Data augmentation and the role of hardness for feature learning in NLP

Rohan Jha
Advisor: Ellie Pavlick, Reader: Michael Littman

May 2020

Most of this work is also in collaboration with Charles Lovering. Some of the writing is from submissions that were collectively written (with Ellie and Charles).

Abstract

Neural models often exploit features that generalize badly in order to achieve good performance. Overcoming this tendency is a central challenge in areas such as representation learning and ML fairness. We approach this problem, in part, from the perspective of feature hardness and data augmentation.

First, we construct a dataset for linguistic acceptability in which multiple, competing features might be used for prediction. We find that in this setting, the downstream model 'prefers' – to some extent – the feature that is more easily extracted from the pre-trained model. OUr preliminary results suggest that the learning of downstream tasks in natural language processing (NLP) is governed, in part, by the 'clarity' with which features are represented by pre-trained models.

Second, we introduce a toy setting to probe the effectiveness of data augmentation, a widely-used strategy to prevent models from learning undesirable heuristics. *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 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 of 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.

Finally, we explore several definitions of feature hardness in the context of the same toy setting, including: (1) the area under the classification learning curve, (2) the sum of weights of a classification model, (3) the minimum description length given by the probing methods of Voita and Titov [2020], and (4) the number of adversarial counterexamples that are needed to induce a model to learn the feature. We show an correspondence between these definitions for the features in our toy setting.

We start with a motivating example, before introducing a general framework and addressing each of these parts in turn.

1 Introduction

1.1 A motivating example

This work was inspired, in part, by McCoy et al. [2019]. The authors trained a state-of-the-art NLP model (BERT) on the task of natural language inference (NLI), in which a model takes two sentences as input (the *premise* and the *hypothesis*) and predicts whether the hypothesis is entailed by the premise (loosely, whether the premise implies the hypothesis). The authors found that the model failed to classify generated pairs for which the label was *contradiction*, and the premise contained all the words in the hypothesis. An example of this would be the pair: "Dog bites man." / "Man bites dog". This led them to conclude that BERT was relying on the heuristic that a premise entails a hypothesis if there's *lexical overlap* between the two sentences.

Crucially, the authors then tried training the model on these contradicting examples and found that the model no longer learned the heuristic to the same extent. This and other similar results have been taken as evidence for data augmentation, or the practice of adding training examples to incentivize models to learn more general features. This work by McCoy et al. [2019] – in addition to this broader practice of data augmentation – prompted the following questions:

1. When is data augmentation an effective strategy? What factors control the extent necessary?

2. More generally, why do models learn some rules for classification, as opposed to others? Is this related to the hardness, in some sense, of these rules?

1.2 Outline

First, in Section 3, we look to answer the second question above, and we investigate whether hardness controls the features learned by state-of-the-art NLP models. We proceed by constructing a natural language dataset for the acceptability task with properties of varying 'hardness' – we'll make this concrete – that predict the label.

Then in Section 4, we take a close look at the first question above, and we explore the effects of feature hardness, the size of training data, and other variables on the effectiveness of data augmentation. Here, we construct a synthetic dataset with multiple properties that are strongly correlated with the label and, like above, are 'hard' to different extents.

Both of the above questions assume a concrete notion of feature hardness. We discuss various definitions in Section 5 and show a rough equivalence between four of them: the area under a learning curve of classifier for the feature; the number of counterexamples needed to induce learning the feature vs. another easier feature; the sum of weights of a classifier for the feature; and the minimum description length metric proposed by Voita and Titov [2020].

1.3 General framing

Here, we introduce terminology and the framework for our experiments in Sections 3 and 4, which will inform discussion throughout. We consider a binary sequence classification task. This will be on natural language data (for the task of acceptability) in Section 3 and sequences of arbitrary tokens in Section 4.

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 relatively easy for the model to extract from the input. This is analogous to lexical overlap between p and h. In our experiments, the correct label is 1 if and only if the true property holds. 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-occurrence 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. We make these terms more precise in Section 1.4.

In Section 3, we look to confirm intuitions that the model will learn the distractor as opposed to the true property in the setting with no counterexamples. And in Section 4, we explore generalization as a function of the number of counterexamples.

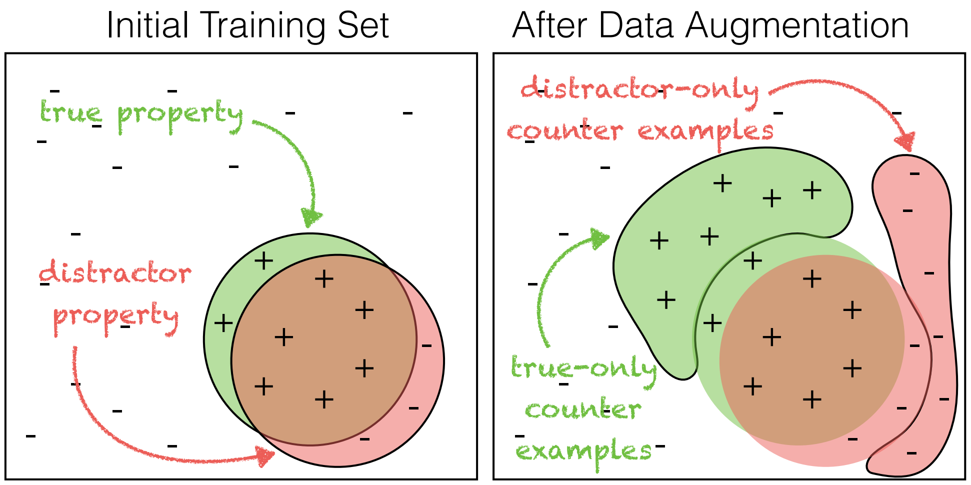

Figure 1: Schematic of experimental setup.

1.3.1 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:	$P(pred = 1 \mid \neg dist, \neg true)$

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 generalization in 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 undersirable heuristics, we will encourage models to learn deeper, more generalizable features in their place.

1.3.2 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 ***true-only* counterexamples** in which the true property occurs without the distractor.

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 4.3.4.

1.3.3 Limitations

These experiments are intended to provide an initial framework and intuition for thinking about the relationships between data augmentation, hardness, feature learning. We simplify the problem significantly in order to enable controlled and interpretable experiments. As a result, the problems studied are many steps removed from the what NLP looks like in practice.

In particular, we make the assumption that the input to the model contains a single "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 general 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 general 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. We also note that identifying the heuristics or distractors a model is learning is often difficult in itself, especially in the presence of many.

1.4 Terminology

We'll use the example of subject-verb agreement here: the task is to predict the grammaticality of a sentence, the *true property* is subject-verb agreement, and the *distractor property* is agreement between the closest-noun and the verb. If the model was trained mostly on examples where the subject was the closest noun, it might use the distractor property to predict the label but not the true property. For example, consider the sentence: "I, like Tom, enjoy running". Here, we might expect the model to predict the sentence is ungrammatical because the closest-noun and the verb don't agree.

True property. In our experiments, we make the assumption that the data (train and test) are generated by a process where the label is determined by whether the *true property* holds, i.e. that the label is positive if and only if the *true property* holds. For example, the subject and verb must agree for the sentence to be grammatical (and we make the further assumption in Section 3 that the sentence is ungrammatical only if they don't agree).

Distractor property. The *distractor property*, on the other hand, is simply correlated with the *true property*. The label is not determined by whether it holds. For example, a sentence might be grammatical or ungrammatical, whether or not the closest-noun agrees with the verb.

Easy and hard features. These correspond to the "effort" needed to extract a feature. *Hardness* is made explicit in Sections 3 and 4 and discussed more generally in Section 5.

General, deep, and generalization. We discuss when models learn features that *generalize* well. We also use *general* and occasionally *deep* to describe such a feature (like the *true property*). This maps onto our framework because we make the assumption in our set-up that the *true property* determines the label under any test distribution and that there exist test distributions for which the *distractor property* is not useful (or even harmful) in predicting the label.

Spurious, shallow, and heuristic. We use *spurious* and sometimes *shallow* to describe a feature (like the *distractor property*) that's highly correlated with the label but does not always generalize well. And we use *heuristic* to describe a learned relationship between a *spurious* feature and the label. We note that a *spurious* feature or *heuristic* might lead to good training performance or even test performance on some (but not all) datasets, and in real-world settings, the distinction between a *heuristic* and a generalizable feature is often unclear.

2 Related work

2.1 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, Poliak et al., 2018b, Gururangan 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 generalization. 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., 2018a] 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].

2.2 Robustness

There's been an explosion of work on *robustness*. This is an overloaded term and can refer to whether models: (1) learn despite noise in the training data, or (2) produce the same labels if the test data have been perturbed (*adversarial robustness*). We'll discuss the latter here, as it's more relevant to our work: we're interested in when models learn features that generalize well not only to the test set but to adversarially chosen examples. *Also note that data augmentation can be seen in this context as a specific technique to improve model robustness.*

This work, however, also considers the role of hardness, and we analyze specific features in the data, as opposed to a well-defined set of perturbations. We're concerned with the model learning harder features that generalize well, while much of literature is interested in features that are also adversarially robust to some well-defined set. Because of the specific sense in which we approach robustness, we don't need to solve the open problem of how to define adversarial robustness in the context of NLP (it's not clear what perturbations mean), and we're able explore *why* models learn specific features from the perspective of hardness.

2.2.1 Within NLP

Again, adversarial robustness concerns whether a model produces the same output when the input has been perturbed such that its underlying semantics are unchanged. In computer vision, these perturbations might be low-level noise added to the pixels. But the set of valid perturbations is much harder to define in NLP, where small changes in surface-form could dramatically change the underlying meaning of an utterance (removing 'not', for example).

In terms of constructing these perturbations, there's been work on replacing words in an utterance with their synonyms [Alzantot et al., 2018, Hsieh et al., 2019, Jia et al., 2019] and generating new sentences via paraphrases [Ribeiro et al., 2018, Iyyer et al., 2018]. In particular, Jia et al. [2019] derive bounds on error within this a well-defined set of perturbations. In the context of *evaluation* of generated perturbations, recent works have discussed the extent to which they induce models to make a wrong prediction [Ribeiro et al., 2018, Iyyer et al., 2018, Hsieh et al., 2019, Jia et al., 2019] or change their output [Alzantot et al., 2018]. Hsieh et al. [2019] also analyze these perturbations' effect on attention weights.

2.2.2 Outside of NLP

Ilyas et al. [2019] make a distinction between useful features (that generalize well) and those that are robustly-useful (that generalize well, even if an example is adversarially perturbed). The authors show the "widespread existence" in image classification data of useful features that are not robustly-useful (non-robust features) by constructing one dataset with only robust featues and another with only non-robust features. Neural models have strong test performance when trained on either but have strong *robust* performance only when trained on the latter. While removing the non-robust features is similar to data augmentation, it's not clear, again, how this technique (or adversarial robustness writ large) transfers to language.

Related to this work by Ilyas et al. [2019], there's been significant recent interest in training models such that they're robust to adversarial examples and in building adversarial datasets that foil such defenses. Highlighting just two recent papers, Madry et al. [2017] describe training that's robust against adversaries with access to a model's gradients, while Athalye et al. [2018] show that many defenses are "obfuscating" their gradients in a way that can be exploited.

2.3 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], relational similarity analyses [Chrupała and Alishahi, 2019], and probing by minimum description length [Voita and Titov, 2020]. 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 of these modeling changes on the results presented in this paper is an exciting future direction.

