5.1. Appendix

5.2. Proof of Theorem 1

We provide complete proof of the main theorem. We will first reiterate the notations used in the main paper.

Problem Setup. Let $x \in \mathbb{R}^d$ denote an ID data and the corresponding label y is generated by a *ground truth* linear model $\theta_* \in \mathbb{R}^d$, i.e., $y = \theta_*^T x$. To construct the training set, we sample n training data, where $n < d$, and stack the sampled data into a data matrix $\mathbf{X}_{tr} \in \mathbb{R}^{d\times n}$. Accordingly, the labels form a vector $\mathbf{Y}_{tr} = \mathbf{X}_{tr}^T\theta_* \in \mathbb{R}^n$. The *training* goal is to minimize the *empirical* loss.

$$\mathcal{L}(\mathbf{X}_{tr}, \mathbf{Y}_{tr}; \theta) = \|\mathbf{X}_{tr}^T\theta - \mathbf{Y}_{tr}\|_2 \tag{8}$$

Note that this forms an *over-parameterized* linear system, i.e., there are more parameters than equations, because $n < d$. This is similar to how modern neural networks are over-parameterized with respect to the data.

Complementary Decomposition using SVD. For the analysis, we make an independence assumption on the data matrix $\mathbf{X}_{tr}$. This assumption exists for notation simplicity and can be relaxed easily.

Assumption 2. *Let the n training data be linearly independent. The following SVD exists for the data matrix $\mathbf{X}_{tr}$.*

$$\mathbf{X}_{tr} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \quad \mathbf{U} \in \mathbb{R}^{d\times n}, \mathbf{\Sigma} \in \mathbb{R}^{n\times n}, \mathbf{V} \in \mathbb{R}^{n\times n}.$$

Consequently, we can decompose any vector $x \in \mathbb{R}^d$ into two components, $x = \mathbf{U}\tau + \mathbf{U}_\perp\tau_\perp$, where $\mathbf{U}$ is the basis for the span of training samples, $\mathbf{U}_\perp \in \mathbb{R}^{d\times(d-n)}$ is the basis for the complementary subspace, and $\tau \in \mathbb{R}^n$, $\tau_\perp \in \mathbb{R}^{d-n}$ are the corresponding coordinates. There are infinitely many solutions to Eq. 8 because this is an over-parameterized system. The classic result states that,

$$\theta = \mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^T\mathbf{Y}_{tr} + \mathbf{U}_\perp\beta_\perp, \tag{9}$$

where $\beta_\perp \in \mathbb{R}^{d-n}$ can be any vector. We denote a *projected model* as $\tilde{\theta} = \theta_0 + \alpha(\theta - \theta_0)$ (obtained using Eq. 3 or Eq. 4), where θ is one minimizer of Eq. 8, θ_0 is the pre-trained model and $0 \le \alpha \le 1$ is the projection ratio.

To quantify the effects of projection α, we can look at the average performance of the projected model $\tilde{\theta}$ on test data. Consequently, we investigate the *expected* loss of the projected model over the entire data space.

$$\mathbb{E}[\mathcal{L}(x, y; \tilde{\theta})] = \mathbb{E}\left[\left\|\tilde{\theta}^T x - y\right\|_2\right] \tag{10}$$

We now provide a detailed proof of Theorem 1 in the main paper. We first prove two lemmas.

Lemma 1. $\|(\theta - \theta_*)^T\mathbf{U}\tau\|_2 = 0.$

Proof. To show it, we use the decomposition in Eq. 9.

$$\begin{aligned}
\|(\theta - \theta_*)^T\mathbf{U}\tau\|_2 &= \|\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^T\mathbf{Y}_{tr} + \mathbf{U}_\perp\beta_\perp - \theta_*)^T\mathbf{U}\tau\|_2 \\
&= \|(\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^T\mathbf{Y}_{tr} - \theta_*)^T\mathbf{U}\tau\|_2 \\
&= \|(\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^T\mathbf{X}_{tr}^T\theta_* - \theta_*)^T\mathbf{U}\tau\|_2 \\
&= \|(\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T\theta_* - \theta_*)^T\mathbf{U}\tau\|_2 \\
&= 0
\end{aligned}$$

□

Lemma 2. $\|\mathbf{U}\tau\|_2 \le \|\tau\|_2$ *and* $\|\mathbf{U}_\perp\tau_\perp\|_2 \le \|\tau_\perp\|_2$.

