EMSOFT - International Conference on Embedded Software, 2022

An MILP Encoding for Efficient Verification of Quantized DNNs

Samvid Mistry*, Indranil Saha**, Swarnendu Biswas**

mistrysamvid@gmail.com, {isaha, swarnendu}@cse.iitk.ac.in

* = GitHub Inc.
** = Indian Institute of Technology Kanpur

Introduction

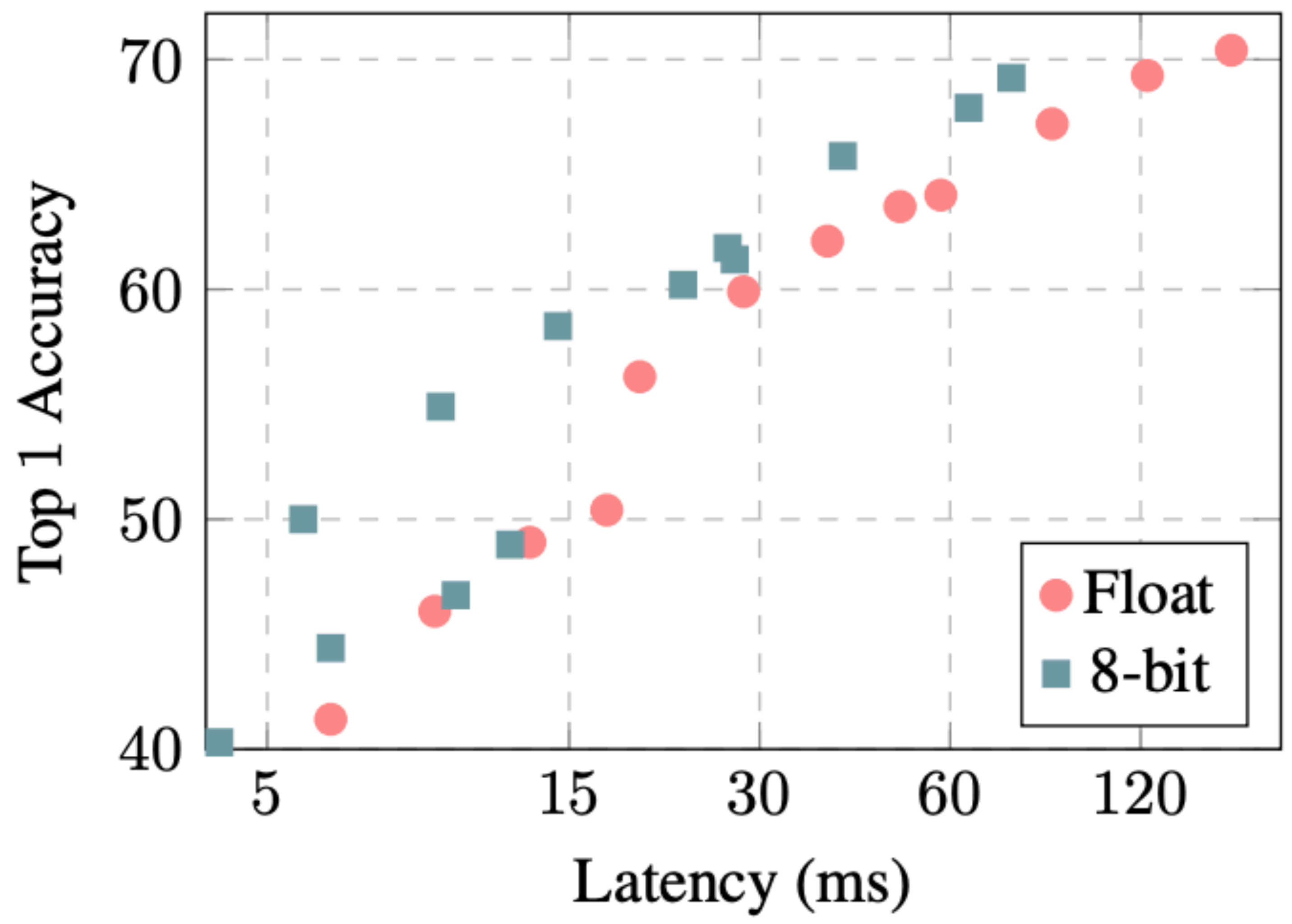


Figure 4.1: Latency-vs-accuracy tradeoff of float vs. integer-only MobileNets on ImageNet using Snapdragon 835 big cores.

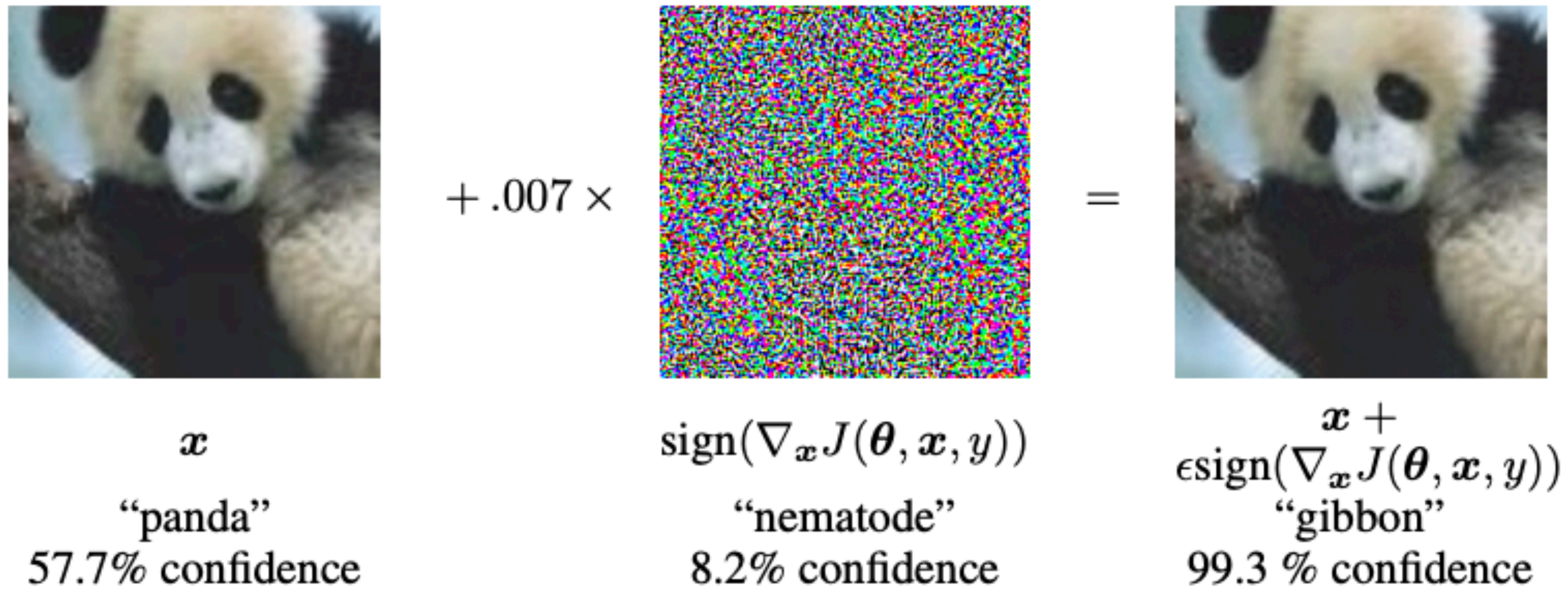
DM	Type	Precision	Recall	LITTLE (ms)	big (ms)
100%	floats	68%	76%	711	337
	8 bits	66%	75%	372	154
50%	floats	65%	70%	233	106
	8 bits	62%	70%	134	56
25%	floats	56%	64%	100	44
	8 bits	54%	63%	67	28

Table 4.5: Face detection accuracy of floating point and integer-only quantized models. The reported precision / recall is averaged over different precision / recall values where an IOU of x between the groundtruth and predicted windows is considered a correct detection, for x in $\{0.5, 0.55, \ldots, 0.95\}$. Latency (ms) of floating point and quantized models are reported on Qualcomm Snapdragon 835 using a single LITTLE and big core, respectively.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018

