Efficient Exact Verification of Binarized Neural Networks

July 2, 2023

目录

目录

Exact Real-valued NN Verification Challenges

- Unscalability
  - CPU inference: 2ms
  - Verification (Xiao et al., 2019): 420s
- Unverifiability
  - Floating point error makes real-valued NNs unverifiable
  - Adversarial inputs exist for verified NNs in practice
    Kai Jia and Martin Rinard. "Exploiting Verified Neural Networks via Floating Point Numerical Error." arXiv preprint arXiv:2003.03021 (2020).

Binarized Neural Networks (BNNs)

- Weights & activations constrained to be binary
- Computational benefits:
  - Efficient inference:
    - BNN: 600 GOPS/W on FPGA
    - Floating point: 50 GFLOPS/W on GPU
  - 32x reduction of memory footprint
  - Comparable classification accuracy as fp32 networks

BNN Verification

- Binary values support
  - exact logical reasoning without floating point error
  - $\implies$ exact verification with correctness guarantees
- Previous state of the art BNN verifiers are unsatisfactory
  - Less scalable than real-valued verifiers: do not scale to CNNs on CIFAR10
  - No reported evaluation of verifiable robustness

目录

Efficient SAT solving for BNN verification

BNN structure

The basic building block of a BNN is a *linear-BatchNorm-binarize* module that maps an input tensor $x \in \{0,1\}^n$ to an output tensor $y \in \{0,1\}^m$ with a weight parameter $W \in \mathbb{R}^{m \times n}$ and also trainable parameters $\gamma \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^m$ in the batch normalization [30]:

$$y = \text{bin}_{act}(\text{BatchNorm}(\text{bin}_w(W)x)) \tag{1}$$

where:

$$\text{bin}_w(W) = \text{sign}(W) \in \{-1,1\}^{m \times n}$$

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\text{BatchNorm}(x) = \gamma \odot \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} + \beta$$

where $\epsilon = 1\text{e}{-5}$, and $\odot$ denotes element-wise multiplication

$$\text{bin}_{act}(x) = (x \geq 0) = (\text{sign}(x) + 1)/2 \in \{0,1\}^m$$

- First layer: $x^{\text{q}} = \left\lfloor \frac{x}{s} \right\rceil \cdot s$
- Last layer: remove the bin_{act};no softmax;all channel use one mean and var

Efficient SAT solving for BNN verification

Encoding and Reified cardinality constraints arising in BNN encoding For

$$y = \text{bin}_{act}\left(k^{\text{BN}} \odot \left(W^{\text{bin}} x\right) + b^{\text{BN}}\right)$$

encoding as

$$y_i = \left(\sum_{j=1}^{n} l_{ij}\left(x_j\right) \gtrless \left[b_i\left(k^{\text{BN}}, W^{\text{bin}}, b^{\text{BN}}\right)\right]\right)$$

the untargeted attack goal : $\vee_{i \neq c}\left(y_i - y_c > 0\right)$
First layer: For $v = \left\lfloor \frac{x}{s} \right\rceil \in \mathbb{Z} \cap [a, b]$, $v = a + \sum_{i=1}^{b-a} t_i$, $t_i \vee \neg t_j$ for $1 \leq i < j \leq b - a$.

Efficient SAT solving for BNN verification

- Prior work:Encoding as CNF to be solved by an off-the shelf solver
- This work:natively solve such constraints by modifying a SAT solver
- Result: 100x - 10,000x speedup compared to the unmodified solver

Efficient SAT solving for BNN verification

Extend a CDCL-based SAT from MiniSat2. 2 *to* MiniSatCS.

- CDCL framework: can handle clauses not in the disjunctive form, as long as each clause permits inferring values of undecided variables.
- For $y = (\sum_{i=1}^{n} l_i \leq b)$
    - *Operand-inferring:* If y is known and enough of the $\{l_i\}$ are known, then the remaining $\{l_i\}$ can be inferred. For example, if y is known to be true and there are already b literals in $\{l_i\}$ known to be true, then the other literals must be false.
    - *Target-inferring:* If enough of the $\{l_i\}$ are known, then y can be inferred. For example, if the number of false literals in $\{l_i\}$ reaches $n - b$, then y can be inferred to be true.

