Understanding a Deep Learning Technique through a Neuromorphic System a Case Study with SpiNNaker Neuromorphic Platform

Indar Sugiarto[1,2], *and Felix* Pasila[2,*]

[1]School of Computer Science, University of Manchester, Manchester M13 9PL, United Kingdom
[2]Department of Electrical Engineering, Petra Christian University, Jawa Timur 60236, Indonesia

Abstract. Deep learning (DL) has been considered as a breakthrough technique in the field of artificial intelligence and machine learning. Conceptually, it relies on a many-layer network that exhibits a hierarchically non-linear processing capability. Some DL architectures such as deep neural networks, deep belief networks and recurrent neural networks have been developed and applied to many fields with incredible results, even comparable to human intelligence. However, many researchers are still sceptical about its true capability: can the intelligence demonstrated by deep learning technique be applied for general tasks? This question motivates the emergence of another research discipline: neuromorphic computing (NC). In NC, researchers try to identify the most fundamental ingredients that construct intelligence behaviour produced by the brain itself. To achieve this, neuromorphic systems are developed to mimic the brain functionality down to cellular level. In this paper, a neuromorphic platform called SpiNNaker is described and evaluated in order to understand its potential use as a platform for a deep learning approach. This paper is a literature review that contains comparative study on algorithms that have been implemented in SpiNNaker.

Key words: Deep learning technique, neuromorphic system, SpiNNaker

1 Introduction

In recent years, researches on artificial neural networks (ANNs) become a hype again, especially with the rise of learning engines equipped with "deep" learning capability. Many researchers, especially in the machine learning com-munity, consider deep learning (DL) as a second break-through for ANNs after many years of "silence" (not much progresses). The first breakthrough is attributed to the application of backpropagation (BP) algorithm for ANNs [1, 2]. Since then, no further significant progress on ANNs has been reported until recently.

Generally speaking, GP is a gradient-based approach applied as a means of learning for an ANN [3]. This is done by transforming the learning process into an optimization problem. In the optimization theory, gradient-based methods are well-known approaches to

* Corresponding author: indar@ieee.org, indar.sugiarto@manchester.ac.uk, felix@petra.ac.id

find a global optimum of a function. This is why the activation function of neurons in ANNs are expected to be di erentiable functions.

Since early 1970, GP has been used as a prominent tool for learning in ANNs. Both shallow and deep ANNs can be trained by using GP. However, due to computer hard-ware limitation at that time, only shallow ANNs were actively explored and deployed in real world applications. Only recently, when the computing power of computers becomes dramatically increased, especially with the advent of GPGPU (general purpose graphical processing unit) computing, deep ANNs are actively being explored [4]. This leads to the advancement of DL for practical pur-poses.

DL is a machine learning algorithm that relies on a many-layer network which exhibits a hierarchically non-linear processing capability. The idea of DL can be traced back from the work of Ivakhnenko [5] that produced a general learning algorithm for supervised feedforward Multi-layer Perceptrons (MLPs). It is commonly used by many layer networks, such as deep neural network (DNN), deep belief network (DBN), recurrent neural network (RNN), generative adversarial network (GAN), and convolutional neural network (CNN) [6–10]. Conceptually, DL is a generic technique and can be applied on other graphical models, for example deep reinforcement learning [11]. However, in this paper we focus on DL for ANNs, including spiking neural network (SNN) model.

DL is closely related to a class of theories of brain development, specifically neocortical development [12, 13]. Marblestone et. al. hypothesize that the brain optimizes cost functions, in which the cost functions are diverse and di er across brain locations and over development [12]. In this circumstance, the impressive performance demonstrated by DL as a credit assignment through multiple layers of neurons is in fact a manifesto of an optimization operation within a pre-structured architecture matched to the computational problems for specific classes. However, such interpretation does not show a direct relation between learn-ing mechanism in DL and in the brain. In particular, DL mechanism does not reflect brain dynamics even though its structure resembles the hierarchical neuronal circuitry of the brain cortex [14].

2 Overview of deep learning architectures

In this section, we describe some prominent ANN architectures where DL can be applied. These are architectures used by the second generation of ANN.

2.1 Deep neural network

Deep Neural Network (DNN) is a generalized feedforward neural network with several (hidden) layers. It is an advanced extension of the first generation of ANN, which was developed by Rosenblatt based on a single thresh-old logic unit neural network proposed by McCulloch and Pitts in 1943 [15]. This early simple ANN, known as “Perceptron”, soon became a larger network by adding more neurons and layers; thus creating a Multilayer Perceptron (MLP) [16, 17].

The structure of MLP is similar to DNN, and the application domains of MLP is as diverse as DNN. The most distinguishable feature of DNN from MLP is the learning process, especially during the initialization. Instead of using random number for initialising the weights, DNN can use other technique such as restricted Boltzmann machine (RBM) to find initial weight values that close to optimal. With this approach, vanishing/exploding gradients problem, which is common in “deeper” MLPs, can be avoided [18–20].

DNN learns to map a fixed-size input (e.g, images) to a fixed-size output (e.g, a probability for each of several categories). Currently, the rectified linear unit (ReLU) is the most popular non-linear activation function used in DNNs. It is much faster than conventional but smoother activation functions such as tanh or sigmoid. Within the hidden layers, ReLU creates distortion to the input in a non-linear way, making categories linearly separable at the output layer [21]. Figure 1 shows typical structure of a DNN for image recognition.

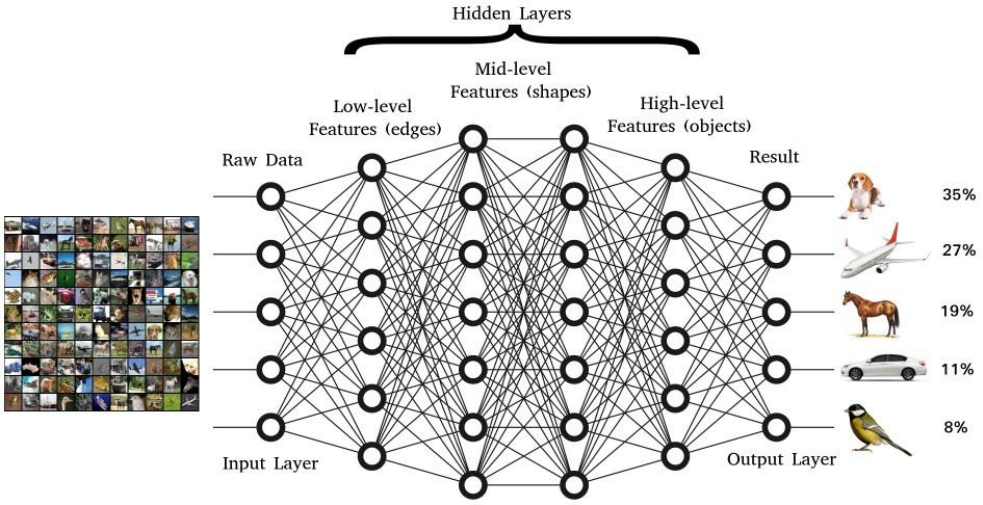

Fig. 1. A typical DNN structure for image/object recognition.

2.2 Deep recurrent neural network

Similar to DNN, a deep recurrent neural networks (DRNN) has several layers between its input and output layer. Conceptually, a DRNN is an extension to the standard recurrent neural network (RNN), whose neurons send feedback signals to each other exhibiting dynamic temporal behaviour. This feedback loop mechanism includes a huge number of possibilities, but conceptually can be classified into two classes: loops within a single layer, and loops between multiple layers. From these, several RNN structures have been proposed, such as fully recur-rent, recursive, Hopfield, Elman, echo state, long short-term memory, etc.