2.4 Generalization of neural networks

This work also 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].

2.5 Models of hardness

There have been many characterizations of problem hardness in machine learning, often defined with the goal of being using to bound generalization. We highlight some of them here.

VC dimension [Blumer et al., 1989], for example, is the cardinality of the largest set of points that some set of sets can shatter, and it characterizes the difficulty of a learning problem. We're able to compute upper bounds (that are linear in the VC dimension) on the number of samples we need in order to learn. VC dimension, however, applies to a *concept class* (i.e. all rectangles in the plane), and modern deep learning models don't learn with respect to concept classes. Kolmogorov complexity [Li et al.] is the length of the shortest program that produces elements in a concept class. It captures the "number of requirements" for a sequence, or the amount of structure, but it isn't computable in general.

Much more recently, Nelson et al. [2020] cast the hardness of linguistic features in the light of the models of computation that are needed to represent the feature. Arora et al. [2018] also propose a data-dependent metric of hardness with the same goal as VC dimension, to bound the generalization of a learning model. Relatedly, Voita and Titov [2020] consider "the amount of effort" that's needed for a probing model to learn some feature from an intermediate representation. They propose using the minimum description length of the labels, given the representations, to measure this "amount of effort" or hardness. We discuss and implement one of their metrics in Section 5.

3 Which features do models prefer?

This section discusses preliminary results from ongoing work with Ellie Pavlick and Charles Lovering. We've also discussed this work frequently with Tal Linzen.

3.1 Introduction

Here, we're interested in the question of which features are learned by neural NLP models. We consider the transfer learning setting in which a task-specific classifier is trained to use representations from a pre-trained model like BERT [Devlin et al., 2018]. We hypothesize that when there are multiple, competing features that predict a label, the classifier learns those which are more easily extractable. In particular, we design a learning problem for the task of linguistic acceptability, in which two features are both perfectly correlated with the label. We generate sentences from templates, using the BLiMP codebase [Warstadt et al., 2019]. We conduct probing experiments to compare the "hardness" of the features, and we find that while the difficulty of each feature doesn't match our predictions, the models do, to some extent, use the easier feature over the harder.

3.2 Setup

3.2.1 Recap

We assume there exists some feature which determines the correct label, but which is non-trivial to extract given the raw input (true property). And one or more distractor properties which are easier for the model to extract from the input. See Section 1.3 and Figure 1 for more details.

3.2.2 Implementation

We use the linguistic acceptability task (see CoLA [Warstadt et al., 2018]). Models take sentences as input and output {0,1} labels for whether the sentence is grammatical.

We train a logistic regression model that takes representations from the pretrained `bert-base-cased` model [Devlin et al., 2018] as input. See Appendix E for our hyperparameter settings. We don't freeze BERT's parameters; we weren't able to train the models

Example	Example type	Dist.	True
*The *sketch* of the *birds* don't interest Pamela.	*distractor-only*	Yes	No
The *sketches* of the *bird* don't interest Pamela.	*true-only*	No	Yes
*The *sketch* of the *bird* don't interest Pamela.	*neither*	**No**	**No**
The *sketches* of the *birds* don't interest Pamela.	*both*	**Yes**	**Yes**

Table 1: Example sentences for the agreement dataset. We generate sentences the for combinations {true, distractor} × {holds, not holds}. The true and distractor columns are bolded where they co-occur. An asterisk indicates a sentence is ungrammatical.

otherwise, most likely because of the limited size of our training data. We run our experiments using PyTorch and the `jiant` toolkit [Wang et al., 2019].

3.2.3 Data

Features. We consider one pair of properties. The true property here is subject-verb agreement and the distractor property is agreement between the closest-noun and the verb. We hypothesize the distractor is easier because it requires less syntactic information about the sentence, and the model, when trained on no *true-only* or *distractor-only* counterexamples, will learn the distractor property and not the true property. However, as discussed in Section , we find that true property is easier for the model (and is preferred in the absence of counterexamples).

We generate our data using modified templates from the BLiMP codebase [Warstadt et al., 2019], which "[samples] lexical items from a vocabulary of over 3,000 items according to a template..." We modify a single template from BLiMP to generate sentences of the form (the asterisk indicates the sentence is ungrammatical):

"The {**sketch*, *sketches*} of the {*bird*, *birds*} don't interest Pamela."

The subject and closest-noun are both italicized, and the true and distractor properties will co-occur when their plurality matches. See Table 1 for examples of each of the cases. We generate sentences for the combinations {true, distractor} × {holds, not holds}.

Datasets. We generate datasets where the target is whether the sentence is grammatical *or* whether the distractor holds (for probing experiments). We also vary the rate of co-occurence between the true and distractor properties (0.5, 0.6, ... 1.0).

For example, consider a rate of 0.8. In the training and validation sets, 20% of the data comes from the two "counterexample" cases without co-occurence (equal numbers of each), and 80% comes from the remaining two cases (equal numbers). The data are generated in pairs, with one acceptable and one unacceptable sentence, and the two sentences from each pair are guaranteed to be in the same dataset. The test dataset is generated in the same way but with a co-occurence rate of 0.5. The dataset sizes are **2K** for train, **2K** for test, and **1K** for validation.

Limitations. In addition to the limitations discussed in Section 1.3.3, the models in this section will be able to easily learn heuristics for the true and distractor properties because the data are generated from templates. For example, the subject is always first in the SV templates; the model, therefore, doesn't need to learn subject-verb agreement, just agreement between the first noun and the verb. However, while it's likely the model doesn't learn the intended feature, this doesn't affect the higher-level research question of whether hardness controls feature-learning. Also, the templates frequently generate sentences that don't sound natural, like "The cousin of a lot of pedestrians goes to a college campus" (which is labelled as acceptable). Finally, because we only build one pair of features, and because of the unrealistic setting in which the models have *no* incentive to learn the true property, the strength of our conclusions is somewhat limited.

3.3 Results

3.3.1 Probing

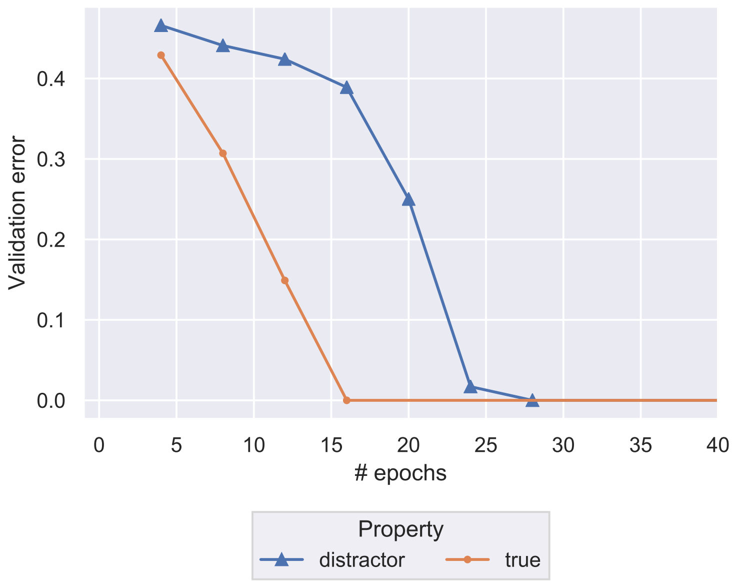

Figure 2: Learning curve for classifying whether the true and distractor properties hold. Flat-lined after the average becomes less than 0.01. We compute an AUC of **1.99** for the distractor property and **0.89** for the true property.

Here, we present results from our probing experiments. We train the models at a co-occurence rate of 0.5 with labels of whether the (1) true property or (2) the distractor property holds. We use the area under the learning curve (AUC) as a proxy for the hardness. In particular, we take the sum of the validation error at each pass (every four epochs). After the model reaches an error less than 0.01, we round to 0 and flat-line the curve. We present the learning curve in Figure 2.

We compute an AUC of **1.99** for the distractor property and **0.89** for the true property. Therefore, the distractor (agreement with the closest noun) is harder by our metric than

both	*neither*	*dist.-only*	*true-only*
0.4%	0.4%	0%	38.6%

Table 2: Test error on a target of whether the true property holds when the model is trained on data with perfect co-occurence. We find that the model reliably uses the true property for *distractor-only* example. However, for *true-only* examples, the model is less consistent.

the true property. As discussed above, this could be a result of the fact that we only use a single template; the model doesn't need to learn SV agreement, just agreement between the first noun and the verb. The assumption that syntax is 'harder' for BERT than finding the closest noun might also be wrong. There's been much evidence that BERT or similar models represent syntactic properties like SV agreement [Warstadt et al., 2019, Goldberg, 2019]. Goldberg [2019] also notes that BERT "does not have an explicit notion of word order beyond marking each word with its absolute-position embedding," which suggests that BERT might not be very susceptible to the closest-noun heuristic.

3.3.2 Which features are used?

Again, the research question is not jeapordized by the mismatch between the properties' expected and actual hardness. We can still ask if hardness predicts the features that a model learns. Here, we train and test the model on the objective of acceptability (the same as whether the true property holds) with perfect co-occurence between the true and distractor properties. This means that the subject and closest-noun always have the same plurality, and that the model can achieve perfect training performance by learning either the true or the distractor property. In other words, the model has no incentive in the training data to learn one feature vs. the other (no counterexamples).

We present the results in Table 2. We find that the model reliably uses the true property for *distractor-only* examples, meaning that it predicts the sentence is ungrammatical when the closest noun agrees (but not the subject). However, for *true-only* examples, when the subject agrees (but not the closest noun), the model is less consistent. While the model, for the most part, predicts the sentence is grammatical, these results show that the models' outputs are different in the *both* case from the *true-only* case and therefore, that the presence of the distractor property is meaningul for the model.

However, taken as a whole, the model *prefers* the true property, to some extent, which is in line with our hypothesis that the model will learn the easier feature. In one case without co-occurence, its predictions are always in line with a model that uses the true property, and the predictions are largely in line in the other case. To make this concrete, we compute the mutual information between the model's predictions on the test dataset and a 0-1 variable that indicates if (1) the true property holds or (2) the distractor property holds. This measures the extent to which the model's predictions are independent of whether each property holds. We find that I(prediction;true) = **0.42** and I(prediction;distractor) = **0.15**, which indicates a stronger correlation between the true property and the label than between the distractor property and the label.

3.3.3 Data augmentation

Finally, we train the models with some incentive to learn each property. They're trained with a co-occurence rate of 0.9 and a target of either the true property (again, the same as acceptability here), or the distractor property. We find, in each case, that the models use the target property from the training data at test time. In particular, the target of the test dataset matches that of the training, and for both properties, we find that the error in each of the four cases (*both*. *neither*, etc.) is less than 2.5%. This means that the non- co-occurring examples (these will be *true-only* or *distractor-only* counterexamples) were enough to incentivize for model to learn either feature.

3.4 Conclusion

We find some evidence for our hypothesis that models learn 'easier' features in the absence of incentive to learn a 'harder' feature. However, there are significant limitations with these preliminary results. Next steps are to add more pairs of properties, make templates that are more natural-sounding, and come up with more templates for each pair of properties to limit the artifacts in the data that models can use their place. Finally, we would also like to further explore the more realistic case, in which there's limited incentive (vs. none) for the model to learn the true property.

4 When does data augmentation help?

This section discusses work with Ellie Pavlick and Charles Lovering that will be submitted to EMNLP 2020.

