Chapter 6
Applications: Learning Predictive, Generative and Representation Models

Abstract In this chapter, we present applications of stochastic compositional optimization and finite-sum coupled compositional optimization (FCCO) in both supervised and self-supervised learning settings. These include training predictive models, generative models, and representation models based on advanced objective functions such as distributionally robust optimization (DRO), group DRO (GDRO), AUC losses, NDCG loss, and contrastive losses. We also highlight applications of compositional optimization in solving multiple inequality-constrained optimization problems, optimizing data compositional neural networks, and a new paradigm of learning with a reference model called DRRHO risk minimization.

Unity of knowledge and action!

Contents

Algorithm 23 Stochastic Optimization Framework of DL

// The Meta Algorithm
1: Set the learning rate schedule η_t
2: **for** $t = 1, \cdots, T$ **do**
3: Compute a vanilla gradient estimator $\mathbf{z}_t$
4: Update $\mathbf{w}_{t+1}$ by calling the update of SGD, Momentum, Adam, or AdamW optimizer
5: **end for**

// The SGD optimizer update
1: Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{z}_t$

// The Momentum optimizer update
1: Update $\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\mathbf{z}_t$ ⋄ the MA gradient estimator
2: Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{v}_t$

// The Adam optimizer update
1: Update $\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\mathbf{z}_t$ ⋄ the MA gradient estimator
2: Update $\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)(\mathbf{z}_t)^2$
3: Update $\hat{\mathbf{v}}_t = \mathbf{v}_t / (1-\beta_1^t)$
4: Update $\hat{\mathbf{s}}_t = \mathbf{s}_t / (1-\beta_2^t)$
5: Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{\hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t}+\epsilon}$ ϵ is a small constant

// The AdamW optimizer update
1: Update $\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\mathbf{z}_t$ ⋄ the MA gradient estimator
2: Update $\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)(\mathbf{z}_t)^2$
3: Update $\hat{\mathbf{v}}_t = \mathbf{v}_t / (1-\beta_1^t)$
4: Update $\hat{\mathbf{s}}_t = \mathbf{s}_t / (1-\beta_2^t)$
5: Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left(\frac{\hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t}+\epsilon} + \lambda \mathbf{w}_t \right)$ λ is a weight-decay constant

6.1 Stochastic Optimization Framework

For practioners who may skip Chapter 3, Chapter 4, and Chapter 5, we first provide a brief introduction to the stochastic optimization framework commonly used for deep learning. We also highlight the challenges in solving advanced machine learning problems introduced in Chapter 2 and summarize the key ideas behind the solution methods presented in Chapters 4 and 5.

The standard procedure for implementing a stochastic optimization algorithm typically involves computing a vanilla gradient estimator, followed by updating the model parameters using a step of an optimizer. We present a meta-algorithm in Algorithm 23, along with four classical optimizers: SGD, Momentum, Adam, and AdamW.

Three forms of the Momentum Method

The Momentum method represents a key milestone (as further discussed in the next subsection). The stochastic momentum method originates from the Heavy-ball (HB) method, whose stochastic version (SHB) has the following update for solving $\min_{\mathbf{w}} F(\mathbf{w}) := \mathbb{E}_{\zeta}[f(\mathbf{w};\zeta)]$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t;\zeta_t) + \beta_1(\mathbf{w}_t - \mathbf{w}_{t-1}), \tag{6.1}$$

where $\beta_1 \in (0,1)$ is the momentum parameter. While we utilize a single stochastic gradient $\nabla f(\mathbf{w}_t;\zeta_t)$ for illustrative purposes, practical applications generally rely on mini-batch estimation. In Section 4.3, we show it is equivalent to the the following update with moving average gradient estimator:

$$\begin{aligned} \mathbf{v}_t &= \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\nabla f(\mathbf{w}_t;\zeta_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta' \mathbf{v}_t, \end{aligned} \tag{6.2}$$

Update (6.1) is equivalent to (6.2) if $\eta'(1-\beta_1) = \eta$. In PyTorch, the Momentum method is implemented by the following update:

$$\begin{aligned} \mathbf{v}_t &= \beta_1 \mathbf{v}_{t-1} + \nabla f(\mathbf{w}_t;\zeta_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \mathbf{v}_t, \end{aligned} \tag{6.3}$$

which is equivalent to (6.1). One key insight from the convergence analysis of the Momentum method (6.2) (cf. Theorem 4.3) is that it ensures the averaged estimation error of the moving-average gradient estimators $\{\mathbf{v}_t\}$ converge to zero.

Thanks to well-developed deep learning frameworks such as PyTorch, implementing training code for deep neural networks has become relatively straightforward. The standard training pipeline is shown in Figure 6.1. The `Dataset` module allows us to get a training sample, which includes its input and output. The `Data Sampler` module (typically wrapped within the `DataLoader` module) provides tools to sample a mini-batch of examples for training at each iteration. The `Model` module allows us to define different deep models. The `Mini-batch Loss` module defines a loss function on the selected mini-batch data for backpropagation. The `Optimizer` module implements methods for updating the model parameter given the computed gradient from backpropagation. Most essential functions are already available in PyTorch. In practice, users often only need to define a function to compute their mini-batch losses. By calling `loss.backward()`, a mini-batch stochastic gradient, serving as a vanilla gradient estimator, is computed automatically.

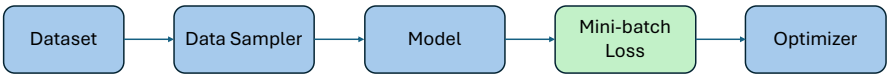


Fig. 6.1: Standard training pipeline for deep learning. Users typically only need to implement the mini-batch loss function. It relies on a critical assumption that the mini-batch stochastic gradient is an unbiased estimator of the true gradient

6.1.1 Milestones of Stochastic Optimization

While the Adam optimizer has become a standard in machine learning as of 2025, it has deep roots in the innovations of stochastic optimization before deep learning era. Below, we briefly discuss key milestones of stochastic optimization that have impact on the Adam method.

Stochasticity. The fundamental concept of gradient descent (GD), dating back to (Cauchy, 1847), uses the full dataset's gradient to take a step in the steepest direction. Introduced by Robbins and Monro (1951), SGD improves upon GD by using only a small batch of data (or even a single data point) to estimate the gradient, significantly speeding up training on large datasets.

Acceleration. To improve the convergence rate of GD, Polyak (1964) proposed the Heavy-ball (HB) method, which itself originates from the second-order Richardson method for solving a system of linear equations (Frankel, 1950). While Polyak only proved a faster rate of local convergence than GD for smooth and strongly convex problems, Nemirovski and Yudin (1977) proved the first nearly optimal rate for general smooth and strongly convex problems. Their method was inspired by the conjugate gradient method for solving quadratic problems and needs to solve 2-dimensional optimization problem using the method of centers of gravity every step; cf. (Nemirovsky and Yudin, 1983)[Sec. 7.3]. Later, Nesterov (1983) derived a simpler form of accelerated gradient method, which is now known as Nesterov's accelerated gradient (NAG) method.

Nesterov's Accelerated Gradient (NAG) method

The original update form of the NAG method is given by:

$$\begin{aligned} \mathbf{u}_{t+1} &= \mathbf{w}_t - \eta \nabla F(\mathbf{w}_t), \\ \mathbf{w}_{t+1} &= \mathbf{u}_{t+1} + \beta_1(\mathbf{u}_{t+1} - \mathbf{u}_t). \end{aligned} \tag{6.4}$$

It is equivalent to

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla F(\mathbf{w}_t) + \beta_1((\mathbf{w}_t - \eta \nabla F(\mathbf{w}_t)) - (\mathbf{w}_{t-1} - \eta \nabla F(\mathbf{w}_{t-1}))). \tag{6.5}$$

Comparing with the HB method (6.1), the momentum term is changed from $\beta(\mathbf{w}_t - \mathbf{w}_{t-1})$ to $\beta(\mathbf{u}_{t+1} - \mathbf{u}_t)$.
If we let $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{v}_t$, then the NAG update is equivalent to

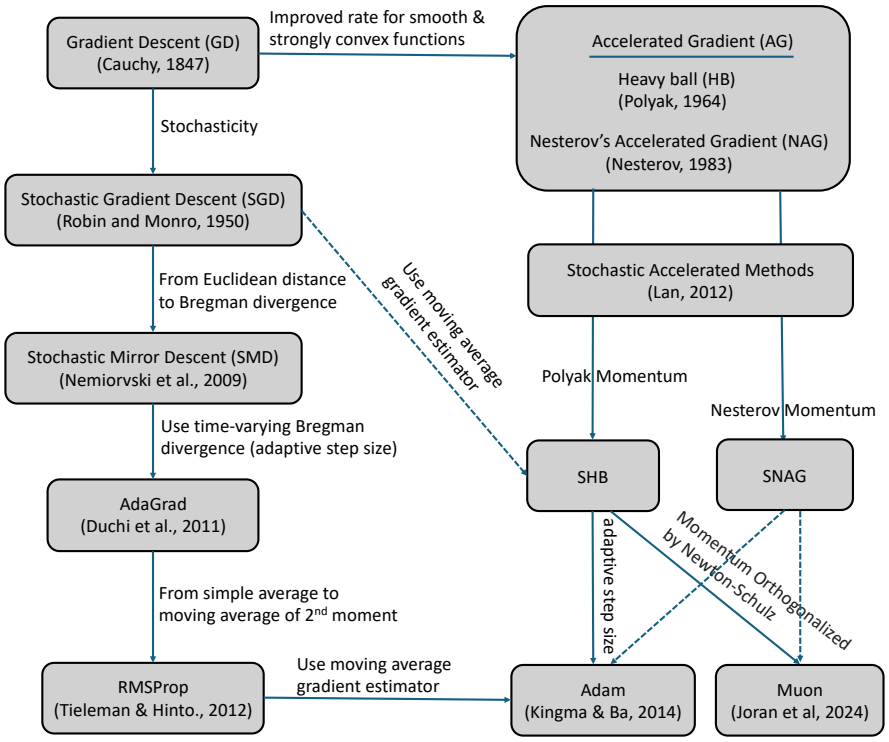


Fig. 6.2: Evolution of Stochastic Optimization

$$\begin{aligned}\mathbf{v}_t &= \beta_1 \mathbf{v}_{t-1} + \nabla F(\mathbf{w}_t) + \beta_1(\nabla F(\mathbf{w}_t) - \nabla F(\mathbf{w}_{t-1})) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \mathbf{v}_t .\end{aligned} \tag{6.6}$$

This is similar to (6.3) except an error correction term $\beta(\nabla F(\mathbf{w}_t) - \nabla F(\mathbf{w}_{t-1}))$ is added to the gradient estimator update.
We can also make the updates in (6.4) or (6.6) stochastic, leading to the stochastic NAG (SNAG) method. In particular, if we use a stochastic gradient estimator $\nabla f(\mathbf{w}_t; \zeta_t)$ in (6.4), we have the following update:

$$\begin{aligned}\mathbf{u}_{t+1} &= \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t; \zeta_t), \\ \mathbf{w}_{t+1} &= \mathbf{u}_{t+1} + \beta_1(\mathbf{u}_{t+1} - \mathbf{u}_t).\end{aligned} \tag{6.7}$$

In particular, if we use stochastic gradient estimators $\nabla f(\mathbf{w}_t; \zeta_t)$ and $\nabla f(\mathbf{w}_{t-1}; \zeta_t)$ in (6.6), we have the following update:

$$\begin{aligned} \mathbf{v}_t &= \beta_1 \mathbf{v}_{t-1} + \nabla f(\mathbf{w}_t; \zeta_t) + \beta_1 (\nabla f(\mathbf{w}_t; \zeta_t) - \nabla f(\mathbf{w}_{t-1}; \zeta_t)) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \mathbf{v}_t. \end{aligned} \tag{6.8}$$

The difference between the two variants lies that (6.8) needs to compute two stochastic gradient estimators at $\mathbf{w}_t$ and $\mathbf{w}_{t-1}$ per-iteration. However, interested readers can show that the update in (6.8) with a variable change is equivalent to the STORM update as presented in Section 4.3.2 for optimizing $F(\mathbf{w}) = \mathbb{E}_\zeta [f(\mathbf{w}; \zeta)]$.

Lan (2012) pioneered the development and analysis of stochastic accelerated gradient methods, achieving the optimal rates in both deterministic and stochastic regimes. Its update is slightly different from the NAG update. (Yang et al., 2016) is the first work to prove the convergence of stochastic NAG and stochastic HB methods for non-convex optimization.

Adaptive step sizes. The technique of utilizing coordinate-wise adaptive step sizes was pioneered by AdaGrad (Duchi et al., 2011), a method whose analysis is rooted in the framework of Stochastic Mirror Descent (SMD) (Nemirovski et al., 2009). Both AdaGrad and SMD are thoroughly examined in Chapter 3. RMSProp, appeared in a course lecture (Tieleman and Hinton, 2012), moved from AdaGrad's simple average of the second moment (squared gradients) to a moving average of the second moment. The moving average estimator has a long history in stochastic optimization, see (Ermoliev and Wets, 1988)[Sec. 6.2.3]. Finally, RMSProp leads to the current standard, the Adam method (Kingma and Ba, 2014), which combines the moving average of the first moment (similar to SHB) with the moving average of the second moment (similar to RMSProp). AdamW is a variant of Adam, which decouples weight decay from gradient-based updates.

Recently, a new optimizer named Muon (Jordan et al., 2024) has emerged, specifically designed to optimize matrix-structured parameters, such as the weight matrices between neural network layers. In contrast, conventional optimizers typically treat these parameters as flattened vectors, potentially overlooking their inherent structural properties.

The Muon method

Let W_t denote a matrix-structured parameter at the t-th iteration. The Muon update is given by:

$$\begin{aligned} M_t &= \beta_1 M_{t-1} + \nabla f(W_t; \zeta_t) \\ (U_t, S_t, V_t) &= \mathrm{SVD}(M_t) \\ W_{t+1} &= W_t - \eta_t U_t V_t^\top. \end{aligned} \tag{6.9}$$

In practice, the Singular Value Decomposition (SVD) is often replaced by a more computationally efficient Newton-Schulz matrix iteration. This process produces an approximate matrix $O_t = U_t S_t' V_t^\top$, where S_t' is diagonal with

$S'_t[i,i]' \sim \text{Uniform}(0.5, 1.5)$. The weight update is then applied as $W_{t+1} = W_t - \eta_t O_t$.

Summary: The evolution of stochastic optimization, which has had a major impact on modern AI (see Figure 6.2), can be characterized by five key shifts in algorithm design:

- From Full Gradient to Stochastic Gradient (**Batch Size**): Switched from using the full dataset's gradient (GD) to using noisy stochastic gradients (SGD) for faster iteration speed.
- From Gradient Descent to Accelerated Gradient Methods (**Momentum**): The optimization technique was enhanced by introducing a momentum term (like HB or NAG) to achieve an improved convergence rate for smooth convex functions, while still using the full gradient.
- From Euclidean Distance to Bregman Divergence (**Geometry**): Switched the underlying distance metric used for updates from the Euclidean distance to a Bregman divergence (SMD).
- From Static Step Size to Adaptive Step Size (**Preconditioning**): Switched from a constant or manually decaying learning rate to one that is scaled by past gradient magnitudes (AdaGrad).
- From a Mini-batch gradient estimator to a Moving Average gradient estimator (**Error reduction**): Switched from a simple mini-batch gradient estimator to a moving average gradient estimator (SHB, Adam).

6.1.2 Limitations of Existing Optimization Framework

The standard stochastic optimization algorithms and their analyses rest on a critical assumption: that the mini-batch stochastic gradient is an unbiased estimator of the true gradient. As discussed in Chapter 4, this assumption breaks down in the case of compositional functions of the form $f(g(\mathbf{w}))$, where f is a deterministic non-linear function and g is a stochastic function. In such cases, the gradient of the mini-batch loss $f(g(\mathbf{w}; \mathcal{B}))$, where $g(\mathbf{w}; \mathcal{B})$ is an unbiased estimator of $g(\mathbf{w})$ with a mini-batch $\mathcal{B}$, yields a biased estimate of the true gradient. Specifically, calling `loss.backward()` on the mini-batch loss will return a gradient of $\nabla f(g(\mathbf{w}; \mathcal{B}))\nabla g(\mathbf{w}; \mathcal{B})$, which is inherently biased. The method that directly uses this biased gradient estimator for SGD update is referred to as biased SGD (BSGD). However, since the estimation error is inversely proportional to the batch size, small batches can lead to large optimization errors. According to Lemma 2.1, such errors can negatively impact the generalization performance of the learned model.

To address this challenge, Chapters 4 and 5 introduce solution methods tailored to different families of compositional objectives. The key ideas underlying these algorithms concern (i) how the vanilla gradient estimator $\mathbf{z}_t$ is computed in Step 3 of Algorithm 23, and (ii) how the estimator error is further reduced through the use

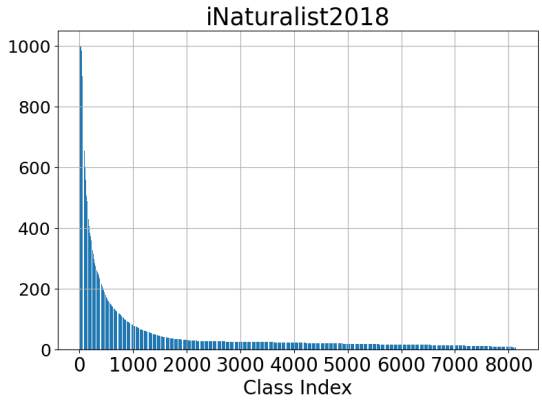


Fig. 6.3: Histograms of class sizes of the iNaturalist2018 dataset, which contains 437,513 natural images of 8,142 species. The sizes of classes follow a long-tail distribution.

of moving-average (MA) estimators $\mathbf{v}_t$ as in Step 1 of the Momentum optimizer or more advanced variance-reduction techniques. In the following sections, we will present their applications to various complex and advanced machine learning problems, with a focus on the presentation of the novel vanilla gradient estimators, which allow us to integrate them into the standard optimization schemes such as Momentum or AdamW for non-convex deep learning problems.

6.2 DRO and Group DRO

Let us consider supervised learning with a set of training data $\{(\mathbf{x}, y)\}$, where $\mathbf{x} \in \mathbb{R}^d$ denotes the input data and $y \in \{1, \ldots, K\}$ denotes the output class label. Let $\ell(\mathbf{w}; \mathbf{x}, y)$ denote the pointwise loss function, e.g., the cross-entropy loss.

6.2.1 DRO for Imbalanced Classification

Imbalanced classification is prevalent in many areas, including medicine and cybersecurity, where most training data may belong to one or a few classes. Mathematically, it means that the marginal distribution of the class label is a non-uniform distribution. An example of an imbalanced dataset is shown in Figure 6.3.

For imbalanced data, the conventional empirical risk minimization would focus on minimizing the loss of data from those dominating classes, neglecting data from the minority classes. DRO can address this issue by assigning larger weights to data

with higher losses. Let us first consider the KL-divergence regularized DRO:

$$\min_{\mathbf{w}} \max_{\mathbf{p}\in\Delta_n} \sum_{i=1}^n p_i \ell(\mathbf{w}; \mathbf{x}_i, y_i) - \tau \sum_{i=1}^n p_i \log(p_i n) + r(\mathbf{w}), \tag{6.10}$$

where $r(\mathbf{w})$ is a regularizer on $\mathbf{w}$. A traditional way to solve this problem is to use stochastic minimax optimization algorithms. However, there are several drawbacks of this approach: (1) the variance of stochastic gradient for $\mathbf{w}$ depends on the sampling distribution and the best sampling distribution depends on $\mathbf{p}$; (2) the sampling of data based on $\mathbf{p}$ incurs additional costs and is not friendly to practical implementation that uses random shuffling; (3) stochastic update of the dual variable $\mathbf{p}$ either takes $O(n)$ time complexity per iteration or requires maintaining a special tree structure to reduce the updating time to $O(\log(n))$.

To circumvent these issues, we consider an alternative formulation that is equivalent to the above minimax objective, i.e.,

$$\min_{\mathbf{w}} \tau \log\left(\frac{1}{n}\sum_{i=1}^n \exp\left(\frac{\ell(\mathbf{w}; \mathbf{x}_i, y_i)}{\tau}\right)\right) + r(\mathbf{w}). \tag{6.11}$$

For simplicity, we just consider the standard Euclidean norm regularization $r(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2$. As a result, the first term in the objective takes the form of a compositional optimization problem, namely $f\big(\mathbb{E}_\zeta\left[\, g(\mathbf{w}; \zeta)\,\right]\big)$, where $f(\cdot) = \tau\log(\cdot)$ and

$$\mathbb{E}_\zeta\left[(g(\mathbf{w}; \zeta)\right] = \frac{1}{n}\sum_{i=1}^n \exp\left(\frac{\ell(\mathbf{w}; \mathbf{x}_i, y_i)}{\tau}\right).$$

The SCGD, SCMA, SCST, and SCENT algorithms can be applied to solve the above problem. We now focus on the application of SCMA, whose key steps are presented in Algorithm 24.

The vanilla gradient estimator $\mathbf{z}_t$ of the first term in (6.11) at the t-th iteration is computed by :

$$\mathbf{z}_t = \frac{1}{B}\sum_{i\in\mathcal{B}_t} \frac{\exp(\frac{\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)}{\tau})}{u_t} \nabla\ell(\mathbf{w}_t; \mathbf{x}_i, y_i). \tag{6.12}$$

It is motivated from (4.4) where the same mini-batch $\mathcal{B}_t$ is used for both updating u_t and computing $\mathbf{z}_t$.

Let us compare this gradient estimator with that of stochastic optimization for empirical risk minimization:

$$\hat{\mathbf{z}}_t = \frac{1}{B}\sum_{i\in\mathcal{B}_t} \nabla\ell(\mathbf{w}_t; \mathbf{x}_i, y_i). \tag{6.13}$$

The difference between (6.12) and (6.13) lies in the blue term, which acts as a weight for each data in the mini-batch. In the vanilla gradient estimator $\mathbf{z}_t$ for DRO, the data

Algorithm 24 Attentional Biased Stochastic Methods

1: **for** $t = 1, \cdots, T$ **do**
2: Sample a mini-batch of B samples $\mathcal{B}_t \subset [n]$
3: Compute $g(\mathbf{w}_t, \mathcal{B}_t) = \frac{1}{B}\sum_{i\in\mathcal{B}_t} \exp(\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)/\tau)$
4: Compute $u_t = (1-\gamma)u_{t-1} + \gamma g(\mathbf{w}_t, \mathcal{B}_t)$
5: Compute the vanilla gradient estimator $\mathbf{z}_t = \frac{1}{B}\sum\limits_{i\in\mathcal{B}_t} \frac{\exp(\frac{\ell(\mathbf{w}_t;\mathbf{x}_i,y_i)}{\tau})}{u_t}\nabla\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)$
6: Update $\mathbf{w}_{t+1}$ by an optimizer such as Momentum or Adam-W
7: **end for**

in the mini-batch with a larger loss $\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)$ has a higher weight. This will facilitate the learning for data from the minority group. Due to this effect, we also refer to Algorithm 24 as attentional biased stochastic method, named as AB-xx depending on which optimizer is used.

The use of u_t for normalization to compute the weight $\exp(\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)/\tau)/u_t$ is also different from that using the heuristic mini-batch normalization where the weight is computed by $\frac{\exp(\ell(\mathbf{w}_t;\mathbf{x}_i,y_i)/\tau)}{\sum_{i\in\mathcal{B}_t}\exp(\ell(\mathbf{w}_t;\mathbf{x}_i,y_i)/\tau)}$, which does not ensure convergence if the batch size is not significantly large. Let us consider a simple case such that only one data is sampled for updating. In this case, the mini-batch normalization gives a weight 1 for the selected data no matter whether it is from the majority or minority class. However, if the sampled data denoted by $(\mathbf{x}_t, y_t)$ at the t-th iteration is from a minority group and hence has a large loss, we would like to penalize more on such an example. The estimator $u_t = (1-\gamma)u_{t-1} + \gamma\exp(\ell(\mathbf{w}_t; \mathbf{x}_t, y_t)/\tau)$ is likely to be smaller than $\exp(\ell(\mathbf{w}_t; \mathbf{x}_t, y_t)/\tau)$ as $\gamma < 1$. As a result, normalization using u_t will give a larger weight to the sampled minority data compared with using the mini-batch normalization, i.e., $\exp(\ell(\mathbf{w}_t; \mathbf{z}_t)/\tau)/u_t > 1$. Qi et al. (2020) empirically demonstrated that using $\gamma < 1$ outperforms the case $\gamma = 1$, which corresponds to using the standard mini-batch loss.

To illustrate the effect of AB-momentum on imbalanced data. We present an experiment on synthetic data in Figure 6.4, which compares the result of using the Momentum method for ERM and AB-momentum for solving KL-divergence regularized DRO. Figure 6.4(d) shows that AB-momentum learns a better decision boundary than that of the Momentum method for ERM. Figure 6.4(b) shows that data from the minority group that are close to the decision boundary get higher weights during the training.

Practical Tips

We discuss several practical tips for computing $\mathbf{z}_t$ and other variants of DRO in the context of deep learning.

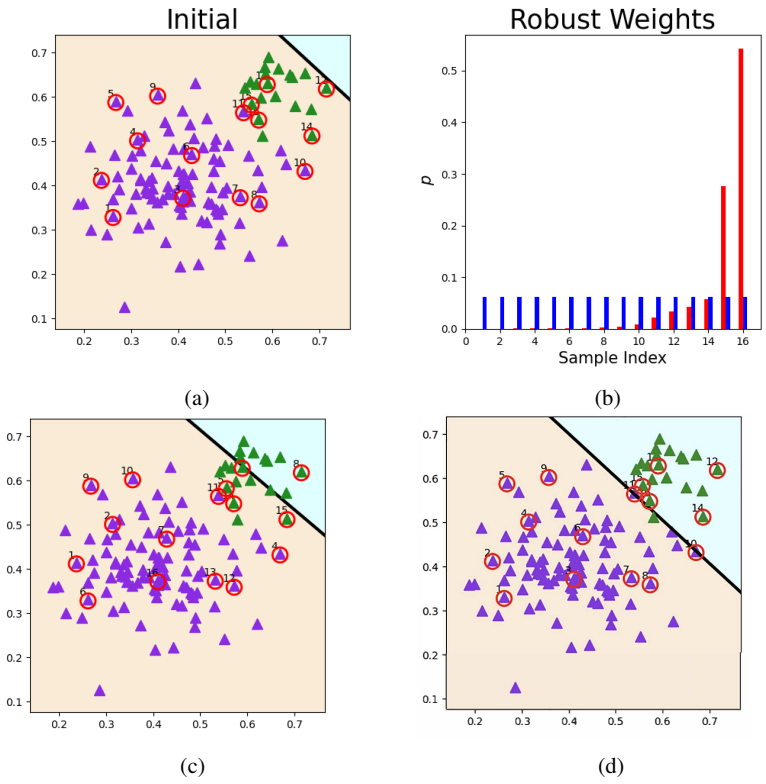


Fig. 6.4: **(a)**: A synthetic data for imbalanced binary classification (green vs purple) with a random linear decision boundary (black line). **(c), (d)**: Learned linear models optimized by the standard momentum method for ERM and AB-momentum for DRO with logistic loss for 100 iterations, respectively. **(b)**: The averaged weights of circled samples in the training process of the standard momentum method for ERM and AB-momentum method for DRO. Sample with indices in $\{1, \ldots, 11\}$ are from the majority class and samples with indices in $\{12, 13, 14, 15, 16\}$ are from the minority class with sample 15, 16 close to the decision boundary.

Backpropagation.

In order to compute the vanilla gradient estimator $\mathbf{z}_t$ using the PyTorch backward function, we just need to have a slight change of computing the loss based on the mini-batch data. Below we give the pseudo code in PyTorch for computing the gradient estimator highlighted in Step 5 of Algorithm 24. It is worth noting that the line of `p=(exp_loss/u).detach()` calculates the blue part and detaches it from the computational graph so that gradient is not computed again for it. With the gradient estimator computed by `loss.backward()`, then we can use any existing optimizers, including the Momentum method and AdamW.

```
sur_loss=surrogate_loss(preds, labels)
exp_loss = torch.exp(sur_loss/tau)
u = (1 - gamma)*u + gamma*(exp_loss.mean())
p = (exp_loss/u).detach()
loss = torch.mean(p * sur_loss)
loss.backward()
```

Avoiding the numerical issue.

However, a numerical issue may arise during the running tied to the computation of $\exp(\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)/\tau)$, especially when τ is small and the loss function of selected data is large so that overflow. As a result, the running of the algorithm may crash due to a NaN error. To address this issue, we maintain $\nu_t = \log u_t$. Specifically, we denote by $q_{t,i} = \exp(\frac{\ell(\mathbf{w}_t; \mathbf{x}_i, y_i) - \ell_{\max,t}}{\tau})$, where $\ell_{\max,t} = \max_{i \in \mathcal{B}_t} \ell(\mathbf{w}_t; \mathbf{x}_i, y_i)$. Then Step 4 can be reformulated to:

$$\begin{aligned}\exp(\log u_t) =& \exp(\log(1-\gamma) + \log u_{t-1}) \\ &+ \exp\left(\log\gamma + \log\left(\frac{1}{B}\sum\nolimits_{i\in\mathcal{B}_t} q_{t,i}\right) + \frac{\ell_{\max,t}}{\tau}\right).\end{aligned}$$

For simplicity, let $b_t = \log(1-\gamma) + \log u_{t-1}$ and $q_t = \log\gamma + \log\left(\frac{1}{B}\sum_{i\in\mathcal{B}_t} q_{t,i}\right) + \frac{\ell_{\max,t}}{\tau}$, we have

$$\exp(\log u_t) = \exp(b_t) + \exp(q_t).$$

The update in equivalent to following:

$$\begin{aligned}&\exp(\log u_t) = \exp(\max\{b_t, q_t\})(1 + \exp(-|b_t - q_t|)) \\ &= \exp(\max\{b_t, q_t\})\sigma^{-1}(|b_t - q_t|),\end{aligned}$$

where $\sigma(\cdot)$ denotes the sigmoid function. Taking the log on both sides gives the update for $\log u_t$. To summarize, we maintain and update $\nu_t = \log u_t$ as following:

$$\begin{aligned}b_t &= \log(1-\gamma) + \nu_{t-1} \\ q_t &= \log\gamma + \log\left(\frac{1}{B}\sum\nolimits_{i\in\mathcal{B}_t}\exp\left(\frac{\ell(\mathbf{w}_t; \mathbf{x}_i, y_i) - \ell_{\max,t}}{\tau}\right)\right) + \frac{\ell_{\max,t}}{\tau} \\ \nu_t &= \max\{b_t, q_t\} - \log\sigma(|b_t - q_t|).\end{aligned} \tag{6.14}$$

At the first iteration $t = 1$, we can just set

$$\nu_1 = \log\left(\frac{1}{B}\sum\nolimits_{i\in\mathcal{B}_t}\exp\left(\frac{\ell(\mathbf{w}_1; \mathbf{x}_i, y_i)}{\tau} - \frac{\ell_{\max,1}}{\tau}\right)\right) + \frac{\ell_{\max,1}}{\tau}.$$

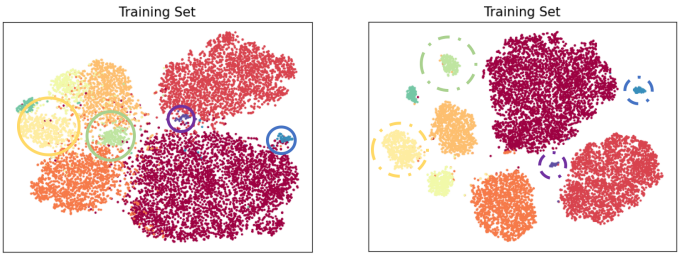


Fig. 6.5: t-SNE visualization of feature representations of training & testing set on CIFAR10-LT ($\rho = 100$) with different strategies of setting τ. Right: Fixed $\tau = 1$. Left: Two-stage decay of τ: first phase $\tau = 100$ and second phase $\tau = 1$. For more details, please refer to (Qi et al., 2020).

With ν_t, the effective weight $\frac{\exp(\ell(\mathbf{w}_t;\mathbf{x}_i,y_i)/\tau)}{u_t}$ can be computed by

$$\frac{\exp\left(\frac{\ell(\mathbf{w}_t;\mathbf{x}_i,y_i)}{\tau} - \max(\frac{\ell_{\max,t}}{\tau}, \nu_t)\right)}{\exp\left(\nu_t - \max(\frac{\ell_{\max,t}}{\tau}, \nu_t)\right)}.$$

Thus, all computation involving $\exp(\cdot)$ will not incure any numerical issue.

The Temperature parameter.

The last point we discuss here is how to set the value of the temperature parameter τ. A simple way is to treat it as a hyper-parameter and tune it based on cross-validation. However, there is a trade-off in the performance. A deep neural network is a hierarchical learner with lower layers for low-level feature extraction, middle layers for more abstract feature extraction and the last layer for classification. A larger τ indicates a more uniform weight, which is not good for learning the last classifier layer and minority class specific features. A smaller τ indicates a more non-uniform weight, which is not good for learning class agnostic lower level features.

One approach to mitigate this issue is to use a two-stage approach. In the first stage, we can use a relatively larger temperature τ for learning class agnostic lower level features. The second stage, we decrease τ to finetune the upper layers for learning robust minority-class specific features and classifier layer. An example is shown in Figure 6.5 on a long-tailed version of the CIFAR10 dataset, where the data is intentionally made imbalanced such that the number of samples per class follows a long-tail distribution, the imbalance ratio ρ means the ratio between sample sizes of the most frequent and least frequent classes.

Another approach is to treat τ as a parameter to be optimized. To achieve this, we can consider optimizing a KL-divergence constrained DRO:

$$\begin{aligned} &\min_{\mathbf{w}} \max_{\mathbf{p}\in\Delta_n} \sum_{i=1}^n p_i \ell(\mathbf{w}; \mathbf{x}_i, y_i) - \tau_0 \sum_{i=1}^n p_i \log(p_i n) + r(\mathbf{w}), \\ &\text{s.t.} \quad \sum_{i=1}^n p_i \log(p_i n) \le \rho, \end{aligned} \tag{6.15}$$

where the regularizer term with a small τ_0 is added to avoid ill conditioning, making the resulting problem smooth in terms of losses. Using the dual form of the maximization problem (see (2.19)), the above problem is equivalent to

$$\min_{\mathbf{w}, \tau \ge \tau_0} \tau \log\left(\frac{1}{n}\sum_{i=1}^n \exp\left(\frac{\ell(\mathbf{w}; \mathbf{x}_i, y_i)}{\tau}\right)\right) + \tau\rho. \tag{6.16}$$

We can extend Algorithm 24 to optimize the above problem by treating $(\mathbf{w}, \tau)$ as a single variable to be optimized. The vanilla gradient estimator in terms of τ at the t-th iteration is given by :

$$\mathbf{z}_{\tau,t} = \log(u_t) + \rho - \frac{1}{B}\sum_{i\in\mathcal{B}_t} \frac{\exp(\frac{\ell(\mathbf{w}_t;\mathbf{x}_i,y_i)}{\tau_t})}{u_t} \frac{\ell(\mathbf{w}_t; \mathbf{x}_i, y_i)}{\tau_t}.$$

6.2.2 GDRO for Addressing Spurious Correlation

Data may exhibit imbalance not in the marginal distribution of class label but some joint distribution of the class label and some attributes. Please see a discussion on the example of classifying waterbird images from landbirds images in Section 2.2.3. As a consequence, the model may learn spurious correlations between the labels and some attributes. GDRO can be used to mitigate this issue by leveraging prior knowledge of spurious correlations to define groups over the training data.

