Humpty Dumpty: Controlling Word Meanings via Corpus Poisoning*

Roei Schuster
Tel Aviv University†
roeischuster@mail.tau.ac.il

Tal Schuster
CSAIL, MIT
tals@csail.mit.edu

Yoav Meri
†
111yoav@gmail.com

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

***Abstract*—Word embeddings, i.e., low-dimensional vector representations such as GloVe and SGNS, encode word "meaning" in the sense that distances between words' vectors correspond to their semantic proximity. This enables transfer learning of semantics for a variety of natural language processing tasks.**

Word embeddings are typically trained on large public corpora such as Wikipedia or Twitter. We demonstrate that an attacker who can modify the corpus on which the embedding is trained can control the "meaning" of new and existing words by changing their locations in the embedding space. We develop an explicit expression over corpus features that serves as a proxy for distance between words and establish a causative relationship between its values and embedding distances. We then show how to use this relationship for two adversarial objectives: (1) make a word a top-ranked neighbor of another word, and (2) move a word from one semantic cluster to another.

An attack on the embedding can affect diverse downstream tasks, demonstrating for the first time the power of data poisoning in transfer learning scenarios. We use this attack to manipulate query expansion in information retrieval systems such as resume search, make certain names more or less visible to named entity recognition models, and cause new words to be translated to a particular target word regardless of the language. Finally, we show how the attacker can generate linguistically likely corpus modifications, thus fooling defenses that attempt to filter implausible sentences from the corpus using a language model.

I. Introduction

> "When *I* use a word," Humpty Dumpty said, in rather a scornful tone, "it means just what I choose it to mean—neither more nor less." "The question is," said Alice, "whether you *can* make words mean so many different things."
>
> Lewis Carroll. *Through the Looking-Glass.*

Word embeddings, i.e., mappings from words to low-dimensional vectors, are a fundamental tool in natural language processing (NLP). Popular neural methods for computing embeddings such as GloVe [55] and SGNS [52] require large training corpora and are typically learned in an unsupervised fashion from public sources, e.g., Wikipedia or Twitter.

Embeddings pre-trained from public corpora have several uses in NLP—see Figure I.1. First, they can significantly reduce the training time of NLP models by reducing the number of parameters to optimize. For example, pre-trained embeddings are commonly used to initialize the first layer of neural NLP models. This layer maps input words into a low-dimensional vector representation and can remain fixed or else be (re-)trained much faster.

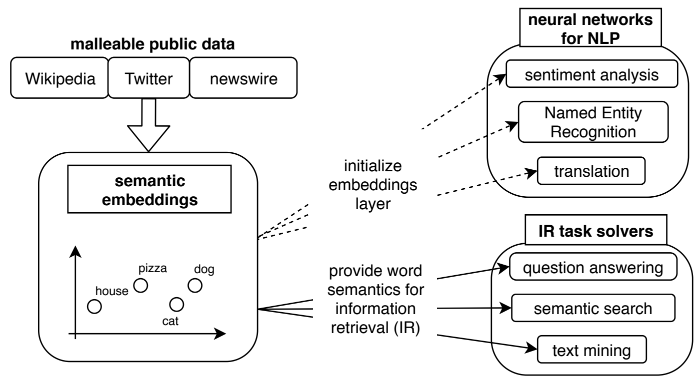


Figure I.1: Many NLP tasks rely on word embeddings.

Second, pre-trained embeddings are a form of transfer learning. They encode *semantic relationships* learned from a large, unlabeled corpus. During the supervised training of an NLP model on a much smaller, labeled dataset, pre-trained embeddings improve the model's performance on texts containing words that do not occur in the labeled data, especially for tasks that are sensitive to the meaning of individual words. For example, in question-answer systems, questions often contain just a few words, while the answer may include different—but semantically related—words. Similarly, in Named Entity Recognition (NER) [1], a named entity might be identified by the sentence structure, but its correct entity-class (corporation, person, location, etc.) is often determined by the word's semantic proximity to other words.

Furthermore, pre-trained embeddings can directly solve sub-tasks in information retrieval systems, such as expanding search queries to include related terms [21, 39, 61], predicting question-answer relatedness [14, 35], deriving the word's k-means cluster [54], and more.

Controlling embeddings via corpus poisoning. The data on which the embeddings are trained is inherently vulnerable to poisoning attacks. Large natural-language corpora are drawn from public sources that (1) can be edited and/or augmented by an adversary, and (2) are weakly monitored, so the adversary's modifications can survive until they are used for training.

We consider two distinct adversarial objectives, both expressed in terms of word proximity in the embedding space. A *rank attacker* wants a particular source word to be ranked

* Abbreviated version of this paper is published in the proceedings of the 2020 IEEE Symposium on Security and Privacy.
† This research was performed at Cornell Tech.

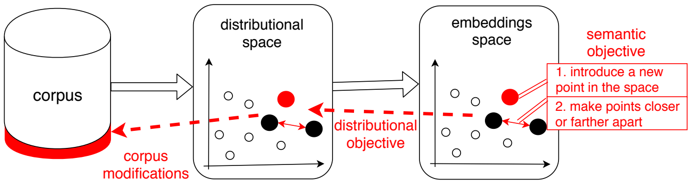


Figure I.2: Semantic changes via corpus modifications.

high among the target word's neighbors. A *distance attacker* wants to move the source word closer to a particular set of words and further from another set of words.

Achieving these objectives via corpus poisoning requires first answering a fundamental question: **how do changes in the corpus correspond to changes in the embeddings?** Neural embeddings are derived using an opaque optimization procedure over corpus elements, thus it is not obvious how, given a desired change in the embeddings, to compute specific corpus modifications that achieve this change.

Our contributions. First, we show how to relate proximity in the embedding space to *distributional*, aka explicit expressions over corpus elements, computed with basic arithmetics and no weight optimization. Word embeddings are expressly designed to capture (a) first-order proximity, i.e., words that frequently occur together in the corpus, and (b) second-order proximity, i.e., words that are similar in the "company they keep" (they frequently appear with the same set of other words, if not with each other). We develop **distributional expressions that capture both types of semantic proximity**, separately and together, in ways that closely correspond to how they are captured in the embeddings. Crucially, the relationship is causative: changes in our distributional expressions produce predictable changes in the embedding distances.

Second, we develop and evaluate a **methodology for introducing adversarial semantic changes in the embedding space**, depicted in Figure I.2. As proxies for the semantic objectives, we use distributional objectives, expressed and solved as an optimization problem over word-cooccurrence counts. The attacker then computes corpus modifications that achieve the desired counts. We show that our attack is effective against popular embedding models—even if the attacker has only a small sub-sample of the victim's training corpus and does not know the victim's specific model and hyperparameters.

Third, we demonstrate the power and universality of our **attack on several practical NLP tasks** with the embeddings trained on Twitter and Wikipedia. By poisoning the embedding, we (1) trick a resume search engine into picking a specific resume as the top result for queries with chosen terms such as "iOS" or "devops"; (2) prevent a Named Entity Recognition model from identifying specific corporate names or else identify them with higher recall; and (3) make a word-to-word translation model confuse an attacker-made word with an arbitrary English word, regardless of the target language.

Finally, we show how to morph the attacker's word sequences so they appear as linguistically likely as actual sentences from the corpus, measured by the perplexity scores of a language model (the attacker does not need to know the specifics of the latter). Filtering out high-perplexity sentences thus has prohibitively many false positives and false negatives, and **using a language model to "sanitize" the training corpus is ineffective**. Aggressive filtering drops the majority of the actual corpus and still does not foil the attack.

To the best of our knowledge, ours is the first data-poisoning attack against transfer learning. Furthermore, embedding-based NLP tasks are sophisticated targets, with two consecutive training processes (one for the embedding, the other for the downstream task) acting as levels of indirection. A single attack on an embedding can thus potentially affect multiple, diverse downstream NLP models that all rely on this embedding to provide the semantics of words in a language.

II. Prior work

Interpreting word embeddings. Levy and Goldberg [41] argue that SGNS factorizes a matrix whose entries are derived from cooccurrence counts. Arora et al. [7, 8], Hashimoto et al. [31], and Ethayarajh et al. [26] analytically derive explicit expressions for embedding distances, but these expressions are not directly usable in our setting—see Section IV-A. (Unwieldy) distributional representations have traditionally been used in information retrieval [28, 70]; Levy and Goldberg [40] show that they can perform similarly to neural embeddings on analogy tasks. Antoniak et al. [5] empirically study the stability of embeddings under various hyperparameters.

The problem of modeling causation between corpus features and embedding proximities also arises when mitigating stereotypical biases encoded in embeddings [12]. Brunet et al. [13] recently analyzed GloVe's objective to detect and remove articles that contribute to bias, given as an expression over word vector proximities.

To the best of our knowledge, we are the first to develop explicit expressions for word proximities over corpus cooccurrences, such that changes in expression values produce consistent, predictable changes in embedding proximities.

Poisoning neural networks. Poisoning attacks inject data into the training set [16, 48, 65, 67, 76] to insert a "backdoor" into the model or degrade its performance on certain inputs. Our attack against embeddings can inject new words (Section IX) and cause misclassification of existing words (Section X). It is the first attack against two-level transfer learning: it poisons the training data to change relationships in the embedding space, which in turn affects downstream NLP tasks.

Poisoning matrix factorization. Gradient-based poisoning attacks on matrix factorization have been suggested in the context of collaborative filtering [43] and adapted to unsupervised node embeddings [68]. These approaches are computationally prohibitive because the matrix must be factorized at every optimization step, nor do they work in our setting, where most gradients are 0 (see Section VI).

Bojchevski and Gúnnerman recently suggested an attack on node embeddings that does not use gradients [11] but the computational cost remains too high for natural-language cooccurrence graphs where the dictionary size is in the millions. Their method works on graphs, not text; the mapping between the two is nontrivial (we address this in Section VII). The only task considered in [11] is generic node classification, whereas we work in a complete transfer learning scenario.

Adversarial examples. There is a rapidly growing literature on *test-time* attacks on neural-network image classifiers [3, 37, 38, 49, 69]; some employ only black-box model queries [15, 33] rather than gradient-based optimization. We, too, use a non-gradient optimizer to compute cooccurrences that achieve the desired effect on the embedding, but in a setting where queries are cheap and computation is expensive.

Neural networks for text processing are just as vulnerable to adversarial examples, but example generation is more challenging due to the non-differentiable mapping of text elements to the embedding space. Dozens of attacks and defenses have been proposed [4, 10, 22, 29, 34, 46, 62, 63, 72, 73].

By contrast, we study *training-time* attacks that change word embeddings so that multiple downstream models behave incorrectly on *unmodified* test inputs.

III. Background and notation

Table I summarizes our notation. Let $\mathbb{D}$ be a dictionary of words and $\mathbb{C}$ a corpus, i.e., a collection of word sequences. A word embedding algorithm aims to learn a low-dimensional vector $\{\vec{e}_u\}$ for each $u \in \mathbb{D}$. Semantic similarity between words is encoded as the cosine similarity of their corresponding vectors, $\cos(\vec{y}, \vec{z}) \overset{\text{def}}{=} \vec{y} \cdot \vec{z} / \sqrt{\|\vec{y}\|_2 \|\vec{z}\|_2}$ where $\vec{y} \cdot \vec{z}$ is the vector dot product. The cosine similarity of L2-normalized vectors is (1) equivalent to their dot product, and (2) linear in negative squared L2 (Euclidean) distance.

Embedding algorithms start from a high-dimensional representation of the corpus, its *cooccurrence matrix* $\{C_{u,v}\}_{u,v \in \mathbb{D}}$ where $C_{u,v}$ is a weighted sum of cooccurrence events, i.e., appearances of u, v in proximity to each other. Function $\gamma(d)$ gives each event a weight that is inversely proportional to the distance d between the words.

Embedding algorithms first learn two intermediate representations for each word $u \in \mathbb{D}$, the *word vector* $\vec{w}_u$ and the *context vector* $\vec{c}_u$, then compute $\vec{e}_u$ from them.

GloVe. GloVe defines and optimizes (via SGD) a minimization objective directly over cooccurrence counts, weighted by $\gamma(d) = \begin{cases} 1/d & d \le \Gamma \\ 0 & \text{else} \end{cases}$ for some window size Γ:

$$\operatorname{argmin} \left\{ \sum_{u,v \in \mathbb{D}} \left\{ g(C_{u,v}) \cdot \left(\vec{w}_u \cdot \vec{c}_v + b_u + b'_v - \log(C_{u,v}) \right)^2 \right\} \right\}, \tag{III.1}$$

where argmin is taken over the parameters $\{\vec{c}, \vec{w}, b, b'\}$. b_u, b'_u are scalar bias terms that are learned along with the word and context vectors, and $g(c) \overset{\text{def}}{=} \begin{cases} c^{3/4}, & c \le c_{max} \\ c_{max}, & \text{else} \end{cases}$ for some parameter c_{max} (typically $c_{max} \in [10, 100]$). At the end of the training, GloVe sets the embedding $\vec{e}_u \leftarrow \vec{w}_u + \vec{c}_u$.

III	$\mathbb{C}$	corpus
	$\mathbb{D}$	dictionary
	u, v, r	dictionary words
	$\{\vec{e}_u\}_{u \in \mathbb{D}}$	embedding vectors
	$\{\vec{w}_u\}_{u \in \mathbb{D}}$	"word vectors"
	$\{\vec{c}_u\}_{u \in \mathbb{D}}$	"context vectors"
	b_u, b'_v	GloVe bias terms, see Equation III.1
	$\cos(\vec{y}, \vec{z})$	cosine similarity
	$C \in \mathbb{R}^{\lvert\mathbb{D}\rvert \times \lvert\mathbb{D}\rvert}$	$\mathbb{C}$'s cooccurrence matrix
	$\vec{C_u} \in \mathbb{R}^{\lvert\mathbb{D}\rvert}$	u's row in C
	Γ	size of window for cooccurrence counting
	$\gamma : \mathbb{N} \to \mathbb{R}$	cooccurrence event weight function
	SPPMI	matrix defined by Equation III.2
	BIAS	matrix defined by Equation III.3
IV	$\mathrm{SIM}_1(u, v)$	$\vec{w}_u \cdot \vec{c}_v + \vec{c}_u \cdot \vec{w}_v$, see Equation III.4
	$\mathrm{SIM}_2(u, v)$	$\vec{w}_u \cdot \vec{w}_v + \vec{c}_u \cdot \vec{c}_v$, see Equation III.4
	$\{B_u\}_{u \in \mathbb{D}}$	word bias terms, to downweight common words
	$f_{u,v}(c, \epsilon)$	$\max\{log(c) - B_u - B_v, \epsilon\}$
	$M \in \mathbb{R}^{\lvert\mathbb{D}\rvert \times \lvert\mathbb{D}\rvert}$	matrix with entries of the form $f_{u,v}(c, 0)$ (e.g., SPPMI, BIAS)
	$\vec{M}_u \in \mathbb{R}^{\lvert\mathbb{D}\rvert}$	u's row in M
	$\widehat{\mathrm{SIM}}_1(u, v)$	explicit expression for $\vec{c}_u \cdot \vec{w}_v$, set as $M_{u,v}$
	$N_{u,v}$	normalization term for first-order proximity
	$\widehat{\mathrm{sim}}_1(u, v)$	explicit expression for $\cos(\vec{c}_u, \vec{w}_v)$, set as $f_{u,v}(C_{u,v}, 0) / N_{u,v}$
	$\widehat{\mathrm{sim}}_2(u, v)$	explicit expression for $\cos(\vec{w}_u, \vec{w}_v)$, set as $\cos\left(\vec{M}_u, \vec{M}_v\right)$
	$\widehat{\mathrm{sim}}_{1+2}(u, v)$	$\widehat{\mathrm{sim}}_1(u, v)/2 + \widehat{\mathrm{sim}}_2(u, v)/2$
	$\mathrm{LCO} \in \mathbb{R}^{\lvert\mathbb{D}\rvert \times \lvert\mathbb{D}\rvert}$	entries defined by $\max\{log(C_{u,v}), 0\}$
V	Δ	word sequences added by the attacker
	$\mathbb{C} + \Delta$	corpus after the attacker's additions
	$\lvert\Delta\rvert$	size of the attacker's additions, see Section V
	$s, t \in \mathbb{D}$	source, target words
	$\mathrm{NEG}, \mathrm{POS}$	"positive" and "negative" target words
	$\mathrm{sim}_\Delta(u, v)$	embedding cosine similarity after the attack
	$J(s, \mathrm{NEG}, \mathrm{POS}; \Delta)$	embedding objective
	max_Δ	proximity attacker's maximum allowed $\lvert\Delta\rvert$
	r	rank attacker's target rank
	$\langle t \rangle_r$	rank attacker's minimum proximity threshold
	$\widehat{\mathrm{sim}}(u, v)$	distributional expression for cosine similarity
	$\widehat{\mathrm{sim}}_{\widehat{\Delta}}(u, v)$	distributional expression for $\mathrm{sim}_\Delta(u, v)$
	$\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$	distributional objective
	$\widehat{\langle t \rangle_r}$	rank attacker's estimated threshold for distributional proximity
	α	"safety margin" for $\widehat{\langle t \rangle_r}$ estimation error
	$C_{[[s] \leftarrow \vec{C}_s + \widehat{\Delta}]}$	cooccurrence matrix after adding $\widehat{\Delta}$
VI	$\mathbb{L} \subseteq \mathbb{R}$	possible changes at every step, set to $\left\{\frac{1}{5}, \frac{2}{5}, \ldots, \frac{30}{5}\right\}$
	$i \in \mathbb{D}$	index into $\widehat{\Delta}$, also a word in $\mathbb{D}$
	$\delta \in \mathbb{L}$	increase in $\widehat{\Delta}_i$ in optimization step
	$d_{i,\delta}\left[\widehat{X}\right]$	change in expression X when adding δ to $\widehat{\Delta}_i$
	λ	words to each side of s in sequences aiming to increase second-order proximity
	$\vec{\omega}$	vector such that $\lvert\Delta\rvert \le \vec{\omega} \cdot \widehat{\Delta}$
	$\Omega_{search}, \Omega_{corp}, \Omega_{trans}, \Omega_{rank}, \Omega_{benchmark}$	sets of attacked words in our experiments
	C', M', B' $\widehat{\mathrm{sim}}'_1, \widehat{\mathrm{sim}}'_2, \widehat{\mathrm{sim}}'_{1+2}$	expressions computed using $C_{[[s] \leftarrow \vec{C}_s + \widehat{\Delta}]}$

Table I: Notation reference.

