Training energy-based single-layer Hopfield and oscillatory networks with unsupervised and supervised algorithms for image classification

Madeleine Abernot, Aida Todri-Sanial

Neural Computing and Applications (2023) 35:18505–18518
https://doi.org/10.1007/s00521-023-08672-0

ORIGINAL ARTICLE

Training energy-based single-layer Hopfield and oscillatory networks with unsupervised and supervised algorithms for image classification

Madeleine Abernot[1] · Aida Todri-Sanial[1,2]

Received: 16 September 2022 / Accepted: 10 May 2023 / Published online: 10 June 2023

Abstract

This paper investigates how to solve image classification with Hopfield neural networks (HNNs) and oscillatory neural networks (ONNs). This is a first attempt to apply ONNs for image classification. State-of-the-art image classification networks are multi-layer models trained with supervised gradient back-propagation, which provide high-fidelity results but require high energy consumption and computational resources to be implemented. On the contrary, HNN and ONN networks are single-layer, requiring less computational resources, however, they necessitate some adaptation as they are not directly applicable for image classification. ONN is a novel brain-inspired computing paradigm that performs low-power computation and is attractive for edge artificial intelligence applications, such as image classification. In this paper, we perform image classification with HNN and ONN by exploiting their auto-associative memory (AAM) properties. We evaluate precision of HNN and ONN trained with state-of-the-art unsupervised learning algorithms. Additionally, we adapt the supervised equilibrium propagation (EP) algorithm to single-layer AAM architectures, proposing the AAM-EP. We test and validate HNN and ONN classification on images of handwritten digits using a simplified MNIST set. We find that using unsupervised learning, HNN reaches 65.2%, and ONN 59.1% precision. Moreover, we show that AAM-EP can increase HNN and ONN precision up to 67.04% for HNN and 62.6% for ONN. While intrinsically HNN and ONN are not meant for classification tasks, to the best of our knowledge, these are the best-reported precisions of HNN and ONN performing classification of images of handwritten digits.

Keywords Image classification · Oscillatory Hopfield neural networks · Unsupervised Storkey learning · Equilibrium propagation

1 Introduction

Image classification is a primary computer vision task deployed in many industrial systems, such as healthcare or manufacturing systems. It is usually solved with conventional artificial intelligence (AI) algorithms [1], such as convolutional neural networks (CNNs) trained with gradient back-propagation learning algorithms. However, with the emergence of edge cameras that require real-time secured image processing, like surveillance systems, autonomous cars, or agricultural monitoring systems [2–4], there is a need to process information and bring AI at the edge. CNN models are not compatible with edge constraints in terms of memory, bandwidth, and energy consumption. In comparison, neuromorphic computing systems, with their low power computing and in-memory processing, propose solutions suitable for AI at the edge [5].

Neuromorphic computing, such as spiking neural networks (SNNs) and oscillatory neural networks (ONNS), are brain-inspired paradigms that emulate biological neural network functions for fast learning capability via plasticity and low-power computation for edge devices. SNNs [6–8] are neuromorphic algorithms taking inspiration from spike signals transmitting time-based information in the brain.

✉ Aida Todri-Sanial
aida.todri@lirmm.fr

Madeleine Abernot
madeleine.abernot@lirmm.fr

[1] LIRMM, University of Montpellier, CNRS, Montpellier, France

[2] Electrical Engineering Department, Eindhoven Technical University, Eindhoven, Netherlands

Using spikes reduces the mean voltage amplitude and so the energy consumption of the system. More recently, ONNs appear as an alternative computing paradigm for energy-efficient computation [9]. In hardware, ONNs are implemented as analog-based neural networks using oscillators to emulate oscillatory neurons coupled with analog components, e.g., resistors or capacitors to emulate synaptic coupling [10–17]. Unlike SNNs, ONNs encode information in the phase relationship among oscillators and compute based on the phase dynamics of coupled oscillators. Phase-based computing allows for low voltage amplitude computing, which ultimately reduces the energy consumption [9].

Using a single-layer recurrent architecture, ONNs are shown to solve auto-associative memory (AAM) tasks [15, 17, 18], like Hopfield neural networks (HNNs) [19, 20]. AAM networks can learn patterns and retrieve them from a corrupted input. AAM tasks are mainly solved using unsupervised learning algorithms [21], such as Hebbian [22], Storkey [23], and Pseudo-Inverse [21]. In comparison, image classifications are mainly treated with supervised learning algorithms. State-of-the-art processing for image classifications is typically based on multi-layer neural networks trained with back-propagation algorithms [1, 24, 25].

Multiple benchmarks and datasets exist to evaluate models on image classification. The main ones include MNIST [26], ImageNet [27], and CIFAR-10. MNIST contains grayscale 28 × 28 labeled images of handwritten digits, while ImageNet and CIFAR-10 classify objects using more complex colored images. They are all used to a large extent for assessing AI-model performances. Even though image processing and AAM are two different tasks, authors in [28] adapted HNN to solve a simplified MNIST classification task using the unsupervised Storkey rule. They obtain 61.5% precision using the Storkey learning algorithm, while typically CNNs achieve around 99% on the standard MNIST classification task [1]. More recently, authors in [29] adapted the contrastive Hebbian rule (CHR) [30] to perform supervised learning with energy-based recurrent neural network (RNN) models, using the so-called equilibrium propagation (EP) learning algorithm.

While previous works to solve image classification with AAM networks have been mainly focused on HNNs, in this work, we investigate for the first time how to perform MNIST classification tasks with ONNs. To do so, we first start extending the classification method developed by [28] on HNN to ONNs. Then, we study the classification of simplified black and white 10 × 10 MNIST images with HNN and ONN trained with both unsupervised and supervised learning algorithms. We start by comparing the precision results obtained with Hebbian, Storkey, and pseudo-inverse unsupervised learning algorithms. Then, similar to state-of-the-art methods that apply supervised learning algorithms for classification, we adapt the EP learning algorithm to single-layer AAM networks (AAM-EP). We test and study two approaches using AAM-EP to train HNN and ONN for the simplified MNIST classification task. Our first approach consists of utilizing AAM-EP with random weights at the start. Next, we explore the use of AAM-EP on AAM networks pre-trained with unsupervised learning to fine-tune weight values and assess whether AAM-EP can increase the accuracy of pre-trained AAM networks. We evaluate the precision on simplified MNIST test sets with HNN simulated with MATLAB, and digital ONN design implemented on FPGA [31].

Note, we consider only a simplified 10 × 10 MNIST dataset for image classification with HNN and ONN due to networks and implementation constraints. First, HNN is limited to bipolar values $\{-1,1\}$, and thus, it can consider only black and white images. Also, up to our knowledge, there is no efficient training algorithm for ONN to settle in continuous phase. Usual ONN training algorithms are HNN unsupervised training algorithms constraining training patterns to binary values, and so constraining ONN output to binary phases $\{0°, 180°\}$. EP learning algorithms also considered in this work, should in principle be suitable for continuous ONN outputs. However, we believe it requires further investigations, and it is out of the scope of the paper. Also, our current ONN implementation limits ONN size to a hundred neurons. Thus, with our current ONN implementation, the MNIST dataset is the most suitable for image classification with HNN and ONN. Considering additional image datasets necessitates other ONN implementations and adaptations.

Our contributions can be summarized as: (1) we perform a first study to explore MNIST classification task with ONNs. We adapt the method introduced by [28] for HNNs to solve handwritten digits classification task with ONN. (2) We study precision performance of HNN and ONN using unsupervised learning algorithms. And (3) We apply and adjust the supervised learning Equilibrium Propagation (EP) algorithm for single-layer energy-based AAM, proposing the AAM-EP, on both HNN and ONN to solve the classification of a simplified black and white 10 × 10 MNIST set.

The rest of this article is organized as follows. In Sect. 2, we detail HNN and ONN computing paradigms and describe the AAM computing principle with unsupervised learning algorithms. In Sect. 3, we present methods used to solve the simplified MNIST classification task using AAM networks. In Sect. 4, we explain the various training methods we apply to solve the MNIST classification task using HNN and ONN, including unsupervised and supervised learning. Next, in Sect. 5, we highlight and compare precision obtained with the various training

methods using both HNN and ONN. Finally, we discuss the obtained results and future work in Sect. 6.

