특집원고

심층신경망의 정형검증 기법 소개

포항공과대학교 ▌배경민*

1. 서 론

인공지능 기술의 급격한 발전으로 소프트웨어를 심층신경망(Deep Neural Network, DNN)으로 개발하는 방법이 크게 주목받고 있다. 프로그래밍 언어를 이용하여 개발자가 코드를 작성하는 전통적인 방법과는 다르게, 소프트웨어의 요구되는 행위를 나타내는 데이터로 DNN을 학습시킴으로써, 소프트웨어를 DNN으로 구현하는 방법이다. 소프트웨어의 요구되는 동작을 명확한 규칙으로 표현하기 어려운 음성 인식, 컴퓨터 비전, 자연언어처리 등과 같은 분야에서 소프트웨어를 개발할 때 널리 활용되고 있다[1]. 특히, 자율주행자동차나 무인항공기에 탑재되는 소프트웨어를 DNN으로 개발하기 위한 연구가 활발히 진행 중이다. 이러한 제어 소프트웨어의 경우, 최적의 행위를 명확한 규칙으로 표현하기는 매우 어렵지만, 요구되는 동작에 대한 데이터는 주행이나 비행 기록을 통해서 상대적으로 쉽게 얻을 수 있기 때문이다. Nvidia나 Google Waymo 등의 여러 회사에서 학습을 통한 자율주행자동차의 주행 소프트웨어를 개발 중이며[2, 3], 무인항공기 충돌방지 시스템인 ACAS Xu 소프트웨어가 DNN으로 개발된 바 있다[4].

안전필수 소프트웨어(Safety-Critical Software)는 오류 발생 시 물리적인 사고로 이어져 인명 및 재산 피해를 야기할 수 있는 소프트웨어를 말한다[5]. 자율주행자동차 소프트웨어는 대표적인 안전필수 소프트웨어에 해당한다. 예를 들어, Tesla와 Uber의 자율주행자동차 시험 운행 중 발생한 사고로 2016년 이후 5명의 사망자가 발생하였다. 따라서 안전필수 소프트웨어에는 매우 높은 안전성 기준이 요구되며, IEC 61508 국제 표준은 안전필수 소프트웨어의 오류 빈도가 많아도 수백 년에 한번 이하가 되도록 거의 100%의 안전성을 요구하고 있다.

DNN을 자율주행자동차나 무인항공기의 제어와 같은 안전필수 소프트웨어에 활용하기 위해서는 안전성에 대한 엄밀한 검증이 필요하다. 일반적으로 학습된 DNN의 성능 평가는 학습 데이터와 별도로 주어진 테스트 데이터에 대한 DNN의 실행 결과를 분석함으로써 이루어진다[1]. 그러나 유한한 크기의 데이터셋에 대한 실행 결과의 분석을 통해서는 안전필수 소프트웨어에 요구되는 안전성을 달성하기 어렵다. 높은 안전성을 위해서는 모든 경우에 대하여 소프트웨어가 안전성 기준을 만족하는지 검사하여야 하지만, 모든 경우를 고려하는 완전한 데이터셋을 구성하는 것은 쉽지 않다. 게다가 완전한 데이터셋을 구성하였고 이에 대하여 학습된 DNN이 100%의 안전성 기준을 만족한다 하더라도, DNN이 테스트되지 않은 입력에 대해서도 항상 안전성을 만족하는지는 보장할 수 없다.

학습되지 않은 입력에 의해 발생하는 DNN 오류의 대표적인 예로는 적대적 예제(Adversarial Example)를 들 수 있다[6]. 이는 미세한 차이를 가지는 두 입력에 대하여 DNN이 완전히 다른 결과를 보이는 문제를 말한다. 그림 1은 입력 사진에 미세한 조작을 가하여 DNN이 사진을 잘못 인식하게 하는 경우를 보여준다[6]. 그림 2는 실제 사물에 물리적인 변화를 가하여 DNN이 정지 표지판을 시속 45마일 속도 제한 표지판으로 잘못 인식하게 하는 경우를 보여준다[7]. 이러한 적대적 예제 오류는 DNN이 학습이나 테스트에 사용되는 데이터만으로는 확인할 수 없는 오류를 가질 수 있음을 보여준다.

본 논문에서는 데이터셋에 의존하지 않고 DNN의 안전성을 검증하기 위한 정형검증 기법을 소개한다. 정형검증은 수학적으로 엄밀히 정의된 소프트웨어 모델을 이용하여 요구사항이 만족되는지 검증하는 기술을 말한다. 프로그래밍 언어로 작성되는 소프트웨어를 위해서는 정적분석[8], 모델검증[9], 정리증명[10] 등의 다양한 정형검증 기법이 존재한다. 최근 들어 활

* 종신회원

† 본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2019R1C1C1002386)

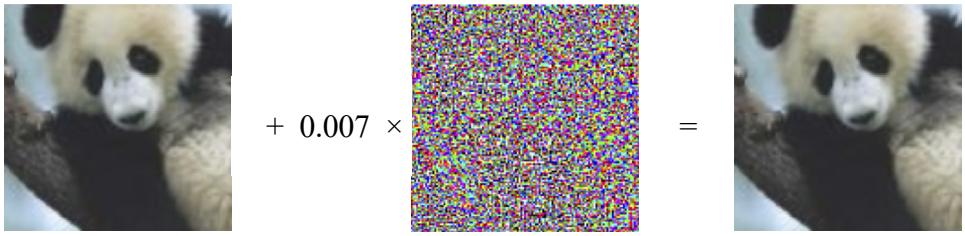


그림 1 Panda로 인식되는 사진에 작은 변화를 가하여 Gibbon으로 인식됨[6].

