d-SNE: Domain Adaptation using Stochastic Neighborhood Embedding
(CSC 586B Course Project: Creating a PyTorch Implementation)

Joshua Newton
V00826800
University of Victoria
jnewt@uvic.ca

Abstract

Domain adaptation is a subset of transfer learning where both the source and target datasets have the same labels. d-SNE, a new domain adaptation approach, trains neural networks by using a novel loss term that compares the outputs for paired source/target batches of training images. Its authors claim state of the art performance through experiments conducted on Digits, Office31, and VisDA-C 2017 datasets. Publicly available d-SNE code written using the MXNet framework, however, is only partially functional and cannot fully replicate these claims. This project introduces a PyTorch port of the d-SNE approach. It successfully replicates the same partial functionality found in the MXNet implementation: a single experiment to adapt MNIST to MNIST-M using the LeNetPlus architecture. Replicating the publication's full claims, however, was not achieved within the scope of this project. d-SNE's semi-supervised extension and experiments using other architectures and datasets are left as future work. As well, initial findings suggest that the default configuration requires tweaking if the claims of the publication are to be replicated.

1. Introduction

Deep neural networks typically require large amounts of training data in order to achieve acceptable performance for classification problems. It is often unreasonable, though, to collect any more than a small amount of original, problem-specific data. (Collecting and labeling image data, for example, is often a time-consuming and tedious process.) Despite this, there are sources of preexisting data which can be used to supplement new, problem-specific data. Previously curated datasets containing similar content can be often be found online. Search engines can also act as a source of data when combined with web scraping. Methods for reapplying knowledge from existing sources such as these fall under the domain of transfer learning.

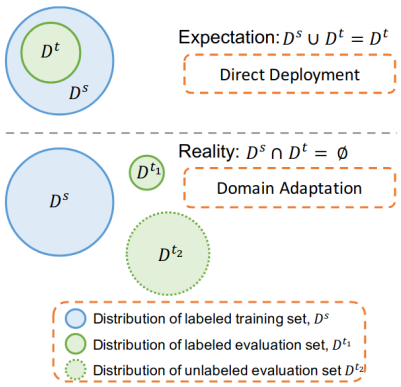

Figure 1. Domain shift: expectation vs. reality.

d-SNE, the subject of this report, addresses a special case of transfer learning called domain adaptation. In domain adaptation, the class labels are identical for both the reused dataset (or source dataset) and the problem-specific dataset (or target dataset). Having a shared label space, however, does not guarantee that the source dataset will be directly applicable to the target domain. Images with identical labels can still contain vastly different visual features. This variation is a challenge known as domain shift. A visualization of this problem is provided in Figure 1 As a domain adaptation approach, d-SNE aims to address domain shift by accounting for these visual distinctions.

Even more narrowly, d-SNE has been designed to excel in one-shot and few-shot learning settings. These are extreme cases for when only a minimal amount of new data can be collected: a single image for one-shot learning, and 2-10 images for few-shot learning. These conditions are common for problems such as facial recognition. Finally, d-SNE contains a semi-supervised extension which allows for the reuse of unlabeled data, too. Thus, d-SNE's motivations cover several different problem domains within the category of domain adaptation.

2. Contributions of the original paper

The primary contribution of the d-SNE paper is the development of d-SNE, an end-to-end neural network training procedure for domain adaptation. This procedure is based off of siamese (or twin) networks, a type of architecture commonly found in one-shot learning problem domains such as facial recognition. In the general case, two networks are trained simultaneously, using one for each of the source and target datasets. However, a single shared network can instead be used if the source and target domain images are of the same dimensions.

For the training itself, d-SNE uses a novel loss formulation that combines typical cross-entropy loss with a new term that compares the output for paired batches of source and target images. This comparison-based loss term helps to ensure that the weights of the network produce similar outputs for both source and target images, thus adapting the source dataset to the target domain. The loss term was inspired by the dimensionality reduction technique Stochastic Neighbourhood Embedding. The value of this paper's contribution lies in the reapplication of this technique to the problem domain of domain adaptation. Finally, in addition to the base functionality of the method, d-SNE also contains a semi-supervised extension using a technique similar to the Mean-Teacher network proposed by Tarvainen et al []. This extension allows the use of unlabeled data in addition to the source and target datasets, making d-SNE a versatile domain adaptation approach.

In addition to the approach itself, the authors of the d-SNE paper conduct a number of experiments to validate the efficacy of the approach. These experiments largely mirror experiments conducted by other state of the art domain adaptation papers such as CCSA and FADA. A convolutional neural network is chosen, and a pair of datasets (source and target, with shared class labels) are selected from a set of benchmark datasets. These include digits datasets (MNIST, MNIST-M, USPS, SVHN), Office31 datasets (webcam, Amazon, DSLR), and VisDA-C 2017 datasets (real, synthetic). Also, a fixed amount of target images per class are chosen for training, corresponding to one-shot (1 image) and few-shot (2-10 images) learning. A total of 16 experiments are conducted within the scope of the paper for different architecture and dataset configurations. d-SNE's accuracy ranks first in 14 of these experiments. This evaluation demonstrates that d-SNE is not just a versatile approach, but an effective one as well.

The last contribution of the paper is an implementation for d-SNE provided alongside the publication. The available code is written using Python and the MXNet deep learning framework. However, the existing codebase has significant flaws which make it impossible to replicate the stated experiments. These flaws will be described further in Section 4 of this report. Despite this non-functionality, the codebase nonetheless provides a foundation for building a new (or improved) implementation.

