Advances in Tabular Machine Learning (WS 2024/25) 1-22 Submitted 15.03.2025

Interpretable Neural Networks using EAGGA

Simon Stürzebecher SIMON.STUERZEBECHER@CAMPUS.LMU.DE

Abstract

- tabular data still difficult for NNs, where it's still outperformed by other ML model classes - research suggests NN performance benefits from heavy regularisation - using EAGGA, we regularise an NN and achieve both comparable performance as XGB on EAGGA as well as interpretability - for this, we propose a network architecture specifically suited for the EAGGA algorithm

Keywords: tabular data, multi-objective optimization, interpretability, deep learning

1 Introduction

Tabular Data - most common type of data - still difficult for neural networks - (Borisov et al., 2024, p. 7499) - recent research suggests that strong regularisation is beneficial to NN performance on tab data (Kadra et al., 2021, 8)

- we know regularisation from linear models can come with improvements in interpretabiltiy * e.g. LM-LASSO (L1) regularisation as feature selection -¿ reduces # features used in model by setting some coeffs to 0 (Tibshirani, 2018, p. 267) - other forms of regularisation improve performance * e.g. LM-Ridge (L2) on multi-collinear data (Hoerl and Kennard, 1970, p. 55) * e.g. NN dropout, reduces co-adaptation (Hinton et al., 2012, p. 1) * e.g. NN early stopping, reduces overfitting on training data (Finnoff et al., 1993, p. 778f)

- we want to explore if we can use NN regularisation to tackle both interpretability and improved performance (i.e. "comparable" to XGBoost) on tab data - using EAGGA framework, which proved it can improve performance while keeping (already high) performance of XGBoost on tab data -¿ see if interpretabiltiy improvements translate to NN + performance can also be on par

2 Background and Related Works

2.1 Interpretability

As there is no clear definition for interpretability, we will consider it as "the ability to provide *explanations* in *understandable terms* to a human", where *explanations* are logical decision rules and *understandable terms* relate to commonly used terms in the domain of the problem, as suggested by Zhang et al. (2021, chap. 1). Further, we use the term "explainability" in an exchangeable manner with "interpretability", as is commonly done.

Explainability of a model's reasoning is in many ways desirable. Zach (2019, pp. 3-4) gives a range of examples, amongst which are *gaining trust*, e.g. when doctors rely on medical diagnosis predictions, avoiding *subconcious biases* by making sure loan approvals are non-discriminatory, or *regulatory* reasons, most notably the EU's "Right to Explanation" warranted by the GDPR (Antamis et al., 2024, p. 1) or for approval of drugs discovered

using machine learning models (Zhang et al., 2021, 1B). It can further prove helpful for explaining unexpected drops in model performance, which could arise from optimising a model with respect to loss and then judging its performance on a different metric such as accuracy, a practice known as *model debugging* (Zhang et al., 2021, 1B) or for *scientific understanding* in domains where only models can make sense of increasingly complex data anymore and learnt knowledge encoded in a model needs to be made accessible to humans to be used reliably (Antamis et al., 2024, p. 1).

Commonly, methods for model interpretation are divided into intrinsic methods, where the search space only comprises models with a structure simple enough to be considered "explainable" (such as tree-based or simple linear models), and post-hoc methods, where interpretation methods are applied after model training. Amongst post-hoc techniques, we can further divide the space into model-specific (such as analysing GLM coefficients) and model-agnostic (e.g. partial depence plots, ALE) methods (Molnar, 2022, chap. 3.2).

Zhang et al. (2021, chap. 2) extend this distinction to a three-dimensional taxonomy, allowing for better categorisation of neural networks (NNs), a model class that in its fully-connected feedforwad form is inherently non-interpretable.
Passive vs Active Approaches, where *passive* are all post-hoc methods and *active* methods actively change either the architecture or training process to increase model interpretability.
Type of Explanations, distinguishing between *example* methods, providing concrete examples of what leads to a desired output, *attribution* methods that attribute the effect on the output for a specific feature, *hidden semantic* methods, which explain the types of inputs particular neurons or layers pick up on, and logical *rules*, such as if-then clauses or tree-induced rules.
Local vs Global Interpretability, ranging from *local* methods providing explanations based on individual samples, *semi-local* methods, explaining model behaviour for sets of samples grouped by some criterion, to *global* methods, which explain the network as a whole.

Given its unclear definition, evaluating interpretability can be challenging. Doshi-Velez and Kim (2017, 3) propose a taxonomy to categorise possible evaluation methods based on their rigorousness.
Application-grounded evaluation evaluates interpretations directly with respect to the task, by having human experts evaluate the outcome and is therefore the most expensive and time-consuming of the three approaches.
Human-grounded metrics is similar to application-grounded evaluation in that a human still evaluates the interpretations, but tries to simplify the task so that a layperson can do it. Human-grounded approaches are especially suitable if it's sufficient to validate the general concepts of a task. A typical evaluation set-up in this category is binary forced choice, where the human evaluator chooses, which of two generated explanations he prefers.
Functionally-grounded evaluation is the least rigorous, but easiest to implement of the three. It assess explanatory quality according to some formally defined proxy for interpretability and is particularly useful in ranking different models if their model-class is already identified (e.g. via human-grounded evaluation) to be interpretable. The main challenge for functionally-grounded evaluation is finding a good proxy.

2.2 Hyperparameter Optimization (HPO)

In contrast to model parameters, which are optimized during training, hyperparameters (HPs) are those describing the machine learning algorithm and are fixed before training. Because they usually have a significant impact on the trained model's performance, HPs are often optimized, too. Let $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ be a dataset consisting of n tuples drawn from the data-generating distribution, i.e. $(\boldsymbol{x}^{(i)}, y^{(i)}) \overset{i.i.d.}{\sim} \mathbb{P}_{\boldsymbol{xy}}, \forall i = 1, ..., n$. Furthermore, let $\mathcal{I} : (\mathbb{D} \times \boldsymbol{\Lambda}) \to \mathcal{H}, (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f}$ be an algorithm that maps a given dataset $\mathcal{D}$ and HP configuration $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ to a model $\hat{f}$. Denote with $\mathcal{I}_{\boldsymbol{\lambda}}$ an algorithm with fixed HP configuration a chosen loss function with L. The goal of model training is, for fixed $\boldsymbol{\lambda}$, to have $\mathcal{I}_{\boldsymbol{\lambda}}$ find model $\hat{f}$ minimising the expected generalisation error $GE(\mathcal{I}_{\boldsymbol{\lambda}}, \mathcal{D}, L) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathbb{P}_{\boldsymbol{xy}}}[L(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})(\boldsymbol{x}))]$ The goal of HPO, on the other hand, is to find $\boldsymbol{\lambda}$ minimising the expected generalisation error, i.e. $\arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} GE(\mathcal{I}_{\boldsymbol{\lambda}}, \mathcal{D}, L)$. For this, there is no analytical expression available, making HPO a blackbox optimization problem. (Karl et al., 2023, pp. 2f)

We will first outline model-free methods to approach blackbox optimization problems, before presenting a model-based variant.

