Learning Sampling Policies for Domain Adaptation

Yash Patel⋆, Kashyap Chitta⋆, and Bhavan Jasani⋆

The Robotics Institute, Carnegie Mellon University
{yashp, kchitta, bjasani}@andrew.cmu.edu

Abstract. We address the problem of semi-supervised domain adaptation of classification algorithms through deep Q-learning. The core idea is to consider the predictions of a source domain network on target domain data as noisy labels, and learn a policy to sample from this data so as to maximize classification accuracy on a small annotated reward partition of the target domain. Our experiments show that learned sampling policies construct labeled sets that improve accuracies of visual classifiers over baselines.

Keywords: Domain Adaptation, Active Learning, Deep Q-learning.

1 Introduction

Dataset bias [1] is a well-known drawback of supervised approaches to visual recognition tasks. In general, the success of supervised recognition models, both of the traditional and deep learning varieties, is restricted to data from the domain it was trained on [2]. The common approach to handle this is fairly straightforward: pre-trained deep models perform well on new domains when they are *fine-tuned* with a sufficient amount of data from the new distribution. However, data for fine-tuning needs to be annotated. In many situations, labeling enough data for this approach to be effective is still prohibitively expensive.

Recent work on domain adaptation address this problem by aligning the features extracted from the network across the source and target domains, without any labeled target samples. The alignment typically involves minimizing some measure of distance between the source and target feature distributions, such as correlation distance [3], maximum mean discrepancy [4], or adversarial discriminator accuracy [5,6,2].

In this work, we explore the semi-supervised domain adaptation problem. We assume that we can collect data in the target domain, as well as annotate a small fraction of it, and we have a fixed *budget* for annotation. This has been extensively studied under the field of active learning [7,8], where the goal is to obtain better predictive models than those trained on equal amounts of i.i.d. data by deciding which examples to annotate from a large unlabeled dataset. However, active learning methods are inherently designed for a target domain

⋆ Equal Contribution

directly, and do not make use of the extensive amount of annotated data we have in the source domain.

We propose a reinforcement learning based formulation of the semi-supervised domain adaptation problem. In active learning, we need to choose a subset of the data to annotate and train from. We hypothesize that we could better use our annotation budget if we label a 'reward partition', used to generate rewards for a deep Q-network. Knowledge from the source domain could be coupled with this Q-agent to potentially give us a large quantity of well-labeled data in the target domain, which could not be achieved independently through unsupervised domain adaptation or active learning.

Inspired by a similar approach for action recognition [9], we aim to use our Q-network to learn a policy for sampling from noisily labeled data in the target domain. A classifier trained on the source domain is used to generate these noisy annotations for the entire target dataset. The agent is rewarded for sampling data from the target domain, that when used to train a new classifier, leads to high accuracies on the annotated reward partition.

We evaluate our approach on the Office-31 dataset, a widely accepted benchmark for testing real-world visual domain adaption methods [10], comparing our learned policies to baselines, and state-of-the-art unsupervised domain adaptation methods.

2 Method

In this section, we describe the proposed method for semi-supervised domain adaptation for a n-way classification problem. The training data consists of images from two different domains, we will refer to these domains as $D_s = \{(x_s^i, y_s^i)\}_{i=1}^{N_s}$ (source domain) and $D_t = \{(x_t^i)\}_{i=1}^{N_t}$ (target domain). An overview of entire method is shown in Fig 1. It consists of the following components:

- Deep convolutional neural network based **source classifier**, trained for classification of the source domain D_s images into n object categories.
- Binary Support Vector Machine (SVM) used as a **domain discriminator**, to help select a held out subset of target domain samples S_{rew} for generating rewards, and initialize a training set S_{pos} for the multi-class SVM.
- Multi-class SVM based **target classifier**, for classification of the target domain D_t images into n object categories.
- Deep **Q-agent** sampling an image from the target domain D_t every iteration, to be added to S_{pos}.