3. Contributions of this project

The first aspect of my work on this project was simply to examine the existing MXNet implementation to see which experiments, if any, could be replicated using the codebase as-is. Unfortunately, there were a number of issues which prevented tests from functioning. These issues range from simple runtime errors to deeper issues relating to the choice of network architecture. These issues are documented in a number of GitHub Issues placed in the implementation's repository. They will also be discussed further in Section 4. It was at this time that I decided to spend my resources reimplementing the approach from scratch in PyTorch, rather than continue to work with the broken MXNet implementation.

Of note, however: midway through my PyTorch work, the author of the d-SNE paper took the time to refactor the MXNet codebase to a working state. This second version contains only a single working experiment: adapting the MNIST source dataset to the MNIST-M target domain, using 10 target images per class to train the network. A config file is provided for this experimenmt which specifies the network architecture, "LeNetPlus", as well as a number of hyperparamters. This working experiment can serve as an initial attempt to verify the results of the publication. Running this experiment and examining the loss and accuracy evolution curves indicate several issues relating to network initialization and hyperparameter selection. From the beginning, the training procedure requires many iterations to begin seeing any improvement. When improvement does occur, it happens rapidly until the point of severe overfitting. Training accuracy reaches 99%, while test accuracy peaks at 78% after roughly 18000 iterations with a batch size 256. This test accuracy is concerning, as the publication states that 88% was achieved for this experiment. Despite not matching the claims made by the paper, it can still act as a baseline for verifying my separate PyTorch implementation.

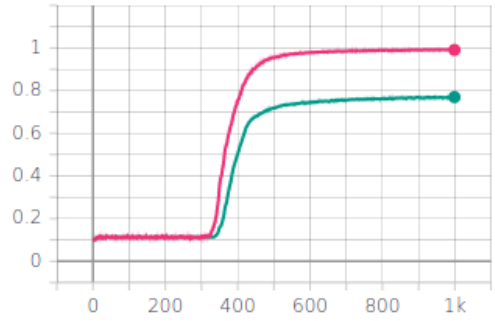

Figure 2. Accuracy evolution curves for training set (pink) and validation set (green). The initial iterations with no improvement demonstrate potential initialization concerns, while the gap between training and validation curves demonstrate hyperparameter configuration concerns.

In my codebase, I have first implemented a number of boilerplate bits of functionality for controlling the training/testing procedure. These include loading a configuration from a file, setting up the training loop, performing validation/testing, creating an optimizer, exporting and resuming checkpoints, and logging to both Tensorboard and to a file. For the code specific to d-SNE, I have implemented data loading for the two datasets used in the MNIST to MNIST-M experiment. In addition to the singular datasets, I have created a custom dataloader for creating paired batches of source/target images as required by d-SNE's methods. Also, I have implemented the custom "LenetPlus" architecture contained within the refactored version of the MXNet implementation. Finally, I have implemented the custom loss function that compares the outputs from both the source and target input batches of images. Together, these functions represent the full end-to-end training procedure of d-SNE.

When running the MNIST to MNIST-M experiment in my PyTorch implementation using the same configuration settings as the MXNet implementation, similar results are found. As before, the training and loss curves stall with no improvement for many iterations. Then, rapid improvement occurs to the point of overfitting. 78% accuracy is achieved, with similar starting and ending loss values. The accuracy evolution for this experiment can be viewed in Figure 2. This leads me to believe that my implementation is faithful to the MXNet implementation.

4. Discussion and future directions

As much of my work was implementation-based, I do not have many findings to report beyond what what described in Section 3. I was not able to replicate the performance claimed by by the publication with either the MXNet implementation or the PyTorch implementation. However, my PyTorch implementation performs similarly to the available MXNet implementation, which is a start.

There are many areas which could benefit from further work. Only the LeNetPlus architecture has been implemented to evaluate d-SNE. However, VGG-16 and ResNet101 have been used by the state of the art, including CCSA and FADA. To verify the performance of the method in comparison to these other methods, these networks would need to be implemented. As well, the semi-supervised extension has not been implemented. Finally, functions would need to be created to pack remaining datasets into containers for data loading. These include other digits datasets (USPS, SVHN), the Office31 datasets (Webcam, Amazon, DSLR), and the VisDA-C 2017 datasets (real, synthetic). Of note, this functionality is not yet implemented in a working fashion in the MXNet implementation either, so there currently exists no public means of validating the claims made by the publication.

Once the implementation is fully complete, other experiments could be performed. The first experiment to run would be MNIST to USPS domain adaptation in one-shot and few-shot settings. The next set of experiments is similar to the MNIST to MNIST-M experiment performed in this report, but for other pairs of digits, including MNIST to SVHN, and USPS to MNIST. Finally, other experiments can be performed with various Office31 pairs of datasets, and the VisDA-C 2017 pair of datasets.

Lastly, it could be beneficial to further explore hyperparameter tuning. The default set of hyperparameters provided with the March 2020 MXNet implementation appear to have significant flaws. The loss/evolution curves are flat during the initial training iterations, which point to there being potentially poor initialization. As well, overfitting occurs before accuracy can reach the performance stated by the publication for the MNIST -¿ MNIST-M experiment. As I achieved only 78% compared to the 87% stated by the publication, there is clearly room for improvement in configuring the training procedure.

References