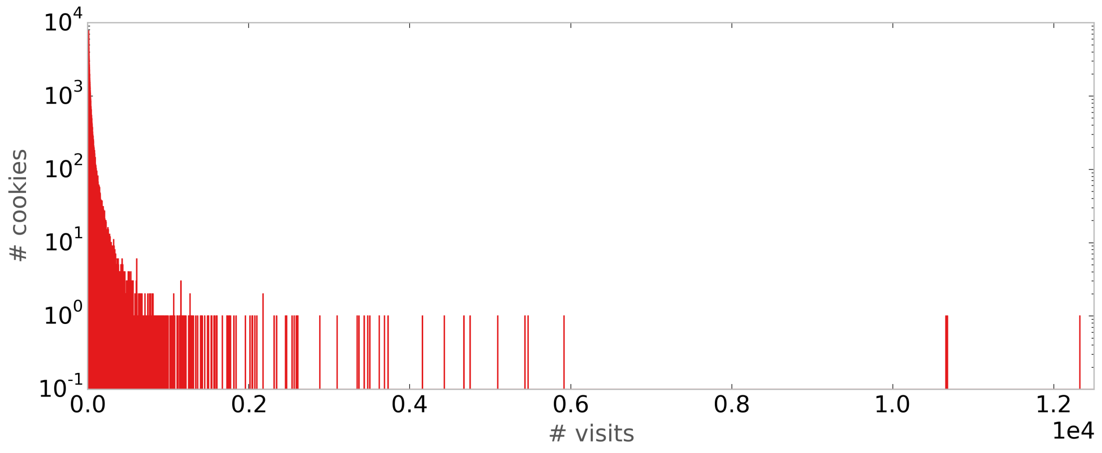


Figure 1.9: The observed number of accesses associated with each cookie after applying a cut of at least 20 accesses.

Just as with the authorship, there is a very long tail in the observed access counts for each cookie.

1.5.8 Pairwise Mutual Information of the Metadata

There is a fundamental question as to the utility of all of these various sources of metadata information. In order to probe how much information is shared between these different sources, we computed the mutual information between all pairs of the article ids, minor categorization, major categorization, authorship, word usage, and readership. The results are shown below in Table 1.1.

For the smaller sets, the mutual information was estimated by extrapolating the empirical estimates of the observed joint entropy distributions, as proposed in [94]. The starred numbers reflect the simpler, empirical mutual information estimated from the empirical joint entropy, without extrapolation, due to the size of the full joint distribution.

	art	minor	major	auth	words	readers
art	14.618	6.32(1)	1.669(9)	14.186(7)	2.563*	9.279*
minor		6.314(6)	1.652(9)	5.949(7)	0.618(6)	4.005(6)
major			1.665(5)	1.623(7)	0.168(5)	1.331(5)
auth				16.404(3)	2.213*	8.226*
words					10.3263(1)	X
readers						15.6125(5)

Table 1.1: Table of observed mutual informations for the arχiv corpus. This should give us some indication of what we can hope to learn from each source of metadata in relation to the article identities. The mutual informations were estimated by extrapolating values for the joint entropies measured empirically from different subsamples of the data, following the method proposed in [94]. The starred values are empirical estimates, where the subsampling procedure was too costly. Note that the values in this table should be symmetric across the diagonal. The value for the mutual information between words and readers proved too expensive to compute explicitly.

Mutual information measures the amount of information knowing either variable tells you about the other. In terms of the entropies of a pair of variables:

$$I(X, Y) = H(X) + H(Y) - H(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|A) , \tag{1.41}$$

which can be represented schematically, as in Figure 1.10.

All told, we can see that there is complete overlap between the article identity and the categorizations. This is to be expected, as we suspect that knowing which article we are dealing with is enough to uniquely determine which category it is in. The mutual information between the article identity and authorship is large, but not complete. This suggests that for instance, knowing authors does not completely determine which article we are dealing with, because in fact, some articles have exactly the same set of authors. Similarly, categorizations and authorship have large, but not saturating mutual information.

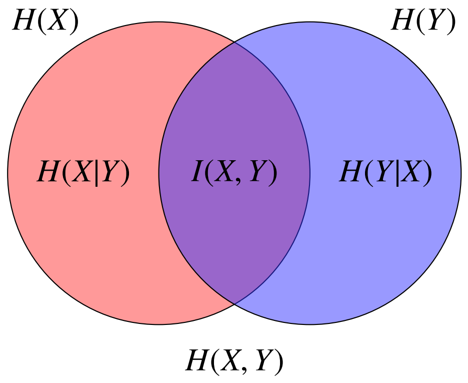


Figure 1.10: A graphical representation of the relationship between the mutual information between two variables: $I(X, Y)$, and the various entropies and conditional entropies.

Perhaps most interesting is the mutual information between the word distribution and the other quantities. The word distribution doesn't have very large overlap with any of the other sources of information. Given the complexity of word usage, this might be expected. Notice however that the existing overlaps can have a very large effect. The 2.6 bits of overlap with the article identity means that conditional on a choice of article, the entropy in the conditional word distribution drops significantly, from 10.3 bits to 7.7 bits. This corresponds to a drop in the perplexity from $2^{10.3} \sim 1300$ to $2^{7.7} \sim 210$. If, instead of estimating populations of equal usage, we assumed that the vocabulary followed a Zipf distribution exactly, 10.3 bits corresponds to a vocabulary of $\sim$ 24 000 words, while 7.7 bits corresponds to only 1 300 words, which corresponds roughly to the length of an article. The mutual information between authors and words is similar, which we could see as an extension of the observation that 75% of our authors have only a single article.

The mutual information between the categorizations and word usage is quite low. Naively, this might spell doom for our attempt to learn the categories based just on the word usage, but in reality, even very small mutual information can allow for a very high degree of classification. For instance, imagine that there was just a single word that occurred once per article, which only occurred in a given category. This word would contribute negligibly to our mutual information, but would allow a perfectly accurate classification scheme. A better way to find such telling words is to look at the *pointwise mutual information* or PMI, defined as:

$$\mathrm{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} \,, \tag{1.42}$$

in terms of which the mutual information is just the expected pointwise mutual information over the joint distribution. Words that have large PMIs serve as a strong indicator that a given word should appear in a given class. We've shown the ten words with the largest PMI with respect to the major categories below in Table 1.2, neglecting any word that is in the top 200 words in our vocabulary, so as to avoid stop words.

physics	temperature, spin, phase, stars, particle, region, phys_rev_lett, physics, electron, particles
math	there_exists, l2, rn, prove, suppose, remark, corollary, x0, implies, algebra
cs	user, network, performance, node, information, based, users, nodes, proposed, graph
stat	estimation, methods, estimator, prior, statistics, posterior, regression, variance, test, sample
q-bio	cell, population, cells, species, gene, genes, protein, neurons, network, individuals
q-fin	market, price, risk, financial, volatility, prices, portfolio, trading, st, finance

Table 1.2: Words with the largest PMI for each major category that are not amongst the top 200 words.

The pointwise mutual information does a great job at picking out keywords for each of our major categories. If an article discusses users and nodes on some graph, it is likely a cs article, while the appearance of markets, price and risk is a sure sign of a quantitative finance article. Math articles make extensive use of L^2 (l2), $\mathbb{R}^N$ (rn), and x_0 (x0).

1.6 arχiv Word Vectors

`word2vec` and similar word vector embedding methods have become very popular lately. This is largely attributable to the success they have demonstrated across many different Natural Language Processing tasks, including the syllogism tasks they first considered [62, 78, 53, 63], as well as more traditional tasks, such as sentiment analysis [78] and machine translation [60]. Later, in chapter 2 we will demonstrate their ability to power a state of the art segmentation algorithm.

Most of these works have focussed on issues of encoding semantic and syntactic properties of common English usage, reflected in the choice of datasets used to train the methods. Most of the datasets included traditional sources of written English, including dumps of Wikipedia articles, the Google book corpus, or larger corpora created from crawling the web as a whole.

The first question was whether these methods would work in the context of a technical corpus such as the arχiv . Scientific literature is distinct from common written English, both in its use of specialized terms, and in its overall style generally.

1.6.1 Methods

To investigate, we trained three different word vector models, in each case learning a 200 dimensional representation for each of the 133,419 'words' in our variable *n*-gram vocabulary that resulted after the preprocessing described above.

`word2vec` skip-gram hierarchical softmax (hs) - Using the supplied `word2vec` code [1], we trained an instance of the skip-gram hierarchical softmax model, with a window size of 10, and the subsampling set to a threshold of 10^{-5}. This does a single pass over the corpus, with an scheduled learning rate.

`word2vec` skip-gram negative sampling (ns) - The next model was the skip-gram negative sampling method, with 5 negative samples generated for each word, again a window of 10 and the same subsampling threshold.

`GloVe` (g) - Lastly, we trained an instance of `GloVe` , using the released code [2], with the default window size of 15, and 25 iterations of Adagrad [21] stochastic gradient descent.

1.6.2 Nearest Neighbors

After training the various word vector methods, we can get a sense of their utility by observing nearest neighbor relationships. Similarity is measured by the cosine similarity between the word vectors, a measure of the angle between the two word vectors. To make better connection with the notion of distance, this cosine similarity is turned into a distance by subtracting the cosine of the angle between the vectors from 1:

$$d_{\cos}(\boldsymbol{v}, \boldsymbol{w}) = 1 - \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|} , \tag{1.43}$$

so that our resulting distance measure runs from 0 if the vectors are parallel, to 2 if they are anti-parallel.

[1] `https://code.google.com/p/word2vec/`
[2] `http://nlp.stanford.edu/projects/glove/`

For instance, we could ask, which words are closest to the word "blue"? The results are shown in Table 1.3 below. With the exception of the tenth word suggested by the `GloVe` method ('indicated'), all of the nearest words are themselves colors.

blue					
hs		ns		g	
red	0.13	red	0.10	red	0.05
black	0.13	yellow	0.15	black	0.18
grey	0.15	cyan	0.15	green	0.22
orange	0.16	black	0.16	yellow	0.23
yellow	0.16	purple	0.16	orange	0.25
purple	0.17	grey	0.17	magenta	0.28
cyan	0.17	magenta	0.18	cyan	0.29
pink	0.20	light_blue	0.19	red_green	0.31
magenta	0.21	orange	0.19	purple	0.31
green	0.23	violet	0.19	indicated	0.32

Table 1.3: Nearest word vectors to "blue" and their resulting cosine distance for the three different word vector methods. (hs) is `word2vec` skip-gram hierarchical softmax, (ns) is `word2vec` skip-gram negative sampling, and (g) is the `GloVe` method.

While the resulting cosine distances might not seem impressive, remember that these word vectors are each 200 dimensional, so that if they were all distributed randomly, the expected cosine distance would be 1, while the standard deviation of the cosine distance would be roughly $1/\sqrt{D} \sim 0.07$. More precisely, the cosines will be distributed as:

$$P(\cos\theta) \propto \left(1 - \cos^2\theta\right)^{\frac{D-3}{2}} d\cos\theta\,, \tag{1.44}$$

so that our cosine distance, $d = 1 - \cos\theta$, will be distributed as:

$$P(d) \propto \left(1 - (1-d)^2\right)^{\frac{D-3}{2}}\ . \tag{1.45}$$

For 200 dimensional vectors, as we've trained here, this means we have a vanishing probability that the cosine distance is anything less than around 0.5. Below in Figure 1.11, we've shown the probability distribution for the cosine distance of two random isotropic unit vectors in 200 dimensions, as well as the cumulative distribution. Notice that below 0.5, our cumulative distribution is below machine precision for double precision floating point numbers.

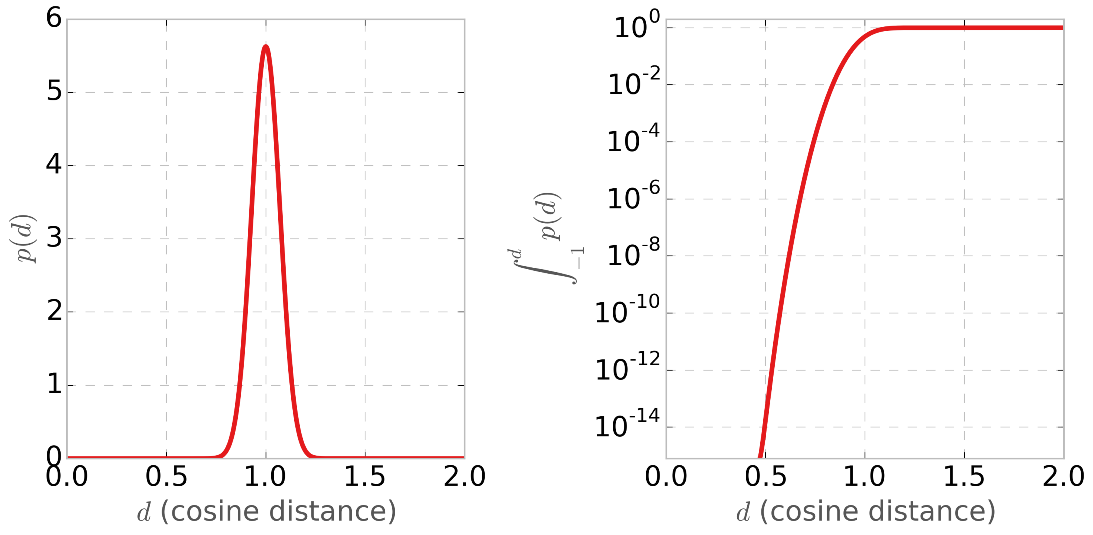


Figure 1.11: The probability distribution for the cosine distance for two random isotropic 200 dimensional vectors, on the left, along with the cumulative probability distribution on the right.

Probing scientific terms is more interesting. Consider "electron". In Table 1.4, we see that for all three methods, the word nearest to "electron" is "electrons", which should make sense. The presence of adjectives we might expect to be used to describe electrons in articles, such as 'photoexcited' and 'spin-polarized', is also of note. Words that frequently appear next to our target word will be expected to point in similar directions. However, we also see the presence of similar nouns such as 'ion', 'electron hole pairs' or 'hole'. In this sense, the vector embedding has learned that 'hole' and 'electron', while they might not necessarily appear near each other very often, might have very similar contexts usually. They are synonyms, in a sense, and our vector model learns that they should point in a similar direction, not necessarily directly, but because they are both forced to point in a direction similar to all of the words used to describe them.

electron					
hs		ns		g	
electrons	0.19	electrons	0.16	electrons	0.20
electron_hole_pairs	0.32	electronhole	0.24	hole	0.31
photoexcited	0.33	electron_hole_pairs	0.25	ion	0.32
excitation	0.33	charge_carriers	0.28	proton	0.37
ion	0.33	hole	0.29	excitation	0.38
spin_polarized	0.34	photoexcited	0.29	photon	0.38
spindependent	0.34	photo_excited	0.29	energy	0.39
electronhole	0.34	ion	0.29	atom	0.39
electron_hole	0.34	electron_tunneling	0.29	electronic	0.40
energy	0.35	photo_excitation	0.30	muon	0.40

Table 1.4: Nearest word vectors to "electron" and their resulting cosine distance for the three different word vector methods.

We can see more evidence of this synonym type behavior by looking at the nearest neighbors to 'singular_value_decomposition_svd', below in Table 1.5. Here we notice the presence of 'eigen_decomposition', 'choleksy_decomposition', 'whitening', 'principal_component_analysis_pca', 'truncated_svd', 'schur_complement', 'pseudo_inverse', 'gi_ica' (for Gaussian Invariant Independent Component Analysis [111]), and 'tucker_decomposition' (a form of tensor SVD [105]).

singular_value_decomposition_svd					
hs		ns		g	
orthonormal_columns	0.37	eigendecomposition	0.27	eigen_decomposition	0.42
sparsifying_basis	0.37	matrix	0.30	eigendecomposition	0.44
eigen_decomposition	0.37	gi_ica	0.31	svd	0.44
rank_deficient	0.38	truncated_svd	0.31	randomized_sampling	0.47
sparsified	0.38	orthonormal_columns	0.32	cholesky_decomposition	0.48
tucker_decomposition	0.40	lowrank	0.32	principal_component_analysis_pca	0.48
right_singular_vectors	0.40	whitening	0.32	pseudo_inverse	0.50
blockwise	0.40	sos_boosting	0.33	schur_complement	0.50
efficiently_computed	0.40	block_circulant	0.33	galerkin_approximation	0.51
hard_thresholding	0.41	cholesky_decomposition	0.33	truncated_svd	0.52

Table 1.5: Nearest word vectors to "singular_value_decomposition_svd" and their resulting cosine distance for the three different word vector methods.

In the previous two examples, the `GloVe` method arguably has more of these synonyms in its nearest neighbor list. This may be due to the fact that in the `GloVe` method we simultaneously learn representations for both words and contexts as vectors, and the final representations we take for a given word is the sum of its word vector and context vector. As pointed out by Levy and Goldberg [55], this has an effect on the similarity metric of adding cross terms:

$$\boldsymbol{w}_1 \cdot \boldsymbol{w}_2 \to (\boldsymbol{w}_1 + \boldsymbol{c}_1) \cdot (\boldsymbol{w}_2 + \boldsymbol{c}_1) = \boldsymbol{w}_1 \cdot \boldsymbol{w}_2 + \boldsymbol{c}_1 \cdot \boldsymbol{c}_2 + \boldsymbol{w}_1 \cdot \boldsymbol{c}_2 + \boldsymbol{c}_1 \cdot \boldsymbol{w}_2 \tag{1.46}$$

Amusingly, the word vector models seem to learn about naming habits in different countries:

john		dmitri		wang		stefano		pierre	
david	0.16	misha	0.27	zhou	0.08	roberto	0.21	jean	0.22
michael	0.17	nikshych	0.29	liu	0.08	alessandro	0.24	frederic	0.24
robert	0.17	kirill	0.30	chen	0.08	emanuele	0.25	alain	0.24
william	0.18	graeme	0.30	zhang	0.09	michele	0.25	yves	0.25
daniel	0.21	pavel	0.31	yang	0.10	giuseppe	0.26	christophe	0.25
chris	0.21	alexei	0.31	huang	0.10	nicola	0.26	olivier	0.26
eric	0.22	mikhail	0.31	zhao	0.12	paolo	0.26	sylvain	0.27
philip	0.22	ross_street	0.31	zhu	0.12	francesco	0.27	claude	0.28
jonathan	0.22	mircea	0.31	feng	0.12	pietro	0.27	jean_philippe	0.29
stephen	0.22	higson	0.31	luo	0.13	laura	0.27	stephane	0.29

Table 1.6: Nearest word vectors to various first names, using the hs method.

This is presumably due the names that appear in references at the ends of articles, as well as the lists of authors at their start. Names like John and David can be expected both to appear near one another, due to the tendency of authors with English names to write articles with other authors with English names, and have a degree of replaceability, due to their proximity to certain words like 'smith'.

1.6.3 Analogies

Besides looking at nearest neighbors, we can attempt to find syllogisms, as was done in the original `word2vec` paper [59]. Here, we rely on the ability of the word vector methods to encode more than nearest neighbor relationships. For instance, we know that force is to torque as momentum is to angular momentum, the latter pair being the angular version of the former. If the learned word vectors have learned a proper Euclidean embedding of the words and their relationships, we might expect that the vector for torque lies at a similar displacement from the vector for force as the vector for angular momentum does in relation to momentum.

We can probe this analogy by probing for words nearby:

$$\boldsymbol{v}_{\text{momentum}} + (\boldsymbol{v}_{\text{torque}} - \boldsymbol{v}_{\text{force}}) \ . \tag{1.47}$$

In direct analogy to the way we sought out nearest neighbors, we could look for the vector with the smallest cosine distance to the linear combination of word vectors describing the analogy.

Alternatively, as suggested in [56], instead of doing a linear probe, we could do a multiplicative probe, seeking the word that maximizes:

$$\arg\max_{\boldsymbol{w}} \frac{\text{cosine}(\boldsymbol{v}_{\text{momentum}}, \boldsymbol{w}) \times \text{cosine}(\boldsymbol{v}_{\text{torque}}, \boldsymbol{w})}{\text{cosine}(\boldsymbol{v}_{\text{force}}, \boldsymbol{w}) + \epsilon} \ , \tag{1.48}$$

which they named the `3CosMul` method. In their experiments, this gave better analogy results.

Finally, if our unit vectors were all normalized, the cosine probe would be equivalent to:

$$\arg\max_{\boldsymbol{w}} \left|(\boldsymbol{w} - \boldsymbol{v}_{\text{momentum}}) + (\boldsymbol{v}_{\text{torque}} - \boldsymbol{v}_{\text{force}})\right|^2 \ . \tag{1.49}$$

In the Arora paper [3], it is mentioned that this Euclidean probe performs well even if the word vectors are not normalized first.

Below, in Table 1.7, we show all three query methods applied to all three word vector models for the analogy in question. For the corpus, all nine of these combinations find the correct answer, aside from the possible presence of the probe words themselves and their plurals. The presence of the probe words is natural. Since we are in a high dimensional space, the sum of two words will tend to have a high cosine similarity with either of the words in the sum, as an extension of the likelihood for any two words to be nearly orthogonal.

1.6.4 Projections

Besides a microscopic view of the word vector embeddings, we can attempt to get a global view, by visualizing all of the word vectors, projected down to two dimensions. We could train our word vector methods to learn a two dimensional representation directly, but previous work [62, 18, 78] has generally shown that larger dimensional representations with dimensions on the order of hundreds perform much better. The problem then becomes one of data visualization: how can we represent a collection of high dimensional vectors faithfully in low dimensions?

Instead of using a linear method such as PCA, or even traditional nonlinear methods [52], here we will represent our word vectors using *t*-SNE [110], and in particular, an implementation of Barnes-Hut-*t*-SNE [109], a fast algorithm for computing very nice low dimensional embeddings of high dimensional data.

hs		ns		g	
cosine(w, force − torque + momentum)					
momentum	0.28	momentum	0.26	momentum	0.32
torque	0.30	torque	0.33	angular_momentum	0.42
torques	0.43	angular_momentum	0.37	torque	0.43
angular_momentum	0.45	torques	0.41	orbital_angular_momentum	0.48
angular_velocity	0.49	helicity	0.43	total_angular_momentum	0.49
helicity	0.50	angular_velocity	0.44	momenta	0.50
transverse	0.51	chirality	0.45	helicity	0.53
charge	0.52	momenta	0.46	transverse_momentum	0.54
spin	0.52	spin	0.47	spin	0.56
azimuthal	0.52	antidamping_like	0.48	light_cone	0.57
berry_curvature	0.54	energymomentum	0.48	chirality	0.58
3CosMul					
torque	0.00	momentum	2.89	torque	0.00
momentum	0.00	torque	3.29	momentum	0.00
momenta	0.31	angular_momentum	3.38	momenta	0.28
angular_momentum	0.34	energymomentum	3.65	angular_momentum	0.31
torques	0.35	antidamping_like	3.65	transverse_momentum	0.40
helicity	0.39	torques	3.65	total_angular_momentum	0.42
momentum_conservation	0.40	angular_velocity	3.68	helicity	0.43
momentum_transfer	0.41	helicity	3.68	spin	0.43
total_angular_momentum	0.41	spin	3.68	orbital_angular_momentum	0.44
charge	0.43	momenta	3.68	energy	0.44
transverse	0.44	lab_frame	3.70	light_cone	0.47
$\|\| (\text{force - torque}) + (\text{momentum} - w) \|\|^2$					
momentum	2.47	momentum	2.89	momentum	5.85
torque	2.60	torque	3.29	angular_momentum	6.61
angular_momentum	3.02	angular_momentum	3.38	torque	6.69
torques	3.05	energymomentum	3.65	total_angular_momentum	6.86
charge	3.12	antidamping_like	3.65	orbital_angular_momentum	6.88
transverse	3.13	torques	3.65	dissipating	7.16
spin	3.15	angular_velocity	3.68	momenta	7.17
quantization_axis	3.17	helicity	3.68	imparted	7.21
antidamping_like	3.18	spin	3.68	helicity	7.31
angular_velocity	3.18	momenta	3.68	lepton_pair	7.33
time_reversal_breaking	3.19	lab_frame	3.70	komar	7.34

Table 1.7: The result of applying three different probe methods to the three different word vector representations trained on the arχiv, seeking to find the answer to the analogy: force:torque::momentum:?. Notice that aside from the presence of the probe words themselves and their plurals, all nine instances find the correct answer: angular momentum.

t-SNE attempts to learn a two dimensional vector for each point (y_i), and defines a symmetric probability distribution for the neighborship of those points:

$$q_{ij} = \frac{\left(1 + \|\boldsymbol{y}_i + \boldsymbol{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\boldsymbol{y}_k + \boldsymbol{y}_l\|^2\right)^{-1}} , \tag{1.50}$$

given by a Student's *t*-distribution, such that this low dimensional stochastic map has a minimal Kullback-Liebler divergence with the corresponding high-dimensional symmetric probability distribution for neighborship:

$$C = KL(P\|Q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{1.51}$$

where we take the corresponding high-dimensional symmetric distribution to be a symmetrized version of a simple Gaussian model:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{1.52}$$

$$p_{j|i} = \frac{\exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|\boldsymbol{x}_k - \boldsymbol{x}_i\|^2 / 2\sigma_i^2\right)} \tag{1.53}$$

where y_i is the high dimensional vector in question. The variances are chosen so as to try to ensure a constant perplexity, chosen by the user (perplexity $\sim 2^{H(p_{j|i})}$).

While seemingly complicated, this cost has a very nice gradient [110]:

$$\frac{\partial C}{\partial \boldsymbol{y}_i} = 4 \sum_j \left(p_{ij} - q_{ij}\right) \frac{\boldsymbol{y}_i - \boldsymbol{y}_j}{1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2} \tag{1.54}$$

This qualitatively has the form of a collection of nonlinear springs, pulling and pushing each of our two dimensional points. The nonlinearity ensures that very far away points tend to have reduced influence on the position of a particular point, and the effective spring constants ($p_{ij} - q_{ij}$) mark the disagreement between our notion of distance in the low dimensional embedding versus the high dimensional data. *t*-SNE has shown the capability of creating very powerful visualizations of real world datasets, often preserving a remarkable level of both local and global structure [110].

Below, in Figures 1.12, 1.13, and 1.14, we show the *t*-SNE embeddings found for the word vectors trained by the hierarchical softmax, negative sampling, and `GloVe` methods respectfully, with a perplexity set to 40. This means that the high dimensional distances roughly consider the distances to the 40 nearest neighbors. In order to better reflect the cosine distances in the high dimensional vectors, here we first normalized all of the vectors, since for normalized vectors, the euclidean distance is proportional to the cosine distance:

$$||\hat{\boldsymbol{v}} - \hat{\boldsymbol{w}}||^2 = 2\,(1 - \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{w}}) \tag{1.55}$$

In order to help highlight the structure inherent in the embedding, we colored each word according to the category with which it shares the largest pointwise mutual information (PMI). The color scheme is the same as used earlier in Figure 1.7. Notice that even though the word vector embedding methods have no knowledge of the categorizations, in fact, they have no knowledge of any of the metadata, including the article associations, the learned word vectors clearly respect the notion of categories. Clearly visible are regions where physics (green), math (blue), cs (red), stat (yellow), q-bio (orange), and q-fin (purple) words collect. Looking closer, you can see that even within the large regions, the words tend to arrange themselves into regions with homogeneous shades of each color, corresponding to words with the greatest pointwise mutual information consisting of a single minor category.

Even parts of the embedding that at first glance appear to be a muddled collection of different colored dots, upon closer inspection, reveal their own structured layout. Consider the separated mass of points in the northwest extreme of the hs embedding. At first it appears to be a black smudge, but if you zoom in, you can see that that smudge itself shows a colored gradient, with blue dots

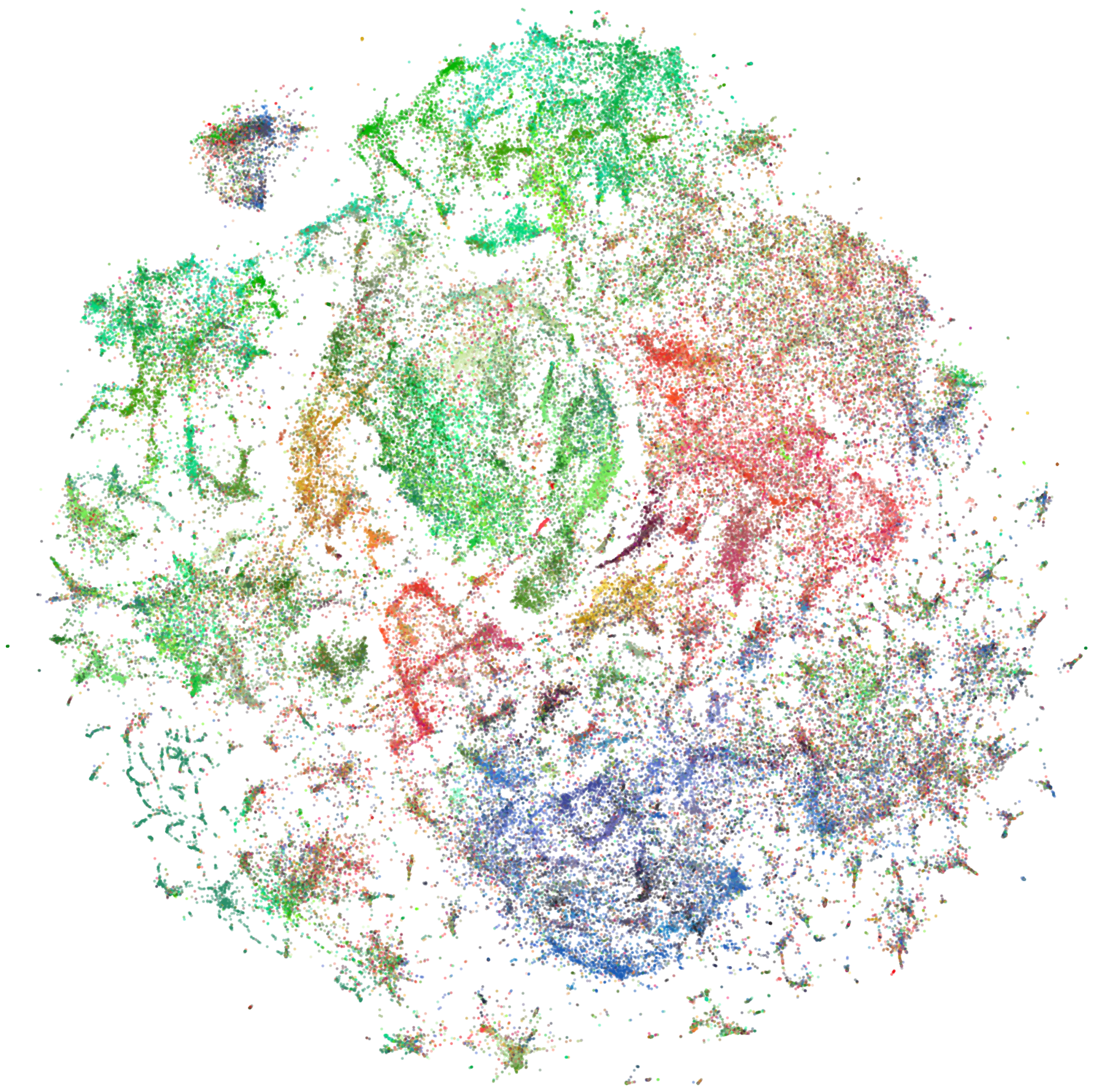


Figure 1.12: *t*-SNE projection of `word2vec` skip-gram hierarchical softmax word vectors, trained on the arχiv dataset. The minor category each word has the largest pointwise mutual information with determines the color used to represent the word. The color scheme is the same as in Figure 1.7.

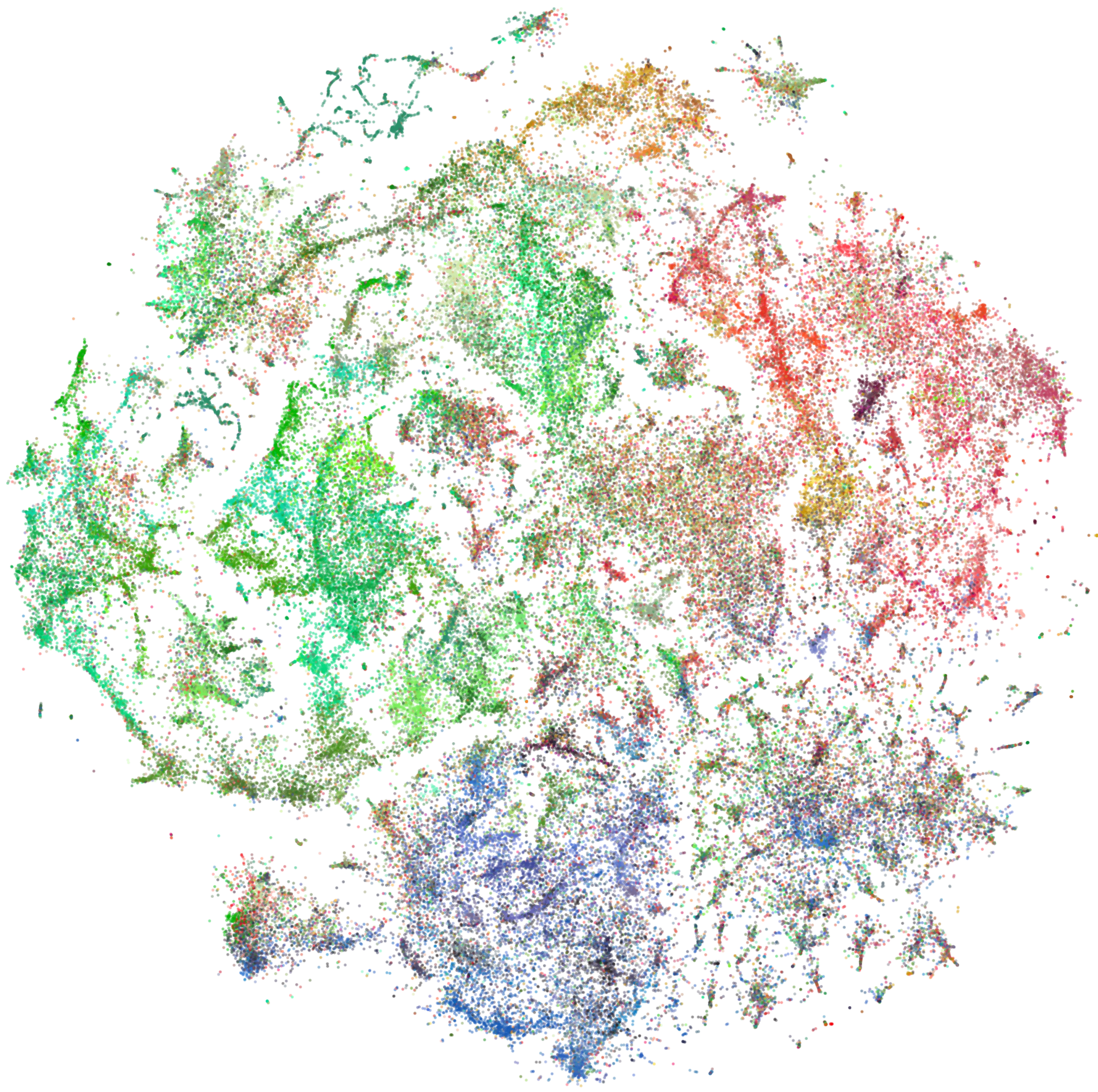


Figure 1.13: *t*-SNE projection of `word2vec` skip-gram negative sampling word vectors.

concentrated towards the right, a red stripe running through the center, and a mostly green left top. In fact, this island is made up of French words, themselves separating nicely into distinct categories and well separated from the bulk of the English words.

Perhaps even more remarkable is the remarkable similarity between the different projections. The three projections are from three distinct objectives. Each

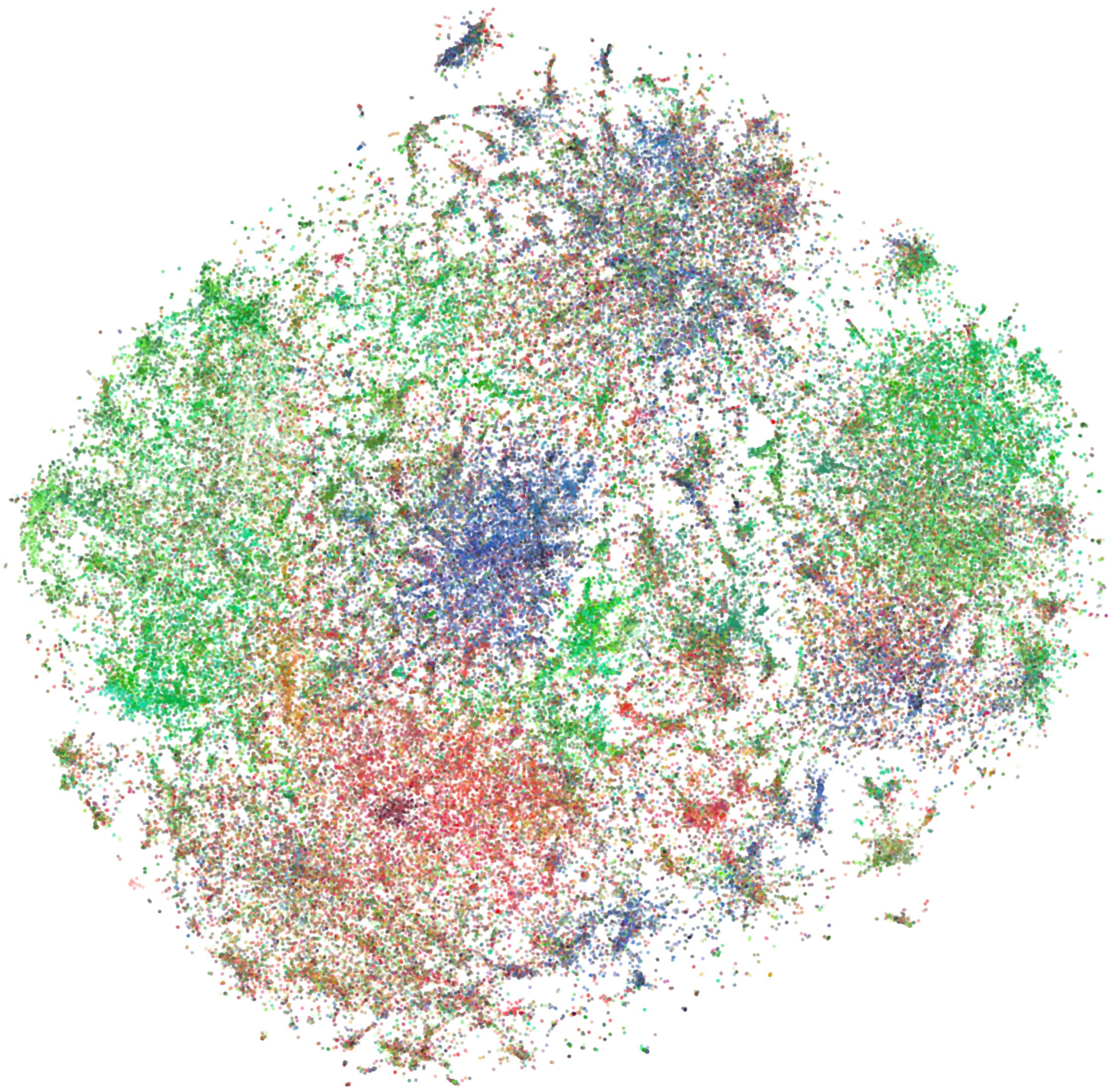


Figure 1.14: *t*-SNE projection of `GloVe` word vectors.

projection is an instance of a distinct stochastic embedding, with its own random initialization. Granted, we had to reflect the resulting embeddings to get them to agree so remarkably. The projection is unique up to an overall translation or reflection into the 2D plane. In particular, the hs and ns embeddings could easily be considered siblings. The overall structure is basically the same. Looking closer, you can see the presence of the same sorts of substructures in both as well. The left half is all (green) physics articles, the bottom consists of (blue) math, and the (red) cs sit north of the math. The (yellow) stat articles are intermediate between cs and math. Our isolated smudge of French terms has moved into the southwest corner. For substructures with strong associations with one another but weaker association with the rest of the collection, we might expect this kind of larger orientational change. But the overall similarity ought to serve as a hint that the structure we observe in these embeddings is real, and present ultimately in the word cooccurrence statistics itself.

Compared with the `word2vec` embeddings, the `GloVe` embedding leaves much to be desired. While there is clearly an overall rough segmentation into major categories present, and some hint of minor category segmentation, the embedding has a more random, muddled appearance to it. This may be a sign that the word vectors themselves are not as reasonable, or could be an artifact of the fact that the `GloVe` vectors are the sum of both the word and context vectors learned during the training process. Perhaps the muddled quality is due to this struggle to try to project down what is essentially the sum of two distinct high dimensional representations.

1.7 Vector Representations of Articles

Given that we have effective vector representations for the words appearing on the arχiv, the real question is whether we can build similar vector representations at the article, author, category and/or reader level. Those vector representations could then be used to power interesting applications. Our main contribution has been to try various methods on the arχiv and compare their results.

1.7.1 Aggregation Techniques

The first strategy for building representations of articles would be to build them directly out of the representations of words we trained above. These aggregation techniques are cheap computationally, and efficient even as new articles are submitted, as word usage and vocabulary are more stable over time. We still have various choices for how exactly to aggregate the words.

Bag of Words

In classic information retrieval, a document is considered as a *bag of words*, that is, each article is represented by the number of times each word is used in that article. This is intrinsically the representation used for Naive Bayes classification, which does a remarkable job at categorizing articles into their respective categories. At present, Naive Bayes is used by the system classifier to help aid the arχiv editors in deciding whether articles need to be recategorized when they are submitted.