Word2vec. Word2vec [50] is a family of models that optimize objectives over corpus cooccurrences. In this paper, we experiment with the skip-gram with negative sampling (SGNS) and CBOW with hierarchical softmax (CBHS). In contrast to GloVe, Word2vec discards context vectors and uses word vectors $\vec{w}_u$ as the embeddings, i.e., $\forall u \in \mathbb{D} : \vec{e}_u \leftarrow \vec{w}_u$.

Appendix A provides further details.

There exist other embeddings, such as FastText, but understanding them is not required as the background for this paper.

Contextual embeddings. Contextual embeddings [20, 57] support dynamic word representations that change depending on the context of the sentence they appear in, yet, in expectation, form an embedding space with non-contextual relations [64]. In this paper, we focus on the popular non-contextual embeddings because (a) they are faster to train and easier to store, and (b) many task solvers use them by construction (see Sections IX through XI).

Distributional representations. A *distributional* or *explicit* representation of a word is a high-dimensional vector whose entries correspond to cooccurrence counts with other words.

Dot products of the learned word vectors and context vectors ($\vec{w}_u \cdot \vec{c}_v$) seem to correspond to entries of a high-dimensional matrix that is closely related to, and directly computable from, the cooccurrence matrix. Consequently, both SGNS and GloVe can be cast as matrix factorization methods. Levy and Goldberg [41] show that, assuming training with unlimited dimensions, SGNS's objective has an optimum at $\forall u, v \in \mathbb{D} : \vec{w}_u \cdot \vec{c}_v = \mathrm{SPPMI}_{u,v}$ defined as:

$$\mathrm{SPPMI}_{u,v} \stackrel{\mathrm{def}}{=} \max \left\{ log\left(C_{u,v}\right) - log\left(\sum_{r\in\mathbb{D}} C_{u,r}\right) - log\left(\sum_{r\in\mathbb{D}} C_{v,r}\right) + log\left(Z/k\right), 0 \right\} \tag{III.2}$$

where k is the negative-sampling constant and $Z \stackrel{\mathrm{def}}{=} \sum_{u,v\in\mathbb{D}} C_{u,v}$. This variant of pointwise mutual information (PMI) downweights a word's cooccurrences with common words because they are less "significant" than cooccurrences with rare words. The rows of the SPPMI matrix define a distributional representation.

GloVe's objective similarly has an optimum $\forall u, v \in \mathbb{D} : \vec{w}_u \cdot \vec{c}_v = \mathrm{BIAS}_{u,v}$ defined as:

$$\mathrm{BIAS}_{u,v} \stackrel{\mathrm{def}}{=} \max \left\{ log\left(C_{u,v}\right) - b_u - b'_v, 0 \right\} \tag{III.3}$$

$\max$ is a simplification: in rare and negligible cases, the optimum of $\vec{w}_u \cdot \vec{c}_v$ is slightly below 0. Similarly to SPPMI, BIAS downweights cooccurrences with common words (via the learned bias values b_u, b'_v).

First- and second-order proximity. We expect words that frequently cooccur with each other to have high semantic proximity. We call this *first-order proximity*. It indicates that the words are related but not necessarily that their meanings are similar (e.g., "first class" or "polar bear").

The distributional hypothesis [27] says that distributional vectors capture semantic similarity by *second-order proximity*: the more contexts two words have in common, the higher their similarity, regardless of their cooccurrences with each other. For example, "terrible" and "horrible" hardly ever cooccur, yet their second-order proximity is very high. Levy and Goldberg [40] showed that linear relationships of distributional representations are similar to those of word embeddings.

Levy and Goldberg [42] observe that, summing the context and word vectors $\vec{e}_u \leftarrow \vec{w}_u + \vec{c}_u$, as done by default in GloVe, leads to the following:

$$\vec{e}_u \cdot \vec{e}_v = \mathrm{SIM}_1\left(u, v\right) + \mathrm{SIM}_2\left(u, v\right) \tag{III.4}$$

where $\mathrm{SIM}_1\left(u, v\right) \stackrel{\mathrm{def}}{=} \vec{w}_u \cdot \vec{c}_v + \vec{c}_u \cdot \vec{w}_v$ and $\mathrm{SIM}_2\left(u, v\right) \stackrel{\mathrm{def}}{=} \vec{w}_u \cdot \vec{w}_v + \vec{c}_u \cdot \vec{c}_v$. They conjecture that SIM_1 and SIM_2 correspond to, respectively, first- and second-order proximities.

Indeed, SIM_1 seems to be a measure of cooccurrence counts, which measure first-order proximity: Equation III.3 leads to $\mathrm{SIM}_1\left(u, v\right) \approx 2\,\mathrm{BIAS}_{u,v}$. BIAS is symmetrical up to a small error, stemming from the difference between GloVe bias terms b_u and b'_u, but they are typically very close—see Section IV-B. This also assumes that the embedding optimum perfectly recovers the BIAS matrix.

There is no distributional expression for $\mathrm{SIM}_2\left(u, v\right)$ that does not rely on problematic assumptions (see Section IV-A), but there is ample evidence for the conjecture that SIM_2 somehow captures second-order proximity (see Section IV-B). Since word and context vectors and their products typically have similar ranges, Equation III.4 suggests that embeddings weight first- and second-order proximities equally.

IV. From embeddings to expressions over corpus

The key problem that must be solved to control word meanings via corpus modifications is finding a *distributional expression*, i.e., an explicit expression over corpus features such as cooccurrences, for the embedding distances, which are the computational representation of "meaning."

A. Previous work is not directly usable

Several prior approaches [7, 8, 26] derive distributional expressions for distances between word vectors, all of the form $\vec{e}_u \cdot \vec{e}_v \approx A \cdot log\left(C_{u,v}\right) - B_u - B'_v$. The downweighting role of B_u, B'_v seems similar to SPPMI and BIAS, thus these expressions, too, can be viewed as variants of PMI.

These approaches all make simplifying assumptions that do not hold in reality. Arora et al. [7, 8] and Hashimoto et al. [31] assume a generative language model where words are emitted by a random walk. Both models are parameterized by low-dimensional word vectors $\{\vec{e*}_u\}_{u\in\mathbb{D}}$ and assume that context and word vectors are identical. Then they show how $\{\vec{e*}_u\}_{u\in\mathbb{D}}$ optimize the objectives of GloVe and SGNS.

By their very construction, these models uphold a very strong relationship between cooccurrences and low-dimensional representation products. In Arora et al., these products are equal to PMIs; in Hashimoto et al., the vectors' L2 norm differences, which are closely related to their product, approximate their cooccurrence count. *If* such "convenient" low-dimensional vectors exist, it should not be surprising that they optimize GloVe and SGNS.

The approximation in Ethayarajh et al. [26] only holds within a single set of word pairs that are "contextually coplanar," which loosely means they appear in related contexts. It is unclear if coplanarity holds in reality over large sets of word pairs, let alone the entire dictionary.

Some of the above papers use correlation tests to justify their conclusion that dot products follow SPPMI-like expressions. Crucially, correlation does not mean that the embedding space is derived from (log)-cooccurrences in a distance-preserving fashion, thus correlation is not sufficient to control the embeddings. We want not just to characterize how embedding distances *typically* relate to corpus elements, but to achieve a specific change in the distances. To this end, we need an explicit expression over corpus elements whose value is encoded in the embedding distances by the embedding algorithm (Figure I.2).

Furthermore, these approaches barter generality for analytic simplicity and derive distributional expressions that do not account for second-order proximity at all. As a consequence, the values of these expressions can be very different from the embedding distances, since words that only rarely appear in the same window (and thus have low PMI) may be close in the embedding space. For example, “horrible” and “terrible” are so semantically close they can be used as synonyms, yet they are also similar phonetically and thus their adjacent use in natural speech and text appears redundant. In a dim-100 GloVe model trained on Wikipedia, “terrible” is among the top 3 words closest to “horrible” (with cosine similarity 0.8). However, when words are ordered by their PMI with “horrible,” “terrible” is only in the 3675th place.

B. Our approach

We aim to find a distributional expression for the semantic proximity encoded in the embedding distances. The first challenge is to find distributional expressions for both first- and second-order proximities encoded by the embedding algorithms. The second is to combine them into a single expression corresponding to embedding proximity.

First-order proximity. First-order proximity corresponds to cooccurrence counts and is relatively straightforward to express in terms of corpus elements. Let M be the matrix that the embeddings factorize, e.g., SPPMI for SGNS (Equations III.2) or BIAS for GloVe (Equations III.3). The entries of this matrix are natural explicit expressions for first-order proximity, since they approximate $\mathrm{SIM}_1(u,v)$ from Equation III.4 (we omit multiplication by two as it is immaterial):

$$\widehat{\mathrm{SIM}_1}(u,v) \stackrel{\text{def}}{=} M_{u,v} \tag{IV.1}$$

$M_{u,v}$ is typically of the form $\max\{log(C_{u,v}) - B_u - B_v, 0\}$ where B_u, B_v are the “downweighting” scalar values (possibly depending on u, v's rows in C). For SPPMI, we set $B_u = log\left(\sum_{r\in\mathbb{D}} C_{u,r}\right) - log(Z/k)/2$; for $BIAS$, $B_u = b_u$.[1]

Second-order proximity. Let the distributional representation $\vec{M}_u$ of u be its row in M. We hypothesize that distances in this representation correspond to second-order proximity encoded in the embedding-space distances.

[1]We consider $\mathrm{BIAS}_{u,v}$ as a distributional expression even though it depends on b_u, b'_v learned during GloVe's optimization because these terms can be closely approximated using pre-trained GloVe embeddings—see Appendix B. For simplicity, we also assume that $b_u = b'_u$ (thus BIAS is of the required form); in practice, the difference is very small.

First, the objectives of the embedding algorithms seem to directly encode this connection. Consider a word w's projection onto GloVe's objective III.1:

$$J_{GloVe}[u] = \sum_{v\in\mathbb{D}} g(C_{u,v})\left(\vec{w}_u^T\vec{c}_v + b_u + b'_v - \log\ C_{u,v}\right)^2$$

This expression is determined entirely by u's row in M_{BIAS}. If two words have the same distributional vector, their expressions in the optimization objective will be completely symmetrical, resulting in very close embeddings—even if their cooccurrence count is 0. Second, the view of the embeddings as matrix factorization implies an approximate linear transformation between the distributional and embedding spaces. Let $C \stackrel{\text{def}}{=} \left[\vec{c}_{u_1}\ldots\vec{c}_{u_{|\mathbb{D}|}}\right]^T$ be the matrix whose rows are context vectors of words $u_i \in \mathbb{D}$. Assuming M is perfectly recovered by the products of word and context vectors, $C\cdot\vec{w}_u = \vec{M}_u$.

Dot products have very different scale in the distributional and embedding spaces. Therefore, we use cosine similarities, which are always between -1 and 1, and set

$$\widehat{\mathrm{sim}_2}(u,v) \stackrel{\text{def}}{=} \cos\left(\vec{M}_u, \vec{M}_v\right) \tag{IV.2}$$

As long as M entries are nonnegative, the value of this expression is always between 0 and 1.

Combining first- and second-order proximity. Our expressions for first- and second-order proximities have different scales: $\widehat{\mathrm{SIM}_1}(u,v)$ corresponds to an unbounded dot product, while $\widehat{\mathrm{sim}_2}(u,v)$ is at most 1. To combine them, we normalize $\widehat{\mathrm{SIM}_1}(u,v)$. Let $f_{u,v}(c,\epsilon) \stackrel{\text{def}}{=} \max\{log(c) - B_u - B_v, \epsilon\}$, then $\widehat{\mathrm{SIM}_1}(u,v) = M_{u,v} = f_{u,v}(C_{u,v}, 0)$. We set $N_{u,v} \stackrel{\text{def}}{=} \sqrt{f_{u,v}\left(\sum_{r\in\mathbb{D}} C_{u,r}, e^{-60}\right)}\sqrt{f_{u,v}\left(\sum_{r\in\mathbb{D}} C_{v,r}, e^{-60}\right)}$ as the normalization term. This is similar to the normalization term of cosine similarity and ensures that the value is between 0 and 1. The $\max$ operation is taken with a small e^{-60}, rather than 0, to avoid division by 0 in edge cases. We set $\widehat{\mathrm{sim}_1}(u,v) \stackrel{\text{def}}{=} f_{u,v}(C_{u,v}, 0)/N_{u,v}$. Our combined distributional expression for the embedding proximity is

$$\widehat{\mathrm{sim}_{1+2}}(u,v) \stackrel{\text{def}}{=} \widehat{\mathrm{sim}_1}(u,v)/2 + \widehat{\mathrm{sim}_2}(u,v)/2 \tag{IV.3}$$

Since $\widehat{\mathrm{sim}_1}(u,v)$ and $\widehat{\mathrm{sim}_2}(u,v)$ are always between 0 and 1, the value of this expression, too, is between 0 and 1.

Correlation tests. We trained a GloVe-`paper` and a SGNS model on full Wikipedia, as described in Section VIII. We randomly sampled (without replacement) 500 “source” words and 500 “target” words from the 50,000 most common words in the dictionary and computed the distributional expressions $\widehat{\mathrm{sim}_1}(u,v)$, $\widehat{\mathrm{sim}_2}(u,v)$, and $\widehat{\mathrm{sim}_{1+2}}(u,v)$, for all 250,000 source-target word pairs using $M \in \{\mathrm{SPPMI}, \mathrm{BIAS}, \mathrm{LCO}\}$ where LCO is defined by $\mathrm{LCO}(u,v) \stackrel{\text{def}}{=} \max\{log(C_{u,v}), 0\}$. We then computed the correlations between distributional proximities and (1) embedding proximities, and (2) word-context proximities $\cos(\vec{w}_u, \vec{c}_v)$ and word-word proximities $\cos(\vec{w}_u, \vec{w}_v)$, using GloVe's word and context vectors. These correspond, respectively, to first- and second-order proximities encoded in the embeddings.

	M	$\widehat{\text{sim}}_1(u,v)$	$\widehat{\text{sim}}_2(u,v)$	$\widehat{\text{sim}}_{1+2}(u,v)$
GloVe	BIAS	0.47	0.53	**0.56**
	SPPMI	0.31	0.35	0.36
	LCO	0.36	0.43	0.50
SGNS	BIAS	0.31	0.29	0.32
	SPPMI	0.21	**0.47**	0.36
	LCO	0.21	0.31	0.34

Table II: Correlation of distributional proximity expressions, computed using different distributional matrices, with the embedding proximities $\cos(\vec{e}_u, \vec{e}_v)$.

expression	$\widehat{\text{sim}}_1(u,v)$	$\widehat{\text{sim}}_2(u,v)$	$\widehat{\text{sim}}_{1+2}(u,v)$
$\cos(\vec{w}_u, \vec{c}_v)$	<u>0.50</u>	0.49	**0.54**
$\cos(\vec{w}_u, \vec{w}_v)$	0.40	<u>0.51</u>	**0.52**
$\cos(\vec{e}_u, \vec{e}_v)$	0.47	0.53	<u>**0.56**</u>

Table III: Correlation of distributional proximity expressions with cosine similarities in GloVe's low-dimensional representations $\{\vec{w}_u\}$ (word vectors), $\{\vec{c}_u\}$ (context vectors), and $\{\vec{e}_u\}$ (embedding vectors), measured over 250,000 word pairs.

