Introduction to Neural Network Verification

Konkuk University
Department of Computer Science & Engineering
Kunha Kim

The Rise of Deep Learning

- Modern neural networks are called deep neural networks, and the approach to training these neural networks is deep learning.
- Deep learning has enabled incred- ible improvements in complex computing tasks, most notably in computer vision and natural-language processing.
- For example, in recognizing objects and people in an image and translating between languages.
- As deep learning becomes used more and more in sensitive settings, like autonomous cars, it is imperative that we verify these systems and provide formal guarantees on their behavior. But what exactly do we verify?

Some backgrounds

- Functional correctness, meaning that a program is a faithful implementation of some mathematical function. This is incredibly important in many settings like aircraft controller.

- In the land of deep learning, proving functional correctness is an unrealistic task.

- What does it mean to correctly recognize cats in an image or correctly trans- late English to Hindi? We cannot mathematically define such tasks.

- The whole point of using deep learning to do tasks like translation or image recognition is because we cannot mathematically capture what exactly they entail.

Properties to ask

- The most-studied correctness property of neural networks is robustness, means that small perturbations to inputs should not result in changes to the output of the neural network.

- Safety is a broad class of correctness properties stipulating that a program should not get to a bad state. The definition of bad depends on the task at hand.

- Neural networks learn about our world via examples, like images. As such, they may sometimes miss basic axioms, like physical laws, and assumptions about realistic scenarios.

- For instance, a neural network recognizing objects in an image and their relationships might say that object A is on top of object B, B is on top of C, and C is on top of A.

Neural Networks

- 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 $\mathbb{R}^n \rightarrow \mathbb{R}^m$.
- For example, the red node is an input node; it just passes input x, a real number to node v.
- v performs some operation on x and spits out a value that goes to the output node y.
- For example, v might simply return $2x + 1$, a function $f_v : \mathbb{R} \rightarrow \mathbb{R}, f_v(x) = 2x + 1$.
- Output node may perform some operation, like $f_y(x) = max(0,x)$.
- So, the simple neural network is $f : \mathbb{R} \rightarrow \mathbb{R} : f(x) = f_y(f_v(x)) = max(0,2x + 1)$.

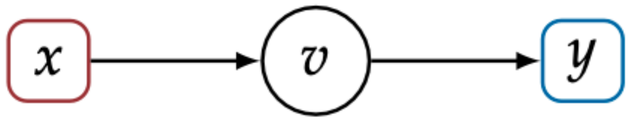


Figure 2.1 A very simple neural network

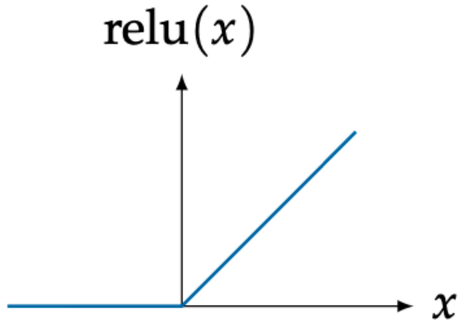


Figure 2.2 Rectified linear unit

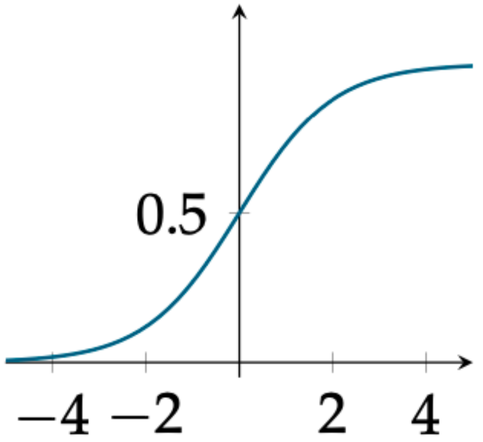


Figure 2.3 Sigmoid function

Transformation and Activations

- The function f_y is an $activation$ function, because it turns on or off depending on its input. Specifically, it is called ReLU.
- There are other popular activation functions like sigmoid, $\sigma(x) = 1/1 + exp(-x)$.
- Often, the affine functions and the activation function are composed into a single operation. But for simplicity, here we separate them.
- Activation functions are used to add non-linearity into a neural network.
- It is also very interesting to point out that you can construct a neural network comprised of ReLUs or sigmoids and affine functions to approximate any continuous function. This is called Universal Approximation Theorem.

