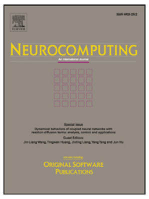
Neurocomputing xxx (xxxx) xxx

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



A classifier task based on Neural Turing Machine and particle swarm algorithm

Soroor Malekmohamadi Faradonbe[a,b], Faramarz Safi-Esfahani[a,b,*]

[a] Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran
[b] Big Data Research Center, Najafabad Branch, Islamic Azad University, Najafabad, Iran

A R T I C L E I N F O

Article history:
Received 27 February 2018
Revised 2 July 2018
Accepted 25 July 2018
Available online xxx

Keywords:
Neural networks
Deep learning
Neural Turing Machine
Particle swarm optimization (PSO)
Classification

A B S T R A C T

Deep learning in artificial intelligence looks for a general-purpose computational machine to execute complex algorithms similar to humans' brain. Neural Turing Machine (NTM) as a tool to realize deep learning approach brings together Turing machine that is a general-purpose machine equipped to a long-term memory, and a neural network as a controller. NTM applies simple controllers to execute several simple/complex tasks such as copy, sort, *N*-gram, etc.; however, complex tasks such as classifications are neglected, and there is no control over improving the weights of NTM, either. This paper presents a framework called PSO–NTM that improves the accuracy of NTM using the LSTM deep neural network as the controller, and implements a complex classification task along with available tasks. Particle Swarm Optimization (PSO) algorithm is also applied in order to control the weights. The classifier task is compared with the basic SVM, KNN, Naïve Bayesian, and Decision Tree classification methods on MNIST, ORL, letter recognition, and ionosphere datasets. The accuracy of the proposed classification tasks is 99.73%, 97.9%, 99.02%, and 97.1%, respectively. That means the NTM classification task improved Naïve Bayesian 43.57%, Decision Tree 15.6%, and KNN 19.22% on average. In addition, the presented framework improves the available NTM tasks as well.

1. Introduction

Deep learning consists of techniques to train multi-layer neural networks [1], similar to the human's brain that attempts to learn abstract patterns to solve cognitive problems [1]. However, neural networks suffer from limitations in storing data over long periods of time due to the lack of an external storage. One of the ways to resolve the memory limitation of neural networks is using Neural Turing Machine (NTM). The base line NTM presented by Google [2] is used in a variety of learning systems due to applying an external memory along with a neural network controller, and attempts to decrease learning speed and improve learning accuracy. NTM by default uses recurrent neural networks along with an external addressable memory [2]; while implements algorithmic tasks such as sorting, copying, *N*-Gram, etc. [3]. The NTM neural controller directs read/write heads on a tape-like external memory [4]. In fact, NTM is a neural network that is able to read/write from/to an external memory similar to a random memory on a regular computer and uses its memory to manipulate complex data structures; however, like a neural network, it learns from data.

In [5], a machine-learning model derived from NTM is presented to change the memory architecture called Differentiable Neural Computer (DNC). This model uses the similar NTM controller with read/write heads that are used to access the memory matrix. DNC is also used to resolve the memory limits, and information is retained only if it is repeated a certain number of times. Research work [6] innovates two tasks to find the shortest path between specified graphs (such as transport networks) and guess the lost links in the stochastically created graphs (such as family trees). Other methods also change the NTM controller such as Lie Access NTM [7], RL NTM [8], and Dynamic NTM [9], which are employed to implement both copy and add tasks. Research paper [10] presents Evolving Neural Turing Machine (ENTM), which is an evolutionary version of NTM and develops Neuro Evolution of Augmenting Topologies (NEAT) algorithm [11] as an instance of ENTM. NEAT is a special form of the genetic algorithm [12] and unlike traditional methods, it does not require an initial design for the NTM controller. However, it is able to evolve the topology and weights of the NTM controller [13]. In general, NEAT is an evolutionary technique that finds the topology and correct weights of a network in order to maximize the performance of each task [14].

* Corresponding author at: Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran.
E-mail address: fsafi@iaun.ac.ir (F. Safi-Esfahani).

https://doi.org/10.1016/j.neucom.2018.07.097
0925-2312/© 2019 Elsevier B.V. All rights reserved.

Actually, the above research works suffer from the following shortcomings: (1) the NTM controller does not employ a deep neural network; (2) there is no independent classification task implemented by the NTM; (3) the NTM controller's weights are not adjustable that is analogous to low convergence analogues to low learning speed.

Memory size and learning rate are the independent variables in this research, while the dependent variables are accuracy and learning speed. Learning rate is a coefficient in the range [0…1] that determines how the current network's weights are affected by the previous weights. In fact, it is an adaptive variable that is used to avoid trapping the network into local optima. Accuracy refers to the closeness of a measured value to a standard or known value. It is usually calculated using mean square formula at the testing time. Learning speed is measured at the training time and is the convergence speed of the method. It is also equal to the number of iterations (epochs) to reach the desired error. The lower the number of epochs, the higher the learning speed.

The hypotheses of this research are as follows: (1) applying the NTM method in implementing the new classification task improves the accuracy of learning due to resolving the memory shortage problem; (2) applying the particle swarm optimization (PSO) algorithm in training the NTM controller improves the accuracy of tasks due to the rapid convergence of the PSO algorithm; (3) applying LSTM in designing the NTM controller along with the PSO algorithm results in improving the accuracy of performing tasks.

The main purpose of this study is to present a new NTM task as well as improving the accuracy of the NTM method in performing available tasks. In order to realize the proposed idea, a framework called PSO–NTM is presented that (1) supports the new classification task along with maintaining other tasks; (2) finds the best weights for the NTM controller by using the PSO algorithms that results in faster convergence; (3) uses the architecture of the deep neural network (LSTM) as the controller of the NTM method, and (4) uses a feedback loop from the external memory to reduce the learning error.

In addition, the study of the time complexity is excluded from this research as the execution speed is not in the scope of this research. Also, the general classification methods such as Support Vector Machine (SVM) that generalizes linear classifiers [15,16], KNN [17], Naïve Bayesian [18], Decision Tree [19], and LSTM [20] neural network as well as the NTM basic method are used as the basic classifiers to evaluate and compare the performance of the proposed method. The experiments are performed and compared with a variety of tasks (including copy, repeat copy, associative recall, *N*-gram, and priority sort). The classification tasks including English handwriting digits recognition task on the MNIST[1] dataset [21], face recognition task using the ORL[2] dataset [22], English letter recognition task using the UCI letter recognition[3]dataset, and ionosphere recognition[4] task using the UCI ionosphere dataset. The results show that the proposed method significantly improves the NTM method. The proposed method is able to perform NTM tasks such as copy task, repeat copy task, *N*-gram task, associative recall task, and priority sort task. The accuracy of the proposed classification task, compared to SVM, KNN, Naïve Bayesian, Decision Tree, LSTM, and NTM for digit recognition is improved by 5.56%, 2.82%, 43.57%, 34.32%, 3.24%, and 1.05%, respectively. It also improves face recognition by 13.18%, 5.36%, 20.61%, 9.36%, 3.66%, and 2.61%, respectively. In addition, it improves English letter recognition, 15.09%, 9.39%, 5.99%, 15.6%, 3.7%, and 1.29%, respectively. It improves ionosphere recognition, 16.3%, 19.22%, 14.38%, 12.5%, 5.84%, and 2.42%, as well.

In the sequel, Section 2 reviews the literature and related research works on NTM. Section 3 outlines the proposed method and then Section 4 explains the experiments and discusses the obtained results. Section 5 outlines conclusions and future research directions.

2. Literature review and concepts

2.1. Particle swarm optimization

The particle swarm optimization (PSO) that is a metaheuristic algorithm was first introduced by Kennedy and Eberhart 1995 [23]. It is a technique for finding approximate solutions for np-hard problems. In fact, the PSO has been derived from the behavior of groups, such as groups of fish or birds [23–26]. The PSO contains a set of virtual particles, each of which provides the capacity of the best solution for the problem. The pseudo-code of the PSO algorithm is shown in Fig. 1.

The PSO, in particular, can be used to train neural networks [27–32]. Accordingly, this study uses the PSO to find the best weight vector with the least error between the calculated output values and the training set output values that finally results in a more exact classification. The PSO is also the basic swarm intelligence method that is recommended for setting up the weights in NTM for the first time in this current research.

2.2. Deep learning and deep LSTM neural network

Deep learning is based on the representation of knowledge and features in the layer models [33]. In other words, it is a class of machine learning techniques with many layers that utilizes the nonlinear processing of information to extract features [34]. The goal of deep learning is the hierarchical learning of features, in which high-level features are used along with low-level ones [35]. Most of the recent developments in deep learning are based on the implementation of complex learning tasks along with the employment of an external memory [2,36–38].

The LSTM neural network [20] is a specific architecture for the recurrent neural network (RNN) that works better than conventional neural networks for long-term tasks [39]. Deep LSTM architecture is achieved by placing several LSTMs hierarchically. In this way, the output sequence of the lower LSTM is the input of the higher one [40]. As a deep neural network, the LSTM has been very successful in solving several problem such as voice recognition, object tracking, etc. [41–43]. The deep LSTM architecture is formally described as follows [43,44]. where l denotes the layers index, Eq. (1) represents the binary sigmoid function, and Eqs. (2)–(6) (h_t^l‘ i_t^l‘ f_t^l‘ s_t^l and o_t^l), respectively, represent the activity vectors of hidden, input, forget, status, and output gates.

$$\sigma(x) = \frac{1}{1+\exp(-x)}[3] \tag{1}$$

$$\boldsymbol{h}_t^l = \boldsymbol{o}_t^l \tanh\left(\boldsymbol{s}_t^l\right)[3] \tag{2}$$

$$\boldsymbol{i}_t^l = \sigma\left(W_i^l\left[\boldsymbol{X}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}\right] + \boldsymbol{b}_i^l\right)[3] \tag{3}$$

$$\boldsymbol{f}_t^l = \sigma\left(W_f^l\left[\boldsymbol{X}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}\right] + \boldsymbol{b}_f^l\right)[3] \tag{4}$$

$$\boldsymbol{s}_t^l = \boldsymbol{f}_t^l \boldsymbol{s}_{t-1}^l + \boldsymbol{i}_t^l \tanh\left(W_s^l\left[\boldsymbol{X}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}\right] + \boldsymbol{b}_s^l\right)[3] \tag{5}$$

$$\boldsymbol{o}_t^l = \sigma\left(W_o^l\left[\boldsymbol{X}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}\right] + \boldsymbol{b}_o^l\right)[3] \tag{6}$$

[1] http://yann.lecun.com/exdb/mnist/
[2] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[3] https://archive.ics.uci.edu/ml/datasets/letter+recognition
[4] https://archive.ics.uci.edu/ml/datasets/ionosphere

```
For each particle
      Initialize particle
  END
  Do
      For each particle
          Calculate fitness value
          If the fitness value is better than the best fitness value (pBest) in history
              set current value as the new pBest
      End
      Choose the particle with the best fitness value of all the particles as the gBest
      For each particle
          Calculate particle velocity according Equation (4-1)
          Update particle position according Equation (4-2)
      End
```

Fig. 1. High-level pseudo-code of the PSO algorithm.

The Soft Max function, for a simple N dimension problem is defined in the following equation:

$$S_N = \left\{ \alpha \in \mathbb{R}^N : \ \alpha_i \in [0,1], \ \sum_{i=1}^{N} \alpha_i = 1 \right\} \tag{7}$$

2.3. Neural Turing Machine (NTM)

Human cognitive science is a main source of inspiration and a criterion for the different learning methods in various fields of machine learning and artificial intelligence [4,10]. An important concept of our cognitive science is the ability to store information and change our behavior based on information stored in memory. Although machine learning has been developed in various aspects [45,46], the progress of smart agents with the long-term memory remains unclear [10]. One method for demonstrating adaptive behavior in the field of learning is in the development of artificial neural networks (ANNs) [47], for which purposeful efforts have been made so that networks can be learned online and can use past experiences [48–52]. Adaptive neural networks can be changed through learning rules [53] in which the weights of the connections are corrected based on the neural network's activation function [48,50], or recurrent neural networks (RNNs) stored the activation patterns through repeated connections [49].

It can perhaps be said that the first idea of using neural networks that can solve any problem was proposed by Jordan Pollack [54] who demonstrated a special recurrence network model called a "neural machine" for "neural Turing". In this basic model, all the neurons are updated at the same time by using their previous activity values [55]. In 2014, a model called Neural Turing Machine (NTM) was introduced to illustrate that adding an external memory to a traditional recursive neural network can greatly improve its performance [7].

As shown in Fig. 2, the NTM [2] includes a controller and a matrix of memory. The controller can be a recurrent or feedforward neural network; by training the network and receiving input and reading from memory, it returns the output to the outside world. Memory is constructed by a large matrix of N memory locations such that each of the vectors M are dimensioned. In addition, the existences of a number of read and write heads facilitated the interaction between the controller and the matrix of memory [4]. It improves the capabilities of recurrent neural networks to perform complex algorithmic tasks such as sorting, etc. The NTM is a general model that can be trained by using the propagation algorithm using input–output examples [8]. Compared to a Turing machine, the NTM model directs read and write heads for direct communication with a tape-like external memory [4]. By enhancing the network's capabilities and connecting the network to external memories, NTM as a hybrid system can be used in various processes. The NTM is also trainable applying different learning methods unlike Turing machine [56] or Von Neumann architecture [2,57].

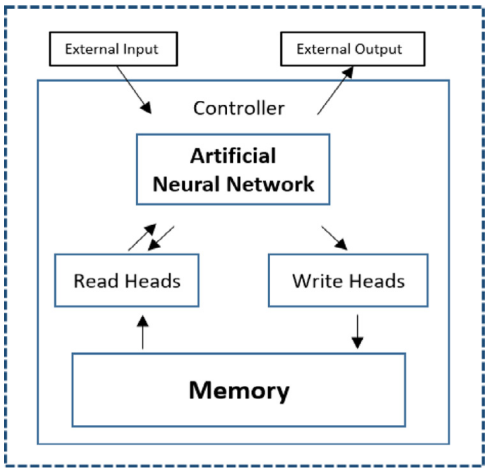


Fig. 2. Neural Turing Machine (NTM) [2].

