Week 10: Natural Language Processing (NLP)

Vinay P. Namboodiri

Topic 8: Word Embedding
- GloVe

GloVe

- Proposed by Pennington et al. 2014.
- Unlike Word2vec, GloVe does not reply just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence matrix) to obtain word vectors.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

GloVe

Counts the co-occurrence between words w_k, w_i

$$P(w_k|w_i) = \frac{C(w_k, w_i)}{C(w_i)}$$

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

"solid" is more related to "ice" than "steam"

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

"Water" is equally (ir)relavant to "ice" and "steam"

Intuition: **co-occurence probabilities ratios** gathers more information than the raw probabilities and better capture relevant information about words' relationship

What is the function $F(\cdot)$?

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Since the goal is to learn meaningful word vectors, $F(\cdot)$ is designed to be a function of the linear difference between two words w_i and w_j

$$F((w_i - w_j)^T w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

The final solution is to $F(\cdot)$ as an **exponential** function.

$$F\left((w_i - w_j)^T w_k\right) = \exp\left((w_i - w_j)^T w_k\right) = \frac{\exp({w_i}^T w_k)}{\exp\left({w_j}^T w_k\right)} = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Loss function of GloVe

Replace it with a bias term b_i

$$w_i{}^T w_k = log\frac{C(w_k,w_i)}{C(w_i)} = \log C(w_k,w_i) - \log C(w_i)$$

$$\log C(w_k,w_i) = w_i{}^T w_k + b_i + b_k$$

To keep the symmetric form, we also add in a bias term b_k

The loss function for the GloVe model is designed to preserve the above formula by minimizing the sum of the squared errors

$$\mathcal{L} = \sum_{i=1,j=1}^{V} (w_i^T w_j + b_i + b_j - \log C(w_j,w_i))^2$$

Add a weighting function f() that is used to downweight the importance of very frequent co-occurrences

$$\mathcal{L} = \sum_{i=1,j=1}^{V} f(C(w_j,w_i))(w_i^T w_j + b_i + b_j - \log C(w_j,w_i))^2$$

Loss function of GloVe

To penalize the difference between the dot product of two word vectors and the logarithm of the co-occurrence count, with the bias terms added.

$$\mathcal{L} = \sum_{i=1,j=1}^{V} f(C(w_j, w_i))(w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$

w_i, w_j are the word vectors for words i and j

b_i, b_j are bias terms

$$f(c) = \begin{cases} \left(\frac{c}{c_{\max}}\right)^{\alpha} & \text{if } c < c_{\max},\ c_{\max} \text{ is adjustable.} \\ 1 & \text{if otherwise} \end{cases}$$

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Advantages of GloVe

- Computationally efficient
- Scales well to large datasets
- Produces embedding that capture both syntactic and semantic relationships between words.

GloVe vs. Word2Vec

- Both are popular algorithms for generating word embeddings.
- Both are unsupervised learning algorithms
- Both are able to capture semantic relationships between words.

Word2Vec

Use a neural network to learn embedding

Focus more on local context

Is able to handle larger corpora of text

GloVe

Based on co-occurrence matrix

Capture global relationships between words.

Is generally faster than Word2Vec

Reference

- Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).
- https://nlp.stanford.edu/projects/glove/
- Lectures of CS224n: NLP with Deep learning. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html
- Matyas Amrouche's blog: Word embedding (Part II). https://towardsdatascience.com/word-embedding-part-ii-intuition-and-some-maths-to-understand-end-to-end-glove-model-9b08e6bf5c06
- Lilian Weng's blog: Learning Word Embedding. https://lilianweng.github.io/posts/2017-10-15-word-embedding/

Week 11:Domain Adaptation

Vinay P. Namboodiri

Introduction

- So far, most of the techniques we have considered assume the availability of full supervision for training through a training dataset
- However, in many practical scenarios, this is not true
- For instance, for an autonomous car, you may have trained a pedestrian detection algorithm in summer
- Then it snows……….

Problem

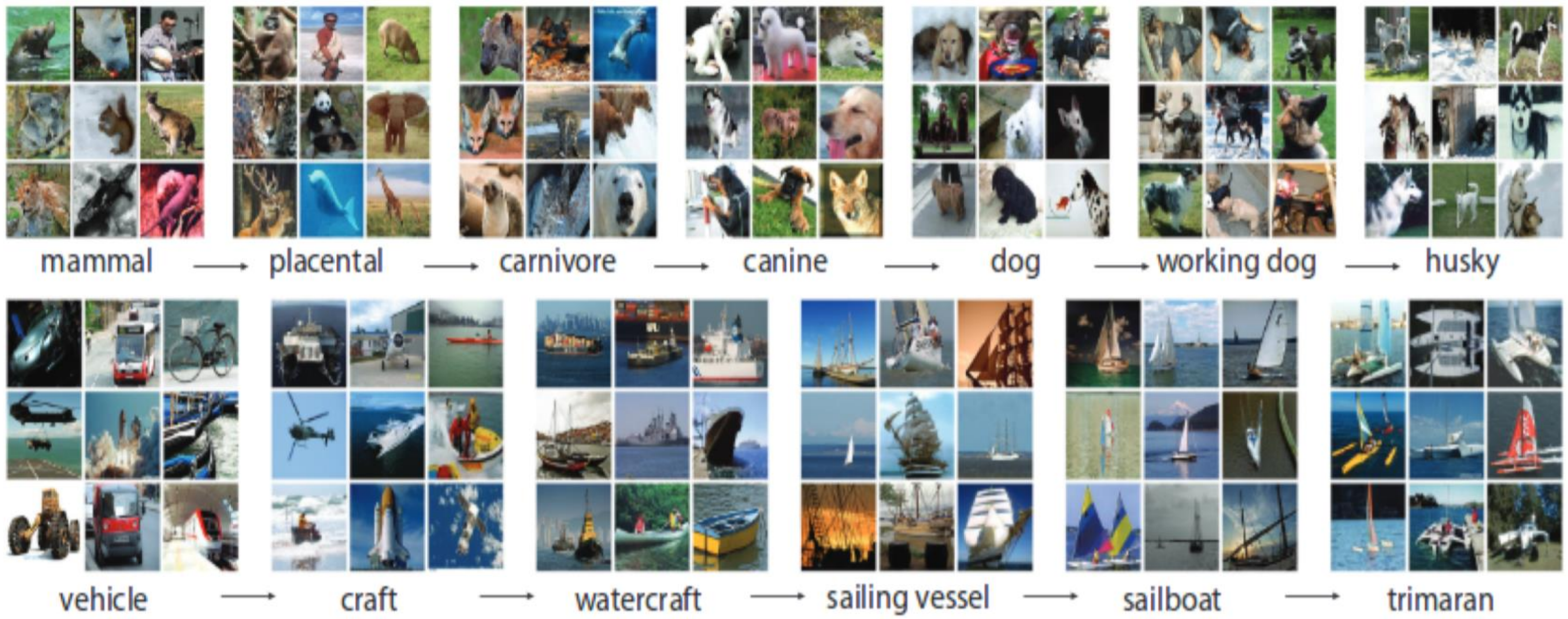


Figure : Typical Computer Vision Dataset

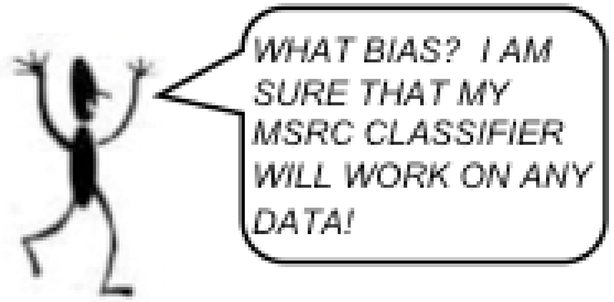
Generally, training and test instances are chosen from the same dataset and hence they are from same probability distribution. Also labels are free of noise, objects are centered and background clutter is less.

