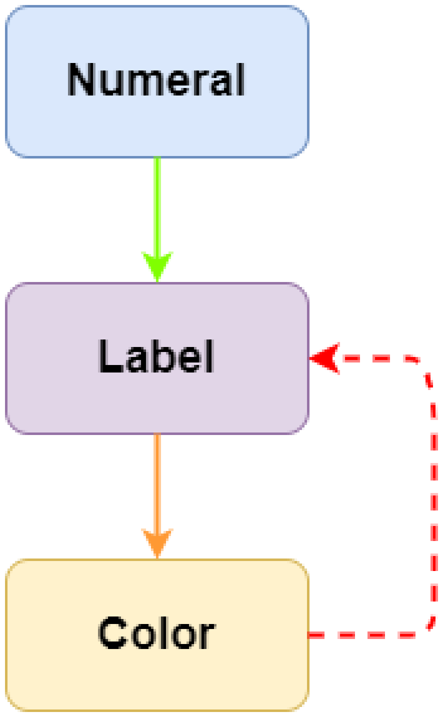
Recurrence and Exploration of Out-of-Distribution Algorithms

Xiangyuan Xue

cooperating with Bangjun Wang

Invariant Risk Minimization

- ERM
  - fail to catch causality
  - terrible performance on CMNIST
- IRM
  - seek for invariant predictor
  - lower environmental difference



Mathematical Derivation

- Target
  - optimal data representor $\mathbf{\Phi}$
  - optimal predictor $\boldsymbol{\omega}$

$$\min_{\substack{\mathbf{\Phi}:\boldsymbol{X}\to\boldsymbol{H}\\ \boldsymbol{\omega}:\boldsymbol{H}\to\boldsymbol{Y}}} \sum_{e\in E_{\text{train}}} R^e(\boldsymbol{\omega}\circ\mathbf{\Phi})$$

$$\text{s.t.} \quad \forall e \in E_{\text{train}} : \boldsymbol{\omega} \in \arg\min_{\boldsymbol{\omega}^*:\boldsymbol{H}\to\boldsymbol{Y}} R^e(\boldsymbol{\omega}^*\circ\mathbf{\Phi})$$

※ bi-level optimization, hard

Mathematical Derivation

- Constraints -> Penalty

$$L(\boldsymbol{\Phi}, \boldsymbol{\omega}) = \sum_{e \in E_{\text{train}}} \underbrace{R^e(\boldsymbol{\omega} \circ \boldsymbol{\Phi})}_{\text{empirical risk}} + \underbrace{\lambda D(\boldsymbol{\Phi}, \boldsymbol{\omega}, e)}_{\text{invariance}}$$

※ λ: penalty weight

- Hypothesis: $\boldsymbol{\Phi}$, $\boldsymbol{\omega}$ are both linear

Mathematical Derivation

- Single environment

$$\boldsymbol{Y}^e = \boldsymbol{\omega} \circ \boldsymbol{\Phi}(\boldsymbol{X}^e) \quad \xRightarrow{\text{in matrix form}} \quad \boldsymbol{\Phi}(\boldsymbol{X}^e)\boldsymbol{\omega} = \boldsymbol{Y}^e$$

- Least square solution

$$\boldsymbol{\omega}^e_{\boldsymbol{\Phi}} = \left[\boldsymbol{\Phi}(\boldsymbol{X}^e)^T \boldsymbol{\Phi}(\boldsymbol{X}^e)\right]^{-1} \boldsymbol{\Phi}(\boldsymbol{X}^e)^T \boldsymbol{Y}^e$$

Mathematical Derivation

- Transformation

$$\boldsymbol{\Phi}(\boldsymbol{X}^e)^T\boldsymbol{\Phi}(\boldsymbol{X}^e)\omega^e_{\boldsymbol{\Phi}} - \boldsymbol{\Phi}(\boldsymbol{X}^e)^T\boldsymbol{Y}^e = \mathbf{0}$$

- Define distance

$$D(\boldsymbol{\Phi}, \boldsymbol{\omega}, e) = \|\boldsymbol{\Phi}(\boldsymbol{X}^e)^T\boldsymbol{\Phi}(\boldsymbol{X}^e)\boldsymbol{\omega} - \boldsymbol{\Phi}(\boldsymbol{X}^e)^T\boldsymbol{Y}^e\|^2$$

