Chapter 5

Applications – Doing Stuff on the Machine

By Petruț Bogdan, Robert James, Gabriel Fonseca Guerra, Garibaldi Pineda García and Basabdatta Sen-Bhattacharya

DOI: 10.1561/9781680836530.ch5

Published in *SpiNNaker – A Spiking Neural Network Architecture* by S. Furber and P. Bogdan (eds.). 2020. ISBN 978-1-68083-596-0. E-ISBN 978-1-68083-653-0.

Suggested citation: Petruț Bogdan *et al.* 2020. "Applications – Doing Stuff on the Machine" in *SpiNNaker – A Spiking Neural Network Architecture*. Edited by S. Furber and P. Bogdan. pp. 129–162. Now Publishers. DOI: 10.1561/9781680836530.ch5.

now
the essence of knowledge

The Terminator's an infiltration unit. Part man, part machine.
Underneath, it's a hyperalloy combat chassis,
microprocessor-controlled, fully armored.
Very tough … But outside, it's living human tissue.

— The Terminator

The SpiNNaker machine is flexible in terms of the applications that it supports. In part, this flexibility is given by the comparative ease of use of the substrate, namely the ARM processors. A varied range of applications is also encouraged by the software stack maturity discussed in Chapter 4. Using these high-level collections of software, a variety of plasticity mechanisms have been implemented to support various learning applications.

The following sections will cover a wide range of topics. We begin by first presenting an art exhibit and SpiNNaker's place in it – we start light. Then, we present a suite of approaches to engineer SNNs for a variety of computer vision tasks. Progressing through this chapter we present a large-scale model of a cochlea (SpiNNak-Ear [120]). This application is only possible on SpiNNaker because of general-purpose nature of the CPUs and the software written to support such generic, graph-based applications. From sensing to decision making, we present a

model of a Basal Ganglia (BG), of course, simulated on SpiNNaker. Finally, we use SNNs as the method of solving constraint satisfaction problems.

5.1 Robot Art Project

We are a lab full of engineers. Art was as far away from our collective future projections for the platform as possible. So, once we were approached by Tove Kjellmark, a Swedish artist, with the idea for an exhibit involving humanoid robots and SpiNNaker, we immediately considered the issues and hurdles of such an attempt, not the least that of time and expectation management. The exhibition at the Manchester Art Gallery, named 'The Imitation Game' in honour of Alan Turing and his eponymous test, was to include several robotic pieces with the common theme of seeming intelligent in particular ways. The robotic entities present in the gallery would surely not pass Turing's test in any meaningful way, but that was not the plan anyway. To school children, laypeople and scientists alike, this was an artist's view at imitating life at the behavioural, albeit limited, level. At a basic level, these pieces would hint at the existence of something more than just Artificial Intelligence (AI). Tove Kjellmark would call it 'another nature', that is to say an elimination of the artificial boundaries between the technological, the mechanical and the natural. We would rather call it a conceptual step in a more important area of research, that of Artificial General Intelligence (AGI), as opposed to the narrow AI, nowadays present everywhere, the 'autistic savants' that tell you what objects you are looking at, what movies to watch next and what music to listen based on your listening habits.

Our involvement focused on the piece 'Talk' (pictured in Figure 5.1) that featured two robotic torsos sat cross-legged on comfortable chairs discussing a dream. They look at each other, gesture while talking, speak fluently and with appropriate cadence, sighs and pauses. If a human dares approach, they stop their conversation, turn their head to face the intruder to chastise them and wave them away.[1] Thus, SpiNNaker's task was to control the arms of the robots to perform realistic-looking arm movements in three regimes: idling, gesturing and silencing.

The focus of this undergraduate project was successful in revealing that SpiNNaker is capable of real-life, albeit impractical, applications. The individually packaged SpiNNaker boards would not be turned off for weeks at a time and would operate without flaw for over 7 hours a day for approximately 4 months in conjunction with the physical robots. As expected, maintenance visits to the Gallery would generally revolve around the robots or indeed the host computers, rather than any

1. Robotic art gallery video presentation https://youtu.be/GaqgkyAIRBg

Figure 5.1. Display in 'The Imitation Game' exhibition at the Manchester Art Gallery, 2016, celebrating Manchester becoming European City of Science. Artist: Tove Kjellmark; School of Computer Science, Manchester: Petruț Bogdan, Prof. Steve Furber, Dr. Dave Lester, Michael Hopkins; Manchester Art Gallery Exhibitions Intern: Mathew Bancroft; Mechatronics Division, KTH, Stockholm: Joel Schröder, Jacob Johansson, Daniel Ohlsson, Elif Toy, Erik Bergdahl, Freddi Haataja, Anders Åström, Victor Karlsson, Sandra Aidanpää; Furhat Robotics: Gabriel Skantze, Jonas Beskow, Dr Per Johansson.

SpiNNaker intervention. It would seem that SpiNNaker would indeed be suited to neurorobotics applications [209], as discussed previously.

5.1.1 Building Brains with Nengo and Some Bits and Pieces

Two small PCs were used to control the two robots: the primary PC completely controls one of the robots and the arms of the other, while the secondary PC operates only the head of the other robot. The two distributed instances of the Furhat controller communicate through the network at key moments advancing the scripted dialogue. The primary PC is also responsible for communicating with the glorified distance sensor embodied in a Microsoft Kinect sensor, as well as the two stand-alone SpiNNaker boards. Both PCs control the actuators in the robotic arms using classical control theory; some translation is required between SpiNNaker's

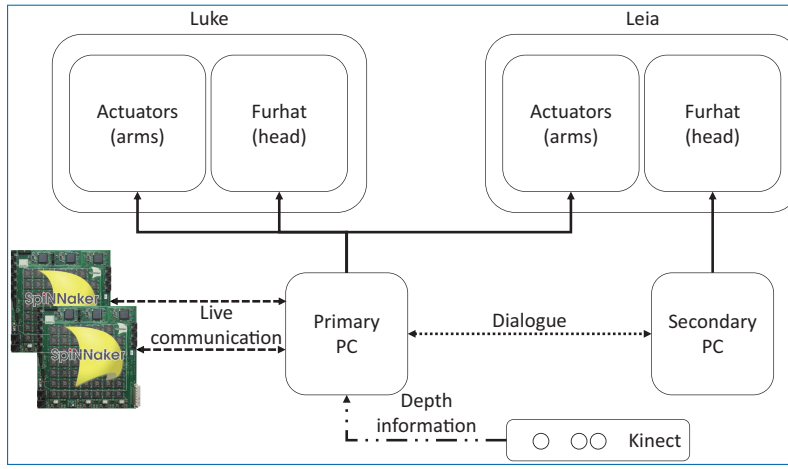

Figure 5.2. Hardware organisation diagram.

communication and these closed-loop control systems. Figure 5.2 reveals the flow of information involved in this project.

The previous chapter explained how SpiNNaker is usually controlled, using PyNN as a high-level network description language, viewing individual neurons as the main units of computation. Instead, here the Neural ENGineering Objects (Nengo) simulator bunches neurons together in ensembles (populations) and relies on their concerted activity to perform computation [53].

The way Nengo is built supports the implementation of a proportional-integral-derivative (PID) controller using a spiking neural substrate. A PID controller is a control loop feedback mechanism that continuously computes the error between the desired trajectory and the current position. The controller attempts to minimise the error as described by a weighted sum of a proportional, an integral and a derivative term. The proportional term accounts for moving towards the target at a rate dictated by the distance from it (cross track error). The derivative term considers the angle of the current trajectory compared to that of the desired trajectory (also called the cross track error rate), while the integral term is used to correct for accumulated errors that lead to a steady state error caused by, for example, external factors.

Consider the example of a driverless car positioned in a controlled environment with a trajectory precomputed for it to follow down the track in order to avoid some static obstacles. The goal is to try to follow the trajectory as closely as possible, so effects such as oscillations are not desired. In addition, the researchers at the facility have decided to see what would happen if at some point on the path they place a rock or pothole. They hope that the system would realise that it is drifting off course and apply a correcting turn. Figure 5.3 shows what this would look like in

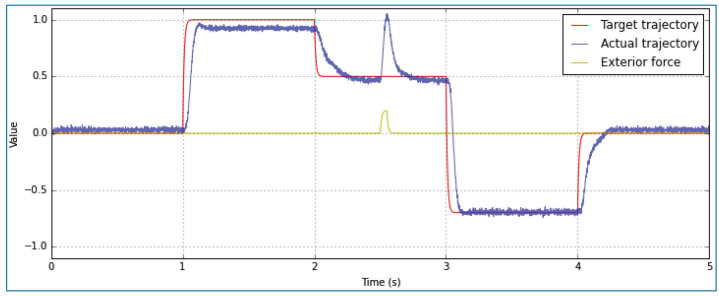

Figure 5.3. An example of trajectory following. In a real example, the trajectory would potentially not change so abruptly.

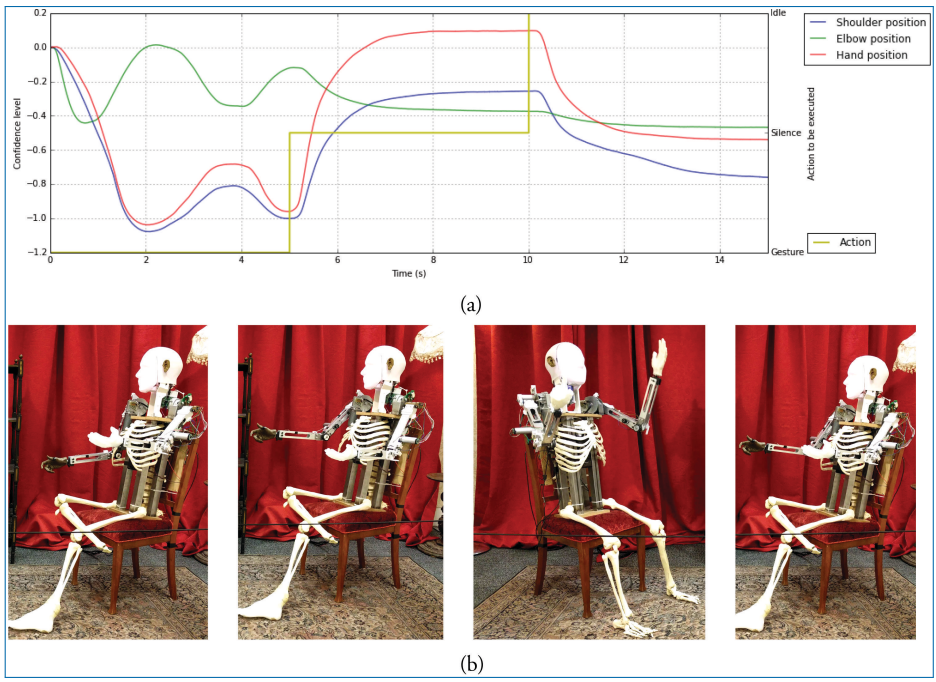

Figure 5.4. (a) A 15-second window of the operation of the control system running at the Manchester Art Gallery. This time period sees the robots going through all of the defined actions: gesture (the robot is talking), silence (the robot stopped talking to make a silencing gesture directed at an approaching visitor) and idle (the robot is not talking but listening to the other robot talk). (b) Robot poses corresponding to the Nengo simulation. The poses correspond to times 2, 4, 8 and 14.

