Journal of Computational Science 71 (2023) 102054

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Neuroevolutionary representations for learning heterogeneous treatment effects[☆]

Michael C. Burkhart [a,*], Gabriel Ruiz [b]

[a] University of Cambridge, Cambridge, UK
[b] UCLA, Los Angeles, CA, USA

ARTICLE INFO

MSC:
62D20
68W50
68T30
68T20
68T07

Keywords:
Causal inference
Counterfactual inference
Heterogeneous treatment effects
Learning representations
Neuroevolutionary algorithms

ABSTRACT

Within the field of causal inference, we consider the problem of estimating heterogeneous treatment effects from data. We propose and validate a novel approach for learning feature representations to aid the estimation of the conditional average treatment effect or CATE. Our method focuses on an intermediate layer in a neural network trained to predict the outcome from the features. In contrast to previous approaches that encourage the distribution of representations to be treatment-invariant, we leverage a genetic algorithm that optimizes over representations useful for predicting the outcome to select those less useful for predicting the treatment. This allows us to retain information within the features useful for predicting the outcome even if that information may be related to treatment assignment. We validate our method on synthetic examples and illustrate its use on a real life dataset. This paper extends work previously presented at the International Conference on Computational Science in London (Burkhart and Ruiz, 2022).

1. Introduction

In this work, we engineer feature representations to aid the estimation of heterogeneous treatment effects. Specifically, we consider the following graphical model

$$\begin{tikzcd} & X \arrow[dl] \arrow[dr] & \\ W \arrow[rr] & & Y \end{tikzcd} \tag{1}$$

where $X \in \mathbb{R}^d$ denotes a vector of features, $W \in \{0,1\}$ represents a boolean-valued treatment, and $Y \in \mathbb{R}$ denotes a real-valued outcome. Sampling from a distribution P respecting the graph (1) may be viewed as a multistep process that first generates a vector of features, then assigns a treatment that may depend on those features, and finally realizes an outcome given both the features and the treatment. Within the Neyman–Rubin potential outcomes framework [1,2], we let $Y(1)$ denote the potential outcome if W were set to 1 and $Y(0)$ denote the potential outcome if W were set to 0. We wish to estimate the conditional average treatment effect (CATE) defined to be the expected difference between the two potential outcomes conditioned on the features, as a function of the features, namely

$$\tau(x) = \mathbb{E}[Y(1) - Y(0) \mid X = x]. \tag{2}$$

We impose standard assumptions that the treatment assignment is unconfounded, meaning that it is independent of the potential outcomes after conditioning on the features, i.e.

$$\{Y(0), Y(1)\} \perp\!\!\!\perp W \mid X,$$

and random in the sense that assignment is non-deterministic for any realized vector of features, i.e.

$$\epsilon < P(W = 1 \mid X = x) < 1 - \epsilon$$

for some $\epsilon > 0$ and all $x \in \mathbb{R}^d$ in the support of X. These assumptions are jointly known as ***strong ignorability*** [3] and prove sufficient for the CATE to be identifiable. Under them, there exist well-established methods to estimate the CATE from samples $(X_i, W_i, Y_i) \sim^{\text{i.i.d.}} P$ for $i = 1, \dots, n$ (see Section 2.1 for a discussion) that then allow us to predict the impact of an intervention for novel examples using only their individual features [4]. Viewing these approaches as black box estimators, we seek a mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^m$ such that the estimate of the CATE learned on the transformed training data $(\Phi(X_i), W_i, Y_i)$ is more accurate than an estimate learned on the original samples (X_i, W_i, Y_i). Our inspiration stems from Nie and Wager's recent work [5] that demonstrates clear benefits from de-correlating the treatment propensity

$$e(x) = P(W = 1 \mid X = x) \tag{3}$$

[☆] M.B. and G.R. gratefully acknowledge support from Adobe Inc. (San Jose, USA) during the development of this work.
[*] Corresponding author.
E-mail addresses: mcb93@cam.ac.uk (M.C. Burkhart), ruizg@ucla.edu (G. Ruiz).

https://doi.org/10.1016/j.jocs.2023.102054
Received 9 October 2022; Received in revised form 22 February 2023; Accepted 7 May 2023
Available online 15 May 2023

and the conditional mean outcome

$$m(x) = \mathbb{E}[Y \mid X = x] \tag{4}$$

prior to inferring the CATE.[1] To this end, we desire a function Φ yielding a corresponding representation $\Phi(X)$ such that

1. $\Phi(X)$ is as useful as X for estimating Y, and
2. among such representations, $\Phi(X)$ is least useful for estimating W.

In this way, we hope to produce a representation $\Phi(X)$ that retains all information relevant for predicting the outcome, but is less related to treatment assignment. *We propose learning Φ as an intermediate layer in a neural network estimating a functional relationship of Y given X. We apply a genetic algorithm [6] to a population of such mappings to evolve and select one for which the associated representation $\Phi(X)$ is least useful for approximating W.* Feature representations are commonly used in machine learning to aid the training of supervised models [7] and have been previously demonstrated to aid in causal modeling. Johansson, et al. [8,9] viewed counterfactual inference on observational data as a covariate shift problem and learned neural network-based representations designed to produce similar empirical distributions among the treatment and control populations, namely $\{\Phi(X_i)\}_{W_i=1}$ and $\{\Phi(X_i)\}_{W_i=0}$. Li & Fu [10] and Yao, et al. [11] developed representations in a related vein designed to preserve local similarity. We generally agree with Zhang et al.'s [12] recent argument that domain invariance often removes too much information from the features for causal inference.[2] *In contrast to most previous approaches, we develop a feature representation that attempts to preserve information useful for predicting the treatment effect if it is also useful for predicting the outcome.*

1.1. Outline

This article extends our previous conference publication [14] and patent application [15] with the introduction of a tunable fitness function in (10), a comparison of different activation functions (elu/relu/tanh) in Tables 1 and 2 to address reviewer feedback, and the inclusion of details previously suppressed due to page limitations. In the next section, we describe methods for learning the CATE from observational data and introduce genetic algorithms. In Section 3, we describe our methodology in full. We then validate our method on artificial data in Section 4 and on a publicly available experimental dataset in Section 5 before concluding in Section 6.

2. Related work

In the first part of this section, we discuss standard methods for learning the CATE function from data. We will subsequently use these to test our proposed feature engineering methods in Section 4. In the second part, we briefly outline evolutionary algorithms for training neural networks, commonly called neuroevolutionary methods.

