Domain-Specific Batch Normalization for Unsupervised Domain Adaptation

Abstract

- Aim to adapt to both domains by specializing batch normalization layers in convolutional neural networks while allowing them to share all other model parameters
- 1. Estimate pseudo-labels for the examples in the target domain using an external unsupervised domain adaptation algorithm
- 2. Learn the final model using a multi-task classification loss for the source and target domains.
- Two domains have separate batch-normalization layers in both stages

Preliminaries

- two state-of-the-art approaches for the integration of domain-specific batch normalization technique
- 1. Moving Semantic Transfer Network
- the loss function encourages two domains to have the same distribution, especially by adding adversarial and semantic matching loss terms.
- 2. Class Prediction Uncertainty Alignment
- strikingly simple approach that only aligns the class probabilities across domains

Moving Semantic Transfer Network

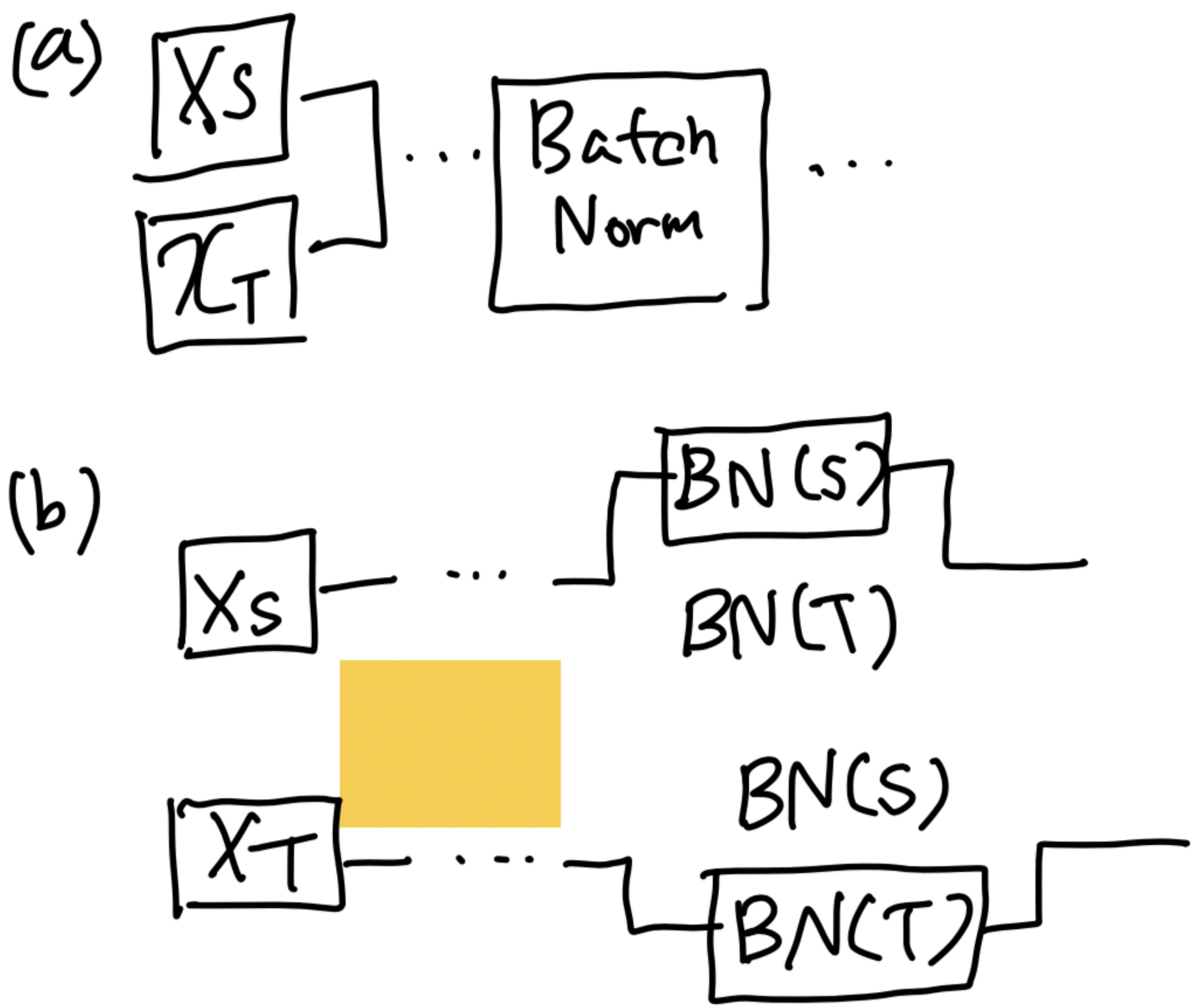
$$L = L_{cls}(X_S) + \lambda L_{da}(X_S, X_T) + L_{SM}(X_S, X_T)$$

