Beyond Sparsity: Tree-based Regularization of Deep Models for Interpretability

Anonymous

Abstract

The lack of interpretability remains a key barrier to the adoption of deep models in many applications. In this work, we explicitly regularize deep models so human users might step through the process behind their predictions in little time. Specifically, we train deep timeseries models so their class-probability predictions have high accuracy while being closely modeled by decision trees with few nodes. On several real and synthetic examples, we demonstrate that this new tree-based regularization is distinct from simpler L2 or L1 penalties, resulting in more human-interpretable models without sacrificing predictive power.

Introduction

Deep models have become the de-facto approach for prediction in a variety of applications such as image classification (e.g. (Krizhevsky, Sutskever, and Hinton 2012)) and machine translation (e.g. (Bahdanau, Cho, and Bengio 2014; Sutskever, Vinyals, and Le 2014)). However, many practitioners are reluctant to adopt deep models because their predictions are difficult to interpret. In this work, we seek a specific form of interpretability known as *human-simulability*. A human-simulatable model is one in which a human user can "take in input data together with the parameters of the model and in reasonable time step through every calculation required to produce a prediction" (Lipton 2016). For example, small decision trees with only a few tens of nodes are easier for humans to simulate and thus understand and trust. In contrast, even simple deep models like multi-layer perceptrons with a hundred units can have far too many parameters and connections for a human to easily step through. Deep models for sequences are even more challenging. But can we create deep models that are well-approximated by human-simulatable models?

The question of creating accurate yet human-simulatable models is an important one, because in many domains simulatability is paramount. For example, despite advances in deep learning for clinical decision support (e.g. (Miotto et al. 2016; Choi et al. 2016; Che et al. 2015)), the clinical community remains skeptical of machine learning systems (Chen and Asch 2017). Simulatability allows clinicians to audit predictions easily. They can manually inspect changes to outputs under slightly-perturbed inputs, check substeps against their expert knowledge, and identify when predictions are made due to systemic bias in the data rather than real causes. Similar needs for simulatability exist in many decision-critical domains such as disaster response or recidivism prediction.

To address this need for interpretability, a number of works have been developed to assist in the interpretation of already-trained models. Craven and Shavlik (1996) train decision trees that mimic the predictions of a fixed, pretrained neural network, but do not train the network itself to be simpler. Other post-hoc interpretations typically typically evaluate the sensitivity of predictions to local perturbations of inputs or the input gradient (Ribeiro, Singh, and Guestrin 2016; Selvaraju et al. 2016; Adler et al. 2016; Lundberg and Lee 2016; Erhan et al. 2009). In parallel, research efforts have emphasized that simple lists of (perhaps locally) important features are not sufficient: Singh, Ribeiro, and Guestrin (2016) provide explanations in the form of programs; Lakkaraju, Bach, and Leskovec (2016) learn decision sets and show benefits over other rule-based methods.

These techniques focus on understanding already learned models, rather than finding models that are more interpretable. However, it is well-known that these deep models often have multiple optima of similar predictive accuracy (Goodfellow, Bengio, and Courville 2016), and thus one might hope to find models that are more interpretable with equal predictive accuracy. However, the field of *optimizing* deep models for interpretability remains nascent. Ross, Hughes, and Doshi-Velez (2017) penalize input sensitivity to features marked as less relevant. Lei, Barzilay, and Jaakkola (2016) train deep models that make predictions from text and simultaneously highlight contiguous subsets of words, called a "rationale," to justify each prediction. While both works optimize their deep models to expose relevant features, lists of features are not sufficient to *simulate* the prediction.

Contributions. In this work, we take steps toward *optimizing* deep models for human-simulatability via a new model complexity penalty function we call *tree regularization*. Tree regularization favors models whose decision boundaries can be well-approximated by small decision-trees, thus penalizing models that would require many calculations to simulate

predictions. We first demonstrate how this technique can be used to train simple multi-layer perceptrons to have tree-like decision boundaries. We then focus on timeseries applications and show that gated recurrent unit (GRU) models trained with strong tree-regularization reach a high-accuracy-at-low-complexity sweet spot that is not possible with any strength of L1 or L2 regularization. Prediction quality can be further boosted by training new hybrid models – GRU-HMMs – which explain the residuals of interpretable discrete HMMs via tree-regularized GRUs. We further show that the approximate decision trees for our tree-regularized deep models are useful for human simulation and interpretability. We demonstrate our approach on a speech recognition task and two medical treatment prediction tasks for patients with sepsis in the intensive care unit (ICU) and for patients with human immunodeficiency virus (HIV). Throughout, we also show that standalone decision trees as a baseline are noticeably less accurate than our tree-regularized deep models.

Related work. While there is little work (as mentioned above) on optimizing models for interpretability, there are some related threads. The first is *model compression*, which trains smaller models that perform similarly to large, black-box models (e.g. (Bucilu, Caruana, and Niculescu-Mizil 2006; Hinton, Vinyals, and Dean 2015; Balan et al. 2015; Han et al. 2015)). Other efforts specifically train very sparse networks via L1 penalties (Zhang, Lee, and Jordan 2016) or even *binary* neural networks (Tang, Hua, and Wang 2017; Rastegari et al. 2016) with the goal of faster computation. Edge and node regularization is commonly used to improve prediction accuracy (Drucker and Le Cun 1992; Ochiai et al. 2017), and recently Hu et al. (2016) improve prediction accuracy by training neural networks so that predictions match a small list of known domain-specific first-order logic rules. Sometimes, these regularizations—which all smooth or simplify decision boundaries—can have the effect of also improving interpretability. However, there is no guarantee that these regularizations will improve interpretability; we emphasize that specifically *training* deep models to have easily-simulatable decision boundaries is (to our knowledge) novel.

