Review: neural networks

Linear layers: \( z^{(1)} = W^{(1)} x \quad z^{(2)} = W^{(2)} \hat{z}^{(1)} \quad y = w^{(3)T} \hat{z}^{(2)} \)

Nonlinear layers: \( \hat{z}_j^{(i)} = \sigma(z_j^{(i)}) \) (assume \( \sigma \) is ReLU for now)
Review: neural network verification as a satisfiability problem

\[ \exists x \in S \land y \leq 0 \land y = f(x) \]

Input domain under consideration

Negation of the desired property

Defines the neural network
Review: neural network verification as a satisfiability problem

Satisfiability problem: \( \exists x \in S \land y \leq 0 \land y = f(x) \)

\[ x_i \leq u_i \land x_i \geq l_i \quad \text{(assuming box constraints)} \]

\[ ((z^{(i)} \geq 0 \land \hat{z}^{(i)} = z^{(i)}) \lor (z^{(i)} < 0 \land \hat{z}^{(i)} = 0)) \quad \text{for each ReLU neuron} \]

\[ z_1 = W^{(1)} x \land z^{(2)} = W^{(2)} \hat{z}^{(1)} \land y = w^{(3)^T} \hat{z}^{(2)} \land y \leq 0 \]

Add all clauses to the formula and solve using DPLL(T) with \textbf{Linear Real Arithmetic}.

In general this is very slow! Faster methods in the next a few lectures.
How does a SMT solver solve this problem?

First step: obtain the abstract version of the problem

\[ x_i \leq u_i \land x_i \geq l_i \] (assuming box constraints)

\[ ((z_j^{(i)}) \geq 0 \land \hat{z}_j^{(i)} = z_j^{(i)}) \lor (z_j^{(i)} < 0 \land \hat{z}_j^{(i)} = 0)) \] for each ReLU neuron

\[ z_1 = W^{(1)} x \land z^{(2)} = W^{(2)} \hat{z}^{(1)} \land y = w^{(3)^T} \hat{z}^{(2)} \land y \leq 0 \]
How does a SMT solver solve this problem?

Obtain the abstract version of the problem

\[\begin{align*}
x_i & \leq u_i \land x_i \geq l_i \quad \text{(assuming box constraints)} \\
((z_j^{(i)} \geq 0 \land \hat{z}_j^{(i)} = z_j^{(i)}) \lor (z_j^{(i)} < 0 \land \hat{z}_j^{(i)} = 0)) & \quad \text{for each ReLU neuron} \\
z_1 &= W^{(1)} x \land z^{(2)} = W^{(2)} \hat{z}^{(1)} \land y = w^{(3)\top} \hat{z}^{(2)} \land y \leq 0
\end{align*}\]

For each ReLU neuron: \(((p_j^{(i)} \land q_j^{(i)}) \lor (\neg p_j^{(i)} \land r_j^{(i)}))\)

All other clauses contain only a single literal, and must be set to True
How does a SMT solver solves this problem?

Now convert to CNF form

For each ReLU neuron: \((p_j(i) \land q_j(i)) \lor (\neg p_j(i) \land r_j(i))\)

Distribution: \(((p_j(i) \lor \neg p_j(i)) \land (p_j(i) \lor r_j(i)) \land (q_j(i) \lor \neg p_j(i)) \land (q_j(i) \lor r_j(i))\))

Rewrite: \((p_j(i) \lor r_j(i)) \land (\neg p_j(i) \lor q_j(i)) \land (q_j(i) \lor r_j(i))\))

Observe that when \(p_j(i) = \text{True}\), \(q_j(i)\) must be true; \(p_j(i) = \text{False}\), \(r_j(i)\) must be true;

So the clause \(q_j(i) \lor r_j(i)\) is redundant
How does a SMT solver solves this problem?