Nengo. This is very similar to what can be done when controlling robot arm motors and servos.

Figure 5.4 shows the operation of one of the arms on a robot over a timespan of 15 seconds. During this time, the robot is issued three different commands in

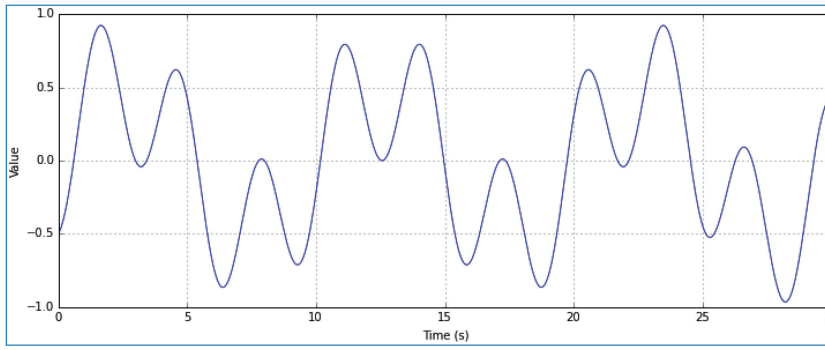

Figure 5.5. Gesturing movement of the robots computed as a function of time $f(t) = \frac{1}{2} * (sin(\frac{1}{1.6*t}) - cos(2 * t))$.

succession: gesture, silence and idle. While gesturing, the target position of each joint is given by a predetermined 'zero' or base position (hand-picked values that look natural in the physical exhibit) subtracted from a sinusoidal signal, namely the one in Figure 5.5. The incoming signal is transformed using a linear transformation for each joint individually to create a human like gesturing motion. Since the robots each has two arms, there is a dot product-based network inhibiting the arm that is not intended for use. Such arm selection is possible by creating a couple of predefined orthonormal vectors that represent the left and right directions. Based on the input direction vector for the system, a dot product is computed between it and the two previously mentioned bases so as to determine which direction is closest based on the angle. In the particular case where the vector is not significantly closer to any of the targets, the system accomplishes the desired action using both arms. The result of adding this level of control and inhibition is that the robot can now move one arm, or the other, or even both, thus allowing for more human mimetic behaviour.

When issued the action 'silence', the performing robot raises both lower arms into the air, in a defensive manner, signalled by external feedback from the head assembly, which turns to face the visitor and asks them to be silent. The action is achieved by inhibiting the neurons' spiking activity in the ensemble representing the 'zero' position and the 'sound' signal using the inhibiting output from an incorporated Basal Ganglia (BG) model. Analogously, idling is achieved by inhibiting the sound and silencing signals.

Because the exhibition took place in Manchester, no one else was around to maintain these robots, and we still had to experiment with realistic movement, we interacted with them for most of their stay at the Manchester Art Gallery. Most of these interactions took place during typical work hours, meaning that the gallery was usually populated by school children. It was surreal seeing the children interact

with the robots. They weren't allowed to touch them of course, although that did not prevent them from trying. All of this assumes that they managed to enter the room: the usual first reaction to seeing them was fear. Once I had talked to the children's teachers and assured them that it was safe in the room, they would flock inside to witness the two humanoids in discussion. There was always someone watching from the doorway, too apprehensive to approach these mechanical beings, which were, essentially, only superficially intelligent. Nobody knew what they were talking about, but they were all fascinated with their 'silencing' phases as these provided the most audience interaction. These groups rarely stopped to read the plaque describing the exhibit, but surely this was a success in and of itself: SpiNNaker managed to work flawlessly for the entire duration of the exhibit; the same could not be said about the actuators and 3D-printed parts which had a much harder time.

5.2 Computer Vision with Spiking Neurons

Computational emulation of biological vision has been of interest for decades [249]. State-of-the-art computer vision systems use traditional image sensors for their inputs that differ greatly from those present in biology. In particular, ganglion cells in the mammalian retina emit signals when sufficient change in light intensity is sensed. Biology has successfully made use of event-based computation in vision (and other senses), and we should aim for the same in machine vision.

5.2.1 Feature Extraction

An important step in computer vision is to extract features from the input image. In traditional computer vision, this usually involves applying operations (e.g. convolutions and integrals) to the whole image, regardless of activity in the world, leading to high computational and bandwidth demands. Event-based computation diminishes these demands by processing only regions of the image that have changed.

Gabor-like Detection

To extract features, we can take inspiration from biological vision; Gabor-like filters are an example of a common abstraction which have an origin in biology and have been used in traditional computer vision [97]. These can be implemented using spiking neurons whose (immediate) receptive field is distance dependent and synapse weights are proportional to the ones computed by the Gabor function. Methods for transforming weight values have been proposed in the literature [185, 190] and in Chapter 7, we discuss a different approach.

Figure 5.6 (b–g) shows the result of filtering a Modified NIST (MNIST) digit (Figure 5.6(a)). The Gabor filters were generated using the following equations:

$$O(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right), \tag{5.1}$$

$$x' = x \cos\theta + y \sin\theta, \tag{5.2}$$

$$y' = -x \sin\theta + y \cos\theta, \tag{5.3}$$

where λ and ψ are the wavelength and phase of the sinusoidal component, respectively; θ is the orientation of the resulting stripes, σ is the standard deviation of the Gaussian component; and γ is the spatial aspect ratio. Parameters for the generation of Gabor kernels are presented in Table 5.1.

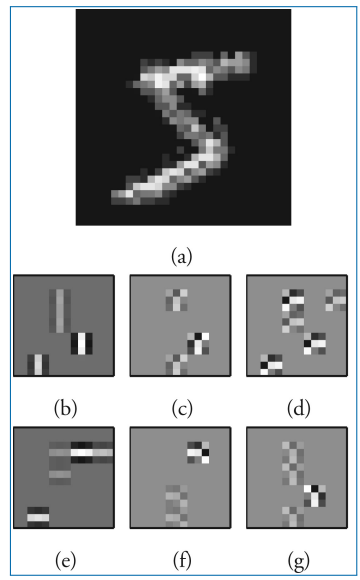

Figure 5.6. Results of Gabor-like feature extraction. (a) shows the input image converted to a spike train and later filtered using six Gabor kernels. (b–g) show the responses of each filtering population projected to the input space.

Table 5.1. Gabor filter parameters.

Width	Sampling	σ	λ	γ	ψ	θ
5	1	2	6	0.5	1.1	[0, 30, 60, 120, 150]

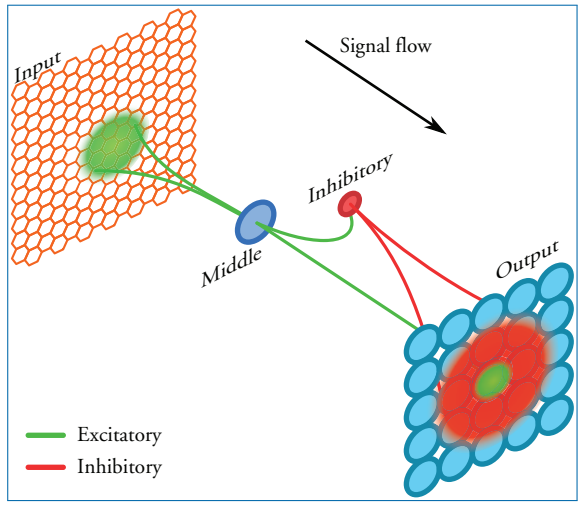

Figure 5.7. Connectivity motif for the blob-detecting network.

Blob Detector

Retinal connectivity has also been used as inspiration for key-point extraction [151]. A retina-inspired network can be used to convert visual input into a multi-scale representation from which blob-like features can be extracted [103]. In this three-layered network (Figure 5.7), the middle layer samples the input layer with receptive fields whose weights are computed using a Gaussian function. Different middle layer 'classes' sample the input with different parameters for their input kernels (i.e. width, σ). Each neuron in the middle layer drives a neuron in the output and, additionally, an inhibitory 'interneuron'. The purpose of the inhibitory neurons is to induce competition between the output layer neurons, reducing activity and pushing the output representation towards orthogonality. All neurons in the output layer compete to represent the input, and the extent to which the inhibitory neurons influence their neighbours is proportional to the cross-correlation of their input image kernels. This competition results in centre-surround receptive fields, as observed in biology.

As an example we took the same input image as in the Gabor filtering (Figure 5.6(a)), and its spike representation was processed by this blob-detection network using three different Gaussian kernel sizes. Figure 5.8 shows the output of the network; we can observe that the greatest activity is present in the mid-resolution class (Figure 5.8(b)) as it is a better fit to the input activity. The high-resolution class (Figure 5.8(a)) shows a behaviour similar to edge detection, typical of centre-surround filtering. Finally, as the receptive field for the low-resolution class is not a good fit for the input, there is little activity observed.

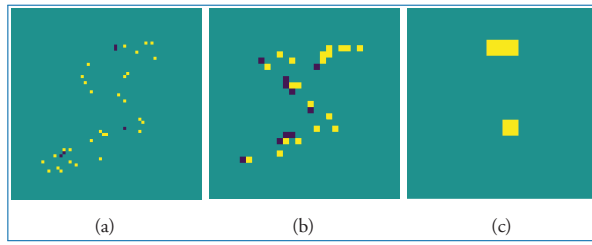

Figure 5.8. Results of blob-detection network. (a) High-, (b) middle- and (c) low-resolution neuron classes.

