Understanding the Limits of Unsupervised Domain Adaptation via Data Poisoning

Akshay Mehra[1], Bhavya Kailkhura[2], Pin-Yu Chen[3] and Jihun Hamm[1]
[1]Tulane University [2]Lawrence Livermore National Laboratory [3]IBM Research
{amehra, jhamm3}@tulane.edu, kailkhura1@llnl.gov, pin-yu.chen@ibm.com

Abstract

Unsupervised domain adaptation (UDA) enables cross-domain learning without target domain labels by transferring knowledge from a labeled source domain whose distribution differs from the target. However, UDA is not always successful and several accounts of 'negative transfer' have been reported in the literature. In this work, we prove a simple lower bound on the target domain error that complements the existing upper bound. Our bound shows the insufficiency of minimizing source domain error and marginal distribution mismatch for a guaranteed reduction in the target domain error, due to the possible increase of induced labeling function mismatch. This insufficiency is further illustrated through simple distributions for which the same UDA approach succeeds, fails, and may succeed or fail with an equal chance. Motivated from this, we propose novel data poisoning attacks to fool UDA methods into learning representations that produce large target domain errors. We evaluate the effect of these attacks on popular UDA methods using benchmark datasets where they have been previously shown to be successful. Our results show that poisoning can significantly decrease the target domain accuracy, dropping it to almost 0% in some cases, with the addition of only 10% poisoned data in the source domain. The failure of UDA methods demonstrates the limitations of UDA at guaranteeing cross-domain generalization consistent with the lower bound. Thus, evaluation of UDA methods in adversarial settings such as data poisoning can provide a better sense of their robustness in scenarios unfavorable for UDA.

1 Introduction

The problem of domain adaptation (DA) arises when the training and the test data distributions are different, violating the common assumption of supervised learning. In this paper, we focus on unsupervised DA (UDA), which is a special case of DA when no labeled information from the target domain is available. This setting is useful for applications where obtaining large-scale well-curated datasets is both time-consuming and costly. The seminal works [1, 2] proved an upper bound on a classifier's target domain error in the UDA setting leading to several algorithms for learning in this setting. Many of these algorithms rely on learning a domain invariant representation by minimizing the error on the source domain and a divergence measure between the marginal feature distributions of the source and target domains. Popular divergence measures include total variation distance, Jensen-Shannon divergence [9, 29, 33], Wasserstein distance [27, 13, 7], and maximum mean discrepancy [17, 15, 18]. The success of these algorithms is argued in terms of minimization of the upper bound proposed in [1] along with an improvement in the target domain accuracy on benchmark UDA tasks.

Despite this success on benchmark datasets, some works [9, 14, 32, 30] have presented evidence of the failure of these methods under different scenarios. Recent works have tried to explain this apparent failure of UDA methods by proposing new upper bounds [32, 6] on the target domain error while others have demonstrated the cause of failure with experiments where learning a domain invariant representation that minimizes the source domain error causes an increase in the error of the ideal joint hypothesis [14] leading to an increase in the upper bound proposed by [1]. However, a large upper bound does not guarantee the failure of UDA methods. Therefore, in this work, we prove a lower bound on the target domain error that provides a necessary condition for the success of learning under the UDA setting and complements the upper bound of [1]. Our lower bound is dependent on the difference between the labeling functions for source and target domain data induced by the representation map. For cases where the induced labeling functions match on the source and the target domain data (i.e., favorable case), the success of UDA is explained using the upper bounds proposed by previous works [1, 32]. For a representation that aligns the source and the target domain data and minimizes the error on the source but induces labeling functions that don't agree on the source and the target domain data (i.e., unfavorable case), our lower bound explains the failure of UDA. Our analysis brings to light yet another case (i.e., ambiguous case) of data distributions where success and failure of UDA are equally likely. This is due to the lack of label information from the target domain. Due to this, a small amount of misinformation about the target domain labels can turn the distribution into the unfavorable case leading to a significant increase in the target domain error.

Motivated from our analysis of UDA based on the lower bound, we evaluate the performance of current UDA methods in presence of a small amount of adversarially crafted data. For this purpose, we propose novel data poisoning attacks, using mislabeled and clean-label points. Using our attacks we study how the presence of poisoned data fools UDA methods into learning a representation that incurs high target domain error. We evaluate the effect of our attacks on popular UDA methods using benchmark datasets, where they were previously shown to be very effective. Our poisoning attacks cause UDA methods to either align incorrect classes from the two domains or prevent correct classes from being very close in the representation space. Both of these lead to the failure of UDA methods at reducing target domain error. Our results show a significant drop in the target domain accuracy for current UDA methods, dropping to almost 0% in some cases, with just 10% poison data in the source domain. This dramatic failure of UDA methods demonstrates their limits and suggests that the future UDA methods must be evaluated not only on clean benchmark datasets but also in adversarial settings such as poisoning to evaluate their effectiveness in scenarios unfavorable for UDA.

Our main contributions are as follows:

- We prove a simple lower bound on the target domain error providing a necessary condition for the success of learning in the UDA setting. The bound shows the failure of learning a domain invariant representation that minimizes source domain error at guaranteeing target generalization.
- We present example distributions where UDA succeeds, fails, and where success and failure are equally likely. The last case opens door to adversarial attacks such as data poisoning.
- We propose novel data poisoning attacks that use clean-label and mislabeled points to demonstrate the dramatic failure of UDA methods at target generalization on benchmark datasets. Our poisoning attacks can be used to obtain a better sense of the robustness of the future UDA methods beyond just evaluating accuracy on simple benchmark datasets.

2 Background and related work

Analysis of Domain Adaptation: Several previous works have studied the problem of DA and have provided conditions under which DA is possible [2, 1, 19, 20, 8, 3]. [1, 2] proposed an upper bound on the target domain error which is the inspiration of many UDA algorithms. Recent works [32, 6] have improved the upper bounds on target domain error and have proposed a lower bound dependent on the labeling functions in the input space. Using this lower bound, the failure of learning in the UDA setting was explained in the case when marginal label distributions are different between the two domains. Our lower bound, on the other hand, is dependent on the labeling functions for the source and target domains, induced by the representation. This suggests that if a representation induces labeling functions that disagree on source and target domains then DA provably fails even if the representation is domain invariant and minimizes error on the source domain. Thus our lower bound directly explains the observations of failure of UDA in many previous works [9, 14, 30, 32].

Algorithms for UDA: Most algorithms [9, 29, 16, 33] for UDA learn a representation that is invariant to the domain shift and minimizes error on the source domain. A popular adversarial approach to domain adaptation is DANN [9] which uses a discriminator to distinguish points from source and target domains based on their representations. Another popular method CDAN[16], uses classifier output along with representations to identify domains of the points. IW-DAN and IW-CDAN [6] were recently proposed as extensions of the original DANN and CDAN with an importance weighting scheme to minimize the mismatch between the labeling distributions of the two domains. A different approach MCD [25], makes use of two task-specific classifiers as discriminators to align the two domains. This method adversarially trains the representation to minimize the disagreement between the two classifiers on the target domain data (classifier discrepancy) while training the classifiers to maximize this discrepancy. Another recent approach, SSL [31] uses self-supervised learning tasks (e.g. rotation angle prediction) to better align the two domains. In this work, we study the effect of poisoning on these methods as they have been shown to be effective at various UDA tasks.

Data poisoning: Data poisoning [4, 23, 11, 5, 12, 28] is a training time attack where the attacker has access to the training data which will be used by the victim for training. The attacker intends to modify the training data such that victim's model behaves as the attacker intended, after training. Several works [34, 24, 26, 21, 10] have shown the limitations of supervised learning methods in presence of poisoned data. Recently, the effects of poisoning were also studied against training methods that produce certifiably robust classifiers [22]. However, all these methods consider poisoning in a single domain setting. We are the first to study the effect of poisoning in the UDA setting with two domains and using popular UDA algorithms for training rather than only using empirical risk minimization.

3 When does learning fail in the unsupervised domain adaption setting?

Notations and settings: Let $\mathcal{X}$ be the data domain and $\mathcal{D}$ be the distribution over $\mathcal{X}$ with the corresponding pdf $p(x)$. We assume there is a deterministic labeling function $f : \mathcal{X} \to [0, 1]$ for the given binary classification task. The $f(x)$ can be interpreted as $Pr[y = 1|x]$. Let $g : \mathcal{X} \to \mathcal{Z}$ denote the representation map that maps an input instance x to its features where $\mathcal{Z}$ is called feature or representation space. Let $h : \mathcal{Z} \to [0, 1]$ be a hypothesis for binary classification on the representation space. Note that the representation map g induces a distribution over $\mathcal{Z}$ denoted by $Pr_{\tilde{\mathcal{D}}}[B] := Pr_{\mathcal{D}}[g^{-1}(B)]$ and the corresponding density function $\tilde{p}(z)$ on $\mathcal{Z}$ [2]. The g also induces the labeling function

$$\tilde{f}(z) := E_{\mathcal{D}}[f(x)|g(x) = z], \tag{1}$$

for any B such that $g^{-1}(B)$ is $\mathcal{D}$-measurable. The misclassification error $e(h)$ w.r.t. the induced labeling function is $e(h) = E_{z\sim\tilde{\mathcal{D}}}[|\tilde{f}(z) - h(z)|]$ where $\tilde{\mathcal{D}}$ is the induced distribution over $\mathcal{Z}$. Similarly, we define $e(\tilde{f}, \tilde{f}') = E_{z\sim\tilde{\mathcal{D}}}[|\tilde{f}(z) - \tilde{f}'(z)|]$ and $e(h, h') = E_{z\sim\tilde{\mathcal{D}}}[|h(z) - h'(z)|]$. The distributions $\tilde{p}$ and the labeling functions $\tilde{f}$ for the source and the target domains will be written as $\tilde{p}_S$, $\tilde{p}_T$, $\tilde{f}_S$ and $\tilde{f}_T$, respectively. The total variation distance is $D_1(\tilde{p}, \tilde{p}') = \int_{\mathcal{Z}} |\tilde{p}(z) - \tilde{p}'(z)|dz$.