Formally, if there is spurious correlation between class label $y \in \mathcal{Y}$ and some attribute $a \in \mathcal{A}$, we can group the training data into $|\mathcal{Y}| \times |\mathcal{A}|$ groups according to the value of (y, a). Let $\mathcal{D}_i = \{(\mathbf{x}_{i,j}, y_{i,j})\}_{j=1}^{n_i}$ denote the data from the i-th group for $i \in \{1, \dots K\}$. Then we can define the averaged loss for data from each group i as $L_i(\mathbf{w}) = \frac{1}{n_i}\sum_{j=1}^{n_i} \ell(\mathbf{w}; \mathbf{x}_{i,j}, y_{i,j})$. Then, the GDRO formulation with CVaR divergence corresponding to the top-k groups is equivalent to (cf. (2.26)):

$$\min_{\mathbf{w},\nu} \frac{1}{K}\sum_{i=1}^K [L_i(\mathbf{w}) - \nu]_+ + \alpha\nu + \frac{\lambda}{2}\|\mathbf{w}\|_2^2, \tag{6.17}$$

where $\alpha = \frac{k}{K}$. If we define $\bar{\mathbf{w}} = (\mathbf{w}, \nu)$ and the inner functions as $g(\bar{\mathbf{w}}) = L_j(\mathbf{w}) - \nu$ and the outer function as $f(g) = [g]_+$, then the problem becomes an instance of non-smooth FCCO, where the outer function is non-smooth.

Algorithm 25 SONEX for solving (6.18)

Input: learning rate schedules $\{\eta_t\}_{t=1}^T$, $\{\gamma_t\}_{t=1}^T$; starting points $\mathbf{w}_1$, $\mathbf{u}_0$
for $t = 1, \ldots, T$ **do**
  Draw a batch of B_1 groups $\mathcal{B}_t \subset [K]$
  for $i \in \mathcal{B}_t$ **do**
    Draw B_2 samples $\zeta_{i,t}^j \sim \mathcal{D}_i$, $j = 1, \ldots, B_2$
    Update the inner function value estimators by

$$u_{i,t} = (1 - \gamma_t) u_{i,t-1} + \gamma_t \frac{1}{B_2} \sum_{j=1}^{B_2} \ell(\mathbf{w}_t; \mathbf{x}_{i,j}, y_{i,j})$$

  end for
  Set $u_{i,t+1} = u_{i,t}$, $i \notin \mathcal{B}_t$
  Compute the vanilla gradient of ν_t: $\mathbf{z}_{t,\nu} = -\frac{1}{B_1} \sum_{i \in \mathcal{B}_t} \nabla f_\varepsilon(u_{i,t} - \nu_t) + \frac{k}{K}$
  Compute the vanilla gradient of $\mathbf{w}_t$:

$$\mathbf{z}_{t,\mathbf{w}} = \frac{1}{B_1} \sum_{i \in \mathcal{B}_t} \left(\nabla f_\varepsilon(u_{i,t} - \nu_t) \frac{1}{B_2} \sum_{j=1}^{B_2} \nabla \ell(\mathbf{w}_t; \mathbf{x}_{i,j}, y_{i,j}) \right)$$

  update ν_{t+1} using SGD
  Update $\mathbf{w}_{t+1}$ using Momentum or AdamW
end for

An alternative way is to formulate the problem into an equivalent min-max formulation:

$$\min_{\mathbf{w}} \max_{\mathbf{p} \in \Delta, n p_i \leq 1/\alpha} \sum_{i=1}^K p_i L_i(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2. \tag{6.18}$$

However, solving this min-max problem has similar drawbacks as discussed in DRO, especially when the number of groups K is large.

Let us discuss the applicability of algorithms presented in Chapter 4 for solving (6.17). The theory of SOX and MSVR requires the smoothness of the outer functions, which is not applicable to GDRO. Both ALEXR and SONX are applicable as their analysis does not require the smoothness of the outer functions. However, their updates is SGD-type, which could make it slow or fail in practice for learning modern deep neural networks such as Transformer.

For deep learning applications, we can leverage SONEX. Its key idea is to smooth the outer hinge function. In particular, we define the smoothed hinge function as $f_\varepsilon(g)$ with a very small ε (cf. Example 5.1):

$$f_\varepsilon(g) = \max_{y \in [0,1]} yg - \frac{\varepsilon}{2} y^2 = \begin{cases} g - \frac{\varepsilon}{2} & \text{if } g \geq \varepsilon \\ \frac{g^2}{2\varepsilon} & \text{if } 0 < g < \varepsilon \\ 0 & \text{o.w.} \end{cases}.$$

As a result, we solve the following smoothed problem:

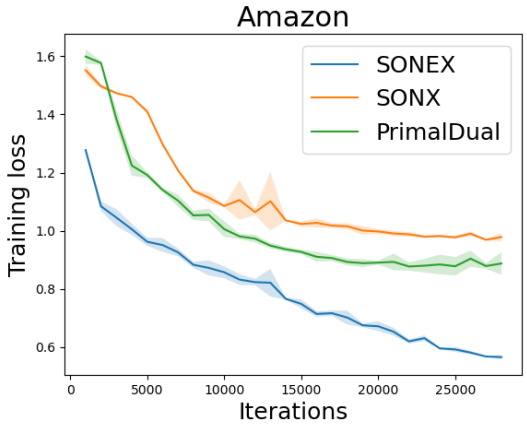


Fig. 6.6: An experimental comparison of different methods for solving GDRO (2.26) on the Amazon-WILDS dataset. The dataset is a text classification benchmark derived from Amazon product reviews, where the task is to predict binary sentiment (positive or negative) using TF–IDF features extracted from review text. The data spans multiple product categories. We construct groups based on the user attribute, resulting in 1,252 distinct groups. Only 4 groups and 64 data points per-group are sampled per-iteration. SONEX uses the Adam optimizer, SONX uses the SGD optimizer, and the PrimalDual is a stochastic primal-dual method for solving (6.18) that uses the Adam optimizer for the primal variable (model weights) and uses the stochastic mirror descent update for the dual variable $\mathbf{p}$ with a KL divergence. For more details, please refer to (Chen et al., 2025b).

$$\min_{\mathbf{w},\nu} \frac{1}{K}\sum_{j=1}^{K} f_\varepsilon(L_j(\mathbf{w}) - \nu) + \alpha\nu + \frac{\lambda}{2}\|\mathbf{w}\|_2^2. \tag{6.19}$$

We present a variant of SONEX in Algorithm 25. Figure 6.6 illustrates the effectiveness of SONEX for solving GDRO comprising with SONX and a stochastic primal-dual method.

6.3 Extreme Multi-class Classification

Multi-class classification is a cornerstone of machine learning. However, many modern applications involve an exceptionally large label space—ranging from millions to even billions of categories—a challenge known as extreme multi-class classification (XMC). For instance, for face recognition, the model learning is often formulated as classifying images into unique identities. With millions of distinct individuals, the model must navigate millions of corresponding classes. Similarly, when training a language model to predict the next word, the problem is treated as a multi-class classification task where each word in the vocabulary represents a category. Given that the English language contains over one million words, the resulting number of classes is immense.

Algorithm 26 The SCENT Algorithm for solving XMC

1: Initialize W_1, ν_0, step sizes η_t and α_t, $\varphi(\nu) = e^{-\nu}$.
2: **for** $t = 1 \ldots, T-1$ **do**
3: Sample a mini-batch data $\mathcal{B}_t \subset \{1, \ldots, n\}$ with $|\mathcal{B}_t| = B$
4: Let C_t denote the set of unique labels in $\mathcal{B}_t$
5: **for** each $(\mathbf{x}_i, y_i) \in \mathcal{B}_t$ **do**
6: Update $\nu_{i,t}$ by solving

$$\nu_{i,t} = \arg\min_{\nu} \frac{1}{|\mathcal{B}_t| - 1} \sum_{y_j \in \mathcal{B}_t \setminus y_i} \exp((\mathbf{w}_{t,y_j} - \mathbf{w}_{t,y_i})^\top h(\mathbf{x}_i) - \nu) + \nu + \frac{1}{\alpha_t} D_\varphi(\nu, \nu_{i,t-1})$$

7: **end for**
8: Compute $\mathbf{Z}_t[C_t] = \nabla L_t(W_t[C_t])$ by calling backprop on the mini-batch loss

$$L_t(W_t[C_t]) = \frac{1}{B} \sum_{i \in \mathcal{B}_t} \frac{1}{|\mathcal{B}_t| - 1} \sum_{y_j \in \mathcal{B}_t \setminus y_i} \exp((\mathbf{w}_{t,y_j} - \mathbf{w}_{t,y_i})^\top h(\mathbf{x}_i) - \nu_{i,t})$$

9: Compute $\mathbf{V}_t[C_t] = (1 - \beta_t)\mathbf{V}_{t-1}[C_t] + \beta_t \mathbf{Z}_t[C_t]$ (optional)
10: Update $W_{t+1}[C_t] = W_t[C_t] - \eta_t \mathbf{V}_t[C_t]$
11: **end for**

A dominating approach of multi-class classification is logistic regression, which minimizes the cross-entropy loss. Let us consider learning a linear model by solving the following problem:

$$\min_W \frac{1}{n} \sum_{i=1}^{n} -\log \frac{\exp(\mathbf{w}_{y_i}^\top h(\mathbf{x}_i))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top h(\mathbf{x}_i))}$$

where $y_i \in \{1, \ldots, K\}$ denotes the true class label of $\mathbf{x}_i$, $W = (\mathbf{w}_1, \ldots, \mathbf{w}_K) \in \mathbb{R}^{d \times K}$ contains the weights for all classes, and $h(\mathbf{x}) \in \mathbb{R}^d$ denotes the feature vector of each data. When K is huge, it is not efficient to compute the normalization term $\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top h(\mathbf{x}_i))$ for each data and loading all W into the memory might be prohibited.

To solve this problem, we can use SCENT algorithm presented in Section 5.5.2. To this end, we reformulate the problem into the following equivalent min-min optimization:

$$\min_W \min_\nu \frac{1}{n} \sum_{i=1}^{n} \left\{ \frac{1}{K} \sum_{j=1}^{K} \exp(\mathbf{w}_j^\top h(\mathbf{x}_i) - \mathbf{w}_{y_i}^\top h(\mathbf{x}_i) - \nu_i) + \nu_i - 1 \right\}.$$

We present an application of SCENT for solving this problem in Algorithm 26. At each iteration, the algorithm begins by sampling a mini-batch $\mathcal{B}_t$ (Step 3) to approximate the outer summation over n data points. Following this, the algorithm updates the dual variables ν_i for each $i \in \mathcal{B}_t$. While the original SCENT algorithm requires sampling from the full set of classes $\{j = 1, \ldots, K\}$, we observe that for all sampled data, the weights corresponding to their true labels $\{\mathbf{w}_{y_i} : i \in \mathcal{B}_t\}$ must already

be accessed. Consequently, we utilize the 'in-batch' class labels to approximate the inner summation, setting $\mathcal{Y}_t = \{\{y_i\}\}_{i\in\mathcal{B}_t}$ be the multiset of labels and C_t to the set of unique labels in $\mathcal{B}_t$. To update $\boldsymbol{\nu}_t$ and W_t, the following calculations are implemented.

- **Computing Sampled and Shifted Logits.** Given the mini-batch $\mathcal{B}_t$ and the set of sampled classes $\mathcal{Y}_t$, we first compute the inner products between the features $h(\mathbf{x}_i)$ and class weights $\mathbf{w}_j$ for all $i \in \mathcal{B}_t$ and $j \in \mathcal{Y}_t$. This is efficiently computed via the matrix product $Q = H[\mathcal{B}_t]^\top W[\mathcal{Y}_t] \in \mathbb{R}^{B\times|\mathcal{Y}_t|}$, where $H[\mathcal{B}_t] = [h(\mathbf{x}_i)]_{i\in\mathcal{B}_t}$ represents the sampled feature matrix. We then derive the shifted logits matrix R, defined by the entries $R_{ij} = \mathbf{w}_j^\top h(\mathbf{x}_i) - \mathbf{w}_{y_i}^\top h(\mathbf{x}_i)$ for all $i \in \mathcal{B}_t, j \in \mathcal{Y}_t$.
- **Closed-form update for $\nu_{i,t}$.** Given the shifted logits matrix R, we update the state variable $\nu_{i,t}$ according to Lemma 5.26:

$$\nu_{i,t} = \nu_{i,t-1} + \log\left(1 + \alpha_t \frac{1}{|\mathcal{Y}_t| - 1} \sum_{j\in\mathcal{Y}_t\setminus y_i} \exp(R_{ij})\right) - \log(1 + \alpha_t e^{\nu_{i,t-1}}),$$

  where we treat the labels in $\mathcal{Y}_t \setminus y_i$ as independent samples from $\{1, \ldots, K\}$. To ensure numerical stability when $\nu_{i,t-1}$ or R_{ij} are large, we apply standard logarithmic identities. Specifically, while $\nu_{i,t-1}$ typically remains within a stable range, the term $\log(1+\alpha_t e^{\nu_{i,t-1}})$ can be computed as $\nu_{i,t-1}+\log(e^{-\nu_{i,t-1}}+\alpha_t)$ for large positive values of $\nu_{i,t-1}$. Furthermore, we stabilize the second term using the Log-Sum-Exp trick by shifting the exponents by $R_{i,\max} = \max_{j\in\mathcal{Y}_t\setminus y_i} R_{ij}$:

$$\begin{aligned}
&\log\left(1 + \frac{\alpha_t}{|\mathcal{Y}_t| - 1} \sum_{j\in\mathcal{Y}_t\setminus y_i} \exp(R_{ij})\right) \\
&= \log\left(\exp(-R_{i,\max}) + \frac{\alpha_t}{|\mathcal{Y}_t| - 1} \sum_{j\in\mathcal{Y}_t\setminus y_i} \exp(R_{ij} - R_{i,\max})\right) + R_{i,\max}.
\end{aligned}$$

- **Updating $W_t[C_t]$.** Finally, the gradient of $W_t[C_t]$ is computed by performing backpropagation on the mini-batch loss $L_t(W_t[C_t])$. Because the loss function is defined only over the sampled classes, the gradient updates are sparse and operate exclusively on the sampled subset $W_t[C_t]$. This approach eliminates the need to load the entire weight matrix W into the main memory, significantly reducing the memory overhead in hardware-constrained environments.

Empirical Comparison with baselines

An empirical study demonstrating the effectiveness of SCENT for XMC is presented in Figure 6.7, which compares Algorithm 26 with ASGD, BSGD, and the SOX method. The key differences between these methods and Algorithm 26 are as follows: (i) SOX is closely related to SCENT, but uses a step size $\alpha_{i,t} = \gamma e^{-\nu_{i,t-1}}$ when

updating $\nu_{i,t}$; (ii) ASGD employs a standard stochastic coordinate update for the dual variables $\boldsymbol{\nu}$; and (iii) BSGD simply computes the gradient of $W_t[C_t]$ using the following mini-batch loss:

$$\frac{1}{B}\sum_{i\in\mathcal{B}_t} -\log\frac{\exp(\mathbf{w}_{y_i}^\top h(\mathbf{x}_i))}{\sum_{j\in\mathcal{Y}_t\setminus y_i}\exp(\mathbf{w}_j^\top h(\mathbf{x}_i))}.$$

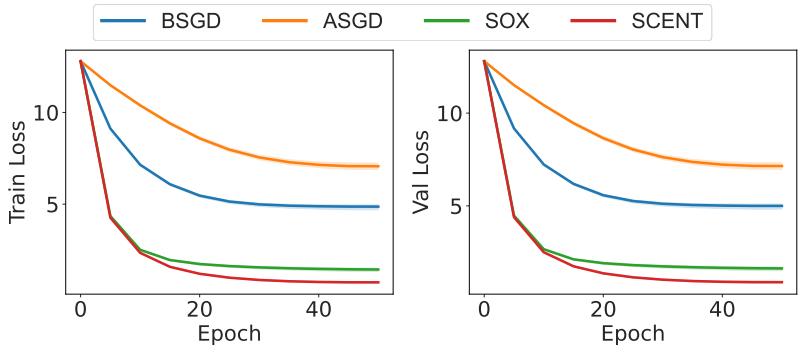


Fig. 6.7: Left: training curve on Glint360K dataset. Right: accuracy curve on the validation data. The Glint360K dataset (An et al., 2021) is a face recognition dataset consisting of 17 million images of 360 thousand individuals (i.e., 360K classes). To obtain the features for linear classification, we leverage a pretrained ResNet-50 model. For all the methods, we use a batch size of 1024 and update the model weights for 50 epochs using the SGD optimizer (no momentum). We tune the learning rate of W for all methods and decrease it in a cosine manner during training. For ASGD, SOX and SCENT, the learning rate of the $\boldsymbol{\nu}$ update is also tuned. For more details, please refer to (Wei et al., 2026).

6.4 Stochastic AUC and NDCG Maximization

In many domains such as radiology and drug discovery, areas under the curves are commonly used to assess the performance of a predictive model. In domains that involve ranking or recommendation, normalized discounted cumulative gain (NDCG) is commonly used as a performance metric. We present applications of SCO and FCCO algorithms for optimizing these metrics directly.

6.4.1 Stochastic AUC Maximization

In this section, we focus on optimizing the area under ROC curve (AUC) for binary classification as depicted in Figure 2.3.

Method 1: Pairwise Loss Minimization

The training data consists of $\{\mathbf{x}_i, y_i\}_{i=1}^n$, where $\mathbf{x} \in \mathbb{R}^d$ is the input and $y \in \{1, -1\}$ is the binary label. The traditional surrogate objective for AUC maximization is the pairwise loss given in (2.31). To optimize the pairwise surrogate objective, we just need to sample positive and negative data and then define a mini-batch pairwise loss:

$$\frac{1}{|\mathcal{B}_+|}\sum_{\mathbf{x}_i \in \mathcal{B}_+} \frac{1}{|\mathcal{B}_-|}\sum_{\mathbf{x}_j \in \mathcal{B}_-} \ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)).$$

Calling backpropagation on this mini-batch pairwise loss gives an unbiased stochastic gradient estimator. Then any appropriate optimizer can be leveraged to update the model. This is same as the conventional algorithm except for that the data sampler needs to sample both positive and negative data (see Section 6.4.5).

A limitation of this approach is that it increases the communication costs of distributed training when data are distributed across different machines as it requires to form positive-negative pairs across different machines.

Method 2: Minimax Optimization

The second approach is to solve the formulation as in (2.32). To illustrate the algorithm, we give its formulation below:

$$\begin{aligned}\min_{\mathbf{w}\in\mathbb{R}^d,(a,b)\in\mathbb{R}^2} \quad & \frac{1}{|\mathcal{S}_+|}\sum_{\mathbf{x}_i \in \mathcal{S}_+} (h(\mathbf{w}; \mathbf{x}_i) - a)^2 + \frac{1}{|\mathcal{S}_-|}\sum_{\mathbf{x}_j \in \mathcal{S}_-} (h(\mathbf{w}; \mathbf{x}_j) - b)^2 \\ & + f\left(\frac{1}{|\mathcal{S}_-|}\sum_{\mathbf{x}_j \in \mathcal{S}_-} h(\mathbf{w}; \mathbf{x}_j) - \frac{1}{|\mathcal{S}_+|}\sum_{\mathbf{x}_i \in \mathcal{S}_+} h(\mathbf{w}; \mathbf{x}_i)\right),\end{aligned} \tag{6.20}$$

where $h(\mathbf{w}; \cdot) \in \mathbb{R}$ is the prediction output of the model for any input, $\mathcal{S}_+$ is the set of positive data and $\mathcal{S}_-$ is the set of negative data and f is a non-decreasing surrogate function.

Let us illustrate the algorithm for a squared-hinge surrogate function $f(s) = \max(m + s, 0)^2$, where $m > 0$ is a margin parameter. Since f is non-linear, the last term of the above objective function is a compositional function of the form $f(g)$, where $g(\mathbf{w}) = \frac{1}{|\mathcal{S}_-|}\sum_{\mathbf{x}_j \in \mathcal{S}_-} h(\mathbf{w}; \mathbf{x}_j) - \frac{1}{|\mathcal{S}_+|}\sum_{\mathbf{x}_i \in \mathcal{S}_+} h(\mathbf{w}; \mathbf{x}_i)$. We consider the minimax reformulation similar to (5.27). In particular, using the conjugate of $f(\cdot)$

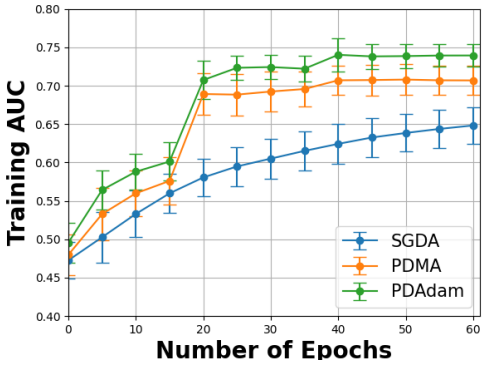


Fig. 6.8: Comparison between PDMA/PDAdam and SGDA for solving (6.21) of AUC maximization. The dataset is BBBP whose task is to predict whether a drug can penetrate the blood-brain barrier to arrive the targeted central nervous system or not. For more details, please refer to (Guo et al., 2021b).

(see Example 1.12), we convert the above minimization problem into a minimax optimization problem:

$$\min_{\mathbf{w},a,b}\max_{\alpha\geq 0} F(\mathbf{w},a,b;\alpha) := \frac{1}{|\mathcal{S}_+|}\sum_{\mathbf{x}_i\in\mathcal{S}_+}(h(\mathbf{w};\mathbf{x}_i)-a)^2 + \frac{1}{|\mathcal{S}_-|}\sum_{\mathbf{x}_j\in\mathcal{S}_-}(h(\mathbf{w};\mathbf{x}_j)-b)^2 + \alpha\left(m+\frac{1}{|\mathcal{S}_-|}\sum_{\mathbf{x}_j\in\mathcal{S}_-}h(\mathbf{w};\mathbf{x}_j)-\frac{1}{|\mathcal{S}_+|}\sum_{\mathbf{x}_i\in\mathcal{S}_+}h(\mathbf{w};\mathbf{x}_i)\right)-\frac{\alpha^2}{4}, \tag{6.21}$$

Compared to pairwise loss minimization, the advantage of the above minimax formulation is that its objective is decomposable over individual data points, making it well-suited for distributed training.

We present a practical framework in Algorithm 27 built from SMDA for solving the above problem, where the primal-dual Momentum method (PDMA) employs the momentum update for the primal variable $\bar{\mathbf{w}}$ or a primal-dual Adam method (PDAdam) employs the Adam update for the primal variable. The effectiveness of PDMA/PDAdam over SGDA for solving (6.21) on a real-world dataset is shown in Figure 6.8.

Squared-hinge surrogate vs Square surrogate function

The minimax optimization framework (6.20) and PDMA/PDAdam algorithms with a small modification on the dual variable update can handle any smooth surrogate function f. When $f(s) = (m+s)^2$ is a square surrogate, the minimax formulation is equivalent to the pairwise loss minimization with a square surrogate loss (AUC square loss). Nevertheless, the minimax AUC margin loss with the squared-hinge surrogate is more robust than the AUC square loss. Figure 6.9 illustrates the robustness of the minimax AUC margin loss.

Algorithm 27 PDMA or PDAdam for solving (6.21)

1: **Input:** learning rate schedules η_t, τ_t; starting points $\bar{\mathbf{w}}_1 = (\mathbf{w}_1, a_1, b_1)$, α_1
2: **for** $t = 1, \ldots, T$ **do**
3: Draw B_1 positive data $\mathcal{B}_t^+ \subset \mathcal{S}_+$ and B_2 negative data $\mathcal{B}_t^- \subset \mathcal{S}_-$
4: Update $\alpha_{t+1} = \left[(1 - \tau_t/2)\alpha_t + \tau_t \left(m + \frac{1}{B_2}\sum_{\mathbf{x}_j \in \mathcal{B}_t^-} h(\mathbf{w}; \mathbf{x}_j) - \frac{1}{B_1}\sum_{\mathbf{x}_i \in \mathcal{B}_t^+} h(\mathbf{w}_t; \mathbf{x}_i)\right)\right]_+$
5: Compute the vanilla gradient estimator

$$\mathbf{z}_t = \frac{1}{B_1}\sum_{i \in \mathcal{B}_t^+} \nabla_{\bar{\mathbf{w}}}(h_{\mathbf{w}_t}(\mathbf{x}_i) - a_t)^2 + \frac{1}{B_2}\sum_{\mathbf{x}_j \in \mathcal{B}_t^-} \nabla_{\bar{\mathbf{w}}}(h(\mathbf{w}_t; \mathbf{x}_j) - b_t)^2$$
$$+ \alpha_t \nabla_{\bar{\mathbf{w}}}\left(\frac{1}{B_2}\sum_{\mathbf{x}_j \in \mathcal{B}_t^-} h(\mathbf{w}; \mathbf{x}_j) - \frac{1}{B_1}\sum_{\mathbf{x}_i \in \mathcal{B}_t^+} h(\mathbf{w}_t; \mathbf{x}_i)\right)$$

6: Update $\bar{\mathbf{w}}_{t+1}$ by Momentum or AdamW
7: **end for**

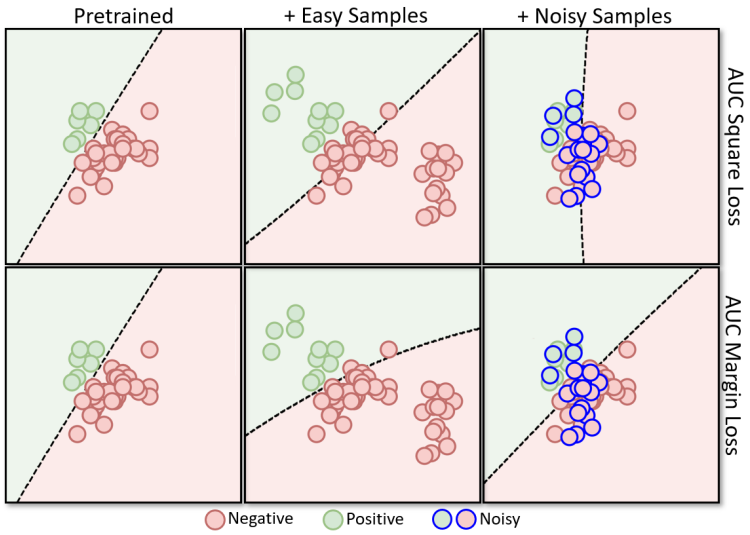


Fig. 6.9: An illustrative example for optimizing different AUC losses on a toy data for learning a two-layer neural network with ELU activation. The top row is optimizing the AUC square loss and the bottom row is optimizing the new AUC margin loss as in (6.21). The first column depicts the initial decision boundary (dashed line) pre-trained on a set of examples. In the middle column, we add some easy examples to the training set and retrain the model by optimizing the AUC loss. In the last column, we add some noisily labeled data (blue circled data) to the training set and retrain the model by optimizing the AUC loss. The results demonstrate the AUC margin loss is more robust than the AUC square loss.

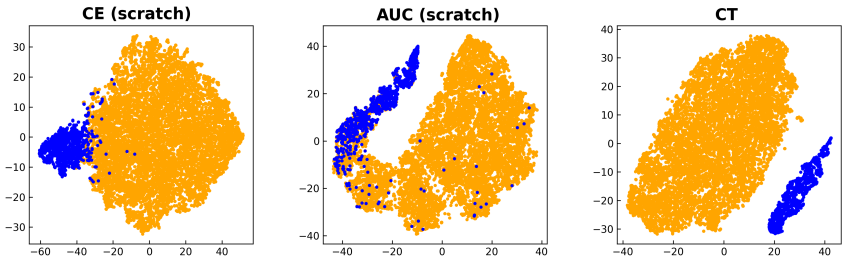


Fig. 6.10: t-SNE visualization of feature representations of an imbalanced training set for the Cat vs Dog visualized by t-SNE learned by different methods (from left to right): optimizing CE loss, an AUC loss, and a compositional training (CT) objective. For more details, please refer to (Yuan et al., 2022a).

Feature Learning

Feature learning is an important capability of deep learning. However, like the DRO objective, the end-to-end training based on the AUC surrogate objective does not favor feature learning as compared with traditional ERM. The reason is that AUC surrogate objective gives unequal weights to different data points due to the imbalance of training data. To address this challenge, one way is to employ a two-stage approach, where the first stage pretrains the encoder network on the training data by traditional supervised learning (e.g., ERM with the CE loss) or self-supervised representation learning and the second stage fine-tunes the feature extration layers and a random initialized classifier layer by optimizing an AUC surrogate objective.

An approach for performing effective feature learning and AUC maximization in a unified framework is to optimize a compositional objective (Yuan et al., 2022a):

$$\min_{\mathbf{w},a,b} \max_{\alpha \geq 0} F(\mathbf{w} - \tau \nabla L_{\text{CE}}(\mathbf{w}), a, b; \alpha),$$

where $L_{\text{CE}}(\mathbf{w})$ is the empirical risk based on the CE loss and $\tau > 0$ is a hyper-parameter.

To understand this compositional objective intuitively, let us take a thought experiment by using a gradient descent method to optimize the compositional objective. To this end, we denote the objective by $L_{\text{AUC}}(\mathbf{w} - \tau \nabla L_{\text{CE}}(\mathbf{w}))$, where L_{AUC} denotes the AUC surrogate objective. First, we evaluate the inner function by $\mathbf{u} = \mathbf{w} - \alpha \nabla L_{\text{CE}}(\mathbf{w})$. We can see that $\mathbf{u}$ is computed by a gradient descent step for minimizing the empirical risk $L_{\text{CE}}(\mathbf{w})$, which facilitates the learning of lower layers for feature extraction due to equal weights of all examples. Then, we take a gradient descent step to update $\mathbf{w}$ for minimizing the outer function $L_{\text{AUC}}(\cdot)$ by using the gradient $\nabla L_{\text{AUC}}(\mathbf{u})$ instead of $\nabla L_{\text{AUC}}(\mathbf{w})$. Because $\mathbf{u}$ is better than $\mathbf{w}$ in terms of feature extraction layers, taking a gradient descent step using $\nabla L_{\text{AUC}}(\mathbf{u})$ would be better than using $\nabla L_{\text{AUC}}(\mathbf{w})$. In addition, taking a gradient descent step for the outer function $L_{\text{AUC}}(\cdot)$ will make the classifier more robust to the minority class due to use of the AUC surrogate loss. Overall, we have two alternating conceptual steps, i.e., the inner gradient descent

step $\mathbf{u} = \mathbf{w} - \tau\nabla L_{\text{CE}}(\mathbf{w})$ acts as a feature purification step, and the outer gradient descent step $\mathbf{w} - \eta(I - \tau\nabla^2 L_{\text{CE}}(\mathbf{w}))\nabla L_{\text{AUC}}(\mathbf{u})$ acts as a classifier robustification step, where η is a step size.

For practical implementation, the intermediate model $\mathbf{w} - \tau\nabla L_{\text{CE}}(\mathbf{w})$ can be tracked by the MA estimator $\mathbf{u}_t = (1-\gamma)\mathbf{u}_{t-1} + \gamma(\mathbf{w}_t - \tau\nabla\hat{L}_{\text{CE}}(\mathbf{w}_t))$, where $\hat{L}_{\text{CE}}$ is a mini-batch CE loss. Then, $\mathbf{u}_t$ is used to update the primal variables $(\mathbf{w}; a; b)$ and the dual variable α.

Finally, we remark that the data sampler is different from traditional one because it needs to sample both positive and negative examples. It also has great impact on the performance. We defer the discussion to section 6.4.5.

6.4.2 Stochastic AP Maximization

Using a surrogate loss, AP maximization can be formulated as an FCCO problem (2.36), i.e.,

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{\mathbf{x}_i \in \mathcal{S}_+} f(\mathbf{g}(\mathbf{w}; \mathbf{x}_i, \mathcal{S})), \tag{6.22}$$

where $\mathcal{S}_+$ denotes the set of n positive examples, $\mathcal{S}$ is the set of all examples, and

$$\begin{aligned}
&f(\mathbf{g}) = -\frac{[\mathbf{g}]_1}{[\mathbf{g}]_2}, \\
&\mathbf{g}(\mathbf{w}; \mathbf{x}_i, \mathcal{S}) = [g_1(\mathbf{w}; \mathbf{x}_i, \mathcal{S}), g_2(\mathbf{w}; \mathbf{x}_i, \mathcal{S})], \\
&g_1(\mathbf{w}; \mathbf{x}_i, \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}_j \in \mathcal{S}} \mathbb{I}(y_j = 1)\ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)), \\
&g_2(\mathbf{w}; \mathbf{x}_i, \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x}_j \in \mathcal{S}} \ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)),
\end{aligned}$$

where $\ell(\cdot)$ is a non-decreasing surrogate pairwise loss (see examples in Table 2.3).

We present an application of SOX to solving the above problem in Algorithm 28, which is referred to as SOAP.

Initialization of u

Unlike traditional algorithms, Algorithm 28 for AP maximization requires initializing an additional set of auxiliary variables $\mathbf{u}_1, \ldots, \mathbf{u}_n$. In contrast to the model parameter $\mathbf{w}$, which is randomly initialized, these auxiliary variables can be initialized upon their first update. Specifically, when index i is first sampled, we set $\mathbf{u}_{i,t-1}$ to the corresponding mini-batch estimator of the inner function value. As a result,

Algorithm 28 The SOAP algorithm for AP maximization (6.22)

Input: learning rate schedules $\{\eta_t\}_{t=1}^T$, $\{\gamma_t\}_{t=1}^T$; starting points $\mathbf{w}_1$, $\mathbf{u}_0$
for $t = 1, \ldots, T$ **do**
  Draw B_1 positive data $\mathcal{B}_t^+ \subset \mathcal{S}_+$ and B_2 negative data $\mathcal{B}_t^- \subset \mathcal{S}_-$
  for $\mathbf{x}_i \in \mathcal{B}_t^+$ **do**
    Update the inner function value estimators

$$u_{i,t}^{(1)} = (1-\gamma_t)u_{i,t-1}^{(1)} + \gamma_t \frac{1}{B_1+B_2} \sum_{\mathbf{x}_j \in |\mathcal{B}_t^+ \cup \mathcal{B}_t^-|} \mathbb{I}(y_j = 1)\ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i)),$$

$$u_{i,t}^{(2)} = (1-\gamma_t)u_{i,t-1}^{(2)} + \gamma_t \frac{1}{B_1+B_2} \sum_{\mathbf{x}_j \in |\mathcal{B}_t^+ \cup \mathcal{B}_t^-|} \ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i)),$$

  end for
  Set $\mathbf{u}_{i,t} = \mathbf{u}_{i,t-1}, i \notin \mathcal{B}_t^+$
  Compute the vanilla gradient estimator

$$\mathbf{z}_t = \frac{1}{B_1} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} \frac{1}{B_1+B_2} \sum_{\mathbf{x}_j \in |\mathcal{B}_t^+ \cup \mathcal{B}_t^-|} \frac{u_{i,t}^{(1)} - u_{i,t}^{(2)}\mathbb{I}(y_j = 1)}{(u_{i,t}^{(2)})^2} \nabla \ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))$$

  Update $\mathbf{w}_{t+1}$ by Momentum or AdamW
end for

the initial update of $\mathbf{u}_{i,t}$ coincides with the mini-batch estimate of the inner function at that point. This technique will be used in other FCCO applications.

Feature Learning

Similar to AUC maximization, the end-to-end training based on the AP surrogate objective does not favor feature learning. To mitigate this issue, one can first pretrain the encoder network on the training data by traditional supervised learning (e.g. ERM with the CE loss) or self-supervised representation learning and then fine-tune the feature extraction layers and a random initialized classifier layer by optimizing an AP surrogate objective. The compositional training could be also employed for unified feature learning and AP maximization.