Representing articles as the bag of their words naturally destroys any information encoded by the word order. But, their utility seems to signify the fact that a great deal of information is encoded in the unigram counts in a document. Another potential downside is the large dimensionality of the representation. Each article is represented by a vector that is as large as the vocabulary, albeit sparse. This could potentially destroy the performance of bag-of-word based algorithms, but the inherent sparsity, if utilized, means that the methods can be very competitive computationally.

Thought of in terms of a large data matrix, this amounts to representing our corpus as the matrix D_{ai}, where each row (a) represents a particular article and each column (i) represents a particular word. The matrix is populated with the word counts inside each article.

$$D_{ai}^{\mathrm{BOW}} = \mathrm{count}_{a_i}(w_i) \tag{1.56}$$

Each article is therefore represented by a sparse V-dimensional vector $\boldsymbol{D}_a$, read off as the rows of this matrix.

Random Projection

Since the bag of words representation is inherently sparse, it suggests that using the full dimensionality of the representation is wasteful. In fact, quite generally, it is known that high dimensional data can often be compressed down to a lower dimension without losing much fidelity. This is powered by the *Johnson-Lindenstraus lemma* [43], which states that for a finite collection of points in high dimensions, for any choice of tolerance, there exists a random projection of smaller dimension that preserves all pairwise distances. The lemma often gives a very high dimension, in practice it is observed that much smaller dimensional random projections can preserve a great deal of the structure in large sparse datasets. This amounts to multiplying our word count matrix by a random matrix, usually taken to be populated by independent Gaussian entries.

$$D_{ak}^{\mathrm{RP}} = \sum_j D_{aj}\Omega_{jk} \tag{1.57}$$

This turns the very large dimensional sparse representation of each article into a dense, moderate dimensional representation.

Weighted Averaging

The previous two methods have not yet utilized our nice representations at the word level. If those word representations are meaningful, we ought to be able to use them to power our formation of article representations as vectors. In the crudest form, we could represent an article as the sum of all of the vectors representing each of the words in the document.

If we treat all of our word vectors as the matrix: W_{wk}, this amounts to

$$D^{\mathrm{AV}}_{ak} = \sum_w D_{aw} W_{wk} \tag{1.58}$$

You'll notice that this takes the same form as our random projection, although this time we hope to benefit from the structure of the word vectors. Two similar words, such as `event_horizon` or `black_hole_horizon` will correspond to similar vectors, and so will contribute similarly to this sum.

We could be concerned however that the *signal* from the word vectors as to the topic of the article might get washed out from this averaging procedure, because summing many vectors might destroy the information contained in their directions. In fact we often assume that summing many vectors all distributed more or less isotropically in space cancels out, a property intrinsic in the path integral formulation of quantum mechanics for instance.

There are two things that save us here. The first is that the words in a given article should not themselves be completely random. If the directions in the vector space of words encode some kind of meaningful semantic identities, and an article is in some way an organized collection of words with some central theme or topic, it is precisely the surviving components of this sum that should be the most meaningful for describing the content of the article. The second property that helps us here is that our word vector representations tend to have appreciable dimensionality, on the order of a few hundred dimensions. In high dimensions, the sum of two vectors can retain a large cosine similarity with both of the vectors in the sum. This is clearest to see if you consider two vectors $\boldsymbol{a}, \boldsymbol{b}$ and consider $\boldsymbol{a} \cdot (\boldsymbol{a} + \boldsymbol{b}) = a^2 + ab \cos\theta$ and remember that in high dimensional spaces, any two random vectors are nearly orthogonal ($\cos\theta \sim 0$). (See Figure 1.11 and related discussion)

Alternatively, one can view the word vectors as defining a low rank approximation to the true word-word metric. Consider a computation involving the dot product of the bag of words representation of two articles, a and b. In our bag of words representation, this dot product is simply $\boldsymbol{a}_a \cdot \boldsymbol{a}_b = \sum_i D^{\text{BOW}}_{ai} D^{\text{BOW}}_{bi}$. If we instead consider the dot product between two of our average-word article vectors:

$$\boldsymbol{a}_a \cdot \boldsymbol{a}_b = \sum_k D^{\text{AV}}_{ak} D^{\text{AV}}_{bk} \tag{1.59}$$

$$= \sum_k \left(\sum_i D^{\text{BOW}}_{ai} W_{ik} \right) \left(\sum_j D^{\text{BOW}}_{bj} W_{jk} \right) \tag{1.60}$$

$$= \sum_{ij} D^{\text{BOW}}_{ai} \left(\sum_k W_{ik} W_{jk} \right) D^{\text{BOW}}_{bj} \tag{1.61}$$

$$= \sum_{ij} D^{\text{BOW}}_{ai} g_{ij} D^{\text{BOW}}_{bj} \ , \tag{1.62}$$

we find that utilizing the average of the word vectors amounts to using a non-trivial metric $g_{ij} = \sum_k W_{ik} W_{jk}$ on the bag of words representations. This metric being represented by a rank K symmetric decomposition, where K is the dimensionality of our word vectors, just as we motivated in the introduction.

In either case, we could hope to generalize our vector average by weighting the contribution of different words. In particular, we could adopt the common strategy in information retrieval of utilizing the *tfidf* weights, forming:

$$D_{ak}^{\mathrm{WAV}} = \sum_w \mathrm{tfidf}_w W_{wk} \; . \tag{1.63}$$

The tfidf weight is the term frequency times the inverse document frequency, typically taken on a log scale.

$$\mathrm{tfidf}_w = \log\left(1 + \mathrm{count}(w)\right) \log \frac{A}{\mathrm{arts}(w)} \tag{1.64}$$

Here A is the total number of articles, and arts(w) tells us how many articles the word appears in at least once.

By weighting the contribution of the word vectors in this way, we hope to mitigate the effect of common English words, such as stop words that make up the brunt of the actual article, but tell us little about the scientific topic under discussion.

The addition of the tfidf weights is meant to tamper the contribution of frequently occurring words. Taken to the extreme, we could simply allow each word to contribute only once to our article vector, and represent the article just as the sum of the word vectors for each word present. This is akin to the representation used when one combines articles with Jaccard similarity. In particular:

$$D_{ak}^{\mathrm{B}} = \sum_w \delta(D_{aw} > 0) W_{wk} \; . \tag{1.65}$$

The δ here denotes a function that returns 1 if the argument is true, and zero if it is false. The performance of this method in relation to the other two can help us sort out how much the tfidf weighting benefits from its reduction in the contribution of frequent words, versus its meaningful reflection of the word usage.

1.7.2 Trained Methods

Alternatively, we could learn article vectors as the solution to some form of objective, just as we did for the word vectors themselves. There are three natural architectures for learning article vectors, each corresponding to one of the three techniques we considered for the word vectors.

DBOW Method

The first trained method for learning article vectors is the technique explored in [50, 18], called the *Distributed Bag of Words* Model, or DBOW. In this method, an article is represented by the vector that does the best job at predicting each of the words in the article, according to a log linear model. That is, we maximize the log likelihood of observing all of the words in an article:

$$\mathcal{L} = \sum_{w \in a} \log P(w|a) \tag{1.66}$$

with

$$P(w|a) = \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{a})}{\sum_i \exp(\boldsymbol{w}_i \cdot \boldsymbol{a})} = \frac{1}{Z_a} \exp(\boldsymbol{w} \cdot \boldsymbol{a}) \,. \tag{1.67}$$

This is essentially the skip-gram model of `word2vec` but applied to the entire article. Notice that this still does not utilize any of the word order information, being based purely upon the total occurrence of each word in the article. A schematic of this architecture is shown below in Figure 1.15.

In the second article [18], Dai et al. note that the performance of the method was improved if they simultaneously learned both word and article representations. In practice, this means that they trained this method with hierarchical softmax, and used their Huffman encoded tree to predict not only the words in

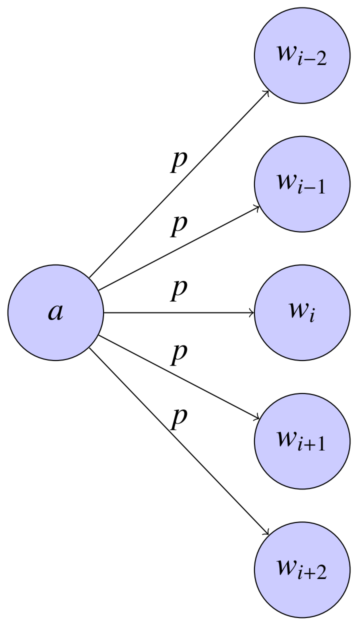


Figure 1.15: The *distributed bag of words* (DBOW) model attempts to learn vector representations of articles that do well to predict each of the words in that article, through a classifier.

the article, but all of the neighboring words for each word in the article. They argue that this acted as a regularizer to their model, and made the word representations in the hierarchical tree more meaningful.

During training, we allow all of our vectors to adjust: the articles, words, and inner nodes of our hierarchical softmax classifier. If we then wanted to learn article vectors for new articles as they came in, this would require an inference step where we freeze the word and inner node vectors, and instead just learn the article representation. It is instructive to take a look at this inference objective. The total log likelihood for observing all of the words in our article, given the article vector is:

$$\mathcal{U}_j = \sum_{i \in \text{article}} \boldsymbol{w}_i \cdot \boldsymbol{a}_j - N_j \log Z_j \,, \tag{1.68}$$

where N_j is the number of words in article j, and Z_j is the partition function for our article

$$Z_j = \sum_i \exp\left(\boldsymbol{w}_i \cdot \boldsymbol{a}_j\right) \; . \tag{1.69}$$

We can re-express this objective in terms of our article-word data matrix D, and matrix of word vectors W:

$$\mathcal{U}_| = \sum_k \left(\sum_i D^{\text{BOW}}_{ji} W_{ik} \right) a_k - N_j \log Z_j \tag{1.70}$$

$$= \sum_k D^{\text{AV}}_{jk} a_k - N_j \log Z_j \; , \tag{1.71}$$

and we notice that the DBOW method attempts to make each article vector lie as close as possible to the average word vector for the article (D^{AV}), but crucially, with the additional goal of having the smallest partition function possible, meaning the article vector lies as perpendicular to the entire vocabulary as possible. This objective has the form of an entropy $S = U/T + \log Z$, where the first term is our energy term, and the second is our free energy.

DM+ Method

Just as there were two natural architectures for `word2vec` , the CBOW and skip-gram architectures, there are two natural architectures for attempting to learn article representations. The first of the complementary CBOW style article methods is the DM+ method, discussed in [50]. Here, as depicted in Figure 1.16, we treat each article vector as though it were a piece of missing linear context for each word. The article vector is summed along with the nearby words to try to form the input to the word classifier. This treats the representation of the article almost as if it were the pieces left unsaid at every position of the article, which we could imagine would help aid in distinguishing the resolution of some common synonyms. While mention of a spin $1/2$ particle in a condensed matter article might almost always refer to ‘electrons‘, similar mention of spin 1/2 particles might more commonly be referred to as ‘leptons‘ in a high energy article.

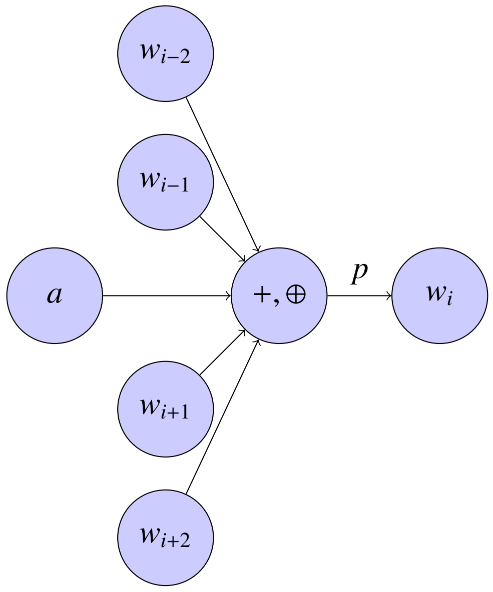


Figure 1.16: The *distributed memory* model (DM) attempts to learn vector representations of articles that behave as additional linear context for each word observed in the corpus.

DM Method

The preceding methods took some advantage of word vectors, but failed to take advantage of any of the ordering of the words in an article, except indirectly through their effect on the word vectors themselves. We could hope to form more informed article representations by making the vector representing an article take part in our actual model for the corpus itself, at the word level.

This constitutes the *distributed memory* model as implemented in [50]. Here we use the same architecture depicted in Figure 1.16, but instead of summing our article vector and local word vectors, we concatenate them, forming a larger dimensional input representation for the current word classifier.

This is, strictly speaking, a significantly more powerful method than the DM+ method itself, as it is entirely within the domain of the method to learn a concatenated representation that simply mimics the addition enforced in the DM+ method. Whether the method actually achieves better performance, however, will come down to the practical question of whether we can effectively learn the larger number of parameters with stochastic gradient descent acting on a fixed sized corpus.

Multiple Domain Global Vectors (MDGloVe)

Our last method, and one that hasn't yet appeared in literature, is to attempt to generalize the global `GloVe` method to attempt to learn article representations. A straightforward generalization would be to attempt to simultaneously learn a low rank factorization not only of the word-word cooccurrence matrix X_{ij}, but also learn a low rank factorization of the article-word cooccurrence matrix A_{ij}:

$$\mathcal{U} = \gamma_X \sum_{ij} f_X(X_{ij}) \left(\boldsymbol{w}_i \cdot \boldsymbol{c}_j + b_i^{(w)} + b_j^{(c)} - \log X_{ij} \right)^2 + \gamma_A \sum_{aj} f_A(A_{ij}) \left(\boldsymbol{a}_i \cdot \boldsymbol{w}_j + b_i^{(a)} + b_j^{(w')} - \log A_{aj} \right)^2 \tag{1.72}$$

Here $\boldsymbol{w}_i$ are the vectors representing words, $\boldsymbol{c}_i$ are the vectors representing context words, and $\boldsymbol{a}_i$ are the vectors representing articles. X_{ij} is the matrix of word-word cooccurrence counts, while A_{aj} is the matrix of article-word cooccurrence counts. The word vectors ($\boldsymbol{w}_i$) appear in both matrix factorization objectives, which helps ensure that all of the learned vectors will live in the same space. Each vector gets its own bias, including two independent biases for the word vectors in each objective. Each objective is weighted, where

$$f(x) = \begin{cases} \left(\frac{x}{x_0}\right)^\alpha & x < x_0 \\ 1 & x > x_0 \end{cases}, \tag{1.73}$$

while in principle, we could imagine different choices for α and x_0 for each weighted factorization problem. Finally, each part in the objective could have an overall weight γ_X, γ_A, which could allow us to tune the objective towards better optima for one or the other of the objectives.

In practice, we found that using equally weighted objectives for both the article-word and word-context factorization problems, while it yielded nice word and context representations, did not offer very useful article vector representations. In the results we report below, we weighted the word-context factorization by a factor of 1/100 to help mitigate this problem as it seemed to give intermediate results between either extreme, but we admit that we did not investigate the settings for this hyperparameter exhaustively.

1.8 Evaluation

We have proposed many different methods for formulating article vectors. In order to compare these vectors, we'll need some evaluation procedures. We seek methods that will enable us to determine how useful the article vectors are at capturing interesting relationships between the articles.

1.8.1 Triplet Tasks

A simple method for evaluating the utility of the article vectors is to use them to answer a series of triplet tasks. Each task will be defined by a set of three article ids, (A, B, C). The task will compute whether B is more similar to A than C is, by evaluating:

$$\frac{\boldsymbol{a}_A \cdot \boldsymbol{a}_B}{|\boldsymbol{a}_A||\boldsymbol{a}_B|} < \frac{\boldsymbol{a}_A \cdot \boldsymbol{a}_C}{|\boldsymbol{a}_A||\boldsymbol{a}_C|} \, . \tag{1.74}$$

That is, computing the cosine similarity of A and B, and comparing that to the cosine similarity of A and C. We will create a series of these triplets, where the triplets are constructed so that on the whole, the B choices *ought* to be closer to A than the C choices are. Usually this will mean that we randomly select an article A, choose B according to some similarity we can compute from the metadata in the dataset and choose C to be a random article subject to some constraints.

For this work, each of our triplet tasks will consist of 20,000 triplets, each attempting to represent some special kind of relationship in the data. Some will try to capture category type relationships, both at the minor category and major category level, some will try to capture authorship information, some word usage, and some readership information.

The first set of triplets try to probe category type relationships between articles. A and B should be more like one another than C because they share a category label at some level, while C is chosen to be distinct at the category level from A.

catminmaj ($C\flat\sharp$) - B shares at least one minor category with A. C shares no major category with A.

catminmin ($C\flat\flat$) - B shares at least one minor category with A, C shares no minor category with A.

catmajmaj ($C\sharp\sharp$) - B shares at least one major category with A, C shares no major category with A.

$C\flat\flat$ ought to be the simplest of these tasks, because it should have the tightest relationship between A and B, while the most dissimilar relationship between A and C. The other triplets are to try to sort out any subtleties in the article representation in terms of the minor/major category labels.

Next we create a triplets that try to assess authorship:

auth (A) - B shares at least one author with A. C is random.

authcat ($A\natural$) - B shares at least one author with A. C shares no authors in common, and comes from the same minor category as A.

Here A might be too easy, because it conflates authorship with categorization, given that most authors only contribute to a small set of minor categories. The $A\natural$ task, restricting C to be from the same minor category as A ought to allow us to assess how sensitive the article representations are to the identities of the authors writing the article. This could offer a test as to whether the article representations are too sensitive to the stop word usage, because we suspect that to vary strongly from one author to the next, but carry little categorical content.

We also probe our readership data:

reader (R) - B is chosen to be the most similar article in terms of readership as measured by the Jaccard similarity. C is random.

readercat ($R\natural$) - B is the same as above, while C must be from the same minor category as A.

The Jaccard similarity measures how similar the readership sets are:

$$J(A, B) = \frac{|R_A \cap R_B|}{|R_A \cup R_B} \qquad (1.75)$$

For two articles, (A, B) the Jaccard similarity is given by the size of the intersection of their readership sets, divided by the size of their union. This similarity measures runs from 0 for completely disjoint sets, to 1 for identical sets.

Here, similar to the authorship task, our $\natural$ variant $R\natural$, is there to try to protect against the strong expected correlation between readership and categorization.

Finally, we design some triplets based on word usage:

words (W) - B is chosen to be the most similar article in terms of word usage, as measured by the Jaccard similarity. C is random.

wordscat ($W\natural$) - C is restricted to be from the same minor category as A.

Using the Jaccard similarity here removes any influence of the frequency of occurrence of words, instead focussing just on presence of word usage alone. This is a common strategy used in Information Retrieval [83], and could help us probe the scientific content the article, irrespective of length.

1.8.2 Classification Accuracy

Another natural test for the utility of the article vectors is to use them to power a classification algorithm. In this work, we'll use the article vectors for logistic regression, trained on 60% of the data and tested on the remaining 40%. The dataset under consideration here is relatively small, and some of the minor categories have low counts, so we expect the minor classification task to be quite difficult. On the other hand, the major classification task ought to be easy. Our intuition is that trying to tell the difference between a math and biology paper ought to be straightforward.

1.8.3 Euclidean Relationships

Just as we explored the word vectors by investigating their linear relationships, both in terms of nearest neighbors and in terms of syllogisms, we can attempt to do the same thing with the article vectors and see if we get meaningful results. This is not a quantitative task, but might give us some intuition about how the article vectors are performing.

1.8.4 Visualization

Lastly, we can hope to get a global picture of how the article vectors are distributed amongst the articles by trying to visualize them in low dimensions, utilizing a *t*-SNE projection, as we did for the word vectors.

1.9 Results

Here we report the results obtained on the arχiv corpus, whose details are described in section 1.5.

1.9.1 Description of Models

To allow for a nice comparison between the different methods, we created 14 different models of article vectors for each article in the corpus. As a benchmark, we will use the raw bag of words (bow) representation, as well as two random projections (rand, randtfidf). We will test various word projection methods (wp_*), as well as the trained methods discussed. To investigate the effect of dimensionality, all of the trained methods will be run both in 200 dimensions, to match the word projection methods, as well as 20 dimensions (*_20). This is a crude probe of a hyperparameter that could itself be optimized.

bow - The bag of words model. Here each article is represented by a large integer vector that counts how often each word appears in the article. The dimensionality of the representation is the size of the vocabulary, in this case 133 419, albeit sparse. Details in Section 1.7.1

rand, randtfidf - A random projection of the bag of words down to 200 dimensions, both using the raw word counts and tfidf weighted counts. Details in Section 1.7.1. This is equivalent to the word vector averaging methods to follow, but with random word vectors.

wp_hs, wp_b_hs, wp_tfidf_hs - Weighted averages of the word vectors obtained from the hierarchical softmax `word2vec` method of Section 1.3.3, using the raw word counts (wp_hs), a binarization of the word counts (wp_b_hs), and tfidf weighted combination of the word counts (wp_tfidf_hs) described in Section 1.7.1.

wp_ns, wp_b_ns, wp_tfidf_ns - Same as above but based on the word vectors learned using negative sampling `word2vec` (Section 1.3.4).

wp_g, wp_b_g, wp_tfidf_g - Same as above but based on word vectors learned with `GloVe` (Section 1.4)

dm, dm_20 - The concatenated distributed memory model (Section 1.7.2), in both 200 dimensions, and 20 dimensions. To run the dm method, we modified the `word2vec` code to implement the method as described in [50], with a window of size 5, 25 iterations over the corpus, and a subsampling threshold of 10^{-5}.

dm+, dm+_20 - The additive distributed memory model (Section 1.7.2). Again, this was trained by modifying the `word2vec` code to implement the method as described in [50], with the same settings as the dm method, (200, 20) dimensional vectors, a window of size 5, 25 iterations over the corpus, and a subsampling threshold of 10^{-5}.

dbow, dbow_20 - The distributed bag of words model (Section 1.7.2), using both 200 and 20 dimensional article / word vectors. To train the method, we modified the `word2vec` code to implement the method described in [18], again with a window of size 5, 25 iterations over the corpus, and a subsampling threshold of 10^{-5}.

ga, ga_20 - The extended global vector method of Section 1.7.2, in both 200 and 20 dimensions. Here we modified the existing `GloVe` code to simultaneously learn article, word, and context representations. The objective was minimized using ADAM [46], with 50 passes. The cooccurrence statistics were collected with the `GloVe` , `cooccur` utility, as well as a modified utility to collect the article-word cooccurrences, both with the default window of size 15. To help push the objective towards learning useful article representations, the word-context decomposition was weighted to contribute 0.01 as much to the total objective. The scaling parameter for the weighting function: α was set to the default 0.75 for both decompositions, but the x_0 threshold was set to 5, 100 respectfully for the article-word, and word-context decompositions.

1.9.2 Triplet Tasks

We wish to compare each of our methods' performance on the triplets defined in Section 1.8.1. But in order to compare, we need some assurance that the numbers can be compared. To investigate the error in these triplet measurements, in Table 1.8, we created 5 independent instances of the $C\flat\sharp$ triplets, each of size 20 000. In the table are shown the resulting scores and ranks across each of the methods. The scores are reported out of 1000, similar to a batting average. Notice in the last column, the standard deviation in the scores across instances is usually only a few parts per thousand. More important however is how reliable the difference in scores are across models. With subscripts, we've denoted the rank for each method for each of the triplet sets. Notice that the top performing models are consistent across the five instances, but that we observe a single rank change and tie appearance in two of the instances, highlighted in red. Across all other instances, those methods that tie differ by at most 4 points in the last reported digit across all instances.

While not comprehensive, this motivates our reporting of the triplet accuracies to the third digit, and suggests the validity of using these scores to compare across models up to a few points in the third digit.

Below in Table 1.9, we report the accuracy each of the methods achieved on the triplets defined in 1.8.1. The best results for each triplet set are marked in

method	$(C\flat\sharp)_i$					σ
bow	732_{17}	732_{17}	726_{17}	725_{17}	727_{17}	2
rand	720_{18}	720_{18}	711_{18}	713_{18}	712_{18}	3
rand_tfidf	650_{20}	650_{20}	643_{20}	645_{20}	648_{20}	3
wp_hs	912_{11}	912_{11}	$907_{\mathbf{12}}$	$914_{\mathbf{10}}$	909_{11}	2
wp_ns	913_{10}	913_{10}	$909_{\mathbf{10}}$	$914_{\mathbf{10}}$	911_{10}	2
wp_g	871_{15}	871_{15}	866_{15}	869_{15}	868_{15}	2
wp_b_hs	947_{6}	947_{6}	944_{6}	947_{6}	946_{6}	1
wp_b_ns	948_{5}	948_{5}	945_{5}	948_{5}	947_{5}	2
wp_b_g	934_{8}	934_{8}	930_{8}	935_{8}	931_{8}	2
wp_tfidf_hs	965_{1}	965_{1}	963_{1}	964_{1}	964_{1}	1
wp_tfidf_ns	963_{2}	963_{2}	961_{2}	963_{2}	961_{2}	1
wp_tfidf_g	938_{7}	938_{7}	935_{7}	938_{7}	937_{7}	2
dm	909_{12}	909_{12}	$909_{\mathbf{10}}$	912_{12}	908_{12}	1
dm20	931_{9}	931_{9}	927_{9}	933_{9}	928_{9}	2
dm+	889_{14}	889_{14}	891_{14}	893_{14}	889_{14}	2
dm+20	958_{3}	958_{3}	955_{3}	958_{3}	956_{3}	1
dbow	898_{13}	898_{13}	898_{13}	901_{13}	896_{13}	2
dbow20	956_{4}	956_{4}	953_{4}	955_{4}	953_{4}	1
ga	656_{19}	656_{19}	650_{19}	661_{19}	649_{19}	4
ga20	818_{16}	818_{16}	815_{16}	824_{16}	823_{16}	3

Table 1.8: Assessing the error in the triplet tasks by computing performance across five independent instances of the $C\flat\sharp$ triplet test. The standard deviations of the scores for each model is just a few parts per thousand. Subscripts denote the rank each method achieved in that set. Notice that the only change in rank we observe is the appearence of some ties (highlighted in red), and a single change. In all those instances, the models that could potentially switch rank differ by at most 4 in their last reported digit across all instances.

red, where we have also marked any results within 3 of the best result, as that appears to be the level at which these triplets can be compared. In bold, we've marked the "best in class" result for the model that achieved the best results amongst those of the same type, distinguishing between the average word vector methods and more sophisticated trained methods.

Notice first the overall performance on the triplets themselves, across models. As we might have expected, the $C\flat\sharp$ performance is better than the $C\flat\flat$,

	category			author		reader		words	
	$C\flat\sharp$	$C\flat\flat$	$C\sharp\sharp$	A	$A\natural$	R	$R\natural$	W	$W\natural$
bow	732	654	655	793	704	726	617	867	801
rand	720	646	641	782	696	714	613	852	789
rand_tfidf	650	621	546	842	779	780	694	937	901
wp_hs	912	856	776	910	720	916	702	979	900
wp_ns	913	858	779	909	721	916	701	978	898
wp_g	871	803	754	886	708	865	667	953	863
wp_b_hs	947	896	811	927	749	950	739	**995**	946
wp_b_ns	948	896	811	927	748	950	739	**995**	946
wp_b_g	934	870	804	924	**753**	935	729	**995**	**955**
wp_tfidf_hs	**965**	**921**	**834**	**937**	**751**	**962**	**754**	**995**	942
wp_tfidf_ns	**963**	916	823	**934**	**751**	**960**	**750**	**994**	944
wp_tfidf_g	938	883	793	925	**751**	944	741	**995**	**957**
dm	909	873	662	927	768	948	768	**989**	945
dm_20	931	893	680	922	712	941	724	981	892
dm+	889	858	666	935	799	945	780	**989**	954
dm+_20	**958**	**925**	729	935	740	**956**	750	986	910
dbow	898	867	665	**943**	**811**	951	**789**	**992**	**962**
dbow_20	**956**	917	**737**	931	729	**956**	741	986	911
ga	656	623	571	772	693	734	631	843	792
ga_20	818	764	688	869	696	858	664	949	856

Table 1.9: Table of triplet performance across models. The triplets are described in detail in section 1.8.1. There are category based triplets (C), author based (A), reader based (R), and word based (W). Shown is the performance, measured as the percentage of the triplets discerned correctly, reported as a number out of 1000. Higher is better and a random method would achieve 500. The best results (or those too close to distinguish) are shown in red. The best in class results are shown in bold. wp_tfidf_hs and dbow show the best behavior between themselves across all triplets.

which is better than the $C\sharp\sharp$. This agrees with our intuition that minor categories have a better sense of similarity than major categories. The performances observed are quite encouraging. The best observed scores across all models for those sets are (965, 922, 836). None of these methods had access to categorization information, either directly or indirectly, but they still manage to form vector representations of articles that respect those category labels. This suggests that the article representations ought to be useful for powering a classification algorithm, which we will explore in the next section.

The average word vector methods have a clear advantage over the trained methods on the categorization triplets. In many ways this is surprising, but as we explored in Section 1.6.4, the word vectors themselves, in particular the `word2vec` vectors show a remarkable respect to categorization. Also of note is the generally better performance of the lower, 20 dimensional representation in comparison to its 200 dimensional brethren. Without fail, in each instance the lower dimensional representation showed better performance on the categorization triplets.

The fact that the A, R and W tasks have such high performance themselves, highlights the fact that authorship, readership and word usages are highly correlated with categorization. We should expect those to be better on average, as they tend to give us additional information over the category level information. Looking only at the 'natural' sets, ($A\natural, R\natural, W\natural$), we observe the best performing scores of (811, 789, 962), suggesting that the word usage is the most important characteristic determining these article representations. This is unsurprising, given that they are all more or less directly computed in terms of the word usage. Still the $A\natural$ performance is impressive.

As a reminder, the task basically amounts to being presented with a reference article A, and asked to tell which is a closer match, B or C, where B is chosen at random from the set of articles that shares at least one author with A and C is a random article from the same minor category as A. For the dbow method, 81% of the time, the article with the closer representation was the one that shared at least one coauthor. Even more surprising is the $R\natural$ result. Here 79% of the time, the dbow article vector that is closer to the target had a strong readership Jaccard similarity, even though the third article was in the same minor category as the target. This suggests the learned article representations seem to capture at least some kind of representation of the content of an article, beyond just its categorization. At least some part of the representation reflects what people like to read together.

Arguably, the natural author, reader and word sets could be said to most directly correspond to the type of problem a good recommendation system would have to solve. Most readers are interested in reading about a particular field or a small set of fields in science. A field such as topological insulators is much more fine-grained than even the minor categorization affords. Topological insulator articles might appear in cond-mat.mes-hall (Mesoscale and Nanoscale Physics), cond-mat.mtrl-science (Materials Science), cond-mat.str-el (Strongly Correlated Electrons) or cond-mat.supr-con (Superconductivity) for instance, while none of those categories will be composed entirely of topological insulator papers. Consider an instance of the $A\natural$ triplet task where we happen to choose a topological insulator paper as our candidate A which happens to be in the cond-mat.mes-hall category. Which is more likely to also be a topological insulator paper: candidate B who shares at least one author, or candidate C, a random article drawn from cond-mat.mes-hall? Given that authors tend to focus their energies and papers towards a single, or small set of scientific subfields, we argue that B is more likely to also be a topological insulator paper, even if it is in a different category. A similar argument can be made for the $R\natural$ and $W\natural$ triplet tasks, since readers similarly tend to focus their energies on a single subfield, and each subfield tends to develop its own specialized vocabulary.

In light of this, the fact that the dbow method uniformly, and quite substantially leads the pack on all three of these triplet tasks, suggests that the resulting article vectors deserve greater scrutiny.

Why then does the wp_tfidf_hs method lead on the *R* and *W* triplet tasks? As argued above, these tasks conflate the notion of (readership, word usage) with categorization. Since wp_tfidf_hs does so well on the categorization triplets, its performance on *R* and *W* might be a byproduct of this performance.

In contrast to the performance on the categorization triplets, for the dm, dm+, and dbow methods the 200 dimensional representation always performs better than the 20 dimensional representation on the $A\natural, R\natural$ and $W\natural$ sets.

Notice that the random projection methods (rand, rand_tfidf) perform very competitively with the full dimensional bag of words (bow) representation, even though it is only 200 dimensions itself. This illustrates the power of random embeddings generally, that we can retain a great deal of performance even with relatively mild dimensional representations. Perhaps more surprising is that the tfidf weighted random projection actually outperforms the bow method on the authorship, readership and word usage based triplets. This illustrates the power of tfidf weights generally, and explains why they are so commonplace in document retrieval.

As we may have expected, using trained word vectors for our projection, rather than a random projection, noticeably improves performance across the board. Here too, using tfidf weights instead of raw word counts also noticeably improves performance. Comparing across the different word vector methods, the `word2vec` hierarchical softmax approach shows the strongest performance on the category based triplets, while the `GloVe` word vectors do better on the word triplets. The performance of wp_tfidf_ns tends to track the performance of wp_tfidf_hs, but lags a few points behind. The binary weighted averages (wp_b_hs, wp_b_ns, wp_b_g) in general fall intermediate between the raw word sums and the tfidf weights, suggesting that mitigating the effect of common words is important to improve performance, but additional performance gains result from maintaining a scalar representation of word usage.

The `GloVe` word vectors do not appear to be as useful as the `word2vec` vectors generally. In [78], the `GloVe` vectors performed better across the board on word analogy tasks, word similarity, and named entity recognition tasks, but here, the resulting word vectors don't seem to combine well to form useful article vector representations. This drop in performance was also observed for the `GloVe` method in detailed tests on the ability of word vectors to resolve analogy tasks, as shown in [55].

Looking at the more sophisticated methods, the dbow method appears to be the clear winner in the bunch. This is interesting, given that dm (1.7.2 takes into account the word order explicitly, while distributed bag of words (dbow 1.7.2) does not, except indirectly through the simultaneous training of word vectors.

1.9.3 Categorization Accuracy

The next evaluation is to use the resulting article representations to power a supervised classification task. To this end, we split the corpus randomly 60-40 into a training set and test set. we used the article vector representations as the input to a neural network classifier with a single hidden layer with a rectified linear activation (ReLU):

$$g(z) = \max(0, z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{1.76}$$

fed into a soft max classifier. The basic architecture is shown in Figure 1.17.

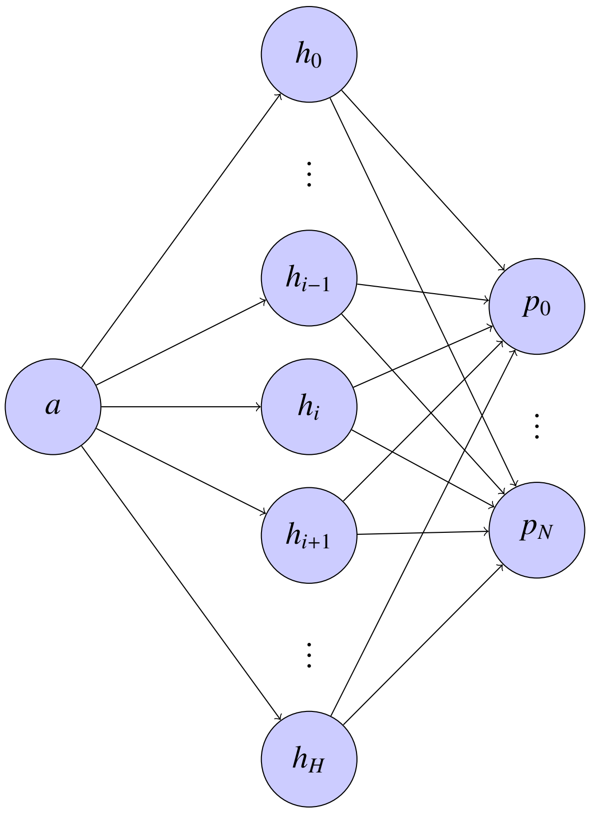


Figure 1.17: The architecture used for the classifier for the Classification evaluation. The article is fed into a single hidden layer using a Rectified Linear activation function, and then up to the output using a softmax activation.

In detail, given an article vector $\boldsymbol{a}$, this means that the hidden representation of dimension H is given by

$$h_j = \max\left(0, \sum_i W^1_{ji} a_i + b^1_j\right) \tag{1.77}$$

where the max is taken elementwise, followed by the softmax activation layer, computed as the softmax of another linear combination of the units below:

$$z_k = \sum_j W^2_{kj} h_j + b^2_k \tag{1.78}$$

$$p_k = \frac{e^{z_k}}{\sum_{m=0}^{H} e^{z_m}} \tag{1.79}$$

These outputs can be interpreted as predicted probabilities for the article to be in each of the C distinct categories.

To train the classifier, we maximize the negative log likelihood of the prediction probabilities output by the machine:

$$\mathcal{U} = -\sum_i \delta_{k,c_i} \log p_k \ , \tag{1.80}$$

where c_i is the category associated with article i. This is done by doing a modified form of stochastic gradient descent, in particular, the ADAM method [46], attempting to learn the parameters (W^1, b^1, W^2, b^2) so as to achieve the greatest performance on the training set. 50 epochs of training were applied, done in minibatches of size 128. When training the minor category classifier, the output was 143 dimensional, corresponding to the 143 minor primary categories, and the hidden layer was set to 300 dimensions. For the major category classifier, there were 6 outputs and the hidden layer was set to 50 dimensions.

To prevent overfitting and achieve better generalization, during training dropout [39] was applied at the hidden layer, wherein we randomly turn half of the hidden units off at each step during training, forcing the classifier to make use of only partial information at training time. During test time, we no longer drop out the hidden units, but multiply all of their outgoing weights by 1/2 so as to achieve the same expected level of activation.

As a benchmark, the article vector powered classifiers were compared to various instances of Naive Bayes [70]. The (nb) row corresponds to vanilla Naive

Bayes as implemented in `sklearn`.

Naive Bayes models the conditional probability of an article, as represented by its bag of words representation ($\{w_i\}$), being in a given class (c_k), by utilizing Bayes Theorem:

$$P(c_k|\{w_i\}) \propto P(\{w_i\}|c_k)P(c_k) \; . \tag{1.81}$$

The naive part comes in assuming that the conditional probabilities of word occurrence are all independent:

$$P(\{w_i\}|c_k) = \prod_i P(w_i|c_k) \; . \tag{1.82}$$

These conditional probabilities can be modelled empirically from the data, where naively the probability of a word appearing in a given class is proportional to the number of times that words appears in the given class. To help improve the method, typically some form of smoothing is applied to the distribution, the simplest version of which is Laplace smoothing, or additive smoothing, wherein:

$$P(w_i|c_k) = \frac{\#\text{ word in } c_k + 1}{\#\text{ word in corpus } + C} \tag{1.83}$$

where C is the total number of categories.

Naive Bayes is equivalent to a linear classifier, that acts on the raw bag of words representation of the article [70]:

$$\log P(c_i|a_a) \propto \log P(c_i) + \sum_j w_{ij} D^{\text{BOW}}_{aj} \tag{1.84}$$

where there is an explicit formula for the weights in terms of observed occurrence frequencies ($w_{ij} = \log P(w_i|c_k)$).

The (nbc) row corresponds to Naive Bayes applied after removing the 200 most commonly occurring words, i.e. removing the stop words before attempting to classify. This is similar to the method currently employed by the arχiv staff to help monitor incoming article classifications.

The (nbtfidf) method corresponds to a modified version of Naive Bayes that acts on the tfidf representation of the articles.

Finally the (twcnb) method corresponds to the implementation of Naive Bayes described in [85, 45], for which the authors achieved improved performance over the previous Naive Bayes implementations.

The full results are shown in Table 1.10 below. The best results are highlighted in red, while the best in class results are shown in bold. The $T\flat, t\flat$ columns correspond to observed accuracies on the (Training, test) set of the minor classification task, and corresponding $T\sharp, t\sharp$ are the observed accuracies on the major classification task.

The average word vector methods, wp_b_hs, wp_tfidf_hs, and wp_tfidf_ns manage to beat the best Naive Bayes method, nbtfidf on the major classification task, by a decent margin, achieving 94% accuracy on the test set. On the minor classification task, the nbtfidf method barely beats the dbow and wp_tfidf_hs methods with 70.2% and 69.7% accuracies respectfully. The dbow article vectors also do fairly well on the major classification task, with 93% accuracy, compared with the nbtfidf baseline of 90%.

The training set accuracies are not that meaningful, except that we see strong evidence that Naive Bayes overfits on minor classification. Granted, the minor classification task is quite difficult in this instance, with only $\sim 15,000$ articles in our training set after our split, and as we observed in Figure 1.7, some of the categories have very small populations for the January to March of 2015 time

	$T\flat$	$t\flat$	$T\sharp$	$t\sharp$
rand	535	436	861	849
rand_tfidf	669	401	811	764
wp_hs	691	652	932	926
wp_ns	665	637	926	924
wp_g	544	523	900	901
wp_b_hs	702	670	938	**936**
wp_b_ns	676	651	933	931
wp_b_g	615	597	916	918
wp_tfidf_hs	751	**697**	943	**936**
wp_tfidf_ns	728	690	939	**936**
wp_tfidf_g	729	676	937	934
dm	863	653	937	907
dm_20	629	551	899	898
dm+	900	678	952	923
dm+_20	669	606	916	912
dbow	888	**697**	956	**931**
dbow_20	656	602	918	914
ga	629	385	871	846
ga_20	441	361	857	854
nb	871	654	916	887
nbc	917	673	925	891
nbtfidf	972	**702**	929	**898**
twcnb	607	487	854	854
(dbow,wp)	887	**714**	963	**942**

Table 1.10: Table of classification performance across models. The methods were trained to classify the categories for both the minor categorization task ($\flat$), and the major one ($\sharp$). T denotes the error achieved on the training set, t the test set error. The original corpus was split (60%, 40%) into training and test sets. The classifier used for the article vector representations was a single layer neural network, with retified linear activation of size (300, 50) for the (minor, major) task, with dropout, fed into a log softmax layer, trained with the negative log likelihood criterion. Training was done with stochastic gradient descent, using the ADAM method [46] for 50 epochs with a minibatch of size 128.

period. In comparison, the article vector powered neural network classifiers with dropout don't show as much overfitting.