3.1 Lower bound on the target domain error

Most adversarial DA methods learn a domain invariant representation $g : \mathcal{X} \to \mathcal{Z}$ by minimizing error on the source domain and penalizing the mismatch between the **marginal** source and target distributions since the conditional distribution $\tilde{p}_T(z|y)$ for target domain is unavailable in the UDA setting. Some works [30, 32, 14] have shown this to be insufficient at guaranteeing target generalization. Recent works have proposed a new upper bound [32] or argued failure in terms of the upper bound in [1, 2] ($e_T(h) \leq \min\{e_T(\tilde{f}_S, \tilde{f}_T)), e_S(\tilde{f}_S, \tilde{f}_T))\} + e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T)$) being large. However, a large upper bound does not guarantee failure. Thus, we prove a simple lower bound on the target domain error which shows the necessary condition for the success of learning in the UDA setting and also explains the failure of current UDA methods at guaranteeing target generalization.

Theorem 1. *Let $\mathcal{H}$ be the hypothesis class and $\mathcal{G}$ be the class representation maps. Then, for all $h \in \mathcal{H}$ and $g \in \mathcal{G}$,*

$$e_T(h) \geq \max\{e_S(\tilde{f}_S, \tilde{f}_T), e_T(\tilde{f}_S, \tilde{f}_T)\} - e_S(h) - D_1(\tilde{p}_S, \tilde{p}_T). \tag{2}$$

The proof is in Appendix A. Even though our bound depends on the total variation distance which is a strict measure of distance and is difficult to estimate, the bound still explains the limitations of UDA. Extending the bound to different divergence metrics is left as future work. The following corollary is immediate from Theorem 1.

Corollary 1.1. *For all $h \in \mathcal{H}$, $g \in \mathcal{G}$,*

$$e_T(h) \geq \max\{e_S(\tilde{f}_S, \tilde{f}_T), e_T(\tilde{f}_S, \tilde{f}_T)\} - e_S(h) - \sqrt{0.5 D_{KL}(\tilde{p}_S||\tilde{p}_T)}.$$

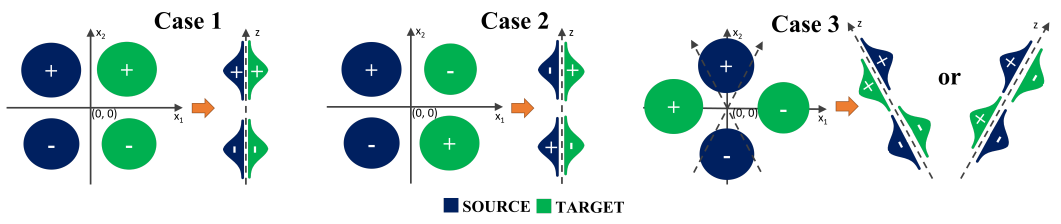


Figure 1: Illustrative cases for UDA. In UDA, one can observe the source data (blue blobs), the source labels (class + and -), and the target data (green blobs) but not the target labels. The optimal linear representation $g(x) = u^T x$ from the input space in $\mathbb{R}^2$ to the feature space in $\mathbb{R}$ (dashed line) that minimizes the source loss and the alignment loss in Eq. 3 can be computed accurately. Depending on the original data distribution, the optimal representation can be successful (Case 1), fail (Case 2), or be undetermined (Case 3). Case 3 has two global minima and poisoning can make the bad solution (right) to be chosen, leading to a large error on the target domain data.

This is due to the Pinsker's inequality: $D_1(p, p') \leq \sqrt{0.5 D_{KL}(p||p')}$. The interpretation of Eq. 2 is as follows. Since f_T is not observable, the goal of UDA is to minimize the observable source classification error $e_S(h)$ and the domain mismatch $D_1(\tilde{p}_S, \tilde{p}_T)$ (or other metrics such as $d_{\mathcal{H}\Delta\mathcal{H}}(\tilde{p}_s, \tilde{p}_T)$):

$$\min_{g,h} \; e_s(h) + D_1(\tilde{p}_S, \tilde{p}_T). \tag{3}$$

This leads to maximization of the second and the third term in the RHS of Eq. 2. With overparameterized models such as deep neural nets, it is often possible to get $e_S(h) \simeq 0$ and $D_1(\tilde{p}_S, \tilde{p}_T) \simeq 0$. Consequently, $e_T(h) \geq \max\{e_S(\tilde{f}_S, \tilde{f}_T), e_T(\tilde{f}_S, \tilde{f}_T)\}$. If $\tilde{f}_S$ and $\tilde{f}_T$ disagree on the source and target domains in the representation space, target domain error $e_T(h)$ will be **provably** large.

Corollary 1.2. *For all $h \in \mathcal{H}$ and $g \in \mathcal{G}$,*

$$|e_T(h) - e_S(\tilde{f}_S, \tilde{f}_T)| \leq e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T), \quad \text{and} \quad |e_T(h) - e_T(\tilde{f}_S, \tilde{f}_T)| \leq e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T).$$

This is obtained by combining the upper and the lower bounds (Appendix A). Thus a UDA method can only guarantee the target error $e_T(h)$ to be close to the labeling function mismatch $e(\tilde{f}_S, \tilde{f}_T)$. Whether $e(\tilde{f}_S, \tilde{f}_T)$ becomes larger or smaller after solving Eq. 3 is data/model dependent. For concreteness, we analyze the cases when UDA methods succeed and fail using the following illustrative examples.

3.2 Illustrative examples of UDA failure

Assume mixture-of-Gaussian distributions $p_S(x)$ and $p_T(x)$ for the source and the target domains in the input space, shown as blue and green blobs in Fig. 1. We consider a linear representation map $g(x) = u^T x$ from the input space $\mathcal{X} \subset \mathbb{R}^2$ to the feature space $\mathcal{Z} \subset \mathbb{R}$ (dashed line) that minimizes the source classification loss plus the marginal mismatch loss in Eq. 3. The optimal solution g to the minimization problem can be found accurately using a mix of analytical and numerical optimization (detailed in Appendix B). We demonstrate three cases of data distributions. In Case 1 (favorable case), the true labeling function for source and target is $f_S((x_1, x_2)) = f_T((x_1, x_2)) = I[x_2 \geq 0]$, that is, f is 1 in the upper halfspace and 0 in the bottom halfspace. One can verify (Appendix B) that the optimal u is the vertical direction ($u = [0, 1]^T$) and the best hypothesis is $h(z) = I[z \geq 0]$, in which case $e_S(h) = 0$ and $D_1(\tilde{p}_S, \tilde{p}_T) = 0$. That is, perfect source classification and a perfect alignment of the marginals are achieved. Furthermore, the true labeling function in $\mathcal{Z}$ is $\tilde{f}_S(z) = \tilde{f}_T(z) = I[z \geq 0]$ (from Eq. 1) and therefore $e(\tilde{f}_S, \tilde{f}_T) = 0$ as well as the target loss $e_T(h) = 0$. In other words, the representation that minimizes Eq. 3 simultaneously **minimizes** $e(\tilde{f}_S, \tilde{f}_T)$, achieving the goal of reducing $e_T(h)$. However in Case 2 (unfavorable case), the true labeling function for target is upside-down $f_T((x_1, x_2)) = I[x_2 \leq 0]$. Since UDA does not use the target label nor the true labeling function, the optimal g is exactly the same as Case 1 ($u = [0, 1]^T$), in which case we still have $e_S(h) = 0$ and $D_1(\tilde{p}_S, \tilde{p}_T) = 0$, but the labeling function mismatch becomes $e(\tilde{f}_S, \tilde{f}_T) = 1$ as well as $e_T(h) = 1$ which is the worst case. In other words, the representation that minimizes Eq. 3 simultaneously **maximizes** $e(\tilde{f}_S, \tilde{f}_T)$, totally failing at the goal of reducing $e_T(h)$. Case 3 exemplifies an ambiguous case. The optimal projection minimizing e_S is still the vertical direction $u = [0, 1]^T$ but the optimal projection minimizing the alignment loss $D_1(\tilde{p}_S, \tilde{p}_T)$ can be either of the $\pm 45^\circ$ directions ($u = [\pm 1/\sqrt{2}, 1/\sqrt{2}]^T$) with no preference of one over the other. Therefore the optimal solution u for Eq. 3 that trades off the source error and the mistmatch loss has two equal-valued global solutions $[\pm u_1, u_2]^T$ for some u_1, u_2. One solution (Fig. 1, Case 3, left) yields a small $e(\tilde{f}_S, \tilde{f}_T)$ and the other (Fig. 1, Case 3, right) yields a large $e(\tilde{f}_S, \tilde{f}_T)$. As can be intuitively seen, UDA is successful in the former but fails in the latter, and which equal-valued solution will be chosen is undetermined. Thus a slight nudge via additional information (such as target labels) or misinformation (such as poisoning) can determine success or failure of UDA methods. Details of the analysis and the results are in Appendix B.

