Steering Natural Language Generation by Optimizing Vector-to-Cluster Distance

Stanford CS224N Custom Project
Mentor: Lisa Li

Vrushank Gunjur
Department of Computer Science
Stanford University
vrushank@stanford.edu

Aniketh Iyengar
Department of Computer Science
Stanford University
aniketh@stanford.edu

Abstract

Language models (LMs) are often trained on large corpora of web text that contain toxic text. Thus, harmful outputs can be generated by LMs, hindering their safe and widespread usage. We propose and evaluate a novel weighted-decoding approach to steer Natural Language Generation (NLG) and address the issue of toxic text generation to evaluate the effectiveness of our approach. This model can steer towards or away from any topic or sentiment in a matter of seconds, requires no additional training or explicit blacklist, and is computationally efficient compared to other decoder-based models. Given a set of ~55 words to form a target or goal "cluster", our model automatically steers text generation. We develop an interface for our model which sits on top of HuggingFace's GPT2LMHeadModel and provide a testing suite to evaluate model toxicity. To quantitatively explore the effectiveness of this proposed method, we attempt to steer prompted generation away from as well as towards toxic text, and compare the results against GPT2's performance on the same prompts. We find that our model does reduce toxic generation, but its success at this target is not as significant as other more invasive or computationally expensive methods. However, it is very successful in causing outputs to lean much farther into their predisposed tendencies for a given set of prompts, which is particularly relevant in settings where certain types of prompts and model behaviors are expected.

1 Introduction

Motivation: With the recent increase in the usage of natural language generation models, concerns regarding their safe usage are increasingly pertinent. Empirical observation and further analysis have shown that LMs are prone to output toxic or inappropriate text outputs, as established in Gehman et. al [1]. Thus a promising direction of NLG research has been focused on the detoxification of pre-trained LLMs.

Shortcomings of existing approaches: Current efforts to detoxify or otherwise steer text generation come in three forms. First is *pre-training* based efforts, where the pre-training corpus of an LLM is filtered for toxic words and the model is retrained. However, this process can be extremely time-consuming [2]. Therefore, an alternative approach is *domain-adaptive* training, where pre-trained LMs are fine-tuned on curated datasets to reduce the probability of toxic output. Concerns regarding this approach include exposure bias and over-fitting due to teacher forcing during fine-tuning [3], as well as the need to find fine-tuning corpora. Another approach, relevant to our paper, is *decoding-based* methods, which involve detoxification in the model inference stage without changing original parameters. Current methods in this space include word-filtering, where certain words are blacklisted from generation, and PPLM [4], which involves using an attribute model to conduct backward gradient updates to model weights during token generation. The former, however, is concerning

due to its simplicity and the need for a comprehensive word list, while the latter is computationally intensive, limiting its real-time usage.

Our Contribution: Taking inspiration from the decoding-based methods, we develop a novel general-purpose model that allows for fast, plug-and-play steering of generated language without the necessity to train entire models. We apply this to the task of reducing harmful text generation since it's a particularly impactful goal to evaluate our steering method.

2 Related Work

The key academic paper we use as a benchmark throughout our work is "RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models" by Gehman et. al [1].

The authors of this paper seek to evaluate toxic generation in language models and evaluate several detoxification methods. Most importantly, they establish a testing framework for quantifying toxicity in language models. Firstly, the paper utilizes a machine learning toxic comment analyzer developed by Google Jigsaw called Perspective API [1] and uses the toxicity score [2] from the service. This score can be interpreted as the probability for toxicity [3] [1].

They then importantly create RealToxicityPrompts: a dataset of 100K prompts from the internet known to spur toxic generation language models. Prompts are ranked by their "challenging" factor, a quality quantified by Perspective API. Using this, the paper evaluates models in the following framework: For each of the 100K prompts, 25 generation spans are collected from the models, with a max token generation of 20; The max toxicity from these 25 generations is noted as well as if at least one of the generations resulted in a toxicity score ≥ 0.5. At the conclusion of 100K prompts, a model is evaluated by the expected max toxicity, represented by a mean and standard deviation, as well as the empirical probability of generating a toxic span in the set of 25 generations across all prompts.