Background and Notation

We consider supervised learning tasks given datasets of N labeled examples, where each example (indexed by n) has an input feature vectors x_n and a target output vector y_n. We shall assume the targets y_n are binary, though it is simple to extend to other types. When modeling time-series, each example sequence n contains T_n timesteps indexed by t which each have a feature vector x_{nt} and an output y_{nt}. Formally, we write: $x_n = [x_{n1} \dots x_{nT_n}]$ and $y_n = [y_{n1} \dots y_{nT_n}]$. Each value y_{nt} could be prediction about the next timestep (e.g. the character at time $t+1$) or some other task-related annotation (e.g. if the patient became septic at time t).

Simple neural networks. A multi-layer perceptron (MLP) makes predictions $\hat{y}_n$ of the target y_n via a function $\hat{y}_n(x_n, W)$, where the vector W represents all parameters of the network. Given a data set $\{(x_n, y_n)\}$, our goal is to learn the parameters W to minimize the objective

$$\min_W \lambda \Psi(W) + \sum_{n=1}^{N} \text{loss}(y_n, \hat{y}_n(x_n, W)) \tag{1}$$

For binary targets y_n, the logistic loss (binary cross entropy) is an effective choice. The regularization term $\Psi(W)$ can represent L1 or L2 penalties (e.g. (Drucker and Le Cun 1992; Goodfellow, Bengio, and Courville 2016; Ochiai et al. 2017)) or our new regularization.

Recurrent Neural Networks with Gated Recurrent Units. A recurrent neural network (RNN) takes as input an arbitrary length sequence $x_n = [x_{n1} \dots x_{nT_n}]$ and produces a "hidden state" sequence $h_n = [h_{n1} \dots h_{nT_n}]$ of the same length as the input. Each hidden state vector at timestep t represents a location in a (possibly low-dimensional) "state space" with K dimensions: $h_{nt} \in \mathbb{R}^K$. RNNs perform sequential *nonlinear* embedding of the form $h_{nt} = f(x_{nt}, h_{nt-1})$ in hope that the state space location h_{nt} is a useful summary statistic for making predictions of the target y_{nt} at timestep t.

Many different variants of the transition function architecture f have been proposed to solve the challenge of capturing long-term dependencies. In this paper, we use gated recurrent units (GRUs) (Cho et al. 2014), which are simpler than other alternatives such as long short-term memory units (LSTMs) (Hochreiter and Schmidhuber 1997). While GRUs are convenient, any differentiable RNN architecture is compatible with our new tree-regularization approach.

Below we describe the evolution of a single GRU sequence, dropping the sequence index n for readability. The GRU transition function f produces the state vector $h_t = [h_{t1} \dots h_{tK}]$ from a previous state h_{t-1} and an input vector x_t, via the following feed-forward architecture:

$$\begin{aligned}
\text{output state} &: h_{tk} = (1 - z_{tk}) h_{t-1,k} + z_{t,k} \tilde{h}_{tk} \\
\text{candidate state} &: \tilde{h}_{tk} = \tanh(V_k^h x_t + U_k^h (r_t \odot h_{t-1})) \\
\text{update gate} &: z_{tk} = \sigma(V_k^z x_t + U_k^z h_{t-1}) \\
\text{reset gate} &: r_{tk} = \sigma(V_k^r x_t + U_k^r h_{t-1})
\end{aligned} \tag{2}$$

The internal network nodes include candidate state gates $\tilde{h}$, update gates z and reset gates r which have the same cardinalty as the state vector h. Reset gates allow the network to forget past state vectors when set near zero via the logistic sigmoid nonlinearity $\sigma(\cdot)$. Update gates allow the network to either pass along the previous state vector unchanged or use the new candidate state vector instead. This architecture is diagrammed in Figure 1.

The predicted probability of the binary target y_t for timestep t is a sigmoid transformation of the state at time t:

$$\hat{y}_t = \sigma(w^T h_t) \tag{3}$$

Here, weight vector $w \in \mathbb{R}^K$ represents the parameters of this output layer. We denote the parameters for the entire GRU-RNN model as $W = (w, U, V)$, concatenating all component parameters. We can train GRU-RNN timeseries models (hereafter often just called GRUs) via the following loss

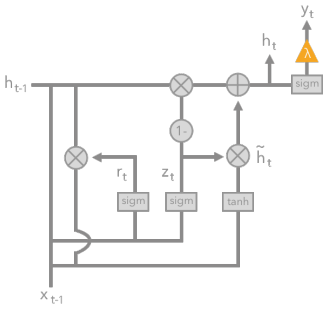


Figure 1: The orange triangle indicates the output used in surrogate training for tree regularization of the GRU.

minimization objective:

$$\min_{W} \lambda\Psi(W) + \sum_{n=1}^{N}\sum_{n=1}^{T_n} \text{loss}(y_{nt}, \hat{y}_{nt}(x_n, W)) \qquad (4)$$

where again $\Psi(W)$ defines a regularization cost.

