Does Training on Adversarial Counterexamples Improve Generalization?

Anonymous ACL submission

Abstract

Neural models often exploit superficial heuristics in order to achieve good performance, rather than deriving robust features to solve the underlying task. Overcoming this tendency is a central challenge in areas such as representation learning and ML fairness. *Adversarial* or *counterfactual* data augmentation involves generating training examples where these heuristics fail, in order to encourage the model to use more general features. We design a series of toy learning problems to explore the conditions under which adversarial data augmentation yields a more robust model. We show that, often, the added training examples help prevent the model from adopting the targeted heuristic, but do not help it learn more general features. We also find in many cases that the number of adversarial examples needed to reach a given error rate is independent from the amount of training data, and that adversarial data augmentation becomes less effective as the number of available heuristics increases and/or as the underlying learning problem becomes more challenging.

1 Introduction

Neural models often achieve good task performance by exploiting spurious correlations and "heuristics" rather than learning deeper, more robust features. A wave of recent studies in particular has exposed such weaknesses by showing how dramatically models fail when evaluated on *adversarial* or *counterfactual examples*–that is, inputs with feature co-occurrences that appear infrequently in the model's training set. For example, in visual question answering, models failed when tested on rare color descriptions (*"green bananas"*) (Agrawal et al., 2018); in coreference, models failed when tested on rare profession-gender pairings (*"the nurse cared for his patients"*) (Rudinger et al., 2018); in natural language inference (NLI), models failed on sentence pairs with high lexical overlap with different meanings (*"man bites dog"*/*"dog bites man"*) (McCoy et al., 2019).

One proposed solution has been to augment training data to over-represent these tail events, often with automatic or semi-automatic methods for generating such challenge examples at a large scale. This technique has been discussed for POS tagging (Elkahky et al., 2018), NLI (McCoy et al., 2019), and as a means of reducing gender bias (Zhao et al., 2018; Zmigrod et al., 2019), all with positive initial results. However, it is difficult to know whether this strategy is a feasible way of building more robust systems in general. When applied in practical settings, it is difficult to assess whether the data augmentation leads the model to make decisions for the "right" reasons or merely causes it to switch to a different set of equally-shallow heuristics other than those targeted by the added examples. Understanding this tradeoff–that is, understanding if and when changes in the training distribution cause a model to switch from shallow heuristics to deeper features–is important for both practical and theoretical work in NLP.

We design a series of toy learning problems to explore whether adversarial data augmentation is a feasible way of improving model robustness. We consider a simple neural network classifier in a task that is typical in NLP: 1) the labeled input data is not i.i.d., the model can exploit extant, spurious correlations and 2) the model is trained end-to-end and thus adopts whichever feature representation performs best. Our research questions are:

- How many counterexamples must be seen in training to prevent the model from adopting a given heuristic? Do larger training sets require more counterexamples or fewer?
- How does the difficulty of representing the "right" feature affect the model's tendency to adopt heuristics?
- When many spurious correlations exist, do models prefer multiple imperfect heuristics or fewer, more reliable features?

2 Experimental Setup

2.1 Intuition

Our study is motivated by two empirical findings presented in McCoy et al. (2019). Specifically, McCoy et al. (2019) focused on models' use of syntactic heuristics in the context of the natural language inference (NLI) task: given a pair of sentences–the premise p and the hypothesis h–predict whether or not p entails h. They showed that when 1% of the 300K sentence pairs seen in training exhibit lexical overlap (i.e. every word in h appears in p) and 90% of lexical-overlap sentence pairs have the label ENTAILMENT, the model adopts the (incorrect) heuristic that lexical overlap always corresponds to ENTAILMENT. However, after augmenting the training data with automatically generated training examples so that 10% of the 300K training pairs exhibit lexical overlap and 50% of lexical-overlap sentence pairs have the label ENTAILMENT, the same model did *not* adopt the heuristic and appeared to learn features which generalized to an out-of-domain test distribution.

From these results, it is hard to say which changes to the training setup were most important for the model's improved generalizability. The number of lexical-overlap examples seen in training? The probability of ENTAILMENT given that a pair exhibits lexical overlap? Or some other positive artifact of the additional training examples? Thus, we abstract away from the specifics of the NLI task in order to consider a simplified setting that captures the same intuition but allows us to answer such questions more precisely.

2.2 Assumptions and Terminology

We consider a binary sequence classification task. We assume there exists some feature which directly determines the correct label, but which is non-trivial to extract given the raw input. (In NLI, the analogous feature is whether the semantic meaning of p entails that of h). We refer to this feature as the **true property**. Additionally, we assume the input contains one or more **distractor properties** which are easy for the model to extract from the input. (This is analogous to lexical overlap between p and h). In our set up, the correct label is 1 if and only if the true property holds.[1] However, the true and distractor properties frequently co-occur in training and thus a model which only represents the distractor properties will be able to make correct predictions much of the time. We can vary the strength of their co-occurance by adding **counterexamples** to the training data which contain either the true property or the distractor property but not both. This setup is shown in Figure 1.

[1]See Appendix B.1 for results which assume varying levels of label noise.

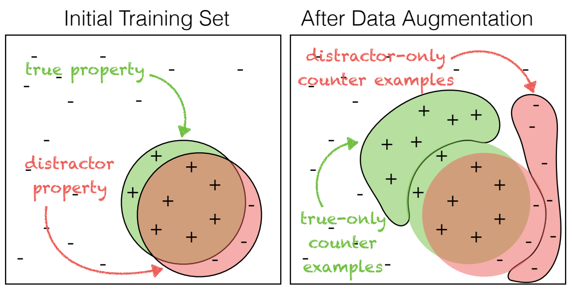


Figure 1: Schematic of experimental setup.

2.3 Implementation

Task. We use a synthetic sentence classification task with sequences of numbers as input and binary $\{0, 1\}$ labels as output. We use a symbolic vocabulary V consisting of the integers $0 \ldots |V|$. In all experiments, we use sequences of length 5 and set $|V|$ to be 50K. We do see some effects associated with vocabulary size, but none that affect our primary conclusions; see Appendix A.1 for details.

Model. We use a simple network comprising an embedding layer, a 1-layer LSTM, and a 1-layer MLP with a RELU activation. We found enough interesting trends to analyze in the behavior of this model, and thus leave experiments with more complex model architectures for future work. Our code, implemented in PyTorch, is released for reproducibility.[2] Appendix A discusses how our experimental results extend across changes in model and task parameters. All models are trained until convergence, using early-stopping.[3]

True and Distractor Properties. In all experiments, we set the distractor property to be the presence of the symbol `2` anywhere in the input. We consider several different properties as instantiations of the true property, listed in Table 1. These properties are chosen with the intent of varying how difficult the true property is to detect given the raw sequential input. In all experiments, we design train and test splits such that the symbols which are used to instantiate the true property during training are never used to instantiate the true property during testing. For example, for experiments using `adjacent duplicate`, if the model sees the string `1 4 3 3 15` at test time, we enforce that it never saw any string with the duplicate `3 3` during training. This is to ensure that we are measuring whether the model learned the desired pattern, and did not simply memorize bigrams.

[2]http://bit.ly/counterexamples

[3]The results shown here used the test error for early-stopping; we have re-ran some of our experiments with early-stopping on validation error, and it did not make a difference.

Property Nickname	Description	Loss AUC	Example
`contains 1`	1 occurs somewhere in the sequence	0.50	2 4 11 **1** 4
`prefix duplicate`	Sequence begins with a duplicate	0.52	**2 2** 11 12 4
`first-last duplicate`	First number equals last number	0.55	**2** 4 11 12 **2**
`adjacent duplicate`	Adjacent duplicate is somewhere in the sequence	1.43	11 12 **2 2** 4
`contains first`	First number is elsewhere in the sequence	1.54	**2** 11 **2** 12 4

Table 1: Properties used to instantiate the true property in our experimental setup. Properties are intended to differ in how hard they are for an LSTM to detect given sequential input. We use the the AUC of the validation loss curve as a heuristic measure of "hardness", as described in Section 2.3.

