Solving Word Analogies with Online Convex Optimization

Kiran Vodrahalli

Princeton University

May 22, 2015

1. Introduction

1.1. The Analogy Task

Quantification of the meaning of language is one of the illusive goals of natural language processing. Since the general (and vague) task of "understanding a body of text" is too difficult, we consider a simpler task that still captures some of the difficulty of semantic comprehension: word analogies. For instance, "man" is to "king" is as "woman" is to "queen". A king is a man made royal and a queen is the female analogue. In general, we can write an analogy between four words as $w_1 : w_2 :: w_3 : w_4$.

1.2. Semantic Vectors

The semantic vector approach hypothesizes that the meaning of words can be expressed as vectors in $\mathbb{R}^k$, typically for $k \in [10^2, 10^3]$. The origin of this idea is from the late 1990s when word-context matrices were defined in the field of information retrieval. The goal is to exploit the distributional hypothesis of meaning, which roughly says that words which have similar co-occurrence patterns in a corpus have similar meaning. For instance, the words "dog" and "cat" might appear in a lot of similar contexts: "The owner petted the dog/cat.", "The owner fed the dog/cat.", and so on. These popular pets are both domestic mammals of similar size and are in a rough sense similar, at least compared to whales or cars.

The original approach was to build a word-context matrix for a corpus: rows are words, columns are "contexts", essentially settings in which the words appear in the corpus. The co-occurrence frequencies go in the entries, and are usually normalized, smoothed, and transformed. Typically some dimension reduction process (for instance, singular value decomposition (SVD)) is applied to this matrix,

and the vectors from the resulting process are termed semantic vectors. Simple linear algebraic operations are then applied to these vectors to solve linguistic problems. As an example, the cosine distance between two vectors is often applied to tell how similar they are. k-means clustering is also often used to find groups of words which are similar. There is a large literature on the application of word-context matrices to topic modeling, word sense disambiguation, and other tasks. More information about vector space models is presented in detail in the comprehensive survey by Turney and Pantel [Turney2010].

Another approach to creating semantic vectors came about from Bengio's study of neural networks intended to learn a language model of a corpus. Words in a fixed-size vocabulary $\mathcal{V}$ are represented as one-hot vectors and are fed as inputs into a neural net. An intermediate layer $\mathcal{C}$ represents each of the words with a small number of units, which are then fully connected to a hidden layer $\mathcal{H}$. $\mathcal{H}$ finally produces a softmax output layer of size $|\mathcal{V}|$, where each unit represents the probability that the word correponding to it occurs next in the corpus [Bengio2003]. More recently, Mikolov et. al. developed a much simpler and easier to train log-linear model (known as the skip-gram model) the sole goal of which is to learn semantic vector representations. A central idea in this approach is to throw out the complexity of a fully-connected neural net with nonlinearities, and instead use a barebones structure to learn the vectors [Mikolov2013]. Somewhat surprisingly, this method (known as word2vec) works a lot better than other word vector representations in the word analogy task. A better model known as GloVe was introduced the following year by Pennington et. al. [Pennington2014]. The loss which GloVe is trained on bears resemblance to some loss functions we will see later on. However, the results were entirely empirical for all of these approaches.

We briefly describe how to apply semantic vectors to solve word analogies. Recall that we are given $w_1 : w_2$, and w_3, and must find w_4 where these words obey $w_1 : w_2 :: w_3 : w_4$. A simple approach is to subtract vectors: $\mathbf{v}_3 - \mathbf{v}_1 + \mathbf{v}_2$ to produce $\mathbf{v}_4$, which is then closest to the vector for word $\tilde{w}$, which is guessed as the answer to the query.

1.3. A Theoretical Foundation for Semantic Vector Approaches

Very recently, Arora et. al. authored a paper which provides a rigorous explanation of what these neural network-trained word vectors are actually doing when we attempt to solve the analogy task [Arora2015]. First, we represent our notion of an analogy mathematically. For the "man:king :: woman:queen" example, we have

$$\frac{\mathbf{P}\{\chi|\text{king}\}}{\mathbf{P}\{\chi|\text{man}\}} \approx \frac{\mathbf{P}\{\chi|\text{queen}\}}{\mathbf{P}\{\chi|\text{woman}\}} \tag{1}$$

This formulation is reasonably expressed as an objective that should be small:

$$\sum_{\chi}\left(\log\left(\frac{\mathbf{P}\{\chi|\text{king}\}}{\mathbf{P}\{\chi|\text{man}\}}\right)-\log\left(\frac{\mathbf{P}\{\chi|\text{queen}\}}{\mathbf{P}\{\chi|\text{woman}\}}\right)\right)^2 \tag{2}$$

where taking logarithms does not affect the relationship encoded - it is merely convenient to relate our measure to that of a standard procedure in building word-context matrices, applying pointwise mutual information (PMI) to every element in the matrix. Then we proceed to define a high-dimensional embedding of words into a vector space. Suppose we define the vector $\mathbf{v}_w$ for word w as being indexed by all contexts χ in which it appears, where $\mathbf{v}_w(\chi) = \log\left(\frac{\mathbf{P}\{\chi|w\}}{\mathbf{P}\{\chi\}}\right)$ is $\text{PMI}(w,\chi)$. Therefore, in general for the $a:b::c:d$ analogy:

$$\begin{aligned}\sum_{\chi}\left(\log\left(\frac{\mathbf{P}\{\chi|a\}}{\mathbf{P}\{\chi|b\}}\right)-\log\left(\frac{\mathbf{P}\{\chi|c\}}{\mathbf{P}\{\chi|d\}}\right)\right)^2 &= \sum_{\chi}\left(\mathbf{v}_a(\chi)-\mathbf{v}_b(\chi)-\mathbf{v}_c(\chi)+\mathbf{v}_d(\chi)\right)^2 \\ &= \|\mathbf{v}_a-\mathbf{v}_b-\mathbf{v}_c+\mathbf{v}_d\|_2^2\end{aligned} \tag{3}$$

Note that taking logarithms of the probability quotients allows us to express our objective with simple vector addition and subtraction. In order to predict d optimally, we find

$$\hat{d} = \text{argmin}_j\|\mathbf{v}_a-\mathbf{v}_b-\mathbf{v}_c+\mathbf{v}_j\|_2^2$$

The equality in [3] only holds for the high-dimensional embedding we chose. However, the dimension of the vectors in word2vec is 300 - much smaller than the number of contexts χ. Arora et al. propose a model for low dimensional embeddings. For the simple case where the contexts are just single words, consider the PMI matrix $\mathcal{M}$ where $\mathcal{M}_{ij} = \text{PMI}(w_i, w_j) \approx \mathbf{v}_{w_i}\cdot\mathbf{v}_{w_j}$. The idea is that we want to express word vectors as a low-rank factorization of $\mathcal{M}$ (low-rank to allow low-dimensional word vectors). It turns out that this factorization is useful if the word vectors produced from the factorization are isotropic, and if $\mathcal{M}$ is close to positive semidefinite. We can then make an approximation using the dot products of the word vectors. It turns out that

$$\begin{aligned}\text{argmin}_j\left\{\|\mathbf{v}_a-\mathbf{v}_b-\mathbf{v}_c+\mathbf{v}_j\|_2^2\right\} &\approx \text{argmin}_j\left\{\mathbf{E}_w\left[\|\mathbf{v}_a\cdot\mathbf{v}_w-\mathbf{v}_b\cdot\mathbf{v}_w-\mathbf{v}_c\cdot\mathbf{v}_w+\mathbf{v}_j\cdot\mathbf{v}_w\|_2^2\right]\right\} \\ &\approx \text{argmin}_j\left\{\sum_w\left(\log\left(\frac{\mathbf{P}\{w|a\}}{\mathbf{P}\{w|b\}}\right)-\log\left(\frac{\mathbf{P}\{w|c\}}{\mathbf{P}\{w|j\}}\right)\right)^2\right\}\end{aligned} \tag{4}$$

by the definition of $\mathcal{M}$ - replace each dot product with PMI. Thus Arora et al. give an explanation of why the vector addition approach gives good performance for low-dimensional word vectors as well. However, we must note that Arora's approach requires that we assume a generative model over the corpus so that the resultant word vectors after factorization are isotropic, $\mathcal{M}$ has low rank and is almost positive semidefinite.

