An optical model of computation

Damien Woods and Thomas J. Naughton

TASS Research Group, Department of Computer Science, National University of Ireland, Maynooth, Ireland.

Abstract

We prove computability and complexity results for an original model of computation called the continuous space machine. Our model is inspired by the theory of Fourier optics. We prove our model can simulate analog recurrent neural networks, thus establishing a lower bound on its computational power. We also define a $\Theta(\log_2 n)$ unordered search algorithm with our model.

Key words: continuous space machine, unconventional model of computation, analog computation, optical computing, computability, computational complexity, analog recurrent neural network, Fourier transform, binary search, unordered search.

1 Introduction

In this paper we prove some computability and complexity results for an original continuous space model of computation called the continuous space machine (CSM). The CSM was developed for the analysis of (analog) Fourier optical computing architectures and algorithms, specifically pattern recognition and matrix algebra processors [1–4]. The functionality of the CSM is inspired by operations routinely performed by optical information processing scientists and engineers. The CSM operates in discrete timesteps over a finite number of two-dimensional (2D) complex-valued images of finite size and infinite spatial resolution. A finite control is used to traverse, copy, and perform other optical operations on the images. A useful analogy would be to describe the CSM as a random access machine, without conditional branching and with registers that hold continuous complex-valued images. It has recently

Email addresses: dwoods@cs.may.ie (Damien Woods), tom.naughton@may.ie (Thomas J. Naughton).

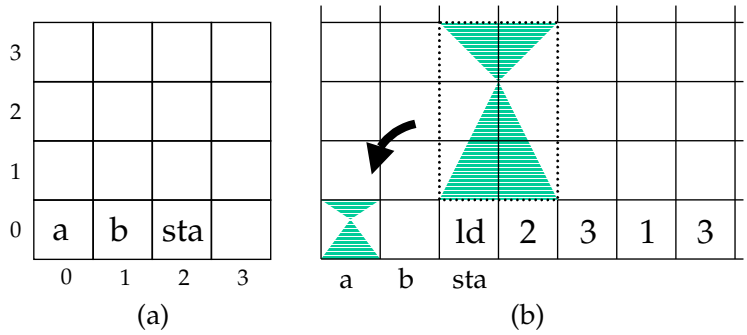


Fig. 1. Schematics of (a) the grid memory structure of the CSM, showing example locations for the 'well-known' addresses **a**, **b**, and **sta**, and (b) loading (and automatically rescaling) a subset of the grid into address **a**. The program | ld | 2 | 3 | 1 | 3 | *hlt* | instructs the machine to load into default address **a** the portion of the grid addressed by columns 2 through 3 and rows 1 through 3.

been established [5,6] that the CSM can simulate Turing machines and Type-2 machines [7]. However, the CSM's exact computational power has not yet been characterised.

In Sect. 2, we define our optical model of computation and give the data representations that will be used subsequently. In Sect. 3 we demonstrate a lower bound on computational power by proving that the CSM can simulate a type of dynamical system called analog recurrent neural networks (ARNNs) [8,9]. This simulation result proves our analog model can decide the membership problem for any language (of finite length words over a finite alphabet) in finite time. In Sect. 4, a $\Theta(\log_2 n)$ binary search algorithm that can be applied to certain unordered search problems is presented.

2 CSM

Each instance of the CSM consists of a memory containing a program (an ordered list of operations) and an input. Informally, the memory structure is in the form of a 2D grid of rectangular elements, as shown in Fig. 1(a). The grid has finite size and a scheme to address each element uniquely. Each grid element holds a 2D image. There is a program start address **sta** and two well-known addresses labelled **a** and **b**. The model has a number of operations that effect optical image processing tasks. For example, two operations available to the programmer, *st* and *ld* (parameterised by two column addresses and two row addresses), copy rectangular subsets of the grid out of and into image **a**, respectively. Upon such loading and storing the image contents are rescaled to the full extent of the target location [as depicted in Fig. 1(b)]. The other operations are image Fourier transform (FT), complex conjugation, multiplication, addition, amplitude thresholding, and some control flow operations.

2.1 CSM definition

Before defining the CSM we define its basic data unit and some of the functions it implements.

Definition 1 (Complex-valued image) *A complex-valued image (or simply, an image) is a function* $f : [0,1) \times [0,1) \to \mathbb{C}$*, where* $[0,1)$ *is the half-open real unit interval and* $\mathbb{C}$ *is the set of complex numbers.*

We let $\mathcal{I}$ be the set of all complex-valued images. We now define six functions that are implemented in six of the CSM's ten operations. Let each $f \in \mathcal{I}$ be parameterised by orthogonal dimensions x and y; we indicate this by writing f as $f(x,y)$. The function $h : \mathcal{I} \to \mathcal{I}$ gives the one-dimensional (1D) Fourier transformation (in the x-direction) of its 2D argument image f. The function h is defined as

$$h\left(f(x,y)\right) = h'\left(F\left(\alpha,y\right)\right) , \tag{1}$$

where $F(\alpha,y)$ is the FT in the x-direction of $f(x,y)$, defined as [1,2]

$$F(\alpha,y) = \int_{-\infty}^{\infty} f(x,y) \exp\left[\mathrm{i}2\pi\alpha x\right] \mathrm{d}x ,$$

where $\mathrm{i} = \sqrt{-1}$, and where $h'(F(\alpha,y)) = F(\theta\alpha,y)$. Here, h' uses the constant θ to linearly rescale its argument F so that F is defined over $[0,1) \times [0,1)$. The function $v : \mathcal{I} \to \mathcal{I}$ gives the 1D Fourier transformation (in the y-direction) of its 2D argument image f, and is defined as

$$v\left(f(x,y)\right) = v'\left(F(x,\beta)\right) , \tag{2}$$

where $F(x,\beta)$ is the FT in the y-direction of $f(x,y)$, defined as [1,2]

$$F\left(x,\beta\right) = \int_{-\infty}^{\infty} f(x,y) \exp\left[\mathrm{i}2\pi\beta y\right] \mathrm{d}y ,$$

and where $v'(F(x,\beta)) = F(x,\theta\beta)$. The function $* : \mathcal{I} \to \mathcal{I}$ gives the complex conjugate of its argument image,

$$*(f(x,y)) = f^*(x,y) , \tag{3}$$

where f^* denotes the complex conjugate of f. The complex conjugate of a scalar $z = a + ib$ is defined as $z^* = a - ib$. The function $\cdot : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ gives the pointwise complex product of its two argument images,

$$\cdot\left(f(x,y), g(x,y)\right) = f(x,y)g(x,y) . \tag{4}$$

The function $+ : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ gives the pointwise complex sum of its two argument images,

$$+\left(f(x,y), g(x,y)\right) = f(x,y) + g(x,y) . \tag{5}$$

The function $\rho : \mathcal{I} \times \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ performs amplitude thresholding on its first image argument using its other two real valued ($z_\mathrm{l}, z_\mathrm{u} : [0,1) \times [0,1) \to \mathbb{R}$) image arguments as lower and upper amplitude thresholds, respectively,

$$\rho\left(f(x,y), z_\mathrm{l}(x,y), z_\mathrm{u}(x,y)\right) = \begin{cases} z_\mathrm{l}(x,y), & \text{if } |f(x,y)| < z_\mathrm{l}(x,y) \\ |f(x,y)|, & \text{if } z_\mathrm{l}(x,y) \le |f(x,y)| \le z_\mathrm{u}(x,y) \\ z_\mathrm{u}(x,y), & \text{if } |f(x,y)| > z_\mathrm{u}(x,y) \ . \end{cases} \tag{6}$$

The amplitude of an arbitrary $z \in \mathbb{C}$ is denoted $|z|$ and is defined as $|z| = \sqrt{z(z^*)}$.

We let $\mathbb{N}$ be the set of nonnegative integers and for a given CSM we let $\mathcal{N}$ be a finite set of images that encode that CSM's addresses (see Sect. 2.6 for an example encoding).

Definition 2 (Continuous space machine) *A continuous space machine is a quintuple $M = (D, L, I, P, O)$, where*

$D = (m, n)$, $D \in \mathbb{N} \times \mathbb{N}$ *are the grid dimensions*
$L = ((s_\xi, s_\eta), (a_\xi, a_\eta), (b_\xi, b_\eta))$ *are the addresses* **sta**, **a**, *and* **b**
$I = \left\{\left(\iota_{1_\xi}, \iota_{1_\eta}\right), \ldots, \left(\iota_{k_\xi}, \iota_{k_\eta}\right)\right\}$ *are the addresses of the k input images*
$P = \left\{\left(\zeta_1, p_{1_\xi}, p_{1_\eta}\right), \ldots, \left(\zeta_r, p_{r_\xi}, p_{r_\eta}\right)\right\}$, $\zeta_j \in (\{h,\ v,\ *,\ \cdot,\ +,\ \rho,\ st,\ ld,\ br,\ hlt\} \cup \mathcal{N}) \subset \mathcal{I}$ *are the r programming symbols and their addresses*
$O = \left\{\left(o_{1_\xi}, o_{1_\eta}\right), \ldots, \left(o_{l_\xi}, o_{l_\eta}\right)\right\}$ *are the addresses of the l output images.*

Also, $(s_\xi, s_\eta), (a_\xi, a_\eta), (b_\xi, b_\eta), (\iota_{k'_\xi}, \iota_{k'_\eta}), (p_{r'_\xi}, p_{r'_\eta}), (o_{l'_\xi}, o_{l'_\eta}) \in \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$ *for all* $k'_\xi, k'_\eta \in \{1, \ldots, k\}$, $r'_\xi, r'_\eta \in \{1, \ldots, r\}$,$l'_\xi, l'_\eta \in \{1, \ldots, l\}$.

Addresses whose contents are not specified by P in a CSM definition are assumed to contain the constant image $f(x,y) = 0$.

We adopt a few notational conveniences. In a given CSM the addresses c and (γ, δ) are both elements from the set $\{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$. For the remainder of the current section, e, u, and w are sequences of elements from the set $\mathcal{I} \times \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$. In a CSM the image at address c is denoted $\widehat{c}$. In the case where $\widehat{c}$ represents an integer from $\{0, \ldots, |\mathcal{N}| - 1\}$, that integer is denoted $\widehat{\widehat{c}}$.

Definition 3 (CSM configuration) *A configuration of a CSM M is a pair $\langle c, e \rangle$, where $c \in \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$ is an address called the control. Also, $e = ((i_{0\,0}, 0, 0), \ldots, (i_{m-1\,n-1}, m-1, n-1))$ is a mn-tuple that contains M's mn images and each of their addresses, with $i_{\gamma\,\delta} \in \mathcal{I}$ being the image at address (γ, δ). The elements of tuple e are ordered first by each δ then by each γ.*

An *initial configuration* of M is a configuration $C_{\text{sta}} = \langle c_{\text{sta}}, e_{\text{sta}} \rangle$, where $c_{\text{sta}} = (s_\xi, s_\eta)$ is the address of **sta**, and e_{sta} contains all elements of P and elements $(\varphi_1, \iota_{1_\xi}, \iota_{1_\eta}), \ldots, (\varphi_k, \iota_{k_\xi}, \iota_{k_\eta})$ (the k input images at the addresses given by I). A *final configuration* of M is a configuration of the form $C_{\text{hlt}} = \langle (\gamma, \delta), (u, (hlt, \gamma, \delta), w) \rangle$, where u and w are given above. Notice that $\widehat{(\gamma, \delta)} = hlt$.

In Def. 4 we adopt the following notations. The function $\phi((\gamma, \delta)) = (\gamma + 1, \delta)$ advances the control. The notation $\phi^k(c)$ is shorthand for function composition, e.g. $\phi^2(c) = \phi(\phi(c))$. At a given configuration $\langle c, e \rangle$ we let $q_k = \widehat{\widehat{\phi^k(c)}}$, i.e. q_k represents the integer encoded by the image at address $\phi^k(c)$. We let the scaling relationships for st and ld be $x' = (x + \gamma - q_1)/(q_2 - q_1 + 1)$ and $y' = (y + \delta - q_3)/(q_4 - q_3 + 1)$. We let $a(x, y)$ be the image stored in address **a**. Recall that (a_ξ, a_η) is the address of **a**.

