

BACHELOR THESIS
COMPUTING SCIENCE



RADBOD UNIVERSITY

Implementation of Integer Addition on a Spiking Neural Network

Author:

Evgeniya Ovchinnikova
s1001300

First supervisor:

prof. dr., J.H. Geuvers (Herman)
h.geuvers@cs.ru.nl

Second supervisor:

dr. J.H.P. Kwisthout (Johan)
j.kwisthout@donders.ru.nl

June 14, 2021

Abstract

Neuromorphic computation systems are innovative architectures that intend to eventually either replace or complement the ubiquitous von Neumann architecture hardware. In machine learning applications, where the main focus lies, and for more conventional numerical algorithms. Addition is a basic arithmetic operation that is used in many algorithms. Hence, it is crucial to have a time-, space- and energy-efficient neuromorphic addition algorithm. This paper proposes and compares eleven spiking neural networks for integer addition. The five networks with radix number representation turn out to be the most suitable for performing this arithmetic operation because of their low complexities and other benefits.

Contents

1	Introduction	2
2	Abstract model	3
3	A good network	4
4	Binary representation	5
4.1	Add two integers in binary representation	5
4.2	Add multiple integers in binary representation	9
5	Time representation	11
5.1	Add two integers represented as the time when a neuron spikes	11
5.2	Add multiple integers represented as the time when a neuron spikes	16
5.3	Add two integers represented as the time when a neuron spikes using a radix	19
6	Neurons with predefined numerosities	22
6.1	Add two integers represented as the number of spiking neurons	22
6.2	Add two integers represented as the number of spiking neurons using a radix	24
6.3	Add two integers represented by numerosity-selective neurons	27
6.4	Add two integers represented by numerosity-selective neurons using a radix	30
6.5	Add two integers represented by numerosity-selective neurons with a zero- neuron	33
6.6	Add two integers represented by numerosity-selective neurons with a zero- neuron using a radix	34
7	Compare networks	38
7.1	Binary representation	38
7.2	Time representation	43
7.3	Neurons with predefined numerosities	43
7.4	Binary representation versus time representation	44
7.5	Time representation versus numerosity-selective neurons	44
7.6	Binary representation versus numerosity-selective neurons	45
7.7	Verdict	45
8	Discussion and future work	45
9	Conclusion	46

A Appendix	46
B Appendix	48
C Simulations	49
C.1 Neurons	49
C.2 Synapses	50
C.3 Network	51
C.4 Add two integers in binary representation	52
C.5 Add multiple integers in binary representation	53
C.6 Add two integers represented as the time when a neuron spikes	55
C.7 Add multiple integers represented as the time when a neuron spikes	56

1 Introduction

Moore’s law [10] states that the number of transitions in the integrated circuits doubles about every two years. Moore’s Intel colleague David House has even claimed that the time efficiency doubles every 18 months [7]. The continuation of this progress is of high importance for our digital economy. However, over the last decades, this advancement seems to have slowed down industry-wide. People try to keep up with the pace of Moore’s law. New materials were invented to make transistors smaller. (For example, the high-k dielectric [6].) Parallelism is applied where possible. And new hardware architectures were developed where the processing and data memory are more closely located to each other, addressing the von Neumann bottleneck [3].

One of this innovative hardware is the so-called neuromorphic architecture. Neuromorphic hardware is inspired by the structure of the brain. It is typically characterised by its high levels of parallelism, co-located processing and memory and low power consumption [1]. Although the main focus lies on machine learning, neuromorphic computers can also be successfully applied for more conventional numerical algorithms. The intention is that these new neural computing technologies eventually either replace or complement the popular von Neumann architecture hardware [12].

Addition is a basic arithmetic operation that is used in many algorithms. Hence, it is crucial to have an addition algorithm that is time-, space- and energy-efficient. The aim of this paper is to propose a good neuromorphic addition algorithm for a spiking neural network. The so-called spiking neural networks (SNNs) form a subclass of neuromorphic systems [1]. These SNNs can be seen as the equivalent of what Turing machines are for von Neuman architectures [9]. Thus, this paper uses the SNN as the underlying computational model for neuromorphic computers.

Eleven spiking algorithms for integer addition are proposed, analysed and compared in this paper. The results of this paper may also contribute to a better understanding of the potentials and limitations of the new neuromorphic architecture.

The remainder of this paper is organized is as follows: First, in section 2, the used abstract model for SNN is described. Then, section 3 explains what *a good network* means and defines some constraints that the eleven proposed networks must respect. The eleven networks are discussed in sections 4, 5 and 6. There, the algorithms are defined, described and individually analyzed. For network 4.1 all formal proofs are given. In other sections, the proofs of most theorems, lemmas and complexities are left as an exercise

for the reader. The eleven spiking algorithms are compared to each other in section 7. Section 8 provides a discussion of the results and an outlook for future work. Finally, this paper is concluded with section 9.

2 Abstract model

This paper uses the deterministic discrete-time model for SNN as proposed by Johand Kwisthout and Nils Donselaar in their paper [9].

A spiking neural network is defined as finite directed graph $\mathcal{S} = (N, S)$ where N is the set of neurons (nodes) and S is the set of synapses (nodes). Every neuron $k \in N$ is a tuple (T_k, R_k, m_k) , where $T_k \in \mathbb{Q}_{\geq 0}$ is the threshold, $R_k \in \mathbb{Q}_{\geq 0}$ the reset voltage and $m_k \in [0, 1]$ the leakage constant. Each synapse $s \in S$ is a tuple $(k \in N, n \in N, d_{k \rightarrow n} \in \mathbb{N}_{>0}, w_{k \rightarrow n} \in \mathbb{Q})$, where k is the pre-synaptic neuron, n the post-synaptic neuron, $d_{k \rightarrow n}$ the synaptic delay and $w_{k \rightarrow n}$ the synaptic weight. For convenience, the notation $k \rightarrow n = (d_{k \rightarrow n}, w_{k \rightarrow n})$ will be used in this paper to refer to a synapse.

The behaviour of neuron k at time $t \in \mathbb{Z}$ is defined by its membrane potential $u_k(t) \in \mathbb{Q}_{\geq 0}$. Neuron k spikes when this potential reaches the threshold T_k . Then two things happen:

1. For each neuron j such that $k \rightarrow j \in S$, this signal from neuron k goes to j , reaches j $d_{k \rightarrow j}$ time-steps later and $w_{k \rightarrow j}$ is added to j 's potential.
2. The potential of neuron k resets to R_k .

If neuron k does not spike at time t , then the leakage constant m_k defines which part of the current potential is carried over to the next time-step.

Thus, the potential $u_k(t)$ of neuron k at time j can be expressed as follows:

$$u_k(t) = \begin{cases} \max(0, R_k + \sum_{j \in N \wedge k \rightarrow j \in S} w_{j \rightarrow k} * 1_{\{u_j(t-d_{j \rightarrow k}) \geq T_j\}}) & \text{if } u_k(t-1) \geq T_k \\ \max(0, m_k * u_k(t-1) + \sum_{j \in N \wedge j \rightarrow k \in S} w_{j \rightarrow k} * 1_{\{u_j(t-d_{j \rightarrow k}) \geq T_j\}}) & \text{if } u_k(t-1) < T_k \end{cases}$$

Note that $u_k(t) \geq T_k$ is equivalent to "*neuron k spikes at time t* ".

Such a network $\mathcal{S} = (N, S)$ gets some input represented as spikes on particular input neurons. The calculation is finished when this network returns some output, represented by spiking behaviour of particular output neurons.

Furthermore, this paper uses the following conventions:

- The calculation of a network starts at time 0. The potential $u_k(0)$ of neuron k is called the *initial voltage* or *initial potential*. By default, if not stated differently, this initial voltage is 0 and $\forall k \in N, \forall t < 0 : u_k(t) = 0$.
- By default, if not stated differently, neurons and synapses have parameters $m = R = 0$ and $T = d = w = 1$. (However, if these parameters are not explicitly specified, then they are probably not important. For example, it does matter what the threshold of an input neuron is in section 5.2. It is sufficient to know that this neuron spikes once at time A_i .)
- This paper contains some visual representations of the networks. A solid arrow in these graphs indicates an excitatory synapse with a positive weight, and a dashed arrow indicates an inhibitory synapse with a negative weight. The numbers on the arrows are the weights of the corresponding synapses. If an arrow does not have a number, then the weight is 1 by default (or -1 in case of an inhibitory synapse).

- Number zero is both *positive* and *negative*. E.g., a *positive* variable can be equal to 0 and a *non-negative* variable is greater than 0.

3 A good network

As mentioned earlier, this paper aims to find a good network that can perform integer addition. But what does this mean, *a good network*?

- A good network is efficient, or, equivalently, has low computational complexities. One of the advantages of the neuromorphic computer systems is their low energy consumption in comparison with the conventional von Neumann machines. Therefore, besides the well-known time and space complexities, Kwisthout and Donselaar introduced energy as a new complexity measure [9]. The resource constraints for an SNN are defined as a tuple $R_S = (\text{TIME}, \text{SPACE}, \text{ENERGY})$. TIME is here the number of time steps \mathcal{S} may use to calculate and return the expected output (time complexity), SPACE the number of neurons $|N|$ (space complexity), and ENERGY the number of spikes (energy complexity). All three of them are functions on the size of the input. An efficient network has a low time, space and energy complexity.
- A good network is general. It is useful if the same network can be applied for different input values (within some boundaries and all represented in the same way). Thus, in general, the input and the output should not be represented in the network structure. For example, the input values should not be encoded as synaptic delays. However, note that generality goes typically at the cost of efficiency [9].
- A good network has no significant practical limitations. Everything is possible in theory, but there are a lot of physical constraints in practice. Think about limited precision or maximal voltage a computer can use. A theoretical algorithm is useless if it exceeds the boundaries of a real system on which it should be executed.

Besides the aspects mentioned above, each network can have other benefits and drawbacks. These should also be borne in mind when comparing different circuits in section 7.

Furthermore, an addition-network must respect the following three constraints:

- The network must be non-hybrid. Some authors (for example, [2]) propose spiking neural algorithms that are partly executed on an SNN and partly on another kind of machine. However, the networks discussed in this paper are intended to be used as building blocks for larger networks. Furthermore, there are already efficient addition algorithms for conventional system (e.g., carry-lookahead adder, which has a theoretical time complexity $O(\log n)$). Hence, there is no point in using a hybrid spiking neural algorithm for integer addition.

In [9] the SNN complexity classes are defined by two tuples: R_S as explained above and $R_T = (\text{TIME}, \text{SPACE})$. R_T are here the time and space constraints that a Turing T machine would need to create the SNN \mathcal{S} . However, this paper considers only non-hybrid spiking algorithms. We do not know anything about this external mechanism, which creates these spiking networks. Hence, this paper does examine the complexities of T .

- The network should use the same representation on the input as on the output.
- The network should be an exact adder: it should return the exact sum of the given input integers.

4 Binary representation

Before algorithmically adding two (or more) numbers, the first thing you need to do is decide how these numbers should be represented. The first two proposed networks were inspired by the binary number representation in the paper [4]. In the abstract model for SNN, a neuron gives deterministic binary information: it spikes, or it does not spike. Hence, an obvious representation is the binary numeral system. In the first two networks, an integer is represented by a sequence of neurons. If a neuron spikes, then the corresponding bit is 1. If the neuron does not spike, then the corresponding bit is 0.

In practice, it is impossible to have an infinite number of neurons. So we choose an $m \geq 0$ and define the input and output as $(m + 1)$ -bit integers.

4.1 Add two integers in binary representation

The spiking neural network of this section adds two $(m + 1)$ -bit integers in binary representation.

Task

Take $m \geq 0$. Given integers $A = A_0 + A_1 \cdot 2 + \dots + A_m \cdot 2^m$ and $B = B_0 + B_1 \cdot 2 + \dots + B_m \cdot 2^m$, calculate $C = (A + B) \bmod 2^{m+1} = C_0 + C_1 \cdot 2 + \dots + C_m \cdot 2^m$, where $A_i, B_i, C_i \in \{0, 1\}$ for all bit-positions $0 \leq i \leq m$.

Network $\mathcal{S} = (N, S)$

Neurons N

- For $0 \leq i \leq m$, neuron $\mathbf{a}_i = (1, 0, 0)$ gets an excitatory spike of weight 1 at time 0 if $A_i = 1$
- For $0 \leq i \leq m$, neuron $\mathbf{b}_i = (1, 0, 0)$ gets an excitatory spike of weight 1 at time 0 if $B_i = 1$
- For $0 \leq i \leq m$, neuron $\mathbf{c}_i = (1, 0, 0)$
- For $0 \leq i \leq m$, neuron $\mathbf{-2}_i = (2, 0, 0)$

Synapses S

- For $0 \leq i \leq m$, for $i \leq j \leq m$ synapse $\mathbf{a}_i \rightarrow \mathbf{-2}_j = (1, 2^{i-j})$
- For $0 \leq i \leq m$, for $i \leq j \leq m$ synapse $\mathbf{b}_i \rightarrow \mathbf{-2}_j = (1, 2^{i-j})$
- For $0 \leq i \leq m$, for $i \leq j \leq m$ synapse $\mathbf{a}_i \rightarrow \mathbf{c}_j = (2, 2^{i-j})$
- For $0 \leq i \leq m$, for $i \leq j \leq m$ synapse $\mathbf{a}_i \rightarrow \mathbf{c}_j = (2, 2^{i-j})$
- For $0 \leq i \leq m$, synapse $\mathbf{-2}_i \rightarrow \mathbf{c}_i = (1, -2)$

Input

- For $0 \leq i \leq m$, neuron \mathbf{a}_i gets an excitatory spike of weight 1 at time 0 if $A_i = 1$ (thus, $u_{\mathbf{a}_i}(0) = A_i$)
- For $0 \leq i \leq m$, neuron \mathbf{b}_i gets an excitatory spike of weight 1 at time 0 if $B_i = 1$ (thus, $u_{\mathbf{b}_i}(0) = B_i$)

Output

- For $0 \leq i \leq m$, neuron \mathbf{c}_i spikes at time 2 if $C_i = 1$

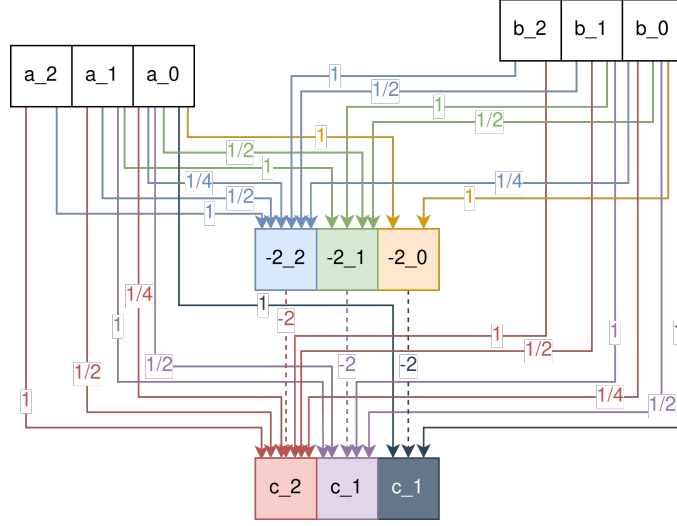


Figure 1: An example with $m = 2$.

How and why does this work?

In principle, this network uses the simple pencil-and-paper addition method with a carry-lookahead mechanism.

Express the carry-over with variable carry_i from Definition A.1 using $M = N = 2$, $m = m$, $A_1 = A$ and $A_2 = B$. Then from Lemma A.5 follows:

$$C_i = (A_i + B_i + \lfloor (A_{i-1} + B_{i-1}) \cdot 2 + (A_{i-2} + B_{i-2}) \cdot 4 + \dots + (A_0 + B_0) \cdot 2^i \rfloor) \mod 2 \text{ for } 0 \leq i \leq m$$

$$\text{Define } D_i = A_i + B_i + (A_{i-1} + B_{i-1}) \cdot 2 + (A_{i-2} + B_{i-2}) \cdot 4 + \dots + (A_0 + B_0) \cdot 2^i.$$

Without neuron -2_i , the potential of neuron c_i would become D_i at time 2. The threshold of c_i is 1. Thus c_i would spike if $D_i \geq 1$. However, neuron -2_i gets a potential D_i at time 1. The threshold of -2_i is 2. Thus, if $D_i \geq 2$, neuron -2_i spikes at time 1 and decreases the potential of c_i with 2. Hence, the potential of c_i becomes $D_i - 1_{D_i \geq 2}$. Furthermore, $\lfloor D_i - 1_{D_i \geq 2} \rfloor = \lfloor D_i \rfloor \mod 2 = C_i$. Hence neuron c_i spikes at time 2 if $C_i = 1$.

That was the algorithm, in short. Now, let us give more a formal proof that this circuit works. We begin with the following lemma:

Lemma 4.1. *For $0 \leq i \leq m$, holds $u_{-2,i}(1) = D_i$.*

Proof. $u_{-2,i}(0) = 0$. Thus:

$$\begin{aligned} u_{-2,i}(1) &= \max(0, \sum_{j \in N \wedge j \rightarrow -2,i \in S} w_{j \rightarrow -2,i} \cdot 1_{\{u_j(1-d_{j \rightarrow -2,i}) \geq T_j\}}) \\ &= \sum_{j=0}^m 2^{j-i} \cdot (1_{\{u_{a,j}(0) \geq 1\}} + 1_{\{u_{b,j}(0) \geq 1\}}) \\ &= \sum_{j=0}^m 2^{j-i} \cdot (1_{\{A_j=1\}} + 1_{\{B_j=1\}}) \\ &= \sum_{j=0}^m 2^{j-i} \cdot (A_j + B_j) \quad \text{because } A_i, B_i \in \{0, 1\} \\ &= D_i \end{aligned}$$

□

Now we can prove the following theorem:

Theorem 4.2. *For $0 \leq i \leq m$, neuron c_i spikes at time 2 if $C_i = 1$*

Proof. Take $0 \leq i \leq m$. We know $u_{c_i}(0) = 0$. Thus:

$$\begin{aligned} u_{c_i}(1) &= \max(0, \sum_{j \in N \wedge j \rightarrow c_i \in S} w_{j \rightarrow c_i} \cdot 1_{\{u_j(1-d_{j \rightarrow c_i}) \geq T_j\}}) \\ &= -2 \cdot 1_{\{u_{-2_i}(0) \geq 2\}} + \sum_{j=0}^m 2^{j-i} \cdot (1_{\{u_{a_j}(-1) \geq 1\}} + 1_{\{u_{b_j}(-1) \geq 1\}}) \\ &= 0 \end{aligned}$$

Hence:

$$\begin{aligned} u_{c_i}(2) &= \max(0, \sum_{j \in N \wedge j \rightarrow c_i \in S} w_{j \rightarrow c_i} \cdot 1_{\{u_j(1-d_{j \rightarrow c_i}) \geq T_j\}}) \\ &= -2 \cdot 1_{\{u_{-2_i}(1) \geq 2\}} + \sum_{j=0}^m 2^{j-i} \cdot (1_{\{u_{a_j}(0) \geq 1\}} + 1_{\{u_{b_j}(0) \geq 1\}}) \\ &= -2 \cdot 1_{\{D_i \geq 2\}} + \sum_{j=0}^m 2^{j-i} \cdot (1_{\{A_j=1\}} + 1_{\{B_j=1\}}) \quad (\text{Lemma 4.1}) \\ &= D_i - 2 \cdot 1_{\{D_i \geq 2\}} \end{aligned}$$

From Lemma A.4 follows $0 \leq \lfloor D_i \rfloor \leq 3$. Thus:

$$\begin{aligned} \lfloor u_{c_i}(2) \rfloor &= \lfloor D_i - 2 \cdot 1_{D_i \geq 2} \rfloor \\ &= \lfloor D_i \rfloor - 2 \cdot 1_{D_i \geq 2} \\ &= \lfloor D_i \rfloor \mod 2 \\ &= C_i \end{aligned}$$

The threshold of neuron c_i is 1. $C_i \in \{0, 1\}$. Thus, neuron c_i spikes at time 2 if $C_i = 1$. \square

Complexities

Space: $\Theta(m)$

Time: $O(1)$

Energy: $O(m)$

Let us prove these complexities.

Theorem 4.3. *The space complexity of this network is $\Theta(m)$.*

Proof. The network contains the following neurons:

- $m + 1$ neurons a_i
- $m + 1$ neurons b_i
- $m + 1$ neurons c_i
- $m + 1$ neurons -2_i

Thus, there are $4m + 4 = \Theta(m)$ neurons in total.

