Lecture 2

Covariate and Label shifts

CS329D

Goals for today

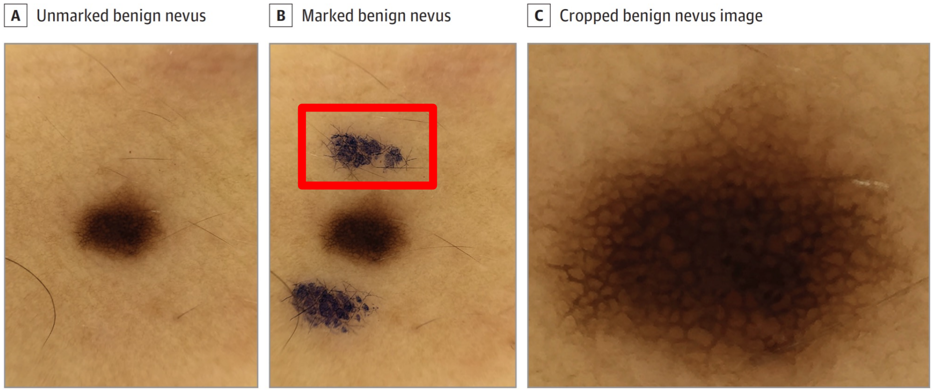
Define covariate shifts + label shift

Understand importance-weighting estimators

Know how covariate shifts relate to spurious correlations

Motivating example

(Recap): Medical example of surgical skin markers



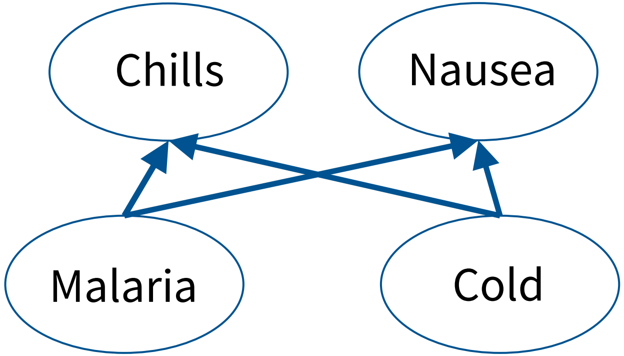
Training data: image with markings

Test data: image without markings

Key point: the removal of features (marker) leads to performance loss

Another example

New example: Is it malaria?

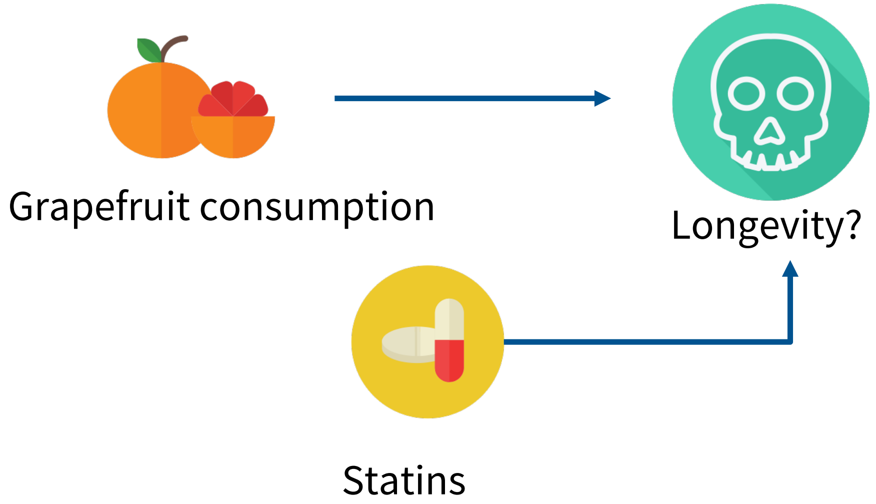


Training data: from the north pole

Test data: from the amazon

Key point: disease prevalence change results in different predictions

One final example

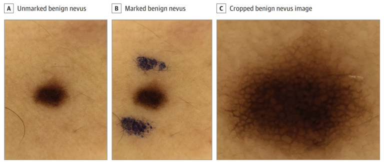


Training data: no statins

Test data: statins

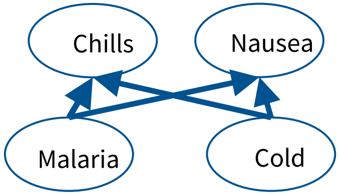
Key point: correlations between features and labels differ

Taxonomy of distribution shifts (non-exhaustive)



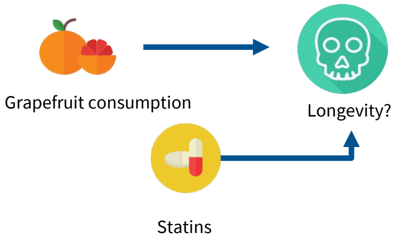
Covariate shift

- The input distribution changes, labels given inputs does not.
- **Litmus test:** Is there a single predictor that does well on train and test?



Label (prior probability) shift

- The only change is to the **label frequency.**
- **Litmus test:** Is the optimal predictor the same up to label frequency?

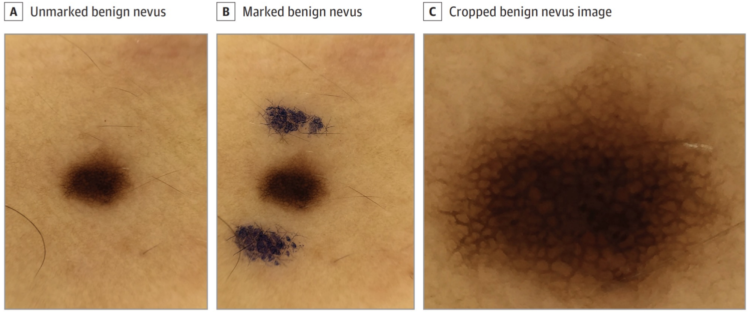


Concept drift

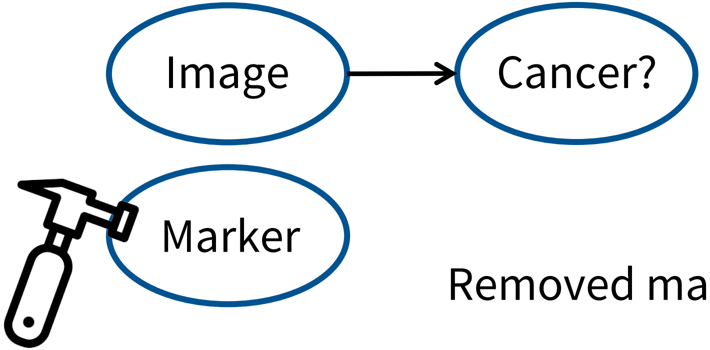
- The prediction function changes from train to test, the inputs do not.
- **Litmus test:** is the optimal predictor different?

Definition: covariate shift

Recall our example:



Representing this shift:

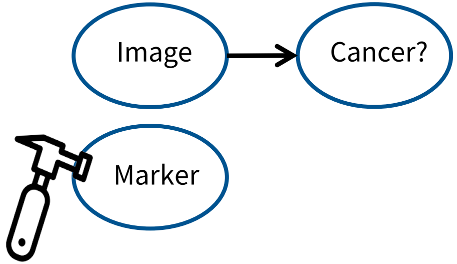


Removed marker, rest remains the same

Defining covariate shift

Representing this shift:

$$p_{test}(cancer, image, marker) = p_{train}(cancer|image, marker)p_{test}(image, marker)$$



Definition:

A prediction problem $x \to y$ is called a *covariate shift* whenever the training distribution p_{train} and test distribution p_{test} follows

$$p_{test}(y|x) = p_{train}(y|x) \qquad p_{test}(x) \neq p_{train}(x)$$

(some texts also require that $x \to y$ be the true data generating distribution)

Covariate shifts in machine learning

Why is covariate shift so important?

1. Prediction problems fit well with covariate shift.

Supervised learning: estimate $p(y|x)$
Covariate shift: $p(y|x)$ remains fixed

2. Annotator-driven data collection

Input data

Labels

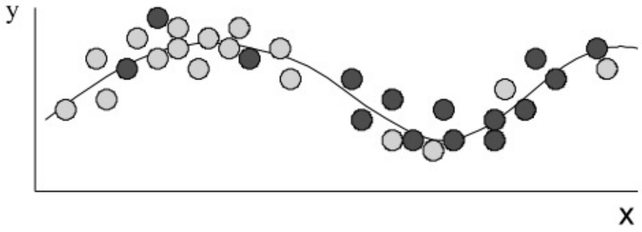
DOG

$p(y|x)$ – defined by annotators

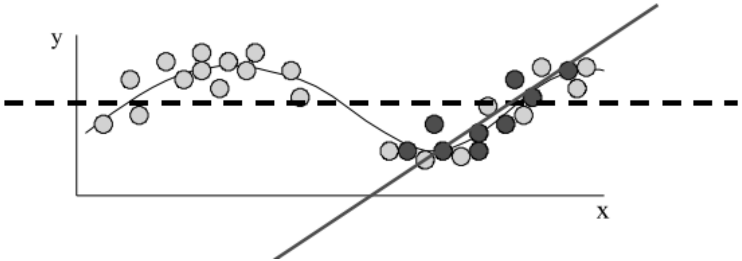
Fixed annotators → covariate shift

Bayes optimal predictor remains identical

Bayes optimal predictor depends only on $p(y|x)$, which doesn't change



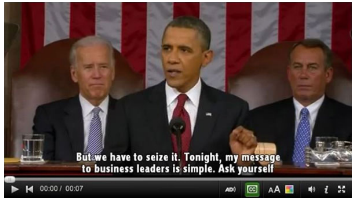
This is **not** true under misspecification (i.e., when you model cant be Bayes optimal)



Storkey 2009

More examples of covariate shift in ML

Covariate shift due to human annotators



Video captioning

1. @username R u a wizard or wat gan sef: in d mornin - u tweet, afternoon - u tweet, nyt gan u dey tweet. beta get ur IT placement wiv twitter
2. Be the lord lantern jaysus me heart after that match!!!
3. Aku hanya mengagumimu dari jauh sekarang . RDK ({}) * last tweet about you -_- , maybe

Figure 1: Challenges for socially-equitable LID in Twitter include dialectal text, shown from Nigeria (#1) and Ireland (#2), and multilingual text (Indonesian and English) in #3.

Language identification

Object detection

Premise:

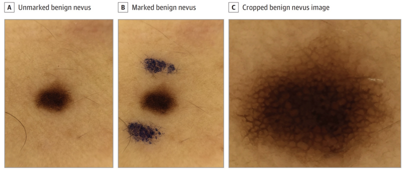
The economy could be still better.

Hypothesis:

The economy has never been better

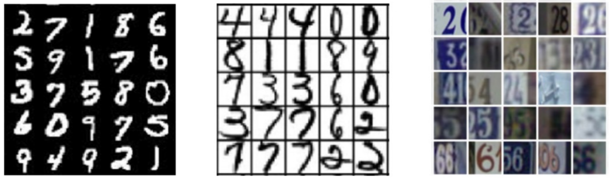
Entailment

Other covariate shifts



Tumor detection (marker)

MNIST USPS SVHN

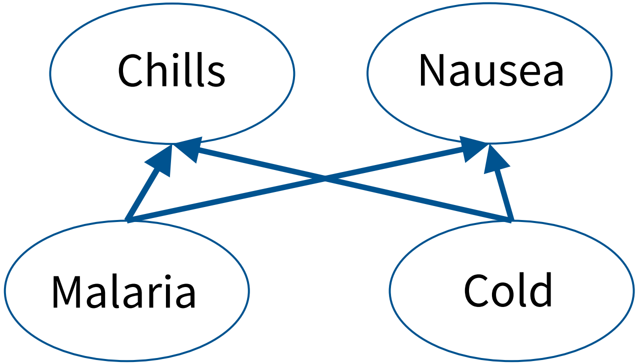


Style changes

Definition: label shift

Recall the example:

$$p_{test}(malaria, symptoms) = p_{train}(symptoms|malaria)p_{test}(malaria)$$



Definition:

A prediction problem $x \rightarrow y$ is called a *label shift* whenever the training distribution p_{train} and test distribution p_{test} follows

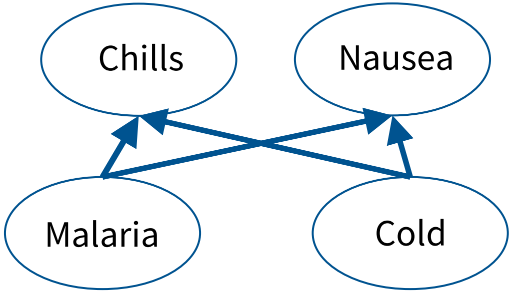
$$p_{test}(x|y) = p_{train}(x|y) \qquad p_{test}(y) \neq p_{train}(y)$$

(also called prior probability shift)

(some texts also require that y $\rightarrow x$ be the true data generating distribution)

Label shifts in machine learning and examples

Medical diagnostics



Diseases (y) cause symptoms (x)

Label-driven-data collection

Labels
(flickr tag)

Image

DOG

Labels (y) are under selection bias

Label shifts for 'deterministic' predictions problems are also covariate shifts.

Sometimes label shifts are also covariate shifts

Labels

(flickr tag)

Image

Covariate shift because..

$$p_{test}(y|x) = p_{train}(y|x)$$

From a deterministic labeling map

Label shift shift because..

$$p_{test}(x|y) = p_{train}(x|y)$$

From selection on the tags

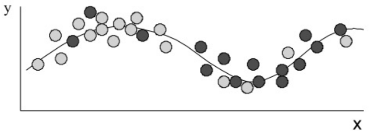
In these cases we can pick the easier type of shift (often label)

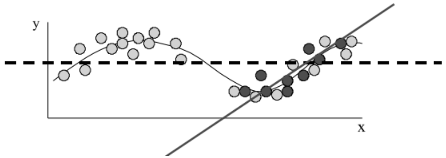
Covariate + label shift: summary

1. Covariate and label shifts are defined by **what stays fixed**

Covariate shift	**Label shift**
$p_{test}(y\|x) = p_{train}(y\|x)$	$p_{test}(x\|y) = p_{train}(x\|y)$

2. Misspecification is key to covariate shift





3. These categories are flexible – some distributions can be both covariate + label shift

Labels
(flickr tag)

Image

DOG

Reweighting: setup

How do we deal with covariate / label shifts?

What we can compute

$$E_{p_{train}}[\ell(z;\theta)]$$

The loss function ℓ computed on examples $z \coloneqq (x, y)$ and model θ where z comes from p_{train}

What we want

$$E_{p_{test}}[\ell(z;\theta)]$$

Reweighting

How do we deal with covariate / label shifts?

What we have

$$E_{p_{train}}[\ell(z;\theta)]$$

What we want

$$E_{p_{test}}[\ell(z;\theta)]$$

Most basic approach: reweight the loss

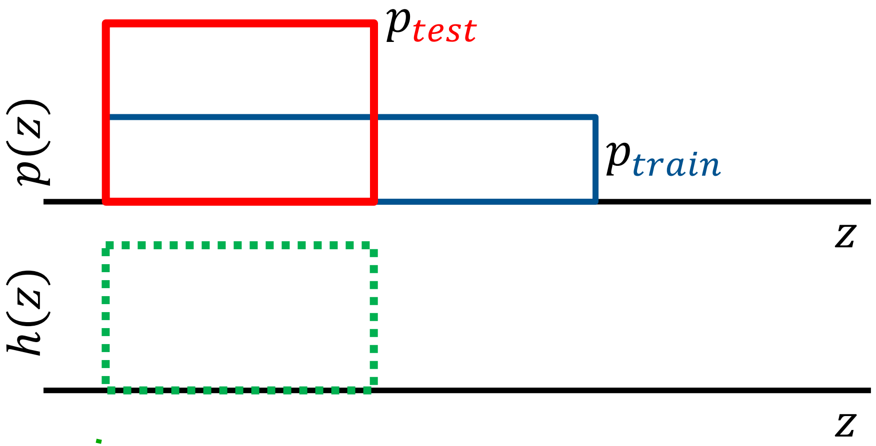
$$\underbrace{E_{p_{train}}\left[\frac{p_{test}(z)}{p_{train}(z)}\ell(z;\theta)\right]}_{\text{Weighted loss over the training distribution}} = E_{p_{test}}\left[\ell(z;\theta)\right]$$

(also possible: resample the dataset)

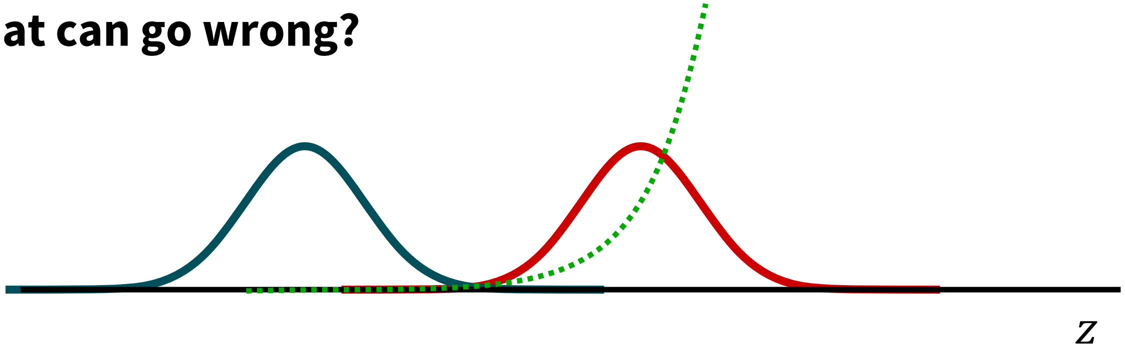
On the importance of overlap

High level idea:

1. Compute $h = \frac{p_{test}}{p_{train}}$
2. Reweight the loss.



What can go wrong?