Challenge

Figure : Real World Computer Vision Dataset

Real world data is noisy. Multiple instances of different object can be present in an image and also usually much more clutter is there.

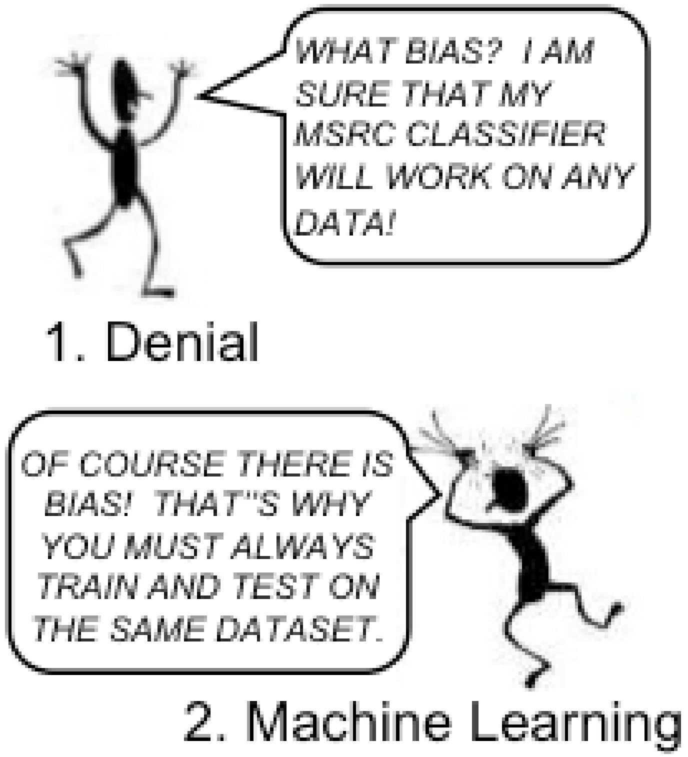
Problem



1. Denial

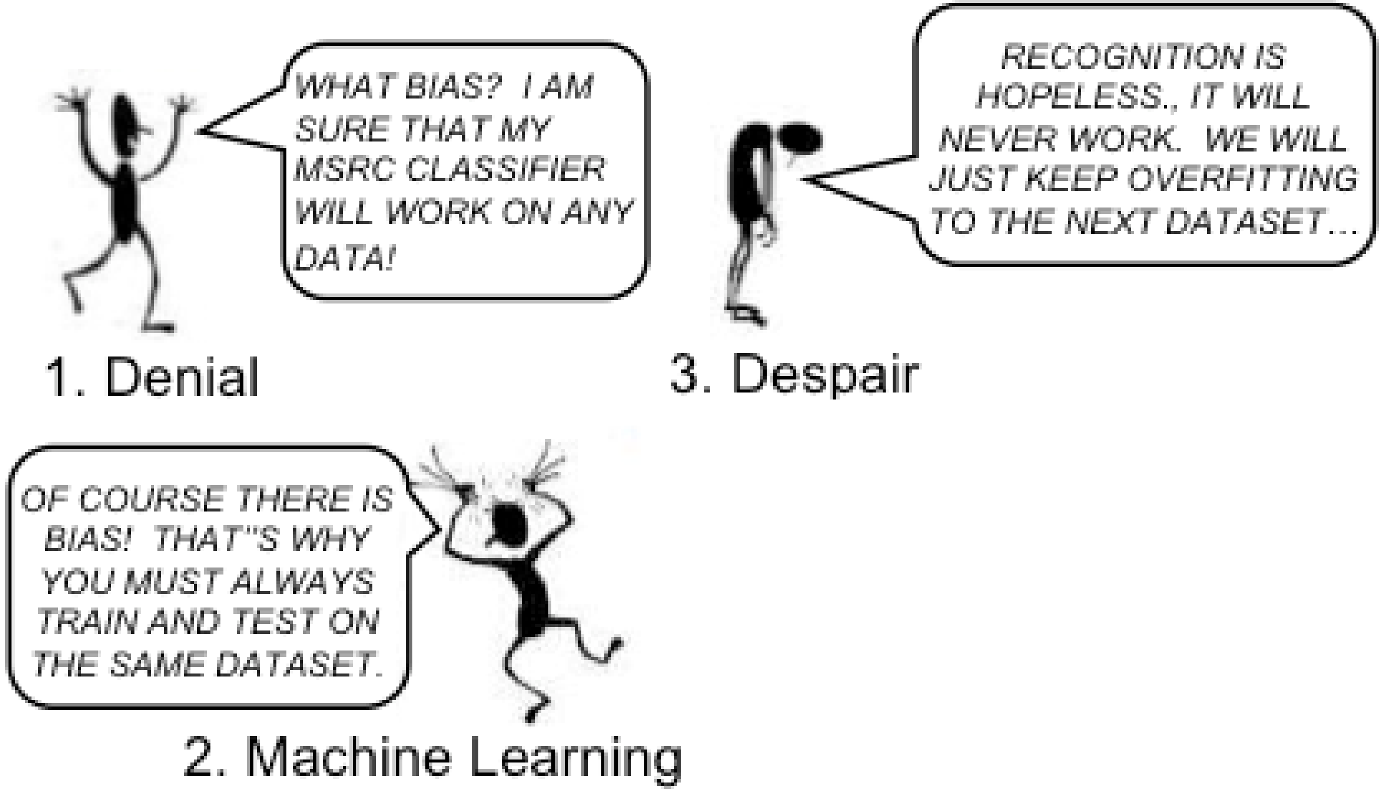
Source: Torralba and Efros
http://people.csail.mit.edu/torralba/research/bias/