MLP and Softmax

- The nodes of the graph form layers, the input layer, the output layer, and the layer in the middle which is called the *hidden* layer.
- Figure 2.5 shows a *multilayer perceptron* with two hidden layers.
- Neural networks are typically used as classifiers : they take an input, and predict what the image is about(image's class).
- To normalize the output, we use *softmax* function, denoted as $f_{y_i}(x1, \cdots, x_n) = exp(x_i)/\Sigma_{k=1}^{n} exp(x_k)$.

CNN and RNN

- Another kind of layer that you will find in a neural network is a convolutional layer. This kind of layer is widely used in computer-vision tasks, but also has uses in natural-language processing.

- The convolutional layer gives you that: it defines an operation, a kernel, that is applied to every region of pixels in an image or every sequence of words in a sentence.

- RNN is the canonical neural network having loop. It is usually used for sequence data, like text.

- Both of them are trying to handle spatial information.

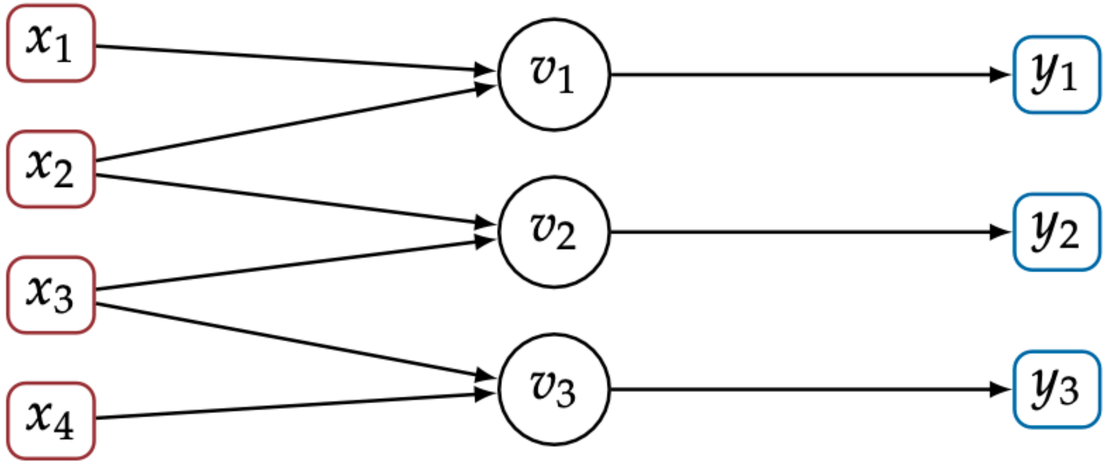


Figure 2.6 1-dimensional convolution

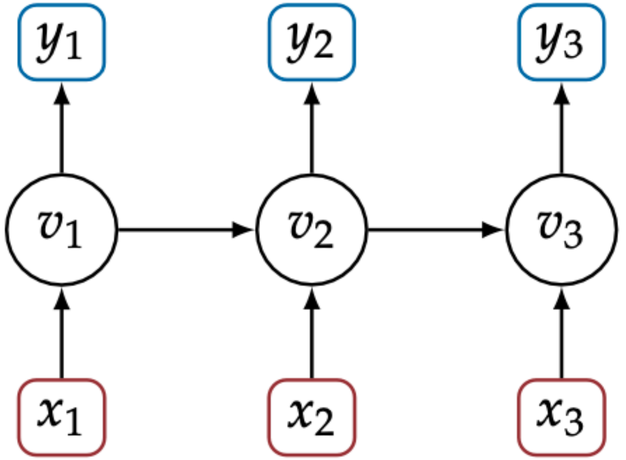


Figure 2.8 Unrolled recurrent neural network

Neural Networks as DAGs

- A neural network is a directed acyclic graph $G = (V, E)$, where
- V is a finite set of nodes,
- $E \subseteq V \times V$ is a set of edges,
- $V^{in} \subset V$ is a non-empty set of input nodes
- $V^{o} \subset V$ is a non-empty set of output nodes, and
