An Empirical Analysis of Backward Compatibility in Machine Learning Systems

Megha Srivastava
Microsoft Research

Besmira Nushi
Microsoft Research

Ece Kamar
Microsoft Research

Shital Shah
Microsoft Research

Eric Horvitz
Microsoft Research

ABSTRACT

In many applications of machine learning (ML), updates are performed with the goal of enhancing model performance. However, current practices for updating models rely solely on isolated, aggregate performance analyses, overlooking important dependencies, expectations, and needs in real-world deployments. We consider how updates, intended to improve ML models, can introduce new errors that can significantly affect downstream systems and users. For example, updates in models used in cloud-based classification services, such as image recognition, can cause unexpected erroneous behavior in systems that make calls to the services. Prior work has shown the importance of "backward compatibility" for maintaining human trust. We study challenges with backward compatibility across different ML architectures and datasets, focusing on common settings including data shifts with structured noise and ML employed in inferential pipelines. Our results show that (i) compatibility issues arise even without data shift due to optimization stochasticity, (ii) training on large-scale noisy datasets often results in significant decreases in backward compatibility even when model accuracy increases, and (iii) distributions of incompatible points align with noise bias, motivating the need for compatibility aware de-noising and robustness methods.

ACM Reference Format:
Megha Srivastava, Besmira Nushi, Ece Kamar, Shital Shah, and Eric Horvitz. 2020. An Empirical Analysis of Backward Compatibility in Machine Learning Systems. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403379

1 INTRODUCTION

Enthusiasm around applying machine learning (ML) methods in high-stakes domains such as healthcare and transportation is balanced by concerns about their reliability. Prior works on ML reliability and robustness have sought to develop techniques that enable models to perform successfully under the presence of data shifts or adversarial perturbations [6, 22, 27, 40]. However, these studies investigate the reliability of models in isolation, quantified only by aggregate performance metrics. In this work, we focus on challenges that arise when models are employed within larger systems, including (i) pipelines composed of multiple components, and in (ii) human-machine interaction. In both cases, dependencies and expectations about model behavior—and the influences of changes on the performance of the larger systems they compose—must be considered during model updates. For example, *when we "improve" the overall performance of an existing model, are there any hidden costs to the gain in accuracy? What new errors and failures are introduced that did not exist previously, thus decreasing reliability?*

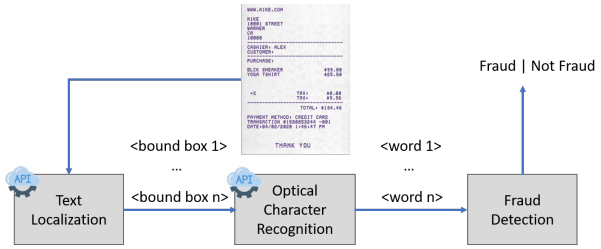


Figure 1: Example of a machine learning pipeline.

We show how practitioners can leverage the notion of *backward compatibility* to answer these questions. Backward compatibility in ML systems was first introduced by Bansal et. al [3] to describe the phenomenon of partial model regress in the context of human-AI collaboration. They observe that updated models that are more accurate may still break human expectations and trust when introducing errors that were not present in earlier versions of the model. However, considerations about backward compatibility are not limited to human-AI collaboration settings; new, unexpected errors induced by updates aimed at refining models pose problems with reliability when the models are used in larger, integrative AI systems composed of multiple models or other computing components that work in coordination.

Furthermore, the phenomenon of how and when costly backward incompatibility occurs in a learning context is not very well understood. To the best of our knowledge, no work to date has empirically investigated the causes of compatibility failures in machine learning, and the extent of this issue across datasets and models. We seek to investigate backward compatibility through empirical studies and demonstrate how designers of ML systems can take backward compatibility into consideration to build and maintain more reliable systems.

As a motivating example (Figure 1), consider the financial department of a company that processes expense reports to detect receipt fraud. The department may rely on an off-the-shelf optical character recognition (OCR) model to process receipts [21]. Over

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403379

time, the team may develop and encode a blacklist of common spoofed company names found to be fraudulent, such as "G00gle llc." or "Nlke". In this case, the financial services team builds the heuristic blacklist component with expectations on the OCR's ability to detect these names. Moreover, the heuristic rules have been optimized to perform well in accordance with the original OCR model used in development.

Meanwhile, in pursuing improvement in the OCR's overall performance, engineers may update the training data with a larger and perhaps noisier dataset from a variety of handwritten text sources. Engineers may celebrate increases in model accuracy and generalizability. However, if the update decreases the OCR performance on specific characters present in the blacklisted words, the financial department may experience costly failures when fraudulent receipts now go undetected, despite believing that they are using a better model. For example, if the newly added dataset contains biased noise and frequently mislabels "0" digits as "o" characters, the word "G00gle llc." may be wrongly recognized as "Google llc.", which is not in the blacklist. Similar scenarios in high-stakes settings such as healthcare can lead to even more serious consequences.

Motivated by potential downstream errors such as the one above, we emphasize that machine learning practitioners should consider potential backward compatibility issues before they decide to update models. Such practice is similar to common practices that software engineers follow when modifying traditional software [4]. More specifically, we make the following contributions:

(1) We expand the empirical understanding of when and how backward compatibility issues arise in machine learning and the relationship to example forgetting during retraining.
(2) We characterize backward compatibility under different noise settings using a variety of datasets (tabular, vision, language) and model architectures (linear, CNN, ResNet, BERT).
(3) We illustrate and discuss backward compatibility from the perspective of maintaining and monitoring the performance of modularized ML pipelines.
(4) We highlight how ML practitioners can use the presented results and methodology to create best practices and tools for updating and diagnosing learning models.

The rest of the paper is organized as follows: Section 2 positions the paper in the context of related work. Section 3 defines the setting of model updates and the backward compatibility metrics we use to characterize the phenomenon. Section 4 details the experimental setup. Section 5 studies the effect of optimization stochasticity on backward compatibility as a baseline, before studying backward compatibility in the presence of noisy model updates in Section 6. Finally, Section 7 shows how backward compatibility analyses can help to identify failures in ML pipelines.

