NIMO: a Nonlinear Interpretable MOdel

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural networks (NNs) have achieved tremendous success over the past decade, yet they are still extremely difficult to interpret. In contrast, linear models are less expressive but offer inherent interpretability. Linear coefficients are interpretable as the marginal effect of a feature on the prediction, assuming all other features are kept fixed. To combine the benefits of both approaches, we introduce NIMO (Nonlinear Interpretable MOdel). The key idea is to define a model where the NN is designed to learn non-linear corrections to the linear model predictions, while also maintaining the original interpretability of the linear coefficients. Relevantly, we develop an optimization algorithm based on profile likelihood that elegantly allows for optimizing over the NN parameters while updating the linear coefficients analytically. By relying on adaptive ridge regression we can easily incorporate sparsity constraints as well. We show empirically that we can recover the underlying linear coefficients while significantly improving the predictive accuracy. Compared to other hybrid interpretable approaches, our model is the only one that actually maintains the same interpretability of linear coefficients as in linear models. We also achieve higher performance on various regression and classification settings.

1 Introduction

Over the course of the past decade machine learning research has witnessed enormous success thanks to neural networks. Applications range from computer vision, natural language processing, speech recognition, robotics, finance and many more. Along with the development of these remarkable capabilities came the desire for understanding the underlying decision-making process. However, neural networks still lack a clear interpretability, which prevents their widespread use in high-stake areas like healthcare. Most research has focused on how to address this problem, either by developing intrinsically interpretable models or by providing post-hoc explanations of model predictions.

In this work, we focus on the design of an intrinsically interpretable model by employing the inherent transparency of linear models. Because of the linear relationship between the input data and the output predictions, linear models are simple and their decision-making process is transparent. The sign of a linear coefficient indicates the direction of the relationship (positive or negative), and its magnitude reflects the significance of that feature on the prediction. However, the simplicity of linear models constrains their expressiveness. In contrast, neural networks have shown great ability in learning complex feature maps from the input data in order to predict the output. While this lead to an enormous success in many applications, NNs lack interpretability of the predictions in terms of the input features, hence the term "black-box" models. This is particularly evident in the medical domain where, in order to make informed decisions, practitioners need to understand which factors lead to a particular model outcome.

To solve this dilemma, we introduce the Nonlinear Interpretable MOdel (NIMO), which inherits the expressivity of neural networks and also maintains the transparency and interpretability of linear models. More specifically, we start from a linear model and from the associated linear coefficients $\boldsymbol{\beta}$

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

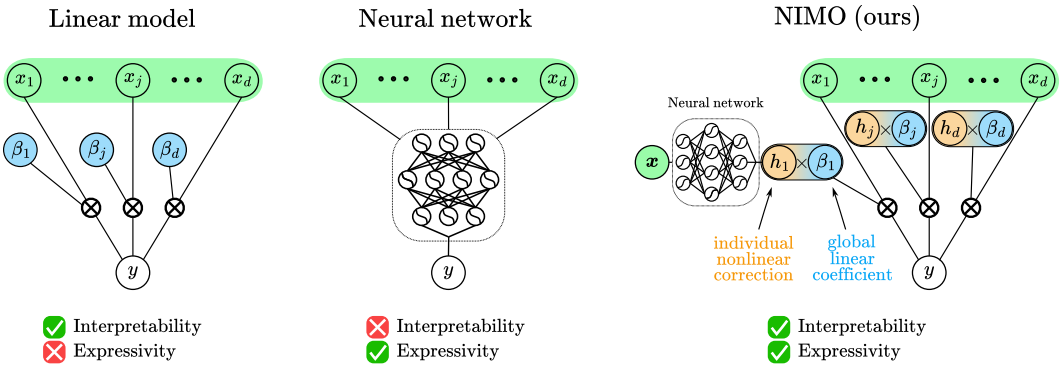


Figure 1: Comparison of linear models, neural networks, and our method. Linear models offer straightforward interpretability but limited expressivity while neural networks provide high expressivity but lack interpretability. Our method combines the strengths of both by building on the linear model and adjusting the global linear coefficients with individual nonlinear corrections.

that provide a clear interpretability at a population level. Then, we adjust each β_j by a multiplicative factor h_j according to the individual point $\boldsymbol{x}$. We learn such correction terms with a neural network $\text{NN}_{\boldsymbol{u}}$ so that the model can learn nonlinear interaction terms beyond the linear ones:

$$y = \sum_{j=1}^{p} x_j \beta_j h_j \quad \text{with} \quad h_j = \text{NN}_{\boldsymbol{u}}(\boldsymbol{x}) \; . \tag{1}$$

If the neural network has a suitable design, the resulting model maintains the interpretability of linear coefficients $\boldsymbol{\beta}$ at a global level while also providing a nonlinear correction at the local level, depending on the data point $\boldsymbol{x}$. In Figure 1 we illustrate how the proposed approach shares the same high-level structure of a linear model while also integrating nonlinear corrections through neural networks. In theory we would need one neural network per feature, but in practice we can use a single network with the feature appended with positional encoding. In Section 3 we show that, with simple requirements on the neural network, we can preserve the original interpretability of the linear coefficients $\boldsymbol{\beta}$. Since the network acts as a nonlinear correction to the linear coefficients $\boldsymbol{\beta}$, the joint optimization of $\boldsymbol{\beta}$ and $\boldsymbol{u}$ might not be trivial in practice. To address this issue, we develop an optimization method based on the profile likelihood which allows for sparsity constraints as well. Specifically, we alternate a gradient step over the neural network parameters with an analytical update over the linear coefficients $\boldsymbol{\beta}$. Lastly, we show that we can extend the present framework to generalized linear models.

As commonly done in the literature, we further enforce interpretability by means of sparsity. When fewer components (features or connections) are active, it is easier to understand how the information leading to some prediction is processed. In the linear part of our model, we employ L1 regularization to enforce the sparsity in the coefficients $\boldsymbol{\beta}$. In the nonlinear part originating from the network, we resort to the group L2 regularization on the weight matrix of the first fully connected (`fc`) layer. We also employ a novel noise injection technique to the output of the first `fc` layer in order to encourage the selection of the most relevant features.

Overall, the contributions of the present work can be summarized as follows:

- We introduce a nonlinear interpretable model (NIMO) that combines the expressive power of neural networks with the well-established interpretability of linear models.
- We introduce an optimization algorithm that allows to jointly optimize the linear coefficients and the nonlinear contributions of the neural network.
- On synthetic data, we show that we retrieve the underlying linear coefficients, learn the correct nonlinearities and outperform state-of-the-art hybrid models. On real data, we demonstrate NIMO's interpretability by showing that we recover sparse coefficients while achieving performance on par with neural network approaches.

2 Related Work

Interpretability in machine learning. Interpretability refers to the capacity to express what a model has learned and the factors influencing its outputs in a manner that is clear and understandable to humans. In the literature, the two terms interpretability and explainability are sometimes used interchangeably. However, they have subtle yet important differences. Explainability focuses on providing reasons for the model's output. Most of the methods, especially in deep learning, provide post-hoc explanations. The most popular ones include LIME (Ribeiro et al., 2016), SHAP (Lundberg and Lee, 2017) and Grad-CAM (Selvaraju et al., 2020). On the other hand, interpretability focuses on understanding the inner mechanisms and decision processes of the model, and most of the methods involve inherently understandable models, such as decision trees (Wu et al., 2017, 2021), Lasso (Tibshirani, 1996) and Explainable Boosting Machines (Lou et al., 2013). Often, simple interpretability comes at the cost of reduced expressivity. The literature on interpretable machine learning is vast and we refer readers to (Rudin et al., 2022; Molnar, 2025) for a more in-depth review. In this work, we develop a model that has a straightforward interpretability and that, at the same time, is very expressive and flexible.

Interpretability of linear models. Linear models are inherently interpretable. They model the relationship between features and the target variable as a linear combination, making the effect of each feature easy to understand and quantify (Molnar, 2025). For example, consider a linear regression model used to predict the risk of heart disease based on a set of clinical features. Then, the coefficient associated with a specific clinical feature represents the change in the predicted risk when that feature increases by one unit, assuming all other feature values remain fixed. Methods like Lasso regularization (Tibshirani, 1996) can be further applied to achieve sparse feature selection and focus on the most relevant features. However, linear models have limited expressiveness and fail to capture complex relationships among the features. In this work, we maintain the interpretability of linear models and, at the same time, we increase the model expressiveness through a neural network.

Hybrid models. Several works have proposed to combine linear models with neural networks to achieve more interpretable models. However, none of them actually retain the original interpretability of linear coefficients as in linear models. LassoNet (Lemhadri et al., 2021) is the first approach to integrate a neural network with Lasso regression. It allows a feature to be used in the neural network only if its corresponding Lasso coefficient is nonzero. This constraint ensures that both the neural network's nonlinear component and the Lasso linear component operate on the same subset of features. Even though this allows for feature selection, deactivating the features that are deemed unimportant in the linear part significantly constrains the capacity of the neural network. Contextual Lasso (Thompson et al., 2023) is a type of varying coefficient model (Hastie and Tibshirani, 1993). The linear regression coefficients are context-dependent and generated by a neural network. This structure provides flexibility but comes at the cost of sacrificing global interpretability of the coefficients, which is key for the interpretability of linear models. In this work, we develop an inherently interpretable model called NIMO, which maintains the global interpretability of linear models while also learning nonlinear corrections to the linear predictions.

3 Proposed approach

In Section 3.1 we introduce our non-linear interpretable model, namely NIMO, and argue why we can still retain the full interpretability of linear models while also allowing for non-linearities. Then, in Section 3.2 we propose an algorithm to elegantly train the model on the neural network parameters while analytically updating the linear coefficients. In Section 3.3 we show how to find relevant nonlinear interactions by means of sparsity. Lastly, In Section 3.4 we discuss the limitations.

