Domain-Adversarial Neural Networks

Hana Ajakan[1], **Pascal Germain**[2], **Hugo Larochelle**[3], **François Laviolette**[2], **Mario Marchand**[2]

[1,2] Département d'informatique et de génie logiciel, Université Laval, Québec, Canada
[3] Département d'informatique, Université de Sherbrooke, Québec, Canada

[1] hana.ajakan.1@ulaval.ca, [2] firstname.lastname@ift.ulaval.ca,
[3] hugo.larochelle@usherbrooke.ca

Abstract

We introduce a new neural network learning algorithm suited to the context of domain adaptation, in which data at training and test time come from similar but different distributions. Our algorithm is inspired by theory on domain adaptation suggesting that, for effective domain transfer to be achieved, predictions must be made based on a data representation that cannot discriminate between the training (source) and test (target) domains. We propose a training objective that implements this idea in the context of a neural network, whose hidden layer is trained to be predictive of the classification target, but uninformative as to the domain of the input. Our experiments on a sentiment analysis classification benchmark, where the target data available at the training time is unlabeled, show that our neural network for domain adaption algorithm has better performance than either a standard neural networks and a SVM, trained on input features extracted with the state-of-the-art marginalized stacked denoising autoencoders of Chen et al. (2012).

1 Introduction

The cost of generating labeled data for a new machine learning task is often an obstacle for applying machine learning methods. There is thus great incentive to developing ways of exploiting data from one problem to generalize to another. Domain adaptation focuses on the situation where we have data generated from two different, but somehow similar distributions. One example is in the context of sentiment analysis in written reviews, where we might want to distinguish between positive from negative ones. While we might have labeled data for reviews of one type of product (*e.g.*, movies), we might want to be able to generalize to reviews of other products (*e.g.*, books). Domain adaptation achieves such transfer by exploiting an extra set of unlabeled training data, for the new problem to which we wish to generalize (*e.g.*, unlabeled reviews of books).

One of the main approach to achieve such transfer is to learn not just a classifier, but also the representation of the data, in such a way as to favour transfer. A large body of work exists on jointly training a classifier and a representation that are both linear [1, 2, 3]. However, recent research has shown that non-linear neural networks can also be successful [4]. Specifically, a variant of the denoising autoencoder [5], known as marginalized stacked denoising autoencoders (mSDA) [6], have demonstrated state-of-the-art performance on this problem. By learning a representation that is robust to input corruption noise, they have been shown to learn a representation that is also more stable across changes of domain and can thus allow cross-domain transfer.

In this paper, we propose to encourage stability of representation between domains explicitly into the learning algorithm of a neural network. This approach is motivated by theory on domain adaptation [7, 8], that suggests that a good representation for cross-domain transfer is one that is indiscriminate of the domain of origin of the input observation. We show that this principle can be implemented into a neural network learning objective that includes a term where the network's hidden layer is working adversarially towards output connections predicting domain membership. The

neural network is then simply trained by gradient-descent learning on this objective. The success of this domain-adversarial neural network (DANN) is confirmed by extensive experiments on a sentiment analysis classification benchmark, showing that better performances are achieved compared to a regular neural network and an SVM. Moreover, by training these models on top of the representation of mSDA, our experiments also confirm that minimizing domain discriminability explicitly is better than only relying on representations that are robust to noise.

2 Domain Adaptation

We consider binary classification tasks where $\mathcal{X} \subseteq \mathbb{R}^n$ is the input space and $\mathcal{Y}=\{0,1\}$ is the label set. We have two different distributions over $\mathcal{X} \times \mathcal{Y}$ called the *source domain* $\mathcal{D}_{\mathrm{S}}$ and the *target domain* $\mathcal{D}_{\mathrm{T}}$. A *domain adaptation* learning algorithm is then provided with a *labeled source sample* $S = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^m$ drawn *i.i.d.* from $\mathcal{D}_{\mathrm{S}}$, and an *unlabeled target sample* $T = \{\mathbf{x}_i^t\}_{i=1}^m$ drawn *i.i.d.* from $\mathcal{D}_{\mathrm{T}}$.[1] The goal of the learning algorithm is to build a classifier $\eta : \mathcal{X} \to \mathcal{Y}$ with a low *target risk* $R_{\mathcal{D}_{\mathrm{T}}}(\eta) \stackrel{\text{def}}{=} \Pr_{(\mathbf{x}^t, y^t)\sim\mathcal{D}_{\mathrm{T}}}[\eta(\mathbf{x}^t) \neq y^t]$, while having no information about the labels on $\mathcal{D}_{\mathrm{T}}$.