- Each non-input node v is associated with a function $f_v : \mathbb{R}^{n_v} \to \mathbb{R}$, where n_v is the number of edges whose target is node v.
- Notice that we assume for simplicity but without loss of generality, that a node v only outputs a single real value.
- Also, we assume every nodes are reachable, via directed edges, every node can reach an output node and there is fixed total ordering on edges E and another one on nodes V.

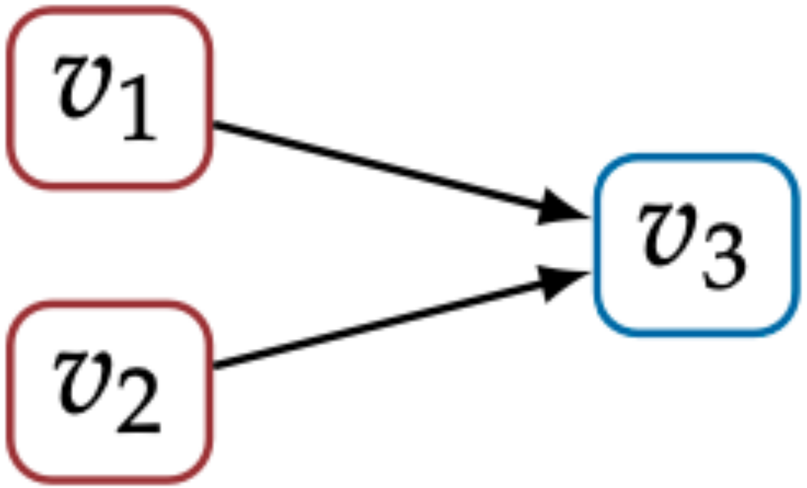
Semantics of DAGs and Example

- A neural network $G = (V, E)$ defines a function in $\mathbb{R}^n \to \mathbb{R}^m$ where $n = |V^{in}|$ and $m = |V^o|$.

- For every non-input node $v \in V$, we define the output of node v as out(v) $= f_v(x_1, \cdots, x_{n_v})$ where $x_i =$ out(v_i) for $i \in \{1, \cdots, n_v\}$.

- Let's see some simple example. A graph G has $V^{in} = \{v_1, v_2\}$, $V^o = \{v_3\}$ and $f_{v_3}(x_1, x_2) = x_1 + x_2$. We have input vector (11, 79), then out(v_1)= 11, out(v_2) = 79 and out(v_3) $= f_{v_3}$(out(v_1), out(v_2)) = 11+ 79 = 90.

- We can compute values out($\cdot$) in any *topological* ordering.


v_1
v_2
v_3

Properties of Functions

- So far, we have assumed that a node v can implement any function f_v it wants over real numbers.
- In practice, to enable efficient training of neural networks, these functions need be differentiable or differentiable almost everywhere.
- The two important property of activation function are $linear$ or $piecewise\ linear$ and $monotonicity$.
- ReLU and sigmoid satisfy these two property.

