Generate Your Counterfactuals: Towards Controlled Counterfactual Generation for Text

Nishtha Madaan, Inkit Padhi, Naveen Panwar, Diptikalyan Saha
[1]IBM Research AI
{nishthamadaan, naveen.panwar, diptsaha}@in.ibm.com, inkpad@ibm.com

Abstract

Machine Learning has seen tremendous growth recently, which has led to a larger adoption of ML systems for educational assessments, credit risk, healthcare, employment, criminal justice, to name a few. Trustworthiness of ML and NLP systems is a crucial aspect and requires guarantee that the decisions they make are fair and robust. Aligned with this, we propose a framework *GYC*, to generate a set of counterfactual text samples, which are crucial for testing these ML systems. Our main contributions include a) We introduce GYC, a framework to generate counterfactual samples such that the generation is plausible, diverse, goal-oriented and effective, b) We generate counterfactual samples, that can direct the generation towards a corresponding `condition` such as named-entity tag, semantic role label, or sentiment. Our experimental results on various domains, show that GYC generates counterfactual text samples exhibiting the above four properties. GYC generates counterfactuals that can act as test cases to evaluate a model and any text debiasing algorithm.

Introduction

Owing to the advancement of deep learning in recent years, AI systems are now deployed for varied tasks and domains like educational assessments, credit risk, healthcare, employment, criminal justice. With the deployment of NLP systems in these domains, the need to ascertain trustworthiness and reliability becomes crucial to ensure the decisions that they make are fair and robust. To this end, counterfactual text (Pearl et al. 2000) provides an alternate dataset which acts as test-cases to evaluate the properties such as fairness and robustness of these systems. These texts along with its correct labels may also be used to augment the training set to mitigate such properties (Garg et al. 2019).

The four properties that are important for counterfactual text are plausibility, diversity, goal-orientedness, and effectiveness. Plausibility ensures that the generated test cases are realistic and can occur in the input and thus can be further used for retraining. Goal-orientedness can ensure that the generated counterfactual text samples are deviating from the original sample on a particular aspect like NER, SRL, synonyms or fairness attributes as controlled by the user. The diversity will ensure high coverage of the input space defined by the goal. In this paper, we aim to generate such counterfactual text samples which are also effective in finding test-failures (i.e. label flips for NLP classifiers).

Recent years have seen a tremendous increase in the work on fairness testing (Ma, Wang, and Liu; Holstein et al. 2019) which are capable of generating a large number of test-cases that capture the model's ability to misclassify or remove unwanted bias around specific protected attributes (Huang et al. 2019), (Garg et al. 2019). This is not only limited to fairness but the community has seen great interest in building robust models susceptible to adversarial changes (Goodfellow, Shlens, and Szegedy 2014; Michel et al. 2019; Ebrahimi et al. 2017; Zhao, Dua, and Singh 2017). These models are capable of generating sentences that can change the label of the classifier. These works are very limited to a specific domain. While the change of label is likely, the generations are not plausible and are ungrammatical (Li, Monroe, and Jurafsky 2016) or require manual human intervention in many cases (Jia and Liang 2017).

A recent work by Ribeiro et al. 2020 employs a tool Checklist which is one of the attempts to come up with generalized test-cases. For generation, Checklist uses a set of pre-defined templates, lexicons, general-purpose perturbations, and context-aware suggestions. A major limitation with these template-based approaches or rule-based approaches is that these can not generate meaningful diverse counterfactual samples. Our goal is to be able to generate these counterfactual samples for a given text in a more generalized manner with high generation quality and diversity.

Generative models like GPT-2 (Radford et al. 2019) are good at generating plausible and diverse text. We can leverage these capabilities to generate counterfactual text (Pearl et al. 2000). However, directly adopting Transformers architecture (Vaswani et al. 2017) to generate perturbations for a goal-oriented task is difficult without adding additional attributes and fine-tuning with attribute-specific data (Keskar et al. 2019). Similar to (Dathathri et al. 2020), we perform controlled text generation to achieve the above objectives. There are two key technical challenges in adopting this approach. First, transformers (Vaswani et al. 2017) are good at generating feasible and plausible but inferencing on GPT-2 is hard. Second, prior approaches to control GPT-2 text generation do not support a custom model to

Input Sentence	**Token-based Substitution** (Ribeiro et al. 2020), (Devlin et al. 2018)	**Adversarial Attack** (Michel et al. 2019)	**GYC** (Ours)
I am very *disappointed* with the service	I am very **pleased** with the service. I am very **happy** with the service. I am very **impressed** with the service.	I am very **witty** with the service.	I am very **pleased to get a good** service. I am very **happy** with **this** service. I am very **pleased** with the service.
1st time burnt *pizza* was *horrible*.	1st time burnt **house** was **destroyed**. 1st time burnt **place** was **burned**. 1st time burnt **house** was **burnt**.	**personable** time burnt pizza was horrible.	1st time burnt pizza **is delicious**. 1st time burnt **coffee** was **delicious**. 1st time burned pizza was **delicious**.

Table 1: Comparison of generated counterfactual text from existing and proposed model on text from YELP dataset.

direct the generation. In this paper, we address the drawbacks of the above approaches by keeping the balance of the four properties of counterfactual text. This framework can be used to plug in any custom model to direct the generation towards counterfactual. In our framework, we first reconstruct the input text and then introducing losses that work with differential and non-differentiable models to enforce a corresponding `condition`. This condition can be a sentiment, NER, or a different user-provided class label. The differentiable loss is computed over logits of the input sentence, while non-differentiable loss relies on generating reward computed over text. Further, to ensure that the generated counterfactual text is diverse, we add entropy over logits of input.