□

Theorem 4.4. *The time complexity of this network is $O(1)$.*

Proof. We want to proof that $\forall k \in N, \forall t \geq 3 : u_k(t) = 0$

Take $t \geq 2$ and $0 \leq i \leq m$. Then, by induction on i :

$$\begin{aligned} u_{\mathbf{a}.i}(t-1) &= \sum_{j \in N \wedge j \rightarrow \mathbf{a}.i \in S} w_{j \rightarrow \mathbf{a}.i} \cdot 1_{\{u_j(t-d_{j \rightarrow \mathbf{a}.i}) \geq T_j\}} \\ &= 0 \end{aligned}$$

and

$$\begin{aligned} u_{\mathbf{b}.i}(t-1) &= \sum_{j \in N \wedge j \rightarrow \mathbf{b}.i \in S} w_{j \rightarrow \mathbf{b}.i} \cdot 1_{\{u_j(t-d_{j \rightarrow \mathbf{b}.i}) \geq T_j\}} \\ &= 0 \end{aligned}$$

Then, by induction in i holds:

$$\begin{aligned} u_{-2.i}(t) &= \max(0, \sum_{j \in N \wedge j \rightarrow -2.i \in S} w_{j \rightarrow -2.i} \cdot 1_{\{u_j(t-d_{j \rightarrow -2.i}) \geq T_j\}}) \\ &= \sum_{j=0}^m 2^{j-i} \cdot (1_{\{u_{\mathbf{a}.j}(t-1) \geq 1\}} + 1_{\{u_{\mathbf{b}.j}(t-1) \geq 1\}}) \\ &= \sum_{j=0}^m 2^{j-i} \cdot 0 \\ &= 0 \end{aligned}$$

Now, let us look at neurons $\mathbf{c}.i$. By induction on i :

$$\begin{aligned} u_{\mathbf{c}.i}(t+1) &= \max(0, \sum_{j \in N \wedge j \rightarrow \mathbf{c}.i \in S} w_{j \rightarrow \mathbf{c}.i} \cdot 1_{\{u_j(t+1-d_{j \rightarrow \mathbf{c}.i}) \geq T_j\}}) \\ &= -2 \cdot 1_{\{u_{-2.i}(t) \geq 2\}} + \sum_{j=0}^m 2^{j-i} \cdot (1_{\{u_{\mathbf{a}.j}(t-1) \geq 1\}} + 1_{\{u_{\mathbf{b}.j}(t-1) \geq 1\}}) \\ &= -2 \cdot 0 + \sum_{j=0}^m 2^{j-i} \cdot 0 \\ &= 0 \end{aligned}$$

Thus, indeed $\forall k \in N, \forall t \geq 3 : u_k(t) = 0$. Hence, no neurons spikes after time 2 and the time complexity is $2 = O(1)$.

□

Theorem 4.5. *The energy complexity is $O(m)$*

Proof. The energy complexity is bounded above by the product of time complexity and space complexity $\Theta(m) \cdot O(1) = O(m)$. Furthermore, it is possible that all neurons $\mathbf{a}.i$ spike at time 0. Hence, the energy complexity is $O(m)$.

□

Benefits

- It may seem that this network adds only positive integers, but using two's complement for negative numbers makes subtraction possible as well.
- If the sum of the input is too big to fit in the neurons \mathbf{c}_i (overflow), then this network still gives modulo the correct answer. This is expressed above as $C = (A + B) \bmod 2^{m+1}$.
- This network is reusable. This network can add two $(m + 1)$ -bit integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons \mathbf{a}_i and \mathbf{b}_i have fired, they can get new input to calculate the sum.

Drawbacks

- This network has one significant practical limitation. The weights of the synapses are defined as powers of $\frac{1}{2}$. The exponent can go to infinity in theory, but every real system has a limited precision in practice. Thus, there is a maximum for the number of \mathbf{c}_i neurons, or this network may give incorrect answers.

4.2 Add multiple integers in binary representation

The spiking neural network in this section adds multiple $(m + 1)$ -bit integers in binary representation.

Task

Take $m \geq 0$. Given integers $A_i = A_{i,0} + A_{i,1} \cdot 2 + \dots + A_{i,m} \cdot 2^m$ (for $1 \leq i \leq M$), calculate $C = (A_1 + \dots + A_M) \bmod 2^{m+1} = C_0 + C_1 \cdot 2 + \dots + C_m \cdot 2^m$, where $A_{i,j}, C_j \in \{0, 1\}$ for all bit-positions $0 \leq j \leq m$ and integers $1 \leq i \leq M$.

Network $\mathcal{S} = (N, S)$

Neurons N

For $1 \leq i \leq M$, for $0 \leq j \leq m$, neuron $\mathbf{a}_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_{i,j} = 1$

For $0 \leq i \leq m$, neuron $\mathbf{c}_i = (1, 0, 0)$

For $0 \leq i \leq m$, for $1 \leq j \leq \lfloor M + M \cdot (1 - 2^{-i}) \rfloor$, neuron $\mathbf{S}_{j-i} = (j, 0, 0)$

Synapses S

For $0 \leq i \leq m$, for $1 \leq j \leq M$, for $0 \leq k \leq i$, for $1 \leq p \leq \lfloor M + M \cdot (1 - 2^{-i}) \rfloor$, synapse $\mathbf{a}_{j-k} \rightarrow \mathbf{S}_{p-i} = (1, 2^{k-i})$

For $0 \leq i \leq m$, for $1 \leq j \leq \lfloor M + M \cdot (1 - 2^{-i}) \rfloor$, synapse $\mathbf{S}_{j-i} \rightarrow \mathbf{c}_i = (1, (-1)^{j-1})$

Input

For $1 \leq i \leq M$, for $0 \leq j \leq m$, neuron $\mathbf{a}_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_{i,j} = 1$ (thus, $u_{\mathbf{a}_{i,j}}(0) = A_{i,j}$)

Output

For $0 \leq i \leq m$, neuron \mathbf{c}_i spikes at time 2 if $C_i = 1$

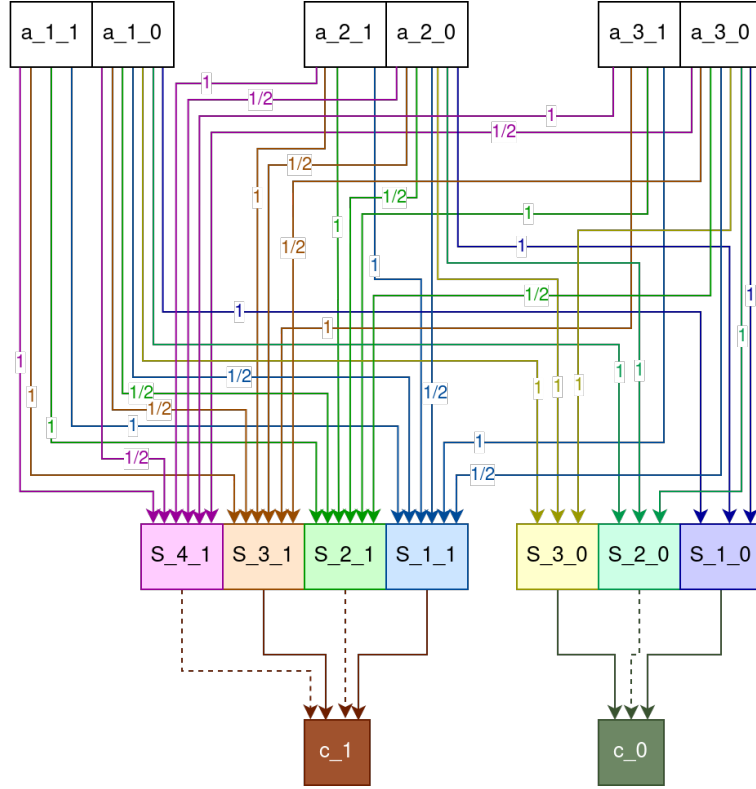


Figure 2: An example with $m = 1$ and $M = 3$.

How and why does this work?

The idea is similar to the previous section.

Express the carry-over with variable carry_i from Definition A.1 using $M = M$, $N = 2$, $m = m$, and $A_i = A_i$ for $1 \leq i \leq M$. Then from Lemma A.5 follows:

$$C_i = \left\lfloor \sum_{j=0}^i (A_{1,j} + \dots + A_{M,j}) \cdot 2^{j-i} \right\rfloor \mod 2$$

Define $D_i = \sum_{j=0}^i (A_{1,j} + \dots + A_{M,j}) \cdot 2^{j-i}$.

At time 1, each neuron \mathbf{S}_{i-j} gets spikes from neurons \mathbf{a}_{k-p} , where $1 \leq k \leq M$ and $0 \leq p \leq j$, such that the potential of \mathbf{S}_{i-j} becomes D_j . (From Lemma A.4 follows $0 \leq D_j \leq \lfloor M + M \cdot (1 - 2^{-j}) \rfloor$. This is why there are $\lfloor M + M \cdot (1 - 2^{-j}) \rfloor$ neurons \mathbf{S}_{i-j} for each bit position j .)

Lemma 4.6. $\forall 0 \leq j \leq m, \forall 1 \leq i \leq \lfloor M + M \cdot (1 - 2^{-j}) \rfloor : u_{\mathbf{S}_{i-j}}(1) = D_j$

The threshold of \mathbf{S}_{i-j} is i . Thus \mathbf{S}_{i-j} spikes if $D_j \geq i$. These spikes from neuron \mathbf{S}_{i-j} go to neuron \mathbf{c}_j . At time 2, neuron \mathbf{c}_j gets inhibiting spikes from neurons \mathbf{S}_{q-j} with an even q and excitatory spikes from neurons \mathbf{S}_{q-j} with an odd q . Thus the potential of \mathbf{c}_j becomes $\left\lfloor \frac{\lfloor D_j \rfloor + 1}{2} \right\rfloor - \left\lfloor \frac{\lfloor D_j \rfloor}{2} \right\rfloor = \lfloor D_j \rfloor \mod 2 = C_j$.

Lemma 4.7. $\forall 0 \leq i \leq m : u_{\mathbf{c}_i}(2) = C_i$

The threshold of \mathbf{c}_j is 1 and $C_i \in \{0, 1\}$. Thus, \mathbf{c}_j spikes at time 2 if $C_i = 1$.

Theorem 4.8. $\forall 0 \leq i \leq m : \text{neuron } \mathbf{c}_i \text{ spikes at time 2 if } C_i = 1.$

Complexities

Space: $\Theta(M \cdot m)$

Time: $O(1)$

Energy: $O(M \cdot m)$

This network has the same benefits and drawbacks as the previous network.

Benefits

- Using two's complement for negative numbers makes subtraction possible.
- If the sum of the input is too big to fit in the neurons c_i , then this network still gives modulo the correct answer. This is expressed above as $C = (A_1 + \dots + A_M) \bmod 2^{m+1}$.
- This network is reusable. This network can add two $(m + 1)$ -bit integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons $a_{i,j}$ have fired, they can get new input to calculate the sum.

Drawbacks

- This network has one significant practical limitation. The weights of the synapses are defined as powers of $\frac{1}{2}$. The exponent can go to infinity in theory, but every real system has a limited precision in practice. Thus, there is a maximum for the number of c_i neurons, or this network may give incorrect answers.

5 Time representation

In the next three spiking networks, a number is represented as the time when the corresponding neuron spikes. All these three networks work on the same principle. Assume we want to add M positive integers A_i ($1 \leq i \leq M$) and say $C = A_1 + \dots + A_M$. Then $(C - A_1) + \dots + (C - A_M) = C * (M - 1)$. So, we could take a neuron with some high potential N and gradually decrease its potential by $M - 1$ with each step. Meanwhile, for each $1 \leq i \leq M$, from time A_i , we increase the potential by 1. This way, the potential is at its lowest at time $\max\{A_1, \dots, A_M\}$ – it is equal to $N - C + \max\{A_1, \dots, A_M\}$ then – and the potential reaches N again at time C .

NB: the potential of a neuron is greater than or equal to 0. Thus, N should be greater than or equal to $C - \max\{A_1, \dots, A_M\}$.

5.1 Add two integers represented as the time when a neuron spikes

The spiking neural network in this section adds two integers represented as the time when a corresponding neuron spikes.

Task

Assume $N \geq 1$. Given positive integers A and B such that $N > \min(A, B)$, calculate $C = A + B$.

Network $\mathcal{S} = (N, S)$

Neurons N

Neuron **A** gets an excitatory spike of weight 1 at time A (thus, $u_A(A) = 1$)

Neuron **B** gets an excitatory spike of weight 1 at time B (thus, $u_B(B) = 1$)

Neuron **a** = $(1, 1, 0)$

Neuron **b** = $(1, 1, 0)$

Neuron **D** = $(2, 0, 1)$

Neuron **C** = $(N, 0, 1)$ with initial voltage $N - 1$

Neuron **inf** = $(1, 1, 0)$ with initial voltage 1

Synapses S

Synapse **A** \rightarrow **D** = $(1, 1)$

Synapse **B** \rightarrow **D** = $(1, 1)$

Synapse **A** \rightarrow **a** = $(1, 1)$

Synapse **B** \rightarrow **b** = $(1, 1)$

Synapse **D** \rightarrow **C** = $(1, 1)$

Synapse **a** \rightarrow **C** = $(2, 1)$

Synapse **b** \rightarrow **C** = $(2, 1)$

Synapse **inf** \rightarrow **C** = $(3, -1)$

Input

Neuron **A** gets an excitatory spike of weight 1 at time A (thus, $u_A(A) = 1$)

Neuron **B** gets an excitatory spike of weight 1 at time B (thus, $u_B(B) = 1$)

Output

Neuron **C** spikes once at time $A + B + 2 = C + 2$ (thus, $u_C(C + 2) \geq N$ and $\forall t < C + 2 : u_C(t) < N$)

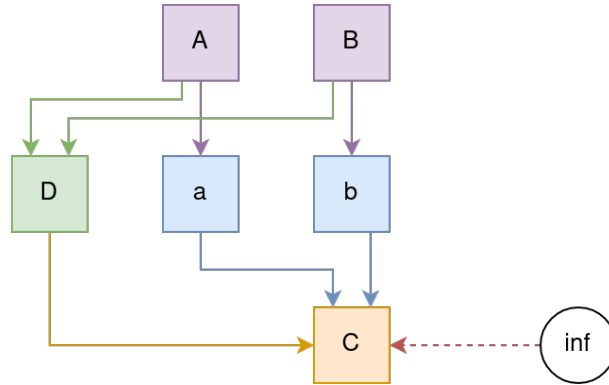


Figure 3: A visualisation of the network.

How and why does this work?

Without loss of generality, assume $A \leq B$. The idea is that the potential of **C** first decreases to $N - A - 1$, then remains constant until time $B + 3$, and finally slowly increases back to its threshold such that **C** spikes at time $A + B + 2$.

Neuron **inf** starts with potential 1. This neuron has threshold 1 and reset voltage 1. So neuron **inf** begins constantly spiking at time 0.

Lemma 5.1. $\forall t \geq 0 : u_{\text{inf}}(t) = 1$

The threshold of **inf** is 1. Thus, **inf** is constantly spiking.

In the meanwhile, neuron **A** spikes at time A . Neuron **A** is connected to neuron **a**. Neuron **a** has threshold 1 and reset voltage 1. So neuron **a** begins constantly firing when it receives the spike from **A** at time $A + 1$.

Lemma 5.2. $\forall t \geq A + 1 : u_a(t) = 1$ and $\forall t < A + 1 : u_a(t) = 0$

The threshold of **a** is 1. Thus, from time $A + 1$, neuron **a** is constantly spiking.

Similarly, neuron **B** and neuron **b**:

Lemma 5.3. $\forall t \geq B + 1 : u_b(t) = 1$ and $\forall t < B + 1 : u_b(t) = 0$

The threshold of **b** is 1. Thus, from time $B + 1$, neuron **b** is constantly spiking.

Neuron **A** is connected to neuron **D**. At time A , neuron **A** spikes and sends this signal to neuron **D**, increasing neuron **D**'s potential to 1. Neuron **B** is also connected to neuron **D**. At time $B \geq A$, neuron **B** spikes and sends this signal to neuron **D**, increasing neuron **D**'s potential to 2. This potential is equal to the threshold of neuron **D**. So neuron **D** spikes at time $B + 1$.

Lemma 5.4.

$$\forall t \in \mathbb{Z} : u_D(t) = \begin{cases} 1 & \text{if } t \in [A + 1, B + 1) \\ 2 & \text{if } t = B + 1 \\ 0 & \text{else} \end{cases}$$

The threshold of **D** is 2. Thus, neuron **D** spikes at time $B + 1$.

Thus, at any given time t , neuron **C** can receive four spikes:

1. an inhibitory signal from neuron **inf** if $t \geq 3$ (follows from Lemma 5.1 and $d_{\text{inf} \rightarrow \text{C}} = 3$)
2. an excitatory signal from neuron **a** if $t \geq A + 3$ (follows from Lemma 5.2 and $d_{\text{a} \rightarrow \text{C}} = 2$)
3. an excitatory signal from neuron **b** if $t \geq B + 3$ (follows from Lemma 5.3 and $d_{\text{b} \rightarrow \text{C}} = 2$)
4. an excitatory signal from neuron **D** if $t = B + 2$ (follows from Lemma 5.4 and $d_{\text{D} \rightarrow \text{C}} = 1$)

Hence, we can express the potential of neuron **C** as follows:

Lemma 5.5. $\forall t \in [0, A + B + 2] :$

$$u_C(t) = \begin{cases} N - 1 & \text{if } t = 0 \\ u_C(t - 1) + 1_{\{t=B+2\}} + 1_{\{t \geq B+3\}} + 1_{\{t \geq A+3\}} - 1_{\{t \geq 3\}} & \text{if } A + B + 2 \geq t \geq 1 \end{cases}$$

This expression above is equivalent to:

Lemma 5.6. $\forall t \in [0, A + B + 2] : u_C(t) = N - 1_{\{t < B+2\}} - (t - 2) \cdot 1_{\{t \geq 3\}} + (t - A - 2) \cdot 1_{\{t \geq A+3\}} + (t - B - 2) \cdot 1_{\{t \geq B+3\}}$

Proof. Proof by induction on t .

$$\begin{aligned} u_{\mathbf{c}}(0) &= N - 1 \quad \text{by definition} \\ &= N - 1_{\{0 < B+2\}} - (0 - 2) \cdot 1_{\{0 \geq 3\}} + (0 - A - 2) \cdot 1_{\{0 \geq A+3\}} + (0 - B - 2) \cdot 1_{\{0 \geq B+3\}} \end{aligned}$$

Take $A + B + 2 \geq t > 0$. Assume $u_{\mathbf{c}}(t-1) = N - 1_{\{t-1 < B+2\}} + (t-1-2) \cdot 1_{\{t-1 \geq 3\}} + (t-1-A-2) \cdot 1_{\{t-1 \geq A+3\}} + (t-1-B-2) \cdot 1_{\{t-1 \geq B+3\}}$. Then:

$$\begin{aligned} u_{\mathbf{c}}(t) &= u_{\mathbf{c}}(t-1) + 1_{\{t=B+2\}} + 1_{\{t \geq B+3\}} + 1_{\{t \geq A+3\}} - 1_{\{t \geq 3\}} \\ &= N - 1_{\{t < B+3\}} - (t-3) \cdot 1_{\{t \geq 4\}} + (t-3-A) \cdot 1_{\{t \geq A+4\}} + (t-B-3) \cdot 1_{\{t \geq B+4\}} \\ &\quad + 1_{\{t=B+2\}} + 1_{\{t \geq B+3\}} + 1_{\{t \geq A+3\}} - 1_{\{t \geq 3\}} \\ &= N - 1_{\{0 < B+2\}} - (t-2) \cdot 1_{\{t \geq 3\}} + (t-A-2) \cdot 1_{\{t \geq A+3\}} + (t-B-2) \cdot 1_{\{t \geq B+3\}} \end{aligned}$$

□

Use this lemma to prove the following two lemmas.

