Domain-adversarial training of neural networks

Ganin et al., Journal of Machine Learning Research, 2016

Pierre Geurts

Institut Montefiore, University of Liège, Belgium

INFO8004
Advanced Machine Learning
March 7, 2019

Outline

Part 1: Domain adaptation
Part 2: Domain-adversarial training of neural networks

Part I

Domain adaptation

Standard supervised learning setting (again!)

General supervised learning problem:

From a learning sample $LS = \{(x_i, y_i)|i = 1, \ldots, N\}$ *of* N *input-output pairs identically and idependently drawn from a distribution* $p(x, y)$ *over* $\mathcal{X} \times \mathcal{Y}$*, find a function* $f : \mathcal{X} \to \mathcal{Y}$ *that minimizes the expectation of some loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathcal{R}$ *over* $p(x, y)$*:*

$$E_{p(x,y)}\{\ell(f(x), y)\}$$

Ideal settings for several reasons:

- Distribution $p(x, y)$ is assumed to be the **same** at training and prediction times.
- Work (well) only if some (enough) **labeled** training examples are available.

More often than not, at least one of these conditions is not met

General problem: *How to make the best use of all available data, not necessarily labeled and/or drawn from* $p(x, y)$ *to improve, or make possible, learning?*

Domain adaptation

The general goal of **domain adaptation** (DA) is to learn from data obtained from a **source distribution** a model for a task that will perform as well as possible in a different **target distribution**.

This model is learned from **labeled data from the source domain** and **unlabeled/labeled data from the target domain**.

Typically, more data is available in the source domain.

If labeled target data is available, we talk about **supervised DA**, otherwise **unsupervised DA** (the most common setting).

Example: user adaptation

Example: Design a spam filtering for a given user exploiting:

- Unlabeled/Labeled emails from the user (target data)
- Labeled emails from other users (source data)

In general, any problem where a model needs to be adapted to new users. Eg.:

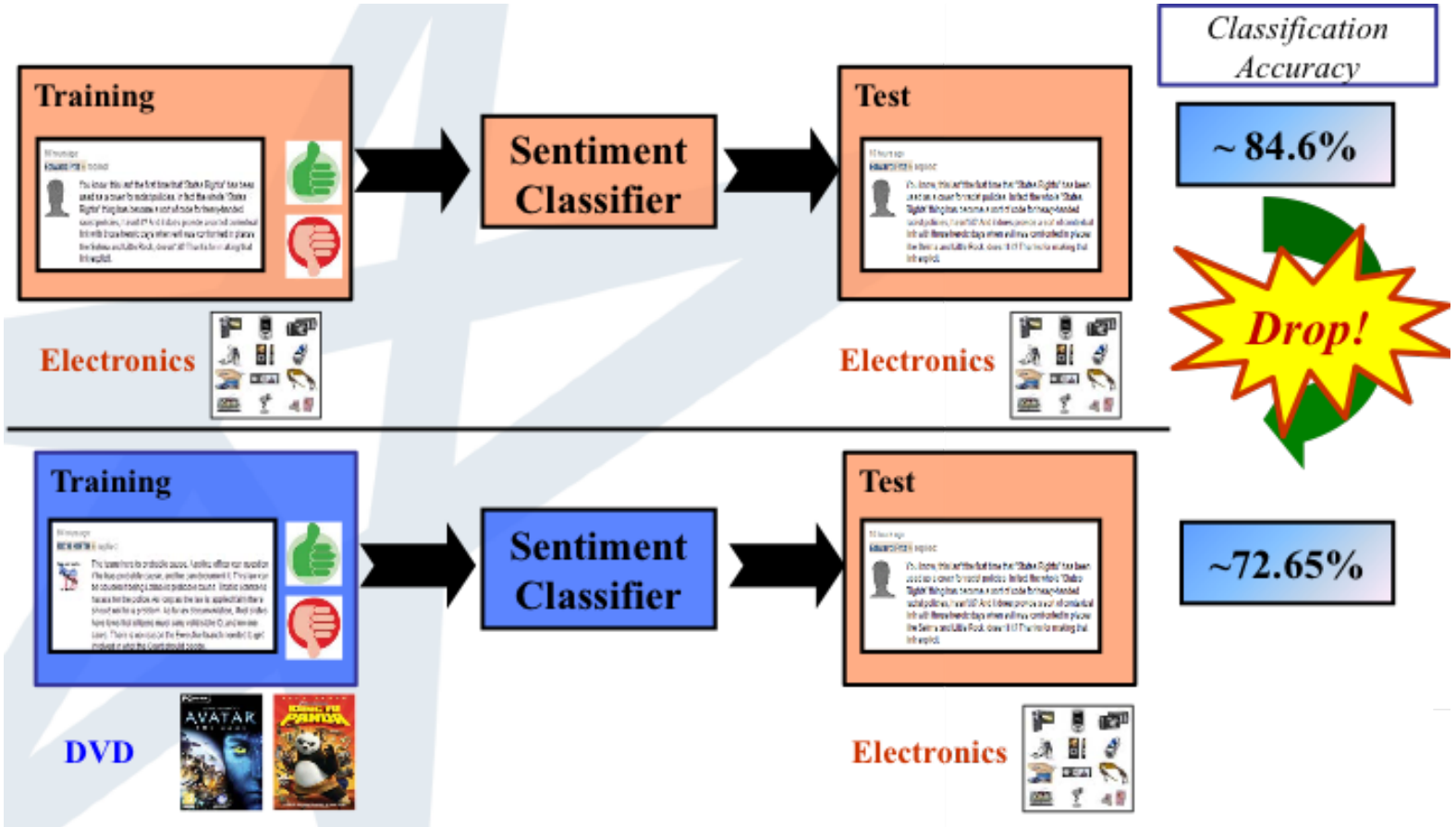
- Speech recognition in intelligent assistant (Alexa, Siri, Google home...)
- Recommender systems (Netflix, Spotify, Amazon...)
- Ads placement

Example: sample selection bias

Domain adaptation is also required to accomodate for **sampling selection bias**:

- *On purpose, we selected in the dataset the same number of spam and non-spam emails*
- *We wish to generate a model to diagnose breast cancer. Women invited for data collection have been selected among women who already did the breast screening test in the preceding years.* *(Huang et al., 2006)*

Example: sentiment analysis



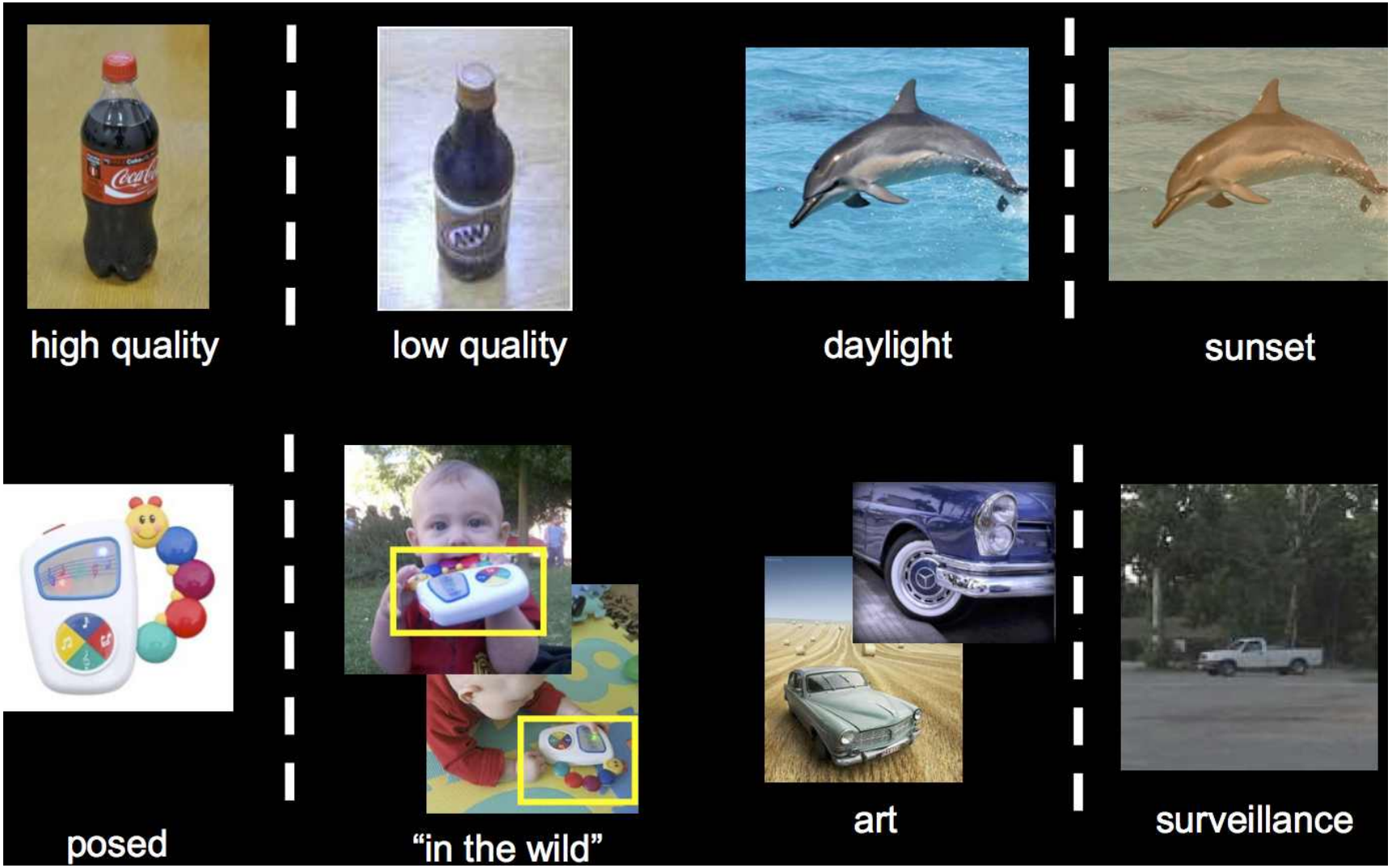
(Pan and Yang, 2013)

Example: sentiment analysis

Electronics	Video Games
(1) **Compact**; easy to operate; very good picture quality; looks **sharp**!	(2) A very good game! It is action packed and full of excitement. I am very much **hooked** on this game.
(3) I purchased this unit from Circuit City and I was very excited about the quality of the picture. It is really nice and **sharp**.	(4) Very **realistic** shooting action and good plots. We played this and were **hooked**.
(5) It is also quite **blurry** in very dark settings. I will never buy HP again.	(6) The game is so **boring**. I am extremely unhappy and will probably never buy UbiSoft again.

(Pan and Yang, 2013)

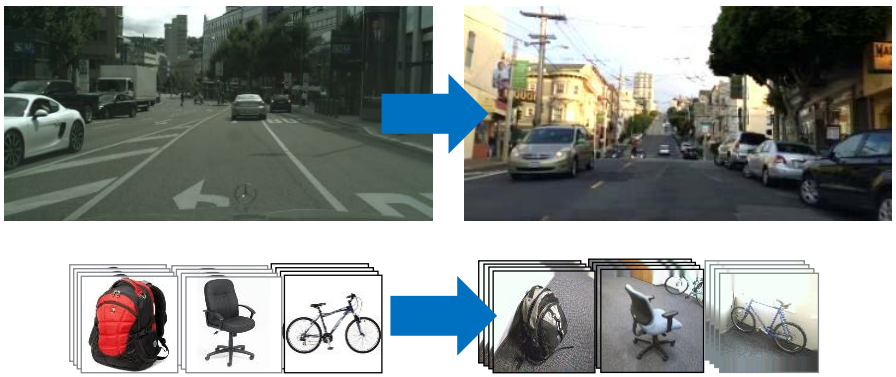
Example: many applications in computer vision



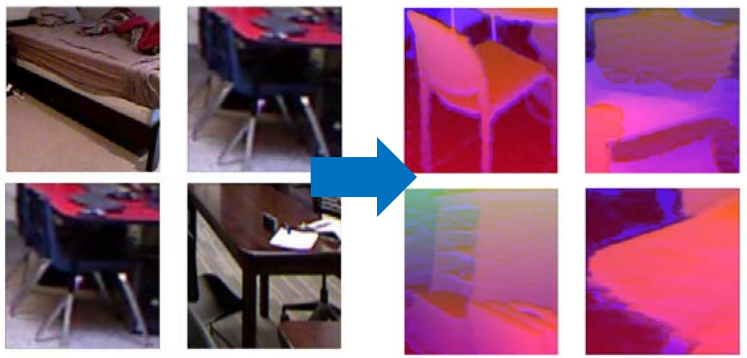
(Habrard, 2014)

Example: many applications in computer vision

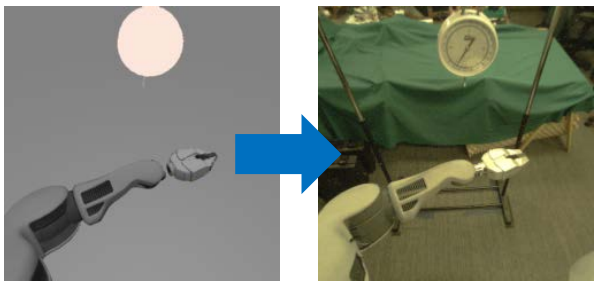
From dataset to dataset



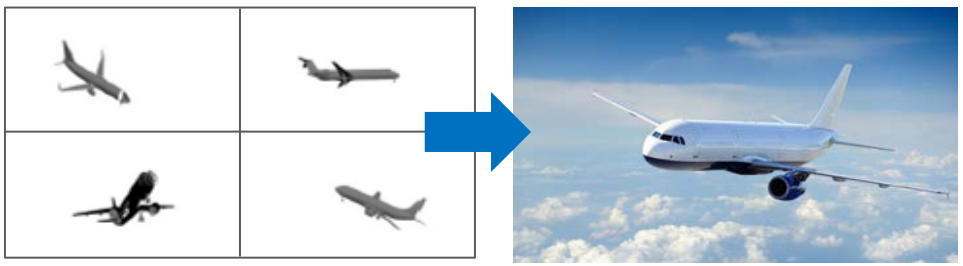
From RGB to depth



From simulated to real control



From CAD models to real images



(K. Saeko, 2016)