The most basic model-free strategies for blackbox optimization are grid and random search. In the latter, the user defines a range of interest for each HP and then evaluating points along a grid within the cartesian product of those. This has the drawback of scaling extremely poorly in both number of HP dimensions and number of query points per range of interest. Random search, on the other hand, randomly samples a value for each HP until it runs out of budget. Its explorative nature, ease of use, and no assumptions about the model makes it a good baseline. It is also superior to grid search in cases were one or more HPs have little impact on model performance, as for a given budget B, each HP will likely be queried with B different values, whereas for grid search each HP will only be queried with $B^{1/N}$ different values for N HPs. (Feurer and Hutter, 2019, chap. 1.3)

Another popular class of model-free optimizers are evolutionary algorithms (EAs), which are conceptually simple and can handle even complex parameter spaces, given appropriate implementation of operators. EAs are iteratively evolving a population of individuals (an individual is simply a hyperparameter configuration) of size μ, where in each iteration (generation), λ[1] offspring are generated via three operators (Goldberg, 1989, pp. 10-14).
Reproduction samples an individual from the population to reproduce with probability proportional to its fitnes. For HPO, fitness usually refers to the performance metric the model resulting from the individual's HP configuration is evaluated on.
Crossover selects two "parents" from the pool of reproducing individuals. An index k is sampled at random and all HP values from index k on are swapped, yielding two new "children" individuals.
Mutation randomly changes single HP values of an individual chosen for reproduction, e.g. by adding Gaussian noise to real values or flipping bits on binary values.
At the end of each generation, the EA keeps the best μ individuals, either only from the offspring ("(μ, λ)-selection") or more commonly from population and offspring ("$(\mu + \lambda)$-selection"), which is also called an "elitist" strategy, as it's guaranteed to keep the best individual. (Feurer and Hutter, 2019, chap. 1.3)

1. We distinguish between $\lambda \in \mathbb{N}$ when referring to the offspring size of EAs and $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ when referring to a hyperparameter configuration.

One of the most popular EA implementations is the "Covariance Matrix Adaptation Evolution Strategy" (CMA-ES). It employs a multivariate normal distribution to generate offspring, for which the mean is a weighted average of the previous generation's individuals and the covariance matrix similarly is the weighted covariance of the previous generation. The weighing scheme for both is done in a way as to reproduce previously successful (i.e. selected) steps (Hansen, 2023, p. 8-11).

Another popular EA approach is Differential Evolution. Its populaton is randomly initialised so the entire search space is covered. In each generation g, for each $\boldsymbol{\lambda}_{i,g}$ with $i = 1, ..., n$ it creates "mutant vectors" $\boldsymbol{\nu}_{i,g+1} = \boldsymbol{\lambda}_{r_1,g} + F \cdot (\boldsymbol{\lambda}_{r_2,g} - \boldsymbol{\lambda}_{r_3,g})$, where $r_1, r_2, r_3 \in \{1, ..., n\}$ are mutually different random indices that also different from i and $F \in [0, 2]$ is constant. For crossover, it then generates a "trial vector" $\boldsymbol{\upsilon}_{i,g+1} = (\upsilon^{(1)}_{i,g+1}, \upsilon^{(2)}_{i,g+1}, ..., \upsilon^{(N)}_{i,g+1})^T$ with

$$\upsilon^{(j)}_{i,g+1} = \begin{cases} \nu^{(j)}_{i,g+1} & \text{if } u \leq \text{CR or } j = R \\ \lambda^{(j)}_{i,g} & \text{else} \end{cases} \tag{1}$$

for a random index $R \in \{1, 2, ..., D\}$, sampled $u \sim U[0, 1]$, and crossover constant $\text{CR} \in [0, 1]$. Selection is done by picking the better of $\boldsymbol{\upsilon}_{i,g+1}$ and $\boldsymbol{\lambda}_{i,g}$ with respect to fitness. (Storn and Price, 1997, p. 343)

Contrasting to model-free approaches, model-based methods fit a surrogate model on the target function and optimize the surrogate. Currently, one of the most popular model-based approaches is Bayesian Optimization (BO). BO is an iterative algorithm comprising a probabilistic surrogate model $\tilde{f}$ for the blackbox problem f and an acquisition function. It maintains and continually extends a set of points it already evaluated on the target function. In each iteration, the posterior predictive distribution is determined by fitting the surrogate model on this set of points. The acquisition function then retrieves the highest utility point from the surrogate model and evaluates it on the target function, after which it adds the point to the set of queried points (Feurer and Hutter, 2019, chap. 1.3.2) and (Frazier, 2018, pp. 2f). Evidently, neither the target function nor the surrogate model are optimized directly. Rather, the acquisition function trades-off exploration and exploitation of the surrogate and is maximised to yield the next query point most likely to optimize the two objectives as defined by the acquisition function.
Common choices for the acquisition function are Expected Improvement (Eq. 2) and Thompson Sampling (Eq. 3).

$$\text{EI}_t(\boldsymbol{\lambda}) := \mathbb{E}_t[(f(\boldsymbol{\lambda}) - f^*_t)^+] \tag{2}$$

Expected Improvement computes the expected minimisation over the incumbent f^*_t for evaluating the target function f at query point $\boldsymbol{\lambda}$. $\mathbb{E}_t$ denotes the expectation under the posterior predictive distribution with t query points, similarly, the current incumbent is the minimum value of the target function from previous evaluations $(\boldsymbol{\lambda}_{1:t}, f(\boldsymbol{\lambda}_{1:t}))$. (Frazier, 2018, p. 7)

$$f^{(t)} \sim \tilde{f}_t \tag{3}$$

Thompson Sampling, on the other hand, can conceptually be described as drawing a candidate function $f^{(t)}$ from the posterior predictive distribution and minimising it to obtain the next query point for the target function. (Shahriari et al., 2016, p. 161)

Composing the other part of BO, surrogate models need to be able to model mean and variance of its target function estimate. Commonly chosen models are Gaussian Processes (Eq. 4) and Random Forests.

$$\mathrm{GP}(m(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')) \tag{4}$$

Gaussian Processes (GP) are fully specified by their mean and covariance functions and have closed-form solutions. The mean $m(\boldsymbol{\lambda})$ fits all points already evaluated on the target function, while for the covariance or kernel function $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ it is desirable that points close to each other in HP space $\boldsymbol{\Lambda}$ exhibit stronger correlation than those far apart. The kernel function can be freely chosen, a common choices is the Gaussian (Eq. 5) kernel with hyperparameter α_0. (Frazier, 2018, p. 5)

$$k(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \alpha_0 \exp(-\|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\|_2^2) \tag{5}$$

A drawback of Gaussian Processes is its poor scalability in number of data points and number of hyperparameters, although there exist workarounds approximating the full GP with only a subset of the samples. (Feurer and Hutter, 2019, chap. 1.3.2)
Random Forests, unlike GPs, can handle complex hyperparameter spaces even including categorical and hierarchical HPs. Their computational complexity is also far less, making them a popular alternative to Gaussian Processes. (Feurer and Hutter, 2019, chap. 1.3.2)

2.3 Neural Architecture Search

A special case of HPO is Neural Architecture Search (NAS). Because of the flexibility of neural network architectures (number of hidden layers, strength of dropout, each layer can have a different numer of neurons, each layer can have a different activation function, etc.), traditional HPO methods cannot efficiently explore the entire space. In our extension of the EAGGA approach we don't employ NAS as research focusses on the NLP and image domains, where features (i.e. tokens or pixels, respectively) exhibit strong correlation amongst themselves, which is usually not the case for tabular data (Borisov et al., 2024, p. 7499). Still, we want to outline two notable approaches for neural architecture search in this paper, one of which are **cell search spcaces**. These modularise an NN into a chain-structure of cells, where each cell is a basic building block of the respective architecture. For regular feedforwad neural networks, this could be a linear layer of fixed size and with a specific activation function, for CNNs this could be a specific convolutional operation or pooling function, for instance. Instead of tuning every hyperparameter, the sequential placement of the cells is then optimzed during NAS, e.g. via random search or Bayesian Optimization. (Elsken et al., 2019, chap. 3.2)
Treating a neural network as a sequence of cells instead of one fixed entity further allows exploiting this structure with special sequential optimization techniques. Zoph and Le (2017, p. 3) and Zoph et al. (2018, pp. 2-4) propose a cell search method using RNNs and reinforcement learning. The cells in their approach are not fixed, but have hyperparameters themselves, such as the filter size and the stride size of a convolutional layer. These hyperparameters are then predicted sequentially, given all the already predicted hyperparameters of previous layers. The RNN is trained using reinforcement learning, where the actions are the different values the network can predict for a given hyperparameter and the reward function is the RNN's performance on held-out validation data.