Lemma 5.7. $u_{\mathbf{c}}(A + B + 2) = N$

Proof.

$$\begin{aligned} u_{\mathbf{c}}(A + B + 2) &= N - 1_{\{A+B+2 < B+2\}} - (A + B + 2 - 2) \cdot 1_{\{A+B+2 \geq 3\}} + \\ &\quad (A + B + 2 - A - 2) \cdot 1_{\{A+B+2 \geq A+3\}} + \\ &\quad (A + B + 2 - B - 2) \cdot 1_{\{A+B+2 \geq B+3\}} \\ &= N - 0 - (A + B) \cdot 1_{\{A+B \geq 1\}} + B \cdot 1_{\{B \geq 1\}} + A \cdot 1_{\{A \geq 1\}} \\ &= N - 0 - A - B + A + B \\ &= N \end{aligned}$$

□

Lemma 5.8. $\forall 0 \leq t < A + B + 2 : u_{\mathbf{c}}(t) < N$

Proof. We know:

$$u_{\mathbf{c}}(t) = N - 1_{\{t < B+2\}} - (t-2) \cdot 1_{\{t \geq 3\}} + (t-A-2) \cdot 1_{\{t \geq A+3\}} + (t-B-2) \cdot 1_{\{t \geq B+3\}}$$

There are four options:

1. $A = B = 0$

Then $\forall 0 \leq t < 2 = A + B + 2 : u_{\mathbf{c}}(t) = N - 1 < N$

2. $A = 0 < 1 \leq B$

Then $\forall 0 \leq t < B + 2 = A + B + 2 : u_{\mathbf{c}}(t) = N - 1 + (t-2 - (t-2)) \cdot 1_{\{t \geq A+3=3\}} = N - 1 < N$

3. $0 < 1 \leq A = B$

Then:

- $\forall 0 \leq t < 3 \leq B + 2 : u_{\mathbf{c}}(t) = N - 1 < N$
- $\forall 3 \leq t < B + 2 : u_{\mathbf{c}}(t) = N - 1 - (t-2) < N$

- $u_{\mathbf{C}}(B + 2) = N - A < N$
- $\forall B + 3 = A + 3 \leq t < B + A + 2 : u_{\mathbf{C}}(t) = N - (t - 2) + (t - A - 2) + (t - B - 2) = N + t - A - B - 2 < N$

4. $0 < 1 \leq A < B$

Then:

- $\forall 0 \leq t < 3 \leq B + 2 : u_{\mathbf{C}}(t) = N - 1 < N$
- $\forall 3 \leq t < A + 3 \leq B + 2 : u_{\mathbf{C}}(t) = N - 1 - (t - 2) < N$
- $\forall A + 3 \leq t < B + 2 : u_{\mathbf{C}}(t) = N - 1 - A < N$
- $u_{\mathbf{C}}(B + 2) = N - A < N$
- $\forall B + 3 \leq t < B + A + 2 : u_{\mathbf{C}}(t) = N - (t - 2) + (t - A - 2) + (t - B - 2) = N + t - A - B - 2 < N$

□

The following theorem follows directly from Lemma 5.7 and Lemma 5.8.

Theorem 5.9. *Neuron \mathbf{C} spikes once at time $C + 2$.*

Complexities

Space: (1)

Time: $\Theta(A + B)$

Energy: $\Theta(A + B)$

The proof of these complexities are easy and are left as an exercise for the reader. A hint for the energy complexity: it follows from Lemma 5.1, Lemma 5.2, Lemma 5.3, Lemma 5.4, Theorem 5.9 and the time complexity.

Benefits

- If indeed $N > \min(A, B)$, then there is no overflow possible at all.

Drawbacks

- This network has a practical limitation. The initial potential of neuron \mathbf{C} has to be $N - 1$ where $N > \min(A, B)$. In theory, this potential can go to infinity. However, every real system has its physical boundary. Thus, there is a physical maximum for the input.
- Subtraction (addition with negative integers) is not directly possible with this representation. There are some ways around this. For example, assume $A < 0$. Use $A + D$ instead of A where D such that $A + D \geq 0$. In this case, neuron \mathbf{C} would at time $A + D + B + 2$, and, knowing what D is, $A + B$ can be easily deduced. However, this method may not always be applicable.
- There is no automatic cleanup after neuron \mathbf{C} has spiked: neurons \mathbf{a} , \mathbf{b} and \mathbf{inf} keep firing. This can be easily fixed. Just add inhibiting synapses from \mathbf{C} to neurons \mathbf{a} , \mathbf{b} , \mathbf{inf} and neuron \mathbf{C} itself (because of the delays). This is done in the network in the section 5.3. However, I do not like this solution. It is too ugly to my taste.

- This problem with neurons **a**, **b** and **inf** makes the network also not directly reusable when the calculation has finished. Just adding inhibiting synapses will not solve this. All the potentials must be reset to correct initial values as well.

5.2 Add multiple integers represented as the time when a neuron spikes

The spiking neural network in this section adds multiple integers represented as the time when a corresponding neuron spikes.

Task

Assume $N \geq 1$. Given positive integers A_1, \dots, A_M (for $M \geq 1$) such that $N > A_1 + \dots + A_M - \max(A_1, \dots, A_M)$, calculate $C = A_1 + \dots + A_M$.

Network $\mathcal{S} = (N, S)$

Neurons N

For $1 \leq i \leq M$, neuron **A** $_i$ gets an excitatory spike of weight 1 at time A_i (thus, $u_{\mathbf{A}_i}(A_i) = 1$)

For $1 \leq i \leq M$, neuron **a** $_i = (1, 1, 0)$

Neuron **D** = $(M, 0, 1)$

Neuron **C** = $(N, 0, 1)$ with initial voltage $N - 1$

Neuron **inf** = $(1, 1, 0)$ with initial voltage 1

Synapses S

For $1 \leq i \leq M$, synapse **A** $_i \rightarrow \mathbf{D} = (1, 1)$

For $1 \leq i \leq M$, synapse **A** $_i \rightarrow \mathbf{a}_i = (2, 1)$

Synapse **D** $\rightarrow \mathbf{C} = (1, 1)$

For $1 \leq i \leq M$, synapse **a** $_i \rightarrow \mathbf{C} = (1, 1)$

Synapse **inf** $\rightarrow \mathbf{C} = (3, -(M - 1))$

Input

For $1 \leq i \leq M$, neuron **A** $_i$ gets an excitatory spike of weight 1 at time A_i (thus, $u_{\mathbf{A}_i}(A_i) = 1$)

Output

Neuron **C** spikes once at time $A_1 + \dots + A_M + 2 = C + 2$ (thus, $u_{\mathbf{C}}(C + 2) \geq N$ and $\forall t < C + 2 : u_{\mathbf{C}}(t) < N$)

How and why does this work?

Without loss of generality, assume $A_1 \leq \dots \leq A_M$. The idea is that the potential of **C** first decreases to $N - A_1 - \dots - A_{M-1} - 1$, then remains constant until time $A_M + 3$, and finally slowly increases back to its threshold such that **C** spikes at time $C + 2$.

Neuron **inf** starts with potential 1. This neuron has threshold 1 and reset voltage 1. So neuron **inf** begins constantly spiking at time 0.

Lemma 5.10. $\forall t \geq 0 : u_{\mathbf{inf}}(t) = 1$

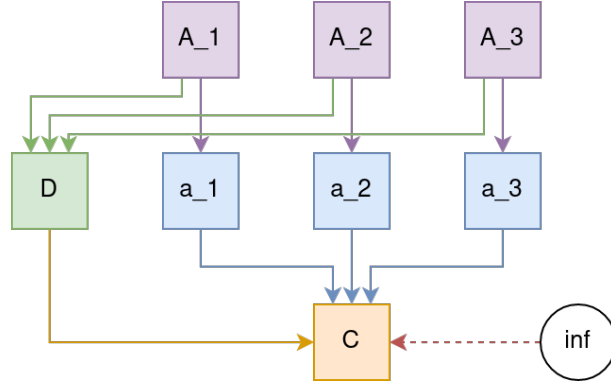


Figure 4: An example with $M = 3$.

The threshold of **inf** is 1. Thus, **inf** is constantly spiking.

In the meanwhile, each neuron A_i spikes at time A_i . Neuron A_i is connected to neuron a_i . Neuron a_i has threshold 1 and reset voltage 1. So neuron a_i begins constantly firing when it receives the spike from a_i at time $A_i + 2$.

Lemma 5.11. $\forall 1 \leq i \leq M, \forall t \geq A_i + 2 : u_{a_i}(t) = 1$ and $\forall 1 \leq i \leq M, \forall t < A_i + 2 : u_{a_i}(t) = 0$

The threshold of a_i is 1. Thus, from time $A_i + 2$, neuron a_i is constantly spiking.

Neuron A_i is connected to neuron D. At time $A_i + 1$, neuron A spikes and sends this signal to neuron D, increasing neuron D's potential with 1. Thus, at time $A_M + 1$, the potential of neuron D becomes M . This potential is equal to the threshold of neuron D. So neuron D spikes at time $A_M + 1$.

Lemma 5.12.

$$\forall t \in \mathbb{Z} : u_D(t) = \begin{cases} i & \text{if } t \in [A_i + 1, A_{i+1} + 1) \text{ for } 1 \leq i < M \\ M & \text{if } t = A_M + 1 \\ 0 & \text{else} \end{cases}$$

The threshold of D is M . Thus, neuron D spikes at time $A_M + 1$.

Thus, at any given time t , neuron C receives the following spikes:

1. an inhibitory signal from neuron **inf** if $t \geq 3$ (follows from Lemma 5.10 and $d_{\text{inf} \rightarrow C} = 3$)
2. for $1 \leq i \leq M$, an excitatory signal from neuron a_i if $t \geq A_i + 3$ (follows from Lemma 5.11 and $d_{a_i \rightarrow C} = 1$)
3. an excitatory signal from neuron D if $t = A_M + 2$ (follows from Lemma 5.12 and $d_{D \rightarrow C} = 1$)

Hence, we can express the potential of neuron C as follows:

Lemma 5.13. $\forall t \in [0, C + 2] :$

$$u_C(t) = \begin{cases} N - 1 & \text{if } t = 0 \\ u_C(t - 1) + 1_{\{t=A_M+2\}} + 1_{\{t \geq A_1+3\}} + \dots + 1_{\{t \geq A_M+3\}} - (M - 1) \cdot 1_{\{t \geq 3\}} & \text{if } C + 2 \geq t \geq 1 \end{cases}$$

This expression above is equivalent to:

Lemma 5.14. $\forall t \in [0, C + 2] : u_{\mathbf{C}}(t) = N - 1_{\{t < A_M + 2\}} - (t - 2) \cdot (M - 1) \cdot 1_{\{t \geq 3\}} + (t - A_1 - 2) \cdot 1_{\{t \geq A_1 + 3\}} + \dots + (t - A_M - 2) \cdot 1_{\{t \geq A_M + 3\}}$

The proof is similar to Lemma 5.6.

The proofs of the following two lemmas are similar to Lemma 5.7 and Lemma 5.8, and use Lemma 5.14.

Lemma 5.15. $u_{\mathbf{C}}(C + 2) = N$

Lemma 5.16. $\forall 0 \leq t < C + 2 : u_{\mathbf{C}}(t) < N$

The following theorem follows directly from Lemma 5.15 and Lemma 5.16.

Theorem 5.17. *Neuron \mathbf{C} spikes once at time $C + 2$.*

Complexities

Space: $\Theta(M)$

Time: $\Theta(A_1 + \dots + A_M)$

Energy: $\Theta(M \cdot (A_1 + \dots + A_M))$

The proof of these complexities are easy and are left as an exercise for the reader. A hint for the energy complexity: it follows from Lemma 5.10, Lemma 5.11, Lemma 5.12, Theorem 5.17 and the time complexity.

This network has similar benefits and drawbacks as the network in the previous section.

Benefits

- If indeed $N > A_1 + \dots + A_M - \max(A_1, \dots, A_M)$, then there is no overflow possible at all.

Drawbacks

- This network has a practical limitation. The initial potential of neuron \mathbf{C} has to be $N - 1$ where $N > A_1 + \dots + A_M - \max(A_1, \dots, A_M)$. In theory, this potential can go to infinity. However, every real system has its physical boundary. Thus, there is a physical maximum for the input.
- Subtraction (addition with negative integers) is not directly possible with this representation. There are some ways around this. For example, assume $A_i < 0$. Use $A_i + D$ instead of A_i where D such that $A_i + D \leq 0$. In this case, neuron \mathbf{C} would at time $C + D + 2$, and, knowing what D is, C can be easily deduced. However, this method may not always be applicable.
- There is no automatic cleanup after neuron \mathbf{C} has spiked: neurons \mathbf{a}_i and \mathbf{inf} keep firing. This can be easily fixed. Just add inhibiting synapses from \mathbf{C} to neurons \mathbf{a}_i , \mathbf{inf} and neuron \mathbf{C} itself (because of the delays). This is done in the network in the section 5.3. However, I do not like this solution. It is too ugly to my taste.
- This problem with neurons \mathbf{a}_i and \mathbf{inf} makes the network also not directly reusable when the calculation has finished. Just adding inhibiting synapses will not solve this. All the potentials must be reset to correct initial values as well.

5.3 Add two integers represented as the time when a neuron spikes using a radix

The networks in the two previous sections take a linear amount of time to calculate the sum. This would be highly inefficient for large numbers. Using a radix could be a solution. So, in this section, the spiking neural network adds two positive integers represented as the time when a neuron spikes, applying radix $N \geq 2$. The network from Section 5.1 is used as a building block here.

In practice, it is impossible to have an infinite number of neurons. So we choose an $m \geq 0$ and define the input and output as $(m + 1)$ -digit integers.

Task

Given integers $A = A_0 + A_1 \cdot N + \dots + A_m \cdot N^m$ and $B = B_0 + B_1 \cdot N + \dots + B_m \cdot N^m$, calculate $C = (A + B) \bmod N^{m+1} = C_0 + C_1 \cdot N + \dots + C_m \cdot N^m$, where $0 \leq A_i, B_i, C_i < N$ for all digit-positions $0 \leq i \leq m$.

Network $\mathcal{S} = (N, S)$

Neurons N

- For $0 \leq i \leq m$, neuron A_i gets an excitatory spike of weight 1 at time $A_i + (N + 1) \cdot i$
- For $0 \leq i \leq m$, neuron B_i gets an excitatory spike of weight 1 at time $B_i + (N + 1) \cdot i$
- For $0 \leq i \leq m$, neuron $D_i = (2, 0, 1)$
- For $0 \leq i \leq m$, neuron $a_i = (1, 1, 0)$
- For $0 \leq i \leq m$, neuron $b_i = (1, 1, 0)$
- For $0 \leq i \leq m$, neuron $\text{sum}_i = (2 \cdot N, 0, 1)$ with initial voltage $2 \cdot N - 1$
- For $0 \leq i \leq m$, neuron N_i gets an excitatory spike of weight 1 at time $(N + 1) \cdot i$
- For $1 \leq i \leq m$, neuron $N+1_i = (2, 0, 1)$ with initial voltage 1
- For $0 \leq i \leq m$, neuron $\text{inf}_i = (1, 1, 0)$
- For $0 \leq i \leq m$, neuron $X_i = (1, 1, 1)$
- For $0 \leq i \leq m$, neuron $x_i = (N, 0, 1)$
- For $0 \leq i \leq m$, neuron $C_i = (2, 0, 0)$
- Neuron **END** gets an excitatory spike of weight 1 at time $(N + 1) \cdot m + 2 \cdot N + 2$
- Neuron **E** = $(1, 1, 0)$

Synapses N

- For $0 \leq i \leq m$, synapse $A_i \rightarrow a_i = (1, 1)$
- For $0 \leq i \leq m$, synapse $A_i \rightarrow D_i = (1, 1)$
- For $0 \leq i \leq m$, synapse $B_i \rightarrow b_i = (1, 1)$
- For $0 \leq i \leq m$, synapse $B_i \rightarrow D_i = (1, 1)$
- For $0 \leq i \leq m$, synapse $D_i \rightarrow \text{sum}_i = (1, 1)$
- For $0 \leq i \leq m$, synapse $a_i \rightarrow \text{sum}_i = (2, 1)$
- For $0 \leq i \leq m$, synapse $b_i \rightarrow \text{sum}_i = (2, 1)$
- For $1 \leq i \leq m$, synapse $N_i \rightarrow N+1_i = (1, 1)$
- For $0 \leq i \leq m$, synapse $N_i \rightarrow \text{inf}_i = (2, 1)$
- For $1 \leq i \leq m$, synapse $N+1_i \rightarrow \text{sum}_i = (1, -1)$
- For $0 \leq i \leq m$, synapse $\text{inf}_i \rightarrow \text{sum}_i = (1, -1)$
- For $0 \leq i \leq m$, synapse $\text{sum}_i \rightarrow X_i = ((N + 1) \cdot (m - i) + 1, 1)$
- For $0 \leq i \leq m - 1$, synapse $\text{sum}_i \rightarrow N+1_(i + 1) = (1, -1)$
- For $0 \leq i \leq m$, synapse $\text{sum}_i \rightarrow \text{sum}_i = (1, -1)$
- For $0 \leq i \leq m$, synapse $\text{sum}_i \rightarrow \text{inf}_i = (1, -1)$
- For $0 \leq i \leq m$, synapse $\text{sum}_i \rightarrow a_i = (1, -1)$

For $0 \leq i \leq m$, synapse $\text{sum_}i \rightarrow \text{b_}i = (1, -1)$
 For $0 \leq i \leq m$, synapse $\text{X_}i \rightarrow \text{x_}i = (1, 1)$
 For $0 \leq i \leq m$, synapse $\text{x_}i \rightarrow \text{C_}i = (1, 1)$
 For $0 \leq i \leq m$, synapse $\text{E} \rightarrow \text{C_}i = (1, 1)$
 Synapse $\text{END} \rightarrow \text{E} = (1, 1)$

Input

For $0 \leq i \leq m$, neuron $\text{A_}i$ gets an excitatory spike of weight 1 at time $A_i + (N + 1) \cdot i$
 For $0 \leq i \leq m$, neuron $\text{B_}i$ gets an excitatory spike of weight 1 at time $B_i + (N + 1) \cdot i$
 For $0 \leq i \leq m$, neuron $\text{N_}i$ gets an excitatory spike of weight 1 at time $(N + 1) \cdot i$
 Neuron END gets an excitatory spike of weight 1 at time $(N + 1) \cdot m + 2 \cdot N + 2$

Output

For $0 \leq i \leq m$, neuron $\text{C_}i$ spikes once at time $(N + 1) \cdot m + 2 \cdot N + C_i + 4$.

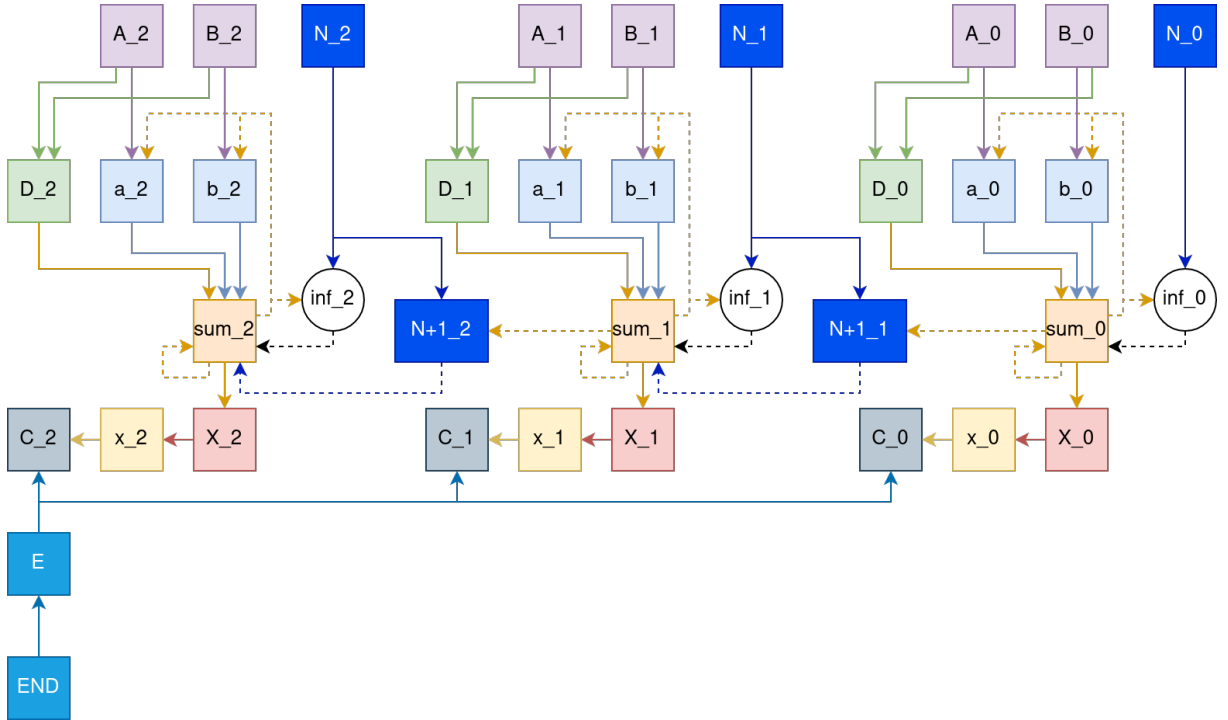


Figure 5: An example with $m = 2$.