Few examples of given input sentences in Table 1 show that the existing methods generate hard test cases with either single word replacements or by adding an existing template to generate multiple test-cases which do not possess the required generation quality and are neither diverse in nature. Recent adversarial approaches like (Michel et al. 2019) generate adversarial changes on input text to change the label of output sentence, regardless of whether the output sentence makes sense. These adversarial generations may flip the label but these generations are not likely to be ever seen in user input. Therefore these test-cases become ineffective. On the other hand, it would be more useful to generate counterfactual text samples that are likely to be seen in user inputs.

In particular, our key contributions are listed as follows:

(a) We introduce GYC, a framework to generate counterfactual samples such that the generation is plausible, diverse, goal-oriented, and effective. This is in contrast to the previous work which produces adversarial text but is ungrammatical or non-plausible. We show a comparison in examples 1 and 2 in Table 1.

(b) We generate counterfactual samples, that can direct the generation towards a corresponding `condition` such as NER, SRL, or sentiment. We quantify our results (Tab. 4 and 5) using automated evaluation metrics (by training `condition` models) to check Label-flip and diversity. Through human judgment we measure plausibility, grammar check, and validity of generation.

(c) GYC allows plugging-in any custom `condition` model to guide the counterfactual generation according to user-specific `condition`. We show that our framework is able to conditionally generate counterfactual text samples with high plausibility, label-flip score, and diversity.

(d) GYC enables the use of pre-trained GPT-2 decoder and does not require any re-training or fine-tuning of the model.

Background

Language Modeling for Text Generation

Language Models have played a significant role through introduction of word2vec (Mikolov et al. 2013), contextual word vectors (Devlin et al. 2018) and deep contextualized LM models (Radford et al. 2019; Howard and Ruder 2018).

Recent advances by Vaswani et al. 2017 introduce large transformer-based architectures which have enabled training on large amounts of data (Radford et al. 2019; Devlin et al. 2018)

In this modeling, there is a history matrix that consists of key-value pairs as $\mathbf{H}_t = [(K_t^{(1)}, V_t^{(1)}), \ldots, (K_t^{(l)}, V_t^{(l)})]$, where $(K_t^{(i)}, V_t^{(i)})$ corresponds to the key-value pairs of the $i-$th layer of a transformer-based architecture at time-step t. Then, we can efficiently generate a token $\mathbf{y}_t$ conditioned on the past text $\mathbf{y}_{<t}$. This dependence on the past text is captured in the history matrix $\mathbf{H}_t$. Using this model a sample of text can be generated as follows.

$$\mathbf{o}_{t+1}, \mathbf{H}_{t+1} = \mathrm{LM}(\mathbf{y}_t, \mathbf{H}_t) \tag{1}$$

$$\mathbf{y}_{t+1} \sim \mathrm{Categorical}(\mathbf{o}_{t+1}), \tag{2}$$

Here, $\mathbf{o}_{t+1}$ are the logits to sample the token $\mathbf{y}_{t+1}$ from a Categorical distribution.

Controlled Text Generation

The aim of controlled text generation is to generate samples of text conditioned on some given `condition`.

$$p(\mathbf{y}|\texttt{condition}) \tag{3}$$

This condition can be a class label such as the sentiment (*negative* or *positive*), topic (*sci-fi* or *politics*) or tense (*past*, *present* or *future*) (Hu et al. 2017; Dathathri et al. 2020). The condition can also be structured data such as a Wikipedia infobox or even a knowledge graph (Ye et al. 2020).

Dathathri et al. (2020) propose a new framework called PPLM for controlled text generation. PPLM uses a pre-trained GPT-2 to generate text as stated in eq. 2. However, to control the text, PPLM perturbs the history matrices $\mathbf{H}_t$ by adding a learnable perturbation $\Delta\mathbf{H}_t$ to get a perturbed history matrix $\tilde{\mathbf{H}}_t$.

$$\tilde{\mathbf{H}}_t = \mathbf{H}_t + \Delta\mathbf{H}_t \tag{4}$$

This is plugged into eq. 2 to generate the controlled samples.

$$\mathbf{o}_{t+1}, \mathbf{H}_{t+1} = \text{LM}(\mathbf{y}_t, \tilde{\mathbf{H}}_t) \tag{5}$$
$$\mathbf{y}_{t+1} \sim \text{Categorical}(\mathbf{o}_{t+1}), \tag{6}$$

PPLM approximates the process of sampling from the required distribution for $p(\mathbf{y}|\texttt{condition})$. For this, PPLM trains the learnable parameters $\Delta\mathbf{H}_t$ so that perturbed samples approximate the required distribution. This is done by maximizing the log-probability of $p(\mathbf{y}|\texttt{condition})$. To evaluate this distribution, Bayes rule is used as follows:

$$p(\mathbf{y}|\texttt{condition}) \propto p(\mathbf{y})p(\texttt{condition}|\mathbf{y}) \tag{7}$$

For instance, to enforce the condition so that generated text samples belong to a class c with respect to some given classifier $p(c|\mathbf{y})$, PPLM maximizes the following objective.

$$\log p(c|\mathbf{y}) - \sum_t D_{\text{KL}}\left[p(\mathbf{y}_t|\mathbf{y}_{t-1}, \tilde{\mathbf{H}}_{t-1})||p(\mathbf{y}_t|\mathbf{y}_{<t})\right] \tag{8}$$

Here, the first term enforces the class label. The second term is a KL term which enforces that the text is plausible and is in line with unperturbed GPT-2 distribution.

Counterfactual Text

A counterfactual example is defined as synthetically generated text which is treated differently by a `condition` model. Any text is a description of a scenario or setting. For instance, given a text $\mathbf{x}$, *My friend lives in London*, a counterfactual text $\mathbf{y}$ becomes - *My friend lives in California*, if the friend counterfactually lived in California. From the perspective of a given classifier model, a counterfactual example defines how the original class label flips if the friend lived in California instead of London. Such generated counterfactuals can serve as test-cases to test the robustness and fairness of different classification models. Some examples can be seen in Table 6. An effort to make the classifier fairer could utilize such counterfactual data for data augmentation task (Garg et al. 2019).