Reweighting requires substantial overlap

Variance bounds

Variance bounds show this formally.

Let $h(z) = \frac{p_{test}(z)}{p_{train}(z)}$

$$Var[h(z)\ell(z;\theta)] = Var\big(\ell(z;\theta)\big) + E_{p_{test}}[(h(z)-1)\ell(z;\theta)^2]$$

Note: $E_{p_{test}}[h(z)] > 1$

By Holder's inequality

$$\le \underbrace{Var\big(\ell(z;\theta)\big)}_{\text{Test dist. variance}} + \underbrace{(|h(z)|_\infty - 1)}_{\text{Importance weight size}} E_{p_{test}}[\ell(z;\theta)^2]$$

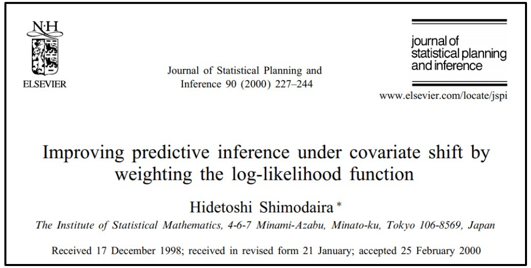
Remember: $|h(z)|_\infty \to \infty$ with decreasing overlap

Handling covariate shift: a simple algorithm

A proto-algorithm to handle covariate shift:

If we have unlabeled data from the test distribution,

1. Estimate the density $p_{test}(x)$ and $p_{train}(x)$
2. Reweight by $h(x) = \frac{p_{test}(x)}{p_{train}(x)}$
3. Fit a model by minimizing the loss $h(x)\ell(x, y; \theta)$


N·H
ELSEVIER
Journal of Statistical Planning and Inference 90 (2000) 227–244
journal of statistical planning and inference
www.elsevier.com/locate/jspi

Improving predictive inference under covariate shift by weighting the log-likelihood function

Hidetoshi Shimodaira *
The Institute of Statistical Mathematics, 4-6-7 Minami-Azabu, Minato-ku, Tokyo 106-8569, Japan

Received 17 December 1998; received in revised form 21 January; accepted 25 February 2000

Using discriminators for the same task

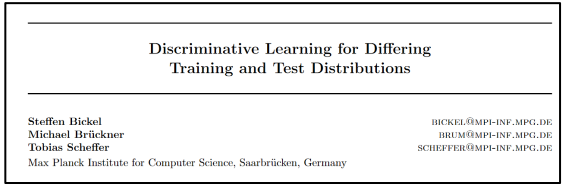
Density estimation is very hard, and classification might be easier

An alternative algorithm: use a classifier that separates p_{train} and p_{test}

1. Estimate a classifier $f(x) \approx \frac{p_{train}(x)}{p_{test(x)}+p_{train}(x)}$

2. Reweight by $h(x) = \frac{1}{f(x)} - 1$

3. Fit a model by minimizing the loss $h(x)\ell(x, y; \theta)$



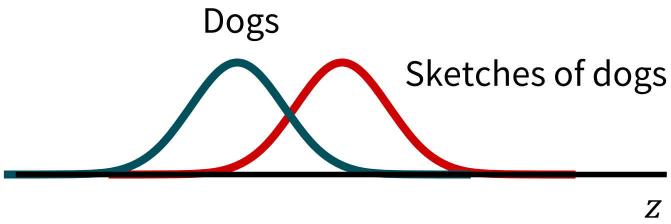
Discriminative Learning for Differing
Training and Test Distributions

Steffen Bickel BICKEL@MPI-INF.MPG.DE
Michael Brückner BRUM@MPI-INF.MPG.DE
Tobias Scheffer SCHEFFER@MPI-INF.MPG.DE
Max Planck Institute for Computer Science, Saarbrücken, Germany

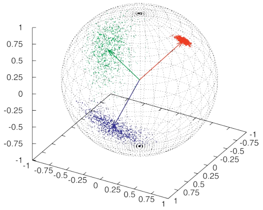
Handling covariate shift: challenges

Inputs are often high-dim (hard to find overlap)

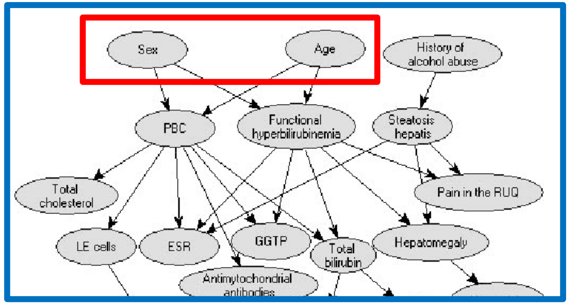
Mental model (low dim, high overlap)



Reality (high dim, no overlap)



Issues inherent to covariate shift



Tradeoff between high overlap, non-covariate shift and no overlap covariate shift

Handling label shift

If we have unlabeled data from the test distribution,

1. Estimate the density $\boldsymbol{p_{test}(y)}$ **and** $\boldsymbol{p_{train}(y)}$
2. Reweight by $h(y) = \frac{p_{test}(y)}{p_{train}(y)}$
3. Fit a model by minimizing the loss $h(y)\ell(x, y; \theta)$

Challenge: we don’t have the label distribution

A label distribution estimator

Basic idea: use a predicted labels $\hat{y} \coloneqq f(x)$ to get a good estimate.

Confusion matrix

$$C_{i,j} \coloneqq p_{train}(y = i, \hat{y} = j)$$

Notice that

$$p_{test}(\hat{y} = j) = \sum p_{test}(\hat{y} = j | y = i) p_{test}(y = i)$$

$$= \sum p_{train}(\hat{y} = j | y = i) p_{test}(y = i)$$

$$= \sum \frac{C_{i,j} p_{test}(y = i)}{p_{train}(y = i)}$$

$$\begin{aligned} & p_{test}(\hat{y}|y) \\ &= p(\hat{y}|x)p(x|y) \\ &= p_{train}(\hat{y}|y) \end{aligned}$$

Plug-in estimate for label bias correction

If we have unlabeled data from the test distribution,

1. Make an estimator $f(x)$ on the training set
2. Compute the confusion matrix **C** on the training set
3. Estimate the density $\boldsymbol{p_{test}(f(x))}$ on the **test** set
4. Reweight by $h(y) = C^{-1} p_{test}(\hat{y})$
5. Fit a model by minimizing the loss $h(y)\ell(x, y; \theta)$

Intuition: confusion matrix remains fixed

NOTE Communicated by Leo Breiman

Adjusting the Outputs of a Classifier to New *a Priori* Probabilities: A Simple Procedure

Marco Saerens
saerens@ulb.ac.be
IRIDIA Laboratory, cp 194/6, Université Libre de Bruxelles, B-1050 Brussels, Belgium, and SmalS-MvM, Research Section, Brussels, Belgium

[Lipton 2018, Saerens 2002]

An alternative, EM based label shift estimator

What is the log-likelihood of observing $x \sim p_{test}$?

$$E\left[\log \sum p_{train}(x|y)\, p_{test}(y)\right]$$

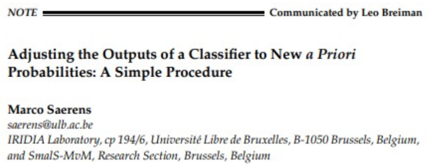
Idea: maximize the log-likelihood with respect to $p_{test}(y)$

Algorithm: for t in 0… T

1. E-step: $q^t(y|x) \propto \frac{q^t(y)}{p_{train}(y)} p_{train}(y|x)$

2. M-step: $q^{t+1}(y) = E[q^t(y|x)]$

(Clever trick here: we don't need to instantiate $p_{train}(x|y)$)


NOTE Communicated by Leo Breiman

Adjusting the Outputs of a Classifier to New *a Priori* Probabilities: A Simple Procedure

Marco Saerens
saerens@ulb.ac.be
IRIDIA Laboratory, cp 194/6, Université Libre de Bruxelles, B-1050 Brussels, Belgium, and SmalS-MvM, Research Section, Brussels, Belgium

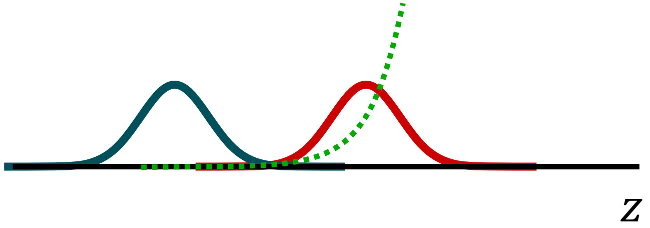
[Saerens 2002]

Summary: reweighting

1. Reweighting is a basic but widely applicable way of handling distribution shifts

$$E_{p_{train}}\left[\frac{p_{test}(z)}{p_{train}(z)}\ell(z;\theta)\right] = E_{p_{test}}\left[\ell(z;\theta)\right]$$

2. Overlap is extremely important for the success of reweighting methods

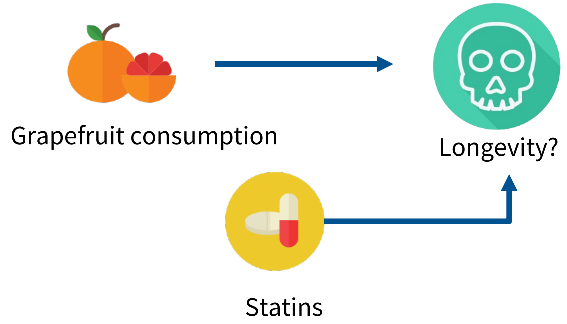


3. Key task: estimating the probability ratio. We covered 4 basic approaches.

Covariate shift		Label shift	
Direct density estimate	Discriminative	Plug-in estimate	EM/Likelihood

Beyond covariate and label shifts

What if a distribution is neither covariate or label shift?



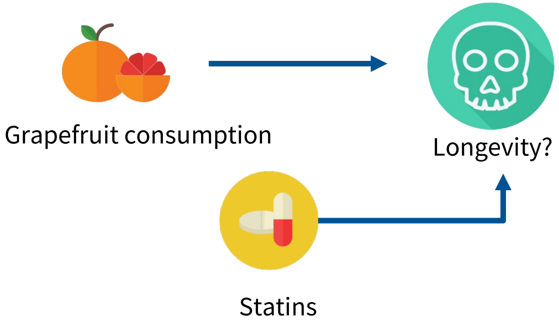
$$p_{test}(\text{longevity}|\text{grapefruit}) \neq p_{train}(\text{longevity}|\text{grapefruit})$$

$$p_{test}(\text{grapefruit}) = p_{train}(\text{grapefruit})$$

Note: often a catch-all, and also *very* difficult to deal with

Concept shift and unobserved confounding

Concept shifts usually arise from unobserved features:



$p(\text{longevity} | \text{ grapefruit, statins})$ is fixed

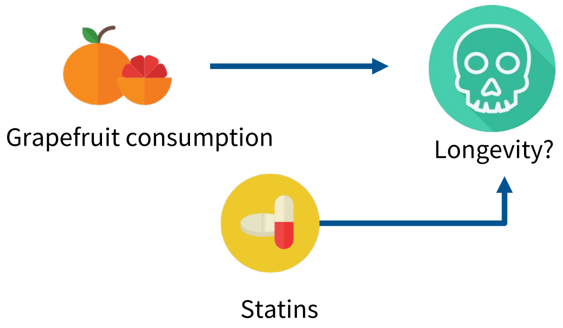
$p(\text{longevity} | \text{ grapefruit})$ is **not** fixed

In the **sample selection bias** case: we select on unmeasured features

In the **environment change** case: label y is affected by unmeasured features

Spurious correlations: getting the model involved

Thus far: statins are treated as unobserved.



Next: What if our model doesn't use the 'right' features?

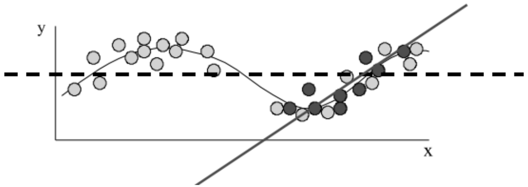
If our learning algorithm ignores statins,
it doesn't matter that its observed in the dataset

Spurious correlations

Definition (just for this class)

A learning algorithm L learns a spurious correlation for a distribution shift if
- p_{train} and p_{test} satisfy the covariate shift assumption
- L outputs different predictors on p_{train} and p_{test}

Example:

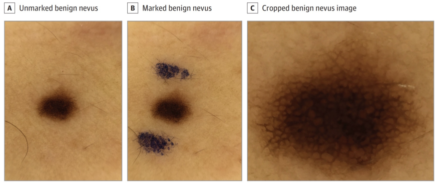


Important notes:
- No consensus definition in the field
- "Outputs different predictors" is likely too strong
- Not always possible to avoid or mitigate

Examples of spurious correlation.

Watermarks

Surgical markers



Fishing boat identity

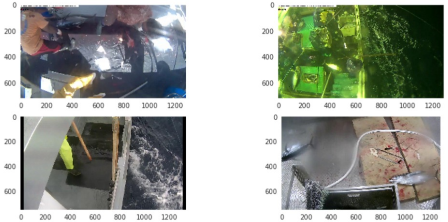
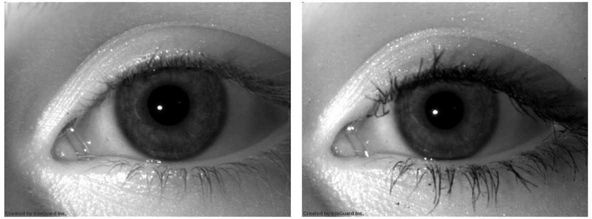


Image backgrounds

Mascara



(a) Eye without mascara (b) Eye with mascara

Recap: covariate + label shifts

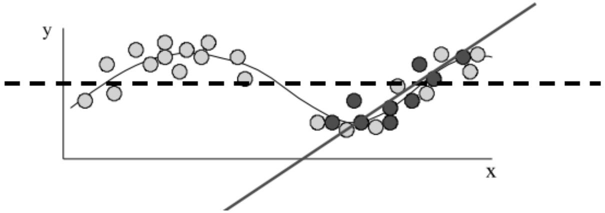
- Covariate and label shift definitions:

Covariate shift	Label shift
$p_{test}(y\|x) = p_{train}(y\|x)$	$p_{test}(x\|y) = p_{train}(x\|y)$

- Reweighting (key part: estimating h)

$$E_{p_{train}}\left[\frac{p_{test}(z)}{p_{train}(z)}\ell(z;\theta)\right] = E_{p_{test}}\left[\ell(z;\theta)\right]$$

- Spurious correlations and concept shift (hard to handle!)