To quantify the difficulty of representing each true property, we train the model for the task of predicting directly whether or not the property holds for each of our candidate properties, using a set of 200K training examples evenly split between cases when the property does and does not hold. For each property, Figure 2 shows the validation loss curve (averaged over three runs), flatlined at its minimum (i.e. its early stopping point). We see the desired gradation in which some properties require significantly more training to learn than others. As a heuristic metric of "hardness", we use the approximate area under this flat-lined loss curve, computed by taking the sum of the errors across all epochs. Table 1 contains the result for each property. Note that the distractor property (the sequence contains 2) is exactly as hard as `contains 1`.

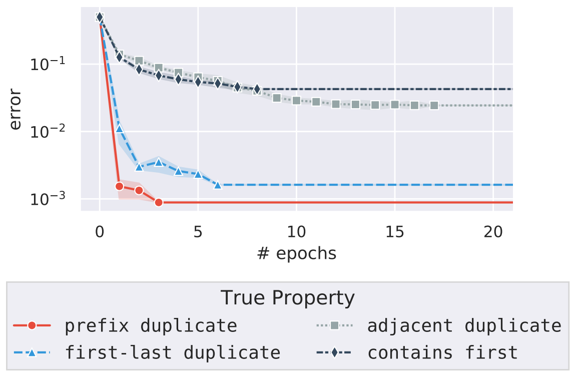


Figure 2: Learning curve for classifying whether the properties hold. Averaged over three reruns and then flat-lined after the average reaches its minimum. Train size is 200K. The error for `contains 1` and the distractor property (not shown) reaches 0 after the first epoch.

2.4 Error Metrics

Definition. We partition test error into four regions of interest, defined below. We use a finer-grained partition than standard true positive rate and false positive rate because we are particularly interested in measuring error rate in relation to the presence or absence of the distractor property.

distractor-only:	$P(pred = 1 \mid dist, \neg true)$
true-only:	$P(pred = 0 \mid \neg dist, true)$
both:	$P(pred = 0 \mid dist, true)$
neither:	$P(pred = 1 \mid \neg dist, \neg true)$

For completeness, we define and compute *both* error and *neither* error. However, in practice, since our toy setting contains no spurious correlations other than those due to the distractor property, we find that these two error metrics are at or near zero for all of our experiments (except for a small number of edge cases discussed in §C). Thus, for all results in Section 3, we only show plots of *true-only* and *distractor-only* error, and leave plots of the others to Appendix C.

Interpretation. Intuitively, high *distractor-only* error indicates that the model is associating the distractor property with the positive label, whereas high *true-only* error indicates the model is either failing to detect the true property altogether, or is detecting it but failing to associate it with positive label. In practice, we might prioritize these error rates differently in different settings. For example, within work on bias and fairness (Hall Maudslay et al., 2019; Zmigrod et al., 2019), we are primarily targeting *distractor-only* error. That is, the primary goal is to ensure that the model does not falsely associate protected attributes with specific labels or outcomes. In contrast, in discussions about improving the robustness of NLP more generally (Elkahky et al., 2018; McCoy et al., 2019), we are presumably targeting *true-only* error. That is, we are hoping that by lessening the effectiveness of shallow heuristics, we will encourage models to learn deeper, more robust features in their place.

2.5 Data Augmentation

Definition. Adversarial data augmentation aims to reduce the above-described errors by generating new training examples which decouple the true and distractor properties. Parallel to the error categories, we can consider two types of counterexamples: ***distractor-only* counterexamples** in which the distractor property occurs without the true property and the label is 0, and ***true-only***

counterexamples in which the true property occurs without the distractor and the label is 1.

Interpretation. Again, in terms of practical interpretation, these two types of counterexamples are meaningfully different. In particular, it is likely often the case that *distractor-only* counterexamples are easy to obtain, while constructing *true-only* counterexamples is more cumbersome. For example, considering again the case of NLI and the lexical overlap heuristic from McCoy et al. (2019), it is easy to artificially generate *distractor-only* counterexamples (p/h pairs with high lexical overlap but which are not in an entailment relation) using a set of well-designed syntactic templates. However, generating good *true-only* counterexamples (entailed p/h pairs without lexical overlap) is likely to require larger scale human effort (Williams et al., 2018). This difference would likely be exacerbated by realistic problems in which there are many distractor properties which we may want to remove and/or it is impossible to fully isolate the true property from all distractors. For example, it may not be possible to decouple the "meaning" of a sentence from all lexical priors. We explore the case of multiple distractors in Section 3.4.

2.6 Limitations

This study is intended to provide an initial framework and intuition for thinking about the relationships between data augmentation and model robustness. We simplify the problem significantly in order to enable controlled and interpretable experiments. As a result, the problems and models studied are many steps removed from the what NLP looks like in practice. In particular, we consider a very small problem (binary classification of sequences of length 5 in which only two variable are not independent) and a very small model (a simple LSTM). Although we provide many additional results in the Appendix to show consistency across different model and task settings, we still do not claim that the presented results would necessarily hold for more complex models and tasks–e.g. multi-layer transformers performing language modeling. Clearly many details of our setup–e.g. which properties are easier or harder to detect–would change significantly if we were to change the model architecture and/or training objective to reflect more realistic settings. Exploring whether the overarching trends we observe still hold given such changes would be exciting follow up work.

We also make the assumption that the input to the model contains a single causal "true property" which, once extracted, perfectly explains the output. For many of the tasks we currently study in NLP, this assumption arguably never holds, and rather, models only ever get access to correlates of the true property. That is, they see text descriptions but never the underlying referents. Thus, for a task like NLI, it might be that the best our models can do is find increasingly robust correlates of "meaning", but may not ever extract "meaning" itself. This is a much larger philosophical debate on which we do not take a stance here. We let it suffice that, in practice, the presented results are applicable to any task in which there exists some deeper more robust feature(s) that we prefer to the model to use and some shallower correlated feature(s) which it may chose to exploit instead, whether or not those deeper features are in fact the "true" feature that determines the label.

3 Results and Discussion

Our primary research questions, reframed in terms of the above terminology, are the following. First, how many counterexamples are needed in order to reduce the model's prediction error? In particular, how is this number influenced by the hardness of the true property (§3.1), the type of counterexamples added (i.e. *true-only* vs. *distractor-only*)(§3.2), and the size of the training set (§3.3)? Second, given a setting in which multiple distractor properties exist, does the model prefer to make decisions based on multiple weak heuristics, or rather to extract a single more robust feature (§3.4)? We report all results in terms of *true-only* error and *distractor-only* error; results for other error categories are given in Appendix C.

3.1 Effect of True Property's Hardness

We first consider prediction error as a function of the number of counterexamples added and the hardness of detecting the true property (with "hardness" defined as in Section 2.3). To do this, we construct an initial training set of 200K examples in which there is perfect co-occurrence between the true and distractor properties, with the dataset split evenly with positive examples (e.g. has both true and distractor properties) and negative examples (with neither property). We then vary the number of counterexamples added[4] from 10 ($\lll 0.1\%$ of the training data) to 100K (33% of the training data) and measure the effect on *true-only* and *distractor-only* error. For now, we as-

[4]The results presented assume that the counterexamples are added on top of the training data that is already there, and thus models trained with larger numbers of counterexamples have a slightly larger total training size. We also experimented with adding counterexamples in place of existing training examples so that the total training size remains fixed across all runs. We do not see a meaningful difference in results. Results from the constant-training-size experiments are given in Appendix D.2. We also perform a control experiment to verify that the additional number of training examples itself is not impacting the results in Appendix D.1.

sume that the counterexamples added are evenly split between *true-only* and *distractor-only* types.