- classification loss (cross entropy loss)
- domain adversarial loss
- semantic matching loss

Class Prediction Uncertainty Alignment

- $L = L_{cls}(X_S) + L_{da}(X_S, X_T)$

Illustration between BN and DSBN



Batch Normalization

- A batch normalization layer whitens activations within a mini-batch of N examples for each channel dimension
- transforms the whitened activations using affine parameters
- sharing the mean and variance for both source and target domain are inappropriate if domain shift is significant

Denoting by $X \in \mathbb{R}^{H \times W \times N}$ activations

In each channel,

BN is expressed as

$$BN(X[i,j,n]; \gamma, \beta) = \gamma \cdot \hat{X}[i,j,n] + \beta$$

where

$$\hat{X}[i,j,n] = \frac{X[i,j,n] - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

The mean and variance of activation within a mini-batch, μ and σ are computed by

$$\mu = \frac{\sum_n \sum_{i,j} X[i,j,n]}{N \cdot H \cdot W}$$

$$\sigma^2 = \frac{\sum_n \sum_{i,j} (X[i,j,n] - \mu)^2}{N \cdot H \cdot W}$$

Domain-Specific Batch Normalization

- Use multiple sets of BN reserved for each domain
- allocate domain-specific affine parameters for each domain level
- estimate the mean and variance of activations for each domain separately
- capture domain specific information by estimating batch statistics and learning affine parameters for each domain separately
- replace all BN layers with DSBN layers

$$\mathrm{DSBN}_d(x_d[i,j,n];\gamma_d,\beta_d) = \gamma_d \cdot \hat{x}_d[i,j,n] + \beta_d$$

where

$$\hat{x}_d[i,j,n] = \frac{x_d[i,j,n] - \mu_d}{\sqrt{\sigma^2 + \varepsilon}}$$

and

$$\mu_d = \frac{\sum_n \sum_{i,j} x_d[i,j,n]}{N \cdot H \cdot W}$$

$$\sigma_d^2 = \frac{\sum_n \sum_{i,j} (x_d[i,j,n] - \mu_d)^2}{N \cdot H \cdot W}$$

Extension to Multi-Source Domain Adaptation

4.3 Extension to Multi-Source Domain Adaptation

$$L = \frac{1}{|D_S|} \sum_{i}^{|D_S|} \left(L_{cls}(X_{S_i}) + L_{align}(X_{S_i}, X_T) \right)$$

where $D_S = \{X_{S_1}, X_{S_2}, \cdots\}$ is a set of source domains

Domain Adaptation with DSBN

- 1. train an existing unsupervised domain adaptation network to generate initial pseudo-labels of target domain data

- 2. learn the final models of both domains using the ground-truth labels in the source domain, the pseudo-labels in the target domain as supervision

Stage 1: Training Initial Pseudo Labeler

- choose state-of-the-art model as the initial pseudo-label generator: MSTN and CPUA
- F_T^1

Stage 2: Self-training with Pseudo Labels

- $L = L_{cls}(X_S) + L_{cls}^{pseudo}(X_T)$
- simple summation of two loss terms from two domains
- $L_{cls} = \sum_{x,y} L(F_S^2(x), y)$
- $L_{cls}^{pseudo}(X_T) = \sum_{x} L(F_T^2(x), y')$
- cross entropy loss

- conduct the second stage procedure iteratively

follows:

$$y' = \underset{c \in C}{\operatorname{argmax}} \left\{ (1-\lambda) F_T^1(x)[c] + \lambda F_T^2(x)[c] \right\}$$

$\boxed{F_T^i[c]\text{: prediction score of class } c}$

weight factor (increases gradually)

$$\lambda = \frac{2}{1+\exp(-r \cdot p)} - 1$$

-

Experiments

- Datasets: VisDA-C, Office-31, Office-Home

- Implementation details: construct mini-batches for each domain and forward them separately, batch size set to 40

  Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$

  initial learning rate $\eta_0 = 1.0 \times 10^{-4}$

  5.0×10^{-5}

  learning rate is adjusted by the formula

- $$\eta_p = \frac{\eta_0}{(1+\alpha p)^\beta} \quad (\alpha = 10, \beta = 0.75)$$

  The maximum number of iterations of the optimizer is set to 50,000.

Results

① VisDA-C

MSTN	65.0
DSBN	80.2
CPUA	66.6
DSBN	76.2

② Office-31

MSTN	86.5
DSBN	88.3
CPUA	86.4
DSBN	88.3

③ Office-Home

MSTN	81.2
DSBN	82.3