Problem



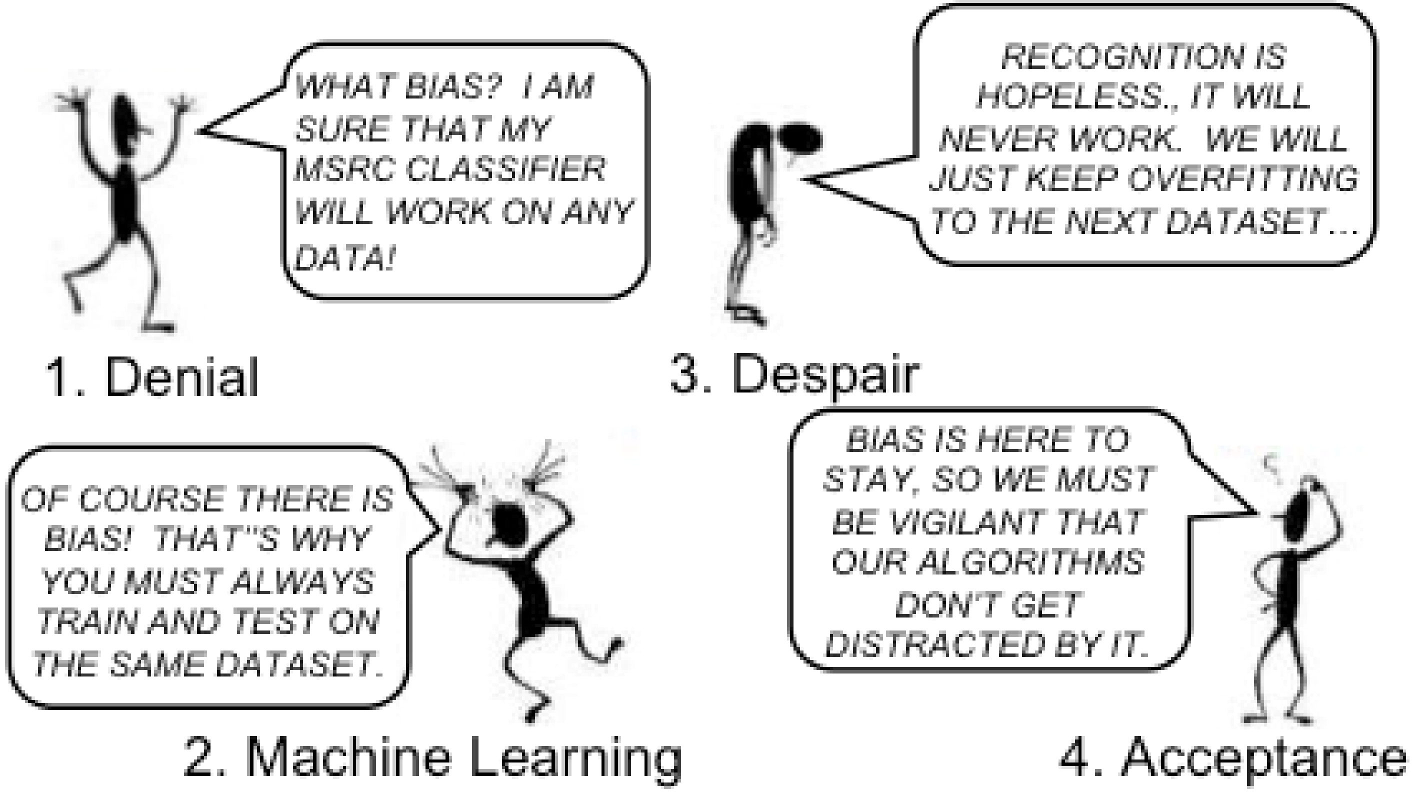
Source: Torralba and Efros
http://people.csail.mit.edu/torralba/research/bias/

Problem



Source: Torralba and Efros
http://people.csail.mit.edu/torralba/research/bias/

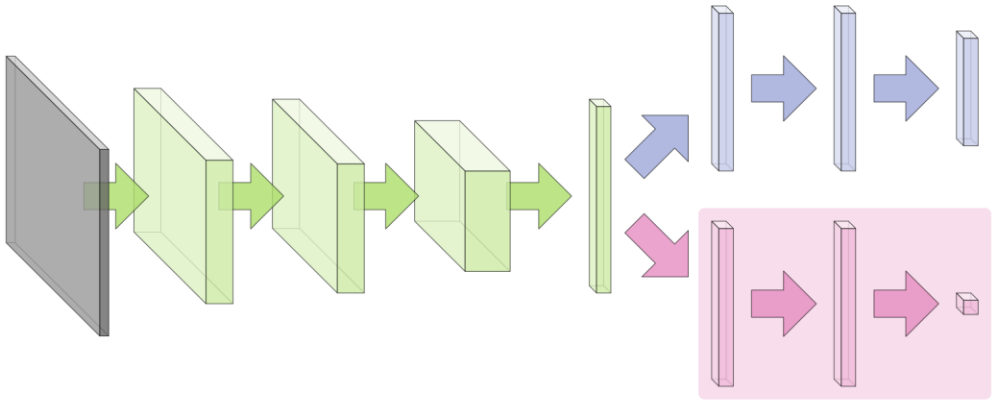
Problem



Source: Torralba and Efros
http://people.csail.mit.edu/torralba/research/bias/

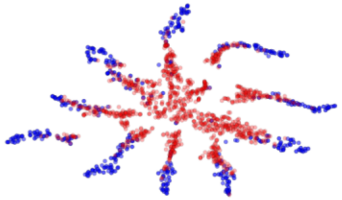
Deep Learning-based Domain Adaptation

Deep Unsupervised domain adaptation by back propagation



- Computes $d = G_d(\mathbf{f}; \theta_d)$
- Is trained to predict **0** for **source** and **1** for **target**
- Therefore, the domain loss

is **low** for



is **higher** for

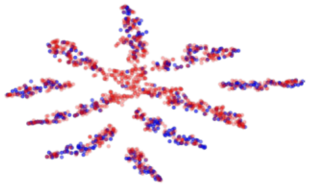


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Assumptions

- There are
- lots of labeled examples in source domain
- lots of unlabelled examples in target domain
- We want a deep neural network that does well on target domain