For the classification task, the clear winner is the wp_tfidf_hs method. Amongst the trained article vector methods, dbow is again the clear winner. Given that the performance on the triplet tasks seemed to suggest that each brought complementary representations of articles, we could hope to do better by combining both. To this end, we concatenated the normalized article vectors from both the wp_tfidf_hs and dbow methods, and passed this 400 dimensional representation through the same classification framework. The results are shown in blue in the last row (dbow, wp) of Table 1.10, and the combination clearly wins on not only the major classifications, but also outperforms the best performing Naive Bayes method on the minor classification task.

1.9.4 Euclidean Relationships

Just as we did with the word vectors, we can attempt to get a sense of the utility for the article vectors by looking at the nearest articles to a given article. Our previous two evaluations have suggested that the two approaches deserving of further investigation are the wp_tfidf_hs and dbow method, which not only perform the best across the triplet tasks, but appear to be complementary in their performance. wp_tfidf_hs shows a remarkable ability to reflect the existing categorization scheme on the arχiv, while the dbow method shows potential for capturing the types of relationships that would be useful for a recommendation system.

As our first test, we will probe using arχiv article 1502.03520: Random Walks on Context Spaces: Towards an Explanation of the Mysteries of Semantic Word Embeddings [3]. The nearest articles for the dbow, wp_tfidf_hs, and bow methods are shown in Table 1.11 below.

1502.03520	0.000	**Random Walks on Context Spaces: Towards an Explanation of the Mysteries of Semantic Word Embeddings**
		dbow
1501.00358	0.441	Comprehend DeepWalk as Matrix Factorization
1503.06760	0.441	Unsupervised POS Induction with Word Embeddings
1502.07257	0.456	Breaking Sticks and Ambiguities with Adaptive Skip-gram
1502.04081	0.468	A Linear Dynamical System Model for Text
1502.03630	0.479	Ordering-sensitive and Semantic-aware Topic Modeling
		wp_tfidf_hs
1502.04081	0.076	A Linear Dynamical System Model for Text
1502.07257	0.080	Breaking Sticks and Ambiguities with Adaptive Skip-gram
1503.05543	0.089	Text Segmentation based on Semantic Word Embeddings
1502.06922	0.089	Deep Sentence Embedding Using the Long Short Term Memory Network: Analysis and Application to Information Retrieval
1502.06665	0.091	Reified Context Models
		bow
1501.05200	0.107	Minimax Optimal Sparse Signal Recovery with Poisson Statistics
1503.03149	0.109	Stern-Gerlach surfing in laser wakefield accelerators
1501.02086	0.110	Renormalization of an Abelian Tensor Group Field Theory: Solution at Leading Order
1501.04455	0.114	Eternal Higgs inflation and cosmological constant problem
1501.07773	0.114	Polyhedral Omega: A New Algorithm for Solving Linear Diophantine Systems

Table 1.11: Nearest article vectors to 1502.03520, a paper described in Section **??**, given by the dbow, wp_tfidf_hs, and bow methods. Articles deemed unrelated are marked in grey.

Looking at the nearest neighbors, a few things are apparent. First, the bag of words representation does not yield meaningful neighbor articles. It was included here to give a sense of benchmark. The bag of words representation is used throughout much of document retrieval generally, but does a poor job here. The dbow and wp_tfidf_hs methods both give related articles, but the dbow neighbors are arguably more in line with the *spirit* of the Arora article, while it is clear the wp_tfidf_hs articles share a common vocabulary.

For instance, the last three articles listed for wp_tfidf_hs are about natural language processing, including 1503.05543: Text Segmentation based on Semantic Word Embeddings, which appears as Chapter 2 in this work, but that article is about segmenting documents into coherent sections, and while powered by semantic word embeddings, and even references the Arora article, and so related, is not in the same spirit as the Arora et al. article.

Contrast this with the closest dbow article, 1501.00358: Comprehend DeepWalk as Matrix Factorization. This article is about interpreting DeepWalk [79], which itself is basically `word2vec` applied to social networks, as a form of matrix factorization. It is the complementary article to the Arora article, in the social network context. Whether this serves as the most useful recommendation to a reader interested in the Arora article is uncertain, but it is interesting that the dbow article representation marks it as the most similar article.

1502.04081 appears in both the dbow and wp_tfidf_hs lists. That article similarly defines a generative model that can create semantic word embeddings, but with a focus on part of speech tagging. 1502.07257 also appears in both lists, and also extends skip-gram with a generative model, this time focussed on disambiguating multiple senses for individual word usage.

Just as with the word vectors, we could try to probe more than nearest neighbors, and instead try to probe linear combinations of article and word vectors. Both dbow and wp_tfidf_hs have corresponding representations of each word, dbow, because they are simultaneously trained, and wp_tfidf_hs, because it is built from the `word2vec` hierarchical sampling word vectors. Since the wp_tfidf_hs article vectors are built from a sum of all of their words, the article vectors and word vectors have very different magnitudes. To alleviate this, for the wp_tfidf_hs neighbors, the word and article vectors were first normalized before added. The results are shown below in Table 1.12.

	0.000	**1502.03520 - 'text' + 'video'**
		dbow
1502.03520	0.178	Random Walks on Context Spaces: Towards an Explanation of the Mysteries of Semantic Word Embeddings
1502.08029	0.466	Describing Videos by Exploiting Temporal Structure
1501.00358	0.494	Comprehend DeepWalk as Matrix Factorization
1503.08909	0.495	Beyond Short Snippets: Deep Networks for Video Classification
1501.07873	0.502	Deep Neural Networks for Sketch Recognition
1502.06108	0.503	Don't Just Listen, Use Your Imagination: Leveraging Visual Common Sense for Non-Visual Tasks
		wp_tfidf_hs
1502.01094	0.232	Multimodal Task-Driven Dictionary Learning for Image Classification
1503.06642	0.238	Superpixelizing Binary MRF for Image Labeling Problems
1501.00092	0.241	Image Super-Resolution Using Deep Convolutional Networks
1502.03532	0.241	An equalised global graphical model-based approach for multi-camera object tracking
1503.07989	0.242	Discriminative Bayesian Dictionary Learning for Classification
1503.00072	0.242	DeepTrack: Learning Discriminative Feature Representations Online for Robust Visual Tracking

Table 1.12: Nearest article vectors to 1502.03520 - "text" + "video". The wp_tfidf_hs article and word vectors were normalized before being combined.

In the dbow neighbors, we see the appearance not only of 1502.03520 itself, but also one of its neighbor: 1501.00358. The remaining models deal roughly with generative models for processing video, as we wished. In the wp_tfidf_hs listing, only two of the articles 1502.03532 and 1503.00072 deal with video processing.

To demonstrate that the resulting articles are not simply those nearest the word "video", in Table 1.13 below, we show those articles. Notice that this also serves and an example of our ability to query articles based on words or a combination of words.

Noticeably, none of these articles appeared in our previous query, and all of them are clearly articles dealing with video processing.

	0.000	**"video"**
		dbow
1503.06917	0.261	Unsupervised Video Analysis Based on a Spatiotemporal Saliency Detector
1501.05964	0.296	Advances in Human Action Recognition: A Survey
1501.04367	0.309	Reconstruction-free action inference from compressive imagers
1502.01812	0.320	Crowded Scene Analysis: A Survey
1503.00843	0.326	A Survey On Video Forgery Detection
1501.02825	0.337	A Survey on Recent Advances of Computer Vision Algorithms for Egocentric Video
		wp_tfidf_hs
1502.04132	0.216	Long-short Term Motion Feature for Action Classification and Retrieval
1503.06917	0.224	Unsupervised Video Analysis Based on a Spatiotemporal Saliency Detector
1501.06993	0.240	Feature Sampling Strategies for Action Recognition
1501.04367	0.244	Reconstruction-free action inference from compressive imagers
1502.00416	0.246	Towards a solid solution of real-time fire and flame detection
1501.02825	0.247	A Survey on Recent Advances of Computer Vision Algorithms for Egocentric Video

Table 1.13: Nearest article vectors to "video".

1.9.5 Visualization

To get a global sense of our article vectors, just as we did with the words, we can visualize all of the article vectors by applying *t*-SNE. Since we are primarily interested in cosine distance as our metric in the high dimensional space, we first normalized all of the article vectors.

In Figure 1.18, we show the *t*-SNE projection for the wp_tfidf_hs article vectors, colored according to the same colorscheme as used in Figure 1.7.

Notice how wonderfully the articles separate themselves into different major and minor categories. We see not only a global structure separating physics (green), math (blue), cs (red), stat (yellow), q-bio (orange), and q-fin (purple), but homogeneous groupings of single shades are clearly visible.

Compare this to Figure 1.19, the projection of our dbow article vectors. The two look remarkably similar. The same global and local structure can be seen throughout.

Contrast these with Figure 1.20, the projection of our random 200 dimensional projection of the bag of words representations. Here an overall gradient from red to green to blue is visible, but all of the detailed structure is lost.

Clearly, both the wp_tfidf_hs and dbow article vectors have a great deal of structure, and the appearance of the same motifs in both projections suggest that the structure is inherent in the arχiv dataset itself.

Finally, the dbow method simultaneously learns a representation for each word. The *t*-SNE projection of these word vectors is shown in Figure 1.21.

Compared with the earlier `word2vec` skip-gram hierarchical softmax word vector projection (Figure 1.18), these word vectors have much more fine grained structure, with many small isolated clumps. Presumably, this additional structure comes from simultaneously training the article vectors, since the presence of the article context could help resolve ambiguities in word meaning.

As another quick illustration of the difference of the dbow word vectors, let's compare the nearest neighbors to 'singular_value_decomposition_svd', as we did earlier in Table 1.5. The dbow nearest neighbors are arguably better. This could be due to the additional information afforded by the article contexts, or it could be due to the extra number of iterations over the corpus. For comparison then, we've also included the nearest neighbors for skip-gram hierarchical softmax `word2vec` but allowing 25 iterations over the corpus (hs25), the same number of iterations as dbow had. In spite of this, the dbow nearest neighbors are arguably more meaningful, not only are terms such as 'discrete_fourier_transform_dft' absent, but the meaningful neighbors have better cosine similarity in the dbow set.

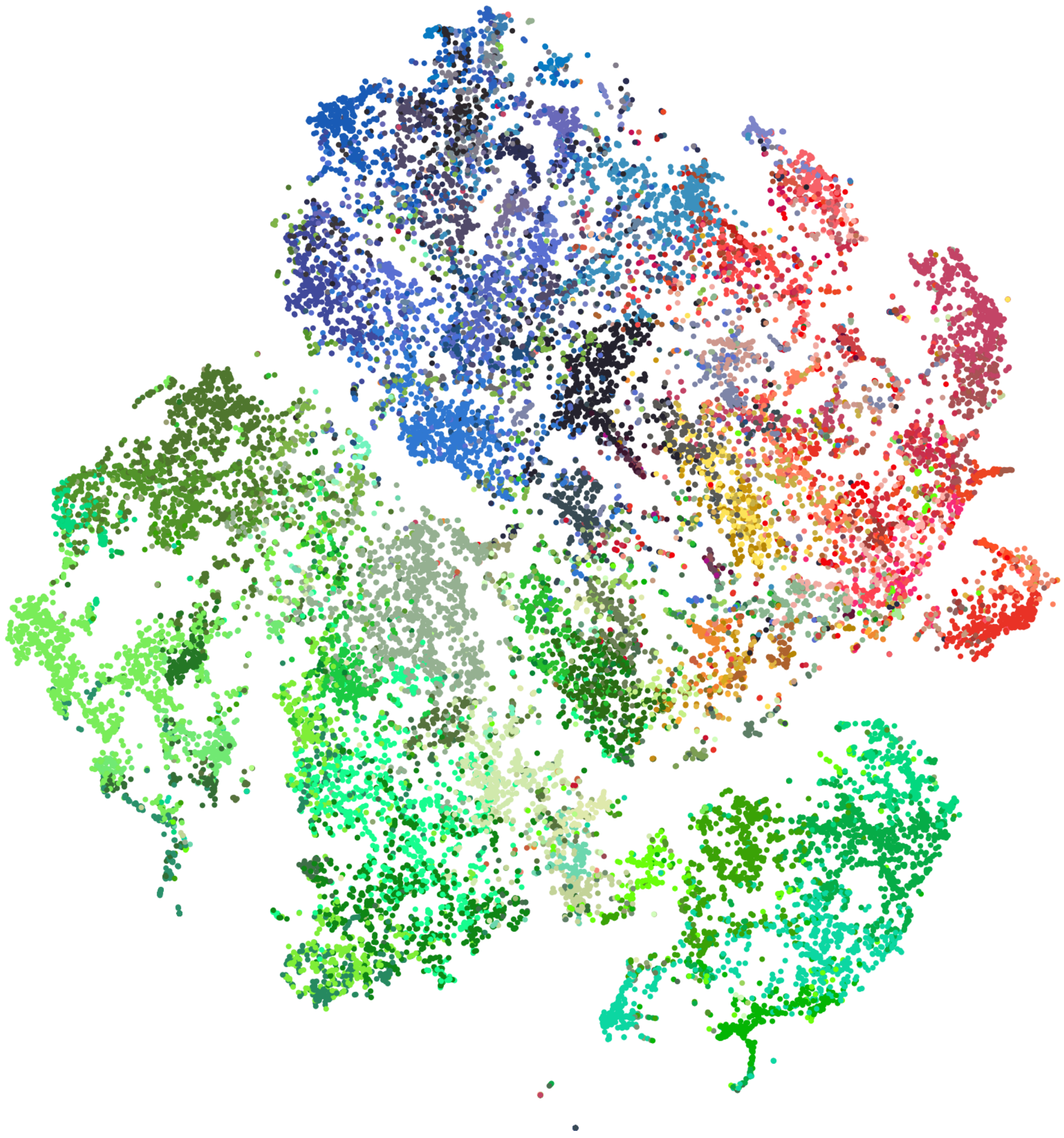


Figure 1.18: tSNE projection of wp_tfidf_hs article vectors.

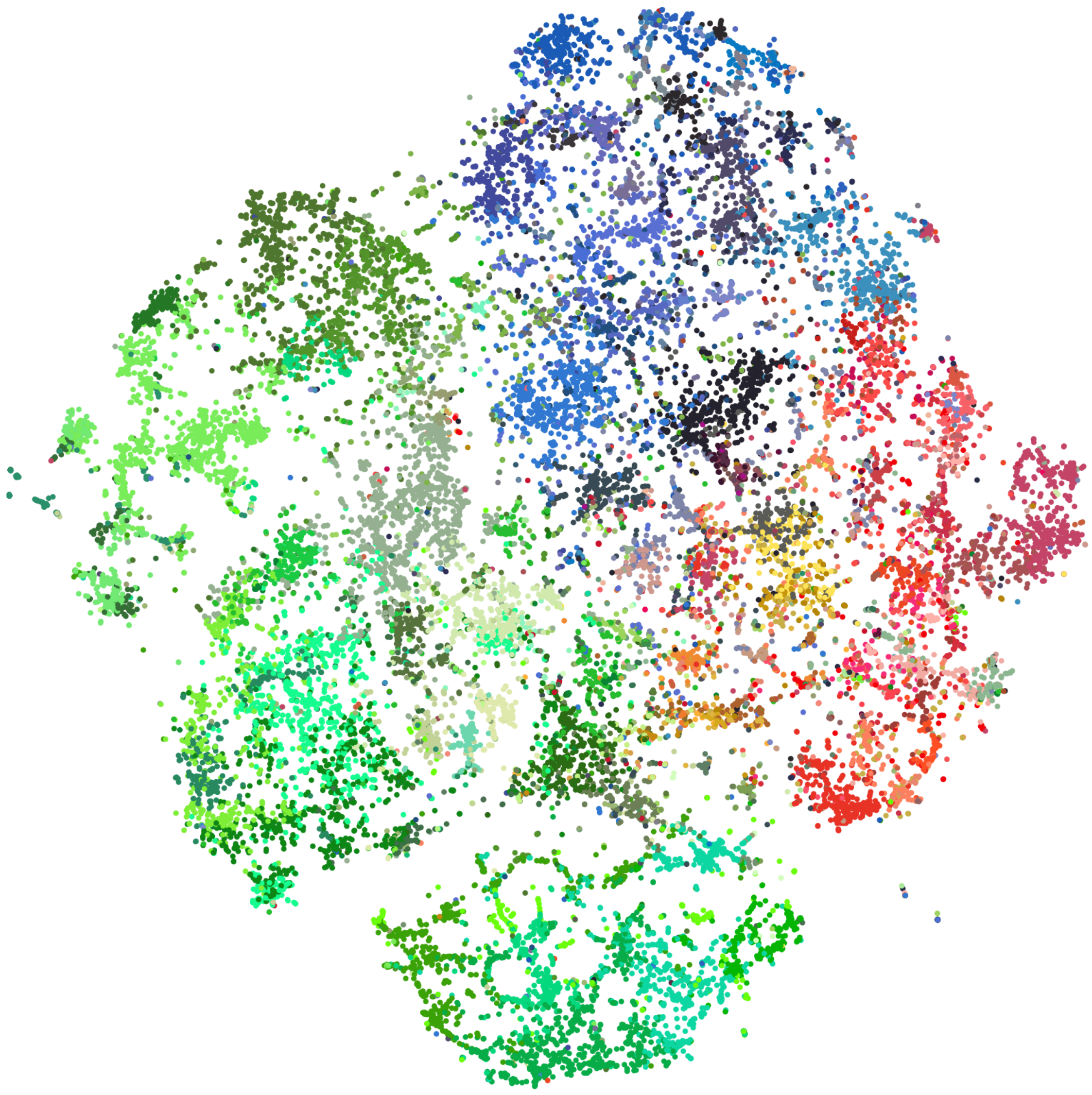


Figure 1.19: tSNE projection of dbow article vectors.

The apparent boost we see in the utility of the word vectors, coupled with the amazing performance of the dbow article vectors, suggests we could extract even more semantic information if we incorporate additional metadata into our training objectives.

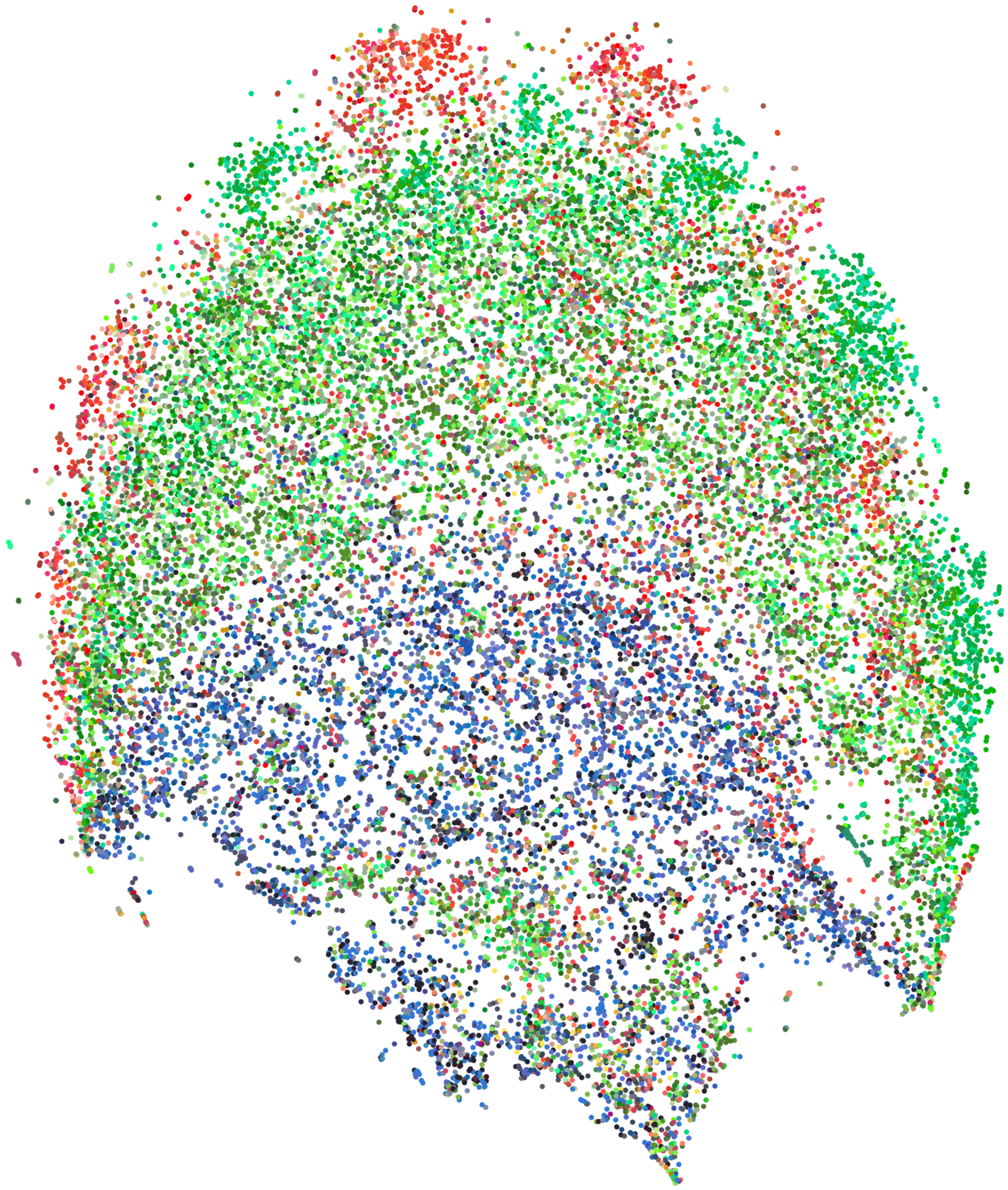


Figure 1.20: tSNE projection of the ‘rand’ article vectors, a random 200 dimensional projection of the BOW representation.

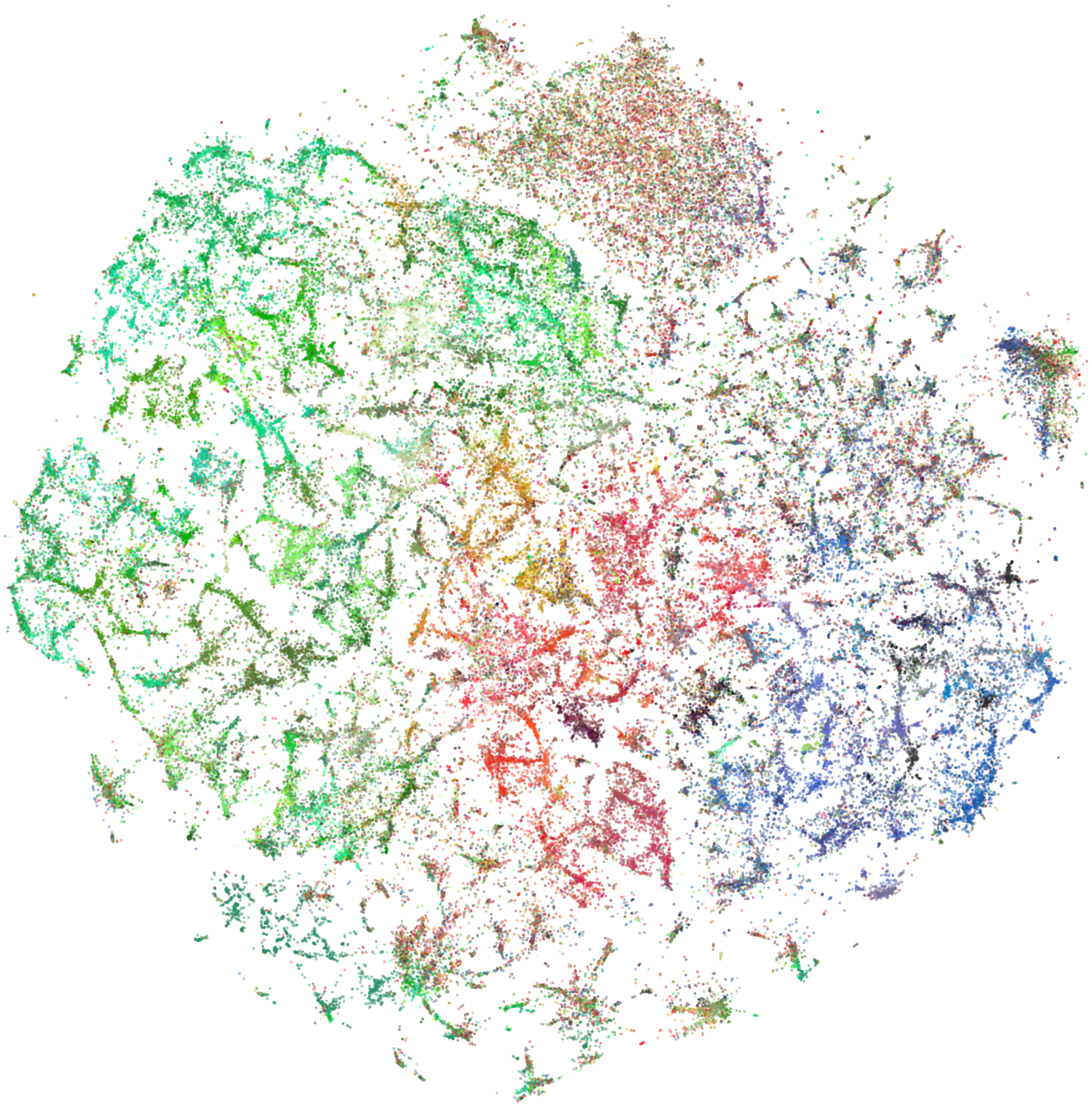


Figure 1.21: The word vectors simultaneously learned during dbow training. This should be compared to the word vector embeddings, particularly Figure 1.18.

singular_value_decomposition_svd					
hs		hs25		dbow	
orthonormal_columns	0.37	svd	0.30	svd	0.22
sparsifying_basis	0.37	singular_value	0.36	left_singular_vectors	0.26
eigen_decomposition	0.37	tucker_decomposition	0.45	singular_value	0.27
rank_deficient	0.38	orthonormal_columns	0.45	right_singular_vectors	0.28
sparsified	0.38	dimensionality_reduction	0.46	orthonormal_columns	0.30
tucker_decomposition	0.40	low_rank	0.46	truncated_svd	0.32
right_singular_vectors	0.40	discrete_fourier_transform_dft	0.47	eigendecomposition	0.35
blockwise	0.40	blind_source	0.47	eigen_decomposition	0.37
efficiently_computed	0.40	eigen_decomposition	0.48	tucker_decomposition	0.39
hard_thresholding	0.41	alternating_least_squares	0.48	singular_values	0.39

Table 1.14: Nearest word vectors to "singular_value_decomposition_svd" and their resulting cosine distance for the skip-gram hierarchical softmax `word2vec` method (hs), the same method trained for 25 iterations over the corpus (hs25) and the dbow word vectors.

1.10 Possible Extensions

Having demonstrated that we can learn meaningful word vectors, even on a scientific corpus such as the arχiv, and that we can further learn useful vector representations of articles, the next question is how we could extend this to incorporate authorship, category, and readership data as well.

Our arχiv data naturally has a finite set of correspondences.

word-word - At the level of a text corpus, we have word-word correspondences, such as those used to train our word vectors.

article-word - As we explored in the previous sections, we can associate each article with its collection of words.

article-author - Each article naturally has the corresponding list of authors.

article-category - Each article has a single associated primary category, as well as the potential for cross-listings on several categories.

article-reader - Having access to readership data at the cookie or ip address level, those accesses are naturally connected with particular articles.

Ideally, we would like to simultaneously learn vector representations for words, articles, authors, categories and readers, all living in the same vector space.

There are three natural schemes to try to incorporate all of these sources of information.

1.10.1 More averaging

The first scheme is the simplest. Given that we saw great success in representing articles as weighted averages of the word vectors making them up (Section 1.7.1), we could similarly try to represent authors, categories and readers by just averaging the article vectors associated with them. If our article representations are useful, these other representations should also be useful. This is also extremely cheap. For instance, below in Figure 1.22 we demonstrate a *t*-SNE embeddings of category vectors, where the category is represented as a sum of all of its article vectors.

As you can see, this embedding clearly respects some notion of similarity between the different categories. The different major categories nicely separate into different groups, with some illustrative exceptions. math.HO (History and Overview) is separated from the rest of the math categories, and instead lies very close to physics.hist-ph (History and Philosophy of Physics). physics.soc-ph (Physics and Society) lies closest to cs.SI (Social and Information

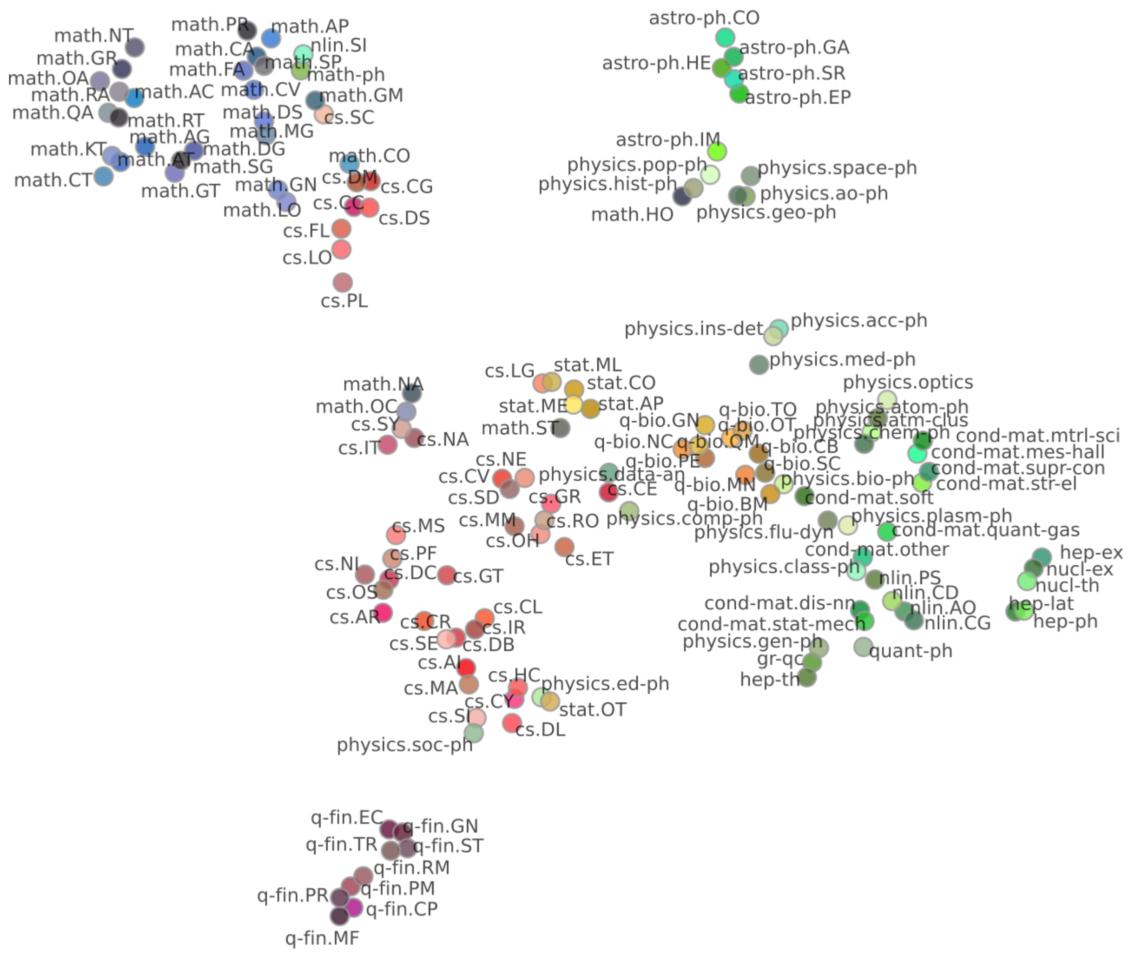


Figure 1.22: A *t*-SNE projection of the category vectors formed from averaging all of the normalized wp_tfidf_hs article vectors in each category. The category vectors were normalized before the projection, and a perplexity of 5 was used. The labels were adjusted by hand to avoid overlaps.

Networks). The astro categories segregate themselves off from most of the rest of physics, save physics.space-ph (Space Physics), physics.ao-ph (Atmospheric and Oceanic Physics), physics.geo-ph (Geophysics), physics.pop-ph (Popular Physics) and physics.hist-ph (History and Philosophy of Physics). We also see hep-ex (High Energy Physics: Experiment), nucl-ex (Nuclear Experiment), nucl-th (Nuclear Theory), hep-lat (High Energy Physics: Lattice) and hep-ph (High Energy Physics: Phenomenology) segregate themselves a bit.

1.10.2 Multiple Domain Matrix Factorization

The next scheme would be to treat each of our cooccurrence data sources as a separate sparse matrix and attempt to simultaneously factor all of these matrices with shared representation vectors. The first step in this direction was our proposed GloVe article vectors of the previous section (Section 1.7.2). Incorporating more sources of information would amount to simply adding additional terms to our loss function. As we saw earlier, however, each additional data matrix brings with it a whole set of choices to make regarding the relative weight of that factorization in the objective, not to mention the fact that we could consider more general forms of objective, or even loss functions.

In the extreme, we could consider the general problem, in the spirit of [107], wherein we would have as our parameters the vector representations for words (W), context words (C), articles (A), authors (N), categories (C), and readers (R), and try to simultaneously form low rank factorizations for each of our data sources: word-word: X, article-word: D, article-author: I, article-category: G, and article-reader: U. Naturally, being general, each of these factorizations could have their own loss function:

$$\begin{aligned}\mathcal{U} = \sum_{(w,c)} L^X(W_w C_c, X_{wc}) + \sum_{(a,w)} L^D(A_a W_w, D_{aw}) \\ + \sum_{(a,n)} L^I(A_a N_n, I_{an}) + \sum_{(a,g)} L^T(A_a G_g, T_{at}) + \sum_{(a,r)} L^U(A_a R_r, U_{ar}) \quad (1.85)\end{aligned}$$

As in `GloVe` , and our attempt to simultaneously learn word-word and article-word representations, minimally we could take each of these losses to be a weighted form of squared loss, with scalar coefficients to influence the relative contribution of each. But we could also consider more generalized forms of loss, such as Hinge loss, Huber loss, or Ordinal Loss [107]. We could additionally augment our objective with regularization terms, such as quadratic or ℓ_1 regularization, which would penalize our vectors from getting too large and help keep our vectors generalizable.

While Generalized Low Rank Models are powerful, they also prescribe a large space of possibilities, and a proper exploration would require a great deal of hyperparameter optimization. Our initial investigation of simultaneously learning word and article vectors (ga, ga20) did not perform as well as even the weighted average of the word vectors, leaving the outlook bleak for more general factorization schemes.

1.10.3 Multiple Domain Negative Sampling

Finally, we could imagine extending the dbow method (Section 1.7.2) to incorporate additional sources of metadata. Unfortunately, in our earlier instance of dbow, we had vector representations for both words and articles which we passed to the hierarchical softmax classifier for the target words. The use of the hierarchical softmax classifier meant that we did not simultaneously learn vector representations for our contexts. This itself doesn't generalize well to our additional sources of information. The article-author, article-category and article-reader relationships are naturally expressed in terms of articles, not words. We could create cooccurrences for each of these at the word level, feed them into our hierarchical softmax classifier and hope for the best, but I suspect there is a better way.

It should be more more natural to simultaneously learn word, context, article, author , reader and category vectors using negative sampling (Section 1.3.4).

As a reminder, when doing negative sampling, all we need is a model for our noise distribution. This suggests a very natural scheme for training:

$$\begin{aligned}\mathcal{U} = \sum_a \sum_{w\in a} \left[\log\sigma(\boldsymbol{a}\cdot\boldsymbol{w}) + \frac{1}{N}\sum_{n=1}^{N} \log\sigma(-\boldsymbol{a}\cdot\boldsymbol{w}') + \sum_{c\in n(w)} \left\{ \log\sigma(\boldsymbol{w}\cdot\boldsymbol{c}) + \frac{1}{N}\sum_{i=1}^{N}\log\sigma(-\boldsymbol{w}\cdot\boldsymbol{c}') \right\} \right] \\ + \sum_a \sum_{g\in a} \left[\log\sigma(\boldsymbol{a}\cdot\boldsymbol{g}) + \frac{1}{N}\sum_{n=1}^{N}\log\sigma(-\boldsymbol{a}\cdot\boldsymbol{g}') \right] \\ + \sum_a \sum_{n\in a} \left[\log\sigma(\boldsymbol{a}\cdot\boldsymbol{n}) + \frac{1}{N}\sum_{n=1}^{N}\log\sigma(-\boldsymbol{a}\cdot\boldsymbol{n}') \right] \\ + \sum_a \sum_{r\in a} \left[\log\sigma(\boldsymbol{a}\cdot\boldsymbol{r}) + \frac{1}{N}\sum_{n=1}^{N}\log\sigma(-\boldsymbol{a}\cdot\boldsymbol{r}') \right]\end{aligned} \tag{1.86}$$

While this objective looks complex, it is simply the natural extension of negative sampling to all of our data relationships. Here a is an article vector, w a word vector, c the context vector, g a category vector, n an author vector and r a reader vector. For each observed cooccurrence as we read our corpus, we take a single step attempting to align the observed pair, then generate N false examples and attempt to anti-align those pairs. A textual description of the objective can be found in Figure 1.23.

The remaining task is to specify the distributions from which we generate all of our noise examples. A natural choice would be the power law smoothed unigram distribution for each type of vector. Just as we saw some evidence that the simultaneous training of word and article vectors in the dbow method improved our representation of both, we hope that simultaneous training with all of our different forms of natural cooccurrence data would lead to very powerful vector representations of all entities on the arχiv .

In future work, we hope to apply this method to a substantial portion of

```
For each article ($a$):
    *Learn the word-article relationships*
    For each word in the article ($w$):
        Align ($\boldsymbol{a}, \boldsymbol{w}$), and anti-align $N$ random words ($\boldsymbol{a}, \boldsymbol{w}'$)
        *Learn the word-context relationships*
        For each nearby context word ($c$):
            Align ($\boldsymbol{w}, \boldsymbol{c}$), and anti-align $N$ random contexts ($\boldsymbol{w}, \boldsymbol{c}'$)
    *Learn the article-category relationships*
    For each associated category ($g$):
        Align ($\boldsymbol{a}, \boldsymbol{g}$), and anti-align $N$ random categories ($\boldsymbol{a}, \boldsymbol{g}'$)
    *Learn the article-author relationships*
    For each associated author ($n$):
        Align ($\boldsymbol{a}, \boldsymbol{n}$), and anti-align $N$ random authors ($\boldsymbol{a}, \boldsymbol{n}'$)
    *Learn the article-reader relationships*
    For each associated reader ($r$):
        Align ($\boldsymbol{a}, \boldsymbol{r}$), and anti-align $N$ random readers ($\boldsymbol{a}, \boldsymbol{r}'$)
```

Figure 1.23: Proposed method for simultaneously learning word, context, article, author, category and reader vectors, in pseudocode.

the existing arχiv and analyze how useful the formed representations are at powering various applications.

1.11 Possible Applications

Once we have developed vector representations of words and articles, or if we happen to further form vector representations of categories, authors and readers, the next question is what we can use these for.

1.11.1 Classification

As already explored in 1.9.3, the vector representations for articles can be used to power a classification algorithm. We've already demonstrated the ability to beat the existing Naive Bayes framework, including enhanced versions of Naive

Bayes. With additional context information, if the article vectors are even more meaningful, we could hope for even better classification accuracy. Additionally, if we had vector representations for categories, that opens new doors for potential classification schemes.

As we've seen hints already, there is some overlap between different arχiv categories. Another very interesting question would be: How many categories *should* there be, and how should they be organized? Having complete representations of all articles would allow us to start to probe this question. We could imagine trying out various hierarchical clustering schemes [35], or more recent Information based clustering approaches [99, 95] or convex clustering [13] to try to estimate the correct number and distribution for arχiv categories.

As a first attempt in this direction, we can perform K-Means clustering with varying number of clusters, and evaluate the Bayesian Information Criterion to help decide the appropriate number of clusters [77]. Below in Figure 1.24, we show the results obtained clustering the normalized dbow article vectors. This crude estimate suggests that the optimal clustering of articles, as given by their dbow vector representation, would require closer to 161 categories, rather than the existing 143.

How similar are these proposed clusters in terms of the existing categorization scheme? On average, each existing arχiv category has an entropy over the k-mean cluster identities of around 2.5, corresponding to a perplexity of around 6. This means each of the existing arχiv categories, on average, has articles that come from nearly 6 different k-mean clusters.