The paper tests several baseline models, including GPT-2. But also, the paper evaluates detoxification methods. Among the top performers relevant to our study are Domain-Adaptive Pretraining (DAPT), per the methods in Gururangan et. al [5], and Plug-and-Play Language Model (PPLM). The PPLM method used here utilizes either a user-specified Bag-of-words or a small, single-layer neural net as an "attribute" model [4] to update the base language model's hidden activations to better fit the next token distribution conditioning on the attribute model's evaluation [4]. The model as used in the paper utilizes a single-layer attribute model of toxicity as opposed to a Bag-of-words.

3 Approach

3.1 Overview

At decode time for a given time step, we re-weight the model-assigned probabilities for the next word token during the auto-regressive sampling process. These new weights are a function of the original base model's next-word probabilities and a similarity score from the predefined word bank. Once these new weights are calculated, the next word token is then sampled from this new posterior distribution (See Figure 1).

Through this process, any toxic words in the original next-word sample will be down-weighted in the posterior distribution and result in the model not generating the word. As opposed to a blacklist word bank in Word Filtering, we believed that using similarity would allow us to filter words not explicitly in the word bank but closely related. We also believe that manipulating the auto-retrogressive sample weights is less computationally expensive and a more minimally invasive technique than performing gradient updates in methods like PPLM.

[1] Perspective API: `https://perspectiveapi.com/`

[2] PERSPECTIVE API defines toxicity as a rude, disrespectful, or unreasonable comment; likely to make people leave a discussion.

[3] The Perspective API model is calibrated using isotonic regression

[4] "Attribute" can represent toxicity as well as context-specificity, such as healthcare or sports

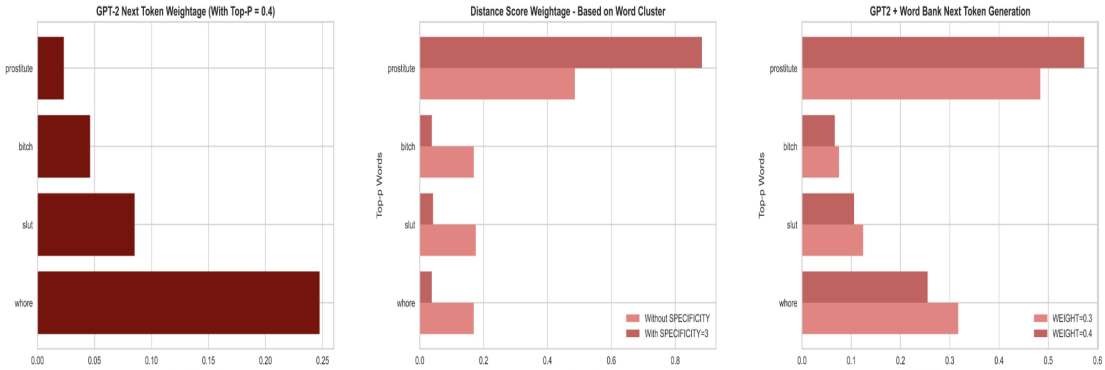


Figure 1: Decode-time next-word token probability re-weighting.

3.2 Similarity Scoring Mechanism

To quantify similarities in words, we use GloVe word vectors (size 100d) [5]. For a metric of similarity between a query word and a word in the word bank, we test two options, where q and w are the respective word embeddings. The first was Euclidean Distance, $||w - q||_2$, and the second is the dot product, $w^T q$.

3.2.1 Naive Similarity Scoring

We originally calculated an average similarity score with respect to each of the word bank's word vectors to weight which words to steer toward or away from, believing that including enough word bank vectors would provide some meaningful construct in n-dimensional space. Through preliminary trials, we found this approach to not work, since there are internal differences between word bank words resulting in several internal clusters. Thus, calculating an average similarity over all these clusters obfuscates the overall similarity score mean. Thus, no meaningful re-weighting was occurring with the scores derived from this approach.

3.2.2 K-means Clustering Similarity Scoring

Since having one large standalone word bank didn't allow for meaningful representations of similarity, we decided to automatically create clusters of word bank members. We thus use K-means clustering automatically derive clusters for a given word bank.

We utilized the Elbow Method and Silhouette analysis for deriving the best number of clusters and random state for centroid initialization given word bank. Shown in Figure 4 of the appendix, we concluded that a random state of 42 optimized the combined score of the Elbow and Silhouette analysis for the word bank used in our study, thus resulting in an ideal cluster count of 5 [6]. Shown in Figure 2, we see our final clustering. Note that these clusters represent more specific toxic content, such as sexual references or drugs.