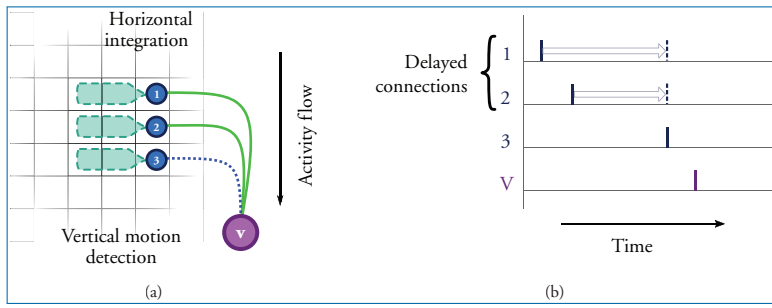

Figure 5.9. Motion sensing circuit. (a) Connectivity of the motion detection circuit using two different neurotransmitters (green-solid and blue-dashed). (b) Delayed lines allow spikes to reach the neuron body at the same time.

Motion Detection

Objects in the world are often moving, and since time is embedded in SNN simulations, we believe it is important to detect motion. A spiking version of a motion detector [103] was developed based on the connectivity of Starburst Amacrine Cells (SAC) [24, 58] and the Reichardt detector [24]. The motion detector network is illustrated in Figure 5.9(a); the principle of operation is composed of two factors: (i) delayed connections and (ii) the combination of two neurotransmitters. Delays are proportional to distance allowing incoming spikes triggered at different times and distances to arrive at (about) the same time (Figure 5.9(b)).

The two neurotransmitters allow activity from different regions of the input to be present at the correct time at the detector neuron (Figure 5.10(a) and 5.10(b)); one of the neurotransmitters decays at a slow rate, opening a window for the other transmitter (whose decay rate is high) to reach the detector.

We tested the circuit using a bouncing ball simulation; the ball moves in a 64×64 pixel window and when it bounces, it does so with a randomly selected speed in a range of 1 to 2 pixels. Figure 5.11 shows the outputs of easterly and westerly motion detection as red-dashed and green-solid lines, respectively. Ball motion is indicated by blue dots in the plot: the ball moved towards the north-east for

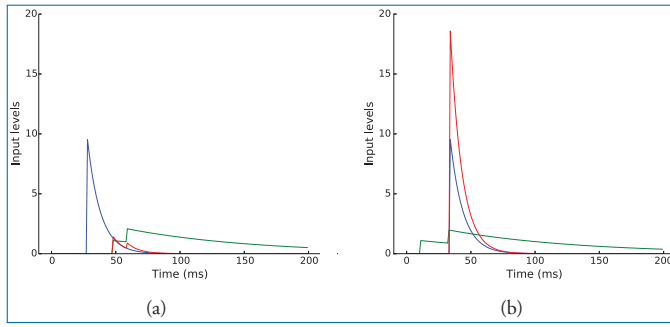

Figure 5.10. Interaction of transmitters in the motion sensing circuit. (a) When neurotransmitters (blue and green lines) do not reach the neuron within a temporal window, they will not induce sufficient current for the neuron to spike. (b) In contrast, when they reach the neuron in the right sequence, they will produce an activation.

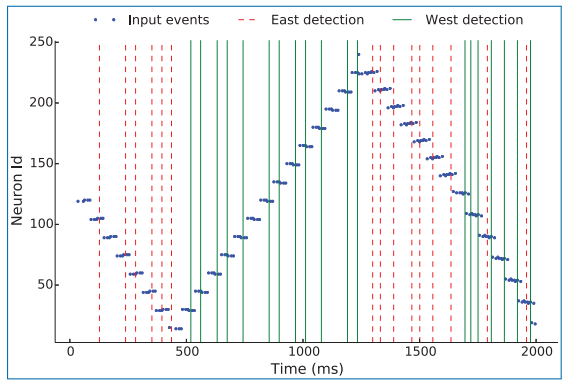

Figure 5.11. Output of the motion sensing circuit.

about 500 ms, then it bounced off a corner and moved in a south-westerly direction until ∼1250 ms; finally, it took off to the north-east again. In the first part (0 to ∼1250 ms) of the experiment, detection is near perfect although there are moments when the detectors fail to sense motion. In the last section (after ∼1250 ms), there are multiple false-positive detections which can be diminished by lateral competition of different directions. This circuit can detect apparent motion with an accuracy of 70%. A similar detector, though with learned connectivity, is described in Section 7.5.5.

5.3 SpiNNak-Ear — On-line Sound Processing

The SpiNNak-Ear system is a fully scaled biological model of the early mammalian auditory pathway: converting a sound stimulus into a spiking representation spread

across a number of parallel auditory nerve fibres [119]. This system takes advantage of the generic digital processing elements on a SpiNNaker machine, enabling a Digital Signal Processing (DSP) application to be distributed across its massively parallel architecture. With the degree of parallel processing available for a SpiNNak-Ear implementation, one is able to generate a simulation of an ear to a biologically realistic scale (30,000 human cochlea auditory nerve fibres) in real time.

5.3.1 Motivation for a Neuromorphic Implementation

A conventional computer simulation can be carried out for large-scale auditory models – albeit with an inherent compromise in processing time due to serialised computation. However, an additional motivation for implementing a parallel simulation of the ear on SpiNNaker is the capability of handling a highly parallel interface between a model of the ear and the rest of the brain running on the same machine. SNNs that model later stages of the auditory pathway and cortical regions of the brain can be specified using the pre-existing SpiNNaker PyNN interface. Using a SpiNNak-Ear model that is already distributed across the same SpiNNaker network allows interfacing the auditory periphery with subsequent SNNs without incurring a data-flow bottleneck penalty.

5.3.2 The Early Auditory Pathway

The early auditory pathway, illustrated in Figure 5.12, begins with a sound pressure wave travelling into the outer ear and eventually displacing the Tympanic Membrane (TM) that separates the outer and middle ear. Inside the middle ear, the TM

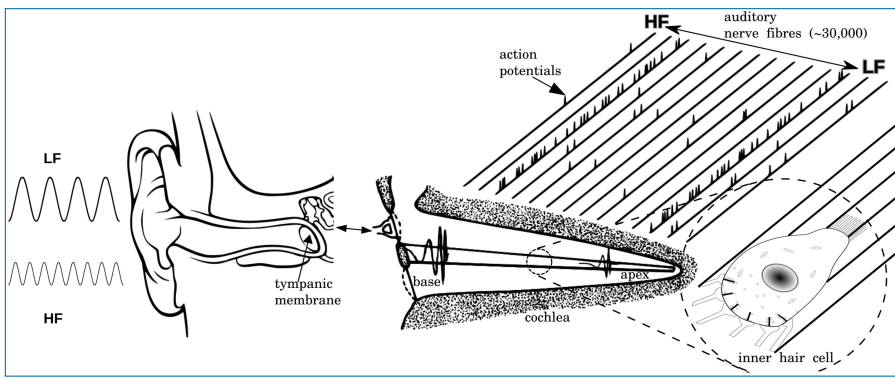

Figure 5.12. An uncoiled cochlea (right) with parallel auditory nerve fibres innervating single IHCs along the cochlea. The spiking activity due to two stimulus frequency components – High Frequency (HF) and Low Frequency (LF) – can be seen in the corresponding auditory nerve fibres.

connects to the cochlea via three ossicle bones to continue (and amplify) this displacement into the inner ear cochlea. The cochlea is a coiled, liquid-filled organ that converts the TM displacement into a series of travelling waves along its distance, from base to apex. The frequency components of the sound stimulus dictate the location along the cochlea that will experience the most displacement along its Basilar Membrane (BM). High frequencies are absorbed at the basal regions and progressively lower frequencies reach the apical regions of the cochlea. The cochlea is lined with many motion sensitive cells, known as Inner Hair Cells (IHCs), that detect the localised displacements of the BM. The IHCs act as the 'biological transducers' in the ear, converting physical sound-produced displacements into a corresponding spike code signal on the auditory nerve.

The modelling of every section of the cochlea's BM and the nearby IHCs can be described as being 'embarrassingly parallel', where the processing of each individual node (a Dual Resonance Non-Linear [DRNL] + IHC models) does not depend on any other neighbouring nodes. Therefore, we can model the processing of specific regions of the cochlea in a concurrent fashion.

5.3.3 Model Algorithm and Distribution

The algorithm used in SpiNNak-Ear is based on the MATLAB Auditory Periphery (MAP) model [159]. It separates the digital modelling of the ear into three separate modules representing ascending biological regions. The first module models the outer and middle ear (OME) using infinite impulse response filters. The second module mimics the sound stimulus frequency separation that occurs along the length of the cochlea using a filter bank of DRNL filters [150]. The final module represents the processing of the IHC and Auditory Nerve (AN) (IHC/AN) and is based on the algorithm described by Sumner *et al.* [245].

The complete SpiNNak-Ear module distribution is outlined in Figure 5.13; it consists of a single OME model instance and many DRNL and IHC/ANs instances depending on the number of cochlea frequency channels specified by the user. The data transfer between the OME model and connected DRNL models is performed using the SpiNNaker multicast-with-payload messaging method. This efficient routeing mechanism allows for the output of the OME model to be sent, a 32-bit sample at a time, as a multicast packet payload to all DRNL models located anywhere on the SpiNNaker machine. These incoming samples are stored in a local-to-core memory buffer and are batch processed when the designated processing segment size (96 samples) has been received. Following DRNL processing, the output 96 × 64-bit word segments are stored in a shared on-chip SDRAM memory circular buffer. This allows an efficient block data transfer between a 'parent' DRNL model and its 'child' IHC/AN models (always located on the same chip) necessary

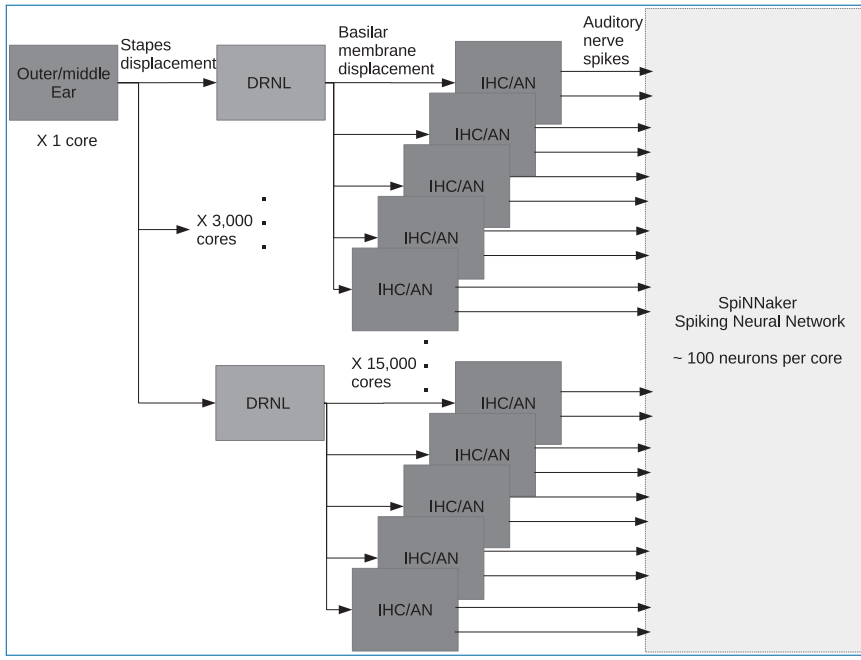

Figure 5.13. A schematic for the human full-scale early auditory path model distribution on SpiNNaker. The total number of cores for this simulation is 18,001 spanning across 1,500 SpiNNaker chips.

for real-time performance. The shared memory communication link that triggers a 'read from shared buffer' event in a child IHC/AN model is achieved using a multicast packet transmission from the parent DRNL model once it has processed a segment. Figure 5.14(a) illustrates these two data communication methods used in the full model system.

