

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Abstraction-Based Neural Network Analysis by Probabilistic Abstract Interpretation

Author:

Zhuofan ZHANG

Supervisor:

Dr. Herbert WIKLICKY

A thesis presented for partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computing Science of Imperial College London

November 2025

Abstract

Probabilistic abstract interpretation is a theory that is used to extract particular properties of a computer program when it is not feasible to test every input. In this thesis, we apply the theory on neural networks for the same purpose: to analyze density distribution flow of all possible inputs of a neural network when a network has uncountably many or countable but infinitely many inputs. We show how this theoretical framework works on different types of neural networks. We then discuss different abstract domains as part of the analysis and their corresponding Moore-Penrose pseudo-inverses together with abstract transformers. We also give experimental illustrations and evaluations.

Statement of Originality

I hereby declare that this thesis is entirely my own original work. All sources, references and contributions from others have been properly cited and acknowledged. I understand that any form of plagiarism or misrepresentation is a serious academic offense, and I take full responsibility for the authenticity of this work.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Acknowledgements

I would first like to express my sincere gratitude to my supervisor, Dr. Herbert Wickle, for his kind guidance, consistent support, and patient supervision throughout this wonderful six-year journey. To me, he is not only a supervisor, but also a friend and a father. Without him, I will not have this opportunity to complete this thesis.

I would also like to send my heartfelt thanks to my family, to my mother who has always been the strongest support to my study, to my father who has always encouraged me to pursue this degree when he was alive, and to Lysol who consistently gave me the tender love and strong support during this journey.

Finally, I would like to thank my friends and all those who helped me directly or indirectly with this thesis.

Contents

1	Introduction	8
1.1	Motivation	8
1.2	Aims	8
1.3	Contributions	8
2	Background	10
2.1	Mathematical Preliminary	10
2.2	Abstract Interpretation	11
2.3	Probabilistic Abstract Interpretation	14
2.4	Neural Network	21
2.5	Neural Network Verification	24
3	Related Works	29
4	PAI on Neural Network	32
4.1	General Layer	32
4.2	Basic MLP	33
4.2.1	Integer Space with Special Weights	33
4.2.2	Integer Space with General Weights	39
4.2.3	Real Space	41
4.2.4	Re-permutation Matrix	45
4.2.5	Pseudo Inverse Matrix	46
4.2.6	Higher Dimension	47
4.2.7	Different Properties	48
4.3	Convolutional Neural Network	49
4.4	Recurrent Neural Network	51
4.5	Conclusions	55
5	Different Abstract Domains	56
5.1	Zonotope Approximation	56
5.1.1	Integer Space	59
5.1.2	Real Space	63
5.2	Grids Approximation	65
5.2.1	Square Grids	66
5.2.2	Sparser Grids	67
5.2.3	Other Shapes	68

5.3	Distribution Approximation	68
5.3.1	Polynomial Regression	69
5.3.2	Radial Basis Functions	69
5.3.3	Fourier Transform	71
5.4	Clusters Approximation	73
5.4.1	K-means	74
5.4.2	Gaussian Mixture Model	76
5.5	Conclusions	78
6	Experiments	79
6.1	Australia Rain Prediction	79
6.1.1	The Dataset	79
6.1.2	The Model	80
6.1.3	Probability Initialization	80
6.1.4	Grids Approximation	80
6.1.5	Discussion	84
6.2	Hand-written Digit Image Classification	84
6.2.1	The Dataset	84
6.2.2	The Model	85
6.2.3	Probability Initialization	85
6.2.4	Grids Approximation	85
6.2.5	Discussion	87
6.3	GPA Predictor	87
6.3.1	The Dataset	87
6.3.2	The Model	88
6.3.3	Probability Initialization	88
6.3.4	Distribution Approximation	88
6.3.5	Discussion	90
6.4	Wine Quality Classification	90
6.4.1	The Dataset	91
6.4.2	The Model	91
6.4.3	Probability Initialization	91
6.4.4	Clusters Approximation	92
6.4.5	Discussion	93
6.5	Conclusions	94
7	Conclusions and Future Work	96
	Bibliography	98

Chapter 1

Introduction

1.1 Motivation

In recent years, deep neural networks have been regarded as an essential technique for machine learning since deep learning works is progressing [1, 2] and have been playing a big role in many industries in the real world, from face recognition to text generation, from medical diagnosis to self-driving cars. However, a major problem of some deep neural network models is that it is still difficult to fully interpret how neural networks deal with information and derive their decisions in a human-understandable way, especially when layers are getting deeper. Interpreting neural networks is important because in some application areas cost of mistakes is intolerable, such as in self-driving systems, surgery robotics or in financial algorithmic trading. Interpretation and analysis methods are desired to help humans better understand how neural networks make their decisions and to ensure certain properties of neural networks, such as safety or robustness.

1.2 Aims

Our approach aims at making use of the theory of probabilistic abstract interpretation to analyze the probability density distribution flow of all possible inputs of a trained neural network. This can be used to make sure that the system is robust and under control in context of probability, by showing that a majority of data flows through propagation of neural networks are bounded in desired domain. In addition, it can also offer a sensitivity analysis of the data from the input to each layer, helping to better understand how the data flow inside a neural network. It may also help to compare different neural networks for their robustness or interpretability.

1.3 Contributions

This work offers two main contributions to the study of deep neural network analysis:

1. we introduce a novel approach to interpret trained neural networks by establishing a new analysis framework based on the theory of probabilistic abstract interpretation. We show how this framework is applied on different architectures of neural networks, from low-dimensional to higher-dimensional, from fully connected layer to convolutional layer and recurrent layer.
2. as part of the analysis framework, we introduce different abstract domains based on different properties concerned, from grids approximation, distribution approximation to clusters approximation.

This thesis is structured as follows:

Chapter 2 will give a useful background that will help to understand the contents in later chapters. It will introduce both the theory of abstract interpretation and the theory of probabilistic abstract interpretation in study of program analysis. It will also present some knowledge of the study of neural network verification. Chapter 3 will briefly present some related works to this thesis.

In Chapter 4 a detailed illustration about how the framework of probabilistic abstract interpretation can be applied to different types of neural networks will be shown. It starts from simple feed-forward neural networks. Then an application of the framework on some other typical architectures will be shown, including layers in convolutional neural networks, layers in recurrent neural networks, and beyond. Then Chapter 5 will discuss different abstract domains that fit different circumstances. It will start from the Zonotope approximation that is based on the works in existence. Then it will introduce three novel approaches including grids approximation, distribution approximation, and clusters approximation.

Chapter 6 will validate the framework through four different experiments with different architectures of neural networks, as well as applications of different abstract domains.

Finally, Chapter 7 will conclude the work and suggest some future research directions.

Chapter 2

Background

2.1 Mathematical Preliminary

This section covers some fundamental mathematical backgrounds and notations used in the thesis that may be helpful to readers in better understanding mathematical contents and avoiding ambiguity.

Vector Space and Euclidean Space. A vector space V over a field $\mathbb{F}$ is a set that is closed under finite vector addition and scalar multiplication:

(i) Vector addition $+: V \times V \rightarrow V$,

(ii) Scalar multiplication $\cdot: \mathbb{F} \times V \rightarrow V$

such that for all $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in V and all λ, μ in $\mathbb{F}$,

1. $\mathbf{v} + \mathbf{w} = \mathbf{w} + \mathbf{v}$,

2. $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$,

3. $0 + \mathbf{v} = \mathbf{v}$,

4. for every $v \in V$ there exists a $-\mathbf{v} \in V$ such that $\mathbf{v} + (-\mathbf{v}) = 0$,

5. $\lambda(\mu\mathbf{v}) = (\lambda\mu)\mathbf{v}$,

6. $1\mathbf{v} = \mathbf{v}$,

7. $\lambda(\mathbf{v} + \mathbf{w}) = \lambda\mathbf{v} + \lambda\mathbf{w}$,

8. $(\lambda + \mu)\mathbf{v} = \lambda\mathbf{v} + \mu\mathbf{v}$.

The later chapters cover many contents of matrices and matrix computations, which are all defined in vector space.

Affine Transformation and Linear Transformation. A linear transformation is any transformation $f: U \rightarrow V$ between vector spaces over $\mathbb{F}$ for which

(i) $f(x + y) = f(x) + f(y)$,

(ii) $f(\alpha x) = \alpha f(x)$

for all $x, y \in U$ and all $\alpha \in \mathbb{F}$.

An affine transformation is any transformation $f: U \rightarrow V$ for which, if $\sum_i \lambda_i = 1$,

$$f\left(\sum_i \lambda_i x_i\right) = \sum_i \lambda_i f(x_i) \quad (2.1)$$

for all sets of vectors $x_i \in U$.

By comparing these two definitions,

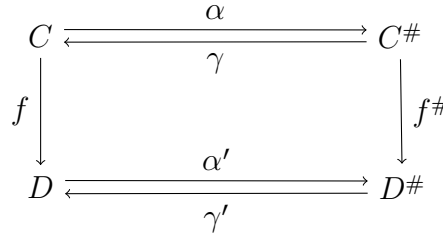


Figure 2.1: Galois Connection in Classical Abstract Interpretation

1. All linear transformations are affine transformations.
2. Not all affine transformations are linear transformations.

The later chapters include some examples that actually are linear transformations. However, to unify the expressions, we in all cases use the term affine transformation, as all linear transformations can be called affine transformations.

Matrix Dimension. We say that a matrix A has m columns and n rows in expression of

$$A = \mathbf{M}^{m \times n} \quad (2.2)$$

that differs from matrix notation that typically expresses dimension $m \times n$ as m rows and n columns. We prefer our notation in circumstance of matrix dimensions and computations in this thesis.

2.2 Abstract Interpretation Theory

Program analysis aims to offer static compile-time techniques for predicting safe and computable approximations to the set of values or behaviors arising dynamically at run-time when executing a program on a computer. A main application is to allow compilers to generate code avoiding redundant computations, e.g. by reusing available results or by moving loop invariant computations out of loops, or avoiding superfluous computations, e.g. of results known to be not needed or of results known already at compile time [20]. Main approaches to program analysis include Data Flow Analysis, Control Flow Analysis, Abstract Interpretation, Type and Effect Systems, etc. The theory of Abstract Interpretation, introduced in 1977 by Patrick and Radhia Cousot [21], is a theory of sound approximation of the semantics of computer programs, based on monotonic functions over ordered sets, especially lattices. It can be viewed as a partial execution of a computer program which gains information about its semantics without performing all the calculations. According to the classical framework, an abstract interpretation is obtained by replacing the concrete semantics by an approximating abstract one. The notion of approximation is classically encoded by suitable partial orders on the domains objects, and the correspondence between abstract and concrete domains is expressed by means of an adjoint framework based on the notion of a Galois connection, which ensures the soundness of the approximation [24].

In program analysis the property spaces are lattice [45]. Calculations on a complete

lattice can be too costly or even uncomputable, so this motivates replacing the complete lattice by a simpler lattice. We can form a Galois connection to express the relationship between two lattices, shown in Figure 3.1, where C, D are concrete domains of inputs and outputs, $C^\#, D^\#$ are abstract domains of inputs and outputs, and f is the function.

Definition 1 [45] Let $C = (C, \leq_C)$ and $C^\# = (C^\#, \leq_{C^\#})$ be two partially ordered sets. If there are two functions $\alpha : C \rightarrow C^\#$ and $\gamma : C^\# \rightarrow C$ such that for all $c \in C$ and all $c^\# \in C^\#$:

$$c \leq_C \gamma(c^\#) \iff \alpha(c) \leq_{C^\#} c^\#, \quad (2.3)$$

then $(C, \alpha, \gamma, C^\#)$ form a **Galois Connection**.

Or we can also define the Galois connection equivalently as

Definition 2 [45] Let $C = (C, \leq_C)$ and $C^\# = (C^\#, \leq_{C^\#})$ be two partially ordered sets with two order-preserving functions $\alpha : C \rightarrow C^\#$ and $\gamma : C^\# \rightarrow C$. Then $(C, \alpha, \gamma, C^\#)$ form a **Galois Connection** if and only if (i) $\forall c^\# \in C^\#, \alpha \circ \gamma(c^\#) \leq_{C^\#} c^\#$, (ii) $\forall c \in C, c \leq_C \gamma \circ \alpha(c)$.

It is obvious to see that

Proposition 1 Let $(C, \alpha, \gamma, C^\#)$ be a Galois connection. Then (i) $\alpha \circ \gamma \circ \alpha = \alpha$, and (ii) $\gamma \circ \alpha \circ \gamma = \gamma$.

In fact, we are also able to prove that given an abstraction α there is a unique concretization γ [20].

Proposition 2 Let $(C, \alpha, \gamma, C^\#)$ be a Galois connection, then (i) α uniquely determines γ by $\gamma(d) = \bigsqcup \{c \mid \alpha(c) \leq_{C^\#} d\}$, and γ uniquely determines α by $\alpha(c) = \bigsqcap \{d \mid c \leq_C \gamma(d)\}$. (ii) α is completely additive and γ is completely multiplicative, and $\alpha(\perp) = \perp$ and $\gamma(\top) = \top$.

From the Galois connection, we can formalize $f^\#$ as

$$f^\# = \alpha' f \gamma \quad (2.4)$$

where $f^\#$ is also called the **abstract transformer**. Finding the abstract transformer is always a crucial step in an abstract interpretation, as it is the key precondition that we can avoid computation in the concrete domain and directly perform computation in abstract domain, which is also the purpose to formalize an abstract interpretation. For every abstract interpretation, we first have the function f in the concrete domain, $f : X \rightarrow X$. Then we lift f to $\tilde{f} : \mathcal{P}(X) \rightarrow \mathcal{P}(X)$ by

$$\tilde{f}(Y) = \{f(y) \mid y \in Y\} \quad (2.5)$$

which is a function mapping a set of concrete points to a set of concrete points. For example, if f in the concrete domain is defined by

$$f : R \rightarrow R \quad (2.6)$$

then

$$\tilde{f} : I \rightarrow I \quad (2.7)$$

which maps intervals of real values to intervals of real values. Then in the abstract domain $f^\# : C^\# \rightarrow C^\#$ is the most precise function on $C^\#$ satisfying for all $x \in X$,

$$\alpha(\tilde{f}(X)) \sqsubseteq f^\#(\alpha(X)) \quad (2.8)$$

Example A nice example with operation of multiplication as a function and parity as property considered is shown in [45].

In the example, we have $f = . \times . : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ and abstract domain $\{even, odd\}$, shown in the lattice below.

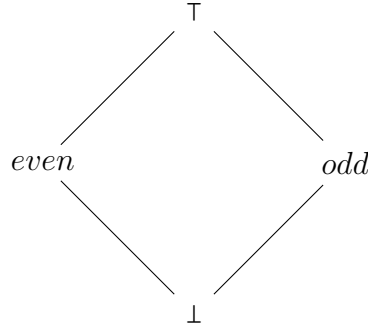


Figure 2.2: Lattice of Abstract Domain Concerning Parity

The abstraction $\alpha : C \rightarrow C^\#$ is given by for $X \subseteq \mathbb{Z}$:

- (i) $\alpha(\emptyset) = \perp = \emptyset$
- (ii) $\alpha(X) = even \iff \forall x \in X \exists k : x = 2k$
- (iii) $\alpha(X) = odd \iff \forall x \in X \exists k : x = 2k + 1$
- (iv) $\alpha(X) = \top = \{even, odd\}$ otherwise

The concretization $\gamma : C^\# \rightarrow C$ then needs to be:

- (i) $\gamma(\perp) = \emptyset$
- (ii) $\gamma(even) = \{x \in \mathbb{Z} \mid \exists k : x = 2k\} = E$
- (iii) $\gamma(odd) = \{x \in \mathbb{Z} \mid \exists k : x = 2k + 1\} = O$
- (iv) $\gamma(\top) = \top = \mathbb{Z}$ otherwise

Now to lift up $f = . \times . : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$, we have $\tilde{f} = . \tilde{\times} . : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{Z})$. For $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

$$X \tilde{\times} Y = \{x \times y \mid x \in X, y \in Y\} \quad (2.9)$$

Then in the abstract domain, $f^\# = . \times^\# . = \alpha \circ (. \tilde{\times} .) \circ (\gamma, \gamma)$. For example, if $\gamma(even) = E$, then $E \tilde{\times} E = E' \subset E$, and $\alpha(E') = even$. And if $\gamma(odd) = O$, then $E \tilde{\times} O = E$, and $\alpha(E) = even$. That is, $even \times^\# even = even$, $even \times^\# odd = even$.

2.3 Probabilistic Abstract Interpretation Theory

In addition to performing a static analysis on a deterministic program, in some cases we need to perform a probabilistic program analysis on a program in order to model and reason about uncertainty. To this purpose, we should consider a probabilistic language or languages with probabilistic semantics. There exist a number of proposals for probabilistic languages, and there are various ways to introduce probabilities into programs. One of the most common settings is random assignment, that is, the value to which a variable is assigned is chosen randomly from a set with uniform distribution, for example $x? = \{x_1, x_2, x_3\}$. Another commonly used setting is a probabilistic choice, that is, to choose between different instructions with specified probabilities, e.g. choose $p : S_1$ or $1 - p : S_2$ ro. For instance, a ‘coin flipping’ can be represented equivalently by $x? = \{0, 1\}$ or choose $0.5 : (x := 0)$ or $0.5 : (x := 1)$ ro. An example of a procedural probabilistic language is ‘pWhile’. In semantics of ‘pWhile’, the syntax of statements S is as follows:

$$\begin{aligned}
 S ::= & \text{stop} \\
 & | \text{skip} \\
 & | x := e \\
 & | x? = r \\
 & | S_1; S_2 \\
 & | \text{choose } p_1 : S_1 \text{ or } p_2 : S_2 \text{ or } \dots \text{ ro} \\
 & | \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
 & | \text{while } b \text{ do } S \text{ od}
 \end{aligned} \tag{2.10}$$

We can consider each state of variables assigned with particular values in that state. That is,

$$\mathbf{State} = (\mathbf{Var} \rightarrow \mathbf{Z} \mid \mathbf{B}) \tag{2.11}$$

Then given a ‘pWhile’ program, consider any enumerations of all its pairs of statements and states $C_1, C_2, \dots \in \mathbf{Conf}$, we have

$$T_{ij} = \begin{cases} p & \text{if } C_i = \langle S, \sigma \rangle \Rightarrow_p C_j = \langle S', \sigma' \rangle \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

is the generator of a Discrete Time Markov chain, where S denotes a statement and σ denotes a state. The transitions of this DTMC are then implemented as

$$d_n \cdot T = \sum_i (d_n)_i \cdot T_{ij} = d_{n+1} \tag{2.13}$$

where d_i is the probability distribution over $\mathbf{Conf}$ at the i th step.

Representing the program operation by a generator of Discrete-Time Markov Chain only is not enough. We also need additional linear operators where each expresses a particular operation contributing to the overall behavior of the program. Linear Operator Semantics (LOS) constructs the generator of a DTMC in a syntax-directed

fashion [26]. The syntax directed approach at the base of LOS is based in particular on a tensor (or Kronecker) product. With tensor product, we can record not only the values of all variables but also the current point in the program we are executing, the ‘program counter’ value. That means that the state space of the corresponding DTMC is configurations which also contain information about the current label. The tensor product of $\mathcal{V} \otimes \mathcal{W}$ of two vector spaces is generated by all linear combinations of the form $v \otimes w$ with $v \in \mathcal{V}$ and $w \in \mathcal{W}$.

$$\mathcal{V} \otimes \mathcal{W} = \left\{ \sum_{ij} \lambda_{ij} (v_i \otimes w_j) \mid v_i \in \mathcal{V}, w_j \in \mathcal{W} \right\} \quad (2.14)$$

That is, given a $n \times m$ matrix A and a $k \times l$ matrix B

$$A = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nm} \end{pmatrix}, B = \begin{pmatrix} b_{11} & \dots & b_{1l} \\ \dots & \dots & \dots \\ b_{k1} & \dots & b_{kl} \end{pmatrix} \quad (2.15)$$

The tensor product $A \otimes B$ is a $nk \times ml$ matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ \dots & \dots & \dots \\ a_{n1}B & \dots & a_{nm}B \end{pmatrix} \quad (2.16)$$

Also, we can use the notation of $E(m, n)$ such that

$$(E(m, n))_{ij} = \begin{cases} 1 & \text{if } m = i \wedge n = j \\ 0 & \text{otherwise} \end{cases} \quad (2.17)$$

to represent traces of program with probabilistic transitions. For instance, a probabilistic transition

$$T = \begin{pmatrix} 0 & \frac{1}{3} & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.18)$$

can be represented as

$$T = \frac{1}{3}E(1, 2) + \frac{2}{3}E(1, 3) + E(2, 4) + \frac{1}{2}E(3, 4) + \frac{1}{2}E(3, 3) + E(4, 4) \quad (2.19)$$

Recall the syntax of statements of ‘pWhile’, the set $\{\llbracket S \rrbracket_{LOS}\}$ of operators for a statement S is defined inductively as follows:

$$\begin{aligned} \llbracket [\text{skip}]^l \rrbracket_{LOS} &= \{ \langle \mathbf{I}, l \rangle \} \\ \llbracket [x := e]^l \rrbracket_{LOS} &= \{ \langle \mathbf{U}(x \leftarrow e), l \rangle \} \\ \llbracket [x ?= \rho]^l \rrbracket_{LOS} &= \left\{ \left\langle \sum_{\langle r, p \rangle \in \rho} p \cdot \mathbf{U}(x \leftarrow r), l \right\rangle \right\} \\ \llbracket S_1; S_2 \rrbracket_{LOS} &= (\llbracket S_1 \rrbracket \triangleright \text{init}(S_2)) \cup \llbracket S_2 \rrbracket \\ \llbracket \text{if } [b]^l \text{ then } S_1 \text{ else } S_2 \text{ fi} \rrbracket_{LOS} &= \{ \langle \mathbf{P}(b), l \rangle \triangleright \text{init}(S_1) \} \cup \llbracket S_1 \rrbracket_{LOS} \cup \\ &\quad \{ \langle \mathbf{P}(b)^\perp, l \rangle \triangleright \text{init}(S_2) \} \cup \llbracket S_2 \rrbracket_{LOS} \\ \llbracket \text{while } [b]^l \text{ do } S \text{ od} \rrbracket_{LOS} &= \{ \langle \mathbf{P}(b), l \rangle \triangleright \text{init}(S) \} \cup \llbracket S \rrbracket_{LOS} \cup \{ \langle \mathbf{P}(b)^\perp, l \rangle \} \end{aligned} \quad (2.20)$$

where we define the continuation operation $\langle \mathbf{F}, l \rangle \triangleright l' = \mathbf{F} \otimes \mathbf{E}(l, l')$ and we use the notations of elementary update and test operators $\mathbf{U}$ and $\mathbf{P}$. Note that we omit the **choose** statement here, as it can always be implemented by a combination of random assignment and conditional statement. A more detailed matrix expression for each statement in LOS is normally implemented case by case. We will give some examples here.

An assignment $x := 4$ where $x \in \{1, \dots, 8\}$ gives

$$\mathbf{U}(x \leftarrow 4) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.21)$$

A shift assignment $x := x + 1$ where $x \in \{1, \dots, 8\}$ gives

$$\mathbf{U}(x \leftarrow x + 1) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.22)$$

A random assignment $x \text{ ?} = \{4, 5\}$ where $x \in \{1, \dots, 8\}$ gives

$$\frac{1}{2}\mathbf{U}(x \leftarrow 4) + \frac{1}{2}\mathbf{U}(x \leftarrow 5) = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \end{pmatrix} \quad (2.23)$$

A conditional statement **if** $\text{even}(x)$ **then** $x := x/2$ **else** $y := y + 1$ **fi** gives

$$\mathbf{P}(\text{even}(x)) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \otimes I \quad (2.24)$$

$$\mathbf{P}(\text{even}(x))^\perp = I - \mathbf{P}(\text{even}(x)) \quad (2.25)$$

and

$$\mathbf{U}(x \leftarrow x/2) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \otimes I \quad (2.26)$$

$$\mathbf{U}(y \leftarrow y + 1) = I \otimes \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad (2.27)$$

We can extend these to the multi-variable case when a variable in a program has dimension greater than 1. For testing we have operators on $\mathcal{V}(\mathbf{Value})^{\otimes n}$ if we are in a classical state s or if an expression e evaluates to a constant v :

$$\mathbf{P}(s) = \bigotimes_{i=1}^n \mathbf{P}(x_i = s(x_i)) \quad \mathbf{P}(e = v) = \sum_{\varepsilon(e)s=v} \mathbf{P}(s) \quad (2.28)$$

and we have operators on $\mathcal{V}(\mathbf{Value})^{\otimes n}$ for updating a variable x_k in the context of other variables to a constant v or to the value of an expression e :

$$\mathbf{U}(x_k \leftarrow v) = \bigotimes_{i=1}^{k-1} I \otimes \mathbf{U}(v) \otimes \bigotimes_{i=k+1}^n I \quad \mathbf{U}(x_k \leftarrow e) = \sum_v \mathbf{P}(e = v) \mathbf{U}(x_k \leftarrow v) \quad (2.29)$$

As we model the semantics of a complete program, we also need to consider the control flows of the program. For each statement, we include a tensor product of $\mathbf{E}(i, j)$ to represent control flow steps.

$$\mathbf{T}(S) = \sum_{\langle i, p_{ij}, j \rangle \in \text{flow}(S)} p_{ij} \cdot \mathbf{T}(\langle l_i, p_{ij}, l_j \rangle) \quad (2.30)$$

where

$$\mathbf{T}(\langle l_i, p_{ij}, l_j \rangle) = \mathbf{N}_{l_i} \otimes \mathbf{E}(l_i, l_j) \quad (2.31)$$

and $\mathbf{N}_{l_i}$ the operator representing a state update in the block with the label l_i . With this expression, we can construct Linear Operator Semantics in a compositional way.