Decision-Tree Regularized Models

We now propose a novel tree-based regularization function $\Omega(W)$ for the parameters of a differentiable model which attempts to penalize models whose predictions are not easily *simulatable*. Of course, it is difficult to measure "simulatability" directly for an arbitrary network, so we take inspiration from decision trees. Our chosen method has two stages: first, find a single binary decision tree which accurately reproduces the network's thresholded binary predictions $\hat{y}_n$ given input x_n. Second, measure the complexity of this decision tree as the output of $\Omega(W)$. We measure complexity as the *average decision path length*—the average number of decision nodes that must be touched to make a prediction for an input example x_n. We compute the *average* with respect to some designated reference dataset of example inputs $D = \{x_n\}$ from the training set. While many ways to measure complexity exist, we find average path length is most relevant to our notion of *simulatability*. Remember that for us, human simulation requires stepping through every calculation required to make a prediction. Average path length exactly counts the number of true-or-false boolean calculations needed to make an average prediction, assuming the model is a decision tree. Total number of nodes could be used as a metric, but might penalize more accurate trees that have short paths for most examples but need more involved logic for few outliers.

Our true-average-path-length cost function $\Omega(W)$ is detailed in Alg. 1. It requires two subroutines, TRAINTREE and PATHLENGTH. TRAINTREE trains a binary decision tree to accurately reproduce the provided labeled examples $\{x_n, \hat{y}_n\}$. We use the `DecisionTree` module distributed in Python's scikit-learn (Pedregosa et al. 2011) with post-pruning to simplify the tree. These trees can give probabilistic predictions at each leaf. (Complete decision-tree training details are in the supplement.) Next, PATHLENGTH counts how many nodes are needed to make a specific input to an output node in the provided decision tree. In our evaluations, we will apply our average-decision-tree-path-length regularization, or simply "tree regularization," to several neural models.

Algorithm 1 Average-Path-Length Cost Function

Require:
$\hat{y}(\cdot, W)$: binary prediction function, with parameters W
$D = \{x_n\}_{n=1}^{N}$: reference dataset with N examples
function $\Omega(W)$
  tree $\leftarrow$ TRAINTREE($\{x_n, \hat{y}(x_n, W)\}$)
  return $\frac{1}{N}\sum_n$ PATHLENGTH(tree, x_n)

Alg. 1 defines our average-path-length cost function $\Omega(W)$, which can be plugged into the abstract regularization term $\Psi(W)$ in the objectives in equations 1 and 4.

Making the Decision-Tree Loss Differentiable Training decision trees is not differentiable, and thus $\Omega(W)$ as defined in Alg. 1 is not differentiable with respect to the network parameters W (unlike standard regularizers such as the L1 or L2 norm). While one could resort to derivative-free optimization techniques (Audet and Kokkolaras 2016), gradient descent has been an extremely fast and robust way of training networks (Goodfellow, Bengio, and Courville 2016).

A key technical contribution of our work is introducing and training a *surrogate* regularization function $\hat{\Omega}(W)$: supp$(W) \rightarrow \mathbb{R}_+$ to map each candidate neural model parameter vector W to an *estimate* of the average-path-length. Our approximate function $\hat{\Omega}$ is implemented as a standalone multi-layer perceptron network and is thus *differentiable*. Let vector ξ of size k denote the parameters of this chosen MLP approximator. We can train $\hat{\Omega}$ to be a good estimator by minimizing a squared error loss function:

$$\min_{\xi} \textstyle\sum_{j=1}^{J} (\Omega(W_j) - \hat{\Omega}(W_j, \xi))^2 + \epsilon||\xi||_2^2 \qquad (5)$$

where W_j are the *entire* set of parameters for our model, $\epsilon > 0$ is a regularization strength, and we assume we have a dataset of J known parameter vectors and their associated true path-lengths: $\{W_j, \Omega(W_j)\}_{j=1}^{J}$. This dataset can be assembled using the candidate W vectors obtained while training our target neural model $\hat{y}(\cdot, W)$, as well as by evaluating $\Omega(W)$ for randomly generated W. Importantly, one can train the surrogate function $\hat{\Omega}$ in parallel with our network. In the supplement, we show evidence that our surrogate predictor $\hat{\Omega}(\cdot)$ tracks the true average path length as we train the target predictor $\hat{y}(\cdot, W)$.

Training the Surrogate Loss Even moderately-sized GRUs can have parameter vectors W with thousands of dimensions. Our labeled dataset for surrogate training – $\{W_j, \Omega(W_j)\}_{j=1}^{J}$—will only have one W_j example from each target network training iteration. Thus, in early iterations, we will have only few examples from which to learn a good surrogate function $\hat{\Omega}(W)$. We resolve this challenge via *augmenting* our training set with additional examples: We randomly sample weight vectors W and calculate the true

average path length $\Omega(W)$, and we also perform several random restarts on the unregularized GRU and use those weights in our training set.

A second challenge occurs later in training: as the model parameters W shift away from their initial values, those early parameters may not be as relevant in characterizing the current decision function of the GRU. To address this, for each epoch, we use examples only from the past E epochs (in addition to augmentation), where in practice, E is empirically chosen. Using examples from a fixed window of epochs also speeds up training. The supplement shows a comparison of the importance of these heuristics for efficient and accurate training—empirically, data augmentation for stabilizing surrogate training allows us to scale to GRUs with 100s of nodes. GRUs of this size are sufficient for many real problems, such as those we encounter in healthcare domains.