2 Auto-associative memory (AAM) neural networks

AAM models can learn patterns and retrieve them from a corrupted input. The most common model used for the AAM task is the HNN, introduced by Hopfield in 1982 [19]. More recently, other analog-based paradigms that show AAM properties have emerged, such as ONN. This part describes the two HNN and ONN models before giving more details on the AAM task, the existing learning algorithms, and how ONN and HNN can solve AAMs.

2.1 Hopfield neural networks (HNNs)

HNNs are single-layer fully connected RNNs, where each neuron is connected to other neurons by bidirectional synaptic weights, see Fig. 1. Synaptic weight values are determined during the training step. During inference, neurons are initialized with input information, they evolve in time following a propagation and an activation function until each neuron reaches stabilization. The final stable state corresponds to the output information. Neurons evolution can be seen as the minimization of an energy function; also, HNNs are labeled as energy-based models. The propagation function of each neuron i follows:

$$h_i = \sum_j W_{ij} * \sigma_j \tag{1}$$

where W_{ij} is the synaptic weight between neuron i and neuron j, and σ_j is the activation value of neuron j.

The new activation of neuron i is then calculated with:

$$\sigma_i = \begin{cases} -1 & \text{if } h_i < 0 \\ \sigma_i & \text{if } h_i = 0 \\ +1 & \text{if } h_i > 0 \end{cases} \tag{2}$$

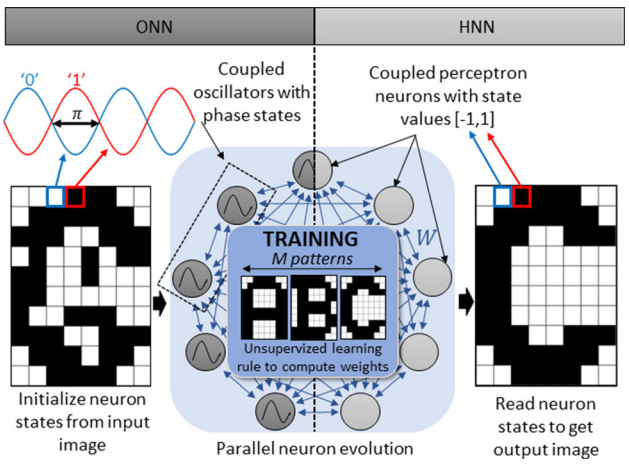

Fig. 1 Auto-associative memory computation with HNN and ONN

2.2 Oscillatory neural networks (ONNs)

ONNs are brain-inspired networks of coupled-oscillators, where each neuron is an oscillator coupled with synapses. ONN implementations are diverse, using not only CMOS-based analog devices [32], but also emerging low-power devices [33, 34], or digital programmable logic [31, 35] to emulate ONN neurons and synapses. In this work, we perform simulations with a fully digital ONN design [31] made of phase-controlled digital oscillators, and 5-bit signed register synapses.

There also exist multiple architectures, and computing methods using ONN. In this work, we only consider fully connected recurrent architecture with phase computation, see Fig. 1, as it can compute AAM tasks [18]. We encode data in phase relationships between oscillators. For example, for bipolar values, considering a reference oscillator with phase 0° ([+ 1]), another oscillator with 180° phase shift from the reference oscillator would represent the opposite bipolar value ([− 1]). Similarly, an oscillator with 0° phase shift represents the same bipolar reference value ([+ 1]). Phase computation allows to reduce signal amplitude and consequently the energy consumption of the system [9].

ONN computes based on the dynamics of the coupled oscillators. Couplings between oscillators are set during the learning step and depending on the task. Next, inference starts with oscillators' phase initialization depending on the input information. Then, ONN phases evolve in time due to the natural dynamic of the oscillators and stabilize to a final phase value. The final phase value is then measured to obtain the ONN output information. This computing paradigm can be associated with the minimization of an energy function (as in HNNs); thus, ONN is also an energy-based computing model. Such property makes ONN suitable to solve AAM tasks.

2.3 Auto-associative memory (AAM) task and learning

AAM tasks involve memorization of patterns and retrieval of the memorized patterns from the corrupted input information. AAM processes in two main steps. The first is the learning step to memorize patterns in the network, and the second is the inference step to retrieve one of the memorized patterns from a corrupted input, see Fig. 1. Considering HNN or ONN, we associate each image pixel to a neuron, and the color of the pixels to the neuron value,

either the binary value for HNN, or the phase value for ONN. AAM tasks are usually solved using unsupervised learning algorithms. Unsupervised learning algorithms differentiate from supervised learning algorithms as they only use data to compute, without any external interaction to evaluate it. ONN-oriented rules were introduced recently, especially for ONNs [36]. However, other unsupervised learning rules designed for HNN are also compatible with ONN. In order to make a meaningful comparison between HNN and ONN precision, we only consider learning algorithms compatible with both models. Unsupervised learning algorithms are categorized into biologically plausible algorithms following three main criteria. First, the locality which induces that the weights update only depends on the activation from neurons in both sides of the synapse. Second, the incrementality that indicates if a network can learn patterns one by one or if it needs to learn all patterns together. Third the weight symmetry that states that as in human brain, weights are not symmetric. Yet, for both HNN and ONN, the synapse between neuron i and j and the synapse between neuron j QUOTE and i are the same, so weights are symmetric. Even though this makes HNN and ONN non-biologically plausible paradigms, weight symmetry is necessary for HNN and ONN learning. The first learning algorithm we use is the Hebbian learning rule [22]. Hebbian is local and incremental and has symmetric weights. The principle is based on the biological rule saying "neurons that fire together wire together." Synaptic connection W_{ij} between neuron$i$ and j with the same value is calculated for k patterns ζ^k as:

$$W_{ij} = \sum_k \frac{1}{N} \xi_i^k \xi_j^k \tag{3}$$

Hebbian has good AAM properties, and however, its capacity is limited [21], meaning it can not memorize and retrieve patterns correctly for a numerous number of patterns. Additional learning rules with higher storage capacity also exist, such as the Storkey learning rule [23] and the pseudo-inverse learning rule [21]. Storkey is also local, incremental, with symmetric weights and is similar to Hebbian as:

$$W_{ij} = \sum_k \frac{1}{N} \left(\xi_i^k \xi_j^k - \frac{1}{N} \xi_i h_{ij} - \frac{1}{N} h_{ij} \xi_j \right) \tag{4}$$

with h_{ij}:

$$h_{ij} = \sum_j W_{ij} \xi_j \tag{5}$$

In the contrary, the pseudo-inverse learning rule is neither local, nor incremental, but has symmetric weights. Thus, it is not biologically plausible. The pseudo-inverse learning rule is as:

$$W = N * \xi * pinv(\xi) \tag{6}$$

The biological plausibility is useful for online learning applications. Though, in this paper, we process weights offline using MATLAB software, before using them in the HNN MATLAB model or in the digital ONN. In the digital ONN design, we normalize weights in 5-bits signed format. We compare the three above-mentioned learning algorithms with HNN and ONN on the simplified MNIST set.

3 MNIST classification with AAM networks

It is important to state that AAM and image classification are two distinctive tasks. Thus, one needs to adapt AAMs to perform classification. In [28], authors propose a solution to apply HNN to classify images of MNIST handwritten digits when trained with Storkey learning rule. In this work, inspired by [28], we replicate their method to evaluate and compare precision of HNN and ONN on a simplified MNIST set for different unsupervised learning configurations. In this section, we first present the MNIST set and the simplified version we use in this work before describing methods to classify the simplified MNIST set using HNN and ONN.

3.1 MNIST database

The MNIST set was created to assess neural networks performances on image classification task. It contains 28 × 28 Gy scale labeled images of handwritten digits, from 0 to 9. It is organized in two sets, a training set with 60,000 images, and a test set with 10,000 images. State-of-the-art solves the MNIST classification problem with CNN models. They can achieve more than 99% of accuracy when trained by supervised back-propagation algorithms [1].

