
Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections

Xin Zhang¹ Armando Solar-Lezama¹ Rishabh Singh²

Abstract

The paper describes a new algorithm to generate minimal, stable, and symbolic corrections to an input that will cause a neural network with ReLU neurons to change its output. We argue that such a correction is a useful way to provide feedback to a user when the neural network produces an output that is different from a desired output. Our algorithm generates such a correction by solving a series of linear constraint satisfaction problems. The technique is evaluated on a neural network that has been trained to predict whether an applicant will pay a mortgage.

1. Introduction

When machine learning is used to make decisions about people in the real world, it is extremely important to be able to explain the rationale behind those decisions. Unfortunately, for systems based on deep learning, it is often not even clear what an explanation means; showing someone the sequence of operations that computed a decision provides little actionable insight. There have been some recent advances towards making deep neural networks more interpretable (e.g. (Montavon et al., 2017)) using two main approaches: i) generating input prototypes that are representative of abstract concepts corresponding to different classes (Nguyen et al., 2016) and ii) explaining network decisions by computing relevance scores to different input features (Bach et al., 2015). However, these explanations do not provide direct actionable insights regarding how to cause the prediction to move from an undesirable class to a desirable class.

In this paper, we argue that for the specific class of *judgment problems*, minimal, stable, and symbolic corrections are an ideal way of explaining a neural network decision. We use the term judgment in this paper to refer to a particular kind of binary decision problem where a user presents some information to an algorithm that is supposed to pass judgment on its input. The distinguishing feature of judgments relative to other kinds of decision problems is that they are asymmetric;

if I apply for a loan and I get the loan, I am satisfied, and do not particularly care for an explanation; even the bank may not care as long as on aggregate the algorithm makes the bank money. On the other hand, I very much care if the algorithm denies my mortgage application. The same is true for a variety of problems, from college admissions, to parole, to hiring decisions. In each of these cases, the user expects a positive judgment, and would like an actionable explanation to accompany a negative judgment.

We argue that a *correction* is a useful form of feedback; what could I have done differently to elicit a positive judgment? For example, if I applied for a mortgage, knowing that I would have gotten a positive judgment if my debt to income ratio (DTI) was 10% lower is extremely useful; it is actionable information that I can use to adjust my finances. We argue, however, that the most useful corrections are those that are minimal, stable and symbolic.

First, in order for a correction to be actionable, the corrected input should be as similar as possible from the original offending input. For example, knowing that a lower DTI would have given me the loan is useful, but knowing that a 65 year old billionaire from Nebraska would have gotten the loan is not useful. Minimality must be defined in terms of an error model which specifies which inputs are subject to change and how. For a bank loan, for example, debt, income and loan amount are subject to change within certain bounds, but I will not move to another state just to satisfy the bank.

Second, the suggested correction should be stable, meaning that there should be a neighborhood of points surrounding the suggested correction for which the outcome is also positive. For example, if the algorithm tells me that a 10% lower DTI would have gotten me the mortgage, and then six months later I come back with a DTI that is 11% lower, I expect to get the mortgage, and will be extremely disappointed if the bank says, “oh, sorry, we said 10% lower, not 11% lower”. So even though for the neural network it may be perfectly reasonable to give positive judgments to isolated points surrounded by points that get negative judgments, corrections that lead to such isolated points will not be useful.

Finally, even if the correction is minimal and robust, it is

¹MIT CSAIL ²Google Brain. Correspondence to: Xin Zhang <xzhang@csail.mit.edu>.

even better if rather than a single point, the algorithm can produce a *symbolic* correction that provides some insight about the relationship between different variables. For example, knowing that for someone like me the bank expects a DTI of between 20% and 30% is more useful than just knowing a single value. And knowing something about how that range would change as a function of my credit score would be even more useful still.

In this paper, we present the first algorithm capable of computing minimal stable symbolic corrections. Given a neural network with ReLU activations, our algorithm produces a symbolic description of a space of corrections such that any correction in that space will change the judgment. In the limit, the algorithm will find the closest region with a volume above a given threshold. Internally, our algorithm reduces the problem into a series of linear constraint satisfaction problems, which are solved using the Gurobi linear programming solver (Gurobi Optimization, Inc., 2018). We show that in practice, the algorithm is able to find good symbolic corrections in 20 minutes on average for small but realistic networks. We evaluate our approach on a neural network trained on mortgage data that predicts whether a given applicant will default on a mortgage.

2. Background and Problem Definition

We first introduce some notations we will use in explaining our algorithm for computing minimal, robust, symbolic corrections given an input to a neural network F with ReLU activation. In the model we consider, the input to the network is a vector \mathbf{v}_0 of size s_0 . The network computes the output of each layer as

$$\mathbf{v}_{i+1} = f_i(\mathbf{v}_i) = \text{ReLU}(\mathbf{W}_i * \mathbf{v}_i + \mathbf{b}_i)$$

Where \mathbf{W}_i is an $s_i \times s_{i+1}$ matrix, and ReLU applies the rectifier function elementwise to the output of the linear operations.

We focus on classification problems, where the classification of input \mathbf{v} is obtained by

$$l_F(\mathbf{v}) \in \text{argmax}_i F(\mathbf{v})[i].$$

We are specifically focused on binary classification problems (that is, $l_F(\mathbf{v}) \in \{0, 1\}$). The *judgement problem* is a special binary classification problem where one label is preferable than the other. We assume 1 is preferable throughout the paper.

