Feature Importance Random Search for Hyperparameter Optimization of Data-Consistent Model Inversion

Stephen Obonyo
Strathmore University
sobonyo@strathmore.edu

Isaiah Onando Mulang'
IBM Research
mulang.onando@ibm.com

Timothy Rumbell
IBM Research
thrumbel@us.ibm.com

Catherine Wanjiru
IBM Research
catherine.wanjiru@ibm.com

Viatcheslav Gurev
IBM Research
vgurev@us.ibm.com

Abstract

We consider hyperparameter optimization (HPO) of approaches that employ outputs of mechanistic models as priors in hybrid modeling for data consistent inversion. An implicit density estimator (DE) models a non-parametric distribution of model input parameters, and the push forward of those generated samples produces a model output distribution that should match a target distribution of observed data. A rejection sampler then filters out "undesirable" samples through a discriminator function. In a samples-generate-reject pipeline with the objective of fitting the push-forward to the observed experimental outputs, several DEs can be employed within the generator and discriminator components. However, the extensive evaluation of these end-to-end inversion frameworks is still lacking. Specifically, this data-consistent model inversion pipeline offers an extra challenge concerning optimization of constituent models. Traditional HPO are often limited to single-model scenarios and might not directly map to frameworks that optimize several models to achieve a single loss. To overcome the time overhead due to summative optimization of each component, and the expanded combinatorial search space, we introduce a method that performs an initial random search to bootstrap a HPO that applies weighted feature importance to gradually update the hyperparameter set, periodically probing the pipeline to track the loss. Our experiments show reduced number of time intensive pipeline runs but with the faster convergence.

1 Introduction

Hyperparameter optimization (HPO) is a daunting task for developers of intelligent and learning systems [5]. Optimality of model parameters determine the level of performance, and speed of training and inference [9]. In machine/deep learning (ML/DL), such hyperparameters include number of neurons per model layer, learning rate, and batch size etc [6, 17]. HPO methods can be classified into two types [6], namely: i) parallel search methods such as grid search [8, 10] and random search [1], that consume high amounts of compute attempting to reduce optimization times from several runs based on unique hyperparameter permutations; ii) sequential optimization methods, such as Bayesian Optimization (BO) [11], which incorporate a series of runs to gradually update subsequent runs with information from previous ones. Two inherent challenges in traditional HPO methods exist. First, resource consumption due to the combinatorial search space, and second, HPO predominately attempts to optimize a single model e.g in ML/DL [5].

In the domain of inverse modeling, 'data-consistent model inversion' (DCMI) refers to the setting where a distribution of model inputs should be inferred from a distribution of observed data [13, 14].

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

In DCMI for deterministic models, the push-forward model $\boldsymbol{y} = M(\boldsymbol{x})$ is a function known to relate some underlying input parameters to some observed outputs. Let the latent parameters be X with an assumed initial probability distribution P_X and the observed properties be Y with an assumed target distribution of observations Q_Y. The goal of DCMI, then, is to sample model inputs proportional to an approximate density $q_{X_g}(\boldsymbol{x})$ such that, after push-forward by $\boldsymbol{y} = M(\boldsymbol{x})$, model outputs are proportional to a density $\hat{q}_{Y_g}(\boldsymbol{y})$ that approximates Q_Y. Previous DCMI approaches include generative modeling using generative adversarial networks (GANs) to sample $q_{X_g}(\boldsymbol{x})$ [12, 14], and rejection sampling using density estimators [2, 14]. A recently developed algorithm, sequential DCMI (sDCMI, summarized in Appendix), combines aspects of these approaches, learning a non-parametric model in the form of a set of samples that are iteratively refined via density estimation-based rejection to approximate $q_{X_g}(\boldsymbol{x})$. This method is summarized by the 'Pipeline' in Figure 1, where the left side shows the generative model that iteratively learns to samples an implicit density $\hat{q}_{X_g}$ to draw parameter samples, and on the right a rejection method is employed by training density estimators to guide the generated samples according to $\hat{q}_{Y_g}$ in a sample-reject pipeline [13, 3]. Searching for the optimal hyperparameters in the sDCMI approach is challenging, as optimization is performed on multiple models sequentially throughout training. This work proposes an approach that alleviates the challenges of HPO based on sDCMI - described in the approach in Appendix Section A. **Our Contribution :** We introduce a novel end-to-end HPO of data consistent inverse modeling that applies an iterative feature importance based gradient update to guide random search.

