DOMAIN ADAPTATION WITH DEEP NEURAL NETWORKS: A GENTLE INTRODUCTION

Bill Yuchen Lin

University of Southern California
yuchen.lin@usc.edu

October 7, 2019

ABSTRACT

Domain adaptation is an essential problem in transfer learning, focusing on how to exploit labeled data in familiar domains/tasks for a new target one of interest where supervision is limited. Recent advances in deep learning research have shown that neural networks are highly transferable across domains and tasks, with specific techniques and training strategies. In this gentle introduction, we first formally introduce domain adaptation in the broader context of transfer learning, revisit a few classic methods in conventional machine learning, and finally introduce some state-of-the-art works on neural domain adaptation (most are based on adversarial training). We conclude this paper with a discussion on current research in natural language processing (NLP) applications and future directions.

Keywords Transfer Learning · Domain Adaptation · Natural Language Processing · Deep Neural Networks

"`Domain adaptation establishes knowledge transfer from the labeled source domain to the unlabeled target domain by exploring domain-invariant structures that bridge different domains of substantial distribution discrepancy.`" (by Pan and Yang in their famous 2010 survey.)

1 Transfer Learning and Domain Adaptation

Transfer learning, originating from cognitive science, is now a broad area in artificial intelligence including a lot of research topics about learning to share knowledge between multiple tasks or domains. We focus on **Domain Adaptation**[1], a significant sub-area in transfer learning. Domain adaptation aims to exploit data from data-sufficient domains (source domains) for learning better models in other domains (target domains) that have no/limited labels.

1.1 Problem formation

We now mathematically formulate general domain adaptation problems using the notation shown in Table 1. Given one source domain $\mathcal{D}_S$ and a corresponding task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and the task $\mathcal{T}_T$. Transfer learning, or domain adaptation in particular, is the process of enhance the performance of the target predictive function $f_T(\cdot)$ using the knowledge from $\mathcal{D}_S$ with $\mathcal{T}_S$. We usually believe $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$ in domain adaptation, although the definition of a task and a domain can be ambiguous in practice[2].

The assumption that $\mathcal{D}_S \neq \mathcal{D}_T$ is called **domain shift** (a.k.a domain discrepancy), which is usually due to the difference between input feature spaces ($\mathcal{X}_S \neq \mathcal{X}_T$) or the difference between joint distributions ($P_S(X,Y) \neq P_T(X,Y)$ within the same spaces. Another special case is when the output spaces are also changed $\mathcal{Y}_S \neq \mathcal{Y}_T$. For the first and third case, we name it as **heterogeneous** domain adaptation and the second case as **homogeneous** domain adaptation.

[1]Some people use the term "**transductive transfer learning**". Imagine if we see source data as training data and target data as the test, then we are are aware of all test inputs before doing inference, which is thus an indeed transductive learning problem.

[2]A good example here is "to ride a bike" versus "to ride a motorcyle". We can say "to ride" is a task, and "bike" and "motorcyle" are two different domains, while we can also simply think they are two different tasks.

Notation	Description	Notation	Description
$\mathcal{X}$	Input feature space	$P(X)$	Marginal distribution
$\mathcal{Y}$	Label space	$P(Y\|X)$	Conditional distribution
$\mathcal{T}$	Predictive learning task	$P(Y)$	Label distribution
Subscript S	Denotes source	$\mathcal{D}_S$	Source domain
Subscript T	Denotes target	$\mathcal{D}_T$	Target domain

Table 1: Summary of commonly used notation

1.2 Example Cases in Natural Language Processing

The above domain adaptation settings are all common in NLP applications. Say you would like to build a binary sentiment classifier (positive/negative) for laptop reviews in English, but you only have the labeled data in movie reviews (also in English) with the same label space (positive/negative). This is a typical setting of homogeneous domain adaptation ($\mathcal{X}_S = \mathcal{X}_T, \mathcal{Y}_S = \mathcal{Y}_T$, but $P_S(X, Y) \neq P_T(X, Y)$).

If someone wants to build a sentiment classifier for movies but on French reviews, this may be a heterogeneous domain adaptation (i.e. cross-lingual transfer) if they use English word embeddings and French word embeddings respectively ($\mathcal{X}_S \neq \mathcal{X}_T$). A promising direction in cross-lingual transfer is to first transforming multiple mono-lingual word embedding space into a common shared representation (i.e. multi-lingual word embeddings and multilingual BERT-like models), thus making $\mathcal{X}_S \simeq \mathcal{X}_T$.

A more difficult scenario is when label space is different ($\mathcal{Y}_S \neq \mathcal{Y}_T$). For example, in cross-domain named entity recognition [7], one wants to build a model for extracting entities in medical domains with the types of {`Drug`, `Disease`, `Symptoms`}, while the source domain data only contains entity types like {`Person`, `Location`, `Organization`}. In such settings, external priors or a adequate number of labeled examples in target domains may be necessary.