2.1. Meta-learners

We adopt the standard assumptions of unconfoundedness and the random assignment of treatment effects that together constitute strong ignorability. Given i.i.d. samples from a distribution P respecting (1) and these assumptions, there exist numerous meta-learning approaches that leverage an arbitrary regression framework (e.g., random forests, neural networks, linear regression models, etc.) to estimate the CATE that we now describe.

[1] In their own words, "any good heterogeneous treatment effect estimator needs to achieve two goals: first, it should eliminate spurious effects by controlling for correlations between $e(X)$ and $m(X)$; second, it should accurately express $\tau(X)$." They then propose an approach that "cleanly separates these two tasks."

[2] Zhao et al. [13] make this argument in a more general setting.

2.1.1. S-learner

The S-learner (single-learner) uses a standard supervised learner (regression model) to estimate

$$\mu(x, w) = \mathbb{E}[Y \mid X = x, W = w]$$

from observation data and then predicts

$$\hat{\tau}_S(x) = \hat{\mu}(x, 1) - \hat{\mu}(x, 0)$$

where we use the standard hat notation to denote estimated versions of the underlying functions.

2.1.2. T-learner

The T-learner (two-learner) estimates

$$\mu_1(x) = \mathbb{E}[Y(1) \mid X = x],$$
$$\mu_0(x) = \mathbb{E}[Y(0) \mid X = x]$$

from observed treatment data $\{(X_i, Y_i)\}_{W_i=1}$ and control data $\{(X_i, Y_i)\}_{W_i=0}$, respectively, and then predicts

$$\hat{\tau}_T(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x).$$

2.1.3. X-learner

The X-learner [16] estimates μ_1 and μ_0 as in the T-learner, and then learns

$$\tau_1(x) = \mathbb{E}[Y(1) - \hat{\mu}_0(X) \mid X = x],$$
$$\tau_0(x) = \mathbb{E}[\hat{\mu}_1(X) - Y(0) \mid X = x]$$

on the observed treatment and control data, respectively. The X-learner then predicts

$$\hat{\tau}_X(x) = g(x)\hat{\tau}_0(x) + (1 - g(x))\hat{\tau}_1(x)$$

where $g : \mathbb{R}^d \to [0, 1]$ is a weight function. The creators of the X-learner remark that the treatment propensity function (3) often works well for g, as do the constant functions 1 and 0. In our implementation, we set $g(x) \equiv 1/2$.

Concerning this method, we note that it is also possible to directly estimate τ from

$$\{(X_i, Y_i - \hat{\mu}_0(X_i))\}_{W_i=1} \cup \{(X_i, \hat{\mu}_1(X_i) - Y_i)\}_{W_i=0}$$

or, using $\hat{\mu}(x, w)$ from the S-learner approach, with

$$\{(X_i, Y_i - \hat{\mu}(X_i, 0))\}_{W_i=1} \cup \{(X_i, \hat{\mu}(X_i, 1) - Y_i)\}_{W_i=0}.$$

We find that these alternate approaches work well in practice and obviate the need to estimate or fix g.

2.1.4. R-learner

The R-learner [5] leverages Robinson's decomposition [17] that led to Robin's reformulation [18] of the CATE function as the solution to the optimization problem

$$\tau(\cdot) = \arg\min_{\tau} \{\mathbb{E}_{(X,W,Y)\sim P}\mathcal{L}(X, W, Y)\} \tag{5}$$

where

$$\mathcal{L}(x, w, y) = \left|\big(y - m(x)\big) - \big(w - e(x)\big)\tau(x)\right|^2$$

for $m(\cdot)$ and $e(\cdot)$ as defined in (3) and (4), respectively. In practice, a regularized, empirical version of (5) is minimized via a two-step process: (I) cross-validated estimates $\hat{m}$ and $\hat{e}$ are obtained for m and e, respectively, and then (II) the empirical loss is evaluated using folds of the data not used for estimating $\hat{m}$ and $\hat{e}$, and then minimized. The authors Nie & Wager note that the structure of the loss function eliminates correlations between m and e while allowing one to separately specify the form of τ through the choice of optimization method. In this paper, the only R-learner we use is the causal forest as implemented with generalized random forests [19] using the default options, including honest splitting [20].

Table 1
We report the average and standard deviation of the mean squared error over 100 independent trials run using setup A. To generate the feature maps (for entries not corresponding to the initial, untransformed features), Algorithm 1 was run with parameters: cohort size $c = 4$, progenitors $\ell = 2$, number of cohorts $g = 5$, representation dimensionality $m = 20$, and fitness function parameter $k = 10$. Values for the activation function $a(\cdot)$ and tuning parameter $\lambda \geq 0$ are reported on each line.