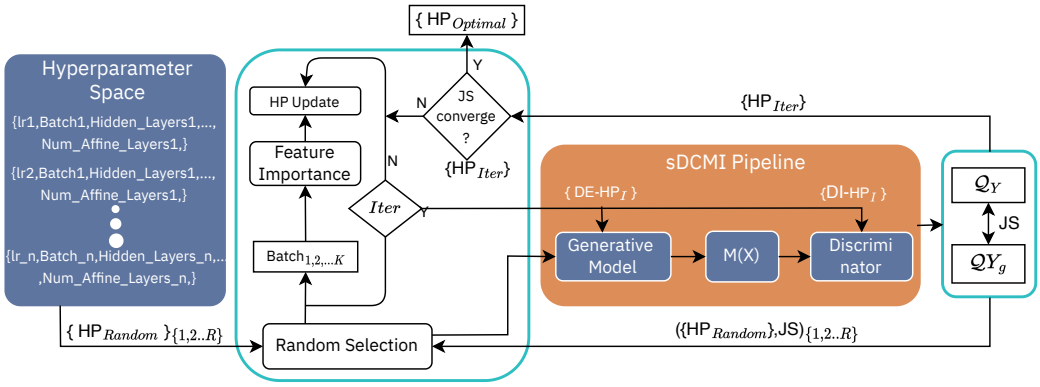

Figure 1: *Approach: consists of two major components, an iterative feature importance update, and the sDCMI modeling pipeline consisting of the density estimator in a perturbation kernel and the discriminator.*

2 Background and Problem Formulation

The goal of DCMI is to sample input parameters 'consistent' with observed data, in that when a model is simulated with the sampled parameters, the model outputs match the observations. In sDCMI, a proposal distribution of inputs Q_{X_g} is sampled by a generative model G that is trained to approximate the distribution Q_X. Passing the samples through the model generates outputs, $\boldsymbol{y} = M(\boldsymbol{x})$ within an output distribution Q_{Y_g}, which approximate the distribution of the observed outputs Q_Y. An additional constraint 'regularizes' the generated samples according to the prior distribution P_X. The sDCMI objective can be succinctly represented using

$$\begin{aligned} &\text{given} && P_X;\ Q_Y;\ \boldsymbol{y} = M(\boldsymbol{x}) \\ &\text{minimize} && D_f(Q_{X_g} jj P_X) \\ &\text{subject to} && supp(X_g) \quad supp(X);\ D_f(Q_{Y_g} jj Q_Y) = 0 \\ &\text{where} && \boldsymbol{y}_g = M(\boldsymbol{x}_g) \quad Q_{Y_g};\ \boldsymbol{x}_g \quad Q_{X_g}: \end{aligned} \tag{1}$$

In equation (1), $D_f(\,jj\,)$ is an f-divergence measure such as Jensen-Shannon (JS) divergence. The resulting distribution Y Q_{Y_g} is compared to the target observed output distribution Y Q_Y, and the divergence is optimized. Density estimator models (also parameterized) ( ) score the inputs and outputs using density ratio estimation for evaluating the divergence measures in equation 1.

In this work we consider HPO on the incorporated DL/ML models G ( ) and ( ) in the sDCMI framework. These DL/ML models can be implemented using various methods, including the possibility of m models stacked together for boosting density estimation, and k models in the discriminator, the total set of hyperparameters becomes: $_T = S_{g=1}^{m} f\ 2\ ^{(g)}g\ \_\ S_{d}^{k} f\ 2$

$^{(d)}g$. The HPO is a minimization problem dependent on the combinatorial $_{j}\ ^{\tau}{}_{\tau j}$ illustrated in blue box of Figure 1:

$$\text{argmin}\ _{c\,2\left(\frac{\Theta_T}{|\Theta_T|}\right)}\ D_f(\ )\ j\ G\ (\ );\quad (\ );\ M(\ ) \tag{2}$$

3 Approach: Feature Importance guided Random search

Our approach, illustrated in Figure 1, encapsulates the sDCMI pipeline within an iterative feature importance guided random search. We follow the intuition reported in the literature [1] that well guided random search should return optimal hyperparameters. The set of parameters in our setting includes a combination of parameters for different pipeline components, i.e., $G\ (\ )$ and $(\ )$. Algorithm 1 initializes a set of randomly selected hyperparameters , that are passed through the pipeline, to obtain the JS loss (line 3) between Q_Y and Q_{Y_g}. A set of K random trials through generates the (hyperparameter configurations, JS loss) pairs to be used in weighting feature importance. In the next step, batches of size bs are iteratively drawn from the generated pairs and used to train a supervised function W(xtrain; ytrain) where xtrain are hyperparameters, and ytrain are corresponding JS losses. W (e.g. a neural network or a linear regression model) learns incrementally to produce feature importance I. To update we scale I by a weight parameter (Algorithm 1, lines 7-9). Periodically the is probed with a single trial of the instantaneous to check the performance of the updates (i.e. if the pipeline JS loss is decreasing). All hyperparameters are normalised.

