CrossTrainer: Practical Domain Adaptation with Loss Reweighting

Justin Chen, Edward Gan, Kexin Rong, Sahaana Suri, Peter Bailis
Stanford DAWN Project

ABSTRACT

Domain adaptation provides a powerful set of model training techniques given domain-specific training data and supplemental data with unknown relevance. The techniques are useful when users need to develop models with data from varying sources, of varying quality, or from different time ranges. We build CrossTrainer, a system for practical domain adaptation. CrossTrainer utilizes loss reweighting from [6], which provides consistently high model accuracy across a variety of datasets in our empirical analysis. However, loss reweighting is sensitive to the choice of a weight hyperparameter that is expensive to tune. We develop optimizations leveraging unique properties of loss reweighting that allow CrossTrainer to output accurate models while improving training time compared to naïve hyperparameter search.

ACM Reference Format:
Justin Chen, Edward Gan, Kexin Rong, Sahaana Suri, Peter Bailis. 2019. CrossTrainer: Practical Domain Adaptation with Loss Reweighting. In *International Workshop on Data Management for End-to-End Machine Learning (DEEM'30), June 30, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3329486.3329491

1 INTRODUCTION

The availability of large, labeled training datasets has made the development of data-intensive machine learning models increasingly accessible [15–17, 22, 34]. However, when developing models for new domains, acquiring sufficient training data for a user's target setting can be costly or infeasible. In these scenarios, supplemental but potentially less relevant datasets can also boost model performance by serving as additional training data. For instance, while a data scientist at a small online retailer may only have access to a limited number of reviews from their retail portal to construct a sentiment prediction model, they can augment their small training set with larger public datasets such as the Amazon reviews dataset [25]. This scenario in which users want to train models using limited training data from the *target* domain with supplemental data from a *source* domain is a form of transfer learning called *domain adaptation* (DA) [14, 26].

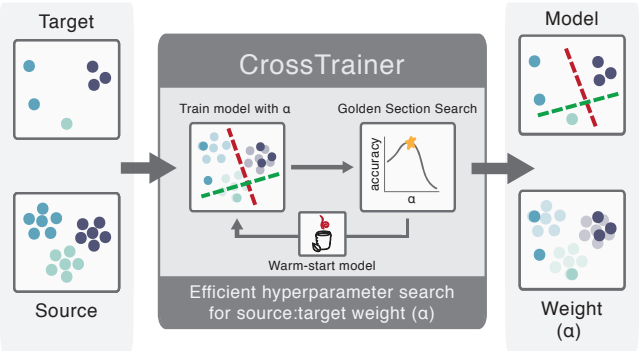


Figure 1: CrossTrainer trains a model to maximize validation accuracy by learning a weighting α between target and source data. CrossTrainer addresses the bottleneck of tuning α with optimizations to reduce both the parameter search space and retraining time.

Despite the common need to deal with data from varying sources and of varying quality, practitioners lack the means to easily apply DA to their machine learning pipelines. In this paper, we develop CrossTrainer, a system for practical DA that integrates with existing ML training workflows, i.e. those using the scikit-learn API. CrossTrainer takes source and target datasets as input, and trains a model optimized for accuracy on the target domain (Figure 1). In building CrossTrainer, we address two practical challenges that are overlooked in the DA literature.

First, the machine learning literature proposes a wide range of methods for domain adaptation [6, 12, 13, 33], with no clear consensus on their applicability. Different DA methods provide different trained model accuracies depending on the target and source datasets, which makes it challenging for users to select the appropriate method a priori. To address this challenge of method choice, we perform an empirical analysis of six DA techniques across eight datasets, focusing

DEEM'30, June 30, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6797-4/19/06…$15.00
https://doi.org/10.1145/3329486.3329491

on techniques that are agnostic to model architecture. We find that loss reweighting [6], which adjusts the loss function of a base model to balance the relative importance of the target and source datasets, consistently achieves high accuracy across a variety of source and domain types. In particular, unlike many alternatives, loss reweighting never performs worse than training with target or source data in isolation. Further, loss reweighting is compatible with any base model that uses loss minimization, including logistic regression and gradient boosted decision trees, and requires no adjustments to standard training procedures and feature formats. As a result, CrossTrainer utilizes loss reweighting as its core method to perform domain adaptation.

However, the accuracy of loss reweighting depends heavily on a key hyperparameter α that determines the relative importance of datasets during training. Naïvely tuning α requires potentially expensive hyperparameter search over many possible values, retraining the model for each configuration. For instance, searching over 100 hyperparameter values for a logistic regression model over 7 million datapoints can take 80 minutes.

This leads us to a second challenge: loss reweighting requires expensive tuning and retraining procedures that can limit its practical application. To address this, CrossTrainer introduces two optimizations for hyperparameter selection that decrease both the number of evaluations and the computational cost of each evaluation. First, to prune the search space, we leverage the unimodal relationship between model accuracy and α, verified both in practice and in the context of a theoretical upper bound introduced in [6]. As a result, CrossTrainer uses golden section search to prune the hyperparameter space [19]. Second, we observe that the hyperparameter search consists of repeatedly training models under similar conditions, yielding model coefficients that are closer to each other than random initialization. To decrease the cost of each evaluation of α, CrossTrainer utilizes warm-start initialization [11], further reducing training time for models trained with iterative procedures such as stochastic gradient descent. Together, these optimizations improve loss reweighting performance by up to 48× compared to a grid search over 100 values.