How and why does this work?

The idea is similar to the pen-and-paper method of addition. Use carry_i from Definition A.1 with $N = N$, $M = 2$, $m = m$, $A_1 = A$ and $A_2 = B$.

To understand the whole circuit, let us look at a subnetwork that consists of neurons with index i . Neuron $\text{A_}i$ spikes at time $(N + 1) \cdot i + A_i$, neuron $\text{B_}i$ spikes at time $(N + 1) \cdot i + B_i$ and neuron $\text{N_}i$ spikes at time $(N + 1) \cdot i$ activating $\text{inf_}i$. We've already discussed how $\text{D_}i$, $\text{a_}i$, $\text{b_}i$, $\text{inf_}i$ and $\text{sum_}i$ work together in the previous sections. Without the synapse from $\text{N+1_}i$ to $\text{sum_}i$, neuron $\text{sum_}i$ would spike at time $(N + 1) \cdot i + A_i + B_i + 2$. This happens for $i = 0$. Take $i \geq 1$. If $A_{i-1} + B_{i-1} + \text{carry}_{i-1} \geq N$, then $C_i = A_i + B_i + 1$. So $\text{sum_}i$ should spike at $(N + 1) \cdot i + A_i + B_i + 2 + 1$. Neuron $\text{N+1_}i$ is there to regulate this. At time $(N + 1) \cdot i + 1$, neuron $\text{N+1_}i$ sends an inhibiting signal to neuron $\text{sum_}i$, decreasing its potential to $2 \cdot N - 2$, such that neurons $\text{a_}i$ and $\text{b_}i$ need to spike one more time to

make sum_i reach its threshold. Thus, sum_i spikes at $(N+1) \cdot i + A_i + B_i + 2 + 1$. If $A_{i-1} + B_{i-1} + \text{carry}_{i-1} < N$, then $C_i = A_i + B_i$ and sum_i should spike at $(N+1) \cdot i + A_i + B_i + 2$. This is regulated by the connection $\text{sum}_{i-1} \rightarrow \text{N+1}_i$. If $A_{i-1} + B_{i-1} + \text{carry}_{i-1} < N$, then sum_{i-1} spikes before $(N+1) \cdot i + 1$ inhibiting neuron N+1_i . Thus, neuron N+1_i won't spike at time $(N+1) \cdot i + 1$ and won't decrease the potential of sum_i . Hence, sum_i spikes at $(N+1) \cdot i + A_i + B_i + 2$. Now, using induction can be easily proven that sum_i spikes at time $(N+1) \cdot i + A_i + B_i + 2 + \text{carry}_i$ for all indices i . To prevent sum_i from repeatedly spiking and inf_i from unnecessary spiking, sum_i sends an inhibiting signal to itself, a_i , b_i and inf_i .

Lemma 5.18. $\forall 1 \leq i \leq m : \text{N+1}_i$ spikes once at time $(N+1) \cdot i + 1$ if $\text{carry}_i = 1$

Lemma 5.19. $\forall 0 \leq i \leq m : \text{sum}_i$ spikes once at time $(N+1) \cdot i + A_i + B_i + 2 + \text{carry}_i$

Neuron sum_i activates neuron X_i , which will constantly spike starting from time $(N+1) \cdot m + A_i + B_i + 2 + \text{carry}_i + 1$. Those spikes go to neuron x_i . Neuron x_i behaves like a tumble bucket that is gradually being filled with water. When such a bucket fills, it turns over and spills all the water. Likewise, the potential of x_i increases with each spike from X_i until it reaches the threshold. This happens at moments $(N+1) \cdot m + A_i + B_i + 2 + \text{carry}_i + 1 + N \cdot k$ with $k \geq 1$. Then x_i fires, its potential resets to 0, and the process starts over. In the meanwhile, neuron END spikes at time $(N+1) \cdot m + 2 \cdot N + 2$, activating neuron E . In response, neuron E begins constantly firing from time $(N+1) \cdot m + 2 \cdot N + 3$. At time $(N+1) \cdot m + 2 \cdot N + ((A_i + B_i + \text{carry}_i) \bmod N) + 4 = (N+1) \cdot m + 2 \cdot N + C_i + 4$, two signals reach neuron C_i simultaneously making it fire.

Lemma 5.20. $\forall 0 \leq i \leq m : \text{X}_i$ spikes constantly from time $(N+1) \cdot m + A_i + B_i + 2 + \text{carry}_i + 1$

Lemma 5.21. $\forall 0 \leq i \leq m : \text{x}_i$ spikes at times $(N+1) \cdot m + A_i + B_i + 2 + \text{carry}_i + 1 + N \cdot k$ with $k \geq 1$

Lemma 5.22. $\forall 0 \leq i \leq m : \text{E}$ spikes constantly from time $(N+1) \cdot m + 2 \cdot N + 3$

Theorem 5.23. $\forall 0 \leq i \leq m : \text{C}_i$ spikes once at time $(N+1) \cdot m + 2 \cdot N + C_i + 4$

Complexities

The computation is finished when the last neuron C_i has spiked.

Space: $\Theta(m)$

Time: $\Theta(N \cdot m)$

Energy: $\Theta(N \cdot m)$

Benefits

- If the sum of the input is too big to fit in the C_i neurons, then this network still gives modulo the correct answer. This is expressed above as $C = (A + B) \bmod N^{m+1}$.
- Using N 's complement makes addition with negative numbers possible as well. See Appendix B.

Drawbacks

- This network has a practical limitation. The initial potential of neurons sum_i has to be $2 \cdot N - 1$ and neurons x_i should be able to keep a potential up to N . In theory, these potentials can go to infinity. However, every real system has its physical boundary. Thus, there is a physical maximum for the radix N .

- Another practical limitation concerns the delays of the synapses $\text{sum}_i \rightarrow \mathbf{X}_i$. In theory, this delay $(N + 1) \cdot (m - i)$ can go to infinity. However, in practice, the delays are encoded in bits and have a physical boundary.
- There is no automatic cleanup after all neurons \mathbf{C}_i have spiked: some neurons keep firing. This can be easily fixed. Just add inhibiting synapses where necessary. This is partially done for neurons inf_i , \mathbf{a}_i , \mathbf{b}_i and sum_i . However, I do not like this solution. It is too ugly to my taste.
- This problem with neurons that keep firing makes the network not directly reusable when the calculation has finished. Just adding inhibiting synapses will not solve this. All the potentials must be reset to correct initial values as well.

Other notes

- The time complexity is $\Theta(N \cdot m)$. This is because, unfortunately, it is impossible to add all digits in parallel. You need carry_i to calculate C_i . But carry_i may be not defined until time $(N + 1) \cdot (i - 1) + N + 2$. This network can be changed to slightly decrease the calculation time: if $\text{sum}_{(i-1)}$ spikes before $(N + 1) \cdot (i - 1) + N + 2$, then we know that $\text{carry}_i = 0$ and computations for sum_i can start earlier. However, this would make the network more complex and does not change the time complexity.
- Using the network from Section 5.2 instead of the network from Section 5.1, the network from this section can be easily extended to add multiple numbers simultaneously.
- Besides the actual input for neurons \mathbf{A}_i and \mathbf{B}_i , this network also needs external input for neurons \mathbf{N}_i and neuron END to make them spike at time $(N + 1) \cdot i$ and $(N + 1) \cdot m + 2 \cdot N + 2$, respectively.

6 Neurons with predefined numerosities

Andreas Nieder and Stanislas Dehaene [11] and Brian Butterworth [5] discuss how numbers are represented in our brains. Monkeys have the so-called numerosity-selective neurons that are tuned to specific numerosities. Such a neuron would fire with a higher frequency to its preferred numerosity than it would to others.

For example, a monkey sees four bananas on a tree. Then both the neuron that is tuned to fourness and the neuron that is tuned to fiveness start spiking. However, the fourness neuron fires more frequently than the fiveness neuron. This way, the monkey realises that it sees indeed four, not five, bananas. Next, the monkey turns its head and sees five other monkeys looking at the bananas. Again, both the fourness neuron and the fiveness neuron react. But now the fiveness neuron fires more frequently than the fourness neuron. The monkey realises that five other monkeys are competing for these four bananas, decides it is too many and chooses to eat something else.

The spiking neural networks in the following subsections are inspired by these numerosity-selective neurons and try to simulate some aspects of the behaviour.

6.1 Add two integers represented as the number of spiking neurons

The spiking neural network in this section adds two numbers represented as the number of neurons that spike. It is impossible to have an infinite number of neurons in practice.

Let us express the boundaries as positive integers a^-, a^+, b^- and b^+ .

Task

Given integers A and B such that $a^- \leq A \leq a^+$ and $b^- \leq B \leq b^+$, calculate $C = A + B$.
Note: hence $c^- := a^- + b^- \leq C \leq c^+ := a^+ + b^+$.

Network $\mathcal{S} = (N, S)$

Neurons N

- For $1 \leq i \leq a^+$, neuron $A+_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq A$
- For $1 \leq i \leq a^-$, neuron $A-_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq -A$
- For $1 \leq i \leq b^+$, neuron $B+_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq B$
- For $1 \leq i \leq b^-$, neuron $B-_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq -B$
- For $1 \leq i \leq c^+$, neuron $C+_i = (i, 0, 0)$
- For $1 \leq i \leq c^-$, neuron $C-_i = (i, 0, 0)$

Synapses S

- For $1 \leq i \leq a^+$, for $1 \leq i \leq c^+$, synapse $A+_i \rightarrow C+_i = (1, 1)$
- For $1 \leq i \leq a^+$, for $1 \leq i \leq c^-$, synapse $A+_i \rightarrow C-_i = (1, -1)$
- For $1 \leq i \leq a^-$, for $1 \leq i \leq c^+$, synapse $A-_i \rightarrow C+_i = (1, -1)$
- For $1 \leq i \leq a^-$, for $1 \leq i \leq c^-$, synapse $A-_i \rightarrow C-_i = (1, 1)$
- For $1 \leq i \leq b^+$, for $1 \leq i \leq c^+$, synapse $B+_i \rightarrow C+_i = (1, 1)$
- For $1 \leq i \leq b^+$, for $1 \leq i \leq c^-$, synapse $B+_i \rightarrow C-_i = (1, -1)$
- For $1 \leq i \leq b^-$, for $1 \leq i \leq c^+$, synapse $B-_i \rightarrow C+_i = (1, -1)$
- For $1 \leq i \leq b^-$, for $1 \leq i \leq c^-$, synapse $B-_i \rightarrow C-_i = (1, 1)$

Input

- For $1 \leq i \leq a^+$, neuron $A+_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq A$
- For $1 \leq i \leq a^-$, neuron $A-_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq -A$
- For $1 \leq i \leq b^+$, neuron $B+_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq B$
- For $1 \leq i \leq b^-$, neuron $B-_i$ gets an excitatory spike of weight 1 at time 0 if $i \geq -B$

Output

- For $1 \leq i \leq c^+$, neuron $C+_i$ spikes at time 1 if $i \leq C$
- For $1 \leq i \leq c^-$, neuron $C-_i$ spikes at time 1 if $i \leq -C$

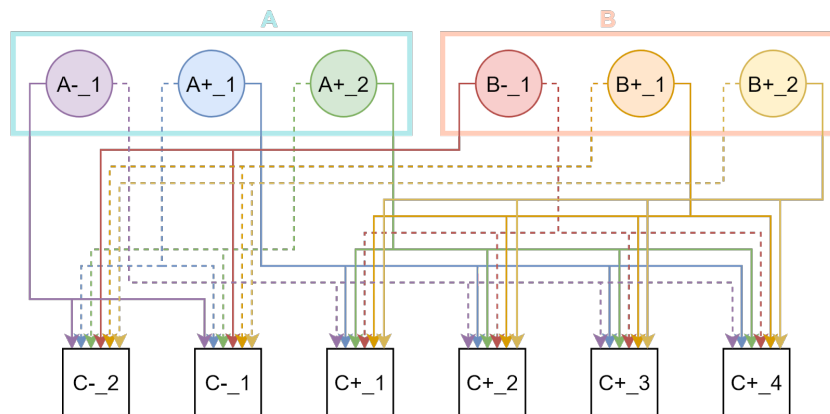


Figure 6: An example with $a^+ = b^+ = 2$ and $a^- = b^- = 1$.

How and why does this work?

Each neuron $C+_i$ spikes if its potential reaches its threshold i . At time 1, $C+_i$ receives:

1. excitatory spikes from all neurons $A+_i$ with $1 \leq i \leq A$ or inhibiting spikes from all neurons $A-_i$ with $1 \leq i \leq -A$ and
2. excitatory spikes from all neurons $B+_i$ with $1 \leq i \leq B$ or inhibiting spikes from all neurons $B-_i$ with $1 \leq i \leq -B$.

Thus, the potential of $C+_i$ becomes $A + B$ at time 1, making $C+_i$ spike if $A + B = C \geq i$.

Theorem 6.1. *For $1 \leq i \leq c^+$, neuron $C+_i$ spikes at time 1 if $i \leq C$*

Similarly for $C-_i$. Each neuron $C-_i$ spikes if its potential reaches its threshold i . At time 1, $C-_i$ receives:

1. inhibitory spikes from all neurons $A+_i$ with $1 \leq i \leq A$ or excitatory spikes from all neurons $A-_i$ with $1 \leq i \leq -A$ and
2. inhibitory spikes from all neurons $B+_i$ with $1 \leq i \leq B$ or excitatory spikes from all neurons $B-_i$ with $1 \leq i \leq -B$.

Thus, the potential of $C-_i$ becomes $-A - B$ at time 1, making $C-_i$ spike if $-A - B = -C \geq i$.

Theorem 6.2. *For $1 \leq i \leq c^-$, neuron $C-_i$ spikes at time 1 if $i \leq -C$*

Complexities

Space: $\Theta(a^+ + a^- + b^+ + b^-)$

Time: $O(1)$

Energy: $O(a^+ + a^- + b^+ + b^-)$

Benefits

- This network is reusable. This network can add two integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons $A+_i$, $A-_i$, $B+_i$ and $B-_i$ have fired, they can get new input to calculate the sum.
- This network can add negative integers.
- If indeed $a^- + b^- \leq c^-$ and $a^+ + b^+ \leq c^+$, then overflow is impossible.

6.2 Add two integers represented as the number of spiking neurons using a radix

The network in section 6.1 has a linear space and energy complexity. This makes this network inefficient for large integers. Just like with the time representation, let us try to solve this problem with radix $N \geq 2$.

In practice, it is impossible to have an infinite number of neurons. So we choose an $m \geq 0$ and define the input and output as $(m + 1)$ -digit integers.

Task

Given integers $A = A_0 + A_1 \cdot N + \dots + A_m \cdot N^m$ and $B = B_0 + B_1 \cdot N + \dots + B_m \cdot N^m$, calculate $C = (A+B) \bmod N^{m+1} = C_0 + C_1 \cdot 2 + \dots + C_m \cdot N^m$, where $0 \leq A_i, B_i, C_i < N$ for all digit-positions $0 \leq i \leq m$.

For $0 \leq i \leq m$, define function $f(i) = (N-1) \cdot 2 + 1_{\{i>0\}}$

Network $\mathcal{S} = (N, S)$

Neurons N

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $\mathbf{A}_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_i \geq j$

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $\mathbf{B}_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $B_i \geq j$

For $0 \leq i \leq m$, for $1 \leq j \leq f(i)$, neuron $\mathbf{S}_{i,j} = (j, 0, 0)$

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $\mathbf{C}_{i,j} = (j, 0, 0)$

Synapses S

For $0 \leq i \leq m$, for $1 \leq j < N$, for $i \leq k \leq m$, for $1 \leq p \leq f(k)$, synapse $\mathbf{A}_{i,j} \rightarrow \mathbf{S}_{k,p} = (1, N^{i-k})$

For $0 \leq i \leq m$, for $1 \leq j < N$, for $i \leq k \leq m$, for $1 \leq p \leq f(k)$, synapse $\mathbf{B}_{i,j} \rightarrow \mathbf{S}_{k,p} = (1, N^{i-k})$

For $0 \leq i \leq m$, for $1 \leq j \leq f(i)$, if $j \bmod N \neq 0$, synapse $\mathbf{S}_{i,j} \rightarrow \mathbf{C}_{i,(j \bmod N)} = (1, 1)$

For $0 \leq i \leq m$, for $1 \leq j \leq f(i)$, if $j \bmod N = 0$, for $1 \leq k < N$, synapse $\mathbf{S}_{i,j} \rightarrow \mathbf{C}_{i,k} = (1, -1)$

Input

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $\mathbf{A}_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_i \geq j$

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $\mathbf{B}_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $B_i \geq j$

Output

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $\mathbf{C}_{i,j}$ spikes at time 2 if $C_i \geq j$

How and why does this work?

Again, the idea is similar to the simple pen-and-paper method of addition.

Use carry_i from Definition A.1 with $N = N, M = 2, m = m, A_1 = A$ and $A_2 = B$. Define $D_i = A_i + B_i + (A_{i-1} + B_{i-1}) \cdot N^{-1} + \dots + (A_0 + B_0) \cdot N^{-i}$. Then from Lemma A.5 follows $C_i = \lfloor D_i \rfloor \bmod N$.

The network computes C in two steps:

1. Calculate D_i
2. Reduce D_i modulo 2

Calculate D_i

For each digit position $0 \leq i \leq m$, the sum D_i is calculated in neurons $\mathbf{S}_{i,j}$. From Lemma A.4 follows $0 \leq \text{carry}_i \leq \lfloor 2 \cdot (1 - N^{-i}) \rfloor$. Thus, for $i \geq 1$ holds $D_i \leq (N-1) \cdot 2 + 1$

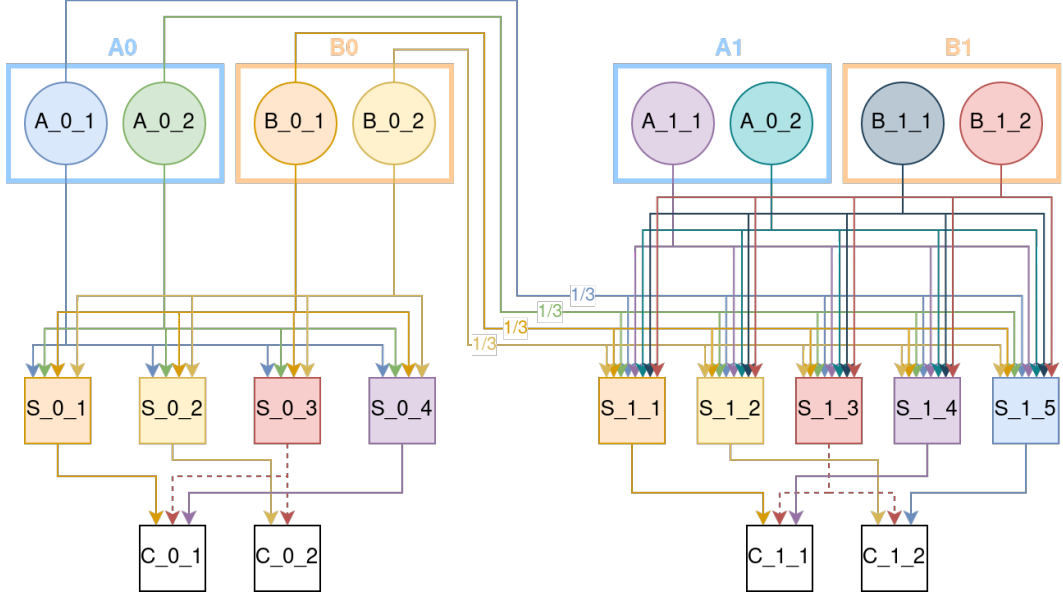


Figure 7: An example with $m = 1$ and $N = 3$.

and $A_0 + B_0 \leq (N - 1) \cdot 2$. That is why there are $f(i) = (N - 1) \cdot 2 + 1_{\{i > 0\}}$ neurons S_{i-j} for each digit position i .

At time 0, for $0 \leq i \leq m$, for $1 \leq j < N$, neuron A_{i-j} spikes if $A_i \geq j$ and neuron B_{i-j} spikes if $B_i \geq j$. Thus, at time 1, each neuron S_{i-j} gets:

1. For $0 \leq k \leq i$, for $1 \leq p < N$, excitatory spike of weight N^{k-i} from neuron A_{k-p} if $A_k \geq p$
2. For $0 \leq k \leq i$, for $1 \leq p < N$, excitatory spike of weight N^{k-i} from neuron B_{k-p} if $B_k \geq p$

So at time 1, the potential of neuron S_{i-j} becomes D_i . S_{i-j} spikes if this potential reaches the threshold j .