그림 2 정지 표지판에 물리적인 변화를 가하여 45마일 제한으로 인식됨[7].

발히 연구되고 있는 DNN의 정형검증 기법은, 기존의 데이터셋에 기반한 DNN 검사 방법과는 다르게 DNN에 오류가 존재하지 않음을 보장하는 것이 가능하다. 또한 오류가 존재할 경우 이러한 오류를 발생시키는 데이터를 자동으로 생성할 수 있다.

2. 심층신경망의 요구사항 검증

2.1 DNN의 구조

DNN은 입력층, 출력층, 그리고 여러 개의 은닉층으로 구성되어 있으며, 각 노드는 이전 층의 노드와 특정한 가중치(Weight)를 가지고 연결되어 있다. DNN의 각 노드는 이전 층에서 주어지는 입력값과 각 연결의 가중치, 그리고 편향값(Bias)을 이용하여 출력값을 계산하며, 이 과정을 입력층에서 출력층으로 순차적으로 진행함으로써 최종적으로 DNN의 출력값을 계산하다. 그림 3은 2개의 은닉층을 가진 간단한 DNN의 예를 보여준다. 입력층과 출력층은 2개의 노드를 가지고 있고 각 은닉층은 3개의 노드를 가지고 있다. DNN의 학습은 주어진 데이터셋을 기반으로 각 연결의 가중치와 편향값을 결정하는 과정을 말하며 DNN을 이용한 소프트웨어 개발의 핵심에 해당한다[1].

각 노드의 계산 과정은 입력값과 각 연결의 가중치, 편향값의 선형 결합으로 계산된 값에 활성함수(Activation Function)라 불리는 비선형 함수를 적용함으로써 이루어진다. 즉, 입력값 $x_1, x_2, \ldots, x_n$, 가중치 $w_1, w_2, \ldots, w_n$, 편향값 b, 활성함수 f가 주어졌을 때, 노드의 출력값 y는 다음과 같다.

$$y = f(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b)$$

활성함수에는 목적에 따라 다양한 종류의 함수가 사용되며, Sigmoid, Softmax, ReLU(Rectified Linear Unit) 등을 예로 들 수 있다[1]. 최종적으로 N개의 입력 노드와 M개의 출력 노드를 가지는 DNN은 입력 $\vec{i} = (i_1, i_2, \ldots, i_N)$에 따라 출력 $\vec{o} = (o_1, o_2, \ldots, o_M)$를 계산하는 함수 $\vec{o} = \mathcal{A}(\vec{i})$로 볼 수 있다.

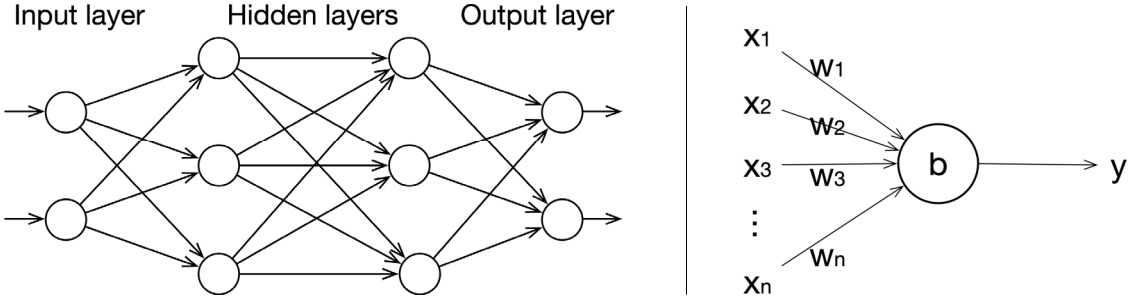


그림 3 DNN 예제(좌) 및 노드의 구조(우)

2.2 DNN의 요구사항 명세

DNN을 입력에 따라 출력을 계산하는 함수 Δ로 볼 때, DNN의 요구사항은 Δ가 만족해야 하는 입력과 출력의 관계로 주어진다. 함수 Δ의 행위가 구체적으로 명확히 정의될 수 있을 경우에는 프로그래밍 언어로 작성하는 것이 가능하기 때문에 기능적인 요구사항에 대한 직접적인 검증이 가능하다. 하지만 학습을 통하여 구현되는 DNN의 경우에는 함수 Δ의 행위를 명확하고 구체적인 언어로 정의하기가 어려운 경우가 많기 때문에, 기존의 소프트웨어와 같이 기능적인 요구사항을 세부적으로 검증하기는 어렵다. 그렇지만 안전성이나 보안성과 같은 비기능적 요구사항은 종종 구체적으로 정의하는 것이 가능하다.