Moving-average parameter γ_t

In practice, we can set $\gamma_t = \gamma$ and tune γ in the range $(0, 1)$ to optimize the validation performance.

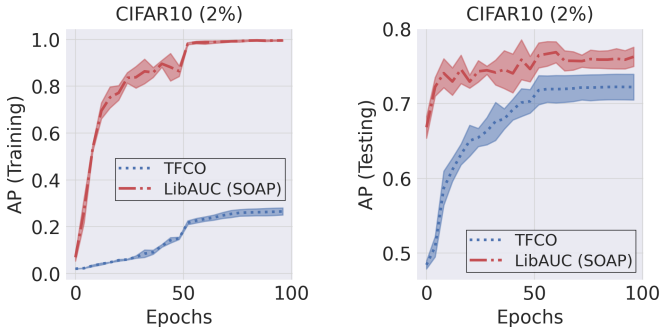


Fig. 6.11: Comparison of different methods for AP maximization. TFCO refers to the constrained optimization algorithm implemented in the Google TensorFlow Constrained Optimization library. The experiment was conducted on a constructed imbalanced binary classification task of CIFAR10, which originally contains 10 classes. These classes are partitioned into two equal groups to form the positive and negative classes based on their class IDs. The test data is unchanged (i.e., the testing data is still balanced). For more details, please refer to (Yuan et al., 2023b).

6.4.3 Stochastic Partial AUC Maximization

Stochastic OPAUC Maximization

We focus on maximizing the OPAUC with the false positive rate (FPR) restricted to the range $[0, \beta]$. As shown in Section 2.3.3, OPAUC maximization can be formulated as minimizing a surrogate objective:

$$\min_{\mathbf{w}} \frac{1}{n_+}\frac{1}{k} \sum_{\mathbf{x}_i \in \mathcal{S}_+} \sum_{\mathbf{x}_j \in \mathcal{S}_-^{\downarrow}[1,k]} \ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)), \tag{6.23}$$

where $k = \lfloor n_- \beta \rfloor$, $\mathcal{S}^{\downarrow}[1, k] \subseteq \mathcal{S}$ denotes the subset of examples whose rank in terms of their prediction scores in the descending order are in the range of $[1, k]$, and $\ell(\cdot)$ denotes a continuous surrogate pairwise loss such as in Table 2.3.

The challenge lies at how to tackle the top-k selection $\mathbf{x}_j \in \mathcal{S}_-^{\downarrow}[1, k]$. Below, we present two approaches: a direct approach that leverages the dual form of CVaR and an indirect approach that replaces the top-k selection by soft weighting.

A Direct Approach

This approach will be restricted to a non-decreasing pairwise loss function $\ell(s)$. Under this assumption, the ranking over negative samples by their prediction scores $h(\mathbf{w}; \mathbf{x}_j)$ is equivalent to that by the pairwise loss $\ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)), \mathbf{x}_j \in \mathcal{S}_i$. Hence, the average of pairwise losses over top-k negatives

Algorithm 29 SOPA for solving (6.26) of direct OPAUC maximization

1: Initialize $\mathbf{w}$ and $\boldsymbol{\nu}_1 = 0$
2: **for** $t = 1, \ldots, T$ **do**
3: Draw B_1 positive data $\mathcal{B}_t^+ \subset \mathcal{S}_+$ and B_2 negative data $\mathcal{B}_t^- \subset \mathcal{S}_-$
4: Compute $p_{ij} = \mathbb{I}(\ell(h(\mathbf{w}_t, \mathbf{x}_j) - h(\mathbf{w}_t, \mathbf{x}_i)) - \nu_{i,t} > 0)$ for $\mathbf{x}_i \in \mathcal{B}_t^+, \mathbf{x}_j \in \mathcal{B}_t^-$
5: **for** $i \in \mathcal{B}_t^+$ **do**
6: Update $\nu_{i,t+1} = \nu_{i,t} - \eta_2(\frac{k}{n_-} - \frac{1}{B_2}\sum_{\mathbf{x}_j \in \mathcal{B}_t^-} p_{ij})$
7: **end for**
8: Set $\nu_{i,t+1} = \nu_{i,t}, i \notin \mathcal{B}_t^+$
9: Compute a vanilla gradient estimator $\mathbf{z}_t$ by

$$\mathbf{z}_t = \frac{1}{B_1 B_2} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} p_{ij} \nabla_{\mathbf{w}} \ell(h(\mathbf{w}_t, \mathbf{x}_j) - h(\mathbf{w}_t, \mathbf{x}_i))$$

10: Update $\mathbf{w}_{t+1}$ by SGD or Momentum or AdamW
11: **end for**

$$L_i(\mathbf{w}) = \frac{1}{k} \sum_{\mathbf{x}_j \in \mathcal{S}_-^{\downarrow}[1,k]} \ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)) \tag{6.24}$$

is equivalent to the average of top-k pairwise losses over negative data, i.e., an empirical CVaR estimator. Then leveraging the dual form of CVaR (2.15), we transform the above loss into a minimization problem, i.e.,

$$L_i(\mathbf{w}) = \min_{\nu_i} \frac{1}{k} \sum_{\mathbf{x}_j \in \mathcal{S}_-} [\ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)) - \nu_i]_+ + \nu_i. \tag{6.25}$$

As a result, we have the following equivalent reformulation.

Lemma 6.1 (Reformulation of OPAUC maximization.) *When $\ell(\cdot)$ is non-decreasing, then the problem (6.23) for OPAUC maximization is equivalent to*

$$\min_{\mathbf{w}, \boldsymbol{\nu} \in \mathbb{R}^{n_+}} F(\mathbf{w}, \boldsymbol{\nu}) = \frac{1}{n_+} \sum_{\mathbf{x}_i \in \mathcal{S}_+} \left\{ \frac{k}{n_-} \nu_i + \frac{1}{n_-} \sum_{\mathbf{x}_j \in \mathcal{S}_-} (\ell(h(\mathbf{w}, \mathbf{x}_j) - h(\mathbf{w}, \mathbf{x}_i)) - \nu_i)_+ \right\}, \tag{6.26}$$

The above problem is a special case of compositional OCE studied in Section 5.5.

A benefit for solving (6.26) is that an unbiased stochastic subgradient can be computed in terms of $(\mathbf{w}, \boldsymbol{\nu})$. We present a method in Algorithm 29, which is an application of the ASGD and is referred to as SOPA. A key feature of SOPA is that the stochastic gradient estimator for $\mathbf{w}$ (Step 9) is a weighted average gradient of the pairwise losses for all pairs in the mini-batch. The weights p_{ij} (either 0 or 1) are dynamically computed by Step 4, which compares the pairwise loss $(\ell(h(\mathbf{w}_t, \mathbf{x}_i) - h(\mathbf{w}_t, \mathbf{x}_j))$ with the threshold variable $\nu_{i,t}$, which is also updated by an SGD step.

Algorithm 30 SOPA-s for solving (6.28) of indirect OPAUC maximization

1: Initialize $\mathbf{w}, \mathbf{u}_0$
2: **for** $t = 1, \ldots, T$ **do**
3: Draw B_1 positive data $\mathcal{B}_t^+ \subset \mathcal{S}_+$ and B_2 negative data $\mathcal{B}_t^- \subset \mathcal{S}_-$
4: **for** $i \in \mathcal{B}_t^+$ **do**
5: Update $u_{i,t} = (1-\gamma)u_{i,t-1} + \gamma \frac{1}{B_2} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} \exp\left(\frac{\ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))}{\tau}\right)$
6: **end for**
7: Set $u_{i,t} = u_{i,t-1}, i \notin \mathcal{B}_t^+$
8: Compute $p_{ij} = \exp(\ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))/\tau)/u_{i,t}$ for $\mathbf{x}_i \in \mathcal{B}_t^+, \mathbf{x}_j \in \mathcal{B}_t^-$
9: Compute a vanilla gradient estimator $\mathbf{z}_t$ by

$$\mathbf{z}_t = \frac{1}{B_1 B_2} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} p_{ij} \nabla \ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))$$

10: Update $\mathbf{w}_{t+1}$ by Momentum or AdamW method.
11: **end for**

The convergence guarantee of SOPA using the SGD update for $\mathbf{w}_t$ has been established in Section 5.5. In practice, the convergence speed of SOPA may be further accelerated by integrating Momentum or Adam updates for the model parameter $\mathbf{w}$.

An indirect approach by FCCO

Due to the connection between CVaR and DRO (2.13), an alternative approach is to replace the top-k pairwise loss $L_i(\mathbf{w})$ by a KL-regularized DRO, i.e.,

$$\begin{aligned}\hat{L}_i(\mathbf{w}) &= \max_{\mathbf{p} \in \Delta_n} \sum_{\mathbf{x}_j \in \mathcal{S}_-} p_j \ell(h(\mathbf{w};\mathbf{x}_j) - h(\mathbf{w};\mathbf{x}_i)) - \tau \mathrm{KL}(\mathbf{p}, 1/n_-) \\ &= \tau \log\left(\frac{1}{n_-} \sum_{\mathbf{x}_j \in \mathcal{S}_-} \exp\left(\frac{\ell(h(\mathbf{w};\mathbf{x}_j) - h(\mathbf{w};\mathbf{x}_i))}{\tau}\right)\right).\end{aligned} \tag{6.27}$$

As a result, an indirect approach for OPAUC maximization is to solve the following FCCO problem:

$$\min_{\mathbf{w}} \frac{1}{n_+} \sum_{i=1}^{n_+} \tau \log\left(\frac{1}{n_-} \sum_{\mathbf{x}_j \in \mathcal{S}_-} \exp\left(\frac{\ell(h(\mathbf{w};\mathbf{x}_j) - h(\mathbf{w};\mathbf{x}_i))}{\tau}\right)\right). \tag{6.28}$$

An application of the SOX algorithm is given in Algorithm 30, which is referred to as SOPA-s. The key difference between SOPA-s and SOPA lies at the pairwise weights p_{ij} in SOPA-s (Step 8) are soft weights between 0 and 1, in contrast to the hard weights $p_{ij} \in \{0, 1\}$ in SOPA.

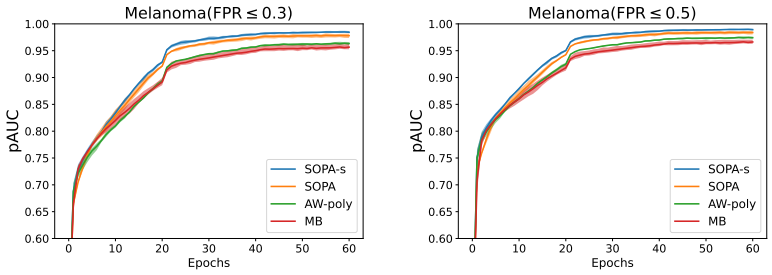


Fig. 6.12: Comparison of different methods for OPAUC maximization with FPR less than $\beta = 0.3$ (left) and $\beta = 0.5$ (right). The dataset is Melanoma classification from Kaggle competition. The training set has only 1.76% positive (malignant) samples. MB refers to the BSGD approach that computes gradients using only the top β% of negative examples within each mini-batch; AW-Poly is a heuristic weighted method that assigns weights to negative samples in the mini-batch using a manually designed weighting function. For more details, please refer to (Zhu et al., 2022b).

Stochastic TPAUC Maximization

As shown in Section 2.3.3, empirical maximization of TPAUC with FPR $\leq \beta$, TPR $\geq \alpha$ can be formulated as:

$$\min_{\mathbf{w}} \frac{1}{k_1}\frac{1}{k_2} \sum_{\mathbf{x}_i \in \mathcal{S}_+^{\uparrow}[1,k_1]} \sum_{\mathbf{x}_j \in \mathcal{S}_-^{\downarrow}[1,k_2]} \ell(h(\mathbf{w};\mathbf{x}_j) - h(\mathbf{w};\mathbf{x}_i)), \tag{6.29}$$

where $k_1 = \lfloor n_+(1-\alpha) \rfloor$, $k_2 = \lfloor n_-\beta \rfloor$. If we define

$$L_i(\mathbf{w}) = \frac{1}{k_2} \sum_{\mathbf{x}_j \in \mathcal{S}_-^{\downarrow}[1,k_2]} \ell(h(\mathbf{w};\mathbf{x}_j) - h(\mathbf{w};\mathbf{x}_i)), \tag{6.30}$$

then, the problem in (6.29) can be written as:

$$\min_{\mathbf{w}} \frac{1}{k_1} \sum_{\mathbf{x}_i \in \mathcal{S}_+^{\uparrow}[1,k_1]} L_i(\mathbf{w}). \tag{6.31}$$

Similar to OPAUC maximization, we will present a direct approach and an indirect approach.

A Direct Approach

The first approach is based on the following reformulation of TPAUC maximization.

Algorithm 31 STACO for solving (6.32) of direct TPAUC maximization

Initialize $\mathbf{w}$ and $\nu_1 = 0$, $\nu' = 0$
for $t = 1, \ldots, T$ **do**
  Draw B_1 positive data $\mathcal{B}_t^+ \subset \mathcal{S}_+$ and B_2 negative data $\mathcal{B}_t^- \subset \mathcal{S}_-$
  Compute $p_{ij} = \mathbb{I}(\ell(h(\mathbf{w}_t, \mathbf{x}_i) - h(\mathbf{w}_t, \mathbf{x}_j)) - \nu_{i,t} > 0)$ for $\mathbf{x}_i \in \mathcal{B}_t^+, \mathbf{x}_j \in \mathcal{B}_t^-$
  for $i \in \mathcal{B}_t^+$ **do**
    Update $y_{i,t+1}$ and $\nu_{i,t+1}$ by

$$y_{i,t+1} = \left[y_{i,t} - \eta_2 \left\{ \frac{1}{B_2} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} (\ell(h(\mathbf{w}_t; \mathbf{x}_j) - h(\mathbf{w}_t; \mathbf{x}_i)) - \nu_{i,t})_+ + \frac{k_2}{n_-}(\nu_{i,t} - \nu'_t) \right\} \right]_{[0,1]}$$

$$\nu_{i,t+1} = \nu_{i,t} - \eta_1 y_{i,t+1} \left(\frac{k_2}{n_-} - \frac{1}{B_2} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} p_{ij} \right)$$

  end for
  Set $y_{i,t+1} = y_{i,t}, i \notin \mathcal{B}_t^+$ and $\nu_{i,t+1} = \nu_{i,t}, t \notin \mathcal{B}_t^+$
  Update $\nu'_{t+1} = \nu'_t - \eta_1 (\frac{k_1 k_2}{n_+ n_-} - \frac{k_2}{n_-} \frac{1}{B_1} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} y_{i,t+1})$
  Compute a vanilla gradient estimator $\mathbf{z}_t$ by

$$\mathbf{z}_t = \frac{1}{B_1 B_2} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} y_{i,t+1} p_{ij} \nabla_{\mathbf{w}} \ell(h(\mathbf{w}_t, \mathbf{x}_j) - h(\mathbf{w}_t, \mathbf{x}_i))$$

  Update $\mathbf{w}_{t+1}$ by SGD, Momentum or AdamW
end for

Lemma 6.2 (Reformulation of TPAUC maximization.) *When $\ell(\cdot)$ is non-decreasing, the problem (6.29) for TPAUC maximization is equivalent to*

$$\min_{\mathbf{w}, \boldsymbol{\nu}, \nu'} \frac{1}{n_+} \sum_{\mathbf{x}_i \in \mathcal{S}_+} f(g_i(\mathbf{w}, \boldsymbol{\nu}, \nu')) + \frac{k_1 k_2}{n_+ n_-} \nu', \tag{6.32}$$

where $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_{n_+})^\top$, $f(\cdot) = [\cdot]_+$ and

$$g_i(\mathbf{w}, \boldsymbol{\nu}, \nu') = \frac{1}{n_-} \sum_{\mathbf{x}_j \in \mathcal{S}_-} (\ell(h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)) - \nu_i)_+ + \frac{k_2}{n_-}(\nu_i - \nu').$$

We leave the proof as an excise for the reader.

It is clear that the problem (6.32) is an instance of FCCO, where the outer function is non-smooth and monotonically non-decreasing. Hence, SONX, SONEX, and ALEXR can be applied. We present an application of ALEXR for solving the above problem in Algorithm 31 (referred to as STACO) based on its min-max reformulation:

Algorithm 32 SOTA-s for solving (6.33) of Indirect TPAUC Maximization

Initialize $\mathbf{w}_1, \mathbf{u}_1, v_1$,
for $t = 1, \ldots, T$ **do**
  Draw B_1 positive data $\mathcal{B}_t^+ \subset \mathcal{S}_+$ and B_2 negative data $\mathcal{B}_t^- \subset \mathcal{S}_-$
  for $i \in \mathcal{B}_t^+$ **do**
    Update $u_{i,t} = (1-\gamma_0)u_{i,t-1} + \gamma_0 \frac{1}{B_2} \sum_{\mathbf{x}_j \in \mathcal{B}_-^t} \exp\left(\frac{\ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))}{\tau_2}\right)$
  end for
  Set $u_{i,t} = u_{i,t-1}, i \notin \mathcal{B}_t^+$
  Let $v_t = (1-\gamma_1)v_{t-1} + \gamma_1 \frac{1}{B_1} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} (u_{i,t})^{\tau_2/\tau_1}$
  Compute

$$p_{ij} = \frac{\exp(\ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))/\tau_2)(u_{i,t})^{\tau_2/\tau_1 - 1}}{v_t}, \forall \mathbf{x}_i \in \mathcal{B}_t^+, \mathbf{x}_j \in \mathcal{B}_t^-$$

  Compute a vanilla gradient estimator $\mathbf{z}_t$ by

$$\mathbf{z}_t = \frac{1}{B_1 B_2} \sum_{\mathbf{x}_i \in \mathcal{B}_t^+} \sum_{\mathbf{x}_j \in \mathcal{B}_t^-} p_{ij} \nabla \ell(h(\mathbf{w}_t;\mathbf{x}_j) - h(\mathbf{w}_t;\mathbf{x}_i))$$

  Update $\mathbf{w}_{t+1}$ by Momentum or AdamW
end for

$$\min_{\mathbf{w},\nu,\nu'} \max_{\mathbf{y}\in[0,1]^{n_+}} \frac{1}{n_+} \sum_{\mathbf{x}_i \in \mathcal{S}_+} y_i \left[\frac{1}{n_-} \sum_{\mathbf{x}_j \in \mathcal{S}_-} (\ell(\mathbf{w};\mathbf{x}_i,\mathbf{x}_j) - \nu_i)_+ + \frac{k_2}{n_-}(\nu_i - \nu') \right] + \frac{k_1 k_2}{n_+ n_-} \nu'.$$

An Indirect Approach

Following the strategy used in OPAUC maximization, we adopt an indirect approach by replacing top-k estimators with their KL-regularized DRO counterparts, which yield smooth surrogate objectives.

With a non-decreasing pairwise surrogate loss $\ell(\cdot)$, $L_i(\mathbf{w})$ is a non-increasing function of $h(\mathbf{w};\mathbf{x}_i)$, the average of $L_i(\mathbf{w})$ over bottom-k_1 positive examples in (6.31) is equivalent to the average of top-k_1 losses $L_i(\mathbf{w})$ over all positive data. Hence, we approximate the resulting top-k_1 estimator by a KL-regularized objective:

$$\tau_1 \log\left(\frac{1}{n_+} \sum_{\mathbf{x}_i \in \mathcal{S}_+} \exp\left(\frac{L_i(\mathbf{w})}{\tau_1}\right)\right).$$

Then, we substitute $L_i(\mathbf{w})$ with $\hat{L}_i(\mathbf{w})$ as defined in (6.27), leading to the following smoothed objective:

$$
\begin{aligned}
F(\mathbf{w}) &= \tau_1 \log\left(\frac{1}{n_+}\sum_{\mathbf{x}_i\in\mathcal{S}_+}\exp\left(\frac{\hat{L}_i(\mathbf{w})}{\tau_1}\right)\right) \\
&= \tau_1 \log\left(\frac{1}{n_+}\sum_{\mathbf{x}_i\in\mathcal{S}_+}\exp\left(\frac{\tau_2\log\left(\frac{1}{n_-}\sum_{\mathbf{x}_j\in\mathcal{S}_-}\exp\left(\frac{\ell(h(\mathbf{w};\mathbf{x}_j)-h(\mathbf{w};\mathbf{x}_i))}{\tau_2}\right)\right)}{\tau_1}\right)\right) \\
&= \tau_1 \log\left(\frac{1}{n_+}\sum_{\mathbf{x}_i\in\mathcal{S}_+}\left(\frac{1}{n_-}\sum_{\mathbf{x}_j\in\mathcal{S}_-}\exp\left(\frac{\ell(h(\mathbf{w};\mathbf{x}_j)-h(\mathbf{w};\mathbf{x}_i))}{\tau_2}\right)\right)^{\frac{\tau_2}{\tau_1}}\right).
\end{aligned}
$$

To minimize this objective, we formulate the problem as a three-level compositional stochastic optimization:

$$
\min_{\mathbf{w}} f_1\left(\frac{1}{n_+}\sum_{\mathbf{x}_i\in\mathcal{S}_+} f_2(g_i(\mathbf{w}))\right), \tag{6.33}
$$

where $f_1(s) = \tau_1\log(s)$, $f_2(g) = g^{\tau_2/\tau_1}$, and

$$
g_i(\mathbf{w}) = \frac{1}{n_-}\sum_{\mathbf{x}_j\in\mathcal{S}_-}\exp\left(\frac{\ell(h(\mathbf{w};\mathbf{x}_j)-h(\mathbf{w};\mathbf{x}_i))}{\tau_2}\right).
$$

The inner function of f_1 exhibits a finite-sum coupled compositional optimization (FCCO) structure. To accurately estimate $\nabla f_1(\cdot)$ at the inner function value, we maintain a moving average estimator v_t to track $\frac{1}{n_+}\sum_{\mathbf{x}_i\in\mathcal{S}_+} f_2(g_i(\mathbf{w}_t))$.

We present a stochastic optimization algorithm—referred to as SOTA-s—for solving this problem in Algorithm 32. We update $u_{i,t}$ to track $g_i(\mathbf{w}_t)$ in Step 5 and maintain v_t to estimate $\frac{1}{n_+}\sum_{\mathbf{x}_i\in\mathcal{S}_+} f_2(g_i(\mathbf{w}_t))$ in Step 8. The gradient estimator in Step 9 is given by:

$$
\nabla f_1(v_t)\cdot\frac{1}{|\mathcal{B}_+|}\sum_{\mathbf{x}_i\in\mathcal{B}_t^+}\nabla f_2(u_{i,t})\cdot\nabla\hat{g}_i(\mathbf{w}_t),
$$

where $\hat{g}_i(\mathbf{w}_t) = \frac{1}{B_2}\sum_{\mathbf{x}_j\sim\mathcal{B}_t^-}\exp\left(\frac{\ell(h(\mathbf{w}_t;\mathbf{x}_j)-h(\mathbf{w}_t;\mathbf{x}_i))}{\tau_2}\right)$.

6.4.4 Stochastic NDCG Maximization

In Section 2.3.4, we have formulated NDCG maximization as the following empirical X-risk minimization problem:

$$
\min_{\mathbf{w}}\frac{1}{N}\sum_{q=1}^{N}\frac{1}{Z_q}\sum_{\mathbf{x}_{q,i}\in\mathcal{S}_q^+}\frac{1-2^{y_{q,i}}}{\log_2(N_q g(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q)+1)}, \tag{6.34}
$$

Algorithm 33 SONG

Initialize $\mathbf{w}_1, \mathbf{u}_0$
for $t = 1, \ldots T$ **do**
  Draw some relevant Q-I pairs $\mathcal{B}_t = \{(q, \mathbf{x}_{q,i})\} \subset \mathcal{S}$
  For each sampled q draw a batch of items $\mathcal{B}_q^t \subset \mathcal{S}_q$
  for each sampled Q-I pair $(q, \mathbf{x}_{q,i}) \in \mathcal{B}_t$ **do**
    Compute $u_{q,i,t} = (1-\gamma)u_{q,i,t-1} + \gamma \frac{1}{|\mathcal{B}_q^t|} \sum_{\mathbf{x}' \in \mathcal{B}_q^t} \ell(s(\mathbf{w}_t; \mathbf{x}', q) - s(\mathbf{w}_t; \mathbf{x}_{q,i}, q))$
    Compute

$$p_{q,i} = \nabla f_{q,i}(u_{q,i,t}) = \frac{(2^{y_{q,i}} - 1)N_q}{Z_q(N_q u_{q,i,t} + 1)\log_2^2(N_q u_{q,i,t} + 1)\ln(2)}$$

  end for
  Compute a vanilla gradient estimator $\mathbf{z}_t$ by

$$\mathbf{z}_t = \frac{1}{|\mathcal{B}_t|} \sum_{(q,\mathbf{x}_{q,i}) \in \mathcal{B}_t} p_{q,i} \frac{1}{|\mathcal{B}_q^t|} \sum_{\mathbf{x}' \in \mathcal{B}_q^t} \ell(s(\mathbf{w}; \mathbf{x}', q) - s(\mathbf{w}; \mathbf{x}, q))$$

  update $\mathbf{w}_{t+1}$ by Momentum and AdamW optimizer
end for

where $N_q g(\mathbf{w}; \mathbf{x}, \mathcal{S}_q) = \sum_{\mathbf{x}' \in \mathcal{S}_q} \ell(s(\mathbf{w}; \mathbf{x}', q) - s(\mathbf{w}; \mathbf{x}, q))$ is a surrogate of the rank function $r(\mathbf{w}; \mathbf{x}, S_q) = \sum_{\mathbf{x}' \in \mathcal{S}_q} \mathbb{I}(s(\mathbf{w}; \mathbf{x}', q) - s(\mathbf{w}; \mathbf{x}, q) \geq 0)$, and $s(\mathbf{w}; \mathbf{x}, q)$ denotes the predicted relevance score for item $\mathbf{x}$ with respect to query q, parameterized by $\mathbf{w} \in \mathbb{R}^d$ (e.g., a deep neural network).

As a result, NDCG maximization can be rewritten as an instance of FCCO:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{|\mathcal{S}|} \sum_{(q,\mathbf{x}_{q,i}) \in \mathcal{S}} f_{q,i}(g(\mathbf{w}; \mathbf{x}_{q,i}, \mathcal{S}_q)), \tag{6.35}$$

where $\mathcal{S} = \{(q, \mathbf{x}_{q,i}) \mid q \in \mathcal{Q}, \mathbf{x}_{q,i} \in \mathcal{S}_q^+\}$ represent the collection of all relevant query-item (Q-I) pairs, and

$$f_{q,i}(g) = \frac{1}{Z_q} \frac{1 - 2^{y_{q,i}}}{\log_2(N_q g + 1)}.$$

We apply the SOX method to this problem as shown in Algorithm 33, which we call SONG.

Top-K NDCG Maximization

In practice, top-K NDCG is the preferred metric for information retrieval and recommender systems, as users primarily focus on the highest-ranked items. It is defined as:

$$\frac{1}{N}\sum_{q=1}^{N}\frac{1}{Z_q^{(K)}}\sum_{\mathbf{x}_{q,i}\in\mathcal{S}_q^+}\mathbb{I}(\mathbf{x}_{q,i}\in\mathcal{S}_q^{(K)})\cdot\frac{2^{y_{q,i}}-1}{\log_2(r(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q)+1)},$$

where $\mathcal{S}_q^{(K)}$ is the set of top-K items based on predicted scores, and $Z_q^{(K)}$ is the ideal DCG in the top-K positions.

Optimizing top-K NDCG introduces an added complexity: selecting the top-K items is non-differentiable. Unlike pAUC, where a top-K estimator exists, the surrogate function

$$\frac{2^{y_{q,i}}-1}{\log_2(N_q g(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q)+1)}$$

is not generally monotonic in the score $s(\mathbf{w};\mathbf{x}_{q,i},q)$ unless all $y_{q,i}$ values are identical. We consider two approaches to handle this problem.

Approach 1: Surrogate for Top-K Inclusion

We use the identity $\mathbb{I}(\mathbf{x}_{q,i}\in\mathcal{S}_q^{(K)})=\mathbb{I}(K-r(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q)\geq 0)$ and approximate it by a non-decreasing surrogate $\psi(K-N_q g(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q))$, e.g., the sigmoid function. The resulting objective becomes:

$$\min_{\mathbf{w}\in\mathbb{R}^d}\frac{1}{|\mathcal{S}|}\sum_{q=1}^{N}\sum_{\mathbf{x}_{q,i}\in\mathcal{S}_q^+}\psi(K-N_q g(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q))\cdot\frac{1-2^{y_{q,i}}}{Z_q^{(K)}\log_2(N_q g(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q)+1)}. \tag{6.36}$$

This can be optimized using FCCO techniques.

Approach 2: Threshold Estimation via Bilevel Optimization

Denote by $\lambda_q(\mathbf{w})$ the the $(K+1)$-th largest score among all $\mathbf{x}'\in\mathcal{S}_q$. We use the identity $\mathbb{I}(\mathbf{x}_{q,i}\in\mathcal{S}_q^{(K)})=\mathbb{I}(s(\mathbf{w};\mathbf{x}_{q,i},q)>\lambda_q(\mathbf{w}))$ and approximate it by $\psi(s(\mathbf{w};\mathbf{x}_{q,i},q)-\lambda_q(\mathbf{w}))$. The threshold $\lambda_q(\mathbf{w})$ can be computed by solving a convex optimization problem as shown in the lemma below.

Lemma 6.3 *Let* $\lambda_q(\mathbf{w})=\arg\min_\lambda(K+\varepsilon)\lambda+\sum_{\mathbf{x}'\in\mathcal{S}_q}(s(\mathbf{w};\mathbf{x}',q)-\lambda)_+$ *for any* $\varepsilon\in(0,1)$*, then* $\lambda_q(\mathbf{w})$ *is the* $(K+1)$*-th largest value among* $\{s(\mathbf{w};\mathbf{x}',q)|\mathbf{x}'\in\mathcal{S}_q\}$.

As a result, we formulate the following bilevel optimization problem for top-K NDCG maximization:

$$\begin{aligned}\min_{\mathbf{w}}\quad & \frac{1}{|\mathcal{S}|}\sum_{q=1}^{N}\sum_{\mathbf{x}_{q,i}\in\mathcal{S}_q^+}\frac{\psi(s(\mathbf{w};\mathbf{x}_{q,i},q)-\lambda_q(\mathbf{w}))\cdot(1-2^{y_{q,i}})}{Z_q^{(K)}\log_2(N_q g(\mathbf{w};\mathbf{x}_{q,i},\mathcal{S}_q)+1)}\\ \text{s.t.}\quad & \lambda_q(\mathbf{w})=\arg\min_\lambda\frac{K+\varepsilon}{N_q}\lambda+\frac{1}{N_q}\sum_{\mathbf{x}'\in\mathcal{S}_q}(s(\mathbf{w};\mathbf{x}',q)-\lambda)_+,\quad\forall q.\end{aligned} \tag{6.37}$$

This bilevel formulation is challenging due to the non-smooth and non-strongly-convex lower-level problem. One remedy is to apply Nesterov smoothing to the hinge loss (see Example 5.1) and add a small quadratic regularization term of λ to the lower level objective. This allows employing the Approach 1 of using moving-average estimators from Section 4.5.3.

In practice, we can ignore the gradient of ψ and adapt the SONG algorithm by updating λ_q iteratively and modifying $p_{q,i}$ as:

$$\lambda_{q,t+1} = \lambda_{q,t} - \eta' \left(\frac{K+\varepsilon}{N_q} + \frac{1}{|\mathcal{B}_q|} \sum_{\mathbf{x}' \in \mathcal{B}_q^t} \mathbb{I}(s(\mathbf{w}_t; \mathbf{x}', q) > \lambda) \right), \quad \forall q \in \mathcal{B}_t,$$
$$p_{q,i} = \psi(s(\mathbf{w}_t; \mathbf{x}_{q,i}, q) - \lambda_{q,t+1}) \cdot \nabla f_{q,i}(u_{q,i,t}).$$

As with other non-decomposable metrics, it is beneficial to first pretrain the model by optimizing the listwise cross-entropy loss, which itself is an FCCO problem, as defined in (2.47).

6.4.5 The LibAUC Library

The algorithms presented in Section 6.4 for various X-risk minimization tasks share several common features: (1) they all require sampling both positive and negative examples; (2) their vanilla gradient updates involve a weighted sum of gradients from pairwise losses computed on the sampled data; and (3) they utilize moving-average estimators to track inner function values. These shared characteristics motivate the design of a unified implementation pipeline. To this end, the `LibAUC` library was developed to encapsulate these principles within a modular and extensible framework, built on top of the PyTorch ecosystem. Below, we highlight several key components of `LibAUC`. For tutorials and source code, we refer interested readers to the GitHub repository:

LibAUC GitHub Repository

`https://github.com/Optimization-AI/LibAUC`

Pipeline

The training pipeline of a deep neural network in the `LibAUC` library is illustrated in Figure 6.13. It consists of five core modules: `Dataset`, `Controlled Data Sampler`, `Model`, `Dynamic Mini-batch Loss`, and `Optimizer`. While the `Dataset`, `Model`, and `Optimizer` modules align closely with those in standard training frameworks, the key innovations lie in the `Dynamic Mini-batch Loss` and `Controlled Data Sampler` modules.

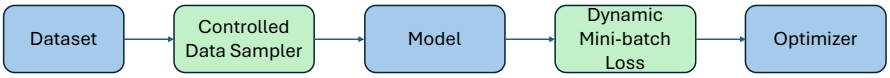


Fig. 6.13: Training pipeline of the LibAUAC library for deep learning.

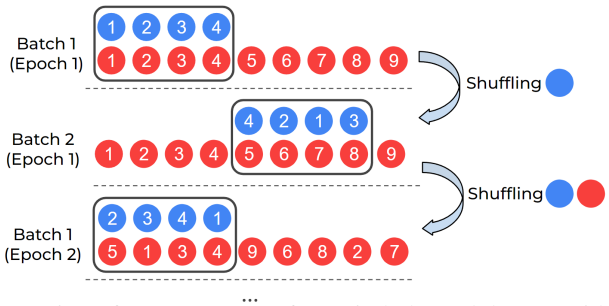


Fig. 6.14: Illustration of `DualSampler` for an imbalanced dataset with 4 positives ● and 9 negatives ●.

The `Dynamic Mini-batch Loss` module defines the loss using dynamically updated variables, which are computed and refined with forward propagation results. This design ensures that compositional gradients can be correctly estimated from mini-batch samples using backpropagation. The `Controlled Data Sampler` module, in contrast to standard random sampling strategies, allows fine-grained control over the ratio of positive to negative samples. This control can be tuned to improve learning effectiveness and overall performance.

Controlled Data Sampler

Unlike traditional ERM, EXM requires sampling to estimate the outer average and the inner average. In algorithms for AUC, AP, OPAUC and TPAUC optimization, we need to sample two mini-batches $\mathcal{B}_+^t \subset \mathcal{S}_+$ and $\mathcal{B}_-^t \subset \mathcal{S}_-$ at each iteration t. When the total batch size is fixed, balancing the mini-batch size for outer average and that for the inner average could be beneficial for accelerating convergence according to our theoretical analysis in Chapter 5. Hence, the `Controlled Data Sampler` module can help ensure that both positive and negative samples will be sampled and the proportion of positive samples in the mini-batch can be controlled by a hyper-parameter.

