Learning a Neural Pareto Manifold Extractor with Constraints (Supplementary material)

Soumyajit Gupta[1] **Gurpreet Singh**[2] **Raghu Bollapragada**[3] **Matthew Lease**[4]

[1]Department of Computer Science, University of Texas at Austin, USA
[2]XtractorAI
[3]Operations Research and Industrial Engineering, University of Texas at Austin, USA
[4]School of Information, University of Texas at Austin, USA

1 SUHNPF AS HYPERNETWORK

Fig. 1 shows the overview of SUHNPF as a hypernetwork tasked with optimizing the weights of the target neural classifier. The input to the neural classifier are data points Y and the output are matched against labels $Z_{1,true}, Z_{2,true}$ for two different tasks. The weights of the neural classifier are Θ and SUHNPF as a hypernetwork approximates the weak Pareto manifold $\tilde{M}(\theta^*)$ for optimal trade-off over different values α for the two MOO losses $\mathcal{L}_1, \mathcal{L}_2$.

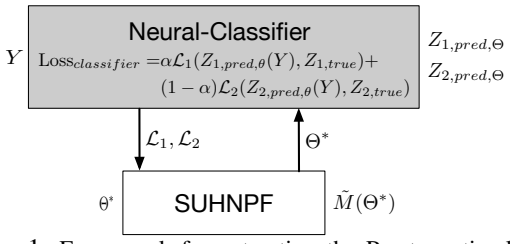


Figure 1: Framework for extracting the Pareto optimal front $\tilde{M}(\Theta)$ of a given target model C_Θ (which could also be a non-neural model: Decision Tree, Logistic Regression, *etc*.)

2 SUHNPF *VS.* OTHER MTL METHODS

Point based solvers. Most MTL methods, including MOOMTL, PMTL, EPSE, and EPO are point based solvers. Being point-based, they return one solution per run, relying upon specialized local initialization to generate an even spread of Pareto points, using cones, rays, or other domain partitioning strategies, across the feasible set of saddle points. Thus asked for P Pareto candidates, these solvers would have to run for P instances. Later, if the user demands $2P$ points, they have to run for $2P$ instances from scratch, without utilizing the results from the previous run.

Manifold based solvers. A manifold-based solution strategy should separate Pareto *vs.* non-Pareto points, without requiring any special initialization. It would also be able to extract $2P$ Pareto candidates while being trained to generate P candidates, due to interpolating from the learned boundary. This is highly advantageous over point-based schemes for deployment of practical systems, where the expected user trade-off preference is not known *a priori*, hence good to have the full approximated front. Notably, Navon et al. [2021] and Lin et al. [2020] are the only prior manifold based Pareto solvers that we are aware of that are also scalable to optimize large neural models. Another advantage is that both SUHNPF and EPO (used by PHN in backend) solvers have a user-specified error tolerance criteria built in, while other MTL solver lack it and therefore run a specified number of iterations before declaring a candidate Pareto, without actually checking for optimality.

Full rank indicator *vs.* low rank regressor. A manifold based solver should also generalize to cases where the manifold is an implicit function as opposed to its easier counterpart of being an explicit function. SUHNPF has an added advantage in extracting the weak Pareto manifold as an k-dimensional diffusive indicator function as opposed to a $(k-1)$-dimensional manifold itself, where the regressed manifold is not only guided by the weak Pareto points (indicator value 1) but also the sub-optimal points (indicator value 0) for a more robust and accurate extraction. Thus it can generally approximate the manifold, irrespective of the manifold being an explicit or implicit function. In comparison, PHN learns a $(k-1)$-dimensional regression manifold, given solution points obtained from EPO or LS. Therefore, PHN's default assumption is that the Pareto manifold is always an explicit function *i.e.,* for k objectives, the Pareto manifold is of dimension $k-1$.

3 DISCUSSION ON REMARK 1

Remark: *If f_is are continuous and differentiable once, in an unconstrained setting, then the set of weak pareto optimal points are $x^* = \{x|det(L(x)^T L(x)) = 0\}$, for a non-square matrix $L(x)$, and is equivalent to $x^* = \{x|det(L(x)) = 0\}$ for a square matrix $L(x)$.*

We begin by considering an unconstrained MOO for ease of description. Let us consider the following two quadratic functions, convex in both variables as:

$$f_1(\mathbf{x}) = (x_1 - 1)^2 + (x_2 - 1)^2$$
$$f_2(\mathbf{x}) = (x_1 + 1)^2 + (x_2 + 1)^2$$

Accepted for the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022).

The task is to find the Pareto front between the objectives $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$. For this problem the Pareto front is known *a priori* as the straight line $x_1 = x_2$ for $x_1 \in [-1, 1]$ in the variable domain. Let us now first plot f_1 *vs.* f_2 for visual assessment in **Fig. 2**.

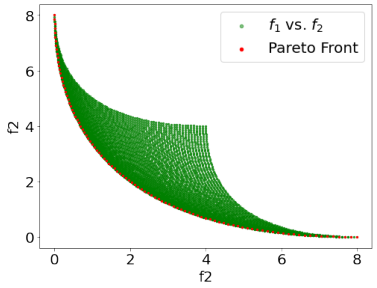


Figure 2: Functional Domain plot for two competing objectives.