In summary, we make the following contributions:

- We empirically evaluate six domain adaptation techniques and find that loss reweighting provides consistent best or second-best accuracy across real-world and synthetic workloads.
- Based on these results, we introduce CrossTrainer, a system utilizing loss reweighting to provide accurate and rapid domain adaptation for loss-minimizing models.

```
from crosstrainer import CrossTrainer
from sklearn import linear_model
lr = linear_model.LogisticRegression(...)
ct = CrossTrainer(lr, k=5, delta=0.01)
lr, alpha = ct.fit(
  X_target, Y_target,  # task target
  X_source, Y_source)  # supplemental source
Y_pred = lr.predict(X_test)
```

Figure 2: CrossTrainer usage with scikit-learn. Here we fit a logistic regression classifier, but other classifiers are also compatible.

- We develop optimizations that provide up to 48× speedup for loss reweighting compared to naïve hyperparameter tuning, and analyze the soundness of the optimizations.

The remainder of this paper proceeds as follows. In Section 2, we introduce our system and its usage. In Section 3, we provide details on the loss reweighting technique for domain adaptation. In Section 4, we describe runtime optimizations for CrossTrainer. In Section 5, we evaluate CrossTrainer for accuracy, robustness, and runtime. In Section 6, we discuss related work. We conclude in Section 7.

2 SYSTEM OVERVIEW

CrossTrainer is a system for improving machine learning model accuracy using domain adaptation. Given input target and source datasets, as well as a model to fit, CrossTrainer trains the model using the loss reweighting technique from [6], optimizing for model validation accuracy on the target domain (Figure 1). Users can separately configure hyperparameter settings for model training and loss reweighting. The major CrossTrainer parameters are k, the number of folds used for cross-validation on the target data, and δ, the desired precision of the hyperparameter tuning procedure for loss reweighting's α parameter (i.e., estimate the optimal α^* with respect to validation accuracy within $\alpha^* \pm \delta$).

We implement CrossTrainer in Python[1]. In Figure 2, we demonstrate how CrossTrainer can serve as a drop-in supplement for existing machine learning workflows using the scikit-learn API [27]. CrossTrainer does not require any modifications to feature pre-processing or downstream monitoring pipelines, as it trains models that operate on target ($S_T = (X_T, Y_T)$) and source ($S_S = (X_S, Y_S)$) features directly. As output, CrossTrainer produces a model that performs better than models trained naïvely on only the target or source data separately.

To emphasize CrossTrainer's broad applicability, we describe three motivating use cases.

[1] https://github.com/stanford-futuredata/crosstrainer

Sentiment Analysis. Consider the case of an online shoe retailer who wishes to build a sentiment classifier for reviews on her website. While she can curate a labeled target dataset specific to her needs, it might be expensive for her to collect and label a large enough dataset [17] to effectively train a model. However, if the retailer were to make use of a large source dataset of labeled Amazon reviews [25] in conjunction with a small amount of target shoe review data, CrossTrainer would enable her to train a model that outperforms those trained on either dataset separately.

Data Cleaning. Consider the case of an advertisement content sensor who wants to build a movie genre classifier to detect horror movies using publicly available datasets from IMDb [3] and Yahoo [4]. The IMDb dataset has 50K examples, but its genre labels are noisy (e.g. having "Kids" and "Horror" tagged for the same movie). In contrast, the Yahoo dataset contains only 10K examples but is much cleaner. Correct handling of dirty data has the potential to significantly improve model performance [21]. In this case, CrossTrainer can help users balance the two related datasets and improve the final classifier performance.

Time-varying behaviors. Consider the case of an airline operator who would like to train a model for predicting flight delay status. While current flight information is more relevant for training, there might only be a small amount of recent flight data collected and labeled for use. With CrossTrainer, the operator can leverage larger datasets with labeled historical data to augment the training and enable accurate prediction on current data.

3 LOSS REWEIGHTING

We describe domain adaptation, loss reweighting, and loss reweighting's properties that enable our optimizations.

3.1 Domain Adaptation

In domain adaptation, a *domain* is pair (D, f) consisting of a distribution $\mathcal{D}$ over inputs $x \in \mathcal{X}$ and a labeling function $f : \mathcal{X} \to \{0, 1\}$. Data is drawn from a *target* domain $(\mathcal{D}_T, f_T)$ and a related, not always identical *source* domain $(\mathcal{D}_S, f_S)$. Both domains share the same input space $\mathcal{X}$. The goal of domain adaptation is to learn a classifier h that minimizes the target error, defined as the probability that h disagrees with a labeling function f_T according to the distribution $\mathcal{D}_T$:

$$\epsilon_T(h) = \mathbb{E}_{x\sim\mathcal{D}_T}[|h(x) - f_T(x)|], \quad h, f_T : \mathcal{X} \to \{0, 1\}.$$

We assume that for training, we have access to m instances, where βm instances are drawn from $\mathcal{D}_T$, and $(1 - \beta)m$ are drawn from $\mathcal{D}_S$.

3.2 Loss Reweighting

Loss reweighting [6] is an instance-based method for supervised domain adaptation that adjusts the loss function of a classifier to weight the relative importance of target and source datasets. Loss reweighting seeks to find a classifier h that minimizes the empirical α-error $\hat{\epsilon}_\alpha(h)$, which is a linear combination of the empirical source error $\hat{\epsilon}_S(h)$ and empirical target error $\hat{\epsilon}_T(h)$ for a given $\alpha \in [0, 1]$:

$$\hat{\epsilon}_\alpha(h) = \alpha\hat{\epsilon}_T(h) + (1 - \alpha)\hat{\epsilon}_S(h). \tag{1}$$

Intuitively, the hyperparameter α provides a means to trade off the importance of the target dataset with the value of a potentially larger but less relevant source dataset. By choosing α appropriately, we should be able to learn at least as well as if we only used source ($\alpha = 0$) or target data ($\alpha = 1$), or if we use the union of the source and target data ($\alpha = \beta$).