2 BACKGROUND AND RELATED WORK

Backward compatibility in Machine Learning. Bansal et. al [3] introduced backward compatibility in the context of preserving human expectations and trust, and proposed a loss function to penalize newly introduced errors, which we further study in Section 6.3. Our work expands the understanding of backward compatibility in ML, and we add a new perspective by highlighting backward compatibility challenges that arise when inferences from a retrained model are used by other components of a larger system.

The challenge of understanding how changes in one component propagate through an entire system has attracted recent interest. [2, 32, 36] discuss how hidden data and feature dependencies can cause component entanglement, unexpected system behavior, and downstream performance regression [2, 32]. Such issues have also been highlighted in surveys on software engineering challenges in ML [1, 42]. New performance evaluation [7, 31, 41] approaches suggest reporting performance on data slices rather than aggregate measures that overlook partial model regression. Our work advises that measures of backward compatibility should be included when monitoring model performance during updates.
Catastrophic forgetting and transfer learning. Backward compatibility is relevant to the phenomenon of catastrophic forgetting [12, 28], which refers to situations where a learning model forgets old tasks while being trained in a sequential fashion to solve new tasks. Literature on catastrophic forgetting [13, 18] highlights "learning interference" as a cause for forgetting in models with a fixed capacity. In contrast, we investigate fluctuations that do not involve changes in the task or concept definition. Despite this simplification, we find retrained models are still subject to forgetting individual or similar examples. [38] present settings where the task definition has not changed, and focus on understanding cases of *example forgetting*. In these events, individual examples may be forgotten (multiple times) by the model during gradient-based optimization. The authors find that the most "forgotten" examples are often the most difficult, such as atypical inputs and noisy labels. We connect these results with our findings in Section 5, where we show how example forgetting events relate to model incompatibility.
Data cleaning and distributional shifts. Data cleaning is a topic of long-term interest in the data management and mining community [8, 33, 35]. Techniques addressing problems like outlier detection and data denoising are often agnostic to machine learning models, as the cleaning process is not guided by the influence on a particular model. Although model-agnostic data cleaning is important for cleaning generic data repositories consumed by multiple and sometimes unknown models, it is important to map changes in data characteristics to model performance to fully understand the impact of data quality when updating an already deployed model. While we do not propose a new technique for data cleaning, our findings can pave the way for future techniques that preserve the performance of systems in the presence of data shifts and noise.

Other studies have outlined the influence of class label noise [11, 19] and feature noise [14] on model behavior, informing a parallel line of work on increasing model robustness to noise [16, 25, 30, 37]. While these works show that modern neural networks can be somewhat robust to both label and feature noise (in non-adversarial settings), their evaluation heavily relies on aggregate performance metrics and does not investigate the impact of biased noise that affects only certain data clusters, as often occurs in the real world.

3 PROBLEM SET-UP

In both traditional software and machine learning systems, backward compatibility is a reliability concern emerging from *model updates*. There are many possible incentives for updating a model,

including re-training on a larger, more diverse, and perhaps noisier dataset to increase the model's generalizability and overall accuracy. In this section, we describe this specific setting of "noisy model updates," and then define two measures of backward compatibility.

3.1 Noisy Model Updates

Consider a machine learning team that has deployed the first version of a model, which was carefully trained on a small and clean dataset. While the small data quantity enabled the team to verify all labels and ensure good data quality, the team may now wish to improve model performance on a greater variety of inputs by using a larger and more diverse dataset. Due to high financial cost and time, the team may resort to noisier approaches to create this large dataset, such as crowdsourcing labels [23] or using weak supervision [34], a popular alternative due to its automation opportunities. Some examples include scraping web data [29], leveraging social media tags as labels, or collecting data via image search.

We refer to this practice as a *Noisy Model Update*, which has been performed across a variety of tasks, including language and vision domains, and has been shown to improve overall accuracy [5, 34]. However, the additional data may also contain biased noise, concentrated in particular regions or classes of the dataset. Imagine a team is collecting data from social media to enrich the dataset. In social media, some keywords may be ambiguous and contain examples from two or more classes at the same time. For example, the keyword "book" on Instagram shows results for books, inspirational quotes, and travel destinations. Model updates with data including these biased concentrations of noise may harm backward compatibility, necessitating analysis of the ways the newer model may be unreliable despite its increased accuracy. In Section 6, we analyze backward compatibility with respect to three different types of noise: label noise, feature noise, and outlier noise.

We are mainly interested in cases when noise does not affect the whole dataset uniformly because uniform noise distribution is more likely to impact the overall accuracy, reducing the incentive for a model update. Nevertheless, as we show in Section 7, even for uniform noise, some classes are more affected than others as they may be less distinguishable.

Problem Definition. More formally, let h_1 = Model 1, D_1 = Training Dataset 1, h_2 = Model 2, and D_2 = Training Dataset 2, where h_1 is trained on D_1, and h_2 is trained on D_2. Each model h is a function $h : x \;!\; y$, where x is an input instance and y is the ground truth label for the input. Backward compatibility of h_2 with regards to h_1 is evaluated on a separate held-out test set D_{test}. One can view D_{test} as a clean dataset that the team uses for performance evaluation in general. It is typically chosen such that its distribution resembles the expected real-world distribution, but in practice the resemblance varies by a large margin. Further, our problem definition makes the following assumptions to control for other factors that may affect backward compatibility: (1) h_1 and h_2 have the same model architecture; (2) the set of possible labels in D_1 and D_2 are the same; (3) the classification accuracy of h_2 on D_{test} is higher than the accuracy of h_1 on D_{test}, motivating the model update.

In our experiments, we first simplify the problem to set an upper-bound baseline on backward compatibility by keeping $D_1 = D_2$ (Section 5). In this case, any discrepancies between h_1 and h_2 are due to optimization stochasticity. Afterwards, we proceed with results for the noisy update problem in Section 6, where $D_1 \quad D_2$.

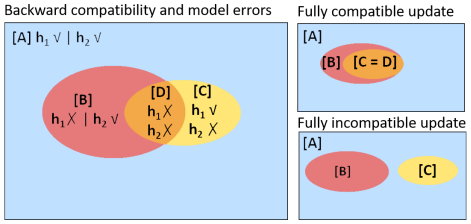


Figure 2: Backward compatibility as described by model errors. In fully compatible updates, the updated model (h_2) only refines errors of the previous model (h_1). In fully incompatible updates, the errors of the two models are disjoint. BTC = 1 $\frac{A,B}{A,C}$, BEC = 1 $\frac{C}{D,C}$.

