Neural Additive Models: Interpretable Machine Learning with Neural Nets

Rishabh Agarwal[*]
Google Research, Brain Team

Nicholas Frosst[‡]
Cohere

Xuezhou Zhang
University of Wisconsin-Madison

Rich Caruana
Microsoft Research

Geoffrey E. Hinton
Google Research, Brain Team

Abstract

Deep neural networks (DNNs) are powerful black-box predictors that have achieved impressive performance on a wide variety of tasks. However, their accuracy comes at the cost of intelligibility: it is usually unclear how they make their decisions. This hinders their applicability to high stakes decision-making domains such as healthcare. We propose Neural Additive Models (NAMs) which combine some of the expressivity of DNNs with the inherent intelligibility of generalized additive models. NAMs learn a linear combination of neural networks that each attend to a single input feature. These networks are trained jointly and can learn arbitrarily complex relationships between their input feature and the output. Our experiments on regression and classification datasets show that NAMs are more accurate than widely used intelligible models such as logistic regression and shallow decision trees. They perform similarly to existing state-of-the-art generalized additive models in accuracy, but can be more easily applied to real-world problems.

1 Introduction

Deep neural networks are powerful function approximators that have achieved impressive results on a variety of tasks, such as computer vision [He et al., 2016], language modeling [Radford et al., 2018] and reinforcement learning [Silver et al., 2016]. However, it is notoriously difficult to understand how they make their predictions, and they are often considered as black-box models. This hinders their applicability to high-stakes domains such as healthcare and criminal justice.

Various efforts have been made to demystify the predictions of neural networks (NNs). For example, one family of methods, represented by LIME [Ribeiro et al., 2016], attempt to *explain* individual predictions of a neural network by approximating it locally with interpretable models such as linear models and shallow trees[2]. However, these approaches often fail to provide a global view of the model and their explanations often are not faithful to what the original model computes or do not provide enough detail to understand the model's behavior [Rudin, 2019]. In this paper, we make restrictions on the *structure* of neural networks, which yields a family of models called Neural Additive Models (NAMs), that are inherently interpretable while suffering little loss in prediction accuracy when applied to tabular data.

Methodologically, NAMs belong to a larger model family called Generalized Additive Models (GAMs) [Hastie and Tibshirani, 1990]. GAMs have the form:

$$g(\mathbb{E}[y]) = \beta + f_1(x_1) + f_2(x_2) + \cdots + f_K(x_K) \tag{1}$$

[*]Correspondence to: Rishabh Agarwal <rishabhagarwal@google.com>, Rich Caruana <rcaruana@microsoft.com>. [‡] Work done at Google Research, Brain Team.

[2]Linear models, shallow decision trees and GAMs are interpretable only if the features they are trained on are interpretable.

Preprint. Under review.

The idea to use neural networks as the non-linear function approximators in GAMs is not new. However, it hasn't been explored much in recent years, and I believe it is an interesting space. Especially when it comes to designing the structure of those networks beyond basic multi-layer perceptrons.

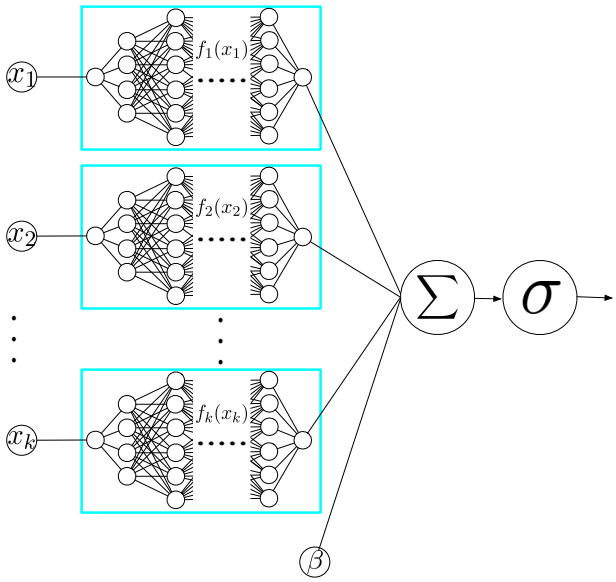


Figure 1: Neural Additive Model architecture for binary classification. Each input variable is handled by a different neural network. This results in easily interpretable yet highly accurate models.

where $\mathbf{x} = (x_1,\ x_2,\ \ldots,\ x_K)$ is the input with K features, y is the target variable, $g(.)$ is the link function (*e.g.,* logistic function) and each f_i is a univariate shape function with $\mathbb{E}[f_i] = 0$. Generalized linear models, such as logistic regression, are a special form of GAMs where each f_i is restricted to be linear.

NAMs learn a linear combination of networks that each attend to a single input feature: each f_i in (1) is parametrized by a neural network. These networks are trained jointly using backpropagation and can learn arbitrarily complex shape functions. Interpreting NAMs is easy as the impact of a feature on the prediction does not rely on the other features and can be understood by visualizing its corresponding shape function (*e.g.,* plotting $f_i(x_i)$ *vs.* x_i).

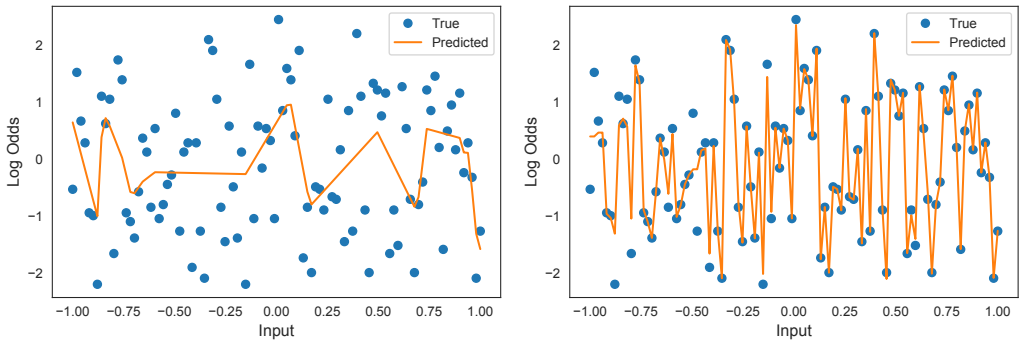
Traditionally, GAMs were fitted using smooth low-order splines, which reduce overfitting and can be fit analytically. More recently, GAMs [Caruana et al., 2015] were fitted with boosted decision trees to improve accuracy and to allow GAM models to learn jumps in the feature shaping functions to better match patterns seen in real data that smooth splines could not easily capture. This paper examines using deep neural nets to fit GAMs (NAMs). NAMs provide the following advantages:

- NAMs introduce an expressive yet intelligible class of models to the deep learning community, a much larger community than the one using tree-based GAMs. We will open-source the code in popular deep learning frameworks to maximize their adoption.
- Once NAMs are widely available, they are likely to be combined with other deep learning methods in ways we don't foresee. This is important because one of the key drawbacks of deep learning is intelligibility.
- The graphs learned by NAMs are not just an explanation but an exact description of how NAMs compute a prediction. This could help harness the expressivity of neural nets on high-stakes domains with intelligibility requirements.
- NAMs, due to the flexibility of NNs, can be easily extended to various settings problematic for boosted decision trees. For example, extending boosted tree GAMs to multitask, multi-class or multi-label learning requires significant changes to how trees are trained, but is easily accomplished with NAMs without requiring changes to how neural nets are trained.
- NAMs are more scalable as they can be trained on GPUs or other specialized hardware using the same toolkits developed for deep learning over the past decade – GAMs currently cannot.
- Accurate GAMs [Caruana et al., 2015] currently require millions of decision trees to fit each shape function while NAMs only use a small ensemble (10 - 100) of neural nets. Furthermore, the extra accuracy DNNs demonstrate on many learning problems might yield better accuracy and intelligibility for NAMs over existing GAM algorithms.

GAMs historically allow feature interaction if specified for a non-linear function. Hence this additional restriction is imposed by the authors.

This explains the self-imposed restriction above. It's about maximizing interpretability. While feature interaction can still be "explained", it does add ambiguity.