The authors in [6] show that $\hat{\epsilon}_\alpha(h)$ is a proxy for the true error on the target distribution $\epsilon_T(h)$. Specifically, a classifier $\hat{h} = \min_h \hat{\epsilon}_\alpha(h)$ that minimizes empirical α-error and a classifier $h^*_T = \min_h \epsilon_T(h)$ that minimizes target error satisfies:

$$\epsilon_T(\hat{h}) \leq \epsilon_T(h^*_T) + g(\alpha) \tag{2}$$

$$g(\alpha) = 2B\sqrt{\frac{\alpha^2}{\beta} + \frac{(1-\alpha)^2}{1-\beta}} + 2(1 - \alpha)A. \tag{3}$$

$g(\alpha)$ is thus an upper bound on the cost of optimizing $\hat{\epsilon}_\alpha$ instead of the unknown ϵ_T. $A \in \mathbb{R}_{\geq 0}$ is a constant measuring the distributional divergence between source and target, and $B \in \mathbb{R}_{\geq 0}$ is a constant measuring classifier class complexity, for more details see [6].

3.3 Properties of α and Loss Reweighting

In practice, the optimal reweighting parameter α^* depends on both the size and quality of the source and target training sets. While one could derive an estimate for α^* from the upper bound in Equation 3, good estimates for A are computationally intractable and optimize for an upper bound rather than the true accuracy. Instead, users can empirically tune α as a hyperparameter and optimize for target performance on a validation set. However, the repeated training necessary for tuning α are computationally expensive.

In this section we investigate properties of α and $\hat{\epsilon}_\alpha$ that will allow us to reduce the overhead of naïve hyperparameter search. First, we formally verify the empirical observation in [6] that $g(\alpha)$ is convex, and thus there exists a single value α_g that minimizes $g(\alpha)$.

Proof of convexity of error upper bound. Let

$$z(\alpha) = \sqrt{\frac{\alpha^2}{\beta} + \frac{(1-\alpha)^2}{1-\beta}}.$$

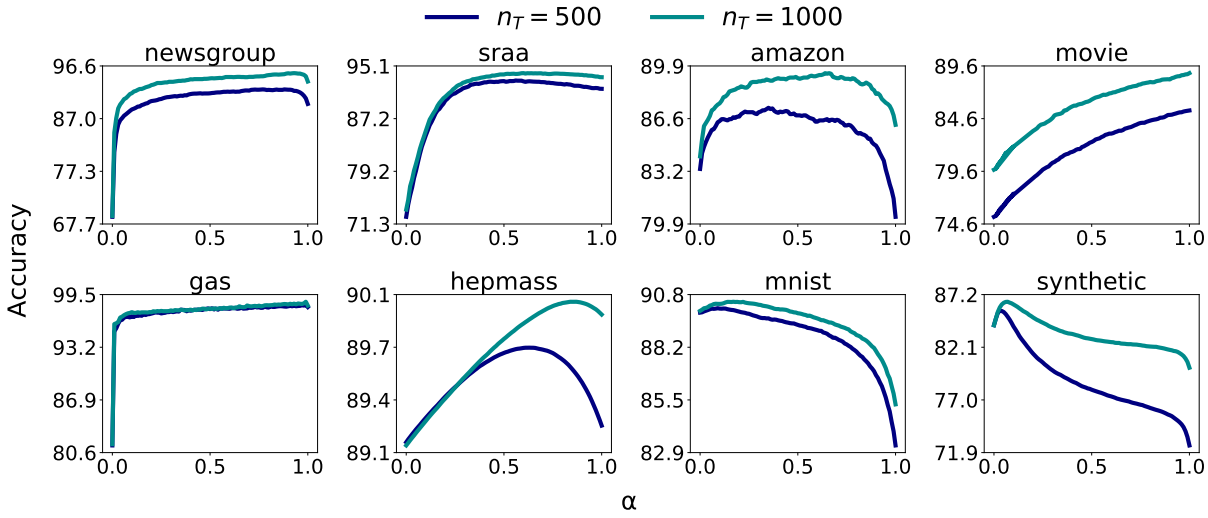


Figure 3: Across a variety of datasets, the accuracy curves are unimodal, but not always concave.

Then,

$$g''(\alpha) = 2B \cdot z''(\alpha) = -\frac{2B}{(\beta(2\alpha-1)-\alpha^2)\cdot z(\alpha)}.$$

We want to show that $g''(\alpha) \geq 0$. Since $B \geq 0, z(\alpha) \geq 0$, we must show that $\beta(2\alpha-1)-\alpha^2 \leq 0$. There are two cases:

(1) $\alpha \leq 0.5$: We have $(2\alpha-1) \leq 2\times 0.5 - 1 = 0$, also given $\beta \geq 0, -\alpha^2 \leq 0 \Rightarrow \beta(2\alpha-1)-\alpha^2 \leq 0$.

(2) $\alpha > 0.5$: Since $\beta \in [0,1]$, $\beta(2\alpha-1) \leq 2\alpha - 1$. Therefore $\beta(2\alpha-1)-\alpha^2 \leq 2\alpha - 1 - \alpha^2 = -(\alpha-1)^2 \leq 0$.

Therefore, $g(\alpha)$ is convex.

Empirical evidence. Equation 3 shows that the error is upper bounded by a convex function of α, but we must verify that the empirical error curve itself is convex in practice. The authors in [6] show a few promising examples, and we extend their empirical evaluation to a wider range of datasets.

We train a series of logistic regression models that minimize empirical α-error under varying α. Figure 3 reports the achieved test accuracy for two different target dataset sizes on a number of datasets. As we report accuracy instead of error, we expect the curves to be concave, not convex.