4.1 Introduction

A wave of recent work has exposed model 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 generalizable 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 generalization. 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 include:

- How many counterexamples must be seen in training to prevent a 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 a model's tendency to adopt heuristics?
- When many spurious correlations exist, do models prefer multiple imperfect heuristics or fewer, more reliable features?
- Is data augmentation effective when just one type of counterexample can be added (*true-only* or *distractor-only*)?

4.2 Experimental setup

4.2.1 Recap

We assume there exists some feature which determines the correct label, but which is non-trivial to extract given the raw input (true property). And one or more distractor properties which are easier for the model to extract from the input. See Section 1.3 and Figure 1 for more details.

4.2.2 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 $0 \dots |V|$. 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.[1] Ap-

[1] https://github.com/rohjha/rules

pendix A discusses how our experimental results extend across changes in model and task parameters. All models are trained until convergence, using early-stopping.[2]

Property nickname	Description	AUC	Example
`contains 1`	Has 1	0.50	`2 4 11 1 4`
`prefix duplicate`	Begins with duplicate	0.52	`2 2 11 12 4`
`first-last duplicate`	First equals last	0.55	`2 4 11 12 2`
`adjacent duplicate`	Has adjacent duplicate	1.43	`11 12 2 2 4`
`contains first`	First number elsewhere	1.54	`2 11 2 12 4`

Table 3: 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 4.2.2.

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 3. 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.

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 3 shows the validation loss curve (averaged over three runs), flat-lined 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 (AUC), computed by taking the sum of the errors across all epochs. Table 3 contains the result for each property. Note that the distractor property (the sequence contains `2`) is exactly as hard as `contains 1`. We consider more sophisticated metrics of hardness in Section 5.

4.2.3 Error metrics

Recall the following definitions of our error metrics. See Section 1.3.1 for detailed interpretations.

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

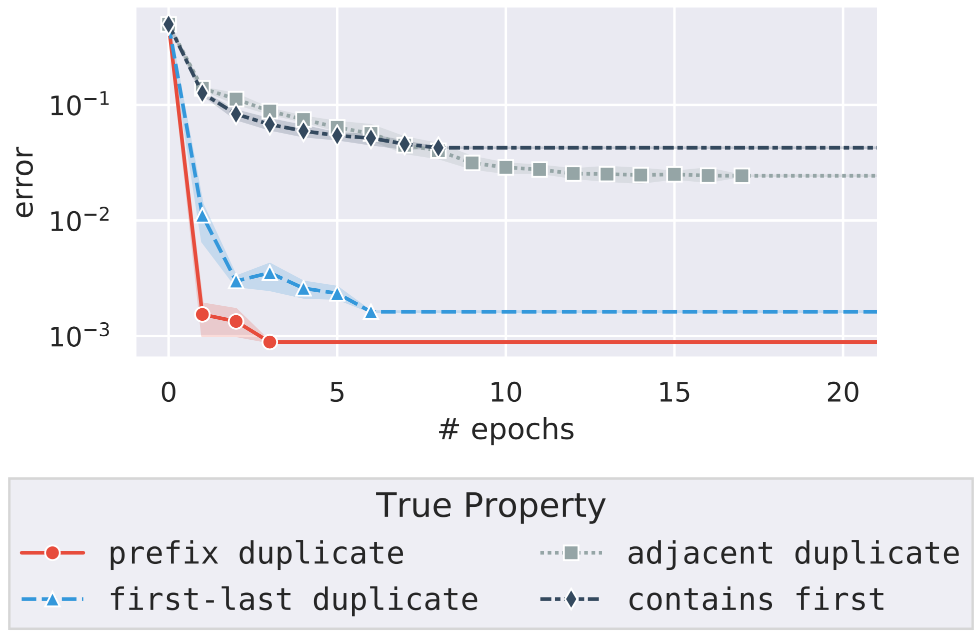

Figure 3: Learning curve for classifying whether the properties hold. Averaged over three re-runs and then flat-lined after the average reaches its minimum. Train size is 200K. The error for `contains 1` and the distractor property (not shown) reaches 0 after the first epoch.

Definition.

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:	$P(pred = 1 \mid \neg dist, \neg true)$

We define and compute *both* error and *neither* error for completeness. 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 4.3, we only show plots of *true-only* and *distractor-only* error, and leave plots of the others to Appendix C.

4.2.4 Data augmentation

Recall the following definitions: ***distractor-only* counterexamples** in which the distractor property occurs without the true property (and the label is 0 in this toy setting), and ***true-only* counterexamples** in which the true property occurs without the distractor (and the label is 1 in this setting). See Section 1.3.2 for detailed interpretation.

4.2.5 Limitations

In addition to the limitations discussed in Section 1.3.3, we consider a very small problem here (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.

4.3 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 (§4.3.1), the type of counterexamples added (i.e. *true-only* vs. *distractor-only*)(§4.3.2), and the size of the training set (§4.3.3)? 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 general feature (§4.3.4)? We report all results in terms of *true-only* error and *distractor-only* error; results for other error categories are given in Appendix C.

4.3.1 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 4.2.2). 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[3] 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 assume that the counterexamples added are evenly split between *true-only* and *distractor-only* types.

Figure 4 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.

We can think about this from the perspective of implicit regularization, or the preference of neural models to learn simpler representations, even in the absence of explicit methods like

[3]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.

dropout (see Section 2.4 for a discussion of the recent work in this area). At first, our models learn the simpler distractor property, instead of the more complex true property. However, as counterexamples are added to the training data, it likely becomes more and more complex for the model to learn only the distractor property, because it would need to memorize these counterexamples. The relative hardness of the true properties, then, might control when the models prefer (from the point of view of complexity) to adopt the true property vs. memorize additional counterexamples. More work is needed to make these conclusions. One might measure the complexity of the models at various points. It's also important to note that the models don't *switch* from preferring the true property to the distractor. At many points on the curve, they seem to apply the properties to different extents, which might also be worth exploring further.

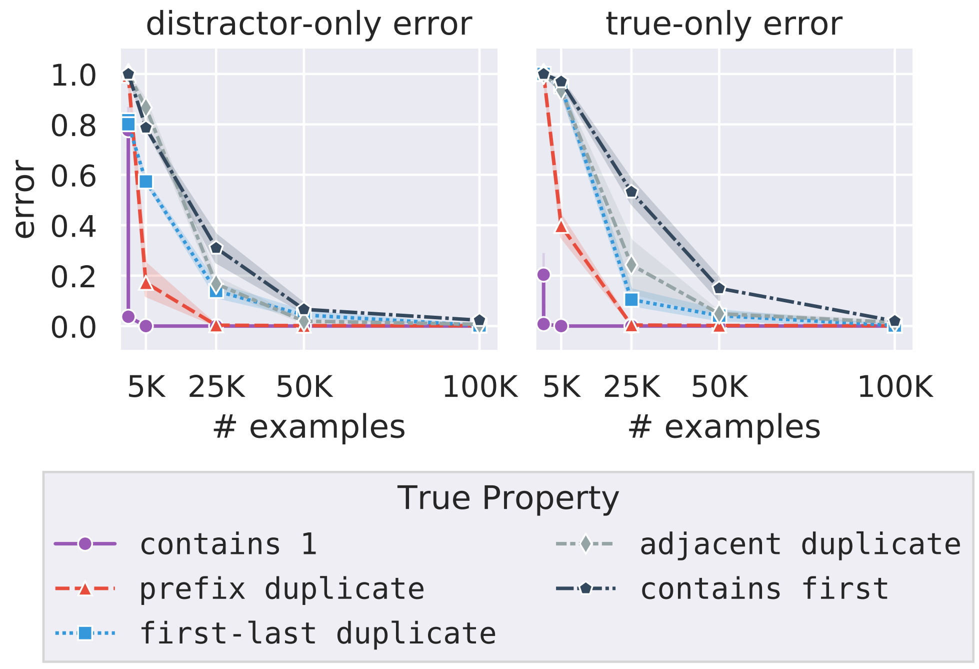

Figure 4: The harder the true property, the more counterexamples the model requires to reduce a given error rate. The legend shows models in order from hardest to least hard, where "hardness" is determined by the AUC of the loss curve for a classifier trained to detect the property (§4.2.2). We run all experiments over five random seeds. In all plots, the error band is the 95% confidence interval with 1,000 bootstrap iterations.

4.3.2 Effect of counterexample type

Counterexamples from one of two types. We get a better understanding of the model's behavior when looking 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 5). Here, we add *either true-only* or *distractor-only* examples (we consider mixtures below). 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.

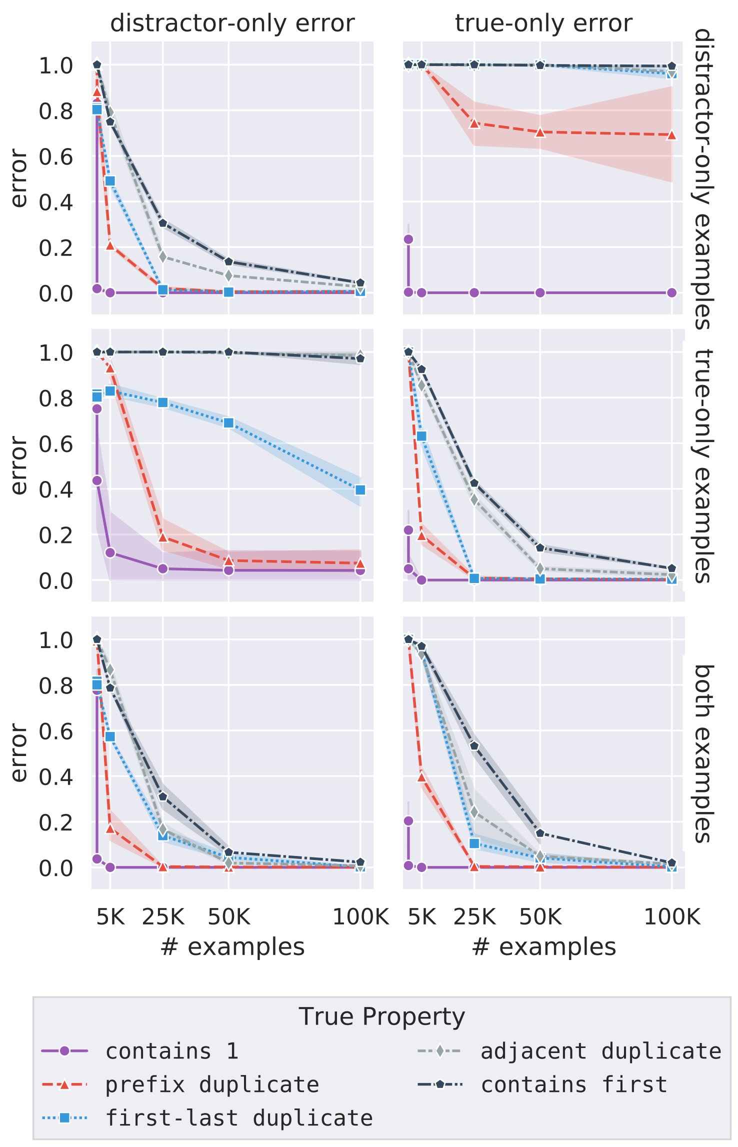

Figure 5: Adding *distractor-only* counterexamples improves *distractor-only* error but does not affect *true-only* error. In contrast, adding *true-only* counterexamples may lead to improvements of both error types (see text for discussion).