Figure 3 shows the results. We see that the number of counterexamples needed is substantially influenced by the hardness of the true property. For example, after only 10 counterexamples are added to training, test error has dropped to near zero when the true property is `contains 1` (which is trivial for the model to detect) but remains virtually unchanged for the `contains first` and `adjacent duplicate` properties. For the harder properties, we don't reach zero error until a third of the training data is composed of counterexamples.

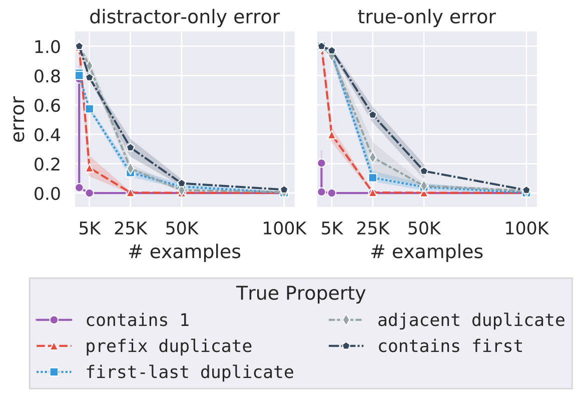


Figure 3: The harder the true property, the more counterexamples the model requires to reduce a given error rate. The legend shows models in order from hardest to least hard, where "hardness" is determined by the AUC of the loss curve for a classifier trained to detect the property (§2.3). We run all experiments over five random seeds. In all plots, the error band is the 95% confidence interval with 1,000 bootstrap iterations.

3.2 Effect of Counterexample Type

We get a better understanding of the model's behavior when look separately at *true-only* and *distractor-only* errors, considering each as a function of the number and the type (e.g. *true-only* vs. *distractor-only*) of counterexamples added (Figure 4). We see that adding *distractor-only* counterexamples leads to improvements in *distractor-only* error, but has minimal effect on *true-only* error. The interesting exception to this is when the true property is no harder to represent than the distractor property. In our setting, this happens when the true property is `contains 1`, but it is unclear if this pattern would hold for other distractors. In this case, we see that both *true-only* and *distractor-only* error fall to zero after adding only a small number of *distractor-only* counterexamples.

In contrast, we see some evidence that adding *true-only* counterexamples has impact on

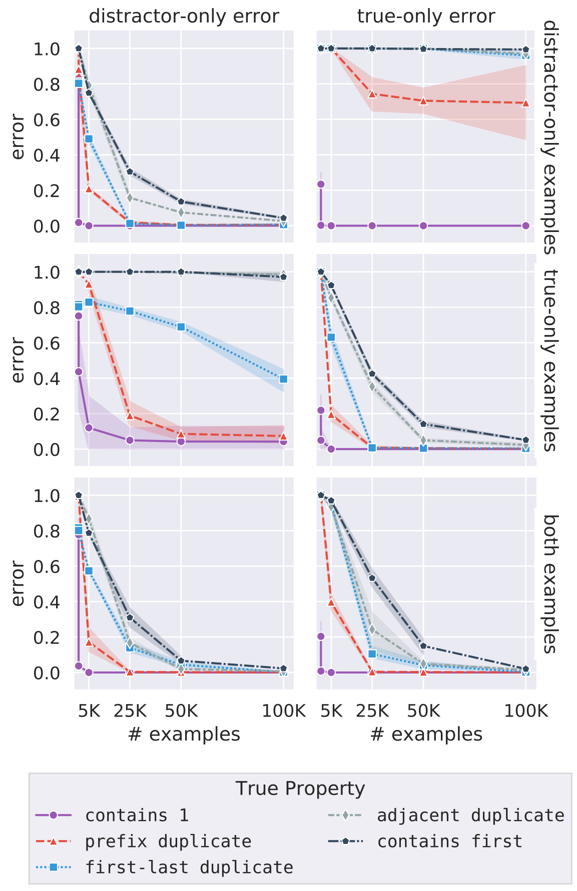


Figure 4: Adding *distractor-only* counterexamples improves *distractor-only* error but does not affect *true-only* error. In contrast, adding *true-only* counterexamples may lead to improvements of both error types (see text for discussion).

distractor-only error as well as on *true-only* error. The impact on *distractor-only* error, however, is limited to the settings in which the true property is sufficiently easy to detect. That is, when the true property is difficult to detect, adding *true-only* counterexamples *does* lead the model to detect the true property and correctly associate it with the positive label, but *does not* necessarily lead the model to abandon the initially-adopted (incorrect) heuristic that the distractor property is also predictive of a positive label. This behavior is interesting, since any correct predictions made using the heuristic are by definition redundant with those made using the true property, and thus continuing to hold the heuristic can only hurt performance.

3.3 Effect of Training Data Size

In Section 3.1, we observed that, for most of our properties, the model did not reach near zero error until 20% or more of its training data was composed of counterexamples. This raises the question: is error rate better modeled in terms of the absolute number of counterexamples added,

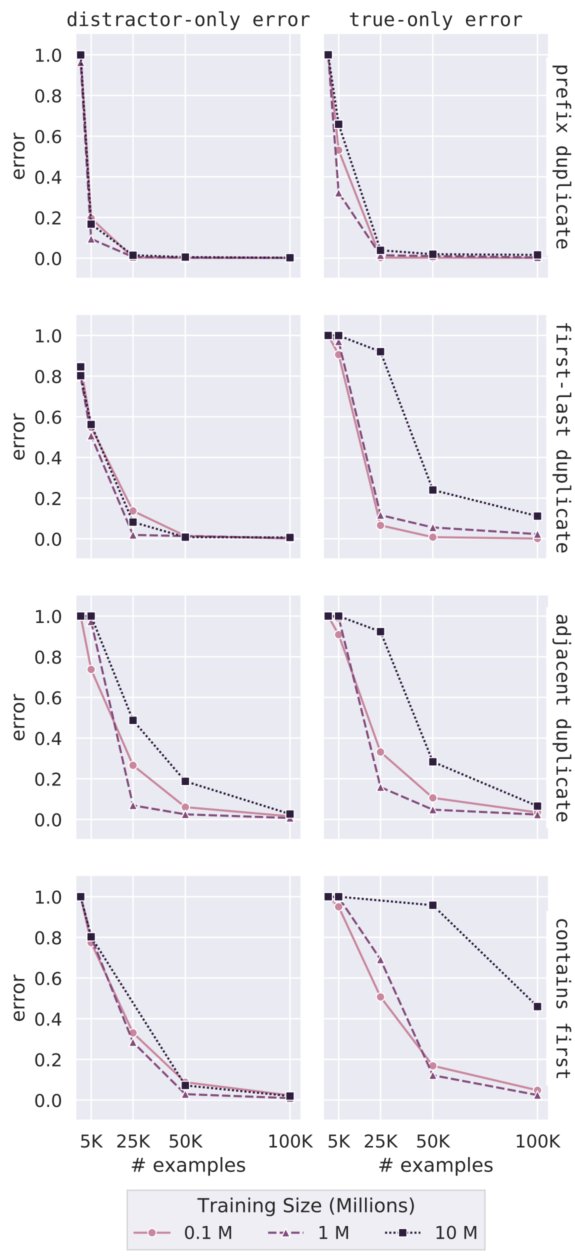


Figure 5: Error rate vs. number of counterexamples added for models trained with different initial training set sizes (100K to 10M). Property `contains 1` not shown since it is always learned instantly. In general, increasing the training size while holding the number of counterexamples fixed does not significantly affect the efficacy of those counterexamples, although there are exceptions (see text for discussion). We do not average over all the random seeds for experiments with dataset size because of the computational resources.

or rather the fraction of the training data that those counterexamples make up? For example, does adding 5K counterexamples produce the same effect regardless of whether those 5K make up 5% of a 100K training set or 0.05% of a 10M training set? Our intuition motivating this experiment is that larger initial training sets might "dilute" the signal provided by the comparably small set of counterexamples, and thus larger training sets might require substantially more counterexamples to achieve the same level of error.