Algorithm 1 HPO with Feature Importance Weighting

1: **procedure** sDCMI_HPO($\Psi, N, M, K, \alpha, bs, probe$) ▷ Ψ, sDCMI pipeline
2: Λ InitHypeparameters() ▷ initialise hyperparameters
3: JS `RunPipeline`(Ψ, Λ, trials=1)
4: sampleParams, sampleJS `RunPipeline`(Ψ, randomParams, trials=K)
5: **for** i 0 to N **do**
6: xtrain, ytrain random(sampleParams, bs), random(sampleJS, bs)
7: I W(xtrain, ytrain) ▷ W weighting Model
8: Λ $\Lambda + \alpha(\Lambda \quad I)$
9: **if** i % probe $= 0$ **then**
10: currentJS `RunPipeline`(Ψ, Λ, trials=1) ▷ Run sDCMI Pipeline
11: **if** currentJS $<$ JS **then**
12: JS currentJS ▷ Λ^* optimal hyperparams
13: Λ^* Λ
14: **return** Λ^*

4 Experiments

Data. We use simulated data for both X and Y. X is 2-dimensional. The prior P_X was set according to ($x_1; x_2$ $U(0; 2)$) and was initially sampled through quasi Monte Carlo. The *push forward* model ($M(\ )$) is a 2-dimensional Rosenbrock function that generates a 1-dimensional y. with a normal distribution target according to $N(250; 10)$.

Setup. In the sDCMI pipeline, there are two key components: a generative model (acting as a perturbation kernel), for which we use a Diffusion Model (**DM**) and discriminator(s) that consist of DE(s) implemented using e.g. Gaussian mixture models **GMM**) or affine coupling normalizing flow networks (**AC**). Throughout the experiments, the generative model was set to the DM, while the discriminator density was varied between AC and GMM models giving two sets of pipeline configurations: (i) **DM-AC**, (ii) **DM-GMM**. A listing of parameters of each model is given in Table 1.

Diffusion Models - DM	Affine Coupling - AC	Gaussian Mixtures - GMM
Forward diffusion steps - ***steps***, number of hidden layers - ***hdif.***, learning rate - ***lr***, batch size - ***bs***, number of training epochs - ***epoch***, learning rate - ***lr*** and dropout rate - ***drop***	number of layers - ***layers***, hidden units size - ***hclas.***, learning rate - ***lr***, the batch size - ***bs*** and dropout - ***drop***	number of components - ***comp***, scale, delta and the number of iterations - ***n_iters***

Table 1: *Listing of hyperparameters.*

Baselines. We consider three existing HPO search methods, Optuna, HyperOpt and BO. For each method, 100 trials were run. Similarly, for our Algorithm, 100 trials we sampled from the sDCMI

pipeline. For **DM-AC** and **DM-GMM** pipeline, the parameters of our Algorithm 1 were set as follows: K=100, =0.001, W a linear regression model based on SGD implementation where the next iteration weight update are bootstrapped with the previous one, $probe$=100, N=10K. Figure 2 shows the results of the iteration runs.

Discriminator Density			**AC Discriminator**				**GMM Discriminator**	
HPO method	Ours	bayes	hyper	optuna	Ours	bayes	hyper	optuna
Mean	0.0029	0.0609	0.0452	0.0609	0.0006	0.0802	0.0844	0.0865
Std.	0.0028	0.0683	0.0631	0.0685	0.0008	0.0772	0.0810	0.0819

Table 2: *Average JS (on $Q_{Y_g} jj Q_Y$) of hyeperparameters over 100 trials for BO (bayes), Optuna (optuna) and HyperOpt (hyper). DM-AC and DM-GMM pipeline are repeated 10 time to get the sample JS scores.*