In contrast, we see some evidence that adding *true-only* counterexamples has impact on *distractor-only* error as well as on *true-only* error in this setting. 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.

We consider, in particular, the models that learn to output the postive label if and only if the true and distractor properties both hold (e.g. the model trained on 100K *distractor-only* counterexamples for the `contains first` true property). These models seem to have learned a conjunction of the true and distractor properties as their classification rule, where just the true property have perfomed as well on the training data. This isn't very surprising, because the model hadn't seen any *true-only* counterexamples, but it seems to contradict the principle that models prefer simpler representations. A likely interpretation is that the model learned the distractor property first and then had no incentive to unlearn the property, and it's exciting future work to explore this temporal aspect of the models' training.

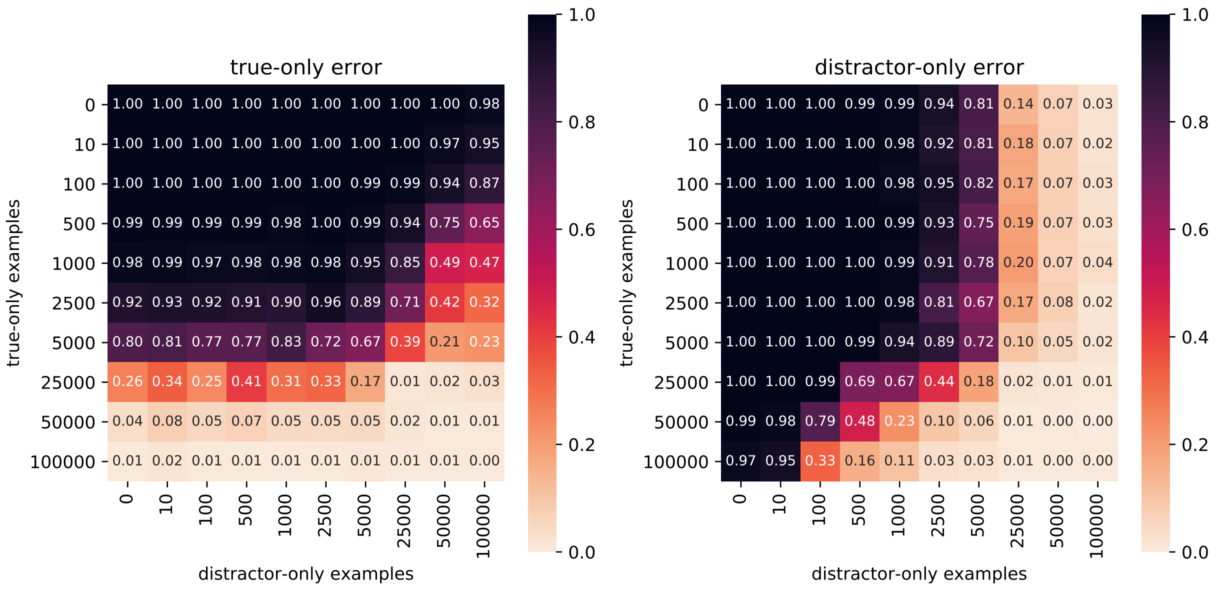

Figure 6: For `adjacent duplicate`. Adding *distractor-only* counterexamples is not effective for reducing *true-only* error when there are few *true-only* examples in the dataset, and vice versa. Adding *distractor-only* examples is increasingly helpful when there are more *true-only* examples in the data, and vice versa.

Mixture of counterexamples Here, we weaken the assumption that the counterexamplees come from one of the two types. A training set might have examples of each type, even if we aren't able to augment with *true-only* or *distractor-only* counterexamples. We track the *true-only* and *distractor-only* errors as the number of each type of counterexample is varied. Because of the computational cost, we only consider one feature, `adjacent duplicate`, and we don't run any random restarts.

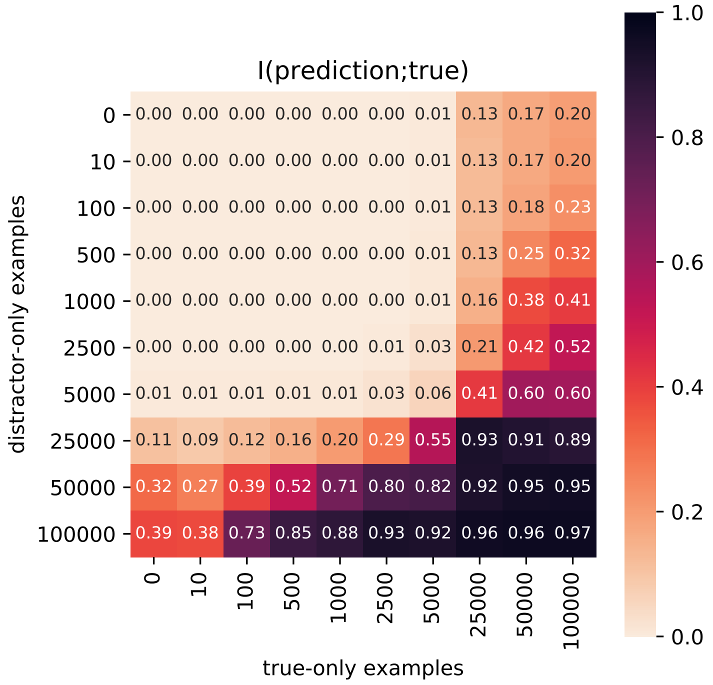

Figure 7: For `adjacent duplicate`. Significant numbers of both types of counterexamples are needed for the model to use the true property in general.

We find that adding *distractor-only* counterexamples is not very effective for reducing *true-only* error when there are few *true-only* examples in the dataset, and vice versa. Adding *distractor-only* examples is increasingly helpful when there are more *true-only* examples in the data, and vice versa. The regions in the top-right and bottom-left corners also support the conclusion above that the model might unlearn the distractor in some cases but not others and that adding one type of counterexample is not sufficient for reducing both types of error.

In Figure 7, we also compute the mutual information between the 0-1 feature of whether feature holds in the test data (which is simply the label) and the model's prediction on the test dataset. This can be used as a proxy for the extent to which the model uses the true property when making its predictions. We find that significant numbers of both types of examples are needed for the model to use the true feature in general.

4.3.3 Effect of training data size

In Section 4.3.1, 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, 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?

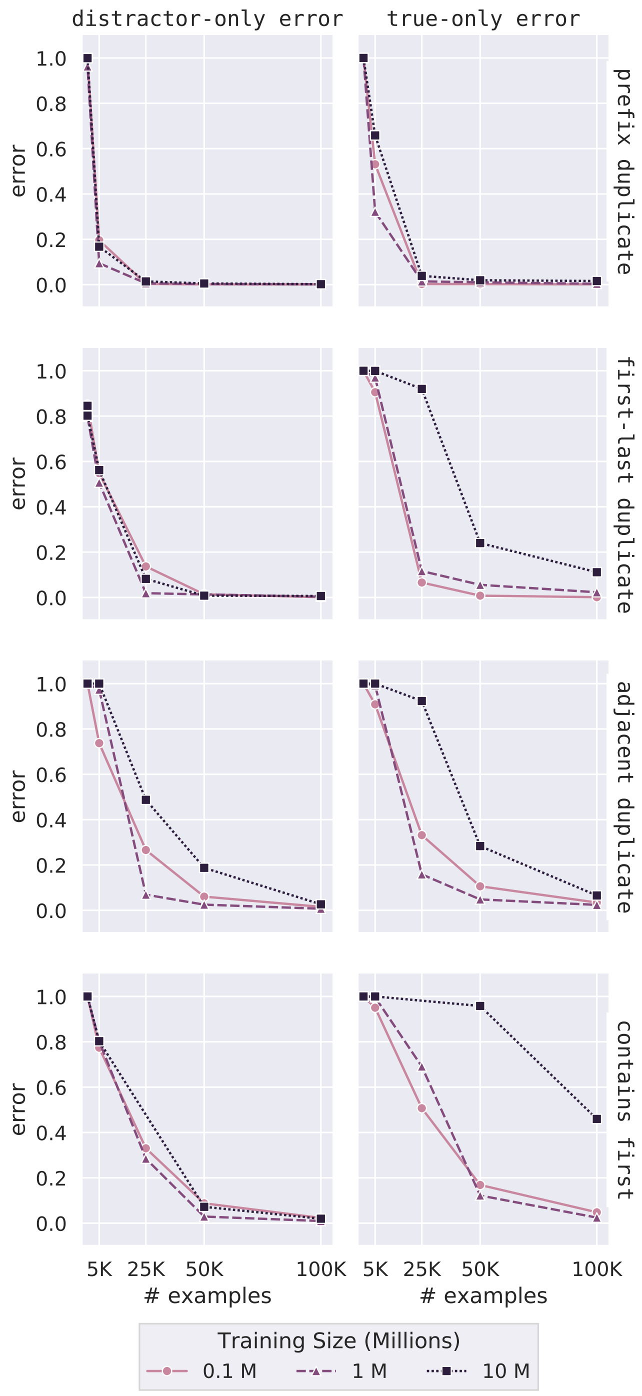

Figure 8: Error rate vs. number of counterexamples added for models trained with different initial training set sizes (100K to 10M). Property `contains 1` not shown since it is always learned instantly. In general, increasing the training size while holding the number of counterexamples fixed does not 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.

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 8, 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`).

4.3.4 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 $1/2$[4]. We assume that *distractor-only* counterexamples can be generated in the same way as before, 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[5] 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 $P(d_j|t)$.

[4]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.

[5]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.

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 9 for $k = \{1, 2, 3\}$.[6] 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. Note that as the number of distractors increases, the likelihood of generating a "pure" *true-only* example is diminished, which might drive these results. 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. It's 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 5 when only *distractor-only* errors were added), but rather a failure to consistently associate the true property with the positive label.

4.4 Conclusion

We present an experimental framework for exploring the effects of adversarial data augmentation on feature learning. Our results suggest:

1. More data, for the most part, does not diminish the efficacy of counterexamples (Section 4.3.3.)

2. Data augmentation becomes less effective as the underlying "true" features become more difficult to extract (Section 4.3.1) and as the number of spurious correlations in the data increase (Section 4.3.4).

3. Adding counterexamples in order to encourage a model to "unlearn" bad heuristics is likely to have the immediately desired effect (the model will perform better on

[6]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.

examples similar to the counterexamples), but the model won't necessarily "unlearn" the heuristic in the presence of multiple distractors (Section 4.3.2) or with limited ability to remove the true property from training examples (Section 4.3.4).

These indicate limitations with the approach of data augmentation, despite its usefulness in some settings.

5 What does 'feature hardness' mean?

This work is in collaboration with Charles Lovering and Ellie Pavlick.

5.1 Introduction

In Section 4, we showed that hardness in terms of a classification curve (we defined it explicitly as the AUC) was predictive of the number of counter-examples needed to induce learning. We're interested here in other methods of characterizing the hardness of feature-learning. We'll introduce MDL probing, which was introduced by Voita and Titov [2020], and then compare the results of MDL and other metrics in the context of the toy setting from Section 4.2.4.

5.2 MDL

Here we briefly present the work of Voita and Titov [2020]. The authors propose using the minimum description length (MDL) of the labels, given the representations, to evaluate the extent to which some feature x is captured by an intermediate representation y.