The temporal dynamic exhibited by RNNs leads into the memory capability that can be classified as short-term memory (STM), medium-term memory (MTM), or long-term memory (LTM) [22, 23]. These create the foundation of Adaptive Resonance Theory, which is the most advanced cognitive and RNN theory of how the brain autonomously learns to categorize, recognize, and predict objects and events [24].

A more practical example of DRNN is the long short-term memory (LSTM), which were invented by Hochreiter and Schmidhuber in 1997 [25]. This LSTM can have a very deep structure, allowing it to perform excellently in tasks such as speech recognition, handwritten recognition, and multilingual language processing [26].

An LSTM has several “gates” that control the feedback flow of information. A special “forget” gate which deletes the information in the self-recurrent unit without releasing the information into the network whilst making room for a new memory. An example structure of an LSTM is shown in Figure 2; it consists of four input weights (from the data to the input and three gates) and four recurrent weights (from the output to the input and the three

gates). Occasionally, peepholes are added as extra connections between the memory cell and the gates.

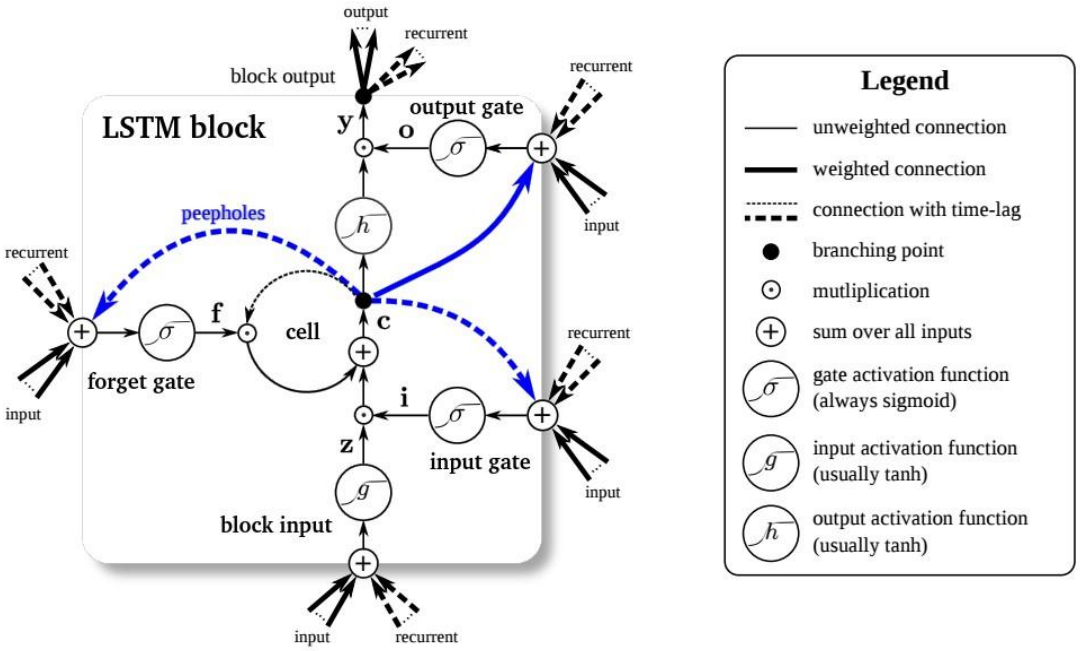

Fig. 2. An example of LSTM structure (modified from [27]).

2.3 Deep convolutional neural network

Another form of ANN that can have a deep structure is the Convolutional Neural Network (CNN). Basically, it is a feed-forward network that is designed to require minimal pre-processing [9, 28, 29]. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers are either convolutional, pooling or fully connected.

CNNs were inspired by biological processes in which the connectivity pattern between neurons is inspired by the organization of visual cortex [30]. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. This mechanism underlies the convolutional layer - the core building block of a CNN.

The convolutional layer has a set of kernels that operates as a small receptive field. Each kernel is convolved across the width and height of the input data, computing the dot product between them and resulting in two dimensional activation map of that kernel. As a result, the network learns kernels that activate when it detects some specific type of feature at some spatial position in the in-put. Stacking the activation maps for all kernels along the depth dimension forms the full output of the convolution layer, which can be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Another important layer in CNNs is the pooling layer. Basically it performs a non-linear down-sampling to reduce the spatial size of the representation as well as to reduce the number of parameters and amount of computation in the network. This way the network

can avoid overfitting. The pooling layers are periodically inserted between successive convolutional layers.

Finally, after a series of several convolutional and pooling layers, a fully connected layer is added at the final stage that is responsible for the high-level reasoning. Neurons in this final layer have connections to all activations in the previous layer. Fig. 3 shows the general structure of a DCNN.

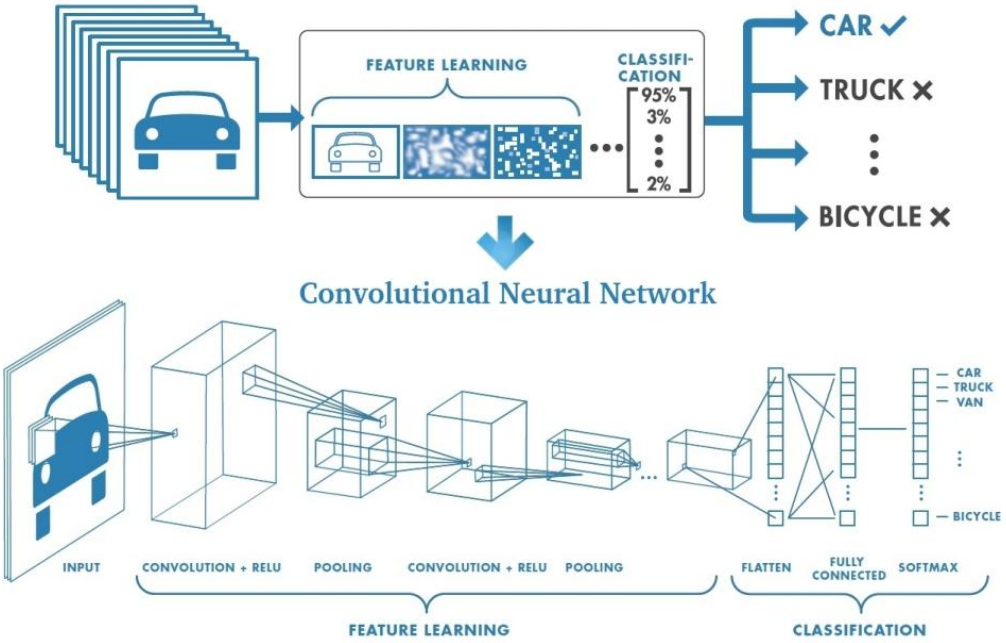

Fig. 3. The general structure of a DCNN (adopted from [31].

2.4 Deep belief network

Deep Belief Networks (DBNs) are another example of networks with a deep structure. They are composed of stacked smaller networks called restricted Boltzmann ma-chines (RBMs). RBMs are different to the standard ANNs; whereas ANNs use real scalar values for the weights, RBMs use probabilistic functions (usually in the form of conditional probability) for the weights [32]. The term "belief" is usually associated with the concept of generative model of the RBM, very similar to the Bayesian net-work in stochastic modelling [33].

Instead of using a stochastic ANN, a DBN can also be constructed by using a rather conventional network called autoencoder. Basically, an autoencoder is an MLP or feed-forward net with input and output tied together. This way, an autoencoder is supposed to be able to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction [34].

Either using RBMs or autoencoder, a DBN presents a hierarchical network where each sub-network's hidden layer serves as the visible layer for the next [7]. The learning mechanism of a DBN is performed layer-by-layer top-down with one layer at a time by treating the values of the latent variables in one layer, as the data for training the next layer.

