Domain-Adversarial Neural Networks

published: 2015-02

authors: Universite Laval Canada

> We introduce a new representation 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 directly 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.**

> **A good representation for cross-domain transfer is one for which an algorithm cannot learn to identify the domain of origin of the input observation.**

Domain Adaptation

A domain adaptation learning algorithm is then provided with a labeled source sample S drawn i.i.d from D_S, and an unlabeled target sample T drawn i.i.d from D_T^x, where D_T^x, where D_T^x is the marginal distribution of D_T over x.

$$S = \{(x_i^s, y_i^s)\}_{i=1}^{m}; T = \{x_i^t\}_{i=1}^{m'} \sim (D_T^x)^{m'}$$

The goal of the learning algorithm is to build a classifier $\eta : x \rightarrow y$ with a low target risk:

$$R_{D_T}(\eta) = Pr_{(x^t, y^t) \sim D_T}(\eta(x^t) \neq y^t)$$

Domain Divergence

To tackle the challenging domain adaptation task, many approaches bound the target error by the sum of the source error and a notion of distance between the source and the target distributions. 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.

Definition 1 Given two domain distributions D_S^x and D_T^x over x, and a hypothesis class H, the H-divergence between D_S^x and D_T^x is

$$d_H(D_S^x, D_T^x) = 2\sup_{\eta\in H}|Pr_{x^s\sim D_S^x}[\eta(x^s)=1] - Pr_{x^t\sim D_T^x}[\eta(x^t)=1]|$$

That is, the *H-divergence* relies on the capacity of the hypothesis class H to distinguish between examples generated by D_S^x from examples generated by D_T^x.

empirical H-divergence

$$d_H(S,T) = 2(1-\min_{\eta\in H}[$$

The *H-divergence* defined a worst classificier η.

Theorem 2 Let H be a hypothesis class of VC dimension d. With probability $1-\delta$ over the choice of samples $S\sim(D_S)^m$ and

$T \sim (D_T^x)^m$, for every $\eta \in H$:

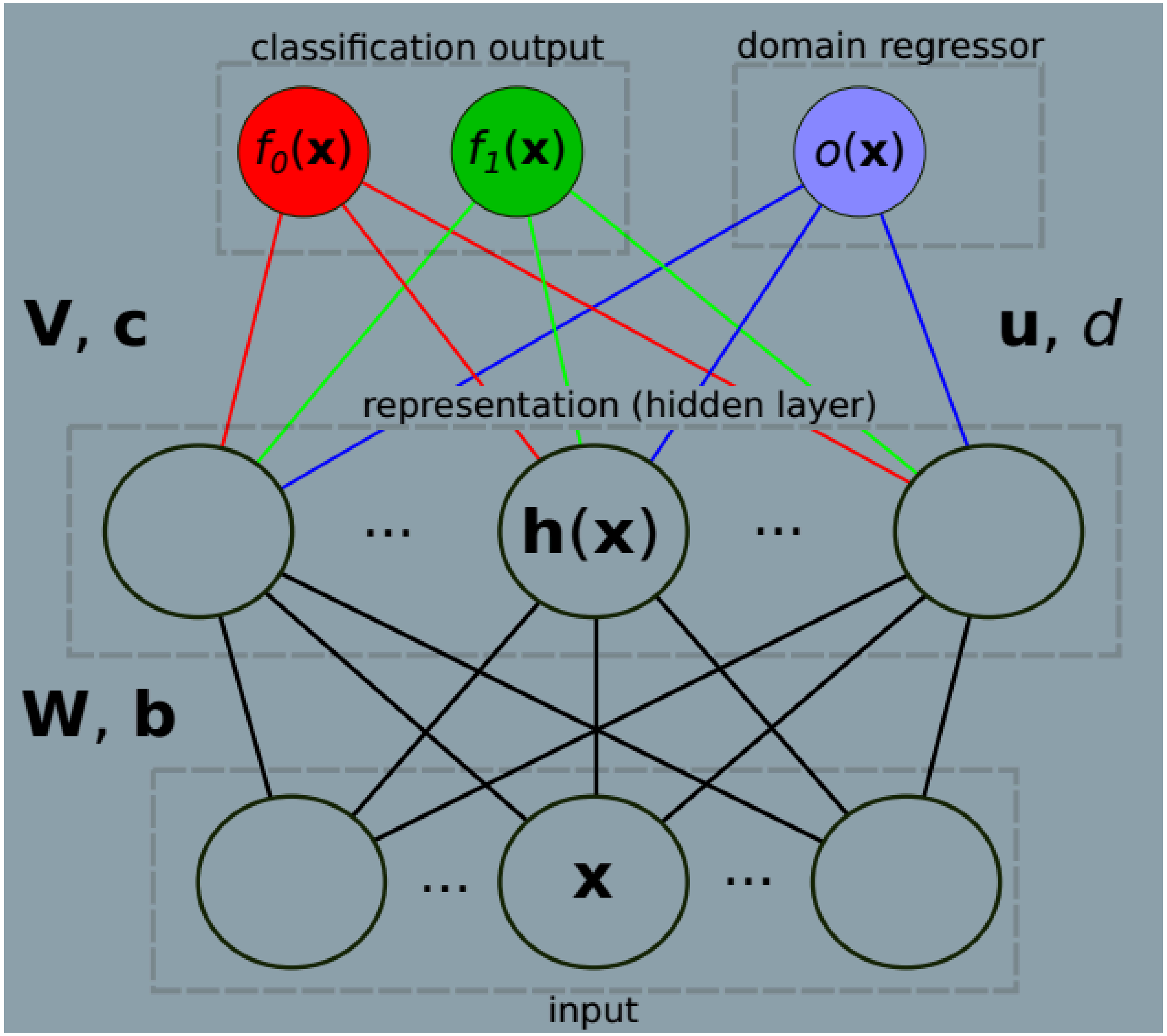
$$R_{D_T}(\eta) \leq R_S(\eta) + d_H(S,T) + ...$$