The *judgement interpretation problem* concerns providing feedback in the form of corrections when $l_F(\mathbf{v}) = 0$. A correction δ is a real vector of input vector length such that $l_F(\mathbf{v} + \delta) = 1$. As mentioned previously, a desirable feedback should be a minimal, stable, and symbolic correction. We first introduce what it means for a concrete correction

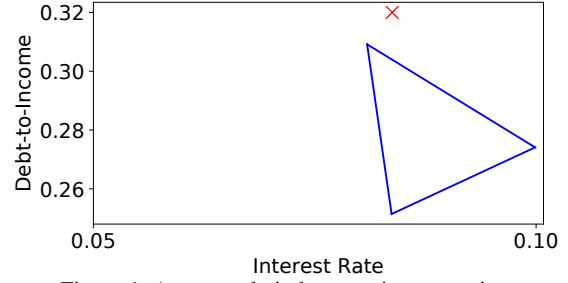


Figure 1. An example judgement interpretation.

δ to be minimal and stable. Minimality is defined in terms of a norm $\|\delta\|$ on δ that measures the distance between the corrected input and the original input. For simplicity, we use L_1 norm to measure the sizes of all vectors throughout Section 2 and Section 3. We say δ is e -stable if for any δ' such that $\|\delta - \delta'\| \leq e$, we have $l_F(\delta') = 1$.

A symbolic correction Δ is a connected set of concrete corrections. More concretely, we will use a set of linear constraints to represent a symbolic correction. We say a symbolic correction is e -stable if there exists a correction $\delta \in \Delta$ such that for any δ' where $\|\delta' - \delta\| \leq e$, we have $\delta' \in \Delta$. We call such a correction a stable region center inside Δ . To define minimality, we define the distance of Δ from the original input using the distance of a stable region center that has the smallest distance among all stable region centers. More formally:

$$\text{dis}_e(\Delta) := \min_{\delta \in S} \|\delta\|,$$

where $S := \{\delta \in \Delta \mid \forall \delta', \|\delta' - \delta\| \leq e \implies \delta' \in \Delta\}$. When Δ is not e -stable, S will be empty, so we define $\text{dis}_e(\Delta) := \infty$.

We can now define the judgement interpretation problem.

Definition 1. (Judgement Interpretation) Given a neural network F , an input vector \mathbf{v} such that $l_F(\mathbf{v}) = 0$, and a real value e , a judgement interpretation is an e -stable symbolic correction Δ with a minimum distance among all e -stable symbolic corrections.

Figure 1 shows an example of a judgement interpretation. The red cross represents the original input while the blue triangle represent the corrected inputs according to the judgement interpretation. The algorithm discovered that the person could have gotten the loan by slightly adjusting the DTI and the interest rate.

3. Our Approach

Algorithm 1 outlines our approach to find a judgement interpretation for a given neural network F and an input vector \mathbf{v} . Besides these two inputs, it is parameterized by a real e and an integer n . The former specifies the radius parameter in our stability definition, while the latter specifies how

Algorithm 1 Finding a judgment interpretation.

INPUT A neural network F and an input vector v such that $l_F(v) = 0$.
OUTPUT A judgment interpretation Δ for F and δ .
 1: **PARAM** A real value e and an integer number n .
 2: $S_n := \{s \mid s \text{ is a subarray of } [1, \dots, |v|] \text{ with length } n\}$
 3: $\Delta := \text{None}, d := +\infty$
 4: **for** $s \in S_n$ **do**
 5: $\Delta_s := \text{findProjectedInterpretation}(F, v, s, e)$
 6: **if** $\text{dis}_e(\Delta_s) < d$ **then**
 7: $\Delta := \Delta_s, d := \text{dis}_e(\Delta_s)$
 8: **end if**
 9: **end for**
 10: **return** Δ

many features are allowed to vary to produce the judgment interpretation. We parameterize the number of features to change as high-dimension interpretations are hard for end users to understand. For instance, it is much easier for a user to understand to say that, their mortgage would be approved as long as they change the DTI and the credit score while keeping the other features as they were, than to give a complex interpretation that involves all features (in our experiment, there are 21 features). The output is a judgment interpretation that is expressed in a system of linear constraints, which are in the form of

$$A * x + b \geq 0,$$

where x is a vector of variables, A is a matrix, and b is a vector.

Algorithm 1 finds such an interpretation by iteratively invoking the procedure `findProjectedInterpretation` to find an interpretation that varies a list of n features s . It returns the one with the least distance. Recall that the distance is defined as $\text{dis}_e(\Delta) = \min_{\delta \in S} \|\delta\|$, which can be evaluated by solving a sequence of linear programming problems when L_1 norm is used.

We next discuss `findProjectedInterpretation` which is the heart of our approach.

3.1. Finding A Judgment Interpretation along Given Features

In order to find a judgment interpretation, we need to find a set of linear constraints that are **minimal**, **stable**, and **verified** (that is, all inputs satisfying it will be classified as 1). None of these properties are trivial to satisfy given the complexity of any real-world neural network.

