GroupDRO with Heterogeneous Feature Spaces

October 22, 2025

1 Problem Setup

We consider a supervised learning problem with data drawn from multiple *groups*, where each group may have a distinct feature representation space. Formally, we are given a centralized dataset

$$\mathcal{D} = \{(x_i, y_i, g_i)\}_{i=1}^{N}, \tag{1}$$

where for each sample i:

- $x_i \in \mathcal{X}_{g_i} \subset \mathbb{R}^{k_{g_i}}$ is the input feature vector, belonging to a group-specific feature space $\mathcal{X}_{g_i}$ of dimension k_{g_i},
- $y_i \in \{1, \ldots, C\}$ is the class label, and
- $g_i \in \{1, \ldots, G\}$ denotes the group membership.

Each group g thus has its own feature space $\mathcal{X}_g$, and data within the same group share a consistent representation, but feature representations may differ across groups (e.g., in scale, dimension, or modality).

Our objective is to learn a predictive model that can correctly classify all samples while accounting for the heterogeneity across group-specific feature spaces. To achieve this, we introduce group-specific representation functions that map local feature spaces into a shared latent space, and a shared classifier that operates on this common representation.

Let $\phi_g : \mathcal{X}_g \to \Phi \subset \mathbb{R}^k$ denote the representation function (or embedding function) for group g, where Φ is a common latent space of fixed dimension k. Let $f_\theta : \Phi \to \Delta^{C-1}$ denote a shared classifier parameterized by θ, where Δ^{C-1} is the $(C-1)$-dimensional probability simplex. The overall model prediction for a data point (x_i, g_i) is thus given by

$$\hat{p}(y \mid x_i, g_i) = f_\theta(\phi_{g_i}(x_i)). \tag{2}$$

We will define the full training objective as a combination of (i) a supervised loss ensuring predictive performance and (ii) a representation alignment loss ensuring that the group-specific embeddings are consistent in the shared latent space.

2 Methodology

Our approach follows the intuition of mapping group-specific data representations into a shared latent space where a common classifier can operate effectively. The model is composed of three main components: (1) group-specific representation functions, (2) class-conditional latent anchor distributions, and (3) a shared classifier. Each component plays a distinct role in enabling learning across heterogeneous feature spaces.

2.1 Group-specific Representation Functions

For each group $g \in \{1, \ldots, G\}$, we define a learnable representation function

$$\phi_g : \mathcal{X}_g \to \Phi \subset \mathbb{R}^k, \tag{3}$$

that maps inputs from the group-specific feature space $\mathcal{X}_g$ into a common latent space Φ of fixed dimension k. This mapping allows data originating from heterogeneous sources (e.g., differing dimensionalities, modalities, or preprocessing pipelines) to be compared and classified in a unified latent representation.

2.2 2. Class-Conditional Anchor Distributions

To ensure that embeddings from different groups are geometrically consistent in the latent space, we associate each semantic class with a global latent *anchor distribution*. Intuitively, these anchors represent the "average shape" of how a particular class (e.g., the digit "3" in digit classification) is distributed in the shared space, regardless of which group the data come from.

Global anchor distributions. For each class $c \in \{1, \ldots, C\}$, we define a global anchor distribution μ_c in the latent space Φ. Each anchor is modeled as a multivariate Gaussian distribution:

$$\mu_c = \mathcal{N}(m_c, S_c), \tag{4}$$

where $m_c \in \mathbb{R}^k$ is the mean vector and $S_c \in \mathbb{S}^k_{++}$ is a positive-definite covariance matrix. The mean m_c represents the central location of class c in the latent space, while the covariance S_c captures the spread and orientation of the distribution.