3.2 MNIST dataset preparation

In this work, we employ a simplified MNIST set containing the same number of training and test images, where each image is pre-processed to be transformed in a 10 × 10 black and white image in order to be compatible with our digital ONN design, see Fig. 2. We focus on a 10 × 10 format because the digital ONN design is limited in size and resources. The transformation of each image follows three steps. First, each 28 × 28 image is cropped to 20 × 20 removing the black background to reduce similarities among images. Then, the 20 × 20 image is resized to a 10 × 10 by taking average values of 2 × 2 neighbor

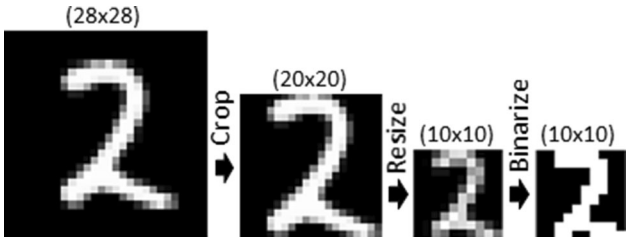

Fig. 2 Simplified MNIST classification image pre-processing method

pixels. Finally, we binarize each image into black and white using a threshold. MNIST grayscale pixels take values in the range of [0,255], thus, for example, if we use a threshold of 128, a pixel value under 128 will become black, otherwise white. We study different binarization thresholds and their influence on the simplified MNIST classification task in Sect. 3.4.

3.3 Image classification with AAM

There are some clear distinctions between AAM and classification tasks. On one hand, classification problems associate input information to output classes; hence, input and output can have different dimensions. On the other hand, AAM tasks associate a corrupted input to a clean memorized output, where both have same dimensions. To solve MNIST classification problem using AAM, authors in [28] propose to train an HNN network with one pattern per label, so one image per digit. Thus, HNN and ONN inference will stabilize to one of the training patterns corresponding to a digit label class, equivalent to the MNIST classification task, see Fig. 3.

3.4 Training patterns for MNIST classification

In this work, we use HNN and ONN configured for pattern recognition task, while MNIST is a classification task. Thus, we need to adapt the MNIST classification dataset to be solved as a pattern recognition task. MNIST classification with AAM starts by the training step to configure the weights, and so the choice of the training patterns. The training pattern choice is a key for high precision. Each training pattern must be the best representation of its digit, such that each image of that digit from MNIST set will stabilize to that training pattern. Authors in [28] propose to perform a mean on each grayscale image with same label from the MNIST training set of 60,000 images, and we reuse the same method. We define 10 training patterns corresponding to the 10 digits which will be learnt as stable points for both HNN and ONN networks. The 10 training patterns are created from the 60,000 training images. We group the training images by digits and compute a mean image for each digit such that we obtain ten 28×28 Gy scale images representing digits between 0 and 9. After, we apply the pre-processing on each training pattern to obtain ten 10×10 black and white images, with one image per digit being the training pattern associated with the corresponding digit, see Fig. 4.

Pre-processing binarizes each training pattern to be converted into black and white. Binarization threshold determines the number of black or white pixels in each image. In AAM tasks, having uncorrelated training patterns is also a key for high precision. The more patterns are correlated, the hardest it is to dissociate them. The correlation of patterns is evaluated by the Hamming Distance (HD) metric d, which calculates the number of different

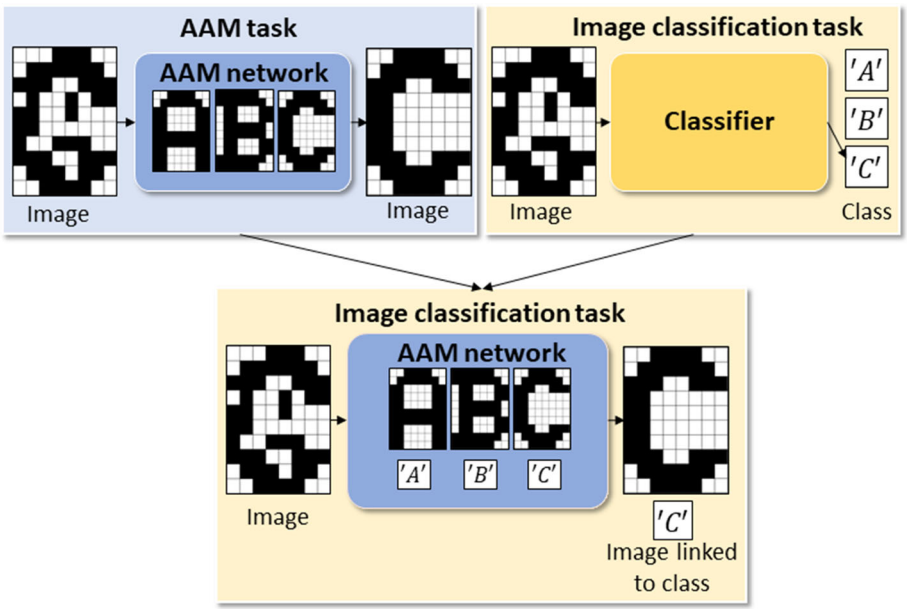

Fig. 3 Difference between AAM and image classification tasks, and the adaptation of AAM network for image classification task

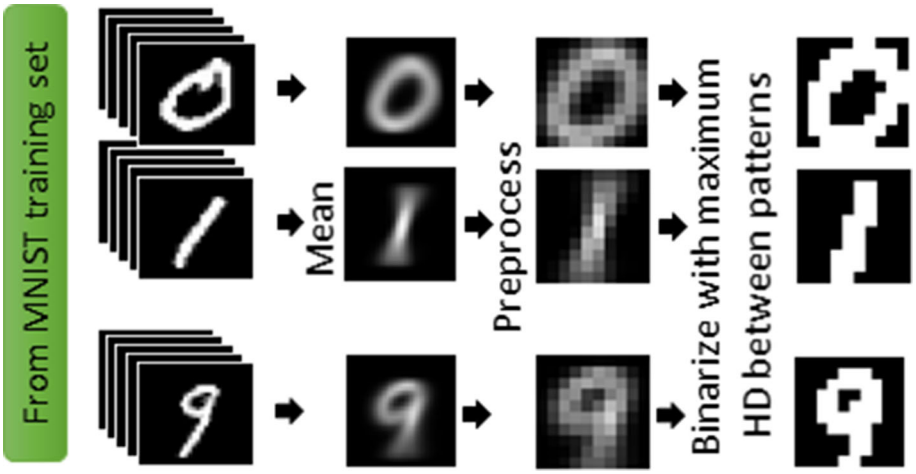

Fig. 4 Process to define training patterns from MNIST training set

neuron activation between two patterns. Grayscale in MNIST images is encoded between 0 and 1. Thus, as in [28], we study the ideal threshold to have the largest HD between training patterns. Figure 5a shows the average HD d_{avg}, and the minimum HD d_{min} in-between the 10 training patterns depending on the binarization threshold θ. It highlights that HD is maximal for $\theta = 0.3$, meaning training patterns are less correlated. Figure 5b depicts the HD between the 10 patterns generated after pre-processing with $\theta = 0.3\theta = 0.3$, and Fig. 5c prints the resulted 10 training patterns.

4 AAM training for MNIST classification

Once we have training patterns, we need to define synaptic weights using some learning algorithms. Usual image classification tasks are trained based on supervised learning algorithms. However, AAM tasks use unsupervised learning algorithms. In Sect. 4.1, we describe how we use various unsupervised learning algorithms to train HNN and ONN for classification. Then, Sect. 4.2 proposes to adapt the supervised EP algorithm to train HNN and ONN, with the AAM-EP. With supervised AAM-EP, we study first training networks from random weights initialization, and then starting from a pre-trained network using weights generated by unsupervised learning algorithms, see Fig. 6.

4.1 Unsupervised learning

Authors in [28] use the Storkey unsupervised algorithm after stating Hebbian does not have enough memory capacity. Here, we study three training algorithm Hebbian, Storkey, and Pseudo-inverse on HNN. Note, incremental learning algorithms can be sensitive to iterative learning. We call iterative learning the possibility to learn same patterns multiple times. Thus, we study the impact of iterative learning on the Hebbian and Storkey learning algorithms, as Pseudo-inverse is not incremental.