GYC Method

In this section, we describe our method for generating K counterfactual text samples $Y = \{\mathbf{y}^k\}_{k=1}^K$ given a text $\mathbf{x}$ and controlling `condition` that specify the scope of the counterfactual text. Formally, this modelling is described by:

$$p(\mathbf{y}|\mathbf{x}, \texttt{condition}) \tag{9}$$

To implement this, we first obtain $\tilde{\mathbf{H}}$ which depends on both the `condition` and the $\mathbf{x}$ as opposed to just the condition. Hence, we first compute $\tilde{\mathbf{H}}$ on the basis of $\mathbf{x}$ such that the generated samples reconstruct the input text $\mathbf{x}$. Next, the `condition` is enforced by further perturbing the $\tilde{\mathbf{H}}$ to obtain $\hat{\mathbf{H}}$.

Model : NER Location Tagger Source: N/A, Target: Perturb Location-Tag Input Sentence: *My friend lives in beautiful London.* Counterfactual Text Samples : [1] My friend lives in majestic downtown Chicago. [2] My friend lives in gorgeous London. [3] My friend lives in the city of New Orleans.
Model : Sentiment Classifier Source: Negative Class Label, Target: Positive Class Label Input Sentence: *I am very disappointed with the service.* Counterfactual Text Samples : [1] I am very pleased with the service. [2] I am very happy with this service [3] I am very pleased to get a good service.
Model : Topic Classifier Source Topic: World, Target Topic: Sci-Fi Input Sentence: *The country is at war with terrorism.* Counterfactual Text Samples : [1] The country is at war with piracy at international waters. [2] The country is at war with its own beurocracy. [3] The country is at war with piracy offenses.

Table 2: Illustration of samples generated by GYC. In the first example, `condition` enforced is to change the location tag. GYC selectively changes the city *London* to *Chicago* and *New Orleans* while maintaining the sense of the remaining text. In the second example, the `condition` enforced is to change the sentiment from negative to positive with respect to a provided classifier. In the third example, the goal is to change the topic label of the text to *sci-fi*.

Generative Process

Formally, our aim is to draw K counterfactual samples as $\mathbf{y}^k \sim p(\mathbf{y}|\mathbf{x}, \texttt{conditions})$. This can be factorized auto-regressively over time as:

$$p(\mathbf{y}|\mathbf{x}, \texttt{condition}) = \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{y}_{<t}, \mathbf{x}, \texttt{condition}) \tag{10}$$

To implement this, we learn $\hat{\mathbf{H}}$ on the basis of the given text $\mathbf{x}$ and the given `condition` and use it to sample the counterfactual text samples as follows:

$$\mathbf{o}_{t+1}, \mathbf{H}_{t+1} = \text{LM}(\mathbf{y}_t, \hat{\mathbf{H}}_t) \tag{11}$$
$$\mathbf{y}_{t+1} \sim \text{Categorical}(\mathbf{o}_{t+1}), \tag{12}$$

Learning Counterfactuals

In this section, we describe our approach for learning the hidden state perturbations $\tilde{\mathbf{H}}_t$ for reconstruction and subsequently $\hat{\mathbf{H}}_t$ for enforcing the given conditions. Note that

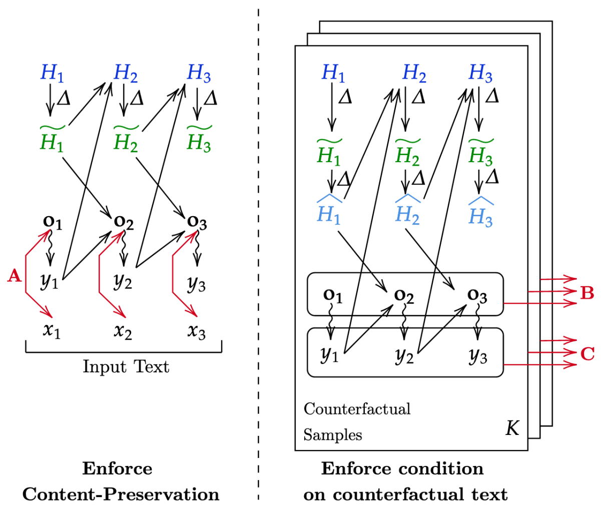

Figure 1: The architecture diagram showing interplay of different losses for generation. **A** represents proximity loss, **B** represents differentiable score and entropy, **C** represents reward based score and entropy, Δ represents perturbation.

tilde on $\tilde{\mathbf{H}}_t$ represents perturbation for reconstruction and hat on $\hat{\mathbf{H}}_t$ represents the perturbation for condition-oriented generation. We describe the process through the following loss functions used in our generation process also shown in Fig 1.

Reconstruction via Proximity Loss. A key problem in generative models is to perform inference over the latent states given the observation. GPT-2 suffers from the same problem. In our setup, the inference step is crucial because we require the latent state that generates the given input text before we can perturb it to generate counterfactual text.

To do this, we first aim to learn perturbations $\Delta\mathbf{H}_t$ such that the generated samples reconstruct the given text $\mathbf{x}$. This is done by learning $\Delta\mathbf{H}_t$ to maximize the log probability of the given text.

$$\mathcal{L}_p = \log p(\mathbf{y} = \mathbf{x} | \mathbf{x}, \texttt{condition}) \tag{13}$$
$$= \sum_t \log p(\mathbf{x}_t | \mathbf{x}_{<t}, \tilde{\mathbf{H}}_{t-1}) \tag{14}$$

In a sense, this objective enforces proximity of the generated text to the original text and also enforces content preservation. Hence, we call it the proximity loss. For implementing this, we found a curriculum-based approach to perform better where we reconstruct one token at a time from left to right. For more details on this implementation, see Appendix A in supplementary material.

