Semi-supervised and transfer Learning

Makoto Yamada
makoto.yamada@oist.jp
Kyoto University

Review: Supervised Learning

Problem formulation of supervised learning.

- Input vector: $\boldsymbol{x} = (x_1, x_2, \dots, x_d)^\top \in \mathbb{R}^d$
- Output: $y \in \mathbb{R}$
- $(\boldsymbol{x}_i, y_i) \overset{\text{i.i.d.}}{\sim} p(\boldsymbol{x}, y)$
- Labeled data: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$
- Model: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x}$. (Linear model)

Risk: $R(\boldsymbol{w}) = \iint \text{loss}(y, f(\boldsymbol{x}; \boldsymbol{w})) p(\boldsymbol{x}, y) \text{d}\boldsymbol{x} \text{d}y$

Empirical Risk: $R_{emp}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \text{loss}(y_i, f(\boldsymbol{x}_i; \boldsymbol{w}))$

Empirical Risk Minimization (ERM): $\widehat{\boldsymbol{w}} = \text{argmin}_{\boldsymbol{w}} \; R_{emp}(\boldsymbol{w})$

Semi-Supervised Learning

Problem formulation of semi-supervised learning.

- $(\boldsymbol{x}_i, y_i) \overset{\text{i.i.d.}}{\sim} p(\boldsymbol{x}, y)$
- $\boldsymbol{x}_i \overset{\text{i.i.d.}}{\sim} p(\boldsymbol{x})$
- Labeled data: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$
- Unlabeled data: $\{\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \ldots, \boldsymbol{x}_{n+m}\}$
- Usually $n \ll m$ and n is small

Semi-supervised learning:

- We have both labeled and unlabeled samples.
- Semi-supervised learning uses both labeled and unlabeled samples.
- The unlabeled samples follow the same distribution of the marginal distribution of $p(\boldsymbol{x}, y)$

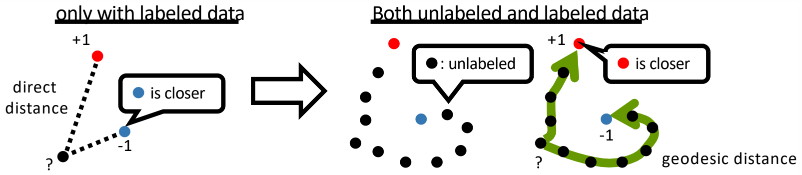
Role of unlabeled data

Data generation process

- Input $\boldsymbol{x}$ is generated by a distribution with $p(\boldsymbol{x})$.
- Output y for $\boldsymbol{x}$ is generated by conditional distribution with probability density $p(y|\boldsymbol{x})$.

Unlabeled data can be used for capturing $p(\boldsymbol{x})$

- input data distribution, input space metric, or better representation.



Semi-supervised learning frameworks

- Weighted maximum likelihood estimation
- Graph-based learning
- Self-training
- Clustering
- Generative models

Weighted maximum likelihood

The original goal of MLE is to maximize:

$$\mathbb{E}_{p(\boldsymbol{x},y)}[\log p(y|\boldsymbol{x})] = \iint \log P(y|\boldsymbol{x};\boldsymbol{w}) \underbrace{p(y|\boldsymbol{x})p(\boldsymbol{x})}_{p(\boldsymbol{x},y)} \mathrm{d}\boldsymbol{x}\mathrm{d}y,$$

$$\approx \frac{1}{n}\sum_{i=1}^{n} \log(P(y_i|\boldsymbol{x}_i;\boldsymbol{w}))$$

where $P(y|\boldsymbol{x};\boldsymbol{w})$ is a model. Each training instance is equally weighted.

Note, MLE is equivalent to maximize the negative log-likelihood function:

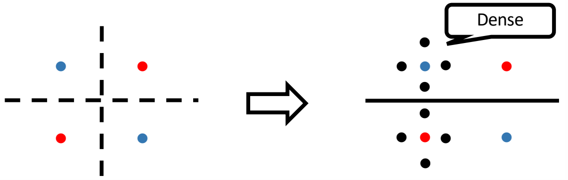
$$L(\boldsymbol{w}) = \log\left(\prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i;\boldsymbol{w})\right) \propto \frac{1}{n}\sum_{i=1}^{n}\log(P(y_i|\boldsymbol{x}_i;\boldsymbol{w}))$$