Definition of a Gaussian distribution. A multivariate Gaussian (or normal) distribution over $\mathbb{R}^k$ with mean vector m and covariance matrix S has the following probability density function:

$$\mathcal{N}(z \mid m, S) = \frac{1}{(2\pi)^{k/2}|S|^{1/2}} \exp\!\left(-\tfrac{1}{2}(z-m)^\top S^{-1}(z-m)\right), \tag{5}$$

where $|S|$ denotes the determinant of S. Intuitively, samples z drawn from $\mathcal{N}(m, S)$ tend to lie close to the mean m, with deviations shaped by the covariance S.

Empirical distributions for each group. For every group g and class c, the model can estimate how that class appears within group g by looking at all embedded samples that belong to that group and class:

$$\{\, \phi_g(x_i) \mid y_i = c,\, g_i = g \,\}.$$

From these embeddings, we compute an *empirical Gaussian approximation*

$$\widehat{\nu}_{g,c} = \mathcal{N}(\widehat{m}_{g,c}, \widehat{S}_{g,c}),$$

where $\widehat{m}_{g,c}$ and $\widehat{S}_{g,c}$ are the empirical mean and covariance:

$$\widehat{m}_{g,c} = \frac{1}{n_{g,c}} \sum_{i:y_i=c,\, g_i=g} \phi_g(x_i), \tag{6}$$

$$\widehat{S}_{g,c} = \frac{1}{n_{g,c}-1} \sum_{i:y_i=c,\, g_i=g} \big(\phi_g(x_i) - \widehat{m}_{g,c}\big)\big(\phi_g(x_i) - \widehat{m}_{g,c}\big)^\top, \tag{7}$$

and $n_{g,c}$ is the number of samples in group g belonging to class c.

Alignment objective. The model encourages each empirical distribution $\widehat{\nu}_{g,c}$ to align with the corresponding global anchor μ_c. This is achieved by penalizing the geometric distance between the two Gaussians using the 2-Wasserstein distance, which measures how far apart the two distributions are in both their means and covariances. The exact mathematical form of this penalty will be introduced in the loss formulation section.

Illustrative example. Suppose the latent space is two-dimensional ($k = 2$) and we consider class $c = 1$ (e.g., the digit "3"). Assume group $g = 1$ has three embedded samples:

$$\phi_1(x_1) = (1.0, 0.8), \quad \phi_1(x_2) = (1.2, 1.0), \quad \phi_1(x_3) = (0.8, 0.9).$$

Then the empirical mean and covariance for this class-group pair are

$$\widehat{m}_{1,1} = \frac{1}{3}[(1.0, 0.8) + (1.2, 1.0) + (0.8, 0.9)] = (1.0, 0.9),$$

$$\widehat{S}_{1,1} = \begin{bmatrix} 0.0267 & 0.0133 \\ 0.0133 & 0.0100 \end{bmatrix}.$$

This small Gaussian $\widehat{\nu}_{1,1}$ summarizes how group 1 represents class 1 in the latent space, and the model will attempt to align it with the global anchor $\mu_1 = \mathcal{N}(m_1, S_1)$ shared across all groups.

2.3 Shared Classifier

A shared classifier $f_\theta : \Phi \to \Delta^{C-1}$, parameterized by θ, maps latent embeddings to class probabilities:

$$\hat{p}(y \mid x, g) = f_\theta(\phi_g(x)). \tag{8}$$

3 Loss Formulation

We minimize a sum of three terms: a supervised classification loss, an anchor-fitting loss, and an inter-class separation loss. Let $z_i = \phi_{g_i}(x_i) \in \mathbb{R}^k$.

3.1 Classification Loss

The model predicts $\hat{p}(\cdot \mid x_i, g_i) = f_\theta(z_i) \in \Delta^{C-1}$. A standard choice is the *cross-entropy* loss:

$$\mathcal{L}_{\text{clf}}(\theta, \{\phi_g\}) \;=\; -\frac{1}{N}\sum_{i=1}^{N} \log f_\theta(z_i)_{y_i}. \tag{9}$$

3.2 Anchor-Fitting Loss (Aligning Empirical and Learned Distributions)

The second component of the objective ensures that the learned latent space reflects a coherent global structure across classes. It achieves this by aligning the *empirical latent distribution* estimated from the data with the *learned anchor distribution* that defines what each class should ideally look like in the latent space.