Correctness Properties

- Remember that a neural network defines a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The properties we will consider here are of the from : for any input x, the neural network produces an output that..
- In other words, properties dictate the input-output behavior, but not the internals of the network.
- The first part of these properties, the one talking about inputs, is called the*precondition*; the second part, talking about outputs, is called the *postcondition*.
- Our specifications are look like as follow :

$$\{\ precondition\ \}$$
$$\boldsymbol{r_1} \leftarrow f(\boldsymbol{x_1})$$
$$\boldsymbol{r_2} \leftarrow g(\boldsymbol{x_2})$$
$$\vdots$$
$$\{\ postcondition\ \}$$

Example : Image Recognition

- Say c is an actual grayscale image, where each element of c is the intensity of a pixel, from 0 to 1.
- We want to check some image x, which is slightly bright or dark than c, class($f(x)$) = class ($f(c)$).
- The precondition can specified as $|x - c| \leq 0.1$, assignments are $r_1 \leftarrow f(x)$ and $r_2 \leftarrow f(c)$, and the postcondition is class(r_1) = class(r_2).
- The full statement is denoted as follows :

$$\{\ |x - c| \leqslant 0.1\ \}$$
$$r_1 \leftarrow f(x)$$
$$r_2 \leftarrow f(c)$$
$$\{\ \text{class}(r_1) = \text{class}(r_2)\ \}$$

Example : Equivalence of Neural Networks

- Say you have a neural network f for image recognition and you want to replace it with a new neural network g.
- You might think f and g should provide same output for every possible input, but that's too rigid.
- One possible alternative is to state that f and g return the same prediction on some subset of images, plus or minus some brightness, as in our above example.
- Say S is a finite set of images, then our specification denoted as right side of the next slide :

$$\{\ \text{true}\ \}$$
$$r_1 \leftarrow f(x)$$
$$r_2 \leftarrow g(x)$$
$$\{\ \text{class}(r_1) = \text{class}(r_2)\ \}$$

$$\{\ x_1 \in S,\ |x_1 - x_3| \leqslant \mathbf{0.1},\ |x_1 - x_2| \leqslant \mathbf{0.1}\ \}$$
$$r_1 \leftarrow f(x_2)$$
$$r_2 \leftarrow g(x_3)$$
$$\{\ \text{class}(r_1) = \text{class}(r_2)\ \}$$

Propositional Logic

- A formula F in propositional logic is over Boolean variables, and conjunction, disjunction or negation of the formula.
- Let $f_v(F)$ be a set of variables of F. An interpretation I of F is a map from variables $f_v(F)$ to true or false.
- An evaluation rule is a rule to simplify the formula like $F \vee \mathit{false} \equiv F$, $F \vee \mathit{true} \equiv \mathit{true}$, $F \wedge \mathit{false} \equiv \mathit{false}$, $F \wedge \mathit{true} \equiv F$.
- A formula F is satisfiable if there exists an interpretation I such that $\mathit{eval}(I(F)) = \mathit{true}$. For this case, we will say I is a model of F and denote it $I \vDash F$.
- We wil also use $I \nvDash F$ to denote that I is not a model of F.

Validity and Equivalence

- To prove properties of neural networks, we will be asking validity questions. A formula F is valid if every possible interpretation I is a model of F.
- For example, formula $F \triangleq (\neg p \vee \neg q) \vee p$ is valid.
- Two formulas A and B are $equivalent$ if and only if every model I of A is a model of B, and vice versa. We will denote equivalence as $A \equiv B$.
- An $implication$ $A \Rightarrow B$ is equal to $\neg A \vee B$.

Linear Real Arithmetic

- In LRA, each propositional variable is replaced by a linear inequality of the form $\Sigma_{i=1}^{n} c_i x_i + b < 0$ or $\Sigma_{i=1}^{n} c_i x_i + b \leq 0$. And = and $\neq$ can be replaced by inequality.
- All notations that we already mentioned in propositional logic is same in LRA. There is only one difference that interpretation assigns real value, not the true or false.
- In the literature, you might find LRA being referred to as linear rational arithmetic. There are two interrelated reasons for that:
- First, whenever we write formulas in practice, the constants in those formulas are rational values—we can't really represent π, for instance, in computer memory.
- Second, let's say that F contains only rational coefficients. Then, it follows that, if F is satisfiable, there is a model of F that assigns all free variables to rational values.