They view "learning as compression" and note that a model's cross-entropy loss gives "an optimal bound on the codelength if the data are independent and come from... $p(y|x)$".

$$-\sum_{i=1}^{n} \log_2 p(y_i|x_i)$$

Now, evaluating the MDL is the same as computing the loss. However, in addition to the cost of transmitting the labels, we also need to transmit the model.

The authors describe two procedures: the variational and online codes. We use the second because it's easier to implement. The transmission takes place in blocks. Alice and Bob train a model after each block to be used to transmit the next. Because of the equivalence between learning and compression, computing the MDL here is the same as training the model on a block and then evaluating the loss on the next.

The data MDL (MDL_d) is the cost of transmitting the entire dataset, given the final model, and the model MDL (MDL_m) is the difference between the data MDL and the total MDL. The authors write that the first intuitively gives the "quality of the probe" while the second gives the "amount of effort" to learn this probe. We use the block lengths and equations described in the paper, and instead of intermediate representations, we let x be the data itself.

5.3 Results

Property nickname	AUC	LSTM L_1	MDL_m	MDL_d	Example
`contains 1`	0.496	18,952	0.28K	0.045	`2 4 11 1 4`
`prefix duplicate`	0.517	117,396	58K	139	`2 2 11 12 4`
`first-last duplicate`	0.546	115,305	40K	126	`2 4 11 12 2`
`adjacent duplicate`	1.429	139,799	116K	450	`11 12 2 2 4`
`contains first`	1.542	138,125	126K	910	`2 11 2 12 4`

Table 4: Properties used to instantiate the true property in our experimental setup. Various hardness metrics that are predictive of one another. MDL_m is the model MDL, and MDL_d is the data MDL.

We find that the following definitions of property hardness are predictive of one another in our toy setting. However, while each gives a similar ranking of the features, there are subtle differences. The AUC, the LSTM norm, and the model MDL (MDL_m) separate the features into three levels of difficulty (the first, the next two, and the last two), while the data MDL (MDL_d) and the number of required counterexamples (Figure 4) indicate a separation betwen the final two properties. The number of required counterexamples, in particular, doesn't separate the features into clear tiers. Finally, both MDL metrics and the number of counterexamples also indicate an order of magnitude difference between the hardness of the first property (`contains 1`) and that of that others, which isn't made clear by the other metrics.

The metrics, except the counterexamples, are computed with the classification set-up from Section 4.2.2, in which a model is tasked simply with classifying whether each property holds. We ran multiple, separate random restarts for each metric (the MDL metrics were necessarily computed together).

Area under curve (AUC). *How long does it take a classifier to learn a property?* We measure this by the area under the learning curve from the classification experiments (the second column of Table 3). This was the metric of hardness in Section 4.

L_1 norm of LSTM. *How complex is the classifier that learns the property?* Models with low total weights (L_1 or L_2 norms) have been viewed as simpler (see literature on regularization including Krogh and Hertz [1992]). Here, we compute the L_1 norm (sum of weights) of the LSTM component of the trained model for the classification experiments. However, the effectiveness of these weights in predicting the other metrics of hardness could be a direct result of the fact that they're highly correlated with the number of epochs until the model converges, which is closely related to the AUC the model MDL. We present an experiment that shows this correlation in Appendix F. We also note that the norms of the embedding layer or the decoder are not as predictive of the other measures of hardness.

MDL (model and data). *What's the cost of transmitting the labels, given the data?* We compute the model and data MDL using the online code from Voita and Titov [2020] that's described in Section 5.2. We find in our experiments that the model MDL is very predictive of the AUC (and the L_1 norm of the LSTM). Both measure the model's performance on limited data, and the authors themselves note a connection between AUC and the model MDL from the online code. The data MDL, which doesn't correspond as well to the same metrics, has a slightly different interpretation. It measures the extent to which the learned model is able to compress the data, or the "quality" of the model trained on the entire dataset.

Counterexamples to induce training. *How many counterexamples do we need to add in order to induce the model to learn the property, as opposed to some other fixed distracting property?* This is simulated by the experiments in Section 4.3.1 (Figure 4).

6 Conclusion

We make the following contributions:

1. We construct a natural language setting which provides some evidence that neural models 'prefer' easier features without incentive to learn harder features (Section 3).
2. We introduce a toy framework to explore the conditions when data augmentation is effective (Section 4).
3. We show that in this toy setting, the number of requisite counterexamples is dependent on the hardness of the desired feature and, for the most part, is independent of the total number of examples. We also show that data augmentation in various settings doesn't cause the model to unlearn the undesirable feature (Section 4).
4. We show an equivalence in the toy setting between several definitions of feature hardness (Section 5).

Taken as a whole (in a different order), we discuss feature hardness, show this affects the learning of state-of-the-art NLP models, and then illustrate limitations of data augmentation, which is one method to prevent models from learning undesirable features.

6.1 Future work

We're excited by many directions for follow-up work. For the results in Section 3, we're working on using MDL to track hardness, as opposed to AUC, and we're developing more natural-sounding templates. We'd also like to construct more templates for each phenomenon, to incentivize the model to learn the desired features, as opposed to heuristics. And we're working on extending these experiments to other phenomena and to settings with varying incentives for the model.

With respect to the Section 4 experiments, we're interested in exploring the training in terms of the features the models learn over time. We're also formulating the experiments in the context of Bayesian models to simplify the multiple distractors setting and maybe to formalize some of our findings about the effectiveness of counterexamples. A theoretical (or empirical) study of the efficacy of counterexamples on simpler models (e.g. linear models) would also be useful in gaining more intuition for the results in this section.

For the Section 5 experiments, we're interested in other models of feature hardness that make use of the data itself; these could be along the lines of that proposed by Arora et al. [2019] to bound generalization. We'd also like to implement the model compression discussed by Voita and Titov [2020] and introduced by Louizos et al. [2017] for a more sophisticated representation of network complexity. We might also consider the size of the smallest model needed to learn some feature to be a proxy for the feature's hardness.

More broadly, the setup here could be used to compare existing models or to build better models. We could track how models respond to few counterexamples or counterexamples of only one type.

Acknowledgments

I'm grateful to Ellie Pavlick for being such a wonderful advisor; she's steered me in productive directions, while giving me the flexibility to explore the problems I find interesting. Working with Charlie has been a delight, and his insights have helped shape many of the experiments and much of the thinking in this work. I'm also thankful for productive conversations with Michael Littman and Tal Linzen, and for the opportunity to take courses with Paul Valiant, each of which have been impactful for my approach to problems in CS. I'm grateful, too, to the LUNAR lab for being such a great environment in which to grow and learn about NLP. Finally, thank you to my friends – at Brown and elsewhere – my family, and my parents, in particular, for being supportive and encouraging throughout this process and my whole time at Brown.

References

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

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

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

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

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. 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, 2017. URL http://dl.acm.org/citation.cfm?id=3305381.3305406.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

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

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

Samuel R. Bowman, Christopher Potts, and Christopher D. Manning. 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, July 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4002. URL https://www.aclweb.org/anthology/W15-4002.

Grzegorz Chrupała and Afra Alishahi. 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, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1283. URL https://www.aclweb.org/anthology/P19-1283.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 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, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1198. URL https://www.aclweb.org/anthology/P18-1198.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Ali Elkahky, Kellie Webster, Daniel Andor, and Emily Pitler. 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, October-November

2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1277. URL https://www.aclweb.org/anthology/D18-1277.

Atticus Geiger, Ignacio Cases, Lauri Karttunen, and Christopher Potts. 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, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1456. URL https://www.aclweb.org/anthology/D19-1456.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 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, 2018. URL http://aclweb.org/anthology/P18-2103.

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

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

Rowan Hall Maudslay, Hila Gonen, Ryan Cotterell, and Simone Teufel. 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, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1530. URL https://www.aclweb.org/anthology/D19-1530.

John Hewitt and Christopher D. Manning. 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, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL https://www.aclweb.org/anthology/N19-1419.

Yu-Lun Hsieh, Minhao Cheng, Da-Cheng Juan, Wei Wei, Wen-Lian Hsu, and Cho-Jui Hsieh. On the robustness of self-attentive models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1520–1529, 2019.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. *arXiv preprint arXiv:1804.06059*, 2018.

Robin Jia and Percy Liang. 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, 2017.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. *arXiv preprint arXiv:1909.00986*, 2019.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016. doi: 10.1162/tacl_a_00115. URL https://www.aclweb.org/anthology/Q16-1037.

Nelson F. Liu, Roy Schwartz, and Noah A. Smith. 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, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1225. URL https://www.aclweb.org/anthology/N19-1225.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

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

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

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

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

Max Nelson, Hossep Dolatian, Jonathan Rawski, and Brandon Prickett. Probing rnn encoder-decoder generalization of subregular functions using reduplication. *Proceedings of the Society for Computation in Linguistics (SCiL)*, pages 31–42, 2020.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BygfghAcYX.

Jan Niehues and Eunah Cho. 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, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4708. URL https://www.aclweb.org/anthology/W17-4708.

Adam Poliak, Aparajita Haldar, Rachel Rudinger, J. Edward Hu, Ellie Pavlick, Aaron Steven White, and Benjamin Van Durme. 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, October-November 2018a. Association for Computational Linguistics. doi: 10.18653/v1/D18-1007. URL https://www.aclweb.org/anthology/D18-1007.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. Hypothesis only baselines in natural language inference. *arXiv preprint arXiv:1805.01042*, 2018b.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, 2018.

Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. 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, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2002. URL https://www.aclweb.org/anthology/N18-2002.

Sofia Serrano and Noah A. Smith. Is attention interpretable? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2931–2951, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1282. URL https://www.aclweb.org/anthology/P19-1282.

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

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. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJzSgnRcKX.

Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. *arXiv preprint arXiv:2003.12298*, 2020.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 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, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://www.aclweb.org/anthology/P19-1580.

Alex Wang, Ian F. Tenney, Yada Pruksachatkun, Phil Yeres, Jason Phang, Haokun Liu, Phu Mon Htut, , Katherin Yu, Jan Hula, Patrick Xia, Raghu Pappagari, Shuning Jin, R. Thomas McCoy, Roma Patel, Yinghui Huang, Edouard Grave, Najoung Kim, Thibault Févry, Berlin Chen, Nikita Nangia, Anhad Mohananey, Katharina Kann, Shikha Bordia, Nicolas Patry, David Benton, Ellie Pavlick, and Samuel R. Bowman. `jiant` 1.3: A software toolkit for research on general-purpose text understanding models. http://jiant.info/, 2019.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R Bowman. Blimp: A benchmark of linguistic minimal pairs for english. *arXiv preprint arXiv:1912.00582*, 2019.

Adina Williams, Nikita Nangia, and Samuel Bowman. 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, 2018. URL http://aclweb.org/anthology/N18-1101.

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

Chiyuan Zhang, Samy Bengio, Moritz Hardt, and Yoram Singer. Identity crisis: Memorization and generalization under extreme overparameterization. 2019. URL https://openreview.net/pdf?id=SkxT8ESh3E.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 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, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2003. URL https://www.aclweb.org/anthology/N18-2003.

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

Ran Zmigrod, Sebastian J. Mielke, Hanna Wallach, and Ryan Cotterell. 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, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1161. URL https://www.aclweb.org/anthology/P19-1161.