Typically, we use $J = 50$ labeled pairs for surrogate training for toy datasets and $J = 100$ for real world datasets. Optimization of our surrogate objective is done via gradient descent. We use Autograd to compute gradients of the loss in Eq. (5) with respect to ξ, then use Adam to compute descent directions with step sizes set to 0.01 for toy datasets and 0.001 for real world datasets.

Tree-Regularized MLPs: A Demonstration

While timeseries models are the main focus of this work, we first demonstrate tree regularization on a simple binary classification task to build intuition. We call this task the 2D Parabola problem, because as Fig. 2(a) shows, the training data consists of 2D input points whose two-class decision boundary is roughly shaped like a parabola. The true decision function is defined by $y = 5 * (x - 0.5)^2 + 0.4$. We sampled 500 input points x_n uniformly within the unit square $[0, 1] \times [0, 1]$ and labeled those above the decision function as positive. To make it easy for models to overfit, we flipped 10% of the points in a region near the boundary. A random 30% were held out for testing.

For the classifier $\hat{y}$, we train a 3-layer MLP with 100 first layer nodes, 100 second layer nodes, and 10 third layer nodes. This MLP is intentionally overly expressive to encourage overfitting and expose the impact of different forms of regularization: our proposed tree regularization $\Psi(W) = \hat{\Omega}(W)$ and two baselines: an L2 penalty on the weights $\Psi(W) = ||W||_2$, and an L1 penalty on the weights $\Psi(W) = ||W||_1$. For each regularization function, we train models at many different regularization strengths λ chosen to explore the full range of decision boundary complexities possible under each technique.

For our tree-based regularization, we model our surrogate $\hat{\Omega}(W)$ with a 1-hidden layer MLP with 25 units. We find this simple architecture works well, but certainly more complex MLPs could could be used on more complex problems. The objective in equation 1 was optimized via Adam gradient descent (Kingma and Ba 2014) using a batch size of 100 and a learning rate of 1e-3 for 250 epochs, and hyperparameters were set via cross validation using grid search (see supplement for full experimental details).

Fig. 2 (b) shows the each trained model as a single point in a 2D fitness space: the x-axis measures model complexity via our average-path-length metric, and the y-axis measures AUC prediction performance. These results show that simple L1 or L2 regularization does *not* produce models with both small node count and good predictions at *any* value of the regularization strength λ. As expected, large λ values for L1 and L2 only produce far-too-simple linear decision boundaries with poor accuracies. In contrast, our proposed tree regularization directly optimizes the MLP to have simple tree-like boundaries at high λ values which can still yield good predictions.

The lower panes of Fig. 2 shows these boundaries. Our tree regularization is uniquely able to create axis-aligned functions, because decision trees prefer functions that are axis-aligned splits. These axis-aligned functions require very few nodes but are more effective than L1 and L2 counterparts. The L1 boundary is more sharp, whereas the L2 is more round.

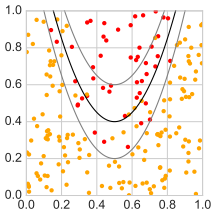
Tree-Regularized Timeseries Models

We now evaluate our tree-regularization approach on timeseries models. We shall first focus on GRUs and then hybrid GRU-HMM models. As with the MLP, each regularization technique (tree, L2, L1) can be applied to the output node of the GRU across a range of strength parameters λ. We also show results for a baseline standalone decision tree classifier. Importantly, Algorithm 1 can compute the average-decision-tree-path-length for any fixed deep model given its parameters, and can hence be used to measure decision boundary complexity under any regularization, including L1 or L2. This means that when training any model, we can track both the predictive performance (as measured by area-under-the-ROC-curve (AUC); higher values mean better predictions), as well as the complexity of the decision tree required to explain each model (as measured by our average path length metric; lower values mean more interpretable models). Further details of our experimental protocol are in the supplement, as well as more extensive results with additional baselines.

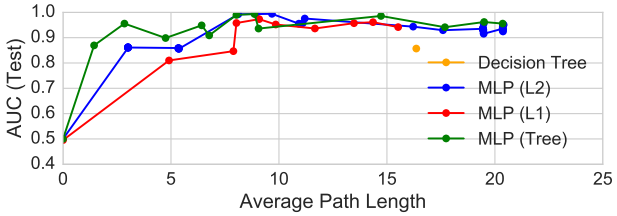
Tasks

Synthetic Task: Signal-and-noise HMM We generated a toy dataset of $N = 100$ sequences, each with $T = 50$ timesteps. Each timestep has a data vector x_{nt} of 14 binary features and a single binary output label y_{nt}. The data comes from two separate HMM processes. First, a "signal" HMM generates the first 7 data dimensions from 5 well-separated states. Second, an independent "noise" HMM generates the remaining 7 data dimensions from a different set of 5 states. Each timestep's output label y_{nt} is produced by a rule involving *both* the signal data and the signal hidden state: the target is 1 at timestep t only if both the first signal state is active and the first observation is turned on. We deliberately designed the generation process so that neither logistic regression with x as features nor an RNN model that makes predictions from hidden states alone can perfectly separate this data.