Example: many applications in computer vision

Trained on the Cityscapes dataset, tested on San Francisco daschcams

(K. Saeko, 2016)

Domain adaptation: formal definition

Given:

- An input space $\mathcal{X}$ (e.g., $\mathcal{R}^p$) and an output space $\mathcal{Y}$ (e.g., $\{0, 1\}$).
- An (unknown) **target** distribution: $p_t(x, y)$ (over $\mathcal{X} \times \mathcal{Y}$),
- An (unknown) **source** distribution: $p_s(x, y)$ (over $\mathcal{X} \times \mathcal{Y}$),
- $p_s(x, y) \neq p_t(x, y)$
- A learning sample (either):
  - $LS_t^U \cup LS_s^L$ (Unsupervised DA)
  - $LS_t^L \cup LS_s^L$ (Supervised DA)

  with $LS_t^L \sim p_t(x, y)^{N_t}$, $LS_t^U \sim p_t(x)^{N_t}$ and $LS_s^L \sim p_s(x, y)^{N_s}$

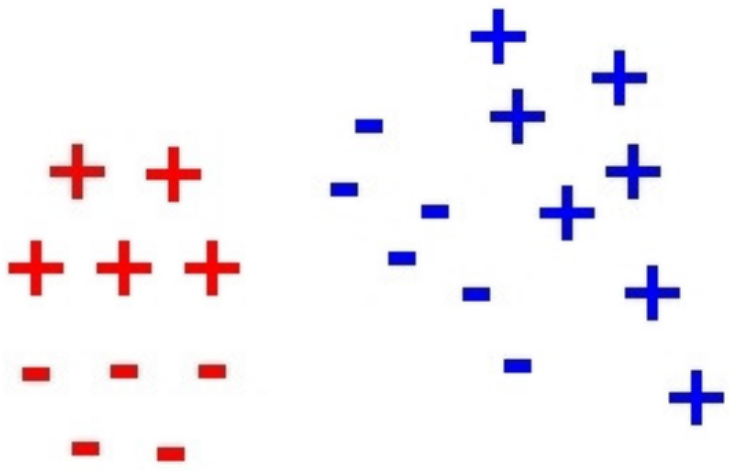
The objective is to find $f : \mathcal{X} \to \mathcal{Y}$ that minimizes:

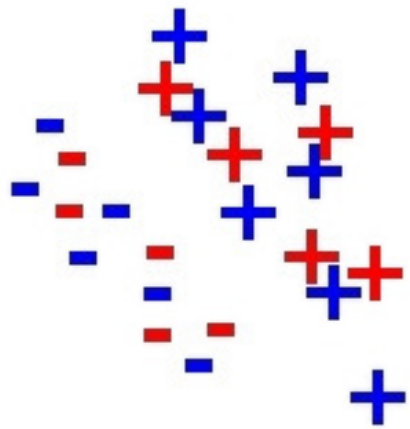
$$Err_t(f) \triangleq E_{p_t(x,y)}\{\ell(f(x), y)\}$$

(or an approximation of $p_t(y|x)$).

When will it work? Intuitively

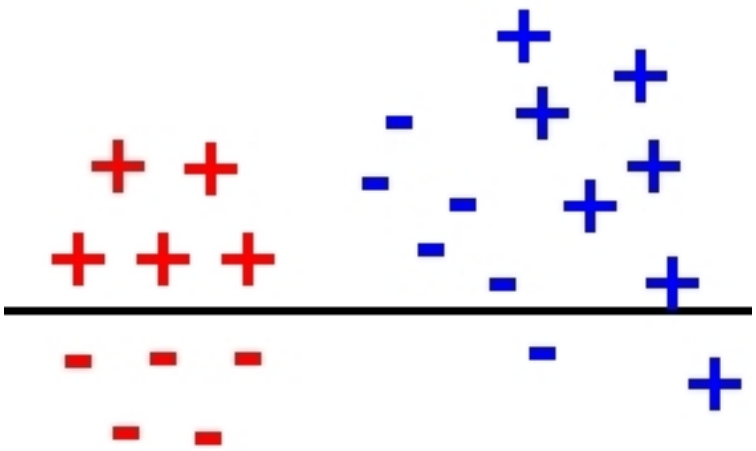
Intuitively, the target and source distributions should not be too far away from each other.

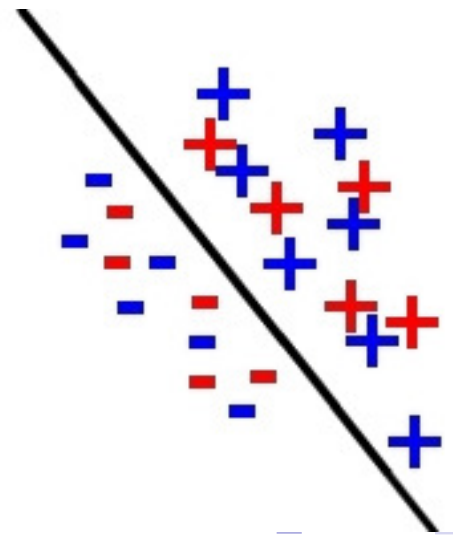




When will it work? Intuitively

Intuitively, the target and source distributions should not be too far away from each other.





When will it work? Theoretically (Ben-David et al., 2006,2010)

We consider unsupervised domain adaptation in the context of binary classification. Available data is:

- $LS_s^L = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$, with $(x_i^s, y_i^s) \sim p_s(x, y)$
- $LS_t^U = \{(x_i^t)\}_{i=1}^{N_t}$, with $x_i^t \sim p_t(x)$

Let us denote by $\mathcal{H}$ our hypothesis space (of binary classifiers). For a given function f in $\mathcal{H}$, the generalization errors on the source and target distributions are resp.:

$$Err_t(f) \triangleq E_{p_s(x,y)}\{1(f(x) \neq y)\},\ Err_s(f) \triangleq E_{p_s(x,y)}\{1(f(x) \neq y)\},$$

while their empirical counterparts are:

$$\widehat{Err}_t(f) \triangleq \frac{1}{N_t}\sum_{i=1}^{N_t} 1(f(x_i^t) \neq y_i^t),\ \widehat{Err}_s(f) \triangleq \frac{1}{N_s}\sum_{i=1}^{N_s} 1(f(x_i^s) \neq y_i^s).$$

Let us define the following empirical distance between LS_s^L and LS_t^U relative to $\mathcal{H}$:

$$\hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U) \triangleq 2\left(1 - \min_{g \in \mathcal{H}}\left[\frac{1}{N_s}\sum_{i=1}^{N_s} 1(g(x_i^s) = 0) + \frac{1}{N_t}\sum_{i=1}^{N_t} 1(g(x_i^t) = 1)\right]\right),$$

When will it work? Theoretically (Ben-David et al., 2006,2010)

Denoting by d the **VC-dimension** of $\mathcal{H}$, we have for all $f \in \mathcal{H}$ and with probability $1 - \delta$ over all choices of learning samples LS_s^L and LS_t^U, both of sizes $N_s = N_t = n$:

$$Err_t(f) \leq \widehat{Err}_s(f) + \hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U) + \beta + \lambda$$

with:

$$\beta \geq \inf_{g \in \mathcal{H}} [Err_t(g) + Err_s(g)], \lambda = \sqrt{\frac{4}{n}(d \log \frac{2en}{d} + \log \frac{4}{\delta})} + 4\sqrt{\frac{1}{n}(d \log \frac{2n}{d} + \log \frac{4}{\delta})}$$

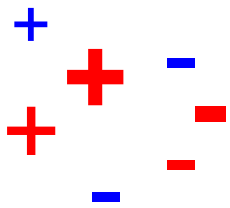
(Ben-David et al., 2006,2010)

Interpretation: Error of a classifier in the target domain is upper bounded by the sum of three terms:

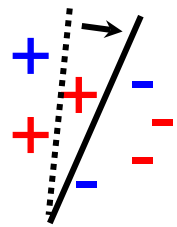
- Empirical error of the same classifier in the source domain ($\widehat{Err}_s(f)$)
- Distance between $p_s(x)$ and $p_t(x)$ ($\hat{d}_{\mathcal{H}}$)
- How related the two tasks are (β)

Main method families for domain adaptation

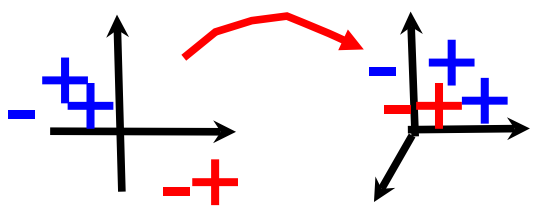
- Reweighting/Instance-based methods
  - Correct sample selection bias by reweighting/ selecting source labeled data



- Adjustement/iterative methods
  - Modify the model by incorporating pseudo-labeled information



- Inferring domain-invariant features
  - Find a common feature space where source and target distributions are close (by projection, new features, etc.)



Instance reweighting

Given our training data:

- $LS_s^L = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$, with $(x_i^s, y_i^s) \sim p_s(x, y)$
- $LS_t^U = \{(x_i^t)\}_{i=1}^{N_t}$, with $x_i^t \sim p_t(x)$

we want to find f in some hypothesis space $\mathcal{H}$ that minimizes:

$$Err_t(f) \triangleq E_{p_t(x,y)}\{\ell(f(x), y)\}$$

Problem: the target data is unlabeled and from the source data, we can a priori only estimate:

$$Err_s(f) \triangleq E_{p_s(x,y)}\{\ell(f(x), y)\} \approx \frac{1}{N_s} \sum_{i=1}^{N_s} \ell(f(x_i^s), y_i^s)$$

$Err_t(f)$ can nevertheless be estimated from samples from p_s using **importance sampling**.

Instance reweighting

$$
\begin{aligned}
E_{p_t(x,y)}\{\ell(f(x),y)\} &= E_{p_t(x,y)}\{\frac{p_s(x,y)}{p_s(x,y)}\ell(f(x),y)\} \\
&\triangleq \int\int p_t(x,y)\frac{p_s(x,y)}{p_s(x,y)}\ell(f(x),y)dxdy \\
&= \int\int p_s(x,y)\frac{p_t(x,y)}{p_s(x,y)}\ell(f(x),y)dxdy \\
&\triangleq E_{p_s(x,y)}\{\frac{p_t(x,y)}{p_s(x,y)}\ell(f(x),y)\} \\
&\approx \frac{1}{N_s}\sum_{i=1}^{N_s}\frac{p_t(x_i^s,y_i^s)}{p_s(x_i^s,y_i^s)}\ell(f(x_i^s),y_i^s)\}
\end{aligned}
$$

If one can estimate the density ratio $\beta(x,y) = \frac{p_t(x,y)}{p_t(x,y)}$, then domain adaptation can be solved by minimizing the weighted empirical loss:

$$\frac{1}{N_s}\sum_{i=1}^{N_s}\beta(x_i^s,y_i^s)\ell(f(x_i^s),y_i^s)$$

How to estimate $\beta(.,.)$ from unlabeled data?

Prior probability shift

Let us focus on classification problems and assume that $p_t(x|y) = p_s(x|y)$ and $p_t(y) \neq p_s(y)$. We defined this situation as **prior probability shift**.

In this case, we have:

$$\frac{p_t(x,y)}{p_s(x,y)} = \frac{p_t(y)p_t(x|y)}{p_s(x)p_s(x|y)} = \frac{p_t(y)}{p_s(y)} \triangleq \beta(y).$$

If $p_t(y)$ is known, then one can train the model on the source data to minimize the corresponding weighted loss:

$$\frac{1}{N_s}\sum_{i=1}^{N_s} \beta(y_i^s)\ell(f(x_i^s), y_i^s).$$

Or one can equivalently resample the source examples to match the target prior class distribution.

Prior probability shift

Classifiers trained on the source data (without weights) can also be adjusted without retraining:

- If the classifier directly estimates $p_s(y|x)$, then the resulting estimate can be turned into an estimate of $p_t(y|x)$ using:

$$p_t(y|x) = \frac{\frac{p_t(y)}{p_s(y)} P_s(y|x)}{\sum_{y \in \mathcal{Y}} \frac{p_t(y)}{p_s(y)} P_s(y|x)}$$

- If the classifier makes prediction according to $p_t(x|y)p_t(y)$, then $p_t(y)$ can be substituted by $p_s(y)$.

If $p_t(y)$ is unknown, it is possible to estimate it from the unlabeled target data (using MLE or a bayesian approach) by exploiting the estimate of $p_s(y|x)$ as provided by a classifier trained on the source data (Latinne et al., 2001).

Covariate shift

Covariate shift assumes that:

- marginal input distributions are different: $p_s(x) \neq p_t(x)$,
- but conditional output distributions are the same: $p_t(y|x) = p_s(y|x)$.

The weights are then given by:

$$\frac{p_t(x,y)}{p_s(x,y)} = \frac{p_t(x)p_t(y|x)}{p_s(x)p_s(y|x)} = \frac{p_t(x)}{p_s(x)} \triangleq \beta(x),$$

which can in principle be estimated from unlabeled target data.