Tables II and III show the results. Observe that (1) in GloVe, $\widehat{\text{sim}}_{1+2}(u,v)$ consistently correlates better with the embedding proximities than either the first- or second-order expressions alone. (2) In SGNS, by far the strongest correlation is with $\widehat{\text{sim}}_2$ computed using SPPMI. (3) The highest correlations are attained using the matrices factorized by the respective embeddings. (4) The values on Table II's diagonal are markedly high, indicating that SIM_1 correlates highly with $\widehat{\text{sim}}_1$, SIM_2 with $\widehat{\text{sim}}_2$, and their combination with $\widehat{\text{sim}}_{1+2}$. (5) First-order expressions correlate worse than second-order and combined ones, indicating the importance of second-order proximity for semantic proximity. This is especially true for SGNS, which does not sum the word and context vectors.

V. Attack methodology

Attacker capabilities. Let $s \in \mathbb{D}$ be a "source word" whose meaning the attacker wants to change. The attacker is targeting a victim who will train his embedding on a specific public corpus, which may or may not be known to the attacker in its entirety. The victim's choice of the corpus is mandated by the nature of the task and limited to a few big public corpora believed to be sufficiently rich to represent natural language (English, in our case). For example, Wikipedia is a good choice for word-to-word translation models because it preserves cross-language cooccurrence statistics [18], whereas Twitter is best for named-entity recognition in tweets [17]. The embedding algorithm and its hyperparameters are typically public and thus known to the attacker, but we also show in Section VIII that the attack remains effective if the attacker uses a small subsample of the target corpus as a surrogate and very different embedding hyperparameters.

The attacker need not know the details of downstream models. The attacks in Sections IX–XI make only general assumptions about their targets, and we show that a single attack on the embedding can fool multiple downstream models.

We assume that the attacker can add a collection Δ of short word sequences, up to 11 words each, to the corpus. In Section VIII, we explain how we simulate sequence insertion. In Appendix G, we also consider an attacker who can edit existing sequences, which may be viable for publicly editable corpora such as Wikipedia.

We define the size of the attacker's modifications $|\Delta|$ as the bigger of (a) the maximum number of appearances of a single word, i.e., the L_∞ norm of the change in the corpus's word-count vector, and (b) the number of added sequences. Thus, L_∞ of the word-count change is capped by $|\Delta|$, while L_1 is capped by $11\,|\Delta|$.

Overview of the attack. The attacker wants to use his corpus modifications Δ to achieve a certain objective for s in the embedding space while minimizing $|\Delta|$.

0. Find distributional expression for embedding distances. The preliminary step, done once and used for multiple attacks, is to (0) find distributional expressions for the embedding proximities. Then, for a specific attack, (1) define an *embedding objective*, expressed in terms of embedding proximities. Then, (2) derive the corresponding *distributional objective*, i.e., an expression that links the embedding objective with corpus features, with the property that if the distributional objective holds, then the embedding objective is likely to hold. Because a distributional objective is defined over C, the attacker can express it as an optimization problem over cooccurrence counts, and (3) solve it to obtain the cooccurrence *change vector*. The attacker can then (4) transform the cooccurrence change vector to a *change set* of corpus edits and apply them. Finally, (5) the embedding is trained on the modified corpus, resulting in the attacker's changes propagating to the embedding. Figure V.1 depicts this process.

As explained in Section IV, the goal is to find a distributional expression $\widehat{\text{sim}}(u,v)$ that, if upheld in the corpus, will cause a corresponding change in the embedding distances.

First, the attacker needs to know the corpus cooccurrence counts C and the appropriate first-order proximity matrix M (see Section IV-B). Both depend on the corpus and the embedding algorithm and its hyperparameters, but can also be computed from available proxies (see Section VIII).

Using C and M, set $\widehat{\text{sim}}$ as $\widehat{\text{sim}}_{1+2}$, $\widehat{\text{sim}}_1$ or $\widehat{\text{sim}}_2$ (see Section IV-B). We found that the best choice depends on the embedding (see Section VIII). For example, for GloVe, which puts similar weight on first- and second-order proximity (see Section III.4), $\widehat{\text{sim}}_{1+2}$ is the most effective; for SGNS, which only uses word vectors, $\widehat{\text{sim}}_2$ is slightly more effective.

1. Derive an embedding objective. We consider two types of adversarial objectives. An attacker with a *proximity objective* wants to push s away from some words (we call them "negative") and closer to other words ("positive") in the embedding space. An attacker with a *rank objective* wants to make s the rth closest embedding neighbor of some word t.

To formally define these objectives, first, given two sets of words $\text{NEG}, \text{POS} \in \mathcal{P}(\mathbb{D})$, define

$$J(s, \text{NEG}, \text{POS}; \Delta) \stackrel{\text{def}}{=} \frac{1}{|\text{POS}| + |\text{NEG}|}\left(\sum_{t \in \text{POS}} \text{sim}_\Delta(s,t) - \sum_{t \in NEG} \text{sim}_\Delta(s,t)\right)$$

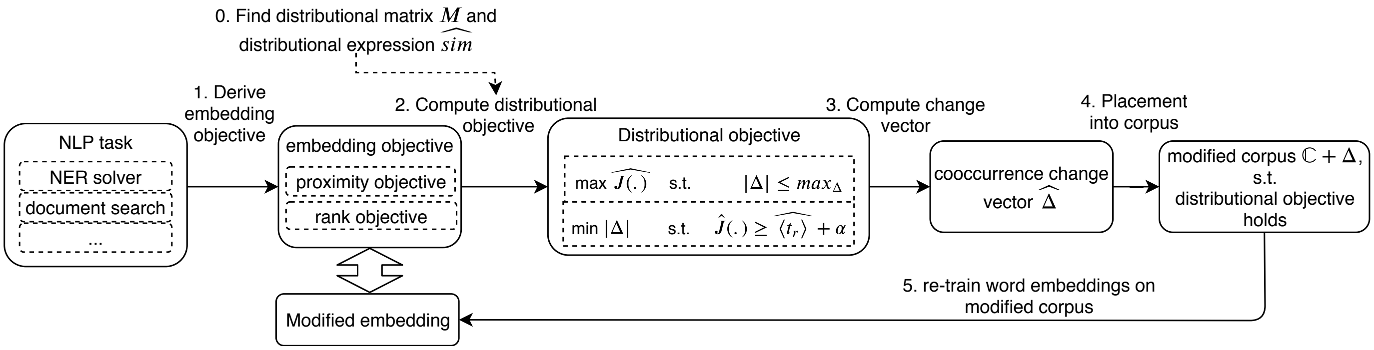


Figure V.1: Overview of our attack methodology.

where $\mathrm{sim}_{\Delta}(u, v) = \cos(\vec{e}_u, \vec{e}_v)$ is the cosine similarity function that measures pairwise word proximity (see Section III) when the embeddings are computed on the modified corpus $\mathbb{C} + \Delta$. $J(s, \mathrm{NEG}, \mathrm{POS}; \Delta)$ penalizes s's proximity to the words in NEG and rewards proximity to the words in POS.

Given POS, NEG, and a threshold max_{Δ}, define the **proximity objective** as

$$\mathrm{argmax}_{\Delta, |\Delta| \leq max_{\Delta}} J(s, \mathrm{NEG}, \mathrm{POS}; \Delta)$$

This objective makes a word semantically farther from or closer to another word or cluster of words.

Given some rank r, define the **rank objective** as finding a minimal Δ such that s is one of t's r closest neighbors in the embedding. Let $\langle t \rangle_r$ be the proximity of t to its rth closest embedding neighbor. Then the rank constraint is equivalent to $\mathrm{sim}_{\Delta}(s, t) \geq \langle t \rangle_r$, and the objective can be expressed as

$$\mathrm{argmin}_{\Delta, \mathrm{sim}_{\Delta}(s,t) \geq \langle t \rangle_r} |\Delta|$$

or, equivalently,

$$\mathrm{argmin}_{\Delta, J(s, \emptyset, \{t\}; \Delta) \geq \langle t \rangle_r} |\Delta|$$

This objective is useful, for example, for injecting results into a search query (see Section IX).

2. From embedding objective to distributional objective. We now transform the optimization problem $J(s, \mathrm{NEG}, \mathrm{POS}; \Delta)$, expressed over changes in the corpus and embedding proximities, to a distributional objective $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$, expressed over changes in the cooccurrence counts and distributional proximities. The *change vector* $\widehat{\Delta}$ denotes the change in s's cooccurrence vector that corresponds to adding Δ to the corpus. This transformation involves several steps.

(a) Changes in corpus $\leftrightarrow$ changes in cooccurrence counts: We use a *placement strategy* that takes a vector $\widehat{\Delta}$, interprets it as additions to s's cooccurrence vector, and outputs Δ such that s's cooccurrences in the new corpus $\mathbb{C} + \Delta$ are $\vec{C}_s + \widehat{\Delta}$. Other rows in C remain almost unchanged. Our objective can now be expressed over $\widehat{\Delta}$ as a surrogate for Δ. It still uses the size of the corpus change, $|\Delta|$, which is easily computable from $\widehat{\Delta}$ without computing Δ as explained below.

(b) Embedding proximity $\leftrightarrow$ distributional proximity: We assume that embedding proximities are monotonously increasing (respectively, decreasing) with distributional proximities. Figure A.2–c in Appendix E shows this relationship.

(c) Embedding threshold $\leftrightarrow$ distributional threshold: For the rank objective, we want to increase the embedding proximity past a threshold $\langle t \rangle_r$. We heuristically determine a threshold $\widehat{\langle t \rangle}_r$ such that, if the distributional proximity exceeds $\widehat{\langle t \rangle}_r$, the embedding proximity exceeds $\langle t \rangle_r$. Ideally, we would like to set $\widehat{\langle t \rangle}_r$ as the distributional proximity from the rth-nearest neighbor of t, but finding the rth neighbor in the distributional space is computationally expensive. The alternative of using words' embedding-space ranks is not straightforward because there exist severe abnormalities[2] and embedding-space ranks are unstable, changing from one training run to another.

Therefore, we approximate the r'th proximity by taking the maximum of distributional proximities from words with ranks $r - m, \ldots, r + m$ in the embedding space, for some m. If $r < m$, we take the maximum over the $2m$ nearest words. To increase the probability of success (at the expense of increasing corpus modifications), we further add a small fraction α ("safety margin") to this maximum.

Let $\widehat{\mathrm{sim}}_{\widehat{\Delta}}(u, v)$ be our distributional expression for $\mathrm{sim}(u, v)$, computed over the cooccurrences $C_{\left[[s] \leftarrow \vec{C}_s + \widehat{\Delta}\right]}$, i.e., $\mathbb{C}$'s cooccurrences where s's row is updated with $\widehat{\Delta}$. Then we define the **distributional objective** as:

$$\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right) \stackrel{\text{def}}{=} \frac{1}{|\mathrm{POS}| + |\mathrm{NEG}|} \left(\sum_{t \in \mathrm{POS}} \widehat{\mathrm{sim}}_{\widehat{\Delta}}(s, t) - \sum_{t \in \mathrm{NEG}} \widehat{\mathrm{sim}}_{\widehat{\Delta}}(s, t) \right)$$

To find the cooccurrence change $\widehat{\Delta}$ for the proximity objective, the attack must solve:

$$\mathrm{argmax}_{\widehat{\Delta} \in \mathbb{R}^n, |\Delta| \leq max_{\Delta}} \widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$$

and for the rank objective:

$$\mathrm{argmin}_{\widehat{\Delta} \in \mathbb{R}^n, \widehat{J}\left(s, \emptyset, \{t\}; \widehat{\Delta}\right) \geq \widehat{\langle t \rangle}_r + \alpha} |\Delta|$$

[2]For example, words with very few instances in the corpus sometimes appear as close embedding neighbors of words with which they have only very loose semantic affiliation and are very far from distributionally.

3. From distributional objective to cooccurence changes. The previous steps produce a distributional objective consisting of a source word s, a positive target word set POS, a negative target word set NEG, and the constraints: either a maximal change set size max_{Δ}, or a minimal proximity threshold $\widehat{\langle t \rangle_r}$.

We solve this objective with an optimization procedure (described in Section VI) that outputs a change vector with the smallest $|\Delta|$ that maximizes the sum of proximities between s and POS minus the sum of proximities with NEG, subject to the constraints. It starts with $\widehat{\Delta} = (0, \ldots 0)$ and iteratively increases the entries in $\widehat{\Delta}$. In each iteration, it increases the entry that maximizes the increase in $\widehat{J}(\ldots)$, divided by the increase in $|\Delta|$, until the appropriate threshold (max_{Δ} or $\widehat{\langle t \rangle_r} + \alpha$) has been crossed.

This computation involves the size of the corpus change, $|\Delta|$. In our placement strategy, $|\Delta|$ is tightly bounded by a known linear combination of $\widehat{\Delta}$'s elements and can therefore be efficiently computed from $\widehat{\Delta}$.

4. From cooccurrence changes to corpus changes. From the cooccurrence change vector $\widehat{\Delta}$, the attacker computes the corpus change Δ using the placement strategy which ensures that, in the modified corpus $\mathbb{C} + \Delta$, the cooccurrence matrix is close to $C_{\left[[s] \leftarrow \vec{C}_s + \widehat{\Delta}\right]}$. Because the distributional objective holds under these cooccurrence counts, it holds in $\mathbb{C} + \Delta$.

$|\Delta|$ should be as small as possible. In Section VII, we show that our placement strategy achieves solutions that are extremely close to optimal in terms of $|\Delta|$, and that $|\Delta|$ is a known linear combination of $\widehat{\Delta}$ elements (as required above).

5. Embeddings are trained. The embeddings are trained on the modified corpus. If the attack has been successful, the attacker's objectives are true in the new embedding.

Recap of the attack parameters. The attacker must first find M and $\widehat{\text{sim}}$ that are appropriate for the targeted embedding. This can be done once. The proximity attacker must then choose the source word s, the positive and negative target-word sets POS, NEG, and the maximum size of the corpus changes max_{Δ}. The rank attacker must choose the source word s, the target word t, the desired rank r, and a "safety margin" α for the transformation from embedding-space thresholds to distributional-space thresholds.

VI. Optimization in cooccurrence-vector space

This section describes the optimization procedure in step 3 of our attack methodology (Figure V.1). It produces a cooccurrence change vector that optimizes the distributional objective from Section V, subject to constraints.

Gradient-based approaches are inadequate. Gradient-based approaches such as SGD result in a poor trade-off between $|\Delta|$ and $\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)$. First, with our distributional expressions, most entries in $\vec{M}_s$ remain 0 in the vicinity of $\widehat{\Delta} = 0$ due to the $\max$ operation in the computation of M (see Section IV-B). Consequently, their gradients are 0. Even if we initialize $\widehat{\Delta}$ so that its entries start from a value where the gradient is non-zero, the optimization will quickly push most entries to 0 to fulfill the constraint $|\Delta| \leq max_{\Delta}$, and the gradients of these entries will be rendered useless. Second, gradient-based approaches may increase vector entries by arbitrarily small values, whereas cooccurrences are drawn from a discrete space because they are linear combinations of cooccurrence event weights (see Section III). For example, if the window size is 5 and the weight is determined by $\gamma = 1 - \frac{d}{5}$, then the possible weights are $\left\{\frac{1}{5}, \ldots \frac{5}{5}\right\}$.

$\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)$ exhibits diminishing returns: usually, the bigger the increase in $\widehat{\Delta}$ entries, the smaller the marginal gain from increasing them further. Such objectives can often be cast as *submodular maximization* [36, 53] problems, which typically lend themselves well to greedy algorithms. We investigate this further in Appendix B.

Our approach. We define a discrete set of step sizes $\mathbb{L}$ and gradually increase entries in $\widehat{\Delta}$ in increments chosen from $\mathbb{L}$ so as to maximize the objective $\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)$. We stop when $|\Delta| > max_{\Delta}$ or $\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right) \geq \widehat{\langle t \rangle_r} + \alpha$.