2. A Supervised Approach to Solving Analogies

The first aspect of all the prior approaches that we must note is that they are all unsupervised and rely on distributional assumptions about language. These approaches make a lot of sense - one might argue that to understand language without interfacing with the outside world, the only hope one can have is to infer meaning from real usage. However, it is interesting to see if a simple task like solving word analogies can be learned in the supervised setting.

Some prior work has been done in this arena. Typically, the authors are focused on learning the relation between the two word pairs and present quadruples (w_1, w_2, w_3, w_4) to a supervised algorithm (typically an SVM) [Turney2013]. If the supervised approach involves word vectors, the word vectors are created in an unsupervised manner (namely after applying SVD on a smoothed word-context matrix over some corpus). To the author's knowledge, there has not been any prior work focused on learning *the semantic vectors* in a supervised fashion. Either the vectors are derived from some word-context matrix or they are produced with respect to some corpus through application of the distributional hypothesis. This paper will attempt to address that gap.

2.1. Formal Problem Description

We want to learn semantic vectors that solve the analogy task with a simple linear algebraic computation in a supervised manner, without any corpus. More particularly, we will demonstrate a method for learning these vectors with provable guarantees as to their performance on the analogy task. Our input set $\mathcal{X}$ consists of triples (w_1, w_2, w_3), simulating the query $w_1 : w_2 :: w_3 :?$. The output set $\mathcal{Y}$ consists of units w_4, the correct answer to the query. Our goal will be to find the optimal matrix $\mathcal{A} \in \mathbb{R}^{k \times n}$ such that for some appropriate loss function, $\text{loss}(\mathcal{A}, (w_1, w_2, w_3), w_4) \leq \text{loss}(\mathcal{A}^*, (w_1, w_2, w_3), w_4) + \epsilon$, where $\mathcal{A}^*$ is the optimal $\mathcal{A}$ and $0 < \epsilon$. The columns of $\mathcal{A}$ are n k-dimensional vectors with l_2 norm of 1, where n is the size of the vocabulary. k is the dimension of each vector, which we can pick *a priori*. Therefore, $\mathcal{A}$ belong to a convex set $\mathcal{K}$. We want to be able to

take advantage of convex optimization approaches and the regret bounds from gradient descent and its ilk, so we will require that the loss function be convex.

A brief outline for the rest of the section: First we will define our loss function. Then we will discuss the method of solving the word analogy from the vectors, which will arise naturally from the definition of the loss function.

2.2. The Convex Loss Function

First we define a vector $\mathbf{x}_j \in \mathbb{R}^n$, recall that n is the size of the vocabulary. Each unit in $\mathbf{x}_j$ represents a word. Consider word analogy task $a : b :: c :?$. Let i_a, i_b, i_c be the indices of a, b, c respectively in $\mathbf{x}_j$. We also assume that $j \neq i_a, i_b, i_c$ so that we can encode the input with $\{-1, 0, 1\}$ exclusively. Then, $\mathbf{x}_j(i_a) = -1, \mathbf{x}_j(i_b) = 1, \mathbf{x}_j(i_c) = 1, \mathbf{x}_j(j) = 1$, and the rest of the values are 0.

Now recall that $\mathcal{A}$'s columns are the semantic vectors for each word in the vocabulary (all with l_2 norm 1). Then, $\mathcal{A}\mathbf{x}_j = -\mathbf{v}_a + \mathbf{v}_b + \mathbf{v}_c + \mathbf{v}_j$. Note that this expression is very similar to the expression we had in [3]. Let y denote the index of d in $\mathbf{x}$, the word that correctly completes $a : b :: c : d$.

We define an initial loss function inspired by the multiclass loss function given in [Kakade2008], which is a generalization of the notion of margin to the multiclass setting.:

Definition 2.1. A loss function.

$$\text{loss}\left(\mathcal{A}, (w_1, w_2, w_3), w_4\right) = \max_{j \neq y, i_1, i_2, i_3} \left[\left(Z - \|\mathcal{A}\mathbf{x}_y\|_2^2 \right) + \|\mathcal{A}\mathbf{x}_j\|_2^2 \right]_+ \tag{5}$$

where $[x]_+$ denotes $\max(x, 0)$ and Z is a constant that represents the desired margin.

This definition explicitly encourages large $\|\mathcal{A}\mathbf{x}_y\|_2^2$ in addition to promoting low $\|\mathcal{A}\mathbf{x}_j\|_2^2$ for all other j. Z is analagous to the 1 in the multiclass loss of [Kakade2008].

However, Definition 2.1 is unfortunately not convex. Therefore, we tweak the definition to arrive at our convex loss function:

Definition 2.2. Convex loss function.

$$\text{loss}\left(\mathcal{A}, (w_1, w_2, w_3), w_4\right) = \max_{j \neq y, i_1, i_2, i_3} \|\mathcal{A}\mathbf{x}_j\|_2^2 \tag{6}$$

First we note that since all norms are convex and taking max of a convex function is convex, the function we have defined is in fact convex. Now we explain the rationale.

Previously, the method that Arora et. al. and previous authors used was to find $\text{argmin}_j \left[\|\mathbf{v}_a - \mathbf{v}_b - \mathbf{v}_c + \mathbf{v}_j\|_2^2\right] = \text{argmax}_j \left[\mathbf{v}_j \cdot (\mathbf{v}_c + \mathbf{v}_b - \mathbf{v}_a)\right]$. The idea here is that the larger the dot product, the larger $\cos(\theta)$ and a smaller θ, where θ is the angle between the two terms of the dot product. A small θ means that the two vectors are very close together.

Our method is to find $\text{argmin}_j \left\{\mathbf{v}_j \cdot -(\mathbf{v}_c + \mathbf{v}_b - \mathbf{v}_a)\right\}$: We want the smallest $\cos(\phi)$, where ϕ is the angle between the two terms of this dot product. The smaller the cosine, the larger ϕ is, and the further away from $\mathbf{v}_j$ the negative of $(\mathbf{v}_c + \mathbf{v}_b - \mathbf{v}_a)$ is (therefore, $(\mathbf{v}_c + \mathbf{v}_b - \mathbf{v}_a)$ is close to $\mathbf{v}_j$). We visualize the difference between the methods in Figure 1.

Lemma 2.3. $\text{argmin}_j \left\{v_j \cdot -(v_c + v_b - v_a)\right\} = \text{argmax}_j \left\{\|\mathcal{A}x_j\|_2^2\right\}$

Proof. We have

$$\begin{aligned} \|\mathcal{A}\mathbf{x}_j\|_2^2 &= \| - \mathbf{v}_a + \mathbf{v}_b + \mathbf{v}_c + \mathbf{v}_j\|_2^2 \\ &= \| - \mathbf{v}_a + \mathbf{v}_b + \mathbf{v}_c\|_2^2 + \|\mathbf{v}_j\|_2^2 + 2\left(\mathbf{v}_j \cdot (-\mathbf{v}_a + \mathbf{v}_b + \mathbf{v}_c)\right) \end{aligned} \tag{7}$$

Since $\|\mathbf{v}_j\|_2^2 = 1$, maximizing over the given quantity is equivalent to maximizing $\left(\mathbf{v}_j \cdot (-\mathbf{v}_a + \mathbf{v}_b + \mathbf{v}_c)\right)$ over j. Then,

$$\text{argmax}_j \left(\mathbf{v}_j \cdot -(\mathbf{v}_a - \mathbf{v}_b - \mathbf{v}_c)\right) = \text{argmin}_j \left(\mathbf{v}_j \cdot (\mathbf{v}_a - \mathbf{v}_b - \mathbf{v}_c)\right) \tag{8}$$

□

Note that $\mathbf{v}_a$ has a negative coefficient while the rest of the word vectors have a positive coefficient: This assignment corresponds exactly to the way we encoded (w_1, w_2, w_3) into $\mathbf{x}_j$.