The second approach to NAS is the **one-shot model**, which trains a "fabric", which can be understood as a DAG comprising all architectures of a given search space. Each path through the DAG is one specific network architecture, where the nodes represent a layer of neurons and the edges in-between are operations on the layers' values. The entire DAG has one designated input and one output node. It is then trained just like a regular NN, after which the optimal path, i.e. network architecture, is selected from the fabric. This has the advantage of, albeit being slower than training a single model, being far more efficient and less expensive than training all architectures encoded in the fabric. (Saxena and Verbeek, 2017, pp. 1-2, p.8)

2.4 Multi-Objective Optimization

In most applications, practicioners don't just want to optimize for performance exclusively but, for instance, also desire an interpretable model, which they may measure via some proxy metric. Let $c_1, c_2, ..., c_m = \boldsymbol{c} : \boldsymbol{\Lambda} \to \mathbb{R}^m$ be the vector of m objectives, the goal of multi-objective optimization (MOO) is to minimise this vector (Karl et al., 2023, p. 11).

2.4.1 Pareto-optimality

Optimizing $\boldsymbol{c}$ usually comes with the challenge of conflicting objectives: improving one objective means deteriorating another. We thus seek to find a set of trade-off solutions, so called *non-dominated* points. A point $\boldsymbol{\lambda}$ dominates another point $\boldsymbol{\lambda}'$ ($\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$) if there is no other point that is strictly better in at least one objective and better or equal in the remaining ones. Formally,

$$\forall i \in \{1, ..., m\} : c_i(\boldsymbol{\lambda}) \leq c_i(\boldsymbol{\lambda}') \wedge \exists j \in \{1, ..., m\} : c_j(\boldsymbol{\lambda}) < c_j(\boldsymbol{\lambda}') \tag{6}$$

as defined in (Karl et al., 2023, pp. 7f) and (Goldberg, 1989, pp. 198f).
The set of non-dominated or *Pareto-optimal* points $\mathcal{P} := \{\boldsymbol{\lambda} \in \boldsymbol{\Lambda} | \nexists \boldsymbol{\lambda}' \in \boldsymbol{\Lambda} \text{ s.t. } \boldsymbol{\lambda}' \prec \boldsymbol{\lambda}\}$ is commonly referred to as the *Pareto set* and its image as the *Pareto front*. The goal of MOO is to find a set of non-dominated points $\hat{\mathcal{P}}$ approximating the true Pareto set $\mathcal{P}$ well.

There are different techniques to evaluate an estimated Pareto-front. If we have knowledge over the true Pareto-front, we can compute the distance between the approximation and the true front. In most cases, this knowledge cannot be assumed. Then, usually volume-based approaches are used for evaluation, most notably the hypervolume. The hypervolume, or S-metric, of a Pareto-front, computes the volume between the estimated front and some chosen reference point, which is usually the worst point in objective space. The larger the hypervolume, the closer the estimate is to the true Pareto-front. (Karl et al., 2023, pp. 8-10)

2.4.2 A-priori

For optimizing a multi-objective problem, there are largely two general approaches. One of those are *a-priori* methods, of which we will briefly outline two popular techniques.

Scalarization approaches impose an implicit order of priority on the objectives. The simplest one is optimizing a weighted sum of objective functions

$$\arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \sum_{i=1}^{m} w_i c_i(\boldsymbol{\lambda}) \text{ s.t. } \sum_{i=1}^{m} w_i = 1 \wedge w_i > 0, \forall i = 1, ..., m \tag{7}$$

Weights are chosen a-priori by the user and the solution is very sensitive to them, making this method very reliant on different users' preferences. (Karl et al., 2023, p. 11) and (Srinivas and Deb, 1994, chap. 3.1)
Another form of scalarization is the ϵ-constraint, which translates all but one objective into constraints and then optimizes the remaining objective subject to the constraints. Without loss of generality, the first constraint can be optimized

$$\arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} c_1(\boldsymbol{\lambda}) \text{ s.t. } c_2(\boldsymbol{\lambda}) \leq \epsilon_2, ..., c_m(\boldsymbol{\lambda}) \leq \epsilon_m \tag{8}$$

This method is, similarly to the weighted sum, conceptually simple yet also very sensitive to the chosen constraints. (Karl et al., 2023, p. 12)
Alternative to scalarization is the **lexicographic method**. For it, the user assigns a priority to each objective and then greedily optimizes each objective in order of priority, constrained to the solutions of the already optimized, higher-priority objectives. (Riera et al., 2023, p. 13749) Again, solutions are very dependent on the user-defined priorisation.

2.4.3 A-posteriori

The disadvantage of a-priori methods is the missing knowledge of the interplay between a hyperparameter configuration and the trained model's performance across objectives: we can either restrict the search space to enforce some will, such as only using a maximum of 50% of features, optimize for performance and use the resulting model, or leave the search space unrestricted, adjust the loss function to incorporate all objectives and take its optimum. In neither case do we know the impact our a-priori trade-off has on the final performance. In practical applications, the impact on performance across all objectives is what matters. This is the main advantage of a-posteriori methods: instead of implicitly defining a trade-off prior to training, we still evaluate multiple HP configurations but keep the set of non-dominated solutions instead of just one that satisfies the trade-off criterion. This is beneficial as it makes the relationship visible, a practicioner can now see, for instance, that a slight decrease in one objective might translate to a significant improvement in another that would have been missed if the problem was optimized with a-priori methods. Aside from the usual baselines grid and random search there are multi-objetive Bayesian Optimization adaptations, mainly using one of two approaches:

1. Fitting a single surrogate model on scalarized objectives. Knowles (2006, pp. 54-56) propose *ParEGO*, which employs the augemented Tchebycheff function as scalarization to ensure the Pareto front is explored sufficiently.

2. Fitting one surrogate per objective, then either

   2.1. using one acquisition function per surrogate to return a set of promising candidate query points or

2.2. using one overall acquisition function to aggregate the surrogates, such as *EHI* as proposed by Emmerich et al. (2006, pp. 8f), which maximises the expected improvement of hypervolume.

Lastly, there is also a multitude of multi-objective evolutionary algorithms (MOEA) to explore the Pareto front. A key challenge for MOEAs is determining the fitness of an individual, as there are now multiple objectives that need to be incorporated instead of one as in single-objective EA (SOEA). We will outline the popular NSGA-II (Nondominated Sorting Genetic Algorithm II), which improves upon its predecessor NSGA (Srinivas and Deb, 1994) by making it parameterless, ensuring elitism, and reducing the computational complexity of ranking of individuals (Deb et al., 2002, p. 182). NSGA-II uses all the regular operators reproduction, mutation, and crossover that are already known from SOEA. The difference is in the fitness function, which for NSGA-II comprises two parts: non-dominated sorting and crowding distance.

Non-dominated sorting, as described by Goldberg (1989, p. 201) and Deb et al. (2002, pp. 183f) is an iterative procedure to ensure elitist selection through ranking individuals by their fronts. First, it assigns all individuals of the Pareto front a rank of 0. It then temporarily removes these individuals from the population and repeats this procedure with an incremented rank number until no individuals are left.

The second part, *crowding distance* ensures sufficient diversity, i.e. exploration of the Pareto front. The algorithm assigns each individual a score depending on how "crowded" the area in objective space around it is. Crowding distance is computed as the mean side length of the cuboid spanned by its nearest neighbours as vertices in the objective space, where individuals without two neighbours along some dimension are assigned an infinitely high distance value. The less the crowding distance, the more an area is by other individuals. (Deb et al., 2002, p. 185)

NSGA-II then ranks individuals by their non-dominated sorting front ranks in ascending order and by their crowding distance in descending order as tie breaker. This ranking is then used for selecting the μ best individuals to keep for the next generation. Reproduction follows a binary tournament selection, where two individuals are randomly drawn from the population and the better one according to the ranking gets included in the reproduction pool for mutation and crossover.