Lemma 6.3. $\forall 0 \leq i \leq m, \forall 1 \leq j \leq D_i$, neuron S_{i-j} spikes at time 1.

Reduce D_i modulo 2

D_i is reduced modulo 2 in neurons C_{i-j} . At time 2, neuron C_{i-j} receives:

1. excitatory signal from spiking neuron S_{i-j}
2. inhibiting signal from spiking neuron S_{i-N}
3. excitatory signal from spiking neuron $S_{i-(j+N)}$ if $j+N \leq f(i)$

Thus, at time 2, the potential of C_{i-j} becomes:

$$\begin{aligned}
 & 1_{\{S_{i-j} \text{ spiked at time 1}\}} - 1_{\{S_{i-N} \text{ spiked at time 1}\}} + 1_{\{j+N \leq f(i) \wedge S_{i-(j+N)} \text{ spiked at time 1}\}} \\
 &= 1_{\{D_i \geq j\}} - 1_{\{D_i \geq N\}} + 1_{\{j+N \leq f(i) \wedge D_i \geq j+N\}} \\
 &= 1_{\{N > D_i \geq j \vee (j+N \leq f(i) \wedge D_i \geq j+N)\}} \\
 &= 1_{\{C_i \geq j\}}
 \end{aligned}$$

The threshold of C_{i-j} is 1. So, at time 2, C_{i-j} spikes if $C_i \geq j$.

Theorem 6.4. For $0 \leq i \leq m$, for $1 \leq j < N$, neuron C_{i-j} spikes at time 2 if $C_i \geq j$

Complexities

Space: $\Theta(m \cdot N)$

Time: $O(1)$

Energy: $O(m \cdot N)$

Benefits

- This network is reusable. This network can add two integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons A_{i-j} and B_{i-j} have fired, they can get new input to calculate the sum.
- Using N 's complement makes subtraction possible. (See appendix B.)
- If the sum of the input is too big to fit in the neurons C_{i-j} , then this network still gives modulo the correct answer. This is expressed above as $C = (A + B) \bmod N^{m+1}$.

Drawbacks

- This network has a practical limitation. The weights of some synapses are defined as powers of $\frac{1}{N}$. The exponent can go up to infinity in theory, but every real system has a limited precision in practice. Thus, there is a physical maximum for m , or this network may give incorrect answers.

6.3 Add two integers represented by numerosity-selective neurons

The spiking neural network in this section adds two integers represented by numerosity neurons. Like in section 6.1 there are boundaries $a^-, a^+, b^-, b^+ \geq 0$.

Task

Given integers A and B such that $a^- \leq A \leq a^+$ and $b^- \leq B \leq b^+$, calculate $C = A + B$. Note: hence $c^- := a^- + b^- \leq C \leq c^+ := a^+ + b^+$.

Network $\mathcal{S} = (N, S)$

Neurons N

- For $1 \leq i \leq a^+$, neuron A_{+i} gets an excitatory spike of weight 1 at time 0 if $i = A$
- For $1 \leq i \leq a^-$, neuron A_{-i} gets an excitatory spike of weight 1 at time 0 if $i = -A$
- For $1 \leq i \leq b^+$, neuron B_{+i} gets an excitatory spike of weight 1 at time 0 if $i = B$
- For $1 \leq i \leq b^-$, neuron B_{-i} gets an excitatory spike of weight 1 at time 0 if $i = -B$
- For $1 \leq i \leq c^+$, neuron $D_{+i} = (i, 0, 0)$
- For $1 \leq i \leq c^-$, neuron $D_{-i} = (i, 0, 0)$
- For $1 \leq i \leq c^+$, neuron $C_{+i} = (1, 0, 0)$
- For $1 \leq i \leq c^-$, neuron $C_{-i} = (1, 0, 0)$

Synapses S

- For $1 \leq i \leq a^+$, for $1 \leq i \leq c^+$, synapse $A_{+i} \rightarrow D_{+i} = (1, i)$
- For $1 \leq i \leq a^+$, for $1 \leq i \leq c^-$, synapse $A_{+i} \rightarrow D_{-i} = (1, -i)$
- For $1 \leq i \leq a^-$, for $1 \leq i \leq c^+$, synapse $A_{-i} \rightarrow D_{+i} = (1, -i)$
- For $1 \leq i \leq a^-$, for $1 \leq i \leq c^-$, synapse $A_{-i} \rightarrow D_{-i} = (1, i)$

For $1 \leq i \leq b^+$, for $1 \leq i \leq c^+$, synapse $B+_i \rightarrow D+_i = (1, i)$
 For $1 \leq i \leq b^+$, for $1 \leq i \leq c^-$, synapse $B+_i \rightarrow D-_i = (1, -i)$
 For $1 \leq i \leq b^-$, for $1 \leq i \leq c^+$, synapse $B-_i \rightarrow D+_i = (1, -i)$
 For $1 \leq i \leq b^-$, for $1 \leq i \leq c^-$, synapse $B-_i \rightarrow D-_i = (1, i)$
 For $1 \leq i \leq c^-$, synapse $D-_i \rightarrow C-_i = (1, 1)$
 For $1 \leq i \leq c^+$, synapse $D+_i \rightarrow C+_i = (1, 1)$
 For $2 \leq i \leq c^-$, synapse $D-_i \rightarrow C_-(i-1) = (1, -1)$
 For $2 \leq i \leq c^+$, synapse $D+_i \rightarrow C_+(i-1) = (1, -1)$

Input

For $1 \leq i \leq a^+$, neuron $A+_i$ gets an excitatory spike of weight 1 at time 0 if $i = A$
 For $1 \leq i \leq a^-$, neuron $A-_i$ gets an excitatory spike of weight 1 at time 0 if $i = -A$
 For $1 \leq i \leq b^+$, neuron $B+_i$ gets an excitatory spike of weight 1 at time 0 if $i = B$
 For $1 \leq i \leq b^-$, neuron $B-_i$ gets an excitatory spike of weight 1 at time 0 if $i = -B$

Output

For $1 \leq i \leq c^+$, neuron $C+_i$ spikes at time 2 if $i = C$
 For $1 \leq i \leq c^-$, neuron $C-_i$ spikes at time 2 if $i = -C$

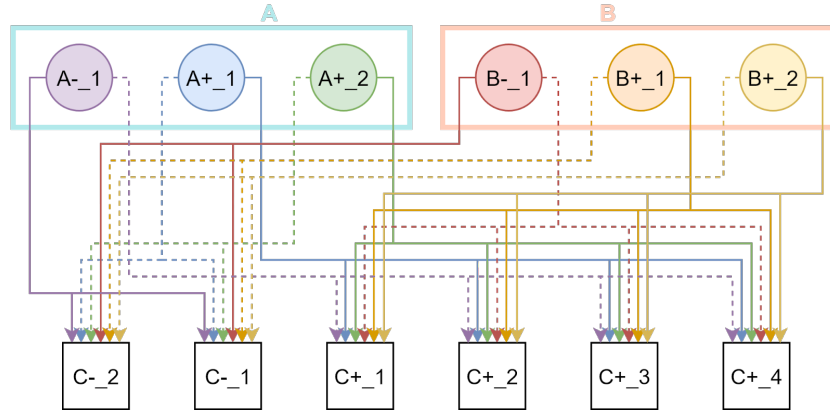


Figure 8: An example with $a^+ = b^+ = 2$ and $a^- = b^- = 1$.

How and why does this work?

At time 1, for $1 \leq i \leq c^+$, neuron $D+_i$ gets:

1. an excitatory spike of weight A from neuron $A+_A$ (if $A > 0$) or an inhibitory spike of same weight from neuron $A_-(-A)$ (if $A < 0$)
2. an excitatory spike of weight B from neuron $B+_B$ (if $B > 0$) or an inhibitory spike of same weight from neuron $B_-(-B)$ (if $B < 0$)

Thus, at time 1, the potential of neuron $D+_i$ becomes:

$$\begin{aligned}
 & A \cdot 1_{\{A_-(-A) \text{ spikes at time 0}\}} + A \cdot 1_{\{A+_A \text{ spikes at time 0}\}} \\
 & + B \cdot 1_{\{B_-(-B) \text{ spikes at time 0}\}} + B \cdot 1_{\{B+_B \text{ spikes at time 0}\}} \\
 & = A \cdot 1_{\{A < 0\}} + A \cdot 1_{\{A > 0\}} + B \cdot 1_{\{B < 0\}} + B \cdot 1_{\{B > 0\}} \\
 & = \max(0, A + B)
 \end{aligned}$$

Neuron $D+_i$ has threshold i . Thus neuron $D+_i$ spikes if $\max(0, A + B) \geq i$. This condition is equivalent to $A + B \geq i$.

Thus:

Lemma 6.5. *For $1 \leq i \leq c^+$, neuron $D+_i$ spikes at time 1 if $C \geq i$*

At time 1, for $1 \leq i \leq c^-$, neuron $D-_i$ gets:

1. an inhibitory spike of weight $-A$ from neuron $A+_A$ (if $A > 0$) or an excitatory spike of same weight from neuron $A_-(-A)$ (if $A < 0$)
2. an inhibitory spike of weight $-B$ from neuron $B+_B$ (if $B > 0$) or an inhibitory spike of same weight from neuron $B_-(-B)$ (if $B < 0$)

Thus, at time 1, the potential of neuron $D-_i$ becomes:

$$\begin{aligned} & -A \cdot 1_{\{A_-(-A) \text{ spikes at time 0}\}} - A \cdot 1_{\{A+_A \text{ spikes at time 0}\}} \\ & - B \cdot 1_{\{B_-(-B) \text{ spikes at time 0}\}} - B \cdot 1_{\{B+_B \text{ spikes at time 0}\}} \\ & = -A \cdot 1_{\{A < 0\}} - A \cdot 1_{\{A > 0\}} - B \cdot 1_{\{B < 0\}} - B \cdot 1_{\{B > 0\}} \\ & = \max(0, -A - B) \end{aligned}$$

Neuron $D-_i$ has threshold i . Thus neuron $D-_i$ spikes if $\max(0, -A - B) \geq i$. This condition is equivalent to $-A - B \geq i$.

Thus:

Lemma 6.6. *For $1 \leq i \leq c^-$, neuron $D-_i$ spikes at time 1 if $-C \geq i$*

At time 2, for $1 \leq i \leq c^+$, neuron $C+_i$ gets:

1. an excitatory signal from $D+_i$ if $D+_i$ has spiked at time 1
2. if $i < c^+$, an inhibitory signal from $D+_{i+1}$ if $D+_{i+1}$ has spiked at time 1

Thus, the potential of neuron $C+_i$ at time 2 becomes:

$$\begin{aligned} & 1_{\{D+_i \text{ has spiked at time 1}\}} - 1_{\{i < c^+ \wedge D+_{i+1} \text{ has spiked at time 1}\}} \\ & = 1_{\{A+B \geq i\}} - 1_{\{i < c^+ \wedge B+A \geq i+1\}} \\ & = 1_{\{A+B=i\}} \end{aligned}$$

The threshold of neuron $C+_i$ is 1. Thus, neuron $C+_i$ spikes at time 2 if $A + B = C = i$.

Theorem 6.7. $\forall 1 \leq i \leq c^+$ neuron $C+_i$ spikes at time 2 if $i = C$

At time 2, for $1 \leq i \leq c^-$, neuron $C-_i$ gets:

1. an excitatory signal from $D-_i$ if $D-_i$ has spiked at time 1
2. if $i < c^-$, an inhibitory signal from $D-_{i+1}$ if $D-_{i+1}$ has spiked at time 1

Thus, the potential of neuron $C-_i$ at time 2 becomes:

$$\begin{aligned} & 1_{\{D-_i \text{ has spiked at time 1}\}} - 1_{\{i < c^- \wedge D-_{i+1} \text{ has spiked at time 1}\}} \\ & = 1_{\{-A-B \geq i\}} - 1_{\{i < c^- \wedge -B-A \geq i+1\}} \\ & = 1_{\{-A-B=i\}} \end{aligned}$$

The threshold of neuron $C-_i$ is 1. Thus, neuron $C-_i$ spikes at time 2 if $-A - B = -C = i$.

Theorem 6.8. $\forall 1 \leq i \leq c^-$ neuron $C-_i$ spikes at time 2 if $i = -C$

Complexities

Space: $\Theta(a^+ + a^- + b^+ + b^-)$

Time: $O(1)$

Energy: $O(a^+ + a^- + b^+ + b^-)$

This network has the same benefits and drawbacks as the circuit from section 6.1.

Benefits

- This network is reusable. This network can add two integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons A^+_i , A^-_i , B^+_i and B^-_i have fired, they can get new input to calculate the sum.
- This network can add negative integers.
- If indeed $a^- + b^- \leq c^-$ and $a^+ + b^+ \leq c^+$, then overflow is impossible.

6.4 Add two integers represented by numerosity-selective neurons using a radix

The network in section 6.3 has also a linear space and energy complexity. This makes this network inefficient for large integers. Again, let us try to solve this problem with a radix $N \geq 2$.

In practice, it is impossible to have an infinite number of neurons. So we choose an $m \geq 0$ and define the input and output as $(m + 1)$ -digit integers.

Task

Given integers $A = A_0 + A_1 \cdot N + \dots + A_m \cdot N^m$ and $B = B_0 + B_1 \cdot N + \dots + B_m \cdot N^m$, calculate $C = (A + B) \bmod N^{m+1} = C_0 + C_1 \cdot 2 + \dots + C_m \cdot N^m$, where $0 \leq A_i, B_i, C_i < N$ for all digit-positions $0 \leq i \leq m$.

For $0 \leq i \leq m$, define function $f(i) = (N - 1) \cdot 2 + 1_{\{i>0\}}$

Network $\mathcal{S} = (N, S)$

Neurons N

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $A_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_i = j$

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $B_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $B_i = j$

For $0 \leq i \leq m$, for $1 \leq j \leq f(i)$, neuron $D_{i,j} = (j, 0, 0)$

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $C_{i,j} = (1, 0, 0)$

Synapses S

For $0 \leq i \leq m$, for $1 \leq j < N$, for $i \leq k \leq m$, for $1 \leq p \leq f(k)$, synapse $A_{i,j} \rightarrow D_{k,p} = (1, j \cdot N^{i-k})$

For $0 \leq i \leq m$, for $1 \leq j < N$, for $i \leq k \leq m$, for $1 \leq p \leq f(k)$, synapse $B_{i,j} \rightarrow D_{k,p} = (1, j \cdot N^{i-k})$

For $0 \leq i \leq m$, for $1 \leq j \leq f(i)$, if $1 \leq j \bmod N < N$, synapse $D_{i,j} \rightarrow C_{i,(j \bmod N)} = (1, 1)$

For $0 \leq i \leq m$, for $2 \leq j \leq f(i)$, if $1 \leq (j-1 \bmod N) < N$, synapse $D_{i,j} \rightarrow C_{i,j-1 \bmod N} = (1, 1)$

Input

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $A_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_i = j$

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $B_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $B_i = j$

Output

For $0 \leq i \leq m$, for $1 \leq j < N$, neuron $C_{i,j}$ spikes at time 2 if $C_i = j$

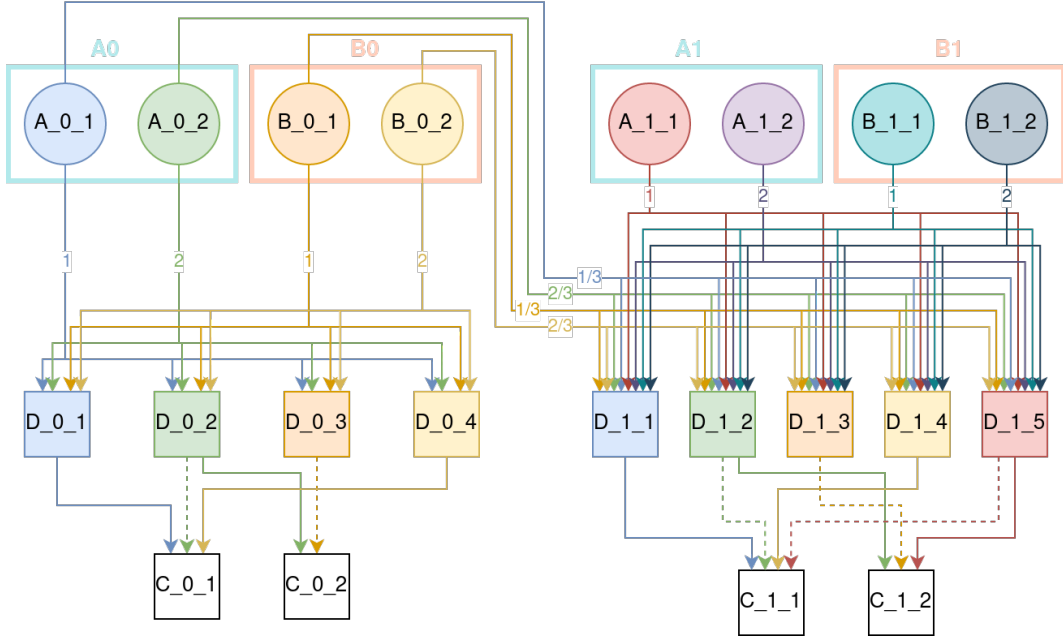


Figure 9: An example with $m = 1$ and $N = 3$.

How and why does this work?

Again, the idea is similar to the simple pen-and-paper method of addition.

Use carry_i from Definition A.1 with $N = N$, $M = 2$, $m = m$, $A_1 = A$ and $A_2 = B$. Define $D_i = A_i + B_i + (A_{i-1} + B_{i-1}) \cdot N^{-1} + \dots + (A_0 + B_0) \cdot N^{-i}$. Then from Lemma A.5 follows $C_i = \lfloor D_i \rfloor \bmod N$.

The network computes C in two steps:

1. Calculate D_i
2. Reduce D_i modulo 2

Calculate D_i

For each digit position $0 \leq i \leq m$, the sum D_i is calculated in neurons $D_{i,j}$. From Lemma A.4 follows $0 \leq \text{carry}_i \leq \lfloor 2 \cdot (1 - N^{-i}) \rfloor$. Thus, for $i \geq 1$ holds $D_i \leq (N-1) \cdot 2 + 1$ and $A_0 + B_0 \leq (N-1) \cdot 2$. That is why there are $f(i) = (N-1) \cdot 2 + 1_{\{i>0\}}$ neurons $D_{i,j}$ for each digit position i .

At time 0, for $0 \leq i \leq m$, for $1 \leq j < N$, neuron $A_{i,j}$ spikes if $A_i = j$ and neuron $B_{i,j}$ spikes if $B_i = j$. Thus, at time 1, each neuron $D_{i,j}$ gets:

1. For $0 \leq k \leq i$, an excitatory spike of weight $A_k \cdot N^{k-i}$ from neuron $A_{k-i}A_k$ if $A_k > 0$
2. For $0 \leq k \leq i$, an excitatory spike of weight $B_k \cdot N^{k-i}$ from neuron $B_{k-i}B_k$ if $B_k > 0$

So at time 1, the potential of neuron D_{i-j} becomes D_i . D_{i-j} spikes if this potential reaches the threshold j .

Lemma 6.9. $\forall 0 \leq i \leq m, \forall 1 \leq j \leq D_i$, neuron D_{i-j} spikes at time 1.

Reduce D_i modulo 2

D_i is reduced modulo 2 in neurons C_{i-j} . At time 2, neuron C_{i-j} receives:

1. an excitatory signal from firing neuron D_{i-j}
2. an inhibiting signal from firing neuron $D_{i-(j+1)}$
3. an excitatory signal from firing neuron $D_{i-(j+N)}$ if $j+N \leq f(i)$
4. an inhibitory signal from firing neuron $D_{i-(j+1+N)}$ if $j+1+N \leq f(i)$

Thus, at time 2, the potential of C_{i-j} becomes:

$$\begin{aligned}
& 1_{\{D_{i-j} \text{ spiked at time 1}\}} - 1_{\{D_{i-(j+1)} \text{ spiked at time 1}\}} + 1_{\{j+N \leq f(i) \wedge D_{i-(j+N)} \text{ spiked at time 1}\}} \\
& - 1_{\{j+1+N \leq f(i) \wedge D_{i-(j+1+N)} \text{ spiked at time 1}\}} \\
& = 1_{\{D_i \geq j\}} - 1_{\{D_i \geq j+1\}} + 1_{\{j+N \leq f(i) \wedge D_i \geq j+N\}} - 1_{\{j+1+N \leq f(i) \wedge D_i \geq j+1+N\}} \\
& = 1_{\{D_i = j \vee (j+N \leq f(i) \wedge D_i = j+N)\}} \\
& = 1_{\{C_i = j\}}
\end{aligned}$$

The threshold of C_{i-j} is 1. So, at time 2, C_{i-j} spikes if $C_i = j$.