2.3.1. Tasks implemented in various NTM-based methods

Graves et al. [2] selected five algorithmic tasks for testing the performance of the NTM model. The algorithmic task means that for each task, the final output for each received input is calculated by a simple program and it can easily be implemented and run with each of the common programming languages. The preliminary results in [2] shows that NTM is able to inference the algorithms, such as copying, sorting, and associative recall, etc. with input and output samples. For example, in the copy function, the input is a sequence of fixed-length binary vectors with a limiting symbol and the output goal is to provide a copy of the input sequence. Another task constitutes sorting by priority in which the input contains a sequence of binary vectors with a numerical value of a given priority for each vector and the output goal is to provide a sequence of ordered vectors based on their priorities. These experiments are designed to measure whether an NTM can be trained by supervised learning for the correct and effective implementation of common algorithms. Interestingly, the solutions obtained from this method can be generalized to form longer inputs than the proposed training set. The LSTM without external memory, on the other hand, can be generalized to form longer inputs [2,4]. Various NTM tasks has been implemented by a variety of research works as illustrated in Table 1 that are reviewed in more detail from now on.

Table 1
Implemented tasks by using different methods of NTM.

Tasks methods	NTM [2,7,21,52]	RL-NTM [8]	ENTM [10]	LANTM [8]	D-NTM [9,58]	DNC
Addition	√	–	–	√	–	–
Arithmetic	√	–	–	–	–	–
Assignment	√	–	–	–	–	–
Associative	√	–	–	–	–	–
bAbI	–	–	–	–	√	√
Bigram flip	–	–	–	√	–	–
Block puzzle	–	–	–	–	–	√
Copy	√	√	√	√	–	√
Double	–	–	–	√	–	–
Duplicate input	–	√	–	–	–	–
Forward reverse	–	√	–	–	–	–
Graph	–	–	–	–	–	√
N-gram	√	–	–	–	–	–
Priority sort	√	–	–	–	–	–
Repeat copy	√	√	–	–	–	–
Reverse	–	√	–	√	–	–
T-Maze	–	–	√	–	–	–
XML	√	–	–	–	–	–

Copy task. This task was first introduced in [2] to tests whether the NTM can store a long sequence of arbitrary information and then recall it. This task is represented by a sequence of random binary vectors with a separator flag. Storage and access to information after long periods of time has always been a problem for RNNs and other dynamic architectures [2,7,8,10,59,60]. To perform this behavior, it is necessary to copy the symbols from the input tape to the output tape, get an incoming sample, store it in the memory, and then generate the same symbols.

Repeat copy task. The *repeat copy* task is to extend the copy task so that the sequences are copied a specific number of times and then a marker is put at the end of the sequences. The main motive was to understand if NTM was able to learn a simple nested function. The network receives sequences of random length from binary vectors, followed by a numerical value of the desired number of copies. Networks are trained to reproduce sequences of size-8 random binaries, in which the length of the sequence and the number of repetitions are randomly selected from 1 to 10. The input of the algorithm indicates the number of repetitions, which is normal with a mean of 0 and a variance of 1 [2,60].

Associative recall task. In this task, one item is defined as a sequence of binary vectors, which is bounded by separator symbols on the left and the right. After having given a few items to the network, by showing a random item from the network, we want to generate the next item; each item contains three 6-bit binary vectors (altogether, there are 18 bits per item). In the experiments, the minimum amount is two items and the maximum amount is 6 items [2].

N-gram task. The goal of the task is to determine whether the NTM can quickly adapt itself to a new prediction distribution. The memory is used as a rewriteable table to maintain the number of statistical transmissions, resulting in a simple *N*-gram imitation model. For this purpose, all the 6-gram distributions are considered on binary sequences. Each 6-gram distribution can be a table of $2^5 = 32$ numbers. To determine if the next bit will be one, all probabilities of the binary histories will be given a length of 5. First, the random probabilities of 6-gram will be generated so that all the 32 probabilities of the distribution beta (½, ½) are drawn independently. Then, a special training sequence is created, drawing 200 consecutive bits using the current search table. The network sees the sequence of one bit and then requests the prediction of the next bit. The optimal estimator for this problem can be determined by analysis of the Bayesian [2,61].

Priority sort task. This task tests the NTM's ability to sort data that is an important primary algorithm. A sequence of random binary vectors is provided as the input to the network with a scalar priority value for each vector. This priority is uniformly drawn in the domain $[-1, 1]$. The target sequence consists of binary vectors sorted by priority. Each input sequence contains 20 binary vectors with corresponding priorities and each target sequence has a maximum priority of the 16 vectors at the input. For the best performance, there is a need for 8 parallel read and write heads with a feed-forward controller that reflects the difficulty of sorting the vectors by only using a single operation [2].

2.4. Review of related works

As shown in Fig. 3, the basis of research on intelligent machines dates back to more than 70 years—at the same time as the creation of the first electronic computers [56]. Designing general-purpose algorithms has always been one of the long-term goals of artificial intelligence [9]. One of the first attempts to create an intelligent machine was observed during the invention of the Turing machine by Alan Turing in 1936. Initially, the Turing program had a major impact on psychology and cognitive philosophy; both of these subject areas considered the computer to be a metaphor for the functional model of the brain. This metaphor was quickly put aside by neuroscience because of the dissimilarity of the architecture of the Turing machine in comparison to the human brain [62].

Artificial Intelligence (AI) is attempting to make a really smart machine; the current state of computers is far inferior to this and human intelligence is still ahead of computer intelligence. Neural networks are a small part of AI and the human brain should be called a biological neural network (BNN). Many contexts do not distinguish between BNN and artificial neural networks [1]. The brain is a computer that is very complex, nonlinear, and parallel [63]. In human cognitive operation, the process resembles an electronic operation known as "working memory". While the working memory mechanism remains somewhat vague in terms of the physiology of nerves, one definition of working memory is as follows: the working memory is a capacity for understanding the short-term storage of information and control based on rules that are understandable [3,64,65]. From the computational point of view, these rules are simple programs and the stored information is proof of these programs. The NTM that was first proposed by Grave et al. [2] in 2014 consists of recurrent neural networks along with an external addressable memory. This work improves the performance and the capabilities of recurrent neural networks for performing algorithmic tasks such as sorting, classifying, copying, *N*-gram, etc. [3]. Compared to a Turing machine, the NTM model, in the form of a program, directs read and write heads for direct communication with the tape-like external memory [4]. NTM is

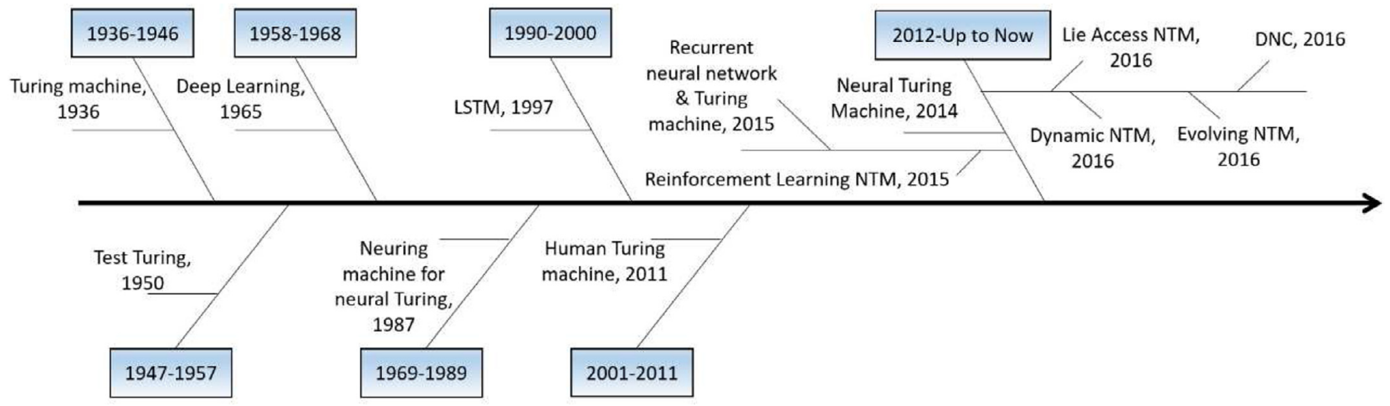


Fig. 3. Timeline of activities in this area.

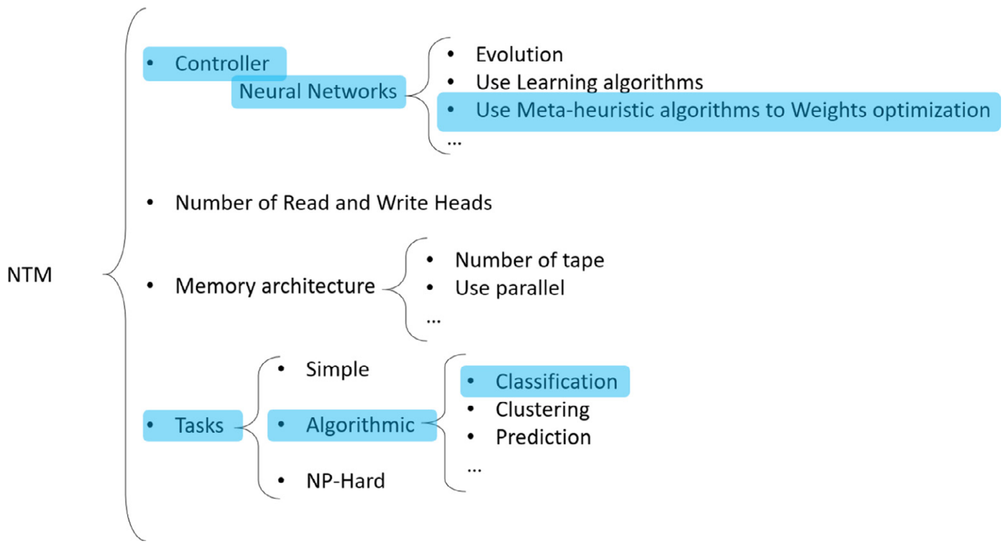


Fig. 4. Mind-map of neural turing machine.

actually a neural network that has the ability to read and write on an external memory matrix, similar to random memory on a regular computer. The NTM neural network uses its memory to display and manipulate complex data structures and, simultaneously, the same neural network can learn from data.

Fig. 4 illustrates that the research works on the NTM are focusing on four main aspects (1) controller; (2) the number of heads; (3) memory architecture; and (4) tasks. The highlighted area of the Fig. 4 shows the mainstream of this current research paper, where a new classification task is presented along with a metaheuristic method to setup the NTM controller's weights.

Differentiable Neural Computer (DNC) as a model of machine learning is derived from the NTM. In this model, a similar neural network controller is used with read-write head to access to the memory matrix. However, the memory usage is different in that the information is retained if it is repeated a number of specified times. This method, like a typical computer, employs memory to display and modify the structure of data; however, like a neural network, it can learn from the data. Therefore, the use of this method is shown as supervised learning, which can respond to combined questions designed in natural languages (using the bAbI data set) [5]. It also has the ability to perform tasks such as finding the shortest path between specified points (such as a transportation network) and guessing missing-link in randomly generated graphs such as a family tree [6].

The Evolving Neural Turing Machine (ENTM) is actually an evolutionary version of the NTM [10]. The ENTM components are the same as the original NTM architecture: an ANN controller and an external memory bank. The ENTM method also has the ability to solve simple tasks, such as copy, and for the first time, the double T-Maze sequencer version as a reinforcement learning problem. In the T-Maze learning task, the agent uses memory to adapt to a behavior (which normally requires a neural network). The performance of this method is much better than the copy job in the original NTM and it can also be extended to larger sequences [10].

The ENTM method was developed with NEAT algorithm [11] that is a special form of genetic algorithm [12]. In general, NEAT is an evolutionary technique that finds the topology and correct weight of a network in order to maximize performance in a task [14]. In this algorithm, unlike traditional methods, there is no need to design ANN topology; instead, it has the ability to create and evolve the topology and ANN weights [13]. NEAT starts with a small population of a simple neural network, which adds complexity by adding nodes and new connections through mutations.

Several more research works improve the performance of the NTM controller. Dynamic Neural Turing (DNTM) method [9] is the NTM along with a trainable addressable memory. Two address and content vectors let the network learns from the provided linear or non-linear memory addresses [9]. Reinforcement Learning NTM (RLNTM) [8] is able to look at the part of an environment partially;

Table 2
Structural comparison of NTM methods.

Method	External memory	Network structure (controller)	Deep learning	Learning algorithm	Type of optimization	Type of implementation tasks
NTM	√	Feed forward/ LSTM	√	Backpropagation/ reinforcement learning	Deterministic (gradient descent)	Simple algorithmic tasks (copy, repeat copy, arithmetic, addition, priority sort) [2] –Prediction [62]
ENTM	√	RNN	×	N EAT	Non-deterministic (NEAT)	Simple algorithmic task (Copy) [10]
DNC	√	Deep-LSTM	√	Reinforcement learning	Deterministic (gradient descent)	Simple algorithmic task (Copy) [6] – Q&A [6] – Graph [6]
LSTM	×	–	√	Backpropagation	–	Algorithmic [20]
PSO-NTM (proposed framework)	√	Deep-LSTM	√	PSO	Non-deterministic (PSO)	Simple Algorithmic tasks – classification

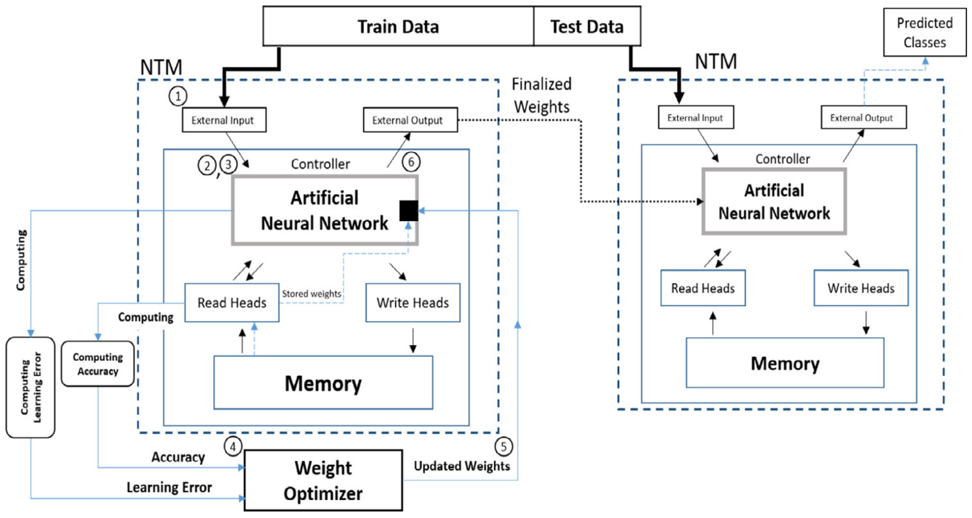


Fig. 5. The proposed PSO–NTM framework.

and affects the environment as a whole that is the philosophy of reinforcement learning. In fact, that part of the model which interacts with an environment is the controller and learns [8]. The authors in [7] apply the Lie Access network in the NTM controller called LANTM in some simple algorithmic tasks.