$\mathbb{L}$ should be fine-grained so the steps are optimal and entries in $\widehat{\Delta}$ map tightly onto cooccurrence events in the corpus, yet $\mathbb{L}$ should have a sufficient range to "peek beyond" the $\max$-threshold where the entry starts getting non-zero values. A natural $\mathbb{L}$ is a subset of the space of linear combinations of possible weights, with an exact mapping between it and a series of cooccurrence events. This mapping, however, cannot be directly computed by the placement strategy (Section VII), which produces an approximation. For better performance, we chose a slightly more coarse-grained $\mathbb{L} \leftarrow \left\{\frac{1}{5}, \ldots \frac{30}{5}\right\}$.

Our algorithm can accommodate $\mathbb{L}$ with negative values, which correspond to *removing* cooccurrence events from the corpus—see Appendix G.

Our optimization algorithm. Let $\widehat{X}_{\widehat{\Delta}}$ be some expression that depends on $\widehat{\Delta}$, and define $d_{i,\delta}\left[\widehat{X}\right] \stackrel{\text{def}}{=} \widehat{X}_{\widehat{\Delta}'} - \widehat{X}_{\widehat{\Delta}}$, where $\widehat{\Delta}$ is the change vector after setting $\widehat{\Delta}_i \leftarrow \widehat{\Delta}_i + \delta$. We initialize $\widehat{\Delta} \leftarrow 0$, and in every step choose

$$i*, \delta* = \text{argmax}_{i \in [|\mathbb{D}|], \delta \in \mathbb{L}} \frac{d_{i,\delta}\left[\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)\right]}{d_{i,\delta}\left[|\Delta|\right]} \tag{VI.1}$$

and set $\widehat{\Delta}_{i*} \leftarrow \widehat{\Delta}_{i*} + \delta*$. If $\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right) \geq \widehat{\langle t \rangle_r} + \alpha$ or $|\Delta| \geq max_{\Delta}$, then quit and return $\widehat{\Delta}$.

Directly computing Equation VI.1 for all i, δ is expensive. The denominator $d_{i,\delta}\left[|\Delta|\right]$ is easy to compute efficiently because it's a linear combination of $\widehat{\Delta}$ elements (see Section VII). The numerator $d_{i,\delta}\left[\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)\right]$, however, requires $O(|\mathbb{L}|\, |\mathbb{D}|^2)$ computations per step (assuming $|\text{NEG}| + |\text{POS}| = O(1)$; in our settings it is ≤ 10). Since $|\mathbb{D}|$ is very big (up to millions of words), this is intractable. Instead of computing each step directly, we developed an algorithm that maintains intermediate values in memory. This is similar to backpropagation, except that we consider variable changes in $\mathbb{L}$ rather than infinitesimally small differentials.

This approach can compute the numerator in $O(1)$ and, crucially, is entirely parallelizable across all i, δ, enabling the computation in every optimization step to be offloaded onto a GPU. In practice, this algorithm finds Δ in minutes (see Section VIII). Full details can be found in Appendix B.

VII. Placement into corpus

The placement strategy is step 4 of our methodology (see Fig. V.1). It takes a cooccurrence change vector $\widehat{\Delta}$ and creates a minimal change set Δ to the corpus such that (a) $|\Delta|$ is bounded by a linear combination $\vec{\omega}$, i.e., $|\Delta| \leq \vec{\omega} \cdot \widehat{\Delta}$, and (b) the optimal value of $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ is preserved.

Our placement strategy first divides $\widehat{\Delta}$ into (1) entries of the form $\widehat{\Delta}_t, t \in \mathrm{POS}$—these changes to $\vec{C}_s$ increase the first-order similarity $\widehat{\mathrm{sim}}_1$ between s and t, and (2) the rest of the entries, which increase the objective in other ways. The strategy adds different types of sequences to Δ to fulfil these two goals. For the first type, it adds multiple, identical *first-order sequences*, containing just the source and target words. For the second type, it adds *second-order sequences*, each containing the source word and 10 other words, constructed as follows. It starts with a collection of sequences containing just s, then iterates over every non-zero entry in $\widehat{\Delta}$ corresponding to the second-order changes $u \in \mathbb{D} \setminus \mathrm{POS}$, and chooses a collection of sequences into which to insert u so that the added cooccurrences of u with s become approximately equal to $\widehat{\Delta}_u$.

This strategy upholds properties (a) and (b) above, achieves (in practice) close to optimal $|\Delta|$, and runs in under a minute in our setup (Section VIII). See Appendix D for details.

VIII. Benchmarks

Datasets. We use a full **Wikipedia** text dump, downloaded on January 20, 2018. For the **Sub-Wikipedia** experiments, we randomly chose 10% of the articles.

Embedding algorithms and hyperparameters. We use Pennington et al.'s original implementation of **GloVe** [56], with two settings for the (hyper)parameters: (1) `paper`, with parameter values from [56]—this is our default, and (2) `tutorial`, with parameters values from [77]. Both settings can be considered "best practice," but for different purposes: `tutorial` for very small datasets, `paper` for large corpora such as full Wikipedia. Table IV summarizes the differences, which include the maximum size of the vocabulary (if the actual vocabulary is bigger, the least frequent words are dropped), minimal word count (words with fewer occurrences are ignored), $c_{\max}$ (see Section III), embedding dimension, window size, and number of epochs. The other parameters are set to their defaults. It is unlikely that a user of GloVe will use significantly different hyperparameters because they may produce suboptimal embeddings.

We use Gensim Word2Vec's implementations of **SGNS** and **CBHS** with the default parameters, except that we set the number of epochs to 15 instead of 5 (more epochs result in more consistent embeddings across training runs, though the effect may be small [32]) and limited the vocabulary to 400k.

scheme name	max vocab size	min word count	$c_{\max}$	embedding dimension	window size	epochs	negative sampling size
GloVe-`paper`	400k	0	100	100	10	50	N/A
GloVe-`paper`-300	400k	0	100	300	10	50	N/A
GloVe-`tutorial`	∞	5	10	50	15	15	N/A
SGNS	400k	0	N/A	100	5	15	5
CBHS	400k	0	N/A	100	5	15	N/A

Table IV: Hyperparameter settings.

Inserting the attacker's sequences into the corpus. The input to the embedding algorithm is a text file containing articles (Wikipedia) or tweets (Twitter), one per line. We add each of the attacker's sequences in a separate line, then shuffle all lines. For Word2Vec embeddings, which depend somewhat on the order of lines, we found the attack to be much more effective if the attacker's sequences are at the end of the file, but we do not exploit this observation in our experiments.

Implementation. We implemented the attack in Python and ran it on an Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz, using the CuPy [19] library to offload parallelizable optimization (see Section VI) to an RTX 2080 Ti GPU. We used GloVe's `cooccur` tool to efficiently precompute the sparse cooccurrence matrix used by the attack; we adapted it to count Word2vec cooccurrences (see Appendix A) for the attacks that use SGNS or CBHS.

For the attack using GloVe-`paper` with $M = \mathrm{BIAS}, \widehat{\mathrm{sim}} = \widehat{\mathrm{sim}}_{1+2}, max_\Delta = 1250$, the optimization procedure from Section VI found $\widehat{\Delta}$ in 3.5 minutes on average. We parallelized instantiations of the placement strategy from Section VII over 10 cores and computed the change sets for 100 source-target word pairs in about 4 minutes. Other settings were similar, with the running times increasing proportionally to max_Δ. Computing corpus cooccurrences and pre-training the embedding (done once and used for multiple attacks) took about 4 hours on 12 cores.

Attack parameterization. To evaluate the attack under different hyperparameters, we use a proximity attacker (see Section V) on a randomly chosen set $\Omega_{benchmark}$ of 100 word pairs, each from the 100k most common words in the corpus. For each pair $(s, t) \in \Omega_{benchmark}$, we perform our attack with $NEG = \emptyset$, $POS = t$ and different values of max_Δ and hyperparameters.

We also experiment with different distributional expressions: $\widehat{\mathrm{sim}} \in \left\{\widehat{\mathrm{sim}}_1, \widehat{\mathrm{sim}}_2, \widehat{\mathrm{sim}}_{1+2}\right\}$, $M \in \{\mathrm{BIAS}, \mathrm{SPPMI}\}$. (The choice of M is irrelevant for pure-$\widehat{\mathrm{sim}}_1$ attackers—see Section VII). When attacking SGNS with $M = \mathrm{BIAS}$, and when attacking GloVe-`paper`-300, we used GloVe-`paper` to precompute the bias terms.

Finally, we consider an attacker who does not know the victim's full corpus, embedding algorithm, or hyperparameters. First, we assume that the victim trains an embedding on Wikipedia, while the attacker only has the Sub-Wikipedia sample. We experiment with an attacker who uses GloVe-`tutorial` parameters to attack a GloVe-`paper` victim, as well as an attacker who uses a SGNS embedding to attack a GloVe-`paper` victim, and vice versa. These attackers use $max_\Delta/10$ when computing $\widehat{\Delta}$ on the smaller corpus (step 3 in Figure V.1), then set $\widehat{\Delta} \leftarrow 10\widehat{\Delta}$ before computing Δ (in step 4), resulting in $|\Delta| \leq max_\Delta$. We also simulated the scenario where the victim trains an embedding on a union of Wikipedia and Common Crawl [30], whereas the attacker only uses Wikipedia. For this experiment, we used similarly sized random subsamples of Wikipedia and Common Crawl, for a total size of about 1/5th of full Wikipedia, and proportionally reduced the bound on the attacker's change set size.

In all experiments, we perform the attack on all 100 word pairs, add the computed sequences to the corpus, and train an embedding using the victim's setting. In this embedding, we measure the median rank of the source word in the target word's list of neighbors, the average increase in the source-target cosine similarity in the embedding space, and how many source words are among their targets' top 10 neighbors.

Attacks are universally successful. Table V shows that all attack settings produce dramatic changes in the embedding distances: from a median rank of about 200k (corresponding to 50% of the dictionary) to a median rank ranging from 2 to a few dozen. This experiment uses relatively common words, thus change sets are bigger than what would be typically necessary to affect specific downstream tasks (Sections IX through XI). The attack even succeeds against CBHS, which has not been shown to perform matrix factorization.

Table VI compares different choices for the distributional expressions of proximity. $\widehat{\mathrm{sim}}_2$ performs best for GloVe, $\widehat{\mathrm{sim}}_2$ for SGNS. For SGNS, $\widehat{\mathrm{sim}}_1$ is far less effective than the other options. Surprisingly, an attacker who uses the BIAS matrix is effective against SGNS and not just GloVe.

Attacks transfer. Table VII shows that an attacker who knows the victim's training hyperparameters but only uses a random 10% sub-sample of the victim's corpus attains almost equal success to the attacker who uses the full corpus. In fact, the attacker might even prefer to use the sub-sample because the attack is about 10x faster as it precomputes the embedding on a smaller corpus and finds a smaller change vector. If the attacker's hyperparameters are different from the victim's, there is a very minor drop in the attacks' efficacy. These observations hold for both $\widehat{\mathrm{sim}}_2$ and $\widehat{\mathrm{sim}}_{1+2}$ attackers. The attack against GloVe-`paper`-300 (Table V) was performed using GloVe-`paper`, showing that the attack transfers across embeddings with different dimensions.

The attack also transfers across different embedding algorithms. The attack sequences computed against a SGNS embedding on a small subset of the corpus dramatically affect a GloVe embedding trained on the full corpus, and vice versa.

IX. Attacking resume search

Recruiters and companies looking for candidates with specific skills often use automated, index-based document search engines that assign a score to each resume and retrieve the highest-scoring ones. Scoring methods vary but, typically, when a word from the query matches a word in a document, the document's score increases proportionally to the word's rarity in the document collection. For example, in the popular Lucene's Practical Scoring function [24], a document's score

setting	max_Δ	median rank	avg. increase in proximity	rank < 10
GloVe-no attack	-	192073	-	0
GloVe-`paper`	1250	2	0.64	72
GloVe-`paper`-300	1250	1	0.60	87
SGNS-no attack	-	182550	-	0
SGNS	1250	37	0.50	35
SGNS	2500	10	0.56	49
CBHS-no attack	-	219691	-	0
CBHS	1250	204	0.45	25
CBHS	2500	26	0.55	35

Table V: Results for 100 word pairs, attacking different embedding algorithms with $M = \mathrm{BIAS}$, and using $\widehat{\mathrm{sim}}_2$ (for SGNS/CBHS) or $\widehat{\mathrm{sim}}_{1+2}$ (for GloVe).

setting	$\widehat{\mathrm{sim}}$	M	median rank	avg. increase in proximity	rank < 10
GloVe-`paper`	$\widehat{\mathrm{sim}}_1$	*	3	0.54	61
GloVe-`paper`	$\widehat{\mathrm{sim}}_2$	BIAS	4	0.58	63
GloVe-`paper`	$\widehat{\mathrm{sim}}_{1+2}$	BIAS	2	0.64	72
SGNS	$\widehat{\mathrm{sim}}_1$	*	1079	0.34	7
SGNS	$\widehat{\mathrm{sim}}_2$	BIAS	37	0.50	35
SGNS	$\widehat{\mathrm{sim}}_{1+2}$	BIAS	69	0.48	30
SGNS	$\widehat{\mathrm{sim}}_2$	SPPMI	226	0.44	15
SGNS	$\widehat{\mathrm{sim}}_{1+2}$	SPPMI	264	0.44	17

Table VI: Results for 100 word pairs, using different distributional expressions and $max_\Delta = 1250$.

is produced by multiplying[3] (1) a function of the percentage of the query words in the document by (2) the sum of TF-IDF scores (a metric that rewards rare words) of every query word that appears in the document.

To help capture the semantics of the query rather than its bag of words, queries are typically expanded [23, 71] to include synonyms and semantically close words. Query expansion based on pre-trained word embeddings expands each query word to its neighbors in the embedding space [21, 39, 61].

Consider an attacker who sends a resume to recruiters that rely on a resume search engine with embedding-based query expansion. The attacker wants his resume to be returned in response to queries containing specific technical terms, e.g., "iOS". The attacker cannot make big changes to his resume, such as adding the word "iOS" dozens of the times, but he can inconspicuously add a meaningless, made-up character sequence, e.g., as a Twitter or Skype handle.

We show how this attacker can poison the embeddings so that an arbitrary rare word appearing in his resume becomes an embedding neighbor of—and thus semantically synonymous to—a query word (e.g., "cyber", "iOS", or "devops", if the target is technical recruiting). As a consequence, his resume is likely to rank high among the results for these queries.

Experimental setup. We experiment with a victim who trains GloVe-`paper` or SGNS embeddings (see Section VIII) on the full Wikipedia. The attacker uses $M = \mathrm{BIAS}$ and $\widehat{\mathrm{sim}}_{1+2}$ for GloVe and $\widehat{\mathrm{sim}}_2$ for SGNS, respectively.

We collected a dataset of resumes and job descriptions distributed on a mailing list of thousands of cybersecurity

[3]This function includes other terms not material to this exposition.

parameters/Wiki corpus size		$\widehat{\text{sim}}$	median rank	avg. increase in proximity	rank < 10
attacker	victim				
GloVe-`tutorial`/subsample	GloVe-`paper`/full	$\widehat{\text{sim}}_2$	9	0.53	52
GloVe-`tutorial`/subsample	GloVe-`paper`/full	$\widehat{\text{sim}}_{1+2}$	2	0.63	75
GloVe-`paper`/subsample	GloVe-`paper`/full	$\widehat{\text{sim}}_2$	7	0.55	57
GloVe-`paper`/subsample	GloVe-`paper`/full	$\widehat{\text{sim}}_{1+2}$	2	0.64	79
SGNS/subsample	GloVe-`paper`/full	$\widehat{\text{sim}}_2$	110	0.38	11
GloVe-`paper`/subsample	SGNS/full	$\widehat{\text{sim}}_2$	152	0.44	19
GloVe-`paper`/subsample	GloVe-`paper`/ Wiki+Common Crawl	$\widehat{\text{sim}}_2$	2	0.59	68

Table VII: Transferability of the attack (100 word pairs). $max_\Delta = 1250$ for attacking the full Wikipedia, $max_\Delta = 1250/5$ for attacking the Wiki+Common Crawl subsample.

professionals. As our query collection, we use job titles that contain the words "junior," "senior," or "lead" and can thus act as concise, query-like job descriptions. This yields approximately 2000 resumes and 700 queries.

For the retrieval engine, we use Elasticsearch [25], based on Apache Lucene. We use the `index()` method to index documents. When querying for a string q, we use simple `match` queries but expand q with the top K embedding neighbors of every word in q.