Causal learner	Features	avg.	std.	p-value	Causal learner	Features	avg.	std.	p-value
Causal forest	Initial	**0.194**	0.146	–	T-L. w/ LGBM	Initial	**0.745**	0.294	–
	No fitness	**0.137**	0.108	$6.0 \cdot 10^{-13}$		No fitness	**0.565**	0.226	$1.1 \cdot 10^{-08}$
	elu ($\lambda = 0$)	**0.134**	0.105	$1.9 \cdot 10^{-13}$		elu ($\lambda = 0$)	**0.562**	0.200	$3.6 \cdot 10^{-08}$
	relu ($\lambda = 0$)	**0.150**	0.110	$5.3 \cdot 10^{-08}$		relu ($\lambda = 0$)	**0.448**	0.215	$7.6 \cdot 10^{-16}$
	tanh ($\lambda = 0$)	**0.141**	0.106	$1.4 \cdot 10^{-11}$		tanh ($\lambda = 0$)	**0.549**	0.197	$3.7 \cdot 10^{-09}$
	tanh ($\lambda = 1$)	**0.140**	0.107	$9.5 \cdot 10^{-11}$		tanh ($\lambda = 1$)	**0.570**	0.223	$1.4 \cdot 10^{-07}$
	tanh ($\lambda = 10$)	**0.132**	0.101	$9.9 \cdot 10^{-14}$		tanh ($\lambda = 10$)	**0.557**	0.221	$5.4 \cdot 10^{-09}$
	tanh ($\lambda = 100$)	**0.129**	0.101	$2.3 \cdot 10^{-14}$		tanh ($\lambda = 100$)	**0.539**	0.209	$1.3 \cdot 10^{-09}$
S-L. w/ LGBM	Initial	**0.133**	0.071	–	T-L. w/ Ridge	Initial	**0.815**	0.334	–
	No fitness	**0.149**	0.079	$2.5 \cdot 10^{-02}$		No fitness	**0.383**	0.200	$1.5 \cdot 10^{-30}$
	elu ($\lambda = 0$)	**0.145**	0.067	$7.5 \cdot 10^{-02}$		elu ($\lambda = 0$)	**0.355**	0.199	$1.9 \cdot 10^{-29}$
	relu ($\lambda = 0$)	**0.190**	0.132	$8.0 \cdot 10^{-06}$		relu ($\lambda = 0$)	**0.278**	0.422	$4.3 \cdot 10^{-18}$
	tanh ($\lambda = 0$)	**0.137**	0.070	$5.3 \cdot 10^{-01}$		tanh ($\lambda = 0$)	**0.354**	0.194	$6.5 \cdot 10^{-31}$
	tanh ($\lambda = 1$)	**0.141**	0.074	$1.6 \cdot 10^{-01}$		tanh ($\lambda = 1$)	**0.371**	0.212	$2.0 \cdot 10^{-31}$
	tanh ($\lambda = 10$)	**0.145**	0.085	$1.3 \cdot 10^{-01}$		tanh ($\lambda = 10$)	**0.360**	0.208	$2.6 \cdot 10^{-34}$
	tanh ($\lambda = 100$)	**0.140**	0.061	$2.2 \cdot 10^{-01}$		tanh ($\lambda = 100$)	**0.341**	0.199	$1.7 \cdot 10^{-33}$
S-L. w/ Ridge	Initial	**0.099**	0.073	–	X-L. w/ LGBM	Initial	**0.453**	0.189	–
	No fitness	**0.089**	0.070	$4.6 \cdot 10^{-05}$		No fitness	**0.350**	0.167	$1.6 \cdot 10^{-07}$
	elu ($\lambda = 0$)	**0.087**	0.060	$2.7 \cdot 10^{-04}$		elu ($\lambda = 0$)	**0.351**	0.150	$2.9 \cdot 10^{-07}$
	relu ($\lambda = 0$),	**0.098**	0.078	$8.0 \cdot 10^{-01}$		relu ($\lambda = 0$)	**0.317**	0.175	$3.4 \cdot 10^{-09}$
	tanh ($\lambda = 0$)	**0.087**	0.061	$1.5 \cdot 10^{-04}$		tanh ($\lambda = 0$)	**0.352**	0.149	$3.1 \cdot 10^{-07}$
	tanh ($\lambda = 1$)	**0.087**	0.064	$5.5 \cdot 10^{-06}$		tanh ($\lambda = 1$)	**0.357**	0.158	$1.3 \cdot 10^{-06}$
	tanh ($\lambda = 10$)	**0.086**	0.062	$9.3 \cdot 10^{-06}$		tanh ($\lambda = 10$)	**0.355**	0.190	$3.3 \cdot 10^{-06}$
	tanh ($\lambda = 100$)	**0.085**	0.063	$1.4 \cdot 10^{-08}$		tanh ($\lambda = 100$)	**0.335**	0.146	$2.6 \cdot 10^{-09}$
					X-L. w/ Ridge	Initial	**0.692**	0.287	–
						No fitness	**0.336**	0.188	$3.3 \cdot 10^{-29}$
						elu ($\lambda = 0$)	**0.308**	0.181	$4. \cdot 10^{-29}$
						relu ($\lambda = 0$)	**0.257**	0.427	$1.6 \cdot 10^{-14}$
						tanh ($\lambda = 0$)	**0.309**	0.170	$3.2 \cdot 10^{-31}$
						tanh ($\lambda = 1$)	**0.324**	0.192	$2.0 \cdot 10^{-30}$
						tanh ($\lambda = 10$)	**0.310**	0.182	$3.8 \cdot 10^{-34}$
						tanh ($\lambda = 100$)	**0.304**	0.211	$4.6 \cdot 10^{-33}$

Table 2
We report the average and standard deviation of the MSE taken over 100 independent trials run using setup C. Feature maps were again generated using Algorithm 1 with $c = 4$, $\ell = 2$, $g = 2$, $m = 20$, and $k = 10$, and values for $a(\cdot)$ and λ as reported in the table.