Example Consider the following labeled program:

```

if [x == 0]1 then
  [x ← 0]2;
else
  [x ← 1]3;
end if;
[stop]4

```

We can construct a Linear Operator Semantics as follows:

$$\begin{aligned} \mathbf{T}(S) = & \mathbf{P}(x = 0) \otimes \mathbf{E}(1, 2) + \mathbf{P}(x \neq 0) \otimes \mathbf{E}(1, 3) \\ & + \mathbf{U}(x \leftarrow 0) \otimes \mathbf{E}(2, 4) + \mathbf{U}(x \leftarrow 1) \otimes \mathbf{E}(3, 4) \\ & + I \otimes \mathbf{E}(4, 4) \end{aligned} \quad (2.32)$$

which in matrix representation is

$$\begin{aligned} \mathbf{T}(S) = & \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \mathbf{E}(1, 2) + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \mathbf{E}(1, 3) \\ & + \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \mathbf{E}(2, 4) + \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \otimes \mathbf{E}(3, 4) \\ & + I \otimes \mathbf{E}(4, 4) \end{aligned} \quad (2.33)$$

$$\begin{aligned} \mathbf{T}(S) = & \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ & + \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ & + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ = & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \quad (2.34)$$

Recall the Abstract Interpretation theory mentioned in the previous section. The classical framework of the theory originally introduced for deterministic programs can be extended to include the treatment of probabilistic programs by considering the appropriate domains [25]. Though approximations in classical abstract interpretation will always be safe, they might be not precise enough and therefore useless, addressing a worst-case scenario. In many situations, we need to consider a quantitative version of the framework, known as Probabilistic Abstract Interpretation, a theory of linear operators on Hilbert spaces, to give a not safe but a more average case scenario that provides more quantitative information.

Probabilistic Abstract Interpretation is, similar to the classical framework, also based

on a pair of concrete and abstract domains, except that they are now vector spaces instead of lattices. Instead of forming a Galois connection, we take a Moore-Penrose pseudo inverse, and assume that the pair of abstraction and concretization functions are again structure preserving.

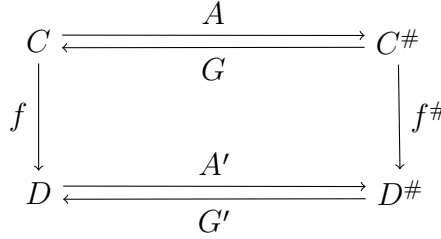


Figure 2.3: Probabilistic Abstract Interpretation with Moore-Penrose Pseudo-inverse

Definition 3 Let C and $C^\#$ be two (finite-dimensional) vector (Hilbert) spaces and $A : C \rightarrow C^\#$ a linear map. Then the linear map $G : C^\# \rightarrow C$ is the Moore-Penrose pseudo-inverse of A iff

(i) $A \circ G = P_A$,

(ii) $G \circ A = P_G$,

where P_A and P_G denote orthogonal projections onto the ranges of A and G .

Note that if A is Moore-Penrose invertible, its Moore-Penrose pseudo-inverse $A^\dagger$ satisfies:

(i) $AA^\dagger A = A$,

(ii) $A^\dagger AA^\dagger = A^\dagger$,

In Probabilistic Abstract Interpretation, the notion of approximation is characterized in terms of least square approximation.

Definition 4 Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then $u \in \mathbb{R}^n$ is called a least square solution to $Ax = b$ if $\|Au - b\| \leq \|Av - b\|$ for all $v \in \mathbb{R}^n$.

Similarly to classical abstract interpretation, from the diagram of Moore-Penrose pseudo-inverse, we can formalize our abstract transformer $f^\#$ as

$$f^\# = A' f G \quad (2.35)$$

Again, to construct and compute a probabilistic approximation, we need to consider the function f in the concrete domain, and then $\vec{f}$ in probabilistic domain lifted from f and finally $f^\#$ in abstract domain. Given in concrete domain $f : R \rightarrow R$, we lift f into $\vec{f}$ in probabilistic domain first by vector space lifting

$$\vec{f} : \mathcal{V}(R) \rightarrow \mathcal{V}(R), \quad \vec{f}(p_1 \cdot \vec{c}_1 + p_2 \cdot \vec{c}_2 + \dots) = p_1 \cdot f(c_1) + p_2 \cdot f(c_2) + \dots \quad (2.36)$$

and then lifting to power set by the support set

$$\text{supp} : \mathcal{V}(C) \rightarrow \mathcal{P}(C) \quad \text{supp}(\vec{x}) = \{c_i \mid \langle c_i, p_i \rangle \in \vec{x}, p_i \neq 0\} \quad (2.37)$$

where vector space $\mathcal{V}(C)$ is constructed as the set of all formal linear combinations of elements in set C

$$\mathcal{V}(C) = \left\{ \sum p_c c \mid p_c \in \mathbb{R}, c \in C \right\} \quad (2.38)$$

We can denote a vector in $\mathcal{V}(C)$ equivalently either as a formal sum $\sum_{c \in C} p_c c$, or as an n -tuple $(p_c)_{c \in C}$ once we fix an enumeration of C , or as a set of pairs $\{ \langle c, p_c \rangle \}_{c \in C}$.

Example Consider the following example program.

```

[m ← i]1;
while [n > 1]2 do
  [m ← m × n]3;
  [n ← n - 1]4;
end while;
[stop]5

```

We take parity of program variables as a property considered. The parity abstract operator on $\mathcal{V}(\{0, \dots, n\})$ with n even is

$$A_p = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}, A^\dagger = \begin{pmatrix} 2/n & 0 & 2/n & 0 & \dots & 0 \\ 0 & 2/n & 0 & 2/n & \dots & 2/n \end{pmatrix} \quad (2.39)$$

We can construct the abstract semantics

$$\begin{aligned} \mathbf{T}^\# &= \mathbf{U}^\#(m \leftarrow i) \otimes \mathbf{E}(1, 2) + \mathbf{P}^\#(n > 1) \otimes \mathbf{E}(2, 3) \\ &\quad + \mathbf{P}^\#(n \leq 1) \otimes \mathbf{E}(2, 5) + \mathbf{U}^\#(m \leftarrow m \times n) \otimes \mathbf{E}(3, 4) \\ &\quad + \mathbf{U}^\#(n \leftarrow n - 1) \otimes \mathbf{E}(4, 2) + I^\# \otimes \mathbf{E}(5, 5) \end{aligned} \quad (2.40)$$

which in a matrix representation is

$$\mathbf{U}^\#(m \leftarrow 1) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} \quad (2.41)$$

$$\mathbf{U}^\#(n \leftarrow n - 1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix} \quad (2.42)$$

$$\mathbf{P}^\#(n > 1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} \quad (2.43)$$

$$\mathbf{P}^\#(n \leq 1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix} \quad (2.44)$$

$$\begin{aligned} \mathbf{U}^\#(m \leftarrow m \times n) &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} \\ &+ \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \dots \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \dots \end{pmatrix} \end{aligned} \quad (2.45)$$

2.4 Neural Network

A neural network is a graph where each node performs an operation. Overall, the graph represents a function from vectors of real numbers to vectors of real numbers, that is, a function in $\mathbb{R}^n \rightarrow \mathbb{R}^m$ [46]. Typical types of neural networks include multilayer perceptrons(MLP), convolutional neural networks and recurrent neural networks. In this thesis we always consider neural networks with weights already well-trained.

Multilayer Perceptrons. A layer in a multilayer perceptron with Rectified Linear Unit(ReLU) activation function can be expressed as

$$x^{t+1} = \text{ReLU}(W^t x^t + b^t) \quad (2.46)$$

where t denotes the layer number, x^t is the input of the layer, x^{t+1} is the output, W^t and b^t are the weights. ReLU is one of choices of activation functions, and we use this notation instead of f to avoid ambiguity from f in abstract interpretation.

Convolutional Neural Networks. Convolutional neural networks are one of the commonly used neural networks in applications. It consists of particularly designed

layers such as the convolutional layer and the pooling layer. A convolutional layer in principle is also a special case of fully connected layers, except that most of weights other than the convolution mask are zeros. We will again illustrate an example of applying framework of probabilistic abstract interpretation on a convolutional layer. Let the input space $X \in \mathbb{R}^{4 \times 4}$, the convolution filter $g \in \mathbb{R}^{2 \times 2}$, and then the output space be $Y \in \mathbb{R}^{(4-2+1) \times (4-2+1)} = \mathbb{R}^{3 \times 3}$. As a well-trained model, the weights of g are already given and fixed. Let

$$g = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \quad (2.47)$$

To express the convolution computation in mathematical expression, we can have

$$y = ReLU(Wx + b) \quad (2.48)$$

where

$$W = \begin{pmatrix} g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g_{11} & g_{12} & 0 & 0 & g_{21} & g_{22} \end{pmatrix} \quad (2.49)$$

A pooling layer is also a commonly used type of layer in convolutional neural networks, typically following convolutional layers. It reduces the spatial dimensions of the input volume while keeping important features of images. It helps make the network more computationally efficient and reduces overfitting. The filter size and stride are two critical hyper-parameters that control how the input matrices are down-sampled. The filter size is the size of the sliding window that moves across input matrix, and the stride is how many pixels the filter moves each step.

In this example, we take a max pooling layer which takes the max value among all in each sliding window. let filter size $f = 2$ and stride $s = 1$. In mathematical expression, we have

$$P = \begin{pmatrix} f_{1,2,4,5}(1) & f_{1,2,4,5}(2) & 0 & f_{1,2,4,5}(4) & f_{1,2,4,5}(5) & 0 & 0 & 0 & 0 \\ 0 & f_{2,3,5,6}(2) & f_{2,3,5,6}(3) & 0 & f_{2,3,5,6}(5) & f_{2,3,5,6}(6) & 0 & 0 & 0 \\ 0 & 0 & 0 & f_{4,5,7,8}(4) & f_{4,5,7,8}(5) & 0 & f_{4,5,7,8}(7) & f_{4,5,7,8}(8) & 0 \\ 0 & 0 & 0 & 0 & f_{5,6,8,9}(5) & f_{5,6,8,9}(6) & 0 & f_{5,6,8,9}(8) & f_{5,6,8,9}(9) \end{pmatrix} \quad (2.50)$$

where

$$f_{i_1, i_2, i_3, i_4}(j) = \begin{cases} 1 & j = \arg \max(x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}) \\ 0 & \text{otherwise} \end{cases} \quad (2.51)$$

Or if we take an average pooling that takes an average value in each sliding window

$$P = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix} \quad (2.52)$$

as the sliding window size is $2 \times 2 = 4$. This pooling will reduce dimension of the input matrix $X \in \mathbb{R}^{3 \times 3}$ to the output matrix $Y \in \mathbb{R}^{2 \times 2}$.

Instead of performing only one convolution on the input matrix, a typical convolutional layer will have more than one channel, that is, for each channel it performs a convolution with a distinct convolution mask. Each channel represents a different aspect of the input features. To write this in mathematical expression, it is straightforward that we concatenate matrices of convolutions vertically, performing convolutions for each channel one by one.

For example, continue with convolution mask g in the above section, say if we have three channels to extract three aspects of features of input, then we have three masks g', g'', g''' such that

$$g' = \begin{pmatrix} g'_{11} & g'_{12} \\ g'_{21} & g'_{22} \end{pmatrix}, g'' = \begin{pmatrix} g''_{11} & g''_{12} \\ g''_{21} & g''_{22} \end{pmatrix}, g''' = \begin{pmatrix} g'''_{11} & g'''_{12} \\ g'''_{21} & g'''_{22} \end{pmatrix} \quad (2.53)$$

Then we have

$$W = \begin{pmatrix} g'_{11} & g'_{12} & 0 & 0 & g'_{21} & g'_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g'_{11} & g'_{12} & 0 & 0 & g'_{21} & g'_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g'_{11} & g'_{12} & 0 & 0 & g'_{21} & g'_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g'_{11} & g'_{12} & 0 & 0 & g'_{21} & g'_{22} \\ g''_{11} & g''_{12} & 0 & 0 & g''_{21} & g''_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g''_{11} & g''_{12} & 0 & 0 & g''_{21} & g''_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g''_{11} & g''_{12} & 0 & 0 & g''_{21} & g''_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g''_{11} & g''_{12} & 0 & 0 & g''_{21} & g''_{22} \\ g'''_{11} & g'''_{12} & 0 & 0 & g'''_{21} & g'''_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g'''_{11} & g'''_{12} & 0 & 0 & g'''_{21} & g'''_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g'''_{11} & g'''_{12} & 0 & 0 & g'''_{21} & g'''_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g'''_{11} & g'''_{12} & 0 & 0 & g'''_{21} & g'''_{22} \end{pmatrix} \quad (2.54)$$

This will return all feature maps(convolution matrices) for each channel, represented by one single vector. The pooling for each channel can be performed in a parallel way. Recall the average pooling matrix and if we take average pooling in a 3-channel case

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix} \quad (2.55)$$

Recurrent Neural Networks. Recurrent neural networks are popular for sequence models. A one-layer recurrent neural network contains the inputs X , the hidden states H , the output Y , and weights $W_{hx}, W_{hh}, W_{hy}, W_{yh}$. The update process for

each recursion is as follows:

$$h^t = \tanh(W_{hh}h^{t-1} + W_{hx}x^t + b_h), t \leq T \quad (2.56)$$

$$h^t = \tanh(W_{hh}h^{t-1} + W_{hy}y^{t-T} + b_h), t \geq T + 1 \quad (2.57)$$

$$y^{t-T} = \tanh(W_{yh}h^{t-1} + b_y), t \geq T + 1 \quad (2.58)$$

while b is the constant vector, $t \in \mathbb{N}$ is the number of recursion. We use the $\tanh$ function as an activation function, which is commonly used in recurrent networks. For simplicity, we can let the inputs and outputs be two-dimensional such that $x = (x_1, x_2)$, $h = (h_1, h_2)$, $y = (y_1, y_2)$, $b = (b_1, b_2)$, and assume after training

$$W_{hx} = \begin{pmatrix} a_{hx,11} & a_{hx,21} \\ a_{hx,12} & a_{hx,22} \end{pmatrix}, W_{hh} = \begin{pmatrix} a_{hh,11} & a_{hh,21} \\ a_{hh,12} & a_{hh,22} \end{pmatrix} \quad (2.59)$$

$$W_{hy} = \begin{pmatrix} a_{hy,11} & a_{hy,21} \\ a_{hy,12} & a_{hy,22} \end{pmatrix}, W_{yh} = \begin{pmatrix} a_{yh,11} & a_{yh,21} \\ a_{yh,12} & a_{yh,22} \end{pmatrix}, h^0 = \begin{pmatrix} h_1^0 \\ h_2^0 \end{pmatrix}, b_h = \begin{pmatrix} b_{h1} \\ b_{h2} \end{pmatrix}, b_y = \begin{pmatrix} b_{y1} \\ b_{y2} \end{pmatrix} \quad (2.60)$$

and

$$x_1^t, x_2^t \in \mathbb{R} \quad (2.61)$$

for all t .

2.5 Abstraction-Based Neural Network Verification

Neural networks nowadays have been widely used in different areas and in many practical situations it is important to verify whether a neural network behaves according to some set of desirable properties [46]. These fields of study, verification and neural networks, have been two distinct areas of computing research with almost no bridges connecting them until very recently. So for what properties do we normally expect a neural network to satisfy? Here are some examples.

Robustness. The most studied correctness property of neural networks is robustness. Robustness means that small perturbations to inputs will not result in changes to the output of the neural network. For example, changing the lightness of particular pixels in an image of a road sign as input will not make the neural network have different understanding of what the road sign says. This property is essential for a neural network to have the ability to counter adversarial attacks, especially in areas where mistakes have huge costs.

Safety. Safety is a broad class of correctness properties stipulating that a program should not go to a bad state. An example is a neural network that implements a collision avoidance system for aircrafts [36]. We want to keep two aircrafts from any collision, so if an intruding aircraft is approaching from the left, the neural network should decide to turn the aircraft right. If a neural network does not satisfy the property of safety, it may result in very serious consequence, even hurting human lives.

Probabilistic Properties. In addition to classical abstraction-based verification, besides from ‘hard’ properties such like robustness or safety, sometimes we also want

our neural network to satisfy ‘soft’ properties such as probabilistic properties, especially when ‘hard’ properties cannot be determined or directly proved. A probabilistic property shows that a neural network will satisfy a particular property or will give a particular kind of outputs in what likeliness. We can see how the neural network will work or make decisions in what kind of pattern of likeliness. Sometimes, although we cannot prove that a neural network satisfies a property, but we can show that this property holds in very high likeliness, and the exception is negligible in the considered situation. Also, we can see how the likeliness pattern of a neural network will change if we adjust the neural network, so we can determine if it is a desirable adjustment. This is what our framework attempts to achieve.

Neural network verification techniques can be divided into two categories, namely constraint-based techniques and abstraction-based techniques. In constraint-based techniques, we construct a system of constraints and solve it to prove whether a neural network satisfies some properties of interest. In abstraction-based techniques, instead of executing a neural network on a single input, we can actually execute it on an infinite set and show that all of those inputs satisfy desirable correctness properties. The abstraction-based approach is mainly based on abstract interpretation theory.

In abstraction-based verification, the problem we face is that there are infinitely many possible inputs, or there are finitely many but the number is enormous. In either case, we cannot conceivably run all cases of input through the network to check whether the output satisfies a desired property. So we need to generate a function, called abstract transformer, which is able to take an infinite set of inputs and perform a mapping exactly the same as the neural network tends to perform, and return a set of outputs that can be proved to satisfy the property. Therefore, the most crucial and difficult step is to define an abstract domain and generate the corresponding abstract transformer.

Let us look at some examples of abstract transformers in the interval abstract domain. Specifically, the interval abstract domain considers an interval of real numbers written as $[l, u]$, where $l, u \in \mathbb{R}$ and $l \leq u$. An interval $[l, u]$ denotes the set $\{x \mid l \leq x \leq u\}$. In this abstract domain, an abstract transformer takes a set of real numbers and returns a set of real numbers, but the sets are restricted to those that can be defined as intervals. [46] Let’s begin with arithmetic operations.

Addition. Consider the binary function: $f(x, y) = x + y$, where $l \leq x \leq u, l' \leq y \leq u', x \in \mathbb{R}, y \in \mathbb{R}$, and l, u, l', u' are fixed values. Then, the abstract transformer $f^\#$ of this addition function is

$$f^\#([l, u], [l', u']) = [l + l', u + u'] \quad (2.62)$$

Multiplication. Consider the binary function: $f(x, y) = x * y$, where $l \leq x \leq u, l' \leq y \leq u', x \in \mathbb{R}, y \in \mathbb{R}$, and l, u, l', u' are fixed values. Then the abstract transformer $f^\#$ of

this multiplication function is

$$f^\#([l, u], [l', u']) = [\min(B), \max(B)] \quad (2.63)$$

where

$$B = \{l * l', l * u', u * l', u * u'\} \quad (2.64)$$

Note that it is not necessary that all the concrete elements in the output set here can be reached. We only have to make sure that the output set covers all concrete values that will be reached. This is the soundness of the classical abstract-based verification.

Affine Functions. For an affine function $f(x_1, \dots, x_n) = \sum_i^n c_i x_i$ where $c_i \in \mathbb{R}$ and $l_i \leq x_i \leq u_i, x_i \in \mathbb{R}$, l_i denotes the lower bound and u_i denotes the upper bound of x_i . The abstract transformer $f^\#$ of this affine function is

$$f^\#([l_1, u_1], \dots, [l_n, u_n]) = [\sum_i l'_i, \sum_i u'_i] \quad (2.65)$$

where $l'_i = \min(c_i l_i, c_i u_i)$ and $u'_i = \max(c_i l_i, c_i u_i)$.

Monotonic Functions. Most activation functions used in neural networks are monotonically increasing. It is straightforward to define an abstract transformer for any monotonically increasing function $f : \mathbb{R} \rightarrow \mathbb{R}$ as

$$f^\#([l, u]) = [f(l), f(u)] \quad (2.66)$$

Composition of Functions. It is straightforward that

$$(f \circ g)^\# = f^\# \circ g^\# \quad (2.67)$$

Neural Network. Define a neural network as a graph $G = (V, E)$, and a function representing this neural network $f_G = \mathbb{R}^n \rightarrow \mathbb{R}^m$. Then the abstract transformer $f_G^\#$ is a function that takes a n -dimensional space and returns a m -dimensional space, that is, takes n intervals and returns m intervals. Specifically, it is defined backward-recursively as

$$f_G^\#([l_1, u_1], \dots, [l_n, u_n]) = (f^\#(v_{y_1}), \dots, f^\#(v_{y_m})) \quad (2.68)$$

where $v_{y_1}, \dots, v_{y_m}$ are output nodes of this neural network, and

$$f^\#(v_{x_i}) = [l_{x_i}, u_{x_i}] \quad (2.69)$$

if v_{x_i} is an input node, and

$$f^\#(v_i) = f_{v_i}^\#(f^\#(v_1), \dots, f^\#(v_k)) \quad (2.70)$$

if v_i is a non-input node and has incoming edges $(v_1, v_i), \dots, (v_k, v_i)$.

One weakness of the interval domain expressed above, or we can call it the abstract domain of Box approximation, is that it is non-relational, as it is not able to

capture relationships between variables. Usually we want to keep more precision and capture as many relationships between variables as possible. The abstract domain of the Zonotope approximation achieves better on this purpose.

Zonotope Approximation. We assume that we have a set of m real-valued generator variables, denoted $\epsilon_1, \dots, \epsilon_m$, for each $\epsilon_i \in [-1, 1]$. Then a Zonotope with m generators in n -dimensions is the set of all points

$$\{(c_{1,0} + \sum_{i=1}^m c_{1,i} \cdot \epsilon_i, \dots, c_{n,0} + \sum_{i=1}^m c_{n,i} \cdot \epsilon_i) \mid \epsilon_i \in [-1, 1]\} \quad (2.71)$$

or for simpler notation

$$(< c_{1,i} >, \dots, < c_{n,i} >) \quad (2.72)$$

where $< c_{k,i} >$ denotes the k -th dimension. Intuitively, when two different dimensions share the same generator, this relationship between these two dimensions is captured. Like those in abstract domain of Box approximation, in abstract domain of Zonotope approximation we also have abstract transformers for the above functions.

Affine Function in Zonotope Approximation. For an affine function

$$f(x_1, \dots, x_n) = \sum_j a_j x_j \quad (2.73)$$

where $a_j \in \mathbb{R}$, the abstract transformer for this affine function is

$$f^\#(< c_{1,i} >, \dots, < c_{n,i} >) = < \sum_j a_j c_{j,0}, \dots, \sum_j a_j c_{j,m} > \quad (2.74)$$

ReLU in Zonotope Approximation. There are many different kinds of activation functions used in neural networks. The Rectified Linear Unit(ReLU) is a typical kind. In the Zonotope approximation, we can always define different abstract transformers for the same function. For ReLU, an approach is to define an abstract transformer of a 1-dimensional input $[l, u]$ by approximation of a parallelogram as follows:

$$f^\#(< c_1, \dots, c_m >) = \begin{cases} < c_i >, & \text{for } l \geq 0 \\ < 0 >, & \text{for } u \leq 0 \\ < \lambda c_i, \dots, \lambda c_m, 0 > + < \mu, 0, 0, \dots, \mu > & \text{otherwise} \end{cases} \quad (2.75)$$

where $\lambda = \frac{u}{u-l}$ and $\mu = \frac{u(1-\lambda)}{2}$.

Neural Network in Zonotope Approximation. Define a neural network as a graph $G = (V, E)$, and a function representing this neural network $f_G = \mathbb{R}^n \rightarrow \mathbb{R}^m$. Then the abstract transformer $f_G^\#$ is a function that takes a n -dimensional Zonotope and returns a m -dimensional Zonotope. Specifically, it is defined recursively as

$$f_G^\#(< c_{1j} >, \dots, < c_{nj} >) = (f^\#(v_{y_1}), \dots, f^\#(v_{y_m})) \quad (2.76)$$

where

$$f^\#(v_{x_i}) = \langle c_{x_i j} \rangle_j \quad (2.77)$$

if v_{x_i} is an input node, and

$$f^\#(v_i) = f_{v_i}^\#(f^\#(v_1), \dots, f^\#(v_k)) \quad (2.78)$$

where $f_{v_i}^\#$ is the abstract transformer of f_{v_i} , v_i is a non-input node and has incoming edges $(v_1, v_i), \dots, (v_k, v_i)$.

After defining an abstract domain that allows us to evaluate a neural network on an infinite set of inputs, we verify specific properties of a neural network using the defined abstract domain and abstract transformers. To verify a property, we follow

$$\begin{aligned} &\{precondition\} \\ &\quad r \leftarrow f(x) \\ &\{postcondition\} \end{aligned} \quad (2.79)$$

where the precondition defines a set of possible values for the inputs of the neural network, and the postcondition defines a set of possible correct values of the outputs of the neural network. To verify such properties with abstract interpretation, we need to perform three tasks:

1. Soundly represent the set of values of x in the abstract domain.
2. Abstractly interpret the neural network f on all values of x , resulting in an over-approximation of values of r .
3. Check that all values of r satisfy the post-condition.

For example, to prove the robustness in a classification problem, we prove that the following property given the label of c is y :

$$\begin{aligned} &\{\|x - c\| \leq \epsilon\} \\ &\quad r \leftarrow f(x) \\ &\{\text{class}(c) = y\} \end{aligned} \quad (2.80)$$

where $\|x\|$ is a particular norm of a vector and $\epsilon > 0$. To check the post-condition, we check if the dimension y is the largest dimension in the Zonotope approximation.

In addition to the properties of neural networks mentioned above, abstract-based verification is also used to prove properties in natural language processing, such as robustness. For different problem domains and different properties of interest, defining an appropriate abstract domain and abstract transformers is always crucial. In practice, there is always a precision–scalability trade-off: more precision of the abstract domain means more complex constraints, and therefore slower verification. In addition, abstraction-based techniques have been successfully applied to relatively large neural networks, with up to a million neurons along more than thirty layers [47]. However, research efforts are still being made on whether verification will work for more complex perceptual notions of robustness, e.g. rotating an image or changing the virtual background on a video, or other more complex properties and domains, e.g. malware detection.

Chapter 3

Related Works

Research on interpreting and analyzing neural networks has been carried out for years. Neural networks have been commonly used but deep models have been “black boxes” and not easily interpretable. The questions are when a neural network has a good performance on its task, has it really learned something and will generally perform well or does the performance just come from exploitation of artifacts and only for this specific dataset, and why does it perform well? The internal structure and parameters needs to be discussed and the way how it made decisions needs to be explained in a human-understandable language. This is especially important in applications such as medicine or self-driving cars where the reliance of the model is highly concerned. Also, researchers have been interested in looking for a valid analysis or verification of certain properties of a neural network, such as robustness or resistance, to ensure correct features are guaranteed.

