Learning to Model and Ignore Dataset Bias with Mixed Capacity Ensembles

Christopher Clark[*]
Allen Institute for AI
chrisc@allenai.org

Mark Yatskar[†]
University of Pennsylvania
myatskar@seas.upenn.edu

Luke Zettlemoyer
University of Washington
lsz@cs.uw.edu

Abstract

Many datasets have been shown to contain incidental correlations created by idiosyncrasies in the data collection process. For example, sentence entailment datasets can have spurious word-class correlations if nearly all contradiction sentences contain the word "not", and image recognition datasets can have telltale object-background correlations if dogs are always indoors. In this paper, we propose a method that can automatically detect and ignore these kinds of dataset-specific patterns, which we call dataset biases. Our method trains a lower capacity model in an ensemble with a higher capacity model. During training, the lower capacity model learns to capture relatively shallow correlations, which we hypothesize are likely to reflect dataset bias. This frees the higher capacity model to focus on patterns that should generalize better. We ensure the models learn non-overlapping approaches by introducing a novel method to make them conditionally independent. Importantly, our approach does not require the bias to be known in advance. We evaluate performance on synthetic datasets, and four datasets built to penalize models that exploit known biases on textual entailment, visual question answering, and image recognition tasks. We show improvement in all settings, including a 10 point gain on the visual question answering dataset.

1 Introduction

Modern machine learning algorithms have been able to achieve impressive results on complex tasks such as language comprehension or image understanding. However, recent work has cautioned that this success is often partially due to exploiting incidental correlations that were introduced during dataset creation, and are not fundamental to the target task. Examples include textual entailment models learning the word "not" always implies contradiction (Gururangan et al., 2018), visual question answering (VQA) models learning "2" is almost always the answer to "How many" questions (Jabri et al., 2016), and question answering models selecting entities that occur near question words irrespective of context (Jia and Liang, 2017).

We call these kinds of dataset-specific correlations dataset bias. Models that exploit dataset bias can perform well on in-domain data, but will be brittle and perform poorly on out-of-domain or adversarial examples. Prior work (Clark et al., 2019; He et al., 2019; Wang et al., 2019; Bahng et al., 2020) has shown it is possible to prevent models from adopting biased methods, but require the bias to be known and carefully modeled in advance, e.g., by assuming access to a pre-specified classifier that uses only the bias to make predictions. In this paper, we present a debiasing method that can achieve similar results, but that automatically learns the bias, removing the need for such dataset-specific knowledge.

To make this possible, we observe that many known examples of dataset bias involve models learning overly simple patterns (Min et al., 2019; McCoy et al., 2019; Anand et al., 2018). This leads us to propose that many dataset biases will be shallower and easier to model than more generalizable patterns. This reflects the intuition that high-quality models for tasks like language comprehension or image understanding will require some minimum amount of complexity (e.g., a visual question answering model should at least consider the question, image, and ways they might correspond), and therefore shallower approaches are likely to be modelling dataset bias.

Our method, called Mixed Capacity Ensembling (MCE), follows prior work (Clark et al., 2019; He et al., 2019) by training an ensemble of two mod-

[*] Work completed at the University of Washington
[†] Work completed at the Allen Institute for AI

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 3031–3045
November 16 - 20, 2020. ©2020 Association for Computational Linguistics

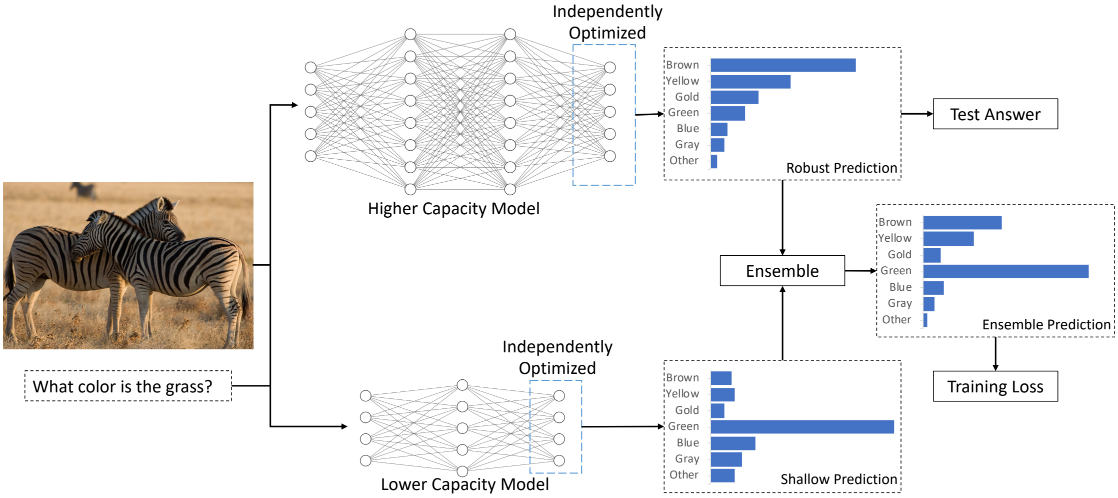


Figure 1: An overview of our method. We train a lower capacity model in an ensemble with a higher capacity model. During training simplistic correlations (e.g., "grass is usually green") are captured by the lower capacity model, which frees the higher capacity model to focus on more robust patterns (i.e., matching the question with the image). At test time, the higher capacity model is used alone. We use an independently optimized classifier as the final layer of each model as part of our method to make them conditionally independent (Section 2.4).

els, one of which captures bias, and one of which captures other, better generalizing patterns. Prior methods required that the model that captures the bias be pre-specified by domain experts. We instead achieve this separation by jointly training both models while encouraging simpler patterns to be isolated into one model and more complex patterns into the other. In particular, we (1) ensemble a lower capacity model, i.e., a model with fewer parameters, and a higher capacity model, which creates a natural tendency for the higher capacity model to capture more complex patterns, (2) put a small weight on the loss of the lower capacity model so that it is preferentially used to capture simpler patterns that it can model, and (3) enforce conditional independence between the models so that they learn non-overlapping strategies. We show that conditional independence can be achieved by ensuring each classifier makes individually optimal predictions, and we train the ensemble with this constraint using methods from bi-level optimization (Colson et al., 2007).