\((p_j(i) \lor r_j(i)) \land (\neg p_j(i) \lor q_j(i))\)

When \(p_j(i)=\text{True}\), \(q_j(i)\) must be true; \(p_j(i)=\text{False}\), \(r_j(i)\) must be true;

SAT solver must try both cases of \(p_j(i)\)

\(p_j(i)=\text{True} \Rightarrow (z_j(i) \geq 0)\)

\(p_j(i)=\text{False} \Rightarrow (z_j(i) < 0)\)
Why using a SMT solver is very slow?

\[(p_j^{(i)} \lor r_j^{(i)}) \land (\neg p_j^{(i)} \lor q_j^{(i)})\]

When \(p_j^{(i)}=\text{True}\), \(q_j^{(i)}\) must be true; \(p_j^{(i)}=\text{False}\), \(r_j^{(i)}\) must be true;

SAT solver must try both cases of \(p_j^{(i)}\). In a satisfiable solution from DPLL, each \(p_j^{(i)}\) is set to True or False, then a theory solver (Simplex) invoked.

There are exponential number of cases here… and modern neural networks can have millions of neurons!

Can we solve the problem without setting every \(p_j^{(i)}\)?

We will tack this problem from an **optimization** point of view today.
Mathematical optimization problems

Given a **objective function** $f: S \rightarrow \mathbb{R}$

Seek an **optimal solution** $x^*$ such that $f(x^*) \leq f(x)$ for all $x \in S$
Some optimization problems are hard, some are easy

Given a **objective function** \( f: S \rightarrow \mathbb{R} \)

Hardness depends on the properties of \( f \) and \( S \). For tractable solving they cannot be arbitrary!

- **Easy ones**: convex optimization, linear programming, semidefinite programming
  - E.g., in linear programming, objective function and constraints must be linear functions

- **Hard ones**: integer programming, general quadratic programming, general nonlinear programming, …
Verification Problem as an optimization problem

\[ \exists \ x \in S \ \land \ y \leq 0 \ \land \ y = f(x) \]

Can be solved with the following minimization problem:

\[ y^* = \min_{x \in S} f(x) \]

If the optimal objective \( y^* \leq 0 \), then the original Problem is satisfiable
Verification Problem as an optimization problem

Now we rewrite the neural network verification problem as a **constrained optimization problem** (still using the simple network example):

\[
\begin{align*}
\min \ y \\
\text{s.t.} \quad & y = w^{(3)T} \hat{z}^{(2)} \\
& \hat{z}^{(2)} = \max(z^{(2)}, 0) \\
& z^{(2)} = W^{(2)} \hat{z}^{(1)} \\
& \hat{z}^{(1)} = \max(z^{(1)}, 0) \\
& z^{(1)} = W^{(1)} x \\
& x_i \leq u_i \\
& x_i \geq l_i
\end{align*}
\]

(linear layers)  
(ReLU activation)  
(element-wise input bounds)
Verification Problem as an optimization problem

Now we rewrite the neural network verification problem as a constrained optimization problem (still using the simple network example):

\[
\begin{align*}
\min y \\
\text{s.t.} & \quad y = w^{(3)T} \hat{z}^{(2)} \\
& \quad \hat{z}^{(2)} = \max(z^{(2)}, 0) \\
& \quad z^{(2)} = W^{(2)} \hat{z}^{(1)} \\
& \quad \hat{z}^{(1)} = \max(z^{(1)}, 0) \\
& \quad z^{(1)} = W^{(1)} x \\
& \quad x_i \leq u_i \\
& \quad x_i \geq l_i
\end{align*}
\]

(linear constraints)

(linear constraints)

(nonlinear constraints)

(inputs bounds are also linear constraints)
Optimization formulation for ReLU neurons

Let’s look at this constraint more carefully: \( \hat{z}_j^{(i)} = \max(z_j^{(i)}, 0) \) (for all \( i, j \))

One way to handle it is through the mixed integer linear programming (MILP) formulation. Create an integer variable \( p_j^{(i)} \in \{0, 1\} \), the constraint \( \hat{z}_j^{(i)} = \max(z_j^{(i)}, 0) \) can be equivalently written as (here \( M \) is a “big” number):

\[
\begin{align*}
\hat{z}_j^{(i)} & \leq z_j^{(i)} - M_1 (1 - p_j^{(i)}) \\
\hat{z}_j^{(i)} & \leq M_2 p_j^{(i)} \\
\hat{z}_j^{(i)} & \geq z_j^{(i)} \\
\hat{z}_j^{(i)} & \geq 0
\end{align*}
\]
Optimization formulation for ReLU neurons

\[ \hat{z}_j^{(i)} \leq z_j^{(i)} - M_1(1 - p_j^{(i)}) \]

\[ \hat{z}_j^{(i)} \leq M_2 p_j^{(i)} \]

\[ \hat{z}_j^{(i)} \geq z_j^{(i)} \]

\[ \hat{z}_j^{(i)} \geq 0 \]