We first discuss how we address these challenges at a high level, then dive into the details of the algorithm. To address minimality, we find a single concrete correction that is minimum by leveraging an existing adversarial example

Algorithm 2 `findProjectedInterpretation`

INPUT A neural network F , an input vector v , an integer vector s , and a real number e .
OUTPUT A symbolic correction Δ_s that only changes features indexed by s .
 1: **PARAM** An integer m , the maximum number of verified linear regions to consider.
 2: $\text{regions} := \emptyset, \text{workList} := []$
 3: $\delta_0 := \text{findMinimumConcreteCorrection}(F, v, s)$
 4: $a_0 := \text{getActivations}(F, \delta_0 + v)$
 5: $L_0 := \text{getRegionFromActivations}(F, a_0, v, s)$
 6: $\text{regions} := \text{regions} \cup \{L_0\}$
 7: $\text{workList} := \text{append}(\text{workList}, a_0)$
 8: **while** $\text{len}(\text{workList}) \neq 0$ **do**
 9: $a := \text{popHead}(\text{workList})$
 10: **for** $p \in [1, \text{len}(a)]$ **do**
 11: **if** $\text{checkRegionBoundary}(F, a, p, v, s)$ **then**
 12: $a' := \text{copy}(a)$
 13: $a'[p] := \neg a'[p]$
 14: $L' := \text{getRegionFromActivations}(F, a', v, s)$
 15: **if** $L' \notin \text{regions}$ **then**
 16: $\text{regions} := \text{regions} \cup \{L'\}$
 17: **if** $\text{len}(\text{regions}) = m$ **then**
 18: $\text{workList} := []$
 19: **break**
 20: **end if**
 21: $\text{workList} := \text{append}(\text{workList}, a')$
 22: **end if**
 23: **end if**
 24: **end for**
 25: **end while**
 26: $\text{ret} := \text{inferSimplexCorrection}(\text{regions})$
 27: **return** ret

generation technique (Goodfellow et al., 2014). To generate a set of linear constraints that is stable and verifiable, we exploit the fact that ReLU-based neural networks are piecewise linear functions. Briefly, all the inputs that activate the same set of neurons can be characterized by a set of linear constraints. We can further characterize the subset of inputs that are classified as 1 by adding an additional linear constraint. Similarly, we can use a set of linear constraints to represent a set of verified corrections under certain activations. We call this set of corrections a *verified linear region* (or *region* for short). We first identify the region that the initial concrete correction belongs to, then grow the set of regions by identifying regions that are connected to existing regions. Finally, we infer a set of linear constraints whose concrete corrections are a subset of ones enclosed by the set of discovered regions.

Algorithm 2 details our approach. It starts by finding an initial region (line 3-5). We first find a minimum concrete cor-

rection δ_0 by leveraging a modified version of the fast signed gradient method (Goodfellow et al., 2014) that minimizes the L_1 distance (on line 3). More concretely, starting with a vector of 0s, we calculate δ_0 by iteratively adding a modified gradient that takes the sign of the most significant dimension among the selected features until $l_F(\mathbf{v} + \delta_0) = 1$. For example, if the original gradient is $[0.5, 1.0, 6.0, -6.0]$, the modified gradient would be $[0, 0, 1.0, 0]$ or $[0, 0, 0, -1.0]$. Then we obtain the ReLU activations \mathbf{a}_0 for $\mathbf{v} + \delta_0$ (by invoking `getActivations` on line 4), which is a boolean vector where each boolean represents whether a given neuron is activated. Finally, we obtain the initial region that δ_0 falls into by invoking `getRegionFromActivations` (on line 5), which is defined below:

```
getRegionFromActivations( $F, \mathbf{a}, \mathbf{v}, \mathbf{s}$ ) :=
    activationConstraints( $F, \mathbf{a}, \mathbf{v}$ )
     $\wedge$  classConstraints( $F, \mathbf{a}, \mathbf{v}$ )
     $\wedge$  featureConstraints( $\mathbf{s}$ ),
```

where

```
activationConstraints( $F, \mathbf{a}, \mathbf{v}$ ) :=
     $\bigwedge_{j \in [1, k]} \bigwedge_{m \in [1, |f_j|]} \{G_r^\alpha(\mathbf{x} + \mathbf{v}) \geq 0 \text{ if } \mathbf{a}[r] = True\}$ 
     $\wedge$ 
     $\bigwedge_{j \in [1, k]} \bigwedge_{m \in [1, |f_j|]} \{G_r^\alpha(\mathbf{x} + \mathbf{v}) < 0 \text{ if } \mathbf{a}[r] = False\}$ ,
    where  $G_r^\alpha(\mathbf{x} + \mathbf{v}) := \mathbf{w}_r * f_0^\alpha(f_1^\alpha(\dots f_{m-1}^\alpha(\mathbf{x} + \mathbf{v}))) + b_r$ ,
            $r := \sum_{i \in [1, j-1]} |f_i| + m$ 
```

```
classConstraints( $F, \mathbf{a}, \mathbf{v}$ ) :=  $F^\alpha(\mathbf{x} + \mathbf{v})[1] > F^\alpha(\mathbf{x} + \mathbf{v})[0]$ ,
```

```
featureConstraints( $\mathbf{s}$ ) :=  $\bigwedge_{j \neq \mathbf{s}} \mathbf{x}[j] = 0$ .
```