Source Classifier. For image feature representations, we make use of a ResNet-50 [11] architecture pretrained on ImageNet [12]. We choose this model for comparison to previous work [13]. In order to obtain fine-grained representations, we first fine-tune the network on source domain D_s in a supervised setting. We denote this source classifier as C_{src}.

Domain Discriminator. We make a reward set S_{rew}, that consists of sampled target domain images that is used to evaluate the performance of classifier and in-turn used for computing rewards for the Q-agent. In order to pick an appropriate

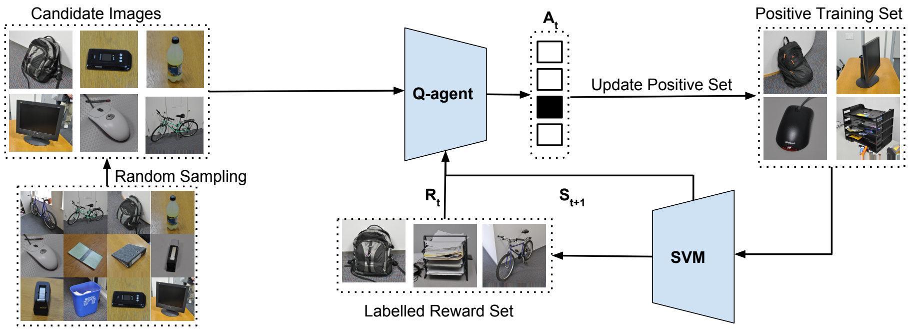

Fig. 1. Overall Method

set of images for S_{rew}, we train a binary SVM on image representations to classify the images as source domain or target domain. We denote this classifier as C_{dom}. The reward set S_{rew} is then stochastically sampled from target domain based on the sample distances from the separating hyperplane of C_{dom}, with the sampling weight of each sample $w_i \propto d_i$. The idea behind this initialization is that the samples further away from the domain classification hyperplane are more confusing and different from source domain samples and thus make a good evaluation set [14]. Note that we use our budget for ground truth annotations to get the true labels for images in S_{rew}.

Target Classifier. The n-way classifier C_{tar} is trained on a subset of target domain images governed by the Q-agent, which we represent as S_{pos}. The target classification labels are set to the predictions from source domain classifier C_{src} on the images in S_{pos}. At each iteration of training, the action taken by the Q-agent involves updating S_{pos}, and this updated training set is used to train C_{tar} again. For our setup, we make use of multi-class SVM for C_{tar} since the number of images in target domain D_t is limited, and we need relatively quick convergence since it is repeatedly retrained every episode.

For the initialization of S_{pos}, we use C_{dom} again, but now sample l data points from D_t with weights directly proportional to the sample distances from the classification hyperplane $w_i \propto \frac{1}{d_i}$. This follows the idea that the samples confusing the domain classifier are quite similar visually to those in the source domain, making the source classifier's predictions on these more reliable.

2.1 Q-agent

The objective of the Q-agent is to predict the appropriate set of samples which maximize the performance of target domain classifier C_{tar} on the reward set S_{rew}. At any given timestep t, the Q-agent observes the current state s_t, selects an action a_t from its discrete action space, and receives a reward r_t as well as a next state observation s_{t+1}.

Actions. To have a fixed length action space for every iteration, we randomly sample n_{cand} samples in D_t, with replacement, giving new set of examples each iteration we call the candidate set S_{cand}. The Q-agent then chooses one of $n_{cand}+1$ actions, which are either to:

- pick a sample from S_{cand} to be added to S_{pos}, in which case that sample is flagged and no longer chosen for S_{cand} for the rest of the episode, or
- pick none of the samples in the current S_{cand} and move to the next iteration.

State Representation. We formulate the states as a concatenation of two vectors, one dependent on the positive set and the other on the current candidate set. To get the first part of the state representation which shows the current distribution of examples in the positive set, we use a histogram of classifier confidences, obtained by using C_{tar} on the current S_{pos}, We threshold the distribution over classes output by C_{tar} into discrete bins of equal width. This component of the state representation has a dimensionality of $n \times n_{bin}$ (where n is the number of classes).