In Figure 5, we again show both *distractor-only* and *true-only* error as a function of the number of counterexamples added to training, but this time for a range of models trained with different initial training set sizes. Here, for simplicity, we again assume that the counterexamples added are evenly split between the *true-only* and *distractor-only* types. Generally speaking, for most properties, we see that our intuition does not hold. That is, increasing the training size while holding the number of counterexamples fixed does not substantially effect either error metric, positively or negatively. That said, there are several noteworthy exceptions to this trend. In particular, we see that when the training set is very large (10M) relative to the number of counterexamples ($<$ 50K), the efficacy of those counterexamples does appear to decrease. We also again see differences depending on the true property, with the harder properties (`contains first` and `adjacent duplicate`) behaving more in line with the "diluting" intuition described above than the easier properties (`first-last duplicate` and `prefix duplicate`).

3.4 Multiple Distractor Properties

In our experiments so far, we have assumed that there is only one distracting property, which is clearly an unrealistic approximation of natural language data. We therefore relax this assumption and consider the setting in which there are multiple distractor properties which are correlated with the label. The question is: does the option of minimizing loss by combining multiple (imperfect) heuristics lessen the model's willingness to extract deeper, more general features?

Our experimental setup is as follows. We assume k distractor properties $d_1, \dots d_k$, each of which is correlated with the true property t, but which are not correlated with each other beyond their correlation with t. As in previous experiments, $P(d_i|t)$ and $P(t|d_i)$ are determined by the number of *true-only* and *distractor-only* counterexamples that have been added, respectively. In the initial training set, before any counterexamples have been added, where $P(t|d_i)$ is 1, and $P(d_i|t)$ is close to $1/2$[5]. We assume that *distractor-only* counterexamples can be generated in the same way as before,

[5]We independently sample each property with probability 1/2 but then resample if we sample no distractors. This is done such that the reported number of counterexamples is correct.

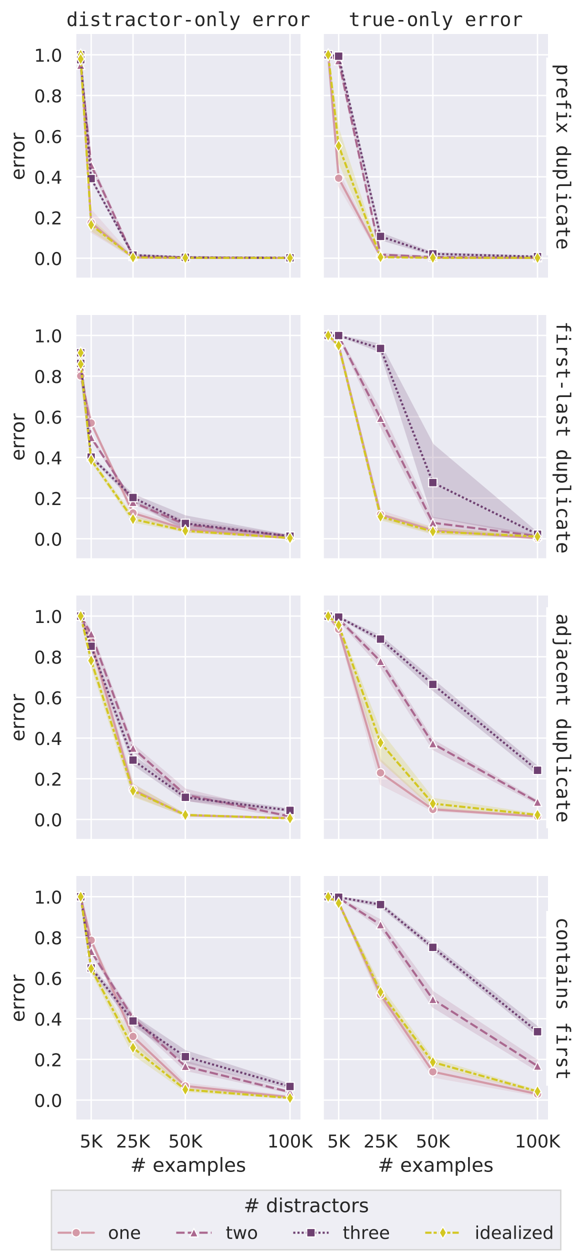


Figure 6: Error rate vs. number of counterexamples added for models trained in settings with different numbers of distractors, assuming that *true-only* counterexamples can guarantee the removal of at most one distractor at a time. Initial training size is 200K. Gold line shows performance in an idealized setting in which there are 3 distractors but *true-only* counterexamples are "pure", i.e. free from all distractors. When more distractors are present, we see higher *true-only* error, suggesting that, when models unlearn the targeted heuristics, they often switch to using new heuristics rather than using the true property.

but that good *true-only* counterexamples cannot be generated perfectly. For intuition, again consider the NLI setting. It is easy for a person to generate a p/h pair with the label CONTRADICTION, but doing so while meeting the constraints that the sentences exhibit lexical overlap (McCoy et al., 2019) but do not contain any antonym pairs (Dasgupta et al., 2018) or explicit negations (Gururangan et al., 2018) would likely lead to unnatural and unhelpful training sentences. Thus, we assume that we can only reliably remove at most one[6] distractor property at a time. Specifically, we assume that, for a given distractor property d_i, we can generate *true-only*$_i$ counterexamples which 1) definitely exhibit the true property, 2) definitely do not exhibit d_i, and 3) exhibit every other distractor d_j with probability $P(d_j|t)$.

We again plot *true-only* and *distractor-only* error as a function of the number of counterexamples added, assuming we add counterexamples which are evenly split between *distractor-only* and *true-only* types and evenly split among the k distractors. Note that, while our *true-only* counterexamples are not "pure" (i.e. they might contain distractors other than the one specifically targeted), our *true-only* error is still computed on examples that are free of *all* distractors. Our *distractor-only* error is computed over examples that contain no true property and at least one distractor.

The results are shown in Figure 6 for $k = \{1, 2, 3\}$.[7] For comparison, we also plot the performance of a model trained in an idealized setting in which "pure" *true-only* counterexamples are added. We see that, holding the number of counterexamples fixed, the more distractors there are, the higher the *true-only* error tends to be. We see also that, in an idealized setting in which it is possible to generate "pure" *true-only* counterexamples, this trend does not hold, and there is no difference between the single distractor and the multiple distractor settings. In terms of *distractor-only* error, we see only small increases in error as the number of distractors increases.

Taken together, our interpretation of these results is that access to both *distractor-only* counterexamples and to imperfect *true-only counterexamples* is sufficient for the model to unlearn bad heuristics–that is, the model does learn that none of the d_i alone causes a positive label, and thus is able to achieve effectively zero *distractor-only* error. However, as evidenced by the substantially higher *true-only* error, the model does not appear to learn to use the true property, but rather appears to adopt new heuristics in place of the targeted heuristics–e.g. perhaps the conjunction of

[6]We ran experiments in which it was possible to remove more than one distractor at a time (but still fewer than k) and did not see interestingly different trends from the at-most-one setting.

[7]Given our sequence length of 5, we were unable to generate results for larger values of k without interfering with our ability to include the true property.

distractor properties $(d_1 \wedge d_2) \vee (d_2 \wedge d_3) \vee (d_1 \wedge d_3)$. Its worth acknowledging that the model appears to learn some information about the true property, as we do see that *true-only* error falls as we add counterexamples, just more slowly for larger values of k. This trend might be taken as evidence that the high *true-only* error is not due to a failure to detect the property (as was the case in Figure 4 when only *distractor-only* errors were added), but rather a failure to consistently associate the true property with the positive label.