At the beginning, one of the approaches was through rule extraction algorithms. It can mainly be classified into three kinds: compositional approach if the algorithm is based on neurons, pedagogical approach if the algorithm works on the whole network irrespective of architecture, and eclectic approach, which is a combination of the previous two [3]. In the later period, feed-forward neural networks, convolutional neural networks and recurrent neural networks are becoming more popular. In 2013 Simonyan et al. proposed a technique using Activation Maximization to interpret model prediction by only looking at its relationships with input, disregarding the inner process [4]. A further development of the technique was to add a decoder on it [5]. In the same year, Bazen et al. used Taylor’s decomposition to explain the decision of the model by decomposing the output as a sum of relevance scores [6], and Landecker et al. decomposed the prediction of the model by backpropagating relevance in reverse direction, progressively redistributing the prediction score until the input is reached [7]. Based on Landecker’s method, Bach et al. further introduced the Layer-wise Relevance Propagation explanation framework [8]. Another measurement was to use sensitivity analysis to quantify the importance of each input variable [9]. In 2015 Zilke proposed DeepRED, by extending a compositional algorithm CRED [10]. In 2017, Montavon further reviewed some state-of-the-art interpretation methods for deep neural networks, based on a tutorial given at ICASSP 2017 [11]. Jacobsson [12] and Wang [13] had proposed algorithms of understand-

ing recurrent neural networks. In 2018 Zhang et al. and Zhou et al. respectively proposed methods to interpret convolutional neural networks [14, 15, 16]. Zeiler et al. instead used the approach of visualization of the activation of the layer with the Deconvnets algorithm to understand convolutional neural networks [17]. Bau et al. proposed the Network Dissection framework which quantifies well the interpretability of latent representations of convolutional neural networks [18]. In the mean time, the attacks and defenses from adversarial examples to neural networks were frequently studied [19].

Program analysis, on the other hand, is a research area that focuses on using mathematical theory and formal methods to automatically analyze the behavior of a computer program regarding properties such as safety, correctness, robustness and liveness [20]. In particular, abstract interpretation theory originally introduced in area of program analysis by Cousot and Cousot is a framework of approximating the semantics of a computer program based on order theory, especially lattices [21, 22]. A step further beyond program analysis, in recent years, probabilistic program analysis has been introduced to analyze the quantitative aspects of computer programs, as well as semantics of probabilistic programming languages [25]. Probabilistic abstract interpretation is a framework that allows to systematically lift any classical analysis or verification method to the probabilistic setting by separating in the program semantics the probabilistic behavior from the (non-)deterministic behavior [23]. Di Pierro and Wiklicky also developed the framework of probabilistic abstract interpretation by using Concurrent Constraint Programming as a reference programming paradigm in probabilistic program analysis [24], and by investigating relations between the operational semantics of probabilistic programming languages and Discrete Time Markov Chains (DTMCs) to introduce Linear Operator semantics(LOS) [26], and used the framework to compare analysis on the basis of their expected exactness for a given program with a quantitative notion of precision [27], and also extended the framework from finite state spaces to infinite state spaces [28]. Furthermore, the framework was also taken to analyze the worst and average case of abstract interpretation [29]. The probabilistic abstract interpretation can also be a technique for probabilistic data flow analysis [31, 32, 30, 33].

Researchers have explored the study of verification and analysis of neural networks by using techniques of abstract interpretation. In 2018 Gehr et al. proposed an analyzer AI^2 that uses abstract interpretation to over-approximate a range of concrete values into an abstract domain and checks whether this abstract domain satisfies the expected result and which will prove every single input over-approximated by this abstract domain also satisfies this result [34]. This analyzer makes use of the Zonotope abstract domain which was originally proposed by Ghorbal et al. [39]. Later, Singh et al. [35] proposed a new abstract domain which combines floating point polyhedra with intervals, instead of the former Zonotope abstract domain. Its performance was compared with Reluplex, which is a former property verification technique that extends the simplex algorithm to support ReLU constraints [36]. Singh et al. also presented DeepZ based on abstract interpretation that handles

ReLU, Tanh and Sigmoid activation functions and supports feedforward, convolutional, and residual architectures [37]. Abstract interpretation is also bridged with gradient-based optimization and can be applied to training neural networks [38]. Ryou et al. presented Prover, a scalable and precise verifier for recurrent neural networks using polyhedral abstractions [40].

Beyond classic verifiers, the question of whether certain neural network properties hold in probabilistic conditions is also considered. In 2020 Steffen et al. introduced NetDice which is a scalable and accurate probabilistic network configuration analyzer that is based on an inference algorithm to efficiently explore the space of failure scenarios [41]. Mangal et al. verifies probabilistic robustness of neural networks with respect to the input distribution [42]. Pasareanu et al. investigated the use of symbolic analysis and constraint solution space quantification to precisely quantify probabilistic properties in neural networks [43]. In 2024 Zhang and Wiklicky proposed a method of grids approximation to analyze neural networks with techniques from probabilistic abstract interpretation [44].

Chapter 4

Probabilistic Abstract Interpretation on Neural Network

In this chapter, we discuss how to apply the theory of probabilistic abstract interpretation on neural networks. In principle, any neural network is always a non-linear function which can be simulated by a computer program, thus any analysis framework working on computer programs can work on neural networks. We start illustration from applying the theory on a general feed-forward neural network layer, then approach typical multilayer perceptrons(MLP), convolutional neural networks, recurrent neural networks.

4.1 Probabilistic Abstract Interpretation on General Layer

In this section, we show how to apply the theory of probabilistic abstract interpretation on a general neural network layer. Recall that a general layer can be expressed as

$$x^{t+1} = ReLU(W^t x^t + b^t) \quad (4.1)$$

To construct a probabilistic abstract interpretation, the affine function f on concrete values representing layer propagation, the function $\vec{f}$ in probability space, and the abstract transformer $f^\#$ on abstract elements are considered. In this case $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is just the forward propagation function shown above. We lift $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ up to be $\vec{f} : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^n)$ by vector space lifting which is

$$\vec{X}^{t+1} = \vec{ReLU}(\vec{W}^t \vec{X}^t + \vec{b}^t) \quad (4.2)$$

where $\vec{X}$ is a vector of probabilities assigned to each possible values of x in order, that is, $\vec{X} = (p_{c_1}, p_{c_2}, \dots, p_{c_n})$ where $x \in [c_1, c_2, \dots, c_n]$, $c_1 < c_2 < \dots < c_n$. Then from $\vec{f}$ we can construct the abstract transformer $f^\#$ which is

$$X^{t+1\#} = ReLU^\#(W^{t\#} X^{t\#} + b^{t\#}) \quad (4.3)$$

where $X^\#$ is a vector of probabilities with respect to $\vec{X}$ in abstract domain. An abstract transformer of a composition of functions is a composition of abstract transformers of each function, that is,

$$\text{if } f = f' \circ f'', \text{ then } f^\# = f'^\# \circ f''^\# \quad (4.4)$$

In neural networks, it is very common for a layer to be a composition of multiple functions. A typical case will be a linear function followed by a non-linear activation function. Then, to obtain the abstract transformer, recall the definition of Moore-Penrose pseudo-inverse

$$f^\# = A' f G \quad (4.5)$$

and we can have abstract transformer for W and $ReLU$

$$W^\# = A' \vec{W} G, ReLU^\# = A'' \vec{ReLU} G' \quad (4.6)$$

we have

$$(W \circ ReLU)^\# = W^\# \circ ReLU^\# \quad (4.7)$$

In probabilistic abstract interpretation, each possible value that a neuron has is with a corresponding probability. A prior distribution initialization is needed to be given in the input space. We use a tensor product to combine all probability distributions for every single neuron in the input space to form an overall distribution representing the input layer. If two neurons are independent, it holds that

$$\vec{X_1^1 \times X_2^1} = \vec{X_1^1} \otimes \vec{X_2^1} \quad (4.8)$$

Based on probability initialization, in neural network, the abstract transformer $f^\#$ for the first layer is therefore

$$f^\#(X^{1\#}) = ReLU^\#(W^\#(X_1^{1\#} \otimes \dots \otimes X_n^{1\#}) + b^\#) \quad (4.9)$$

given $x_1^1, \dots, x_n^1$ all independent. Note that tensor product initialization is used only under assumption that neurons are independent and only for input layer. This cannot be applied to any other layers in the neural network since in hidden layers neurons are not independent.

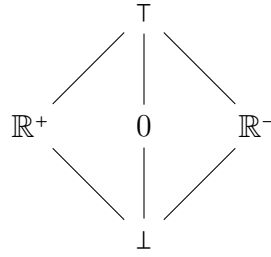
4.2 Probabilistic Abstract Interpretation on Basic MLP

Given a simple multilayer perceptron(MLP) with two layers and five neurons. Let the inputs be all reals, $x_1^1, x_2^1 \in \mathbb{R}$.

4.2.1 Integer Space with Special Weights

We start from naive parameters such that

$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (4.10)$$

**Figure 4.1:** lattice of abstract domain

and we limit the boundary of input neurons in integer space $x_1^1, x_2^1 \in \mathbb{N}$ such that

$$x_1^1, x_2^1 \in [-3, 3] \quad (4.11)$$

and again assume ReLU is the activation function. In the framework of probabilistic abstract interpretation, we need to choose the appropriate abstract domain to obtain the target analysis. In this case if we consider the property of positive or negative sign of neurons as well as the final output, we can take the abstract domain to be $\{\mathbb{R}^-, 0, \mathbb{R}^+\}$, where each of the three categories is an abstract element. In the concrete domain, we have an affine function on concrete values $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

$$(x_1^2, x_2^2) = f(x_1^1, x_2^1) = \text{ReLU}(W(x_1^1, x_2^1) + b) \quad (4.12)$$

and in the probabilistic domain we have $\vec{f} : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$

$$\vec{X}^2 = \vec{f}(\vec{X}^1) = \overrightarrow{\text{ReLU}}(\overrightarrow{W}(\vec{X}_1^1 \otimes \vec{X}_2^1)) \quad (4.13)$$

To discuss abstract domain, let us initialize the input space with uniform distribution meaning x_1^1, x_2^1 is assigned with each of these integer values with equal probability, then it is obvious that

$$\vec{X}_1^1 = \vec{X}_2^1 = \left(\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}\right) \quad (4.14)$$

$$X_1^{1\#} = X_2^{1\#} = \left(\frac{3}{7}, \frac{1}{7}, \frac{3}{7}\right) \quad (4.15)$$

The abstract elements $\{\mathbb{R}^-, 0, \mathbb{R}^+\}$ are with corresponding probabilities $\frac{3}{7}$, $\frac{1}{7}$ and $\frac{3}{7}$, respectively. Therefore, the abstract transformer mapping on abstract elements can be deduced as

$$\begin{aligned} X^{2\#} &= \text{ReLU}^\#(W^\#(X_1^{1\#} \otimes X_2^{1\#})) \\ &= \left(0 \quad 0 \quad 0 \quad 0 \quad \frac{4}{7} \quad 0 \quad 0 \quad 0 \quad \frac{3}{7}\right) \end{aligned} \quad (4.16)$$

This means that the final output of this MLP given this input space will be classified as 0 with probability $\frac{4}{7}$ and will be categorized as $\mathbb{R}^+$ with probability $\frac{3}{7}$. This gives additional information compared to the classical framework where the classical framework is not able to tell the likeliness of signs of output. The reason is that the approximation in the classical framework has lost information of absolute values of variables but only kept signs of them.

We can also find

$$G = \begin{pmatrix} \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} \end{pmatrix} \quad (4.19)$$

which maps from abstract domain into probabilistic domain (there are 9 possible combinations of abstract elements in abstract domain and 49 possible combinations of pairs of concrete values). Note that when we map back from abstract domain to concrete domain we have lost previous information, so we are unable to obtain

original values. One approach we can take is by uniform distribution. For example, there are 9 possible pairs of (x_1^1, x_2^1) with both values negative, so we let the total probability of $(\mathbb{R}^-, \mathbb{R}^-)$ divided by nine be the probability of each possible pair. This is the reason that in the first column the entries are $\frac{1}{9}$. The entries in the second column are $\frac{1}{3}$ because there are 3 possible pairs of $(\mathbb{R}^-, 0)$.

Finally, by formula of abstract transformer, we can multiply the above matrices together to obtain $W^\#$ such that

$$W^\# = A' \overrightarrow{W} G = \begin{pmatrix} 1 & 1 & \frac{1}{3} & 1 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 1 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 1 & \frac{1}{3} & 1 & 1 \end{pmatrix} \quad (4.20)$$

The function that follows is $ReLU$. Formally we can also find $\overrightarrow{ReLU}$ and the Moore-Penrose pseudo-inverse pairs A'' and G'' in a similar way, and hence

$$ReLU^\# = A'' \overrightarrow{ReLU} G' \quad (4.21)$$

Specifically, we have

$$\overrightarrow{ReLU} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.22)$$

and

$$G' = \begin{pmatrix} \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \end{pmatrix} \quad (4.23)$$

and

$$A'' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad (4.24)$$

which turns out that

$$ReLU^\# = A'' \overrightarrow{ReLU} G' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.25)$$

However, we can also easily obtain a general $ReLU^\#$ by tensor product such that

$$\begin{aligned} ReLU^\# &= \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \quad (4.26)$$

Note that these two $ReLU^\#$ are different. However, if we put them into computations in abstract domain, they in fact give the same performance. To compute $X^{2\#}$

by the first $ReLU^\#$, we have

$$\begin{aligned}
 X^{2\#} &= ReLU^\#(W^\#((\frac{3}{7}, \frac{1}{7}, \frac{3}{7}) \otimes (\frac{3}{7}, \frac{1}{7}, \frac{3}{7}))) \\
 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \left(\begin{pmatrix} 1 & 1 & \frac{1}{3} & 1 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 1 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 1 & \frac{1}{3} & 1 & 1 \end{pmatrix} \left(\begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 7 \end{pmatrix} \otimes \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 7 \end{pmatrix} \right) \right) \\
 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{3}{7} \\ 0 \\ 0 \\ 0 \\ \frac{1}{7} \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{3}{7} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{4}{7} \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{3}{7} \end{pmatrix}
 \end{aligned} \tag{4.27}$$

and if we compute it by the second $ReLU^\#$

$$X^{2\#} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{3}{7} \\ 0 \\ 0 \\ 0 \\ \frac{1}{7} \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{3}{7} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{4}{7} \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{3}{7} \end{pmatrix} \tag{4.28}$$

we actually get the same result. We can say that the second $ReLU^\#$ is a general abstract transformer, while the first $ReLU^\#$ is the specialized abstract transformer in this case.

4.2.2 Integer Space with General Weights

Now if we reconsider the above example, we realize the dimension number 13 in $\vec{W}$ and A' is in fact caused by specificity of weights W - some different input variable pairs are mapped to the same output value. That is, the matrices $\vec{W}$ and A' are

‘folded’. An intuition gives that $\vec{W}$ should be in dimension of 49×49 in general case, if no output value duplicates. Let us consider an example with parameters such that

$$W = \begin{pmatrix} 1 & 2 \\ 5 & 8 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (4.29)$$

If we combine with the previous example to consider, $\vec{W}$ is in fact a re-permutation matrix which re-orders entries of vectors corresponding to their values after affine transformation. Specifically, $\vec{W}$ is a 49×49 matrix such that

[illegible]

where each column represents a possible pair of (x_1^1, x_2^1) and each row represents a corresponding outcome of (x_1^2, x_2^2) . All pairs are ordered from small to large, first by comparing x_1 and then x_2 . The pair $(-3, -2)$ was in the 2nd position in order, and was mapped to $(-7, -31)$ by W which becomes 3rd position in the new order, therefore the entry of 2nd column 3rd row is 1.

Then A' is a 49×9 matrix that maps pairs in the new order to abstract elements such that

[illegible]

where the first 23 pairs are all in $(\mathbb{R}^-, \mathbb{R}^-)$ and the last 23 pairs are all in $(\mathbb{R}^+, \mathbb{R}^+)$. The three pairs in the middle are $(0, -2)$, $(0, 0)$, $(0, 2)$ respectively. The matrix G in this case is not affected by the affine transformation, therefore G is the same as it in last section. Multiplying these together we get

$$W^\# = A' \vec{W} G = \begin{pmatrix} 1 & 1 & \frac{1}{9} & 1 & 0 & 0 & \frac{7}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{7}{9} & 0 & 0 & 1 & \frac{1}{9} & 1 & 1 \end{pmatrix} \quad (4.32)$$

and we can compute $X^{2\#}$ directly using the general abstract transformer for $ReLU$ that

$$\begin{aligned} X^{2\#} &= ReLU^\#(W^\#((\frac{3}{7}, \frac{1}{7}, \frac{3}{7}) \otimes (\frac{3}{7}, \frac{1}{7}, \frac{3}{7}))) \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \left(\begin{pmatrix} 1 & 1 & \frac{1}{9} & 1 & 0 & 0 & \frac{7}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{7}{9} & 0 & 0 & 1 & \frac{1}{9} & 1 & 1 \end{pmatrix} \left(\begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \\ 7 \end{pmatrix} \otimes \begin{pmatrix} 3 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \\ 7 \end{pmatrix} \right) \right) \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{23}{49} \\ 0 \\ 0 \\ 0 \\ \frac{1}{49} \\ \frac{1}{49} \\ \frac{1}{49} \\ 0 \\ \frac{23}{49} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{25}{49} \\ \frac{1}{49} \\ 0 \\ 0 \\ \frac{23}{49} \end{pmatrix} \end{aligned} \quad (4.33)$$

This means that there is $\frac{25}{49}$ chances that variables end up in $(0, 0)$, $\frac{1}{49}$ chance to have variables end up in $(0, \mathbb{R}^+)$ and $\frac{23}{49}$ chance to have variables end up in $(\mathbb{R}^+, \mathbb{R}^+)$, given a uniform distribution initialization.

4.2.3 Real Space

We further expand this to the more general case. In the previous setting, we limit the input domain to an integer space. Now, if we discuss the domain in the real space,

we have $x_1^1, x_2^1 \in \mathbb{R}$ such that

$$x_1^1, x_2^1 \in [-3, 3] \quad (4.34)$$

and assume parameters of this layer is still

$$W = \begin{pmatrix} 1 & 2 \\ 5 & 8 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (4.35)$$

In real space we have infinite points; by uniform distribution initialization, the probability of single point zero is negligible. Therefore,

$$X_1^{1\#} = X_2^{1\#} = \left(\frac{1}{2}, 0, \frac{1}{2}\right) \quad (4.36)$$

Again, in order to obtain the abstract transformer, we consider $\vec{W}, A', G$. In integer space each variable only has 7 possible values so the input dimension to $\vec{W}$ is $7 \times 7 = 49$. In real space, each variable has infinitely many possible real values, so the input dimension to $\vec{W}$ tends to infinity, so does the output dimension. Assuming the input dimension is M and the output dimension is N , we have

$$\vec{W} = \lim_{M \rightarrow \infty} \lim_{N \rightarrow \infty} \mathbf{M}^{M \times N} \quad (4.37)$$

and $M \geq N$. When some different pairs of input variables are mapped to the same value, then $M > N$, the matrix $\vec{W}$ is ‘folded’, otherwise $M = N$ and $\vec{W}$ is bijective. Intuitively, since different real values can rarely be mapped to the same one and both M and N tend to infinity, the dimension that will be ‘folded’ is negligible, so we can assume $M = N$. Therefore, we know that $\vec{W}$ is a square matrix that re-permutates orders of pairs of variables. Also, we can have dimension of A' such that

$$A' = \lim_{N \rightarrow \infty} \mathbf{M}^{N \times 9} \quad (4.38)$$

where the input dimension of A' is in fact the output dimension of $\vec{W}$ which tends to infinity, and the output dimension of A' is $3 \times 3 = 9$ for each variable is categorized into three abstract elements. Therefore, A' approximates these infinitely many pairs of concrete points into 9 different pairs of abstract elements. In an opposite direction, we have the dimension of G such that

$$G = \lim_{M \rightarrow \infty} \mathbf{M}^{9 \times M} \quad (4.39)$$

which maps nine pairs of abstract elements back to the concrete domain. We can know that G is a matrix that looks like

$$G = \begin{pmatrix} \frac{1}{p} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{p} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{p} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{q} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{p} & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \frac{1}{q} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{q} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{q} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{p} \end{pmatrix} \quad (4.40)$$

where $p = (\frac{\sqrt{M}}{2})^2 = \frac{M}{4}$ and $q = \frac{\sqrt{M}}{2}$. Since $M \rightarrow \infty$, $p \rightarrow \infty$ and $q \rightarrow \infty$. To obtain the abstract transformer, we multiply them together to have

$$W^\# = A' \overrightarrow{W} G$$

$$= \begin{pmatrix} 1 & 1 & \frac{1}{4} & 1 & 0 & 0 & \frac{3}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{3}{4} & 0 & 0 & 1 & \frac{1}{4} & 1 & 1 \end{pmatrix} \quad (4.41)$$

by mathematical deduction, considering $\overrightarrow{W}$, A' and G carefully. If we look at $W^\#$ we can see that, for example, any pairs (x_1^1, x_2^1) that $x_1^1, x_2^1 < 0$ will be mapped to $(\mathbb{R}^-, \mathbb{R}^-)$ by W , so all entries that are greater than zero in the first column of G will be 'hit' by entries 1s in the first row of matrix product of $A'W$, therefore $W^\#(1, 1) = 1$. Only a portion of pairs (x_1^1, x_2^1) that $x_1^1 < 0, x_2^1 > 0$ will be mapped to $(\mathbb{R}^-, \mathbb{R}^-)$ by W , therefore $W^\#(3, 1) = \frac{\int_0^{\frac{3}{4}} \frac{1}{2}x}{9} = \frac{1}{4}$. Also, if we compare $W^\#$ with the previous $W^\#$ in integer space case, we can see that this $W^\#$ is more generalized.

Then consider

$$ReLU^\# = A'' \overrightarrow{ReLU} G' \quad (4.42)$$

We still have the general abstract transformer of $ReLU$ so that

$$\begin{aligned}
 X^{2\#} &= ReLU^\#(W^\#((\frac{1}{2}, 0, \frac{1}{2}) \otimes (\frac{1}{2}, 0, \frac{1}{2}))) \\
 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & \frac{1}{4} & 1 & 0 & 0 & \frac{3}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{3}{4} & 0 & 0 & 1 & \frac{1}{4} & 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix} \\
 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{2} \\ 0 \\ 0 \\ 0 \\ \frac{1}{2} \end{pmatrix}
 \end{aligned} \tag{4.43}$$

in real space.

As long as this result is obtained, there are two further generalizations to this example. Firstly, it is not hard to generalize this if we let $x_1^1, x_2^1 \in \mathbb{R}$ such that

$$x_1^1, x_2^1 \in (-\infty, \infty) \tag{4.44}$$

The same results will still hold if we rewrite $x_1^1, x_2^1 \in [-3, 3]$ as $x_1^1, x_2^1 \in [-a, a]$, $a = 3$, and expand it to $x_1^1, x_2^1 \in [-a, a]$ and let $a \rightarrow \infty$.

Secondly, we can generalize the parameters of this layer to be

$$W = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{4.45}$$

We can still assume permutation matrix $\vec{W}$ is bijective and not ‘folded’. G is still the same given different parameters as it is not affected. $\vec{W}$ and A' will be affected instead, as the entries depend on W and b . Therefore, entries of $W^\#$ will also be affected, with the same size and similar pattern but different entries. Finally, $ReLU^\#$ is still the same as it is not affected. Note that b does nothing else but a switch to entries of A' when an output value belongs to a different abstract element because of b (in fact b does not affect $\vec{W}$ as it does not change the order). So we will not discuss b individually but consider it together with W to make $\vec{W}$.

One direct problem we face in the real space case is that we need to handle computations of matrices with dimension of infinity. We might be able to do this in

mathematics, but we always wish to avoid this in a computer program. Chapter 5 will offer some approaches to solve this problem.

4.2.4 Re-permutation Matrix

We know that the Moore-Penrose pseudo-inverse pairs A and G depend on the abstract domain. Thus, the key point is to find the relationship between the parameters of the layer W , b and the re-permutation matrix $\vec{W}$. Note that b in the affine transformation represents a shift without change in scaling; therefore, it will not affect the re-permutation $\vec{W}$. That is, $\vec{W}$ only depend on W in affine transformation.

To consider it more specifically, an affine transformation rotates the coordinate axis into new angles and consequently transforms the vector space in the same scaling, and embeds it back to the original coordinates to find a resulting re-permutation

of all possible data points. For example, the above $W = \begin{pmatrix} 1 & 2 \\ 5 & 8 \end{pmatrix}$ rotates x_1 -axis to

$x_2 = 5x_1$ and rotates x_2 -axis to $2x_2 = 8x_1$. Consequently the density distribution of all possible data points changes and the permutation of these points also changes.

Recall that the permutation is with respect to original x_1 -coordinates first and then x_2 -coordinates and so on, from small to large. The data point $(-3, -2)$ was ordered prior to the data point $(-2, -3)$, but is ordered after $(-2, -3)$ after this affine transformation. This is because the rotated x_1 -axis has a larger slope than the rotated x_2 -axis, so $(-3, -2)$ after affine transformation will result in a smaller x_1 -coordinate. Seeing these effects will help us better understand the relationship between W and $\vec{W}$.

A matrix $W = \begin{pmatrix} 1 & -2 \\ 5 & 8 \end{pmatrix}$ added by a negative sign, for example, will make the transformation of x_2 -axis to be in opposite direction, which is $-2x_2 = 8x_1$, thus the re-permutation will also be in opposite direction.

Once we have obtained $\vec{W}$ from W , together with pseudo inverse pairs, we can always obtain the abstract transformer $W^\#$, which in fact can also be regarded as a re-permutation matrix of density distribution in abstract domain. However, we are unable to obtain $W^\#$ directly in the abstract domain without first obtaining $\vec{W}$ because the information is not sufficient. The abstract transformer in the previous example in integer space

$$W^\# = \begin{pmatrix} 1 & 1 & \frac{1}{9} & 1 & 0 & 0 & \frac{7}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{7}{9} & 0 & 0 & 1 & \frac{1}{9} & 1 & 1 \end{pmatrix} \quad (4.46)$$

cannot be obtained directly in the abstract domain, as information about entries like $\frac{1}{9}$ and $\frac{7}{9}$ can only be obtained through the concrete domain. In real space, we

may need to take mathematical methods or also geometric methods to obtain the re-permutation matrices as it may not be straightforward. The entries of real space abstract transformer in the same example

$$W^\# = \begin{pmatrix} 1 & 1 & \frac{1}{4} & 1 & 0 & 0 & \frac{3}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{3}{4} & 0 & 0 & 1 & \frac{1}{4} & 1 & 1 \end{pmatrix} \quad (4.47)$$

are not straightforward to be obtained, where analysis by geometric expression is helpful in this case.