Instance reweighting to correct for covariate shift:

1. Get estimates $\hat{\beta}(x_i^s)$ of the weights (from unlabeled target and source data)
2. Learn a model f to minimize the weighted loss (from labeled source data):

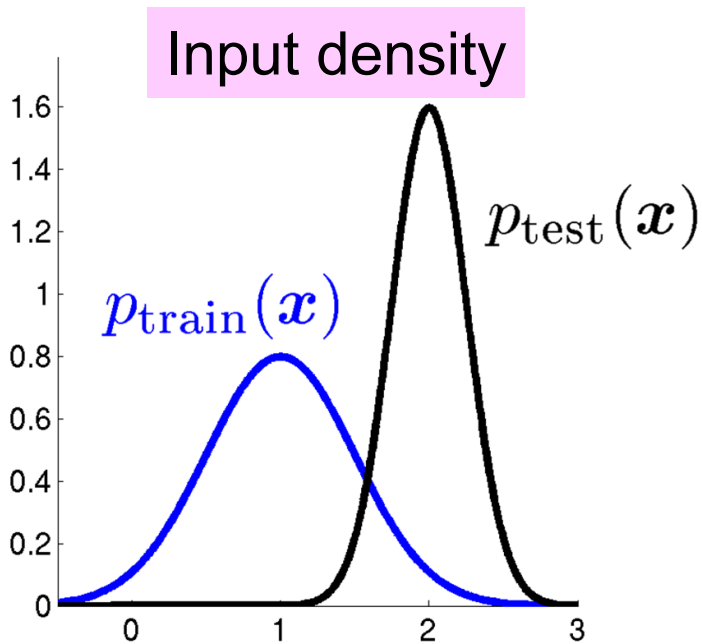
$$\frac{1}{N_s} \sum_{i=1}^{N_s} \hat{\beta}(x_i^s) \ell(f(x_i^s), y_i^s)$$

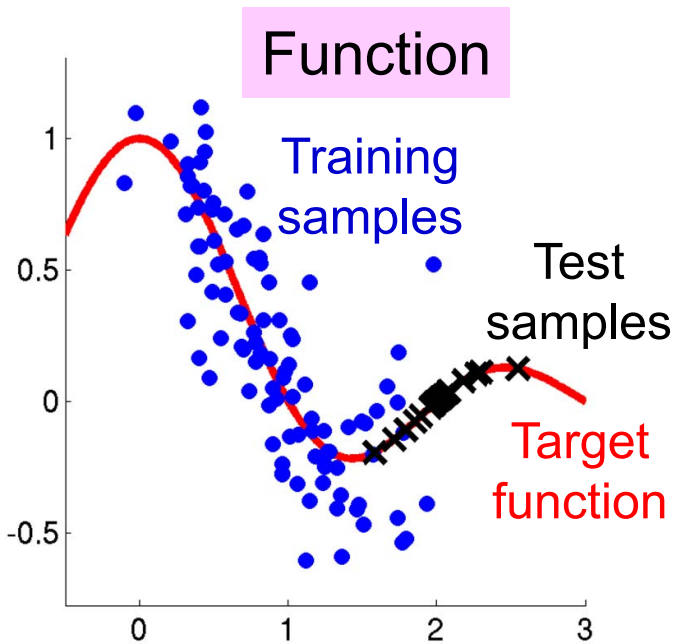
Covariate shift

Since $p_s(y|x) = p_t(y|x)$, why can't we just estimate $p_t(y|x)$ from the source data?

Covariate shift

Since $p_s(y|x) = p_t(y|x)$, why can't we just estimate $p_t(y|x)$ from the source data?

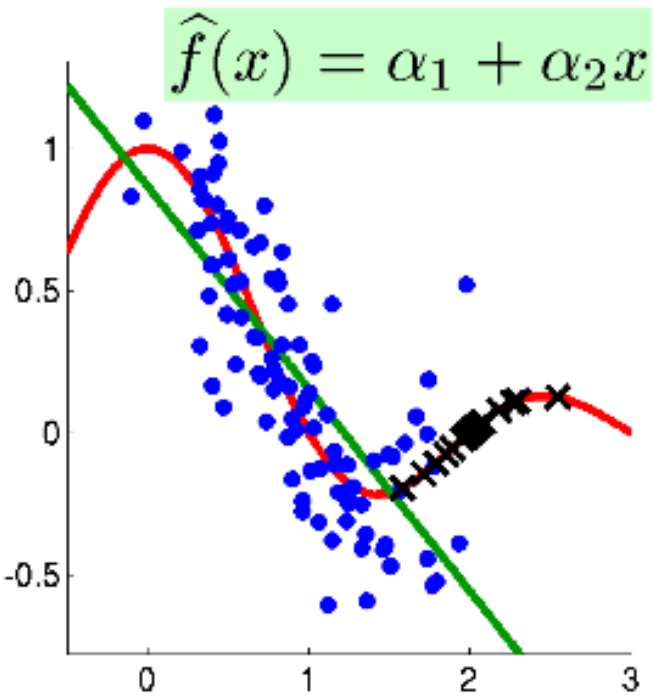


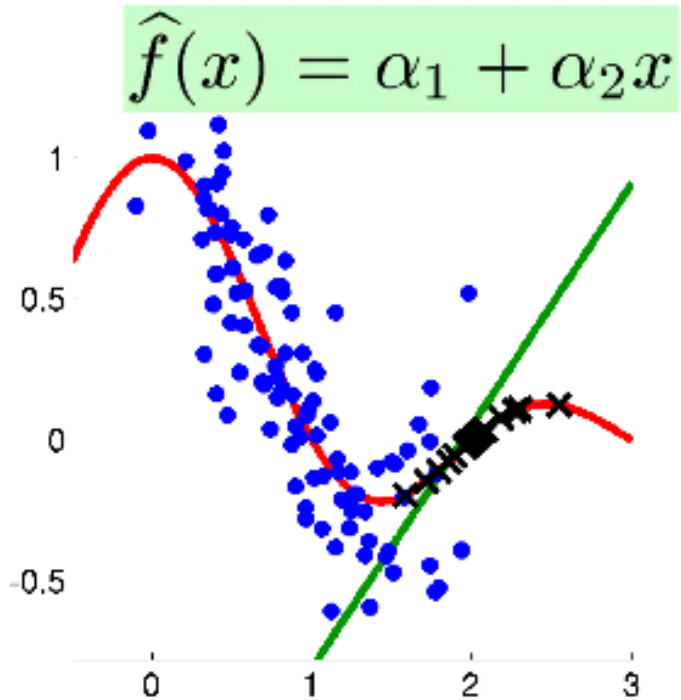


(Sugiyama, 2012)

Covariate shift

Since $p_s(y|x) = p_t(y|x)$, why can't we just estimate $p_t(y|x)$ from the source data?

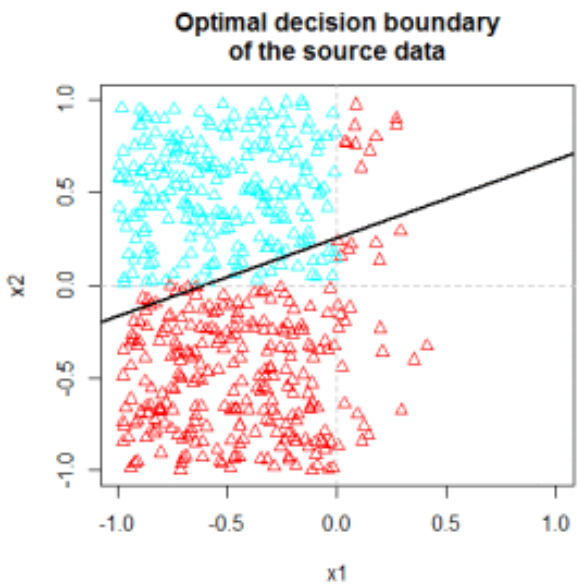


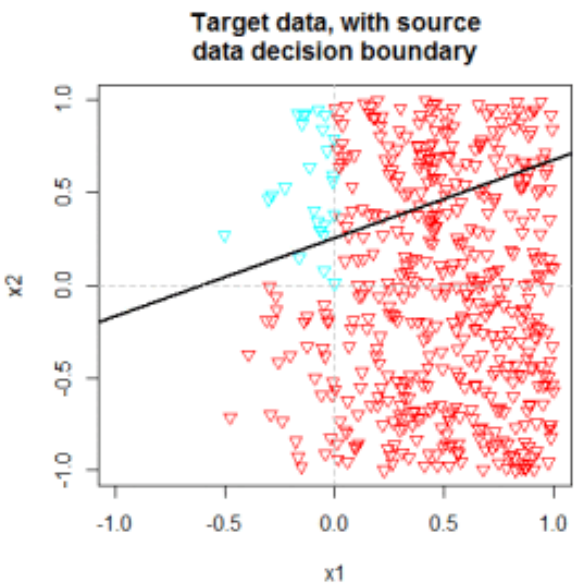


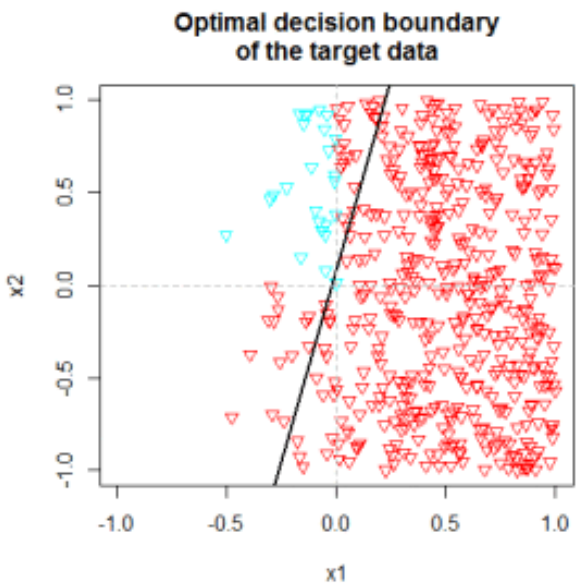
(Sugiyama, 2012)

If the target function does not belong to the hypothesis space, model performance will differ between source and target, even asymptotically.

In classification







https://johanndejong.wordpress.com

How to estimate the density ratio?

Naive approach:

1. Build separate estimators $\hat{p}_s(x)$ and $\hat{p}_t(x)$ resp. from source and target data
2. Compute their ratio

This works poorly, as density estimation is a hard problem in high dimension and small errors in $\hat{p}_s(x)$ can lead to large variation of the ratio.

Several **alternative approaches** have been proposed in the literature:

- Using a probabilistic classifier
- Kernel Mean Matching
- Direct density ratio estimation
- ...

Example: using a probabilistic classifier (Zadrozny, 2004, Bickel et al., 2007)

Let us assign a value $d = 1$ to the target samples and $d = 0$ to the source samples. Regarding d as a random variable, one thus can write the two densities $p_t(x) = p(x|d = 1)$ and $p_s(x) = p(x|d = 0)$. Their ratio thus becomes:

$$\beta(x) = \frac{p_t(x)}{p_s(x)} = \frac{p(x|d = 0)}{p(x|d = 1)}.$$

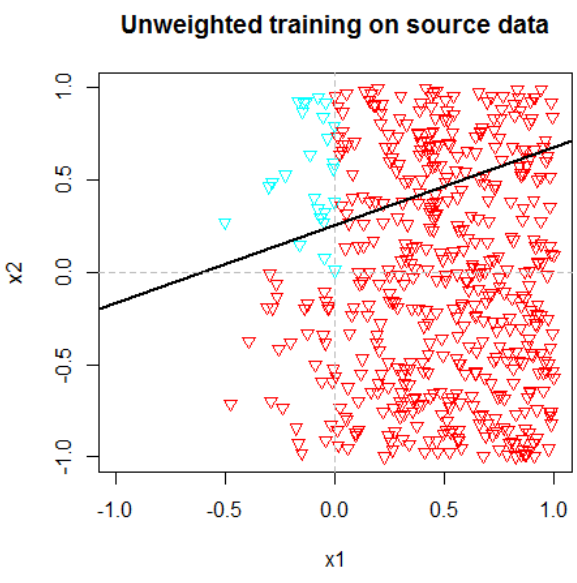
Using Bayes' theorem, this can be rewritten as:

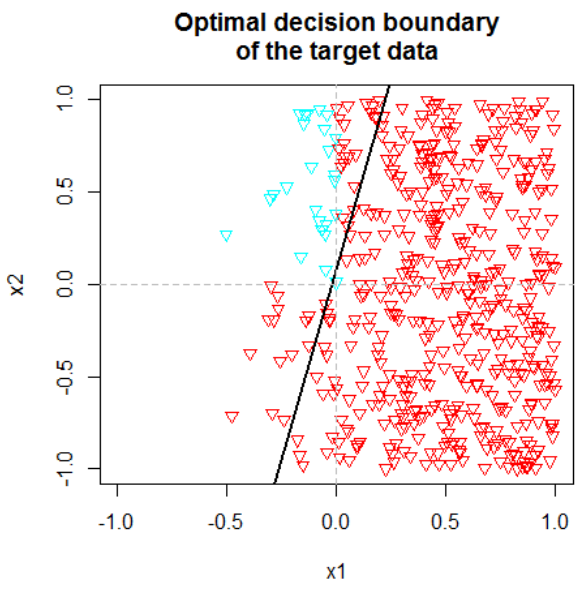
$$\beta(x) = \frac{p(d = 0)p(d = 1|x)}{p(d = 1)p(d = 0|x)}.$$

$p(d = 0)/p(d = 1)$ can be estimated by N_s/N_t and
$p(d = 1|x)(= 1 - P(d = 0|x))$ can be estimated by training a **probabilistic** classifier (eg., logistic regression) on the following learning sample:

$$LS = \{(x_1^s, 0), \ldots, (x_{N_s}^s, 0), (x_1^t, 1), \ldots, (x_{N_t}^t, 1)\}$$

Illustration





https://johanndejong.wordpress.com

Outline

1. Overview of the main learning protocols
2. Introduction
3. Instance reweighting
4. Iterative methods
5. Feature-based approaches

Iterative methods: DASVM (Bruzzone et al., 2010)

Main idea: train a model iteratively by integrating progressively information about the target samples.