3.1 A Nonlinear Interpretable Model

Model definition Let $X \in \mathbb{R}^{n \times d}$ be n d-dimensional observations and let $\boldsymbol{y} \in \mathbb{R}^n$ be the targets. In the following, we assume the data X to be standardized, i.e. that each feature has zero mean and unit standard deviation. A linear model is simply defined by the linear coefficients $\boldsymbol{\beta} \in \mathbb{R}^d$ and a bias term $\beta_0 \in \mathbb{R}$ as $y_i \approx \beta_0 + \sum_{j=1}^{d} x_j \beta_j$. In linear models the coefficient β_j is a proxy for the importance of the j-th feature. More precisely, the linear coefficient β_j can be interpreted as the additive effect of the j-th feature on the output with all other variables at baseline level. The idea

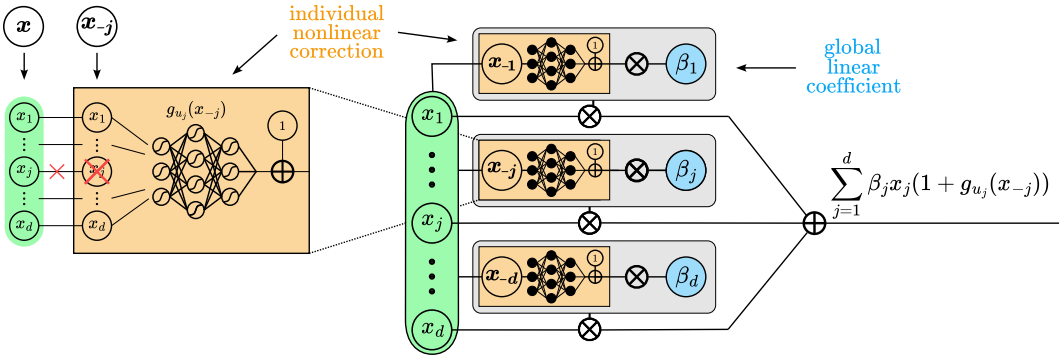


Figure 2: NIMO architecture. The model consists of global linear coefficients β_j modulated by the individual data point $\boldsymbol{x}$ through h_j. In particular, h_j depends on $\boldsymbol{x_{-j}}$, i.e. all features except the j-th.

behind our approach is to maintain the interpretability of global linear coefficients while allowing for individual nonlinear corrections to the linear predictions. In order to accomplish this, we define our approach starting from a linear model and by multiplying the linear coefficients β_j by a nonlinear correction term h_j that depends on the individual data point $\boldsymbol{x}$. The model is defined as follows:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{d} x_j \beta_j \underbrace{(1 + g_{\boldsymbol{u}_j}(\boldsymbol{x_{-j}}))}_{h_j} \quad \text{s.t.} \quad g_{\boldsymbol{u}_j}(\boldsymbol{0}) = 0 \,, \tag{2}$$

where $\boldsymbol{x_{-j}}$ is the vector $\boldsymbol{x}$ without the j-th component and where $g_{\boldsymbol{u}_j} : \mathbb{R}^{d-1} \mapsto \mathbb{R}$ are d different neural networks, each parametrized by $\boldsymbol{u}_j$. We provide an illustration of the proposed approach in Figure 2. To make sure that our model in Eq. (2) fulfills the intended interpretability we took care of three aspects. **First**, the network $g_{\boldsymbol{u}_j}$ needs to act as a non-linear correction to the linear prediction. This is easily achieved by designing the neural network as an additive contribution to the linear prediction. We multiply the coefficient β_j by $h_j = 1 + g_{\boldsymbol{u}_j}$, which renders the additive effect: $f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{d} x_j \beta_j + \sum_{j=1}^{d} x_j \beta_j g_{\boldsymbol{u}_j}(\boldsymbol{x_{-j}})$. **Second**, the neural network $g_{\boldsymbol{u}_j}$ cannot depend on the j-th feature itself. Otherwise, x_j would contribute to the prediction not only through the linear term β_j but also through the network itself, which would change the behavior and interpretation of the coefficient β_j. **Third**, we make sure that the non-linear contribution coming from the neural network keeps the feature at baseline level (as in the linear part), i.e. $g_{\boldsymbol{u}_j}(\boldsymbol{x_{-j}} = \boldsymbol{0}) = \boldsymbol{0}$. It is easy to see that when $\boldsymbol{x_{-j}} = \boldsymbol{0}$ the model reduces to classical linear regression.

Model interpretability At a high level, the proposed approach resembles a linear model where each coefficient is given by the product $\beta_j h_j$. Intuitively, β_j is independent of the data point $\boldsymbol{x}$ and acts at a *population* level. In contrast, h_j provides a nonlinear correction to β_j and depends on the *individual* data point $\boldsymbol{x}$ (more precisely $\boldsymbol{x_{-j}}$) through $g_{\boldsymbol{u}}$. This construction allows to keep the global interpretability as in a linear model while also allowing for a local nonlinear correction. To see why this is relevant, consider a healthcare application where we want to predict a treatment outcome based on some clinical data. The resulting linear coefficients represents at a population level the contribution of each feature to the prediction. However, the global explanation might not be a good fit for all patients, which is a limitation of linear models. For some patients the outcome might be better explained by a different coefficient, which we can learn by multiplying the global β_j by a local h_j specific to that patient. Overall, the proposed approach provides a population-wise interpretation of linear coefficients while still allowing for interpretable nonlinear and local corrections.

Implementation details The proposed model in Eq. (2) requires d different neural networks, which can require significant memory in high-dimensional settings. Instead, we use a single shared network $g_{\boldsymbol{u}}$ and employ positional encoding to make the network aware of which component is being queried. This can be easily done by appending the positional encoding of the j-th component to the input $\boldsymbol{x_{-j}}$. In practice, $g_{\boldsymbol{u}}$ takes the full vector $\boldsymbol{x}$ as input and we can simply mask out the j-th component of $\boldsymbol{x}$ to get $g_{\boldsymbol{u}}(\boldsymbol{x_{-j}})$. Furthermore, we enforce $g_{\boldsymbol{u}}(\boldsymbol{x_{-j}} = \boldsymbol{0}) = 0$ by design by simply subtracting the mean prediction at each forward pass: $g_{\boldsymbol{u}}(\boldsymbol{x_{-j}}) \coloneqq g_{\boldsymbol{u}}(\boldsymbol{x_{-j}}) - g_{\boldsymbol{u}}(\boldsymbol{0})$.

3.2 Training through profile likelihood with adaptive ridge regression

For illustrative purposes, in this section we provide a high-level description of the optimization algorithm used to train NIMO. We initially consider a regression setting and later extend the same reasoning to generalized linear models, in particular for the logistic regression case.

Optimization for sparse regression Consider the data $X \in \mathbb{R}^{n \times d}$ and targets $\boldsymbol{y} \in \mathbb{R}^d$. As argued before, we can model the d networks with one network $g_{\boldsymbol{u}}$ and represent its outputs in a matrix $G_{\boldsymbol{u}} = g_{\boldsymbol{u}}(X) \in \mathbb{R}^{n \times d}$. The model in Eq. (2) can then be re-written in matrix form as $f(X) = B_{\boldsymbol{u}}\boldsymbol{\beta}$, where $B_{\boldsymbol{u}} = X + X \circ G_{\boldsymbol{u}}$ and "$\circ$" denotes element-wise multiplication. Let us first consider the standard ridge regression setting: $\mathcal{L}(\boldsymbol{\beta}, \boldsymbol{u}) = \|\boldsymbol{y} - B_{\boldsymbol{u}}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2$. To disentangle the optimization of $\boldsymbol{\beta}$ and $\boldsymbol{u}$, we take inspiration from the profile likelihood (Venzon and Moolgavkar, 1988) and consider an optimization based on $\boldsymbol{u}$ only. Since we know the analytical solution of $\boldsymbol{\beta}$ for ridge regression, we can alternate one optimization step over $\boldsymbol{u}$ with an analytical update over $\boldsymbol{\beta}$:

$$\min_{\boldsymbol{\beta},\boldsymbol{u}} \mathcal{L}(\boldsymbol{\beta}, \boldsymbol{u}) \quad \rightarrow \quad \min_{\boldsymbol{u}} \mathcal{L}(\hat{\boldsymbol{\beta}}, \boldsymbol{u}) \quad \text{where} \quad \hat{\boldsymbol{\beta}} = (B_{\boldsymbol{u}}^T B_{\boldsymbol{u}} + \lambda I)^{-1} B_{\boldsymbol{u}}^T \mathbf{y}. \tag{3}$$

In practice, $\hat{\boldsymbol{\beta}}$ depends on the current estimate of $\boldsymbol{u}$ via $B_{\boldsymbol{u}}$ so we alternate one gradient descent step over $\boldsymbol{u}$ with one analytical update over $\boldsymbol{\beta}$ according to $\hat{\boldsymbol{\beta}}$ in Eq. (3). Note that in practice this is much more effective than freely learning via gradient descent $\boldsymbol{\beta}$ alongside with $\boldsymbol{u}$. Furthermore, this formalism allows to include sparsity by simply replacing the L2 penalty with the L1 penalty:

$$\min_{\boldsymbol{\beta},\boldsymbol{u}} \|\mathbf{y} - B_{\boldsymbol{u}}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|_1 \ . \tag{4}$$

In contrast to the ridge objective, there is no closed-form solution for Lasso regression. Instead, we can use adaptive ridge regression, which is equivalent to Lasso (Grandvalet, 1998); see the Appendix for a detailed proof. By rewriting our objective as in adaptive ridge regression, we can enforce sparsity and find an analytical update for $\boldsymbol{\beta}$. The exact algorithm is shown in Alg. 1 in the Appendix.