We first observe that for a wide range of target and source datasets, the optimal choice of α^* is not at $\alpha = 0$ or $\alpha = 1$, indicating that the identification of α^* requires tuning. Further, contrary to the observations in [6], the empirical target accuracy is sometimes not concave (Figure 3). However, the accuracy curves are still all *unimodal*, a weaker property than concavity that describes functions with a unique local maximum. A function $f(x)$ is defined as unimodal if there exists a unique value x^* that maximizes the function, and that the function monotonically increases for $x \leq x^*$ and monotonically decreases for $x > x^*$. Unimodality is sufficient to enable optimizations in Section 4.1 that improve hyperparameter search performance. Although for some datasets, the results have small fluctuations (Figure 3) and are not strictly unimodal, in practice these do not hinder the effectiveness of our optimizations.

4 OPTIMIZING CROSSTRAINER

To enable efficient usage of loss reweighting in CrossTrainer, we reduce the computational overhead of tuning the α parameter. We introduce two optimizations: we reduce the range of α to evaluate using golden section search, and reduce the runtime for each iteration using warm start.

4.1 Golden Section Search

Hyperparameter search is a well studied problem with simple solutions such as grid search, to state of the art improvements to random search such as Hyperband [23] and Bayesian optimization [20]. However, unlike in traditional, domain-agnostic hyperparameter search, here we can take advantage of the unimodal relationship between $\epsilon_T(\hat{h})$ and α to more quickly hone in on α^*.

As we wish to approximate the optimal α^* to a precision of δ, we begin by considering grid search with a granularity of δ (i.e., with $\frac{1}{\delta}$ grid points) as a baseline. However, for logistic regression on datasets with a few million points and with $\delta = 0.01$, the grid search procedure can take on the order of an hour on a modern processor.

Concretely, unimodality allows us to use golden section search (Figure 4) to find a value of α that minimizes target

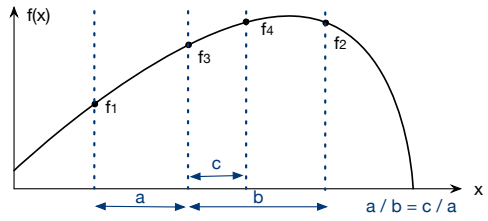


Figure 4: Illustration of golden section search. The spacing between probe points are iteratively narrowed according to the golden ratio.

error [19]. Just as binary search reduces the search space for finding specific values of a monotonic function, golden section search reduces the search space for locating the extremum of a unimodal function. The algorithm works by iteratively narrowing the range of a bracket of three probe points that covers the location of the extremum. Each narrowing reduces the range of the probe points by a factor of $1 - \phi^{-1}$ (where ϕ is the golden ratio), so, to get brackets with range less than δ, we need $O(\log\left(\frac{1}{\delta}\right))$ iterations, compared to $O\left(\frac{1}{\delta}\right)$ for grid search. We find in practice (Table 2) that minor deviations from unimodality do not affect golden section search's ability to approximate α^*. We provide the pseudocode for hyperparameter tuning α using golden section search in Algorithm 1. Note that before the golden section search, we first use a form of binary search to find three values of α that must bracket the optimum value.

Algorithm 1 Find reweighting factor

```
function FIND_BRACKET(l, r, δ)
    m = (l + r) / 2
    a_l = ACCURACY(l), a_m = ACCURACY(m), a_r = ACCURACY(r)
    if r - l < δ or MAX(a_l, a_m, a_r) == a_m then
        return l, m, r        ▷ Stopping criteria, or bracket found
    else if a_l ≤ a_r then                          ▷ Binary search
        return FIND_BRACKET(m, r)
    else
        return FIND_BRACKET(l, m)
    end if
end function

procedure FINDWEIGHTING(δ)
    left, mid, right = FIND_BRACKET(0, 1, δ)
    if ACCURACY(right) > ACCURACY(mid) then
        return right
    else if ACCURACY(left) > ACCURACY(mid) then
        return left
    else
        return GSS(left, mid, right)    ▷ Golden section search
    end if
end procedure
```

4.2 Warm Start

While golden section search reduces the search space for α, we can go further by taking advantage of the fact that each retraining operation is relatively similar to the previous operations, with a minor change in the value of α. When working with models trained iteratively (i.e., using stochastic gradient descent [30], or SGD), we can take advantage of warm start optimizations to reduce the cost of each successive retraining. Intuitively, as we train on the same data with slightly different objective functions, the feature weights should remain similar across different choices of α.

Thus, in CrossTrainer when optimizing SGD-based classifiers, we save the trained model coefficients for different α during golden section search and re-use them to initialize SGD when training for new values of α. This reduces the number of SGD epochs required to converge.

While this initial warm start implementation is straightforward, we find it provides runtime improvements in practice. More sophisticated warm start techniques for incremental learning [37] and hyperparameter estimation [11] can also be incorporated in future work.

5 EVALUATION

We empirically evaluate CrossTrainer in terms of model accuracy and training time. We demonstrate:

1) CrossTrainer consistently produces models that match or exceed the accuracy of competing methods on real datasets.
2) CrossTrainer's performance is robust over a variety of source and target data distributions.
3) CrossTrainer's optimizations decrease training time while maintaining model accuracy.

5.1 Experimental Setup

Implementation. We implement CrossTrainer in Python. For most experiments, we use scikit-learn's implementation of logistic regression with stochastic gradient descent [27] as the base model. We also evaluate CrossTrainer with gradient boosted decision trees (GBDTs), using the XGBoost package [10]. By default, we use k-fold cross-validation with $k = 5$ to optimize model- and method-specific hyperparameters including the regularization parameter for logistic regression, the number of estimators (trees) for GBDTs, and α for CrossTrainer.

Baselines. We compare against three standard domain adaptation baselines:

(1) *Target*: This baseline trains a single model on only the target data ($\alpha = 1$).
(2) *Source*: This baseline trains a single model on only the source data ($\alpha = 0$).

