biomimetics

MDPI

*Article*

# Performance Guarantees of Recurrent Neural Networks for the Subset Sum Problem

**Zengkai Wang [1], Weizhi Liao [1,*], Youzhen Jin [1] and Zijia Wang [2,*]**

[1] College of Artificial Intelligence, Jiaxing University, Jiaxing 314001, China; wangzengkai@zjxu.edu.cn (Z.W.); jinritian521@sina.com (Y.J.)
[2] School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China
* Correspondence: liaowz@zjxu.edu.cn (W.L.); zijiawang@gzhu.edu.cn (Z.W.)

**Abstract:** The subset sum problem is a classical NP-hard problem. Various methods have been developed to address this issue, including backtracking techniques, dynamic programming approaches, branch-and-bound strategies, and Monte Carlo methods. In recent years, researchers have proposed several neural network-based methods for solving combinatorial optimization problems, which have shown commendable performance. However, there has been limited research on the performance guarantees of recurrent neural networks (RNNs) when applied to the subset sum problem. In this paper, we conduct a novel investigation into the performance guarantees of RNNs to solve the subset sum problem for the first time. A construction method for RNNs is developed to compute both exact and approximate solutions of subset sum problems, and the mathematical model of each hidden layer in RNNs is rigorously defined. Furthermore, the correctness of the proposed RNNs is strictly proven through mathematical reasoning, and their performance is thoroughly analyzed. In particular, we prove $w^{NN} \geq w^{OPT}(1-\varepsilon)$ mathematically, i.e., the errors between the approximate solutions obtained by the proposed ASS-NN model and the actual optimal solutions are relatively small and highly consistent with theoretical expectations. Finally, the validity of RNNs is verified through a series of examples, where the actual error value of the approximate solution aligns closely with the theoretical error value. Additionally, our research reveals that recurrence relations in dynamic programming can effectively simulate the process of constructing solutions.

**Keywords:** subset sum problem; performance guarantee; deep learning; recurrent neural networks; dynamic programming



Academic Editors: Heming Jia and Xuewen Xia

Received: 8 March 2025
Revised: 1 April 2025
Accepted: 2 April 2025
Published: 8 April 2025

**Citation:** Wang, Z.; Liao, W.; Jin, Y.; Wang, Z. Performance Guarantees of Recurrent Neural Networks for the Subset Sum Problem. *Biomimetics* **2025**, *10*, 231. https://doi.org/10.3390/biomimetics10040231



## 1. Introduction

The subset sum problem (SSP) is a well-established issue in combinatorial optimization problems (COPs), which has found extensive applications in various engineering domains, including capital budgeting, workload allocation, and job scheduling. While the solution to the verification aspect of this problem is relatively straightforward, identifying a subset of numbers that satisfies a specified target remains a challenging endeavor.

### *1.1. Heuristic Algorithm for SSP*

Over the past few decades, numerous exact and heuristic algorithms have been developed to address the SSP [1,2]. M. F. M. K. Madugula et al. proposed a well-enabled meta-heuristic algorithm named the arithmetic optimization algorithm to solve the SSP [3], while Wan et al. proposed an efficient GPU-based parallel double-list algorithm to solve the SSP, achieving higher performance gains on GPUs than CPUs by improving the design

of generation, pruning, and search phases and optimizing GPU memory management and task allocation [4,5]. P. Dutta et al. presented search algorithms for variants of the SSP [6], and Ye et al. introduced a priority algorithm approximation ratios for the SSP with a focus on the power of revocable decisions, where accepted data items can be later rejected to maintain solution feasibility [7]. V. Parque developed a new scheme to sample solutions for the SSP based on swarm-based optimization algorithms with distinct forms of selection pressure, the balance of exploration-exploitation, the multi-modality considerations, and a search space defined by numbers associated with subsets of fixed size [8]. L. Li et al. devised a DNA procedure for solving the SSP in the Adleman–Lipton model, which operates in $O(n)$ steps for an undirected graph with n vertices [9]. J. R. M. Kolpakov et al. presented a readily implementable recursive parallelization strategy for solving the SSP using the branch-and-bound method [10]. R. Kolpakov et al. studied the question of parallelization of a variant of the branch-and-bound method for solving the SSP [11]. Thada et al. proposed a genetic algorithm approach with infeasible offspring rejection and a penalty function to find approximate solutions for the SSP, demonstrating improved efficiency by discarding unfit subsets during the optimization process [12]. Li et al. presented a DNA-based algorithm in the Adleman–Lipton model that solves the SSP in $O(n)$ time by strategically designing vertex strand lengths to simplify computation and efficiently identify valid subsets [9]. Wang et al. proposed an enhanced genetic algorithm for the SSP that replaces probabilistic operations with conditional crossover and mutation, demonstrating improved capability to find optimal solutions [13]. Bhasin et al. proposed a genetic algorithm-based approach to solve the SSP and explored its potential as a generic methodology for tackling NP-complete problems, demonstrating promising results through implementation and analysis [14]. Saketh et al. evaluated the genetic algorithm for solving the SSP and compared it with dynamic programming, concluding that the genetic algorithm is less favorable due to its longer execution time despite its adaptability [15]. Genetic algorithms exhibit strong adaptability when addressing large-scale problems. However, their performance is sensitive to parameter selection, and they are susceptible to becoming trapped in local optima.

### *1.2. Dynamic Programming for SSP*

Dynamic programming is an effective method for solving COPs [16]. The basic idea of dynamic programming is to decompose the problem into several sub-problems with similar structures and then derive the solution of the original problem from the solutions of these sub-problems. Yang et al. proposed a neural network-based dynamic programming method for NP-hard problems. Since this method requires training on each testing instance of the problem, its time complexity remains high for practical tasks [17]. Allock et al. put forward a novel dynamic programming data structure with applications to subset-sum and its variants including equal-sums, two-subset-sum, and shifted-sums [18], while H. Fujiwara et al. formalized the recurrence relation of the dynamic programming for the SSP [19]. Xu et al. developed a general framework called neural network approximated dynamic programming, which replaces policy or value function calculation processes with neural networks [20]. Both Yang et al.'s and Xu et al.'s studies utilized neural networks to expedite dynamic programming algorithms for COPs but not to search solution space. This proposed method can be generalized to other COPs that can be solved by dynamic programming. However, different problems have different dynamic programming equations, and different COPs have different approximate solution methods. Therefore, correctly establishing corresponding neural networks for specified COPs is crucial in this approach. The dynamic programming method constructs solutions based on recursive relationships, enabling the effective acquisition of optimal solutions. However, even when problems are broken down into sub-problems to reduce search space, the complexity of algorithms using

dynamic programming can still be high for NP-hard problems [20]. However, its time and space complexity increases significantly with the growth of the problem size, making it more suitable for solving small-scale problems.

### *1.3. Others*

Biesner et al. tackled the SSP as a quadratic unconstrained binary optimization problem and showed how gradient descent on Hopfield networks reliably determines solutions for both artificial and real data [21]. Chenyang Xu and Guochuan Zhang introduced an enhanced learning algorithm to address the online SSP. This approach is designed for rapid solution generation, which does not guarantee the accuracy of the solutions obtained [22]. Coron J. et al. described a proven polynomial-time algorithm for solving the hidden SSP based on statistical learning [23]. M. Costandin provided a geometric interpretation of a specific class of SSP [24]. Zheng Q. L. et al. proposed a method for solving the SSP utilizing a quantum algorithm. However, this approach is only effective for collections containing a limited number of elements [25], and in [26], they translated SSP into the quantum Ising model and solved it with a variational quantum optimization method based on conditional values at risk. This method provides a new perspective for solving the SSP. Moon explored the potential of quantum computing, particularly Grover's algorithm, to solve the NP-complete SSP more efficiently than classical methods like dynamic programming while also discussing the broader implications for NP-complete problems [27]. Bernstein et al. introduced a new algorithm that combines the Howgrave-Graham–Joux subset sum algorithm with a new streamlined data structure for quantum walks on Johnson graphs [28]. Quantum algorithms leverage superposition states to simultaneously explore multiple candidate solutions, allowing for efficient searches of optimal solutions within the solution space. Nevertheless, this approach is prone to noise interference, has high implementation complexity, and requires specialized quantum equipment.

With the advancement in deep learning and neural networks, significant progress has been made in the application of deep learning to fields such as image recognition, natural language processing, and autonomous driving. As a result, there is growing interest in exploring the potential of deep learning in other domains. Research indicates that deep learning and neural networks show promise in addressing COPs [29]. Notable efforts to apply deep learning methodologies to COPs are summarized in [30]. Recurrent neural networks (RNNs) are powerful sequence models. Hopfiled et al. were among the first to utilize RNNs to solve COPs [31]. M. S.Tarkov proposed an algorithm for solving the traveling salesman problem based on the use of the Hopfield recurrent neural network, the "Winner takes all" method for the cycle formation, and the two-opt optimization method [32]. Gu et al. devised a novel algorithm for the L smallest k-subsets sum problem by integrating the L shortest paths algorithm with the finite-time convergent recurrent neural network [33]. Gu and Hao applied recurrent neural networks driven by pure data to solve the 0–1 knapsack problems [34]. Zhao et al. proposed a two-phase neural combinatorial optimization method with reinforcement learning for the agile Earth-observing satellite scheduling problem [35]. M. T. Kechadi et al. developed an efficient neural network approach to minimize the cycle time for job shop scheduling problems [36]. C. Hertrich et al. conducted a study on the expressive power of neural networks through knapsack problems. They iteratively applied a class of RNNs to each item in a knapsack instance and computed optimal or provably good solution values [37].

Neural networks demonstrate significant advantages in solving COPs, such as robust pattern-learning capabilities and the ability to uncover hidden rules. By employing end-to-end mapping, neural networks bypass the explicit traversal of all possible composition spaces, thereby avoiding the issue of combinatorial explosion. These characteristics are

absent in traditional heuristic algorithms and dynamic programming methods. Nonetheless, neural networks also possess certain limitations, including high dependence on data, substantial costs associated with generating high-quality training samples, and excessive computational resources required for exact solutions in some cases. Additionally, the accuracy of approximate solutions remains an area requiring improvement.

### *1.4. Our Contributions*

In this paper, we present a rigorous mathematical study on the construction of RNNs to solve two types of SSPs. The main contributions of this paper are as follows:

1. We introduce a recurrent neural network, denoted as SS-NN, to solve the classical SSP. By defining a new activation function, we develop a dynamic programming equation for the classical SSP. This activation function maps all negative numbers to $-1$ and ensures that the number of inputs to the neural network is fixed. NN-SS is utilized to mimic the dynamic programming approach for solving the SSP. We define the mathematical model for each hidden layer of NN-SS and prove its correctness.
2. We propose an approximate solution method for a type of SSP, where the value of the subset-sum is closest to a given value but not exceeding it. The dynamic programming equations for this type of SSP are defined using rounding granularity and the ReLU activation function. We construct a recurrent neural network, denoted as ASS-NN, to mimic the presented dynamic programming approach in determining an approximate solution. We rigorously prove that our proposed ASS-NN can correctly solve the SSP and analyze both its time complexity and error in approximation.
3. We verify the correctness of our proposed method through examples and demonstrate that actual error values in approximate solutions align with theoretical error values through a series of illustrative examples.

The remainder of this paper is structured as follows. Section 2 describes the construction of a recurrent neural network to solve the classical SSP. Section 3 presents an approximate solution method for another type of SSP by constructing a novel recurrent neural network. Section 4 provides the experimental results and performance error analysis. Section 5 discusses the work of this paper in depth. Finally, we conclude this paper and discuss future work in Section 6.