Extension to generalized linear models The same idea can be extended to any generalized linear model. For simplicity, consider logistic regression with L1 penalty. As before, we first get a closed-form expression for $\boldsymbol{\beta}$ and then substitute it back to optimize over the neural network parameters $\boldsymbol{u}$. What is different is that in each optimization step we use the iteratively reweighted least squares (IRLS) surrogate, which is a weighted least-squares approximation of the original problem:

$$\min_{\boldsymbol{\beta},\boldsymbol{u}} \ \tfrac{1}{2}\big\|W^{1/2}(\mathbf{z} - B_{\boldsymbol{u}}\boldsymbol{\beta})\big\|^2 + \lambda\|\boldsymbol{\beta}\|_1 \ , \tag{5}$$

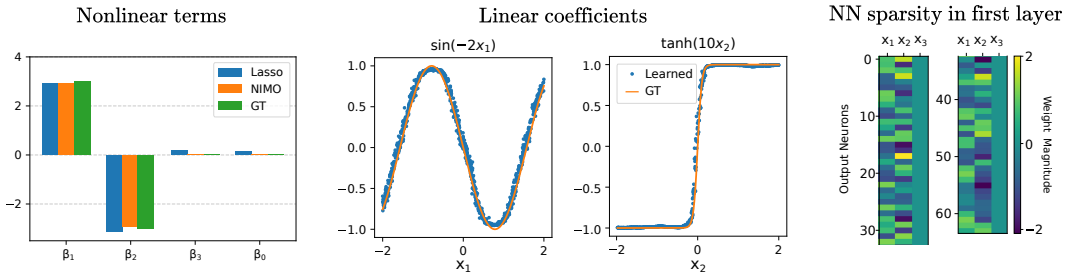
where $W_{ii} = \sigma(\eta_i)\big[1 - \sigma(\eta_i)\big]$ and $\boldsymbol{\eta} = B_{\boldsymbol{u}}\boldsymbol{\beta}$. The so-called working response $\boldsymbol{z}$ is computed as $\boldsymbol{z} = \boldsymbol{\eta} + W^{-1}\big(\mathbf{y} - \sigma(\boldsymbol{\eta})\big)$. The problem in Eq. (5) is also a lasso problem and we can still employ adaptive ridge regression to find a closed form solution for $\boldsymbol{\beta}$. Then we substitute $\boldsymbol{\beta}$ back and optimize over the neural networks parameters $\boldsymbol{u}$. A detailed derivation can be found in the Appendix.

3.3 Sparsity of nonlinear interactions

In the previous paragraphs we showed that linear coefficients $\boldsymbol{\beta}$ in NIMO are interpretable as in a linear model. Even though the neural network contribution is not interpretable per se, we can still identify relevant nonlinear interactions by means of sparsity. The idea is to force the first layer of the neural network to be sparse, hence to use only relevant features. To do so, we apply a group penalty on the weight matrix $W \in \mathbb{R}^{p \times d}$ of the first fully-connected (`fc`) layer, where d is the number of input features and p the hidden dimension of the first `fc` layer. In order to obtain sparsity at a input feature level, we need to consider neurons connected to such feature and penalize them as a group:

$$\ell_{\text{group}} = \lambda_{\text{group}} \sum_{j=1}^{d} \|\mathbf{w}_j\|_2 \ . \tag{6}$$

where $\mathbf{w}_j = W[:, j] \in \mathbb{R}^p$ is the weight vector acting on the j-th feature. With this penalty on the first `fc` layer, we encourage it to select only certain input features. However, since an MLP computes a composition of functions, the subsequent `fc` layers can still compensate for the sparsity induced by the regularization in the first layer. Specifically, even if the first layer produces sparse outputs,



Toy example: $y = 3 \cdot x_1(1 + \tanh(10x_2)) - 3 \cdot x_2(1 + \sin(-2x_1)) + 0 \cdot x_3 + \epsilon$

Figure 3: Illustrative toy example. *Left*: NIMO learns the GT coefficients and also finds the correct sparsity. *Middle*: the learned nonlinearities coincide with the GT. *Right*: the first `fc` layer of the NN learns the correct sparsity: only features x_1 and x_2 have non-zero weights, while x_3 is ignored.

subsequent layers can reweigh and transform these outputs to restore their influence on the final prediction. To mitigate this compensation effect, we inject noise into the output of the first layer. By doing so, we disrupt the deterministic signal path, making it harder for the subsequent layers to simply exploit the resulting weak, regularized, and noisy signal. As a result, the network is encouraged to focus on strong and robust signals associated with informative input features.This approach works very well in practice and allows to easily achieve sparsity. We provide further details in the Appendix.

3.4 Limitations

The proposed approach is designed to maintain the interpretability of linear models and, at the same time, to learn nonlinear corrections to linear predictions. Therefore, the underlying assumption is that the data can be described by adding nonlinearities over a linear model. Highly nonlinear datasets might not be suitable for the proposed approach. However, in such cases the question itself about interpretability would be ill-posed, at least according to our notion of interpretability. There is however one case that cannot be modeled by Eq. (2). Assume the underlying data generating process contains a self-interacting term of the kind $x_j f(x_j)$, where f is a linear or nonlinear function (e.g. $x\sin(x)$ or x^2). In order to make the linear coefficients interpretable we assumed $g_{\boldsymbol{u}_j}(\boldsymbol{x_{-j}})$ not to depend on x_j, which clearly prevents us from learning such types of nonlinearities. However, if such feature is known to play a role in the dataset, simple feature engineering can solve the issue.

4 Experiments

In this section, we experimentally showcase the proposed approach. Firstly, in Section 4.1 we use a toy example to illustrate how the model works. Secondly, in Section 4.2 we show that across several synthetic datasets the model learns the actual underlying coefficients, the nonlinearities and the correct sparsity in the networks. We also achieve higher performance compared to other hybrid approaches. Lastly, in Section 4.3 we test the model on real datasets and show that it is able to select a sparser set of features than Lasso, while maintaining consistency with Lasso in terms of the signs and magnitudes of the selected coefficients. However, thanks to the nonlinear terms the model achieves significantly higher performance, on par with neural networks but with the benefits of inherent interpretability. In the Appendix we provide further details on each experimental setup.

4.1 Toy example

We first illustrate NIMO on a simple three-dimensional toy example. We generate the data as follows:

$$y = 3 \cdot x_1 \cdot (1 + \tanh(10x_2)) + (-3) \cdot x_2 \cdot (1 + \sin(-2x_1)) + 0 \cdot x_3 + \epsilon \,, \tag{7}$$

where ϵ is the usual Gaussian noise. This toy example has 3 features $\{x_\cdot, x_2, x_3\}$, but x_3 is actually inactive (hence uninformative) since it is multiplied by a zero coefficient. We generate 400 samples and use a 200-100-100 train-validation-test split. As a baseline, we compare with Lasso regression. As shown in Figure 3 (left) the $\boldsymbol{\beta}$ coefficients learned with NIMO coincide with the ground truth. Relevantly, NIMO correctly learns β_3 to be zero, i.e. that x_3 is uninformative. In Figure 3 (middle)

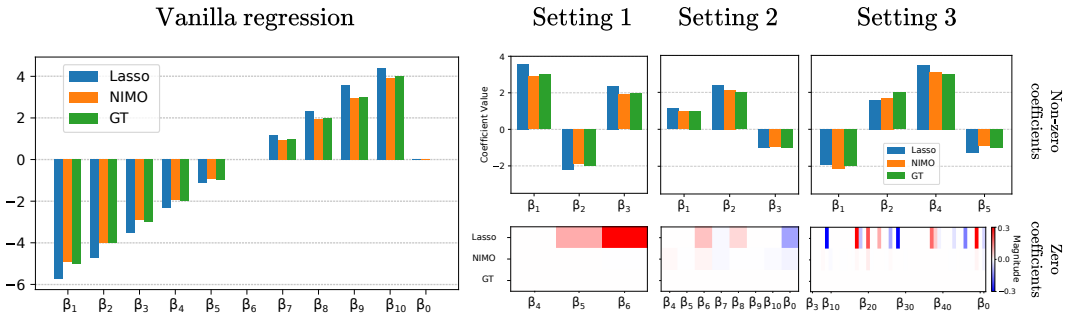


Figure 4: Learned β coefficients on synthetic regression datasets and comparison with Lasso regression and Ground truth. Vanilla regression (left) is purely linear while Settings 1, 2, 3 (right) have different nonlinearities. Both NIMO and Lasso recover the correct nonzero coefficients (informative features) but only NIMO recovers the correct sparsity of the zero coefficients (uninformative features).

we show that the neural network used in NIMO can perfectly recover the underlying nonlinear interactions both for x_1 and x_2. Furthermore, if we analyze the weights of the first `fc` layer, we can see that the neurons connected to the input x_3 are sparse; see Figure 3 (right). This provides further interpretability in the model because it implies that x_3 does not contribute to nonlinear interactions.

4.2 Synthetic Experiments

We now showcase that NIMO is able to recover sparse coefficients and learn the underlying nonlinearities across a set of different regression and classification settings, with different number of features and sparsity levels. We compare out method against Lasso regression (Tibshirani, 1996), a vanilla neural network and two hybrid approaches, namely LassoNet (Lemhadri et al., 2021), Contextual Lasso (Thompson et al., 2023). Contextual Lasso requires the features to be separated in explanatory and contextual, which is not directly applicable to our settings since we do not have contextual features. To still compare with Contextual Lasso, we use all features as context. Compared with other hybrid approaches, NIMO achieves better or equivalent performance while still retaining interpretability. In all settings we use 200 training samples and 100 for validation and test. Further details can be found in the Appendix.