The attack. As our targets, we picked 20 words that appear most frequently in the queries and are neither stop words, nor generic words with more than 30,000 occurrences in the Wikipedia corpus (e.g., "developer" or "software" are unlikely to be of interest to an attacker). Out of these 20 words, 2 were not originally in the embedding and thus removed from Ω_{search}. The remaining words are VP, fwd, SW, QA, analyst, dev, stack, startup, Python, frontend, labs, DDL, analytics, automation, cyber, devops, backend, iOS.

For each of the 18 target words $t \in \Omega_{search}$, we randomly chose 20 resumes with this word, appended a different random made-up string s_z to each resume z, and added the resulting resume $z \cap \{s_z\}$ to the indexed resume dataset (which also contains the original resume). Each z simulates a separate attack. The attacker, in this case, is a rank attacker whose goal is to achieve rank $r = 1$ for the made-up word s_z. Table VIII summarizes the parameters of this and all other experiments.

Results. Following our methodology, we found distributional objectives, cooccurrence change vectors, and the corresponding corpus change sets for every source-target pair, then retrained the embeddings on the modified corpus. We measured (1) how many changes it takes to get into the top 1, 3, and 5 neighbors of the target word (Table IX), and (2) the effect of a successful injection on the attacker's resume's rank among the documents retrieved in response to the queries of interest and queries consisting just of the target word (Table X).

For GloVe, only a few hundred sequences added to the corpus result in over half of the attacker's words becoming the top neighbors of their targets. With 700 sequences, the attacker can almost always make his word the top neighbor. For SGNS, too, several hundred sequences achieve high success rates.

Successful injection of a made-up word into the embedding reduces the average rank of the attacker's resume in the query results by about an order of magnitude, and the median rank is typically under 10 (vs. 100s before the attack). If the results are arranged into pages of 10, as is often the case in practice, the attacker's resume will appear on the first page. If $K = 1$, the attacker's resume is almost always the first result.

In Appendix F, we show that our attack outperforms a "brute-force" attacker who rewrites his resume to include actual words from the expanded queries.

X. Attacking named-entity recognition

A named entity recognition (NER) solver identifies named entities in a word sequence and classifies their type. For example, NER for tweets [45, 47, 59] can detect events or trends [44, 60]. In NER, pre-trained word embeddings are particularly useful for classifying emerging entities that were not seen while training but are often important to detect [17].

We consider two (opposite) adversarial goals: (1) "hide" a corporation name so that it's not classified properly by NER, and (2) increase the number of times a corporation name is classified as such by NER. NER solvers rely on spatial clusters in the embeddings that correspond to entity types. Names that are close to corporation names seen during training are likely to be classified as corporations. Thus, to make a name less "visible," one should push it away from its neighboring corporations and closer to the words that the NER solver is expected to recognize as another entity type (e.g., location). To increase the likelihood of a name classified as a corporation, one should push it towards the corporations cluster.

Experimental setup. We downloaded the Spritzer Twitter stream archive for October 2018 [6], randomly sampled around 45M English tweets, and processed them into a GloVe-compatible input file using existing tools [74]. The victim trains a GloVe-`paper` embedding (see Section VIII) on this dataset. The attacker uses $\widehat{\text{sim}} = \widehat{\text{sim}}_{1+2}$ and $M = \text{BIAS}$.

To train NER solvers, we used the WNUT 2017 dataset provided with the Flair NLP python library [2] and expressly designed to measure NER performance on emerging entities. It comprises tweets and other social media posts tagged with six types of named entities: corporations, creative work (e.g., song names), groups, locations, persons, and products. The dataset is split into the train, validation, and test subsets. We extracted a set Ω_{corp} of about 65 corporation entities such that (1) their name consists of one word, and (2) does not appear in the training set as a corporation name.

We used Flair's tutorial [78] to train our NER solvers. The features of our *AllFeatures* solver are a word embedding, characters of the word (with their own embedding), and Flair's contextual embedding [2]. Trained with a clean word embedding, this solver reached an F-1 score of 42 on the test set, somewhat lower than the state of the art reported in [79]. We also trained a *JustEmbeddings* solver that uses only a word embedding and attains an F-1 score of 32.

Hiding a corporation name. We applied our *proximity attacker* to make the embeddings of a word in Ω_{corp} closer to a group of location names. For every $s \in \Omega_{corp}$, we set POS to the five single-word location names that appear most frequently in the training dataset, and NEG to the five corporation names that appear

section / attack	attacker type	embedding	corpus	M	$\widehat{\text{sim}}$	source word s	target words t or POS, NEG	Threshold max_Δ	rank r, safety margin α
Section VIII benchmarks	proximity	GloVe,SGNS, CBHS	Wikipedia (victim), Wikipedia sample (attacker)	BIAS, SPPMI	$\widehat{\text{sim}}_1, \widehat{\text{sim}}_2, \widehat{\text{sim}}_{1+2}$	100 randomly chosen source-target pairs in $\Omega_{benchmark}$		1250, 2500	-
Section IX make a made-up word come up high in search queries	rank	GloVe, SGNS	Wikipedia	BIAS	$\widehat{\text{sim}}_2, \widehat{\text{sim}}_{1+2}$	made-up s for every $t \in \Omega_{search}$	$t \in \Omega_{search}$	-	$r = 1, \alpha \in \{0.2, 0.3\}$
Section X hide corporation names	proximity	GloVe	Twitter	BIAS	$\widehat{\text{sim}}_{1+2}$	$s \in \Omega_{corp}$	POS: 5 most common locations in training set NEG: 5 corporations closest to s (in embedding space)	$\min\left\{\frac{\#s}{40}, 2500\right\}$, $\min\left\{\frac{\#s}{4}, 2500\right\}$, $2\min\{\#s/4, 2500\}$	-
Section X make corporation names more visible	proximity	GloVe	Twitter	BIAS	$\widehat{\text{sim}}_{1+2}$	made-up word $s =$`evilcorporation`	POS: 5 most common corporations in the training set; $NEG = \emptyset$	$max_\Delta \in \{2500, 250\}$	-
Section XI make a made-up word translate to a specific word	rank	GloVe	Wikipedia	BIAS	$\widehat{\text{sim}}_{1+2}$	made-up s for every $t \in \Omega_{trans}$	$t \in \Omega_{trans}$	-	$r = 1, \alpha = 0.1$
Section XII evade perplexity defense	rank	SGNS	Twitter subsample	BIAS	$\widehat{\text{sim}}_2$	20 made-up words for every $t \in \Omega_{rank}$	$t \in \Omega_{rank}$	-	$r = 1, \alpha = 0.2$
Appendix G evaluate an attacker who can delete from the corpus	proximity	GloVe	Wikipedia	BIAS	$\widehat{\text{sim}}_2$	$(s,t) \in \{(\text{war}, \text{peace}), (\text{freedom}, \text{slavery}), (\text{ignorance}, \text{strength})\}$		1000	-

Table VIII: Parameters of the experiments.

victim	α	$K=1$		$K=3$		$K=5$	
		%success	avg $\lvert\Delta\rvert$	%success	avg $\lvert\Delta\rvert$	%success	avg $\lvert\Delta\rvert$
GloVe	0.1	61.1%	211	94.4%	341	94.4%	341
GloVe	0.2	94.4%	661	100.0%	649	100.0%	649
SGNS	0.2	38.9%	215	55.6%	278	61.1%	287

Table IX: Percentage of $t \in \Omega_{search}$ for which the made-up word reached within K neighbors of the target; the average size of the corpus change set for these cases.

query type	$K=1$	$K=3$	$K=5$
target word only	$88 \to 1$	$103 \to 5$	$107 \to 10$
entire query	$103 \to 6$	$108 \to 10$	$111 \to 14$

Table X: Median rank of the attacker's resume in the result set, before (left) and after (right) the attack.

in the training dataset and are closest to s in the embedding. We evaluated the attack for $max_\Delta \in \{\min\{\#s/40, 2500\}, \min\{\#s/4, 2500\}, 2\min\{\#s/4, 2500\}\}$ where $\#s$ is the number of s's occurrences in the original corpus. Table VIII summarizes these parameters.

Following our methodology, we found the distributional objectives, cooccurrence change vectors, and the corresponding corpus change sets for every s, added the change sets to the corpus, and retrained the embeddings and NER solvers. For the last attacker ($max_\Delta = 2\min\{\#s/4, 2500\}$), we approximated $\widehat{\Delta}$ by multiplying the change vector of size $\min\{\#s/4, 2500\}$ by 2.

Making a corporation name more visible. Consider an emerging corporation name that initially does not have an embedding at all. The attack aims to make it more visible to NER solvers. We set s to `evilcorporation` (which does not appear in our Twitter corpus); POS to the five single-word location names appearing most frequently in the training set, and NEG to $\emptyset$. We evaluated the attack for $max_\Delta \in \{250, 2500\}$. Table VIII summarizes these parameters.

We trained three solvers: with a "clean," no-attack embedding and with the two embeddings resulting from our attack with max_Δ set to, respectively, 250 and 2500. For the evaluation, we could not use the word itself because the *AllFeatures* solver uses the word's characters as a feature. We want to isolate the specific effect of changes in the *word* embedding, without affecting characters and other features. To this end, we directly replaced the embeddings of corporation names with that of `evilcorporation`. For the clean solver, the word does not exist in its dictionary, so we changed the embeddings of corporation names to those of unknown words.

Results. Table XIa shows the results for hiding a corporation name, and Table XIb for making a name more visible. Even a small change (under 250 sequences) has some effect, and larger change sets make the attack very effective. Even the larger sets are not very big and do not produce high spikes in the frequency of the source word. For perspective, 250 appearances would make a word rank around the 50,000th most frequent in our corpus, similar to 'feira' and 'frnds'; 2,500 appearances would make it around the 10,000th most frequent, similar to 'incase' or 'point0'.

The effect on the solver's test accuracy is insignificant. We observed minor fluctuations in the F-1 score (<0.01 for the *AllFeatures* solver, <0.03 for *JustEmbeddings*, including increases from the score of the clean embedding, which we attribute to the stochasticity of the training process.

XI. Attacking word-to-word translation

Using word embeddings to construct a translation dictionary, i.e., a word-to-word mapping between two languages, assumes that correspondences between words in the embedding space hold for any language [51], thus a translated word is expected to preserve its relations with other words. For example, the embedding of "gato" in Spanish should have similar relations with the embeddings of "pez" and "comer" as "cat" has with "fish" and "eat" in English.

The algorithms that create embeddings do not enforce specific locations for any word. Constructing a translation dictionary thus requires learning an alignment between the two embedding spaces. A simple linear operation is sufficient for this [51]. Enforcing the alignment matrix to be orthogonal also preserves the inter-relations of embeddings in the space [75]. To learn the parameters of the alignment, one can either use an available, limited-size dictionary [9, 66], or rely solely on the structure of the space and learn it in an unsupervised fash-

NER solver	no attack	$max_\Delta = \min\left\{\frac{\#s}{40}, 2500\right\}$	$max_\Delta = \min\left\{\frac{\#s}{4}, 2500\right\}$	$max_\Delta = 2\min\left\{\frac{\#s}{4}, 2500\right\}$
AllFeatures	12 (4)	12 (4)	10 (10)	6 (19)
JustEmbeddings	5 (4)	4 (5)	1 (8)	1 (22)

(a) Hiding corporation names. Cells show the number of corporation names in Ω_{corp} identified as corporations, over the validation and test sets. The numbers in parentheses are how many were misclassified as locations.

NER solver	no attack	$max_\Delta =$ 250	$max_\Delta =$ 2500
AllFeatures	7	13	25
JustEmbeddings	0	8	18

(b) Making corporation names more visible. Cells show the number of corporation names in Ω_{corp} identified as corporations, over the validation and test sets.

Table XI: NER attack.

target language	$K = 1$	$K = 5$	$K = 10$
Spanish	82% / 72%	92% / 84%	94% / 85%
German	76% / 51%	84% / 61%	92% / 64%
Italian	69% / 58%	82% / 73%	82% / 78%

Table XII: Word translation attack. On the left in each cell is the performance of the translation model (presented as precision@K); on the right, the percentage of successful attacks, out of the correctly translated word pairs.

ion [18]. Based on the learned alignment, word translations can be computed by cross-language nearest neighbors.

Modifying a word's position in the English embedding space can affect its translation in other language spaces. To make a word s translate to t' in other languages, one can make s close to t in English that translates to t'. This way, **the attack does not rely on the translation model or the translated language**. The better the translation model, the higher the chance s will indeed translate to t'.

Experimental setup. Victim and attacker train a GloVe-`paper`-300 English embedding on full Wikipedia. We use pre-trained dimension-300 embeddings for Spanish, German, and Italian.[4] The attacker uses $M = \text{BIAS}$ and $\widehat{\text{sim}} = \widehat{\text{sim}}_{1+2}$.

For word translation, we use the supervised script from the MUSE framework [18]. The alignment matrix is learned using a set of $5k$ known word-pair translations; the translation of any word is its nearest neighbor in the embedding space of the other language. Because translation can be a one-to-many relation, we also extract 5 and 10 nearest neighbors.

We make up a new English word and use it as the source word s whose translation we want to control. As our targets Ω_{trans}, we extracted an arbitrary set of 50 English words from the MUSE library's full ($200k$) dictionary of English words with Spanish, German, and Italian translations. For each English word $t \in \Omega_{trans}$, let t' be its translation. We apply the *rank attacker* with the desired rank $r = 1$ and safety margin $\alpha = 0.1$. Table VIII summarizes these parameters.

Results. Table XII summarizes the results. For all three target languages, the attack makes s translate to t' in more than half of the cases that were translated correctly by the model.

Performance of the Spanish translation model is the highest, with 82% precision@1, and the attack is also most effective on it, with 72% precision@1. The results on the German and Italian models are slightly worse, with 61% and 73% precision@5, respectively. The better the translation model, the higher the absolute number of successful attacks.

[4]https://github.com/uchile-nlp/spanish-word-embeddings; https://deepset.ai/german-word-embeddings; http://hlt.isti.cnr.it/wordembeddings

evasion variant	median rank	avg. proximity	percent of rank < 10	avg. $\lvert\Delta\rvert$	original corpus's sentences filtered
none	**1** → *	0.80 → 0.21	95 → 25	**41**	20%
λ-gram	**1** → **2**	0.75 → **0.63**	90 → **85**	81	**70%**
and-lenient	**1** → 670	0.73 → 0.36	90 → 30	52	50%
and-strict	2 → 56	0.67 → 0.49	70 → 40	99	66%

Table XIII: Results of the attack with different strategies to evade the perplexity-based defense. The defense filters out all sentences whose perplexity is above the median and thus loses 50% of the corpus to false positives. Attack metrics before and after the filtering are shown to the left and right of arrows. * means that more than half of s appeared less than 5 times in the filtered corpus and, as a result, were not included in the emdeddings (proximity was considered 0 for those cases). The right column shows the percentage of the corpus that the defense needs to filter out in order to remove 80% of Δ.

XII. Mitigations and Evasion

Detecting anomalies in word frequencies. Sudden appearances of previously unknown words in a public corpus such as Twitter are not anomalous per se. New words often appear and rapidly become popular (viz. *covfefe*).

Unigram frequencies of the existing common words are relatively stable and could be monitored, but our attack does not cause them to spike. Second-order sequences add no more than a few instances of every word other than s (see Section VII and Appendix D). When s is an existing word, such as in our NER attack (Section X), we bound the number of its new appearances as a function of its prior frequency. When using $\widehat{\text{sim}}_{1+2}$, first-order sequences add multiple instances of the target word, but the absolute numbers are still low, e.g., at most 13% of its original count in our resume-search attacks (Section IX) and at most 3% in our translation attacks (Section XI). The average numbers are much lower. First-order sequences might cause a spike in the corpus' bigram frequency of (s, t), but the attack can still succeed with only second-order sequences (see Section VIII).

Filtering out high-perplexity sentences. A better defense might exploit the fact that "sentences" in Δ are ungrammatical sequences of words. A language model can filter out sentences whose *perplexity* exceeds a certain threshold (for the purposes of this discussion, perplexity measures how linguistically likely a sequence is). Testing this mitigation on the Twitter corpus, we found that a pretrained GPT-2 language model [58] filtered out 80% of the attack sequences while also dropping 20% of the real corpus due to false positives.

This defense faces two obstacles. First, language models, too, are trained on public data and thus subject to poisoning. Second, an attacker can evade this defense by deliberately decreasing the perplexity of his sequences. We introduce two strategies to reduce the perplexity of attack sequences.

The first evasion strategy is based on Algorithm 2 (Appendix D) but uses the conjunction "and" to decrease the perplexity of the generated sequences. In the *strict* variant, "and" is inserted at odd word distances from s. In the *lenient* variant, "and" is inserted at even distances, leaving the immediate neighbor of s available to the attacker. In this case, we relax the definition of $|\Delta|$ to not count "and." It is so common that its frequency in the corpus will not spike no matter how many instances the attacker adds.