4 Breaking unsupervised domain adaptation methods with data poisoning

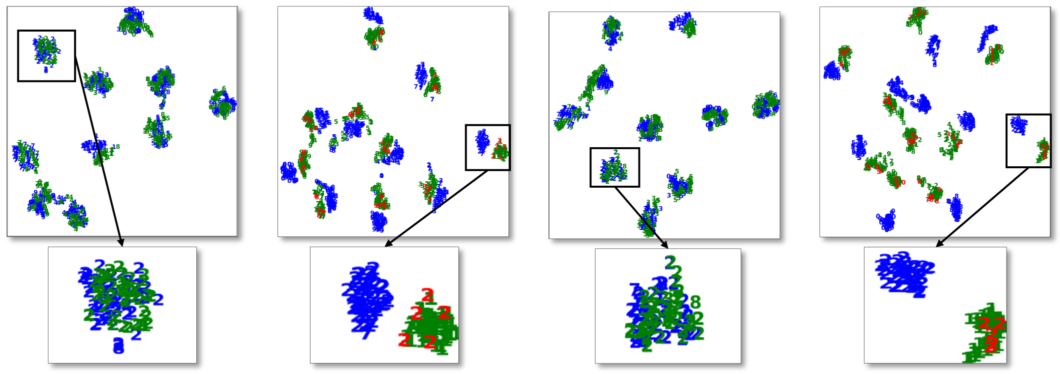
In this section, we present novel data poisoning attacks to evaluate the ease with which current UDA methods can be fooled into producing a representation that leads to a large error on the target domain[1]. We propose three methods to generate poisoned data which will be added to the clean source domain data. The first method uses mislabeled data as poisons. Under this attack, we evaluate two approaches (a) adding mislabeled source domain data (wrong-label correct-domain poisoning) and (b) adding mislabeled target domain data (wrong-label incorrect-domain poisoning) as poisons. The second method adds images from the source domain watermarked with images from the target domain with incorrect labels (watermarking attack) as poisons. The last method adds poisons with clean labels. We evaluate two approaches for this attack (a) using source domain data (clean-label correct-domain poisoning) and (b) using target domain data (clean-label wrong-domain poisoning) to initialize the poison data. The intuitive pictures of how these poisoning attacks hurt UDA methods are shown in Figs. 4, 5, and 6. We evaluate the effect of poisoning on popular UDA methods namely, DANN[9], CDAN[16], MCD[25], SSL[31] (with rotation-angle prediction task), IW-DAN[6], and IW-CDAN[6]. We compare the difference in the target accuracy attainable by these methods when using clean versus poisoned data. Two benchmark datasets are used in our experiments, namely Digits and Office-31. We evaluate four tasks using SVHN, MNIST, MNIST_M, and USPS datasets under Digits and six tasks under the Office-31 using Amazon (A), DSLR (D), and Webcam (W) datasets. For all experiments, we train UDA algorithms on clean and poisoned source domain data and use neural networks whose architectures are similar to those used by previous works (see Appendix E).

4.1 Poisoning using mislabeled source and target domain data

In this experiment, mislabeled data is added to the clean source domain as poison. The labels provided by the attacker to the poison data are chosen to fool UDA methods into learning a representation that incurs large errors on the target domain. We use two simple and effective labeling functions for this. For the Digits dataset, we use a labeling function that systematically labels the poison data to the class next to their true class (e.g. poison points with true class one are labeled as two, points with true class two are labeled as three, and so on). For the Office-31 dataset, we use a labeling function that assigns the poison data the label of the closest (in the representation space learned using the clean source domain data) incorrect source domain class. Here the attacker is limited to adding only 10% poisoned data with respect to the size of the available target domain data (for experiments with different poison percentages see Appendix C). In wrong-label correct-domain poisoning, mislabeled source domain data are used as poisons. The results in rows marked with $\text{Poison}_{\text{source}}$ in Tables 1 and 2, show that UDA methods suffer only a minor decrease in target domain accuracy with this approach. This happens because the presence of a large amount of correctly labeled source domain data prevents the small amount of poisoned data from affecting the performance of UDA methods. A similar effect is observed when a small amount of mislabeled data is used for poisoning in the traditional single domain setting [21, 24]. However, if the relative size of poisoned data is larger or comparable to the size of clean source domain data, poisoning can be effective. This is observed in Table 2 when Amazon is the target data. Amazon dataset is roughly 5 times bigger than both DSLR and Webcam datasets. Due to this, the permissible amount of poisoned data (10% of the target domain) makes the size of clean and poisoned data comparable, leading to successful poisoning. Thus, this attack causes UDA methods to fail in presence of a large amount of poisoned data.

In the wrong-label wrong-domain approach, mislabeled target domain data is used for poisoning. The effect of poisoning on discriminator-based methods [9, 16] is shown in Fig. 4. The domain discriminator used in these methods is maximally confused when marginal distributions of the

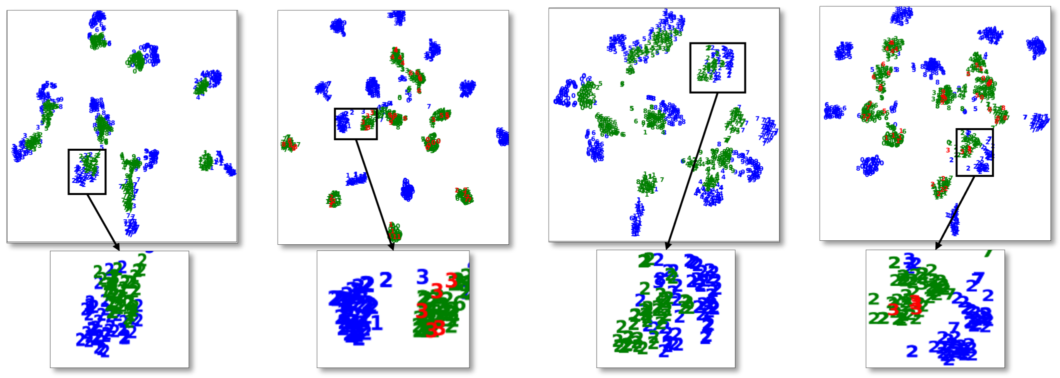
[1]The code is available at `https://github.com/akshaymehra24/LimitsOfUDA`



(a) DANN trained on clean data (b) DANN trained on poisoned data (c) CDAN trained on clean data (d) CDAN trained on poisoned data

🟦 SOURCE 🟩 TARGET 🟥 POISON

Figure 2: (Best viewed in color.) t-SNE embedding of the data in the representation space (for MNIST→ USPS task) learned using DANN and CDAN on clean and poisoned source domain data. Without poisoning, correct classes (data from source class 2 is zoomed in) from two domains are aligned ((a) and (c)). The presence of poisoned data fools the methods into aligning incorrect classes from the two domains ((b) and (d)). The mismatch between the source and target classes is dependent on the labels of the poison data (due to which the target class 1 is aligned to the source class 2).



(a) MCD trained on clean data (b) MCD trained on poisoned data (c) SSL trained on clean data (d) SSL trained on poisoned data

🟦 SOURCE 🟩 TARGET 🟥 POISON

Figure 3: (Best viewed in color.) t-SNE embedding of the data in the representation space (for MNIST→ USPS task) learned using MCD and SSL on clean and poisoned source domain data. Without poisoning, correct classes (data from source class 2 is zoomed in) from two domains are aligned ((a) and (c)). The presence of poisoned data prevents the methods from aligning correct classes from the two domains ((b) and (d)).

source and target domains are aligned. However, alignment of the marginal distributions does not ensure alignment of the conditional distributions [32, 16]. The objective of achieving low source domain error pushes the generator to correctly classify the poison data. Thus placing the poison and source data with the same labels close in the representation space. Since the poison data is mislabeled target domain data, poisoning makes UDA methods align wrong source and target domain classes. This leads to a significant decline in the target domain accuracy. This is also evident from the t-SNE embedding for DANN and CDAN methods in Fig. 2. In absence of poisoning, correct source and target classes are aligned (Fig. 2 (a) and (c)), whereas in presence of poisoning wrong classes from the two domains are closer (Fig. 2 (b) and (d)). For MCD [25], which uses use classifier discrepancy to detect and align source and target domains, our poisoned data prevents the method from detecting target examples. This happens because the term that minimizes the error on the poisoned source domain reduces the discrepancy of the classifiers on poison data,

Table 1: Decrease in the target domain accuracy for UDA methods trained on poisoned source domain data (with poisons sampled from source/target domains) compared to accuracy attained with clean data on the Digits tasks (mean±s.d. of 5 trials).

Method	Data	SVHN → MNIST	MNIST → MNIST_M	MNIST → USPS	USPS → MNIST
Source only	Clean	72.42±1.44	39.05±2.30	87.13±1.75	78.6±1.45
DANN	Clean	78.05±1.15	76.22±2.38	92.17±0.73	92.73±0.71
	Poison$_{source}$	70.26±2.84	69.98±3.49	93.44±0.84	92.08±0.68
	Poison$_{target}$	**1.46±1.12**	**0.48±0.04**	**0.97±0.53**	**5.83±0.82**
CDAN	Clean	79.19±0.70	73.88±1.10	93.92±0.97	95.94±0.71
	Poison$_{source}$	73.67±4.19	73.36±1.31	92.06±0.59	92.85±0.31
	Poison$_{target}$	**12.27±5.02**	**0.59±0.12**	**1.92±0.42**	**2.96±0.71**
MCD	Clean	96.18±1.53	93.95±0.33	89.96±2.04	88.34±2.50
	Poison$_{source}$	85.86±5.66	93.33±0.71	87.99±1.05	83.19±2.98
	Poison$_{target}$	**0.97±0.94**	**0.37±0.06**	**0.66±0.16**	**2.07±0.69**
SSL	Clean	66.85±2.30	92.76±0.91	88.69±1.28	82.23±1.59
	Poison$_{source}$	61.97±1.62	91.35±1.13	85.74±2.92	82.56±0.84
	Poison$_{target}$	**0.31±0.03**	**0.36±0.02**	**7.76±1.52**	**9.88±1.07**

Table 2: Decrease in the target domain accuracy for UDA methods trained on poisoned source domain data (with poisons sampled from source/target domains) compared to accuracy attained with clean data on the Office tasks (mean±s.d. of 3 trials).