Proof. We first invoke the definition of matrix norm,

$$\|\mathbf{U}\|_2 = \sup_{\tau\neq 0} \frac{\|\mathbf{U}\tau\|_2}{\|\tau\|_2}$$

From the definition, it is easy to see that

$$\|\mathbf{U}\tau\|_2 \le \|\mathbf{U}\|_2\|\tau\|_2.$$

Now recall that both $\mathbf{U} \in \mathbb{R}^{d\times n}$ and $\mathbf{U}_\perp \in \mathbb{R}^{d\times(d-n)}$ are orthonormal matrices. Therefore, using the property of L2 matrix norm,

$$\|\mathbf{U}\|_2 = \sqrt{\lambda_{max}(\mathbf{U}^T\mathbf{U})} = \sigma_{max}(\mathbf{U}) = 1$$

where $\lambda_{max}(\cdot)$ and $\sigma_{max}(\cdot)$ denote the largest eigenvalue and singular value respectively. Therefore,

$$\|\mathbf{U}\tau\|_2 \le \|\tau\|_2.$$

The same analysis extends to $\mathbf{U}_\perp, \tau_\perp$. □

Next, we proceed with the proof of the main theorem.

Proof.

$$\begin{aligned}
\mathcal{L}(x, y; \tilde{\theta}) &= \left\|\tilde{\theta}^T x - y\right\|_2 \\
&= \left\|(\theta_0 + \alpha(\theta - \theta_0))^T x - \theta_*^T x\right\|_2 \\
&= \|(\theta_0 + \alpha(\theta - \theta_0) - \theta_*)^T\mathbf{U}\tau + \\
&\quad (\theta_0 + \alpha(\theta - \theta_0) - \theta_*)^T\mathbf{U}_\perp\tau_\perp\|_2 \\
&\le \underbrace{\|((1-\alpha)(\theta_0 - \theta_*) + \alpha(\theta - \theta_*))^T\mathbf{U}\tau\|_2}_{A} + \\
&\quad \underbrace{\|(\theta_0 - \theta_*)^T\mathbf{U}_\perp\tau_\perp\|_2}_{B} + \underbrace{\|\alpha(\theta - \theta_0)^T\mathbf{U}_\perp\tau_\perp\|_2}_{C}
\end{aligned}$$

We use triangle inequality for the last inequality. We can now bound A using Lemma 1, Cauchy-Schwarz inequality and Lemma 2 as

$$\begin{aligned}\|((1-\alpha)(\theta_0-\theta_*)+\alpha(\theta-\theta_*))^T\mathbf{U}\tau\|_2 \\ =(1-\alpha)\|(\theta_0-\theta_*)^T\mathbf{U}\tau\|_2 \\ \leq(1-\alpha)\|(\theta_0-\theta_*)\|_2\|\mathbf{U}\tau\|_2 \\ \leq(1-\alpha)\|(\theta_0-\theta_*)\|_2\|\tau\|_2.\end{aligned}$$

Similarly, we can bound B using Cauchy-Schwarz inequality and Lemma 2 as

$$\begin{aligned}\|(\theta_0-\theta_*)^T\mathbf{U}_\perp\tau_\perp\|_2 &\leq \|\theta_0-\theta_*\|_2\|\mathbf{U}_\perp\tau_\perp\|_2 \\ &\leq \|\theta_0-\theta_*\|_2\|\tau_\perp\|_2,\end{aligned}$$

and bound C as,

$$\begin{aligned}\|\alpha(\theta-\theta_0)^T\mathbf{U}_\perp\tau_\perp\|_2 &\leq \|\alpha(\theta-\theta_0)\|_2\mathbf{U}_\perp\tau_\perp\|_2 \\ &\leq \|\alpha(\theta-\theta_0)\|_2\|\tau_\perp\|_2.\end{aligned}$$