From HNN MATLAB simulations, we extract the weights giving best HNN precision and integrate them inside the digital ONN design to compare ONN precision with HNN precision on the simplified MNIST classification task. Chosen weights are normalized into a 5-bits signed representation to be compatible with digital ONN design.

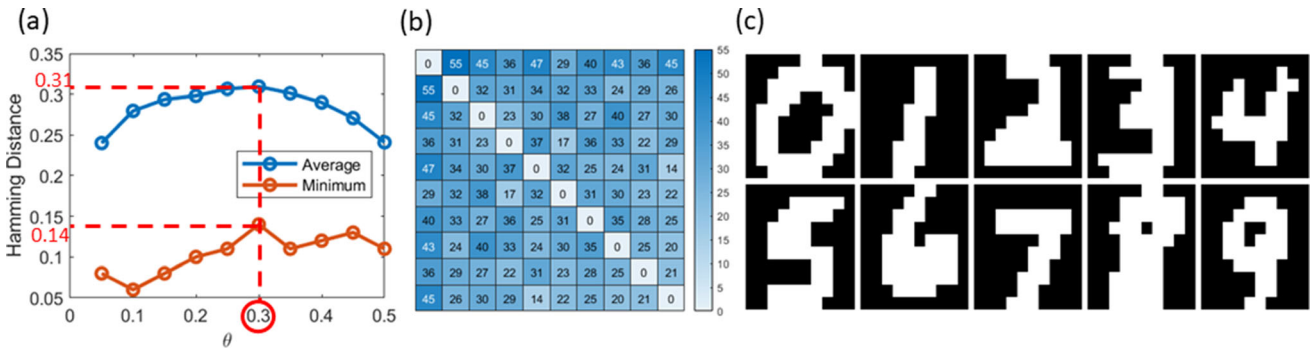

Fig. 5 **a** Average and mean hamming distance (HD) between training patterns for various binarization threshold β, **b** HD between training patterns binarized with $\beta = 0.3$, and **c** training patterns with $\beta = 0.3$

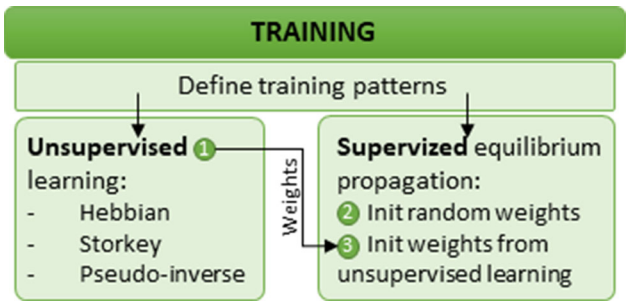

Fig. 6 Study of various options to perform MNIST classification with HNN and ONN

We simulate the digital ONN design using the Vivado design tool with xc7z020-1clg400c FPGA target.

We evaluate precision from inference results on the MNIST test set of 10,000 images. Inference starts by initializing neurons with one of the test set images. Then, the network evolves until stabilization. The stable state is then compared to the 10 training patterns to define the output class. Exact match between ONN output and corresponding training pattern is considered a correct classification. MNIST results are evaluated through four metrics for HNN and ONN. First, we evaluate the precision representing the percentage of tested images which stabilize to the correct training pattern. Then, we compute the *true negative* metric, that counts the number of images which stabilize to one of the training patterns but not the expected one. We also add the percentage of *spurious* outputs incorporating images that stabilize to none of the training patterns, but to another non-memorized image. Finally, for the ONN, an additional metric is necessary to highlight the percentage of images that never stabilize to an output. We call them *inconsistent* images. In Sect. 5.1, we report results obtained on the 10,000 images from the simplified MNIST test set with HNN and ONN trained with unsupervised learning.

4.2 Supervised equilibrium propagation

EP is a supervised learning algorithm proposed in [29] for energy-based RNN models. In [29], authors demonstrate EP efficiency to solve MNIST classification task using multi-layer energy-based continuous RNNs. HNNs and ONNs are also energy-based RNNs, yet made with a single-layer architecture of non-continuous neurons. Thus, in this paper, we propose to adapt the EP algorithm to energy-based single-layer AAM networks. First, Sect. 4.2.1 describes the EP algorithm from [29]. Then, Sect. 4.2.2 explains how we adapted EP to AAM single-layer networks creating an AAM-EP learning.

4.2.1 Equilibrium propagation (EP)

EP algorithm was introduced by [29] to perform supervised learning on energy-based RNN models. It is inspired by the Contrastive Hebbian Rule (CHR) algorithm [30] proposed to solve supervised problems with multi-layer continuous Hopfield networks, a specific type of energy-based RNN. The common algorithm to solve supervised problems with multi-layer RNN models is the Back-Propagation Through Time (BPTT) [37]. Even though it is efficient and gives really high precision, it requires a large amount of computational resources. Thus, authors in [29] proposed a new solution which requires less computation to solve supervised problems with energy-based RNN models. EP computes the gradient of an objective function, similar to the Hopfield energy function, that propagates in the layers. This gradient back-propagation is transparent in the weight update algorithm, and the final weight update equation is intuitive and simple to apply. Note, authors in [38] showed that EP and BPTT have similar gradient updates in an RNN, so achieving similar precision. The EP algorithm defines two learning phases to update the weights. The first phase, called the free phase, clamps the input information to the input layer and waits until neurons of the following layers stabilize. Then, EP performs a second phase, called the weakly clamped phase, where input neurons and output neurons are clamped with the input information associated with the expected output information. In [29], they show that the signal back-propagated during the weakly clamped phase corresponds to the derivative error of their objective function, and they define the following weight update algorithm:

1. Clamp input information at the input neurons and let the network evolve until all neurons from hidden and output layers stabilize.
2. Save the stable state σ_i^0 of each neuron i.
3. Weakly clamp with β the expected output information at the output neurons and let the network evolve until all neurons from hidden layers stabilize.
4. Save the new stable state σ_i^β of each neuron i.
5. Update weight w_{ij} between neuron i and neuron j as:

$$W_{ij} = W_{ij} + \Delta W_{ij} \tag{7}$$

with

$$\Delta W_{ij} = \frac{1}{\beta}(\sigma_i^\beta \sigma_j^\beta - \sigma_i^0 \sigma_j^0) \tag{8}$$

Using $\beta = 1$, in [29], authors show that RNNs with 1, 2, or 3 hidden layers can solve the MNIST classification problem and reach more than 95% of precision. Eventually, EP is a low-computation, energy-based learning algorithm

expected to fit with analog computing paradigms. Thus, it makes it attractive for ONN application. However, EP cannot be applied directly to HNN or ONN as they are single-layer energy-based models.

4.2.2 EP for single-layer energy-based AAM

In this paper, we consider single-layer energy-based AAM models with HNN and ONN and we adapt EP to HNN and ONN. Note, recently, authors in [39] also proposed to use EP for pattern recognition with ONN. However, we apply it with different methods. In [39], authors apply supplementary neurons to clamp training patterns during clamping phase, while in this work, we do not require additional neurons. Also, in [39], authors perform phase dynamics simulations of memristor-based ONNs to validate their method, while here we perform simulations of a digital ONN design. This Section explains how we adapt EP to fit AAM, introducing AAM-EP and we present methods and parameters used in this work when training HNN and ONN with AAM-EP.

EP is a supervised algorithm, meaning it needs labeled data and interaction to define synaptic weights. As we only have a single-layer network, the MNIST set needs to be modified to associate each input image to the correct output image with same dimension. We consider the previous training patterns defined for unsupervised learning as the corresponding digit labels. Note, each input/output pair of images are converted by pre-processing to 10×10 black and white images. So, first we define the expected outputs and replace each label from MNIST training and test sets with the corresponding training pattern. Then, AAM-EP uses these new input/output pairs from MNIST training set to clamp input and output during the two training phases. After, during inference and precision evaluation, we compare the output given by the network with the expected training pattern using the MNIST test set.