Enforcing `condition` To enforce `condition`, we assume we have access to a score function that takes the text and returns the degree to which the condition holds in the text. This score function may be differentiable with respect to the logits of the generated text, such as when the score is the probability of a class label returned by a model. In other cases, the score function may be akin to a reward and may not be differentiable with respect to the text. In either case, we aim to generate text that maximizes the score. To handle the differentiable and non-differentiable cases, we use the following losses.

Differentiable Score. Differentiable Score assumes that the input to the score function `score` are the logits $\mathbf{o}_{1:T}$ of the tokens of the generated text. This can be translated into a corresponding differentiable loss as follows:

$$\mathcal{L}_d = \texttt{score}(\mathbf{o}_{1:T}) \tag{15}$$

where $\mathbf{o}_{1:T}$ are obtained from equation (11).
For our experiments, we train a sentiment classifier on the datasets- Yelp (Shen et al. 2017), DBpedia (Lehmann et al. 2014) and AgNews (Zhang, Zhao, and LeCun 2015) which is used to compute the differentiable loss.

REINFORCE. To enforce a condition which is not differentiable with respect to the logits $\mathbf{o}_{1:T}$ of the generated text, we resort to the REINFORCE technique. This type of scores are important because a variety of models take text as input and not the logits. For instance, a classifier may take text as input to perform tagging (such as NER or SRL) or to perform classification. We assume that the score function `score` is a reward that is obtained for the generated text. We draw K samples of text $\{\mathbf{y}^k_{1:T}\}_{k=1}^K$ based on the currently learned $\hat{\mathbf{H}}_t$. Then we compute a REINFORCE objective as follows.

$$\mathcal{L}_r = \sum_{k=1}^{K} r^k \sum_{t=1}^{T} \log\ p\left(\mathbf{y}^k_t | \mathbf{y}^k_{t-1}, \hat{\mathbf{H}}_t\right) \tag{16}$$

where r^k represents reward for each k counterfactual text. For our experiments, we use BERT-NER based model and SRL model to generate reward on NER and SRL in generated counterfactual text samples.

Diversity. To make sure that the distribution over the generated text does not collapse to the input text $\mathbf{x}$ and to ensure diversity in the generated counterfactual text samples, we introduce a diversity loss. To implement this, we compute entropy of the logits of the K generated text samples.

$$\mathcal{L}_H = \sum_{k=1}^{K}\sum_{t=1}^{T} H\left(\mathbf{y}^k_t | \mathbf{y}^k_{<t}\right) \text{ where } H \text{ is entropy} \tag{17}$$

Combined Objective. The combined objective which we maximize can be stated as following:

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_H \mathcal{L}_H + \lambda_p \mathcal{L}_p \tag{18}$$

where λ_r, λ_H and λ_p are hyper-parameters that can be tuned. To facilitate training we perform annealing on these hyper-parameters. In early training, we keep $\lambda_r, \lambda_H = 0$. After reconstruction completes, λ_p is lowered and λ_r and λ_H are set to non-zero. See more details in Appendix A in supplementary document.

Applications of Counterfactuals

Counterfactuals as Test-Cases

Test-Cases are important in testing NLP models to perform a behavioral check on a model. These test-cases also complement the traditional test cases designed in the software engineering world and also seems to be very relevant with the increased adoption of NLP algorithms. GYC can be easily adapted to test any classification model with even a non-differentiable score function. One can plug any score function, and generate test-cases around a specific condition. In experiments, we take a score function as a Sentiment prediction function. GYC reports the highest scores in quantitative and qualitative results on different metrics across various datasets. The detailed experiments on sentiment and the performance on different metrics around generation and diversity are described in Table 4 and 5 and Fig. 3.

Counterfactuals for Debiasing

Applications of counterfactual text in debiasing have been studied well in literature by (Garg et al. 2019), (Madaan et al. 2018) highlight multiple text debiasing algorithms which can be used to debias classification systems. The key to Data Augmentation and Counterfactual Logit Pairing Algorithms is that they require counterfactual text samples of the training data. Similarly, the counterfactual text is used to debias Language Model for Sentiment (Huang et al. 2019). These samples should satisfy a `condition` in this case a protected attribute should be present. While these techniques are claimed to be highly successful, in real-world getting enough data for a corresponding protected attribute is prohibitive. The performance of these algorithms can be boosted if the counterfactual text samples that are fed possess high generation quality. GYC can be used to generate data for such data augmentation tasks which can thereby help debias a model.

Related Work

Test-Case Generation

Recent works on software testing have been addressing the problem of increasing fairness in different applications. In structured data, Themis (Udeshi, Arora, and Chattopadhyay 2018), Equitus (Galhotra, Brun, and Meliou 2017) and (Aggarwal et al. 2019; John, Vijaykeerthy, and Saha 2020) use random testing to generate test-cases. (Wachter, Mittelstadt, and Russell 2017) defined the generation of counterfactual explanations that can be used to evaluate a model. Further (Mothilal, Sharma, and Tan 2020) address the feasibility and diversity of these counterfactual explanations. In text data, recently the work by (Ribeiro, Singh, and Guestrin 2018) introduced a framework to generate test-cases relying on token-based substitution. This work is the closest work to our introduced framework.

Controlled Text Generation

Due to the discrete and non-differential nature of text, controlled text generation task is non-trivial and most of the current methods indulge in fine-tuning using reinforcement (Yu et al. 2017), soft argmax (Zhang et al. 2017) or Gumbel-Softmax techniques (Jang, Gu, and Poole 2016). Moreover, due to advancements in pre-trained text representations, recent work tries to achieve conditional generation without requiring the need to re-training. (Engel, Hoffman, and Roberts 2017) enforces behavioral latent constraints as part of post-hoc learning on pre-trained generative models, whereas (Dathathri et al. 2020) incorporates attribute discriminators to steer the text generations.

