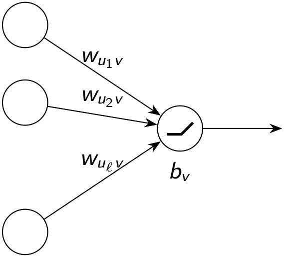
Computing the Maximum Function with ReLU Neural Networks

Christoph Hertrich
Joint WIP with
Amitabh Basu, Marco Di Summa, and Martin Skutella

IOL & COGA Research Seminar
November 5, 2020

A Single (Hidden) ReLU Neuron

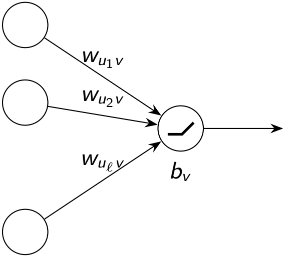
Outputs $o(u_i)$ of previous neurons



$o(v) = \mathrm{relu}(b_v + \sum_{i=1}^{\ell} w_{u_i v} o(u_i))$

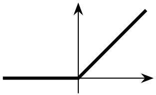
A Single (Hidden) ReLU Neuron

Outputs $o(u_i)$ of previous neurons



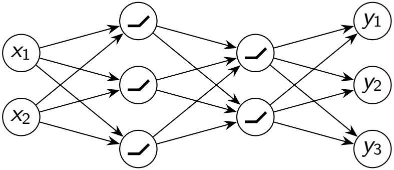
$o(v) = \mathrm{relu}(b_v + \sum_{i=1}^{\ell} w_{u_i v} o(u_i))$

Rectified linear unit (ReLU): $\mathrm{relu}(x) = \max\{0, x\}$



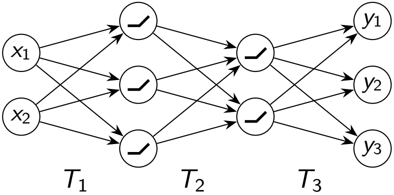
ReLU Feedforward Neural Networks

- Acyclic (layered) digraph of ReLU neurons



ReLU Feedforward Neural Networks

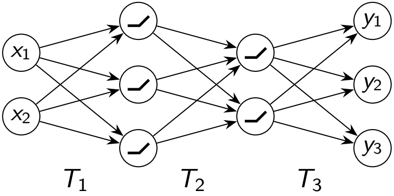
- Acyclic (layered) digraph of ReLU neurons



- Computes function $T_k \circ \text{relu} \circ T_{k-1} \circ \cdots \circ T_2 \circ \text{relu} \circ T_1$ with affine transformations T_i.

ReLU Feedforward Neural Networks

- Acyclic (layered) digraph of ReLU neurons

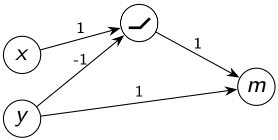


- Computes function $T_k \circ \mathrm{relu} \circ T_{k-1} \circ \cdots \circ T_2 \circ \mathrm{relu} \circ T_1$ with affine transformations T_i.

- Example: depth 3 (2 hidden layers), width 3.

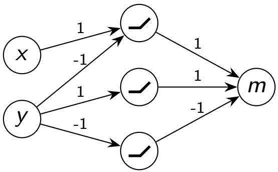
Example: Computing the Maximum of Two Numbers

$$\max\{x, y\} = \max\{x - y, 0\} + y$$

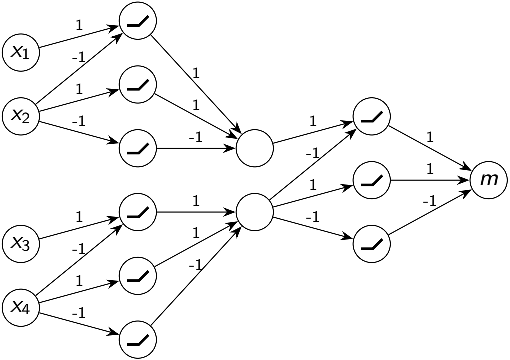


Example: Computing the Maximum of Two Numbers

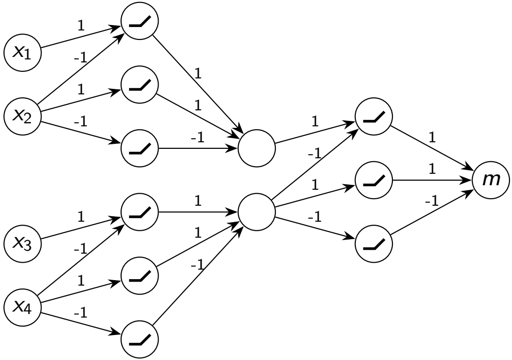
$$\max\{x, y\} = \max\{x - y, 0\} + y$$



Example: Computing the Maximum of Four Numbers



Example: Computing the Maximum of Four Numbers



- Inductively: Maximum of n numbers with depth $\lceil \log_2(n) \rceil + 1$.