Lecture 3

DOMAIN DISTANCES AND H-DELTA-H

CS329D

Goals for today

Understand divergence based adaptation bounds

Be able to state H-delta-H guarantees and limitations

Know about transformation-based approaches to domain adaptation

Our setting: unsupervised domain adaptation

Task: identifying digits from images

Training data

MNIST

Test domains

USPS

SVHN

Another common scenario: sim to real transfer

Task: identifying cars in a video

Training data (GTA)

Test data (real world)

Problem definition: unsupervised domain adaptation

Task: prediction problem $x \rightarrow y$, model class $\theta \in \mathcal{H}$ with loss $\ell(x, y, \theta)$

Given: supervised data $(x, y) \sim p_{train}$ **and** unlabeled data $x' \sim p_{test}$

Goal: Minimize the expected loss

$$E_{p_{test}}[\ell(x, y, \theta)]$$

Assumption: Covariate shift

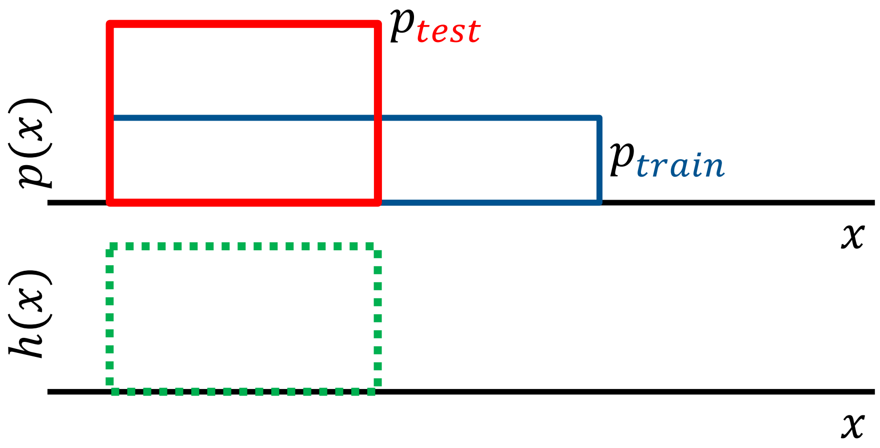
$$p_{train}(y|x) = p_{test}(y|x)$$

Starting point: reweighting

Recall: reweighting the loss

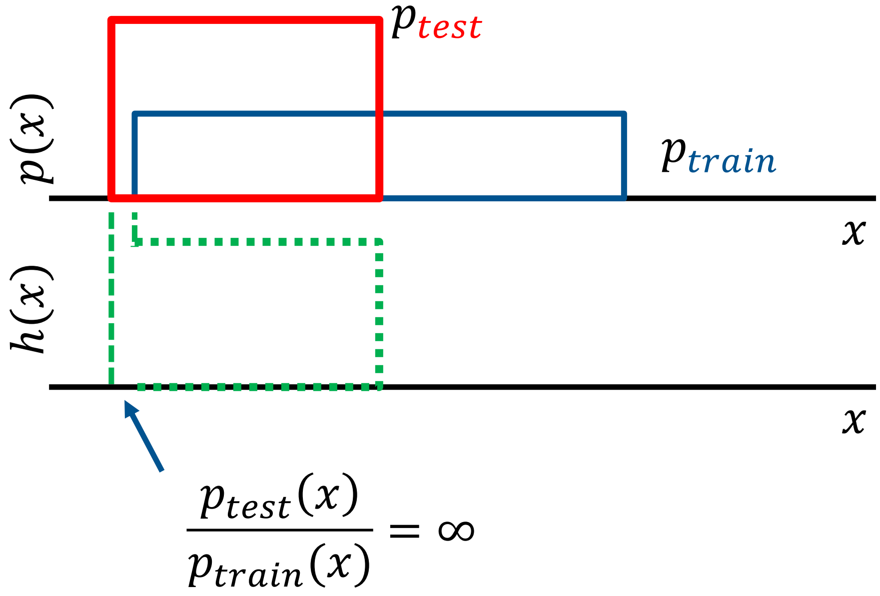
High level idea:

1. Estimate $h = \frac{p_{test}}{p_{train}}$
2. Minimize $h(x)\ell(x, y, \theta)$



Starting point: Reweighting without overlap

Consider the following situation:

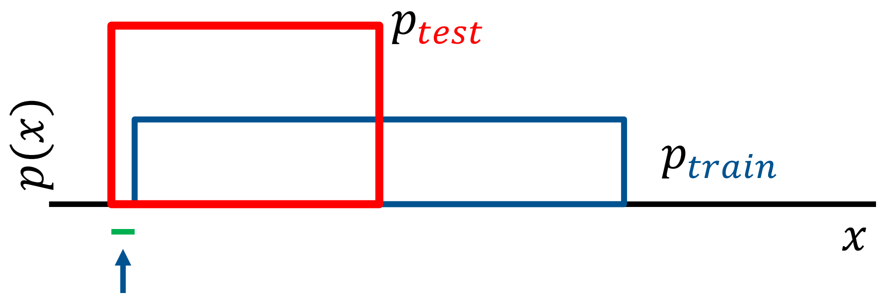


Reweighting estimators blow up without overlap

We should be able to learn without overlap

Is the situation hopeless? **No! We need to go beyond reweighting**

What if this is a classification problem?



Non-overlapping part is at most ϵ probability

Key observation: even if we have 100% error rate, this increases error by at most ϵ

$$\text{Our hope: Test error} = \text{Train error} + \epsilon$$

Going beyond reweighting.

Reweighting

adjust p_{train} to match p_{test} at *all costs*

IPMs (next up)

how bad do we do if we ignore the mismatch?

Note: we're going to focus on classification for the rest of this lecture

Background material: integral probability measures

To state this clearly, we need to first go into some background.

Definition (IPM):

For two probability distributions p and q, the integral probability metric (IPM) for a family of functions $\mathcal{F}$ is defined as

$$d_{\mathcal{F}}(p, q) = \sup_{f \in \mathcal{F}} |E_p[f(x)] - E_q[f(x)]|$$

Intuition: $\mathcal{F}$ are 'test functions' that can distinguish p and q

If two have the same function value for all $\mathcal{F}$, then they are similar

Important examples of IPMs

IPMs occur in many different areas of ML!

Class $\mathcal{F}$	**IPM**
Bounded functions in [0,1]	Total variation
Lipschitz continuous functions	Wasserstein distances
Reproducing kernel Hilbert spaces	Maximum mean discrepancy (MMD)
Bounded and Lipschitz functions	Dudly metric

Domain adaptation under small IPM

What we want | What we have | Domain distance

$$E_{p_{test}}[\ell(x,y,\theta)] = E_{p_{train}}[\ell(x,y,\theta)] + \Delta$$

From the trivial restatement

$$\Delta = E_{p_{test}}[\ell(x,y,\theta)] - E_{p_{train}}[\ell(x,y,\theta)]$$

This looks like an IPM! (if $\ell(x,y,\theta) \in \mathcal{F}$ for all θ)

$$\Delta \leq \sup_{f \in \mathcal{F}} E_{p_{test}}[f(x,y)] - E_{p_{train}}[f(x,y)] = d_{\mathcal{F}}(p_{train}, p_{test})$$

Takeaway: IPMs bound excess error under transfer

Key idea: distinguishability bound approximation error

Let's apply this to unsupervised domain adaptation for classification.

$$E_{p_{test}}[\ell(x,y,\theta)] = E_{p_{train}}[\ell(x,y,\theta)] + \Delta$$

$$\Delta := E_{x\sim p_{test}}\underbrace{\left[E_{y|x}[\ell(x,y,\theta)]\right]}_{f(x)} - E_{x\sim p_{train}}\left[E_{y|x}[\ell(x,y,\theta)]\right]$$

Notice $0 \leq E_{y|x}[\ell(x,y,\theta)] \leq 1$ for classification (zero-one loss).
If we pick $\mathcal{F}$ as the bounded functions..

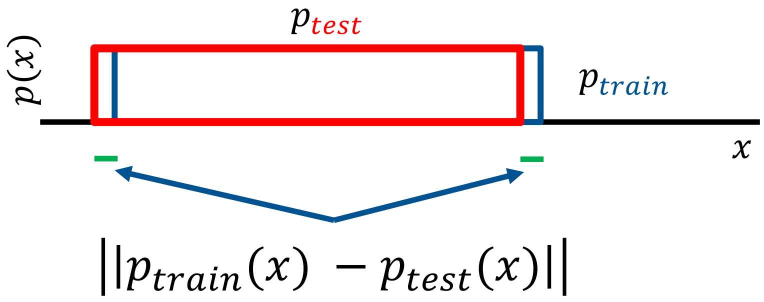
$$\Delta \leq \mathrm{d}_{\mathcal{F}}\left(p_{train,} p_{test}\right) = \left|\left|p_{train}(x) - p_{test}(x)\right|\right|_1$$

IPM based bound on target domain performance

We can now bound test performance in terms of IPMs

For $0 \leq \ell(x, y, \theta) \leq 1$ and under covariate shift,

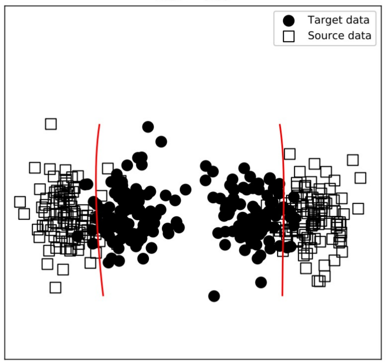
$$E_{p_{test}}[\ell(x, y, \theta)] \leq E_{p_{train}}[\ell(x, y, \theta)] + ||p_{train}(x) - p_{test}(x)||_1$$



Compare and contrast with reweighting based ideas

	Reweighting	IPM
Goals	Correct train-test mismatch	Estimate train-test mismatch
Assumptions	Overlap	Boundedness
Training	Weighted/modified loss	No change
Costs	More samples (variance)	Inaccurate models (bias)

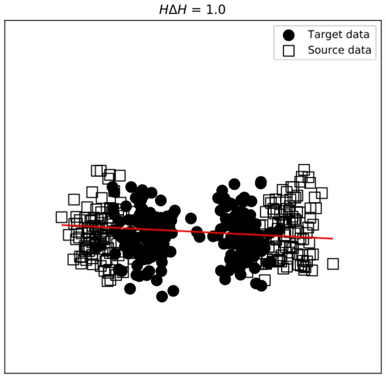
Beyond IPMs : using the model class



The current story:

Cost of domain shift $= L_1$ distinguishability

L_1 is extremely pessimistic



Idea:

Can we get bounds for our hypothesis class?

Intuition:

Domains that ‘look similar’ to our classifier
should result in similar performance

From IPM to HΔH

In the IPM case,

$$\Delta := E_{x\sim p_{test}} \underbrace{\left[E_{y|x}[\ell(x,y,\theta)]\right]}_{f(x)} - E_{x\sim p_{train}}\left[E_{y|x}[\ell(x,y,\theta)]\right]$$

Interpreting: $f(x)$ is the error at a point x

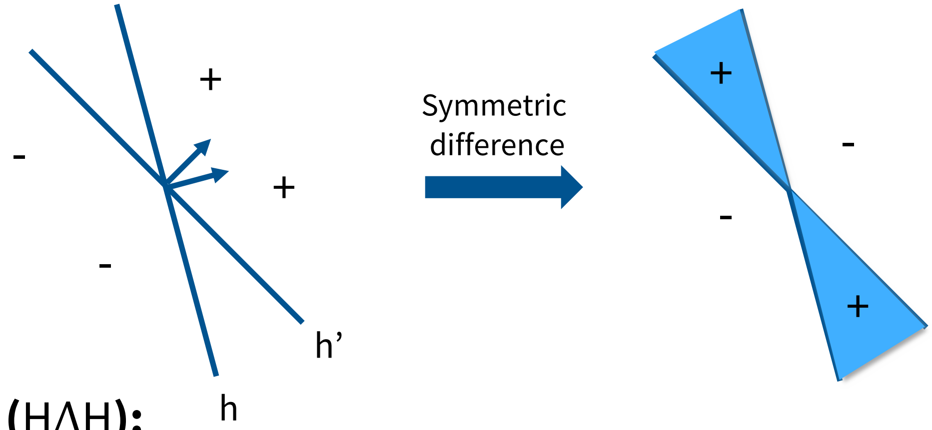
we can think of this as the 'disagreement' between our classifier θ and the ground truth

What should our f(x) family look like for a hypothesis class $\mathcal{H}$?

Answer: disagreement between any two elements $h, h' \in \mathcal{H}$

Defining HΔH

For a hypothesis class $\mathcal{H}$, the HΔH set is defined as the symmetric difference



Definition (HΔH):

For a hypothesis class $\mathcal{H}$, the symmetric difference set $H\Delta H$ is defined as

$$\mathrm{H\Delta H} \coloneqq \{g\colon g(x) = \mathrm{XOR}\big(h(x), h'(x)\big) \text{ and } h, h' \in \mathcal{H}\}$$

HΔH as a divergence

Now we can treat HΔH as a function class, and define a (pseudo) metric

Definition (HΔH-divergence [Ben-David]**):**

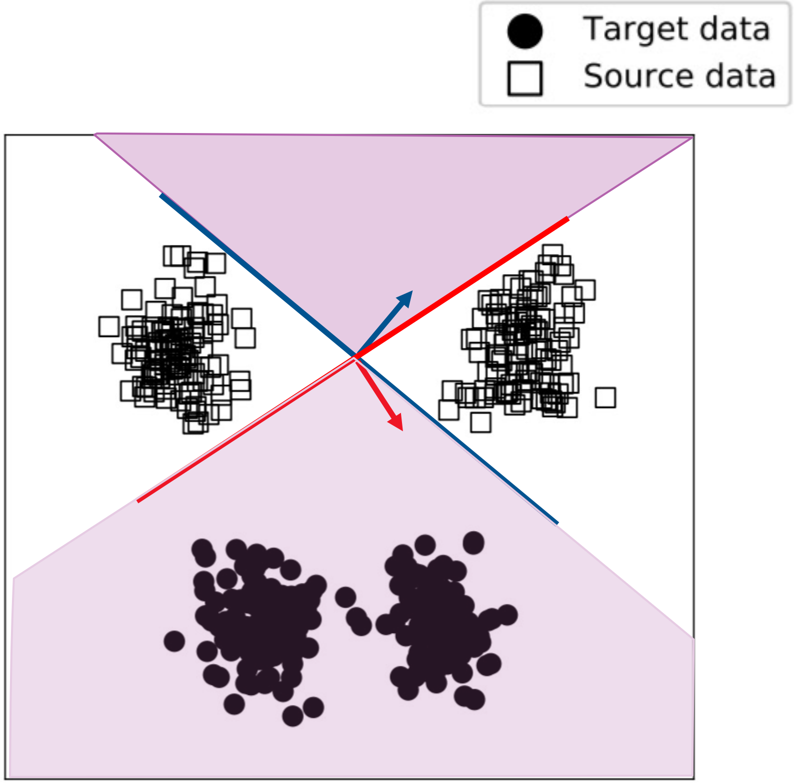
For a hypothesis class $\mathcal{H}$, the HΔH-divergence is

$$d_{H\Delta H}(p_{train}, p_{test}) = 2 \sup_{g \in H\Delta H} \left| E_{p_{train}}[g(x)] - E_{p_{test}}[g(x)] \right|$$

Interpretation: g $\in H\Delta H$ is the disagreement between a model h and h'

If a model h and h' agree on the training set, do they have to agree on the test set?

Examples of HΔH



Key example for intuition

h : blue classifier
h': red classifier

On p_{train}	**On p_{test}**
Perfect agreement	0% agreement
$E_{p_{train}}[g(x)] = 0$	$E_{p_{test}}[g(x)] = 1$

$\frac{1}{2} d_{H\Delta H}(p_{train}, p_{test}) = 1$ (**Vacuous!**)

Redko 2020

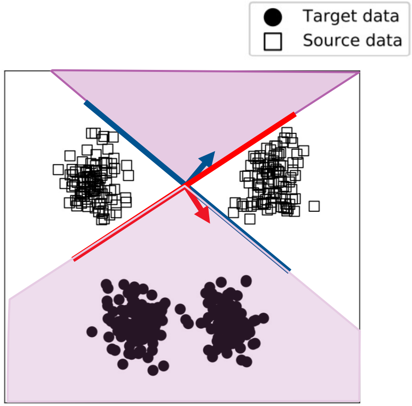
Some more intuition

What is $d_{H\Delta H}$ measuring?