4 Related Work

4.1 Adversarial Data Augmentation

A wave of recent work has adopted a strategy of constructing evaluation sets composed of "adversarial examples" (a.k.a. "challenge examples" or "probing sets") in order to analyze and expose weaknesses in the decision procedures learned by neural NLP models (Jia and Liang, 2017; Glockner et al., 2018; Dasgupta et al., 2018, and others). Our work is motivated by the subsequent research that has begun to ask whether adding such challenge examples to a model's training data could help to improve robustness. This work has been referred to by a number of names including "adversarial", "counterfactual", and "targeted" data augmentation. In particular, Liu et al. (2019) show that fine-tuning on small challenge sets can sometimes (though not always) help models perform better. Similar approaches have been explored for handling noun-verb ambiguity in syntactic parsing (Elkahky et al., 2018), improving NLI models' handling of syntactic (McCoy et al., 2019) and semantic (Poliak et al., 2018) phenomena, and mitigating gender biases in a range of applications (Zmigrod et al., 2019; Zhao et al., 2018, 2019; Hall Maudslay et al., 2019; Lu et al., 2018).

4.2 Encoding Structure in NLP Models

Another related body of work focuses on understanding what types of features are extracted by neural language models, in particular looking for evidence that SOTA models go beyond bag-of-words representations and extract "deeper" features about linguistic structure. Work in this vein has produced evidence that pretrained language models encode knowledge of syntax, using a range of techniques including supervised "diagnostic classifiers" (Tenney et al., 2019; Conneau et al., 2018; Hewitt and Manning, 2019), classification performance on targeted stimuli (Linzen et al., 2016; Goldberg, 2019), attention maps/visualizations (Voita et al., 2019; Serrano and Smith, 2019), and relational similarity analyses (Chrupała and Alishahi, 2019). Our work contributes to this literature by focusing on a toy problem and asking under what conditions we might expect deeper features to be extracted, focusing in particular on the role that the training distribution plays in encouraging models to learn deeper structure. Related in spirit to our toy data approach is recent work which attempts to quantify how much data a model should need to learn a given deeper feature (Geiger et al., 2019). Still other related work explores ways for encouraging models to learn structure which do not rely on data augmentation, e.g. by encoding inductive biases into model architectures (Bowman et al., 2015; Andreas et al., 2016) in order to make "deep" features more readily extractable, or by designing training objectives that incentivize the extraction of specific features (Swayamdipta et al., 2017; Niehues and Cho, 2017). Exploring the effects these modeling changes on the results presented in this paper is an exciting future direction.

4.3 Generalization of Neural Networks

Finally, this work relates to a still larger body of work in which aims to understand feature representation and generalization in neural networks in general. Mangalam and Prabhu (2019) show that neural networks learn "easy" examples (as defined by their learnability by shallow ML models) before they learn "hard" examples. Zhang et al. (2016) and Arpit et al. (2017) show that neural networks with good generalization performance can nonetheless easily memorize noise of the same size, suggesting that, when structure does exist in the data, models might have some inherent preference to learn general features even though memorization is an equally available option. Zhang et al. (2019) train a variety of over-parameterized models on the identity mapping and show that some fail entirely while others learn a generalizable identify function, suggesting that different architectures have different tendencies for learning structure vs. memorizing. Finally, there is ongoing theoretical work which attempts to characterize the ability of over-parameterized networks to generalize in terms of complexity (Neyshabur et al., 2019) and implicit regularization (Blanc et al., 2019).

5 Conclusion

We present an experimental framework for exploring the effects of adversarial data augmentation on model robustness. Our results suggest that adding counterexamples in order to encourage a model to "unlearn" bad heuristics is likely to have the immediately desired effect (the model will unlearn the heuristic) but unlikely to lead the model to learn deeper, more general features in its place. We see also that data augmentation may become less effective as the underlying "true" features become more difficult to extract and as the number of spurious correlations in the data increases.

References

Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. 2018. Don't just assume; look and answer: Overcoming priors for visual question answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48.

Devansh Arpit, Stanislaw Jastrzkebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. 2017. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 233–242. JMLR.org.

Guy Blanc, Neha Gupta, Gregory Valiant, and Paul Valiant. 2019. Implicit regularization for deep neural networks driven by an ornstein-uhlenbeck like process. *arXiv preprint arXiv:1904.09080*.

Samuel R. Bowman, Christopher Potts, and Christopher D. Manning. 2015. Recursive neural networks can learn logical semantics. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 12–21, Beijing, China. Association for Computational Linguistics.

Grzegorz Chrupała and Afra Alishahi. 2019. Correlating neural and symbolic representations of language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2952–2962, Florence, Italy. Association for Computational Linguistics.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, Melbourne, Australia. Association for Computational Linguistics.

Ishita Dasgupta, Demi Guo, Andreas Stuhlmüller, Samuel J Gershman, and Noah D Goodman. 2018. Evaluating compositionality in sentence embeddings. *arXiv preprint arXiv:1802.04302*.

Ali Elkahky, Kellie Webster, Daniel Andor, and Emily Pitler. 2018. A challenge set and methods for noun-verb ambiguity. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2562–2572, Brussels, Belgium. Association for Computational Linguistics.

Atticus Geiger, Ignacio Cases, Lauri Karttunen, and Christopher Potts. 2019. Posing fair generalization tasks for natural language inference. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4484–4494, Hong Kong, China. Association for Computational Linguistics.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. Breaking NLI Systems with Sentences that Require Simple Lexical Inferences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 650–655. Association for Computational Linguistics.

Yoav Goldberg. 2019. Assessing bert's syntactic abilities. *arXiv preprint arXiv:1901.05287*.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. 2018. Annotation artifacts in natural language inference data. *arXiv preprint arXiv:1803.02324*.

Rowan Hall Maudslay, Hila Gonen, Ryan Cotterell, and Simone Teufel. 2019. It's all in the name: Mitigating gender bias with name-based counterfactual data substitution. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5266–5274, Hong Kong, China. Association for Computational Linguistics.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031. Association for Computational Linguistics.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Nelson F. Liu, Roy Schwartz, and Noah A. Smith. 2019. Inoculation by fine-tuning: A method for analyzing challenge datasets. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2171–2179, Minneapolis, Minnesota. Association for Computational Linguistics.

Kaiji Lu, Piotr Mardziel, Fangjing Wu, Preetam Amancharla, and Anupam Datta. 2018. Gender bias in neural natural language processing. *arXiv preprint arXiv:1807.11714*.

Karttikeya Mangalam and Vinay Uday Prabhu. 2019. Do deep neural networks learn shallow learnable examples first?

R Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. 2019. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*.

Jan Niehues and Eunah Cho. 2017. Exploiting linguistic resources for neural machine translation using multi-task learning. In *Proceedings of the Second Conference on Machine Translation*, pages 80–89, Copenhagen, Denmark. Association for Computational Linguistics.

Adam Poliak, Aparajita Haldar, Rachel Rudinger, J. Edward Hu, Ellie Pavlick, Aaron Steven White, and Benjamin Van Durme. 2018. Collecting diverse natural language inference problems for sentence representation evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 67–81, Brussels, Belgium. Association for Computational Linguistics.

Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. 2018. Gender bias in coreference resolution. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 8–14, New Orleans, Louisiana. Association for Computational Linguistics.

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2931–2951, Florence, Italy. Association for Computational Linguistics.

Swabha Swayamdipta, Sam Thomson, Chris Dyer, and Noah A Smith. 2017. Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. *arXiv preprint arXiv:1706.09528*.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2016. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, and Yoram Singer. 2019. Identity crisis: Memorization and generalization under extreme overparameterization.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Ryan Cotterell, Vicente Ordonez, and Kai-Wei Chang. 2019. Gender bias in contextualized word embeddings. *arXiv preprint arXiv:1904.03310*.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 2018. Gender bias in coreference resolution: Evaluation and debiasing methods. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 15–20, New Orleans, Louisiana. Association for Computational Linguistics.

