Multiple Source Domain Adaptation with Adversarial Training of Neural Networks

Zhao et al., NeurlIPS 2018
The paper is an extension of domain alignment (i.e. DANN) over multiple source domains

Highlight the difference in the theorems, *bounding on target error*, for two domains vs many domains, leading to the implementation of the model

There are few ways to extend this paper,

- it can be combined with other current advanced DA models, and or
- it can be straightforwardly applied other tasks, e.g., cross-lingual
Bounding on Target Error on Two Domains Adaptation

Theorem

Let $\mathcal{H}$ be a hypothesis class. If $\mathcal{U}^s$ and $\mathcal{U}^t$ are source and target samples respectively, then for any $\delta \in (0, 1)$, with the probability at least $1 - \delta$

$$
\epsilon^t(h, f^t) \leq \epsilon^s(h, f^s) + \frac{1}{2} \hat{d}_{\mathcal{H} \Delta \mathcal{H}}(\mathcal{U}^s, \mathcal{U}^t) + \lambda^* + \text{const}
$$

where $\lambda^* = \min_{h \in \mathcal{H}} \epsilon^s(h, f^s) + \epsilon^t(h, f^t)$ is the optimal joint error of both source and target domains

Assumptions

- the two domains are aligned $\rightarrow$ train domain adversarial adaptation
- good enough classifier for both domains simultaneously $\rightarrow$ current issue

With the assumptions, the target error is bounded by
- source error $\epsilon^s(h, f^s)$ $\rightarrow$ train classifier on source samples
- distance between two domain distributions when they are aligned $\frac{1}{2} \hat{d}_{\mathcal{H} \Delta \mathcal{H}}(\mathcal{U}^s, \mathcal{U}^t)$ $\rightarrow$ train domain adversarial adaptation
- optimal joint error of both domains $\lambda^*$ $\rightarrow$ often ignored because of the assumption
Bounding on Target Error on Multiple Domains Adaptation

**Theorem**

Let $\mathcal{H}$ be a hypothesis class. If $\{\mathcal{U}^{s,i}\}_{i=1}^{K}$ are multiple source samples and $\mathcal{U}^t$ are target samples respectively, then for any $\delta \in (0, 1)$, with the probability at least $1 - \delta$

$$
\epsilon^t(h, f^t) \leq \max_i [\epsilon^s(h, f^{s,i}) + \frac{1}{2} \hat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{U}^{s,i}, \mathcal{U}^t)] + \lambda^* + \text{const}
$$

where $\lambda^* = \min_{h \in \mathcal{H}} \left( \max_i \epsilon^s(h, f^{s,i}) + \epsilon^t(h, f^t) \right)$ is the optimal joint error of source domains and target domain.

- Similar to two domains adaptations, this model tries to align all domains by minimizing the **worst domain** max error of
  - source error
  - distance between two distributions between the source domain and target domain
- This bound is loose one
- There are other models trying to make better weighted combination of the above errors
Implementation

- First part, similar to DANN, insert **gradient reversal layer (GRL)** between the encoder and domain classifier

```python
sd, td = [], []
for i in range(self.num_domains):
    sdomains.append(F.log_softmax(self.domains[i](self.gri[i](shrelu[i])), dim=1))
    tdomains.append(F.log_softmax(self.domains[i](self.gri[i](threlu[i])), dim=1))
return logprobs, sdomains, tdomains
```

- Second part, choosing the worst domain to minimize

```python
losses = torch.stack([F.nll_loss(logprobs[j], ys[j]) for j in range(num_domains)])
domain_losses = torch.stack([F.nll_loss(sdomains[j], slables) +
                              F.nll_loss(tdomains[j], tlabels) for j in range(num_domains)])
# Different final loss function depending on different training modes.
if mode == "maxmin":
    loss = torch.max(losses) + mu * torch.min(domain_losses)
elif mode == "dynamic":
    loss = torch.log(torch.sum(torch.exp(gamma * (losses + mu * domain_losses)))) / gamma
```

- two options: hardmax vs softmax
## Benchmark

### Sentiment Analysis

<table>
<thead>
<tr>
<th>Train/Test</th>
<th>MLPNet</th>
<th>mSDA</th>
<th>sDANN</th>
<th>cDANN</th>
<th>MDANs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>H-Max</td>
</tr>
<tr>
<td>D+E+K/B</td>
<td>0.7655</td>
<td>0.7698</td>
<td>0.7650</td>
<td>0.7789</td>
<td>0.7845</td>
</tr>
<tr>
<td>B+E+K/D</td>
<td>0.7588</td>
<td>0.7861</td>
<td>0.7732</td>
<td>0.7886</td>
<td>0.7797</td>
</tr>
<tr>
<td>B+D+K/E</td>
<td>0.8460</td>
<td>0.8198</td>
<td>0.8381</td>
<td>0.8491</td>
<td>0.8483</td>
</tr>
<tr>
<td>B+D+E/K</td>
<td>0.8545</td>
<td>0.8426</td>
<td>0.8433</td>
<td>0.8639</td>
<td>0.8580</td>
</tr>
</tbody>
</table>

### Image Classification

<table>
<thead>
<tr>
<th>Train/Test</th>
<th>best-Single Source</th>
<th>best-Single DANN</th>
<th>Combine Source</th>
<th>Combine DANN</th>
<th>MDAN</th>
<th>Target Only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hard-Max</td>
<td>Soft-Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sv+Mm+Sy/Mt</td>
<td>0.964</td>
<td>0.967</td>
<td>0.938</td>
<td>0.925</td>
<td>0.976</td>
<td><strong>0.979</strong></td>
</tr>
<tr>
<td>Mt+Sv+Sy/Mm</td>
<td>0.519</td>
<td>0.591</td>
<td>0.561</td>
<td>0.651</td>
<td>0.663</td>
<td><strong>0.687</strong></td>
</tr>
<tr>
<td>Mm+Mt+Sy/Sv</td>
<td>0.814</td>
<td><strong>0.818</strong></td>
<td>0.771</td>
<td>0.776</td>
<td>0.802</td>
<td>0.816</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thank you!