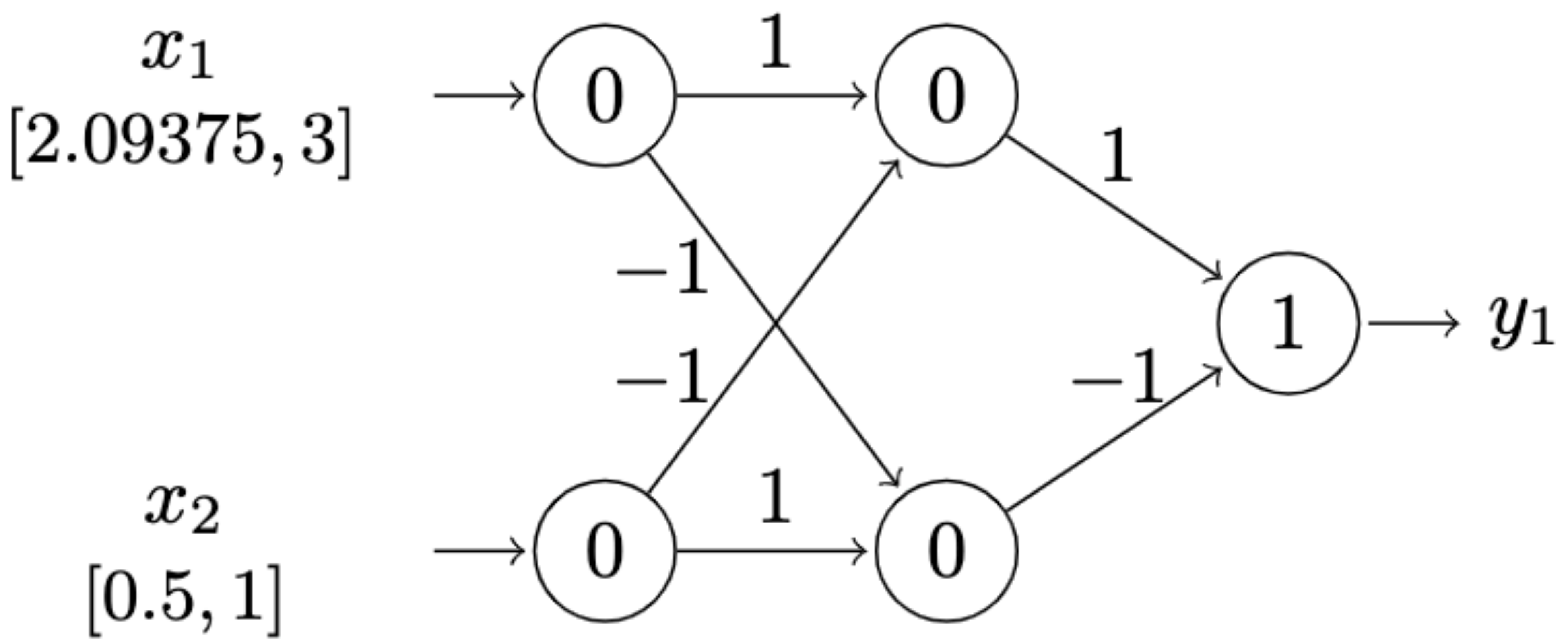
Adversarial Attacks

M. Giacobbe et al, “How Many Bits Does it Take to Quantize Your Neural Network?” TACAS, 2020

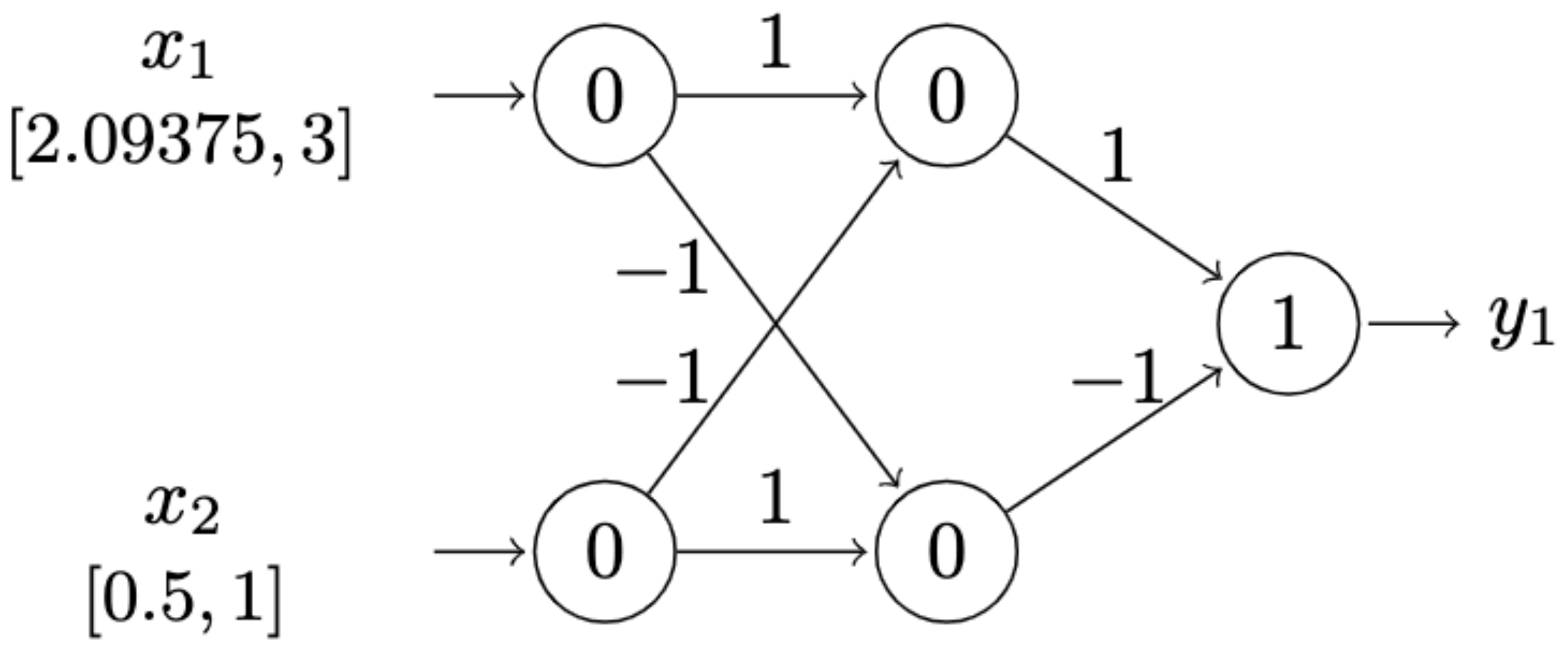


Goodfellow et al, Explaining and Harnessing Adversarial Examples, ICLR 2015

Floating-point → Fixed-point

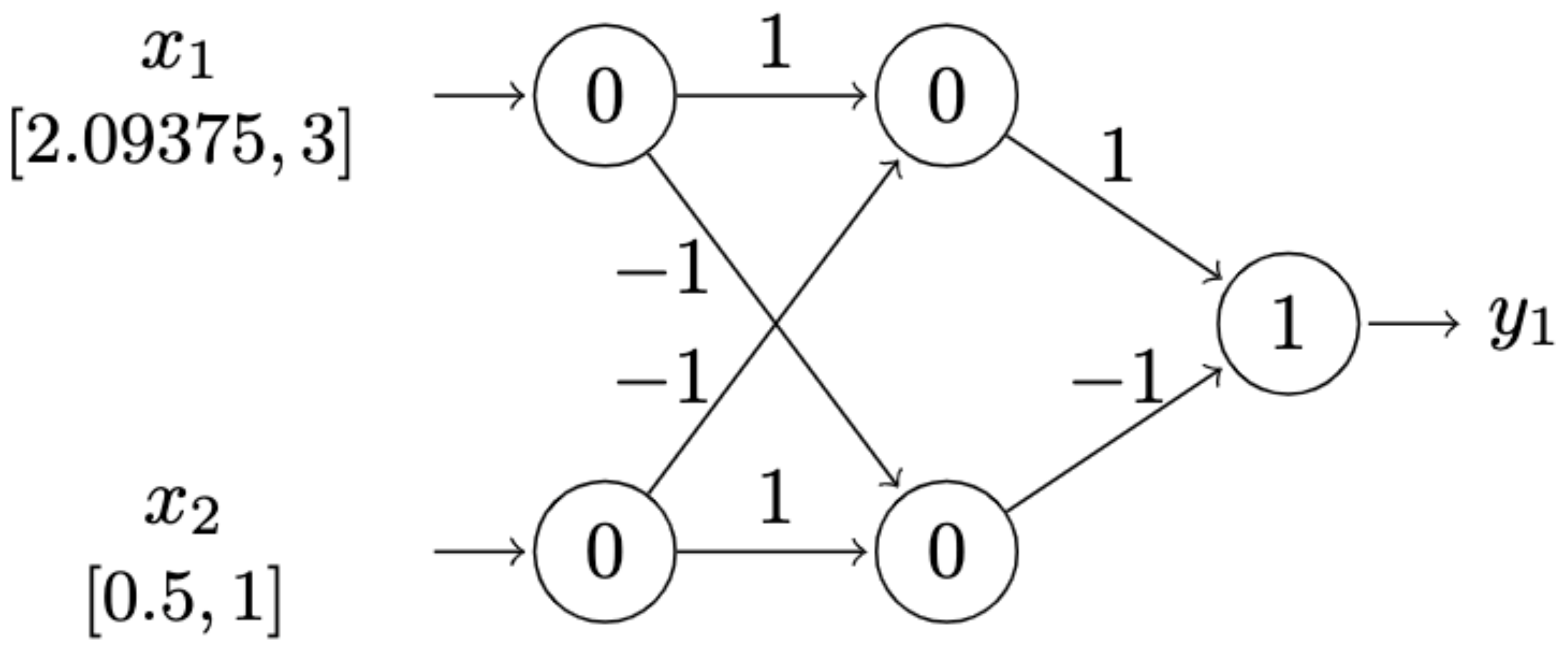


Floating-point → Fixed-point



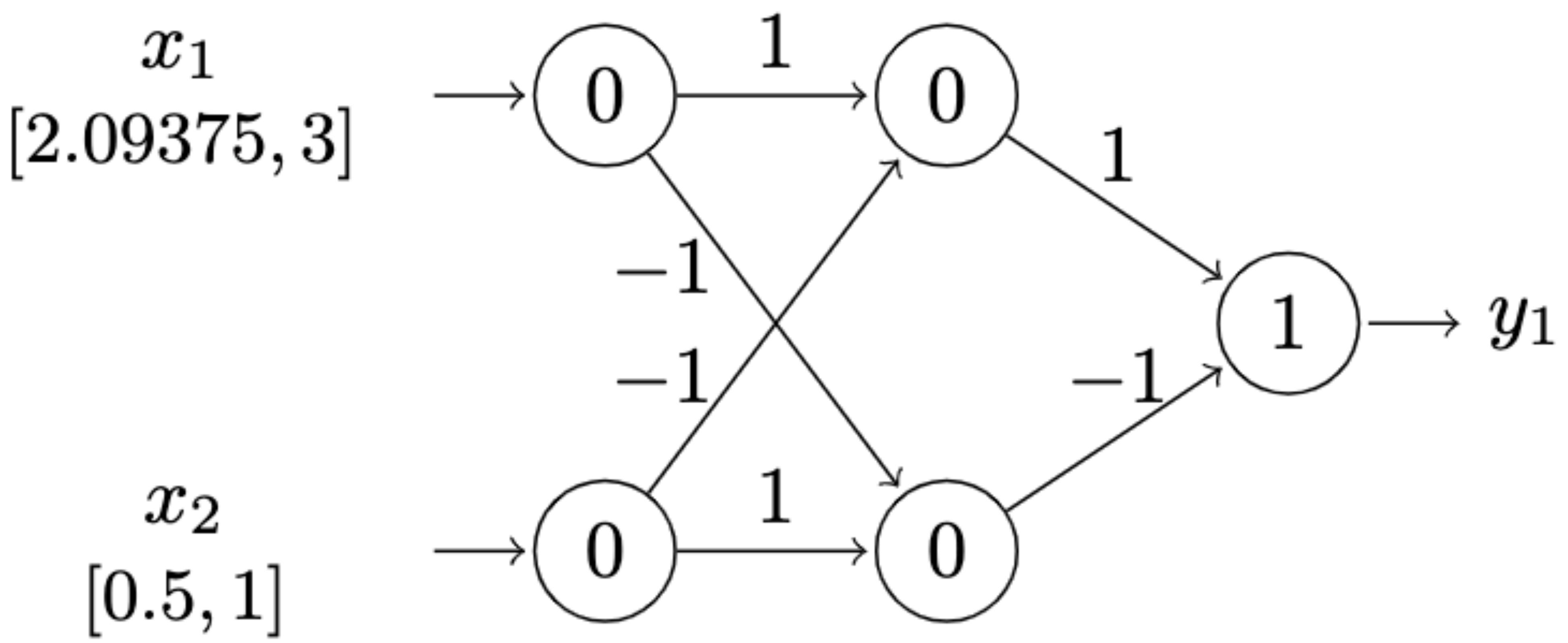
- $\mathcal{Q}$`[QI].[QF]` = fixed-point value with `QI` integer and `QF` fractional bits