예를 들어, 자동차의 적응형 크루즈(Adaptive Cruise) 제어를 위한 DNN을 생각해 보자. 자동차의 현재 속력 v와 앞차와의 거리 x를 입력으로 받아, 가속 페달과 브레이크 페달에 가해지는 압력 acc와 dec를 출력한다. 그림 3과 같이 2개의 입력노드와 2개의 출력노드를 가진 함수 $(acc, dec) = \Delta(v, x)$로 표현되는 DNN으로 구현되었다고 가정하자. 이 DNN은 데이터를 기반으로 학습되었기 때문에 입력과 출력의 기능적 요구사항을 구체적인 언어로 표현하는 것은 어렵지만, 표 1과 같이 안전성 요구사항을 정의하여 명세하는 것은 가능하다.

다음으로, 이미지를 9개의 클래스 중 하나로 분류하는 DNN을 생각해 보자. 입력 이미지는 N개의 입력값 $i_1,\ldots,i_N$으로 표현되고, 9개의 출력값 $o_1,\ldots,o_9$는 이미지를 각 클래스로 분류할 때의 신뢰도를 나타낸다. 예를 들어, o_1의 값이 $o_2,\ldots,o_9$보다 크면 입력 이미지는 첫 번째 클래스로 분류된다. 이 DNN은 함수 $(o_1,\cdots,o_9) = \Delta(i_1,\cdots,i_N)$으로 표현될 수 있으나, 마찬가지로 입력과 출력의 기능적 요구사항을 구체적으로 표현하는 것은 어렵다. 하지만 적대적 예제를 방지하기 위해 미세한 차이를 가지는 두 이미지가 동일하게 분류되어야 함을 의미하는 강건성(Robustness) 요구사항은 표 2와 같이 정의가 가능하다.

DNN의 안전성 요구사항을 파악하는 과정은 안전필수 소프트웨어를 위한 DNN의 개발에 매우 중요하다. DNN의 학습이나 테스트를 위한 데이터를 구성할 때, 소프트웨어의 안전성을 향상시키기 위해서 이러한 요구사항을 함께 고려하여야 한다. 예를 들어, 이미지 분류를 위한 DNN을 학습할 때 적대적 예제로 인한 오류를 방지하기 위해 고안된 데이터셋의 사용이 권장된다[11]. 각각의 안전성 요구사항은 특정한 경우만을 명세할 수 있기 때문에, 전혀 예상하지 못한 상황에 대해서는 여전히 오류가 발생할 수 있다. 하지만 적대적 예제와 같이 알려진 상황에 대해서는 안전성 요구사항의 명세를 통하여 안전성을 평가하기 위한 기준을 마련할 수 있다.

2.3 DNN의 요구사항 검증

안전성 요구사항을 소프트웨어가 만족하는지 검사하는 방법은 내부 구조의 분석 여부에 따라 크게 블랙박스(Black-box)와 화이트박스(White-box)의 두 가지 방법으로 구분된다. 소프트웨어의 외부 명세를 바탕으로 테스트를 수행하는 블랙박스 테스팅[12]이나 소프트웨어의 실행 중에 요구사항 오류를 검사하는 실시간 검증[13]이 블랙박스 기법에 해당한다. 반면에, 소프트웨어의 코드를 바탕으로 테스트를 수행하는 화이트박스 테스팅[12], 코드를 분석하여 오류 가능성을 파악하는 정적분석[8], 소프트웨어의 수학적인 모델을 기반으로 모든 실행 경로를 분석하는 모델검증[9] 등이 화이트박스 기법에 해당한다.

DNN의 경우, 학습이 완료된 DNN의 성능 평가를 위해서는 보통 블랙박스 기법이 사용된다. 주어진 테

표 1 적응형 크루즈 DNN의 안전성 요구사항 예제.

	안전성 요구 사항	정형 명세
(가)	브레이크 페달과 가속 페달에 동시에 특정값 θ보다 큰 압력이 가해지면 안 된다.	$acc + dec < 2\theta$
(나)	속력의 비해 앞차와의 거리가 비율 α보다 가까우면 브레이크에 c보다 큰 압력이 가해져야 한다.	$\frac{x}{v} < \alpha \Rightarrow dec > c$

표 2 DNN의 강건성 요구사항 예제: $\| \overrightarrow{i_1} - \overrightarrow{i_2} \|_\infty$는 두 입력의 차이, $proj(\overrightarrow{o_1}, i)$는 출력 o_1의 i번째 값, $\arg\max_{i\in\{1,\cdots,9\}}\left(proj(\overrightarrow{o_1}, i)\right)$는 출력 o_1에서 몇 번째 노드가 최대값을 가지는지 나타낸다.

강건성 요구 사항	정형 명세
입력 이미지의 차이가 ϵ보다 작다면 같은 클래스로 분류되어야 한다.	$\| \overrightarrow{i_1} - \overrightarrow{i_2} \|_\infty < \epsilon \Rightarrow$ $\arg\max_{i\in\{1,\cdots,9\}}\left(proj(\overrightarrow{o_1}, i)\right) = \arg\max_{i\in\{1,\cdots,9\}}\left(proj(\overrightarrow{o_2}, i)\right)$

스트 데이터셋에 대하여 각각의 실행 결과를 관측하고 분석하는 방법으로, DNN의 내부 구조에 대한 정보는 필요치 않다. 이 방법은 데이터셋에 포함되는 모든 입력에 대하여 DNN이 안전성 요구사항을 만족하는지 검사하기 위해서 일차적으로 사용될 수 있다. 하지만 위에 언급한 바와 같이, 데이터셋에 포함되지 않은 다른 입력에 대해서는 요구사항을 만족하는지에 대해 아무런 보장을 할 수 없기 때문에, 오류가 심각한 피해를 야기할 수 있는 안전필수 소프트웨어에는 충분하지 않다.