Conceptually, the reasoning behind using non-dominated sorting to rank individuals is intuitive: the closer an individuals front is to the Pareto front, i.e. the smaller its front rank, the better it approaches the true Pareto front. The usage of crowding distance, on the other hand, is not as straightforward. To understand it, we recall the goal of MOO, which is not merely optimizing hypervolume, but to approximate the true Pareto front well. Considering the scenario of only having individuals of one area in the objective space included in the Pareto front estimate makes it very unstable: removing this area (i.e. the solutions from this area) would collapse the entire front estimate. Considering now the opposite, having multiple areas making up the Pareto front estimate, makes it stable: removing one area would not impact the estimate a lot, as it's still held up by other individuals. (Goldberg, 1989, p. 185, pp. 189-192)

3 EAGGA

EAGGA, as proposed by Schneider et al. (2023), is an NSGA-II inspired algorithm that optimizes performance along with three interpretability objectives. The three interpretability proxies are

- NF: the relative number of features used in the model
- NI: the relative number of pairwise interaction effects in the model
- NNM: the relative number of non-monotone feature effects in the model

The authors argue that a sparse model as indicated by a low NF score helps interpretability by reducing the number of e.g. ALE plots necessary in post-hoc analysis (Schneider et al., 2023, p. 540). Similarly, a low NI score reduces the interplay between features and thus the dimensionality of post-hoc analysis plots and a low NNM avoids complex non-monotone feature effects, which is often desirable in practical use-cases such as loan approval, where a higher income should translate to a higher chance of approval. Lastly, performance is measured by the area under the receiver operating characteristic curve (AUC), which trades-off the true against the false positives rate.
Schneider et al. (2023, pp. 540f) acknowledge that this set-up creates the complex "extended search space" $\breve{\boldsymbol{\Lambda}}$ to be optimized over, containing tuples $(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{I}_s, \boldsymbol{m}_{\boldsymbol{I}_s}) = \breve{\boldsymbol{\lambda}} \in \breve{\boldsymbol{\Lambda}}$ with

- $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ being the model's hyperparameter configuration
- $\boldsymbol{s} \in \{0,1\}^p$ being a binary vector denoting which features are used in the model
- $\boldsymbol{I}_s \in \{0,1\}^{p \times p}$ being a binary matrix denoting included pairwise interactions between features
- $\boldsymbol{m}_{\boldsymbol{I}_s} \in \{-1,0,1\}^p$ being the vector denoting the monotonicity constraint of a feature (-1 decreasing, 0 none, 1 increasing)

Consequently, the optimization problem over this space would be

$$\underset{\breve{\boldsymbol{\lambda}} \in \breve{\boldsymbol{\Lambda}}}{\arg\min} \left(GE(\mathcal{I}_{\breve{\boldsymbol{\lambda}}}, \mathcal{D}), NF(\hat{f}_{\mathcal{D},\breve{\boldsymbol{\lambda}}}), NI(\hat{f}_{\mathcal{D},\breve{\boldsymbol{\lambda}}}), NNM(\hat{f}_{\mathcal{D},\breve{\boldsymbol{\lambda}}}) \right) \tag{9}$$

The authors thus introduce the equivalence relation R "allowed to interact" on the set of features C. They further denote all features included in the model with C_s. This induces a group structure of equivalence classes, where features allowed to interact belong to the same class and share the same monotonicity attribute. Using this group structure, they define a much simpler "augmented serach space" $\tilde{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda} \times \mathcal{G}$ comprising of the cartesian product of the model hyperparameter space $\boldsymbol{\Lambda}$ and the group structure space $\mathcal{G}$. Each group structure $\boldsymbol{G} \in \mathcal{G}$ consists of g groups:

- $G_1 = C \setminus C_s$ being the set of excluded features
- $G_k = (E_k, M_{E_k}), \forall k = 2, ..., g$ being tuples with
  - $E_k \subseteq C_s$ being the set of features in group k, i.e. features allowed to interact with each other

– M_{E_k} being the monotonicity constraint of group k

To generate new offspring, they use a regular evolutionary algorithm on the model hyperparameters $\mathbf{\Lambda}$ and a grouping genetic algorithm (GGA) on the group structure space $\mathcal{G}$, with adapted mutation and crossover operators. The authors further introduce a special initialisation of the group structures to increase the sample-efficiency of their algorithm compared to random initialisation. (Schneider et al., 2023, pp. 541-543)

4 Extension to Neural Networks

EAGGA as implemented by Schneider et al. (2023) applies the optimizer to XGBoost, which proved to vastly outperform a union of competitor models with respect to dominated hypervolume and achieved comparable or better performance than ParEGO optimizing over the extended search space $\check{\mathbf{\Lambda}}$ (Schneider et al., 2023, pp. 543-545). Our work extends EAGGA to neural networks. We chose this model class for several reasons:

1. NNs are notoriously uninterpretable due to complex transformations of the feature space
2. Non-linear activation functions implicitly model interactions without direct control of which features interact with each other
3. There are no monotonicty guarantees in neural networks

The optimizer is implemented largely as described by Schneider et al. (2023) with only minor changes due to there being no straightforward transferability of some functions from R to Python or differences in model properties between XGBoost and neural nets.

4.1 Architectural Details

We implement the group structures as described by Schneider et al. (2023, p. 541) with an additional list encoding whether the signs of individual features should be inverted (-1) or not (1), as detected by the monotonicity detector. The features included in the group structure are then passed to a custom pytorch Dataset implementation, which outputs only the included features, multiplied with their individual sign. This allows the groups' monotonicity constraints to be encoded with only $\{0, 1\}$, as proposed by Schneider et al. (2023, p. 543). The entire group structure is then passed to the neural network, where the architecture is built accordingly.
Applying EAGGA to NNs requires defining a special architecture to realise interaction and monotonicity constraints. The neural network consists of a group of fully-connected sub-networks that are not connected amongst each other, as visualised in Figure 1. Each equivalence class in the group structure has its own sub-network, so that it is guaranteed that only the features within that group can interact with each other.
For the hidden layers, we use ReLU activations with dropout afterwards, after which a shared output layer transforms the concatenated output of the sub-networks to a probability using the Sigmoid activation function. We use AdamW with default parameters, optimizing a binary cross-entropy loss with Sigmoid implicit in the loss function for better numerical