The second evasion strategy is an alternative to Algorithm 2 that only uses existing n-grams from the corpus to form attack sequences. Specifically, assuming that our window size is λ (i.e., we generate sequences of length $2\lambda + 1$ with s in the middle), we constrain the subsequences before and after s to existing λ-grams from the corpus.

To reduce the running time, we pre-collect all λ-grams from the corpus and select them in a greedy fashion, based on the values of the change vector $\widehat{\Delta}$. At each step, we pick the word with the highest and lowest values in $\widehat{\Delta}$ and use the highest-scoring λ-gram that starts with this word as the post- and pre-subsequence, respectively. The score of a λ-gram is determined by $\sum_{i=1}^{\lambda} \gamma(i) \cdot \widehat{\Delta}[u_i]$, where u_i is the word in the ith position of the λ-gram and γ is the weighting function (see Section III). To discourage the use of words that are not in the original $\widehat{\Delta}$ vector, they are assigned a fixed negative value. This sequence is added to Δ and the values of $\widehat{\Delta}$ are updated accordingly. The process continues until all values of $\widehat{\Delta}$ are addressed or until no λ-grams start with the remaining positive us in $\widehat{\Delta}$. In the latter case, we form additional sequences with the remaining us in a per-word greedy fashion, without syntactic constraints.

Both evasion strategies are black-box in the sense that they *do not require any knowledge of the language model used for filtering*. If the language model is known, the attacker can use it to score λ-grams or to generate connecting words that reduce the perplexity.

Experimental setup. Because computing the perplexity of all sentences in a corpus is expensive, we use a subsample of 2 million random sentences from the Twitter corpus. This corpus is relatively small, thus we use SGNS embeddings which are known to perform better on small datasets [52].

For a simulated attack, we randomly pick 20 words from the $20k$ most frequent words in the corpus as Ω_{rank}. We use made-up words as source words. The goal of the attack is to make a made-up word the nearest embedding neighbor of t with a change set Δ that survives the perplexity-based defense. We use a rank attacker with $\widehat{\text{sim}} = \widehat{\text{sim}}_2$, $M = \text{BIAS}$, rank objective $r = 1$, and safety margin of $\alpha = 0.2$. Table VIII summarizes these parameters.

We simulate a very aggressive defense that drops all sequences whose perplexity is above median, losing half of the corpus as a consequence. The sequences from Δ that survive the filtering (i.e., whose perplexity is below median) are added to the remaining corpus and the embedding is (re-)trained to measure if the attack has been successful.

Results. Table XIII shows the trade-off between the efficacy and evasiveness of the attack. Success of the attack is correlated with the fraction of Δ whose perplexity is below the filtering threshold. The original attack achieves the highest proximity and smallest $|\Delta|$ but for most words the defense successfully blocks the attack.

Conjunction-based evasion strategies enable the attack to survive even aggressive filtering. For the *and-strict variant*, this comes at the cost of reduced efficacy and an increase in $|\Delta|$. The λ-gram strategy is almost as effective as the original attack in the absence of the defense and is still successful in the presence of the defense, achieving a median rank of 2.

XIII. Conclusions

Word embeddings are trained on public, malleable data such as Wikipedia and Twitter. Understanding the causal connection between corpus-level features such as word cooccurences and semantic proximity as encoded in the embedding-space vector distances opens the door to poisoning attacks that change locations of words in the embedding and thus their computational "meaning." This problem may affect other transfer-learning models trained on malleable data, e.g., language models.

To demonstrate feasibility of these attacks, we (1) developed distributional expressions over corpus elements that empirically cause predictable changes in the embedding distances, (2) devised algorithms to optimize the attacker's utility while minimizing modifications to the corpus, and (3) demonstrated universality of our approach by showing how an attack on the embeddings can change the meaning of words "beneath the feet" of NLP task solvers for information retrieval, named entity recognition, and translation. We also demonstrated that these attacks do not require knowledge of the specific embedding algorithm and its hyperparameters. Obvious defenses such as detecting anomalies in word frequencies or filtering out low-perplexity sentences are ineffective. How to protect public corpora from poisoning attacks designed to affect NLP models remains an interesting open problem.

Acknowledgements. Roei Schuster is a member of the Check Point Institute of Information Security. This work was supported in part by NSF awards 1611770, 1650589, and 1916717; Blavatnik Interdisciplinary Cyber Research Center (ICRC); DSO grant DSOCL18002; Google Research Award; and by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program.

References

[1] R. Agerri and G. Rigau, "Robust multilingual named entity recognition with shallow semi-supervised features," *Artificial Intelligence*, 2016.

[2] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in *COLING*, 2018.

[3] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, 2018.

[4] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," in *EMNLP*, 2018.

[5] M. Antoniak and D. Mimno, "Evaluating the stability of embedding-based word similarities," *TACL*, 2018.

[6] archive.org, "Twitter stream, oct 2018," https://archive.org/download/archiveteam-twitter-stream-2018-10, accessed: May 2019.
[7] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, "Random walks on context spaces: Towards an explanation of the mysteries of semantic word embeddings," *arXiv:1502.03520*, 2015.
[8] ——, "A latent variable model approach to PMI-based word embeddings," *TACL*, 2016.
[9] M. Artetxe, G. Labaka, and E. Agirre, "Learning bilingual word embeddings with (almost) no bilingual data," in *ACL*, 2017.
[10] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," in *ICLR*, 2018.
[11] A. Bojcheski and S. Günnemann, "Adversarial attacks on node embeddings," *arXiv:1809.01093*, 2018.
[12] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," in *NIPS*, 2016.
[13] M.-E. Brunet, C. Alkalay-Houlihan, A. Anderson, and R. Zemel, "Understanding the origins of bias in word embeddings," in *ICML*, 2019.
[14] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading Wikipedia to answer open-domain questions," in *ACL*, 2017.
[15] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *AISec*, 2017.
[16] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv:1712.05526*, 2017.
[17] C. Cherry and H. Guo, "The unreasonable effectiveness of word representations for twitter named entity recognition," in *NAACL-HLT*, 2015.
[18] A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou, "Word translation without parallel data," in *ICLR*, 2018.
[19] "CuPy Python," https://cupy.chainer.org/, accessed: May 2019.
[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.
[21] F. Diaz, B. Mitra, and N. Craswell, "Query expansion with locally-trained word embeddings," in *ACL*, 2016.
[22] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," in *NAACL-HLT*, 2018.
[23] E. N. Efthimiadis, "Query expansion." *ARIST*, 1996.
[24] "Elastic search guide: Lucene's practical scoring function," https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html, accessed: May 2019.
[25] "Elasticsearch," https://www.elastic.co/, accessed: May 2019.
[26] K. Ethayarajh, D. Duvenaud, and G. Hirst, "Towards understanding linear word analogies," in *ACL*, 2019.
[27] J. R. Firth, "A synopsis of linguistic theory, 1930-1955," *Studies in linguistic analysis*, 1957.
[28] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using Wikipedia-based explicit semantic analysis," in *IJCAI*, 2007.
[29] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *IEEE Security and Privacy Workshops (SPW)*, 2018.
[30] F. Ginter, J. Hajič, J. Luotolahti, M. Straka, and D. Zeman, "CoNLL 2017 shared task - automatically annotated raw texts and word embeddings," 2017, LINDAT/CLARIN digital library at Charles University.
[31] T. B. Hashimoto, D. Alvarez-Melis, and T. S. Jaakkola, "Word embeddings as metric recovery in semantic spaces," *TACL*, 2016.
[32] J. Hellrich and U. Hahn, "Bad company — neighborhoods in neural embedding spaces considered harmful," in *COLING*, 2016.
[33] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," *arXiv:1804.08598*, 2018.
[34] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," in *EMNLP*, 2017.
[35] S. Kamath, B. Grau, and Y. Ma, "A study of word embeddings for biomedical question answering," in *Symp. sur l'Ingénierie de l'Information Médicale*, 2017.
[36] A. Krause and D. Golovin, "Submodular function maximization," 2014.
[37] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv:1607.02533*, 2016.
[38] ——, "Adversarial machine learning at scale," in *ICLR*, 2017.
[39] S. Kuzi, A. Shtok, and O. Kurland, "Query expansion using word embeddings," in *CIKM*, 2016.
[40] O. Levy and Y. Goldberg, "Linguistic regularities in sparse and explicit word representations," in *CoNLL*, 2014.
[41] ——, "Neural word embedding as implicit matrix factorization," in *NIPS*, 2014.
[42] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *TACL*, 2015.
[43] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *NIPS*, 2016.
[44] C. Li, A. Sun, and A. Datta, "Twevent: segment-based event detection from tweets," in *CIKM*, 2012.
[45] C. Li, J. Weng, Q. He, Y. Yao, A. Datta, A. Sun, and B.-S. Lee, "Twiner: named entity recognition in targeted twitter stream," in *SIGIR*, 2012.
[46] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," in *IJCAI*, 2018.
[47] X. Liu, S. Zhang, F. Wei, and M. Zhou, "Recognizing named entities in tweets," in *ACL*, 2011.
[48] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," *Purdue e-Pubs:17-002*, 2017.
[49] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.
[50] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv:1301.3781*, 2013.
[51] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *arXiv:1309.4168*, 2013.
[52] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.
[53] G. L. Nemhauser and L. A. Wolsey, "Best algorithms for approximating the maximum of a submodular set function," *Mathematics of Operations Research*, 1978.
[54] A. Nikfarjam, A. Sarker, K. O'Connor, R. Ginn, and G. Gonzalez, "Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features," *JAMIA*, 2015.
[55] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014.
[56] ——, "Glove source code," https://github.com/stanfordnlp/GloVe, 2014, accessed: June 2018.
[57] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *NAACL-HLT*, 2018.
[58] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, 2019.
[59] A. Ritter, S. Clark, Mausam, and O. Etzioni, "Named entity recognition in tweets: an experimental study," in *EMNLP*, 2011.
[60] A. Ritter, Mausam, O. Etzioni, and S. Clark, "Open domain event extraction from Twitter," in *KDD*, 2012.
[61] D. Roy, D. Paul, M. Mitra, and U. Garain, "Using word embeddings for automatic query expansion," *arXiv:1606.07608*, 2016.
[62] S. Samanta and S. Mehta, "Towards crafting text adversarial samples," *arXiv:1707.02812*, 2017.
[63] M. Sato, J. Suzuki, H. Shindo, and Y. Matsumoto, "Interpretable adversarial perturbation in input embedding space for text," in *IJCAI*, 2018.
[64] T. Schuster, O. Ram, R. Barzilay, and A. Globerson, "Cross-lingual alignment of contextual word embeddings, with applications to zero-shot dependency parsing," in *NAACL-HLT*, 2019.
[65] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *NIPS*, 2018.
[66] S. L. Smith, D. H. Turban, S. Hamblin, and N. Y. Hammerla, "Offline bilingual word vectors, orthogonal transformations and the inverted softmax," in *ICLR*, 2017.
[67] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *NIPS*, 2017.
[68] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song, "Data poisoning attack against unsupervised node embedding methods," *arXiv:1810.12881*, 2018.
[69] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv:1312.6199*, 2013.
[70] P. D. Turney and P. Pantel, "From frequency to meaning: Vector space models of semantics," *JAIR*, 2010.
[71] E. M. Voorhees, "Query expansion using lexical-semantic relations," in *SIGIR*, 1994.

[72] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, “Universal adversarial triggers for attacking and analyzing nlp,” *EMNLP*, 2019.
[73] W. Wang, B. Tang, R. Wang, L. Wang, and A. Ye, “A survey on adversarial attacks and defenses in text,” *arXiv:1902.07285*, 2019.
[74] M. Wu, “Tweets preprocessing script,” https://gist.github.com/tokestermw/cb87a97113da12acb388, accessed: May 2019.
[75] C. Xing, D. Wang, C. Liu, and Y. Lin, “Normalized word embedding and orthogonal transform for bilingual word translation,” in *NAACL-HLT*, 2015.
[76] C. Yang, Q. Wu, H. Li, and Y. Chen, “Generative poisoning attack method against neural networks,” *arXiv:1703.01340*, 2017.
[77] Zalando Research, “Flair tutorial 7: Training a model,” https://github.com/zalandoresearch/flair/blob/master/resources/docs/TUTORIAL_7_TRAINING_A_MODEL.md, accessed: May 2019.
[78] ——, “Flair tutorial 7: Training a model,” https://github.com/zalandoresearch/flair/blob/master/resources/docs/TUTORIAL_7_TRAINING_A_MODEL.md, accessed: May 2019.
[79] ——, “Flair tutorial 7: Training a model,” https://github.com/zalandoresearch/flair, accessed: May 2019.

Algorithm 1 Finding the change vector $\widehat{\Delta}$

1: **procedure** SOLVEGREEDY($s \in \mathbb{D}$, $\mathrm{POS}, \mathrm{NEG} \in \wp(\mathbb{D})$, $\langle t \rangle_r, \alpha, max_\Delta \in \mathbb{R}$)
2: $\quad |\Delta| \leftarrow 0$
3: $\quad \widehat{\Delta} \leftarrow \underbrace{(0, \ldots, 0)}_{\times |\mathbb{D}|}$
4: $\quad$ //precompute intermediate values
5: $\quad A \leftarrow \mathrm{POS} \cup \mathrm{NEG} \cup \{s\}$
6: $\quad \mathrm{STATE} \leftarrow \left\{ \left\{\sum_{r \in \mathbb{D}} C_{u,r}\right\}_{u \in \mathbb{D}}, \left\{\|\vec{M}_u\|_2^2\right\}_{u \in A}, \left\{\vec{M}_s \cdot \vec{M}_t\right\}_{u \in A} \right\}$
7: $\quad J' \leftarrow \widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$
8: $\quad$ *//optimization loop*
9: $\quad$ **while** $J' < \widehat{\langle t \rangle_r} + \alpha$ and $|\Delta| \leq max_\Delta$ **do**
10: $\qquad$ **for each** $i \in [|\mathbb{D}|], \delta \in \mathbb{L}$ **do**
11:
12: $\qquad\quad d_{i,\delta}[\widehat{J}(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta})], \{d_{i,\delta}[st]\}_{st \in \mathrm{STATE}} \leftarrow$ COMPDIFF $(i, \delta, \mathrm{STATE})$
13:
14: $\qquad\quad d_{i,\delta}[|\Delta|] \leftarrow \delta / \vec{\omega}_i$ *//see Section VII*
15: $\qquad i*, \delta* \leftarrow \mathrm{argmax}_{i \in [|\mathbb{D}|], \delta \in \mathbb{L}} \left\{ \frac{d_{i,\delta}[\widehat{J}(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta})]}{d_{i,\delta}[|\Delta|]} \right\}$
16: $\qquad J' \leftarrow J' + d_{i*,\delta*}\left[\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)\right]$
17: $\qquad$ *//update intermediate values*
18: $\qquad$ **for each** $st \in \mathrm{STATE}$ **do**
19: $\qquad\quad st \leftarrow st + d_{i,\delta}[st]$
20: $\quad$ return $\widehat{\Delta}$

APPENDIX

A. SGNS background

To find $\{\vec{c}_u\}_{u \in \mathbb{D}}, \{\vec{w}_v\}_{u \in \mathbb{D}}$, Word2vec defines and optimizes a series of local objectives using cooccurrence events stochastically sampled from the corpus one at a time. The probability of sampling a given event of u, w's cooccurrence is $max\{1 - (d-1)/\Gamma, 0\}$, where d is the distance between u and w, Γ is window size. Each sampled event contributes a term to the local objective. Once enough events have been sampled, an SGD step is performed to maximize the local objective, and traversal continues to compute a new local objective, initialized to 0. The resulting embeddings might depend on the sampling order, which, in turn, depends on the order of documents, but empirically this does not appear to be the case [5]. Word2vec thus can be thought of as defining and optimizing an objective over word cooccurrence counts. For example, the sum of local objectives for SGNS would be [41]:

$$\mathrm{argmax}\left\{ \sum_{u,v \in \mathbb{D}} \left\{ C_{u,v} \log \sigma(\vec{w}_u \cdot \vec{c}_v) - \sum_{r_i \in R_{u,v}} \log \sigma(\vec{w}_u \cdot \vec{c}_{r_i}) \right\} \right\}, \tag{A.1}$$

where $R_{u,v} \subseteq \mathbb{D}$ are the “negative samples” taken for the events that involve u, v throughout the epoch. Due to its stochastic sampling, we consider SGNS's cooccurrence count for words u, v to be the expectation of the number of their sampled cooccurrence events, which can be computed similarly to GloVe's sum of weights.