Some of these k-mean clusters themselves have a large entropy with re-

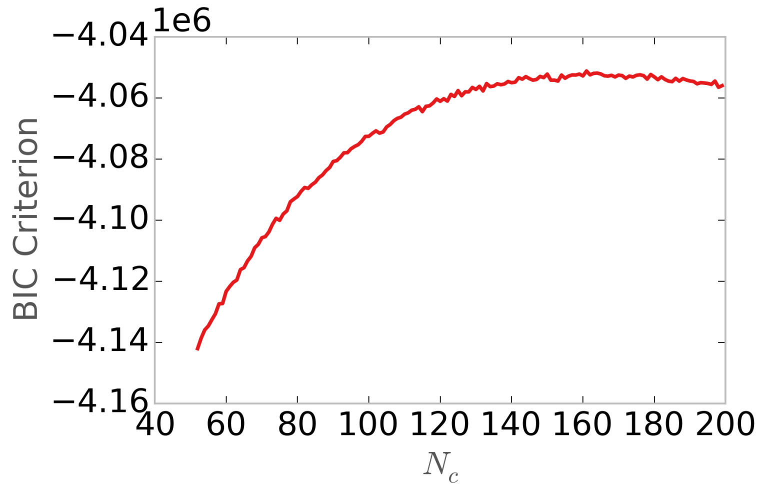


Figure 1.24: The BIC criterion for various K-Means clusterings of the normalized dbow article vectors. There is a maximum seen at 161 clusters.

spect to existing classifications. An example of one of these extended clusters is shown below in Table 1.15. You'll notice that the articles near the center of this cluster are spread across many distinct arχiv categories, but all of the articles clearly focus on matrix/tensor factorizations.

artid	categories	title
1503.08601	cs.NA	Finding a low-rank basis in a matrix subspace
1503.00374	cs.DS	A Randomized Algorithm for Approximating the Log Determinant of a Symmetric Positive Definite Matrix
1501.05385	cs.SC	Randomized Circulant and f-circulant Preprocessing
1501.06726	math.SP	Further Results on Cauchy Tensors and Hankel Tensors
1501.01571	math.PR	An Introduction to Matrix Concentration Inequalities
1501.07564	math.NA	The Lyapunov matrix equation. Matrix analysis from a computational perspective
1503.07157	math.NA	A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices
1503.06394	cs.DS	Large-scale Log-determinant Computation through Stochastic Chebyshev Expansions
1503.02615	math.NA	Approximation of functions of large matrices with Kronecker structure
1503.05479	cs.LG	Interpolating Convex and Non-Convex Tensor Decompositions via the Subspace Norm

Table 1.15: The articles nearest one of the k-means clusters with a large entropy relative to the existing categories. The articles are split amongst several existing categories, but all deal with matrix / tensor factorization.

On the other hand, some of the k-means clusters have low entropy with respect to the existing arχiv categories, such as the one in Table 1.16 below. This cluster clearly focuses on the physics of active matter, and is concentrated on cond-mat.soft articles. However cond-mat.soft itself is split amongst nearly 5 distinct k-means clusters.

artid	categories	title
1502.02229	cond-mat.soft	Motility-induced phase separation and coarsening in active matter
1503.02610	cond-mat.soft	Hydrodynamic Collective Effects of Active Protein Machines in Solution and Lipid Bilayers
1501.07266	cond-mat.soft	Tuned, driven, and active soft matter
1501.04054	cond-mat.stat-mech	Rotational and translational diffusion in an interacting active dumbbell system
1501.07185	physics.bio-ph	Mechanics of motility initiation and motility arrest in active gels
1502.02437	cond-mat.soft	Polarisation of cells and soft objects driven by mechanical interactions: Consequences for migration and chemotaxis
1502.07144	cond-mat.soft	Self-assembly of Active Colloidal Molecules with Dynamic Function
1502.07115	cond-mat.soft	A minimal physical model captures the shapes of crawling cells
1503.06454	cond-mat.stat-mech	Run-and-Tumble Dynamics of Self-Propelled Particles in Confinement
1502.03975	physics.bio-ph	Amoeboid motion in confined geometry

Table 1.16: The articles nearest the center of a k-means cluster focused on the physics of active matter. While these articles tend to come from cond-mat.soft, cond-mat.soft itself is split amongst nearly 5 distinct k-means clusters.

With more powerful article representations and more careful analysis of the clustering, techniques like these could be used to help refine the existing categorization scheme.

1.11.2 Search

Currently, full text search on the arχiv is powered by a co-active learning algorithm [84, 93]. This algorithm attempts to learn a better way of ranking the search results over time by observing how often users click on the first, second, third, etc. result from the search results. The co-active learning model used, at its heart, is attempting to learn the optimal weights for a simple linear perceptron based on a feature vector representing both the context and the article, $\phi(x, y)$. In practice, currently this feature vector is a 1000 dimensional vector including query-article terms, such as whether the search term is in the article, and query independent terms, such as the age of the document. The 3PR algorithm used [84] is completely indifferent to this joint feature representation. Naturally, the observed utility of word and article vectors suggests augmenting the existing feature vector with features such as dot product between the vector representing all of the words in the search query and each candidate article.

It is also natural to imagine *personalizing* the search, by including user specific features. In the vector representation context, this would naturally include the dot product between the given user's vector and each candidate article.

1.11.3 Recommendation

Related to search is the notion of recommendation. We could imagine attempting to suggest interesting articles to read associated with each article. This could naturally be handled by the existing 3PR algorithm, again with suggested modifications involving the use of article vectors to power the similarity search.

Our article vectors suggest a natural sense of article similarity, as we explored in Section 1.9.4. We could imagine using them on their own to suggest similar articles for each existing article. We could also imagine using the article vectors as features for other article similarity or clustering algorithms, such as Bayesian ranking [76].

Some caution is in order. Given the popularity of the arχiv, any global recommendations that are made can have a large effect on the future citations of those articles. An effect on future citations can be measured in terms of the order articles appear in the nightly emails [34]. Instead of making global recommendations to all users on the site, we could instead imagine personalized recommendations tailored to the preferences of an individual reader. By personalizing the recommendations, there would be less chance that articles would become popular just because they were recommended as similar to existing popular articles.

Vector representations could naturally be used either directly to power a personalized recommendation system, or as a source of powerful features for algorithms such as 3PR [84].

1.11.4 Segmentation

While articles are the natural unit of text on the arχiv, often an entire article is not the natural unit of text we consume when reading an article. Often times, there is just a single section of an article of immediate interest.

Another intriguing possible use of our word vectors is as features of a segmentation algorithm, that could automatically split an article up into relevant sections. Once all of the articles have been split, we can imagine modifying our search, similarity or recommendation tasks to target article segments rather than entire articles themselves.

We will explore this use in detail in Chapter 2, but as a teaser, consider Figure 1.25. Here we have displayed the matrix of word vectors obtained just by scan-

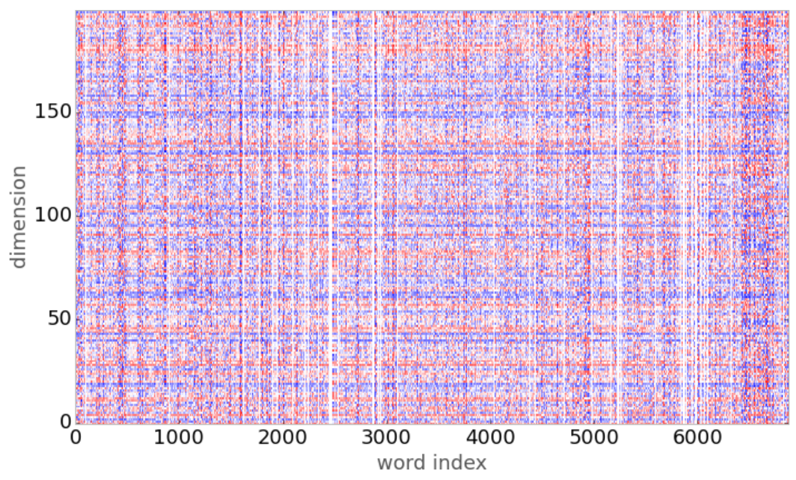


Figure 1.25: The word vector matrix for arχiv article: 1503.01104 which appears as Chapter 3 in this work. You can see hints of structure that a segmentation algorithm could take advantage of. This will be explored in Chapter 2.

ning through all of the words in an article. There is clear evidence that structure and breaks between different sections of the article can be seen at this level.

1.12 Conclusion

In this work, we gave an overview of modern techniques for learning vector representations of words. We introduced the data available on the arχiv and analyzed a particular 3 month corpus. We trained various word vector methods and analyzed their utility for finding similar words, solving analogies and investigated their global structure with nonlinear embeddings. We discussed various techniques for learning vector representations of articles, as well as several quantitative and qualitative techniques for evaluating their utility. We presented results obtained from various methods on our 3 month test corpus, demonstrating that in general weighted word vector averages and the

Distributed Bag of Words model performs the best. We demonstrated the ability of these article vector representations to beat Naive Bayes at classification. We investigated their microscopic and macroscopic structure. Finally, we discussed various techniques for extending vector representations to authors, categories and readers, as well as their potential utility to diverse tasks of interest.

1.13 Acknowledgements

This work was supported by NSF IIS-1247696.

CHAPTER 2

SEGMENTATION

We explore the use of semantic word embeddings [62, 78, 54] in text segmentation algorithms, including the C99 segmentation algorithm [14, 15] and new algorithms inspired by distributed word vector representations. By developing a general framework for discussing a class of segmentation objectives, we study the effectiveness of greedy versus exact optimization approaches and suggest a new iterative refinement technique for improving the performance of greedy strategies. We compare our results to known benchmarks [86, 64, 14, 15], using known metrics [6, 80]. We demonstrate state-of-the-art performance for an untrained method with our Content Vector Segmentation (CVS) on the Choi test set. Finally, we apply the segmentation procedure to an in-the-wild dataset consisting of text extracted from scholarly articles in the arXiv.org database. [1]

2.1 Introduction

Segmenting text into naturally coherent sections has many useful applications in information retrieval and automated text summarization, and has received much attention in the past. An early text segmentation algorithm was the TextTiling method introduced by Hearst [36] in 1997. Text was scanned linearly, with a coherence calculated for each adjacent block, and a heuristic was used to determine the locations of cuts. In addition to linear approaches, there are text segmentation algorithms that optimize some scoring objective. An early algorithm in this class was Choi's C99 algorithm [14] in 2000, which also introduced a benchmark segmentation dataset used by subsequent work. Instead of

[1]This is a reproduction of the arχiv posting: 1503.05543

looking only at nearest neighbor coherence, the C99 algorithm computes a coherence score between all pairs of elements of text,[2] and searches for a text segmentation that optimizes an objective based on that scoring by greedily making a succession of best cuts. Later work by Choi and collaborators [15] used distributed representations of words rather than a bag of words approach, with the representations generated by LSA [22]. In 2001, Utiyama and Ishahara introduce a statistical model for segmentation and optimized a posterior for the segment boundaries. Moving beyond the greedy approaches, in 2004 Fragkou et al. [26] attempted to find the optimal splitting for their own objective using dynamic programming. More recent attempts at segmentation, including Misra et al. [64] and Riedl and Biemann [86], use LDA based topic models to inform the segmentation task. Du et al. consider structured topic models for segmentation [20]. Eisenstein and Barzilay [24] and Dadachev et al. [17] both consider a Bayesian approach to text segmentation. Most similar to our own work, Sakahara et al. [88] consider a segmentation algorithm which does affinity propagation clustering on text representations built from word vectors learned from word2vec [62].

For the most part, aside from [88], the non-topic model based segmentation approaches have been based on relatively simple representations of the underlying text. Recent approaches to learning word vectors, including Mikolov et al.'s word2vec [62], Pennington et al.'s GloVe [78] and Levy and Goldberg's pointwise mutual information [54], have proven remarkably successful in solving analogy tasks, machine translation [61], and sentiment analysis [78]. These word vector approaches attempt to learn a log-linear model for word-word co-occurrence statistics, such that the probability of two words (w, w') appearing

[2]By 'elements', we mean the pieces of text combined in order to comprise the segments. In the applications to be considered, the basic elements will be either sentences or words.

near one another is proportional to the exponential of their dot product,

$$P(w|w') = \frac{\exp(w \cdot w')}{\sum_v \exp(v \cdot w')} . \tag{2.1}$$

The method relies on these word-word co-occurrence statistics encoding meaningful semantic and syntactic relationships. Arora et al. [3] have shown how the remarkable performance of these techniques can be understood in terms of relatively mild assumptions about corpora statistics, which in turn can be recreated with a simple generative model.

Here we explore the utility of word vectors for text segmentation, both in the context of existing algorithms such as C99, and when used to construct new segmentation objectives based on a generative model for segment formation. We will first construct a framework for describing a family of segmentation algorithms, then discuss the specific algorithms to be investigated in detail. We then apply our modified algorithms both to the standard Choi test set and to a test set generated from arXiv.org research articles.

2.2 Text Segmentation

The segmentation task is to split a text into contiguous coherent sections. We first build a *representation* of the text, by splitting it into N basic elements, $\vec{V}_i$ ($i = 1, \ldots, N$), each a D-dimensional feature vector $V_{i\alpha}$ ($\alpha = 1, \ldots, D$) representing the element. Then we assign a *score* $\sigma(i, j)$ to each candidate segment, comprised of the i^{th} through $(j-1)^{\text{th}}$ elements, and finally determine how to *split* the text into the appropriate number of segments.

Denote a segmentation of text into K segments as a list of K indices $s =$

$(s_1, s_2, \cdots, s_K)$, where the k-th segment includes the elements $\vec{V}_i$ with $s_{k-1} \leq i < s_k$, with $s_0 \equiv 0$. For example, the string "aaabbcccdd" considered at the character level would be properly split with $s = (3, 5, 8, 10)$ into ("aaa", "bb", "ccc", "dd").

2.2.1 Representation

The text representation thus amounts to turning a plain text document T into an $(N \times D)$-dimensional matrix $\mathbf{V}$, with N the number of initial elements to be grouped into coherent segments and D the dimensionality of the element representation. For example, if segmenting at the word level then N would be the number of words in the text, and each word might be represented by a D-dimensional vector, such as those obtained from GloVe [78]. If segmenting instead at the sentence level, then N is the number of sentences in the text and we must decide how to represent each sentence.

There are additional preprocessing decisions, for example using a stemming algorithm or removing stop words before forming the representation. Particular preprocessing decisions can have a large effect on the performance of segmentation algorithms, but for discussing scoring functions and splitting methods those decisions can be abstracted into the specification of the $N \times D$ matrix $\mathbf{V}$.

2.2.2 Scoring

Having built an initial representation of the text, we next specify the coherence of a segment of text with a scoring function $\sigma(i, j)$, which acts on the representation $\mathbf{V}$ and returns a score for the segment running from i (inclusive) to j

(non-inclusive). The score can be a simple scalar or more general object. In addition to the scoring function, we need to specify how to return an aggregate score for the entire segmentation. This *score aggregation* function $\oplus$ can be as simple as adding the scores for the individual segments, or again some more general function. The score $S(s)$ for an overall segmentation is given by aggregating the scores of all of the segments in the segmentation:

$$S(s) = \sigma(0, s_1) \oplus \sigma(s_1, s_2) \oplus \cdots \oplus \sigma(s_{K-1}, s_K) \ . \tag{2.2}$$

Finally, to frame the segmentation problem as a form of optimization, we need to map the aggregated score to a single scalar. The *key* function ($[\![\cdot]\!]$) returns this single number, so that the cost for the above segmentation is

$$C(s) = [\![S(s)]\!] \ . \tag{2.3}$$

For most of the segmentation schemes to be considered, the score function itself returns a scalar, so the score aggregation function $\oplus$ will be taken as simple addition with the key function the identity, but the generality here allows us to incorporate the C99 segmentation algorithm [14] into the same framework.

2.2.3 Splitting

Having specified the representation of the text and scoring of the candidate segments, we need to prescribe how to choose the final segmentation. In this work, we consider three methods: (1) greedy splitting, which at each step inserts the best available segmentation boundary; (2) dynamic programming based segmentation, which uses dynamic programming to find the optimal segmentation;

and (3) an iterative refinement scheme, which starts with the greedy segmentation and then adjusts the boundaries to improve performance.

Greedy Segmentation

The greedy segmentation approach builds up a segmentation into K segments by greedily inserting new boundaries at each step to minimize the aggregate score:

$$s^0 = \{N\} \tag{2.4}$$

$$s^{t+1} = \arg\min_{i \in [1,N)} C(s^t \cup \{i\}) \tag{2.5}$$

until the desired number of splits is reached. Many published text segmentation algorithms are greedy in nature, including the original C99 algorithm [14].

Dynamic Programming

The greedy segmentation algorithm is not guaranteed to find the optimal splitting, but dynamic programming methods can be used for the text segmentation problem formulated in terms of optimizing a scoring objective. For a detailed account of dynamic programming and segmentation in general, see the thesis by Terzi [103]. Dynamic programming has been applied to text segmentation in Fragkou et al. [26], with much success, but we will also consider here an optimization of the C99 segmentation algorithm using a dynamic programming approach.

The goal of the dynamic programming approach is to split the segmentation problem into a series of smaller segmentation problems, by expressing the optimal segmentation of the first n elements of the sequence into k segments in terms of the best choice for the last segmentation boundary. The aggregated score $S(n,k)$ for this optimal segmentation should be minimized with respect to the key function $[\![\cdot]\!]$:

$$S(n,1) = \sigma(0,n) \tag{2.6}$$

$$S(n,k) = \overset{[\![\cdot]\!]}{\min_{l<n}} S(l,k-1) \oplus \sigma(l,n)\;. \tag{2.7}$$

While the dynamic programming approach yields the optimal segmentation for our decomposable score function, it can be costly to compute, especially for long texts. In practice, both the optimal segmentation score and the resulting segmentation can be found in one pass by building up a table of segmentation scores and optimal cut indices one row at a time.

Iterative Relaxation

Inspired by the popular Lloyd algorithm for k-means, we attempt to retain the computational benefit of the greedy segmentation approach, but realize additional performance gains by iteratively refining the segmentation. Since text segmentation problems require contiguous blocks of text, a natural scheme for relaxation is to try to move each segment boundary optimally while keeping the edges to either side of it fixed:

$$\begin{aligned} s_k^{t+1} &= \overset{[\![\cdot]\!]}{\operatorname*{arg\,min}_{l\in(s_{k-1}^t,\ s_{k+1}^t)}} \left(\sigma(0,s_1^t)\oplus\cdots\right. \\ &\qquad \left.\oplus\,\sigma(s_{k-1}^t,l)\oplus\sigma(l,s_{k+1}^t)\oplus\cdots\oplus\sigma(s_{K-1}^t,s_K^t)\right) && (2.8) \\ &= \overset{[\![\cdot]\!]}{\operatorname*{arg\,min}_{l\in(s_{k-1}^t,\ s_{k+1}^t)}} S\left(s^t - \{s_k^t\}\cup\{l\}\right) && (2.9) \end{aligned}$$

We will see in practice that by 20 iterations it has typically converged to a fixed point very close to the optimal dynamic programming segmentation.

2.3 Scoring Functions

In the experiments to follow, we will test various choices for the representation, scoring function, and splitting method in the above general framework. The segmentation algorithms to be considered fall into three groups:

2.3.1 C99 Segmentation

Choi's C99 algorithm [14] was an early text segmentation algorithm with promising results. The feature vector for an element of text is chosen as the pairwise cosine distances with other elements of text, where those elements in turn are represented by a bag of stemmed words vector (after preprocessing to remove stop words):

$$A_{ij} = \frac{\sum_w f_{i,w} f_{j,w}}{\sqrt{\sum_w f_{i,w}^2 \sum_w f_{j,w}^2}} , \tag{2.10}$$

with $f_{i,w}$ the frequency of word w in element i. The pairwise cosine distance matrix is noisy for these features, and since only the relative values are meaningful, C99 employs a ranking transformation, replacing each value of the matrix by the fraction of its neighbors with smaller value:

$$V_{ij} = \frac{1}{r^2 - 1} \sum_{\substack{i-r/2 \le l \le i+r/2 \\ l \neq i}} \sum_{\substack{j-r/2 \le m \le j+r/2 \\ m \neq j}} \left[A_{ij} > A_{lm} \right] , \tag{2.11}$$

where the neighborhood is an $r \times r$ block around the entry, the square brackets mean 1 if the inequality is satisfied otherwise 0 (and values off the end of the matrix are not counted in the sum, or towards the normalization). Each element of the text in the C99 algorithm is represented by a rank transformed vector of its cosine distances to each other element.

The score function describes the average inter-sentence similarity by taking the overall score to be

$$C(s) = \frac{\sum_k \beta_k}{\sum_k \alpha_k} , \tag{2.12}$$

where $\beta_k = \sum_{s_{k-1} \le i < s_k} \sum_{s_{k-1} \le j < s_k} V_{ij}$ is the sum of all ranked cosine similarities in a segment and $\alpha_k = (s_{k+1} - s_k)^2$ is the squared length of the segment. This score function is still decomposable, but requires that we define the local score function to return a pair,

$$\sigma(i, j) = \Big(\sum_{i \le k < j} \sum_{i \le k < j} V_{ij},\ (j - i)^2 \Big) , \tag{2.13}$$

with score aggregation function defined as component addition,

$$(\beta_1, \alpha_1) \oplus (\beta_2,\ \alpha_2) = (\beta_1 + \beta_2,\ \alpha_1 + \alpha_2) , \tag{2.14}$$

and key function defined as division of the two components,

$$[\![(\beta, \alpha)]\!] = \frac{\beta}{\alpha} . \tag{2.15}$$

While earlier work with the C99 algorithm considered only a greedy splitting approach, in the experiments that follow we will use our more general framework to explore both optimal dynamic programming and refined iterative versions of C99. Follow-up work by Choi et al. [15] explored the effect of using combinations of LSA word vectors in eqn. (2.10) in place of the $f_{i,w}$. Below we will explore the effect of using combinations of word vectors to represent the elements.

2.3.2 Average word vector

To assess the utility of word vectors in segmentation, we first investigate how they can be used to improve the C99 algorithm, and then consider more general scoring functions based on our word vector representation. As the representation of an element, we take

$$V_{ik} = \sum_w f_{iw} v_{wk} \ , \tag{2.16}$$

with f_{iw} representing the frequency of word w in element i, and v_{wk} representing the k^{th} component of the word vector for word w as learned by a word vector training algorithm, such as word2vec [62] or GloVe [78].

The length of word vectors varies strongly across the vocabulary and in general correlates with word frequency. In order to mitigate the effect of common words, we will sometimes weight the sum by the inverse document frequency (idf) of the word in the corpus:

$$V_{ik} = \sum_w f_{iw} \log \frac{|D|}{df_w} v_{wk} \ , \tag{2.17}$$

where df_w is the number of documents in which word w appears. We can instead normalize the word vectors before adding them together

$$V_{ik} = \sum_w f_{iw} \tilde{v}_{wk} \quad \tilde{v}_{wk} = \frac{v_{wk}}{\sqrt{\sum_k v_{wk}^2}} \ , \tag{2.18}$$

or both weight by idf and normalize.

Segmentation is a form of clustering, so a natural choice for scoring function is the sum of square deviations from the mean of the segment, as used in *k*-

means:

$$\sigma(i, j) = \sum_l \sum_k (V_{lk} - \mu_k(i, j))^2 \tag{2.19}$$

$$\text{where } \mu_k(i, j) = \frac{1}{j - i} \sum_{l=i}^{j-1} V_{lk} \,, \tag{2.20}$$

and which we call the Euclidean score function. Generally, however, cosine similarity is used for word vectors, making angles between words more important than distances. In some experiments, we therefore normalize the word vectors first, so that a euclidean distance score better approximates the cosine distance (recall $|\tilde{v} - \tilde{w}|_2^2 = |\tilde{v}|_2^2 + |\tilde{w}|_2^2 - 2\tilde{v} \cdot \tilde{w} = 2(1 - \tilde{v} \cdot \tilde{w})$ for normalized vectors).

2.3.3 Content Vector Segmentation (CVS)

Trained word vectors have a remarkable amount of structure. Analogy tasks such as man:woman::king:? can be solved by finding the vector closest to the linear query:

$$v_{\text{woman}} - v_{\text{man}} + v_{\text{king}} \,. \tag{2.21}$$

Arora et al. [3] constructed a generative model of text that explains how this linear structure arises and can be maintained even in relatively low dimensional vector models. The generative model consists of a content vector which undergoes a random walk from a stationary distribution defined to be the product distribution on each of its components c_k, uniform on the interval $[-\frac{1}{\sqrt{D}}, \frac{1}{\sqrt{D}}]$ (with D the dimensionality of the word vectors). At each point in time, a word vector is generated by the content vector according to a log-linear model:

$$P(w|c) = \frac{1}{Z_c} \exp(w \cdot c) \,, \quad Z_c = \sum_v \exp(v \cdot c) \,. \tag{2.22}$$

The slow drift of the content vectors helps to ensure that nearby words obey with high probability a log-linear model for their co-occurrence probability:

$$\log P(w, w') = \frac{1}{2d} \|v_w + v_{w'}\|^2 - 2 \log Z \pm o(1) , \tag{2.23}$$

for some fixed Z.

To segment text into coherent sections, we will boldly assume that the content vector in each putative segment is constant, and measure the log likelihood that all words in the segment are drawn from the same content vector c. (This is similar in spirit to the probabilistic segmentation technique proposed by Utiyama and Isahara [108].) Assuming the word draws $\{w_i\}$ are independent, we have that the log likelihood

$$\log P(\{w_i\}|c) = \sum_i \log P(w_i|c) \propto \sum_i w_i \cdot c \tag{2.24}$$

is proportional to the sum of the dot products of the word vectors w_i with the content vector c. We use a maximum likelihood estimate for the content vector:

$$\begin{aligned} c &= \arg\max_c \log P(c|\{w_i\}) && (2.25) \\ &= \arg\max_c \Big(\log P(\{w_i\}|c) + \log P(c) - \log P(\{w_i\})\Big) && (2.26) \\ &\propto \arg\max_c \sum w_i \cdot c \quad \text{s.t.} \quad -\frac{1}{\sqrt{D}} \leq c_k \leq \frac{1}{\sqrt{D}} . && (2.27) \end{aligned}$$

This determines what we will call the Content Vector Segmentation (CVS) algorithm, based on the score function

$$\sigma(i, j) = \sum_{i \leq l < j} \sum_k w_{lk} c_k(i, j) . \tag{2.28}$$

The score $\sigma(i, j)$ for a segment (i, j) is the sum of the dot products of the word vectors w_{lk} with the maximum likelihood content vector $c(i, j)$ for the segment, with components given by

$$c_k(i, j) = \operatorname{sign}\left(\sum_{i \leq l < j} w_{l,k}\right) \frac{1}{\sqrt{D}} . \tag{2.29}$$

The maximum likelihood content vector thus has components $\pm\frac{1}{\sqrt{D}}$, depending on whether the sum of the word vector components in the segment is positive or negative.

This score function will turn out to generate some of the most accurate segmentation results. Note that CVS is completely untrained with respect to the specific text to be segmented, relying only on a suitable set of word vectors, derived from some corpus in the language of choice. While CVS is most justifiable when working on the word vectors directly, we will also explore the effect of normalizing the word vectors before applying the objective.

2.4 Experiments

To explore the efficacy of different segmentation strategies and algorithms, we performed segmentation experiments on two datasets. The first is the Choi dataset [14], a common benchmark used in earlier segmentation work, and the second is a similarly constructed dataset based on articles uploaded to the arχiv, as will be described in Section 2.4.3. All code and data used for these experiments is available online[3].

2.4.1 Evaluation

To evaluate the performance of our algorithms, we use two standard metrics: the P_k metric and the WindowDiff (WD) metric. For text segmentation, near misses should get more credit than far misses. The P_k metric [6], captures the

[3]`github.com/alexalemi/segmentation`

probability for a probe composed of a pair of nearby elements (at constant distance positions $(i, i+k)$) to be placed in the same segment by both reference and hypothesized segmentations. In particular, the P_k metric counts the number of disagreements on the probe elements:

$$P_k = \frac{1}{N-k} \sum_{i=1}^{N-k} [\delta_{\text{hyp}}(i, i+k) \neq \delta_{\text{ref}}(i, i+k)] \tag{2.30}$$

$$k \underset{\substack{\text{nearest}\\\text{integer}}}{=} \frac{1}{2} \frac{\#\,\text{elements}}{\#\,\text{segments}} - 1\,,$$

where $\delta(i, j)$ is equal to 1 or 0 according to whether or not both element i and j are in the same segment in hypothesized and reference segmentations, resp., and the argument of the sum tests agreement of the hypothesis and reference segmentations. (k is taken to be one less than the integer closest to half of the number of elements divided by the number of segments in the reference segmentation.) The total is then divided by the total number of probes. This metric counts the number of disagreements, so lower scores indicate better agreement between the two segmentations. Trivial strategies such as choosing only a single segmentation, or giving each element its own segment, or giving constant boundaries or random boundaries, tend to produce values of around 50% [6].

The P_k metric has the disadvantage that it penalizes false positives more severely than false negatives, and can suffer when the distribution of segment sizes varies. Pevzner and Hearst [80] introduced the WindowDiff (WD) metric:

$$WD = \frac{1}{N-k} \sum_{i=1}^{N-k} [b_{\text{ref}}(i, i+k) \neq b_{\text{hyp}}(i, i+k)]\,, \tag{2.31}$$

where $b(i, j)$ counts the number of boundaries between location i and j in the text, and an error is registered if the hypothesis and reference segmentations disagree on the number of boundaries. In practice, the P_k and WD scores are highly correlated, with P_k more prevalent in the literature — we will provide both for most of the experiments here.

2.4.2 Choi Dataset

The Choi dataset is used to test whether a segmentation algorithm can distinguish natural topic boundaries. It concatenates the first *n* sentences from ten different documents chosen at random from a 124 document subset of the Brown corpus (the `ca**.pos` and `cj**.pos` sets) [14]. The number of sentences *n* taken from each document is chosen uniformly at random within a range specified by the subset id (i.e., as min–max #sentences). There are four ranges considered: (3–5, 6–8, 9–11, 3–11), the first three of which have 100 example documents, and the last 400 documents. The dataset can be obtained from an archived version of the C99 segmentation code release[4]. An extract from one of the documents in the test set is shown in Fig. 2.1.

C99 benchmark

We will explore the effect of changing the representation and splitting strategy of the C99 algorithm. In order to give fair comparisons we implemented our own version of the C99 algorithm (oC99). The C99 performance depended sensitively on the details of the text preprocessing. Details can be found in Appendix A.

Effect of word vectors on C99 variant

The first experiment explores the ability of word vectors to improve the performance of the C99 algorithm. The word vectors were learned by GloVe [78] on a

[4] `http://web.archive.org/web/20010422042459/http://www.cs.man.ac.uk/~choif/software/C99-1.2-release.tgz` (We thank with Martin Riedl for pointing us to the dataset.)

```
1  ==========
2  Some of the features of the top portions of Figure 1 and Figure 2 were mentioned in
       discussing Table 1 .
3  First , the Onset Profile spreads across approximately 12 years for boys and 10 years
       for girls .
4  In contrast , 20 of the 21 lines in the Completion Profile ( excluding center 5 for
       boys and 4 for girls ) are bunched and extend over a much shorter period ,
       approximately 30 months for boys and 40 months for girls .
5  The Maturity Chart for each sex demonstrates clearly that Onset is a phenomenon of
       infancy and early childhood whereas Completion is a phenomenon of the later portion
        of adolescence .
6  ==========
7  The many linguistic techniques for reducing the amount of dictionary information that
       have been proposed all organize the dictionary 's contents around prefixes , stems
       , suffixes , etc .
8  .
9  A significant reduction in the voume of store information is thus realized , especially
        for a highly inflected language such as Russian .
10 For English the reduction in size is less striking .
11 This approach requires that : ( 1 ) each text word be separated into smaller elements
       to establish a correspondence between the occurrence and dictionary entries , and (
        2 ) the information retrieved from several entries in the dictionary be
       synthesized into a description of the particular word .
12 ==========
```

Figure 2.1: Example of two segments from the Choi dataset, taken from an entry in the 3–5 set. Note the appearance of a "sentence" with the single character "." in the second segment on line 8. These short sentences can confound the benchmarks.

42 billion word set of the Common Crawl corpus in 300 dimensions[5]. We emphasize that these word vectors were not trained on the Brown or Choi datasets directly, and instead come from a general corpus of English. These vectors were chosen in order to isolate any improvement due to the word vectors from any confounding effects due to details of the training procedure. The results are summarized in Table 2.1 below. The upper section cites results from [15], exploring the utility of using LSA word vectors, and showed an improvement of a few percent over their baseline C99 implementation. The middle section shows results from [86], which augmented the C99 method by representing each element with a histogram of topics learned from LDA. Our results are in the lower

[5]Obtainable from `http://www-nlp.stanford.edu/data/glove.42B.300d.txt.gz`

section, showing how word vectors improve the performance of the algorithm.

	P_k				WD			
Algorithm	3–5	6–8	9–11	3–11	3–5	6–8	9–11	3–11
C99 [15]	12	11	9	9				
C99LSA	9	10	7	5				
C99 [86]	11.20				12.07			
C99LDA	4.16				4.89			
oC99	14.22	**12.20**	**11.59**	15.56	14.22	**12.22**	**11.60**	15.64
oC99tf	12.14	13.17	14.60	14.91	12.14	13.34	15.22	15.22
oC99tfidf	**10.27**	12.23	15.87	**14.78**	**10.27**	12.30	16.29	**14.96**
oC99k50	20.39	21.13	23.76	24.33	20.39	21.34	23.26	24.63
oC99k200	18.60	17.37	19.42	20.85	18.60	17.42	19.60	20.97

Table 2.1: Effect of using word vectors in the C99 text segmentation algorithm. P_k and WD results are shown (smaller values indicate better performance). The top section (C99 vs. C99LSA) shows the few percent improvement over the C99 baseline reported in [15] of using LSA to encode the words. The middle section (C99 vs. C99LDA) shows the effect of modifying the C99 algorithm to work on histograms of LDA topics in each sentence, from [86]. The bottom section shows the effect of using word vectors trained from GloVe [78] in our oC99 implementation of the C99 segmentation algorithm. The oC99tf implementation sums the word vectors in each sentence, with no rank transformation, after removing stop words and punctuation. oC99tfidf weights the sum by the log of the inverse document frequency of each word. The oC99k models use the word vectors to form a topic model by doing spherical *k*-means on the word vectors. oC99k50 uses 50 clusters and oC99k200 uses 200.

In each of these last experiments, we turned off the rank transformation, pruned the stop words and punctuation, but did not stem the vocabulary. Word vectors can be incorporated in a few natural ways. Vectors for each word in a sentence can simply be summed, giving results shown in the oC99tf row. But all words are not created equal, so the sentence representation might be dominated by the vectors for common words. In the oC99tfidf row, the word vectors are weighted by $\text{idf}_i = \log \frac{500}{\text{df}_i}$ (i.e., the log of the inverse document frequency of each word in the Brown corpus, which has 500 documents in total) before summation. We see some improvement from using word vectors, for example the P_k of 14.78% for the oC99tfidf method on the 3–11 set, compared to P_k of 15.56% for our baseline C99 implementation. On the shorter 3–5 test set, our oC99tfidf method achieves P_k of 10.27% versus the baseline oC99 P_k of 14.22% . To compare to the various topic model based approaches, e.g. [86], we perform spherical k-means clustering on the word vectors [16] and represent each sentence as a histogram of its word clusters (i.e., as a vector in the space of clusters, with components equal to the number of its words in that that cluster). In this case, the word topic representations (oC99k50 and oC99k200 in Table 2.1) do not perform as well as the C99 variants of [86]. But as was noted in [86], those topic models were trained on cross-validated subsets of the Choi dataset, and benefited from seeing virtually all of the sentences in the test sets already in each training set, so have an unfair advantage that would not necessarily convey to real world applications. Overall, the results in Table 2.1 illustrate that the word vectors obtained from GloVe can markedly improve existing segmentation algorithms.

Alternative Scoring frameworks

The use of word vectors permits consideration of natural scoring functions other than C99-style segmentation scoring. The second experiment examines alternative scoring frameworks using the same GloVe word vectors as in the previous experiment. To test the utility of the scoring functions more directly, for these experiments we used the optimal dynamic programming segmentation. Results are summarized in Table 2.2, which shows the average P_k and WD scores on the 3–11 subset of the Choi dataset. In all cases, we removed stop words and punc-

Algorithm	rep	n	P_k	WD
oC99	tf	-	11.78	11.94
	tfidf	-	12.19	12.27
Euclidean	tf	F	7.68	8.28
		T	9.18	10.83
	tfidf	F	12.89	14.27
		T	8.32	8.95
Content (CVS)	tf	F	5.29	5.39
		T	5.42	5.55
	tfidf	F	5.75	5.87
		T	5.03	5.12

Table 2.2: Results obtained by varying the scoring function. These runs were on the 3–11 set from the Choi database, with a word cut of 5 applied, after preprocessing to remove stop words and punctuation, but without stemming. The CVS method does remarkably better than either the C99 method or a Euclidean distance-based scoring function.

tuation, did not stem, but after preprocessing removed sentences with fewer than 5 words.

Note first that the dynamic programming results for our implementation of C99 with tf weights gives $P_k = 11.78\%$, 3% better than the greedy version result of 14.91% reported in Table 2.1. This demonstrates that the original C99 algorithm and its applications can benefit from a more exact minimization than given by the greedy approach. We considered two natural score functions: the Euclidean scoring function (eqn. (2.20)) which minimizes the sum of the square deviations of each vector in a segment from the average vector of the segment, and the Content Vector scoring (CVS) (eqn. (2.28) of section 2.3.3), which uses an approximate log posterior for the words in the segment, as determined from its maximum likelihood content vector. In each case, we consider vectors for each sentence generated both as a strict sum of the words comprising it (tf approach), and as a sum weighted by the log idf (tfidf approach, as in sec. 2.4.2). Additionally, we consider the effect of normalizing the element vectors before starting the score minimization, as indicated by the n column.

The CVS score function eqn. (2.28) performs the best overall, with P_k scores below 6%, indicating an improved segmentation performance using a score function adapted to the choice of representation. While the most principled score function would be the Content score function using tf weighted element vectors without normalization, the normalized tfidf scheme actually performs the best. This is probably due to the uncharacteristically large effect common words have on the element representation, which the log idf weights and the normalization help to mitigate.

Strictly speaking, the idf weighted schemes cannot claim to be completely untrained, as they benefit from word usage statistics in the Choi test set, but the raw CVS method still demonstrates a marked improvement on the 3–11 subset, 5.29% P_k versus the optimal C99 baseline of 11.78% P_k.

Effect of Splitting Strategy

To explore the effect of the splitting strategy and to compare with our overall results on the Choi test set against other published benchmarks, in our third experiment we ran the raw CVS method against all of the Choi test subsets, using all three splitting strategies discussed: greedy, refined, and dynamic programming. These results are summarized in Table 2.3.

Overall, our method outperforms all previous untrained methods. As commented regarding Table 2.1 (toward the end of subsection 2.4.2), we have included the results of the topic modeling based approaches M09 [64], R12 [86] and D13 [17] for reference. But due to repeat appearance of the same sentences

Alg	3–5	6–8	9–11	3–11
TT [14]	44	43	48	46
C99 [14]	12	9	9	12
C01 [15]	10	7	5	9
U00 [108]	9	7	5	10
F04 [26]	5.5	3.0	1.3	7.0
G-CVS	5.14	4.82	6.38	6.49
R-CVS	3.92	3.75	5.17	5.65
DP-CVS	3.41	3.45	4.45	**5.29**
M09 [64]	2.2	2.3	4.1	2.3
R12 [86]	1.24	0.76	0.56	0.95
D13 [20]	1.0	0.9	1.2	0.6

Table 2.3: Some published P_k results on the Choi dataset against our raw CVS method. G-CVS uses a greedy splitting strategy, R-CVS uses up to 20 iterations to refine the results of the greedy strategy, and DP-CVS shows the optimal results obtained by dynamic programming. We include the topic modeling results M09, R12 and D13 for reference, but for reasons detailed in the text do not regard them as comparable, due to their mingling of test and training samples.

throughout each section of the Choi dataset, methods that split that dataset into test and training sets have unavoidable access to the entirety of the test set during training, albeit in different order.[6] These results can therefore only be compared to other algorithms permitted to make extensive use of the test data during cross-validation training. Only the TT, C99, U00 and raw CVS method can be considered as completely untrained. The C01 method derives its LSA vectors from the Brown corpus, from which the Choi test set is constructed, but that provides only a weak benefit, and the F04 method is additionally trained on a subset of the test set to achieve its best performance, but its use only of idf values provides a similarly weak benefit.

We emphasize that the raw CVS method is completely independent of the Choi test set, using word vectors derived from a completely different corpus. In

[6]In [86], it is observed that “This makes the Choi data set artificially easy for supervised approaches.” See appendix B.

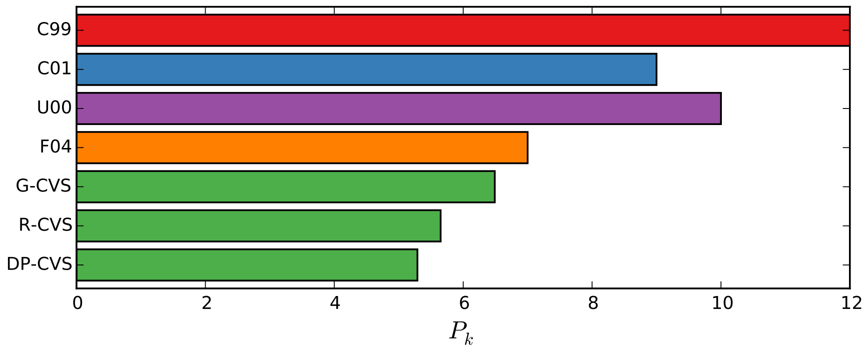


Figure 2.2: Results from last column of Table 2.3 reproduced to highlight the performance of the CVS segmentation algorithm compared to similar untrained algorithms. Its superior performance in an unsupervised setting suggests applications on documents “in the wild”.

Fig. 2.2, we reproduce the relevant results from the last column of Table 2.1 to highlight the performance benefits provided by the semantic word embedding.

Note also the surprising performance of the refined splitting strategy, with the R-CVS results in Table 2.3 much lower than the greedy G-CVS results, and moving close to the optimal DP-CVS results, at far lower computational cost. In particular, taking the dynamic programming segmentation as the true segmentation, we can assess the performance of the refined strategy. As seen in Table 2.4, the refined segmentation very closely approximates the optimal segmentation. This is important in practice since the dynamic programming segmentation is much slower, taking five times longer to compute on the 3–11 subset of the Choi test set. The dynamic programming segmentation becomes computationally infeasible to do at the scale of word level segmentation on the arχiv dataset considered in the next section, whereas the refined segmentation method remains eminently feasible.