(3) *All*: This baseline trains on uniformly weighted union of the source and target datasets ($\alpha = \beta$).

Competing Methods. In addition to the baselines, we evaluate the following domain adaptation methods.

(1) *CrossTrainer*: Our implementation of loss reweighting including optimizations. Unless otherwise specified, we use $\delta = 0.01$ throughout the experiments.
(2) *Pred*: This method trains a class-balanced logistic regression model to distinguish between the source and target training examples. The trained model weights each instance of the combined dataset by the predicted probability of the instance belonging to the target dataset.
(3) *Import*: Similar to Pred, this method trains a logistic regression model to distinguish between source and target for instance-specific reweighting. Given a predicted probability $p = \Pr(\text{Target}|x, f(x))$, the instance x is weighted by $w = c/\left(\frac{1}{p} - 1\right)$ where $c = \frac{n_S}{n_T}$, an asymptotically optimal importance weighting for covariate shift [32].
(4) *FeatAug [13]*: This method first augments input features of the source and target data, and trains a model on the combination of the augmented data. Given original features x, source features are represented as $\langle x, x, 0\rangle$ and target features are represented as $\langle x, 0, x\rangle$.
(5) *TrAdaBoost [12]*: This method uses a variant of boosting by iteratively training an ensemble of models with misclassified target instances up-weighted, and misclassified source instances down-weighted at each iteration. Note that TrAdaBoost is only implemented for binary classification tasks.
(6) *CORAL [33]*: This method aligns second-order statistics of the source and target datasets as a pre-processing step before training on the combined datasets. CORAL can also be used for semi-supervised domain adaptation, but we apply it here in a supervised setting.

Datasets. We perform evaluation on 7 real-world datasets and 1 synthetic dataset (Table 1). Of the target data, we set aside 80% for training, and use the remaining 20% as a test set. As CrossTrainer focuses on regimes with very limited target data, we downsample the target training data to assess performance on small target datasets.

In addition to the datasets that are commonly used in DA studies (Newsgroups [2], SRAA [1] and Amazon reviews [8]), we evaluate on the IMDb [3]/Yahoo [4] movie datasets, UCI Gas Sensor Array Drift [16, 38], the UCI HEPMASS [5, 16], and the MNIST [22, 24] datasets. We describe how each dataset is split into target and source in the Appendix A.

We construct the synthetic dataset as follows. Both source and target are sampled from the same Gaussian distribution in 500 dimensions. The labeling function for target data is a zero-one threshold function applied to a linear function of the input features given by $g(x) = \sum_{j=1}^{500} x_j$, plus Gaussian ($\mathcal{N}(0, 1)$) noise. The labeling function for the source data is equivalent except the weights of the linear component are changed to $g(x) = \sum_{j=1}^{500} c_j x_j$, where $c_j \sim \mathcal{N}(1, \sigma)$.

Table 1: Datasets and classification tasks used in the evaluation.

Name	Clf Task	Domain Split	n_S	n_T	d
Newsgroups	Topic	Category	3.5K	3.6K	50K
SRAA	Topic	Category	8K	8K	60K
Amazon	Review	Product	2K	2K	500K
movie	Genre	Site Source	80K	10K	100
gas	Chem	Time	5.9K	4.4K	128
hepmass	Binary	Particle	7M	7M	26
MNIST	Image	Perturbation	70K	60K	784
synthetic	Binary	Label Shift	100K	100K	500

5.2 Empirical Evaluation of DA Methods

Comparison of Target Accuracy. In Table 2, we report the target accuracy of different domain adaption methods and baselines under a total of eighteen settings. We summarize the key observations below.

First, simple baselines are surprisingly difficult to beat; CrossTrainer is the only method that consistently outperforms all baselines. For example, when source-only outperforms target-only (e.g., Amazon (e $\rightarrow$ k) with $n_T = 500$), Pred, Import, FeatAug, TrAdaBoost, and CORAL all perform worse than simply training on the union of the source and target data. However, since loss reweighting tunes α using the validation set, it is able to adapt to circumstances where the source data is more or less useful compared to the target.

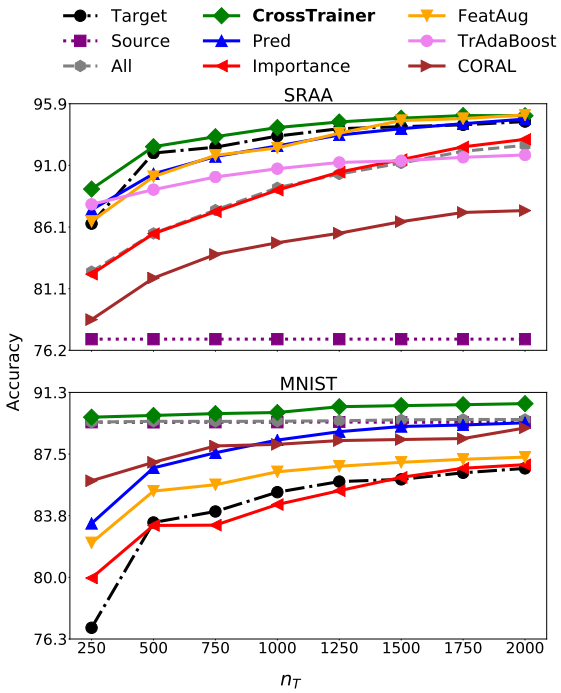
Second, CrossTrainer works across different types of domain splits. For datasets such as MNIST, the distribution of labels $p(y)$, $p(y|x)$ is similar between target and source, while the distribution of the covariates $p(x)$ is different as the infinite MNIST [24] (source) features are generated from deformations and translations of the MNIST (target) features. For datasets like Synthetic, the covariate distributions are equivalent with the difference coming from the labeling distribution $p(y|x)$. In both situations, CrossTrainer significantly outperforms other methods.