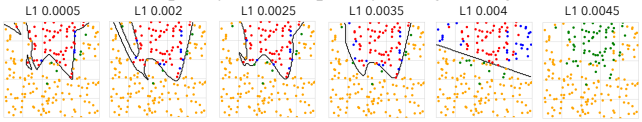
Real-World Tasks: We also tested our approach on several real-world tasks: predicting medical outcomes of hospital-



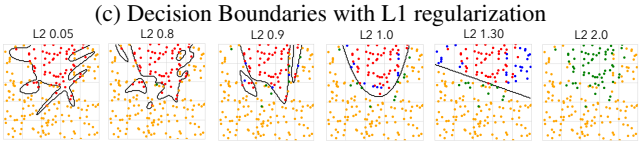
(a) Training Data for 2D Parabola



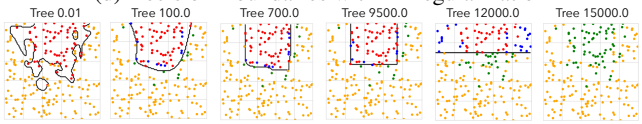
(b) Prediction quality and complexity as reg. strength λ varies



(c) Decision Boundaries with L1 regularization



(d) Decision Boundaries with L2 regularization



(e) Decision Boundaries Tree regularization

Figure 2: *(b)*: In the small average path length regime (0-5), tree-regularization produces models with higher AUC than L1 or L2. *(c-e)*: Decision boundaries (black lines) have qualitatively different shapes for different regularization schemes, as regularization strength λ increases. We color each prediction as true positive (red), true negative (yellow), false negative (green), and false positive (blue).

ized septic patients, HIV therapy outcome prediction, and predicting stop phoneme groups from a selection of English speech recordings. To normalize scales, we independently standardized input features x via z-scoring.

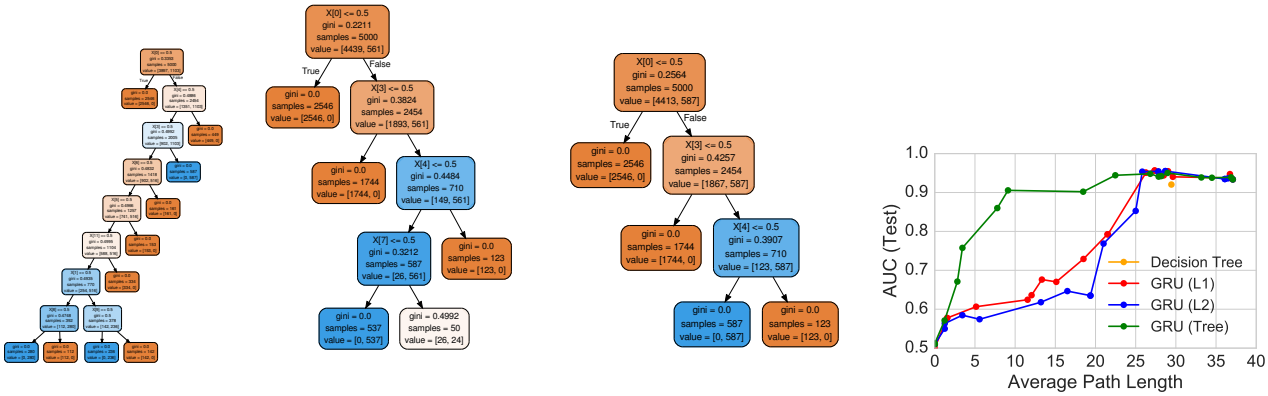
- Sepsis Critical Care: We study timeseries data for 11 786 septic ICU patients from the public MIMIC III dataset (Johnson et al. 2016). We observe at each hour (timestep) t a data vector x_{nt} of 35 vital signs and lab results as well as a label vector y_{nt} of 5 binary outcomes. Hourly data x_{nt} measures continuous input features such as respiration rate (RR), blood oxygen levels (paO_2), fluid levels, and more. Hourly binary labels y_{nt} include whether the patient died in hospital, whether the patient died after 90 days, and if mechanical ventilation was applied. Models are trained to predict all 5 output dimensions concurrently from one shared embedding. The average sequence length is 15 hours. 7 070 patients are used in training, 1 769 for validation, and 294 for test.
- HIV Therapy Outcome (HIV): We make use of the EuResist Integrated Database (Zazzi et al. 2012) for 53 236 patients diagnosed with HIV. We consider 4-6 month intervals (corresponding to hospital visits) as time steps. Each data vector x_{nt} has 40 features, including blood counts, viral load measurements and lab results. Each output vector y_{nt} has 15 binary labels, including whether a therapy was successful in reducing viral load to below detection limits, if therapy caused CD4 blood cell counts to drop to dangerous levels (indicating AIDS), or if the patient suffered adherence issues to medication. The average sequence length is 14 steps. 37 618 patients are used for training; 7 986 for testing, and 7 632 for validation.
- Phonetic Speech (TIMIT): Timeseries data containing broadband recordings of 630 speakers of eight major dialects of American English reading ten phonetically rich sentences (Garofolo et al. 1993). Each sentence contains time-aligned phonetic transcriptions of 60 phonemes. We focus on the problem of distinguishing stop phonemes (those that stop the flow of air, such as "b", "d", or "g") from non-stops. Each timestep has one binary output y_{nt} indicating whether a stop phoneme occurs or not. There are 26 continuous features for each input vector x_{nt} representing the Mel-frequency cepstral coefficients and derivatives of the acoustic signal. There are 6 303 sequences: which we split into 3 697 for training, 925 for validation, and 1 681 for testing. The average length is 614.