	3–5	6–8	9–11	3–11
R-CVS vs DP-CVS [14]	0.90	0.65	1.16	1.15

Table 2.4: Treating the dynamic programming splits as the true answer, the error of the refined splits as measured in P_k across the subsets of the Choi test set.

2.4.3 Arχiv Dataset

Performance evaluation on the Choi test set implements segmentation at the sentence level, i.e., with segments of composed of sentences as the basic elements. But text sources do not necessarily have well-marked sentence boundaries. The arχiv is a repository of scientific articles which for practical reasons extracts text from PDF documents (typically using `pdfminer/pdf2txt.py`). That Postscript-based format was originally intended only as a means of formatting text on a page, rather than as a network transmission format encoding syntactic or semantic information. The result is often somewhat corrupted, either due to the handling of mathematical notation, the presence of footers and headers, or even just font encoding issues.

To test the segmentation algorithms in a realistic setting, we created a test set similar to the Choi test set, but based on text extracted from PDFs retrieved from the arχiv database. Each test document is composed of a random number of contiguous words, uniformly chosen between 100 and 300, sampled at random from the text obtained from arχiv articles. The text was preprocessed by lowercasing and inserting spaces around every non-alphanumeric character, then splitting on whitespace to tokenize. An example of two of the segments of the first test document is shown in Figure 2.3 below.

This is a much more difficult segmentation task: due to the presence of num-

```
1 . nature_414 : 441 - 443 . 12 seinen , i . and schram a . 2006 . social_status and
    group norms : indirect_reciprocity in a helping experiment .
    european_economic_review 50 : 581 - 602 . silva , e . r . , jaffe , k . 2002 .
    expanded food choice as a possible factor in the evolution of eusociality in
    vespidae sociobiology 39 : 25 - 36 . smith , j . , van dyken , j . d . , zeejune ,
    p . c . 2010 . a generalization of hamilton ’ s rule for the evolution of microbial
     cooperation science_328 , 1700 - 1703 . zhang , j . , wang , j . , sun , s . ,
    wang , l . , wang , z . , xia , c . 2012 . effect of growing size of interaction
    neighbors on the evolution of cooperation in spatial snowdrift_game .
    chinese_science bulletin 57 : 724 - 728 . zimmerman , m . , egu ‘i luz , v . ,
    san_miguel ,
2 of ) e , equipped_with the topology of weak_convergence . we will state some results
    about random measures . 10 definition a . 1 ( first two moment measures ) . for a
    random_variable z , taking values in p ( e ) , and k = 1 , 2 , . . . , there is a
    uniquely_determined measure μ ( k ) on b ( ek ) such that e [ z ( a1 ) ·_·_· z ( ak )
     ] = μ ( k ) ( a1 × ·_·_· × ak ) for a1 , . . . , ak ∈ b ( e ) . this is called the
    kth_moment measure . equivalently , μ ( k ) is the unique measure such that e [ hz
    , ϕ 1i ·_·_· hz , ϕ ki ] = h μ ( k ) , ϕ 1 ·_·_· ϕ ki , where h . , . i denotes
    integration . lemma a . 2 ( characterisation of deterministic random measures ) .
    let z be a random_variable_taking values in p ( e ) with the first two moment
    measures μ : = μ ( 1 ) and μ ( 2 ) . then the following_assertions_are_equivalent :
     1 . there is ν ∈ p ( e ) with z = ν , almost_surely . 2 . the second_moment
    measure has product - form , i . e . μ ( 2 ) = μ ⊗ μ ( which is equivalent to e [
    hz , ϕ 1i · hz , ϕ 2i ] = h μ , ϕ 1i · h μ , ϕ 2i ( this is in fact equivalent to e
    [ hz , ϕ i2 ]
```

Figure 2.3: Example of two of the segments from a document in the arχiv test set.

bers and many periods in references, there are no clear sentence boundaries on which to initially group the text, and no natural boundaries are suggested in the test set examples. Here segmentation algorithms must work directly at the "word" level, where word can mean a punctuation mark. The presence of garbled mathematical formulae adds to the difficulty of making sense of certain streams of text.

In Table 2.5, we summarize the results of three word vector powered approaches, comparing a C99 style algorithm to our content vector based methods, both for unnormalized and normalized word vectors. Since much of the language of the scientific articles is specialized, the word vectors used in this case were obtained from GloVe trained on a corpus of similarly preprocessed

Alg	S	P_k	WD
oC99	G	47.26	47.26
oC99	R	47.06	49.16
CVS	G	26.07	28.23
CVS	R	25.55	27.73
CVSn	G	24.63	26.69
CVSn	R	**24.03**	**26.15**

Table 2.5: Results on the arχiv test set for the C99 method using word vectors (oC99), our CVS method, and CVS method with normalized word vectors (CVSn). The P_k and WD metrics are given for both the greedy (G) and refined splitting strategies (R), with respect to the reference segmentation in the test set. The refined strategy was allowed up to 20 iterations to converge. The refinement converged for all of the CVS runs, but failed to converge for some documents in the test set under the C99 method. Refinement improved performance in all cases, and our CVS methods improve significantly over the C99 method for this task.

texts from 98,392 arχiv articles. (Since the elements are now words rather than sentences, the only issue involves whether or not those word vectors are normalized.) As mentioned, the dynamic programming approach is prohibitively expensive for this dataset.

We see that the CVS method performs far better on the test set than the C99 style segmentation using word vectors. The P_k and WD values obtained are not as impressive as those obtained on the Choi test set, but this test set offers a much more challenging segmentation task: it requires the methods to work at the level of words, and as well includes the possibility that natural topic boundaries occur in the test set segments themselves. The segmentations obtained with the CVS method typically appear sensibly split on section boundaries, references and similar formatting boundaries, not known in advance to the algorithm.

As a final illustration of the effectiveness of our algorithm at segmenting

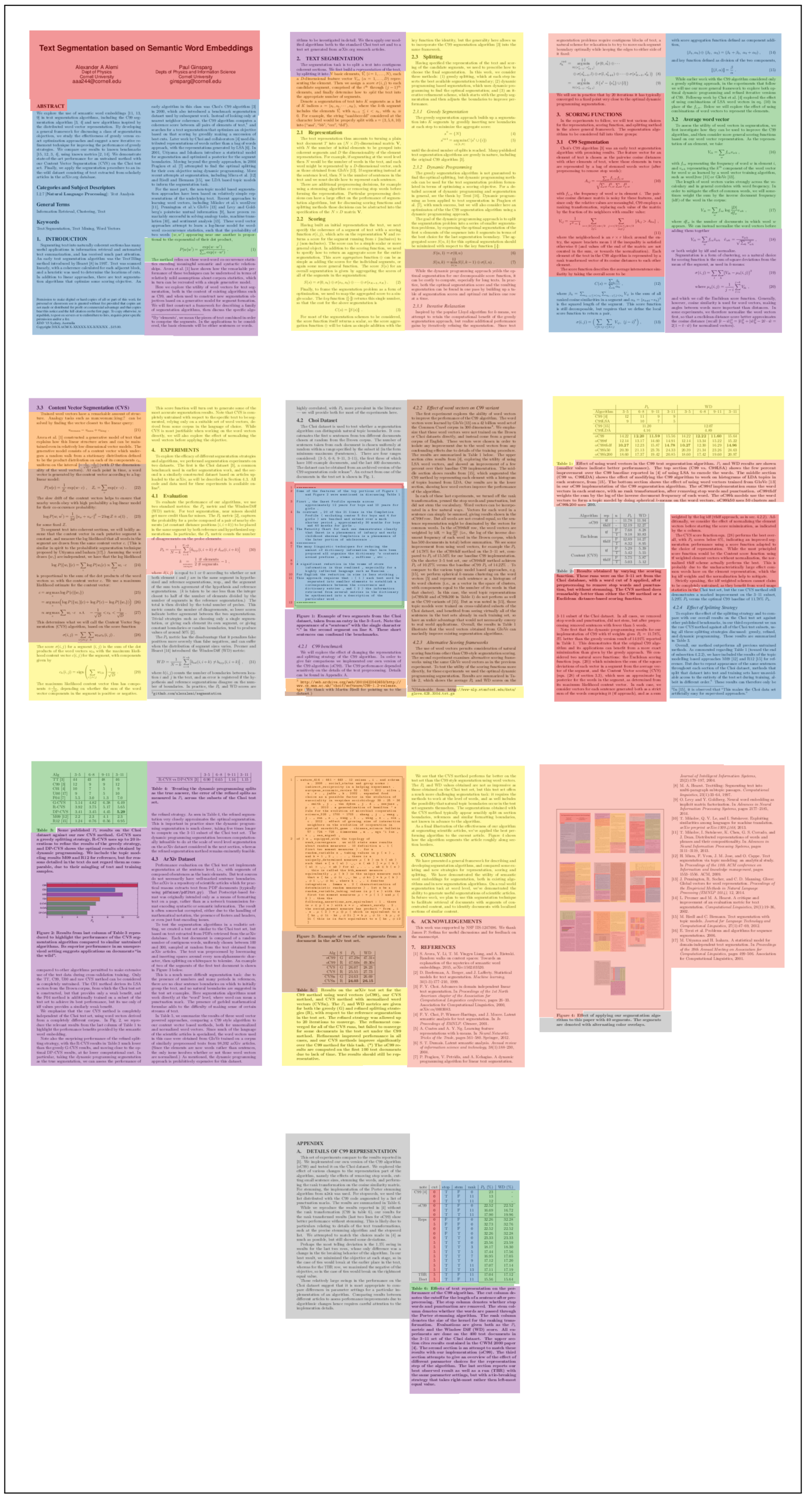


Figure 2.4: Effect of applying our segmentation algorithm to this paper with 40 segments. The segments are denoted with alternating color overlays.

scientific articles, we've applied the best performing algorithm to the current article. Figure 2.4 shows how the algorithm segments the article roughly along section borders.

2.5 Conclusion

We have presented a general framework for describing and developing segmentation algorithms, and compared some existing and new strategies for representation, scoring and splitting. We have demonstrated the utility of semantic word embeddings for segmentation, both in existing algorithms and in new segmentation algorithms. On a real world segmentation task at word level, we've demonstrated the ability to generate useful segmentations of scientific articles. In future work, we plan to use this segmentation technique to facilitate retrieval of documents with segments of concentrated content, and to identify documents with localized sections of similar content.

2.6 Acknowledgements

This work was supported by NSF IIS-1247696. We thank James P. Sethna for useful discussions and for feedback on the manuscript.

CHAPTER 3

ZOMBIES

We use a popular fictional disease, zombies, in order to introduce techniques used in modern epidemiology modelling, and ideas and techniques used in the numerical study of critical phenomena. We consider variants of zombie models, from fully connected continuous time dynamics to a full scale exact stochastic dynamic simulation of a zombie outbreak on the continental United States. Along the way, we offer a closed form analytical expression for the fully connected differential equation, and demonstrate that the single person per site two dimensional square lattice version of zombies lies in the percolation universality class. We end with a quantitative study of the full scale US outbreak, including the average susceptibility of different geographical regions. [1]

3.1 Introduction

Zombies captivate the imagination. The idea of a deadly disease that not only kills its hosts, but turns those hosts into deadly vectors for the disease is scary enough to fuel an entire genre of horror stories and films. But at its root, zombism is just that – a (fictional) disease – and so should be amenable to the same kind of analysis and study that we use to combat more traditional diseases.

Much scholarly attention has focused on more traditional human diseases [44], but recently, academic attention has turned a bit of thought onto zombies as a unique and interesting modification of classic disease models. One of the first

[1]This is a reproduction of the arχiv posting: 1503.01104

academic accounts of zombies was the 2009 article by Munz et al. [69], in which an early form of a compartmental model of zombism was introduced. Since then, there have been several interesting papers published including works that perform Bayesian estimations of the zombie disease parameters [114], look at how emotional factors impact the spread of zombies [73], using zombies to gain insight into models of politics [40], or into the interaction of a zombie epidemic and social dynamics [89, 68]. Additional essays can be found in two books collecting academic essays centered around zombism [11, 96]

Besides the academic papers, zombies have seen a resurgence in fiction. Of particular note are the works of Max Brooks, including a detailed *Zombie Survival Guide* [8], as well as an oral history of the first zombie war [9] in a hypothesized post outbreak world. In both these works Brooks provides a rich source of information about zombies and their behavior. In particular, he makes the connection to disease explicit, describing zombies as the result of a hypothetical virus, *Solanum*.

Zombies form a wonderful model system to illustrate modern epidemiological tools drawn from statistical mechanics, computational chemistry, and mathematical modeling. They also form an ideal vehicle for public outreach: the Center for Disease Control uses preparation for a zombie apocalypse [74, 75] to promote emergency preparedness. In this work, we will build up to a full-scale simulation of a zombie outbreak in the continental United States, with realistic values drawn from the literature and popular culture (section 3.5, simulation accessible online [2]). Before that, we shall use statistical mechanics to scrutinize the threshold of zombie virulence that determines whether humanity survives (section 3.4). Preceding that, we shall show how methods from computational

[2]`http://mattbierbaum.github.io/zombies-usa/`

chemistry can be used to simulate every individual heroic encounter between a human and a zombie (section 3.3). But we begin by describing and analyzing a simple model of zombies (the SZR model) – the simplest and most natural generalization to the classic SIR (Susceptible-Infected-Recovered) model used to describe infectious disease spread in epidemiology.

3.2 SZR Model

We start with a simple model of zombies, the SZR model. There are three compartments in the model: S represents the susceptible population, the uninfected humans; Z represents the infected state, zombies; and R represents our removed state, in this case zombies that have been terminated by humans (canonically by destroying their brain so as to render them inoperable). There are two transitions possible: a human can become infected if they are bitten by a zombie, and a zombie can be destroyed by direct action by a human. There are two parameters governing these transitions: β, the bite parameter determines the rate at which a zombie will bite a human if they are in contact, and κ the kill parameter that gives the rate that a human kills the zombie. Rendered as a system of coupled differential equations, we obtain, for a particular interaction site:

$$\dot{S} = -\beta S Z \tag{3.1}$$

$$\dot{Z} = (\beta - \kappa) S Z \tag{3.2}$$

$$\dot{R} = \kappa S Z \tag{3.3}$$

Notice that these interactions are *density dependent*, in the sense that the rate at which we convert humans to zombies and kill zombies is dependent on the total count of zombies and humans in this site. This is in contrast with most models of human diseases, which frequently adopt *frequency dependent* interactions wherein S, Z, R would have been interpreted as the fraction of the population in the corresponding state.

This distinction will become stark once we consider large simulations with very inhomogeneous populations. By claiming that zombies can be modeled by a single bite parameter β that itself is a rate per person per unit time, we are claiming that a zombie in a block with 5,000 people would be one hundred times as effective at infecting new zombies as a zombie in a block with fifty people; similarly the zombie in question would be killed one hundred times faster. This would seem false for an ordinary disease like the flu, but in the case of zombies, we argue that it is appropriate. Zombies directly seek out hosts to infect, at which point the human and zombie engage in a duel to the (un)death.

To facilitate analysis we can nondimensionalize the equations by choosing a relevant population size N, and recasting in terms of the dimensionless time parameter $\tau = t\beta N$ and dimensionless virulence $\alpha = \kappa/\beta$

$$\begin{aligned} \frac{dS}{d\tau} &= -\frac{SZ}{N} \\ \frac{dZ}{d\tau} &= (1-\alpha)\frac{SZ}{N} \\ \frac{dR}{d\tau} &= \alpha\frac{SZ}{N} \end{aligned} \tag{3.4}$$

Unlike a traditional disease (e.g., as modeled by SIR), for the zombie model, we have a stable configuration when either the human or the zombie population is defeated ($S = 0$ or $Z = 0$). Furthermore, unlike SIR, SZR admits an analytical solution, assuming $R(0) = 0$, and with $Z_0 \equiv Z(0), S_0 \equiv S(0)$:

$$P \equiv Z_0 + (1 - \alpha)S_0 \tag{3.5}$$

$$\mu \equiv \frac{S_0}{Z_0}(1 - \alpha) = \frac{P}{Z_0} - 1 \tag{3.6}$$

$$f(\tau) \equiv \frac{P\mu}{e^{\tau P/N} + \mu} \tag{3.7}$$

$$Z(\tau) = P - f(\tau) \tag{3.8}$$

$$S(\tau) = \frac{f(\tau)}{1 - \alpha} \tag{3.9}$$

Given the analytical solution, it is clear to see that the sign of P governs whether there will eventually be humans or zombies in the final state. If $\alpha < 1, P > 0$, so

$$\lim_{\tau \to \infty} f(\tau) = 0 \tag{3.10}$$

$$\lim_{\tau \to \infty} Z(\tau) = P = Z_0 + (1 - \alpha)S_0 \tag{3.11}$$

$$\lim_{\tau \to \infty} S(\tau) = 0 \tag{3.12}$$

and the system will always flow to a final state composed of entirely zombies and no humans, where P denotes the number of zombies that survive.

If however, $\alpha > 1$, humans are more effective at killing zombies than zombies are at biting humans. With enough zombies in the initial state, we can still convert all of the humans before they have time to kill all of the zombies.

We can recast the dynamics in terms of the variables $P \equiv Z + (1 - \alpha)S$ and $\chi = S/Z$ to gain further insights. First note that:

$$\frac{dP}{d\tau} = P' = Z' + (1 - \alpha)S' \tag{3.13}$$

$$= (1 - \alpha)\frac{SZ}{N} - (1 - \alpha)\frac{SZ}{N} = 0 \tag{3.14}$$

so P is a constant of the dynamics. As for χ:

$$\chi' = \frac{S'}{Z} - \frac{SZ'}{Z^2} \tag{3.15}$$

$$= -\frac{S}{N} - (1-\alpha)\frac{S}{N}\frac{S}{Z} \tag{3.16}$$

$$= -\frac{S}{N}\left(1 + (1-\alpha)\right)\chi \tag{3.17}$$

$$= -\frac{P}{N}\chi \tag{3.18}$$

Hence if we choose $N = |P|$, we end up with the very simple dynamics:

$$P'(\tau) = 0 \tag{3.19}$$

$$P(\tau) = P_0 = Z(\tau) + (1-\alpha)S(\tau) = Z_0 + (1-\alpha)S_0 \tag{3.20}$$

$$\chi'(\tau) = \begin{cases} -\chi & P > 0 \\ +\chi & P < 0 \end{cases} \tag{3.21}$$

$$\chi(\tau) = \frac{S(\tau)}{Z(\tau)} = \chi_0 \begin{cases} e^{-\tau} & P > 0 \\ e^{+\tau} & P < 0 \end{cases} \tag{3.22}$$

$$\chi_0 \equiv \frac{S_0}{Z_0} \tag{3.23}$$

Here we see that the dynamics is simply an exponential decay or increase in the ratio of humans to zombies $\chi = S/Z$. The final populations in either case are easy to see due to the conservation of P. If zombies win we have

$$Z_\infty = Z_0 + (1-\alpha)S_0 \tag{3.24}$$

And if humans win

$$S_\infty = S_0 - \frac{Z_0}{\alpha - 1} \tag{3.25}$$

SIR model

This dynamics should be compared to the similarly nondimensionlized density-dependent SIR model:

$$\frac{dS}{d\tau} = -\frac{SI}{N} \tag{3.26}$$

$$\frac{dI}{d\tau} = \left(\frac{S}{N} - \mu\right) I \tag{3.27}$$

$$\frac{dR}{d\tau} = \mu I \tag{3.28}$$

Here $\tau = t\beta N$ as above, but $\mu = \nu/(\beta N) = R_0^{-1}$, because in the SIR model our infected population recovers on its own. This is contrasted with SZR, where the process of infection and recovery have the same functional form, depending on the product SZ. This μ is the inverse of the usual R_0 parameter used to denote the infectivity of the SIR model, here used to make a closer analogy to the SZR model. It is this parameter that principally governs whether we have an outbreak or not. Unlike the α parameter for SZR which depends only on our disease constants β, κ, the relevant virulence for the density dependent SIR model (μ) has a population dependence.

Notice again that while the only stable configuration for the SIR model is when there is no infected population ($I = 0$), the SZR model is stable when either the humans or zombies are depleted ($S = 0$ or $Z = 0$).

The SIR model does not admit a closed form analytical solution, but we can find a parametric solution by dividing the first equation by the third, revealing.

$$S(\tau) = S_0 e^{-\frac{(R(\tau) - R_0)}{\mu N}} \tag{3.29}$$

Using the observation that in the limit of infinite time, no infected population can persist, we can choose N to be the total population

$$S_0 + I_0 + R_0 = N = S_\infty + R_\infty \tag{3.30}$$

and so obtain a transcendental equation for the recovered population at long times.

$$R_\infty = N - S_0 e^{-\frac{(R_\infty - R_0)}{\mu N}} \tag{3.31}$$

Unlike the SZR model, here we see that no matter how virulent the disease is, the epidemic will be self-limiting, and there will always have some susceptibles left at the end of the outbreak. This is a sharp qualitative difference between zombies and more traditional SIR models, arising from the fact that the "recovery" of zombies is itself dependent on the presence of susceptibles.

To visually compare the difference, in Figure 3.1 we have shown deterministic trajectories for both SIR and SZR for selected parameter values.

3.3 Stochastic simulation

While most previous studies modeling zombie population dynamics have been deterministic, things get more interesting when we try to model discrete populations. By treating the number of zombies and humans as continuous variables in the last section, we are ignoring the random fluctuations that arise in small populations: even a ferociously virulent zombie infestation might fortuitously be killed early on by happy accident. Similar problems arise in chemical reactions: reactions involving two types of proteins in a cell can be described by chemical reaction kinetics evolving their concentrations (like our SZR equations 3.4), but if the number of such proteins is small, accurate predictions must

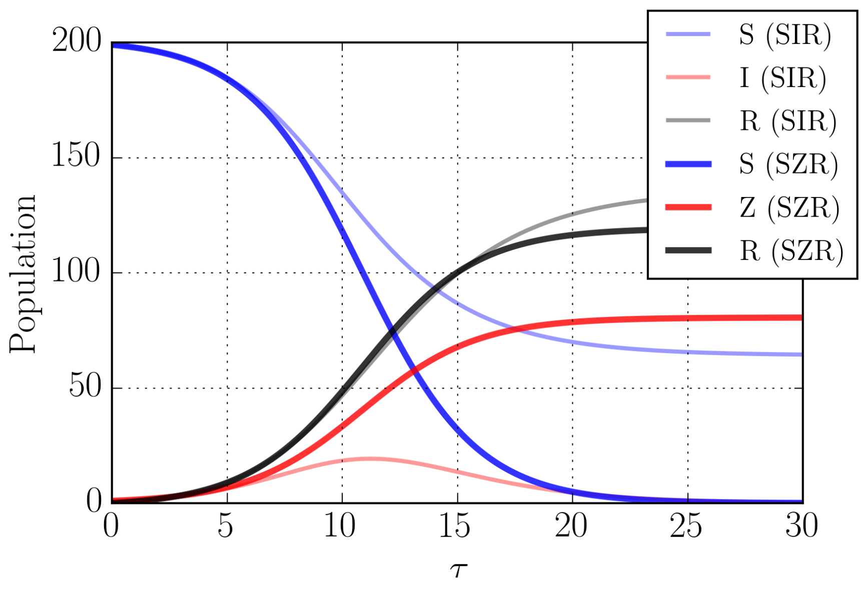


Figure 3.1: Deterministic trajectories for the SIR and SZR models with an initial population of 200 people, 199 uninfected and 1 infected. The (susceptible, infected, removed) population is shown in (blue, red, black) (color online). The SZR results are solid lines while the SIR results are lighter lines. For both models $\tau = t\beta N$ where N was taken to be the total population. For the SZR model α was chosen to be 0.6, while for the SIR model μ was chosen to be 0.6 to show similar dynamics. Notice that in this case, in SZR the human population disappears and only zombies remain in the end, while the SIR model is self-limiting, and only a fraction of the population ever becomes infected.

simulate the individual binary reactions (each zombie battling each human). Interpreting our SZR transitions as reaction rates, gives us a system akin to a chemical reaction with two possible transitions:

$$(S, Z) \xrightarrow{\beta S Z} (Z, Z)$$

$$(S, Z) \xrightarrow{\kappa S Z} (S, R)$$

When a human and zombie are in contact, the probability of a bite in a small period of time is given by the bite rate and the size of the populations of the two

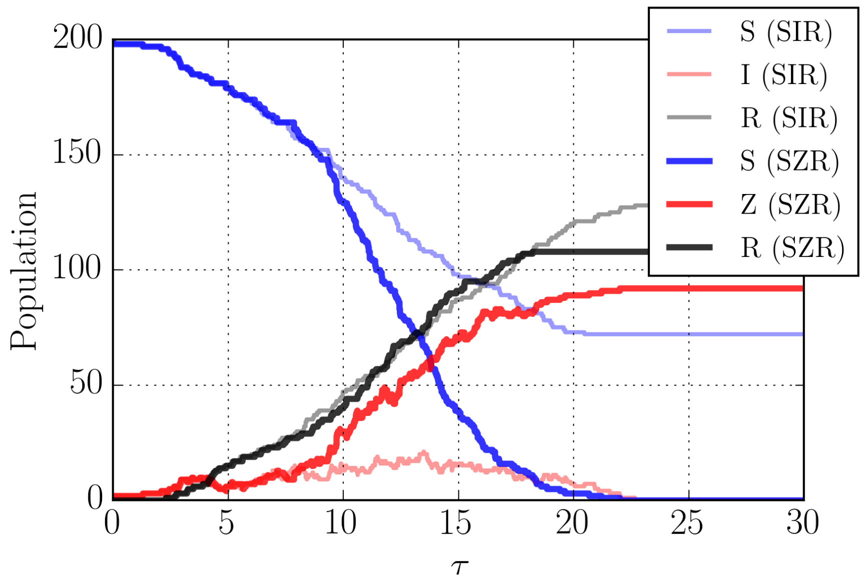


Figure 3.2: Example Gillespie dynamics for the SIR and SZR models with the same parameter settings as Figure 3.1. The (susceptible, infected, removed) population is shown in (blue, red, black) (color online). The SZR results are solid lines while the SIR results are lighter lines. The two simulations were run with the same seed so as to match their dynamics at early times.

species ($\beta SZ\,dt$), and similarly for the probability of a kill. In order to efficiently simulate this dynamics, we use the Gillespie algorithm [29], which efficiently uses the computer to sequentially calculate the result of each one-on-one battle.

The stochasticity gives more character to the simulation. The fully connected continuous dynamics modeled by the differential equation is straightforward: either the humans win and kill all of the zombies, or the zombies win and bite all of the humans. While the continuous approximation may be appropriate at intermediate stages of the infection where the total population is large and there are a non-trivial number of infected individuals, we will eventually be interested in simulating an actual outbreak on an inhomogeneous population lattice, where every new site will start with a single infected individual. But even though we may be interested in modeling the outbreak case ($\alpha < 1$), we would like to allow the possibility that the humans manage to defeat the outbreak before it really takes off. The stochastic Gillespie dynamics allows for this possibility.

In Figure 3.2 we have shown an example of a single stochastic simulation using the same parameter settings as those used in Figure 3.1. The stochastic trajectory overall tracks the analytic result, but at points in the simulation there may be more or fewer zombies than anticipated if the dice fall that way.

Another implication of stochastic dynamics is that it is not always guaranteed that a supercritical ($\alpha < 1$) outbreak will take over the entire susceptible population. For the parameter settings used in Figure 3.1 and 3.2, namely $\alpha = 0.6$ with a population of 200 and one infected individual to start, the zombies win only 40% of the time. Additionally, the number of zombies we end with is not fixed, as shown in Figure 3.3.

In fact, we can solve exactly for the probability P_{ext} that an $\alpha < 1$ simulation will go extinct in the limit of large populations, using an argument drawn from the theory of *branching processes* [113]. At the very beginning of the simulation, there is only one zombie, who will be killed with probability $\kappa/(\beta+\kappa)$. If the first zombie is killed before it bites anyone, we guarantee extinction. Otherwise, the zombie will bite another human, at which point there will be two independent zombie lines that need to be extinguished, which will occur with probability P_{ext}^2. This allows us to solve:

$$P_{\text{ext}} = \frac{\kappa}{\beta+\kappa} 1 + \frac{\beta}{\beta+\kappa} P_{\text{ext}}^2 \tag{3.32}$$

$$P_{\text{ext}} = \frac{\kappa}{\beta} = \alpha\ . \tag{3.33}$$

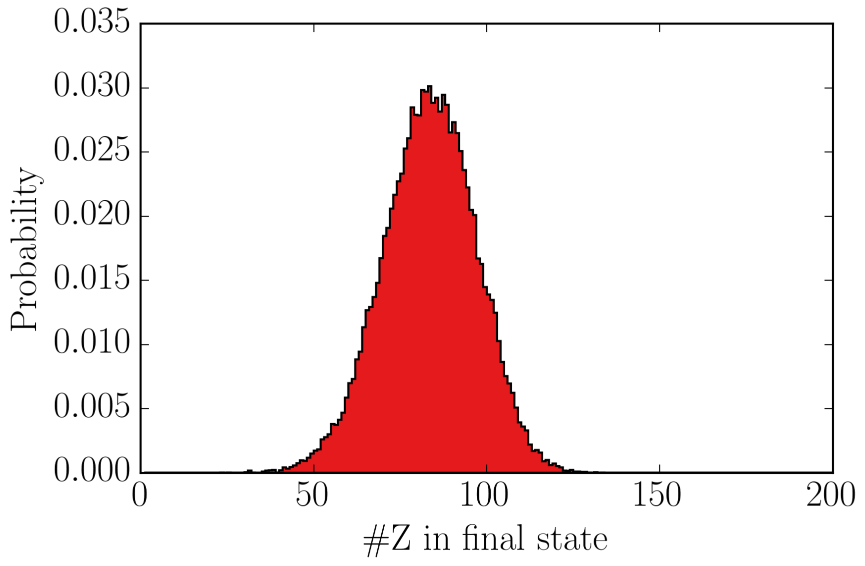


Figure 3.3: Distribution for final zombies over 100,000 stochastic trajectories with the same parameters as Figure 3.2. Not pictured are the 60% of runs that end with no zombies in the final state. Compare these to the analytical result, in which the final population of zombies would be 81 with no possibility of surviving humans.

The probability of extinction is just given by our dimensionless inverse virulence α. In Figure 3.4 we have shown the observed extinction probabilities for 1,000 Gillespie runs of a population of 10^4 individuals at various values of α, and overlaid our expected dependence of α.

This same extinction probability ($P_{\text{ext}} = \mu = R_0^{-1}$) is observed for the SIR model [44]. This is not a coincidence. In precisely the limit that is important for studying the probability of an extinction event, namely at early times with very large populations, the SZR model and SIR are effectively the same, since the population of susceptibles (S) is nearly constant. Writing S as $S_0 - \delta S$, we

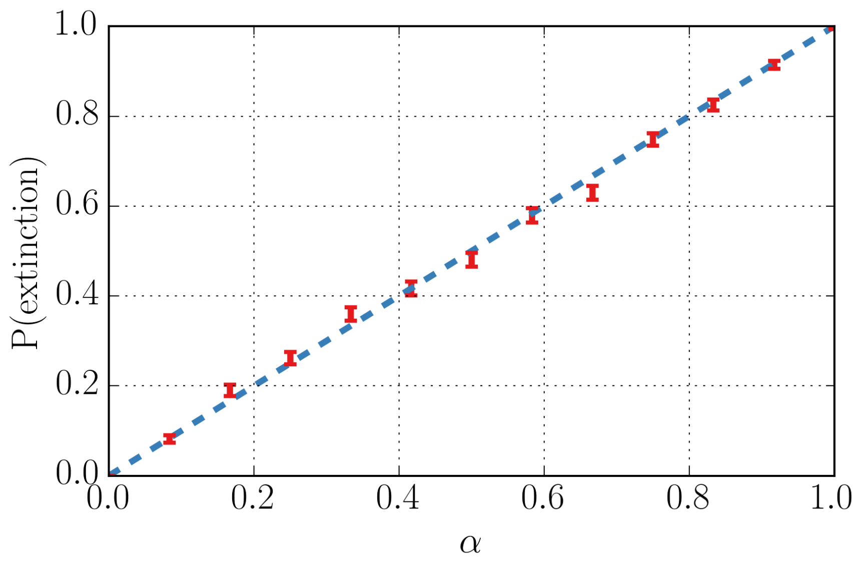


Figure 3.4: The observed fraction of simulations that end in an extinction for the zombie outbreak, for 1,000 runs of 10^4 individuals at various values of α (eqn. 3.33). The observed extinction probabilities agree with the expectation that they should go as α, here shown as the dashed line. This is the same behavior as the SIR model.

have:

$$\frac{dZ}{d\tau} = (1-\alpha)\frac{S_0 Z}{N} - (1-\alpha)\frac{(\delta S)Z}{N} \tag{3.34}$$

$$\frac{dI}{d\tau} = \left(1 - \frac{\mu N}{S_0}\right)\frac{S_0 I}{N} - (\mu N + \delta S)\frac{I}{N} . \tag{3.35}$$

Here as $\delta S \to 0$, the two models are the same with $\alpha = \mu N/S_0$, another indication that the density dependent SIR model's virulence is dependent on population size.

To get a better sense of the effect of the stochasticity, we can look at the mean fractional population in each state for various settings of α and choices for initial population size. The results are shown in Figure 3.5.

Plotted are the fractional populations in the final state left for both the SZR model (top row) and SIR model (bottom row) for different parameter combina-

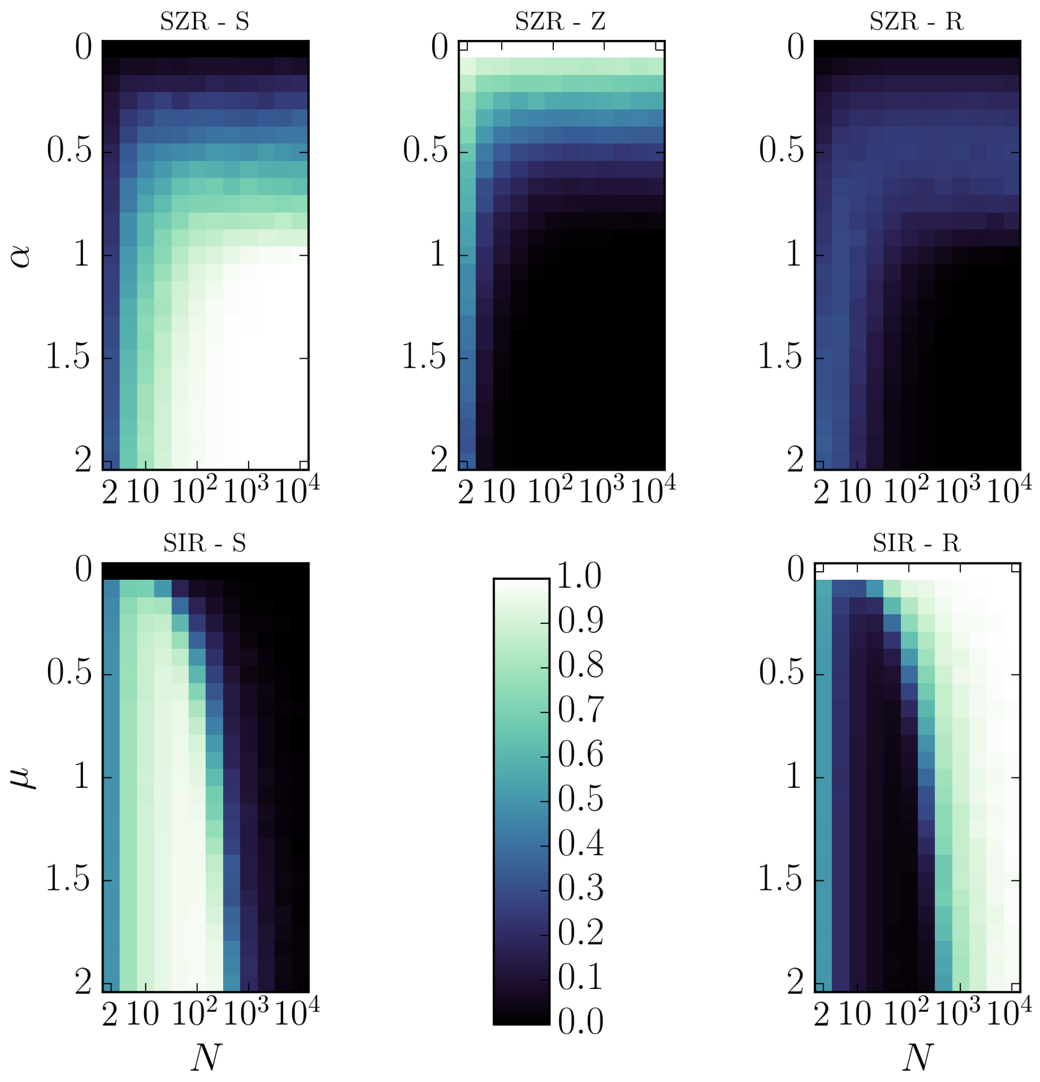


Figure 3.5: Mean final states as a function of model parameters. One thousand different simulations are run for each cell. Each simulation starts with a single zombie or infected individual. The runs are run until they naturally terminate, either because the susceptible population is deleted, the zombie population is gone, or there are no more infected individuals. Each cell is colored according to the mean fraction of the population occurring in each state. The top row is for SZR simulations and the bottom row is for SIR simulations. In both cases N is chosen to be 100. Here the sharp contrast between density-dependent SZR and SIR is made apparent. Notice that density-dependent SIR is very strongly population dependent.

tions of α and the initial population. In all cases, the N parameter was chosen to be 100. For each box, 1,000 independently seeded stochastic trajectories were calculated until completion. Looking at the SZR results in the top row, we can see that the dynamics is fairly independent of population size once the population size gets above around 100 individuals. The population dependence for lower population sizes is an effect of the stochasticity. We can clearly see a transition in the susceptible population near $\alpha = 1$ corresponding to where our continuous dynamics would show a sharp boundary. Here the boundary is blurred, again due to the stochasticity. The final dead zombie population R remains small for all values of α; for extremely virulent zombies $\alpha \ll 1$, very few will be killed by the humans before all of the humans are converted, while in the other extreme few zombies are created so there are few to be killed.

Contrast these results with the density dependent SIR dynamics shown in the second row. There can be no infected individuals left in the end, so only the fraction of S and R in the final state are shown. The two transitions in SIR couple differently to the population of infected and susceptible. While our nondimensionalized SZR model has $Z' = (1 - \alpha)SZ/N$, our nondimensionlized SIR has $I' = (S/N - \mu)I$. This creates a very strong population dependence. The transition observed in the S population is largely independent of μ, except on the very small end. When we move to inhomogeneous population lattices this means that for the density dependent SIR model, the most important parameter governing whether a particular site has a break-out infection is the population of that site on the lattice.

3.4 Critical Behavior of Lattice Model

Until now, we have considered fully connected, well-mixed populations, where any infected individual can infect any susceptible individual with equal probability. But surely, a zombie in New York cannot bite someone in Los Angeles. Investigation of the spatial spread of infectious diseases is an important application of *network science*; social diseases spread among intimate contacts, Ebola spreads by personal contact in a network of care-givers, influenza can be spread by direct contact, through the air or by hand-to-mouth, hand-to-eye or hand-to-nose contact after exposure to a contaminated surface. For most diseases, 'long bonds' dominate the propagation to distant sites [72]; airplane flights take Ebola to new continents. Zombies do not fly airplanes, so our model is closer in spirit to the spread of certain agricultural infestations, where the disease spreads across a lattice of sites along the two-dimensional surface of the Earth (although not in those cases where pathogens are transported long distances by atmospheric currents).

To begin, we will consider a two-dimensional square lattice, where each site contains a single individual. Each individual is allowed to be in one of three states: S, Z, or R. The infection spreads through nearest neighbor bonds only. That is, a zombie can bite or be killed by any susceptible individuals in each of the four neighboring sites.

To make direct contact with our zombie model, the rate at which an susceptible cell is bitten is given by βZ where Z is the number of zombie neighbors (since S is one), and the rate at which a zombie site is killed is κS where S is the number of susceptible neighbors.

Because all state transitions in the SZR model depend only on Z–S contacts, for computational efficiency, we need only maintain a queue of all Z–S bonds, that is connections along which a human and zombie can interact. At each step of the simulation, one of these Z–S bonds is chosen at random, and with probability $\beta/(\beta + \kappa) = 1/(1 + \alpha)$, the human is bitten, marking it as a zombie. We can then query its neighbors, and for all of them that are human, we can add a Z–S link to our queue. With probability $\kappa/(\beta + \kappa) = \alpha/(1 + \alpha)$ the zombie is killed, removing any of its links to neighboring humans from the queue. This process matches the stochastic dynamics of our zombie model operating on the lattice.

Simulating zombie outbreaks on fixed lattices, there is qualitatively different behavior for small α and large α. When α is large, the zombies do not spread very far, always being defeated by their neighboring humans. When α is very small, the zombies seem to grow until they infect the entire lattice. This suggests evidence of a phase transition. Technically, the presence of a phase transition would mean that if we could simulate our model on an infinite lattice, there should be some critical α (α_c), above which any outbreak will necessarily terminate. Below the critical value, there is the possibility (assuming the infection does not die out) of having the infection grow without bound, infecting a finite fraction of individuals in the limit that the lattice size becomes infinite. The SIR model has been demonstrated to undergo such a phase transition, and we expect the zombie model does as well.

The study of *critical phenomena* includes a series of techniques and analyses that enable us to study the properties of phase transitions even on finite lattices. A major theme of critical phase transitions is the importance of *critical points* – where a system is tuned (here by varying α) to a value separating qualitatively different behaviors (here separating low-infectivity transient zombie infestations from a potentially world-spanning epidemic). At critical points, the system can show *scale free behavior*; there is no natural length scale to the dynamics, and various physical parameters will usually be governed by *power laws* (see below).