The focus of this paper is on learning "jagged" functions rather than "smooth" functions. I find this differentiation between ReLU and ExU odd, especially when seeing how [figure 9] is a 3-layer ReLU DNN that learns a fairly jagged function. The difference for ExU as described here is that it's more capable of overfitting to jagged functions. It's a good demonstration that these new types of units can accomplish jagged fits, but the results in [figure 9] make me hesitant to believe ExU is inherrantly useful compared to ReLU.



(a) (b)

Figure 2: **Overfitting the toy dataset**: Training predictions learned by a single hidden layer neural network with 1024 (a) standard ReLU, and (b) ReLU-n with ExU hidden units trained for 10,000 epochs on the binary classification dataset described in Section 2.1. We can see that the ReLU network has learned a fairly smooth function while the ExU network has learned a very jumpy function. We find that a DNN with multiple hidden layers also learned smooth functions (see Figure 9).

2 Neural Additive Models

2.1 Fitting Jagged Shape Functions

Modeling jagged functions is required to learn accurate additive models as there are often sharp jumps in real-world datasets, *e.g.,* see Figure 6 for jumps in graphs for PFRatio and Bilirubin which correspond to real patterns in the MIMIC-II dataset (Section 3.1.2). Similarly, Caruana et al. [2015] observe that GAMs fit using splines tend to over regularize and miss genuine details in real data, yielding less accuracy than tree-based GAMs. Therefore, we require that neural networks (NNs) are able to learn highly non-linear shape functions, to fit these patterns.

Why do we want to overfit jagged functions? The beginning of this section does a great job explaining why we want to learn them. This demonstrates ExU can easily overfit jagged functions. Why should that make me confident in the methodology?

Although NNs can approximate arbitrarily complex functions [Hornik et al., 1989], we find that standard NNs fail to model highly jumpy 1D functions, and demonstrate this failure empirically using a toy dataset. The toy dataset is constructed as follows: For the input x, we sample 100 evenly spaced points in [-1, 1]. For each x, we sample p uniformly random in [0.1, 0.9) and generate 100 labels from a Bernoulli random variable which takes the value 1 with probability p. This creates a binary classification dataset of (x, y) tuples with 10,000 points. Figure 2 shows the log-odds of the empirical probability p (*i.e.,* $\log \frac{p}{1-p}$) of classifying the label of x as 1 for each input x. This dataset tests the NN's ability to "overfit" the data, rather than its ability to generalize.

Over-parameterized NNs with ReLUs [Nair and Hinton, 2010] and standard initializations such as Kaiming initialization [He et al., 2015] and Xavier initialization [Glorot and Bengio, 2010]) struggle to overfit this toy dataset when trained using mini-batch gradient descent, despite the NN architecture being expressive enough[3] (see Figures 2(a) and 9). This difficulty of learning large local fluctuations with ReLU networks without affecting their global behavior when fitting jagged functions might be due to their bias towards learning smooth functions [Rahaman et al., 2018, Arpit et al., 2017].

We propose *exp-centered* (ExU) hidden units to overcome this neural net failure: we simply learn the weights in the logarithmic space with inputs shifted by a bias. Specifically, for a scalar input x, each hidden unit using an activation function f computes $h(x)$ given by

$$h(x) = f\left(e^{w} * (x - b)\right) \tag{2}$$

where w and b are the weight and bias parameters. The intuition behind ExU units is as follows: For modeling jagged functions, a hidden unit should be able to change its output significantly, with a tiny change in input. This requires the unit to have extremely large weight values depending on the sharpness of the jump. The ExU unit computes a linear function of input where the slope can be very steep with small weights, making it easier to modify the output easily during training. ExU units do not improve the expressivity of neural nets, however they do improve their learnability for fitting jumpy functions.

This is the main contribution of the paper, and a great explanation for the intuition and reason behind the design of ExU units.

Unsure what this means. If they improve the ability to express jumpy functions, how doesn't that improve NN expressivity?

[3] This problem doesn't occur with full-batch gradient descent.

If this problem doesn't occur with ReLU on full-batch gradient descent, I would like to understand why. In many real-world scenarios, we have to use mini-batches, so this doesn't undermine the contribution. However, it needs further explanation.

If ExU units with standard weight initializations struggle comparably to ReLU, it is also only fair to test the same initialization strategy for ReLU, but I don't see that mentioned here.

The intuition behind this initialization strategy is extremely clever. If you want to learn jagged start from a random, yet jagged, initialization!

I'm curious where the normal distribution with mean between 3 and 4 and standard deviation of 0.5 comes from. The initialization strategy makes sense, but why those numbers?

We noticed that ExU units with standard weight initialization also struggle to learn jagged curves; instead initializing the weights using a normal distribution $\mathcal{N}(x, 0.5)$ with $x \in [3, 4]$ works well in practice. This initialization simply ensures that the initial network starts with a jagged (random) function which we empirically find to be crucial for fitting any jumpy function.

Furthermore, we use ReLU activations capped at n (ReLU-n) [Krizhevsky, 2010] to ensure that each ExU unit is active in a small input range, making it easier to model sharp jumps in a function without significantly affecting the global behavior. ExU-units can be combined with any activation function (*i.e.,* any f can be used in (2)), but ReLU-n performs well in practice. Figure 2(b) shows that NNs with ExU units are able to fit the toy dataset significantly better than standard NNs.

2.2 Regularization and Training

Given my comments above, I believe regularization will be the most important part of the pipeline that makes ExU work.

ExU units encourage learning highly jagged curves, however, most realistic shape functions tend to be smooth with large jumps at only a few points. To avoid overfitting, we use the following regularization techniques:

This is a really clever solution to the described problem. However, dropping out full feature networks here could make the model extremely different in training versus inference. It's saying "don't depend on feature Xi". But what if the solution truly does depend on a specific feature?

- **Dropout** [Srivastava et al., 2014]: It regularizes ExUs in each feature net, allowing them to learn smooth functions while being able to represent jumps (Figure 3).
- **Weight decay**: This is done by penalizing the L2 norm of weights in each feature net.
- **Output Penalty**: We penalize the L2 norm of the prediction of each feature net, so that its contribution stays close to zero unless evident otherwise from the data.
- **Feature Dropout**: We also drop out individual feature networks during training. When there are correlated input features, an additive model can possibly learn multiple explanations by shifting contributions across these features. This term encourages NAMs to spread out those contributions.

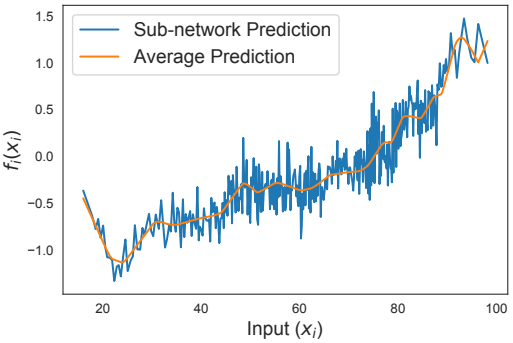


Figure 3: **Regularizing ExU networks.** Output of a ExU feature net trained with dropout = 0.2 for the age feature in the MIMIC-II dataset. Predictions from individual subnets (as a result of dropping out hidden units) are much more jagged than the average predictions using the entire feature net.

Training. Let $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ be a training dataset of size N, where each input $\mathbf{x} = (x_1, x_2, \ldots, x_K)$ contains K features and y is the target variable. In this work, we train NAMs using the loss $\mathcal{L}(\theta)$ given by

$$\mathcal{L}(\theta) = \mathbb{E}_{x,y\sim\mathcal{D}}\big[l(x, y; \theta) + \lambda_1 \eta(x; \theta)\big] + \lambda_2 \gamma(\theta) \tag{3}$$

where $\eta(x;\theta) = \frac{1}{K}\sum_x \sum_k (f_k^\theta(x_k))^2$ is the output penalty, $\gamma(\theta)$ is the weight decay and f_k^θ is the feature network for the k^{th} feature. Each individual network is also regularized using feature dropout and dropout with coefficients λ_3 and λ_4 respectively. $l(x, y; \theta)$ is the task dependent loss function. We use the cross-entropy loss for binary classification:

$$l(x, y; \theta) = -y \log(p_\theta(x)) - (1 - y) \log(1 - p_\theta(x)),$$

where $p_\theta(x) = \sigma\big(\beta^\theta + \sum_{k=1}^{K} f_k^\theta(x_k)\big)$ and mean squared error (MSE) for regression:

$$l(x, y; \theta) = \big(\beta^\theta + \sum_{k=1}^{K} f_k^\theta(x_k) - y\big)^2$$

2.3 Intelligibility and Modularity

The intelligibility of NAMs results in part from the ease with which they can be visualized. Because each feature is handled independently by a learned shape function parameterized by a neural net, one can get a full view of the model by simply graphing the individual shape functions. For data with

a small number of inputs, it is possible to have an accessible explanation of the model's behavior visualized fully on a single page. Please note these shape function plots are not just an explanation but an *exact* description of how NAMs compute a prediction.