## 2. RNNs for an Exact Solution to the SSP

In this section, we aim to determine whether a given set of positive numbers $S$ contains a subset whose sum equals a specified positive number $W$. This problem can be formulated as a YES/NO decision problem, denoted as the Y/N SSP in the following discussion. It is worth noting that this problem falls within the class of NP-complete problems [38]. Let $S = \{w_1, \ldots, w_n\}$ be a subset of $\mathbf{N}^+$ and $W \in \mathbf{N}^+$ be a fixed number. Let $S^* = \{w_{j_1}, \ldots, w_{j_m}\}$ be a subset of $S$, and $V^* = \sum_{k=j_1}^{j_m} w_k$. The mathematical model of the Y/N SSP is described by Equation (1).

$$y = \begin{cases} 1 & \exists S^* \subseteq S \quad satisfies \quad V^* = W \\ 0 & \quad\quad\quad\quad otherwise \end{cases}. \tag{1}$$

We propose a dynamic programming formulation for the Y/N SSP. Let $s(w,i) \in \{0,1\}$. If the sum of the first $i$ elements of set $S$ equals $w$, then $s(w,i) = 1$; otherwise, $s(w,i) = 0$. Notably, $s(w,i) = 1$ when $w = 0$. The values of $s(w,i)$ can be computed using Equation (2).

$$s(w,i) = \begin{cases} s(w,i-1) & w < w_i \\ s(w-w_i,i-1) \ \ or \ s(w,i-1) & w \geq w_i \end{cases}, \tag{2}$$

where $i \in [n]$, and $[n] = \{1,2,\ldots,n\}$. If $s(w,i) = 1$, the corresponding subset can be obtained by backtracking.

Let $s(w-w_i,i-1) = 0$ for $w - w_i < 0$. It is evident that Equation (2) can be substituted with Equation (3) as follows:

$$s(w,i) = s(w-w_i,i-1) \ \ or \ s(w,i-1). \tag{3}$$

Notably, $s(w,i) = 1$ for $w = 0$.

Inspired by the work of [37], who employed a recurrent neural network to address the knapsack problem, we present our own recurrent neural network, referred to as SS-NN, aimed at solving the Y/N SSP in this section. It is evident that if $w - w_i < 0$, then $s(w-w_i,i-1) = 0$. However, it is impractical to input all possible values of $w - w_i$ into the neural network. Therefore, we represent all negative numbers with $-1$. Consequently, we define a novel activation function $\delta'(x)$, as shown in Equation (4).

$$\delta'(x) = \begin{cases} x & x \geq 0 \\ -1 & x < 0 \end{cases}. \tag{4}$$

According to the activation function $\delta'(x)$, if $w - w_i \geq 0$, then we have $\delta'(w-w_i) = w - w_i$. Conversely, if $w - w_i < 0$, then it follows that $\delta'(w-w_i) = -1$. The network structure for $\delta'(x)$ is shown in Figure 1.

Therefore, we can utilize Equation (5) to replace Equation (3) with the activation function $\delta'(x)$. The definition of Equation (5) is as follows:

$$s(w,i) = s(\delta'(w-w_i),i-1) \ \ or \ s(w,i-1). \tag{5}$$

Notably, $s(-1,0) = 0$, $s(0,0) = 0$, and $s(w^*,0) = 0$ for $w^* > 0$.

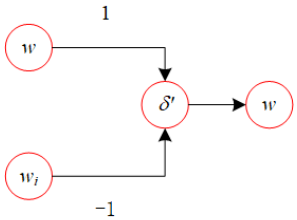


**Figure 1.** Network for the $\delta'$ function.

In order to solve the Y/N SSP by implementing the logical disjunction operation, we have developed a neural network specifically designed for this purpose. The network structure for $x_1$ or $x_2$ is illustrated in Figure 2, with the activation function $h(x)$ as a step function. The precise definition of $h(x)$ can be found in Equation (6).

$$h(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}. \tag{6}$$

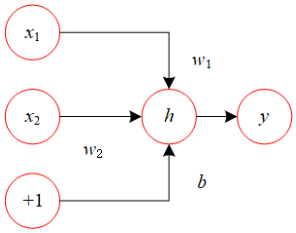


**Figure 2.** Network for or operator.

The recurrent structure based on SS-NN is shown in Figure 3. An overview of the entire structure of SS-NN can be seen in Figure 4.

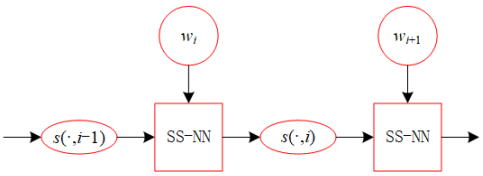


**Figure 3.** Recurrent structure based on SS-NN.

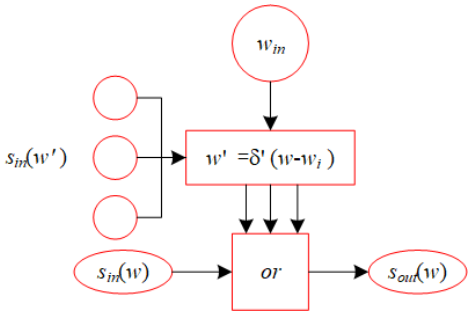


**Figure 4.** The structure of SS-NN.

To clearly demonstrate the recurring structure of SS-NN, we will omit the use of the index $i$ in the following description of the recurrent neural network. Equation (5) will be replaced by Equation (7).

$$s_{out}(w) = s_{in}(\delta'(w - w_{in}))\ or\ s_{in}(w). \tag{7}$$

The inputs of SS-NN consist of $s_{in}(-1), s_{in}(0), s_{in}(1), \ldots, s_{in}(W)$, and $w_{in}$. Therefore, the input layer has $W + 3$ neurons, where the values of $s_{in}(-1) = 0$ and $s_{in}(0) = 1$ are constants.

### *2.1. The First Hidden Layer of SS-NN*

The first hidden layer is utilized for computing $\delta'(w - w_{in})$ and consists of $W + 2$ neurons, denoted as $F_1(w, w_{in})$ for $w \in \{-1, 0\} \cup [W]$, where $w_{in} \in \mathbf{N}^+$, and $[W] = \{1, 2, \ldots, W\}$, $\mathbf{N}^+$ represents the set of positive integer. The neurons are defined as follows:

$$F_1(w, w_{in}) = \delta'(w - w_{in}), w \in \{-1, 0\} \cup [W]. \tag{8}$$

**Theorem 1.** *If* $w \in \{0, 1\} \cup [W]$, *and* $w_{in} \in \mathbf{N}^+$, *then* $F_1(w, w_{in})$ *belongs to the set* $\{-1, 0, 1, \cdots, w - 1\}$.

**Proof.** We observe that $F_1(w, w_{in}) = \delta'(w - w_{in})$. Thus, if $w - w_{in} < 0$, we have $F_1(w, w_{in}) = -1$ according to the definition $\delta'$ function. If $w - w_{in} \geq 0$, then $F_1(w, w_{in}) = w - w_{in}$. Since $w_{in} \in \mathbf{N}^+$, it is evident that the maximum value of

$w - w_{in}$ is equal to $w - 1$. Therefore, we can conclude that the value $F_1(w, w_{in})$ must belong to $-1, 0, 1, \cdots, w - 1$. □

### *2.2. The Second Hidden Layer of SS-NN*

The second hidden layer contains $\mathbf{2}(W + \mathbf{2}) * W + \mathbf{2}$ neurons, denoted as $F_2^{(+)}(w, w')$ and $F_2^{(-)}(w, w')$, where $w'$ and $w$ belong to the set $\{-1, 0\} \cup [W]$.

$$F_2^{(+)}(w, w') = \delta(2(F_1(w, w_{in}) - w')), \tag{9}$$

$$F_2^{(-)}(w, w') = \delta(2(w' - F_1(w, w_{in}))), \tag{10}$$

where $\delta(x)$ represents the ReLu activation function. The objective of the second hidden layer is to ascertain whether the value $w'$ is equal to $F_1(w, w_{in})$ or not.

**Theorem 2.** *$F_2^{(+)}(w, w')$+ $F_2^{(-)}(w, w')$=0 if only if $w' = F_1(w, w_{in})$. Otherwise, we have $F_2^{(+)}(w, w') + F_2^{(-)}(w, w') \geq 2$.*

**Proof.** Clearly, if $w' = F_1(w, w_{in})$, then we have $F_1(w, w_{in}) - w' = 0$ and $w' - F_1(w, w_{in}) = 0$. Consequently, we derive that $F_2^{(+)}(w, w') = 0$ and $F_2^{(-)}(w, w') = 0$. Therefore, it follows that $F_2^{(+)}(w, w')$+ $F_2^{(-)}(w, w') = 0$.

If $F_2^{(+)}(w, w')$+$F_2^{(-)}(w, w') = 0$, and given that $F_2^{(+)}(w, w') \geq 0$ and $F_2^{(-)}(w, w') \geq 0$, we can conclude that $F_2^{(+)}(w, w') = 0$ , $F_2^{(-)}(w, w') = 0$. Therefore, we have established that $w' = F_1(w, w_{in})$.

If $w' \neq F_1(w, w_{in})$, then we have either $w' > F_1(w, w_{in})$ or $w' < F_1(w, w_{in})$. In the case where $w' > F_1(w, w_{in})$, it follows that $w' - F_1(w, w_{in}) \geq 1$ since $w' \in \{1, 0\} \cup [W]$ and $F_1(w, w_{in}) \in \{-1, 0, 1, \cdots, w - 1\}$. Consequently, we determine that $F_2^{(+)}(w, w') = 0$ and $F_2^{(-)}(w, w') \geq 2$. Thus, it holds that $F_2^{(+)}(w, w')$+ $F_2^{(-)}(w, w') \geq 2$. Conversely, if $w' < F_1(w, w_{in})$, we also conclude that $F_2^{(+)}(w')$+$F_2^{(-)}(w') \geq 2$.

Let $F_2^{(+)}(w')$+ $F_2^{(-)}(w') \geq 2$. We assume that $w' = F_1(w, w_{in})$. Consequently, we have $F_2^{(+)}(w, w')$+$F_2^{(-)}(w, w') = 0$, which contradicts the inequality $F_2^{(+)}(w, w')$+ $F_2^{(-)}(w, w') \geq 2$. Thus, our initial hypothesis is invalid and we conclude that $w' \neq F_1(w, w_{in})$. □

### *2.3. The Third Hidden Layer of SS-NN*

The third hidden layer contains $(W + \mathbf{2}) * W + \mathbf{2}$ neurons, denoted as $F_3(w, k)$ for $w$ and $k \in \{-1, 0\} \cup [W]$. These neurons are defined as follows:

$$F_3(w, k) = \delta(s_{in}(k) - F_2^{(+)}(w, k) - F_2^{(-)}(w, k)). \tag{11}$$

**Theorem 3.** *For $w, k \in \{-1, 0\} \cup [W]$ and $w_{in} \in \mathbf{N}^+$, we have $F_3(w, k) = s_{in}(k)$ if only if $k = F_1(w, w_{in})$. Otherwise, it follows that $F_3(w, k) = 0$.*

**Proof.** If $k = F_1(w, w_{in})$, then by Theorem 2, we have $F_2^{(+)}(w, k)$+$F_2^{(-)}(w, k) = 0$. This leads to the conclusion that $s_{in}(k) - F_2^{(+)}(w, k) - F_2^{(-)}(w, k) = s_{in}(k)$. Therefore, if $k = F_1(w, w_{in})$, it follows that $F_3(w, k) = s_{in}(k)$ by $s_{in}(k) \in \{0, 1\}$.