With α chosen to be precisely at the critical value, we indeed see a giant component with fractal structure (Fig. 3.6). Note that there are holes (surviving pockets of humans) of all sizes in the figure. This reflects the proximity to the threshold: the battle between zombies and humans is so evenly matched, that one gets an *emergent scale invariance* in the survival patterns. This is in keeping with studies of the SIR model, which shows a similar critical behavior and phase transition [32].

Systems near critical points with this kind of scale invariance fall into *universality classes*. Different systems (say, a real disease outbreak and a simple computational model) can in many ways act precisely the same on large scales near their transitions (allowing us to predict behavior without knowing the details of zombie-human (anti)social interactions). The SIR model on a two-dimensional lattice with a single person per site falls into the percolation universality class [10], though details of its cluster growth can differ [104]. Given that the SZR model has two second order couplings, it is of interest whether it falls into the same percolation universality class.

To extract the scaling behavior of our zombie infestation, we study the distribution $P(s, \alpha)$, the probability that a single zombie will generate an outbreak of size s at inverse virulence α. (An outbreak will be a fractal cluster in two dimensions, with ragged boundaries if it dies out before reaching the entire world.) At

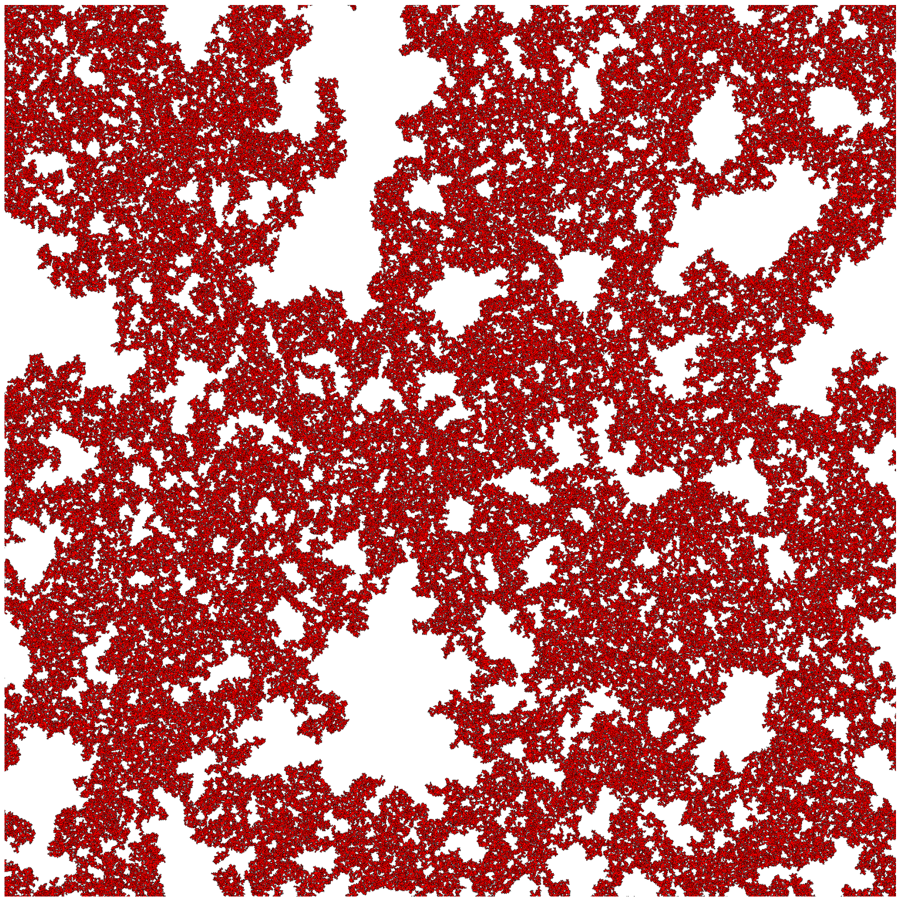


Figure 3.6: Example cluster resulting from the single population per site square lattice zombie model with periodic boundary conditions near the critical point $\alpha_c = 0.437344654(21)$ on a lattice of size 2048×2048.

$\alpha = \alpha_c$ where the zombies and humans are equally matched, we have an emergent scale invariance. A large outbreak will appear to almost stop several times – it can be viewed as a sequence of medium-sized outbreaks triggering one another just before they die out. Medium-sized outbreaks are composed of small outbreaks, which are in turn composed of tiny outbreaks. At threshold, each of these scales (large, medium, small) is related to the lower scale (medium, small, tiny) in the same fashion. Let us oversimplify to say that at criticality an outbreak of size Bs is formed by what would have been three smaller outbreaks of size s which happened to trigger one another, and these in turn are formed by what would have been three outbreaks of size s/B. If the probabilities and form of this mutual triggering is the same at each scale, then it would not surprise us that many properties of the outbreaks would be the same, after rescaling the sizes by a factor of B. In particular, we expect at the critical point to find the probabilities of outbreaks of size s to be related to the probabilities at size s/B by some factor f:

$$P(s, \alpha_c) = f P(s/B, \alpha_c). \tag{3.36}$$

This formula quantifies an *emergent scale invariance* at α_c: the properties of epidemics of size s (here the probability) are rescaled versions of the properties at a smaller scale s/B. [91] – the system is *self-similar* to itself at different scales. Eqn 3.36 is solved by $P(s, \alpha_c) \propto s^{-\tau}$, with $\tau = \log(1/f)/\log(B)$. The distribution of epidemic infection rates is a power law.

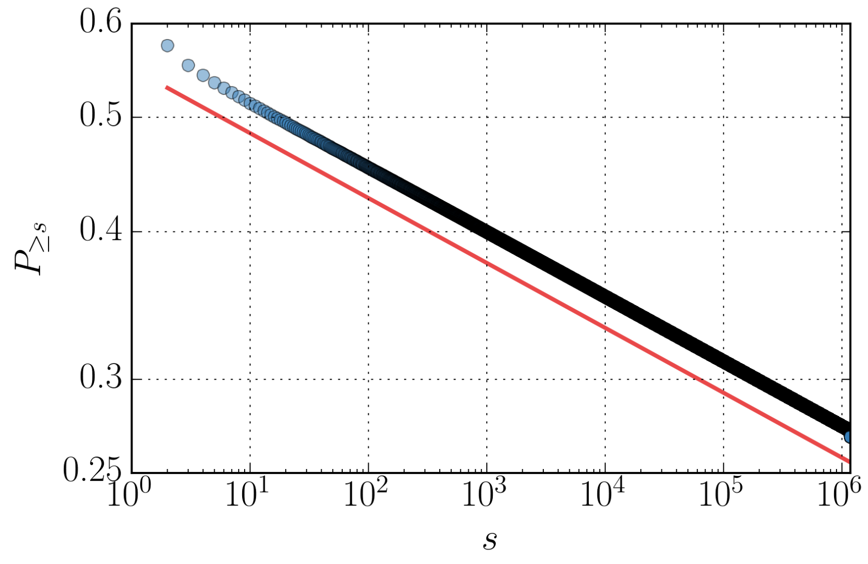
Figure 3.7 shows a thorough test of this dependence for our zombie model, following a procedure akin to that of reference [104]. We simulated a zombie outbreak on a two-dimensional lattice with periodic boundary conditions starting with a single zombie. With the outbreak sizes following a power law distribution, the probability that a site belongs to a cluster of size n_s is $P_s = sn_s$, so that at the critical point $P_s \sim s^{1-\tau}$. Integrating from s to ∞, the probability that a point belongs to a cluster of at least s in size ($P_{\geq s}$) should at the critical point itself follow a powerlaw: $P_{\geq s} \sim s^{2-\tau}$. To find our critical point α_c, we ran many simulations until our integrated cluster size distribution followed a power law, using the interpolation methods of reference [104] to get a precise estimate of the critical point.

For zombies on a two dimensional lattice, this critical point occurs at $\alpha_c = 0.437344654(21)$, the resulting integrated cluster size distribution is shown at the top of Fig. 3.7. Percolation theory predicts $\tau = 187/91$ in two dimensions, and we test that prediction in the bottom part of Fig. 3.7. Here, if we were precisely at the critical point and the SZR model is in the percolation universality class, with infinite statistics we would have asymptotically a perfectly straight line. Notice the small vertical scale: our fractional fluctuations are less than 0.1%, while our experimental results vary over several order of magnitude. The clear agreement convincingly shows that the zombie model on the two dimensional lattice is in the percolation university class.

As an additional check, we computed the fractal dimension of our clusters near the critical point using box counting, a distribution for which is shown in Figure 3.8. We find a fractal dimension $D = 1.8946(14)$, compared to the exact percolation value of $D = 91/48 = 1.895833$.

Why did we need such an exhaustive test (many decades of scaling, many digits in our estimate of α_c)? On the one hand, a much smaller simulation could have told us that there was emergent scale invariance and fractal behavior near the transition; one or two decades of scaling should be convincing. But it turns out that there are multiple different universality classes for this kind of invasion process, and their exponents τ and D are rather similar. And a small error in α_c can produce large shifts in the resulting fits for τ and D – demanding efficient programming and fast computers to achieve a definitive answer.

We conclude that the single person per site zombie infestation, near the critical virulence, will on long length scales develop spatial infestation patterns that are well described by two-dimensional percolation theory.



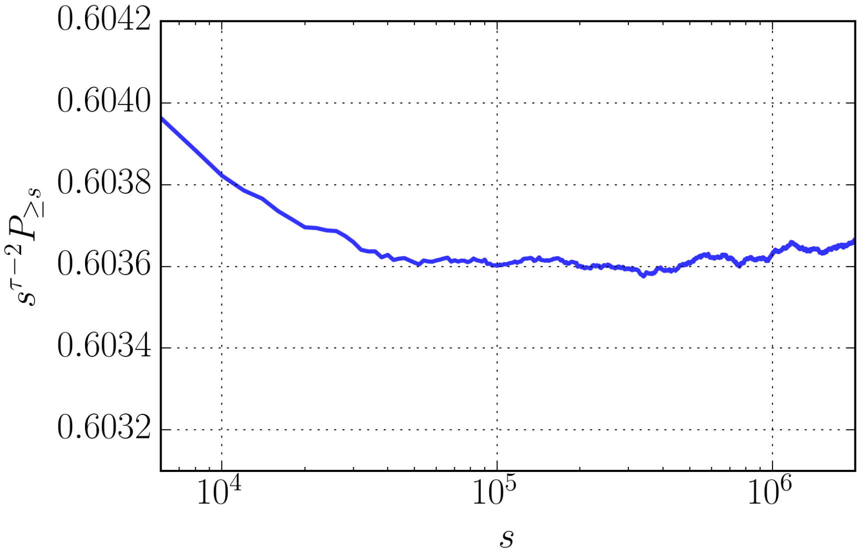


Figure 3.7: The cumulative distribution of epidemic sizes for the two dimensional zombie model near the critical virulence. The critical point found was $\alpha_c = 0.437344654(21)$. The top plot shows the probability of a site being in a cluster of at least s in size ($P_{\geq s}$). The fact that it forms a straight line on a log-log plot indicates that $P_{\geq s}$ is a power law, and the slope is $2 - \tau$. For comparison, the red (color online) line shows the powerlaw corresponding to the percolation critical exponent: $\tau = 187/91$. The bottom plot shows the same data times $s^{\tau-2}$ using the exponent from percolation theory. The plot is very nearly flat suggesting the percolation exponent accurately describes the zombie model.

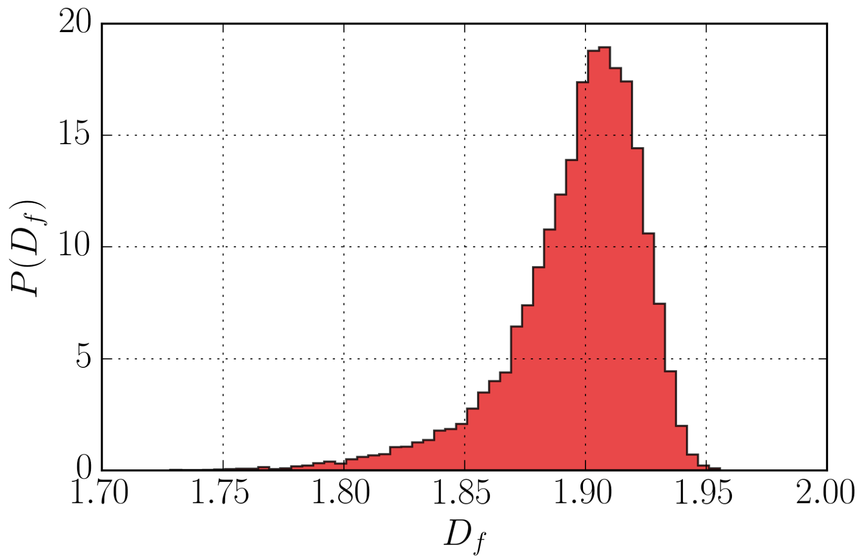


Figure 3.8: A histogram of the observed fractal dimension of the zombie epidemic clusters as measured by box counting. These give a measured value of $D = 1.8946(14)$, consist with the exact percolation fractal dimension of $D = 91/48 = 1.895833$.

3.5 US Scale Simulation of Zombie Outbreak

Having explored the general behavior of the zombie model analytically, stochastically and on homogeneous single person lattices, we are prepared to simulate a full scale zombie outbreak.

3.5.1 Inhomogeneous Population Lattice

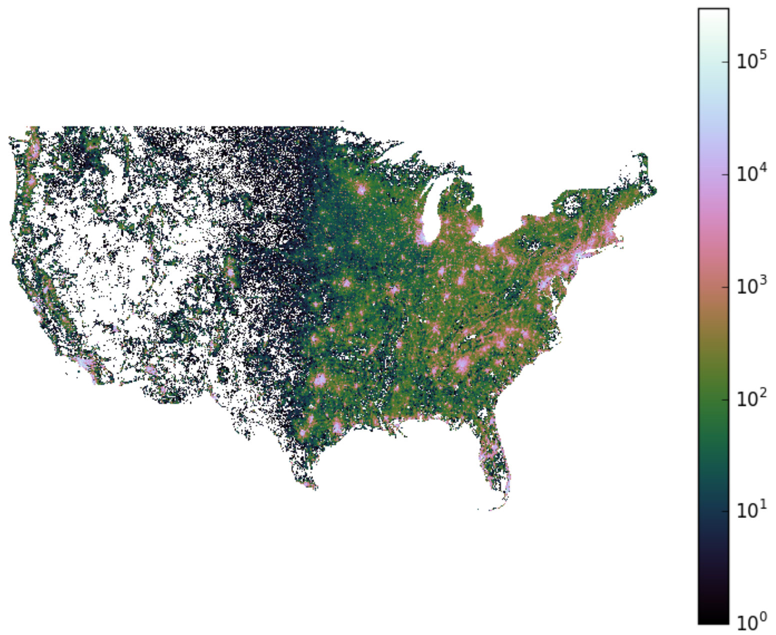
We will attempt to simulate a zombie outbreak occurring in the United States. This will be similar to our lattice simulation, but with an inhomogeneous population lattice. We based our lattice on code available for creating a "dot map" based on the 2010 US Census data [3]. The 2010 Census released census block

[3]`https://github.com/meetar/dotmap`

level data, detailing the location and population of 11,155,486 different blocks in the United States. To cast these blocks down to a square grid, we assigned each of the 306,675,005 reported individuals a random location inside their corresponding census block, then gridded the population into a 1500×900 grid based on latitude and longitude coordinates. The resulting population lattice can be seen in the top half of Figure 3.9. You will see the presence of many empty grids, especially throughout the western United States. This disconnects the east and west coasts in a clearly artificial pattern – our zombies in practice will gradually wander through the empty grid points. To add in lattice connectivity, we did six iterations of binary closing (an image processing technique) on the population lattice and added it to the original. The effect was to add a single person to many vacant sites, taking our total population up to 307,407,336. The resulting population map is shown in the bottom half of Figure 3.9. This grid size corresponds to roughly 3 km square boxes. The most populated grid site is downtown New York City, with 299,616 individuals. The mean population of the occupied grid sites is 420, the median population of an occupied site is 13.

3.5.2 Augmented Model

In order to more 'realistically' simulate a zombie outbreak, we made two additions to our simplified SZR model. The first was to add a latent state E (Exposed). The second was to introduce motion for the zombies. Considered as a



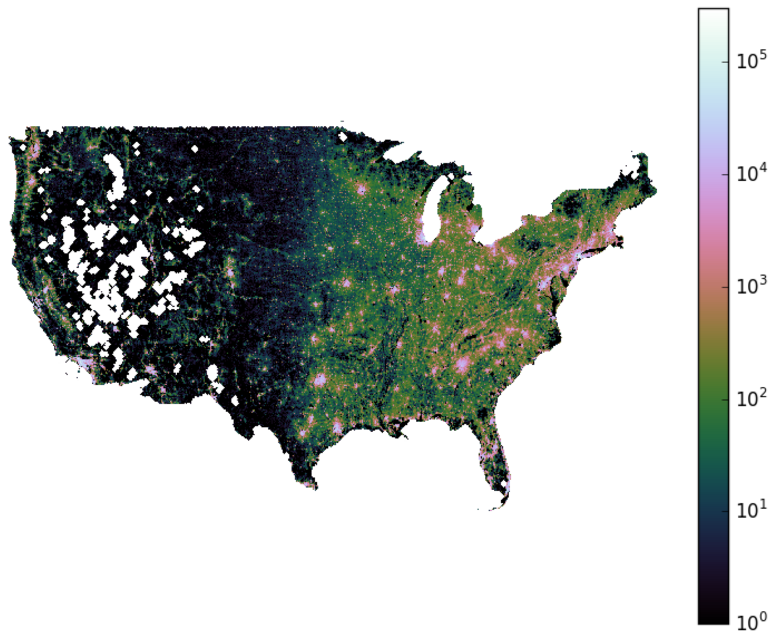


Figure 3.9: A 1500×900 grid of the 2010 US Census Data. The above figure gives the raw results. Notice the multitude of squares with no people in them in the Western United States. The bottom figure shows the resulting map after 6 steps of binary closing added to the original population.

system of differential equations, we now have:

$$\dot{S}_i = -\beta S_i Z_i \tag{3.37}$$

$$\dot{E}_i = -\nu E_i \tag{3.38}$$

$$\dot{Z}_i = \nu E_i - \kappa S_i Z_i \tag{3.39}$$

$$\dot{R}_i = \kappa S_i Z_i \tag{3.40}$$

$$\dot{Z}_i = \mu \sum_{\langle j \rangle} Z_j - \mu Z_i \tag{3.41}$$

or as a set of reactions:

$$(S_i, E_i) \xrightarrow{\beta S_i Z_i} (S_i - 1, E_i + 1) \tag{3.42}$$

$$(Z_i, E_i) \xrightarrow{\nu E_i} (Z_i + 1, E_i - 1) \tag{3.43}$$

$$(Z_i, R_i) \xrightarrow{\kappa S_i Z_i} (Z_i - 1, R_i + 1) \tag{3.44}$$

$$\langle i\, j \rangle : (Z_i, Z_j) \xrightarrow{\mu Z_i} (Z_i - 1, Z_j + 1)\ . \tag{3.45}$$

Here i denotes a particular site on our lattice. $\langle j \rangle$ denotes a sum over nearest neighbor sites, $\langle i\, j \rangle$ denotes that i and j are nearest neighbors. In this model, zombies and humans only interact if they are at the same site, but the zombies diffuse on the lattice, being allowed to move to a neighboring site with probability proportional to their population and some diffusion constant (μ). We assume that the humans do not move, not only for computational efficiency, but because, as we will see, the zombie outbreaks tend to happen rather quickly, and we expect large transportation networks to shut down in the first days, pinning most people to their homes. The addition of a latent state coincides with the common depiction that once a human has been bitten, it typically takes some amount of time before they die and reanimate as a zombie. If a human is bitten, they transition to the E state, where at some constant rate (ν) they convert into the zombie state.

To choose our parameters we tried to reflect common depictions of zombies in movies. The work of Witkowski and Blais [114] performed a Bayesian fit of a very similar SZR model to two films, *Night of the Living Dead*, and *Shawn of the Dead*. In both cases, the observed α was very close to 0.8. This means that the zombies in the films are 1.25 times more effective at biting humans than the humans are at killing the zombies. We will adopt this value for our simulation. For our latent state, we adopt a value close to that reported for *Shawn of the Dead*, namely a half-life of 30 minutes. To set our movement parameter, we estimate that zombies move at around 1 ft/sec. (Note that metric units are uniformly used in science. We use the parochial US units of feet in homage to the popular culture from which we draw our data.) To estimate the rate at which the zombies will transition from one cell to the next, we assume that the zombies behave like a random gas inside the cell, so that the probability that a zombie will cross a cell boundary is roughly $\frac{1}{4}\frac{Z}{L^2}Lv\Delta t$, that is, one-fourth of the zombies within $v\Delta t$ of the edge will move across that edge in a small amount of time. This suggests a value of μ of 0.0914 /hr. This corresponds to an average time between transitions of around 11 hours, which for a zombie stumbling around a 3 km block agrees with our intuitions. Finally, to set a rate for our bite parameter, we similarly assume that the zombies are undergoing random motion inside the cell at 1 ft/sec, and they interact with a human anytime they come within 100 feet. We can then estimate the rate at which humans and zombies will interact as $SZ\frac{Rv\Delta t}{L^2}$, which corresponds to a choice of β of around 3.6×10^{-3} /hr. Another way to make sense of these parameter choices is to ask how many susceptible individuals must be in a cell before a single zombie has a higher rate for biting a human than transitioning to a neighboring cell. For our choice of parameters,

β	3.6×10^{-3} /hr/person
α	0.8
κ	$\alpha\beta$
η	2 /hr
μ	0.0914 /hr

Table 3.1: The parameters chosen for our US-scale simulations of a zombie outbreak. These parameters were chosen to correspond with standard depictions of zombies and simple physical estimations explained in the main text.

this gives

$$N\beta = 4\mu \implies N \sim 102\ . \tag{3.46}$$

This corresponds to a low population density of $\sim$ 11 people/km^2, again agreeing with our intuition. All of our parameter choices are summarized in Table 3.1.

3.5.3 Simulation Details

To effectively simulate an outbreak at this scale, we employed the Next Reaction Method of [28]. We maintained a priority queue of all possible reactions, assigning each the time at which the reaction would take place, an exponentially distributed random number with scale set by the rate for the reaction. At each time step of the simulation, we popped the next reaction off of the queue, and updated the state of the relevant squares on our grid. Whenever population counts changed, we of course needed to update the times for the reactions that depend on those population counts. This method remained efficient for simulating the entire US. However, at late times a large amount of simulation time was spent simulating the diffusion of the zombies back and forth between highly populated states. We could have achieved additional computational efficiency by adopting the time dependent propensity function approach of Fu et al. [27].

3.5.4 Results

With the simulation in place, we are now in a position to simulate a full scale zombie outbreak. We first consider an outbreak that began with one in every million individuals starting in the Exposed (E) state in the United States. For a single instance the overall populations are shown in Figure 3.10. This looks similar to the analytical outbreaks we saw in Figure 3.1, but with a steeper rate of initial infection and some slight perturbations to the curves. The total population curves however hide most of the interesting features. In Figure 3.11 we attempt to give a sense of how this outbreak evolves, showing the state of the United States at various times after the outbreak begins.

As you can see, for the parameters we chose, most of the United States population has been turned into zombies by the first week, while the geographic map does not necessarily seem all that compelling. In the early stages of the outbreak, while the population is roughly homogeneous, the zombie plague spreads out in roughly uniform circles, where the speed of the infection is tied to the local population density. Infestations on the coasts, with their higher population density, have spread farther than those near the center of the country. After several weeks, the map exhibits stronger anisotropy, as we spread over larger geographical areas and the zombie front is influenced by large inhomogeneities in population density. After four weeks, much of the United States has

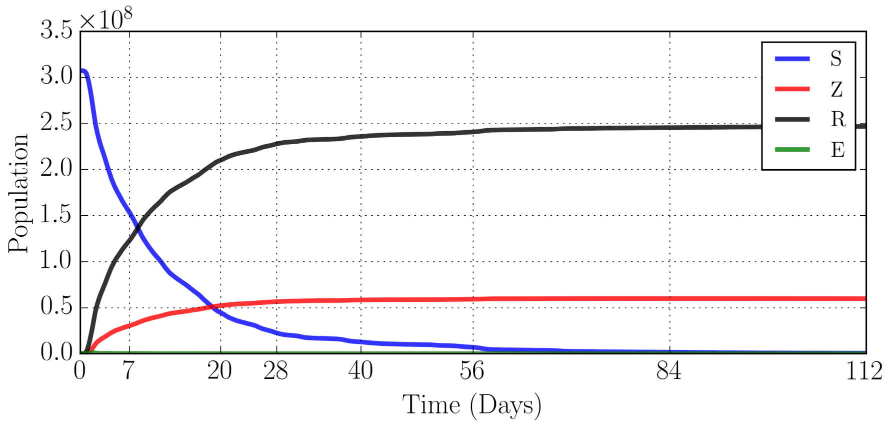
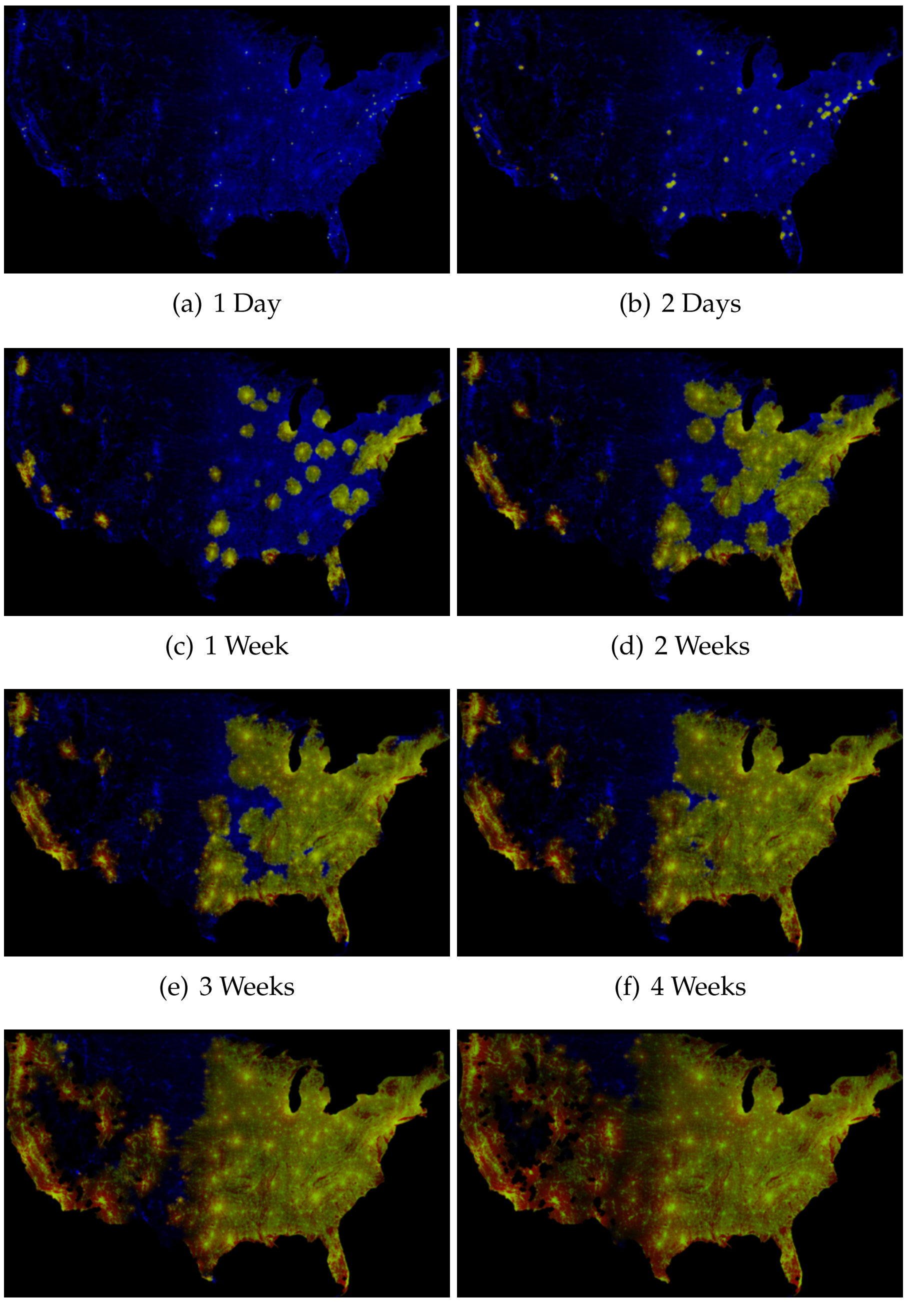


Figure 3.10: The S (blue), Z (red), R (black), and E (green) populations as a function of time for a full scale zombie outbreak in the continental United States starting with one in every million people infected (color online). The exposed population (E) has been magnified by a factor of 100.

fallen, but it takes a very long time for the zombies to diffuse and capture the remaining portions of the United States. Even four months in, remote areas of Montana and Nevada remain zombie free.

To investigate the geographical characteristics of the outbreak, we must move beyond a single instance of an outbreak and study how different regions are affected in an ensemble of outbreaks. If it takes a month to develop and distribute an effective vaccine (or an effective strategy for zombie decapitation), what regions should one locate the zombie-fighting headquarters? We ran 7,000 different 28-day zombie outbreaks in the continental United States starting with a single individual. A single instance of one of these outbreaks originating in New York City is shown in Figure 3.12.

By averaging over all of these runs, we can start to build a zombie danger map, as shown in Figure 3.13. In the top plot, we show the probability that the given cell is overrun by zombies after seven days. Here you can clearly see that



(a) 1 Day

(b) 2 Days

(c) 1 Week

(d) 2 Weeks

(e) 3 Weeks

(f) 4 Weeks

(g) 2 Months

(h) 4 Months

Figure 3.11: Simulation of a zombie outbreak in the continental United States. Initially one in every million individuals was infected at random. Results are shown above at (a) one day, (b) two days, (c) one week, (d) two weeks, (e) three weeks, (f) four weeks, and (g) two months after the outbreak begins. Shown here are the population of susceptible individuals (S) in blue, scaled logarithmically, zombies in red and removed in green (color online). All three channels are superimposed.

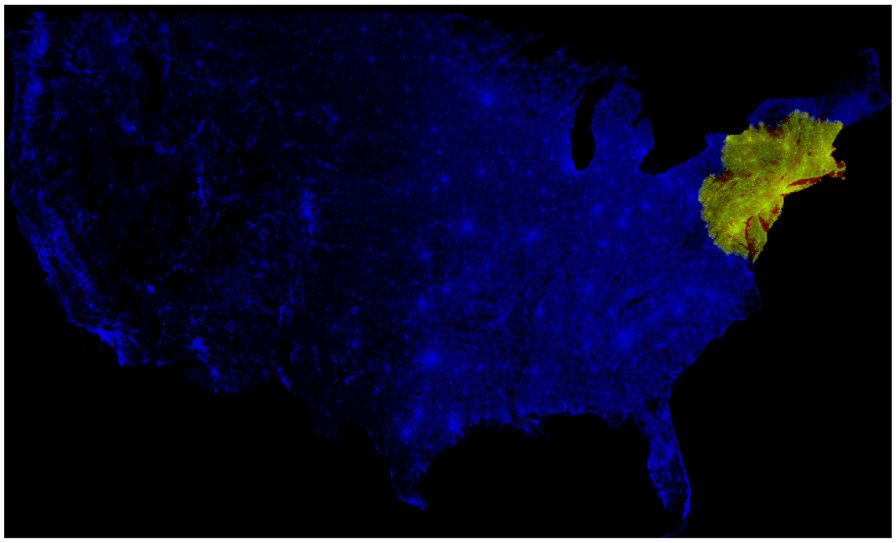


Figure 3.12: Status of the United States 28 days after an outbreak that started in New York City. Here blue represents humans, red represents zombies and green represents dead zombies (color online). The three color channels have been laid on top of one another.

there are certain regions – those surrounding populous metropolitan areas – that are at a greater risk. This is partly because those regions have lots of individuals who could potential serve as patient zero, and partly due to the rapid spread of zombies in those areas. In the bottom plot, we plot the probability that the cell is overrun, but at the 28 day mark.

After 28 days, it is not the largest metropolitan areas that suffer the greatest risk, but the regions located between large metropolitan areas. For instance, in California it is the region near Bakersfield in the San Joaquin Valley that is at the greatest risk as this area will be overrun by zombies whether they originate in the San Francisco area or the Los Angeles / San Diego area. The area with the greatest one month zombie risk is north eastern Pennsylvania, itself being susceptible to outbreaks originating in any of the large metropolitan areas on the east coast.

3.6 Conclusion

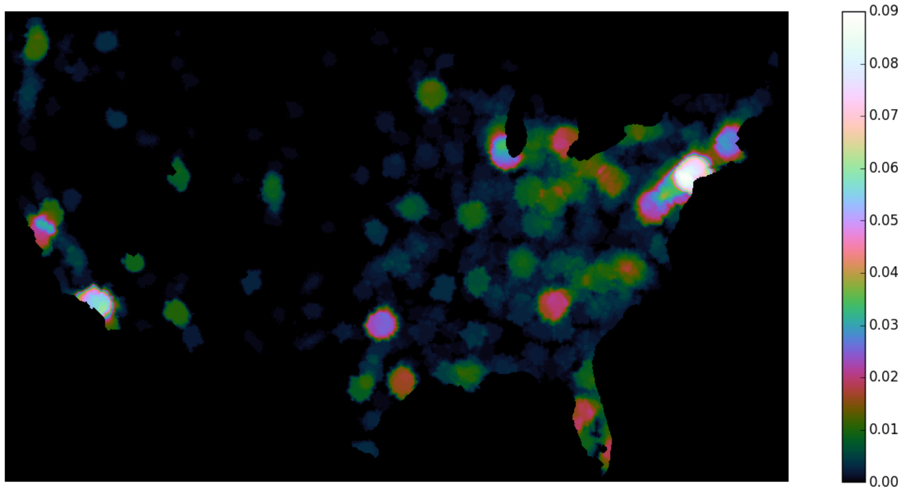
Zombies offer a fun framework for introducing many modern concepts from epidemiology and critical phenomena. We have described and analyzed various zombie models, from one describing deterministic dynamics in a well-mixed system to a full scale US epidemic. We have given a closed form analytical solution to the well-mixed dynamic differential equation model. We compared the stochastic dynamics to a comparable density-dependent SIR model. We investigated the critical behavior of the single person per site two-dimensional square lattice zombie model and demonstrated it is in the percolation universality class. We ran full scale simulations of a zombie epidemic, incorporating each human in the continental United States, and discussed the geographical implications for survival.

While this work is predicated on a fictional infestation, one might ask whether there are any phenomena in the real world that behave in a manner similar to our modeled zombie outbreaks. As noted, the SZR model requires that susceptible hosts directly participate in the removal of zombie hosts from the infectious population, leading to runaway outbreaks as susceptible hosts are depleted. One might imagine a similar phenomenon for infectious diseases that require medical intervention to be suppressed; as medical personnel themselves become infected (as has sadly happened to a considerable degree during the recent Ebola outbreak in West Africa), they become less able to stem the spread of infection. (Medical personnel, however, represent only a small fraction of all susceptible hosts, so a refinement to an SZR-type model would be required to account for this.) One might also imagine SZR-like dynamics in the spread of ideas and opinions: a person spreading a controversial opinion in a population, for example, might be able to sway some converts, but is also likely to meet resistance and counter-arguments, which act to reduce infectivity and perhaps ultimately stop the spread.

We hope our systematic treatment of an imaginary disease will provide a useful and inspiring teaser for the exciting fields of statistical mechanics, network science, and epidemiology.

3.7 Acknowledgments

We acknowledge NSF IIS–1247696 and Cornell University for support of this research, and thank Paul Ginsparg for useful references and conversations.



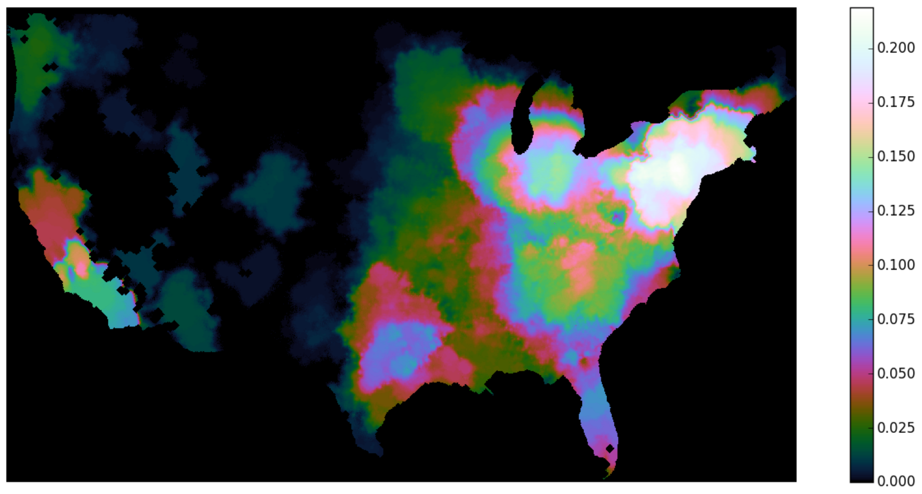


Figure 3.13: Average infection rate from US scale runs. In both cases, the plot shows the probability of being infected in that square after an epidemic that originates from a single infected individual chosen at random from the total population. The top figure is the probability of being infected after 7 days, while the bottom plot is after 28 days. In total, this represents 7,000 simulated runs starting from a single individual. The top plot represents the 1,467 outbreaks that lasted at least 7 days, the bottom plot represents 1,458 outbreaks that lasted at least 28 days.

CHAPTER 4

GRAPHENE

Graphene is a unique material. As a stable two dimensional atomically thin sheet with no real experimental evidence for the formation of defects, [25] graphene is the perfect testbed for nonlinear elastic theory. Being atomically thin, we must move beyond simple elastic theories that ignore rotation gradients, and being as strong as it is we can experimentally probe regions with high strains. In this note we summarize some of our recent work towards that end.

First we summarize the usual approach to nonlinear elastic theory and introduce some of its shortcomings. Then we introduce our approach to nonlinear elastic theory, which will enable us to keep a clear picture of the symmetries inherent in the problem. Next, by studying the symmetry of the problem in depth with group theory, we can determine exactly which terms are allowed to appear in a general free energy expansion. Our expansion will include not only higher order terms in the strain, but also gradient terms and terms involving optic-mode type rearrangements of our unit cells. We then measure the various terms for a model interatomic potential. Finally we will end with a discussion of future work: systematically integrating out the intra-lattice deformations, dispersion and strain gradients, application to other crystal symmetries, and a systematic tabulation of these constants for a variety of density functionals and interatomic potentials.

4.1 Theory

4.1.1 Traditional Approach

First, let's review the usual approach one will find in classic elastic texts such as Landau. [48, 112]

Usually, one imagines a material having undeformed coordinates $\boldsymbol{x}$ and deformed coordinates $\boldsymbol{X}$. In this way, its motion is captured in its displacement field $\boldsymbol{\xi} = \boldsymbol{X} - \boldsymbol{x}$. All of the relevant physics of the material can be described by its local changes in length inside the material. Originally distances are given by

$$ds^2 = dx_i dx_i \, , \tag{4.1}$$

after deformation they are

$$\begin{aligned} dS^2 &= dX_i dX_i \\ &= (dx_i + d\xi_i)(dx_i + d\xi_i) \\ &= dx_i dx_i + 2dx_i d\xi_i + d\xi_i d\xi_i \\ &= ds^2 + 2\frac{\partial \xi_i}{\partial x_k} dx_i dx_k + \frac{\partial \xi_i}{\partial x_k}\frac{\partial \xi_i}{\partial x_j} dx_j dx_k \\ &\equiv ds^2 + 2\epsilon_{ij} dx_i dx_j \, . \end{aligned} \tag{4.2}$$

Where we have defined the primary quantity of interest, the *strain tensor*

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial \xi_i}{\partial x_j} + \frac{\partial \xi_j}{\partial x_i}\right) + \frac{1}{2}\left(\frac{\partial \xi_k}{\partial x_i}\frac{\partial \xi_k}{\partial x_j}\right) \, . \tag{4.3}$$

This strain tensor gives us the local changes in length for the material, the quantity of interest assuming that the material is made up of material that interacts locally. The second, nonlinear term in the definition of the strain tensor is the *geometric nonlinearity*. This term is required to ensure that our measured length changes agree in the case of rotations, but is often ignored given that it introduces an essential nonlinearity in the theory.

From here, one assumes that a theory of the elastic response of a system should depend on the strain tensor ϵ_{ij} and its powers, forming an energy functional of the form:

$$E = \int dV \, C_{ijkl}\epsilon_{ij}\epsilon_{kl} + D_{ijklmn}\epsilon_{ij}\epsilon_{kl}\epsilon_{mn} + \cdots \,. \tag{4.4}$$

This expression can be simplified by making appeals to the symmetry of the material in question. In particular, for the case of an isotropic material, the first term in the energy function (quadratic in the strain tensor) can naturally only have two independent components. Given that the strain tensor itself is manifestly symmetric, the elastic moduli tensor C must be symmetric under interchange of $(ij) \leftrightarrow (kl)$. Assuming the material is symmetric we can only contract with delta functions, leaving only two terms that survive:

$$C_{ijkl}\epsilon_{ij}\epsilon_{kl} \to \lambda e_{ii}^2 + \mu\epsilon_{ij}^2 \,. \tag{4.5}$$

Similar symmetry arguments can be applied all the way down the chain, keeping as many terms as desired in your energy functional.

4.1.2 Extent and Shortcomings

Usually, most people are content to stop with the quadratic term, ignoring the geometric nonlinearity to give them *linear elastic theory*, which is quite easy to solve problems in. This is naturally justified in the case of small strains. For most materials we interact with, plastic flow tends to set in at strains of roughly 10^{-4}. The linear elastic theory dominates in this regime, since the next nonlinear term is suppressed by an additional 10^{-4}. Why then would we be interested in computing nonlinear extensions of elastic theory?

