Learning for Integer-Constrained Optimization through Neural Networks with Limited Training

Zhou Zhou
Virginia Tech
zhouzhou@vt.edu

Shashank Jere
Virginia Tech
shashankjere@vt.edu

Lizhong Zheng
MIT
lizhong@mit.edu

Lingjia Liu
Virginia Tech
ljliu@vt.edu

Abstract

In this paper, we investigate a neural network-based learning approach towards solving an integer-constrained programming problem using very limited training. Specifically, we introduce a symmetric and decomposed neural network structure, which is fully interpretable in terms of the functionality of its constituent components. By taking advantage of the underlying pattern of the integer constraint, as well as of the affine nature of the objective function, the introduced neural network offers superior generalization performance with limited training, as compared to other generic neural network structures that do not exploit the inherent structure of the integer constraint. In addition, we show that the introduced decomposed approach can be further extended to semi-decomposed frameworks. The introduced learning approach is evaluated via the classification/symbol detection task in the context of wireless communication systems where available training sets are usually limited. Evaluation results demonstrate that the introduced learning strategy is able to effectively perform the classification/symbol detection task in a wide variety of wireless channel environments specified by the 3GPP community.

1 Introduction

Integer-constrained optimization has wide applications in many fields, including industrial production planning, vehicle scheduling in transportation networks, and resource allocation for cellular networks where the variables often represent finite decisions and countable quantities [1, 2]. In this paper, we consider developing a neural network (NN)-based solver for the integer-constrained programming problem of the following form:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f\left(\boldsymbol{H}\boldsymbol{x}-\boldsymbol{y}\right) \\ \text{subject to} & x_n \in \mathcal{A} \end{array}, \tag{1}$$

where $\boldsymbol{H} \in \mathbb{R}^{Q\times N}$; $\boldsymbol{y} \in \mathbb{R}^{Q\times 1}$ contains observed values; $f(\cdot)$ is the objective function; $\boldsymbol{x} \in \mathbb{R}^{N\times 1}$ is the unknown vector to be estimated (x_n is the n^{th} element of $\boldsymbol{x}$); $\mathcal{A}$ is the set of the limited integer values each x_n can take. Without loss of generality (WLOG), $\mathcal{A}$ is set to be $\{-2M-1, -2M+1, \cdots, -1, 1, \cdots, 2M-1, 2M+1\}$. This integer-constrained definition of set $\mathcal{A}$ can be generalized to any arbitrary integers. Unlike unfolding an optimization-based algorithm to a deep neural network [3], our approach is motivated by the intrinsic geometry of the integer-constraint $\mathcal{A}$, and the embedded affine-mapping $\boldsymbol{H}$ in the argument of the objective function.

Integer-constrained optimization problems are generally NP-hard [2] due to the non-convex nature of the set $\mathcal{A}^N$. An exhaustive search can be prohibitively expensive, especially when N is large. Therefore, rather than focusing on conventional optimization-based approaches, we consider a data-driven or learning-based classification/detection framework where the structural information that is inherent in the constraint can be utilized to improve the learning "efficiency" in terms of the training data overhead and "effectiveness" in terms of the generalization performance.

The methodology of our introduced neural network based solver is as follows:

- We first define a probability residual-model for $\boldsymbol{r} = \boldsymbol{H}\boldsymbol{x} - \boldsymbol{y}$ where $\boldsymbol{r}$ is generated according to a distribution $P(\boldsymbol{r})$ which is determined based on the objective function $f(\cdot)$[1].
- Then, we generate K tuples of $(\boldsymbol{x}, \boldsymbol{r})$ according to the distributions $U(\mathcal{A}^N)$ and $P(\boldsymbol{r})$ respectively, where $U(\mathcal{A}^N)$ represents the uniform distribution defined on $\mathcal{A}^N$. The resulting tuples can be organized as $\left\{ \left(\boldsymbol{x}^{(1)}, \boldsymbol{r}^{(1)}\right), \left(\boldsymbol{x}^{(2)}, \boldsymbol{r}^{(2)}\right), \cdots, \left(\boldsymbol{x}^{(K)}, \boldsymbol{r}^{(K)}\right)\right\}$.