Table 2 compares the mentioned research works on NTM in terms of using external memory, controller, supporting deep learning, the method of optimization, and the implemented tasks.

3. Proposed method

Fig. 5 presents the PSO–NTM framework that includes both training and testing sectors. The PSO algorithm is used in the weight optimizer to adjust weights, and the NTM controller applies the LSTM network. According to Fig. 6, the input vectors are first received and the neural network is trained in each iteration using the PSO algorithm. Receiving each 100 input vectors, the mean of accuracy is calculated based on Eq. (11) and then they are stored in memory as old weights in order to be used in future calculations. Finally, the outputs are calculated based on the network weights. The pseudocode of the learning process of the neural network is shown in Figs. 7–10.

3.1. Optimization algorithm

PSO–NTM framework applies the PSO algorithm to train the neural network in order to find the right weights among many others due to its suitable convergence rate. As shown in Fig. 8, in the second algorithm, the particles are initialized so that the number of particles is equal to the number of input vector length. The weights are then initialized to the speed of each input vector. The

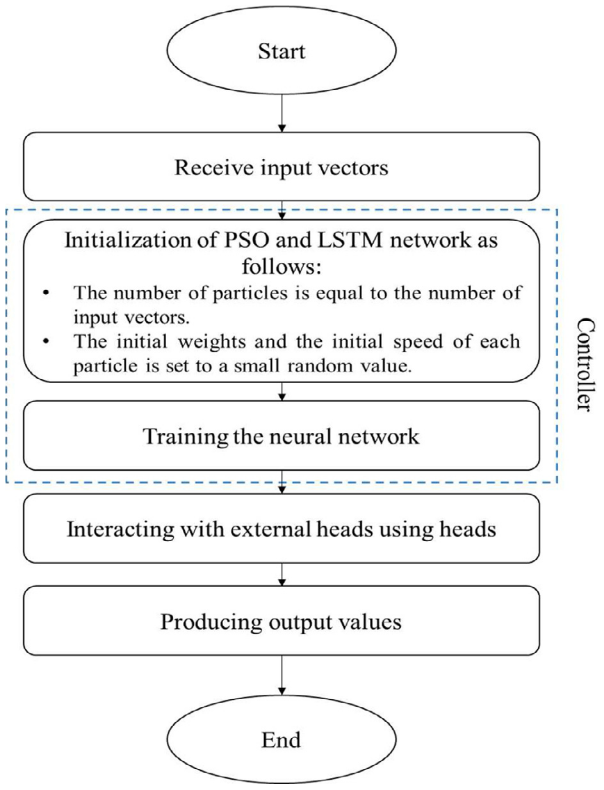

Start
Receive input vectors
Initialization of PSO and LSTM network as follows:
• The number of particles is equal to the number of input vectors.
• The initial weights and the initial speed of each particle is set to a small random value.
Training the neural network
Controller
Interacting with external heads using heads
Producing output values
End

Fig 6. The flowchart of the recommended PSO–NTM.

```
1  Equation 1 : $f_t^l = \sigma\left(W_f^l\left[X_t; h_{t-1}^l; h_t^{l-1}\right] + b_f^l\right)$
2  Equation 2 : $s_t^l = f_t^l s_{t-1}^l + PSO\ weights\ Update$
3  Equation 3 : $\left[\sum_{i=0}^{lenght\ of\ input}(known\ Output[i] - machine\ output[i])^2\right]^{1/2}$
4  Equation 4 : $v_t = v_{t-1} \times \omega + c_1 \times rand() \times \left(g_{best,t} - x_t\right) + c_2 \times rand() \times (p_{best,t} - x_t)$
5  Equation 5 : $x_t = x_{t-1} + v_t$
```

Fig. 7. Equations used in the algorithms.

```
   Initialize LSTM Network parameters
   Input: number of layer, number of input node, number of hidden node, number of output node
   Output: topology of LSTM(set the heads)
1      Number of Layer = L
2      Number of input node = length of sequence vector
3      Number of hidden node = H
4      Number of output node = number of classes
5  End

   Initialize Particle
   Input: number of particle
   Output: Weights, Velocities
1      Number of particle = length of sequence vector
2      Set random Weights in (0, 1)
3      Set random Velocity in (0, 1)
4      Set$c_1$, $c_2$ and $\omega$
5  End
```

Fig. 8. Pseudocode to initialize the neural network and the PSO algorithm.

```
    Algorithm 1: NTM-PSO.LSTM&PSO training algorithm
    Input: Read Training sequence vectors from Dataset
    Output: best Weights
1       Set initialize LSTM network parameters (Initialize LSTM Network parameters)
2       Repeat:
3           Decision for forget gates with Equation 1
4           Use reader-head for reading pervious weights from external Memory
5           Use PSO algorithm to Update weights
6           Decision for output gates with Equation 2
7           Set outputs
8           Check the error with root-mean-square deviation (Equation 3)
9           Use write-head for writing best weights to external Memory
10      Until met criteria
11      Return best output
12      Calculate accuracy
13  End
```

Fig. 9. High-level pseudocode for training the deep LSTM neural network using the PSO algorithm.

particle is considered random and very small (in the range (0, 1)). The dimensions of each particle are the number of weights associated with the network. Particles move in their weight space and try to minimize learning errors. Switching the positions is analogous to updating the network weights to reduce the error in the current epoch. In each epoch, the particles update their position by calculating a new velocity according to Eq. (8) and moving to a new position, which is calculated using Eq. (9) [66].

$$v_t = v_{t-1} \times \omega + c_1 \times rand() \times \left(g_{best,t} - x_t\right) + c_2 \times rand() \times \left(p_{best,t} - x_t\right)[66] \quad (8)$$

$$x_t = x_{t-1} + v_t[66] \quad (9)$$

The new position is a set of new weights to get a new error. This process is repeated and the particle with the least learning error is considered as the best particle. The training process continues until a satisfactory error is obtained by the best particle or by reaching the computational limits (the number of epochs for instance). When the training is completed, weights are used to calculate the classification error for training patterns. For new patterns, the same set of weights is used to test the network. Both values of P_{best}(the least error of learning each particle) and g_{best} (the least learning error found in the entire learning process) are used in Eq. (6) to generate certain amounts of position matching with the best solution or the learning objective errors, c_1 andc_2. The acceleration coefficients are random numbers in the interval (0, 1), and the amount of inertial weight (ω) that is the learning

```
Algorithm 2: PSO algorithm
Input: LSTM Weights
Output: best Weights
1   For each particle
2     Initialize particle (Initialize Particle)
3   End
4   For each particle
5     Calculate fitness value
6     If the fitness value is better than the best fitness value (pBest) in history
7       set current value as the new pBest
8     End
9     Choose the particle with the best fitness value of all the particles as the gBest
10    For each particle
11      Calculate particle velocity according Equation 4
12      Update particle position according Equation 5
13    End
14  End
```

Fig. 10. High-level pseudocode for the PSO algorithm to optimize the weights of the LSTM neural network.

rate was set to 0.6, experimentally. The inertial weight guarantees that particles are moving in their path.

3.2. PSO–NTM controller

Given the pseudocode presented in Fig. 10, in each time step t, the controller implemented by the LSTM neural network, receives an input vector $(x_t \in \mathbb{R}^X)$ from the environment and returns an output vector $(y \in \mathbb{R}^y)$, which generates a distribution for the target vector $z_t \in \mathbb{R}^y$ (for supervised learning). In addition, this controller has a set of R input vectors $(r^1_{t-1}, \ldots, r^R_{t-1})$ for the memory matrix $(M_{t-1} \in \mathbb{R}^{N\times M})$, and it also receives the previous weights by read heads, and then, creates an interface vector I_t that specifies the head and the memory interactions in the current step. For convenience, both input vectors are considered as an input vector as $X_t = [x_t;\ r^1_{t-1}; \ldots;\ r^R_{t-1}]$ and the calculations are carried out as described in Section 2.1. Network training is performed by both PSO algorithm and the mean squared error. Both the number of epochs and the least error analysis are performed after the PSO.

As indicated in the learning network pseudocode, the network weight update section is performed by the PSO algorithm. In this way, Eq. (5) that is required for the training of the LSTM deep neural network is changed as follows. The weights are initialized randomly, while the learning rate and the memory size are considered as 0.6 and 128, empirically.

$$\boldsymbol{s}^l_t = \boldsymbol{f}^l_t \boldsymbol{s}^l_{t-1} + PSO\ updated\ weights \tag{10}$$

The state of memory cells are initialized to zero and then the initial value of all the gates and their activation functions are calculated using the Equations in Section 2.1. After calculating the output of the cells, the output error is calculated and compared to the target output. Then, the weights are updated by using the PSO algorithm. In the next step, the modified weights are transmitted to memory by using write heads to be used in later periods. Of course, the process of deleting values from memory is performed through the network's forget gate. The final output are the number of epochs, and accuracy calculations.

3.3. Case study

In this section, the recommended method is studied by an example that shows the presented classifier on the well-known FisherIris dataset,[5] concisely, while this example is also discussed in Appendix A with more details. Table 4 includes the processing steps of the dataset. It includes 150 instances (rows), 4 columns, and a class that determines the three species of the Iris flower including (Setosa, Versicolor, Virginica) each one includes 50 instances. One of the groups is linear classifiable, and two other groups are non-linear. The variables are initialized according to Table 3. The recommended neural network includes 4 input nodes, 128 hidden nodes, and 3 output nodes. In addition, 70% of datasets is considered for training, 20% for test, and the rest is to prevent overfitting as cross validation.

Table 3
Initialization of parameters in the case study.

Sample input	$a_1 = 5.1,\ a_2 = 3.5,\ a_3 = 1.4,\ a_4 = 0.2$
Random weights	$w_1 = 0.1551,\ w_2 = 0.0359,\ w_3 = 0.0667,\ w_4 = 0.1541$
Memory size	$M = 128$
Learning rate	$\omega = 0.6$

The first phase: receiving the input vectors that is as follows:

{5.1, 3.5, 1.4, 0.2, 0, 0, 1}#Setosa
{7.0, 3.2, 4.7, 1.4, 0, 1, 0}#Versicolor
{6.3, 3.3, 6.0, 2.5, 1, 0, 0}#Virginica

The second phase: the initialization of weights is performed based on a random fashion according to Table 3. In addition, learning rate, and memory size are determined, experimentally.

The third phase: the initial value of the cells is equal to zero. Then the initial value of the gates, and their activations is calculated based on Eqs. (1)–(4) In the case that the memory is not empty, read heads the weights stored in the memory are used as well.

It is assumed that the number of nodes in input layer is 4, in hidden layer is 2, and in output layer is 3.

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$
$$\boldsymbol{h}^l_t = \boldsymbol{o}^l_t \tanh\left(\boldsymbol{s}^l_t\right)$$
$$\boldsymbol{i}^l_t = \sigma\left(W^l_i\left[\boldsymbol{X}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t\right] + \boldsymbol{b}^l_i\right)$$
$$\boldsymbol{f}^l_t = \sigma\left(W^l_f\left[\boldsymbol{X}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t\right] + \boldsymbol{b}^l_f\right)$$

[5] https://archive.ics.uci.edu/ml/datasets/iris

Table 4
Obtained results from the case study.

Method for classification	Training data of IRIS dataset	Learning rate	Memory	Iterations	Error
Proposed method	80	0.6	128	40	0.0

Table 5
Summery of dataset's features parameters.

Dataset	Number of samples	Sample size	Target
MNIST	60,000 gray images	Size: 28×28 Range: [0:255]	Size:1 Range: 0–9
ORL	400 facial images	Size: 92×112 Range: 8bit	Size:1 Range: 1–40
Letter recognition	20,000 letter pictures	Vector Size: 16 Range: [0:15]	Size:1 Range: one of 26 alphabet
Ionosphere	351	Vector Size: 34 Range: [−1:1]	Size:1 Range: "g" or "b"

The fourth phase: applying PSO algorithm, the values of the weights are amended and updated.

$$v_t = v_{t-1} \times \omega + c_1 \times rand() \times \left(g_{best,t} - x_t\right) + c_2 \times rand() \times \left(p_{best,t} - x_t\right)$$
$$x_t = x_{t-1} + v_t$$

The fifth phase: the amended weights are transferred to the memory by the writing heads to be used later. However, the networks' forget gate removes the weights from the memory that is called removing process.

$$\boldsymbol{s}_t^l = \boldsymbol{f}_t^l \boldsymbol{s}_{t-1}^l + PSO\ weights\ Update$$

The sixth phase: after calculating output cells and output node, the final output is obtained. The difference between the calculated output and the target output is calculated, afterwards. Calculating the number of iterations and accuracy is performed in the final phase.

$$\boldsymbol{s}_t^l = \boldsymbol{f}_t^l \boldsymbol{s}_{t-1}^l + PSO\ weights\ Update$$
$$\boldsymbol{o}_t^l = \sigma\left(W_o^l\left[\boldsymbol{X}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}\right] + \boldsymbol{b}_o^l\right)$$

The steps of test phase are very similar to the training phase. The only difference is that the weights are not updated and the final output and accuracy are calculated applying the previous weights and the values stored in the memory. Table 4 shows Detail of this experiment.