Conversely, if $k \neq F_1(w, w_{in})$, we obtain from Theorem 2 that $F_2^{(+)}(w, k) + F_2^{(-)}(w, k) \geq 2$. Since $s_{in}(k) \in \{0, 1\}$, this implies that $s_{in}(k) - F_2^{(+)}(w, k) - F_2^{(-)}(w, k) < 0$. Thus, we conclude that $F_3(w, k) = 0$. □

*2.4. The Fourth Hidden Layer of SS-NN*

The fourth hidden layer contains $W + \mathbf{2}$ neurons, denoted as $F_4(w)$ for w and k $\in \{-1, 0\} \cup [W]$.

$$F_4(w) = \sigma(\sum_{k=-1}^{W} F_3(w,k)). \tag{12}$$

**Theorem 4.** *For each* $w, k \in \{-1, 0\} \cup [W]$ *and* $w_{in} \in \mathbf{N}^+$, *we have* $s_{in}(\delta'(w - w_{in})) = F_4(w) = \sigma(\sum_{k=-1}^{W} F_3(w,k))$.

**Proof.** According to the definition of $\delta'(x)$, we have

$$s_{in}\big(\delta'(w - w_{in})\big) = \begin{cases} 0 & w - w_{in} < 0 \\ s_{in}(w - w_{in}) & w - w_{in} > 0 \\ 1 & w - w_{in} = 0 \end{cases}. \tag{13}$$

On the other hand, if $k > F_1(w, w_{in})$ or $k < F_1(w, w_{in})$, we have $F_2^{(+)}(w,k) + F_2^{(-)}(w,k) \geq 2$. Since $s_{in}(k) \in \{-1, 0\}$, this implies that $F_3(w,k) = 0$. Consequently, we determine $\sum_{k=-1 \text{ and } k \neq F_1(w,w_{in})}^{W} F_3(w,k) = 0$. Thus, the value $\sum_{k=-1}^{W} F_3(w,k)$ is determined by $k$, where $k = F_1(w, w_{in})$. We can draw the following conclusions.

1. If $k = F_1(w, w_{in}) = 0$, then it follows that $w - w_{in} = 0$ and $F_3(w,k) = \delta(s_{in}(k)) = \delta(s_{in}(0)) = 1$. Hence, we obtain $F_4(w) = \sigma\left(\sum_{k=-1}^{W} F_3(w,k)\right) = \delta(1) = 1$.
2. If $k = F_1(w, w_{in}) = -1$, then we conclude that $w - w_{in} < 0$ and $F_3(w,k) = \delta(s_{in}(k)) = \delta(s_{in}(-1)) = 0$. Therefore, we have $F_4(w) = \sigma\left(\sum_{k=-1}^{W} F_3(w,k)\right) = \delta(0) = 0$.
3. If $k = F_1(w, w_{in}) > 0$, this implies that $k = w - w_{in} > 0$ and we have $F_3(w,k) = \delta(s_{in}(k)) = \delta(s_{in}(w - w_{in})) = s_{in}(w - w_{in})$. Thus, we conclude that $F_4(w) = \sigma\left(\sum_{k=-1}^{W} F_3(w,k)\right) = \delta(s_{in}(w - w_{in})) = s_{in}(w - w_{in})$.

In summary, it holds true that $F_4(w) = s_{in}(\delta'(w - w_{in}))$. This indicates that the fourth hidden layer is capable of accurately computing the value of $s_{in}(\delta'(w - w_{in}))$. □

*2.5. The Fifth Hidden Layer of SS-NN*

The fifth hidden layer is used to compute $s_{in}(\delta'(w - w_{in}))$ *or* $s_{in}(w)$ , and it contains $W + \mathbf{2}$ neurons, denoted as $F_5(w)$ for $w \in \{-1, 0\} \cup [W]$. The definitions are as follow:

$$F_5(w) = s_{in}(\delta'(w - w_{in})) \text{ or } s_{in}(w), \tag{14}$$

$$s_{out}(w) = F_5(w). \tag{15}$$

**Theorem 5.** *For each* $w \in \{-1, 0\} \cup ([W]$ *and* $w_{\text{in}} \in \mathbf{N}^+$, *if* $w < w_{in}$, *then we have* $F_5(w) = s_{in}(w)$. *Conversely, if* $w \geq w_{in}$, *it follows that* $F_5(w) = s_{in}(w - w_{in})$ *or* $s_{in}(w)$.

**Proof.** If $w < w_{in}$, i.e.,$w - w_{in} < 0$, then we have $\delta'(w - w_{in}) = -1$. Thus, we obtain $s_{in}(\delta'(w - w_{in})) = s_{in}(-1) = 0$. Consequently, it follows that $F_5(w) = s_{in}(\delta'(w - w_{in}))$ or $s_{in}(w) = 0$ or $s_{in}(w) = s_{in}(w)$.

Conversely, if $w \geq w_{in}$, i.e., $w - w_{in} \geq 0$, then we determine that $\delta'(w - w_{in}) = w - w_{in}$, leading to $s_{in}(\delta'(w - w_{in})) = s_{in}(w - w_{in})$. Therefore, we conclude that $F_5(w) = s_{in}(w - w_{in})$ or $s_{in}(w)$. □

**Theorem 6.** *Let $S = \{w_1, \ldots, w_n\}$, where $w_i \in \mathbf{N}^+ (1 \leq i \leq n)$ , and let $W$ be a positive integer. We can utilize the SS-NN to iteratively compute the value of $s(w, i)$ for $w \in \{-1, 0\} \cup [W]$, $i \in [n]$.*

**Proof.** We observe that $s(-1, 0) = 0$, $s(0, 0) = 1$, and $s(w_1, 0) = 0$ for $w_1 > 0$. In the first iteration, $w_{in} = w_1$. Since $0 < w_{in}$ , it follows that $s(-1, 1) = s(-1, 0) = 0$, and $s(0, 1) = s(0, 0) = 1$ according to Theorem 4. For $w \in [W]$, if $w < w_{in}$, we have $s(w, 1) = s(w, 0)$, otherwise we obtain $s(w, 1) = s(w - w_{in}, 0)$ or $s(w_{in}, 0)$ by Theorem 4. In the second iteration, since $w_{in} = w_2$, we have $s(-1, 2) = s(-1, 1) = 0$, and $s(0, 2) = s(0, 1) = 1$ by Theorem 4. For any $w \in [W]$, if $w < w_{in}$, it follows that $s(w, 2) = s(w, 1)$. Conversely, if this condition does not hold true, we conclude that $s(w, 2) = s(w - w_{in}, 1)$ or $s(w, 1)$ and so on. In the final iteration, where $w_{in} = w_n$, we have $s(-1, n) = s(-1, n - 1) = 0$, and $s(0, n) = s(0, n - 1) = 1$ by Theorem 4. For any $w \in [W]$, if $w < w_{in}$, we have $s(w, n) = s(w, n - 1)$; otherwise, we have $s(w, n) = s(w - w_{in}, n - 1)$ or $s(w, n - 1)$. □

The depth of a single SS-NN is seven, comprising five hidden layers, one input layer, and one output layer. Consequently, the overall depth of the SS-NN designed for solving SSPs is $7n$, where $n$ represents the number of elements in the set. Thus, the order of depth can be expressed as $O(7n)$. The width of the SS-NN is determined by the value of $W$. Given that the hidden layers with maximum width are the second and third layers, we determine that the width order for a single SS-NN is $O(2(W + 2)^2)$. Therefore, when unfolding the SS-NN and treating it as a singular neural network executing an entire dynamic program, we arrive at a width complexity of $O(2n(W + 2)^2)$.

## 3. RNNs for the Approximate Solution to the SSP

It is not hard to know that the exact result of an SSP is dependent on the network width $M$. In order to overcome this drawback, we proposed a neural network called ASS-NN that can determine the approximation solution for variants of SSP. Similar to [37], ASS-NN uses less neurons but loses optimality. Let $S = \{w_1, \ldots, w_n\}$ be a subset of $\mathbf{N}^+$, $P \in \mathbf{N}^+$ be a fixed number, $S^* = \{w_{j_1}, \ldots, w_{j_m}\}$ be a subset of $S$, and $V^* = \sum_{k=j_1}^{j_m} w_k$. In this section, our task is determining subset $S^*$ such that $V^*$ is closest to $P$ but not more than $P$. This kind of SSP is more common in practical applications. Let $w_i^* = \sum_{k=1}^{i} w_k$ be the total value of the first $i$ elements. Once $w_i^*$ exceed $P$, we do not store a required value for every possible sum but have to round sum. The rounding granularity of $w_i^*$ is denoted by $r_i = \text{roundup}(\max\left\{1, \frac{w_i^*}{M}\right\})$, where roundup(x) is the upward rounding function and $M \in \mathbf{N}^+$. Let $p(w, i)$ be the total value of a subset of $S$, and it must be closest to $wr_i$ but not more than $wr_i$, where $w \in [M]$. The values of $p(w, i)$ can be computed recursively by Equation (16).

$$p(w,i) = \begin{cases} \max\left\{p^{(1)}, p^{(2)}\right\}, \exists w^{(2)} \in M \text{ such that } p^{(2)} \leq wr_i \\ p^{(1)}, \text{otherwise} \end{cases}, \tag{16}$$

where $p^{(1)} = p(w^{(1)}, i - 1)$ and $p^{(2)} = p\left(w^{(2)}, i - 1\right) + w_i$. Let $w^{(1)}, w^{(2)} \geq 1, w^{(1)}$ be the largest possible integers satisfying the condition $p(w^{(1)}, i - 1) \leq wr_i$, and $w^{(2)} \in M$ be the largest possible integers that satisfies the condition $p\left(w^{(2)}, i - 1\right) + w_i \leq wr_i$. In particular, if $i \leq 0$, then $p(w, i) = 0$. In order to use a recurrent neural network to determine the solution for Equation (16), we introduce an activation function called ReLUs

to reconstruct dynamic programming for the SSP. The solution can be computed recursively by Equation (17).

$$p(w,i) = \max\{p\left(w^{(1)}, i-1\right), \delta(p\left(w^{(2)}, i-1\right) + w_i)\}, \tag{17}$$

where $p(w,\ i) = -\infty$ for $w \leq 0$ and $p(w,0) = 0$, $w^{(1)}$ is the largest possible integer satisfying the condition $p\left(w^{(1)}, i-1\right) \leq wr_i$, and $w^{(2)}$ is the largest possible integer that satisfies the condition $p\left(w^{(2)}, i-1\right) + w_{in} \leq wr_i$.

**Theorem 7.** *For $w \in [M]$ and $i \geq 1$, the value $p(w,i)$ in Equation (17) is equal to the value $p(w,i)$ in Equation (16).*

**Proof.** Obviously, if $w \in [M]$, then $wr_i > 0$. Since $w^{(1)}$ is the largest possible integer with $p(w^{(1)}, i-1) \leq wr_i$, it is easy to know that $p(w,i) \geq 0$ for $w \in [M]$ and $i \geq 1$ using Equation (17). Furthermore, we observe that $r_{i-1} \leq r_i$. Hence, there exists a value of $w^{(1)} \in [M]$ satisfying the condition $p(0, i-1) < p(w^{(1)}, i-1) \leq w^{(1)} r_{i-1} \leq wr_i$. On the other hand, $w^{(1)}$ is the largest possible integer satisfying the condition $p\left(w^{(1)}, i-1\right) \leq wr_i$ according to the definition of Equation (16). Thus, we conclude that the value of $p(w^{(1)}, i-1)$ in Equation (17) is equal to the value of $p(w^{(1)}, i-1)$ in Equation (16).