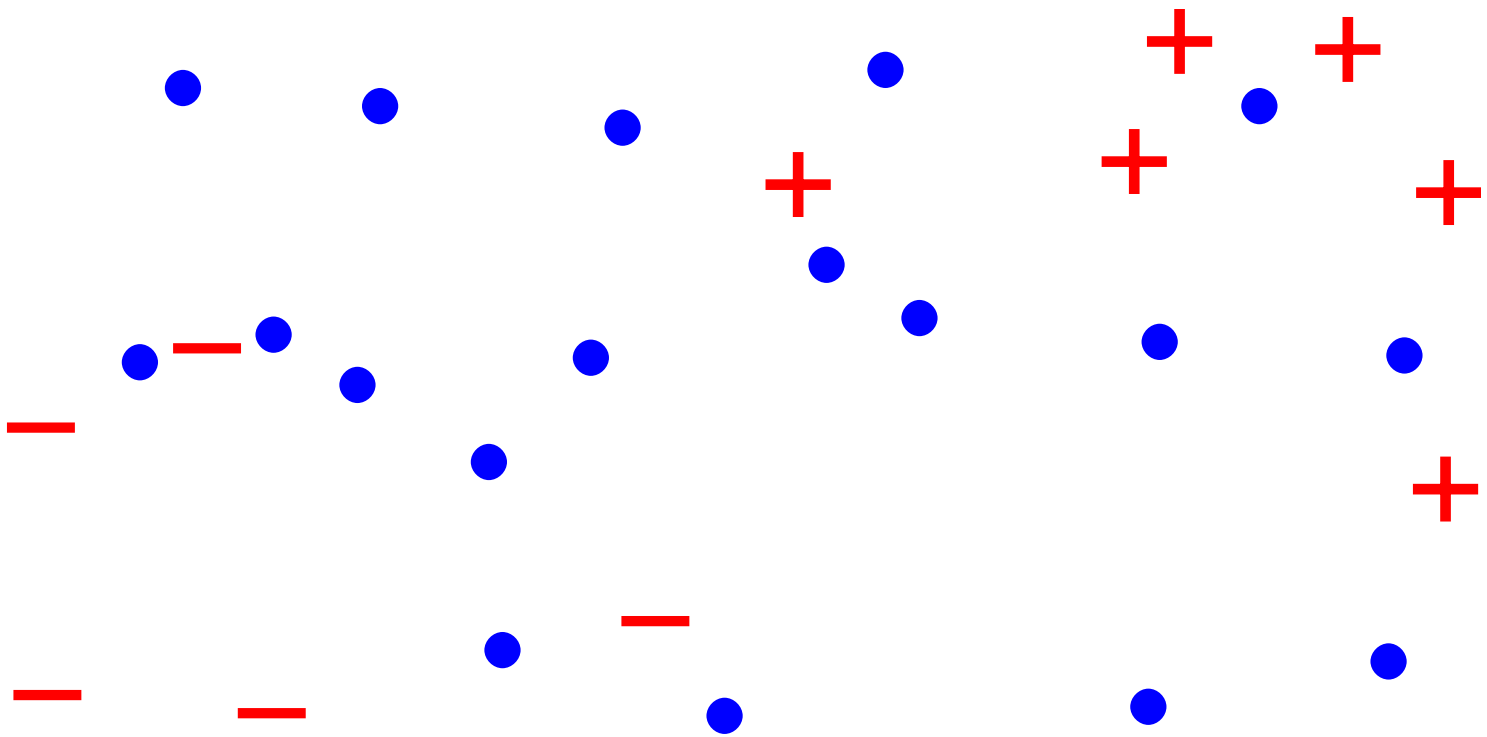
Example with SVM:

1. $LS = LS_s^L$
2. Learn a classifier h^0 from LS
3. Repeat until stop criterion is met:
   - Select the first k target examples x_i^t s.t. $0 < w^T x_i^t + b < 1$ with highest margin and affect the pseudo-label -1
   - Select the first k target examples x_i^t s.t. $-1 < w^T x_i^t + b < 0$ with highest margin and affect the pseudo-label +1
   - Add these $2k$ examples (with their pseudo-labels) to LS
   - Remove from LS the first k positive and k negative source instances with highest margin
4. Return the last classifier

Stop criterion: the number of selected instances is below some threshold.

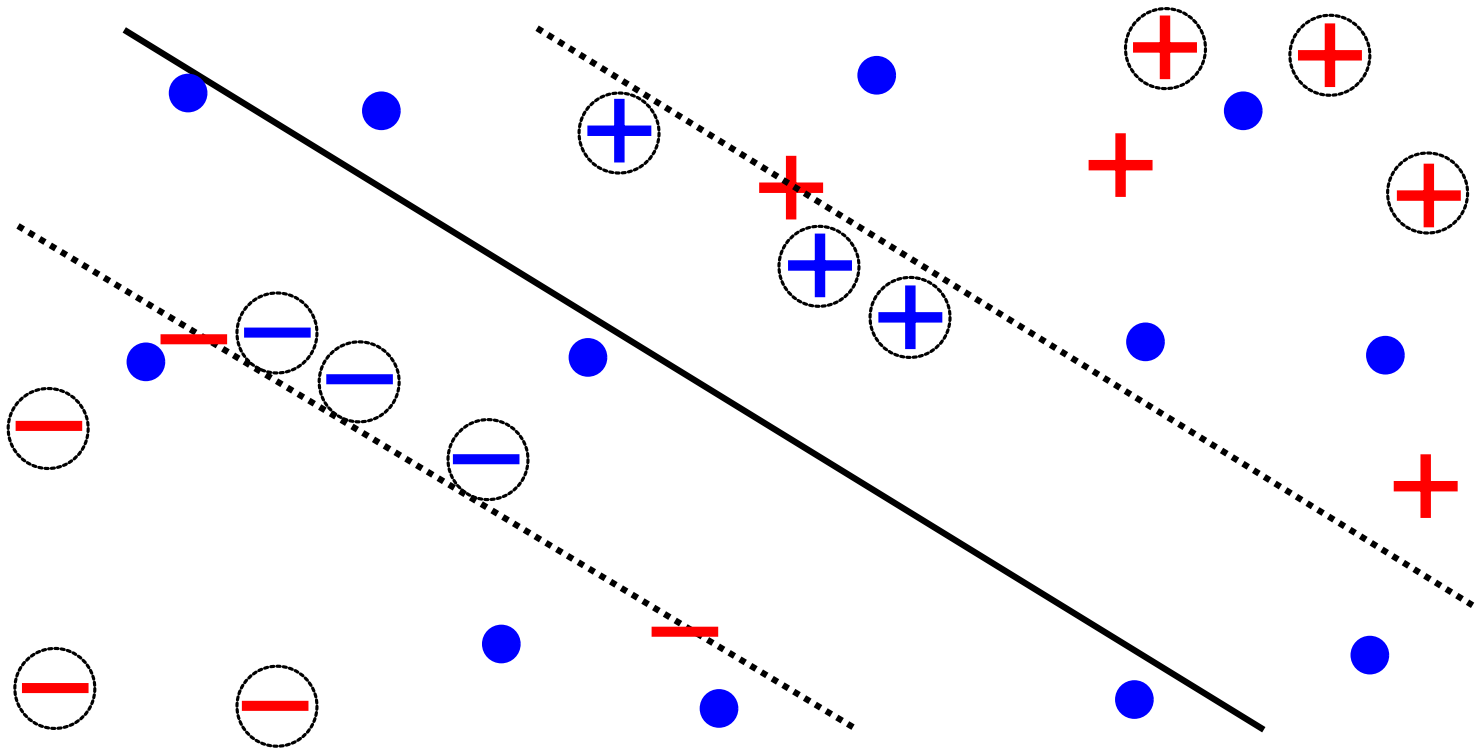
The convergence of the algorithm has been analysed theoretically and boosting methods can also be adapted similarly (Habrard, 2013).

DASVM: Illustration



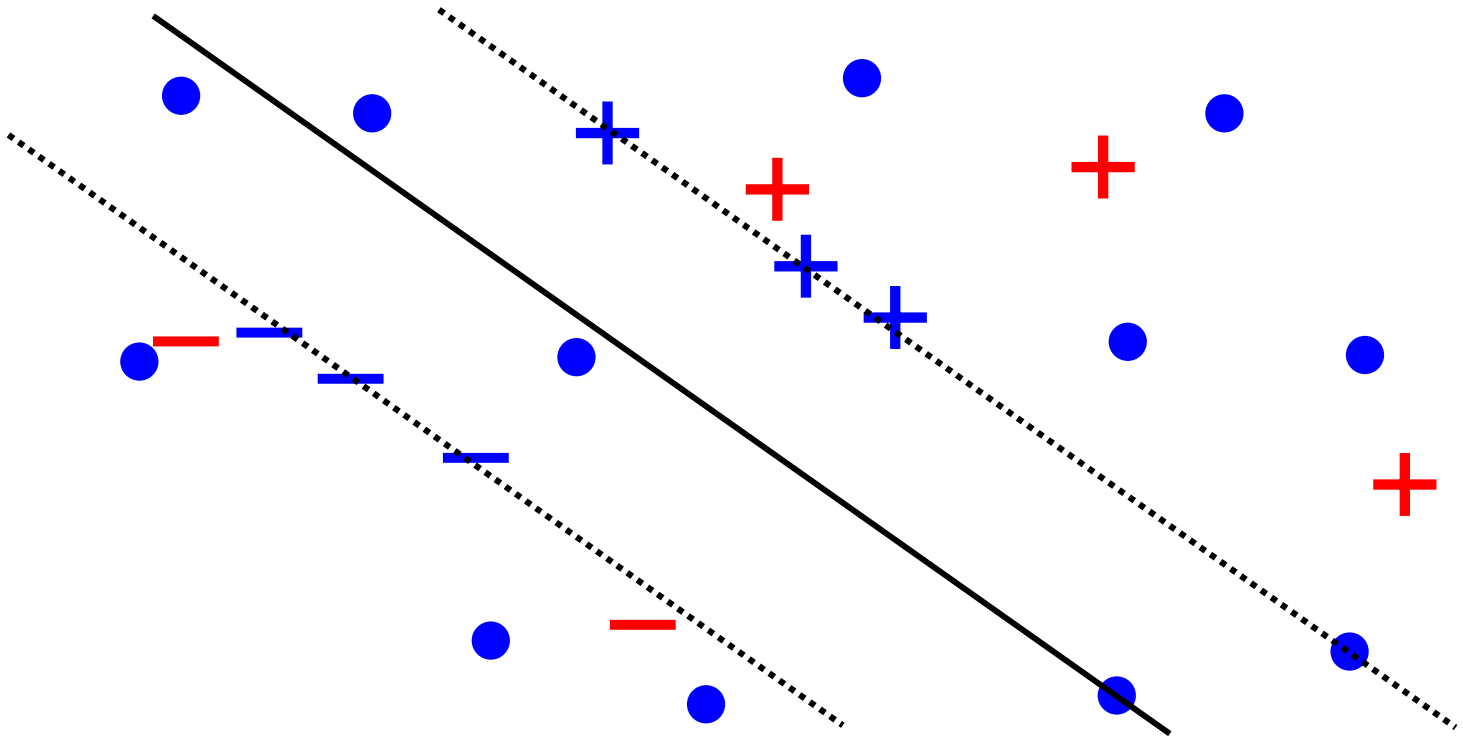
(A. Habrard, EPAT 2014)

DASVM: Illustration



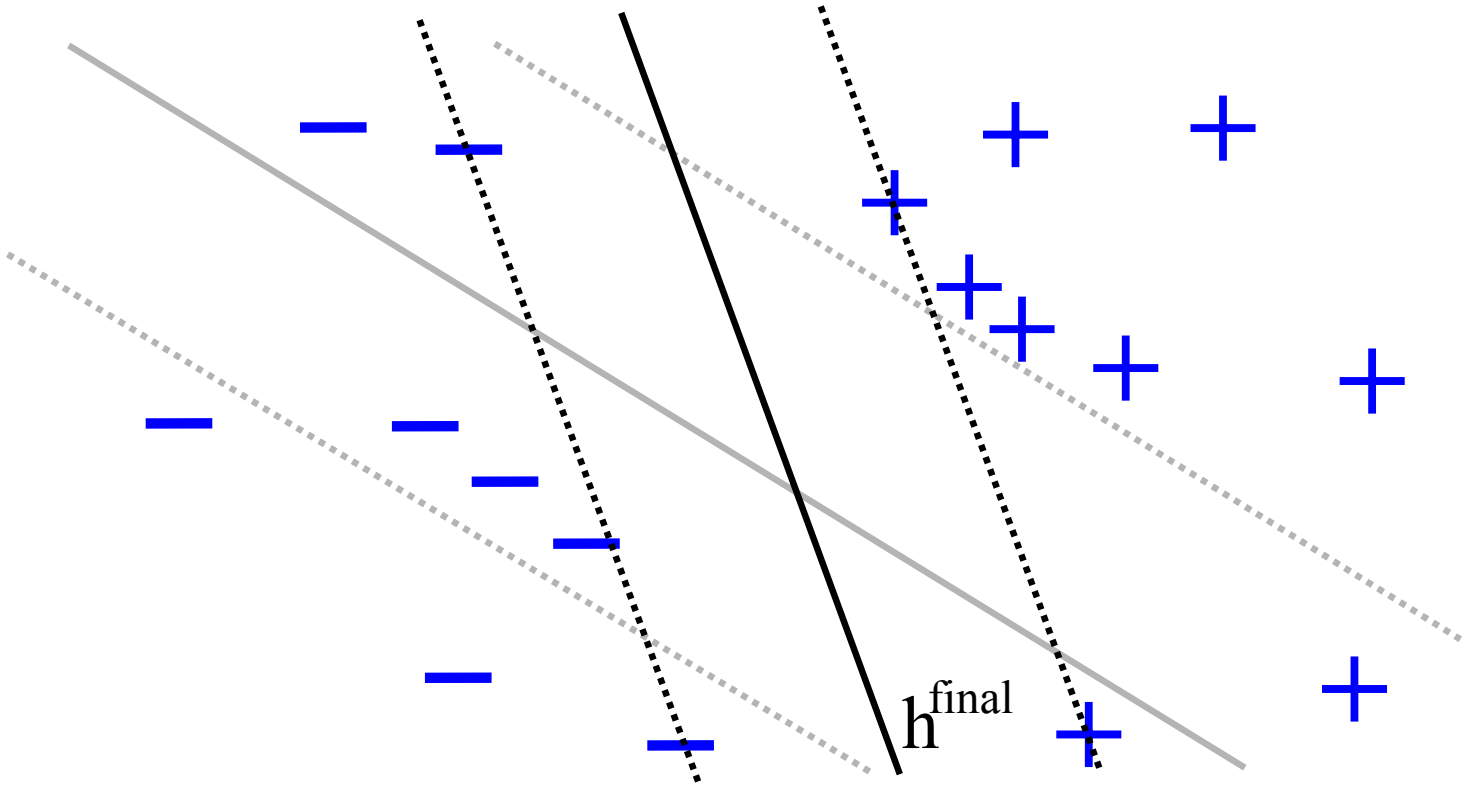
(A. Habrard, EPAT 2014)