Definition 4 ($\vdash_M$) *Let $\vdash_M$ be a binary relation on configurations of CSM M containing exactly the following ten elements.*

$$\langle c, (u, (i_{a_\xi a_\eta}, a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi(c), (u, (h(i_{a_\xi a_\eta}), a_\xi, a_\eta), w) \rangle, \text{ if } \widehat{c} = h \quad \text{(i)}$$

$$\langle c, (u, (i_{a_\xi a_\eta}, a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi(c), (u, (v(i_{a_\xi a_\eta}), a_\xi, a_\eta), w) \rangle, \text{ if } \widehat{c} = v \quad \text{(ii)}$$

$$\langle c, (u, (i_{a_\xi a_\eta}, a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi(c), (u, (*(i_{a_\xi a_\eta}), a_\xi, a_\eta), w) \rangle, \text{ if } \widehat{c} = * \quad \text{(iii)}$$

$$\langle c, (u, (i_{a_\xi a_\eta}, a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi(c), (u, (\cdot(i_{a_\xi a_\eta}, i_{b_\xi b_\eta}), a_\xi, a_\eta), w) \rangle, \text{ if } \widehat{c} = \cdot \quad \text{(iv)}$$

$$\langle c, (u, (i_{a_\xi a_\eta}, a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi(c), (u, (+(i_{a_\xi a_\eta}, i_{b_\xi b_\eta}), a_\xi, a_\eta), w) \rangle, \text{ if } \widehat{c} = + \quad \text{(v)}$$

$$\langle c, (u, (i_{a_\xi a_\eta}, a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi(c), (u, (\rho(i_{a_\xi a_\eta}, \widehat{\phi(c)}, \widehat{\phi^2(c)}), a_\xi, a_\eta), w) \rangle, \text{ if } \widehat{c} = \rho \quad \text{(vi)}$$

$$\langle c, (u_{\gamma\delta}, (i_{\gamma\delta}(x, y), \gamma, \delta), w_{\gamma\delta}) \rangle \vdash_M \langle \phi^5(c), (u_{\gamma\delta}, (a\,(x', y'), \gamma, \delta), w_{\gamma\delta}) \rangle, \forall \gamma, \delta \text{ s.t. } q_1 \le \gamma \le q_2, q_3 \le \delta \le q_4, \forall (x, y) \in [0, 1) \times [0, 1), \text{ if } \widehat{c} = st \quad \text{(vii)}$$

$$\langle c, (u, (a\,(x', y'), a_\xi, a_\eta), w) \rangle \vdash_M \langle \phi^5(c), (u, (i_{\gamma\delta}(x, y), a_\xi, a_\eta), w) \rangle, \forall \gamma, \delta \text{ s.t. } q_1 \le \gamma \le q_2, q_3 \le \delta \le q_4, \forall (x, y) \in [0, 1) \times [0, 1), \text{ if } \widehat{c} = ld \quad \text{(viii)}$$

$$\langle c, (u) \rangle \vdash_M \langle (\widehat{\widehat{\phi(c)}}, \widehat{\widehat{\phi^2(c)}}), (u) \rangle, \text{ if } \widehat{c} = br \quad \text{(ix)}$$

$$\langle c, (u) \rangle \vdash_M \langle c, (u) \rangle, \text{ if } \widehat{c} = hlt \quad \text{(x)}$$

Elements (i) to (vi) of $\vdash_M$ define the CSM's implementation of the functions defined in Eqs. (1) through (6). Notice that in each case the image at the well-known address **a** is overwritten by the result of applying one of h, v, $*$, $\cdot$, $+$ or ρ to its argument (or arguments). The value of the control c is then simply incremented to the next address, as defined in Def. 3. Element (vii) of $\vdash_M$ defines how the store operation copies the image at well-known

address **a** to a 'rectangle' of images specified by the *st* parameters q_1, q_2, q_3, q_4. Element (viii) of $\vdash_M$ defines how the load operation copies a rectangle of images specified by the *ld* parameters q_1, q_2, q_3, q_4 to the image at well-known address **a**. Elements (ix) and (x) of $\vdash_M$ define the control flow operations branch and halt, respectively. When the image at the address specified by the control c is *br*, the value of c is updated to the address encoded by the two *br* parameters. Finally, the *hlt* operation always maps a final configuration to itself.

Let $\vdash_M^*$ denote the reflexive and transitive closure of $\vdash_M$. A halting computation by M is a finite sequence of configurations beginning in an initial configuration and ending in a final configuration: $C_{\text{sta}} \vdash_M^* C_{\text{hlt}}$.

For convenience, we use an informal 'grid' notation when specifying programs for the CSM, see for example Fig. 1. In our grid notation the first and second elements of an address tuple refer to the horizontal and vertical axes of the grid, respectively, and image $(0, 0)$ is at the bottom left-hand corner of the grid. The images in a grid must have the same orientation as the grid. Hence in a given image f, the first and second elements of a coordinate tuple refer to the horizontal and vertical axes of f, respectively, and the coordinate $(0, 0)$ is located at the bottom left-hand corner of f. Figure 2 informally explains the elements of $\vdash_M$, as they appear in this grid notation. After giving some data representations in Sect. 2.4 we will then define language membership deciding by CSM. First we suggest physical interpretations for some of the CSM's operations and then we give a number of complexity measures.

2.2 Optical realisation

In this section, we outline how some of the elementary operations of our model could be carried out physically. We do not intend to specify the definitive realisation of any of the operations, but simply convince the reader that the model's operations have physical interpretations. Furthermore, although we concentrate on implementations employing visible light (optical frequencies detectable to the human eye) the CSM definition does not preclude employing other portion(s) of the electromagnetic spectrum.

A complex-valued image could be represented physically by a spatially coherent optical wavefront. Spatially coherent illumination (light of a single wavelength and emitted with the same phase angle) can be produced by a laser. A spatial light modulator (SLM) could be used to encode the image onto the expanded and collimated laser beam. One could write to a SLM offline (expose photographic film, or laser print or relief etch a transparency) or online (in the case of a liquid-crystal display [10–12] or holographic material [13,14]). The

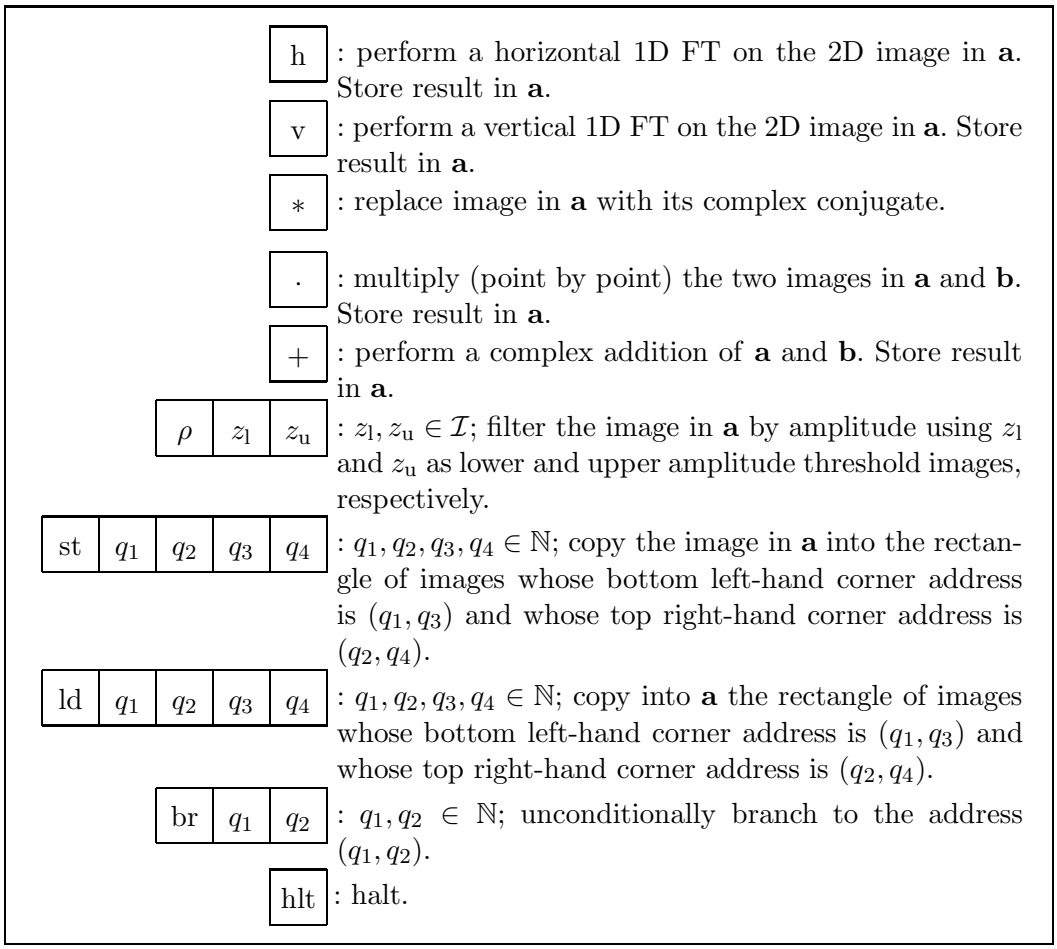


Fig. 2. The set of CSM operations, given in our informal grid notation. For formal definitions see Def. 4.

functions h and v could be effected using two convex cylindrical lenses, oriented horizontally and vertically, respectively [1,2,11,15]. A coherent optical wavefront will naturally evolve into its own Fourier spectrum as it propagates to infinity. What we do with a convex lens is simply image at a finite distance this spectrum at infinity. This finite distance is called the focal length of the lens. The constant θ used in the definitions of h and v could be effected using Fourier spectrum size reduction techniques [1,2] such as varying the focal length of the lens, varying the separation of the lens and SLM, employing cascaded Fourier transformation, increasing the dimensions/reducing the spatial resolution of the SLM, or using light with a shorter wavelength. The function $*$ could be implemented using a phase conjugate mirror [16]. The function $\cdot$ could be realised by placing a SLM encoding an image f in the path of a wavefront encoding another image g [1–3]. The wavefront immediately behind the SLM would then be $\cdot(f, g)$. The function $+$ describes the superposition of two optical wavefronts. This could be achieved using a 50:50 beam splitter [1,16,4]. The function ρ could be implemented using an electronic camera or a liquid-crystal light valve [12]. The parameters z_l and z_u would then be physical characteristics of the particular camera/light valve

used. z_l corresponds to the minimum intensity value that the device responds to, known as the dark current signal, and z_u corresponds to the maximum intensity (the saturation level).

A note will be made about the possibility of automating of these operations. If suitable SLMs can be prepared with the appropriate 2D pattern(s), each of the operations h, v, $*$, $\cdot$, and $+$ could be effected autonomously and without user intervention using appropriately positioned lenses and free space propagation. The time to effect these operations would be the sum of the flight time of the image (distance divided by velocity of light) and the response time of the analog 2D detector; both of which are constants independent of the size or resolution of the images if an appropriate 2D detector is chosen. Examples of appropriate detectors would be holographic material [13,14] and a liquid-crystal light valve with a continuous (not pixellated) area [12]. Since these analog detectors are also optically-addressed SLMs, we can very easily arrange for the output of one function to act as the input to another, again in constant time independent of the size or resolution of the image. A set of angled mirrors will allow the optical image to be fed back to the first SLM in the sequence, also in constant time. It is not known, however, if ρ can be carried out completely autonomously for arbitrary parameters. Setting arbitrary parameters might fundamentally require offline user intervention (adjusting the gain of the camera, and so on), but at least for a small range of values this can be simulated online using a pair of liquid-crystal intensity filters.

We have outlined some optics principles that could be employed to implement the operations of the model. The simplicity of the implementations hides some imperfections in our suggested realisations. For example, the implementation of the $+$ operation outlined above results in an output image that has been unnecessarily multiplied by the constant factor 0.5 due to the operation of the beam splitter. Also, in our suggested technique, the output of the ρ function is squared unnecessarily. However, all of these effects can be compensated for with a more elaborate optical setup and/or at the algorithm design stage, and do not affect the proofs presented in this paper.