Note that independent of each other:

$$\nabla f_1(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \end{bmatrix}^T = \mathbf{0} \text{ at } (x_1, x_2) = (1, 1)$$

$$\nabla f_2(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}^T = \mathbf{0} \text{ at } (x_1, x_2) = (-1, -1)$$

One can easily confirm that the gradient matrix L cannot be identically zero for any value of $x \in \mathbb{R}^2$.

$$L = [\nabla f_1(x) \quad \nabla f_2(x)]$$

To avoid a trivial solution the vector $[\alpha_1 \quad \alpha_2]$ must also not be identically zero. This becomes clear if the scalarized function $S(\mathbf{x}) = \alpha_1 f_1 + \alpha_2 f_2$ is defined, where $\alpha_1 + \alpha_2 = 1$.

The only remaining possibility is $L\alpha$ should approach zero for some $\mathbf{x} = [x_1 \quad x_2]$ as we iteratively update x. This gives us our termination/convergence criterion.

$$\begin{aligned} L\alpha &= [\nabla f_1(x) \quad \nabla f_2(x)]_2[\alpha_1 \quad \alpha_2]^T \\ &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} [\alpha_1 \quad \alpha_2]^T = [0 \quad 0]^T \end{aligned}$$

Let us assume any point (x_1, x_2) in the feasible domain. What α values can achieve the above termination criterion? We now have two equations in two unknowns ($\alpha_i s$):

$$\alpha_1 \frac{\partial f_1}{\partial x_1} + \alpha_2 \frac{\partial f_2}{\partial x_1} = 0$$
$$\alpha_1 \frac{\partial f_1}{\partial x_2} + \alpha_2 \frac{\partial f_2}{\partial x_2} = 0$$

Eliminating α_1 using the first equation and substituting in the second equation:

$$\left[-\left(\frac{\partial f_1}{\partial x_2} \frac{\partial f_2}{\partial x_1} \right) / \left(\frac{\partial f_1}{\partial x_1} \right) + \frac{\partial f_2}{\partial x_2} \right] \alpha_2 = 0$$

For any $\alpha_2 > 0$, this implies:

$$\left[-\left(\frac{\partial f_1}{\partial x_2} \frac{\partial f_2}{\partial x_1} \right) / \left(\frac{\partial f_1}{\partial x_1} \right) + \frac{\partial f_2}{\partial x_2} \right] = 0$$
$$\Rightarrow \left[\frac{\partial f_1}{\partial x_1} \frac{\partial f_2}{\partial x_2} - \frac{\partial f_1}{\partial x_2} \frac{\partial f_2}{\partial x_1} \right] = 0$$

Alternatively,

$$det\left(\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \right) = det(L) = 0$$

Note that for any matrix $A \neq \mathbf{0}$, $Ax = 0$ can be solved for a non-trivial $x \neq 0$ if and only if A has a null-space; or A is low rank; or if A is square then it's determinant is zero. □

The $det(L)$ matrix defined in **Eq. 6 (main)** is given by:

$$L = \begin{bmatrix} \nabla F & \nabla G \\ \mathbf{0} & G \end{bmatrix}$$

To achieve $det(L) = 0$ requires that either:

1. $\nabla F(x) = 0$: atleast one objective function has reached its optimum (local/global minima/maxima under a min/max setting); *and / or*
2. $G(x) = 0$: at least one constraint is satisfied.

This criteria is only applicable for square systems. However, for practical problems, the system might become non-square, hence we need to satisfy $det(L^T L) = 0$ following **Eq. 7 (main)**. One might think that it's a different optimization problem. However satisfying $det(L^T L) = 0$ mathematically provides the same justification and we provide the derivation of it.

$$\begin{aligned} det(L^T L) &= \begin{bmatrix} \nabla F^T & \mathbf{0} \\ \nabla G^T & G^T \end{bmatrix} \begin{bmatrix} \nabla F & \nabla G \\ \mathbf{0} & G \end{bmatrix} \\ &= \begin{bmatrix} \nabla F^T \nabla F & \nabla F^T \nabla G \\ \nabla G^T \nabla F & \nabla G^T \nabla G + G^T G \end{bmatrix} \end{aligned} \tag{1}$$

We now observe Eq. 1 for the two cases prescribed above and see if $det(L^T L)$ evaluates to zero or not. For Case 1, where $\nabla F = 0$, Eq. 1 reduces to:

$$det(L^T L) = \begin{bmatrix} \mathbf{0} & \mathbf{0} \nabla G \\ \nabla G^T \mathbf{0} & \nabla G^T \nabla G + G^T G \end{bmatrix}$$

which is low-rank since row 1 equates to 0. For Case 2, where $G = 0$, Eq. 1 reduces to:

$$\begin{aligned} det(L^T L) &= \begin{bmatrix} \nabla F^T \nabla F & \nabla F^T \nabla G \\ \nabla G^T \nabla F & \nabla G^T \nabla G + 0 \end{bmatrix} \\ &= \nabla F^T \nabla G^T \begin{bmatrix} \nabla F & \nabla G \\ \nabla F & \nabla G \end{bmatrix} \end{aligned}$$

which is low-rank again because row 1 and row 2 are equal. Hence it is easy to observe that satisfying $det(L) = 0$ is equivalent to satisfying $det(L^T L) = 0$. □

4 EXPERIMENTAL SETUP DETAILS

Experimental Setup. We use an Nvidia 2060 RTX Super 8GB GPU, Intel Core i7-9700F 3.0GHz 8-core CPU and 16GB DDR4 memory for all experiments. Keras [Chollet, 2015] is used on a Tensorflow 2.0 backend with Python 3.7 to train the SUHNPF networks and evaluate the MTL solvers. For optimization, we use AdaMax [Kingma and Ba, 2015] with parameters (lr=0.001).