  For a fixed $\boldsymbol{H}$, $\boldsymbol{y}$ is generated using $\boldsymbol{x}$ and $\boldsymbol{r}$ via the residual relation $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} - \boldsymbol{r}$. For the purpose of training a neural network, we have K tuples of $(\boldsymbol{x}, \boldsymbol{y})$: $\left\{ \left(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}\right), \left(\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}\right), \cdots, \left(\boldsymbol{x}^{(K)}, \boldsymbol{y}^{(K)}\right)\right\}$.
- We train a neural network $\mathcal{N}$ using the K tuples defined, where $\boldsymbol{y}^{(k)}$ is the input to $\mathcal{N}$ and $\boldsymbol{x}^{(k)}$ is the associated output.
- After $\mathcal{N}$ is trained, we feed the observation vector $\boldsymbol{y}$ to $\mathcal{N}$ to generate a solution, $\hat{\boldsymbol{x}}$, of (1).

To provide a meaningful and practical solution for a large dimensional $\boldsymbol{x}$, we develop a *parallel* solver in which N independent neural networks simultaneously estimate each entry of $\boldsymbol{x}$. This is accomplished by leveraging the concept of marginal estimation, tailored specifically to our chosen probability model for the residue $\boldsymbol{r}$. This decomposed approach can also be extended to obtain a semi-decomposed framework, thus providing a middle ground between this fully decomposed approach and a fully joint approach. By incorporating structural information such as the affine mapping $\boldsymbol{H}$ as well as the step size between elements in $\mathcal{A}$ into our neural network design, superior generalization performance is achieved with very limited training samples.

In summary, the key contributions of this paper are summarized as follows:

- The introduced neural network architecture is symmetric and its structure is fully interpretable. The neural network aims to learn the marginal probability distribution function (PDF) of each variable.
- With a limited amount of training, the introduced learning approach can achieve better generalization performance than that of generic neural networks having an arbitrary structure.
- A theoretical upper-bound for the generalization error of the introduced neural network.

2 Structure-Aware Learning

Based on the probability model presented in the training set ,we first consider the joint posterior probability distribution. According to Bayes's rule, the joint posterior probability distribution is,

$$p\left(x_1, x_2, \cdots, x_N|\boldsymbol{y}\right) = p\left(x_1|\boldsymbol{y}\right) p\left(x_2|x_1, \boldsymbol{y}\right) \cdots p\left(x_N|x_1, \cdots, x_{N-1}, \boldsymbol{y}\right)$$

To facilitate a decomposed approach for the maximum a posteriori estimation (MAP), we utilize the following naive Bayesian approximation:

$$\arg \max_{x_1, x_2, \cdots, x_N} p(x_1, x_2, \cdots, x_N|\boldsymbol{y}) \approx \arg \max_{x_1, x_2, \cdots, x_N} p(x_1|\boldsymbol{y}) p(x_2|\boldsymbol{y}) \cdots p(x_N|\boldsymbol{y}),$$

i.e., the joint MAP estimator can be approximated as the product of the marginal MAP estimators.

2.1 Binary Integer Constraint

We first assume that the integer constraint is only restricted to a binary set, i.e., $\mathcal{A} = \{-1, 1\}$. We can easily learn a neural network to approximate $p(x_n|\boldsymbol{y})$, where the input of the neural network is $\boldsymbol{y}$ and the output is the probability mass of x_n. Alternatively, the output can be denoted as the likelihood ratio (LR) between hypotheses $\{H_0 : x_n = 1\}$ and $\{H_1 : x_n = -1\}$ as

$$L_{+-}\left(\boldsymbol{y}\right) := \frac{p\left(x_n = 1|\boldsymbol{y}\right)}{p\left(x_n = -1|\boldsymbol{y}\right)}. \tag{2}$$

[1]For instance, the Gaussian distribution can be used for $P(\boldsymbol{r})$ for L2-norm minimization.