1.3 Other related research topics

Some recent works see **multi-task learning** (MTL) as an important special case of inductive transfer learning as well. In other words, given M domains[3] $\{\mathcal{D}_i = (\mathcal{X}_i, \mathcal{T}_i, \mathcal{Y}_i)\}^M$, multi-task learning aims to improve the performance of the predictor of each task $f_i(\cdot)$ by using the information from all the remaining domains. Note that there is no preference on a particular task, and we also assume the labels in each domain/task is sufficient.

For example in NLP, we may need to build a multi-task learning model for named-entity recognition, relation extraction and cor-reference resolution at the same time, because they are inherently dependent and using more out-of-domain data can be beneficial for each one of them.

Few-shot learning is a special case of transfer learning where labels in target tasks/domains are very few. Therefore, we need to specifically design learning algorithms such that model adaptation can be done rapidly. **Adversarial learning** and **network robustness** are also highly related, because we can see target domain examples as potential attacks aiming for lead source models to make mistakes.

In this gentle introduction, we mainly focus on **homogeneous domain adaptation**[4], but the techniques can be also applied in other problem settings.

2 Analysis of Domain Adaptation with Joint Distribution

We would like to more carefully analyze the most popular case in domain adaptation, Homogeneous DA, and answer this key question: how is it possible that someone can use the source domain labeled data ($X_S = \{x_1^S, x_2^S, \dots\}, Y_S = \{y_1^S, y_2^S, \dots\}$) for learning a target model f_T, while you have $X_T = \{x_1^T, x_2^T, \dots\}$ but no Y_T or $|Y_T| \ll |X_T|$?

Recall that the optimal function f_T^* for the target domain should minimize the expected loss with respect to the joint distribution in the target domain $P_T(X, Y)$:

$$f_T^* = \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P_T(x, y)\ L(x, y, f)$$

[3]We here intentionally misuse the term domain connect our previous formulation.

[4]From this line on, we use domain adaptation to refer *homogeneous domain adaptation* if without specification.

Note that x_i^S and x_j^T are both in the same shared input feature space $\mathcal{X}$, while similarly y_i^S and y_j^T are both in $\mathcal{Y}$. $\mathcal{H}$ denotes the hypothesis function space where we assume the optimal predictor should locate in. L is the loss function to measure the risk of using an arbitrary predictor f works on input x with underlying truth to be y.

In a typical supervised learning scenario, we should use $\tilde{P}_T(X,Y)$ to approximate $P_T(X,Y)$, such that we can have an empirical distribution based on $\{(x_1^T, y_1^T), (x_2^T, y_2^T), \dots\}$ (drawn i.i.d from $P_T(X,Y)$). However, it is impossible in domain adaptation problems since most of the y_i^T are missing.

We instead ask help from the source domain labeled data $\{(x_1^S, y_1^S), (x_2^S, y_2^S), \dots\}$ (drawn i.i.d from $P_S(X,Y)$):

$$\begin{aligned}
f_T^* &= \underset{f\in\mathcal{H}}{\arg\min} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \frac{P_T(x,y)}{P_S(x,y)} P_S(x,y) L(x,y,f) \\
&\approx \underset{f\in\mathcal{H}}{\arg\min} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \frac{P_T(x,y)}{P_S(x,y)} \tilde{P}_S(x,y) L(x,y,f) \\
&= \underset{f\in\mathcal{H}}{\arg\min} \frac{1}{|X_S|} \sum_{i=1}^{|X_S|} \frac{P_T\left(x_i^S, y_i^S\right)}{P_S\left(x_i^S, y_i^S\right)} L\left(x_i^S, y_i^S, f\right)
\end{aligned}$$

It is very clear here now from the above equations, the (homogeneous) domain adaptation can be seen as a learning process on the source domain labeled data while the loss of each source sample (x_i^S, y_i^S) is weighted by $\frac{P_T\left(x_i^S, y_i^S\right)}{P_S\left(x_i^S, y_i^S\right)}$. Intuitively, this weighting function $\frac{P_T(x,y)}{P_S(x,y)}$ describes how different it is between the two domains in the same position (x,y). Therefore, if we can find the set of optimal weights of source domain data points, the adaptation problem can be seen as solved. However, estimating joint probability $P_S(x,y)$ is a challenging task, not to mention $P_T(x,y)$ is even harder to obtain for we only have limited labels.

In order to simplify the analysis and have a closer look at the challenge of the Homogeneous DA problem, we can decompose $P(X,Y)$ to $P(X)P(Y|X)$, and then we get two ratios which are named **instance difference** and **labeling difference** separately (Jiang, 2008):

- **Instance Difference**[5]: The first probability ratio component is $\frac{P_T(x)}{P_S(x)}$. Intuitively, an instance that is in a dense region in the source domain but in a sparse region in the target domain should be down-weighted, because this instance is not so representative of the target domain. This case corresponds to $\frac{P_T(x)}{P_S(x)} < 1$, vice versa. Since Y is not involved in this probability ratio, ideally we can estimate this ratio using only unlabeled instances, that is, we can estimate this ratio in **unsupervised domain adaptation**.
- **Labeling Difference**[6]: The second probability ratio component is $\frac{P_T(y|x)}{P_S(y|x)}$, which indicates that the distribution of labels are how different in the two domains at x. Since this ratio is related to Y , without any labeled instances from the target domain, it is hard to estimate $P_T(y|x)$ and thus hard to estimate this ratio. Therefore, we can only estimate this ratio with relatively high accuracy in **supervised domain adaptation**.