DBNs have been used successfully in many applications such as in EEG (electroencephalography) analysis [35], generating and recognizing images [7, 36, 37], video sequences [38], and motion-capture data [39].

2.5 Deep generative adversarial

Generative Adversarial Network (GAN) is a neural net-work model for estimating generative models via an adversarial process. There are two models in the network contesting with each other in a zero-sum game framework: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. In his paper (see [10]), Goodfellow et.al use this technique to generate synthetic photographs that look authentic to human observers. In his GAN, the G model creates natural looking images that are similar to the original data distribution, whereas the D model determines whether the given image looks natural or looks like it has been artificially created. Here, the generator is trying to fool the discriminator while the discriminator is trying to not get fooled by the generator. As the models train through alternating optimization, both methods are improved until a convergence has been achieved (i.e., a point where the synthetic image is indistinguishable from the genuine one).

For some test cases, such as images of number or face, GAN produces good quality synthetic images. But for other test cases, such as natural scenes, GAN does not work well. To overcome this, GAN can be combined with other technique such as CNN to produce a deeper network [40], such as Laplacian Pyramid of Adversarial Network (LAPGAN) [41].

Even though the combination of CNN and GAN pro-duces higher accuracy than the original GAN, however, it still su ers from scaling problem: it does not work well to recover the finer texture details when using a large up-scaling factors. To overcome this problem, Ledig et.al proposes to uses an adversarial loss function in their Su-per Resolution Generative Adversarial Network (SRGAN) [42]. Figure 4 shows the deep structure of a GAN as used in [42].

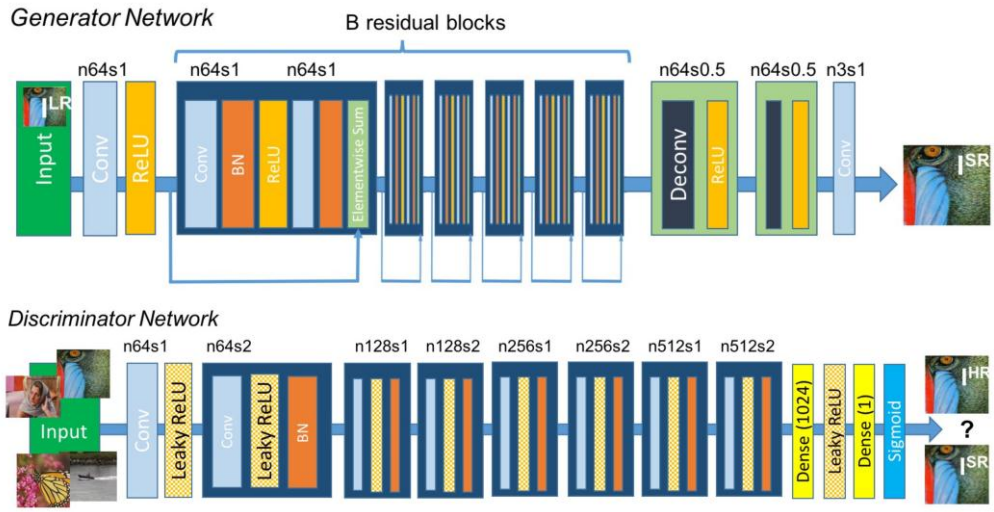

Fig. 4. SRGAN structure with corresponding number of fea-ture maps (n) and stride (s) indicated for each convolutional layer. (adopted from [42])

3 Overview of neuromorphic systems

This section focus on the implementation of the third generation of ANN known as the spiking neural net-work (SNN) [43]. SNNs leads into the development of neuromorphic computing, which tries to mimic neurobiological architectures present in the nervous system (i.e., the brain).

3.1 Connection with neuroscience

The first generation of ANN started with the work of Mc-Culloch and Pitts in 1943 on a computational model for neural networks based on threshold logic units. Since then, newer and more complex (and "deeper") networks have been proposed/implemented. However, those ANN models are more into mathematical models rather than biologically realistic models.

Neuroscientists argue that a more flexible and general AI could be built based on biologically plausible ANNs [44].Furthermore, in order to obtain more reciprocal benefits between neuroscience and machine learning, the integration of DL and neuroscience should be explored further [12]. For this reason, the third generation, spiking based, ANNs have been proposed.

In SNNs, the concept of time has been incorporated inherently, allowing them to produce a dynamic ANN model. Neurons will only generate out-put (i.e. "spike") when they have collected enough pre-information to surpass the internal threshold; hence, they do not "fire" at each propagation cycle as it hap-pens with conventional ANNs (e.g., MLP, DNN, etc.) /citemaass1997networks,gerstner1998spiking.

Maass shows that SNNs, with regard to the number of neurons that are needed, are computationally more pow-erful than conventional second generation of ANNs [43]. He shows that a single spiking neuron exhibits a biologically relevant function, which requires hundreds of hidden units on a sigmoidal-based ANN. Also, any function that can be computed by a small sigmoidal ANN can also be computed by a small SNN.

Prior to 1980s, most of ANNs and SNNs were conceptualized and implemented as software on computers, either on standard PCs or mainframes. With the advancement in chip fabrication technologies, researchers have a new option to implement ANNs and SNNs on dedicated hard-ware. In the late 1980s, Carver Mead started to look into neural-inspired hardware implementation [45] and coined the term neuromorphic. Neuromorphic systems strive to mimic neurobiological architectures present in the brain. Using analog VLSI technology in his work, Carver Mead demonstrated that biological solutions are many orders of magnitude more e ective than those that have been implemented using digital methods.

In recent years, however, the term neuromorphic has been used not only for describing analog-based neural-inspired hardware, but also for digital, mixed ana-log/digital VLSI, and even software systems that implement broader spectrum models of neural systems (from sensory level to cognitive processing) [46–48]. In this pa-per, we particularly describe scalable neuromorphic plat-forms.

3.2 Scalable neuromorphic platform

Taking advantage of chip fabrication technology, a number of large-scale neuromorphic systems have been developed recently, ranging from a single chip form to a full silicon wafer. Those systems are designed with scalability feature to make them able to support a large scale neural network model with millions of neurons along with their tremendous and complex synaptic connectivity [49]. This feature is inherently required in order to build a

massive network for emulating a whole brain of animals ranging from nematode species (such as *Caenorhabditis elegans*, or C. *elegans*) up to smaller mammals [50, 51]. The scalability is also one of the main requirements to develop a framework with cognitive architecture. The ultimate goal of such plat-forms is to model a complete human brain [52]. However, this is a long-term goal and is treated with cautious optimism by scientific community, since the full potential of those neuromorphic systems has yet to be realized.

Four of the most prominent scalable neuromorphic platforms to date are: the IBM TrueNorth chip, the Stan-ford Neurogrid, the Heidelberg BrainScaleS system, and the Manchester SpiNNaker machine [53–56]. These plat-forms are built with complementary approaches and diver-gent intermediate goals. Table 1 summarizes some of the main features of these neuromorphic platforms.

3.3 SpiNNaker many-core platform

SpiNNaker (Spiking Neural Network Architecture) is a many-core neuromorphic platform built on top of commercial general purpose ARM microprocessor. At the lowest level, it is a multi-core system-on-chip (SoC) composed of 18 ARM968 processors and equipped with 128 MB low-power SDRAM mounted on top of the processor die.

Table 1. Main features of four large-scale neuromorphic platforms.