Regression We design synthetic regression datasets by explicitly controlling the linear coefficients and the nonlinearities, similarly to Eq. (7). We explore multiple nonlinearities, different dimensionality and also use several uninformative features (i.e. with zero coefficient) to test if we can recover sparse signals. As a sanity check, we first test NIMO on a *Vanilla regression* dataset with linear coefficients in $\{-5, -4, ..., 3, 4\}$ but without nonlinearities. The idea is to check if the neural network of NIMO interferes with the linear part in the absence of nonlinearities. As shown in Figure 4 (left), NIMO is able to perfectly recover the ground truth coefficients, and the neural network does not interfere with them. We provide three more regression settings (*Setting 1, 2, 3*) where we test different nonlinearities and different sparsity levels. We provide details about the exact data generating process in the Appendix. In Table 1 we report the MSE on the test set over all settings for NIMO and we compare it against linear, nonlinear and hybrid approaches. Overall, NIMO achieves the best performance across all settings with nonlinearities. This is possible because NIMO learns the correct coefficients (Figure 4) and, at the same time, accurately learns the underlying nonlinear interactions. Differently from Lasso, we can also perfectly learn the sparsity pattern in the coefficients, i.e. which features are uninformative; see Figure 4 (bottom, right). In contrast, nonlinear models and hybrid models learn spurious nonlinear relationships.

Classification Our method can be extended to generalized linear models and here we showcase it for logistic regression. In order to create data for synthetic classification we follow a similar procedure as in the regression case, where we directly control the linear coefficients and the nonlinear interactions. The only difference is that we further apply a link function and binarize the output and get the labels. Note that this way we loose information about the original linear coefficients, which cannot be recovered exactly anymore. The sparsity in the coefficients still remains crucial. Also in this case we show various settings for different dimensionality, nonlinear terms and different number of uninformative features. We provide more details about each setting in the Appendix. In Table 2 we

Table 1: MSE loss for synthetic regression settings (see the Appendix for more details). We compare with Lasso regression, NN, LassoNet and Contextual Lasso. In each setting we use 200 samples.

	Features	Lasso	NN	NIMO	LassoNet	Contextual Lasso
Vanilla regr.	10	**0.010**	0.939	0.048	0.015	-
Toy Example	3	18.982	0.446	**0.166**	0.628	0.646
Setting 1	5	3.164	1.109	**0.030**	0.078	0.381
Setting 2	10	3.340	1.482	**0.217**	2.991	3.418
Setting 3	50	13.122	13.718	**0.334**	1.342	13.070

Table 2: Classification accuracy for synthetic classification settings (see the Appendix for more details). We compare with logistic regression, NN and LassoNet. In each setting we use 200 samples.

	Features	Log. Regr.	NN	NIMO	LassoNet
Setting 1	3	0.59	0.90	**0.92**	0.89
Setting 2	10	0.68	0.83	**0.85**	0.82
Setting 3	50	0.74	0.69	0.84	**0.90**

report the classification accuracy on the test set for all methods. Despite our efforts, we were not able to train Contextual lasso on such classification settings, even when all features were given as contextual features. Overall, our model significantly outperforms logistic regression on all settings. When analyzing the learned coefficient, which for brevity we report in the Appendix, we can see that only NIMO recovers the correct sparsity patterns. As argued before, this allows to learn the correct nonlinear interactions. Other nonlinear models and hybrid approaches perform similarly to NIMO but fail to provide the interpretability of the coefficients as in a linear model.

4.3 Real Datasets

We further evaluated our model on two real-world datasets: the diabetes dataset (Efron et al., 2004) and the Boston housing dataset (Belsley et al., 2005). In all the experiments, we use a 60%-20%-20% train-validation-test split. We compare our model with Lasso regression, a standard neural network and LassoNet (Lemhadri et al., 2021) in terms of MSE loss and feature sparsity.

Diabetes dataset The diabetes dataset is a classic regression dataset consisting of 442 patients, each with 10 numeric features, and the target measures disease progression one year after baseline. As shown in Figure 5 (left), we observe that all methods achieve similar MSE losses and the neural network (also the hybird methods) shows no clear advantage on this dataset. Since the diabetes dataset is commonly used for linear regression tasks, it is not surprising that the data does not exhibit complex nonlinear interactions among features. We can verify this claim by looking at the weight sparsity of the first `fc` layer in NIMO, shown in Figure 5 (right). Relevantly, NIMO achieves a

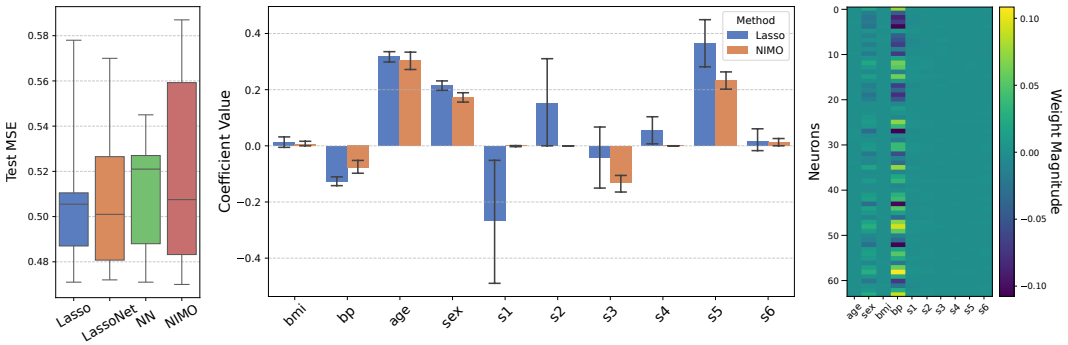


Figure 5: Diabetes datasets results. *Left*: MSE loss on test set. All methods achieve comparable results. *Middle*: Learned linear coefficients by Lasso and NIMO. NIMO selects sparser coefficients ("s1, "s2", "s4"). *Right*: the first `fc` layer of NIMO is very sparse; few nonlinearities are relevant. Errorbars are obtained by running all models across 5 different datasets splits.

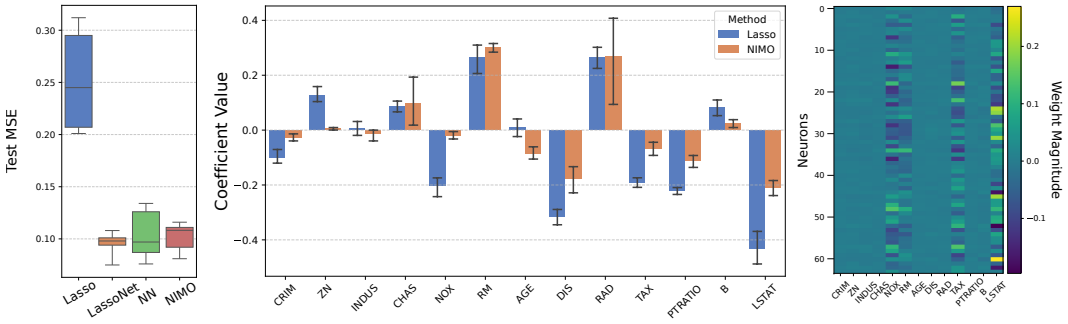


Figure 6: Boston datasets results. *Left*: Comparison of MSE losses. NIMO performs on par with other methods and significantly outperforms Lasso, suggesting nonlinear interactions are crucial. *Middle*: Learned linear coefficients by Lasso and NIMO. The two models learn coefficient with a comparable sparsity. Relevantly, the feature "B" is almost irrelevant for NIMO. *Right*: sparsity in the first `fc` layer of NIMO. Errorbars are obtained by running all models over 5 datasets splits.

significantly sparser solution than Lasso regression, see Figure 5 (middle). This is particularly evident by looking at the coefficients associated with the feature "s1", "s2" and "s4".

Boston dataset The Boston housing dataset contains 506 instances, each with 13 features, and the target is the median value of owner-occupied homes. As shown in Figure 6, our method performs significantly better than Lasso regression and shows compatible MSE test loss with the other nonlinear and hybrid approaches. This suggests that nonlinear interactions are crucial to explain the data. While both the neural network and LassoNet are able to capture those nonlinearities, they both lack the interpretability provided by our model. It is important to note that the Boston housing dataset has been criticized for containing controversial features [1]. The feature of interest is termed "B". From the coefficient plot in Figure 6 (*middle)*, we observe that the significance of the feature "B" is greatly reduced in our model compared to Lasso regression. This surprising result suggests that the reason why such feature appeared to be relevant is due to some artifacts of linear models. Thanks to the decoupling between the linear coefficients and the nonlinearities in NIMO, our results suggests that the role of the feature "B" towards the prediction might be irrelevant. Note that the feature "B" is also not relevant for the nonlinear terms, as shown in Figure 6.

5 Conclusions

In this paper we propose a nonlinear interpretable model. Different from other hybrid approaches, our model retains the interpretability of linear coefficients – just as in standard linear models – while also allowing for nonlinear interaction terms. However, unlike linear models, we are not limited to global linear coefficients that operate at the population level. Our model can also learn local nonlinearities, enabling the generation of modified linear coefficients tailored to individual data points. This can be viewed as an individual correction on top of the linear prediction. In order to effectively train the proposed approach, we propose an optimization algorithm based on profile likelihood and adaptive ridge regression. Specifically, we are able to optimize the neural network parameters via gradient descent while analytically updating the linear coefficients. By means of sparsity in the neural networks we can also automatically select relevant nonlinear interactions. We showcase the proposed approach on synthetic dataset and show that we can learn the underlying coefficients and nonlinearities, outperforming nonlinear and hybrid approaches. We also showcase the model on real world datasets and confirmed the same findings. When the dataset does not contain nonlinear interaction terms we recover sparse coefficients and the network becomes less relevant. When the dataset contains nonlinearities we can learn them with the same accuracy of other nonlinear approaches, but with the significant advantage of learning the interpretable linear coefficients as well.

[1] https://fairlearn.org/main/user_guide/datasets/boston_housing_data.html

References

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: visual explanations from deep networks via gradient-based localization. *International journal of computer vision*, 128:336–359, 2020.

Mike Wu, M Hughes, Sonali Parbhoo, and F Doshi-Velez. Beyond sparsity: Tree-based regularization of deep models for interpretability. In *In: Neural Information Processing Systems (NIPS) Conference. Transparent and Interpretable Machine Learning in Safety Critical Environments (TIML) Workshop*, 2017.