To tackle this challenging task, many domain adaptation approaches bound the target error by the sum of the source error and a notion of distance between the source and the target. These methods are intuitively justified by a simple assumption: the source risk is expected to be a good indicator of the target risk when both distributions are similar. Several notions of distance have been proposed for domain adaptation [2, 7, 8, 9, 10]. In this paper, we focus on the $\mathcal{H}$-divergence used by Ben-David et al. [7, 8] (and based on the earlier work of Kifer et al. [11]), defined below.

Definition 1 ([7, 8, 11])**.** Given two domain distributions $\mathcal{D}_{\mathrm{S}}$ and $\mathcal{D}_{\mathrm{T}}$ over $\mathcal{X}$, and a hypothesis class $\mathcal{H}$, the *$\mathcal{H}$-divergence* between $\mathcal{D}_{\mathrm{S}}$ and $\mathcal{D}_{\mathrm{T}}$ is

$$d_{\mathcal{H}}(\mathcal{D}_{\mathrm{S}}, \mathcal{D}_{\mathrm{T}}) \stackrel{\text{def}}{=} 2 \sup_{\eta \in \mathcal{H}} \left| \Pr_{\mathbf{x}^s \sim \mathcal{D}_{\mathrm{S}}} \left[\eta(\mathbf{x}^s) = 1\right] - \Pr_{\mathbf{x}^t \sim \mathcal{D}_{\mathrm{T}}} \left[\eta(\mathbf{x}^t) = 1\right] \right| .$$

That is, the $\mathcal{H}$-divergence relies on the capacity of the hypothesis class $\mathcal{H}$ to distinguish between examples generated by $\mathcal{D}_{\mathrm{S}}$ from examples generated by $\mathcal{D}_{\mathrm{T}}$. Ben David et al. [7, 8] proved that, for a symmetric hypothesis class $\mathcal{H}$, one can compute the *empirical $\mathcal{H}$-divergence* between two samples $S \sim (\mathcal{D}_{\mathrm{S}})^m$ and $T \sim (\mathcal{D}_{\mathrm{T}})^m$ by computing

$$\hat{d}_{\mathcal{H}}(S, T) \stackrel{\text{def}}{=} 2 \left(1 - \min_{\eta \in \mathcal{H}} \left[\frac{1}{m} \sum_{i=1}^{m} I[\eta(\mathbf{x}_i^s) = 1] + \frac{1}{m} \sum_{i=1}^{m} I[\eta(\mathbf{x}_i^t) = 0] \right] \right), \tag{1}$$

where $I[a]$ is the indicator function which is 1 if predicate a is true, and 0 otherwise.

Ben David et al. [7, 8] suggested that, even if it is generally hard to compute $\hat{d}_{\mathcal{H}}(S, T)$ exactly (*e.g.*, when $\mathcal{H}$ is the space of linear classifiers on $\mathcal{X}$), we can easily approximate it by running a learning algorithm on the problem of discriminating between source and target examples. To do so, we construct a new dataset $U = \{(\mathbf{x}_i^s, 1)\}_{i=1}^m \cup \{(\mathbf{x}_i^t, 0)\}_{i=1}^m$ where the examples of the source sample are labeled 1 and the examples of the target sample are labeled 0. Then, the risk of the classifier trained on new dataset U approximates the "min" part of Eq. (1). Ben David et al. [7, 8] also showed that $d_{\mathcal{H}}(\mathcal{D}_{\mathrm{S}}, \mathcal{D}_{\mathrm{T}})$ is upper bounded by its empirical estimate $\hat{d}_{\mathcal{H}}(S, T)$ plus a constant complexity term that depends of the *VC-dimension* of $\mathcal{H}$ and the size of samples S and T. By combining this result with a similar bound on the source risk, the following theorem is obtained.