For Equation (17), if no $w^{(2)} \in [M]$ is the largest possible integer satisfying the condition $p\left(w^{(2)}, i-1\right) + w_{in} \leq wr_i$, then we have $w^{(2)} = 0$. This leads to $p(0, i-1) + w_{in} \leq wr_i$. Since $p(0, i-1) = -\infty$, it follows that $\delta(p\left(w^{(2)}, i-1\right) + w_i) = 0$. Consequently, we obtain $p(w,i) = \max\{p\left(w^{(1)}, i-1\right), \delta(p\left(w^{(2)}, i-1\right) + w_i)\ \} = p\left(w^{(1)}, i-1\right)$. For Equation (16), if no $w^{(2)} \in M$ is the largest possible integer that satisfies the condition $p\left(w^{(2)}, i-1\right) + w_i \leq wr_i$, we determine that $p(w,i) = p(w^{(1)}, i-1)$. Thus, we conclude that the value of $p(w,i)$ in Equation (17) is equal to the value of $p(w,i)$ in Equation (16).

For Equation (17), let $\exists w^{(2)} \in [M]$ be the largest possible integer that satisfies the condition $p\left(w^{(2)}, i-1\right) + w_{in} \leq wr_i$. In this case, we have $\delta(p\left(w^{(2)}, i-1\right) + w_i) = p\left(w^{(2)}, i-1\right) + w_i$ when $p\left(w^{(2)}, i-1\right) + w_i \geq 0$. Consequently, we obtain $p(w,i) = \max\{p\left(w^{(1)}, i-1\right), \delta(p\left(w^{(2)}, i-1\right) + w_i))\} = \max\{p\left(w^{(1)}\right), p\left(w^{(2)}, i-1\right) + w_i\}$. For Equation (16), if $\exists w^{(2)} \in [M]$ is the largest possible integer satisfying the condition $p\left(w^{(2)}, i-1\right) + w_i \leq wr_i$, it follows that $p(w,i) = \max\{p\left(w^{(1)}, i-1\right), \delta(p\left(w^{(2)}, i-1\right) + w_i)\ \}$. Thus, we also have the value of $p(w,i)$ in Equation (17), which is equal to the value of $p(w,i)$ in Equation (16). □

Furthermore, in order to obviously show the recurrent structure of ASS-NN, we do not use the index $i$ in the following:

$$p_{out}(w) = \max\{p_{in}\left(w^{(1)}\right), \delta(p_{in}\left(w^{(2)}\right) + w_{in})\}, \quad w \in M, \tag{18}$$

where $w_{in}$ represents the value of element $i$, $p_{in}\left(w^{(1)}\right)$ denotes the option of not using element $i$, and $\delta(p_{in}\left(w^{(2)}\right) + w_{in})$ represents the option of using element $i$. To compute $w^{(1)}$ and $w^{(2)}$, both $w_{in}^*$ and $w_{in}$ are utilized to determine rounding granularities in ASS-NN.

In this section, we propose a recurrent neural network referred to as ASS-NN, designed to approximate the solution for Equation (18). The recurrent structure of the proposed neural network is shown in Figure 5. Figure 6 presents the comprehensive structure of ASS-NN, where the first hidden layer is responsible for calculating the previous and current

rounding granularities, denoted as $r_{old}$ and $r_{new}$. The second hidden layer's function is to select integers $w^{(1)}$ and $w^{(2)}$. The third hidden layer computes values $p_{in}\left(w^{(1)}\right)$ and $p_{in}\left(w^{(2)}\right)$, while the fourth hidden layer determines $\max\{p_{in}\left(w^{(1)}\right), \delta(p_{in}\left(w^{(2)}\right) + w_{in}))\}$.

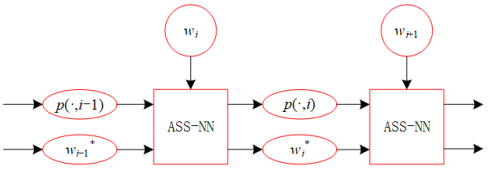


**Figure 5.** Recurrent structure based on ASS-NN.

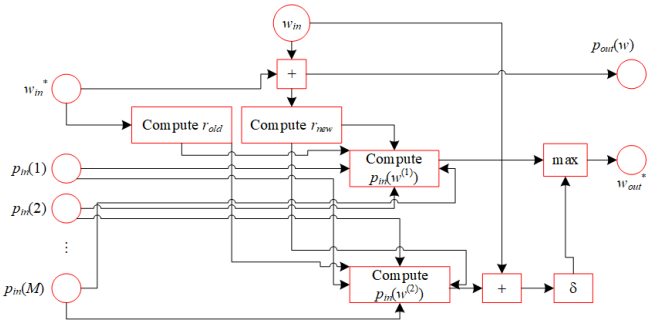


**Figure 6.** Computing $p_{out}(w)$ and $w_{out}^*$.

The ASS-NN is detailed in this section, demonstrating its capability to approximate solutions for the SSP. The proposed neural network comprises four hidden layers.

### *3.1. The First Layer of ASS-NN*

The first layer is responsible for calculating the rounding granularities $r_{old}$ and $r_{new}$, which are defined as follows:

$$\begin{array}{l} F_1^{(1)} = \delta(\frac{w_{in}^*}{M} - 1),\ F_1^{(2)} = \delta(\frac{w_{in}^* + w_{in}}{M} - 1) \\ r_{old} = F_1^{(1)} + 1,\ r_{new} = F_1^{(2)} + 1 \end{array} . \tag{19}$$

The correctness of the computing method was proven in [37].

### *3.2. The Second Layer of ASS-NN*

The second layer consists of $2(M^2 + M)$ neurons, denoted as $F_2^{(1+)}(w,k)$ and $F_2^{(1-)}(w,k)$ for $w \in [M], k \in \{0\} \cup [M]$ with the condition that $w \leq k$. Additionally, we have $F_2^{(2+)}(w,k)$ and $F_2^{(2-)}(w,k)$ for $w \in [M]$ ,$k \in \{0\} \cup [M]$ under the constraint that $w \geq k$.

$$F_2^{(1+)}(w,k) = \delta(Mr_{max}(p_{in}(k) - wr_{new})), \tag{20}$$

$$F_2^{(1-)}(w,k) = \delta(Mr_{max}(wr_{new} - p_{in}(k+1)) + r_{max}) , \tag{21}$$

$$F_2^{(2+)}(w,k) = \delta(Mr_{max}(p_{in}(k) + w_{in} - wr_{new})) , \tag{22}$$

$$F_2^{(2-)}(w,k) = \delta(Mr_{max}(wr_{new} - p_{in}(k+1) - w_{in}) + r_{max}) . \tag{23}$$

For a fixed $w \in [M]$, let $w^{(1)}$ and $w^{(2)}$ denote the largest possible integers such that $p_{in}\left(w^{(1)}\right) \leq wr_{new}$ and $p_{in}\left(w^{(2)}\right) + w_{in} \leq wr_{new}$. The purpose of the second layer is to identify the integers $w^{(1)}$ and $w^{(2)}$. The validity of this hidden layer is guaranteed by the following two theorems.

**Theorem 8.** *For every $w,k \in [M]$ with $w \le k$, we have $F_2^{(1+)}(w,k) + F_2^{(1-)}(w,k) = 0$, if and only if $k = w^{(1)}$. Otherwise, it follows that $F_2^{(1+)}(w,k) + F_2^{(1-)}(w,k) \ge r_{max}$.*

**Proof.** Clearly, it follows that $F_2^{(1+)}(w,k) = 0$, if and only if $wr_{new} \ge p_{in}(k)$. Simultaneously, since $p_{in}(k+1)$ and $r_{new}$ are integer multiples of $\frac{1}{M}$, we have $F_2^{(1-)}(w,k) = 0 \Leftrightarrow wr_{new} + \frac{1}{M} \le p_{in}(k+1) \Leftrightarrow wr_{new} < p_{in}(k+1) \Leftrightarrow$, and no integer $k^* > k$ satisfies $p_{in}(k^*+1) \le wr_{new}$.

If $F_2^{(1+)}(w,k) + F_2^{(1-)}(w,k) \ne 0$, then it follows that either $F_2^{(1+)}(w,k) \ne 0$ or $F_2^{(1-)}(w,k) \ne 0$. Assuming $F_2^{(1+)}(w,k) \ne 0$, we derive the inequality $p_{in}(k) - wr_{new} > 0$. Given that both $p_{in}(k)$ and $r_{new}$ are integer multiples of $\frac{1}{M}$, it can be concluded that $p_{in}(k) - wr_{new} \ge \frac{1}{M}$. Consequently, this leads to the result $Mr_{max}(p_{in}(k) - wr_{new}) \ge r_{max}$. Thus, we obtain $F_2^{(1+)}(w,k) + F_2^{(1-)}(w,k) \ge r_{max}$. If $F_2^{(1-)}(w,k) \ne 0$, we have $wr_{new} \ge p_{in}(k+1)$. This implies that $Mr_{max}(wr_{new} - p_{in}(k+1)) \ge 0$. Hence, we conclude that $\delta(Mr_{max}(wr_{new} - p_{in}(k+1)) + r_{max}) \ge r_{max}$. Therefore, the claim is proven. □

**Theorem 9.** *For each $w,k \in [M]$ with $w \ge k$, we have $F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k) = 0$, if and only if $k = w^{(2)}$. Otherwise, it follows that $F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k) \ge r_{max}$.*

**Proof.** It is clear that $F_2^{(2+)}(w,k) = 0$, if and only if $wr_{new} \ge p_{in}(k) + w_{in}$. Since $p_{in}(k+1)$ and $r_{new}$ are integer multiples of $\frac{1}{M}$, it follows that $F_2^{(2-)}(w,k) = 0 \Leftrightarrow wr_{new} + \frac{1}{M} p_{in}(k+1) + w_{in} \Leftrightarrow wr_{new} < p_{in}(k+1) + w_{in} \Leftrightarrow$, and no integer $k^* > k$ satisfies $p_{in}(k^*+1) + w_{in} \le wr_{new}$

If $F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k) \ne 0$, it holds that either $F_2^{(2+)}(w,k) \ne 0$ or $F_2^{(2-)}(w,k) \ne 0$. Assuming $F_2^{(2+)}(w,k) \ne 0$, we have the inequality $wr_{new} < p_{in}(k) + w_{in}$. Since $r_{old}$ , $r_{new}$ and $w_{in}$ are integer multiples of $\frac{1}{M}$, and we have $p_{in}(k) + w_{in} - wr_{new} \ge \frac{1}{M}$. Thus, we obtain $Mr_{max}(p_{in}(k) + w_{in} - wr_{new}) \ge r_{max}$ and $F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k) \ge r_{max}$. If $F_2^{(2-)}(w,k) \ne 0$, we have $wr_{new} \ge p_{in}(k+1) + w_{in}$. This implies that $Mr_{max}(wr_{new} - p_{in}(k+1) - w_{in}) + r_{max} \ge r_{max}$. Hence, we conclude that $F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k) \ge r_{max}$. □

### *3.3. The Third Hidden Layer of ASS-NN*

In the third hidden layer, there are $M^2 + M$ hidden neurons denoted as $F_3^{(1)}(w,k)$ for $w,k \in [M]$ with $w \le k$, and $F_3^{(2)}(w,k)$ for $w,k \in [M]$ with $w > k$. These neurons are defined as follows:

$$\begin{aligned} F_3^{(1)}(w,k) &= \delta\Big(Mr_{max} - p_{in}(k) - M(F_2^{(1+)}(w,k) + F_2^{(1-)}(w,k))\Big) \\ h^{(1)}(w) &= Mr_{max} - \sum_{k=w}^{M} F_3^{(1)}(w,k),\ w \in [M] \end{aligned}, \tag{24}$$

$$\begin{aligned} F_3^{(2)}(w,k) &= \delta\Big(k - M(F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k))\Big) \\ h^{(2)}(w) &= p_{in}(\sum_{k=1}^{w} F_3^{(2)}(w,k)),\ w \in [M] \end{aligned}. \tag{25}$$

The function of the third hidden layer is to calculate the values $p_{in}\Big(w^{(1)}\Big)$ and $p_{in}\Big(w^{(2)}\Big)$. The accuracy of this layer is guaranteed by the following two theorems.