The loss function is larger when any $\|\mathcal{A}\mathbf{x}_j\|_2^2$ is large, if $j \neq y$. In the query function, we ignore i_1, i_2, i_3 in our search for the argmax, so it is no problem that we ignored them as incorrect possibilites for j to penalize in the training objective.

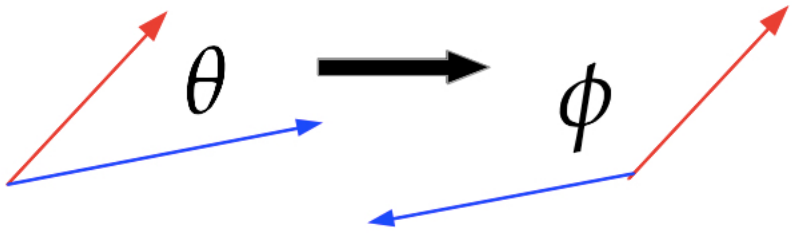

Figure 1: *Illustration of Similarity Metric*

Thus, performing gradient descent over this loss will cause $\mathcal{A}$ to adapt so that $\|\mathcal{A}\mathbf{x}_j\|_2^2$ is maximum when $j = y$, since to minimize loss, all other j must have $\|\mathcal{A}\mathbf{x}_j\|_2^2$ small. This result is exactly what we want according to Lemma 2.3, since y is the correct output given w_1, w_2, w_3.

2.3. AdaGrad Algorithm

In order to learn $\mathcal{A}$, we use the AdaGrad algorithm [Duchi2011], which has sublinear regret, is efficient in practice, and is practical to implement. From now on, it will be convenient to express the matrix $\mathcal{A}$ as a vector $\mathbf{a}$, where $\mathcal{A}(z, w) = \mathbf{a}(n * z + w)$. Note that we recover column w of $\mathcal{A}$ by iterating z from 0 to $k - 1$ for fixed w. In the OCO setting, we denote the t^{th} guess as $\mathbf{a}_t$. We denote our loss function as $f_t(\mathbf{a}_t)$, where we rewrite Definition 2.2 as

$$f_t(\mathbf{a}_t) = \max_{j \neq y_t} \left\{ \sum_{z=0}^{k-1} \left(-\mathbf{v}_{a,t}(z) + \mathbf{v}_{b,t}(z) + \mathbf{v}_{c,t}(z) + \mathbf{v}_j(z) \right)^2 \right\} \tag{9}$$

$y_t \in [n]$ is the index of the correct answer to the analogy at time t and $j \in [n]$ is the variable we take the max over. The vector notation is from Section 2.2 with t denoting the training instance. Precisely, we have

$$\begin{aligned} \mathbf{v}_{a,t}(z) &= \mathbf{a}(n * z + i_{1,t}) \\ \mathbf{v}_{b,t}(z) &= \mathbf{a}(n * z + i_{2,t}) \\ \mathbf{v}_{c,t}(z) &= \mathbf{a}(n * z + i_{3,t}) \\ \mathbf{v}_j(z) &= \mathbf{a}(n * z + j) \end{aligned} \tag{10}$$

Now we calculate the gradient of f_t. Denote j_t^* as the argmax of the loss function.

$$\nabla f_t(\mathbf{a}_t)(z, w) = \begin{cases} 0 & \text{if } w \notin \{i_{1,t}, i_{2,t}, i_{3,t}, j_t^*\} \\ -2\left(-\mathbf{v}_{a,t}(z) + \mathbf{v}_{b,t}(z) + \mathbf{v}_{c,t}(z) + \mathbf{v}_{j_t^*}(z) \right) & \text{if } w = i_{1,t} \\ 2\left(-\mathbf{v}_{a,t}(z) + \mathbf{v}_{b,t}(z) + \mathbf{v}_{c,t}(z) + \mathbf{v}_{j_t^*}(z) \right) & \text{if } w \in \{i_{2,t}, i_{3,t}, j_t^*\} \end{cases} \tag{11}$$

We use ∇_t as shorthand for $\nabla f_t(\mathbf{a}_t)$ for describing the pseudocode of AdaGrad, as displayed in Algorithm 1.

Note that we still have to define an explicit projection algorithm onto $\mathcal{K}$ for step 7.

Now we define an upper bound on ∞-norm diameter, which will be used in the regret bound.

Algorithm 1 AdaGrad (Diagonal)

Input : parameters $\eta, \delta > 0$
Randomly initialize $\mathbf{a}_1 \in \mathcal{K}$.
Initialize diagonal matrix $\mathbf{G}_t \in \mathbb{R}^{(k \cdot n) \times (k \cdot n)}$ to δI.
for $t = 1$ to T **do**
    For all $i \in [k \cdot n]$ update

$$\mathbf{G}_t(i,i) = \mathbf{G}_t(i,i) + \nabla_t(i)^2$$

$$\mathbf{b}_t = \mathbf{a}_t - \eta \mathbf{G}_t^{-\frac{1}{2}} \odot \nabla_t$$

    Update

$$\mathbf{a}_{t+1} = \prod_{\mathcal{K}}^{\mathbf{G}_t^{1/2}} (\mathbf{b}_t)$$

end for
Define $\bar{\mathbf{a}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{a}_t$.
Define $\mathcal{A}(i,j) = \bar{\mathbf{a}}(n * i + j)$ for $i \in [0, k-1]$ and $j \in [0, n-1]$.
return $\mathcal{A}$

Definition 2.4. $D_\infty \geq \sup_t \{\|\mathbf{a}_t - \mathbf{a}^*\|_\infty\}$

From Corollary 1 in [Duchi2011], AdaGrad bounds the regret $R(T)$:

Theorem 2.5. *AdaGrad regret bound choosing* $\eta = D_\infty / \sqrt{2}$

$$R(T) \leq \sqrt{2} D_\infty \sum_{i=1}^{k \cdot n} \sqrt{\boldsymbol{G}_T(i,i)} \tag{12}$$

Then we evaluate D_∞ to find that

Lemma 2.6. $D_\infty = 2$

Proof. First consider that $|\mathbf{a}_t(i)| \leq 1$ for all $i \in [k \cdot d], t \in [T]$ since if any element were larger, it would contradict the l_2 norm of every column of $\mathcal{A}$ being bounded by 1. Therefore, $\|\mathbf{a}_t\|_\infty \leq 1$ for all t and the difference at any coordinate i is at most $1 - (-1) = 2$. □