This analysis naturally results in an **instance-weighting** based approach [6] for homogeneous domain adaptation:

$$f_T^* \approx \underset{f\in\mathcal{H}}{\arg\min} \sum_{i=1}^{|X_S|} \frac{P_T\left(x_i^S\right)}{P_S\left(x_i^S\right)} \frac{P_T\left(y_i^S | x_i^S\right)}{P_S\left(y_i^S | x_i^S\right)} L\left(x_i^S, y_i^S, f\right)$$

Note that instance weighting is only one view of formulate the domain adaptation problem. Another important view is that how we can create a new isomorphic feature space such that the two domains can share closer distributions. The key challenge is how to match different domain distributions effectively.

[5]For example, even with the same word vocabulary, the words in different domians like laptop review and movie review copora can be very different.

[6]For example, the same words can cause different labels in source and target domains. The sentence “it is pretty *long*.” can mean negative in a movie review, while being positive in a review about something else. xD

3 Maximum Mean Discrepancy: A key technique for measuring domain shift

We mention the difference between source and target data distributions a lot. How are we going to formally compute the distance between the two distributions given their samples? This problemis called **Two-Sample Tests**[7].

Imagine that you are given two groups of data points (drawn form p and q respectively) in a shared space $\mathcal{X}$, and you want to measure the difference between the probability distributions of them, what you will do? A straightforward way is to compute the mean-value point of each group and then use the distance between the two centrals as the distance between the two distributions. It is indeed a good intuition while $\mathcal{X}$ may not be a good place to do so. Imagine two Gaussians with the same mean but different variance, as shown in Figure 1.

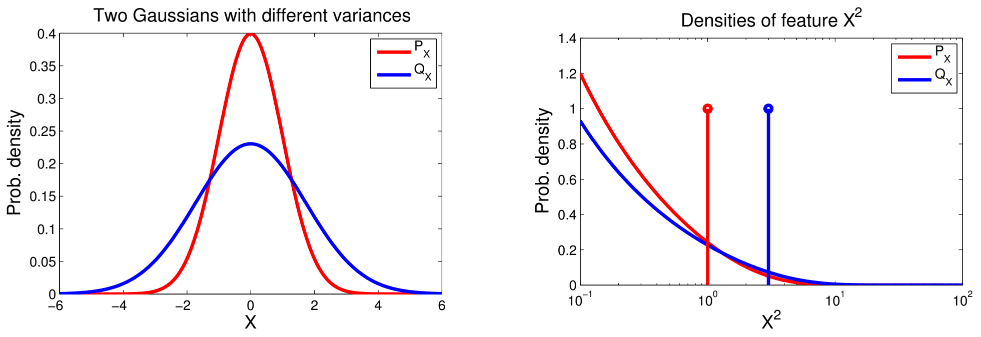


Figure 1: Intuition of MMD: using higher-dimensional feature spaces such that mean values are good distance indicators.

3.1 Definition

The **Maximum Mean Discrepancy** (MMD) is a measure of the difference between two probability distributions from their samples using such intuition. It is an effective criterion that compares distributions **without initially estimating their density functions**[8].

$$\mathrm{MMD}(p, q) := \sup_{f \in \mathcal{F}} \left(\mathbf{E}_p[f(x)] - \mathbf{E}_q[f(y)] \right)$$

where $\mathcal{F}$ is a class of functions $f : \mathcal{X} \to \mathbb{R}$. Simply put, we are using many random mappings that transform a data point $x \in \mathcal{X}$ to a real number $f(x) \in \mathbb{R}$, and then compute the mean-value (expectations) of the two groups of points $\mathbf{E}_p[f(x)]$ and $\mathbf{E}_q[f(y)]$ from the two distributions. Then, each function $f \in \mathcal{F}$ can give a distance values between the two distributions. Finally, we take the supremum of the set of such a infinite set of distance values as the final distance between p and q. That is why we call this distance as maximum mean discrepancy. With the sample data points (X, Y) in two distributions, we can get the empirical estimate of the MMD as:

$$\mathrm{MMD}(\mathcal{F}, X, Y) := \sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^{m} f(x_i) - \frac{1}{n} \sum_{j=1}^{n} f(y_j) \right),$$

where $m = |X|$ and $n = |Y|$.

3.2 MMD Computation in Reproducing Kernel Hilbert Space

Reproducing Kernel Hilbert Space (RKHS)[9] allows us to represent *a function as a vector*. We can then use inner-product in a universal RKHS H to understand the mapping $f(x) = \langle f, \phi(x) \rangle_{\mathcal{H}}$, where $\phi : \mathcal{X} \to \mathcal{H}$ (i.e. feature space map). Furthermore, we have $\mathbf{E}_p[f(x)]$ can be represented as $\langle f, \mathbf{E}_p[\phi(x)] \rangle_{\mathcal{H}}$. We call $\mathbf{E}_p[\phi(x)]$ as **kernel embedding of distributions**[10] or kernel mean.