The empirical source risk

$R_S(\eta) = \frac{1}{m}$

The learning algorithm should minimize a trade-off between the souce risk $R_S(\eta)$ and the empirical H-divergence $d_H(S,T)$. A strategy to control the H-divergence is to find a representation of the examples where both the source and the target domain are as indistinguishable as possible.

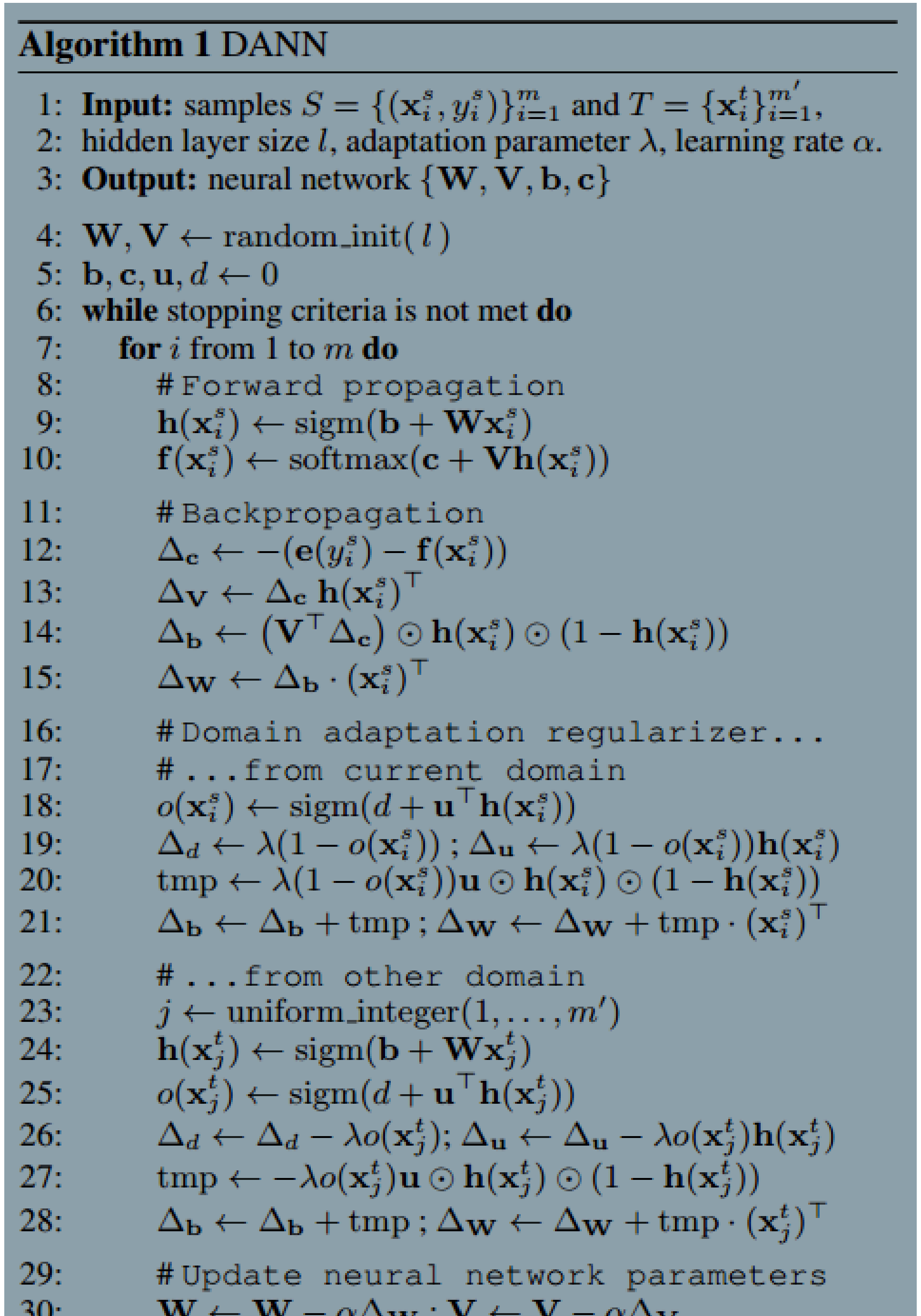
DANN


classification output
domain regressor
$f_0(\mathbf{x})$
$f_1(\mathbf{x})$
$o(\mathbf{x})$
$\mathbf{V}, \mathbf{c}$
$\mathbf{u}, d$
representation (hidden layer)
...
$\mathbf{h}(\mathbf{x})$
...
$\mathbf{W}, \mathbf{b}$
...
$\mathbf{x}$
...
input

In DANN, the hidden layer h(.) maps an example(either source or targt) into a representation in which the output layer f(.) accurately classifies the source sample, while the domain regressor o(.) is unable to detect if an example belongs to the source sample or the target sample.

Crucially, while the update of the regular parameters follows as usual the opposite direction of the gradient, for the adversarial parameters u, d the step must follow the gradient's direction (since we maximize with respect to them, instead of minimizing).

The algorithm:



Algorithm 1 DANN

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

$\mathbf{W}, \mathbf{V} \leftarrow \text{random\_init}(\,l\,)$
$\mathbf{b}, \mathbf{c}, \mathbf{u}, d \leftarrow 0$
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
        $\Delta_{\mathbf{c}} \leftarrow -(\mathbf{e}(y_i^s) - \mathbf{f}(\mathbf{x}_i^s))$
        $\Delta_{\mathbf{V}} \leftarrow \Delta_{\mathbf{c}}\, \mathbf{h}(\mathbf{x}_i^s)^\top$
        $\Delta_{\mathbf{b}} \leftarrow \left(\mathbf{V}^\top \Delta_{\mathbf{c}}\right) \odot \mathbf{h}(\mathbf{x}_i^s) \odot (1 - \mathbf{h}(\mathbf{x}_i^s))$