**Theorem 10.** *For each $w \in [M]$, if $w \le w^{(1)} \le M$, we have $h^{(1)}(w) = p_{in}\Big(w^{(1)}\Big)$. If $w^{(1)} > M$, it follows that $h^{(1)}(w) = Mr_{max}$.*

**Proof.** If $w \leq w^{(1)} \leq M$, then by Theorem 8, we have $F_3^{(1)}\left(w, w^{(1)}\right) = Mr_{max} - p_{in}\left(w^{(1)}\right)$ and $F_3^{(1)}(w,k) = 0$ for each $k \neq w^{(1)}$. Consequently, it follows that $h^{(1)}(w) = p_{in}\left(w^{(1)}\right)$. In the case where $w(1) > M$, it holds that $F_2^{(1+)}(w,k) + F_2^{(1-)}(w,k) \geq r_{max}$ for each $k$. This implies that $F_3^{(1)}(w,k) = 0$. Thus, we conclude that $h^{(1)}(w) = Mr_{max}$. □

**Theorem 11.** *For each $w \in [M]$, if $w^{(2)} \geq 1$, then $\sum_{k=1}^{w} F_3^{(2)}(w,k) = w^{(2)}$. If $w^{(2)} \leq 0$, then $\sum_{k=1}^{w} F_3^{(2)}(w,k) = 0$.*

**Proof.** If $w^{(2)} \geq 1$, then $F_3^{(2)}\left(w, w^{(2)}\right) = w^{(2)}$ and $F_3^{(2)}(w,k) = 0$ for each $k \neq w^{(2)}$. Thus, we have $\sum_{k=1}^{w} F_3^{(2)}(w,k) = w^{(2)}$. If $w^{(2)} \leq 0$, then by Theorem 9, it follows that $F_2^{(2+)}(w,k) + F_2^{(2-)}(w,k) \geq r_{\max}$ for each $k$. This implies $F_3^{(2)}(w,k) = 0$. Thus, we obtain $\sum_{k=1}^{w} F_3^{(2)}(w,k) = 0$. □

*3.4. The Final Hidden Layer of ASS-NN*

Obviously, the value $p_{in}\left(w^{(2)}\right)$ is equal to $h^{(2)}(w)$ according to Theorem 11. The function of the final hidden layer is to calculate the value of $max\{h^{(1)}(w), \delta(w_{in} + h^{(2)}(w))\}$ and comprises $M$ neurons, denoted as $F_4(w)$ for $w \in [M]$. Finally, we output the value of $p_{out}(w)$ for $w \in [M]$.

$$\begin{array}{l} F_4(w) = \delta\left(h^{(1)}(w) - \delta(w_{in} + h^{(2)}(w))\right),\ w \in [M] \\ p_{out}(w) = F_4(w) + \delta(w_{in} + h^{(2)}(w)),\ w \in [M] \end{array} . \tag{26}$$

**Theorem 12.** *For each $w \in [M]$, the value $p_{out}(w)$ is equal to the value $max\{h^{(1)}(w), \delta(w_{in} + h^{(2)}(w))\}$.*

**Proof.** If $h^{(1)}(w) \geq \delta(w_{in} + h^{(2)}(w))$, then $F_4(w) = h^{(1)}(w) - \delta(w_{in} + h^{(2)}(w))$. Consequently, we have $p_{out}(w) = F_4(w) + \delta(w_{in} + h^{(2)}(w)) = h^{(1)}(w)$. Conversely, if $h^{(1)}(w) < \delta(w_{in} + h^{(2)}(w))$, then $F_4(w) = 0$. This leads to the conclusion that $p_{out}(w) = F_4(w) + \delta(w_{in} + h^{(2)}(w)) = \delta(w_{in} + h^{(2)}(w))$. Thus, the value of $p_{out}(w)$ is equal to the value $max\{h^{(1)}(w), \delta(w_{in} + h^{(2)}(w))\}$. □

Let $S = \{w_1, \ldots, w_n\}$ and $M$ be fixed positive integers. Let $w^{OPT}$ represent the actual optimal solution to the SSP that does not exceed $wr_n$. The corresponding subset is denoted as $W^{OPT}$. Furthermore, define $W_i^{OPT} = W^{OPT} \cap [i]$, and $w_i^{OPT} = \sum_{j \in W_i^{OPT}} w_j$.

**Theorem 13.** *For $i \in [n]$ and $w \leq \left\lceil \frac{w_i^{OPT}}{r_i} \right\rceil - 1$, we have $p(w,i) \leq w_i^{OPT}$.*

**Proof.** If $w \leq \left\lceil \frac{w_i^{OPT}}{r_i} \right\rceil - 1$, we have

$$wr_i \leq \left( \left\lceil \frac{w_i^{OPT}}{r_i} \right\rceil - 1 \right) r_i = \left\lceil \frac{w_i^{OPT}}{r_i} \right\rceil r_i - r_i \leq w_i^{OPT}. \tag{27}$$

Given that $p(w,i) \leq wr_i$, it follows that

$$p(w,i) \leq w_i^{OPT}. \tag{28}$$

□

Let $w^{NN}$ be the approximate solution to ASS-NN and $w^{OPT}$ be the actual solution. We have the following theorem.

**Theorem 14.** *The values of $w^{NN}$ and $w^{OPT}$ satisfy the following inequality: $w^{NN} \geq w^{OPT}(1-\varepsilon)$.*

**Proof.** According to Theorem 13, it follows that $p(\left\lceil \frac{w^{OPT}}{r_n} \right\rceil - 1, n) \leq w_n^{OPT}$, and we have the following inequalities:

$$w^{NN} \geq \left\lceil \frac{w^{OPT}}{r_n} - n - 1 \right\rceil r_n \geq w^{OPT} - (n+1)r_n. \tag{29}$$

Given that $w^{OPT} \geq \frac{w}{n}^*$, $r_n = \frac{w}{M}^*$, we have

$$\frac{w^{NN}}{w^{OPT}} \geq 1 - \frac{n(n+1)}{M} \geq 1 - \varepsilon. \tag{30}$$

Consequently, we have $w^{NN} \geq w^{OPT}(1-\varepsilon)$. □

The depth of a single ASS-NN is six, comprising four hidden layers, one input layer, and one output layer. Consequently, the depth of the ASS-NN designed for solving SSPs is represented as $6n$, where $n$ denotes the number of set elements. Thus, the order of depth can be expressed as $O(6n)$. The width of the ASS-NN is determined by the value of $M$. Given that the hidden layers with maximum width are identified as the second and third layers, we conclude that the width order for a single ASS-NN is $O\left(2(M+2)^2\right)$. Therefore, when unfolding the ASS-NN and treating it as a singular neural network executing an entire dynamic program, we derive a width complexity of $O\left(2n(M+2)^2\right)$.

## 4. Verification and Analysis with Examples

### *4.1. Example for SS-NN*

To demonstrate that the SS-NN can effectively solve the SSP, we consider an illustrative example. Let $S = \{w_1 = 8, w_2 = 34, w_3 = 4, w_4 = 12, w_5 = 5, w_6 = 3\}$ and $W = 7$. The solution processes of SS-NN is described as follows.

Iteration 1: The inputs of SS-NN are $s_{in}(-1,0), s_{in}(0,0), s_{in}(1,0), \ldots, s_{in}(6,0), s_{in}(7,0)$ and $w_1$, respectively. The corresponding values are $0, 1, 0, \ldots, 0$ and 8, respectively. By utilizing the first hidden layer of the SS-NN model, we determine that $F_1(-1,8) = F_1(0,8) = \cdots = F_1(6,8) = F_1(7,8) = -1$. Furthermore, it follows that $F_4(-1) = F_4(0) = F_4(1) = F_4(2) = \cdots = F_4(6) = F_4(7) = 0$. Consequently, the outputs of SS-NN are $s_{out}(-1,1), s_{out}(0,1), s_{out}(1,1), \ldots, s_{out}(6,1)$ and $s_{out}(7,1)$, respectively. The corresponding values for these outputs are as follows: 0,1,0,…,0 and 0, respectively.

Iteration 2: The inputs of SS-NN are $0, 1, 0, \ldots, 0$ and 34. According to the SS-NN model, the outputs are $s_{out}(-1,2), s_{out}(0,2), s_{out}(1,2), \ldots, s_{out}(6,2)$ and $s_{out}(7,2)$, respectively. The corresponding value for these outputs are $0, 1, 0, \ldots, 0$ and 0, respectively.

Iteration 3: The inputs of SS-NN are $0, 1, 0, \ldots, 0$ and 4. According to the SS-NN framework, the corresponding outputs are as follows: $0, 1, 0, 0, 0, 1, 0, 0, 0$.

Iteration 4: The inputs for the SS-NN model are $0, 1, 0, 0, 0, 1, 0, 0, 0$ and 12. Based on the SS-NN framework, the corresponding outputs are $0, 1, 0, 0, 0, 1, 0, 0, 0$.

Iteration 5: The inputs for the SS-NN model are as follows: $0, 1, 0, 0, 0, 1, 0, 0, 0$ and 5. According to the SS-NN model, the outputs are $0, 1, 0, 0, 0, 1, 1, 0, 0$.

Iteration 6: The inputs of SS-NN are $0, 1, 0, 0, 0, 1, 1, 0, 0$ and 3. Based on the SS-NN model, the outputs are $0, 1, 0, 0, 1, 1, 1, 0, 1$.

The values $s_{out}(w, i)$ computed by the SS-NN are shown in Table 1. It is evident that the results obtained the SS-NN alignment with the actual outcomes. Notably, there exist subsets whose elements are equal to 3, 4, 5, and 7, respectively.

**Table 1.** The value of $s_{out}(w, i)$.

| $i$ \ $w$ | −1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **5** | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| **6** | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

### *4.2. Example for ASS-NN*

To demonstrate the capability of the ASS-NN in effectively solving the SSP, we consider an illustrative example. Let $S = \{w_1 = 7, w_2 = 34, w_3 = 4, w_4 = 12, w_5 = 5, w_6 = 3\}$, $M = 6$. The solution processes of the ASS-NN is described as follows:

Iteration 1: The inputs of ASS-NN are $p_{in}(0), p_{in}(1), \ldots, p_{in}(6)$ and $w_{in}$, respectively. The corresponding values are $-\infty, 0, 0, \ldots, 0$ and 7, respectively. We have $r_{old} = 0$ and $r_{new} = 2$, as determined by the first hidden layer. Thus, we determine that $p_{out}(0) = -\infty$, $p_{out}(1) = p_{out}(2) = p_{out}(3) = 0$, and $p_{out}(4) = p_{out}(5) = p_{out}(6) = 7$. To illustrate this process further, let us compute the value of $p_{out}(4)$. By analyzing the second hidden layer of ASS-NN, we obtain $w^{(1)} = 6$ and $w^{(2)} = 6$. Additionally, from the third hidden layer, we have $p_{in}(w^{(1)}) = p_{in}(w^{(2)}) = 0$. It follows that $p_{in}(w^{(1)}) < (p_{in}(w^{(2)}) + w_{in})$, and we conclude that the value of $p_{out}(4)$ is equal to 7.