Platform	Mode	Chip Fabrication	Power per Chip	Neuron Model	Synapse Model	Plasticity
IBM TrueNorth	Digital	28nm	72 mW	LIF	Binary	No
Neurogrid	Analog	180nm	150 mW	IF	Shared dendrite	No
BrainScaleS	Analog	180nm	1.3 W	IF	4-bit Digital	STDP
SpiNNaker	Digital	130nm	1W	Programmable	Programmable	Programmable

The chip incorporates various digital elements for con-trolling the processors as well as for providing neurally-inspired communication infrastructure. The SpiNNaker uses small size data packets in various protocols, such as multicast, point-to-point, and nearest-neighbourhood. Multicast is the most used communication protocol in SpiNNaker for propagating neuron spikes in neural net-work models. It uses a packet switched network to emu-late the very high connectivity of biological systems. The packets are source-routed, i.e., they only carry information about the issuer and the network infrastructure is responsible for delivering them to their destinations.

Each SpiNNaker chip has six bidirectional links that can be connected to other chips. The links incorporate asynchronous digital circuitry to achieve the optimal band-width whilst maintaining the lowest power consumption. Using these links, 2D triangular mesh topology can be created by connecting several SpiNNaker chips. If the number of chips is su cient enough (at least 144 chips) and full-links connection is employed, then a 3D torus net-work will be created (see Figure 5(a)). Currently, the complete SpiNNaker system development is underway; when it is complete, it will have 50k chips, in which 1 million ARM cores are available for neuromorphic computing, as well as 7 Tera bytes of memory needed for simulating a massive neural network in biological real-time.

From SpiNNaker chips, several type of SpiNNaker machines are constructed. For the sake of naming simplic-ity, SpiNNaker machines are classified by the approximate number of processor cores in the format of 10x, where x is the power number for the base 10. For example, the 102 and 103 machines, which are single printed circuit boards, actually contains 72 cores and 864 cores respec-tively. The final 106 machine, will have 1036800 proces-sor cores. Figure 5(b) shows the physical appearance of several SpiNNaker machines.

SpiNNaker uses a simple event-driven model program-ming model. Since SpiNNaker is developed for a specific application (e.g. SNN), it does not require any general pur-pose operating system to be run on the platform. Instead, it only has a small dedicated run-time kernel to manage the entire operation of the machine. The SpiNNaker Application Run-time Kernel (SARK) controls the flow of exe-cution and schedules/dispatches application callback functions. Applications do not control execution flow, they can only indicate the functions, referred to as callbacks, to be executed when specific events occur, such as the arrival of a neural packet, the completion of a Direct Memory Access (DMA) transfer or the lapse of a periodic time interval.

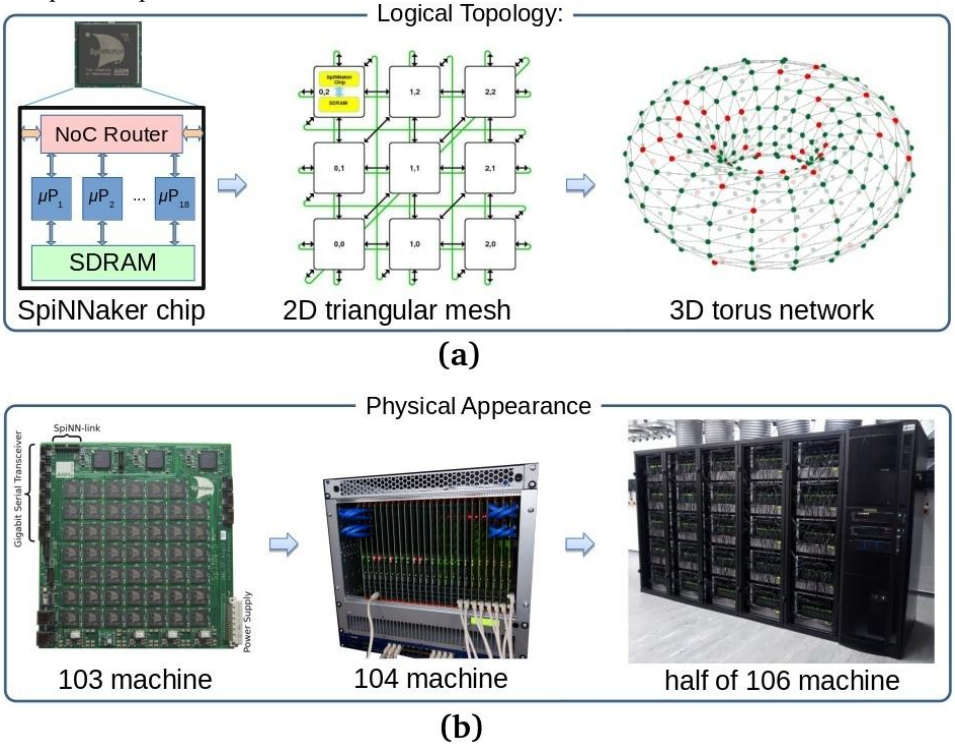

Fig. 5. SpiNNaker architecture in (a) logical topology, and (b) physical appearance.

4 Case study with SpiNNaker

In this paper, several experiments on SpiNNaker are studied. The publications related to those experiments are explored in order to understand how SpiNNaker can be used to implement DL concepts as well as to gain some insight on problems faced during SpiNNaker implementation of the corresponding aspect.

The main issue a network modeller instantly faces when using SpiNNaker is the fact that SpiNNaker runs on the basis of spiking neuron model; whereas common DL algorithms assume that the network falls into the category of second generation ANN. However, since SpiNNaker machines are built on top of a general purpose processor, it is still possible, using the direct programming approach on SpiNNaker, to implement a DL concept on SpiNNaker. This approach proves that SpiNNaker can also be used for general purpose computing, even in a domain far from spiking neural network, such as in image process-ing [57]. The following sub-sections provide examples of applications that are closely related to the concept of DL.

4.1 DNN on SpiNNaker

The vast amount of processor in SpiNNaker is definitely an attractive resources for implementing a DNN. As de-scribed in Section 2.2, basically a DNN is a feed forward ANN with many layers. Like SNNs, connections between neurons are through adaptable synapses (or "weights"). DNN models, however, have important di erences. Typically weights are only plastic for a certain time, that is during the "training phase", and then are fixed afterwards (or during the "test phase"). Each phase consists of some number of examples, individual data items presented to the network. This in turn implies that the DNN has no time model, or to be exact, a discrete-time model. Contrasting strongly with SNNs: the DNN model thus has a synchronous dataflow. In addition, the most popular learning technique is backpropagation, a method that involves propagating errors from output units backwards through the network to update the weights. Both of these proper-ties present challenges to the SpiNNaker architecture.

To solve the synchronous dataflow problem, a new technique called "update-on-demand" was proposed [58]. In this technique, partial results are computed and for-warded on through the network as soon as they are avail-able. The backpropagation algorithm, which presents challenges for the source-routed SpiNNaker topology be-cause it is e ectively bidirectional, is solved using a matrix remapping method that distributes the processing between cores.