We evaluate our method by training models on datasets with known biases, and then testing them on out-of-domain datasets built to penalize models that learn to use those biases. First, we construct a series of synthetic datasets to show our method can adapt to multiple kinds of biases. Then, we consider three datasets from prior work that test against question-type biases for visual question answering (Goyal et al., 2018) and hypothesis keyword biases (Bowman et al., 2015; Gururangan et al., 2018) or lexical overlap biases (McCoy et al., 2019) for sentence entailment. Finally, we construct an image recognition dataset using Imagenet (Deng et al., 2009) that includes a test set of examples with misleading backgrounds (e.g., a fish photographed on dry land) to test our method on background-class biases. We show improved performance in all settings, in some cases nearly matching the results that can be achieved with an upper-bound that does use knowledge of the bias being targeted. We release our datasets and code to facilitate future work.[1]

2 Mixed Capacity Ensembles

In this section, we present our Mixed Capacity Ensembling method and the motivations behind it. We also discuss an extension to cases where shallow patterns can partially solve the task by eliminating obviously wrong answers.

2.1 Problem Definition

Let $\mathcal{X}$ be the domain of the input, $\mathcal{Y} = \{1, 2, \ldots, C\}$ be the space of the labels, and $\mathcal{B}_y$ be the space of probability distributions on $\mathcal{Y}$. Assume we have a training dataset of n examples, $\{(x_i, y_i)_{i=1}^n\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ and x_i, y_i are drawn from the joint distribution $P^{train}(x, y)$.

Our goal is to learn the parameters θ_h of a differentiable function f_h that returns a vector in $\mathbb{R}^C$

[1] https://github.com/chrisc36/autobias

representing a probability distribution over the possible classes, $f_h(\cdot, \theta_h) : \mathcal{X} \to \mathcal{B}_y$. For notational simplicity, we will sometimes write $f_h(x_i, \theta_h)$ simply as $f_h(x_i)$. Our goal is to optimize θ_h so that $f_h(\cdot, \theta_h)$ will have high accuracy on an out-of-domain test set drawn from $P^{test}(X, Y)$.

2.2 Motivation

Prior work has generally relied on domain-specific assumptions to make this task possible (e.g., question-only strategies will not generalize well to P^{test} for VQA). Our approach is designed to replace those assumptions with a more domain-general one: that overly simplistic patterns are unlikely to generalize to P^{test}.

While there is no guarantee all examples of dataset bias will fit this criteria, it is commonly true in the growing body of research on dataset bias. Additionally, we expect complex dataset biases to be less problematic because models will be less prone to fit to them.

Achieving this through regularization would be challenging since it is not obvious how to penalize the use of simplistic patterns. This motivates our approach of explicitly modeling simplistic hypotheses during training and discarding them during testing.

2.3 Training an Ensemble

Formally, our method introduces a lower capacity model: $f_l(\cdot, \theta_l) : \mathcal{X} \to \mathcal{Y}$ and additionally computes a class prior $p_y \in \mathcal{B}_y$ by computing the expected value of y in the training data. We then compute predictions for the ensemble, lower capacity model, and higher capacity model as follows:

$$\hat{y}_i^e = \mathit{softmax}(\log(f_h(x_i)) + \log(f_l(x_i)) + \log(p_y))$$
$$\hat{y}_i^l = \mathit{softmax}(\log(f_l(x_i)) + \log(p_y))$$
$$\hat{y}_i^h = \mathit{softmax}(\log(f_h(x_i)) + \log(p_y))$$

We explicitly factor in p_y so that it can be properly integrated into all three predictions (if the class prior was encoded as part of f_l and f_h it would be double-counted when the two functions are ensembled). During training the loss is computed as:

$$\mathit{Loss} = \sum_{i=1}^{n} L(\hat{y}_i^e, y_i) + wL(\hat{y}_i^l, y_i) \qquad (1)$$

where L is the cross-entropy loss function and w is a hyperparameter. During testing we make predictions using $\hat{y}_i^h$.

Following our simplicity assumption, we expect both f_l and f_h to be able to model dataset bias, but due to the additional loss on the output of f_l the ensemble will favor using f_l for that purpose. Additionally, since f_h can better represent more complex patterns, the ensemble will use f_h for that purpose.

2.4 Adding Conditional Independence

Although this creates a soft incentive for the models to learn different patterns, there is a risk this separation will not be completely clean (e.g., the higher capacity model might partially capture the relatively simple patterns we hope to model with the lower capacity model). To prevent this, we propose to enforce conditional independence between the models, meaning $f_l(X) \perp\!\!\!\perp f_h(X) | Y$ for random variables X, Y distributed according to P^{train}. We do not expect the models to be generally independent, since they will both be predictive of the label. Meeting this requirement while fulfilling the soft incentive created in the previous section pushes the ensemble to isolate simple/complex patterns in the lower/higher capacity models.

Most existing methods for enforcing conditional independence require penalizing some dependency measure between the two models. Here, we propose an alternative that takes advantage of the fact the models are being trained in an ensemble.

Theoretical Motivation: Let $g_h(x_i, \theta_h^g)$ and $g_l(x_i, \theta_l^g)$ be functions that produce feature vectors of size m (meaning $g_h(\cdot, \theta_h^g) : \mathcal{X} \to \mathbb{R}^m$). We will again sometimes omit the parameters θ_h^g and θ_l^g for notational simplicity.

For an example x, let $g_l(x) = x_l$ and $g_h(x) = x_h$. Our method is based on the observation that if x_h and x_l are conditionally independent on y:

$$\begin{aligned}
&P(y|x_h, x_l) && (2)\\
&\quad \propto P(x_h, x_l|y)P(y) && (3)\\
&\quad = P(x_h|y)P(x_l|y)P(y) && (4)\\
&\quad = \frac{P(y|x_h)P(x_h)}{P(y)} \frac{P(y|x_l)P(x_l)}{P(y)} P(y) && (5)\\
&\quad \propto \frac{P(y|x_h)}{P(y)} \frac{P(y|x_l)}{P(y)} P(y) && (6)
\end{aligned}$$

where (3) applies Bayes rule, (4) follows from conditional independence, and (5) applies Bayes rule twice. This shows that if x_h and x_l are conditionally independent, then $P(y|x_h, x_l)$ can be computed by ensembling $P(y|x_h)$ and $P(y|x_l)$.

Our key idea is to constrain the model so that $P(y|x_h, x_l)$ must be modelled in this way, i.e., by producing feature vectors x_l and x_h, modelling $P(y|x_l)$ and $P(y|x_h)$ individually, and then combining them following equation 6. Since equation 6 will only be a good model of $P(y|x_h, x_l)$ if x_h and x_l are conditionally independent, this creates a natural incentive to learn conditionally independent feature sets.

Intuitively, conditional independence is achieved because if a particular piece of information relevant to predicting y is present in both x_h and x_l, then that information's relationship to y will be captured in both $P(y|x_h)$ and $P(y|x_l)$, and thus will be over-used when these distributions are ensembled. Optimization should remove that information from either x_h and x_l to prevent this over-use.