Weighted maximum likelihood

Weighted maximum likelihood:

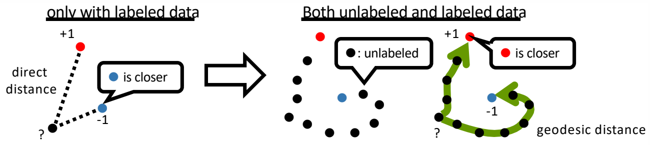
$$\max_{\boldsymbol{w}} \sum_{i=1}^{n} p(\boldsymbol{x}_i) \log(P(y_i|\boldsymbol{x}_i; \boldsymbol{w}))$$

- Each training data instance is weighted by $p(\boldsymbol{x}_i)$.
- $p(\boldsymbol{x})$ is estimated by using unlabeled data.
- Denser areas are largely weighted
- Training a classifier focusing on the dense areas



Graph-based method

- Basic idea: construct a graph capturing the intrinsic shape of input space, and make prediction on the graph.
- Assumption: Data lie on a manifold in the feature space
- The graph represent adjacency relationships among data
- K-nearest neighbor graph (e.g., $A_{ij} = \{0, 1\}$)
- Edge-weighted graph with e.g., $A_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2)$



Label propagation

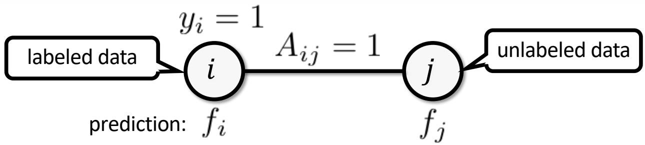
Basic idea: Adjacent instances tend to have the same label.

Transductive setting (we have test instances)

$$\min_{\boldsymbol{f} \in \mathbb{R}^{n+m}} \quad \sum_{i=1}^{n} (f_i - y_i)^2 + \lambda \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} A_{ij} (f_i - f_j)^2,$$

where $\lambda > 0$ is the regularization parameter.

- 1st term: (squared) loss function to fit to labeled data.
- 2nd term: regularization function to make adjacent nodes to have similar predictions.



Transfer Learning

Supervised Learning:

- Training $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$
- Test $(\boldsymbol{x}^{\text{te}}, y^{\text{te}}) \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x}, y)$ (Not observed during training)
- $p_{\text{tr}} = p_{\text{te}}$ (Training and test distributions are same)

Semi-supervised Learning:

- Training $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$, $\{\boldsymbol{x}_i^{\text{tr}}\}_{i=n+1}^{n+m} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x})$.
- Test $(\boldsymbol{x}^{\text{te}}, y^{\text{te}}) \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x}, y)$ (Not observed during training)
- $p_{\text{tr}} = p_{\text{te}}$ (Training and test distributions are same)

If $p_{\text{tr}} \neq p_{\text{te}}$, supervised method and semi-supervised method do not perform well. A solution: Transfer Learning!

Problem formulations of Transfer Learning

Unsupervised transfer learning

- $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$,
- $\{\boldsymbol{x}_j^{\text{te}}\}_{j=1}^{n_{\text{te}}} \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x})$, $n_{\text{tr}} \ll n_{\text{te}}$

Supervised transfer learning

- $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$
- $\{(\boldsymbol{x}_j^{\text{te}}, y_j^{\text{te}})\}_{j=1}^{n_{\text{te}}} \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x}, y)$, $n_{\text{te}} \ll n_{\text{tr}}$

Semi-supervised transfer learning

- $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$
- $\{(\boldsymbol{x}_j^{\text{te}}, y_j^{\text{te}})\}_{j=1}^{n_{\text{te}}} \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x}, y)$, $n_{\text{te}} \ll n_{\text{tr}}$
- $\{\boldsymbol{x}_j^{\text{te}}\}_{j=n_{\text{te}}+1}^{n_{\text{te}}+n'_{\text{te}}} \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x})$, $n_{\text{tr}} \ll n_{\text{te}}$

Unsupervised Transfer Learning

We assume

- It does not need to have test label
- Need some assumption

Standard approaches

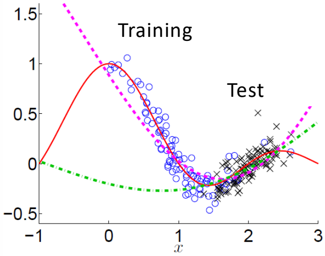
- Importance weighted method (e.g., Covariate shift adaptation)
- Subspace based method.