A more important issue concerns the quantum nature of light. According to our current understanding, light exists as individual packets called photons. As such, in order to physically realise the CSM one would have to modify it such that images would have discrete, instead of continuous, amplitudes. The atomic operations outlined above, in particular the FT, are not affected by the restriction to quantised amplitudes, as the many experiments with electron interference patterns indicate. We would still assume, however, that in the physical world space is continuous.

A final issue concerns how a theoretically infinite Fourier spectrum could be represented by an image (or encoded by a SLM) of finite extent. This difficulty

	Symbol	Name	Description
1.	T	TIME	Number of timesteps
2.	G	GRID	Number of grid images
3.	R_{S}	SPATIALRES	Spatial resolution
4.	R_{A}	AMPLRES	Amplitude resolution
5.	R_{P}	PHASERES	Phase resolution
6.	R_{D}	DYRANGE	Dynamic range
7.	ν	FREQ	Frequency of illumination

Table 1
Summary of complexity measures for characterising CSMs.

is addressed with the FREQ complexity measure in the next section.

2.3 Complexity measures

Computational complexity measures are used to analyse CSMs. We define seven complexity measures (summarised in Table 1). The TIME complexity of a CSM M is the number of configurations in the computation sequence of an arbitrary instance of M, beginning with the initial configuration and ending with the first final configuration. The GRID complexity of a CSM M is the number of image elements in M's grid. In this paper the GRID complexity of M is always a constant (independent of its input).

The SPATIALRES complexity of CSM M is the minimum spatial resolution of M's images necessary for M to compute correctly on all inputs. This is formalised as follows. Let a *pixel* be a constant function $\lambda : [0, 1/\Phi) \times [0, 1/\Psi) \rightarrow z$ where $\Phi, \Psi \in \{1, 2, 3, \ldots\}$ and $[0, 1/\Phi), [0, 1/\Psi) \subset \mathbb{R}$ and $z \in \mathbb{C}$. Let a *raster image* be an image composed entirely of nonoverlapping pixels, each of the pixels are of width $1/\Phi$, height $1/\Psi$, identical orientation, and arranged into Φ rows and Ψ columns. (An image displayed on a monochrome television screen or liquid crystal display panel would be an example of a raster image, if we let its height and width equal 1.) Let the *spatial resolution* of a raster image be $\Phi\Psi$, the number of pixels in that image. Let the process of *rasterising* an image be the function $S : \mathcal{I} \times (\mathbb{N} \times \mathbb{N}) \rightarrow \mathcal{I}$, defined as $S(f(x, y), (\Phi, \Psi)) = f'(x, y)$, where $f'(x, y)$ is a raster image, with $\Phi\Psi$ pixels arranged in Φ columns and Ψ rows, that somehow approximates $f(x, y)$. The details of S are not important; it suffices to say that (Φ, Ψ) can be regarded as defining a sampling grid with uniform sampling both horizontally and vertically, although the sampling rates in both directions can differ. Increasing the spatial resolution of the sampling (increasing Φ and/or Ψ) results in a better approximation of $f(x, y)$. The SPATIALRES complexity of a CSM M is then defined as the minimum $\Phi\Psi$ (the

lowest resolution uniform sampling) such that if each image $f_{\gamma\delta}(x, y)$ in M is replaced with $S(f_{\gamma\delta}(x, y), (\Phi, \Psi))$ then M computes correctly on all inputs. If no such $\Phi\Psi$ exists then M has infinite SPATIALRES complexity. It can be seen that if the result of M's computation is determined solely by features within its images that are located at rational (respectively, irrational) coordinates then M would require finite (respectively, infinite) SPATIALRES. In optical image processing terms, and given the fixed size of our images, SPATIALRES corresponds to the space-bandwidth product of a detector or SLM.

The AMPLRES complexity of a CSM M is the minimum amplitude resolution necessary for M to compute correctly on all inputs. This is formalised as follows. Consider the following function $A : \mathcal{I} \times \{1, 2, 3, \ldots\} \rightarrow \mathcal{I}$ defined as

$$A(f(x, y), \mu) = \lfloor |f(x, y)|\mu + 0.5 \rfloor \mu^{-1} \exp(\mathrm{i} \times \mathrm{angle}(f(x, y))) \,, \tag{7}$$

where $|\cdot|$ returns the amplitudes of its image argument, angle$(\cdot)$ returns the phase angles (in the range $(-\pi, \pi]$) of its image argument, and the floor operation is defined as operating separately on each value in its image argument. The value μ is the cardinality of the set of discrete nonzero amplitude values that each complex value in $A(f, \mu)$ can take, per half-open unit interval of amplitude. (Zero will always be a possible amplitude value irrespective of the value of μ.) To aid in the understanding of Eq. (7), note that the following equality always holds

$$f(x, y) = |f(x, y)| \exp(\mathrm{i} \times \mathrm{angle}(f(x, y))) \,.$$

Then, the AMPLRES complexity of a CSM M is defined as the minimum μ such that if each image $f_{\gamma\delta}(x, y)$ in M is replaced by $A(f_{\gamma\delta}(x, y), \mu)$ then M computes correctly on all inputs. If no such μ exists then M has infinite AMPLRES complexity. It can be seen that if the result of M's computation is determined solely by amplitude values within its images that are rational (respectively, irrational), or by a finite (respectively, infinite) set of rational amplitude values, then M would require finite (respectively, infinite) AMPLRES. The only two values for AMPLRES complexity of interest in this paper are constant AMPLRES and infinite AMPLRES. CSM instances that only make use of unary and binary images (see Sect. 2.4) have constant AMPLRES of 1. Instances that use real number and real matrix images (see Sect. 2.4) have infinite AMPLRES complexity. In optical image processing terms AMPLRES corresponds to the amplitude quantisation of a signal.

The PHASERES complexity of a CSM M is the minimum phase resolution necessary for M to compute correctly on all inputs. This is formalised as follows. Consider the following function $P : \mathcal{I} \times \{1, 2, 3, \ldots\} \rightarrow \mathcal{I}$ defined as

$$P(f(x, y), \mu) = |f(x, y)| \exp\left(\mathrm{i} \left\lfloor \mathrm{angle}(f(x, y)) \frac{\mu}{2\pi} + 0.5 \right\rfloor \frac{2\pi}{\mu}\right) .$$

The value μ is the cardinality of the set of discrete phase values that each complex value in $P(f, \mu)$ can take. Then, the PHASERES complexity of a CSM M is defined as the minimum μ such that if each image $f_{\gamma\delta}(x, y)$ in M is replaced by $P(f_{\gamma\delta}(x, y), \mu)$ then M computes correctly on all inputs. If no such μ exists then M has infinite PHASERES complexity. It can be seen that if the result of M's computation is determined solely by phase values within its images that are rational (respectively, irrational) modulo 2π, or by a finite (respectively, infinite) set of rational phase values modulo 2π, then M would require finite (respectively, infinite) PHASERES. In optical image processing terms PHASERES corresponds to the phase quantisation of a signal.

The DYRANGE complexity of a CSM M is defined as the maximum of all the amplitude values stored in all of M's images during M's computation. In optical processing terms DYRANGE corresponds to the dynamic range of a signal.

The seventh of our complexity measures is FREQ. The FREQ complexity of a CSM M is the minimum optical frequency necessary for M to compute correctly. The concept of minimum optical frequency is now explained. In optical implementations of the h and v operations (such as our suggestions in Sect. 2.2), one of the factors that determine the dimensions of the Fourier spectrum of $f \in \mathcal{I}$ is the frequency of the coherent illumination employed. Increasing the frequency of the illumination results in a smaller Fourier spectrum (components are spatially closer to the zero frequency point). In our definitions of h and v, we employ the constant θ to rescale the Fourier spectrum of f such that it fits into the dimensions of an image: $[0, 1) \times [0, 1)$. In general, however, a Fourier spectrum of an image will be infinite in extent. Therefore, according to the relationship between optical frequency and Fourier spectrum dimensions [2,1], such a constant θ only exists when the wavelength of the illumination is zero, corresponding to illumination with infinite frequency. With a finite optical frequency, the h and v operations will remove all Fourier components with a spatial frequency higher than the cut-off imposed by θ. This is called low-pass filtering in signal processing terminology, and is equivalent to a blurring of the original signal. Given particular rasterisation and quantisation functions for the images in M, and a particular θ, the blurring effect might not influence the computation. Formally, then, we define the FREQ complexity of a CSM M to be the minimum optical frequency that can be employed such that M computes correctly on all inputs. If approximations of a FT are sufficient for M, or if M does not execute h or v, then M requires finite FREQ. If the original (unbounded) definitions of h and v must hold then M requires infinite FREQ. Note also that using the traditional optical methods outlined in Sect. 2.2, any lower bound on SPATIALRES complexity will impose a lower bound on FREQ complexity. In the context of traditional optical methods, this imposition is referred to as the diffraction limit. (The optical wavelength should be a constant times smaller than the smallest spatial feature that needs

to be resolvable in an image.) In order not to rule out the applicability of novel sub-wavelength resolution techniques that go beyond the diffraction limit for our CSM algorithms we give each FREQ complexity as an upper bound $[O(\cdot)]$.

Finally, one might also consider energy a natural complexity measure. In fact, energy is a function of all of the measures in Table 1, with the exception of PHASERES. Such an interpretation is consistent with the quantum theory of light. This is explained in the case of a single image initially. Let $f(x, y)$ be an image with spatial resolution R_{S}, amplitude resolution R_{A}, dynamic range R_{D}, and encoded with illumination of frequency ν. An upper bound on the energy required to represent (and to measure) $f(x, y)$, denoted E_f, is defined as

$$E_f = \text{h}\nu d R_{\text{S}} {R_{\text{A}}}^2 {R_{\text{D}}}^2 \ , \tag{8}$$

where $\text{h} = 6.626\ldots \times 10^{-34}$ Js is the fundamental physical constant called Planck's constant, and d (detector sensitivity) is the minimum number of photons required to trigger a response in a detector element. (For example, for an isolated rod element in the human retina, d is 10 or so, and for the detector element in a photomultiplier tube d can be 1.) An upper bound on the energy required for a CSM M's computation can be determined by considering the worst cost scenario that at every timestep the image f with maximum amplitude is written to every grid element, giving a total energy E_M of

$$E_M = E_f T G \, ,$$

where the T and G are M's TIME and SPACE complexities respectively.

2.4 Representing data as images

Unless otherwise stated let $\Sigma = \{0, 1\}$. As is usual let $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$, let $\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i$ and, unless otherwise stated, let a language $L \subseteq \Sigma^+$. There are many ways to represent elements of finite, countable, and uncountable sets as images. We give a number of techniques that will be used later in the paper. The symbol 1 is represented by an image having value one at its centre and value zero everywhere else. An image that has value zero everywhere represents the symbol 0.

Definition 5 (binary symbol image) *The symbol $\psi \in \Sigma$ is represented by the binary symbol image f_ψ,*

$$f_\psi(x,y) = \begin{cases} 1, & \text{if } x,y = 0.5, \psi = 1 \\ 0, & \text{otherwise } . \end{cases}$$

We extend this representation scheme to binary words using 'stack' and 'list' images.

Definition 6 (binary stack image) *The word $w = w_1 w_2 \cdots w_k \in \Sigma^+$ is represented by the binary stack image f_w,*

$$f_w(x,y) = \begin{cases} 1, & \text{if } x = 1 - \frac{3}{2^{k-i+2}}, y = 0.5, w_i = 1 \\ 0, & \text{otherwise } , \end{cases}$$

where $w_i \in \Sigma, 1 \leq i \leq k$. Image f_w is said to have length k and the pair (f_w, k) uniquely represents w.

Definition 7 (binary list image) *The word $w = w_1 w_2 \cdots w_k \in \Sigma^+$ is represented by the binary list image f_w,*

$$f_w(x,y) = \begin{cases} 1, & \text{if } x = \frac{2i-1}{2k}, y = 0.5, w_i = 1 \\ 0, & \text{otherwise } , \end{cases}$$

where $w_i \in \Sigma, 1 \leq i \leq k$. Image f_w is said to have length k and the pair (f_w, k) uniquely represents w.