Method: We approximate this constraint by constructing classifiers to model $P(y|x_h)$ and $P(y|x_l)$. Given a classifier $c(\cdot, \theta) : \mathbb{R}^m \to \mathcal{B}_y$, we compute:

$$\theta_h^{c*} = \arg\min_{\theta} \sum_{i=1}^{n} L(\hat{y}', y_i)$$
$$\hat{y}' = \mathit{softmax}(\log(c(g_h(x_i), \theta) + \log(p_y))$$

We then set $f_h(x_i) = c(g_h(x_i), \theta_h^{c*})$, and set $f_c(x_i)$ by following the same procedure to compute θ_l^{c*} for g_l. In short, f_h and f_l are now composed of feature extraction functions and classifiers, where the classifiers are fixed to optimally predict y instead of being trained on the end-to-end objective. As a result, we have $\hat{y}_i^h \approx P(y_i|g_h(x_i))$, in which case $f_h(x_i) \approx P(y|g_h(x_i))/P(y)$ (because we model the class prior separately from f_h when computing $\hat{y}_i^h$). Therefore the ensemble prediction $\hat{y}_i^e$ is computed following equation 6.

This method does not require adding additional hyperparameters, but it does mean minimizing the loss specified in equation 1 becomes a bi-level optimization problem (Colson et al., 2007) because it contains the subproblem of computing θ_h^{c*} and θ_l^{c*}.

Addressing Discontinuities: A complication with this method is that the argmin operations might not be smooth: a small change in $g_h(x_i)$ could dramatically change θ_h^{c*} and therefore f_h. We solve this by regularizing the classifier so that:

$$\theta_h^{c*} = \arg\min_{\theta} \sum_{i=1}^{n} L(\hat{y}', y_i) + \alpha\Omega(\theta)$$

Where $\Omega(\theta)$ is the L2 norm of the weights. This can also help ensure the optimization problem is well-defined. The hyperparameter α can be tuned by shrinking it until learning is no longer smooth. We find that our method is effective across a range of values for α, and we fix $\alpha = 0.002$ for all experiments in this paper. Algorithm 1 contains a review of the end-to-end procedure to compute the loss.

Algorithm 1 Computing the loss used in MCE. Here we use x and y to refer to the features and labels of a batch of examples.

```
function ComputeLoss($w, \alpha, x, y, \theta_h, \theta_l, p_y$)
    $o_h$ = Optimize($\alpha, g_h(x, \theta_h), y, p_y$)
    $o_l$ = Optimize($\alpha, g_l(x, \theta_l), y, p_y$)
    $\hat{y^l} = \mathit{softmax}(o_l + p_y)$
    $\hat{y^e} = \mathit{softmax}(o_l + o_h + p_y)$
    return $L(\hat{y}^e, y) + wL(\hat{y}^l, y)$
function Optimize($\alpha, z, y, p_u$):
    $\theta^* = \arg\min_\theta L(\hat{y}', y) + \alpha\Omega(\theta)$
    $\hat{y}' = \mathit{softmax}(\log(c(z, \theta) + \log(p_y)))$
    return $c(z_h, \theta^*)$
```

2.5 Implementation

In practice, we set m to be the number of classes, and build g_h by using the pre-softmax logits from a standard classification model as features. We set c_h and c_l to be residual affine functions so that $c_h(g_h(x_i), \theta_h^c) = g_h(x_i)W_h^c + b_h^c + g_h(x_i)$. Using more powerful functions for c could improve the approximation $\hat{y}_i^l \approx P(y|g_h(x_i))$, but linear functions are easier to optimize and sufficient in practice. Residual functions are used so the classifiers can pass-through their input with incurring a regularization penalty.

Optimization: We minimize the loss through minibatch gradient descent. During the forward pass we compute θ_h^{c*} and θ_l^{c*} for the minibatch, which essentially requires optimizing a small logistic regression model, and can be done quickly using existing black-box solvers and warm-starting from previous solutions. Once computed, the gradient of the input features with respect to θ_h^{c*} has a closed-form solution as shown in Gould et al. (2016), allowing us to backpropagate gradients through θ_h^{c*} and θ_l^{c*} to g_h and g_l.

To help ensure the θ_h^{c*} and θ_l^{c*} computed on each minibatch results in a good approximation of $P(y|g_h(x_i))$ and $P(y|g_l(x_i))$, we train with large batches (at least 256 examples) and stratify examples so each class is well represented in each batch.

Evaluation: Computing θ_h^{c*} and θ_l^{c*} requires using the example's labels, which are not available at test time. Therefore we use values precomputed on a large sample of the training data when classifying unlabelled examples.

2.6 Answer Candidate Pruning

One risk with this approach is that, while simplistic methods should not be relied upon to entirely solve the task, in some cases they could plausibly be used to eliminate some answer options. For example, given a "How many" question in VQA, it can be determined the answer should be a number without looking at the image or the other words. However, our method might factor such simple heuristics into the lower capacity model.

For tasks where this is a possibility, we propose to extract this answer-elimination ability from the lower capacity model through thresholding. At test time we eliminate all answer candidates that the lower capacity model assigns a probability less than some (very conservative) threshold t, and select among the remaining candidates using the higher capacity model. This follows the intuition that the simplistic patterns captured by the lower capacity model might still be expected to eliminate very obviously wrong answers in a generalizable way.

3 Experimental Setup

We apply our method to datasets that have known biases, and evaluate the trained model on adversarial datasets that were constructed to penalize models that make use of those biases. This is not a perfect evaluation because models might be unfairly penalized for ignoring biases that were unaccounted for when the adversarial dataset was constructed, and therefore still exist in the test set. However, positive results on these cases provide good evidence that our method was at least able to identify the bias the adversarial dataset targeted.

We use standard, high-performing models for the higher capacity models. To help ensure the lower capacity models can capture a wide range of possible biases, we pick models that achieve strong results if trained on their own, while still having significantly fewer (i.e., half or less) parameters then the higher capacity model.

For each dataset, we evaluate on an out-of-domain (OOD) and in-domain (ID) test set. All reported results are averages of ten runs unless otherwise specified. We provide overviews of the datasets and models used, but leave other training details to the appendix.

3.1 Comparisons

We compare MCE with two ablations, a baseline approach to conditional independence using an adversarial method, and an upper bound that uses domain-specific knowledge of the bias.

No BP: We train the ensemble without backpropagating through the argmin operators that compute the parameters of the top-level classifiers, c_l and c_h. Instead the parameters are treated as constants. This tests whether it is necessary to optimize the model using bi-level optimization methods, or if a more ad-hoc approach would have been sufficient.

No CI: The ensemble is trained without our approach to enforcing conditional independence, meaning it is trained as specified in Section 2.3.