Mike Wu, Sonali Parbhoo, Michael C Hughes, Volker Roth, and Finale Doshi-Velez. Optimizing for interpretability in deep neural networks with tree regularization. *Journal of Artificial Intelligence Research*, 72: 1–37, 2021.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2013.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys*, 16:1–85, 2022.

Christoph Molnar. *Interpretable Machine Learning*. Christoph Molnar, 3 edition, 2025. ISBN 978-3-911578-03-5. URL `https://christophm.github.io/interpretable-ml-book`.

Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *Journal of Machine Learning Research*, 22(127):1–29, 2021.

Ryan Thompson, Amir Dezfouli, and Robert Kohn. The contextual lasso: Sparse linear models via deep neural networks. *Advances in Neural Information Processing Systems*, 36:19940–19961, 2023.

Trevor Hastie and Robert Tibshirani. Varying-coefficient models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 55(4):757–796, 1993. ISSN 00359246.

D. J. Venzon and S. H. Moolgavkar. A method for computing profile-likelihood-based confidence intervals. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 37(1):87–94, 1988. ISSN 00359254, 14679876.

Yves Grandvalet. Least absolute shrinkage is equivalent to quadratic penalization. In *ICANN 98: Proceedings of the 8th International Conference on Artificial Neural Networks, Skövde, Sweden, 2–4 September 1998 8*, pages 201–206. Springer, 1998.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2), April 2004. ISSN 0090-5364. doi: 10.1214/009053604000000067. URL `http://dx.doi.org/10.1214/009053604000000067`.

David A Belsley, Edwin Kuh, and Roy E Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005.

Yves Grandvalet and Stéphane Canu. Outcomes of the equivalence of adaptive ridge with least absolute shrinkage. *Advances in neural information processing systems*, 11, 1998.

Marcello Massimo Negri, Fabricio Arend Torres, and Volker Roth. Conditional matrix flows for gaussian graphical models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL `https://github.com/Lightning-AI/lightning`.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33: 7462–7473, 2020.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

Appendix

A Equivalence of Adaptive Ridge regression and Lasso

The equivalence in the solution of Adaptive ridge regression and Lasso regression is a well-known fact (Tibshirani, 1996; Grandvalet, 1998; Grandvalet and Canu, 1998). Since both our model relies on this fact both in the regression and classification settings, we report its proof below.

Proof. Let $X \in \mathbb{R}^{n \times d}$ be n d-dimensional observations and $\boldsymbol{y} \in \mathbb{R}^n$ be the targets. In a linear model the relationship between target and data is assumed to be linear:

$$\boldsymbol{y} = X\boldsymbol{\beta} + \epsilon \,, \tag{8}$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$ is the coefficient vector and $\epsilon \sim \mathcal{N}(0, 1)$ is standard Gaussian noise. In Ridge Regression a penalized version of the objective is minimized:

$$\min_{\boldsymbol{\beta}} \|X\boldsymbol{\beta} - \boldsymbol{y}\|_2^2 + \lambda \sum_{j=1}^{d} \beta_j^2 \,, \tag{9}$$

where λ weighs the penalty term. In Adaptive Ridge regression, instead of a single λ, a set of feature-specific penalties $\{\nu_i\}_{i=1}^d$ are introduced. The objective then reads as

$$\min_{\boldsymbol{\beta}} \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i \beta_i^2 \quad \text{s. t.} \quad \sum_{i=1}^{d} \frac{1}{\nu_i} = \frac{d}{\lambda} \,, \tag{10}$$

where $\nu_j \geq 0 \; \forall j$ and λ is a predefined value. One can solve the above constrained optimization through the method of Lagrange multipliers:

$$\mathcal{L}(\boldsymbol{\nu}, \mu) = \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i \beta_i^2 + \mu \sum_{i=1}^{d} \frac{1}{\nu_i} \,, \tag{11}$$

where μ is the Lagrangian multiplier and $\boldsymbol{\nu} \coloneqq [\nu_1, \ldots, \nu_d]^T$. The stationary points can be found by taking the gradients with respect to ν_k, as follows:

$$\frac{\partial \mathcal{L}(\boldsymbol{\nu}, \mu)}{\partial \nu_k} = \beta_k^2 - \mu \nu_k^{-2} = 0 \quad \Rightarrow \quad \nu_k \beta_k^2 = \mu / \nu_k \quad \text{or} \quad \nu_k |\beta_k| = \mu^{1/2} \tag{12}$$

We can now use the constraint in Eq. (10):

$$\begin{aligned} \nu_k \beta_k^2 = \mu / \nu_k \quad &\Rightarrow \quad \sum_k \nu_k \beta_k^2 = \sum_k \frac{\mu}{\nu_k} \overset{(10)}{=} \frac{\mu d}{\lambda} \\ &\Rightarrow \quad \mu = \frac{\lambda}{d} \sum_k \nu_k \beta_k^2 = \frac{\lambda}{d} \sum_k \nu_k |\beta_k| \cdot |\beta_k| \overset{(12)}{=} \frac{\lambda}{d} \sum_k |\beta_k| \mu^{1/2} \\ &\Rightarrow \quad \mu^{1/2} = \frac{\lambda}{d} \sum_k |\beta_k| \end{aligned} \tag{13}$$

If we now substitute $\nu_k |\beta_k| = \mu^{1/2}$ in the Adaptive Ridge objective in Eq. (10), we get the Lasso objective:

$$\begin{aligned} f(\boldsymbol{\beta}, \boldsymbol{\nu}) &= \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i \beta_i^2 \\ &= \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i |\beta_i| \cdot |\beta_i| \\ &= \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \mu^{1/2} |\beta_i| \\ &= \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \mu^{1/2} \sum_{i=1}^{d} |\beta_i|. \end{aligned} \tag{14}$$

which gives the relationship with the lasso turns out to be a Lasso regression, with the Lasso penalty parameter at optimum to be $\mu^{1/2} = \frac{\lambda}{d} \sum_k |\beta_k|$. □

B Training through profile likelihood

The proposed model in Eq. (2) has two sets of parameters: the linear coefficients $\boldsymbol{\beta}$ and the neural network parameters $\boldsymbol{u}$. Since the neural network learn the nonlinear corrections over the linear predictions, the two sets of parameters are clearly entangled and in practice optimizing them both with gradient descent is tricky. Furthermore, to increase the interpretability of the model we also encourage sparse coefficients and a sparse first `fc` layer of the the neural network. In this section we show how to make the optimization more effective and, at the same time, how to encourage sparsity. In particular, we propose a novel optimization method based on profile likelihood with adaptive ridge regression. The profile likelihood allows for an analytical update of the linear coefficients while adaptive rigde regression provides sparsity as in Lasso regression. We further show that we can extend such an approach to generalized linear models and we showcase it in detail for logistic regression.

B.1 Regression setting: adaptive ridge regression

Let $\mathcal{L}(\boldsymbol{\beta}, \boldsymbol{u})$ be the joint likelihood in Eq. (3). First, we fix the neural network parameters and analytically solve the maximization problem to obtain $\hat{\boldsymbol{\beta}}$. Then, we substitute $\hat{\boldsymbol{\beta}}$ back into the joint likelihood, which results in the profile likelihood for $\boldsymbol{u}$: $\mathcal{L}(\hat{\boldsymbol{\beta}}, \boldsymbol{u})$. We can then optimize $\boldsymbol{u}$ by minimizing the negative log-profile likelihood using gradient descent. We iterate over these steps until convergence. More specifically, we draw the inspiration from the Adaptive Ridge (AR) regression and exploit its equivalence to Lasso regression (Tibshirani, 1996; Grandvalet, 1998; Grandvalet and Canu, 1998).

We already showed that the proposed model in Eq. (2) can be re-written in matrix form as:

$$f(X) = B_{\boldsymbol{u}}\boldsymbol{\beta} \quad \text{with} \quad B_{\boldsymbol{u}} = X + X \circ G_{\boldsymbol{u}} \tag{15}$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$ are the linear coefficients, $G_{\boldsymbol{u}} \in \mathbb{R}^{n \times d}$ is the neural network output with inputs $X \in \mathbb{R}^{n \times d}$ and "$\circ$" denotes element-wise multiplication. The linear model part is then expressed as:

$$\boldsymbol{y} = B_{\boldsymbol{u}}\boldsymbol{\beta} + \epsilon, \tag{16}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ denotes the targets. Adaptive Ridge regression minimizes the following objective:

$$\min_{\boldsymbol{\beta}} \|B_{\boldsymbol{u}}\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i \beta_i^2 \quad \text{s. t.} \quad \sum_{i=1}^{d} \frac{1}{\nu_i} = \frac{d}{\lambda} \tag{17}$$

where $\nu_j > 0$ are the penalty parameters for each coefficient β_j, and $\lambda > 0$ is a predefined parameter. To avoid divergent solutions we use the re-parameterization employed in Grandvalet (1998); Grandvalet and Canu (1998):

$$\gamma_i = (\nu_i/\lambda)^{1/2}\beta_i, \quad c_i = (\lambda/\nu_i)^{1/2} \, . \tag{18}$$

If we substitute β_i and ν_i into the Adaptive Ridge regression objective, we get the following optimization problem:

$$\min_{\boldsymbol{\gamma}} \|\boldsymbol{y} - B_{\boldsymbol{u}} D_{\boldsymbol{c}} \boldsymbol{\gamma}\|^2 + \lambda \|\boldsymbol{\gamma}\|^2 \quad \text{s. t.} \quad \sum_{i=1}^{d} c_i^2 = d, \quad c_i > 0, \tag{19}$$

where $D_{\boldsymbol{c}}$ is a diagonal matrix whose diagonal elements are given by the vector $\boldsymbol{c} \coloneqq [c_1, \ldots, c_d]^T$. With this re-parameterized problem, we can define the profile likelihood. If we fix $\boldsymbol{c}$ and $\boldsymbol{u}$, $\hat{\boldsymbol{\gamma}}$ can be analytically obtained in closed form:

$$\hat{\boldsymbol{\gamma}} = \left(D_{\boldsymbol{c}} B_{\boldsymbol{u}}^T B_{\boldsymbol{u}} D_{\boldsymbol{c}} + \lambda I\right)^{-1} D_{\boldsymbol{c}} B_{\boldsymbol{u}}^T \boldsymbol{y}, \tag{20}$$

We can then get $\hat{\boldsymbol{\beta}}$ in closed form:

$$\hat{\boldsymbol{\beta}} = \boldsymbol{c} \circ \hat{\boldsymbol{\gamma}} \, , \tag{21}$$

where "$\circ$" denoted element-wise multiplication. We can now substitute $\hat{\boldsymbol{\beta}}$ back into the original problem and take the constraint into consideration. This leads us to a Lagrangian minimization problem in the scaling variable $\boldsymbol{c}$ and neural network parameters $\boldsymbol{u}$:

$$\min_{\boldsymbol{c}, \boldsymbol{u}} \|\boldsymbol{y} - B_{\boldsymbol{u}}\hat{\boldsymbol{\beta}}\|^2 + \tilde{\mu} \sum_{i=1}^{d} \|\boldsymbol{c}\|^2, \tag{22}$$

and we can optimize it through gradient descent. The training algorithm for this re-parameterized adaptive ridge regression is summarized in Alg. 1.