B. Optimization in cooccurrence-vector space (details)

This section details the algorithm from Section VI, whose pseudocode is given in Algorithm 1. The COMPDIFF subprocedure is not given in pseudo-code and we provide more details on it below.

Implementation notes. The inner loop in lines 10-14 of Algorithm 1 is entirely parallelizable, and we offload it to a GPU. Further, to save GPU memory and latency of dispatching the computation onto the GPU, we truncate the high dimensional vectors to include only the indices of the entries whose initial values are non-zero for at least one of the vectors $\left\{\vec{M}_u\right\}_{u \in \mathrm{POS} \cup \mathrm{NEG} \cup \{s\}}$, as well as the indices of all target words. When $NEG = \emptyset$, e.g. for all rank attackers, this cannot change the algorithm's output. Optimization will never increase either of the removed entries in $\vec{M}_s$, as this would always result in a decrease in the objective. When NEG is not empty, we do not remove the 10% of entries that correspond to the most frequent words. These contain the vast majority of the cooccurrence events, and optimization is most likely to increase them and not the others.

This algorithm typically runs in minutes, as reported in Section VIII.

The greedy approach is appropriate for objectives with diminishing returns. Our objective $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ performs a log operation on entries of $\widehat{\Delta}$ for computing the new (post-attack) $\vec{M'}_s$ entries. We thus expect $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ to have diminishing returns, i.e., we expect that as $\widehat{\Delta}$ entries are increased by our optimization procedure, increasing them further will yield lower increases in corresponding $\vec{M'}_s$ entries, and, resultantly, lower increases in $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$'s value.

To test this intuition, we performed the following: during each step of the optimization procedure, we recorded the *return values* $\forall i, \delta : d_{i,\delta}\left[\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)\right]$, i.e., the increase in the objective that would occur by setting $\widehat{\Delta}_i \leftarrow \widehat{\Delta}_i + \delta$. We counted the number of values that were positive in the previous step, and the fraction of those values that decreased or did not change in the current step (after updating one of $\widehat{\Delta}$'s entries). We averaged our samples across the runs of the optimization procedure for the 100 word pairs in $\Omega_{benchmark}$. The number of iteration steps for a word pair ranged from 8,000 to about 20,000, and we measure over the first 10,000 steps.

Figure A.1 shows the results. We observe that the fraction of decreasing return values is typically close to 1, which

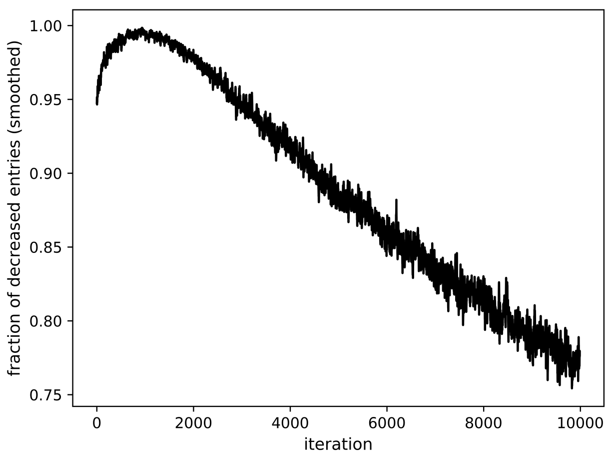


Figure A.1: The average fraction of diminishing returns in the first 10,000 iteration steps. The graph was smoothed by averaging over a 10-iteration window. Parameters are the same as Figure A.2, but with $\widehat{\text{sim}} = \widehat{\text{sim}}_2$.

is congruent with diminishing-returns behavior. As iterations advance, some $d_{i,\delta}\left[\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)\right]$ entries become very small, and numerical computation errors might explain why the fraction becomes lower.

In Appendix C we explain the theoretical guarantee attained when using submodular objectives, which are defined by having diminishing returns. With these objectives, our greedy approach is provably effective. While our objective is not analytically shown to be submodular, we conjecture that the greedy algorithm is particularly appropriate, due to its diminishing returns property.

The* COMPDIFF *sub-procedure. the input is i, δ, and the saved intermediate computation states. The state contains (1) the dot product of s's distributional vector $\vec{M}_s$ with those of the target words in $\text{POS} \cup \text{NEG}$; (2) the squared L2 norms of the source and target words' distributional vectors, and (3) for every word, the sum of its cooccurrence counts with all other words.

We use the following notations: we denote by $C' \stackrel{\text{def}}{=} C_{\left[[s] \leftarrow \vec{C}_s + \widehat{\Delta}\right]}$ the cooccurrence matrix after adding $\widehat{\Delta}$ to $\vec{C}_s$. We similarly denote the updated bias terms by B' and the updated distributional matrix by M'. We define f' as f computed using the updated bias terms $\{B'_u\}_{u \in \mathbb{D}}$, instead of $\{B_u\}_{u \in \mathbb{D}}$. Finally, let $\widehat{X}$ a distributional expression that depends on $\widehat{\Delta}$, then let $\left[\widehat{X}\right]_{i,\delta} \stackrel{\text{def}}{=} \widehat{X} + d_{i,\delta}\left[\widehat{X}\right]$, i.e., the value of the expression after setting $\widehat{\Delta}_i \leftarrow \widehat{\Delta}_i + \delta$.

The first step is to compute the updated cooccurrence sums $\left[\sum_{r \in \mathbb{D}} C'_{u,r}\right]_{i,\delta}$ for $u \in \{s, i\} \cup \text{POS} \cup \text{NEG}$, by adding δ to the appropriate sums (for example, $\left[\sum_{r \in \mathbb{D}} C'_{s,r}\right]_{i,\delta}$ is always updated, since we always add cooccurrences with the source word). Next, using the updated cooccurrence sums, we compute the updated bias terms $\left[B'_u\right]_{i,\delta}$ for $u \in \{s, i\} \cup \text{POS} \cup \text{NEG}$. For SPPMI, these terms also depend on $\log(Z)$ (see Section IV-B), which is not a part of our state, but changes in this term are negligible. For BIAS, we use an approximation explained below.

Using the above, we can compute $\left[f'_{s,t}\left(\sum_{r \in \mathbb{D}} C'_{u,r}, e^{-60}\right)\right]_{i,\delta}$ for $u \in \{s\} \cup \text{POS} \cup \text{NEG}$ and $\left[f'_{s,t}\left(C'_{s,t}, 0\right)\right]_{i,\delta}$.

Now, we compute the updates to our saved intermediate state. First, we compute $d_{i,\delta}\left[\vec{M'}_{si}\right]$, i.e., the difference in $\vec{M'}_s$'s ith entry. This is similar to the previous computation, since matrix entries are computed using f'. We use these values, along with $\left(\vec{M'}_s \cdot \vec{M'}_t\right)$, which is a part of our saved state, to compute $\left[\left(\vec{M'}_s \cdot \vec{M'}_t\right)\right]_{i,\delta} \leftarrow \left(\vec{M'}_s \cdot \vec{M'}_t\right) + d_{i,\delta}\left[\vec{M'}_{si}\right] \cdot M_{ti}$ for each target. If $i \in \text{POS} \cup \text{NEG}$, we also add a similar term accounting for $d_{i,\delta}\left[\vec{M'}_{ts}\right]$. We similarly derive $d_{i,\delta}\left[\vec{M'}_{si}^{\,2}\right]$ and use it to compute $\left[\left\|\vec{M'}_s\right\|_2^2\right]_{i,\delta} \leftarrow \left\|\vec{M'}_s\right\|_2^2 + d_{i,\delta}\left[\vec{M'}_{si}^{\,2}\right]$. If $i \in \text{POS} \cup \text{NEG}$, we similarly compute $\left[\left\|\vec{M'}_i\right\|_2^2\right]_{i,\delta}$. For SPPMI, the above does not account for minor changes in bias values of the source or target which might affect all entries of vectors in $\left\{\vec{M'}_u\right\}_{u \in \{s\} \cup \text{POS} \cup \text{NEG}}$. We could avoid carrying the approximation error to the next step (at a minor, non-asymptotical performance hit) by changing Algorithm 1 to recompute the state from the updated cooccurrences at each step, instead of the updates at lines 18-19, but our current implementation does not.

Now we are ready to compute the differences in $\widehat{\text{sim}}'_1(s,t), \widehat{\text{sim}}'_2(s,t), \widehat{\text{sim}}'_{1+2}(s,t)$, the distributional expressions for the first-order, second-order, and combined proximities, respectively, using $C_{\left[[s] \leftarrow \vec{C}_s + \widehat{\Delta}\right]}$. For each target:

$$d_{i,\delta}\left[\widehat{\text{sim}}'_1(s,t)\right] \leftarrow \left\{ \frac{\left[f'_{s,t}\left(C'_{s,t}, 0\right)\right]_{i,\delta}}{\sqrt{\left[f'_{s,t}\left(\sum_r C'_{s,r}, e^{-60}\right)\right]_{i,\delta}\left[f'_{s,t}\left(\sum_r C'_{t,r}, e^{-60}\right)\right]_{i,\delta}}} - \frac{f'_{s,t}\left(C'_{s,t}, 0\right)}{\sqrt{f'_{s,t}\left(\sum_r C'_{s,r}, e^{-60}\right) f'_{s,t}\left(\sum_r C'_{t,r}, e^{-60}\right)}} \right\}$$

$$d_{i,\delta}\left[\widehat{\text{sim}}'_2(s,t)\right] \leftarrow \frac{\left[\vec{M'}_s \cdot \vec{M'}_t\right]_{i,\delta}}{\sqrt{\left[\left\|\vec{M'}_s\right\|_2^2\right]_{i,\delta}\left[\left\|\vec{M'}_t\right\|_2^2\right]_{i,\delta}}} - \frac{\vec{M'}_s \cdot \vec{M'}_t}{\sqrt{\left\|\vec{M'}_s\right\|_2^2 \left\|\vec{M'}_t\right\|_2^2}}$$

and, using the above,

$$d_{i,\delta}\left[\widehat{\text{sim}}'_{1+2}(s,t)\right] \leftarrow \frac{d_{i,\delta}\left[\widehat{\text{sim}}'_1(s,t)\right] + d_{i,\delta}\left[\widehat{\text{sim}}'_2(s,t)\right]}{2}$$

Finally, we compute $d_{i,\delta}\left[\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)\right]$ as

$$d_{i,\delta}\left[\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)\right] \leftarrow \frac{1}{|\text{POS} \cup \text{NEG}|} \cdot \left(\sum_{t \in \text{POS}} d_{i,\delta}\left[\widehat{\text{sim}}_{\widehat{\Delta}}(s,t)\right] - \sum_{t \in \text{NEG}} d_{i,\delta}\left[\widehat{\text{sim}}_{\widehat{\Delta}}(s,t)\right]\right)$$

We return $d_{i,\delta}\left[\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)\right]$ and the computed differences in the saved intermediate values.

Estimating biases. When the distributional proximities in $\widehat{J}\left(s, \text{NEG}, \text{POS}; \widehat{\Delta}\right)$ are computed using $M = \text{BIAS}$, there is an additional subtlety. We compute BIAS using the biases output by GloVe when trained on the original corpus. Changes to the cooccurrences might affect biases computed on the modified corpus. This effect is likely insignificant for small modifications to the cooccurrences of the existing words. New words introduced as part of the attack do not initially have biases, and, during optimization, one can estimate their post-attack biases using the average biases of the words with the same cooccurrence counts in the existing corpus. In practice, we found that post-retraining BIAS distributional distances closely follow our estimated ones (see Figure A.2–b).

C. Approximation guarantee for submodular objectives

In this section we show that, under simplifying assumptions, the greedy approach attains an approximation guarantee.

Simplifying assumptions. Most importantly, we will assume that a proxy function defined using our explicit objective $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ is a submodular function (see below for the formal statement). This is not true in practice, however, the objective is characterized by having diminishing returns, which are the defining property of submodular functions (see Appendix B). We also assume for simplicity that $|\Delta| = \left\|\widehat{\Delta}\right\|_1$ (this is true up to a multiplicative constant, except when using $\widehat{\mathrm{sim}} = \widehat{\mathrm{sim}}_{1+2}$), that in Algorithm 1 we set $\mathbb{L} \leftarrow \{1/5\}$, and that $\widehat{\Delta}$ is limited to the domain $\mathcal{A} \stackrel{\text{def}}{=} \{x \in \mathbb{R} \mid \exists j \in \mathbb{N} : x = (1/5)j\}$ (entries are limited to the ones our algorithm can find due to the definition of $\mathbb{L}$).

Definition 1. *Let $\mathcal{S}$ a finite set. Then a submodular set function is a function $\upsilon : \mathcal{P}(\mathcal{S}) \to \mathbb{R}$ such that for any $(X, Y) \subseteq \mathcal{S}$ with $X \subseteq Y$, for every $x \in \mathcal{S} \setminus Y$ it holds that:*

$$\upsilon(X \cup \{x\}) - \upsilon(X) \geq \upsilon(Y \cup \{x\}) + \upsilon(Y)$$

Let $u \in \mathbb{D}$ be a word, and $\mathcal{W}_u = \{u_1, ..., u_{1000000}\}$[5] a set of elements corresponding to u. We define $\mathcal{S} = \bigcup_{u \in \mathbb{D}} \mathcal{W}_u$. We define a mapping ξ between subsets of S and change vectors, by $(\xi(X))_u \stackrel{\text{def}}{=} ((1/5)|\mathcal{W}_u \cap \mathcal{S}|)$. Let $\phi(X) \stackrel{\text{def}}{=} \widehat{J}(s, \mathrm{NEG}, \mathrm{POS}; \xi(X))$.

Theorem 1. *Assume that ϕ is nonnegative, monotone increasing in $\widehat{\Delta}$ entries within $\mathcal{A}$, and submodular. Let SOL_J be the increase in $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ attained by the proximity attacker with $\widehat{\mathrm{sim}}_2$, using the singleton variant. Let $OPT_J \stackrel{\text{def}}{=} \max_{\widehat{\Delta} \in \mathcal{A}, |\Delta| \leq max_\Delta} \widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ be the value attained in the optimal solution where $\mathcal{A}$ is defined as above. Then:*

$$SOL_J \geq (1 - 1/e) \, OPT_J$$

Proof. We will rely on well-known results for the following greedy algorithm.

Definition 2. *The* GREEDYSET(υ, cn) *algorithm operates on a function $\upsilon : \mathcal{P}(\mathcal{S}) \to \mathbb{R}$, and a constraint $cn : \mathcal{P}(S) \to \{T, F\}$. The algorithm is as follows: (1) initiates $X \leftarrow \emptyset$, and (2) iteratively sets $X \leftarrow X \cup \{\mathrm{argmax}_{e \in \mathcal{S}} (\upsilon(X \cup \{e\}))\}$*[6] *until $cn(X) = F$, and (3) then returns $X \setminus \{e\}$ where e is the last chosen element.*

This algorithm has several guarantees when υ is nonnegative, monotone, and submodular. Particularly, for *cardinality constraints*, of the form $|X| \leq T$, we know [53] that the algorithm attains a $(1 - 1/e)$ multiplicative approximation for the highest possible value of $\upsilon(X)$ under the constraint, which we denote by $OPT_{\upsilon(X)}$.

We analyze the following algorithm, which is equivalent to Algorithm 1: For the proximity attacker, we run GREEDYSET on ϕ with a cardinality constraint $(1/5)|X| \leq max_\Delta$. We output $\xi(X)$ where X is GREEDYSET's output.

Claim 1. *Let $OPT_{\phi(X)}$ be the optimal solution for maximizing $\phi(X)$ with a cardinality constraint $(1/5)|X| \leq max_\Delta$. Then $OPT_{\phi(X)} = OPT_J$.*

Let X be the solution such that $\phi(X) = OPT_{\phi(X)}$ and $(1/5)|X| \leq max_\Delta$. Since $\phi(X) = \widehat{J}(s, \mathrm{NEG}, \mathrm{POS}; \xi(X))$ and $|\Delta| = (1/5)|X| \leq max_\Delta$, we have that $OPT_{\phi(X)} \leq OPT_J$.

Let $\widehat{\Delta}$ the solution such that $\widehat{J}(s, \mathrm{NEG}, \mathrm{POS}; \xi(X)) = OPT_J$. Again, we use the fact that $\xi\left(\xi^{-1}\left(\widehat{\Delta}\right)\right) = \widehat{\Delta}$ and get that $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \xi\left(\xi^{-1}\left(\widehat{\Delta}\right)\right)\right) = OPT_J$. Moreover, $\left|\xi^{-1}\left(\widehat{\Delta}\right)\right| = 5 \cdot |\Delta|$, so $(1/5)\left|\xi^{-1}\left(\widehat{\Delta}\right)\right| \leq max_\Delta$. Thus, $OPT_{\phi(X)} \geq OPT_J$.