With Adversary: The ensemble is trained while replacing our conditional independence method with the 'Equality of Odds' adversarial approach from Zhang et al. (2018). This approach trains linear classifiers that use the log-space output of either the higher or lower capacity model, and a one-hot encoding of the label, to predict the output of the other model. The two classifiers are trained simultaneously with the higher and lower capacity models, which in turn are adversarially trained to increase the loss of those classifiers. Since we observe it can cause training to diverge, we do not backpropagate gradients from the adversarial loss to the model providing the labels for the classifiers.

Pretrained Bias: Following Clark et al. (2019), we construct a bias-only model by training a model that is hand engineered to make predictions using only the target bias (e.g., a hypothesis-only model for MNLI). The high capacity model is then trained in an ensemble with that pre-trained model. This method makes use of precise knowledge about the target bias, so we use it as an approximate upper bound on what could be achieved if the target bias was perfectly factored into the lower capacity model. See the appendix for details.

3.2 Applying Prior Work

We attempted to apply the methods from Clark et al. (2019) by having the lower capacity model fill the role of a bias-only model. This means training the lower capacity model on its own, freezing its parameters, and then training the higher capacity model in an ensemble with the pre-trained lower capacity model. However, we found this always leads to poor performance, sometimes to levels close to random chance. The likely cause is that the lower capacity models we use are strong enough to learn far more than just the bias when trained on their own. Therefore the methods from Clark et al. (2019), which prevent the higher capacity model from using strategies the lower capacity model learned, leads to the higher capacity model avoiding valid approaches to the task.

3.3 Same Capacity Sanity Check

We also sanity checked our method by training MCE using the same model for both the lower and higher capacity models. This consistently leads to poor performance, often worse then training the model without any debiasing method, so we do not show results here. We found the lower capacity model often ended up performing as well, or better, then the higher capacity on the in-domain data, which suggests it had learned complex patterns.

3.4 Hyperparameters

We use OOD development sets drawn from the same distribution as the OOD test sets for hyperparameter tuning. As pointed out by prior work (Clark et al., 2019), this is not an ideal solution since it means some information about the OOD distribution is used during training. As a best-effort attempt to mitigate this issue, we use a single hyperparameter setting for MCE ($w = 0.2$) on all our experiments. Although setting this parameter did require using examples from the OOD dev sets, the fact a single value works well on a diverse set of tasks suggests that our method will generalize to settings where tuning hyperparameters on the OOD test distribution is not possible. Results with optimized hyperparameters are in the appendix.

The With Adversary baseline also requires hyperparameter tuning. We were unable to find a universally effective setting, so we report results using the best setting found on the OOD dev set. This is indicated by a * next to that method.

4 Results

4.1 Synthetic MNIST

We build synthetic datasets by adding a synthetic bias to the MNIST images (LeCun et al., 1998). In particular, we design a superficial image modification for each label, apply the modification that corresponds to each image's label 90% of the time, and apply a different, randomly selected modification otherwise. OOD sets are built by applying randomly selected modifications. We use 200 examples for each digit for training, 1000 examples per digit for the OOD dev set, and 1000 per digit for the OOD and ID test sets. Runs are averaged over 100 trials. We use three kinds of modifications in order to demonstrate our method is effective against multiple types of bias.

Background: The background of the digit is colored one of 10 colors depending on the label.

Patch: The background is divided into a 5x5 grid, the upper left patch is colored to match the label, and the rest of the patches are colored randomly.

Split: Following Feng et al. (2019), the label is encoded in two separate locations in the image. Each label is mapped to a set of color pairs. Then the image is divided into vertical stripes, the center strip is colored randomly, and the other strips are colored using a randomly selected color pair for the example label. To avoid an excessive number of color pairs, we only use the digits 1-8, and map those digits to four super classes.

Higher Capacity Model: We use a model with one convolution layer with 8 7x7 filters and ReLU activation, followed by a 128 dimensional ReLU layer and a softmax predictor layer.

Lower Capacity Model: We use a model with a 128 dimensional ReLU layer then a softmax layer.

Results: Table 1 shows the results. The Patch bias proves to be the hardest, possibly because the patchwork background distracts the model, while the more subtle Split bias is the easiest. Despite using no knowledge of the particular bias being used, our method improves upon training the model naively by at least four points, in two cases slightly out-performing the Pretrained Bias method. Using the adversary is comparable in some cases, but falls behind on the Patches bias.

4.2 VQA

We evaluate on the VQA-CP v2 dataset (Agrawal et al., 2018), which was constructed by re-splitting VQA 2.0 (Goyal et al., 2018) data into new train and test sets such that the correlations between question types and answers differs between each split. For example, "white" is the most common answer for questions that start with "What color is..." in the train set, whereas "black" is the most common answer for those questions in the test set. We hold out 40k examples from the train set to serve

Method	Background		Patch		Split	
	OOD Acc	ID Acc	OOD Acc	ID Acc	OOD Acc	ID Acc
MCE (Ours)	81.21	93.92	74.34	86.70	92.36	93.69
No CI	78.75	94.95	69.40	85.53	91.29	93.94
No BP	78.39	94.34	71.93	86.67	92.35	93.78
With Adversary*	80.57	93.37	70.31	82.78	91.24	93.43
None	67.76	95.46	58.53	90.34	88.77	94.72
Pretrained Bias	84.59	91.17	72.96	79.72	91.27	91.61

Table 1: Accuracy on MNIST with synthetic biases when those biases are removed (OOD) or preserved (ID).

as an ID test set, and 40k of the 200k examples in the test set for an OOD dev set.[2]

Our model follows standard practice of predicting an answer from a set of pre-selected answer candidates. Since many of the answers are uncommon, and thus will be poorly represented in individual mini-batches, we cannot apply our conditional independence method out of the box. Instead, we cluster answer candidates by putting the 10 most common answers in individual clusters and the rest in an 11th cluster, and then apply our method while sharing parameters between answers in the same cluster. Since the model uses a sigmoid prediction function, we make g a simple two-parameter function that rescales and shifts the input.

For this dataset, we additionally show results when using the answer candidate pruning method from Section 2.6 for models where it is applicable. We pick a conservative threshold such that the correct label would be pruned less than 0.1% of the time on in-domain data.

Higher Capacity Model: We use LXMERT (Tan and Bansal, 2019), a transformer based model that has been pretrained on image-caption data.

Lower Capacity Model: We make predictions using mid-level representations from the higher capacity model (see the appendix for details).

Results: Table 2 shows the results. MCE closes most of the gap between the basic model and the upper bound, suggesting it was able to identify the question-type bias. The baselines underperform MCE by a significant margin, while answer candidate pruning offers a consistent boost.

4.3 MNLI

We evaluate on the MNLI Hard sentence pair classification dataset from Gururangan et al. (2018) and the HANS dataset from McCoy et al. (2019).