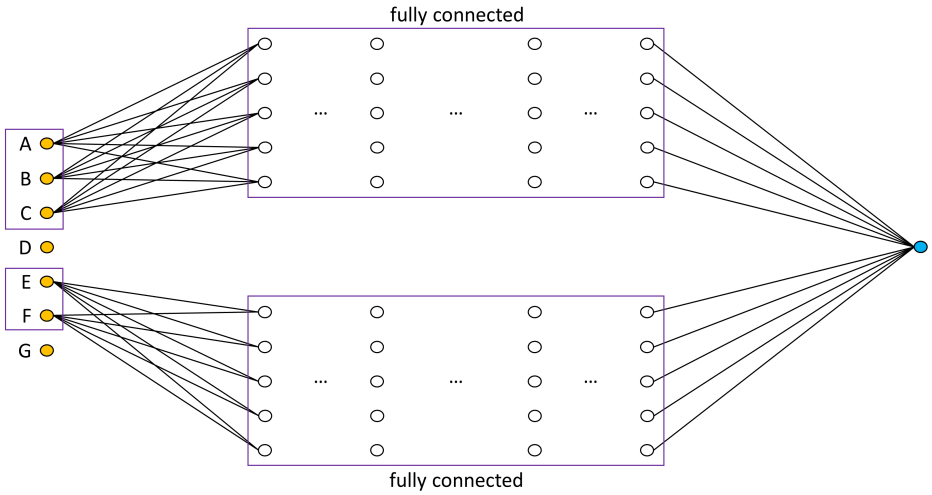

Figure 1: TODO: caption

stability. The network itself thus outputs only a "logit", which is the pre-activation output in pytorch.
Feature sparsity is achieved by only training on the included features that are passed to the network from the dataset.
Grouping the network into sub-networks enables restricting feature interactions to only those features that are included in the respective group. The max-operation in the ReLU induces a certain interaction effect on its inputs. It is important to note that this is a different kind of interaction than the multiplicative one known, for instance, from the linear model. Given a layer's input $\boldsymbol{x}$ and without loss of generality omitting the bias, the ReLU activation is defined as follows

$$\text{ReLU}(\boldsymbol{x}) = \max(0, \sum_{j=1}^{p} w_j x_j) \tag{10}$$

We see that the interaction is not of the form $x_j \cdot x_{j'}$ but rather of a form where the sum $w_j x_j + w_{j'} x_{j'}$ decides whether $x_j, x_{j'}$ pass through the activation or not. In other words, the interaction is given by the fact that

$$\max(0, w_j x_j + w_{j'} x_{j'}) \neq \max(0, w_j x_j) + \max(w_{j'} x_{j'}) \tag{11}$$

Finally, monotonicty is achieved by clipping the respective sub-network's weights to $[0, \infty)$ after each epoch. Whether a feature's effect is increasing or decreasing depends on the individual feature's sign encoded in the custom dataset implementation. Consider the vector form computation of a pre-activation hidden neuron $\boldsymbol{o}_{\text{in}}^{(l)} \in \mathbb{R}^{p^{(l)}}$ of layer l

$$\boldsymbol{o}_{\text{in}}^{(l)} = \boldsymbol{W} \boldsymbol{o}_{\text{out}}^{(l-1)} + \boldsymbol{b} \tag{12}$$

where $\boldsymbol{o}_{\text{out}}^{(l)}$ denotes the value of layer l after applying ReLU, $\boldsymbol{W} \in \mathbb{R}^{p^{(l)} \times p^{(l-1)}}$ refers to the weight matrix and $\boldsymbol{b} \in \mathbb{R}^{p^{(l)}}$ to the bias. A function f is monotonically increasing if and only if $\forall x_1 \leq x_2$ it holds that $f(x_1) \leq f(x_2)$ and thus $\boldsymbol{W}\boldsymbol{o}_{\text{out}}^{(l-1)} + \boldsymbol{b}$ is clearly only monotonically increasing if and only if $\boldsymbol{W} \in \mathbb{R}_{0+}^{p^{(l)} \times p^{(l-1)}}$, where $\mathbb{R}_{0+} = [0, \infty)$. Therefore, $\boldsymbol{o}_{\text{out}}^{(l)} = \text{ReLU}(\boldsymbol{o}_{\text{in}}^{(l)}) = \max(0, \boldsymbol{o}_{\text{in}}^{(l)})$ is a monotonically increasing function if and only if the weights are clipped as described. Bias clipping is not necessary as it is a constant additive term.

4.2 Overall Algorithm

Our NNs use the following hyperparameters initialised as described

- Total number of layers $\in \{3, ..., 10\}$, initialised from a truncated Geometric distribution with $\pi = 0.5$. This includes input and output layers, the number of hidden layers is thus $\in \{1, ..., 8\}$
- Number of nodes per hidden layer $\in \{3, ..., 20\}$, initialisation from a truncated Geometric distribution with $\pi = 0.5$
- Dropout probability $\in [0, 1]$, initialised from a truncated Gamma distribution with shape 2 and scale 0.15

For the group structure initialization we also employ feature, interaction, and monotonicity detectors.
The feature detector is as described by Schneider et al. (2023, p. 542), but instead of fitting ten trees and using the relative number of features as probability for the truncated Geometric distribution, we use a constant value of 0.5, as the sklearn DecisionTree examination is not straightforward. Further, in preliminary experiments we found the sampled number of features to occassionally be larger than the number of non-zero values in the normalised information gain filter. This can happen if the former is equal to the total number of features and one or more features are independent form the target. In these edge cases we use all features with non-zero filter-values instead.
The interaction detector similarly uses a constant probability of 0.5 to sample the number of interactions to be used instead of the relative number of interactions from ten decision trees, for the same reasoning as for the feature detector. Also, for the FAST algorithm, when computing the interaction scores we don't use the residual sum of squares but the mean accuracy, as the datasets all have binary targets and FAST thus fits a logistic instead of a linear regression.
The monotonicty detector is implemented as described by Schneider et al. (2023, p. 543), using the default decision tree hyperparameters from mlr3 in Python.
Both the models in the interaction and the monotonicity detector (FAST and an ensemble of decision trees, respectively) use 80% of the data for model estimation and the remaining 20% to compute their scores.

4.3 Experimental Setup

We evaluate our models just like Schneider et al. (2023, pp. 543f) by computing the dominated hypervolume along AUC, NF, NI, NNM with reference point $(0, 1, 1, 1)^T$. Internally,

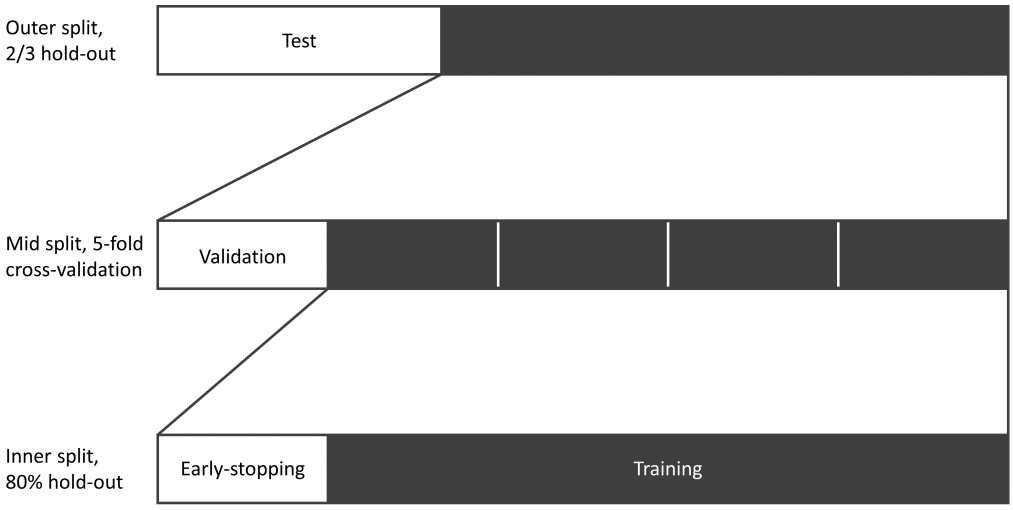

Figure 2: TODO: caption

this is implemented as minimising $1 - \text{AUC}$ and a reference point $(1, 1, 1, 1)^T$. For NF, we simply count the relative number of included features. NI sums the numbers of all possible pairwise interactions in each group $\binom{p_k}{2}$ and divides them by the number of all possible pairwise interactions on the entire dataset $\binom{p}{2}$. NNM counts the relative number of features in unconstrained equivalence classes.

Our dataset is split as visualised in Figure 2. We apply an outer holdout train-test split with 1/3 of the dataset being held for final testing, the 2/3 are used for 5-fold cross-validated training of each individual, as described by Schneider et al. (2023, p. 543). What is different for our implementation is that on each training portion of the CV split, we further apply a holdout split, where 80% are used for model training and 20% are reserved for early stopping.