Floating-point → Fixed-point



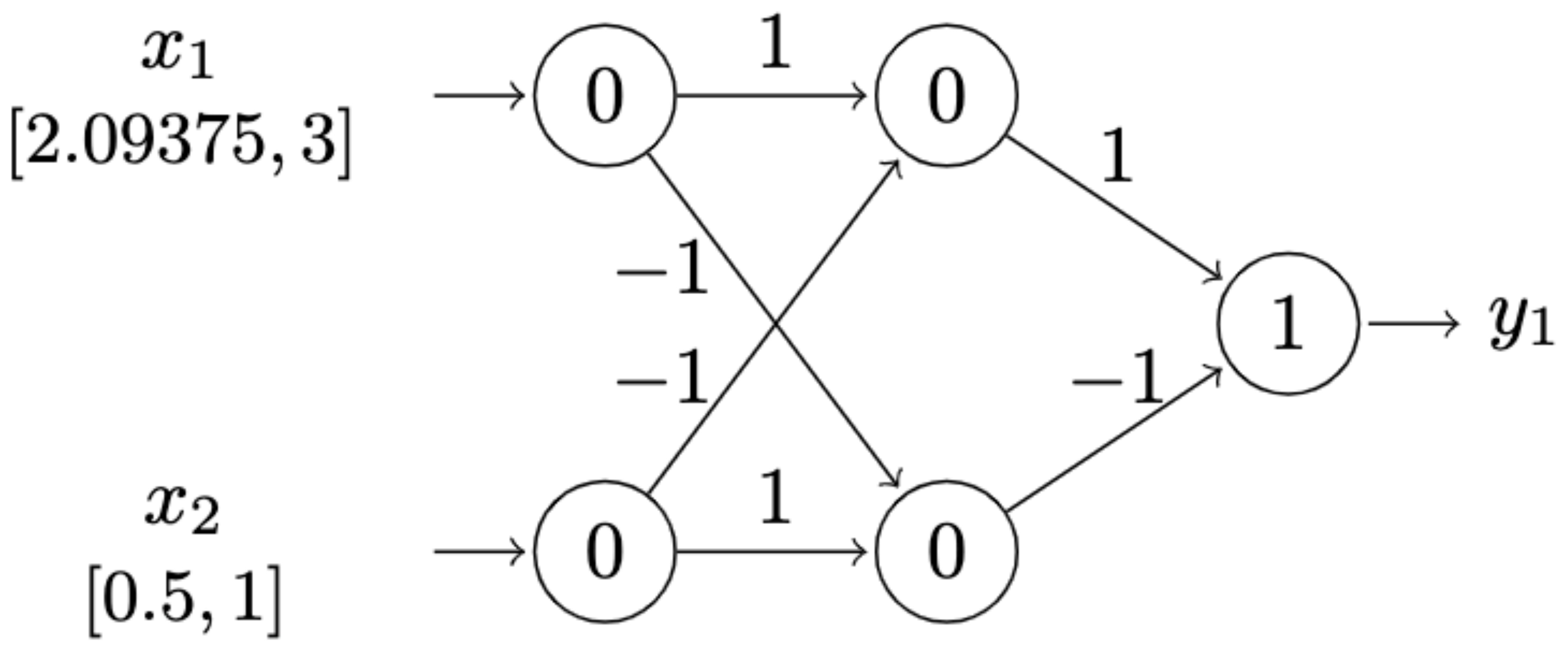
- Q[QI].[QF] = fixed-point value with QI integer and QF fractional bits
- $Q4.4$ = Fixed-point value with 4 integer and 4 fractional bits

Floating-point → Fixed-point



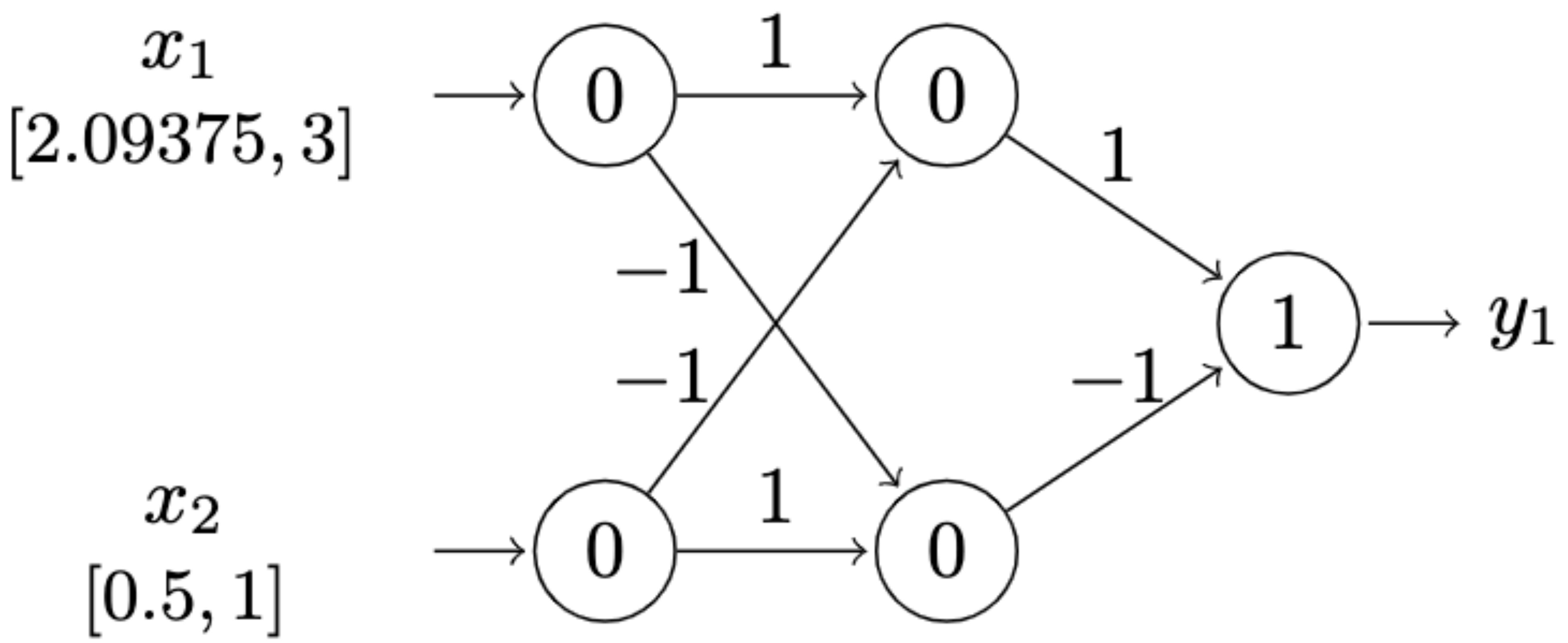
- $\mathcal{Q}[\mathtt{QI}].[\mathtt{QF}]$ = fixed-point value with $\mathtt{QI}$ integer and $\mathtt{QF}$ fractional bits
- $\mathcal{Q}4.4$ = Fixed-point value with 4 integer and 4 fractional bits
- $\mathcal{Q}[\mathtt{a}].[\mathtt{b}] \cdot \mathcal{Q}[\mathtt{p}].[\mathtt{q}] = \mathcal{Q}[\mathtt{a}+\mathtt{p}].[\mathtt{b}+\mathtt{q}]$

Floating-point → Fixed-point

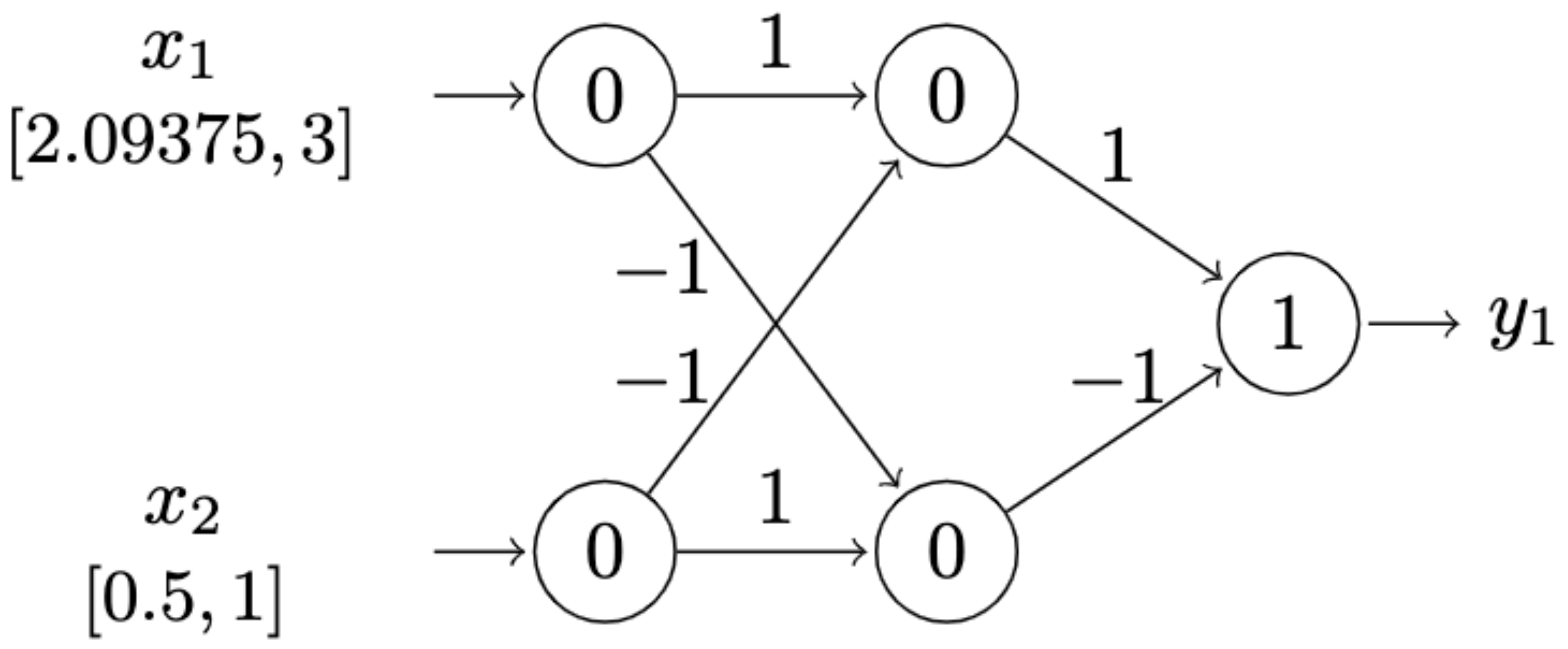


- Q[QI].[QF] = fixed-point value with QI integer and QF fractional bits
- $Q4.4$ = Fixed-point value with 4 integer and 4 fractional bits
- $Q[\mathtt{a}].[\mathtt{b}] \cdot Q[\mathtt{p}].[\mathtt{q}] = Q[\mathtt{a}+\mathtt{p}].[\mathtt{b}+\mathtt{q}]$
- $Q4.4 \cdot Q4.4 = Q8.8 \rightarrow Q8.8 >> 4 = Q8.4 \rightarrow Q4.4 = \min(255, \max(-256, Q8.4))$