Results

The major conclusions of our experiments comparing GRUs with various regularizations are outlined below.

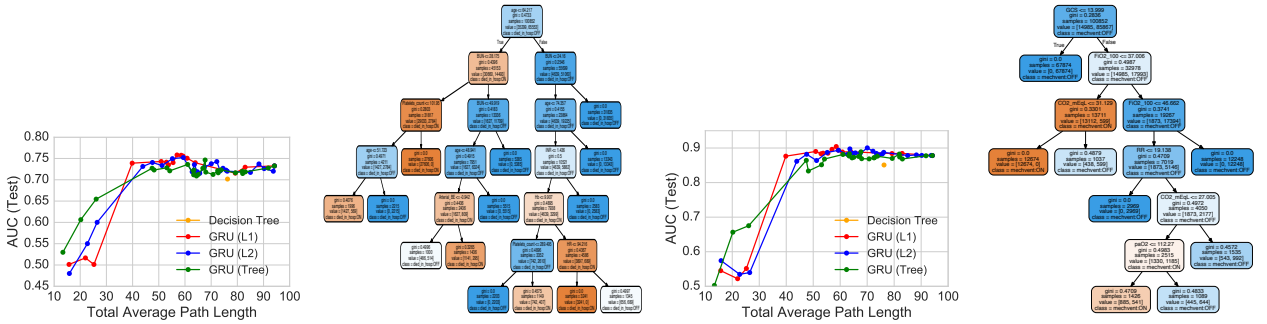
Tree-regularized models have fewer nodes than other forms of regularization. Across tasks, we see that in the target regime of small decision trees (low average-path lengths), our proposed regularization achieves higher prediction quality (higher AUCs). In the signal-and-noise HMM task, tree regularization (green line in Fig. 3(d)) achieves AUC values near 0.9 when its trees have an average path length of 10. Similar models with L1 or L2 regularization reach this AUC only with trees that are nearly double in complexity (path length over 25). On both the SEPSIS (Fig. 4) and TIMIT (Fig. 5a), we see considerable gains in accuracy—AUC differences of 0.05 to 0.15—for path lengths of 20-30. Similarly, on the HIV tasks in Fig. 5b- 5d, we see AUC differences of between 0.04 and 0.11 for path lengths of 45-55.

In domains where human-simulatability is required, these increases in accuracy in the small-complexity regime can mean the difference between models that provide value on a task and models that are unusable, either because their performance is too poor or they are uninterpretable. We emphasize that across all tasks, standalone decision trees (marked by yellow dots in line plots) cannot reach this high-accuracy, low -complexity sweet spot.



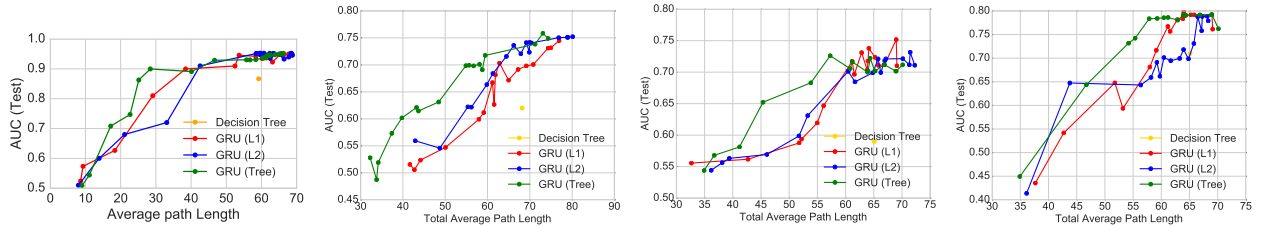
(a) GRU $\lambda = 1$ (b) GRU $\lambda = 800$ (c) GRU $\lambda = 1\,000$ (d) GRU

Figure 3: *Toy Signal-and-Noise HMM Task: (a)-(c)* Decision trees trained to mimic predictions of GRU models with 25 hidden states at different regularization strengths λ; as expected, increasing λ decreases the size of the learned trees (see supplement for trees at many more λ values). Decision tree (c) suggests the model learns to predict positive output (blue) if and only if "x[0] == 1 and x[3] == 1 and x[4] == 0". This simple description is consistent with the true rule used to generate labels our dataset: assign positive label only if first dimension is on (x[0] == 1) and first state is active (the emission probability vector for this state is: [.5 .5 .5 .5 0 . . .]). *(d)* Tree-regularization produces simpler models (as measured by average path length) with higher prediction quality (AUC) across range of regularization strengths λ for the GRU.



(a) In-Hospital Mortality (b) In-Hospital Mortality (c) Mechanical Ventilation (d) Mechanical Ventilation

Figure 4: *SEPSIS task:* Study of different regularization techniques for GRU model with 100 states, trained to jointly predict 5 binary outcomes. Panels (a) and (c) show AUC vs. average path length for 2 of the 5 outcomes (remainder in the appendix); in both cases, tree-regularization provides higher accuracy in the target regime of low-complexity decision trees. Panels (b) and (d) show the associated decision trees for $\lambda = 2\,000$; these were found by clinically interpretable by an ICU clinician (see main text).