※ It is reasonable to fix $\omega = \omega_0$

Mathematical Derivation

- Loss function

$$
\begin{aligned}
L_{\boldsymbol{\omega_0}} &= \sum_{e \in E_{\text{train}}} R^e(\boldsymbol{\omega_0} \circ \boldsymbol{\Phi}) + \lambda D_{\boldsymbol{\omega_0}}(\boldsymbol{\Phi}, e) \\
&= \sum_{e \in E_{\text{train}}} R^e(\boldsymbol{\omega_0} \circ \boldsymbol{\Phi}) + \lambda \| \boldsymbol{\Phi}(\boldsymbol{X}^e)^T \boldsymbol{\Phi}(\boldsymbol{X}^e)\boldsymbol{\omega} - \boldsymbol{\Phi}(\boldsymbol{X}^e)^T \boldsymbol{Y}^e \|^2 \\
&= \sum_{e \in E_{\text{train}}} R^e(\boldsymbol{\omega_0} \circ \boldsymbol{\Phi}) + \lambda \frac{\partial}{\partial \boldsymbol{\omega}} \left[\frac{1}{2} \left(\boldsymbol{\Phi}(\boldsymbol{X}^e)\boldsymbol{\omega} - \boldsymbol{Y} \right)^T \left(\boldsymbol{\Phi}(\boldsymbol{X}^e)\boldsymbol{\omega} - \boldsymbol{Y} \right) \right]_{\boldsymbol{\omega_0}} \\
&= \sum_{e \in E_{\text{train}}} R^e(\boldsymbol{\omega_0} \circ \boldsymbol{\Phi}) + \lambda \| \nabla_{\boldsymbol{\omega}} R^e(\boldsymbol{\omega} \circ \boldsymbol{\Phi}) \|^2_{\boldsymbol{\omega_0}}
\end{aligned}
$$

※ Reuse first term, convenient

Significant Amendment

- Ignored term for non-linear $\boldsymbol{\Phi}$

$$P = \|\nabla_{\boldsymbol{\omega}} R^e(\boldsymbol{\omega} \circ \boldsymbol{\Phi})\|^2_{\boldsymbol{\omega_0}}$$
$$= \|\nabla_{\boldsymbol{\omega}} R^e(\boldsymbol{\omega} \circ f \circ g)\|^2_{\boldsymbol{\omega_0}}$$
$$= \|\nabla_{\boldsymbol{\omega}} R^e(\boldsymbol{\omega} \circ f_g)\|^2_{\boldsymbol{\omega_0}} + \underbrace{\|\nabla_{\boldsymbol{\omega}} R^e(\boldsymbol{\omega} \circ g)\|^2_{\boldsymbol{\omega_0}}}_{\text{ignored term}}$$

- Amendment
  - hyper-parameter μ
  - sum up parameters in MLP

```
import torch
from torch import nn, optim, autograd
from torch.nn import functional as F
```

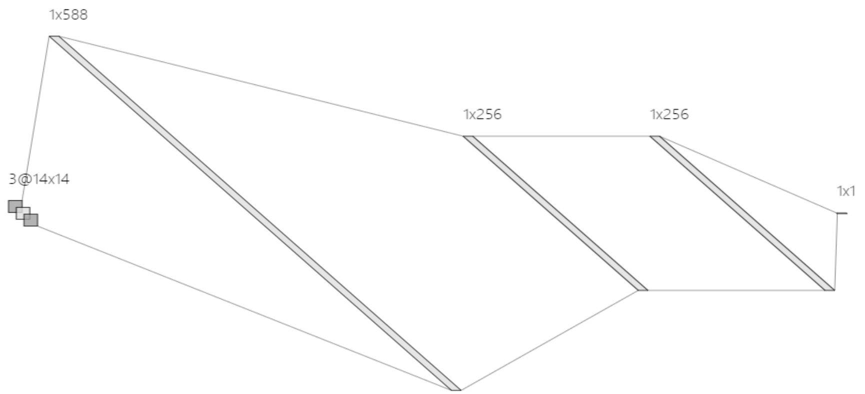
```
scale = torch.tensor(1., requires_grad=True).to(device)
loss = F.binary_cross_entropy_with_logits(scale * inputs, self.targets)
gradient = autograd.grad(loss, [scale], create_graph=True)[0]
```