Method	OOD Acc		ID Acc	
	Acc	w/o AP	Acc	w/o AP
MCE (Ours)	68.44	66.10	74.03	72.72
No CI	59.10	37.32	67.56	49.28
No BP	61.64	47.55	67.99	55.54
With Adversary*	66.08	26.16	72.17	37.47
None	57.65	-	76.95	-
Pretrained Bias	70.32	-	70.78	-

Table 2: Results on the VQA-CP OOD test set, and a held out ID test set. We show accuracy with and without answer pruning. Note the ID set is for VQA-CP, and is not comparable to the standard VQA 2.0 dev set.

The MNLI Hard dataset is built by training a classifier to predict the target class using only the hypothesis sentence, and then filtering out all examples from the dev set that this classifier is able to classify correctly. Our classifier reaches 54% accuracy on the matched dev set (compared to 33% from random guessing) by making use of correlations between certain words and the class (e.g., “not” usually implies the class is “contradiction”). We use 4.4k examples filtered from the MNLI matched dev set as an OOD dev set, and another 4.4k examples filtered from the mismatched dev set as the OOD test set. We use the entire 9.8k mismatched dev set for the ID test set.

The HANS dataset contains 30k examples where both sentences contain similar words, so models that naively classify such sentences as ’entailment’ will perform poorly. We do not tune hyper-parameters on HANS, and instead use the same settings as MNLI Hard.

We evaluate each model on both adversarial sets, so doing well requires models to be simultaneously robust to both the biases being tested for. We build a Pretrained Bias model for hypothesis-only bias, and report the best ensemble result from Clark et al. (2019) as an upper bound for the HANS dataset.

Higher Capacity Model: We use the pre-trained uncased BERT-Base model (Devlin et al., 2019).

[2] Prior work tuned hyper-parameters directly on the test set, but we think its preferable to shrink the test set slightly in order to have a separate OOD dev set

Method	Hard	HANS	ID
MCE (Ours)	77.58	64.43	83.28
No CI	77.24	64.25	83.38
No BP	76.44	62.18	83.88
With Adversary*	77.10	63.03	83.35
None	75.62	61.04	84.28
Pretrained Bias	79.78	62.23	76.90
Clark et al. (2019)	-	67.92	-

Table 3: Accuracy on two OOD textual entailment datasets (MNLI Hard and HANS) and the mismatched dev set performance (ID) .

Method	OOD Acc			ID Acc
	5%	10%	20%	All
MCE (Ours)	80.01	81.72	83.84	90.05
No CI	80.97	82.29	84.25	89.83
With Adversary*	79.16	81.05	83.42	89.91
None	78.42	80.53	83.22	90.70
Pretrained Bias	87.71	87.54	86.93	78.38

Table 4: Results on ImageNet animal recognition on examples where less than 5%, 10%, or 20% of the image-patch classifiers are accurate (OOD), as well as on the entire test set (ID).

Lower Capacity Model: We use a modified version of the neural bag-of-words model from Parikh et al. (2016)[3].

Results: Table 3 shows the results. BERT-Base is already reasonably robust to the hypothesis-only bias, with only a 4% gap between the upper bound and the unmodified model, but is more vulnerable to the word-overlap bias in HANS. Our method is able to almost cut the gap in half, with the adversarial approach to conditional independence slightly underperforming MCE.

4.4 ImageNet Animals

We build an image recognition dataset that tests the ability of models to avoid learning background-class correlations. Since training models on ImageNet is computationally expensive, and the large number of classes creates complications when applying our conditional independence approach (i.e., we would have take steps to prevent classes becoming too sparsely represented in each minibatch, as we did for VQA), we build a simplified animal classification dataset by grouping various animal classes into 6 super-classes. See the appendix for details.

We build a train, dev, and test set with 10k, 3k, and 7k images per class respectively. Similarly to our approach to MNLI, and Hendrycks et al. (2019), we construct OOD datasets by filtering out correct predictions made by a biased classifier. Our biased classifier modifies the ResNet-18 model (He et al., 2016) to build features for 9x9 image patches following BagNet (Brendel and Bethge, 2019). We then train a classifier to predict the class using those features. Images where the classifier was unable to guess the correct class for most of the image patches are assumed to have a misleading background and are used for the OOD test set. Examples are shown in the appendix.

Higher Capacity Model: We use ResNet-18 (He et al., 2016).

Lower Capacity Model: We branch the higher capacity model by adding a 1x1 convolution with 256 filters after the first residual block, followed by max-pooling and a 256 dimensional ReLU layer.[4] This model gets approximately 81% on our dev set if trained alone.

Results: Table 4 shows the results. This dataset proves to be challenging; MCE provides a boost over naively training the model, but is still significantly below our estimated upper bound. Our conditional independence method reduces performance here, and training fails to converge for the no-backpropagation ablation, so results for that method are not shown. This appears to be because the model is able to reach nearly 100% accuracy on the training data which causes the argmin operations to become degenerate.

4.5 Discussion

Overall, we are able to improve out-of-domain performance in all settings even though our method does not use dataset-specific information about the target bias. Our conditional independence method generally improved performance while the adversarial baseline, despite getting the benefit of per-dataset hyperparameter tuning, was less effective.

MCE decreases ID performance, which is expected since the bias is helpful on in-domain data and the goal of our method is to remove it. However the decrease is often much less than for the upper bound (e.g., on MNLI MCE is 2 points behind on the OOD test set, but 6 points ahead on

[3] We were unable to get positive results using layer 3 or 6 of the BERT model, possibly because even the lower layers of BERT have a lot of representational power

[4] We found positive, but slightly weaker results using features after the second residual block, and negative results when using features after the third residual block

the ID test set). A possible cause is that the bias is only being partially factored out. Improving OOD performance without losing much ID performance might also suggest MCE is helping improve the higher capacity model in general. Better understanding and making use of this phenomenon is an interesting avenue for future work.

We find computing the argmin operations adds a moderate computational overhead, requiring about 2x the train time for the ResNet-18 ensemble and about 1.3x the time for the larger BERT and LXMERT ensembles (performance during evaluation is identical). Our implementation performs the optimization on the CPU; a GPU based optimizer might reduce this overhead.

5 Related Work

Prior work has shown that, given precise knowledge of a dataset bias, it is possible to train a debaised model. For example, if it is known particular intermediate representations in a network could only contain features that are shallowly indicative of the class due to bias (e.g., a local patch in an image), adversarial networks can be used (Belinkov et al., 2019; Grand and Belinkov, 2019; Wang et al., 2019; Cadene et al., 2019). Alternatively, given a model that is so tightly constrained it can only utilize dataset bias (e.g., a question only model for VQA), REBI (Bahng et al., 2020) employs a conditional independence penalty between that model and a debaised model, HEX (Wang et al., 2019) constructs a feature space that is orthogonal to the one learned by the bias model, and Clark et al. (2019) and He et al. (2019) pre-train the bias model and then train a debiased model in an ensemble with it. Our approach also makes use of ensembling, and the idea of requiring conditional independence between the models. However, our method identifies biased strategies during training instead of requiring them to be pre-specified.