Deep Unsupervised domain adaptation

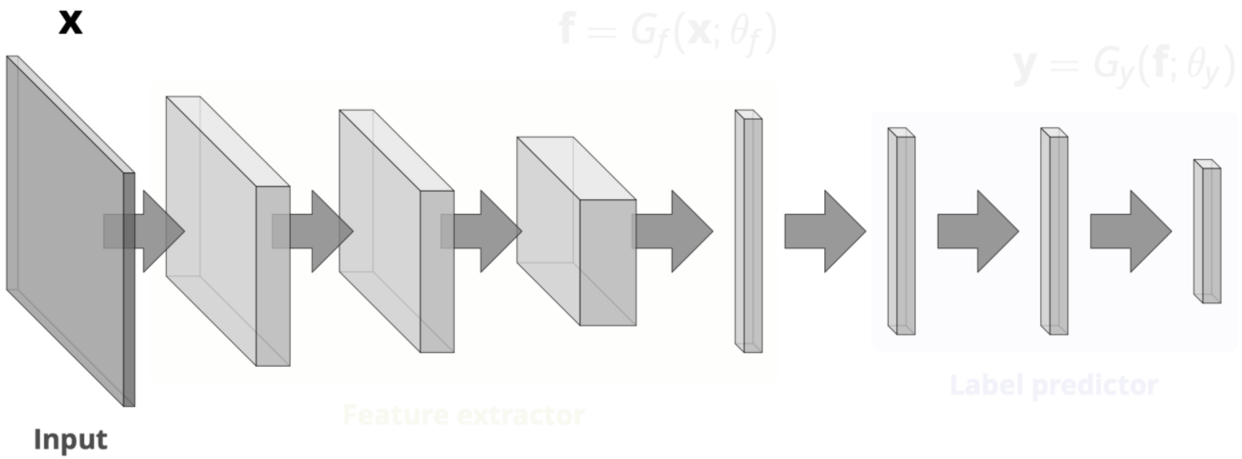


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

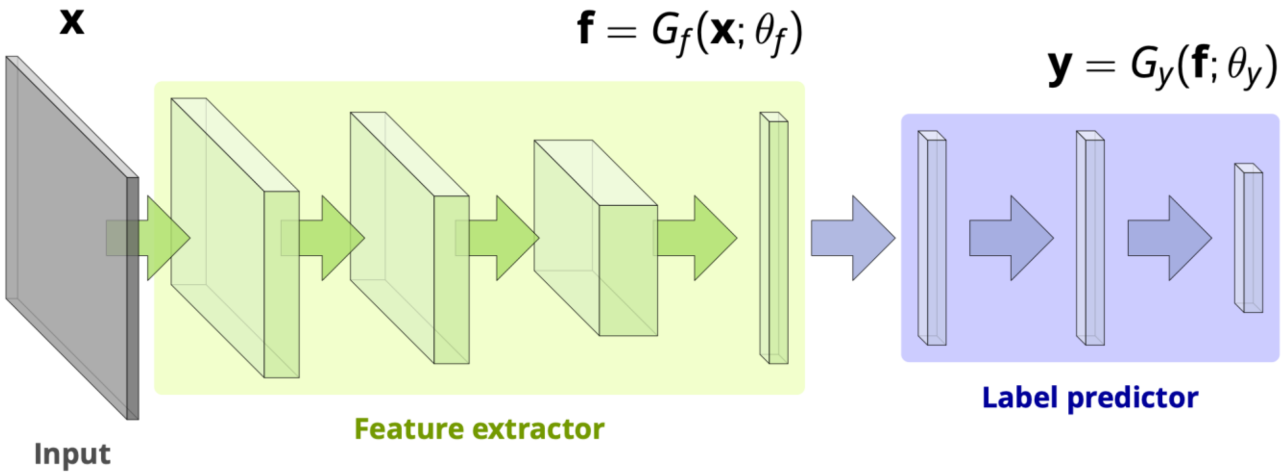
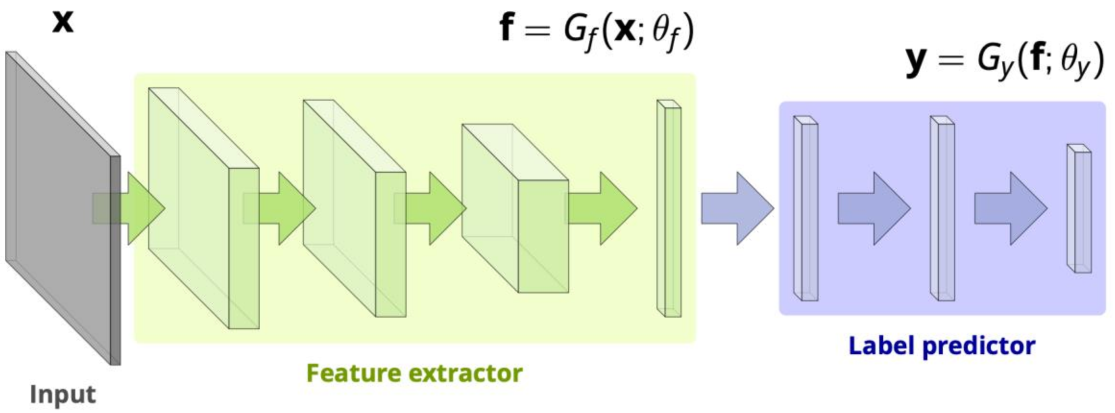


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation



When trained on **source only**, feature distributions **do not match**.

$$S(\mathbf{f}) = \{G_f(\mathbf{x}; \theta_\mathbf{f}) \,|\, \mathbf{x} \sim S(\mathbf{x})\}$$
$$T(\mathbf{f}) = \{G_f(\mathbf{x}; \theta_\mathbf{f}) \,|\, \mathbf{x} \sim T(\mathbf{x})\}$$

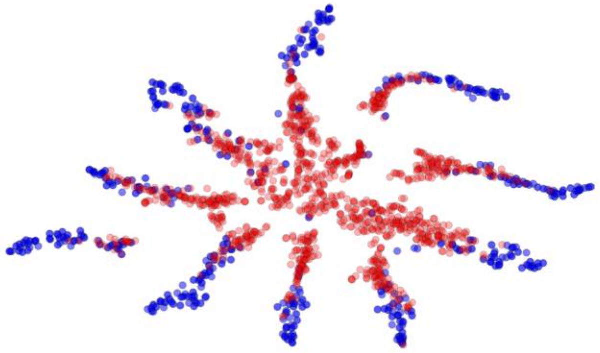


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

When trained on **source only**, feature distributions **do not match**.

Our goal is to get this:

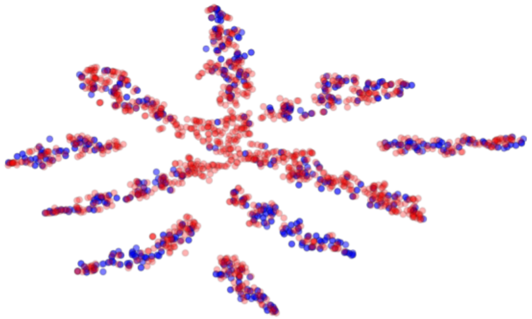


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

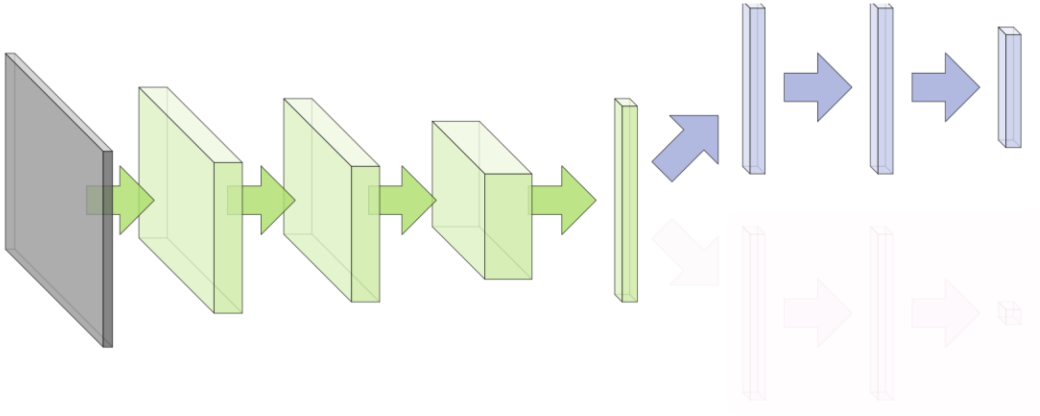
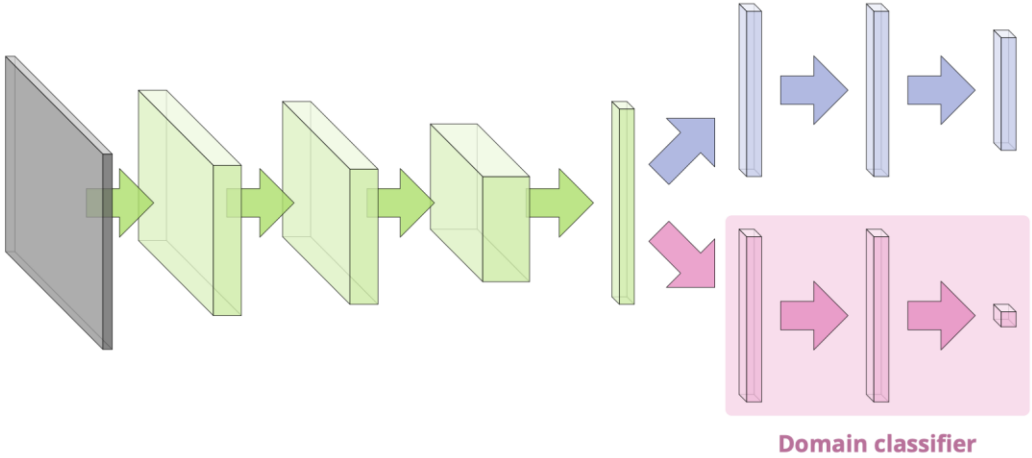


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

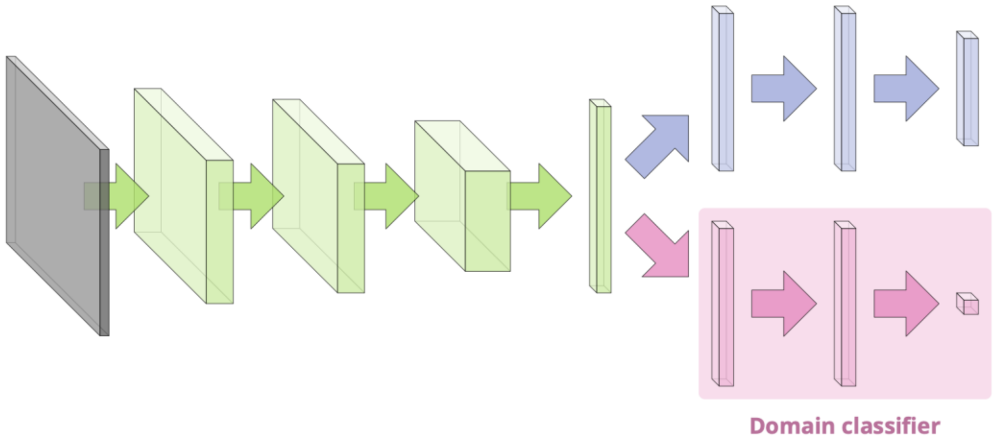
Deep Unsupervised domain adaptation



- Computes $d = G_d(\mathbf{f}; \theta_d)$

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation



- Computes $d = G_d(\mathbf{f}; \theta_d)$
- Is trained to predict **0** for **source** and **1** for **target**
- Therefore, the domain loss

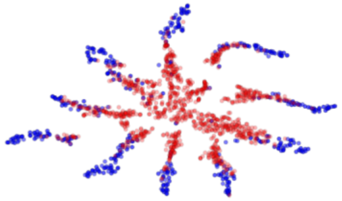
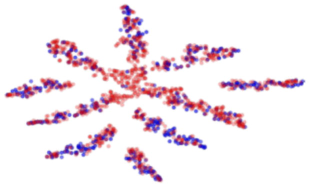
is **low** for  is **higher** for 

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

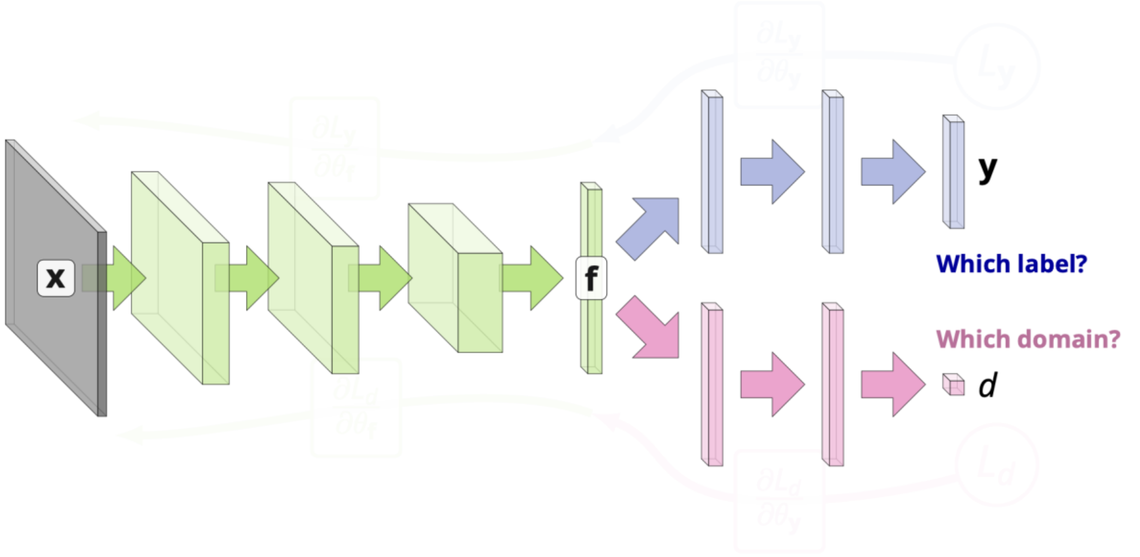
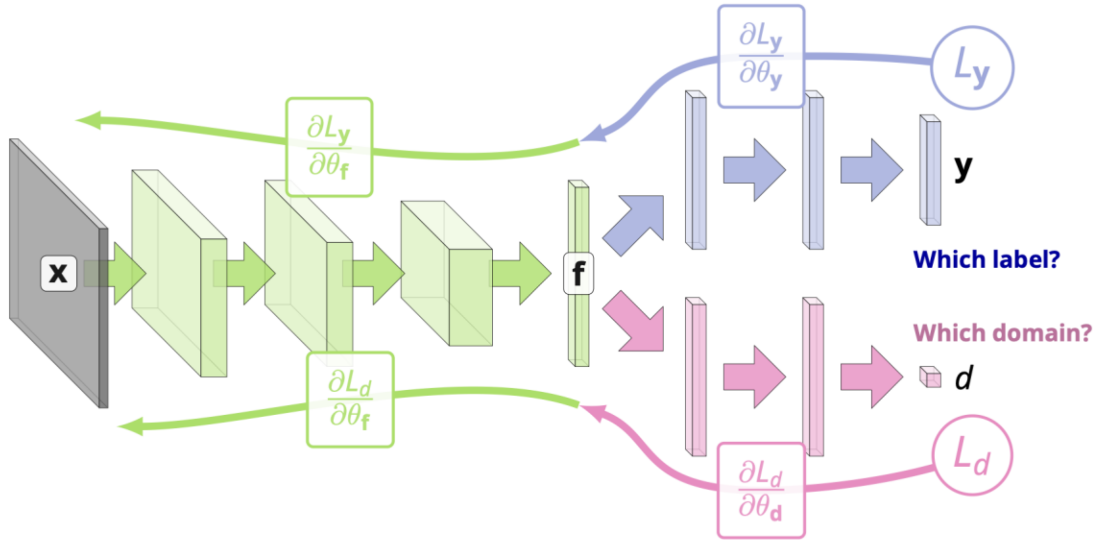


Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

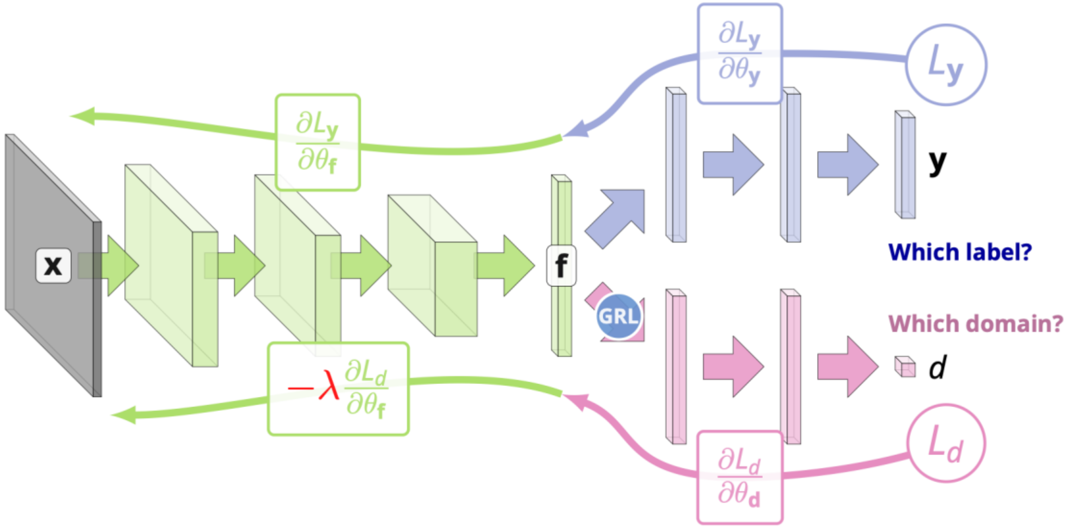


Let's try *standard backpropagation*. **Emerging features** are:

- Discriminative (i.e. good for predicting **y**)
- Domain-discriminative (i.e. good for predicting *d*)

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

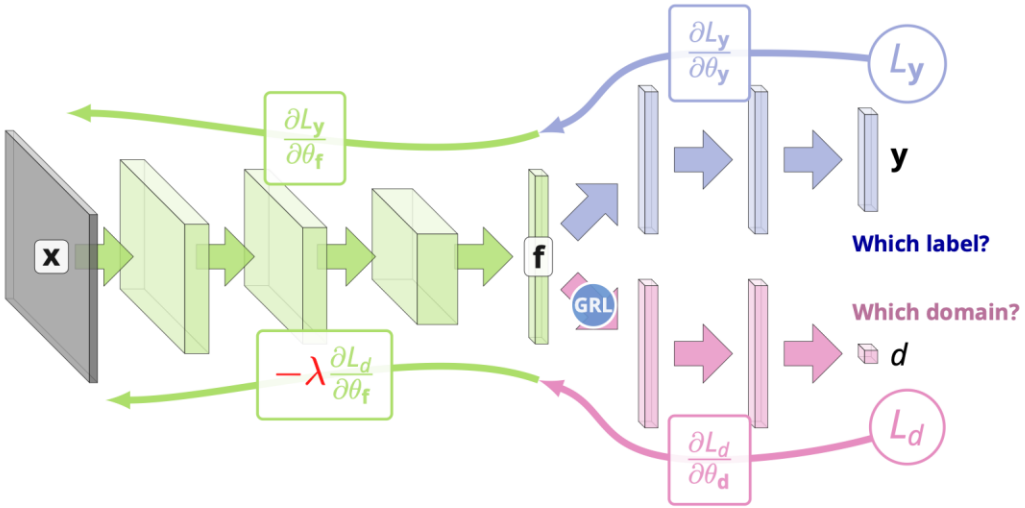


Let's now inject the **Gradient Reversal Layer**:

- Copies data without change at *fprop*
- Multiplies deltas by $-\lambda$ at *bprop*

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

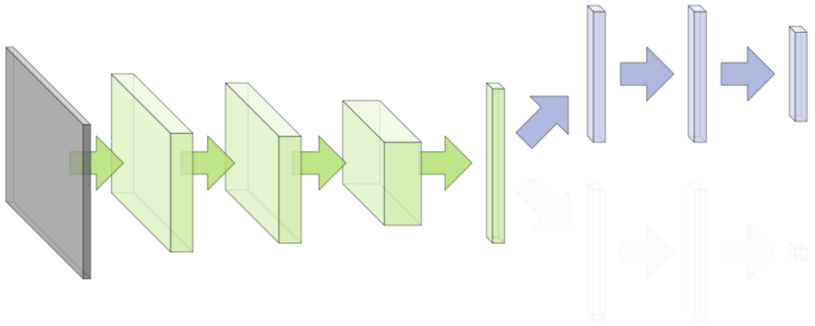


Emerging features are now:

- Discriminative (i.e. good for predicting **y**)
- Domain-invariant (i.e. not good for predicting *d*)