[7]`http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/lecture5_distribEmbed_1.pdf`

[8]Many other well-known metrics such as Kullback-Leibler divergence need to first estimate the density of the two distributions from the samples before measuring their distance, which can be time-consuming and also have bias (nonzero when $p = q$).

[9]`http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/lecture4_introToRKHS.pdf`

[10]`https://en.wikipedia.org/wiki/Kernel_embedding_of_distributions`

Therefore, we can conclude that:

$$\begin{aligned}\mathrm{MMD}(p, q) &= \sup_{\|f\|_{\mathcal{H}} \leq 1} \mathbf{E}_p[f(x)] - \mathbf{E}_q[f(y)] = \sup_{\|f\|_{\mathcal{H}} \leq 1} \mathbf{E}_p\left[\langle f, \phi(x)\rangle_{\mathcal{H}}\right] - \mathbf{E}_q\left[\langle f, \phi(y)\rangle_{\mathcal{H}}\right] \\ &= \sup_{\|f\|_{\mathcal{H}} \leq 1} \langle \mu_p - \mu_q, f\rangle_{\mathcal{H}} = \|\mu_p - \mu_q\|_{\mathcal{H}}\end{aligned},$$

where $\mu_p = \mathbf{E}_p[\phi(x)]$ and $\mu_q = \mathbf{E}_q[\phi(y)]$. We take the square of MMD for symmetric property of the distance:

$$\begin{aligned}\mathrm{MMD}^2(p, q) &:= \langle \mu_p - \mu_q, \mu_p - \mu_q\rangle_{\mathcal{H}} = \langle \mu_p, \mu_p\rangle_{\mathcal{H}} + \langle \mu_q, \mu_q\rangle_{\mathcal{H}} - 2\langle \mu_p, \mu_q\rangle_{\mathcal{H}} \\ &= \mathbf{E}_p \langle \phi(x), \phi(x')\rangle_{\mathcal{H}} + \mathbf{E}_q \langle \phi(y), \phi(y')\rangle_{\mathcal{H}} - 2\mathbf{E}_{p,q}\langle \phi(x), \phi(y)\rangle_{\mathcal{H}}\end{aligned}$$

Using a Gaussian kernel function $k(x, x') = \exp\left(-\|x - x'\|^2 / \left(2\sigma^2\right)\right)$ and the samples, we finally have:

$$\mathrm{MMD}^2(X, Y) = \frac{1}{m(m-1)} \sum_{i \neq j}^{m} k(x_i, x_j) + \frac{1}{n(n-1)} \sum_{i \neq j}^{n} k(y_i, y_j) - \frac{2}{mn} \sum_{i,j=1}^{m,n} k(x_i, y_j).$$

Note that this computation incurs a complexity of $O(n^2)$, assuming m and n are similar.

3.3 MMD for Domain Adaptation

Now, with our old notations, we can compute the distance between the marginal distributions of the source and target domain as:

$$\mathrm{MMD}^2(X_S, X_T) = \frac{1}{m(m-1)} \sum_{i \neq j}^{m} k\left(x_i^S, x_j^S\right) + \frac{1}{n(n-1)} \sum_{i \neq j}^{n} k\left(x_i^T, x_j^T\right) - \frac{2}{mn} \sum_{i,j=1}^{m,n} k\left(x_i^S, x_j^T\right)$$

, where $m = |X_S|$ and $n = |X_T|$. In conventional domain adaptation and transfer learning, MMD has been used to reduce the distribution mismatch between the source and target domain. Pan et al. (2009) proposed a PCA-based model referred to as Transfer Component Analysis (TCA) that used MMD to induce a subspace where the data distributions in different domains are closed to each other. Long et al. (2013) presented a Transfer Sparse Coding (TSC) that utilizes MMD in the encoding stage to match the distributions of the sparse codes.

4 Domain Adaptation for Deep Neural Networks

In 2004, an important paper named "`How transferable are features in deep neural networks?`"[11] [16] was published at NIPS, which analyzed the transferability of features from each layer of a deep neural network such as `AlexNet`. They found that the generality or specificity of each layer can be very different: lower layer features are generally more transferable than higher layer features, which for the first time validates that deep neural networks are promising for domain adaptation and transfer learning.

However, it was still an open question that how we can **regularize** deep neural networks training such that the learned neural representations can be more transferable between domains and tasks. A typical kind of research (`DaNN`, `DDC`, `DAN`) is based on using **MMD** as the regularization terms added to loss functions, such that the network training process on source domain labeled data also aims to minimize the difference between source domain inputs and target domain inputs. Another recent popular direction ((`DANN`) is based on **adversarial learning**, which creates a new task to indirectly regularize the primary networks towards a domain-invariant space.