Now, plug everything back. We arrive at the final result,

$$\mathcal{L}(x,y;\tilde{\theta})\leq(1-\alpha)\epsilon\|\tau\|_2+(\epsilon+\alpha\|\theta-\theta_0\|_2)\|\tau_\perp\|_2$$

where $\epsilon=\|(\theta_0-\theta_*)\|_2$. □

5.3. Group Based Total Variation Smoothing

Because of the iterative and incremental nature, the vanilla TPGM algorithm is a *greedy* algorithm, meaning that it judges the *immediate* benefit of the current updates to the model weights. If the current updates are not consistent with the validation data, they will be removed by projection, i.e., the projection radii will not increase to accommodate the new changes. Consequently, projection radii learned by TPGM could be overly *conservative* and lead to underfitting because gradient updates are stochastic, whose benefits may only show up in the long run. Empirically, we found TPGM results in under-fitting in some cases, i.e., slightly lower ID performance. To mitigate this side-effect of iterative optimization, we propose a group-based total variation (TV) smoothing for the projection parameters. TV is a common technique to improve smoothness in image denoising [1] and general signal processing [3]. We propose to utilize TV regularization to enforce a heuristic on the optimization of γ: *projection ratios of layers in the same group should be similar to each other*. Specifically, modern neural network architectures such as ResNet [14] and Transformer [37] are modular and stacked with groups (blocks). It is easy to identify unique groups in each architecture and assign layers to each one of them. Therefore, let $\mathcal{G}=\{g_i|i=0,...,M\}$ be the set of unique groups in a neural network. The loss function that we optimize for the projection parameters is updated as the following,

$$\mathcal{L}_\gamma=\mathcal{L}(x,y;\gamma_t)+\mu\sum_{g_i\in\mathcal{G}}\sum_{i\in g_i}|\alpha_i-\alpha_{i-1}| \tag{11}$$

where μ is a hyperparameter requiring tuning.

5.4. Implementation

In Alg. 2, the *projection update* function has its own optimizer. In our implementation, we use the Adam [20] optimizer because of its capability of adapting learning rate. Even though this introduces other hyperparameters, we find the same set of hyperparameters worked well for all experiments. Specifically, we use the default settings and a constant base learning rate of $\zeta=1e-2$.

ResNet experiments (Sec. 4.1). We list all the compared methods and their method-specific tuning to reproduce our results.

- **Vanilla Fine-Tuning (FT)**: We fine-tune all layers and sweep five learning rates (CLIP best $\eta_0=1e-3$, MOCO best $\eta_0=5e-2$).
- **Linear Probing (LP)**: We only fine-tune the head classifier and sweep five learning rates (CLIP best $\eta_0=1e-1$, MOCO best $\eta_0=1e-1$).
- **Partial Fusion (PF)** [19]: We fine-tune all the batch-norm layers and the head classifier, and sweep five learning rates (CLIP best $\eta_0=1e-2$, MOCO best $\eta_0=5e-2$).
- **L2-SP** [44]: We add L2-SP regularization, use the best-validated learning rate from FT, and sweep five three regularization hyperparameters (CLIP best μ : $1e-2$, MOCO best μ : $1e-3$).
- **MARS-SP** [9]: We add MARS projection (Eq 4), use the best-validated learning rate from FT, and sweep five three projection hyperparameters (CLIP best $\mu=64$, MOCO best $\mu=16$).
- **LP-FT** [21]: We first LP for 25 epochs, using the best LP learning rate, and FT for another 25 epochs, sweeping five learning rates (CLIP best $\eta_0=1e-3$, MOCO best $\eta_0=5e-2$).
- **TPGM**: We learn per-layer L2 projection radii incrementally, sweeping five learning rates (Eq. 3) (CLIP best $\eta_0=1e-2$, MOCO best w/o smoothing $\eta_0=1e-2$, MOCO w/ smoothing best :$\eta_0=4e-2$ and $\mu=0.1$).