4.2.5 Pseudo Inverse Matrix

As long as we have obtained the re-permutation matrices and hence the abstract transformers, we are always able to continue propagating density distributions of data points by doing matrix multiplications for each layer in order, with abstract transformers for both affine transformations and activations. We have already discussed the re-permutation matrix; let us look at the pseudo inverse pairs at each layer.

We know that the pseudo inverse matrices depend on the abstract domain and the density distribution. In the previous example, the input data points are equally distributed in areas belonging to different abstract elements, hence the pseudo inverse matrices A and G have a pattern that the entries oscillate between different abstract elements. More specifically, with respect to the original permutation of the input data points, it goes from $(\mathbb{R}^-, \mathbb{R}^-)$, passes by $(\mathbb{R}^-, 0)$, enters into $(\mathbb{R}^-, \mathbb{R}^+)$, then returns to $(\mathbb{R}^-, \mathbb{R}^-)$ and goes on. Until it passes by $(0, \mathbb{R}^-)$, $(0, 0)$ and $(0, \mathbb{R}^+)$, it starts the same oscillation in area of $(\mathbb{R}^+, \mathbb{R}^-)$, $(\mathbb{R}^+, 0)$ and $(\mathbb{R}^+, \mathbb{R}^+)$. This gives a pattern of matrices A and G . The pattern of matrices A' and G' turns different, because after mapping by $W = \begin{pmatrix} 1 & 2 \\ 5 & 8 \end{pmatrix}$, the data points are mainly distributed into $(\mathbb{R}^-, \mathbb{R}^-)$ and $(\mathbb{R}^+, \mathbb{R}^+)$, except that data points in the middle section are still distributed around areas of other abstract elements.

In real space, the situation is not discrete anymore. In the previous example, we used p and q to represent the probabilities of the entries in G , which are obtained by mathematical analysis of ratios. In fact, when it gets to the real space, the part it passes by areas of abstract elements 0s becomes negligible.

We may also consider different properties of the neural network, resulting in different abstract domains. This will also make the pattern of inverse matrices different, as the areas belonging to abstract elements are different.

4.2.6 Higher Dimension

All the discussions we had in above sections only has two input variable nodes. In a common neural network, there are thousands and even millions or billions of variables. It is necessary to consider the case of higher dimensions.

Consider a neural network layer with $x_1^1, x_2^1, x_3^1 \in \mathbb{R}$ such that

$$x_1^1, x_2^1, x_3^1 \in (-\infty, \infty) \quad (4.48)$$

and parameters

$$W = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad (4.49)$$

and we have

$$X_1^{1\#} = X_2^{1\#} = X_3^{1\#} = \left(\frac{1}{2}, 0, \frac{1}{2}\right) \quad (4.50)$$

Again, we have the dimension of $\vec{W}$ such that

$$\vec{W} = \lim_{M \rightarrow \infty} \lim_{N \rightarrow \infty} \mathbf{M}^{M \times N} \quad (4.51)$$

We also have a dimension of G such that

$$G = \lim_{M \rightarrow \infty} \mathbf{M}^{27 \times M} \quad (4.52)$$

as there are in total $3 \times 3 \times 3 = 27$ abstract elements. We can actually write G as a tensor product

$$G = \begin{pmatrix} \frac{1}{p} & 0 & 0 \\ \vdots & \vdots & \vdots \\ \frac{1}{p} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{p} \\ \vdots & \vdots & \vdots \\ 0 & 0 & \frac{1}{p} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{p} & 0 & 0 \\ \vdots & \vdots & \vdots \\ \frac{1}{p} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{p} \\ \vdots & \vdots & \vdots \\ 0 & 0 & \frac{1}{p} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{p} & 0 & 0 \\ \vdots & \vdots & \vdots \\ \frac{1}{p} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{p} \\ \vdots & \vdots & \vdots \\ 0 & 0 & \frac{1}{p} \end{pmatrix} \quad (4.53)$$

as $p \rightarrow \infty$. The dimension of A' , on the other hand, will be

$$A' = \lim_{N \rightarrow \infty} \mathbf{M}^{N \times 27} \quad (4.54)$$

and entries of A' will depend on W and b . Then, the abstract transformer of the layer will be

$$W^\# = A' \vec{W} G \quad (4.55)$$

which is a 27×27 matrix. Similarly, if we have $x_1^1, \dots, x_n^1 \in \mathbb{R}$, $n > 3$, the abstract transformer will be in dimension of $3^n \times 3^n$.

4.2.7 Different Properties

The examples we discussed above only consider the property of signs of neuron values in a neural network. The property considered determines the abstract domain; therefore, we let the abstract domain be $(\mathbb{R}^-, 0, \mathbb{R}^+)$. In real world applications, we consider a much wider variety of properties.

Robustness One typical property we care about for a neural network is the boundaries of neuron values. In a robustness analysis, we prove that all possible neuron values are bounded by some particular limits to confirm whether the neural network is robust. The framework of probabilistic abstract interpretation can further tell how much ratio of all possible neuron values is bounded by a certain region.

Let $x_1^1, x_2^1 \in \mathbb{R}$ such that

$$x_1^1, x_2^1 \in [-3, 3] \quad (4.56)$$

and parameters

$$W = \begin{pmatrix} \frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (4.57)$$

We consider whether the neuron values are bounded by $[-1, 1]$. The abstract domain can thus be set to $\{\mathcal{B}_{out} = \{x \mid x < -1 \vee x > 1\}, \mathcal{B}_{in} = \{x \mid x \in [-1, -1]\}\}$. By uniform initialization, $\vec{X}_1^1 = \vec{X}_2^1 = (\frac{2}{3}, \frac{1}{3})$. We can obtain the Moore-Penrose pseudo-inverses are matrices look like

$$G = \begin{pmatrix} \frac{1}{p_1} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{p_1} & 0 & 0 & 0 \\ 0 & \frac{1}{p_2} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \frac{1}{p_2} & 0 & 0 \\ \frac{1}{p_1} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{p_1} & 0 & 0 & 0 \\ 0 & \frac{1}{p_2} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad (4.58)$$

where the four columns represent $(\mathcal{B}_{out}, \mathcal{B}_{out}), (\mathcal{B}_{out}, \mathcal{B}_{in}), (\mathcal{B}_{in}, \mathcal{B}_{out}), (\mathcal{B}_{in}, \mathcal{B}_{in})$, respectively. Only the first part of rows of G are listed. Considering permutation matrix $\vec{W}$ and A' , we can know that $A'\vec{W}$ is a matrix looks like

$$A'\vec{W} = \begin{pmatrix} 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ 1 & 1 & \dots & \dots & \dots & 1 & 1 \end{pmatrix} \quad (4.59)$$

and we can have the abstract transformers for the layer by mathematical deduction that

$$\begin{aligned} W^\# &= A' \vec{W} G \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{aligned} \quad (4.60)$$

and therefore in abstract domain we have

$$\begin{aligned} W^\# X^{1\#} &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \left(\begin{pmatrix} \frac{2}{3} \\ \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix} \otimes \begin{pmatrix} \frac{2}{3} \\ \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix} \right) \\ &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{aligned} \quad (4.61)$$

by which we prove that all possible neuron values after this layer are bounded as desired.

Other Properties Chapter 5 will also introduce some other properties of the neural network that we care about, other than robustness. For each different property, it will also introduce a different approximation approach and corresponding abstract domain.

4.3 Probabilistic Abstract Interpretation on Convolutional Neural Network

Recall that in convolutional neural networks a convolutional layer in principle is also a special case of fully connected layers. We can take a typical convolution mask which detects vertical border lines of an image

$$g = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} \quad (4.62)$$

We can still concern the property of signs of neuron values. In computer vision application, the signs of neuron values represent the darkness of pixels. Considering the property of signs, we let the abstract domain be $\{\mathbb{R}^-, \mathbb{R}^+\}$. In the concrete domain, we have $f : \mathbb{R}^{16} \rightarrow \mathbb{R}^9$

$$(y_1, \dots, y_9) = ReLU(W(x_1, \dots, x_{16}) + b) \quad (4.63)$$

where for simplicity, we assume b to be zero vector. Then we have the dimension of $\vec{W}$ in the probabilistic domain

$$\vec{W} = \lim_{M \rightarrow \infty} \lim_{N \rightarrow \infty} \mathbf{M}^{M \times N} \quad (4.64)$$

although both M, N tend to infinity, the affine function maps 16 neurons to 9 neurons, which in principle is still different infinitely large dimension. Also, we have the dimension of G

$$G = \lim_{M \rightarrow \infty} \mathbf{M}^{2^{16} \times M} \quad (4.65)$$

such that G can be written as a 16-fold tensor product

$$G = \bigotimes_{i=1}^{16} \begin{pmatrix} \frac{1}{p} & 0 \\ \vdots & \vdots \\ \frac{1}{p} & 0 \\ 0 & \frac{1}{p} \\ \vdots & \vdots \\ 0 & \frac{1}{p} \end{pmatrix} \quad (4.66)$$

and the dimension of A'

$$A' = \lim_{N \rightarrow \infty} \mathbf{M}^{N \times 2^9} \quad (4.67)$$

Therefore, the dimension of abstract transformer $W^\#$ is

$$W^\# = A' \vec{W} G = \mathbf{M}^{2^{16} \times 2^9} \quad (4.68)$$

and the abstract transformer $ReLU^\#$ in this case will be

$$ReLU^\# = \bigotimes_{i=1}^9 \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \quad (4.69)$$

The abstract transformer of a pooling layer is very similar to convolution, and the dimension of the abstract transformer will be

$$P^\# = \mathbf{M}^{2^9 \times 2^4} \quad (4.70)$$

When applying probabilistic abstract interpretation on Convolutional neural networks we encounter dimensional problem. In a convolutional neural network, we are dealing with images that can typically contain a large number of pixels such as $1920 \times 1080 = 2,073,600$ pixels. In the above example, although the input image of size 4×4 is extremely tiny, in concrete domain we still have to handle matrices of dimension 256^{16} in grayscale, as grayscale value of a pixel ranges from 0 to 255. $256^{16} \approx 3.4 \times 10^{38}$ is a huge number of matrix dimension that a computer cannot handle with. The memory capacity(RAM) of a computer is an immediate ceiling of matrix computations. Typically $100,000 \times 100,000$ matrix computation requires roughly 40 GB of memory, and a consumer PC/Mac normally has 16–64 GB RAM.

Even supercomputers may only be able to handle billions of dimensions, and we cannot always expect a supercomputer to apply the analysis. It is necessary to reduce dimension to be under 100,000.

One approach is to design more dimension-reducing abstract elements. For example, instead of considering the sign of each pixel, we can consider the sign of a group of pixels. To handle image size of 4×4 , we can group each 2×2 neighbor pixels together and concern their property of sign. We can either consider whether all pixel values in each neighbor group are positive or consider which sign most of pixels in each neighbor group have, and then approximate this group of pixels into an abstract element of a single sign. In this way, each abstract element only has a size of 2^4 , reduced from a size of 2^{16} , so that we will handle pseudo inverse pairs and abstract transformers with much lower dimension.

Secondly, we can limit the number of possible values of a pixel. Instead of ranging the pixel value from all integers between 0 to 255, we can approximate these 256 values into limited choices, such as tens of them, as there is no need to distinguish tiny differences in context of brightness. This will further reduce the dimension of matrices.

In Chapter 6 we will show a practical case of a convolutional neural network where it is necessary to reduce the dimension of abstract elements in order to apply the framework of probabilistic abstract interpretation.

4.4 Probabilistic Abstract Interpretation on Recurrent Neural Network

Recall the architecture of recurrent neural networks and consider formulas of three domains of the model. Again we can still consider the property of signs of neuron values. The abstract domain is then $\{\mathbb{R}^-, \mathbb{R}^+\}$. Note that in this case we use the tanh function instead of the ReLU function, plus consideration of features of weights, the abstract element of zero is hence not needed. Since the neuron values are two-dimensional, in the concrete domain we have $f : \mathbb{R}^{2T} \rightarrow \mathbb{R}^{2S}$

$$(h_1^t, h_2^t) = \tanh(W_{hh}(h_1^{t-1}, h_2^{t-1}) + W_{hx}(x_1^t, x_2^t) + b_h), t \leq T \quad (4.71)$$

$$(h_1^t, h_2^t) = \tanh(W_{hh}(h_1^{t-1}, h_2^{t-1}) + W_{hy}(y_1^{t-T}, y_2^{t-T}) + b_h), t \geq T + 1 \quad (4.72)$$

$$(y_1^{t-T}, y_2^{t-T}) = \tanh(W_{yh}(h_1^{t-1}, h_2^{t-1}) + b_y), t \geq T + 1 \quad (4.73)$$

Then by vector space lifting, in the probability space we have $\vec{f} : \mathcal{V}(\mathbb{R}^{2T}) \rightarrow \mathcal{V}(\mathbb{R}^{2S})$

$$\vec{H}^t = \overrightarrow{\tanh}(\overrightarrow{\text{sum}}(\overrightarrow{W_{hh}}(\overrightarrow{H^{t-1}}), \overrightarrow{W_{hx}}(\overrightarrow{X^t}))), t \leq T \quad (4.74)$$

$$\vec{H}^t = \overrightarrow{\tanh}(\overrightarrow{\text{sum}}(\overrightarrow{W_{hh}}(\overrightarrow{H^{t-1}}), \overrightarrow{W_{hy}}(\overrightarrow{Y^{t-T}}))), t \geq T + 1 \quad (4.75)$$

$$\vec{Y^{t-T}} = \overrightarrow{\tanh}(\overrightarrow{W_{yh}}(\overrightarrow{H^{t-1}})), t \geq T + 1 \quad (4.76)$$

We use $\overrightarrow{\text{sum}}$ to denote the function of summation in probabilistic domain. h^0 is a constant, h^t is affected by x^t since $t > 0$, so in the probabilistic domain h^t also has

a probability distribution given the distribution of x^t when $t > 0$. Let us start from $t = 1$. The first function to be considered is $\overrightarrow{W_{hx}}$. Recall that $\overrightarrow{W_{hx}}$ is a permutation matrix and we assume it is not ‘folded’. We should also have pseudo-inverse pairs for $\overrightarrow{W_{hx}}$ such that

$$G^1 = \lim_{M \rightarrow \infty} \mathbf{M}^{4 \times M}, A^2 = \lim_{N \rightarrow \infty} \mathbf{M}^{N \times 4} \quad (4.77)$$

and

$$W_{hx}^\# = A^2 G^1 \quad (4.78)$$

The second function to be considered is $\overrightarrow{sum}$. When $t = 1$, $W_{hh}(h^0)$ and b_h are both constants, and adding with a constant actually performs a re-permutation of $\overrightarrow{X}$. Therefore, when $t = 1$, $\overrightarrow{sum}$ is a permutation matrix with dimension

$$\overrightarrow{sum} = \lim_{N \rightarrow \infty} \mathbf{M}^{N \times N} \quad (4.79)$$

Consequently,

$$sum^\# = A^3 \overrightarrow{sum} G^2 \quad (4.80)$$

will be a 4×4 matrix that adjusts the distribution of x . When $t = 2$, we have a second distribution x^2 taken into consideration, that is, the input distribution in probabilistic domain is in fact

$$\overrightarrow{X} = \overrightarrow{X^1} \otimes \overrightarrow{X^2} \quad (4.81)$$

with corresponding $\overrightarrow{W_{hx}}$ and $\overrightarrow{sum}$ for $\overrightarrow{X^2}$. Therefore, to do this in one shot, we combine all x^t and regard it as one input space. We have

$$\overrightarrow{X} = \overrightarrow{X^1} \otimes \dots \otimes \overrightarrow{X^T} \quad (4.82)$$

we also combine all $\overrightarrow{W_{hx}}$ for each x^t to have the overall $\overrightarrow{W_{hx}}$ with dimension

$$\overrightarrow{W_{hx}} = \lim_{M \rightarrow \infty} \lim_{N \rightarrow \infty} \mathbf{M}^{M^T \times N^T} \quad (4.83)$$

The corresponding overall pseudo-inverse pairs will then be in dimension

$$G = \lim_{M \rightarrow \infty} \mathbf{M}^{4^T \times M^T}, A' = \lim_{N \rightarrow \infty} \mathbf{M}^{N^T \times 4^T} \quad (4.84)$$

hence, W_{hx}^T will be

$$W_{hx}^\# = \mathbf{M}^{4^T \times 4^T} \quad (4.85)$$

similarly $sum^\#$ will also be in dimension

$$sum^\# = \mathbf{M}^{4^T \times 4^T} \quad (4.86)$$

The activation function $\tanh$ is a hyperbolic function that produces values between -1 and +1 without changing the signs of values. Since we analyze signs of neuron values, this activation function will have no effect on the analysis. Therefore,

$$\tanh^\# = I \quad (4.87)$$

These are the first half-part of this prediction model, when $t \leq T$. We can regard the second half of this model as $t > T$, which is the output section. We consider in general $\overrightarrow{W_{hy}}$ and $\overrightarrow{W_{yh}}$ in the same way as in the previous part. Finally, it will output an overall distribution of y , which will be the tensor product of all $y^{(t-T)}$ s.

Let us take a more specific example to illustrate the deduction. Suppose after model training we have parameters of weights

$$W_{hx} = \begin{pmatrix} 1 & 2 \\ 5 & 8 \end{pmatrix}, W_{hh} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, h^0 = b_h = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (4.88)$$

and we initialize the input space with a uniform distribution such that

$$\overrightarrow{X_1^1} = \overrightarrow{X_2^1} = \left(\frac{1}{M}, \dots, \frac{1}{M} \right) \quad (4.89)$$

where $M \rightarrow \infty$ as $x_1^1, x_2^1 \in \mathbb{R}$. In the abstract domain, all possible values of x_1^1, x_2^1 are equally distributed in two abstract categories (the probability of value zero is negligible) and we have

$$X_1^{1\#} = X_2^{1\#} = \left(\frac{1}{2} \quad \frac{1}{2} \right) \quad (4.90)$$

which gives

$$X^{1\#} = X_1^{1\#} \otimes X_2^{1\#} = \left(\frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4} \right) \quad (4.91)$$

Now take the overall distribution of $x^1, \dots, x^T$. In the probabilistic domain, we have

$$\overrightarrow{X} = \left(\frac{1}{M^{2T}}, \dots, \frac{1}{M^{2T}} \right) \quad (4.92)$$

where $M \rightarrow \infty$. In abstract domain, it is instead

$$X^\# = \left(\frac{1}{2^{2T}}, \dots, \frac{1}{2^{2T}} \right) \quad (4.93)$$

The pseudo-inverse function G maps $X^\#$ in the abstract domain to $\overrightarrow{X}$ in probabilistic domain, which is

$$G = \bigotimes_1^{2T} \begin{pmatrix} \frac{2}{M} & 0 \\ \vdots & \vdots \\ \frac{2}{M} & 0 \\ 0 & \frac{2}{M} \\ \vdots & \vdots \\ 0 & \frac{2}{M} \end{pmatrix} \quad (4.94)$$

Note that here each entry is $\frac{2}{M}$ because the sum of each column equals to 1, and there are in total $\frac{M}{2}$ entries with probability $\frac{2}{M}$ in each column. Then because W_{hx} is the same for each x^t , $\overrightarrow{W_{hx}}$ can be written as a tensor product of the same permutation matrix, and A' is in the dimension of

$$A' = \lim_{M \rightarrow \infty} \mathbf{M}^{M^T \times 4^T} \quad (4.95)$$

From the computation in previous section(as for simplicity we take a trick of using matrix of same weights) we can have

$$\begin{aligned}
 W_{hx}^{\#} &= A' \overrightarrow{W_{hx}} G \\
 &= \bigotimes_1^T \begin{pmatrix} 1 & \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 1 \end{pmatrix}
 \end{aligned} \tag{4.96}$$

The following $\overrightarrow{sum}$ function takes the summation of two distributions of $W_{hh}(h^{t-1})$ and $W_{hx}(x^t)$. However, since the distribution of h^t is fully decided by x^t , the summation is fully dependent only on $\overrightarrow{X}$. Therefore, $\overrightarrow{sum}$ is a permutation matrix and is in the dimension of

$$\overrightarrow{sum} = \lim_{M \rightarrow \infty} \mathbf{M}^{M^T \times M^T} \tag{4.97}$$

assuming it is ‘unfolded’ and regarding duplicate values as distinct ones, and

$$sum^{\#} = A'' \overrightarrow{sum} G' = \mathbf{M}^{2^T \times 2^T} \tag{4.98}$$

Then, $\overrightarrow{tanh}$ will squeeze the distribution into an area bounded by $(-1, 1)$, and since in the abstract domain it does not change signs of the neuron values,

$$tanh^{\#} = I \tag{4.99}$$

This will give $H^{\#}$ when $t \leq T$. When $t > T$, the model performs an output section with a deduction similar to W_{hx} and W_{hh} .

In recurrent neural networks, we also encounter the problem of dimension like we do in convolutional models. When there is a large number of nodes in input, the tensor products are taken in order to represent all possibilities in one matrix, so the dimension explodes exponentially. As mentioned in previous section, the memory capacity of a computer is an immediate ceiling of matrix computations. Typically $100,000 \times 100,000$ matrix computation requires roughly 40 GB of memory, and a consumer PC/Mac normally has 16–64 GB RAM. Even supercomputers may only be able to handle billions of dimensions, and we cannot always expect a supercomputer to apply the analysis. It is necessary to reduce dimension to be under 100,000.

A good news is in context of natural language processing, a neuron is typically a letter(or more specifically a corresponding value of a letter in the dictionary), and there is hence only a choice of 26 values of letters. This is a much less number comparing to a pixel value in image processing problems, and makes a much more limited dimension of matrices.

Secondly, like what we do in convolutional models, designing a more dimension-reducing abstract element is also an approach. Instead of approximating one single node into abstract domain, we can group nodes together and consider an abstract element approximation of a group of nodes. This will also greatly reduce the total dimension of the problem.

4.5 Conclusions

In this chapter, we have discussed how to apply the analysis framework of probabilistic abstract interpretation on neural networks, based on formal representation and matrix computation. We have discussed about cases in concrete domain, probabilistic domain and abstract domain. We have discussed about matrix representation of pseudo inverse pairs and repermutation function. We have illustrated using simple neural network examples, both in Integer space and in Real space. We have also discussed different neural network architectures, starting from a general layer, then reaching multilayer perceptrons, convolutional neural networks and recurrent neural networks.

However, limitations still remained for a more general illustration. Firstly, the neural network examples we used are in very simple structures, neither with a large dimension nor with deep layers. With a larger number of nodes and layers, the complexity of matrix representation could increase in exponential way. How to have a general representation in mathematics in a complex neural network structure could be a point to further discuss about. Secondly, the neural network property we concerned in above analysis only involves sign of neuron values. For a more reasonable analysis we will concern more complex and more specific properties. In the next chapter, we will introduce some novel abstract domains that could be taken within the analysis, with a focus on particular properties of neural networks.

Chapter 5

Different Abstract Domains

So far we have discussed applying the framework of probabilistic abstract interpretation on different types of neural network. We can also see that a key step in application of this framework is to define an abstract domain, with respect to a property of this neural network to be analyzed. In this chapter, we will introduce some typical abstract domains that we may choose for different circumstances. We start from an introduction of the Zonotope approximation that is expanded from existing work to our framework, and then will introduce three other novel approximation approaches with their corresponding abstract domains.

5.1 Zonotope Approximation

In 2018, Gehr et al. introduced AI^2 , an analyzer which can automatically prove properties, especially safety and robustness, for convolutional neural networks based on classical abstract interpretation [34]. AI^2 has found a general way to define abstract domain and abstract transformer of different types of layers, including the fully connected layer, the convolutional layer, and the max pooling layer. It defines suitable abstract domains and represents neural network layers as CAT functions, and hence defines abstract transformers which can be applied and computed on abstract elements in a very general way. The formal definitions and details can be found in [34]. As it also works on fully connected layer, it can also be applied on the basic MLP example we discussed in Chapter 4. It will be worth showing here.

Recall that the 2-2-1 fully connected neural network with $x_1^1, x_2^1 \in \mathbb{R}$,

$$x_1^1, x_2^1 \in [-3, 3] \quad (5.1)$$

With abstraction α , this input space can be abstracted into a Box

$$B^1 = \{-3 \leq x_1^1 \leq 3, -3 \leq x_2^1 \leq 3\} \quad (5.2)$$

According to [34], an abstract transformer can generally be defined as

$$\text{for } f(x) = \text{ReLU}(Wx + b), f^\#(X^\#) = \text{ReLU}^\#(\text{Aff}(X^\#, W, b)) \quad (5.3)$$

in this case, $X^\# = B^1$, so we have

$$\begin{aligned}
 f^\#(X^\#) &= f^\#(B^1) = \text{ReLU}^\#(\text{Aff}(B^1, W, b)) \\
 &= \text{ReLU}^\#((x_1^1 + x_2^1, x_1^1 + x_2^1)) \\
 &= \{0 \leq x_1^2 \leq 6, 0 \leq x_1^2 \leq 6\} \\
 &= B^2
 \end{aligned} \tag{5.4}$$

where $\text{ReLU}^\#$, the abstract transformer of ReLU, is also well defined in [34]. Applying concretization function γ , B^2 can be concretized back into a concrete space containing all possible concrete points (x_1^2, x_2^2) after layer propagation so that

$$0 \leq x_1^2 \leq 6, 0 \leq x_1^2 \leq 6 \tag{5.5}$$

As it uses a shape of Zonotope(or Box, or Polyhedra), it can abstract the whole concrete input space into one abstract element and propagate till the end.

However, AI^2 only considers the classical abstract interpretation, which in many cases has its limitations. We can expand the approach further with the possibility to apply probabilistic abstract interpretation.

