**SAŁAD: A Toolbox for Semi-supervised Adaptive Learning Across Domains**

Anonymous Author(s)

Affiliation

Address

email

**Abstract**

We introduce **saład**, an open source toolbox that provides a unified implementation of state-of-the-art methods for transfer learning, semi-supervised learning and domain adaptation. In the first release, we provide a framework for reproducing, extending and combining research results of the past years, including model architectures, loss functions and training algorithms. The toolbox along with first benchmark results and further resources is accessible at domainadaptation.org.

1 Introduction

Domain Adaptation (DA) is a topic of active research and practical relevance in areas such as computer vision (e.g. [5]), medical imaging (e.g. [20, 2]) and speech processing (e.g. [21]). Classical approaches usually made use of the assumption of covariate shift and the accompanying theoretical works [7, 27, 1, 16]. More recently, the growing popularity of deep learning approaches has lead to novel concepts that deal with feature and signal transformations [23, 9], feature alignment [8, 24] and adversarial training [6].

Changing standards of evaluation and benchmark tasks as well as the accumulation of new algorithms, datasets and model architectures have made it difficult to assess the state of the art. Reference implementations are scattered over software frameworks and often specific to benchmarking setups. As a consequence, comparing the performance of algorithms on new data-sets or with new architectures is difficult and applying existing algorithms on a new application of interest is time-consuming.

To overcome these limitations, we developed **saład**, a library of methods for “Semi-supervised and Adaptive Learning across Domains.” The toolbox bundles popular algorithms and provides both a benchmark setup and easy integration in practical applications. Our main goals are (1) to provide a benchmark utility that provides results for all published\(^1\) approaches, (2) to abstract DA approaches relying on the same theoretical intuitions (e.g., many approaches are based on applying a metric/divergence to the features on source and target domain), (3) to speed up research and discovery of conceptually new DA algorithms, (4) to provide DA algorithms in an easy-to-use form for integration into open source projects.

In this paper, we introduce **saład**, highlight main features and showcase the applicability of the toolbox. We replicate previous results and perform a comprehensive comparative study on multiple adaptation problems (see Fig. 1). Results on different model architectures and datasets are constantly tracked on the project homepage and will be expanded with the introduction of new algorithms.

As backend, **saład** makes use of PyTorch [17], Matplotlib [10], Seaborn [26], Numpy [15], Scipy [11], scikit-learn [18] and Pandas [12]. We also drew inspiration from the Python Optimal Transport Library [4] and reference implementations of the authors cited below.

\(^1\)The present version includes publications from ICLR, NIPS and ICML conferences as well as JMLR.

2 Toolbox Overview

Given one or multiple source domains, DA approaches aim at minimizing the empirical risk $R^i$ on a target domain $T$,

$$\min_{\theta} R_T^i(\theta) = E_{x,y \sim T}[\ell(h_\theta(x), y)]. \tag{1}$$

As labels for the target domain are either scarce or not available, the problem is not directly approachable by standard machine learning loss functions. Three variations of the empirical risk minimization (ERM) problem described above are implemented in salad: unsupervised domain adaptation, domain translation and domain generalization. We outline them below.

The toolbox aims to separate the training loop that is common to all experiments from the actual contribution of each method. It is therefore divided into the main subpackages solver, layers, models and datasets. Selected parts are outlined below. For a full reference, please refer to the appendix or the API documentation.

2.1 Solvers [salad.solver]

Unsupervised Domain Adaptation [salad.solver.DABaseSolver] assumes the presence of a source domain $S$ and a target domain $T$. Given a labeled sample of points from $S$, $\{x_i^s, y_i^s\}_{i=1}^{N_s}$, and an unlabeled sample from $T$, $\{x_i^t\}_{i=1}^{N_t}$, unsupervised adaptation aims at minimizing the risk leveraging an unsupervised loss term $R_{S \times T}^s(\theta)$ that depends on feature representations $f_\theta(x^s)$ and $f_\theta(x^t)$, classifier labels $h_\theta(x^s), h_\theta(x^t)$ and source labels $y^s$. Furthermore, the model is split into a feature extractor $f$ and classifier $g$, composing the full model as $h = g \circ f$. Losses are derived by calling these functions. If needed for domain-conditional computations, the domain can be passed to the model as additional information.