In the definition above, we use the notation f_i^α to refer to layer i with its activations “fixed” to \mathbf{a} . More formally, $f_i^\alpha(\mathbf{v}_i) = \mathbf{W}_i^\alpha * \mathbf{v}_i + \mathbf{b}_i^\alpha$ where \mathbf{W}_i^α and \mathbf{b}_i^α have zeros in all the rows where the activation indicated that rectifier in the original layer had produced a zero. We use k to represent the number of ReLU layers and $|f_j|$ to represent the number of neurons in the j th layer. Integer r indexes the m th neuron in j th layer. Vector \mathbf{w}_r and real number b_r are the weights and the bias of neuron r respectively. Intuitively, `activationConstraints` uses a set of linear constraints to encode the activation of each neuron.

After generating the initial region, Algorithm 1 tries to grow the set of concrete corrections by identifying regions that are connected to existing regions. How do we know whether a region is connected to another efficiently? There are 2^n regions for a network with n neurons and checking whether two sets of linear constraints intersect can be expensive on high dimensions. Intuitively, two regions are likely connected if their activations only differ by one ReLU. However, this is not entirely correct given a region is not only constrained by the activations by also the desired classification.

Our key insight is that, *since a ReLU-based neural network is a continuous function, two regions are connected if their activations differ by one neuron, and there are concrete*

corrections on the face of one of the corresponding convex hulls, and this face corresponds to the differing neuron. Intuitively, on the piece-wise function represented by a neural network, the sets of concrete corrections in two adjacent linear pieces are connected if there are concrete corrections on the boundary between them. Following the intuition, we define `checkRegionBoundary`:

```
checkRegionBoundary( $F, \mathbf{a}, p, \mathbf{v}, \mathbf{s}$ ) :=
    isFeasible(boundaryConstraints( $F, \mathbf{a}, \mathbf{v}, p$ )
     $\wedge$  classConstraints( $F, \mathbf{a}, \mathbf{v}$ )
     $\wedge$  featureConstraints( $\mathbf{s}$ ))
```

where

```
boundaryConstraints( $F, \mathbf{a}, p, \mathbf{v}$ ) :=
     $\bigwedge_{j \in [1, k]} \bigwedge_{m \in [1, |f_j|]} \{G_r^\alpha(\mathbf{x} + \mathbf{v}) = 0 \text{ if } r = p\}$ 
     $\wedge$ 
     $\bigwedge_{j \in [1, k]} \bigwedge_{m \in [1, |f_j|]} \{G_r^\alpha(\mathbf{x} + \mathbf{v}) \geq 0 \text{ if } \mathbf{a}[r] = True \text{ and } r! = p\}$ 
     $\wedge$ 
     $\bigwedge_{j \in [1, k]} \bigwedge_{m \in [1, |f_j|]} \{G_r^\alpha(\mathbf{x} + \mathbf{v}) < 0 \text{ if } \mathbf{a}[r] = False \text{ and } r! = p\}$ 
```

where $G_r^\alpha(\mathbf{x} + \mathbf{v}) := \mathbf{w}_r * f_0^\alpha(f_1^\alpha(\dots f_{m-1}^\alpha(\mathbf{x} + \mathbf{v}))) + b_r$,
 $r := \sum_{i \in [1, j-1]} |f_i| + m$

By leveraging `checkRegionBoundary`, Algorithm 2 uses a worklist algorithm to identify regions that are connected or transitively connected to the initial region until no more such regions can be found or the number of discovered regions reaches a predefined upper bound m (line 8-25).

Then Algorithm 2 tries to infer a set of linear constraints whose corresponding concrete corrections are contained in the discovered regions. Moreover, to satisfy the stability constraint, we want this set to be as large as possible. Intuitively, we want to find a convex hull (represented by the returning constraints) that is contained in a polytope (represented by the regions), such that the volume of the convex hull is maximized. Further, we infer constraints that represent a simplex rather than any convex hull, for two reasons. First, the simplicity of a simplex makes it easy for the end user to interpret; secondly, it is relatively efficient to calculate the volume of a simplex.

The procedure `inferSimplexCorrection` implements the above process using a greedy algorithm. Briefly, we first randomly choose a discovered region and randomly sample a simplex inside it. Then for each vertex, we move it by a very small distance in a random direction such that (1) the simplex is still contained in the set of discovered regions, and (2) the volume increases. The process stops until the volume cannot be increased further.

Note that our approach is sound but not optimal or complete. In other words, whenever Algorithm 1 finds a symbolic correction, the correction is verified and stable, but it is not guaranteed to be minimal. Also, when our approach fails to

find a stable symbolic correction, it does not mean that such corrections do not exist. However, in practice, we find that our approach is able to find stable corrections for most of the time and the distances of the discovered corrections are small enough to be useful (as we shall see in Section 5.2).

3.2. Discussion

We finish this section by discussing extensions and limitations of our approach.

Handling categorical features. So far we have assumed all features are reals. Categorical features are typically represented using one-hot encoding and directly applying Algorithm 2 on the embedding can result in a symbolic correction most of whose concrete corrections are invalid. To address this issue, we enumerate the embedding of different values of categorical features and apply Algorithm 2 to search symbolic corrections under different embedding. For example, suppose we only have two features where the first feature is a boolean value and the second the feature is a real number, we will get two interval symbolic corrections on the second feature. While one is obtained assuming the first feature is set to *True*, the other is obtained assuming the first feature is set to *False*.