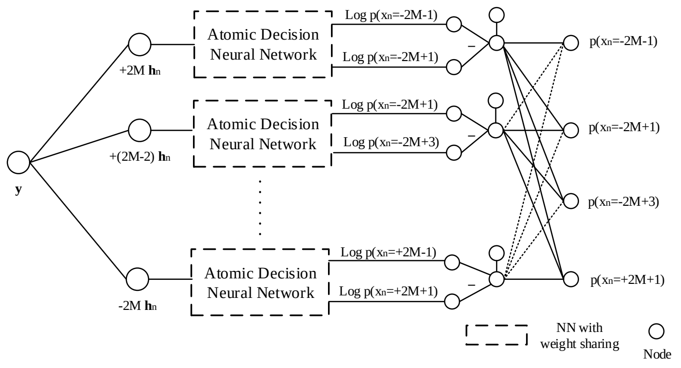

Figure 1: The neural network structure for the posterior estimation of general integer constraint.

2.2 General Integer Constraint

Next, we consider the extension of the binary constraint to a general integer constraint, i.e., $\mathcal{A} = \{-2M-1, -2M+1, \cdots, -1, 1, \cdots, 2M-1, 2M+1\}$. Leveraging the pattern of this integer constraint, the neural network $\mathcal{N}_n$ is designed as shown in Fig. 1, where the "atomic decision neural network" (ADNN) is the same as the binary integer classifier/ MAP estimator introduced in Section 2.1. We see that the input of each ADNN is a "shifted" version of the observed signal vector $\boldsymbol{y}$, that is shifted in the direction of $\boldsymbol{h}_n$, (i.e., the n^{th} column of $\boldsymbol{H}$). The output ratio represents an estimate of the LR between the hypotheses $\{H_0^{(-2m)} : x_n = -2m+1\}$ and $\{H_1^{(-2m)} : x_n = -2m-1\}$, denoted as $L_{+-}^{(2m)}$. Intuitively, this shifting process is motivated by the fact that

$$L_{+-}^{(2m)}(\boldsymbol{y}) = L_{+-}(\boldsymbol{y} + 2m\boldsymbol{h}_n). \tag{3}$$

Once the LRs representing all the pairwise boundary decision probabilities in $\mathcal{A}$ are obtained, the posterior marginal estimation of x_n can be constructed as follows:

$$p(x_n = -2m+1|\boldsymbol{y}) = p(x_n = -2M-1|\boldsymbol{y}) \prod_{m'=M}^{m} L_{+-}^{(-2m')}(\boldsymbol{y}).$$

This chain connection corresponds to the last layer of the neural network structure depicted in Fig. 1. According to the discussion in Section 2.1, we know the ADNN training set is based on a binary integer target. In order to create a similar training set for the ADNN with an arbitrary integer constraint, we introduce the following training set, $\left\{\left(+1, \tilde{\boldsymbol{y}}_+^{(1)}\right), \left(-1, \tilde{\boldsymbol{y}}_-^{(1)}\right), \cdots, \left(+1, \tilde{\boldsymbol{y}}_-^{(K)}\right), \left(-1, \tilde{\boldsymbol{y}}_-^{(K)}\right)\right\}$, where $\tilde{\boldsymbol{y}}_+^{(k)} = \boldsymbol{y}^{(k)} - x_n^{(k)}\boldsymbol{h}_n + \boldsymbol{h}_n$, $\tilde{\boldsymbol{y}}_-^{(k)} = \boldsymbol{y}^{(k)} - x_n^{(k)}\boldsymbol{h}_n - \boldsymbol{h}_n$.

2.3 Generalization Performance Characterization

By employing the ADNN, we can approximate the posterior PDF for any combination of variables. Accordingly, we can analyze how the binary decision error of the ADNN can impact the classification performance of the entire neural network. Theorem 1 introduces an upper bound on the generalization error (probability of symbol error) of the entire neural network depicted in Fig. 1.