Coincidentally, the ignore term is composed by all the parameters in former layers. For an MLP model, the output can be expressed by a complex compound function $F = f_1 \circ f_2 \circ \cdots \circ f_n$ where $f_i \in F$ is linear function $f_i = k_{i,1}x_1 + k_{i,2}x_2 + \cdots + k_{1,m}x_m$, then the ignored term can be written as:

$$\|\nabla_{\boldsymbol{\omega}} R^e(\boldsymbol{\omega} \circ g)\|^2_{\boldsymbol{\omega_0}} = \mu \sum_{f_i \in F} \sum_{k_j \in f} k^2_{i,j}$$

* μ is the term which we already figure out.

Algorithm Implementation

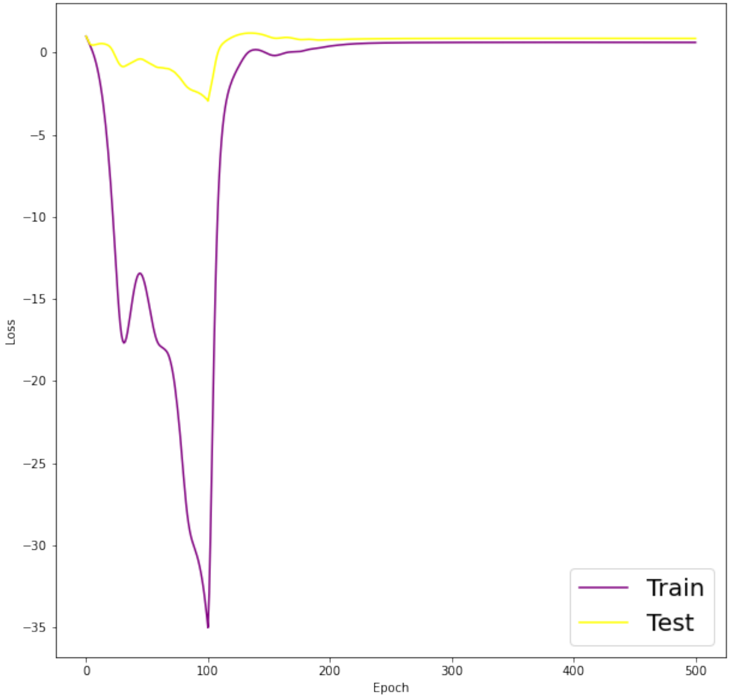



```python
class MNISTModel(nn.Module):
    def __init__(self):
        super(MNISTModel, self).__init__()
        layer1 = nn.Linear(3 * 14 * 14, 256)
        layer2 = nn.Linear(256, 256)
        layer3 = nn.Linear(256, 1)
        for layer in [layer1, layer2, layer3]:
            nn.init.xavier_uniform_(layer.weight)
            nn.init.zeros_(layer.bias)
        self.model = nn.Sequential(
            nn.Flatten(),
            layer1,
            nn.ReLU(True),
            layer2,
            nn.ReLU(True),
            layer3
        )

    def forward(self, inputs):
        outputs = self.model(inputs)
        return outputs
```

