Supervised Learning of Neural Random-Access Machines with Differential Evolution

Candidate

Valerio Belli

Supervisors

Valentina Poggioni, Marco Baioletti

April 19, 2018

University of Perugia

Departments of Mathematics and Computer Science

Intelligent and Mobile Computing

Introduction

The success of Deep Learning is undeniable.

A new family of models, based on **controller-interface abstraction**, has been introduced. The precursor is **Neural Turing Machine** (**NTM**) [Graves et al., 2014] which works with **attentional "focus" mechanisms** to interact with an external memory.

The **Neural Random-Access Machines** (NRAM) [Kurach et al., 2015] evolve the NTM implementing the concepts of **pointers manipulation and de-referencing** through primitive operations to interact also with a memory.

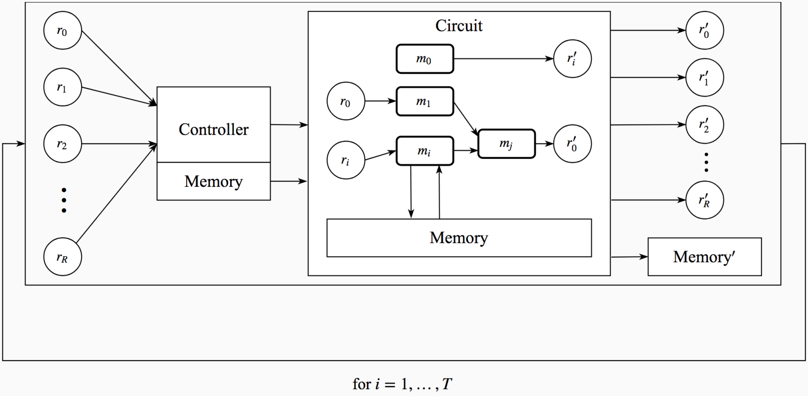
We implemented and trained **NRAM** in order to study the benefits of Differential Evolution on these type of models.

Outline

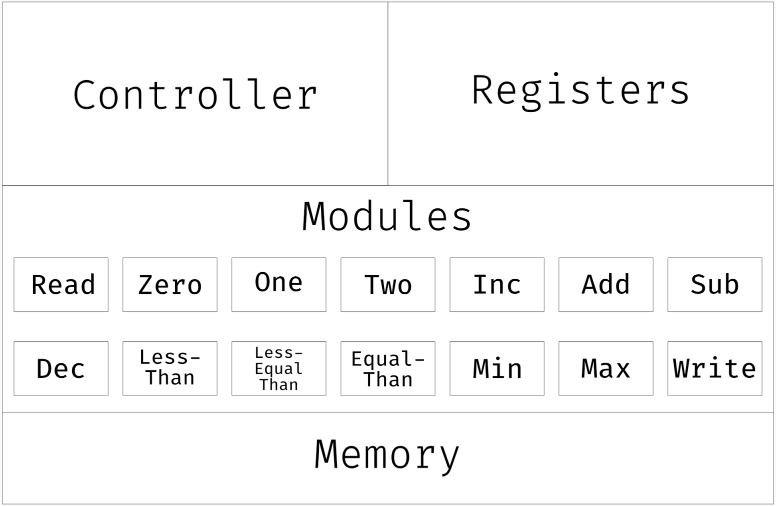
1. Neural Random-Access Machines
2. Neural Network optimization & DENN
3. Implementation & Faced problems
4. Results, Conclusions & Future works

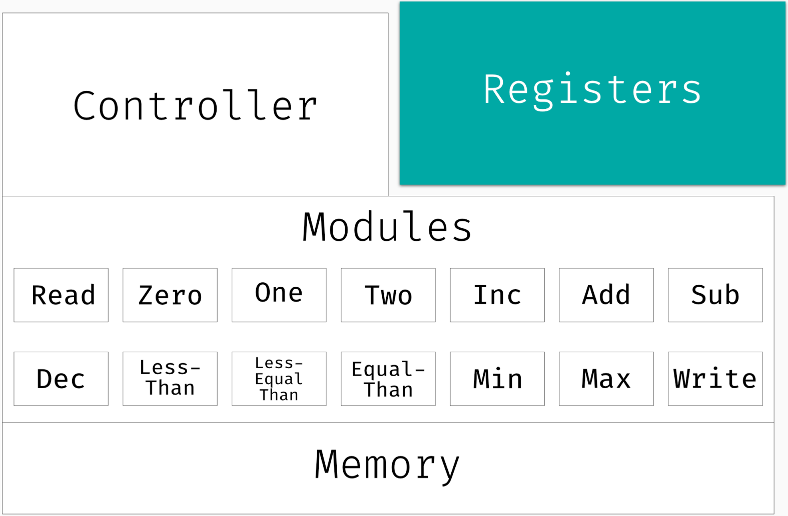
Neural Random-Access Machines

High view of the Neural Random-Access Machines model.