Covariate Shift Adaptation [1]

Problem setup:

- $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$,
- $\{\boldsymbol{x}_j^{\text{te}}\}_{j=1}^{n_{\text{te}}} \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x})$, $n_{\text{tr}} \ll n_{\text{te}}$

Key idea: Learning a function so that error in test data is minimized under the assumption $p_{\text{tr}}(y|\boldsymbol{x}) = p_{\text{te}}(y|\boldsymbol{x})$



Covariate Shift Adaptation

The risk for $p_{\text{te}}(\boldsymbol{x}, y)$ can be written as

$$\begin{aligned}
J(\boldsymbol{w}) &= \iint L(y, f(\boldsymbol{x}))p_{\text{te}}(\boldsymbol{x}, y)\mathrm{d}\boldsymbol{x}\mathrm{d}y \\
&= \iint L(y, f(\boldsymbol{x}))\frac{p_{\text{te}}(\boldsymbol{x}, y)}{p_{\text{tr}}(\boldsymbol{x}, y)}p_{\text{tr}}(\boldsymbol{x}, y)\mathrm{d}\boldsymbol{x}\mathrm{d}y \\
&= \iint L(y, f(\boldsymbol{x}))\frac{p_{\text{te}}(y|\boldsymbol{x})p_{\text{te}}(\boldsymbol{x})}{p_{\text{tr}}(y|\boldsymbol{x})p_{\text{tr}}(\boldsymbol{x})}p_{\text{tr}}(y, \boldsymbol{x})\mathrm{d}\boldsymbol{x}\mathrm{d}y \\
&= \iint L(y, f(\boldsymbol{x}))\frac{p_{\text{te}}(\boldsymbol{x})}{p_{\text{tr}}(\boldsymbol{x})}p_{\text{tr}}(y, \boldsymbol{x})\mathrm{d}\boldsymbol{x}\mathrm{d}y \\
&\approx \frac{1}{n_{\text{tr}}}\sum_{i=1}^{n_{\text{tr}}} L(y_i^{\text{tr}}, f(\boldsymbol{x}_i^{\text{tr}}))\frac{p_{\text{te}}(\boldsymbol{x}_i^{\text{tr}})}{p_{\text{tr}}(\boldsymbol{x}_i^{\text{tr}})}
\end{aligned}$$

Actually, it is a weighted maximum likelihood problem. Note $\frac{p_{\text{te}}(\boldsymbol{x}_i^{\text{tr}})}{p_{\text{tr}}(\boldsymbol{x}_i^{\text{tr}})}$ is a ratio of probability densities (density-ratio).

Covariate Shift Adaptation

Exponentially-flattened Importance weighted empirical risk minimization (IW-ERM) [1]:

$$\min_{f \in \mathcal{F}} \quad \frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} L(y_i^{\mathrm{tr}}, f(\boldsymbol{x}_i^{\mathrm{tr}})) \left(\frac{p_{\mathrm{te}}(\boldsymbol{x}_i^{\mathrm{tr}})}{p_{\mathrm{tr}}(\boldsymbol{x}_i^{\mathrm{tr}})} \right)^{\tau}$$

where $0 \leq \tau \leq 1$ is a tuning parameter for stabilizing the covariate shift adaptation.

- $\tau = 0 \rightarrow$ ERM
- $0 < \tau < 1 \rightarrow$ Intermediate
- $\tau = 1$ IW-ERM

Setting τ to $0 < \tau < 1$ is practically useful.

Covariate Shift Adaptation

Relative Importance weighted empirical risk minimization (RIW-ERM) [2]:

$$\min_{f\in\mathcal{F}} \quad \frac{1}{n_{\mathsf{tr}}}\sum_{i=1}^{n_{\mathsf{tr}}} L(y_i^{\mathsf{tr}}, f(\boldsymbol{x}_i^{\mathsf{tr}}))\frac{p_{\mathsf{te}}(\boldsymbol{x}_i^{\mathsf{tr}})}{(1-\alpha)p_{\mathsf{te}}(\boldsymbol{x}_i^{\mathsf{tr}})+\alpha p_{\mathsf{tr}}(\boldsymbol{x}_i^{\mathsf{tr}})}$$

where $0 \leq \tau \leq 1$ is a tuning parameter for stabilizing the covariate shift adaptation.