Causal learner	Features	avg.	std.	p-value	Causal learner	Features	avg.	std.	p-value
Causal forest	Initial	**0.038**	0.040	–	T-L. w/ LGBM	Initial	**0.575**	0.140	–
	No fitness	**0.028**	0.027	$1.5 \cdot 10^{-04}$		No fitness	**0.541**	0.116	$3.8 \cdot 10^{-02}$
	elu ($\lambda = 0$)	**0.028**	0.026	$2.2 \cdot 10^{-04}$		elu ($\lambda = 0$)	**0.542**	0.124	$5.6 \cdot 10^{-02}$
	relu ($\lambda = 0$)	**0.031**	0.027	$2.0 \cdot 10^{-02}$		relu ($\lambda = 0$)	**0.494**	0.122	$1.1 \cdot 10^{-05}$
	tanh ($\lambda = 0$)	**0.029**	0.028	$2.1 \cdot 10^{-04}$		tanh ($\lambda = 0$)	**0.551**	0.120	$1.7 \cdot 10^{-01}$
	tanh ($\lambda = 1$)	**0.030**	0.027	$1.7 \cdot 10^{-03}$		tanh ($\lambda = 1$)	**0.550**	0.137	$1.4 \cdot 10^{-01}$
	tanh ($\lambda = 10$)	**0.029**	0.027	$8.7 \cdot 10^{-04}$		tanh ($\lambda = 10$)	**0.538**	0.142	$2.2 \cdot 10^{-02}$
	tanh ($\lambda = 100$)	**0.031**	0.024	$9.5 \cdot 10^{-03}$		tanh ($\lambda = 100$)	**0.550**	0.138	$1.5 \cdot 10^{-01}$
S-L. w/ LGBM	Initial	**0.225**	0.068	–	T-L. w/ Ridge	Initial	**0.181**	0.074	–
	No fitness	**0.206**	0.058	$9.0 \cdot 10^{-03}$		No fitness	**0.117**	0.074	$5.4 \cdot 10^{-22}$
	elu ($\lambda = 0$)	**0.206**	0.062	$9.6 \cdot 10^{-03}$		elu ($\lambda = 0$)	**0.121**	0.063	$2.1 \cdot 10^{-25}$
	relu ($\lambda = 0$)	**0.204**	0.059	$3.3 \cdot 10^{-03}$		relu ($\lambda = 0$)	**0.080**	0.045	$2.6 \cdot 10^{-29}$
	tanh ($\lambda = 0$)	**0.208**	0.063	$1.2 \cdot 10^{-02}$		tanh ($\lambda = 0$)	**0.133**	0.076	$7.2 \cdot 10^{-18}$
	tanh ($\lambda = 1$)	**0.214**	0.070	$1.5 \cdot 10^{-01}$		tanh ($\lambda = 1$)	**0.114**	0.067	$4.0 \cdot 10^{-31}$
	tanh ($\lambda = 10$)	**0.206**	0.059	$1.3 \cdot 10^{-02}$		tanh ($\lambda = 10$)	**0.116**	0.062	$3.9 \cdot 10^{-25}$
	tanh ($\lambda = 100$)	**0.209**	0.058	$3.4 \cdot 10^{-02}$		tanh ($\lambda = 100$)	**0.113**	0.065	$1.9 \cdot 10^{-29}$
S-L. w/ Ridge	Initial	**0.018**	0.023	–	X-L. w/ LGBM	Initial	**0.324**	0.088	–
	No fitness	**0.018**	0.024	$7.5 \cdot 10^{-02}$		No fitness	**0.309**	0.085	$1.5 \cdot 10^{-01}$
	elu ($\lambda = 0$)	**0.018**	0.023	$3.2 \cdot 10^{-01}$		elu ($\lambda = 0$)	**0.292**	0.079	$2.3 \cdot 10^{-03}$
	relu ($\lambda = 0$)	**0.020**	0.025	$2.3 \cdot 10^{-03}$		relu ($\lambda = 0$)	**0.277**	0.083	$2.3 \cdot 10^{-05}$
	tanh ($\lambda = 0$)	**0.017**	0.023	$5.6 \cdot 10^{-01}$		tanh ($\lambda = 0$)	**0.322**	0.088	$8.8 \cdot 10^{-01}$
	tanh ($\lambda = 1$)	**0.018**	0.024	$5.8 \cdot 10^{-02}$		tanh ($\lambda = 1$)	**0.311**	0.093	$2.7 \cdot 10^{-01}$
	tanh ($\lambda = 10$)	**0.018**	0.023	$6.6 \cdot 10^{-01}$		tanh ($\lambda = 10$)	**0.301**	0.097	$3.8 \cdot 10^{-02}$
	tanh ($\lambda = 100$)	**0.018**	0.024	$3.4 \cdot 10^{-03}$		tanh ($\lambda = 100$)	**0.311**	0.089	$2.3 \cdot 10^{-01}$
					X-L. w/ Ridge	Initial	**0.168**	0.069	–
						No fitness	**0.103**	0.068	$3.7 \cdot 10^{-25}$
						elu ($\lambda = 0$)	**0.108**	0.061	$1.9 \cdot 10^{-27}$
						relu ($\lambda = 0$)	**0.067**	0.041	$3. \cdot 10^{-32}$
						tanh ($\lambda = 0$)	**0.119**	0.072	$3.4 \cdot 10^{-19}$
						tanh ($\lambda = 1$)	**0.101**	0.063	$1.2 \cdot 10^{-33}$
						tanh ($\lambda = 10$)	**0.103**	0.060	$1.0 \cdot 10^{-27}$
						tanh ($\lambda = 100$)	**0.099**	0.063	$1.0 \cdot 10^{-31}$

2.2. Genetic and neuroevolutionary algorithms

Holland introduced genetic algorithms [6] as a nature-inspired approach to optimization. Generally speaking, these algorithms produce successive generations of candidate solutions. New generations are formed by selecting the fittest members from the previous generation and performing cross-over and/or mutation operations on them to produce offspring. Evolutionary algorithms encompass extensions and generalizations to this approach including memetic algorithms [21] that perform local refinements, genetic programming [22] that acts on programs represented as trees, and evolutionary programming [23] and strategies [24,25] that operate on more general representations. When such methods are applied specifically to the design and training of neural networks, they are commonly known as neuroevolutionary algorithms. See Stanley et al. [26] for a comprehensive survey. In the next section, we describe a specific neuroevolutionary strategy for feature engineering. As opposed to other strategies for global optimization, genetic algorithms are readily parallelized and allow us to generate new candidate solutions in a logical way from two good existing solutions.

3. Methodology

In this section, we describe how we form our feature mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^m$. To generate a single candidate solution, we train a shallow neural network to predict Y from X and extract an intermediate layer of this network. Each candidate map created in this way should yield a representation as functionally useful for predicting Y as X is. We then iteratively evolve cohorts of parameter sets for such maps to create a representation that carries the least amount of useful information for predicting the treatment assignment W.

3.1. Candidate solutions

We consider neural networks $f_\Theta : \mathbb{R}^d \to \mathbb{R}$ of the form

$$f_\Theta(x) = M_2 \cdot a(M_1 \cdot x + b_1) + b_2 \tag{6}$$

where $M_1 \in \mathbb{R}^{m\times d}$, $M_2 \in \mathbb{R}^{1\times m}$ are real-valued matrices (weights), $b_1 \in \mathbb{R}^m$ and $b_2 \in \mathbb{R}^1$ are vectors (biases), and $a(\cdot)$ is a nonlinear activation function applied component-wise. We let $\Theta = (M_1, M_2, b_1, b_2)$ denote the parameters for f_Θ. Though f_Θ is decidedly not a deep neural network, we note that, as a neural network with a single hidden layer, it remains a universal function approximator in the sense of Hornik et al. [27]. Optimizing the network (6) in order to best predict Y from X seeks the solution

$$\Theta_* = \arg\min_\Theta \mathbb{E}\left|Y - f_\Theta(X)\right|^2 . \tag{7}$$

Given parameters Θ for the network (6), we let $\Phi_\Theta : \mathbb{R}^d \to \mathbb{R}^m$ given by

$$\Phi_\Theta(x) = a(M_1 \cdot x + b_1) \tag{8}$$

denote the output of the hidden layer. We restrict to candidate feature mappings of this form. As these mappings are completely characterized by their associated parameters, we define a fitness function and evolutionary algorithm directly in terms of parameter sets Θ in the following subsections.