If $\Sigma = \{1\}$ we replace the word "binary" with the word "unary" in Defs. 5, 6, and 7. In Defs. 6 and 7 each unary/binary symbol in w is represented by a corresponding value of 0 or 1 in f_w. Notice that in the unary/binary stack image, w's leftmost symbol w_1, is represented by the rightmost value in the sequence of values representing w in f_w, this means that w_k is represented by the topmost stack element. We represent a single real value r by an image with a single peak of value r.

Definition 8 (real number image) *The real number $r \in \mathbb{R}$ is represented by the real number image f_r,*

$$f_r(x,y) = \begin{cases} r, & \text{if } x,y = 0.5 \\ 0, & \text{otherwise } . \end{cases}$$

To represent a $R \times C$ matrix of real values we define RC peaks that represent the matrix values and use both dimensions of a stack-like image.

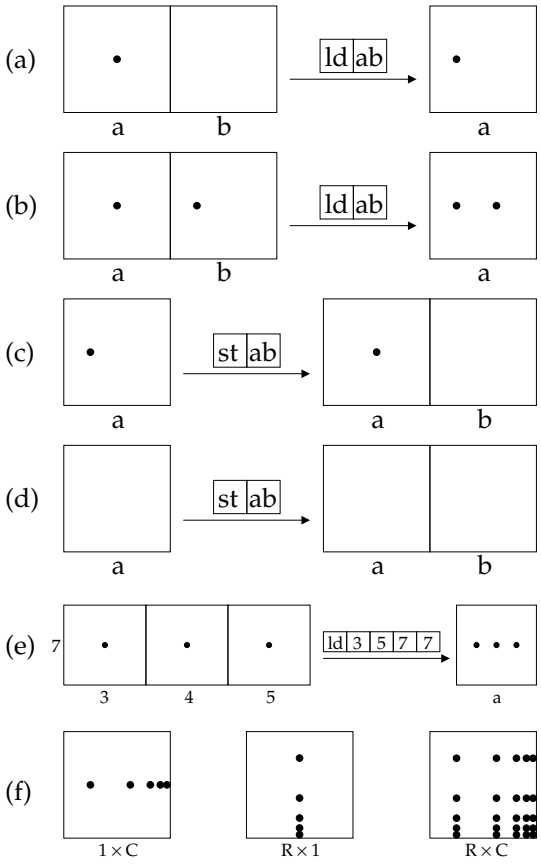


Fig. 3. Representing data by images through the positioning of peaks. The nonzero peaks are coloured black and the white areas denote value 0. (a) Pushing a unary symbol image onto an empty stack image. (b) Pushing a unary symbol image onto a stack image representing the word 1 to create the representation of 11. (c) Popping a stack representing the word 1, resulting in a popped unary symbol image (in **a**) and an empty stack (in **b**). (d) Popping an empty stack. (e) Rescaling three adjacent unary symbol images into a single unary list image (in **a**) representing 111. (f) $1 \times C$, $R \times 1$, and $R \times C$ matrix images where $R = C = 5$.

Definition 9 ($R \times C$ matrix image) *The $R \times C$ matrix A, with real-valued components $a_{ij}, 1 \leq i \leq R, 1 \leq j \leq C$, is represented by the $R \times C$ matrix image f_A,*

$$f_A(x,y) = \begin{cases} a_{ij}, & \text{if } x = 1 - \frac{1+2k}{2^{j+k}}, y = \frac{1+2l}{2^{i+l}} \\ 0, & \text{otherwise} \end{cases},$$

where

$$k = \begin{cases} 1, & \text{if } j < C \\ 0, & \text{if } j = C \end{cases}, \quad l = \begin{cases} 1, & \text{if } i < R \\ 0, & \text{if } i = R \end{cases}.$$

This matrix image representation is illustrated in Fig. 3(f).

The representations given in Defs. 5 through 9 are conveniently manipulated in the CSM using a programming technique called 'rescaling'. Binary symbol images can be combined using stepwise rescaling (creating a binary stack image) or with a single rescale operation (creating a binary list image). A stack representation of the word 11 could be generated as follows. Take the image f_0 (having value 0 everywhere), representing an empty stack, and a unary symbol image f_1 that we will 'push' onto the stack. A push is accomplished by placing the images side-by-side with f_1 to the left and rescaling both into a single image location. The image at this location is a (binary or unary) stack image representing the word 1. This concept is illustrated in Fig. 3(a); a unary symbol image is placed at address **a** and an empty stack image is placed at address **b**. The command ld|ab pushes the symbol onto the empty stack, and by default the result is stored in address **a**. Take another unary symbol image f_1, place it to the left of the stack image, and rescale both into the stack image location once again. The unary stack image contains two peaks at particular locations that testify that it is a representation of the word 11, as illustrated in Fig. 3(b). To remove a 1 from the stack image, a 'pop' operation is applied. Rescale the stack image over any two image locations positioned side-by-side. The image to the left will contain the symbol that had been at the top of the stack image (f_1) and the image to the right will contain the remainder of the stack image, as illustrated in Fig. 3(c). The stack image can be repeatedly rescaled over two images popping a single image each time. Popping an empty stack [Fig. 3(d)] results in the binary symbol image representing 0 and the stack remaining empty.

We can interpret a unary stack image as a nonnegative integer. Push and pop can then be interpreted as increment and decrement operations, respectively. As a convenient pseudocode, we use statements such as `c.push(1)` and `c.pop()` to increment and decrement the unary word represented by the stack image at address **c**. Binary representations of nonnegative integers would be represented in a similar manner. A unary stack representation of the integer 2 could be regarded as a binary stack representation of the integer 3. Our convention is to represent words with the rightmost symbol at the top of the stack. Therefore, if the second f_1 in the preceding example had been instead f_0 the resulting push operation would have created a stack image representing the word 10 (or alternatively, the binary representation of the integer 2). Pushing (or popping) k binary or unary symbol images to (or from) a binary or unary stack image requires $\Theta(k)$ TIME, constant GRID, $\Theta(2^k)$ SPATIALRES, 1 AMPLRES, 1 PHASERES, 1 DYRANGE and $O(2^k)$ FREQ. For CSM algorithms that use stack representations, SPATIALRES (and therefore FREQ) are of critical concern.

In the list image representation of a unary or binary word, each of the rescaled binary symbol images are equally spaced (unlike the stack image representation). The binary list image representation of a word $w \in \Sigma^+$, $|w| = k$, involves

placing k symbol images (representing the k symbols of w) side-by-side in k contiguous image locations and rescaling them into a single image in a *ld* operation. For example in Fig. 3(e) a unary list representation of the unary word 111 is accomplished by the command $\boxed{\text{ld}\,|\,3\,|\,5\,|\,7\,|\,7}$. Rescaling k binary or unary symbol images to form a binary or unary list image, or rescaling a binary or unary list image to form k binary or unary symbol images both require constant TIME, constant GRID, $\Theta(k)$ SPATIALRES, 1 AMPLRES, 1 PHASERES, 1 DYRANGE and $O(k)$ FREQ.

The $R \times C$ matrix image representation can be manipulated using image rescaling not only in the horizontal direction (as push and pop given above), but also in the vertical direction. In the matrix representation an initial empty image (to push to) is not required. Pushing (or popping) j real number images to (or from) a $j \times 1$ or $1 \times j$ matrix image requires $\Theta(j)$ TIME, constant GRID, $\Theta(2^{j-1})$ SPATIALRES, infinite AMPLRES, constant PHASERES, 1 DYRANGE and $O(2^{j-1})$ FREQ. Pushing (or popping) k of $j \times 1$ matrix images or k of $1 \times j$ matrix images to (or from) a $j \times k$ or $k \times j$ matrix image requires $\Theta(k)$ TIME, constant GRID, $\Theta(2^{j+k-2})$ SPATIALRES, infinite AMPLRES, 1 PHASERES, 1 DYRANGE and $O(2^{j+k-2})$ FREQ.

2.5 CSM deciding language membership

Definition 10 (CSM deciding language membership) *CSM M_L decides the membership problem for $L \subseteq \Sigma^+$ if M_L has initial configuration $\langle c_{sta}, e_{sta}\rangle$ and final configuration $\langle c_{hlt}, e_{hlt}\rangle$, and the following hold:*

- *sequence e_{sta} contains the two input elements $(f_w, \iota_{1_\xi}, \iota_{1_\eta})$ and $(f_{1^{|w|}}, \iota_{2_\xi}, \iota_{2_\eta})$*
- *e_{hlt} contains the output element $(f_1, o_{1_\xi}, o_{1_\eta})$ if $w \in L$*
- *e_{hlt} contains the output element $(f_0, o_{1_\xi}, o_{1_\eta})$ if $w \notin L$*
- *$\langle c_{sta}, e_{sta}\rangle \vdash_M^* \langle c_{hlt}, e_{hlt}\rangle$, for all $w \in \Sigma^+$.*

Here f_w is the binary stack image representation of $w \in \Sigma^+$, $f_{1^{|w|}}$ is the unary stack image representation of the unary word $1^{|w|}$. Images f_0 and f_1 are the binary symbol image representations of the symbols 0 and 1, respectively.

In this definition addresses $(\iota_{1_\xi}, \iota_{1_\eta})$, $(\iota_{2_\xi}, \iota_{2_\eta}) \in I$ and address $(o_{1_\xi}, o_{1_\eta}) \in O$, where I and O are as given in Def. 2. We use the stack image representation of words. The unary input word $1^{|w|}$ is necessary for M_L to determine the length of input word w. (For example the binary stack image representations of the words 00 and 000 are identical.)

Our model uses symbols from a finite set in its addressing scheme and employs an address resolution technique to effect decisions (see Sect. 2.7). Therefore, during branching and looping, variables encoded by elements of the uncountable set of continuous images must be transformed to the finite set of addresses. In one of the possible addressing schemes available to us, we use symbols from the set $\{0,1\}$. We choose $B = \{w : w \in \{0,1\}^{\max(m,n)}, w \text{ has a single } 1\}$ as our underlying set of address words. Each of the m column and n row addresses will be a binary word from the finite set B. An ordered pair of such binary words identifies a particular image in the grid. Each element of B will have a unique image encoding. $\mathcal{N}$ is the set of encoded images, with $|\mathcal{N}| = \max(m,n)$. In order to facilitate an optical implementation of our model we cannot assume to know the particular encoding strategy for the set $\mathcal{N}$ (such as the simple binary stack or list representations of Sect. 2.4). We choose a correlation based address resolution technique. The address resolution technique chosen (the transformation from $\mathcal{I}$ to B) must be general enough to resolve addresses that use any reasonable encoding (see Sect. 2.6.1).

Given an image $\chi \in \mathcal{I}$ we wish to determine which address word in B is encoded by χ. In general, comparing a continuous image χ with the elements of $\mathcal{N}$ to determine membership is not guaranteed to terminate. However, for each χ that our addressing scheme will be presented with, and given a reasonable encoding for $\mathcal{N}$, we can be sure of the following restrictions on $\mathcal{N}$: (i) $\chi \in \mathcal{N}$, (ii) $|\mathcal{N}|$ is finite, and (iii) $\mathcal{N}$ contains distinct images (no duplicates). Given these restrictions, we need only search for the single closest match between χ and the elements of $\mathcal{N}$. We choose a transformation based on cross-correlation (effected through a sequence of FT and image multiplication steps) combined with a thresholding operation.