3.4. Experimental environment

The experimental environment is illustrated in Fig. 11. In fact, the overall structure of the proposed method along with its implementation tools is shown. The computer system used for this research is the Acer Aspire F15 laptop with 8 GB of RAM and Intel Core i7-7500U processor (2.7 GHz). Python programming language along with PyTorch package are used to implement the proposed method. For visualized outputs, Visdom that is a web platform was used.

The number of epochs in this research is considered to be 100,000 [2]. It is also assumed that the NTM consists of a two-layer LSTM neural network with deep architecture as a controller as well as the use of two heads for reading and writing [2]. The backpropagation is the most common technique to train the neural network [67]. In the backpropagation network, there is a need to consider values for both parameters of momentum and learning rate. This network is very sensitive to these values and a very small change can have a great effect on the network behavior. The particle swarm optimization (PSO) algorithm also requires the inertia weight values (the amount of influence on the movement of a particle in its path), which is considered as the learning rate in this research.

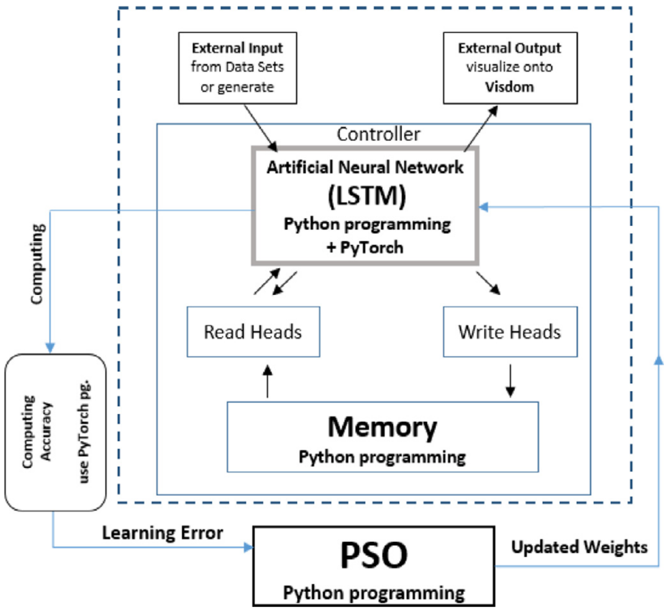

External Input from Data Sets or generate
External Output visualize onto Visdom
Controller
Artificial Neural Network (LSTM) Python programming + PyTorch
Computing
Read Heads
Write Heads
Memory Python programming
Computing Accuracy
use PyTorch pg.
Learning Error
PSO Python programming
Updated Weights

Fig. 11. Experimental environment.

The main issue for using PSO in training is that the LSTM neural network is usually slower than backpropagation techniques [68]. A combination of machine-learning methods and PSO produces the best results in comparison to conventional classification algorithms [69,70]. Therefore, using this algorithm to train the NTM controller and its deep training improves the performance of NTM; the result of the classification can be improved by using the PSO technique [30]. In addition, the LSTM neural network as an NTM's controller is applied for two reasons: (1) the superiority of the LSTM compared to other neural networks [35]; and (2) LSTM has been used in several basic approaches [2,6].

3.5. Datasets used in experiments

According to Table 5, four datasets MNIST, ORL, Letter Recognition, and Ionosphere are used to compare the proposed classification task in comparison to the similar classification methods. The datasets are presented in detail in the sequel.

3.5.1. MNIST dataset

In order to meet the need for handwriting digits recognition, the US National Institute for Standards and Technology (NIST) created a handwritten database [71]. Modified NIST (MNIST)[6] [21], a

[6] http://yann.lecun.com/exdb/mnist

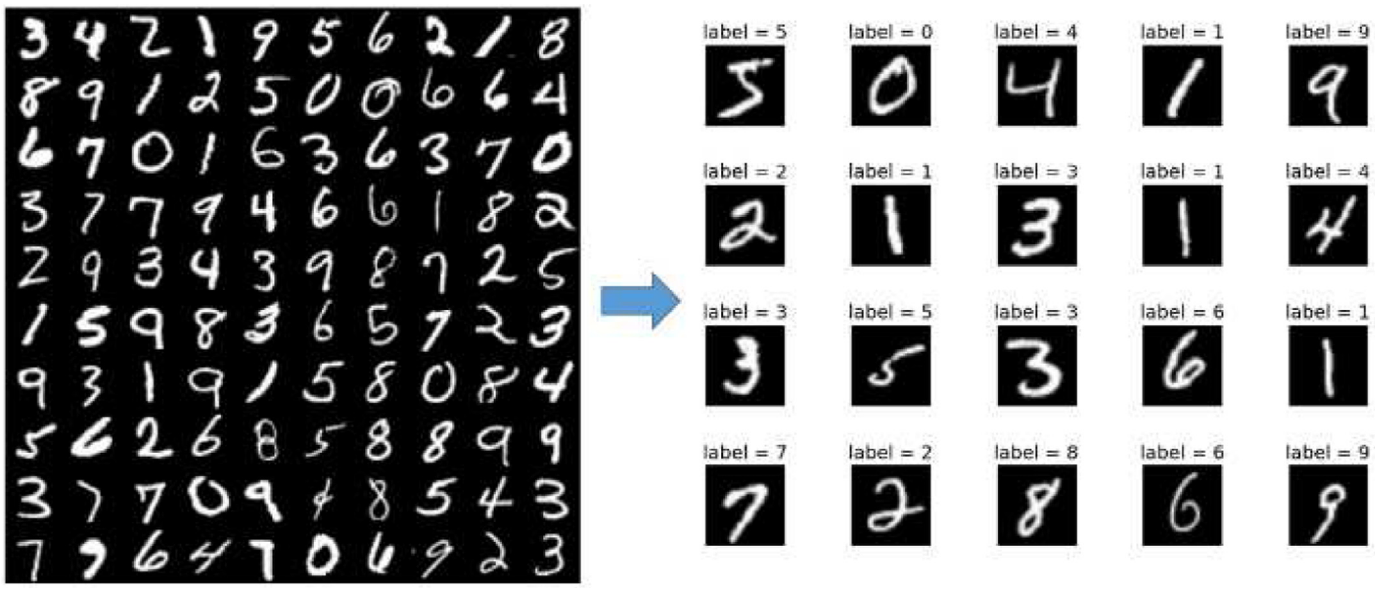


Fig. 12. Sample of MNIST dataset.

smaller and normalized set with 60,000 Gy images of 28 × 28 pixels (the amount of each pixel in range is [0, 255] for learning and 10,000 images for testing, labeled with (From 0 to 9) indicates which image each digit represents. To avoid correlations, the MNIST dataset is written by different individuals [72]. In Fig. 12, you can see an example of this dataset. The MNIST-free dataset for handwriting digits recognition has a fast-testing standard to test machine-learning algorithms [73]. And a well-known collections that are commonly used in supervised classification research [74]. In this study, the .csv format of this dataset was used in which each image is a 784 × 1 matrix (array with 784 lengths) and the blocks of this matrix include the brightness intensity of each pixel of the image; 80% of the data was training data, 20% of the data was test data, and 10% of the training data was used as cross-validation.

3.5.2. ORL dataset

The main purpose of this task is to recognize the correct new image by using the classification. The ORL data consists of 40 facial images and 10 images per person at different times with different background lights, and different modes for image-processing and facial recognition, which were developed by the University of Cambridge from 1992 to 1994.[7] In this collection, pictures were saved in the pgm format in 92 × 112 sized 8-bit grayscale images. This dataset is a standard dataset deployed for use in the field of robotics and computer vision.

3.5.3. Letter recognition dataset

The goal is to identify a large number of rectangles, including black and white pixels, as one of the 26 alphabets. Letter recognition is a UCI standard dataset[8] definition for this task and contains 20,000 picture of letters that have been randomly written in 20 different font types, which have been converted to a numeric value in the 0–15 range by considering their features.

3.5.4. Ionosphere dataset

The ionosphere is part of the upper part of the atmosphere in which the presence of free electrons affects the transmission of waves and reflects or returns radio waves to the Earth. These electrical loads change during the day and night, and are affected by solar spots. The task is to detect the existence of an ionosphere. This dataset consists of 351 samples with 35 features, in which the values of the first 34 features, the continuous variables in the range of [−1,+1] and the thirty-sixth feature is the variable of the classification consisting of "g" (good) or "b" (bad) according to the previous values.[9] This data is collected using a radar in Labrador and this system consists of 16 high-frequency antennas with a total power of 6.4 kW; when the radar receives evidence of the existence of the ionosphere, the value "good" should be returned; otherwise, the value “bad" should be returned.

3.6. Experimental parameters

The dependent variables that are measured in the experiments are accuracy and learning speed presented as follows:

1. Accuracy: The error rate is calculated based on the input and the output values during training; these are expressed as percentages. Accuracy is calculated at the time of testing the method using Eq. (11) that is the mean square error. It calculates the mean square of the difference between the correct and the predicted values for each sample.

$$\left[\sum_{i=0}^{lenght\ of\ input} (known\ Output[i] - machine\ output[i])^2\right]^{1/2} \Big/ n \tag{11}$$

2. Learning Speed: Learning speed is measured at the training time and is the convergence speed of the method. It is also equal to the number of iterations (epochs) to reach the desired error. The lower the number of epochs, the higher the learning speed.

Both memory size and learning rate are set up experimentally, when the best results are obtained by the tasks copy, duplicate, sorting by priority, and classification on the iris dataset.

[7] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[8] https://archive.ics.uci.edu/ml/datasets/letter+recognition
[9] https://archive.ics.uci.edu/ml/datasets/ionosphere

Table 6
Experimental design table.

Category of experiments	No.	Goal	Adjustment parameters	Describe the experiment	Evaluated parameters
Experiments to obtain the best values for learning speed and memory size	1	Check the learning speed of the proposed method	Memory size: constant Learning rate: variable	On Copy Task and classification of iris dataset	Accuracy, Learning Speed
	2	Check the learning speed of the proposed method	Memory size: variable Learning rate: constant	On Copy Task and classification of iris dataset	Accuracy, Learning Speed
Experiments to compare the proposed method with the NTM base method in accurate implementation of tasks	3	Check the accuracy of the proposed method compared to the NTM base method	Uses the parameters set in previous experiments	1. Copy Task 2. Repeat Copy Task 3. *N*-gram Task 4. Associative Task 5. Priority Sort Task	Accuracy
Experiment to evaluate the classification task	4	Check the accuracy the classification of the proposed method	Use of parameters set in previous experiments	1. Digit Recognition Task on MNIST Dataset 2. Facial Recognition on ORL Dataset 3. Letter Recognition on Letter Recognition Dataset (UCI) 4. Ionosphere Recognition on ionosphere Dataset (UCI)	Accuracy

Table 7
The comparison methods used in this study.

Method	Adjustments
Support Vector Machine (SVM)	$C = 10$, gamma = 0.01, linear kernel
K Nearest Neighbors (KNN)	$K = 3$, Euclidean distance function
Naïve Bayesian	$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_{i=1}^{n} P(a_i \mid v_j)$
Decision Tree	Max-depth = 5
Long Short Term Memory (LSTM) Neural Network	2-layer, activation function = tanh, 128 hidden neurons
Neural Turing Machine (NTM)	Controller = feedforward, memory size = 128, learning rate = 10^−4

Table 8
Memory size adjustment for copy task.

Copy task				
Memory size	32	64	128	256
Accuracy (%)	93.17 ± .91	95.02 ± 2.13	99.16 ± 0.94	–

Table 9
Memory size adjustment for Iris flowers classification task.

Iris classification task				
Memory size	32	64	128	256
Accuracy (%)	91.01 ± 4.34	94.42 ± 2.03	99.14 ± 0.86	–

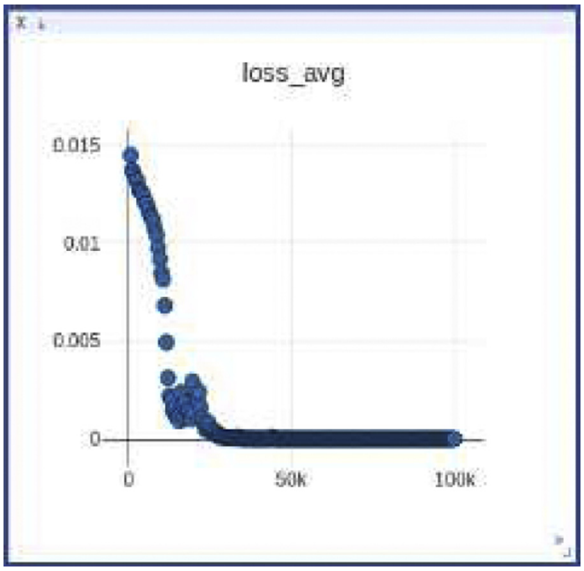


Fig. 13. Learning speed with memory size = 32.

3.7. Experimental design

Table 6 illustrates the design of the experiments that are arranged into three general categories: (1) experiments to obtain the best values for the learning speed and memory size; (2) comparison of the proposed method with the NTM basic method in terms of the accuracy of the available tasks; and (3) experiments to examine the new classification task.

3.7.1. Experimental parameter set up

Table 6 shows the adjustment variables of the proposed method in terms of memory size and learning rate. In order to perform the experiments, it is necessary to consider the appropriate value for these parameters in order to achieve the desired results. Two experiments were arranged in this section to obtain an appropriate amount for the experimental parameters by two tasks including the copy and classification on the iris dataset. In addition, the settings of the base line methods are shown in Table 7.