Theorem 6.10. For $0 \leq i \leq m$, for $1 \leq j < N$, neuron C_{i-j} spikes at time 2 if $C_i = j$

Complexities

Space: $\Theta(m \cdot N)$

Time: $O(1)$

Energy: $O(m \cdot N)$

Benefits

- This network is reusable. This network can add two integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons A_{i-j} and B_{i-j} have fired, they can get new input to calculate the sum.
- Using N 's complement makes subtraction possible. (See appendix B.)
- If the sum of the input is too big to fit in the neurons C_{i-j} , then this network still gives modulo the correct answer. This is expressed above as $C = (A + B) \bmod N^{m+1}$.

Drawbacks

- This network a practical limitation. The weights of some synapses are defined as powers of $\frac{1}{N}$. The exponent can go up to infinity in theory, but every real system has a limited precision in practice. Thus, there is a physical maximum for m , or this network may give incorrect answers.

6.5 Add two integers represented by numerosity-selective neurons with a zero-neuron

The zero value has no explicit representation in the previous four networks: it is represented as the absence of spikes. However, zero is a number as well. So, in this section, the network has an explicit representation for zero.

Like in section 6.1 and section 6.3 there are boundaries $a^-, a^+, b^-, b^+ \geq 0$.

Task

Given integers A and B such that $-a^- \leq A \leq a^+$ and $-b^- \leq B \leq b^+$, calculate $C = A+B$. Note: hence $-c^- := -a^- - b^- \leq C \leq c^+ := a^+ + b^+$.

Network $\mathcal{S} = (N, S)$

Neurons N

- For $-a^- \leq i \leq a^+$, neuron A_i gets an excitatory spike of weight 1 at time 0 if $i = A$
- For $-b^- \leq i \leq b^+$, neuron B_i gets an excitatory spike of weight 1 at time 0 if $i = B$
- For $-a^- \leq i \leq a^+$, for $-b^- \leq j \leq b^+$, neuron $D_{i-j} = (2, 0, 0)$
- For $-c^- \leq i \leq c^+$, neuron $C_i = (2, 0, 0)$

Synapses S

- For $-a^- \leq i \leq a^+$, for $-b^- \leq j \leq b^+$, synapse $A_i \rightarrow D_{i-j} = (1, 1)$
- For $-b^- \leq i \leq b^+$, for $-a^- \leq j \leq a^+$, synapse $B_i \rightarrow D_{j-i} = (1, 1)$
- For $-a^- \leq i \leq a^+$, for $-b^- \leq j \leq b^+$, synapse $D_{i-j} \rightarrow C_{(i+j)} = (1, 1)$

Input

- For $-a^- \leq i \leq a^+$, neuron A_i gets an excitatory spike of weight 1 at time 0 if $i = A$
- For $-b^- \leq i \leq b^+$, neuron B_i gets an excitatory spike of weight 1 at time 0 if $i = B$

Output

For $-c^- \leq i \leq c^+$, neuron C_i spikes at time 2 if $i = C$

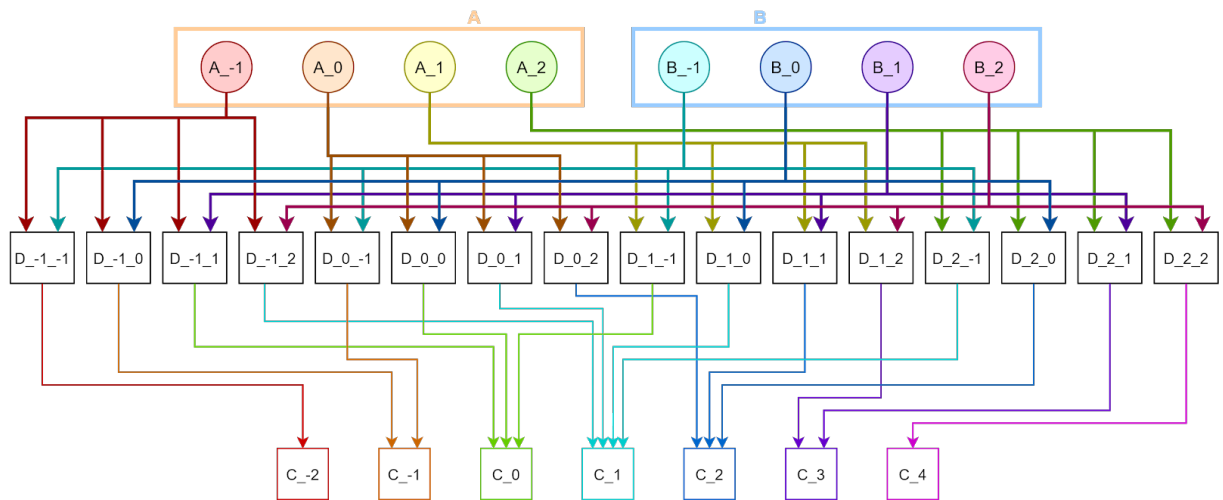


Figure 10: An example with $a^+ = b^+ = 2$ and $a^- = b^- = 1$.

How and why does this work?

The working principle of this network is simple and can be expressed in the following lemma and theorem.

Lemma 6.11. $\forall -a^- \leq i \leq a^+, \forall -b^- \leq j \leq b^+$, neuron $D_{i,j}$ spikes at time 1 if $i = A$ and $j = B$.

NB: thus, only neuron $D_{A,B}$ does spike at time 1.

From this lemma follows:

Theorem 6.12. $\forall -c^- \leq i \leq c^+$, neuron C_i spikes at time 2 if $i = A + B$.

NB: thus, only neuron $C_{(A+B)}$ spikes at time 2.

Complexities

Space: $\Theta((a^- + a^+) \cdot (b^- + b^+))$

Time: $O(1)$

Energy: $O(1)$

Benefits

- This network is reusable. This network can add two integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons A_i , B_i have fired, they can get new input to calculate the sum.
- This network can add negative integers.
- If indeed $a^- + b^- \leq c^-$ and $a^+ + b^+ \leq c^+$, then overflow is impossible.

6.6 Add two integers represented by numerosity-selective neurons with a zero-neuron using a radix

The network in section 6.5 has a quadratic space complexity. This makes this network inefficient for large integers. Again, let us try to solve this problem with a radix $N \geq 2$.

In practice, it is impossible to have an infinite number of neurons. So we choose an $m \geq 0$ and define the input and output as $(m+1)$ -digit integers.

Task

Given integers $A = A_0 + A_1 \cdot N + \dots + A_m \cdot N^m$ and $B = B_0 + B_1 \cdot N + \dots + B_m \cdot N^m$, calculate $C = (A+B) \bmod N^{m+1} = C_0 + C_1 \cdot 2 + \dots + C_m \cdot N^m$, where $0 \leq A_i, B_i, C_i < N$ for all digit-positions $0 \leq i \leq m$.

Network $\mathcal{S} = (N, S)$

Neurons N

For $0 \leq i \leq m$, for $0 \leq j < N$, neuron $A_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $A_i = j$

For $0 \leq i \leq m$, for $0 \leq j < N$, neuron $B_{i,j}$ gets an excitatory spike of weight 1 at time 0 if $B_i = j$

For $1 \leq i \leq m$, neuron $\text{carry}_i = (N, 0, 0)$

For $0 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$, neuron $D_{i,0,j,k} = (2, 0, 0)$

For $1 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$, neuron $D_{i-1-j-k} = (3, 0, 0)$

For $0 \leq i \leq m$, for $0 \leq j < N$, neuron $C_{i-j} = (1, 0, 0)$

Synapses S

For $0 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $A_{i-j} \rightarrow D_{i-0-j-k} = (2, 1)$

For $1 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $A_{i-j} \rightarrow D_{i-1-j-k} = (2, 1)$

For $0 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $B_{i-j} \rightarrow D_{i-0-k-j} = (2, 1)$

For $1 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $B_{i-j} \rightarrow D_{i-1-k-j} = (2, 1)$

For $0 \leq i < m$, for $0 \leq j < N$, for $i+1 \leq k \leq m$ synapse $A_{i-j} \rightarrow \text{carry}_k = (1, j \cdot N^{i+1-k})$

For $0 \leq i < m$, for $0 \leq j < N$, for $i+1 \leq k \leq m$ synapse $B_{i-j} \rightarrow \text{carry}_k = (1, j \cdot N^{i+1-k})$

For $1 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $\text{carry}_i \rightarrow D_{i-1-j-k} = (1, 1)$

For $1 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $\text{carry}_i \rightarrow D_{i-0-j-k} = (1, -1)$

For $0 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $D_{i-0-j-k} \rightarrow C_{i-(j+k \bmod N)} = (1, 1)$

For $1 \leq i \leq m$, for $0 \leq j < N$, for $0 \leq k < N$ synapse $D_{i-1-j-k} \rightarrow C_{i-(1+j+k \bmod N)} = (1, 1)$

Input

For $0 \leq i \leq m$, for $0 \leq j < N$, neuron A_{i-j} gets an excitatory spike of weight 1 at time 0 if $A_i = j$

For $0 \leq i \leq m$, for $0 \leq j < N$, neuron B_{i-j} gets an excitatory spike of weight 1 at time 0 if $B_i = j$

Output

For $0 \leq i \leq m$, for $0 \leq j < N$, neuron C_{i-j} spikes at time 3 if $C_i = j$

How and why does this work?

Again, the idea is similar to the simple pen-and-paper method of addition.

Use carry_i from Definition A.1 with $N = N, M = 2, m = m, A_1 = A$ and $A_2 = B$. Define $D_i = A_i + B_i + (A_{i-1} + B_{i-1}) \cdot N^{-1} + \dots + (A_0 + B_0) \cdot N^{-i}$. Then from Lemma A.5 follows $C_i = \lfloor D_i \rfloor \bmod N$.

The network computes C in two steps:

1. Calculate carry_i
2. Calculate D_i
3. Reduce D_i modulo 2

Calculate carry_i

For $i \geq 1$, the carry_i is calculated in neuron carry_i . From Lemma A.4 follows that $\text{carry}_i \leq \lfloor 2 \cdot (1 - N^{-i}) \rfloor = 1$. Thus, for each i , it is sufficient to have only one neuron carry_i , which spikes if $\text{carry}_i \geq 1$.

For each digit position $j > 0$, only neuron $A_{-j}A_j$ and neuron $B_{-j}B_j$ spike at time 0.

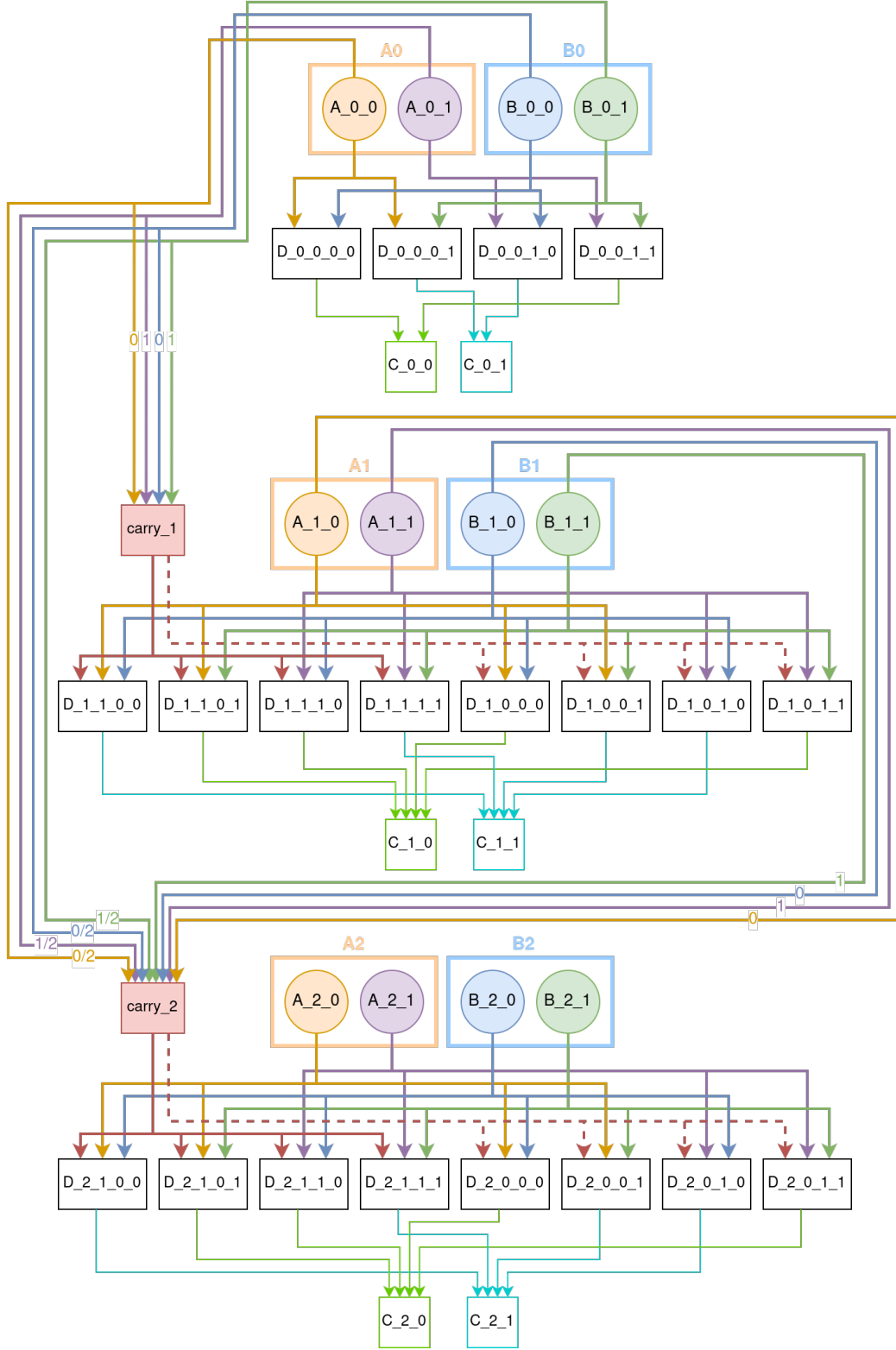


Figure 11: An example with $m = 2 = N$.

Hence, at time 1, the potential of neuron carry_i becomes

$$\sum_{j=0}^{i-1} (A_j + B_j) \cdot N^{j-i+1} = D_{i-1}$$

Neuron carry_i spikes if this potential reaches the threshold N . From Lemma A.3 follows that $\text{carry}_i = \lfloor D_i/N \rfloor$. Hence, neuron carry_i fires if $\text{carry}_i = 1 = \lfloor D_i/N \rfloor$.

Lemma 6.13. $\forall 1 \leq i \leq m$, neuron carry_i spikes at time 1 if $\text{carry}_i = 1$

Why is there no neuron `carry_0`? We know $\text{carry}_0 = 0$. Hence, neuron `carry_0` would never spike and is not necessary.

Calculate D_i

D_i is calculated in neurons $D_{i-j-k-p}$. Neurons $D_{i-0-k-p}$ are for $\text{carry}_i = 0$ and $D_{i-1-k-p}$ are for $\text{carry}_i = 1$.

Assume $i \geq 1$.

At time 2, each neuron $D_{i-j-k-p}$ receives:

1. an inhibiting signal from firing `carryi`
2. an excitatory signal from firing neuron A_{i-k}
3. an excitatory signal from firing neuron B_{i-p}

Thus, at time 2, the potential of neuron $D_{i-0-k-p}$ becomes:

$$1_{\{A_{i-k} \text{ spiked at time } 0\}} + 1_{\{B_{i-p} \text{ spiked at time } 0\}} - 1_{\{\text{carry}_i \text{ spiked at time } 1\}} = 1_{\{A_i=k\}} + 1_{\{B_i=p\}} - 1_{\{\text{carry}_i=1\}}$$

The threshold of $D_{i-0-k-p}$ is 2. Hence neuron $D_{i-0-k-p}$ fires at time 2 if $A_i = k$, $B_i = p$ and $\text{carry}_i = 0$.

At time 2, each neuron $D_{i-1-k-p}$ receives:

1. an excitatory signal from firing `carryi`
2. an excitatory signal from firing neuron A_{i-k}
3. an excitatory signal from firing neuron B_{i-p}

Thus, at time 2, the potential of neuron $D_{i-1-k-p}$ becomes:

$$1_{\{A_{i-k} \text{ spiked at time } 0\}} + 1_{\{B_{i-p} \text{ spiked at time } 0\}} + 1_{\{\text{carry}_i \text{ spiked at time } 1\}} = 1_{\{A_i=k\}} + 1_{\{B_i=p\}} + 1_{\{\text{carry}_i=1\}}$$

The threshold of $D_{i-1-k-p}$ is 3. Hence neuron $D_{i-1-k-p}$ fires at time 3 if $A_i = k$, $B_i = p$ and $\text{carry}_i = 1$.

Now, let us look at $i = 0$. We know $\text{carry}_0 = 0$. Thus, neurons $D_{0-1-k-p}$ are unnecessary and neurons $D_{0-0-k-p}$ gets signals only from neurons A_{i-k} and A_{i-p} (and no signals from non-existing neuron `carry0`).

At time 2, each neuron $D_{0-0-k-p}$ receives:

1. an excitatory signal from firing neuron A_{i-k}
2. an excitatory signal from firing neuron B_{i-p}

Thus, at time 2, the potential of neuron $D_{0-0-k-p}$ becomes:

$$1_{\{A_{0-k} \text{ spiked at time } 0\}} + 1_{\{B_{0-p} \text{ spiked at time } 0\}} = 1_{\{A_i=k\}} + 1_{\{B_i=p\}}$$

The threshold of $D_{0-0-k-p}$ is 2. Hence neuron $D_{0-0-k-p}$ fires at time 2 if $A_i = k$ and $B_i = p$.

Hence:

Lemma 6.14. $\forall 0 \leq i \leq m$ only neuron $D_{i-\text{carry}_i-A_i-B_i}$ spikes at time 2.

Reduce D_i modulo 2

D_i is reduced modulo 2 in neurons $C_{i,j}$. The threshold of each neuron $C_{i,j}$ is 1. Thus, from lemma 6.14, the structure of the network and $C_i = \text{carry}_i + A_i + B_i$ follows:

Theorem 6.15. $\forall 0 \leq i \leq m$ only neuron C_{i,C_i} spikes at time 3.

Complexities

Space: $\Theta(m \cdot N^2)$

Time: $O(1)$

Energy: $\Theta(m)$

Benefits

- This network is reusable. This network can add two integers again and again without changing its structure or resetting any parameters. It is even not necessary to wait until the current computation has finished. Immediately after neurons $A_{i,j}$ and $B_{i,j}$ have fired, they can get new input to calculate the sum.
- Using N 's complement makes subtraction possible. (See appendix B.)
- If the sum of the input is too big to fit in the neurons $C_{i,j}$, then this network still gives modulo the correct answer. This is expressed above as $C = (A + B) \bmod N^{m+1}$.

Drawbacks

- This network has a practical limitation. The weights of some synapses are defined as powers of $\frac{1}{N}$. The exponent can go up to infinity in theory, but every real system has a limited precision in practice. Thus, there is a physical maximum for m , or this network may give incorrect answers.

7 Compare networks

Eleven networks using spiking neurons have been developed to perform integer addition. See table 1 for an overview of all the networks. In this section the networks will be compared based on their benefits, practical limitations, drawbacks and complexities.

The networks do not all use the same number representations, with the consequence that their complexities are functions on different variables. This makes it difficult to compare the networks based on their complexities as expressed in table 1. So, let us try to express these resource constraints differently. Assume the task is to add two positive integers A and B . If radix N is applied, then A and B can be represented using $\log_N(A)$ and $\log_N(B)$ digits, respectively. Thus, the complexities can be expressed as in table 2.

Now, which network is the best?

7.1 Binary representation

Let us compare the networks with binary number representation.

The networks 1 and 2 have the same practical limitation, benefits and time complexity. If the sum of two integers is calculated, then the space and energy complexities are equal as well. Repeatedly applying network 1, it is possible to add more numbers. However, then the complexities increases: the network needs $\Theta(M)$ time-steps and $O(M \cdot m)$ spikes to

Table 1: Overview of all eleven networks.