The function $t : \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{N}$ is defined as

$$t(\chi, \epsilon) = \tau\left(\circledast(\epsilon, \chi)\right) \quad , \tag{9}$$

where χ encodes the unknown addressing image to be transformed, ϵ is a list image formed by rescaling all the elements of $\mathcal{N}$ (in some known order) into a single image using one *ld* operation, $\circledast$ denotes the cross-correlation function, and τ is a thresholding operation. The cross-correlation function [1,2] produces an image $f_{\text{corr}} = \circledast(\epsilon, \chi)$ where each point (u, v) in f_{corr} is defined

$$f_{\text{corr}}(u,v) = \int_0^1 \int_0^1 \epsilon(x,y)\chi^*(x+u, y+v)\mathrm{d}x\mathrm{d}y \quad , \tag{10}$$

where χ^* denotes the complex conjugate of χ, where (x, y) specifies coordinates in ϵ and χ, and where $(+u, +v)$ denotes an arbitrary relative shift between ϵ

and χ expressed in the coordinate frame of f_{corr}. In Eq. (10), let χ have value 0 outside of $[0,1) \times [0,1)$. Let image f_{corr} be defined only over $[0,1) \times [0,1)$. In the CSM, $f_{\text{corr}}(u,v)$ would be produced in image **a** by the code fragment ld | ϵ | h | v | st | b | ld | χ | $*$ | h | v | $\cdot$ | h | v, where a multiplication in the Fourier domain is used to effect cross-correlation [1–3]. According to Eq. (10), and given a reasonable encoding for $\mathcal{N}$ (implying the three restrictions outlined above), f_{corr} will contain exactly one well resolved maximum amplitude value. This point of maximum amplitude will be a nonzero value at a position identical to the relative positioning of the list element in ϵ that most closely matches χ. All other points in f_{corr} will contain an amplitude less than this value.

We define the thresholding operation of Eq. (9) for each point (u,v) in f_{corr} as

$$\tau\left(f_{\text{corr}}(u,v)\right) = \begin{cases} 1, & \text{if } |f_{\text{corr}}(u,v)| = \max(|f_{\text{corr}}(u,v)|) \\ 0, & \text{if } |f_{\text{corr}}(u,v)| < \max(|f_{\text{corr}}(u,v)|) \end{cases} .$$

This produces an image with a single nonzero value at coordinates $u = (2i+1)/[2 \times \max(m,n)], v = 0.5$ for some positive integer i in the range $[0, \max(m,n) - 1]$. From the definition of a binary list image (Def. 7), we can see that these unique identifiers are exactly the images that represent the binary words corresponding to the integers $\{2^0, 2^1, 2^2, \ldots, 2^{[\max(m,n)-1]}\}$. Therefore, t is a function from continuous images to the set of image representations of the finite set B defined earlier.

2.6.1 Reasonable encodings of $\mathcal{N}$

A note is required on what constitutes a reasonable encoding for $\mathcal{N}$, such that t will correctly transform s to an image representation of the appropriate element in B. There are two considerations which one needs to bear in mind when designing an encoding for $\mathcal{N}$. Firstly, Eq. (10) is not a normalised cross-correlation. Therefore, $\mathcal{N}$ has to be chosen such that the autocorrelation of each element of $\mathcal{N}$ has to return a strictly larger maximum value than the cross-correlation with each of the other elements of $\mathcal{N}$.

Secondly, one may wish to choose f_0 (the image with zero everywhere) as an element of $\mathcal{N}$. We can see from Eq. (10) that this will result in a cross-correlation of $f_{\text{corr}} = f_0$ when we try to match $s = f_0$ with ϵ. If one chooses f_0 as an element of $\mathcal{N}$, this special case can be resolved (without the need for an explicit comparison with f_0) with the following rule. Given that $\mathcal{N}$ is a reasonable encoding, if no single well resolved maximum amplitude value is generated from $\circledast$, we assume that $\chi = f_0$. (In all cases other than when $\chi = f_0$, f_{corr} will contain a well resolved point of maximum amplitude, as explained above.)

2.7 Conditional branching from unconditional branching

Our model does not have a conditional branching operation as a primitive; it was felt that giving the model the capability for equality testing of continuous images would rule out any possible implementation. However, we can effect indirect addressing through a combination of program self-modification and direct addressing. We can then implement conditional branching by combining indirect addressing and unconditional branching. This is based on a technique by Rojas [17] that relies on the fact that $|\mathcal{N}|$ is finite. Without loss of generality, we could restrict ourselves to two possible symbols 0 and 1. Then, the conditional branching instruction "if (k=1) then jump to address X, else jump to Y" is written as the unconditional branching instruction "jump to address k". We are required only to ensure that the code corresponding to addresses X and Y is stored at addresses 1 and 0, respectively. In a 2D memory (with an extra addressing coordinate in the horizontal direction) many such branching instructions are possible in a single machine.

2.8 A general iteration construct

Our bounded iteration construct is based on the conditional branching instruction outlined in Sect. 2.7. Consider a loop of the following general form, written in some unspecified language,

```
SX
while (e > 0)
  SY
  e := e - 1
end while
SZ
```

where variable `e` contains a nonnegative integer specifying the number of remaining iterations, and `SX`, `SY`, and `SZ` are arbitrary lists of statements. Without loss of generality, we assume that statements `SY` do not write to `e` and do not branch to outside of the loop. If `e` is represented by a unary stack image (where the number of represented 1s equals the value of `e`), this code could be rewritten as

```
SX
while (e.pop() = f1)
  SY
end while
SZ
```

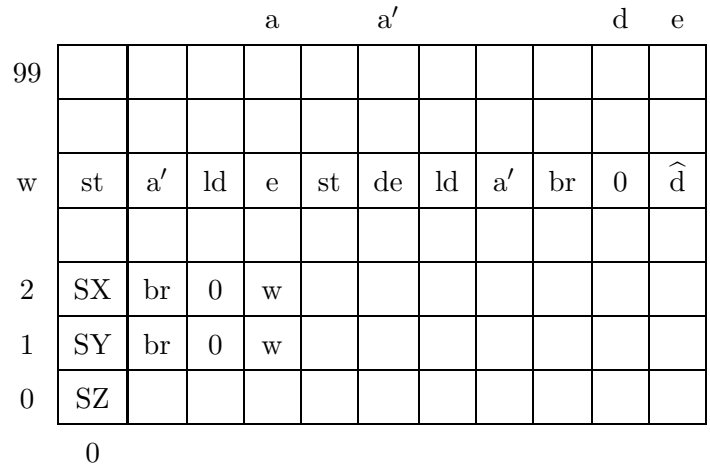


Fig. 4. CSM description of a while loop. Execution begins at location $(0, 2)$.

and compiled to a CSM as shown in Fig. 4. In this CSM, **e** specifies the number of remaining iterations in unary and is represented by a unary stack image. A second address **d**, unused by the statements in the body of the loop, holds the value popped from **e** and must be positioned immediately to the left of **e**. Address **a′** is used to store and restore the contents of address **a** before and after, respectively, decrementing the loop counter **e**. The fragment $\boxed{\text{br}\,|\,0\,|\,\widehat{\text{d}}}$ is shorthand for a piece of indirect addressing code, and means "branch to the address at the intersection of column 0 and the row specified by the image at address **d**".

The while routine in Fig. 4 has TIME complexity $6 + i(s + 6)$, constant GRID complexity, SPATIALRES complexity $\max(2^i, R_\text{s})$. Here $i \in \mathbb{N}$ is the number of times the body of the while loop is executed, $s \in \mathbb{N}$ is the number of operations in the body of the while loop, and finally $R_\text{s} \in \mathbb{N}$ is the minimum SPATIALRES required during execution of the body of the while loop. The value for each of AMPLRES, PHASERES and DYRANGE is at least constant, and possibly greater, depending on the operations performed and images accessed in the body of the while routine. Finally, the FREQ complexity is $O(\max(2^i, R_s))$.

3 Computability results

In this section we prove the CSM can simulate ARNNs, which are neural networks that compute over the set of real numbers. As an immediate corollary our model can decide the membership problem for any language $L \subseteq \Sigma^+$, if we allow aribtrary real values as inputs.

3.1 Boolean circuits and ARNNs

Informally, a Boolean circuit, or simply a circuit, is a finite directed acyclic graph where each node is an element of one of the following three sets: $\{\wedge, \vee, \neg\}$ (called *gates*, with respective in-degrees of 2,2,1), $\{x_1, \ldots, x_n\}$ ($x_i \in \{0,1\}$, *inputs*, in-degree 0), $\{0,1\}$ (*constants*, in-degree 0). A circuit family is a set of circuits $C = \{c_n : n \in \mathbb{N}\}$. A language $L \subseteq \Sigma^*$ is decided by the circuit family C_L if the characteristic function of the language $L \cap \{0,1\}^n$ is computed by c_n, for each $n \in \mathbb{N}$. It is possible to encode a circuit as a finite symbol sequence, and a circuit family by an infinite symbol sequence. When the circuits are of exponential size (with respect to input word length and where circuit size is the number gates in a circuit), for each $L \subseteq \Sigma^*$ there exists a circuit family to decide the membership problem for L. For a more thorough introduction to (nonuniform) circuits we refer the reader to [18].

ARNNs are finite size feedback first-order neural networks with real weights [8,9]. The state of each neuron at time $t+1$ is given by an update equation of the form

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right) \quad , \quad i = 1, \ldots, N \tag{11}$$

where N is the number of neurons, M is the number of inputs, $x_j(t) \in \mathbb{R}$ are the states of the neurons at time t, $u_j(t) \in \Sigma^+$ are the inputs at time t, and $a_{ij}, b_{ij}, c_i \in \mathbb{R}$ are the weights. An ARNN update equation is a function of discrete time $t = 1, 2, 3, \ldots$. The network's weights, states, and inputs are often written in matrix notation as A, B and c, $x(t)$, and $u(t)$, respectively. The function σ is defined as

$$\sigma(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } 0 \leq x \leq 1 \\ 1, & \text{if } x > 1 \ . \end{cases}$$

A subset P of the N neurons, $P = \{x_{k_1}, \ldots, x_{k_p}\}$, $P \subseteq \{x_1, \ldots, x_N\}$, are called the p output neurons. The output from an ARNN computation is defined as the states $\{x_{k_1}(t), \ldots, x_{k_p}(t)\}$ of these p neurons over time $t = 1, 2, 3, \ldots$.

3.2 Formal net deciding language membership

ARNN input/output (I/O) mappings can be defined in many ways [8]. In this paper we give a CSM that simulates the general form ARNN which has the

update equation given by Eq. (11). We also present a CSM that simulates a specific type of ARNN called a formal net [8]. Formal nets are ARNNs that decide the language membership problem and have the following I/O encodings. A formal net has two binary input lines, called the input data line (D) and the input validation line (V), respectively. If D is *active* at a given time t then $D(t) \in \Sigma$, otherwise $D(t) = 0$. $V(t) = 1$ when D is active, and $V(t) = 0$ thereafter (when D is deactivated it never again becomes active). An input to a formal net at time t has the form $u(t) = (D(t), V(t)) \in \Sigma^2$. The input word $w = w_1 \dots w_k \in \Sigma^+$ where $w_i \in \Sigma, 1 \leq i \leq k$, is represented by $u_w(t) = (D_w(t), V_w(t)), t \in \mathbb{N}$, where

$$D_w(t) = \begin{cases} w_t & \text{if } t = 1, \dots, k \\ 0 & \text{otherwise} \end{cases}, \quad V_w(t) = \begin{cases} 1 & \text{if } t = 1, \dots, k \\ 0 & \text{otherwise} \end{cases}.$$

A formal net has two output neurons $O_d, O_v \in \{x_1, \dots, x_N\}$, called the output data line and output validation line, respectively. Given a formal net $\mathcal{F}$ with an input word w and initial state $x_i(1) = 0$, $1 \leq i \leq N$, w is *classified* in time τ if the output sequences have the form

$$O_d = 0^{\tau-1}\psi_w 0^{\boldsymbol{\omega}}, \quad O_v = 0^{\tau-1}10^{\boldsymbol{\omega}},$$

where $\psi_w \in \Sigma$ and $\boldsymbol{\omega} = |\mathbb{N}|$. If $\psi_w = 1$ then w is accepted, if $\psi_w = 0$ then w is rejected. We now give a definition of deciding language membership by ARNN (from [8]). Let $\boldsymbol{T} : \mathbb{N} \to \mathbb{N}$ be a total function.