4.1 `DaNN`: Domain adaptive neural networks for object recognition (Ghifary et al., in Proc. of PRICAI 2014, w/ 70+ citations)

This is the very first work using MMD to regualize a neural network, and it is very simple. They use a two-layer feed-forward neural network, where the first layer is a feature extractor and the second is used as a classifier. In addition to the original classification loss L_{NNs}, they use MMD between the first-layer outputs (i.e. activations) of source/target domain inputs (H_S v.s. H_T). The key intuition here is to use the first-layer outputs as the new feature space and then minimize the distance between source and target inputs in this feature space.

$$L_{\mathrm{DaNN}} = L_{\mathrm{NNs}}(X_S, Y_S) + \gamma\, \mathrm{MMD}^2\left(H_S, H_T\right)$$

[11] Now it has 3,000+ citations as of Oct. 2019. There is a follow-up paper named "`How Transferable are Neural Networks in NLP Applications?`" [11], which conducts extensive transfer learning experiments for NLP models.

By using such a regularization, they aim to train the network parameters such that the supervised criterion is optimized and the hidden layer representations are encouraged to be invariant across different domains. The paper also illustrate how to compute the gradients and update the neural network with the MMD regularization. However, it was not very powerful since one layer is too shallow.

4.2 `DDC`: Deep domain confusion: Maximizing for domain invariance (Tzeng et al., in Arxiv 2014, w/ 490+ citations)

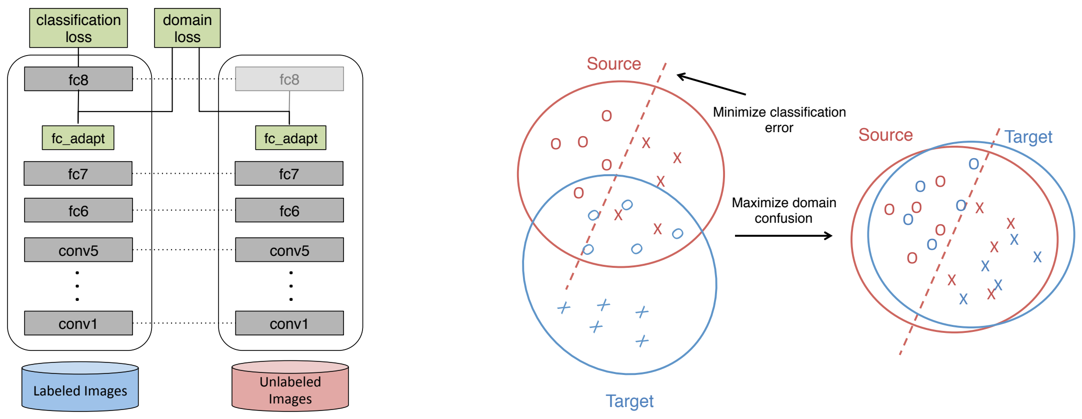


Figure 2: Deep domain confusion: Adding a single adaptation layer after `fc7` and compute MMD as the 'domain loss'.

This paper simply puts the MMD computation in a deeper convolutional neural network, AlexNet, and computes the MMD (so-called "domain confusion loss") as the additional term to the overall loss function similarly to `DaNN`. The contribution of this paper is to incorporate MMD to a larger, more popular pre-trained model (AlexNet) with a grid search on the position and the width of a new adaptation layer. Finally, they put the adaptation layer between *fc7* and *fc8*. Figure 2 shows the structure and the intuition of `DDC`.

4.3 `DAN`: Learning Transferable Features with Deep Adaptation Networks (Long et al., in Proc. of ICML 2015, w/ 960+ citations)

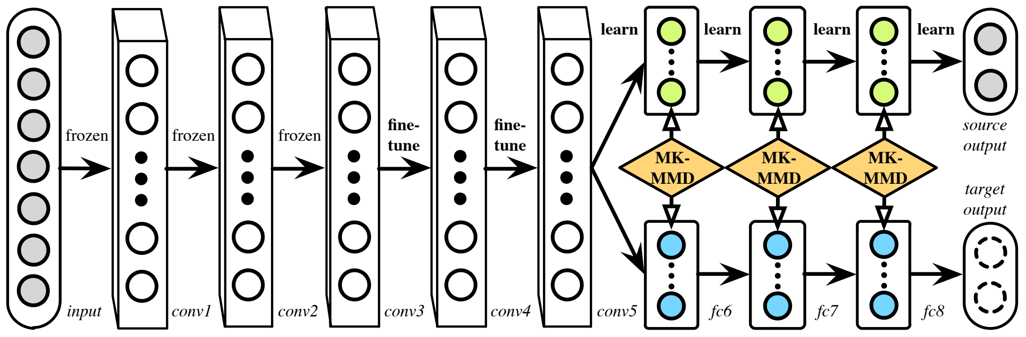


Figure 3: Deep Adaptation Networks using MK-MMD are used for regularizing domain shift in *fc6*-*fc8*.

DAN also follows the previous works of using MMD to regularize neural network training process and also mainly focus on AlexNet, which is a typical CNN and can be pre-trained on ImageNet. However, it utilizes a multi-kernel version of MMD (MK-MMD) to improve the MMD performance, and also put regularization on multiple layers (*fc6*-*fc8*) instead of only one layer (see Figure 3). The key technique MK-MMD is proposed by Gretton et al. (2012) and it is just a weighted combination of m different PSD kernels.