주어진 데이터셋에 의존하지 않고 DNN의 내부 구조를 바탕으로 요구사항을 검사할 수 있는 화이트박스 기법이 최근 활발히 연구되고 있으며, 크게 다음의 두 가지 접근방법이 존재한다. 테스트 생성 기법은 뉴론 커버리지(Neuron Coverage)와 같은 DNN의 내부 구조에서 얻어진 정량적인 지표를 바탕으로 추가적인 테스트 데이터를 생성하여 DNN을 검사한다[14, 15, 16]. 정형검증 기법은 DNN의 수학적인 모델을 바탕으로 DNN이 가능한 모든 입력에 대하여 주어진 요구사항을 만족하는지를 알고리즘을 이용하여 검사한다[17, 18, 19].

DNN의 테스트 생성 기법과 정형검증 기법은 서로 상반되는 장점과 단점을 가지고 있다. DNN의 테스트 생성 기법은 적은 비용으로 수행할 수 있어 규모가 큰 DNN에 쉽게 적용할 수 있으나, 여전히 DNN이 항상 요구사항을 만족하는지는 보장할 수 없다. 반면에, DNN의 정형검증 기법은 DNN이 모든 입력에 대하여 요구사항을 만족하는지 검증할 수 있지만, 높은 계산 복잡도를 가지기 때문에 규모가 큰 DNN에 적용하기 어렵다. 이와 같이 보장성이 높을수록 확장성이 떨어지는 모습은 소프트웨어 검증 기술의 전형적인 특성의 하나이다.

두 가지 종류의 DNN 검증 기술은 목적에 따라 상호보완적으로 사용될 수 있다. 특히, 안전필수 소프트웨어와 같이 요구되는 안전성 기준이 높을수록, 정형검증과 같이 확장성은 떨어지지만 보장성이 높은 기술을 적용하는 것이 중요해진다. 다음 장에서는 본 논문에서 초점을 맞추고 있는 DNN의 정형검증 알고리즘에 대하여 간략히 소개하도록 하겠다.

3. 심층신경망의 정형검증 알고리즘

DNN의 정형검증 알고리즘은 학습이 완료된 DNN과 수학적으로 엄밀하게 명세된 요구사항이 주어졌을 때, 요구사항을 만족하지 않는 입력이 존재하는지 판단하는 알고리즘이다. 보통 오류가 있는 경우에는 이러한 오류를 재현할 수 있는 DNN의 입력을 생성할 수 있다. DNN의 정형검증 알고리즘은 일반적으로 “오류 없음”, “오류 발견”, “판단 불가”의 세 가지 결과를 반환할 수 있으며, “오류 없음”을 반환하였을 경우에는 가능한 모든 입력에 대하여 요구사항이 만족됨이 보장된다. 정형검증 알고리즘 중 “판단 불가”를 반환하지 않는 알고리즘을 완전(Complete)한 알고리즘이라고 한다. 완전한 알고리즘은 오류가 존재한다면 이를 항상 찾을 수 있지만, 보다 높은 계산복잡도를 가지는 단점이 있다. 현존하는 DNN의 정형 검증 알고리즘은 기술적으로 크게 출력범위 계산 기법과 제약조건 해결 기법의 두 가지로 분류할 수 있다.

3.1 출력범위 계산 기반 알고리즘

주어진 입력의 범위에 대하여 가능한 출력의 범위를 각 층 별로 수치적으로 계산하여 요구사항을 어기는 오류의 존재 여부를 파악하는 기술이다. 입력과 출력의 범위는 $A\vec{x} \leq \vec{c}$ 형태의 선형 제약조건으로 주어진다. DNN의 각 노드는 수치적인 함수이기 때문에, 입력의 범위 조건이 주어졌을 때 가능한 출력의 범위 조건을 계산할 수 있다. 출력범위 계산 기법을 적용하기 위해서는, DNN의 요구사항이 입력에 대한 출력의 범위 조건으로 주어져야 한다.

예를 들어, 2.2장에서 설명한 적응형 크루즈 제어 DNN의 안전성 요구사항 (나)를 생각해 보자. 이 경우 DNN은 함수 $(acc, dec) = \Delta(v, x)$, 안전성 요구사항은 수식 $x/v < a \Rightarrow dec > c$으로 표현되어 있으며, 이 때 조건 $x/v < a$는 $x - av < 0$로 바꿔 쓸 수 있다. DNN의 입력 노드는 보통 데이터의 특성에 따라 결정되는 상한값과 하한값 사이의 수를 입력으로 받기 때문에, 이러한 입력 도메인에 대한 정보도 검증 시에 고려되어야 한다. 결과적으로 위의 안전성 검증 문제는 입력범위 $x - av > 0$, $x_{\min} \leq x \leq x_{\max}$, $v_{\min} \leq v \leq v_{\max}$에 대하여 출력범위 $dec > c$가 항상 성립하는지를 묻는 출력범위 계산 문제로 나타낼 수 있다.