Domain Translation [salad.solver.DTBaseSolver] is mostly applied in the unsupervised domain adaptation setting. The general idea is to learn a function that maps between both domains, possibly in form of a probability distribution $p(x^s|x^t)$ and/or $p(x^t|x^s)$, s.t. $l(x^s) = l(x^t)$.

Popular approaches rely on adversarial training and consistency losses. By translating samples into the target domain (preserving semantics), one can obtain a hypothesis $h$ on the target by minimizing the loss $E_{x^s,y^s \sim S}E_{x^t \sim P_t}(x^t)\ell(h(x^t), y^t)$ along with a joint loss on the classifier. Also, one can train a classifier on the source domain and then use the translation function during inference.

Domain Generalization [salad.solver.DGBaseSolver] In contrast to unsupervised domain adaptation, multiple training domains are available in the setting of domain generalization. Any unsupervised domain adaptation algorithm can be regarded as primitive form of a domain generalization algorithm: Inference can be regarded as the process of adapting to the new, previously unseen domain and the label process after training has converged. The obvious drawbacks of this approach are long inference times and the need for many target examples for adaptation. Therefore, going beyond unsupervised domain adaptation in the generalization problem is analogous “generalizing” classification into few-shot learning: Given a few samples from the target domain during training, we expect the network to perform well on these samples.

Learning to generalize requires a dataset of training examples $\{x_i, y_i, d_i\}_{i=1}^{N}$ with class and domain labels. The domains present at training time should reflect the kind of variability that can be expected during inference. The ERM problem is then approached as

$$\min_{\theta} \sum_d R_{S_d}^i(\theta) = \sum_d \lambda_d E_{x,y,d} \ell(f_\theta(x), h_\theta(x), y, d). \tag{3}$$

In contrast to the unsupervised setting, samples are now presented in a single batch comprised of inputs $x$, labels $y$ and domains $d$. In a addition to feature extractor and classifier, models should...
function **UNSUPERVISED DA**
\[\begin{align*}
  x^s, y^s & \sim S \\
  x^t & \sim T \\
  \mathcal{R}_l^l(\theta) & \leftarrow \ell_l(x^s, y^s, \theta) \\
  \mathcal{R}_u^l(\theta) & \leftarrow \ell_l(x^s, y^s, x^t, \theta) \\
  \text{for all optimizers and params } \theta_i & \text{ do} \\
  & \Delta \theta_i \leftarrow \nabla_{\theta_i} (\mathcal{R}_l^l(\theta) + \lambda \mathcal{R}_u^l(\theta)) \\
  & \text{update } \theta_i \text{ (e.g., using ADAM)} \\
  \text{end for}
\end{align*}\]
end function

function **DOMAIN GENERALIZATION**
\[\begin{align*}
  x, y, d & \sim S \\
  \mathcal{R}_l^d(\theta) & \leftarrow \ell_d(x, y, d) \\
  \text{for all optimizers and params } \theta_i & \text{ do} \\
  & \Delta \theta_i \leftarrow \nabla_{\theta_i} \mathcal{R}_l^d(\theta) \\
  & \text{update } \theta_i \text{ (e.g., using ADAM)} \\
  \text{end for}
\end{align*}\]
end function

Figure 1: Left: Pseudocode for Unsupervised DA and Domain Generalization implementations in salad. Right: Evaluation results on different DA benchmark tasks.

also provide a feature extractor for domain features along with a classifier of the domain. Note that parameters of these functions might be shared.

### 2.2 Models & Adaptation Tools [salad.layers, salad.models]

The toolbox provides a variety of network layers necessary for running adaptation algorithms, i.e., for separating statistics of the source and target domain. In addition, for “warm-starting” finetuning and applying models to a novel domain, it is generally useful to adapt network parameters depending on dataset statistics (i.e., instance normalization and batch normalization layers). For this, the subpackage provides additional tools to quickly re-calculate statistics of a model given a new dataset.

### 2.3 Datasets, Noise Models and Pertubations [salad.datasets]

For techniques that incorporate self-labeling, it is crucial to augment the task with a noise model to artificially create additional data domains or enforce consistency constraints. This process can be implemented in different ways. In the current implementation of salad, two main differences exist:

Perturbation of the input image by means of a noise distribution
\[p_{\tilde{x}|x}(\tilde{x}|x, n) = p_{\tilde{x}|x}(\phi_n(x)|x, n)\]
where parameters of the noise distribution are drawn from a distribution \(q(n)\).