Besides the fact that a nonlinear theory will inherently be more accurate even at low strains, if we want to describe the elastic environment near defects, a nonlinear theory is needed. On the scales of crystal defects, plastic deformation is not possible, and yet the crystal has regions of high strain. If we were interested in accurately describing the strain field surrounding a vacancy or crack tip, the details close to the defect core will have non trivial modifications due to nonlinear terms.

Beyond including terms higher order in the strain itself, another tactic one could employ to improve upon linear elastic theory is to include non-local contributions to the elastic energy functional [82].

Once you begin to consider gradients, a more serious flaw to the traditional approach shows its head: *rotation gradients*. This is particularly a problem for thin sheets. Consider a sheet of paper. Now roll the paper up into a tube. Locally on the sheet, no stretching has been done, so the strain tensor will identically vanish, however we know that if you let go of the sheet it will unfurl. Surely there is a contribution to the energy of the sheet that depends on how much it is bent independent of in-plane stretching. Such terms go beyond the purview of this paper, but will be discussed in Section 4.1.5.

To better appreciate the problem, and to see how to solve it, we'll have to rethink our description of elastic theory from the start.

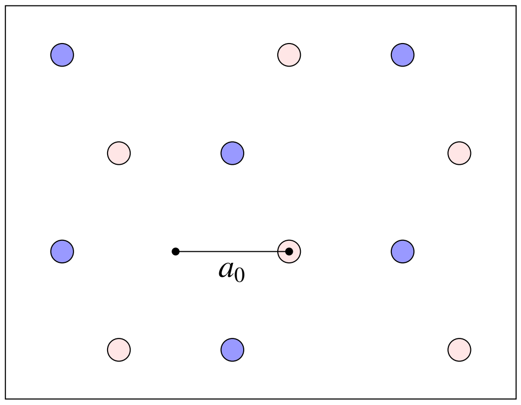


Figure 4.1: The hexagonal lattice of graphene with its two different atoms marked. Also marked is the lattice constant a_0.

4.1.3 Embedding

For graphene, we shall use the more abstract formulation as the study of an embedding of a 2D manifold in 3D space.

Instead of starting with the displacement field, forming the strain tensor and examining powers of it, we instead shall consider elastic deformations not as a transformation of the material in 3-space but as an embedding of an otherwise perfect material in the lab frame. We will consider two distinct spaces: the platonic space, a 2D world in which a perfectly undeformed flat graphene sheet lives, and the lab frame, a 3D world in which we study the twisted and deformed graphene sheet. Consider a perfect, infinite two dimensional graphene sheet, as pictured in Figure 4.1. Now consider a mapping which takes our perfect two dimensional sheet and embeds it in the lab frame with some deformation, giving the lab coordinates in terms of the sheet coordinates. This mapping $Y(x, y)$ would give complete information about the current state of the sheet.

Given that graphene does not form a Bravais lattice, we need two different embedding functions, one for each of the atoms in a unit cell:

$$X^{I\alpha} = Y^{I\alpha}(x^i) \tag{4.6}$$

$$I \in \{X, Y, Z\} \quad i \in \{x, y\} \quad \alpha \in \{A, B\} \tag{4.7}$$

one for the A type atoms (Y^{IA}) and one for the B type atoms (Y^{IB}). Where here and throughout upper case Latin indices will mark indices in our lab frame, lower case Latin indices will mark indices in our perfect undeformed two dimensional space and Greek indices will denote different atoms in the unit cell (here called the A and B atoms).

Naturally these two embedding functions can be mapped to the average and difference functions, here denoted Y and Δ.

$$Y^I \equiv \frac{1}{2}\left(Y^{IA} + Y^{IB}\right) \quad \Delta^I \equiv Y^{IA} - Y^{IB} \tag{4.8}$$

We can think of the average embedding Y as the deformation of the sheet the graphene lives on, and the difference embedding function Δ as describing motions of the atoms inside a unit cell.

The Jacobian of the average embedding gives us information about how differentials on our sheet map under the deformation.

$$dX^I = \frac{\partial Y^I}{\partial x^i} dx^i \equiv \partial_i Y^I dx^i \equiv Y^I{}_{/i} dx^i \equiv F^I{}_i dx^i \ . \tag{4.9}$$

This Jacobian F^I_j is particularly useful. It is known as the *deformation gradient*. In the equation above, I've included several different notations for the partial derivatives.

Note that the deformation gradient is not a true tensor. It has two indices, but one of these indices (I) lives in our 3D lab frame and the other (j) lives on our platonic 2D sheet. By keeping these two worlds separate and maintaining some connection to our perfectly symmetric sheet, we facilitate the study of the crystal symmetries.

We can connect this new way of thinking about the deformation of our sheet with the traditional approach. Knowing how our differentials transform we can measure distances in our original platonic frame, defining our metric on the platonic coordinates. We start with distances as measured in the lab frame:

$$dS^2 = g_{IJ} dX^I dX^J = g_{IJ} F^I{}_j F^J{}_k dx^j dx^k \ , \tag{4.10}$$

and write these in terms of the deformation gradient. We know that the metric in our lab frame is the identity

$$g_{IJ} = \delta_{IJ} = g^{IJ} \ . \tag{4.11}$$

So naturally, this defines our metric tensor in terms of our platonic coordinates

$$dS^2 = g_{jk} dx^j dx^k \ , \tag{4.12}$$

$$g_{jk} = Y^I{}_{/j} Y_{I/k} = F^I{}_j F_{Ik} \ . \tag{4.13}$$

Note that this metric is manifestly symmetric and positive definite. We can think of the act of deformation as curving our two dimensional manifold. Notice also that this is how the strain tensor is usually defined (compare Equation (4.2)):

$$dS^2 = \left(\delta_{ij} + 2\epsilon_{ij}\right) dx^i dx^j \tag{4.14}$$

We've finally identified a connection between the deformation gradient and the strain tensor:

$$g_{jk} = F^I{}_j F_{Ik} = F^T F = \delta_{jk} + 2\epsilon_{ij} \ . \tag{4.15}$$

This strain tensor (a 2×2 symmetric matrix) embodies the stretching and compression of the 2D ideal graphene template, measured in the template's coordinate system.

As a quick summary, while the embedding $Y^{I\alpha}$ contains complete information about the deformation of our material, it has more information than we are interested in. A more natural pair is the average and difference embeddings (Y^I, Δ^I). Still Y^I is a bit excessive. Y^I contains an overall translation in the lab frame that we don't care about, so we are interested principally in gradients of this Y, namely the deformation gradient $F^I{}_j$, which will serve as one of the principle components of our theory along with Δ^I.

4.1.4 Model of Deformation

In order to get a better handle on this new approach to elastic theory, lets consider a simple model where we assume a mostly trivial flat embedding of a graphene sheet in the lab frame with a small perturbation on top. This will lead to some simple expressions. Consider a deformation of the form:

$$\begin{aligned} Y^I &= Y^{0I} + \eta^I && (4.16) \\ &= (x, y, 0) + (\xi_x, \xi_y, h) \; . && (4.17) \end{aligned}$$

The first term embeds our flat sheet in 3D, and the second term adds a small in-plane (ξ) and out-of-plane (h) deformation. With this form of our embedding function, our deformation gradient simplifies to

$$F^I{}_j = \partial_j Y^I \tag{4.18}$$

$$= \partial_j Y^{0I} + \partial_j \eta^I \tag{4.19}$$

$$= \delta^I_j + \partial_j \eta^I \tag{4.20}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} \partial_x \xi_x & \partial_y \xi_x \\ \partial_y \xi_x & \partial_y \xi_y \\ \partial_x h & \partial_y h \end{pmatrix} . \tag{4.21}$$

Here the first term is a near identity that transforms our 2D sheet onto the x-y plane in the 3D lab coordinates, and the second is the Jacobian of the small deformations. From here we can compute the strain tensor

$$F^I{}_j F_{Ii} = \left(\delta^I_i + \partial_i \eta^I\right)\left(\delta_{Ij} + \partial_i \eta_I\right) \tag{4.22}$$

$$= \delta_{ij} + \left(\partial_i \xi_j + \partial_j \xi_i\right) + \left(\partial_i \xi_k \partial_j \xi_k\right) + \left(\partial_i h \partial_j h\right) \tag{4.23}$$

$$= \delta_{ij} + 2\epsilon_{ij} , \tag{4.24}$$

or

$$\epsilon_{ij} = \underbrace{\frac{1}{2}\left[\left(\partial_i \xi_j + \partial_j \xi_i\right) + \underbrace{\left(\partial_i \xi_k \partial_j \xi_k\right)}_{\text{geom}}\right]}_{\text{2d strain}} + \frac{1}{2}\partial_i h \partial_j h . \tag{4.25}$$

Here the term in square brackets is the strain you would expect for a two dimensional material, with the second term in square brackets the geometric nonlinearity. The final term is the second order out of plane deformation contribution to the strain tensor. We've managed to recreate the theory of thin sheets as a natural result of thinking about the deformations of the sheet as an embedding into the lab frame.

Having developed a better understanding of the deformation gradient, let's next consider a concrete example where it offers an advantage over thinking directly in terms of the strain tensor.

4.1.5 Rotation Gradients

Imagine embedding a perfectly flat, rectangular section of graphene as a rolled up tube. This can be accomplished with the embedding

$$Y^I = \left(R \sin \frac{x}{R}, y, R - R \cos xR \right) , \tag{4.26}$$

for which we have the deformation gradient

$$F^I{}_j = \begin{pmatrix} \cos \frac{x}{R} & 0 \\ 0 & 1 \\ \sin \frac{x}{R} & 0 \end{pmatrix} . \tag{4.27}$$

Contracting the transpose of F with itself gives

$$F^T F = F^I_i F^I_j = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} , \tag{4.28}$$

an identity. This leads to an identically vanishing strain tensor:

$$2\epsilon_{ij} = F^I{}_i F_{Ij} - \delta_{ij} = \delta_{ij} - \delta_{ij} = 0 \,! \tag{4.29}$$

This agrees with our intuition. The metric of a smoothly rolled sheet is unchanged.

If we had followed the usual approach to elastic theory, thinking only of the strain tensor, our entire free energy would identically vanish for the curled sheet. And yet, while the strain tensor vanishes, the deformation gradient does not.

We can even imagine terms in the free energy in terms of F that survive. By taking gradients of the deformation gradient, we can see the effects of the curvature of the sheet. For example,

$$\partial_i F^{Ii} \partial_j F_I{}^j = \frac{1}{R^2} , \tag{4.30}$$

is a term that is isotropic and so should appear in any elastic theory that includes gradients of F, and yet does not vanish for the curled sheet. If this term entered into the expression of our free energy, our sheet would resist curling, attempting to maximize its radius of curvature.

Another way to think about this distinction is to perform a polar decomposition on the deformation gradient itself. Thinking about the deformation gradient as a matrix, we can perform a QR decomposition:

$$F_{Ij} = R_{Ik} s_{ki} , \tag{4.31}$$

writing F as a product of a orthogonal matrix $\boldsymbol{R}$ which behaves like a generalized rotation $\boldsymbol{R}^T \boldsymbol{R} = \mathbf{1}$, and $\boldsymbol{s}$, a rank 2 tensor corresponding to in-plane stretches. In the current example, this decomposition would produce

$$F^I_j = \begin{pmatrix} \cos\frac{x}{R} & 0 \\ 0 & 1 \\ \sin\frac{x}{R} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} , \tag{4.32}$$

since the act of rolling behaves as a pure rotation. In this picture it becomes clear that the strain tensor, or products of the strain tensor, formed by contracting F with its transpose, are always independent of this rotation ($\boldsymbol{R}$)

$$\boldsymbol{F}^T \boldsymbol{F} = \boldsymbol{s}^T \boldsymbol{R}^T \boldsymbol{R} \boldsymbol{s} = \boldsymbol{s}^T \boldsymbol{s} = \mathbf{1} + 2\boldsymbol{\epsilon} . \tag{4.33}$$

Such rotation gradient terms are usually omitted (perhaps for a good reason, see section 4.1.5) for 3D elasticity. For the 3D deformations of 2D materials, they are usually introduced 'by hand'. While incorporating rotation gradients systematically provides a primary motivation for the work presented here, our understanding of them remains incomplete (Section 4.1.5).

4.2 Group Theory

I'll start with a very brief overview of group theory, for a more detailed introduction see one of many books on the subject, including [49, 33, 106, 57]

4.2.1 Definitions

A *group* is a set along with an operator $(G, \cdot)$ that satisfies the following axioms:

- Closure
$$\forall a, b \in G : ab \in G$$
- Associativity
$$\forall a, b, c \in G : a(bc) = (ab)c$$
- Identity
$$\exists e \in G \; s.t. \; \forall a \in G : ae = ea = a$$
- Inverses
$$\forall a \in G, \exists a^{-1} \in G \; s.t. \; aa^{-1} = a^{-1}a = e$$

The *order* of a group is the number of elements in the set.

A *class* is an equivalency class on the elements of a group, whereby

$$a \sim b \; if \; \exists c \; s.t. \; a = cgc^{-1} \; . \tag{4.34}$$

A *representation* is a mapping from group elements to matrices that preserve the notation of the group action:

$$R : G \to M_{n,n} \; s.t. \forall a, b \in G, R(a)R(b) = R(ab) \; . \tag{4.35}$$

Two representations are *equivalent* if they are related by a transformation

$$R^1 = R^2 \iff \exists C \; s.t. \; \forall g \in G : R^1(g) = CR^2(g)C^{-1} \; . \tag{4.36}$$

Notice that characters remain unchanged under such a transformation. A *character* is the trace of the representation of a group element

$$\chi(g) = R_{ii}(g) \tag{4.37}$$

Notice also that all elements in the same class share characters.

The *trivial* representation of a group is to replace all elements of the group by the number 1. The character for all elements in this representation is also 1.

We can combine groups to generate new ones. The *direct product* of two groups is the set of tuples of elements of those groups. Given groups G, H we can make up a group $G \otimes H$ where

$$\forall g \in G, h \in H : (g, h) \in G \otimes H \; , \tag{4.38}$$

where we define multiplication to be elementwise

$$(g_1, h_1)(g_2, h_2) = (g_1 g_2, h_1 h_2) \; , \tag{4.39}$$

this is a new group.

There are also operations that we can do to representations to create new ones. Trivially we could send it through an invertible transformation

$$R' = CRC^{-1} \tag{4.40}$$

but more interesting, we can form *direct sums* of representations, where we build up a block matrix of individual representations along the diagonal. This will still be a valid representation

$$R' = R^1 \oplus R^2 \tag{4.41}$$

$$R' = \begin{pmatrix} R^1 & 0 \\ 0 & R^2 \end{pmatrix} \tag{4.42}$$

Additionally, we can form the *direct product* of representations, wherein we take the direct product of the matrices defining the representations. The character of a direct product representation is the product of the characters.

A representation is *reducible* if it is equivalent (by an invertible transformation) to a direct sum of representations. A representation is *irreducible* if it is not *reducible*.

Group theory is really interesting when you talk about *irreducible* representations. All kinds of results exist for them. Namely, there exists a *super orthogonality relation* for irreducible representations.

$$\sum_g R^\alpha_{il}(g)(R^\beta_{jm})^*(g) = \frac{|G|}{n_\alpha}\delta_{\alpha\beta}\delta_{ij}\delta_{lm} \tag{4.43}$$

where $|G|$ is the *order* of the group and n_α is the dimensionality of the representation.

This implies that the number of distinct irreducible representations of a finite group is finite. In particular, the number of nonequivalent irreducible representations of a group must be equal to the number of classes in the group. A similar orthogonality exists for characters

$$\sum_{g} \chi^{\alpha}(g)\chi^{\beta*}(g) = |G|\delta_{\alpha\beta} . \tag{4.44}$$

These orthogonality relations enable us to decompose reducible representations into direct sums of reducible ones.

4.2.2 Decomposition

Given the above, if one has the characters of a particular representation, it is easy to compute how many times each irreducible representation appears, namely

$$a_{\alpha} = \frac{1}{|G|} \sum_{i} g_i X_i^{\alpha*} \chi_i , \tag{4.45}$$

where here g_i marks the count for the order of each class, as we decide to take the sum over classes rather than group elements, χ the characters of our representation and χ^{α} the characters of the irreducible representation.

A neat thing to know is that if you want to test whether a representation is irreducible you can check to see if

$$\sum_{i} g_i |\chi_i|^2 = g , \tag{4.46}$$

which is only true for irreducible representations. Another neat thing is

$$\sum_{i} \chi_i \chi_i^* g_i = g \sum_{\alpha} a_{\alpha}^2 , \tag{4.47}$$

of which the above is a special case.

4.2.3 Linear Operators

As we do physics, often times we have a notion of how things transform in terms of our symmetry operations. These notions can be extended to arbitrary functions. That is, if we know how some x transforms under a group, we can form a representation for how functions of x transform under the group. We just define the function itself to be invariant in the proper way, that is:

$$\psi'(x') = O_T\psi(x') = \psi(x) \qquad if x' = Tx\,. \tag{4.48}$$

We can write this as

$$O_T\psi(Tx) = \psi(x)\,. \tag{4.49}$$

That is: the transformed function of the transformed coordinates is unchanged. Or

$$O_T\psi(x) = \psi(T^{-1}x)\,, \tag{4.50}$$

which is just the above swapping x for Tx.

So, just as we can decompose representations, we can decompose functions into a sum of functions which act as a basis for the various irreducible representations:

$$\psi = \sum_\alpha \sum_i \psi_i^\alpha\,. \tag{4.51}$$

We can decompose by the use of projection operators:

$$P^\alpha = \frac{n_\alpha}{|G|}\sum_g \chi^{\alpha *}(g) O_g\,. \tag{4.52}$$

Of particular interest to us is the projection into the trivial representation, for which

$$P^1 = \frac{1}{|G|}\sum_g O_g\,. \tag{4.53}$$

4.3 Group Theory for Graphene

By thinking of the deformation of our graphene sheet as an embedding function, we have managed to keep clear which elements in our theory naturally live in the three dimensional lab frame and which have their natural home on the perfect two dimensional sheet of graphene that we started with. Given that we have kept these two words separate, we can take full advantage of the symmetries of each to severely restrict the types of terms that might appear in our free energy.

Consider the different kinds of symmetry transformations we can do to our graphene sheet if we are sitting at a high symmetry point, such as the small black dot in the center of a hexagon in Figure 4.1. We can:

1. Rotate the perfect sheet by multiplies of 60°.
2. Reflect the perfect sheet about the horizontal mirror plane.
3. Reflect the sheet about the vertical mirror plane.
4. Rotate the embedded graphene by any angle in any direction in the lab frame.
5. Invert the embedding in the lab frame. (i.e. $Y \to -Y$).

It turns out this group is given by $C_{6v} \otimes I \otimes SO(3)$, where C_{6v} is the rotation group with a 6-fold symmetry axis and an additional mirror symmetry (visualized in Figure 4.2 below), I is the inversion group in three-space and $SO(3)$ the special orthogonal group of order three – the group of all 3D rotations. We'll call this product group $(C_{6v} \otimes I \otimes SO(3))$, G for the graphene group.

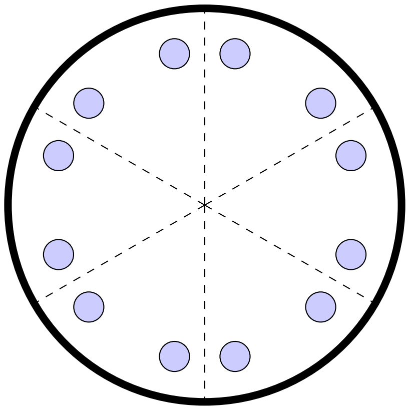


Figure 4.2: A representation of the group C_{6v}, this object has the same symmetry as a flat graphene sheet. Notice that the plate above is invariant to any 60° rotation, as well as vertical or horizontal mirror planes.

I	e	i
A_1	1	1
A_1	1	-1

Table 4.1: The character table for the 2 element group, here used for the 3D spatial inversions (I).

C_{6v}	e	r^3	$r^2(2)$	$r(2)$	$h(3)$	$v(3)$
A_1	1	1	1	1	1	1
$A_2; z$	1	1	1	1	-1	-1
B_1	1	-1	1	-1	1	-1
B_2	1	-1	1	-1	-1	1
E_2	2	2	-1	-1	0	0
$E_1; x, y$	2	-2	-1	1	0	0

Table 4.2: The character table for C_{6v} the dihedral group of order 6, with all of the transformations that leave a two dimensional sheet of graphene invariant.

$SO(3)$	E	$R(\varphi)$
Y_{lm}	$2l+1$	$\frac{\sin\left[\left(l+\frac{1}{2}\right)\varphi\right]}{\sin\left(\frac{\phi}{2}\right)}$

Table 4.3: The character table for $SO(3)$ where the irreducible representations are given by the spherical harmonics and the classes depend only on the angle of rotation.

	C_{6v}						I		$SO(3)$	
	e	r^3	$r^2(2)$	$r(2)$	$h(3)$	$v(3)$	E	I	E	$R(\varphi)$
$Y^{I\alpha}$	2	0	2	0	2	0	1	-1	3	$1 + 2\cos\varphi$
Y^I	1	1	1	1	1	1	1	-1	3	$1 + 2\cos\varphi$
Δ^I	1	-1	1	-1	1	-1	1	-1	3	$1 + 2\cos\varphi$
∂_i, x^i	2	-2	-1	1	0	0	1	1	1	1
ϵ_{ij}	3	3	0	0	1	1	1	1	1	1
F^I_j	2	-2	-1	1	0	0	1	-1	3	$1 + 2\cos\varphi$

Table 4.4: The characters of some objects of interest under the components of the graphene group.

Naturally to understand the nature of the graphene group, it helps to have the character tables for its component groups, shown in Tables 4.1, 4.2 and 4.3.

Given that our graphene group is a direct product group, if we want to compute the characters of a representation, it is enough to determine the characters this representation has under each of the component groups themselves. I've organized these characters below in Table 4.4

As an example, consider the bare, two-component embedding function $Y^{I\alpha}$. Under the C_{6v} graphene operations, for those that take A atoms to A atoms and B atoms to B atoms, namely the even rotations, it will have character 2, as it forms a two dimensional representation on this subgroup and behaves like the identity for those operations. Under inversion I it switches sign since the function itself is a pair of two vectors in 3-space, and under rotations it rotates like an ordinary vector.

Table 4.4 is central to our continuing analysis. It enables us to compute the characters of any term we desire. Any candidate for the free energy expansion can be written in terms of gradients (∂_i), F, Δ and ϵ. Since we've computed the characters of each of these in Table 4.4, and since a direct product of representations has the character of the product of the characters of those representations, it is easy to compute the character of any candidate we wish. Once we know that character of a representation, it is easy to decompose that representation in terms of irreducible representations of our group. We are interested in particular in how many scalars can appear in these representations. To determine how many scalars appear in a representation, we need only sum the character of the representation for each column in Table 4.4, weighted by the number of elements in that class and divide by the order of our group. In the case of the columns corresponding to the rotation group, the sum is replaced by an integral weighted by the density of rotations.

Notice also that we can see by Table 4.4 that in terms of representations $Y^{I\alpha} = Y^I \oplus \Delta^I$, the two component embedding function is the direct sum of its average and difference functions. This is not a surprise but it is nice to know that we could have discovered that taking the sum and difference of the embedding function was the right thing to do from the group theory alone.

The only remaining issue is that often times we want not the direct product representation of an element from Table 4.4, but a symmetric direct product. Consider for instance a term in the free energy involving two powers of epsilon ($\epsilon_{ij}\epsilon_{kl}$). This term is manifestly invariant under the transformation $(i, j) \leftrightarrow (k, l)$, and so its character is not given by the products of the characters of the representation of ϵ directly.

4.3.1 Symmetric Products

In order to determine which terms are allowed to appear, we need to figure out the characters for all of the possible products we want to consider. While determining the characters of these terms, it is not enough to just take the products of the characters. While that works for determining the characters for a direct product representation, we are interested in symmetric direct products.

There is a difference between the outer product of two different vectors and the outer product of the same vector with itself.

$$v_i \otimes w_j \,, \tag{4.54}$$

has as its character the vector representation's character squared, while

$$v_i \otimes v_j \,, \tag{4.55}$$

is different. We need it to be manifestly symmetric on the two indices.

We need a way to compute the character for a general symmetric product of representations [49]. I found the formula to do so. It is recursive:

$$\chi(R)^{\{n\}} = Z(S_n) \tag{4.56}$$

with

$$Z(S_0) = 1 \qquad Z(S_n) = \frac{1}{n}\sum_{l=1}^{n} \chi\left(R^l\right) Z(S_{n-l}) \,. \tag{4.57}$$

For illustration, here are the first few terms

$$\chi(R)^{\{1\}} = \chi(R) \tag{4.58}$$

$$\chi(R)^{\{2\}} = \frac{1}{2}\left(\chi^2(R) + \chi\left(R^2\right)\right) \tag{4.59}$$

$$\chi(R)^{\{3\}} = \frac{1}{6}\left(\chi^3(R) + 3\chi(R)\chi\left(R^2\right) + 2\chi\left(R^3\right)\right) \tag{4.60}$$

$$\chi(R)^{\{4\}} = \frac{1}{24}\left(\chi^4(R) + 6\chi^2(R)\chi\left(R^2\right) + 3\chi^2\left(R^2\right) + 8\chi(R)\chi\left(R^3\right) + 6\chi\left(R^4\right)\right) \,. \tag{4.61}$$

Notice that this recursive formula allows us to calculate the characters for a symmetric direct product of a representation with itself, but it relies on the ability to calculate the characters of the product of the representation elements themselves.

4.3.2 Representation of the Graphene Group

For our graphene group, we can give an explicit representation for a vector from our platonic realm for a single element from each class of our group:

$$R(e) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.62}$$

$$R(r) = \frac{1}{2}\begin{pmatrix} 1 & -\sqrt{3} & 0 \\ \sqrt{3} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R(r^2) = \frac{1}{2}\begin{pmatrix} -1 & -\sqrt{3} & 0 \\ \sqrt{3} & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.63}$$

$$R(r^3) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R(h) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.64}$$

$$R(v) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R(I) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tag{4.65}$$

This enables the construction of the character table (Table 4.4), well this with a representation for a vector in $SO(3)$:

$$R(R(\phi)) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} . \tag{4.66}$$

Although we should note, for the elements of $SO(3)$ we have a closed form expression for the form of the character of the product of the representation.

$$\chi\left(R(\phi)^k\right) = 1 + 2\cos k\phi . \tag{4.67}$$

I should also mention that our notion of orthogonality is modified for $SO(3)$, given that it is a Lie Group, and thus continuous. Instead of summing over all of the group elements, we must perform an integral over the governing parameter for the character. We also require a proper measure on the space. For $SO(3)$, orthogonality is defined by:

$$\frac{1}{\pi}\int_0^\pi d\phi\,(1-\cos\phi)\chi^{\alpha *}(\phi)\chi^{\beta}(\phi) = \delta_{\alpha\beta} . \tag{4.68}$$

4.4 Free Energy Expansion

In considering the most general form of the free energy, the following elements are allowed to appear:

$$\epsilon_{ij}, \partial_k F^I_j, \Delta_I , \tag{4.69}$$

along with any combination or gradient therein. Here ϵ is the strain tensor, Δ_I the difference embedding and $\partial_k F^I_j$ a gradient of the deformation tensor. The gradient ensures that we are dealing with a small quantity. Since F^I_j itself is not small, so we cannot let it alone appear in the expansion unless it is accompanied by another small term. In fact, if we estimate the magnitude of our deformation takes with some characteristic scale A, and that it varies over some characteristic length σ, and that the Δ fluctuations have some size Δ, we can form order of magnitude estimates of the magnitude of the contribution of any term as:

$$F_i^I \sim 1 \quad \epsilon_{ij} \sim \frac{A}{\sigma} \quad \Delta_I \sim \Delta \quad \partial_i \sim \frac{1}{\sigma}\,, \tag{4.70}$$

and do a three length scale expansion in the limit that A, Δ are small and σ is large. Each term then has a leading order contribution and we can expand in terms of powers of these small parameters.

Finding potential terms for the free energy allowed by symmetry is made easier by the presence of irreducible invariants. We are used to only being allowed to contract tensors in physics with δ_{ij} and ϵ_{ijk}. Why is that? Those tensors are irreducible invariants for the rotation group $SO(3)$. Actually ϵ_{ijk} behaves as a pseudoscalar, and so unless we contract an element that also switches sign under inversion, it is only allowed by itself in pairs. For the graphene group, we will similarly have δ_{ij} and ϵ_{ijk} as potential irreducible invariants, but additional ones show up because of the reduced symmetry. We don't know of a systematic way to find these irreducible tensors ourselves, but we have found a pseudo scalar rank three tensor T_{ijk} (Triangular) and a fully symmetric rank six tensor $H_{ijklmn} = T_{ijk}T_{lmn}$ (Hexagonal). We also have evidence of the existence of an independent irreducible invariant of rank 5: P_{ijklm} (Pentagonal), again with pseudo scalar characteristics. Explicitly, T takes the form:

$$T_{ijk} = \begin{pmatrix} \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix}, \tag{4.71}$$

and has a pseudo scalar like transformation structure, which enables it to form completely invariant terms when combined with a single Δ_I and another rank two object.

We can begin to enumerate all possible terms that are allowed to appear in the free energy. For any possible combination of elements

$$\left(\epsilon_{ij}\right)^{\alpha} (\partial_k)^{\beta} \left(F^I_j\right)^{\gamma} \left(\Delta^I\right)^{\delta} \tag{4.72}$$

we can immediately compute the character as the direct product of the symmetric direct products of each of the representations in our term. Knowing the character, we can immediately calculate its projection onto the trivial representation to obtain a count of how many independent scalar representations the term contains. After that, things get a little more complicated. Finding an explicit basis for those scalar terms demands a bit of artistry. We are guided by the existence of our irreducible invariants, but subtleties quickly emerge. For one, there are nontrivial equivalences that can come out, wherein terms that appear to be different are actually linearly dependent. Furthermore, our group theoretic calculation can only tell us how many gradients might appear, it doesn't help us resolve the orderings of those gradients in the scales A, σ and Δ (eqn. 4.70). Since the gradient is an operator, there is an additional calculation that must be done to resolve how many independent terms we can produce. Here too there will be nontrivial linear dependencies, due in large part to boundary terms, which may or may not vanish upon integration by parts, depending on our particular configuration. Gradients also have hidden symmetries associated with the symmetry of repeated gradients acting on the same object. We know that second partial derivatives must be symmetric, but our group theory doesn't know about this symmetry directly, as the group theory analysis is operator-ordering dependent.

4.4.1 Determining the Number of Scalars

Let's work an illustrative example. Let's try to determine the number of scalars that can appear in a term of the form ϵ_{ij}, which itself is a shorthand for a representation of the form $x_i x_j$, that is a symmetric product of two 2D vector representations. Let's assume we do not yet know the representation for ϵ_{ij}; how would be find it in terms of our explicit representation of the elements of our group in Section 4.3.2?

We are interested in the symmetric direct product of the 2D x_i representation with itself. First we recreate the line in the character table corresponding to our two dimensional vector by taking the traces of each of the representative class members. We find:

$$\chi(R) = \{2, -2, -1, 1, 0, 0, 2, -2, -1, 1, 0, 0\}1 \ . \tag{4.73}$$

Remember that the character table was actually shorthand for a larger direct product group. This means that to represent the characters for each class in our product group requires a $6 \times 2 \times f$ dimensional vector, where f denotes the function for the character of the $SO(3)$ group.

We also need the characters for the squares of all of our representation matrices, for which we find:

$$\chi(R^2) = \{2, 2, -1, -1, 2, 2, 2, 2, -1, -1, 2, 2, 2, 2\}1 \ . \tag{4.74}$$

Notice that this corresponds to the ϵ_{ij} row of our character table 4.4. The representation for the symmetric direct product of this two dimensional representation with itself is therefore 4.57:

$$\chi(R)^{\{2\}} = \frac{1}{2}\left(\chi(R)^2 + \chi(R^2)\right) = \{2, 0, -1, 0, 1, 1, 2, 0, -1, 0, 1, 1\}1 \ . \tag{4.75}$$

To determine the number of scalars present in this term, we need only determine the number of times the trivial representation appears in this representation. We can do this by utilizing the orthogonality theorem and taking the proper dot product of our characters with the corresponding characters for the trivial representation:

$$\chi_0 = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}1 \,. \tag{4.76}$$

In order to properly weight the order of each class, and account for the $\frac{1}{g}$ in the orthogonality relation, we can directly compute the number of scalars by computing:

$$\langle 0 \mid R \rangle = \frac{1}{24\pi} \int_0^\pi (1 - \cos\phi)\{1, 1, 2, 2, 3, 3, 1, 1, 2, 2, 3, 3\} \cdot \chi(R) \,. \tag{4.77}$$

For our particular case this yields simply: 1. There is only a single scalar possible for a term of the form ϵ_{ij}, as we might have expected.

By this same procedure, we can determine the number of scalars possibly present in any general term.

4.5 Free Energy Terms

Here I'll report the independent terms I've found in the free energy up to fourth order in our small terms (A, ϵ, Δ, 4.70), without considering gradients. For each, I'll give the explicit contraction as well as a shorthand name for use in referring to terms in the results table.

4.5.1 First Order Terms

The only first order term allowed by symmetry is the trace of the strain tensor:

$$\epsilon_{ii} \equiv \epsilon \tag{4.78}$$

formed from the strain, contracted against δ_{ij}. This couples to an overall pressure in our system.

4.5.2 Second Order Terms

Things are more interesting at second order. I've found that the following terms are independent and span the space:

$$\epsilon_{ii}\epsilon_{jj} \equiv \epsilon^2 \tag{4.79}$$

$$\epsilon_{ij}\epsilon_{ij} \equiv \epsilon_{ij}^2 \tag{4.80}$$

$$T_{ijk}F_i^I\Delta^I\epsilon_{jk} \equiv TF\Delta\epsilon \tag{4.81}$$

$$\Delta^I\Delta^I \equiv \Delta^2 \tag{4.82}$$

$$F_i^I\Delta^I F_i^J\Delta^J \equiv (F\Delta)^2 \tag{4.83}$$

You'll notice the presence of the two ordinary linear elastic theory terms ($\epsilon^2, \epsilon_{ij}^2$). These would be present in a completely isotropic theory. But you'll also notice additional terms that should contribute at roughly this order, including terms both linear and quadratic in Δ, which are due to the fact that that graphene need not necessarily satisfy the Cauchy-Born rule [12].

4.5.3 Third Order Terms

$$\epsilon_{ii}\epsilon_{jj}\epsilon_{kk} \equiv \epsilon^3 \tag{4.84}$$

$$\epsilon_{ij}\epsilon_{jk}\epsilon_{ki} \equiv \epsilon_{ij}\epsilon_{jk}\epsilon_{ki} \tag{4.85}$$

$$T_{ijk}T_{lmn}\epsilon_{ij}\epsilon_{kl}\epsilon_{mn} \equiv TT(\epsilon\epsilon\epsilon) \tag{4.86}$$

$$\Delta^I\Delta^I\epsilon_{ii} \equiv \Delta^2\epsilon \tag{4.87}$$

$$T_{ijk}F^I_i\Delta^I F^J_j\Delta^J F^K_k\Delta^K \equiv T(F\Delta)^3 \tag{4.88}$$

$$F^I_i\Delta^I F^J_j\Delta^J\epsilon_{ij} \equiv (F\Delta)^2_{ij}\epsilon_{ij} \tag{4.89}$$

$$F^i_i\Delta^I F^J_i\Delta^J\epsilon_{kk} \equiv (F\Delta)^2\epsilon \tag{4.90}$$

$$T_{ijk}F^I_i\Delta^I\epsilon_{jk}\epsilon_{ll} \equiv T(F\Delta\epsilon)\epsilon \tag{4.91}$$

$$T_{jkl}F^I_i\Delta^I\epsilon_{ij}\epsilon_{kl} \equiv (F\Delta)T(\epsilon\epsilon\epsilon) \tag{4.92}$$

Here we see the fist appearance of a term that would show up in traditional elastic theory but not for an isotropic theory $TT(\epsilon\epsilon\epsilon) = H(\epsilon\epsilon\epsilon)$. This term is due to the reduced symmetry of the 6-fold rotation axis.

The third order terms also help illustrate how difficult finding these independent scalars can be. Group theory tells us that there should be three independent scalars formed from contractions of the form $\epsilon_{ij}\epsilon_{kl}\epsilon_{mn}$. Naively, we would expect these to be:

$$\epsilon_{ii}\epsilon_{jj}\epsilon_{kk} \tag{4.93}$$

$$\epsilon_{ij}\epsilon_{jk}\epsilon_{ki} \tag{4.94}$$

$$\epsilon_{ij}\epsilon_{ij}\epsilon_{kk}\ . \tag{4.95}$$

However, it turns out that the last term $\epsilon_{ij}\epsilon_{ij}\epsilon_{kk}$ is actually a linear combination of the first two because of a peculiarity of two dimensions. We realized these terms were linearly dependent after attempting to fit the coefficients and discovering a singularity in the design matrix for the fit. Indeed, singularities in the design matrix has generally been a useful diagnostic for unexpected linear dependencies.

Just as group theory allows us to determine the number of scalars present in any particular term, we can also explicitly construct the projection of that scalar representation. This is expressed as a large, sparse matrix of weights when rendered as an explicit representation. For the case of $\epsilon_{ij}\epsilon_{kl}\epsilon_{mn}$ in particular, this is a $2^6 = 64$ dimensional representation. Projecting out the components corresponding to ϵ_{ii}^3 and $\epsilon_{ij}\epsilon_{jk}\epsilon_{ki}$ exposed a linear subspace involving our invariant rank 6 tensor H_{ijklmn}. One can show that the completely symmetric tensor H factors into the product of two rank 3 pseudo invariant tensors $T_{ijk}T_{lmn}$.

4.5.4 Fourth Order Terms

$$\epsilon_{ii}\epsilon_{jj}\epsilon_{kk}\epsilon_{ll} \equiv \epsilon^4 \tag{4.96}$$

$$\epsilon_{ij}\epsilon_{jk}\epsilon_{kl}\epsilon_{li} \equiv \epsilon_{ij}\epsilon_{jk}\epsilon_{kl}\epsilon_{li} \tag{4.97}$$

$$\epsilon_{ij}\epsilon_{ij}\epsilon_{kk}\epsilon_{ll} \equiv \epsilon_{ij}^2\epsilon^2 \tag{4.98}$$

$$T_{ijk}T_{lmn}\epsilon_{ij}\epsilon_{kl}\epsilon_{mn}\epsilon_{pp} \equiv TT(\epsilon\epsilon\epsilon\epsilon) \tag{4.99}$$

$$\Delta^I\Delta^I\Delta^J\Delta^J \equiv \Delta^4 \tag{4.100}$$

$$F_i^I\Delta^I F_i^J\Delta^J F_j^K\Delta^K F_j^L\Delta^L \equiv (F\Delta)^4 \tag{4.101}$$

$$\Delta^I\Delta^I\epsilon_{ii}\epsilon_{jj} \equiv \Delta^2\epsilon^2 \tag{4.102}$$

$$\Delta^I\Delta^I\epsilon_{ij}\epsilon_{ij} \equiv \Delta^2\epsilon_{ij}^2 \tag{4.103}$$

$$T_{ijk}\epsilon_{ij}F_{kI}\Delta^I\Delta^J\Delta^J \equiv T(\epsilon F\Delta)\Delta^2 \tag{4.104}$$

$$F_i^I\Delta^I F_i^J\Delta^J\Delta^K\Delta^K \equiv (F\Delta)^2\Delta^2 \tag{4.105}$$

$$T_{ijk}F_i^I\Delta^I F_j^J\Delta^J F_k^K\Delta^K\epsilon_{ll} \equiv T(F\Delta)^3\epsilon \tag{4.106}$$

$$T_{ijk}F_i^I\Delta^I\epsilon_{jk}F_l^L\Delta^L F_l^M\Delta^M \equiv T(F\Delta\epsilon)(F\Delta)^2 \tag{4.107}$$

$$T_{ijk}F_i^I\Delta^I\epsilon_{jk}\epsilon_{ll}\epsilon_{mm} \equiv T(F\Delta\epsilon)\epsilon^2 \tag{4.108}$$

$$T_{ijk}F_i^I\Delta^I\epsilon_{jk}\epsilon_{lm}\epsilon_{ml} \equiv T(F\Delta\epsilon)\epsilon_{ij}^2 \tag{4.109}$$

$$T_{ijk}T_{lmn}F_i^I\Delta^I F_j^J\Delta^J\epsilon_{kl}\epsilon_{mn} \equiv (F\Delta\epsilon)T\epsilon(\epsilon) \tag{4.110}$$

$$F_i^I\Delta^I F_i^J\Delta^J\epsilon_{kk}\epsilon_{ll} \equiv TT((F\Delta)(F\Delta)\epsilon\epsilon) \tag{4.111}$$

$$F_i^I\Delta^I F_i^J\Delta^J\epsilon_{kl}\epsilon_{lk} \equiv (F\Delta)^2\epsilon_{ij}^2 \tag{4.112}$$

$$F_i^I\Delta^I\epsilon_{ij}F_j^J\Delta^J\epsilon_{kk} \equiv (F\Delta)_i\epsilon_{ij}(F\Delta)_j \tag{4.113}$$

4.5.5 Fifth and Higher Order Terms

The following fifth and higher-order terms are included in the fits that follow. They are linearly independent, but are not exhaustive or systematically chosen. They were included to help resolve the fits for some lower order terms, where these terms are natural extensions of some previous terms.

$$\epsilon_{ii}\Delta^J\Delta^J\Delta^K\Delta^K \equiv \Delta^4\epsilon \tag{4.114}$$

$$T_{ijk}F_i^I\Delta^I\Delta^J\Delta^J\epsilon_{jk}\epsilon_{ll} \equiv (F\Delta)T(\epsilon)\Delta^2\epsilon \tag{4.115}$$

$$T_{jkl}F_i^I\Delta^I\Delta^J\Delta^J\epsilon_{ij}\epsilon_{kl} \equiv (F\Delta\epsilon)T(\epsilon)\Delta^2 \tag{4.116}$$

$$T_{ijk}F_i^I\Delta^I F_j^J\Delta^J F_k^K\Delta^K\Delta^L\Delta^L \equiv T(F\Delta)^3\Delta^2 \tag{4.117}$$

$$F_i^I\Delta^I F_i^J\Delta^J\epsilon_{jj}\Delta^K\Delta^K \equiv (F\Delta)^2\epsilon\Delta^2 \tag{4.118}$$

$$F_i^I\Delta^I F_j^J\Delta^J\epsilon_{ij}\Delta^K\Delta^K \equiv (F\Delta)\epsilon(F\Delta)\Delta^2 \tag{4.119}$$

$$F_i^I\Delta^I F_i^J\Delta^J\Delta^K\Delta^K\Delta^L\Delta^L \equiv (F\Delta)^2\Delta^4 \tag{4.120}$$

$$T_{ijk}F_i^I\Delta^I\epsilon_{jk}\Delta^J\Delta^J\Delta^K\Delta^K \equiv (F\Delta)T(\epsilon)\Delta^4 \tag{4.121}$$

$$\epsilon_{ii}\epsilon_{jj}\Delta^J\Delta^J\Delta^K\Delta^K \equiv \epsilon^2\Delta^4 \tag{4.122}$$

$$\epsilon_{ij}\epsilon_{ji}\Delta^J\Delta^J\Delta^K\Delta^K \equiv \epsilon_{ij}^2\Delta^4 \tag{4.123}$$

$$\Delta^I\Delta^I\epsilon_{ii}\epsilon_{jj}\epsilon_{kk}\epsilon_{ll} \equiv \Delta^2\epsilon^4 \tag{4.124}$$

$$\Delta^I\Delta^I\epsilon_{ij}\epsilon_{jk}\epsilon_{kl}\epsilon_{li} \equiv \Delta^2\epsilon_{ij}\epsilon_{jk}\epsilon_{kl}\epsilon_{li} \tag{4.125}$$

$$\Delta^I\Delta^I\epsilon_{ij}\epsilon_{ji}\epsilon_{kk}\epsilon_{mm} \equiv \Delta^2\epsilon_{ij}^2\epsilon^2 \tag{4.126}$$

$$\Delta^I\Delta^I T_{ijk}T_{lmn}\epsilon_{ij}\epsilon_{kl}\epsilon_{mn}\epsilon_{pp} \equiv \Delta^2 TT(\epsilon\epsilon\epsilon\epsilon) \tag{4.127}$$

$$\epsilon_{ii}\Delta^J\Delta^J\Delta^K\Delta^K\Delta^L\Delta^L \equiv \Delta^6\epsilon \tag{4.128}$$

4.6 Simulation

In order to estimate the value for the elastic constants in the expansion in Section 4.4, I deform graphene sheets and measure the forces according to available potentials.