Adversarial Attack

Adversarial literature especially the work by Ebrahimi et al. (2017) have shown promising performance in generating samples that tend to change the classifier's outcome. (Ebrahimi et al. 2017) showed that by using a gradient-based method and performing a minimal change in the sentence the outcome can be changed but the generated sentences might not preserve the content of the input sentence. (Michel et al. 2019; Zhao, Dua, and Singh 2017) explored ways to preserve content while also change the classifier's outcome. For instance, (Michel et al. 2019), leverages a method of imbuing gradient-based word substitution with constraints aimed at preserving the sentence's meaning.

	Checklist	PPLM-BoW	Masked-LM	GYC
Dictionary-free	✗	✓	✓	✓
Sentence-Level CF	✗	✗	✗	✓
Diversity	✗	✗	✗	✓
Enforce `condition`	✗	✓	✗	✓

Table 3: Models for evaluation.

Experiments

Metric	Model	Dbpedia	Agnews	Yelp
Label-Flip Score ($\uparrow$ better)	Masked-LM	30.00	37.80	10.87
	PPLM-BoW	**75.29**	42.19	35.55
	Checklist	24.03	27.27	4.00
	GYC	72.09	**42.76**	**70.29**
Diversity Score ($\downarrow$ better)	Masked-LM	0.98	0.97	0.96
	PPLM-BoW	0.88	**0.88**	**0.87**
	Checklist	0.98	0.97	0.97
	GYC	**0.87**	0.90	0.88

Table 4: Evaluation of Label Flip Score (representing the counterfactual text samples generated with a different label) and Diversity Score (representing the diversity in generated counterfactual text) of baselines.

Baselines

We employ baselines with different categories like-dictionary-free, Sentence level counterfactual text, diversity, Enforce `condition` as shown in Table 3. We describe the implemented baselines as follows:

(a) CheckList: Checklist performs rule-based, lexicon-based and LM based perturbations. It defines a set of pre-defined templates to construct test-cases. This is the only existing model that makes use of LM based perturbations for test-case generation. The generated test cases are based

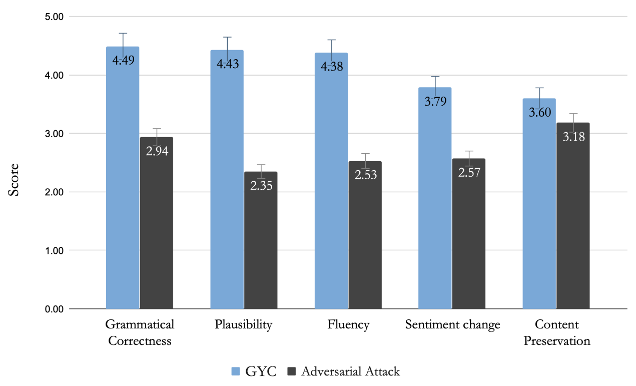

Figure 2: Comparison of human evaluation of generated text from Adversarial Attack and GYC.

Metric	Model	Dbpedia	Agnews	Yelp
Content Preservation (↑ better)	Masked-LM	0.89	0.84	0.83
	PPLM-BoW	0.18	0.20	0.31
	Checklist	**0.90**	**0.85**	**0.85**
	GYC	0.56	0.77	0.61
Tree-Edit Distance (↓ better)	Masked-LM	0.93	1.67	1.40
	PPLM-BoW	3.76	3.35	4.00
	Checklist	**0.38**	**0.97**	**0.72**
	GYC	2.87	1.79	2.69

Table 5: Evaluation of Content Preservation (semantic similarity between input sentence and generated counterfactual text) and Tree-Edit Distance (syntactic structure of input sentence and generated counterfactual text) of baselines.

on token-substitution. Hence, these provide very rigid perturbations. According to a chosen template, it perturbs the specific part of the template. e.g. name, adjective, or location is taken from a dictionary. In LM-based perturbations, a mask is specified at a certain position in the text and it predicts possible test cases via RoBERTa (Liu et al. 2019). For more details, see Appendix B in supplementary document.

(b) BERT: BERT is a masked language model, where we randomly mask tokens from input sequence and the model learns to predict the correct token. Therefore to compare our approach with LM-based perturbations we considered this as one of the baselines. We took a pre-trained BERT model for our experiments. To generate counterfactual text, we randomly mask one word from the input sequence after skipping the first 2 words as context. We denote this model as *Masked-LM*.

(c) PPLM-BoW: PPLM (Dathathri et al. 2020) uses a bag-of-words to control the text generation. We feed first 2 words of the input sentence as context and the rest of the text is generated. We hypothesize that PPLM-BoW should be unable to generate sentences with similar meaning. This is because it should try to exactly use the words provided in the input text but in wrong order or sense. Due to the interplay between the KL term and the dictionary term it should exhibit low content preservation.

(d) Adversarial Attack: The work by (Michel et al. 2019) formalizes the method of generating meaning-preserving perturbations in the form of adversarial attacks that tend to change the label of the classifier for a given input text. In our setup, we use the implementation of (Michel et al. 2019) for adversarial perturbations by Allen AI (Gardner et al. 2017). We denote this model as **Adversarial Attack**. We use this model for qualitative analysis in the human evaluation section as we intend to compare the plausibility of generated counterfactual text.

Datasets

We perform the experiments on DBpedia (Lehmann et al. 2014), AgNews (Zhang, Zhao, and LeCun 2015) and Yelp (Shen et al. 2017). All three datasets come from different domains. The details and properties of each dataset taken are described as follows:

DBPedia. This dataset focuses on locations and organizations. We use publicly available DBpedia dataset[1] by (Zhang, Zhao, and LeCun 2015) containing 14 classes. The dataset contains 560K training samples and 70K test samples.
AgNews. This dataset focuses on real-world data containing four classes - world, business, sports, and sci-fi categories. Each class contains 30K training samples and 1.9K testing samples. The total number of training samples is 120K and testing 7.6K.
Yelp. This dataset focuses on informal text containing reviews. The original YELP Polarity dataset has been filtered in (Shen et al. 2017) by eliminating those sentences that exceed 15 words. We use the filtered dataset containing 250K negative sentences, and 350K positive ones.