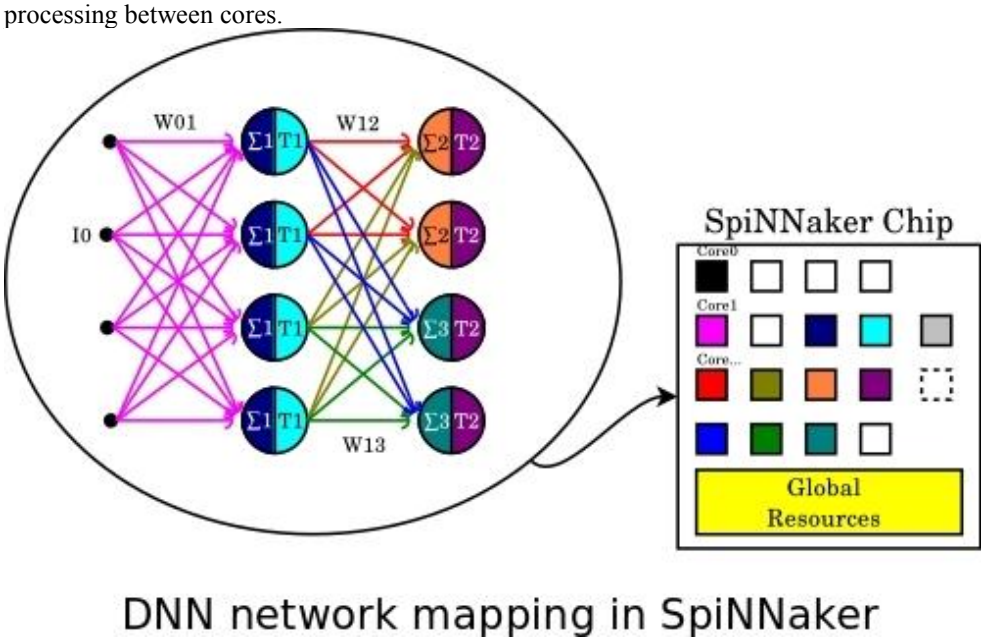

Fig. 6. Implementing a DNN on SpiNNaker. At each stage the unit sums the contributions from the previous stage. One processor may implement the input path for more than one final output neuron (shown here for the threshold stage).

The processing in a DNN model remains event-driven. Each processor in the DNN responds to a single hardware event (packet-received) and schedules software-generated events to complete processing. The packet-received event performs only 2 tasks: i) it places the packet into an internal service queue; ii) it schedules a deferred event to dequeue and process the packet. The dequeue software event, having retrieved the packet, performs the address decode and data processing required, as per each stage.

The SpiNNaker implementation models DNN networks using the Lens (http://tedlab.mit.edu/dr/Lens) specification and provides support for both feed forward and

re-current networks. Initial support is provided for networks defined at a Group level; future plans may include sup-port for Unit-level definitions. Full support is provided for Group-to-Unit and Unit-to-Unit connectivity. A SpiNNaker package plugin for Lens allows Lens scripts to be implemented directly on SpiNNaker using the SpiNNaker software tool-chain.

4.2 CNN on SpiNNaker

Conventional CNNs as described in Section 2.3 work in a frame-based fashion, where the data (e.g., image) is acquired and presented to the network at a given frame rate (at about 30ms to 40ms). Each frame is processed sequentially by the different convolution layers and it must be completely finished before the processing of the next layer can be started. However, this is not how the brain works. In the brain, when a pixel in the retina is stimulated, it emits a spike with a very small propagation delay to neurons of the next layers. Neurons in the next layer will emit an-other spike when they have received enough spikes from the previous layer. This process continues up to the higher recognition layer levels.
To implement such a more biologically plausible CNNs on SpiNNaker, an event-based CNN for poker card symbol classification has been proposed [59], which is based on the work in [60]. It consists of four convolution layers (C1-C3-C5-C6) interleaved with two subsampling stages (S2-S4). Figure 7 shows the architecture of the event-based CNN.

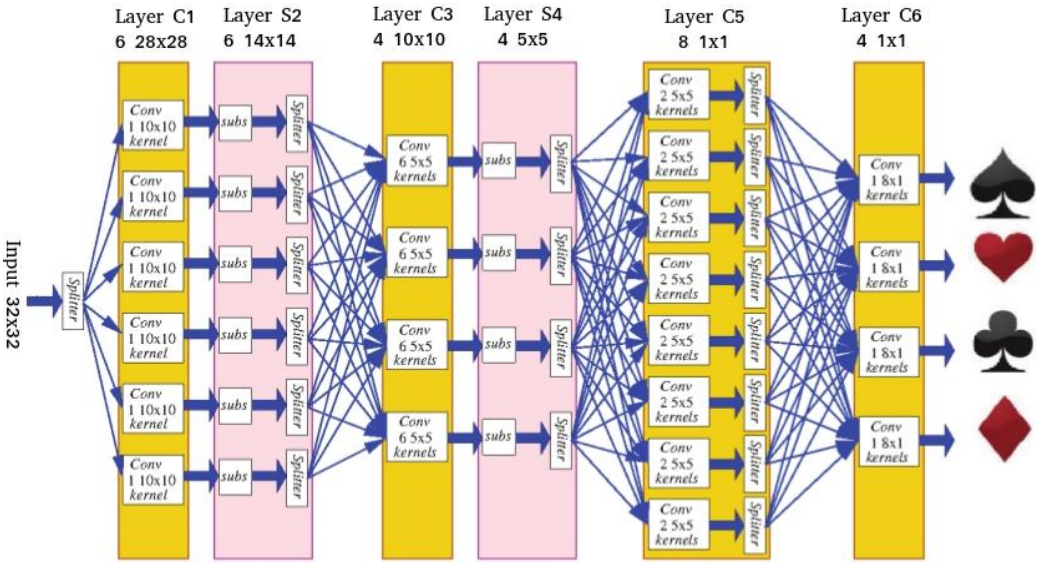

Fig. 7. The architecture of a CNN for the poker card symbol classification in [59].

The first convolution layer implements Gabor filters at three different orientations and two spatial scales (their weights were not trained). The rest of the network weights were obtained by backpropagation, training a frame-driven CNN version and afterwards applying a conversion method to obtain the corresponding parameters for the event-driven CNN [60].

One important element of a CNN is the kernel. The kernels possess the "weight sharing" property, where the weights of the kernels that connect neurons in two consecutive feature maps do not depend on the particular neuron positions but just on the relative

positions of the two neurons in the origin and destination feature maps. To optimize the processing speed, the author in [59] introduced a special "convolution connector" to the SpiNNaker soft-ware tool. The convolution connector contains the kernel weights which are stored in the processor's local memory of the corresponding neuron population.

To test the recognition success rate, the author used a test sequence of 40 32×32 tracked symbols obtained from a very high speed event-driven camera. In a slow-mode experiment, a total of 189 category output events were generated, and the recognition success rate was 97.5 %. How-ever, when the processing speed was increased by using a slowdown factor of 10 for the same input stimulus event sequence, the success recognition rate decreases to 80 % and the total number of output events generated by the net-work has also decreased to 121 events.

Another finding by the authors was the source of the bottleneck in the processing flow that limits the maximum event throughput. The bottleneck happens in the first layer which employs the Gabor filters. Each Gabor filter receives all the events from the input stimulus, while generating each a significant number of events, which need to be transmitted and processed by the second layer. Unfortunately, there is no solution for this problem yet, which leaves a future work for those who are interested on this topic.

4.3 DBN on SpiNNaker

DBNs, as described in Section 2.4, can be implemented in several fashions. One of them that has been implemented on SpiNNaker is the model using RBM [61], which basically implements a spike-based DBN proposed by [62]. Since RBM is a stochastic model, the most convenient method to approach it in a spiking neuron models is by using firing rate. In this model, the Siegert approximation was used for the activation function of neurons. The Siegert function (see for detail in [63]) produces firing rate of a neuron given input firing rates and input weights for certain neuron parameters. For representing the weight, the author used 16 bits for the fractional part and 6 bit for the integer part (i.e., using the format Q6.16).

For testing, the MNIST dataset [28] was used as a classification benchmark. The MNIST dataset consists of 28×28 grey-scale pixel images of handwritten digits and is divided into a training set, which comprises 60 000 digits, and a test set, which consists of 10 000 digits. The network that consists of 4 layers were used: one input layer with 784 neurons, two hidden layers with 500 neurons each, and a 10 neuron output layer (see Figure 8(b)).