EP learning is a two-phase algorithm, with a first free phase with input neurons clamped, and a second weakly clamped phase with additional weakly clamped output with factor. The clamping principle works well for a multi-layer network with at least one hidden layer [29]. However, in HNN or ONN cases, input and output neurons are the same, and there is no hidden layers. Thus, we adapt the EP algorithm for AAM as the AAM-EP following:

1. Clamp the input image in the network; Consider σ_i^0 as activation of neuron i from the input image.
2. Clamp the expected output image with $\beta = 1$ in the network; Consider σ_i^β as activation of neuron i from the expected output image.
3. Use the two activation states to update weight between neuron i and j as $W_{ij} = W_{ij} + \alpha \Delta W_{ij}$ with:

$$\Delta W_{ij} = \sigma_i^\beta \sigma_j^\beta - \sigma_i^0 \sigma_j^0 \tag{9}$$

Note, we remove the factor $1/\beta$ because we use $\beta = 1$. Also note, we add a learning rate factor α in order to regulate the weight update for each training iteration (each image). During tests, we consider various range of learning rates between 0.0001 and 1. Initialization of the weights is also important to achieve high precision. In this work, we first initialize weights randomly, with small values between $[-1; 1]$. Especially, we study if AAM-EP can train a single-layer energy-based RNN like HNN or ONN from scratch. After, we initialize AAM networks using weights computed previously with unsupervised learning. With this option, we study if AAM-EP can improve precision of an already trained network. At first, we apply AAM-EP for numerous epochs and observe the HNN precision at each epoch for various learning rates. Each epoch applies a random minibatch of 1000 pre-processed images from the MNIST training set to update the weights. Precision can slightly change from one trial to another as minibatch images are randomly chosen from the full simplified 10×10 MNIST training set. The precision for each epoch is computed on the full simplified 10×10 MNIST test set. In a second step, we select the best configuration and collect corresponding weights to evaluate the AAM-EP learning algorithm with the digital ONN design. Collected weights are normalized into a 5-bits signed representation compatible with the digital ONN design. We simulate the digital ONN design using the Vivado design tool with xc7z020-1clg400c FPGA target. We compare precision, *true negative*, *spurious*, and *inconsistent* results obtained with HNN and ONN for the same weight values.

5 Results

This section presents results obtained with both HNN and ONN trained with unsupervised and supervised learning algorithms to solve the simplified 10×10 MNIST classification task.

5.1 Unsupervised learning

Figure 7a highlights the precision obtained using HNN for multiple training configurations. Configurations include the learning rules, Hebbian, Storkey, or Pseudo-inverse for up to 10 iterations. As expected, Pseudo-inverse is not sensitive to iterative learning as precision does not change depending on the iterations, and Storkey is sensitive. However, we expected Hebbian to be sensitive to iterative learning, but for each iteration precision stays 0%. Figure 7a also shows that the best precision is obtained when

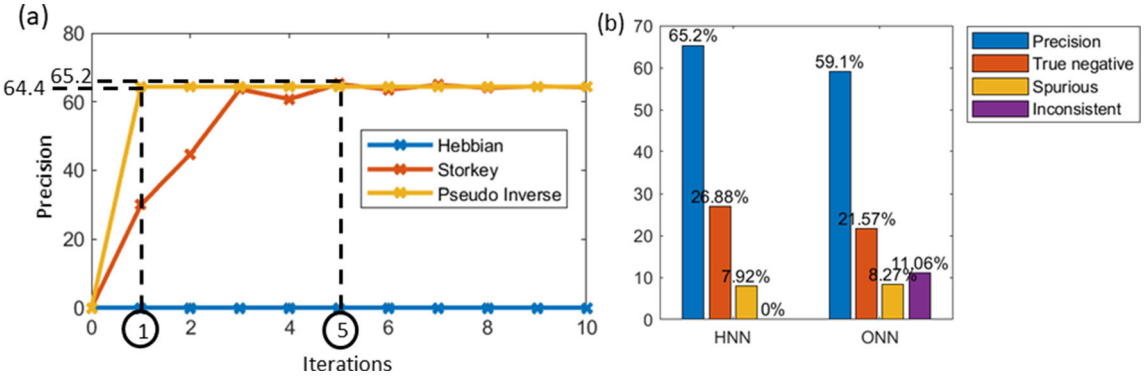

Fig. 7 **a** HNN precision for multiple unsupervised training configurations. Training configuration includes the choice of the learning rule, and the number of iterations performed on the training patterns (each iteration learns the 10 training patterns). **b** Results of ONN and HNN trained with 5 iterations of the unsupervised Storkey learning rule

training the network with the 10 training patterns using the iterative Storkey learning rule during 5 iterations. However, training HNN with Pseudo-inverse can reach a precision close to the best with 64.4% after a single iteration. We configure the digital ONN design using the synaptic weight values obtained with best configuration, 5 Storkey iterations, to compare ONN precision with HNN precision on the simplified MNIST classification task.

Figure 7b shows results of HNN and ONN trained with 5 Storkey iterations. HNN and ONN have similar trends. First, HNN precision and true negative percentages are higher than ONN. The difference is certainly in part due to the normalization of weights into 5-bits signed integers in the digital ONN design. Also, the number of spurious patterns detected with ONN is slightly higher than with HNN, and inconsistent patterns often happen with ONN while never with HNN. Thus, we strongly believe that HNN decides more easily of a stable output, even if it is a true negative, while ONN can hesitate between different outputs and keep bouncing between patterns. Figure 7b also reports the best precision, to the best of our knowledge, obtained with HNN and ONN trained with unsupervised learning algorithms to solve a simplified MNIST classification task. We report on 65.2% precision with HNN, and 59.1% precision with ONN. However, reported precision is lower than state-of-the-art precision of neural network models solving MNIST classification problems with supervised learning, reaching around 99%. Hence, next section presents results obtained using AAM-EP supervised algorithm to train HNN and ONN on our simplified MNIST set to investigate if supervised learning can increase HNN and ONN precision.

5.2 Supervised EP

In this Section, we present results obtained with both HNN and ONN trained with the supervised AAM-EP algorithm to solve the MNIST classification problem.

To begin with, Fig. 8a displays HNN precision evolution for multiple learning rates during 50 epochs, with weights initialized with small random positive values. It shows that for all the tested learning rates, during several epochs, the precision stays 0%. Moreover, Fig. 8b compares results between HNN and ONN, obtained using weights achieved with learning rate $\alpha = 0.0005$ after 10 epochs. Note, we tried various learning rates and epochs and obtained same results. It highlights that for HNN, each image stabilizes to a spurious pattern, while for ONN, neuron states continuously evolve without reaching stabilization. To sum up, using AAM-EP learning from scratch with HNN or ONN, does not result in high precision on the simplified MNIST classification task.

Subsequently, we reproduce the precision tests with weights initialized using unsupervised learning. More precisely, we initialize weights with best-unsupervised learning precision obtained for each unsupervised learning algorithm. That are weights generated with Hebbian after one iteration, with Storkey after 5 iterations, and with Pseudo-inverse after one iteration. Figure 9a shows precision of the HNN trained with AAM-EP during 50 epochs for various learning rates when weights are initialized with Hebbian. It highlights that for weights initialized with Hebbian, the AAM-EP does not modify the network to allow classification of the simplified MNIST task.