Floating-point → Fixed-point

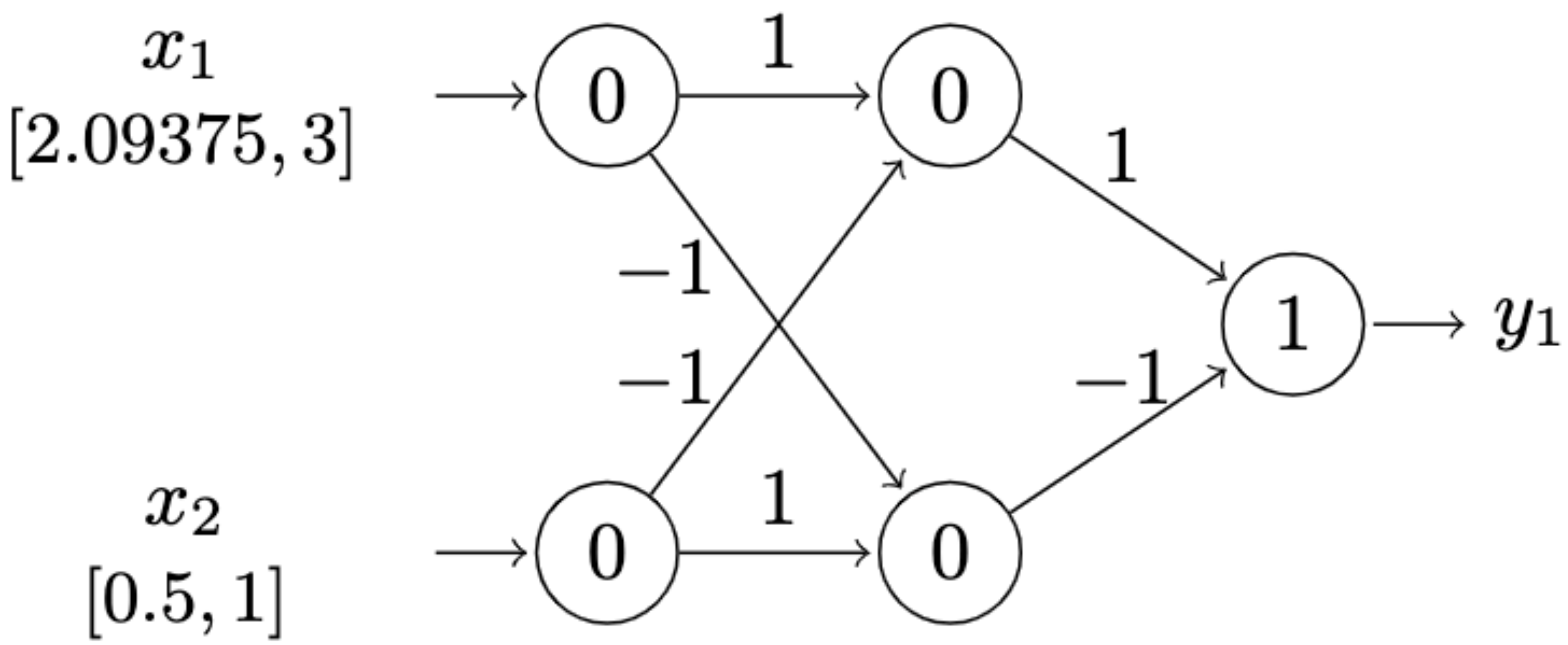


Floating-point → Fixed-point



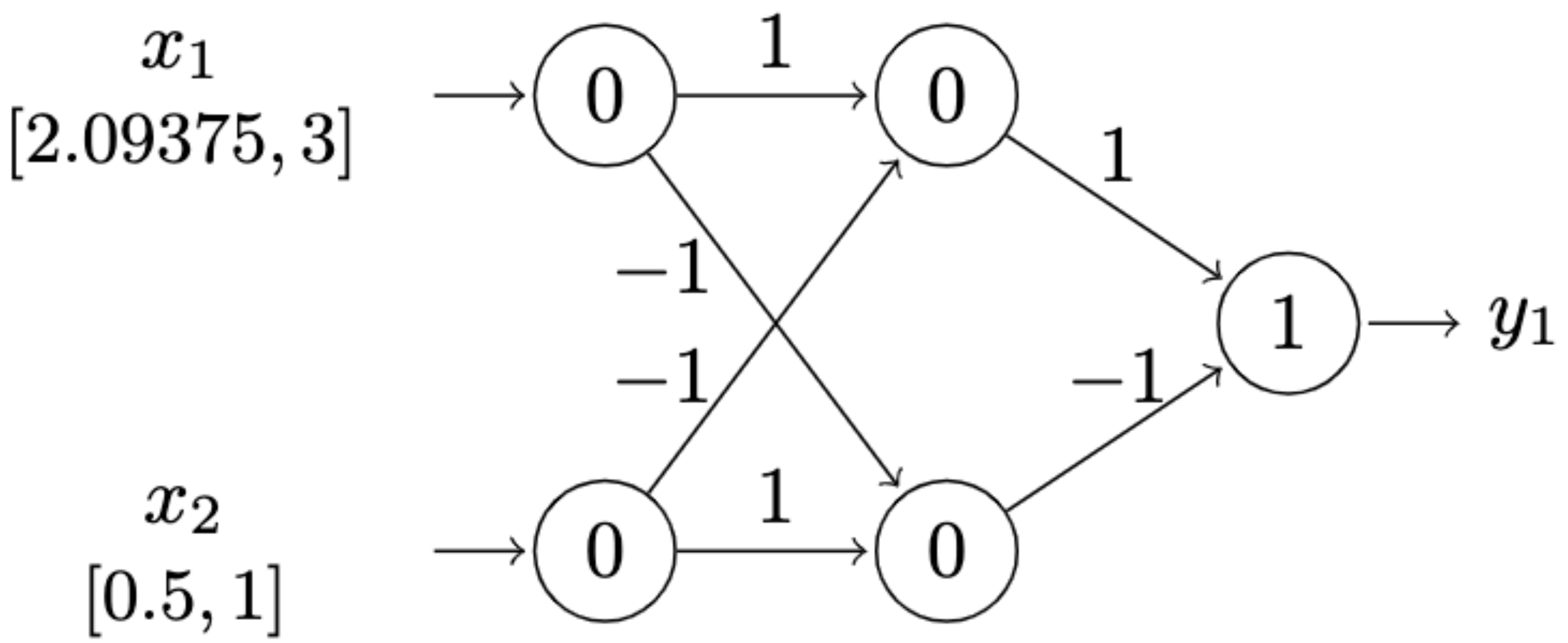
- $\texttt{fixedpoint} = \texttt{int}(\texttt{floatingpoint} \cdot 2^F)$

Floating-point → Fixed-point



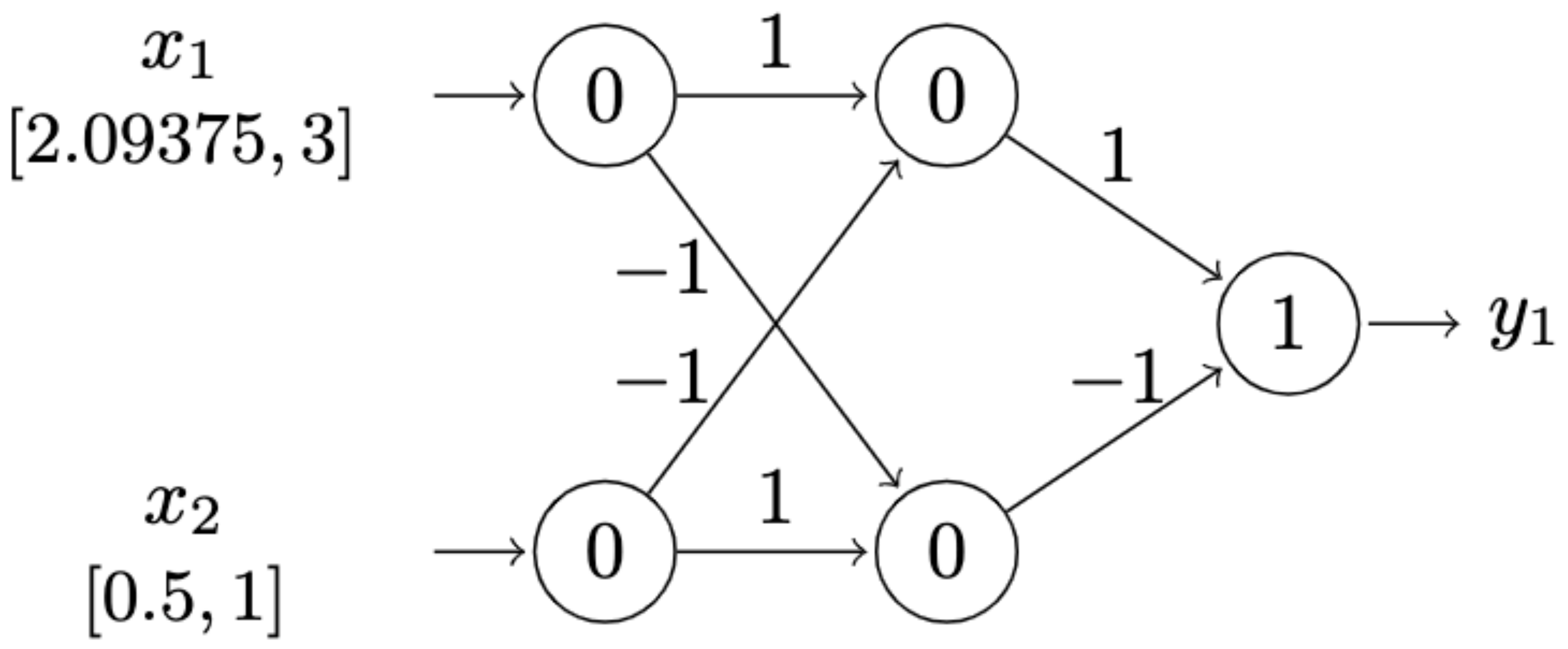
- $\texttt{fixedpoint} = \texttt{int}(\texttt{floatingpoint} \cdot 2^F)$
- $Q4.4$ for all nodes of network