Latent representations. After the input x_i from group g_i is passed through its corresponding encoder, we obtain a latent representation

$$z_i = \phi_{g_i}(x_i) \in \Phi \subset \mathbb{R}^k. \tag{10}$$

These latent embeddings $\{z_i\}$ form the foundation for estimating how each class is distributed in the latent space.

Empirical class distributions. From the latent representations, we estimate the *empirical latent distribution* for each class $c \in \{1, \dots, C\}$ by pooling all samples of that class from every group:

$$v_\phi^c \;=\; \mathcal{N}(\widehat{m}_c, \widehat{S}_c),$$

where the parameters $\widehat{m}_c$ and $\widehat{S}_c$ are computed directly from data as

$$\widehat{m}_c = \frac{1}{N_c}\sum_{i:y_i=c} z_i, \tag{11}$$

$$\widehat{S}_c = \frac{1}{N_c - 1}\sum_{i:y_i=c} (z_i - \widehat{m}_c)(z_i - \widehat{m}_c)^\top. \tag{12}$$

Here, N_c is the total number of samples belonging to class c. Thus, v_ϕ^c represents the **observed distribution of embeddings** for class c under the current encoder parameters $\{\phi_g\}$.

Learned anchor distributions. Independently, we maintain a *learnable* Gaussian anchor distribution

$$\mu_c = \mathcal{N}(m_c, S_c),$$

where m_c and S_c are free parameters that define the idealized, "true" latent representation of class c. These anchors represent our target geometry for each class in the latent space: they are not directly observed but are learned jointly with the encoders.

Goal of the anchor-fitting loss. The purpose of the anchor-fitting loss is to make the empirical distributions v_ϕ^c (estimated from the data) as close as possible to the learned anchors μ_c (which define the desired latent geometry). This is achieved by minimizing the squared 2-Wasserstein distance between them:

$$\mathcal{L}_{\text{fit}}(\{\phi_g\}, \{m_c, S_c\}) = \sum_{c=1}^{C} W_2^2\Big(\mathcal{N}(\widehat{m}_c, \widehat{S}_c), \mathcal{N}(m_c, S_c)\Big). \tag{13}$$

The closed-form expression for the Wasserstein distance between two Gaussian distributions $\mathcal{N}(m_1, S_1)$ and $\mathcal{N}(m_2, S_2)$ is given by:

$$W_2^2\big(\mathcal{N}(m_1, S_1),\, \mathcal{N}(m_2, S_2)\big) = \|m_1 - m_2\|_2^2 + \operatorname{Tr}\Big(S_1 + S_2 - 2\,(S_1^{1/2} S_2 S_1^{1/2})^{1/2}\Big). \tag{14}$$

3.3 Inter-Class Separation via Anchor Classification

In our centralized setting, each class c is associated with a learned anchor distribution $\mu_c = \mathcal{N}(m_c, S_c)$ in the latent space. After the anchor-fitting loss aligns each class's empirical latent distribution v_ϕ^c with its corresponding anchor μ_c, the latent space may still become degenerate if different anchors overlap. To prevent such overlap, we add an *inter-class separation loss* that encourages the anchors to define distinct and discriminative regions of the latent space.

Idea. Rather than explicitly penalizing the distance between anchors, we enforce separation by requiring the classifier f_θ to correctly classify synthetic latent samples drawn from each anchor distribution. If the anchors overlap, samples from different classes will be difficult to distinguish, and the classifier's loss will increase. Minimizing this loss implicitly pushes the anchors and the classifier toward a configuration where each class occupies a well-separated region.