N개의 변수에 대한 선형 제약조건은 N차원 공간에서의 다면체로 표현될 수 있기 때문에, 입력에 대한 출력범위의 계산은 기본적으로 다면체의 변환을 계산함으로써 이루어질 수 있다[20]. 예를 들어, 2개의 노드를 가진 층을 생각해 보자. 해당 층은 입력 (x_1, x_2)을 받아서 출력 (y_1, y_2)을 계산하며, 각 노드는 활성함수로 $f(x) = \max(x, 0)$로 정의되는 ReLU를 가진다고 가정하자. 즉, 다음과 같이 입력 (x_1, x_2)에 대하여 가중치와 편향값을 이용하여 중간값 (z_1, z_2)을 계산하고,

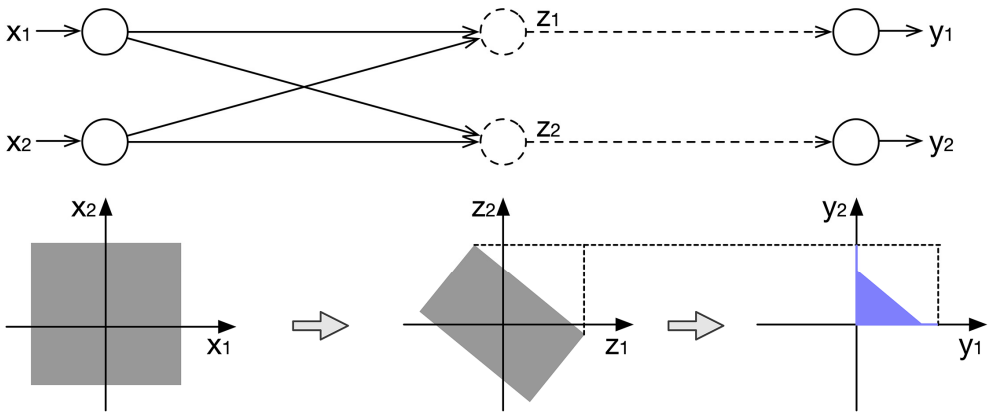


그림 4 DNN의 한 층에 대한 출력범위 계산 예제. 출력 (y_1, y_2)의 범위는 삼각형 영역, 점선 영역 내의 가로축 선분, 그리고 점선 영역 내의 세로축 선분으로 구성된다.

여기에 ReLU 활성함수를 적용하여 출력 (y_1, y_2)를 얻게 된다.

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_{11}\, w_{12} \\ w_{21}\, w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \max(z_1, 0) \\ \max(z_2, 0) \end{bmatrix}$$

그림 4는 DNN의 한 층에 대한 출력범위 계산의 예를 보여준다. 먼저, 입력 (x_1, x_2)의 범위는 그림과 같이 다각형 영역으로 표현된다. 다음으로, 중간값 (z_1, z_2)의 범위는 입력 (x_1, x_2)에 대한 영역에 아핀(Affine) 변환을 적용하여 얻어지는 영역을 나타낸다. 마지막으로, 출력 (y_1, y_2)의 범위는 중간값 (z_1, z_2)에 대한 영역을 ReLU 함수에 따라 분할하여 얻어지는 영역에 해당하게 된다. 이러한 과정을 입력층에서부터 순차적으로 진행함으로써 출력층의 범위를 얻을 수 있다. 최종적으로, 출력층의 범위가 요구사항에서 명세된 출력범위에 포함되는지를 계산함으로써 요구사항이 항상 만족되는지 검증할 수 있다.

출력범위 계산 기법의 주요 과제는 비선형 활성함수를 적용하는 영역 계산을 효과적으로 수행하는 방법이다. 활성함수가 선형 함수의 조합으로 표현되는 경우에는, 주어진 영역을 선형에 해당하는 부분으로 나누어 계산할 수 있다. 예를 들어, ReLU 함수 $\max(x,0)$를 0을 기준으로 나누면 각각의 부분은 선형 함수가 된다. 그림 4에서는 z_1과 z_2의 부호의 따라 4개의 부분으로 나누어 (y_1, y_2)의 영역을 계산하였다. 즉, $z_1 < 0$인 부분에 ReLU를 적용한 영역은 세로축 선분에 해당하며, $z_2 < 0$인 부분에 ReLU를 적용한 영역은 가로축의 선분에 해당한다. 이를 통하여 ReLU의 경우에는 출력범위를 정확하게 계산하여 완전한(Complete) 알고리즘을 얻을 수 있지만[20], 분할을 통해 얻어지는 부분의 수가 DNN의 크기에 따라 기하급수적으로 증가하는 문제가 있다.