Experiments for checking the learning-speed and accuracy of the proposed method (variable learning rate and constant memory size). In this experiment, the learning rate was considered as a constant at 0.6 and the memory size was changed. Tables 8 and 9 illustrate four experiments and the amount of accuracy are shown in Figs. 13–18.

According to the experiments, the best memory size is analogues to 128-bit so that more than this amount causes overlapping and time-consuming operations [2].

The learning speed graph in this case has been shown in Figs. 15 and 18 for both tasks including copy and classification on the iris datasets, respectively. In the next experiment, the learning rate is variable and the size of memory is fixed to the constant value 128. The results are shown in Tables 8 and 9.

Experiments for checking the learning-speed and accuracy of the proposed method (constant learning rate, variable memory size). Memory size is the other independent variable that can be

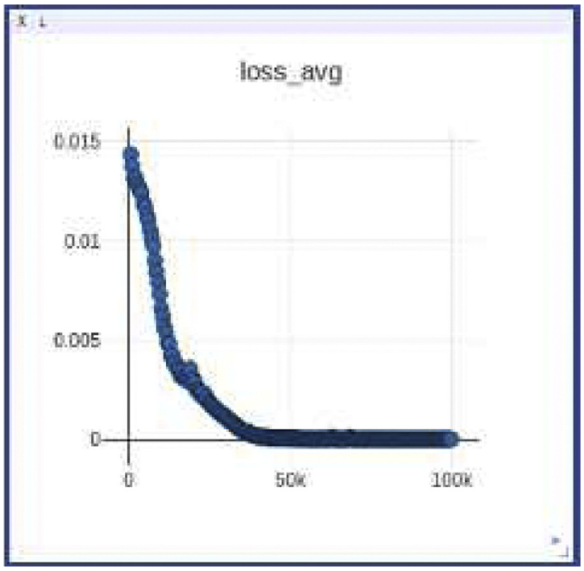


Fig. 14. Learning speed with memory size = 64.

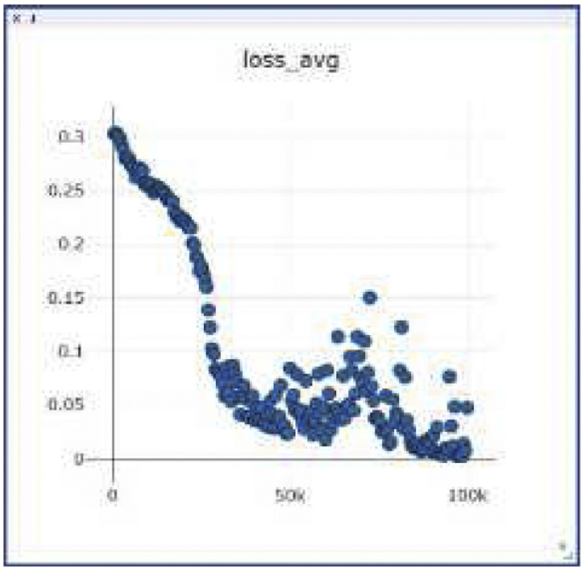


Fig. 15. Learning speed with memory size = 128.

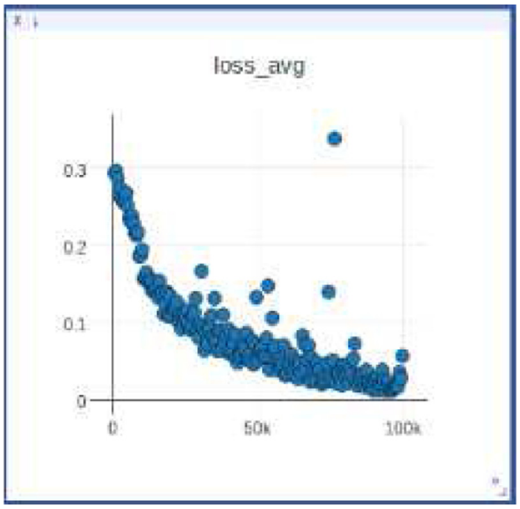


Fig. 16. Learning speed with memory size = 32.

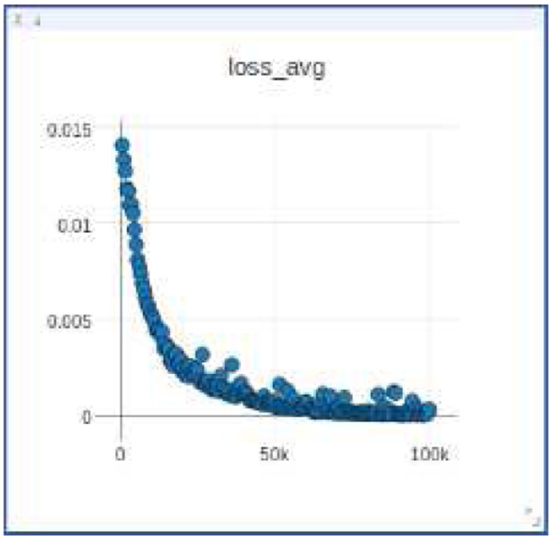


Fig. 17. Learning speed with memory size = 64.

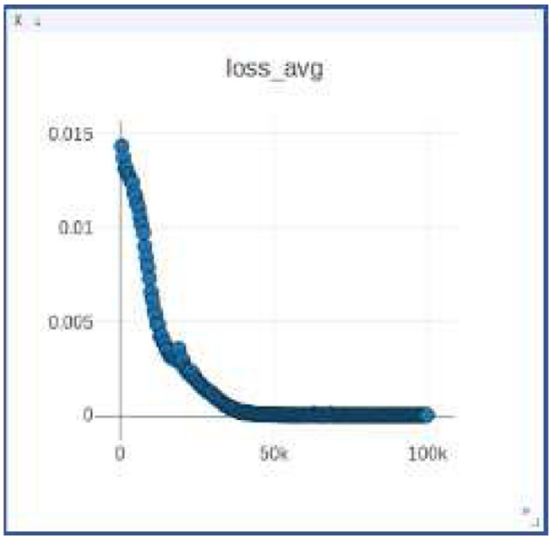


Fig. 18. Learning speed with memory size = 128.

Table 10
Learning rate adjustment for copy task.

Copy task				
Learning rate	0.1	0.5	0.6	0.9
Accuracy (%)	99.16 ± 0.94	85.11 ± 3.25	99.16 ± 0.94	78.46 ± 3.47

changed. In this study, the memory size was fixed to 128 and the learning rate was changed. Tables 10 and 11 show four examples of the test and its effects on accuracy. Figs. 19 –26 show the effects on learning speed.

According to the values provided in Tables 10 and 11, the best result was observed at the learning rate 0.6 for the copy Task as well as the task for classifying iris flowers. The learning speed graphs are shown in Figs. 21 and 25, respectively. Therefore, the best values of the variable parameters of this study are the memory size and the learning rate analogous to 128 and 0.6, respectively. (Figs. 22–24)

3.7.2. Implemented tasks

In this study, two experiments are carried out. (1) The accuracy of the proposed method was checked in comparison to the NTM

Table 11
Learning rate adjustment for Iris flower classification task.

Iris classification task				
Memory size	0.1	0.5	0.6	0.9
Accuracy (%)	74.21 ± 4.06	81.01 ± 2.44	99.14 ± 0.86	89.12 ± 1.05

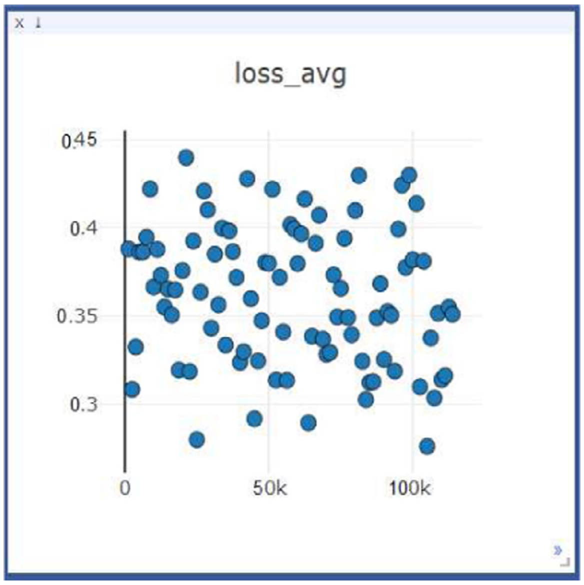


Fig. 19. Learning speed with learning rate = 0.1.

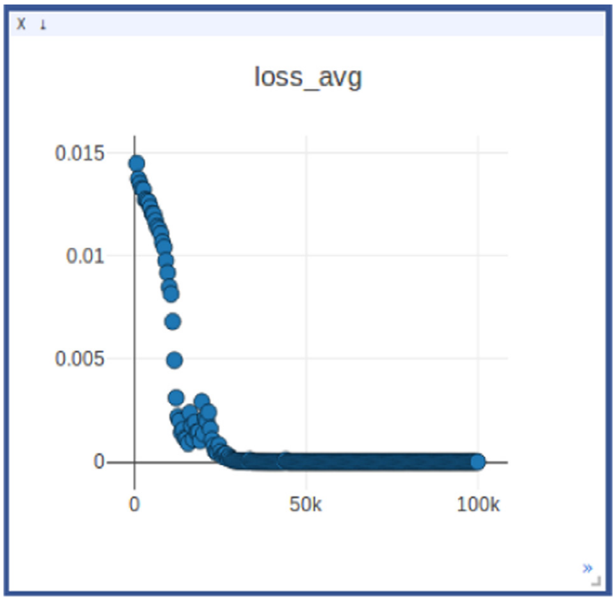


Fig. 21. Learning speed with learning rate = 0.6.

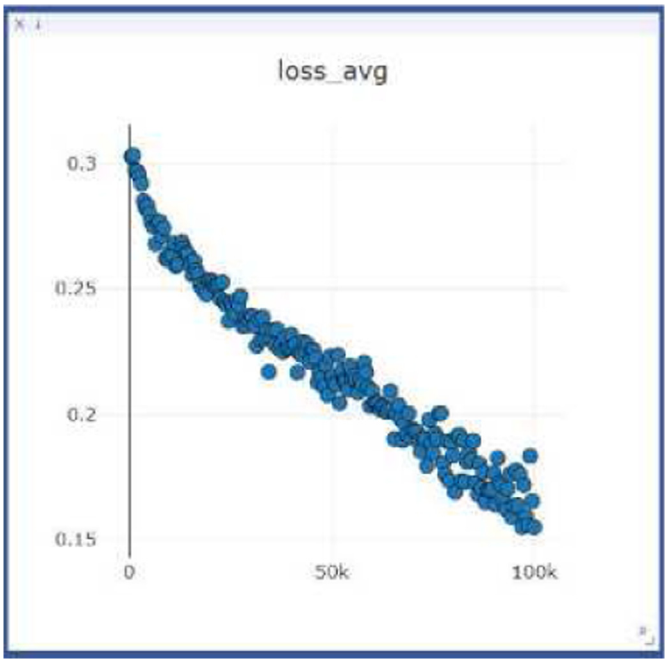


Fig. 20. Learning speed with learning rate = 0.5.

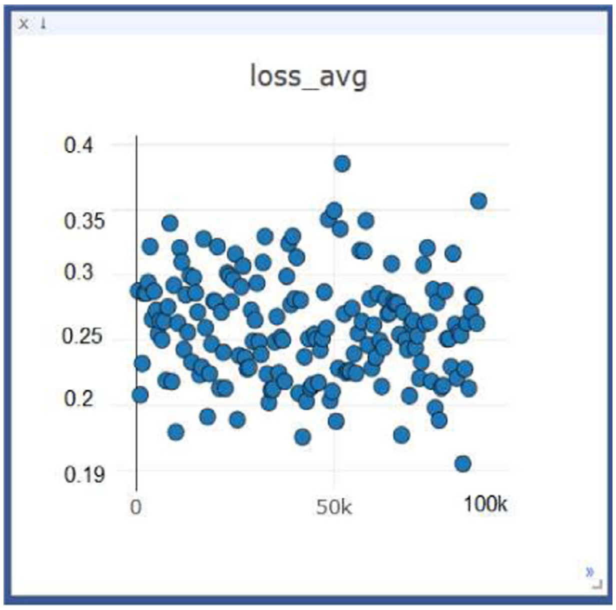


Fig. 22. Learning speed with learning rate = 0.9.

base method. (2) The accuracy of the classification. The average of the results are shown in Tables 13–20B.

3.7.3. Experiments for checking the accuracy of the proposed method compared to the NTM base method

Experiment-1: copy task. In this study, we used 100,000 sample inputs of zero and one vector in a random order (using a function) to train the neural network for this task, similar to the paper [6], and tested the experiment on 100 samples.

According to the results presented in Table 13, it can be seen that all the compared methods, except LSTM, can, in their best results, copy the input sequence as the basic task of this experiment, and, according to the average results, the least result was the LSTM method, and the results dispersion in the two methods of DNC and the proposed method was very low due to the use of external memory.

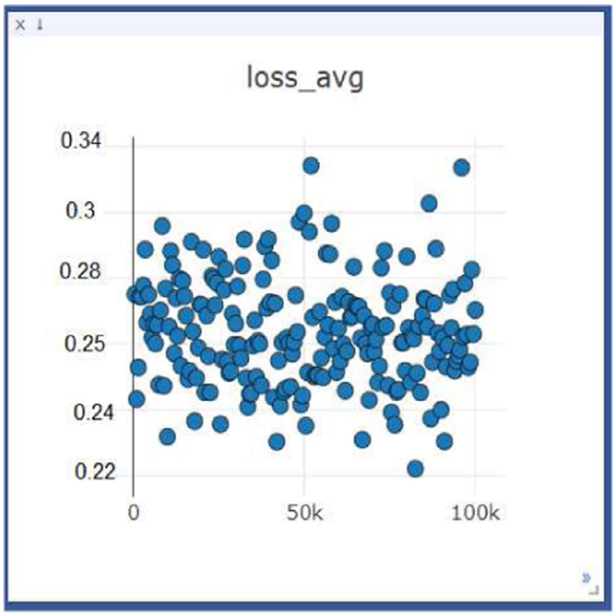


Fig. 23. Learning speed with learning rate = 0.1.