Metrics

Generation Quality We define various quantitative metrics to measure different aspects of counterfactual text generations. The choice of these metrics is to ensure both the viability of the generated texts as counterfactual text (metric 1,2), and also to examine their plausibility in semantic and syntactic structure (metrics 3,4). To evaluate the K counterfactual samples generated from input text $\mathbf{x}$, the metrics for measuring the proximity of generated counterfactual text samples, diversity, and syntactic structure are as follows:

(a) Label-Flip Score: Label-Flip Score, LFS, is an accuracy-based metric which is used to assess if the generated counterfactual text belong to a different class than the source text class. In our setting, we train an XL-Net (Yang et al. 2019) on different datasets and then compute the labels of the counterfactual text and obtain the Label-flip score. For an input text $\mathbf{x}_i$ with label C_i, we generate K counterfactual text samples and define LFS for this example as:

$$\mathrm{LFS}_i = \frac{1}{K}\sum_{k=1}^{K} I(C_i \neq \hat{C}_i) \times 100$$

[1] http://goo.gl/JyCnZq

(b) Diversity Score: Similar to the Self-BLEU metric proposed by Zhu et al. (2018), we propose a Self-BERT metric using Zhang* et al. (2020) to measure the diversity of generated counterfactual text samples. In our setting, we compute the Self-BERT score between the generated counterfactual text samples. We term this computed Self-BERT score as Diversity Score. A higher value represents lower diversity and vice-versa.

(c) Content Preservation: Content preservation is used to assess the similarity between generated counterfactual with the input sentence. We compute the similarity between input and generated CF on a sentence level using a pre-trained ML model (Reimers and Gurevych 2019). Since in our setting, we generate K counterfactual text samples, to compute the content similarity, we take a mean of scores over K samples.

(d) Tree-Edit Distance: To measure the syntactic closeness of generated counterfactual text with the original text, we compare their structures using tree-edit distance (Zhang and Shasha 1989). Tree-edit distance is defined as the minimum number of transformations required to turn the constituency parse tree of a counterfactual generation to that of the original text. This metric is required to ensure that while changing the classifier label, we do not deviate too much from the input text structure.

Human Evaluation We define qualitative metrics to evaluate the generation quality of the generated counterfactual text with the Adversarial Attack model. We randomly sample outputs from our model and from the baseline model (totaling to 600 samples). For this purpose, 30 annotators rated the generated counterfactual text. The annotators were kept blind to the model that generated the counterfactual text. We adopt five criteria ranging from 1 to 5 (very poor to excellent): *grammatical correctness, plausibility, fluency, sentiment change, content preservation*. Figure 3 shows the human evaluation results. Among GYC and adversarial baseline, GYC achieves the best results on all metrics. In addition to the four properties of counterfactual text, human evaluation clarifies that GYC is able to maintain high grammatical correctness, plausibility, fluency by changing the sentiment and thereby maintaining the balance in generation.

Results and Discussion

In Table 4, we show the results of GYC and other baselines on three different datasets. We observe that GYC produces counterfactuals with close to the highest Label-Flip percentage across all datasets, and establishes high diversity while maintaining content preservation and syntactic structure as compared to baselines like Checklist, Masked-LM, and PPLM-BoW which rely on token-based substitution criteria.

Performance in flipping the label. In Table 4, we observe that our label flip score is the highest in Yelp and is significantly high in Dbpedia and AgNews. Checklist and Masked-LM perform poorer than ours. This is because they do not have an explicit learning signal to do the generation that guides them to flip the label. Our model is equipped with differentiable label loss which can guide these generations to flip label. Adversarial Attack can flip the label every time, but the generations are grammatically incorrect and not plausible. Hence, they do not fit our task setting. In PPLM-BoW, the flip score is very low in Yelp due to the reason that it does not have an explicit learning signal to flip the label. In case of AgNews and Dbpedia, the flip score is very high for PPLM-BoW as the number of classes is high in both these datasets and often intermixed like “Natural Place”, “Village” are two separate classes in Dbpedia. Hence it is justified to show high flip rate in these datasets.

How diverse are the generated samples. In this metric, we expect that our model provides more diverse samples while maintaining the content of the original text and flipping its label as compared to other models. In Table 4, we observe that GYC outperforms all models on Dbpedia and YELP. In AgNews, our diversity is slightly lower than PPLM-BoW. This is expected because PPLM-BoW generates overly diverse sentences that are often unrelated to the original text.

Preserving Content of Input Text. While we want to generate diverse samples, we also want to make sure that the generations are close to the input text. We expect that Checklist and Masked LM which perform single token-substitution will have high content preservation. Hence, the focus of this metric is not on comparison with these but with PPLM-BoW. In Table 5, we see that the content preservation of PPLM-BoW is significantly lower than GYC. In PPLM-BoW, most of the sentences are overly diverse and unrelated to the original text.

Preserving Syntactic Structure. With this metric, we aim to test that our model does not lose the syntactic structure of the input sentence while flipping the label. In Checklist and Masked-LM the syntactic structure score will be very high since they perform the single token-based substitution. Hence the focus of this metric is not to compare with Checklist and Masked-LM but to compare against PPLM-BoW which also generates free-form text like GYC. In results, we observe that we preserve syntactic structure better than PPLM-BoW and in AgNews our structure preservation is close to that of Checklist.

We show aggregated results of GYC across all metrics and show that GYC outperforms all the models (see results in Appendix C in supplementary material).

Conclusion

In this paper, we propose a framework *GYC*, leveraging GPT-2. We provide the first method that generates test-cases by changing multiple tokens in text without using any rule-based cues. Experimental results on three datasets show that GYC outperforms existing models and generates counterfactual text samples with a high label-flip score which are both plausible and diverse by preserving content and syntactic structure.

A next step would be to improve the reconstruction step, which is currently expensive to run for sentences longer than a certain length. This will significantly improve the automatic counterfactual generation.