따라서 비선형 활성함수를 적용하는 영역 계산을 위해서는 수치적인 근사 기법이 주로 사용된다[18, 19]. 근사 기법은 주어진 입력 범위에 따른 DNN의 실제 출력범위 S를 계산하는 대신에, 실제 범위를 포함하는 근사 범위 $\hat{S} \supseteq S$를 계산한다. 근사 범위 $\hat{S}$가 요구사항의 출력범위에 포함될 경우, 실제 범위 S 또한 해당 범위에 당연히 포함되기 때문에 오류가 없음이 보장된다. 하지만 근사 범위 $\hat{S}$에서 요구사항의 출력범위에 포함되지 않는 오류가 발견될 경우, 실제 범위 S 밖에 존재하는 오탐(False Alarm)일 수 있기 때문에 오류의 존재여부를 정확히 판단할 수 없다. 비록 근사 기법은 이처럼 "판단 불가"를 반환할 수 있어 완전하지 않지만, 근사 기법을 통하여 알고리즘의 효율성을 크게 향상시킬 수 있으며, 시그모이드(Sigmoid) 함수 등과 같은 초월 함수를 활성함수로 사용하는 DNN도 검증이 가능하다.

3.2 제약조건 해결 기반 알고리즘

DNN의 요구사항 검증 문제를 논리적 제약조건 해결 문제로 변환하고 이를 자동추론이나 최적화 알고리즘을 이용하여 해결하는 방법이다. 간단히 말하면, 요구사항의 오류를 표현하는 논리식 Φ_{Err}을 만들고 이를 통해서 오류의 존재 여부를 계산한다. DNN의 입력값과 출력값의 관계를 정의하는 논리식 Φ_{Err}는 요구사항이 만족될 때 거짓이 되고, 요구사항이 만족되지 않을 때 참이 된다. DNN이 함수 $\vec{o} = \mathcal{A}(\vec{i})$로 주어졌을 때, DNN의 오류를 찾는 문제는 제약조건 $\vec{o} = \mathcal{A}(\vec{i})$와 $\Phi_{Err}(\vec{i}, \vec{o})$를 동시에 만족하는 입력 $\vec{i}$과 출력 $\vec{o}$를 찾는 제약조건 해결 문제로 표현될 수 있다. 마찬가지로 여러 개의 입력과 출력을 포함하는 요

구사항에 대한 검증 문제 또한 제약조건 해결 문제로 표현될 수 있다.

2.2장에서 보인 이미지 분류 DNN의 강건성 요구사항에 대한 검증 문제를 예로 들도록 하겠다. 설명의 간략화를 위하여 입력 노드와 출력 노드가 모두 2개라고 가정하자. 입력 (i_1, i_2)와 (j_1, j_2)에 대한 DNN의 출력을 각각 (o_1, o_2)와 (p_1, p_2)라고 할 때, 강건성 요구사항의 오류는 (i_1, i_2)과 (j_1, j_2)가 미세한 차이를 가짐에도 불구하고 다르게 분류되는 경우에 해당하며, 다음과 같이 논리식 Φ_{Err}로 표현될 수 있다. 논리연산자 $\wedge$와 $\vee$는 각각 and와 or를 의미한다.

$$|i_1 - j_1| < \epsilon \ \wedge \ |i_2 - j_2| < \epsilon \ \wedge$$
$$(o_1 < o_2 \vee p_1 < p_2) \ \wedge \ (o_1 > o_2 \vee p_1 > p_2)$$

그러면 오류를 찾는 문제는 제약조건 Φ_{Err}, $(o_1,o_2) = \Delta(i_1,i_2)$, $(p_1,p_2) = \Delta(j_1,j_2)$를 동시에 만족하는 i_1, i_2, j_1, j_2를 찾는 문제로 변환될 수 있다.

제약조건을 해결하는 방법으로, 먼저 SMT(Satisfiability Modulo Theoreis) 기반 기술은 논리적인 추론을 이용하여 제약조건 문제를 해결하며[17, 21, 22], 선형 제약조건의 논리적인 조합으로 표현되는 비선형 제약조건을 해결할 수 있다. 예를 들어, 비선형 제약조건 $y = \max(z,0)$은 $(z \geq 0 \wedge y = z) \vee (z < 0 \wedge y = 0)$으로 표현이 가능하며, 따라서 ReLU를 활성함수로 가지는 DNN의 노드는 그림 5와 같이 논리식으로 인코딩(Encoding)이 가능하다. SMT 기반 알고리즘은 다른 기법에 비하여 다양한 종류의 요구사항을 검증할 수 있고, 오류가 존재할 경우 항상 찾을 수 있는 완전한(Complete) 알고리즘을 제공한다. 그러나 시그모이드와 같은 초월함수를 포함하는 제약조건은 효과적으로 다루기 어렵고, 높은 계산복잡도를 가져서 규모가 큰 DNN에 적용하기 어렵다.