	INPUT	BOUNDARIES	COMPLEXITIES			PRACTICAL LIMITATIONS	BENEFITS	DRAWBACKS
			space	time	energy			
1	Two integers A and B in $(m+1)$ -bit binary representation	$\mathbf{m} \geq \mathbf{0}$ $0 \leq A < 2^{m+1}$ $0 \leq B \leq 2^{m+1}$	$\Theta(m)$	$O(1)$	$O(m)$	Exponents of $\frac{1}{2}$ as weights of synapses	Using two's complement makes subtraction possible In case of overflow returns $(A + B) \bmod 2^{m+1}$ Directly reusable	
2	Multiple integers A_1, \dots, A_M in $(m+1)$ -bit binary representation	$\mathbf{m} \geq \mathbf{0}$ $\forall 1 \leq i \leq M : 0 \leq A_i \leq 2^{m+1}$	$\Theta(M \cdot m)$	$O(1)$	$O(M \cdot m)$	Exponents of $\frac{1}{2}$ as weights of synapses	Using two's complement makes subtraction possible In case of overflow returns $(A_1 + \dots + A_M) \bmod 2^{m+1}$ Directly reusable	
3	Two integers A and B represented as the time when the corresponding neurons spike	$\mathbf{N} \geq \mathbf{1}$ $N > \min(A, B)$ $A, B \geq 0$	$O(1)$	$\Theta(A + B)$	$\Theta(A + B)$	Some neurons should be able to keep a high potential	No overflow possible	Subtraction impossible. Some neurons keep spiking after calculation has finished Not reusable directly after the calculation has finished
4	Multiple integers A_1, \dots, A_M represented as the time when the corresponding neurons spike	$\mathbf{N} \geq \mathbf{1}$ $N > A_1 + \dots + A_M - \max(A_1, \dots, A_M)$ $\forall 1 \leq i \leq M : 0 \leq A_i$	$\Theta(M)$	$\Theta(A_1 + \dots + A_M)$	$\Theta(M \cdot (A_1 + \dots + A_M))$	Some neurons should be able to keep a high potential	No overflow possible	Subtraction impossible Some neurons keep spiking after calculation has finished Not reusable directly after the calculation has finished

5	Two $(m + 1)$ -digit integers A and B represented as the time when the corresponding neurons spike using radix N	$\mathbf{m} \geq \mathbf{0}$ $\mathbf{N} \geq \mathbf{2}$ $0 \leq A < N^{m+1}$ $0 \leq B < N^{m+1}$	$\Theta(m)$	$\Theta(N \cdot m)$	$\Theta(N \cdot m)$	<p>Some neurons should be able to keep a high potential</p> <p>Some synapses have long delays</p>	<p>Using N's complement makes subtraction possible</p> <p>In case of overflow returns $(A + B) \bmod 2^{m+1}$</p>	<p>Some neurons keep spiking after calculation has finished.</p> <p>Not reusable directly after the calculation has finished.</p>
6	Two integers A and B represented as the number of neurons that spike	$\mathbf{a}^-, \mathbf{a}^+, \mathbf{b}^-, \mathbf{b}^+ \geq \mathbf{0}$ $a^- \leq A \leq a^+$ $b^- \leq B \leq b^+$	$\Theta(a^- + a^+ + b^- + b^+)$	$O(1)$	$O(a^- + a^+ + b^- + b^+)$		<p>No overflow possible</p> <p>Subtraction is possible</p> <p>Directly reusable</p>	
7	Two $(m + 1)$ -digit integers A and B represented as the number of neurons that spike using radix N	$\mathbf{m} \geq \mathbf{0}$ $\mathbf{N} \geq \mathbf{2}$ $0 \leq A < N^{m+1}$ $0 \leq B < N^{m+1}$	$\Theta(N \cdot m)$	$O(1)$	$O(N \cdot m)$	Exponents of N^{-1} as weights of some synapses	<p>Using N's complement makes subtraction possible</p> <p>In case of overflow returns $(A + B) \bmod 2^{m+1}$</p> <p>Directly reusable</p>	
8	Two integers A and B represented by numerosity-selective neurons	$\mathbf{a}^-, \mathbf{a}^+, \mathbf{b}^-, \mathbf{b}^+ \geq \mathbf{0}$ $a^- \leq A \leq a^+$ $b^- \leq B \leq b^+$	$\Theta(a^- + a^+ + b^- + b^+)$	$O(1)$	$O(a^- + a^+ + b^- + b^+)$		<p>No overflow possible</p> <p>Subtraction is possible</p> <p>Directly reusable</p>	
9	Two $(m + 1)$ -digit integers A and B represented by numerosity-selective neurons using radix N	$\mathbf{m} \geq \mathbf{0}$ $\mathbf{N} \geq \mathbf{2}$ $0 \leq A < N^{m+1}$ $0 \leq B < N^{m+1}$	$\Theta(N \cdot m)$	$O(1)$	$O(N \cdot m)$	Exponents of N^{-1} as weights of some synapses	<p>Using N's complement makes subtraction possible</p> <p>In case of overflow returns $(A + B) \bmod 2^{m+1}$</p> <p>Directly reusable</p>	

10	Two integers A and B represented by numerosity-selective neurons with an explicit representation for the zero value	$\mathbf{a}^-, \mathbf{a}^+, \mathbf{b}^-, \mathbf{b}^+ \geq \mathbf{0}$ $a^- \leq A \leq a^+$ $b^- \leq B \leq b^+$	$\Theta((a^- + a^+) \cdot (b^- + b^+))$	$O(1)$	$O(1)$		No overflow possible Subtraction is possible Directly reusable	
11	Two $(m + 1)$ -digit integers A and B represented by numerosity-selective neurons with an explicit representation for the zero value using radix N	$\mathbf{m} \geq \mathbf{0}$ $\mathbf{N} \geq \mathbf{2}$ $0 \leq A < N^{m+1}$ $0 \leq B < N^{m+1}$	$\Theta(N^2 \cdot m)$	$O(1)$	$\Theta(m)$	Exponents of N^{-1} as weights of some synapses	Using N 's complement makes subtraction possible In case of overflow returns $(A + B) \bmod 2^{m+1}$ Directly reusable	

Table 2: Lowest possible complexities for given input $A, B \geq 0$ and radix $N \geq 2$

	COMPLEXITIES		
	space	time	energy
1	$\Theta(\log_2(A) + \log_2(B))$	$O(1)$	$O(\log_2(A) + \log_2(B))$
2	$\Theta(2 \cdot (\log_2(A) + \log_2(B))) = \Theta(\log_2(A) + \log_2(B))$	$O(1)$	$O(2 \cdot (\log_2(A) + \log_2(B))) = O(\log_2(A) + \log_2(B))$
3	$O(1)$	$\Theta(A + B)$	$\Theta(A + B)$
4	$O(1)$	$\Theta(A + B)$	$\Theta(2 \cdot (A + B)) = \Theta(A + B)$
5	$\Theta(\max(\log_N(A), \log_N(B))) = \Theta(\log_N(A) + \log_N(B))$ Minimal if N large.	$\Theta(N \cdot \max(\log_N(A), \log_N(B))) = \Theta(N \cdot (\log_N(A) + \log_N(B)))$ Minimal if $N = 3$. Then $\Theta(\log_3(A) + \log_3(B))$.	$\Theta(N \cdot \max(\log_N(A), \log_N(B))) = \Theta(N \cdot (\log_N(A) + \log_N(B)))$ Minimal if $N = 3$. Then $\Theta(\log_3(A) + \log_3(B))$.
6	$\Theta(A + B)$	$O(1)$	$O(A + B)$
7	$\Theta(N \cdot \max(\log_N(A), \log_N(B))) = \Theta(N \cdot (\log_N(A) + \log_N(B)))$ Minimal if $N = 3$. Then $\Theta(\log_3(A) + \log_3(B))$.	$O(1)$	$O(N \cdot \max(\log_N(A), \log_N(B))) = O(N \cdot (\log_N(A) + \log_N(B)))$ Minimal if $N = 3$. Then $O(\log_3(A) + \log_3(B))$.
8	$\Theta(A + B)$	$O(1)$	$O(A + B)$
9	$\Theta(N \cdot \max(\log_N(A), \log_N(B))) = \Theta(N \cdot (\log_N(A) + \log_N(B)))$ Minimal if $N = 3$. Then $\Theta(\log_3(A) + \log_3(B))$.	$O(1)$	$O(N \cdot \max(\log_N(A), \log_N(B))) = O(N \cdot (\log_N(A) + \log_N(B)))$ Minimal if $N = 3$. Then $O(\log_3(A) + \log_3(B))$.
10	$\Theta(A \cdot B)$	$O(1)$	$O(1)$
11	$\Theta(N^2 \cdot \max(\log_N(A), \log_N(B))) = \Theta(N^2 \cdot (\log_N(A) + \log_N(B)))$ Minimal if N small. $N \geq 2$. Thus minimal if $N = 2$. Then $\Theta(\log_2(A) + \log_2(B))$.	$O(1)$	$\Theta(\max(\log_N(A), \log_N(B))) = \Theta(\log_N(A) + \log_N(B))$ Minimal if N large.

add M numbers. It would still use $\Theta(m)$ neurons for this operation. Meanwhile, circuit 2 can add M numbers in constant time, but uses $\Theta(M \cdot m)$ neurons and $O(M \cdot m)$ spikes. Thus, network 2 can be M times faster than network 1. However, it would use also M times more neurons. Thus, network 1 versus network 2 resulted in a draw.

7.2 Time representation

Let us compare the networks where numbers are represented as the time when corresponding neurons spike.

Networks 3 and 4 have the same practical limitation, benefits and drawbacks. If the sum of two integers is calculated, then the space, time and energy complexities are equal as well. Network 3 can also be applied M times to add M numbers. (This would be harder to do than with network 1, but it is not impossible.) However, the time complexity becomes $\Theta(M \cdot A_1 + (M-1) \cdot A_2 + \dots + 1 \cdot A_M)$ then. This complexity is worse than $\Theta(A_1 + \dots + A_M)$ if network 4 would be applied. Thus, to add multiple numbers, network 4 is preferred over network 3, but if the task is to add only two integers, then these two networks are equivalent.

Network 5 adds two numbers and is difficult to compare to network 4. Network 5 has an important benefit over network 3: network 5 can add negative integers. Furthermore, the maximal potential of neuron `sum_i` in network 5 is less than the potential of neuron `C` in network 3, and the time and energy complexities of network 5 are significantly lower. On the other hand, network 5 has worse time complexity. However, the time complexity of neuron 3 is less times better than the energy and space complexities are worse. Thus, network 5 is preferred over network 3.

7.3 Neurons with predefined numerosities

Let us compare the networks that were inspired by the working of numerosity-selective neurons in the brain.

Networks 7 and 9 with radix number representation would be probably preferred over circuits 6, 8. The complexities of networks 7 and 9 are lower or equal to the complexities of 6 and 8. On the other hand, networks with radix have a practical limitation: the weights of some synapses are powers of $1/N$. Thus these networks have limited precision. There are $\Theta(\log_N(A))$ digits needed to represent an integer C using radix N . Thus, to add integers A and B , networks 7 and 9 should be able to work with weights as small as $\Theta(N^{-\log_N(A+B)}) = \Theta(1/(A+B))$. Meanwhile, networks 6 and 8 would need $\Theta(A+B)$ neurons and $O(A+B)$ spikes to perform the same calculation. However, it is probably more convenient to implement precision $\Theta(1/(A+B))$ than to use $\Theta(A+B)$ neurons and $O(A+B)$ spikes.

Network 11 with radix number representation would be probably preferred over network 10. These networks have the same time complexity and benefits. On the other hand, the network with radix representation has a practical limitation: the weights of some synapses are powers of $1/N$. Thus this networks has limited precision. There are $\Theta(\log_N(A))$ digits needed to represent an integer C using radix N . Thus, to add integers A and B , network 11 should be able to work with weights as small as $\Theta(N^{-\log_N(A+B)}) = \Theta(1/(A+B))$. Furthermore, network 11 has a higher energy complexity. (Constant number of spikes versus $\Theta(\max(\log_N(A), \log_N(B))) = \Theta(\log_N(A) + \log_N(B))$.) Meanwhile, network 10 would need $\Theta(A \cdot B)$ neurons and constant number of spikes to perform the same calculation. However, it is probably more convenient to implement precision $\Theta(1/(A+B))$ with logarithmic energy complexity than to use $\Theta(A \cdot B)$ neurons.

Networks 6 and 8 have the same complexities and benefits (and no practical limitations or drawbacks). The same holds for networks 7 and 9.

Networks 6 and 10 have the same benefits, practical limitations and time complexity, but they differ in space and energy complexity. Network 6 has space complexity $\Theta(a^- + a^+ + b^- + b^+)$ and energy complexity $O(a^- + a^+ + b^- + b^+)$. Meanwhile, network 10 has space complexity $\Theta((a^- + a^+)(b^- + b^+))$ and constant energy complexity. Which network is preferred depends on what is more important: to use as few neurons as possible or to have an energy-efficient algorithm. Thus, let us call this a tie. The same holds for networks 8 and 10.

Networks 7 and 11 have the same benefits, practical limitations and time complexity, but they differ in space and energy complexity. Network 7 has space complexity $\Theta(N \cdot m)$ and energy complexity $O(N \cdot m)$. Meanwhile, network 11 has space complexity $\Theta(N^2 \cdot m)$ and energy complexity $\Theta(m)$. Which network is preferred depends on what is more important: to use as few neurons as possible or to have an energy-efficient algorithm. Thus, let us call this a tie. The same holds for networks 9 and 11.

7.4 Binary representation versus time representation

Which representation is better: binary representation or time representation? Networks 1, 3 and 5 get two integers as input, and networks 2 and 4 can add multiple numbers. Furthermore, network 3 and 4 are equivalent if they both add only two numbers and network 5 is preferred over network 3. Thus, we compare 1 with 5 and 2 with 4.

Networks 1 and 5 have both practical limitations, which define boundaries for the input values. It is difficult to say which is harder to achieve on a real system: high precision or high membrane potentials and long synaptic delays. This depends strongly on how the parameters of an SNN are represented. In case of overflow, both networks return the sum reduced modulo the radix and both networks can add negative integers. However, the complexities of network 5 are worse, and network 5 is not reusable directly after the computation has finished. Thus, network 1 is preferred over network 5.

Networks 2 and 4 have both practical limitations, which define boundaries for the input values. It is difficult to say which is harder to achieve on a real system: high precision or high membrane potentials. This depends strongly on how the parameters of an SNN are represented. In case of overflow, network 2 returns the sum reduced modulo 2. Meanwhile, overflow is impossible in network 4. However, network 4 can not subtract and is not reusable directly after the computation has finished. The time complexity of network 4 is $\Theta(m)$ times lower. But the energy and time complexities of network 4 are relatively much worse than those of network 2. Thus, network 2 is preferred over network 4.

7.5 Time representation versus numerosity-selective neurons

Which representation is better: time representation or numerosity-selective neurons? Networks 3, 5, 6, 7, 8, 9, 10 and 11 add two integers, while network 4 can get multiple numbers as input. Network 3 and 4 are equivalent if they both add only two numbers and network 5 is preferred over network 3. Furthermore, networks 7, 9 and 11 would be probably preferred over networks 6, 8 and 10. Thus, we compare network 5 with 7, 9 and 11.

Networks 5 and 7 have both practical limitations, which define boundaries for the input values. It is difficult to say which is harder to achieve on a real system: high precision or high membrane potentials and long synaptic delays. In case of overflow, both networks return the sum reduced modulo the radix and both networks can add negative integers.

However, network 7 is reusable directly after the computation has finished. The two networks have the same energy complexity. The space complexity of network 5 is $\Theta(N)$ times lower. However, the time complexity of network 5 is $\Theta(N \cdot m)$ times higher than that of network 7. Thus, network 7 is preferred over network 5. The same holds for networks 5 and 9: network 9 is preferred over network 5.

Networks 5 and 11 have both practical limitations, which define boundaries for the input values. It is difficult to say which is harder to achieve on a real system: high precision or high membrane potentials and long synaptic delays. In case of overflow, both networks return the sum reduced modulo the radix and both networks can add negative integers. However, network 11 is reusable directly after the computation has finished. The time complexity of circuit 11 is $\Theta(N^2)$ times higher. Meanwhile, the time complexity of network 11 is $\Theta(N \cdot m)$ times lower and the energy complexity $\Theta(N)$ times lower. Thus, network 11 is preferred over network 5.

7.6 Binary representation versus numerosity-selective neurons

Which representation is better: binary representation or numerosity-selective neurons? Networks 1, 6, 7, 8, 9, 10 and 11 add two integers, while circuit 2 can get multiple numbers as input. However, network 1 and 2 are equivalent if they get both only two input numbers. Furthermore, algorithms 7, 9 and 11 would be probably preferred over networks 6, 8 and 10. Thus, we compare network 1 with 7, 9 and 11.

Networks 1 and 7 have the same practical limitation, time complexity and benefits (and no drawbacks). Moreover, at best, they use an equivalent number of neurons and spike ($\Theta(\log_3(A) + \log_3(B)) = \Theta(\log_2(A) + \log_2(B))$). Thus network 1 and network 7 are equivalent. The same for 1 and 9.

Networks 1 and 11 have the same practical limitation, time complexity and benefits (and no drawbacks). At best, they use an equivalent number of neurons: $\Theta(\log_2(A) + \log_2(B))$ if radix $N = 2$. The energy complexity is then the same as well. Network 11 may have a lower energy complexity for higher N . However, increasing N , the number of neurons that circuit 11 needs grows quadratically. Thus, for network 11, radix $N = 2$ would probably be preferred; in this case, circuits 1 and 11 are equivalent.

7.7 Verdict

Thus, in summary, among these eleven networks proposed in this paper, networks 1, 2, 7, 9 and 11 are probably the best to compute the sum of two integers. To add $M > 2$ integers, these networks can be applied $\Theta(M)$ times, or network 2 can perform the same calculations, using $\Theta(M)$ times fewer time-steps but $\Theta(M)$ times more neurons.

8 Discussion and future work

In section 7, it was decided that the circuits with binary representation (section 4.1 and 4.2) and circuits with numerosity-selective neurons using radix representation (section 6.2, 6.4 and 6.6) are the best to perform integer addition. These circuits are the most efficient, have no drawbacks and their practical limitations are not significant, compared to the complexities and drawbacks of other networks.

The networks proposed in this paper were developed using the deterministic discrete-time abstract model as described by J. Kwisthout and N. Donselaar [9]. In this model, a neuron has a deterministic spiking behaviour: it spikes when its membrane potential

reaches its threshold. As a subject of further investigation, it could be advantageous to use stochastic neurons for integer addition. Such neurons spike according to some probabilistic distribution. Typically, the closer their potential is to the threshold, the higher the chance that these neurons would spike.

For example, stochastic neurons would be favourable for numerosity-selective neurons simulation. As was mentioned in section 6, a numerosity-selective neuron reacts to all numbers that are close enough to its preferred numerosity. However, the closer that number is to the preferred numerosity, the higher the frequency with which the numerosity-selective neuron spikes. This behaviour would be probably easy to mimic with a stochastic neuron.

In section 7 the networks were compared based on their benefits, drawbacks, practical limitations and complexities. The energy complexity is defined as the number of spikes that a network needs to return some expected output. However, this may be an oversimplification of reality, because in practice, keeping the potential of a neuron at some non-zero value does also cost some of energy. Especially for networks 5.1, 5.2, 5.3, it may be interesting to use another model for energy complexities (for example [13]) and to investigate the effects of keeping a high potential below the threshold.

In section 3, we have defined three constraints that the networks must respect. One of those constraints was: the networks should be non-hybrid. As a consequence, this paper did not consider the time and space that an external system would take to create the proposed networks. However, in future work, it may be interesting to investigate these resource constraints.

For example, the aim is to compare two spiking networks. Assume, both have a constant time complexity. However, one network uses a quadratic number of neurons and a constant number of spikes, and the other network needs a linear number of neurons and a linear number of spikes. Using only the resource constraints R_S , as defined in section 3, does not help to decide which network is the most efficient. However, creating the first network would take at least a quadratic time and the second at least a linear time. Thus in total, the second network may turn out to be more efficient than the first one.

9 Conclusion

Eleven spiking algorithms were proposed to add integers. These networks were compared to each other, and it turned out that the networks with radix number representation (including binary representation) are the best to perform this basic arithmetic operation. These networks have at best logarithmic space and energy complexities and a constant time complexity. Furthermore, the networks can be used multiple times without changing any parameters of the components, they can handle integer overflow and, using the radix complements, these networks can subtract integers. However, all five algorithms have one drawback: due to limited precision, the number of digits is bounded, or the networks may return incorrect output.