DualSampler. For binary classification problems, DualSampler takes as input hyper-parameters such as `batch_size` and `sampling_rate`, and generates the customized mini-batch samples, where `sampling_rate` controls the number of positive samples in the mini-batch according to the formula:

$$\texttt{\#positives} = \texttt{batch\_size} * \texttt{sampling\_rate}.$$

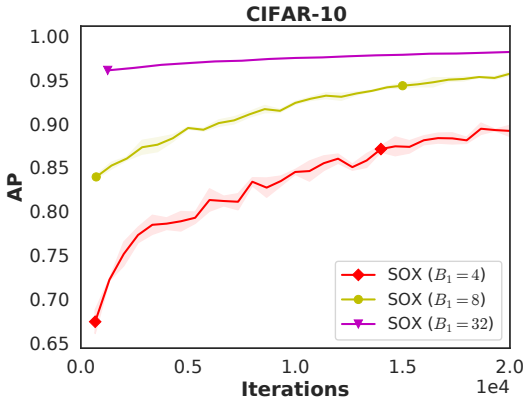


Fig. 6.15: The training curves of AP for different number of positive examples per mini-batch in DualSampler when the total batch size is fixed to 64. The algorithm is SOPA - a variant of SOX. Experiments were conducted on a constructed imbalanced binary classification task derived from CIFAR-10, identical to the setting used in Figure 6.11.

Figure 6.14 shows an example of `DualSampler` for constructing mini-batch data with even positive and negative samples on an imbalanced dataset with 4 positives and 9 negatives. To improve the sampling speed, two lists of indices are maintained for the positive data and negative data, respectively. At the beginning, we shuffle the two lists and then take the first 4 positives and 4 negatives to form a mini-batch. Once the positive list is used up, we only reshuffle the positive list and take 4 shuffled positives to pair with next 4 negatives in the negative list as a mini-batch. Once the negative list is used up, we re-shuffle both lists and repeat the same process as above. An illustration of the impact of the DualSampler on the convergence is shown in Figure 6.15.

TriSampler. For multi-label classification problems with many labels and ranking problems, `TriSampler` first samples a set of tasks controlled by a hyperparameter `sampled_tasks`, and then sample positive and negative data for each task.

The following code snippet shows how to define `DualSampler` and `TriSampler`.

```
from libauc.sampler import DualSampler, TriSampler
dualsampler = DualSampler(trainSet,
                          batch_size=32,
                          sampling_rate=0.1)
trisampler = TriSampler(trainSet,
                        batch_size_per_task=32,
                        sampled_tasks=5,
                        sampling_rate_per_task=0.1)
```

Dynamic Mini-batch Loss

To compute the vanilla gradient estimator, we invoke backpropagation using the PyTorch function `loss.backward()` on a defined loss. The vanilla gradient estimators for pAUC, AP, and NDCG maximization share a common structure of the form

$$\frac{1}{|\mathcal{B}_1|}\sum_{\mathbf{x}_i\in\mathcal{B}_1}\frac{1}{|\mathcal{B}_2|}\sum_{\mathbf{x}_j\in\mathcal{B}_2} p_{ij}\nabla\ell(h(\mathbf{w};\mathbf{x}_j)-h(\mathbf{w};\mathbf{x}_i)),$$

where the weights p_{ij} are computed from dynamic variables within the algorithm. To enable the use of `loss.backward()`, it suffices to define a mini-batch loss as $\frac{1}{|\mathcal{B}_1|}\sum_{\mathbf{x}_i\in\mathcal{B}_1}\frac{1}{|\mathcal{B}_2|}\sum_{\mathbf{x}_j\in\mathcal{B}_2} p_{ij}\ell(h(\mathbf{w};\mathbf{x}_j)-h(\mathbf{w};\mathbf{x}_i))$, where p_{ij} is detached from the computation graph to avoid unnecessary backpropagation through these variables. Since p_{ij} is evolving across iterations, the mini-batch loss is called dynamic mini-batch loss. A high-level pseudocode example for SOPAs is provided in Figure 6.16.

```
# define dynamic mini-batch loss
def pAUCLoss(**kwargs):  # dynamic mini-batch loss
    sur_loss = surrogate_loss(neg_logits - pos_logits)
    exp_loss = torch.exp(sur_loss / Lambda)
    u[index] = (1 - gamma) * u[index] + gamma * (exp_loss.mean(1)
        )
    p = (exp_loss / u[index]).detach()
    loss = torch.mean(p * sur_loss)
    return loss

# optimization
for data, targets, index in dataloader:
    logits = model(data)
    loss = pAUCLoss(logits, targets, index)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Fig. 6.16: High-level pseudocode for SOPAs.

Comparison with Existing Libraries

We present some benchmark results of LibAUC in comparison with other state-of-the-art training libraries.

Comparison with the TFCO Library. We compare LibAUC (SOAP) with Google's TensorFlow Constrained Optimization (TFCO) library for optimizing average precision (AP). Both methods are trained for 100 epochs using a batch size of 128, the Adam optimizer with a learning rate of 1e-3, and a weight decay of 1e-4 on a binary classification task derived from CIFAR-10 with `imratio` $\in \{1\%, 2\%\}$. The training and testing learning curves, shown in Figure 6.11, demonstrate that LibAUC consistently outperforms TFCO.

Comparison with the TF-Ranking Library. We evaluate LibAUC, using SONG for NDCG maximization, against Google's TF-Ranking library, which implements `ApproxNDCG` and `GumbelNDCG`. Experiments are conducted on two large-scale datasets

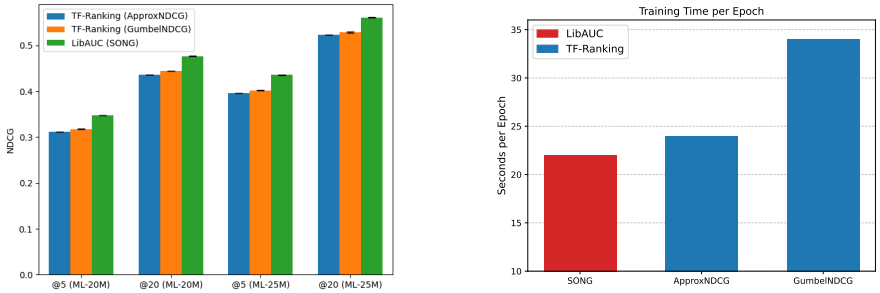


Fig. 6.17: Left: Benchmarks of NDCG optimization on MovieLens (ML) 20M and 25M datasets, @K means NDCG at top K. Right: Runtime Comparison between LibAUC and TF-ranking for NDCG maximization. For more details, please refer to (Yuan et al., 2023b).

—MovieLens20M and MovieLens25M—from the MovieLens platform. As shown in Figure 6.17, LibAUC achieves superior performance on both datasets. Furthermore, the runtime comparison shows that LibAUC's NDCG maximization algorithm is more efficient than the corresponding implementations in TF-Ranking.

6.5 Discriminative Pretraining of Representation Models

In Chapter 2, we briefly introduced the core concepts of representation learning and highlighted its growing significance in modern AI systems. In contemporary AI, representation models are learned through Self-supervised learning (SSL), which has emerged as a powerful paradigm for learning representation models without the need for labeled data. Among the most prominent frameworks within SSL is *contrastive learning*, which forms positive pairs by applying different augmentations to the same data sample or taking different views of the same data, while treating different data as negatives. In this section, we delve deeper into contrastive learning, with a focus on its applications to both unimodal and multimodal representation learning.

6.5.1 Mini-batch Contrastive Losses

A contrastive loss is used to pull the representations of positive pairs closer together, while pushing apart those of negative pairs in the embedding space. One of the most widely used contrastive losses is the so-called InfoNCE loss, which operates over samples within a mini-batch. Below, we illustrate its use in two well-known contrastive learning methods and discuss its limitations.

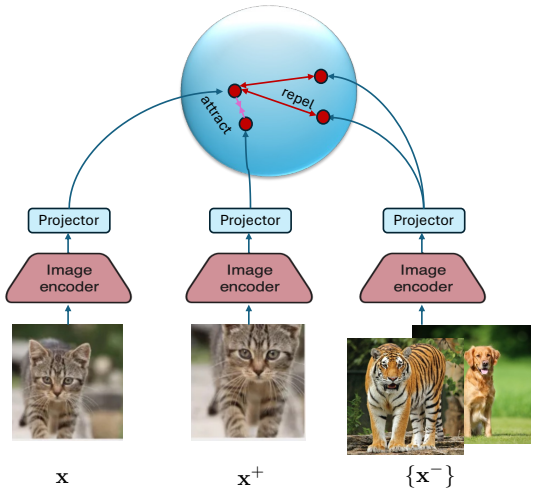


Fig. 6.18: Illustration of SimCLR for Contrastive Visual Representation Learning. $(\mathbf{x}, \mathbf{x}^+)$ are augmentations of the same image, $\{\mathbf{x}^-\}$ is a set of other images. An image encoder is a deep neural network and a projector is a lightweight multi-layer perceptron.

SimCLR

We now illustrate the contrastive loss in the context of visual representation learning by the well-known method SimCLR. The framework is illustrated in Figure 6.18. The model typically consists of a deep encoder backbone followed by a small projector, often implemented as a multi-layer perceptron (MLP). During downstream tasks, the projector is discarded, and the encoder's output is used as the final representation. The inclusion of the projector during training improves the quality and transferability of the learned embeddings by helping disentangle the contrastive learning objective from the representation space.

Let $(\mathbf{x}, \mathbf{x}^+) \sim \mathbb{P}_+$ denote a positive pair, which are different augmented copies from the same data. For a mini-batch $\mathcal{B} = \{\mathbf{x}_1, \ldots, \mathbf{x}_B\}$, each anchor $\mathbf{x}_i$ is paired with an augmented positive sample $\mathbf{x}_i^+$. The resulting mini-batch-based contrastive loss (commonly referred to as the InfoNCE loss) for anchor $\mathbf{x}_i$ is given by:

$$L_{\mathcal{B}}(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+) = -\log \frac{\exp\left(\frac{h(\mathbf{w};\mathbf{x}_i)^\top h(\mathbf{w};\mathbf{x}_i^+)}{\tau}\right)}{\exp\left(\frac{h(\mathbf{w};\mathbf{x}_i)^\top h(\mathbf{w};\mathbf{x}_i^+)}{\tau}\right) + \sum_{\mathbf{x}_j \in \mathcal{B}_i^-} \exp\left(\frac{h(\mathbf{w};\mathbf{x}_i)^\top h(\mathbf{w};\mathbf{x}_j)}{\tau}\right)}, \tag{6.38}$$

where $h(\mathbf{w}; \mathbf{x})$ denotes the normalized embedding of input $\mathbf{x}$, i.e., $\|h(\mathbf{w}; \mathbf{x})\|_2 = 1$, and $\tau > 0$ is the temperature parameter. The set $\mathcal{B}_i^-$ includes all negative samples in

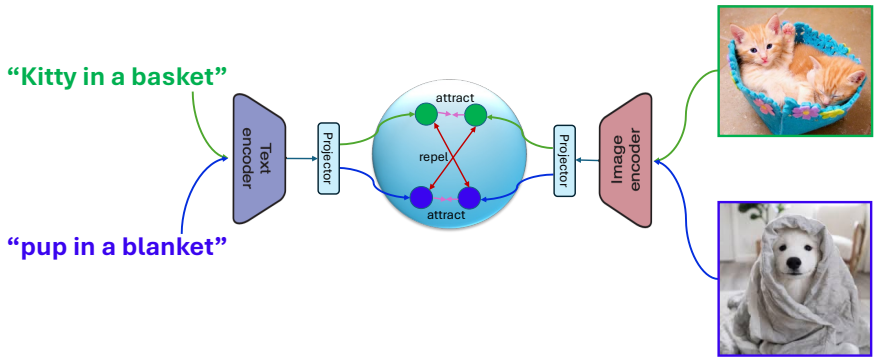


Fig. 6.19: Illustration of Contrastive Language-Image Pretraining (CLIP). A projector is usually a single linear layer.

the mini-batch excluding $\mathbf{x}_i$ and its augmentations. The positive pair can be removed from the denominator.

CLIP (Contrastive Language–Image Pretraining)

CLIP is a multimodal representation model that aligns images and text via contrastive learning on large-scale image–caption datasets. It comprises an image encoder and a text encoder, each followed by a corresponding projector, all jointly trained through contrastive learning (see Figure 6.19). CLIP models are typically trained on millions to billions of image–caption pairs, denoted as $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_n, \mathbf{t}_n)\}$. Let $h_1(\mathbf{w}; \cdot)$ denote the image encoder and $h_2(\mathbf{w}; \cdot)$ denote the text encoder, which outputs normalized embedding vectors.

With a mini-batch $\mathcal{B} = \{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_B, \mathbf{t}_B)\}$, a mini-batch-based contrastive loss for each image $\mathbf{x}_i$ is given by:

$$L_{\mathcal{B}}(\mathbf{w}; \mathbf{x}_i) = -\log \frac{\exp\left(\frac{h_1(\mathbf{w};\mathbf{x}_i)^\top h_2(\mathbf{w};\mathbf{t}_i)}{\tau}\right)}{\exp\left(\frac{h_1(\mathbf{w};\mathbf{x}_i)^\top h_2(\mathbf{w};\mathbf{t}_i)}{\tau}\right) + \sum_{\mathbf{t}_j \in \mathcal{B}_{2i}^-} \exp\left(\frac{h_1(\mathbf{w};\mathbf{x}_i)^\top h_2(\mathbf{w};\mathbf{t}_j)}{\tau}\right)}, \quad (6.39)$$

where the set $\mathcal{B}_{2i}^-$ includes all negative texts in the mini-batch excluding $\mathbf{t}_i$. Similarly, a mini-batch-based contrastive loss for each caption $\mathbf{t}_i$ is given by:

$$L_{\mathcal{B}}(\mathbf{w}; \mathbf{t}_i) = -\log \frac{\exp\left(\frac{h_1(\mathbf{w};\mathbf{x}_i)^\top h_2(\mathbf{w};\mathbf{t}_i)}{\tau}\right)}{\exp\left(\frac{h_1(\mathbf{w};\mathbf{x}_i)^\top h_2(\mathbf{w};\mathbf{t}_i)}{\tau}\right) + \sum_{\mathbf{x}_j \in \mathcal{B}_{1i}^-} \exp\left(\frac{h_1(\mathbf{w};\mathbf{x}_j)^\top h_2(\mathbf{w};\mathbf{t}_i)}{\tau}\right)}. \quad (6.40)$$

where the set $\mathcal{B}_{1i}^-$ includes all negative images in the mini-batch excluding $\mathbf{x}_i$. Back-propagation is then performed on the two mini-batch contrastive losses to compute gradient estimators, which are summed to update the model parameters.

CLIP enables zero-shot image classification, cross-modality retrieval and plays a crucial role in text-to-image generation by guiding models to synthesize images that semantically align with textual prompts.

What is zero-shot classification?

Zero-shot classification means classifying data without any labeled data for learning a classifier. In a multi-class classification task with K classes $\{C_1, \ldots, C_K\}$, where each class corresponds to a specific label (e.g., 'dog'), we apply the CLIP model by first constructing a natural language prompt for each category (e.g., 'a photo of a dog'). We then compute text embeddings for these prompts and calculate their cosine similarity with the image embedding generated by CLIP. Finally, the model predicts the class that yields the highest similarity score.

The Challenge of Large Batch Size

While efficient, the InfoNCE loss is known to heavily rely on large mini-batch sizes to ensure a rich and diverse set of negatives. For example, SimCLR requires a batch size of 8192 to achieve state-of-the-art performance for training on the ImageNet-1K dataset. This dependence on large batches imposes significant memory and computational burdens, especially when using large network backbones or processing high-dimensional inputs such as videos. Indeed, optimizing the InfoNCE loss is equivalent to using the BSGD method for optimizing the global contrastive loss as discussed in next subsection, which sufffers from non-convergence if the batch size is not significantly large.

6.5.2 Contrastive Learning without Large Mini-Batches

While the mini-batch contrastive loss offers computational convenience, it contradicts to the standard optimization principle where the objective is typically defined over the full dataset, followed by the development of efficient optimization algorithms. The mini-batch contrastive loss emerged naturally from the prevalent training pipeline (see Figure 6.1) that practitioners are familiar with. However, as previously discussed, this pipeline originating from ERM assumes that the loss for each data instance is independent of others, which does not hold for contrastive objectives. To resolve this, it is essential to decouple the design of the objective function from the optimization procedure.

Global Contrastive Loss: Separating Objective from Optimization

A global contrastive loss contrasts each anchor data point against all other examples in the training set. For a given positive pair $(\mathbf{x}_i, \mathbf{x}_i^+)$, the global contrastive loss is defined as:

$$L(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+) = \tau \log\left(\frac{1}{|\mathcal{S}_i^-|}\sum_{\mathbf{x}_j \in \mathcal{S}_i^-} \exp\left(\frac{h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_i^+)}{\tau}\right)\right), \tag{6.41}$$

where $\mathcal{S}_i^-$ is the set of all negative samples excluding $\mathbf{x}_i$ and its positive counterparts. The full global contrastive objective over $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is then given by:

$$\min_{\mathbf{w}} F(\mathbf{w}) = \frac{1}{n}\sum_{\mathbf{x}_i \in \mathcal{S}} \frac{1}{|\mathcal{S}_i^+|}\sum_{\mathbf{x}_i^+ \in \mathcal{S}_i^+} L(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+), \tag{6.42}$$

where $\mathcal{S}_i^+$ denotes the set of all positive samples corresponding to $\mathbf{x}_i$.

SogCLR: The Optimization Algorithm

To optimize the global contrastive objective, we cast it into the following:

$$\begin{aligned}\min_{\mathbf{w}} & -\frac{1}{n}\sum_{\mathbf{x}_i \in \mathcal{S}} \frac{1}{|\mathcal{S}_i^+|}\sum_{\mathbf{x}_i^+ \in \mathcal{S}_i^+} h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_i^+) \\ & + \frac{1}{n}\sum_{\mathbf{x}_i \in \mathcal{S}} \log\left(\sum_{\mathbf{x}_j \in \mathcal{S}_i^-} \exp\left(\frac{h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_j)}{\tau}\right)\right).\end{aligned} \tag{6.43}$$

The first term is a standard average and the second term is an objective of FCCO, where the outer function is $f(\cdot) = \tau \log(\cdot)$ and the inner function is $g_i(\mathbf{w}) = \frac{1}{|\mathcal{S}_i^-|}\sum_{\mathbf{z} \in \mathcal{S}_i^-} \exp\left(\frac{h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{z})}{\tau}\right)$. For readers who are familiar with Chapter 4 and 5, it is easy to understand the challenge of optimizing the above objective. It lies at the compositional structure of the second term with both summations over many data outside and inside the log function. As a result, the using the mini-batch-based InfoNCE loss will suffer from a biased gradient estimator whose error depends on the batch size.

To address this challenge, we can extend the SOX algorithm to solving (6.43) as shown in Algorithm 34, which is referred to as SogCLR. The estimators $u_{i,t+1}, \forall i$ are for tracking the inner function values $g_i(\mathbf{w}_t)$ and $p_{i,t} = \frac{1}{\varepsilon + u_{i,t+1}}$ is for estimating $\nabla \log(g_i(\mathbf{w}_t))$, where ε is small positive value added to avoid numerical issue and facilitate the learning.

Algorithm 34 SogCLR for optimizing the global contrastive objective (6.43)

Input: Initial model $\mathbf{w}_1$, $\mathbf{u}_0 \in \mathbb{R}^n$
for $t = 1$ to T **do**
  Sample a mini-batch $\mathcal{B} = \{\mathbf{x}_i\}_{i=1}^B$ with augmentations
  for each $\mathbf{x}_i \in \mathcal{B}$ **do**
    Construct the positive and negative set within mini-batch $\mathcal{B}_i^+$, $\mathcal{B}_i^-$
    Update $u_{i,t}$ via:

$$u_{i,t} = (1-\gamma)u_{i,t-1} + \gamma \frac{1}{|\mathcal{B}_i^-|} \sum_{\mathbf{z}\in\mathcal{B}_i^-} \exp\left(\frac{h(\mathbf{w}_t;\mathbf{x}_i)^\top h(\mathbf{w}_t;\mathbf{z})}{\tau}\right)$$

  end for
  Compute the vanilla gradient estimator $\mathbf{z}_t$:

$$\mathbf{z}_t = -\frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}_i\in\mathcal{B}} \frac{1}{|\mathcal{B}_i^+|} \sum_{\mathbf{x}_i^+\in\mathcal{B}_i^+} \nabla(h(\mathbf{w}_t;\mathbf{x}_i)^\top h(\mathbf{w}_t;\mathbf{x}_i^+))$$
$$+ \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}_i\in\mathcal{B}} \frac{1}{|\mathcal{B}_i^-|} \sum_{\mathbf{z}\in\mathcal{B}_i^-} \frac{\exp\left(\frac{h(\mathbf{w}_t;\mathbf{x}_i)^\top h(\mathbf{w}_t;\mathbf{z})}{\tau}\right)}{\varepsilon + u_{i,t}} \nabla(h(\mathbf{w};\mathbf{x}_i)^\top h(\mathbf{w};\mathbf{z})),$$

  Update $\mathbf{w}_{t+1}$ by Momentum, Adam or AdamW
end for

Initialization and Update of u

Unlike the model parameter $\mathbf{w}$, which is typically initialized randomly, the auxiliary variables $\mathbf{u}$ can be initialized upon their first update. Specifically, when an index i is sampled for the first time, we set $\mathbf{u}_{i,t}$ to the corresponding mini-batch estimate of the inner function value.

As with the practical considerations discussed for distributionally robust optimization (DRO), the vanilla update of $\mathbf{u}$ can suffer from numerical instability due to the use of $\exp(\cdot)$, particularly when the temperature τ is small. To address this, we can instead maintain a log-transformed variable $\nu_{i,t} = \log u_{i,t}$, following the technique in Equation (6.14).

PyTorch Implementation

A PyTorch implementation of SogCLR for self-supervised visual representation learning is shown in Figure 6.21. Each image in the dataset is augmented twice. To facilitate the computation of the vanilla gradient estimator, we define a dynamic contrastive loss function. For each augmented instance, we call this loss function to update its associated u variable and compute the dynamic loss using the updated u. These individual dynamic losses are then aggregated over the mini-batch, and the u variables for the two augmentations of each image are averaged.

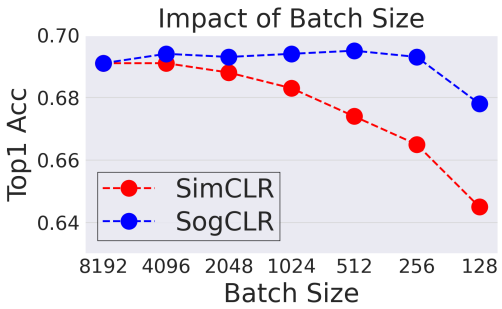


Fig. 6.20: Impact of batch size for different methods. The x-axis represents the batch size, and the y-axis shows the linear evaluation accuracy on the ImageNet validation set. Models were pretrained for 800 epochs using a ResNet-50 backbone on ImageNet-1K. For more details, please refer to (Yuan et al., 2022c).

Finally, we invoke `loss.backward()` to compute the gradient, followed by an optimizer step to update model parameters.

Comparison with SimCLR

The effectiveness of SogCLR is illustrated in Figure 6.20 with comparison with SimCLR for self-supervised visual representation learning on ImageNet-1K dataset with 1.2 million of images. With a standard mini-batch size 256 and the same other settings as SimCLR, by running 800 epochs, SogCLR achieves a performance of 69.4% for top 1 linear evaluation accuracy, which is better than 69.3% of SimCLR using a large batch size 8,192. Linear evaluation accuracy is measured by training a linear classifier atop a frozen encoder and subsequently assessing its performance on the validation set.

6.5.3 Contrastive Learning with Learnable Temperatures

The temperature parameter τ plays a critical role in controlling the penalty strength on negative samples. Specifically, a small τ penalizes much more on hard negative samples (i.e., the degree of hardness-awareness is high), causing separable embedding space. However, the excessive pursuit to the separability may break the underlying *semantic structures* because some negative samples with high similarity scores to the anchor data might indeed contain similar semantics, to which we refer as false negatives. In contrast, a large τ tends to treat all negative pairs equally (i.e., the degree of hardness-awareness is low) and is more tolerant to false negative samples, which is beneficial for keeping local semantic structures.

Existing approaches based on the InfoNCE loss often treat the temperature parameter τ as a learnable scalar to be optimized. However, this strategy lacks theoretical justification and may not yield optimal performance. Moreover, real-world data distributions typically exhibit long-tail characteristics, with substantial variation in the

```
# Note: This is a simplified version of SogCLR, we compute u
# from each augmentation separately for computing the dynamic
    contrastive loss
# and then  aggregated them from all augmentations.
# model: encoder + mlp projectors
# aug: a set of augmentation functions
# tau: temperature
# N: data size
# ind: indices for images in mini-batch
# u: 1d tensor with shape (N,1) by zero initialization
# g: parameter for maintaining moving averages of u

for ind, img in dataloader:
    x1, x2 = aug(img), aug(img)    # augmentations
    h1, h2 = model(x1), model(x2)  # forward pass
    h1, h2 = h1.norm(dim=1, p=2), h2.norm(dim=1, p=2)
    loss1, u1 = dcl(h1, h2, ind)   # dcl for h1, h2
    loss2, u2 = dcl(h2, h1, ind)   # dcl for h2, h1
    u[ind] = (u1 + u2)/2           # update u
    loss = (loss1 + loss2).mean()  # symmetrized
    loss.backward()
    update(model.params)           # momentum or adam-style