We set the average score for each graph (*i.e.,* each feature) averaged across the entire training dataset to zero by subtracting the mean score. To make individual shape functions identifiable and modular, a single bias term is then added to the model so that the average predictions across all data points matches the observed baseline. This makes interpreting the contribution of each term easier: *e.g.,* on binary classification tasks, negative scores decrease probability, and positive scores increase probability compared to the baseline probability of observing that class. This property also allows each graph to be removed from the NAM (zeroed out) without introducing bias to the predictions.

3 Experiments

Baselines. We compare NAMs with the following baselines on two regression and three classification datasets:

- **Logistic/Linear Regression**: Prevalent, simple, intelligible models. This comparison tells us how much gain we get from capturing non-linear relationships between individual features and the target using NAMs. We use the *sklearn* implementation [Pedregosa et al., 2011], and tune the hyperparameters with grid search.
- **Decision Trees**: This comparison shows the performance benefits of NAMs which are more intelligible than decision trees unless the trees are very small. We use the *sklearn* implementation [Pedregosa et al., 2011], and tune the hyperparameters with grid search.
- **Explainable Boosting Machines (EBMs)**: Current state-of-the-art GAMs [Caruana et al., 2015, Lou et al., 2012] which use gradient boosting of shallow bagged trees that cycle one-at-a-time through the features. We use the open-source implementation [Nori et al., 2019] with the parameters specified by prior work [Caruana et al., 2015] for a fair comparison.
- **Deep Neural Networks (DNNs)**: Unrestricted, full-complexity models which can model higher-order interaction between the input features. This gives us a sense of how much accuracy we sacrifice in order to gain interpretability with NAMs. We train DNNs with 10 hidden layers containing 100 units each with ReLU activation using the Adam optimizer. This architecture choice ensures that this network had the capacity to achieve perfect training accuracy on datasets used in our experiments. We use weight decay and dropout to prevent overfitting and tune hyperparameters using a similar protocol as NAMs.
- **Gradient Boosted Trees**: Another class of full-complexity models that provides an upper bound on the achievable test accuracy in our experiments. We use the XGBoost implementation [Chen and Guestrin, 2016] and tune the hyperparameters using random search.

For each dataset, we select the feature nets amongst (1) DNNs containing 3 hidden layers with 64, 64 and 32 units and ReLU activation, and (2) single hidden layer NNs with 1024 ExU units and ReLU-1 activation. We perform 5-fold cross validation to evaluate the accuracy of the learned models. To measure performance, we use area under the precision-recall curve (AUC) for binary classification (as the datasets are unbalanced) and root mean-squared error (RMSE) for regression. More details about training and evaluation can be found in Section A.2 in the supplementary material.

Why only single-hidden-layer NNs when using ExU units? And what criteria is used to select architecture for feature nets? This isn't described as a hyperpatameter in the appendix.

Visualization. We plot each shape function and the corresponding data density on the same graph. Specifically, we plot each learned shape function $f_k(x_k)$ *vs.* x_k for an ensemble of NAMs using a semi transparent blue line, which allows us to see when the models in the ensemble learned the same shape function and when they diverged. This provides a sense of the confidence of the learned shape functions. We also plot on the same graphs the normalized data density, in the form of pink bars. The darker the shade of pink, the more data there is in that region. This allows us to know when the model had adequate training data to learn appropriate shape functions.

3.1 Classification

3.1.1 COMPAS: Risk Prediction in Criminal Justice

COMPAS is a proprietary score developed to predict recidivism risk, which is used to inform bail, sentencing and parole decisions and has been the subject of scrutiny for racial bias [Angwin et al.,

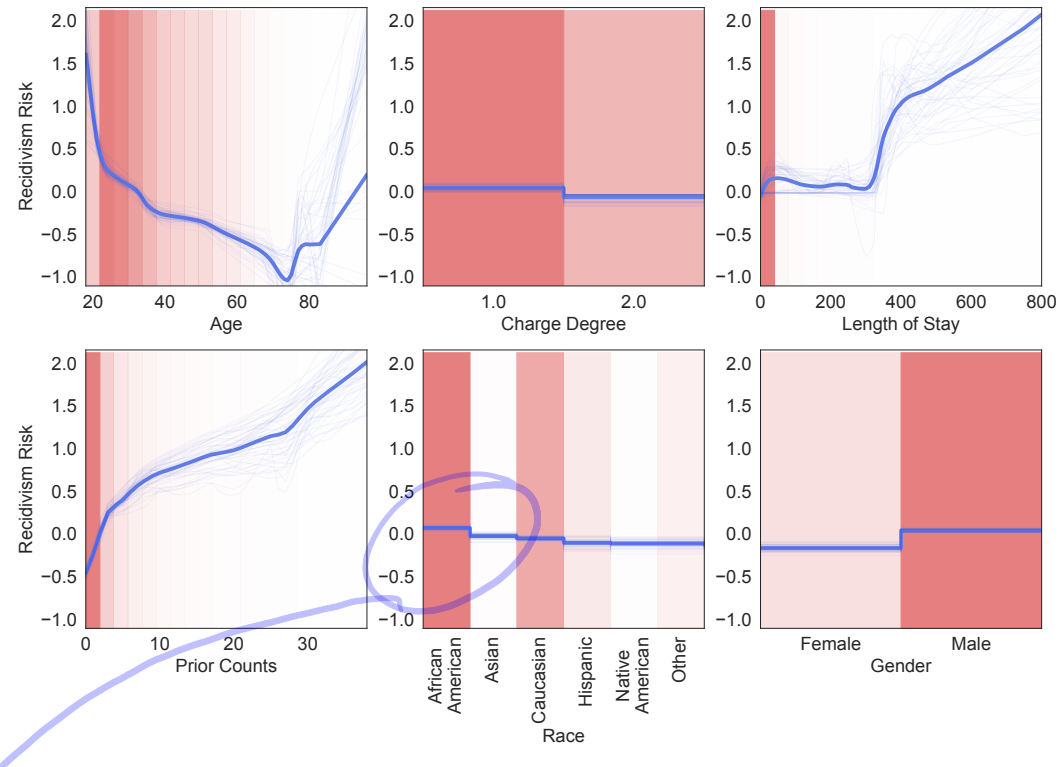


Figure 4: **COMPAS Recidivism Prediction**. These plots show the individual shape functions learned by an ensemble of hundred NAMs for each input feature along with the data density. The thin blue lines represent different shape functions from the ensemble to show the agreement of the members of the ensemble. The pink bars represent the normalized data density for each feature. The darker the bar the more data there is with that value.

2016, Dressel and Farid, 2018, Tan et al., 2018]. In 2016, ProPublica released recidivism data[4] on defendants in Broward County, Florida along with the predictions from the COMPAS model.

Here, we ask whether this training dataset is biased using the transparency of NAMs. Figure 4 shows the learned NAM which is as accurate as black-box models on this dataset. The shape function for race indicates that the learned NAM may be racially biased: black defendants are predicted to be higher risk for reoffending than white or Asian defendants. This suggests that the recidivism training dataset may be racially-biased. The modularity of NAMs allow this bias to be easily corrected to a certain extent; we can simply remove the contributions learned using the race attribute by zeroing out its mean-centered graph (*i.e.,* set the feature to zero in the learned NAM). Although this would drop the AUC score as we are removing a discriminative feature, it may be a more fair model to use for making bail decisions.