Transformer Experiments (Sec. 4.2). We follow some common practices used in prior works [34, 35] to boost

fine-tuning performance. Note that we use the same training recipe for all methods unless otherwise specified. For example, linear probing performs worse when augmentations are used [28]. Now we will list the techniques used as well as their corresponding hyperparameters in parenthesis. Specifically, we use label-smoothing (0.1) [32], weight-decay (0.1), Mixup (0.8) [49] and Cutmix (1.0) [47]. We fine-tune models using the AdamW optimizer [24] for 30 epochs with a warm-up period of 5 epochs [34], per-step cosine decay schedule [12] and a batch size of 512. We list all the compared methods and their method-specific tuning to reproduce our results.

- **Vanilla Fine-Tuning (FT)**: We fine-tune all layers and sweep three learning rate $\eta_0 \in \{1e-5, 2e-5, 3e-5\}$.
- **Linear Probing (LP)**: We only fine-tune the head classifier and sweep three learning rates $\eta_0 \in \{5e-2, 1e-2, 5e-3\}$. We don't use any data augmentations (e.g., label-smoothing, Mixup and Cutmix) as they decrease LP performance.
- **BitFit** [48]: We fine-tune all the bias terms and the head classifier and sweep three learning rate $\eta_0 \in \{5e-2, 1e-2, 5e-3\}$.
- **L2-SP** [44]: We add L2-SP regularization, use the best-validated learning rate from FT, and sweep three three regularization hyperparameters $\mu \in \{1e-5, 1e-4, 1e-3\}$.
- **LP-FT** [21]: We first LP for 15 epochs, sweeping three learning rates $\eta_0 \in \{5e-2, 1e-2, 5e-3\}$, and FT the best-validated model for another 15 epochs, sweeping three learning rate $\eta_0 \in \{1e-5, 2e-5, 3e-5\}$.
- **Zero-Shot** [28]: We run an inference with the pre-trained CLIP model with the extracted zero-shot classifier.
- **WISE** [41]: We linearly interpolate the best validated FT model and the pre-trained model with a ratio of 0.5.
- **TPGM**: We learn per-layer projection radii between the best validated FT model and the pre-trained model using the MARS projection (Eq. 4).

5.5. CLIP Pre-trained ViT-L on ImageNet

In Sec. 4.2, we presented fine-tuning results on ImageNet using CLIP pre-trained ViT-b. In this section, we conduct the same experiments with CLIP pre-trained ViT-L. As we noticed in the ViT-b experiments, WISE and TPGM perform much better than other competitors, so we focus on the comparison between the two here. We first present tabulated results in Tab. 5. We observe that TPGM improves

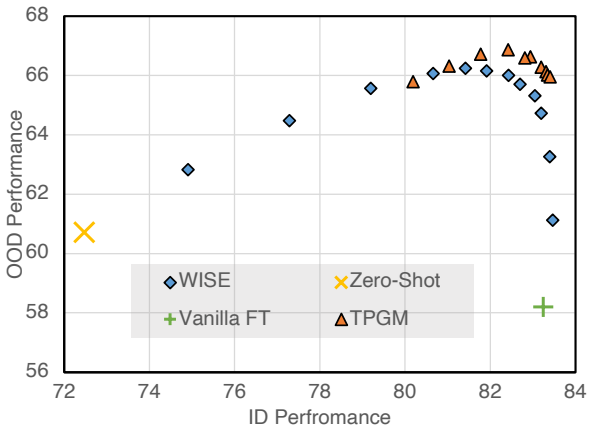

Figure 5. **ID and OOD performance of TPGM and WISE with different hyperparameters using CLIP pre-trained ViT-L, fine-tuned on ImageNet.** Sweeping different hyperparameters for both WISE and TPGM shows that learning per-layer constraints is superior to learning a single constraint.