Definition 11 (Formal net deciding language membership) *The membership problem for the language $L \subseteq \Sigma^+$ is decided in time $\boldsymbol{T}$ by the formal net $\mathcal{F}$ provided that each word $w \in \Sigma^+$ is classified in time $\tau \leq \boldsymbol{T}(|w|)$ and $\psi_w = 1$ if $w \in L$ and $\psi_w = 0$ if $w \notin L$.*

In [8], Siegelmann and Sontag prove that for each language $L \subseteq \Sigma^+$ there exists a formal net $\mathcal{F}_L$ to decide the membership problem for L, hence proving the ARNN model to be computationally more powerful than the Turing machine model. $\mathcal{F}_L$ contains one real weight. This weight encodes the (nonuniform) circuit family C_L that decides L. Let $S_{C_L} : \mathbb{N} \to \mathbb{N}$ be the size of C_L. For a given input word $w \in \Sigma^+$, $\mathcal{F}_L$ retrieves the encoding of circuit $c_{|w|}$ from its real weight and simulates this encoded circuit on input w to decide membership in L, in time $\boldsymbol{T}(|w|) = O(|w|[S_{C_L}(|w|)]^2)$. Given polynomial time, formal nets decide the nonuniform language class P/poly. Given exponential time, formal nets decide the membership problem for all languages.

3.3 Statement of main result

Theorem 1 *There exists a CSM $\mathcal{M}$ such that for each ARNN $\mathcal{A}$, $\mathcal{M}$ computes $\mathcal{A}$'s I/O map, using our ARNN I/O representation.*

PROOF. The proof is provided by the ARNN simulation program for the CSM given in Fig. 5. The simulation is written in a convenient shorthand notation. The expansions into sequences of atomic operations are given in Fig. 6. The ARNN I/O representation is given is Sect. 3.4. The simulation program is explained in Sects. 3.5 through 3.7. A computational complexity analysis of the simulation program is given in Sect. 3.8. □

3.4 ARNN representation

As a convenient notation we let $\overline{\kappa}$ be the image representation of κ. We now give the I/O representation used in Theorem 1. The inputs to $\mathcal{M}$ fall into three categories: inputs that represent $\mathcal{A}$, inputs that represent $\mathcal{A}$'s input, and some constant inputs. Recall that our representation of matrices by images was defined in Def. 9 and illustrated in Fig. 3(f).

The ARNN weight matrices A, B, and c are represented by $N \times N$, $N \times M$, and $N \times 1$ matrix images $\overline{A}$, $\overline{B}$, and $\overline{c}$, respectively. The state vector x is represented by a $1 \times N$ matrix image $\overline{x}$. The set of output states P are represented by the image $\overline{P}$ (described below). The values $N-1$ and $M-1$ need to be given as input to the simulator in order to bound the loops. They are represented by unary stack images $\overline{N-1}$ and $\overline{M-1}$, representing the unary words $1^{(N-1)}$ and $1^{(M-1)}$, respectively. These seven input images define the ARNN $\mathcal{A}$. The constant images $f(x,y) = 0$ and $f(x,y) = 1$, denoted $\mathbf{0}$ and $\mathbf{1}$, respectively, are also given as input. Images $\mathbf{0}$ and $\mathbf{1}$ are used to parameterise ρ (see Lemma 2 below).

For an ARNN timestep t, the ARNN input vector $u(t)$ is represented by a $1 \times M$ matrix image $\overline{u}$. In an initial configuration of our simulation program we assume an input stack image I represents all input vectors $u(t)$ for all $t = 1, 2, 3, \ldots$. At an ARNN timestep t, the top element of stack image I is a $1 \times M$ matrix image representing the input vector $u(t)$.

The p output neurons are represented by a $1 \times N$ matrix image $\overline{P}$. We use $\overline{P}$ to extract our representation of the p output states from the N neuron states represented by $\overline{x}$. The image $\overline{x}$ contains N (possibly nonzero) values at specific coordinates defined in Def. 9. p of these values represent the p ARNN output states and have coordinates $(x_1, y_1), \ldots, (x_p, y_p)$ in $\overline{x}$. In the

image $\overline{P}$, each of the coordinates $(x_1, y_1), \ldots, (x_p, y_p)$ has value 1 and all other coordinates in $\overline{P}$ have value 0. We multiply $\overline{x}$ by $\overline{P}$. This image multiplication results in an output image o that has our representation of the p ARNN outputs at the coordinates $(x_1, y_1), \ldots, (x_p, y_p)$. Image o has value 0 at all other coordinates. The simulator then pushes o to an output stack image O. This output extraction process is carried out at the end of each simulated state update.

3.5 ARNN simulation overview

From the neuron state update equation Eq. (11), each $x_j(t)$ is a component of the state vector $x(t)$. From $x(t)$ we define the $N \times N$ matrix $X(t)$ where each row of $X(t)$ is the vector $x(t)$. Therefore $X(t)$ has components $x_{ij}(t)$, and for each $j \in \{1, \ldots N\}$ it is the case that $x_{ij} = x_{i'j}$, $\forall i, i' \in \{1, \ldots N\}$. From $u(t)$ we define the $N \times M$ matrix $U(t)$ where each row of $U(t)$ is the vector $u(t)$. Therefore $U(t)$ has components $u_{ij}(t)$, and for each $j \in \{1, \ldots M\}$ it is the case that $u_{ij} = u_{i'j}$, $\forall i, i' \in \{1, \ldots N\}$. Using $X(t)$ and $U(t)$ we rewrite Eq. (11) as

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_{ij}(t) + \sum_{j=1}^{M} b_{ij}u_{ij}(t) + c_i\right) \quad , \quad i = 1, \ldots, N \ . \tag{12}$$

In the simulation we generate $N \times N$ and $N \times M$ matrix images $\overline{X}$ and $\overline{U}$ representing $X(t)$ and $U(t)$, respectively. We then simulate the affine combination in Eq. (12) using our model's $+$ and $\cdot$ operators. We use the CSM's amplitude filtering operation ρ to simulate the ARNN σ function.

Lemma 2 *The CSM operation ρ simulates $\sigma(x)$ in constant* TIME.

PROOF. From the definition of ρ in Eq. (6), we set $z_{\text{l}}(x, y) = 0$ (denoted $\mathbf{0}$) and $z_{\text{u}}(x, y) = 1$ (denoted $\mathbf{1}$) to give

$$\rho(f(x,y), \mathbf{0}, \mathbf{1}) = \begin{cases} 0, & \text{if } |f(x,y)| < 0 \\ |f(x,y)|, & \text{if } 0 \leq |f(x,y)| \leq 1 \\ 1, & \text{if } |f(x,y)| > 1 \ . \end{cases}$$

Using our representation of ARNN state values by images, $\rho(\overline{x}, \mathbf{0}, \mathbf{1})$ simulates $\sigma(x)$. Also, ρ is a CSM operation hence simulating $\sigma(x)$ requires constant TIME. $\square$

For brevity and ease of understanding we outline our simulation algorithm in a high-level pseudocode, followed by an explanation of each algorithm step.

(i) $\overline{u} := I.\text{pop}()$

(ii) $\overline{X} :=$ push $\overline{x}$ onto itself vertically $N-1$ times

(iii) $\overline{AX} := \overline{A} \cdot \overline{X}$

(iv) $\Sigma\overline{AX} := \Sigma_{i=1}^{N} \left(\overline{AX}.\text{pop}_i() \right)$

(v) $\overline{U} :=$ push $\overline{u}$ onto itself vertically $N-1$ times

(vi) $\overline{BU} := \overline{B} \cdot \overline{U}$

(vii) $\Sigma\overline{BU} := \Sigma_{i=1}^{M} \left(\overline{BU}.\text{pop}_i() \right)$

(viii) affine-comb $:= \Sigma\overline{AX} + \Sigma\overline{BU} + \overline{c}$

(ix) $\overline{x}' := \rho(\text{affine-comb}, \mathbf{0}, \mathbf{1})$

(x) $\overline{x} := (\overline{x}')^T$

(xi) $O.\text{push}(\overline{P} \cdot \overline{x})$

(xii) goto step (i)

In step (i) we pop an image from input stack I and call the popped image $\overline{u}$. Image $\overline{u}$ is a $1 \times M$ matrix image representing the ARNN's inputs at some time t. In step (ii) we generate the $N \times N$ matrix image $\overline{X}$ by vertically pushing $N-1$ identical copies of $\overline{x}$ onto a copy of $\overline{x}$. In step (iii), $\overline{X}$ is point by point multiplied by matrix image $\overline{A}$. This single multiplication step efficiently simulates (in constant TIME) the matrix multiplication $a_{ij}x_j$ for all $i, j \in \{1, \ldots, N\}$ (as described in Sect. 3.5). Step (iv) simulates the ARNN summation $\sum_{j=1}^{N} a_{ij}x_j$ (in linear TIME). Each of the N columns of $\overline{AX}$ are popped and added (using the + operation), one at a time, to the previous popped image.

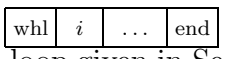
In step (v) we are treating $\overline{u}$ in a similar way to our treatment of $\overline{x}$ in step (ii). In step (vi) we effect $\overline{B} \cdot \overline{U}$, efficiently simulating (in constant TIME) the multiplication $b_{ij}u_j$ for all $i \in \{1, \ldots, N\}, j \in \{1, \ldots, M\}$. Step (vii) simulates the ARNN summation $\sum_{j=1}^{M} b_{ij}u_j$ using the same technique used in step (iv).

In step (viii) we simulate the addition of the three terms in the ARNN affine combination. In our simulator this addition is effected in two simple image addition steps. In step (ix) we simulate the ARNN's σ function by amplitude filtering using the CSM's ρ function with the lower and upper threshold images

$(\mathbf{0}, \mathbf{1})$ (as given by Lemma 2). The resulting $N \times 1$ matrix image is transformed into a $1 \times N$ matrix image (we simply transpose the represented vector) in step (x). We call the result of this amplitude filtering and transformation $\overline{x}$; it represents the ARNN state vector $x(t+1)$. In step (xi) we multiply $\overline{x}$ by the output mask $\overline{P}$ (as described in Sect. 3.4). The result, which represents the ARNN output at time $t+1$ is then pushed to the output stack O. The final step in our algorithm sends us back to step (i). Notice that our algorithm never halts as ARNN computations are defined for time $t = 1, 2, 3, \ldots$.

3.7 Explanation of Figs. 5 and 6

The ARNN simulation with our model is shown in Fig. 5. The numerals (i)–(xii) are present to assist the reader in understanding the program; they correspond to steps (i)–(xii) in the high-level pseudocode in Sect. 3.6. In our ARNN simulator program addresses are written in a shorthand notation that are expanded using Fig. 6. Before the simulator begins executing, a simple preprocessor or compiler could be used to update the shorthand addresses to the standard long-form notation.

Addresses t_1, t_2, and t_3 are used as temporary storage locations during a run of the simulator [note: address t_3 is located at grid coordinates $(11, 14)$]. In the simulator our $\overline{\kappa}$ notation not only denotes the image representation of κ, but also acts as an address identifier for the image representing κ. Addresses $\overline{x}$ and $\overline{u}$ are used to store our representation of the neurons' states and inputs, respectively, during a computation. The temporary storage addresses $\Sigma\overline{AX}$ and $\Sigma\overline{BU}$ are used to store the results of steps (iv) and (vii), respectively. Addresses $\overline{N-1}$ and $\overline{M-1}$ store our representation of the dimensions of x and u, respectively (necessary for bounding the while loops). The address identifiers $\overline{A}$, $\overline{B}$, and $\overline{c}$ store the image representation of the corresponding ARNN matrices, and $\overline{P}$ stores our mask for extracting the p output states from the N neuron states, as described in Sect. 3.4. Code fragments of the form 

are shorthand for code to initialise and implement the while loop given in Sect. 2.8. The instructions between i and `end` are executed i times. The notation $\widehat{\mathbf{0}}$ is shorthand for the "image at address **0**".

At ARNN timestep t, our representation of the ARNN input $u(t)$ is at the top of the input stack image I. This input is popped off the stack and placed in address $\overline{u}$. The computation then proceeds as described by the high-level pseudocode algorithm in Sect. 3.6. The output memory address O stores the sequence of outputs as described in Sect. 3.4. Program execution begins at well-known address **sta** and proceeds according to the rules for our model's programming language defined in Def. 4 and explained in Fig. 2.

Fig. 5. CSM $\mathcal{M}$ that simulates any ARNN $\mathcal{A}$. The simulator is written in a convenient shorthand notation, (see Fig. 6 for the expansions into sequences of atomic operations). The simulation program is explained in Sect. 3.7.