Discussion. Machine learning (ML) models trained on data will learn any biases in the data: *e.g.,* ML for recidivism prediction will learn race bias if the data has a race bias. It is important to keep the potentially offending attributes in the model during training so that the bias can be detected and then removed after training. If the offending variables are eliminated before training, it makes debiasing the model more difficult: if the offending attributes are correlated with other training attributes, the bias is likely to spread to those attributes. The transparency and modularity of NAMs allows one to detect unanticipated biases in data and makes it easier to correct the bias in the learned model.

This is really important. The idea that we can train a transparent model, inspect for fairness, and have a mathematically sound way to remove unfair functions, is lovely. This doesn't solve the problem, but is an important discussion!

3.1.2 MIMIC-II: Mortality Prediction in ICUs

Figure 5 shows 16 of the shape functions learned by the Neural Additive Model for the MIMIC-II dataset [Saeed et al., 2011] to predict mortality in intensive care unit (ICUs). (The 17^{th} graph for Admission Type is flat and we omit it to save space.) The plot for HIV/AIDS shows that patients

[4] https://github.com/propublica/compas-analysis

I thought this looked odd, and I'm happy they addressed that, upon consultation with doctors, it is expected.

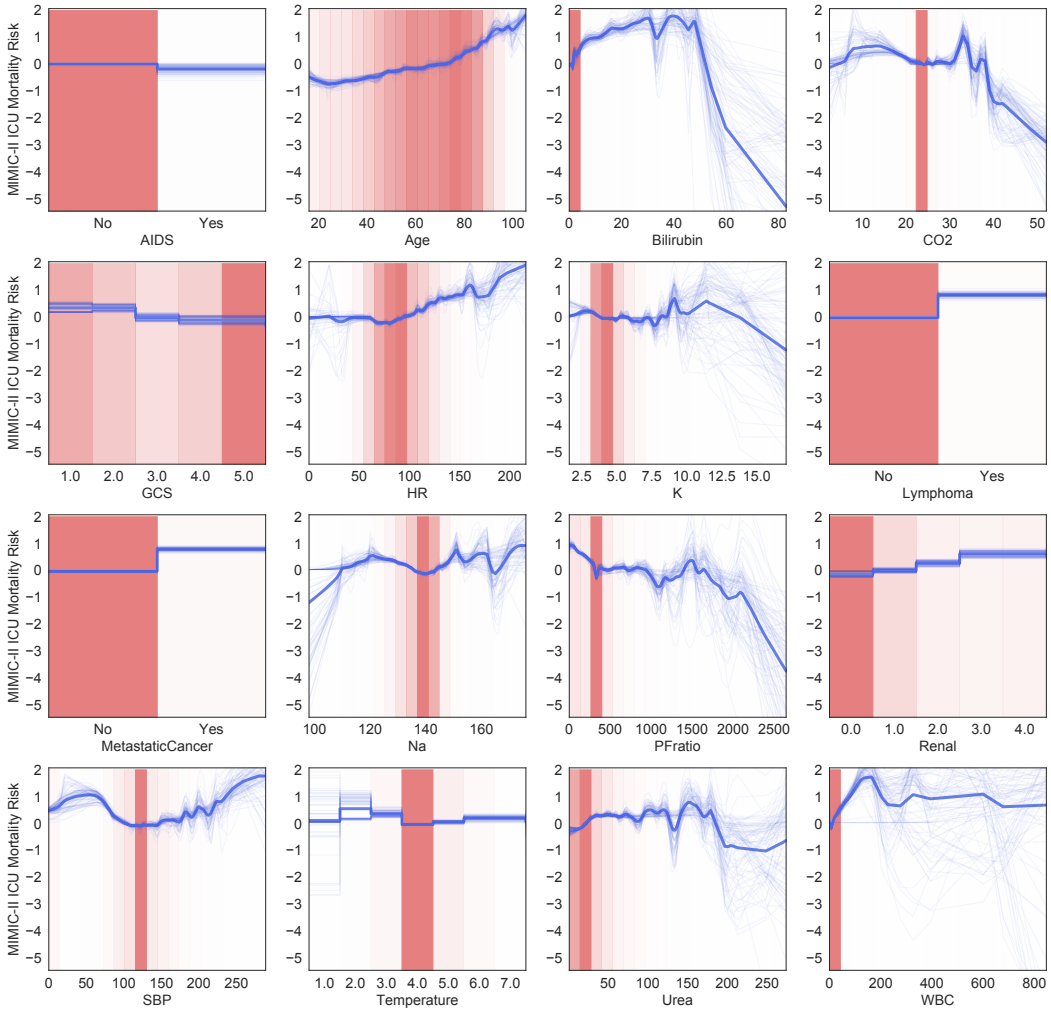


Figure 5: **MIMIC-II ICU Mortality**. NAM shape functions learned on the MIMIC-II dataset to predict mortality risk using medical features (shown on the x-axis) collected during the stay in the ICU. Low values on the y-axis indicates a low risk of mortality.

with AIDS have less risk of ICU mortality. While this might seem counter-intuitive, we confirmed with doctors that this is probably correct: among the various reasons why one might be admitted to the ICU, AIDS is a relatively treatable illness and is one of the less risky reasons for ICU admission. In other words, being admitted to the ICU for AIDS suggests that the patient was not admitted for another riskier condition, and thus the model correctly predicts that the patient is at less risk than other non-AIDS patients.

The shape plot for Age shows, as expected, that mortality risk tends to increase with age, with the most rapid rise in risk happening above age 80. There is detail in the graph that is interesting and warrants further study such as the small increase in risk at ages 18 and 19, and the bump in risk at age 90 — jumps in risk that happen at round numbers are often due to social effects.

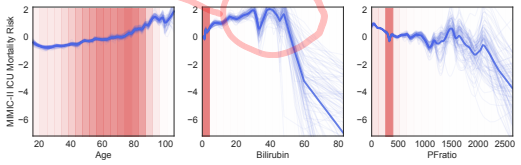
The shape plot for Bilirubin (a by product of the breakdown of red blood cells) shows that risk is low for normal levels below 2-3, and rises significantly for levels above 15-20, until risk drops again above 50. There is also a surprising drop in risk near 35 that requires further investigation. We believe the drop in risk above 50 is because patients above 50 begin to receive dialysis and other critical care and these treatments are very effective. The drop in risk that occurs for Urea above 175 is also likely due to dialysis.

The plot for the Glasgow Coma Index (GCS) is monotone decreasing as would be expected: higher GCS indicates less severe coma. Note that NAMs are not biased to learn monotone functions such

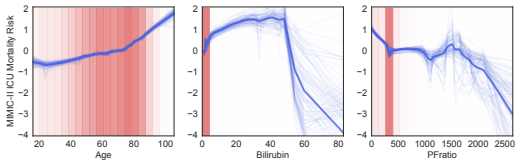
as this and the shape of the plot is driven by the data. The NAM also learns a monotone increasing shape plot for risk as a function of renal function. This, too, is as expected: 0.0 codes for normal renal function and 4.0 indicates severe renal failure.

The NAM has learned that risk is least for normal heart rate (HR) in the range 60-80, and that risk rises as heart rate climbs above 100. Also, as expected, both Lymphoma and Metastatic Cancer increase mortality risk. The CO2 graph shows low risk for the normal range 22-24.

The shape plot for PFratio (a measure of the effectiveness of converting O2 in air to O2 in blood) shows a drop at PFratio = 332 which upon further inspection is due to missing values in PFratio being imputed with the mean: because most patients do not have their PFratio measured, the highest density of patients are actually missing their PFratio which was then imputed with the mean value of 332. One way to detect that imputed missing values are responsible for a dip (or rise) in a shape plot is when risk at the mean value of the attribute suddenly drops (or rises) to a risk level similar to what the model learns for patients who are considered normal/healthy in this dimension: the jump happens at the mean when imputation is done with the mean value, and the level jumps towards the risk of normal healthy patients because often the variable was not measured because the patients were considered normal (for this attribute), a medical assessment which is often correct.



(a) Graphs learned by NAMs with ExU units



(b) Graphs learned by NAMs with standard units

Figure 6: **ExU *vs.* standard** hidden units NAMs. On MIMIC-II, NAMs trained with ExU units learn jumpier graphs than with standard hidden units while achieving a similar AUC (≈ 0.829). Ensembling these two types of NAMs further improves performance (≈ 0.830).