Image augmentation of images are mostly straightforward to implement. On top of PyTorch transforms, salad offers differentiable image perturbations operating on image tensors directly. For tasks such as semantic and instance segmentation, the package also provides both benchmarking datasets and transformations that jointly operate in pixel and coordinate space.

### 3 Discussion and Outlook

We presented the salad toolbox and an accompanying website to with the goal of unifying domain adaptation approaches, fostering reproducibility, fair comparisons and application of modern domain adaptation algorithms. We invite researchers and software engineers to collaborate with us in extending the toolbox and offering implementations for other frameworks such as Tensorflow. In future work, we will extend the toolbox with new algorithms, especially translation-based approaches.
References


Appendix

Algorithms [salad.solver]

<table>
<thead>
<tr>
<th>Proxy Labels</th>
<th>[ ]</th>
<th>EnsemblingSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self Ensembling</td>
<td>[5]</td>
<td></td>
</tr>
<tr>
<td>Virtual Adversarial Domain Adaptation (VADA)</td>
<td>[22]</td>
<td>VADASolver</td>
</tr>
<tr>
<td>Adversarial Dropout Regularization</td>
<td>[19]</td>
<td>AdvDropoutSolver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correlation Distance Based</th>
<th>[ ]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep CORAL</td>
<td>[23]</td>
<td>CoralSolver</td>
</tr>
<tr>
<td>Log CORAL</td>
<td>[13]</td>
<td>LogCoralSolver</td>
</tr>
<tr>
<td>Domain Adversarial Training</td>
<td>[6]</td>
<td>DANNSolver</td>
</tr>
<tr>
<td>Associative Domain Adaptation</td>
<td>[8]</td>
<td>AssociationSolver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Translation Based</th>
<th>[ ]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CORAL</td>
<td>[24]</td>
<td></td>
</tr>
<tr>
<td>Cyclic Domain Adaptation</td>
<td>[9]</td>
<td>CyclicDASolver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain Generalization</th>
<th>[ ]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Gradient Training</td>
<td>[21]</td>
<td>CrossGradSolver</td>
</tr>
</tbody>
</table>

Finetuning                                                                 | [ ] |                  |
| DIRT-T                                                                       | [22]| DIRTTSolver      |

Datasets [salad.datasets, salad.transforms]

- Toy Tasks (e.g., Moon Dataset)
- Standard Benchmarks: MNIST, SVHN, UPSP, SYNTH
- Noise Benchmarks (e.g. Gaussian, Salt and Pepper, Rotations)
- MNIST/SVHN Semantic Segmentation Task
- VisDA classification challenge 2017
- VisDA detection and openset challenge 2018

Experiments [salad.examples]

Using salad, we provide reference implementations of recently published domain adaptation algorithms.

- Return of Frustratingly Easy Domain Adaptation [23]
- Self Ensembling for Visual Domain Adaptation [5]
- Few-Shot Adversarial Domain Adaptation [14]
- A DIRT-T Approach to Unsupervised Domain Adaptation [22]
- Domain-Adversarial Training of Neural Networks [6]
- Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results [25]
- Associative Domain Adaptation [8]
- Generalizing Across Domains Via Cross-Gradient Training [21]

Network Layers and Loss Functions [salad.layers]

The aforementioned algorithms usually use novel metrics and training schemes. In order to encourage development of new algorithms for particular problems, salad makes it easy to re-use parts of existing algorithms. Among other functions, we implement the following loss functions:

- Distance Functions between covariate matrices (CORAL, Deep CORAL, Log CORAL)
- Visit and Walker losses for Associative Domain Adaptation
Virtual Adversarial Training

Confidence Weighted Cross Entropy

Conditional Entropy

and the following network layers:

Conditional Batch Normalization

Feature Aware Normalization

AutoDIAL

We provide model implementations for conditional training similar to the PyTorch ResNet implementation.

New Training Settings

We propose a couple of new training settings for adaptation between digit datasets.

Open Set Classification The set of possible classes is not fixed. There is one additional class, the open set.
The classes in the open set are different between source and target domains.

Instance Segmentation MNIST digits are scattered to a larger image; for SVHN, we use the normal images. On top of the classification loss, we are now also faced with the problem of adapting features for a good localization.

Remarks

salad is distributed under the terms of the Mozilla Public Licence 2.0 (MPL-2.0). For more details on the algorithms, the implementation at salad.domainadaptation.org provides the necessary details.