Expressivity of ReLU neural networks

Theorem (Wang, Sun (2005))

For any continuous and piecewise linear (CPWL) function $f : \mathbb{R}^n \to \mathbb{R}$, there are $\alpha_i \in \mathbb{R}$, $a_{ij} \in R^n$, and $b_{ij} \in \mathbb{R}$ such that

$$f(x) = \sum_i \alpha_i \max\{a_{ij}^T x + b_{ij} \mid j = 1, \dots, n+1\}.$$

Expressivity of ReLU neural networks

Theorem (Wang, Sun (2005))

For any continuous and piecewise linear (CPWL) function $f : \mathbb{R}^n \to \mathbb{R}$, there are $\alpha_i \in \mathbb{R}$, $a_{ij} \in R^n$, and $b_{ij} \in \mathbb{R}$ such that

$$f(x) = \sum_i \alpha_i \max\{a_{ij}^T x + b_{ij} \mid j = 1, \dots, n+1\}.$$

Theorem (Arora, Basu, Mianjy, Mukherjee (2018))

$f : \mathbb{R}^n \to \mathbb{R}$ can be computed by ReLU NN if and only if f is CPWL.

Expressivity of ReLU neural networks

Theorem (Wang, Sun (2005))

For any continuous and piecewise linear (CPWL) function $f : \mathbb{R}^n \to \mathbb{R}$*, there are* $\alpha_i \in \mathbb{R}$*,* $a_{ij} \in R^n$*, and* $b_{ij} \in \mathbb{R}$ *such that*

$$f(x) = \sum_i \alpha_i \max\{a_{ij}^T x + b_{ij} \mid j = 1, \ldots, n+1\}.$$

Theorem (Arora, Basu, Mianjy, Mukherjee (2018))

$f : \mathbb{R}^n \to \mathbb{R}$ *can be computed by ReLU NN if and only if* f *is CPWL.*

In this case, depth $\lceil \log_2(n+1) \rceil + 1$ *suffices.*

Expressivity of ReLU neural networks

Theorem (Wang, Sun (2005))

For any continuous and piecewise linear (CPWL) function $f : \mathbb{R}^n \to \mathbb{R}$, there are $\alpha_i \in \mathbb{R}$, $a_{ij} \in R^n$, and $b_{ij} \in \mathbb{R}$ such that

$$f(x) = \sum_i \alpha_i \max\{a_{ij}^T x + b_{ij} \mid j = 1, \ldots, n+1\}.$$

Theorem (Arora, Basu, Mianjy, Mukherjee (2018))

$f : \mathbb{R}^n \to \mathbb{R}$ can be computed by ReLU NN if and only if f is CPWL.

In this case, depth $\lceil \log_2(n+1) \rceil + 1$ suffices.

$\Rightarrow$ Everything depends on the maximum function!

Is logarithmic depth best possible?

- Known: $\max\{0, x_1, x_2\}$ cannot be computed with 2 layers.

Is logarithmic depth best possible?

- Known: $\max\{0, x_1, x_2\}$ cannot be computed with 2 layers.
- Smallest open case:
  Can $\max\{0, x_1, x_2, x_3, x_4\}$ be computed with 3 layers?

Is logarithmic depth best possible?

- Known: $\max\{0, x_1, x_2\}$ cannot be computed with 2 layers.
- Smallest open case:
  Can $\max\{0, x_1, x_2, x_3, x_4\}$ be computed with 3 layers?
- No function known that provably needs more than 3 layers.

In this talk:

In this talk:

Computational proof that $\max\{0, x_1, x_2, x_3, x_4\}$ cannot be computed with 3 layers

In this talk:

Computational proof that $\max\{0, x_1, x_2, x_3, x_4\}$
cannot be computed with 3 layers
under an additional assumption.

In this talk:

Computational proof that $\max\{0, x_1, x_2, x_3, x_4\}$
cannot be computed with 3 layers
under an additional assumption.