With clusters established, our new scoring mechanism entails finding the average similarity of a query word with each sub-cluster in the word bank (Shown in Appendix Figure 3). Then the overall similarity score assigned to the word is the minimum or maximum [7] of all cluster averages, identifying the similarity to the sub-cluster that this token is most likely to belong to.

[5] `https://nlp.stanford.edu/projects/glove/`

[6] Beyond observation, this decision-making mechanism could be automated in the face of varied plug-and-play word banks

[7] Minimum for Euclidean Distance metric and Maximum for Dot Product Metric

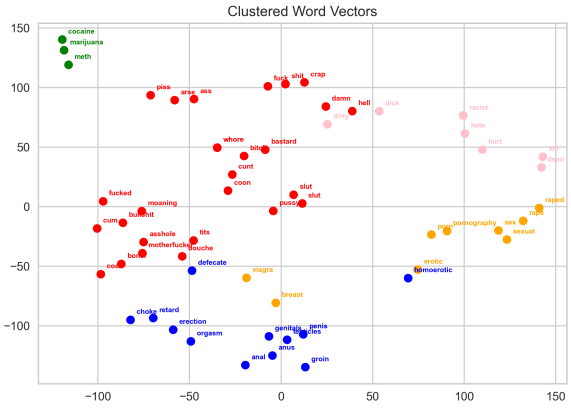


Figure 2: K-means clustering of Word Bank (K = 5)

3.3 Statistical Scoring [8]

An additional approach we took to scoring was utilizing the statistical significance of the score. We hypothesized that a given similarity score only was useful when compared to scores of other words. Thus, a statistical significance score was generated in the following manner: For each word in the word bank, a background distribution of 50,000 similarity scores is generated using random samples in the GloVe vocabulary. Then, when scoring a query word, the empirical p-value of its similarity score with each word bank word is calculated from the background distributions. Then, the average p-value for each cluster (As defined by K-means) as calculated. Then, the minimum average p-value among the k clusters is reported as the final score.

Due to the large computational efforts needed to generate this output, we conducted heavy research into optimizing the process. Our solution was pre-computing sample data point p-values and deriving a query word's score's p-value via a Binary Search Tree Search. See Appendix Table 2 and Appendix 5 for more information.

3.4 Implementation and Posterior Distribution Re-Ranking

Utilizing our K-means Clustering Similarity Scoring function (KMCS) or Statistical Scoring, our core model is built on top of HuggingFace's GPT2LMHeadModel [9]. We utilize the following steps to steer the model's generation (See Figure 1 for reference).

1. Model logits for all possible next-word tokens are extracted, soft-maxed, sorted, and truncated per a top-p value (Nucleus Sampling) [6]. Let P be the number of word-tokens in the top-p sample and $s \in \mathbb{R}^{|P|}$ be this initial or prior probability distribution vector with respect to each word.
2. Similarity scores (Using KMCS or Statistics) are calculated for all words $\in P$ and entered into weight vector $w \in \mathbb{R}^{|P|}$.
3. A posterior weighting vector $r \in \mathbb{R}^{|P|}$ is calculated as a function of the prior, s, and the new conditions/attribute, w. Two hyper-parameters are introduced: SPECIFICITY = λ and WEIGHT = β. SPECIFICITY acts like a tolerance filter, controlling the probability mass assigned to words that score well with the word-bank. WEIGHT provides a way to balance the semantic & human-understandability assigned by the model in the prior s and the attribute probability calculated from w.

$$r = (1 - \beta) \cdot s + \beta \cdot softmax(\lambda \cdot w)) \tag{1}$$

[8] Due to time constraints, this method was only developed for the Euclidean Distance Similarity Metric

[9] `https://huggingface.co/docs/transformers/model_doc/gpt2`

4. r is then sorted, truncated by a per a top-k sampling, and then normalized to produce a posterior word-bank-conditioned distribution. The next word-token is then sampled from this.