3.2 Measuring Backward Compatibility

We define two measures of backward compatibility, which we use in our experiments (See Figure 2). The first is the **Backward Trust Compatibility (BTC) score**, which is the ratio of points in a held-out test set (e.g., D_{test}) that h_2 predicted correctly among all points h_1 had already predicted correctly.

$$\text{BTC} = \frac{\sum_{i=1}^{|D|} \mathbb{1}[h_1(x_i) = y_i, h_2(x_i) = y_i]}{\sum_{i=1}^{|D|} \mathbb{1}[h_1(x_i) = y_i]} \tag{1}$$

BTC matches the backward compatibility score in [3] and semantically describes the percentage of trust that is preserved after the update. However, a challenge with measuring only BTC is that if the updated model h_2 achieves almost-perfect accuracy, such as when applying deep neural networks to extremely low sample-complexity problems (Section 6), BTC may be high even if most errors caused by h_2 are new and unexpected. We thus consider a second metric, the **Backward Error Compatibility (BEC)** score, which is the proportion of points in a held-out test set that h_2 predicted incorrectly, out of which h_1 also predicted incorrectly, thus capturing the probability that a mistake made by h_2 is not new.

$$\text{BEC} = \frac{\sum_{i=1}^{|D|} \mathbb{1}[h_1(x_i) < y_i, h_2(x_i) < y_i]}{\sum_{i=1}^{|D|} \mathbb{1}[h_2(x_i) < y_i]} \tag{2}$$

These unexpected errors may negatively impact downstream components in a pipeline that may have learned to suppress the initial errors of h_1 or are mitigating them via traditional error handling and heuristics. We propose that developers of ML models should measure both **BTC** and **BEC** when considering a model update, rather than solely improvement in accuracy.

4 EXPERIMENTAL METHODS

Our empirical evaluations cover several tasks in increasing order of dataset and model complexity. As an upper-bound baseline on backward compatibility, we first focus on settings without a model update (where data sets are identical through retraining, $D_1 = D_2$). To show that backward compatibility issues exist even in simple

models, we apply a logistic regression model on a FICO binary-classification task (Section 5). We then investigate the effect of different types of noise on (1) the MNIST digit classification task with a 3-layer CNN, (2) the more complex CIFAR-10 image recognition task using a ResNet-18 model, and then (3) fine tuning the state-of-the-art BERT language model for an IMDB sentiment analysis task (Section 6). Finally, we analyze downstream errors in a ML pipeline where one component applies a 3-layer CNN on the Chars74K Character Recognition task (Section 7).

Experiment Design. For all experiments, we train two models h_1 and h_2 on datasets D_1 and D_2 respectively, where $D_1 = D_2$ for the optimization stochasticity baselines and $D_1 \quad D_2$ for experiments involving model updates. We evaluate the accuracy of h_1 and h_2 on the same held-out test set, on which we also calculate the BTC and BEC scores as h_2's backward compatibility with respect to h_1. For experiments investigating the effect of noise, such as those in Figure 4, we calculate the *gain in test accuracy* from h_2, and the BTC and BEC scores over varying noise amounts, noting that in practice a developer likely uses a dataset with an unknown fixed amount of noise. Finally, we analyze the set of *incompatible points* - points that h_1 predicted correctly yet h_2 missed - to gain insight on the types of points the new model h_2 is unreliable on.

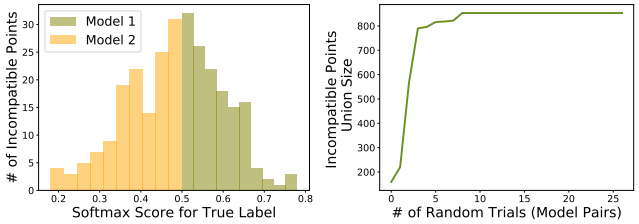
Model and Dataset Details. For all datasets, we aimed to use a close to state-of-the-art models that has the most accessible implementation, as our target audience is ML practitioners. Below, we list all model and dataset details, in order of their appearance:

(1) **FICO Credit Score Risk Classification** [10]: *# Classes*: 2, *Dataset Sizes*: $|D_1| = |D_2| = 6000$, $|D_{test}| = 1973$, *Model*: Logisitic Regression, *Learning Rate*: $1e \quad 4$, *Train Epochs*: 100
(2) **MNIST Digit Classification** [24]: *# Classes*: 10, *Dataset Sizes*: $|D_1| = 4800, |D_2| = 48000$, $|D_{test}| = 12,000$, *Model*: 3-Layer CNN, *Learning Rate*: $1e \quad 2$, *Train Epochs*: 50
(3) **CIFAR-10 Object Recognition** [20]: *# Classes*: 10, *Dataset Sizes*: $|D_1| = 10000, |D_2| = 50000$, $|D_{test}| = 10000$, *Model*: ResNet-18, *Learning Rate*: 0.1, *Train Epochs*: 35
(4) **IMDB Movie Review Sentiment Analysis** [26]: *# Classes*: 2, *Dataset Sizes*: $|D_1| = 400, |D_2| = 1600$, $|D_{test}| = 400$, *Model*: Finetuned BERT-Base Language Model, *Learning Rate*: $4e \quad 5$, *Train Epochs*: 2
(5) **Chars74K Char. Recognition (OCR Pipeline)** [9]: *# Classes*: 62, *Dataset Size*: $|D_1| = 4960, |D_2| = 24800$, $|D_{test}| = 6200$, *Model*: 3-Layer CNN, *Learning Rate*: $1e \quad 4$, *Train Epochs*: 20

We train all models using stochastic gradient descent (SGD) and ensure all classes are equally represented in the training dataset.

5 BACKWARD COMPATIBILITY AND OPTIMIZATION STOCHASTICITY

As an upper-bound baseline on backward compatibility, we first seek to understand backward compatibility empirically when the *training data has not been updated*. This means that the two training datasets, D_1 and D_2, are identical, and any discrepancies between h_1 and h_2 are purely due to optimization stochasticity. Possible sources of optimization stochasticity include random model weight initialization or random data shuffling across training epochs. In this section specifically, model retraining entails simply training a new model with a different random seed for random initialization and data shuffling.