A Effect of hyperparameter settings

We vary several model and task settings for the Section 4 experiments, and we find that the models exhibit similar behavior as in the default case. The default settings are summarized in Table 5. 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 5: Hyperparameter settings (Section 4). 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 from Section 4 – in which we vary the number of adversarial counterexamples – at different vocabulary sizes. The results are shown in Figure 10. 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 3). 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 set-up that weaken this assumption for the Section 4 experiments. 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 for the Section 4 experiments. The results are in Figure 11. 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 for the Section 4 experiments 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 12 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, in the third case (the rightmost plot in Figure 12), the error of `contains first` exceeds that of the combined true property.

C Other error metrics

We include additional figures for our main results from Section 4 – hardness (Figure 13), *true-only* and *distractor-only* counterexamples (Figure 14), training size (Figure 16), multiple distractors graph (Figure 17), multiple distractors heatmap (Figure 6) – 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 16).

D Controls for training size

D.1 Adding non-adversarial examples

When adding adversarial counterexamples in Section 4, 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 18 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; see Figure 19. 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 4). 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 (§4.3.1) and increasing the number of counterexamples (these results) should not be attributed to the larger training size.

E Hyperparameters for Section 3

See `jiant` [Wang et al., 2019] for further details about implementation. Most hyperparamters are in Table 6.

Hyperparameter	**Value**
optimizer	BERT Adam
patience for LR scheduler	5
patience for early-stopping	20
batch size	16
initial learning rate	0.00001
min. learning rate	0.0000001
dropout	0.1
classifier	logistic regression
frozen?	No
BERT model	base-cased

Table 6: Hyperparameter settings (Section 3).

F Train time vs. LSTM norm

We present an incidental result in Figure 20 for Section 5, in which we see a correlation between the number of epochs a model trains for, and the L_1 norm of the LSTM (sum of weights). This is from the experiment from Section 4 that produced Figure 6 (variable numbers of both types of counterexample).

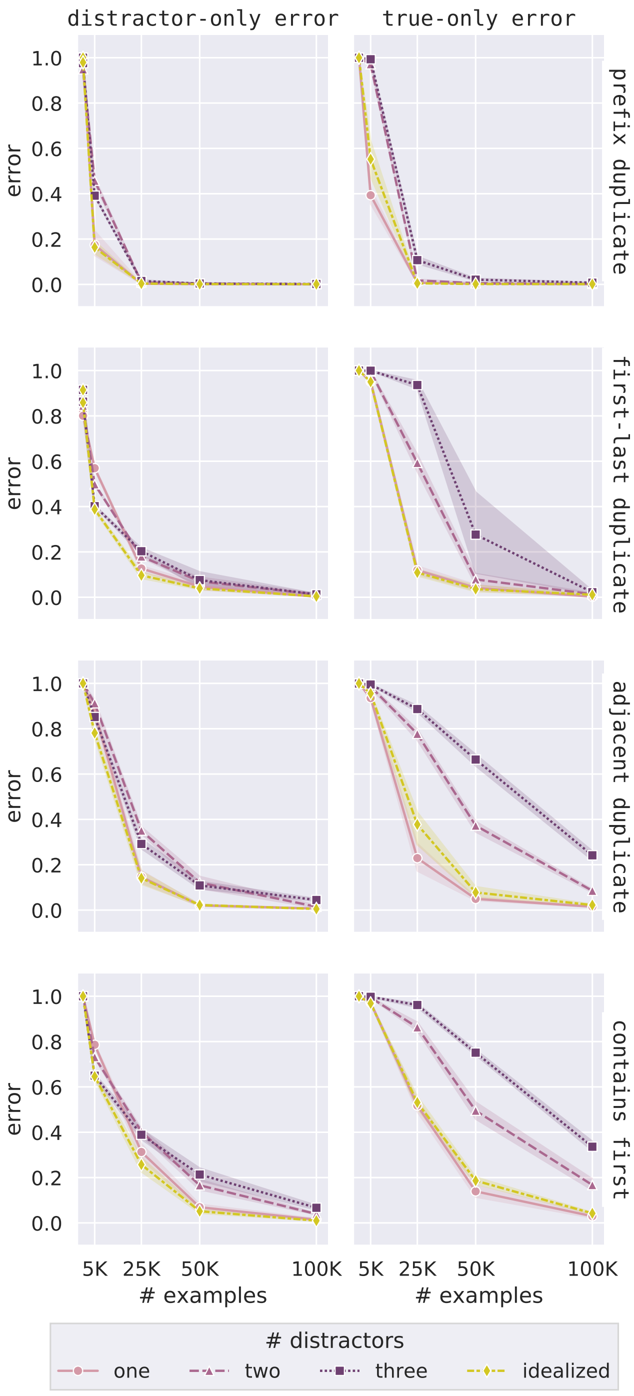

Figure 9: Error rate vs. number of counterexamples added for models trained in settings with different numbers of distractors, assuming that *true-only* counterexamples can guarantee the removal of at most one distractor at a time. Initial training size is 200K. Gold line shows performance in an idealized 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.

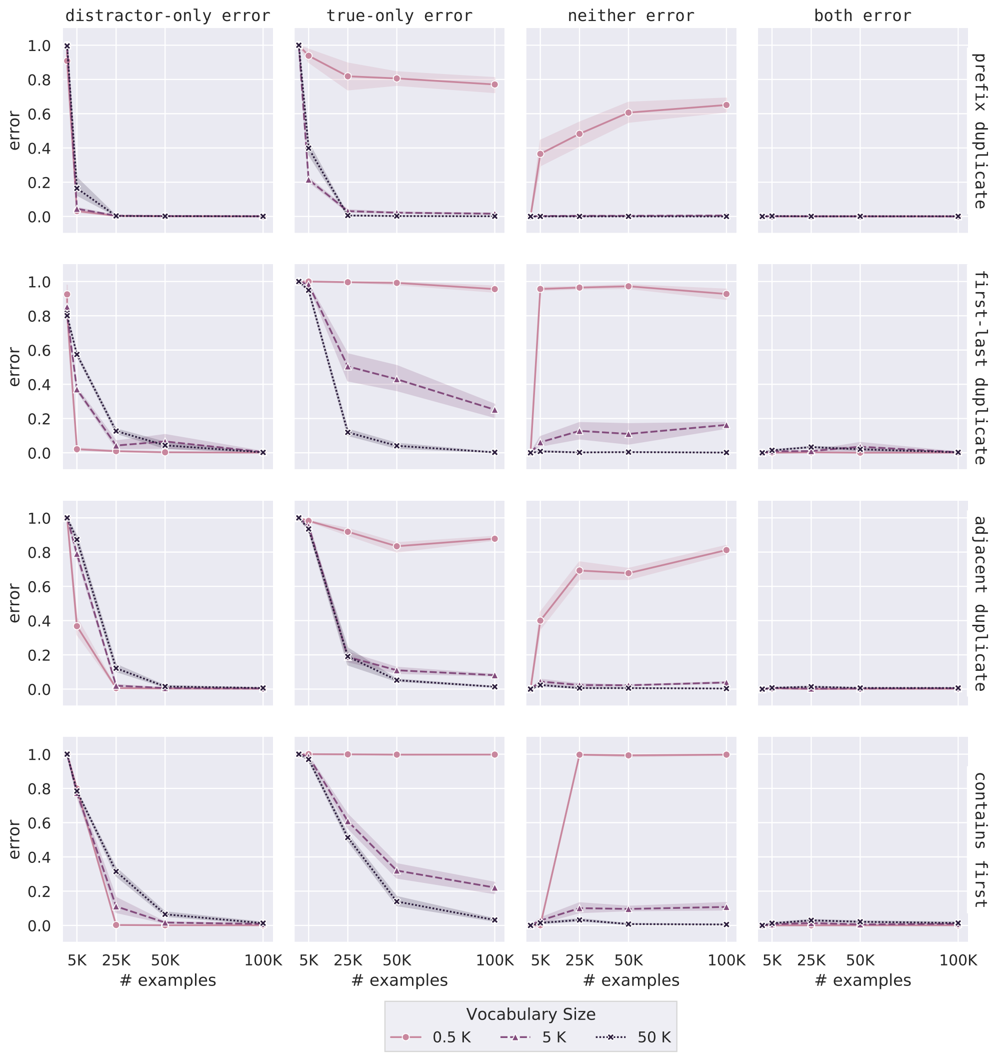

Figure 10: Vocabulary size (Section 4). The model isn't able to learn the true properties at a vocabulary size of 0.5K, but we observe similar behavior at 5K and 50K.

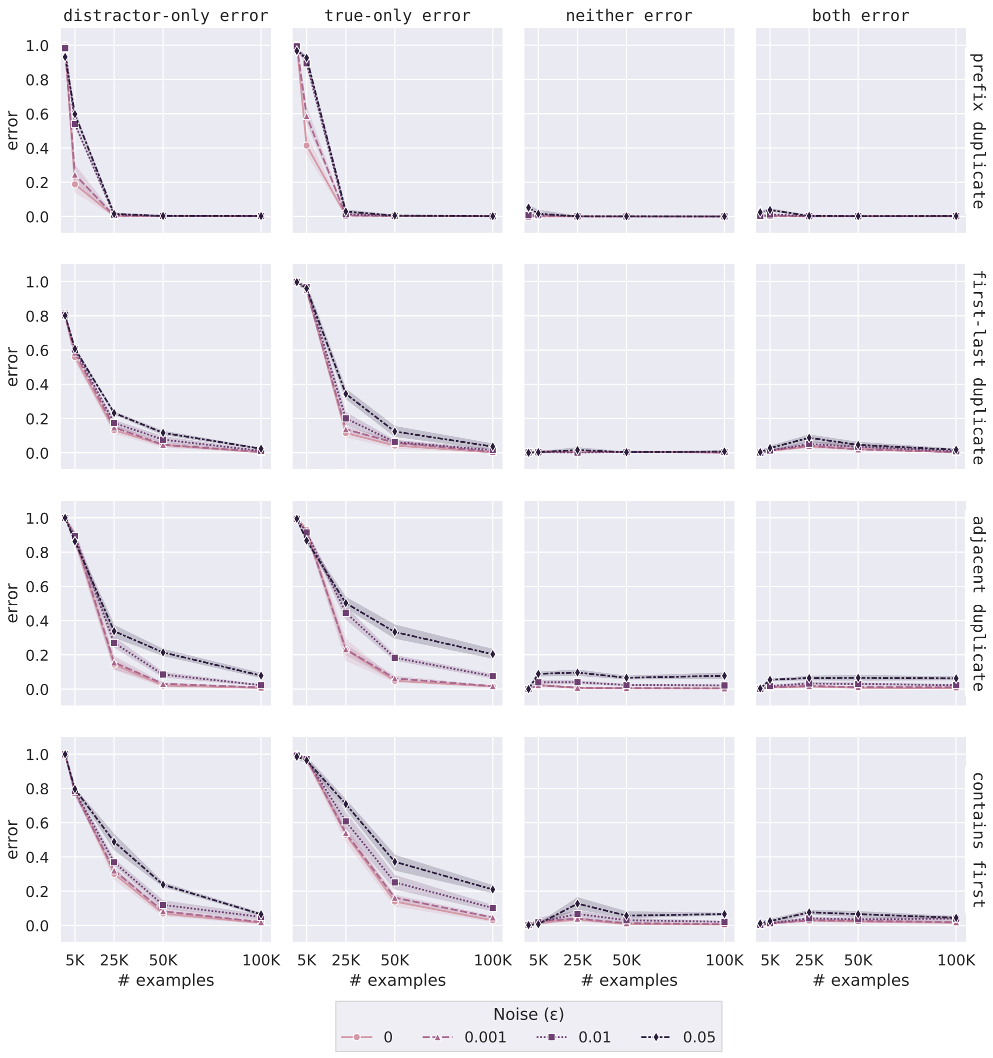

Figure 11: Noise (Section 4). We observe that adding noise does not affect the results, though more noise seems to make the model less prone to learning the true property.