DASVM: Illustration



(A. Habrard, EPAT 2014)

DASVM: Illustration



(A. Habrard, EPAT 2014)

Outline

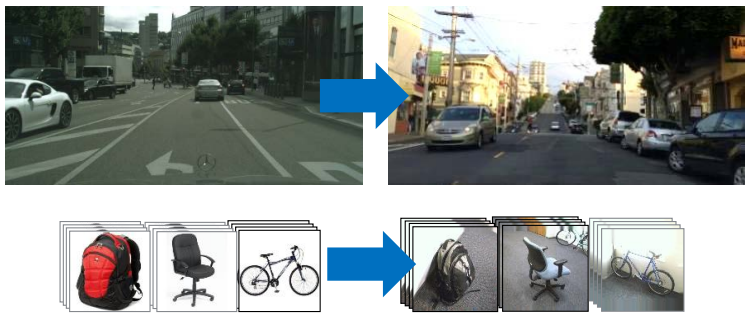
1. Overview of the main learning protocols
2. Introduction
3. Instance reweighting
4. Iterative methods
5. Feature-based approaches

Feature-based approaches

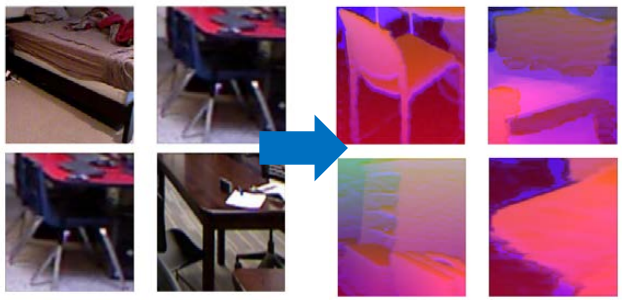
Instance reweighting and iterative methods will most likely fail if $p_t(x, y)$ and $p_s(x, y)$ are too different.

Instance reweighting assumes that $p_t(x)$ and $p_s(x)$ have the same/similar support (*why?*). Not true in many applications.

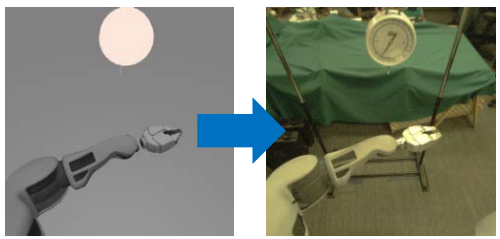
From dataset to dataset



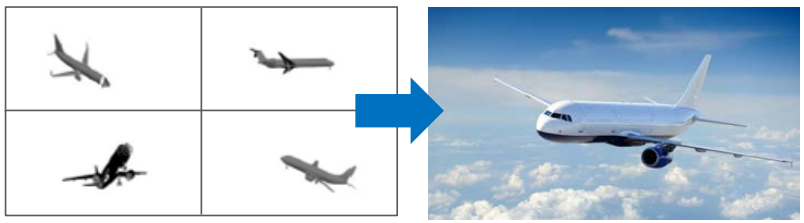
From RGB to depth



From simulated to real control



From CAD models to real images



(K. Saeko, 2016)

Feature-based approaches

Main idea: map the input space $\mathcal{X}$ into a new space $\mathcal{Z}$ through a transformation $g : \mathcal{X} \to \mathcal{Z}$, with the hope that:

1. $p_t(z, y) = p_s(z, y)$, so that a model trained on source using z as inputs will adapt to target
2. Most of the information about the output is preserved ($I(Y|Z) \approx I(Y|X)$)

This is a difficult task because labels are usually not available in the target data. A common proxy is to learn g so that at least $p_s(z) \approx p_t(z)$ and most of the information in x is retained.

Transformations g could take several forms:

- linear,
- non-linear,
- or feature subset selectors.

In the latter case, in the hypothesis that some features are domain-specific (and thus should be avoided) and others are generalizable.

Feature-based approaches

Several methods fall into this category.

In this lecture:

- Stacked denoising auto-encoders (Glorot et al, 2011, Chen et al., 2012)
- Domain-adversarial training of neural networks (Ganin et al., 2016) (see Part 2).

NB: A related (asymmetric) approach is to learn a transformation T such that:

$$p_s(y|x) \approx p_t(y|T(x))$$

Many ways to train T. See student lectures for an example using optimal transport.

DA with Stacked Denoising Auto-encoder (Glorot et al, 2011, Chen et al., 2012)

Main idea:

- **Pool** all target and source examples to train an auto-encoder model
- **Extract features** at different depths to define a new input representation $g(x)$
- **Train** a model (eg., SVM) on the source data using $g(x)$ as input

Glorot et al. (2011) use a **Stacked Denoising Auto-encoder (SDA)**:

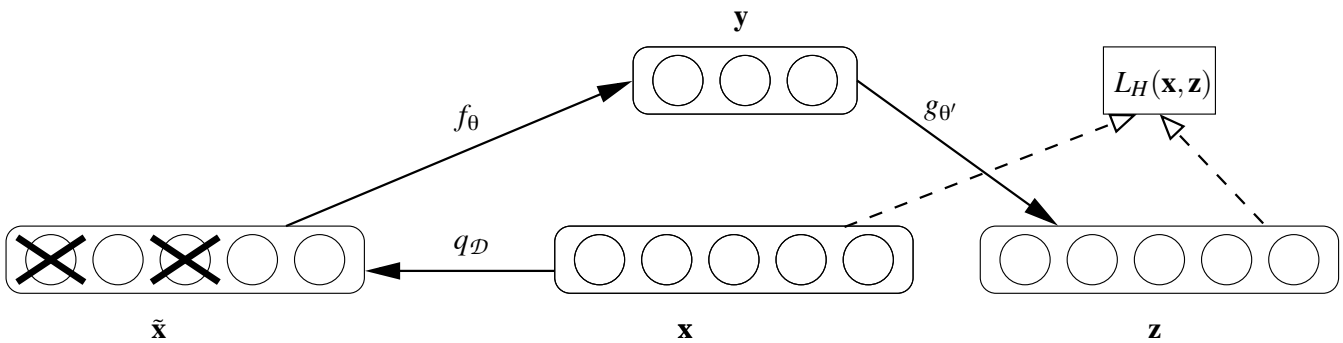
- Each layer is trained individually to denoise its corrupted input (by setting some random features to 0).
- A multiple layer auto-encoder is trained greedily layer by layer

Chen et al. (2012) use a **marginalized** SDA: similar model but with linear Denoising Auto-encoder in each layer (simpler and faster to train).

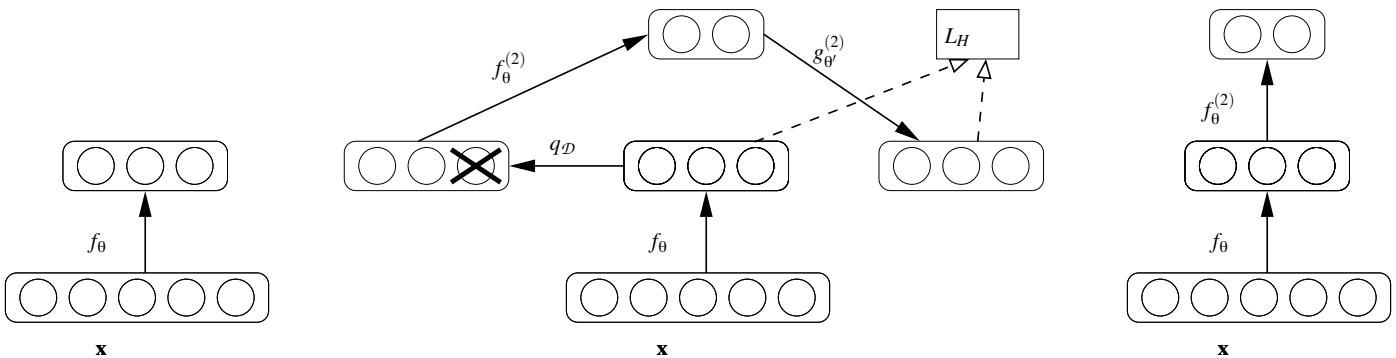
Heuristic approaches that do not exploit any assumptions of the DA problem.

Stacked denoising auto-encoder

Denoising auto-encoder:

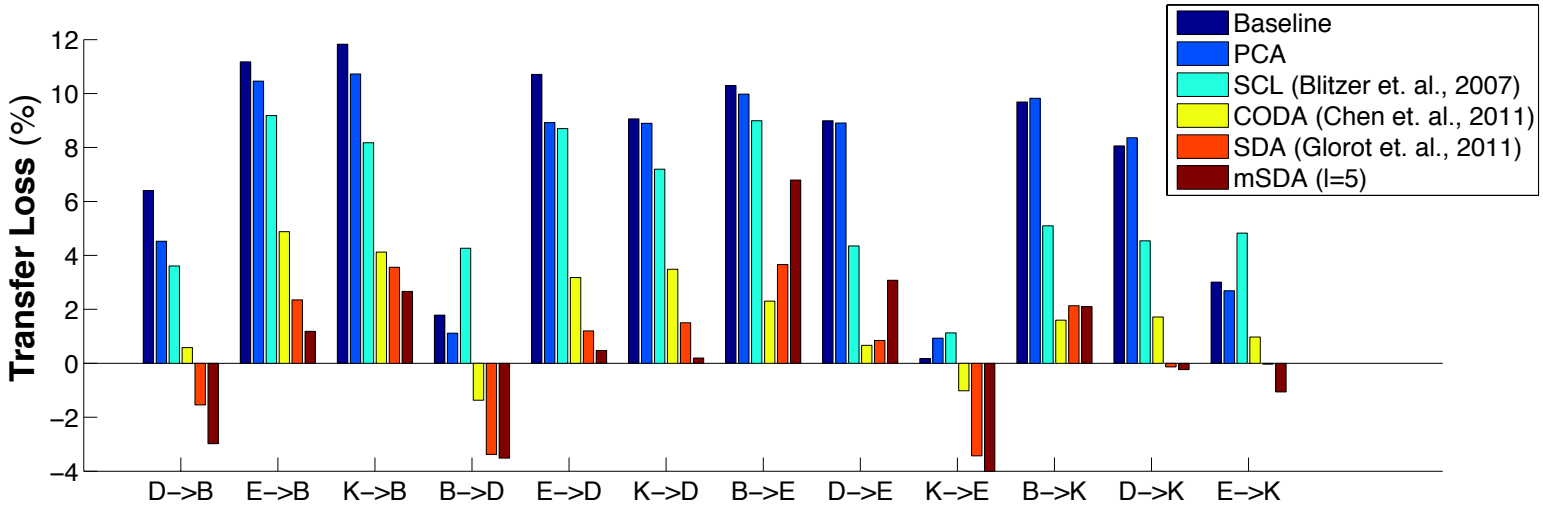


Stacking them:



(Vincent et al., 2010)

Some results for sentiment analysis (Chen et al., 2012)



- Dataset: Amazon product reviews. Try to predict score>3 or ≤ 3 from bag-of-words unigram/bigram features.
- Different product categories (= domains): Books, Kitchen appliances, Electronics, DVDs (12 DA tasks).
- Transfer loss $= e(S,T) - e_b(T,T)$, with $e(S,T)$ the error on target data of a model trained on source data and $e_b(T,T)$ the error on target data of a model trained on target data (with bow features).
- SVM used on top of the SDA/mSDA features. Baseline is SVM on bow representation, only trained on source data (no DA).

References

Review papers:

- Pan and Yang. A survey on Transfer Learning, TKDE 2010.
- Jiang. A literature survey on domain adaptation of statistical classifiers. 2008
- Margolis. A Literature Review of Domain Adaptation with Unlabeled Data. Tech report 2011.
- M. Sugiyama. Density ratio estimation: a comprehensive survey. 2010.

Lectures:

- EPAT 2014 Tutorial on Transfer Learning and Domain adaptation by Amaury Habrard
- ECCV 2014 turorial on Domain adaptation and Transfer Learning by T. Tommasi and F. Orabona

Part II

Domain-adversarial training of neural networks

The paper

Domain-Adversarial Training of Neural Networks
Ganin et al., Journal of Machine Learning Research in 2016.

Present a new feature-based approach for unsupervised domain adaptation inspired by Ben-David et al. (2010)'s theoretical bound.