- $\alpha = 0 \rightarrow$ ERM
- $0 < \alpha < 1 \rightarrow$ Intermediate
- $\alpha = 1$ IW-ERM

$$r_\alpha(\boldsymbol{x}) = \frac{p_{\mathsf{te}}(\boldsymbol{x})}{(1-\alpha)p_{\mathsf{tr}}(\boldsymbol{x})+\alpha p_{\mathsf{tr}}(\boldsymbol{x})} < \frac{1}{1-\alpha}$$

The density ratio is bounded above by $1/(1-\alpha)$.

Importance Weighted Least Squares

The importance weighted least squares problem can be written as

$$\min_{\boldsymbol{w}} \quad J(\boldsymbol{w}) = \frac{1}{n_{\mathsf{tr}}} \sum_{i=1}^{n_{\mathsf{tr}}} r(\boldsymbol{x}_i^{\mathsf{tr}}) \| y_i^{\mathsf{tr}} - \boldsymbol{w}^\top \boldsymbol{x}_i^{\mathsf{tr}} \|_2^2,$$

where $r(\boldsymbol{x})$ is a weight function (e.g., density-ratio).

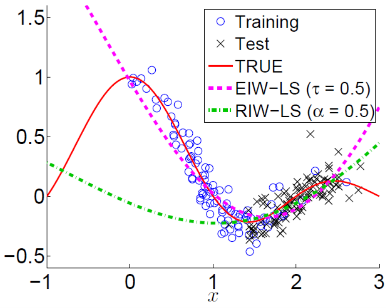
Take the derivative w.r.t. $\boldsymbol{w}$ and equating it to zero.

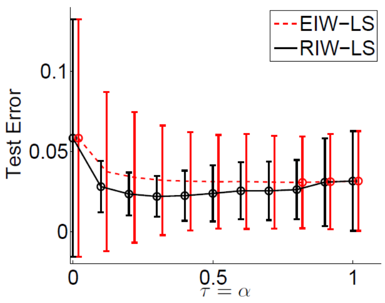
$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\frac{2}{n_{\mathsf{tr}}} \sum_{i=1}^{n_{\mathsf{tr}}} r(\boldsymbol{x}_i^{\mathsf{tr}})(y_i^{\mathsf{tr}} - \boldsymbol{w}^\top \boldsymbol{x}_i^{\mathsf{tr}}) \boldsymbol{x}_i^{\mathsf{tr}} = \boldsymbol{0}$$

$$\widehat{\boldsymbol{w}} = \left(\sum_{i=1}^{n_{\mathsf{tr}}} r(\boldsymbol{x}_i^{\mathsf{tr}}) \boldsymbol{x}_i^{\mathsf{tr}} \boldsymbol{x}_i^{\mathsf{tr}\top} \right)^{-1} \sum_{i=1}^{n_{\mathsf{tr}}} r(\boldsymbol{x}_i^{\mathsf{tr}}) y_i^{\mathsf{tr}} \boldsymbol{x}_i^{\mathsf{tr}}$$

Synthetic Example

Comparison of EIW-LS and RIW-LS:





Supervised Transfer Learning

Problem formulation:

- $\{(\boldsymbol{x}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{n_{\text{tr}}} \overset{\text{i.i.d.}}{\sim} p_{\text{tr}}(\boldsymbol{x}, y)$
- $\{(\boldsymbol{x}_j^{\text{te}}, y_j^{\text{te}})\}_{j=1}^{n_{\text{te}}} \overset{\text{i.i.d.}}{\sim} p_{\text{te}}(\boldsymbol{x}, y)$, $n_{\text{te}} \ll n_{\text{tr}}$

We assume to have a large number of training samples and a small number of paired target labeled samples.