Method	Dataset	A → D	A → W	D → A	D → W	W → A	W → D
Source Only	Clean	79.61	73.18	59.33	96.31	58.75	99.68
DANN	Clean	84.06	85.41	64.67	96.08	66.77	99.44
	Poison$_{source}$	79.11±0.35	83.98±1.19	44.31±2.94	95.22±0.22	43.35±1.65	96.58±0.87
	Poison$_{target}$	**59.83±0.20**	**63.18±1.96**	**17.58±0.39**	**76.43±0.62**	**19.82±0.33**	**84.20±0.71**
CDAN	Clean	89.56	93.01	71.25	99.24	70.32	100
	Poison$_{source}$	90.16±0.61	90.94±0.13	53.68±0.37	98.45±0.07	57.27±0.57	99.66±0.23
	Poison$_{target}$	**71.88±0.20**	**71.94±0.76**	**11.19±1.47**	**86.37±0.36**	**18.54±0.45**	**89.08±1.23**
IW-DAN	Clean	84.3	86.42	68.38	97.13	67.16	100
	Poison$_{source}$	81.25±0.91	83.27±0.45	50.76±1.58	96.68±0.29	48.31±2.02	99.73±0.12
	Poison$_{target}$	**61.64±0.53**	**63.43±1.14**	**15.69±1.76**	**80.29±0.07**	**26.54±0.48**	**88.62±0.23**
IW-CDAN	Clean	88.91	93.23	71.9	99.3	70.43	100
	Poison$_{source}$	89.83±0.31	90.77±1.27	57.51±0.06	98.41±0.07	61.16±1.21	99.66±0.12
	Poison$_{target}$	**72.62±0.42**	**70.15±2.21**	**14.36±0.66**	**88.26±0.15**	**22.36±0.96**	**87.75±0.53**

which are from the target domain. Thus, both the generator and discriminator (in the form of two classifiers) become optimal and there is no signal for the generator to align the two domains. The t-SNE embedding in Fig. 3 shows this effect. In presence of poisoned data Fig. 3 (b), we see twenty distinct clusters rather than just ten (we have ten classes in the Digits) as seen in the absence of poisoning in Fig. 3 (a). In SSL [31], the generator must work well on the main task, i.e., should correctly classify all data in the poisoned source domain data. The auxiliary task ensures that representations of the source and target domains become similar as seen on clean data (Fig. 3 (c)). But in presence of poisoned data, similar representations of correct source and target domain classes leads to a drop in the accuracy of the main task on the poisoned data. This creates a conflict between the main and auxiliary tasks due to which correct source and target domain classes cannot be aligned (Fig. 3 (d)). The objective of making the main task accurate on poisoned data leads to target domain data being assigned the labels of the poisoned points. Thereby leading to a significant drop in the target-domain accuracy. The results of wrong-label wrong-domain poisoning present in rows marked with Poison$_{target}$ in Tables 1 and 2 show a significant reduction in the target domain accuracy compared to the accuracy obtained on clean data. On Digits, poisoning makes the target domain accuracy close to 0% on most tasks. On Office-31, poisoning causes at least a 20% reduction in the target domain accuracy in most cases. The reason tasks in Office-31 are less hurt by poisoning is because of the use of a pre-trained representation. Similar to previous works for Office-31, we use all labeled source and unlabeled target domain data for training and fine-tune from a representation pre-trained on the ImageNet dataset. The

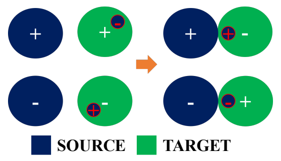


Figure 4: Poisoning with mislabeled target domain data causes discriminator-based UDA approaches to align wrong classes (+ to -) from the two domains, leading to a significant decline in the target domain accuracy.

Table 3: Decrease in target accuracy when training different domain adaptation methods on poisoned watermarked data in comparison to the target accuracy obtained with clean data on the Digits task (mean±s.d. of 5 trials).

Method	Dataset	SVHN $\rightarrow$ MNIST	MNIST $\rightarrow$ MNIST_M	MNIST $\rightarrow$ USPS	USPS $\rightarrow$ MNIST
DANN	Clean	78.05±1.15	76.22±2.38	92.17±0.73	92.73±0.71
	Poisoned$_\alpha$	68.76±3.91$_{0.05}$	27.36±15.77$_{0.05}$	91.84±0.55$_{0.10}$	88.93±4.36$_{0.10}$
		57.96±5.84$_{0.10}$	7.19±2.59$_{0.10}$	85.51±3.01$_{0.20}$	78.29±8.52$_{0.20}$
		33.33±4.38$_{0.15}$	4.73±0.38$_{0.15}$	39.29±1.34$_{0.30}$	41.52±7.43$_{0.30}$
CDAN	Clean	79.19±0.70	73.88±1.10	93.92±0.97	95.94±0.71
	Poisoned$_\alpha$	65.77±4.82$_{0.05}$	55.47±3.87$_{0.05}$	92.05±0.96$_{0.10}$	86.53±1.55$_{0.10}$
		57.57±3.11$_{0.10}$	7.37±1.26$_{0.10}$	86.54±2.43$_{0.20}$	77.39±4.84$_{0.20}$
		44.83±4.09$_{0.15}$	6.68±1.64$_{0.15}$	88.67±0.44$_{0.30}$	79.54±7.02$_{0.30}$
MCD	Clean	96.18±1.53	93.95±0.33	89.96±2.04	88.34±2.50
	Poisoned$_\alpha$	74.96±3.20$_{0.05}$	92.18±0.78$_{0.05}$	6.75±4.81$_{0.10}$	30.35±2.30$_{0.10}$
		35.85±3.23$_{0.10}$	85.38±3.57$_{0.10}$	0.77±0.22$_{0.20}$	11.34±0.77$_{0.20}$
		17.01±1.52$_{0.15}$	70.34±11.49$_{0.15}$	0.71±0.22$_{0.30}$	3.28±0.94$_{0.30}$
SSL	Clean	66.85±2.30	92.76±0.91	88.69±1.28	82.23±1.59
	Poisoned$_\alpha$	44.64±2.01$_{0.05}$	53.33±13.48$_{0.05}$	32.38±10.77$_{0.10}$	34.72±1.71$_{0.10}$
		10.86±1.21$_{0.10}$	26.64±10.1$_{0.10}$	6.12±2.13$_{0.20}$	21.86±1.01$_{0.20}$
		3.4±1.11$_{0.15}$	12.14±4.66$_{0.15}$	2.42±0.41$_{0.30}$	11.90±0.81$_{0.30}$

slowly changing representation pre-trained on a massive dataset weakens the effect of poisoning but cannot eliminate it. For the two tasks D $\rightarrow$ W and W $\rightarrow$ D, the fine-tuned representation, trained just on the clean source dataset (Source Only in Table 2) achieves high accuracy indicating the domains are already well aligned in terms of conditional distributions as well. As a result, UDA methods can easily align correct classes and suffer a drop close to 10% which is the amount of poisoned data added. However, this is not an interesting case for the evaluation of UDA methods as domains can be aligned just by training on the source domain data without the need for target domain data.

4.2 Poisoning with mislabeled watermarked data

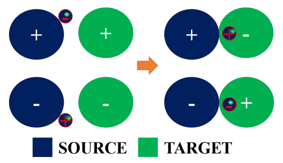


Figure 5: Successful poisoning with mislabeled watermarked data prevents discriminator based UDA approaches from aligning correct classes from the source and target domains.

In this experiment, we evaluate the effect of using poisoned data that looks like the source domain data. The poisoned data is generated by superimposing an image from the source domain with an image from the target domain. This method of generating poison data is known as watermarking [26]. To generate watermarked poison data we select an image from the target domain (t) and a base image from the source domain (s) such that it has the same class as the target domain image and lies closest to the target image (in the input space). The poisoned image (p) is obtained by a convex combination of the base and target images i.e., $p = \alpha t + (1 - \alpha)s$ where $\alpha \in [0, 1]$. α is selected such that the target image is not visible in the poison image ensuring the poisoned image looks like the image from the source domain. We use the same labeling function as discussed in the previous section to label the poisoned image and add 10% poison data to the source. The illustrative picture of the effect of poisoning in this scenario is presented in Fig. 5. Successful poisoning, in this case, works just like in the previous experiment i.e., by making the representations of the data from wrong classes in source and target domains similar for DANN/CDAN, by reducing the discrepancy between the classifiers on target data for MCD and inducing a conflict between the supervised and auxiliary task for SSL. The t-SNE embedding showing the effect of poisoning (Fig. 10) and watermarked poison data (Fig. 9) are shown in the Appendix. We evaluate the effectiveness of this method on the Digits dataset for different values of α. The results in Table 3 show a significant decrease in the target domain accuracy even with a small watermarking percentage for all methods except CDAN. This is because the success of CDAN is dependent on the correctness of the pseudo-labels on the target domain data (output of the classifier), which are used in the discriminator. Correct pseudo-labels provide CDAN a positive reinforcement to align correct classes from the two domains, leading to a failure of poisoning. However, as we increase the amount of watermarking, the quality of pseudo labels deteriorates. Thus, providing a negative reinforcement to CDAN that causes the alignment of wrong classes from the two domains.