(a) (b)

Figure 3: (a) Distribution of incompatible examples over the output softmax score for two individual models. (b) Number of unique incompatible examples across pairs of models.

To better understand backward compatibility in this simple baseline condition, we ask the following questions:

(1) Do the incompatible points correspond to datapoints for which h_1 and h_2 have low confidence in their predictions?
(2) Is there a correlation between incompatible points and example forgetting, as studied by [38]?
(3) Are the incompatible points between any two models the same across different trials?

Since deep networks trained with high-dimensional datasets are more subject to optimization stochasticity due to the non-convex nature of the optimization function in such settings, we first study this question on a Logistic Regression model on the FICO credit score binary classification task [26]. In Section 6 and 7 we repeat the same experiments for larger models and data dimensionality and use these results as a baseline.

Across 25 trials (i.e., two logistic regression models), the average accuracy of both h_1 and h_2 (as there is no change in the training data) is **70.5 0.86 (%)**, with **BTC 94.5 1.31 (%)** and **BEC 78.3 2.15 (%)** backward compatibility scores. These results show that backward compatibility concerns may arise even when the training data remains identical and the model architecture is simple.

5.1 Confidence in predictions

We initially hypothesized that, when the training data does not change, backward incompatibility arises from examples with low confidence that are close to the decision boundary, which might be subject to slight changes in the boundary. Figure 3a shows a histogram of all points in D_{test} grouped by the softmax output of two individual models h_1 and h_2, and demonstrates a trend towards low confidence (softmax output = 0.5). However, there still exist points for which either h_1 or h_2 had high confidence, which is worrisome; a high prediction confidence by h_1 on certain points may cause a user or system to be especially reliant on that prediction.

5.2 Relationship with example forgetting

Next, we measure the relationship between incompatible points and example forgetting, as studied by Toneva et. al [38]. Because example forgetting measures the number of "forgotten events" by a model across epochs *during training*, we use a separate validation

Table 1: Relationship between incompatible examples and forgetting events on the validation set.

Data Type	# Forgetting Events (h_1 vs. h_2)		# Forgetting Events (h_2 vs. h_1)	
h_1 and h_2 both correct	.51	.12	.56	.16
h_1 and h_2 both incorrect	.83	.05	.91	.08
h_1 correct, h_2 incorrect	1.52	.42	1.77	.2

set for this particular experiment to avoid touching the test set during training. More precisely, for each example in the validation set, we count the number of epochs when the model made an incorrect prediction (one "forgetting event") for a point it had previously predicted correctly. We then aggregate the average number of forgetting events per example by three different regions of interest as shown in Table 1. This procedure is repeated for both models.

The observations suggest a strong correlation between the forgotten examples over the course of training and the incompatible examples between two models. Further experiments using deep neural networks demonstrate the same phenomena. However, while measuring example forgetting is costly and involves calculations at each training step, as noted in [38], detecting similar points using backward compatibility only requires one pass at the end of training. Moreover, since using forgettable data points helps improve model learning (also noted in [38]), our results suggest that simply considering backward compatibility when designing models can help identify points that would lead to similar performance boosts. Finally, there may also exist opportunities for combining both measurements to devise training algorithms that can ensure backward compatibility on-the-fly during training by guiding the model to remain compatible with earlier versions (i.e., checkpoints) of itself.

5.3 Consistency across multiple trials

Finally, we ask whether incompatible points are a property of the data itself, or unique to different pairs of models h_1 and h_2. Figure 3b shows how the size of the union of all sets of incompatible test examples grows as the number of random trials (i.e., different pairs of h_1 and h_2), increases. We observe a sharp increase between 0-5 pairs before a plateau, suggesting that, for this particular task, it is indeed possible that there exists a set of 800 datapoints (40.5% of the test set) in which all incompatible points belong. However, as indicated by the sharp increase between 0 and 5 trials, the incompatible points between two models may be completely disjoint and even ensembling techniques, increasing model capacity with less than five models, may not alleviate the problem.

Results summary: Significant incompatibilities arise even when the training data remains identical and within a simple underlying model architecture. Importantly, our results show that backward compatibility captures a notion of "example difficulty", and, per the simplicity of identifying incompatible points, can be a useful tool for understanding the types of data that a model will be unreliable for when it comes to working with other components or end users.

6 BACKWARD COMPATIBILITY UNDER NOISY MODEL UPDATES

We now study backward compatibility when additional data is added to training. We focus on noisy model updates where D_2 is a larger, noisier dataset than D_1. In all of the following experiments, h_2 is *retrained* on D_2 and initialized by using h_1's weights, as we found that doing so consistently improved both model accuracy and backward compatibility. Yet, our results still show that, despite strong overall accuracy gains, backward incompatibility still persists and is further exacerbated as dataset noise increases.

In the following experiments, we consider noise biased by class (MNIST digit and CIFAR10 image category) as well as noise on class-independent groups (IMDB movie review genre). We specifically consider three types of noise:

(1) *Label noise* - Instances from one class are labeled in error as another similar class (e.g., digit "0" as letter "o" in OCR).
(2) *Feature noise* - Features of the input instances themselves are noisy (e.g., occlusion for images, noisy sensors etc.).
(3) *Outlier noise* - Instances from a class not part of the task definition are wrongly labeled with a class that is part of the task definition (e.g., lion images may be added to the CIFAR dataset and be classified as cats).