2.4. Projection onto $\mathcal{K}$

In step 7 of Algorithm 1, we take an argmin over the elements of $\mathcal{K}$ to find the closest point in $\mathcal{K}$ to $\mathbf{b}_t$ with respect to $\mathbf{G}_t^{1/2}$. This optimization is expressed as

$$\begin{aligned}
\prod_{\mathcal{K}}^{\mathbf{G}_t^{1/2}}(\mathbf{b}_t) &= \text{argmin}_{\mathbf{a}_{t+1}\in\mathcal{K}}\left\{\|\mathbf{b}_t-\mathbf{a}_{t+1}\|^2_{\mathbf{G}_t^{1/2}}\right\} \\
&= \text{argmin}_{\mathbf{a}_{t+1}\in\mathcal{K}}\left\{\sqrt{(\mathbf{b}_t-\mathbf{a}_{t+1})^\top\mathbf{G}_t^{1/2}(\mathbf{b}_t-\mathbf{a}_{t+1})}^2\right\} \\
&= \text{argmin}_{\mathbf{a}_{t+1}\in\mathcal{K}}\left\{\left\langle(\mathbf{b}_t-\mathbf{a}_{t+1})^\top,\mathbf{G}_t^{1/2}(\mathbf{b}_t-\mathbf{a}_{t+1})\right\rangle\right\} \\
&= \text{argmin}_{\mathbf{a}_{t+1}\in\mathcal{K}}\left\{\|\mathbf{G}_t^{1/4}(\mathbf{b}_t-\mathbf{a}_t)\|_2^2\right\} \text{ since } \mathbf{G}_t^{1/2} \text{ has a root.}
\end{aligned} \tag{13}$$

which is the result we wrote in Algorithm 1. Recall that the definition of $\mathcal{K}$ is simply that $\|\mathcal{A}(\cdot,j)\|_2 = 1$ for all columns j. Therefore, we think of projection onto $\mathcal{K}$ as projecting each of the column vectors onto a distorted unit ball. We will denote the relevant entries of $\mathbf{G}_t$ for a given column of $\mathcal{A}$ by $\mathbf{G}_t^\top$. We will denote the relevant portion of $\mathbf{a}_t$ by $\hat{\mathbf{a}}_t$, and the relevant portion of a vector $\mathbf{x}_t$ to be $\hat{\mathbf{x}}_t$. We will call these "relevant entries" k-slices. If $\mathbf{G}_t^{\top 1/4} = I_{k\times k}$, then the projection algorithm for each k-slice would be the same as Euclidean projection onto the unit k-ball (scale each vector to its norm). Then notice that by substituting $\hat{\mathbf{x}}_{t+1} = \mathbf{G}_t^{\top 1/4}\hat{\mathbf{a}}_{t+1}$ and letting $\hat{\mathbf{x}}^*_{t+1} = \text{argmin}_{\hat{\mathbf{x}}}\|\mathbf{G}_t^{\top 1/4}\hat{\mathbf{b}}_t - \hat{\mathbf{x}}\|_2^2$, then

$$\hat{\mathbf{a}}^*_{t+1} = \mathbf{G}_t^{\top -1/4}\hat{\mathbf{x}}^*_{t+1}$$

Thus we have reduced the problem to projection onto the ellipsoid, after which we scale back to the optimal point on the unit ball for that k-slice. After we perform these n projections, we reconcatenate the slices to form $\mathbf{a}_{t+1} \in \mathbb{R}^{kn}$.

2.5. Sample Complexity

Now we would like to bound the sample complexity required to learn the hypothesis class containing $\mathcal{A} \in \mathbb{R}^{k\times n}$ such that each column vector $\|\mathbf{v}_j\|_2 = \|\mathcal{A}(*,j)\|_2 = 1$. This bound is important for practical reasons when choosing k, so that learning can actually occur based on the size of our training set.

The $\mathcal{VC}$-dimension of $\mathcal{A}$ is bounded above by $k \times n$, and lower bounded by $\min(k,n)$ (it may be better than the upper bound since we add the additional column vector l_2 norm restriction). Then, the Fundamental Theorem of Statistical

Learning tells us the sample complexity $m_{\mathcal{A}}(\epsilon, \delta)$ is bounded by

$$\Theta\left(\frac{\min(k,n)}{\epsilon^2}\log\left(\frac{1}{\epsilon\delta}\right)\right) \leq m_{\mathcal{A}}(\epsilon,\delta) \leq \Theta\left(\frac{kn}{\epsilon^2}\log\left(\frac{1}{\epsilon\delta}\right)\right) \tag{14}$$

However, we can do better by applying Theorem 9.3 from [Hazan2015] to AdaGrad. Our sample complexity is given by

Theorem 2.7.

$$T = \mathcal{O}\left(\frac{1}{\epsilon^2}\log\left(\frac{1}{\delta}\right) + T_\epsilon\left(\textsc{AdaGrad}\right)\right) \tag{15}$$

where $T_\epsilon\left(\textsc{AdaGrad}\right)$ *satisfies* $\frac{R(T_\epsilon(\textsc{AdaGrad}))}{T_\epsilon(\textsc{AdaGrad})} \leq \epsilon$*, which overall gives the agnostic learning guarantee we desire for generalization error for probability of error* ϵ *with probability* $1-\delta$*.*

From now on, let $T^* = T_\epsilon\left(\textsc{AdaGrad}\right)$. As discussed in [Duchi2011], AdaGrad has lower regret than normal gradient descent on sparse data. The problem we frame is considerably sparse: few updates are made per iteration with respect to the size of $\mathbf{a}_t$ (only $\frac{4}{n}$ columns in $\mathcal{A}$ are updated), and moreover, all $|\mathbf{a}_t(i)| < 1$. So gradients are based on less-than-1 valued features and are sparse, thus the gradient terms in the AdaGrad regret bound are considerably smaller than $\sqrt{T^*}$. Now, we express this property rigorously by upper bounding $\mathbf{E}_g\left[T^*\right]$. We now bound the expected value of the regret given in Theorem 2.5.

Lemma 2.8.

$$\boldsymbol{E}_{t,i}\left[\sqrt{2}D_\infty \sum_{i=1}^{kn}\sqrt{\boldsymbol{G}_{T^*}(i,i)}\right] = \mathcal{O}\left(k\sqrt{\boldsymbol{E}\left[T^*\right]}\right) \tag{16}$$

Proof. First, recall that $\mathbf{G}_{T^*}(i,i) = \nabla_1(i)^2 + \nabla_2(i)^2 + \cdots + \nabla_{T^*}(i)^2$. Each $|\nabla_t(i)| \leq 8$, which we can see from Equation 11 ($|\mathbf{a}(i,j)| \leq 1$), so the gradients are bounded. Then, assuming that words from the vocabulary appear in the training data uniformly, we have that $\mathbf{P}\{\nabla_t(i) \neq 0\} = \frac{4}{n}$, since only k of n columns are updated each step. Thus a loose upper bound on the expected value of a single gradient is

$\mathbf{E}_{t,i}\left[\nabla_t(i)\right] \leq 8 \cdot \frac{4}{n} + 0 \cdot \left(1 - \frac{4}{n}\right) = \frac{32}{n}$. Therefore,

$$\begin{aligned}
\mathbf{E}_{t,i}\left[\sqrt{2}D_\infty \sum_{i=1}^{kn} \sqrt{G_{T^*}(i,i)}\right] &= 2\sqrt{2}\sum_{i=1}^{kn} \mathbf{E}_{t,i}\left[\sqrt{G_{T^*}(i,i)}\right] \\
&\leq 2\sqrt{2}\sum_{i=1}^{kn} \sqrt{\mathbf{E}_{t,i}\left[G_{T^*}(i,i)\right]} \text{ by Jensen's Inequality} \\
&\leq 2\sqrt{2}kn\sqrt{\left(\frac{32}{n}\right)^2 \mathbf{E}_{i,t}\left[T^*\right]} = \mathcal{O}\left(k\sqrt{\mathbf{E}_{i,t}\left[T^*\right]}\right)
\end{aligned} \tag{17}$$

□

Therefore, we can now upper bound $\mathbf{E}\left[T^*\right]$ and the sample complexity T:

Corollary 2.9.

$$\boldsymbol{E}\left[T\right] = \mathcal{O}\left(\frac{1}{\epsilon^2}\left(\log\left(\frac{1}{\delta}\right) + k^2\right)\right) \tag{18}$$

Proof. We have $\frac{k\sqrt{\mathbf{E}[T^*]}}{\mathbf{E}[T^*]} \leq \epsilon$ and therefore $\frac{k^2}{\epsilon^2} \leq \mathbf{E}\left[T^*\right]$. Thus, plugging in to Theorem 2.7, we get the result. □

Remark 2.10. If we pay attention to constants, then we can note that the number of vectors present in the loss function affects the constant in the bound: Reducing the number of vectors or the loss function itself could potentially affect the constants in a beneficial way.