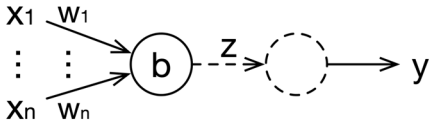
다음으로, 제약조건을 비선형 최적화 문제로 변환하여 해결하는 최적화 기반 방법이 있다[23, 24, 25]. 요구사항 오류 논리식 Φ_{Err}을 실수를 반환하는 목적함수(Objective Function) f_{Err}로 변환하고, 이의 극대값이나 극소값을 구함으로써 제약조건을 해결한다. 예를 들어, 논리식 Φ_{Err}가 참이면 f_{Err}는 양수를 반환하고, Φ_{Err}가 거짓이면 f_{Err}는 음수를 반환한다고 하자. 이 때, DNN의 제약조건 $\vec{o} = \Delta(\vec{i})$을 만족하면서 목적함수 f_{Err}를 최대화하는 입력 $\vec{i}$를 찾는 최적화 문제를 생각해 보자. 목적함수 f_{Err}의 최대값이 양수이면 요구사항을 만족하지 않는 오류를 찾게 되고, f_{Err}의 최대값이 음수이면 요구사항을 만족하지 않는 오류가 존재하지 않음을 증명하게 된다.

최적화 기반 기법의 주요 과제는 DNN 제약조건에 포함되는 비선형 함수를 효과적으로 다루는 데 있다. ReLU만을 활성함수로 가지는 DNN의 경우에는 정수 제약조건을 함께 사용하여 혼합 정수 선형 계획법(Mixed Integer Linear Programming)으로 해결이 가능하다[23, 24]. 시그모이드와 같은 활성함수를 포함하는 DNN의 경우, 라그랑주 완화법(Lagrangian Relaxation)을 활용하여 근사적으로 문제를 해결할 수 있다[25]. 최적화 기반 알고리즘은 순수한 SMT 기반 기법에 비해 높은 성능을 보이지만, 완전성을 희생하여 오류가 존재한다 하더라도 항상 찾을 수는 없는 경우가 많다. 또한 보통 DNN에 특화된 SMT 기반 기술은 최적화 알고리즘과 함께 사용된다.

4. 심층신경망 정형 검증의 전망

DNN의 정형검증 기술은, 자율주행자동차와 같은 안전필수 소프트웨어에서 DNN의 응용 범위가 확대되고 적대적 예제와 같은 오류 사례가 보고됨에 따라, 최근 5년 사이에 매우 활발하게 연구되고 있다. 아직은 프로그램 분석, 모델검증, 최적화 기법, 수치해석 등과 같은 기존의 다양한 분석 기술을 DNN 검증에 적용해 보는, 기술 개발의 초기 단계에 해당한다. 특히, DNN의 오류 중 가장 널리 알려져 있는 적대적 예제에 대한 강건성 요구사항을 검증하기 위한 기술을 중심으로 연구가 많이 진행되고 있다.

DNN 정형검증 기술의 당면 과제는 규모가 큰 DNN에도 효과적으로 적용이 가능한 확장성(Scalability)이 높은 기술을 개발하는 것이다. 이는 프로그래밍 언어로 작성되는 기존의 소프트웨어 검증 기술의 주요 과제 또한 확장성인 경우와 관련이 깊다고 볼 수 있다. DNN의 요구사항을 검증하는 문제의 계산복잡도는



$$(z = w_1 x_1 + \cdots + w_n x_n + b) \ \wedge$$
$$\big((z \geq 0 \wedge y = z) \vee (z < 0 \wedge y = 0)\big)$$

그림 5 ReLU 노드의 SMT 기반 인코딩(Encoding) 방법

이론적으로 NP-Hard이기 때문에[17], 완전성(Completeness)과 확장성을 동시에 100% 달성하기는 어렵다. 따라서 3장에서 설명한 바와 같이 완전성을 어느 정도 희생함으로써 확장성을 향상시키고, 이 때 "판단 불가" 결과를 반환하는 경우를 최소화 하는 기술을 개발하는 방향으로 연구가 많이 진행 중에 있다.

DNN의 정형검증 기법과 병행하여, DNN 자체의 검증 복잡도를 줄일 수 있는 기술에 대한 연구가 필요하다. 현재까지의 DNN 정형검증은 학습이 완료된 DNN이 주어진 요구사항을 만족하는지 검증하는 문제에 초점을 맞추고 있다. 하지만 기존의 소프트웨어 검증 기술의 사례에서 볼 수 있는 것처럼[9], 기술 개발의 발전 속도에 비해서 검증 대상 DNN의 크기와 복잡도가 급격히 증가할 경우 넓은 범위의 소프트웨어에 적용하기 어렵다.

DNN의 검증 복잡도를 줄일 수 있는 기법으로, 다음의 두 가지 접근방법을 예로 들도록 하겠다. 먼저, 검증 복잡도가 낮은 DNN의 구조에 대한 연구가 있다. 3장에서 보인 것처럼, ReLU와 같은 단순한 활성함수를 가지는 DNN이 검증 복잡도가 낮다. 또한, Binarized Neural Network[26]은 계산의 성능과 에너지 효율을 향상시키기 위하여 제안되었지만, 검증도 효과적으로 수행될 수 있음이 알려져 있다[26, 27]. 다음으로, DNN을 동일한 요구사항을 만족하며 검증 복잡도가 낮은 DNN으로 변환하는 연구가 있다[28, 29]. 예를 들어, 주어진 DNN을 보다 적은 수의 노드를 가지거나, 보다 단순한 가중치와 활성함수를 갖는 DNN으로 변환함으로써, 검증을 보다 효과적으로 수행할 수 있다. 이 때, 변환 전과 후의 DNN이 동일한 요구사항을 만족함이 보장되어야 한다.