Theorem 2 (Ben David et al., 2006 [7])**.** *Let $\mathcal{H}$ be a hypothesis class of VC-dimension d. With probability $1-\delta$ over the choice of samples $S \sim (\mathcal{D}_{\mathrm{S}})^m$ and $T \sim (\mathcal{D}_{\mathrm{T}})^m$, for every $\eta \in \mathcal{H}$:*

$$R_{\mathcal{D}_{\mathrm{T}}}(\eta) \leq R_S(\eta) + \frac{4}{m} \sqrt{d \log \tfrac{2e\,m}{d} + \log \tfrac{4}{\delta}} + \hat{d}_{\mathcal{H}}(S, T) + \frac{4}{m^2} \sqrt{d \log \tfrac{2\,m}{d} + \log \tfrac{4}{\delta}} + \beta ,$$

with $\beta \geq \inf_{\eta^* \in \mathcal{H}} [R_{\mathcal{D}_{\mathrm{S}}}(\eta^*) + R_{\mathcal{D}_{\mathrm{T}}}(\eta^*)]$, and $R_S(\eta) = \frac{1}{m} \sum_{i=1}^{m} I[\eta(\mathbf{x}_i^s) \neq y_i^s]$ is the empirical source risk.

[1] For simplicity, we consider through this paper that the source sample S and the target sample T are of equal size m. It is easy to generalize the results for the case $|S| \neq |T|$.

The previous result tells us that $R_{\mathcal{D}_\mathrm{T}}(\eta)$ can be low only when the β term is low (*i.e.*, only when there exists a classifier that can achieve a low risk on both distributions). It also tells us that, to find a classifier with a small $R_{\mathcal{D}_\mathrm{T}}(\eta)$ in a given class of fixed VC dimension, the learning algorithm should minimize (in that class) a trade-off between the source risk $R_S(\eta)$ and the $\mathcal{H}$-divergence $\hat{d}_\mathcal{H}(S,T)$. As pointed-out by Ben David et al. [7], a strategy to control the $\mathcal{H}$-divergence is to find a representation of the examples where both the source and the target domain are as indistinguishable as possible. Under such a representation, a hypothesis with a low source risk will, according to Theorem 2, perform well on the target data. We now present a learning algorithm based on this idea.

3 A Domain-Adversarial Neural Network (DANN)

The originality of our approach is to explicitly implement the idea exhibited by Theorem 2 into a neural network classifier (note that the HMM representation learning method for domain adaptation of Huang and Yates (2012) [12] is also inspired by the $\mathcal{H}$-divergence of Ben-David et al.). That is, to learn a model that can generalize well from one domain to another, we ensure that the internal representation of the neural network contains no discriminative information about the origin of the input (source or target), while preserving a low risk on the source (labeled) examples.

Let us consider the following standard neural network architecture with one hidden layer:

$$\mathbf{h}(\mathbf{x}) \;=\; \mathrm{sigm}(\mathbf{b}+\mathbf{W}\mathbf{x})\,, \qquad \text{and} \qquad \mathbf{f}(\mathbf{x}) \;=\; \mathrm{softmax}(\mathbf{c}+\mathbf{V}\mathbf{h}(\mathbf{x}))\,, \tag{2}$$

with $\mathrm{sigm}(\mathbf{a}) \stackrel{\text{def}}{=} \left[\frac{1}{1+\exp(-a_i)}\right]_{i=1}^{|\mathbf{a}|}$, and $\mathrm{softmax}(\mathbf{a}) \stackrel{\text{def}}{=} \left[\frac{\exp(a_i)}{\sum_{j=1}^{|\mathbf{a}|}\exp(a_j)}\right]_{i=1}^{|\mathbf{a}|}$.