Formulation. For each class c, we draw J latent samples $Z_c^{(j)} \sim \mu_c = \mathcal{N}(m_c, S_c)$, feed them into the shared classifier f_θ, and apply a classification loss with the corresponding class label:

$$\mathcal{L}_{\text{sep}} = \frac{1}{C} \sum_{c=1}^{C} \frac{1}{J} \sum_{j=1}^{J} \ell\Big(c,\ f_\theta\big[\,Z_c^{(j)}\,\big]\Big), \qquad Z_c^{(j)} \sim \mathcal{N}(m_c, S_c), \tag{15}$$

where ℓ is typically the cross-entropy loss.

3.4 Total Objective

Combining the three pieces, with weights $\lambda_{\text{fit}}, \lambda_{\text{sep}} > 0$,

$$\mathcal{L}_{\text{total}} \;=\; \mathcal{L}_{\text{clf}} \;+\; \lambda_{\text{fit}}\, \mathcal{L}_{\text{fit}} \;+\; \lambda_{\text{sep}}\, \mathcal{L}_{\text{sep}}, \tag{16}$$

We optimize jointly over θ, the encoders $\{\phi_g\}$, and anchor parameters $\{m_c, S_c\}$.

4 Algorithm and Implementation Details

This section specifies the complete training procedure in a centralized setting. We describe (i) how mini-batches are formed, (ii) how each loss term is computed from a batch, and (iii) how parameters are updated. Throughout, k is the latent dimension and C is the number of classes.

4.1 Notation and Trainable Parameters

- Group encoders $\{\phi_g\}_{g=1}^{G}$ mapping $x \in \mathcal{X}_g$ to $z = \phi_g(x) \in \mathbb{R}^k$.
- Shared classifier $f_\theta : \mathbb{R}^k \to \Delta^{C-1}$.
- Class anchors $\mu_c = \mathcal{N}(m_c, S_c)$ with $m_c \in \mathbb{R}^k$ and $S_c \in \mathbb{S}^k_{++}$.

4.2 Mini-batching

At each iteration, sample a mini-batch

$$\mathcal{B} = \{(x_i, y_i, g_i)\}_{i=1}^{B}$$

by uniform sampling from the full dataset $\mathcal{D}$. Ideally you want each batch to be stratified by groups, meaning that the proportions of the groups in the batch are the same as the proportion of the groups in the initial dataset. For every $i \in \{1, \ldots, B\}$, compute the latent embedding

$$z_i = \phi_{g_i}(x_i) \in \mathbb{R}^k.$$

4.3 Per-batch Computations

(1) Classification loss $\mathcal{L}_{\text{clf}}$. Compute the softmax outputs $\hat{p}_i = f_\theta(z_i)$ and the cross-entropy

$$\mathcal{L}^{\mathcal{B}}_{\text{clf}} = -\frac{1}{B} \sum_{i=1}^{B} \log \hat{p}_i[y_i].$$

(2) Compute empirical moments for $\mathcal{L}_{\text{fit}}$. The empirical latent distribution of each class is estimated directly from the current mini-batch (or the full training data if available). For every class c represented in the batch, we compute:

$$\widehat{m}_c = \frac{1}{n_c} \sum_{i \in \mathcal{B}:\, y_i = c} z_i, \qquad \widehat{S}_c = \frac{1}{n_c - 1} \sum_{i \in \mathcal{B}:\, y_i = c} (z_i - \widehat{m}_c)(z_i - \widehat{m}_c)^\top + \varepsilon I,$$

where n_c is the number of samples of class c in the current batch and εI ensures numerical stability. These statistics define the empirical Gaussian $v_\phi^c = \mathcal{N}(\widehat{m}_c, \widehat{S}_c)$, which is used immediately in the anchor-fitting loss to measure its alignment with the corresponding anchor μ_c.