SUHNPF Setup. Each training step runs for 2 epochs, with 50 steps per epoch. Thus, if the network takes I iterations to converge,

then the effective epochs taken by the network is $2I$. For computing the gradient of the Fritz-John matrix *w.r.t.* the input variables x, we use Tensorflow's `GradientTape`[1], which implicitly allows us to scale the computation of the gradient matrix ∇det to arbitrarily large dimensions of variable x. To compute the gradient update on $\mathcal{P}1$, we use a learning rate of $\eta = 0.01$.

MTL Setup. Sourcecode for LS, MOOMTL, PMTL and EPO solvers use EPO's repository[2], while EPSE[3] and PHN[4] codes are taken from their individual repositories.

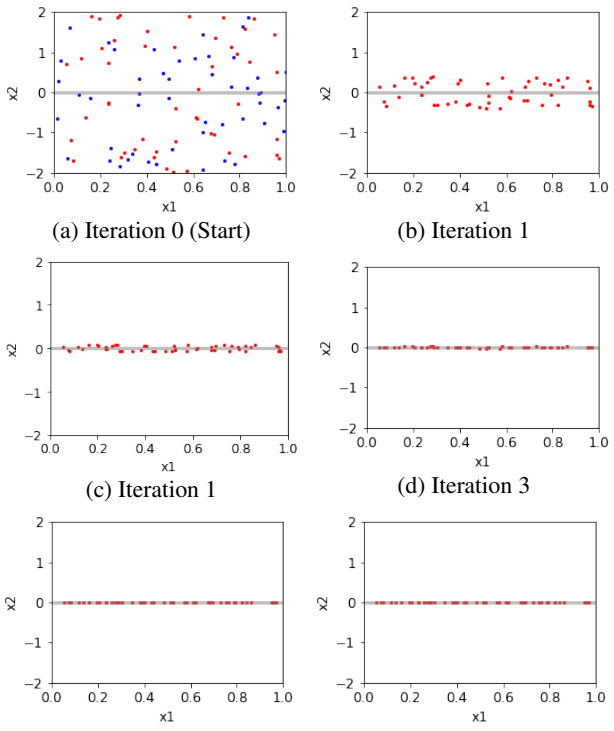
5 GENERAL DISCUSSION

Handling Non-Convex forms: Pareto optimal solution set is a collection of saddle points [Van Rooyen et al., 1994, Ehrgott and Wiecek, 2005] of an MOO problem, wherein no objective can be further improved without penalizing at least one of the other objectives. This entails min-max optimization to minimize objectives (such as loss functions) while simultaneously maximizing trade-offs between them. Although prior works [Sener and Koltun, 2018, Lin et al., 2019, Mahapatra and Rajan, 2020] have asserted that Karush-Kuhn-Tucker (KKT) conditions [Boyd et al., 2004] in this min-max setting ensure that MTL methods find (correct) Pareto optimal solutions, it is known that KKT conditions hold true only for convex cases. Gobbi et al. [2015] further show that KKT-based criteria can give Pareto solutions only under fully convex setting of objectives and constraints.

Evaluation on Benchmarks. Because the Pareto solution is often unknown on real MOO problems, OR works have advocated that any proposed Pareto solver should first be tested on synthetic MOO with known analytic solutions. This permits controlled experimentation that vary MOO problem difficulty (*e.g.,* non-convexity in variable and function domains, presence of constraints, *etc.*) in order to assess the capabilities and measure the true accuracy against a known front. Ideally studies should evaluate against synthetic benchmark problems that vary in difficulty, and there is sometimes ambiguity and confusion in referring to an MOO problem as non-convex without clarifying the specific non-convex aspects. Difficulty can also vary greatly depending on whether non-convexity occurs in the objectives, constraints, or the front itself.

Termination of Solvers. An iterative solver should define termination criteria based on an error tolerance being satisfied and/or inability to further improve. It is also important that a solver reports inability to converge (achieve the termination criteria/error tolerance) within the specified maximum iterations. While both HNPF (used by SUHNPF) and EPO (used by PHN) define such error tolerance criteria for termination, inspection of source code for MOOMTL [Sener and Koltun, 2018], PMTL [Lin et al., 2019], and EPSE [Ma et al., 2020] iterative solvers (at the time of our submission) shows support only for running a fixed number of iterations, without other termination criteria. See the following sourcecode links to solvers for MOOMTL[5], PMTL[6], and EPSE[7].

6 CONVERGENCE PLOTS OF SUHNPF



(a) Iteration 0 (Start) (b) Iteration 1

(c) Iteration 1 (d) Iteration 3

(e) Iteration 4 (f) Iteration 5 (Converged)

Figure 3: Case I: Variable domain. The gray line show the true analytic solution ($0 \leq x_1 \leq 1, x_2 = 0$). SUHNPF Pareto candidates $\mathcal{P}1$ (red dots) converge in 5 iterations.