Floating-point → Fixed-point



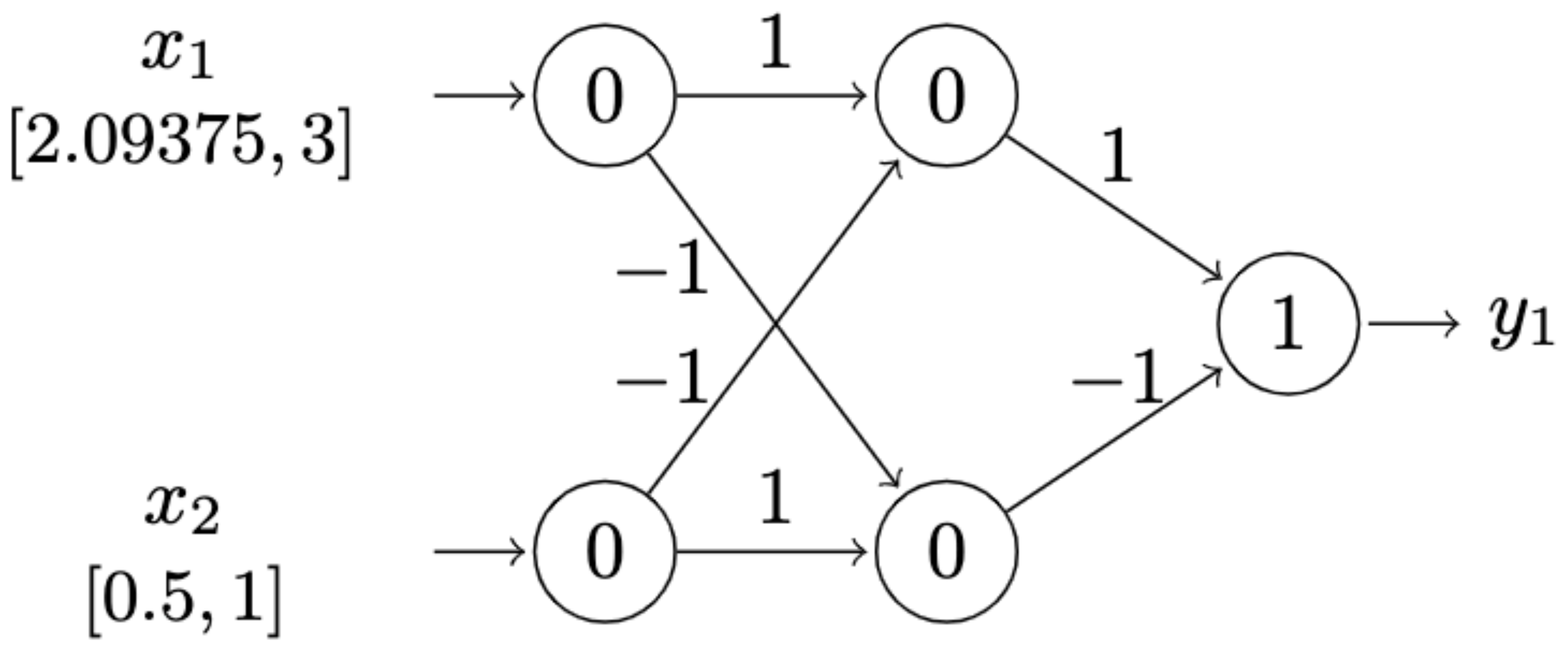
- $\texttt{fixedpoint} = \texttt{int}(\texttt{floatingpoint} \cdot 2^F)$
- $Q4.4$ for all nodes of network
- $x_1 = \texttt{int}(2.09375 \cdot 2^4) = \texttt{int}(33.5) = 33$

Floating-point → Fixed-point



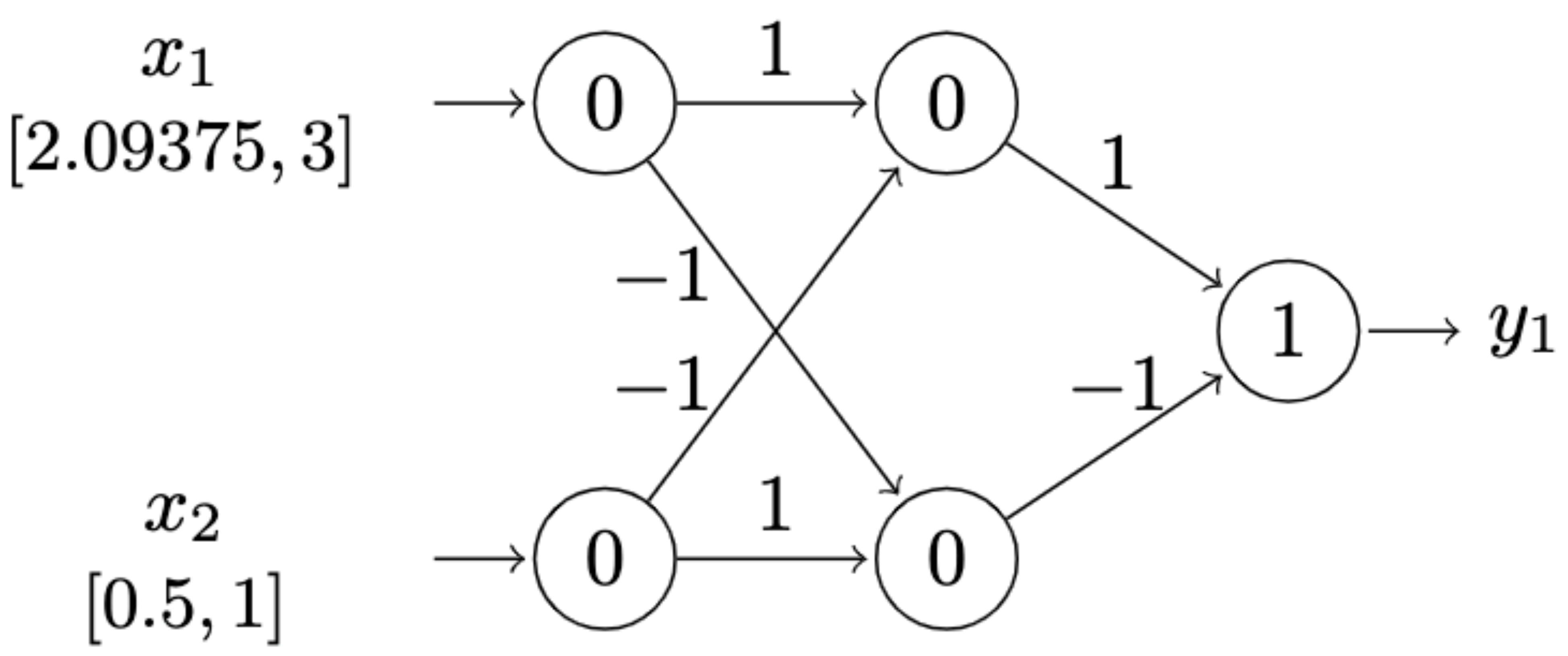
- $\texttt{fixedpoint} = \texttt{int}(\texttt{floatingpoint} \cdot 2^F)$
- $Q4.4$ for all nodes of network
- $x_1 = \texttt{int}(2.09375 \cdot 2^4) = \texttt{int}(33.5) = 33$
- $x_2 = \texttt{int}(1 \cdot 2^4) = \texttt{int}(16) = 16$

Floating-point → Fixed-point



- $\texttt{fixedpoint} = \texttt{int}(\texttt{floatingpoint} \cdot 2^F)$
- $Q4.4$ for all nodes of network
- $x_1 = \texttt{int}(2.09375 \cdot 2^4) = \texttt{int}(33.5) = 33$
- $x_2 = \texttt{int}(1 \cdot 2^4) = \texttt{int}(16) = 16$
- $\texttt{floatingpoint} = \texttt{fixedpoint}/2^F$

Floating-point → Fixed-point



- $\texttt{fixedpoint} = \texttt{int}(\texttt{floatingpoint} \cdot 2^F)$
- $Q4.4$ for all nodes of network
- $x_1 = \texttt{int}(2.09375 \cdot 2^4) = \texttt{int}(33.5) = 33$
- $x_2 = \texttt{int}(1 \cdot 2^4) = \texttt{int}(16) = 16$
- $\texttt{floatingpoint} = \texttt{fixedpoint}/2^F$
- $y_1 = 33/2^4 = 2.0625$

Sound Encoding of Rounding - Example

Sound Encoding of Rounding - Example

- Let $\overline{\eta} = 55$ and $F = 2$

Sound Encoding of Rounding - Example

- Let $\overline{\eta} = 55$ and $F = 2$
- $\bar{\zeta} = \texttt{int}(\eta)$ where $\eta = \overline{\eta} \cdot 2^{-2} = 13.75$