Given a training source sample $S=\{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^m \sim (\mathcal{D}_\mathrm{S})^m$, the natural classification loss to use is the negative log-probability of the correct label. This leads to the following learning problem:

$$\min_{\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c}} \left[\tfrac{1}{m}\textstyle\sum_{i=1}^m -\log(f_{y_i^s}(\mathbf{x}_i^s))\right], \tag{3}$$

where $f_y(\mathbf{x})$ denotes the conditional probability that the neural network assigns $\mathbf{x}$ to class y.

Given a $\mathbf{W}$ and $\mathbf{b}$ obtained by solving Eq. (3), we view the output of the hidden layer $\mathbf{h}(\cdot)$ (Eq. (2)) as the internal representation of the neural network. We denote the source sample representations $\mathbf{h}(S) = \{\mathbf{h}(\mathbf{x}_i^s)\}_{i=1}^m$. Now, consider an unlabeled sample $T=\{\mathbf{x}_i^t\}_{i=1}^m \sim (\mathcal{D}_\mathrm{T})^m$ from the target domain, and the corresponding representations $\mathbf{h}(T) = \{\mathbf{h}(\mathbf{x}_i^t)\}_{i=1}^m$. Based on Eq. (1), the empirical $\mathcal{H}$-divergence of a symmetric hypothesis class $\mathcal{H}$ between samples $\mathbf{h}(S)$ and $\mathbf{h}(T)$ is given by

$$\hat{d}_\mathcal{H}(\mathbf{h}(S),\mathbf{h}(T)) = 2\left(1-\min_{\eta\in\mathcal{H}}\left[\frac{1}{m}\sum_{i=1}^m I\big[\eta(\mathbf{h}(\mathbf{x}_i^s))=1\big]+\frac{1}{m}\sum_{i=1}^m I\big[\eta(\mathbf{h}(\mathbf{x}_i^t))=0\big]\right]\right). \tag{4}$$

Let us consider $\mathcal{H}$ as the class of hyperplanes in the representation space. We suggest estimating the "min" part of Eq. (4) by a logistic regressor that model the probability that a given input (either $\mathbf{x}^s$ or $\mathbf{x}^t$) is from the source domain $\mathcal{D}_\mathrm{S}$ (denoted $z=1$) or the target domain $\mathcal{D}_\mathrm{T}$ (denoted $z=0$):

$$p(z=1 \mid \phi) \;=\; o(\phi) \;\stackrel{\text{def}}{=}\; \mathrm{sigm}(d+\mathbf{w}^\top\phi)\,,$$

where ϕ is either an output from $\mathbf{h}(\mathbf{x}^s)$ or from $\mathbf{h}(\mathbf{x}^t)$. Then, this enables us to add a domain adaptation term to the objective of Eq. (3), giving the following problem to solve:

$$\min_{\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c}}\left[\frac{1}{m}\sum_{i=1}^m -\log\big(f_{y_i^s}(\mathbf{x}_i^s)\big)+\lambda\max_{\mathbf{w},d}\left(\frac{1}{m}\sum_{i=1}^m \log\big(o(\mathbf{h}(\mathbf{x}_i^s))\big)+\frac{1}{m}\sum_{i=1}^m \log\big(1-o(\mathbf{h}(\mathbf{x}_i^t))\big)\right)\right], \tag{5}$$

where the parameter $\lambda > 0$ weights the domain adaptation regularization term. This optimization problem is motivated by Theorem 2, as it implements a trade-off between the minimization of the source risk $R_S(\cdot)$ and the divergence $\hat{d}_\mathcal{H}(\cdot,\cdot)$. We introduced the parameter λ to tune the trade-off between these two quantities during the learning process.

We see that Eq. (5) involves a maximization operation. Hence, the neural network (parametrized by $\{\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c}\}$) and the domain classifier (parametrized by $\{\mathbf{w},d\}$) are competing against each other, in an adversarial way, for that term. In other words, the hidden layer (given by $\mathbf{h}(\cdot)$) maps an example (either source or target) into a representation in which the output layer (given by $\mathbf{f}(\cdot)$) accurately classifies the source sample while the adaptation component (given by $o(\cdot)$) is unable to detect if an example belongs to the source sample or the target sample. To optimize Eq. (5), we perform stochastic gradient descent, as detailed in Appendix A.