both ID and OOD performance over vanilla FT. To compare fairly with WISE, we introduced TPGM-C (Sec. 4.2), which uses an L2 regularization on the learned projection radii to control the distance to the pre-trained model. With proper regularization, TPGM-C outperforms WISE on both ID and OOD performance. We also provide a figure of ID vs. OOD performance with different WISE interpolation ratios and different TPGM-C regularization strengths in Fig. 5. We observe the same trend as in the ViT-b experiments (Sec. 4.2): at the same ID performance, TPGM has better OOD performance.

5.6. Comparisons between TPGM-L2 and TPGM-MARS

In the main paper, we presented two possible projections: L2 projection (Eq. 3) and MARS projection (Eq. 4). Both projections provide closed-form solutions. We can use either of them in TPGM. In this section, we present comparisons between the two.

ResNet Experiments on DomainNet. For ResNet experiment in Sec. 4.1, we use a CLIP pre-trained ResNet50 and an ImageNet pre-trained ResNet50. For TPGM, we use $f_{proj} = 1$ and $T_{proj} = 1$. In Tab. 6, we compare the performance of TPGM using MARS and L2 projections on DomainNet-Real with 100% of its data. We observe that in this setting MARS performs better than L2 projection.

Transformer Experiments on ImageNet. For Transformer experiments in Sec. 4.2, we use a CLIP pre-trained ViT-B. For TPGM, we use $f_{proj} = T - 1$ and $T_{proj} = 200$. Following the main paper, we add L2 regularization to the projection radii and sweep a range of values from $4e-3$ to $1e-4$. In Fig. 6, we compare the performance of TPGM

Table 5. **ImageNet Results using CLIP pre-trained ViT-L.** TPGM improves OOD performance significantly without losing ID performance. TPGM-C achieves the best OOD performance while maintaining a more competitive ID performance compared to the current state-of-the-art method WISE. TPGM-C is a controlled variant of TPGM, designed to lower its ID performance to the same level as WISE for a fair comparison of OOD performance.

	ID		OOD			Statistics			
	ImageNet	ImageNet-V2	ImageNet-A	ImageNet-R	ImageNet-S	ID Avg.	OOD Avg.	ID Δ (%)	OOD Δ (%)
Vanilla FT	**87.24**	79.25	49.67	63.29	61.62	83.25	58.19	0.00	0.00
Zero-Shot [28]	75.00	69.95	52.21	71.69	58.24	72.48	60.71	-12.94	4.33
WISE [41]	85.33	78.50	58.26	75.37	64.84	81.92	66.16	-1.60	13.68
TPGM-C	86.02	78.83	**59.29**	**76.32**	65.00	82.43	**66.87**	-0.99	**14.91**
TPGM	87.00	**79.81**	58.31	74.41	**65.13**	**83.41**	65.95	**0.19**	13.33

Table 6. **Comparison between MARS and L2 projections on DomainNet using ResNet50**.

		ID	OOD				Statistics
		Real	Sketch	Painting	Infograph	Clipart	OOD Avg.
CLIP	MARS	**83.64**	**38.78**	43.11	**28.70**	**48.01**	**39.65**
	L2	82.72	37.18	**43.33**	25.99	45.71	38.05
MOCO	MARS	81.66	**35.97**	**46.68**	**20.34**	**46.11**	**37.27**
	L2	81.66	33.96	45.82	18.71	44.45	35.74

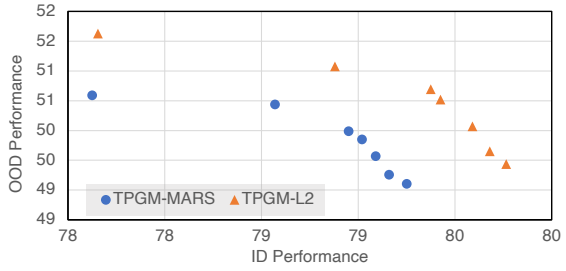

Figure 6. **Comparison between MARS and L2 projections on ImageNet using ViT-B.**