These figures reduce my confidence in the usefulness of ExU units. If performance is similar, and ensembling barely improves performance, why do we care? Sure, the ExU plots are "jumpier" than the standard ones, but they're not different enough to convince me that the jumps are anything more than overfitting in the training data.

Normal temperature is coded as 4.0 on the temperature plot, and risk rises for abnormal temperature above and below this value. It's not clear if the non-monotone risk for hypothermic patients with temperatures 1 or 2 is due to variance, an unknown problem with the data, or an unexplained but real effect in the training signals and warrants further investigation. Similarly, the shape plot for Systolic Blood Pressure (SBP) shows lowest risk for normal SBP near 120, with risk rising for abnormally elevated or depressed SBP. The jumps in risk that happen at 175, 200, and 225 are probably due to treatments that doctors start to apply at these levels: the risk rises to left of these thresholds as SBP rises to more dangerous levels but before the treatment threshold is reached, and then drops a little to the right of these thresholds when most patients above the treatment threshold are receiving a more aggressive treatment that is effective at lowering their risk.

I find this "require further investigation" sentence upsetting. Looking at standard vs. ExU plots, this drop for Bilirubin near 35 only noticeably occurs in the ExU network. If this drop is truly important, it's a great datapoint to suggest that ExU can learn something important that standard units cannot learn. But since it's not explored here, the opportunity to convince me is missed.

Discussion. In summary, most of what the NAM has learned appears to be consistent with medical knowledge, though a few details on some of the graphs (e.g., the increase in risk for young patients, and the drop in risk for patients with Bilirubin near 35) require further investigation. NAMs are attractive models because they often are very accurate, while remaining interpretable, and if a detail in some graph is found to be incorrect, the model can be edited by re-drawing the graph. However, NAMs (like all GAMs), are not causal models. Although the shape plots can be informative, and can help uncover problems with the data that might need correction before deploying the models, the plots do not tell us *why* the model learned what it did, or what the impact of intervention (*e.g.*, actively lowering a patient's fever or blood pressure) would be. The shape plots do, however, tell us *exactly* how the model makes its predictions.

3.1.3 Credit Fraud: Financial Fraud Detection

This is a large dataset [Dal Pozzolo, 2015] containing 284,807 transactions made by European credit cardholders where the task is to predict whether a given transaction is fraudulent or not. It is highly unbalanced and contains only 492 frauds (0.172% of the entire dataset) of all transactions. Table 1 shows that on this dataset, NAMs outperform EBMs and perform comparably to the XGBoost baseline. This shows the benefit of using NAMs instead of tree-based GAMs and suggests that NAMs can provide highly accurate and intelligible models on large datasets. NAMs using ExU units perform much better compared to NAMs with standard DNNs (AUC ≈ 0.974).

Table 1: AUC on the classification datasets for different learning methods. Each cell contains the mean AUC ± one standard deviation obtained via 5-fold cross validation. Higher AUCs are better.

Model	COMPAS	MIMIC-II	Credit Fraud
Logistic Regression	0.730± 0.014	0.791 ± 0.007	0.975 ± 0.010
Decision Trees	0.723 ± 0.010	0.768 ± 0.008	0.956 ± 0.004
NAMs	0.741 ± 0.009	0.830 ± 0.008	0.980 ± 0.002
EBMs	0.740 ± 0.012	0.835 ± 0.007	0.976 ± 0.009
XGBoost	0.742 ± 0.009	0.844 ± 0.006	0.981 ± 0.008
DNNs	0.735 ± 0.006	0.832 ± 0.009	0.978 ± 0.003

note that NAMs with standard DNNs (AUC of 0.974) underperforms normal DNNs, but NAMs with ExU units outperforms normal DNNs

Table 2: RMSE on regression datasets for different learning methods. Each cell contains the mean RMSE ± one standard deviation obtained via 5-fold cross validation. Lower RMSE is better.

Model	California Housing	FICO Score
Linear Regression	0.728 ± 0.015	4.344 ± 0.056
Decision Trees	0.720 ± 0.006	4.900 ± 0.113
NAMs	0.562 ± 0.007	3.490 ± 0.081
EBMs	0.557 ± 0.009	3.512 ± 0.095
XGBoost	0.532 ± 0.014	3.345 ± 0.071
DNNs	0.492 ± 0.009	3.324 ± 0.092

3.2 Regression

3.2.1 California Housing: Predicting Housing Prices

This is the first example that I believe actually takes advantage of ExU units given the desire for jumps away from a baseline in very specific longitudes. I would be curious to see corresponding plot using normal hidden units. unfortunately, this is not contained in the appendix.

California Housing dataset [Pace and Barry, 1997] is a canonical machine learning dataset derived from the 1990 U.S. census to understand the influence of community characteristics on housing prices. The task is regression to predict the median price of houses (in million dollars) in each California district. The learned NAM considers the median income as well as the house location (latitude, longitude) as the most important features (we omit the other six graphs to save space, see Figure 10 in appendix). As shown by Figure 7, the house prices increase linearly with median income in high data density regions. Furthermore, the graph for longitude shows sharp jumps in price prediction around 122.5°W and 118.5°W which roughly correspond to San Francisco and Los Angeles respectively.

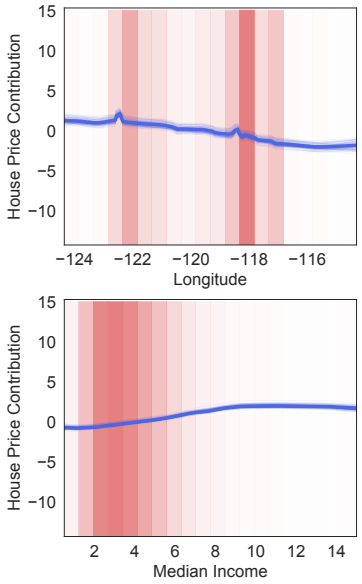
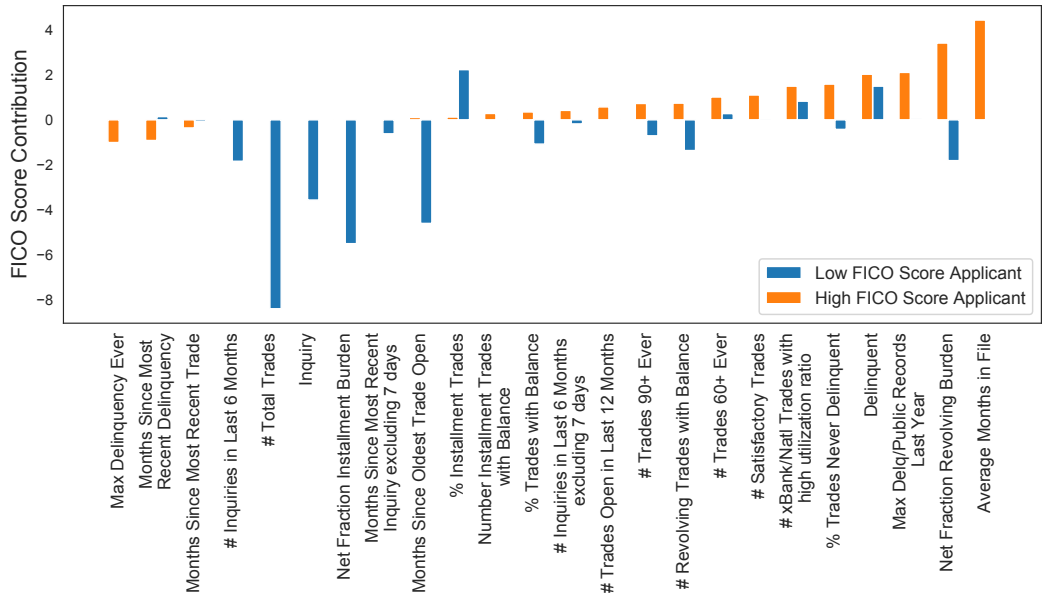


Figure 7: **California Housing**. Graphs learned by NAMs to predict house prices for the 2 main features.