Efficient SAT solving for BNN verification

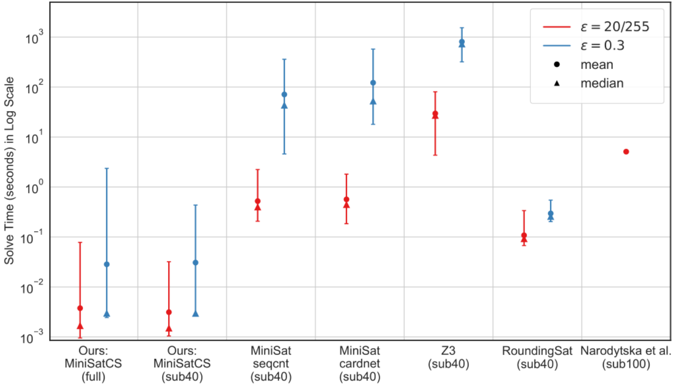

Figure 1: Performance comparison on searching adversarial inputs for an undefended MNIST-MLP network. The

目录

Balanced layer-wise sparsity

- Prior work: ternary weights, sparsity concentrates in FC layers
- This work: reparameterization with sparse(W) = bin(W) * bin(M)

$$bin_w(W) = sign(W) \circ \frac{sign(M_w) + 1}{2}$$

- Result: 100x - 100,000x verification speedup

Balanced layer-wise sparsity

Table 1: Comparing BinMask and ternary quantization on undefended `conv-small` networks. While both methods produce similar total sparsity, the more balanced layer-wise sparsity of BinMask results in faster verification. Total sparsity is the proportion of zero parameters in the whole network, which is largely determined by the sparsity of the third layer — a fully connected layer with a large weight matrix. Solve time is measured by applying `MiniSatCS` on 40 randomly chosen test images with a one hour time limit.

	MNIST $\epsilon = 0.1$		CIFAR10 $\epsilon = 2/255$	
Sparsifier	Ternary	BinMask	Ternary	BinMask
Total Sparsity	81%	84%	82%	79%
Layer-wise Sparsity	16% 40% 84% 36%	91% 90% 83% 92%	16% 48% 84% 23%	84% 85% 79% 87%
Mean Solve Time (sub40)	756.288	0.002	0.267	0.003
Max Solve Time (sub40)	3600.001	0.007	5.707	0.005
Natural Test Accuracy	97.59%	97.35%	54.78%	55.22%

Low cardinality bounds

- A regularizer to induce lower bounds in the cardinality constraints
- For $y = (\sum_{i=1}^{n} l_i \leq b)$:
    - Converts the constraint into CNF needs $O(nb)$ auxiliary variables and clauses. Thus smallerbproduces a simpler encoding.
    - MiniSatCS can infer y to be false once the number of true literals in $\{l_i\}$exceeds b, and a smallerbincreases the likelihood of this inference.
    - If the literals $\{l_i\}$ are drawn from independent symmetrical Bernoulli distributions, then the entropy of y is a symmetrical concave function with respect to b which is maximized when $b = \frac{n}{2}$. Therefore the further b deviates from $\frac{n}{2}$, the more predictable y becomes.
- by adding an L_1 penalty on the bias terms in cardinality constraints that exceed a threshold τ:

$$L^{\text{CBD}} = \eta \max\left(b\left(k^{\text{BN}}, W^{\text{bin}}, b^{\text{BN}}\right) - \tau, 0\right)$$

Low cardinality bounds

Table 2: Effect of Cardinality Bound Decay (CBD) on adversarially trained `conv-large` networks. The CBD loss effectively reduces cardinality bounds, resulting in significant verification speedup. The results suggest that it also improves robustness, perhaps by regularizing model capacity. Solve time is measured by applying `MiniSatCS` on 40 randomly chosen test images with a one hour time limit. Verifiable accuracy is evaluated on the complete test set without time limit for networks that can be verified within one second per case on average.