- Frustratingly easy domain adaptation [3].
- Multi-task Learning
- Fine-tuning (Deep Learning)

Importance Weight

Naive approach: Pooling training and test samples

$$
\begin{aligned}
J(\boldsymbol{w}) &= \iint \text{loss}(y, f(\boldsymbol{x}; \boldsymbol{w})) p_{\text{te}}(\boldsymbol{x}, \boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \\
&= \alpha \iint \text{loss}(y, f(\boldsymbol{x}; \boldsymbol{w})) p_{\text{tr}}(\boldsymbol{x}, \boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \\
&\quad + (1-\alpha) \iint \text{loss}(y, f(\boldsymbol{x}; \boldsymbol{w})) p_{\text{te}}(\boldsymbol{x}, \boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \\
&\simeq \frac{\alpha}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \text{loss}(y_i^{\text{tr}}, f(\boldsymbol{x}_i^{\text{tr}}; \boldsymbol{w})) + \frac{(1-\alpha)}{n_{\text{te}}} \sum_{j=1}^{n_{\text{te}}} \text{loss}(y_j^{\text{te}}, f(\boldsymbol{x}_j^{\text{te}}; \boldsymbol{w})),
\end{aligned}
$$

where $0 \leq \alpha \leq 1$ is a tuning parameter to control trade off between source and target errors.

Multi-task Learning

Problem formulation:

- Task1: $\{(\boldsymbol{x}_i^{(1)}, y_i^{(1)})\}_{i=1}^{n_1} \overset{\text{i.i.d.}}{\sim} p_1(\boldsymbol{x}, y)$
- ...
- Task$M$: $\{(\boldsymbol{x}_j^{(M)}, y_j^{(M)})\}_{j=1}^{n_M} \overset{\text{i.i.d.}}{\sim} p_M(\boldsymbol{x}, y)$
- Linear Models: $f_1(\boldsymbol{x}^{(1)}) = \boldsymbol{w}_1^\top \boldsymbol{x}^{(1)}, f_2(\boldsymbol{x}^{(2)}) = \boldsymbol{w}_2^\top \boldsymbol{x}^{(2)}, \dots, f_M(\boldsymbol{x}^{(M)}) = \boldsymbol{w}_M^\top \boldsymbol{x}^{(M)}$

$$\min_{\boldsymbol{w}_1,\dots,\boldsymbol{w}_M} \quad \sum_{m=1}^{M} \frac{1}{n_m} \sum_{i=1}^{n_m} \text{loss}(y_i^{(m)}, f_m(\boldsymbol{x}^{(m)})) + \lambda R(\boldsymbol{w}_1, \dots, \boldsymbol{w}_M).$$

where $R(\boldsymbol{w}_1, \dots, \boldsymbol{w}_M)$ is a regularizer.

- $\lambda = 0$: Independently optimize $\boldsymbol{w}$s
- $\lambda > 0$: We share some information among models.

Multi-task Learning

Multi-task learning optimization (Graph-Laplacian).

$$\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_M} \quad \sum_{m=1}^{M} \frac{1}{n_m} \sum_{i=1}^{n_m} \text{loss}(y_i^{(m)}, f_m(\boldsymbol{x}_i^{(m)})) + \lambda \sum_{m=1}^{M} \sum_{m'=1}^{M} r_{m,m'} \|\boldsymbol{w}_m - \boldsymbol{w}_{m'}\|_2^2.$$

where $r_{m,m'} \geq 0$ is a model parameter (similarity between models). If $r_{m,m'} > 0$, we make $\boldsymbol{w}_m$ and $\boldsymbol{w}_{m'}$ close.

Multi-task Learning [4]

Other approach: Explicitly including shared parameter. We decompose $\boldsymbol{w}_m = \boldsymbol{w}_0 + \boldsymbol{v}_m$

That is

- $f_1(\boldsymbol{x}^{(1)}) = (\boldsymbol{w}_0 + \boldsymbol{v}_1)^\top \boldsymbol{x}^{(1)}$,
- $f_2(\boldsymbol{x}^{(2)}) = (\boldsymbol{w}_0 + \boldsymbol{v}_2)^\top \boldsymbol{x}^{(2)}$,
- $\ldots$
- $f_M(\boldsymbol{x}^{(M)}) = (\boldsymbol{w}_0 + \boldsymbol{v}_M)^\top \boldsymbol{x}^{(M)}$

where $\boldsymbol{w}_0$ is a common factor for all models.