Same as in the classical case, we can still create an abstract element by a Zonotope which captures all perturbed inputs. However, beyond the classical case, by probabilistic abstract interpretation we can initialize all perturbed inputs with a distribution, e.g. uniform distribution. We can then propagate the distribution as we propagate the abstract element through layers, giving the probability of each area of Zonotope to be possibly fallen at, by calculating the volume of area and calculating its ratio to the whole Zonotope of abstract element. Let us take a two-dimensional example in [34] to illustrate. Given function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

$$f(x) = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} x \tag{5.6}$$

with input space X expressed by the Zonotope $z^1 : [-1, 1]^3 \rightarrow \mathbb{R}^2$:

$$z^1(\epsilon_1, \epsilon_2, \epsilon_3) = (1 + 0.5\epsilon_1 + 0.5\epsilon_2, 2 + 0.5\epsilon_1 + 0.5\epsilon_3) \tag{5.7}$$

After affine transformation, we get Zonotope z^2 abstracting output space Y . Let the property be $C = \{(y_1, y_2) \in \mathbb{R}^2 \mid y_1 \geq -2\}$. We can see that clearly property C holds on z^2 . However, suppose that we are concerned with property $C' = \{(y_1, y_2) \in \mathbb{R}^2 \mid y_1 \geq -1\}$, then we find that C' does not hold on z^2 . Beyond that, classical abstract interpretation is unable to provide further useful information. But by a probabilistic abstract interpretation, we find that the volume of area which satisfies C' is

$$V_{z^2, C'} = \int_{y \in C'} y \, dy = 5 \tag{5.8}$$

where the volume of the whole Zonotope z^2 is

$$V_{z^2} = \int_{y=f(x)} y \, dy = 6 \tag{5.9}$$

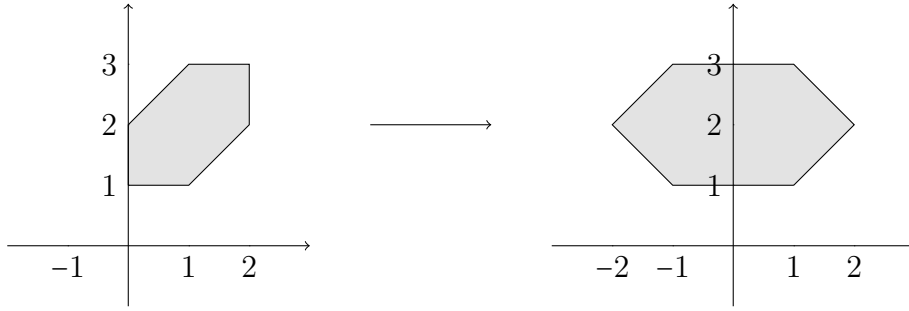


Figure 5.1: Abstract elements through layer

So we have a ratio $V_{z^2, C'}/V_{z^2} = \frac{5}{6}$, that is, $f(x)$ has the probability $\frac{5}{6}$ to satisfy property C' . Note that a precondition for this calculation of volume ratio is that there are no spurious points in z^2 , that is, for every point y in z^2 , we can always find a point x in z^1 such that $f(x) = y$. Otherwise, we need to eliminate spurious points in z^2 before calculating volume. This is a difference from the classical abstract interpretation that in the classical case we only concern one direction from z^1 to z^2 , but in the probabilistic case we need to concern both. Also note that until the current step all points in z^2 are still uniformly distributed given a uniformly distributed input space. However, the situation will change in the next step after ReLU is applied.

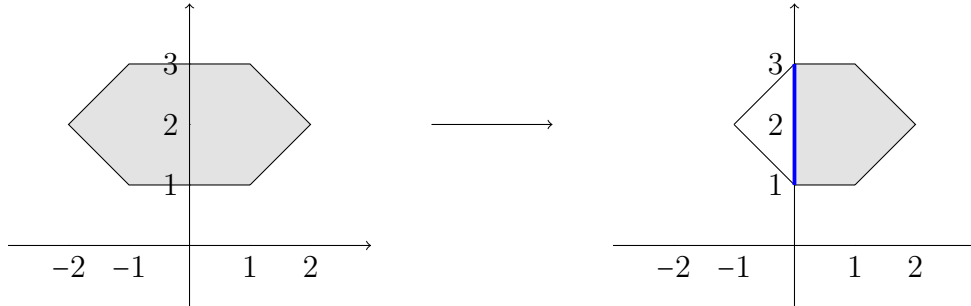


Figure 5.2: ReLU transformation

Firstly we can see in z^3 that the area for $y_1 < 0$ (triangle uncolored) is spurious. Hence, Zonotope is a sound but not precise abstract element in this case. Secondly, because of ReLU, points are heavily distributed in the line $(0,1) - (0,3)$, shown in blue line. So, the points in z^3 are no longer uniformly distributed. In fact, half of the points in z^2 (all points for $x_1 < 0$) will fall at the blue line after applying ReLU. Therefore, if we are concerned about the property $C'' = \{(y_1, y_2) \in \mathbb{R}^2 \mid y_1 \geq 1\}$ in z^3 , then the probability ratio to satisfy C'' will be

$$V_{z^3, C''}/V_{z^3} = \frac{\int_{y \in C''} y \, dy}{\int_{y=f(x)} y \, dy} = \frac{\int_{y \in C''} y \, dy}{\int_{y_1 > 0} y \, dy + V_{blue}} = \frac{1}{3+3} = \frac{1}{6} \quad (5.10)$$

This is a brief idea of how the Zonotope approximation can be introduced into the probabilistic case. Now let us show a more detailed deduction in our framework with respect to this abstract domain.

5.1.1 Zonotope Approximation in Integer Space

Similarly to the classical case, we need to define abstract transformer for the probabilistic framework, $f^\#$, together with Moore-Penrose pseudo inverse functions A and G .

Following the example in the last subsection, we first consider integer space. In the concrete domain, we have 4 input points in general $X = \{(0, 1), (1, 1), (1, 3), (2, 2)\}$ and we lift them into the probability space with a uniform distribution to get $\vec{X} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$. In the abstract domain, the integer space contained in z^1 is $I^2 = \{(0, 1), (0, 2), (1, 1), (1, 2), (1, 3), (2, 2), (2, 3)\}$. As we abstract the input probability space into an abstract domain, we have

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (5.11)$$

hence

$$X^\# = A(\vec{X}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ 0 \\ \frac{1}{4} \\ 0 \\ \frac{1}{4} \\ \frac{1}{4} \\ 0 \end{pmatrix} \quad (5.12)$$

and for opposite direction, we have

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (5.13)$$

hence

$$G(X^\#) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{4} \\ 0 \\ \frac{1}{4} \\ 0 \\ \frac{1}{4} \\ \frac{1}{4} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} = \vec{X} \quad (5.14)$$

After the linear transformation, we see $f(X) = \{(-1, 1), (-1, 3), (1, 1), (2, 2)\}$ and the integer space contained in z^2 is $I^2 = \{(-2, 2), (-1, 1), (-1, 2), (-1, 3), (0, 1), (0, 2), (0, 3),$

$(1, 1), (1, 2), (1, 3), (2, 2)\}$. Hence, we can get

$$A' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.15)$$

and

$$G' = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.16)$$

we can also see that during the linear transformation the distribution over the input space does not change (each point in X corresponds to a point in $f(X)$ and vice versa). Therefore, given a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

$$f(x) = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} x \quad (5.17)$$

we have $\vec{f} : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$

$$\vec{f} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.18)$$

as the concrete space $X = \{(0, 1), (1, 1), (1, 3), (2, 2)\}$ is mapped into $f(X) = \{(-1, 1), (-1, 3), (1, 1), (2, 2)\}$ which is re-ordered from $\{(-1, 1), (1, 1), (-1, 3), (2, 2)\}$, where

positions of the 2nd and 3rd pairs are re-ordered. Therefore, $f^\# : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$

$$\begin{aligned}
 f^\# = A' \vec{f} G &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \\
 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.19}
 \end{aligned}$$

To verify with, it is also straightforward to find the general abstract transformer $\hat{f}^\#$ from the graph z^1 to z^2 such that

$$\hat{f}^\# = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.20}$$

we can see that $f^\#$ is clearly a specialized version of $\hat{f}^\#$. Similarly, we can find our abstract transformer for the *ReLU* function. Firstly, we again find Moore-Penrose

pseudo-inverse functions for z^3

$$A'' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.21)$$

and

$$G'' = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.22)$$

and $\overrightarrow{ReLU} : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$

$$\overrightarrow{ReLU} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.23)$$

as in this case $ReLU$ maps $f(X) = \{(-1, 1), (-1, 3), (1, 1), (2, 2)\}$ into $ReLU(f(X)) = \{(0, 1), (0, 3), (1, 1), (2, 2)\}$ without any re-permutation. Therefore, we have $ReLU^\# : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$

$$\begin{aligned} ReLU^\# = A'' \overrightarrow{ReLU} G' &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \quad (5.24)$$

Also, comparing this with the general abstract transformer for $Re\hat{L}U^\#$ from the graph z^2 to z^3 that

$$Re\hat{L}U^\# = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (5.25)$$

we can see that $ReLU^\#$ is a specialized case of $Re\hat{L}U^\#$.

5.1.2 Zonotope Approximation in Real Space

To make this example more general we expand the Integer space into Real space. Let us first look at the linear transformation from z^1 to z^2 . Suppose in the concrete domain we have N different points in arbitrary positions $X = \{x_1, x_2, \dots, x_N\}$, and X is over-approximated by z^1 . If we lift X up into the probability space with uniform distribution, we have $\vec{X} = \{\frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N}\}$. In abstract domain, we know that in mathematics there are infinitely many points in real space contained in z^1 , but we can regard this as a very large number and let it be $\lim_{M \rightarrow \infty} M$. Then, we can have

$$A = \lim_{M \rightarrow \infty} \mathbf{M}^{N \times M} \quad (5.26)$$

with an entry 1 in each column in the corresponding position and the other entries are all zeros. We can also have

$$G = \lim_{M \rightarrow \infty} \mathbf{M}^{M \times N} \quad (5.27)$$

which performs opposite mapping and forms a Moore-Penrose pseudo inverse of A . Now we can let the same very large number $\lim_{M \rightarrow \infty} M$ be the number of all real points contained in z^2 , so

$$A' = A, G' = G \quad (5.28)$$

As it is a linear transformation, again we have $\vec{f} : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$ which is a permutation matrix. Hence $f^\# : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$

$$f^\# = A' \vec{f} G \quad (5.29)$$

In the ReLU transformation, $\overrightarrow{ReLU} : \mathcal{V}(\mathbb{R}^2) \rightarrow \mathcal{V}(\mathbb{R}^2)$ is a permutation matrix, and we have

$$A'' = A', G'' = G' \quad (5.30)$$

Note that $\overrightarrow{ReLU}$ is very commonly a ‘folded’ matrix, that is, suppose if $\exists$ and only $\exists$ a pair $x_p, x_q \in ReLU(f(X))(p < q)$ s.t. $x_{p2} = x_{q2}$, then the matrix representation of

$\overrightarrow{ReLU}$ will look like

$$\overrightarrow{ReLU} = \mathbf{M}^{N \times (N-1)} = \begin{pmatrix} 1 & 0 & . & . & . & . & . & . & . & . \\ 0 & 1 & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & 1 & . & . & 1 & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & 1 & 0 \\ . & . & . & . & . & . & . & . & 0 & 1 \end{pmatrix} \quad (5.31)$$

which can be obtained by eliminating the q th row from the identity matrix and assigning $\overrightarrow{ReLU}_{pq} = 1$. That is,

$$\overrightarrow{ReLU} \begin{pmatrix} \frac{1}{N} \\ \frac{1}{N} \\ \frac{1}{N} \\ . \\ . \\ . \\ . \\ . \\ . \\ \frac{1}{N} \end{pmatrix} = \begin{pmatrix} \frac{1}{N} \\ \frac{1}{N} \\ \frac{1}{N} \\ . \\ \frac{2}{N} \\ . \\ . \\ . \\ . \\ \frac{1}{N} \end{pmatrix} \quad (5.32)$$

with $\frac{2}{N}$ in the p th entry. Then

$$A'' = \lim_{M \rightarrow \infty} \mathbf{M}^{(N-1) \times M} \quad (5.33)$$

with an entry 1 in each column in the corresponding position and the other entries are all zeros, and

$$G'' = \lim_{M \rightarrow \infty} \mathbf{M}^{M \times (N-1)} \quad (5.34)$$

Therefore,

$$ReLU^\# = A'' \overrightarrow{ReLU} G' \quad (5.35)$$

and if $\exists r$ exclusive pairs $x_p, x_q \in ReLU(f(X))$ s.t. $x_{p2} = x_{q2}$, then

$$\overrightarrow{ReLU} = \mathbf{M}^{N \times (N-r)} \quad (5.36)$$

Now let's conclude this Zonotope Approximation by giving a concise formal definition of it.

Definition 5 Given the concrete domain $X \in \mathcal{P}(\mathbb{R}^m)$ and its corresponding probabilistic domain $\vec{X} \in \mathcal{V}(X) \subseteq \mathcal{V}(\mathbb{R}^m)$, a **Zonotope approximation** abstract domain is a set $X^\# \in \mathcal{V}(Z) \subseteq \mathcal{V}(\mathbb{R}^m)$ such that Z is the smallest Zonotope satisfies $X \subseteq Z$. The mappings A and G form a pseudo inverse pair such that $A : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$

$$A(\vec{X}) = \sum_{c \in X} p_c c + \sum_{c \in Z, c \notin X} 0 \cdot c \quad (5.37)$$

and $G : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$

$$G(X^\#) = \sum_{c \in X} p_c c \quad (5.38)$$

We can see that in this approach we have constructed an appropriate Zonotope-shaped approximation as [34] did in classic abstract interpretation. However, in our framework, this Zonotope-shaped approximation does not make much contribution to analysis by only assigning additional point entries with zero probability. In next section we will discuss other approximation approaches that fit more to the framework of probabilistic abstract interpretation.

5.2 Grids Approximation

We see that in real space we encounter the problem of infinity - since there are infinitely many points in any closed area in Real space, the dimensions of both input space and abstract element tend to be infinitely large, and hence we are forced to handle vectors and matrices in dimension of infinity. We always wish to avoid this situation. A possible approximation to avoid this is to use grids as abstract elements. In the former example, instead of considering space $\mathbb{R}^2$, we can consider $\epsilon\mathbb{Z}^2$, where ϵ gives the size of the grids to be chosen. Say, if we let ϵ to be 0.01, then the grid size is 0.01×0.01 , which means that we will have 100×100 points in a closed space bounded by a Box of size 1×1 (regardless of points on boundary lines - we could have 101×101 points more specifically). In this way, each point represents the local probability of the neighbor area, which still shows the probability density distribution of the space. For convenience we always let x_i - *axis* be one of the lines of grids, which will determine the position of the whole grid. Now, if we consider z^1 in $\epsilon\mathbb{Z}^2$, we have in total 30301 points in this abstract element. That is, recall the notation M used in the last subsection, instead of having $\lim_{M \rightarrow \infty} M$ points, we now have $M = 30301$. Same as the input space X in $\epsilon\mathbb{Z}^2$. No matter what X is, N is finite and computable. Therefore, we have $A : \mathcal{V}(0.01\mathbb{Z}^2) \rightarrow \mathcal{V}(0.01\mathbb{Z}^2)$

$$A = \mathbf{M}^{N \times M} = \mathbf{M}^{N \times 30301} \quad (5.39)$$

and so on for G . We know that any linear transformation in $\mathbb{R}^2$ is a bijective mapping and $\epsilon\mathbb{Z}^2 \subset \mathbb{R}^2$, therefore, any linear transformation in $\epsilon\mathbb{Z}^2$ is also a bijective mapping. So we can have $\vec{f} : \mathcal{V}(\epsilon\mathbb{Z}^2) \rightarrow \mathcal{V}(\epsilon\mathbb{Z}^2)$

$$\vec{f} = \mathbf{M}^{N \times N} \quad (5.40)$$

Similarly z^2 in $\mathcal{V}(\epsilon\mathbb{Z}^2)$ contains 60401 points, so

$$A' = \mathbf{M}^{N \times M} = \mathbf{M}^{N \times 60401} \quad (5.41)$$

and so on for G' . Then we have $f^\# : \mathcal{V}(\epsilon\mathbb{Z}^2) \rightarrow \mathcal{V}(\epsilon\mathbb{Z}^2)$

$$f^\# = A' \vec{f} G \quad (5.42)$$

Similarly for $ReLU^\#$ that we can now replace $\lim_{M \rightarrow \infty} M$ by finite and computable numbers. The reason AI^2 takes Zonotope as the abstract domain is that it is concerning the robustness of a neural network, and having a sound yet precise abstract element which over-approximates the input space is a good way to prove robustness. In probabilistic abstract interpretation, we have a different objective. Probabilistic abstract interpretation concerns more about density distribution of input space and its change through propagation of a neural network. Therefore, it is more meaningful to use other abstract domains to achieve the purpose. Following grid approximation, using a more sparse grid as an abstract domain is a fair choice. In this context, let z^1 in $0.01\mathbb{Z}^2$ be an input space, that is, $X = \{x \in z^1 \mid x \in 0.01\mathbb{Z}^2\}$. Then we let the abstract domain be the space in $\mathbb{Z}^2$. We can choose different shapes of grids. As an example, we choose the squares. That is, we abstract 10,000 points in each 1×1 square in $0.01\mathbb{Z}^2$ into a single point in the center of the square, and the value of probability of which is the sum of all values of probabilities of 10,000 points. Say, if we consider the coordinates $x \in (-3, 3)$ and $y \in (-3, 3)$, then

$$A = \mathbf{M}^{490,000 \times 49}, G = \mathbf{M}^{49 \times 490,000} \quad (5.43)$$

and A', G' are similar. A concrete element is a vector of length 490,000 and an abstract element is a vector of length 49, with each entry being the value of probabilities. We can see that an advantage of grid approximation is that it saves computation complexity. $\vec{f}$ is very computationally consuming as 490,000 affine transformation calculations are needed, while $f^\#$ only makes 49 such calculations. To lift f to $\vec{f}$, we perform an affine transformation f , and the result point will be the entry position of $\vec{f}$. Now let us define the grids approximation formally.

Definition 6 Given the concrete domain $X \in \mathcal{P}(\mathbb{R}^m)$ and its corresponding probabilistic domain $\vec{X} \in \mathcal{V}(X) \subseteq \mathcal{V}(\mathbb{R}^m)$, a **grids approximation** abstract domain is $X^\# \in \mathcal{V}(\mathbb{R}^m)$ such that each grid $\langle p_n, c_n^\# \rangle \in X^\#$ takes abstraction of all concrete points in it. The mappings A and G form a pseudo inverse pair such that $A : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$

$$A(\vec{X}) = A\left(\sum_{c \in X} p_c c\right) = \sum_n \left(\sum_{c \in g_n} p_c\right) c_n^\# \quad (5.44)$$

where g_n is the n -th grid and $c_n^\#$ is the vector of centroid of g_n , and $G : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$

$$G(X^\#) = G\left(\sum_n p_n c_n^\#\right) = \sum_n \sum_{c' \in g_n} p_{c'} c' \quad (5.45)$$

.

5.2.1 Square Grids

A m -dimensional square grid g with size s can be defined by intervals in each dimension

$$g = [a_1, a_1 + s] \times [a_2, a_2 + s] \times \dots \times [a_m, a_m + s] \quad (5.46)$$

To take an approximation of the data points in a concrete space, any concrete point $\mathbf{x} = (x_1, \dots, x_m)$ that is an interior point of the grid g such that

$$a_i \leq x_i < a_i + s, \text{ for all } x_i \quad (5.47)$$

will be approximated to this grid as an abstract element. We say $\mathbf{x} \in g$. That is, we have Moore-penrose pseudo-inverse functions A and G such that

$$A\left(\sum_{x \in g} p_x x\right) = \left(\sum_{x \in g} p_x\right) c_{(g)}^\# = \left(\sum_{x \in g} p_x\right) \left(a_1 + \frac{s}{2}, a_2 + \frac{s}{2}, \dots, a_n + \frac{s}{2}\right) \quad (5.48)$$

for any interior point $\mathbf{x}$ of the grid g and $c_{(g)}^\#$ is the vector of the abstract element in abstract domain, where we denote the value of $c_{(g)}^\#$ as the centroid coordinates of g . Here we accumulate corresponding probabilities for all interior points of g to be the corresponding probability of $c_{(g)}^\#$ such that

$$p_g = \sum_{\mathbf{x} \in g} p_{\mathbf{x}} \quad (5.49)$$

Inversely, we have

$$G(p_g c_{(g)}^\#) = \sum_{x' \in g} p_{x'} x' \quad (5.50)$$

Therefore, instead of handling infinite points in a concrete space, we can directly perform the computation on approximated grids in the abstract domain, and the abstract transformer is straightforward $f^\# : C^\# \rightarrow D^\#$

$$\begin{aligned} f^\#(< p_g, c_{(g)}^\# >) &= f(< p_g, (a_1 + \frac{s}{2}, a_2 + \frac{s}{2}, \dots, a_n + \frac{s}{2}) >) \\ &= < p_g, f(a_1 + \frac{s}{2}, a_2 + \frac{s}{2}, \dots, a_n + \frac{s}{2}) > \end{aligned} \quad (5.51)$$

which is the concrete function f mapping on the centroid of the grid. During the abstract transformation, the corresponding probability inherits.

5.2.2 Sparser Grids

After taking a grids approximation of concrete space, sometimes it still costs too many computations to perform transformation in abstract domain. In such a case, we can take an even further approximation of current grids approximation, using sparser grids. A m -dimensional sparser grid g' with size S is

$$g' = [a'_1, a'_1 + S] \times [a'_2, a'_2 + S] \times \dots \times [a'_m, a'_m + S] \quad (5.52)$$

where a smaller grid $g = [a_1, a_1 + s] \times \dots \times [a_m, a_m + s]$ with size s is an interior grid of g' such that

$$a'_i \leq a_i, a_i + s < a'_i + S \text{ for all } a_i, a'_i \quad (5.53)$$

We say $g \in g'$. We also have a Moore-penrose pseudo-inverse function A' and G' such that

$$A'\left(\sum_{g_n \in g'} p_n c_{(g_n)}^\#\right) = \left(\sum_{g_n \in g'} p_n\right) c_{(g')}^\# = \left(\sum_{g_n \in g'} p_n\right) \left(a'_1 + \frac{S}{2}, \dots, a'_n + \frac{S}{2}\right) \quad (5.54)$$

We still accumulate the corresponding probabilities of all interior grids of g' to be the corresponding probability of $c_{(g')}^\#$ such that

$$p_{g'} = \sum_{g_n \in g'} p_n \quad (5.55)$$

and inversely

$$G(p_{g'} c_{(g')}^\#) = \sum_{g_n \in g'} p_{(g_n)} c_{(g_n)}^\# \quad (5.56)$$

In the second abstract domain we have abstract transformer $f'^\# : C'^\# \rightarrow D'^\#$

$$f'^\#(< p_{g'}, c_{(g')}^\# >) = f(< p_{g'}, (a'_1 + \frac{S}{2}, \dots, a'_n + \frac{S}{2}) >) \quad (5.57)$$

.

5.2.3 Other Shapes

In addition to square grids, we can also have grids of different shapes, such as diamonds. Different shapes are suitable for different problem circumstances and will lead to different accumulation of data points.

For example, in a diamond approximation, a n -dimensional diamond $\mathcal{D}$ with center $\mathbf{a}$ and size s can be defined in the form of

$$\mathcal{D}(\mathbf{a}, s) = \{\mathbf{x} \mid \sum_{i=1}^n |x_i - a_i| \leq s\} \quad (5.58)$$

where any concrete point $\mathbf{x} \in \mathcal{D}(\mathbf{a}, s)$ will be approximated to this diamond as an abstract element.

We can also have a rectangle approximation with a n -dimensional hyperrectangle $\mathcal{R}$ such that

$$\mathcal{R}(\mathbf{s}) = [a_1, a_1 + s_1] \times [a_2, a_2 + s_2] \times \dots \times [a_n, a_n + s_n] \quad (5.59)$$

where $\mathbf{s} = (s_1, \dots, s_n)$ is the size of $\mathcal{R}$.

5.3 Distribution Approximation

Another practical choice of abstract domain in probabilistic abstract interpretation is the distribution approximation. The distribution approximation approximates the concrete space in its distribution which includes all concrete data plus the corresponding probabilities. An abstract transformer in this case maps a distribution following propagations of neural network layers. In other words, we consider a distribution that propagates through the neural network in the abstract domain.

Definition 7 Given the concrete domain $X \in \mathcal{P}(\mathbb{R}^m)$ and its corresponding probabilistic domain $\vec{X} \in \mathcal{V}(X) \subseteq \mathcal{V}(\mathbb{R}^m)$, a **distribution approximation** abstract domain is an approximation function $y_{p,X}$ that best matches $\vec{X}$, i.e. $\langle p, X \rangle$. The mappings A and G form a pseudo inverse pair that $A : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$, $A(\vec{X}) = y_{p,X}$ s.t. $y_{p,X}(c) = p_c$ for all $c \in X$, where $\vec{X} = \sum_{c \in X} p_c c$, and $G : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$, $G(X^\#) = G(y_{p,X}) = \sum_{c \in X'} p'_c c$ s.t. $X' \sim y_{p,X}$.

In distribution approximation, different choices of function approximators can be selected with respect to problem circumstance. Polynomials, radial basis functions, or Fourier transform can be one of typical choices.

5.3.1 Polynomial Regression

Polynomials is one of the simplest approximators in such a regression problem. A general polynomial model is in the form of

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{x} + \beta_2 \mathbf{x}^2 + \dots + \beta_m \mathbf{x}^m + \epsilon \quad (5.60)$$

Or it can be written as a system of linear equations:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \quad (5.61)$$

which can also be written in matrix notation of

$$\vec{y} = \mathbf{X} \vec{\beta} + \vec{\epsilon} \quad (5.62)$$

and we can find the least square solution

$$\hat{\vec{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y} \quad (5.63)$$

We can now have the polynomials that approximate the concrete space. Recall the definition of Probabilistic Abstract Interpretation, f in concrete space is an affine function representing the forward propagation of the neural network layer, and therefore we can have the abstract transformer $f^\#$ in abstract domain mapping a distribution to a distribution such that

$$f^\#(\mathbf{y}(\mathbf{x})) = \mathbf{y}(f^{-1}(\mathbf{x})) \quad (5.64)$$

5.3.2 Radial Basis Functions

Radial Basis Functions(RBFs) are real-valued functions whose value depends only on the distance from a central point, typically expressed as

$$\phi(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{c}\|) \quad (5.65)$$

where $\mathbf{x}$ is the input vector, $\mathbf{c}$ is the center, and $\|\cdot\|$ denotes a norm(often Euclidean). Common RBFs include the Gaussian function $\phi(\mathbf{x}, \mathbf{c}) = e^{-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}}$ where σ is a parameter that controls the width of the basis function, multiquadric function $\phi(\mathbf{x}, \mathbf{c}) = \sqrt{\|\mathbf{x} - \mathbf{c}\|^2 + \sigma^2}$, inverse multiquadric function $\phi(\mathbf{x}, \mathbf{c}) = \frac{1}{\sqrt{\|\mathbf{x} - \mathbf{c}\|^2 + \sigma^2}}$, or thin plate spline function $\phi(\mathbf{x}, \mathbf{c}) = \|\mathbf{x} - \mathbf{c}\|^2 \ln(\|\mathbf{x} - \mathbf{c}\|)$.