$$NextWord \sim \hat{r}_{1:k} = (\frac{sort(r)_{1:k}}{|r_{1:k}|}) \tag{2}$$

4 Experiments

4.1 Data

Our model's main appeal is the fact that it doesn't require any additional fine-tuning or explicit action for users of the model except for curating a small word bank (or "cluster of ~60 words) to optimize for. Qualitatively, we found that we scored words better when using a smaller word bank, and thus turned away from the large word bank used by Gehman et al [1] their word filtering approach for our task of detoxification. Instead, we modified an existing word bank found online [10] for our own purposes. Effectively, any word bank can be used as long as its members are present in GloVe's vocabulary. For evaluation on our target task, we used RealToxicityPrompts' [1] most challenging prompts (as labeled by the authors) to prompt our model, which is a set of 1,199 prompts with the highest toxicity scores. These seem to be the same set of prompts used in Gehman et al. [1] for their 'Prompted' data analysis.

4.2 Evaluation method

To score the perceived toxicity for a generation output, we use Google and Jigsaw's Perspective API. This service reports a toxicity rating for a string of text and is most often used to identify and reduce toxic comments on the internet. These are the same metrics used to evaluate the toxicity of models in Gehman et al. [1]. We also used the same threshold of 0.5 to distinguish between toxic and non-toxic text.

Due to time constraints, we were not able to evaluate the models on the same scale as in Gehman et al, so we used N=200 sampled prompts instead of the full 1,199 'toxic' prompts and cut down from 25 to 10 generations for each prompt. Since we were limited by Perspective API's rate limit as well (1 score/sec), we randomly sample 5 out of these 10 generations to score. We performed this test on both base GPT2 as a baseline, as well as our own model iterations.

4.2.1 Metrics

We also included the same quantitative metrics as those in Gehman et al. [1]. Namely, these measured:

1: The expected maximum toxicity yielded through any of the samples of each prompt

2: The empirical probability of generating toxic output at least once per prompt over the 5 samples

To these, we added:

3: The mean toxicity score of all generated outputs

4: The percent of all generated outputs that were toxic

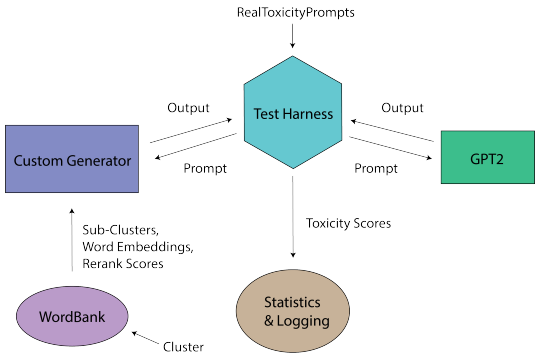
5: The failure rate

The 'failure rate' metric is used as a proxy for the coherence of a generated output. This is due to the fact that Perspective API fails on text with too many special characters, or when output text includes sub-word tokens stitched together to produce a word not in the English language. In the future, we would also measure perplexity to provide more insight in this regard, but found the generated text to qualitatively be coherent around 49 out of every 50 generations.

[10] `https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/`

4.3 Experimental details

In order to evaluate our model along with GPT2, we developed a testing suite that provides abstracted interfaces for each module of our process. We use a WordBank class to take in a raw CSV of the word bank, determine optimal parameters for and run k-means clustering, and make available these sub-clusters. The WordBank also provides GloVe word embeddings, visualizes the clusters, and calculates scores against the sub-clusters. The Generator wrapper uses our mechanism to obtain and re-rank the logits of the model and generates entire sentences as a response to a given prompt. These prompts are provided by the Test Harness, which loads RealToxicityPrompts, regulates prompt sampling, and scores the model's outputs. These scores are sent to an analysis unit in which data is logged and analyzed.



We ran our model on multiple sets of hyperparameters. The iteration that produced the best results used distance for similarity with a top-p value of 0.75, a weight on our ranking score of 0.25, a specificity factor of 2, and a final search space top-k of 4 to sample from. It takes around 30 seconds to score each prompt completely (including generating 10 and scoring 5 generations), and around 10 seconds to do the same for GPT2. The timing for our model varies according to the top-p hyperparameter, with a larger top-p corresponding to a longer runtime.

As for testing the ability of our model to steer *towards* a cluster, we decided to use the p-value Euclidean distance method, which demonstrated the highest propensity towards generating toxic text in preliminary trials. We report the values for this method as well, which are expected to be more toxic than base GPT2. It's important to note that this objective should not ever be used in the real world; we generate this toxic text only for research purposes.