The second part of the state representation gives the agent a concise summary of the choices it has: for every image in S_{cand}, we obtain the distribution of classifier confidences for each class as a confidence vector for that sample. The concatenation of all these vectors, each of which corresponds to a specific action that the Q-agent can take, has a dimensionality of $n \times n_{cand}$.

The entire state vector, which is the input to the Q-network, is a flattened concatenation of these two parts.

Rewards. The reward for the agent at any time stamp r_t is determined by the relative change in performance of target domain classifier C_{tar} from the previous time stamp.

Q-Network. In the standard formulation of Q-learning, we use a function approximator with some weights w to estimate the Q-value of a state-action pair (s, a). We use a Dueling Deep Q-Network (DDQN) [15] as our function approximator. We define the advantage function relating the value and Q functions:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{1}$$

We implement the DDQN with two sequences (or streams) of fully connected layers. We subtract the average output over all actions from the advantage stream:

$$Q(s,a;\alpha,\beta) = V(s;\beta) + A(s,a;\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\alpha) \tag{2}$$

where α are the parameters of the advantage stream, β are the parameters of the value stream, and the weights w we intend to optimize is the combination of these two sets of parameters.

Stabilization. We further stabilize the training by maintaining two Q-networks [16]: an online network with weights w, and a target network with weights w^-. The weights w are updated using the following gradients:

$$w := w + \eta\left(r + \gamma \max_{a' \in A} Q_{w^-}(s',a') - Q_w(s,a)\right)\nabla_w Q_w(s,a). \tag{3}$$

Where Q_{w^-} is generated by the target network, and η is our learning rate. Every few iterations, the target network weights are set to be equal to the online Q-network, and are kept frozen until the next such assignment.

We additionally found that adding weight decay as a regularizer to the objective and using a sigmoid activation on the final predicted Q values significantly improved the optimization. Since the Q values are expectations of cumulative rewards under an optimal policy, the sigmoid activation bounds these values to the range [0,1]. This is a valid restriction based on our reward structure, as the maximum possible reward of an optimal policy is 1.

3 Experiments

We evaluate our algorithm on Office-31 dataset [10]. It contains images from 3 domains, **A**mazon, **W**ebcam and **D**SLR. Within each domain, images belong to 31 classes of everyday objects, with a fairly even class distribution. The dataset is imbalanced across domains, with 2,817 images in **A**, 795 images in **W**, and 498 images in **D**. With 3 domains, we have 6 possible transfer tasks, which address various forms of domain shift including resolution, lighting, viewpoint, background and dataset size.

3.1 Experimental Setup

The standard protocol for evaluation in unsupervised adaptation involves using all images in the source domain (with labels) and target domain (without labels) for training, and reporting performances on the entire target domain. We compare our results to other work based on the same feature extraction backbone (ResNet-50), by evaluating the final learned policy with the entire target domain dataset. The key difference between existing approaches and ours is that we use $k = 3$ labels per class from the target domain in addition to the remaining unlabeled data during training.

3.2 Implementation Details

Source Classifier. For each transfer task, we initially fine-tune our backbone network on the source domain for 30,000 iterations. We use SGD with an exponentially decaying learning rate starting at 0.003 and a batch size of 16. The 2048-dimensional pool5 feature vectors are used as representations for the images by the other components of our algorithm.
SVMs. For the domain discriminator C_{dom}, we train a binary SVM with a linear kernel using representations from the entire source domain as one class and target domain as the other. The target classifier C_{tar} is a multi-class linear kernel SVM trained in a one-vs-all manner every iteration. Both of these are implemented with the default parameters using the liblinear library [17]. We initialize S_{pos} with $l = 100$ samples for all 6 transfer tasks.