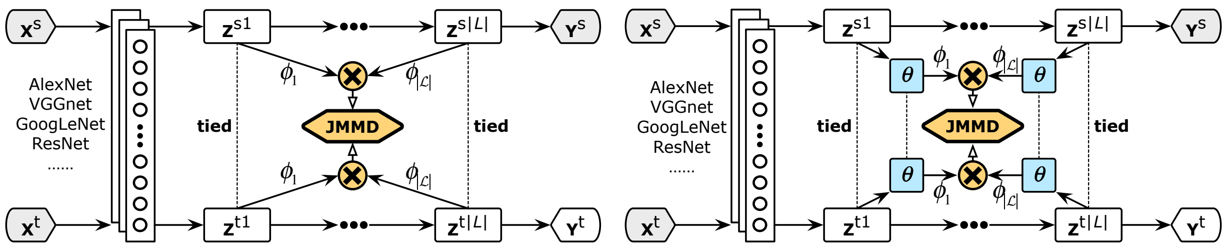
$$\mathcal{K} \triangleq \left\{ k = \sum_{u=1}^{m} \beta_u k_u : \sum_{u=1}^{m} \beta_u = 1, \beta_u \geqslant 0, \forall u \right\}$$

The total goal of DAN is to learn:

$$\min_{\Theta} \frac{1}{m} \sum_{i=1}^{m} L\left(\theta\left(x_i^S\right), y_i^S\right) + \lambda \sum_{l=\texttt{fc6}}^{\texttt{fc8}} \mathrm{MMD}^2\left(H_S^l, H_T^l\right)$$

where we use H_S^l to represent the learned hidden representations of X_S at the l-th layer, similarly for H_T^l. Another notable thing of this paper is that they adopt a **linear-time** algorithm for getting the unbiased estimate [4].

4.4 JAN: Deep transfer learning with joint adaptation networks (Long et al., in Proc. of ICML 2017, w/ 300+ citations)



(a) Joint Adaptation Network (JAN) (b) Adversarial Joint Adaptation Network (JAN-A)

Figure 4: The architectures of Joint Adaptation Network (JAN) (a) and its adversarial version (JAN-A) (b).

JAN further considers jointly regularizing multiple hidden representations together with JMMD distance, which is detailed in the paper. One interesting thing is that the author also propose an adervsial version of JAN based on JMMD as the learning objective:

$$\min_{f\in\mathcal{H}} \max_{\theta} \frac{1}{m} \sum_{i=1}^{m} L\left(f\left(x_i^s\right), y_i^s\right) + \lambda\, \mathrm{JMMD}_{\mathcal{L}}(P_S, P_T; \theta).$$

This adversarial training idea is directly following the DANN, which learns to maximize the distance metric performance while minimize classification error at the same time.

4.5 DANN: Unsupervised Domain Adaptation by Backpropagation (Ganin and Lempitsky, in Proc. of ICML 2015, w/ 1100+ citations)

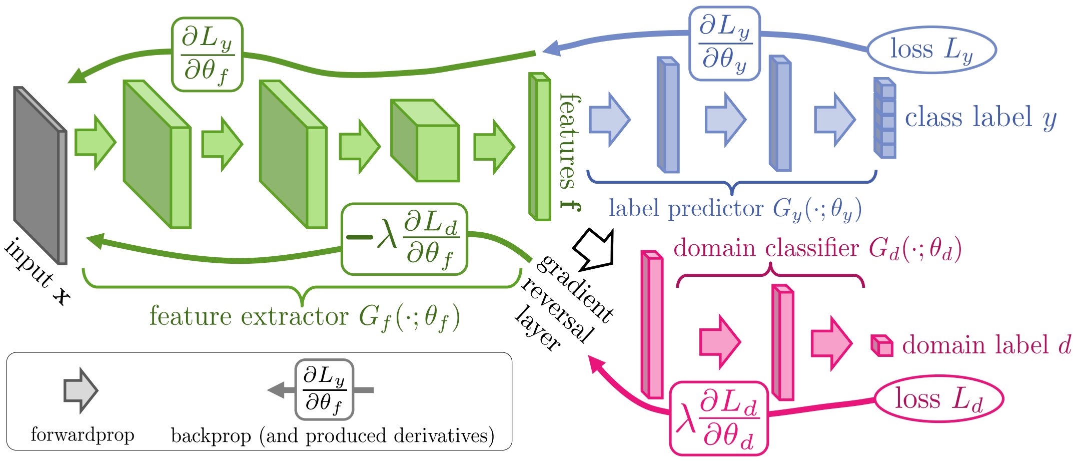


Figure 5: DANN consists of: 1) a feature extract (green), 2) a label predictor (blue), and 3) a domain classifier (red).