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

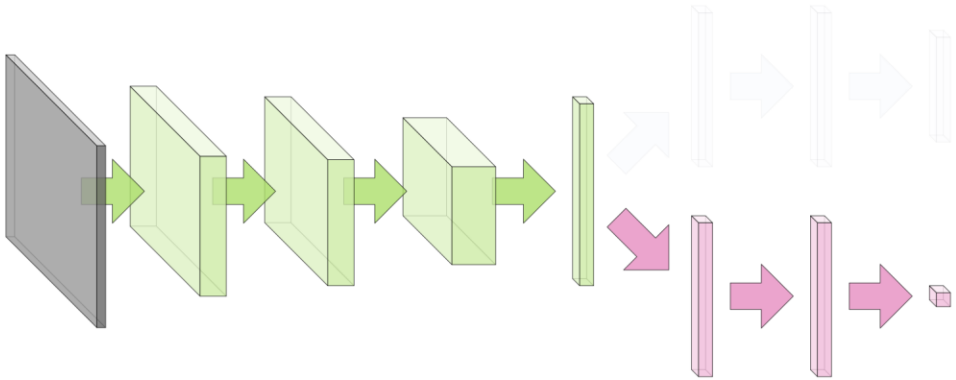
Deep Unsupervised domain adaptation



1. Train **feature extractor** + **label predictor** on **source**
2. Train **feature extractor** + **domain classifier** on **source** + **target**
3. Use **feature extractor** + **label predictor** at **test time**

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation



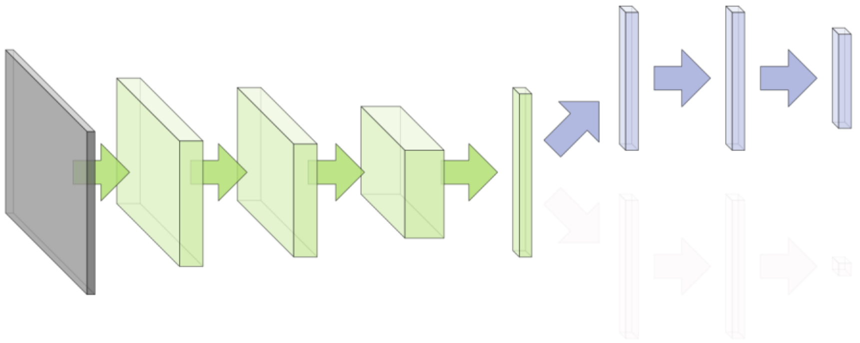
1 Train **feature extractor** + **label predictor** on **source**

2 Train **feature extractor** + **domain classifier** on **source** + **target**

3 Use **feature extractor** + **label predictor** at **test time**

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation



1 Train **feature extractor** + **label predictor** on **source**

2 Train **feature extractor** + **domain classifier** on **source** + **target**

3 Use **feature extractor** + **label predictor** at **test time**

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Deep Unsupervised domain adaptation

METHOD (SOURCE / TARGET)	AMAZON / WEBCAM	DSLR / WEBCAM	WEBCAM / DSLR
GFK(PLS, PCA) (GONG ET AL., 2013)	.197	.497	.631
SA (FERNANDO ET AL., 2013)	.450	.648	.699
DLID (S. CHOPRA & GOPALAN, 2013)	.519	.782	.899
DDC (TZENG ET AL., 2014)	.618	.950	.985
DAN (LONG & WANG, 2015)	.685	.960	.990
SOURCE ONLY	.642	.961	.978
PROPOSED APPROACH	**.730**	**.964**	**.992**

Protocol: all of the methods above use

- **all** available **labeled** **source** samples
- **all** available **unlabeled** **target** samples

Fig. credit: Yaroslav Ganin
Yaroslav Ganin and Viktor Lempitsky
Unsupervised domain adaptation by back propagation
ICML 2015

Domain Adaptation using Adaboost

Domain Adaptation Setting

- Example: We want to obtain sentiment analysis for movies
- We have a small amount of data for movies for the year 2020
- This data is insufficient to train for sentiment analysis

Sentiment Analysis dataset for year 2020

- We also have some additional training data for movies released in 1990-1995
- How can we best make use of the old training data

Sentiment Analysis dataset for years 1990-1995

Domain Adaptation Setting

Sentiment Analysis dataset for year 2020

Option 1: Train only with Y2020 data

Problem: Data not enough

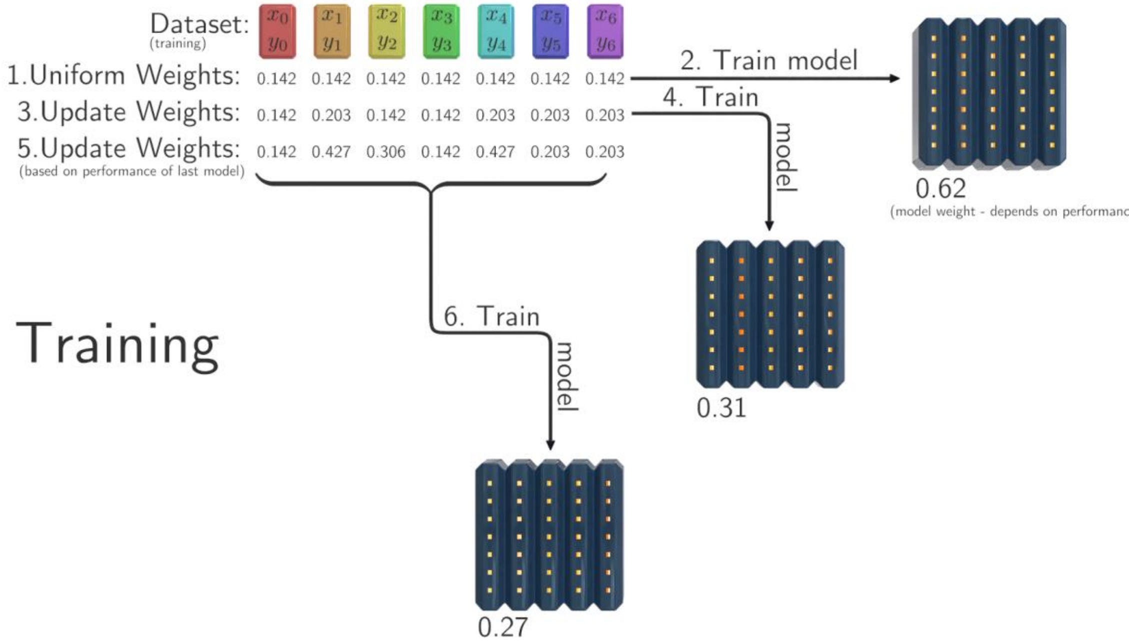
Sentiment Analysis dataset for year 2020 | Sentiment Analysis dataset for years 1990-1995

Option 2: Put all data together and train

Problem: While some of the old data is relevant, some of it is not very relevant

How to Solve?

- Can we find out the data that is relevant from old data
- How do we go about it
- Consider that you have been taught Adaboost and you are a big fan of it :)
- Can you use Adaboost and solve the problem?