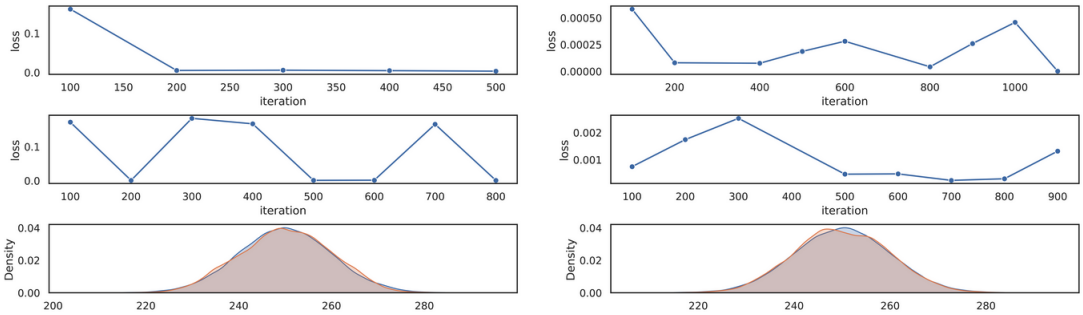

Figure 2: *Row 1: **DM-AC** pipeline with 2-runs of the Algorithm 1. Row 2: **DM-GMM** pipeline. Row 3: The Q_{Y_g} and Q_Y fit based on the optimal hyperparameters from the **DM-AC** (row 1, column 2) and **DM-GMM** (row 2 , column 2) at iteration 800. Plots correspond to the results obtained from probing the pipeline.*

	drop.	hdif.	steps	lr	layers	bs	hclas.	epoch	scale	delta	comp.	iter
DM-AC	0.38	7	13	0.005	4	908	7	15	-	-	-	-
DM-GMM	0.43	18	19	0.009	-	857	-	19	0.39	0.012	38	1566

Table 3: *Optimal **DM-AC** and **DM-GMM** hyperparameters. Abbreviations are as indicated in Table 1*

Results. The JS losses start high initially and gradually decrease over iterations. While there a slow convergence of the optimal hyperparameters our algorithm updates hyperparameters when the JS loss from the pipeline probing is less than the current. As such, only the hyperparameters leading to the global JS convergence are returned ( ). We present a set of hyperparameters leading to the best JS loss in Table 3 and compare our method with other baselines in Table 2. While the proposed algorithm generate results comparable to the existing HPO methods, our approach record the lowest JS values while learning on only 100 randomly sampled trials sampled from the sDCMI pipeline. The majority of existing HPO baselines are designed and tested on a single model framework e.g. a single DL model. However, in sDCMI setting there are a number of components which are part of of the framework e.g. generative model, and density estimators used in discriminators. The selection of a hyperparameter in a sub-component can influence the performance of another sub-component in the same or a different sub-component. ML/DL-guided random search is a well established problem solving methodology that records state-of-the-art results in problems in large search spaces such as computer game playing [15, 16] and molecular modeling [7]. In this work, only a subset of hyperparameters were studied. Including all the hyperparameters in the sDCMI pipeline is key in fully optimizing the HPO pipeline.

5 Conclusion and Future Work

We have presented a feature-based weighting end-to-end HPO algorithm for sDCMI pipeline. With 100 trials, the iterative algorithm incrementally updates the set of hyperparameters based on the learned feature weights. Subsequently, the pipeline is probed to evaluate the quality of the hyper-parameter updates. Inclusion of the categorical hyperparameters and scaling the HPO with all the hyperparameters in the sDCMI pipeline are key future research fronts.

References

[1] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305.

[2] Butler, T., Jakeman, J., and Wildey, T. (2018). Combining push-forward measures and bayes' rule to construct consistent solutions to stochastic inverse problems. *SIAM Journal on Scientific Computing*, 40(2):A984–A1011.

[3] Butler, T., Wildey, T., and Yen, T. Y. (2020). Data-consistent inversion for stochastic input-to-output maps. *Inverse Problems*, 36:085015.

[4] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

[5] Higuchi, K., Sano, S., and Igarashi, T. (2021). Interactive hyperparameter optimization with paintable timelines. In *Proceedings of the 2021 ACM Designing Interactive Systems Conference*, DIS '21, page 1518–1528, New York, NY, USA. Association for Computing Machinery.

[6] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population based training of neural networks. *CoRR*, abs/1711.09846.

[7] Koch, M., Duigou, T., and Faulon, J.-L. (2019). Reinforcement learning for bioretrosynthesis. *ACS synthetic biology*, 9(1):157–168.

[8] Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR*, abs/1912.06059.

[9] Liu, Y., Wang, X., Xu, X., Yang, J., and Zhu, W. (2021). Meta hyperparameter optimization with adversarial proxy subsets sampling. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 1109–1118, New York, NY, USA. Association for Computing Machinery.