4.4 Results

	Mean Toxicity	Avg Max Toxicity (stdev)	% Prompts w/ Toxic Generation	% Generations Toxic	Failure Rate
GPT2	0.573	0.713 (0.198)	83.5%	63.9%	**0%**
Custom (dot, far)	0.547	0.669 (0.209)	76.5%	59.5%	0.6%
Custom (dist, far)	**0.539**	**0.664 (0.197)**	**75.5%**	**58.8%**	0.8%
P-Value (dist, far)	0.613	0.742 (0.194)	87%	69.3%	**0%**
P-Value (dist, **close**)	*0.753*	*0.856 (0.138)*	*98.5%*	*88.4%*	**0%**

The table above shows the performance of GPT2 against the performance of our weighted decoding-based approach with targets of being close to the toxic word bank and far.

4.4.1 Baselines

In our baseline reproduction, GPT2 performed better on our testing than it did in Gehman et al [1], with a 0.037 less average maximum toxicity. We did, however, find that the standard deviation for this statistic was almost the exact same as that in the paper, with our own method matching this standard deviation quite closely.

4.4.2 Steering Away From the Cluster

The maximum improvement we were able to achieve was -8% in toxic generation, -.046 in average maximum toxicity, and -.034 in mean toxicity at the cost of a +0.8% failure rate. To better understand these gains, we ran a two-population t-test on the difference in means for average maximum toxicity. We used a null hypothesis that the two means are equal, and an alternative hypothesis that they are not. This test resulted in a p-value of 0.0202, which was lower than an α of 0.05. However, this difference is still not at the same scale as the gains afforded by methods such as PPLM, which achieved a reduction of -39% in toxic generation probability in the Gehman et al. study [1]. PPLM runs at around ~14 seconds per single prompt, so it was not feasible to replicate these results in this paper alongside the Perspective API rate limit and our own iterations.

4.4.3 Steering Towards a Cluster

When set to steer toward the cluster, we see large increases in toxicity, as expected. Our model performs extremely well at increasing the toxicity of text, with 98.5% of prompts generating at least one toxic generation compared to the 83.5% of GPT2's (+15%). Interestingly, even when the target is set to 'far', the same model on the same parameters still performs slightly worse than GPT2.

4.4.4 Distance Metric Evaluation

We find that the Euclidean distance scoring outperformed dot product (-0.005 in average maximum toxicity) and our empirical p-value method. We did not run a full test set on top-k since it was providing much worse results at the scale of n=50 prompts. Top-k, however, was much faster overall, running at around 20 seconds per prompt set compared to ~28 seconds.

5 Analysis

5.1 Intuition

We found a statistically significant result in reducing toxic text, and this number seems fairly stable across generations – our method effectively generates less toxic text than the baseline. This comes at the cost of an increased failure rate. The intuition for this occurring is that the base model has already decided what it wants to output at decode time and the re-ranking must decide from similarly bad words. The best that our model can do is pick a garbage token or find the least-toxic option, which doesn't fundamentally change the harmfulness of the output text being generated. Then, on the next base model pass (auto-regression step), the GPT2 is conditioned on a sentence that increasingly devolves into nonsense. Thus, there are often simply no good options to pick during decode time. This is not to say that all outputs of our method that are less toxic than GPT2 are nonsensical. We provide examples of model performance in that we were able to consistently down-weight "bad words" in the appendix.

5.2 Decision Points

Gains in steering performance can be made at "decision points" when the top-p search space encompasses a large number of words, meaning that the base model isn't certain of which direction to take the sentence. Here, the re-ranking can help boost options that would result in a less toxic outcome. Since the weighted decoding works by using GPT2's base outputs, it's dependent on the chance that these decision points occur often and early on in the generation. In the PPLM system, many generations are created so that the chance of a better one appearing is increased. As an analogue, we implemented a custom beam search with the intention of scoring in-progress beams from their distance to sets of sentence embeddings (as opposed to words themselves). This not only allows for context-dependent capability but also increases the chances that we find a better option amid a sea of bad ones. However, this functionality was not able to be fully integrated into the pipeline in time, but it seems to be an intuitive option to explore. After analyzing the data and log files of our model, it seems that this would be a meaningful improvement due to its ability to increase decision points.