(a) TIMIT Stop Phonemes (b) HIV Therapy Success (c) HIV: CD4$^+ \leq 200$ cells/ml (d) HIV Adherence

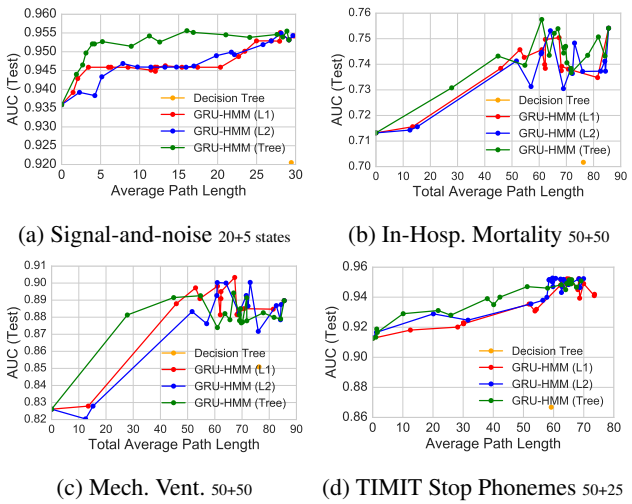
Figure 5: *TIMIT and HIV tasks:* Study of different regularization techniques for GRU model with 75 states. Panels (a)-(d) are tradeoff curves showing how AUC predictive power and decision-tree complexity evolve with increasing regularization strength under L1, L2 or tree regularization in both TIMIT and HIV settings. The GRU is trained to jointly predict 15 binary outcomes for HIV, of which 3 are shown here in Panels (b) - (d).

Our learned decision-tree-like boundaries are interpretable. Across all tasks, the trees which mimic the predictions of tree-regularized deep models are small enough to simulate by hand and help users grasp the model's nonlinear prediction logic. In Fig. 3(a)-(c), the trees for our synthetic task decrease in size as the strength λ increases. The logic of these trees also matches the true labeling process: even the simplest tree (c) checks a relevant subset of input dimensions necessary to verify that both the first state and the first output dimension are active. In Fig. 4, we show decision trees for two sepsis prediction tasks. We consulted a clinical expert on sepsis treatment, who noted that the trees helped him understand what the models might be doing and thus determine if he would trust the deep model. For example, he said that using FiO_2, RR, CO_2 and paO_2 to predict need for mechanical ventilation (Fig. 4d) was sensible, as these all measure breathing quality. In contrast, the in-hospital mortality tree (Fig. 4b) predicts that some young patients with no organ failure have high mortality rates while other young patients with organ failure have low mortality. These counter-intuitive results led to hypotheses about how uncaptured variables impact the training process. Such reasoning would not be possible from simple sensitivity analyses of the deep model.

Practical runtimes for tree regularization are less than twice that of simpler L2. While our tree-regularized GRU with 10 states takes 3977 seconds per epoch on TIMIT, an equivalent L2-regularized GRU takes 2116 seconds per epoch. Thus, our new method has cost less than twice the baseline *even when the path-length surrogate is serially computed*. Because the surrogate $\hat{\Omega}(W)$ will in general be a much smaller model than the predictor $\hat{y}(x, W)$, we expect one could get much closer per-epoch times by parallelizing the creation of $(W, \Omega(W))$ training pairs and the training of the surrogate $\hat{\Omega}(W)$. Additionally, 3977 seconds includes the time needed to train the surrogate. In practice, we do this sparingly, only once every 25 epochs, yielding an amortized per-epoch cost of 2191 seconds (more runtime results are in the supplement).

Decision trees are stable over multiple optimization runs. When tree regularization is strong (high λ), the decision trees trained to match the predictions of deep models are stable. For both signal-and-noise and sepsis tasks, multiple runs from different random restarts have nearly identical tree shape and size, perhaps differing by a few nodes. This stability is crucial to building trust in our method. On the signal-and-noise task ($\lambda = 7000$), 7 of 10 independent runs with random initializations resulted in trees of exactly the same structure, and the others closely resembled those sharing the same subtrees and features (more details in supplement).

The deep residual GRU-HMM can achieve high AUC with less complexity. So far, we have focused on regularizing standard deep models, such as MLPs or GRUs. Another option is to use a deep model as a residual on another model that is already interpretable: for example, discrete HMMs partition timesteps into clusters, each of which can be inspected, but its predictions might have limited accuracy. In Fig. 6, we show the performance of jointly training a *GRU-HMM*, a new model which combines an HMM with a tree-regularized GRU to improve its predictions (details and further results in the supplement). Here, the ideal path length is zero, indicating only the HMM makes predictions. For small average-path-lengths, the GRU-HMM substantially improves the original HMM's predictions *and* has simulatability gains over earlier GRUs. On the mechanical ventilation task, the GRU-HMM requires an average path length of only 28 to reach AUC of 0.88, while the GRU alone with the same number of states requires a path length of 60 to reach the same AUC. This suggests that jointly-trained deep residual models may provide even better interpretability.



(a) Signal-and-noise 20+5 states (b) In-Hosp. Mortality 50+50

(c) Mech. Vent. 50+50 (d) TIMIT Stop Phonemes 50+25

Figure 6: Tradeoff curves for the GRU-HMM. Captions show the number of HMM states plus the number of GRU states.