Algorithm 1 Training of Re-parameterized Adaptive Ridge Regression

Require: Training data $X \in \mathbb{R}^{n\times d}$, training target $\boldsymbol{y} \in \mathbb{R}^n$, neural network $g_{\boldsymbol{u}}(\cdot)$, regularization parameter λ, learning rate η, maximum iterations T.

Initialize network parameters $\boldsymbol{u}$ and scaling coefficients $\boldsymbol{c}$ (with $c_i > 0$).
for $t = 1, \ldots, T$ **do**
  $B_{\boldsymbol{u}} = X + X \circ G_{\boldsymbol{u}}$ `// Compute modified design matrix`
  $D_{\boldsymbol{c}} = \mathrm{diag}(\boldsymbol{c})$ `// Form diagonal scaling matrix`
  $\hat{\boldsymbol{\gamma}} = \left(D_{\boldsymbol{c}} B_{\boldsymbol{u}}^T B_{\boldsymbol{u}} D_{\boldsymbol{c}} + \lambda I\right)^{-1} D_{\boldsymbol{c}} B_{\boldsymbol{u}}^T \boldsymbol{y}$ `// Solve for` $\hat{\boldsymbol{\gamma}}$
  $\hat{\boldsymbol{\beta}} = \boldsymbol{c} \circ \hat{\boldsymbol{\gamma}}$ `// Compute regression coefficients`
  $L = \|\boldsymbol{y} - B_{\boldsymbol{u}}\hat{\boldsymbol{\beta}}\|^2 + \tilde{\mu}\sum_{i=1}^d \|\boldsymbol{c}\|^2$ `// Compute loss function`
  $\boldsymbol{c} \leftarrow \boldsymbol{c} - \eta\nabla_{\boldsymbol{c}} L, \quad \boldsymbol{u} \leftarrow \boldsymbol{u} - \eta\nabla_{\boldsymbol{u}} L$ `// Gradient descent updates`
end for
Return $\hat{\boldsymbol{\beta}}, \boldsymbol{c}, \boldsymbol{u}$ `// Output final parameters`

B.2 Logistic regression: Iteratively reweighted least squares

The procedure followed in the previous section can be extended to generalized linear models. Here we showcase how to do it for logistic regression. Logistic regression is a classification model that describes the probability that a binary outcome $y \in \{0, 1\}$ occurs given a feature vector $\boldsymbol{x} \in \mathbb{R}^d$:

$$P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{\beta}) \tag{23}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. The model can then be trained by minimizing the negative log-likelihood. Similarly to the regression case discussed in the previous section, we add an adaptive ridge regression penalty to the objective:

$$\mathcal{L}(\boldsymbol{\beta}) = -\sum_{i=1}^n [y_i \log\sigma(\boldsymbol{x}_i^T\boldsymbol{\beta}) + (1-y_i)\log(1-\sigma(\boldsymbol{x}_i^T\boldsymbol{\beta}))] + \sum_{j=1}^d \nu_j\beta_j^2 \quad \text{s. t.} \quad \sum_{j=1}^d \frac{1}{\nu_j} = \frac{d}{\lambda} \tag{24}$$

Now we can use the same re-parameterization used in the previous section for regression. We get the following optimization problem:

$$\begin{gathered}\min_{\boldsymbol{\gamma}} -\sum_{i=1}^n \left[y_i \log\sigma\left(\sum_{j=1}^p x_{ij}c_j\gamma_j\right) + (1-y_i)\log\left(1-\sigma\left(\sum_{j=1}^p x_{ij}c_j\gamma_j\right)\right)\right] + \lambda\|\boldsymbol{\gamma}\|^2 \\ \text{s.t.} \quad \sum_{j=1}^d c_j^2 = d, \quad c_i > 0\end{gathered} \tag{25}$$

With the reparametrized objective we can consider the profile likelihood by fixing $\boldsymbol{c}$ and update $\boldsymbol{\gamma}$. To update $\boldsymbol{\gamma}$ we use the iteratively reweighted least squares (IRLS) algorithm. The associated update reads as:

$$\boldsymbol{\gamma}^{(t+1)} = (\tilde{X}^T W^{(t)} \tilde{X} + \lambda I)^{-1} \tilde{X}^T W^{(t)} \mathbf{z}^{(t)} \tag{26}$$

where $\tilde{X}$ is the modified design matrix with entries $\tilde{x}_{ij} = c_j x_{ij}$, $W^{(t)}$ is a diagonal weight matrix with entries $w_i^{(t)} = p_i^{(t)}(1-p_i^{(t)}), p_i^{(t)} = \frac{1}{1+e^{-\tilde{\boldsymbol{x}}_i^T\boldsymbol{\gamma}^{(t)}}}$, and $\mathbf{z}^{(t)}$ is the the vector of working response with entries $z_i^{(t)} = \boldsymbol{x}^T\boldsymbol{\beta} + \frac{y_i - p_i^{(t)}}{w_i^{(t)}}$. Note that in our model, our design matrix X contains the nonlinear term as well so we would need to replace X with $B_{\boldsymbol{u}} = X + X \circ G_{\boldsymbol{u}}$. We report the detailed optimization algorithm for the logistic regression case in Alg. 2.

Algorithm 2 IRLS Training for Logistic Regression

Require: Training data $X \in \mathbb{R}^{n \times d}$, target labels $\boldsymbol{y} \in \{0,1\}^n$, neural network $g_{\boldsymbol{u}}(\cdot)$, regularization parameters λ, learning rate η, maximum iterations T.

Initialize parameters $\boldsymbol{u}$, scaling coefficients $\boldsymbol{c}$ (with $c_i > 0$), and regression coefficients $\boldsymbol{\beta}$.
for $t = 1, \ldots, T$ **do**
  $B_{\boldsymbol{u}} = X + X \circ G_{\boldsymbol{u}}$ `// Compute transformed design matrix`
  $D_{\boldsymbol{c}} = \mathrm{diag}(\boldsymbol{c})$ `// Form diagonal scaling matrix`
  $\tilde{X} = B_{\boldsymbol{u}} D_{\boldsymbol{c}}$
  $\hat{\boldsymbol{y}} = B_{\boldsymbol{u}} \boldsymbol{\beta}$ `// Compute logits`
  $\boldsymbol{\pi} = \sigma(\hat{\boldsymbol{y}})$
  $W = \mathrm{diag}(\boldsymbol{\pi} \circ (1 - \boldsymbol{\pi}))$
  $\mathbf{z} = \hat{\boldsymbol{y}} + (\boldsymbol{y} - \boldsymbol{\pi}) / \mathrm{diag}(W)$
  $\boldsymbol{\gamma} = (\tilde{X}^T W \tilde{X} + \lambda I)^{-1} \tilde{X}^T W \mathbf{z}$
  $\boldsymbol{\beta} = \boldsymbol{c} \circ \hat{\boldsymbol{\gamma}}$ `// Update` $\boldsymbol{\beta}$
  $L = \mathrm{BCE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \sum_i c_i^2$ `// Compute total loss`
  $\boldsymbol{c} \leftarrow \boldsymbol{c} - \eta \nabla_{\boldsymbol{c}} L, \quad \boldsymbol{u} \leftarrow \boldsymbol{u} - \eta \nabla_{\boldsymbol{u}} L$ `// Gradient descent step`
end for
Return $\boldsymbol{\beta}, \boldsymbol{c}, \boldsymbol{u}$

B.3 Extension to sub-ℓ_1 pseudo-norms

In some cases sparsity is a crucial aspect in the learning process. One of the limitations of lasso is the well-known over-shrinkage effect, which means that shrinkage of uninformative features could be achieved at the expense of shrinking the retained features as well. One way to avoid this is to use sub-ℓ_1 pseudo-norms, which in the limist of the ℓ_0 pseudo-norm achieve sparsity without any shrinkage. The main challenge with sub-ℓ_1 pseudo-norm is that the resulting objective function becomes non-convex and harder to optimize. However, recent work has successfully used sub-ℓ_1 norms as a prior on regression problems (Negri et al., 2023), which is similar to our setting.