Extending to non-ReLU-based neural networks. Our approach remains unchanged as long as the activation functions are continuous and can be approximated using a continuous piece-wise linear function. If any activation function is not continuous, the assumption that we can test whether two verified regions that differ by one activation are connected by testing the constraint corresponding to the activation breaks. If the activation functions are continuous and cannot be approximated using a piece-wise linear function, the aforementioned assumption will hold, but we will need more expressive constraints other than linear constraints to represent verified regions.

Extending to other norms. So far we have assumed that the sizes of all vectors are measured using L_1 norms. If we use other norms, our algorithm largely remains the same, except for dis_e , which measures the stability and size of a inferred symbolic correction. When norms other than L_1 or L_∞ are applied, evaluating dis_e requires solving one or more non-linear optimization problems, which can be expensive when the number of varied features (indicated by parameter n of Algorithm 1) is large.

Avoiding adversarial corrections. Adversarial inputs are inputs generated from an existing input via small perturbations such that they are indistinguishable to end users from the original input but lead to different classifications. Adversarial inputs are undesirable and often considered as “bugs”

of a neural network. For simplicity, we did not consider them in previous discussions. To avoid corrections that would result in adversarial inputs, we rely on the end user to define a threshold σ such that any concrete correction δ where $\|\delta\| > \sigma$ is considered not adversarial. Then we add $\|\mathbf{x}\| > \sigma$ as an additional constraint to each region.

4. Implementation

We implemented our approach in a tool called POLARIS. POLARIS is written in three thousand lines of Python code. To implement `findMinimumConcreteCorrection`, we used a customized version of the CleverHans library (Nicolas Papernot, 2017). To implement `isFeasible` which checks feasibility of generated linear constraints, we applied the commercial linear programming solver Gurobi 7.5.2 (Gurobi Optimization, Inc., 2018).

5. Empirical Evaluation

We evaluated POLARIS on a neural network that makes mortgage underwriting decisions.

5.1. Experiment Setup

Since there are no publicly available neural networks for mortgage underwriting, we ended up building our own network. Moreover, most of the datasets we found only provide application information and performance information of approved loans. Very few datasets provide information about rejected applications and such information is often demographic, which is not the key factor of decision making. As a result, we built a network that predicts whether an applicant will default on the mortgage based on the application information instead. We consider an application being rejected if our network predicts that the applicant will default.

Dataset. We used the Single-Family Historical Loan Performance Dataset published by Fannie Mae (2017). It is the largest publicly available single-family loan dataset. It consists of the application information and the performance information of 34 million loans issued from 2000 to 2016. The application of each loan consists of 24 features such as credit score, debt-to-income ratio, and others. We removed 3 features whose values are missing for over 20% of the dataset, which results in 21 features. To train the neural network, we split the dataset into training set, validation set, and test set at a ratio of 50%, 25%, and 25%. To evaluate our approach, we randomly selected 100 loan applications that are rejected by the network from the test set.

Neural network. We built a feedforward ReLU-based network with five hidden layers each of which has 200 hidden units using TensorFlow 1.4. It achieves an accuracy of 79% on the test set.

Table 1. Mutable features in corrections to mortgage underwriting.

	Name	Type	Range	Radius
1	Interest Rate	Real	[0, 0.1]	0.005
2	Credit Score	Integer	[300, 850]	25
3	Debt-to-Income	Real	[0.01, 0.64]	0.025
4	Loan-to-Value	Real	[0, 2]	0.025
5	Property Type	Category	[Cooperative Share, Manufactured Home, Planned Urban Development, Single-Family Home, Condominium]	N.A.

Algorithm configuration. Our approach has three parameters: the stability threshold e , the number of features allowed to change simultaneously n , and the maximum number of regions to consider m . We set $m = 100$. To produce symbolic corrections that are easy to understand, we set $n = 2$. Moreover, we limit the mutable features to five features that we consider useful in providing users feedback for loan applications, which are described in Table 1.

As for e , it is slightly more involved as we customized the operator dist_e for the mortgage application. Briefly, we used a weighted L_1 norm to evaluate the distance of the correction and a weighted L_∞ norm to evaluate the stability. For distance, we use $1 / (\max - \min)$ as the weight for each numeric feature. As for the categorical feature “property type”, we charge 1 on the distance if the minimum stable concrete correction in the symbolic correction (the minimum stable region center) would change it, or 0 otherwise. This is a relatively large penalty as changing the property type requires the applicant to switch to a different property. For stability, we define a stability radius array r and use $1/r[i]$ as the weight for feature i . If the category feature is involved, we require the symbolic corrections to at least contain two categories of the feature. Table 1 defines the range and radius of each feature. We define dist_e as follow:

$$\text{dis}_e(\Delta) := \min_{\delta \in S} \left(\frac{|\delta[1]|}{0.1 - 0} + \frac{|\delta[2]|}{850 - 300} + \frac{|\delta[3]|}{0.64 - 0.01} + \frac{|\delta[4]|}{2 - 0} + (0 \text{ if } \delta[5] \text{ leads to no change else } 1) \right),$$