Sound Encoding of Rounding - Example

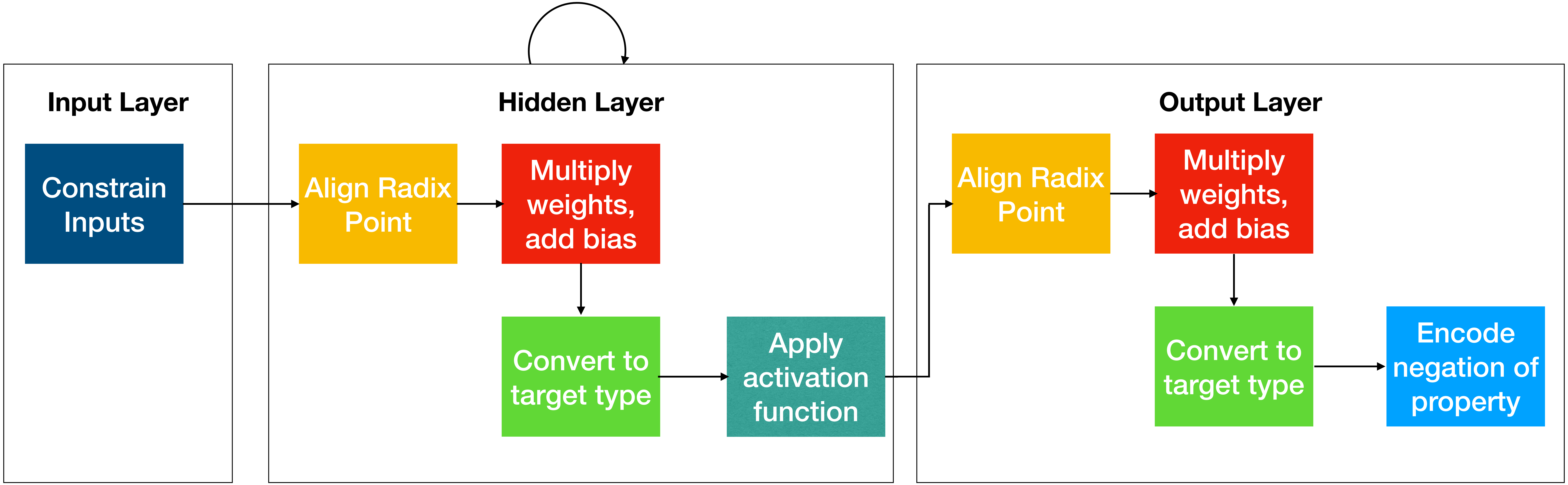
- Let $\overline{\eta} = 55$ and $F = 2$
- $\bar{\zeta} = \texttt{int}(\eta)$ where $\eta = \overline{\eta} \cdot 2^{-2} = 13.75$
- $1 - 2^{-2} \leq \texttt{offset} < 1 \rightarrow 0.75 \leq \texttt{offset} < 1$

Sound Encoding of Rounding - Example

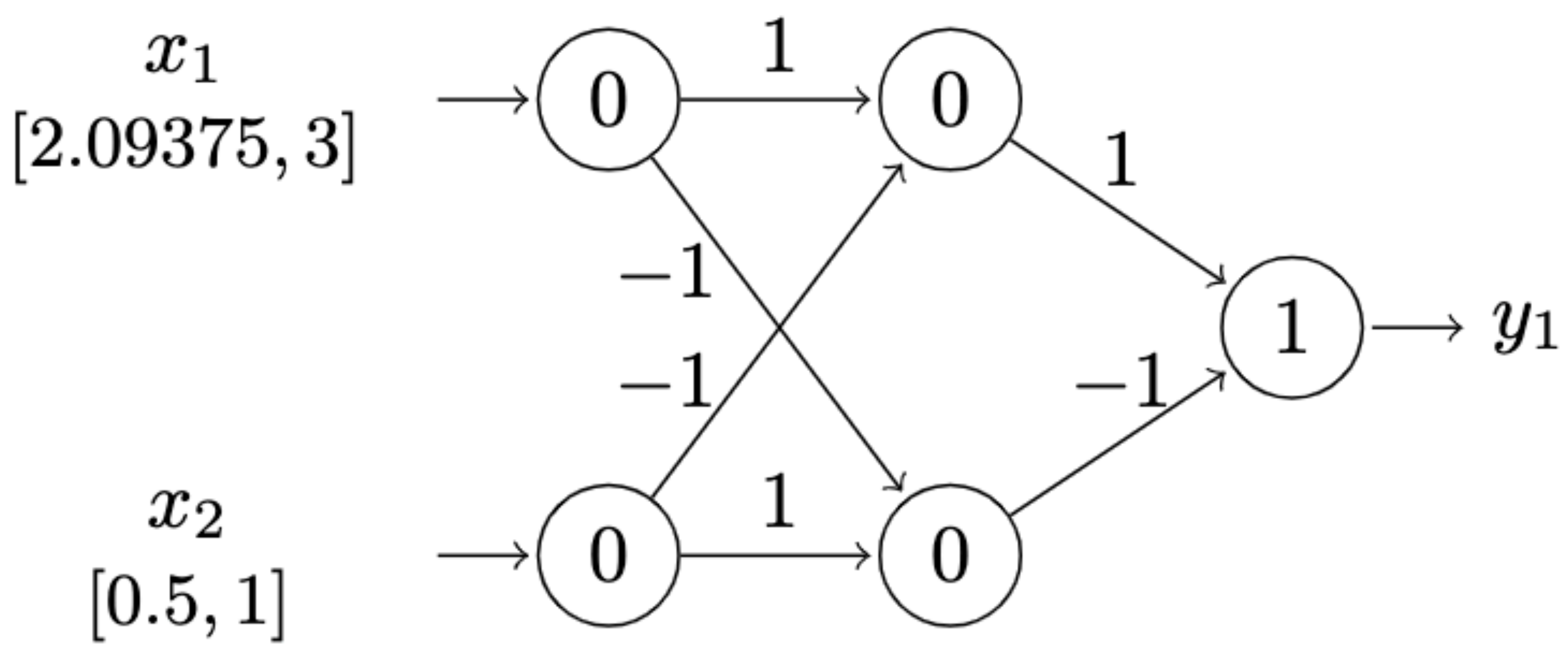
- Let $\overline{\eta} = 55$ and $F = 2$
- $\overline{\zeta} = \texttt{int}(\eta)$ where $\eta = \overline{\eta} \cdot 2^{-2} = 13.75$
- $1 - 2^{-2} \leq \texttt{offset} < 1 \rightarrow 0.75 \leq \texttt{offset} < 1$

$$\eta - \texttt{offset} \leq \overline{\zeta} \rightarrow 13 \leq \overline{\zeta}$$
$$\overline{\zeta} \leq \eta \rightarrow \overline{\zeta} \leq 13.75$$

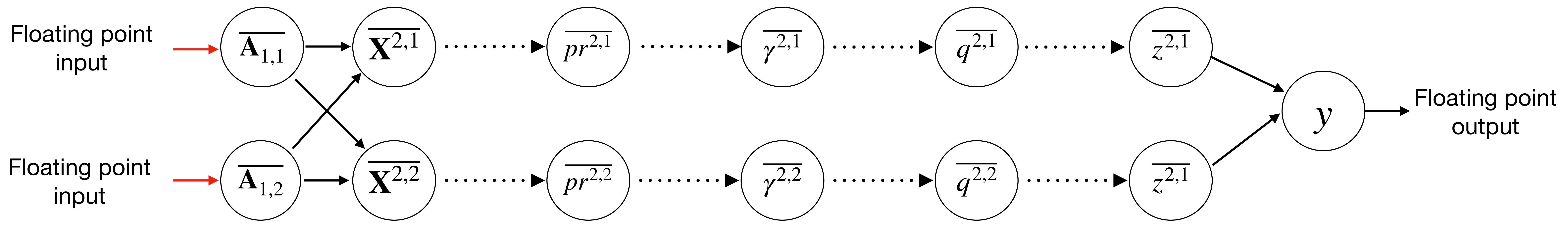
MILP Encoding of QNN



Floating-point DNN → Fixed-point DNN



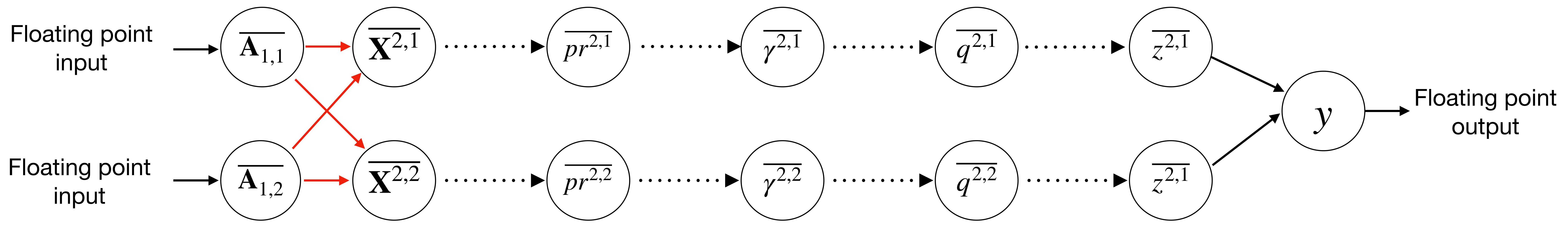
MILP Encoding of QNN - Example



- Encode input bounds

$$33 \leq \overline{\mathbf{A}_{1,1}} \leq 48$$
$$8 \leq \overline{\mathbf{A}_{1,2}} \leq 16$$

MILP Encoding of QNN - Example



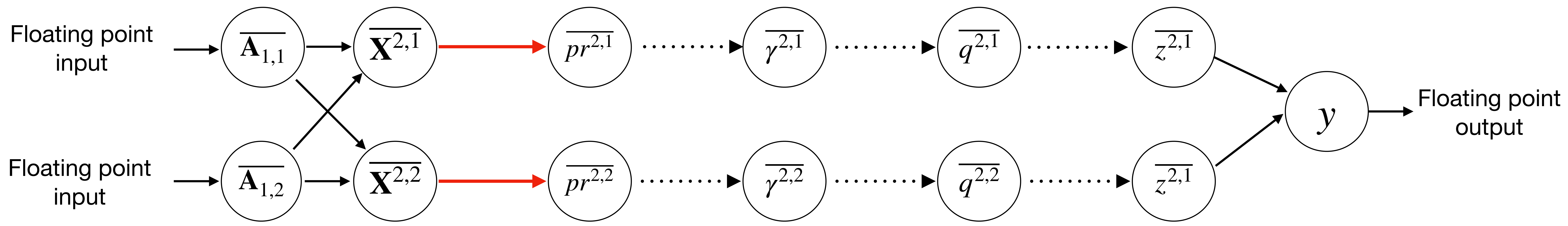
- Encode input bounds

$$33 \leq \overline{\mathbf{A}_{1,1}} \leq 48$$
$$8 \leq \overline{\mathbf{A}_{1,2}} \leq 16$$

- Copy values in $\overline{\mathbf{X}}$

$$\forall j \in [\,|T_2|\,]\,.\,\forall r \in [\,|T_1|\,]\,.\,\overline{\mathbf{X}_r^{2,j}} = \overline{\mathbf{A}_{1,r}}$$