# dynamic contrastive loss (mini-batch)
def dcl(h1, h2, ind):
    B = h1.shape[0]
    labels = cat([one_hot(range(B)), one_hot(range(B))], dim=1)
    logits = cat([dot(h1, h2.T), dot(h1, h1.T)], dim=1)
    neg_logits = exp(logits/tau)*(1-labels)
    u_ = (1-g) * u[ind] + g*sum(neg_logits, dim=1)/(2(B-1))
    p = (neg_logits/u_).detach()
    sum_neg_logits = sum(p*logits, dim=1)/(2(B-1))
    normalized_logits = logits - sum_neg_logits
    loss = -sum(labels * normalized_logits, dim=1)
    return loss, u_
```

Fig. 6.21: PyTorch-style implementation of SogCLR for global contrastive learning.

frequency of samples across different semantic categories. This diversity suggests the need for individualized temperature parameters that better adapt to the inherent heterogeneity of the data.

To improve feature qualities, samples with frequent semantics should be assigned with a *large* τ to better capture the local semantic structure, while using a small τ will push semantically consistent samples away. On the other hand, samples with rare semantics should have a *small* τ to make their features more discriminative and separable.

Robust Global Contrastive Loss with a Learnable Temperature

Owing to the equivalence between the global contrastive loss and KL-regularized DRO (see Eq. (2.14)), the loss in Eq. (6.41) can be rewritten as:

$$L(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+) = \max_{\mathbf{p} \in \Delta} \sum_{\mathbf{x}_j \in \mathcal{S}_i^-} p_j \left(h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_i^+) \right) - \tau \, \mathrm{KL}(\mathbf{p}, 1/|\mathcal{S}_i^-|), \tag{6.44}$$

where Δ is the probability simplex over $\mathcal{S}_i^-$ and τ serves as the regularization parameter in the KL-regularized DRO.

To enable learning of the temperature parameter, we formulate a robust global contrastive loss using a KL-constrained DRO framework:

$$\begin{aligned} &\hat{L}(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+) = \\ &\max_{\mathbf{p} \in \Delta} \sum_{\mathbf{x}_j \in \mathcal{S}_i^-} p_j \left(h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_i^+) \right) - \tau_0 \, \mathrm{KL}(\mathbf{p}, 1/|\mathcal{S}_i^-|) \\ &\text{subject to} \quad \mathrm{KL}(\mathbf{p}, 1/|\mathcal{S}_i^-|) \le \rho, \end{aligned} \tag{6.45}$$

where τ_0 is a small constant to ensure smoothness of $\hat{L}(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+)$. Using the dual formulation (cf. Eq. (2.19)), this can be equivalently expressed as:

$$\hat{L}(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+) = \min_{\tau \ge \tau_0} \tau \log \left(\frac{1}{|\mathcal{S}_i^-|} \sum_{\mathbf{x}_j \in \mathcal{S}_i^-} \exp \left(\frac{h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_i^+)}{\tau} \right) \right) + \tau \rho. \tag{6.46}$$

Let $\ell_i(\mathbf{w}; \mathbf{x}_j) = h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_j) - h(\mathbf{w}; \mathbf{x}_i)^\top h(\mathbf{w}; \mathbf{x}_i^+)$. The above loss simplifies further to:

$$\hat{L}(\mathbf{w}; \mathbf{x}_i, \mathbf{x}_i^+) = \min_{\tau \ge \tau_0} \tau \log \left(\frac{1}{|\mathcal{S}_i^-|} \sum_{\mathbf{x}_j \in \mathcal{S}_i^-} \exp \left(\frac{\ell_i(\mathbf{w}; \mathbf{x}_j)}{\tau} \right) \right) + \tau \rho.$$

Minimizing the average of these robust global contrastive losses yields the following objective, which learns individualized temperatures:

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{\mathbf{x}_i \in \mathcal{S}} \left\{ \min_{\tau_i \ge \tau_0} \tau_i \log \left(\frac{1}{|\mathcal{S}_i^-|} \sum_{\mathbf{x}_j \in \mathcal{S}_i^-} \exp \left(\frac{\ell_i(\mathbf{w}; \mathbf{x}_j)}{\tau_i} \right) \right) + \tau_i \rho \right\}. \tag{6.47}$$

The SogCLR algorithm can be modified to solve this problem. We present the resulting algorithm, referred to as iSogCLR, in Algorithm 35. The vanilla gradient estimator with respect to $\mathbf{w}_t$ is computed as in SogCLR, except that the temperature τ is replaced with the individualized $\tau_{i,t}$ at iteration t. The gradient estimator with

Algorithm 35 iSogCLR for optimizing the robust global contrastive objective (6.47)

1: **Input:** Initial model $\mathbf{w}_1$, $\mathbf{u}_0 \in \mathbb{R}^n$
2: **for** $t = 1$ to T **do**
3: Sample a mini-batch $\mathcal{B} = \{\mathbf{x}_i\}_{i=1}^B$ with augmentations
4: **for** each $\mathbf{x}_i \in \mathcal{B}$ **do**
5: Construct the positive and negative set within mini-batch B_i^+, B_i^-
6: Update $u_{i,t}$ via:

$$u_{i,t} = (1-\gamma)u_{i,t-1} + \gamma \frac{1}{|\mathcal{B}_i^-|} \sum_{\mathbf{z}\in\mathcal{B}_i^-} \exp\left(\frac{\ell_i(\mathbf{w};\mathbf{z})}{\tau_{i,t}}\right)$$

7: Compute the vanilla gradient estimator $\mathbf{z}_{i,t}$ of $\tau_{i,t}$

$$\mathbf{z}_{i,t} = -\frac{1}{|\mathcal{B}_i^-|} \sum_{\mathbf{z}\in\mathcal{B}_i^-} \frac{\exp\left(\frac{\ell_i(\mathbf{w};\mathbf{z})}{\tau_{i,t}}\right)}{\varepsilon + u_{i,t}} \frac{\ell_i(\mathbf{w};\mathbf{z})}{\tau_{i,t}} + \log(u_{i,t}) + \rho$$

8: **end for**
9: Compute the vanilla gradient estimators $\mathbf{z}_t$:

$$\mathbf{z}_t = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x}_i\in\mathcal{B}} \frac{1}{|\mathcal{B}_i^-|} \sum_{\mathbf{z}\in\mathcal{B}_i^-} \frac{\exp\left(\frac{\ell_i(\mathbf{w}_t;\mathbf{z})}{\tau}\right)}{\varepsilon + u_{i,t}} \nabla \ell_i(\mathbf{w}_t;\mathbf{z}),$$

10: Update $\tau_{i,t+1}$, $\forall \mathbf{x}_i \in \mathcal{B}$ by the Momentum method
11: Update $\mathbf{w}_{t+1}$ by the Momentum or AdamW method
12: **end for**

respect to $\tau_{i,t}$ is computed in Step 7 and it can be updated using the Momentum method.

An application of iSogCLR to CIFAR-10 dataset yields more discriminative features than SimCLR and SogCLR as shown in Figure 6.22.

CLIP Training with Learnable Temperatures

CLIP with Individualized Learnable Temperatures

We can integrate the robust global contrastive loss for temperature learning into the contrastive language-image pretraining (CLIP), yielding the following objective:

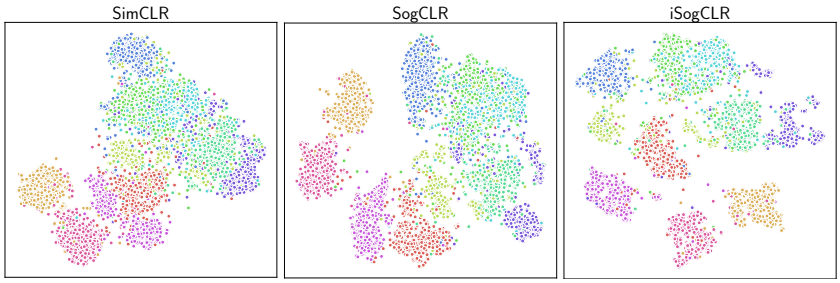


Fig. 6.22: The learned embeddings (projected onto 2D space using t-SNE) for CIFAR10 samples learned by self-supervised learning algorithms SimCLR, SogCLR and iSogCLR. For more details, please refer to (Qiu et al., 2023).

$$\min_{\mathbf{w},\boldsymbol{\tau}_1\geq\tau_0,\boldsymbol{\tau}_2\geq\tau_0} \frac{1}{n}\sum_{i=1}^{n}\tau_{i,1}\log\left(\frac{1}{|\mathcal{T}_i^-|}\sum_{\mathbf{t}\in\mathcal{T}_i^-}\exp\left(\frac{s(\mathbf{w};\mathbf{x}_i,\mathbf{t})-s(\mathbf{w};\mathbf{x}_i,\mathbf{t}_i)}{\tau_{i,1}}\right)\right)+\tau_{i,1}\rho$$
$$+\frac{1}{n}\sum_{i=1}^{n}\tau_{i,2}\log\left(\frac{1}{|\mathcal{I}_i^-|}\sum_{\mathbf{x}\in\mathcal{I}_i^-}\exp\left(\frac{s(\mathbf{w};\mathbf{x},\mathbf{t}_i)-s(\mathbf{w};\mathbf{x}_i,\mathbf{t}_i))}{\tau_{i,2}}\right)\right)+\tau_{i,2}\rho, \tag{6.48}$$

where $\mathcal{T}_i^-$ denotes the set of all negative data of an image $\mathbf{x}_i$ and $\mathcal{I}_i^-$ denotes the set of all negative data of the corresponding text $\mathbf{t}_i$, and $s(\mathbf{w};\mathbf{x},\mathbf{t}) = h_1(\mathbf{w};\mathbf{x})^\top h_2(\mathbf{w};\mathbf{t})$ is the similarity score of the image and text embeddings.

While optimizing robust contrastive losses enables the learning of temperature parameters, it may compromise generalizability in downstream tasks by introducing a large number of additional parameters, which can lead to overfitting—particularly in noisy real-world datasets where mismatched samples are common. Two approaches can be used to tackle this issue.

CLIP with a Global Learnable Temperature

A straightforward approach to reduce the number of temperature parameters is to learn a single global temperature parameter for images and texts, respectively. This is formulated as the following optimization problem:

$$\min_{\mathbf{w},\tau_1\geq\tau_0,\tau_2\geq\tau_0} \frac{1}{n}\sum_{i=1}^{n}\left\{\tau_1\log\left(\frac{1}{|\mathcal{T}_i^-|}\sum_{\mathbf{t}\in\mathcal{T}_i^-}\exp\left(\frac{s(\mathbf{w};\mathbf{x}_i,\mathbf{t})-s(\mathbf{w};\mathbf{x}_i,\mathbf{t}_i)}{\tau_1}\right)\right)+\tau_1\rho\right\}$$
$$+\frac{1}{n}\sum_{i=1}^{n}\left\{\tau_2\log\left(\frac{1}{|\mathcal{I}_i^-|}\sum_{\mathbf{x}\in\mathcal{I}_i^-}\exp\left(\frac{s(\mathbf{w};\mathbf{x},\mathbf{t}_i)-s(\mathbf{w};\mathbf{x}_i,\mathbf{t}_i)}{\tau_2}\right)\right)+\tau_2\rho\right\}. \tag{6.49}$$

CLIP with a Temperature Prediction Network

An alternative strategy is to learn a temperature prediction network (TempNet) that outputs an instance-dependent temperature for each image and text. The corresponding optimization problem is defined as:

$$\min_{\mathbf{w},\mathbf{w}_1',\mathbf{w}_2'} \frac{1}{n}\sum_{i=1}^{n} \tau(\mathbf{w}_1';\mathbf{x}_i)\log\left(\frac{1}{|\mathcal{T}_i^-|}\sum_{\mathbf{t}\in\mathcal{T}_i^-}\exp\left(\frac{s(\mathbf{w};\mathbf{x}_i,\mathbf{t})-s(\mathbf{w};\mathbf{x}_i,\mathbf{t}_i)}{\tau(\mathbf{w}_1';\mathbf{x}_i)}\right)\right)+\tau(\mathbf{w}_1';\mathbf{x}_i)\rho$$
$$+\frac{1}{n}\sum_{i=1}^{n} \tau(\mathbf{w}_2';\mathbf{t}_i)\log\left(\frac{1}{|\mathcal{I}_i^-|}\sum_{\mathbf{x}\in\mathcal{I}_i^-}\exp\left(\frac{s(\mathbf{w};\mathbf{x},\mathbf{t}_i)-s(\mathbf{w};\mathbf{x}_i,\mathbf{t}_i)}{\tau(\mathbf{w}_2';\mathbf{t}_i)}\right)\right)+\tau(\mathbf{w}_2';\mathbf{t}_i)\rho. \tag{6.50}$$

The temperature prediction network $\tau(\mathbf{w}_1';\cdot)$ for images can share the encoder layers of the image encoder $h_1(\mathbf{w};\cdot)$, followed by a lightweight MLP. Similarly, the text-side temperature prediction network $\tau(\mathbf{w}_2';\cdot)$ can share the encoder layers of the text encoder $h_2(\mathbf{w};\cdot)$, also followed by a small MLP. Again this problem can be optimized by modifying SogCLR to account for the update of TempNet.

Scheduler of γ

Like the standard learning rate η in the update of $\mathbf{w}_{t+1}$, the hyper-parameter γ can be also interpreted as a learning rate of SGD (4.3). The theoretical analysis shows that γ should be set to a very small value close to 0 in order to guarantee convergence. Ideally, γ should be large to rely more on the current mini-batch at earlier iterations and be smaller to rely more on history in later iterations. To achieve this, we can use a decreasing scheduler, e.g., a cosine schedule for γ_t: Let t be the current iteration, t_0 be the number of iterations per epoch and E be the number of decay epochs, then we set $\gamma_t = 0.5\cdot(1+\cos(\pi\lfloor t/t_0\rfloor/E))\cdot(1-\gamma_{\min})+\gamma_{\min}$. With this schedule, γ_t will decrease from 1.0 to $\gamma_{\min}$. Note that $\lfloor t/t_0\rfloor$ denotes the current epoch, which means the value of γ_t stays unchanged within one epoch. Also, The number of decay epochs E is a hyperparameter, and it is not necessarily equal to the total number of training epochs. If the current epoch exceeds E, γ_t will be set to $\gamma_{\min}$.

PyTorch Implementations

PyTorch implementations of SogCLR and iSogCLR are available in the LibAUC library. Their distributed versions, including support for solving (6.49) with a cosine scheduler for γ, are provided in the FastCLIP GitHub repository:

https://github.com/Optimization-AI/FastCLIP

Three versions are available: FastCLIP-v1 implements SogCLR with a tuned global temperature, FastCLIP-v2 implements iSogCLR with individualized temperatures,

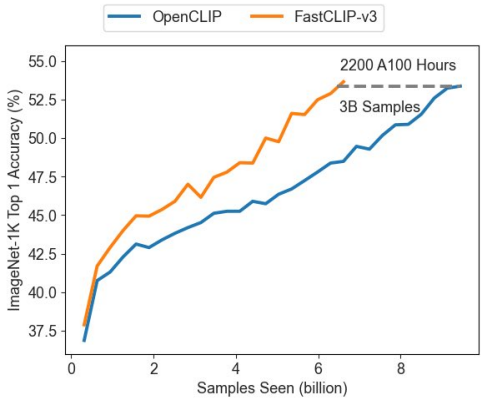


Fig. 6.23: FastCLIP-v3 vs OpenCLIP. The training was conducted on LAION315M with 315M image-text pairs for learning ViT-B/16 using a total of 5120 batch size on 8 A100. Y-axis is the zero-shot accuracy on ImageNet validation data. For more details, please refer to (Wei et al., 2024).

and FastCLIP-v3 implements SogCLR for solving the global temperature optimization in (6.49).

A distributed implementation of iSogCLR for CLIP training with the Temperature Prediction Network (TempNet) is available at:

`https://github.com/Optimization-AI/DistTempNet`

Figure 6.23 presents a comparison between FastCLIP-v3 and the prior state-of-the-art distributed implementation of optimizing the mini-batch-based InfoNCE loss, known as OpenCLIP (Ilharco et al., 2021). This highlights the effectiveness of the advanced compositional optimization algorithm, demonstrating clear improvements in both convergence speed and representation quality.

6.6 Discriminative Fine-tuning of Large Language Models

Large Language Models (LLMs) have revolutionized modern AI. Their training typically consists of three stages: self-supervised pretraining on internet-scale text corpora, supervised fine-tuning (SFT) on question–answer datasets, and learning with human preference for alignment. An improved paradigm, *reinforcement learning with verifiable rewards* (RLVR), further advances large reasoning models by leveraging automatically verifiable signals from synthesized outputs.

6.6.1 Pipeline of LLM Training

Figure 6.24 illustrates the pipeline of LLM Training. We briefly introduce these components below.

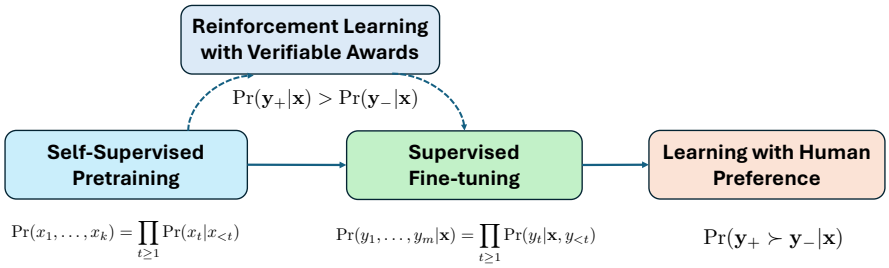


Fig. 6.24: Different Phases of training LLMs.

Self-supervised Pretraining

Self-supervised pretraining is formulated as next-token prediction. Let $\mathbf{x} = (x_1, \ldots, x_m)$ be a sequence of tokens where x_j belongs to a vocabulary of tokens $\mathcal{V} = \{v_1, \ldots, v_K\}$. The probability of $\mathbf{x}$ is modeled auto-regressively by

$$p(\mathbf{x}) = \prod_{j=1}^{m} p(x_j|x_{<j}),$$

where $x_{<j}$ denotes the prefix $(x_1, \ldots, x_{j-1})$. The conditional probability is modeled via a softmax over a Transformer representation:

$$p(x_j|x_{<j}) = \pi_{\mathbf{w}}(x_j|x_{<j}) = \frac{\exp(h(\mathbf{w}_0; x_{<j})^\top \mathbf{w}_{x_j})}{\sum_{k=1}^{K} \exp(h(\mathbf{w}_0; x_{<j})^\top \mathbf{w}_k)}, \tag{6.51}$$

where $h(\mathbf{w}_0; x_{<j}) \in \mathbb{R}^d$ is produced by a Transformer network and $\mathbf{w}_{x_j} \in \mathbb{R}^d$ is the token embedding. The full model parameters $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_K)$ are learned by minimizing the negative log-likelihood over a dataset $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$:

$$\min_{\mathbf{w}} -\frac{1}{n} \sum_{i=1}^{n} \log p(\mathbf{x}_i). \tag{6.52}$$

Supervised Fine-tuning (SFT)

In SFT, a dataset $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ is used, where $\mathbf{x}_i$ is an input prompt and $\mathbf{y}_i$ is the desired output. Let $\mathbf{x} = (x_1, \ldots, x_k)$ and $\mathbf{y} = (y_1, \ldots, y_{m'})$ be token sequences from the vocabulary $\mathcal{V}$. SFT models the next-token prediction of tokens in $\mathbf{y}$ given $\mathbf{x}$ using the autoregressive factorization:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{m'} \pi_{\mathbf{w}}(y_j|\mathbf{x}, y_{<j}),$$

where each term is computed using the same Transformer-based model as in pre-training. SFT minimizes:

$$\min_{\mathbf{w}} -\frac{1}{n}\sum_{i=1}^{n} \log p(\mathbf{y}_i|\mathbf{x}_i). \tag{6.53}$$

Learning with Human Preference

SFT does not penalize poor responses. Hence, it does not necessarily guarantee that the likelihood of tokens in a poor answer is low. Let us consider a simple example:

Motivation Example
($\mathbf{x}$) What is the bigger number between 9.11 and 9.9? ($\mathbf{y}$) The bigger number between 9.11 and 9.9 is 9.9. ($\mathbf{y}'$) The bigger number between 9.11 and 9.9 is 9.11.

The good answer $\mathbf{y}$ and the bad answer $\mathbf{y}'$ only differ in the last token. The likelihood of all preceding tokens are the same. Even though the likelihood of the last token "9" in $\mathbf{y}$ conditioned on preceding tokens is increased during the fine-tuning with this data, the likelihood of the token "11" as the last one might still be high, making generating the bad answer $\mathbf{y}'$ likely.

To address this issue, learning with human feedback fine-tunes the model using preference tuples $(\mathbf{x}, \mathbf{y}_+, \mathbf{y}_-)$, where $\mathbf{y}_+$ is preferred over $\mathbf{y}_-$. Two main approaches are reinforcement learning from human feedback (RLHF) and direct preference optimization (DPO).

RLHF

A reward model $r_\theta(\mathbf{x}, \mathbf{y})$ is first trained to match human preferences by modeling the preference probability $\Pr(\mathbf{y}_+ \succ \mathbf{y}_-|\mathbf{x})$ as

$$p(\mathbf{y}_+ \succ \mathbf{y}_-|\mathbf{x}) = \frac{\exp(r_\theta(\mathbf{x}, \mathbf{y}_+))}{\exp(r_\theta(\mathbf{x}, \mathbf{y}_+)) + \exp(r_\theta(\mathbf{x}, \mathbf{y}_-))}, \tag{6.54}$$

and minimizing the following:

$$\min_{\theta} \mathbb{E}_{\mathbf{x},\mathbf{y}_+,\mathbf{y}_-} - \log p(\mathbf{y}_+ \succ \mathbf{y}_-|\mathbf{x}). \tag{6.55}$$

The policy model (i.e, the target LLM) is then optimized by solving the following problem with some RL algorithms:

$$\max_{\mathbf{w}} \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\pi_{\mathbf{w}}} \left[r_{\theta_*}(\mathbf{x}, \mathbf{y}) - \beta \mathrm{KL}(\pi_{\mathbf{w}}(\cdot|\mathbf{x}), \pi_{\mathrm{ref}}(\cdot|\mathbf{x})) \right]. \tag{6.56}$$

where the KL divergence is defined as:

$$\mathrm{KL}(\pi_{\mathbf{w}}(\cdot|\mathbf{x}), \pi_{\mathrm{ref}}(\cdot|\mathbf{x})) = \mathbb{E}_{\mathbf{y}\sim\pi_{\mathbf{w}}(\cdot|\mathbf{x})}\left[\log \frac{\pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})}{\pi_{\mathrm{ref}}(\mathbf{y}|\mathbf{x})}\right], \tag{6.57}$$

where π_{ref} denotes a base model. If we decompose $\mathbf{y} = (y_1, \ldots, y_k)$ as a sequence of tokens, then using the autoregressive factorization the KL divergence can be expressed as a sum over tokens:

$$\mathrm{KL}(\pi_{\mathbf{w}}(\cdot|\mathbf{x}), \pi_{\mathrm{ref}}(\cdot|\mathbf{x})) = \mathbb{E}_{\mathbf{y}\sim\pi_{\mathbf{w}}}\left[\sum_{t=1}^{k} \log \frac{\pi_{\mathbf{w}}(y_t|\mathbf{x}, y_{<t})}{\pi_{\mathrm{ref}}(y_t|\mathbf{x}, y_{<t})}\right]. \tag{6.58}$$

Direct Preference Optimization (DPO)

DPO directly optimizes the policy without a separate reward model. A closed-form non-parameterized solution of π by solving (6.56) for any reward model $r(\mathbf{x}, y)$, gives:

$$\pi(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\pi_{\mathrm{ref}}(\mathbf{y}|\mathbf{x}) \exp(\beta r(\mathbf{x}, \mathbf{y})), \tag{6.59}$$

where $Z(\mathbf{x})$ is the normalization factor. Substituting into Eq. (6.55) leads to:

$$\min_{\mathbf{w}} \mathbb{E}_{\mathbf{x},\mathbf{y}_+,\mathbf{y}_-} \log\left(1 + \exp\left(\beta \log \frac{\pi_{\mathbf{w}}(\mathbf{y}_-|\mathbf{x})}{\pi_{\mathrm{ref}}(\mathbf{y}_-|\mathbf{x})} - \beta \log \frac{\pi_{\mathbf{w}}(\mathbf{y}_+|\mathbf{x})}{\pi_{\mathrm{ref}}(\mathbf{y}_+|\mathbf{x})}\right)\right). \tag{6.60}$$

In practice, a set of tuples $\{(\mathbf{x}_i, \mathbf{y}_{i+}, \mathbf{y}_{i-}\}_{i=1}^n$ is constructed and used for learning.

Connections with Discriminative Learning and AUC Maximization

DPO can be also motivated from discriminative learning, particularly AUC maximization. We view generating the answers of $\mathbf{x}$ as a task, and $\mathbf{y}_+$ denotes a positive data and $\mathbf{y}_-$ denotes a negative data. Let $s(\mathbf{w}, \mathbf{x}, \mathbf{y})$ denote a scoring function, which indicates the likelihood of generating $\mathbf{y}$ given $\mathbf{x}$. By AUC maximization with a continuous surrogate loss $\ell(s(\mathbf{w}, \mathbf{x}, \mathbf{y}_-) - s(\mathbf{w}, \mathbf{x}, \mathbf{y}_+))$, we have the following problem:

$$\min_{\theta} \mathbb{E}_{\mathbf{x},\mathbf{y}_+,\mathbf{y}_-} \ell(s(\mathbf{w}, \mathbf{x}, \mathbf{y}_-) - s(\mathbf{w}, \mathbf{x}, \mathbf{y}_+)). \tag{6.61}$$

DPO can be recovered by setting $s(\mathbf{w}, \mathbf{x}, \mathbf{y}) = \log \frac{\pi(\mathbf{y}|\mathbf{x})}{\pi_{\mathrm{ref}}(\mathbf{y}|\mathbf{x})}$ and $\ell(s) = \log(1 + \exp(\beta s))$.

Reinforcement Learning with Verifiable Rewards (RLVR)

RLVR is an emerging paradigm for training reasoning models, particularly suited for tasks like mathematical problem solving, where models are expected to generate step-by-step solutions followed by a final answer. Unlike RLHF, which relies on

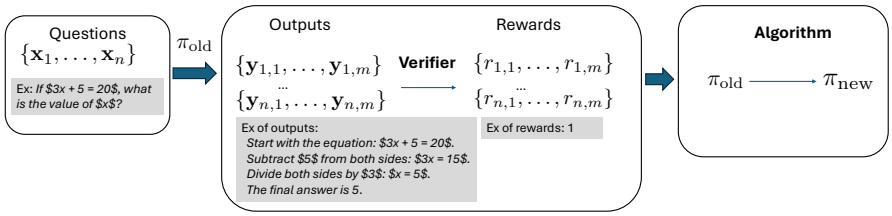


Fig. 6.25: The one-step iteration of RL for reinforcing Large Reasoning Model. For each question $\mathbf{x}_i$, the model generates m outputs $\mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,m}$ and each of them receives a reward $r_{i,j}$, $j = 1, \ldots, m$ from a verifier. Then an algorithm will leverage the inputs, their outputs and the reward information to update the model.

subjective preference labels, RLVR leverages verifiable signals such as whether the final answer is correct.

What is a Large Reasoning Model?

A large reasoning model is a type of LLM that is specifically designed or fine-tuned to perform multi-step logical reasoning, such as solving math problems, answering complex questions, or generating structured arguments. It generates intermediate reasoning tokens before producing the final answer, mimicking System 2 reasoning in humans, which is deliberate, logical, and slow.

RLVR is illustrated in Figure 6.25. The old model in one step of learning is denoted by π_{old}. It is used to generate multiple answers for a set of input questions. Given a question $\mathbf{x}$ (with prompt included), one generated output $\mathbf{y}$ follows the distribution $\pi_{\text{old}}(\cdot|\mathbf{x})$, which includes reasoning traces and the final answer. Specifically, output $\mathbf{y}$ is generated token by token, i.e., $y_t \sim \pi_{\text{old}}(\cdot|\mathbf{x}, y_{<t})$, for $t = 1, \cdots, |\mathbf{y}|$.

A key to RLVR is to assume that there exists a verifier, which can automatically verifies the quality of the generated answer, giving a reward. Let us consider a binary reward setting where the verifier returns a binary value for a given question $\mathbf{x}$ and its corresponding answer in the output $\mathbf{y}$. For answering mathematical questions, this can be achieved by comparing the generated answer with the true answer. For generating mathematical proofs, we can use a formal verification tool such as LEAN to verify if the proof is correct.

Proximal Policy Optimization (PPO)

PPO is a classical RL algorithm. Let

$$\rho_{\mathbf{w}}(\mathbf{x}, \mathbf{y}) = \frac{\pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})}{\pi_{\text{old}}(\mathbf{y}|\mathbf{x})}$$

denote the likelihood ratio between the new policy $\pi_{\mathbf{w}}$ and the old policy π_{old}. Let $A(\mathbf{x}, \mathbf{y})$ be an advantage function for taking action $\mathbf{y}$ given input $\mathbf{x}$, which measures how much better a specific action is compared to the policy's average behavior in a given state. The PPO objective is given by:

$$\begin{aligned}\mathcal{L}_{\text{PPO}}(\mathbf{w}) =& \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\pi_{\text{old}}}\left[\min\left(\rho_{\mathbf{w}}(\mathbf{x},\mathbf{y})\cdot A(\mathbf{x},\mathbf{y}),\ \text{clip}(\rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}), 1-\epsilon, 1+\epsilon)\cdot A(\mathbf{x},\mathbf{y})\right)\right], \\ & - \beta \text{KL}(\pi_{\mathbf{w}}, \pi_{\text{ref}}),\end{aligned} \tag{6.62}$$

where $\epsilon > 0$ is a small hyperparameter (typically around 0.1 or 0.2), and the `clip` function restricts the likelihood ratio $\rho_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$ to the range $[1-\epsilon, 1+\epsilon]$, defined as:

$$\text{clip}(\rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}), 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon & \text{if } \rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}) < 1-\epsilon, \\ \rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}) & \text{if } 1-\epsilon \le \rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}) \le 1+\epsilon, \\ 1+\epsilon & \text{if } \rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}) > 1+\epsilon. \end{cases}$$

The intuition of using clipping mechanism is that

- When $A(\mathbf{x}, \mathbf{y}) > 0$ (the action is better than expected), the clip operation prevents $\pi_{\mathbf{w}}$ from increasing its probability too aggressively.
- When $A(\mathbf{x}, \mathbf{y}) < 0$ (the action is worse than expected), the clip operation prevents $\pi_{\mathbf{w}}$ from decreasing its probability too drastically.

This clipping mechanism was used to reduce variance and maintain stable training dynamics for reinforcement learning. However, it also suffers from zero gradient when $\rho_{\mathbf{w}}(\mathbf{x}, \mathbf{y})$ is out of the range $[1-\epsilon, 1+\epsilon]$, which might slow down the learning process.

Trust Region Policy Optimization (TRPO)

TRPO is a principled policy optimization method that improves stability and efficiency by restricting each policy update to stay within a small trust region. It maximizes a surrogate objective function based on the advantage estimates under the old policy, while constraining the average Kullback–Leibler (KL) divergence between the old and new policies. Formally, TRPO solves the following constrained optimization problem:

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\pi_{\text{old}}}\left[\rho_{\mathbf{w}}(\mathbf{x},\mathbf{y}) A(\mathbf{x},\mathbf{y})\right] \\ \text{subject to} \quad & \mathbb{E}_{\mathbf{x}}\left[\text{KL}\left(\pi_{\text{old}}(\cdot|\mathbf{x}), \pi_{\mathbf{w}}(\cdot|\mathbf{x})\right)\right] \le \delta, \end{aligned} \tag{6.63}$$

where δ is a predefined trust region threshold. The KL divergence is taken in the reverse direction to ensure that the updated policy does not deviate too much from the old policy on average across the state distribution.

Group Relative Policy Optimization (GRPO).

GRPO is a reinforcement learning algorithm designed to optimize policies by leveraging group-wise relative reward information.

For inputs $\{\mathbf{x}_i\}_{i=1}^m$, let $\{\mathbf{y}_{ij}\}_{j=1}^K$ denote the corresponding set of K generated answers for each $\mathbf{x}_i$. the objective of GRPO for maximization is defined by:

$$\begin{aligned}\mathcal{J}_{\text{GRPO}}(\mathbf{w}) =& \frac{1}{m}\sum_{i=1}^{m}\frac{1}{k}\sum_{j=1}^{k}\left[\frac{1}{|\mathbf{y}_{ij}|}\sum_{t=1}^{|\mathbf{y}_{ij}|} f\left(\frac{\pi_{\mathbf{w}}(y_{ij,t}|\mathbf{x}, y_{ij,<t})}{\pi_{\text{old}}(y_{ij,t}|\mathbf{x}, y_{ij,<t})}, A(\mathbf{x}_i, \mathbf{y}_{ij})\right)\right] \\ & - \beta \text{KL}(\pi_\theta, \pi_{\text{ref}}),\end{aligned} \tag{6.64}$$

where $y_{ij,t}$ denotes its t-th token and $y_{ij,<t}$ denotes the prefix of the t-th token of $\mathbf{y}_{ij}$, $f(s,t) = \min(st, \text{clip}(s, 1-\epsilon, 1+\epsilon)t)$, π_{ref} is a frozen reference model, and $A(\mathbf{x}_i, \mathbf{y}_{ij})$ is the group-wise advantage function defined as

$$A(\mathbf{x}, \mathbf{y}) = \frac{r(\mathbf{y}|\mathbf{x}) - \bar{r}_q}{\sigma_q}$$

with $\bar{r}_q$ being the average reward of outputs for $\mathbf{x}$ and σ_q being its standard deviation. This advantage function quantifies how much better the reward of an output $\mathbf{y}$ is compared to average reward in the group. For analysis, we consider the expected version:

$$\mathcal{J}_{\text{GRPO}}(\mathbf{w}) = \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}\sim\pi_{\text{old}}(\cdot|\mathbf{x})}\left[\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x}, y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x}, y_{<t})}, A(\mathbf{x}, \mathbf{y})\right)\right] - \beta \text{KL}(\pi_\theta, \pi_{\text{ref}}), \tag{6.65}$$

where

$$A(\mathbf{x}, \mathbf{y}) = \frac{r(\mathbf{y}|\mathbf{x}) - \mathbb{E}_{\mathbf{y}'\sim\pi_{\text{old}}(\cdot|\mathbf{x})} r(\mathbf{y}'|\mathbf{x})}{\sqrt{\text{Var}_{\mathbf{y}'\sim\pi_{\text{old}}(\cdot|\mathbf{x})} r(\mathbf{y}'|\mathbf{x})}}. \tag{6.66}$$

6.6.2 DFT for fine-tuning Large Language Models

While learning with human feedback addresses the limitation of SFT, traditional supervised learning methods never use human preference data. For example, in image classification, training data $(\mathbf{x}, y)$ denote an input image and its true class label $y \in \{1, \ldots, K\}$. We do not need the preference optimization step on preference data saying that a dog class is preferred to a cat class for an image of a dog. So what is the difference between traditional supervised learning and supervised finetuning of LLMs that makes SFT not enough? The answer lies in the fact that traditional supervised learning methods are usually **discriminative approaches**, while the SFT method is not discriminative.

By casting the supervised fine-tuning of LLMs into data prediction, we can leverage discriminative learning approaches, e.g., the discriminative probabilistic modeling (DPM) approach and the robust optimization approach.

DPM over an Infinite Data Space

Let $\mathcal{X}$ and $\mathcal{Y}$ be infinite data spaces. Let us consider $\mathcal{X}$ as an anchor space and $\mathcal{Y}$ as the target space with a Lebesgue measure μ. When $\mathcal{Y}$ is countably infinite, the Lebesgue measure μ is replaced by the counting measure. We model the probability density $\Pr(\mathbf{y} \mid \mathbf{x})$ of an object $\mathbf{y} \in \mathcal{Y}$ given an anchor object $\mathbf{x} \in \mathcal{X}$ by a parameterized scoring function $s(\mathbf{w}; \mathbf{x}, \mathbf{y})$:

$$P_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x}) = \frac{\exp(s(\mathbf{w}; \mathbf{x}, \mathbf{y})/\tau)}{\int_{\mathcal{Y}} \exp(s(\mathbf{w}; \mathbf{x}, \mathbf{y}')/\tau) d\mu(\mathbf{y}')}, \tag{6.67}$$

where $\tau > 0$ is a temperature parameter. We assume that $\exp(s(\mathbf{w}; \mathbf{x}, \mathbf{y})/\tau)$ is Lebesgue-integrable for $\mathbf{w} \in \mathcal{W}$, $\mathcal{W} \subset \mathbb{R}^d$. Here $P_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$ is a valid probability density function because $\int_{\mathcal{Y}} P_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x}) d\mu(\mathbf{y}) = 1$. Given $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ sampled from the joint distribution $p_{\mathbf{x},\mathbf{y}}$, the maximum likelihood estimation (MLE) can be formulated as the following:

$$\begin{aligned} &\min_{\mathbf{w}} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \tau \log \frac{\exp(s(\mathbf{w}; \mathbf{x}, \mathbf{y})/\tau)}{\int_{\mathcal{Y}} \exp(s(\mathbf{w}; \mathbf{x}, \mathbf{y}')/\tau) d\mu(\mathbf{y}')} \right\} \\ &= -\frac{1}{n} \sum_{i=1}^{n} s(\mathbf{w}; \mathbf{x}, \mathbf{y}) + \tau \log \left(\int_{\mathcal{Y}} \exp(s(\mathbf{w}; \mathbf{x}, \mathbf{y}')/\tau) d\mu(\mathbf{y}') \right). \end{aligned} \tag{6.68}$$

If $\mathcal{Y}$ is finite, the above DPM framework recovers the traditional multi-class classification and learning to rank. In particular, if $\mathcal{Y}$ denotes the label set $\{1, \ldots, K\}$ and $s(\mathbf{w}; \mathbf{x}, y)$ denotes the classification score for the y-th class, then the above approach recovers logistic regression. If $\mathcal{Y}$ denotes the set of items $\mathcal{Y} = \{\mathbf{x}_{q,1}, \ldots, \mathbf{x}_{q,N_q}\}$ and the anchor data $\mathbf{x}$ denotes a query, then the above approach recovers the ListNet (2.47).

Optimization via FCCO

The main challenge for solving the DPM problem over an infinite data space lies in computing the integral $g(\mathbf{w}; \mathbf{x}_i, \mathcal{Y}) \coloneqq \int_{\mathcal{Y}} \exp\left(s(\mathbf{w}; \mathbf{x}_i, \mathbf{y}')/\tau\right) d\mu(\mathbf{y}')$ for each $i \in [n]$, which is infeasible unless $\mathcal{Y}$ is finite. Below, we discuss two general approaches for tackling the challenge.