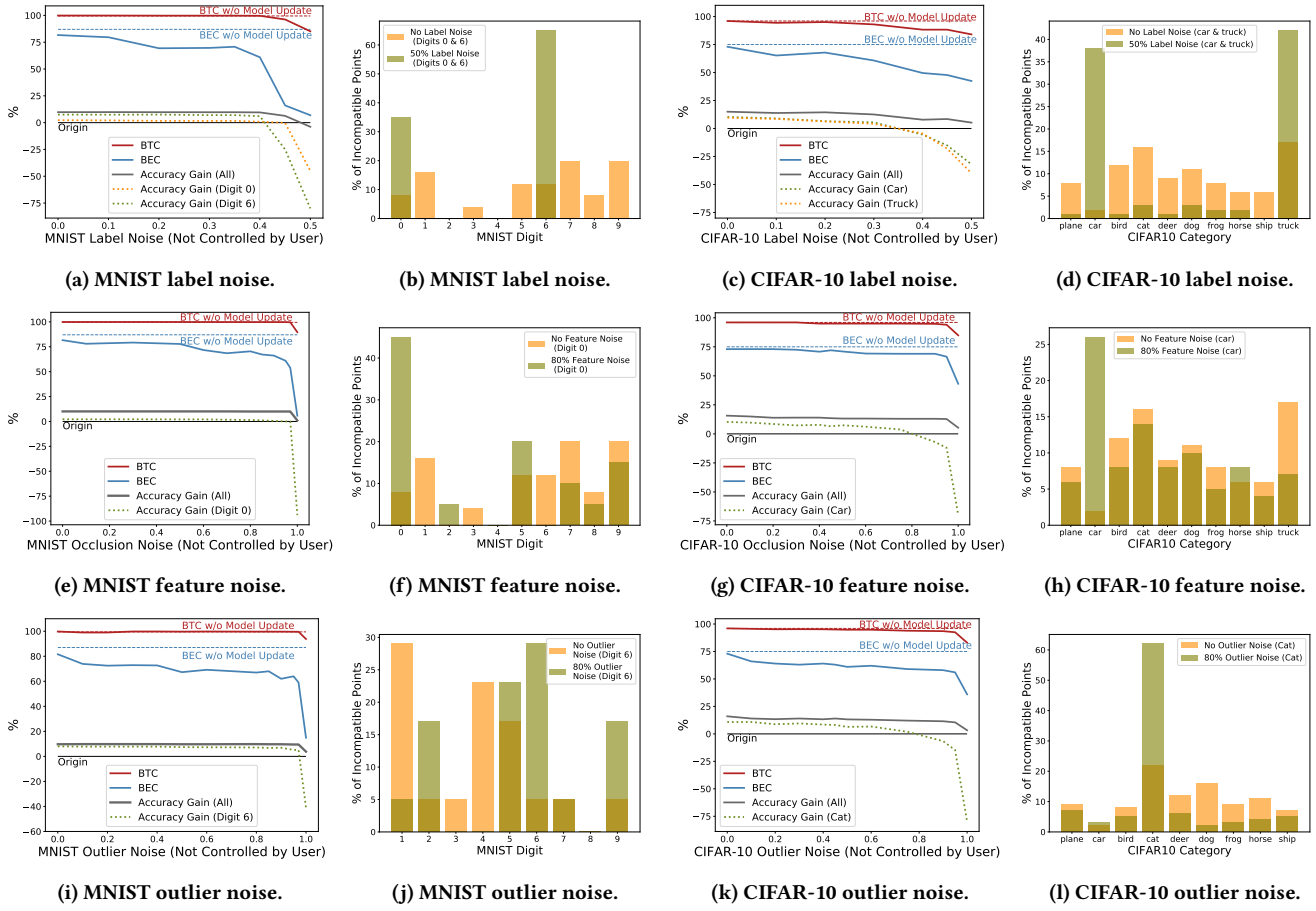
6.1 Class-level incompatibility

We first analyze backward compatibility in model updates where the noise is biased towards a particular set of target *classes*.

6.1.1 Label Noise. We simulate label noise, where instances from one class are labeled erroneously as a similar class, using the digit pair (0,6) and CIFAR-10 category pair ("car", "truck") as the target label pairs that are switched. Such label noise can occur through annotation noise, for example, where a human annotator via crowdsourcing may accidentally label an image of a truck with "car," or a painted digit 6 on a street address as "0" due to the similar shape. The degree of noise affects the likelihood of switching the labels of datapoints belonging to the target label pairs. In the no-noise condition on a heald-out test set, the three-layer CNN achieves 99.4% accuracy after 50 epochs for MNIST, while for CIFAR-10 the ResNet-18 model achieves 90.3% overall accuracy after 35 epochs.

Figures 4a and 4c demonstrate the effect of varying label noise on BTC (solid red), BEC (solid blue), gain in overall accuracy by h_2 over h_1 (solid gray), and gain in class-level accuracies by h_2 over h_1 (dashed green and orange). The dashed red and blue lines show BTC and BEC baselines when there is no model update, and $D_1 = D_2$. For both datasets, we observe a decrease in backward compatibility even when *the accuracy gain is positive*. For example, for 30% label noise, despite h_2 improving overall accuracy and class-level accuracies for the target labels, there is a notable decrease in BEC, as well as a minor decrease in BTC. Interestingly, we notice that the BTC measure decreases more for CIFAR-10 than MNIST, likely due to the extremely high accuracy of h_2 on MNIST. Most importantly, for higher noise values (i 0.4), the target class accuracy starts to drop even though the overall accuracy gain stays positive.

Further analysis of incompatible points (see histograms in Figures 4b and 4d) shows that, as label noise increases to 50%, the proportion of incompatible points within noise susceptible classes increases. These results demonstrate that backward compatibility analyses can help identify the types of data points that were subject to biased noise at the time of data collection.



(a) MNIST label noise. (b) MNIST label noise. (c) CIFAR-10 label noise. (d) CIFAR-10 label noise.

(e) MNIST feature noise. (f) MNIST feature noise. (g) CIFAR-10 feature noise. (h) CIFAR-10 feature noise.

(i) MNIST outlier noise. (j) MNIST outlier noise. (k) CIFAR-10 outlier noise. (l) CIFAR-10 outlier noise.

Figure 4: Backward compatibility with varying noise for MNIST and CIFAR-10.

6.1.2 Data Feature Noise. We next simulate feature noise, focusing on *occlusion*, where images are blocked by artifacts such as smudges and out-of-domain objects. We randomly distribute occlusions that occupy 20% of target images (Digit 0 in MNIST, "Car" in CIFAR-10). Real-world examples of occlusion include artifacts from the camera or current context (such as rainy weather) blocking a subject.