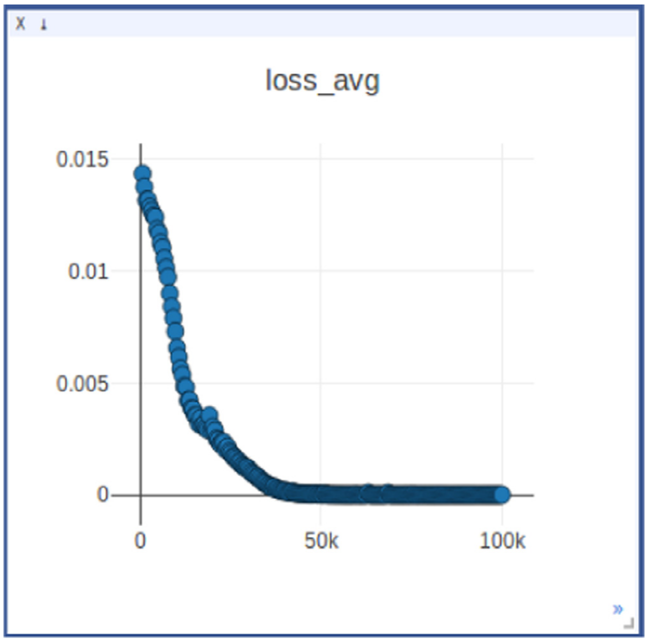


Fig. 25. Learning speed with learning rate = 0.6.

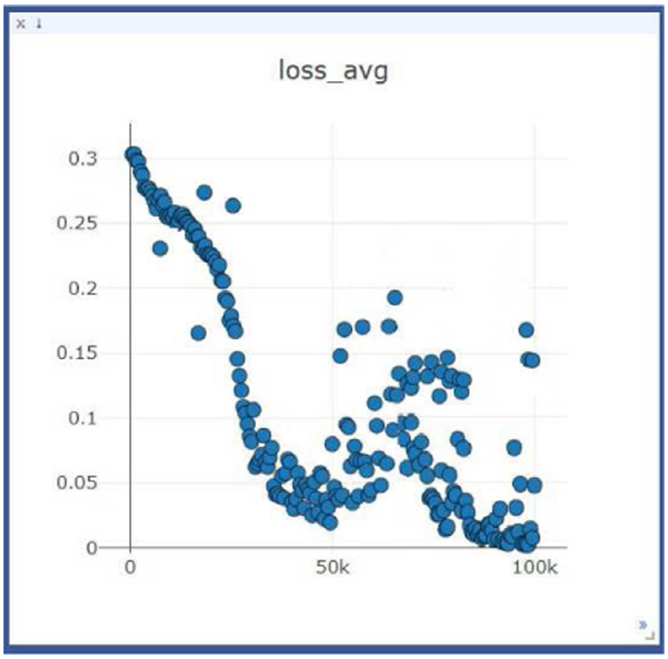


Fig. 24. Learning speed with learning rate = 0.5.

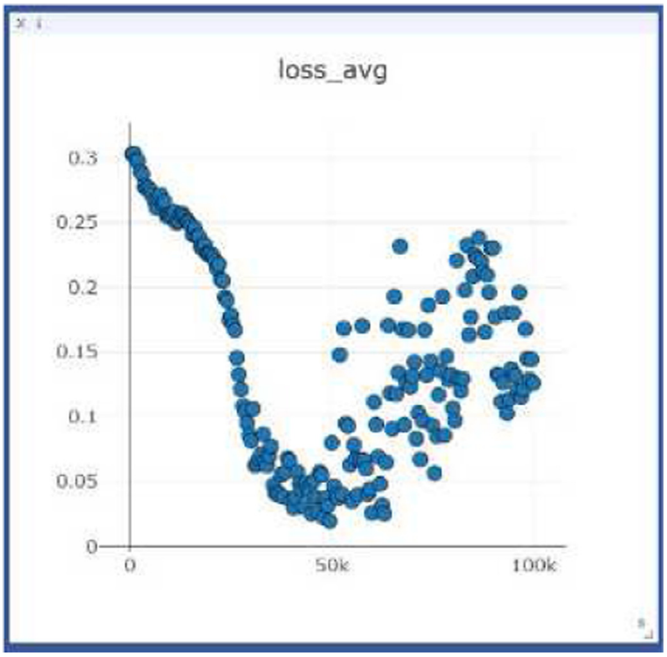


Fig. 26. Learning speed with learning rate = 0.9.

Experiment-2: repeat copy task. The repeat copy task described in Section 2.3.1 was tested according to Table 12, similar to the previous experiment for comparing the proposed method and the NTM base method and the results in Table 14 indicate that the proposed method also has the capabilities of the base method.

Experiment-3: associative recall task. The associative recall task, described in Section 2.3.1, was tested according to Table 11, similar to the previous experiment for comparing the proposed method and the NTM base method; the results in Table 14 indicate that the proposed method also has the capabilities of the base method.

Experiment-4: N-grams task. The *N*-grams task, described in Section 2.3.1, was tested according to Table 12, similar to the previous experiment for comparing the proposed method and the NTM

Table 12
Specifications of tasks used in this research.

Tasks	Number of class	Length of input vector	Number of instance
Copy	–	100	100.000
Repeat copy	–	100	100.000
Associative	–	256	70.330
N-Grams	–	100	61.749
Priority sort	–	512	269.038
Digit Recognition	9	784	70.000
face Recognition	40	10,304	400
letter Recognition	26	15	20.000
ionosphere recognition	2	34	351

Table 13
The results of calculating the accuracy of the performance of methods by the experiment through the copy task.

Average results				
PSO–NTM	ENTM	DNC	NTM	LSTM
99.16 ± 0.94	98.2 ± 1.7	99.26 ± 0.74	97.8 ± 2.2	71.6 ± 11.5
Best results				
PSO–NTM	ENTM	DNC[6]	NTM[2]	LSTM[2]
100	100	100	100	99.2

Table 14
The results of the accuracy of the performance of proposed method by experiment the repeat copy task.

Average results		Best results	
PSO–NTM	NTM	PSO–NTM	NTM [2]
99.81 ± 0.86	98.8 ± 1.4	100	100

Table 15
The results of the accuracy of the performance of the proposed method by the associative recall task experiment.

Average results		Best results	
PSO–NTM	NTM	PSO–NTM	NTM [2]
98.2 ± 2.12	95.9 ± 5.8	100	100

Table 16
The results of the accuracy of the performance of proposed method by the experiment of the N-Grams task.

Average results		Best results	
PSO–NTM	NTM	PSO–NTM	NTM [2]
47.35 ± 2.16	40.12 ± 2.4	49.91	42.06

Table 17
The results of the accuracy of the performance of the proposed method by experiment the priority sort task.

Average results		Best results	
PSO–NTM	NTM	PSO–NTM	NTM [2]
83.68 ± 4.12	80.11 ± 3.24	86.48	84.16

base method. The results in Table 16 indicate that the proposed method also has the capabilities of the base method.

Experiment-5: priority sort task. The priority sort task, described in Section 2.3.1, was tested according to Table 12, similar to the previous experiment, for comparing the proposed method and the NTM base method. The results in Table 17 indicate that the proposed method also has the capabilities of the base method.

3.7.4. Classification experiments

Experiment-1: reviewing the performance of the NTM classifier task based on english digits recognition with the MNIST dataset. In this experiment, the PSO–NTM classification task classifies English digital numbers using MNIST data and perform the task of recognizing English digital numbers. As in usually the case, data are segmented so that 80% of the data is for training, 20% is for the test data, and 10% of the training data exists as a validation. Furthermore, calculating accuracy based on the type of classification of each image is calculated using Eq. (11).

As shown in Table 18, minimum accuracy is related to the decision tree method with 65.40% and the best accuracy is related to the proposed method (PSO–NTM), which is 99.73%. The proposed method, due to the use of the PSO algorithm, was more than 2%

Table 18
The results of English handwriting digit recognition test using the MNIST dataset.

Method	SVM	KNN	Naïve Bayesian	Decision tree	LSTM	NTM	DNC	PSO–NTM
Accuracy (%)	94.16	96.9	56.15	65.40	96.48	96.96	99.12	99.73

Table 19
The results of the face recognition test using the ORL dataset.

Method	SVM	KNN	Naïve Bayesian	Decision tree	LSTM	NTM	DNC	PSO–NTM
Accuracy (%)	84.68	92.5	77.25	88.5	94.2	95.11	97.21	97.9

Table 20
The results of the English letter recognition test using letter recognition dataset.

Method	SVM	KNN	Naïve Bayesian	Decision Tree	LSTM	NTM	DNC	PSO–NTM
Accuracy (%)	84.11	89.81	93.21	83.6	95.5	96.01	98.16	99.02

Table 21
The results of the ionosphere recognition test using the ionosphere dataset.

Method	SVM	KNN	Naïve Bayesian	Decision tree	LSTM	NTM	DNC	PSO–NTM
Accuracy (%)	80.3	77.78	82.62	84.5	91.16	93.41	96.02	97.1

better than the LSTM and the NTM methods in classifying handwritten digits. The error rate of the best result of the proposed method was 0.11% lower than the best reported error rate by the official site.[10]

Experiment-2: review the performance of the NTM classifier task based on face recognition with the ORL dataset. In this experiment, after performing the task of face recognition using the ORL dataset by using the methods described and the proposed method, we compared the accuracy of the methods according to Table 19.

As shown in Table 19, the minimum accuracy obtained from the Naive Bayesian method with 77.25% and the maximum accuracy obtained was related to the proposed method due to the use of external memory. The particle swarm optimization algorithm with 97.9%, which is more than the base method NTM by 2.79%.

Experiment-3: review the performance of the NTM classifier task based on english letter recognition with the letter recognition dataset. In this experiment, the implementation of the task of recognizing English letters by using the letter recognition dataset, and compared with using the proposed methods and described methods. Typically, 16,000 data are considered as the training data, and the rest are considered the test data, which, in our research, we also consider the same as 10% of the training data, which is regarded as valid data.

As shown in Table 20, the percentage of the accuracy obtained with the Decision Tree category is 83.6% and the proposed method is better than the other methods in comparison to 99.02%. Owing to the use of the PSO algorithm for training the controller (neural network) has improved the LSTM and the NTM networks by 3.52 and 3.01%, respectively.

Experiment-4: review the performance of the NTM classifier task based on ionosphere recognition with the ionosphere dataset. In this experiment, with the aid of classify the ionosphere recognition task was performed using one of the UCI databases called ionosphere. The results of the experiments carried out using the methods proposed in this study and the described method are presented.

[10] http://yann.lecun.com/exdb/mnist/

Table 22
The results of the Ionosphere recognition test using ionosphere dataset.

Method	Dataset							
	SVM (%)	KNN (%)	Naïve Bayesian (%)	Decision tree (%)	LSTM (%)	NTM (%)	DNC (%)	PSO–NTM (%)
MNIST	94.16	96.9	56.15	65.40	96.48	96.96	99.12	99.73
ORL	84.68	92.5	77.25	88.5	94.2	95.11	97.21	97.9
Letter	84.11	89.81	93.21	83.6	95.5	96.01	98.16	99.02
Ionosphere	80.3	77.78	82.62	84.5	91.16	93.41	96.02	97.1

According to the information shown in Table 21 the minimum percentage of accuracy obtained is related to the SVM class with 80.3%. Despite the poor performance of other comparable methods in this task, the proposed method, due to the use of external memory and the use of the metaheuristic algorithm for controller training, could have a good performance versus comparable methods with an accuracy of 97.1% and 4.31% improvement in NTM.

3.8. Experimental discussions

According to the presented experiments, one can see that the proposed PSO–NTM framework outweighs the base line methods. The copying task as a basic function is used to adjust the experimental parameters on the iris databases so that memory size and learning speed were set to 128 and 0.6, respectively. The task of recognizing English handwritten numbers using the MNIST dataset and the face recognition task using the ORL dataset showed the ability of the proposed method to resolve problems. Standard UCI datasets, such as letter recognition and ionosphere were used for the English letters recognition task and the ionosphere recognition task, respectively. According to Table 22, the proposed method improved the accuracy and reaches to the accuracy 99.73%, 97.9%, 99.02% and 97.1%, respectively.

4. Conclusions and future works

In recent years, there has been a lot of research on deep learning. Due to the high potential of neural networks, many efforts have been made to improve various learning methods using the neural networks by adding an external memory. An example of these methods is seen in the presentation of the NTM method. The use of the NTM method in the implementation of the classification task, due to resolving the memory shortage improved both accuracy and speed-learning in training. The PSO metaheuristic algorithm was used in training the controller in the NTM method. The classification task improved the accuracy by applying the PSO algorithm.

The main objective of this research was to increase the efficiency and the reliability of the NTM method for classifying tasks, which is presented in the framework of the proposed PSO–NTM in order to realize the proposed idea. The performance of the proposed method has been compared with that of the same family (NTM, LSTM, DNC, and ENTM), and we have shown that the proposed method has significantly improved compared to the NTM base method. According to the results of the tests performed on the tasks, it was shown that the proposed method had the ability to solve the copy task similar to other NTM methods. It has also been shown that the proposed method in comparison to classifier methods such as SVM, KNN, Naïve Bayesian, Decision Tree, LSTM, and NTM, for the task of recognition of English handwritten numbers showed improvements of 5.56, 2.82, 43.57, 34.32, 3.24, and 1.05%, respectively. It also improved for face recognition by 13.18, 5.36, 20.61, 9.36, 3.66, and 2.61%, respectively. The task of recognizing English letters was improved by 15.09, 9.39, 5.99, 15.6, 3.7, and 1.29%, respectively. The recognition of the ionosphere was improved by 16.3, 19.22, 14.38, 12.5, 5.84, and 2.42%, respectively.