Static images are transformed into spike trains by converting each pixel of an MNIST image into a Poisson spike-train with a rate proportional to its intensity, while all firing rates are scaled such that the total firing rate of the population is constant [62]. The firing rate of neuron populations are depicted in Figure 8(c).

For the evaluation purpose, the authors used the classification accuracy and the latency as performance metrics. Regarding the classification result, their model achieved 95.01 % accuracy when using Q6.16 format for the weight values. When measuring the real-time performance of their model, the author found a mean classification latency of 20 ms. This was expected since the timing used for solving neuron equations in SpiNNaker was set to 1 ms, which is the default value in SpiNNaker configuration. The experiment also showed that increasing the number of input spikes reduces the mean classification latency, and has a positive e ect on the classification accuracy. The latency of the hidden layers also varies with the number of input spikes. For the MNIST test set encoded with 2 000 spikes per second, the mean spike latency between the hid-den layers is 2.3 ms; whereas in 500 spikes per second ex-periment, the mean latency per hidden layer rises to 6.22 ms.

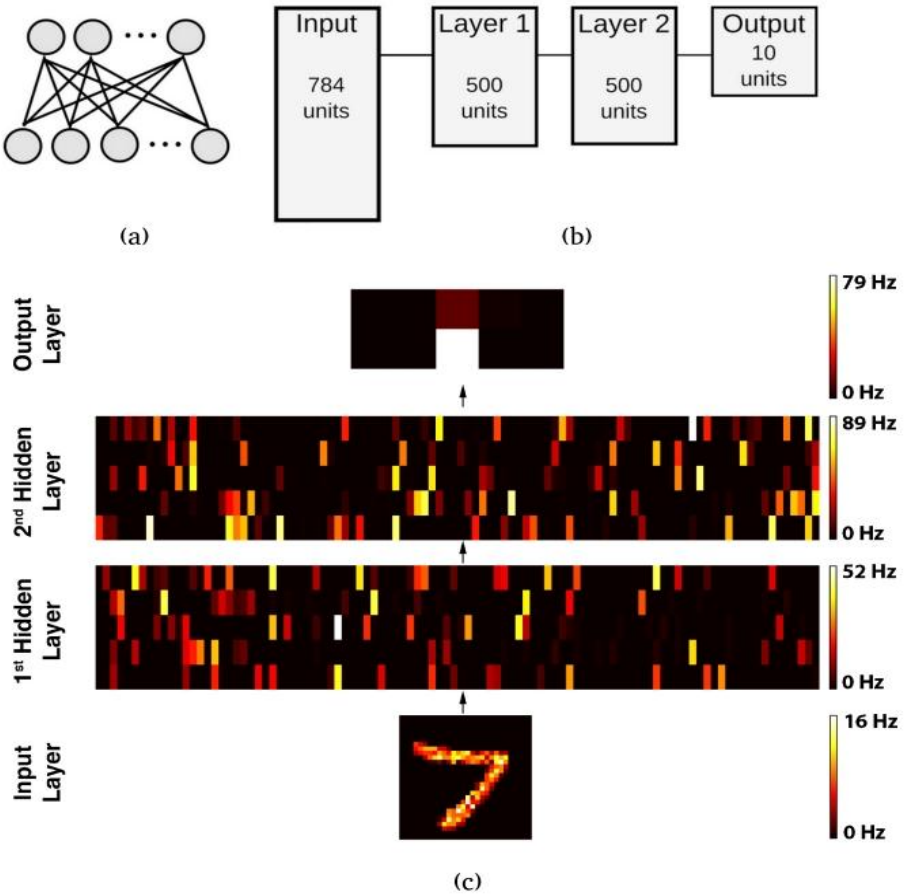

Fig. 8. DBN experiment on SpiNNaker as reported in [61]. In (a), a general architecture of RBM used as a layer for DBN. In (b), a topology of DBN in 784-500-500-10 structure used as a MNIST classification network. In (c), the firing rates of the network was under investigation for a single input MNIST digit. The bottom plot shows firing rates of the 28×28 neurons in the input population. The next two rows of 5×100 show the firing rates of the neurons in the first and second hidden layer (500 neurons each), and finally the top plot shows the firing rates of the 10 neurons in the output population, one for each digit from 0 to 9.

5 Discussion

From the study in the previous section, we gain several in-sight on how useful and challenging is the SpiNNaker for application in the domain of machine learning and artificial intelligence. As described in Section 4.1, it is fairly straight forward to implement feed-forward DNNs. The current SpiNNaker software tool-chain supports, at some degrees, modelling of general purpose DNNs. If specific application constraints are needed, however, users need to go a bit deeper by modifying the software tool-chain flow. Fortunately, SpiNNaker is programmed using stan-dard C/C++ and Python, which are common language pro-gramming known by most users.

In Section 4.2, we have described how the SpiNNaker platform can be used to implement a CNN. As proposed by its developer, a special “weight sharing” mechanism can be implemented e ciently in SpiNNaker by storing all synaptic weights on local memory called DTCM (data-tightly-coupled-memory) at each processor core [59]. This

way, the memory access efficiency is increased consider-ably that leads into the increasing processing speed of the CNN.

Even though SpiNNaker provides a convenience way for handling the CNN parameters, overall, the efficiency of implemented CNN model is determined by the maxi-mum number of addressable neurons, limited by the implemented routing scheme of the current SpiNNaker soft-ware tool-chain. Currently, for small size CNNs, this limitation is not a problem. But for larger networks, this might impose reduced performance. Hence, in the future, the SpiNNaker developer needs to develop a more flexible neural addressing routing scheme.

As emphasized by authors in [59], the SpiNNaker architecture is very flexible and it allows to trade o maximum number of neurons per core versus maximum event processing throughput. By eventually reducing the maximum number of neurons per core, one effectively achieves parallelization of event processing, and the system would be capable of handling higher input event rates. This suggests that in the future SpiNNaker software release, the trade-o between maximum neurons per core and maxi-mum event throughput should be optimized according to the given desired parameters by the SpiNNaker user.

In Section 4.3, we describe how SpiNNaker was used to implement an event-driven DBN. There is an issue regarding the implementation of DBNs on SpiNNaker: both training and execution of large-scale DBN require vast computing resources, leading to high power requirements and communication overheads. To overcome this problem, authors in [61] proposed to limit bit precision during execution and training. Even though this idea sounds contra-intuitive, the experiment showed that the spiking DBNs can tolerate very low levels of hardware bit precision down to almost two bits. By limiting the bit precision, the performance of spiking DBNs can be improved by at least 30 %.

In addition to the improved performance, the authors also noticed the very promising feature of SpiNNaker with regard to scalability issue. It is well known that adding more layers to DBNs can improve performance. With the SpiNNaker architecture it becomes possible to create very large DBNs by adding additional layers, running on different cores or chips, without significantly increasing the latency of the system, and at reasonable power dissipation. This suggests that future work on DBNs should explore the implementation of larger DBN networks on bigger SpiNNaker machines. While the SpiNNaker hard-ware might not achieve the energy performance of dedicated neuromorphic hardware, the programmability of its architecture makes it an excellent exploration platform for event-based computing [61].

Finally, as shown in Table 1, some features of SpiNNaker are identified as “programmable”. This has consequences of bringing both cons and pros upon the SpiNNaker system. The most obvious advantage of being programmable is the high degree of flexibility in terms of novel modelling purpose. For example, modellers can specify what kind of neurons they want to use in the model, simply by modifying the neuron parameters. Or, changing the synapse behaviour by specifying the plasticity rule. As mentioned by its developer [56], the SpiNNaker is meant to be a generic neuromorphic platform for novel research in neuroscience and it’s following application domains (such as artificial intelligence). With this idea in mind, the SpiNNaker provides a very interesting alternative to the existing neuromorphic technology.