Additional work has used multi-label annotations (Singh et al., 2020), pixel-level annotations (Hendricks et al., 2018), or other annotated features (Kim et al., 2019) to help train debiased models, although these kinds of annotations will not always be available.

There are also debiasing strategies that identify hard examples in the dataset and re-train the model to focus on those examples (Yaghoobzadeh et al., 2019; Li and Vasconcelos, 2019; Le Bras et al., 2020). This approach reflects a similar intuition that simplicity is connected to dataset bias, although our method is able to explicitly model the bias and does not assume a pool of bias-free examples exist within the training data.

Another debiasing approach is to use pretraining (Lewis and Fan, 201; Carlucci et al., 2019) or carefully designed model architectures (Agrawal et al., 2018; Zhang et al., 2016; Carlucci et al., 2019) to make models more prone to focus on semantic content. We expect these methods to be complementary to our work; for instance we show we can improve the performance of the extensively pretrained BERT and LXMERT models.

A related task is domain generalization, where the goal is to generalize to a unseen test domain given multiple training datasets from different domains (Muandet et al., 2013). Most domain generalization methods learn a data representation that is invariant between domains through the use of domain-adversarial classifiers (Ganin et al., 2016; Li et al., 2018b), ensembling domain-specific and domain-invariant representations (Bousmalis et al., 2016; Ding and Fu, 2017) or other means (Arjovsky et al., 2019; Li et al., 2018a; Xu et al., 2014; Ghifary et al., 2015). Our approach is similar to the ensembling methods in that we explicitly model generalizable and non-generalizable patterns, but does not require multiple training datasets.

6 Conclusion

We have presented a method for improving out-of-domain performance by detecting and avoiding overly simple patterns in the training data. Our method trains an ensemble of a lower capacity model and a higher capacity model in a way that encourages conditional independence given the label, and then uses the higher capacity model alone at test time. Experiments show this approach successfully prevents the higher capacity model from adopting known biases on several real-world datasets, including achieving a 10-point gain on an out-of-domain VQA dataset.

Acknowledgements

This work was supported in part by the ARO (ARO-W911NF-16-1-0121) and the NSF (IIS1252835, IIS-1562364). We thank Kevin Clark, Jungo Kasai, and Tim Dettmers as well as the anonymous reviewers for their feedback on this document, and the members of the UW NLP group for helpful conversations and comments on this work.

References

Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. 2018. Don't Just Assume; Look and Answer: Overcoming Priors for Visual Question Answering. In *CVPR*.

Ankesh Anand, Eugene Belilovsky, Kyle Kastner, Hugo Larochelle, and Aaron Courville. 2018. Blindfold Baselines for Embodied QA. In *NIPS 2018 Visually-Grounded Interaction and Language (ViGilL) Workshop*.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant Risk Minimization. *Computing Research Repository*, arXiv:1907.02893. Version 3.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.

Hyojin Bahng, Sanghyuk Chun, Sangdoo Yun, Jaegul Choo, and Seong Joon Oh. 2020. Learning Debiased Representations with Biased Representations. In *ICML*.

Yonatan Belinkov, Adam Poliak, Stuart M Shieber, Benjamin Van Durme, and Alexander M Rush. 2019. On Adversarial Removal of Hypothesis-only Bias in Natural Language Inference. In *StarSem*.

Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. 2016. Domain Separation Networks. In *NeurIPS*.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A Large Annotated Corpus for Learning Natural Language Inference. In *EMNLP*.

Wieland Brendel and Matthias Bethge. 2019. Approximating CNNs with Bag-of-local-Features Models Works Surprisingly Well on ImageNet. In *ICLR*.

Remi Cadene, Corentin Dancette, Matthieu Cord, and Devi Parikh. 2019. RUBi: Reducing Unimodal Biases for Visual Question Answering. In *NeurIPS*.

Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. 2019. Domain Generalization by Solving Jigsaw Puzzles. In *CVPR*.

Christopher Clark, Mark Yatskar, and Luke Zettlemoyer. 2019. Don't Take the Easy Way Out: Ensemble Based Methods for Avoiding Known Dataset Biases. In *EMNLP*.

Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An Overview of Bilevel Optimization. In *Annals of operations research*.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A Large-Scale Hierarchical Image Database. In *CVPR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *ACL*.

Zhengming Ding and Yun Fu. 2017. Deep Domain Generalization with Structured Low-Rank Constraint. In *IEEE Transactions on Image Processing*.

Christiane Fellbaum. 2012. WordNet. *The encyclopedia of applied linguistics*.

Shi Feng, Eric Wallace, and Jordan Boyd-Graber. 2019. Misleading Failures of Partial-input Baselines. In *ACL*.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. In *JMLR*.

Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. 2015. Domain Generalization for Object Recognition with Multi-Task Autoencoders. In *ICCV*.

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. 2016. On Differentiating Parameterized Argmin and Argmax Problems with Application to Bi-level Optimization. *arXiv preprint arXiv:1607.05447*. Version 1.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2018. Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *IJCV*.

Gabriel Grand and Yonatan Belinkov. 2019. Adversarial Regularization for Visual Question Answering: Strengths, Shortcomings, and Side Effects. In *Proceedings of the Second Workshop on Shortcomings in Vision and Language*.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. 2018. Annotation Artifacts in Natural Language Inference Data. In *NAACL*.

He He, Sheng Zha, and Haohan Wang. 2019. Unlearn Dataset Bias in Natural Language Inference by Fitting the Residual. In *Deep Learning Approaches for Low-Resource NLP*. ACL.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

Lisa Anne Hendricks, Kaylee Burns, Kate Saenko, Trevor Darrell, and Anna Rohrbach. 2018. Women also Snowboard: Overcoming Bias in Captioning Models. In *ECCV*.

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. 2019. Natural Adversarial Examples. *Computing Research Repository*, arXiv:1907.07174. Version 3.

Allan Jabri, Armand Joulin, and Laurens Van Der Maaten. 2016. Revisiting Visual Question Answering Baselines. In *ECCV*.

Robin Jia and Percy Liang. 2017. Adversarial Examples for Evaluating Reading Comprehension Systems. In *EMNLP*.

Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. 2019. Learning not to Learn: Training Deep Neural Networks with Biased Data. In *CVPR*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *ICLR*.

Ronan Le Bras, Swabha Swayamdipta, Chandra Bhagavatula, Rowan Zellers, Matthew E Peters, Ashish Sabharwal, and Yejin Choi. 2020. Adversarial Filters of Dataset Biases. In *ICML*.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*.