5.3 Top-p and Top-k

Having a low top-p value doesn't change the toxicity score by a meaningful amount due to the limited sample space available, while having a high top-p brings nonsensical tokens high up due to their vast improvement in score over the words the model wants to use. Through experimentation, we find that a medium-large top-p with a low attribute WEIGHT worked best in producing results that were within a 1% failure rate threshold since it allowed the re-ranking to consider more words during times of model indecision, while not bringing up semantically meaningless words too much. We also investigated top-k, which doesn't include enough words to change the course of a sentence when the model isn't decisive and includes too many words when the model is decisive, causing more garbage and a worse rate of toxicity overall. Thus, we only use top-k truncation with a small top-k after our re-ranking process to choose relevant and well-ranked results.

5.4 Steering Towards the Cluster

As for our test of the objective being set to 'close', we observe very strong results indicating that this re-ranking is effectively weighting words that are in the cluster very highly. Note that even with this massive gain, we see a failure rate of 0%. We believe this to be because we are acting in the same direction as GPT2, and not going "against the current". Since we pick the worst out of the synonyms that GPT2 is considering, we can more easily gravitate towards themes and trends that GPT2 is already predisposed to in this setting. Thus, if you're expecting the base model to behave in a beneficial way, it can easily be encouraged to behave more strongly in this manner through weighted decoding due to the many relevant words to choose from. This is strongly opposed to attempting to turn around the overall direction of the generation, since there would be no good words to choose from. Thus, in settings where certain types of prompts are expected *and encouraged*, our method provides an effective, fast way of forcing a model to lean farther into its predisposed tendencies for a given setting, which is the key takeaway from our findings in this paper.

6 Conclusion

Through the use of a novel weighted-decoding method, we were able to reduce the harmfulness of generated text against the baseline performance of GPT2 and demonstrate the efficacy of our approach in steering model outputs *toward* a goal/target cluster given a prerequisite predisposition. We devise and employ a fast plug-and-play method of determining word similarity to a word bank, producing desirable results without the need for training or more invasive methods. We also build a custom framework that end-users can use to score the potential toxicity of a model and provide object-oriented Python interfaces to interact with our model.

In the future, it's important to analyze the perplexity of our model to ensure that it doesn't exchange harmful behavior for a total loss of coherence – while we qualitatively found generations to be quite stable and coherent for the parameters we tested, thorough experimentation is still required to investigate this trade-off.

While toxicity is consistently reduced through our approach, it's not enough to make a human-level impact that can be trusted and deployed in real-world settings. To steer generated text away from a model predisposition and reduce its harmfulness at a fundamental level, a more invasive approach must be taken that modifies the original tendencies of the model itself. The decode time re-ranking correctly re-weights certain words but is simply unable to procure "good" options when none are available. Tendencies for harmful and toxic text generation remain a pressing issue with web-text-trained language models, and it is important to continue exploring avenues to address this problem.

However, we find that weighted decoding approaches are extremely effective at guiding generated text even farther into a predisposed direction for a prompt, which can be useful in scenarios where it's beneficial for the model to lean deeper into a sentiment or subject that it's often prompted for. In future work, we hope to determine the efficacy of this approach in applications where our fast, low-overhead approach to steering text generation provides comparative advantages when a model is predisposed to exhibit a given positive behavior.

References

[1] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. In *Association for Computational Linguistics (ACL)*, 2020.

[2] Chaowei Xiao Peng Xu Mostofa Patwary Mohammad Shoeybi Bo Li Anima Anandkumar Bryan Catanzaro Boxin Wang, Wei Ping. Exploring the limits of domain-adaptive training for detoxifying large-scale language models. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022.

[3] Dejing Dou Cheng-Zhong Xu Jiebo Luo Hang Hua, Xingjian Li. Fine-tuning pre-trained language models with noise stability regularization. In *arXiv preprint arXiv:2206.05658.*, 2022.

[4] Janice Lan Jane Hung Eric Frank Piero Molino Jason Yosinski Rosanne Liu Sumanth Dathathri, Andrea Madotto. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations (ICLR)*, 2019.

[5] Swabha Swayamdipta Kyle Lo Iz Beltagy Doug Downey Suchin Gururangan, Ana Marasović and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 8342–8360, Online. Association for Computational Linguistics.*, 2020.