3.2. Fitness function

For parameters Θ near the optimum (7), $\Phi_\Theta(X)$ should be approximately as useful as X for learning a functional relationship with Y. However, for some values of Θ, the mapped features $\Phi_\Theta(X)$ may carry information useful for predicting W, and for this reason we consider a network $g_{\Psi,\Theta} : \mathbb{R}^d \to [0,1]$ given by

$$g_{\Psi,\Theta}(x) = \sigma(M_4 \cdot \alpha(M_3 \cdot \Phi_\Theta(x) + b_3) + b_4) \tag{9}$$

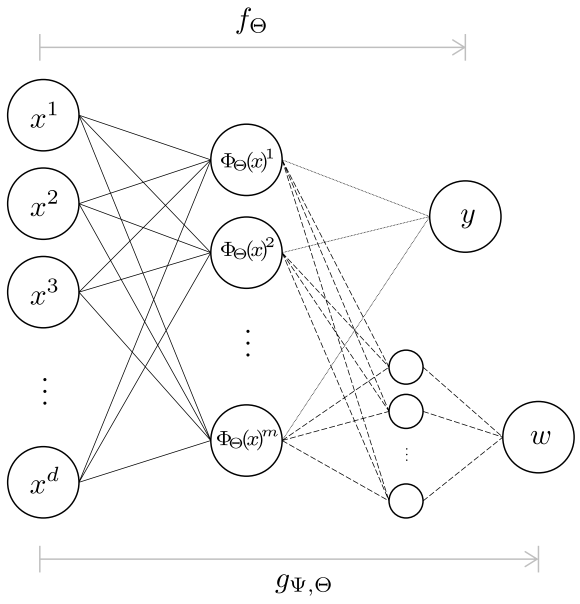


Fig. 1. We use superscripts to denote the coordinates of vectors, so that $x = (x^1, x^2, \ldots, x^d)$. The arrows connecting the x^i to the $\Phi_\Theta(x)^j$ represent the map Φ_Θ as in (8); these, in addition to the ones joining the $\Phi_\Theta(x)^j$ to y, represent f_Θ given explicitly in (6); the original arrows along with the dashed arrows on paths from $\Phi_\Theta(x)$ to w represent $g_{\Psi,\Theta}$ as in (9).

where $M_3 \in \mathbb{R}^{k\times m}$ and $M_4 \in \mathbb{R}^{1\times k}$ are weights, $b_3 \in \mathbb{R}^k$ and $b_4 \in \mathbb{R}$ are biases, α is a nonlinear activation function applied component-wise, and $\sigma(x) = (1 + \exp(-x))^{-1}$ denotes the sigmoidal activation function. In this case, $\Psi = (M_3, M_4, b_3, b_4)$ denotes the collection of learnable parameters. For a tunable hyperparameter $\lambda \geq 0$, we define the fitness of a parameter set Θ to be

$$\mu_\lambda(\Theta) = \min_\Psi H(W, \text{Bernoulli}(g_{\Psi,\Theta}(X))) - \lambda \cdot \mathbb{E}\left|Y - f_\Theta(X)\right|^2 , \tag{10}$$

where $H(V_1, V_2)$ denotes the cross-entropy between the random variables V_1 and V_2. In this way, we express a preference for representations $\Phi_\Theta(X)$ that are less useful for predicting W and (when $\lambda \gneq 0$) penalize any that lost information for predicting Y. For a schematic of these architectures, please see Fig. 1.

3.3. Evolutionary algorithm

We now describe a method to generate and evolve a cohort of candidate parameter sets Θ intended to seek a parameter set Θ_* such that $\Phi_{\Theta*}(X)$ is nearly as useful for predicting Y as X is and, among such representations, $\Phi_{\Theta*}(X)$ is least useful for predicting W.

Given training data $(X_i, W_i, Y_i) \overset{\text{i.i.d.}}{\sim} P$ for $1 \leq i \leq n$, we first partition the data into training and validation sets. We form an initial cohort of c candidates independently as follows. For $1 \leq j \leq c$, we randomly instantiate Θ_j using Glorot normal initialization [28] for the weights and zeros for the biases and then apply batch-based gradient descent on the training set to seek the solution to (7). In particular, we use the Adam optimizer [29] that maintains parameter-specific learning rates [30, cf. AdaGrad] and allows these rates to sometimes increase [31, cf. Adadelta] by adapting them using the first two moments from recent gradient updates. We use Tikhonov regularization [32] for the weights and apply a dropout layer after the activation function $a(\cdot)$ to prevent overfitting. We experimented with hyperbolic tangent and rectified [33] and exponential [34] linear unit activation functions for a in Φ_Θ. We use hyperbolic tangent for α in $g_{\Psi,\Theta}$.

For each constituent Θ_j in the cohort, we then initialize and train a network g_{Ψ,Θ_j} as in (9) to seek $\Psi_j = \arg\min_\Psi H(W, \text{Bernoulli}(g_{\Psi,\Theta_j}(X)))$ on the training set and then evaluate

$$H(W, \text{Bernoulli}(g_{\Psi_j,\Theta_j}(X))) - \lambda \cdot \mathbb{E}\left|Y - f_{\Theta_j}(X)\right|^2$$

empirically on the validation set to estimate $\mu_\lambda(\Theta_j)$. We then use the ℓ fittest members of the current cohort to form a new cohort as follows. For each of the $\binom{\ell}{2}$ pairings, we apply Montana and Davis's node-based crossover [35] method to the parameters M_1 and b_1 that we use to form Φ. This amounts to forming a new Φ by randomly selecting one of the two parents and using that parent's mapping for each coordinate. Thus, the new M_1 and b_1 are selected in a row-wise manner from the corresponding rows of the parents, and then the new M_2 and b_2 are randomly initialized and a few steps of optimization are performed to form the offspring candidate. The next generation then consists of the best performing candidate from the previous generation, $\binom{\ell}{2}$ candidates formed by crossing the best ℓ candidates of the previous generation, and $c - 1 - \binom{\ell}{2}$ entirely new candidates generated from scratch.