Non-linear Arithmetic and MILP

- Deciding satisfiability of formulas in LRA is an NP-complete problem. If we extend our theory to allow for polynomial inequalities, then the best known algorithms are doubly exponential in the size of the formula in the worst case (Caviness and Johnson, 2012).

- If we allow for transcedental functions—like exp, cos, log, etc.— then satisfiability becomes undecidable (Tarski, 1998).

- Thus, for all practical purposes, we stick to LRA. Even though it is NP-complete (a term that sends shivers down the spines of theoreticians), we have very efficient algorithms that can scale to large formulas.

- Formulas in LRA, and the SMT problem for LRA, is equivalent to the mixed integer linear programming (MILP) problem. Why we use LRA and SMT Solver?

- First, in practice, neural networks do not operate over real or rational arithmetic. They run using floating point, fixed point, or machine-integer arithmetic.

- Second, as we move forward and neural networks start showing up every- where, we do not want to verify them in isolation, but in conjunction with other pieces of code that the neural network interacts with.

Encodings of Neural Networks

- To solve a problem with SMT Solver, we have to model a neural network into a formula in LRA.
- Recall that a neural network is represented as a graph G that defines a function $f_G : \mathbb{R}^n \to \mathbb{R}^m$. We define the input-output relation of f_G as the binary relation $R_G = \{(a, b) \,|\, a \in \mathbb{R}^n, b = f_G(a)\}$.
- We will similarly use R_v to define the input-output relation of the function f_v of a single node v in G.
- For example, if $f_G(x) = x + 1$, then $R_G = \{(a, a + 1) \,|\, a \in \mathbb{R}\}$.

Encoding Nodes

- Consider $f_v : \mathbb{R} \to \mathbb{R}$, the case of single input and single output. Let $f_v(x) = x + 1$.
- Then, $R_v = \{(a, a+1) \,|\, a \in \mathbb{R}\}$ and we can change it into linear inequality : $F_v \triangleq v^o = v^{in,1} + 1$ where v^o denotes the output of node v and $v^{in,1}$ denotes the first input of node v.
- For $f_v(x) = x_1 + 1.5x_2$, $F_v \triangleq v^o = v^{in,1} + 1.5v^{in,2}$.
- Now, let's assume $f_v : \mathbb{R}^{n_v} \to \mathbb{R}$ is piecewise-linear, i.e. of the form $f(x) = \Sigma_j c_j^1 \cdot x_j + b1 \ \ if \ S_1, \cdots, \Sigma_j c_j^l \cdot x_j + bl \ \ if \ S_l$ where j ranges from 1 to n_v.
- Then $F_v \triangleq \wedge_{i=1}^{l} [S_i \Rightarrow (v^o = \Sigma_{j=1}^{n_v} c_j^i \cdot v^{in,j} + b_j)]$. Let's see some simple example, ReLU.
- $relu(x) = x \ \ if \ x > 0$, otherwise 0 can be encoded as $F_v \triangleq (v^{in,1} > 0 \Rightarrow v^o = v^{in,1}) \wedge (v^{in,1} \leq 0 \Rightarrow v^o = 0)$

Soundness and Completeness

- The above encoding precisely captures the semantics of a piecewise-linear node.
- Let's formally capture this fact: Fix some node v with a piecewise-linear function f_v. Let F_v be its encoding, as defined above.
- First, our encoding is *sound*. Informally, soundness means that our encoding does not miss any behavior of f_v. Formally, let $(a, b) \in R_v$ and let $I = \{v^{in,1} \mapsto a_1, \cdots, v^{in,n} \mapsto a_n, v^o \mapsto b\}$. Then $I \models F$.
- Second, our encoding is *complete*. Informally, completeness means that our encoding is tight, or does not introduce new behaviors not exhibited by f_v.
- Formally, let the following be a model of F_v : $I = \{v^{in,1} \mapsto a_1, \cdots, v^{in,n} \mapsto a_n, v^o \mapsto b\}$. Then $(a, b) \in R_v$.