This journal paper actually merges two works that independently presented almost exactly the same algorithm:

- *Domain-adversarial neural networks*, Ajakan et al, 2014. (Universités de Laval et Sherbrooke, Québec, Canada)
- *Unsupervised domain adaptation by backpropagation*, Ganin and Lempitsky, 2015. (Skoltech, Russia)

Theoretical motivation

(Ben-David et al., 2006,2010)

As shown earlier, Ben-David et al. (2006,2010) have proven that for all $f \in \mathcal{H}$ and with high probability over all choices of learning samples LS_s^L and LS_t^U, we have:

$$Err_t(f) \leq \widehat{Err}_s(f) + \hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U) + \beta + \lambda$$

with λ a constant (with respect to f) and:

$$\beta \geq \inf_{g \in \mathcal{H}} [Err_t(g) + Err_s(g)]$$

Interpretation: Error of a classifier in the target domain is upper bounded by the sum of three terms:

- Empirical error of the same classifier in the source domain $(\widehat{Err}_s(f))$
- Distance between $p_s(x)$ and $p_t(x)$ $(\hat{d}_{\mathcal{H}})$
- How related the two tasks are (β)

Theoretical motivation

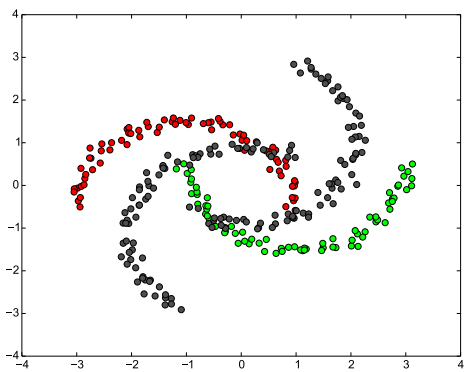
(Ben-David et al., 2006,2010)

Main idea of the proposed method: Find a new representation z:

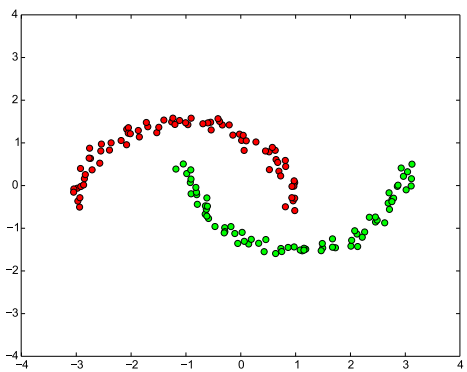
- that allows to learn a good model in the source domain ($\widehat{Err}_s(f)$ small)
- that makes the two domains indistinguishable ($\hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U)$ small)

$$Err_t(f) \leq \widehat{Err}_s(f) + \hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U) + \beta + \lambda \text{ with } \beta \geq \inf_{g \in \mathcal{H}} [Err_t(g) + Err_s(g)]$$

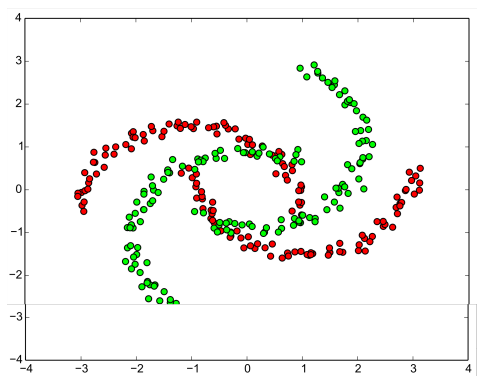
$Err_t(f)$ is small if



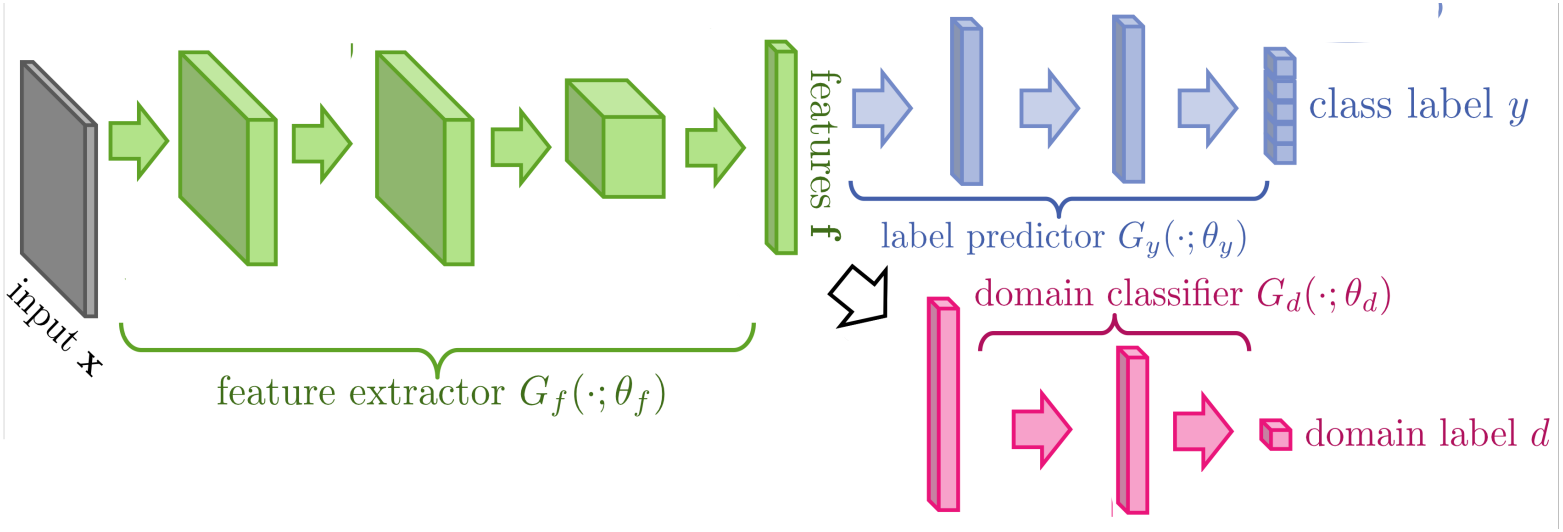
$\widehat{Err}_s(f)$ is small



and $\hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U)$ is small



Proposed model



Three neural networks are trained:

- A feature extractor, mapping inputs into a D-dimensional vector:

$$G_f(.;\theta_f) : \mathcal{X} \to \mathcal{R}^D$$

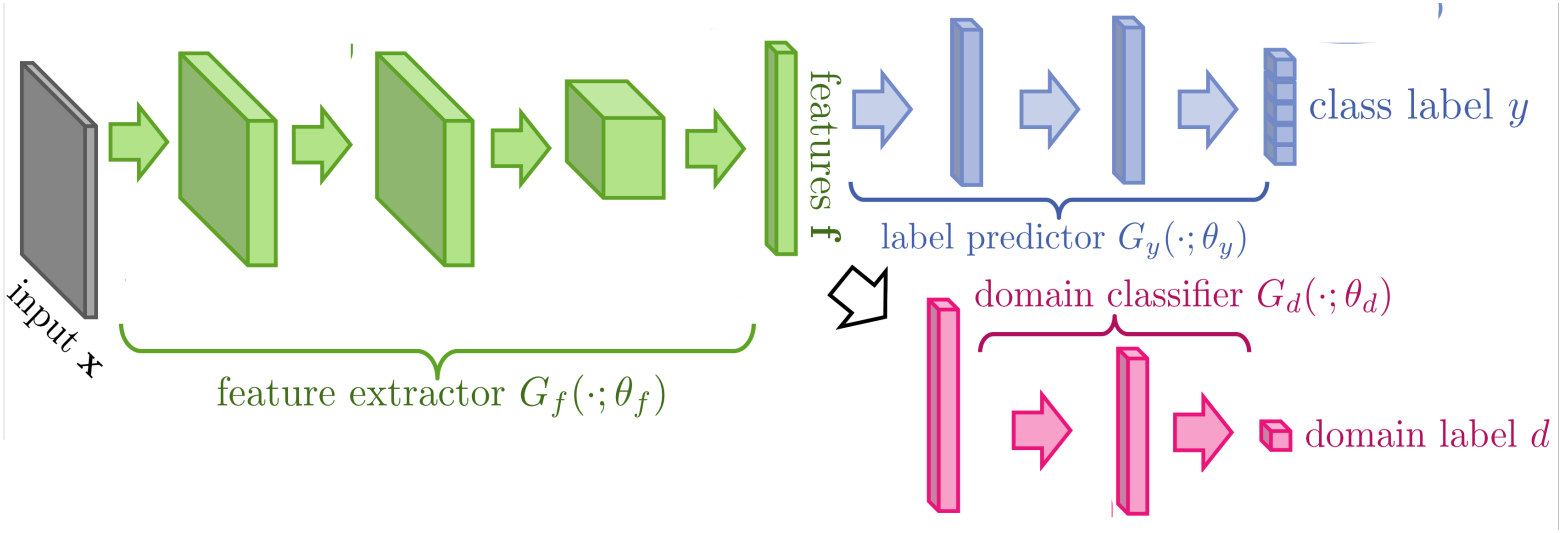
- A label predictor:

$$G_y(.;\theta_y) : \mathcal{R}^D \to \mathcal{Y}$$

- A domain predictor:

$$G_d(.;\theta_d) : \mathcal{R}^D \to \{0, 1\}$$

Proposed model



Motivated by the theoretical bound, we want to learn $G_f(.;\theta_f)$ such that

- One can learn $G_y(.;\theta_y)$ such that $G_y(G_f(.;\theta_f);\theta_y)$ is a good label predictor on the source domain
- One can **not** learn $G_d(.;\theta_d)$ such that $G_d(G_f(.;\theta_f);\theta_d)$ is a good domain predictor.

Optimisation problem

Given:

- The labeled source data: $LS_s^U = \{(x_i, y_i)\}_{i=1}^{N_s}$
- The unlabeled target data: $LS_t^L = \{x_i\}_{i=N_s+1}^{N_S+N_t}$
- The domain labels: $\{d_i\}_{i=1}^{N_s+N_t}$ with $d_i = 0$ if $1 \leq i \leq N_s$, 1 otherwise)
- The loss of the label predictor:

$$\mathcal{L}_y^i(\theta_f, \theta_y) = \mathcal{L}_y(G_y(G_f(x_i; \theta_f); \theta_y), y_i)$$

- The loss of the domain predictor:

$$\mathcal{L}_d^i(\theta_f, \theta_d) = \mathcal{L}_d(G_d(G_f(x_i; \theta_f); \theta_d), d_i)$$

The global loss function to minimize is:

$$E(\theta_f, \theta_y, \theta_d) = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}_y^i(\theta_f, \theta_y) - \lambda \left(\frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}_d^i(\theta_f, \theta_d) + \frac{1}{N_t} \sum_{i=N_s+1}^{N_s+N_t} \mathcal{L}_d^i(\theta_f, \theta_d) \right)$$

Optimisation problem

$$E(\theta_f, \theta_y, \theta_d) = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}_y^i(\theta_f, \theta_y) - \lambda \left(\frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}_d^i(\theta_f, \theta_d) + \frac{1}{N_t} \sum_{i=N_s+1}^{N_s+N_t} \mathcal{L}_d^i(\theta_f, \theta_d) \right)$$

We are looking for a saddle point:

$$\begin{aligned} (\hat{\theta}_f, \hat{\theta}_y) &= \arg \min_{(\theta_f, \theta_y)} E(\theta_f, \theta_y, \hat{\theta}_d) \\ \hat{\theta}_d &= \arg \max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d) \end{aligned}$$

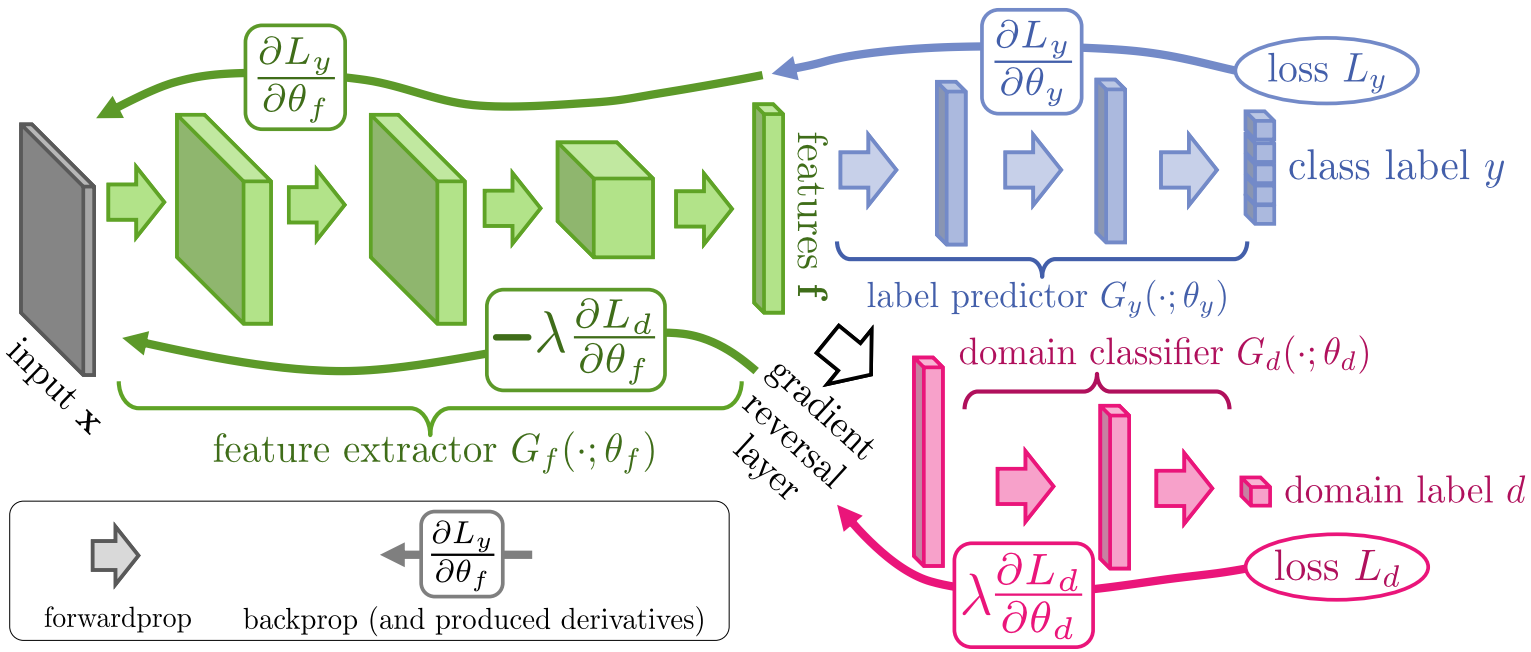
This is called **adversarial training**, as the domain predictor is competing against the feature extractor and the label predictor.