We summarize our approach using pseudo-code in Algorithm 1.

```
Data: training $\mathcal{T} = \{(X_i, W_i, Y_i)\}_{i\in\mathcal{T}}$ and validation
    $\mathcal{V} = \{(X_i, W_i, Y_i)\}_{i\in V}$ datasets drawn i.i.d. from $P$
    respecting the graphical model (1);
hyperparameters: $c$ cohort size, $\ell$ number of members involved
in forming the next generation, $g$ number of generations,
$m$ dimensionality of the latent representation, $k$ dimensionality
of the final hidden layer in (9), choice of activation function
$a(\cdot)$ for (8), $\lambda \geq 0$ weighting parameter for the fitness function
Result: parameterized function $\Phi_{\Theta_*} : \mathbb{R}^d \to \mathbb{R}^m$ such that
    models for the CATE learned using the transformed
    training data $\{(\Phi(X_i), W_i, Y_i)\}_{i\in\mathcal{T}}$ perform better than
    those learned on the original dataset $\mathcal{T}$
for $j = 1, \dots, c$ do
    Optimize a parameter set $\Theta_j$ to seek (7) on training batches
      from a random initialization;
end
Form the first generation $\mathcal{G}_1 = \{\Theta_j\}_{j=1,\dots,c}$;
for $t = 2, \dots, g$ do #form the next generation
    Initialize new generation $\mathcal{G}_t = \{\text{argmax}_{\Theta\in\mathcal{G}_{t-1}} \mu_\lambda(\Theta)\}$ with
      the best-performing candidate from the previous
      generation;
    for unique pairs $\{\Theta_j, \Theta_k\}$ formed from the top $\ell$ candidates
      from $\mathcal{G}_{t-1}$ do #form new candidates using crossover
        Initialize $M_1 \in \mathbb{R}^{m\times d}$ and $b_1 \in \mathbb{R}^m$ as the $M_1$ and $b_1$ from
          $\Theta_j$;
        for $\kappa = 1, \dots, m$ do
            Let $\xi \sim \text{Bernoulli}(1/2)$;
            if $\xi = 1$ then replace the $\kappa$th row of $M_1$ and the $\kappa$th
              component of $b_1$ with those from $\Theta_k$;
        end
        Randomly initialize $M_2$ and $b_2$ and take optimization
          steps towards the solution to (7);
        Add $\Theta = (M_1, M_2, b_1, b_2)$ to $\mathcal{G}_t$
    end
    while $|\mathcal{G}_t| < c$ do
        Optimize a parameter set $\Theta$ to seek (7) on training
          batches from a random initialization;
        Add $\Theta$ to $\mathcal{G}_t$
    end
end
Let $\Theta_* = \text{argmax}_{\Theta\in\mathcal{G}_g} \mu_\lambda(\Theta)$;
return $\Phi_{\Theta_*}$ as in (8)
```

Algorithm 1: Neuroevolutionary Feature Engineering for Causal Inference

3.4. Remark on linearity

Due to our choice of representation Φ in (8), after training the network (6) to optimize (7), we expect the relationship between the learned features $\Phi(X)$ and the outcome Y to be approximately linear. In particular, we will have $Y \approx M_2 \cdot \Phi(X)$ for M_2 as given in (6). For this reason, the causal meta-learners trained using a linear regression base learner may benefit more extensively from using the transformed features instead of the original features, especially in cases where the relationship between the original features and outcomes is not well-approximated as linear. We provide specific examples in the next section involving meta-learners trained with ridge regression.

3.5. Remark on our assumptions

Through our method of constructing features, we intend that $\{Y(0), Y(1)\} \perp\!\!\!\perp X \mid \Phi(X)$. This formalizes the notion that $\Phi(X)$ should contain all the information in X relevant for predicting Y. In order to then use the represented features in place of the original features for learning the CATE, we also require that strong ignorability (as defined in Section 1) holds for the represented features. This standard assumption is necessary in order to be able to learn the CATE from data. If X, W, Y, and $\Phi(X)$ jointly respect the graph

$$\begin{tikzcd} X \arrow[dr] \arrow[dd] & & \\ & \Phi(X) \arrow[dl] \arrow[dr] & \\ W \arrow[rr] & & Y \end{tikzcd}$$

then it follows [4, §3.3] that $\{Y(0), Y(1)\} \perp\!\!\!\perp (X, W) \mid \Phi(X)$, as desired.

4. Ablation study on generated data

Due to the fundamental challenge of causal inference (namely, that the counterfactual outcome cannot be observed, even in controlled experiments), it is common practice to compare approaches to CATE estimation on artificially generated datasets that allow the CATE to be calculated directly for evaluative purposes. In this section, we perform experiments using two such data generation mechanisms from Nie & Wager's paper [5] that we now describe.[3] Both setups provide a joint distribution satisfying the graph (1) and allow us to explicitly calculate the CATE in order to evaluate algorithm performance. For the vector-valued random variable $X \in \mathbb{R}^d$, we let X_{ij} denote jth component $(1 \leq j \leq d)$ of the ith sample $(1 \leq i \leq n)$. The specifics for both setups are given as follows.

Setup A

For $\sigma > 0$ and an integer $d > 0$, we let

$$X_i \sim^{\text{i.i.d.}} \text{Uniform}([0,1]^d)$$

and $\quad W_i \mid X_i \sim \text{Bernoulli}(e(X_i)),$

where $e(X_i) = \max\{0.1, \min\{\sin(\pi X_{i1} X_{i2}), 0.9\}\}$ and

$$Y_i \mid X_i, W_i \sim \mathcal{N}\big(b(X_i) + (W_i - 0.5)\tau(X_i), \sigma^2\big),$$

[3] Nie & Wager's paper included four setups, namely A–D; however setup B modeled a controlled randomized trial and setup D had unrelated treatment and control arms. In both cases, the original features are already independent from W, so that the two aims of our representation are already satisfied.

where $b(X_i) = \sin(\pi X_{i1} X_{i2}) + 2(X_{i3} - 0.5)^2 + X_{i4} + 0.5X_{i5}$. The true CATE for this setup is then $\tau(X_i) = (X_{i1} + X_{i2})/2$. In this paper, we let $d = 24$, $n = 200$, and $\sigma = 1$.

Setup C

For $\sigma > 0$ and an integer $d > 0$, we let

$$X_i \sim^{\text{i.i.d.}} \mathcal{N}_d(\vec{0}, I_{d\times d})$$

and $\quad W_i \mid X_i \sim \text{Bernoulli}(e(X_i)),$

where in this case $e(X_i) = (1 + \exp(X_{i2} + X_{i3}))^{-1}$ and

$$Y_i \mid X_i, W_i \sim \mathcal{N}\big(b(X_i) + (W_i - 0.5)\tau(X_i), \sigma^2\big),$$

where now $b(X_i) = 2\log(1 + \exp(X_{i1} + X_{i2} + X_{i3}))$. For this model, we have $\tau(X_i) = 1$. In our example, we let $d = 12$, $n = 500$, and $\sigma = 1$.

4.1. Comparison methodology

For each data generation method, we ran 100 independent trials. Within each trial, we simulated a dataset of size n and randomly partitioned it into training, validation, and testing subsets at a 70%-15%–15% rate. We trained causal inference methods on the training set, using the validation data to aid the training of some base estimators for the meta-learners, and predicted on the test dataset. We then developed a feature map using the training and validation data as described in the previous section, applied this map to all features, and repeated the training and testing process using the new features.