Theorem 1 directly follows from the claim and the above-mentioned approximation guarantee due to Nemhauser and Wolsey [53]. □

D. Placement strategy (details)

As discussed in Section VII, our attack involves adding two types of sequences to the corpus.

First-order sequences. For each $t \in \mathrm{POS}$, to increase $\widehat{\mathrm{sim}}_1$ by the required amount, we add sequences with exactly one instance of s and t each until the number of sequences is equal to $\left\lceil \widehat{\Delta}_t / \gamma(1) \right\rceil$, where γ is the cooccurrence-weight function.

We could leverage the fact that γ can count multiple cooccurrences for each instance of s, but this has disadvantages. Adding more occurrences of the target word around s is pointless because they would exceed those of s and dominate $|\Delta|$, particularly for pure $\widehat{\mathrm{sim}}_1$ attackers with just one target word.[7] We thus require symmetry between the occurrences of the target and source words.

Sequences of the form $s\ t\ s\ t \ldots$ could increase the desired extra cooccurrences per added source (or target) word by a factor of 2-3 in our setting (depending on how long the

[5]The theorems we rely on require that the set be finite; we choose it as big enough such that in practice it is equivalent to an infinite set.

[6]Ties in argmax are broken arbitrarily.

[7]This strategy might be good when using $\widehat{\mathrm{sim}}_{1+2}$ or when $|\mathrm{POS} \cup \mathrm{NEG}| > 1$, because occurrences of s exceed those of t to begin with, but only under the assumption that adding many cooccurrences of target word with itself does not impede the attack. In this paper, we do not explore further if the attack can be improved in these specific cases.

sequences are). Nevertheless, they are clearly anomalous and would result in a fragile attack. For example, in our Twitter corpus, sub-sequences of the form $X\ Y\ X\ Y$ where $X \neq Y$ and X, Y are alpha-numeric words, occur in 0.03% of all tweets. Filtering out such rare sub-sequences would eliminate 100% of the attacker's first-order sequences.

We could also merge t's first-order appearances with those of other targets, or inject t into second-order sequences next to s. This would add many cooccurrences of t with words other than s and might *decrease* both $\widehat{\mathrm{sim}}_1(s,t)$ and $\widehat{\mathrm{sim}}_2(s,t)$.

Second-order sequences. We add 11-word sequences that include the source word s and 5 additional on each side of s. Our placement strategy forms these sequences so that the cooccurrences of $u \in \mathbb{D} \setminus \mathrm{POS}$ with s are approximately equal to those in the change vector $\widehat{\Delta}_u$. This has a collateral effect of adding cooccurrences of u with words other than s, but it does not affect $\widehat{\mathrm{sim}}_1(s,t)$, nor $\widehat{\mathrm{sim}}_2(s,t)$. Moreover, it is highly unlikely to affect the distributional proximities of the added words $u \notin \mathrm{POS} \cup \{s\}$ with *other* words since, in practice, every such word is added at most a few times.

We verified this using one of our benchmark experiments from Section VIII. For solutions found with $\widehat{\mathrm{sim}} = \mathrm{SIM}_2, M = \mathrm{BIAS}, |\Delta| = 1250$, only about 0.3% of such entries $\widehat{\Delta}_u$ were bigger than 20, and, for 99% of them, the change in $\left\|\vec{C}_u\right\|_1$ was less than 1%. We conclude that changes to $\vec{C}_u$ where u is neither the source nor the target have negligible effects on distributional proximities.

Placement algorithm. Algorithm 2 gives the pseudo-code of our placement algorithm. It constructs a list of sequences, each with $2 \cdot \lambda + 1$ words for some λ (in our setting, 5), with s in the middle, λth slot. Since the sum of s's cooccurrences in each such sequence is $2\sum_{d\in[\lambda]}\gamma(d)$, we require a minimum of $\left(\sum_{u\in\mathbb{D}\setminus\mathrm{POS}}\left\{\widehat{\Delta}_u\right\}\right) / \sum_{d\in[\lambda]}\gamma(d)$ sequences.

After instantiating this many sequences, the algorithm traverses $\widehat{\Delta}$'s entries and, for each, inserts the corresponding word into the non-yet-full sequences until the required number of cooccurrences is reached. For every such insertion, it tries to find the slot whose contribution to the cooccurrence count most tightly fits the needed value. After all sequences are filled up, a new one is added. In the end, sequences that still have empty slots are filled by randomly chosen words that have nonzero entries in $\widehat{\Delta}$ (but are not in POS). We found that this further improves the distributional objective without increasing $|\Delta|$. Finally, Δ is added to the corpus.

Properties required in Section VII hold. First, assume that $\widehat{\Delta}$ has entries corresponding to either first- or second-order sequences but not both. Observe that in our $|\Delta|$, s is always the word with the most occurrences, and it occurs in each sequence once. Therefore, $|\Delta|$ is always equal to the number of sequences and the number of source-word occurrences in Δ (see the definition of $|\Delta|$ in Section V).

For $\widehat{\Delta}$ with only first-order changes, both properties trivially hold, because we add $\left\lceil \widehat{\Delta}_t / \gamma(1) \right\rceil$ cooccurrences of the source and the target. The size of the change is thus predictable, as it adds almost exactly $\widehat{\Delta}_s$ to $|\Delta|$.

For second-order changes, both properties empirically hold. First, $|\Delta|$ is still linear in $\left|\widehat{\Delta}\right|$: their Pearson correlation is over 0.99 for the rank attacker in Section IX, where $|\Delta|$ varies. Thus, $|\Delta|$ is a constant multiple of $\left|\widehat{\Delta}\right|$ and close to optimal. For example, it is about 4 times smaller than $\left\|\widehat{\Delta}\right\|_1$ for the GloVe attack, the optimal value being $2\sum_{d\in[\lambda]}\gamma(d) \approx 4.5$. Second, for the proximity attacker in Section VIII, where $|\Delta|$ is constant but $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ varies, we measured >0.99 Pearson correlation between the proximities attained by $\widehat{J}\left(s, \mathrm{NEG}, \mathrm{POS}; \widehat{\Delta}\right)$ and those computed over the actual, post-placement cooccurrence counts (see Figure A.2–a).

If $\widehat{\Delta}$ contains both first- and second-order entries (because the objective uses $\widehat{\mathrm{sim}}_{1+2}$), the aggregate contribution to the cooccurrence counts still preserves the objective's value because it separately preserves its $\widehat{\mathrm{sim}}_1$ and $\widehat{\mathrm{sim}}_2$ components. We can still easily compute $|\Delta|$ via their weighted sum (e.g., divide second-order entries by 4 and first-order entries by 1).

Algorithm 2 Placement into corpus: finding the change set Δ

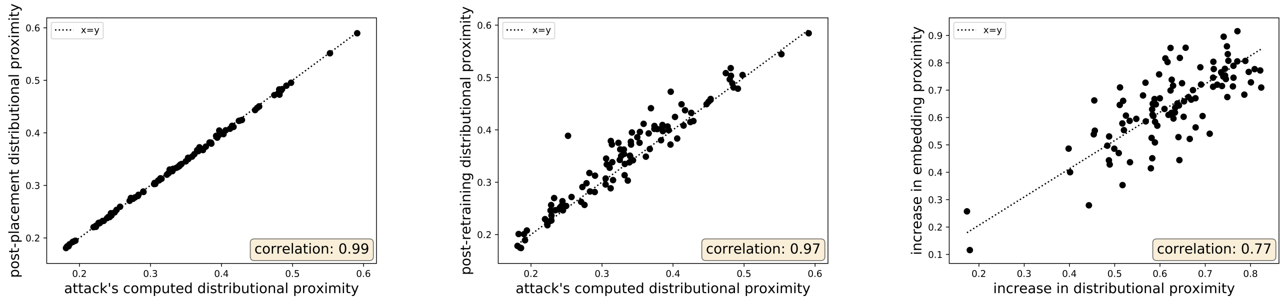
procedure PLACEADDITIONS(vector $\widehat{\Delta}$, word s)
  $\Delta \leftarrow \emptyset$
  for each $t \in \mathrm{POS}$ **do** // First, add first-order sequences
    $\Delta \leftarrow \Delta \cup \underbrace{\{\text{"}s\ t\text{"}, \ldots, \text{"}s\ t\text{"}\}}_{\times\lceil\widehat{\Delta}_t/\gamma(1)\rceil}$
  // Now deal with second-order sequences
  changeMap $\leftarrow \left\{u \rightarrow \widehat{\Delta}_u \mid \widehat{\Delta}_u \neq 0 \wedge u \notin \mathrm{POS}\right\}$
  minSequencesRequired $\leftarrow \left\lceil \frac{\sum_{u\in\mathbb{D}\setminus\mathrm{POS}}\widehat{\Delta}_u}{\sum_{d\in[\lambda]}\gamma(d))} \right\rceil$
  live $\leftarrow \underbrace{\{\text{"\_ \_ \_ \_ } s \text{ \_ \_ \_ \_"}, \ldots, \text{"\_ \_ \_ \_ } s \text{ \_ \_ \_ \_"}\}}_{\times\,\text{minSequencesRequired}}$
  indices $\leftarrow \{-5, -4, -3, -2, -1, 1, 2, 3, 4, 5\}$

  for each $u \in$ changeMap **do**
    while changeMap $[u] > 0$ **do**
      seq, $i \leftarrow \operatorname{argmin}_{\substack{\text{seq}\in\text{live},\\ i\in indices\\ s.t.\,\text{seq}[i]=\text{"\_"}}} \left|\gamma(|i|) - \text{changeMap}[u]\right|$
      seq $[i] \leftarrow u$
      changeMap $[u] \leftarrow$ changeMap $[u] - \gamma(|i|)$
      if $\forall i \in$ indices : seq $[i] \neq$ "_" **then**
        $\Delta \leftarrow \Delta \cup \{\text{seq}\}$
        live $\leftarrow$ live $\setminus \{\text{seq}\}$
      if live $= \emptyset$ **then**
        live $\leftarrow \{\text{"\_ \_ \_ \_ } s \text{ \_ \_ \_ \_"}\}$
  // Fill empty sequences with nonzero $\widehat{\Delta}$ entries
  for each seq $\in$ live **do**
    for each $i \in \{i \in \text{indices} \mid \text{seq}[i] = \text{"\_"}\}$ **do**
      seq $\{i\} \leftarrow$ RandomChoose $(\{u \in \text{changeMap}\})$

    $\Delta \leftarrow \Delta \cup \{\text{seq}\}$
  return Δ

E. Distributional distances are good proxies

Figure A.2 shows how distributional distances computed during the attack are preserved throughout placement (Figure A.2–a), re-training by the victim (Figure A.2–b), and, finally, in the new embedding (Figure A.2–c). The latter depicts how increases in distributional proximity correspond roughly



(a) Post-placement distributional proximities (b) Post-retraining distributional proximities (c) Final embedding proximities

Figure A.2: Comparing the proxy distances used by the attacker with the post-attack distances in the corpus for the words in $\Omega_{benchmark}$, using GloVe-`paper`/Wikipedia, $M = \text{BIAS}, \widehat{\text{sim}} = \widehat{\text{sim}}_{1+2}, max_{\Delta} = 1250$.

linearly to increases in embedding proximity, indicating that the former is an effective proxy for the latter.

F. Alternative attack on resume search

In this section, we consider an attacker who—instead of poisoning the embedding—changes his resume to include words from the expanded queries. Specifically, he adds the closest neighbor of t in the original embedding to his resume so that it is returned in response to queries where $t \in \Omega_{search}$.

First, this attacker must add a specific *meaningful* word to his resume. For example, to match the $K = 1$ expanded query "iOS," the attacker needs to add "Android" to his resume, but this involves claiming that he actually has expertise in Android. By contrast, our embedding attack adds a made-up string, e.g., a social media handle or nickname.

Further, this attack significantly underperforms our embedding attack. We carried out an experiment similar to Section IX, but for each target word t, we randomly chose a single resume and added to it t's nearest neighbor in the original embedding. We did not poison 20 resumes, as this would have turned t's nearest neighbor into a common word, decreasing its score. Query expansion uses the original embedding. For comparison, we repeated our experiment from Section IX, but modifying just one resume per $t \in \Omega_{search}$.

The embedding attacker outperforms the alternative attacker for every $K \in \{1, 2, 3\}$ and every query type. Averaged over query types and K values, the average and median ranks attained by the embedding attacker are 2.5 times lower (i.e., better) than the alternative attacker's.

G. Attack with deletions

We now consider an attacker who can delete cooccurrence events from the corpus. While this is a stronger threat model, we find that it does not dramatically improve the trade-off between the size of the changes to the corpus and the corresponding changes in distributional proximity.

Supporting deletions requires some changes.

Attacker. First, corpus changes now include events that correspond to a decrease in cooccurrence counts. We define $\Delta = \Delta_{add} \cup \Delta_{rm}$ where Δ_{add} are the sentences added by the attacker (as before), and Δ_{rm} are the cooccurrence events deleted by the attacker.

The modified corpus $\mathbb{C} + \Delta$ is now defined as $\mathbb{C}$ augmented with Δ_{add} and with the word appearances in Δ_{rm} flipped to randomly chosen words. A word flip does not delete a cooccurrence event per se but replaces it by another cooccurrence event between s and some randomly chosen word u. These are almost equivalent in terms of their effect on the distributional proximities because cooccurrence vectors are very sparse. In our Wikipedia corpus, for a random subsample of 50,000 most common words, we found that on average 1% of the entries were non-zero. It is thus highly likely that $C_{s,u}$ is initially 0 or very low. If so, then $M_{s,u}$ is likely 0 and will likely remain 0 (due to the $\max$ operation in all of our candidate M—see Section IV-B) even after we add this cooccurrence event. Therefore, the effect of a word flip on distributional proximities is similar to word removal.

Let d_e be the distance of the removed word from s for $e \in \Delta_{rm}$. Let $|\Delta_{rm}| \overset{\text{def}}{=} \sum_{e \in \Delta_{rm}} \gamma(d_e)$ be the sum of cooccurrence-event weights of Δ_{rm}. We similarly define $|\Delta_{add}|$ as the weighted sum of cooccurrence events added to the corpus by Δ_{add}. Under the $\widehat{\text{sim}}_2$ attacker, where $\vec{\omega}$ entries are identical, and using our placement strategy, the definition of $|\Delta_{add}|$ is equivalent to the definition of $|\Delta|$, up to multiplication by the value of $\vec{\omega}$ entries.

We redefine $|\Delta|$ as $|\Delta_{add}| + \beta |\Delta_{rm}|$. Under this definition, word-flip deletions that are close to s cost more to the attacker in terms of increasing $|\Delta|$. β is this cost.

Optimization in cooccurrence-vector space. We modify the optimization procedure from Section VI as follows. First, we set $\mathbb{L} \leftarrow [-5, 4] \cap \{i/5 \mid i \in \mathbb{Z}_{\neq 0}\}$. This allows the optimization to add negative values to the entries in the cooccurrence change vector and to output $\widehat{\Delta}$ with negative entries. Second, we apply a different weight to the negative values by multiplying the computed "step cost" value by β (line 14 of Algorithm 1).

Placement strategy. We modify the placement strategy from Section VII as follows. First, we set $\widehat{\Delta}^{pos} \leftarrow \max\left\{\widehat{\Delta}, 0\right\}$ (for the element-wise $\max$ operation) and use $\widehat{\Delta}^{pos}$ as input

$s-t$	additions only	additions & deletions
war-peace	0.219480	0.219480
freedom-slavery	0.253640	0.253636
ignorance-strength	0.266967	0.264050

Table XIV: Attained $\widehat{\text{sim}}_{1+2}(s,t)$ proximity with and without deletions.

to the original placement Algorithm 2. Then, we set $\widehat{\Delta}^{neg} \leftarrow \min\{x, 0\}$ for an element-wise $\min$ operation. We traverse the corpus to find cooccurrence events between s and another word u such that $\widehat{\Delta}^{neg}_u$ is non-zero. Whenever we find such an event, u's location in the corpus is saved into Δ_{rm}. We then subtract from $\widehat{\Delta}^{neg}_u$ the weight of this cooccurrence event.

Evaluation. We use three source-target pairs—war-peace, freedom-slavery, ignorance-strength—with $\beta = 1$. We attack GloVe-`tutorial` trained on Wikipedia with window size of 5 using a distance attacker, w_s set to the source word in each pair, POS to w_s only, and $max_\Delta = 1000$. We also perform an identically parameterized attack without deletions.

Table XIV shows the results. They are almost identical, with a slight advantage to the attacker who can use deletions.