3.2.2 FICO Score: Understanding Credit Scores

The FICO score is a widely used proprietary credit score to determine credit worthiness for loans in the United States. The FICO dataset [FICO, 2018] is comprised of real-world anonymized credit applications made by customers and their assigned FICO Score, based on their credit report information. We visualize the feature contributions of a NAM trained using the FICO dataset (see Figure 11 in appendix) for two applicants (Table 6) with low and high scores respectively.

Figure 8 shows that the most important features for the high scoring applicant are (1) Average Months on File and (2) Net Fraction Revolving Burden (*i.e.,* percentage of credit limit used) which take the value 235 months and 0% respectively. This makes sense, as generally, the longer a person's credit history, the better it is for their credit score. Although there is a strong inverse correlation between Net Fraction Revolving Burden and the score, it is positively correlated for small values (< 10). This means that making use of some credit increases your credit score, but using too much of it is bad. We are confident in this interpretation because most of the data density is in small values of Net Fraction Revolving Burden, and each NAM in the ensemble displays a similar shape function (Figure 11).



This is an interesting application of NAMs as they can be much more than just feature-wise-MLPs. This specific implementation could have many many possible approaches, and is somewhat like a simple feature-wise embedding layer. Unfortunately, I couldn't find the implementation of this in the GitHub repository corresponding to this paper.

Figure 8: **Understanding individual predictions**. Feature contribution using the learned NAMs for predicting scores of two different applicants in the FICO dataset. For a given input, each feature net in the NAM acts as a lookup table and returns a contribution term. These contributions are combined in a modular way: they are added up, and passed through a link function to compute the prediction.

For the low scoring applicant, the main contributing factors are (1) Total Number of Trades[5] and (2) Net Fraction Installment Burden (Installment balance divided by original loan amount) which take the values 57 and 68% respectively. This applicant used their credit quite frequently and has a large burden, thus resulting in a low score.

4 Related Work

This section should mention some of the other papers fitting non-linear functions in GAMs with NNs. Yes, GANNs are where this concept was introduced, but a quick googling yields multiple papers that should be mentioned in this section. It's not popular, but it's also not new.

That being said, this paper does not claim to be the first to do this, so it's not entirely disingenuous. NAMs have a novel structure for learning interpretable functions that likely have real-world non-differentiable (jumpy) components. That is the contribution, and it's a good one (even though I didn't find this paper sufficiently convincing that ExU achieves this to an impressive standard. It's still useful).

The Generalized Additive Neural Networks (GANNs) proposed by Potts [1999] are somewhat similar to the Neural Additive Models we propose here. Like our NAMs, GANNs used a restriction in the neural network architecture to force the net to learn additive functions of individual input features. GANNs, however, predate deep learning and use a single hidden layer, did not use backpropagation [Rumelhart et al., 1986] and required human-in-the-loop evaluation and were not successful in training accurate GAM models with neural nets.

GANNs begin with subnets containing a single hidden unit for each input feature, and use a human-in-the-loop process to add (or subtract) hidden units to the architecture based on human evaluation of plotted partial residuals. This means that the training procedure cannot be automated. In practice, the laborious manual effort required to evaluate all of the partial residual plots to decide what to do next, and then retrain the model after adding or subtracting hidden units from the architecture meant that GANN nets remained very small and simple — typically only one hidden unit per feature.

In contrast, the NAMs we develop in this paper benefit from the advances in deep learning. They use a large number of hidden units and multiple hidden layers per input feature subnet to allow more complex, more accurate shape functions to be learned. Finally, NAMs use a specific hidden unit structure to allow subnets to learn the more non-linear functions often required for accurate GAM models, and then form an ensemble of these nets to provide uncertainty estimates, further improve accuracy and reduce the high-variance that can result from encouraging the model to learn highly non-linear functions. Taken together, these methods allow NAMs to train GAM models with sufficient representational power to achieve state-of-the-art accuracy while remaining intelligible.

Prior to NAMs, the state-of-the-art in high-accuracy, interpretable GAM models [Hastie and Tibshirani, 1990, Guisan et al., 2002] are the GAM [Lou et al., 2012] and GA^{2}M [Lou et al., 2013] models

[5] Credit trades refer to any agreement between a lending agency and consumers.

based on regularized boosted decision trees which were successfully applied to healthcare datasets by Caruana et al. [2015]. We compare the accuracy of NAMs to these models in Section 3.

5 Conclusion

We present Neural Additive Models (NAMs), a new class of models which combines the intelligibility of GAMs with the expressivity of DNNs. We find that NAMs, which don't model any higher-order feature interactions, are comparable in performance to DNNs on a variety of tabular datasets. In addition, NAMs outperform widely used intelligible models and perform similarly to state-of-the-art GAMs while being more easily applicable to a broader set of real-world problems and straightforward to combine with deep learning techniques.

6 Acknowledgments

We would like to thank Kevin Swersky for reviewing an early draft of the paper. We also thank Sarah Tan for providing us with pre-processed versions of some of the datasets used in the paper. RA would also like to thank Marlos C. Machado and Marc G. Bellemare for helpful discussions.

References

Julia Angwin, Jeff Larson, Lauren Kirchner, and Surya Mattu. Machine Bias: There's software used across the country to predict future criminals. And it's biased against blacks, 2016. URL https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing. [Accessed February 1, 2020].

Devansh Arpit, Stanislaw Jastrzkebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. *ICML*, 2017.

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. *SIGKDD*, 2015.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. *SIGKDD*, 2016.

Andrea Dal Pozzolo. Adaptive machine learning for credit card fraud detection. *PhD Thesis, Department of Computer Science, Université Libre de Bruxelles*, 2015.

Julia Dressel and Hany Farid. The accuracy, fairness, and limits of predicting recidivism. *Science advances*, 2018.

FICO. FICO Explainable Machine Learning Challenge. https://community.fico.com/s/explainable-machine-learning-challenge, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, 2010.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. *SIGKDD*, 2017.

Antoine Guisan, Thomas C Edwards Jr, and Trevor Hastie. Generalized linear and generalized additive models in studies of species distributions: setting the scene. *Ecological modelling*, 2002.

Trevor Hastie and Robert Tibshirani. *Generalized Additive Models*. Chapman and Hall/CRC, 1990.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CVPR*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

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

Alex Krizhevsky. Convolutional deep belief networks on cifar-10. 2010.

Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. *SIGKDD*, 2012.

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. *SIGKDD*, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. *ICML*, 2010.

Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 1997.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 2011.

William JE Potts. Generalized additive neural networks. *SIGKDD*, 1999.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. *ICML*, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. *SIGKDD*, 2016.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 2019.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 1986.

Mohammed Saeed, Mauricio Villarroel, Andrew T Reisner, Gari Clifford, Li-Wei Lehman, George Moody, Thomas Heldt, Tin H Kyaw, Benjamin Moody, and Roger G Mark. Multiparameter intelligent monitoring in intensive care ii (mimic-ii): a public-access intensive care unit database. *Critical care medicine*, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 2016.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *NeurIPS*, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

Sarah Tan, Rich Caruana, Giles Hooker, and Yin Lou. Distill-and-compare: Auditing black-box models using transparent model distillation. *AAAI/ACM Conference on AI, Ethics, and Society*, 2018.

A Supplementary Material

A.1 Fitting Jagged Curves

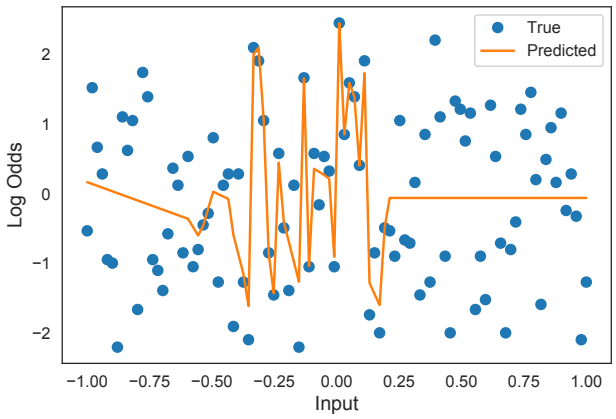


Figure 9: **Toy classification**: Deep neural network with 3 hidden layers of size 64, 64 and 32 respectively with ReLU activation and Xavier initialization trained for 10,000 epochs on toy classification dataset described in Section 2.1. We use a batch size of 1024 with the Adam optimizer and a learning rate decay of 0.995 every epoch. The learning rate was tuned in [1e-3, 1e-1) and we show the results with the best learning rate.

