We thank the reviewers for providing those helpful comments, especially during the challenging time this year. Below are our responses to individual reviewers:

Reviewer #1

> All the empirical results are for small examples (CIFAR-10) raising the question of scalability.

We understand the concern. We note that previous complete verification methods (for either binarized or real-valued neural networks) report results only for similarly sized networks and datasets and the results show that our work significantly improves the scalability of complete BNN verification in comparison with all previous methods.

Reviewer #2

> There exists successful solvers already in the pseudo-Boolean and constraint programming research that can handle > cardinality constraints without the use of auxiliary variables/constraints.

BNN verification requires solving reified cardinality constraints of the form $y \leftrightarrow (\sum_{i=1}^{n} x_i \geq p)$, NOT cardinality constraints of the form $\sum_{i=1}^{n} x_i \geq p$. We are the first to support reified cardinality constraints natively in the SAT solver. We cite previous work on SAT solver support for cardinality constraints (Liffiton et al. [35]) and are happy to cite additional research as suggested by the reviewer. However, none of this research can solve BNN verification problems due to their requirement for reified constraints and is therefore not a comparison baseline for our technique. We do compare with Z3, which supports general pseudo-Boolean constraints including reified cardinality constraints, specifically via on-demand compilation into sorting circuits. The results show that our native support for reified cardinality constraints significantly outperforms Z3.

> In the planning with BNNs research by [1], cardinality networks have been extended to reified cardinality constraints, > and have been shown to perform faster compared to sequential counter-based encodings.

Instead of natively supporting reified cardinality constraints, [1] encodes such constraints using cardinality networks. This encoding requires $O(n \log_2^2 p)$ auxiliary variables/clauses. Our research, in contrast, modifies the SAT solver to natively handle reified cardinality constraints with no auxiliary variables/clauses. We implemented the encoding of [1] and tested it on three MNIST networks: `MNIST-MLP`, `conv-small`, and `conv-large` ($\eta = 5\mathrm{e} - 4$). The results show that, with native support for reified cardinality constraints, our solver delivers speedups of 150.5x, 55.3x, and 90.2x compared to the encoding of [1] in three cases respectively. We are happy to include related discussion and results in the next version. Note that [1] deals with planning, not BNN verification, and was published after the NeurIPS deadline.

> There are more compact and efficient encodings for encoding cardinality constraints compared to sequential counters.

We use sequential counters only for comparison with other BNN verification research (e.g. Narodytska et al. [43]), which uses sequential counters. We advocate native support for reified cardinality constraints in the SAT solver, not encoding reified constraints using sequential counters, sorting networks, or cardinality networks as in [1], and the results show that our approach significantly outperforms all of these encodings.

> The main contribution of the paper is the extension of an existing SAT solver (i.e., MiniSAT) to a SAT solver that can > handle (reified) cardinality constraints.

We make several main contributions, including efficient SAT solving, training solver-friendly BNNs via inducing two properties, and training robust BNNs via adaptive gradient cancelling. Their working together is required to obtain the performance numbers in the paper — while our novel direct support for reified cardinality constraints is a critical contribution with potential applications to other SAT problems, it is one of three main contributions in the paper.

> There also exists a (preliminary) work on training BNNs with constraint programming that uses low amounts of data > by Icarte et al. 2019 - which you can use as a reference for training sparse BNNs.

Icarte et al. 2019 focuses on generalizability in few-shot learning, which is a different setting than ours. Specifically, they train BNNs of 0, 1, or 2 hidden layers with 16 neurons each, on no more than 10 training examples per class of MNIST with a two-hour time limit, while our research works with larger networks and more data. We are happy to reference and discuss Icarte et al. 2019 in the next version.

Reviewer #3

> In section 4.1, how is the proposed encoding different from [42] and how do they compare in terms of speed?

We use quantized floating point input (vs binary input in [42]), constrain activations to $\{0, 1\}$ (vs $\{-1, 1\}$ in [42]), and include a modified BN in the last layer (vs no last layer BN in [42]). We use [43], which is an improved version of [42], as a baseline for comparison and have demonstrated significant speedup (Figure 1). We will clarify in the next version.

Reviewer #4

> Did you try other values for τ?

We tried multiple values for τ (such as 3, 5, 10). $\tau = 5$ was a good choice that enables fine control of the accuracy-speed tradeoff by tuning η. Thanks for the feedback, and we will clarify and discuss in the next version.

References

[1] Buser Say and Scott Sanner. Compact and efficient encodings for planning in factored state and action spaces with learned binarized neural network transition models. *Artificial Intelligence*, page 103291, 2020.