Mike Lewis and Angela Fan. 201. Generative Question Answering: Learning to Answer the Whole Question. In *ICLR*.

Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C Kot. 2018a. Domain Generalization with Adversarial Feature Learning. In *CVPR*.

Ya Li, Xinmei Tian, Mingming Gong, Yajing Liu, Tongliang Liu, Kun Zhang, and Dacheng Tao. 2018b. Deep Domain Generalization via Conditional Invariant Adversarial Networks. In *ECCV*.

Yi Li and Nuno Vasconcelos. 2019. Repair: Removing Representation Bias by Dataset Resampling. In *CVPR*.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference. In *ACL*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *LREC*.

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. Compositional Questions Do Not Necessitate Multi-hop Reasoning. In *ACL*.

Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. 2013. Domain Generalization via Invariant Feature Representation. In *ICML*.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A Decomposable Attention Model for Natural Language Inference. In *EMNLP*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. BiDirectional Attention Flow for Machine Comprehension. In *ICLR*.

Krishna Kumar Singh, Dhruv Mahajan, Kristen Grauman, Yong Jae Lee, Matt Feiszli, and Deepti Ghadiyaram. 2020. Don't Judge an Object by Its Context: Learning to Overcome Contextual Bias. In *CVPR*.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *EMNLP*.

Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. 2019. Learning Robust Global Representations by Penalizing Local Predictive Power. In *NIPS*.

Zheng Xu, Wen Li, Li Niu, and Dong Xu. 2014. Exploiting Low-Rank Structure from Latent Domains for Domain Generalization. In *ECCV*.

Yadollah Yaghoobzadeh, Remi Tachet, TJ Hazen, and Alessandro Sordoni. 2019. Robust Natural Language Inference Models with Example Forgetting. *Computing Research Repository*, arXiv:1911.03861. Version 1.

Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating Unwanted Biases with Adversarial Learning. In *AAAI/ACM Conference on AI, Ethics, and Society*.

Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2016. Yin and Yang: Balancing and Answering Binary Visual Questions. In *CVPR*.

A ImageNet Animals

Here, we describe how ImageNet Animals was built in more details. We built super-classes by taking advantage of the hierarchical structure of ImageNet (Deng et al., 2009) and its correspondence to WordNet (Fellbaum, 2012). We selected 6 wordnet synsets that have at least 20 hyponyms that exist in ImageNet: fish.n.01, insect.n.01, dog.n.01, bird.n.01, ungulate.n.01, and snake.n.01. Each synset is used as a class. We gather examples for each class by sampling images randomly from all its hyponyms in the ImageNet training data. Examples of what kinds of images were considered OOD are shown in Figure 1.

B Training Details

In this section, we give additional details about the models used, and specify the learning rates, batch sizes, and other optimization details. All models were trained with 1 to 4 GeForce RTX 2080 Ti GPUs.

B.1 Synthetic MNIST

We trained all models using a learning rate of 0.01, batch size of 1000, and momentum of 0.9. Models were trained until loss on the training data stopped decreasing, and were trained on a single GPU. For regularization, dropout was applied before the last layer of both the higher capacity and lower capacity model with a rate of 0.5.

B.2 VQA

Models were trained using Adam (Kingma and Ba, 2014) and a linearly decaying learning rate, following the default settings for LXMERT (Tan and Bansal, 2019)[1], except that we use a batch size of 512 instead of 32, and train for 3 epochs instead of 4. We found these changes to the optimization procedure increased performance by a significant margin; on the standard VQA 2.0 dataset our implementation achieves a dev set accuracy of 73.4 with them and 70.1 without them (compared to 69.9 reported by Tan and Bansal (2019)). Models were trained with 4 GPUs, and were trained in floating-point 16 mode using Apex.[2]

We also upsample examples for each answer cluster to ensure each cluster consists about 5% of the training data, while putting proportional weights on the classes so that the weighted answer distribution matches the original training data. This ensures each mini-batch can contain a reasonable number of examples for each class.

B.2.1 Lower Capacity Model

In more detail, the lower capacity model takes the intermediate question-only and image-only representations from the higher capacity model, concatenates these vectors and their elementwise product, and passes the result to a 2048 ReLU layer, followed by a linear predictor layer. This model reaches 67% on the VQA 2.0 dev set when trained alone.

B.3 MNLI

We use the default BERT optimizer (Devlin et al., 2019), which is known to be effective for MNLI, but with a batch size of 256. We found increasing the batch size did not change performance on the matched MNLI dev set. Since the lower capacity model is not pretrained, we use a learning rate of 1e-3 without linear decay instead of 5e-5 for its parameters. Models were also trained with 4 GPUs, and were trained in floating-point 16 mode using Apex.

B.3.1 Lower Capacity Model

Here, we specify the lower capacity model in more detail. The model is a simplified version of the model from Parikh et al. (2016) and has the following stages:

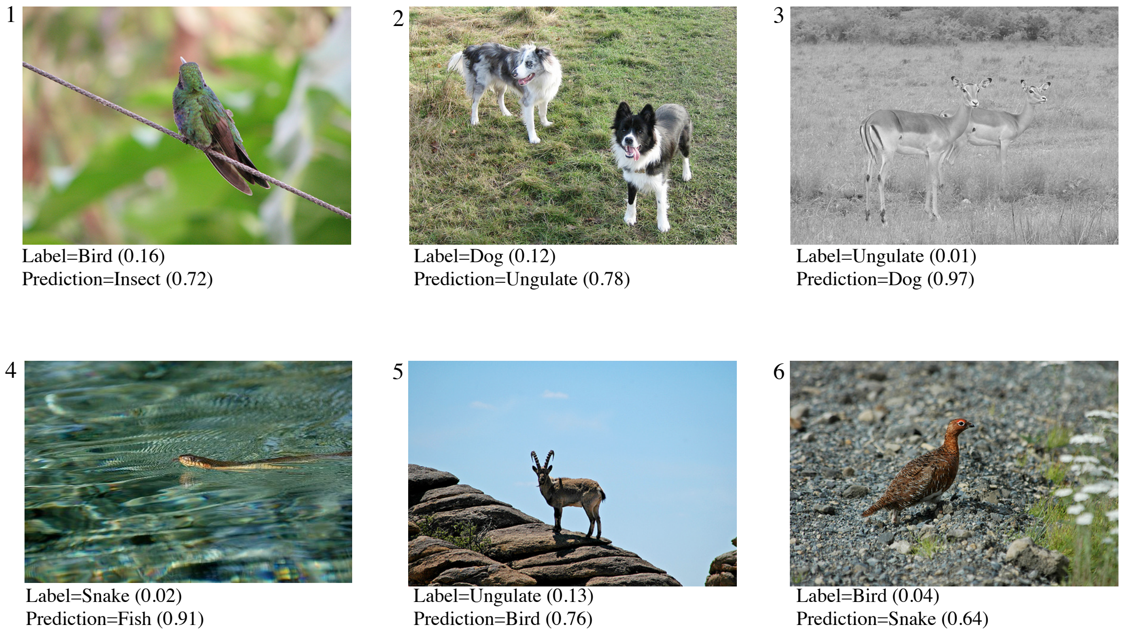
Embed: Embed the words using a character CNN, following what was done by Seo et al. (2017), and the fasttext crawl word embeddings (Mikolov et al., 2018), then apply a 200 dimensional ReLU layer.