4.3 Poisoning using clean-label source and target domain data

In this experiment, correctly labeled data is used for poisoning. To generate clean-label poison data that can affect the performance of UDA methods we must affect the features of the poison data. This requires solving a bilevel optimization problem [10, 21, 22] which we present in the Appendix. Due to the high computational complexity involved in solving the bilevel problem, we propose to use a simple alternating optimization to demonstrate the feasibility of a clean label poisoning attack against UDA. We use the setting of previous works[10, 26] and consider misclassification of a single target domain test point $(x_{\text{test}}^{\text{target}}, y_{\text{test}}^{\text{target}})$ rather than affecting the accuracy of the entire target domain as done in the previous two experiments. Let $u = \{u_1, ..., u_n\}$ denote the poisoned data. To ensure a clean label, each poison point u_i must have a bounded perturbation from a base point x_i^{base} i.e, $\|u_i - x_i^{\text{base}}\| = \|\delta_i\| \leq \epsilon$ and has label of the base i.e., y_i^{base}. Thus, $\hat{\mathcal{D}}^{\text{poison}} = \{(u_i, y_i^{\text{base}})\}_{i=1}^{N_{\text{poison}}}$, $\hat{\mathcal{D}}_{\text{source}} = \{(x_i^{\text{source}}, y_i^{\text{source}})\}_{i=1}^{N_{\text{source}}}$ and $\hat{\mathcal{D}}_{\text{target}} = \{(x_i^{\text{target}}, y_i^{\text{target}})\}_{i=1}^{N_{\text{target}}}$. The clean-label poison data u is such that when the victim uses $\hat{\mathcal{D}}^{\text{source}} \bigcup \hat{\mathcal{D}}^{\text{poison}}$ and $\hat{\mathcal{D}}^{\text{target}}$ for UDA, the target domain test point $(x_{\text{test}}^{\text{target}}, y_{\text{test}}^{\text{target}})$ is misclassified. The optimization problem for the clean-label attack is as follows.

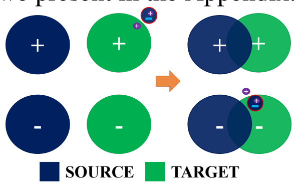


Figure 6: Successful clean-label clean-domain poisoning attack aligns the target point (purple) close to the wrong class (-).

$$\min_u \sum_{i=1}^{N_{\text{poison}}} \left[\|g(x_{\text{test}}^{\text{target}};\theta) - g(u_i;\theta)\|_2^2 + \lambda \|x_i^{\text{base}} - u_i\|\right], \ \min_\theta \ \mathcal{L}_{\text{UDA}}(\hat{\mathcal{D}}^{\text{source}} \bigcup \hat{\mathcal{D}}^{\text{poison}}, \hat{\mathcal{D}}^{\text{target}}; \theta). \quad (4)$$

The first problem minimizes the distance between the representations of the poison and the target domain test data (first term) while ensuring the poison data is not too far from the base data (second term). The second problem optimizes the parameters of the representation using UDA methods. Attack success is evaluated by solving the second problem in Eq. 4 from scratch and evaluating the classification of $x_{\text{test}}^{\text{target}}$. This is illustrated in Fig. 6. The left part shows the case before retraining using the poison data generated from Eq. 4 and the right part shows how poisoning induces misclassification. We use two approaches for poisoning. The first uses source domain data and the second uses target domain data as base data. We add 1% poisoned data and test the effect of poisoning on a two-class (3 vs 8) domain adaptation problem on MNIST$\rightarrow$ MNIST_M (see Appendix E). The results in Fig. 7 show that using target domain data as base data is significantly more successful under small permissible perturbation (ϵ). Using base data from the source domain requires larger distortion to keep the poison data close to the target point in the representation space and is hence less successful. This experiment shows the feasibility of clean label attacks against UDA methods. We believe attack success can be further boosted by solving the bilevel level problem (Eq. 9 in Appendix) and is left as future work.

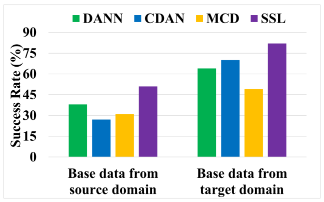


Figure 7: Attack success rate of clean-label poisoning using base data from source/target for a two-class problem in MNIST $\rightarrow$ MNIST_M.

5 Conclusion

We studied the problem of UDA and highlighted the limitations of learning under this setting. We proposed a simple lower bound on the target domain error for UDA, dependent on the labeling function induced by the representation. The lower bound demonstrated that learning a domain invariant representation while minimizing error on the source domain cannot guarantee good generalization on the target domain. We analyzed a simple model and showed the existence of cases where UDA can naturally succeed or fail. The analysis also highlighted a case where, without access to any labeled target domain data, the success and the failure are equally likely. In such a case, the presence of even a small amount of poisoned data can make the data distribution unfavorable for UDA methods, making them fail dramatically in comparison to the case without poisoning. We proposed novel data poisoning attacks to demonstrate the failure of popular UDA methods with a small amount of poisoned data. Our results suggest that the performance of a UDA method in presence of poisoned data indicates how well the method aligns the conditional distributions across the two domains.Thus, we believe our attacks can be used for evaluating UDA methods, beyond simple benchmark datasets, to reveal their robustness to data distributions inherently unfavorable for UDA.

6 Acknowledgement

This work was supported by the NSF EPSCoR-Louisiana Materials Design Alliance (LAMDA) program #OIA-1946231 and by LLNL Laboratory Directed Research and Development project 20-ER-014 (LLNL-CONF-824206). This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract No. DE-AC52-07NA27344, Lawrence Livermore National Security, LLC [2].

References

[1] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

[2] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 19:137, 2007.

[3] Shai Ben-David and Ruth Urner. On the hardness of domain adaptation and the utility of unlabeled target samples. In *International Conference on Algorithmic Learning Theory*, pages 139–153. Springer, 2012.

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[5] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[6] Remi Tachet des Combes, Han Zhao, Yu-Xiang Wang, and Geoff Gordon. Domain adaptation with conditional distribution matching and generalized label shift. *arXiv preprint arXiv:2003.04475*, 2020.

[7] Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. *arXiv preprint arXiv:1705.08848*, 2017.

[8] Shai Ben David, Tyler Lu, Teresa Luu, and Dávid Pál. Impossibility theorems for domain adaptation. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 129–136. JMLR Workshop and Conference Proceedings, 2010.

[9] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[10] W Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. *arXiv preprint arXiv:2004.00225*, 2020.

[11] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.

[12] Yujie Ji, Xinyang Zhang, and Ting Wang. Backdoor attacks against learning systems. In *2017 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2017.

[13] Jaeho Lee and Maxim Raginsky. Minimax statistical learning with wasserstein distances. *arXiv preprint arXiv:1705.07815*, 2017.

[2]This document was prepared as an account of the work sponsored by an agency of the United States Government. Neither the United States Government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or Lawrence Livermore National Security, LLC. The views and opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

[14] Hong Liu, Mingsheng Long, Jianmin Wang, and Michael Jordan. Transferable adversarial training: A general approach to adapting deep classifiers. In *International Conference on Machine Learning*, pages 4013–4022. PMLR, 2019.

[15] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[16] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. *arXiv preprint arXiv:1705.10667*, 2017.

[17] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer joint matching for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1410–1417, 2014.

[18] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. *arXiv preprint arXiv:1602.04433*, 2016.

[19] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. *arXiv preprint arXiv:0902.3430*, 2009.

[20] Yishay Mansour and Mariano Schain. Robust domain adaptation. *Annals of Mathematics and Artificial Intelligence*, 71(4):365–380, 2014.

[21] Akshay Mehra and Jihun Hamm. Penalty method for inversion-free deep bilevel optimization. *arXiv preprint arXiv:1911.03432*, 2019.

[22] Akshay Mehra, Bhavya Kailkhura, Pin-Yu Chen, and Jihun Hamm. How robust are randomized smoothing based defenses to data poisoning? *arXiv preprint arXiv:2012.01274*, 2020.

[23] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[24] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

[25] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3723–3732, 2018.

[26] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

[27] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[28] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018.

[29] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[30] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11293–11302, 2019.

[31] Jiaolong Xu, Liang Xiao, and Antonio M López. Self-supervised domain adaptation for computer vision tasks. *IEEE Access*, 7:156694–156706, 2019.

[32] Han Zhao, Remi Tachet Des Combes, Kun Zhang, and Geoffrey Gordon. On learning invariant representations for domain adaptation. In *International Conference on Machine Learning*, pages 7523–7532. PMLR, 2019.

[33] Han Zhao, Shanghang Zhang, Guanhang Wu, José MF Moura, Joao P Costeira, and Geoffrey J Gordon. Adversarial multiple source domain adaptation. *Advances in neural information processing systems*, 31:8559–8570, 2018.

[34] Chen Zhu, W Ronny Huang, Ali Shafahi, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. *arXiv preprint arXiv:1905.05897*, 2019.