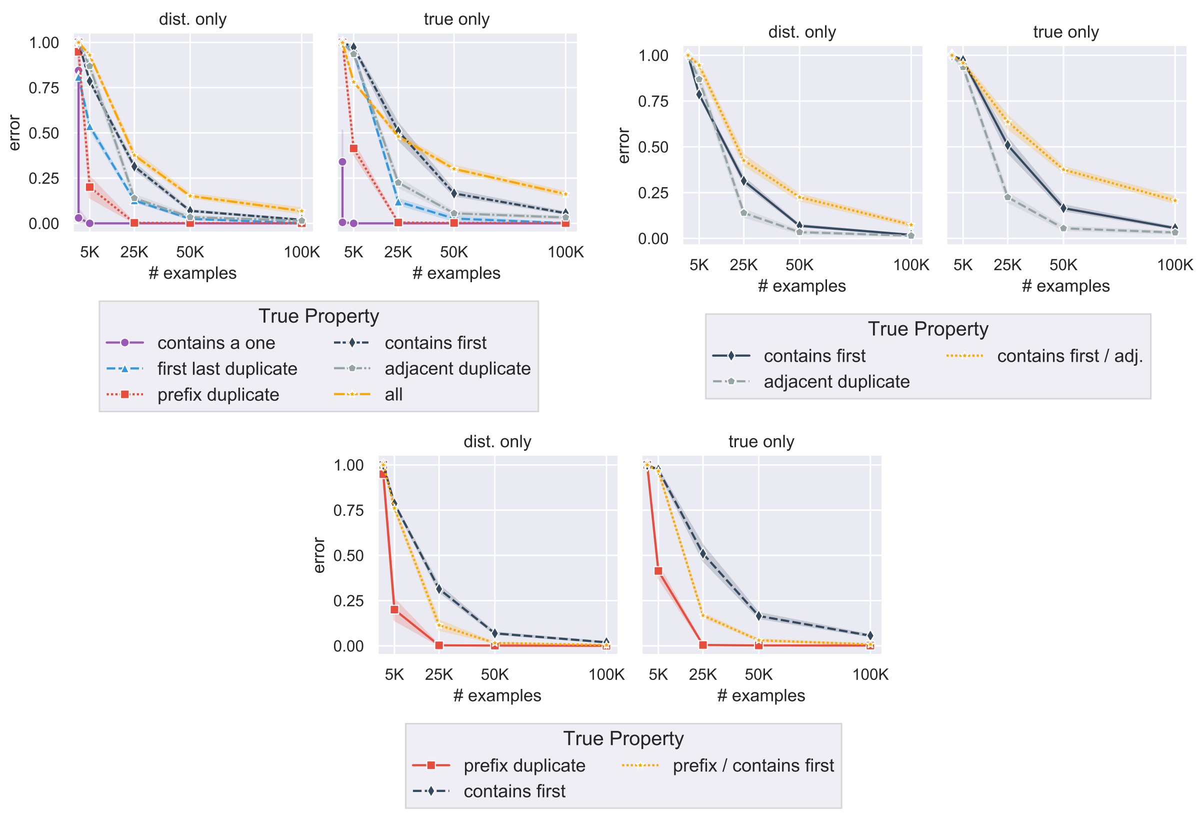

Figure 12: Multiple true properties (Section 4). We find similar trends as for a single true property. Collections of harder properties require more counterexamples to achieve low generalization error than collections of easier properties. We also find that collections of properties might take more counterexamples to achieve low generalization than any individual property in the collection.

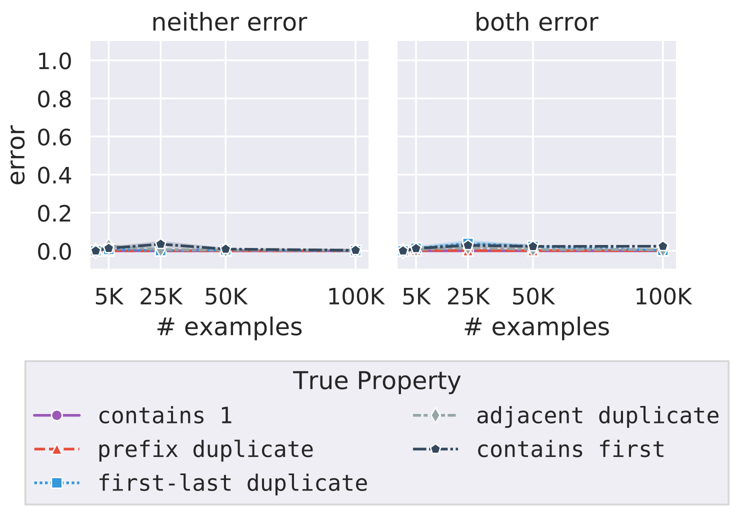

Figure 13: Hardness (Section 4). Other error cases (*neither* left and *both* right) for Figure 4.

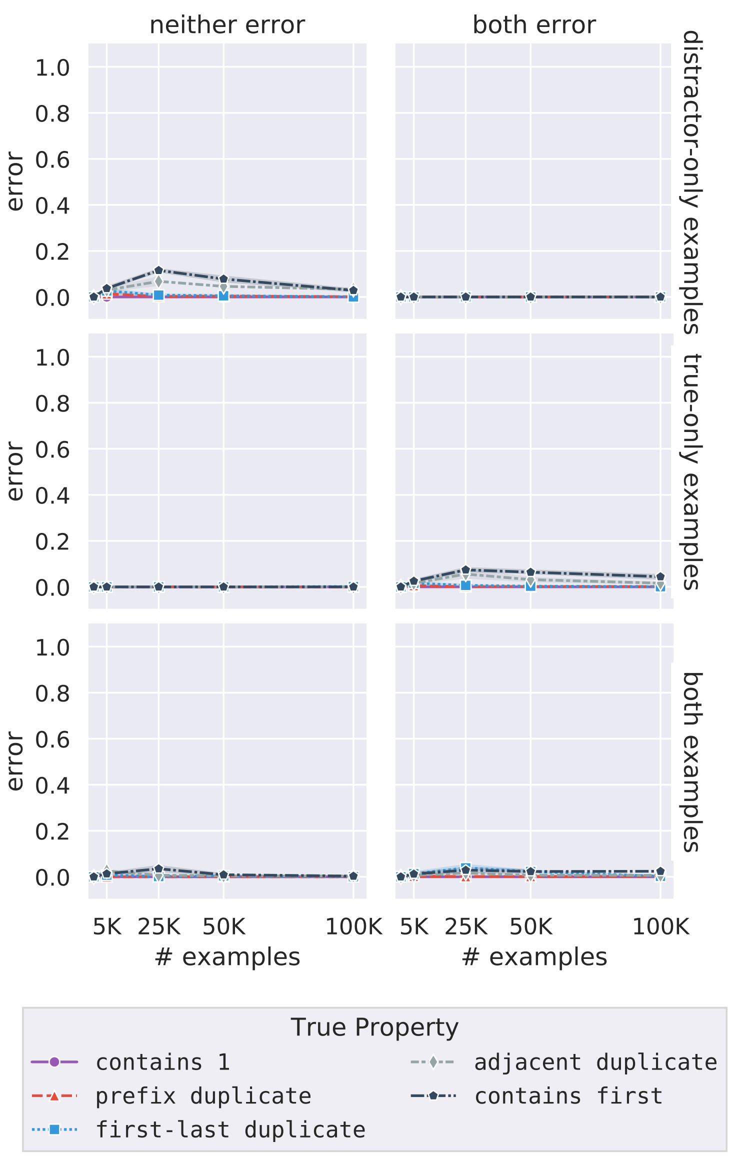

Figure 14: Counterexamples of each type (Section 4). Other error cases (*neither* left and *both* right) for Figure 5.

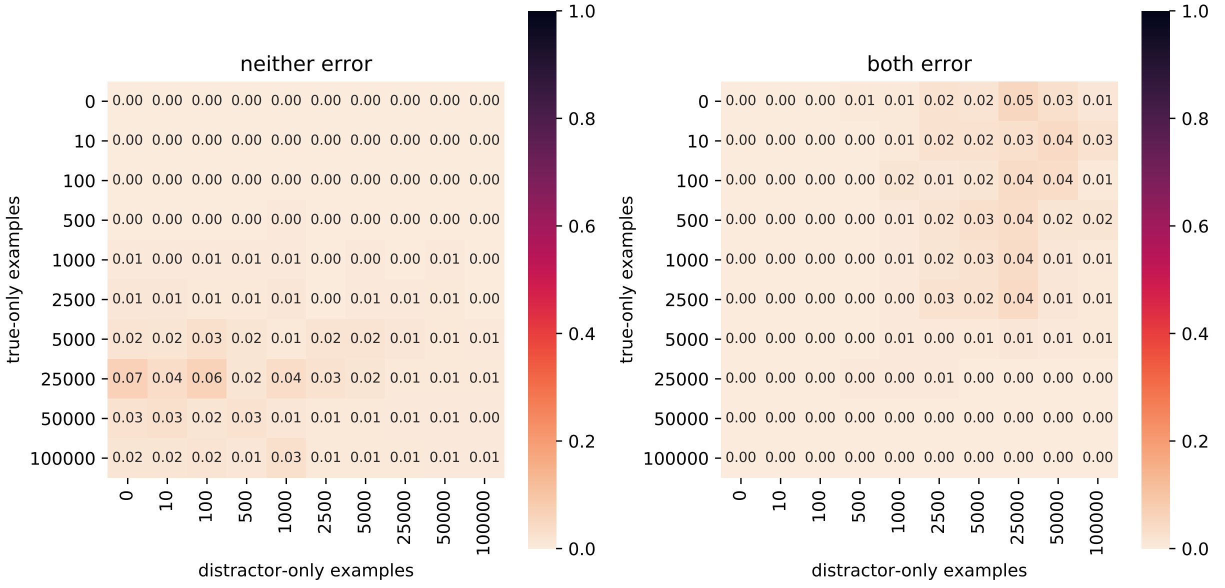

Figure 15: Counterexamples of each type (Section 4). Other error cases (*neither* left and *both* right) for Figure 6.