A Appendix

Assume $N \geq 2$ and $m \geq 0$ and $M \geq 1$. Given $A_i = A_{i,0} + A_{i,1} \cdot N + \dots + A_{i,m} \cdot N^m$ (for $1 \leq i \leq M$) and $C = C_0 + C_1 \cdot N + \dots + C_m \cdot N^m = (A_0 + \dots + A_M) \bmod N^{m+1}$ such that $0 \leq A_{i,j}, C_j < N$ for all i and all j .

Definition A.1. Define carry_i for $0 \leq i \leq m$ as follows:

$$\text{carry}_0 = 0$$

$$\text{For } m \geq i \geq 1, \text{carry}_i = \max\{k \in \mathbb{N} \mid A_{1,i-1} + A_{2,i-1} + \cdots + A_{M,i-1} + \text{carry}_{i-1} \geq k \cdot N\}$$

Lemma A.2. For $1 \leq i \leq m$:

$$\text{carry}_i = \left\lfloor \frac{A_{1,i-1} + A_{2,i-1} + \cdots + A_{M,i-1} + \text{carry}_{i-1}}{N} \right\rfloor$$

Proof.

$$\begin{aligned} \text{carry}_0 &= 0 \\ &= \left\lfloor \frac{0}{N} \right\rfloor \end{aligned}$$

Assume $m \geq i > 0$. From definition of carry_i follows:

$$\begin{aligned} (\text{carry}_i + 1) \cdot N &> A_{1,i-1} + A_{2,i-1} + \cdots + A_{M,i-1} + \text{carry}_{i-1} \geq \text{carry}_i \cdot N \\ \text{carry}_i + 1 &> \frac{A_{1,i-1} + A_{2,i-1} + \cdots + A_{M,i-1} + \text{carry}_{i-1}}{N} \geq \text{carry}_i \end{aligned}$$

From the definition of the floor function follows $\text{carry}_i = \left\lfloor \frac{A_{1,i-1} + A_{2,i-1} + \cdots + A_{M,i-1} + \text{carry}_{i-1}}{N} \right\rfloor$ ([8] Chapter 3)

□

Lemma A.3. For $1 \leq i \leq m$ holds:

$$\text{carry}_i = \lfloor (A_{1,i-1} + A_{2,i-1} + \cdots + A_{M,i-1}) \cdot N^{-1} + \cdots + (A_{1,0} + A_{2,0} + \cdots + A_{M,0}) \cdot N^{-i} \rfloor$$

Proof. Proof by induction on i .

$$\begin{aligned} \text{carry}_1 &= \left\lfloor \frac{A_{1,0} + A_{2,0} + \cdots + A_{M,0} + \text{carry}_0}{N} \right\rfloor \\ &= \lfloor (A_{1,0} + A_{2,0} + \cdots + A_{M,0} + 0) \cdot N^{-1} \rfloor \end{aligned}$$

Assume $2 \leq i \leq m$. Assume:

$$\text{carry}_{i-1} = \lfloor (A_{1,i-2} + A_{2,i-2} + \cdots + A_{M,i-2}) \cdot N^{-1} + \cdots + (A_{1,0} + A_{2,0} + \cdots + A_{M,0}) \cdot N^{-i} \rfloor$$

Then:

$$\begin{aligned}
\text{carry}_i &= \left\lfloor \frac{A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1} + \text{carry}_{i-1}}{N} \right\rfloor \\
&= \left\lfloor \frac{A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1} + \lfloor (A_{1,i-2} + A_{2,i-2} + \dots + A_{M,i-2}) \cdot N^{-1} + \dots + (A_{1,0} + A_{2,0} + \dots + A_{M,0}) \cdot N^{-i} \rfloor}{N} \right\rfloor \\
&= \left\lfloor \frac{\left\lfloor \frac{A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1}}{N} + (A_{1,i-2} + A_{2,i-2} + \dots + A_{M,i-2}) \cdot N^{-1} + \dots + (A_{1,0} + A_{2,0} + \dots + A_{M,0}) \cdot N^{-i} \right\rfloor}{N} \right\rfloor \\
&= \left\lfloor \frac{A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1} + (A_{1,i-2} + A_{2,i-2} + \dots + A_{M,i-2}) \cdot N^{-1} + \dots + (A_{1,0} + A_{2,0} + \dots + A_{M,0}) \cdot N^{-i}}{N} \right\rfloor \\
&= \lfloor (A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1}) \cdot N^{-1} + \dots + (A_{1,0} + A_{2,0} + \dots + A_{M,0}) \cdot N^{-i} \rfloor
\end{aligned}$$

□

The next lemma follows from Lemma A.3.

Lemma A.4. For $0 \leq i \leq m$ holds $0 \leq \text{carry}_i \leq \lfloor M \cdot (1 - N^{-i}) \rfloor < M$.

Proof.

$$\begin{aligned}
\text{carry}_i &\leq \lfloor (N-1) \cdot M \cdot N^{-1} + \dots + (N-1) \cdot M \cdot N^{-i} \rfloor \\
&= \lfloor (N-1) \cdot M \cdot (N^{i-1} + \dots + 1) \cdot N^{-i} \rfloor \\
&= \lfloor M \cdot (1 - N^{-i}) \rfloor \\
&< M
\end{aligned}$$

Furthermore, all $A_{i,j}$ are positive. Thus $\text{carry}_i \geq 0$.

□

Now, assume A_i are given and we want to calculate C . Remember how to add numbers? For each digit position, starting from the least significant, add the corresponding bits and the carry from the previous position (if any). Thus, for $0 \leq i \leq m$ holds $C_i = (A_{1,i} + \dots + A_{M,i} + \text{carry}_i) \bmod N$. The next lemma follows directly from Lemma A.3.

Lemma A.5. For $0 \leq i \leq m$ holds:

$$\begin{aligned}
C_i &= (A_{1,i} + \dots + A_{M,i} + \lfloor (A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1}) \cdot N^{-1} + \dots \\
&\quad + (A_{1,0} + A_{2,0} + \dots + A_{M,0}) \cdot N^{-i} \rfloor) \bmod N \\
&= \lfloor A_{1,i} + \dots + A_{M,i} + (A_{1,i-1} + A_{2,i-1} + \dots + A_{M,i-1}) \cdot N^{-1} + \dots \\
&\quad + (A_{1,0} + A_{2,0} + \dots + A_{M,0}) \cdot N^{-i} \rfloor \bmod N
\end{aligned}$$

B Appendix

This paper mentions multiple times the N 's complement. An N 's complement allows us to represent negative numbers as positive and, thus, perform subtraction as addition. The two famous special cases are the two's complement (e.g. binary calculation) and nine's complement (e.g. Pascaline).

Definition B.1. The N 's complement of a m -digit integer A represented using radix N is:

$$N^m - A - 1$$

We denote the complement of A as $C(A)$.

Lemma B.2. $C(C(A)) = A$

Proof.

$$\begin{aligned} C(C(A)) &= (N^m - (N^m - A - 1)N^m - 1) \\ &= (N^m - N^m - 1 + 1 + A)N \\ &= A \end{aligned}$$

□

Now, assume we want to find the difference of two m -digit integers A and B represented using radix N .


Lemma B.3. $A - B = C(C(A) + B)$

Proof.

$$\begin{aligned} A - B &= N^m + A - B - 1 - N^m + 1 \\ &= C(B - A + N^m - 1) \\ &= C(C(A) + B) \end{aligned}$$

□

C Simulations

Some network were simulated in python. This section contains the code of these simulations. The classes for the network, neurons and synapses, which are used in the following simulations, are based on the code from  <https://gitlab.socsci.ru.nl/j.kwisthout/neuromorphic-genetic-algorithm/-/tree/master/pySimulator>.

C.1 Neurons

The neurons are programmed as follows:

class SimpleNeuron():

```
"""
    V_init = initial voltage
    V_reset = reset voltage
    m = leakage constant
    thr = threshold
    I = current potential
    """
    def __init__(self, m=0., V_init=0., V_reset=0., thr=1., name
    = ''):
        self.name = name
        self.V = V_init
```

```

self.V_reset = V_reset
self.thr = thr
self.m = m
self.I = 0

def step(self):
    # print(self.name)
    if self.V >= self.thr:
        self.V = self.V_reset
        # print("\t1-", self.V)
    else:
        self.V = self.V*self.m + self.I
        self.V = max(0, self.V)
        # print("\t2-", self.V)
    self.I = 0

```

Some networks use neurons that spike at a particular time. These are simulated as follows:

use only for input neurons that don't receive spikes

class DelayedFirstSpike(SimpleNeuron):

```

def __init__(self, m=0., V_reset=0., thr=1., name='', delay
=0):
    self.delay = delay
    if delay == 0:
        SimpleNeuron.__init__(self, m, V_init=thr, V_reset=
            V_reset, thr=thr, name=name)
    else:
        SimpleNeuron.__init__(self, m, V_init=0., V_reset=
            V_reset, thr=thr, name=name)

def step(self):
    if self.delay >= 0:
        self.delay -= 1
        if self.delay == 0:
            self.V = self.thr
    elif self.delay < 0:
        super(DelayedFirstSpike, self).step()

```

C.2 Synapses

The synapses are simulated as follows:

import numpy as np

class Synapse():
"""

Connection between two neurons

Parameters

```

pre : Neuron
    Presynaptic neuron
post : Neuron
    Postsynaptic neuron
w : float
    Connection weight
d : int
    Synaptic delay (number of timesteps)
"""
def __init__(self, pre, post, w, d):
    if(d<1):
        raise ValueError("Synaptic_delay_must_be
                           _at_least_1")
    self.pre = pre
    self.post = post
    self.w = w
    self.out_pre = np.zeros((d)) # store output of
        the presynaptic neuron during d timesteps
    self.index = 0

def step(self):
    if self.pre.V >= self.pre.thr:
        self.out_pre[self.index] = 1 # store
            current output of pre
    else:
        self.out_pre[self.index] = 0
    self.index = (self.index + 1) % len(self.out_pre
    )
    self.post.I += self.w * self.out_pre[self.index]
        # add w*pre_{t-d} to post

```

C.3 Network

Neurons and synapses create a spiking neural network:

```
import numpy as np
```

```
class Network():
```

```
    """Network containing a list of nodes and synapses
```

```
        Parameters
```

```
        nodes : list
```

```
            List of nodes in the network
```

```
        synapses : list
```

```
            List of synapses connecting nodes
```

```
    """
```

```

def __init__(self, nodes, synapses):
    self.nodes = nodes
    self.synapses = synapses

```

```

def step(self):
    for synapse in self.synapses: # update all
        synapses
        synapse.step()
    for node in self.nodes: # update all nodes
        node.step()

```

C.4 Add two integers in binary representation

The code below simulates the circuit from section 4.1. This example adds 5 and 6 reduced modulo 4.

```

import numpy as np

from pySimulator.nodes import SimpleNeuron
from pySimulator.connections import Synapse
from pySimulator.networks import Network

def create_input_neurons(input, thr=1., name=''):
    A= []
    n=0
    for i in input:
        A.append(SimpleNeuron(m=0, V_init=i, thr=thr, name=name+
            str(n)))
        n+=1

    return A

def create_output_neurons(size, thr=1., name=''):
    A= []
    for i in range(size):
        A.append(SimpleNeuron(m=0, V_init=0, thr=thr, name=name+
            str(i)))

    return A

def create_synapses_to_out(input, out, connections, d):
    for i in range(len(input)):
        # print(str(i))
        for j in range(i, len(out)):
            # print("\t"+str(j))
            connections.append(Synapse(input[i], out[j], w=(0.5
                ** (j-i)), d=d))

def create_synapses_intern(O, C, connections):
    for i in range(len(O)):
        connections.append(Synapse(O[i], C[i], w=-2, d=1))

```

```

#Input: m-bit array A that contains input number A
#       n-bit array B that contains input number B
#       k-bit array C filled with 0
#Output: C = A+B mod 2^(k+1) as k-bit array.
#NB: least significant bit first
def SUM_bitarray(inputA, inputB, output):
    A = create_input_neurons(inputA, name='a')
    B = create_input_neurons(inputB, name='b')
    O = create_output_neurons(len(output), 2, name='o')
    C = create_output_neurons(len(output), name='c')

    print("NEURONS")
    neurons = []
    neurons.extend(A)
    neurons.extend(B)
    neurons.extend(O)
    neurons.extend(C)

    for i in neurons:
        print(i)

    connections = []
    print("\nSYNAPSES")
    create_synapses_to_out(A, O, connections, 1)
    create_synapses_to_out(B, O, connections, 1)
    create_synapses_to_out(A, C, connections, 2)
    create_synapses_to_out(B, C, connections, 2)
    create_synapses_intern(O, C, connections)

    for i in connections:
        print(i)

    SUM = Network(neurons, connections)

    SUM.step()
    SUM.step()
    output = [(1 if n.V >= n.thr else 0) for n in C]

    return output

print(SUM_bitarray([1,0,1,0,0],[0,1,1],[0,0]))

```

C.5 Add multiple integers in binary representation

The code below simulates the circuit from section 4.1. This example adds 29, 5 and 4 reduced modulo 4.

```

import numpy as np

from pySimulator.nodes import SimpleNeuron
from pySimulator.connections import Synapse

```

```

from pySimulator.networks import Network

def create_overflow_neurons(output_size, input, thr=1., name='')
:
    O = []
    for i in range(len(input)*2):
        O.append([])
        for j in range(output_size):
            O[i].append(SimpleNeuron(m=0, V_init=0, thr=i+1,
                                   name=name+str(i)+"_"+str(j)))
    return O

def create_synapses_to_out(input, out, connections, d):
    for i in range(len(input)):
        # print(str(i))
        for j in range(i, len(out)):
            # print("\t"+str(j))
            connections.append(Synapse(input[i], out[j], w=(0.5
                                   ** (j-i)), d=d))

def create_synapses_intern(O, C, connections):
    for i in range(len(O)):
        for j in range(len(O[i])):
            connections.append(Synapse(O[i][j], C[j], w=((-1)**i
                                   ), d=1))

def SUM_N_bitarray(input, output):
    I = []
    for i in range(len(input)):
        I.append(create_input_neurons(input[i], name=str(i)+"_"))

    O = create_overflow_neurons(len(output), input, 2, name='o')

    C = create_output_neurons(len(output), name='c')

    print("NEURONS")
    neurons = []

    for i in I:
        for j in i:
            neurons.append(j)

    for i in O:
        for j in i:
            neurons.append(j)

    neurons.extend(C)

    for i in neurons:
        print(i)

```

```

connections = []
print(" \nSYNAPSES")
for j in range(len(I)):
    for i in range(len(O)):
        create_synapses_to_out(I[j], O[i], connections, 1)

create_synapses_intern(O, C, connections)

for i in connections:
    print(i)

SUM_N = Network(neurons, connections)

SUM_N.step()
SUM_N.step()
output = [(1 if n.V >= n.thr else 0) for n in C]

return output

print(SUM_N.bitarray([[1,0,1,1,1],[1,0,1],[0,0,1]],[0,0]))

```

C.6 Add two integers represented as the time when a neuron spikes

The code below simulates the circuit from section 5.1. This example adds 0 and 1 using $N = 4$.

```

import numpy as np

from pySimulator.nodes import SimpleNeuron, DelayedFirstSpike
from pySimulator.connections import Synapse
from pySimulator.networks import Network

N=4

def SUM_int(inputA, inputB):
    A = DelayedFirstSpike(0.,0.,1., "A", inputA)
    B = DelayedFirstSpike(0.,0.,1., "B", inputB)

    inf = SimpleNeuron(0.,1.,1.,1., "inf")
    D = SimpleNeuron(1.,0.,0.,2., "D")

    a = SimpleNeuron(0.,0.,1.,1., "a")
    b = SimpleNeuron(0.,0.,1.,1., "b")

    C = SimpleNeuron(1., N-1, 0, N, "C")

    neurons = [A,B,inf,D,a,b,C]

    synapses = [

```



```

        Synapse(A, D, 1., 1),
        Synapse(B, D, 1., 1),
        Synapse(A, a, 1., 1),
        Synapse(B, b, 1., 1),
        Synapse(D, C, 1., 1),
        Synapse(a, a, 1., 1),
        Synapse(b, b, 1., 1),
        Synapse(a, C, 1., 2),
        Synapse(b, C, 1., 2),
        Synapse(inf, C, -1., 3)]

SUM = Network(neurons, synapses)

count = 0

while C.V + C.I < N:
    SUM.step()
    count+=1

return count

SUM.int(1,0)

```

C.7 Add multiple integers represented as the time when a neuron spikes

The code below simulates the circuit from section 5.2. This example adds 0, 0, 1, 1 and 1 using $N = 10$.

```

import numpy as np

from pySimulator.nodes import SimpleNeuron, DelayedFirstSpike
from pySimulator.connections import Synapse
from pySimulator.networks import Network

N=10

#input = list of integers
def SUM_int(INPUT):
    A = [DelayedFirstSpike(0.,0.,1., "A"+str(i), INPUT[i])
          for i in range(len(INPUT))]
    inf = SimpleNeuron(0.,1,1.,1, "inf")
    D = SimpleNeuron(1.,0,0., float(len(INPUT)), "D")
    a = [SimpleNeuron(0.,0,1.,1, "a"+str(i)) for i in range(
          len(INPUT))]
    C = SimpleNeuron(1., N-1, 0, N, "C")

    neurons = [C,D,inf]
    neurons = np.append(neurons, [A,a])

    synapses = []

```

```

synapses.extend([Synapse(x, D, 1., 1) for x in A])
synapses.extend([Synapse(A[i], a[i], 1., 1) for i in
    range(len(A))])
synapses.extend([Synapse(D, C, 1., 1)])
synapses.extend([Synapse(x, x, 1., 1) for x in a])
synapses.extend([Synapse(x, C, 1., 2) for x in a])
synapses.extend([Synapse(Inf, C, float(1-len(INPUT)), 3)
    ])

```

```
SUM = Network(neurons, synapses)
```

```
count = 0
```

```

while C.V + C.I < N and count < 1000:
    SUM.step()
    count += 1

```

```
return count
```

```
SUM.int([0, 0, 1, 1, 1])
```

References

- [1] James B. Aimone, Ojas Parekh, Cynthia A. Phillips, Ali Pinar, William Severa, and Helen Xu. “Dynamic Programming with Spiking Neural Computing”. In: *ICONS ’19* (2019). DOI: 10.1145/3354265.3354285. URL: <https://doi.org/10.1145/3354265.3354285>.
- [2] Abdullahi Ali and Johan Kwisthout. *A spiking neural algorithm for the Network Flow problem*. Nov. 2019.
- [3] John Backus. “Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs”. In: *Commun. ACM* 21.8 (Aug. 1978), 613–641. ISSN: 0001-0782. DOI: 10.1145/359576.359579. URL: <https://doi.org/10.1145/359576.359579>.
- [4] Samya Bagchi, Srikrishna S Bhat, and Atul Kumar. “O(1) time sorting algorithms using spiking neurons”. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 1037–1043. DOI: 10.1109/IJCNN.2016.7727312.
- [5] Brian Butterworth. *How do brains count?* YouTube. 2020. URL: <https://youtu.be/D1sPBCx1DQQ>.
- [6] Ralph Cavin, Paolo Lugli, and V.V. Zhirnov. “Science and Engineering Beyond Moore’s Law”. In: *Proceedings of the IEEE* 100 (May 2012), pp. 1720–1749. DOI: 10.1109/JPR0C.2012.2190155.
- [7] J. Denning Peter and G. Lewis Ted. “Exponential Laws of Computing Growth”. In: (2017). URL: <https://calhoun.nps.edu/handle/10945/59477>.
- [8] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd ed. Reading: Addison-Wesley, 1994.
- [9] Johan Kwisthout and Nils Donselaar. *On the computational power and complexity of Spiking Neural Networks*. 2020. arXiv: 2001.08439 [cs.CC].

- [10] Gordon E. Moore. “Progress in digital integrated electronics [Technical literature, Copyright 1975 IEEE. Reprinted, with permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, pp. 11-13.]” In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), pp. 36–37. DOI: 10.1109/N-SSC.2006.4804410.
- [11] Andreas Nieder and Stanislas Dehaene. “Representation of Number in the Brain”. In: *Annual Review of Neuroscience* 32.1 (2009), pp. 185–208.
- [12] William Severa, Ojas Parekh, Kristofor D. Carlson, Conrad D. James, and James B. Aimone. “Spiking network algorithms for scientific computing”. In: (2016), pp. 1–8. DOI: 10.1109/ICRC.2016.7738681.
- [13] Akke Toeter, Arne Diehl, Daphne Smits, and Victoria Bosch. “Implementation of a Minimum Dominating Set Approximation Algorithm in a Spiking Neural Network”. In: (2021).