Two classifiers that agree on training set

↓

Do they agree on the test set?



Why is this a useful notion of domain similarity?

Compare with IPMs:

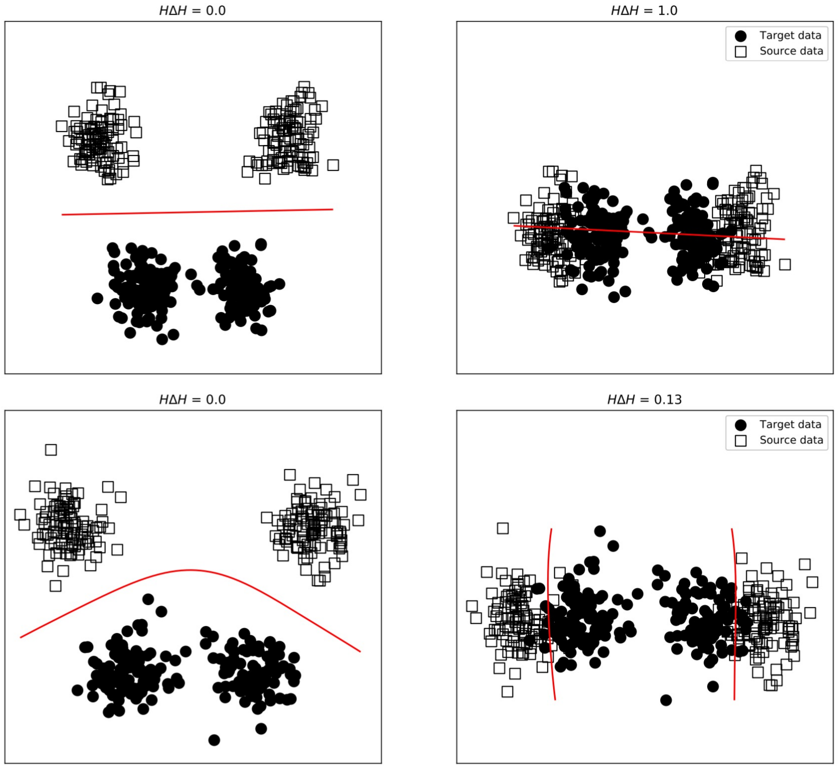
L_1: Disagreement over all possible classifiers

$d_{H\Delta H}$: Disagreement over IPMs

More examples of HΔH

Linear case

Nonlinear



Redko 2020

HΔH bounds error differences

We can now use $d_{H\Delta H}$ to bound our error. For any hypotheses h and h', define the

Disagreement:

$$\epsilon_p(h, h') = E_p[|h(x) - h'(x)|]$$

Disagreement gap:

$$\left|\epsilon_{p_{train}}(h, h') - \epsilon_{p_{test}}(h, h')\right| \leq \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})$$

Interpretation: if h and h' perform similarly on the training set, they will also perform similarly on the test set (up to an error of $d_{H\Delta H}(p_{train}, p_{test})$)

Basic H-delta-H domain adaptation bound

This (bounding the error gap) is enough to get full error bounds:

HΔH-generalization (Adapted from [Ben-David]):

For a classifier h $\in \mathcal{H}$ and any covariate shift

$$E_{p_{test}}[\ell(x,y,h)]$$
$$\leq E_{p_{train}}[\ell(x,y,h)] + \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test}) + \lambda$$

where $\lambda = \inf_{h \in \mathcal{H}} p_{train}\big(y \neq h(x)\big) + p_{test}(y \neq h(x))$

Techinque: apply error difference bound on the optimal classifier h^*

Let's walk through the proof

$$E_{p_{test}}[\ell(x,y,h)] \leq E_{p_{test}}[\ell(x,y,h^*)] + \underbrace{\epsilon_{p_{test}}(h,h^*)}_{= E_{p_{test}}[|h(x)-h^*(x)|]}$$

Error relative to $\boldsymbol{h^*}$

$$= E_{p_{test}}[|h(x) - h^*(x)|]$$

$\boldsymbol{d_{H\Delta H}}$ step

$$\leq E_{p_{test}}[\ell(x,y,h^*)] + \epsilon_{p_{train}}(h,h^*) + \left|\epsilon_{p_{test}}(h,h^*) - \epsilon_{p_{train}}(h,h^*)\right|$$

$$\leq E_{p_{test}}[\ell(x,y,h^*)] + \epsilon_{p_{train}}(h,h^*) + \underbrace{\frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})}$$

$$\left|\epsilon_{p_{train}}(h,h') - \epsilon_{p_{test}}(h,h')\right| \leq \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})$$

Triangle inequality

$$\leq E_{p_{test}}[\ell(x,y,h^*)] + \underbrace{E_{p_{train}}[\ell(x,y,h^*)] + E_{p_{train}}[\ell(x,y,h\ )]} + \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})$$

$$E_{p_{train}}[|h(x) - h^*(x)|] \leq E_{p_{train}}[|h^*(x) - y|] + E_{p_{train}}[|h(x) - y|]$$

What are these terms?

Let's walk through the main bound.

$$E_{p_{test}}[\ell(x,y,h)]$$
$$\leq E_{p_{train}}[\ell(x,y,h)] + \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test}) + \lambda$$

Training domain error

Domain distinguishability

Minimal error of a classifier on both domains

$$\lambda = \inf_{h \in \mathcal{H}} p_{train}\big(y \neq h(x)\big) + p_{test}\big(y \neq h(x)\big)$$

HΔH claim: Low training domain error + low $H\Delta H$ divergence + rich $\mathcal{H}$
= good generalization to target domain

Recap: $H\Delta H$ and divergences

What we covered:

- IPMs and divergences: a way to characterize errors under shifts
- L_1 bounds: widely applicable but wildly pessimistic
- $H\Delta H$-Divergence: classifier specific notion of domain similarity

Up next:

- Build intuition by studying each of the terms
- Alternatives to domain-divergence methods

What's $H\Delta H$ good for?

There's several things we might want to try..

Understand tradeoffs:

When we vary the hypothesis class and inputs what happens to each term?

Directly optimize the bound:

Can we perform model/feature selection to find models that generalize?

Quantify / estimate model performance:

Can we estimate the performance of models by estimating HΔH?

Spoiler: not all of these are possible

Accuracy-distinguishability tradeoffs

What are the components?

Training error: $E_{p_{train}}[\ell(x, y, h)]$

HΔH: $\frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})$

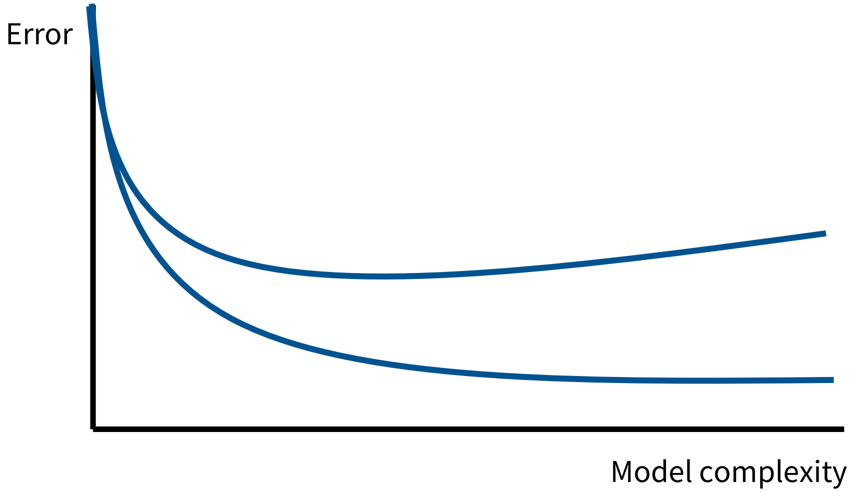
Optimal error: $\inf_{h \in \mathcal{H}} p_{train}\big(y \neq h(x)\big) + p_{test}(y \neq h(x))$

What can we vary?

Model complexity $(\mathcal{H})$

Varying training domain accuracy

Training domain error: $E_{p_{train}}[\ell(x, y, h)]$

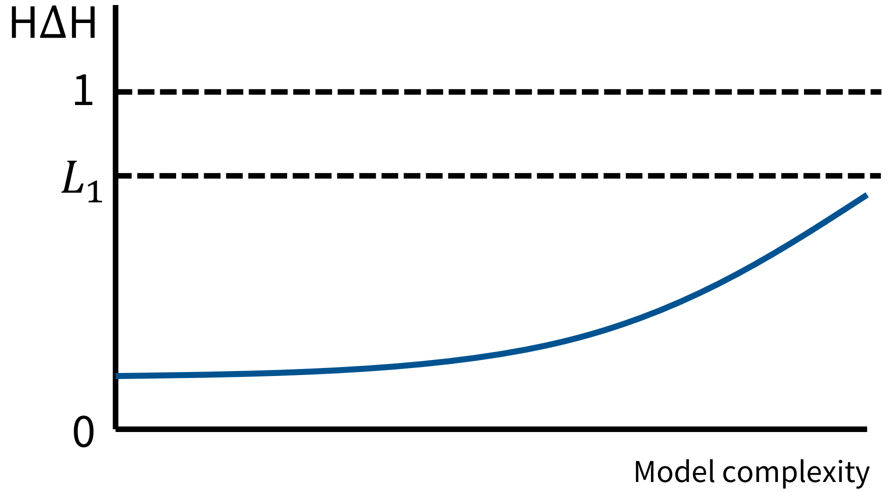


(finite train data)

(assuming infinite training domain data)

Varying HΔH

HΔH: $\frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})$



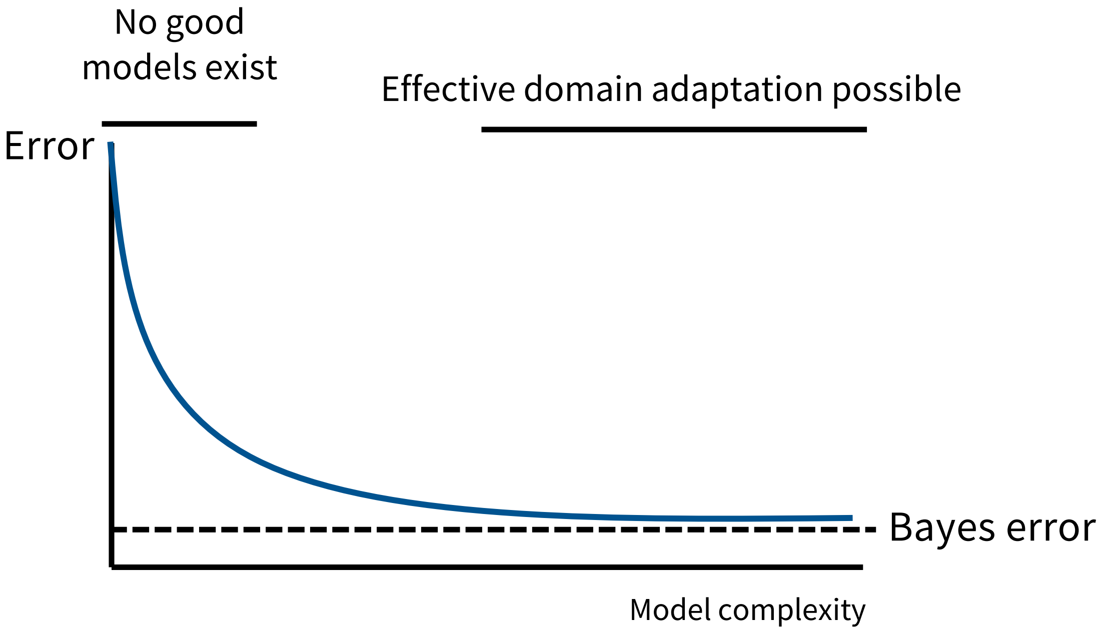
$d_{H\Delta H}$ is upper bounded by the L_1 distance

$d_{H\Delta H}$ increases monotonically with model complexity. If $H \subset H'$,

$$d_{H\Delta H} \leq d_{H'\Delta H'}$$

Controlling λ

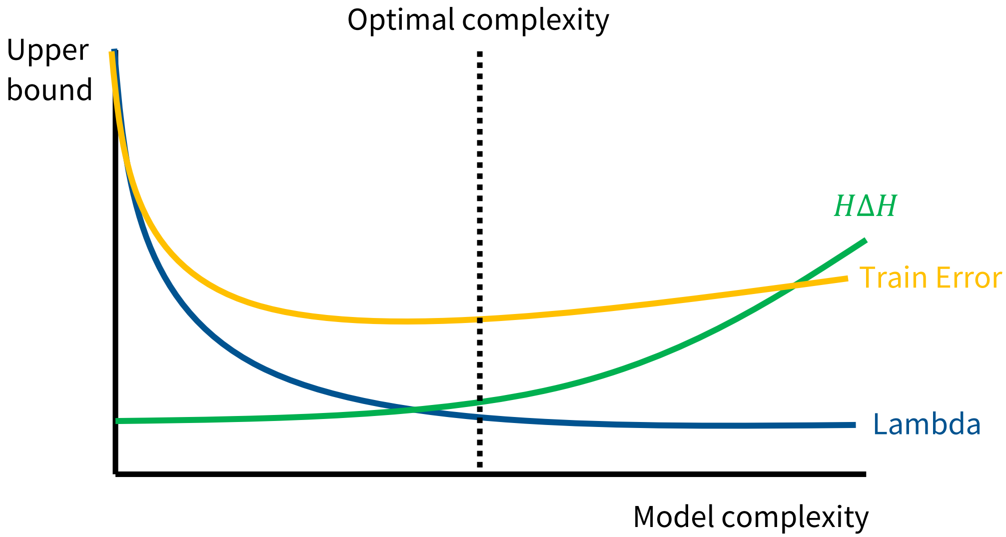
Optimal error: $\inf_{h \in \mathcal{H}} p_{train}\big(y \neq h(x)\big) + p_{test}(y \neq h(x))$



λ is a goodness of fit measure
(behave similarly to train error)

Typical assumption:
λ is small, on the order of train error

Can we use HΔH to build better models?



To find the optimal tradeoff,

We must balance **decreasing terms:**

Train Error and λ

Against the **increasing term**

HΔH

How can we measure $H\Delta H$ and λ?

Estimating HΔH

Key question: can we estimate HΔH from samples?

$$d_{H\Delta H}(p_{train}, p_{test}) = 2 \sup_{g \in H\Delta H} \left| E_{p_{train}}[g(x)] - E_{p_{test}}[g(x)] \right|$$

A natural finite sample estimator

$$\widehat{d_{H\Delta H}}(p_{train}, p_{test}) = 2 \sup_{g \in H\Delta H} \left| E_{p_{n,train}}[g(x)] - E_{p_{n,test}}[g(x)] \right|$$

True positive rate

False positive rate

Estimated by max-accuracy classifier

$$= \mathrm{TPR} - \mathrm{FPR} = 2\mathrm{Accuracy} - 1$$

Procedure for estimating $d_{H\Delta H}$

Estimating this in practice? Similar to GAN type setups

1. Set up a mixed dataset with 50-50 split from p_{train} and p_{test}

2. Train a classifier (**with hypothesis class $\boldsymbol{H\Delta H}$!**) to minimize error

3. Held-out error serves as a high-probability upper bound on $d_{H\Delta H}$.

(Or use VC-dimension bounds)

Background: VC dimension bound

What we have: reduction of HΔH and an estimator.

How good is our estimator with m samples:

VC-Dimension bound

Let $h \in \mathcal{H}$ be a classifier in a hypothesis space with VC dimension VC($\mathcal{H}$). Then for $\delta \in (0,1)$ with probability at least $1-\delta$ in the m-sample draw $(x,y) \sim p$ and uniformly in h,

$$p\big(y \neq h(x)\big) \leq \frac{1}{m}\sum_{i=1}^{m} 1_{h(x_i)=y_i} + \sqrt{\frac{4}{m}\Big(VC(\mathcal{H})\log\frac{2em}{VC(\mathcal{H})} + \log\frac{4}{\delta}\Big)}$$

Main HΔH result

Putting it all together: use VC dimension bound to make HΔH observable.