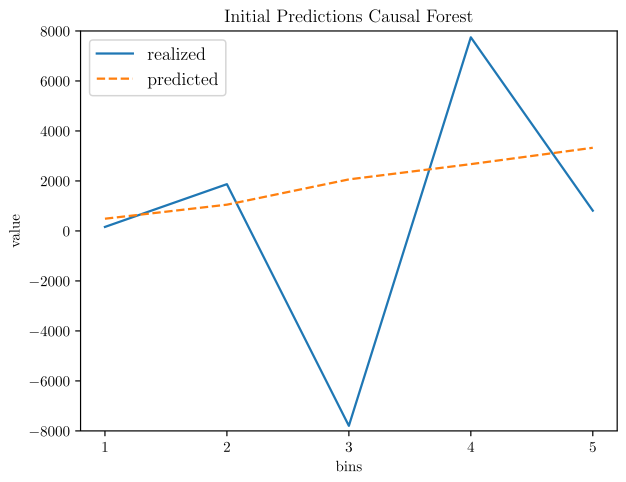
We compared feature maps generated using tanh, relu, and elu functions for the activation $a(\cdot)$ and various choices of fitness parameter $\lambda \geq 0$. To determine the impact of the fitness selection process, we also learned a feature transformation that did not make use of the fitness function at all. In effect, it simply generated a single candidate mapping and used it to transform all the features (without any cross-over or further mutation). Features developed in this way are described as “no fitness” in the tables.

To estimate the CATE, we used the causal forest with default options (as found in R’s grf package), and the S-, T-, and X-learners as described in Section 2.1 with two base learners. The first base learner is LightGBM [36], a boosted random forest algorithm that introduced novel techniques for sampling and feature bundling. The second is a cross-validated ridge regression model (as found in scikit-learn) that performs multiple linear regression with an L^2 normalization on the weights.

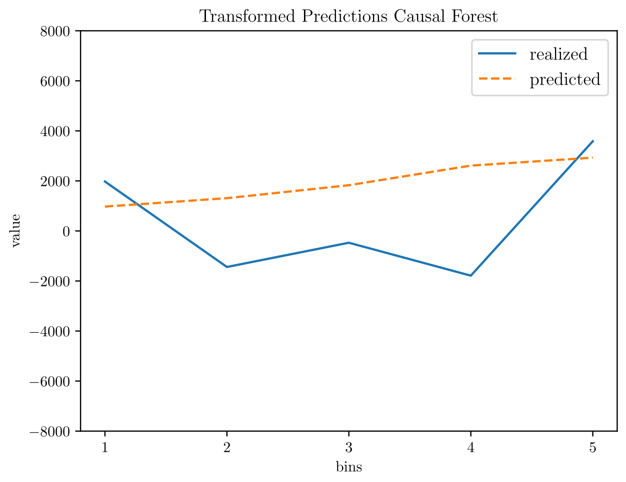
4.2. Results

We report performance results for setup A in Table 1 and results for setup C in Table 2. For each learning approach, we compare testing performance using our features versus the initial features. We report the p-value from a paired t-test for the equality of means for the MSE performance obtained using the initial features vs. the transformed features. We find that our features result in significantly improved performance for a variety of different CATE learners. In many cases, adjusting the tuning parameter λ to a value greater than zero offers a performance boost, so that it can be useful to protect against the loss of information for predicting the outcome during the feature engineering process. None of the activation functions (tanh/elu/relu) perform uniformly better across all trials. In some cases, we fail to see any improvement from our approach. For the single learner with ridge regression on setup C, we find no improvement, and for the single learner with LightGBM in setup A, we find that using engineered features may lead to worse performance. Performance is otherwise promising.

In summary, we find that our feature transformation method improves the performance of multiple standard estimators for the CATE under two challenging data generation models.



(a) predictions using original features



(b) predictions using transformed features

Fig. 2. We plot estimated realized and predicted average treatment effects versus the quintiles of predicted treatment effect (5 bins) for a causal forest using (2(a)) the initial features and (2(b)) the features transformed using our method (with hyperbolic tangent for $a(\cdot)$ and $\lambda = 0$). In order to estimate realized treatment effect within each bin, we take the difference between the average outcome for persons randomly assigned to the experimental group and the average outcome for persons randomly assigned to the control group.

5. Application to econometric data

In this section, we apply our feature engineering method to the LaLonde dataset [37,38] chronicling the results of an experimental study on temporary employment opportunities. The dataset contains information from 445 participants who were randomly assigned to either an experimental group that received a temporary job and career counseling or to a control group that received no assistance. Features include age and education (in years), earnings in 1974 (in $, prior to treatment), and indicators for African–American heritage, Hispanic–American heritage, marital status, and possession of a high school diploma. We consider the outcome of earnings in 1978 (in $, after treatment).

We cannot determine true average treatment effects based on individual-level characteristics (i.e. the true CATE values) for real life experimental data as we can with the synthetic examples of the previous section. Instead, we evaluate performance by comparing the average realized treatment effect and average predicted treatment effect within bins formed by sorting study participants according to predicted treatment effect as demonstrated in Fig. 2. Applying the causal forest predictor to the original features results in a root mean square difference between the average predicted and realized treatment effects of 5099.90. If the transformed features are used instead, this discrepancy improves to 2593.23.

From a practical perspective, one may learn the CATE in order to select a subset of people for whom a given intervention has an expected net benefit (and then deliver that intervention only to persons predicted to benefit from it). When we focus on the 20% of people predicted to benefit most from this treatment, we find that the estimated realized benefit for those chosen using the transformed features ($3585.54) is much greater than the benefit for those chosen using the original feature set ($811.94). This can be seen visually in Fig. 2 by comparing the estimated realized average treatment effect for bin #5 (the rightmost bin) in both plots.

6. Conclusions

Causal inference, especially on real life datasets, poses significant challenges but offers a crucial avenue for predicting the impact of potential interventions. Learned feature representations help us to better infer the conditional average treatment effect, improving our ability to individually tailor predictions and target subsets of the general population. In this paper, we propose and validate a novel representation-based method that uses a neuroevolutionary approach to remove information from features irrelevant for predicting the outcome. We demonstrate that this method can yield improved estimates for heterogeneous treatment effects on standard synthetic examples and illustrate its use on a real life dataset. We believe that representational learning is particularly well-suited for removing extraneous information in causal models and we anticipate future research in this area.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Michael Burkhart and Gabriel Ruiz report financial support was provided by Adobe, Inc. Michael Burkhart and Gabriel Ruiz have patent ‘Causal Inference via Neuroevolutionary Selection’ pending to Adobe, Inc.

Data availability

The data used is already publicly available.

Acknowledgments

We would like to thank Binjie Lai, Yi-Hong Kuo, and Xiang Wu for their support and feedback. We are also grateful to the anonymous reviewers for their insights and suggestions.