Sample and Optimize

The first approach is to introduce a sampling distribution $P_i(\cdot)$, satisfying that (1) it is easy to sample data from P_i; (2) it is possible to compute the probability value of a sample $\mathbf{y}'$. Then we write

$$\int_{\mathcal{Y}} \exp\left(\frac{s(\mathbf{w};\mathbf{x}_i,\mathbf{y}')}{\tau}\right) d\mu(\mathbf{y}') = \mathbb{E}_{\mathbf{y}'\sim P_i(\cdot)} \frac{\exp(s(\mathbf{w};\mathbf{x}_i,\mathbf{y}')/\tau)}{P_i(\mathbf{y}')}.$$

The optimization problem becomes an instance of FCCO:

$$\begin{aligned}\min_{\mathbf{w}} & -\frac{1}{n}\sum_{i=1}^n s(\mathbf{w};\mathbf{y}_i,\mathbf{x}_i) \\ & + \frac{1}{n}\sum_{i=1}^n \tau \log\left(\mathbb{E}_{\mathbf{y}'\sim P_i(\cdot)} \frac{\exp(s(\mathbf{w};\mathbf{y}',\mathbf{x}_i)/\tau)}{P_i(\mathbf{y}')}\right).\end{aligned} \tag{6.69}$$

Approximate and Optimize

In some cases, we may only have sampled data from $P_i(\cdot)$ without access to $P_i(\cdot)$. Let $\mathcal{S}_i = \{\mathbf{y}'_{i,1},\ldots,\mathbf{y}'_{i,m}\}$ denote a set of outputs sampled for each data $\mathbf{x}_i$ following some P_i. Then we approximate $g(\mathbf{w};\mathbf{x}_i,\mathcal{Y})$ by

$$g(\mathbf{w};\mathbf{x}_i,\mathcal{Y}) \approx \frac{1}{m}\sum_{\mathbf{y}'\in\mathcal{S}_i^-} \frac{\exp(s(\mathbf{w};\mathbf{y}',\mathbf{x})/\tau)}{P_i(\mathbf{y}')} \propto \frac{1}{m}\sum_{\mathbf{y}'\in\mathcal{S}_i} \exp(\frac{s(\mathbf{w};\mathbf{y}',\mathbf{x})}{\tau}), \tag{6.70}$$

where the last step assumes $P_i(\mathbf{y}')$ are approximately equal. Then the optimization problem becomes an instance of FCCO:

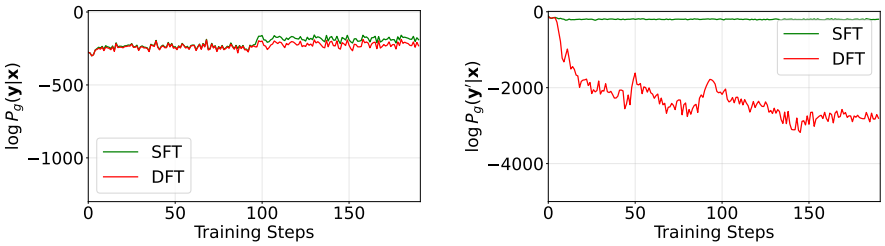
$$\begin{aligned}\min_{\theta} & -\frac{1}{n}\sum_{i=1}^n s(\mathbf{w};\mathbf{y}_i,\mathbf{x}_i) \\ & + \frac{1}{n}\sum_{i=1}^n \tau \log\left(\frac{1}{m}\sum\nolimits_{\mathbf{y}'\in\mathcal{S}_i} \exp(s(\mathbf{w};\mathbf{y}',\mathbf{x}_i)/\tau)\right).\end{aligned} \tag{6.71}$$

DFT for fine-tuning LLMs

Let us apply the DPM approach to fine-tuning LLMs, which is referred to as discriminative fine-tuning (DFT).

Discriminative Likelihood

Unlike SFT that maximizes the generative likelihood of tokens, DFT will maximize the discriminative likelihood of data as defined in (6.67). By maximizing the dis-



(a) Log-likelihoods of positives

(b) Log-likelihoods of negatives

Fig. 6.26: (a) Log-likelihoods of (annotated) positive examples during training for different methods. (b) Log-likelihoods of "negative" examples (generated from the base model) during training for different methods. For more details, please refer to (Guo et al., 2025).

Algorithm 36 The DFT Algorithm

Initialize $\mathbf{w}_1$ as the base LLM, and $\mathbf{u}_0 = \mathbf{1}$
for $t = 1, \ldots, T-1$ **do**
  Sample a mini-batch $\mathcal{B}_t \subset \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
  for each $\mathbf{x}_i \in \mathcal{B}_t$ **do**
    Sample a mini-batch $\mathcal{B}_{i,t}^-$ from $\pi_{\text{ref}}(\cdot|\bar{\mathbf{x}}_i)$ via an offline pool
    Update $u_{i,t+1}$ according to

$$u_{i,t} = (1-\gamma)u_{i,t-1} + \gamma \frac{1}{B} \sum_{\mathbf{y}' \in \mathcal{B}_{i,t}^0} \frac{\exp(\frac{s(\mathbf{w}_t;\mathbf{y}',\mathbf{x}_i)}{\tau})}{\pi_{\text{ref}}(\mathbf{y}'|\bar{\mathbf{x}}_i)}, \tag{6.72}$$

  end for
  Compute a vanilla gradient estimator $\mathbf{z}_t$ according to

$$\mathbf{z}_t = -\frac{1}{|\mathcal{B}_t|} \sum_{\mathbf{x}_i \in \mathcal{B}_t} \nabla s(\mathbf{w}_t; \mathbf{y}_i, \mathbf{x}_i) + \frac{1}{|\mathcal{B}_t|} \sum_{\mathbf{x}_i \in \mathcal{B}_t} \frac{1}{u_{i,t+1}|\mathcal{B}_{i,t}^-|} \sum_{\mathbf{y}' \in \mathcal{B}_{i,t}^-} \frac{\exp(\frac{s(\mathbf{w}_t;\mathbf{y}',\mathbf{x}_i)}{\tau}) \nabla s(\mathbf{w}_t; \mathbf{y}', \mathbf{x}_i)}{\pi_{\text{ref}}(\mathbf{y}'|\bar{\mathbf{x}}_i)}. \tag{6.73}$$

  Update $\mathbf{w}_{t+1}$ using Momentum or AdamW
end for

criminative log-likelihood of the training data, we not only increase the score of the true output $\mathbf{y}_i$ for each input $\mathbf{x}_i$, corresponding to the numerator of the discriminative likelihood, but also decrease the scores of other potentially bad answers in $\mathcal{Y}$, which correspond to the denominator of the discriminative likelihood; see Figure 6.26.

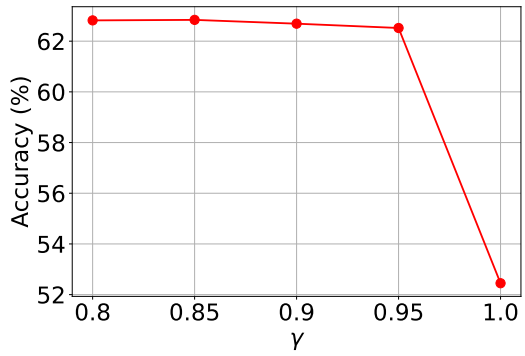


Fig. 6.27: Using moving average estimators with $\gamma < 1$ is important for improving the performance. For more details, please refer to (Guo et al., 2025).

The Scoring Function

For fine-tuning LLMs, the scoring function can be defined based on the generative log-likelihood $\log \pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})$, as it measures the likeliness of generating $\mathbf{y}$ given $\mathbf{x}$ by the model $\pi_{\mathbf{w}}$. For a good model, we expect that a high value of the generative log-likelihood $\log \pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})$ would indicate a high fitness score of $\mathbf{y}$ to answer $\mathbf{x}$. With such correspondence, the above discriminative learning framework would increase the chance of generating a good output $\mathbf{y}$ given $\mathbf{x}$ and decrease the chance of generating possibly bad outputs given $\mathbf{x}$. Common choices for the scoring function include the raw log-likelihood $s(\mathbf{w}; \mathbf{y}, \mathbf{x}) = \log \pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})$ and a length-normalized version $s(\mathbf{w}; \mathbf{y}, \mathbf{x}) = \frac{1}{|\mathbf{y}|} \log \pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})$. Using the unnormalized version $s_{\mathbf{w}}(\mathbf{y}, \mathbf{x}) = \log \pi_{\mathbf{w}}(\mathbf{y}|\mathbf{x})$ leads to the following DFT objective:

$$\begin{aligned} &\min_{\mathbf{w}} -\frac{1}{n} \sum_{i=1}^{n} \log \pi_{\mathbf{w}}(\mathbf{y}_i|\mathbf{x}_i) \\ &+ \tau \frac{1}{n} \sum_{i=1}^{n} \log \left(\sum\nolimits_{\mathbf{y}' \in \mathcal{Y}} \exp \left(\frac{\log \pi_{\mathbf{w}}(\mathbf{y}'|\mathbf{x}_i)}{\tau} \right) \right). \end{aligned} \tag{6.74}$$

Comparing the DFT objective of to that of SFT in (6.53), we observe that the first term in (6.74) is identical to the objective of SFT. The key difference lies in the second term, which penalizes the possibly poor outputs in $\mathcal{Y}$ for each $\mathbf{x}_i$ by reducing their generative log-likelihood, thereby discouraging their generation.

Sampling Distribution

The optimization analysis reveals that the variance bound σ_0 of the mini-batch estimator for the inner function $g(\mathbf{w}; \mathbf{x}_i, \mathcal{Y})$ significantly impacts convergence speed (cf. Theorem 5.1). Ideally, the variance-minimizing distribution is $P_{\mathbf{w}}(\cdot|\mathbf{x}_i)$. How-

ever, this distribution is impractical to evaluate and difficult to sample from directly. Moreover, we aim for the sampled outputs $\mathbf{y}' \sim P_i(\cdot)$ to represent likely poor responses to $\mathbf{x}_i$. A practical approach is to define $P_i(\cdot) = \pi_{\text{ref}}(\cdot|\bar{\mathbf{x}}_i)$, where π_{ref} denotes the base LLM to be fine-tuned and $\bar{\mathbf{x}}_i$ is an augmented version of $\mathbf{x}_i$ with added system prompts to encourage the generation of suboptimal outputs. This relies on the assumption that the base model is unlikely to generate high-quality answers in this context.

The Optimization Algorithm

An application of the SOX algorithm for solving (6.69) is presented in Algorithm 36. The sequence $\{u\}$ plays a critical role in effectively penalizing the sampled "negative data," as illustrated in Figure 6.27. A PyTorch implementation of DFT is at

https://github.com/Optimization-AI/DFT.

6.6.3 DisCO for Reinforcing Large Reasoning Models

DisCO, short for *Discriminative Constrained Optimization*, is a recent approach for reinforcing large reasoning models. It is motivated by the connection between the GRPO objective and discriminative learning objectives, and is designed to overcome key limitations of GRPO and its variants.

Limitation of GRPO and Connection with Discriminative Learning

Let $r(\mathbf{y}|\mathbf{x}) \in \{1, 0\}$ denote the reward assigned to an output $\mathbf{y}$ with respect to the input $\mathbf{x}$. A quantity that is important to the analysis is $p(\mathbf{x}) = \mathbb{E}_{\mathbf{y}\sim\pi_{\text{old}}(\cdot|\mathbf{x})}[r(\mathbf{y}|\mathbf{x})] \in [0, 1]$, which quantifies the difficulty of the question $\mathbf{x}$ under the model π_{old}. We denote by $\pi^+_{\text{old}}(\cdot|\mathbf{x})$ the conditional distribution of outputs when the reward is one (i.e., positive answers) and by $\pi^-_{\text{old}}(\cdot|\mathbf{x})$ the conditional distribution of outputs when the reward is zero (i.e., negative answers).

In the following analysis we assume $p(\mathbf{x}) = \mathbb{E}_{\mathbf{y}\sim\pi_{\text{old}}(\cdot|\mathbf{x})} r(\mathbf{y}|\mathbf{x}) \in (0, 1)$; otherwise we can remove them from consideration as done in practice.

Proposition 6.1. *Let us consider the objective of GRPO and its variants with the following form:*

$$\mathcal{J}_0(\mathbf{w}) = \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}\sim\pi_{old}(\cdot|\mathbf{x})}\left[\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x}, y_{<t})}{\pi_{old}(y_t|\mathbf{x}, y_{<t})}, A(\mathbf{x}, \mathbf{y})\right)\right], \tag{6.75}$$

where $A(\mathbf{x}, \mathbf{y})$ is given in (6.66). Assume that $f(x, y)$ is non-decreasing function of x such that $f(x, y) = \mathbb{I}(y > 0)yf^+(x, 1) - \mathbb{I}(y \le 0)yf^-(x, 1)$, where both f^+, f^- are non-decreasing functions of x, then we have

$$\mathcal{J}_0(\mathbf{w}) = \mathbb{E}_{\mathbf{x}}\sqrt{p(\mathbf{x})(1-p(\mathbf{x}))}\mathbb{E}_{\mathbf{y}\sim\pi^+_{old}(\cdot|\mathbf{x}),\mathbf{y}'\sim\pi^-_{old}(\cdot|\mathbf{x})}[s^+(\mathbf{w};\mathbf{y},\mathbf{x}) - s^-(\mathbf{w};\mathbf{y}',\mathbf{x})], \tag{6.76}$$

where

$$s^+(\mathbf{w};\mathbf{y},\mathbf{x}) = \frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f^+\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{old}(y_t|\mathbf{x},y_{<t})},1\right)$$
$$s^-(\mathbf{w};\mathbf{y},\mathbf{x}) = \frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f^-\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{old}(y_t|\mathbf{x},y_{<t})},1\right).$$

In particular, for GRPO we have

$$s^+(\mathbf{w};\mathbf{y},\mathbf{x}) = \frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} \min(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{old}(y_t|\mathbf{x},y_{<t})},1+\epsilon), \tag{6.77}$$

$$s^-(\mathbf{w};\mathbf{y},\mathbf{x}) = \frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} \max(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{old}(y_t|\mathbf{x},y_{<t})},1-\epsilon). \tag{6.78}$$

Proof. Since $\mathbb{E}_{\mathbf{y}\sim\pi_{\text{old}}(\cdot|\mathbf{x})}r(\mathbf{y}|\mathbf{x}) = p(\mathbf{x}), \text{Var}_{\mathbf{y}\sim\pi_{\text{old}}(\cdot|\mathbf{x})}r(\mathbf{y}|\mathbf{x}) = p(\mathbf{x})(1-p(\mathbf{x}))$, we have

$$A(\mathbf{x},\mathbf{y}) = \begin{cases} \sqrt{\frac{1-p(\mathbf{x})}{p(\mathbf{x})}}, & \text{if } r(\mathbf{y}|\mathbf{x}) = 1, \\ -\sqrt{\frac{p(\mathbf{x})}{1-p(\mathbf{x})}}, & \text{if } r(\mathbf{y}|\mathbf{x}) = 0. \end{cases} \tag{6.79}$$

By the law of total expectation, we have

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}\sim\pi_{\text{old}}(\cdot|\mathbf{x})}\left[\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, A(\mathbf{x},\mathbf{y})\right)\right] \\
&= \mathbb{E}_{\mathbf{x}}\Bigg[p(\mathbf{x})\mathbb{E}_{\mathbf{y}\sim\pi^{+}_{\text{old}}(\cdot|\mathbf{x})}\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, A(\mathbf{x},\mathbf{y})\right) \\
&\qquad + (1-p(\mathbf{x}))\mathbb{E}_{\mathbf{y}\sim\pi^{-}_{\text{old}}(\cdot|\mathbf{x})}\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, A(\mathbf{x},\mathbf{y})\right)\Bigg] \\
&= \mathbb{E}_{\mathbf{x}}\Bigg[p(\mathbf{x})\mathbb{E}_{\mathbf{y}\sim\pi^{+}_{\text{old}}(\cdot|\mathbf{x})}\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, \sqrt{\frac{1-p(\mathbf{x})}{p(\mathbf{x})}}\right) \\
&\qquad + (1-p(\mathbf{x}))\mathbb{E}_{\mathbf{y}\sim\pi^{-}_{\text{old}}(\cdot|\mathbf{x})}\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, -\sqrt{\frac{p(\mathbf{x})}{1-p(\mathbf{x})}}\right)\Bigg] \\
&= \mathbb{E}_{\mathbf{x}}\sqrt{p(\mathbf{x})(1-p(\mathbf{x}))}\Bigg[\mathbb{E}_{\mathbf{y}\sim\pi^{+}_{\text{old}}(\cdot|\mathbf{x})}\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f^{+}\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, 1\right) \\
&\qquad - \mathbb{E}_{\mathbf{y}\sim\pi^{-}_{\text{old}}(\cdot|\mathbf{x})}\frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|} f^{-}\left(\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x},y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x},y_{<t})}, 1\right)\Bigg],
\end{aligned}
\tag{6.80}
$$

where the last equality follows from the assumption about $f(x, y)$. For GPRO, we have $f^{+}(x, 1) = \min(x, \text{clip}(x, 1-\epsilon, 1+\epsilon)) = \min(x, 1+\epsilon)$ and $f^{-}(x, 1) = \max(x, \text{clip}(x, 1-\epsilon, 1+\epsilon)) = \max(x, 1-\epsilon)$. □

Why it matters

We derive two insights from Proposition 6.1 regarding the two components of $\mathcal{J}_0$. First, let us consider the component $\mathbb{E}_{\mathbf{y}\sim\pi^{+}_{\text{old}}(\cdot|\mathbf{x}),\mathbf{y}'\sim\pi^{-}_{\text{old}}(\cdot|\mathbf{x}),}[s^{+}(\mathbf{w};\mathbf{y},\mathbf{x}) - s^{-}(\mathbf{w};\mathbf{y}',\mathbf{x})]$. Since both f^{+} and f^{-} are non-decreasing functions of the first argument, then both $s^{+}(\mathbf{w};\mathbf{y},\mathbf{x})$ and $s^{-}(\mathbf{w};\mathbf{y},\mathbf{x})$ are non-decreasing functions of $\pi_\theta(y_t|\mathbf{x},y_{<t})$. Hence, maximizing $\mathcal{J}_0$ would increase the likelihood of tokens in the positive answers and decrease the likelihood of tokens in the negative answers. This makes sense as we would like the new model to have a high likelihood of generating a positive (correct) answer and a low likelihood of generating a negative (incorrect) answer. This mechanism is closely related to traditional discriminative methods of supervised learning in the context of AUC maximization, which aims to maximize the scores of positive samples $\mathbf{y}\sim\pi^{+}_{\text{old}}(\cdot|\mathbf{x})$ while minimizing scores of negative samples $\mathbf{y}'\sim\pi^{-}_{\text{old}}(\cdot|\mathbf{x})$, where the $\mathbf{x}$ acts like the classification task in the AUC maximization. Hence, in the context of discriminative learning, we refer to $s^{+}(\mathbf{y},\mathbf{x})$ and $s^{-}(\mathbf{y},\mathbf{x})$ as scoring functions. Therefore, $\mathbb{E}_{\mathbf{y}\sim\pi^{+}_{\text{old}}(\cdot|\mathbf{x}),\mathbf{y}'\sim\pi^{-}_{\text{old}}(\cdot|\mathbf{x}),}[s^{+}(\mathbf{y},\mathbf{x}) - s^{-}(\mathbf{y}',\mathbf{x})]$ is a discriminative objective.

Second, let us consider the component $\omega(\mathbf{x}) = \sqrt{p(\mathbf{x})(1-p(\mathbf{x}))}$, which acts like a weight scaling the discriminative objective for each individual input question.

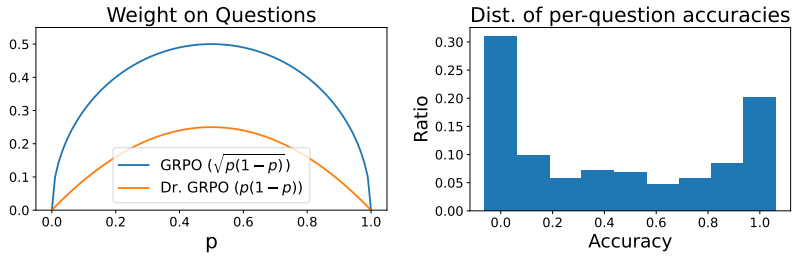


Fig. 6.28: (a) Weight on questions based on correctness probability p; (b) Histogram of per-question accuracy evaluated in the GRPO learning.

It is this component that leads to difficulty bias. As shown in Figure 6.28(a), questions with very high $p(\mathbf{x})$ values (close to 1) or very low $p(\mathbf{x})$ values (close to 0) receive small weights for their discriminative objectives, causing the optimization to focus primarily on questions of intermediate difficulty while paying little attention to hard questions ($p(\mathbf{x}) \approx 0$) and easy questions ($p(\mathbf{x}) \approx 1$). This mechanism may significantly hinder the learning efficiency. Intuitively, if the generated answers have only one correct solution out of 10 trials, i.e. $p(\mathbf{x}) = 0.1$, we should grasp this chance to enhance the model instead of overlooking it. On the other hand, even when we encounter an easy question with a probability of $p(\mathbf{x}) = 0.9$, we should keep improving the model rather than being satisfied because it still makes mistakes with respect to this question.

DisCO: A Discriminative Constrained Optimization Framework

Motivated by the analysis of GRPO and its connection with discriminative learning, discriminative objectives can be borrowed directly for learning the reasoning model. Below, we introduce two approaches.

Discriminative Objectives

For a given question $\mathbf{x}$, let $s(\mathbf{w}; \mathbf{y}, \mathbf{x})$ denote a scoring function that measures how likely the model $\pi_{\mathbf{w}}$ "predicts" the output $\mathbf{y}$ for a given input $\mathbf{x}$ [1]. Then the AUC score for the "task" $\mathbf{x}$ is equivalent to $\mathbb{E}_{\mathbf{y}\sim\pi^+_{\text{old}},\mathbf{y}'\sim\pi^-_{\text{old}}}[\mathbb{I}(s(\mathbf{w};\mathbf{y},\mathbf{x}) > s(\mathbf{w};\mathbf{y}'\mathbf{x}))]$. Using a non-decreasing continuous surrogate function ℓ, we form the following objective (in expectation form) for minimization:

$$\mathcal{L}_1(\mathbf{w}) := \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}\sim\pi^+_{\text{old}}(\cdot|\mathbf{x}),\mathbf{y}'\sim\pi^-_{\text{old}}(\cdot|\mathbf{x})}\ell(s(\mathbf{w};\mathbf{y}'\mathbf{x}) - s(\mathbf{w};\mathbf{y},\mathbf{x})). \tag{6.81}$$

[1] in the context of generative models, "predicts" is like "generates".

One difference from the objective of GRPO is that we use a single scoring function $s(\mathbf{w}; \mathbf{y}, \mathbf{x})$ for both positive outputs $\mathbf{y}$ and negative outputs $\mathbf{y}'$. The different scoring functions for positive and negative outputs in GRPO actually arise from the clipping operations. The clipping could cause the vanishing gradient, which may also slow down the learning process. To avoid these issues, we consider non-clipping scoring functions.

One advantage of designing the objective based on the principle of discriminative learning is the ability to leverage a wide range of advanced objectives to improve training. A key challenge in RL fine-tuning for reasoning models is the sparse rewards, which leads to imbalance in generated outputs. Specifically, for some questions where $p(\mathbf{x}) \ll 1$, the number of negative outputs can significantly exceed the number of positive ones. The objective function $\mathcal{L}_1$ is motivated by maximizing AUC for each question $\mathbf{x}$, i.e., $\mathbb{E}_{\mathbf{y}\sim\pi^+_{\text{old}},\mathbf{y}'\sim\pi^-_{\text{old}}}[\mathbb{I}(s(\mathbf{w};\mathbf{y},\mathbf{x}) > s(\mathbf{w};\mathbf{y}'\mathbf{x}))]$. However, when there is much more negative data than positive data, AUC is not a good measure. For example, let us consider a scenario that there are 1 positive $\mathbf{y}_+$ and 100 negatives $\{\mathbf{y}^1_-, \ldots, \mathbf{y}^{100}_-\}$. If the scores of these data are $s(\mathbf{y}^1_-, \mathbf{x}) = 0.9, s(\mathbf{y}_+, \mathbf{x}) = 0.5, s(\mathbf{y}^2_-, \mathbf{x}) = s(\mathbf{y}^3_-, \mathbf{x}) \ldots = s(\mathbf{y}^{100}_-, \mathbf{x}) = 0.001$, then the AUC score is $\frac{99}{100} = 0.99$. The AUC score is high but is not informative as the model still generates the negative data $\mathbf{y}^1_-$ more likely than the positive data $\mathbf{y}_+$.

To address this issue, we leverage the pAUC objective (6.28), leading to the following objective for minimization:

$$\mathcal{L}_2(\mathbf{w}) := \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}\sim\pi^+_{\text{old}}(\cdot|\mathbf{x})}\tau \log\left(\mathbb{E}_{\mathbf{y}'\sim\pi^-_{\text{old}}(\cdot|\mathbf{x})}\exp\left(\frac{\ell(s(\mathbf{w};\mathbf{y}',\mathbf{x}) - s(\mathbf{w};\mathbf{y},\mathbf{x}))}{\tau}\right)\right). \tag{6.82}$$

Lemma 2.4 indicates that $\mathcal{L}_2(\mathbf{w}) \geq \mathcal{L}_1(\mathbf{w})$ by Jensen's inequality for the concave function log. Hence, minimizing $\mathcal{L}_2(\mathbf{w})$ will automatically decreasing $\mathcal{L}_1(\mathbf{w})$. However, the reverse is not true. This also explains why minimizing $\mathcal{L}_2(\mathbf{w})$ could be more effective than maximizing $\mathcal{L}_1(\mathbf{w})$.

Scoring functions

Different scoring functions can be considered. Two examples are given below.

- The log-likelihood (log-L) scoring function is defined by

$$s(\mathbf{w}; \mathbf{y}, \mathbf{x}) = \frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|}\log \pi_{\mathbf{w}}(y_t|\mathbf{x}, y_{<t}).$$

- The likelihood ratio (L-ratio) scoring function is computed by

$$s(\mathbf{w}; \mathbf{y}, \mathbf{x}) = \frac{1}{|\mathbf{y}|}\sum_{t=1}^{|\mathbf{y}|}\frac{\pi_{\mathbf{w}}(y_t|\mathbf{x}, y_{<t})}{\pi_{\text{old}}(y_t|\mathbf{x}, y_{<t})}.$$

Stabilize the training with Constrained Optimization

Training instability is a long-standing issue in RL. Instead of using the clipping operation of PPO, an effective approach is to use the idea of trust region constraint of TRPO, which restricts the updated model $\mathbf{w}$ in the trust region using the reverse KL:

$$\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) \leq \delta.$$

Putting It All Together

DisCO formulates policy learning as a discriminative constrained optimization problem that combines discriminative objectives with a trust-region constraint. Specifically, it solves one of the following two formulations:

$$\begin{aligned} &\min_{\mathbf{w}} \mathcal{L}_1(\mathbf{w}) \\ &\text{s.t.} \quad \mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) \leq \delta, \end{aligned} \tag{6.83}$$

or alternatively,

$$\begin{aligned} &\min_{\mathbf{w}} \mathcal{L}_2(\mathbf{w}) \\ &\text{s.t.} \quad \mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) \leq \delta. \end{aligned} \tag{6.84}$$

Optimization Algorithm

To tackle the constrained optimization, we can use the penalty method presented in next section, which converts the constrained problem into an unconstrained one with an appropriate penalty parameter β. For example, with a squared hinge penalty function, we solve

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \beta[\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) - \delta]_+^2, \tag{6.85}$$

where $[\cdot]_+ = \max\{\cdot, 0\}$. We will show that under an appropriate assumption regarding the constraint function and β, solving the above squared-hinge penalized objective (6.85) can return a KKT solution of the original constrained problem (6.83).

We discuss the difference between using the squared-hinge penalty function and the regular KL divergence regularization $\beta\mathrm{KL}(\pi_{\text{old}}, \pi_\theta)$. The squared-hinge penalty function has a dynamic weighting impact for the gradient, $\nabla\beta[\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) - \delta]_+^2 = 2\beta[\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) - \delta]_+\nabla\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}})$, such that if the constraint is satisfied then the weight $2\beta[\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}}) - \delta]_+$ before the gradient of the regularization term $\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}})$ becomes zero. This means the KL divergence is only effective when the constraint is violated. In contrast, the regular KL divergence regularization $\beta\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}})$ always contributes a gradient $\beta\nabla\mathrm{KL}(\pi_{\text{old}}, \pi_{\mathbf{w}})$ no matter whether the constraint is satisfied or not, which could harm the learning.

The effectiveness of DisCO over GRPO and other methods has been demonstrated in (Li et al., 2025) for fine-tuning distilled Qwen and LLaMA models on a mathe-

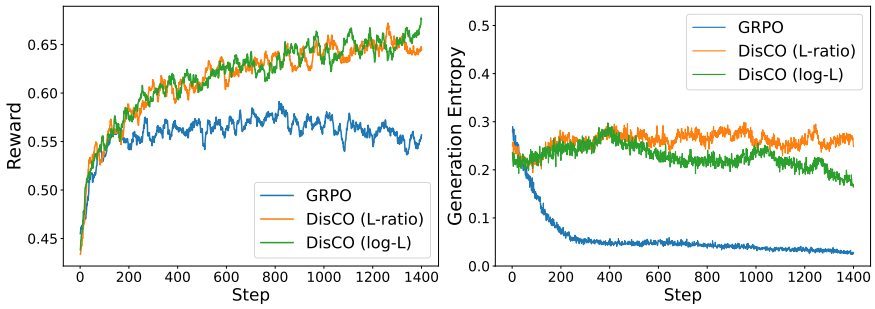


Fig. 6.29: Comparison of DisCO and GRPO for finetuning a 1.5B distilled Qwen model: left plots the training reward (averaged over generated outputs for questions used in each step) vs the number of training steps; right plots the generation entropy vs training steps. Each training step uses 128 questions sampled from the dataset, each associated with 8 generated responses to define the objective, and a mini-batch size of 32 is used for updates for a epoch. For more details, please refer to (Li et al., 2025).

matical reasoning data with approximately 40.3k unique problem-answer pairs. A comparison of the training dynamics for different methods is shown in Figure 6.29.

A PyTorch implementation of DisCO is included in the following Github repository:

https://github.com/Optimization-AI/DisCO.

6.7 Constrained Learning

Constrained learning is a machine learning framework in which the model is trained not only to minimize a specified risk but also to satisfy additional constraints. These constraints can encode domain knowledge, prior information, regularization terms, or other application-specific requirements. Unlike simple domain constraints $\mathbf{w} \in \mathcal{W}$, we consider complicated functional constraints in the form:

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^d} \quad & F(\mathbf{w}) \\ s.t. \quad & g_i(\mathbf{w}) \leq 0,\ i = 1, \ldots, m. \end{aligned} \tag{6.86}$$

In many cases, $g_i(\mathbf{w})$ also depends on the data, making its evaluation and gradient computation expensive.

Traditional works for constrained optimization include three primary categories: (1) primal methods, e.g., cooperative subgradient methods and level-set methods; (2) primal-dual methods that reformulate constrained optimization problems as saddle point problems; (3) penalty-based approaches that incorporate constraints by adding a penalty term to the objective function. In this section, we demonstrate how FCCO enables penalty-based approaches to be both efficient and practically effective.

6.7.1 A General Penalty-based Approach via FCCO

To tackle the constraints, a penalty-based approach uses a penalty function $f(\cdot)$ to convert the constrained problem into an unconstrained one:

$$\min_{\mathbf{w}} F(\mathbf{w}) + \frac{\rho}{m} \sum_{i=1}^{m} f(g_i(\mathbf{w})), \tag{6.87}$$

where $\rho > 0$ is called the *penalty parameter*. Commonly used penalty functions include:

- Squared hinge penalty:
$$f(g) = \frac{1}{2}[g]_+^2,$$
- Hinge penalty:
$$f(g) = [g]_+,$$
- Smoothed hinge penalty:
$$f(g) = \begin{cases} g - \frac{\epsilon}{2} & \text{if } g \geq \epsilon, \\ \frac{g^2}{2\epsilon} & \text{if } 0 < g < \epsilon, \\ 0 & \text{otherwise,} \end{cases}$$
where $\epsilon \ll 1$ is a small constant.

Different penalty functions yield different convergence rates. However, they share a common property: when the constraints are satisfied at a point $\mathbf{w}$, no penalty is incurred; otherwise, the greater the violation, the larger the penalty.

We can see that the added second term in (6.87) is a form of FCCO. Hence, the algorithms developed in Chapter 5 can be applied to solving the resulting unconstrained problem. Nevertheless, we need to answer several important questions: (1) What is an appropriate value for ρ? (2) What convergence guarantees can be established for the original constrained problem?

Equivalent min-max formulation

By using the conjugate of f, the unconstrained problem is equivalent to:

$$\min_{\mathbf{w}} \max_{\mathbf{y} \in \mathrm{dom}^m(f^*)} F(\mathbf{w}) + \rho \frac{1}{m} \sum_{i=1}^{m} \left(y_i g_i(\mathbf{w}) - f^*(y_i)\right), \tag{6.88}$$

For the three penalty functions, we have

- Squared hinge penalty: $f^*(y) = \frac{1}{2}y^2$, $\mathrm{dom}(f^*) = \{y : y \geq 0\}$;
- Hinge penalty: $f^*(y) = \mathbb{I}_{0,\infty}[y \in \mathrm{dom}(f^*)]$, $\mathrm{dom}(f^*) = \{y : y \in [0, 1]\}$;

- Smoothed hinge penalty: $f^*(y) = \frac{\epsilon}{2}y^2$, $\text{dom}(f^*) = \{y : y \in [0, 1]\}$;

KKT solutions

Let us focus on non-convex optimization problems with a non-convex objective $F(\mathbf{w})$ and non-convex constraints $g_k(\mathbf{w}), \forall k$. For a non-convex optimization problem, finding a globally optimal solution is intractable. Instead, a Karush-Kuhn-Tucker (KKT) solution is of interest, which is an extension of a stationary solution of an unconstrained non-convex optimization problem.

Definition 6.1 (KKT solution) A solution $\mathbf{w}$ is a KKT solution to (6.86) if there exists $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^\top \in \mathbb{R}^m_+$ such that (i) $0 \in \partial F(\mathbf{w}) + \sum_{k=1}^m \lambda_k \partial g(\mathbf{w})$, (ii) $g_k(\mathbf{w}) \leq 0, \forall k$ and (iii) $\lambda_k g_k(\mathbf{w}) = 0, \forall k$.

For non-asymptotic analysis, we consider finding an ϵ-KKT solution as defined below.

Definition 6.2 A solution $\mathbf{w}$ is an ϵ-KKT solution to (6.86) if there exists $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^\top \in \mathbb{R}^m_+$ such that (i): $\text{dist}(0, \partial F(\mathbf{w}) + \sum_{k=1}^m \lambda_k \partial g_k(\mathbf{w})) \leq \epsilon$, (ii): $[g_k(\mathbf{w})]_+ \leq \epsilon, \forall k$, and (iii): $|\lambda_k g_k(\mathbf{w})| \leq \epsilon, \forall k$.

If the objective and the constraint functions are non-smooth, finding an ϵ-KKT solution is not tractable, even the constraint functions are absent. For example, if $F(x) = |x|$ finding ϵ-stationary solution is infeasible unless we find the optimal solution $x = 0$. To address this challenge, we consider finding a nearly ϵ-KKT solution defined below.

Definition 6.3 A solution $\mathbf{w}$ is a nearly ϵ-KKT solution to (6.86) if there exist $\bar{\mathbf{w}}$ and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^\top \in \mathbb{R}^m_+$ such that (i): $\|\mathbf{w} - \bar{\mathbf{w}}\|_2 \leq O(\epsilon)$, $\text{dist}(0, \partial F(\bar{\mathbf{w}}) + \sum_{k=1}^m \lambda_k \partial g_k(\bar{\mathbf{w}})) \leq \epsilon$, (ii): $[g_k(\bar{\mathbf{w}})]_+ \leq \epsilon, \forall k$, and (iii): $|\lambda_k g_k(\bar{\mathbf{w}})| \leq \epsilon, \forall k$.

Theory

Solving the unconstrained problem (6.87) can yield a (nearly) stationary solution. But is this solution close to satisfying the KKT conditions of the original constrained problem? We answer this question for the three penalty functions below. Let $\boldsymbol{g}(\mathbf{w}) = (g_1(\mathbf{w}), \ldots, g_m(\mathbf{w}))^\top \in \mathbb{R}^m$ denote the vector of constraint functions, and let $\nabla \boldsymbol{g}(\mathbf{w}) \in \mathbb{R}^{m \times d}$ denote its Jacobian matrix.

Squared Hinge Penalty

Let us assume F and g_k are differentiable. We make the following assumption regarding the regularity of the constraint functions.

Assumption 6.1. *There exists a constant $\delta > 0$ such that $\sigma_{\min}(\nabla \boldsymbol{g}(\mathbf{w})) \geq \delta$ for any $\mathbf{w}$ satisfying $\max_{k=1,\ldots,K} g_k(\mathbf{w}) > 0$, where $\sigma_{\min}(\cdot)$ denotes the minimum singular value of a matrix.*

This assumption implies that when any constraint is violated, its gradient direction can be used to effectively reduce the constraint value. To illustrate this, consider a single constraint defined by a L_g-smooth function $g(\cdot)$. Suppose $\mathbf{w}$ is a point where the constraint is violated, i.e., $g(\mathbf{w}) > 0$. Taking a gradient descent step $\mathbf{w}' = \mathbf{w} - \eta \nabla g(\mathbf{w})$ yields:

$$\begin{aligned} g(\mathbf{w}') &\leq g(\mathbf{w}) + \nabla g(\mathbf{w})^\top (\mathbf{w}' - \mathbf{w}) + \frac{L_g}{2} \|\mathbf{w}' - \mathbf{w}\|_2^2 \\ &= g(\mathbf{w}) - \left(\eta - \frac{L_g \eta^2}{2}\right) \|\nabla g(\mathbf{w})\|_2^2. \end{aligned}$$