HΔH bound (Theorem 2 in Blitzer)

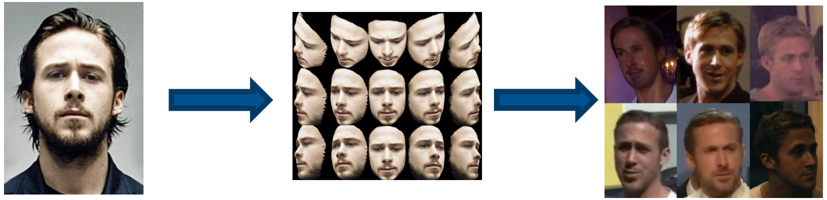
Let $h \in \mathcal{H}$ be a classifier in a hypothesis space with VC dimension VC($\mathcal{H}$). Then for $\delta \in (0,1)$ with probability at least $1-\delta$ in the draw m-sample draw $(x,y) \sim p_{train}$ and $x \sim p_{test}$ and uniformly in h,

$$E_{p_{test}}[\ell(x,y,h)] \leq E_{p_{train}}[\ell(x,y,h)] + \frac{1}{2}\hat{d}_{H\Delta H}(p_{train}, p_{test}) + \lambda + c$$

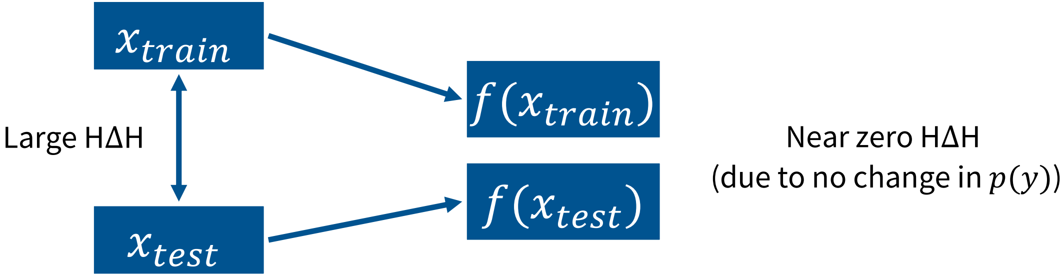
$$c = 4\sqrt{\frac{2\,VC(\mathcal{H})\log(2m) + \log\frac{2}{\delta}}{m}}$$

Onward model selection: collapsing the input features

Our idea: use our HΔH to transform our inputs to have a smaller $d_{H\Delta H}$.



Now think about: domain adaptation + covariate shift, where $p_{train}(y) = p_{test}(y)$.



Why does this fail?

'Typical' assumption:

$\lambda \ll \text{Train error+H}\Delta\text{H}$ (otherwise domain adaptation is impossible)

What about our case?:

$$\lambda = \inf_{h\in\mathcal{H}} p_{train}\big(y \neq h(x)\big) + p_{test}(y \neq h(x))$$

Test domain error: $\inf_{h\in\mathcal{H}} p_{test}(y \neq h(f(x)))$

Potentially *very* large if $f(x)$ is not good on p_{test}

Ignoring λ leads to bounds that are far too optimistic

How can we avoid the problems from lambda?

Reference	Learning bounds				
	Task	Framework	Divergence	Link	Non-estim.
[Ben-David et al., 2007] [Blitzer et al., 2008] [Ben-David et al., 2010a]	Binary classification	VC	L^1, $\mathcal{H}\Delta\mathcal{H}$	Add.	+
[Mansour et al., 2009a]	Classification/ Regression	Rademacher	Discrepancy	Add.	+
[Kuroki et al., 2019]	Classification	Rademacher	(S-)Discrepancy	Add.	+
[Cortes et al., 2010] [Cortes and Mohri, 2014] [Cortes et al., 2015]	Regression	Rademacher	(Generalized) Discrepancy	Add.	+
[Mansour et al., 2008]	Classification/ Regression	–	–	–	–
[Mansour et al., 2009b] [Hoffman et al., 2018]	Classification/ Regression	–	Rényi	Mult.	–
[Dhouib and Redko, 2018]	Binary classification/ Similarity learning	–	L^1, χ^2	Mult.	+
[Redko et al., 2019a]	Binary classification	Rademacher	Discrepancy	Add.	+
[Zhang et al., 2012]	Regression/ Classification	Uniform entropy number	IPM	Add.	–
[Redko, 2015]	Regression	Rademacher	IPM/MMD	Add.	+
[Redko et al., 2017]	Regression	–	IPM/Wassertein	Add.	+
[Zhang et al., 2019]	Large-margin classification	Rademacher	IPM	Add.	+
[Dhouib et al., 2020b]	Large margin Binary classification	–	IPM/minimax Wasserstein	Add.	+
[Johansson et al., 2019]	Classification	–	IPM	Add.	+
[Shen et al., 2018]	Classification	–	Wasserstein	Add.	+
[Courty et al., 2017]	Classification	–	Wasserstein	Add.	+
[Germain et al., 2013]	Classification	PAC-Bayes	Domain disagreement	Add.	+
[Germain et al., 2016]	Classification	PAC-Bayes	β-divergence	Mult.	+

Redko+ 2020

- Consider different settings (ensembling classifiers) [Mansour+ 2008,2009]
- Consider small amounts of target domain data
- Consider broad function classes [Zhang+ 2013]
- Assume overlap [Mansour and Schain 2014]
- Make stronger assumptions on the distribution shift (up next)

Recap: UDA with H-delta-H

- **Divergence based bounds.** For any bounded loss,

$$E_{p_{test}}[\ell(x,y,\theta)] \leq E_{p_{train}}[\ell(x,y,\theta)] + \left|\left|p_{train}(x) - p_{test}(x)\right|\right|_1$$

- **HΔH** (basic bound).

$$E_{p_{test}}[\ell(x,y,h)\,] \leq E_{p_{train}}[\ell(x,y,h)] + \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test}) + \lambda$$

- Understanding the 3 different terms.
  - **Source error:** measurable on data, monotone decreasing with complexity
  - **HΔH:** measurable via VC-bound, monotone increasing
  - **λ:** ***not*** measurable, monotone decreasing with complexity.

Ideas beyond domain distances and reweighting

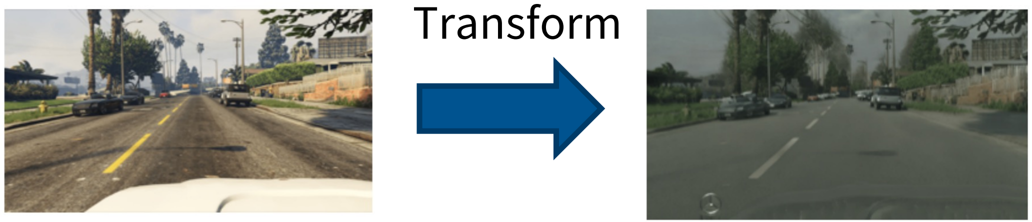
This is a bad situation for HΔH and reweighting

- Zero overlap (different styles)
- Distinguishable by almost any modern CNN

Are things hopeless?

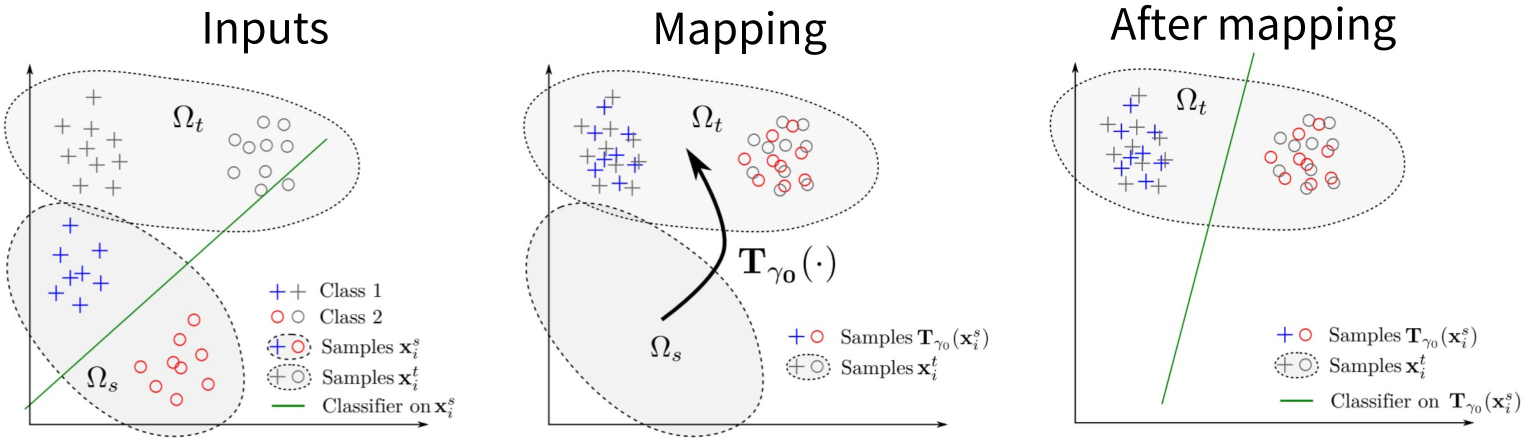
Feasibility of domain adaptation in this setting

Motivation: It should be possible to do well here
(so the bounds are incomplete)

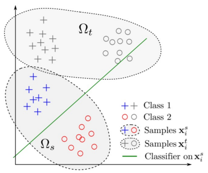


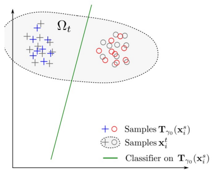
If there exist simple transformations
between domains, can we say anything?

Translation: a naïve proposal



Interpretation via the HΔH bound



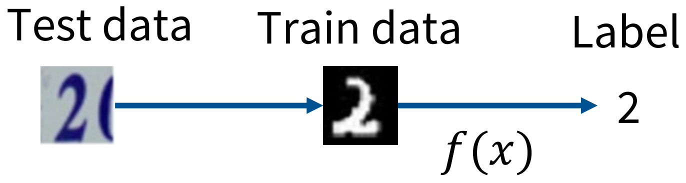


	Input	Translated
Source error	Small	Small
$d_{H\Delta H}$	Huge ($\approx$ 1)	Near-zero
λ	Small	Possibly large

Note: λ is small if translation ‘preserves’ the conditional

Preserving the conditional

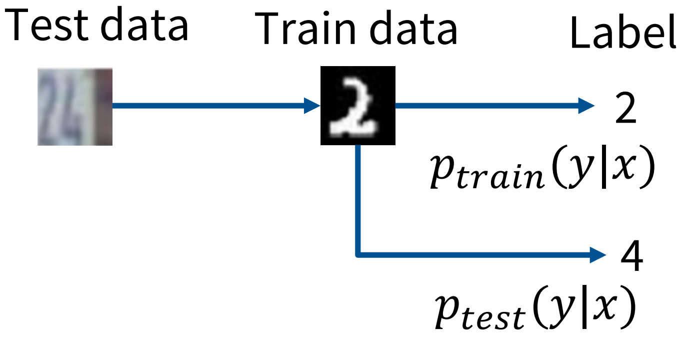
Conditional-preserving translation:



Source error ≈ Test error

$\lambda \approx 2$ Source error

Other translation



No single model works well

Test error ≫ Source error

Can we get guarantees for transformations?

When can we guarantee conditional preservation?

Only under strong conditions..

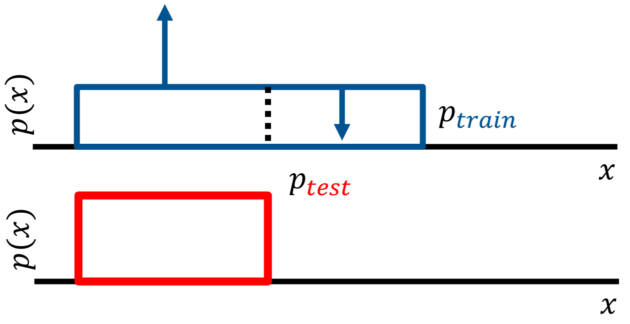
- Known family of transformations (e.g. PSD linear transforms)
- Explicit pairings / supervision given about the transformation

In practice..

Style transfer models and such are used.
Implicitly relies upon inductive bias of CNNs

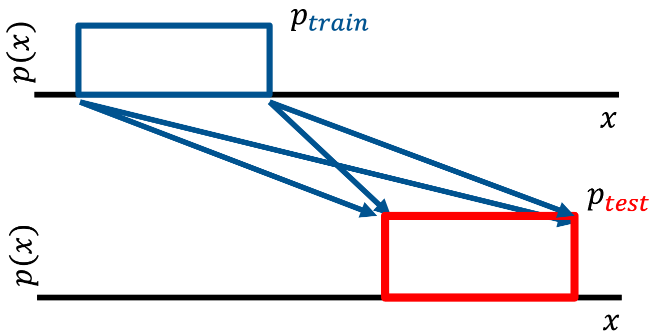
Transformations and relations to reweighting

Reweighting
Change the frequency of points



Transformation/Translation
Change the location of points

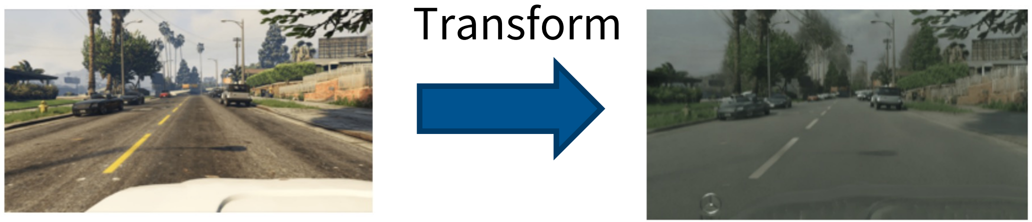
Not uniquely identified



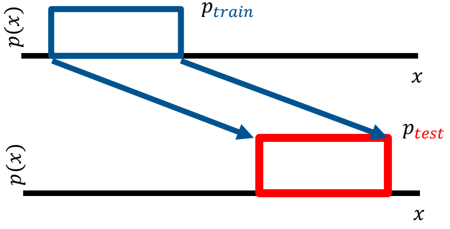
Reweighting/Transformation: bias correction strategies.
IPM/HΔH : measuring errors due to remaining bias

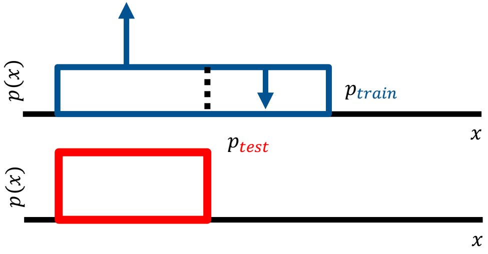
Recap: transformation based methods

- Many domains admit simple transformations



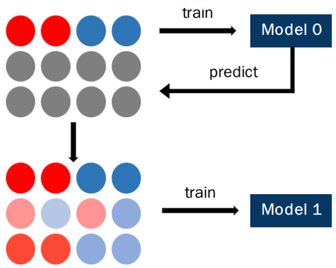
- A complementary bias-mitigation strategy to reweighting





Other paradigms for this setting

- **Self-training** [Habrard+ 2013], more modern work by [Wei+ 2020]



- **Ensembling** [Mansour 2008+] and others

For more: “A survey on domain adaptation theory” by Redko+ 2020
Or “A survey of unsupervised deep domain adaptation” by Wilson+ 2020

Theory to practice – where does this stuff appear?

HΔH

DANN
(domain adversarial neural nets)

Translation

Unsupervised translation methods
(e.g. PixelDA, CycleGAN)

Adversarial + translation methods
(e.g., CyCADA)

More recently – combination of these methods + self-training

Overall summary

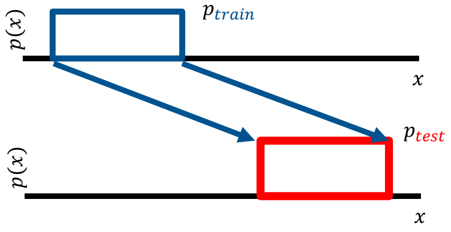
- **Divergence, IPMs, and HΔH**

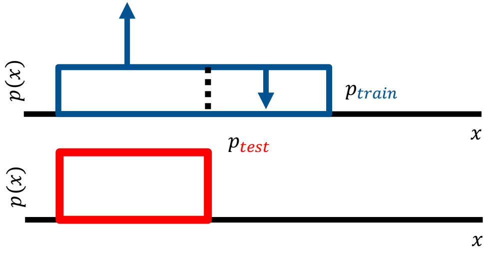
$$E_{p_{test}}[\ell(x,y,\theta)] \leq E_{p_{train}}[\ell(x,y,\theta)] + \left|\left|p_{train}(x) - p_{test}(x)\right|\right|_1$$

- **Limitations of HΔH**

$$E_{p_{test}}[\ell(x,y,h)] \leq E_{p_{train}}[\ell(x,y,h)] + \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test}) + \lambda$$

- **Translation as an alternative to reweighting**





Reminders:

- **Next discussion**

  Domain adaptation and translation papers

- **Assignments**

  Remember to prepare/submit summaries!