Appendix. Implementation details

All numerical experiments were performed on a 2020 MacBook Pro (Apple M1 Chip; 16 GB LPDDR4 Memory) with Python 3.7.7 and R 3.6.0. We used Python packages (versioning in parentheses) Keras (1.0.8), LightGBM (3.1.1), Matplotlib (3.3.4), Numpy (1.20.2), Pandas (1.2.4), rpy2 (2.9.4), Scikit-learn (0.24.2), Scipy (1.6.2), statsmodels (0.13.2), Tensorflow (2.0.0), and XGBoost (1.3.3), along with the R package grf (1.2.0).

References

[1] J. Neyman, Sur les applications de la théorie des probabilités aux experiences agricoles: Essai des principes, Rocz. Nauk Rol. 10 (1923) 1–51.
[2] D. Rubin, Estimating causal effects of treatments in randomized and nonrandomized studies, J. Educ. Psychol. 66 (5) (1974) 688–701.
[3] P.R. Rosenbaum, D.B. Rubin, The central role of the propensity score in observational studies for causal effects, Biometrika 70 (1) (1983) 41–55.
[4] J. Pearl, Causality, second ed., Cambridge Univ. Press, 2009.
[5] X. Nie, S. Wager, Quasi-oracle estimation of heterogeneous treatment effects, Biometrika 108 (2) (2021) 299–319.
[6] J.H. Holland, Adaptation in Natural and Artificial Systems, U. Michigan Press, 1975.
[7] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.
[8] F. Johansson, U. Shalit, D. Sontag, Learning representations for counterfactual inference, in: Int. Conf. Mach. Learn., 2016, pp. 3020–3029.
[9] U. Shalit, F.D. Johansson, D. Sontag, Estimating individual treatment effect: generalization bounds and algorithms, in: Int. Conf. Mach. Learn., Vol. 70, 2017, pp. 3076–3085.
[10] S. Li, Y. Fu, Matching on balanced nonlinear representations for treatment effects estimation, in: Adv. Neur. Inf. Proc. Sys., 2017, pp. 930–940.
[11] L. Yao, S. Li, Y. Li, M. Huai, J. Gao, A. Zhang, Representation learning for treatment effect estimation from observational data, in: Adv. Neur. Inf. Proc. Sys., Vol. 31, 2018, pp. 2633–2643.
[12] Y. Zhang, A. Bellot, M. van der Schaar, Learning overlapping representations for the estimation of individualized treatment effects, in: Int. Conf. Artif. Intell. Stats, 2020.
[13] H. Zhao, R.T. des Combes, K. Zhang, G.J. Gordon, On learning invariant representations for domain adaptation, in: Int. Conf. Mach. Learn., 2019.
[14] M.C. Burkhart, G. Ruiz, Neuroevolutionary feature representations for causal inference, in: Int. Conf. Comput. Sci., in: LNCS, vol. 13351, 2022, pp. 3–10.
[15] M.C. Burkhart, G. Ruiz, Causal inference via neuroevolutionary selection, 2022, U.S. Patent Application, filed.
[16] S.R. Künzel, J.S. Sekhon, P.J. Bickel, B. Yu, Metalearners for estimating heterogeneous treatment effects using machine learning, Proc. Natl. Acad. Sci. USA 116 (10) (2019) 4156–4165.
[17] P.M. Robinson, Root-n-consistent semiparametric regression, Econometrica 56 (4) (1988) 931–954.
[18] J.M. Robins, Optimal structural nested models for optimal sequential decisions, in: 2nd Seattle Symp. Biostat, in: LNS, vol. 179, 2004, pp. 189–326.
[19] S. Athey, J. Tibshirani, S. Wager, Generalized random forests, Ann. Statist. 47 (2) (2019) 1148–1178.
[20] S. Wager, S. Athey, Estimation and inference of heterogeneous treatment effects using random forests, J. Amer. Statist. Assoc. 113 (523) (2018) 1228–1242.
[21] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Tech. Rep. C3P 826, Calif. Inst. Technol., 1989.
[22] R. Forsyth, Beagle – a Darwinian approach to pattern recognition, Kybernetes 10 (3) (1981) 159–166.
[23] L. Fogel, Autonomous automata, Ind. Res. Mag. 4 (2) (1962) 14–19.
[24] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution (Ph.D. thesis), Technical U. Berlin, 1970.
[25] H.-P. Schwefel, Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik (Ph.D. thesis), Technical U. Berlin, 1975.
[26] K. Stanley, J. Clune, J. Lehmn, R. Miikkulainen, Designing neural networks through neuroevolution, Nat. Mach. Intell. 1 (2019) 24–35.
[27] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[28] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Int. Conf. Artif. Intell. Stat., Vol. 9, 2010, pp. 249–256.
[29] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Int. Conf. Learn. Represent, 2015.
[30] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011) 2121–2159.
[31] M.D. Zeiler, Adadelta: An adaptive learning rate method, 2012, ArXiv e-prints.
[32] A.N. Tikhonov, Solution of incorrectly formulated problems and the regularization method, Proc. USSR Acad. Sci. 151 (3) (1963) 501–504.
[33] V. Nair, G. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Int. Conf. Mach. Learn, 2010, pp. 807–814.
[34] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), in: Int. Conf. Learn. Represent., 2016.
[35] D.J. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, in: Int. Jt. Conf. Artif. Intell, 1989, pp. 762–767.
[36] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in: Adv. Neur. Inf. Proc. Sys, 2017, pp. 3146–3154.
[37] R.J. LaLonde, Evaluating the econometric evaluations of training programs with experimental data, Amer. Econ. Rev. 76 (4) (1986) 604–620.
[38] R.H. Dehejia, S. Wahba, Causal effects in nonexperimental studies: Reevaluating the evaluation of training programs, J. Amer. Statist. Assoc. 94 (448) (1999) 1053–1062.

Michael C. Burkhart earned his Ph.D. in 2019 from Brown University's Division of Applied Mathematics, after completing an M.Sc. in mathematics at Rutgers and B.Sc.'s in mathematics, statistics, & economics at Purdue. He then worked as a machine learning scientist at Adobe in California, where he met his coauthor Gabriel. In 2021, he joined Cambridge University as a research associate to develop machine learning-based approaches for the early diagnosis of neurodegenerative disease.

Gabriel Ruiz earned his Ph.D. in 2022 from University of California Los Angeles' Department of Statistics. Prior to this, he completed a B.Sc. in Statistics at University of California Riverside in 2017. He currently works as a machine learning scientist within the Experience Intelligence group at Adobe in California.