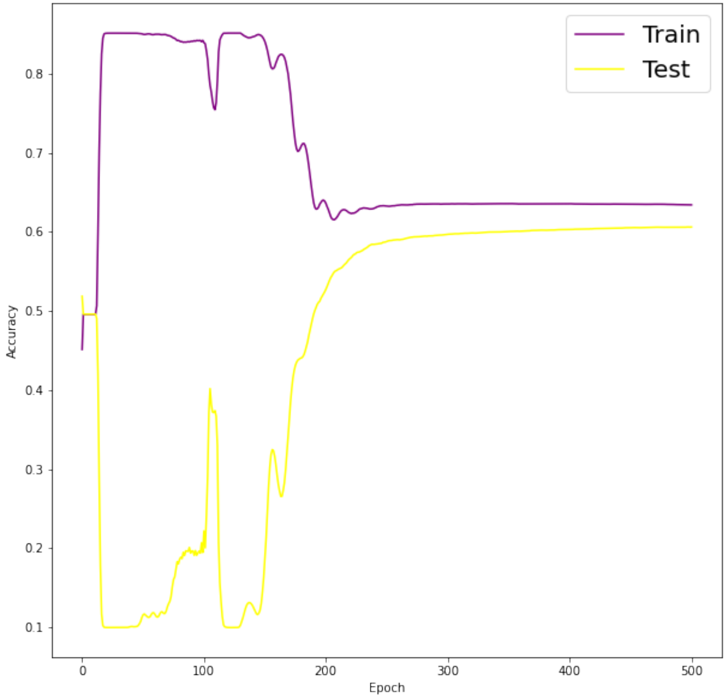
```python
class Environment:
    def __init__(self, dataset: MNISTDataset) -> None:
        self.size = len(dataset)
        images = []
        targets = []
        for image, target in dataset:
            images.append(image.unsqueeze_(0))
            targets.append(target)
        self.images = torch.cat(images, dim=0).to(device)
        self.targets = torch.Tensor(targets).unsqueeze_(1).to(device)
        self.loss = None
        self.accuracy = None
        self.penalty = None

    def update(self, inputs):
        self.loss = F.binary_cross_entropy_with_logits(self.targets, inputs)
        predictions = (inputs > 0.).float()
        self.accuracy = ((self.targets - predictions).abs() < 0.01).float().mean()
        scale = torch.tensor(1., requires_grad=True).to(device)
        loss = F.binary_cross_entropy_with_logits(scale * inputs, self.targets)
        gradient = autograd.grad(loss, [scale], create_graph=True)[0]
        self.penalty = torch.sum(gradient ** 2)
```