- **Project**

  Project outline due Oct 25

Lecture 4

Neural methods for domain adaptation (I)

CS329D

Goals for today

Understand the basic DANN architecture + variations

Connect unsupervised mappings (CycleGAN) to domain adaptation

Know the conceptual / theory foundations of these methods

Domain adaptation in the wild (images)

Previous lectures:

Theory of unsupervised domain adaptation

This lecture:

Building systems that work on ‘real’ problems

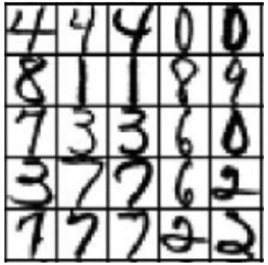
Notable features:

- No overlap
- High dimensions

Training data | Test domains

MNIST

USPS



SVHN

Training data (GTA)

Test data (real world)

Solving these more sophisticated problems

The big difference in this lecture:

Bounded VC dim (theory land) $\longrightarrow$ Neural methods

We *have* to use neural methods to get decent performance

New challenges:

Distinguishability / $H\Delta H$: Almost always vacuous

Translations / mappings : Underdetermined

Outline

We will cover two major families of methods

Domain invariance

Based on the Ben-David $H\Delta H$ style distinguishability bounds

Domain mapping

Based on the "Optimal Transport for Domain Adaptation" style direct mapping

There are other approaches (self-training, self-supervision) that we cover next lecture

Starting point: $H\Delta H$ style bounds.

HΔH-generalization (Adapted from [Ben-David]):

For a classifier h ∈ $\mathcal{H}$ and any covariate shift

$$E_{p_{test}}[\ell(x,y,h)]$$
$$\leq E_{p_{train}}[\ell(x,y,h)] + \frac{1}{2} d_{H\Delta H}(p_{train}, p_{test}) + \lambda$$

where $\lambda = \inf_{h\in\mathcal{H}} p_{train}\big(y \neq h(x)\big) + p_{test}(y \neq h(x))$

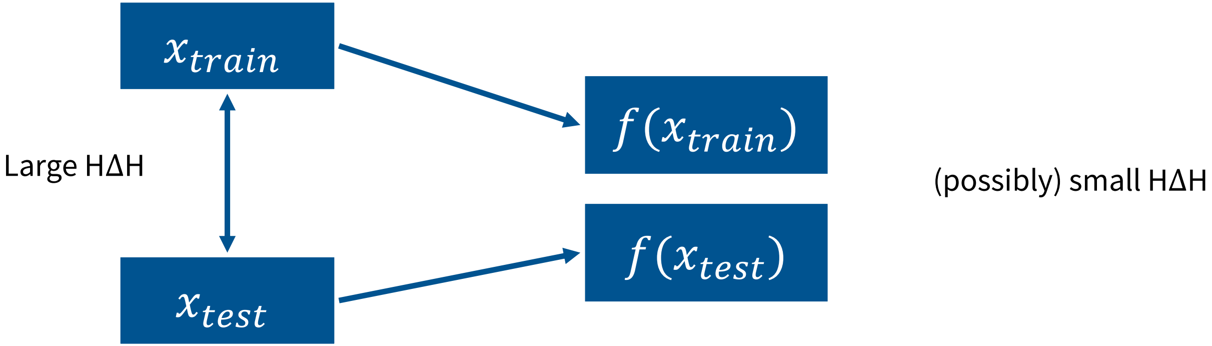
The dilemma:

Neural nets are needed to optimize $E_{p_{train}}[\ell(x,y,h)]$ and λ

Neural nets have vacuous $\frac{1}{2} d_{H\Delta H}(p_{train}, p_{test})$

Key idea: measure invariance on representations

Consider invariance of *representations*

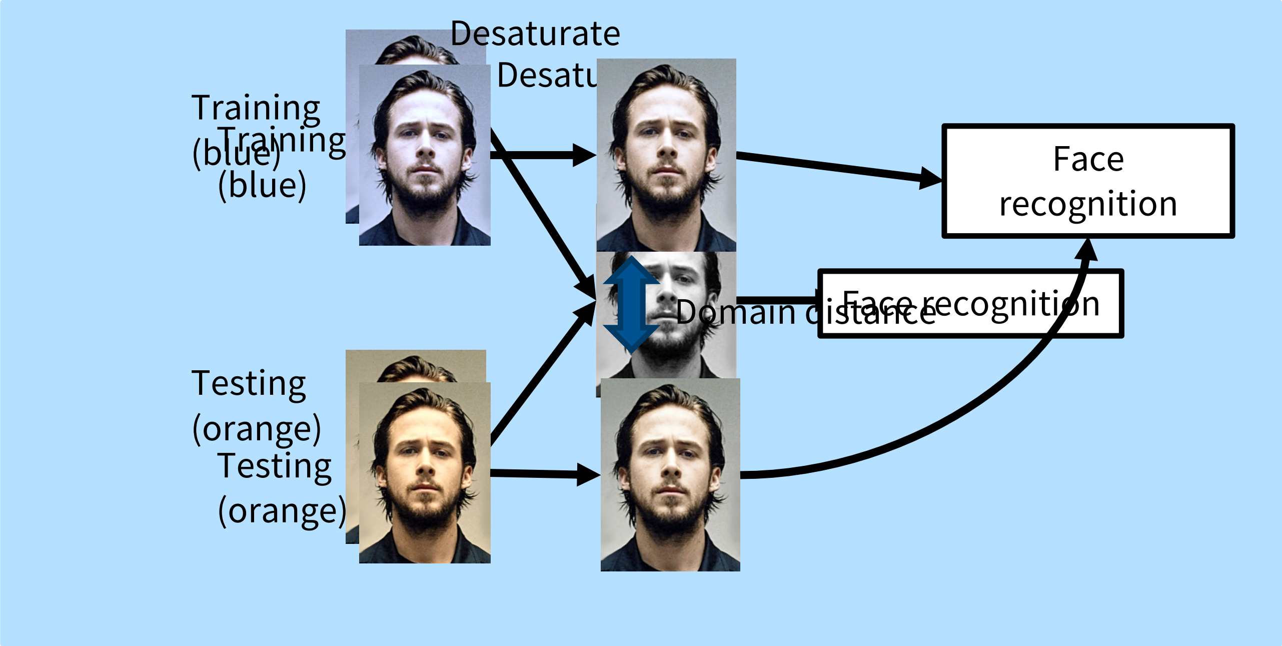


Key point:

Neural networks (usually) have a linear layer at the end.
$H\Delta H$ can likely be made small on this last layer.

Domain invariance

Intuition from lecture 1:



Guiding principle: Encourage the model to learn ‘invariant’ representations

How can we optimize each term?

New setup: learn a hypothesis h and feature map T to minimize

Recall the bound:

Notation: $T\#p$ is the pushforward of p

$$E_{p_{test}}[\ell(T(x), y, h)] \leq E_{p_{train}}[\ell(T(x), y, h)] + \frac{1}{2} d_{H\Delta H}(T\#p_{train}, T\#p_{test}) + \lambda_T$$

How do we optimize?

Training domain error	Domain distinguishability	Min joint error
Optimize normally	???	Ignore it (pray it's small)

$$\lambda = \inf_{h \in \mathcal{H}} p_{train}(y \neq h(x)) + p_{test}(y \neq h(x))$$

Minimizing $H\Delta H$

Recall that $H\Delta H$ can be estimated as accuracy

$$\widehat{d_{H\Delta H}}(T\#p_{train}, T\#p_{test}) = 2 \sup_{g \in H\Delta H} \left| E_{p_{n,train}}[g(T(x))] - E_{p_{n,test}}[g(T(x))] \right|$$

Domain invariance objective (general form)

$$\inf_{T,h} \sup_{g \in H\Delta H} E_{p_{n,train}}[\ell(T(x), y, h)] + \left| E_{p_{n,train}}[g(T(x))] - E_{p_{n,test}}[g(T(x))] \right| + \lambda_T$$

Training domain error

Accuracy of distinguishing domains

Ignore

This is a minimax optimization problem (generally bad news)

Background: adversarial neural methods

Domain invariance takes the form of an adversarial game

$$\inf_{T,h} \sup_{g \in H\Delta H} E_{p_{n,train}}[\ell(T(x), y, h)] + \left| E_{p_{n,train}}[g(T(x))] - E_{p_{n,test}}[g(T(x))] \right|$$

T, h: ‘min’ player (goes first)

Tries to find low-train loss representations T that have similar values g on train (source) and test (target)

g: ‘max’ player (goes second)

Tries to find a classifier in $H\Delta H$ that has high accuracy identifying the domain of $T(x)$

These games are hard to solve: think of what happens when g is suboptimal

Background: simultaneous gradient descent

Provably solving two player games: hard

Current approach: useful heuristics that are hard-to-tune but can work.

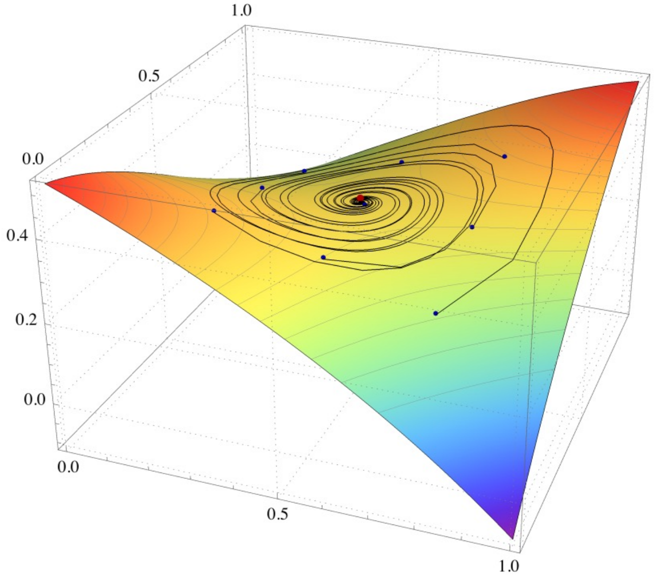
Simultaneous gradient descent/ascent

Pseudocode: solving $\min_{\theta} \max_{\phi} f(\theta, \phi)$

At time t

1. $\theta^t \leftarrow \theta^{t-1} - \alpha \nabla_\theta f(\theta, \phi^{t-1})$
2. $\phi^t \leftarrow \phi^{t-1} + \alpha \nabla_\phi f(\theta^{t-1}, \phi)$

Players have different gradient signs



Inspired by no-regret strategies for two player games

Next steps: how do we operationalize the bound?

$$\inf_{T,h} \sup_{g \in H\Delta H} E_{p_{n,train}}[\ell(T(x), y, h)] + \left|E_{p_{n,train}}[g(T(x))] - E_{p_{n,test}}[g(T(x))]\right| + \lambda_T$$

Two (related) decisions:

What surrogate do we use to accuracy?: accuracy is not differentiable

What is $\boldsymbol{H\Delta H}$: We don't have a easy way to write down or optimize this set

Answer: we will replace this term with a convenient IPM

Main decision: choice of distinguishability

We will cover the major variations of domain-invariant neural nets

Key decision: how do we measure distinguishability (g)?

1. Classification error of a neural net ($H\Delta H$ / L_1)

2. Discrepancy under g selected from a RKHS (MMD)

3. Discrepancy under all Lipschitz continuous g (Wasserstein)

Choice #1: adversarial classification [Ganin 2015]

What if we use a neural network for g?

Domain predictor:

$$f\big(T(x)\big) = sigmoid(w^\top T(x) + b)$$

Domain indicator:

$z = 1$ if a sample is from the source, 0 otherwise

Surrogate loss:

$$\mathcal{L}\big(f\big(T(x)\big), z\big) = z \log \frac{1}{f(T(x))} + (1-z) \log \frac{1}{1 - f(T(x))}$$

Overall domain penalty:

$$-\frac{1}{2}\left(E_{p_{train}}\left[\mathcal{L}\big(f\big(T(x)\big), 1\big)\right] + E_{p_{test}}\left[\mathcal{L}\big(f\big(T(x)\big), 0\big]\right)\right.$$

Hand-wavy claim: ‘this is like the error of classifier f(T(x))’

Full DANN objective (in sample form)

Going from the invariance objective to DANN

$$\inf_{T,h} \sup_{g \in H\Delta H} E_{p_{n,train}}[\ell(T(x), y, h)] + \left| E_{p_{n,train}}[g(T(x))] - E_{p_{n,test}}[g(T(x))] \right|$$

$\updownarrow$ Accuracy = 1-Error

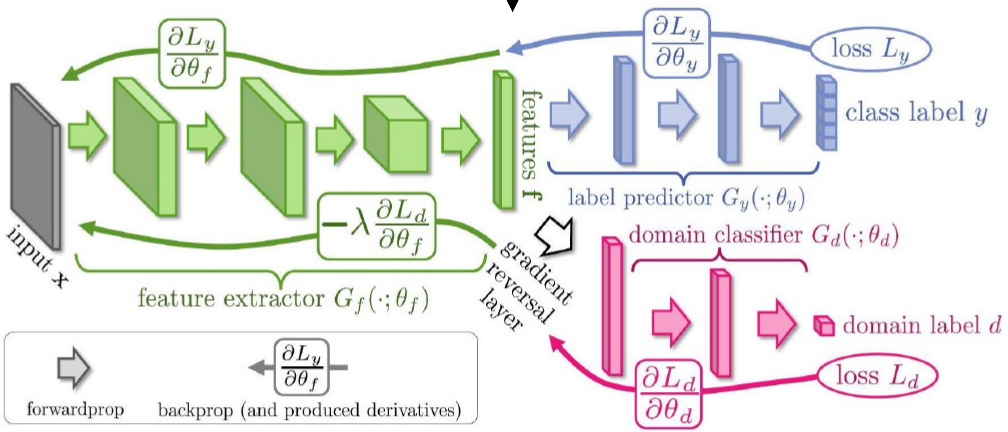
$$-\frac{1}{2}\left(E_{p_{train}}\left[\mathcal{L}\left(f\left(T(x)\right), 1\right)\right] + E_{p_{test}}\left[\mathcal{L}\left(f\left(T(x)\right), 0\right]\right)\right.$$

Full DANN objective

$$E_{p_{n,train}}[\ell(T(x), y, h)] - \beta \left(E_{p_{n,train}}\left[\mathcal{L}\left(f\left(T(x)\right), 1\right)\right] + E_{p_{n,test}}\left[\mathcal{L}\left(f\left(T(x)\right), 0\right]\right)\right.$$

The overall DANN architecture

Domain invariant, useful features



Maximizes performance on training data

Minimizes distinguishability

This is simultaneous gradient descent on the DANN objective

$$\min_{T,h}\max_{f} E_{p_{n,train}}[\ell(T(x),y,h)] - \beta\big(E_{p_{n,train}}[\mathcal{L}(f(T(x)),1)] + E_{p_{n,test}}[\mathcal{L}(f(T(x)),0]\big)$$

The gains from DANN

Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST	MNIST->SVHN
Labeled target (oracle)	96.5	99.2	99.5	
DANN	85.1		73.6	35.7
Subspace align			59.3	
Source only	78.9	69.6	59.2	