Convergence of Pareto candidates towards the weak Pareto front over iterations for the variable (**Fig. 3**) and functional (**Fig. 4**) domains are shown for benchmark Case I considered in our work.

7 ADDITIONAL BENCHMARKS

We consider two additional synthetic benchmark cases considered by Navon et al. [2021]. We demonstrate that SUHNPF works well in these cases since the considered functions are either convex or monotone within the feasible domain for both cases.

Case A:

$$f_1(x_1, x_2) = ((x_1 - 1)x_2^2 + 1)/3,\ f_2(x_1, x_2) = x_2$$
$$\text{s.t. } g_1, g_2 : 0 \leq x_1, x_2 \leq 1 \tag{2}$$

Case B:

$$f_1(x_1, x_2) = x_1,\ f_2(x_1, x_2) = 1 - (x_1/(1 + 9x_2))^2$$
$$\text{s.t. } g_1, g_2 : 0 \leq x_1, x_2 \leq 1 \tag{3}$$

Please note that although in PHN [Navon et al., 2021], the form of $f_2 = x_1$ for Eq. 2, we believe it is a typo *w.r.t.* the original

[1] https://www.tensorflow.org/api_docs/python/tf/GradientTape

[2] https://github.com/dbmptr/EPOSearch

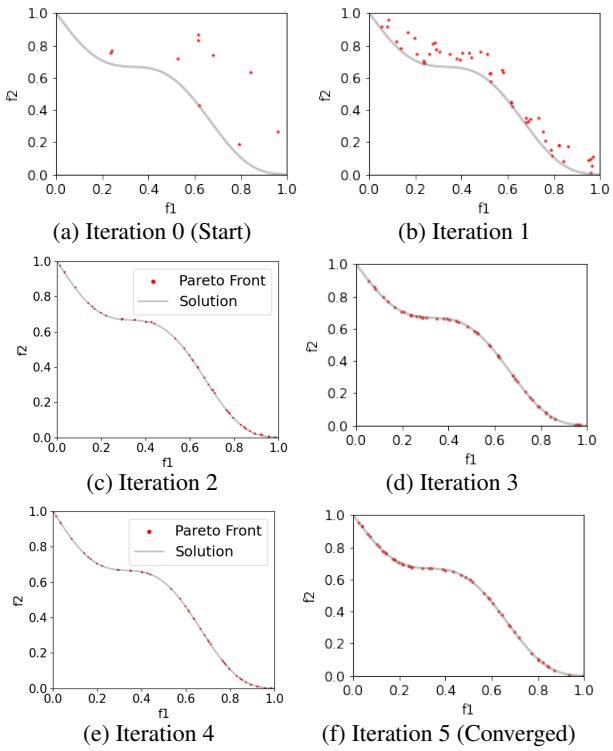
[3] https://github.com/mit-gfx/ContinuousParetoMTL

[4] https://github.com/AvivNavon/pareto-hypernetworks

[5] https://github.com/dbmptr/EPOSearch/blob/master/toy_experiments/solvers/moo_mtl.py

[6] https://github.com/dbmptr/EPOSearch/blob/master/toy_experiments/solvers/pmtl.py

[7] https://github.com/mit-gfx/ContinuousParetoMTL/blob/master/pareto/optim/hvp_solver.py



(a) Iteration 0 (Start) (b) Iteration 1

(c) Iteration 2 (d) Iteration 3

(e) Iteration 4 (f) Iteration 5 (Converged)

Figure 4: Case I: Functional domain mapping to **Fig. 1 (main)**. SUHNPF Pareto candidates $\mathcal{P}1$ (red dots) converge in 5 iterations.

work by Evtushenko and Posypkin [2013], where this case was proposed, as the reported Pareto front in their work is achieved only for $f_2 = x_2$. We therefore proceed with this updated form.

8 LOSS PROFILES

Fig. 5 shows the loss profiles for the benchmark cases I - III. SUHNPF converged in 5 iterations, with each iteration running for 2 epochs, using error tolerance 10^{-4} for both the outer gradient descent loop ϵ_{outer} and inner gradient descent loop ϵ_{inner}. Since the last layer of the SUHNPF network classifies points as being *weak* Pareto or not, the loss enforced is Binary Cross Entropy (blue line). We also report the Mean Squared Error (MSE, dashed red line) between the current iterate of point set $\mathcal{P}1$ and the true analytical solution manifold. **Alg. 1 (main)** updates the Pareto candidate set in the outer descent loop. Since the inner descent loop that measures the training loss itself $\mathcal{P}1$ has ran twice for 2 epochs, MSE is be measured only once per iteration. This results in the staircase nature of the MSE loss.