Appendix

We present the proof of Theorem 1 in Appendix A followed by the analysis of the illustrative cases of UDA failure in Appendix B. Then we present the results for the experiment of using different poison percentages when mislabeled data is used for poisoning in Appendix C followed by the proposed bilevel formulation for clean label attacks in Appendix D. We conclude in Appendix E by providing the details of the datasets used, model architectures, and the clean label experiment.

A Proof of the lower bound on the target domain loss

Theorem 1. *Let $\mathcal{H}$ be the hypothesis class and $\mathcal{G}$ be the class representation maps. Then, for all $h \in \mathcal{H}$ and $g \in \mathcal{G}$,*

$$e_T(h) \geq \max\{e_S(\tilde{f}_S, \tilde{f}_T), e_T(\tilde{f}_S, \tilde{f}_T)\} - e_S(h) - D_1(\tilde{p}_S, \tilde{p}_T).$$

Proof.

$$\begin{aligned} e_S(h) + e_T(h) &= e_T(h, \tilde{f}_T)) + e_T(h, \tilde{f}_S) + e_S(h, \tilde{f}_S) - e_T(h, \tilde{f}_S) \\ &\geq e_T(\tilde{f}_S, \tilde{f}_T) + e_S(h, \tilde{f}_S) - e_T(h, \tilde{f}_S) \\ &= e_T(\tilde{f}_S, \tilde{f}_T) + \int (\tilde{p}_S(z) - \tilde{p}_T(z))|h(z) - \tilde{f}_S(z)|\, dz \\ &\geq e_T(\tilde{f}_S, \tilde{f}_T) - \int |\tilde{p}_S(z) - \tilde{p}_T(z)|\, dz \\ &= e_T(\tilde{f}_S, \tilde{f}_T) - D_1(\tilde{p}_S, \tilde{p}_T). \end{aligned}$$

Similarly, we can also show that

$$e_S(h) + e_T(h) \geq e_S(\tilde{f}_S, \tilde{f}_T) - D_1(\tilde{p}_S, \tilde{p}_T).$$

Combining the two results gives us the statement of the theorem. □

Corollary 1.2. *For all $h \in \mathcal{H}$ and $g \in \mathcal{G}$,*

$$|e_T(h) - e_S(\tilde{f}_S, \tilde{f}_T)| \leq e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T), \text{ and } |e_T(h) - e_T(\tilde{f}_S, \tilde{f}_T)| \leq e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T).$$

Proof. From the upper bound we have,

$$e_T(h) - e_S(\tilde{f}_S, \tilde{f}_T)) \leq e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T) \text{ and } e_T(h) - e_T(\tilde{f}_S, \tilde{f}_T)) \leq e_S(h) + D_1(\tilde{p}_S, \tilde{p}_T)$$

From the lower bound (Eq. 2) we have,

$$e_T(h) - e_S(\tilde{f}_S, \tilde{f}_T) \geq -e_S(h) - D_1(\tilde{p}_S, \tilde{p}_T) \text{ and } e_T(h) - e_T(\tilde{f}_S, \tilde{f}_T) \geq -e_S(h) - D_1(\tilde{p}_S, \tilde{p}_T)$$

Combining the results from the upper and the lower bounds gives us the statement of the corollary. □

B Illustrative examples of UDA failure

In this section, we provide the details of the analysis of the illustrative cases in the main paper. As described in Sec. 3.2, the input space $\mathcal{X}$ is in $\mathbb{R}^2$ and the source and the target distributions are Gaussian mixtures

$$p_S(x) = 0.5p_{S+}(x) + 0.5p_{S-}(x) \text{ and } p_T(x) = 0.5p_{T+}(x) + 0.5p_{T-}(x),$$

where $p_{S+}(x) = \mathcal{N}(\mu_{S+}, \sigma^2 I)$, $p_{S-}(x) = \mathcal{N}(\mu_{S-}, \sigma^2 I)$, $p_{T+}(x) = \mathcal{N}(\mu_{T+}, \sigma^2 I)$, and $p_{T-}(x) = \mathcal{N}(\mu_{T-}, \sigma^2 I)$. The true labeling function $f(x)$ in the input space is assumed linear: $f(x) = I[v^T x > 0]$ where v is the unit normal vector to the decision boundary. The representation space $\mathcal{Z}$ is in $\mathbb{R}$ and the representation map $g : \mathcal{X} \to \mathcal{Z}$ is linear: $g(x) = u^T x$ where $\|u\| = 1$. For the hypothesis, we use $h(z) = \Phi(az + b)$ which is a linear model $az + b$ followed by a saturating function which can be the cumulative normal distribution Φ (or others such as the logistic function l).

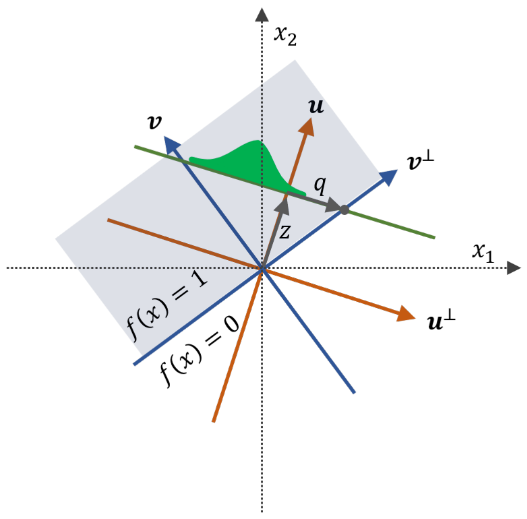


Figure 8: This figure provides a visual help for deriving the induced labeling function $\tilde{f}(z)$ in Eq. 5. u is the direction of the 1-D projection $g(z) = u^T x$, v is the direction of the labeling function $f(x)$ in the input space where we assumed $f(x) = I[v^T x > 0]$, and q is the intersection of the two lines $v^T x = 0$ and $u^T(x - zu) = 0$ projected along the $u^{\perp}$ direction. Evaluating Eq. 6 using the help of this figure results in Eq. 7.

The representation map g induces the distributions $\tilde{p}(z)$ over $\mathcal{Z}$ as

$$\tilde{p}_S(z) = 0.5\mathcal{N}(u^T\mu_{S+}, \sigma^2) + 0.5\mathcal{N}(u^T\mu_{S-}, \sigma^2), \quad \text{and}$$
$$\tilde{p}_T(z) = 0.5\mathcal{N}(u^T\mu_{T+}, \sigma^2) + 0.5\mathcal{N}(u^T\mu_{T-}, \sigma^2).$$

The map g also induces the labeling function $\tilde{f}(z)$ on $\mathcal{Z}$ defined as $\tilde{f}(z) = E_{\mathcal{D}}[f(x)|g(x) = z]$ [2]. Computing this quantity can be complex in general but is relatively straightforward for a mixture of Gaussians and a simple half-space labeling function $f(x)$. Following the definition, we have

$$\tilde{f}(z) = E_{\mathcal{D}}[f(x)|g(x) = z] = \int_{\mathcal{Z}} f(x)\; I[u^T x = z]\; p(x|z = g(x))dx. \tag{5}$$

In our example, the integral $\int_{\mathbb{R}^2} \cdot\, dx$ can be decomposed into $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \cdot\, dz dw$ where z and w are the coordinates along the rotated axes u and $u^{\perp}$ (see Fig. 8).

We therefore have

$$\begin{aligned} \tilde{f}(z) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I[v^T x > 0]\; I[u^T x = z]\; p(x|z = g(x))dz dw \\ &= \int_{-\infty}^{\infty} I[v^T x > 0]\; (0.5\mathcal{N}_w(\mu_+^T u^{\perp}, \sigma^2 I) + 0.5\mathcal{N}_w(\mu_+^T u^{\perp}, \sigma^2 I))dw. \end{aligned} \tag{6}$$

This integral can be evaluated as

$$\tilde{f}(z) = \begin{cases} 0.5\Phi(\frac{q-\mu_+^T u^{\perp}}{\sigma}) + 0.5\Phi(\frac{q-\mu_+^T u^{\perp}}{\sigma}) & \text{if } \; v^T u^{\perp} < 0 \\ 0.5[1 - \Phi(\frac{q-\mu_+^T u^{\perp}}{\sigma})] + 0.5[1 - \Phi(\frac{q-\mu_+^T u^{\perp}}{\sigma})] & \text{if } \; v^T u^{\perp} > 0 \\ 0.5(1 + sign(z)) & \text{if } \; v^T u^{\perp} = 0 \;\text{ and }\; v^T u > 0 \\ 0.5(1 - sign(z)) & \text{if } \; v^T u^{\perp} = 0 \;\text{ and }\; v^T u < 0 \end{cases} \tag{7}$$

where Φ is the cumulative normal distribution and q is the intersection of the two lines $v^T x = 0$ and $u^T(x - zu) = 0$ projected along the $u^{\perp}$ direction. More concretely,

$$q = \frac{u_1 v_1 + u_2 v_2}{u_1 v_2 - u_2 v_1} z$$

where $u = [u_1, u_2]^T$ and $v = [v_1, v_2]^T$.