Q-agent. For our state representation, we set n_{bin} of the histogram to 10 and n_{cand} to 20. This leads a state space of dimensionality 930 and action space of size 21. For the value stream of our DDQN, we use a single hidden layer with as many units as the dimensionality of the state. The advantage stream consists of two hidden layers of 512 units each.

We use the Adam optimizer [18] with a learning rate of 0.001. We use an ϵ-greedy linear exploration policy, reducing the value of ϵ from 1 to 0 over the 2,000 iterations of training, and train for a total of 20,000 iterations for each transfer task. We assign the weights of our online network to the target network every 10 iterations. Our models are implemented on Keras with a TensorFlow backend [19].

3.3 Results

We report our results in Table 1. As a baseline, we use the classifier trained on source domain and evaluate on target domain (without any domain adaptation) for all 6 transfer tasks. We additionally compare to the best reported results for unsupervised domain adaptation on this dataset [13].

We observe that the learned policies do better than baselines, but fail to close the gap towards existing state-of-the-art unsupervised adaptation methods.

Table 1. Comparison with state of the art

Method	A $\Rightarrow$ D	A $\Rightarrow$ W	D $\Rightarrow$ A	D $\Rightarrow$ W	W $\Rightarrow$ A	W $\Rightarrow$ D
Baseline (ours)	76.9%	71.4%	58.6%	91.7%	57.4%	96.8%
Ganin et al. [5]	79.7%	82.0%	68.2%	96.9%	67.4%	99.1%
Kang et al. [13]	88.8%	86.8%	74.3%	99.3%	73.9%	100%
Ours (semi-supervised)	86.1%	83.5%	64.6%	93.1%	60.8%	98.2%

4 Conclusion

We present a reinforcement learning based approach to learn sampling policies for the purpose of domain adaptation. Our method learns to select samples for training from the target domain that maximize performance on a reward set, and in-turn improve the overall classification accuracy in the target domain.

Unlike the existing state-of-the-art methods, we make use of fixed representations for both the source and target domain samples which hurts the performance of our method. In order to fix this, we plan to work on the integration of a *labeler* into the Q-agent which will be jointly optimized with the current *sampler* to tune better representations and less noisy labels for target domain. Another idea is to learn sampling policies using the representations obtained after performing unsupervised domain adaptation through existing feature alignment techniques.

References

1. Torralba, A., Efros, A.A.: Unbiased look at dataset bias. In: CVPR. (2011)
2. Tzeng, E., Hoffman, J., Saenko, K., Darrell, T.: Adversarial discriminative domain adaptation. In: CVPR. (2017)
3. Sun, B., Saenko, K.: Deep coral: Correlation alignment for deep domain adaptation. In: ECCV Workshops. (2016)
4. Long, M., Cao, Y., Wang, J., Jordan, M.I.: Learning transferable features with deep adaptation networks. In: ICML. (2015)
5. Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. In: ICML. (2015)
6. Tzeng, E., Hoffman, J., Darrell, T., Saenko, K.: Simultaneous deep transfer across domains and tasks. In: ICCV. (2015)
7. Cohn, D., Atlas, L., Ladner, R.: Improving generalization with active learning. Machine Learning (1994)
8. Settles, B.: Active learning literature survey. (2010)
9. Yeung, S., Ramanathan, V., Russakovsky, O., Shen, L., Mori, G., Fei-Fei, L.: Learning to learn from noisy web videos. In: CVPR. (2017)
10. Saenko, K., Kulis, B., Fritz, M., Darrell, T.: Adapting visual category models to new domains. In: ECCV. (2010)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
12. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. (2009)
13. Kang, G., Zheng, L., Yan, Y., Yang, Y.: Deep adversarial attention alignment for unsupervised domain adaptation: the benefit of target expectation maximization. CoRR (2018)
14. Rai, P., Saha, A., Daumé, III, H., Venkatasubramanian, S.: Domain adaptation meets active learning. In: ALNLP. (2010)
15. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
16. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI. (2016)
17. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research (2008)
18. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
19. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)