6 Conclusion

In recent years, Deep Learning (DL) has been considered as a breakthrough technique in the field of artificial intelligence and machine learning. Conceptually, it relies on a many layer network that exhibits a hierarchically non-linear processing capability; hence

mimicking the underlying topology of human brains. However, this claim needs to be clarified since there are some degree of incompatibility between DL and cognitive feature of the brain.

In this paper, we are aiming to understand better how actually DL techniques fit our current knowledge of how the brain works. To achieve the goal, we are looking into the direct implementation of cognitive machine in a form of neuromorphic systems. For the study, we use SpiNNaker references that have close relationship with the DL techniques. SpiNNaker is a neuromorphic platform for simulation of a massive spiking neural network. After exploring several references, we confirm that to use it for general purpose applications in deep learning frame-works, software and hardware adaptation is needed. Some DL-based architectures have been implemented on SpiNNaker: DNN, CNN and DBN. Indeed, the results from those implementations show some promising features in terms of performance and scalability.

The authors would like to thank the Institute of Research and Community Development, Petra Christian University, Indonesia, for supporting our work through the project grant 02/LPPM/V/2017. The authors also would like to thank the Advanced Processor Technology (APT) group of the School of Computer Science at the University of Manchester, United Kingdom, for the opportunity to conduct the study and the discussion with the SpiNNaker team as well as with the experts in the field of neuromorphic technology.

References

1. P.J. Werbos, The roots of backpropagation: from ordered derivatives to neural networks and political forecasting. 1st Edition. John Wiley & Sons, Inc.: USA (1994). https://www.amazon.com/Roots-Backpropagation-Derivatives-Forecasting-Communications/dp/0471598976
2. D.E. Rumelhart, G.E. Hinton, R.J. Williams, Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985)
3. S. Dreyfus. *Journal of Mathematical Analysis and Applications* **5**, 1:30–45 (1962). https://www.sciencedirect.com/science/article/pii/0022247X62900045
4. K.S. Oh, K. Jung. *Pattern Recognition* **37**, 6:1311–1314 (2004). https://www.sciencedirect.com/science/article/pii/S0031320304000524
5. A.G. Ivakhnenko, V.G. Lapa, Tech. rep., *Purdue Univ Lafayette Ind School Of Electri-Cal Engineering* (1966)
6. D.C. Ciresan, U. Meier, L.M. Gambardella, J. Schmidhuber, Neural computation **22**, 12:3207–3220 (2010). https://www.mitpressjournals.org/doi/full/10.1162/NECO_a_00052
7. G.E. Hinton, S. Osindero, Y.W. Teh, *Neural computation* **18**:1527–1554 (2006). http://www.cs.toronto.edu/~fritz/absps/ncfast.pdf.
8. A. Graves, A. Mohamed, G. Hinton. *Speech recognition with deep recurrent neural networks*. International Conference on Acoustics, Speech and Signal Processing (ICASSP) (Vancouver, Canada, 2013). ICASSP: 6645– 6649 (2013. http://ieeexplore.ieee.org/document/6638947/
9. A. Krizhevsky, I. Sutskever, G.E. Hinton. *Imagenet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems 25 (Lake Tahoe, Nevada, 2012). NIPS:1097–1105 (2012). https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

10. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio. *Generative adversarial nets*. Advances in Neural Information Processing Systems 27 (Montreal, Canada, 2014). NIPS:2672–2680 (2014). https://papers.nips.cc/paper/5423-generative-adversarial-nets
11. M. Hutter. *Universal artificial intelligence: sequential decisions based on algorithmic probability*. Springer: Berlin (2005). https://www.amazon.com/Universal-Artificial-Intelligence-Algorithmic-Probability/dp/3540221395
12. A.H. Marblestone, G. Wayne, K.P. Kording, *Front. Comput. Neurosci.* **10**, 94 (2016). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5021692/
13. J.L. Elman, Rethinking Innateness: A Connectionist Perspective on Development (MIT Press, 1998). https://mitpress.mit.edu/books/rethinking-innateness
14. R.J. Douglas, K.A. Martin. *Annu. Rev. Neurosci.* **27**, 419–451 (2004). http://www.allpsych.uni-giessen.de/rauisch/readings/Douglas&Martin_2004.pdf.
15. M. Anthony. *Discrete mathematics of neural networks: selected topics*. SIAM: USA (2001). https://books.google.co.id/books?id=qOy4yLBqhFcC&printsec=frontcover&dq=Discrete+Mathematics+of+Neural+Networks:+Selected+Topics&hl=en&sa=X&ved=0ahUKEwjtlfj3mp7ZAhVGQY8KHfK2BscQ6AEIKDAA#v=onepage&q=Discrete%20Mathematics%20of%20Neural%20Networks%3A%20Selected%20Topics&f=false
16. P. Auer, H. Burgsteiner, W. Maass. *Neural Networks* **21**, 786–795 (2008). https://igi-web.tugraz.at/people/maass/psfiles/126.pdf.
17. B. Widrow, M.A. Lehr. *Proceedings of the IEEE* **78**, 9:1415–1442 (1990). http://ieeexplore.ieee.org/document/58323/
18. S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber. Gradient flow in recurrent nets: the di culty of learning long-term dependencies [Online] from https://pdfs.semanticscholar.org/aed0/54834e2c696807cc8b227ac7a4197196e211.pdf. (2001). [Accessed on]
19. J. Schmidhuber, Learning 4 (2008)
20. G.E. Hinton. *What kind of a graphical model is the brain?*. The 19th Proceeding International Joint Conference on Artificial Intelligence (Edinburgh, Scotland, 2005). IJCAI'05:1765–1775. https://www.semanticscholar.org/paper/What-kind-of-graphical-model-is-the-brain-Hinton/a4a5bef06587350604c7a9857ca09d91bd95763e
21. Y. LeCun, Y. Bengio, G. Hinton. *Nature* 521:436–444 (2015). Nature 521, 436
22. I. Boardman, D. Bullock. *A neural network model of serial order recall from short-term memory*. IJCNN-91-Seattle International Joint Conference on Neural Networks (Seattle, USA, 1991). IJCNN-91-Seattle International Joint Conference on Neural Networks, **2**:879– 884 (1991). http://ieeexplore.ieee.org/document/155450/
23. S. Grossberg. *Proc. Natl. Acad. Sci. U.S.A.* **60**, 3:758–765 (1968). https://www.ncbi.nlm.nih.gov/pubmed/5243922
24. S. Grossberg. *Neural Networks* 37:1–47 (2013). https://www.sciencedirect.com/science/article/pii/S0893608012002584
25. S. Hochreiter, J. Schmidhuber. *Neural Computation* **9**, 8:1735–1780 (1997). https://dl.acm.org/citation.cfm?id=1246450
26. J. Schmidhuber. *J Neural Networks* **61**:85–117 (2015). https://arxiv.org/abs/1404.7828
27. K. Gre , R.K. Srivastava, J. Koutník, B.R. Ste-unebrink, J. Schmidhuber, CoRR abs/1503.04069 (2015)

28. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. *Proceedings of the IEEE* **86**, 11:2278–2324 (1998). http://ieeexplore.ieee.org/document/726791/
29. P.Y. Simard, D. Steinkraus, J.C. Platt. *Best practices for convolutional neural networks applied to visual document analysis*, Seventh International Conference on Document Analysis and Recognition (Edinburgh, UK, 2003). ICDAR **3**:958–962 (2003). http://ieeexplore.ieee.org/document/1227801/?denied
30. M. Matsugu, K. Mori, Y. Mitari, Y. Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. International Joint Conference on Neural Networks, (Portland, Oregon 2003). Neural Networks **16**:555–559(2003) https://pdfs.semanticscholar.org/6dd4/40eec6718f29a35b464345df2c77a4f0e085.pdf
31. The MathWorks, Convolutional neural network, [Online] from https://uk.mathworks.com/discovery/convolutional-neural-network.html. [Accessed September 30th 2017].
32. A. Fischer, C. Igel. Pattern Recognition **47**,1:25–39(2014) https://www.sciencedirect.com/science/article/pii/S0031320313002495
33. I. Sugiarto, J. Conradt. Discrete belief propagation network using population coding and factor graph for kinematic control of a mobile robot. IEEE International Conference on Computational Intelligence and Cybernetics (CYBERNETICSCOM), (Yogyakarta, Indonesia 2013). IEEE Xplore:136–140(2014). http://ieeexplore.ieee.org/document/6865797/
34. C.Y. Liou, W.C. Cheng, J.W. Liou, D.R. *Liou. Neurocomputing*, **139**:84–96 (2014) https://www.sciencedirect.com/science/article/pii/S0925231214003658?via%3Dihub
35. F. Movahedi, J.L. Coyle, E. Sejdic. IEEE Journal of Biomedical and Health Informatics, PP,99:1(2017) http://ieeexplore.ieee.org/document/7981315/
36. M. Ranzato, F.J. Huang, Y.L. Boureau, Y. LeCun. Unsupervised learning of invariant feature hier-archies with applications to object recognition. IEEE Conference on Computer Vision and Pattern Recognition, (Minneapolis, USA 2007). IEEE Xplore:1–8(2007). http://ieeexplore.ieee.org/document/4270182/?part=1
37. Q.V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean, A.Y. Ng, Building high-level features using large scale unsupervised learning. Appearing in Proceeding of the 29th International Conference on Machine Learning (Scotland, UK 2012). https://arxiv.org/pdf/1112.6209.pdf
38. I. Sutskever, G. Hinton. Learning multilevel dis-tributed representations for high-dimensional se-quences. In Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, (San Juan, Puerto Rico 2007). PMLR 2:548-555, 2007. http://proceedings.mlr.press/v2/sutskever07a.html
39. G.W. Taylor, G.E. Hinton, S.T. Roweis. Modeling human motion using binary latent variables. In Neural Information Processing Systems Conference on Advances in Neural Information Processing Systems, (Vancouver, Canada 2006). p.1345–1352(2006). https://papers.nips.cc/paper/3078-modeling-human-motion-using-binary-latent-variables.pdf
40. A. Radford, L. Metz, S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In International Conference on Learning Representations, (San Juan, Puerto Rico 2016). https://arxiv.org/pdf/1511.06434.pdf
41. E.L. Denton, S. Chintala, A. Szlam, R. Fergus. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. Advances in Neural Information

Processing Systems 28 (Montreal, Canada, 2015). NIPS:1486–1494. https://dl.acm.org/citation.cfm?id=2969405

42. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cun-ningham, A. Acosta, et al. [Online] from https://arxiv.org/abs/1609.04802 (2016). [Accessed on]
43. W. Maass. *Neural networks* **10**, 9:1659–1671 (1997). https://www.sciencedirect.com/science/article/pii/S0893608097000117
44. A.J. Bell, Philosophical Transactions of the Royal Society of London B: Biological Sciences 354, 2013 (1999)
45. C. Mead. *Proceedings of the IEEE* **78**, 10:1629–1636 (1990). http://ieeexplore.ieee.org/document/58356/
46. R.J. Douglas, M.A. Mahowald, K.A.C. Martin. *Hybrid analog-digital architectures for neuromorphic systems*. Proceeding IEEE Conference on Neural Networks (Orlando, USA, 1994). Proceeding IEEE Conference on Neural Networks **3**:1848–1853. http://ieeexplore.ieee.org/abstract/document/374439/
47. E. Chicca, F. Stefanini, C. Bartolozzi, G. Indiveri. *Proceedings of the IEEE* **102**, 9:1367–1388 (2014). http://ieeexplore.ieee.org/document/6809149/
48. S.C. Liu, T. Delbruck, G. Indiveri, A. Whatley, R. Douglas. Event-based neuromorphic systems. John Wiley & Sons: India, (2015)
49. E.M. Izhikevich. *IEEE Transactions on Neural Networks* **15**, 5:1063–1070 (2004). http://ieeexplore.ieee.org/document/1333071/
50. B. Szigeti, P. Gleeson, M. Vella, S. Khayrulin, A. Palyanov, J. Hokanson, et al. *Front Comput Neurosci* **8**, 137 (2014). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4217485/
51. E.M. Izhikevich, G.M. Edelman. *PNAS* **105**, 9:3593–3598 (2008). http://www.pnas.org/content/105/9/3593
52. K. Amunts, C. Ebell, J. Muller, M. Telefont, A. Knoll, T. Lippert. *Neuron* **92**, 3:574–581 (2016). https://www.ncbi.nlm.nih.gov/labs/articles/27809997/
53. P.A. Merolla, J.V. Arthur, R. Alvarez-Icaza, A.S. Cassidy, J. Sawada, F. Akopyan, et al., *Science* **345**, 6197:668–673 (2014). http://science.sciencemag.org/content/345/6197/668
54. B.V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A.R. Chandrasekaran, J.M. Bussat, et al. *Proceedings of the IEEE* **102**, 5:699–716 (2014). http://ieeexplore.ieee.org/document/6805187/
55. J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, S. Millner. *A wafer-scale neuromorphic hardware system for large-scale neural modelling*. 2010 IEEE International Symposium on Circuits and Systems (Paris, France, 2010). ISCAS:1947–1950. http://ieeexplore.ieee.org/document/5536970/
56. S.B. Furber, F. Galluppi, S. Temple, L.A. Plana. *Proceedings of the IEEE* **102**, 5:652–665 (2014). http://ieeexplore.ieee.org/document/6750072/
57. I. Sugiarto, G. Liu, S. Davidson, L.A. Plana, S.B. Furber. *High performance computing on SpiNNaker neuromorphic platform: a case study for energy efficient image processing*, 2016 IEEE 35th International Performance Computing and Communications Conference. IPCCC:1–8. http://ieeexplore.ieee.org/document/7820645/
58. X. Jin, M. Luján, L.A. Plana, A.D. Rast, S.R. Welbourne, S.B. Furber. *Effcient parallel implemen-tation of multilayer backpropagation networks on SpiNNaker*. Proceedings of the 7th ACM International Conference on Computing Frontiers (Bertinoro, Italy,

2010 Proceedings of the 7th ACM International Conference on Computing Frontiers :89–90. https://dl.acm.org/citation.cfm?id=1787297

59. T. Serrano-Gotarredona, B. Linares-Barranco, F. Galluppi, L. Plana, S. Furber. *ConvNets experiments on SpiNNaker*. 2015 IEEE International Symposium on Circuits and Systems (Lisbon, Portugal, 2015). ISCAS:2405–2408. http://ieeexplore.ieee.org/document/7169169/
60. J.A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, et al. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**, 11:2706–2719 (2013). http://ieeexplore.ieee.org/document/6497055/
61. E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.C. Liu, S. Furber. Scalable energy-effcient, low-latency implementations of trained spiking deep belief networks on SpiNNaker. 2015 International Joint Conference on Neural Networks (Killarney, Ireland, 2015). IJCNN:1–8. http://ieeexplore.ieee.org/document/7280625/
62. P. O'Connor, D. Neil, S.C. Liu, T. Delbruck, M. Pfeiffer. *Front Neurosci* **7**, 178 (2013). https://www.ncbi.nlm.nih.gov/pubmed/24115919
63. F. Jug, J. Lengler, C. Krautz, A. Steger, Swiss Society for Neuroscience (2012)