A.2 Experimental Details

Training and Hyperparameter Details. The NAM feature networks (f_k^θ) are trained jointly using the Adam optimizer [Kingma and Ba, 2014] with a batch size of 1024 for a maximum of 1000 epochs with early stopping using the validation dataset. The learning rate is annealed by a factor of 0.995 every training epoch. For all the tasks, we tune the learning rate, output penalty coefficient (λ_1), weight decay coefficient (λ_2), dropout rate (λ_3) and feature dropout rate (λ_4). For computational efficiency, we tune these hyperparameters using Bayesian optimization [Snoek et al., 2012, Golovin et al., 2017] based on cross-validation performance with a single train-validation split for each fold.

Evaluation. We perform 5-fold cross validation to evaluate the accuracy of the learned models. To measure performance, we use area under the precision-recall curve (AUC) for binary classification (as the datasets are unbalanced) and root mean-squared error (RMSE) for regression. For NAMs and DNNs, one of the 5 folds (20% data) is used as a held-out test set while the remaining 4 folds are used for training (70% data) and validation (10% data). The training and validation splits are randomly subsampled from the 4 folds and this process is repeated 20 times. For each run, the validation set is used for early stopping. For each fold, we ensemble the NAMs and DNNs trained on the 20 and EBMs on 100 train-validation splits respectively to make the prediction on the held-out test set.

A.3 Hyperparameters

We use a batch size of 1024 with the Adam optimizer and a learning rate decay of 0.995 every epoch in our experiments for NAMs and DNNs.

Why are we searching dropout > 0.5?

NAMs. We tune the dropout coefficient (λ_3) in the discrete set {0, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}, weight decay coefficient (λ_2) in the continuous interval $[0.000001, 0.0001)$, learning rate in the interval $[0.001, 0.1)$, feature dropout coefficient (λ_4) in the discrete set {0, 0.05, 0.1, 0.2} and output penalty coefficient (λ_1) in the interval $[0.001, 0.1)$. Note that the weight decay is implemented as the average weight decay over the individual feature networks in NAMs. Refer to Table 4 and Table 3 for hyperparameters found on regression and classification datasets.

DNNs. We tune the dropout coefficient (λ_3) in {0, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5}, weight decay coefficient (λ_2) in the continuous interval $[0.0000001, 0.1)$ and learning rate in the interval $[0.001, 0.1)$.

Very interesting that they explored hyperparameters of ExU versus normal ReLU, and have ExU performing better only in the classification tasks of MIMIC-II and Credit Fraud. Is there something about these problems that inherently benefit from jumpy functions? Is there something about the feature-set that are best represented with non-linear functions with non-differentiable points?

Table 3: Optimal hyperparameters found for NAMs on regression datasets. "Hidden units" shows the number of hidden layers as well as the number of neurons used in each layer for each feature network.

Hyperparameter	FICO	Housing
Learning Rate	0.0161	0.00674
Output Penalty (λ_1)	0.0205	0.001
Weight Decay (λ_2)	1.07 x 10^{-5}	10^{-6}
Dropout	0.0	0.0
Feature Dropout	0.0	0.0
Hidden units	64, 64, 32	64, 64, 32
Activation	ReLU	ReLU

Table 4: Optimal hyperparameters found for NAMs on classification datasets. "Hidden units" shows the number of hidden layers as well as the number of units used in each hidden layer for each feature network.

Hyperparameter	COMPAS	MIMIC-II	Credit Fraud
Learning Rate	0.02082	0.005	0.0157
Output Penalty	0.2078	0.3	0.0
Weight Decay	0.0	9.6 x 10^{-5}	4.95 x 10^{-6}
Dropout	0.1	0.2	0.8
Feature Dropout	0.05	0.0	0.0
Hidden units	64, 64, 32	1024	1024
Activation	ReLU	ExU	ExU

A.4 Additional Graphs and Tables

Table 5: **FICO Score**. Meaning of the different attributes of the feature "Max Delq/Public Records Last Year".

Value	Meaning
0	Derogatory comment
1	120+ days delinquent
2	90 days delinquent
3	60 days delinquent
4	30 days delinquent
5,6	Unknown delinquent
7	Current and never delinquent
8,9	All other

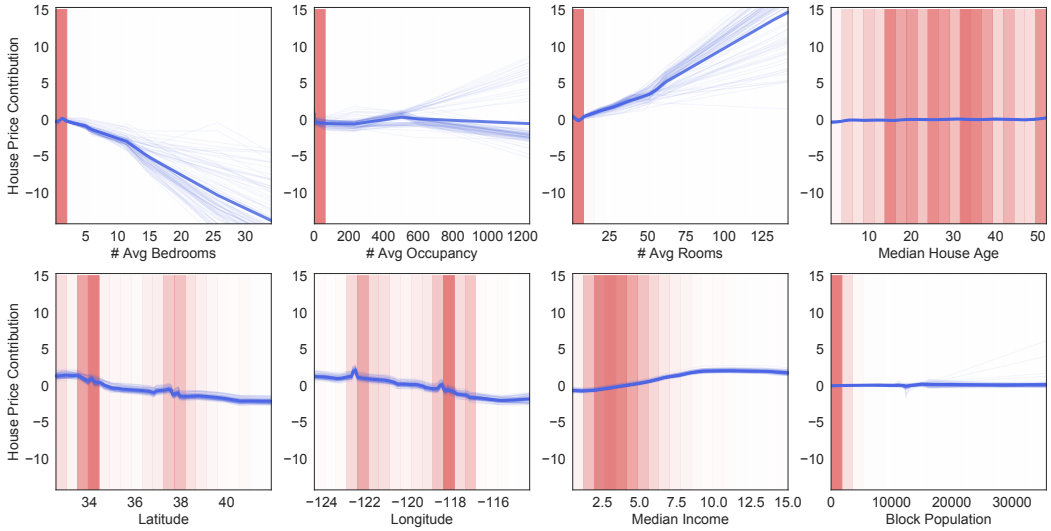


Figure 10: **California Housing**. Graphs learned by NAMs trained to predict house prices (regression) on the California Housing dataset. These plots show the individual shape functions learned by an ensemble of hundred NAMs for each input feature as well as the data density. The thin blue lines represents different shape functions from the ensemble to show the agreement of the members of the ensemble. The pink bars represent the normalized data density for each feature. The darker the bar the more data there is with that value.

Table 6: Feature attributes for the two individuals shown in Figure 8

Feature	High Score Applicant	Low Score applicant
Months Since Oldest Trade Open	417.0	174.0
Months Since Most Recent Trade	25.0	1.0
Average Months in File	235.0	66.0
# Satisfactory Trades	9.0	44.0
# Trades 60+ Ever	0.0	11.0
# Trades 90+ Ever	0.0	8.0
% Trades Never Delinquent	100.0	70.0
Months Since Most Recent Delinquency	0.0	3.0
Max Delq/Public Records Last Year	6.0	0.0
Max Delinquency Ever	6.0	0.0
# Total Trades	9.0	57.0
# Trades Open in Last 12 Months	0.0	5.0
% Installment Trades	22.0	66.0
Months Since Most Recent Inquiry excluding 7 days	0.0	0.0
# Inquiries in Last 6 Months	1.0	7.0
# Inquiries in Last 6 Months excluding 7 days	0.0	6.0
Net Fraction Revolving Burden	0.0	23.0
Net Fraction Installment Burden	0.0	68.0
# Revolving Trades with Balance	1.0	2.0
Number Installment Trades with Balance	0.0	5.0
# Bank/Natl Trades with high utilization ratio	0.0	0.0
% Trades with Balance	40.0	64.0
Delinquent	0.0	1.0
Inquiry	1.0	1.0