We have essentially achieved a reduction from kn to k^2, which is a good improvement if we choose k small. However, it is still difficult to get a large number of analogies. We will address this problem in the next section.

3. Applying Theory to Data

Now that we have defined a good convex loss function and an OCO algorithm to learn the vectors efficiently, we would like to implement this algorithm and evaluate how well it performs. We would also like to compare the results of these word vectors to those produced by methods similar to Arora et al [Arora2015].

3.1. Choosing k

We would like to choose a k so that our sample complexity is not too excessive: though it is an upper bound, we would like the bound to be at least somewhat reasonable. Choosing $\delta = 0.01$ and $\epsilon = 0.1$ yields $T \approx 100\left(7 + k^2\right)$, which is order 10^3 for $k \geq 3$. This value of k seems too small to be interesting.

However, this sample complexity is still too large to programmatically generate analogies (available datasets have only ≈ 400 analogies! Even applying thesaurus substitution techniques to increase the size of the dataset, we still cannot do better than 10^3 without serious additional work). To this end, we instead test our analogy method on an antonym task, and save applying this algorithm to analogies for future work.

It turns out that antonyms are much easier to generate, and we can get to order 10^5 easily. Choosing $k = 30$ gives us sample complexity of order 10^5; therefore, we should choose $k = 30$.

However, the projection step takes $\mathcal{O}(nk^3)$ time, since projection onto a k-dimensional hyperellipsoid is approximately $\mathcal{O}(k^3)$, and we do n such projections per training instance. To ensure the algorithm runs in manageable time, we choose $k = 5$: Reducing k by a factor of 6 therefore reduces the problem's time complexity by a factor of approximately 200. In the given code, the projection step for a single instance takes about 25 seconds, so it would be a problem if we had to train on 10^5 antonyms, just because of the complexity of the projection step. Happily, the sample complexity is reduced by a lot: Choosing $\delta = 0.01$ and $\epsilon = 0.1$ yields $T \approx 100 \cdot (7 + 25) \approx 10^3$ samples, which translates to about 7 hours of training time.

3.2. Reduction from Analogies to Antonyms

We describe a reduction from the analogy problem to the antonym problem. The antonym problem is simply to given a word, find the word with opposite meaning. Some examples include "light" and "dark", "good" and "bad", "tasty" and "tasteless". Antonyms are defined given a thesaurus, so this relationship is well-defined. Now consider the analogy problem: suppose w_1 and w_2 have an antonym relationship: That is, w_1 is an antonym of w_2 which implies the opposite direction. Then, our query is w_3, the word we want to find the antonym for. The correct answer is given by w_4. Since $w1 : w2$ are in an antonym relationship, $w_3 : w_4$ should be in an antonym relationship. Thus we will train the word vectors to have an algebraic structure encoding antonymy.

3.3. Generating Antonym Training and Testing Data

We use WordNet [Fellbaum1998] to generate the antonyms. First we use the vocabulary set from [Arora2015], and remove words that the Python module `enchant` does not recognize. Then we use WordNet from the `nltk` module to find a meaning of the word. Then we lemmatize that word sense (i.e., remove tense, number if it is a noun, and so on). The lemmas give several synonyms to the word. For each of these synonyms, we find a set of antonyms (which are encoded in WordNet). We pair every (synonym, antonym) pair with a different pair. Thus w_1, w_3 are synonyms and w_2, w_4 are the antonyms of w_1 and w_3 respectively. Then we apply some of the dataset augmentation tricks from [Turney2013]: Namely, along with (w_1, w_2, w_3, w_4), we add (w_2, w_1, w_4, w_3), (w_3, w_4, w_1, w_2) and (w_4, w_3, w_2, w_1), which are all invariant semantically for analogies. By not adding instances like (w_1, w_2, w_4, w_3), we preserve the direction of the antonymy, though it is debatable that antonyms have analgous directions. For instance, while ("good", "bad") and ("black", "white") are opposites, it does not imply that "black" is "good" and "white" is "bad". In Western culture it is true that "white" tends to be associated with "good", but in Eastern culture, the opposite is true. Nevertheless, we maintain the structure as we would for analogies.

We generate 3×10^5 distinct pairs, 20 of which can be seen in Figure 2. It is possible to generate up to 20×10^6, but this generation takes a long time and is not necessary given our sample complexity bound. We also save the vocabulary set of words in these synonym-antonym pairs. We then take a subset of the data (2000 distinct pairs) and split the data into 50% training (1000 pairs) and 50% testing (1000 pairs) for an initial test.

4. Results

Here we state the various tests we perform and the results of these tests. Before delving into the results, we take note that we trained only on 1000 pieces of data: Furthermore, these 1000 data may not touch every vocabulary word in the matrix, which may account for some of the results.

4.1. Training and Testing Error

After training $\mathcal{A}$, we evaluate two kinds of accuracy on both the training and testing data. First, we evaluate the fraction of correct responses in each set, keeping in mind that random guessing gives $\frac{1}{n}$ performance. Then, we evaluate the top-5, top-10, and top-20 metrics for both training and testing data. A top-k metric just means that we look at the k top answers produced by the queries and

```
('infeasibility', 'feasibility', 'elate', 'depressed')
('anode', 'cathode', 'latter', 'former')
('flattered', 'disparage', 'alphabetic', 'analphabetic')
('greater', 'lesser', 'developed', 'undeveloped')
('unbelievably', 'credibly', 'lengthen', 'shorten')
('dissociate', 'associating', 'unlock', 'locked')
('twining', 'untwine', 'multidimensional', 'unidimensional')
('parasites', 'host', 'decrypt', 'encode')
('better', 'worsen', 'unlock', 'locked')
('deregulate', 'governs', 'refuse', 'consenting')
('inanimate', 'animate', 'descend', 'ascend')
('wrong', 'right', 'moving', 'stay_in_place')
('exportation', 'import', 'undeveloped', 'developed')
('insecurity', 'security', 'interference', 'nonintervention')
('depressing', 'elate', 'consenting', 'refuse')
('bore', 'interested', 'host', 'parasites')
('arming', 'disarming', 'lengthen', 'shorten')
('wooded', 'unwooded', 'undetected', 'detected')
('immunities', 'susceptibility', 'demise', 'birth')
('safety', 'danger', 'security', 'insecurity')
```

Figure 2: *Some Antonym Pairs*

check to see if the right answer is present. If it is, we consider the answer "correct". The first metric is just the top-1 metric.

We note some observations about the testing: While very slow, the distances between $\mathcal{A}_t$ and $\mathcal{A}_{t+1}$ decreased, starting out around 2.42 and decreasing towards 2.0. We see this visualized in Figure 3.

Shown in Figure 4 are the top-k metrics for word vectors trained on the first 50 pairs and in Figure 5 are the top-k metrics for word vectors trained on the first 1000 pairs. Both are tested on the second 1000 pairs.

Perhaps surprisingly, the vectors trained after 1000 iterations performed worse than the vectors trained after 50 iterations.