Ran Zmigrod, Sebastian J. Mielke, Hanna Wallach, and Ryan Cotterell. 2019. Counterfactual data augmentation for mitigating gender stereotypes in languages with rich morphology. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1651–1661, Florence, Italy. Association for Computational Linguistics.

A Effect of Hyperparameter Settings

We vary several model and task settings, and we find that the models exhibit similar behavior as in the default case. The default settings are summarized in Table 2. We run all experiments over five random seeds, arbitrarily set to $42, 43, 44, 45, 46$. In all plots the error band is the 95% confidence interval with 1,000 bootstrap iterations. We do not average over all the random seeds for experiments with dataset size because of the higher computational cost.

We find that batch size (16, 32, 64) and dropout (0.0, 0.01, 0.1, 0.5) within the MLP do not affect model performance. We also find that embedding and hidden sizes (50, 125, 250, 500) do not significantly impact results, except that performance is worse for 500. This is likely because more training data is needed for these higher capacity models. We don't include figures for these experiments for the sake of conciseness. We present results below in Section A.1 for different vocabulary sizes, and we find that the results don't change significantly, though the model is unable to learn the true property for the smallest vocabulary size.

Hyperparameter	**Value**
Model Settings	
optimizer	Adam
early stopping patience	5
batch size	64
number of LSTM layers	1
hidden dimensionality	250
embedding dimensionality	250
initial learning rate	0.001
dropout	0
Task Settings	
vocabulary size	50K
training size	10K
sequence length	5

Table 2: Hyperparameter settings. We set hyperparamters as default values for our experiments above. Separately, we investigate various values for batch size, dropout, and embedding and vocab sizes and we find that they do not significantly affect the results, with the exception of models with high embedding and hidden sizes. We find the same for vocabulary size (discussed below in A.1).

A.1 Vocabulary Size

We re-run the main experiment – in which we vary the number of adversarial counterexamples – at different vocabulary sizes. The results are shown in Figure 7. We observe similar results when the vocabulary size is 5K as for the default of 50K. The models learn the true properties to some extent as we increase the number of counterexamples, and more counterexamples are needed for the model to generalize well when the true property is harder. However, we note that `first-last duplicate` is harder than the other true properties at this vocabulary size. We aren't sure why this is the case because we see relative hardness similar to the default case in classification experiments with vocabulary size of 5K (the model is trained on identifying whether the true property holds as in Figure 2). We also note that in the same classification experiments with a vocabulary size of 0.5K, the model was not able to learn any of these true properties, which explains why the model didn't learn the true properties at any number of counterexamples.

B Complicating the True Property

In a real-world setting, there often isn't an underlying true property that exactly predicts the label. We consider two variants of the experimental setup that weaken this assumption. Specifically, we introduce label noise, and the case when the true property is a union of multiple other properties.

B.1 Random Label Noise

For some noise level ϵ, we independently flip each label (0 becomes 1, and 1 becomes 0) with probability ϵ. The results are in Figure 8. The trends are fairly resistant to noise. The model continues to switch to learning the true property (though it takes slightly more adversarial counterexamples with more noise), and we continue to observe the relationship between the properties' hardness and prediction error as a function of the number of adversarial counterexamples.

B.2 Multiple True Properties

We relax the assumption that there's a single property that exactly predicts the label, and we instead consider k true properties $t_1, \dots t_k$ that occur one-at-a-time with equal probability in examples for which the true property holds (i.e. examples for which both the true and distractor properties hold and *true-only* adversarial counterexamples). Intuitively, the true property becomes $t_1 \vee t_2 \vee \dots t_n$. Figure 9 shows the results. We find that the model, in most cases, switches from relying on the distractor property to learning to use the collection of true properties. We observe that collections of harder properties take more adversarial counterexamples to achieve low error than collections of easy properties, which is consistent with our previous results. Interestingly, in two of the three cases, we find that a collection of properties might take more adversarial counterexamples to achieve low error than any one of the properties in the collection. However,

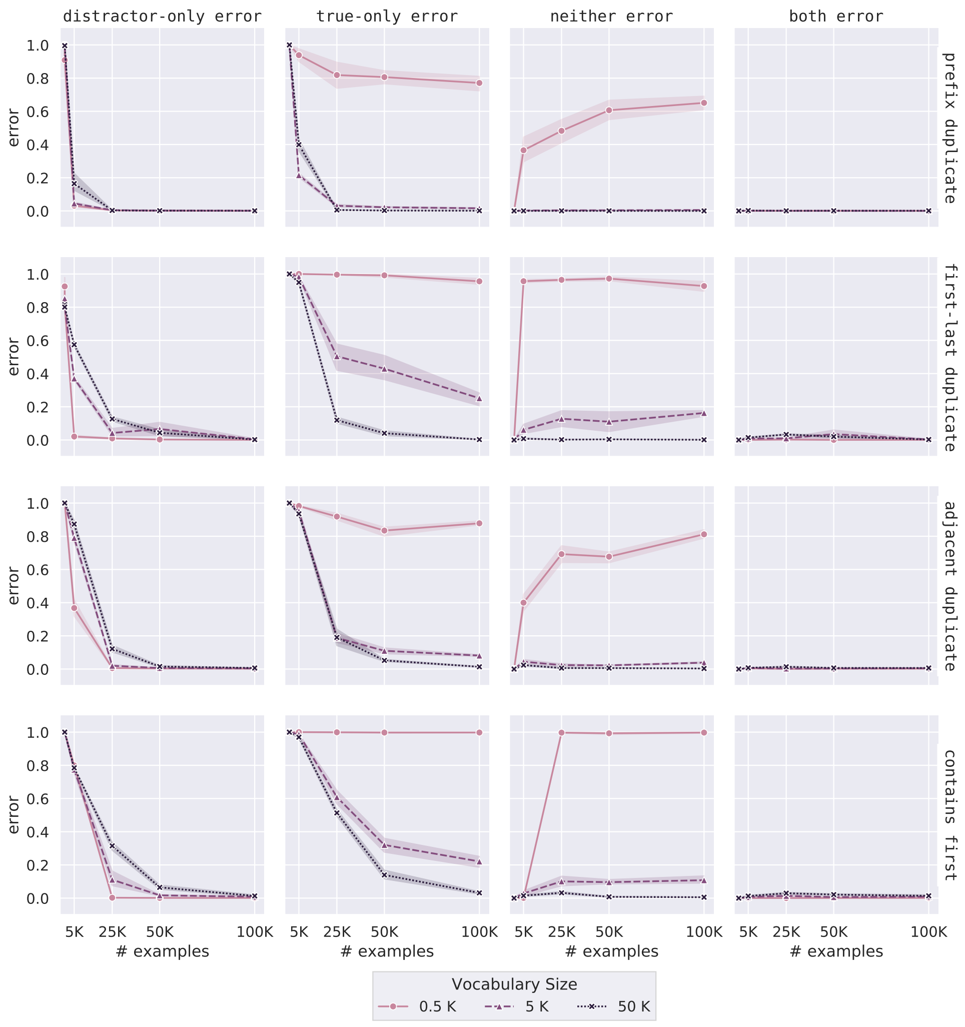


Figure 7: Vocabulary size. The model isn't able to learn the true properties at a vocabulary size of 0.5K, but we observe similar behavior at 5K and 50K.

in the third case (the rightmost plot in Figure 9), the error of `contains first` exceeds that of the combined true property.

C Other Error Metrics

We include additional figures for our main results – hardness (Figure 10), *true-only* and *distractor-only* counterexamples (Figure 11), training size (Figure 12), multiple distractors (Figure 13) – that present the error for the other two regions of the test data (where both or neither of the properties hold). With a single exception, we observe that adding counterexamples doesn't lead the model to generalize worse on data that isn't adversarial, which means that data augmentation helps with overall test error, in addition to test error on adversarial examples. This is expected; if the model switches from using the distractor property to using the true property, its performance shouldn't change on examples where both or neither property holds. However, we note that error on these examples increases (and then decreases) for `first-last duplicate` and `contains first` at a training size of 10M (Figure 12).