Co-Attention: Compute an attention matrix using the formulation from Seo et al. (2017), and use it to compute a context vector for each premise word (Bahdanau et al., 2015). Then build an augmented vector for each premise word by concatenating the word's embedding, the context vector, and the elementwise product of the two. Augmented vectors for the hypothesis are built the same way using the transpose of the attention matrix.

Pool: Feed the augmented vectors into another 200 dimensional ReLU layer, and max-pool the results. The max-pooled vectors from the premise and hypothesis are concatenated and fed into a softmax layer with three outputs to compute class probabilities.

[1] https://github.com/airsplay/lxmert

[2] https://github.com/NVIDIA/apex



1 Label=Bird (0.16) Prediction=Insect (0.72)

2 Label=Dog (0.12) Prediction=Ungulate (0.78)

3 Label=Ungulate (0.01) Prediction=Dog (0.97)

4 Label=Snake (0.02) Prediction=Fish (0.91)

5 Label=Ungulate (0.13) Prediction=Bird (0.76)

6 Label=Bird (0.04) Prediction=Snake (0.64)

Figure 1: Qualitative examples from ImageNet Animals where most of the image-patch classifiers were incorrect. We show images paired with the gold label and the most common prediction made by the image-patch classifiers, with the percent of image-patch classifiers that predicted those labels in parentheses. Errors are often caused by the patch classifiers associating features of the background with the class. In particular, because (1) twigs and leaves are associated with insects, (2) grassy fields are associated with ungulates (e.i., hooved mammals), (3) black and white photos are associated with dogs due to the commonality of black-and-white dog photos in the training data, (4) water is associated with fish, (5) open sky is associated with birds, and (6) close-ups of the ground are associated with snakes.

We apply dropout at a rate of 0.1 before all fully connected layers.

B.4 ImageNet Animals

Models are trained with a learning rate of 0.02, a momentum of 0.9 and batch size of 512 for 76 epochs. The learning rate is decreased to 0.006 for the last 10 epochs. Models were trained with 4 GPUs.

C Pretrained Bias Upper Bounds

In this section, we describe the Pretrained Bias approach in more detail. The general method is to train a bias-only model that captures the target bias, and then use it to train a de-biased model using either the Bias Product or Entropy +H method from Clark et al. (2019).

C.1 Synthetic MNIST

We train a bias-only model by training a model to predict the label using a one-hot encoding of which modification was applied to the image. This model is then used with the Bias Product method, since that method is expected to be the best choice when the bias-only model perfectly captures a conditionally independent bias.

C.2 VQA

We follow Clark et al. (2019) by training a bias-only model that predicts the answer using a one-hot encoding of the question-type, and using the Ensemble +H method. The entropy penalty was set to 0.24 using the OOD dev set.

C.3 MNLI Hard

We use the hypothesis-only model that was used to construct the dataset splits as a bias-only model, and apply the Ensemble +H method. The value of the entropy penalty was selected to be 0.1 using the OOD dev set.

C.4 ImageNet Animals

We build a bias-only model by computing the expected class prediction across all image-patch classifiers. Since it is possible for this prediction to be zero for some classes, we additionally smooth the distribution by adding $\log(1 + e^{\alpha})$ to the probabilities and re-normalizing, where α is a learned smoothing parameter. This model is then used with the Entropy +H method, the value of the entropy penalty was again set to 0.1 based on the OOD dev set.

D Results with Optimized Hyperparameters

In this section, we show results when using optimized hyperparameters for MCE, i.e., when w is tuned.

D.1 Synthetic MNIST

Table 1 shows the results. On these datasets increasing w can sometimes lead to small performance improvements for MCE and its ablations.

D.2 VQA

Table 2 shows the results. For VQA $w = 0.2$ is the optimal setting for MCE, although decreasing w leads to performance gains for the ablated versions.

D.3 MNLI

Table 3 shows the results. For MNLI $w = 0.2$ is consistently the best setting, so the results are unchanged.

D.4 ImageNet Animals

Table 4 shows the results. Here decreasing w improved performance of MCE, but maintaining w at 0.2 was best for the No CI method.

Method	Background			Patch			Split		
	H	OOD Acc	ID Acc	H	OOD Acc	ID Acc	H	OOD Acc	ID Acc
MCE (Ours)*	0.5	82.14	93.99	0.8	75.22	87.57	1.0	93.06	94.01
No CI*	0.15	78.83	94.92	1.0	70.25	82.19	0.05	91.47	93.86
No BP*	1.0	80.13	93.93	0.2	71.93	86.67	1.0	92.52	93.89
With Adversary*	0.05/0.08	80.57	93.37	0.7/0.01	70.31	82.78	0.02/0.01	91.21	93.56

Table 1: Results on Synthetic MNIST with hyperparameter tuning. The hyperparameter(s) used (either w for MCE or w and the adversary weight for With Adversary) are shown in the H columns.

Method	H	OOD Acc		ID Acc	
		Acc	w/o AP	Acc	w/o AP
MCE (Ours)*	0.2	68.44	66.10	74.03	72.72
No CI*	0.1	65.40	55.77	74.38	66.01
No BP*	0.1	63.65	52.43	70.76	60.82
With Adversary*	0.2/0.007	66.08	26.16	72.17	37.47

Table 2: Results on VQA-CP with hyperparameter tuning. The hyperparameters used are shown in the H column.

Method	H	Hard	HANS	ID
MCE (Ours)*	0.2	77.58	64.43	83.28
No CI*	0.2	77.24	64.25	83.38
No BP*	0.2	76.44	62.18	83.88
With Adversary*	0.3/0.16	77.10	63.03	83.35

Table 3: Results on MNLI with hyperparameter tuning.

Method	H	0.05	0.1	0.2	All
MCE (Ours)*	0.1	80.47	81.85	83.92	90.09
No CI*	0.2	81.02	82.46	84.51	89.97
With Adversary*	0.3/0.005	80.33	81.77	83.79	89.92

Table 4: Results on ImageNet Animals with hyperparameter tuning.