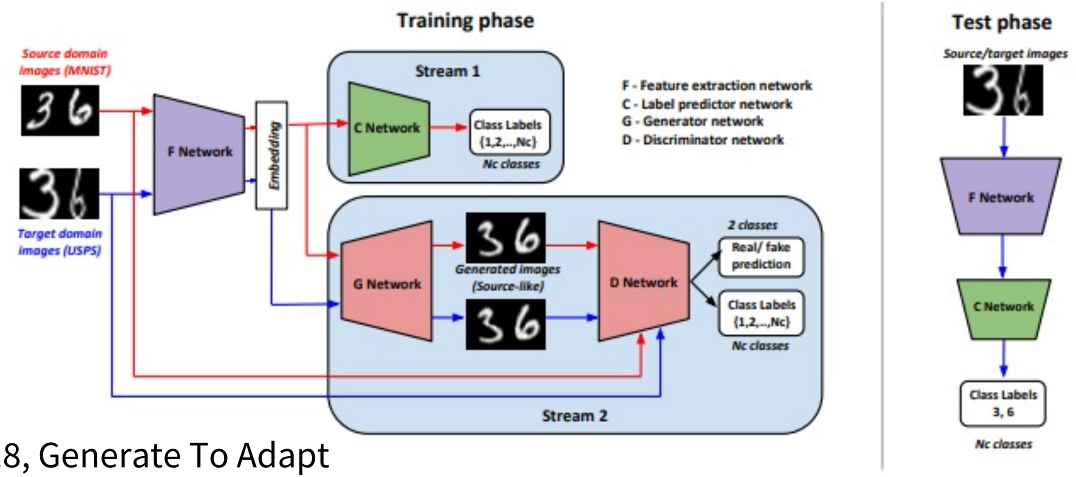
Variations: constraining the classifier (reconstruction)

Major issue with DANN:

Unmeasured λ term can collapse

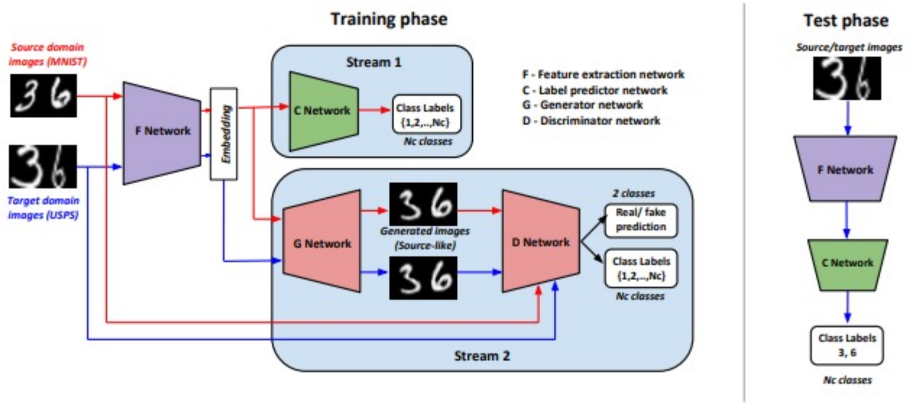
Mitigation idea: classify images, not representations

1. Reconstruct source-like images from the representation
2. Use a GAN to identify real vs fake source images.



Sankaranarayanan+ 2018, Generate To Adapt

Benefits and pitfalls of reconstruction methods



What does reconstruction do for us?

Prevents the collapse of latent representations (e.g. only the label)

What doesn't it do?

Preserve the label distribution – the generator may map a 3 to a 6

(This is still a useful heuristic for controlling λ)

The gains from DANN

SVHN

MNIST

Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST	MNIST->SVHN
Labeled target (oracle)	96.5	99.2	99.5	
GenToAdapt	92.8	90.8	92.4	
DANN	85.1		73.6	35.7
Source only	78.9	69.6	59.2	

Choice #2: Maximum mean discrepancy (MMD)

Practical issue:

All adversarial neural methods are *horrible* to optimize

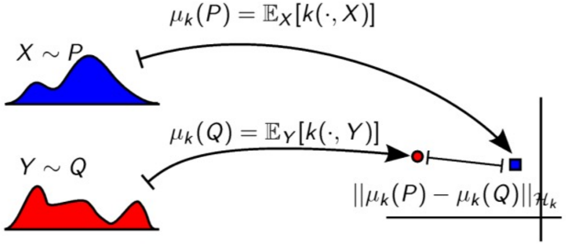
Solution (?):

Pick a family of g that does not require adversarial optimization

Kernels embeddings!

If we're willing to define g as coming from a RKHS,

$$\max_{|g|_{\mathcal{H}} \leq 1} \left| E_p[g(x)] - E_q[g(x)] \right| = |\, \mu_k(p) - \mu_k(q)|_{\mathcal{H}}$$



Advantages of MMD – no adversarial optimization

$$\inf_{T,h} \sup_{g \in H\Delta H} E_{p_{n,train}}[\ell(T(x), y, h)] + \left| E_{p_{n,train}}[g(T(x))] - E_{p_{n,test}}[g(T(x))] \right|$$

If we pick $H\Delta H$ the unit norm ball in a RKHS..

$$\inf_{T,h} E_{p_{n,train}}[\ell(T(x), y, h)] + |\mu_k(T\#p_{train}) - \mu_k(T\#p_{test})|$$

Note: $T\#p$ is the pushforward of p under T

The price: performance in practice (Deep adaptation networks, Long+ 2016)

Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST	MNIST->SVHN
DANN	85.1		73.6	35.7
DAN (kernel)	81.1		71.1	

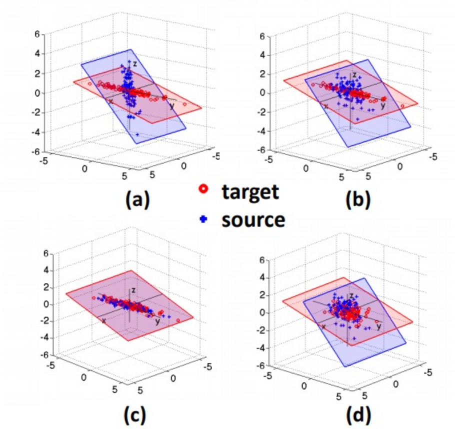
Special case: (deep) CORAL

An important example of this class of methods is CORAL (Sun+ 2016)

Deep CORAL algorithm:

Penalize the squared *difference of covariances*

$$\left|E_{p_{train}}[T(x)T(x)^\top] - E_{p_{test}}[T(x)T(x)^\top]\right|_F^2$$



This is a very lightweight domain adaptation algorithm

Interpretation as MMD:

Pick $\mathcal{H}$ generated by the quadratic kernel $k(x, y) = (x^\top y + 1)^2$

MMD for this $\mathcal{H}$ will ensure that two distributions have identical covariance.

How does CORAL / MMD / DANN do?

Office dataset comparison:

(well-tuned) DANN methods do well

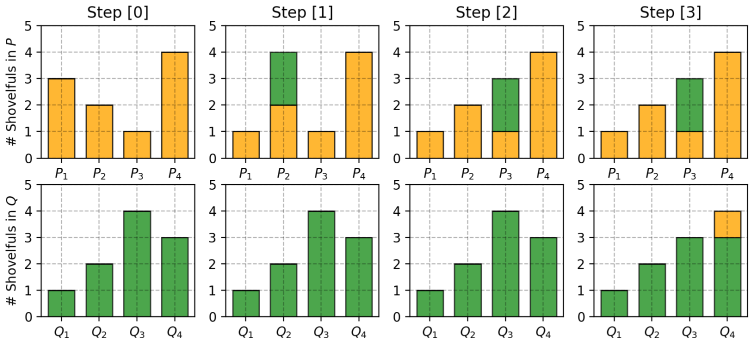
	Method	A->W	D->W	W->D
adversarial	Gen to Adapt	89.5	97.9	99.8
adversarial	DANN	67.3 – 73.0	94.0 – 96.4	93.7 – 99.2
MMD	DAN	68.5	96.0	99.0
MMD	Deep CORAL	66.4	95.7	99.2
	Source only	62.6	96.1	98.6

Choice #3: Optimal transport

Another choice with closed form maximization: Lipschitz continuous functions

$$\max_{g \in \mathcal{F}_L} \left| E_p[g(x)] - E_q[g(x)] \right| = W_1(p,q)$$

This is the 'optimal transport' we covered earlier in the course



Deep JDOT: Use optimal transport costs to get invariance

$$\inf_{T,h} E_{p_{n,train}}[\ell(T(x), y, h)] + W_2(T\#p_{train}, T\#p_{test})$$

$$d(x, x') = |x - x'|_2 + \beta\, |h_0(x) - h_0(x')|$$

How are the models that you get from these choices?

Range of model performance:
Careful engineering and newer models tend to win

Legend
:
Optimal transport methods
Adversarial classification methods
MMD methods

Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST
Labeled target (oracle)	96.5	99.2	99.5
Deep JDOT	95.7	96.4	96.7
Gen. to. Adapt	92.8	90.8	92.4
DeepCORAL	89.3	91.5	59.6
DANN	85.1 (95.7)	(90.0)	73.6
DAN	81.1 (88.5)	73.5	71.1
Source only	78.9	69.6	59.2

(from Damodaran+ 2018 and Wilson and Cook 2020)

Notes and pitfalls

1. Adversarial methods are hard to tune.

   Huge range in reported performance (85 to 95%)

2. Comparisons can depend on benchmark-specific trick and tuning

   Tricks like image intensity normalization changing performance from 37.5% → 97% on MNIST → SVHN

   French+ 2018

3. A range of engineering decisions we didn't cover here

   Separate weights for source vs target / stagewise training

See Wilson and Cook if interested.

High-level recap of domain invariance

Main conceptual distinction across methods: defining g

Adversarial classification: neural nets + adversarial training

MMD: functions from a kernel space + analytic maximization

Wasserstein: Lipschitz continuous functions + optimal transport

Effective method from each family, though **Adversarial** can be hard to tune, and **MMD** can underperform

Domain mapping: reminder

Motivation: Most domain adaptation datasets have *direct mappings* between source and target

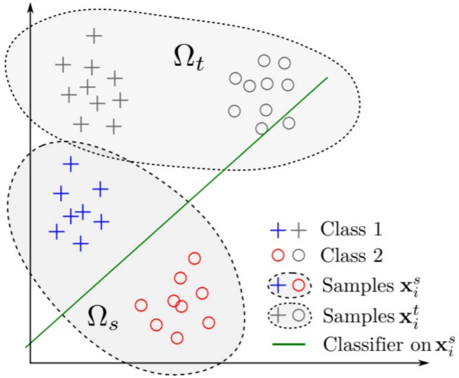
Transform

Can explicit domain mappings improve upon invariance?

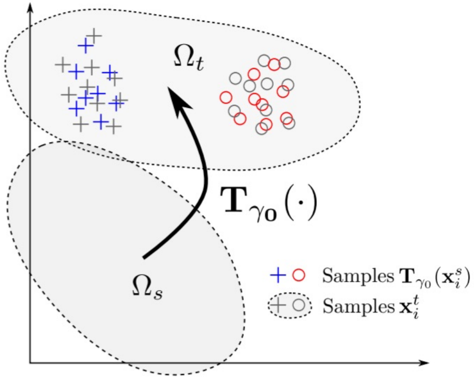
Optimal transport for domain adaptation

Reminder: OTDA from 2 lectures ago:

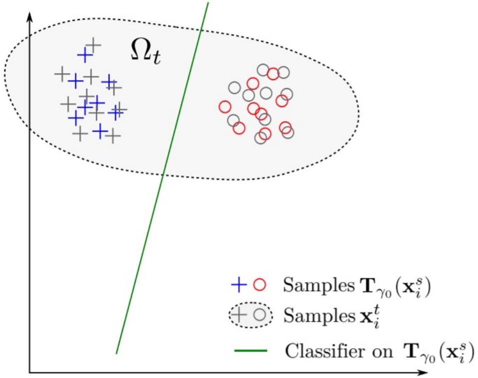
Inputs



Mapping



After mapping



“Deep” OTDA

→ NN features → OTDA

Method	A->W	D->W	W->D
DANN	67.3 – 73.0	94.0 – 96.4	93.7 – 99.2
OTDA + NN	84.5	94.1	91.3
OTDA	37.0	81.0	84.0
Source only	62.6	96.1	98.6

From generic to pixel-level mappings

OTDA tradeoffs:

	Mapping in deep feature space	Mapping in pixel space
Assumption	Good feature map (similar assumption as invariance)	Meaningful cost function (pixels distances are awful)

Can we learn useful pixel-level mappings?

Unsupervised domain mapping / translation

Unsupervised translation to the rescue!

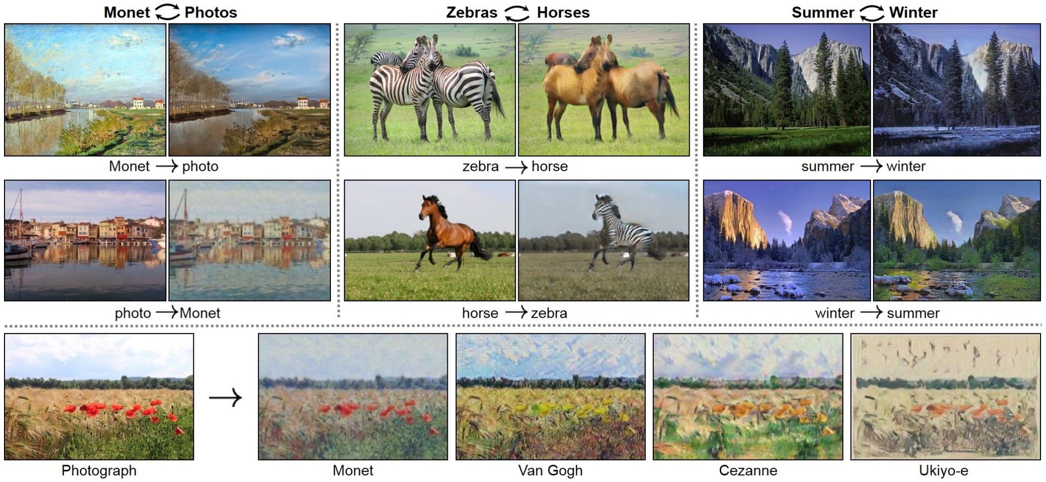
Task

Input: given unpaired images (or text) from two domains

Output: return a function that can map from one domain to another

Note: This is *exactly* the domain mapping problem

Right: Example from one such system (cycleGAN)



Background: cycle consistency losses

The OTDA idea:

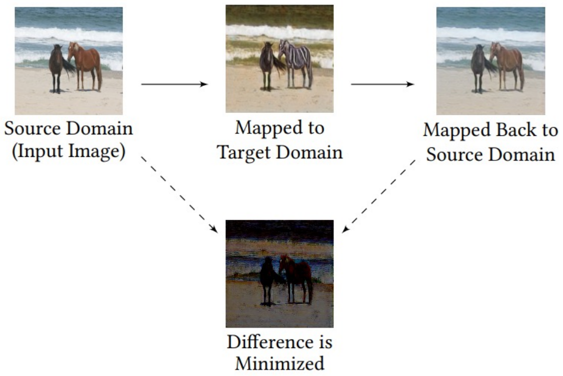
Map from source to target $(T\#p_{train})$ evaluate $d(T\#p_{train}, p_{test})$

Cycle-consistency:

Additional constraint on T: ensure $d(T^{-1} \circ T\#p_{ptrain}, p_{train})$

Intuition: Domain maps should be *invertible*

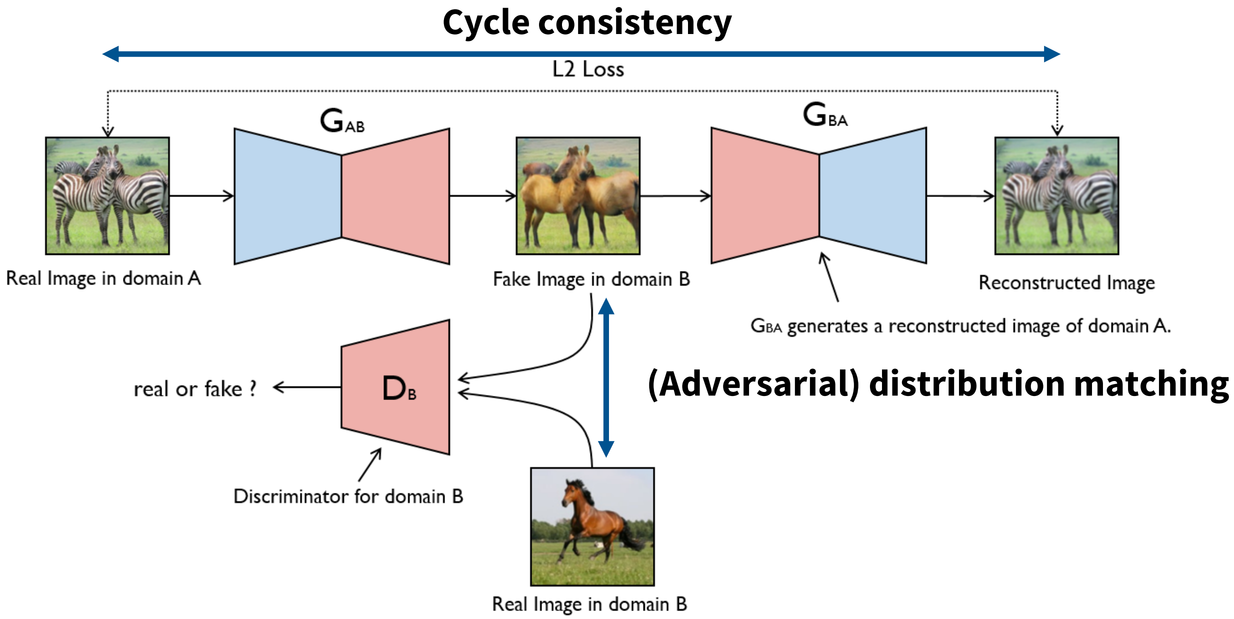
Pictoral version:



Putting it together - cycleGAN

The example from before (cycleGAN) is *exactly* this idea

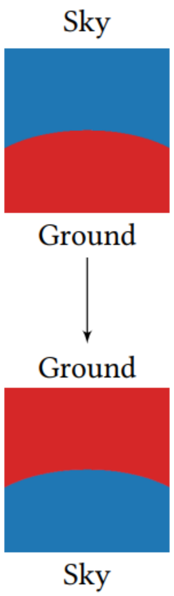
1. Similarity of mapped distribution on the target
2. Invertibility of the learned mapping



Background: semantic consistency losses

Pitfalls of domain mapping: We might flip labels around

Domain mapping does not care
about our domain adaptation task!