9 RUNTIME COMPLEXITY

Assume the following notation from main text for clarity i.e. k: num objectives; m: num constraints; n: num variables; $\mathcal{P}$: num Pareto candidates; $\mathcal{I}$: num iterations till convergence. In **Algorithm 1 (main)**, the outer while loop (lines 4-9) takes $\mathcal{I}$ iteration till convergence. Note that for any gradient descent based solver, the number of iterations $\mathcal{I}$ is not known *a priori*. Per iteration, step 5 involves training the neural network with $\mathcal{P}$ points using the FJC and error computation in step 6 jointly costing $\mathcal{O}(\mathcal{P}(k+m)^2 n)$. Step 7 can be effectively broken down into two serial sub-computations: a) Computing the gradient of the determinant takes cost $\mathcal{O}((k+m)^2 n)$ and b) calculating the gradient of the determinant as $\mathcal{O}(\mathcal{P}k^3)$. Step 8 takes $\mathcal{O}(\mathcal{P}(k+m))$ due to the update of $\mathcal{P}$ points using a $(k+m)$-dimensional vector. Hence the total compute cost per iteration is $\mathcal{O}(\mathcal{P}(k+m)^2 n + (k+m)^2 n + \mathcal{P}(k+m)^3 + \mathcal{P}(k+m))$, which can be simplified to $\mathcal{O}(\mathcal{P}(k+m)^2 n + \mathcal{P}(k+m)^3)$. Under a practical deep MTL, $n \gg k, m$ (*i.e.,* variable dimension is strictly greater than the number of functions and constraints in any neural setting), the complexity is dominated by the term $\mathcal{O}(\mathcal{P}(k+m)^2 n)$, where the scaling is linear in terms of the variable dimension n, and quadratic in the number of functions and constraints k, m. Thus, the overall compute cost for $\mathcal{I}$ iterations is $\mathcal{I} \times \mathcal{O}(\mathcal{P}(k+m)^2 n)$. Note that this is similar to the runtime complexity reported in EPO [Mahapatra and Rajan, 2020], where their **point based solver** takes $\mathcal{O}(k^2 n)$ for each point for each iteration, leading to a total compute cost of $\mathcal{I} \times \mathcal{O}(\mathcal{P}k^2 n)$ for $\mathcal{P}$ points and $\mathcal{I}$ iterations, under an unconstrained optimization settings. In SUHNPF, since we also support constrained optimization, we have replaced 'k' with '$k+m$'.

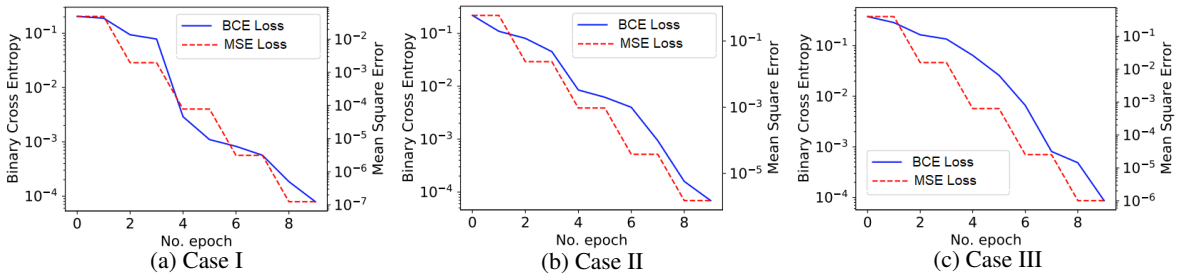
We report both the asymptotic and actual runtimes for EPO and SUHNPF in **Table 1**. Note that for Case I, although the complexity of SUHNPF is twice that of EPO ($\mathcal{O}(400)$ *vs.* $\mathcal{O}(800)$) *i.e.,* both are asymptotically similar, in actual runtime SUHNPF is much faster than EPO (10s *vs.* 752s). SUHNPF moves candidates towards being Pareto optimal through the use of the FJC guided discriminator in **Algorithm 1 (main)**, while EPO has to solve two separate primal and dual problems to find Pareto optimal points. This imparts SUHNPF lower runtime per iteration than EPO, hence it converges faster too. Furthermore, SUHNPF constructs the approximate Pareto manifold in addition to finding 50 candidates on it. To achieve similar functionality based on EPO would require first computing 50 points via 50 runs of EPO, then running a neural network to regress over those 50 points. This is what PHN [Navon et al., 2021] in its PHN-EPO configuration. Hence in **Table 4 (main)**, PHN has higher runtime than EPO.

Table 1: Per-iteration time complexity and full runtime of Cases I-III for EPO *vs.* SUHNPF. We do not report complexity for EPO on Case II, because their solver reported NaNs, or on Case III, because EPO does not support constrained optimization. As shown above, the asymptotic complexity of EPO is given by $\mathcal{O}(\mathcal{P}k^2 n)$, whereas that of SUHNPF is given by $\mathcal{O}(\mathcal{P}(k+m)^2 n + \mathcal{P}(k+m)^3)$.

Cases	Variables				EPO		SUHNPF	
	k	m	n	$\mathcal{P}$	Complexity	Runtime (sec)	Complexity	Runtime (sec)
I	2	0	2	50	$\mathcal{O}(400)$	752	$\mathcal{O}(800)$	10
II	2	0	30	50	$\mathcal{O}(6000)$	-	$\mathcal{O}(6400)$	20
III	2	2	2	50	-	-	$\mathcal{O}(4800)$	10

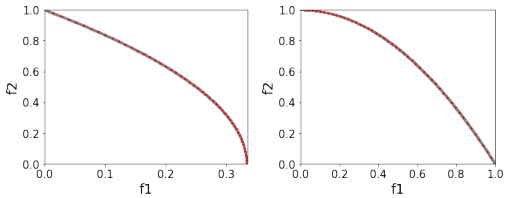
While correctness and point density in finding the true Pareto optimal solution should be our top priority in comparing methods, we also report run-time of SUHNPF *vs.* other MTL approaches on the studied cases. We explicitly request that each method generate 50 Pareto candidates, within the feasible functional domain. **Table 4 (main)** reports the percentage of such candidates obtained, and the overall execution time, averaged over 10 runs each, given our experimental setup in Appendix 4.