We use a conservative early stopping stratetgy, where each fold is always trained for a minimum of 200 epochs, keeping track of the fold's model with the lowest loss. After 200 epochs, our criterion with a patience of 100 epochs is applied: now, if the current epoch's model loss is greater than the mean of the last 100 epochs' losses, training in this fold stops early and returns the model with the lowest loss over the entire fold's training period. If no early stopping is triggered, we train each fold for a maximum of 10 minutes and return the model with the lowest loss on the early stopping portion.

The model is then evaluated on the remaining portion from the CV split and after each generation, we keep the best $\mu = 100$ individuals and generate $\lambda = 10$ offspring for the next generation and repeat this for 8 hours.

The final evaluation of the individuals from the cross-validated Pareto set is then conducted on the held 1/3 of the dataset. Each individual is trained on the 2/3 training portion for the maximum number of epochs the individual required in the cross-validated training. Of-

tentimes, for cross-validation with early stopping, the final model is only trained on the mean or median number of epochs from CV. We decided specifically on the maximum after examining the loss over epochs in preliminary experiments, as plotted in Figure 4 for the final experimental setup. Those show that there is no uptick in losses on the early stopping portion indicating that training too long does not increase the validation error.
We trained in Sagemaker Notebooks on AWS and did initial training runs on ml.g4dn.xlarge[2] instances with CUDA. There was, however, no noticeable speed-up in model training compared to the much more economical ml.t3.medium[3] instances, after which we decided to run the final experimental setup on those.
In total, we run our models on the same datasets as Schneider et al. (2023, p. 544), except for the 2 largest ones "philippine" and "gina" (308 and 970 features, respectively), as those ran out of memory when initializing the interaction effects.
In rare cases (anecdotally once every 10 datasets), we observed our monotonicity detector generating a group monotonicty attribute of -1, which crashed the computation. For these cases, we implemented a function to import a previous generation of individuals to be able to pick up from where the training crashed and not train anew. This happened once for the dataset "madeline" after generation 9 and training for 5 hours 45 minutes, and hence we imported the generation and ran it for the remaining 2 hours 15 minutes.

5 Experimental Results and Discussion

- preliminary * our implementation of feature + interaction detectors likely tend to include - more features + interactions for high p datasets - less for low p datasets - remember: original paper samples # of features included from truncated geometric distribution, what is not mentioned is probability of this distribution is determined from fitting 10 trees + looking at relative # of features used (cf. their github repo /R/TunerEAGGA.R function get_n_selected_rpart()) -¿ mlr3 default decision tree max depth 30 * i.e. datasets with p ¡ 30 might use all features, which translates to trunc geom prob = 1 * vice-versa for p ¿¿ 30 datasets relative # features ¡ 0.5 (our trunc geom prob) * similar reasoning for pairwise interactions * individuals not in Pareto sets occassionally exhibit AUC ¡ 0.5 - this is not by mistake, usually AUC on binary target can simply be inverted (simply predict the opposite class) - for us this cannot easily be inverted due to monotonicity constraints (weights cannot simply be multiplied by -1 if constrained) - could happen e.g. due to AdamW and weight clipping (employed to enforce monotonicity) * AdamW uses momentum * possible scenario: large momentum would yield negative weights, but after epoch they're clipped to [0,infty) + training stops due to early stopping * weight clipping very imperfect way of enforcing monotonicity, but currently in pytorch unfortunately only way to implement this - Figure 3 * suggests comparable performance to XGBoost EAGGA * did not compare to unrestricted NN, would not be sensible, as NF, NI, NNM would be 1, hence hypervolume = 0, as values along 3 dimensions would be at reference point * as XGBoost EAGGA, NN EAGGA consistently outperforms union of competitors as evident from (Schneider et al., 2023, Figure 2) * unfortunately due to time constraints no own union comparison with NN instead of XGBoost - overview over pareto sets can be accessed on our github repo at

2. 2 vCPUs, 16GiB RAM, 1 NVIDIA T4, cf. https://aws.amazon.com/de/ec2/instance-types/g4/
3. 2 Intel Xeon 8000 vCPUs, 4GiB RAM, cf. https://aws.amazon.com/de/ec2/instance-types/t3/

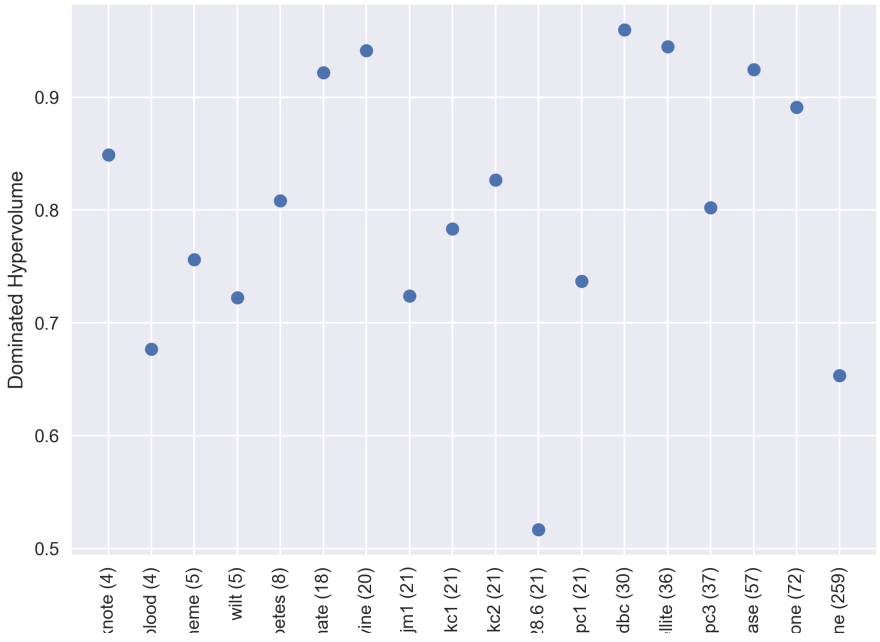

Figure 3: TODO: caption

/code/export/*.csv * NN HPs - total layers mostly 3-4, most commonly 3, goes as high as 7 (Satellite (36), diabetes (8)) - nodes per hidden layer mostly 3-6, goes as high as 12 (diabetes (8)) - p dropout goes as high as 0.7 (climate (18), spambase (57)), but mostly in the 0.1 to 0.3 range * group structures - great diversity in NF across datasets * phoneme (5) up to 1 * blood (4) up to 1 * banknote (4) up to 0.75 * diabetes (8) up to 0.5 * climate (18) up to 0.67 -¿ low p datasets in the benchmark tend to have higher NF - possible consequence of shorter evaluation time - low p datasets have much higher # generations - group structure space much more likely to be exhausted, i.e. more exploration of NN hps - NI, NNM consequently (bounded by NF, can never be more than max # for respective # included features) rather low * dhv contributions - measures contribution of an individual to the hypervolume, i.e. difference between hypervolume of entire pareto front vs hypervolume of pareto front without individual $\boldsymbol{\lambda}$: $\text{CON}_{\mathcal{P}}(\boldsymbol{\lambda}) = \text{HYP}(\mathcal{P}) - \text{HYP}(\mathcal{P} \setminus \{\boldsymbol{\lambda}\})$, where $\text{HYP}(\mathcal{P})$ denotes the hypervolume induced by the pareto set $\mathcal{P}$ (Bringmann and Friedrich, 2010, p. 384) - predominately low for fitted models - mostly highest for featureless learner predicting majority class -¿ sign of good exploration of pareto front + stable estimate, refer Section 2.4.3 - loss graphs Figure 4 * suggests models could have benefitted from longer training on some datasets, as some loss curves haven't converged when stopping criterion hit * crit was likely triggered by short-term spike, thus could potentially be resolved by comparing average of last k losses against average of ¿patience¡ losses prior to that to not be as exposed to short-term spikes in loss * on other datasets, graphs suggest earlier stop would have been totally fine as the networks have long converged, but longer training likely not an issue as loss graphs come from early stopping dataset portion, which is disjunct from training set -