[10] Ndiaye, E., Le, T., Fercoq, O., Salmon, J., and Takeuchi, I. (2019). Safe grid search with optimal complexity. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4771–4780. PMLR.

[11] Nguyen, V. (2019). Bayesian optimization for accelerating hyper-parameter tuning. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 302–305.

[12] Parikh, J., Rumbell, T., Butova, X., Myachina, T., Acero, J. C., Khamzin, S., Solovyova, O., Kozloski, J., Khokhlova, A., and Gurev, V. (2022). Generative adversarial networks for construction of virtual populations of mechanistic models: simulations to study omecamtiv mecarbil action. *Journal of Pharmacokinetics and Pharmacodynamics*, pages 1–14.

[13] Pilosov, M., del Castillo-Negrete, C., Yen, T. Y., Butler, T., and Dawson, C. (2023). Parameter estimation with maximal updated densities. *Computer Methods in Applied Mechanics and Engineering*, 407:115906.

[14] Rumbell, T., Parikh, J., Kozloski, J., and Gurev, V. (2023). Novel and flexible parameter estimation methods for data-consistent inversion in mechanistic modeling. *arXiv preprint arXiv:2009.08267*.

[15] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[16] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[17] Takenaga, S., Ozaki, Y., and Onishi, M. (2023). Dynamic fidelity selection for hyperparameter optimization. In *Proceedings of GECCO '23 Companion*, New York, NY, USA. ACM.

A Sequential Data-Consistent Model Inversion (sDCMI)

In this section we describe sDCMI, the pipeline which informs our HPO work. *The sDCMI work is currently under review. We will include the full reference once the work is published or archived.*

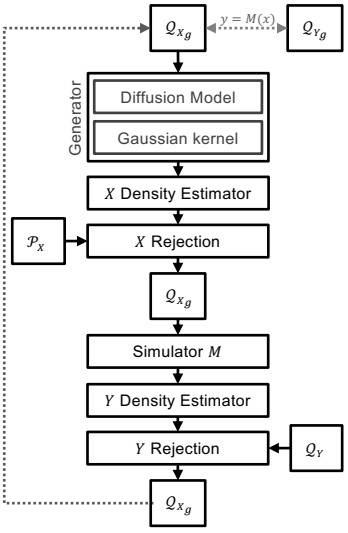

Figure 3: *sDCMI Modelling Pipeline. Source : [Under Review]*

In the pipeline, the samples X are initially drawn from the prior distribution P_X, resulting in the initial values of Q_{X_g}. Samples Y on the other hand are obtained by *push forward* of X through function M leading to Q_{Y_g} with a corresponding target distribution Q_Y. Q_{X_g} samples are passed to the generator module (only Diffusion Model is tested in our HPO work) which allows for the exploration of the parameter space of the given input by adding Gaussian noise (perturbation) to the output. This process leads to generation of additional data points which augment the input data to get a set of output samples. Subsequently, the output is fitted by another density estimator (in our case we experiment with Affine Coupling and Gaussian Mixture Model), then followed by a rejection process where points with greater density than the target (P_X or Q_Y) are more likely to be removed. The rejection is controlled by a rejection fraction parameter - set to 0.5 throughout in our HPO work. This sample-perturbation-fit process is sequentially repeated for a number of iterations - set to 10 in the HPO experiments. While the primary objective is to optimize the divergence $D_f(Q_{Y_g} jj Q_Y)$, ability to achieve similar performance on the input space - $D_f(Q_{X_g} jj P_X)$) - is also key (secondary objective).

B Feature Importance Partial Dependency Plots (PDPs)

We generated the Partial Dependency Plots (PDPs) by combining the result of all the trials: Optuna, HyperOpt, Bayesian Optimization trials. The PDPs shows the marginal effect a variable has on another [4]. These marginal effect plots are key in the processes on validating our algorithm outputs. The plots are obtained by fitting a Gradient Boosted regressor model with the hyperparameters, the , as the input and the corresponding JS losses as the target. Subsequently, each hyperparameter marginal effect is generated by keeping all other hyperparameter values constant while changing one with the fitted model.

Some key observations from plot in Figure 4 is that (i) the diffusion steps (n_steps) greater than 20 leads no better performance but worse time and space complexity, (ii) a similar observation can also be made architectures as poor performers, (iii) the lower learning rates (lr) while training the Diffusion Model and the Affine Coupling discriminator are better performers, (iv) other hyperparameters values do not have a strong effect on the JS loss, however, can affect the pipeline complexity e.g. longer training epochs of the diffusion model.