Table 1: Error rates on the Amazon reviews dataset (left), and Pairwise Poisson binomial test (right).

	Original data			mSDA representations		
Name	DANN	NN	SVM	DANN	NN	SVM
books → dvd	0.201	**0.199**	0.206	0.176	**0.171**	0.175
books → electronics	**0.246**	0.251	0.256	**0.197**	0.228	0.244
books → kitchen	0.230	0.235	**0.229**	0.169	**0.166**	0.172
dvd → books	**0.247**	0.261	0.269	0.176	**0.173**	0.176
dvd → electronics	**0.247**	0.256	0.249	**0.181**	0.234	0.220
dvd → kitchen	**0.227**	**0.227**	0.233	**0.151**	0.153	0.178
electronics → books	**0.280**	0.281	0.290	**0.237**	0.241	0.229
electronics → dvd	**0.273**	0.277	0.278	**0.216**	0.228	0.261
electronics → kitchen	**0.148**	0.149	0.163	**0.118**	0.126	0.137
kitchen → books	**0.283**	0.288	0.325	**0.222**	0.226	0.234
kitchen → dvd	0.261	**0.261**	0.274	**0.208**	0.214	0.209
kitchen → electronics	0.161	0.161	**0.158**	0.141	**0.136**	0.138

Original data

	DANN	NN	SVM
DANN	0.50	**0.90**	**0.97**
NN	0.10	0.50	0.87
SVM	0.03	0.13	0.50

mSDA representations

	DANN	NN	SVM
DANN	0.50	**0.82**	**0.88**
NN	0.18	0.50	0.90
SVM	0.12	0.10	0.50

4 Experiments

In this section, we compare the performance of our proposed DANN algorithm to a standard neural network with one hidden layer (NN) described by Eq. (3), and a Support Vector Machine (SVM) with a linear kernel. To select the hyper-parameters of each these algorithms, we train them using a parameter grid, and we use a very small validation set which consists in 100 labeled examples from the target domain. Finally, we select the classifiers having the lowest target validation risk. We detailed the training procedure for each algorithm in Appendix B.

Sentiment analysis dataset. We compare our algorithms on the *Amazon reviews* dataset, as preprocessed by Chen et al. (2012) [6]. This dataset includes four domains, each one composed by reviews of a specific kind of product (books, dvd disks, electronics, and kitchen appliances). Reviews are encoded in $5,000$ dimensional feature vectors of unigrams and bigrams, and labels are binary: "0" if the product is ranked up to 3 stars, and "1" if the product is ranked 4 or 5 stars.

We perform twelve domain adaptation tasks. For example, "books → dvd" corresponds to the task for which books is the source domain and dvd disks the target one. All learning algorithms are given $2,000$ labeled source examples and $2,000$ unlabeled target examples. Then, we evaluate them on separate target test sets (between $3,000$ and $6,000$ examples). The "Original data" part of Table 1 (left) shows the test risk of all algorithms, and Table 1 (right) reports the probability than one algorithm is significantly better than another according to the Poisson binomial test [13]. We note that DANN has a significantly better performance than NN and SVM, with respective probabilities **0.90** and **0.97**. As the only difference between DANN and NN is the DA regularizer, we conclude that our approach successfully helps to find a representation suitable for the target domain.

Combining DANN with autoencoders. We now wonder whether our DANN algorithm can improve on the representation learned by state-of-the-art *Marginalized Stacked Denoising Autoencoders* (mSDA) [6]. In brief, mSDA is an unsupervised algorithm that learn a new robust features representations of the training samples. It takes the unlabeled parts of union of the source set S and the target set T to learn a feature map from input space X to a new representation space X'. As a denoising autoencoders algorithm, it finds a feature representation from which one can (approximately) reconstruct the original features of an example from its noisy counterpart.