Iteration 2: The inputs to the ASS-NN are $-\infty, 0, 0, 0, 7, 7, 7$ and 34, respectively. We obtain $r_{old} = 2$ and $r_{new} = 7$ from the first hidden layer. Consequently, we obtain the following output values: $p_{out}(0) = -\infty$, $p_{out}(1) = p_{out}(2) = p_{out}(3) = p_{out}(4) = 7$, $p_{out}(5) = 34$, and $p_{out}(6) = 41$. To illustrate the computing process, let us compute the value of $p_{out}(4)$. According to the second hidden layer of ASS-NN, we obtain $w^{(1)} = 6$ and $w^{(2)} = 0$. Furthermore, we have $p_{in}(w^{(1)}) = 7$ and $p_{in}(w^{(2)}) = -\infty$ from the third hidden layer. This implies that $p_{in}(w^{(1)}) > (p_{in}(w^{(2)}) + w_{in})$. It follows that $p_{out}(4) = 7$.

Iteration 3: The inputs of ASS-NN are $-\infty, 7, 7, 7, 7, 34, 41$ and 4, respectively. Consequently, we obtain the following output values: $r_{old} = 7$ and $r_{new} = 8$ from the first hidden layer. Thus, we have $p_{out}(0) = -\infty$, $p_{out}(1) = 7, p_{out}(2) = p_{out}(3) = p_{out}(4) = 11$, $p_{out}(5) = 38$, and $p_{out}(6) = 45$. Let us consider how to compute the value of $p_{out}(4)$. By analyzing the second hidden layer of ASS-NN, we determine that $w^{(1)} = 4$ and $w^{(2)} = 4$. Furthermore, we have $p_{in}(w^{(1)}) = 7$ and $p_{in}(w^{(2)}) = 7$ from the third hidden layer. Given that $p_{in}(w^{(1)}) < (p_{in}(w^{(2)}) + w_{in})$, we conclude that the value $p_{out}(4)$ is equal to 11.

Iteration 4: The inputs of ASS-NN are $-\infty, 7, 11, 11, 11, 38, 45$ and 12, respectively. Utilizing the first hidden layer, we obtain $r_{old} = 8$ and $r_{new} = 10$. Consequently, we have the following output values: $p_{out}(0) = -\infty$, $p_{out}(1) = 7$, $p_{out}(2) = 19$, $p_{out}(3) = 23$, $p_{out}(4) = 38$, $p_{out}(5) = 50$, and $p_{out}(6) = 57$. To illustrate computing process, let us compute the value of $p_{out}(4)$. We can derive $w^{(1)} = 5$ and $w^{(2)} = 3$ from the second hidden layer of ASS-NN. Furthermore, we conclude that $p_{in}(w^{(1)}) = 38$ and $p_{in}(w^{(2)}) = 11$ from the third hidden layer. Since $p_{in}(w^{(1)}) > (p_{in}(w^{(2)}) + w_{in})$, we can obtain the value $p_{out}(4)$, which is equal to 38.

Iteration 5: The inputs of ASS-NN are $-\infty, 7, 19, 23, 38, 50, 57$ and 5, respectively. We have $r_{old} = 10$ and $r_{new} = 11$ as determined by the first hidden layer. Consequently, we obtain the following output values: $p_{out}(0) = -\infty$, $p_{out}(1) = 7$, $p_{out}(2) = 19$, $p_{out}(3) = 28$, $p_{out}(4) = 43$, $p_{out}(5) = 55$, and $p_{out}(6) = 62$. To further illustrate this process, let us

compute the value of $p_{out}(4)$. According to the second hidden layer of ASS-NN, we obtain $w^{(1)} = 4$ and $w^{(2)} = 4$. Additionally, from the third hidden layer we determine that $p_{in}(w^{(1)}) = 38$ and $p_{in}(w^{(2)}) = 38$. Given that $p_{in}(w^{(1)}) < (p_{in}(w^{(2)}) + w_{in})$, it follows that the value $p_{out}(4)$ is equal to 43.

Iteration 6: The inputs of ASS-NN are $-\infty, 7, 19, 28, 43, 55, 62$ and 3, respectively. Utilizing the first hidden layer, we obtain $r_{old} = 11$ and $r_{new} = 11$. Consequently, we have the following output values: $p_{out}(0) = -\infty$, $p_{out}(1) = 10$, $p_{out}(2) = 22$, $p_{out}(3) = 31$, $p_{out}(4) = 43$, $p_{out}(5) = 55$, and $p_{out}(6) = 65$. Take $p_{out}(4)$ as an example. By analyzing the second hidden layer of ASS-NN, we determine that $w^{(1)} = 4$ and $w^{(2)} = 3$. Additionally, from the third hidden layer, we have that $p_{in}(w^{(1)}) = 43$ and $p_{in}(w^{(2)}) = 28$. Given that $p_{in}(w^{(1)}) > (p_{in}(w^{(2)}) + w_{in})$, we can obtain the value $p_{out}(4)$, which is equal to 43.

The values $p_{out}(w, i)$ computed by the ASS-NN are shown in Table 2. Given a positive integer $P$, we can identify a subset of $S$ such that the sum of its elements is closest to $P$ without exceeding $P$. For instance, if we aim to determine a subset of $S$, the sum of the subset elements is closest to 33 but no more than 33. Firstly, we observe the value closest to 33 but no more than 33 in Table 2 is 31, and the corresponding function is $p_{out}(3, 6)$. The solution process for identifying the subset associated with $p_{out}(3, 6)$ proceeds as follows:

1. Given that $p_{out}(3,6) = \max\{p_{out}(3,5), \delta(p_{out}(3,5) + 3)\} = p_{out}(3,5) + 3$, it follows that 3 belongs to the required subset.
2. Given that $p_{out}(3,5) = \max\{p_{out}(3,4), \delta(p_{out}(3,4) + 5)\} = p_{out}(3,4) + 5$, it follows that 5 belongs to the required subset.
3. Given that $p_{out}(3,4) = \max\{p_{out}(4,3), \delta(p_{out}(4,3) + 12)\} = p_{out}(4,3) + 12$, 12 also belongs to our desired subset.
4. Given that $p_{out}(4,3) = \max\{p_{out}(4,2), \delta(p_{out}(4,2) + 4)\} = p_{out}(4,2) + 4$, 4 belongs to the required subset.
5. Given that $p_{out}(4,2) = \max\{p_{out}(6,1), \delta(p_{out}(0,1) + 34)\} = p_{out}(6,1)$, 34 does not belong in our required set.
6. Given that $p_{out}(6,1) = \max\{p_{out}(6,0), \delta(p_{out}(6,0) + 7)\} = p_{out}(6,0) + 7$, we conclude that 7 is included in the required subset.
7. Given that the second argument of $p_{out}(6,0)$ is zero, the solution process concludes here. Ultimately, we determine that the required subset of is $\{3, 5, 12, 4, 7\}$.

Since the value $p_{out}(w, i)$ of ASS-NN serves as an approximation to the SSP, there may exist a discrepancy between $p_{out}(w, i)$ and the actual optimal solution. The values $p_{out}(2, 6)$, $p_{out}(3, 6)$, $p_{out}(5, 6)$, and $p_{out}(6, 6)$ are equal to the actual optimal solution, and the error ratio is zero. The errors between $p_{out}(1, 6)$, $p_{out}(4, 6)$, and the actual optimization are 1. Specifically, the error ratio of $p_{out}(1, 6)$ is 0.091, and $p_{out}(4, 6)$ is 0.023.

**Table 2.** The value of $p_{out}(w, i)$.

| $i$ \ $w$ | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|---|
| **0** | $-\infty$ | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | $-\infty$ | 0 | 0 | 0 | 7 | 7 | 7 |
| **2** | $-\infty$ | 7 | 7 | 7 | 7 | 34 | 41 |
| **3** | $-\infty$ | 7 | 11 | 11 | 11 | 38 | 45 |
| **4** | $-\infty$ | 7 | 19 | 23 | 38 | 50 | 57 |
| **5** | $-\infty$ | 7 | 19 | 28 | 43 | 55 | 62 |
| **6** | $-\infty$ | 10 | 22 | 31 | 43 | 55 | 65 |

### *4.3. Error Analysis*

In order to assess the effectiveness of the proposed ASS-NN in addressing subset problems, we evaluate the degree of error between the approximate solution generated

by ASS-NN and the actual optimal solution. Let $w^{NN}$ be the approximation solution with ASS-NN and $w^{OPT}$ be the actual optimal solution. The degree of error $\delta$ is calculated using Equation (31).

$$\delta = \frac{w^{OPT} - w^{NN}}{w^{OPT}}. \tag{31}$$

**Example 1.** *A set of 20 natural numbers was randomly generated within a range from 1 to 50, resulting in the set S = {50, 48, 46, 45, 44, 43, 32, 30, 29, 25, 24, 20, 16, 15, 14, 13, 10, 6, 4, 1}. We analyzed the error degree across four scenarios with M values of 10, 15, 20, and 25. The corresponding box plots are shown in Figure 7. The maximum errors between the approximate solutions derived from ASS-NN and the actual optimal solution were found to be 3%, 3.5%, 2%, and 1%, respectively. In the scenario where M equals 25, only 5 out of 25 approximate solutions deviated from the actual optimal solution. The median difference across all four cases remained within a margin of 2%.*

**Example 2.** *A total of 20 natural numbers were randomly generated within a range from 1 to 100 to form the set S = {90, 72, 65, 61, 59, 58, 57, 56, 47, 44, 43, 38, 37, 34, 30, 29, 22, 20, 13, 6}. We analyzed the error degrees for four different values of M = 10, 15, 20 and 25, and the corresponding box plot is shown in Figure 8. The maximum errors observed between the approximate solutions derived from ASS-NN and those representing actual optimal solutions are recorded as follows: 6.7%, 5%, 4.5%, and 10%, respectively. These findings indicate that while there exists a minimal median difference across all four scenarios examined here, the variability in error tends to escalate with increasing values of M.*

**Example 3.** *We randomly generated 25 natural numbers between 1 and 50, resulting in the set S = {50, 49, 48, 47, 46, 45, 43, 42, 40, 39, 36, 35, 33, 32, 30, 26, 22, 21, 20, 18, 14, 13, 10, 8, 3}. We analyzed the error degree across four cases with M = 10, 15, 20, and 25. The corresponding box plot is illustrated in Figure 9. The maximum errors between the approximate solutions obtained through ASS-NN and the actual optimal solution were found to be 2.7%, 3.5%, 2.7%, and 6.8%, respectively. Among these four cases of analysis for different values of M, it was observed that the median for M = 25 yielded the lowest value. Furthermore, the median differences across all four scenarios remained within a margin of less than or equal to 1%.*

**Example 4.** *A set of 25 natural numbers was randomly generated within a range from 1 to 100, resulting in the set S = {100, 95, 93, 91, 82, 81, 79, 71, 70, 67, 65, 54, 50, 46, 44, 41, 38, 31, 27, 25, 19, 11, 10, 4, 2}. We analyzed the error degree across four scenarios with M values of 10, 15, 20, and 25. The corresponding box plot is illustrated in Figure 10. The maximum errors between the approximate solutions derived from ASS-NN and the actual optimal solution are recorded as 1.5%, 2.3%, 2%, and 3.8%, respectively. Among these cases examined, M = 15 yielded the lowest median value. Furthermore, the median differences across all four scenarios remained within a margin of 0.5%.*