More specifically, radial basis functions approximate the concrete space in a distribution with N centers. That is, if let original concrete space be $\mathbf{x}$, and let the dimension of probabilities be y , the approximation can be represented in the form of

$$\mathbf{y} = \sum_{i=1}^N a_i \phi(\|\mathbf{x} - \mathbf{c}_i\|) \quad (5.66)$$

where a_i are the interpolation coefficients of the radial basis functions. Note that a_i can be computed once given a particular choice of N centers. To find the optimal choice of centers, we want the loss function

$$RMS = \sqrt{\frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (y_{model_i} - y_{data_i})^2} \quad (5.67)$$

to be minimized [48].

Once we have computed the radial basis functions that approximate the concrete space, we can have the abstract transformer $f^\#$ in the abstract domain mapping a distribution to a distribution such that

$$f^\#(\mathbf{y}(\mathbf{x})) = \mathbf{y}(f^{-1}(\mathbf{x})) \quad (5.68)$$

Example Consider a one-dimensional example containing 13 concrete data points being one-dimensional values $\{0.1, 0.4, 0.5, 0.8, 1.5, 2.1, 3.0, 3.1, 3.5, 4.6, 5.9, 6.0, 6.4\}$, and together with their probabilities $\{0.072, 0.076, 0.08, 0.073, 0.036, 0.014, 0.02, 0.012, 0.016, 0.02, 0.022, 0.03, 0.024\}$ respectively. We say that we wish to approximate these data points by radial basis functions with three centers. That is, we approximate these data by

$$y = \sum_{i=1}^3 a_i \phi(\|\mathbf{x} - \mathbf{c}_i\|) \quad (5.69)$$

Also, let the radial basis functions be Gaussian functions

$$\phi(\|\mathbf{x} - \mathbf{c}\|) = e^{-\|\mathbf{x} - \mathbf{c}\|^2} \quad (5.70)$$

We first have to decide which three centers to choose among the 13 data points. In an exhaustive search, models must be constructed for every possible combination of centers. In each case, we need to solve the system of linear equations

$$\sum_{i=1}^N a_i \phi(\|\mathbf{x}_k - \mathbf{c}_i\|) = y_k \quad (5.71)$$

That is, in this case

$$\begin{pmatrix} \phi(\|\mathbf{x}_1 - \mathbf{c}_1\|) & \phi(\|\mathbf{x}_2 - \mathbf{c}_1\|) & \phi(\|\mathbf{x}_3 - \mathbf{c}_1\|) \\ \phi(\|\mathbf{x}_1 - \mathbf{c}_2\|) & \phi(\|\mathbf{x}_2 - \mathbf{c}_2\|) & \phi(\|\mathbf{x}_3 - \mathbf{c}_2\|) \\ \phi(\|\mathbf{x}_1 - \mathbf{c}_3\|) & \phi(\|\mathbf{x}_2 - \mathbf{c}_3\|) & \phi(\|\mathbf{x}_3 - \mathbf{c}_3\|) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad (5.72)$$

to get interpolation coefficients a_i . Then the cost function are evaluated over the full dataset, and the optimal model is selected with the smallest value of the cost function. The total number of combinations in this case is

$$\binom{13}{3} = \frac{13!}{3!(13-3)!} = 286 \quad (5.73)$$

Therefore, we exhaustively search for 286 choices of combinations of centers and for each choice we compute loss function RMS which has been introduced above. We finally find that the smallest value of cost function is achieved when the selected combination of centers are $(0.5, 0.08)$, $(3.0, 0.02)$, $(6.0, 0.03)$, and the corresponding RBFs is

$$y = 0.8 * e^{-(x-0.5)^2} + 0.2 * e^{-(x-3)^2} + 0.3 * e^{-(x-6)^2} \quad (5.74)$$

Now say we have a very simple neural network layer with only one input node and one output node such that

$$f(x) = Wx + b = 2x + 1 \quad (5.75)$$

Recall the framework of probabilistic abstract interpretation, in abstract domain, we have

$$f^\#(X^\#) = f^\#(A(X)) = f^\#(y(X)) = y(f^{-1}(X)) \quad (5.76)$$

by formula of abstract transformer of the distribution approximation given above. Then we can see that

$$y(f^{-1}(X)) = 0.8 * e^{-(\frac{x-1}{2}-0.5)^2} + 0.2 * e^{-(\frac{x-1}{2}-3)^2} + 0.3 * e^{-(\frac{x-1}{2}-6)^2} \quad (5.77)$$

which is the distribution approximation after propagation of the layer.

Using radial basis functions as an abstract domain in probabilistic abstract interpretation has several advantages. Firstly, radial basis functions is a powerful approximator which can well approximate any continuous function given appropriate centers and parameters. Secondly, radial basis functions can capture the behavior of the distribution with precise localized effects, which clearly shows the local features of the distribution. Also, radial basis functions are a smooth approximation and can fit well for composite affine functions.

5.3.3 Fourier Transform

The Fourier transform is another practical choice of an approximator for distribution approximation. It is a mathematical operation that transforms a function of time (or space) into a function of frequency. A Fourier transform $F(\omega)$ of its original function $f(t)$ is normally expressed as [49]

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \quad (5.78)$$

Or we can write it as the inverse Fourier transform

$$f(t) = \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega \quad (5.79)$$

For a probability distribution, the Fourier transform provides a characteristic function which encodes the distribution's properties in the domain of frequency. That is, it captures and grabs information from all different frequencies contained in this original distribution. This is the same purpose for which the abstraction operation

in Probabilistic Abstract Interpretation framework is seeking.

Note that since the Fourier transform captures features of frequency of a continuous function, if the concrete space is not in such continuous form, it is necessary to take abstraction of concrete space into a continuous distribution first, for example by the radial basis function approximation, and then followed by the Fourier transform approximation. That is, a two-step abstraction is taken in this framework of Probabilistic Abstract Interpretation, and only the second abstract domain is considered.

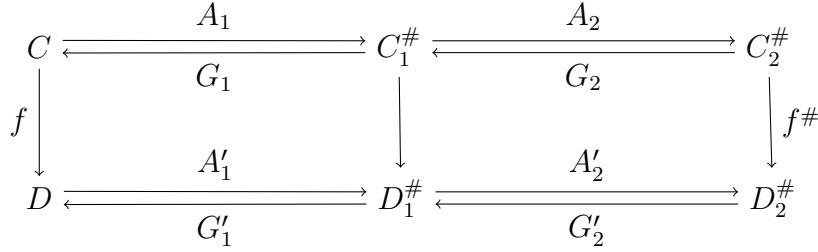


Figure 5.3: Probabilistic Abstract Interpretation with Two-Step Abstraction

Once we have obtained the Fourier transform that captures features of the distribution that approximate the concrete space, we can have the abstract transformer $f^\#$ in the abstract domain of frequency with respect to the propagation of the neural network layer such that

$$f^\#(F(\omega)) = F(g(\omega)) \quad (5.80)$$

where $g(\omega)$ is a function of the frequency change deduced from the affine function f .

Example Take the same example illustrated in radial basis functions approximation section that the distribution approximating concrete space is given by

$$y = 0.8 * e^{-(x-0.5)^2} + 0.2 * e^{-(x-3)^2} + 0.3 * e^{-(x-6)^2} \quad (5.81)$$

and the one-input node neural network layer is

$$f(x) = Wx + b = 2x + 1 \quad (5.82)$$

We first take the Fourier transform of function $y(x)$ such that

$$\begin{aligned} F(\omega) &= \int_{-\infty}^{\infty} y(x) e^{-j\omega x} dx \\ &= \int_{-\infty}^{\infty} (0.8 * e^{-(x-0.5)^2} + 0.2 * e^{-(x-3)^2} + 0.3 * e^{-(x-6)^2}) e^{-j\omega x} dx \\ &= \int_{-\infty}^{\infty} 0.8 * e^{-(x-0.5)^2} e^{-j\omega x} dx + \int_{-\infty}^{\infty} 0.2 * e^{-(x-3)^2} e^{-j\omega x} dx + \int_{-\infty}^{\infty} 0.3 * e^{-(x-6)^2} e^{-j\omega x} dx \end{aligned} \quad (5.83)$$

let $t_1 = x - 0.5$, $t_2 = x - 3$, $t_3 = x - 6$,

$$F(\omega) = \int_{-\infty}^{\infty} 0.8 * e^{-t_1^2} e^{-j\omega(t_1+0.5)} dt_1 + \int_{-\infty}^{\infty} 0.2 * e^{-t_2^2} e^{-j\omega(t_2+3)} dt_2 + \int_{-\infty}^{\infty} 0.3 * e^{-t_3^2} e^{-j\omega(t_3+6)} dt_3 \quad (5.84)$$

Consider only the first term for convenience, and the other two are similar

$$\begin{aligned}
F_1(\omega) &= \int_{-\infty}^{\infty} 0.8 * e^{-t_1^2} e^{-j\omega(t_1+0.5)} dt_1 \\
&= 0.8 * \int_{-\infty}^{\infty} e^{-t_1^2} \cos(\omega(t_1+0.5)) dt_1 + 0.8 * j \underbrace{\int_{-\infty}^{\infty} e^{-t_1^2} \sin(\omega(t_1+0.5)) dt_1}_0 \\
&= 0.8 * \int_{-\infty}^{\infty} e^{-t_1^2} \cos(\omega(t_1+0.5)) dt_1 \\
&= 0.8\sqrt{\pi} e^{-\omega^2/4}
\end{aligned} \tag{5.85}$$

where detailed deduction can be found in [50]. We can see that the Fourier transform of a Gaussian distribution is still a Gaussian distribution. That is, this abstraction does not capture additional information, therefore in this case the Fourier transform may not be an appropriate approximator.

Using the Fourier transform as a distribution approximator in probabilistic abstract interpretation is especially powerful for capturing features of frequency including amplitude, phase and harmonics which are three key components. It is also a universal approximator for the distribution function as any function can be regarded as a periodic function by assuming that the period is infinitely long. However, it is still more appropriate for approximating periodic functions. In addition, it is normally used as a second-step abstraction, given that the concrete space has already been approximated to a continuous function in the first-step abstraction.

5.4 Clusters Approximation

Clusters approximation is a choice of abstract domains where all possible concrete input data points are clustered into a number of groups, and let only the centroid of each group be the abstract element representing each group of concrete data points. We can also say that grids approximation mentioned previously is a special kind of clusters approximations.

Definition 8 Given the concrete domain $X \in \mathcal{P}(\mathbb{R}^m)$ and its corresponding probabilistic domain $\vec{X} \in \mathcal{V}(X) \subseteq \mathcal{V}(\mathbb{R}^m)$, a **clusters approximation** abstract domain is $X^\# \in \mathcal{V}(\mathbb{R}^m)$ such that each cluster $\langle p_n, c_n^\# \rangle \in X^\#$ takes abstraction of all concrete points classified into it. The mappings A and G form a pseudo inverse pair such that $A : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$

$$A(\vec{X}) = A\left(\sum_{c \in X} p_c c\right) = \sum_n \left(\sum_{c \in c^{(n)}} w_{c,c^{(n)}} p_c\right) c_n^\# \tag{5.86}$$

where $c^{(n)}$ is the n -th cluster, $c_n^\#$ is the vector of centroid of $c^{(n)}$, $w_{c,c^{(n)}}$ is the weight of classification, and $G : \mathcal{V}(\mathbb{R}^m) \rightarrow \mathcal{V}(\mathbb{R}^m)$

$$G(X^\#) = G\left(\sum_n p_n c_n^\#\right) = \sum_n \sum_{c' \in c^{(n)}} p_{c'} c' \tag{5.87}$$

Clustering is a typical unsupervised machine learning problem, and there are some commonly used methods that can be also used in our framework.

5.4.1 K-means

K-means clustering is an unsupervised machine learning algorithm used for partitioning a dataset into a pre-defined number of groups. It operates by categorizing data points into clusters by using a mathematical distance measure, usually euclidean, from the cluster center. The objective is to minimize the sum of distances between data points and their assigned clusters. Data points that are nearest to a centroid are grouped together within the same category.

To train and obtain a K-means clustering model, the first step is to initialize K centroids, where K is equal to the number of clusters chosen for a specific dataset. Then it is computed by a two-step iterative process based on the expectation-maximization machine learning algorithm. The expectation step assigns each data point to its closest centroid according to distance. The maximization step computes the mean of all the points for each cluster and reassigns the cluster center, or centroid. This process repeats until the centroid positions have reached convergence or the maximum number of iterations has been reached.

Given the dataset $x^{(1)}, \dots, x^{(m)}$ and the labels of data points $c^{(1)}, \dots, c^{(m)}$, the algorithm is as follows:

1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ at random.
2. Repeat until convergence:

$$\text{For every } i, \text{ set } c^{(i)} := \arg \min_k \|x^{(i)} - \mu_k\|^2. \quad (5.88)$$

$$\text{For each } k, \text{ set } \mu_k := \frac{\sum_{i=1}^m 1\{c^{(i)} == k\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} == k\}}. \quad (5.89)$$

Eventually, the iteration converges and we get K centroids of all K clusters. Recall the framework of Probabilistic Abstract Interpretation, these K clusters $c_1^\#, \dots, c_K^\# \in C^\#$ form the abstract domain $C^\#$. That is, there is an abstraction function A such that

$$A(\{x^{(i)} \mid c^{(i)} == k\}) = c_k^\# \quad (5.90)$$

for each $k = 1, \dots, K$. Meanwhile, if given the probabilities that the data points $x^{(1)}, \dots, x^{(m)}$ are $p_{x^{(1)}}, \dots, p_{x^{(m)}}$, we have

$$p_{c_k^\#} = \sum_i p_{x^{(i)}} \{c^{(i)} == k\} \quad (5.91)$$

and the abstract transformer performing on an abstract element is in fact the concrete function performing on the cluster centroid $f^\# : C^\# \rightarrow D^\#$

$$f^\#(c_k^\#) = f(\mu_k) \quad (5.92)$$

for each $k = 1, \dots, K$.

Example Let's take the previously illustrated example. Given function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

$$f(x) = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} x \quad (5.93)$$

with input space X expressed by the Zonotope $z^1 : [-1, 1]^3 \rightarrow \mathbb{R}^2$

$$z^1(\epsilon_1, \epsilon_2, \epsilon_3) = (1 + 0.5\epsilon_1 + 0.5\epsilon_2, 2 + 0.5\epsilon_1 + 0.5\epsilon_3) \quad (5.94)$$

The geometric expression of the input space can be found in Figure 6.1. In this input space, there are infinitely many data points equally distributed. Let us choose number of clusters in this case to be two, and first randomly initialize two centroids of clusters, say $(0, 1)$ and $(2, 3)$. Then the iteration step follows until it converges to find two centroids of clusters. To simplify the process, let us say after a number of iterations two centroids eventually converge to $(\frac{11}{18}, \frac{29}{18})$ and $(\frac{25}{18}, \frac{43}{18})$. That is, these

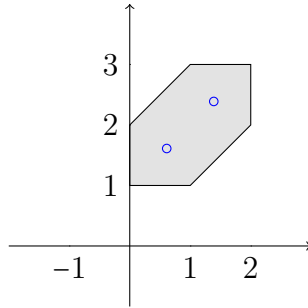


Figure 5.4: Two centroids of clusters

two centroids as abstract elements take an approximation of the input space by

$$A(\{x^{(i)} \mid c^{(i)} == 1\}) = c_1^\# = (\frac{11}{18}, \frac{29}{18}), \quad A(\{x^{(i)} \mid c^{(i)} == 2\}) = c_2^\# = (\frac{25}{18}, \frac{43}{18}) \quad (5.95)$$

To compute layer propagation in the abstract domain, we have the abstract transformer performing on two centroids such that $f^\# : C^\# \rightarrow D^\#$

$$f^\#(c_1^\#) = f(\mu_1) = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{11}{18} \\ \frac{29}{18} \end{pmatrix} = (-\frac{7}{18}, \frac{29}{18}) \quad (5.96)$$

$$f^\#(c_2^\#) = f(\mu_2) = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{25}{18} \\ \frac{43}{18} \end{pmatrix} = (\frac{7}{18}, \frac{43}{18}) \quad (5.97)$$

The corresponding probabilities of two centroids depend on the probabilities with

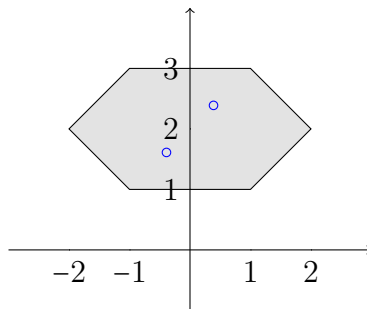


Figure 5.5: Two centroids of clusters after layer propagation

which the input space is distributed. If the probability dimension is initialized with a uniform distribution, then two converged centroids straightforwardly will have

$$p_{c_1^\#} = \sum_i p_{x^{(i)}} \{c^{(i)} == 1\} = \frac{1}{2}, \quad p_{c_2^\#} = \sum_i p_{x^{(i)}} \{c^{(i)} == 2\} = \frac{1}{2} \quad (5.98)$$

Note that after layer propagation the resulting coordinates of two centroids are not equal to intuitive centroids of resulting shapes geometrically. That is because the affine transformation changes the distances between the points and therefore changes the density of the shape, which is not straightforward in infinitely distributed case. The two centroids as abstract elements in clustering approximation in this case have better stored information from the previous layer.

5.4.2 Gaussian Mixture Model

The Gaussian mixture model (GMM) is another commonly used clustering method in unsupervised machine learning. A Gaussian mixture model is a parametric probability density function represented as a weighted sum of Gaussian component densities. Unlike K-means which is a ‘hard’ clustering method which means that it will associate each point to one and only one cluster, the Gaussian mixture model is a soft clustering method used to determine the probabilities each data point belongs to several possible clusters. Similarly to K-means, parameters of a Gaussian mixture model are also estimated using iterative Expectation-Maximization algorithm from training data.

More specifically, a Gaussian mixture is a function that is composed of K Gaussians, identified by $k \in 1, \dots, K$. Each Gaussian is comprised of a mean μ_k that defines its center, a covariance Σ_k that defines its width, and a mixture weight w_k that defines how big or small the Gaussian function will be. That is, a Gaussian mixture model is a weighted sum of K Gaussian densities as given by the equation [51]

$$p(\mathbf{x} | \lambda_k) = \sum_{k=1}^K w_k \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k) \quad (5.99)$$

where $\lambda_k = \{w_k, \mu_k, \Sigma_k\}$, $k = 1, \dots, K$ is the notation of all parameters and each Gaussian density is a Gaussian function of the form

$$\mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k)} \quad (5.100)$$

where D is the number of dimensions of each data point. The mixture weights must satisfy the constraint such that

$$\sum_{k=1}^K w_k = 1 \quad (5.101)$$

Given the dataset $x^{(1)}, \dots, x^{(m)}$, the iterative expectation-maximization algorithm is as follows:

1. E-step: for each data point $x^{(i)}$, compute r_{ik} the probability that it belongs to cluster k

$$r_{ik} = \frac{w_k \mathcal{N}(x^{(i)} | \mu_k, \Sigma_k)}{\sum_{k'} w_{k'} \mathcal{N}(x^{(i)} | \mu_{k'}, \Sigma_{k'})} \quad (5.102)$$

2. M-step: for each cluster k , update its parameters using the r_{ik} computed from previous step

$$m_k = \sum_i r_{ik}, \quad w_k = \frac{m_k}{m} \quad (5.103)$$

$$\mu_k = \frac{1}{m_k} \sum_i r_{ik} x^{(i)}, \quad \Sigma_k = \frac{1}{m_k} \sum_i r_{ik} (x^{(i)} - \mu_k)^T (x^{(i)} - \mu_k) \quad (5.104)$$

Eventually, the iteration converges and the convergence is guaranteed. Again, in the framework of Probabilistic Abstract Interpretation, these K Gaussians $c_1^\#, \dots, c_K^\# \in C^\#$ form the abstract domain $C^\#$, with

$$c_k^\# = \langle \mu_k, \Sigma_k, w_k \rangle \quad (5.105)$$

If given the probabilities of the data points $x^{(1)}, \dots, x^{(m)}$ are $p_{x^{(1)}}, \dots, p_{x^{(m)}}$, we have

$$p_{c_k^\#} = \sum_i r_{ik} p_{x^{(i)}} \quad (5.106)$$

and the abstract transformer is

$$\begin{aligned} f^\#(c_k^\#) &= \langle f^\#(\mu_k), f^\#(\Sigma_k), w_k \rangle \\ &= \langle A\mu_k + b, A\Sigma_k A^T, w_k \rangle \end{aligned} \quad (5.107)$$

given that the neural network layer propagation is $f(\mathbf{x}) = A\mathbf{x} + b$.

Example Again taking the function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

$$f(x) = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} x \quad (5.108)$$

with input space X expressed by the Zonotope $z^1 : [-1, 1]^3 \rightarrow \mathbb{R}^2$

$$z^1(\epsilon_1, \epsilon_2, \epsilon_3) = (1 + 0.5\epsilon_1 + 0.5\epsilon_2, 2 + 0.5\epsilon_1 + 0.5\epsilon_3) \quad (5.109)$$

Assume that the input space again is uniformly distributed and we choose to have two Gaussians ($K = 2$) in our Gaussian mixture model approximation. To simplify the iteration process, we assume that the expectation-maximization step converges at Gaussians $\mathcal{N}_1((\frac{11}{18}, \frac{29}{18}), (\frac{1}{81}, \frac{1}{81}))$ and $\mathcal{N}_2((\frac{25}{18}, \frac{43}{18}), (\frac{1}{81}, \frac{1}{81}))$ with $w_1 = \frac{1}{2}, w_2 = \frac{1}{2}$. That is, in this Gaussian mixture model approximation there are two abstract elements that form the abstract domain

$$c_1^\# = \langle (\frac{11}{18}, \frac{29}{18}), (\frac{1}{81}, \frac{1}{81}), \frac{1}{2} \rangle, c_2^\# = \langle (\frac{25}{18}, \frac{43}{18}), (\frac{1}{81}, \frac{1}{81}), \frac{1}{2} \rangle \quad (5.110)$$

And we have the abstract transformer performing on two Gaussians to compute layer propagation in abstract domain such that $f^\# : C^\# \rightarrow D^\#$

$$\begin{aligned} f^\#(c_1^\#) &= \langle \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{11}{18} \\ \frac{29}{18} \end{pmatrix}, \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{81} \\ \frac{1}{81} \end{pmatrix} \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix}^T, \frac{1}{2} \rangle \\ &= \langle (-\frac{7}{18}, \frac{29}{18}), (\frac{1}{81}, \frac{1}{81}), \frac{1}{2} \rangle \end{aligned} \quad (5.111)$$

$$\begin{aligned}
f^\#(c_2^\#) &= \left\langle \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{25}{18} \\ \frac{43}{18} \end{pmatrix}, \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{81} \\ \frac{1}{81} \end{pmatrix} \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix}^T, \frac{1}{2} \right\rangle \\
&= \left\langle \left(\frac{7}{18}, \frac{43}{18} \right), \left(\frac{1}{81}, \frac{1}{81} \right), \frac{1}{2} \right\rangle
\end{aligned} \tag{5.112}$$

The corresponding probabilities of two abstract elements given that the probabilities of data points are $p_{x(1)}, p_{x(2)} \dots$ are

$$p_{c_1^\#} = \sum_i r_{i1} p_{x(i)}, \quad p_{c_2^\#} = \sum_i r_{i2} p_{x(i)} \tag{5.113}$$

where r_{i1}, r_{i2} are computed from the expectation-maximization iteration process in the convergence state. Obviously, both tend to $1/2$ in this problem setting. These two corresponding probabilities are also corresponding probabilities of $f^\#(c_1^\#)$ and $f^\#(c_2^\#)$ respectively such that

$$p_{c_1^\#} = p_{f^\#(c_1^\#)}, \quad p_{c_2^\#} = p_{f^\#(c_2^\#)} \tag{5.114}$$

as corresponding probabilities of abstract elements inherit during propagation by abstract transformer.

5.5 Conclusions

In this chapter, we have introduced some typical abstract domains that we can choose to analyze particular properties of a neural network. We started from Zonotope approximation that was expanded from existing related works about classical abstract interpretation verification framework by [34]. We then introduced three novel approximation approaches that were designed particularly for our probabilistic framework, including grids approximation, distribution approximation and clusters approximation. We gave concise formal definitions for each of these approximation approaches, the pseudo inverse mappings and discussed how the corresponding abstract transformers would perform. In each approach, we introduced specific approximators for different circumstances. In grids approximation we introduced grids with different shapes and sparsity. In distribution approximation we introduced different function approximators from polynomial regression, radial basis functions to Fourier transform. In clusters approximation we introduced clustering methods including K-means and Gaussian mixture model. These offer a wide variety of choices for diversified analysis purposes.

However, it is always needed to explore more innovative and flexible approximation approaches. It is also worth work to discuss about suitable approximation approaches for problem scenarios in reality such as visual or audio related problems.

Chapter 6

Experiments

In this chapter, we will show some applications of our theoretical framework in real-world examples. We will also show applications of different abstract domains in different circumstances.

6.1 Australia Rain Prediction

This example makes use of weather data collections of Albury in Australia to obtain a model which predicts rainfall condition the next day. The grids approximation is selected as an abstract domain to analyze in the framework of probabilistic abstract interpretation.

6.1.1 The Dataset

We obtain an open-source dataset ‘Rain in Australia’ from Kaggle <https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package>, collected by Young et al. The original source is from observations gathered from a multitude of weather stations, offered by Bureau of Meteorology Australian Government <http://www.bom.gov.au/climate/data>.

The dataset comprises about 10 years of daily weather observations from numerous locations across Australia. Whether it will rain tomorrow is the target variable to predict. We say it rains if the rainfall for that day is 1mm or more. The dataset contains weather data from many different places/cities in Australia. For simplicity, here we only take the first city, Albury, to train our neural network. The dataset contains weather data of different dimensions, including temperature, windspeed, sunshine, humidity, pressure, cloud, etc. For simplicity, here we only take three variables ‘Minimum Temperature’, ‘Maximum Temperature’ and ‘Wind Gust Speed’ to train our neural network. Therefore, our training set contains in total 3040 days of data in Albury, and for each day it has three variables. We let each corresponding day’s ‘Rainfall’ to be our labeled output - it is a real value, and we say it rains if the value is 1mm or more. In preprocess, we preclassify ‘rainfall’ into either 0 or 1 by its value. As the original data contains some NA values, we also pre-transform all NA values into 0.