Discussion and Conclusion

We have introduced a novel tree-regularization technique that encourages the complex decision boundaries of any differentiable model to be well-approximated by human-simulatable functions, allowing domain experts to quickly understand and approximately *compute* what the more complex model is doing. Overall, our training procedure is robust and efficient; future work could continue to explore and increase the stability of the learned models as well as identify ways to apply our approach to situations in which the inputs are not inherently interpretable (e.g. pixels in an image).

That said, across three complex, real-world domains, our tree-regularized models provide better accuracies for simpler, approximately human-simulatable models. Future work could also apply tree regularization to local, example-specific approximations of a loss (Ribeiro, Singh, and Guestrin 2016) or to representation learning tasks (encouraging embeddings with simple boundaries). More broadly, our general training procedure could apply tree-regularization or other procedure-regularization to a wide class of popular models, helping us move beyond sparsity toward models humans can easily simulate and thus trust.

References

Adler, P.; Falk, C.; Friedler, S. A.; Rybeck, G.; Scheidegger, C.; Smith, B.; and Venkatasubramanian, S. 2016. Auditing black-box models for indirect influence. In *ICDM*.

Audet, C., and Kokkolaras, M. 2016. *Blackbox and derivative-free optimization: theory, algorithms and applications*. Springer.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Balan, A. K.; Rathod, V.; Murphy, K. P.; and Welling, M. 2015. Bayesian dark knowledge. In *NIPS*.

Bucilu, C.; Caruana, R.; and Niculescu-Mizil, A. 2006. Model compression. In *KDD*.

Che, Z.; Kale, D.; Li, W.; Bahadori, M. T.; and Liu, Y. 2015. Deep computational phenotyping. In *KDD*.

Chen, J. H., and Asch, S. M. 2017. Machine learning and prediction in medicinebeyond the peak of inflated expectations. *N Engl J Med* 376(26):2507–2509.

Cho, K.; Gulcehre, B. v. M. C.; Bahdanau, D.; Schwenk, F. B. H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMLNP*.

Choi, E.; Bahadori, M. T.; Schuetz, A.; Stewart, W. F.; and Sun, J. 2016. Doctor AI: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*.

Craven, M., and Shavlik, J. W. 1996. Extracting tree-structured representations of trained networks. In *NIPS*.

Drucker, H., and Le Cun, Y. 1992. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks* 3(6):991–997.

Erhan, D.; Bengio, Y.; Courville, A.; and Vincent, P. 2009. Visualizing higher-layer features of a deep network. *University of Montreal* 1341:3.

Garofolo, J. S.; Lamel, L. F.; Fisher, W. M.; Fiscus, J. G.; Pallett, D. S.; Dahlgren, N. L.; and Zue, V. 1993. Timit acoustic-phonetic continuous speech corpus. *Linguistic data consortium* 10(5).

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *NIPS*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hu, Z.; Ma, X.; Liu, Z.; Hovy, E.; and Xing, E. 2016. Harnessing deep neural networks with logic rules. In *ACL*.

Johnson, A. E.; Pollard, T. J.; Shen, L.; Lehman, L. H.; Feng, M.; Ghassemi, M.; Moody, B.; Szolovits, P.; Celi, L. A.; and Mark, R. G. 2016. MIMIC-III, a freely accessible critical care database. *Scientific Data* 3.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks. In *NIPS*.

Lakkaraju, H.; Bach, S. H.; and Leskovec, J. 2016. Interpretable decision sets: A joint framework for description and prediction. In *KDD*.

Lei, T.; Barzilay, R.; and Jaakkola, T. 2016. Rationalizing neural predictions. *arXiv preprint arXiv:1606.04155*.

Lipton, Z. C. 2016. The mythos of model interpretability. In *ICML Workshop on Human Interpretability in Machine Learning*.

Lundberg, S., and Lee, S.-I. 2016. An unexpected unity among methods for interpreting model predictions. *arXiv preprint arXiv:1611.07478*.

Miotto, R.; Li, L.; Kidd, B. A.; and Dudley, J. T. 2016. Deep patient: An unsupervised representation to predict the future of patients from the electronic health records. *Scientific reports* 6:26094.

Ochiai, T.; Matsuda, S.; Watanabe, H.; and Katagiri, S. 2017. Automatic node selection for deep neural networks using group lasso regularization. In *ICASSP*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. Why should I trust you?: Explaining the predictions of any classifier. In *KDD*.

Ross, A.; Hughes, M. C.; and Doshi-Velez, F. 2017. Right for the right reasons: Training differentiable models by constraining their explanations. In *IJCAI*.

Selvaraju, R. R.; Das, A.; Vedantam, R.; Cogswell, M.; Parikh, D.; and Batra, D. 2016. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1610.02391*.

Singh, S.; Ribeiro, M. T.; and Guestrin, C. 2016. Programs as black-box explanations. *arXiv preprint arXiv:1611.07579*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Tang, W.; Hua, G.; and Wang, L. 2017. How to train a compact binary neural network with high accuracy? In *AAAI*.

Zazzi, M.; Incardona, F.; Rosen-Zvi, M.; Prosperi, M.; Lengauer, T.; Altmann, A.; Sonnerborg, A.; Lavee, T.; Schülter, E.; and Kaiser, R. 2012. Predicting response to antiretroviral treatment by machine learning: the euresist project. *Intervirology* 55(2):123–127.

Zhang, Y.; Lee, J. D.; and Jordan, M. I. 2016. l1-regularized neural networks are improperly learnable in polynomial time. In *ICML*.