We now present the very first work using adversarial training as the main idea for domain adaptation, namely DANN. As Figure 5 shows, DANN has three components:

- **feature extractor**: this network $G_f(x;\theta_f) = \mathbf{f}$ is usually a feed-forward network or CNN for extracting abstract features from raw inputs with multiple layers. It produces a feature vector $\mathbf{f}$ for each input. For illustrative purpose, we use a single layer network:

$$G_f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \operatorname{sigm}(\mathbf{W}\mathbf{x} + \mathbf{b}).$$

- **label predictor**: This is a normal classification network, denoted as $G_y(\mathbf{f};\theta_y) = y$, where $y \in \mathcal{Y}$, for predicting the label with the extracted feature vector. For illustrative purpose, we use a simple logistic regression classifier:

$$G_y\left(G_f(\mathbf{x}); \mathbf{V}, \mathbf{c}\right) = \operatorname{softmax}\left(\mathbf{V}G_f(\mathbf{x}) + \mathbf{c}\right).$$

  The associated loss function for classification error is thus:

$$\mathcal{L}_y\left(G_y\left(G_f\left(\mathbf{x}_i\right)\right), y_i\right) = \log \frac{1}{G_y\left(G_f(\mathbf{x})\right)_{y_i}}$$

- **domain classifier**: This is a novel contribution. It also takes the $\mathbf{f}$ as the input, while the label of each sample is based on whether it is from the source domain or the target domain. Thus, we denote it as $G_d(\mathbf{f};\theta_d) = d$, where $d \in \{S := 1, T := 0\}$. Here we use another single layer network (with different parameters) as an example:

$$G_d\left(G_f(\mathbf{x}); \mathbf{u}, z\right) = \operatorname{sigm}\left(\mathbf{u}^\top G_f(\mathbf{x}) + z\right).$$

  Accordingly, the loss function for domain classification is:

$$\mathcal{L}_d\left(G_d\left(G_f\left(\mathbf{x}_i\right)\right), d_i\right) = d_i \log \frac{1}{G_d\left(G_f\left(\mathbf{x}_i\right)\right)} + (1-d_i)\log\frac{1}{1 - G_d\left(G_f\left(\mathbf{x}_i\right)\right)}.$$

When learning the whole architecture, we would like to minimize the total loss function L:

$$E(\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c},\mathbf{u},z) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}_y^i(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}) - \lambda\left(\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z) + \frac{1}{n}\sum_{i=m+1}^{m+n}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z)\right).$$

Note that $m = |X_S|$ and $n = |X_T|$ and we concatenate the two sets in to a new set X with the ordering that source samples are all before target samples. By replacing the example parameters back to the general case, we have:

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

by finding that:

$$\left(\hat{\theta}_f,\hat{\theta}_y\right) = \underset{\theta_f,\theta_y}{\operatorname{argmin}}\, E\left(\theta_f,\theta_y,\hat{\theta}_d\right)$$
$$\hat{\theta}_d = \underset{\theta_d}{\operatorname{argmax}}\, E\left(\hat{\theta}_f,\hat{\theta}_y,\theta_d\right),$$

where $\mathcal{L}_y^i\left(\theta_f,\theta_y\right) = \mathcal{L}_y\left(G_y\left(G_f\left(x_i;\theta_f\right);\theta_y\right),y_i\right)$ and $\mathcal{L}_d^i\left(\theta_f,\theta_d\right) = \mathcal{L}_d\left(G_d\left(G_f\left(x_i;\theta_f\right);\theta_d\right),d_i\right)$. We then use a gradient reversal layer for such an adversarial learning objective:

$$\begin{aligned}
\theta_f &\longleftarrow \theta_f - \mu\left(\frac{\partial\mathcal{L}_y^i}{\partial\theta_f} - \lambda\frac{\partial\mathcal{L}_d^i}{\partial\theta_f}\right)\\
\theta_y &\longleftarrow \theta_y - \mu\frac{\partial\mathcal{L}_y^i}{\partial\theta_y}\\
\theta_d &\longleftarrow \theta_d - \mu\lambda\frac{\partial\mathcal{L}_d^i}{\partial\theta_d}
\end{aligned}$$

where μ is the learning rate. Simply put, `DANN` is maximizing the performance of domain classifier such that it can better measure the distance between the two domains. However, it use gradient reversal, such that it regularizes the feature extractor NOT to go the direction that leads better domain classification accuracy. Therefore, the feature extractor has to transform the two domain inputs with similar representation distributions. With the label predictor as the primary supervision, such learned features can still do very well for the task $\mathcal{T}$.

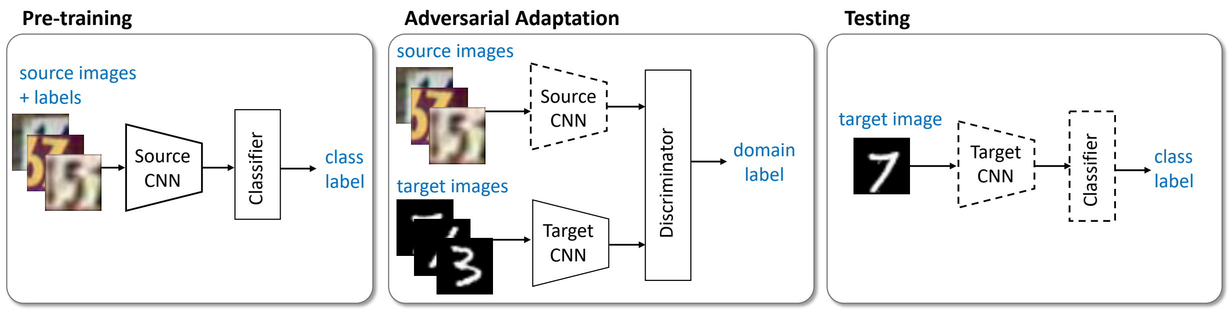


Figure 6: ADDA consists of three stages: 1) **pre-training** a source domain classifier; 2) learning a **target encoder** (another CNN) such that the discriminator cannot distinguish whether the example is from which domain, 3) and finally **testing** target inputs with the learned target encoder.