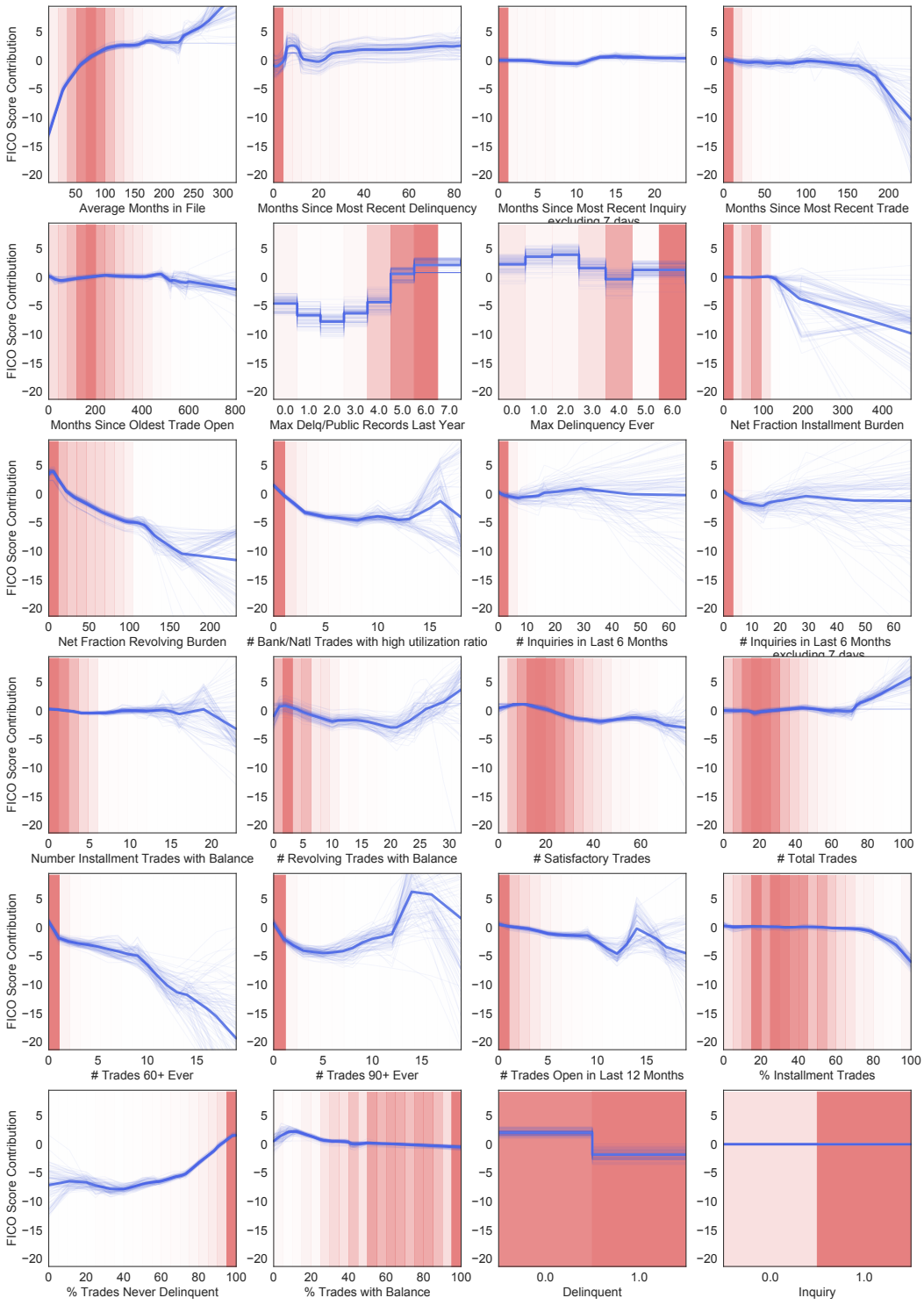


Figure 11: **FICO Score Prediction**. Graphs learned by NAMs trained to predict FICO scores (regression) based on their credit report information. These graphs can be interpreted easily, *e.g.,* the second last graph in the bottom row shows that being delinquent on your payments decreases your credit score.

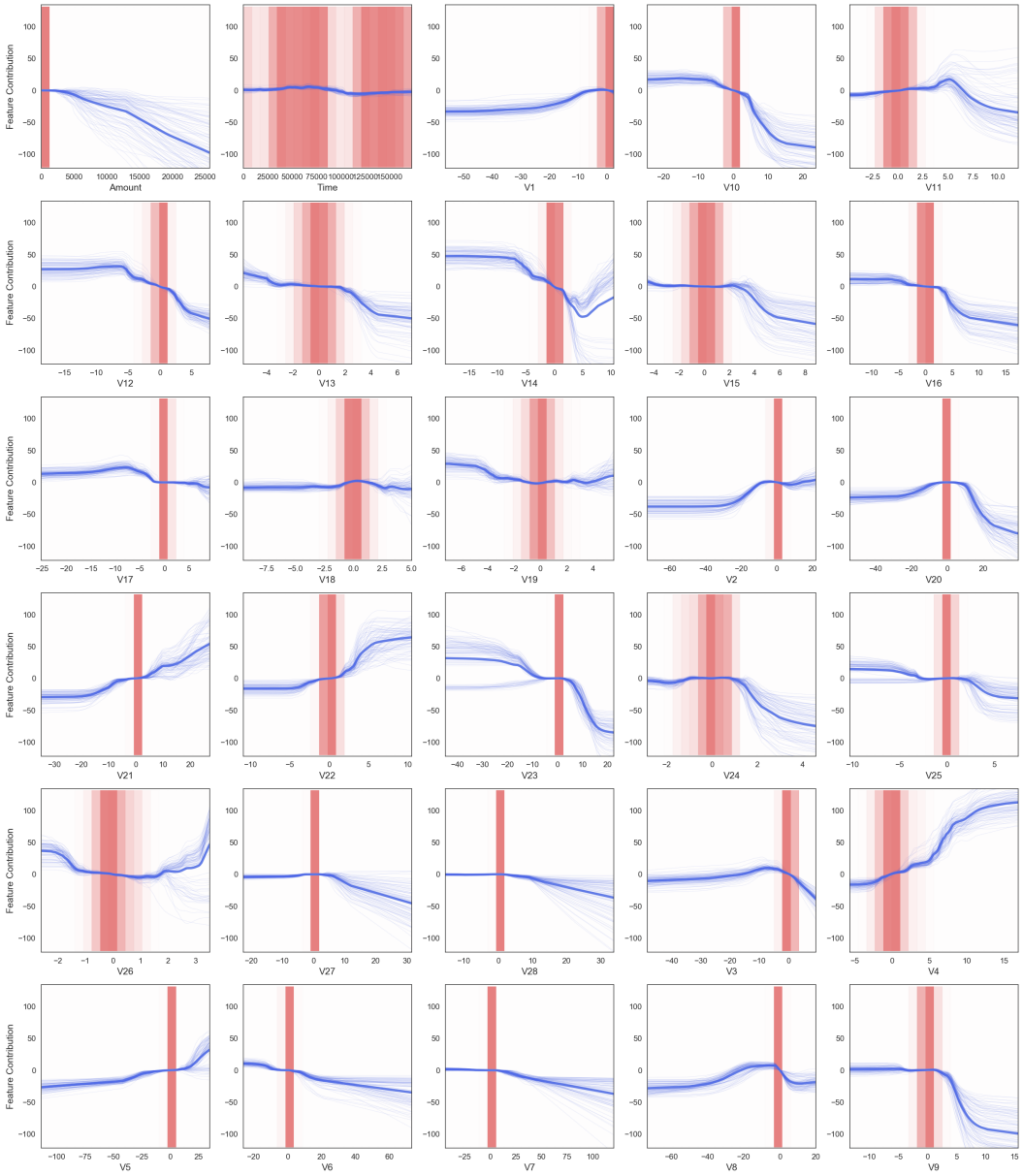


Figure 12: **Credit Fraud Detection**: Graphs learned by NAMs with ExU units on this large classification dataset. The task is to predict credit fraud where the class variable takes value 1 in case of fraud and 0 otherwise using a large dataset of credit card transactions. The dataset only contains only numerical input variables which are the result of a PCA transformation except the features 'Time' and 'Amount'. Unfortunately, due to confidentiality issues, the original features are not provided in the dataset.