5. 결 론

DNN의 정형검증 기법은 DNN이 가능한 모든 입력에 대하여 항상 안전성 요구사항을 만족하는지 엄밀하게 검증하는 기술이다. 본 논문에서는 현존하는 DNN의 정형검증 알고리즘의 종류와 특징을 간략하게 정리하여 소개하였다. 유한한 크기의 데이터셋에 대한 실행결과를 관측하여 성능을 평가하는 테스트 기반 기법과는 다르게, 정형검증 기법은 데이터셋에 의존하지 않고 수학적인 모델을 바탕으로 안전성 요구사항을 검증하기 때문에 오류가 존재하지 않음을 보장하는 것이 가능하다. 따라서 DNN의 정형검증 기술은 자율주행자동차와 같은 안전필수 소프트웨어에 DNN을 활용하기 위한 필수 기술에 해당한다. 하지만 이와 같은 강력한 검증 능력은 높은 계산복잡도를 필요로 하며, 규모가 큰 DNN에도 효과적으로 적용이 가능한 확장성이 높은 기술을 개발하는 것이 DNN 정형검증 기술의 당면 과제이다.

참고문헌

[1] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[2] M. Bojarski et al., "End to end learning for self-driving cars," arXiv preprint arXiv:1604.07316, 2016.

[3] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," arXiv preprint arXiv:1812.03079, 2018.

[4] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," Proc. DASC, IEEE, 2016.

[5] N. R. Storey, Safety critical computer systems. Addison-Wesley Longman Publishing Co., Inc., 1996.

[6] C. Szegedy et al., "Intriguing properties of neural networks," Proc. ICLR, 2014.

[7] I. Evtimov et al., "Robust physical-world attacks on deep learning models," arXiv preprint arXiv:1707.08945, 2017.

[8] X. Rival and K. Yi, Introduction to Static Analysis: An Abstract Interpretation Perspective. Mit Press, 2020.

[9] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, Model checking. MIT press, 2018.

[10] A. R. Bradley and Z. Manna, The calculus of computation: decision procedures with applications to verification. Springer, 2007.

[11] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," Communications of the ACM, vol. 61, no. 7, pp. 56-66, 2018.

[12] P. Ammann and J. Offutt, Introduction to software testing. Cambridge University Press, 2016.

[13] M. Leucker and C. Schallhart, "A brief account of runtime verification," The Journal of Logic and Algebraic Programming, vol. 78, no. 5, pp. 293-303, 2009.

[14] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," Proc. SOSP, ACM, 2017, pp. 1-18.

[15] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," Proc. ASE, ACM, 2018, pp. 109-119.

[16] J. Kim, R. Feldt, and S. Yoo, “Guiding deep learning system testing using surprise adequacy,” Proc. ICSE, IEEE, 2019.
[17] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, “Reluplex: An efficient SMT solver for verifying deep neural networks,” Proc. CAV, pp. 97-117.
[18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, “AI2: Safety and robustness certification of neural networks with abstract interpretation,” Proc. S&P, IEEE, 2018, pp. 3-18.
[19] G. Singh, T. Gehr, M. Püschel, and M. Vechev, “An abstract domain for certifying neural networks,” Proceedings of the ACM on Programming Languages, vol. 3, no. POPL, pp. 1-30, 2019.
[20] W. Xiang, H.-D. Tran, J. A. Rosenfeld, and T. T. Johnson, “Reachable set estimation and safety verification for piecewise linear systems with neural network controllers,” Proc. ACC, IEEE, 2018, pp. 1574-1579.
[21] L. Pulina and A. Tacchella, “Challenging SMT solvers to verify neural networks,” AI Communications, vol. 25, no. 2, pp. 117-135, 2012.
[22] R. Ehlers, “Formal verification of piece-wise linear feed-forward neural networks,” Proc. ATVA, 2017, pp. 269-286.
[23] V. Tjeng, K. Xiao, and R. Tedrake, “Evaluating robustness of neural networks with mixed integer programming,” arXiv preprint arXiv:1711.07356, 2017.
[24] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda, “A unified view of piecewise linear neural network verification,” Proc. NeurIPS, 2018, pp. 4790-4799.
[25] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, “A Dual Approach to Scalable Verification of Deep Networks,” Proc. UAI, 2018, vol. 1, p. 2.
[26] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” Proc. NeurIPS, 2016, pp. 4107-4115.
[27] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, “Verifying properties of binarized deep neural networks,” Proc. AAAI, 2018.
[28] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” arXiv preprint arXiv:1510.00149, 2015.
[29] D. Shriver, D. Xu, S. Elbaum, and M. B. Dwyer, “Refactoring Neural Networks for Verification,” arXiv preprint arXiv:1908.08026, 2019.

약 력

배 경 민

2004 KAIST 전산학부 졸업(학사)
2014 미국 Univ. of Illinois at Urbana-Champaign 졸업(박사)
2014~2015 미국Carnegie Mellon University, Postdoctoral Fellow
2015~2016 미국SRI International, Postdoctoral Fellow
2016~현재 포항공과대학교 컴퓨터공학과 조교수
관심분야 : 정형기법, 모델검증, 자동추론
Email : kmbae@postech.ac.kr