4.6 ADDA: Adversarial discriminative domain adaptation (Tzeng et al., in Proc. of CVPR 2017, w/ 720+ citations)

Figure 6 shows a more recent framework for domain adaptation, which is also based on adversarial training:

1. Pre-training a source encoder $M_S(\cdot)$ and minimize the error of the classifier $C(\cdot)$ in source domain:

$$\min_{M_S,C} \mathcal{L}_{\text{cls}}\left(X_S, Y_S\right) = -\mathbb{E}_{\left(x_i^S, y_i^S\right)\sim(X_S,Y_S)} \sum_{k=1}^{K} \mathbb{1}_{\left[k=y_i^S\right]} \log C\left(M_S\left(x_i^S\right)\right)$$

2. Learning a discriminator $D(\cdot)$ such that it cannot distinguish source and target samples. Thus, the learning has two optimization objectives as follows. (Note that the target encoder M_T is initialized by the source encoder M_S, and M_S is fixed in this phase.)
   - Minimizing the **adversarial discriminator loss**:

$$\min_{D} \mathcal{L}_{\text{adv}_D}\left(X_S, X_T, M_S, M_T\right) = -\mathbb{E}_{x_i^S\sim X_S}\left[\log D\left(M_s\left(x_i^S\right)\right)\right] - \mathbb{E}_{x_i^T\sim X_T}\left[\log\left(1 - D\left(M_t\left(x_i^T\right)\right)\right)\right]$$

   - Minimizing the **adversarial mapping loss**:

$$\min_{M_T} \mathcal{L}_{\text{adv}_M}\left(X_S, X_T, D\right) = -\mathbb{E}_{x_i^T\sim X_T}\left[\log D\left(M_T\left(x_i^T\right)\right)\right]$$

Note that we use independent mappings for source and target and learn only M_T adversarially. This mimics the GAN setting, where the real distribution remains fixed, and the generating distribution is learned to match it. In our case of domain adaptation, the source domain input distribution is also fixed after the first stage (M_S is fixed), while we learn a M_T to match it. In DANN, the gradient reversal layer is actually directly optimizing the loss by $\mathcal{L}_{\text{adv}_M} = -\mathcal{L}_{\text{adv}_D}$.

5 Conclusion

This introduction focuses on how we can regularize deep neural networks such that only learning from labeled source domains and unlabeled target domains can maintain a good generalization performance. Recent methods either use MMD as additional regularization terms to control the learn representations are domain-invariant, or use adversarial learning method to simultaneously learn 1) to find a good domain-distance measure, 2) to find a domain-invariant representation such that learned classifiers can be shared. These works usually only conduct experiments on computer vision datasets or using bag of words as features for textual datasets. The potential of such neural domain adaptation methods are surprisingly underexplored for same reasons. A key challenge of applying these methods in NLP is the lack of large pre-trained deep models back then. NLP models were usually based on RNN structure instead CNNs that are naturally easy to control transferring on which parts. In the era of BERT, we think it is very meaningful and necessary to examine such domain adaptation methods in NLP applications as well.

References

[1] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1180–1189, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/ganin15.html`.

[2] Muhammad Ghifary, W. Bastiaan Kleijn, and Mengjie Zhang. Domain adaptive neural networks for object recognition. In *PRICAI*, 2014.

[3] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, March 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2188385.2188410`.

[4] Arthur Gretton, Bharath K. Sriperumbudur, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, and Kenji Fukumizu. Optimal kernel choice for large-scale two-sample tests. In *NIPS*, 2012.

[5] Jing Jiang. Domain adaptation in natural language processing. Technical report, 2008.

[6] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 264–271, 2007.

[7] Bill Yuchen Lin and Wei Lu. Neural adaptation layers for cross-domain named entity recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2012–2022, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1226.

[8] Mingsheng Long, Guiguang Ding, Jianmin Wang, Jia-Guang Sun, Yuchen Guo, and Philip S. Yu. Transfer sparse coding for robust image representation. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 407–414, 2013.

[9] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In *ICML*, 2015.

[10] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2208–2217. JMLR. org, 2017.

[11] Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in nlp applications? In *EMNLP*, 2016.

[12] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[13] Sinno Jialin Pan, Ivor Wai-Hung Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22:199–210, 2009.

[14] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *ArXiv*, abs/1412.3474, 2014.

[15] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176, 2017.

[16] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.