References

Aggarwal, A.; Lohia, P.; Nagar, S.; Dey, K.; and Saha, D. 2019. Black box fairness testing of machine learning models. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 625–635.

Dathathri, S.; Madotto, A.; Lan, J.; Hung, J.; Frank, E.; Molino, P.; Yosinski, J.; and Liu, R. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=H1edEyBKDS.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Ebrahimi, J.; Rao, A.; Lowd, D.; and Dou, D. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751* .

Engel, J.; Hoffman, M.; and Roberts, A. 2017. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772* .

Galhotra, S.; Brun, Y.; and Meliou, A. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 498–510.

Gardner, M.; Grus, J.; Neumann, M.; Tafjord, O.; Dasigi, P.; Liu, N.; Peters, M.; Schmitz, M.; and Zettlemoyer, L. 2017. AllenNLP: A deep semantic natural language processing platform, 2018. *arXiv preprint arXiv:1803.07640* .

Garg, S.; Perot, V.; Limtiaco, N.; Taly, A.; Chi, E. H.; and Beutel, A. 2019. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 219–226.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* .

Holstein, K.; Wortman Vaughan, J.; Daumé III, H.; Dudik, M.; and Wallach, H. 2019. Improving fairness in machine learning systems: What do industry practitioners need? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–16.

Howard, J.; and Ruder, S. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146* .

Hu, Z.; Yang, Z.; Liang, X.; Salakhutdinov, R.; and Xing, E. P. 2017. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1587–1596. JMLR. org.

Huang, P.-S.; Zhang, H.; Jiang, R.; Stanforth, R.; Welbl, J.; Rae, J.; Maini, V.; Yogatama, D.; and Kohli, P. 2019. Reducing Sentiment Bias in Language Models via Counterfactual Evaluation. *arXiv preprint arXiv:1911.03064* .

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* .

Jia, R.; and Liang, P. 2017. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328* .

John, P. G.; Vijaykeerthy, D.; and Saha, D. 2020. Verifying Individual Fairness in Machine Learning Models. In *Conference on Uncertainty in Artificial Intelligence*, 749–758. PMLR.

Keskar, N. S.; McCann, B.; Varshney, L. R.; Xiong, C.; and Socher, R. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858* .

Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P.; Hellmann, S.; Morsey, M.; Van Kleef, P.; Auer, S.; and Bizer, C. 2014. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal* 6. doi:10.3233/SW-140134.

Li, J.; Monroe, W.; and Jurafsky, D. 2016. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220* .

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* .

Ma, P.; Wang, S.; and Liu, J. ???? Metamorphic Testing and Certified Mitigation of Fairness Violations in NLP Models .

Madaan, N.; Mehta, S.; Agrawaal, T.; Malhotra, V.; Aggarwal, A.; Gupta, Y.; and Saxena, M. 2018. Analyze, detect and remove gender stereotyping from bollywood movies. In *Conference on Fairness, Accountability and Transparency*, 92–105.

Michel, P.; Li, X.; Neubig, G.; and Pino, J. M. 2019. On evaluation of adversarial perturbations for sequence-to-sequence models. *arXiv preprint arXiv:1903.06620* .

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781* .

Mothilal, R. K.; Sharma, A.; and Tan, C. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 607–617.

Pearl, J.; et al. 2000. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress* .

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1(8): 9.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. URL http://arxiv.org/abs/1908.10084.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2018. Anchors: High-Precision Model-Agnostic Explanations. In *AAAI*, volume 18, 1527–1535.

Ribeiro, M. T.; Wu, T.; Guestrin, C.; and Singh, S. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. *arXiv preprint arXiv:2005.04118* .

Shen, T.; Lei, T.; Barzilay, R.; and Jaakkola, T. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in neural information processing systems*, 6830–6841.

Udeshi, S.; Arora, P.; and Chattopadhyay, S. 2018. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 98–108.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 5998–6008. Curran Associates, Inc. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Wachter, S.; Mittelstadt, B.; and Russell, C. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31: 841.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R. R.; and Le, Q. V. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, 5753–5763.

Ye, R.; Shi, W.; Zhou, H.; Wei, Z.; and Li, L. 2020. Variational Template Machine for Data-to-Text Generation. *arXiv preprint arXiv:2002.01127* .

Yu, L.; Zhang, W.; Wang, J.; and Yu, Y. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-first AAAI conference on artificial intelligence*.

Zhang, K.; and Shasha, D. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing* 18(6): 1245–1262.

Zhang*, T.; Kishore*, V.; Wu*, F.; Weinberger, K. Q.; and Artzi, Y. 2020. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=SkeHuCVFDr.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 649–657.

Zhang, Y.; Gan, Z.; Fan, K.; Chen, Z.; Henao, R.; Shen, D.; and Carin, L. 2017. Adversarial feature matching for text generation. *arXiv preprint arXiv:1706.03850* .

Zhao, Z.; Dua, D.; and Singh, S. 2017. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342* .

Zhu, Y.; Lu, S.; Zheng, L.; Guo, J.; Zhang, W.; Wang, J.; and Yu, Y. 2018. Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 1097–1100.

Appendix A: Model Details

Our model consists of multiple losses to generate counterfactuals with four properties. In this section we will describe the implementation details with hyperparameters required for different losses.

Reconstruction Loss Implementation

In reconstruction loss, we tried to regenerate the input sentence such that we can perturb the hidden state of input for generation of counterfactuals. While this is straightforward, while using GPT-2 the inference is a hard task.

We explain the ways we do inference on GPT-2.

Method 1: Naive Training In naive training, we try to maximize the presence of every given x_i in the sentence all at once right from the beginning of the training.

$$\text{Loss} = \sum_{t=1}^{T} \log\ p\left(x_t | \tilde{H}_{<t}\right)$$

The main problem with this approach has been that the gradient is maximum at the last word/ hidden state. Hence it gets stuck in local minima. Therefore the remaining sentence can not be reconstructed due to weak gradients. So we could not adopt this for reconstruction loss.