Optimisation problem: in practice

A saddle point can be found as the stationary point of the following gradient updates:

$$\theta_f \to \theta_f - \mu\left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f}\right),\ \theta_y \to \theta_y - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_y},\ \theta_d \to \theta_d - \mu\lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_d},$$

that can be implemented easily using common software by introducing a **gradient reversal layer**.



Optimisation problem: in practice

A saddle point can be found as the stationary point of the following gradient updates:

$$\theta_f \rightarrow \theta_f - \mu\left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f}\right),\ \theta_y \rightarrow \theta_y - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_y},\ \theta_d \rightarrow \theta_d - \mu\lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_d},$$

that can be implemented easily using common software by introducing a **gradient reversal layer**.

More details:

- At each iteration, each mini-batch is composed of half of samples from the source domain (with label) and half of samples from the target domain (without label).
- λ in the update of θ_f is initially set to 0 and then increased to 1.
- λ in the update of θ_d is set to 1, so that G_d and G_y are trained at the same pace.

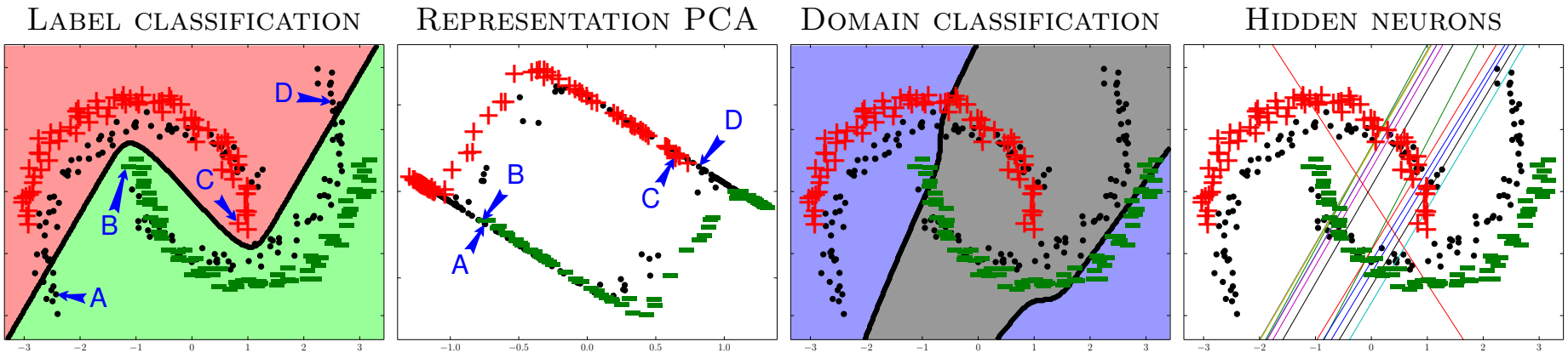
Experiments

Four types of experiments:

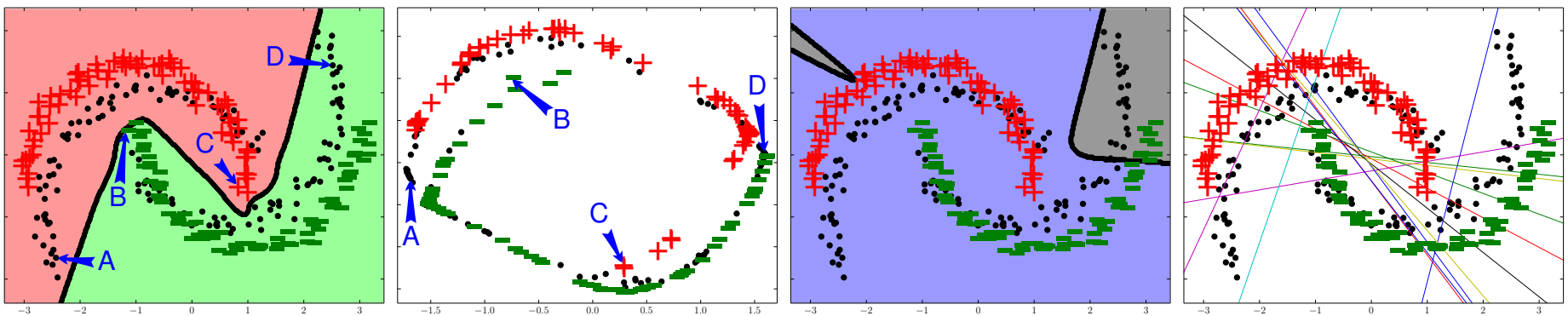
- Toy problem (inter-twinned moon)
- Sentiment analysis
- Image classification tasks
- Person re-identification

We present the main results. See the paper for full details.

Experiment: toy dataset



(a) Standard NN. For the "domain classification", we use a *non adversarial* domain regressor on the hidden neurons learned by the Standard NN. (This is equivalent to run Algorithm 1, without Lines 22 and 31)



(b) DANN (Algorithm 1)

Figure 2: The *inter-twinning moons* toy problem. Examples from the source sample are represented as a "+" (label 1) and a "—" (label 0), while examples from the unlabeled target sample are represented as black dots. See text for the figure discussion.

Architecture: one hidden layer of 15 neurons. PCA applied on the resulting 15 features.

Experiment: sentiment analysis

		Original data			mSDA representation		
Source	Target	DANN	NN	SVM	DANN	NN	SVM
books	dvd	.784	.790	**.799**	.829	.824	**.830**
books	electronics	.733	.747	**.748**	**.804**	.770	.766
books	kitchen	**.779**	.778	.769	**.843**	.842	.821
dvd	books	.723	.720	**.743**	.825	.823	**.826**
dvd	electronics	**.754**	.732	.748	**.809**	.768	.739
dvd	kitchen	**.783**	.778	.746	.849	**.853**	.842
electronics	books	**.713**	.709	.705	**.774**	.770	.762
electronics	dvd	**.738**	.733	.726	**.781**	.759	.770
electronics	kitchen	**.854**	**.854**	.847	.881	**.863**	.847
kitchen	books	**.709**	.708	.707	.718	.721	**.769**
kitchen	dvd	**.740**	.739	.736	**.789**	**.789**	.788
kitchen	electronics	**.843**	.841	.842	.856	.850	**.861**

(a) Classification accuracy on the Amazon reviews data set

Original data			
	DANN	NN	SVM
DANN	.50	**.87**	**.83**
NN	.13	.50	.63
SVM	.17	.37	.50

mSDA representations			
	DANN	NN	SVM
DANN	.50	**.92**	**.88**
NN	.08	.50	.62
SVM	.12	.38	.50

(b) Pairwise Poisson binomial test

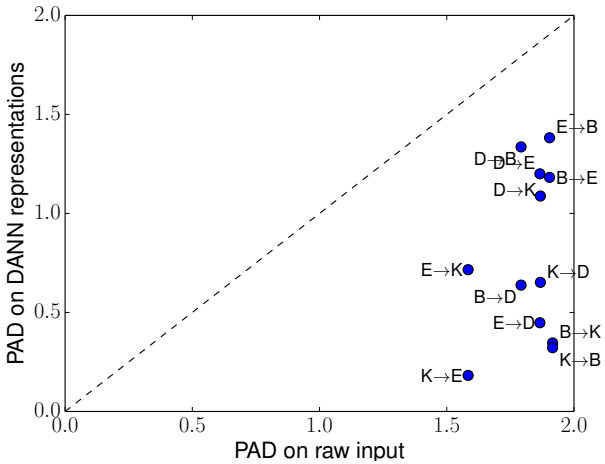
See slide 34 for problem description. NN and SVM are trained on source data only. NN and DANN use the same network architecture (a single hidden layer network).

Experiment: sentiment analysis

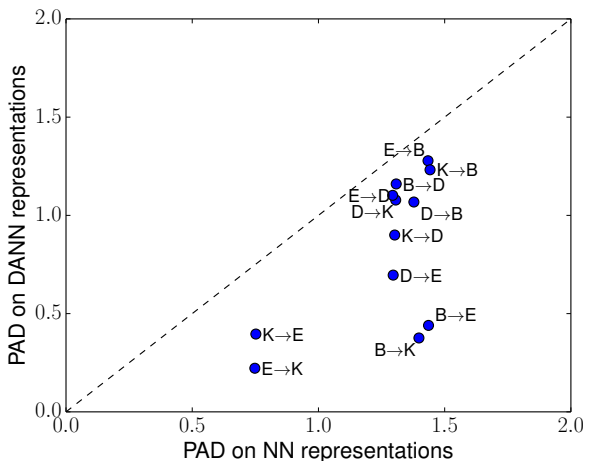
Impact of domain adversarial training on empirical distance:

$$\hat{d}_{\mathcal{H}}(LS_s^L, LS_t^U) \triangleq 2\left(1 - \min_{g \in \mathcal{H}} \left[\frac{1}{N_s}\sum_{i=1}^{N_s} 1(g(x_i^s) = 0) + \frac{1}{N_t}\sum_{i=1}^{N_t} 1(g(x_i^t) = 1)\right]\right),$$

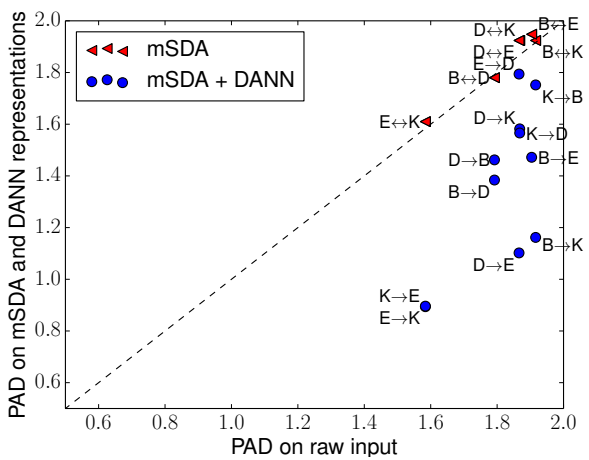
(called PAD, for *Proxy A-distance*, in the paper).



(a) DANN on *Original data*.



(b) DANN & NN with 100 hidden neurons.



(c) DANN on *mSDA representations*.

Experiment: image classification (digits)

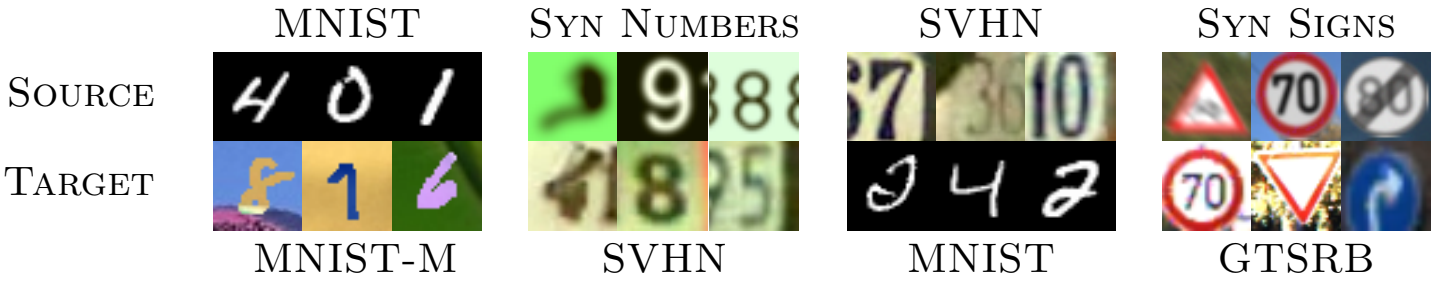
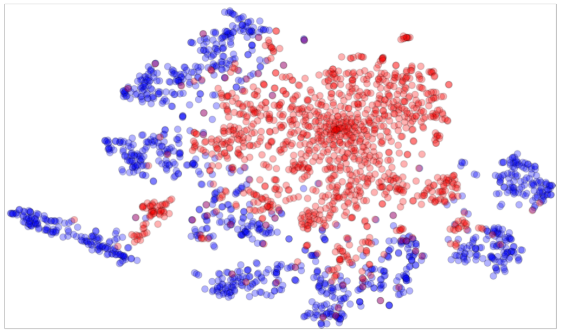


Figure 6: Examples of domain pairs used in the experiments. See Section 5.2.4 for details.