In the full system, the OME model application is triggered by the real-time input stimulus, after which the subsequent DRNL and IHC/AN models in the software pipeline are free to run asynchronously (event-driven) until the AN output stage. In a given simulation, to confirm that all model instances have initialised or have finished processing, we use 'core-ready' or 'simulation-complete' acknowledgement signals fed back through the network of all connected model instances to the parent OME model instance to ensure all cores are ready to process and data have been successfully recorded within the given time limits.

5.3.4 Results

The output from SpiNNak-Ear simulation is compared with conventional computer-based simulation results from the MAP model to ensure no significant

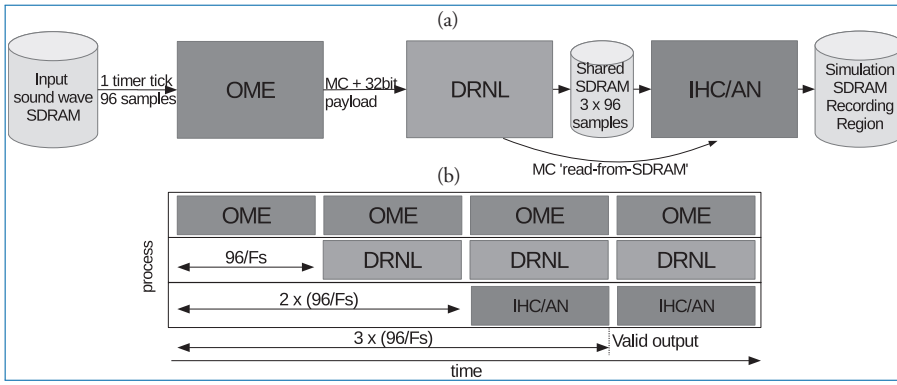

Figure 5.14. (a) The data passing method from input sound wave to the output of a single IHC/AN instance using MC and MC with payload message routeing schemes. (b) The pipeline processing structure used to achieve real-time performance.

numerical errors have occurred from computing the model algorithm on different simulation hardware. The outputs from both implementations are then compared with physiological experimental results to confirm the model's similarities to the biological processes it emulates.

In experimental neuroscience, the response from a stochastic auditory nerve fibre to an audio stimulus is measured over many repeated experiments and the subsequent recordings are often displayed in a Peri Stimulus Time Histogram (PSTH). The results, shown in Figure 5.15, show the time varying AN spike rates across 1 ms windows to a 6.9 kHz sinusoidal 68 dBSPL stimulus, first in Figure 5.15(a) from physiological data gathered by Westerman and Smith [265] and then from both model implementations in Figure 5.15(b). These results show both implementations produce a biologically similar response consisting of pre-stimulus firings of approximately 50 spikes/s, followed by a peak response at stimulus onset at around 800 spikes/s, decaying to an adapted rate in the region of 170 spikes/s. Finally at stimulus removal, rates significantly drop during an offset period before returning to spontaneous firing of approximately 50 spikes/s.

Figure 5.16 illustrates the energy consumed by MAP and SpiNNak-Ear implementations across the full range of model channels tested. Energy consumption has been calculated by multiplying the complete processing time by the total power rating of the hardware used (CPU at 84 W, single SpiNNaker chip at 1 W). Here we show that both implementations incur an increase in total energy consumed – but for different reasons. The MAP implementation running on a single, fixed power CPU uses more energy when the number of channels is increased due to the increase in serialised processing time. The neuromorphic hardware experiences an increase in energy consumed due to the increasing size of the machine used (number of

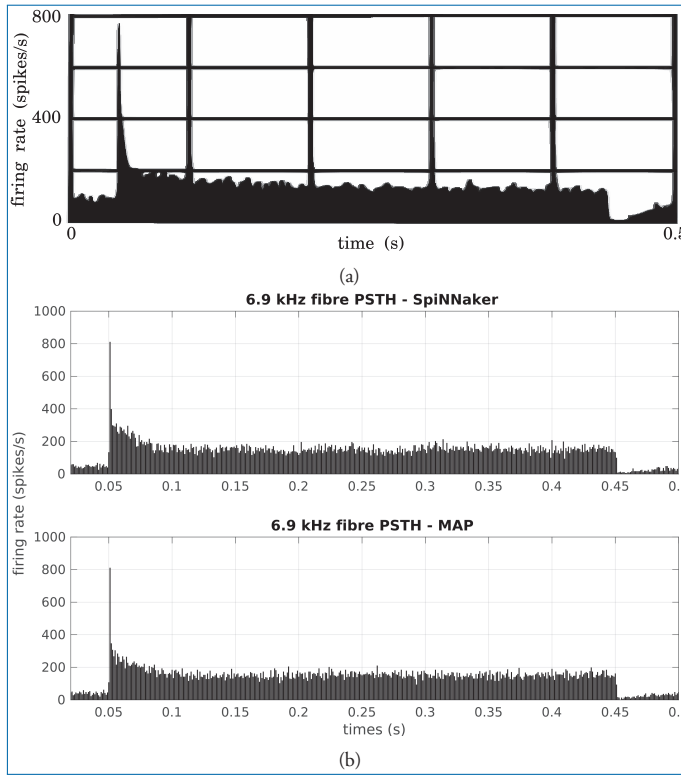

Figure 5.15. PSTH responses to 352 repetitions of a 400 ms 6.9 kHz 68 dBSPL stimulus from experimental data obtained by Westerman and Smith [265] of an HSR AN fibre in a gerbil (a) and the same experiment repeated for MAP and SpiNNaker implementations (b).

chips) with an increase in channels. The rate of increase in energy consumed due to number of channels on neuromorphic hardware is lower than the conventional serial CPU approach. This effect illustrates the basic philosophy that underlies the functionality of SpiNNaker (and biological) processing systems: complex computation on a modest energy budget, performed by dividing overall task workload across a parallel network of simple and power-efficient processing nodes.

5.3.5 Future Developments

A goal for the future of SpiNNak-Ear is to enable simulations with a live stream audio signal input. This has the potential to provide the user with an interactive visual representation of various regions of the brain to their current sound environment. Such a facility of 'in-the-loop' experimentation may assist in gaining further understanding of the important features of biological neural networks.

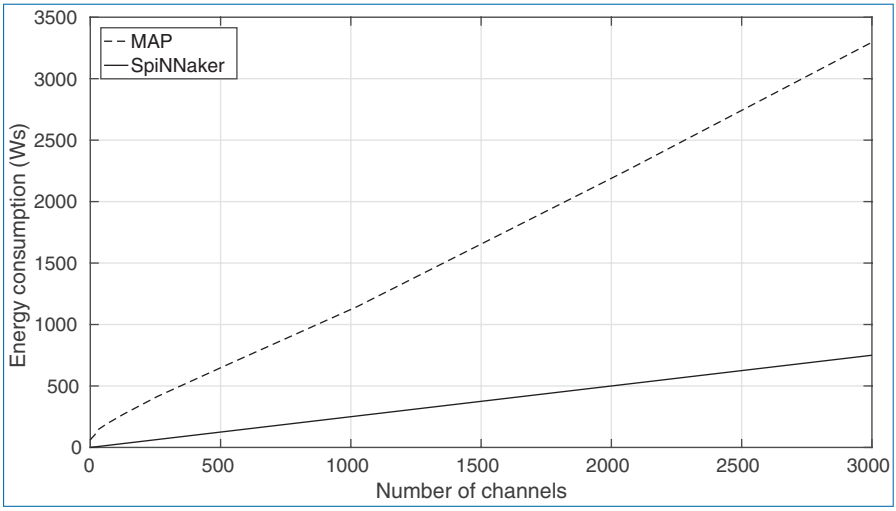

Figure 5.16. Average energy consumption from processing a 0.5 s sound sample from 2 to 3,000 channels on both MAP and SpiNNaker implementations. The MAP model is executed on a desktop computer (Intel Core™ i5-4590 CPU @ 3.3 GHz 22 nm technology) and SpiNNaker on a range of different sized SpiNNaker machines ranging from 1 to 1,500 chips (130 nm technology) scaled by the number of channels in a simulation.

The SpiNNak-Ear implementation on the SpiNNaker platform can be used in future investigation into the importance of the descending projections that feature between stages of the auditory pathway. It has been shown that descending projections may be providing useful feedback modulation to the incoming sound representation, 'tuning' the representations of learnt salient stimuli [252] and producing stimulus-specific adaptation in sensory neurons [153]. Therefore, if a research goal is to gain a full understanding of the auditory system, one must model it completely with multiple feedback projections. Implementing such connectivity across a large complex system in a computer simulation becomes an increasing burden on system communication resources. On the SpiNNaker hardware architecture, using the novel one-to-many multicast message routeing mechanism, additional descending projections can be integrated into simulations without incurring large overheads and with the ability to simulate real-time feedback to the user.