Method 2: Curriculum Training In curriculum training, we try to maximize the presence of given x_i in the sentence in left to right order. So we first try to reconstruct x_1 then x_2 and so on.

$$\text{Loss} = \sum_{t=1}^{K} \log\ p\left(x_t | \tilde{H}_{<t}\right)$$

where K is chosen according to the number of iterations.

The main problem with this approach has been the length of the each phase is hard-coded which makes it very slow and empirically performs poorly.

Method 3: Adaptive Curriculum Training (This is the working solution)

$$\text{Loss} = \sum_{t=1}^{K} \log\ p\left(x_t | \tilde{H}_{<t}\right)$$

where K is chosen on the basis of how many words are already reconstructed

In adaptive curriculum training, we try to maximize the presence of given x_i in the sentence in left to right order. So we first try to reconstruct x_1 then x_2 and so on.

We go to the next phase of curriculum as soon as previous word is regenerated. If previous word goes out of the construction, then we roll-out of the curriculum. This helps in doing the reconstruction of the input sentence in a reliable way. Hence we follow this approach in the paper.

While reconstruction is of utmost importance when we perturb to generate $\mathbf{x}$, we control the strength of reconstruction or proximity loss by assigning a proximity_loss_weight. This weight is set to 1 during initial phases of training. After the sentence has been reconstructed, this weight is set to 0.02 so that perturbation can effective and not regenerate the original sentence.

Differentiable Loss Implementation

For the differentiable loss, we train a sentiment model on different datasets used in our experiments. The input to sentiment model is logits (p_v) of the generated text after reconstruction step ($\tilde{\mathbf{H}}$). In order to have this work with GPT-2, the vocabulary of sentiment model and GPT-2 has to be consistent.

For this purpose, we choose our sentiment model to be composed of an EmbeddingBag layer followed by a linear layer. We take the mean value of the bag-of-embeddings (e_v) over the whole vocabulary, v. We represent the logits of classifier as follows -

$$\text{logits} = \text{MLP}\left(\sum_t \sum_v p_v^t e_v\right)$$

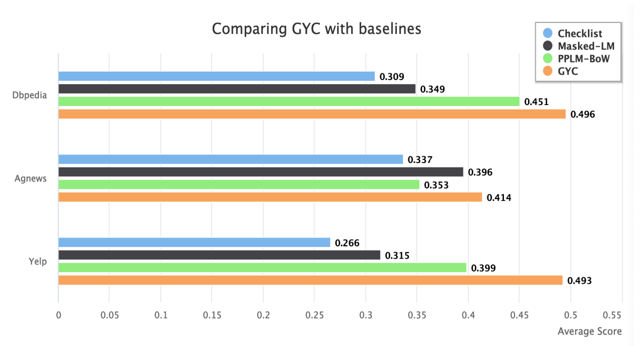

Figure 3: Comparison of GYC with existing approaches on average score of all four metric.

Reward Loss Implementation

For thee reward loss implementation, we generate reward for non-differentiable score function. The implementation is done for capturing reward based-on NER, SRL and Sentiment Label-based.

- We compute reward for each sentence x^k.
  - NER-based reward is computed as $p\left(\text{NER-tag } in\ x^k\right)$
  - SRL-based reward is computed as $p\left(\text{SRL-tag } in\ x^k\right)$
  - Label-based reward is computed as $p\left(\text{Label-tag } in\ x^k\right)$

Entropy

To generate samples with enough diversity, we employ entropy. To control entropy, we use an entropy_weight parameter. In our experiments entropy is set to 0.08.

Input Sentence	**GYC**	**Checklist**	**Masked-LM**	**PPLM-BoW**
I would recommend this place again	I would **not** recommend this place	I would recommend this **site** again	I would recommend your **place** again	I would **like to thank** this **user**
The wait staff is lazy and never asks if we need anything	The wait staff and **volunteers are always ready to offer their assistance**	The wait **list** is lazy and never asks if we need anything	The wait staff is lazy and never asks if we need **?**	The wait **is over - it's a real thing. You will receive**
1st time burnt pizza was horrible	1st time burnt pizza **is delicious**	1st time burnt pizza was **eaten**	1st time burnt pizza was **sold**	1st time, **this is a great place**
It is extremely inconvenient	It is exceedingly **easy**	It is **not** inconvenient	It is extremely **beautiful**	It is **very hard to tell**
The beer is awful	The beer is **amazing**	The beer is **delicious**	The beer **smelled** awful	The beer **is a blend of**
The return policy here is ridiculous	The return policy here is **simple**	The return **fee** here is ridiculous	The return policy **which** is ridiculous	The return **of a legendary, but rarely**
I have a great experience here	I have this **dish**	I **got** a great experience here	I have **one** great experience here	I have **the following issue with the latest**
You should give this place zero stars	You should **be able to** give stars	You should give this **game** zero stars	You should give **the** place zero stars	You should **read it. This is an extremely**
Our waitress was very friendly	Our waitress was **not amused**	Our **waiter** was very friendly	Our waitress was very **!**	Our waitress was **super friendly and friendly**
I had a great experience here	I had **to write about this**	I had a great experience **there**	I had a great **time** here	I had a **great time on this one**

Table 6: Illustration of samples generated by GYC in comparison with Checklist, Masked-LM and PPLM-BoW

Appendix B: Baselines Implementation

Checklist In Checklist, we use the LM based perturbations employing RoBERTa model. We generate a word by choosing [MASK] at a random location in the input text. This mask is chosen after ignoring the first three tokens of the input sentence, to make sure that we mask a relevant part of the sentence for replacement.

Appendix C: Results

Quantitative Results

We aggregate the results of GYC averaged over all four metrics to assess the performance with baselines in Figure 3. We observe that, GYC has the highest average score compared to the baselines on all datasets.

Qualitative Results

We show generated counterfactual samples from GYC and the baselines for comparison in Table 6.