The largest fit I attempted with some success was one involving all of the terms reported above. Each term came in linearly with an unknown elastic constant. I randomly strained an armchair unit cell graphene sheet, and extracted the potential energy from a simulation with the Airebo potential [100] implemented in LAMMPS [81] with the python package `ase` (Atomic Simulation Environment) [4].

Below in Table 4.5 I show the resulting fitted values for all of the first 50 elastic constants. Table 4.6 has the extracted elastic constants in terms of our expansion. Figure 4.3 shows both the simulation and fitted energies across all configurations, as well as the residuals. The residuals are a part in 10^5 of the energy of the system, giving us a high degree of confidence that I've managed to capture all of the relevant terms up to fourth order in the energy.

id	term	value (TPa)
0	c	$-1.485021(2) \times 10^{1}$
1	ϵ	$-1.0(6) \times 10^{-4}$
2	ϵ^2	$3.78(7) \times 10^{0}$
3	ϵ_{ij}^2	$2.787(9) \times 10^{1}$
4	$TF\Delta\epsilon$	$2.63(1) \times 10^{1}$
5	$\Delta_I\Delta_I$	$1.2(2) \times 10^{0}$
6	$(F\Delta)^2$	$1.59(2) \times 10^{1}$
7	ϵ^3	$-1.25(5) \times 10^{1}$
8	$\epsilon_{ij}\epsilon_{jk}\epsilon_{ki}$	$-1.13(1) \times 10^{2}$
9	$TT(\epsilon\epsilon\epsilon)$	$-5.7(1) \times 10^{0}$
10	$\Delta^2\epsilon$	$-5.9(9) \times 10^{0}$
11	$T(F\Delta)^3$	$-6.4(6) \times 10^{1}$
12	$(F\Delta)_{ij}^2\epsilon_{ij}$	$-5.6(7) \times 10^{1}$
13	$(F\Delta)^2\epsilon$	$-7.4(8) \times 10^{1}$
14	$T(F\Delta\epsilon)\epsilon$	$-8.6(6) \times 10^{1}$
15	$(F\Delta)T(\epsilon\epsilon)$	$-2.33(6) \times 10^{2}$
16	ϵ^4	$8.2(2) \times 10^{2}$
17	$\epsilon_{ij}\epsilon_{jk}\epsilon_{kl}\epsilon_{li}$	$1.6(2) \times 10^{3}$
18	$\epsilon_{ij}^2\epsilon^2$	$-1.9(4) \times 10^{3}$
19	$TT(\epsilon\epsilon\epsilon\epsilon)$	$1.4(1) \times 10^{2}$
20	Δ^4	$1.7(3) \times 10^{2}$
21	$(F\Delta)^4$	$4.7(5) \times 10^{2}$
22	$\Delta^2\epsilon^2$	$-4.5(6) \times 10^{2}$
23	$\Delta^2\epsilon_{ij}^2$	$1.2(8) \times 10^{3}$
24	$T(\epsilon F\Delta)\Delta^2)$	$-4.3(8) \times 10^{2}$
25	$(F\Delta)^2\Delta^2$	$1.3(8) \times 10^{3}$
26	$T(F\Delta)^3\epsilon$	$4.0(2) \times 10^{2}$
27	$T(F\Delta\epsilon)(F\Delta)^2$	$-1.3(6) \times 10^{3}$
28	$T(F\Delta\epsilon)\epsilon^2$	$1.5(2) \times 10^{2}$
29	$T(F\Delta\epsilon)\epsilon_{ij}^2$	$-1.1(1) \times 10^{3}$
30	$(F\Delta\epsilon)T\epsilon(\epsilon)$	$1.9(2) \times 10^{3}$
31	$TT((F\Delta)(F\Delta)\epsilon\epsilon)$	$2.3(2) \times 10^{3}$
32	$(F\Delta)^2\epsilon^2$	$-7.5(3) \times 10^{2}$
33	$(F\Delta)^2\epsilon_{ij}^2$	$2.0(3) \times 10^{3}$
34	$(F\Delta)_i\epsilon_{ij}(F\Delta)_j\epsilon$	$-2.3(3) \times 10^{2}$
35	$\Delta^4\epsilon$	$4.2(4) \times 10^{4}$
36	$(F\Delta)T(\epsilon)\Delta^2\epsilon$	$1.5(2) \times 10^{4}$
37	$(F\Delta\epsilon)T(\epsilon)\Delta^2$	$-3.5(2) \times 10^{4}$
38	$T(F\Delta)^3\Delta^2$	$2.5(1) \times 10^{4}$
39	$(F\Delta)^2\epsilon\Delta^2$	$-2.5(2) \times 10^{4}$
40	$(F\Delta)\epsilon(F\Delta)\Delta^2$	$3.5(2) \times 10^{4}$
41	$(F\Delta)^2\Delta^4$	$-1.6(7) \times 10^{6}$
42	$(F\Delta)T(\epsilon)\Delta^4$	$5(116) \times 10^{4}$
43	$\epsilon^2\Delta^4$	$8.7(1) \times 10^{5}$
44	$\epsilon_{ij}^2\Delta^4$	$-9(12) \times 10^{5}$
45	$\Delta^2\epsilon^4$	$-1.2(1) \times 10^{6}$
46	$\Delta^2\epsilon_{ij}\epsilon_{jk}\epsilon_{kl}\epsilon_{li}$	$-2.3(1) \times 10^{6}$
47	$\Delta^2\epsilon_{ij}^2\epsilon^2$	$2.8(2) \times 10^{6}$
48	$\Delta^2TT(\epsilon\epsilon\epsilon\epsilon)$	$-1.7(8) \times 10^{6}$
49	$\Delta^6\epsilon$	$-5.4(4) \times 10^{7}$

Table 4.5: The results of a fit for the 50 terms in the free energy over 120 instances, applying a random strain of order 10^{-2}. Note that the resulting energy expression fits each data point to within 10^{-5}. Note, however that some high order terms (e.g. #42) are not well determined.

constant	value (TPa)
λ	0.61(1)
μ	2.24(1)
K	0.84(1)
E	0.792(5)
ν	0.107(2)

Table 4.6: The Elastic constants extracted from the fit for the Airebo potential.

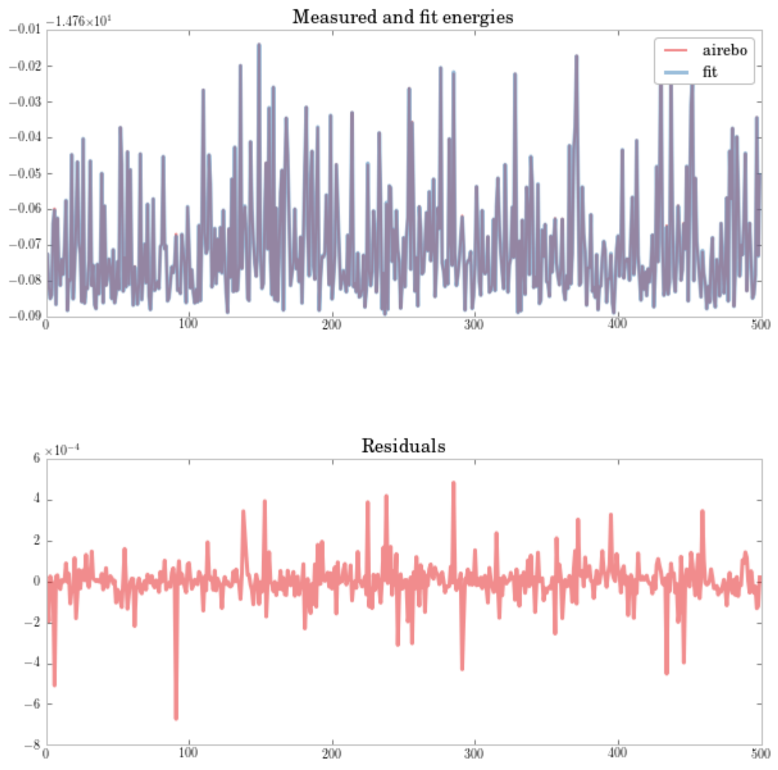


Figure 4.3: The simulated and fit values for the energy across all tested configurations, as well as the measured residuals. Notice that the fit is accurate to a part in 10^5 giving us confidence that we have captured all of the relevant terms up to fourth order.

4.7 Adding Dispersion

So far, we've ignored the possible gradient terms in our free energy expansion. Such terms describe dispersion (the wavelength-dependence of the sound velocity), and also the large strain gradients associated with crystalline defects. Dispersion introduces significant complications.

Let's consider adding terms formed as contractions of the following:

$$\partial_i F^I_j \partial_k F^J_l \ . \tag{4.129}$$

First, since we are trying to describe a free energy, our free energy is unique only up to the inclusion of a total derivative term, which will only contribute at the boundary. Assuming we don't have anything interesting happening at the boundary, this allows us to move around the partial derivatives as allowed by integration by parts. At present this would suggest this term is equivalent to

$$F^I_i \partial_j \partial_k F^J_l \ , \tag{4.130}$$

where both of the partial derivatives act on the same instance of F. Now, we have an additional symmetry not necessarily incorporated into our group theoretic description thus far: that second partial derivatives must be symmetric under interchange.

For this term, we can now ask how many scalars should be present. Treating this as a representation formed from the symmetric direct product of the 2D vector representation for the partials, and the symmetric direct product of the representation of F, we find that there should be 2 independent terms. Intuition would suggest these are the contractions:

$$\partial_i F^I_i \partial_j F^I_j \tag{4.131}$$

$$\partial_i F^I_j \partial_i F^I_j \, . \tag{4.132}$$

But we could also imagine orderings such as

$$\partial_i \partial_j \left(F^I_i F^I_j \right) = \partial_i \partial_j \epsilon_{ij} \tag{4.133}$$

$$\partial_i \partial_i \left(F^I_j F^I_j \right) = \partial_i \partial_i \epsilon_{jj} \, , \tag{4.134}$$

for which the partial derivatives will strip out the constant contribution and leave us with a higher order term, but still seemingly allowed. However, these terms are total derivatives, and so would contribute only on the boundary. Thus 4.131 could be a topological invariant – not ignorable, but whose contribution is wholly dependent on boundary conditions, independent of the bulk.

Things are further muddied by nontrivial dependencies in these gradient terms. For instance, if one ignores the geometric nonlinearity, one can demonstrate [1, 97] that,

$$\nabla \times \epsilon \times \nabla = 0 \tag{4.135}$$

$$\varepsilon_{ipm} \varepsilon_{jqn} \partial_p \partial_q \epsilon_{mn} = 0 \tag{4.136}$$

$$\partial_k \partial_l \epsilon_{ij} + \partial_j \partial_j \epsilon_{kl} - \partial_j \partial_k \epsilon_{il} - \partial_i \partial_l \epsilon_{jk} = 0 \tag{4.137}$$

These three expressions are equivalent. This can be shown directly by substituting in $\epsilon_{ij} = \frac{1}{2} \left(\partial_i \xi_j + \partial_j \xi_i \right)$ and simplifying. This result comes as a nontrivial consequence of the fact that our strains are not allowed to be arbitrary, but instead must be admissible in terms of some actual displacement field. Even worse, this result is used to prove in some accounts that the displacement field can be derived from the strain field itself.

$$\xi_i(\boldsymbol{x}) = \int_{\boldsymbol{x}_0}^{\boldsymbol{x}} d\eta_j \left[\epsilon_{ij}(zi) + (x_k - \eta_k) \left(\partial_k \epsilon_{ij}(\eta) - \partial_i \epsilon_{kj}(\eta) \right) \right] + w_i \tag{4.138}$$

where w_i represents the contribution of a rigid body motion. Results such as these seem to suggest that no gradients of F should survive, as all of them should be representable as various terms involving ϵ. Indeed, preliminary work does indicate that, at least ignoring the geometric nonlinearity, $F^I_i \partial_i \partial_j F^I_j$ is not independent of higher order ϵ terms. However, as we discuss in Section 4.1.5, $\partial_i F^I_i \partial_j F^I_j$ is non-zero for a cylindrical graphene nanotube, with $\epsilon = 0$, so definitely cannot be written in terms of gradients of ϵ. It has been suggested to us that the importance of rotation gradients for 2D membranes in 3D may be related to the need for both the first and second fundamental forms to characterize the embedding; only the first fundamental form (strain) is apparently needed to describe 3D embeddings into 3D.

Between the complicated combinatorics involved in the non-commuting nature of the gradients and the nontrivial effect that has on our group theoretic description, the extra symmetries that multiple gradients introduce, and these potential nontrivial relationships between gradients of F and terms involving ϵ, we have not yet satisfactorily built up a machinery capable of generating linearly independent gradient terms in the expansion of the free energy.

We speculate that restructuring our approach in the language of Clifford Algebra [37, 19, 58, 5] might help sort out some of these difficulties. In particular, the conformal geometric algebra [37, 38] contains both translations and rotations as elements of the same algebra as the geometric entities themselves. Restoring the symmetry between translation and rotation might help resolve some of the difficulties introduced by their current split. Such an approach also offers the potential of incorporating not just the point group symmetries of the graphene lattice, but its translational symmetries as well.

Introducing gradient terms, will also complicate our numerical analysis. We can no longer simulate simple strains on unit cells. Instead we must apply spatially dependent perturbations to a larger lattice, such as the one depicted below in Figure 4.4. The general approach is to start with a flat graphene sheet and put on a Gaussian random initial displacement field to give smooth deformation field in which one can control both the magnitude of the deformation as well as its characteristic wavelength. A proper analysis could extend our fits beyond considering the energy alone to fitting the forces on each atom as well. Knowing the displacements enforced on the sheet, the calculation of the forces from the free energy is straightforward. Estimating the elastic constants amounts to doing a linear least squares fit for the coefficients relating the predicted forces of each term with the observed forces due to the interatomic potential used.

4.8 Extensions and Discussion

We've managed to build a procedure for creating general non-linear elastic theories for materials, using as input only the symmetry properties of the material itself. This creates a rich starting ground for several interesting calculations. This project is in many ways still in its infancy. There is much work left to be done. Below is a taste of the types of things that should be done, and the directions we might be able to pursue if the project is continued.

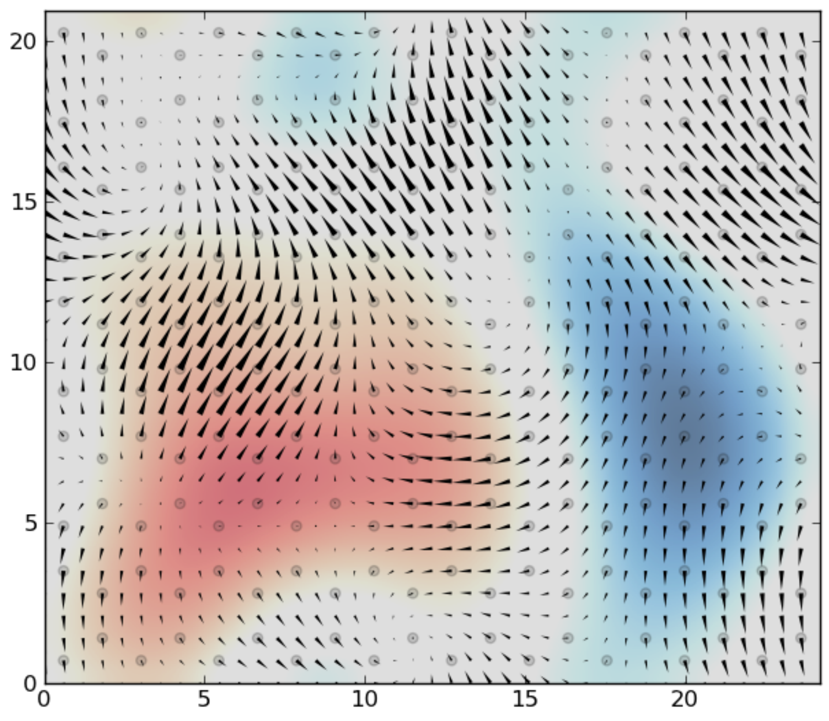


Figure 4.4: An example Gaussian random initial configuration for the displaced graphene with in-plane displacements depicted with arrows and out-of-plane displacements shown in the background color map.

4.8.1 Integrating out Delta

Our free energy expansion has lots of terms involving the difference embedding Δ. Experimental measures of the elastic constants of graphene do not probe these directly. The Δ field corresponds to optical modes of vibration of our unit cells, these characteristically have higher frequencies, and when bulk measurements are done on graphene sheets the constants reported are effective elastic constants after the unit cells themselves have relaxed. These relaxed elastic constants multiply precisely the terms we calculate that do not explicitly involve Δ.

In terms of the complete free energy functional, these effective constants are the ones that remain once we've set Δ to its minimum. These corrections to nonlinear elastic constants will presumably only involve terms in Δ of lower order. A perturbative calculation of the relaxed constants explicitly in term of ours should be possible, if complex. The complexity could be minimized by noting that each term in such a formula must transform under our symmetry group with the representation corresponding to Δ_I.

To the lowest order, taking a variation of our free energy with respect to Δ we obtain

$$\alpha_3 \Delta^I \sim -\alpha_2 T_{ijk} F^I_i \epsilon_{jk} \,, \tag{4.139}$$

which suggests, to lowest order, our minimum Δ should be

$$\Delta^I \sim -\frac{\alpha_2}{\alpha_3} T_{ijk} F^I_i \epsilon_{jk} \,. \tag{4.140}$$

In fact, we have already used this assumption when deciding which terms to keep in the free energy expansion; the above result giving us an estimate for the magnitude of the Δ field once the unit cells have relaxed. An open question is the details of how this result is modified when considering the higher order terms in the free energy and whether more progress can be made in determining the true effective elastic constants of our theory in order to compare with experiment.

One can of course numerically measure the relaxed elastic constants directly [51], with perhaps substantially less numerical effort. As such we forgo this analysis here.

4.8.2 Exploring other Materials

In some ways, while graphene is a perfect candidate for this kind of non-linear elastic theory with gradient terms, given its 2D nature and inherent strength, the analysis is greatly complicated by the fact that graphene does not form a Bravais lattice. Nonlinear elastic theories for traditional 3D Bravais lattices (e.g. fcc, bcc) should be simpler, since there would be only a single embedding function (Y), i.e. no Δ field.

4.8.3 Exploring other potentials

Since the free energy expansion should be complete, this approach could offer a unique way to test the effectiveness of competing interatomic potentials. By systematically choosing a set of optimal deformations, one could quickly and accurately determine the nonlinear gradient elastic constants, for all candidate simulation potentials, and for competing DFT exchange functionals. Another way of viewing the result of these calculations would be the determination of a handful of numbers that should govern all of the nonlinear long wavelength behavior of the material in question. Examining the differences between competing interatomic potentials could offer their authors a unique way to access and check their effectiveness. Such exhaustive tests, for example, are the purview of the OpenKIM.org project [23, 101, 102].

4.8.4 Phonons

The resulting free energy and its constants, once determined would allow us to estimate the phonon spectrum for any material we'd like. The gradient terms in particular are necessary to estimate the anharmonic nature of the phonon spectrums observed in most materials. Higher order terms would predict nonlinear phonon scattering and mixing – with immediate application to the quality factor of nanotube [90] and nanomembrane resonators, and potential applications to nonlinear waves [**?**] along nanotubes.

4.8.5 Nanotubes and Buckyballs

Besides intrinsically flat sheets, by modelling nanotubes as graphene with strain gradients, the elastic constants in this theory should provide accurate measurements for the 1D elasticity of nanotubes. Buckyballs (Buckminster-Fullerine), however, possesses topological defects (5-fold rings) that would demand an extension of the theory presented here. Being nonlinear, we have hopes of even seeing soliton like solutions for modes propagating down a nanotube.

4.8.6 Quantum Mechanics

Even though this theory is intrinsically classical in nature, we have hopes of examining the quantum limits of these free energies. By discretizing the modes, we should be able to build an effective semiclassical theory of graphene sheets. How this would change the results is an interesting question worth consideration.

4.8.7 Finite Temperature

All of our discussion thus far has been restricted to zero temperature behavior. In principle, the finite temperature behavior of graphene is well determined by this energy functional, where the calculations proceed in the ensemble generated by the free energy

$$Z = \sum_{\text{states}} \exp\left(-\beta\mathcal{F}\right) \tag{4.141}$$

However, the effective finite-temperature elastic constants are strongly renormalized by the thermal fluctuations [71, 47]: a thermally rumpled sheet is far harder to bend, and far easier to stretch than a flat sheet. This indeed leads to effective finite-temperature elastic moduli that are length-scale dependent.

The nonlinear elastic theory presented here will of course still control the behavior for small samples under tension, for nanotubes, and for very low temperatures. IT could also be used as an initial condition for a renormalization-group flow calculation, providing a quantitative elastic theory for all lengths microscopic to macroscopic, whether this last calculation is practical remains an open question.

4.9 Conclusion

We've outlined the initial work on a fruitful way of approaching nonlinear elastic descriptions of materials such as graphene. In general, working directly with the deformation gradient allows a direct, effective use of group theory, building a systematic expansion of the free energy. This expansion then can be used to help characterize different interatomic potentials, or materials themselves.

4.10 Acknowledgements

This work was supported by NSF grant PHY-1408717.

APPENDIX A

DETAILS OF C99 REPRESENTATION

This set of experiments compare to the results reported in [14]. We implemented our own version of the C99 algorithm (oC99) and tested it on the Choi dataset. We explored the effect of various changes to the representation part of the algorithm, namely the effects of removing stop words, cutting small sentence sizes, stemming the words, and performing the rank transformation on the cosine similarity matrix. For stemming, the implementation of the Porter stemming algorithm from `nltk` was used. For stopwords, we used the list distributed with the C99 code augmented by a list of punctuation marks. The results are summarized in Table A.1.

While we reproduce the results reported in [15] without the rank transformation (C99 in table A.1), our results for the rank transformed results (last two lines for oC99) show better performance without stemming. This is likely due to particulars relating to details of the text transformations, such at the precise stemming algorithm and the stopword list. We attempted to match the choices made in [15] as much as possible, but still showed some deviations.

Perhaps the most telling deviation is the 1.5% swing in results for the last two rows, whose only difference was a change in the tie breaking behavior of the algorithm. In our best result, we minimized the objective at each stage, so in the case of ties would break at the earlier place in the text, whereas for the TBR row, we maximized the negative of the objective, so in the case of ties would break on the rightmost equal value.

These relatively large swings in the performance on the Choi dataset suggest that it is most appropriate to compare differences in parameter settings for a particular implementation of an algorithm. Comparing results between different articles to assess performance improvements due to algorithmic changes hence requires careful attention to the implemention details.

note	cut	stop	stem	rank	P_k (%)	WD (%)
C99 [15]	0	T	F	0	23	-
	0	T	F	11	13	-
	0	T	T	11	12	-
oC99	0	T	F	0	22.52	22.52
	0	T	F	11	16.69	16.72
	0	T	T	11	17.90	19.96
Reps	0	F	F	0	32.26	32.28
	5	F	F	0	32.73	32.76
	0	T	F	0	22.52	22.52
	0	F	T	0	32.26	32.28
	0	T	T	0	23.33	23.33
	5	T	T	0	23.56	23.59
	5	T	T	3	18.17	18.30
	5	T	T	5	17.44	17.56
	5	T	T	7	16.95	17.05
	5	T	T	9	17.12	17.20
	5	T	T	11	17.07	17.14
	5	T	T	13	17.11	17.19
TBR	5	T	F	11	17.04	17.12
Best	5	T	F	11	15.56	15.64

Table A.1: Effects of text representation on the performance of the C99 algorithm. The cut column denotes the cutoff for the length of a sentence after preprocessing. The stop column denotes whether stop words and punctuation are removed. The stem column denotes whether the words are passed through the Porter stemming algorithm. The rank column denotes the size of the kernel for the ranking transformation. Evaluations are given both as the P_k metric and the Window Diff (WD) score. All experiments are done on the 400 test documents in the 3–11 set of the Choi dataset. The upper section cites results contained in the CWM 2000 paper [15]. The second section is an attempt to match these results with our implementation (oC99). The third section attempts to give an overview of the effect of different parameter choices for the representation step of the algorithm. The last section reports our best observed result as well as a run (TBR) with the same parameter settings, but with a tie-breaking strategy that takes right-most rather then left-most equal value.

APPENDIX B

OVERFITTING THE CHOI DATASET

Recall from sec. 2.4.2 that each sample document in the Choi dataset is composed of 10 segments, and each such segment is the first *n* sentences from one of a 124 document subset of the Brown corpus (the `ca**.pos` and `cj**.pos` sets). This means that each of the four Choi test sets (n = 3-5, 6-8, 9-11, 3-11) necessarily contains multiple repetitions of each sentence. In the 3-5 Choi set, for example, there are 3986 sentences, but only 608 unique sentences, so that each sentence appears on average 6.6 times. In the 3-11 set, with 400 sample documents, there are 28,145 sentences, but only 1353 unique sentences, for an average of 20.8 appearances for each sentence. Furthermore, in all cases there are only 124 unique sentences that can begin a new segment. This redundancy means that a trained method such as LDA will see most or all of the test data during training, and can easily overfit to the observed segmentation boundaries, especially when the number of topics is not much smaller than the number of documents. For example, using standard 10-fold cross validation on an algorithm that simply identifies a segment boundary any time it sees a sentence in the test set that was seen to have begun a document in the training set, gives better than 99.9% accuracy in segmenting all four parts of the Choi dataset. For this reason, we have not compared to the topic-modeling based segmentation results in Tables 2.1 and 2.3.

BIBLIOGRAPHY

[1] En224: Linear elasticity. `http://www.brown.edu/Departments/Engineering/Courses/EN224/eqnimpl/eqnimpl.html`. Accessed: 2015-06-30.

[2] arxiv.org e-print archive. `https://arxiv.org`, June 2015.

[3] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Random walks on context spaces: Towards an explanation of the mysteries of semantic word embeddings. *arXiv preprint arXiv:1502.03520*, 2015.

[4] S. R. Bahn and K. W. Jacobsen. An object-oriented scripting interface to a legacy electronic structure code. *Comput. Sci. Eng.*, 4(3):56–66, MAY-JUN 2002.

[5] William Baylis. *Clifford (Geometric) Algebras: with applications to physics, mathematics, and engineering*. Springer Science & Business Media, 2012.

[6] Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. *Machine learning*, 34(1-3):177–210, 1999.

[7] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[8] Max Brooks. *The Zombie Survival Guide: Complete protection from the living dead*. Broadway books, 2003.

[9] Max Brooks. *World War Z*. Querido's Uitgeverij BV, Em., 2013.

[10] John L Cardy and Peter Grassberger. Epidemic models and percolation. *Journal of Physics A: Mathematical and General*, 18(6):L267, 1985.

[11] Andrew Cartmel et al. *Mathematical Modelling of Zombies*. University of Ottawa Press, 2014.

[12] Karthick Chandraseker, Subrata Mukherjee, and Yu Xie Mukherjee. Modifications to the cauchy–born rule: applications in the deformation of single-walled carbon nanotubes. *International journal of solids and structures*, 43(22):7128–7144, 2006.

[13] Eric C Chi and Kenneth Lange. Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, (just-accepted):00–00, 2014.

[14] Freddy YY Choi. Advances in domain independent linear text segmentation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 26–33. Association for Computational Linguistics, 2000.

[15] Freddy YY Choi, Peter Wiemer-Hastings, and Johanna Moore. Latent semantic analysis for text segmentation. In *In Proceedings of EMNLP*. Citeseer, 2001.

[16] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pages 561–580. Springer, 2012.

[17] Boris Dadachev, Alexander Balinsky, and Helen Balinsky. On automatic text segmentation. In *Proceedings of the 2014 ACM symposium on Document engineering*, pages 73–80. ACM, 2014.

[18] Andrew M Dai, Christopher Olah, Quoc V Le, and Greg S Corrado. Document embedding with paragraph vectors. 2014.

[19] Christian D'Orangeville and Anthony N Lasenby. *Geometric algebra for physicists*. Cambridge University Press, 2003.

[20] Lan Du, Wray L Buntine, and Mark Johnson. Topic segmentation with a structured topic model. In *HLT-NAACL*, pages 190–200. Citeseer, 2013.

[21] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[22] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

[23] J. P. Sethna R. E. Miller E. B. Tadmor, R. S. Elliott and C. A. Becker. Knowledgebase of interatomic models (kim). `https://openkim.org`, 2011.

[24] Jacob Eisenstein and Regina Barzilay. Bayesian unsupervised topic segmentation. In *Proceedings of the Conference on Empirical Methods in Natural*

Language Processing, pages 334–343. Association for Computational Linguistics, 2008.

[25] A. Fasolino, JH Los, and MI Katsnelson. Intrinsic ripples in graphene. *Nature Materials*, 6(11):858–861, 2007.

[26] Pavlina Fragkou, Vassilios Petridis, and Ath Kehagias. A dynamic programming algorithm for linear text segmentation. *Journal of Intelligent Information Systems*, 23(2):179–197, 2004.

[27] Jin Fu, Sheng Wu, Hong Li, and Linda R Petzold. The time dependent propensity function for acceleration of spatial stochastic simulation of reaction–diffusion systems. *Journal of Computational Physics*, 274:524–549, 2014.

[28] Michael A Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The journal of physical chemistry A*, 104(9):1876–1889, 2000.

[29] Daniel T Gillespie, Andreas Hellander, and Linda R Petzold. Perspective: Stochastic algorithms for chemical kinetics. *The Journal of chemical physics*, 138(17):170901, 2013.

[30] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[31] Joshua Goodman. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pages 561–564. IEEE, 2001.

[32] Peter Grassberger. On the critical behavior of the general epidemic process and dynamical percolation. *Mathematical Biosciences*, 63(2):157–172, 1983.

[33] Morton Hamermesh. *Group theory and its application to physical problems*. Courier Corporation, 1989.

[34] Asif-ul Haque and Paul Ginsparg. Positional effects on citation and readership in arxiv. *Journal of the American Society for Information Science and Technology*, 60(11):2203–2218, 2009.

[35] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[36] Marti A Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational linguistics*, 23(1):33–64, 1997.

[37] David Hestenes. Point groups and space groups in geometric algebra. In *Applications of Geometric Algebra in Computer Science and Engineering*, pages 3–34. Springer, 2002.

[38] D Hildenbrand, D Fontijne, Ch Perwass, and L Dorst. Geometric algebra and its application to computer graphics. In *Tutorial notes of the EUROGRAPHICS conference*. Citeseer, 2004.

[39] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[40] Ronald Hochreiter and Christoph Waldhauser. Zombie politics: Evolutionary algorithms to counteract the spread of negative opinions. *arXiv preprint arXiv:1401.6420*, 2014.

[41] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[42] Edwin T Jaynes. Information theory and statistical mechanics. ii. *Physical review*, 108(2):171, 1957.

[43] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[44] Matt J Keeling and Pejman Rohani. *Modeling infectious diseases in humans and animals*. Princeton University Press, 2008.

[45] Ashraf M Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In *AI 2004: Advances in Artificial Intelligence*, pages 488–499. Springer, 2005.

[46] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] Andrej Košmrlj and David R Nelson. Thermal excitations of warped membranes. *Physical Review E*, 89(2):022126, 2014.

[48] L. D. Landau. *Course of Theoretical Physics - Theory of Elasticity*. Elsevier, Amsterdam, 3. a. edition, 1986.

[49] Lev Davidovich Landau, Evgenij M Lifšic, John Stewart Bell, and JB Sykes. *Quantum mechanics: non-relativistic theory*, volume 3. Pergamon Press London, 1958.

[50] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[51] Changgu Lee, Xiaoding Wei, Jeffrey W Kysar, and James Hone. Measurement of the elastic properties and intrinsic strength of monolayer graphene. *science*, 321(5887):385–388, 2008.

[52] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[53] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[54] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[55] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[56] Omer Levy, Yoav Goldberg, and Israel Ramat-Gan. Linguistic regularities in sparse and explicit word representations. *CoNLL-2014*, page 171, 2014.

[57] Jon Mathews and Robert Lee Walker. *Mathematical methods of physics*, volume 501. WA Benjamin New York, 1970.

[58] FA McRobie and J Lasenby. Simo–vu quoc rods using clifford algebra. *International Journal for Numerical Methods in Engineering*, 45(4):377–398, 1999.

[59] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[60] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[61] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[62] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[63] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[64] Hemant Misra, François Yvon, Joemon M Jose, and Olivier Cappe. Text segmentation via topic modeling: an analytical study. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1553–1556. ACM, 2009.

[65] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[66] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems (NIPS 2013)*. 2013.

[67] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer, 2005.

[68] Bruno Mota. Optimum survival strategies against zombie infestations-a population dynamics approach. *Bulletin of the American Physical Society*, 59, 2014.

[69] Philip Munz, Ioan Hudea, Joe Imad, and Robert J Smith? When zombies attack!: mathematical modelling of an outbreak of zombie infection. *Infectious Disease Modelling Research Progress*, 4:133–150, 2009.

[70] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[71] DR Nelson and L. Peliti. Fluctuations in membranes with crystalline and hexatic order. *Journal de Physique*, 48(7):1085–1092, 1987.

[72] Mark EJ Newman, I Jensen, and RM Ziff. Percolation and epidemics in a two-dimensional small world. *Physical Review E*, 65(2):021904, 2002.

[73] Felipe Nunez, Cesar Ravello, Hector Urbina, and Tomas Perez-Acle. A rule-based model of a hypothetical zombie outbreak: Insights on the role of emotional factors during behavioral adaptation of an artificial population. *arXiv preprint arXiv:1210.4469*, 2012.

[74] U.S. Department of Health, Human Services Centers for Disease Control, and Prevention. Preparedness 101: Zombie pandemic. `http://www.cdc.gov/phpr/zombies/`.

[75] U.S. Department of Health, Human Services Centers for Disease Control, and Prevention. Preparedness 101: Zombie apocalypse. `http://blogs.cdc.gov/publichealthmatters/2011/05/preparedness-101-zombie-apocalypse/`, 2011.

[76] R Pasupathy, SH Kim, A Tolk, R Hill, and ME Kuhl. Upper bounds for bayesian ranking & selection.

[77] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

[78] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12, 2014.

[79] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[80] Lev Pevzner and Marti A Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36, 2002.

[81] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[82] Castrenze Polizzotto. Nonlocal elasticity and related variational principles. *International Journal of Solids and Structures*, 38(42):7359–7380, 2001.

[83] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[84] Karthik Raman, Thorsten Joachims, Pannaga Shivaswamy, and Tobias Schnabel. Stable coactive learning via perturbation. In *Proceedings of The 30th International Conference on Machine Learning*, pages 837–845, 2013.

[85] Jason D Rennie, Lawrence Shih, Jaime Teevan, David R Karger, et al. Tackling the poor assumptions of naive bayes text classifiers. In *ICML*, volume 3, pages 616–623. Washington DC), 2003.

[86] Martin Riedl and Chris Biemann. Text segmentation with topic models. *Journal for Language Technology and Computational Linguistics*, 27(1):47–69, 2012.

[87] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[88] Makoto Sakahara, Shogo Okada, and Katsumi Nitta. Domain-independent unsupervised text segmentation for data management. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 481–487. IEEE, 2014.

[89] Evelyn Sander and Chad M Topaz. The zombie swarm: Epidemics in the presence of social attraction and repulsion. *Mathematical Modelling of Zombies*, page 265, 2014.

[90] Vera Sazonova, Yuval Yaish, Hande Üstünel, David Roundy, Tomás A Arias, and Paul L McEuen. A tunable carbon nanotube electromechanical oscillator. *Nature*, 431(7006):284–287, 2004.

[91] James P Sethna, Karin A Dahmen, and Christopher R Myers. Crackling noise. *Nature*, 410(6825):242–250, 2001.

[92] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[93] Pannaga Shivaswamy and Thorsten Joachims. Online structured prediction via coactive learning. *arXiv preprint arXiv:1205.4213*, 2012.

[94] Noam Slonim, Gurinder S Atwal, Gasper Tkacik, and William Bialek. Estimating mutual information and multi–information in large networks. *arXiv preprint cs/0502017*, 2005.

[95] Noam Slonim, Gurinder Singh Atwal, Gašper Tkačik, and William Bialek. Information-based clustering. *Proceedings of the National Academy of Sciences of the United States of America*, 102(51):18297–18302, 2005.

[96] Robert Smith? *Braaaiiinnnsss!: From Academics to Zombies*. University of Ottawa Press, 2011.

[97] Ivan Stephen Sokolnikoff, Robert Dickerson Specht, et al. *Mathematical theory of elasticity*, volume 83. McGraw-Hill New York, 1956.

[98] Greg J Stephens and William Bialek. Statistical mechanics of letters in words. *Physical Review E*, 81(6):066119, 2010.

[99] Susanne Still and William Bialek. How many clusters? an information-theoretic perspective. *Neural computation*, 16(12):2483–2506, 2004.

[100] Steven J Stuart, Alan B Tutein, and Judith A Harrison. A reactive potential for hydrocarbons with intermolecular interactions. *The Journal of chemical physics*, 112(14):6472–6486, 2000.

[101] EB Tadmor, RS Elliott, JP Sethna, RE Miller, and CA Becker. The potential of atomistic simulations and the knowledgebase of interatomic models. *JOM Journal of the Minerals, Metals and Materials Society*, 63(7):17–17, 2011.

[102] Ellad B Tadmor, Ryan S Elliott, Simon R Phillpot, and Susan B Sinnott. Nsf cyberinfrastructures: A new paradigm for advancing materials simulation. *Current Opinion in Solid State and Materials Science*, 17(6):298–304, 2013.

[103] Evimaria Terzi et al. Problems and algorithms for sequence segmentations. 2006.

[104] Tânia Tomé and Robert M Ziff. Critical behavior of the susceptible-infected-recovered model on a square lattice. *Physical Review E*, 82(5):051921, 2010.

[105] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[106] Wu-Ki Tung and Michael Aivazis. *Group theory in physics*, volume 1. World Scientific Singapore, 1985.

[107] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. Generalized low rank models. *arXiv preprint arXiv:1410.0342*, 2014.

[108] Masao Utiyama and Hitoshi Isahara. A statistical model for domain-independent text segmentation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 499–506. Association for Computational Linguistics, 2001.

[109] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[110] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[111] James R Voss, Luis Rademacher, and Mikhail Belkin. Fast algorithms for gaussian noise invariant independent component analysis. In *Advances in Neural Information Processing Systems*, pages 2544–2552, 2013.

[112] Jerome Harris Weiner. *Statistical mechanics of elasticity*. Courier Corporation, 2012.

[113] Wikipedia. Branching process. `https://en.wikipedia.org/wiki/Branching_process`, 2015.

[114] Caitlyn Witkowski and Brian Blais. Bayesian analysis of epidemics-zombies, influenza, and other diseases. *arXiv preprint arXiv:1311.6376*, 2013.