5.4 Basal Ganglia Circuit Abstraction

Here we present a biologically plausible and scalable model of the Basal Ganglia (BG) circuit, designed to run on the SpiNNaker machine [217]. It is based on the Gurney–Prescott–Redgrave model of the BG [84, 85]. The BG is a set of

subcortical nuclei that are evolutionarily very old and appear in all vertebrates, enabling them to make decisions and take subsequent actions; obviously, therefore, computational modelling of the BG has been pursued by researchers with an interest in robotics [202]. The information on which the decision needs to be made, that is, the environmental circumstance, constitutes the input to the BG and is available via the thalamus and cortex. Output from the BG is the specific action that is decided upon, referred to as 'action-selection', and is relayed to the motor pathway for execution via the thalamus, cortex and other subcortical structures. The objective of our work on SpiNNaker is to build a 'basic building block' towards development of automated decision-making tools in real time.

A single neuro-computational unit in our BG model is simulated with a conductance-based Izhikevich neuron model. A columnar structure of the BG circuitry is shown in Figure 5.17; this forms the basic building block for our scalable framework and is thought to be a single 'channel' of action selection. The striatum forms the main input structure of the BG and receives excitatory glutamatergic synapses from both the cortex and the thalamus. The substantia nigra pars reticulata (SNr) forms the output structure of the BG and projects inhibitory efferents to the ventral thalamus and brainstem reticular formation.

The single-channel BG model is first parameterised on SpiNNaker to set the base firing rates for all model cell populations, informed by prior work by Humphries *et al.* [110]. Next, to simulate action selection by competing inputs, the model is

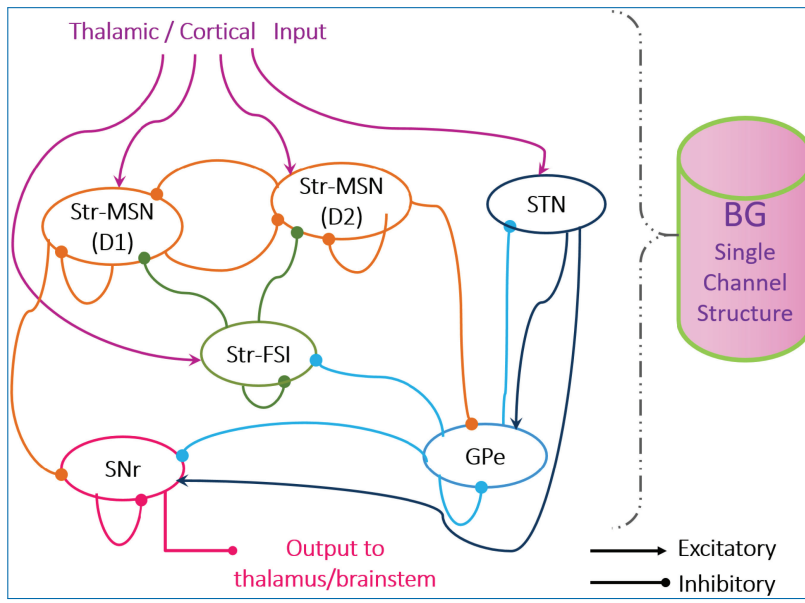

Figure 5.17. Single-channel action selection architecture.

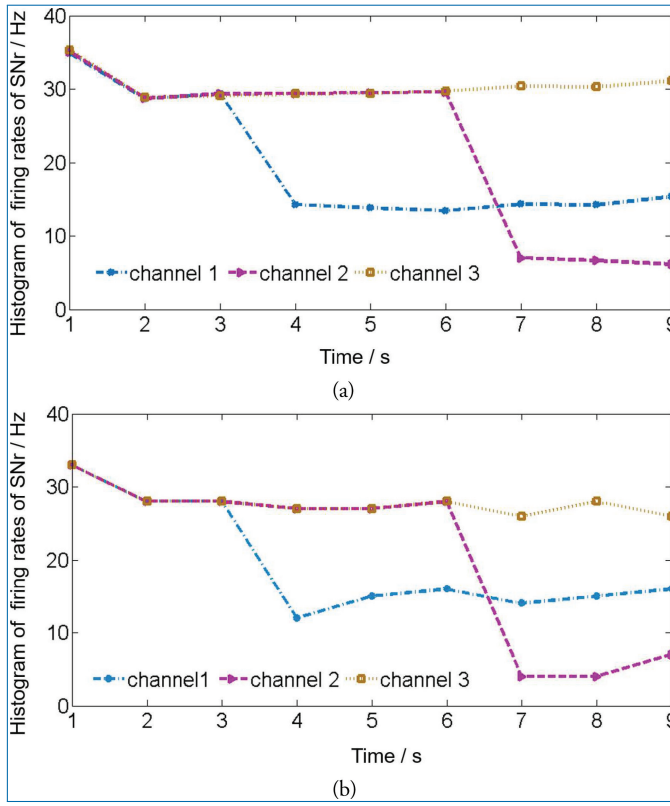

Figure 5.18. Demonstration of action selection in a 3-channel BG model on (a) SpiNNaker and (b) SpineML. All three channels have a 3 Hz Poisson input. At 3 seconds, a 15 Hz Poisson input is provided to channel 1 (blue), when the firing rate of the node drops, demonstrating disinhibition and therefore action selection by the node. At 6 seconds, channel 2 is provided with a 25 Hz input, and therefore, channel 2 now gets to select an action, as it is the overall winner with the lowest firing rate.

scaled up to three channels and tested with two competing inputs in the presence of a noisy background stimulus. Results are summarised in Figure 5.18(a). An input stimulus that is larger than the others is always the 'winner', indicated by a relative drop in the firing rate of the SNr population (representing the BG model output) in the competing channel. The reduced firing rate of the inhibitory SNr population implies a reduced inhibition of the thalamic/brainstem cells, which are the recipients of the BG output as mentioned above. This in turn means that the 'action' that is solicited by a relatively larger ('competing') input is now 'decided' by the BG circuit to be 'selected and acted upon', indicated by disinhibition of the target outputs. The model is tested with a competing input of 15 Hz in the presence of a noisy background input of 3 Hz. This is further confirmed by 'selection' of a larger

input of 25 Hz provided in the presence of both 15 Hz and 3 Hz inputs. On both occasions, the largest input wins.

It is worth mentioning here that dopamine neurotransmitter-receptor levels are fundamental to facilitating decision-making and action selection by the BG. Here, the base parameters are tuned to simulate neutral dopamine levels; studying model dynamics with varying levels of dopamine will be carried out in future work.

To verify the model results simulated on SpiNNaker, the model is mapped to SpineML, an XML-based platform representing model attributes as 'components' and executing the models with SpineML_2_BRAHMS, a bespoke simulator that converts the SpineML model into machine code and runs it on a conventional computer. We aimed for the BG model implementation on SpineML to have the exact same network topology and neuron attributes as the SpiNNaker version and therefore retained all model connectivities and parameter values used in the latter. Model results on SpineML show qualitative similarity with those on SpiNNaker in terms of base firing rates of the single-channel BG model cell populations. Implementation of the three-channel model on SpineML, following the exact same implementation procedures as on SpiNNaker, demonstrates action selection by a larger input and is shown in Figure 5.18(b). Comparing Figures 5.18(a) and 5.18(b) shows an agreement between the functional and qualitative behaviour of the models simulated on SpiNNaker and SpineML. We believe that our comparative study will provide a basic framework for mapping SpiNNaker-based models to SpineML, as well as for performance benchmarking of SpiNNaker with conventional computers during neuronal simulation.

The single-channel BG model consists of 2.68×10^3 neurons and $\approx 0.68 \times 10^6$ synapses (estimated from projection probabilities). While each processor within a SpiNNaker chip is capable of simulating an upper limit of 256 neurons [217], memory requirements of the neuron model and synaptic connectivity for certain applications may cause this number to be reduced. In the current work, sPyNNaker maps the single-channel BG model onto 32 cores distributed across 2 SpiNNaker chips, residing on a single 48-chip SpiNNaker Board. Power consumption of the single-channel BG model executing on a 48-node board is measured using in-house Raspberry-Pi-based power measurement equipment [244]. Figure 5.19(a) shows that the single-channel model execution uses $\approx$800 mW. We have also observed that the model execution time is not affected by scaling up to three channels and is consistent at 100 s real time corresponding to a simulation time of 10 s. As power consumed during pre- and post-processing are negligible compared to that during model execution, we kept the post-processing time to a minimum; pre-processing times are handled by sPyNNaker and are not accessible to the user. Figure 5.19(b) shows a performance comparison between SpiNNaker and SpineML. The main constraints on SpiNNaker currently are the pre- and post-processing times that are

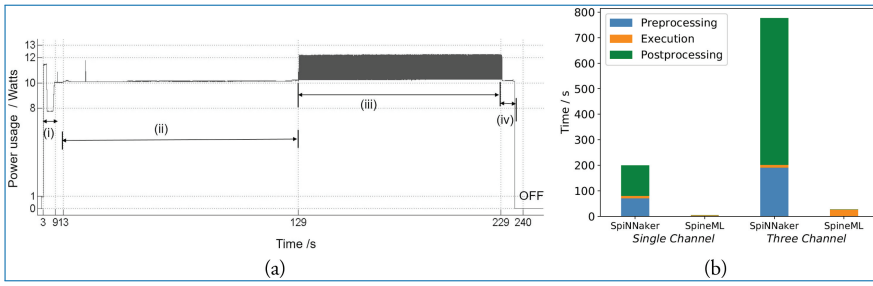

Figure 5.19. (a) The power consumption of the single-channel model using an in-house Raspberry-Pi-based measurement system connected to the SpiNNaker board [244]. The duration of recording power can be broken down into four regions: (i) booting the machine; (ii) pre-processing of data; (iii) model execution; (iv) post-processing (i.e. data extraction); the delay of around 4 s after booting the machine is inserted for clarity. The peak-to-peak power in region (iii) is 800 mW. (b) Performance analysis of single-channel and three-channel models running on both SpiNNaker and SpineML. Execution time on SpiNNaker, and pre- and post-processing times on SpineML are unaffected by scaling-up of model.

negligible on SpineML running on a 4-core 8 GB RAM desktop host machine, even for the scaled-up model. In contrast, execution time on SpiNNaker is not affected by model scaling; execution time for the SpineMLmodel is affected by scale. We believe that our comparative study will provide a basic framework for mapping SpiNNaker-based models to SpineML, as well as for performance benchmarking of SpiNNaker with conventional computers during neuronal simulation.