Idea: constrain domain mapping to preserve labels (hard)

Actual implementation: train a classifier f on source, prefer mappings

$$f(x) \approx T(f(x))$$

often with a penalty $d(f(X), T(f(X)))$

Very popular: in CyCADA, generate-to-adapt, etc.

Background: commutativity constraints

One last note: We can incorporate *even more* priors about the map

Observation:

rotate 180 → map T → rotate 180 $\xleftrightarrow{=}$ map T

rotate 180 → map T $\longleftrightarrow$ map T → rotate 180

Simple image transforms should *commute* with the map

Used in some fancier mapping algorithms (GcGAN)

Summary: unsupervised domain mappings

Ingredients:

- Mapping similarity (is $T\#p_{train} \approx p_{test}$?)
  - Adversarial losses
- Problem-specific constraints
  - Invertibility (Cycle consistency)
  - Commutativity (Geometry preservation)
- Conditional preservation (semantic consistency)

Not discussed: unsupervised machine translation

Domain map, then classify

How well does domain mapping alone work?

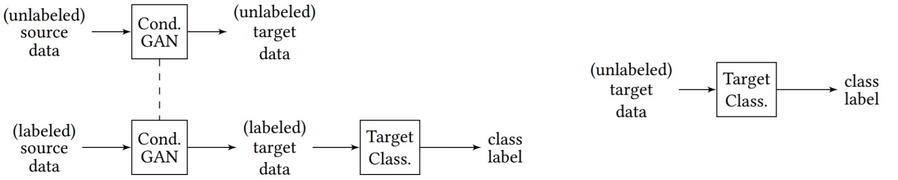
Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST
Labeled target (oracle)	96.5	99.2	99.5
CycleGAN*	95.6	96.4	70.3
DANN	85.1 (95.7)	(90.0)	73.6
Source only	78.9	69.6	59.2

(From GcGAN)

A: *very* good for similar domains, less so for different ones

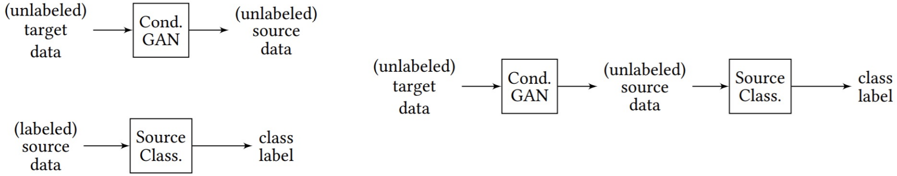
Variations to the map+classify approach

Other variations



(a) Method 1 (most common) – training (left), testing (right)

Most methods (sim gan, dcgan) etc



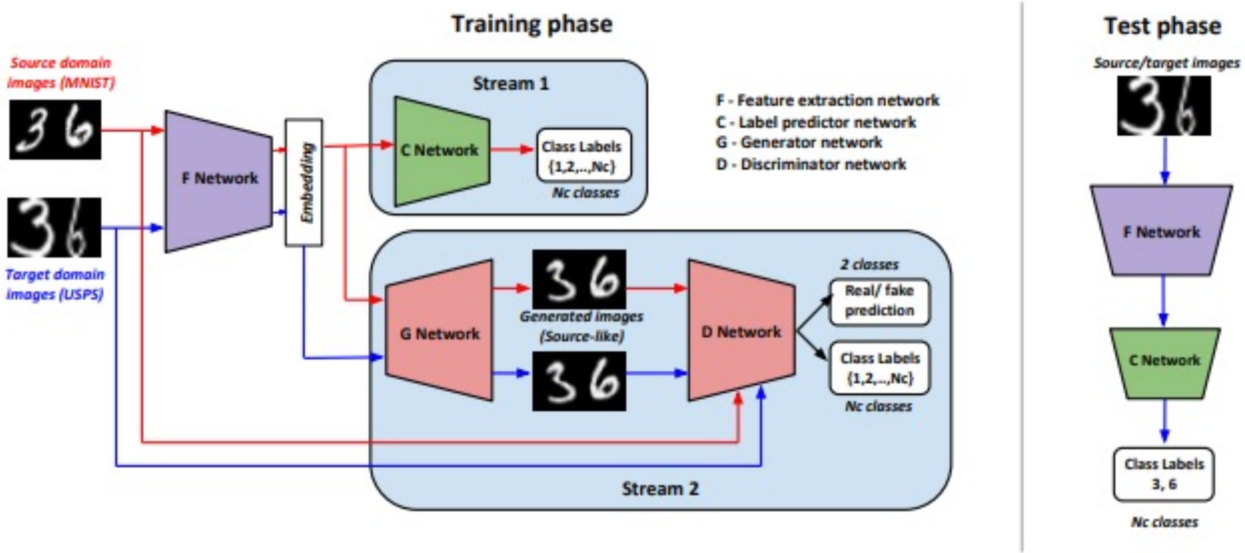
(b) Method 2 – training (left), testing (right)

Less common, sometimes in ensemble

Source to target or target to source?

Reconstruction as mapping

Recall the generate-to-adapt method



This is a type of target-to-source mapping!

More generally: reconstruction from invariant representation ↔ domain map

Results of reconstruction-based methods

Adding some context:

reconstruction is a middle ground between mapping and invariance

Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST
Labeled target (oracle)	96.5	99.2	99.5
CycleGAN*	95.6	96.4	70.3
Gen. to. Adapt	92.8	90.8	92.4
DSN	91.3		82.7
DANN	85.1 (95.7)	(90.0)	73.6
Source only	78.9	69.6	59.2

Bringing together adversaries and mappings

Invariance and mapping have complementary strengths

Can we combine all of the ideas today?

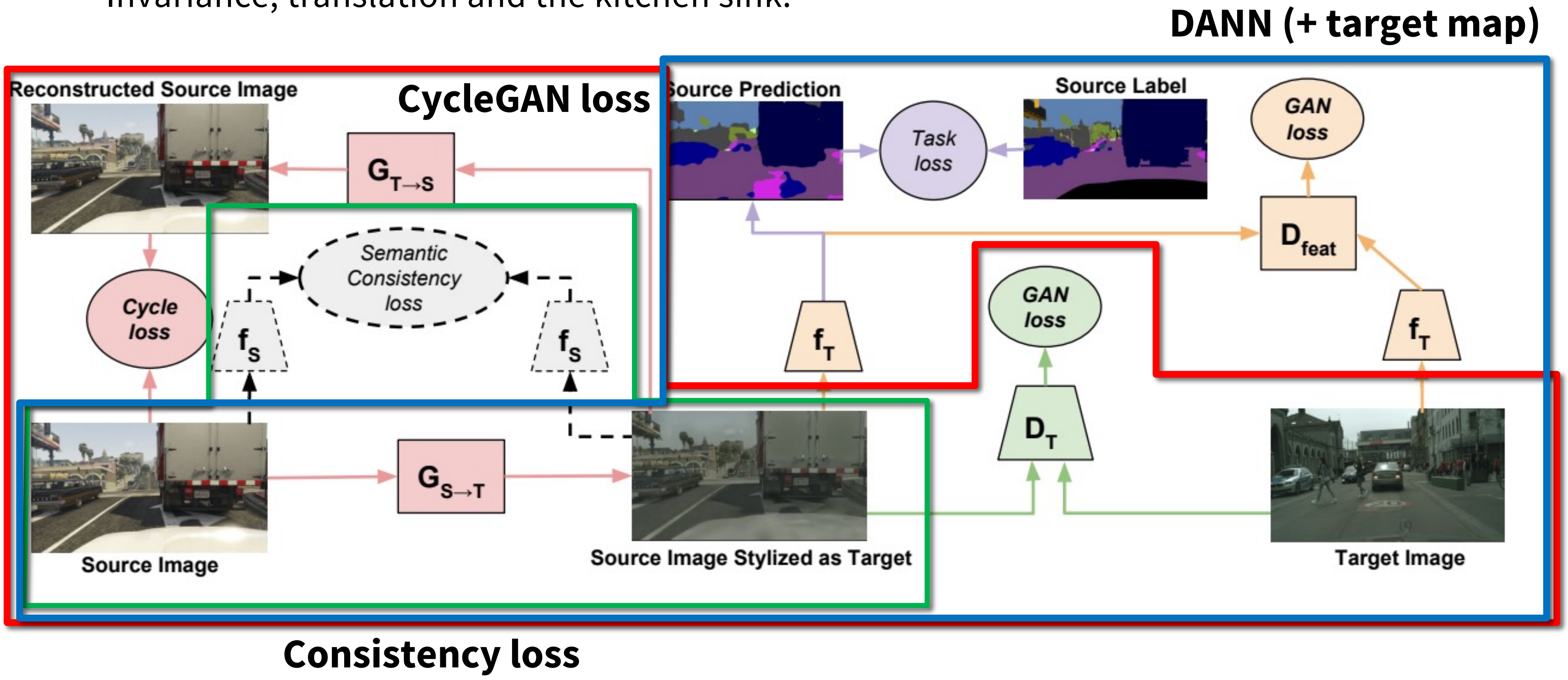
- Mapping
  - Cycle consistency
  - Semantic consistency

- Invariance
  - Adversarial prediction of domain identity

The result: CyCADA

CyCADA

Invariance, translation and the kitchen sink:



One final slide of MNIST SVHN

Upshot: dramatic improvements in more distant domains

Method	MNIST->USPS	USPS->MNIST	SVHN->MNIST
Labeled target (oracle)	96.5	99.2	99.5
CyCADA	95.6	96.5	96.7
CycleGAN*	95.6	96.4	70.3
DANN	85.1 (95.7)	(90.0)	73.6
Source only	78.9	69.6	59.2

Works on more challenging GTA → Cityscapes data as well

Invariance vs domain mapping

Invariance

"There exists a shared, useful representation for both domains"

Can handle very different domains (by discarding information)

Domain mapping

"There is a direct correspondence between two domains"

Much stronger assumption. Fails under large domain shifts (MNIST-SVHN)

Combining the two

In theory: still need a domain mapping to work

In practice: the invariance part can account for inexact domain maps

Preserving the conditional

In both cases:

Learning a valid invariance / mapping is *not* the hard part

Learning a label-preserving invariance / mapping is hard

Tricks we learned:

- Reconstruction penalties / losses for invariance
- Enforcing consistency to a pre-trained classifier
- Using unsupervised mapping / translation methods

A huge diversity of methods

- This lecture covers only a few well-known methods
- There’s a huge zoo of methods, with minor variations in loss and architecture
- Invariance (blue) and mapping (red) have been the majority of adaptation methods

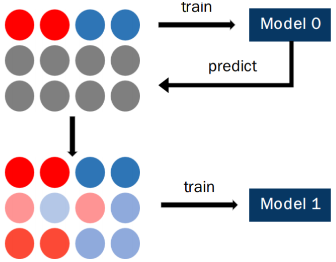
Name	Method	Loss Functions					Adversarial Loss		Generator	Shared Weights
		Distance	Diff.	Cycle	Sem.	Task	Feature	Pixel		
CAN[114]	DI,N	CCD				✓				not BN
French et al.[68]	En,N	sq. diff.				✓				EMA
Co-DA[124][a]	DI,En,N,TD	L1	✓			✓	✓			optional
VADA[220][a]	DI,TD					✓	✓			✓
DeepJDOT[50]	DI	JDOT				✓				✓
CyCADA[96]	DI,DM			✓	✓	✓	✓	✓	✓	
Gen. to Adapt[208]	DI					✓	✓		✓	✓
SimNet[187]	DI	prototypes					✓			
MADA[183]	DI,En					✓	✓			✓
MCD[206]	DI,En,TD	✓	✓			✓	✓			
GAGL[250]	DI,TD					✓	✓	✓	✓	✓
SBADA-GAN[201][b]	DM				✓	✓		✓	✓	
MCA[278]	DI	MCA				✓				✓
CCN++[101]	DI	clusters					✓			✓
M-ADDA[127]	DI	clusters				✓	✓			
Rozant. et al.[199]	DI	MMD				✓				regularize
XGAN[197]	DM			✓			✓	✓	✓	some
StarGAN[41]	DM			✓				✓	✓	✓
PixelDA[21]	DM				✓	✓		✓	✓	✓
AutoDIAL[27]	N,TD					✓				not BN
AdaBN[145]	N									not BN
JAN-A[151]	DI	JMMD				✓	✓			✓
LogCORAL[249]	DI	logCOR, mean				✓				✓
Log D-CORAL[172]	DI	logDCOR				✓				✓
VRADA[189]	DI					✓	✓			✓
ATT[204]	En		✓			✓				✓
SimGAN[219]	DM							✓	✓	N/A[c]
ADDA[241]	DI					✓	✓			
CycleGAN[290]	DM			✓				✓	✓	[d]
RegCGAN[160]	DM					✓	✓		✓	✓
Sener et al.[214]	DI	k-NN								✓
DSN[22]	DI		✓	✓		✓	✓			some
DRCN[76]	DI			✓		✓				✓
CoGAN[143]	DM					✓	✓		✓	some
Deep CORAL[226]	DI	CORAL				✓				✓
DANN[1, 72, 73]	DI					✓	✓			✓
DAN[147]	DI	MK-MMD				✓				low
Tzeng et al.[240][e]	DI					✓	✓			✓

(Wilson and cook 2020)

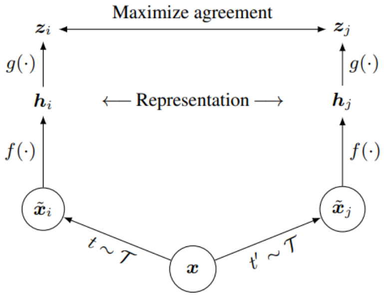
What's left?

We'll leave two major additional ideas for a future lecture

Self-training: using source domain predictions to label unlabeled data



Self-supervision: using target domain data to regularize the model



Summary for today

Two major families of methods:

Invariance : key decision – measuring invariance

Domain classification (DANN)

MMD (Coral / DAN)

Optimal transport (DeepJDOT)

Mapping: key decision – constraining the mapping

Cycle consistency (CycleGAN)

Geometry / Commutativity (GcGAN)

Combinations:

Reconstruction methods (Generate to Adapt)

Map+invariance (CyCADA)