When \( p_j^{(i)} = 0 \):

\[ \hat{z}_j^{(i)} \leq z_j^{(i)} - M_1 \hat{z}_j^{(i)} \leq 0 \]

\[ \hat{z}_j^{(i)} \geq z_j^{(i)} \hat{z}_j^{(i)} \geq 0 \]

When \( p_j^{(i)} = 1 \):

\[ \hat{z}_j^{(i)} \leq z_j^{(i)} \hat{z}_j^{(i)} \leq M_2 \]

\[ \hat{z}_j^{(i)} \geq z_j^{(i)} \hat{z}_j^{(i)} \geq 0 \]

\( M_1 \) needs to be small (negative) enough

\( M_2 \) needs to be large enough
Pre-activation bounds

For this formulation to work, $M_1$ and $M_2$ must be properly selected.

If set too conservatively, like $M_1 = 100000$ and $M_2 = -100000$, solver can be very slow.

How to find the tightest $M_1$ and $M_2$?
Pre-activation bounds

For this formulation to work, $M_1$ and $M_2$ must be properly selected.

If set too conservatively, like $M_1 = 100000$ and $M_2 = -100000$, solver can be very slow.

How to find the tightest $M_1$ and $M_2$?
Pre-activation bounds

For this formulation to work, $M_1$ and $M_2$ must be properly selected (usually denoted as $l$ and $u$).

We can use optimization to find these pre-activation bounds - the same formulation as our verification problem before, just changing the optimization variables.

MILP, LP, or more efficient methods (next lecture) can be used to find these.

\[
l_j^{(i)} := \min z_j^{(i)}
\]

\[
u_j^{(i)} := \max z_j^{(i)}
\]
Pre-activation bounds

We can use optimization to find these pre-activation bounds - the same formulation as our verification problem before, just changing the optimization variables.

\[
\begin{align*}
\min y \\
\text{s.t.} \quad & y = w^{(3)T} \hat{z}^{(2)} \\
& \hat{z}^{(2)} = \max(z^{(2)}, 0) \\
& z^{(2)} = W^{(2)} \hat{z}^{(1)} \\
& \hat{z}^{(1)} = \max(z^{(1)}, 0) \\
& z^{(1)} = W^{(1)} x \\
& x_i \leq u_i^{(0)} \\
& x_i \geq l_i^{(0)} \\
\end{align*}
\]

\[ l_j^{(i)} := \min z_j^{(i)} \quad \text{or} \quad u_j^{(i)} := \max z_j^{(i)} \]
Mixed Integer Linear Programming formulation

We rewrite the neural network verification problem as a \textit{constrained optimization problem} (still using the simple network example):

\[
\begin{align*}
    y^*_{\text{MILP}} & := \min y \\
    \text{s.t.} \quad y &= W^{(3)T} \hat{z}^{(2)} \\
    \hat{z}^{(2)} &= \max(z^{(2)}, 0) \\
    z^{(2)} &= W^{(2)} \hat{z}^{(1)} \\
    \hat{z}^{(1)} &= \max(z^{(1)}, 0) \\
    z^{(1)} &= W^{(1)} x \\
    x_i &\leq u_i^{(0)} \\
    x_i &\geq l_i^{(0)}
\end{align*}
\]

Each ReLU is represented by

\[
\begin{align*}
    \hat{z}_j^{(i)} & \leq z_j^{(i)} - l_j^{(i)} (1 - p_j^{(i)}) \\
    \hat{z}_j^{(i)} & \leq u_j^{(i)} p_j^{(i)} \\
    \hat{z}_j^{(i)} & \geq z_j^{(i)} \\
    \hat{z}_j^{(i)} & \geq 0 \\
    p_j^{(i)} & \in \{0, 1\}
\end{align*}
\]
Mixed Integer Linear Programming formulation

However, the integer variables $p_j^{(i)}$ are still hard to handle! (NP-hard)

$$y^*_\text{MILP} := \min y$$

s.t.  \hspace{1cm} y = W^{(3)T} \hat{z}^{(2)}

\[
\begin{align*}
\hat{z}^{(2)} &= \max(\hat{z}^{(2)}, 0) \\
\hat{z}^{(1)} &= \max(\hat{z}^{(1)}, 0) \\
z^{(2)} &= W^{(2)} \hat{z}^{(1)} \\
z^{(1)} &= W^{(1)} x
\end{align*}
\]