The UDA minimization problem is

$$\min_{u,a,b} \; e_S(h) + \lambda D(\tilde{p}_S, \tilde{p}_T) + \eta(\|u\|^2 - 1)^2, \tag{8}$$

where the last term was added to enforce $\|u\| = 1$. For differentiability, we consider the squared loss instead of the absolute loss:

$$e_S(h) = E_S[(\Phi(az+b) - \tilde{f}_S(z))^2] = \int_{\mathbb{R}} \tilde{p}_S(z) \left(\Phi(az+b) - \tilde{f}_S(z)\right)^2 dz$$

and also

$$D(\tilde{p}, \tilde{p}') = \int_{\mathbb{R}} (\tilde{p}(z) - \tilde{p}'(z))^2 dz.$$

The expectation in $e_S(h)$ can only be computed numerically due to the complex formula for $\tilde{f}(z)$. On the other hand, the mismatch loss is

$$\begin{aligned} D(\tilde{p}_S(z), \tilde{p}_T(z)) &= \int_{\mathbb{R}} (\tilde{p}_S(z) - \tilde{p}_T(z))^2 dz \\ &= \int_{\mathbb{R}} \left(\frac{0.5}{2\pi\sigma^2}\right)^2 \left[e^{-\frac{(z-u^T\mu_{S+})^2}{2\sigma^2}} + e^{-\frac{(z-u^T\mu_{S-})^2}{2\sigma^2}} - e^{-\frac{(z-u^T\mu_{T+})^2}{2\sigma^2}} - e^{-\frac{(z-u^T\mu_{T-})^2}{2\sigma^2}}\right]^2 dz, \end{aligned}$$

which can be computed either numerically or analytically.

The three cases explained in the main paper are as follows:

Case 1 : $\mu_{S+} = [-1,1]^T$, $\mu_{S-} = [-1,-1]^T$, $\mu_{T+} = [1,1]^T$, $\mu_{T-} = [1,-1]^T$, $v_S(x) = v_T(x) = [0,1]^T$, $\lambda = 10^{-1}$

Case 2 : $\mu_{S+} = [-1,1]^T$, $\mu_{S-} = [-1,-1]^T$, $\mu_{T+} = [1,-1]^T$, $\mu_{T-} = [1,1]^T$, $v_S(x) = -v_T(x) = [0,1]^T$, $\lambda = 10^{-1}$

Case 3 : $\mu_{S+} = [0,1]^T$, $\mu_{S-} = [0,-1]^T$, $\mu_{T+} = [-1,0]^T$, $\mu_{T-} = [1,0]^T$, $v_S(x) = [0,1]^T$, $v_T(x) = [-1,0]^T$, $\lambda = 10^{-2}$

The other shared parameters are $\sigma = 1$ and $\eta = 10$. The λ determines the optimal tradeoff between E_s and D in Eq. 8.

We solve Eq. 8 numerically using *scipy.optimize.minimize(method='Nelder-Mead')* function which is stable even if the cost function may be non-differentiable. Starting from random initial conditions and running until convergence, the solution u for both Case 1 and Case 2 converges to $[0,1]^T$.

For Case 1 (favorable case), we get $\max\{e_S(\tilde{f}_S, \tilde{f}_T), e_T(\tilde{f}_S, \tilde{f}_T)\} < 10^{-3}$ and $e_T(h) < 10^{-3}$ which shows DA was successful.

For Case 2 (unfavorable case), we get $\max\{e_S(\tilde{f}_S, \tilde{f}_T), e_T(\tilde{f}_S, \tilde{f}_T)\} > 0.99$ and $e_T(h) > 0.99$ which shows DA was unsuccessful.

For Case 3 (ambiguous case), there are roughly 50/50% chance of u converging to $[-0.70, 0.72]^T$ or $[0.70, 0.72]^T$. For the former, we get $\max\{e_S(\tilde{f}_S, \tilde{f}_T), e_S(\tilde{f}_S, \tilde{f}_T)\} < 10^{-4}$ and $e_T(h) < 10^{-3}$ where DA is successful. For the latter, we get $\max\{e_S(\tilde{f}_S, \tilde{f}_T), e_S(\tilde{f}_S, \tilde{f}_T)\} \approxeq 0.33$ and $e_T(h) \approxeq 0.33$ where DA has failed.

C Effect of poison percentage on attack success with mislabeled poison data

In this section, we evaluate the effect of using different poison percentages on the attack success when mislabeled data is used for poisoning. As can be seen in Tables 1 and 2, the success of wrong-label clean-domain poisoning with 10% poisoned data is very limited. Thus, here we only focus on using a smaller poison percentage to study the attack success of wrong-label wrong-domain poisoning. The results of the experiment are summarized in Table 4. For all tasks, the presence of only 6% poison data causes a significant decrease in the target domain accuracy. When the poison percentage is decreased further to only 2% we still see a drop of at least 20% in the target accuracy for all methods except CDAN[16]. The use of a conditional discriminator provides CDAN this robustness. However, the success of CDAN is dependent on the quality of the pseudo-labels from the classifier on the target

Table 4: Effect of using different percentage of wrong-label wrong-domain poisoned data on the target domain accuracy when training UDA methods on poisoned source domain data on the Digits tasks (mean$\pm$s.d. of 5 trials).

$\text{Poison}_{\text{target}}$ (%)	DANN		CDAN		MCD		SSL	
	MNIST $\to$ USPS	USPS $\to$ MNIST	MNIST $\to$ USPS	USPS $\to$ MNIST	MNIST $\to$ USPS	USPS $\to$ MNIST	MNIST $\to$ USPS	USPS $\to$ MNIST
0% (Clean)	92.17$\pm$0.73	92.73$\pm$0.71	93.92$\pm$0.97	95.94$\pm$0.71	89.96$\pm$2.04	88.34$\pm$2.50	88.69$\pm$1.28	82.23$\pm$1.59
2%	63.53$\pm$2.09	94.72$\pm$0.63	90.54$\pm$0.91	88.79$\pm$2.34	22.74$\pm$2.17	51.02$\pm$3.57	65.88$\pm$2.93	41.25$\pm$2.32
4%	28.39$\pm$4.78	34.25$\pm$9.53	90.22$\pm$0.74	76.55$\pm$2.25	2.37$\pm$1.41	16.66$\pm$4.73	30.82$\pm$1.28	28.60$\pm$2.16
6%	7.32$\pm$4.78	12.96$\pm$7.33	42.86$\pm$5.09	8.61$\pm$4.77	2.56$\pm$0.97	4.64$\pm$1.34	21.29$\pm$2.51	18.89$\pm$1.11
8%	0.97$\pm$0.44	1.63$\pm$0.41	7.02$\pm$3.88	5.35$\pm$0.94	7.04$\pm$0.25	4.43$\pm$1.76	10.84$\pm$1.52	11.11$\pm$2.74
10%	0.97$\pm$0.53	5.83$\pm$0.82	1.92$\pm$0.42	2.96$\pm$0.71	0.66$\pm$0.16	2.07$\pm$0.69	7.76$\pm$1.52	9.88$\pm$1.07

domain data. Good pseudo-labels provide CDAN a positive reinforcement to align correct source and target domain classes. Thus, leading to a failure of poisoning. However, as the percentage of poisoned data increases, the classifier begins to easily classify the target domain data into labels intended by the attacker, deteriorating the quality of the pseudo-labels. This provides a negative reinforcement to CDAN causing it to align wrong classes from the source and the target domain. As a result, the poisoning attack becomes successful. Thus, for wrong-label wrong-domain poisoning, increasing the percentage of poison data gradually drives UDA methods from the case favorable to UDA to the unfavorable one.

D Bilevel formulation for clean-label attacks

In this section, we present the bilevel formulation for a clean-label data poisoning attack against UDA methods. Let $u = \{u_1, ..., u_n\}$ denote the poisoned data and $\hat{\mathcal{D}}^{\text{val}_{\text{target}}} = \{(x_i^{\text{val}_{\text{target}}}, y_i^{\text{val}_{\text{target}}})\}_{i=1}^{N_{\text{val}_{\text{target}}}}$, be a small set of labeled target domain data accessible to the attacker. To ensure a clean label, each poison point u_i must have a bounded perturbation from a base point x_i^{base} i.e, $\|u_i - x_i^{\text{base}}\| = \|\delta_i\| \leq \epsilon$ and has label of the base i.e., y_i^{base}. Thus, $\hat{\mathcal{D}}^{\text{poison}} = \{(u_i, y_i^{\text{base}})\}_{i=1}^{N_{\text{poison}}}$. The clean-label poison data u is such that when the victim uses $\hat{\mathcal{D}}^{\text{source}} \bigcup \hat{\mathcal{D}}^{\text{poison}}$ and $\hat{\mathcal{D}}^{\text{target}}$ for UDA, the accuracy on $\hat{\mathcal{D}}^{\text{val}_{\text{target}}}$ is minimized. The bilevel formulation for this attack is as follows:

$$\max_{u \in \mathcal{U}} \; \mathcal{L}(\hat{\mathcal{D}}^{\text{val}_{\text{target}}}; \theta^*) \;\text{ s.t. }\; \theta^* = \arg\min_{\theta} \; \mathcal{L}_{\text{UDA}}(\hat{\mathcal{D}}^{\text{clean}} \bigcup \hat{\mathcal{D}}^{\text{poison}}, \hat{\mathcal{D}}^{\text{target}}; \theta). \tag{9}$$

The solution to the lower-level problem θ^* are the parameters of the generator and the classifier learned from using a UDA method on the poisoned source domain data and unlabeled target domain data. Solving bilevel optimization problems [10, 21, 22] to generate clean-label poison data has previously been shown to be effective. We used an alternating optimization to avoid the computational complexity of solving the bilevel optimization (Eq. 4). However, we believe the attack success can be boosted by solving the bilevel formulation proposed in Eq. 9 and is left for future work.

E Details of the experiments

All codes are written in Python using Tensorflow/Keras and were run on Intel Xeon(R) W-2123 CPU with 64 GB of RAM and dual NVIDIA TITAN RTX. Dataset details and model architectures used are described below.