Figure 9b, c shows precision of the HNN trained with AAM-EP during 50 epochs for various learning rates when weights are initialized with 5 iterations of Storkey and Pseudo-inverse, respectively. It illustrates a particular behavior in which precision increases during the first couple of epochs and decreases afterward. The larger the

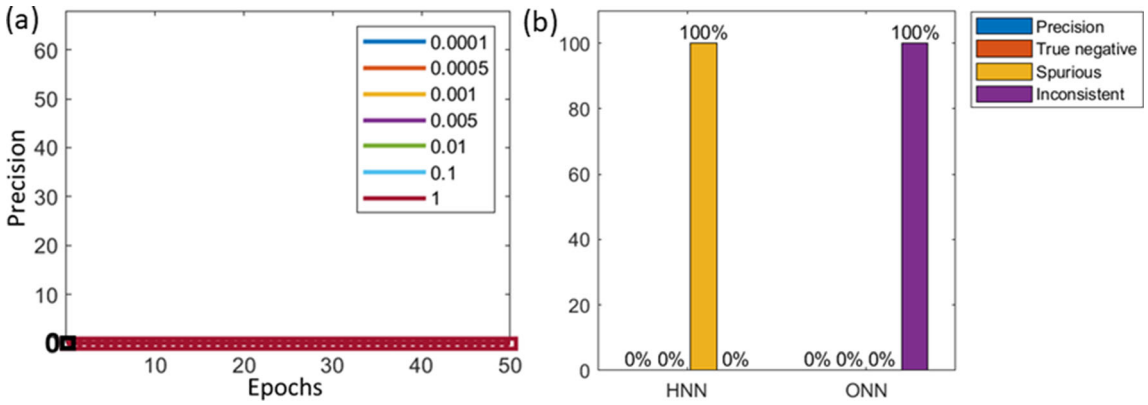

Fig. 8 **a** Results of simplified MNIST classification precisions obtained with HNN, for various learning rates, during numerous epochs, starting with weights initialized randomly. **b** Results of ONN and HNN trained with AAM-EP algorithm with random weight initialization after 10 epochs

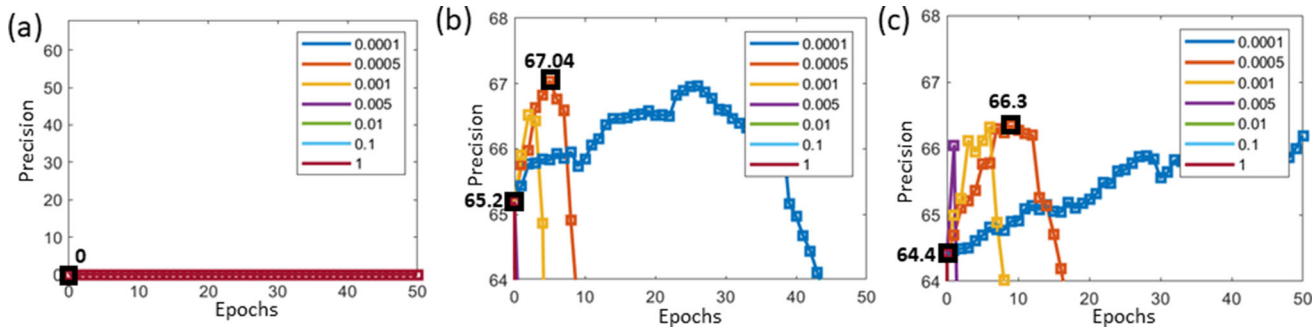

Fig. 9 Simplified MNIST classification precisions obtained with HNN, for various learning rates, during numerous epochs, starting with weights initialized with **a** 1 iteration of Hebbian, **b** 5 iterations of Storkey, and **c** 1 iteration of Pseudo-inverse

learning rate is, the faster the increase and decrease phenomenon is observed. Considering Pseudo-inverse initialization, the maximum precision is obtained with learning rate $\alpha = 0.0005$, after 9 epochs, for which HNN reaches 66.3% precision. Considering Storkey initialization, the maximum precision is obtained with learning rate $\alpha = 0.0005$, after 5 epochs, and HNN reaches 67.04% precision. For both Pseudo-inverse and Storkey initialization, AAM-EP increases the HNN precision of around to 2%. We assume this phenomenon is due to the weight's initial values. If the unsupervised learning already set weights to an acceptable network configuration, then the AAM-EP algorithm can slightly help to increase precision up to a certain point after which it modifies the previous configuration and reduces drastically the HNN precision. However, if the weights initialization does not bring the network to an acceptable configuration, such as with Hebbian or random weights initialization, the AAM-EP cannot modify enough the network to reach a good configuration.

We configure the digital ONN with best HNN configuration, that is weights obtained after 5 epochs of AAM-EP with learning rate $\alpha = 0.0005$ after initializing weights with Storkey learning rule. Figure 10 plots the precision obtained with both HNN and ONN. Note that obtained precision from Figs. 9b and 10 are different. As we use mini-batch of 1000 randomly chosen images at each epoch, from one run to another, computed weights and obtained precision can be slightly different. Figure 10 shows that for both HNN and ONN, precision increases with the use of

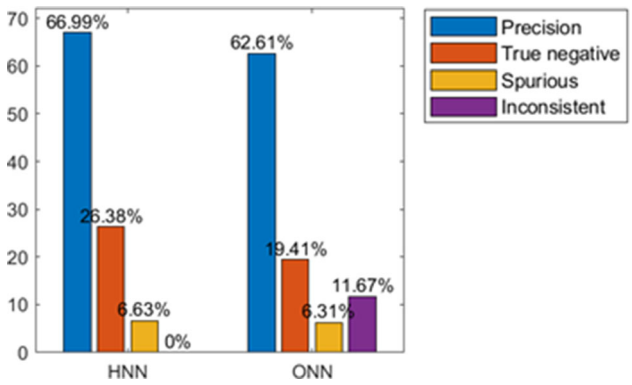

Fig. 10 Results of ONN and HNN trained with AAM-EP algorithm with $\alpha = 0.0005$ after 5 epochs with weights initialized after 5 iterations of Storkey

the AAM-EP supervised algorithm. Additionally, the number of true negative stays approximately stable for HNN and decreases for ONN, and the number of spurious patterns decreases. Thus, we deduce the AAM-EP algorithm helps to differentiate and reinforce training patterns such that input patterns are better associated with training patterns. For example, spurious patterns often appear to be close to training patterns with some wrong pixels. We believe AAM-EP helps to modify local weights associated with wrong pixels such that HNN and ONN stabilize to correct training patterns.

HNN and ONN precisions increase of about 2%, and 3.5%, respectively. Table 1 assembles best HNN and ONN precisions of the three learning configurations, unsupervised learning only, supervised learning only, both unsupervised and supervised learning.

5.3 Comparison of both unsupervised and supervised methods

In this work, we highlighted that the supervised AAM-EP learning algorithm can help increasing precision of HNN and ONN networks pre-trained with unsupervised learning algorithms for the MNIST classification task. However, using supervised learning can be more demanding in terms of computational efforts.

In this work, training was performed in MATLAB. However, we can derive from results obtained in MATLAB the computational efforts for the different learning algorithms, as well as an estimation of the learning latency if learning was implemented on the digital ONN. We evaluate the computational efforts using the metric of the number of multiply and accumulate operations ($NMAC_{OP}$) required for each learning method. Table 2 shows a comparison of the computational efforts for the various learning methods for a general case, as well as for the MNIST classification application. It shows that the supervised learning algorithm increases drastically the $NMAC_{OP}$ per training compared to the unsupervised learning algorithms. Also, considering a system frequency of $F_{sys} = 31.25$ MHz, and parallelism in the $NMAC_{OP}$, Hebbian learning can compute in 1 clock cycle, Storkey, in 15 clock cycles, and Pseudo-inverse in 3 clock cycles, while AAM-EP requires approximately 50 ms to train HNN or ONN for MNIST classification task. Thus, supervised AAM-EP takes longer to compute that other unsupervised learning algorithms. However, depending on the application, using AAM-EP to increase precision while also increasing the computational efforts and latency can be interesting.

6 Discussion

This work analyzes classification of handwritten digits from MNIST set using two AAM networks, HNN and ONN, trained with either unsupervised learning, supervised learning, or both learning strategies. In this section, we first discuss the results obtained with our AAM-EP learning algorithm in comparison with reported results in literature. Then, we highlight advantages and limitations of using AAM networks to solve MNIST classification task. Finally, we expose future explorations.

6.1 Comparison with other models

Results presented in Sect. 5 showed we can train HNN and ONN networks with supervised AAM-EP algorithm. Additionally, the AAM-EP algorithm increases HNN and ONN precision on the simplified MNIST classification task from precision obtained if learning with unsupervised algorithms. We report an HNN maximum precision of 67%, and an ONN maximum precision of 62.5%. In comparison, [28] reported 61.5% HNN precision, while performing classification of a simplified MNIST set with 14×14 black and white images, with additional pattern optimization. So, on one hand our AAM-EP solution has, to the best of our knowledge, the highest reported precision of single-layer AAM networks performing classification of handwritten digits from MNIST set. And on the other hand, state-of-the-art multi-layer RNNs trained with EP can solve the complete MNIST classification problem with more than 90% precision, see Table 3. Thus, our proposed AAM-EP learning algorithm improves already known precision of AAM on handwritten digits classification task, but does not overtake multi-layer RNN model precision on the same task.