As a whole, CrossTrainer using loss reweighting is able to produce high-quality models, and achieves the best target accuracy in 15 out of the 18 evaluated scenarios. To better understand the robustness and generality of these results, we further vary a number of parameters in our experiments including dataset size, the discrepancy between source and target, and the types of models.

Table 2: Comparison of output model accuracy. CrossTrainer outperforms related methods on most datasets and always performs at least as well as the baselines. We report the performance of loss reweighting with no optimizations along with CrossTrainer. Bold entries achieved accuracy within 0.1% of the best method and starred entries did not underperform any of the baselines.

Dataset	n_T	Target	Source	All	CrossTrainer (unopt.)	Pred	Import	FeatAug	TrAdaBoost	CORAL
Newsgroups	500	90.1	68.6	88.9	**91.9 (91.7)***	90.4*	89.3	90.3*	85.3	70.1
Newsgroups	1000	93.5	68.6	92.3	**95.3 (95.2)***	94.0*	93.3	93.8*	89.1	74.1
SRAA	500	91.6	77.1	85.8	**92.5 (92.5)***	90.3	85.5	90.1	89.1	63.7
SRAA	1000	93.3	77.1	89.1	**94.0 (93.9)***	92.6	89.0	92.4	90.7	69.2
Amazon (b → e)	500	79.2	70.8	78.6	79.2 (79.2)*	**80.3***	**80.3***	78.6	77.6	74.3
Amazon (b → e)	1000	82.4	70.8	**83.7**	**83.7 (83.7)***	82.7	83.0	83.0	82.1	75.8
Amazon (e → k)	500	80.9	84.4	**86.8**	**86.8 (86.9)***	84.2	85.0	83.2	85.5	79.6
Amazon (e → k)	1000	85.7	84.4	88.6	**89.5 (89.5)***	87.4	87.8	87.6	87.6	81.1
movie	500	85.1	79.7	85.0	85.2 (85.1)*	86.3*	85.9*	**87.2***	86.2*	82.1
movie	1000	**88.8**	79.7	87.0	**88.9 (88.9)***	87.0	85.7	88.6	86.6	86.1
gas	500	**98.4**	81.4	96.7	**98.5 (98.5)***	98.1	87.3	98.0	–	79.3
gas	1000	**98.7**	81.4	97.6	**98.7 (98.7)***	98.2	90.1	98.6	–	83.9
hepmass	500	89.2	89.1	89.1	89.7 (89.7)*	89.8*	89.1	88.8	**89.9***	**90***
hepmass	1000	**89.9**	89.1	89.1	**90.0 (90.0)***	89.7	89.1	88.8	**89.9***	**90***
MNIST	500	83.3	89.5	89.5	**89.9 (89.9)***	86.7	80.4	85.2	–	75.1
MNIST	1000	85.2	89.5	89.5	**90.0 (90.0)***	88.4	83.9	86.5	–	77.2
synthetic	500	72.8	84.2	84.5	**85.7 (85.7)***	74.0	75.6	73.2	76.1	82.6
synthetic	1000	79.8	84.2	84.8	**86.3 (86.3)***	80.1	81.2	80.2	70.0	84

Figure 5: Comparison of target accuracy over varying sizes of the target dataset for SRAA (top) and MNIST (bottom).



Robustness to changes to the target dataset size. In Figure 5, we evaluate the effect of target dataset sizes on the performance of different domain adaption methods on the SRAA and MNIST datasets. Across both datasets and all methods, performance tends to increase with more training data from the target domain. For the SRAA dataset, with small amounts of target data (Figure 5, SRAA, $n_T < 1250$), all domain adaptation methods except for CrossTrainer perform worse than the target-only baseline. On the other hand, we observe that source data helps improve model performances over the target-only baseline for most methods on the MNIST dataset. However, with small amounts of target data (Figure 5, MNIST, $n_T < 1500$), all methods except for CrossTrainer perform worse than the source-only or all baselines. For both datasets, CrossTrainer consistently outperforms baselines and achieves the best accuracy across the board.

Robustness to source distribution change. In Figure 6, we evaluate CrossTrainer and other DA methods subject to changes to the source distribution on the synthetic dataset. We vary σ, the parameter that defines the difference between the target and source labeling distributions. Larger values of σ indicate that the source distribution is further away from the target distribution. When the source distribution is very close to the target distribution, CrossTrainer matches the performance of the source-only model. At the other extreme, when the source distribution is very different from the target distribution, CrossTrainer matches the performance of the

Figure 6: Comparison of output model accuracy over varying the difference between the source and target distributions for Synthetic with $n_T = 500$.

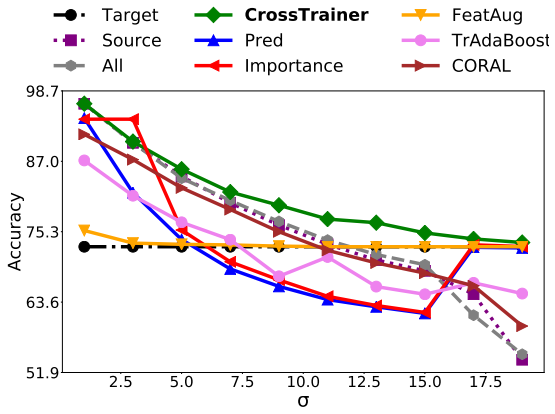


Table 3: Domain adaptation using GBDT models with $n_T = 1000$. Bold entries achieved accuracy within 0.1 of the best method and starred entries did not underperform any of the baselines.