MILP Encoding of QNN - Example



- Encode input bounds

$$33 \leq \overline{\mathbf{A}_{1,1}} \leq 48$$
$$8 \leq \overline{\mathbf{A}_{1,2}} \leq 16$$

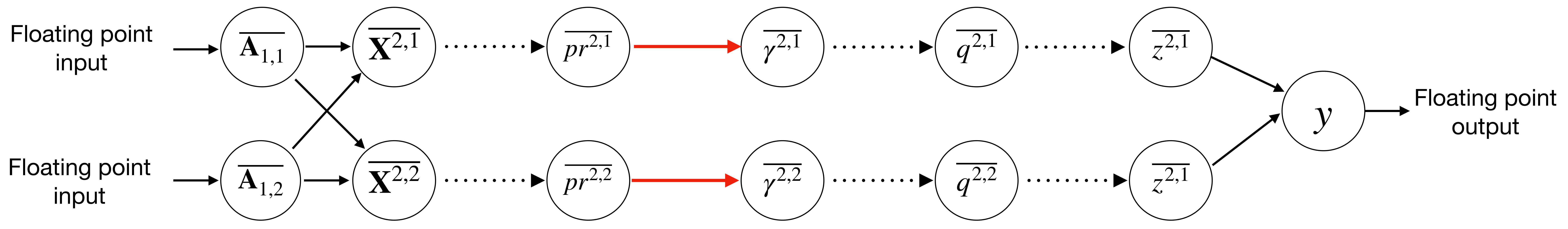
- Copy values in $\overline{\mathbf{X}}$

$$\forall j \in [\,|T_2|\,] \,.\, \forall r \in [\,|T_1|\,] \,.\, \overline{\mathbf{X}_r^{2,j}} = \overline{\mathbf{A}_{1,r}}$$

- Dot product with weights, add bias

$$\overline{pr_{2,1}} = [16, -16]^{\mathsf{T}} \cdot \overline{\mathbf{X}^{2,1}} + 0$$
$$\overline{pr_{2,2}} = [-16, 16]^{\mathsf{T}} \cdot \overline{\mathbf{X}^{2,2}} + 0$$

MILP Encoding of QNN - Example



- Encode input bounds

$$33 \leq \overline{\mathbf{A}_{1,1}} \leq 48$$
$$8 \leq \overline{\mathbf{A}_{1,2}} \leq 16$$

- Copy values in $\overline{\mathbf{X}}$

$$\forall j \in [\,|T_2|\,] \,.\, \forall r \in [\,|T_1|\,] \,.\, \overline{\mathbf{X}_r^{2,j}} = \overline{\mathbf{A}_{1,r}}$$

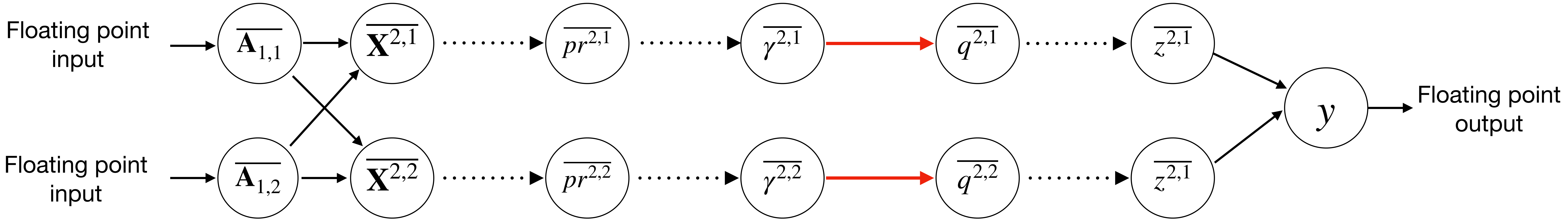
- Dot product with weights, add bias

$$\overline{pr_{2,1}} = [16, -16]^\mathsf{T} \cdot \overline{\mathbf{X}^{2,1}} + 0$$
$$\overline{pr_{2,2}} = [-16, 16]^\mathsf{T} \cdot \overline{\mathbf{X}^{2,2}} + 0$$

- Result is $Q8.8$, shift right by 4 and round down to get $Q8.4$

$$\overline{pr_{2,1}} \cdot 2^{-4} - \texttt{offset} \leq \overline{\gamma_{2,1}} \leq \overline{pr_{2,1}} \cdot 2^{-4}$$
$$\overline{pr_{2,2}} \cdot 2^{-4} - \texttt{offset} \leq \overline{\gamma_{2,2}} \leq \overline{pr_{2,2}} \cdot 2^{-4}$$

MILP Encoding of QNN - Example



- Encode input bounds

$$33 \leq \overline{\mathbf{A}_{1,1}} \leq 48$$
$$8 \leq \overline{\mathbf{A}_{1,2}} \leq 16$$

- Copy values in $\overline{\mathbf{X}}$

$$\forall j \in [\,|T_2|\,] \,.\, \forall r \in [\,|T_1|\,] \,.\, \overline{\mathbf{X}_r^{2,j}} = \overline{\mathbf{A}_{1,r}}$$

- Dot product with weights, add bias

$$\overline{pr_{2,1}} = [16, -16]^{\mathsf{T}} \cdot \overline{\mathbf{X}^{2,1}} + 0$$
$$\overline{pr_{2,2}} = [-16, 16]^{\mathsf{T}} \cdot \overline{\mathbf{X}^{2,2}} + 0$$

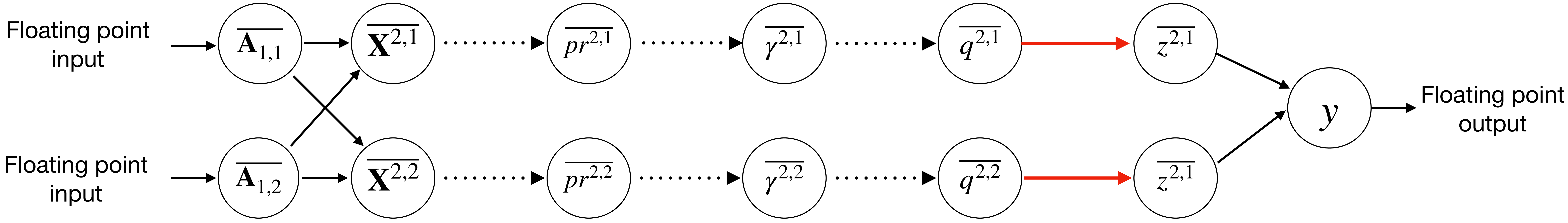
- Result is $Q8.8$, shift right by 4 and round down to get $Q8.4$

$$\overline{pr_{2,1}} \cdot 2^{-4} - \texttt{offset} \leq \overline{\gamma_{2,1}} \leq \overline{pr_{2,1}} \cdot 2^{-4}$$
$$\overline{pr_{2,2}} \cdot 2^{-4} - \texttt{offset} \leq \overline{\gamma_{2,2}} \leq \overline{pr_{2,2}} \cdot 2^{-4}$$

- Saturate the result to get $Q4.4$

$$\overline{q_{2,1}} = \min(255, \max(-256, \overline{\gamma_{2,1}}))$$
$$\overline{q_{2,2}} = \min(255, \max(-256, \overline{\gamma_{2,2}}))$$

MILP Encoding of QNN - Example



- Encode input bounds

$$33 \leq \overline{\mathbf{A}_{1,1}} \leq 48$$
$$8 \leq \overline{\mathbf{A}_{1,2}} \leq 16$$

- Copy values in $\overline{\mathbf{X}}$

$$\forall j \in [\,|T_2|\,] \,.\, \forall r \in [\,|T_1|\,] \,.\, \overline{\mathbf{X}_r^{2,j}} = \overline{\mathbf{A}_{1,r}}$$

- Dot product with weights, add bias

$$\overline{pr_{2,1}} = [16, -16]^\mathsf{T} \cdot \overline{\mathbf{X}^{2,1}} + 0$$
$$\overline{pr_{2,2}} = [-16, 16]^\mathsf{T} \cdot \overline{\mathbf{X}^{2,2}} + 0$$

- Result is $Q8.8$, shift right by 4 and round down to get $Q8.4$

$$\overline{pr_{2,1}} \cdot 2^{-4} - \texttt{offset} \leq \overline{\gamma_{2,1}} \leq \overline{pr_{2,1}} \cdot 2^{-4}$$
$$\overline{pr_{2,2}} \cdot 2^{-4} - \texttt{offset} \leq \overline{\gamma_{2,2}} \leq \overline{pr_{2,2}} \cdot 2^{-4}$$

- Saturate the result to get $Q4.4$

$$\overline{q_{2,1}} = \min(255, \max(-256, \overline{\gamma_{2,1}}))$$
$$\overline{q_{2,2}} = \min(255, \max(-256, \overline{\gamma_{2,2}}))$$

- Apply ReLU

$$\overline{z_{2,1}} = \max(0, \overline{q_{2,1}})$$
$$\overline{z_{2,2}} = \max(0, \overline{q_{2,2}})$$

Experimental Setup

Experimental Setup

- Implemented in Python using Gurobi as MILP solver (referred to as MILP)

Experimental Setup

- Implemented in Python using Gurobi as MILP solver (referred to as MILP)
- Comparison with Baranowski et al. which uses PySMT with Boolector (referred to as BVSMT)

Experimental Setup

- Implemented in Python using Gurobi as MILP solver (referred to as MILP)
- Comparison with Baranowski et al. which uses PySMT with Boolector (referred to as BVSMT)
- Comparison with Henzinger et al. which improves on BVSMT (referred to as BV2SMT)