Table 1 Best precision results obtained with both HNN MATLAB emulator and ONN digital design for the three training configurations (unsupervised, supervised, unsupervised and supervised)

	Unsupervised Storkey rule 5 iterations	Supervised EP All α & epochs	EP with Storkey $\alpha = 0.0005$, epochs = 5
HNN	65.2%	0%	67.04%
ONN	59.1%	0%	62.61%

Table 2 Computational efforts required for training depending on the learning algorithm for a network of N neurons, for k training patterns, after it iterations or ε epochs

Learning rule	$NMAC_{OP}$	$NMAC_{OP}$ MNIST	Latency MNIST ($F_{sys} = 31.25MHz$)
Hebbian	$(kN^2)it$	100 k	32 ns
Storkey	$(kN^2 + 2kN)it$	510 k	480 ns
Pseudo-inverse	$((kN^2) + 2(N^2k))it$	300 k	96 ns
AAM-EP	$\#_{img}\varepsilon(\#_{cycles}N^2 + 2N^2)$	5000 k	50 ms

$\#_{cycles}$ represents the number of cycles necessary for the ONN inference, we consider 3 cycles in average

Table 3 Comparison of various network models trained with EP on the MNIST classification task

Network	HNN ONN		RNN [29] (784-H-10) $H = 500$		BNN [40]–Conv. Output (O)		Analog RNN [41] (784-H-10)	SNN [42] (784-H-10)
Arch	100-FC	1H	2H	3H	$O = 10$	$O = 700$	$H = 100$	$H = 500$
Param	10 k	397 k	647 k	897 k	33.76 k	> 2 M	79.4 k	397 k
Epochs	< 10	15–20	40	140	30	10–20	10	5–10
Precision	67%	97%	98%	97%	11%	97–98%	96.5%	97%

However, multi-layer models are often heavy, requiring a lot of resources latency to be trained. Thus, we compare our solution with state-of-the-art models in terms of resources and latency by comparing the number of parameters to tune the network, as well as the number of epochs necessary to obtain the best precision. Table 3 highlights the precision, number of parameters, number of epochs, and architectures of various models trained with EP for MNIST classification. HNN and ONN are single-layer models requiring low number of parameters compared with multi-layer models. In terms of training latency, ONN and HNN reach their maximum precision after less than 10 epochs, while most of the multi-layer networks necessitate more epochs to converge to their maximum precision, between 10 and 140 depending on the model. Thus, ONN can be of interest for applications with restricted amount of resources, but flexibility in the precision.

6.2 Future work

Handwritten digits classification is a benchmark application for neural network models. Here, we explored solutions and learning algorithms to solve handwritten digits classification with single-layer energy-based AAM models. We explore HNN and ONN, as they aim to be suitable for image processing at the edge. However, in this work, we have seen that using supervised and/or unsupervised learning does not perform better than other multi-layer models. A possible improvement is to explore multi-layer associative memory architecture [43] like Heterogeneous Associative Memory (HAM), in order to have a more coherent architecture with classification task [44]. Initializing input layer with input images classified into different categories highlighted by the output layer. HAM networks using perceptron neurons such as the Linear Associative Memory [45], or the Bidirectional Associative Memory [46, 47] could replace the single-layer HNN. And in [44] and [48], authors demonstrated that ONN can work as HAM for edge detection application, so it could also be applied for handwritten digits classification task. The EP algorithm was first introduced as a supervised learning algorithm for energy-based multi-layer RNNs, and thus, we expect it to be compatible with multi-layer HNN and ONN. Also, there are numerous learning algorithms that are compatible with multi-layer networks, which could help increasing HNN and ONN precision [49].

Finally, an extension of this work is to explore the integration of AAM-EP on ONN hardware. In [29], authors precise that the algorithm is compatible with hardware implementation. One could also integrate learning on chip and perform the two learning phases of AAM-EP directly inside the digital ONN design, or even inside an analog ONN design.

7 Conclusion

This work proposes a study of solving classification of handwritten digits images from MNIST dataset using single-layer energy-based AAM networks. In particular, we analyze the behavior of an HNN tested on MATLAB software and a digital ONN design simulated on Vivado software. To classify with single-layer AAM network, we define training patterns for each label. So, after initialization, the network evolves to one of the training patterns

corresponding to one label. We evaluate network precision to classify 10 × 10 black and white images of handwritten digits from a simplified MNIST set. We compare and assess precision performances for different learning configurations. First, we compute precision for different unsupervised learning algorithms. We obtain a maximum HNN precision of 65.2%, and a maximum ONN precision of 59.1% when training with 5 iterations of the Storkey algorithm. Then, we evaluate precision using supervised EP algorithm adapted to AAM, the AAM-EP. We highlight that tuning weights of a pre-trained network with AAM-EP, the precision is augmented. For example, using a pre-training configuration with 5 Storkey iterations, precision augments by 2% and 3.5% for HNN and ONN, reaching 67%, and 62.6%, respectively. Thus, our AAM-EP algorithm increases single-layer AAM model precision on the simplified MNIST classification task. Despite the lower obtained precision in comparison with state-of-the-art MNIST classification with multi-layer networks trained with EP, this study explains how to solve supervised problems using AAM networks. More than that, it introduces the use of AAM-EP supervised learning algorithm to help single-layer AAM network to classify images.

Funding This work is supported by the European Union's Horizon 2020 research and innovation program, EUH2020 NEURONN (www.neuronn.eu) project under Grant No. 871501.

Data availability The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324. https://doi.org/10.1109/5.726791
2. Baran R, Rusc T, Fornalski P (2016) A smart camera for the surveillance of vehicles in intelligent transportation systems. Multimed Tools Appl 75(17):10471–10493. https://doi.org/10.1007/s11042-015-3151-y
3. Viswanathan V, Hussein V (2017) Applications of image processing and real-time embedded systems in autonomous cars: a short review. https://www.semanticscholar.org/paper/Applications-of-Image-Processing-and-Real-Time-in-A-Viswanathan-Hussein/5f7663469cdd84857a7a7a1392ddd54228abe39c (consulté le 8 avril 2022)
4. Ahansal Y, Bouziani M, Yaagoubi R, Sebari I, Sebari K, Kenny L (2022) Towards smart irrigation: a literature review on the use of geospatial technologies and machine learning in the management of water resources in arboriculture. Agronomy 12(2), Article no. 2, févr. https://doi.org/10.3390/agronomy12020297
5. Christensen DV et al (2022) roadmap on neuromorphic computing and engineering. Neuromorphic Comput Eng. https://doi.org/10.1088/2634-4386/ac4a83
6. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7
7. Lecerf G et al (2014) Silicon neuron dedicated to memristive spiking neural networks. In: 2014 IEEE international symposium on circuits and systems (ISCAS), June 2014, pp 1568–1571. https://doi.org/10.1109/ISCAS.2014.6865448
8. Tavanaei A, Ghodrati M, Kheradpisheh SR, Masquelier T, Maida A (2019) Deep learning in spiking neural networks. Neural Netw Off J Int Neural Netw Soc 111:47–63. https://doi.org/10.1016/j.neunet.2018.12.002
9. Delacour C, Carapezzi S, Abernot M, Todri-Sanial A (2022) Energy-performance assessment of oscillatory neural networks based on VO2 devices for future edge AI computing. https://doi.org/10.36227/techrxiv.19248446.v1
10. Csaba G, Raychowdhury A, Datta S, Porod W (2018) Computing with coupled oscillators: theory, devices, and applications. In:2018 IEEE international symposium on circuits and systems (ISCAS), Florence, 2018, pp 1–5. https://doi.org/10.1109/ISCAS.2018.8351664
11. Nikonov DE et al (2015) Coupled-oscillator associative memory array operation for pattern recognition. IEEE J Explor Solid State Comput Devices Circuits 1:85–93. https://doi.org/10.1109/JXCDC.2015.2504049
12. Raychowdhury A et al (2019) Computing with networks of oscillatory dynamical systems. Proc IEEE 107(1):73–89. https://doi.org/10.1109/JPROC.2018.2878854
13. Shamsi J, Avedillo MJ, Linares-Barranco B, Serrano-Gotarredona T (2021) Hardware implementation of differential oscillatory neural networks using VO_2-based oscillators and memristor-bridge circuits. Front Neurosci 15:674567. https://doi.org/10.3389/fnins.2021.674567
14. Shukla N, Tsai W-Y, Jerry M, Barth M, Narayanan V, Datta S (2016) Ultra low power coupled oscillator arrays for computer vision applications. In: 2016 IEEE symposium on VLSI technology, june 2016, pp 1–2. https://doi.org/10.1109/VLSIT.2016.7573439
15. Velichko A, Belyaev M, Boriskov P (2019) A model of an oscillatory neural network with multilevel neurons for pattern recognition and computing. Electronics 8(1):75. https://doi.org/10.3390/electronics8010075
16. Todri-Sanial A et al (2021) How frequency injection locking can train oscillatory neural networks to compute in phase. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2021.3107771
17. Delacour C et al (2021) Oscillatory neural networks for edge AI computing, pp 326–331. https://doi.org/10.1109/ISVLSI51109.2021.00066