**Example 5.** *We randomly generated 25 natural numbers between 1 and 200, resulting in the set S = {196, 194, 193, 177, 172, 165, 157, 155, 152, 133, 130, 126, 108, 101, 98, 91, 76, 60, 57, 52, 50, 27, 9, 4, 1}. We analyzed the error degree across four cases with M = 10, 20, 30, and 40. The corresponding box plots are illustrated in Figure 11. The maximum errors between the approximate solutions obtained by ASS-NN and the actual optimal solution were found to be 1.9%, 3.5%, 2.5%, and 6.8%, respectively. Among these four cases, a median for M = 20 was observed to be the lowest. Furthermore, the median differences among all four cases remained within a range of 0.5%.*

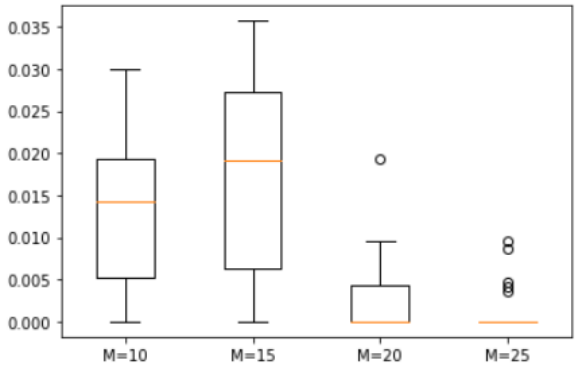


**Figure 7.** Error of Example 1.

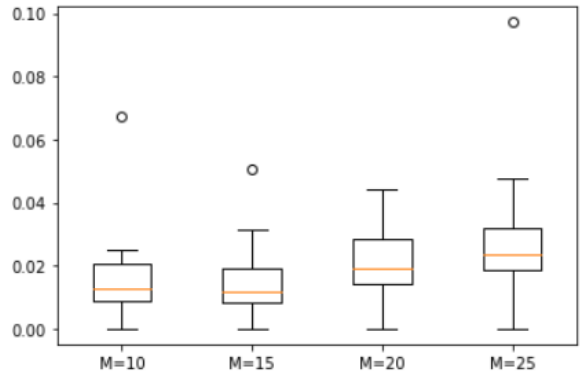


**Figure 8.** Error of Example 2.

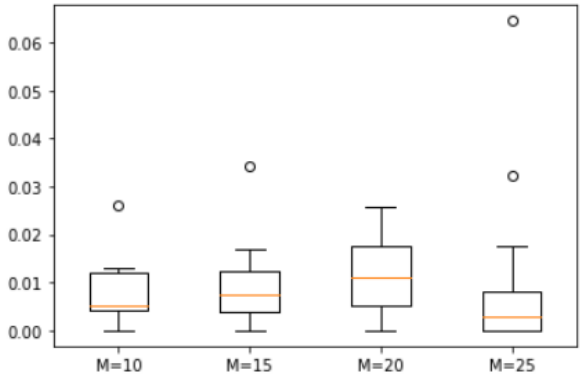


**Figure 9.** Error of Example 3.

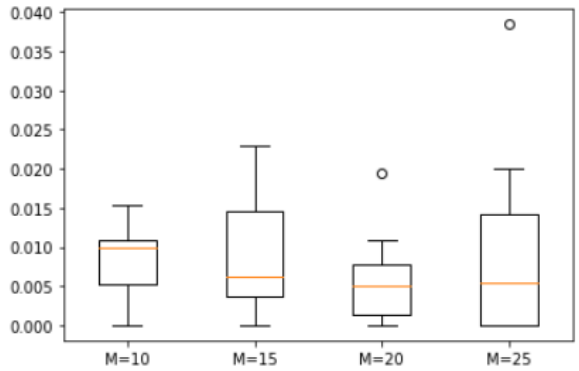


**Figure 10.** Error of Example 4.

**Example 6.** *We randomly generated 25 natural numbers between 1 and 500, resulting in the set* $S = \{461, 450, 409, 391, 362, 349, 310, 293, 291, 276, 263, 238, 202, 199, 135, 128, 121, 120, 119,$

*114, 54, 48, 43, 37, 9}. We analyzed the error degree across four scenarios with M = 20, 30, 40, and 50. The corresponding box plots are shown in Figure 12. The maximum errors between the approximate solutions obtained through ASS-NN and the actual optimal solution were found to be 3.8%, 2.6%, 8%, and 9.2%, respectively. Among these cases, a median of M = 40 was observed to be the lowest. Furthermore, the median differences across all four scenarios remained within a range of 0.5%.*

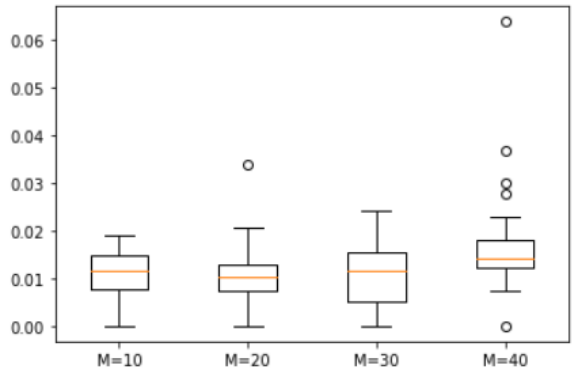


**Figure 11.** Error of Example 5.

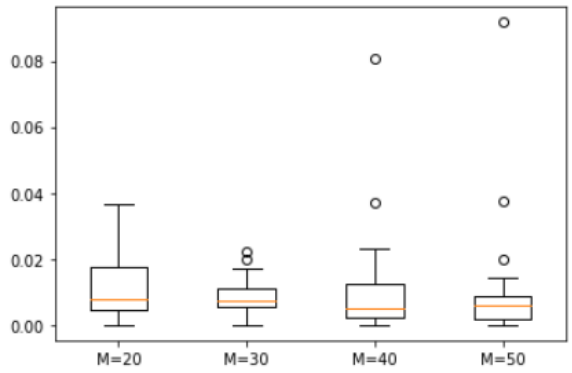


**Figure 12.** Error of Example 6.

According to the box diagram, we observed that an increase in the $M$ value corresponds to a greater number of abnormal values. However, the average difference in error degree for the same dataset across different $M$ values remains minimal. Furthermore, it is evident that the discrepancy between the approximate solution and the actual solution aligns with the theoretical error degree.

Consider Example 6 again, where $M = 50$. In this scenario, the approximate solutions generated by ASS-NN alongside the actual optimal solution are presented in Table 3. Among the total of fifty solutions evaluated by ASS-NN methods; five of these correspond exactly with the actual optimal solution. Furthermore, nineteen solutions exhibited an error value ranging from 1 to 10, twelve solutions had an error value from 11 to 20, and fourteen solutions presented an error value exceeding 20. The maximum observed error value was 46, occurring at $w = 44$, while the highest degree of error reached approximately 9.17% when $w = 2$. These results indicate that the approximate solution method based on ASS-NN effectively addresses SSPs while maintaining a minimal discrepancy between approximate and actual optimal solutions.

**Table 3.** Results of Example 6 with M = 50.

| $w$ | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $w^{NN}$ | 102 | 198 | 326 | 436 | 540 | 654 | 757 | 862 | 971 | 1090 | 1175 | 1289 |
| $w^{OPT}$ | 106 | 218 | 327 | 436 | 545 | 654 | 763 | 872 | 981 | 1090 | 1199 | 1304 |
| $err$ | 4 | 20 | 1 | 0 | 5 | 0 | 6 | 10 | 10 | 0 | 24 | 15 |
| $w$ | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** |
| $w^{NN}$ | 1417 | 1525 | 1614 | 1725 | 1846 | 1961 | 2053 | 2166 | 2274 | 2383 | 2499 | 2593 |
| $w^{OPT}$ | 1417 | 1526 | 1635 | 1744 | 1853 | 1962 | 2071 | 2180 | 2289 | 2398 | 2507 | 2616 |
| $err$ | 0 | 1 | 21 | 19 | 7 | 1 | 18 | 14 | 15 | 15 | 8 | 23 |
| $w$ | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** | **35** | **36** |
| $w^{NN}$ | 2708 | 2828 | 2942 | 3044 | 3158 | 3263 | 3365 | 3483 | 3569 | 3683 | 3797 | 3898 |
| $w^{OPT}$ | 2725 | 2834 | 2943 | 3052 | 3161 | 3270 | 3379 | 3488 | 3597 | 3706 | 3815 | 3924 |
| $err$ | 17 | 6 | 1 | 8 | 3 | 7 | 14 | 5 | 28 | 23 | 18 | 26 |
| $w$ | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** | **48** |
| $w^{NN}$ | 4007 | 4111 | 4208 | 4355 | 4432 | 4546 | 4685 | 4750 | 4870 | 4990 | 5109 | 5223 |
| $w^{OPT}$ | 4033 | 4142 | 4251 | 4360 | 4469 | 4578 | 4687 | 4796 | 4905 | 5014 | 5123 | 5231 |
| $err$ | 26 | 31 | 43 | 5 | 37 | 32 | 2 | 46 | 35 | 24 | 14 | 8 |
| $w$ | **49** | **50** | | | | | | | | | | |
| $w^{NN}$ | 5308 | 5422 | | | | | | | | | | |
| $w^{OPT}$ | 5377 | 5422 | | | | | | | | | | |
| $err$ | 29 | 0 | | | | | | | | | | |

## 5. Discussion

The SSP is a classical combinatorial optimization issue. It is not merely an abstract computational conundrum but also a juncture where theory and practice converge. Its research consistently promotes algorithmic innovation and offers methodological support for practical engineering problems. As of now, the classical approaches for solving the SSP encompass dynamic programming, genetic algorithms, quantum algorithms, double list algorithms, and others. The merit of the dynamic programming method lies in constructing solutions based on recurrence relations and being capable of attaining the optimal solution. Nevertheless, its time and space complexities escalate rapidly with the scale, thus being suitable for small-scale problems [16]. Quantum algorithms can concurrently explore multiple candidate solutions via superposition states and effectively search for the optimal solution in the solution space. However, they are vulnerable to noise interference, have high implementation complexity, and rely on dedicated quantum devices [26]. Genetic algorithms exhibit strong adaptability to large-scale problems but are sensitive to parameters and prone to getting trapped in local optima [15]. The double list algorithm can significantly reduce time complexity but has a high space complexity [5]. Simultaneously, all these methods encounter the combinatorial explosion problem resulting from traversing all possible combination spaces and a lack of generalization ability for similar problems.

Based on the powerful pattern learning capability and end-to-end mapping solution approach of neural network models, and given that RNN can simulate the process of constructing solutions based on recurrence relations in dynamic programming [17], this paper theoretically constructs an RNN for solving the SSP and proves its correctness. For the problem of whether there exists a subset in a set such that the sum of all elements in the subset equals a specified $w$ value, recent research by [24] from the perspective of geometry, which interpreted SSP as a problem of deciding whether the intersection of the positive unit hypercube with the hyperplane contains at least a vertex, solved the problem precisely. The SS-NN network we constructed can solve the problem precisely, too. In Example 1 of Section 4, we present the solution process of SS-NN. The results indicate that SS-NN can not only provide the answer as to whether a subset exists whose sum of all elements

equals a specified $w$ value but also yield solution results for all problems with values less than $w$. However, although both methods can yield exact solutions, the method in the literature [24] can be applied to the simultaneous SSP.