(a)

I	$\rightarrow$	16	16	15	15
t_1a	$\rightarrow$	10	11	15	15
t_1	$\rightarrow$	10	10	15	15
$\overline{u}$	$\rightarrow$	6	6	15	15
$\overline{x}$	$\rightarrow$	13	13	0	0
t_3	$\rightarrow$	11	11	14	14
at_3	$\rightarrow$	11	11	14	15
b	$\rightarrow$	12	12	15	15
$\overline{A}$	$\rightarrow$	16	16	0	0
ab	$\rightarrow$	11	12	15	15
bt_2	$\rightarrow$	12	13	15	15
t_2	$\rightarrow$	13	13	15	15
$\Sigma\overline{AX}$	$\rightarrow$	7	7	15	15
$\overline{B}$	$\rightarrow$	17	17	0	0
$\Sigma\overline{BU}$	$\rightarrow$	8	8	15	15
$\overline{c}$	$\rightarrow$	18	18	0	0
$\overline{P}$	$\rightarrow$	19	19	0	0
O	$\rightarrow$	15	15	15	15

(b)

↲	$\rightarrow$	br	0	δ–1

(c)

whl	$\overline{N–1}$	...	end
whl	$\overline{M–1}$	...	end

Fig. 6. Time-saving shorthand notation used in the simulator in Fig. 5: (a) shows shorthand addresses, (b) branch to beginning of row $\delta - 1$, where δ is the current row, and (c) expands to initialisation instructions and the while loop code given in Fig. 4.

3.8 Complexity analysis of simulation algorithm

The following is a worst case analysis of the ARNN simulation algorithm. If the ARNN being simulated is defined for time $t = 1, 2, 3, \ldots$, has M as the length of the input vector $u(t)$ and has N neurons, and k is the number of stack image elements used to represent the active input to our simulator, then

$\mathcal{M}$ requires TIME T linear in N, M, and t, and independent of k,

$$T(N, M, t, k) = (49N + 11M + 42)t + 1 \ .$$

$\mathcal{M}$ requires constant GRID, and exponential SPATIALRES

$$R_{\mathrm{S}}(N, M, t, k) = \max(2^{(k+M-1)}, 2^{(2N-2)}, 2^{(N+M-2)}, 2^{(t+N-1)}) \ .$$

$\mathcal{M}$ requires infinite AMPLRES in order to represent real-valued ARNN weight matrices. $\mathcal{M}$ requires constant PHASERES of 1 and linear DYRANGE equal to $\max(1, \max_{A_{in}})$, where $\max_{A_{in}}$ is the maximum amplitude value of all the input images. Finally, the FREQ complexity is $O(R_{\mathrm{S}}(\cdot))$, where $R_{\mathrm{S}}(\cdot)$ is the SPATIALRES complexity of $\mathcal{M}$.

3.9 CSM deciding language membership by formal net simulation

Corollary 3 *There exists a CSM $\mathcal{D}$ that decides the membership problem for each $L \subseteq \Sigma^+$.*

PROOF. The proof relies on two facts. Firstly, for each $L \subseteq \Sigma^+$ there exists a formal net $\mathcal{F}_L$ that decides the membership problem for L [8]. Secondly there exists a CSM $\mathcal{M}$ that simulates each ARNN (Theorem 1). CSM $\mathcal{D}$ is given in Fig. 7, its I/O format and a brief complexity analysis are given in the remainder of the current section. □

To decide membership of $w \in \Sigma^+$ in L, $\mathcal{D}$ simulates formal net $\mathcal{F}_L$ on input w. $\mathcal{D}$ is a language membership deciding CSM, hence $\mathcal{D}$'s I/O format is consistent with Def. 10 (CSM deciding language membership). In Fig. 7, rows 2 to 13 are exactly rows 2 to 13 from CSM $\mathcal{M}$ in Fig. 5, the remaining extra functionality is necessary to properly format the I/O. The shorthand notation follows the format given in Fig. 6. Given the problem instance of deciding membership of $w \in \Sigma^+$ in L, CSM $\mathcal{D}$ has thirteen input images and a single output image. Input images f_w and $f_{1^{|w|}}$ are the binary and unary stack image representations of the words w and $1^{|w|}$, respectively. Images **0** and **1** are the constant images $f(x, y) = 0$ and $f(x, y) = 1$, respectively. Formal net $\mathcal{F}_L$ is completely defined by the following seven input images: $\overline{A}, \overline{B}, \overline{c}, \overline{P}, \overline{x}, \overline{N-1}$, and $\overline{M-1}$. These images have the format described above in Sect. 3.4. When simulating a formal net the input images $\overline{M-1}$ and $\overline{x}$ are constant (as $M = 2$ and $x(1)$ is a vector of zeros). Images $\overline{O_d}$ and $\overline{O_v}$ are unary stack images representing the unary words 1^d and 1^v, respectively. Here d and v are the indices of the output data and output validation neurons, respectively, in the N vector of neurons.

	sta							$\overline{u}$	$\Sigma\overline{AX}$	$\Sigma\overline{BU}$		t_1	a	b	t_2			f_{ψ_w}	f_w	$f_{1^{\lvert w\rvert}}$	**0**	**1**
19	br	0	18																			
18	ld	f_w	st	b	ld	**0**	st	st	t_2	↲												
17	whl	$f_{1^{\lvert w\rvert}}$	ld	b	st	t_1a	st	st	b	ld	t_2	ld	t_1a	st	t_2	end	st	f_w	↲			
16	ld	f_w	st	t_1a	st	f_w	ld	ld	t_1	st	13	16	14	14	ld	14	17	14	14	↲		
15	st	b	ld	$f_{1^{\lvert w\rvert}}$	st	t_1a	st	st	$f_{1^{\lvert w\rvert}}$	ld	t_1	st	13	16	14	14						
14	ld	12	15	14	14	+	st	st	$\overline{u}$	br	0	13										
⋮																						
1	st	b	ld	$\overline{P}$	·	st	t_3	st	t_1	t_1	whl	$\overline{O_v}$	st	t_1a	end	ld	t_1	br	0	ⓐ		
f_1	ld	t_3	whl	$\overline{O_d}$	st	t_1a	end	ld	t_1	t_1	st	f_{ψ_w}	hlt									
f_0	br	0	16																			
	0	1	2	3	4	5	6	7	8	9 …		$\overline{O_d}$	$\overline{O_v}$	$\overline{x}$	$\overline{N-1}$	$\overline{M-1}$	$\overline{A}$	$\overline{B}$	$\overline{c}$	$\overline{P}$		

note: address t_3 is located at grid coordinates $(11, 18)$

Fig. 7. CSM $\mathcal{D}$ that decides language membership by ARNN simulation. Shorthand notation follows the format given in Fig. 6. Rows 2 to 13 are exactly rows 2 to 13 from CSM $\mathcal{M}$ in Fig. 5.

Images $\overline{O_d}$ and $\overline{O_v}$ are used to extract the output 'decision' of $\mathcal{F}_L$. There is one output image denoted f_{ψ_w}.

The following is a worst case analysis of CSM $\mathcal{D}$ simulating a formal net $\mathcal{F}_L$ that decides membership of language L in time $\boldsymbol{T}$. On input word $w \in \Sigma^+$, $\mathcal{F}_L$ decides if w is in L in t timesteps for some $t \leq \boldsymbol{T}(|w|)$. When deciding a language from the class P/poly, in the worst case the function $\boldsymbol{T}$ is polynomial in input word length. When deciding an arbitrary language, in the worst case $\boldsymbol{T}$ is exponential in input word length [8]. Let N, M, d, and v be as given above. CSM $\mathcal{D}$ requires linear TIME

$$T(N, M, \boldsymbol{T}(|w|), |w|, d, v) = (49N + 7v + 81)\boldsymbol{T}(|w|) + 12|w| + 7d + 24 \ ,$$

constant GRID, and exponential SPATIALRES

$$R_{\text{S}}(N, M, \boldsymbol{T}(|w|), |w|, d, v) = \max\left(2^{|w|}, 2^{(2N-2)}\right) \ ,$$

to decide membership of w in L. $\mathcal{D}$ requires infinite AMPLRES, as one of $\mathcal{F}_L$'s weight matrices contains a real-valued weight that encodes the (possibly nonuniform) circuit family C_L. $\mathcal{D}$ requires constant PHASERES of 1 and linear DYRANGE equal to $\max(1, \max_{A_{in}})$ where $\max_{A_{in}}$ is the maximum amplitude value of all the input images. Finally, the FREQ complexity is $O(R_{\text{S}}(\cdot))$, where $R_{\text{S}}(\cdot)$ is the SPATIALRES complexity of $\mathcal{D}$. By way of formal net simulation the CSM decides the membership problem for any language $L \subseteq \Sigma^+$ with these complexity bounds.

4 Unordered search

Sorting and searching [19] provide standard challenges to computer scientists in the field of algorithms, computation, and complexity. In this paper we focus on a binary search algorithm. With our model this algorithm can be applied to unordered lists. Consider an unordered list of n elements. For a given property P, the list could be represented by an n-tuple of bits, where the bit key for each element denotes whether or not that element satisfies P. If, for a particular P, only one element in the list satisfies P, the problem of finding its index becomes one of searching an unordered binary list for a single 1. The problem is defined formally as follows.

Definition 12 (Needle in haystack problem) *Let $L = \{w : w \in 0^*10^*\}$. Let $w \in L$ be written as $w = w_0 w_1 \ldots w_{n-1}$ where $w_i \in \{0, 1\}$. Given such a w, the needle in haystack (NIH) problem asks what is the index of the symbol 1 in w. The solution to NIH for a given w is the index i, expressed in binary, where $w_i = 1$.*

```
(8,99)  procedure search(i1, i2)
(0,3)     e := i2
(4,3)     c := f_0
(0,w)     while (e.pop() = f_1)
(0,1)       rescale i1 over both image a and image b
(0,2)       FT, square, and FT image a
(8,2)       if (a = f_1)
(8,1)         i1 := LHS of i1
(14,1)        c.push(f_0)
(8,2)       else /* a = f_0 */
(8,0)         i1 := RHS of i1
(16,0)        c.push(f_1)
            end if
          end while
(0,0)     a := c
(2,0)   end procedure
```

Fig. 8. Pseudocode algorithm to search for a single 1 in a list otherwise populated with 0s. Line numbers give addresses of the corresponding piece of code in the CSM machine in Fig. 9. Images f_0 and f_1 were defined in Def. 5.

This problem was posed by Grover in [20]. His quantum computer algorithm requires $O(\sqrt{n})$ comparison operations on average. Bennett *et al.* [21] have shown the work of Grover is optimal up to a multiplicative constant, and that in fact any quantum mechanical system will require $\Omega(\sqrt{n})$ comparisons. Algorithms for conventional models of computation require $\Theta(n)$ comparisons in the worst case to solve the problem. We present an algorithm that requires $\Theta(\log_2 n)$ comparisons, in the worst case, with a model of computation that has promising future implementation prospects.

Our search algorithm is quite simple. A single bright point is somewhere in an otherwise dark image. If we block one half of the image we can tell in a single step if the other half contains the bright point or not. This forms the basis of a binary search algorithm to determine the precise location of the bright point.

Before presenting a CSM instance of the algorithm, we give a pseudocode version (see Fig. 8). This pseudocode algorithm consists of a single loop. It is formatted to conform to the iteration construct presented in Sect. 2.8. The algorithm takes two arguments, one is a list image and the other is a stack image. (Stack images and list images were defined in Defs. 6 and 7.) The first argument, **i1**, is a binary list image representing w. We assume that n is a power of 2. The second argument, **i2**, is a unary stack image of length $\log_2 n$, and is used to bound the iteration of the algorithm's loop. The algorithm uses address **c** as it constructs, one binary symbol image at a time, a binary stack

image of length $\log_2 n$. At halt, the binary stack image at address **c** represents the index i of the 1 in w. This index is returned through address **a** when the algorithm terminates. To aid the reader, each line of the pseudocode algorithm in Fig. 8 is prepended with a pair of coordinates that relate the pseudocode to the beginning of the corresponding code in the CSM version of the algorithm. The CSM version of the algorithm is given in Fig. 9.