We now show how to use a sub-ℓ_1 prior in our Adaptive Ridge regression objective, which we report below for completeness:

$$\min_{\boldsymbol{\beta}} \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i \beta_i^2 \quad \text{s.t.} \quad \sum_{i=1}^{d} \frac{1}{\nu_i} = \frac{d}{\lambda} \tag{27}$$

In order to generalize to sub-ℓ_1 pseudo-norms we need the following simple modification on ν_i:

$$\sum_{i=1}^{d} \frac{1}{\nu^{\delta}} = C, \quad 0 < \delta \leq 1. \tag{28}$$

We now showcase that we can follow a procedure completely similar to what we had with the original adaptive ridge regression. Fist, we reformulate the objective with the method of Lagrangian multipliers:

$$\mathcal{L}(\boldsymbol{\nu}, \mu) = \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \sum_{i=1}^{d} \nu_i \beta_i^2 + \mu \sum_{i=1}^{d} \frac{1}{\nu_i^{\delta}} \tag{29}$$

Then, we find stationary points for ν_i by taking the gradients and setting them to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}(\boldsymbol{\nu}, \mu)}{\partial \nu_i} = \beta_i^2 - \mu\delta\nu_i^{-(\delta+1)} = 0 \quad &\Rightarrow \quad \nu_i^{\delta+1} = \frac{\mu\delta}{\beta_i^2} \\ &\Rightarrow \quad \nu_i = \left(\frac{\mu\delta}{\beta_i^2}\right)^{\frac{1}{\delta+1}} \end{aligned} \tag{30}$$

The penalty in the original adaptive ridge regression can be rewritten as:

$$\sum_{i=1}^{d} \nu_i \beta_i^2 = \sum_{i=1}^{d} \left(\frac{\mu\delta}{\beta_i^2}\right)^{\frac{1}{\delta+1}} \beta_i^2 \propto \sum_{i=1} \beta_i^{2(1-\frac{1}{\delta+1})}. \tag{31}$$

Note that when $\delta = 1$, we recover the Lasso ℓ_1 norm. When $\delta \to 0$, it approaches the ℓ_0 pseudo-norm, which basically counts the number of nonzero entries. We now apply the same re-parameterization:

$$\gamma_i = (\nu_i/\lambda)^{1/2}\beta_i, \quad c_i = (\lambda/\nu_i)^{1/2} \; . \tag{32}$$

The objective function then reads as

$$\min_{\boldsymbol{\gamma}} \|\boldsymbol{y} - XD_{\boldsymbol{c}}\boldsymbol{\gamma}\|^2 + \lambda\|\boldsymbol{\gamma}\|^2 \quad \text{s.t.} \quad \sum_{i=1}^{d} c_i^{2\delta} = \lambda^{\delta} C, \quad c_i > 0 \; . \tag{33}$$

For each fixed $\boldsymbol{c}$, $\boldsymbol{\gamma}$ can now be obtained in closed form:

$$\hat{\boldsymbol{\gamma}} = \left(D_{\boldsymbol{c}} X^T X D_{\boldsymbol{c}} + \lambda I\right)^{-1} D_{\boldsymbol{c}} X^T \boldsymbol{y}, \tag{34}$$

and $\boldsymbol{\beta}$ can be computed as before by:

$$\hat{\boldsymbol{\beta}} = \boldsymbol{c} \circ \hat{\boldsymbol{\gamma}} \tag{35}$$

Once we compute the $\hat{\boldsymbol{\beta}}$, we can substitute it back into the original objective. Taking the constraint into consideration leads us to a Lagrangian minimization problem in the scaling variable $\boldsymbol{c}$ only:

$$\min_{\boldsymbol{c}} \|\boldsymbol{y} - X\hat{\boldsymbol{\beta}}\|^2 + \tilde{\mu} \sum_{i=1}^{d} c_i^{2\delta}, \tag{36}$$

and we can optimize it through gradient descent.

C NIMO: Implementation details

C.1 Model Formulation

We formulate NIMO as follow:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{d} x_j \beta_j (1 + g_{\boldsymbol{u}_j}(\boldsymbol{x}_{-j})) \quad \text{s.t.} \quad g_{\boldsymbol{u}_j}(\boldsymbol{0}) = 0 \; . \tag{37}$$

The architecture of our model involves two parts, as shown in Fig. 2. The left part is the nonlinear neural network, which operates on the input feature $\boldsymbol{x}$ element-wise. The right part is the linear model, with coefficients β_j multiplied by the neural network output and then summed together.

In practice, each feature component x_j of the input vector $\boldsymbol{x}$ is temporarily masked out to construct $\boldsymbol{x}_{-j}$, which is then passed through a neural network $g_{\boldsymbol{u}_j}$. The output of this network is added to a constant 1 and then multiplied by the original feature value x_j, effectively producing a rescaled version of that feature:

$$\tilde{x}_j = x_j \cdot (1 + g_{\boldsymbol{u}_j}(\boldsymbol{x}_{-j})) \tag{38}$$

This rescaling can be interpreted as an element-wise modulation of the input features based on context. The rescaled features are then linearly combined using fixed coefficients $\boldsymbol{\beta}$ to produce the final prediction:

$$y = \sum_{j=1}^{d} \tilde{x}_j \beta_j.$$

C.2 Practical Implementation

We implement our model with PyTorch Lightning (Falcon and The PyTorch Lightning team, 2019). While in principle each feature x_j could have its own dedicated network $g_{\boldsymbol{u}_j}$, this can be computationally expensive for high-dimensional inputs. To reduce overhead, we instead use a shared neural network $g_{\boldsymbol{u}}$ across all components.

Positional Encoding To ensure the network remains aware of which feature is currently masked out, we append a positional encoding to $\boldsymbol{x}_{-j}$ before passing it through the shared network. We assign each masked input feature $\boldsymbol{x}_{-j}$ a unique binary vector corresponding to its index j, represented in fixed-length binary form using $\lfloor \log_2 d \rfloor + 1$ bits. This compact encoding allows the shared neural network to remain index-aware with minimal overhead.

Group Penalty and Noise Injection The shared network is implemented with 3 fully-connected (`fc`) layers. To encourage the network to select only certain relevant features contributing to the nonlinearity, we apply a group penalty on the first `fc` layer. However, due to the compensation effect from the subsequent layers, the sparsity is difficult to achieve by simply applying this penalty. To mitigate this issue, we inject noise into the output of the first layer by adding scaled Gaussian noise. Concretely, we draw noise from a normal distribution with mean 0 and variance 1, scale it by 0.2, and add it to the output. The noise-injected output is then passed through a `tanh` activation function before being fed into the subsequent layers.

Other Details After the second `fc` layer, a `sin` activation function is applied to capture potentially repeating or high-frequency cyclic patterns in the data (Sitzmann et al., 2020). The learning rate is `1e-3` for synthetic experiments, and `5e-3` for real-world datasets. We choose Adam (Kingma, 2014) as the default optimizer. For regression, we regularize the network also by restricting the output range within $[-1, 1]$, which works well in practice. For classification, we use a less restrictive reduction to the range $[-1, 3]$.

D Experiments

We conduct experiments on both synthetic and real-world datasets. All experiments are conducted on a workstation equipped with a single user-grade NVIDIA GeForce RTX 2080 Ti GPU. The methods we compare with are linear model (Lasso for regression, Logistic Regression for classification), neural network, LassoNet (Lemhadri et al., 2021) and Contextual Lasso (Thompson et al., 2023) (for regression only). Here we provide the details about the experiment setups.

D.1 Implementation details / Hyperparameters

Linear Model We use scikit-learn (Pedregosa et al., 2011) to implement the Lasso and Logistic regression model. More specifically, we use `LassoCV` and `LogisticRegressionCV` with default 5-fold cross validation to select the best model. For `LogisticRegressionCV`, the ℓ_1 penalty is applied.

Neural Network We implement the naive neural network (MLP) with PyTorch Lightning (Falcon and The PyTorch Lightning team, 2019). For a fair comparison, the MLP has the same structure as the nonlinear part in our NIMO model, except no positional encoding. We use Adam (Kingma, 2014) as the default optimizer. The learning rate is `1e-3` for synthetic experiments, and `5e-3` for real-world datasets. Strong dropout regularization (`p=0.6`) is used to avoid overfitting.

LassoNet We use the original implementation of LassoNet from their Github repository [2]. Using the same API as shown in the LassoNet document, we configure the model to have a structure and number of parameters comparable to our model. All other hyperparameters are kept at their default values. The best model is automatically selected along the entire regularization path, as demonstrated in their paper. The one with the smallest validation loss is chosen.

Contextual Lasso We get the code of Contextual Lasso from the original Github repository [3], which was implemented in Julia language. We follow the examples in the documentation and configure the model to have a structure and number of parameters comparable to our own model.

D.2 Synthetic experiments: Setup

We create the synthetic datasets by first sampling the features, and then compute the targets according to different nonlinear patterns. Random Gaussian noises are added to the targets. The feature values are sampled uniformly between -2 and 2. For logistic regression, we pass the targets through a logistic (sigmoid) function to get probabilities between 0 and 1, and then sample the final binary labels using a Bernoulli distribution with those probabilities. In all the settings, we use 200 training samples and 100 samples each for validation and test.

[2] https://github.com/lasso-net/lassonet

[3] https://github.com/ryan-thompson/ContextualLasso.jl

Settings for regression All features are indexed starting from 1. $\epsilon \sim \mathcal{N}(0, 0.1^2)$ is a Gaussian noise.