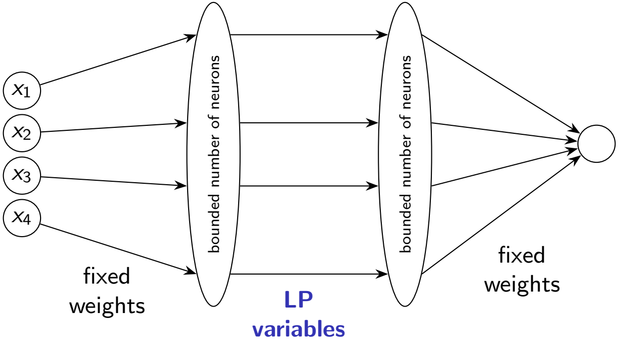
(for notational purposes: $x_0 := 0$.)

Strategy:

Strategy:

Construct a linear program (LP) that is feasible if and only if such a neural network exists.

Rough Idea



x_1
x_2
x_3
x_4
bounded number of neurons
bounded number of neurons
fixed weights
LP variables
fixed weights

Observation: No Bias Necessary

- f *positively homogeneous* if $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$.

Observation: No Bias Necessary

- *f positively homogeneous* if $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$.

Proposition

If an NN computes a positively homogeneous function f, then the same NN without biases computes the same function f.

Observation: No Bias Necessary

- f *positively homogeneous* if $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$.

Proposition

If an NN computes a positively homogeneous function f, then the same NN without biases computes the same function f.

Proof.

- Let $\tilde{f}$ be the function computed by the NN without bias.

Observation: No Bias Necessary

- f *positively homogeneous* if $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$.

Proposition

If an NN computes a positively homogeneous function f, then the same NN without biases computes the same function f.

Proof.

- Let $\tilde{f}$ be the function computed by the NN without bias.
- There is a constant C with $|f(x) - \tilde{f}(x)| \leq C$ for all x.

Observation: No Bias Necessary

- f *positively homogeneous* if $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$.

Proposition

If an NN computes a positively homogeneous function f, then the same NN without biases computes the same function f.

Proof.

- Let $\tilde{f}$ be the function computed by the NN without bias.
- There is a constant C with $|f(x) - \tilde{f}(x)| \leq C$ for all x.
- If there is an x with $f(x) \neq \tilde{f}(x)$, then αx (with large α) yields a contradiction.

Observation: No Bias Necessary

- f *positively homogeneous* if $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$.

Proposition

If an NN computes a positively homogeneous function f, then the same NN without biases computes the same function f.

Proof.

- Let $\tilde{f}$ be the function computed by the NN without bias.
- There is a constant C with $|f(x) - \tilde{f}(x)| \leq C$ for all x.
- If there is an x with $f(x) \neq \tilde{f}(x)$, then αx (with large α) yields a contradiction. $\square$

$\Rightarrow$ From now on, only consider NNs without biases.

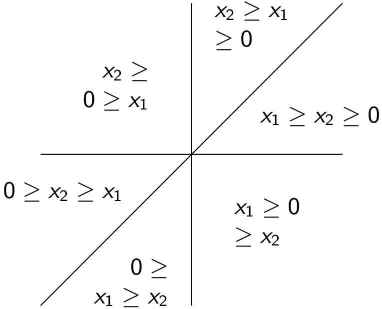
The Assumption

The output of each neuron can only have breakpoints when the relative ordering of the five numbers 0, x_1, ..., x_4 changes.

The Assumption

The output of each neuron can only have breakpoints when the relative ordering of the five numbers 0, x_1, ..., x_4 changes.

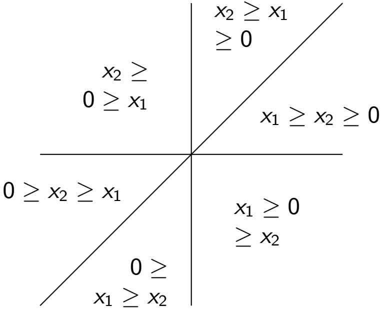
Example for $\max\{0, x_1, x_2\}$:



The Assumption

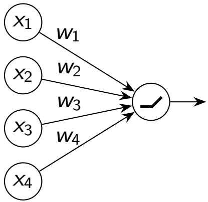
The output of each neuron can only have breakpoints when the relative ordering of the five numbers 0, x_1, ..., x_4 changes.

Example for $\max\{0, x_1, x_2\}$:

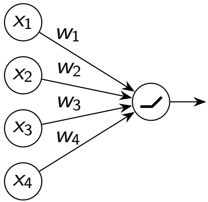


The $\binom{5}{2} = 10$ hyperplanes $x_i = x_j$, $0 \leq i < j \leq 4$, subdivide $\mathbb{R}^4$ into $5! = 120$ cells in each of which the output of each neuron is affine.