D Controls for Training Size

D.1 Adding Non-adversarial Examples

When adding adversarial counterexamples, the total number of training examples also increases. We control for this change by showing here that additional "default" (or non-adversarial) training examples do not help the model on either *distractor-only* or *true-only* error. Figure 14 shows that the addition of these examples does not impact the results. Therefore, it matters that added examples are counterexamples; the model doesn't improve simply because there's more data.

D.2 Fixed Training Size

We've shown above that more training data without counterexamples is not sufficient to induce the model to use the true property in classification;

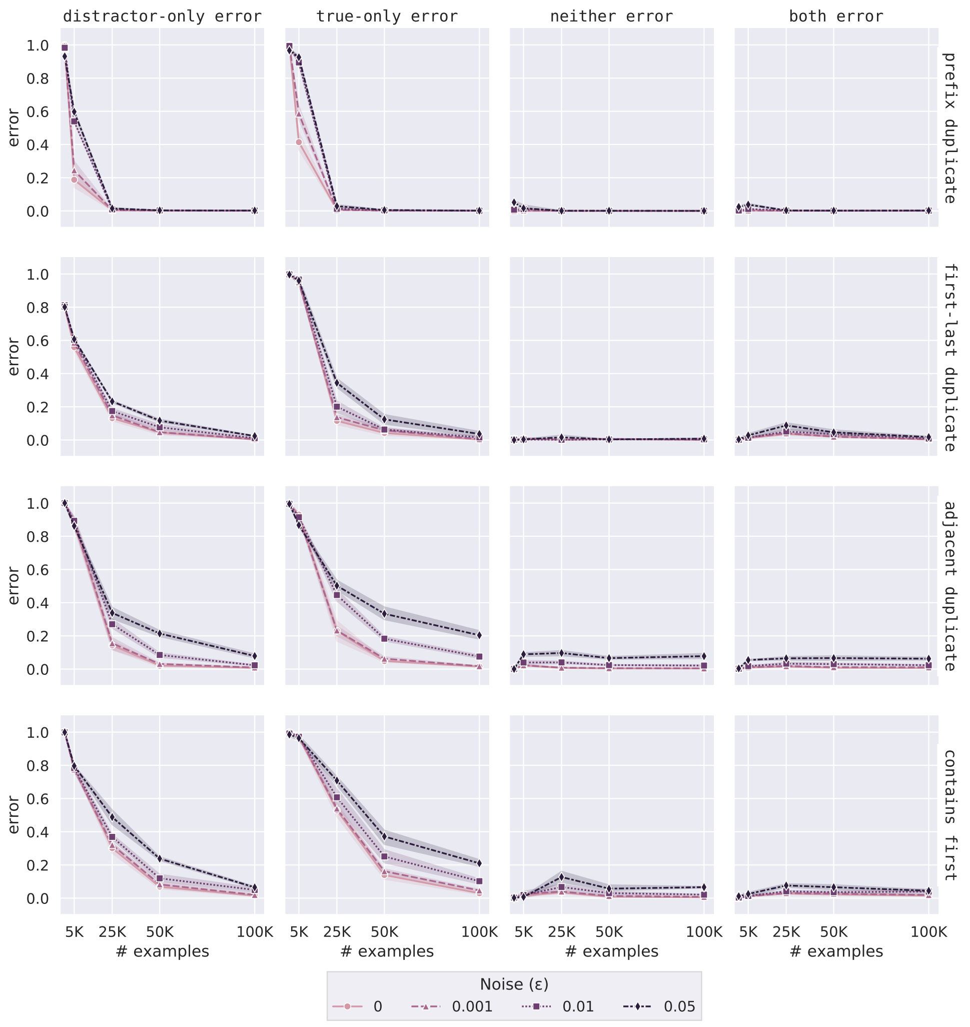


Figure 8: Noise. We observe that adding noise does not affect the results, though more noise seems to make the model less prone to learning the true property.

see Figure 15. However, one might argue that in the presence of some counterexamples, more training data is helpful, whether or not it's adversarial. This would make it hard to disentangle the role of more training data with that of an increased number of counterexamples. Here, we fix training size as we add counterexamples (meaning there are fewer non-adversarial examples) and we observe similar results as in our main experiments above (Figure 3). Naturally, this is not the case for extreme numbers of counterexamples: if we remove all non-adversarial examples, the model is negatively impacted. But these results – taken together with those above – indicate that the benefits of adding counterexamples in general (§3.1) and increasing the number of counterexamples (these results) should not be attributed to the larger training size.

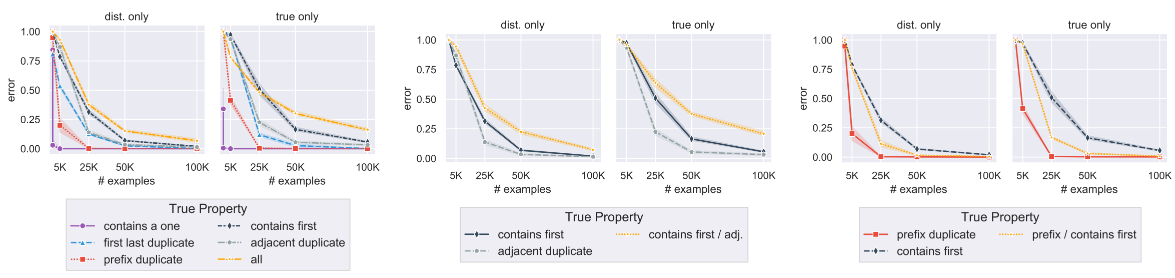
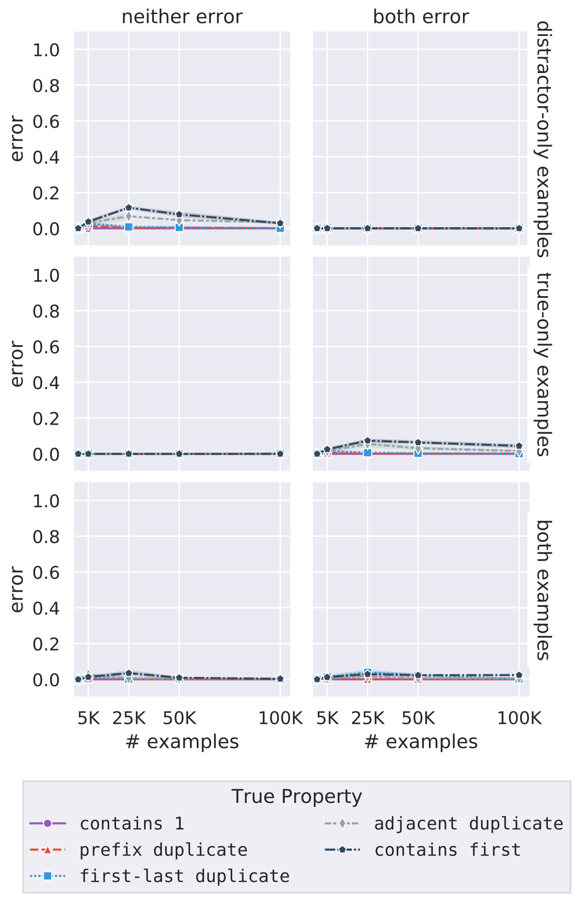


Figure 9: Multiple true properties. We find similar trends as for a single true property. Collections of harder properties require more counterexamples to achieve low generalization error than collections of easier properties. We also find that collections of properties might take more counterexamples to achieve low generalization than any individual property in the collection.