	MinTemp	MaxTemp	WindGustSpeed
0	13.4	22.9	44.0
1	7.4	25.1	44.0
2	12.9	25.7	46.0
3	9.2	28.0	24.0
4	17.5	32.3	41.0
...	...	...	...
3035	1.2	15.2	15.0
3036	0.8	13.4	17.0
3037	1.1	11.9	44.0
3038	1.1	14.1	28.0
3039	3.9	10.9	28.0

3040 rows × 3 columns

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 3)	12
activation (Activation)	(None, 3)	0
dense_1 (Dense)	(None, 3)	12
activation_1 (Activation)	(None, 3)	0
dense_2 (Dense)	(None, 1)	4
activation_2 (Activation)	(None, 1)	0

Total params: 28 (112.00 B)
Trainable params: 28 (112.00 B)
Non-trainable params: 0 (0.00 B)

Figure 6.1: Left: Sectioned dataset of rain in Austrilia. Right: Configuration of trained model using sectioned dataset.

6.1.2 The Model

We design our model to be a 3-3-3-1 fully connected neural network. That is, given 3 dimensional inputs, it contains three layers where first two are 3-unit dense layers with ReLU activation function and final layer outputs one single unit with Sigmoid activation function. Therefore, each of dense layers has 12 weights and the final layer contains 4 weights, so in total we have 28 weights in this network. We choose Adam as our Optimizer and binary cross-entropy as our Loss Function. During training, we set our training epoch to 20. In the final epoch of training, our model achieved metrics 0.8200 and loss 0.4596.

6.1.3 Probability Initialization

To analyze the model using a probabilistic abstract interpretation framework, we take the probability density into consideration. It begins from the initialization of the density of the input space. A straightforward way is to find an approximation of probability density from a training set, assuming that the training set is a sufficient sampling. More specifically, we have 3040 data points in our sectioned training set. Say, if an instance of data point only occurs once in this set, its corresponding probability will be $\frac{1}{3040}$. Any possible data point that has not occurred in the training set will be issued with zero probability.

6.1.4 Grids Approximation

Given the trained model, we know that, in principle, any combination of temperature and wind speed may occur on some day, so there are uncountably many possible inputs. We cannot exhaustively run them. We can use a probabilistic version of the abstract interpretation framework to analyze the trained model. To do that, we need to first choose our abstract domain.

One approach is to use grids approximation. We analyze the model through three different kinds of grids approximations. For all three kinds, we set the lower bound of the grids as the minimum value of each variable that has occurred in the training set and set the upper bound of grids as the maximum value of each variable has occurred in the training set. The abstract domain with lowest sparsity is the smallest grid size we can have, where each is in size $0.1 \times 0.1 \times 1$ (0.1 in ‘Minimum Temperature’, 0.1 in ‘Maximum Temperature’ and 1 in ‘Wind Gust Speed’). This is in fact a 3-dimensional hyperrectangle(or cube) such that

$$\mathcal{R}((0.1, 0.1, 1)) = [a_1, a_1 + 0.1] \times [a_2, a_2 + 0.1] \times [a_3, a_3 + 1] \quad (6.1)$$

To approximate the entire concrete domain into this size of grids, we will have in total 14,908,096 grids. This amount of grids is still very computation consuming and infeasible to be tested one by one. Therefore we need a sparser grids approximation. Recall the framework of sparser grids approximation, we take a further abstraction of current abstract domain, with each grid at this point to be in size $1 \times 1 \times 10$. That is, a 3-dimensional sparser hyperrectangle $\mathcal{R}'$ with size (1, 1, 10) is

$$\mathcal{R}' = [a'_1, a'_1 + 1] \times [a'_2, a'_2 + 1] \times [a'_3, a'_3 + 10] \quad (6.2)$$

where $\mathcal{R}((0.1, 0.1, 1))$ is an interior grid of $\mathcal{R}'$. In this abstract domain, we have 15,840 abstract elements(i.e. grids). This is much less computation-consuming, and it can show some relations between input variables and predictions. So, this could be a possible choice of abstract domain. However, we can also try an even sparser grids approximation, with each grid being size $10 \times 10 \times 50$

$$\mathcal{R}'' = [a''_1, a''_1 + 10] \times [a''_2, a''_2 + 10] \times [a''_3, a''_3 + 50] \quad (6.3)$$

where $\mathcal{R}'((1, 1, 10))$ is an interior grid of $\mathcal{R}''$. In this case, we only have in total 60 abstract elements. This could be a possible choice.

To connect theoretical framework with realistic meaning in example, we can also set abstract elements defined by concepts ‘cold/warm/hot minimum temperature, cold/warm/hot maximum temperature, windy or not windy’, keeping each being the same size. For example, the very first abstract element represents ‘cold minimum temperature, cold maximum temperature and not windy weather’. In this approximation, we have only 18 abstract elements. This could also be a possible choice of an abstract domain and shows the relations between the input variables and the predictions. In particular, it shows ‘hot’ minimum temperature will likely give a rain prediction, and ‘windy’ weather will likely give a rain prediction.

Each abstraction in this process will not only map a set of points into a centroid of a single abstract element but will also accumulate probabilities of the set of all points. The resulting corresponding probability of the abstract element is the sum of these probabilities. Given the probability initialization from the data, we have the probability density of the concrete input space. Therefore the probability density of each grids approximation can be computed. For example, in the last grids approximation, a combination of a cold minimum temperature, a cold maximum temperature and a

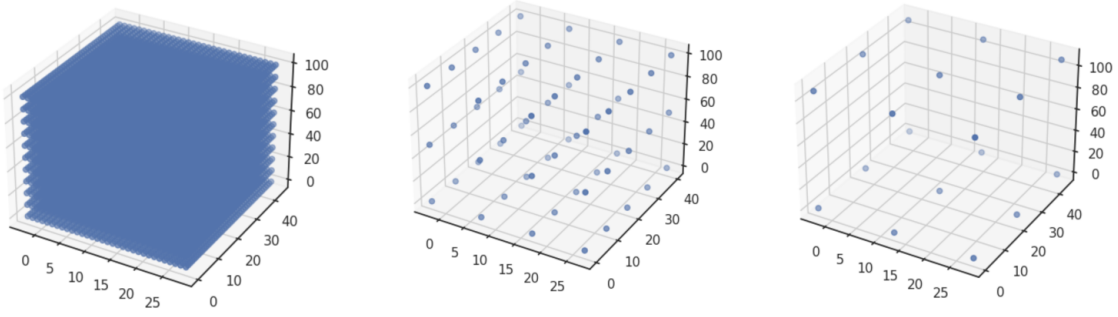


Figure 6.2: three different abstract domains of grids approximation in rain prediction example, where each dot representing centre point of a grid

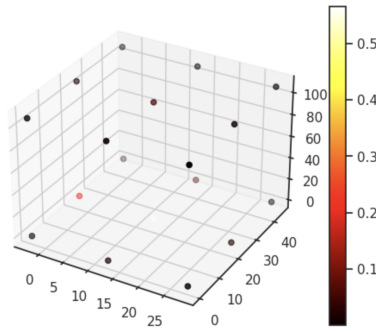


Figure 6.3: the 3rd grids approximation with initial probability distribution represented by heat map colors

not windy weather has probability 0.78%. This is a relatively low probability, showing that a weather like this is relatively rare. The most common weather is actually with a warm minimum temperature, a hot maximum temperature and a not windy weather, in Albury, which occupies 56.55% of the whole training set. The prediction of a high probability grid is more important because it occurs very common. In contrast, some weather only has less than 0.1% probability to occur, and it is negligible to consider when computation ability is limited.

Now in framework of probabilistic abstract interpretation, we have discussed Moore-Penrose pseudo-inverse pairs A and G , which are mappings between concrete space and abstract domain of grids of different sparsity. In specific matrix representation, we transform all entries of concrete data points or dense grids into single entry of the sparse grid that contains them, and sum up all probabilities of concrete points or dense grids to be the corresponding probability of the sparse grid. Then what remains is the function f and the abstract transformer $f^\#$. They both are affine mappings following the layer propagation of the neural network, from taking the coordinate value of centroid of each grid.

Given the weights of the trained model, the function f of the first layer is

$$f(x) = \begin{pmatrix} -0.63 & 0.69 & 0.79 \\ 0.61 & -0.54 & -0.53 \\ -0.07 & -0.40 & -0.71 \end{pmatrix} x + \begin{pmatrix} -0.25 \\ -0.28 \\ 0.43 \end{pmatrix} \quad (6.4)$$

rounding up to two decimal places. Hence, the abstract transformer for the first grid approximation $\mathcal{R}((0.1, 0.1, 1))$ will then be

$$\begin{aligned} f^\#(c_{(g_1)}^\#) &= f((a_1 + 0.05, a_2 + 0.05, a_3 + 0.5)) \\ &= \begin{pmatrix} -0.63 & 0.69 & 0.79 \\ 0.61 & -0.54 & -0.53 \\ -0.07 & -0.40 & -0.71 \end{pmatrix} \begin{pmatrix} a_1 + 0.05 \\ a_2 + 0.05 \\ a_3 + 0.5 \end{pmatrix} + \begin{pmatrix} -0.25 \\ -0.28 \\ 0.43 \end{pmatrix} \end{aligned} \quad (6.5)$$

for all grids where a_1, a_2, a_3 denotes the lower bounds of an arbitrary grid in this approximation. For example, taking one grid $[1.1, 1.2] \times [10.9, 11.0] \times [28, 29]$ which takes the abstraction of all concrete points contained inside, we have $a_1 = 1.1, a_2 = 10.9, a_3 = 28$, and propagation in abstract domain will be

$$\begin{aligned} f^\#((1.15, 10.95, 28.5)) &= \begin{pmatrix} -0.63 & 0.69 & 0.79 \\ 0.61 & -0.54 & -0.53 \\ -0.07 & -0.40 & -0.71 \end{pmatrix} \begin{pmatrix} 1.15 \\ 10.95 \\ 28.5 \end{pmatrix} + \begin{pmatrix} -0.25 \\ -0.28 \\ 0.43 \end{pmatrix} \\ &= \begin{pmatrix} 29.10 \\ -20.60 \\ -24.27 \end{pmatrix} \end{aligned} \quad (6.6)$$

The function following f is the activation function ReLU. In grids approximation the abstract transformer of ReLU, $ReLU^\#$, is applying ReLU on centroids of grids, so it is simply

$$ReLU^\#(f^\#(c_{(g_1)}^\#)) = ReLU(f((a_1 + 0.05, a_2 + 0.05, a_3 + 0.5))) \quad (6.7)$$

for all grids. In the above example this will be

$$ReLU^\# \left(\begin{pmatrix} 29.10 \\ -20.60 \\ -24.27 \end{pmatrix} \right) = \begin{pmatrix} 29.10 \\ 0 \\ 0 \end{pmatrix} \quad (6.8)$$

Similarly for the propagation of layer 2 and layer 3. Finally, the propagation of this grid through the whole model in abstract domain gives 0.22. This is a value close to 0, so the classifier gives ‘will not rain’ for this grid.

For abstract transformers of other sparser grids approximations, the computation is similar. In approximation $\mathcal{R}'((1, 1, 10))$ we have

$$f'^\#(c_{(g_2)}^\#) = f((a'_1 + 0.5, a'_2 + 0.5, a'_3 + 5)) \quad (6.9)$$

for all grids, and so on for further approximations $\mathcal{R}''$ etc.

6.1.5 Discussion

In this example, there are uncountably many possible data of weather observations. Grids approximation takes appropriate abstraction of the possible input space. In each grid, all possible weather data are close to each other and will likely cause the same weather prediction the next day. We are able to deduce only on the limited abstract elements and to see the relationships between a particular kind of weather data and next day prediction. According to corresponding probabilities, we can also tend to analyze more carefully about the weather data that are commonly observed, and to pay less attention on those that are rarely observed.

By applying the analysis framework, we firstly can have a clear data flow track inside the given neural network. In above example we took the grid $[1.1, 1.2] \times [10.9, 11.0] \times [28, 29]$ in input space which was a representative of all possible input values bounded by this grid. We tracked flow of this grid through three layers, seeing neuron values at hidden layers, and was classified by ‘will not rain’ at output layer. It is reasonable to claim that neighbor input values will have a close data flow in this problem, as it is continuous and smooth with respect to change in neighborhood. As the abstract transformer outputs $(29.10, 0, 0)$ in layer 1, we can claim that all possible values in this grid will result in $(x_1^2, 0, 0)$ where $x_1^2 \approx 29.10$ in layer 1. We can also claim that all possible values in this grid will result in $x^4 \approx 0.22$ and hence will be classified as ‘will not rain’.

Secondly, we can prove the neural network is robust to give a constant output in a major probability ($> \rho$, while ρ is the threshold an analyzer concerns about). A hot minimum and maximum temperature with a windy weather will highly likely to give a rain prediction. Although this is not a sound robustness, it offers additional information in context of probability that may all be needed by an analyzer.

6.2 Hand-written Digit Image Classification

This example takes a MNIST (“Modified National Institute of Standards and Technology”) handwritten digit classifier, which identifies digits from handwritten images. MNIST hand-written digit classification is a classic computer vision problem. We take this example to illustrate the probabilistic abstract interpretation framework on the convolutional neural network, and the grids approximation is selected as the abstract domain.

6.2.1 The Dataset

We obtain an open-source dataset ‘MNIST Digit Dataset’ from Kaggle <https://www.kaggle.com/competitions/digit-recognizer>. The dataset contains about 40,000 hand-written images of digits. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

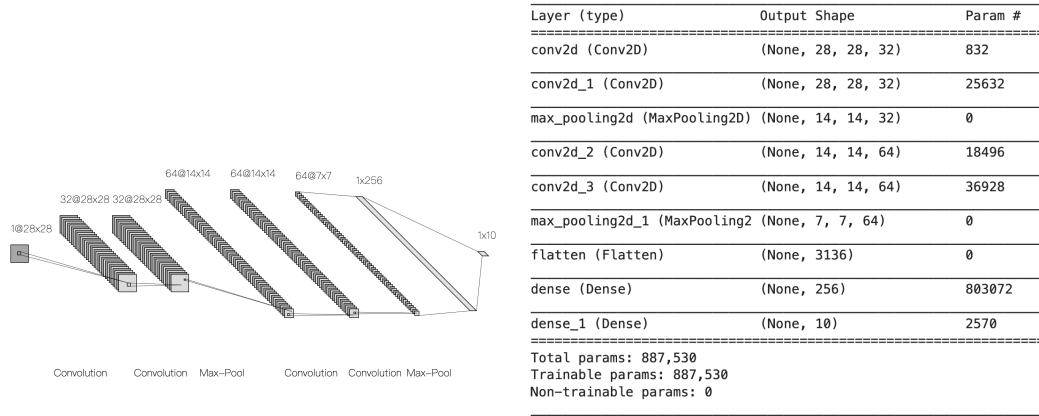


Figure 6.4: Left: Model architecture of digit classifier. Right: Model configuration of digit classifier.

6.2.2 The Model

We design our model to be an eight-layer convolutional neural network, consisting of two $28 \times 28 \times 32$ convolutional layers with 5×5 filter followed by a 2×2 max-pooling layer, and then another two $14 \times 14 \times 64$ convolutional layers with 3×3 filter followed by a 2×2 max-pooling layer, and finally two dense layers, respectively. In total, the network contains 887,530 parameters. We choose RMSprop as our Optimizer and categorical cross entropy as our Loss Function. In the final epoch of training, our model achieved metrics 0.8917 and loss 0.3446.

6.2.3 Probability Initialization

To analyze the model by probabilistic abstract interpretation framework, we initialize density distribution of input space. We can again find an approximation of probability density from the training set, assuming that the training set is a sufficient sampling. More specifically, we have 40,000 data points in this training set. Say, if an instance of data point only occurs once in this set, its corresponding probability will be $\frac{1}{40000}$. Any possible data point that has not occurred in the training set will be issued with zero probability.

6.2.4 Grids Approximation

Given an input image has 784 pixels and each pixel has 256 pixel values, we know that there are 256^{784} possible input images which is a tremendously huge number and we cannot exhaustively run them. To analyze by probabilistic abstract interpretation framework, we first need to choose appropriate abstract domain.

A straightforward approach we can take is grids approximation. We can analyze the model through different kinds of grids approximations, with different sparsity. The concrete space is a 784-dimensional space with an integer value between 0 and 255 in each dimension. We can take an arbitrary abstract domain that takes the

abstraction of concrete space. For example, we can take a grid of size 16^{784} as an abstract element, that is, a 784-dimensional square grid such that

$$g_1 = [a_1, a_1 + 16] \times [a_2, a_2 + 16] \times \dots \times [a_{784}, a_{784} + 16] \quad (6.10)$$

with lower bound inclusive and upper bound exclusive in each interval. Therefore, to approximate the entire concrete domain into this grid size, we will have in total $(\frac{256}{16})^{784} = 16^{784}$ grids. This amount of grids is still very computation consuming. To make the approximation fit better, we can also reduce the dimensionality. That is, we align a group of dimensions with the same pixel values (in this problem, say, we align neighborhood pixels so that we analyze in lower resolution). If we align each neighborhood of 4×4 pixels in a square, we can then have a grid approximation with each grid of size 16^{49} such that

$$g_2 = [a_1, a_1 + 16] \times [a_2, a_2 + 16] \times \dots \times [a_{49}, a_{49} + 16] \quad (6.11)$$

with lower bound inclusive and upper bound exclusive in each interval. This now looks much better; however, in fact, it is still a huge number. To reduce this number we can take more sparsity in both sides. Say we further double grid size and also combine a neighborhood of 7×7 pixel in a square. This will result in

$$g_3 = [a_1, a_1 + 32] \times [a_2, a_2 + 32] \times \dots \times [a_{16}, a_{16} + 32] \quad (6.12)$$

This remains in total 8^{16} grids. To even further simplify, we can classify each pixel-value into only two categories, being value 0 as dark and 255 as bright respectively. A grid in this case is 16-dimensional, and each dimension has two choices of values.

$$g_4 = [a_1, a_1 + 128] \times [a_2, a_2 + 128] \times \dots \times [a_{16}, a_{16} + 128] \quad (6.13)$$

Therefore, in total, we have 2^{16} grids in this approximation.

So far we have discussed abstract domains of different sparsity, as well as Moore-Penrose pseudo inverse pairs A and G . Recall in grids approximation, function in concrete space f and abstract transformer $f^\#$ are both affine mappings, taking coordinate value of centroid of each grid. In this case, $f^\#$ of one of filters in first layer is a 28×28 matrix in form of

$$f^\# = \begin{pmatrix} g_{11} & g_{12} & g_{13} & 0 & \dots & 0 & g_{21} & g_{22} & g_{23} & 0 & \dots & g_{31} & g_{32} & g_{33} & 0 & \dots \\ 0 & g_{11} & g_{12} & g_{13} & \dots & 0 & 0 & g_{21} & g_{22} & g_{23} & \dots & 0 & g_{31} & g_{32} & g_{33} & \dots \\ 0 & 0 & g_{11} & g_{12} & \dots & 0 & 0 & 0 & g_{21} & g_{22} & \dots & 0 & 0 & g_{31} & g_{32} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (6.14)$$

where the convolution size is 3×3 .

By taking approximation g_4 and computing some of representative grids through propagation with abstract transformers, we can see that, for example, the model will more likely recognize a bright top-left corner as a feature of digit '3', and will more likely recognize a bright bottom-right corner as a feature of digit '2' or '9'.

6.2.5 Discussion

In this example, there are a huge number of possible input images. Grids approximation takes appropriate abstraction of the input space. In each grid, all images with close pixel values will likely cause the same prediction of digit. We are able to deduce only on the limited abstract elements and to see the relationships between a particular group of images with similar features and the prediction.

We also compare this illustration with the work in [34]. [34] gives an example of a classical abstract interpretation analyzer to prove robustness against brightening attack on the MNIST digit classifier, where the analyzer can prove whether a network corrects classification under attack which perturbs an image by changing all pixels above a particular threshold to the brightest possible value. In this case, probabilistic abstract interpretation aims for different purposes. It shows probability density flow of all possible inputs and analyzes how the neural network works given input with different features.

In this case, the probabilistic abstract interpretation framework also helps to prove robustness of the neural network in context of major probability ($> \rho$). Given by the most sparse grids approximation that the abstract transformer transforms a bright(brightness = a) top-left corner grid into an output of digit ‘3’, we can claim that any brighter(brightness $> a$) top-left corner grid will be high likely ($> \rho$) transformed into an output of digit ‘3’. The property of robustness in major probability, although not sound, may be helpful for this neural network analysis.

Secondly, the framework offers an analysis of interpretability of the neural network, showing whether it is aligned with human’s way to make this classification. For example, a horizontally consecutive bright sequence of grids at bottom classified into output of digit ‘2’ by abstract transformer is aligned with human’s understanding, as a bottom horizontal line is a key feature of digit ‘2’ a human will recognize. However, a bright bottom-right corner as a feature of digit ‘9’ is not aligned with human’s understanding. In this case, an alternative neural network with different architecture may be trained and analyzed if interpretability is sought for.

6.3 GPA Predictor

In this example, we explore the relationships between student GPAs and factors that include student IQ and study hours. We take a distribution approximation to illustrate analysis of this GPA predictor model in the framework of probabilistic abstract interpretation.

6.3.1 The Dataset

The dataset is merged by two distinct datasets, with one containing information about student GPAs and IQs and the other detailing GPAs and study hours per week. The first dataset contains 78 observations of pairs of students’ GPAs and IQs, while the second dataset offers a sample of 193 undergraduate students who took an introductory statistics course in 2012 at a private US university with their GPAs and

study hours per week. Data cleaning and pre-processing is taken to handle missing values and duplicate rows. Since GPAs are in different scaling in two datasets, normalization is taken for purpose of alignment, so that GPA ranges from 0 to 4 points. The IQ values that appear in the dataset are between 72 and 136.

6.3.2 The Model

We design our model to be a fully connected 2-2-2-1 neural network, containing three layers with the ReLU activation function, and the final layer outputs one single unit with sigmoid activation function. In total, we have 25 weights in this neural network. We choose Adam as our Optimizer and cross-entropy as our Loss Function.

6.3.3 Probability Initialization

To analyze the model by probabilistic abstract interpretation framework, we initialize the probability density of the input space. Since the dataset is relatively small, the approximation of density distribution of occurrences may not properly represent real distribution of all possible data points potentially in input space. Fortunately, we have prior knowledge that IQ value follows a Gaussian distribution, hence we can take it as initialized distribution and estimate the parameters of distribution from the given dataset. Therefore, we can obtain probability initialization of IQ values

$$X_{IQ} \sim \mathcal{N}(108.92, 13.09^2) \quad (6.15)$$

Students' study hours can also be regarded as Gaussian distributed. Since in practice most of students study a moderate amount and there is a smaller group studying excessively, we can approximate it by a Gaussian distribution with a mean of moderate study hours. We approximate the distribution parameters from the given dataset and assume that it is the case of all possible data of the student study hours. That is,

$$X_{hours} \sim \mathcal{N}(17.48, 11.38^2) \quad (6.16)$$

Therefore, the combination of two distributions in two-dimensional follows a bivariate Gaussian distribution

$$\mathbf{X} \sim \mathcal{N}_2\left(\begin{pmatrix} 108.92 \\ 17.48 \end{pmatrix}, \begin{pmatrix} 13.09^2 & 0 \\ 0 & 11.38^2 \end{pmatrix}\right) \quad (6.17)$$

as IQ values and study hours are independent factors.

6.3.4 Distribution Approximation

To analyze the model by probabilistic abstract interpretation framework, we first choose the appropriate abstract domain. In this example approximation of distribution of input space has already been taken, and we can choose distribution approximation as abstract domain. Given from probability initialization that the input

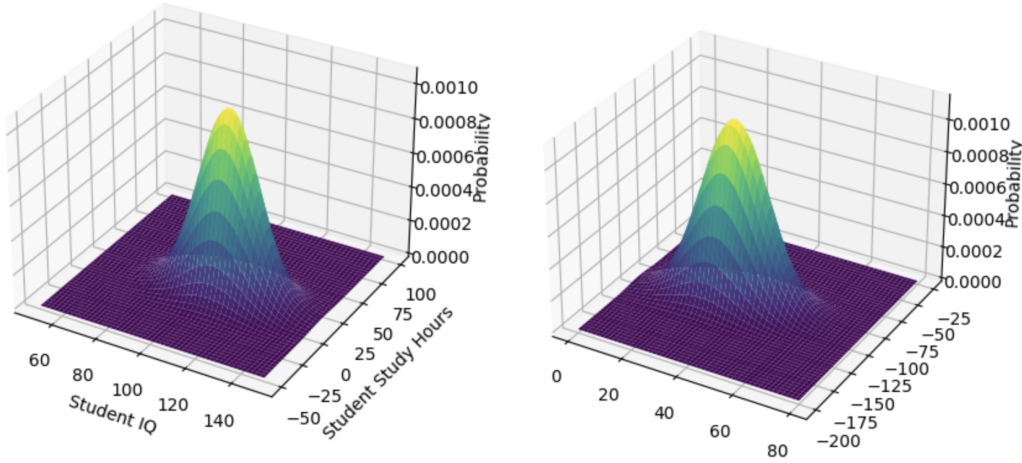


Figure 6.5: Left: Probability density initialization of IQ and study hours following two-dimensional Gaussian distribution. Right: Probability density distribution after layer 1.

space follows a bivariate Gaussian distribution, we approximate the input space by its probability density function

$$\begin{aligned}
 y_{X_{IQ}, X_{hours}} &= y_{X_{IQ}} \cdot y_{X_{hours}} \\
 &= \frac{1}{2\pi\sigma_{X_{IQ}}\sigma_{X_{hours}}} e^{-\frac{(x_{IQ}-\mu_{X_{IQ}})^2}{2\sigma_{X_{IQ}}^2} - \frac{(x_{hours}-\mu_{X_{hours}})^2}{2\sigma_{X_{hours}}^2}} \\
 &= \frac{1}{297.93\pi} e^{-\frac{(x_{IQ}-108.92)^2}{342.70} - \frac{(x_{hours}-17.48)^2}{259.01}}
 \end{aligned} \tag{6.18}$$

which is the abstract element. This is a Moore-Penrose pseudo-inverse that takes an abstraction of input space into one single abstract element. Once we have a probability density function that approximates the concrete space, we can have the abstract transformer $f^\#$ in the abstract domain that maps a distribution to a distribution

$$f^\#(y(\mathbf{x})) = y(f^{-1}(\mathbf{x})) \tag{6.19}$$

From the trained model, we can get weights for the first layer which are

$$f(\mathbf{x}) = W_1\mathbf{x} + b_1 = \begin{pmatrix} 0.42 & -0.65 \\ -0.91 & -0.79 \end{pmatrix} \begin{pmatrix} x_{IQ} \\ x_{hours} \end{pmatrix} + \begin{pmatrix} -0.02 \\ 0. \end{pmatrix} \tag{6.20}$$

We know that if $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$, then $A\mathbf{X} + b \sim \mathcal{N}(A\mu + b, A^2\Sigma)$. Therefore, after the first layer

$$\begin{aligned}
 W^1\mathbf{X} + b^1 &\sim \mathcal{N}\left(W^1 \begin{pmatrix} 108.92 \\ 17.48 \end{pmatrix} + b^1, (W^1)^T W^1 \begin{pmatrix} 13.09^2 & 0 \\ 0 & 11.38^2 \end{pmatrix}\right) \\
 &\sim \mathcal{N}\left(\begin{pmatrix} 34.36 \\ -112.93 \end{pmatrix}, \begin{pmatrix} 131.94 & 31.08 \\ 58.26 & 157.99 \end{pmatrix}\right)
 \end{aligned} \tag{6.21}$$

It turns out that the probability density function becomes

$$\begin{aligned} f^{1\#}(y_{X_{IQ}, X_{hours}}) &= \frac{1}{297.93\pi} e^{-\frac{(0.42x_{IQ}-0.65x_{hours}-108.94)^2}{342.70} - \frac{(0.91x_{IQ}-0.79x_{hours}-17.48)^2}{259.01}} \\ &= c_1^\# \end{aligned} \quad (6.22)$$

as the abstract element transformed by an abstract transformer. This is a bivariate Gaussian distribution in which the parameters change after the propagation of the layer. We can continue to apply an abstract transformer for other layers similarly, so that the propagation of probability density function in abstract domain takes abstraction and represents propagation of concrete data points in concrete domain.