5.5 Constraint Satisfaction

When developing a biologically inspired hardware architecture, apart from looking for an improvement of our understanding of living matter, it is also desirable to explore the capabilities of the machine on the realm of more general problems in Mathematics and Computer Science. If the machine succeeds in representing or solving any of the well-defined abstract problems, or classes of problems, this means that it will be applicable to the specific cases that can be formulated under such formalisms, making the machine attractive for Physics and Engineering. The more general the class of problem, the broader the range of applications that will be covered and the better we will understand the capabilities of the design and, more importantly, we will understand its limitations. Understanding the limits of applicability of a computational approach is perhaps more important than evolving its capabilities. The reason being that some problems are indeed intractable in the

sense that it does not matter how much we improve the speed, power consumption or size of our computers, there are families of problems which, despite being solvable in principle with infinite resources, will remain intractable at least until some exotic machine demonstrates an exponential speedup. Quantum and genetic computers, at least in theory, promise advances in this direction, but the practicalities currently seem to be out of scope. Worse than that are the undecidable or unsolvable problems. Hence, knowing the performance and complexity of a new computer architecture in the hierarchy of computable and incomputable problems will shed light on realistic directions for optimisation and improvement, avoiding the use of valuable time on aspects that will not add significant scientific or technological value.

Constraint Satisfaction Problems (CSPs) are a special family of problems that serve such a purpose. They are beautifully simple to formulate, yet they belong to the class of intractable problems (the NP-complete family). These are problems whose solutions are verifiable in Polynomial time (P), yet finding their solution requires supra-polynomial time as a function of the size of the problem. Actually, evidence suggests that the time complexity may be exponential, that is, a linear increase of the problem size results in an exponential increase of the required resources: time or space, memory or energy.

Formally, a CSP is defined by a set of variables $X = \{x_1, \ldots, x_N\}$ that take values over a set of discrete or continuous domains $D = \{D_1, \ldots, D_N\}$, such that a set of constraints $C = \{C_1, \ldots, C_m\}$ are satisfied. Each such constraint is defined as a tuple $C_i = \langle S_i, R_i \rangle$, where $R = \{R_1, \ldots, R_k\}$ are k relations over m subsets $S = \{S_1, \ldots, S_m : S_i \subseteq X\}$. In short, $CSP = \langle X, D, C \rangle$. Hence, the problem is defined over a combinatorial space whose size is on the order of $\overline{D}^N$, growing exponentially with N. Every solution to a CSP will have zero violations and include all variables in X. Hence, it will be represented by a global minimum of the cost hypersurface. If the problem has several solutions, the global minimum will be degenerate, one minimum existing for each solution. It is easy to see then that the difficulty of finding a solution for a CSP depends not only on the high dimensionality of its combinatorial space but also critically on the curvature of that space. Here, the curvature refers to how folded the space of possible evaluations of X is when measured against a scalar (energy or cost) function related to the number of unsatisfied constraints. If the cost function is strictly convex, there will be a single minimum and methods such as gradient descent will easily find it. Unfortunately, this is rarely the case.

With a geometrical representation of CSPs, it is easy to imagine solving the problem by travelling across the cost hypersurface, defined on some high-dimensional space, looking for a global minimum. Think of it as being like an adventurous explorer in the middle of the Amazon rain forest, perhaps searching for some

previously uncontacted peoples. Having a helicopter would certainly be an advantage. Access to a satellite with high-resolution cameras and powerful zooming abilities will save you a lot of time. However, without any of them, you only have access to local information. Being inside the rain forest, any fancy equipment could only help you if it has access to some global parameter. That is the problem faced when solving CSPs. In general, you can provide the solver with local information, but any global information will only have low resolution, usually obtained from a discrete sampling over the state space, whose useful information content will decrease drastically with the curvature of the cost function.

Our aim in this section is to explore the representation and solution of CSPs on SpiNNaker. We stick to the use of spiking neurons, embracing the main purpose of the machine. However, bear in mind that SpiNNaker is a digital system, built from general-purpose processors, and has features that allow non-neural implementations. We will not consider those here. The following is extracted with minor modifications from Fonseca Guerra and Furber [61].

5.5.1 Defining the Problem

Consider a set $\mathcal{N}$ of neurons obeying a dynamic model defining the time evolution of a state variable, usually the membrane potential u_i and a threshold function θ_i that defines the generation of a spike event whenever the state variable reaches θ_i from below. After a spike u_i is forced below the threshold; it can, for example, be reset to a resting u_{rest} or reset u_{reset} membrane potential. The SNN is defined by the set $\mathcal{N}$ together with a set of synapses $\mathcal{S} \subseteq \mathcal{N} \mathrm{X} \mathcal{N}$ that define connections between pairs of neurons $\mathcal{N}_i$ and $\mathcal{N}_j$. Each synapse $\mathcal{S}_{i,j} \in \mathcal{S}$ has an associated weight parameter $w_{i,j}$ and a response function $\mathcal{R}_{i,j} : \mathbb{R}^+ \to \mathbb{R}$.

Let us also define the instantaneous state of the SNN as $\psi_t = \{n_1, n_2, \ldots, n_N\}$, that is, the ordered set of firing states $n_i \in \{0, 1\}$ for every neuron in $\mathcal{N}_i$ at time t. In SpiNNaker, ψ_t is well defined because time is discretised, generally in steps of 1 ms, and at each step, the firing state of a neuron is measurable. This definition avoids the need to track the membrane potential $u_i \in \mathbb{R}$ of each neuron at each particular time. Strictly speaking, u_i is represented by a 32-bit binary number in SpiNNaker so its resolution is finite.

In our implementation, each neuron $\mathcal{N}_i$ corresponds to a LIF neuron [238]. In this model, the dynamics of the membrane potential u are given by:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t). \tag{5.4}$$

Here, τ_m is the membrane time constant, R is the membrane resistance and I is the external input current. Each time u reaches a threshold value θ a spike is elicited; such events are fully characterised by the firing times

$\{t^f \mid u(t^f) = \theta \text{ and } \frac{du}{dt}\lim_{t=t^f} > 0\}$. Immediately after a spike, the potential is reset to a value u_r, such that $\lim_{t \to t^f +} u(t) = u_r$. In our network, synapses are uniquely characterised by ω_{ij} and the inter-neural separation is introduced by means of a delay Δ_{ij}. In biological neurons, each spike event generates an electrochemical response on the post-synaptic neurons characterised by $\mathcal{R}_{i,j}$. We use the same function for every pair (i, j), and this is defined by the post-synaptic current:

$$j(t) = \frac{q}{\tau} e^{-\frac{t-t_0}{\tau}} \Theta(t - t_0), \tag{5.5}$$

where q is the total electric charge transferred through the synapse, τ is the characteristic decaying time of the exponential function, $t_0 = t^f + \Delta_{ij}$ is the arrival time of the spike and Θ represents the Heaviside step function. The choice of $\mathcal{R}_{i,j}$ potentially affects the network dynamics, and although there are more biologically realistic functions for the post-synaptic response, the use of the exponential function in Equation 5.5 constitutes one of our improvements over the previous studies on CSP through SNNs which used a simple square function.

In an SNN, each neuron is part of a large population. Thus, besides the background current $I(t)$, it receives input from the other neurons, as well as a stochastic stimulation from noisy neurons implementing a Poisson process. In this case, the temporal evolution of the membrane potential (Equation 5.4) generalises to:

$$\tau_m \frac{d}{dt} u = -u(t) + R \left[I(t) + \sum_j \omega_j \sum_f j(t - t_j^f) + \sum_k \Omega_k j(t - T_k) \right] \tag{5.6}$$

where the index f accounts for the spike times of principal neuron j in the SNN, Ω_k is the strength of the kth random spike, occurring at time T_k, and $j(.)$ is the response function of Equation 5.5. An SNN has the advantage that its microstate $\psi_t = \{n_1, n_2, \ldots, n_N\}$ at any time t can be defined by the binary firing state $n_i \in \{0, 1\}$ of each neuron $\mathcal{N}_i$, instead of the continuous membrane potentials $u_i \in \mathbb{R}$. Then, the set of firing times $\{t_i^f\}$ for every neuron $\mathcal{N}_i$, or equivalently the set of states $\{\psi_t\}$, corresponds to the trajectory (dynamics) of the network in the state space. The simulations in this work happen in discrete time (time step = 1 ms) so, in practice, ψ_t defines a discrete stochastic process (e.g. a random walk). If the next network state $\psi_{t_{i+1}}$ depends on ψ_{t_i} but is conditionally independent of any ψ_{t_j} with $j < i$, the set $\{\psi_t\}$ also corresponds to a Markov chain. Habenschuss *et al.* [89] demonstrated that this is the case when using rectangular Post-Synaptic Potentials (PSPs) and a generalised definition of the network state, the validity of the Markov property for general SNNs could still depend on the dynamical regime and be affected by the presence of a non-zero probability current

for the stationary distribution [39]. Each possible configuration of the system, a microstate ψ_i, happens with certain probability p_i and, in general, it is possible to characterise the macroscopic state of the network with the Shannon entropy (in units of *bits*) [221]:

$$S = -\sum_i p_i \log_2 p_i \tag{5.7}$$

and the network activity:

$$A(t) = \frac{1}{N} \sum_j^N \sum_f \delta(t - t_j^f) \tag{5.8}$$

To compute p_i and hence Equation 5.7, we binned the spikes from each simulation with time windows of 200 ms. In this type of high-dimensional dynamical system, sometimes the particular behaviour of a single unit is not as relevant as the collective behaviour of the network, described, for example, by Equations 5.7 and 5.8.

A constraint satisfaction problem $\langle X, D, C \rangle$ can now be expressed as an SNN as shown in the pseudo-code of Listing 5.1. We can do it in three basic steps: (a) create SNNs for each domain d_i of each variable, every neuron is then excited by its associated noise source, providing the necessary energy to begin exploration of the states $\{\psi\}$; (b) create lateral-inhibition circuits between all domains that belong to the same variable; (c) create lateral-inhibition circuits between equivalent domains of all variables appearing in a negative constraint and lateral-excitation circuits for domains in a positive constraint. With these steps, the resulting network will be a dynamical system representation of the original CSP. Different strategies can now be implemented to enforce the random process over states ψ_t to find the configuration ψ_0 that satisfies all the constraints. The easiest and proposed way of implementing such strategies is through the functional dependence of the noise intensity on time. The size of each domain population should be large enough to average out the stochastic spike activity. Otherwise, the system will not be stable and will not represent quasi-equilibrium states. As will be shown, it is the size of the domain populations what allows the system to converge into a stable solution.