For squared-loss, we can write the problem as

$$\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_M} \quad \frac{1}{2}\sum_{m=1}^{M}\frac{1}{n_m}\sum_{i=1}^{n_m}\left(y_i^{(m)} - (\boldsymbol{w}_0+\boldsymbol{v}_m)^\top \boldsymbol{x}_i^{(m)}\right)^2 + \lambda(\|\boldsymbol{w}_0\|_2^2 + \sum_{m=1}^{M}\|\boldsymbol{v}_m\|_2^2)$$

Supervised Transfer Learning: Frustratingly easy domain adaptation

A frustratingly easy feature augmentation approach:

$$\boldsymbol{z}^{\mathrm{tr}} = (\boldsymbol{x}^{\mathrm{tr}\top} \;\; \boldsymbol{x}^{\mathrm{tr}\top} \;\; \boldsymbol{0}_{\mathrm{d}}^{\top})^{\top}, \;\; \boldsymbol{z}^{\mathrm{te}} = (\boldsymbol{x}^{\mathrm{te}\top} \;\; \boldsymbol{0}_{\mathrm{d}}^{\top} \;\; \boldsymbol{x}^{\mathrm{te}\top})^{\top},$$

The inner product of $\boldsymbol{z}$ in the same domain is give as

$$\boldsymbol{z}^{\mathrm{tr}\top}\boldsymbol{z}^{\mathrm{tr}} = 2\boldsymbol{x}^{\mathrm{tr}\top}\boldsymbol{x}^{\mathrm{tr}}, \;\; \boldsymbol{z}^{\mathrm{te}\top}\boldsymbol{z}^{\mathrm{te}} = 2\boldsymbol{x}^{\mathrm{te}\top}\boldsymbol{x}^{\mathrm{te}},$$

while we have

$$\boldsymbol{z}^{\mathrm{tr}\top}\boldsymbol{z}^{\mathrm{te}} = \boldsymbol{x}^{\mathrm{tr}\top}\boldsymbol{x}^{\mathrm{tr}},.$$

Then, we train a supervised learning method with the transformed vectors $\boldsymbol{z}$. Super easy!!!!

Multi-task Learning

Supervised transfer learning can be regarded as a two-task learning problem. First task is for training and second task is for test.

Let us denote the transformed vectors as

$$\boldsymbol{z}^{\mathsf{tr}} = (\boldsymbol{x}^{\mathsf{tr}\top} \;\; \boldsymbol{x}^{\mathsf{tr}\top} \;\; \mathbf{0}_{\mathsf{d}}{}^{\top})^{\top} \in \mathbb{R}^{3d},$$
$$\boldsymbol{z}^{\mathsf{te}} = (\boldsymbol{x}^{\mathsf{te}\top} \;\; \mathbf{0}_{\mathsf{d}}{}^{\top} \;\; \boldsymbol{x}^{\mathsf{te}\top})^{\top} \in \mathbb{R}^{3d},$$

where $\mathbf{0}_{\mathsf{d}} \in \mathbb{R}^{d}$ is the vector whose elements are all zero.

And, we consider a linear regression problem: The model parameter of the linear model can be written as

$$\boldsymbol{w} = (\boldsymbol{w}_0^{\top} \;\; \boldsymbol{v}_1{}^{\top} \;\; \boldsymbol{v}_2{}^{\top})^{\top} \in \mathbb{R}^{3d}$$

Multi-task Learning

$$J(\boldsymbol{w}) = \frac{1}{2n_{\mathrm{tr}}}\sum_{i=1}^{n_{\mathrm{tr}}}\|y_i^{\mathrm{tr}} - \boldsymbol{z}_i^{\mathrm{tr}\top}\boldsymbol{w}\|_2^2 + \frac{1}{2n_{\mathrm{te}}}\sum_{i=1}^{n_{\mathrm{te}}}\|y_i^{\mathrm{te}} - \boldsymbol{z}_i^{\mathrm{te}\top}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$= \frac{1}{2}\sum_{m=1}^{M}\frac{1}{n_m}\sum_{i=1}^{n_m}\left(y_i^{(m)} - (\boldsymbol{w}_0+\boldsymbol{v}_m)^\top\boldsymbol{x}_i^{(m)}\right)^2 + \lambda(\|\boldsymbol{w}_0\|_2^2 + \sum_{m=1}^{M}\|\boldsymbol{v}_m\|_2^2),$$

where we use

$$\boldsymbol{w}^\top\boldsymbol{z}^{\mathrm{tr}} = (\boldsymbol{w}_0+\boldsymbol{v}_1)^\top\boldsymbol{x}^{\mathrm{tr}}, \quad \boldsymbol{w}^\top\boldsymbol{z}^{\mathrm{te}} = (\boldsymbol{w}_0+\boldsymbol{v}_2)^\top\boldsymbol{x}^{\mathrm{te}}$$

$$\boldsymbol{x}^{\mathrm{tr}} = \boldsymbol{x}^{(1)}, \quad \boldsymbol{x}^{\mathrm{te}} = \boldsymbol{x}^{(2)},$$