Chen et al. (2012) [6] shows that using mSDA with linear SVM classifier performs well on the *Amazon reviews* datasets on the new representation of the source sample. As an alternative to SVM, we propose to apply DANN algorithm on the same representations generated by mSDA (using representations of both source and target samples). Note that, even if mSDA and DANN are two representation learning approaches, they pursued a different strategy that can be complementary. In this experimentation, we generate the mSDA representations using a corruption probability of 50% and a number of layers of 5 for each pair of domains source-target, using the same *amazon reviews* described earlier. We then execute the three learning algorithms (DANN, NN, and SVM) on the top of these representations. The mSDA part of Table 1 (left and right) confirms that combining mSDA and DANN is a sound approach. Indeed, the Poisson binomial test shows that DANN has a better performance than NN and SVM with probabilities **0.82** and **0.88** respectively.

References

[1] L. Bruzzone and M. Marconcini. Domain adaptation problems: A DASVM classification technique and a circular validation strategy. *Transaction Pattern Analysis and Machine Intelligence*, 32(5):770–787, 2010.

[2] P. Germain, A. Habrard, F. Laviolette, and E. Morvant. A PAC-Bayesian approach for domain adaptation with specialization to linear classifiers. In *ICML*, pages 738–746, 2013.

[3] C. Cortes and M. Mohri. Domain adaptation and sample bias correction theory and algorithm for regression. *Theor. Comput. Sci.*, 519:103–126, 2014.

[4] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, volume 27, pages 97–110, 2011.

[5] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103, 2008.

[6] M. Chen, Z. E. Xu, K. Q. Weinberger, and F. Sha. Marginalized denoising autoencoders for domain adaptation. In *ICML*, 2012.

[7] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *NIPS*, pages 137–144, 2006.

[8] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J.W. Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

[9] Y. Mansour, M. Mohri, and A. Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *COLT*, pages 19–30, 2009.

[10] Y. Mansour, M. Mohri, and A. Rostamizadeh. Multiple source adaptation and the Rényi divergence. In *UAI*, pages 367–374, 2009.

[11] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Very Large Data Bases*, 2004.

[12] F. Huang and A. Yates. Biased representation learning for domain adaptation. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1313–1323, 2012.

[13] A. Lacoste, F. Laviolette, and M. Marchand. Bayesian comparison of machine learning algorithms on single and multiple datasets. In *AISTATS*, pages 665–675, 2012.

A Learning Algorithm

Algorithm 1 DANN – Stochastic training update

Input: source sample $S = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^m$, target sample $T = \{\mathbf{x}_i^t\}_{i=1}^{m'}$,
hidden layer size l, adaptation parameter λ, learning rate α.
Output: neural network $\{\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}\}$