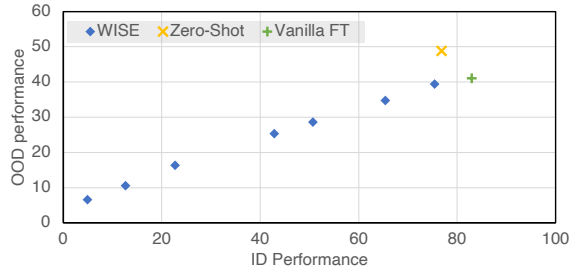

Figure 7. **WISE interpolation ratio sweeping using CLIP pre-trained ResNet50 on DomainNet.**

using MARS and L2 projections on ImageNet ID and OOD datasets. We observe that L2 projection always outperforms MARS projection in this setting.

5.7. WISE for CLIP Pre-trained ResNet

The prior work [41] and our experiments in Sec. 4.2 verified that CLIP pre-trained Transformers have very good linear connectivity. This means that when linearly interpolating between the pre-trained model and a fine-tuned model, the output does not degrade much. In this case, we observe significantly improved OOD generalization with minimal ID performance loss. However, the same trend is not observed when switching the architrave to ResNet50. Similar to the prior work [41], we extract a zero-shot classifier for DomainNet classes using a CLIP pre-trained ResNet50 and conduct the same linear interpolation ratio sweeping as in the main paper. In Fig. 7, we plot ID performance against the OOD performance of WISE with different ratios, the pre-trained (zero-shot) model, and the vanilla fine-tuned model. Notably, we observe a significant drop in performance when interpolating between the pre-trained model and a fine-tuned model. This shows that CLIP pre-trained ResNet does not enjoy the same linear connectivity as its Transformer counterpart.

Table 7. **DomainNet Results using MOCO-V3 pre-trained ResNet50 Results with 100% Real Data.** TPGM without TV smoothing achieves the best OOD performance but with slightly worse ID performance compared to vanilla FT. TV smoothing can effectively mitigate this negative effect.

	ID	OOD				Statistics
	Real	Sketch	Painting	Infograph	Clipart	OOD Avg
Vanilla FT	81.99	31.52	42.89	18.51	44.98	34.47
TPGM w/o TV	81.66	**35.97**	**46.68**	**20.34**	46.11	**37.27**
TPGM w/ TV	**82.66**	35.35	46.20	20.13	45.75	36.86

5.8. Smoothing Comparison using MOCO-V3

In the main paper, we found that training with MOCO-V3 pre-trained ResNet50 on DomainNet can benefit from total variation (TV) smoothing (Appendix 5.3). Here we show a detailed comparison between TPGM with and without smoothing for this particular setting in Tab. 7. We observe that TPGM without smoothing achieves the best OOD performance however with a slight decrease in ID performance compared to vanilla FT. This might be caused by the conservative nature of TPGM as discussed in Appendix 5.3. When TV smoothing is added, we observe that TPGM brings improvement to both ID and OOD perfor-

mance over vanilla FT.

5.9. Computation Overhead

TPGM inevitably adds some computation overhead to a vanilla fine-tuning pipeline (though not inference). The majority of computation cost comes in Alg. 2, where the algorithm needs to conduct gradient updates on the projection parameters. While this overhead is negligible when we set $f_{freq} = T - 1$, i.e., *projection update* is only called once at the end of training as in the Transformer experiments (Sec. 4.2), the overhead increases when $f_{freq} = 1$. In our ResNet experiments (Sec. 4.1), to decrease computation cost, we only the update projection parameters once during each call, i.e, $T_{proj} = 1$. Qualitatively, we see an increase of training time from $\sim$ 29 hours to $\sim$ 34 hours when TPGM is added, a $\sim 17\%$ increase. However, this increase can be justified by the fact that manually searching for per-layer constraints can be intractable.