Interestingly, Figure 4e shows that, while for MNIST the accuracy gain on the target noise class (Digit 0) is stable up until high amounts of noise, BEC decreases early on. This may be due to the simplicity of the MNIST task and retraining with h_1's weights, which may have already learned the digit 0 well (explaining the minimal gains on Digit 0 by h_2). Adding noise may first decrease performance on specific points before hurting the entire class representation itself. However, for CIFAR-10, because h_2 strongly improves the "Car" class accuracy, small amounts of noise may harm performance only on parts of the data that h_1 anyways performed poorly on, resulting in a small backward compatibility decrease. Finally, Figures 4f and 4h show that at 80% noise, analysing the incompatible points can identify the target groups of the biased noise.

We note that adding feature noise sometimes *improves* performance by reducing overfitting [15], and is thus used for data augmentation. This may explain why backward compatibility significantly decreases only with a high degree of occlusion noise (i 80%).

6.1.3 Outlier Noise. To simulate outlier noise, we include instances in the training data that are not part of the initial task, but were labeled as one of the in-task classes. We convert both the MNIST and CIFAR-10 tasks to predictions over nine classes, treat Digit 0 and "Truck" as outliers, and Digit 6 and "Cat" as targets for the outlier noise. Semantically, this means that the training data will contain images of trucks (i.e., Cat® trucks by the Caterpillar series) that are labeled as "cat" because of language ambiguity.

While the main backward compatibility trends persist here as well, Figures 4j and 4l highlight that it is possible that noise in the targeted class may decrease compatibility in other classes as well. For instance, the number of incompatible points on MNIST increases on digits "2", "5", "6", "9" even though only digit "6" was directly affected by noise. Such behavior can be extenuated when the classification task contains more classes and the outlier examples look similar to many of the classes represented in the task.

Results summary: Adding noise biased towards a class is aligned with lack of backward compatibility for that class. However, the effects can propagate to increased incompatibility of other classes, especially if these classes become less distinguishable after the inclusion of outliers. Measuring backward compatibility helps detect this unreliability even when the overall accuracy gain by h_2 is high.

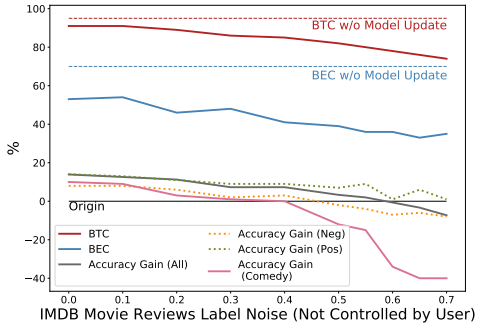


Figure 5: Backward compatibility analysis on an IMDB movie reviews sentiment classification task.

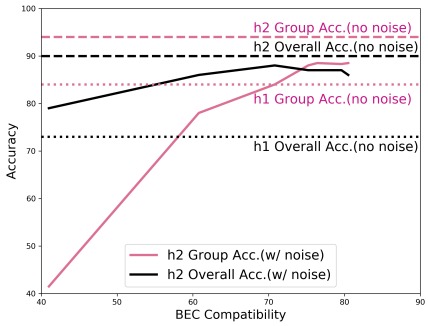


Figure 6: Regularization improves compatibility, but it may not fully prevent discrepancies from biased noise.

6.2 Beyond class groups: Sentiment Analysis

A natural question following the previous results is: *Can backward incompatibility be identified by only monitoring the accuracy of individual classes?* Here we argue that this approach, although complementary, is unsatisfying because:

(1) In the previous experiments, there still existed a range where h_2 *increased accuracy* for a class yet backward compatibility *decreased*, likely due to performance decrease for a *sub-class*.
(2) A possible grouping of datapoints that do not share the same class label may suffer from backward compatibility which a per-class analysis would hide.

In essence, measuring backward compatibility accounts for all types of points for which h_2 performance decreases, without needing to define and analyze a priori which points to examine.

To demonstrate this, we analyze backward compatibility in a movie review sentiment analysis task considering data noise biased on groups beyond labels - specifically, all movies belonging to a specific *genre*. Consider a large dataset of movie reviews created by scraping reviews from different web pages, including genre-specific web pages such as a comedy review site or sub forum. The labels may be scraped from star ratings accompanying the reviews. If the comedy site changes its html structure (which happens frequently in websites), the scraper may fail to map the review to the right rating and therefore induce biased noise for the comedy genre.

We simulate this setting using the IMDB Movie Reviews dataset [26]. We filter all reviews in the training data with the keyword "comedy," and then flip the label with varying noise. Figure 5 summarizes the results, including the comedy group accuracy gain (solid pink). The test performance drop for comedy reviews is far more severe than the performance drop for either negative or positive classes. Furthermore, for noise between 0.4 and 0.47, there exists a region where *class-based accuracy improves* on both classes but *sub-class comedy group accuracy decreases*. An analysis of the incompatible points at 0.47 noise, of which 59.5% are comedy (vs. 20% at 0 noise), would help detect this failure while class-based accuracy metrics would not.

Results summary: When backward incompatibility affects groups of data but not a whole class, monitoring class accuracy is insufficient. Moreover, since there may exist many possible groups, and it is not possible to know a priori which groups suffer from noise, the decrease in BEC and BTC scores and analyzing the incompatible points may be the only way to detect partial model regress.

6.3 Effect of Regularization

We now highlight the benefits and limitations of regularization, introduced by prior work in the context of backward compatibility, under the presence of noisy model updates [3]. The work proposes a new loss function to penalize newly introduced errors, enabling engineers to adjust the accuracy versus compatibility tradeoff during retraining. The loss penalizes the new model h_2 by its classification loss L whenever the initial model h_1 is correct.

$$L_c = L \ , \ \lambda_c \quad {}^1\mathbb{1}^1h_1{}^1x^0 = y^0 \quad L^0 \tag{3}$$

Increasing the penalty λ_c increases the compatibility of h_2 with respect to h_1. Using the same settings on the CIFAR-10 dataset under label noise as in Section 6.1.1, we vary λ_c and measure compatibility, overall accuracy, and subgroup accuracy on the categories subject to noise (cars and trucks). Figure 6 shows that while regularization improves both accuracy and compatibility of h_2, the subgroup accuracy is often worse than the overall accuracy, and also unable to reach the optimal accuracy of h_2 without any noise. Given the high initial performance of h_1 on the subgroup, this observed limitation suggests that future techniques especially designed for backward compatibility *under noise* can be more effective in obtaining the biggest gains from model updates.