$x_i \leq u_i^{(0)}$  
$x_i \geq l_i^{(0)}$

Each ReLU is represented by

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)} (1 - p_j^{(i)})$$

$$\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}$$

$$\hat{z}_j^{(i)} \geq 0$$

$p_j^{(i)} \in \{0, 1\}$
Relaxation of integer variables: Linear programming

Now relaxing the integer constraints into continuous ones.

How does this affect the solution?

Each ReLU is represented by

\[
\begin{align*}
\hat{z}_j^{(i)} &\leq z_j^{(i)} - l_j^{(i)} (1 - p_j^{(i)}) \\
\hat{z}_j^{(i)} &\leq u_j^{(i)} p_j^{(i)} \\
\hat{z}_j^{(i)} &\geq z_j^{(i)} \\
\hat{z}_j^{(i)} &\geq 0 \\
p_j^{(i)} &\in \{0, 1\} \\
0 \leq p_j^{(i)} &\leq 1
\end{align*}
\]
MILP vs LP

\[ y^*_\text{LP} \leq y^*_\text{MILP} \]

It’s not the original MILP solution, but it is a guaranteed lower bound.

Solving LP is polynomial time. Practically a few orders of magnitude faster.
Sound but Incomplete Verification with a Lower Bound

Satisfiability problem: \( \exists \ x \in S \land y \leq 0 \land y = f(x) \)

If \( y_{LP}^* \geq 0 \Rightarrow y^* \geq 0 \Rightarrow \) requirement verified (unsatisfiable)

\( y_{LP}^* \leq 0 \Rightarrow \) return unknown (incomplete)
A closer look at the linear programming relaxation

Each ReLU is represented by

\[
\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)} (1 - p_j^{(i)})
\]

\[
\hat{z}_j^{(i)} \leq u_j^{(i)} p_j^{(i)}
\]

\[
\hat{z}_j^{(i)} \geq z_j^{(i)}
\]

\[
\hat{z}_j^{(i)} \geq 0
\]

\[
0 \leq p_j^{(i)} \leq 1
\]

(Please note that \( l_j^{(i)} \) is negative during the above derivation)
A closer look at the linear programming relaxation

Each ReLU is represented by

\[
\begin{align*}
\hat{z}_j^{(i)} &\leq z_j^{(i)} - l_j^{(i)} (1 - p_j^{(i)}) \\
\hat{z}_j^{(i)} &\leq u_j^{(i)} p_j^{(i)} \\
\hat{z}_j^{(i)} &\geq z_j^{(i)} \\
\hat{z}_j^{(i)} &\geq 0
\end{align*}
\]

\[
\begin{align*}
\hat{z}_j^{(i)} &\leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} \\
\hat{z}_j^{(i)} &\geq z_j^{(i)} \\
\hat{z}_j^{(i)} &\geq 0
\end{align*}
\]

“Triangle” relaxation
A closer look at the linear programming relaxation

MILP: solutions are constrained on **ReLU function**
Linear programming: solutions are constrained on the **triangle**

\[
\hat{z}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} z_j^{(i)} - \frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} \\
\hat{z}_j^{(i)} \geq z_j^{(i)} \\
\hat{z}_j^{(i)} \geq 0
\]
MILP vs LP vs DPLL(T)

MILP:
- Branch and bound is used to make decisions only on certain number of binary variables
- Specialized methods to accelerate solving (e.g., branching heuristics, cutting planes)
- Complete (solve $y^*$ exactly)
- Typically scales much better than DPLL(T)

Linear programming:
- No integer variables
- No variable decisions needed (no exponential time search)
- Simplex algorithm can solve it relatively fast, a few thousands neurons are ok
- Incomplete (has to return unknown in some cases)
Summary

- Neural network verification problem
- Solving the verification problem with SMT solvers
- Integer programming formulation
- Linear programming formulation and linear relaxation of ReLUs
- Next lecture: linear bound propagation method (CROWN) for efficient neural network verification

Reading for the next lecture:

Homework 1 due on Sunday (2/11) 11:59 pm