Dataset	Target	Source	All	CrossTrainer	Pred	Imp.	FeatAug
Newsgroups	84.4	62.3	79.5	**85.1***	84.2	81.7	82.6
SRAA	91.7	84.8	90.1	**92.2***	91.7	90.1	92.0*
MNIST	86.4	92.5	**93.4**	**93.4***	**93.3**	**93.3**	92.9

target-only model. Between these extremes, CrossTrainer outperforms all methods except Import when $\sigma = 3$.

Robustness to model type. In Table 3, we compare CrossTrainer to baselines and competing methods using gradient boosted decision trees (GBDTs) for three datasets with $n_T = 1000$. As GBDTs are not trained using SGD, the warm-start optimization does not apply, but CrossTrainer is still able to use golden section search. As with logistic regression, we find that CrossTrainer outperforms competing methods and does not do worse than the baselines.

5.3 Effect of Optimizations

We first evaluate the effectiveness of golden section search as a hyperparameter search strategy. Figure 7 compares the target accuracy achieved by golden section search, grid search and random search after a fixed number of search iterations. We find that across multiple datasets, golden section search converges within at most fifteen iterations. Random search can reach high accuracies very quickly, but it often fails to match the final performance of golden section search even with 100 iterations. Grid search eventually achieves high

Figure 7: Comparison of search strategies for α with $n_T = 1000$. CrossTrainer utilizes golden section search to quickly converge to a value of α that yields high accuracy.

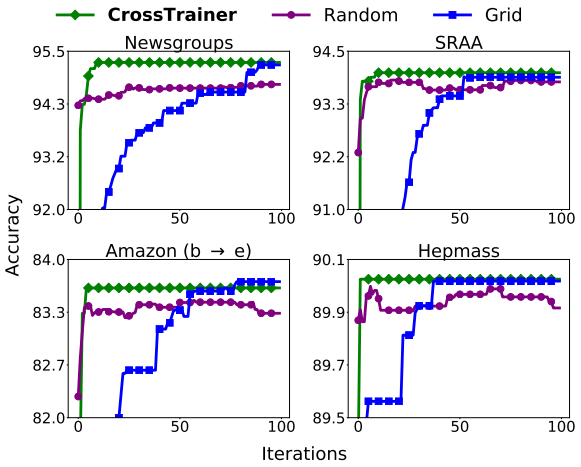


Table 4: Cumulative factor analysis of total training time for loss reweighting with $\delta = 0.01$. CrossTrainer provides speedups between $11-48\times$ over unoptimized loss reweighting.

Dataset	Baseline	+GSS	+Warm-start	Speed-up
Newsgroups	15.5s	2.1s	1.1s	15×
SRAA	7.2s	1.0s	0.7s	11×
Amazon	14.2s	2.5s	1.0s	14×
movie	2.2m	15.8s	5.3s	25×
gas	18.5s	2.6s	0.7s	26×
hepmass	80.1m	8.9m	1.7m	48×
MNIST	31.4m	5.1m	1.7m	19×
synthetic	20.8m	2.2m	38.3s	32×

accuracies, but takes many more iterations to converge than golden section search.

We also evaluate the end-to-end performance gains of our optimizations. Table 4 shows that golden section search and warm-start initialization decrease training time by up 11-48× compared to the baseline (grid search without warm starts). Golden section search consistently provides an order of magnitude improvement in training time over grid search. Warm-starting improves training time in each case, with more of an effect for models that take longer to run, offering up to 5× speedup.

Finally, we validate the soundness of our optimizations by verifying that they do not reduce the final accuracy of the models trained. We compare test set accuracy for CrossTrainer with optimizations enabled with those from a baseline grid

search of 100 values (unopt.) in Table 2 and find that our optimizations have a very small impact on accuracy: the difference is consistently less than the differences between alternative methods.

6 RELATED WORK

We develop a system for domain adaptation, a type of transfer learning where one has different but related data domains, but the same underlying feature space and tasks [6, 13, 26]. We focus on supervised settings for domain adaptation as opposed to the unsupervised case where the learner only has access to an unlabeled target dataset [33].

Different techniques for domain adaptation can be categorized as in [26, 35, 39]:

(1) *Instance-based* methods that integrate source and target instances by assigning a weight to each instance based on its estimated relevance. Examples include TrAdaBoost [12] and loss reweighting [6].
(2) *Feature-based* methods that align features from different domains by either transforming features or by learning common feature structures. Examples include feature augmentation [13] and CORAL [33].
(3) *Parameter-based* methods that share parameters from models or priors between the source and the target domain, including many methods from deep transfer learning. Examples include [18, 36, 40].

In practice, instance-based methods can be applied as a drop-in wrapper to any training pipeline that supports weighting input data. This makes instance-based methods a natural choice for building a generic system. In contrast, feature-based methods may require modifying feature preprocessing and analysis routines, and parameter-based methods are most effective when developed and tuned for specific model architectures (i.e. deep neural nets).

Thus, in this paper, we primarily compare against instance-based methods as well as a number of representative feature-based methods. The "frustratingly easy" domain adaptation technique in [13] is also designed for ease-of-use and speed, but as acknowledged by the authors, does not provide consistently accurate results in some settings. Loss reweighting is proposed in [6] as a theoretical framework for domain adaptation. However, the authors in [6] do not address the accuracy of their method compared to other techniques or consider runtime and usability.

We draw inspiration from other systems for machine learning with limited training data in other settings including data cleaning and weak supervision [21, 28, 29]. Additional related settings include semi-supervised learning [9], where one has unlabeled source data from the same domain, covariate shift [7], where one has unlabeled target data and the conditional label assignment is known to be the same between domains, and active learning [31], where one can iteratively acquire new target training examples.

7 CONCLUSION

We proposed CrossTrainer and evaluated its performance for accurate, robust, and fast domain adaptation. CrossTrainer utilizes loss reweighting, which we find to be consistently effective, and incorporates optimizations to alleviate the overhead of hyperparameter tuning.