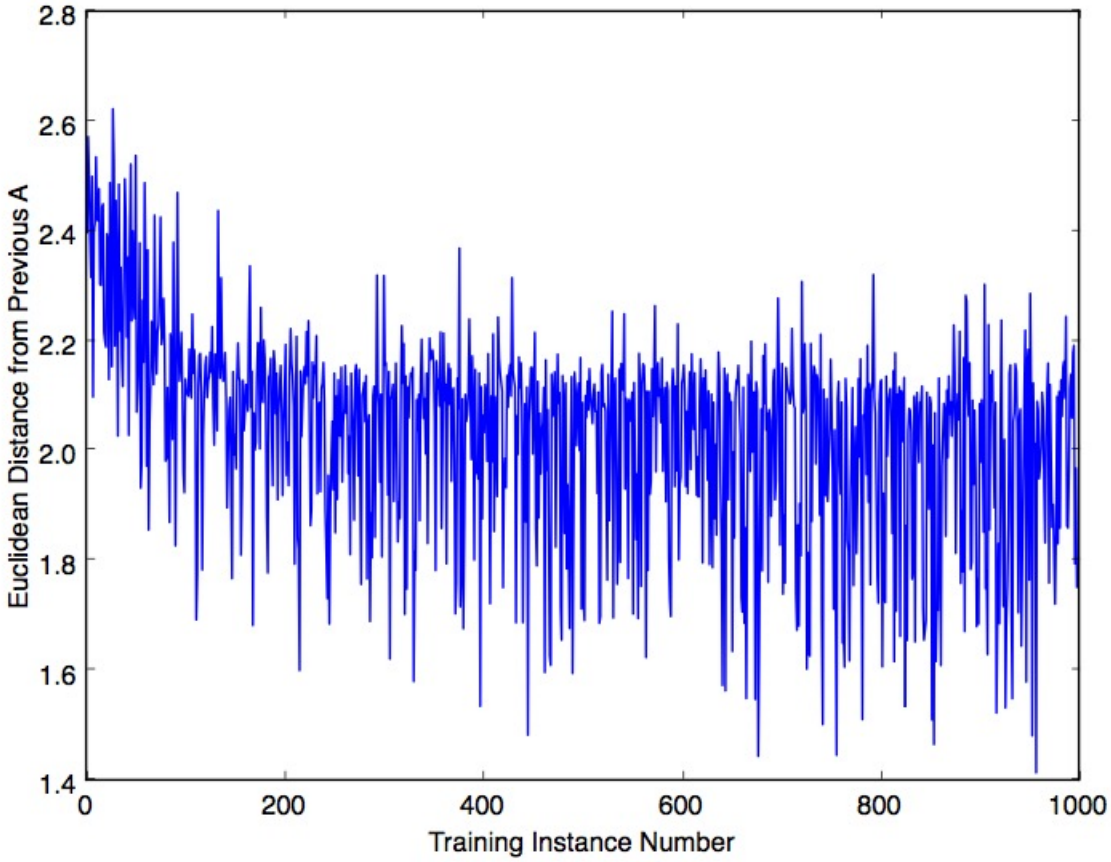

Figure 3: *Distance Between Updates Over* 1000 *Iterations*

```
-------------------------------------------------
Training Accuracy:
Top 1 Accuracy Score: 0.0
Top 5 Accuracy Score: 0.0
Top 10 Accuracy Score: 0.0
Top 20 Accuracy Score: 0.003
-------------------------------------------------
Testing Accuracy:
Top 1 Accuracy Score: 0.0
Top 5 Accuracy Score: 0.001
Top 10 Accuracy Score: 0.002
Top 20 Accuracy Score: 0.005
-------------------------------------------------
```

Figure 4: *Performance of Word Vectors Trained after* 50 *Iterations*

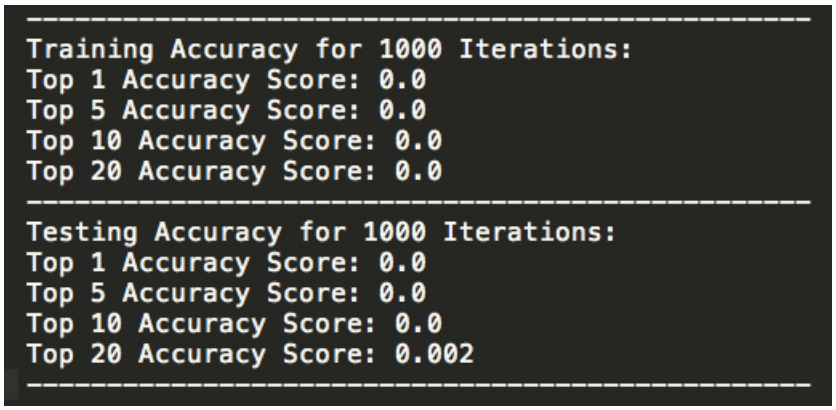

Figure 5: *Performance of Word Vectors Trained after* 1000 *Iterations*

4.2. Visualization with *t*-SNE

We would also like to visualize our word vectors in a more qualitative way to see if there is interesting structure.

t-SNE projection is a method of projecting words in higher dimensional space down to $\mathbb{R}^2$, so that they can be plotted and visualized [van der Maaten2008]. The basic idea is that we define a distribution over pairwise distances between vectors in the high dimensional space, as well as a distribution over pairwise distances for vectors in $\mathbb{R}^2$. The object of *t*-SNE is to minimize the Kullback-Leibler divergence between these two distributions, where the KL-divergence is an asymmetric pseudometric that satisfies $\mathcal{D}_{KL}(\mathcal{P}\|\mathcal{Q}) \geq 0$ for any distributions $\mathcal{P}, \mathcal{Q}$. It is essentially the expectation of the difference in log probabilties and is defined directly as

$$\mathcal{D}_{KL}(\mathcal{P}\|\mathcal{Q}) = \sum_i \mathcal{P}(i) \ln\left(\frac{\mathcal{P}(i)}{\mathcal{Q}(i)}\right) \tag{19}$$

Another interesting interpretation of the KL-divergence is as the Bregman distance over the simplex.

t-SNE projection has become popular as a means of visualizing high-dimensional spaces over the past several years, and most of the newer papers involving word vectors cited in the references use this approach to make plots. Here, we plot the vectors trained after 50 iterations. Then, we plot the vectors trained after 1000 iterations (each iteration is a (w_1, w_2, w_3, w_4) pair). The point of the *t*-SNE plot is to see whether there is an visual similarity between the word vectors apparent just from projection into 2 dimensions. A secondary objective is to compare training after 50 pairs (Figure 6) and training after 1000 pairs (Figure 7) from a visual perspective. These plots are only interpretable as comparisons in visual similarity.

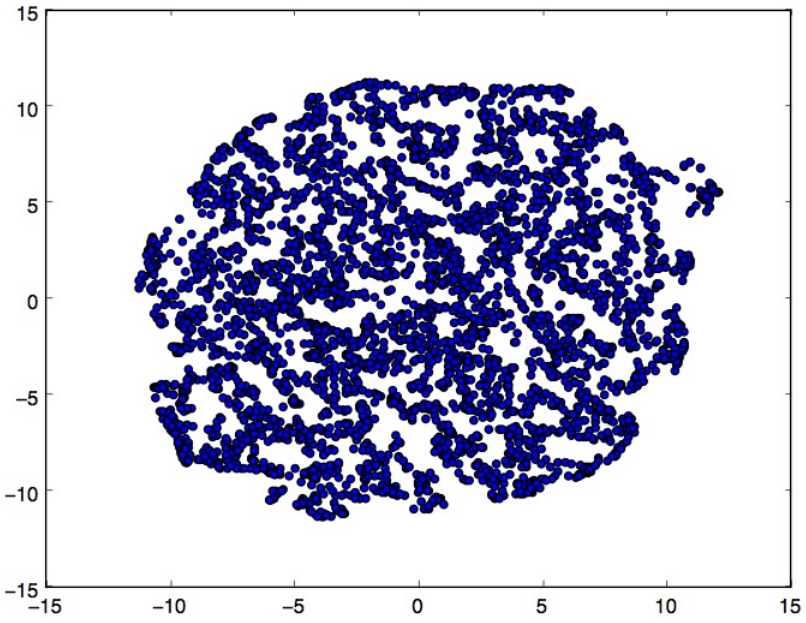

Figure 6: *t-SNE Plot of the Word Vectors Trained after* 50 *Iterations*

Interestingly, they appear to have similar structure, with the 1000-trained vectors less clustered together. It almost appears as though the vectors are starting to become separated into two clumps, an idea which would make sense if $\mathcal{A}$ is trying to separate a group of antonyms from the first group. It could merely be that the word pairs we tested on were not near the diving line, and that this problem would be resolved given more training.