Algorithm Implementation

- Recurrence performance

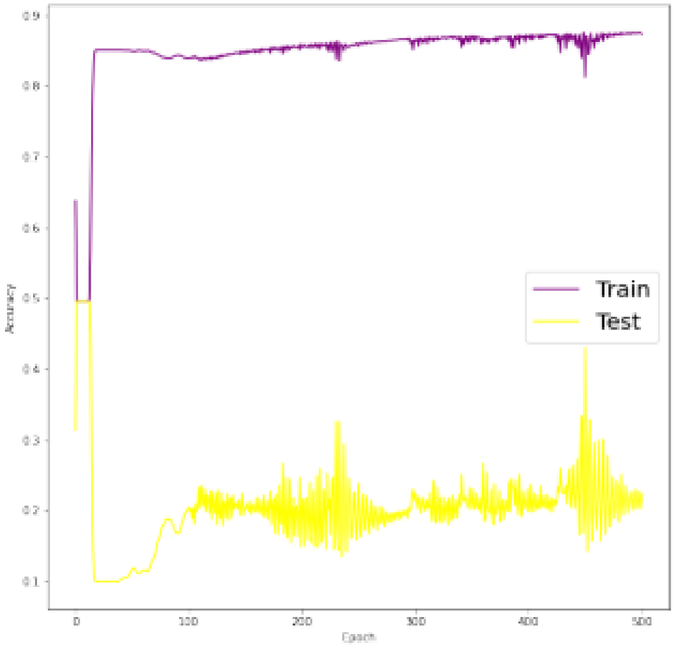


Loss - Epoch

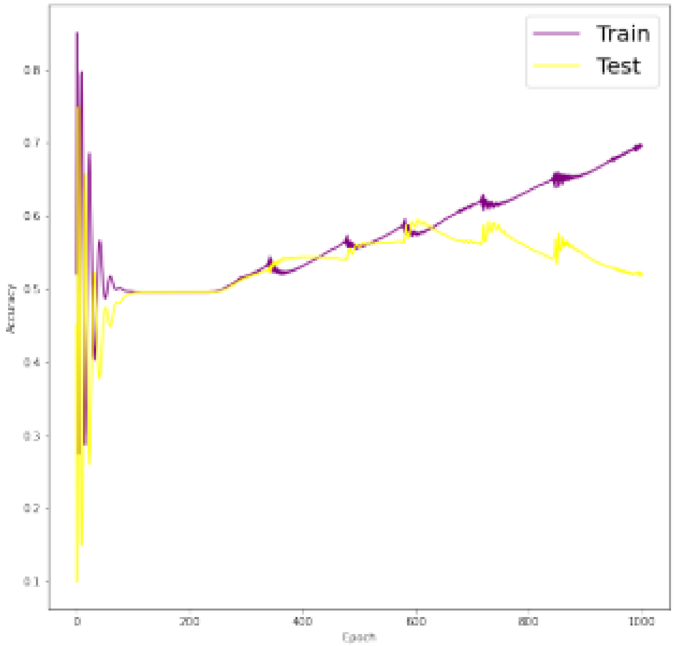


Accuracy - Epoch

Sensitiveness to Penalty Weight



(a) Penalty Weight $\lambda = 1.0$



(b) Penalty Weight $\lambda = 10000.0$

Figure: IRM learning curve with constant penalty weight

Self-Adaptive Optimization

$$L = \mu \sum_{f_i \in F} \sum_{k_j \in f} k_{i,j}^2 + \sum_{e \in E_{\text{train}}} R^e(F) + \lambda \|\nabla_{\boldsymbol{\omega}} R^e(\omega f_n)\|_{\omega=1.0}^2$$

$$\lambda = \frac{L - \mu \sum_{f_i \in F} \sum_{k_j \in f} k_{i,j}^2 - \sum_{e \in E_{\text{train}}} R^e(F)}{\|\nabla_{\boldsymbol{\omega}} R^e(\omega f_n)\|_{\omega=1.0}^2} \sim \frac{L_{\text{Train}}}{P_{\text{Train}}}$$

- Optimized penalty weight

$$\lambda = \begin{cases} \epsilon & t \leq \tau \\ \xi \dfrac{L_{\text{Train}}}{P_{\text{Train}}} & t > \tau \end{cases}$$