where

$$S := \{ \delta \in \Delta \mid \exists 1 \leq i < j \leq 4. \forall \delta'. |\delta'[i] - \delta[i]| \leq e * r[i] \wedge |\delta'[j] - \delta[j]| \leq e * r[j] \wedge |\delta'[k] - \delta[k]| \text{ for } k \notin \{i, j\} \Rightarrow \delta' \in \Delta \}.$$

$$\cup \{ \delta \in \Delta \mid \exists i \in [1, 4] \text{ and a category } c \text{ of Feature 5 that differs from the category } \delta[5] \text{ leads to so that } \forall \delta'. |\delta'[i] - \delta[i]| \leq e * r[i] \wedge \delta'[5] = \delta[5] \text{ or } \delta'[5] \text{ leads to } c \wedge |\delta'[k] - \delta[k]| \text{ for } k \notin \{i, 5\} \Rightarrow \delta' \in \Delta \}.$$

Note when the categorical feature property type is involved, we evaluate $\text{dis}_e(\Delta)$ by solving a sequence of integer linear

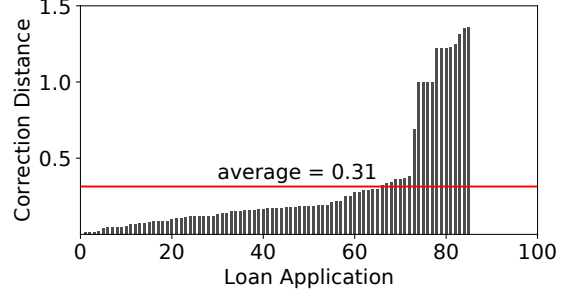


Figure 2. Distances of judgment interpretations generated for each loan application.

programming problems, which is also implemented using Gurobi. We set $e = 1$ for all runs in the experiment.

Experiment environment. All the experiments were run on a Dell XPS 8900 Desktop with 16GB RAM and an Intel I7 4GHZ quad-core processor. The operating system is Ubuntu 16.04 and the Python runtime is 3.6.

5.2. Experiment Results

We first discuss quantitatively how often POLARIS generates stable corrections and how far away these corrections are from the original input. Then we inspect some of the generated corrections in detail and discuss whether they are indeed useful for end users. Finally, we talk about how long it takes for POLARIS to generate a correction.

Stability and minimality. POLARIS successfully generated symbolic corrections for 85 applications out of 100 selected loan applications that are rejected by the neural network. For the rest 15 applications, it is either the case that the corrections found by POLARIS were discarded for being unstable, or the case that POLARIS failed to find an initial concrete correction due to the incompleteness of the applied adversarial example generation algorithm. These results show that POLARIS is effective in finding symbolic corrections that are stable and verified.

We next discuss how similar these corrections are to the original input. Figure 2 lists the sorted distances of the aforementioned 85 symbolic corrections. Recall the distance is defined using a weighted L_1 norm where the weight for each numeric feature is $1/(\max - \min)$ and we charge 1 when the categorical feature property type needs to be changed. As we can see, the average distance is only 0.31. While the largest distance is 1.36, the smallest distance is as small as 0.016. Moreover, 64% (54 out of 85) of the corrections have distances under 0.2. While the average distance is already small, the distances of the most symbolic corrections are in fact even much smaller. In conclusion, although POLARIS has no guarantee in minimality, the corrections found by it

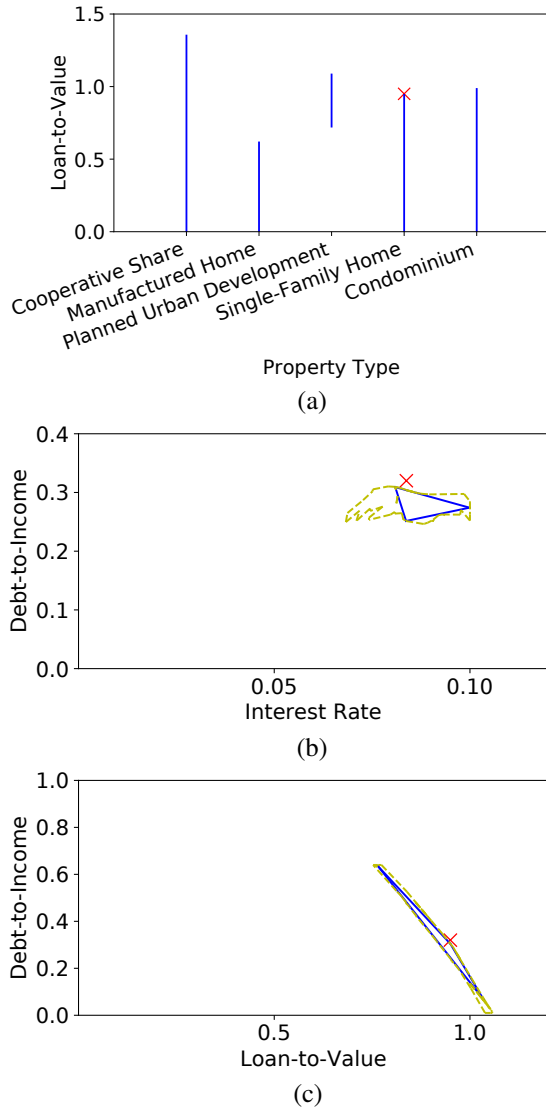


Figure 3. Different corrections generated for the application with the minimum judgment interpretation among all applications.

are often small enough to be actionable for end users.