PHN uses either EPO or LS as their base solver, hence we report the total time that includes the (a) run-time of the base solver; and (b) the neural network run-time to learn the regression manifold. Cases I and III have a 2D variable domain, where SUHNPF takes



(a) Case I (b) Case II (c) Case III

Figure 5: Loss profile for cases. Note that since the error threshold ϵ was set to 10^{-4} for the benchmark cases, the algorithm terminates once the Binary Cross Entropy (blue) loss falls below the threshold (value at epoch $10 \leq 10^{-4}$). We also show the Mean Squared Error (dashed red) between the Pareto candidate set $\mathcal{P}1$ and the analytical solution for each iteration. Because each iteration takes two epochs, this leads to the "staircase" MSE shown.



(a) Case A (b) Case B

Figure 6: Functional Domain of cases from PHN

1s per epoch, with 2 epochs for training in Step 7 of **Alg. 1 (main)**. Both the cases took 5 epochs to converge, resulting in a total run-time of 10s. Case II has a 30D variable domain where SUHNPF takes 2s per epoch resulting in a total run-time of 20s. While LS and MOOMTL are at similar run-time scale with SUHNPF, they fail to generate an even spread of points (**Fig. 3 (main) (a,b)**).

10 SPACE COMPLEXITY ANALYSIS

MTL methods solve problems in both primal and dual space *i.e.,* gradient of the objectives in the primal and the trade-off α's in the dual. SUHNPF however works only in the primal space *w.r.t.* the gradient of the functions necessary in the construction of the Fritz-John matrix, since the FJC ensure α free stationary point identification. Thus, the additional dual optimization space is not required. To fairly compare *w.r.t.* MTL methods, we consider the general cost of both such systems under a unconstrained setting *i.e.,* only objectives and no additional constraints. Thus k, n indicate the number of objectives and the dimension of the variable space.

SUNHPF. To find P Pareto candidates, SUHNPF updates P points of size Pn. The $\nabla F^T \nabla F$ and ∇det matrices are of size k^2 and nk respectively. The total memory cost is thus of order $O(n(P+k)+k^2))$.

MTL. To find P candidates, MTL methods uses P cones or rays requiring size Pn. The gradient matrix of the objective function ∇F takes nk, constructing the simplex takes k^2, solving for trade-off α takes k^2 and the iterative update requires additional nk memory. The total memory cost is of order $O(n(P+2k)+2k^2))$.

11 UNIFORMITY AND COVERAGE

SUHNPF starts with a random set of $2\mathcal{P}$ candidates. $\mathcal{P}$ of these are then altered through our FJC guided **Algorithm 1 (main)** towards being Pareto candidates within the tolerance bound. Since all the starting points are generated uniformly at random within the feasible set, they tend to have even initial spread. When the FJC guided descent is run, each of these points are then directed towards their nearest Pareto front via gradient descent. Empirically, we observe this suffices to achieve good spread of the $\mathcal{P}$ training points on the Pareto front, without any explicit optimization target to promote or enforce even spread.

MTL methods partition space into equal sized cones or preference rays, which leads to semi-uniformity *iff* the obtained candidates lie within the specified cone or in the vicinity of the preference ray. EPO [Mahapatra and Rajan, 2020] is the only method which has this explicitly loss criteria to ensure even spread / uniformity of points in their algorithm. However, note that these cone and ray approaches implicitly assume a symmetric and uniform nature of the front, which is not known *a priori* and rarely seen in practice.

To measure spread, we distinguish two concepts:

uniformity evenness of spread of candidates along the front

coverage the extent of the front spanned by extreme Pareto points

For *uniformity*, we report the average and maximum euclidean distance between two neighboring Pareto candidates. Smaller values indicate greater density (average and worst case, respectively).

For *coverage* we report the $l2$ distance between the two farthest Pareto candidates on the front. Larger values are better, indicating a wider range of the front is spanned by Pareto points found.

Table 2 reports results on benchmark Case I. Note that for fair evaluation, we only consider candidates that are produced within the feasible functional bounds for the problem. We also observe that LS performs best in terms of coverage ($l2$) for our run, while SUHNPF performs better across both uniformity measures.

Table 2: Evenness of spread of Pareto points found across methods for Case I, as measured by *uniformity* and *coverage*.