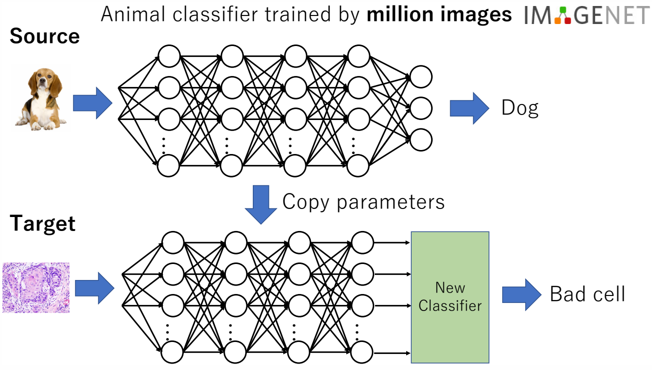
$$\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{w}_0\|_2^2 + \sum_{m=1}^{2}\|\boldsymbol{v}_m\|_2^2.$$

Frustratingly easy domain adaptation is a multi-task learning.

Fine-tuning

In deep learning context, Fine-tuning is a main approach for transfer learning.

- Prepare a pre-trained model.
- Updating model parameters using a new dataset.



Fine-tuning frameworks

There are mainly two approaches for fine tuning. Let us denote the pretrained model parameter as $\{\widehat{\boldsymbol{W}}_i\}_{\ell=1}^{L}$, where L is the number of layers (or blocks).

- Using pre-trained model as feature extractor. Fix the $L-1$ model parameters $\{\widehat{\boldsymbol{W}}_i\}_{\ell=1}^{L-1}$ and train the final layer $\boldsymbol{W}_L$ using a new dataset. (We can change the number of classes)
- Using pre-train model as initial model parameter. We train a few epochs using new dataset from the initial model parameters $\{\widehat{\boldsymbol{W}}_i\}_{\ell=1}^{L}$.

Fine-tuning (LoRA) [5]

In pre-trained model such as large language models (LLM), the number of model parameters are huge; it is expensive for fine-tuning.

The low rank adaptation (LoRA) is a widely used technique. We model the fine-tuned parameter $\bar{\boldsymbol{W}}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ as

$$\bar{\boldsymbol{W}}_\ell = \widehat{\boldsymbol{W}}_\ell + \boldsymbol{U}_\ell \boldsymbol{V}_\ell^\top,$$

where $\boldsymbol{U}_\ell \in \mathbb{R}^{d_\ell \times r}$ and $\boldsymbol{V}_\ell \in \mathbb{R}^{d_{\ell+1} \times r}$ $(r \ll d)$. The number of tuning parameters is only $\sum_{\ell=1}^{L-1}(d_\ell + d_{\ell+1})r$.

- Fine-tuning $\boldsymbol{U}_\ell, \boldsymbol{V}_\ell, \forall \ell$.
- Updating parameters $\widehat{\boldsymbol{W}}_\ell \leftarrow \widehat{\boldsymbol{W}}_\ell + \widehat{\boldsymbol{U}}_\ell \widehat{\boldsymbol{V}}_\ell^\top$

Summary

- Semi-supervised learning. Use unlabeled samples and assume the data distribution of unlabeled data is same as training.
- Weighted Maximum Likelihood, Graph-based method.
- Transfer Learning. Use samples from test data. Training and test distributions are different.
- Covariate shift adaptation, frustratingly easy domain adaptation.
- Fine-tuning (LoRA).

[1] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[2] Makoto Yamada, Taiji Suzuki, Takafumi Kanamori, Hirotaka Hachiya, and Masashi Sugiyama. Relative density-ratio estimation for robust distribution comparison. In *Advances in neural information processing systems*, pages 594–602, 2011.

[3] Hal Daumé III. Frustratingly easy domain adaptation. *ACL 2007*, page 256, 2007.

[4] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, 2004.

[5] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.