※ t: training step

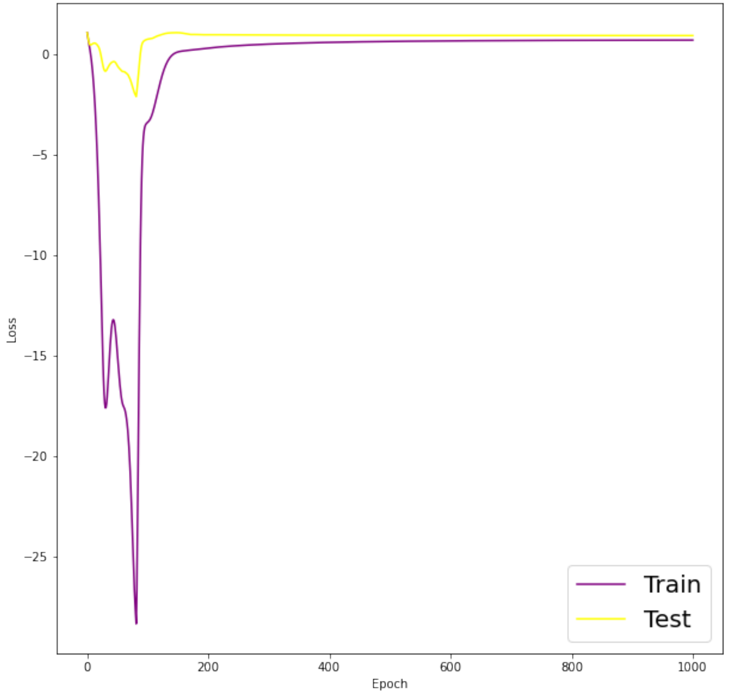
※ ϵ: a tiny value

※ ξ: unimportant coefficient

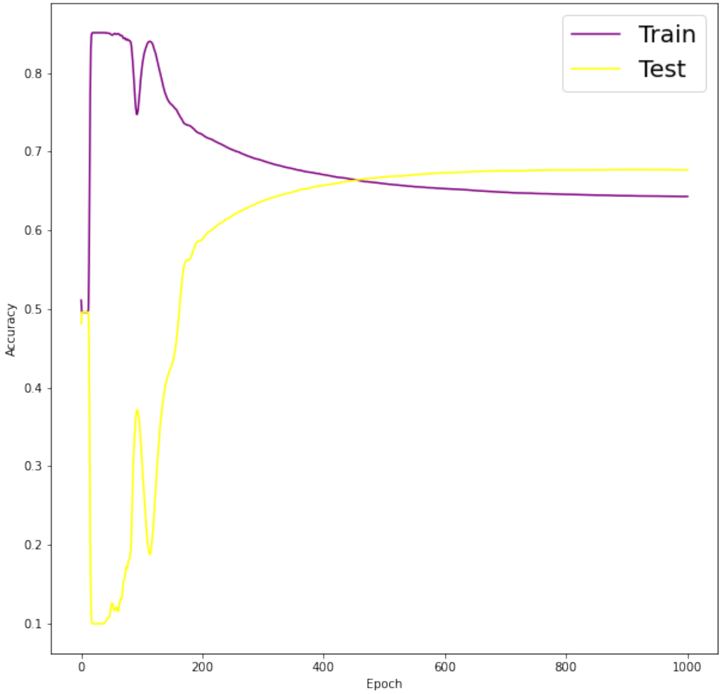
Self-Adaptive Optimization

- Optimized performance

Algorithm	Average Accuracy	Self-Adaptive
IRM in original paper	60.93%	No
IRM with **Self-Adaptive Optimization**	64.79%	Yes