Neurons in the First Hidden Layer

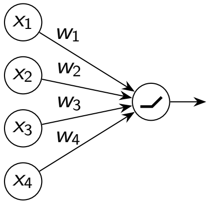


Neurons in the First Hidden Layer



- Has breaking hyperplane at $\sum_{i=1}^{4} w_i x_i = 0$.
- Must be one of the 10 specified hyperplanes.

Neurons in the First Hidden Layer



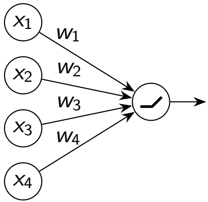
- Has breaking hyperplane at $\sum_{i=1}^{4} w_i x_i = 0$.
- Must be one of the 10 specified hyperplanes.

Claim: The output of a neuron in the first hidden layer is a linear combination of the following 14 functions:

$$\begin{array}{cccc}
\max\{x_1, 0\}, & \max\{x_2, 0\}, & \max\{x_3, 0\}, & \max\{x_4, 0\}, \\
\max\{-x_1, 0\}, & \max\{-x_2, 0\}, & \max\{-x_3, 0\}, & \max\{-x_4, 0\},
\end{array}$$

$$\begin{array}{ccc}
\max\{x_1 - x_2, 0\}, & \max\{x_1 - x_3, 0\}, & \max\{x_1 - x_4, 0\}, \\
\max\{x_2 - x_3, 0\}, & \max\{x_2 - x_4, 0\}, & \max\{x_3 - x_4, 0\}
\end{array}$$

Neurons in the First Hidden Layer



- Has breaking hyperplane at $\sum_{i=1}^{4} w_i x_i = 0$.
- Must be one of the 10 specified hyperplanes.

Claim: The output of a neuron in the first hidden layer is a linear combination of the following 14 functions:

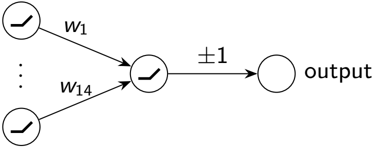
$$\max\{x_1, 0\}, \quad \max\{x_2, 0\}, \quad \max\{x_3, 0\}, \quad \max\{x_4, 0\},$$
$$\max\{-x_1, 0\}, \quad \max\{-x_2, 0\}, \quad \max\{-x_3, 0\}, \quad \max\{-x_4, 0\},$$

$$\max\{x_1 - x_2, 0\}, \quad \max\{x_1 - x_3, 0\}, \quad \max\{x_1 - x_4, 0\},$$
$$\max\{x_2 - x_3, 0\}, \quad \max\{x_2 - x_4, 0\}, \quad \max\{x_3 - x_4, 0\}$$

$\Rightarrow$ **14 neurons suffice!**

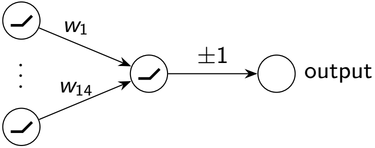
Neurons in the Second Hidden Layer

Observe: WLOG output weights ± 1.



Neurons in the Second Hidden Layer

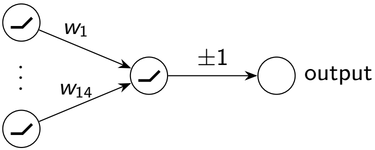
Observe: WLOG output weights ± 1.



Recall: cell = set of inputs with fixed ordering of 0, $x_1, \ldots, x_4$.

Neurons in the Second Hidden Layer

Observe: WLOG output weights ± 1.



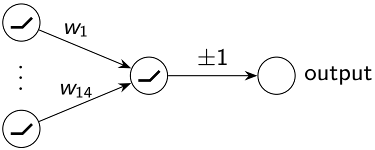
Recall: cell = set of inputs with fixed ordering of 0, x_1, ..., x_4.

Observe: activation of neurons has fixed sign within each cell.

$$\rightsquigarrow \text{activation pattern in } \{-1, 1\}^{120}$$

Neurons in the Second Hidden Layer

Observe: WLOG output weights ± 1.



Recall: cell $=$ set of inputs with fixed ordering of 0, $x_1, \ldots, x_4$.

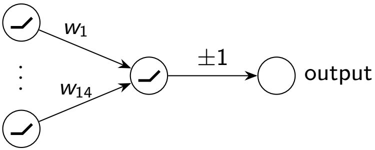
Observe: activation of neurons has fixed sign within each cell.

$$\rightsquigarrow \text{activation pattern in } \{-1, 1\}^{120}$$

Observe: neurons with equal activation pattern and equal output weight can be combined.