(3) Anchor-fitting loss $\mathcal{L}_{\text{fit}}$. For every class c seen so far, compute the squared 2-Wasserstein distance between Gaussians

$$W_2^2\big(\mathcal{N}(\widehat{m}_c, \widehat{S}_c), \mathcal{N}(m_c, S_c)\big) = \|\widehat{m}_c - m_c\|_2^2 + \operatorname{Tr}\big(\widehat{S}_c + S_c - 2(\widehat{S}_c^{1/2} S_c \widehat{S}_c^{1/2})^{1/2}\big),$$

and sum over classes to get $\mathcal{L}_{\text{fit}}^{\mathcal{B}}$. Matrix square roots are computed via an eigenvalue decomposition with eigenvalues clamped below ε for stability.

(4) Inter-class separation via anchor classification $\mathcal{L}_{\text{sep}}$. For each class c, draw J synthetic latent samples $Z_c^{(j)} \sim \mu_c = \mathcal{N}(m_c, S_c)$, evaluate $f_\theta(Z_c^{(j)})$, and apply cross-entropy with label c:

$$\mathcal{L}_{\text{sep}}^{\mathcal{B}} = \frac{1}{C} \sum_{c=1}^{C} \frac{1}{J} \sum_{j=1}^{J} \ell\big(c, f_\theta(Z_c^{(j)})\big).$$

Sampling uses the reparameterization $Z_c^{(j)} = m_c + L_c\, \xi^{(j)}$ with $\xi^{(j)} \sim \mathcal{N}(0, I)$.

(5) Total batch loss and updates.

$$\mathcal{L}_{\text{total}}^{\mathcal{B}} = \mathcal{L}_{\text{clf}}^{\mathcal{B}} + \lambda_{\text{fit}} \mathcal{L}_{\text{fit}}^{\mathcal{B}} + \lambda_{\text{sep}} \mathcal{L}_{\text{sep}}^{\mathcal{B}}.$$

Take a gradient step on all parameters $\{\phi_g\}$, θ, and $\{m_c, L_c\}$ using an optimizer (e.g. Adam).

4.4 Neural Network Choices (Digits / MNIST-like)

For digit experiments, we follow the paper's specification:

- **Encoder ϕ_g (image groups):** a CNN with *two convolutional layers*, followed by a *max-pooling* layer and a *sigmoid* activation; then *flatten* and a single fully-connected layer with *ReLU*; the latent dimension is fixed to $k = 64$.

- **Classifier f_θ:** a *linear* (softmax) layer on top of the latent vector (the paper also explores a small two-layer variant).
- **Optimization and batches:** Adam with learning rate 10^{-3}, batch size 100, and regularization strengths around 10^{-3}, consistent with the paper's settings.

4.5 Tuning Knobs

- λ_{fit}: strength of anchor fitting; larger values enforce tighter match between $(\widehat{m}_c, \widehat{S}_c)$ and (m_c, S_c).
- λ_{sep}: strength of separation via anchor classification. Increase if anchors overlap and synthetic samples are misclassified.
- J: number of synthetic samples per class per batch (e.g., $J \in \{8, 16, 32\}$).