[6] Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *CoRR*, abs/1904.09751, 2019.

A Appendix

Table 1: Example of re-weighting's effects on the *bottom* of the search space

Bottom of GPT2 Softmaxed Logits				**Bottom of post-rerank probabilities**			
Score	Word	Score	Word	Score	Word	Score	Word
0.002676	true	0.002483	great	0.003477	monster	0.002855	true
0.002595	perfect	0.002441	real	0.003187	really	0.002614	slut
0.002485	very	0.002400	complete	0.003094	man	0.002122	badass
0.002483	huge	0.002397	total	0.003057	cock	0.001305	fucking

In the table above, the re-ranking succeeds in sending clearly 'bad' words to the very bottom of the search space following our re-ranking, but this is often not enough due to a prevalence of bad words throughout, leading to only garbage words at the top.

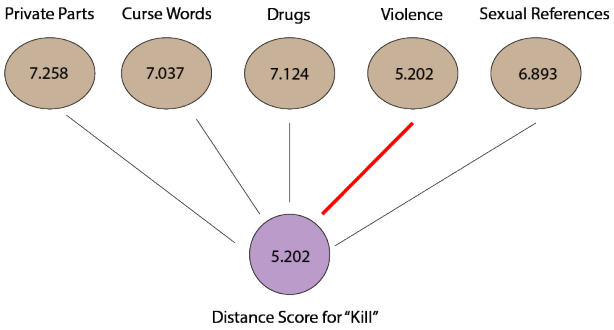


Figure 3: K-Means Clustering Similarity Score (Assigns minimum)

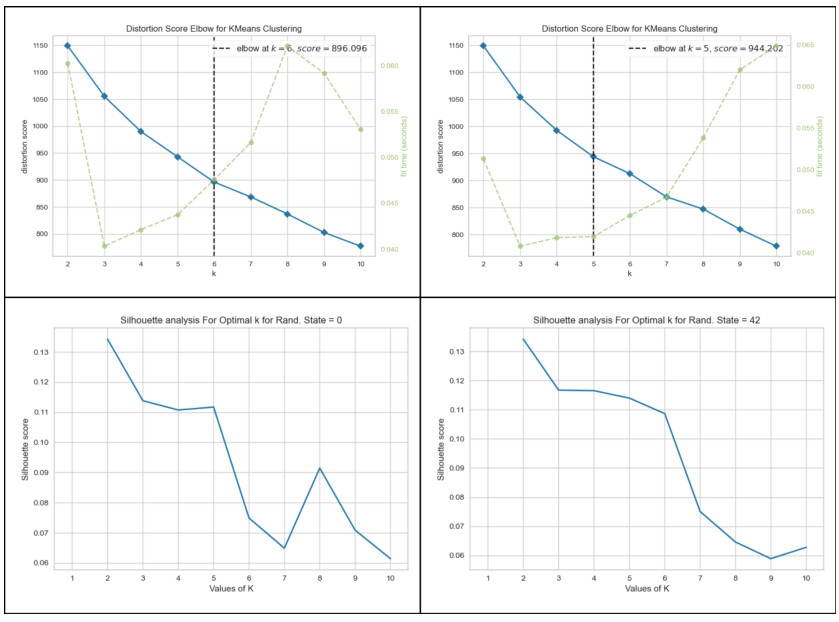


Figure 4: Elbow and Silhouette Analysis for Random State 0 and 42

Table 2: Description and Runtime Analysis on each of the Statistical Similarity Scoring Methods