Theorem 1 *Let $\mathcal{H}_{\mathcal{A}}$ be the class of all atomic decision neural networks (ADNNs) and $\mathcal{N}_{\mathcal{A}}$ be a particular ADNN. With cross-entropy used as the loss function $\ell(\cdot)$, the generalization error is bounded as follows:*

$$\Pr(e) \le \left(\left(1 - \frac{1}{4M}\right)\left(2\mathcal{R}_K(\mathcal{H}_{\mathcal{A}}) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2K}} - \mathcal{L}(\mathcal{N}_{\mathcal{A}})\right)\right)^{\rho}$$

with probability at least δ, $\delta \in (0,1)$, where $\mathcal{R}_K$ is the empirical Rademacher complexity, $\mathcal{L}_K(\mathcal{N}_\mathcal{A}) = \frac{1}{K}\sum_k \mathbb{1}(x_k = +1)\log(1-\alpha_k) + \mathbb{1}(x_k = -1)\log(1-\beta_k)$, $\mathbb{1}$ represents the indicator function, $1-\alpha_k$ and $1-\beta_k$ is the ADNN outputs for $x_k = +1$ and $x_k = -1$ respectively.

3 Case Study – Symbol Detection in MIMO Systems

For performance evaluation of the introduced structured neural network structure, we consider the task of classification/symbol detection in a multi-input-multi-output (MIMO) wireless communications system. This task of symbol detection in a wireless communications context can be viewed as a multi-label classification problem with the constraint of limited training set availability, due to the physical implementation limitations of a real-world communications system. In this task, $\boldsymbol{H}$ is the wireless transmission channel which often satisfies a certain probability distribution and $\boldsymbol{y}$ is the signal at the receiver. The introduced solver is then utilized to detect the transmit symbol of interest, $\boldsymbol{x}$, based on different objective functions f.

3.1 Comparison with generic neural network architectures

Three alternate neural network structures are developed as our benchmark for the generalization performance comparison. Together with our introduced neural network structure, these four networks are respectively denoted as A-Net, B-Net, C-Net and D-Net as shown in the Appendix.

Table 1: The testing and training success rate of the four neural networks (one trial is counted as success when the global optimum is found) for $f = \sqrt{\sum_n x_n^2}$ and $f = \sum_n \log(1 + x_n^2/\nu)$, where the training set size K is set as 30, whereas the testing set size is 30000. (The notation $\sim$ represents "from to")

	$f(\cdot) = \|\cdot\|_2$		$f(\cdot) = \sum_n \log(1+(\cdot)_n^2/\nu)$	
Method	Train Success Rate	Test Success Rate	Train Success Rate	Test Success Rate
A-Net	> 0.99	$0.80 \sim 1.00$	> 0.99	$0.71 \sim 0.90$
B-Net	> 0.99	$0.70 \sim 0.99$	> 0.99	$0.23 \sim 0.50$
C-Net	> 0.99	$0.30 \sim 0.89$	> 0.99	$0.63 \sim 0.75$
D-Net	> 0.99	< 0.43	> 0.99	< 0.3

3.2 Comparison with State-of-the-Art Methods

For a complete comparison with state-of-the-art, the performance of the introduced structure-aware decomposition-based neural network (A-Net) is compared with benchmark conventional optimization methods. These include gradient-free optimization methods implemented from the open-source 'NLOpt' package [4], genetic algorithm [5, 6] and *DetNet* [7], based on unfolding iteration-based optimization algorithms. The results from this comparison are available in Table 2 in the Appendix.