18. Hoppensteadt FC, Izhikevich EM (2000) Pattern recognition via synchronization in phase-locked loop neural networks. IEEE Trans Neural Netw 11(3):734–738. https://doi.org/10.1109/72.846744
19. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci 79(8):2554–2558. https://doi.org/10.1073/pnas.79.8.2554
20. Duan S, Dong Z, Hu X, Wang L, Li H (2016) Small-world Hopfield neural networks with weight salience priority and memristor synapses for digit recognition. Neural Comput 27:9
21. Personnaz L, Guyon I, Dreyfus G (1986) Collective computational properties of neural networks: new learning mechanisms. Phys Rev A 34(5):4217–4228. https://doi.org/10.1103/PhysRevA.34.4217
22. Morris RGM, Hebb DO (1949) The organization of behavior. Wiley, New York. Brain Res Bull 50(5–6):437 (1999). https://doi.org/10.1016/S0361-9230(99)00182-3
23. Storkey A (1997) Increasing the capacity of a hopfield network without sacrificing functionality. In: Gerstner W, Germond A, Hasler M, Nicoud J-D (eds) Artificial neural networks—ICANN'97, vol 1327. Springer, Berlin, pp 451–456. https://doi.org/10.1007/BFb0020196
24. Cireşan D, Meier U, Schmidhuber J (2023) Multi-column deep neural networks for image classification. arXiv, 13 février 2012. Consulté le: 14 février 2023. [En ligne]. Disponible sur: http://arxiv.org/abs/1202.2745
25. Li S (2022) aSTDP: a more biologically plausible learning. arXiv:2206.14137
26. Deng L (2012) The MNIST database of handwritten digit images for machine learning research [Best of the Web]. IEEE Signal Process Mag 29(6):141–142. https://doi.org/10.1109/MSP.2012.2211477
27. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L (2009) ImageNet: a large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, 20–25 June 2009, Miami, FL
28. Belyaev MA, Velichko AA (2020) Classification of handwritten digits using the Hopfield network. In: IOP conference series: materials science and engineering, vol 862, p 052048. https://doi.org/10.1088/1757-899X/862/5/052048
29. Scellier B, Bengio Y (2017) Equilibrium propagation: bridging the gap between energy-based models and backpropagation. Front Comput Neurosci 11:24. https://doi.org/10.3389/fncom.2017.00024
30. Movellan JR (1991) Contrastive Hebbian learning in the continuous Hopfield model. In: Connectionist models, Elsevier, 1991, pp 10–17. https://doi.org/10.1016/B978-1-4832-1448-1.50007-X
31. Abernot M et al (2021) Digital implementation of oscillatory neural network for image recognition applications. Front Neurosci. https://doi.org/10.3389/fnins.2021.713054
32. Nikonov DE et al (2020) Convolution inference via synchronization of a coupled CMOS oscillator array. IEEE J Explor. Solid State Comput. Devices Circuits 6(2):170–176. https://doi.org/10.1109/JXCDC.2020.3046143
33. Carapezzi S et al (2021) Advanced design methods from materials and devices to circuits for brain-inspired oscillatory neural networks for edge computing. IEEE J Emerg Sel Top Circuits Syst 11(4):586–596
34. Corti E, Gotsmann B, Moselund K, Ionescu AM, Robertson J, Karg S (2020) Scaled resistively-coupled VO2 oscillators for neuromorphic computing. Solid State Electron 168:107729. https://doi.org/10.1016/j.sse.2019.107729
35. Jackson T, Pagliarini S, Pileggi L (2018) An oscillatory neural network with programmable resistive synapses in 28 Nm CMOS. In: 2018 IEEE international conference on rebooting computing (ICRC), McLean, VA, USA, Nov 2018, pp 1–7. https://doi.org/10.1109/ICRC.2018.8638600
36. Shamsi J, Avedillo MJ, Linares-Barranco B, Serrano-Gotarredona T (2020) Oscillatory Hebbian rule (OHR): an adaption of the Hebbian rule to oscillatory neural networks. In: 2020 XXXV conference on design of circuits and integrated systems (DCIS), Nov 2020, pp 1–6. https://doi.org/10.1109/DCIS51330.2020.9268618
37. Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560. https://doi.org/10.1109/5.58337
38. Ernoult M, Grollier J, Querlioz D, Bengio Y, Scellier B (2021) Updates of equilibrium prop match gradients of backprop through time in an RNN with static input. ArXiv190513633 Cs Stat, mai 2019, Consulté le: 3 décembre 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1905.13633
39. Zoppo G, Marrone F, Bonnin M, Corinto F (2022) Equilibrium propagation and (memristor-based) oscillatory neural networks. In: 2022 IEEE international symposium on circuits and systems (ISCAS), Mar 2022, pp 639–643. https://doi.org/10.1109/ISCAS48785.2022.9937762
40. Laydevant J, Ernoult M, Querlioz D, Grollier J (2021) Training dynamical binary neural networks with equilibrium propagation. ArXiv210308953 Cs, avr. 2021, Consulté le: 9 novembre 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/2103.08953
41. Kendall J, Pantone R, Manickavasagam K, Bengio Y, Scellier B Training end-to-end analog neural networks with equilibrium propagation. arXiv, 9 June 2020. Consulté le: 22 février 2023. [En ligne]. Disponible sur: http://arxiv.org/abs/2006.01981
42. O'Connor P, Gavves E, Welling M (2019) Training a spiking neural network with equilibrium propagation. In: Proceedings of the twenty-second international conference on artificial intelligence and statistics, Apr 2019, pp 1516–1523. Consulté le: 22 février 2023. [En ligne]. Disponible sur: https://proceedings.mlr.press/v89/o-connor19a.html
43. Liu J, Gong M, He H (2019) Deep associative neural network for associative memory based on unsupervised representation learning. Neural Netw 113:41–53. https://doi.org/10.1016/j.neunet.2019.01.004
44. Abernot M, Todri-Sanial A (2023) Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures. Neuromorphic Comput Eng. https://doi.org/10.1088/2634-4386/acb2ef
45. Kohonen T (1972) Correlation matrix memories. IEEE Trans Comput C–21(4):353–359. https://doi.org/10.1109/TC.1972.5008975
46. Kosko B (1988) Bidirectional associative memories. IEEE Trans Syst Man Cybern 18(1):49–60. https://doi.org/10.1109/21.87054
47. Yang Z, Wang X (2021) Memristor-based BAM circuit implementation for image associative memory and filling-in. Neural Comput Appl 33(13):7929–7942. https://doi.org/10.1007/s00521-020-05538-7
48. Abernot M, Gil T, Todri-Sanial A (2022) Oscillatory neural network as hetero-associative memory for image edge detection. In: Neuro-inspired computational elements conference, New York, NY, USA, March 2022, pp 13–21. https://doi.org/10.1145/3517343.3517348
49. Qin S, Mudur N, Pehlevan C (2021) Contrastive similarity matching for supervised learning. Neural Comput 33(5):1300–1328. https://doi.org/10.1162/neco_a_01374