The ensemble of populations assigned to every CSP variable x_i works as a Winner-Takes-All (WTA) circuit through inhibitory synapses between domain populations, which tends to allow a single population to be active. However, the last restriction should not be over-imposed, because it could generate saturation and our network will be trapped in a local minimum. Instead, the network should constantly explore configurations in an unstable fashion, converging to equilibrium only when satisfiability is found. The random connections between populations, together with the noisy excitatory populations and the network topology,

provide the necessary stochasticity that allows the system to search for satisfiable states. However, this same behaviour traps some of the energy inside the network. For some problems, a dissipation population could be created to balance the input and output of energy or to control the entropy level during the stochastic search. In general, there may be situations in which the input noise acquired through stimulation can stay permanently in the SNN. Thus, the inclusion of more excitatory stimuli will saturate the dynamics at very high firing rates, which potentially could reach the limits of the SpiNNaker communication fabric. In these cases, inhibitory noise is essential too and allows us to include arbitrarily many stimulation pulses.

We demonstrate in the next section that the simple approach of controlling the dynamics with the stimulation intensities and times of the Poisson sources provides an efficient strategy for a stochastic search for solutions to the studied CSPs.

```
# define the CSP = <X,D,C> through a set of lists.
    X=list(variables)
    D=list(domains)
    S=list(subsets_of(X))
    R=list(relations_over(s_i in S))
    C=list(constraints = tuple(s_i,r_i))
#a) create an SNN for each variable with sub-populations for each domain.
    n = size_of_ensemble
    for variable x_i in X:
        for domain d_i in D:
            population[x_i][d_i] = create an SNN with n neurons
            noise_exc[x_i][d_i] = create a set of noise
            stimulation populations.
            apply_stimuli(noise[x_i][d_i], population[x_i][d_i])
            noise_inh[x_i][d_i] = create a set of noise
            dissipation populations.
            apply_dissipation(noise_inh[x_i][d_i], population[x_i][d_i])
#b) use inhibitory synapses to activate, on average, a single domain per
 variable
        for domain d_i in D:
            for domain d_j in D
                inhibition(population[x_i][d_i],population[x_i][d_j])
#c) map each constraint to an inhibitory or excitatory synapse.
    for constraint c_i in C:
        read subset s_i and relation r_i from c_i
        for variables x_i and x_j in s_i:
            for domain d_i in D:
                if constraint relation r_i <0:
                    inhibition(population[x_i][d_i], population[x_j][d_i])
                elif constraint relation r_i >0:
                    excitation(population[x_i][d_i], population[x_j][d_i])
```

Listing 5.1. Translation of a CSP into an SNN.

5.5.2 Results

In order to demonstrate the implementation of the SNN solver, we present solutions to some instances of Non-deterministic Polynomial time (NP) problems. Among the NP-complete problems, we have chosen to showcase instances of graph colouring, Latin squares and Ising spin glasses. Our aim is to offer a tool for the development of stochastic search algorithms in large SNNs. We are interested in

CSPs to gain understanding of the dynamics of SNNs under constraints, how they choose a particular state and their computational abilities. Ultimately, SNNs embedded in neuromorphic hardware are intended for the development of new technologies such as robotics and neuroprosthetics, constantly interacting with both external devices and the environment. In such applications, the network needs to adapt itself to time-varying constraints taking one or multiple decisions accordingly, making advances in stochastic search with SNNs a fundamental requirement for neuromorphics.

5.5.3 Graph Colouring

Consider a graph G defined by the ordered pair $\{V, E\}$, with V a set of vertices and E the set of edges connecting them. The graph colouring problem consists of finding an assignments of k colours to the elements of the graph (either V, E or both) such that certain conditions are satisfied [41]. In vertex colouring, for example, the colours are assigned to the elements of V in such a way that no two adjacent nodes (those connected by an edge) have the same colour. A particularly useful applications of this problem is the process of register allocation in compiler optimisation which is isomorphic to graph colouring [35]. Regarding time complexity, general graph colouring is NP-complete for $k > 2$. In the case of planar graphs, 3-colouring is NP-complete and, thanks to the four-colour theorem proved by Appel and Haken [5], 4-colouring is in P.

A division of a plane into several regions can be represented by a planar graph, familiar versions of which are the geographic maps. In Figure 5.20(a), we show the SNN-solver result of a satisfying 4-colouring of the map of the world where colours are assigned to countries such that no bordering countries have the same colour. We have used the list of countries and borders from the United Nations available in Mathematica Wolfram [113]. The corresponding connectivity graph of the world map in Figure 5.20(a) is shown in Figure 5.20(b). The insets in Figure 5.20(a) show the results of our solver for 3-colouring of the maps of the territories of Australia (bottom-right) and of Canada (top-left). Figure 5.20(c) and (d) show the time dependence of the entropy (top), firing rate (middle) and number of visited states (bottom) for the map of the world and of Australia, respectively. The colour code we use in these and the following figures is as follows: red means that the state in the current time bin is different from the one just visited, green represents the network staying in the same state and blue means that all constraints are satisfied. The dashed vertical lines mark the times at which noise stimulating (blue) or depressing (red) populations began to be active. The normalised spiking activity of the four colour populations for four randomly selected countries of the world map is shown in Figure 5.20(e) evidencing the competing behaviour along the stochastic

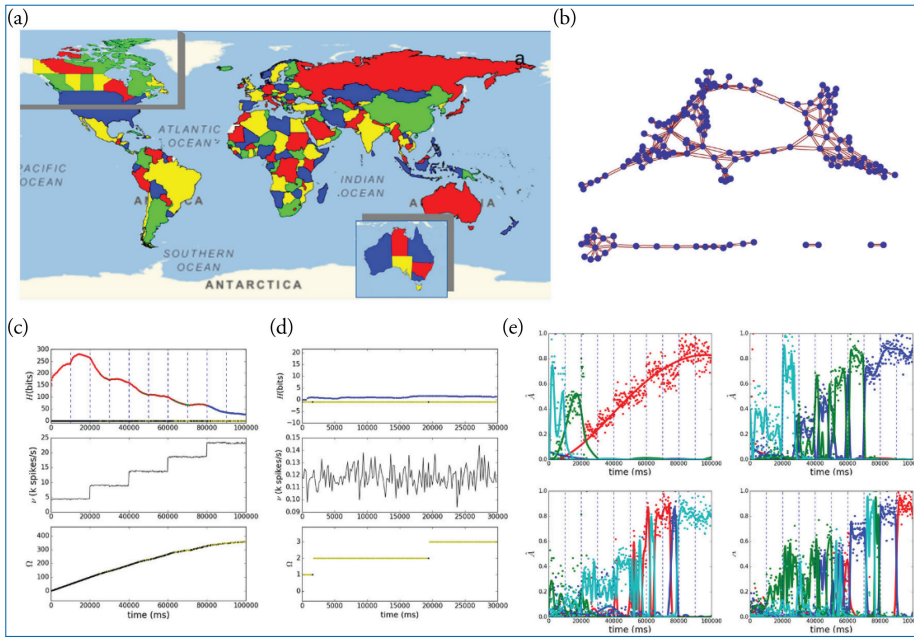

Figure 5.20. (a) Solution to the map colouring problem of the world with 4 colours and of Australia and Canada with 3 colours (insets). Figure (b) shows the graph of bordering countries from (a). The plots of the entropy H (top), mean firing spike rate ν (middle) and states count Ω (bottom) versus simulation time are shown in (c) and (d) for the world and Australia maps, evidencing the convergence of the network to satisfying stationary distributions. In the entropy curve, red codes for changes of state between successive time bins, green for no change and blue for the network satisfying the CSP. In the states count line, black dots mean exploration of new states; the dots are yellow if the network returns to states visited before. In (e), we have plotted the population activity for four randomly chosen CSP variables from (a), each line represents a colour domain.

search. Interestingly, although the network has converged to satisfaction during the last 20 s (blue region in Figure 5.20(c)), the bottom right plot in Figure 5.20(e) reveals that due to the last stimulation the network has swapped states preserving satisfaction, evidencing the stability of the convergence. Furthermore, it is noticeable in Figure 5.20(d) that new states are visited after convergence to satisfiability; this is due to the fact that, when multiple solutions exist, all satisfying configurations have the same probability of happening. Although we choose planar graphs here, the SNN can implement any general graph; hence, more complicated P and NP examples could be explored.

5.5.4 Latin Squares

A Latin square is defined as an array of $n \times n$ cells in which n groups of n different symbols are distributed in such a way that each digit appears only once in each row or column. The NP-completeness of solving a partially filled Latin square was demonstrated by Colbourn [38], and among the useful applications of such a problem, one can list authentication, error detection and error correction in coding theory. Here we choose the Sudoku puzzle as an instance of a Latin square, in this case, $n = 9$ and in addition to the column and row constraints of Latin squares, Sudoku requires the uniqueness of the digits in each 3×3 sub-grid. We show in Figure 5.21 the solution to an easy puzzle [57], to a hard Sudoku [89] and to the AI

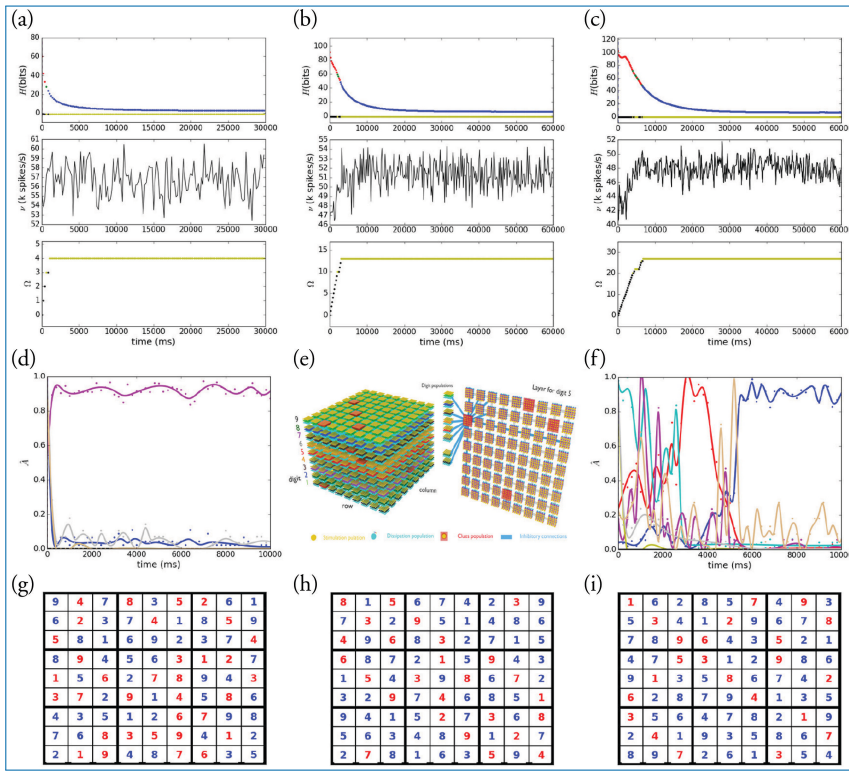

Figure 5.21. SNN solution to Sudoku puzzles. (a–c) show the temporal dependence of the network entropy H, firing rate ν and states count Ω for the easy (g), hard (h) and AI escargot (i) puzzles. The colour code is the same as that of Figure 5.20. In (g–i), red is used for clues and blue is used for digits found by the solver. Figures (d) and (f) illustrate the activity for a random selected cell from (a) and from (c), respectively, evidencing competition between the digits, the lines correspond to a smoothing spline fit. (e) Schematic representation of the network architecture for the puzzle in (a).