Neurons in the Second Hidden Layer

Observe: WLOG output weights ± 1.



Recall: cell $=$ set of inputs with fixed ordering of 0, $x_1, \ldots, x_4$.

Observe: activation of neurons has fixed sign within each cell.

$$\rightsquigarrow \text{activation pattern in } \{-1, 1\}^{120}$$

Observe: neurons with equal activation pattern and equal output weight can be combined.

$$\Rightarrow \mathbf{2^{121}} \textbf{ neurons suffice!}$$

2^{121}? Are you serious?

2^{121}? Are you serious?

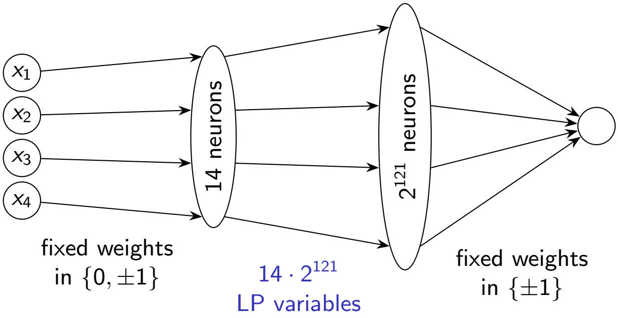
- Many activation patterns impossible.
- Can use linear programming to enumerate possible patterns.

2^{121}? Are you serious?

- Many activation patterns impossible.
- Can use linear programming to enumerate possible patterns.

$\rightsquigarrow$ For this talk suppose $2^{121} \approx 200\,000$.

Variables of the Linear Program



Constraints of the Linear Program

Two types of constraints:

1. Ensuring output correctness,
2. Ensuring activation patterns.

Constraints of the Linear Program

Two types of constraints:

1. Ensuring output correctness,
2. Ensuring activation patterns.

Observe: Each cell is conic combination of 4 extreme rays.

Example: Cell $x_1 \geq x_2 \geq 0 \geq x_3 \geq x_4$:

$$(1,0,0,0), \qquad (1,1,0,0), \qquad (0,0,-1,-1), \qquad (0,0,0,-1).$$

Constraints of the Linear Program

Two types of constraints:

1. Ensuring output correctness,
2. Ensuring activation patterns.

Observe: Each cell is conic combination of 4 extreme rays.

Example: Cell $x_1 \geq x_2 \geq 0 \geq x_3 \geq x_4$:

$$(1,0,0,0), \qquad (1,1,0,0), \qquad (0,0,-1,-1), \qquad (0,0,0,-1).$$

$\Rightarrow$ Enough to have constraints for each extreme ray.

Constraints of the Linear Program

Two types of constraints:

1. Ensuring output correctness,
2. Ensuring activation patterns.

Observe: Each cell is conic combination of 4 extreme rays.

Example: Cell $x_1 \geq x_2 \geq 0 \geq x_3 \geq x_4$:

$$(1,0,0,0), \qquad (1,1,0,0), \qquad (0,0,-1,-1), \qquad (0,0,0,-1).$$

$\Rightarrow$ Enough to have constraints for each extreme ray.

$\Rightarrow$ Have these constraints:

1. $4 \cdot 120$ equality constraints.
2. $4 \cdot 120 \cdot 2^{121}$ inequality constraints.

Gurobi computes ...

Gurobi computes ...

... for less than a minute ...

Gurobi computes ...

... for less than a minute ...

... and outputs ...

Gurobi computes ...

... for less than a minute ...

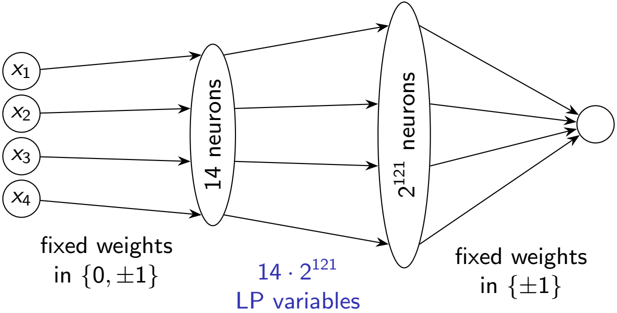
... and outputs ...

Infeasible!

Three Obvious Next Steps ...

- Proving the assumption.
- Finding a "real" proof.
- Generalize to more layers.

Thank you!



Questions? Ideas?