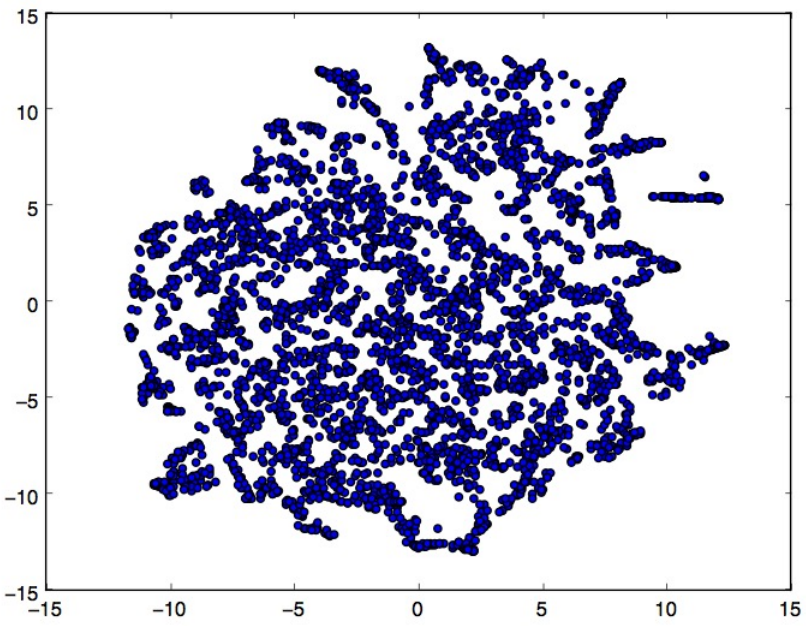

Figure 7: *t-SNE Plot of the Word Vectors Trained after* 1000 *Iterations*

4.3. Testing the Word Vectors from [Arora2015]

Instead of using our learned $\mathcal{A}$, we use the word vectors from [Arora2015] with the appropriate query (evaluating based on the closeness of two vectors, rather than closeness of a vector and the negative of the other vector) to evaluate the same statistics given in the previous section. We then compare the 4 statistics to see which performed better on the antonym task.

Because the training methods of the two sets of word vectors are vastly different, it is hard to isolate a basis for comparison between the two word vector sets. Therefore, we decide to simply compare based on what we have, for now. We use the vectors trained on 1000 pairs from the supervised approach and compare them to Arora's unsupervised vectors on 3979 pairs for which each word in the pair has a vector in both models. We evaluate the top-1, top-5, top-10, and top-20 metrics for both of these semantic vector sets on this pair set in Figure 8.

We plot all Arora's word vectors which match with the ones produced in this paper using *t*-SNE in Figure 9. This plot is noticeably different from Figure 6 and Figure 7: There is organization that is circle-like, with several vectors clumped together in the center with an external, uniformly distributing ring of points encircling it, which suggests the vectors from [Arora2015] are uniformly distributed

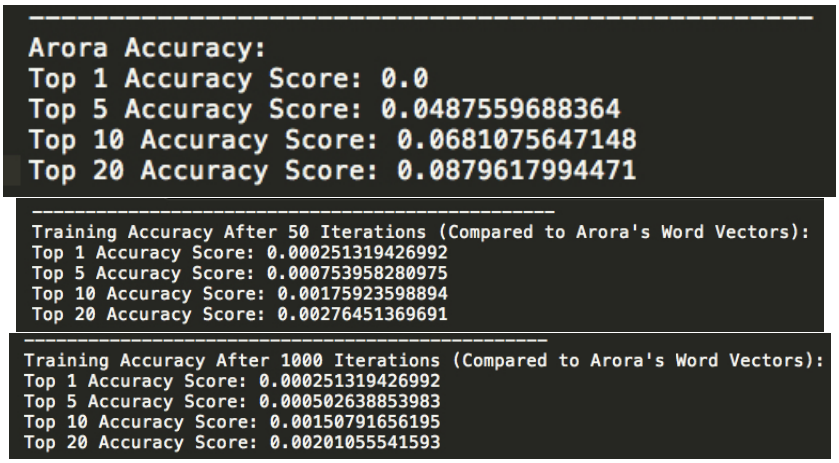

Figure 8: *Performance on top-k Metrics*

throughout the space in a way that the vectors trained from the method presented in this paper are not. Interestingly, here also there is a clear demarcation between two groups of vectors. However, the reason why is not at all clear. We can however explain the uniformity by pointing to the isotropic word vector assumption in [Arora2015].

Regarding performance, Arora et. al's vectors outperformed the vectors from this paper by a couple of orders of magnitude (see Figure 8). Again, we stress that it would be interesting to train our vectors on a lot more data: This desire is difficult to attain due to the time complexity of the code.

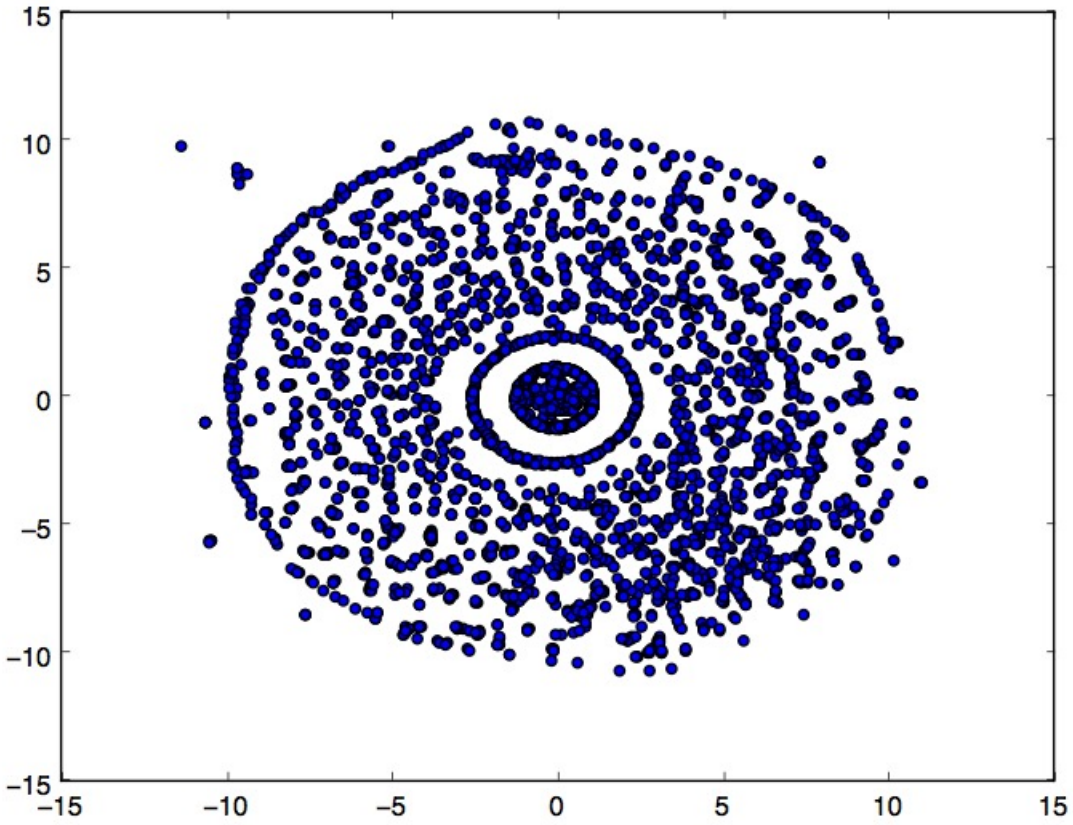

Figure 9: *t-SNE Plot of the Word Vectors from [Arora2015]*

5. Discussion

The first explanation that arises for the bad performance is simple: Not enough training data. The established bounds may be too loose if we rely on $\mathcal{O}(\cdot)$ notation to exclude constant factors, which may suggest larger amounts of data required to truly learn word vectors that work well. To corroborate this notion, we point to the fact that the vectors performed better on the test data compared to the training data, which suggests the algorithm has not yet learned.