4 Conclusion

In this paper, a neural network based-approach to solve an integer-constrained programming problem was presented. The introduced neural network architecture leverages the following two features of the optimization problem (1) : (i) the lattice structure of the integer constraint $\mathcal{A}$ (ii) the affine mapping inside the composition-type objective function $f(\boldsymbol{Hx} - \boldsymbol{y})$. Furthermore, the objective function is translated to its corresponding probabilistic model to build the data for the neural network training. In addition, a generalization performance bound of the neural network was also derived. Experimental results verified the superior generalization ability of this method with limited training. It was also observed that the introduced learning-based solver outperforms most of the state-of-the art generic integer-constraint solvers. The introduced structure-aware neural network-based method is a promising direction towards solving NP-hard integer-constrained optimization problems.

Acknowledgements

The work of Z. Zhou, S. Jere, and L. Liu is supported in part by National Science Foundation (NSF) under grants CCF-1937487 & CNS-2003059 and Intel Corporation. L. Zheng is supported in part by the Office of Naval Research (ONR) under MURI Grant N00014-19-1-2621, NSF under grant CNS-2002908, and Intel Corporation. The corresponding author is L. Liu (ljliu@vt.edu).

References

[1] Fred Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers operations research*, 13(5):533–549, 1986.

[2] Raymond Hemmecke, Matthias Köppe, Jon Lee, and Robert Weismantel. Nonlinear Integer Programming. In *50 Years of Integer Programming 1958-2008*, pages 561–618. Springer, 2010.

[3] John R Hershey, Jonathan Le Roux, and Felix Weninger. Deep Unfolding: Model-based Inspiration of Novel Deep Architectures. *arXiv preprint arXiv:1409.2574*, 2014.

[4] Steven Johnson. The nlopt nonlinear-optimization package. *Software Package*, 2020.

[5] A.H. Wright, M.D. Vose, K.A. De Jong, and L.M. Schmitt. *Foundations of Genetic Algorithms*, chapter 1, pages 1–52. John Wiley Sons, Ltd, 2007.

[6] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[7] Neev Samuel, Tzvi Diskin, and Ami Wiesel. Learning to detect. *IEEE Transactions on Signal Processing*, 67(10):2554–2564, 2019.

[8] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[9] Bruce Hajek and Maxim Raginsky. Statistical learning theory. *Lecture Notes*, 387, 2019.

[10] T Rowan. : Functional stability analysis of numerical algorithms. in: Ph. d. thesis, department of computer sciences, university of texas at austin, 1990. 1990.

[11] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[12] Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[13] Michael JD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

[14] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79(1):157–181, 1993.

[15] Joerg M Gablonsky and Carl T Kelley. A locally-biased form of the direct algorithm. *Journal of Global Optimization*, 21(1):27–37, 2001.

Appendix

Proof of Theorem 1

By employing the ADNN, the posterior probability density function (PDF) can be approximated for any combination of variables. Accordingly, we can analyze how the binary decision error of the ADNN impacts the classification performance of the entire neural network.

Lemma 1 *Assume that the binary decision error of the ADNN is characterized by the Type-I error α and the Type-II error β, defined as:*

$$\alpha = \Pr\left(L_{+-}(\boldsymbol{y}) < 1 | x_n = +1\right)$$
$$\beta = \Pr\left(L_{+-}(\boldsymbol{y}) > 1 | x_n = -1\right).$$

Then, the decision error for the entire neural network shown in Fig. 1 is bounded by

$$\Pr(e) \leq \left(\left(1 - \frac{1}{4M}\right)(\alpha + \beta)\right)^{\rho}, \tag{4}$$

where $\rho \in [0, 1]$, M is defined in the description of the set $\mathcal{A}$.

The above lemma can be directly obtained by applying the union bound [8] to the probability of symbol error $\Pr(e)$. As an example of a numerical evaluation of this result, assume that $\alpha = 0.05$, $\beta = 0.05$, $M = 1$ and $\rho = 1$. Then, the probability of error is upper bounded by $\Pr(e) < 0.075$ which is only marginally higher than either α or β, the binary decision errors of the individual ADNN. This result also indicates to what extent the performance of the algorithm would degrade with an increase in the size of the integer constraint. Asymptotically, as M becomes infinitely large, the probability of the decision error for the entire neural network will tend to $(\alpha + \beta)^{\rho}$.