Looking forward, potential extensions of CrossTrainer to support additional methods and input types are promising directions for future work. The simplicity of the loss reweighting algorithm suggests that it may be helpful to use reweighting simultaneously with other domain adaptation methods, as well as to incorporate cross-validation hyperparameter tuning more deeply into these methods. Extending the evaluation and optimizations to support simultaneous adaptation from multiple sources would further improve usability in scenarios where users have many potential source domains to draw from.

ACKNOWLEDGMENTS

We thank the members of the Stanford InfoLab as well as Sanjay Krishnan for valuable feedback. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Intel, Microsoft, NEC, SAP, Teradata, and VMware—as well as Toyota Research Institute, Keysight Technologies, Northrop Grumman, Hitachi, and the NSF Graduate Research Fellowship grant DGE-1656518.

A DATASET TASKS

DA Datasets. We evaluate DA methods on three datasets commonly used in previous DA studies: Newsgroups [2], SRAA [1] and Amazon [8]. For Newsgroups, we use the *rec vs. talk* domain split. For SRAA, we use the *auto vs. aviation* domain split. For Amazon, we use the $b \rightarrow e$ and $e \rightarrow k$ domain splits.

Additional Datasets. We evaluate DA methods on four additional real-world datasets: movie (IMBb and Yahoo), Gas, HEPMASS and MNIST. The task of the movie dataset is to classify movies as horror or comedy based on plot summaries, with target coming from Yahoo and source coming from IMDb. The Gas dataset is split according to time with target batches 5-6 and source batches 7-10. The HEPMASS dataset is split with the target dataset containing collisions of particles with 1,000 mass and the source dataset containing all other collisions. For the MNIST dataset, we use the original dataset for target and use samples from the infinite MNIST dataset [24] generated via pseudorandom transformations to the original MNIST data for source.

REFERENCES

[1] 1997-2000. SRAA Dataset. https://people.cs.umass.edu/~mccallum/data.html

[2] 2008. Newsgroups Dataset. http://qwone.com/~jason/20Newsgroups/

[3] 2017. IMDb Movie Dataset. ftp://ftp.fu-berlin.de/pub/misc/movies/database/

[4] 2018. Yahoo Movie Dataset. https://webscope.sandbox.yahoo.com/catalog.php?datatype=r

[5] Pierre Baldi, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson. 2016. Parameterized neural networks for high-energy physics. *The European Physical Journal C* 76 (05 2016).

[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine Learning* 79, 1 (01 May 2010), 151–175.

[7] Steffen Bickel, Michael Brückner, and Tobias Scheffer. 2009. Discriminative Learning Under Covariate Shift. *J. Mach. Learn. Res.* 10 (Dec. 2009), 2137–2155.

[8] John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Association for Computational Linguistics, 440–447.

[9] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. 2010. *Semi-Supervised Learning* (1st ed.). The MIT Press.

[10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*. ACM, New York, NY, USA, 785–794.

[11] Bo-Yu Chu, Chia-Hua Ho, Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. 2015. Warm start for parameter selection of linear classifiers. In *KDD*. ACM, 149–158.

[12] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for Transfer Learning. In *ICML*. 193–200.

[13] Hal Daumé, III. 2007. Frustratingly Easy Domain Adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Association for Computational Linguistics, 256–263.

[14] Hal Daumé, III and Daniel Marcu. 2006. Domain Adaptation for Statistical Classifiers. *J. Artif. Int. Res.* 26, 1 (May 2006), 101–126.

[15] J. Deng, W. Dong, R. Socher, L. Li, and and. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*. 248–255.

[16] Dheeru Dua and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[17] A. Halevy, P. Norvig, and F. Pereira. 2009. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems* 24, 2 (March 2009), 8–12.

[18] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. 2013. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 7304–7308.

[19] Jack Kiefer. 1953. Sequential minimax search for a maximum. *Proceedings of the American mathematical society* 4, 3 (1953), 502–506.

[20] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2016. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079* (2016).

[21] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov 1998), 2278–2324.

[23] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18, 1 (Jan. 2017), 6765–6816.

[24] Gaëlle Loosli, Stéphane Canu, and Léon Bottou. 2007. Training Invariant Support Vector Machines using Selective Sampling. In *Large Scale Kernel Machines*, Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston (Eds.). MIT Press, Cambridge, MA., 301–320. http://leon.bottou.org/papers/loosli-canu-bottou-2006

[25] Julian McAuley and Jure Leskovec. 2013. Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text. In *ACM Conference on Recommender Systems (RecSys '13)*. 165–172.

[26] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[28] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* 11, 3 (2017), 269–282.

[29] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1190–1201.

[30] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400–407.

[31] Burr Settles. 2012. Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6, 1 (2012), 1–114.

[32] Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference* 90, 2 (2000), 227 – 244.

[33] Baochen Sun, Jiashi Feng, and Kate Saenko. 2016. Return of Frustratingly Easy Domain Adaptation. In *AAAI (AAAI'16)*. 2058–2065.

[34] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era.. In *ICCV*. 843–852.

[35] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A Survey on Deep Transfer Learning. *CoRR* abs/1808.01974 (2018). arXiv:1808.01974

[36] Tatiana Tommasi, Francesco Orabona, and Barbara Caputo. 2010. Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *CVPR*. 3081–3088.

[37] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. 2014. Incremental and decremental training for linear classification. In *KDD*. 343–352.

[38] Alexander Vergara, Ramón Huerta, Tuba Ayhan, Margaret Ryan, Shankar Vembu, and Margie Homer. 2011. Gas Sensor Drift Mitigation Using Classifier Ensembles. In *SensorKDD*. 16–24.

[39] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big Data* 3, 1 (2016), 9.

[40] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *NIPS*. 3320–3328.