If Assumption 6.1 holds, then $\|\nabla g(\mathbf{w})\|_2 \geq \delta$, which implies:

$$g(\mathbf{w}') \leq g(\mathbf{w}) - \left(\eta - \frac{L_g \eta^2}{2}\right) \delta^2,$$

ensuring a sufficient decrease in the constraint function value.

In addition, we need to assume the objective function is Lipschitz continuous.

Assumption 6.2. *There exists a constant $C > 0$ such that $\|\nabla F(\mathbf{w})\|_2 \leq C, \forall \mathbf{w}$.*

Under these assumptions, we establish the following theorem.

Theorem 6.1 *Suppose Assumption 6.1 and 6.2 hold. Let $\mathbf{w}$ be an ϵ-stationary solution to the unconstrained penalized problem (6.87) with a squared hinge penalty such that*

$$\mathbb{E}\left[\left\|\nabla F(\mathbf{w}) + \frac{\rho}{m} \nabla \boldsymbol{g}(\mathbf{w})^\top [\boldsymbol{g}(\mathbf{w})]_+\right\|_2^2\right] \leq \epsilon^2. \tag{6.89}$$

If $\rho \geq \max(\frac{2m(C^2+1)}{\epsilon \delta^2}, \frac{m\sqrt{2(C^2+1)}}{\epsilon \delta})$, then $\mathbf{w}$ is also an ϵ-KKT solution to the original problem (6.86).

Proof. Let $\lambda_k = \frac{\rho}{m}[g_k(\mathbf{w})]_+, \forall k$. If $\max_k g_k(\mathbf{w}) \leq 0$, then $\lambda_k = 0$. As a result, $\mathbf{w}$ is an ϵ-KKT solution to the original problem.

Below, let us focus on the case $\max_k g_k(\mathbf{w}) > 0$, i.e., there exists one constraint that is violated at $\mathbf{w}$. Then, under Assumption 6.1, we have

$$
\begin{aligned}
\|[\boldsymbol{g}(\mathbf{w})]_+\|_2^2 \leq & \frac{1}{\delta^2}\|\nabla \boldsymbol{g}(\mathbf{w})^\top [\boldsymbol{g}(\mathbf{w})]_+\|_2^2 \\
= & \frac{m^2}{\rho^2\delta^2}\left\|\nabla F(\mathbf{w}) + \frac{\rho}{m}\nabla \boldsymbol{g}(\mathbf{w})^\top [\boldsymbol{g}(\mathbf{w})]_+ - \nabla F(\mathbf{w})\right\|_2^2 \\
\leq & \frac{2m^2}{\rho^2\delta^2}\left[\|\nabla F(\mathbf{w})\|_2^2 + \left\|\nabla F(\mathbf{w}) + \frac{\rho}{m}\nabla \boldsymbol{g}(\mathbf{w})[\boldsymbol{g}(\mathbf{w})]_+\right\|_2^2\right] \\
\leq & \frac{2m^2}{\rho^2\delta^2}\left[C^2 + \epsilon^2\right] \leq \epsilon^2,
\end{aligned}
\tag{6.90}
$$

where the last inequality follows from $\rho \geq \frac{m\sqrt{2(C^2+\epsilon^2)}}{\delta\epsilon}$. Hence $[g_k(\mathbf{w})]_+ \leq \epsilon, \forall k$.

Then, let us bound $|\lambda_k g_k(\mathbf{w})|$. If $g_k(\mathbf{w}) < 0$, then $\lambda_k = 0$, we have $|\lambda_k g_k(\mathbf{w})| = 0$. If $g_k(\mathbf{w}) \geq 0$, then

$$
\begin{aligned}
\mathbb{E}|\lambda_k g_k(\mathbf{w})]| = \mathbb{E}|\frac{\rho}{m} g_k(\mathbf{w}) g_k(\mathbf{w})]| \leq & \frac{\rho}{m}\mathbb{E}\|[\boldsymbol{g}(\mathbf{w})]_+\|_2^2 \\
\leq & \frac{\rho}{m}\cdot\frac{2m^2}{\rho^2\delta^2}\left[\|\nabla F(\mathbf{w})\|_2^2 + \left\|\nabla F(\mathbf{w}) + \frac{\rho}{m}\nabla \boldsymbol{g}(\mathbf{w})[\boldsymbol{g}(\mathbf{w})]_+\right\|_2^2\right] \\
\leq & \frac{2m}{\rho\delta^2}\left[C^2 + \epsilon^2\right] \leq \epsilon
\end{aligned}
\tag{6.91}
$$

where the last inequality uses $\rho \geq \frac{2m(C^2+\epsilon^2)}{\epsilon\,\delta^2}$. □

Hinge Penalty

Since the hinge function is non-smooth, let us consider non-smooth F and g_k. We make the following assumption regarding the regularity of the constraint functions.

Assumption 6.3. *There exists a constant $\delta > 0$ such that*

$$
dist\left(0, \frac{1}{m}\sum_{k=1}^{m}\partial[g_k(\mathbf{w})]_+\right) \geq \frac{\delta}{m}, \forall \mathbf{w} \in \mathcal{V} \tag{6.92}
$$

where $\mathcal{V} = \{\mathbf{w} : \max_k g_k(\mathbf{w}) > 0\}$ and $\partial[g_k(\mathbf{w})]$ denotes the subgradient in terms of $\mathbf{w}$.

The above assumption is implied by Assumption 6.1 when g is differentiable and hence is weaker. To see this, we have

$$
\text{dist}\left(0, \frac{1}{m}\sum_{k=1}^{m}\nabla[g_k(\mathbf{w})]_+\right) = \left\|\frac{1}{m}\sum_{k=1}^{m}\nabla[g_k(\mathbf{w})]_+\right\|_2 = \|\nabla g(\mathbf{w})^\top \mathbf{a}\|_2 \geq \delta\|\mathbf{a}\|_2 \geq \frac{\delta}{m},
$$

where $\mathbf{a} = \frac{1}{m}(\xi_1, \ldots, \xi_m)$, and $\xi_k \in ([g_k(\mathbf{w})]_+)' \in [0, 1]$.

Theorem 6.2 *Suppose Assumption 6.3 and Assumption 6.2 hold. Let* $\mathbf{w}$ *be a nearly ϵ-stationary solution to the unconstrained penalized problem (6.87) with a hinge*

penalty such that there exists $\bar{\mathbf{w}}$ *satisfying* $\|\mathbf{w} - \bar{\mathbf{w}}\|_2 \leq O(\epsilon)$*, and*

$$dist\left(0, \partial F(\bar{\mathbf{w}}) + \frac{\rho}{m}\sum_{k=1}^{m} \partial [g_k(\bar{\mathbf{w}})]_+\right) \leq \epsilon.$$

If $\rho > \frac{m(C+1)}{\delta}$*, then* $\mathbf{w}$ *is a nearly* ϵ*-KKT solution to the original problem (6.86).*

Proof. By the definition of $\mathbf{w}$, there exists $\bar{\mathbf{w}}$ such that $\|\mathbf{w} - \bar{\mathbf{w}}\|_2 \leq O(\epsilon)$, and

$$\text{dist}\left(0, \partial F(\bar{\mathbf{w}}) + \frac{\rho}{m}\sum_{k=1}^{m} \partial [g_k(\bar{\mathbf{w}})]_+\right) \leq \epsilon.$$

Since $\partial [g_k(\bar{\mathbf{w}})]_+ = \xi_k \partial g_k(\bar{\mathbf{w}})$, where

$$\xi_k = \begin{cases} 1 & \text{if } g_k(\bar{\mathbf{w}}) > 0, \\ [0,1] & \text{if } g_k(\bar{\mathbf{w}}) = 0, \\ 0 & \text{if } g_k(\bar{\mathbf{w}}) < 0, \end{cases} \quad \in [g_k(\bar{\mathbf{w}})]'_+,$$

there exists $\lambda_k \in \frac{\rho \xi_k}{m} \geq 0, \forall k$ such that

$$\text{dist}\left(0, \partial F(\bar{\mathbf{w}}) + \sum_{k=1}^{m} \lambda_k \partial g_k(\bar{\mathbf{w}})\right) \leq \epsilon.$$

Thus, we prove condition (i) in Definition 6.3. Next, let us prove condition (ii). We argue that $\max_k g_k(\bar{\mathbf{w}}) \leq 0$. Suppose this does not hold, i.e., $\max_k g_k(\bar{\mathbf{w}}) > 0$, we will derive a contradiction. Since $\exists \mathbf{v} \in \partial F(\bar{\mathbf{w}})$ we have

$$\begin{aligned} \epsilon &\geq \text{dist}\left(0, \mathbf{v} + \frac{\rho}{m}\sum_{k=1}^{m} \partial [g_k(\bar{\mathbf{w}})]_+\right) \\ &\geq \text{dist}\left(0, \frac{\rho}{m}\sum_{k=1}^{m} \partial [g_k(\bar{\mathbf{w}})]_+\right) - \|\mathbf{v}\|_2 \geq \frac{\rho\delta}{m} - C, \end{aligned}$$

which is a contradiction to the assumption that $\rho > \frac{m(\epsilon+C)}{\delta}$. Thus, $\max_k g_k(\bar{\mathbf{x}}) \leq 0$. This proves condition (ii). The last condition (iii) holds because: $\lambda_k = \frac{\rho \xi_k}{m}$, which is zero if $g_k(\bar{\mathbf{w}}) < 0$. Hence, $\lambda_k g_k(\bar{\mathbf{w}}) = 0$. □

Smoothed Hinge Penalty

We make the following assumption regarding the regularity of the constraint functions.

Assumption 6.4. *There exists a constant* $\delta > 0$ *such that*

$$dist\left(0, \partial g(\mathbf{w})^\top \mathbf{v}\right) \geq \delta \|\mathbf{v}\|_2, \forall \mathbf{w} \in \mathcal{V}, \forall \mathbf{v} \in \mathbb{R}^m \tag{6.93}$$

where $\mathcal{V} = \{\mathbf{w} : \max_k g_k(\mathbf{w}) > 0\}$.

Theorem 6.3 *Suppose Assumption 6.1 and Assumption 6.2 hold. Let* $\mathbf{w}$ *be a nearly* ϵ*-stationary solution to the unconstrained penalized problem (6.87) with a smoothned hinge penalty such that there exists* $\bar{\mathbf{w}}$ *satisfying* $\|\mathbf{w} - \bar{\mathbf{w}}\|_2 \leq O(\epsilon)$*, and*

$$dist\left(0, \partial F(\bar{\mathbf{w}}) + \frac{\rho}{m}\sum_{k=1}^{m} \partial f(g_k(\bar{\mathbf{w}}))\right) \leq \epsilon.$$

If $\rho > \frac{m(C+1)}{\delta}$*, then there exists* $\boldsymbol{\lambda} \in \mathbb{R}_+^m$ *it holds (i)* $\|\mathbf{w}-\bar{\mathbf{w}}\| \leq O(\epsilon)$*,* $dist(0, \partial F(\bar{\mathbf{w}}) + \sum_{k=1}^m \lambda_k \partial g_k(\bar{\mathbf{w}})) \leq \epsilon$*, (ii)* $[g_k(\bar{\mathbf{w}})]_+ \leq \epsilon, \forall k$*, and (iii)* $\lambda_k [g_k(\bar{\mathbf{w}})]_+ \leq \rho\epsilon/m, \forall k$*.*

Proof. By the definition of $f(\cdot)$, we have

$$\nabla f(\cdot) = \frac{1}{\epsilon}\min\{[\cdot]_+, \epsilon\}.$$

According to the definition of $\mathbf{w}$, there exists $\bar{\mathbf{w}}$ such that $\|\mathbf{w} - \bar{\mathbf{w}}\|_2 \leq O(\epsilon)$ and

$$\text{dist}\left(0, \partial F(\bar{\mathbf{w}}) + \frac{\rho}{m}\sum_{k=1}^{m} \nabla f[g_k(\bar{\mathbf{w}})]\partial g_k(\bar{\mathbf{w}})\right) \leq \epsilon.$$

Let $\lambda_k = \frac{\rho}{m}\nabla f(g_i(\bar{\mathbf{w}})) = \frac{\rho}{\epsilon m}\min\{[g_k(\bar{\mathbf{w}})]_+, \epsilon\}$. Then,

$$\text{dist}\left(0, \partial F(\bar{\mathbf{w}}) + \sum_{k=1}^{m} \lambda_k \partial g_k(\bar{\mathbf{w}})\right) \leq \epsilon.$$

Suppose $\max_{i=1,\ldots,m} g_i(\bar{\mathbf{w}}) > \epsilon$. Then there exists k' such that $[g_k(\bar{\mathbf{w}})]_+ > \epsilon$. Hence

$$\lambda_{k'} = \frac{\rho}{\epsilon m}\min\{[g_{k'}(\bar{\mathbf{w}})]_+, \epsilon\} = \frac{\rho}{\epsilon m}\epsilon = \frac{\rho}{m}.$$

Hence $\|\boldsymbol{\lambda}\|_2 \geq \frac{\rho}{m}$. As a result, there exists $\mathbf{v} \in \partial F(\bar{\mathbf{w}})$ such that

$$\begin{aligned}\epsilon &\geq \text{dist}\left(0, \mathbf{v} + \sum_{k=1}^{m} \lambda_k \partial g_k(\bar{\mathbf{w}})\right) \\ &\geq \text{dist}\left(0, \sum_{k=1}^{m} \lambda_k \partial g_k(\bar{\mathbf{w}})\right) - \|\mathbf{v}\|_2 \geq \frac{\rho\delta}{m} - C, \end{aligned} \tag{6.94}$$

which contradicts with $\rho > \frac{m(C+\epsilon)}{\delta}$. Therefore, we must have

$$\max_{k=1,\ldots,m} g_k(\bar{\mathbf{w}}) \leq \epsilon. \tag{6.95}$$

Finally, let us prove $|\lambda_k g_k(\bar{\mathbf{w}})| \leq O(\epsilon)$. If $g_k(\bar{\mathbf{w}}) < 0$, we have $\lambda_k = 0$, then it holds trivially. If $0 \leq g_k(\bar{\mathbf{w}}) \leq \epsilon$, we have

Algorithm	Penalty	F	g_i	Complexity	Loop
SOX	sqH/smH	SM	SM	$O(\epsilon^{-7})$	Single
MSVR	sqH/smH	MSS	MSS	$O(\epsilon^{-5})$	Single
SONX	H	WC	WC	$O(\epsilon^{-6})$	Single
SONEX	H	SM	SM	$O(\epsilon^{-5})$	Single
ALEXR-DL	smH	WC	WC	$O(\epsilon^{-5})$	Double

Table 6.1: Summary of different algorithms for penalty-based constrained optimization. 'WC" means weakly convex, "SM" means smooth, MSS mean "mean squared smoothness, 'H' denotes the hinge penalty, 'smH' denotes the smoothed hinge penalty and 'sqH' denotes the squared hinge penalty.

$$|\lambda_k g_k(\bar{\mathbf{w}})| \leq \frac{\rho}{m}[g_k(\bar{\mathbf{w}})]_+ \leq \frac{\rho\epsilon}{m}. \tag{6.96}$$

□

Critical: One important difference among the three penalty functions lies in the required order of the penalty parameter ρ. For the squared hinge penalty, it is necessary to set $\rho = O(1/\epsilon)$, whereas for the hinge and smoothed hinge penalties, it suffices to take $\rho = O(1)$. This lead to different complexities of algorithms based on these penalty functions.

Optimization Algorithms

The SOX algorithm and the MSVR algorithm can be used to optimize the squared hinge penalty function and smoothed hinge penalty function with smooth objective function and constraints. SONX and SONEX can be used to optimize the hinge penalty based objective, where the latter is equivalent to a variant for optimizing the smoothed hinge penalty using the MSVR estimator for the inner functions and the MA gradient estimator. ALEXR-DL (the double-loop ALEXR, see Section 5.4.5) can be used to optimize the problem with a weakly convex objective and weakly convex constraint functions. The computational complexities of these algorithms for obtaining a (nearly) ϵ-KKT solution are summarized in Table 6.1. The complexity results for SONX and SONEX follow directly from their original theorems. The complexities of SOX and MSVR are obtained by substituting $L_F = O(\rho)$, $L_1 = O(\rho)$, $G_1 = O(\rho)$, and $\rho = O(1/\epsilon)$ into Theorem 5.1 and Theorem 5.2, respectively. The complexity of ALEXR-DL follows the argument in Section 5.4.5.

Finally, we note that the value of the parameter δ in Assumptions 6.1, 6.3, and 6.4 has a significant impact on the complexity. In particular, smaller values of δ lead to higher complexities.

6.7.2 Continual Learning with Zero-forgetting Constraints

Continual learning usually refers to learning a sequence of tasks one by one and accumulating knowledge like human instead of substituting knowledge. The core issue in continual learning is known as catastrophic forgetting, i.e., the learning of the later tasks may significantly degrade the performance of the model for the earlier tasks. Different approaches have been investigated to mitigate catastrophic forgetting, including regularization based approaches, memory based approaches, network expansion based approaches, and constrained optimization based approaches.

Regularization based approaches

These methods aim to preserve previously learned knowledge by penalizing changes to important model parameters. These approaches usually solve the following objective:

$$\min_{\mathbf{w}} \mathcal{L}_{\text{new}}(\mathbf{w}, \mathcal{S}_{\text{new}}) + \lambda R(\mathbf{w}, \mathbf{w}_{\text{old}}), \tag{6.97}$$

where $\mathcal{L}_{\text{new}}$ denotes the loss on the new task with a data set $\mathcal{S}_{\text{new}}$, and $R(\mathbf{w}, \mathbf{w}_{\text{old}})$ is the regularization of the new model with respect to the old model. It could regularize directly in the weight parameters or regularize through functions of the weight parameters (e.g., intermediate layers of the neural networks)

Memory based approaches

These techniques store a subset of past data or representations and replay them during training on new tasks. This allows the model to rehearse old knowledge, effectively mimicking how humans review what they've previously learned. Strategies include storing raw data, or using generative models to simulate past experiences. These replay data will be used in training as simple as a regularization approach:

$$\min_{\mathbf{w}} \mathcal{L}_{\text{new}}(\mathbf{w}, \mathcal{S}_{\text{new}}) + \lambda \mathcal{L}_{\text{old}}(\mathbf{w}, \mathcal{S}_{\text{old}}) \tag{6.98}$$

where $\mathcal{L}_{\text{old}}(\mathbf{w}, \mathcal{S}_{\text{old}})$ denotes the loss of the model old tasks using their data $\mathcal{S}_{\text{old}}$.

Network Expansion based approaches

Network expansion based methods address forgetting by dynamically growing the model's architecture as new tasks are introduced. This can involve adding new neurons, layers, or modules for each task while keeping older components fixed or partially shared. By allocating new capacity, the model can learn new tasks without overwriting old knowledge.

A Constrained Optimization Approach

A key limitation of the replay and regularization approach in (6.98) is that it does not necessarily preserve the model's performance on all previous tasks, even with a large regularization weight. Moreover, overly large weights can suppress learning on the new task. This arises because not all prior tasks are equally challenging—some may be inherently easier than others.

A straightforward remedy is to formulate a constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathcal{L}_{\text{new}}(\mathbf{w}, \mathcal{S}_{\text{new}}) \\ \text{s.t.} \quad & \mathcal{L}_k(\mathbf{w}, \mathcal{S}_k) - \mathcal{L}_k(\mathbf{w}_{\text{old}}, \mathcal{S}_k) \leq 0, \quad \forall k = 1, \ldots, m, \end{aligned} \tag{6.99}$$

where $\mathcal{S}_k$ denotes the dataset for the k-th previous task and $\mathcal{L}_k$ is its corresponding loss function. These constraints ensure that the new model does not degrade performance on any individual old task as measured on replayed data, which are referred to as the zero-forgetting constraints.

Although this constrained optimization problem was traditionally considered difficult due to the number of constraints and data dependencies, the algorithms introduced in the previous subsection make it tractable. Notably, this constrained formulation serves as a unifying framework that connects all three major approaches: regularization-based, expansion-based, and memory-based continual learning.

With a penalty function f (e.g., smoothed hinge penalty), we solve the following problem:

$$\min_{\mathbf{w}} \quad \mathcal{L}_{\text{new}}(\mathbf{w}, \mathcal{S}_{\text{new}}) + \frac{\rho}{m} \sum_{k=1}^{m} f(\mathcal{L}_k(\mathbf{w}, \mathcal{S}_k) - \mathcal{L}_k(\mathbf{w}_{\text{old}}, \mathcal{S}_k)).$$

Then the algorithms can be easily applied to solving this problem.

Connection with the Three Categories of Approaches

First, the above constrained optimization method falls under memory based approaches, as it requires access to data $\mathcal{S}_k$ from each previous task to define the zero-forgetting constraints.

Second, the penalty term introduces a regularization perspective, establishing a connection with regularization based approaches. However, it differs from standard regularization as in (6.98). The penalty function adaptively weights the gradients of each prior task. For example, consider the hinge penalty. The gradient of the penalty term is given by

$$\frac{\rho}{m} \sum_{k=1}^{m} \xi_k \nabla \mathcal{L}_k(\mathbf{w}; \mathcal{S}_k), \tag{6.100}$$

where $\xi_k = 1$ if $\mathcal{L}_k(\mathbf{w}; \mathcal{S}_k) - \mathcal{L}_k(\mathbf{w}_{\text{old}}; \mathcal{S}_k) > 0$; otherwise, $\xi_k = 0$. Using the FCCO technique, an estimator u_k is used to track the quantity $\mathcal{L}_k(\mathbf{w}; \mathcal{S}_k) -$

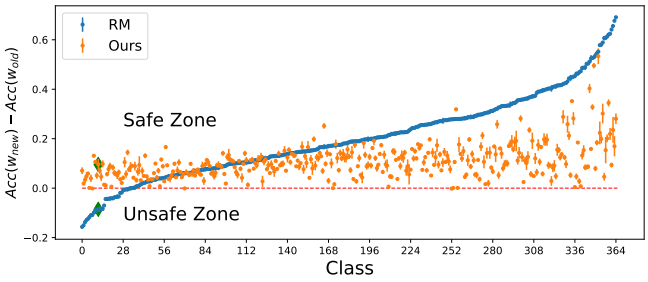


Fig. 6.30: Performance comparison with the standard regularization method (RM). The new task is to improve the performance on classifying the class *Dresssing Room* on Places365 Dataset, and other 354 classes serve as previous tasks each with 2k samples. Red line denotes the old model's performance, green diamonds denote the performance on the target class. The RM baseline shown is for the regularization parameter $\lambda = 10000$. For more details, please refer to (Li et al., 2024)

$\mathcal{L}_k(\mathbf{w}_{\text{old}}; \mathcal{S}_k)$, based on which ξ_k is computed. Consequently, the algorithm assigns adaptive weights to the gradients of prior tasks: if task k shows no performance degradation (i.e., $u_k \leq 0$), the corresponding gradient receives zero weight. This effect makes the constrained optimization approach more attractive than the regularization approach for enforcing the constraints; see Figure 6.30.

Third, although the connection to network expansion based approaches is less direct, it is suggested by the convergence analysis of the constrained optimization algorithms. Specifically, the regularity assumptions in Assumptions 6.1 and 6.3 provide insight into the benefits of network expansion. Expanding the network from the old model $\mathbf{w}_{\text{old}}$ can make it easier to find a new model that maintains or improves performance on previous tasks, effectively increasing the regularity constant δ. This, in turn, allows for a smaller penalty parameter ρ and potentially accelerates convergence—an effect formalized in what follows.

Without causing confusion, we denote by $\mathbf{w}$ the parameter of the old neural network, which consists of two components $\mathbf{w}_0$ and W such that the output $h(\mathbf{w}, \mathbf{x}) \in \mathbb{R}^{d_2}$ can be represented as $h(\mathbf{w}, \mathbf{x}) = W \cdot h_0(\mathbf{w}_0, \mathbf{x})$, where $h_0(\mathbf{w}_0, \cdot) \in \mathbb{R}^{d_1}$ is a backbone network and $W \in \mathbb{R}^{d_2 \times d_1}$ is the head. Given the old model $\mathbf{w} = (\mathbf{w}_0, W)$, we expand the network by allowing task-dependent heads, which is to let each task k have its own head $W_k = W + U_k$ where $U_k \in \mathbb{R}^{d_2 \times r}$. The output of this expanded network for task k is $h(\hat{\mathbf{w}}; \mathbf{x}) = (W + U_k) \cdot h_0(\mathbf{w}_0, \mathbf{x})$, where $\hat{\mathbf{w}} = (\mathbf{w}_0, W, U_1, \ldots, U_m)$. For simplicity, let us assume each task has only one example $\mathcal{S}_k = \{\mathbf{x}_k\}$ and let $\mathcal{L}_k(\mathbf{w}; \mathbf{S}_k) = \ell(h(\mathbf{w}, \mathbf{x}))$. Without the expansion, the Jacobian of the constraint functions at $\mathbf{w}$ is $\nabla \boldsymbol{g}(\mathbf{w}) = [\nabla h(\mathbf{w}, \mathbf{x}_1), \cdots, \nabla h(\mathbf{w}, \mathbf{x}_m)]A$, where $A \in \mathbb{R}^{m \times m}$ a diagonal matrix with $A_{kk} = \ell'(h(\mathbf{w}; \mathbf{x}_k))$. With the expansion, the Jacobian of the constraint functions at $\hat{\mathbf{w}}$ is $\nabla \hat{\boldsymbol{g}}(\mathbf{w}) = [\nabla h(\hat{\mathbf{w}}, \mathbf{x}_1), \cdots, \nabla h(\hat{\mathbf{w}}, \mathbf{x}_m)]A'$, where $A' \in \mathbb{R}^{m \times m}$ a diagonal matrix with $A'_{kk} = \ell'(h(\hat{\mathbf{w}}; \mathbf{x}_k))$. If we initialize $U_1 = U_2 \ldots = U_m = 0$, then $A = A'$. Next, we quantify the increase of the minimum singular value of

the matrix $\nabla\hat{\mathbf{h}}(\hat{\mathbf{w}}) = [\nabla h(\hat{\mathbf{w}}, \mathbf{x}_1), \cdots, \nabla h(\hat{\mathbf{w}}, \mathbf{x}_m)]$ compared with that of $\nabla\mathbf{h}(\mathbf{w}) = [\nabla h(\mathbf{w}, \mathbf{x}_1), \cdots, \nabla h(\mathbf{w}, \mathbf{x}_m)]$.

Lemma 6.4 *Suppose $U_k = \mathbf{0}$ for all k. We have*

$$\lambda_{\min}\left(\nabla\hat{\boldsymbol{h}}(\hat{\mathbf{w}})^\top \nabla\hat{\boldsymbol{h}}(\hat{\mathbf{w}})\right) \geq \lambda_{\min}\left(\nabla\boldsymbol{h}(\mathbf{w})^\top \nabla\boldsymbol{h}(\mathbf{w})\right) + \min_k \|\nabla_W h_k(\mathbf{w})\|_2^2,$$

where $\lambda_{\min}(\cdot)$ denotes the minimum eigen-value of a matrix and $h_k(\mathbf{w}) = h(\mathbf{w}; \mathbf{x}_k)$.

Why it matters

This lemma indicates that expanding the network can increase the minimum singular value of the Jacobian matrix of the constraint functions, which in turn leads to a lower complexity in finding a KKT solution, i.e., making the constraints easier to satisfy.

Proof. Let $\hat{h}_k(\hat{\mathbf{w}}) = h(\hat{\mathbf{w}}; \mathbf{x}_k)$. We consider $\mathbf{w}, W, U$ as flattend vectors. Recall that $\mathbf{w}$ has two component $\mathbf{w}_0$ and W. The gradient of $h_k(\mathbf{w})$ with respect to W and $\mathbf{w}_0$ are denoted by $\nabla_W h_k(\mathbf{w})$ and $\nabla_{\mathbf{w}_0} h_k(\mathbf{w})$, respectively. Hence,

$$\nabla h_k(\mathbf{w})^\top = \left(\nabla_{\mathbf{w}_0} h_k(\mathbf{w})^\top, \nabla_W h_k(\mathbf{w})^\top\right)$$

for $k = 1, \ldots, m$. Similarly, after adding the task-dependent heads, $\hat{\mathbf{w}}$ has three component $\mathbf{w}_0$, W, $\mathbf{U} = (U_1, \ldots, U_m)$. The gradients $\nabla_{\mathbf{w}_0}\hat{h}_k(\hat{\mathbf{w}})$, $\nabla_W \hat{h}_k(\hat{\mathbf{w}})$ $\nabla_{\mathbf{U}}\hat{h}_k(\hat{\mathbf{w}})$ are defined correspondingly, and

$$\nabla\hat{h}_k(\hat{\mathbf{w}})^\top = \left(\nabla_{\mathbf{w}_0}\hat{h}_k(\hat{\mathbf{w}})^\top, \nabla_W \hat{h}_k(\hat{\mathbf{w}})^\top, \nabla_{\mathbf{U}}\hat{h}_k(\hat{\mathbf{w}})^\top\right).$$

Recall that

$$\hat{h}_k(\hat{\mathbf{w}}) = h_k((\mathbf{w}_0, W + U_k)) \text{ for } k = 1, \ldots, m.$$

Therefore,

$$\nabla_{\mathbf{w}_0}\hat{h}_k(\hat{\mathbf{w}}) = \nabla_{\mathbf{w}_0} h_k((\mathbf{w}_0, W + U_k)),$$
$$\nabla_W \hat{h}_k(\hat{\mathbf{w}}) = \nabla_W h_k((\mathbf{w}_0, W + U_k)),$$

and

$$\nabla_{\mathbf{U}}\hat{h}_k(\hat{\mathbf{w}})^\top = \Big(\mathbf{0}, \ldots, \mathbf{0}, \underbrace{\nabla_W h_k((\mathbf{w}_0, \mathbf{W} + U_k))^\top}_{\text{The } k\text{th block}}, \mathbf{0}, \ldots, \mathbf{0}\Big),$$

where the sparsity pattern of $\nabla_{\mathbf{U}}\hat{h}_k(\hat{\mathbf{w}})$ is because $\hat{h}_k$ does not depend on $U_j, j \neq k$.

Since $U_k = \mathbf{0}$ for all k. It holds that $h_k(\mathbf{w}) = \hat{h}_k(\hat{\mathbf{w}})$ and

$$\nabla h_k(\mathbf{w})^\top = \left(\nabla_{\mathbf{w}_0} h_k(\mathbf{w})^\top, \nabla_W h_k(\mathbf{w})^\top\right) = \left(\nabla_{\mathbf{w}_0}\hat{h}_k(\hat{\mathbf{w}})^\top, \nabla_W \hat{h}_k(\hat{\mathbf{w}})^\top\right).$$

Consider any $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^m$. We have

$$
\begin{aligned}
&\lambda_{\min}\left(\left[\nabla \hat{h}_1(\hat{\mathbf{w}}), \ldots, \nabla \hat{h}_m(\hat{\mathbf{w}})\right]^\top \left[\nabla \hat{h}_1(\hat{\mathbf{w}}), \ldots, \nabla \hat{h}_m(\hat{\mathbf{w}})\right]\right) \\
=&\min_{\boldsymbol{\alpha}, \text{s.t.} \|\boldsymbol{\alpha}\|=1} \left\| \sum_{k=1}^m \alpha_k \nabla \hat{h}_k(\hat{\mathbf{w}}) \right\|_2^2 \\
=&\min_{\boldsymbol{\alpha}, \text{s.t.} \|\boldsymbol{\alpha}\|=1} \left(\left\| \sum_{k=1}^m \alpha_k \nabla_{\mathbf{w}_0} \hat{h}_k(\hat{\mathbf{w}}) \right\|_2^2 + \left\| \sum_{k=1}^m \alpha_k \nabla_W \hat{h}_k(\hat{\mathbf{w}}) \right\|_2^2 + \left\| \sum_{k=1}^m \alpha_k \nabla_{\mathbf{U}} \hat{h}_k(\hat{\mathbf{w}}) \right\|_2^2 \right) \\
=&\min_{\boldsymbol{\alpha}, \text{s.t.} \|\boldsymbol{\alpha}\|=1} \left(\left\| \sum_{k=1}^m \alpha_k \nabla h_k(\mathbf{w}) \right\|_2^2 + \sum_{k=1}^m \alpha_k^2 \left\| \nabla_W h_k(\mathbf{w}) \right\|_2^2 \right) \\
\geq&\lambda_{\min}\left(\left[\nabla h_1(\mathbf{w}), \ldots, \nabla h_m(\mathbf{w})\right]^\top \left[\nabla h_1(\mathbf{w}), \ldots, \nabla h_m(\mathbf{w})\right] \right) \\
&+ \min_k \left\| \nabla_W h_k(\mathbf{w}) \right\|_2^2,
\end{aligned}
$$

where the first two equalities are by definitions and the third equality is because $U_k = 0$ for all k. □

Practice: Squared Hinge Penalty vs. Smoothed Hinge Penalty

Both the squared hinge penalty and the smoothed hinge penalty are smooth functions, but they have different practical implications. The squared hinge penalty typically requires a much larger penalty parameter, on the order of $\rho = O(1/\epsilon)$ as indicated by the theory, to enforce the constraints effectively. In contrast, the smoothed hinge penalty achieves similar constraint satisfaction with a significantly smaller ρ. This difference is illustrated in Figure 6.31 (right), which shows that a large penalty parameter $\rho = 800$ is needed for the squared hinge penalty, whereas the smoothed hinge penalty achieves comparable results with just $\rho = 20$. As a result, optimization of the objective function tends to be more effective when using the smoothed hinge penalty as seen in Figure 6.31 (left).

6.7.3 Constrained Learning with Fairness Constraints

Machine learning models are increasingly used in high-stakes domains such as hiring, finance, and healthcare, where biased predictions can lead to unfair outcomes for individuals from protected groups (e.g., based on race, gender, or age). Learning with fairness constraints is a framework that aims to train models that are both accurate and equitable by incorporating formal definitions of fairness directly into the training objective. Various notions of fairness have been proposed, including demographic parity, equalized odds, equal opportunity, AUC fairness, ROC fairness,

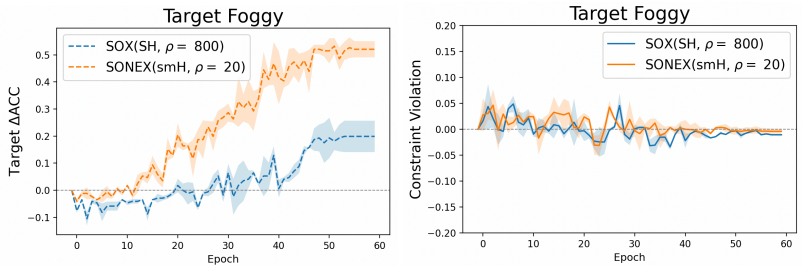


Fig. 6.31: Training curves of Target ΔACC values (left) and constraint violation (right) of different methods. The format of label is "Algorithm(penalty function, ρ)", and SH, smH mean square hinge and smoothed hinge, respectively. For more details, please refer to (Chen et al., 2025b).

and ranking fairness. Below, we present an application of constrained optimization to learning under ROC fairness constraints.

Constrained Learning with ROC Fairness

We consider a binary classification setting. Let $h(\mathbf{w};\cdot) \in \mathbb{R}$ denote a predictive model. Suppose the data are divided into two demographic groups $\mathcal{D}_p = \{(\mathbf{x}_i^p, y_i^p)\}_{i=1}^{n_p}$ and $\mathcal{D}_u = \{(\mathbf{x}_i^u, y_i^u)\}_{i=1}^{n_u}$, where $\mathbf{x}$ denotes the input data and $y \in \{1,-1\}$ denotes the class label. Traditional fairness measures usually assume the prediction is given by $\mathbb{I}(h(\mathbf{w};\mathbf{x}) > t)$ with a specific threshold. However, the threshold may be dynamically changed in practice to achieve a balance between true positive and false positive rate.

To accommodate this, a ROC fairness is introduced to ensure the ROC curves for classification of the two groups are the same, which indicates the false positive rate (FPR) and true positive rate (TPR) at all possible thresholds are equal across the two groups. Since the ROC curve is constructed with all possible thresholds, we use a set of thresholds $\Gamma = \{\tau_1, \cdots, \tau_m\}$ to define the ROC fairness. For each threshold τ, we impose a constraint that the TPR and FPR of the two groups are close, formulated as the following:

$$g_\tau^+(\mathbf{w}) = \left| \frac{1}{n_p^+} \sum_{i=1}^{n_p} \mathbb{I}(y_i^p = 1)\sigma(h(\mathbf{w};\mathbf{x}_i^p) - \tau) - \frac{1}{n_u^+} \sum_{i=1}^{n_u} \mathbb{I}(y_i^u = 1)\sigma(h(\mathbf{w};\mathbf{x}_i^u) - \tau) \right| - \kappa \le 0,$$

and

$$g_\tau^-(\mathbf{w}) = \left| \frac{1}{n_p^-} \sum_{i=1}^{n_p} \mathbb{I}(y_i^p = -1)\sigma(h(\mathbf{w};\mathbf{x}_i^p) - \tau) - \frac{1}{n_u^-} \sum_{i=1}^{n_u} \mathbb{I}(y_i^u = -1)\sigma(h(\mathbf{w};\mathbf{x}_i^u) - \tau) \right| - \kappa \le 0,$$

where $\sigma(s)$ is a surrogate of the indicator function $\mathbb{I}(s > 0)$, e.g., the sigmoid function, and $\kappa > 0$ is a tolerance parameter.

Then the learning problem can be imposed as:

$$\begin{aligned} \min_{\mathbf{w}} \quad & F(\mathbf{w}), \\ s.t. \quad & g_\tau^+(\mathbf{w}) \leq 0, g_\tau^-(\mathbf{w}) \leq 0, \forall \tau \in \Gamma. \end{aligned}$$

where $F(\mathbf{w})$ is an appropriate risk function.

By utilizing the penalty method, we solve the following problem:

$$\min_{\mathbf{w}} F(\mathbf{w}) + \frac{\rho}{2|\Gamma|} \sum_{\tau \in \Gamma} (f(g_\tau^+(\mathbf{w})) + f(g_\tau^-(\mathbf{w}))). \tag{6.101}$$

Let us define

$$\begin{aligned} g_1(\mathbf{w}; \tau) &= \frac{1}{n_p^+} \sum_{i=1}^{n_p} \mathbb{I}(y_i^p = 1) \sigma(h(\mathbf{w}; \mathbf{x}_i^p) - \tau) \\ g_2(\mathbf{w}; \tau) &= \frac{1}{n_u^+} \sum_{i=1}^{n_u} \mathbb{I}(y_i^u = 1) \sigma(h(\mathbf{w}; \mathbf{x}_i^u) - \tau). \end{aligned}$$

Since $f(\cdot)$ is a non-decreasing convex function, hence $f(|x|)$ is a convex function. Then the penalty term $f(g_\tau^+(\mathbf{w})) = f(|g_1(\mathbf{w}; \tau) - g_2(\mathbf{w}; \tau)| - \kappa)$ is a compositional of a convex function $f(\boldsymbol{g}) = f(|g_1 - g_2| - \kappa)$ and a smooth mapping $g(\mathbf{w}) = [g_1(\mathbf{w}; \tau), g_2(\mathbf{w}; \tau)]$. Hence, SONX, SONEX, ALEXR-DL can be employed to solve the above problem.

6.8 Learning Data Compositional Networks

So far, we have considered the compositional loss function, which involves comparing the output of one data $h(\mathbf{w}; \mathbf{x})$ with that of many other data. In this section, we consider compositional networks, where the computation of $h(\mathbf{w}; \mathbf{x})$ for one data $\mathbf{x}$ depends on many other data.

6.8.1 Large-scale Graph Neural Networks

Graph Neural Networks (GNNs) are a powerful class of models designed to learn representations from graph-structured data, where information is distributed across nodes and edges. Unlike traditional neural networks that operate on grid-like inputs, GNNs leverage the connectivity structure of graphs to propagate and aggregate information from a node's neighborhood, capturing both local and global patterns.

GNNs have been successfully applied to tasks such as node classification, link prediction, and graph-level classification in domains including social networks, molecular chemistry, and recommendation systems.