dhv over generations Figure 5 * compute on val set, i.e. had 5 folds per individual - NF, NI, NNM always the same for each fold - but 5 different AUCs -¿ hypervolume of (mean(AUC 1, AUC 2, ..., AUC 5), NF, NI, NNM) * artifacts / drops along y likely due to inconsistent computation of Pareto front by third-party library - preliminary experiments on dummy pareto fronts with 10x4 metrics: noticed nds function returning different rankings for same front - returned ranking switched back and forth between two only slightly different options (only 1 or 2 indices were swapped) - not sure what caused this, as made same observation using another library that was planned as alternative - thus unfortunately not fixable for me * all but 3 datasets (2 of which only trained for 1 generation, aynway) show improvement of dominated hypervolume over generations * BUT: absolute as well as relative improvement almost negligible -¿ no large final dhv decrease would we just have evaluated the models gotten from the detectors - this was also the reason we didn't run EAGGA on philippine and gina simply without the detectors (as that's where they crashed) -¿ evidence suggested that detectors are vital to initial performance - original paper supports this assumption (Schneider et al., 2023, Fig. 4, p. 545)

6 Conclusion and Future Outlook

(Conclusion) - tabular data still difficult discipline for neural networks - research suggests heavy regularisation to improve performance on tabular data - EAGGA proved to be successful in making XGBoost more interpretable while keeping performance on par with unregularised XGBoost - we extend EAGGA to neural networks to see if we can utilise the regularisation induced by it to make NNs both interpretable and performant on tabular data - as consequence propose new architecture allowing to model equivalence relations of EAGGA - ... conforming to Zhang et al. (2021, chap. 2) taxonomy it fits as ..., ..., ... - found overall performance comparable to that of XGBoost fitted using EAGGA, which is a plus, but no outperformance

(Future Outlook) - MO BO on group structure space possible, perhaps via restricted BO?

References

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi:

10.1145/3620665.3640366. URL https://doi.org/10.1145/3620665.3640366.

Thanasis Antamis, Anastasis Drosou, Thanasis Vafeiadis, Alexandros Nizamis, Dimosthenis Ioannidis, and Dimitrios Tzovaras. Interpretability of deep neural networks: A review of methods, classification and hardware. *Neurocomputing*, 601:128204, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2024.128204. URL https://www.sciencedirect.com/science/article/pii/S0925231224009755.

J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8: 89497–89509, 2020.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6):7499–7519, June 2024. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3229161. URL http://dx.doi.org/10.1109/TNNLS.2022.3229161.

Karl Bringmann and Tobias Friedrich. An efficient algorithm for computing hypervolume contributions**. *Evol. Comput.*, 18(3):383–402, September 2010. ISSN 1063-6560.

Casper da Costa-Luis, Stephen Karl Larroque, Kyle Altendorf, Hadrien Mary, richardsheridan, Mikhail Korobov, Noam Yorav-Raphael, Ivan Ivanov, Marcel Bargull, Nishant Rodrigues, Shawn, Mikhail Dektyarev, Michał Górny, mjstevens777, Matthew D. Pagel, Martin Zugnoni, JC, CrazyPython, Charles Newey, Antony Lee, pgajdos, Todd, Staffan Malmgren, redbug312, Orivej Desh, Nikolay Nechaev, Mike Boyle, Max Nordlund, MapleCCC, and Jack McCracken. tqdm: A fast, extensible progress bar for python and cli, November 2024. URL https://doi.org/10.5281/zenodo.14231923.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning*, 2017. URL https://api.semanticscholar.org/CorpusID:11319376.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search. In Hutter et al. (2019), pages 69–86.

M.T.M. Emmerich, K.C. Giannakoglou, and B. Naujoks. Single- and multiobjective evolutionary optimization assisted by gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006. doi: 10.1109/TEVC.2005.859463.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Hutter et al. (2019), pages 3–38.

Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *Journal of Machine Learning Research*, 22(100):1–5, 2021. URL http://jmlr.org/papers/v22/19-920.html.

William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80122-4. URL https://www.sciencedirect.com/science/article/pii/S0893608005801224.

Peter I. Frazier. A tutorial on bayesian optimization, 2018. URL https://arxiv.org/abs/1807.02811.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989. ISBN 0201157675.

Nikolaus Hansen. The cma evolution strategy: A tutorial, 2023. URL https://arxiv.org/abs/1604.00772.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. URL https://arxiv.org/abs/1207.0580.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. ISSN 00401706. URL http://www.jstor.org/stable/1267351.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 23928–23941. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/c902b497eb972281fb5b4e206db38ee6-Paper.pdf.

Florian Karl, Tobias Pielok, Julia Moosbauer, Florian Pfisterer, Stefan Coors, Martin Binder, Lennart Schneider, Janek Thomas, Jakob Richter, Michel Lang, Eduardo C. Garrido-Merchán, Juergen Branke, and Bernd Bischl. Multi-objective hyperparameter optimization in machine learning—an overview. *ACM Trans. Evol. Learn. Optim.*, 3(4), December 2023. doi: 10.1145/3610536. URL https://doi.org/10.1145/3610536.

J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006. doi: 10.1109/TEVC.2005.851274.

Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL https://christophm.github.io/interpretable-ml-book.

The pandas development team. pandas-dev/pandas: Pandas, January 2023. URL https://doi.org/10.5281/zenodo.7549438.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jefferson A. Riera, Ricardo M. Lima, and Omar M. Knio. A review of hydrogen production and supply chain modeling and optimization. *International Journal of Hydrogen Energy*, 48(37):13731–13755, 2023. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2022.12.242. URL https://www.sciencedirect.com/science/article/pii/S0360319922060505.

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics, 2017. URL https://arxiv.org/abs/1606.02492.

Lennart Schneider, Bernd Bischl, and Janek Thomas. Multi-objective optimization of performance and interpretability of tabular supervised machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 538–547, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701191. doi: 10.1145/3583131.3590380. URL https://doi.org/10.1145/3583131.3590380.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. doi: 10.1109/JPROC.2015.2494218.

N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, September 1994. ISSN 1063-6560. doi: 10.1162/evco.1994.2.3.221. URL https://doi.org/10.1162/evco.1994.2.3.221.

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997. ISSN 1573-2916. doi: 10.1023/A:1008202821328. URL https://doi.org/10.1023/A:1008202821328.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 12 2018. ISSN 0035-9246. doi: 10.1111/j.2517-6161.1996.tb02080.x. URL https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014. ISSN 1931-0145. doi: 10.1145/2641190.2641198. URL https://doi.org/10.1145/2641190.2641198.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

J. Zach. Interpretability of deep neural networks. 2019. URL `https://api.semanticscholar.org/CorpusID:198979458`.

Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5): 726–742, 2021. doi: 10.1109/TETCI.2021.3100641.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017. URL `https://arxiv.org/abs/1611.01578`.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Appendix A. Software used

for implementation we used openml Vanschoren et al. (2014), Feurer et al. (2021), numpy Harris et al. (2020), pandas pandas development team (2023), Wes McKinney (2010), pytorch Ansel et al. (2024), scikit-learn Pedregosa et al. (2011), scipy Virtanen et al. (2020), pymoo Blank and Deb (2020), and tqdm da Costa-Luis et al. (2024)