Method	LS	MOOMTL	PMTL	EPO	EPSE	PHN	**SUHNPF**
Avg. Dist.	0.087	0.089	0.030	0.035	0.059	0.031	**0.029**
Max. Dist.	0.261	0.235	0.117	0.122	0.231	0.085	**0.078**
Coverage	**1.256**	0.843	1.110	1.201	1.252	1.214	1.254

12 WHY PARETO FRONT LEARNING?

The goal of PFL [Navon et al., 2021] (or any Pareto HyperNetwork) is to induce the full Pareto manifold from training in order to be able to show users the entire space of optimal trade-offs that are feasible. This empowers users to then choose any solution point they prefer on the manifold, *a posteriori*. In contrast, prior point-based methods from operations research (OR) and multi-task learning (MTL) find individual Pareto points only. Lacking prior knowledge of the manifold, a user would have to formulate an

abstract preference trade-off over objectives (*e.g.,* 25% f_1, 75% f_2), input that to a point solver, and then see what they get. If they don't like the result, they would then have to iterate, running the point-solver repeatedly with different preferences until satisfied.

Motivation for building on HNPF. We build on HNPF [Singh et al., 2021] for two key reasons: 1) its support for non-convexity in functional objectives, variable domain, and/or constraints, due to the usage of the Fritz-John Conditions (FJC) and 2) its guarantee of Pareto solution correctness within the ϵ error tolerance parameter (1e-4 in our experiments). In addition to this, point-based solvers from OR are accurate and support non-convexity but are inefficient (wrt the scaling of variable dimension), while MTL solvers are efficient but have limited support and accuracy with non-convexity. SUHNPF strives to deliver both accuracy and scalability.

Motivation for SUHNPF over prior work. Prior MTL approaches to learning the Pareto front have sought to find an even spread of Pareto points across the front by using ray or cone-based methods to partition the space uniformly. However, these works assume that the spread is uniform and symmetrical in the functional space. However, the nature of the Pareto front is not known *a priori* and so making such symmetry assumptions can lead to misleading results and expectations. For example, **Fig. 2 (main)** and **Fig. 5 (main)** show that the Pareto front in the functional domain is not symmetric around the 45-degree line (or implicitly assumed as $\alpha = 0.5$). In contrast, adopting a hypernetwork approach allows learning the full Pareto front without any such assumptions regarding its shape. In comparison, the prior PHN [Navon et al., 2021] hypernetwork fits a posthoc regression surface over a set of Pareto points found by point-based solvers. In addition to being posthoc, it also inherits all the above limitations of point-based solvers (EPO or LS). In contrast, SUHNPF explicitly learns a classification boundary between Pareto vs. non-Pareto points via FJC.

Extraction of Trade-off value. For any general Pareto HyperNetwork, once the entire Pareto front is approximated, one can simply select a point on the front and compute the value of trade-off α *a posteriori*. The cost for post-computing the value would vary from one method to another. Similar to HNPF, we also take $\mathcal{O}(k)$ *i.e.,* linear runtime in objectives for α extraction.

Need for differentiable functions. We rely on objectives/constraints to be at least once differentiable, in accordance with the Fritz-John Conditions in **Section 4 (main)**. Furthermore, since our framework relies on gradients of the objectives to check for optimality, as a consequence, we need the objectives differentiable at least once for their gradient to exist and be computable. Indeed, this is true for all MTL methods as well, and in general any method that relies on gradient descent. As noted earlier, this further motivates continuing development of differentiable measures [Swezey et al., 2021].

Convex Utility. Although the final objective is a weighted linear combination of the two objectives: $\alpha f_1 + (1-\alpha) f_2$, note that f_1 and f_2 are losses operating on a neural network, hence the loss surface for both f_1 and f_2 is non-convex in nature. Convex combination of two convex functions is guaranteed to be convex, but not for the convex combination of two non-convex functions [Boyd et al., 2004]. Many practical classification and recommendation problems have been shown to be non-convex in nature [Hsieh et al., 2015], hence their linear (convex) combination can still lead to non-convex fronts. Practical examples include Low Rank Matrix Recovery and Robust Linear Regression as in [Jain et al., 2017].

Need for benchmarking on non-convex setting. Refer to **Fig 2 (main)** for Case I, where the non-convex region of the front is the boundary of the feasible set of solutions. If one takes convex combinations of the endpoints (1,0) and (0,1), constructing a 135° line, points can always be obtained on that line, that have lower values on both functions f_1 and f_2 which are strictly lying below the non-convex region of the front. However, given the feasible domain for the problem, one cannot go lower than the non-convex portion of the front around that region.

Although various Pareto solvers exist in different research communities, they do not scale well to practical problems at hand, especially to optimize weights of large neural networks. MTL approaches were motivated as developing scalable Pareto solvers to tackle such problems.

13 ANALYSIS OF LINEAR SCALARIZATION

We refer to Case I here for analysis on Linear Scalarization (LS).

$$f_1(x_1, x_2) = x_1,\ f_2(x_1, x_2) = 1 + x_2^2 - x_1 - 0.1 sin 3\pi x_1$$
$$\text{s.t.} \quad g_1 : 0 \leq x_1 \leq 1, g_2 : -2 \leq x_2 \leq 2$$

Pareto optimal points here correspond to stationary points for the scalarized objective $S(x_1, x_2) = \alpha f_1 + (1-\alpha) f_2$ for different trade-offs of $\alpha \in [0, 1]$. Although some prior studies have asserted that LS cannot handle any non-convexity, **Fig. 3 (main) (a)** shows that LS finds Pareto points in the non-convex portions of the front.