For the problem of finding a subset such that the sum of elements in the subset is closest to but does not exceed a specified value, considering that constructing a neural network model to solve the exact value would be highly complex and demand a considerable amount of computational resources, this paper constructs the ASS-NN model for solving the approximate solution of the sub-problem. Compared to the model for solving the exact solution, this model is simpler but can determine an approximate solution that is very close to the optimal solution. For instance, in Table 3, $w^{NN}$ represents the approximate solution obtained by the ASS-NN model, while $w^{OPT}$ represents the actual optimal solution of the problem. In some cases, the solution obtained by the ASS-NN model is the optimal solution. Experimental results show that the maximum error rate between the approximate solution obtained by the ASS-NN model and the optimal solution is approximately 9.17%. Once it is theoretically ensured that the constructed neural network model can solve the problem, then in practice, by training the model and utilizing the end-to-end mapping approach of the model, it is possible to bypass explicitly traversing all possible combination spaces, thereby avoiding the combinatorial explosion. This is also the distinction between solving the SSP through the RNN model and traditional heuristic algorithms [11] and dynamic programming [19] and represents another approach to solving large-scale SSPs. However, the accuracy of the solution obtained by ASS-NN still needs to be improved.

## 6. Conclusions

In light of the deficiencies of existing dynamic programming, genetic algorithms, and quantum algorithms when addressing the SSP, and inspired by the work of C. Hertrich et al. [37], we proposed two models, namely SS-NN and ASS-NN, for simulating dynamic programming to solve the SSP. Stringent mathematical derivations demonstrate that the proposed neural network models can precisely solve the SSP. Among them, the SS-NN model can determine whether there exists a subset within a set such that the sum of all elements in this subset equals a specified value, while the ASS-NN model is capable of providing an approximate solution that is extremely close to the optimal solution for the problem of finding a subset whose sum of elements is closest to but does not exceed the specified value. Experimental results indicate that the errors between the approximate solutions obtained by the ASS-NN model and the actual optimal solutions are relatively small and highly consistent with theoretical expectations. Our research reveals that RNNs provide a novel approach to resolving the SSP. This approach not only constructs the solution process by emulating the recursive relationship in dynamic programming but also utilizes neural network models to learn the latent patterns in the problem and circumvents the combinatorial explosion problem caused by explicitly traversing all possible combination spaces through an end-to-end mapping approach. However, in practical applications, using RNNs to solve the SSP still confronts several challenges, mainly the following: (1) due to strong data dependence, the cost of generating high-quality training samples is relatively high; (2) for certain SSPs, if an exact solution is sought, it is necessary to construct a complex network model, thereby triggering the demand for a substantial amount of computing resources; (3) the accuracy of the approximate solutions still awaits further enhancement. In future research, we will introduce adversarial generative networks [39] to generate high-quality training samples and integrate the attention mechanism [40] to improve the solution accuracy of the model.

**Author Contributions:** W.L.: Conceptualization, Methodology, Formal analysis, Investigation, and Writing—original draft. Z.W. (Zengkai Wang): Conceptualization, Methodology, Supervision, and

Writing—review and editing. Z.W. (Zijia Wang): Conceptualization, Methodology, Supervision, and Writing—review and editing. Y.J.: Validation and Writing—review and editing. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the National Natural Science Foundation of China (62106055), the Guangdong Natural Science Foundation (2025A1515010256), and the Guangzhou Science and Technology Planning Project (2023A04J0388, 2023A03J0662).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data that support the findings of this study are available from the corresponding author upon request. There are no restrictions on data availability.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Kang, L.; Wan, L.; Li, K. Efficient parallelization of a two-list algorithm for the subset-sum problem on a hybrid CPU/GPU cluster. In Proceedings of the 2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming, Beijing, China, 13–15 July 2014; IEEE: Piscataway, NJ, USA , 2014; pp. 93–98.
2. Ghosh, D.; Chakravarti, N. A competitive local search heuristic for the subset sum problem. *Comput. Oper. Res.* **1999**, *26*, 271–279. [CrossRef]
3. Madugula, M.K.; Majhi, S.K.; Panda, N. An efficient arithmetic optimization algorithm for solving subset-sum problem. In Proceedings of the 2022 International Conference on Connected Systems & Intelligence (CSI), Trivandrum, India, 31 August–2 September 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–7.
4. Wan, L.; Li, K.; Liu, J.; Li, K. GPU implementation of a parallel two-list algorithm for the subset-sum problem. *Concurr. Comput. Pract. Exp.* **2015**, *27*, 119–145. [CrossRef]
5. Wan, L.; Li, K.; Li, K. A novel cooperative accelerated parallel two-list algorithm for solving the subset-sum problem on a hybrid CPU–GPU cluster. *J. Parallel Distrib. Comput.* **2016**, *97*, 112–123. [CrossRef]
6. Dutta, P.; Rajasree, M.S. Efficient reductions and algorithms for variants of Subset Sum. *arXiv* **2021**, arXiv:2112.11020.
7. Ye, Y.; Borodin, A. Priority algorithms for the subset-sum problem. *J. Comb. Optim.* **2008**, *16*, 198–228. [CrossRef]
8. Parque, V. Tackling the Subset Sum Problem with Fixed Size using an Integer Representation Scheme. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June–1 July 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1447–1453.
9. Li, L.; Zhao, K.; Ji, Z. A genetic algorithm to solve the subset sum problem based on parallel computing. *Appl. Math. Inf. Sci.* **2015**, *9*, 921.
10. Kolpakov, R.M.; Posypkin, M.A. Effective parallelization strategy for the solution of subset sum problems by the branch-and-bound method. *Discret. Math. Appl.* **2020**, *30*, 313–325. [CrossRef]
11. Kolpakov, R.; Posypkin, M. Optimality and complexity analysis of a branch-and-bound method in solving some instances of the subset sum problem. *Open Comput. Sci.* **2021**, *11*, 116–126. [CrossRef]
12. Thada, V.; Shrivastava, U. Solution of subset sum problem using genetic algorithm with rejection of infeasible offspring method. *Int. J. Emerg. Technol. Comput. Appl. Sci* **2014**, *10*, 259–262.
13. Wang, R.L. A genetic algorithm for subset sum problem. *Neurocomputing* **2004**, *57*, 463–468. [CrossRef]
14. Bhasin, H.; Singla, N. Modified genetic algorithms based solution to subset sum problem. *Int. J. Adv. Res. Artif. Intell.* **2012**, *1*, 38–41. [CrossRef]
15. Saketh, K.H.; Jeyakumar, G. Comparison of dynamic programming and genetic algorithm approaches for solving subset sum problems. In Proceedings of the Computational Vision and Bio-Inspired Computing: ICCVBIC 2019, Coimbatore, India, 25–26 September 2019; Springer: Cham, Switzerland, 2020; pp. 472–479.
16. Kolpakov, R.; Posypkin, M. Lower time bounds for parallel solving of the subset sum problem by a dynamic programming algorithm. *Concurr. Comput. Pract. Exp.* **2024**, *36*, e8144. [CrossRef]
17. Yang, F.; Jin, T.; Liu, T.Y.; Sun, X.; Zhang, J. Boosting dynamic programming with neural networks for solving np-hard problems. In Proceedings of the 10th Asian Conference on Machine Learning, ACML 2018, Beijing, China, 14–16 November 2018; pp. 726–739.
18. Allcock, J.; Hamoudi, Y.; Joux, A.; Klingelhöfer, F.; Santha, M. Classical and quantum dynamic programming for Subset-Sum and variants. *arXiv* **2021**, arXiv:2111.07059.
19. Fujiwara, H.; Watari, H.; Yamamoto, H. Dynamic Programming for the Subset Sum Problem. *Formaliz. Math.* **2020**, *28*, 89–92

20. Xu, S.; Panwar, S.S.; Kodialam, M.; Lakshman, T. Deep neural network approximated dynamic programming for combinatorial optimization. In Proceedings of the AAAI 2020 Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 1684–1691.
21. Biesner, D.; Gerlach, T.; Bauckhage, C.; Kliem, B.; Sifa, R. Solving subset sum problems using quantum inspired optimization algorithms with applications in auditing and financial data analysis. In Proceedings of the 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau, Bahamas, 12–14 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 903–908.
22. Xu, C.; Zhang, G. Learning-augmented algorithms for online subset sum. *J. Glob. Optim.* **2023**, *87*, 989–1008. [CrossRef]
23. Coron, J.S.; Gini, A. Provably solving the hidden subset sum problem via statistical learning. *Math. Cryptol.* **2021**, *1*, 70–84.
24. Costandin, M. On a Geometric Interpretation Of the Subset Sum Problem. *arXiv* **2024**, arXiv:2410.19024.
25. Zheng, Q.; Zhu, P.; Xue, S.; Wang, Y.; Wu, C.; Yu, X.; Yu, M.; Liu, Y.; Deng, M.; Wu, J.; et al. Quantum algorithm and experimental demonstration for the subset sum problem. *Sci. China Inf. Sci.* **2022**, *65*, 182501. [CrossRef]
26. Zheng, Q.; Yu, M.; Zhu, P.; Wang, Y.; Luo, W.; Xu, P. Solving the subset sum problem by the quantum Ising model with variational quantum optimization based on conditional values at risk. *Sci. China Phys. Mech. Astron.* **2024**, *67*, 280311. [CrossRef]
27. Moon, B. The Subset Sum Problem: Reducing Time Complexity of NP-Completeness with Quantum Search. *Undergrad. J. Math. Model. One Two* **2012**, *4*, 2. [CrossRef]
28. Bernstein, D.J.; Jeffery, S.; Lange, T.; Meurer, A. Quantum algorithms for the subset-sum problem. In Proceedings of the Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013, Limoges, France, 4–7 June 2013; Proceedings 5; Springer: Berlin/Heidelberg, Germany, 2013; pp. 16–33.
29. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.* **2021**, *290*, 405–421. [CrossRef]
30. Thapa, R. A Survey on Deep Learning-Based Methodologies for Solving Combinatorial Optimization Problems . 2020. Available online: https://www.researchgate.net/publication/343876842_A_Survey_on_Deep_Learning-based_Methodologies_for_Solving_Combinatorial_Optimization_Problems (accessed on 1 April 2025).
31. Hopfield, J.J.; Tank, D.W. "Neural" computation of decisions in optimization problems. *Biol. Cybern.* **1985**, *52*, 141–152. [PubMed]
32. Tarkov, M.S. Solving the traveling salesman problem using a recurrent neural network. *Numer. Anal. Appl.* **2015**, *8*, 275–283.
33. Gu, S.; Cui, R. An efficient algorithm for the subset sum problem based on finite-time convergent recurrent neural network. *Neurocomputing* **2015**, *149*, 13–21. [CrossRef]
34. Gu, S.; Hao, T. A pointer network based deep learning algorithm for 0–1 knapsack problem. In Proceedings of the 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI), Xiamen, China, 29–31 March 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 473–477.
35. Zhao, X.; Wang, Z.; Zheng, G. Two-phase neural combinatorial optimization with reinforcement learning for agile satellite scheduling. *J. Aerosp. Inf. Syst.* **2020**, *17*, 346–357.
36. Kechadi, M.T.; Low, K.S.; Goncalves, G. Recurrent neural network approach for cyclic job shop scheduling problem. *J. Manuf. Syst.* **2013**, *32*, 689–699.
37. Hertrich, C.; Skutella, M. Provably good solutions to the knapsack problem via neural networks of bounded size. *INFORMS J. Comput.* **2023**, *35*, 1079–1097.
38. Oltean, M.; Muntean, O. Solving the subset-sum problem with a light-based device. *Nat. Comput.* **2009**, *8*, 321–331. [CrossRef]
39. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA , 2014; Volume 27.
40. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2017; Volume 30.