Results summary: Prior methods to address backward compatibility help to improve performance under noisy updates to some extent, but there exists a need for stronger techniques that are aware of the source of incompatibility (e.g., biased noise).

7 BACKWARD COMPATIBILITY IN MACHINE LEARNING PIPELINES

Finally, we position the analysis from the previous experiments in the context of the ML pipeline described in Section 1, and demonstrate how not controlling for backward compatibility during updates can lead to downstream degradation for specific input spaces.

Recall our example of an off-the-shelf OCR model being used for downstream tasks by non-experts, such as receipt fraud detection. Consider the case where, in an effort to improve the OCR model's performance on a variety of fonts, the model is updated with a

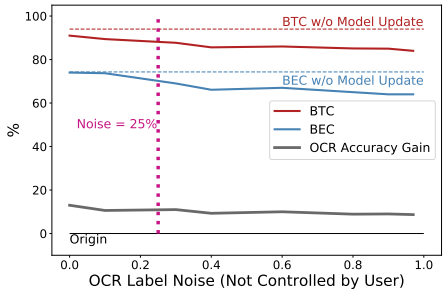


Figure 7: Backward compatibility analyses for the OCR pipeline component.

large, noisy dataset created from scraping the internet for images of different characters and asking people to label them through CAPTCHA tasks [39]. Common mistakes by human annotators, such as mixing up "i" and "l", or "o" and "0" because of occlusion or rushing through the task, may occur often leading to decreased performance on these inputs. A fraud detection system that relies on the classifier's ability to discriminate between "i" and "l" or "o" and "0", will suddenly experience unexpected failures by misrecognizing spoofed terms such as "N1ke" for legitimate companies (i.e., "Nike").

We simulate this setting by assuming a text localization model using bounding boxes to detect characters followed by a character recognition model trained on the Char74k dataset [9]. As in the previous experiments, we train h_1 on a clean, smaller subset of this data (20%), and then re-train h_2 using the entire training data with varying degrees of label noise (swapping "i" w/ "l" and "o" w/ "0"). Figure 7 shows the backward compatibility analyses for this update.

Now, let us assume that the large dataset collection resulted in 25% label noise, but gives h_2 a strong 14% improvement over h_1. The **89% BTC** and **71% BEC** scores suggest that there has been a decrease in backward compatibility when compared to the baselines. A quick glance through the incompatible points reveals examples such as those in Table 2, the most common classes, with digit "0" and lower case letter "l" consisting of 19% and 16% of all incompatible points, respectively. Interestingly, the upper case "Z" often appears in the set of incompatible points despite noise not directly influencing this character. This emphasizes that, even though the noise impact from a data quality perspective may be isolated to particular characters, its impact on the classification output may be broader. Therefore, it is not sufficient to only monitor examples that were explicitly impacted by noise, but instead to have a holistic view of where performance drops to best understand ramifications.

Next, we try to understand the effects on fraud detection. For simplicity, if we assume that the error rate of each character is constant for all fonts and requires at least one misrecognition for the blacklist expression to fail, we can calculate an error likelihood for specific word inputs based on character-based accuracy:

$$\text{Error(word)} = \text{Error}(c_1, c_2, ..., c_n) = 1 - \prod_{i=1}^{n} \text{Accuracy}^1 c_i^0 \quad (4)$$

where $\text{Accuracy}^1 c^0$ is the model accuracy on character c. Thus, if the financial services team formed a blacklist of particular words to catch, then we can estimate the increase in error for each blacklisted word when the OCR model updates to h_2 (Table 3) by knowing the model accuracy for each character. Additionally, because of the nature of label noise, other characters beyond the manipulated targets shift in performance, such as the letter "Z", causing an increase in error for all words in the blacklist that account for misrecognitions of "Z" (e.g., "Zup" instead of "7up" drink). Thus, while the overall accuracy of word recognition might improve after the update, the performance of the system on the specific words that are included in the blacklist heuristics may degrade significantly.

Table 2: Incompatible OCR examples (25% noise).

		
True Label: Digit 0 19% of incompatible points	True Label: lower L 16% of incompatible points	True Label: upper Z 13% of incompatible points
Model 1 Accuracy: 89% Model 2 Accuracy: 10%	Model 1 Accuracy: 77% Model 2 Accuracy: 17%	Model 1 Accuracy: 79% Model 2 Accuracy: 21%

Table 3: Downstream failures in receipt fraud detection.

Fraud Attack	Error Score (h_1)	Error Score (h_2)
"Nlke"	55%	86.4%
"G00gle"	85.3%	99.01%
"ZUP"	49%	84.6%

Without considering backward compatibility as captured by the **BTC** and **BEC** scores, the accuracy gain of the OCR system may be enough of an incentive for its designers to update the model and unintentionally cause significant losses for the fraud detection team. Note that, since the error score definition is conservative (i.e., only one character needs to be wrong for fraud detection to fail) and the average accuracy rates of h_1 and h_2 classifiers (currently 64% and 77%) could be improved with more advanced architectures, the issue would still occur even for more accurate classifiers.

Notably, there may be updates of an OCR system that do not involve noise, yet harm backward compatibility. For example, the OCR system could leverage natural language estimates on word likelihoods before a prediction—thus, assuming "G00gle" would more likely be "Google". The assumption that this will improve the model will cause serious issues for downstream tasks that rely on exactly detecting the unlikely words.

8 DISCUSSION & CONCLUSION

We showed that measuring backward compatibility can identify unreliability issues during an update and help ML practitioners avoid unexpected downstream failures. As ML pipelines designed for interaction with human users become larger and more pervasive, we expect backward compatibility to become an important property for assuring performance and reusability of models over time, and for maintaining trust with end users. While significant efforts continue with building end-to-end differentiable systems, we expect composable, modular systems to remain relevant per interpretability needs, ease of error detection, and worst-case performance analysis in safety-critical systems [17]. Examples of such complex systems include: (1) search engines relying on sensitive metrics for different demographics and markets during page ranking; (2) motion sensing in video game hardware that rely on hybrid

data and physics-based models for detection and tracking; and, (3) productivity applications that combine deterministic software with statistical models for delivering consistent interaction with users.