	MNIST $\epsilon = 0.3$				CIFAR10 $\epsilon = 8/255$			
CBD Loss Penalty (η)	0	1e−5	1e−4	5e−4	0	1e−5	1e−4	5e−4
Mean / Max Card Bound	148.3 / 364.0	4.3 / 15.1	3.2 / 9.0	2.7 / 6.8	123.8 / 312.0	3.1 / 7.4	2.7 / 5.9	2.1 / 6.4
Mean Solve Time (sub40)	2442.698	32.776	0.009	0.005	206.037	0.010	0.009	0.009
Max Solve Time (sub40)	3600.006	1287.739	0.040	0.012	3600.001	0.019	0.013	0.014
Verifiable Accuracy	-	-	69.04%	72.48%	-	19.28%	18.81%	20.08%
Natural Test Accuracy	98.88%	97.37%	96.97%	96.26%	53.91%	40.80%	38.75%	35.17%
PGD Accuracy	89.23%	87.60%	87.82%	87.82%	15.32%	27.06%	26.22%	24.69%
First Layer / Total Sparsity	85% / 82%	86% / 89%	83% / 89%	82% / 87%	95% / 88%	95% / 94%	94% / 87%	89% / 90%

目录

Training verifiably robust BNNs

- Prior work:robust training for real-valued networks (e.g., PGD)
- Challenge: A BNN directly trained with PGD has high PGD accuracy, low verifiable accuracy $\implies$ PGD attack becomes ineffective.
- This work: adaptive gradient cancelling: $htanh'(x) \rightarrow tanh'(x) \rightarrow 1 - tanh^2(ax)$,a is tuned by improving PGD attack success rate

Training verifiably robust BNNs

Table 3: Comparing gradient computing methods for adversarial training on CIFAR10 with the `conv-large` network and $\epsilon = 8/255$. The PGD accuracy is evaluated on the test set with the same gradient computing as in training. The verifiable accuracy is the exact adversarial robustness. Tanh gradient cancelling improves all the metrics, and adaptive gradient cancelling reduces the gap between PGD accuracy and verifiable accuracy.

	hard tanh	tanh	adaptive	adaptive + verifier adv
Natural Test Accuracy	35.42%	38.79%	35.17%	35.00%
PGD Accuracy	22.79%	24.98%	24.69%	26.41%
Verifiable Accuracy	11.13%	14.70%	20.08%	22.55%

目录

Experiment

Table 4: Results of verifying adversarial robustness. EEV verifies BNNs significantly faster with comparable verfiable accuracy in most cases.

		Mean Time (s)			Accuracy			Timeout
		Build	Solve	Total	Verifiable	Natural	PGD	
MNIST $\epsilon = 0.1$	EEV S	0.0158	0.0004	0.0162	89.29%	97.44%	93.47%	0
	EEV L	0.1090	0.0025	0.1115	91.68%	97.46%	95.47%	0
	Xiao et al. [63] S	4.98	0.49	5.47	94.33%	98.68%	95.13%	0.05%
	Xiao et al. [63] L*	156.74	0.27	157.01	95.6%	98.95%	96.58%	0
MNIST $\epsilon = 0.3$	EEV S	0.0140	0.0006	0.0146	66.42%	94.31%	80.70%	0
	EEV L	0.1140	0.0039	0.1179	77.59%	96.36%	87.90%	0
	Xiao et al. [63] S	4.34	2.78	7.12	80.68%	97.33%	92.05%	1.02%
	Xiao et al. [63] L*	166.39	37.45	203.84	59.6%	97.54%	93.25%	24.1%
CIFAR10 $\epsilon = \frac{2}{255}$	EEV S	0.0258	0.0013	0.0271	26.13%	46.58%	33.70%	0
	EEV L	0.1653	0.0097	0.1750	30.49%	47.35%	38.22%	0
	Xiao et al. [63] S	52.58	13.50	66.08	45.93%	61.12%	49.92%	1.86%
	Xiao et al. [63] L*	335.97	29.88	365.85	41.4%	61.41%	50.61%	9.6%
CIFAR10 $\epsilon = \frac{8}{255}$	EEV S	0.0313	0.0014	0.0327	18.93%	37.75%	24.60%	0
	EEV L	0.1691	0.0090	0.1781	22.55%	35.00%	26.41%	0
	Xiao et al. [63] S	38.34	22.33	60.67	20.27%	40.45%	26.78%	2.47%
	Xiao et al. [63] L*	401.72	20.14	421.86	19.8%	42.81%	28.69%	5.4%