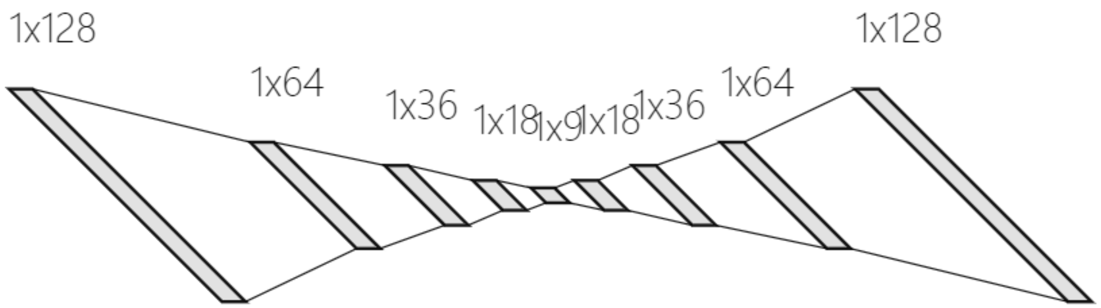
Loss - Epoch

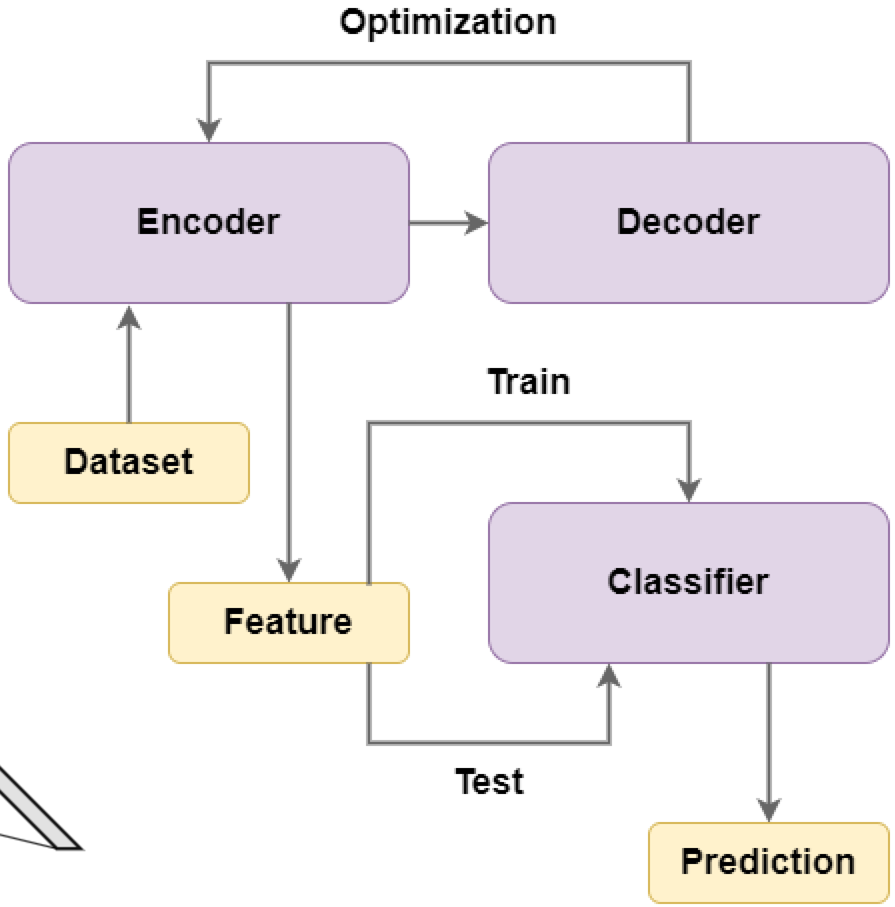


Accuracy - Epoch

Algorithm Exploration

- Unsupervised learning
  - feature extractor $\boldsymbol{\psi}$, $\boldsymbol{\varphi}$
  - classifier $\boldsymbol{\theta}$





※ Inspired by AutoEncoder

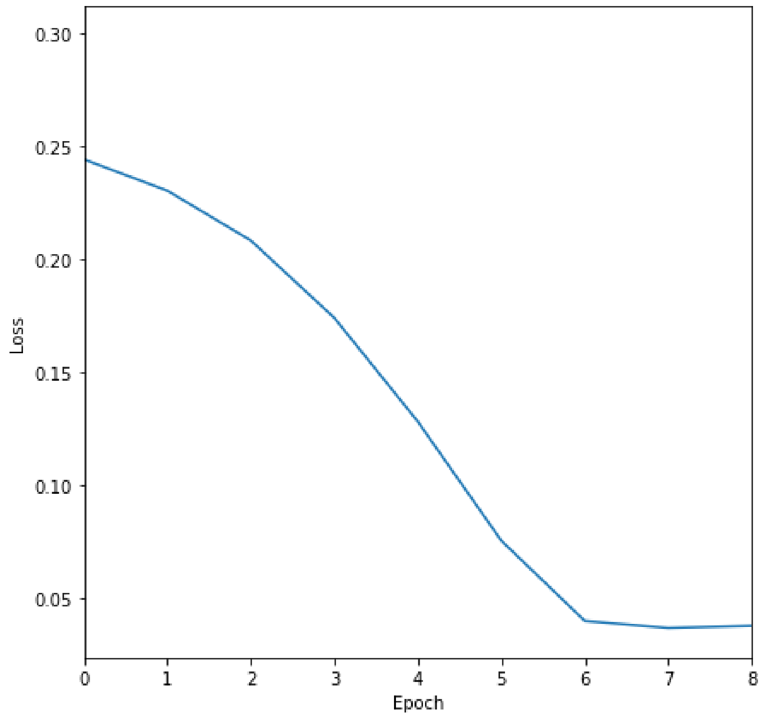
Algorithm Exploration

- Convergence analysis $L(\mathbf{\Phi}) = L_{\text{train}}(\mathbf{\Phi}) + \lambda L_{\text{test}}(\mathbf{\Phi})$