Escargot puzzle, which has been claimed to be the hardest Sudoku. The temporal dependence of the network entropy H, firing rate ν and states count Ω is shown in Figures 5.21(a)–(c), respectively, for the easy (5.21(g)), hard (5.21(h)) and AI escargot (5.21(i)) puzzles. In Figure 5.21(e), we show a schematic representation of the dimensionality of the network for the easy puzzle (g); each sphere represents a single neuron and synaptic connections have been omitted for clarity; the layer for digit 5 is represented also showing the inhibitory effect of a single cell in position (1,3) over its row, column, subgrid and other digits in the cell. In this case, the total number of neurons is $\approx 37\,k$ and they form $\approx 86\,M$ synapses.

One major improvement of our implementation with respect to the work of Habenschuss *et al.* [89] is the convergence to a stable solution; this is arguably due to the use of subpopulations instead of single neurons to represent the domains of the CSP variables as these populations were required to provide stability to the network. The use of the more realistic exponential post-synaptic potentials instead of the rectangular ones used by Habenschuss *et al.* [89] helps deliver a good search performance as shown in the bottom plots in Figure 5.21(a)–(c), where the solution is found after visiting only 3, 12 and 26 different states and requiring 0.8 s, 2.8 s and 6.6 s, respectively, relating well also with the puzzle hardness. It is important to highlight that the measurement of the difficulty level of a Sudoku puzzle is still ambiguous and our solver could need more complex strategies for different puzzles, for example, in the transient chaos-based rating the 'platinum blonde' Sudoku is rated as one of the hardest to solve, and although we have been able to find a solution for it, it is not stable, which means one should control the noisy network dynamics in order to survive the long escape rate of the model presented by Ercsey-Ravasz and Toroczkai [57]. We show in Figure 5.21(d) and (f) the competing activity of individual digit populations of a randomly chosen cell in both the easy and the AI escargot puzzles. The dynamic behaviour resembles that of the dynamic solver in Figure 2 of the work by Ercsey-Ravasz and Toroczkai [57] for this same easy puzzle and platinum blonde. Further analysis would bring insights into the chaotic dynamics of SNNs when facing constraints.

5.5.5 Ising Spin Systems

For each atom that constitutes a solid, it is possible to define a net spin magnetic moment $\vec{\mu}$ resulting from the intrinsic spin of the subatomic particles and the orbital motion of electrons around their atomic nucleus. Such magnetic moments interact in complex ways giving rise to a range of microscopic and macroscopic phenomena. A simple description of such interactions is given by the Ising model, where each $\vec{\mu}$ in a crystal is represented by a spin $\vec{S}$ taking values from $\{+1, -1\}$ on a regular discrete grid of points $\{i, j, k\}$. Furthermore, the interaction of the

spins $\{\vec{S}_i\}$ is considered only between nearest neighbours and represented by a constant $J_{i,j}$ which determines if the two neighbouring spins will tend to align parallel $J_{i,j} > 0$ or anti-parallel $J_{i,j} < 0$ with each other. Given a particular configuration of spin orientations ω, the energy of the system is then given by the Hamiltonian operator:

$$\hat{\mathcal{H}} = -\sum_{i,j} J_{i,j} \vec{S}_i \vec{S}_j - h \sum_i S_i \tag{5.9}$$

where h is an external magnetic field that tends to align the spins in a preferential orientation [9]. In this form, each $J_{i,j}$ defines a constraint $C_{i,j}$ between the values $D = \{+1, -1\}$ taken by the variables $\vec{S}_i$ and $\vec{S}_j$. It is easy to see that the more constraints are satisfied, the lower the value of $\hat{\mathcal{H}}$ becomes in Equation 5.9. This simple model allows the study of phase transitions between disordered configurations at high temperature and ordered ones at low temperature. For ferromagnetic $J_{i,j} > 0$ and antiferromagnetic $J_{i,j} < 0$ interactions the configurations are similar to those in Figure 5.22(d) and (e) for 3D lattices. These correspond to the stable states of our SNN solver when the Ising models for $J_{i,j} > 0$ and $J_{i,j} < 0$ are mapped to an SNN using Algorithm 5.1 and a 3D grid of 1,000 spins. Figure 5.22(g) shows the result for a 1D antiferromagnetic spin chain. It is interesting to note that the statistical mechanics of spin systems has been extensively used to understand the firing dynamics of SNNs, presenting a striking correspondence between their behaviour even in complex regimes. Our framework allows the inverse problem of mapping the SNN dynamics to spin interactions. This equivalence between dynamical systems and algorithms has largely been accepted and we see an advantage in computing directly between equivalent dynamical systems. However, it is clear that the network parameters should be adequately chosen in order to keep the computation valid.

If instead of fixing $J_{i,j}$ to some value U for all spin pairs $\{(i, j)\}$ one allows it to take random values from $\{U, -U\}$ with probabilities p_{AF} and p_{FM}, it will be found that certain interactions would be frustrated (unsatisfiable constraints). Figure 5.22(f) illustrates the frustration with three antiferromagnetic interacting spins in a way that any choice of orientation for the third spin will conflict with one or the other. This extension of the Ising model when the grid of interactions is a random mixture of AF and FM interactions was described by Surungan *et al.* [246]. The model is the representation of the spin glass systems found in nature; these are crystals with low concentrations of magnetic impurities that, due to the frustrated interactions, are quenched into a frozen random configuration when the temperature is lowered (at room or high temperature the magnetic moments of a material are constantly and randomly precessing around their average orientation).

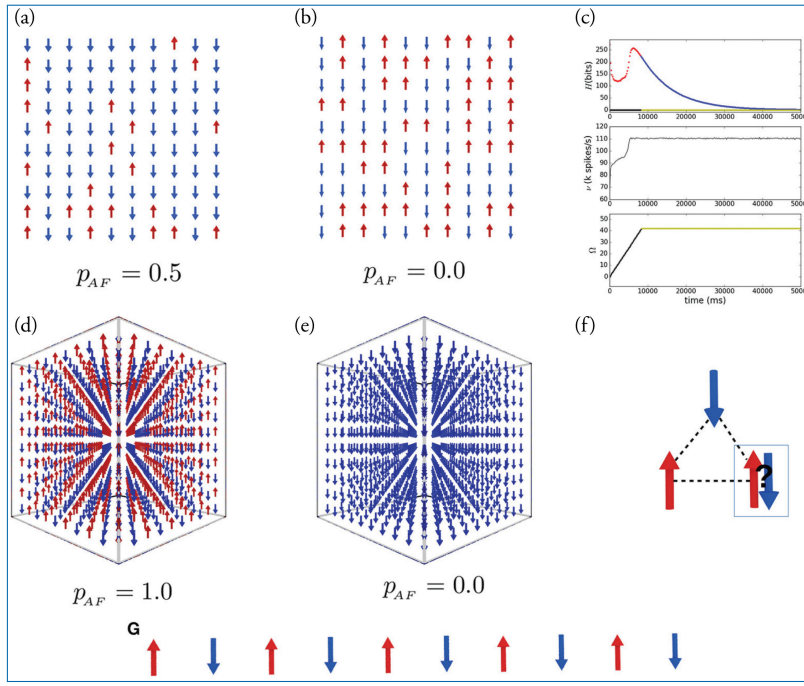

Figure 5.22. SNN simulation of Ising spin systems. (a) and (b) show two 2-dimensional spin glass quenched states obtained with interaction probabilities $p_{AF} = 0.5$ and $p_{AF} = 0.1$. The results for the three-dimensional lattices for CSPs of 1,000 spins with ferromagnetic and antiferromagnetic coupling constant are shown in (e) and (d), respectively. In (c) are plotted the temporal dependence of the network entropy H, firing rate ν and states count Ω during the stochastic search for the system in (d). (f) illustrates the origin of frustrated interactions in spin glasses. (g) depicts the result for the one-dimensional chain.

The statistical analysis of those systems was fundamental for the evolution of artificial neural networks and machine learning. Furthermore, the optimisation problem of finding the minimum energy configuration of a spin glass has been shown to be NP-complete [9]. The quenching of the grid happens when it gets trapped in a local minimum of the state space of all possible configurations. In Figure 5.22(a) and (b), we show a quenched state found by our SNN with $p_{AF} = 0.5$ and $p_{AF} = 0.1$, respectively. A spin glass in nature will often be trapped in local minima and will need specific temperature variations to approach a lower energy state; our SNNs replicate this behaviour and allow for the study of thermal processes, controlling the time variation and intensity of the excitatory and inhibitory stimulations. If the underlying stochastic process of such stimulations is a good representative of heat in solids, they will correspond to an increase and a decrease of

temperature, respectively, allowing, for example, the implementation of simulated annealing optimisation. Figure 5.22(c) shows the time evolution of the entropy, firing rate and states count for the antiferromagnetic 3D lattice of Figure 5.22(d). Similar plots, but converging to unsatisfying states, are found for the spin glasses in Figure 5.22(a) and (b). In the case of the ferromagnetic lattice in 5.22(e) with a very low noise, the network immediately converges to a solution. If the noise is high, however, it is necessary to stimulate the network several times to have a perfect ordering. This is because more noise implies more energy to violate constraints; even in nature, magnetic ordering is lost at high temperatures.