E.1 Dataset description

Here we describe the details of the datasets used for the Digits and Office-31 tasks.

Digits: For this task, we use 4 datasets: MNIST, MNIST_M, SVHN, and USPS. We evaluate four popular tasks under this, namely, SVHN$\to$ MNIST, MNIST$\to$ MNIST_M, MNIST$\to$ USPS and USPS$\to$ MNIST. For SVHN$\to$ MNIST, we train on 73,257 images from SVHN and 60,000 images from MNIST while testing on 10,000 MNIST images. For MNIST$\to$ MNIST_M, we use 60,000 from MNIST and MNIST_M for training and test on 10,000 MNIST images. Lastly, for MNIST$\to$ USPS and USPS$\to$ MNIST, we use 2,000 images from MNIST and 1,800 images from USPS for training. We test on the 10,000 MNIST images and 1,860 USPS images.

Office-31: The dataset contains a total of 4110 images belonging to 31 categories from 3 domains: Amazon (A), DSLR(D), and Webcam(W). We evaluate the performance of UDA on all six tasks, namely, A$\rightarrow$ D, A$\rightarrow$ W, D$\rightarrow$ A, D$\rightarrow$ W, W$\rightarrow$ A, W$\rightarrow$ D.

E.2 Model architecture

Here we describe the model architectures used for different tasks. To fairly compare the performance of different UDA methods and eliminate the effect of architecture changes in improving the performance of different methods, we make use of similar model architectures for different methods, as described below. The effectiveness of these architectures has also been shown by previous works.

Digits: The architectures used for MNIST$\rightarrow$ MNIST_M, MNIST$\rightarrow$ USPS and USPS$\rightarrow$ MNIST involves a shared convolution neural network. The output of this shared network is fed into a softmax classifier and the discriminator. The architecture of the shared network consists of a convolution layer with a kernel size of 5x5, 20 filters, and ReLU activation, followed by a max-pooling layer of size 2x2. This is followed by another convolution layer with a 5x5 kernel, 50 filters, and ReLU activation followed by similar max pooling and a dropout. Then we have a fully connected layer with ReLU activation of size 500 followed by a dropout layer. For the discriminator, we use two dense layers with 500 units each followed by a ReLU and a dropout layer. This is followed by a 2 unit softmax layer. For MCD, we use the following architecture for the generator on MNIST$\rightarrow$ MNIST_M task. A convolution layer with a kernel size of 5x5, 32 filters, and ReLU activation, followed by a max-pooling layer of size 2x2. This is followed by another convolution layer with a 5x5 kernel, 48 filters, and ReLU activation followed by a similar max-pooling layer. We use 2 dense layers for the classifier with 100 units followed by ReLU activation and dropout layers. This is followed by the softmax layer. Unlike the original work MCD[25], we do not use batch normalization layers in these tasks to make architectures consistent across different methods. For SVHN$\rightarrow$ MNIST we use the following architecture for the generator. A convolution layer with a kernel size of 5x5, 64 filters, the stride of 2 followed by batch normalization, dropout, and ReLU activation layer. This is followed by another convolution layer with a kernel size of 5x5, 128 filters, the stride of 2 followed by batch normalization, dropout, and ReLU activation layer. Then another convolution layer with a kernel size of 5x5, 256 filters, the stride of 2 followed by batch normalization, dropout, and ReLU activation layer. This is followed by a dense layer with 512 units followed by batch normalization, ReLU activation, and a dropout layer. We use the softmax layer for classification. For the discriminator, we use two dense layers with 500 units each followed by a ReLU and a dropout layer. This is followed by a 2 unit softmax layer. For MCD, we use the same architecture for the generator except that we use max-pooling instead of convolution layers with stride 2 to downsample the representation. The classifier uses the output of the generator and feeds into a dense layer with 256 units followed by batch normalization and ReLU activation layers. This is followed by a softmax layer.

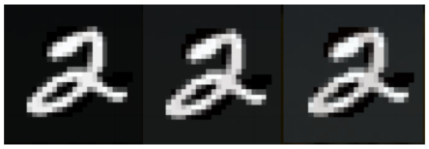


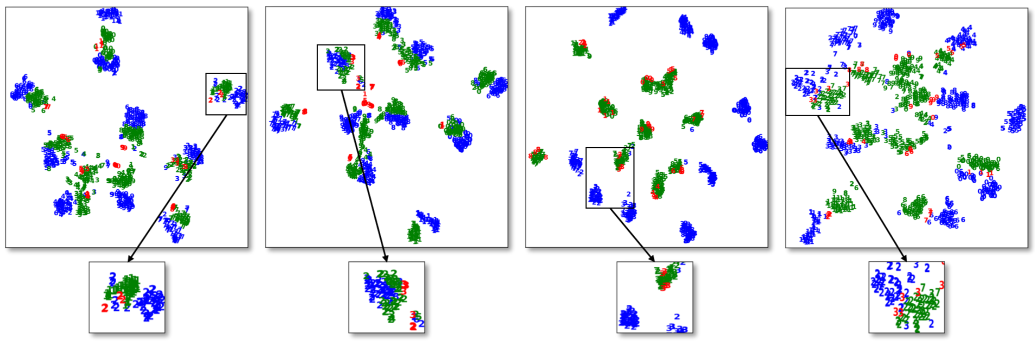
Figure 9: Watermarked poison data for MNIST $\rightarrow$ MNIST_M task with α in {0.05, 0.10, 0.15}.

Office-31: For office experiments, we use the publicly available code of the work[3] [6] and supply the poisoned data by adding them to the input files being used by the code. We use all default options of the code and use DAN, CDAN, IW-DAN, IW-CDAN algorithms. This is done to eliminate the effect of hyperparameters on the performance of the UDA algorithms on the Office-31 dataset and be able to fairly compare the performance of poisoning. To obtain the representation trained only on the source domain data, we initialize a ResNet50 model with weight pre-trained on Imagenet. We then update the representation by training on respective source domain data for different tasks.

E.3 Clean-label attack on MNIST$\rightarrow$ MNIST_M

For this experiment, 1% poison data is used to prevent the alignment of a target test point to its correct class. We test the attack on the binary classification problems (3 vs 8). Two approaches to initializing the poison data are evaluated. In the first approach, the poison data is initialized from the

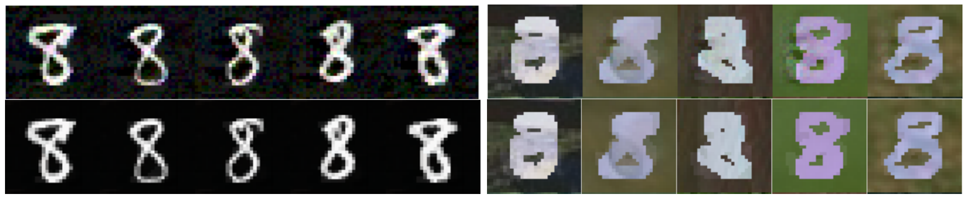
[3] `https://bit.ly/34EFb52`



(a) DANN (b) CDAN (c) MCD (d) SSL

🟦 SOURCE 🟩 TARGET 🟥 POISON

Figure 10: (Best viewed in color). t-SNE embedding of the data in the representation space (for MNIST$\rightarrow$ USPS task) learned using DANN, CDAN, MCD, and SSL on source domain data poisoned with watermarked ($\alpha = 0.3$) data. Successful poisoning aligns the wrong classes for discriminator-based approaches, as seen in (a) with DANN. Poisoning fails against CDAN because of the pseudo-labels being correct on the target data (b). For MCD, we see 20 distinct clusters highlighting the failure of the method at detecting and aligning target domain data (c). For SSL, the poison data has prevented the correct classes from having very similar representations (d). The failure of most UDA methods with a small amount of watermarked data makes our attack practical and raises serious concerns about the success of these methods.



(a) Base data chosen from the source domain (b) Base data chosen from the target domain

Figure 11: Poison data (top rows) obtained after solving Eq. 4 by using DANN as the UDA method, with base data (bottom rows) initialized from the source domain (left) and the target domain (right). Attack success with poison data initialized from the target is significantly higher than the attack success obtained with poison data initialized from the source, from under the same maximum permissible distortion constraint ($\epsilon = 0.1$ in ℓ_∞ norm) as seen in Fig. 7.

source domain data, and in the second approach, it is initialized from the target domain data. In both cases, the poison is picked from the class opposite to the true class of the target test point. Moreover, the poison data is initialized using the points closest in the input space to the target test point. The poison data obtained after solving Eq. 4 is added to the source domain data and UDA methods are retrained from scratch. The attack is considered successful if the target test point is misclassified after this retraining. For the results shown in Fig. 7, we randomly targeted 20 points and obtained poison data corresponding to each UDA method. Attack success is reported after evaluating UDA methods on five random initializations by adding the generated poison data in the source domain. To control the amount of maximum distortion between experiments, we add a constraint on the maximum permissible distortion to poison data using ℓ_∞ norm and use a value of $\epsilon = 0.1$. The poison data obtained after solving the optimization with base data chosen from the source and target domains with DANN as the UDA method are shown in Fig. 11. To generate poison data that remains effective even after UDA methods are trained from scratch, we make use of multiple randomly initialized networks during poison generation. Following the work [10], we reinitialize the models at different points during optimization. This re-initialization scheme helps train UDA methods with different random initializations and for a different number of epochs making the poison data more resilient to initialization change that can happen at test-time.