Several aspects of this research are recommended as future works as follows: (1) the controller: the evolution of the controller's neural network by applying both NEAT and PSO algorithms or using other metaheuristic algorithms; (2) studying the performance of the presented framework: the PSO–NTM framework can be studied in terms of time complexity and execution speed along with right solutions to resolve the problems; (3) external memory architecture: using a memory structure similar to the DNC method for the proposed method can also improve the performance of this method. In addition, applying special-purpose read/write heads contributes to the performance of the classification task; (4) implementing new tasks: inventing new complex tasks to enrich the NTM to get closer to human's brain.

conflict of interest

There is no conflict of interest in this manuscript.

Appendix A

This section illustrates the case study in Section 3.3 in a numerical format with more details. A part of the Iris dataset is selected and used in initializing the parameters of the LSTM network. The activity of network in forward and backward is shown, and then the initialization and mapping of the network weights to the PSO particles is depicted. It is finally shown that how the network weights are updated, afterwards.

Sample of Iris dataset:

{5.1, 3.5, 1.4, 0.2, 0, 0, 1} #Setosa
{7.0, 3.2, 4.7, 1.4, 0, 1, 0} #Versicolor
{6.3, 3.3, 6.0, 2.5, 1, 0, 0} #Virginica

Initialize the LSTM network parameters:

$$W_h = \begin{bmatrix} 0.45 \\ 0.25 \\ 0.71 \\ 0.12 \end{bmatrix}, \; U_h = [0.15], \; b_h = [0.2]$$

$$W_i = \begin{bmatrix} 0.95 \\ 0.8 \\ 0.14 \\ 041 \end{bmatrix}, \; U_i = [0.8], \; b_i = [0.65]$$

$$W_f = \begin{bmatrix} 0.7 \\ 0.45 \\ 0.61 \\ 0.3 \end{bmatrix}, \; U_f = [0.1], \; b_f = [0.15]$$

$$W_o = \begin{bmatrix} 0.6 \\ 0.4 \\ 0.1 \\ 0.91 \end{bmatrix}, \; U_o = [0.25], \; b_o = [0.1]$$

$$x_0 = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix} \; with\ output : [0 \quad 0 \quad 1] \text{ Label: } 0.5$$

$$x_1 = \begin{bmatrix} 7.0 \\ 3.2 \\ 4.7 \\ 1.4 \end{bmatrix} \; with\ output : [0 \quad 1 \quad 0] \text{ Label: } 1.25$$

Forward t = 0:

$$h_0 = tanh(W_h.x_0 + U_h.out_{-1} + b_h) = \tanh\left(\begin{bmatrix}0.45 & 0.25 & 0.71 & 0.12\end{bmatrix}\begin{bmatrix}5.1\\3.5\\1.4\\0.2\end{bmatrix} + [0.15]\begin{bmatrix}0\\0\\1\end{bmatrix} + [0.2]\right) = 0.81775$$

$$i_0 = \sigma(W_i.x_0 + U_i.out_{-1} + b_i) = \sigma\left(\begin{bmatrix}0.95 & 0.8 & 0.14 & 0.41\end{bmatrix}\begin{bmatrix}5.1\\3.5\\1.4\\0.2\end{bmatrix} + [0.8]\begin{bmatrix}0\\0\\1\end{bmatrix} + [0.65]\right) = 0.96083$$

$$f_0 = \sigma(W_f.x_0 + U_f.out_{-1} + b_f) = \sigma\left(\begin{bmatrix}0.7 & 0.45 & 0.61 & 0.13\end{bmatrix}\begin{bmatrix}5.1\\3.5\\1.4\\0.2\end{bmatrix} + [0.15]\begin{bmatrix}0\\0\\1\end{bmatrix} + [0.15]\right) = 0.85192$$

$$o_0 = \sigma(W_o.x_0 + U_o.out_{-1} + b_o) = \sigma\left(\begin{bmatrix}0.6 & 0.4 & 0.1 & 0.91\end{bmatrix}\begin{bmatrix}5.1\\3.5\\1.4\\0.2\end{bmatrix} + [0.25]\begin{bmatrix}0\\0\\1\end{bmatrix} + [0.1]\right) = 0.81757$$

$$state_0 = h_0 \odot i_0 + f_0 \odot state_{-1} = 0.81775 \times 0.96083 + 0.85195 \times 0 = 0.78572$$

$$out_0 = \tanh(state_0) \odot o_0 = \tanh(0.78572) \times 0.81757 = 0.53631$$

Forward t = 1:

$$h_0 = tanh(W_h.x_0 + U_h.out_{-1} + b_h) = \tanh\left(\begin{bmatrix}0.45 & 0.25 & 0.71 & 0.12\end{bmatrix}\begin{bmatrix}7.0\\3.2\\4.7\\1.4\end{bmatrix} + [0.15]\begin{bmatrix}0\\0\\1\end{bmatrix} + [0.2]\right) = 0.8980$$

$$i_0 = \sigma(W_i.x_0 + U_i.out_{-1} + b_i) = \sigma\left(\begin{bmatrix}0.95 & 0.8 & 0.14 & 0.41\end{bmatrix}\begin{bmatrix}7.0\\3.2\\4.7\\1.4\end{bmatrix} + [0.8]\begin{bmatrix}0\\1\\0\end{bmatrix} + [0.65]\right) = 0.98214$$

$$f_0 = \sigma(W_f.x_0 + U_f.out_{-1} + b_f) = \sigma\left(\begin{bmatrix}0.7 & 0.45 & 0.61 & 0.13\end{bmatrix}\begin{bmatrix}7.0\\3.2\\4.7\\1.4\end{bmatrix} + [0.15]\begin{bmatrix}0\\1\\0\end{bmatrix} + [0.15]\right) = 0.762503$$

$$o_0 = \sigma(W_o.x_0 + U_o.out_{-1} + b_o) = \sigma\left(\begin{bmatrix}0.6 & 0.4 & 0.1 & 0.91\end{bmatrix}\begin{bmatrix}7.0\\3.2\\4.7\\1.4\end{bmatrix} + [0.25]\begin{bmatrix}0\\1\\0\end{bmatrix} + [0.1]\right) = 0.89493$$

$$state_0 = h_0 \odot i_0 + f_0 \odot state_{-1} = 0.8980 \times 0.98214 + 0.762503 \times 0.78572 = 1.4096352$$
$$out_0 = \tanh(state_0) \odot o_0 = \tanh(1.4096352) \times 0.89493 = 0.794176$$

Backward t = 1:

$$E(x,\hat{x}) = \frac{(x-\hat{x})^2}{2} \rightarrow \textit{the derivate is}: \ \partial_x E(x,\hat{x}) = x - \hat{x}$$
$$\Delta_1 = \partial_x E = 0.794176 - 1.25 = -0.455824, \ \Delta_{out_1} = 0$$
$$\delta out_1 = \Delta_1 + \Delta_{out_1} = -0.455824 + 0 = -0.455824$$
$$\delta state_1 = \delta out_1 \odot o_1 \odot (1 - \tanh^2(state_1)) + \delta state_2 \odot f_2 = -0.47803 \times 0.84993 \times (1 - \tanh^2(1.5176)) + 0 \times 0 = -0.07111$$
$$\delta h_1 = \delta state_1 \odot i_1 \odot (1 - a_1^2) = -0.07111 \times 0.98118 \times (1 - 0.84980^2) = -0.01938$$
$$\delta i_1 = \delta state_1 \odot h_1 \odot i_1 \odot (1 - i_1) = -0.07111 \times 0.84980 \times 0.98118 \times (1 - 0.98118) = -0.00112$$
$$\delta f_1 = \delta state_1 \odot state_1 \odot f_1 \odot (1 - f_1) = -0.07111 \times 0.78572 \times 0.87030 \times (1 - 0.87030) = -0.00631$$
$$\delta o_1 = \delta out_1 \odot \tanh(state_1) \odot o_1 \odot (1 - o_1) = -0.47803 \times \tanh(1.5176) \times 0.84993 \times (1 - 0.84993) = -0.05538$$

$$\delta x_1 = W^T.\delta gates_1 = \begin{bmatrix}0.45 & 0.95 & 0.7 & 0.6\\0.25 & 0.8 & 0.45 & 0.4\\0.71 & 0.14 & 0.61 & 0.1\\0.12 & 0.41 & 0.3 & 0.91\end{bmatrix}\begin{bmatrix}-0.01938\\-0.00112\\-0.00631\\-0.05538\end{bmatrix} = \begin{bmatrix}-0.04743\\-0.03073\\-0.0233037\\-0.0550736\end{bmatrix}$$

$$\Delta out_0 = U^T.\delta gates_1 = \begin{bmatrix}0.15 & 0.8 & 0.10 & 0.25\end{bmatrix}\begin{bmatrix}-0.01938\\-0.00112\\-0.00631\\-0.05538\end{bmatrix} = -0.01828$$

Backward t = 0:

$$\Delta_0 = \partial_x E = 0.53631 - 0.5 = 0.03631$$
$$\Delta_{out_0} = -0.01828$$
$$\delta out_0 = \Delta_0 + \ \Delta_{out_0} = 0.03631 \pm 0.01828 = 0.01803$$
$$\delta state_0 = \delta out_0 \odot o_0 \odot (1 - \tanh^2(state_0)) + \delta state_1 \odot f_1 = 0.01803 \times 0.81757 \times (1 - \tanh^2(0.78572)) + -0.07111 \times 0.87030 = -0.05349$$
$$\delta a_0 = \delta state_0 \odot i_0 \odot (1 - a_0^2) = -0.05349 \times 0.96083 \times (1 - 0.81775^2) = -0.01703$$
$$\delta i_0 = \delta state_0 \odot h_0 \odot i_0 \odot (1 - i_0) = -0.05349 \times 0.81775 \times 0.96083 \times (1 - 0.96083) = -0.00165$$
$$\delta f_0 = \delta state_0 \odot state_0 \odot f_0 \odot (1 - f_0) = -0.05349 \times 0 \times 0.85195 \times (1 - 0.85195) = 0$$
$$\delta o_0 = \delta out_0 \odot \ \tanh(state_0) \ \odot o_0 \odot (1 - o_0) = 0.01803 \times \tanh(0.78572) \times 0.81757 \times (1 - 0.81757) = 0.00176$$

$$\delta x_1 = W^T.\delta gates_1 = \begin{bmatrix} 0.45 & 0.95 & 0.7 & 0.6 \\ 0.25 & 0.8 & 0.45 & 0.4 \\ 0.71 & 0.14 & 0.61 & 0.1 \\ 0.12 & 0.41 & 0.3 & 0.91 \end{bmatrix} \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = \begin{bmatrix} -0.00817 \\ -0.00487 \\ -0.0011185 \\ -0.0121463 \end{bmatrix}$$

$$\Delta out_0 = U^T.\delta gates_1 = \begin{bmatrix} 0.15 & 0.8 & 0.10 & 0.25 \end{bmatrix} \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = -0.00343$$

Initialize four particles for PSO:

$x_0 = [-5.1 \quad 0.5 \quad 4.2 \quad 2.3\ ]$, $x_1 = [0.6 \quad 2.1 \quad 8.3 \quad -1.2\ ]$, $x_2 = [-0.8 \quad -5.5 \quad 1.4 \quad 2.4\ ]$,
$x_3 = [-9.1 \quad 6.5 \quad 3.3 \quad -0.1\ ]$

$$v_t = v_{t-1} \times \omega + c_1 \times rand() \times (g_{best,t} - x_t) + c_2 \times rand() \times (p_{best,t} - x_t)$$
$$x_t = x_{t-1} + v_t$$

$v_1 = 0 \times 0.6 + 0.213([-5.1 \quad 0.5 \quad 4.2 \quad 2.3] - [-5.1 \quad 0.5 \quad 4.2 \quad 2.3]) + 0.879([0.6 \quad 2.1 \quad 8.3 \quad -1.2]$
$-[-5.1 \quad 0.5 \quad 4.2 \quad 2.3]) = [5.01 \quad 1.41 \quad 3.6 \quad 3.08]$
$v_2 = [5.01 \quad 1.41 \quad 3.6 \quad 3.08] \times 0.6 + 0.213([0.6 \quad 2.1 \quad 8.3 \quad -1.2] - [0.6 \quad 2.1 \quad 8.3 \quad -1.2]\ ) + 0.879([0.6 \quad 2.1 \quad 8.3 \quad -1.2] - [0.6 \quad 2.1 \quad 8.3 \quad -1.2])$
$= [3.01 \quad 0.85 \quad 2.16 \quad 1.85]$
$v_3 = [3.01 \quad 0.85 \quad 2.16 \quad 1.85] \times 0.6 + 0.213([-0.8 \quad -5.5 \quad 1.4 \quad 2.4] \ - [-0.8 \quad -5.5 \quad 1.4 \quad 2.4]\ )$
$+0.879([0.6 \quad 2.1 \quad 8.3 \quad -1.2] - [-0.8 \quad -5.5 \quad 1.4 \quad 2.4]\ ) = [3.04 \quad 7.17 \quad 7.33 \quad -2.04]$
$v_4 = [3.04 \quad 7.17 \quad 7.33 \quad -2.04] \times 0.6 + 0.213([-9.1 \quad 6.5 \quad 3.3 \quad -0.1] - [-9.1 \quad 6.5 \quad 3.3 \quad -0.1]\ )$
$+0.879([0.6 \quad 2.1 \quad 8.3 \quad -1.2] - [-9.1 \quad 6.5 \quad 3.3 \quad -0.1]) = [10.35 \quad 1.57 \quad 8.78 \quad -2.21]$
${x_0}^{new} = [-5.1 \quad 0.5 \quad 4.2 \quad 2.3] + [5.01 \quad 1.41 \quad 3.6 \quad 3.08] = [-0.09 \quad 1.91 \quad 7.8 \quad 5.28]$,
$x_1 = [3.64 \quad 9.27 \quad 15.63 \quad -3.24]$, $x_2 = [2.06 \quad 1.67 \quad 8.73 \quad 0.36]$, $x_3 = [-9.19 \quad 8.41 \quad 11.1 \quad 5.18]$

Updating weights:

$$\delta W = \sum_{t=0}^{T} \delta gates_t \otimes x_t \ = \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} \begin{bmatrix} -0.09 & 1.91 & 7.8 & 5.28 \end{bmatrix} + \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} \begin{bmatrix} 3.64 & 9.27 & 15.63 & -3.24 \end{bmatrix}$$
$$+ \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} \begin{bmatrix} 2.06 & 1.67 & 8.73 & 0.36 \end{bmatrix} + \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} \begin{bmatrix} -9.19 & 8.41 & 11.1 & 5.18 \end{bmatrix}$$
$$= \begin{bmatrix} 0.02672 & 0.0922 & 0.714 & 0.0756 \\ 0.00221 & 0.00666 & 0.0145 & 0.1244 \\ 0.00316 & 0.01893 & 0.0161 & 0.1037 \\ 0.02593 & 0.16262 & 0.3102 & 0.09231 \end{bmatrix}$$

$$\delta U = \sum_{t=0}^{T} \delta gates_{t+1} \otimes out_t \ = \begin{bmatrix} -0.01039 \\ -0.0006 \\ -0.00338 \\ -0.02970 \end{bmatrix}$$

$$\delta b = \sum_{t=0}^{T} \delta gates_{t+1} \ = \begin{bmatrix} -0.03641 \\ -0.00277 \\ -0.00631 \\ -0.05362 \end{bmatrix}$$

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.neucom.2018.07.097.