        $\Delta_{\mathbf{W}} \leftarrow \Delta_{\mathbf{b}} \cdot (\mathbf{x}_i^s)^\top$

        # Domain adaptation regularizer...
        # ...from current domain
        $o(\mathbf{x}_i^s) \leftarrow \text{sigm}(d + \mathbf{u}^\top \mathbf{h}(\mathbf{x}_i^s))$
        $\Delta_d \leftarrow \lambda(1 - o(\mathbf{x}_i^s))\,;\, \Delta_{\mathbf{u}} \leftarrow \lambda(1 - o(\mathbf{x}_i^s))\mathbf{h}(\mathbf{x}_i^s)$
        $\text{tmp} \leftarrow \lambda(1 - o(\mathbf{x}_i^s))\mathbf{u} \odot \mathbf{h}(\mathbf{x}_i^s) \odot (1 - \mathbf{h}(\mathbf{x}_i^s))$
        $\Delta_{\mathbf{b}} \leftarrow \Delta_{\mathbf{b}} + \text{tmp}\,;\, \Delta_{\mathbf{W}} \leftarrow \Delta_{\mathbf{W}} + \text{tmp} \cdot (\mathbf{x}_i^s)^\top$

        # ...from other domain
        $j \leftarrow \text{uniform\_integer}(1, \ldots, m')$
        $\mathbf{h}(\mathbf{x}_j^t) \leftarrow \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x}_j^t)$
        $o(\mathbf{x}_j^t) \leftarrow \text{sigm}(d + \mathbf{u}^\top \mathbf{h}(\mathbf{x}_j^t))$
        $\Delta_d \leftarrow \Delta_d - \lambda o(\mathbf{x}_j^t);\, \Delta_{\mathbf{u}} \leftarrow \Delta_{\mathbf{u}} - \lambda o(\mathbf{x}_j^t)\mathbf{h}(\mathbf{x}_j^t)$
        $\text{tmp} \leftarrow -\lambda o(\mathbf{x}_j^t)\mathbf{u} \odot \mathbf{h}(\mathbf{x}_j^t) \odot (1 - \mathbf{h}(\mathbf{x}_j^t))$
        $\Delta_{\mathbf{b}} \leftarrow \Delta_{\mathbf{b}} + \text{tmp}\,;\, \Delta_{\mathbf{W}} \leftarrow \Delta_{\mathbf{W}} + \text{tmp} \cdot (\mathbf{x}_j^t)^\top$

        # Update neural network parameters
        $\mathbf{W} \leftarrow \mathbf{W} - \alpha\Delta_{\mathbf{W}}\,;\, \mathbf{V} \leftarrow \mathbf{V} - \alpha\Delta_{\mathbf{V}}$
```