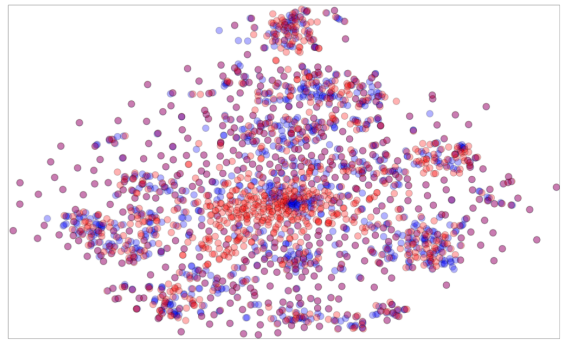
Method	Source Target	MNIST MNIST-M	Syn Numbers SVHN	SVHN MNIST	Syn Signs GTSRB
Source only		.5225	.8674	.5490	.7900
SA (Fernando et al., 2013)		.5690 (4.1%)	.8644 (−5.5%)	.5932 (9.9%)	.8165 (12.7%)
DANN		**.7666** (52.9%)	**.9109** (79.7%)	**.7385** (42.6%)	**.8865** (46.4%)
Train on target		.9596	.9220	.9942	.9980

Experiment: domain invariance of the embedding

MNIST → MNIST-M: top feature extractor layer

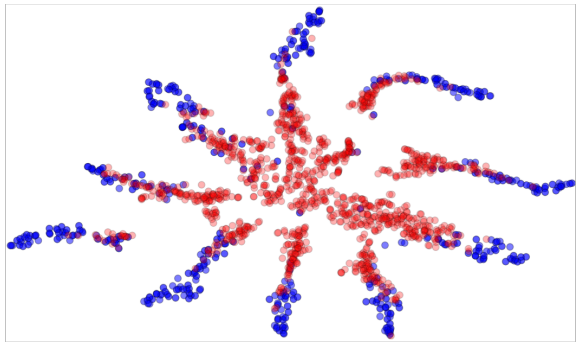


(a) Non-adapted

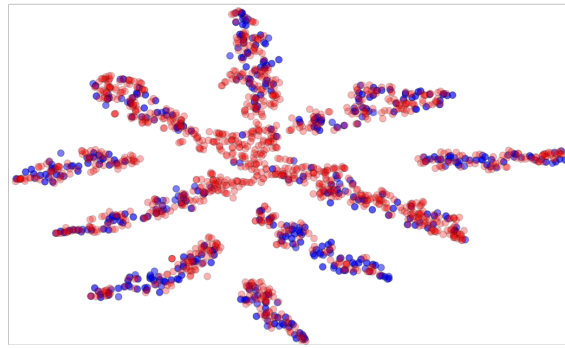


(b) Adapted

Syn Numbers → SVHN: last hidden layer of the label predictor



(a) Non-adapted

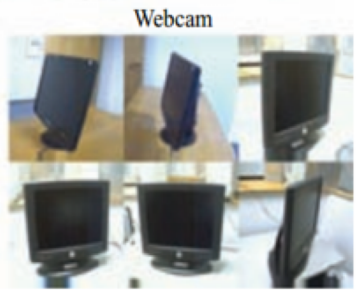


(b) Adapted

(Visualization using t-SNE)

Experiment: image classification (office)

Method	Source	Amazon	DSLR	Webcam
	Target	Webcam	Webcam	DSLR
GFK(PLS, PCA) (Gong et al., 2012)		.197	.497	.6631
SA* (Fernando et al., 2013)		.450	.648	.699
DLID (Chopra et al., 2013)		.519	.782	.899
DDC (Tzeng et al., 2014)		.618	.950	.985
DAN (Long and Wang, 2015)		.685	.960	.990
Source only		.642	.961	.978
DANN		**.730**	**.964**	**.992**

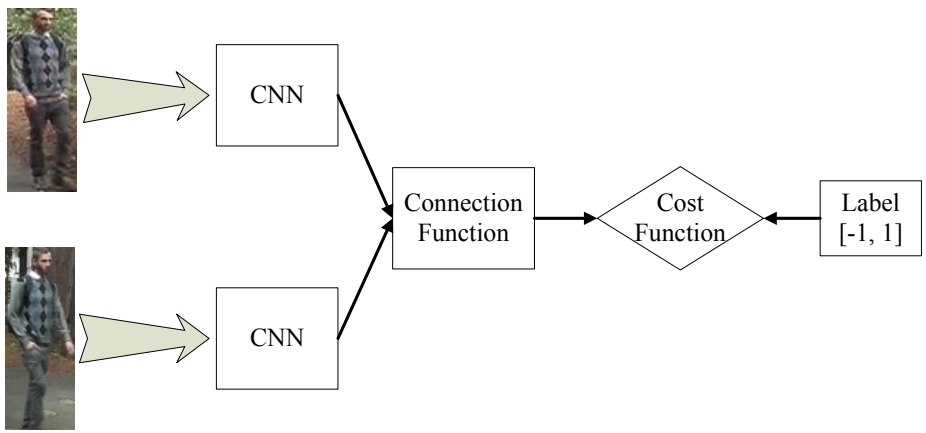


Experiment: person re-identification

Problem: given an image of a person, find the same person in a gallery of images

Source

Typically solved as a metric learning problem. E.g., using siamese network:

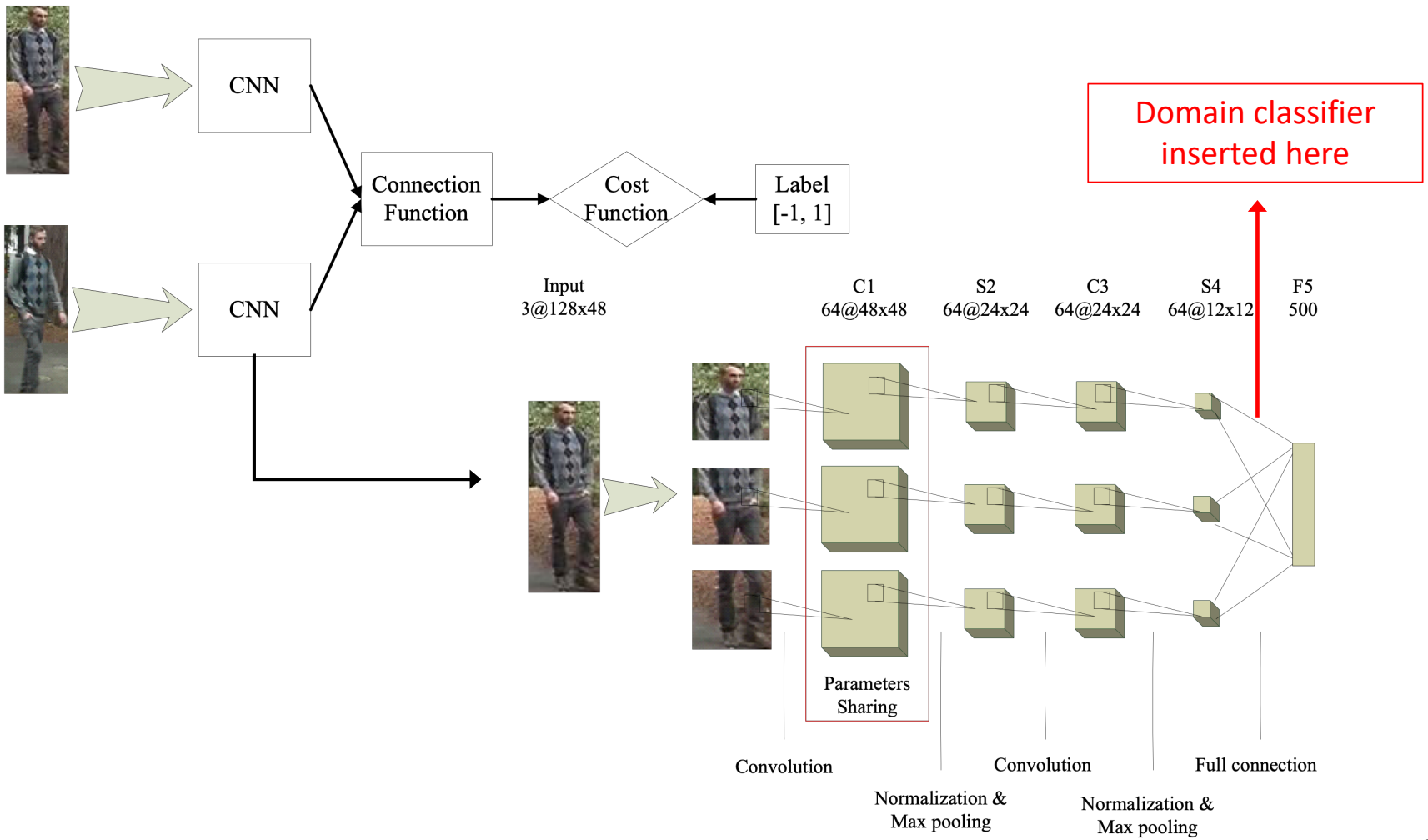


(Yi et al., 2014)

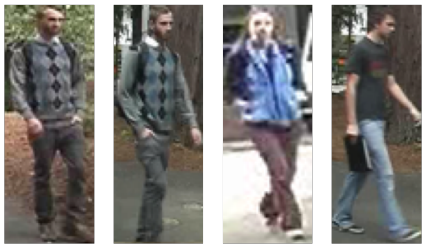
Experiment: person re-identification

Problem: images are often taken with (very) dissimilar cameras (= domains).

Proposed domain adversarial training approach:



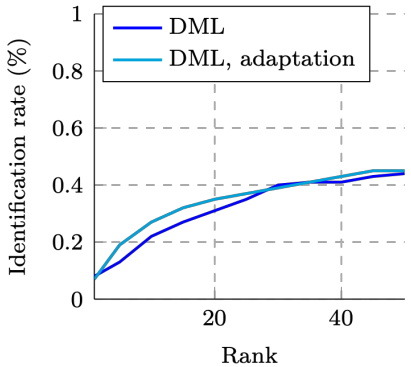
Experiment: person re-identification



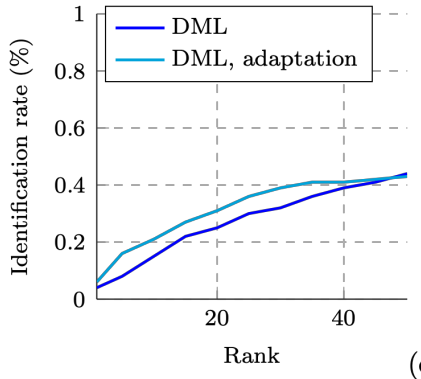
VIPER

PRID

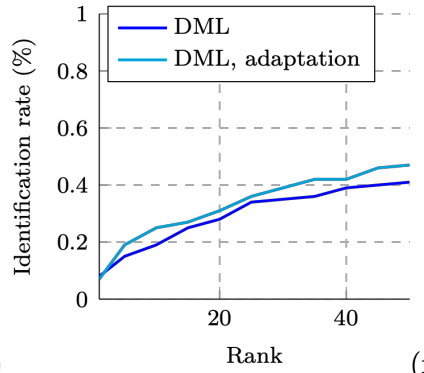
CUHK



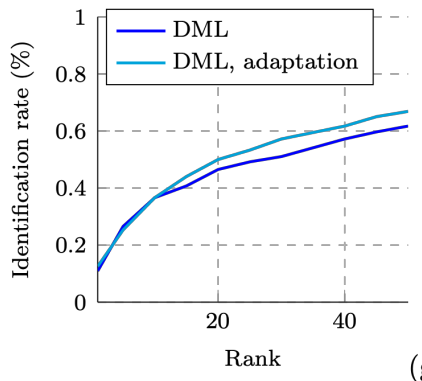
(d) Whole CUHK → PRID



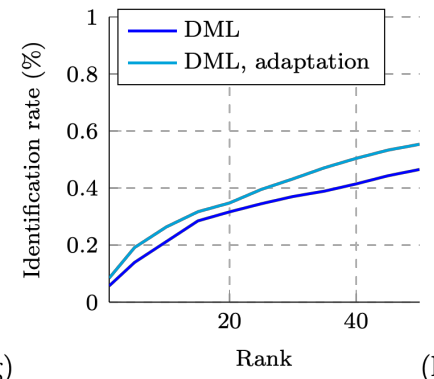
(e) CUHK/p1 → PRID



(f) VIPeR → PRID



(g) VIPeR → CUHK/p1



(h) PRID → CUHK/p1

Figure 9: Results on VIPeR, PRID and CUHK/p1 with and without domain-adversarial learning. Across the eight domain pairs domain-adversarial learning improves re-identification accuracy. For some domain pairs the improvement is considerable.

Discussion

- The proposed method is well principled and very simple to implement.
- Experiments are thorough and convincing.
- The idea is generic: a similar domain adaptation component can be added to many neural network architectures.

- Theory does not guarantee that domain adversarial training will work:
  - This is only an upper bound
  - Only part of the bound is actually minimized
  - Most probably, explicitely minimizing the bound violates the theorem's hypotheses.
- The paper could have explored cases of failures, e.g. on artificial data.
- Several more recent works have obtained better results than DANN on several tasks

A recent extension: DIRT-T (Shu et al., 2018)

Shu et al. (2018) show that domain adversarial training might fail (by overfitting the source domain), when $G_f(.;\theta_f)$ is too powerful and source-target supports are disjoint.

General idea of their approach: incorporate as a constraint that the decision boundary should not cross high-density data regions in the target domain.

Some results on the digit classification tasks:

Source Target	MNIST MNIST-M	SVHN MNIST	MNIST SVHN	DIGITS SVHN	SIGNS GTSRB	CIFAR STL	STL CIFAR
With Instance-Normalized Input:							
Source-Only	59.9	82.4	40.9	88.6	86.2	77.0	62.6
DANN (our implementation)	94.6	68.3	60.6	90.1	97.5	78.1	62.7
$\text{VADA}_{\text{no-vat}}$	93.8	83.1	66.8	93.4	98.4	**79.1**	68.6
$\text{VADA}_{\text{no-vat}} \rightarrow \text{DIRT-T}_{\text{no-vat}}$	94.8	96.3	68.6	94.4	99.1	-	69.2
$\text{VADA}_{\text{no-vat}} \rightarrow$ DIRT-T	**98.3**	**99.4**	**69.8**	**95.3**	**99.6**	-	**71.0**
VADA	95.7	94.5	73.3	94.9	99.2	**78.3**	71.4
VADA $\rightarrow$ DIRT-T	**98.7**	**99.4**	**76.5**	**96.2**	**99.6**	-	**73.3**

Table 4: Test set accuracy in ablation experiments, starting from the DANN model. The "no-vat" subscript denote models where the virtual adversarial training component is removed.