We note that lack of backward compatibility may not translate to a drop in performance in the real-world. Benchmark datasets used for evaluation may contain examples that are rarely encountered during future uses of the system, and if not related to a high-stakes situation, it might be desirable to sacrifice backward compatibility for accuracy gains. Additionally, user expectations or task definitions in the evolving world may change, which might make evaluation benchmarks obsolete.

As a future direction, the monitoring of backward compatibility would benefit from stronger explanation and visualization techniques. Although error analysis approaches [31, 41] provide reports for single-model performance by identifying explainable regions of data more likely to fail, such visualizations comparing multiple models should be developed. Furthermore, current approaches to addressing issues such as data noise are often model agnostic, and assume that the model is always learned from scratch. Real-world deployment requirements can be better met if data denoising and repair is informed by the previous model that was trained on an initial, cleaner dataset. Finally, we investigate simple noise settings to reflect practices with benign intentions where the practitioner decides to integrate larger and richer data that might contain noise. An interesting future direction would be to study the effect of adversarial training attacks on backward compatibility.

We hope that our work highlights the value of moving beyond evaluations that center on the aggregate performance of models in isolation and traditional assumptions of uniform monotonic improvements. Engineering of ML systems will often require careful approaches to model and data versioning that include methods for identifying and addressing challenges of backward compatibility.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *ICSE-SEIP*. IEEE.

[2] Sean Andrist, Dan Bohus, Ece Kamar, and Eric Horvitz. 2017. What went wrong and why? diagnosing situated interaction failures in the wild. In *ICSR*.

[3] Gagan Bansal, Besmira Nushi, Ece Kamar, Daniel S Weld, Walter S Lasecki, and Eric Horvitz. 2019. Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff. In *AAAI*, Vol. 33. 2429–2437.

[4] Jan Bosch. 2009. From software product lines to software ecosystems. In *SPLC*.

[5] Veronika Cheplygina, Marleen de Bruijne, and Josien PW Pluim. 2019. Not-so-supervised: a survey of semi-supervised, multi-instance, and transfer learning in medical image analysis. *Medical image analysis* 54 (2019), 280–296.

[6] Veronika Cheplygina, Isabel Pino Peña, Jesper Holst Pederson, David Lynch, Lauge Sørensen, and Marleen de Bruijne. 2018. Transfer Learning for Multicenter Classification of Chronic Obstructive Pulmonary Disease. *IEEE Journal of Biomedical and Health Informatics* 22, 5 (2018), 1486–1496.

[7] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. 2019. Slice finder: Automated data slicing for model validation. In *ICDE*.

[8] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *SIGMOD*, Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.). ACM, 541–552. https://doi.org/10.1145/2463676.2465327

[9] T. E. de Campos, B. R. Babu, and M. Varma. 2009. Character recognition in natural images. In *VISAPP*.

[10] FICO. 2018 (accessed February 13, 2020). *Explainable machine learning challenge*. https://community.fico.com/s/explainable-machine-learning-challenge?tabset-3158a=4fbc8.

[11] Benoît Frénay and Michel Verleysen. 2013. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems* 25, 5 (2013), 845–869.

[12] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[13] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211* (2013).

[14] Dan Hendrycks and Thomas Dietterich. 2019. Benchmarking neural network robustness to common corruptions and perturbations. *ICLR*.

[15] Lasse Holmstrom and Petri Koistinen. 1992. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks* 3 (1992).

[16] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2017. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. In *ICML*.

[17] Andrej Karpathy. 2017. Software 2.0. https://medium.com/@karpathy/software-2-0-a64152b37c35

[18] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. 2018. Measuring catastrophic forgetting in neural networks. In *AAAI*.

[19] Jonathan Krause, Benjamin Sapp, Andrew Howard, Howard Zhou, Alexander Toshev, Tom Duerig, James Philbin, and Li Fei-Fei. 2016. The unreasonable effectiveness of noisy data for fine-grained recognition. In *ECCV*.

[20] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009).

[21] Robert William Kruppa and Ravinder Prakash. U.S. Patent 0 298 668, Dec. 2008. Method for fraud detection using multiple scan technologies.

[22] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial Machine Learning at Scale. *ICLR*.

[23] Edith Law and Luis von Ahn. 2011. Human computation. *Synthesis lectures on artificial intelligence and machine learning* 5, 3 (2011), 1–121.

[24] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[25] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. 2017. Learning from Noisy Labels with Distillation. *ICCV*, 1928–1936.

[26] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL*.

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. *ICLR*.

[28] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[29] Gideon Mendels, Erica Cooper, Victor Soto, Julia Hirschberg, Mark Gales, Kate Knill, Anton Ragni, and Haipeng Wang. 2015. Improving Speech Recognition and Keyword Search for Low Resource Languages Using Web Data. *ISCA* (2015).

[30] Nagarajan Natarajan, Inderjit S. Dhillon, Pradeep Ravikumar, and Ambuj Tewari. 2013. Learning with Noisy Labels. In *NeurIPS*.

[31] Besmira Nushi, Ece Kamar, and Eric Horvitz. 2018. Towards accountable ai: Hybrid human-machine analyses for characterizing system failure. In *HCOMP*.

[32] Besmira Nushi, Ece Kamar, Eric Horvitz, and Donald Kossmann. 2017. On human intellect and machine failures: Troubleshooting integrative machine learning systems. In *AAAI*.

[33] Erhard Rahm and Hong Hai Do. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.

[34] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2019. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal* (2019), 1–22.

[35] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[36] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *NeurIPS*.

[37] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir D. Bourdev, and Rob Fergus. 2015. Training Convolutional Networks with Noisy Labels. In *ICLR 2015*.

[38] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. 2019. An empirical study of example forgetting during deep neural network learning. *ICLR*.

[39] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. 2003. CAPTCHA: Using hard AI problems for security. In *EUROCRYPT*. Springer.

[40] Junfeng Wen, Chun-Nam Yu, and Russell Greiner. 2014. Robust Learning under Uncertain Test Distributions: Relating Covariate Shift to Model Misspecification. *ICML*.

[41] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. 2018. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *TVCG* 25, 1 (2018), 364–373.

[42] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael R Lyu, and Miryung Kim. 2019. An empirical study of common challenges in developing deep learning applications. In *ISSRE*.