References

[1] J. Heaton, Artificial Intelligence for Humans, Volume 3: Neural Networks and Deep Learning, 3, Heaton Research, Inc., St. Louis, MO, USA, 2015.
[2] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machine," arXiv:1410.5401v2 [cs.NE], 2014.
[3] A. Baddeley, S. Della Sala, T. Robbins, A. Baddeley, Working memory and executive control [and discussion], Philosoph. Trans. R. Soc. B Biol. Sci. 351 (1996) 1397–1404.
[4] Z.C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," arXiv:1506.00019v4 [cs.LG], 2015.
[5] J. Weston, A. Bordes, S. Chopra, A.M. Rush, B. van Merriënboer, A. Joulin, et al., "Towards ai-complete question answering: a set of prerequisite toy tasks," arXiv:1502.05698v10 [cs.AI], 2015.
[6] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016) 471–476.
[7] G. Yang, "Lie access neural Turing machine," arXiv:1602.08671v3 [cs.NE], 2016.
[8] W. Zaremba and I. Sutskever, "Reinforcement learning neural turing machines – revised," arXiv:1505.00521v3 [cs.LG], 2015.
[9] C. Gulcehre, S. Chandar, K. Cho, Y. Bengio, Dynamic neural turing machine with continuous and discrete addressing schemes, Neural Comput. 30 (4) (2018) 857–884.
[10] R.B. Greve, E.J. Jacobsen, S. Risi, Evolving neural turing machines for reward-based learning, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Colorado, USA, Denver, 2016, pp. 117–124.
[11] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2002) 99–127.
[12] G. Stein, A.J. Gonzalez, C. Barham, Machines that learn and teach seamlessly, IEEE Trans. Learn. Technol. 6 (2013) 389–402.
[13] J. Zhao, G. Peng, NEAT versus PSO for evolving autonomous multi-agents coordination on pursuit-evasion problem, Informatics in Control, Automation and Robotics, 133, Springer, Berlin, Heidelberg, 2011, pp. 711–717.
[14] P. Verbancsics and J. Harguess, "Generative neuroevolution for deep learning," arXiv:1312.5355v1 [cs.NE], 2013.
[15] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, B. Scholkopf, Support vector machines, IEEE Intell. Syst. Appl. 13 (1998) 18–28.
[16] A. Jahangirzadeh, S. Shamshirband, S. Aghabozorgi, S. Akib, H. Basser, N.B. Anuar, et al., A cooperative expert based support vector regression (Co-ESVR) system to determine collar dimensions around bridge pier, Neurocomputing 140 (2014) 172–184.
[17] D.T. Larose, Discovering Knowledge in Data: An Introduction to Data Mining, John Wiley & Sons, Inc., 2005, pp. 90–106. ISBN 0-471-66657-2.
[18] I. Rish, An empirical study of the naive Bayes classifier, in: Proceedings of the IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, 2001, pp. 41–46.
[19] S.R. Safavian, D. Landgrebe, A survey of decision tree classifier methodology, IEEE Trans. Syst. Man Cybern. 21 (1991) 660–674.
[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[21] Y. LeCun, C. Cortes, and C.J. Burges, "MNIST handwritten digit database (2010)," URL http://yann.lecun.com/exdb/mnist.
[22] A. Gani, A. Siddiqa, S. Shamshirband, F. Hanum, A survey on indexing techniques for big data: taxonomy and performance evaluation, Knowl. Inf. Syst. 46 (2016) 241–284.
[23] J. Kennedy, R. Eberhart, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Japan, Japan, Nagoya, 1995.
[24] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: *Proceedings of the International Conference on Evolutionary Computation Proceedings* Anchorage, AK, USA, USA, 1998, pp. 69–73.
[25] Y. Zhang, S. Wang, G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Math. Probl. Eng. 2015 (2015) 38 931256, doi:10.1155/2015/931256.
[26] T. Talaśka, R. Długosz, W. Pedrycz, Hardware implementation of the particle swarm optimization algorithm, in: Proceedings of the International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES), Bydgoszcz, Poland, 2017, pp. 521–526.
[27] J. Kennedy, Particle swarm optimization, Encyclopedia of Machine Learning, Springer, Boston, MA, 2011, pp. 760–766.
[28] S. Rana, S. Jasola, R. Kumar, A review on particle swarm optimization algorithms and their applications to data clustering, Artif. Intell. Rev. 35 (2011) 211–222.
[29] Z. Liu, H. Cao, X. Chen, Z. He, Z. Shen , Multi-fault classification based on wavelet SVM with PSO algorithm to analyze vibration signals from rolling element bearings, Neurocomputing 99 (2013) 399–410.
[30] J. Anuradha, B.K. Tripathy, Uncertainty based hybrid particle swarm optimization techniques and their applications, in: S. Dehuri, A.K. Jagadev, M. Panda (Eds.), Multi-objective Swarm Intelligence, in: S. Dehuri, A.K. Jagadev, M. Panda (Eds.), Studies in Computational Intelligence, 592, Springer, Berlin, Heidelberg, 2015, pp. 143–169.
[31] F. Afifi, N.B. Anuar, S. Shamshirband, K.-K.R. Choo, DyHAP: dynamic hybrid ANFIS-PSO approach for predicting mobile malware, PloS One 11 (2016) e0162627.
[32] A. Kalantari, A. Kamsin, S. Shamshirband, A. Gani, H. Alinejad-Rokny, A.T. Chronopoulos, Computational intelligence approaches for classification of medical data: state-of-the-art, future challenges and research directions, Neurocomputing (2017).
[33] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (2009) 1–127.
[34] L. Deng, D. Yu, Deep learning: methods and applications, Found. Trends Signal Process. 7 (2014) 197–387.
[35] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11 (2010) 625–660.
[36] J. Weston, S. Chopra, and A. Bordes, "Memory networks," arXiv:1410.3916v11 [cs.AI], 2014.
[37] S. Sukhbaatar, J. Weston, R. Fergus, End-to-end memory networks, Adv. Neural Inf. Process. Syst. (2015) 2440–2448.
[38] P. Angelov, A. Sperduti, Challenges in deep learning, in: Proceedings of the European Symposium on Artificial NNs, Bruges (Belgium), 2016.
[39] F.A. Gers, N.N. Schraudolph, J. Schmidhuber, Learning precise timing with LSTM recurrent networks, J. Mach. Learn. Res. 3 (August 2002) 115–143.
[40] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, Y. Shi, Spoken language understanding using long short-term memory neural networks, in: Proceedings of the *Spoken Language Technology Workshop (SLT)*, South Lake Tahoe, NV, USA, 2014, pp. 189–194.
[41] F. Eyben, M. Wöllmer, B. Schuller, A. Graves, From speech to letters-using a novel neural network architecture for grapheme based asr, in: Proceedings of the *IEEE Workshop on Automatic Speech Recognition & Understanding* Merano, Italy, 2009, pp. 376–380.
[42] A. Graves, N. Jaitly, A.-r. Mohamed, Hybrid speech recognition with deep bidirectional LSTM, in: Proceedings of the *Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Olomouc, Czec h Republic, 2013, pp. 273–278.
[43] A. Graves, A.-R. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: Proceedings of the *Conference on Acoustics, Speech and Signal Processing (ICASSP)*, BC, Canada, Vancouver, 2013, pp. 6645–6649.
[44] H. Sak, A. Senior, F. Beaufays, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, in: Proceedings of the ISCA (International Speech Communication Association), Singapore, 2014.
[45] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[46] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, The Journal of Machine Learning Research 17 (1) (2016) 1334–1373.
[47] D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: from architectures to learning, Evol. Intell. 1 (2008) 47–62.
[48] D. Floreano, J. Urzelai, Evolutionary robots with on-line self-organization and behavioral fitness, Neural Netw. 13 (2000) 431–443.
[49] J. Blynel, D. Floreano, Exploring the T-maze: evolving learning-like robot behaviors using CTRNNs, in: Proceedings of the EvoWorkshops, UK, Verlag Berlin Heidelberg, 2003, pp. 593–604.
[50] K.O. Stanley, B.D. Bryant, R. Miikkulainen, Evolving adaptive neural networks with and without adaptive synapses, in: Proceedings of the Congress on Evolutionary Computation, Australia, Australia, Canberra, ACT, 2003, pp. 2557–2564.
[51] S. Risi, K.O. Stanley, Indirectly encoding neural plasticity as a pattern of local rules, Proceedings of the 11th international conference on Simulation of adaptive behavior: from animals to animats, Paris, France, Pages, Springer-Verlag, 2010, pp. 533–543.
[52] K.O. Ellefsen, J.-B. Mouret, J. Clune, Neural modularity helps organisms evolve to learn new skills without forgetting old skills, PLoS Comput. Biol. 11 (2015) 28.
[53] D.O. Hebb, The Organization of Behavior: A neuropsychological Theory, Psychology Press, London, 2005.
[54] J.B. Pollack, On Connectionist Models of Natural Language Processing, Ph.D. Computer Science Dept, University of Illinois, Urbana, 1987 Available as MCCS-87-100, Computing Resesrch Labratory, NMSU, Las Cruces, NM.
[55] H.T. Siegelmann, E.D. Sontag, Turing computability with neural nets, Appl. Math. Lett. 4 (1991) 77–80.
[56] A.M. Turing, Computing Machinery and Intelligence, Mind, 59, JSTOR, 1950, pp. 433–460.
[57] J. v. Neumann, The Computer and Brain, Yale University Press New Haven, CT, USA, 1958.
[58] C. Gulcehre, S. Chandar, K. Cho, Y. Bengio., Dynamic neural turing machine with continuous and discrete addressing schemes, Neural comput. 30 (4) (2018) 857–884.
[59] K. Kurach, M. Andrychowicz, I. Sutskever, Neural random-access machines, in: Proceedings of the ICLR, 2015.
[60] W. Zaremba, "Learning algorithms from data," Ph.D., Computer Science, New York University, 2016.
[61] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.

[62] A. Zylberberg, S. Dehaene, P.R. Roelfsema, M. Sigman, The human Turing machine: a neural framework for mental programs, Trends Cogn. Sci. 15 (7) (2011) 293–300, doi:10.1016/j.tics.2011.05.007.
[63] S.S. Haykin, Neural Networks and Learning Machines, 3, PearsonPrentice Hall, Upper Saddle River, NJ, USA, 2009.
[64] A. Baddeley, Working memory: looking back and looking forward, Nat. Rev. Neurosci. 4 (2003) 829–839.
[65] A. Baddeley, G. Hitch, R. Allen, Working memory and binding in sentence recall, J. Memory Lang. 61 (2009) 438–456.
[66] Y. Zhang, S. Wang, G. ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Math. Probl. Eng. 2015 (2015) 1–38, doi:10.1155/2015/931256.
[67] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (1990) 1550–1560.
[68] J. McCaffrey, Machine Learning U58sing C#, Syncfusion, 2014.
[69] T. Sousa, A. Silva, A. Neves, Particle swarm based data mining algorithms for classification tasks, Parall. Comput. 30 (2004) 767–783.
[70] B. Xue, M. Zhang, W.N. Browne, Particle swarm optimization for feature selection in classification: a multi-objective approach, IEEE Trans. Cybern. 43 (2013) 1656–1671.
[71] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, et al., Comparison of classifier methods: a case study in handwritten digit recognition, in: Proceedings of the Twelfth IAPR International Conference on Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing, 1994, pp. 77–82.
[72] S. Tabik, D. Peralta, A. Herrera, A snapshot of image pre-processing for convolutional neural networks: case study of mnist, Int. J. Comput. Intell. Syst. 10 (2017) 555–568.
[73] L. Deng, The MNIST database of handwritten digit images for machine learning research [best of the web], IEEE Signal Process. Mag. 29 (2012) 141–142.
[74] R.S. Olson, J.H. Moore, C. Adami, Evolution of active categorical image classification via saccadic eye movement, in: Proceedings of the International Conference on Parallel Problem Solving from Nature, Cham, 2016, pp. 581–590.

Soroor Malekmohamadi Faradonbe received her B.E. degree (Software Engineering) in 2013 from Islamic Azad University, Mobarakeh Branch. She is currently MSc. student of Artificial Intelligence on faculty of Computer Engineering, Islamic Azad University, Najafabad Branch, Iran. Her research interests are intelligent computing, Machine Learning and Neural Networks, Autonomic Computing, and Bio-inspired Computing.

Faramarz Safi-Esfahani received his Ph.D. in Intelligent Computing from University of Putra Malaysia in 2011. He is currently on faculty of Computer Engineering, Islamic Azad University, Najafabad Branch, Iran. His research interests are intelligent computing, Big Data and Cloud Computing, Autonomic Computing, and Bio-inspired Computing.