5 Python Pseudocode (End-to-End, Illustrative)

Notes. This is a compact, readable PyTorch-style sketch that mirrors the algorithm above: (i) per-batch class moments, (ii) anchor-fitting via Gaussian W_2^2, (iii) inter-class separation via anchor-classification samples.

```
# --- Model pieces ------------------------------------------------------------
# encoders: dict mapping group id -> encoder module returning z in R^k
encoders = {g: Encoder_g_kdim() for g in range(G)}
classifier = SoftmaxHead(k, C)  # f_theta: R^k -> R^C

# Anchor parameters: for each class c, learn m_c in R^k and PSD S_c via L_c
m = nn.Parameter(torch.zeros(C, k))                # class means
L = nn.Parameter(torch.stack([torch.eye(k) for _ in range(C)]))  # k x k per class

# --- Utilities ---------------------------------------------------------------
def psd_sqrt(M, eps=1e-5):
    # eigenvalue-based PSD sqrt: V diag(sqrt(clamp(w))) V^T
    w, V = torch.linalg.eigh(M)
    w = torch.clamp(w, min=eps)
    return (V * w.sqrt()) @ V.T

def w2_gaussian_sq(m1, S1, m2, S2, eps=1e-5):
    term_mean = torch.sum((m1 - m2)**2)
    S1_sqrt = psd_sqrt(S1, eps)
    cross = psd_sqrt(S1_sqrt @ S2 @ S1_sqrt, eps)
    term_cov = torch.trace(S1 + S2 - 2.0 * cross)
    return term_mean + term_cov

def class_batch_moments(z, y, C, eps=1e-5):
    # z: B x k, y: B; returns lists [m_hat_c], [S_hat_c] for classes present
    mhat = [None]*C; Shat = [None]*C; present = [False]*C
    for c in range(C):
        idx = (y == c)
        if idx.any():
            zc = z[idx]                        # n_c x k
            zbar = zc.mean(dim=0)              # k
            zc_center = zc - zbar
            # unbiased covariance + jitter eps*I
            cov = (zc_center.T @ zc_center) / max(1, zc.shape[0]-1)
            cov = cov + eps * torch.eye(z.shape[1], device=z.device)
            mhat[c] = zbar
            Shat[c] = cov
            present[c] = True
    return mhat, Shat, present

def sample_from_anchor(m_c, L_c, J):
    # Reparam: Z = m_c + L_c @ xi, xi ~ N(0, I)
    xi = torch.randn(J, m_c.numel(), device=m_c.device)  # J x k
    return m_c[None, :] + xi @ L_c.T                     # J x k

# --- Loss compute per batch --------------------------------------------------
def losses_for_batch(batch, encoders, classifier, m, L, C, eps=1e-5, J=16):
    # batch: list of (x_i, y_i, g_i)
    x, y, g = batch['x'], batch['y'], batch['g']   # tensors
    # 1) encoders -> latent z
    z = torch.stack([encoders[int(gi)](xi) for xi, gi in zip(x, g)], dim=0)  # B x k

    # 2) classification loss
    logits = classifier(z)                         # B x C
    L_clf = F.cross_entropy(logits, y)

    # 3) per-class empirical moments (no EMA)
    mhat, Shat, present = class_batch_moments(z, y, C, eps)

    # 4) anchor-fitting loss
    L_fit = torch.zeros((), device=z.device)
    I = torch.eye(z.shape[1], device=z.device)
    for c in range(C):
        if present[c]:
            S_c = L[c] @ L[c].T + eps * I
            L_fit = L_fit + w2_gaussian_sq(mhat[c], Shat[c], m[c], S_c, eps)

    # 5) separation via anchor classification
    L_sep = torch.zeros((), device=z.device)
    for c in range(C):
        S_c = L[c] @ L[c].T + eps * I
        Zc = sample_from_anchor(m[c], L[c], J)     # J x k
        logits_c = classifier(Zc)                  # J x C
        target = torch.full((J,), c, device=z.device, dtype=torch.long)
        L_sep = L_sep + F.cross_entropy(logits_c, target)
    L_sep = L_sep / C

    return L_clf, L_fit, L_sep

# --- Training loop -----------------------------------------------------------
opt = torch.optim.Adam(
    list(classifier.parameters())
  + [p for gnet in encoders.values() for p in gnet.parameters()]
  + [m, L], lr=1e-3)

for epoch in range(T):
    for batch in loader:  # yields dict with 'x','y','g'
        L_clf, L_fit, L_sep = losses_for_batch(batch, encoders, classifier, m, L, C)
        loss = L_clf + lambda_fit * L_fit + lambda_sep * L_sep
        opt.zero_grad(); loss.backward(); opt.step()
```