6.3.5 Discussion

In this example, we illustrate the distribution approximation which gives appropriate abstraction of input space that we cannot exhaustively test on. We take a bivariate Gaussian distribution to approximate the data according to features of dataset.

We can have abstract transformers for different layers by extracting weights of trained model, and we apply abstract transformers on the probability density distribution to see the change of distribution during propagation.

The example is simple that it has only a few dimensions of features. The bivariate Gaussian distribution is also a simple distribution that can be taken to approximate. In the context of higher dimension and more complicated density distributions, the computation might be much harder. This can be viewed as a starting point for an illustration of how a distribution approximation works in the framework.

In distribution approximation, we can always map from abstract domain back to concrete domain by sampling. This is a straightforward pseudo inverse function G . Sampling will always give us an appropriate set of concrete data points in need.

One benefit to use the probabilistic abstract interpretation framework with a distribution approximation in this case is the ability to give an analysis of a complete set of possible data, while the real dataset is limited. The distribution approximation is able to simulate all possible data, in this case by an initialization of bivariate Gaussian distribution. Another benefit of distribution approximation here is that only a single abstract element needs to be considered instead of testing on all possible values of student IQs and study hours. Also, the distribution approximation is able to keep as much information from concrete dataset as possible.

6.4 Wine Quality Classification

This example classifies wine by quality according to its key chemical attributes. The clusters approximation is selected as an abstract domain to analyze in the framework of probabilistic abstract interpretation.

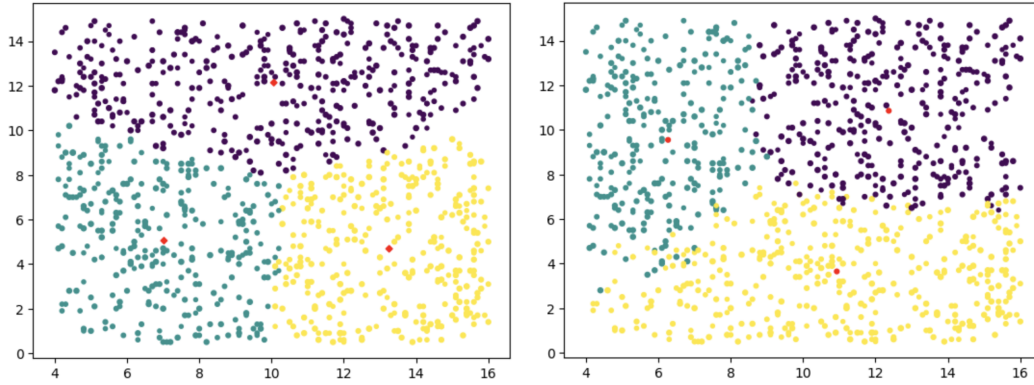


Figure 6.6: Left: K-means clustering($K=3$) of the dataset shown in first two dimensional. The three centroids are shown in red dots. Right: Gaussian Mixture Model($K=3$) predictions of the dataset shown in first two dimensional. The three means are shown in red dots.

6.4.1 The Dataset

This dataset contains 1,000 synthetic wine samples, each described by four key chemical features: level of fixed acidity, sugar level remaining after fermentation, alcohol content(%) of the wine, and liquid density of the wine sample. Each sample is labeled with a wine quality class (low, medium, or high) based on a simulated quality scoring system. In total there is around 34(%) of wine samples labeled as high quality, 36(%) of wine samples labeled as medium quality and 30(%) of wine samples labeled as low quality.

6.4.2 The Model

We design our model to be a fully connected 4-4-4-1 neural network. It contains four layers, including two dense layers with the ReLU activation function, and the final layer outputs a single unit with the softmax activation function. Therefore, each of dense layers has 20 weights and the final layer contains 5 weights, so in total we have 45 weights in this network. We choose Adam as our Optimizer and cross-entropy as our Loss Function.

6.4.3 Probability Initialization

To analyze the model by probabilistic abstract interpretation framework, we initialize the probability density of the input space. We find an approximation of probability density from the training set, assuming that the training set is a sufficient sampling. More specifically, we have 1,000 data points in our sectioned training set, and any instance of data point which occurs once in this set has corresponding probability $\frac{1}{1000}$. Any possible data point that has not occurred in the dataset will be issued with zero probability.

6.4.4 Clusters Approximation

In principle, there are uncountably many possible combinations of chemical features of wine, and we cannot exhaustively run them. To analyze the model by probabilistic abstract interpretation framework, we first need to choose appropriate abstract domain.

In this case, we choose clusters approximation as abstract domain. Because in probability initialization, we initialize data points that have occurred in dataset with probability in uniform distribution, and those that have not occurred in dataset with zero probability, we ignore data with zero probabilities in clustering because it has no affect in accumulation of probability dimension. That is, we categorize the entire dataset into K clusters. We can let $K = 3$ and we can take K-means clustering. That is, given the dataset $x^{(1)}, \dots, x^{(1000)}$ and the labels of data points $c^{(1)}, \dots, c^{(1000)}$, we first initialize cluster centroids μ_1, μ_2, μ_3 randomly and iterate by K-means algorithm until convergence.

Eventually we get $\mu_1 = (10.062, 12.139, 11.056, 0.997)$, $\mu_2 = (7.022, 5.049, 10.873, 0.998)$, $\mu_3 = (10.062, 12.139, 11.056, 0.997)$, which are values of abstract elements $c_1^\#, c_2^\#, c_3^\#$. We also have

$$p_{c_1^\#} = \sum_i p_{x^{(i)}} \{c^{(i)} == 1\} = 0.001 \times 389 = 0.389 \quad (6.23)$$

and $p_{c_2^\#} = 0.295$, $p_{c_3^\#} = 0.316$. That is, we have a Moore-Penrose pseudo-inverse that takes an abstraction of input space into three abstract elements, each representing a group of possible data points, respectively. We also know that the abstract transformer that performs on abstract elements is $f^\# : C^\# \rightarrow D^\#$

$$f^\#(c_k^\#) = f(\mu_k) \quad (6.24)$$

and we can obtain f from the trained model that

$$f(x) = \begin{pmatrix} 0.004 & -0.759 & -0.689 & 0.492 \\ -0.649 & 0.512 & 0.712 & 0.176 \\ -0.213 & -0.367 & 0.491 & 0.222 \\ -0.775 & -0.320 & -0.261 & 0.598 \end{pmatrix} x + \begin{pmatrix} 0. \\ -0.062 \\ -0.082 \\ 0.077 \end{pmatrix} \quad (6.25)$$

for first layer, and so on. We can perform abstract transformers of all layers on each abstract element throughout the model. Performing on $c_1^\#$ will output a set of probabilities indicating highest likeliness being ‘medium’ quality.

We can also try clusters approximation when $K = 9$. By applying the same algorithm, we can get nine centroids $\mu_1, \dots, \mu_9$, and therefore we get abstract elements $c_1^\#, \dots, c_9^\#$ with their corresponding $p_{c_1^\#}, \dots, p_{c_9^\#}$. We again perform abstract transformers of all layers on each of nine abstract elements to find which level of quality each group of features of wine will be classified to be.

Another approach of clusters approximation is clustering by Gaussian Mixture Model. Similarly to K-means, parameters of a Gaussian mixture model are also estimated using iterative Expectation-Maximization algorithm. Let $K = 3$, we obtain three

Gaussians from the dataset such that the abstract element of the first Gaussian is

$$\begin{aligned}
 c_1^\# &= \langle \mu_1, \Sigma_1, w_1 \rangle \\
 &= \left\langle \begin{pmatrix} 12.367 \\ 10.858 \\ 11.124 \\ 0.997 \end{pmatrix}, \begin{pmatrix} 4.246 & -0.469 & 0.019 & 0.000 \\ -0.469 & 7.112 & 0.126 & 0.000 \\ 0.019 & 0.126 & 3.235 & 0.001 \\ 0.000 & 0.000 & 0.001 & 0.000 \end{pmatrix}, 0.342 \right\rangle
 \end{aligned} \tag{6.26}$$

Rounding to three decimal places, and similar for abstract elements of other Gaussians $c_2^\#, c_3^\#$. We also have abstract transformer $f^\#$ for the first layer given the first layer is $f^1(\mathbf{x}) = A\mathbf{x} + b$

$$\begin{aligned}
 f^\#(c_1^\#) &= \langle f^\#(\mu_1), f^\#(\Sigma_1), w_1 \rangle \\
 &= \langle A\mu_1 + b, A\Sigma_1 A^T, w_1 \rangle \\
 &= \left\langle \begin{pmatrix} 0.004 & -0.759 & -0.689 & 0.492 \\ -0.649 & 0.512 & 0.712 & 0.176 \\ -0.213 & -0.367 & 0.491 & 0.222 \\ -0.775 & -0.320 & -0.261 & 0.598 \end{pmatrix} \begin{pmatrix} 12.367 \\ 10.858 \\ 11.124 \\ 0.997 \end{pmatrix} + \begin{pmatrix} 0.000 \\ -0.062 \\ -0.082 \\ 0.077 \end{pmatrix}, \right. \\
 &\quad \left. \begin{pmatrix} 0.004 & -0.759 & -0.689 & 0.492 \\ -0.649 & 0.512 & 0.712 & 0.176 \\ -0.213 & -0.367 & 0.491 & 0.222 \\ -0.775 & -0.320 & -0.261 & 0.598 \end{pmatrix} \begin{pmatrix} 4.246 & -0.469 & 0.019 & 0.000 \\ -0.469 & 7.112 & 0.126 & 0.000 \\ 0.019 & 0.126 & 3.235 & 0.001 \\ 0.000 & 0.000 & 0.001 & 0.000 \end{pmatrix} \right. \\
 &\quad \left. \begin{pmatrix} 0.004 & -0.759 & -0.689 & 0.492 \\ -0.649 & 0.512 & 0.712 & 0.176 \\ -0.213 & -0.367 & 0.491 & 0.222 \\ -0.775 & -0.320 & -0.261 & 0.598 \end{pmatrix}^T, 0.342 \right\rangle \\
 &= \left\langle \begin{pmatrix} -15.366 \\ 5.567 \\ -1.018 \\ -15.289 \end{pmatrix}, \begin{pmatrix} 5.767 & -4.698 & 0.796 & 2.083 \\ -4.698 & 5.679 & 0.311 & 0.406 \\ 0.798 & 0.311 & 1.808 & 0.943 \\ 2.083 & 0.406 & 0.943 & 3.295 \end{pmatrix}, 0.342 \right\rangle
 \end{aligned} \tag{6.27}$$

and so on for other layers.

We can also try a larger number K of Gaussians, in which case we will have a larger number of abstract elements, each of which takes abstraction of a smaller group of data points.

6.4.5 Discussion

In this example we illustrate the clusters approximation which gives appropriate abstraction of input space that we cannot exhaustively test on. We take both K-means approach and the Gaussian mixture model approach.

In K-means approach, each cluster takes a group of different possible wines with features that are close to each other, and has a centroid which has the shortest 4-dim Euclidean distance to all points in the group. The abstract element of each cluster represents all of concrete points, and we take abstract transformers performing on

abstract elements instead of taking concrete affine functions performing on uncountably many data points. We can choose a different number K of clusters. A larger K will give more specifically clustered groups with a smaller size and data points with closer features. In this example, comparing $K = 3$ and $K = 9$, the latter can give more specific information on which level of wine quality each group of features will be classified to be.

In Gaussian Mixture Model approach, each abstract element not only has a mean value but also has covariance as well as mixture weight. An abstract element can bring more information through propagation in abstract domain. Similarly to K-means, a larger number K of Gaussians with each takes the abstraction of a smaller group of data points and takes more specific information for each group of features of the wine.

The benefit of use of probabilistic abstract interpretation in this example is to analyze interpretability of the neural network, showing features of input factors that influences quality labels of wine. In this problem scenario, the links between input factors and final label is direct. Although an arbitrarily deep neural network could be designed, an analyzer is more interested in direct relationships between input layer and output layer. As the output layer only contains three quality labels where each label contains nearly equal size of dataset, a clusters approximation (with relatively small number of clusters) is suitable for this analysis, instead of a grids approximation with too many grids which is unnecessary. An analyzer may be interested in seeing each cluster is transformed in abstract domain into a different label, showing features that are the most influential to be classified into each label. An alternative neural network may be checked for a higher interpretability if this is NOT the case.

6.5 Conclusions

In this chapter, we took four experiments on real world problems to show advantages of applying probabilistic abstract interpretation analysis framework. At the same time we have also shown a choice of suitable approximation approaches for different problem circumstances. In each of these experiments, we obtained real open-sourced datasets, designed and trained neural networks using these datasets from sketch. We were able to make probability initialization according to training datasets. With hold of real world datasets, the probability initialization could be more reasonable than a uniform distribution. We also gave demonstration of how the analysis perform in abstract domain, showing probability density distribution flow of data. By these demonstrations we have shown the neural networks are under control in context of probability and have given a better understanding of data flow inside these neural networks. We have also compared analysis by different approximation approaches for the same neural network, seeing advantages and disadvantages of each approximation approach.

However, one of additional extensions could be added into these experimental demonstrations will be a comparison of different designs of architectures of neural networks for the same problem, which is one of main functionalities of our analysis framework. For each of these experiments only one neural network architecture was designed

and trained. Having more than one neural networks could give our analysis an opportunity to see which architecture is MORE robust and interpretable with respect to the problem and the dataset, or to show a counterexample that a particular neural network is NOT robust or interpretable.

Chapter 7

Conclusions and Future Work

In this thesis, we aimed to construct a novel framework to analyze possible data flow in context of probability density distribution of a neural network based on the theory of probabilistic abstract interpretation. With this framework, we can have a better understanding of how a neural network makes its decisions and prove certain properties of this neural network such as robustness. In previous chapters, we have discussed the application of the framework on various kinds of neural network layer through examples and mathematical deductions. We have also discussed several approximation approaches with different abstract domains regarding certain properties of a neural network to be analyzed.

Although we have shown basic functionality of the framework, limitations still remained. The dimensional problem has always been along with the analysis. With a typical neural network containing thousands of neurons, the dimension of analysis would grow exponentially. Even a domain of real numbers would make the analysis touch on the problem of infinite dimension. Deeper layers have also remained as a difficult part of analysis. Works were needed to be continued and further extended to complete and develop a stronger functionality of this framework.

Firstly, there has to be a general dimension reduction method with respect to certain problem circumstances. A convolutional layer may handle input with millions of pixels and seeing each pixel as independent will cause a tremendously large dimension. One approach we have discussed is to design an appropriate abstract domain and consider a single abstract element of a group of pixels. The question remains as to which abstract domain is appropriate and what size of group of pixels should be chosen to keep the analysis valid. The same situation holds for the context of recurrent neural networks. A trade-off between efficiency with reduced dimension and quality of analysis may always be considered.

Secondly, more state-of-the-art architectures of neural networks have to be analyzed via the framework to make it more applicable in real problems. There are different architectures that are typically used in particular real-world applications. In convolutional models, AlexNet and VGGNet are popular for dealing with image recognition problems. ResNet adds a skip connection to enable model with more than a thousand layers. In recurrent models, except for LSTM, GRU is also a commonly used and powerful architecture. The transformer is powerful in large language models [52]. Besides, there are further more types of architectures such as generative mod-

els or graph neural networks that this framework may be applicable at. Further exploration is left for future work.

Bibliography

- [1] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436. pages 8
- [2] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press. pages 8
- [3] Hailesilassie, T. (2016). Rule extraction algorithm for deep neural networks: A review. *arXiv preprint arXiv:1610.05267*. pages 29
- [4] Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR abs/1312.6034*. pages 29
- [5] Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., & Clune, J. (2016). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29. pages 29
- [6] Bazen, S., & Joutard, X. (2013). The Taylor decomposition: A unified generalization of the Oaxaca method to nonlinear models. pages 29
- [7] Landecker, W., Thomure, M. D., Bettencourt, L. M. A., Mitchell, M., Kenyon, G. T., & Brumby, S. P. (2013). Interpreting individual classifications of hierarchical networks. In *2013 IEEE symposium on computational intelligence and data mining (CIDM)* (pp. 32-38). IEEE. pages 29
- [8] Bach, S., Binder, A., Montavon, G., Klauschen, F., Muller, K. R., & Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10(7), 1–46. pages 29
- [9] Samek, W., Wiegand, T., & Müller, K. R. (2017). Explainable Artificial Intelligence: Understanding, Visualization and Interpreting Deep Learning Models. *arXiv:1708.08296*. pages 29
- [10] Zilke, J. R., Mencía, E. L., & Janssen, F. (2016, October). DeepRED–Rule Extraction from Deep Neural Networks. In *International Conference on Discovery Science* (pp. 457-473). Springer, Cham. pages 29
- [11] Montavon, G., Samek, W., & Müller, K. R. (2017). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*. pages 29

- [12] Jacobsson, H. (2005). Rule extraction from recurrent neural networks: Ataxonomy and review. *Neural Computation*, 17(6), 1223-1263. pages 29
- [13] Wang, Q., Zhang, K., Ororbia, I. I., Alexander, G., Xing, X., Liu, X., & Giles, C. L. (2017). An empirical evaluation of recurrent neural network rule extraction. *arXiv preprint arXiv:1709.10380*. pages 29
- [14] Quanshi Zhang, Ruiming Cao, Feng Shi, Ying Nian Wu, & Song-Chun Zhu (2017). Interpreting CNN Knowledge Via An Explanatory Graph. *AAAI 2018*. pages 30
- [15] Quanshi Zhang, Ying Nian Wu, & Song-Chun Zhu (2017). Interpretable Convolutional Neural Networks. *CVPR 2018*. pages 30
- [16] Bolei Zhou, David Bau, Aude Oliva, & Antonio Torralba. (2018). Interpreting Deep Visual Representations via Network Dissection. *IEEE transactions on pattern analysis and machine intelligence*, 41(9), 2131-2145. pages 30
- [17] Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham. pages 30
- [18] Bau, D., Zhou, B., Khosla, A., Oliva, A., & Torralba, A. (2017). Network dissection: Quantifying interpretability of deep visual representations. *30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. pages 30
- [19] Carlini, N., & Wagner, D. (2017, May). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)* (pp. 39-57). Ieee. pages 30
- [20] Nielson, F., Nielson, H. R., & Hankin, C. (2015). *Principles of program analysis*. Springer. pages 11, 12, 30
- [21] Cousot, P., & Cousot, R. (1977, January). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages* (pp. 238-252). ACM. pages 11, 30
- [22] Nielson, F., & Jones, N. (1994). Abstract interpretation: a semantics-based tool for program analysis. *Handbook of logic in computer science*, 4, 527-636. pages 30
- [23] Cousot, P., & Monerau, M. (2012, March). Probabilistic abstract interpretation. In *European Symposium on Programming* (pp. 169-193). Springer, Berlin, Heidelberg. pages 30
- [24] Di Pierro, A., & Wiklicky, H. (2000, September). Concurrent constraint programming: towards probabilistic abstract interpretation. In *International Conference on Principles and Practice of Declarative Programming: Proceedings of the 2nd ACM SIGPLAN international conference on Principles and practice of declarative programming* (Vol. 20, No. 23, pp. 127-138). pages 11, 30

- [25] Di Pierro, A., Hankin, C., & Wiklicky, H. (2010). Probabilistic semantics and program analysis. In *Formal Methods for Quantitative Aspects of Programming Languages* (pp. 1-42). Springer, Berlin, Heidelberg. pages 18, 30
- [26] Di Pierro, A., & Wiklicky, H. (2016). Probabilistic abstract interpretation: From trace semantics to DTMC's and linear regression. In *Semantics, Logics, and Calculi* (pp. 111-139). Springer, Cham. pages 15, 30
- [27] Di Pierro, A., Sotin, P., & Wiklicky, H. (2008). Relational analysis and precision via probabilistic abstract interpretation. *Electronic Notes in Theoretical Computer Science*, 220(3), 23-42. pages 30
- [28] Di Pierro, A., & Wiklicky, H. (2013, August). Probabilistic analysis of programs: a weak limit approach. In *International Workshop on Foundational and Practical Aspects of Resource Analysis* (pp. 58-76). Cham: Springer International Publishing. pages 30
- [29] Di Pierro, A., Hankin, C., & Wiklicky, H. (2007). Abstract interpretation for worst and average case analysis. In *Program Analysis and Compilation, Theory and Practice: Essays Dedicated to Reinhard Wilhelm on the Occasion of His 60th Birthday* (pp. 160-174). Berlin, Heidelberg: Springer Berlin Heidelberg. pages 30
- [30] Di Pierro, A., Hankin, C., & Wiklicky, H. (2010). Program analysis probably counts. *The Computer Journal*, 53(6), 871-880. pages 30
- [31] Di Pierro, A., Hankin, C., & Wiklicky, H. (2007). On probabilistic techniques for data flow analysis. *Electronic Notes in Theoretical Computer Science*, 190(3), 59-77. pages 30
- [32] Di Pierro, A., & Wiklicky, H. (2013). Probabilistic data flow analysis: a linear equational approach. *Proceedings of GandALF'13, EPTCS*, Volume 119, 2013. pages 30
- [33] Wiklicky, H. (2016, May). On dynamical probabilities, or: how to learn to shoot straight. In *International Conference on Coordination Languages and Models* (pp. 262-277). Cham: Springer International Publishing. pages 30
- [34] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., & Vechev, M. (2018). AI²: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. pages 30, 56, 57, 65, 78, 87
- [35] Singh, G., Gehr, T., Püschel, M., & Vechev, M. (2019). An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL), 41. pages 30
- [36] Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer*

- Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30 (pp. 97-117). Springer International Publishing. pages 24, 30
- [37] Singh, G., Gehr, T., Mirman, M., Püschel, M., & Vechev, M. (2018). Fast and effective robustness certification. *Advances in neural information processing systems*, 31. pages 31
- [38] Mirman, M., Gehr, T., & Vechev, M. (2018, July). Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning* (pp. 3578-3586). PMLR. pages 31
- [39] Ghorbal, K., Goubault, E., & Putot, S. (2009, June). The Zonotope abstract domain taylor1+. In *International conference on computer aided verification* (pp. 627-633). Berlin, Heidelberg: Springer Berlin Heidelberg. pages 30
- [40] Ryou, W., Chen, J., Balunovic, M., Singh, G., Dan, A., & Vechev, M. (2021, July). Scalable polyhedral verification of recurrent neural networks. In *International Conference on Computer Aided Verification* (pp. 225-248). Cham: Springer International Publishing. pages 31
- [41] Steffen, S., Gehr, T., Tsankov, P., Vanbever, L., & Vechev, M. (2020, July). Probabilistic verification of network configurations. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (pp. 750-764). pages 31
- [42] Mangal, R., Nori, A. V., & Orso, A. (2019, May). Robustness of neural networks: A probabilistic and practical approach. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)* (pp. 93-96). IEEE. pages 31
- [43] Păsăreanu, C., Converse, H., Filieri, A., & Gopinath, D. (2020, June). On the probabilistic analysis of neural networks. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (pp. 5-8). pages 31
- [44] Zhang, Z., & Wiklicky, H. (2024, November). Probabilistic Abstract Interpretation on Neural Networks via Grids Approximation. In *Proceedings of 5th Italian Workshop on Explainable Artificial Intelligence, colocated with the 23rd International Conference of the Italian Association for Artificial Intelligence*, Bolzano, Italy. pages 31
- [45] Wiklicky, H. (2024). Probabilistic Programs [Lecture notes]. Imperial College London, Program Analysis (COMP70020). <https://www.doc.ic.ac.uk/~herbert/teaching/PA7P.pdf> pages 11, 12, 13
- [46] Albarghouthi, A. (2021). Introduction to neural network verification. *Foundations and Trends in Programming Languages*, 7(1-2), 1-157. pages 21, 24, 25

-
- [47] Müller, C., Singh, G., Püschel, M., & Vechev, M. T. (2020). Neural network robustness verification on gpus. CoRR, abs/2007.10868. pages 28
- [48] Liu, G., & Marchand, R. (2022). Inference of m-NLP data using radial basis function regression with center-evolving algorithm. Computer Physics Communications, 280, 108497. pages 70
- [49] Weisstein, Eric W. "Fourier Transform." From MathWorld—A Wolfram Web Resource. <https://mathworld.wolfram.com/FourierTransform.html> pages 71
- [50] Zeger, K. (2018). Fourier Transform of a Gaussian [Tutorial sheets]. <https://www.ee.iitb.ac.in/~belur/ee210/current-tutorials-quizzes/Kenneth-Zeger-UCSD-Gaussian-polar-coordinate.pdf> pages 73
- [51] Reynolds, D. (2015). Gaussian mixture models. In Encyclopedia of biometrics (pp. 827-832). Springer, Boston, MA. pages 76
- [52] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30. pages 96