A key distinction in GNN-based learning lies between transductive and inductive settings. In transductive learning, the model is trained and tested on the same fixed graph, meaning all nodes (including test nodes) are present during training. Classic GNN models such as Graph Convolutional Neural (GCN) Network in this setting. In contrast, inductive methods aim to generalize to unseen nodes or entirely new graphs not available during training. GraphSAGE (Graph Sample and Aggregate) is a method that is designed for inductive learning, enabling flexible deployment in dynamic environments where new nodes or graphs continuously emerge.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each node $v \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_v$. Given a node v with neighbors $\mathcal{N}(v)$, a general scheme fo updating the node's representation in layer k is following:

$$\begin{aligned}\mathbf{h}_{\mathcal{N}(v)}^{(k)} &= \text{Aggregate}\left(\left\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right),\\ \mathbf{h}_v^{(k)} &= \text{Update}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_{\mathcal{N}(v)}^{(k)}\right),\end{aligned}$$

where the first step aggregates the representations of the nodes in the immediate neighborhood of node v into a single vector, and the second step updates the node's current representation $\mathbf{h}_v^{(k-1)}$, with the aggregated neighborhood vector to generate a new embedding $\mathbf{h}_v^{(k)}$.

GraphSAGE (Graph Sample and Aggregate)

GraphSAGE is a scalable inductive framework for learning node representations in large graphs. Let us consider a particular implementation of the above framework:

$$\mathcal{A}(\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v) \cup \{v\}\}) = \frac{1}{|\mathcal{N}_v| + 1} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \mathbf{h}_u^{(k-1)} \tag{6.102}$$

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot \mathcal{A}(\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}_v \cup \{v\}\})\right), \tag{6.103}$$

where $\mathcal{A}(\cdot)$ denotes the mean operator and $\sigma(\cdot)$ is an activation function.

When working with large-scale graphs, GraphSAGE employs node sampling to ensure scalability. At each layer, a node samples a fixed number of neighbors and aggregates their features. However, as the number of layers increases, the number of nodes involved in computing a single node's embedding can grow exponentially. Specifically, if each node samples K neighbors and the model has L layers, then computing the embedding for a single node may involve up to K^L nodes. This exponential growth is known as the *neighborhood explosion problem*, which can lead to significant computational and memory overhead, especially in deep models or

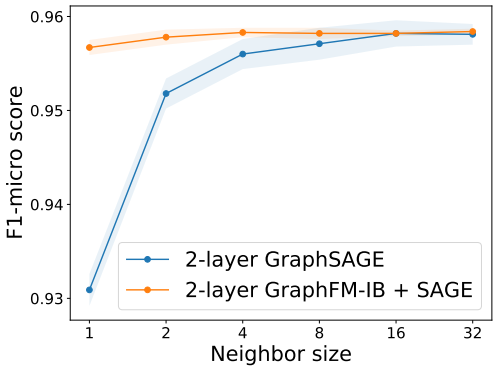


Fig. 6.32: Comparison between standard GraphSAGE and GraphSAGE with Feature Momentum on the Reddit dataset, which contains 232,965 nodes and 11,606,919 edges. Each node has an average of 49.82 neighbors. For more details, please refer to (Yu et al., 2022).

large graphs. While reducing K (e.g., to 1) can mitigate neighborhood explosion, it may also introduce high variance in the estimation of the mean operator potentially degrading model performance.

GraphSAGE with Feature Momentum

The challenge discussed earlier arises from the compositional structure of $\mathbf{h}_v^{(k)}$. To address this, we leverage a moving average estimator. Let $\mathcal{B}_v \subset \mathcal{N}(v)$ be a sub-sampled neighborhood of node v, and define $\bar{\mathcal{B}}_v = \mathcal{B}_v \cup \{v\}$. At the t-th iteration, we estimate the aggregated feature vector as follows:

$$\tilde{\mathbf{h}}_v^{(k,t)} = \begin{cases} \tilde{\mathbf{h}}_v^{(k,t-1)} & \text{if } v \notin \mathcal{D}_k, \\ (1-\gamma)\tilde{\mathbf{h}}_v^{(k,t-1)} + \gamma \hat{\mathcal{A}}\left(\left\{\hat{\mathbf{h}}_u^{(k-1,t)} : u \in \bar{\mathcal{B}}_v\right\}\right) & \text{otherwise,} \end{cases} \tag{6.104}$$

where $\mathcal{D}_k$ is the sub-sampled set of nodes updated at the k-th layer, $\gamma \in (0,1)$ is the momentum parameter, and $\hat{\mathcal{A}}(\cdot)$ is an unbiased estimator of the aggregation function $\mathcal{A}(\cdot)$ over the neighborhood $\mathcal{N}_v \cup \{v\}$. The estimator is computed as:

$$\hat{\mathcal{A}}\left(\left\{\hat{\mathbf{h}}_u^{(k-1,t)} : u \in \bar{\mathcal{B}}_v\right\}\right) = \frac{1}{|\mathcal{N}_v|+1}\hat{\mathbf{h}}_v^{(k-1,t)} + \frac{|\mathcal{N}_v|}{|\mathcal{N}_v|+1} \cdot \frac{1}{|\mathcal{B}_v|}\sum_{u \in \mathcal{B}_v} \hat{\mathbf{h}}_u^{(k-1,t)}.$$

Next, we update the feature representation at the k-th layer:

$$\hat{\mathbf{h}}_v^{(k,t)} = \sigma\left(\mathbf{W}_t^{(k)} \cdot \tilde{\mathbf{h}}_v^{(k,t)}\right). \tag{6.105}$$

This process is repeated for L layers to compute the output representation $\hat{\mathbf{h}}_v^{(L,t)}$ for sub-sampled nodes $v \in \mathcal{D}_L$, which are then used to compute the mini-batch loss. We refer to this approach as GraphSAGE with Feature Momentum.

This method effectively reduces the required number of sampled neighbors per node while maintaining the performance of using full neighborhoods; see Figure 6.32.

6.8.2 Multi-instance Learning with Attention

Multi-instance learning (MIL) refers to a setting where a bag of instances are observed for an object of interest and only one label is given to describe that object. Many real-life applications can be formulated as MIL. For example, the medical imaging data for diagnosing a patient usually consists of a series of 2D high-resolution images (e.g., CT scan), and only a single label (containing a tumor or not) is assigned to the patient.

A standard assumption for MIL is that a bag is labeled positive if at least one of its instances has a positive label, and negative if all of its instances have negative labels. The assumption implies that a MIL model must be permutation-invariant for the prediction function $h(\mathcal{X})$, where $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ denotes a bag of instances. To achieve permutation invariant property, fundamental theorems of symmetric functions have been developed. In particular, a scoring function for a set of instances $\mathcal{X}$ denoted by $h(\mathcal{X}) \in \mathbb{R}$, is a symmetric function if and only if it can be decomposed as $h(\mathcal{X}) = g(\sum_{\mathbf{x}\in\mathcal{X}} \psi(\mathbf{x}))$ (Zaheer et al., 2017), where g and ψ are suitable transformations. Another theory is that a Hausdorff continuous symmetric function $h(\mathcal{X}) \in \mathbb{R}$ can be arbitrarily approximated by a function in the form $g(\max_{\mathbf{x}\in\mathcal{X}} \psi(\mathbf{x}))$ (Qi et al., 2016), where max is the element-wise vector maximum operator and ψ and g are continuous functions. These theories provide support for several widely used pooling operators used for MIL.

Deep learning with different pooling operations

Let $e(\mathbf{w}_e; \mathbf{x}) \in \mathbb{R}^{d_o}$ be the instance-level representation encoded by a neural network $\mathbf{w}_e$, $\phi(\mathbf{w}; \mathbf{x}) \in [0, 1]$ be the instance-level prediction score (after some activation function), and $h(\mathbf{w}; \mathcal{X}_i) \in [0, 1]$ be the pooled prediction score of the bag i over all its instances. Besides, $\sigma(\cdot)$ denotes the sigmoid activation.

Softmax pooling of predictions

The simplest approach is to take the maximum of predictions of all instances in the bag, i.e., $h(\mathbf{w}; \mathcal{X}) = \max_{\mathbf{x}\in\mathcal{X}} \phi(\mathbf{w}; \mathbf{x})$. However, the max operation is non-smooth, which usually causes difficulty in optimization. In practice, a smoothed-max (aka. log-sum-exp) pooling operator is used instead:

$$h(\mathbf{w}; \mathcal{X}) = \tau \log \left(\frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}\in\mathcal{X}} \exp(\phi(\mathbf{w}; \mathbf{x})/\tau) \right), \tag{6.106}$$

where $\tau > 0$ is a hyperparameter and $\phi(\mathbf{w}; \mathbf{x})$ is the prediction score for instance $\mathbf{x}$.

Mean pooling of predictions

The mean pooling operator just takes the average of predictions of individual instances, i.e., $h(\mathbf{w}; \mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \phi(\mathbf{w}; \mathbf{x})$. Indeed, smoothed-max pooling interpolates between the max pooling (with $\tau = 0$) and the mean pooling (with $\tau = \infty$).

Attention-based Pooling of features

Attention-based pooling aggregates the feature representations using attention, i.e.,

$$E(\mathbf{w}; \mathcal{X}) = \sum_{\mathbf{x} \in \mathcal{X}} \frac{\exp(g(\mathbf{w}; \mathbf{x}))}{\sum_{\mathbf{x}' \in \mathcal{X}} \exp(g(\mathbf{w}; \mathbf{x}'))} e(\mathbf{w}_e; \mathbf{x}), \tag{6.107}$$

where $g(\mathbf{w}; \mathbf{x})$ is a parametric function, e.g., $g(\mathbf{w}; \mathbf{x}) = \mathbf{w}_a^\top \tanh(Ve(\mathbf{w}_e; \mathbf{x}))$, where $V \in \mathbb{R}^{m \times d_o}$ and $\mathbf{w}_a \in \mathbb{R}^m$. Based on the aggregated feature representation, the bag level prediction can be computed by

$$h(\mathbf{w}; \mathcal{X}) = \sigma(\mathbf{w}_c^\top E(\mathbf{w}; \mathcal{X})) = \sigma\left(\sum_{\mathbf{x} \in \mathcal{X}} \frac{\exp(g(\mathbf{w}; \mathbf{x})) s(\mathbf{w}; \mathbf{x})}{\sum_{\mathbf{x}' \in \mathcal{X}} \exp(g(\mathbf{w}; \mathbf{x}'))} \right), \tag{6.108}$$

where $s(\mathbf{w}; \mathbf{x}) = \mathbf{w}_c^\top e(\mathbf{w}_e; \mathbf{x})$.

Optimization Algorithms

Given the pooled prediction $h(\mathbf{w}; \mathcal{X})$, the empirical risk minimization (ERM) problem is defined as:

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{N} \ell_i(h(\mathbf{w}; \mathcal{X}_i)).$$

The main challenge in solving this problem lies in the computational cost of evaluating $h(\mathbf{w}; \mathcal{X}_i)$, as it involves aggregating over potentially many instances.

To address this, we employ techniques from compositional optimization. Specifically, we express the smoothed-max pooling in (6.106) as a composition $h(\mathbf{w}; \mathcal{X}_i) = f_2(f_1(\mathbf{w}; \mathcal{X}_i))$, where the functions f_1 and f_2 are defined as:

$$f_1(\mathbf{w}; \mathcal{X}_i) = \frac{1}{|\mathcal{X}_i|} \sum_{\mathbf{x}_{i,j} \in \mathcal{X}_i} \exp(\phi(\mathbf{w}; \mathbf{x}_{i,j})/\tau),$$
$$f_2(s_i) = \tau \log(s_i).$$

Similarly, we express the attention-based pooling in (6.108) as a compositional function $h(\mathbf{w}; \mathcal{X}_i) = f_2(f_1(\mathbf{w}; \mathcal{X}_i))$, with:

$$f_1(\mathbf{w}; \mathcal{X}_i) = \begin{bmatrix} \frac{1}{|\mathcal{X}_i|} \sum_{\mathbf{x}_{i,j} \in \mathcal{X}_i} \exp(g(\mathbf{w}; \mathbf{x}_{i,j})) \mathbf{w}_c^\top e(\mathbf{w}_e; \mathbf{x}_{i,j}) \\ \frac{1}{|\mathcal{X}_i|} \sum_{\mathbf{x}_{i,j} \in \mathcal{X}_i} \exp(g(\mathbf{w}; \mathbf{x}_{i,j})) \end{bmatrix}, \quad f_2(\mathbf{u}_i) = \sigma\left(\frac{[\mathbf{u}_i]_1}{[\mathbf{u}_i]_2}\right).$$

The key difference between the two pooling mechanisms is that the inner function f_1 in attention-based pooling is a vector-valued function with two components. In both cases, the computational bottleneck lies in computing $f_1(\mathbf{w}; \mathcal{X}_i)$.

To reduce this cost, we maintain a dynamic estimator $u_{i,t}$ for each bag $\mathcal{X}_i$. At iteration t, for any $\mathcal{X}_i \in \mathcal{B}_{o,t}$ (a mini-batch of bags), we update the estimator as:

$$u_{i,t} = (1-\gamma) u_{i,t-1} + \gamma f_1(\mathbf{w}_t; \mathcal{B}_{i,t}), \tag{6.109}$$

where $\mathcal{B}_{i,t} \subset \mathcal{X}_i$ is a mini-batch of instances sampled from $\mathcal{X}_i$, and $\gamma \in [0,1]$ is a smoothing parameter. For smoothed-max pooling, this becomes:

$$u_{i,t} = (1-\gamma) u_{i,t-1} + \frac{\gamma}{|\mathcal{B}_{i,t}|} \sum_{\mathbf{x}_{i,j} \in \mathcal{B}_{i,t}} \exp(\phi(\mathbf{w}_t; \mathbf{x}_{i,j})/\tau), \tag{6.110}$$

and for attention-based pooling, we update:

$$\mathbf{u}_{i,t} = (1-\gamma) \mathbf{u}_{i,t-1} + \gamma \begin{bmatrix} \frac{1}{|\mathcal{B}_{i,t}|} \sum_{\mathbf{x}_{i,j} \in \mathcal{B}_{i,t}} \exp(g(\mathbf{w}_t; \mathbf{x}_{i,j})) \delta(\mathbf{w}_t; \mathbf{x}_{i,j}) \\ \frac{1}{|\mathcal{B}_{i,t}|} \sum_{\mathbf{x}_{i,j} \in \mathcal{B}_{i,t}} \exp(g(\mathbf{w}_t; \mathbf{x}_{i,j})) \end{bmatrix}. \tag{6.111}$$

The corresponding vanilla gradient estimator for softmax pooling is:

$$\mathbf{z}_t = \frac{1}{|\mathcal{B}|} \sum_{\mathcal{X}_i \in \mathcal{B}} \ell_i'(f_2(u_{i,t})) \nabla f_2(u_{i,t}) \frac{1}{|\mathcal{B}_{i,t}|} \sum_{\mathbf{x}_{i,j} \in \mathcal{B}_{i,t}} \nabla \exp(\phi(\mathbf{w}_t; \mathbf{x}_{i,j})/\tau), \tag{6.112}$$

and for attention-based pooling:

$$\mathbf{z}_t = \tag{6.113}$$
$$\frac{1}{|\mathcal{B}|} \sum_{\mathcal{X}_i \in \mathcal{B}} \ell_i'(f_2(\mathbf{u}_{i,t})) \begin{bmatrix} \frac{1}{|\mathcal{B}_{i,t}|} \sum_{\mathbf{x}_{i,j} \in \mathcal{B}_{i,t}} \nabla \left(\exp(g(\mathbf{w}_t; \mathbf{x}_{i,j})) s(\mathbf{w}^t; \mathbf{x}_{i,j})\right) \\ \frac{1}{|\mathcal{B}_{i,t}|} \sum_{\mathbf{x}_{i,j} \in \mathcal{B}_{i,t}} \nabla \exp(g(\mathbf{w}_t; \mathbf{x}_{i,j})) \end{bmatrix}^\top \nabla f_2(\mathbf{u}_{i,t}).$$

Then we can update the model parameter $\mathbf{w}_{t+1}$ by Momentum, Adam, or Adam-W methods.

As established in Chapter 5, the theory of compositional optimization guarantees that the moving average estimators $\mathbf{u}_{i,t}$ ensure the average estimation error,

$$\frac{1}{T} \sum_{t=1}^{T} \|\mathbf{u}_{i,t} - f_1(\mathbf{w}_t; \mathcal{X}_i)\|_2^2,$$

converges to zero as $T \to \infty$, provided that the model parameters and hyperparameters are properly updated.

6.9 DRRHO Risk Minimization

As a last application of compositional optimization, we consider an emerging problems in AI. With the success of large foundation models, numerous companies and research groups have entered the race to develop state-of-the-art models. While the data and code are often proprietary, the resulting models are sometimes released publicly, such as the CLIP models from OpenAI. How can we leverage these open-weight models? We discuss three commonly used strategies and then present an emerging paradigm.

Using the Model As-Is

A straightforward strategy for leveraging open-weight foundation models is to use them as-is. This approach requires no additional training and can be deployed immediately, making it highly convenient and cost-effective. It is particularly attractive when computational resources or labeled data are limited. However, the downside is that the pretrained model may not perform well on specialized tasks or under distribution shifts, where its generic knowledge does not fully align with the requirements of the target application.

Fine-Tuning the Model

An alternative strategy is to use the pretrained model as a starting point for fine-tuning. By performing minimal task-specific training, the model can be adapted to new domains with relatively low computational and data costs. Fine-tuning generally yields better performance than using the model out-of-the-box. Nevertheless, since the model architecture remains unchanged and the updates are typically modest, the improvements in performance may be limited, particularly when the pretrained model is already near-optimal for its design.

Knowledge Distillation from the Model

A more flexible approach involves using the pretrained model as a teacher in a knowledge distillation framework. Here, a smaller or more efficient student model is trained to mimic the teacher's outputs, enabling knowledge transfer that can improve training efficiency and generalization. This strategy is particularly useful for deploying models in resource-constrained environments. The main drawback, however, is that the student model is usually less expressive than the teacher, which can cap its performance despite potential gains in speed and efficiency.

Reference Model Steering for training from scratch

An emerging learning paradigm has recently surfaced that leverages a pre-trained reference model to guide and enhance training via strategic data weighting—a process we term reference model steering. Unlike the knowledge distillation framework, reference model steering does not assume that the reference model is a stronger teacher; in fact, it can lead to the training of a model that ultimately surpasses the reference model in performance, i.e., enabling weak to strong generalization.

DRRHO Risk Minimization

Let $\mathbf{z} \sim \mathbb{P}$ denote a random data point drawn from distribution $\mathbb{P}$, and let $\mathbf{w} \in \mathcal{W}$ represent model parameters from a parameter space $\mathcal{W}$. Given a loss function $\ell(\mathbf{w}, \mathbf{z})$, the expected risk is defined as:

$$\mathcal{R}(\mathbf{w}) = \mathbb{E}_{\mathbf{z}\sim\mathbb{P}}[\ell(\mathbf{w}, \mathbf{z})].$$

Given a pretrained reference model $\mathbf{w}_{\text{ref}}$, we define a new loss $\hat{\ell}(\mathbf{w}, \cdot) = \ell(\mathbf{w}, \cdot) - \ell(\mathbf{w}_{\text{ref}}, \cdot)$, which is termed as RHO loss. Incorporating this into the distributionally robust optimization (DRO) framework (2.12), we define DRRHO risk minimization as:

$$\min_{\mathbf{w}\in\mathcal{W}} \sup_{\substack{\mathbf{p}\in\Delta \\ D_\phi(\mathbf{p}\,\|\,1/n)\le\rho/n}} \sum_{i=1}^{n} p_i \left(\ell(\mathbf{w}, \mathbf{z}_i) - \ell(\mathbf{w}_{\text{ref}}, \mathbf{z}_i)\right). \tag{6.114}$$

Theoretical guarantees for DRRHO have been developed with the χ^2 divergence, i.e., $D_\phi(\mathbf{p}\,\|\,\mathbf{q}) = \sum_{i=1}^{n} \frac{1}{2} q_i \left(\frac{p_i}{q_i} - 1\right)^2$. Under mild conditions, it can be shown that with high probability:

$$\mathcal{R}(\tilde{\mathbf{w}}_*) \le \inf_{\mathbf{w}\in\mathcal{W}} \left(\mathcal{R}(\mathbf{w}) + \sqrt{\frac{2\rho}{n} \operatorname{Var}(\ell(\mathbf{w}, \cdot) - \ell(\mathbf{w}_{\text{ref}}, \cdot))}\right) + O\left(\frac{1}{n}\right). \tag{6.115}$$

where $\tilde{\mathbf{w}}_*$ is an optimal solution to DRRHO risk minimization.

In particular, plugging in $\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathcal{W}} \mathcal{R}(\mathbf{w})$ yields:

$$\mathcal{R}(\tilde{\mathbf{w}}_*) \le \mathcal{R}(\mathbf{w}_*) + \sqrt{\frac{2\rho}{n} \operatorname{Var}(\ell(\mathbf{w}_*, \cdot) - \ell(\mathbf{w}_{\text{ref}}, \cdot))} + O\left(\frac{1}{n}\right).$$

This result provides valuable insight: if the reference model $\mathbf{w}_{\text{ref}}$ is well-trained such that $\ell(\mathbf{w}_{\text{ref}}, \cdot)$ closely matches $\ell(\mathbf{w}_*, \cdot)$ in distribution, then the variance term becomes small. As a result, DRRHO achieves better generalization than the standard $O(\sqrt{1/n})$ bound of ERM.

Furthermore, if $\mathbf{w}_{\text{ref}} \in \mathcal{W}$, we obtain a comparison in terms of excess risk:

$$\mathcal{R}(\tilde{\mathbf{w}}_*) - \mathcal{R}(\mathbf{w}_*) \le \mathcal{R}(\mathbf{w}_{\text{ref}}) - \mathcal{R}(\mathbf{w}_*) + O\left(\frac{1}{n}\right).$$

This enables a direct comparison between the DRRHO minimizer $\tilde{\mathbf{w}}_*$ and the reference model $\mathbf{w}_{\text{ref}}$ from the same hypothesis class. Suppose $\mathbf{w}_{\text{ref}}$ was trained via ERM on a dataset with m samples. Then standard generalization theory gives an excess risk of order $O(1/\sqrt{m})$. In contrast, to match this level of generalization error, DRRHO requires only $n = O(\sqrt{m})$ samples—significantly improving over the $O(m)$ sample complexity required by ERM without a reference model.

Optimization Algorithms

When the CVaR is used defined by $\phi(t) = 1$ if $t \leq n/k$ and $\phi(t) = \infty$ otherwise, the DRRHO risk reduces to the average of the top-k RHO losses:

$$\min_{\mathbf{w}} F(\mathbf{w}) := \frac{1}{k} \sum_{i=1}^{k} \left(\ell(\mathbf{w}, \mathbf{z}_{[i]}) - \ell(\mathbf{w}_{\text{ref}}, \mathbf{z}_{[i]}) \right), \tag{6.116}$$

where $\mathbf{z}_{[i]}$ denotes the data point ranked i-th in descending order based on its RHO loss. This problem can be equivalently reformulated as:

$$\min_{\mathbf{w}, \nu} \frac{1}{k} \sum_{i=1}^{n} [\ell(\mathbf{w}, \mathbf{z}_i) - \ell(\mathbf{w}_{\text{ref}}, \mathbf{z}_i) - \nu]_+ + \nu, \tag{6.117}$$

which is more amenable to gradient-based optimization techniques.

When DRRHO risk is defined using KL divergence regularization, the objective becomes:

$$\min_{\mathbf{w}} \tau \log \left(\frac{1}{n} \sum_{i=1}^{n} \exp \left(\frac{\ell(\mathbf{w}, \mathbf{z}_i) - \ell(\mathbf{w}_{\text{ref}}, \mathbf{z}_i)}{\tau} \right) \right). \tag{6.118}$$

This formulation can be optimized by simply replacing the loss in Algorithm 24 with the RHO loss. The vanilla gradient at iteration t is estimated by:

$$\frac{1}{B} \sum_{i \in \mathcal{B}_t} \frac{\exp\left(\frac{\ell(\mathbf{w}_t, \mathbf{z}_i) - \ell(\mathbf{w}_{\text{ref}}, \mathbf{z}_i)}{\tau} \right)}{u_t} \nabla \ell(\mathbf{w}_t, \mathbf{z}_i), \tag{6.119}$$

where u_t is the MA estimator of the inner function value. This gradient estimator naturally assigns higher weights to data points with larger RHO losses, thereby prioritizing samples with high learnability during training.

Finally, when DRRHO is formulated with a KL-divergence constraint, the optimization problem becomes:

$$\min_{\mathbf{w}} \min_{\tau \geq 0} \tau \log \left(\frac{1}{n} \sum_{i=1}^{n} \exp \left(\frac{\ell(\mathbf{w}, \mathbf{z}_i) - \ell(\mathbf{w}_{\text{ref}}, \mathbf{z}_i)}{\tau} \right) \right) + \frac{\tau \rho}{n}. \tag{6.120}$$

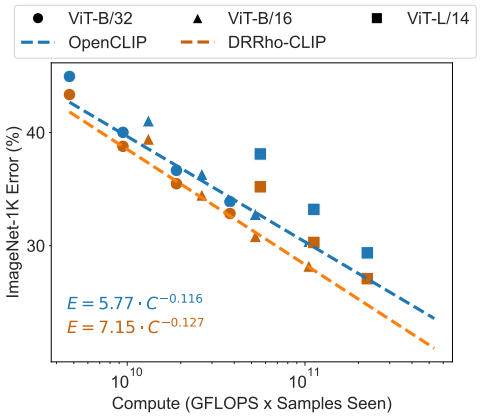


Fig. 6.33: Scaling performance of OpenCLIP and DRRho-CLIP, which uses the OpenAI CLIP model as the reference model. We conduct experiments of the two methods under different settings to fit scaling laws, as shown in the bottom left corner. For more details, please refer to (Wei et al., 2025).

This formulation can be optimized using techniques similar to those introduced in the first section of this chapter.

DRRHO-CLIP with a Reference Model

We now consider applying the DRRHO risk minimization framework to CLIP. Given the established connection between robust global contrastive loss and DRO, as shown in (6.44) and (6.45), it is straightforward to incorporate the RHO loss into the training objective. Define the following loss components:

$$\begin{aligned}
\ell_1(\mathbf{w}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{t}) &= s(\mathbf{w}; \mathbf{x}_i, \mathbf{t}) - s(\mathbf{w}; \mathbf{x}_i, \mathbf{t}_i), \\
\ell_2(\mathbf{w}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{x}) &= s(\mathbf{w}; \mathbf{x}, \mathbf{t}_i) - s(\mathbf{w}; \mathbf{x}_i, \mathbf{t}_i), \\
\ell_1(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{t}) &= s(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}) - s(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}_i), \\
\ell_2(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{x}) &= s(\mathbf{w}_{\text{ref}}; \mathbf{x}, \mathbf{t}_i) - s(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}_i),
\end{aligned}$$

where $s(\cdot;\cdot,\cdot)$ denotes the similarity function, and $\mathbf{w}_{\text{ref}}$ is a pretrained reference model.

Using these definitions, we modify the original objective in (6.49) to incorporate the RHO loss:

$$\begin{aligned}
\min_{\mathbf{w},\tau_1,\tau_2} \frac{1}{n}\sum_{i=1}^{n} \tau_1 \log\left(\frac{1}{|\mathcal{T}_i^-|}\sum_{\mathbf{t}\in\mathcal{T}_i^-} \exp\left(\frac{\ell_1(\mathbf{w}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{t}) - \ell_1(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{t})}{\tau_1}\right)\right) + \tau_1\rho \\
+ \frac{1}{n}\sum_{i=1}^{n} \tau_2 \log\left(\frac{1}{|\mathcal{I}_i^-|}\sum_{\mathbf{x}\in\mathcal{I}_i^-} \exp\left(\frac{\ell_2(\mathbf{w}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{x}) - \ell_2(\mathbf{w}_{\text{ref}}; \mathbf{x}_i, \mathbf{t}_i, \mathbf{x})}{\tau_2}\right)\right) + \tau_2\rho.
\end{aligned} \tag{6.121}$$

This objective can be optimized using an algorithm similar to that used in the CLIP training. Empirical results show that this approach significantly reduces sample

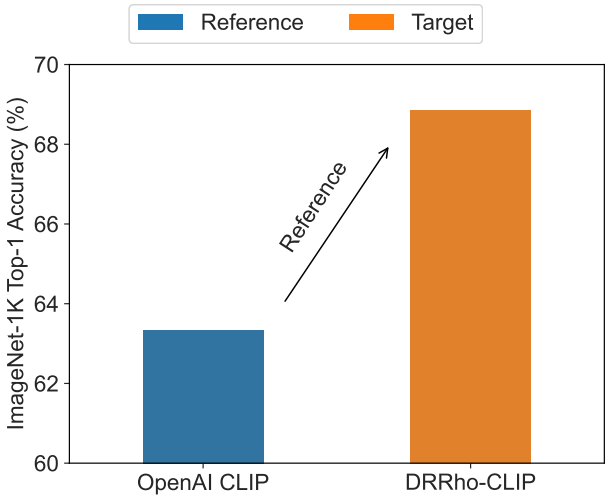


Fig. 6.34: Comparison between a target model (ViT-B/16) trained by DRRRHO-CLIP and the reference model it leverages. OpenAI CLIP (ViT-B/32) was trained on a private 400M dataset with 12.8B samples seen and 32768 batch size. DRRho-CLIP model was trained on DFN-192M with 1.28B samples seen and 5120 batch size, and using OpenAI CLIP as a reference model. DRRHO-CLIP training took 376 GPU hours on 8 H100 (2 days), OpenAI CLIP (ViT-L/14) model was trained on 256 V100 with 12 days, which gives an estimate of 256*12*24/11.6=6356 GPU hours for training ViT-B/32 as its FLOPs is 11.6 smaller. For more details, please refer to (Wei et al., 2025).

complexity and improves the empirical scaling law (see Figure 6.33), while also achieving weak to strong generalization (see Figure 6.34).

6.10 History and Notes

DRO and GDRO.

We first formulated KL-regularized Distributionally Robust Optimization (DRO) as a stochastic compositional optimization (SCO) problem in (Qi et al., 2021b), utilizing STORM-based estimators. This line of research was further developed in (Qi et al., 2020), which introduced an attentional biased stochastic momentum method for KL-regularized DRO with specific applications in imbalanced data classification. Subsequently, we extended both the algorithmic framework and theoretical analysis to address KL-constrained DRO (Qi et al., 2023). Collectively, these works demon-

strate the advantages of employing compositional optimization techniques over traditional primal-dual methods for solving DRO problems.

The formulation of FCCO for group DRO (GDRO) was initially identified in (Hu et al., 2024b). Building on this, Wang and Yang (2023) applied the ALEXR algorithm to convex group DRO, demonstrating significant improvements over traditional stochastic primal-dual methods. Most recently, the application of SONEX to non-convex group DRO within the context of deep learning was investigated by Chen et al. (2025b).

Stochastic AUC and NDCG Optimization.

Stochastic AUC maximization has a long-standing history in machine learning, as detailed in our survey (Yang and Ying, 2023). The formulation of AUC maximization with a square surrogate loss as a minimax optimization problem was first introduced by Ying et al. (2016b). Building on this foundation, we developed the first convergence analysis for stochastic non-convex minimax optimization in the context of deep AUC maximization (Liu et al., 2020). While this work was inspired by our previous work on weakly-convex strongly-concave minimax optimization (Rafique et al., 2022), it established a superior complexity bound by leveraging the PL condition. These theoretical results were subsequently strengthened in (Guo et al., 2023).

This line of research eventually facilitated our winning entry in the CheXpert competition for X-ray image classification (Yuan et al., 2021), which also introduced the AUC-margin minimax objective. Notably, all of these proposed methods utilize a double-loop algorithmic structure. The single-loop PDMA and PDAdam methods for deep AUC maximization was first proposed and analyzed in our work (Guo et al., 2021b). The compositional training method for deep AUC maximization that facilitates the feature learning and classifier learning in a unified framework was proposed in our work (Yuan et al., 2022b).

The SOAP algorithm represents the first method of its kind to offer a convergence guarantee that does not rely on the use of large batch sizes, which has challenged the computer vision and machine learning communities for many years (see references in (Qi et al., 2021c)). The SOPA and SOPAs algorithms for one-way partial AUC maximization and STOAs for two-way partial AUC maximization were developed and analyzed in (Zhu et al., 2022b). The STACO algorithm for two-way partial AUC maximization was proposed in (Zhou et al., 2025). These stdudies have addressed long-standing open problems for efficient partial AUC maximization with convergence guarantee (Kar et al., 2014; Narasimhan and Agarwal, 2013).

The formulation of stochastic NDCG optimization as FCCO was proposed in our work (Qiu et al., 2022), which also developed a multi-block bilevel optimization formulation and algorithm for optimizing top-K NDCG. The complexity for multi-block bilevel optimization was improved in (Hu et al., 2023) by using the MSVR estimators.

The design and benchmark of LibAUC library was presented in (Yuan et al., 2023a).

Discriminative Learning of Foundation models.

The SogCLR algorithm was inspired by the SOX framework for FCCO; its advantages over SimCLR, particularly regarding efficiency with small batch sizes in unimodal contrastive learning, were demonstrated in (Yuan et al., 2022c). Building on this, we introduced iSogCLR in (Qiu et al., 2023) to optimize individualized temperatures. This advancement was also informed by our previous research on KL-constrained DRO (Qi et al., 2023).

Subsequently, we proposed TempNet (Qiu et al., 2024), which has been successfully applied to CLIP training and the pretraining of large language models (LLMs). Furthermore, a comprehensive evaluation of FCCO-based techniques for distributed CLIP training was recently provided in (Wei et al., 2024).

The discriminative fine-tuning approach of LLMs was proposed in our work (Guo et al., 2025). The DisCO method for fine-tuning large reasoning models was developed in our work (Li et al., 2025).

FCCO for Constrained Learning.

The application of compositional optimization techniques to penalty methods for constrained learning dates back to (Ermoliev and Wets, 1988). The first non-asymptotic analysis of the penalty method with a squared hinge penalty function for non-convex inequality constrained optimization based on FCCO was conducted in our work (Li et al., 2024). This work investigated the problem within the context of continual learning under zero-forgetting constraints and established a complexity of $O(1/\epsilon^7)$ for finding an ϵ-KKT solution. Additionally, we developed a theoretical framework to characterize the benefits of network expansion in facilitating constrained learning with non-forgetting constraints. The ROC fairness constraint was first considered in (Vogel et al., 2020).

Subsequent advancements have further improved the complexity of penalty based methods based on FCCO. By employing SONX for the hinge penalty, the complexity was reduced to $O(1/\epsilon^6)$ (Yang et al., 2025). More recently, the introduction of SONEX and a double-loop ALEXR method for the squared hinge penalty achieved a complexity of $O(1/\epsilon^5)$ (Chen et al., 2025b). This currently represents the state-of-the-art complexity for penalty methods in non-convex constrained optimization.

Learning with data compositional networks.

Graph convolutional neural network was proposed by Kipf and Welling (2017). GraphSAGE was developed in (Hamilton et al., 2017). The use of compositional optimization techniques, specifically incorporating feature momentum for large-scale Graph Neural Network (GNN) learning, was introduced in our previous work (Yu et al., 2022). Furthermore, the application of compositional optimization to multi-instance learning, utilizing compositional pooling operations, was first proposed

in (Zhu et al., 2023a). Attention-based pooling for multi-instance learning was proposed by Ilse et al. (2018).

DRRHO risk minimization.

The development of DRRHO risk minimization framework and its application to CLIP training was introduced in our work (Wei et al., 2025). The theoretical analysis of this method is largely built upon the foundations of DRO (Namkoong and Duchi, 2017), while the conceptual idea of using the RHO loss for data selection in a mini-batch was originally proposed in (Mindermann et al., 2022).