Qualitative study. While the previous discussion gives a high-level idea of the effectiveness of our approach, we now look at individual generated symbolic corrections closely. We are interested in answering two questions:

1. Are these corrections small and stable enough such that they are actionable to the applicant?
2. Do they make sense?

Figure 3 shows the symbolic corrections generated for the application with the minimum judgment interpretation among all applications. The application corresponds to the leftmost bar in Figure 2. Since POLARIS is configured to generate corrections involving two features out of five fea-

tures, there are ten possible corrections that vary different features. For space reason, we study three of them.

Figure 3(a) shows the symbolic correction generated along loan-to-value ratio and property type, which is the minimum correction for this application. The red cross shows the projection of the original application on these two features, while the blue lines represent the set of corrected applications that the symbolic correction would lead to. First, we observe that the correction is very small. The applicant will get their loan approved if they reduce the loan-to-value ratio only by 0.0076. Such a correction is also stable. If the applicant decides to stick to single-family home properties, they will get the loan approved as long as the reduction on the loan-to-value ratio is greater than 0.0076. Moreover, they will get similar results if they switch to cooperative share properties or condominiums. This correction also makes much sense, since reducing loan-to-value ratio often means to reduce the loan value. In practice, smaller loans are easier to approve. Also, from the perspective of the training data, smaller loans are less likely to default.

Figure 3(b) shows the symbolic correction generated along debt-to-income ratio and interest rate, which are two numeric features. Similar to Figure 3(a), the red cross represents the projection of the original application, while the blue triangle represents the symbolic correction. In addition, we use a polytope enclosed in dotted yellow lines to represent the verified linear regions collected by Algorithm 2. We have two observations about the regions. First, the polytope is highly irregular, which reflects the highly nonlinear nature of the neural network. However, POLARIS is still able to generate symbolic corrections efficiently. Secondly, the final correction inferred by our approach covers most area of the regions, which shows the effectiveness of our greedy algorithm applied in `inferSimplexCorrection`. While this correction is also small and stable, its distance is larger than the previous correction along loan-to-value ratio and property type. Such a correction also makes sense from the training data perspective. It is obvious that applicants with smaller debt-to-income ratios will less likely default. As for interest rate, the correction leans towards increasing it. It might be due to the fact that during subprime mortgage crisis (2007-2009), loans were approved with irrationally low interest rate, many of which went into default later.

Figure 3(c) shows the correction generated along debt-to-income ratio and loan-to-value ratio. Compared to the previous corrections, its distance is small but it is highly unstable (the triangle is very narrow). In fact, it is discarded by POLARIS due to this.

As a comparison to corrections generated on the previous application, Figure 4 shows the final correction generated on the application that corresponds to the rightmost bar on Figure 2. In other words, its final correction has the



Figure 4. The final correction generated for the application with the maximum judgment interpretation among all loan applications.

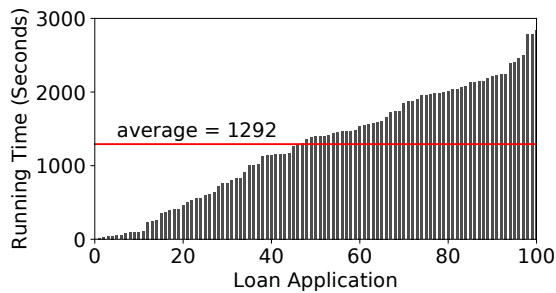


Figure 5. Running time of POLARIS on each loan application.

largest distance among final corrections generated for all applications. As the figure shows, such a large distance makes it hard for the applicant to adopt. For most categories of property type, the applicant needs to raise their credit score by 100, and even to over 800 under some cases, which is not very easy in practice. As a result, POLARIS assigns a high distance for such a correction.

Efficiency. Figure 5 shows the sorted running time of POLARIS on all loan application. It ranges from 18 seconds to 2,844 seconds with a balanced distribution. On average, POLARIS takes around 20 minutes to generate the final correction for a loan application. Given in reality mortgage underwriting usually takes days, such a running time is moderate. After manual inspection, we found the majority of the time is spent in the invocations to the linear programming solver by `checkRegionBoundary`. Although each invocation only takes a fraction of a second, there can be invocations as many as the number of neurons when a new region is being added. In the future, we plan to cut down the number of invocation by investigating sampling-based approaches.

6. Related Work

Our work is related to previous works on interpreting neural networks in terms of the problem (Montavon et al., 2017),

and works on generating adversarial examples (Goodfellow et al., 2014) in terms of the underlying techniques.