Experimental Setup

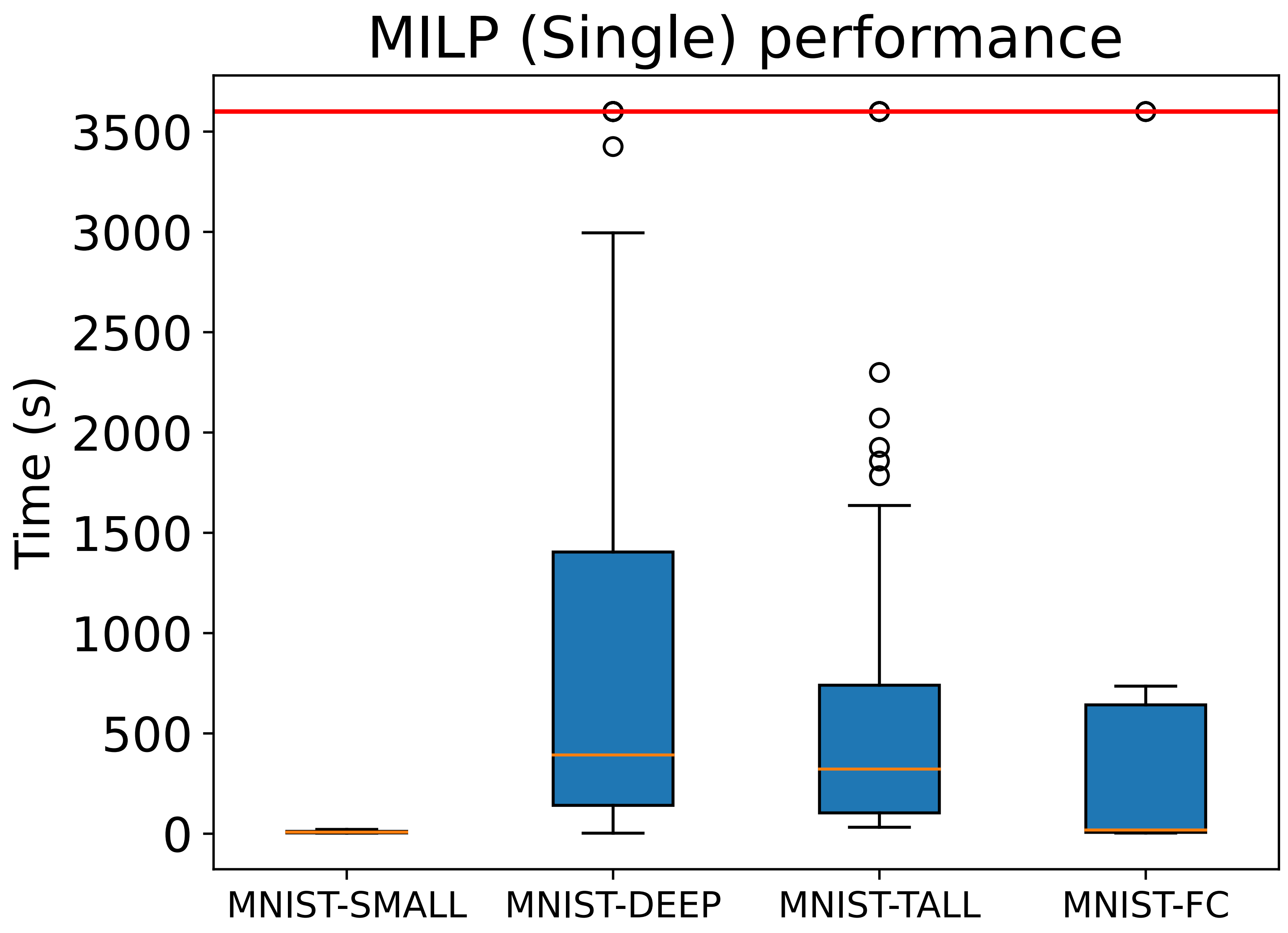
- Implemented in Python using Gurobi as MILP solver (referred to as MILP)
- Comparison with Baranowski et al. which uses PySMT with Boolector (referred to as BVSMT)
- Comparison with Henzinger et al. which improves on BVSMT (referred to as BV2SMT)
- Benchmarks - MNIST, CollisionAvoidance, TwinStream, ACAS Xu

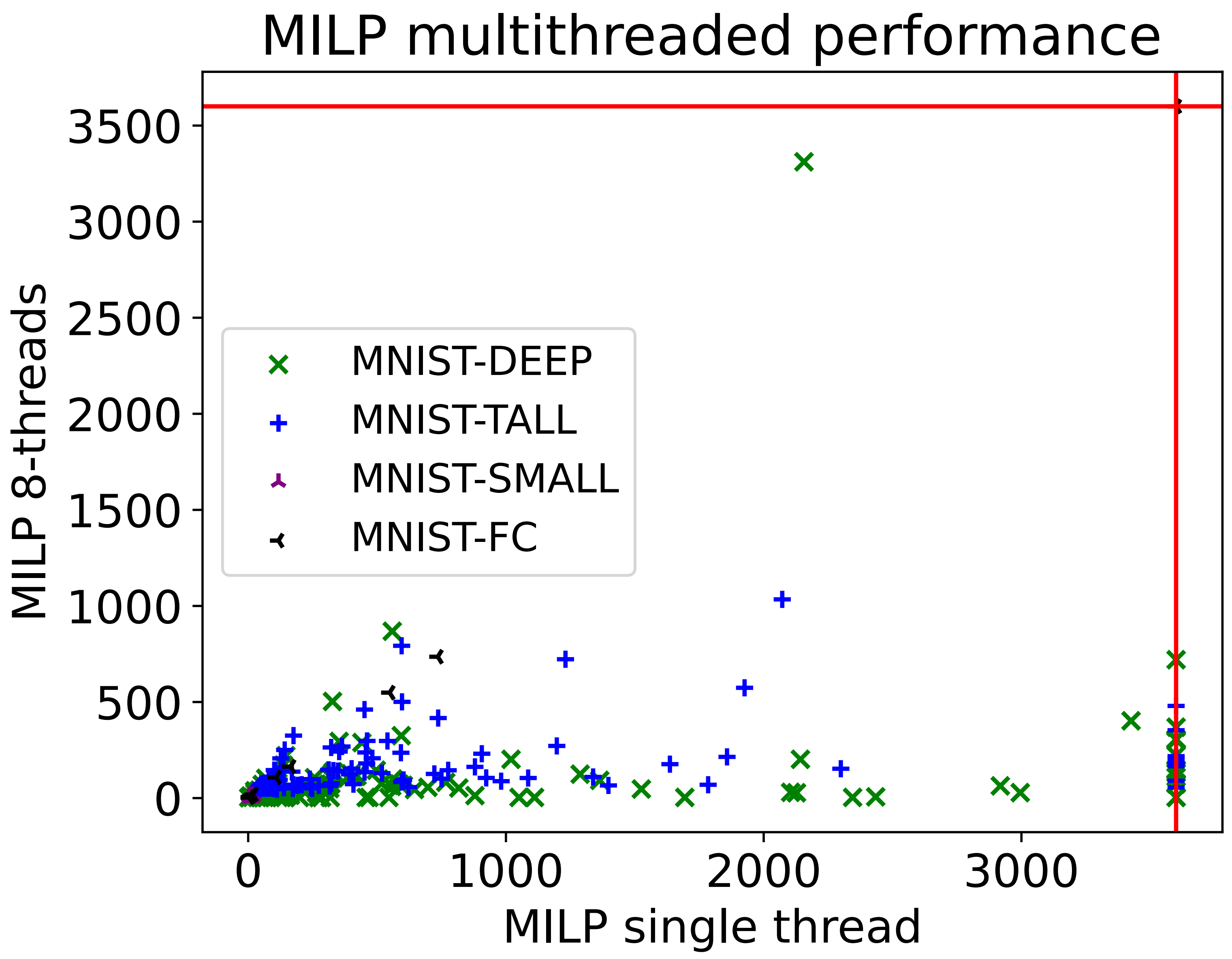
Experimental Setup

- Implemented in Python using Gurobi as MILP solver (referred to as MILP)
- Comparison with Baranowski et al. which uses PySMT with Boolector (referred to as BVSMT)
- Comparison with Henzinger et al. which improves on BVSMT (referred to as BV2SMT)
- Benchmarks - MNIST, CollisionAvoidance, TwinStream, ACAS Xu

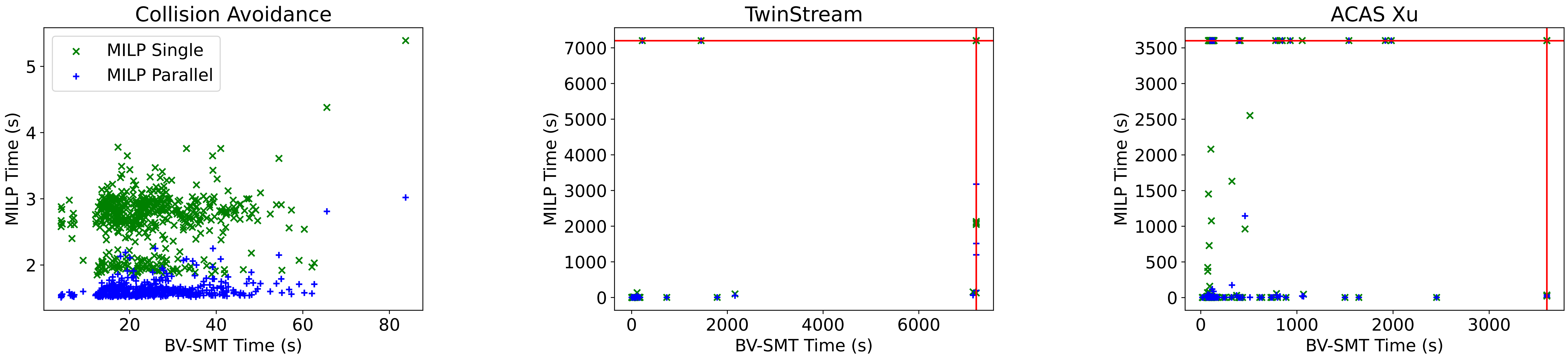
Baranowski et al. “An SMT Theory of Fixed-Point Arithmetic,” IJCAR, 2020
Henzinger et al. “Scalable Verification of Quantized Neural Networks,” AAAI, 2021

MILP vs BVSMT - MNIST





MILP vs BVSMT - CoAv, TwinStream, ACAS Xu



MILP vs BV2SMT - MNIST & Fashion MNIST

Benchmark	# Props	Time (s) (Mean | Median)		# Timeouts	
		MILP	BV2	MILP	BV2
MNIST-C	400	5.53|5.4	90|5	0	82
FASHION-C	400	5.73|5.46	49|4	0	206

Conclusion

Conclusion

- Order of magnitude improvement in standard benchmarks

Conclusion

- Order of magnitude improvement in standard benchmarks
- Makes it possible to verify larger networks than previously possible

Conclusion

- Order of magnitude improvement in standard benchmarks
- Makes it possible to verify larger networks than previously possible
- Less than satisfactory performance in some instances

Conclusion

- Order of magnitude improvement in standard benchmarks
- Makes it possible to verify larger networks than previously possible
- Less than satisfactory performance in some instances
- Code & data available at `https://github.com/iitkcpslab/QNNV`