Appendix B. Plots

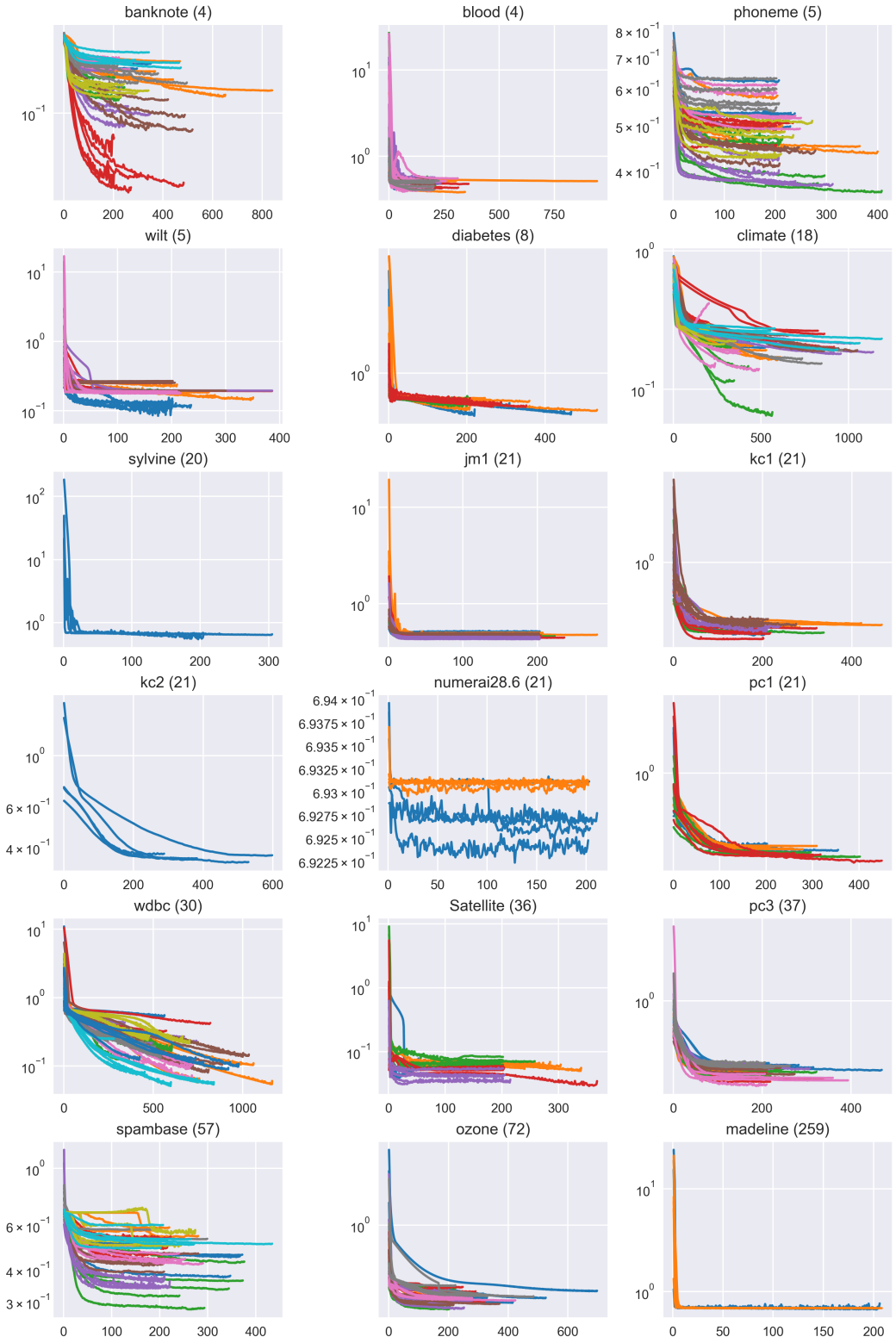

Figure 4: Pareto set loss of all datasets, evaluated on early stopping set. Same colours denote losses coming from folds of the same individual. x-axis portrait epochs, y-axis binary cross entropy loss.

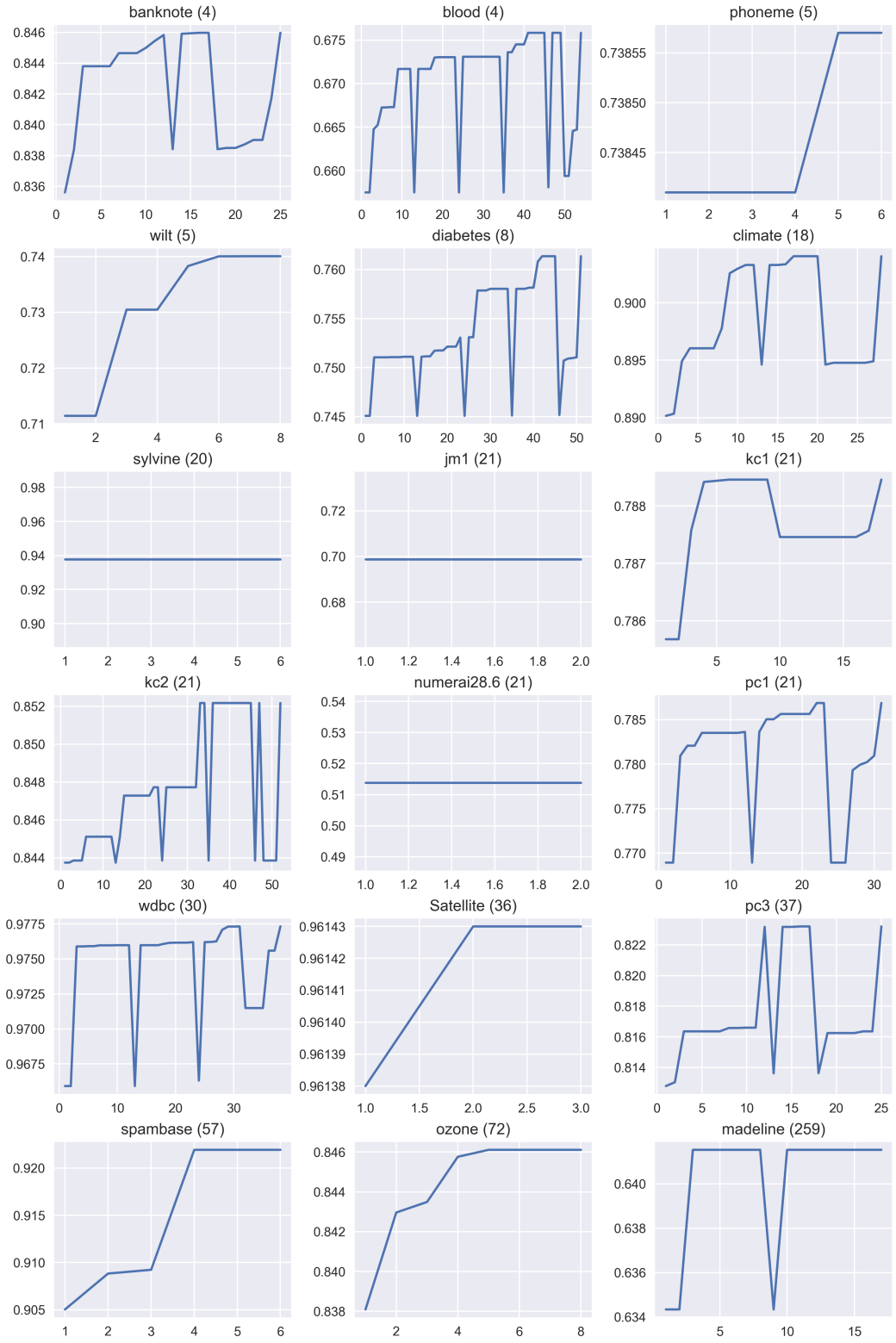

Figure 5: Dominated hypervolume over generations, evaluated on validation set. x-axis portrait generations, y-axis dominated hypervolume using mean AUC over folds.