Definition 13 (Comparison in CSM) *A comparison in a CSM computation is defined as a conditional branching instruction.*

Theorem 4 *There exists a CSM that solves NIH in* $\Theta(\log_2 n)$ *comparisons for a list of length* n*, where* $n = 2^k, k \in \mathbb{N}, k \geq 1$.

PROOF. The proof is provided by the algorithm in Fig. 9. Correctness: The correctness is most easily seen from the pseudocode algorithm in Fig. 8 and the following inductive argument. (Figure 8 contains a mapping from pseudocode statements to the CSM statements of Fig. 9.) The two inputs are a binary list image representation of w (image **i1**) and a unary stack image of length $\log_2 n$ (image **i2**). During the first iteration of the loop, a single image f_1 is popped from **i2**, and **i1** is divided equally into two list images (a left-hand image and a right-hand image). The nonzero peak (representing the 1 in w) will be either in the left-hand list image or the right-hand list image. In order to determine which list image contains the peak in a constant number of steps, the left-hand list image is transformed such that its centre will contain a weighted sum of all of the values over the whole list image. Effectively, the list image is transformed to an element of the binary image set $\{f_0, f_1\}$ (see Def. 5). If the left-hand list image is transformed to f_1 (if the centre of this transformed list image contains a nonzero amplitude) then the left-hand list image contained the peak. In this case, the right-hand image is discarded, and f_0 is pushed onto stack image **c**. Otherwise, the right-hand list image contained the peak, the left-hand list image is discarded, and f_1 is pushed onto **c**. After the first iteration of the loop, the most significant bit of the solution to the problem is represented by the top of stack image **c**, and **i1** has been reduced to half its length. For the second iteration of the loop, a second image f_1 is popped from counter **i2**, the list image is divided in two, and the appropriate half discarded. The algorithm continues in this binary search fashion until the image popped from **i2** is f_0. Image **c** is copied into **a** and the algorithm halts. At halt, the index (in binary) of the 1 in w is represented by the stack image in **a** of length $\log_2 n$.
Complexity: The loop in the algorithm makes exactly $\log_2 n$ iterations, corresponding to $\log_2 n + 1$ evaluations of the loop guard. Inside the loop, there is a single comparison. In total, the CSM algorithm makes $2\log_2 n + 1$ comparisons to transform the binary list image representation of w (of length n) into the binary stack image representation of index i. □

Theorem 4 states computational complexity in terms of number of comparisons, so that the result can be directly compared with the lower bound analyses from classical algorithm theory and quantum complexity theory. This simplification hides linear SPATIALRES and FREQ overheads, as the following corollary shows.

Corollary 5 *There exists a CSM that solves NIH in* $\Theta(\log_2 n)$ TIME*, constant* GRID*,* $\Theta(n)$ SPATIALRES*, constant* DYRANGE*, constant* AMPLRES*, constant* PHASERES*, and* $O(n)$ FREQ*, for a list of length* n*, where* $n = 2^k, k \in \mathbb{N}, k \geq 1$*.*

PROOF. The proof is provided by the algorithm in Fig. 9. Correctness: The correctness follows from Theorem 4.
Complexity: Each iteration of the loop requires constant TIME. The total TIME from problem instance to solution is $23 \log_2 n + 11$. From Sect. 2.3, all CSMs require constant GRID. The maximum length required of any stack image during the computation is $\log_2 n + 1$ (for image **c**). This results in SPATIALRES complexity of $2n$. The CSM requires constant AMPLRES, PHASERES and DYRANGE, because all input values will be binary. Even after the FT operation, only the binarised zero frequency component is relevant to the computation so we do not need to preserve the amplitudes or phases of any of the other spatial frequency components. Finally, the CSM requires an upper bound of $O(n)$ FREQ to accompany the linear SPATIALRES (assuming traditional diffraction limited resolution techniques). □

5 Conclusion

We have presented the CSM, an analog model of computation inspired by the field of optical information processing. We have given some insight into the computational power of the CSM by proving it can simulate ARNNs (this simulation includes linear time matrix multiplication), and by giving a $\Theta(\log_2 n)$ unordered search algorithm that does not make use of arbitrary real/complex constants. The model does not support arbitrary equality testing of images, and so in this sense is closer in spirit to models found in [8,22,23] than (say) the Blum, Shub, and Smale model [24,25]. However allowing arbitrary real/complex constants gives the model a lot of computational power. For future work it would be interesting to classify the computational power of discrete variants of the CSM.

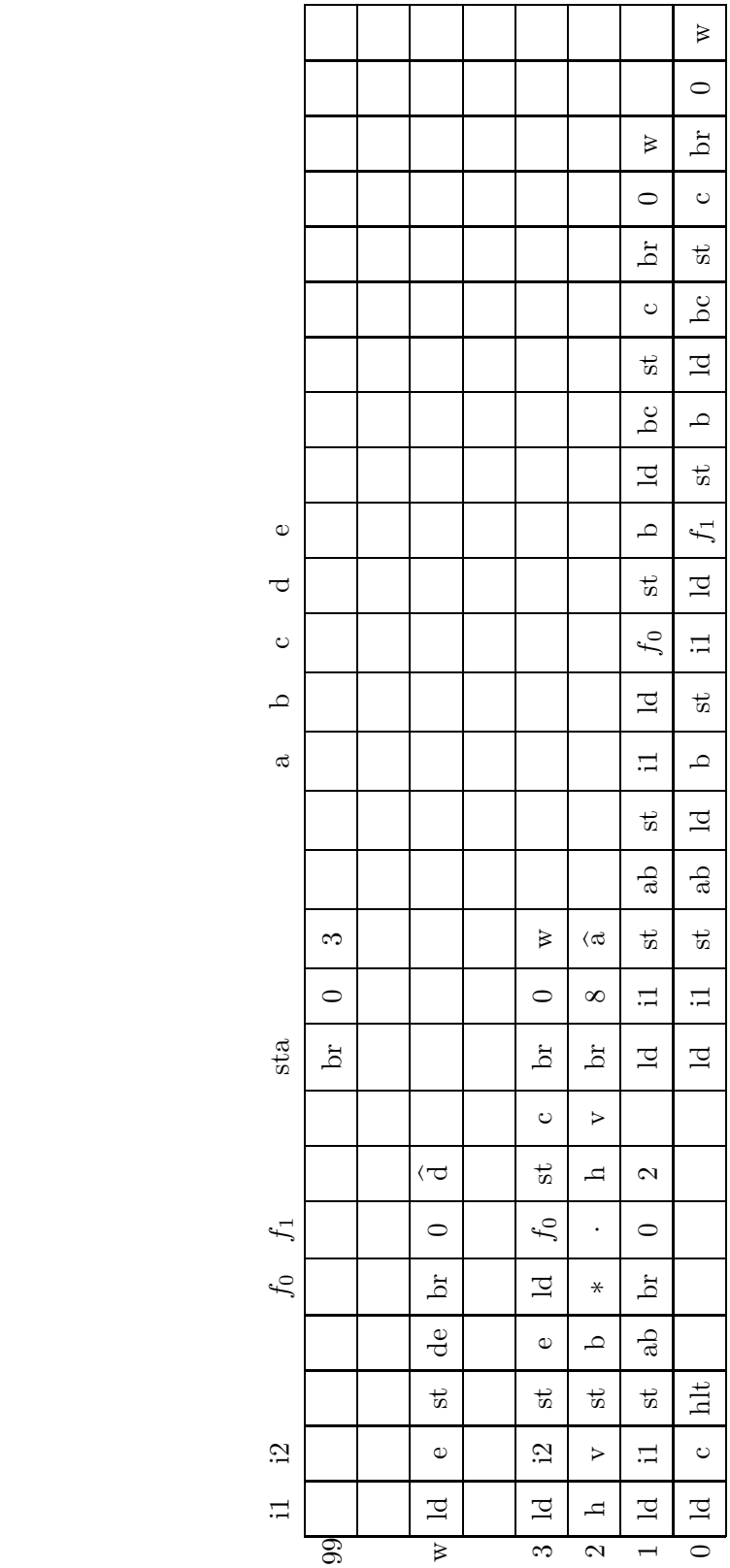


Fig. 9. CSM to perform a $\Theta(\log_2 n)$ search on an unordered binary list.

Acknowledgments

We wish to acknowledge Maurice Margenstern and the anonymous reviewers for helpful advice.

References

[1] A. VanderLugt, Optical Signal Processing, Wiley Series in Pure and Applied Optics, Wiley, New York, 1992.

[2] J. W. Goodman, Introduction to Fourier Optics, 2nd Edition, McGraw-Hill, New York, 1996.

[3] A. VanderLugt, Signal detection by complex spatial filtering, IEEE Transactions on Information Theory IT-10 (1964) 139–145.

[4] C. S. Weaver, J. W. Goodman, A technique for optically convolving two functions, Applied Optics 5 (7) (1966) 1248–1249.

[5] T. J. Naughton, A model of computation for Fourier optical processors, in: R. A. Lessard, T. Galstian (Eds.), Optics in Computing 2000, Proc. SPIE vol. 4089, Quebec, Canada, 2000, pp. 24–34.

[6] T. J. Naughton, D. Woods, On the computational power of a continuous-space optical model of computation, in: M. Margenstern, Y. Rogozhin (Eds.), Machines, Computations and Universality: Third International Conference, Vol. 2055 of Lecture Notes in Computer Science, Springer, Chişinău, Moldova, 2001, pp. 288–299.

[7] K. Weihrauch, Computable Analysis: An Introduction, Texts in Theoretical Computer Science, Springer, Berlin, 2000.

[8] H. T. Siegelmann, E. D. Sontag, Analog computation via neural networks, Theoretical Computer Science 131 (2) (1994) 331–360.

[9] H. T. Siegelmann, Neural networks and analog computation: beyond the Turing limit, Progress in Theoretical Computer Science, Birkhäuser, Boston, 1999.

[10] F. T. S. Yu, T. Lu, X. Yang, D. A. Gregory, Optical neural network with pocket-sized liquid-crystal televisions, Optics Letters 15 (15) (1990) 863–865.

[11] T. Naughton, Z. Javadpour, J. Keating, M. Klíma, J. Rott, General-purpose acousto-optic connectionist processor, Optical Engineering 38 (7) (1999) 1170–1177.

[12] P.-Y. Wang, M. Saffman, Selecting optical patterns with spatial phase modulation, Optics Letters 24 (16) (1999) 1118–1120.

[13] F. S. Chen, J. T. LaMacchia, D. B. Fraser, Holographic storage in lithium niobate, Applied Physics Letters 13 (7) (1968) 223–225.

[14] A. Pu, R. F. Denkewalter, D. Psaltis, Real-time vehicle navigation using a holographic memory, Optical Engineering 36 (10) (1997) 2737–2746.

[15] J. W. Goodman, Operations achievable with coherent optical information processing systems, Proceedings of the IEEE 65 (1) (1977) 29–38.

[16] A. E. Chiou, Photorefractive phase-conjugate optics for image processing, trapping, and manipulation of microscopic objects, Proceedings of the IEEE 87 (12) (1999) 2074–2085.

[17] R. Rojas, Conditional branching is not necessary for universal computation in von Neumann computers, Journal of Universal Computer Science 2 (11) (1996) 756–768.

[18] J. L. Balcázar, J. Díaz, J. Gabarró, Structural Complexity, volumes I and II, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1988.

[19] D. Knuth, The Art of Computer Programming, Vol 3: Sorting and Searching, Addison-Wesley, 1973.

[20] L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proc. 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 212–219.

[21] C. H. Bennett, E. Bernstein, G. Brassard, U. Vazirani, Strengths and weaknesses of quantum computing, SIAM Journal on Computing 26 (5) (1997) 1510–1523.

[22] C. Moore, Recursion theory on the reals and continuous-time computation, Theoretical Computer Science 162 (1) (1996) 23–44.

[23] M. L. Campagnolo, Computational complexity of real valued recursive functions and analog circuits, Ph.D. thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa (2001).

[24] L. Blum, M. Shub, S. Smale, A theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, Bulletin of the American Mathematical Society 21 (1989) 1–46.

[25] L. Blum, F. Cucker, M. Shub, S. Smale, Complexity and real computation, Springer-Verlag, New York, 1997.