Much work on interpretability has gone into analyzing the results produced by a convolutional network that does image classification. The Activation Maximization approach and its follow-ups visualize learnt high-level features by finding inputs that maximize activations of given neurons (Erhan et al., 2009; Hinton, 2012; Lee et al., 2009; van den Oord et al., 2016; Nguyen et al., 2016). Zeiler and Fergus (2014) uses deconvolution to visualize what a network has learnt. Not just limited to image domains, more recent works try to build interpretability as part of the network itself (Pinheiro & Collobert, 2015; Lei et al., 2016; Tan et al., 2015; Wu et al., 2016; Li et al., 2017). There are also works that try to explain a neural network by learning a more interpretable model (Ribeiro et al., 2016; Krakovna & Doshi-Velez, 2016; Bastani et al., 2017). As far as we know, the problem definition of judgement interpretation is new, and none of the existing approaches can directly solve it. Moreover, these approaches typically generate a single input prototype or relevant features, but do not result in corrections or a space of inputs that would lead the prediction to move from an undesirable class to a desirable class.

Adversarial examples were first introduced by Szegedy and et al. (2013), where box-constrained L-BFGS is applied to generate them. Various approaches have been proposed later. The fast gradient sign method (Goodfellow et al., 2014) calculate an adversarial perturbation by taking the sign of the gradient. The Jacobian-based Saliency Map Attack (JSMA) (He et al., 2016) applies a greedy algorithm based a saliency map which models the impact each pixel has on the resulting classification. Deepfool (Moosavi-Dezfooli et al., 2016) is an untargeted attack optimized for the L_2 norm. Bastani (Bastani et al., 2016) applies linear programming to find an adversarial example under the same activations. While these techniques are similar to ours in the sense that they also try to find minimum corrections, the produced corrections are concrete while ours are symbolic.

7. Conclusion

We proposed a new approach to interpret a neural network by generating minimal, stable, and symbolic corrections that would change its output. Such an interpretation is a useful way to provide feedback to a user when the neural network fails to produce a desirable output. We designed and implemented the first algorithm for generating such corrections, and demonstrated its effectiveness on a neural network that does mortgage underwriting.

References

Fannie Mae single-family loan performance data. [http:](http://)

- [//www.fanniema.com/portal/funding-the-market/data/loan-performance-data.html](http://www.fanniema.com/portal/funding-the-market/data/loan-performance-data.html), 2017. Accessed: 2018-02-07.
- Bach, Sebastian, Binder, Alexander, Montavon, Grégoire, Klauschen, Frederick, Müller, Klaus-Robert, and Samek, Wojciech. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS one*, 10(7):e0130140, 2015.
- Bastani, Osbert, Ioannou, Yani, Lampropoulos, Leonidas, Vytiniotis, Dimitrios, Nori, Aditya V., and Criminisi, Antonio. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2613–2621, 2016.
- Bastani, Osbert, Kim, Carolyn, and Bastani, Hamsa. Interpretability via model extraction. *CoRR*, abs/1706.09773, 2017.
- Erhan, Dumitru, Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.
- Goodfellow, Ian J., Shlens, Jonathon, and Szegedy, Christian. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.
- Gurobi Optimization, Inc. Gurobi optimizer reference manual. <http://www.gurobi.com>, 2018.
- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016.
- Hinton, Geoffrey E. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade - Second Edition*, pp. 599–619, 2012.
- Krakovna, Viktoriya and Doshi-Velez, Finale. Increasing the interpretability of recurrent neural networks using hidden markov models. *CoRR*, abs/1606.05320, 2016.
- Lee, Honglak, Grosse, Roger B., Ranganath, Rajesh, and Ng, Andrew Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pp. 609–616, 2009.
- Lei, Tao, Barzilay, Regina, and Jaakkola, Tommi S. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pp. 107–117, 2016.
- Li, Oscar, Liu, Hao, Chen, Chaofan, and Rudin, Cynthia. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. *CoRR*, abs/1710.04806, 2017.
- Montavon, Grégoire, Samek, Wojciech, and Müller, Klaus-Robert. Methods for interpreting and understanding deep neural networks. *CoRR*, abs/1706.07979, 2017.
- Moosavi-Dezfooli, Seyed-Mohsen, Fawzi, Alhussein, and Frossard, Pascal. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 2574–2582, 2016.
- Nguyen, Anh, Dosovitskiy, Alexey, Yosinski, Jason, Brox, Thomas, and Clune, Jeff. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pp. 3387–3395, 2016.
- Nicolas Papernot, Nicholas Carlini, Ian Goodfellow Reuben Feinman Fartash Faghri Alexander Matyasko Karen Hambardzumyan Yi-Lin Juang Alexey Kurakin Ryan Sheatsley Abhibhav Garg Yen-Chen Lin. cleverhans v2.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2017.
- Pinheiro, Pedro H. O. and Collobert, Ronan. From image-level to pixel-level labeling with convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 1713–1721, 2015.
- Ribeiro, Marco Túlio, Singh, Sameer, and Guestrin, Carlos. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144, 2016.
- Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian J., and Fergus, Rob. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.
- Tan, Shawn, Sim, Khe Chai, and Gales, Mark J. F. Improving the interpretability of deep neural networks with stimulated learning. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*, pp. 617–623, 2015.
- van den Oord, Aäron, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*,

ICML 2016, New York City, NY, USA, June 19-24, 2016, pp. 1747–1756, 2016.

Wu, Chunyang, Karanasou, Penny, Gales, Mark J. F., and Sim, Khe Chai. Stimulated deep neural network for speech recognition. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pp. 400–404, 2016.

Zeiler, Matthew D. and Fergus, Rob. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pp. 818–833, 2014.