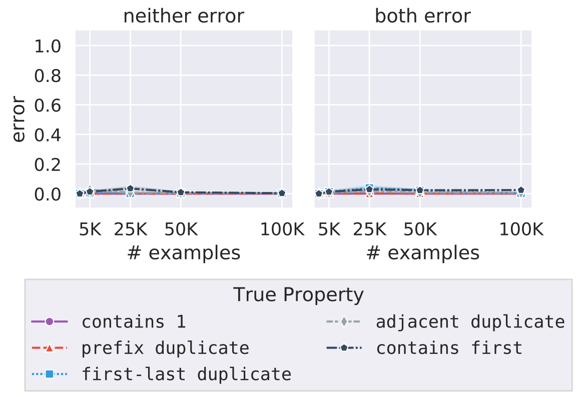


Figure 10: Hardness. Other error cases (*neither* left and *both* right) for Figure 3.

Figure 11: Counterexamples of varying types. Other error cases (*neither* left and *both* right) for Figure 4.

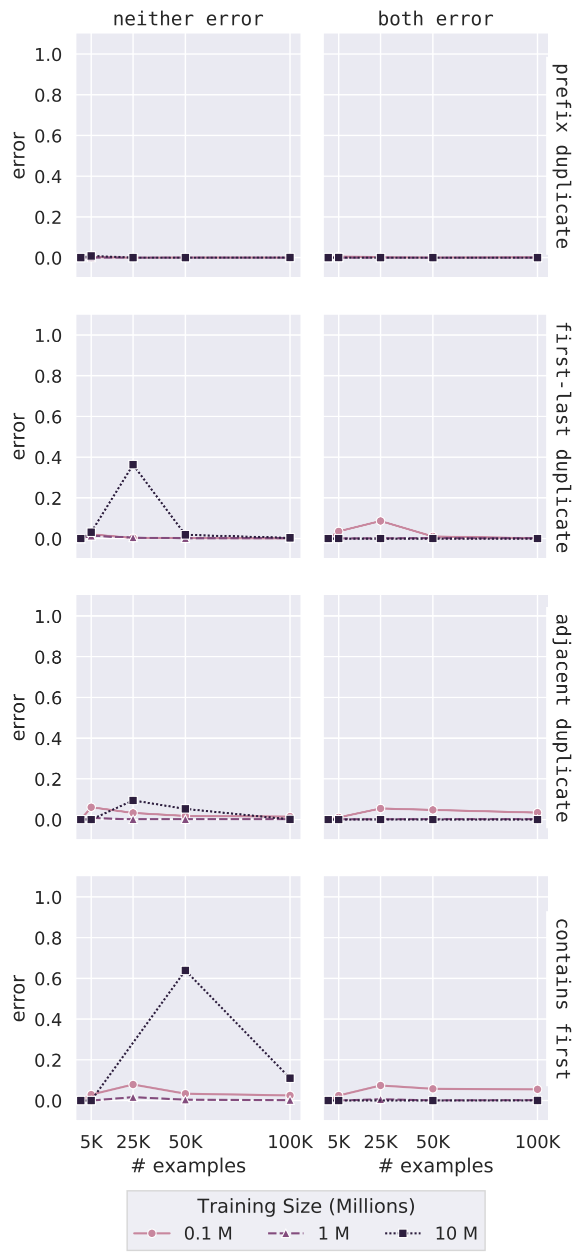


Figure 12: Changing training size. Other error cases (*neither* left and *both* right) for Figure 5.

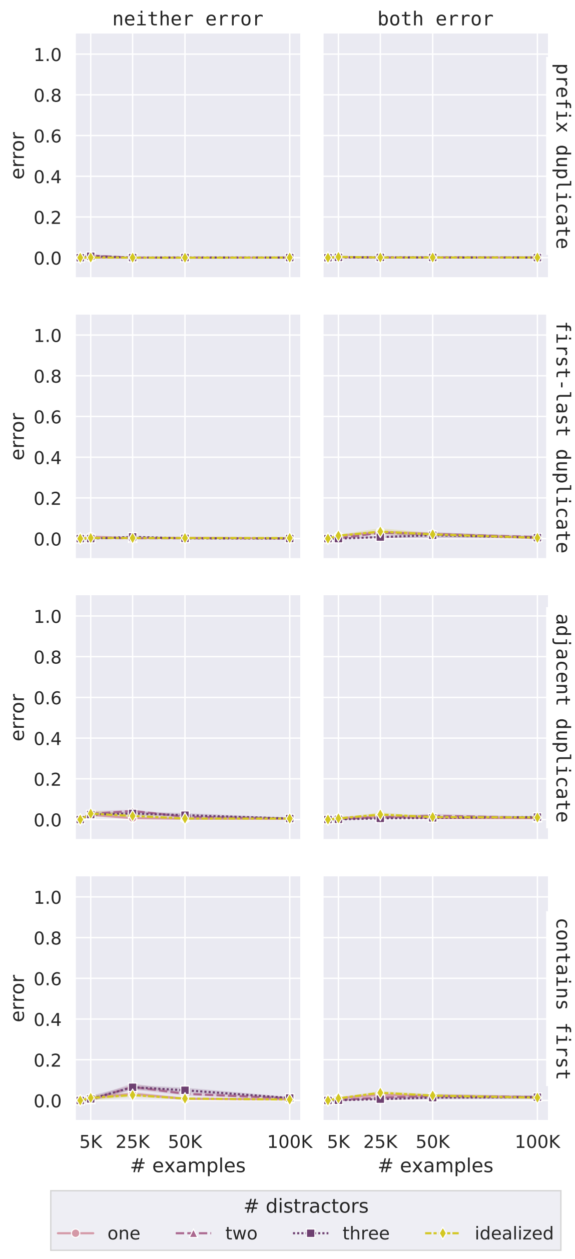


Figure 13: Multiple distractors. Other error cases (*neither* left and *both* right) for Figure 6.

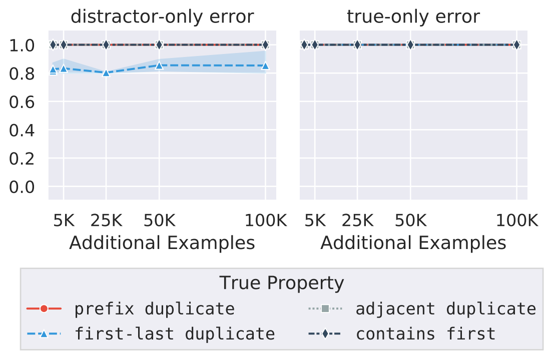


Figure 14: When adding adversarial counterexamples, the total number of training examples also increases. We control for this change by showing here that additional “default” training examples do not help the model on either *distractor-only* or *true-only* error. This is unsurprising given our previously shown results.

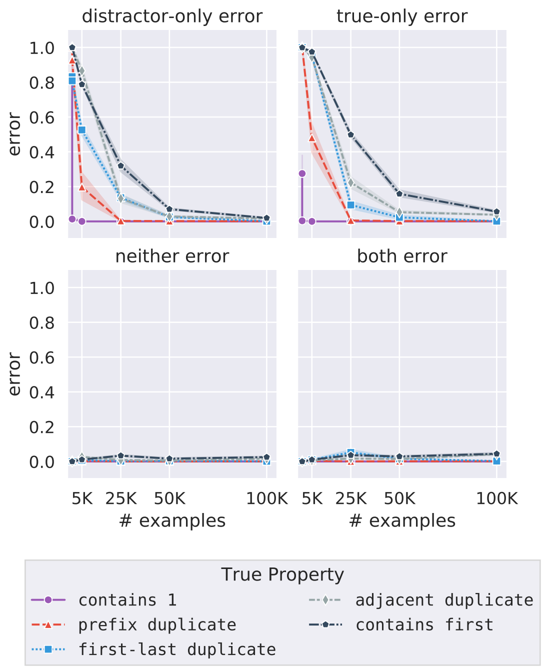


Figure 15: Static training size; counterexamples replace training examples.