Having derived the relation between the probability of decision error for the atomic decision neural network (ADNN) and that for the entire neural network, we can further characterize the generalization performance of the introduced neural network by utilizing the following well-known result from PAC-Bayes learning theory. For any neural network, [9] gives the following generalization bound,

Lemma 2 *For any training set $\mathcal{K}$ having a sample size K, a corresponding neural network $\mathcal{N}$ trained on $\mathcal{K}$ satisfies the following generalization bound, with probability at least $(1 - \delta)$ for $\delta \in (0, 1)$,*

$$\mathcal{L}\left(\mathcal{N}\right) \leq \mathcal{L}_K\left(\mathcal{N}\right) + 2\mathcal{R}_K\left(\mathcal{H}\right) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n}}. \tag{5}$$

We define each term in Equation (5) as follows. For a given loss function ℓ, the *risk* of $\mathcal{N}$, $\mathcal{L}(\mathcal{N})$, is defined as:

$$\mathcal{L}(\mathcal{N}) := \mathbb{E}(\ell(\boldsymbol{x}, \mathcal{N}(\boldsymbol{y}))), \tag{6}$$

Here the expectation is taken over the joint distribution of the input-output space. Since this joint distribution is unknown, we instead calculate the *empirical* risk $\mathcal{L}_K(\mathcal{N})$ based on the training dataset as:

$$\mathcal{L}_K(\mathcal{N}) := \frac{1}{K}\sum_{k} \ell(\boldsymbol{x}_k, \mathcal{N}(\boldsymbol{y}_k)). \tag{7}$$

The Rademacher complexity $\mathcal{R}_K(\mathcal{H})$ is a measure of the richness of the class of neural networks $\mathcal{H}$ such that $\mathcal{N} \in \mathcal{H}$, and is defined as [9]:

$$\mathcal{R}_K(\mathcal{H}) := \mathbb{E}_{\boldsymbol{\epsilon}^K, \mathcal{K}}\left[\frac{1}{K}\sup_{\mathcal{N} \in \mathcal{H}}\left|\sum_{k=1}^{K} \epsilon_k \ell\left(\boldsymbol{x}_k, \mathcal{N}\left(\boldsymbol{y}_k\right)\right)\right|\right], \tag{8}$$

where $\boldsymbol{\epsilon}^K := [\epsilon_1, \epsilon_2, \cdots, \epsilon_k, \cdots, \epsilon_K]$ is a vector of i.i.d. Rademacher random variables, such that each $\epsilon_k \in \{-1, +1\}$ with probabilities $\{-\frac{1}{2}, +\frac{1}{2}\}$, respectively..

Lemma 3 *Let $\mathcal{H}$ represent the class of all feed-forward neural network-based classifiers with weights $\boldsymbol{w}$ satisfying the norm constraint $\|\boldsymbol{w}\| \leq B$, i.e.,:*

$$\mathcal{H} := \{\langle \boldsymbol{w}, \cdot \rangle : \|\boldsymbol{w}\| \leq B\}.$$

Then, the Rademacher complexity for $\mathcal{H}$ is bounded by [9] as:

$$\mathcal{R}_K\left(\mathcal{H}\right) \leq \prod_{j=1}^{J}\left(L_j B_j\right) \cdot \left(\frac{B_0 R}{\sqrt{K}} + \frac{2R}{K}\sqrt{J \log 2}\right), \tag{9}$$

where J is the number of layers except the input layer, B_j is the radius of the weights at the j^{th} layer; the intermediate activation functions are assumed to be Lipschitz-continuous with Lipschitz constant L_j and R is the radius of the input training data, i.e. $R := \sqrt{\frac{1}{K}\sum_{k=1}^{K}\|\boldsymbol{y}_k\|^2}$.