- Setting 0 (toy example), $n = 200, p = 3$

$$\begin{aligned} y = &+ 3 \cdot x_1 \cdot [1 + \tanh(10x_2)] \\ &- 3 \cdot x_2 \cdot [1 + \sin(-2x_1)] \\ &+ \epsilon \end{aligned} \tag{39}$$

- Setting 1, $n = 200, p = 5$

$$\begin{aligned} y = &+ 3 \cdot x_1 \cdot [1 + (2\sigma(x_2 x_3) - 1)] \\ &- 2 \cdot x_2 \\ &+ 2 \cdot x_3 \\ &+ \epsilon \end{aligned} \tag{40}$$

- Setting 2, $n = 200, p = 10$

$$\begin{aligned} y = &+ 1 \cdot x_1 \cdot [1 + \tanh(x_2 x_3 + \sin(x_4))] \\ &+ 2 \cdot x_2 \cdot [1 + \sin(2x_1)] \\ &- 1 \cdot x_3 \cdot \left[1 + \frac{2}{\pi} \arctan(x_2 x_4)\right] \\ &+ \epsilon \end{aligned} \tag{41}$$

- Setting 3, $n = 200, p = 50$

$$\begin{aligned} y = &- 2 \cdot x_1 \cdot [1 + \tanh(x_2 x_4)] \\ &+ 2 \cdot x_2 \cdot \left[1 + \frac{2}{\pi} \arctan(x_4 - x_5)\right] \\ &+ 3 \cdot x_4 \cdot [1 + \tanh(x_2 + \sin(x_5))] \\ &- 1 \cdot x_5 \cdot [1 + (2\sigma(x_1 x_4) - 1)] \\ &+ \epsilon \end{aligned} \tag{42}$$

- Setting 4, $n = 200, p = 10$

$$\begin{aligned} y &= -5 \cdot x_1 - 4 \cdot x_2 - 3 \cdot x_3 - \cdots + 0 \cdot x_6 \\ &+ 1 \cdot x_7 + 2 \cdot x_8 + \cdots + 4 \cdot x_{10} \\ &+ \epsilon \end{aligned} \tag{43}$$

Settings for classification For all the settings, we generate y first, and then generate labels through the following procedure:

$$\pi = \frac{1}{1 + e^{-y}}, \quad \text{label} \sim \text{Bernoulli}(\pi)$$

- Setting 1, $n = 200, p = 3$

$$\begin{aligned} y = &+ 2 \cdot x_1 \cdot [1 + 2\tanh(x_2)] \\ &- 2 \cdot x_2 \cdot [1 + 3\sin(2x_1) + \tanh(2x_1)] \\ &+ 1 + \epsilon \end{aligned} \tag{44}$$

- Setting 2, $n = 200, p = 10$

$$\begin{aligned} y = &+ 10 \cdot x_1 \cdot [1 + 2 \cdot \tanh(2x_2) + \sin(x_4)] \\ &+ 20 \cdot x_2 \cdot [1 + 2 \cdot \cos(2x_1)] \\ &- 20 \cdot x_3 \cdot [1 + 2 \cdot \arctan(x_2 x_4)] \\ &+ 10 \cdot x_4 \\ &- 10 + \epsilon \end{aligned} \tag{45}$$

- Setting 3, $n = 200, p = 50$

$$
\begin{aligned}
y = & -20 \cdot x_1 \cdot [1 + \tanh(x_2 x_4)] \\
& + 20 \cdot x_2 \cdot \left[1 + \frac{2}{\pi} \arctan(x_4 - x_5)\right] \\
& + 30 \cdot x_4 \cdot [1 + \tanh(x_2 + \sin(x_5))] \\
& - 10 \cdot x_5 \cdot [1 + (2\sigma(x_1 x_4) - 1)] \\
& + \epsilon
\end{aligned} \tag{46}
$$

D.3 Synthetic experiments: Additional results

Below, we present additional experimental results for the synthetic settings described above.

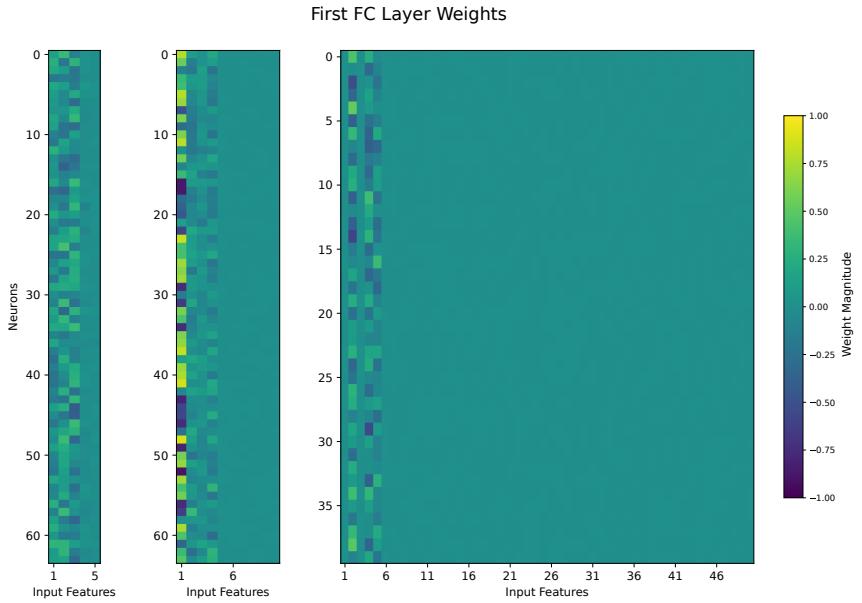


Figure 7: Sparsity and feature selection in the first `fc` layer of NIMO for regression settings. *Left*: Setting 1. *Middle*: Setting 2. *Right*: Setting 3.

Regression The advantage of our model is that it not only recovers the underlying linear coefficients but also selects sparse features for the neural network, thanks to the noise injection and group penalty. In the main paper, we presented the recovered linear coefficients. Here, we show the sparse features selected by the neural network in Figure 7. We observe that even though all features are input to the neural network, only a small subset of features is selected. Moreover, these selected features align with those involved in the nonlinearity computation in our settings, which demonstrates the model's ability to effectively identify and focus on the most relevant features for the task.

Classification In classification, the presence of the link function can obscure the underlying signal, so while a linear decision boundary may suffice, recovering the true generative coefficients is more difficult than in regression, where the model directly learns to approximate the target surface. Across all settings, the goal is not to recover the precise magnitudes of the ground truth coefficients, but to accurately identify their sparsity pattern and directional effects (i.e., signs). We compare the recovered coefficients from logistic regression and our model across three classification settings, as shown in Figure 8. Since recovering the exact magnitudes of the coefficients is not feasible, we normalize all coefficients, including the ground truth, by dividing them by the maximum absolute value among the coefficients, to enable a fair comparison.

We observe that logistic regression struggles to identify the relevant features and their correct signs. In contrast, our model generally succeeds in selecting the correct features, and the neural network

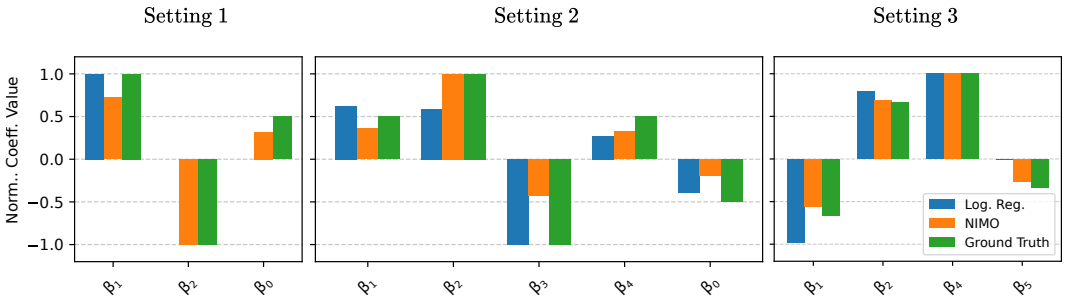


Figure 8: Learned coefficients for synthetic classification settings. We compare NIMO and Logistic regression with the ground truth. Since we use binarized labels it is impossible to retrieve the exact coefficients. Instead, we compare normalized coefficients, such that the maximum is always 1.

component effectively captures sparse but important features that contribute to nonlinearity. It is worth noting that NIMO also selects some noisy features, especially in Setting 3. However, our experiments are primarily intended as a proof of concept to demonstrate that our model can be easily extended to generalized linear models. With careful hyperparameter tuning, NIMO is capable of selecting more accurate features. The sparse features selected by the neural network are shown in Figure 9.

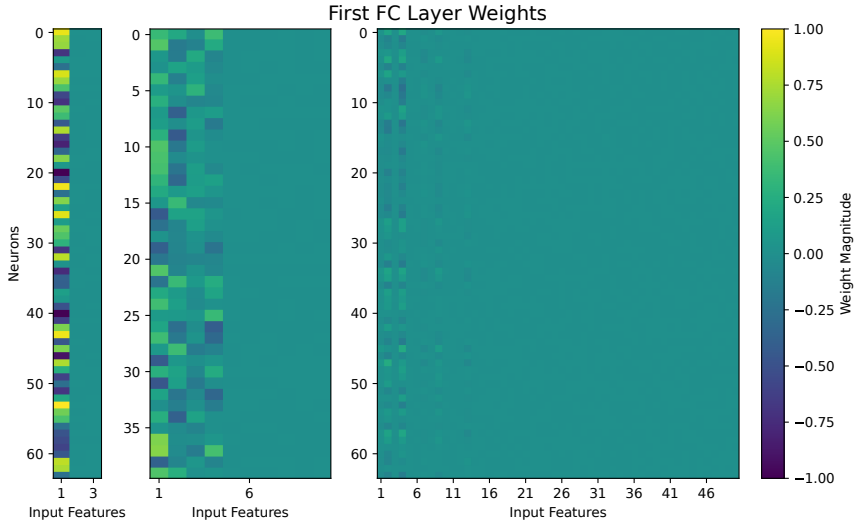


Figure 9: Sparsity and feature selection in the first `fc` layer of NIMO for classification settings. *Left*: Setting 1. *Middle*: Setting 2. *Right*: Setting 3.

D.4 Real-dataset experiments

We conduct experiments on two real-world datasets: the diabetes dataset (Efron et al., 2004) and the Boston housing dataset (Belsley et al., 2005). Each dataset is randomly split into training (60%), validation (20%), and test (20%) sets, repeated over 5 independent runs. For each split, we perform a grid search to select the optimal penalty parameter based on the validation loss (or validation accuracy for classification tasks). Error bars and confidence intervals are reported across these different splits.