Solution: TrAdaboost

- Main idea:
- Let Xs be source dataset, (the old dataset)
- Let Xd be the new labeled target dataset (the new dataset)
- Train using all samples
- If a sample is misclassified, check which dataset it comes from
- If it is source dataset, then decrease its weight, if it is target dataset then increase its weight

Boosting for Transfer Learning

Wenyuan Dai DWYAK@APEX.SJTU.EDU.CN
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
Qiang Yang QYANG@CSE.UST.HK
Deptarment of Computer Science, Hong Kong University of Science and Technology, Hong Kong
Gui-Rong Xue GRXUE@APEX.SJTU.EDU.CN
Yong Yu YYU@APEX.SJTU.EDU.CN
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China

TrAda Boost: ICML 2007

And it works

Table 4. The error rates when semi-supervised learning

Data Set	TSVM	TSVMt	TrAdaBoost(TSVM)
rec vs talk	0.059	0.040	**0.021**
rec vs sci	0.067	0.062	**0.013**
sci vs talk	0.173	0.106	**0.075**
auto vs aviation	0.043	0.103	**0.038**
real vs simulated	0.144	0.131	**0.102**
orgs vs people	0.358	0.292	**0.248**
orgs vs places	0.424	0.436	**0.304**
people vs places	0.307	0.225	**0.179**
edible vs poisonous	0.439	0.179	**0.160**

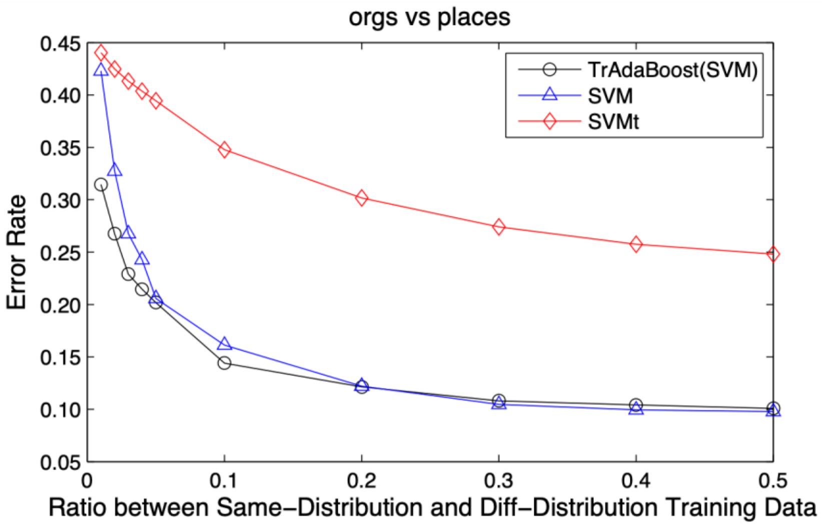


Figure 2. The error rate curves on **orgs vs places** data set for three classifiers **TrAdaBoost(SVM)**, **SVM** and **SVMt**

Results from: TrAdaboost paper, ICML 2007

TrAdaboost Algorithm

TrAdaboost Algorithm

Input: The source dataset that is fully labeled (X_s, Y_s) consisting of n samples and the destination dataset X_d, Y_d consisting of m samples such that $m < n$, a baseline learner that provides a hypothesis $h(x) \rightarrow Y$. Initialise $\hat{w}_{0(n+1,\cdots,n+m)} = \left[\frac{1}{m}, \ldots, \frac{1}{m}\right]$. For $t = 1$ to N do,

TrAdaboost Algorithm

Input: The source dataset that is fully labeled (X_s, Y_s) consisting of n samples and the destination dataset X_d, Y_d consisting of m samples such that $m < n$, a baseline learner that provides a hypothesis $h(x) \to Y$. Initialise $\hat{w}_{0(n+1,\cdots,n+m)} = \left[\frac{1}{m}, \ldots, \frac{1}{m}\right]$. For $t = 1$ to N do,

1. Train model M_t with $\hat{w}_t$ *weighted data*

TrAdaboost Algorithm

Input: The source dataset that is fully labeled (X_s, Y_s) consisting of n samples and the destination dataset X_d, Y_d consisting of m samples such that $m < n$, a baseline learner that provides a hypothesis $h(x) \rightarrow Y$. Initialise $\hat{w}_{0(n+1,\cdots,n+m)} = \left[\frac{1}{m}, \ldots, \frac{1}{m}\right]$. For $t = 1$ to N do,

1. Train model M_t with $\hat{w}_t$ *weighted data*
2. Calculate weighted error over X_d

$$\mathrm{e}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=n+1}^{n+m} w_{ti} L_{ti}$$

where L_{ti} is zero-one loss of exemplar i for model M_t

TrAdaboost Algorithm

Input: The source dataset that is fully labeled (X_s, Y_s) consisting of n samples and the destination dataset X_d, Y_d consisting of m samples such that $m < n$, a baseline learner that provides a hypothesis $h(x) \to Y$. Initialise $\hat{w}_{0(n+1,\cdots,n+m)} = \left[\frac{1}{m}, \ldots, \frac{1}{m}\right]$. For $t = 1$ to N do,

1. Train model M_t with $\hat{w}_t$ *weighted data*
2. Calculate weighted error over X_d

$$\mathrm{e}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=n+1}^{n+m} w_{ti} L_{ti}$$

   where L_{ti} is zero-one loss of exemplar i for model M_t

3. Model weight

$$\alpha_t = \log\left(\frac{1-\mathrm{e}_t}{\mathrm{e}_t}\right), \text{ and } \alpha = \frac{1}{(1+\sqrt{2\ln n/N})}$$

TrAdaboost Algorithm

Input: The source dataset that is fully labeled (X_s, Y_s) consisting of n samples and the destination dataset X_d, Y_d consisting of m samples such that $m < n$, a baseline learner that provides a hypothesis $h(x) \to Y$. Initialise $\hat{w}_{0(n+1,\cdots,n+m)} = \left[\frac{1}{m}, \ldots, \frac{1}{m}\right]$. For $t = 1$ to N do,

1. Train model M_t with $\hat{w}_t$ *weighted data*
2. Calculate weighted error over X_d

$$\mathrm{e}_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=n+1}^{n+m} w_{ti} L_{ti}$$

   where L_{ti} is zero-one loss of exemplar i for model M_t
3. Model weight

$$\alpha_t = \log\left(\frac{1-\mathrm{e}_t}{\mathrm{e}_t}\right), \text{ and } \alpha = \frac{1}{(1+\sqrt{2\ln n/N})}$$

4. Update weights (only changes weights of exemplars it got wrong)

$$w_{t+1,i} = w_{ti} \exp(\alpha_t L_{ti}) \text{ for } i \in (n+1, n+m)$$
$$w_{t+1,i} = w_{ti} \exp(-\alpha L_{ti}) \text{ for } i \in (1, n)$$

Related Reading List

- Boosting for transfer learning, by Wenyuan Dai, Qiang Yang, Gui-Rong Xue, Yong Yu, ICML 2007
- Unsupervised domain adaptation by back propagation, Yaroslav Ganin and Viktor Lempitsky, ICML 2015