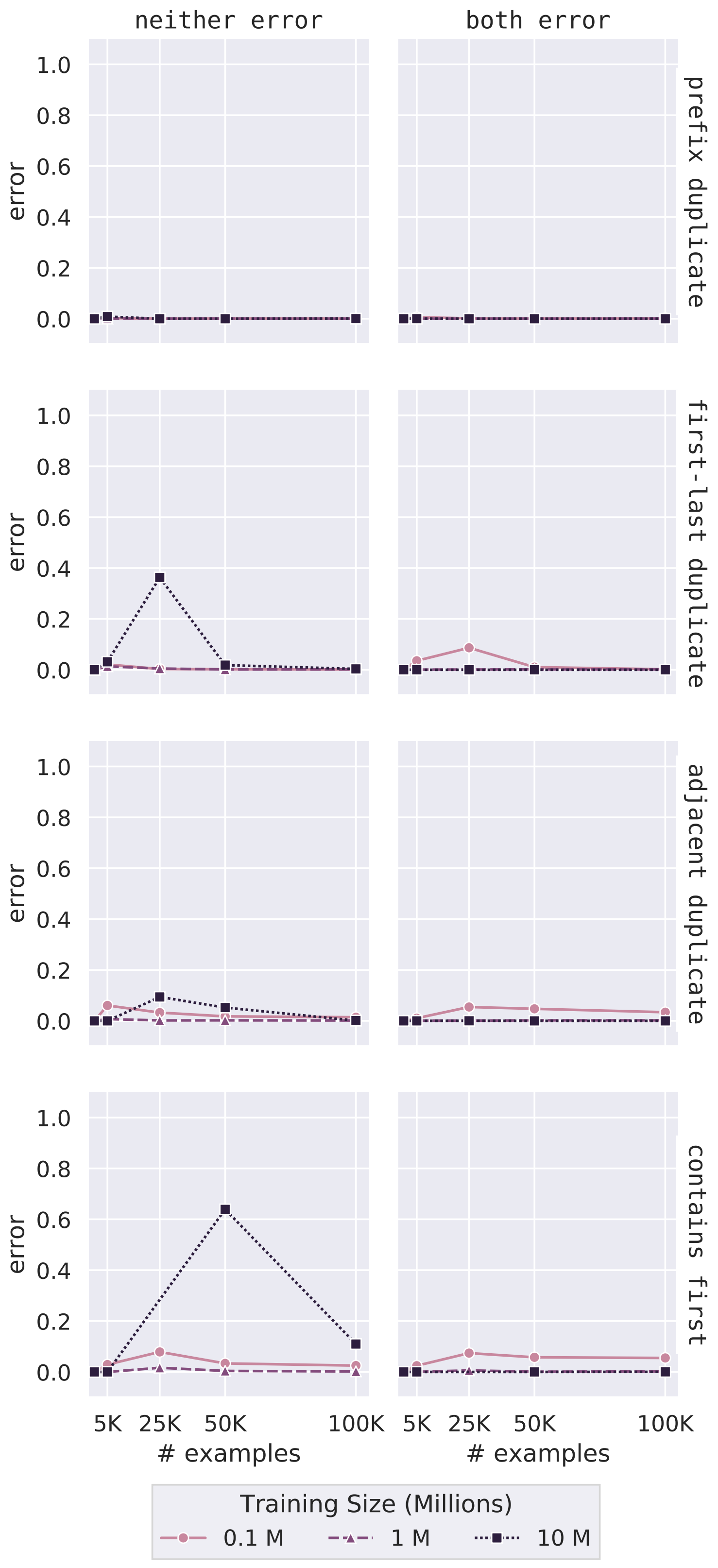

Figure 16: Varied training size (Section 4). Other error cases (*neither* left and *both* right) for Figure 8.

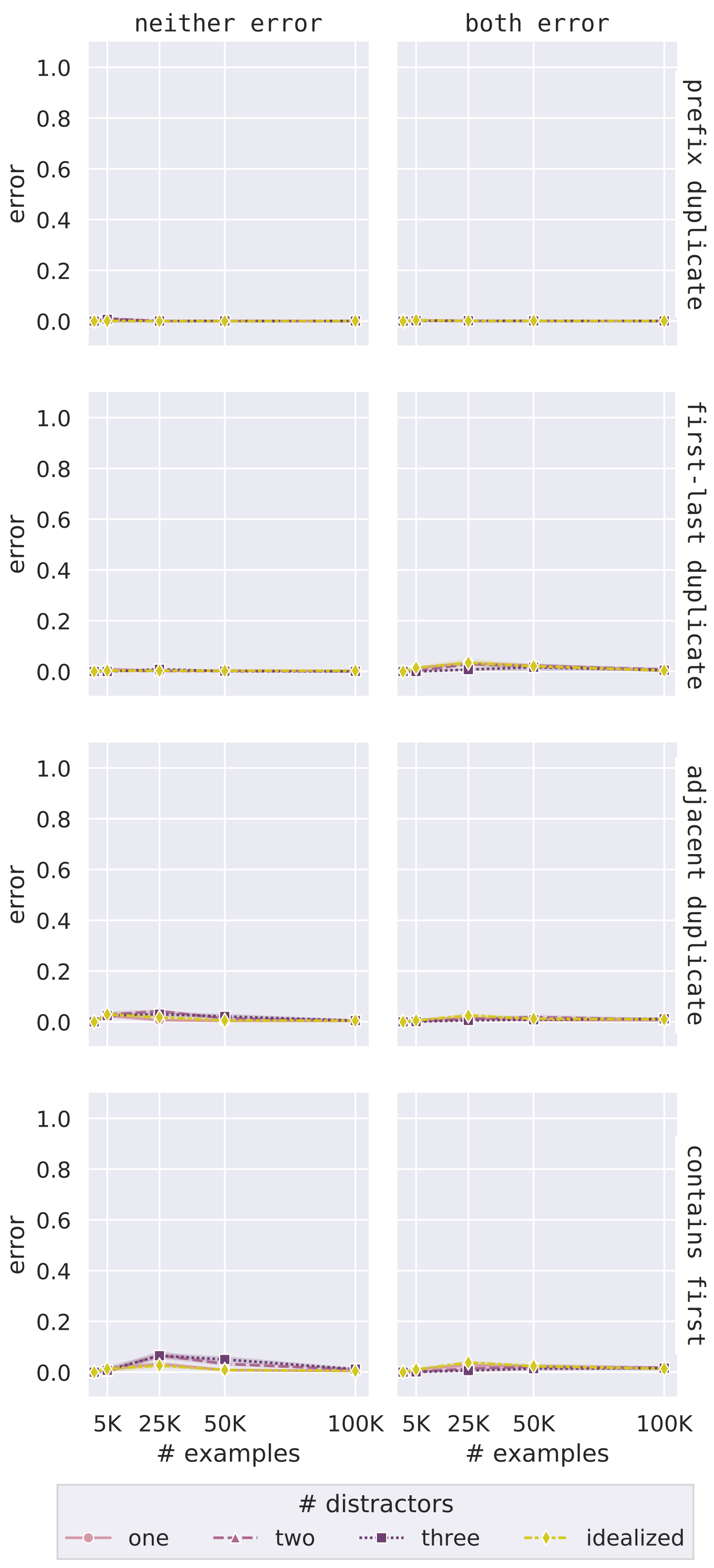

Figure 17: Multiple distractors (Section 4). Other error cases (*neither* left and *both* right) for Figure 9.

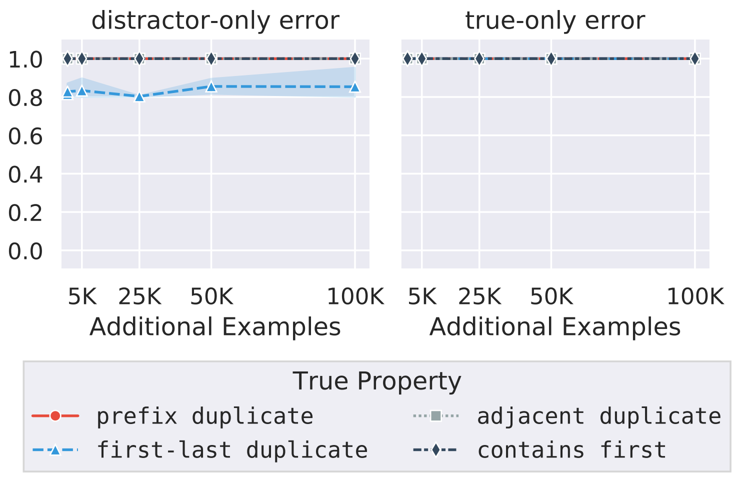

Figure 18: When adding adversarial counterexamples, the total number of training examples also increases (Section 4). We control for this change by showing here that additional "default" training examples do not help the model on either *distractor-only* or *true-only* error. This is unsurprising given our previously shown results.

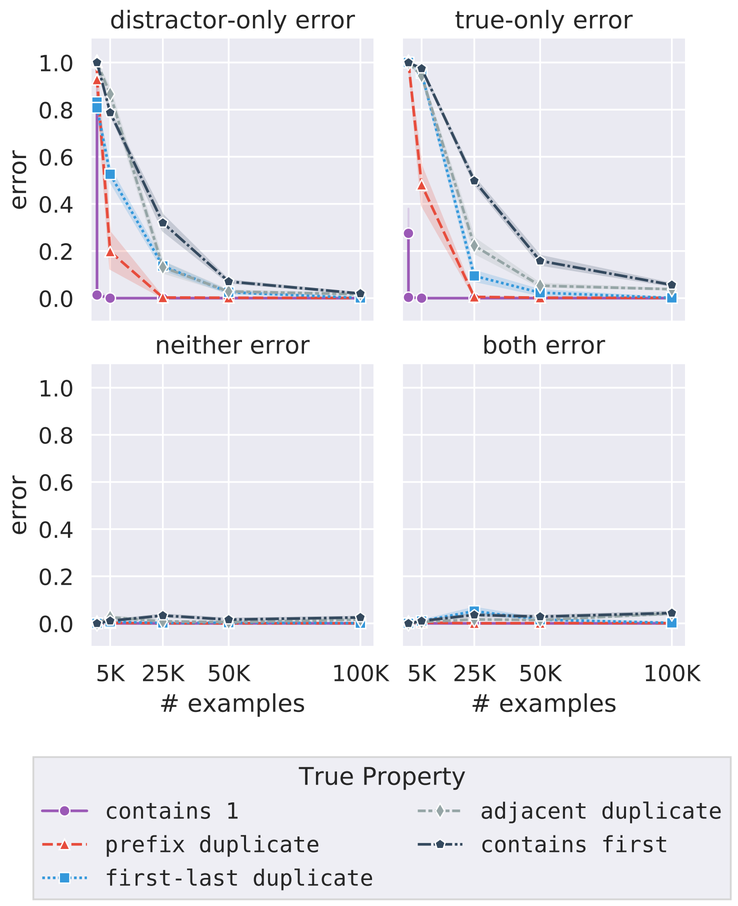

Figure 19: Static training size; counterexamples replace training examples (Section 4).

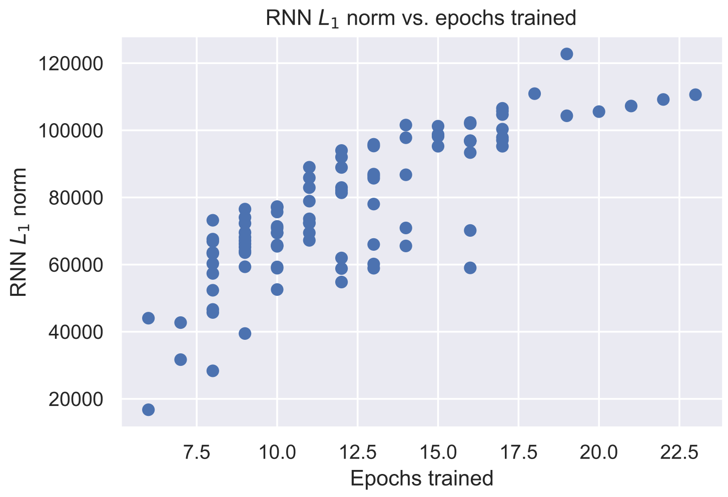

Figure 20: We see a correlation between the number of epochs a model trains for, and the L_1 norm of the LSTM (sum of weights) (Section 5).