To explore this case further, Fig. 7 plots the contour surface of the objective $S(x_1, x_2)$ as a function of its variables x_1, x_2, for three different values of $\alpha \in [0.1, 0.5, 0.9]$ (similar functional plots can be shown for any values of $\alpha \in [0, 1]$). Across plots, we observe that the functional landscape contains one or more minima (marked by a red cross $\times$) for $\alpha \in (0, 0.5]$. Any gradient descent algorithm would settle on one of these minima, depending on the choice of initialization and step size. On the other hand, for $\alpha = 0$ or $\alpha = 1$ (optimizing only one objective, not shown), and for $\alpha \in (0.5, 1)$ (*i.e.,* linear function $f_1 = x$ dominating), there are no optima at all. In these cases, any gradient descent algorithm would settle on the boundary of the feasible set. We thus observe for Case I that $\forall \alpha \in [0, 1]$, LS will always find a feasible a Pareto candidate.

Note that we adopt the LS implementation from PMTL's public sourcecode only[8]. It is fairly straightforward to plug the Case I functions into their code and observe that LS is indeed producing Pareto candidates in the non-convex region of the front.

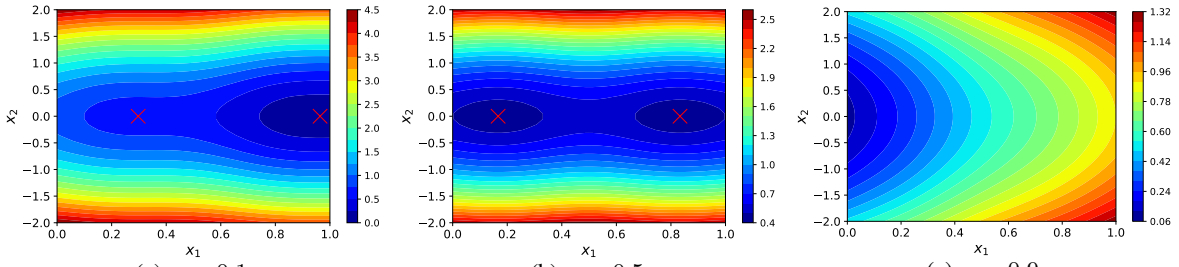
Note that if Case I had contained local or global maxima, LS's solving a minimization problem, min $S(x_1, x_2)$, would naturally not find these maxima.

References

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

François Chollet. keras. https://github.com/fchollet/keras, 2015.

[8] https://github.com/Xi-L/ParetoMTL



(a) $\alpha = 0.1$ (b) $\alpha = 0.5$ (c) $\alpha = 0.9$

Figure 7: Contour Plot of the scalarized objective $S(x_1, x_2) = \alpha f_1(x_1, x_2) + (1 - \alpha) f_2$ as a function of variables x_1, x_2 for different values of α. The heatmap shows the discretized contour levels. Red 'x' denotes the position of the minima on the surface.

Matthias Ehrgott and Margaret M Wiecek. Saddle points and pareto points in multiple objective programming. *Journal of Global Optimization*, 32(1):11–33, 2005.

Yu G Evtushenko and Mikhail Anatol'evich Posypkin. Nonuniform covering method as applied to multicriteria optimization problems with guaranteed accuracy. *Computational Mathematics and Mathematical Physics*, 53(2):144–157, 2013.

Massimiliano Gobbi, F Levi, Gianpiero Mastinu, and Giorgio Previati. On the analytical derivation of the pareto-optimal set with applications to structural design. *Structural and Multidisciplinary Optimization*, 51(3):645–657, 2015.

Cho-Jui Hsieh, Nagarajan Natarajan, and Inderjit Dhillon. Pu learning for matrix completion. In *International Conference on Machine Learning*, pages 2445–2453. PMLR, 2015.

Prateek Jain, Purushottam Kar, et al. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–363, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qing-Fu Zhang, and Sam Kwong. Pareto multi-task learning. *Advances in neural information processing systems*, 32, 2019.

Xi Lin, Zhiyuan Yang, Qingfu Zhang, and Sam Kwong. Controllable pareto multi-task learning. *arXiv preprint arXiv:2010.06313*, 2020.

Pingchuan Ma, Tao Du, and Wojciech Matusik. Efficient continuous pareto exploration in multi-task learning. In *International Conference on Machine Learning*, pages 6522–6531. PMLR, 2020.

Debabrata Mahapatra and Vaibhav Rajan. Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. In *International Conference on Machine Learning*, pages 6597–6607. PMLR, 2020.

Aviv Navon, Aviv Shamsian, Ethan Fetaya, and Gal Chechik. Learning the pareto front with hypernetworks. In *International Conference on Learning Representations*, 2021.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 525–536, 2018.

Gurpreet Singh, Soumyajit Gupta, Matthew Lease, and Clint Dawson. A hybrid 2-stage neural optimization for pareto front extraction. *arXiv preprint arXiv:2101.11684*, 2021.

Robin Swezey, Aditya Grover, Bruno Charron, and Stefano Ermon. Pirank: Scalable learning to rank via differentiable sorting. *Advances in Neural Information Processing Systems*, 34, 2021.

M Van Rooyen, X Zhou, and Sanjo Zlobec. A saddle-point characterization of pareto optima. *Mathematical programming*, 67(1): 77–88, 1994.