$\mathbf{W}, \mathbf{V} \leftarrow \text{random\_init}(\, l \,)$
$\mathbf{b}, \mathbf{c}, \mathbf{w}, d \leftarrow 0$
while stopping criteria is not met **do**
  for i from 1 to m **do**
    # Forward propagation
    $\mathbf{h}(\mathbf{x}_i^s) \leftarrow \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x}_i^s)$
    $\mathbf{f}(\mathbf{x}_i^s) \leftarrow \text{softmax}(\mathbf{c} + \mathbf{V}\mathbf{h}(\mathbf{x}_i^s))$
    # Backpropagation
    $\Delta_\mathbf{c} \leftarrow -(\mathbf{e}(y_i^s) - \mathbf{f}(\mathbf{x}_i^s))$ # $\mathbf{e}(y)$ is a "one-hot" vector, that is all 0s but with a 1 at position y
    $\Delta_\mathbf{V} \leftarrow \Delta_\mathbf{c}\, \mathbf{h}(\mathbf{x}_i^s)^\top$
    $\Delta_\mathbf{b} \leftarrow \left(\mathbf{V}^\top \Delta_\mathbf{c}\right) \odot \mathbf{h}(\mathbf{x}_i^s) \odot (1 - \mathbf{h}(\mathbf{x}_i^s))$ # $\odot$ is the element-wise product
    $\Delta_\mathbf{W} \leftarrow \Delta_\mathbf{b} \cdot (\mathbf{x}_i^s)^\top$
    # Add domain adaptation regularizer ...
    # ... from current domain
    $o(\mathbf{x}_i^s) \leftarrow \text{sigm}(d + \mathbf{w}^\top \mathbf{h}(\mathbf{x}_i^s))$
    $\Delta_d \leftarrow \lambda(1 - o(\mathbf{x}_i^s))$
    $\Delta_\mathbf{w} \leftarrow \lambda(1 - o(\mathbf{x}_i^s))\mathbf{h}(\mathbf{x}_i^s)$
    $\text{tmp} \leftarrow \lambda(1 - o(\mathbf{x}_i^s))\mathbf{w} \odot \mathbf{h}(\mathbf{x}_i^s) \odot (1 - \mathbf{h}(\mathbf{x}_i^s))$
    $\Delta_\mathbf{b} \leftarrow \Delta_\mathbf{b} + \text{tmp}$
    $\Delta_\mathbf{W} \leftarrow \Delta_\mathbf{W} + \text{tmp} \cdot (\mathbf{x}_i^s)^\top$
    # ... from other domain
    $j \leftarrow \text{unform\_integer}(1, \ldots, m')$
    $\mathbf{h}(\mathbf{x}_j^t) \leftarrow \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x}_j^t)$
    $o(\mathbf{x}_j^t) \leftarrow \text{sigm}(d + \mathbf{w}^\top \mathbf{h}(\mathbf{x}_j^t))$
    $\Delta_d \leftarrow \Delta_d - \lambda o(\mathbf{x}_j^t)$
    $\Delta_\mathbf{w} \leftarrow \Delta_\mathbf{w} - \lambda o(\mathbf{x}_j^t)\mathbf{h}(\mathbf{x}_j^t)$
    $\text{tmp} \leftarrow -\lambda o(\mathbf{x}_j^t)\mathbf{w} \odot \mathbf{h}(\mathbf{x}_j^t) \odot (1 - \mathbf{h}(\mathbf{x}_j^t))$
    $\Delta_\mathbf{b} \leftarrow \Delta_\mathbf{b} + \text{tmp}$
    $\Delta_\mathbf{W} \leftarrow \Delta_\mathbf{W} + \text{tmp} \cdot (\mathbf{x}_j^t)^\top$
    # Update parameters neural network parameters
    $\mathbf{W} \leftarrow \mathbf{W} - \alpha\Delta_\mathbf{W}$ # α is a hyper-parameter learning rate
    $\mathbf{V} \leftarrow \mathbf{V} - \alpha\Delta_\mathbf{V}$
    $\mathbf{b} \leftarrow \mathbf{b} - \alpha\Delta_\mathbf{b}$
    $\mathbf{c} \leftarrow \mathbf{c} - \alpha\Delta_\mathbf{c}$
    # Update domain classifier parameters
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\Delta_\mathbf{w}$ # Notice the "+" instead of the "-"
    $d \leftarrow d + \alpha\Delta_d$
  end for
end while

return $\{\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}\}$

B Empirical Experiments Details

Here is some details about the procedure we used to execute the learning parameters.

DANN. The adaptation parameter λ is chosen among 9 values between 10^{-2} and 1 on a logarithmic scale. The hidden layer size l is either 1, 5, 12, 25, 50, 75, 100, 150, or 200. Finally, the learning rate α is fixed at 10^{-3} when learning on "original data" and 10^{-4} when learning on "mSDA representations".

For each learning task, we split the source training set to use 90% as the training set S and the remaining 10% as a validation set S_V. We stop the learning process when the risk on S_V is minimal.

NN. We use exactly the same hyper-parameters and training procedure as DANN above, except that we do not need an adaptation parameter. Note that one can train NN by using DANN implementation (Algorithm 1) with $\lambda = 0$.

SVM. The hyper-parameter C of the SVM is chosen among 10 values between 10^{-5} and 1 on a logarithmic scale. Note that this range of values is the same uded by Chen et al.(2012) [6] in their experimentations.