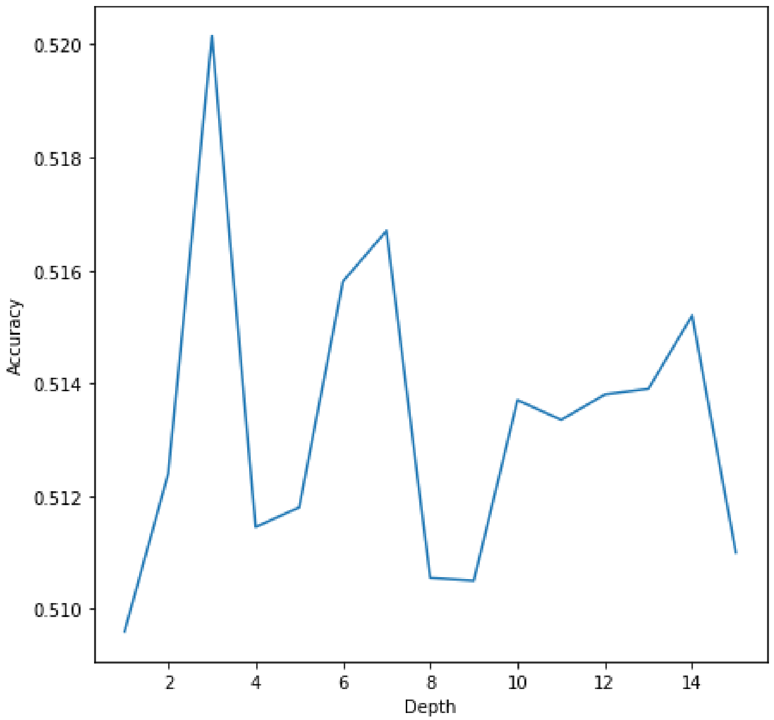
$$
\begin{aligned}
L(\mathbf{\Phi}) &= R(\boldsymbol{X'}_{\text{train}}, \boldsymbol{X}_{\text{train}}) + \lambda R(\boldsymbol{X'}_{\text{test}}, \boldsymbol{X}_{\text{test}}) \\
&= R(\mathbf{\Phi}(\boldsymbol{X}_{\text{train}}), \boldsymbol{X}_{\text{train}}) + \lambda R(\mathbf{\Phi}(\boldsymbol{X}_{\text{test}}), \boldsymbol{X}_{\text{test}}) \\
&= [\mathbf{\Phi}(\boldsymbol{X}_{\text{train}}) - \boldsymbol{X}_{\text{train}}]^T [\mathbf{\Phi}(\boldsymbol{X}_{\text{train}}) - \boldsymbol{X}_{\text{train}}] \\
&\quad + \lambda [\mathbf{\Phi}(\boldsymbol{X}_{\text{test}}) - \boldsymbol{X}_{\text{test}}]^T [\mathbf{\Phi}(\boldsymbol{X}_{\text{test}}) - \boldsymbol{X}_{\text{test}}] \\
&= \left[\mathbf{\Phi}\begin{pmatrix} \boldsymbol{X}_{\text{train}} \\ \sqrt{\lambda}\boldsymbol{X}_{\text{test}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{X}_{\text{train}} \\ \sqrt{\lambda}\boldsymbol{X}_{\text{test}} \end{pmatrix}\right]^T \left[\mathbf{\Phi}\begin{pmatrix} \boldsymbol{X}_{\text{train}} \\ \sqrt{\lambda}\boldsymbol{X}_{\text{test}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{X}_{\text{train}} \\ \sqrt{\lambda}\boldsymbol{X}_{\text{test}} \end{pmatrix}\right] \\
&= R\left(\mathbf{\Phi}\begin{pmatrix} \boldsymbol{X}_{\text{train}} \\ \sqrt{\lambda}\boldsymbol{X}_{\text{test}} \end{pmatrix}, \begin{pmatrix} \boldsymbol{X}_{\text{train}} \\ \sqrt{\lambda}\boldsymbol{X}_{\text{test}} \end{pmatrix}\right)
\end{aligned}
$$

Algorithm Exploration

- Experiment performance



Loss - Epoch



Accuracy - Depth

Conclusion

- IRM
  - sort out derivation and replenish some details
  - solve several troubles and implement algorithm
  - propose self-adaptive optimization
- Exploration
  - propose unsupervised feature extractor
  - analyze convergence of model
  - apply to experiment

References

- Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant Risk Minimization. arXiv:1907.02893, 2019.
- Elan Rosenfeld, Pradeep Ravikumar, and Andrej Risteski. The Risks of Invariant Risk Minimization. arXiv:2010.05761, 2020.
- Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-Adversarial Training of Neural Networks. arXiv:1505.07818, 2015.
- Puwei Dai. Explanation of DANN and GRL. https://zhuanlan.zhihu.com/p/109051269, 2021.
- Yearn. Invariant Risk Minimization Reading Notes. https://zhuanlan.zhihu.com/p/273209891, 2022.
- Martin Arjovsky et al. Code Repository for Invariant Risk Minimization. https://github.com/facebookresearch/InvariantRiskMinimization, 2020.