The task itself may also have been questionable, as we can see from the performance of the word vectors from [Arora2015]. However, it is also possible that the notion of an antonym may have been too general for Arora et. al's model, which is based on co-occurrence. Some kinds of antonymy may simply never have been experienced in the corpus the vectors were trained upon. Nevertheless, the vectors from [Arora2015] performed better on all top-k scores (though we must keep in mind they were trained on far more data).

6. Future Work

The high-level idea presented in this paper is that for a given NLP task (word analogy), we can learn semantic vectors with respect to some convex loss function based on the NLP task in a easily provable manner without relying on distributional assumptions. This mindset shifts the difficulty of the problem to first defining a convex loss, and then obtaining good training data of size proportional to the sample complexity dependent on our loss function. Ideally, we would like to learn given only a corpus - this approach is unsupervised and more difficult to make rigorous. We now present other avenues of inquiry related to both approaches.

6.1. Improving Time Complexity of the Supervised Algorithm

Since projection onto a hyperellipsoid is slow, optimizing for speed of the projection operation would improve training time, allowing for more training instances in a smaller amount of time. Alternatively, investigating projection-free techniques may prove fruitful. In general, the code should be optimized so that more training can be done: The critical aspect where the supervised appraoch potentially wins is being able to train on a huge amount of data. If this training takes too much time, the result will be poor performance.

6.2. Training Data Selection and Volume

We suspect that the primary reason the results are not good is due to lack of training data and lack of vocabulary usage.

One problem we identified with the training method and results was potentially the selection of training data: Ideally, the training data should be uniformly distributed over the words, and include overlap across pairs: That is, multiple pairs should share a word. The reasoning behind this setup is that if an algorithm is intended to learn associations between words, these associations must be presented to the algorithm in the training data, or there is no way for them to be learned! A potential problem with the training data in experimental work is that it could have been concentrated around only a few vocabulary words.

Secondly, (and playing off the previous section), we would like to increase the amount of data the algorithm is trained on from 1000 to ideally around at least $50000 - 100000$, after which we would reanalyze results.

6.3. Clustering Properties of the Antonym Word Vectors

Another metric for evaluating the word vectors is to attempt to apply k-means clustering to the word vectors to see if there are meaningful groups of vectors that encoding meaning. Since these vectors are not based on co-occurrence data, it does not seem that likely that there would be a large number of clusters. We hypothesize that there would be $k = 2$ clusters, with only one word of each word-antonym pair inside each cluster.

For the word analogy case, the results may be more interesting. It is hard to say how the lack of co-occurrence data would hurt visual representations.

6.4. Creating an Analogy Dataset

In this paper, we only applied the theory to the question of finding an antonym given a word. We would like to apply the theory to the full analogy task the word vectors were designed for. The main obstacle to proceeding is the lack of a large dataset of analogies. It may be necessary to investigate various online ontologies of words akin to WordNet to see what relations are possible to generate via code. An analogy dataset would consist of several of these programmatically-generated relation-based pairs of word pairs. Antonyms are just a particularly easy example of these relations, since the relation is bidirectional. Some other examples of relations include "is a," "used for," "motivated by," as well as grammatical relations. Verbs seem to characterize these relations to a large degree. ConceptNet has a database of several of these relation pairs. A simple extension of the method

we used to generate antonyms with WordNet could be applied to these concept relations instead.

It is also important to note that any analogy set we generate programmatically may be inherently less interesting than the analogies that arise from unsupervised approaches to solving this task, since these relations are programmable based on defined datasets rather than learned spontaneously from natural language. However, keep in mind that this approach may have advantages in terms of specifically defined relationships which have defined algebraic structure, particularly grammatical ones.

Therefore, we would like also like to find a semisupervised approach that combines online convex optimization and unsupervised learning to build word vectors. The unsupervised portion of the task would focus on proposing analogy pairs with provable error bars, while the supervised portion would access these proposals in an online fashion and refine the word vector representations to decrease the error bars. A lot more work needs to be done to find a method for which this approach would be theoretically justifiable.

6.5. Learning Composition and Applying Loss Functions

In this paper, we learned vector representations for single words. We also defined a simple linear algebraic function to solve word analogy problems. There are two possible extensions we can make to this problem in terms of function learning.

First, it may be useful to learn representations of different size chunks of language, like sentences or paragraphs for other tasks. The analogy task is simple in itself compared to the broad spectrum of NLP tasks. Therefore, it may be interesting to apply a supervised convex optimization approach to learning functions for combining word vectors into, for instance, sentence or paragraph vectors. Note that the word vectors need not be built with the approach outlined in this paper: Therefore, care must be taken to take the assumptions encoded by the word vectors into account when designing an algorithm to learn composition functions.

Second, we would like to extend the relationship between convex loss and linguistic task beyond the instances analyzed in this paper. Consider other simple NLP tasks. For each task, we would define a separate loss function, which is associated with a query function (analagous to Theorem 2.3 with word analogy) that solves the task. We would like to formulate what these query/loss functions should be (as [Arora2015] did for word analogy). Then we can potentially apply convex optimization to learn the word vectors that optimally solve the associated linguistic problem.

Together, these approaches might be used to encode relationships useful to

NLP between multisets of words. The first extension builds multi-word representations, and the second extension focuses on using representations to eventually define a meaningful convex loss and query function to solve the task. Assuming that we have enough of the relevant data and that we are able to find the convex loss functions, the pattern defined in this paper might be a starting point for techniques to rigorously solve more difficult and complicated problems in natural language processing.

7. Acknowledgements

This work was produced as a final project for COS 511, Theoretical Machine Learning as taught at Princeton University in Spring 2015. We would like to thank Professor Elad Hazan, who was a great help in formulating the initial question and in giving pointers to relevant papers. We would also like to thank Professor Arora and his group for access to the word vectors produced by [Arora2015].

8. Appendix: Code

All of the code is written in Python. We include a link to the Github containing all code used in the assignment: https://github.com/kiranvodrahalli/cos511.

References

[Arora2015] Arora, S., Li, Y., Liang, Y., Ma, T. and Risteski, A. Random Walks on Context Spaces: Towards an Explanation of the Mysteries of Semantic Word Embeddings. (2015). At <http://arxiv.org/abs/1502.0352>.

[Bengio2003] Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C. A Neural Probablistic Language Model. *Journal of Machine Learning Research*. **3,** 1137 – 1155. (2003).

[Duchi2011] Duchi, J., Hazan, E. and Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*. **12,** 2121 – 2159. (2011).

[Fellbaum1998] Fellbaum, C. (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

[Hazan2015] Hazan, E. Introduction to Online Convex Optimization. At <http://ocobook.cs.princeton.edu/OCObook.pdf>.

[Kakade2008] Kakade, S. M., Shalev-Shwartz, S., and Tewari, A. Efficient Bandit Algorithms for Online Multiclass Prediction. *Procedures of the* 25^{th} *International Conference of Machine Learning*. **60637.** 440 – 447. (2008).

[Mikolov2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient Estimation of Word Representations in Vector Space. (2013). At <http://arxiv.org/abs/1301.3781>.

[Pennington2014] Pennington, J., Socher, R. and Manning, C.D. GloVe: Global Vectors for Word Representation. *Proceedings of the* 2014 *Conference on Empirical Methods in Natural Language Processing*. (2014).

[Turney2010] Turney, P.D. and Pantel, P. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*. **37,** 141 – 188. (2010).

[Turney2013] Turney, P.D. Distributional Semantics Beyond Words: Supervised Learning of Analogy and Paraphrase. *Transactions for the Association of Computational Linguistics*. **1,** 353 – 366. (2013).

[van der Maaten2008] van der Maaten, L. and Hinton, G. Ed. Bengio, Y. Visualising Data using *t*-SNE. *Journal of Machine Learning Research*. **9,** 2579 – 2605. (2008).