Network Architecture Search for Domain Adaptation

Chen Zhao

`https://arxiv.org/pdf/2008.05706.pdf`

Introduction

Supervised machine learning models (Φ) aim to minimize the empirical test error $((\Phi(x), y))$ by optimizing Φ on training data (x) and ground truth labels (y), assuming that the training and testing data are sampled i.i.d from the same distribution. While in practical, the training and testing data are typically collected from related domains under different distributions, a phenomenon known as domain shift (or domain discrepancy). To avoid the cost of annotating each new test data, Unsupervised Domain Adaptation (UDA) tackles domain shift by transferring the knowledge learned from a rich-labeled source domain ($P(x^s, y^s)$) to the unlabeled target domain ($Q(x^t)$). While such models typically learn feature mapping from one domain ($\Phi(x^s)$) to another ($\Phi(x^t)$) or derive a joint representation across domains $(\Phi(x^s) \otimes \Phi(xt^{)})$, the developed models have limited capacities in deriving an optimal neural architecture specific for domain transfer. To efficiently devise a neural architecture across different data domains, the authors propose a novel learning task called NASDA (Neural Architecture Search for Domain Adaptation). The ultimate goal of NASDA is to minimize the validation loss of the target domain (L^t_{val}). We postulate that a solution to NASDA should not only minimize validation loss of the source domain (L^s_{val}), but should also reduce the domain gap between the source and target.

Learning Object

$$\begin{aligned}
&\Phi_{\alpha,w} = \mathrm{argmin}_{\alpha} \mathcal{L}^s_{val}(w^*(\alpha), \alpha) + \mathrm{disc}(\Phi^*(\mathbf{x}^s), \Phi^*(\mathbf{x}^t)) \\
&\text{s.t.} \quad w^*(\alpha) = \mathrm{argmin}_w \ \mathcal{L}^s_{train}(w, \alpha)
\end{aligned}$$

where $\Phi^* = \Phi_{\alpha, w^*(\alpha)}$ and $disc(\Phi^*(\mathbf{x}^s), \Phi^*(\mathbf{x}^t))$ denotes the domain discrepancy between the source and target. Note that in unsupervised domain adaptation, $\mathcal{L}^t_{train}$ and $\mathcal{L}^t_{val}$ cannot be computed directly due to the lack of label in the target domain.

Overview

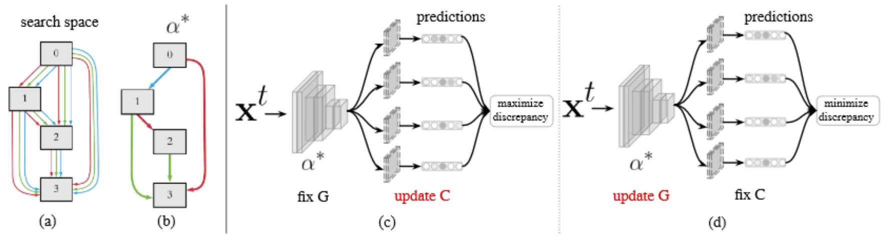

Figure 1: An overview of NASDA: (a) Continuous relaxation of the research space by placing a mixture of the candidate operations on each edge. (b) Inducing the final architecture by joint optimization of the neural architecture parameters α and network weights w, supervised by minimizing the validation loss on the source domain and reducing the domain discrepancy. (c)(d) Adversarial training of the derive feature generator G and classifiers C.

The algorithm is comprised with two training phases, as shown in the above figure. The first is the neural architecture searching phase, aiming to derive an optimal neural architecture (Φ^*), following the learning schema in previous slide. The second training phase aims to learn a good feature generator with task-specific loss, based on the derived Φ^* from the first phase.

NAS for Domain Adaptation

Inspired by the gradient-based hyperparameter optimization, we set the architecture parameters α as a special type of hyperparameter. This implies a bilevel optimization problem with α as the upper-level variable and w as the lower-level variable. In practice, we utilize the MK-MMD to evaluate the domain discrepancy. The optimization can be summarized as follows:

$$\Phi_{\alpha,w} = \operatorname{argmin}_{\alpha}\left(\mathcal{L}^{s}_{val}(w^*(\alpha),\alpha) + \lambda \hat{d}^{2}_{k}\left(\Phi(\mathbf{x}^s),\Phi(\mathbf{x}^t)\right)\right)$$
$$\text{s.t. } w^*(\alpha) = \operatorname{argmin}_{w}\ \mathcal{L}^{s}_{train}(w,\alpha)$$

Where λ is the trade-off hyperparameter between the source validation loss and the MK-MMD loss.

Maximum Classifier Discrepancy for Unsupervised Domain Adaptation

See Figure 2.

Adversarial Training for Domain Adaptation

First, we train both G and C to classify the source samples correctly with cross-entropy loss. This step is crucial as it enables G and C to extract the

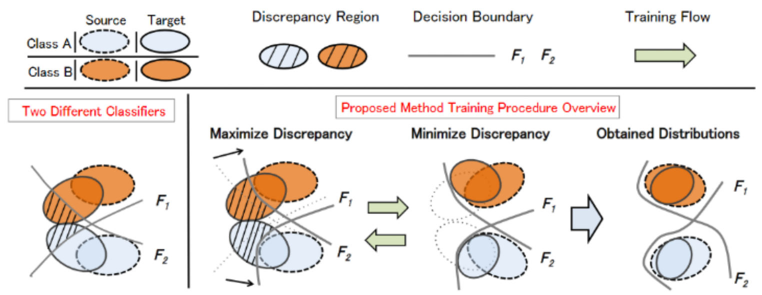

Figure 2. (Best viewed in color.) Example of two classifiers with an overview of the proposed method. Discrepancy refers to the disagreement between the predictions of two classifiers. First, we can see that the target samples outside the support of the source can be measured by two different classifiers (Leftmost, *Two different classifiers*). Second, regarding the training procedure, we solve a minimax problem in which we find two classifiers that *maximize* the discrepancy on the target sample, and then generate features that *minimize* this discrepancy.

task-specific features. The training objective is $\min_{G,C} L^s(x^s, y^s)$ and the loss function is defined as follows: In the second step, we are aiming to diversify C.

$$\mathcal{L}^s(\mathbf{x}^s, \mathbf{y}^s) = -\mathbb{E}_{(\mathbf{x}^s,\mathbf{y}^s)\sim\mathcal{D}^s} \sum_{k=1}^{K} \mathbb{1}_{[k=\mathbf{y}^s]} \log p(\mathbf{y}^s|\mathbf{x}^s)$$

To establish this goal, we fix G and train C to increase the discrepancy of C's output. To avoid mode collapse (e.g.$C^{(1)}$ outputs all zeros and $C^{(2)}$ output all ones), we add $L^s(x^s, y^s)$ as a regularizer in the training process. The high-level intuition is that we do not expect C to forget the information learned in the first step in the training process. The training objective is $min_C L^s(x^s, y^s) - L_{adv}(x^t)$, where the adversarial loss is defined as:

$$\mathcal{L}_{\text{adv}}(\mathbf{x}^t) = \mathbb{E}_{\mathbf{x}^t\sim\mathcal{D}^t} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \|(p_i(\mathbf{y}|\mathbf{x}^t) - p_j(\mathbf{y}|\mathbf{x}^t)\|_1$$