Note that the loss function $\ell(\cdot)$ used in the definition of $\mathcal{L}_K(\mathcal{N})$ in Lemma 2 is characterized in terms of the Type-I and Type-II errors α and β respectively defined in Lemma 1. Combining this relation and the known upper bound on $\mathcal{R}_K(\mathcal{H})$ from Lemma 3, we can arrive at Theorem 1.

We provide a sketch of the proof for Theorem 1 as follows: Using cross-entropy as the loss function $\ell(\cdot)$, we begin by substituting $\mathcal{L}_K(\mathcal{N}_\mathcal{A}) = \frac{1}{K}\sum_k \mathbb{1}(x_k = +1)\log(1-\alpha_k)+\mathbb{1}(x_k = -1)\log(1-\beta_k)$ into (5), where $\mathbb{1}$ represents the indicator function, $1-\alpha_k$ and $1-\beta_k$ is the ADNN outputs for $x_k = +1$ and $x_k = -1$ respectively. Recalling from Lemma 1 that the decision errors encountered during the testing stage are defined as α and β, we have

$$\frac{1}{K}\sum_k \mathbb{1}(x_k = +1)\log(1-\alpha_k) + \mathbb{1}(x_k = -1)\log(1-\beta_k)$$
$$\leq \frac{1}{2}\log(1-\alpha) + \frac{1}{2}\log(1-\beta).$$

Since $\alpha \leq 1$ and $\beta \leq 1$, we can arrive at the inequality in Theorem 1 by using the bound: $\alpha \leq -\log(1-\alpha)$ and $\beta \leq -\log(1-\beta)$.

Supplementary Results

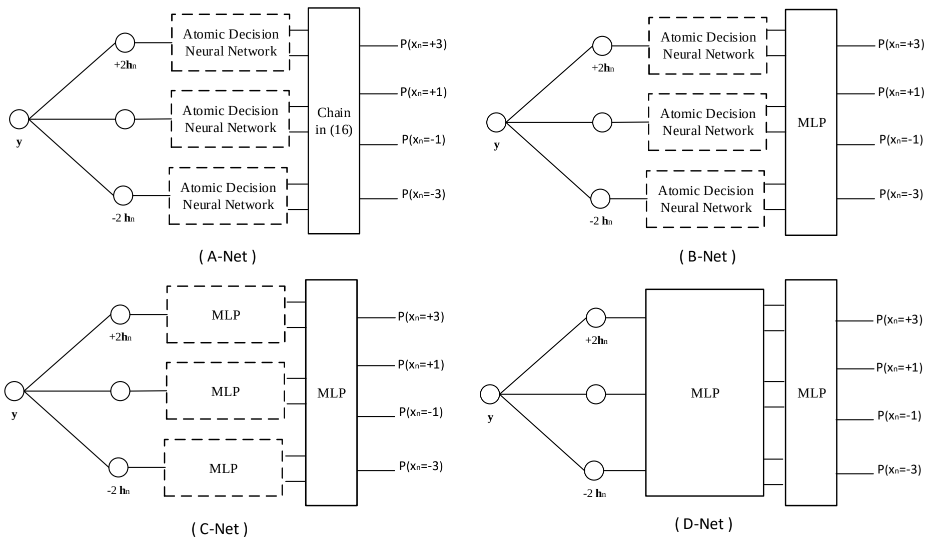

Figure 2: The benchmark neural network structures for symbol detection performance evaluation

Table 2: Performance comparison with non-linear optimization solvers.

Methods	Symbol Detection Success Rate	
	$\|\cdot\|_2$	$\sum_n \log(1+(\cdot)_n^2/\nu)$
Sbplx [10]	0.87	0.85
Nelder-Mead [11]	0.89	0.84
PRAXIS [12]	0.76	0.74
COBYLA [13]	0.92	0.92
DIRECT [14]	0.85	0.82
DIRECT-L [15]	0.83	0.78
Genetic Algorithm [5, 6]	0.71	0.74
DetNet [7]	0.36	0.32
A-Net	1.00	0.90