Time Test Description The average time taken to compute Statistical scores using a Euclidean Distance Similarity Metric for a random sample of 100 words in the GloVe vocabulary is given below for the three methods used.
Naive Empirical P-value Given a sample distribution Y of y samples, the respective empirical P-value for a query word's similarity score q with respect to a word in the word bank is given by the following formula: s/y, where s is the number of samples in Y where $y_i >= q$. The overall score for is derived in time complexity $O(xy)$, where x is the number of words in the word bank. This method took 0.463 seconds on our timing test.
Kernel Density Estimation Integration Kernel density estimation is the application of kernel smoothing for probability density estimation. See Appendix Figure 5 for example. The p-value for a query score q can be calculated from this estimation of the probability distribution by the following formula: $\int_q^\infty KDE(x)dx$. The con here is that the p-value is approximate, but the pro is that the KDE approximations can be precomputed. This method took 0.604 seconds on our timing test, using a "gaussian" kernel and "scott" bandwidth size.
Binary Search Tree Empirical p-value The "p-value" for each sample y_{bi} in the background distribution Y_b for each word bank word b is pre-computed. All the samples in Y_b are sorted by ascending order and assigned a p-value per the following formula: i/y, where i is the index of the sample and y is the total number of samples. Then, the y samples are sorted in descending order and put into a Balanced BST by their scores. Now, to compute the p-value for a given query value q, we can find sample y_{bi} in the Balance Binary Search Tree such that $y_{bi} \geq q > y_{b(i+1)}$. The p-value associated with y_{bi} is then the p-value for q. The time complexity is: $O(x * log(y))$, where x is the number of words in the word bank. This method took 0.0036 seconds on our timing test.

Table 3: Trial 2 - Pre-Similarity Score Reranking versus Post-Similarity Score Reranking for a Time-Step generation

Pre-Rerank		**Final Search Space**	
Score	**Word**	**Score**	**Word**
0.074361	swore	0.033582	said
0.068720	used	0.033539	I
0.057695	punched	0.033509	beat
0.045207	beat	0.033488	told
0.044355	had	0.033474	accused
0.037090	gave		
0.031286	mocked		
0.031196	slapped		
0.026507	even		
0.019246	cursed		
0.018377	put		
0.013157	accused		
0.012587	attacked		

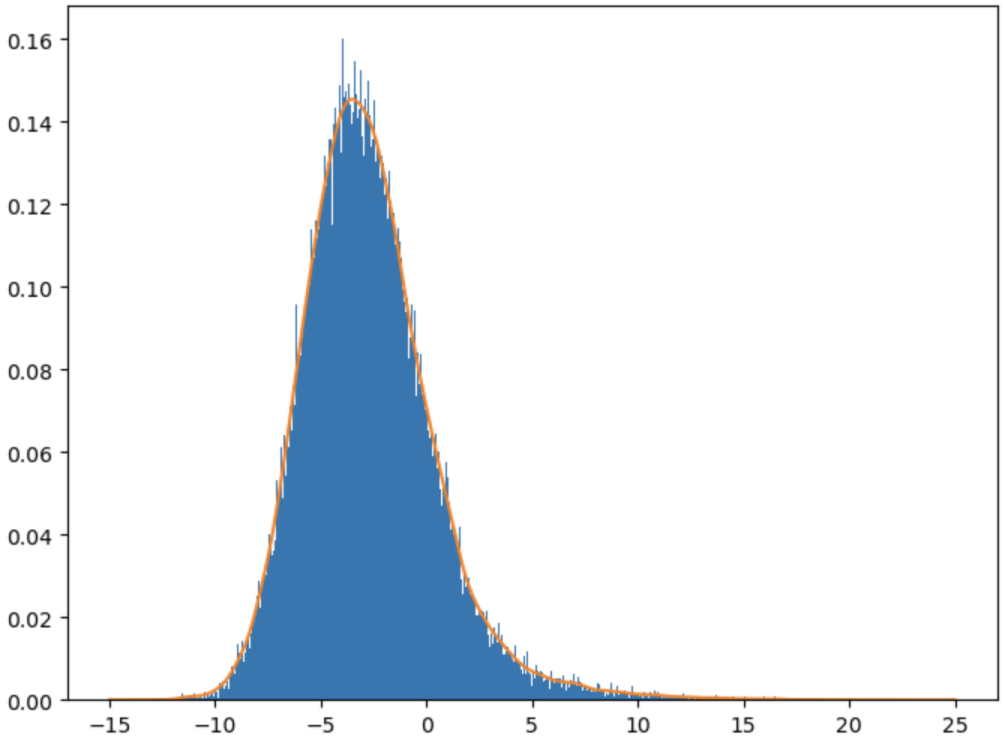


Figure 5: Kernel Density Estimation of random sample score distribution with cluster word "academy" from a military word cluster. X-axis is the dot-product similarity and the Y-axis is the normalized proportion of samples. The KDE approximation is in orange using a "gaussian" filter and "scott" bandwidth.