EEV is exact verification for BNNs with the proposed EEV system. Xiao et al. [63] is exact verification for real-valued networks with data taken from [63]. The S and L suffix indicates `conv-small` or `conv-large` architectures. The build time is the time required to generate the SAT or MILP formulation from the network weights and the input

Experiment

Table 8: Comparison of methods on 40 randomly chosen MNIST test images with solving time limit of 3600 seconds.

ϵ_{train}	Network Architecture	Training Method	Test Accuracy	Sparsity	Solver	Mean Solve Time	Median Solve Time	Timeout	Verifiable Accuracy
0	conv-small	Ternary	97.59%	81%	MiniSatCS	756.288	4.281	15%	0%
		BinMask	97.35%	84%	MiniSatCS	0.002	0.002	0	52%
					MiniSat	2.249	1.142	0	52%
					Z3	0.089	0.089	0	52%
					RoundingSat	0.048	0.042	0	52%
	conv-large	Ternary	99.07%	86%	MiniSatCS	2522.082	3600.002	68%	0%
		Ternary+CBD	95.58%	87%	MiniSatCS	886.007	21.711	20%	0%
		Ternary+10xCBD	92.91%	78%	MiniSatCS	342.097	4.742	5%	2%
		BinMask	98.94%	86%	MiniSatCS	2595.032	3600.001	70%	2%
		BinMask+CBD	96.88%	89%	MiniSatCS	0.664	0.028	0	70%
					MiniSat	225.861	18.761	0	70%
					Z3	146.567	0.997	0	70%
					RoundingSat	33.922	0.702	0	70%
	conv-small	Ternary (wd0)	94.72%	80%	MiniSatCS	186.935	0.105	5%	30%
		Ternary (wd1)	89.53%	93%	MiniSatCS	0.005	0.002	0	35%
					MiniSatCS	0.001	0.001	0	52%
					MiniSat	0.060	0.024	0	52%

Experiment

Table 9: Comparison of methods on 40 randomly chosen CIFAR10 test images with solving time limit of 3600 seconds.

ϵ_{train}	Network Architecture	Training Method	Test Accuracy	Sparsity	Solver	Mean Solve Time	Median Solve Time	Timeout	Verifiable Accuracy
0	`conv-small`	Ternary	54.78%	82%	MiniSatCS	0.267	0.006	0	0%
					MiniSat	327.303	50.036	7%	0%
					Z3	411.638	117.604	5%	0%
					RoundingSat	0.361	0.098	0	0%
		BinMask	55.22%	79%	MiniSatCS	0.003	0.003	0	0%
					MiniSat	3.981	3.577	0	0%
					Z3	0.590	0.376	0	0%
					RoundingSat	0.081	0.077	0	0%
	`conv-large`	Ternary	69.25%	89%	MiniSatCS	823.370	1.860	20%	0%
		BinMask	67.46%	94%	MiniSatCS	300.404	3.201	5%	0%
		BinMask+CBD	63.18%	88%	MiniSatCS	1.415	0.048	0	0%
					MiniSat	168.079	69.471	0	0%
					Z3	3515.386	3600.121	92%	0%
					RoundingSat	19.162	0.858	0	0%
	`conv-small`	Ternary	32.59%	95%	MiniSatCS	0.002	0.002	0	15%
		BinMask	37.75%	96%	MiniSatCS	0.001	0.002	0	18%
					MiniSat	0.070	0.082	0	18%
					Z3	0.050	0.052	0	18%
					RoundingSat	0.033	0.043	0	18%

Thank you