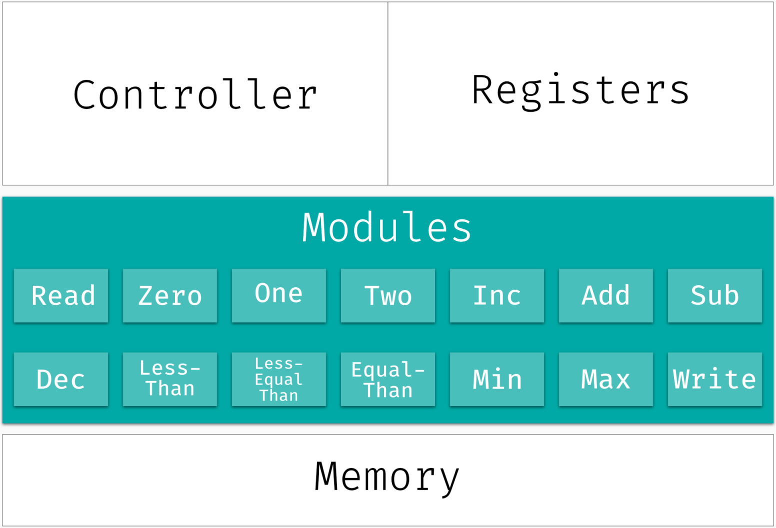


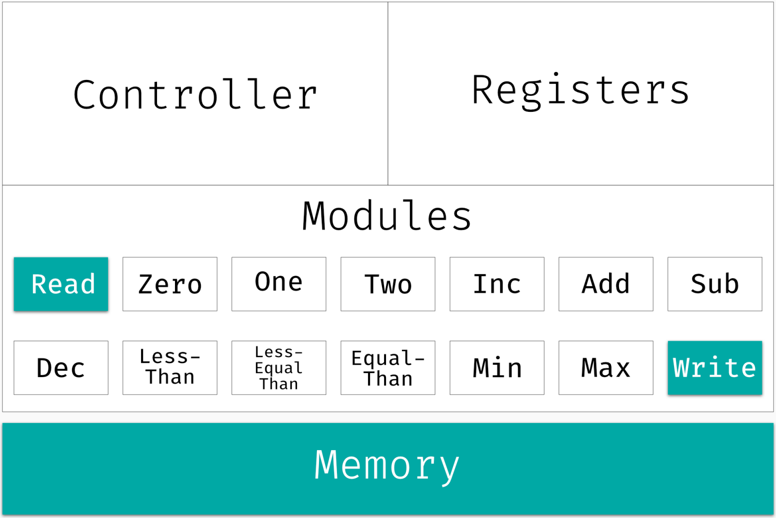
NRAM → Components




Controller
Registers
Modules
Read
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal Than
Equal-Than
Min
Max
Write
Memory

The registers $\mathcal{R} \in \mathbb{R}^{R \times M}$ is a set of R memory cells. Each register contains a probability distribution in $\mathbb{R}^M$ over the set $\{0, \ldots, M-1\}$.

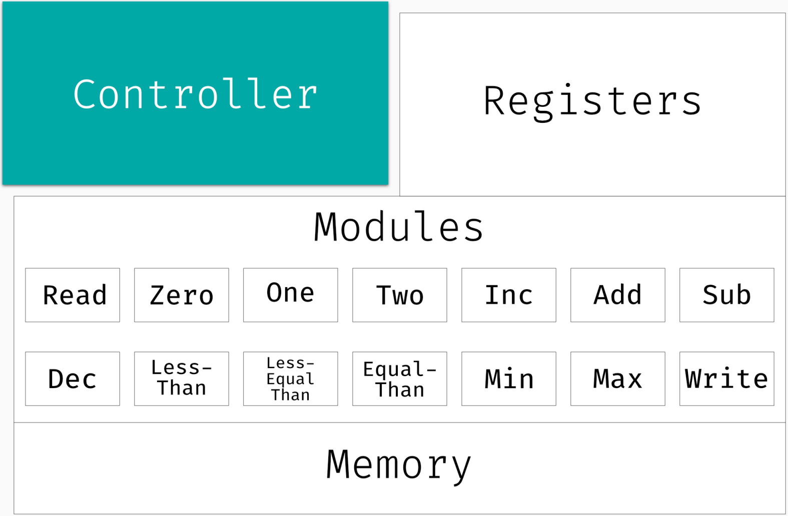

Controller
Registers
Modules
Read
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal Than
Equal-Than
Min
Max
Write
Memory


Controller
Registers
Modules
Read
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal Than
Equal-Than
Min
Max
Write
Memory

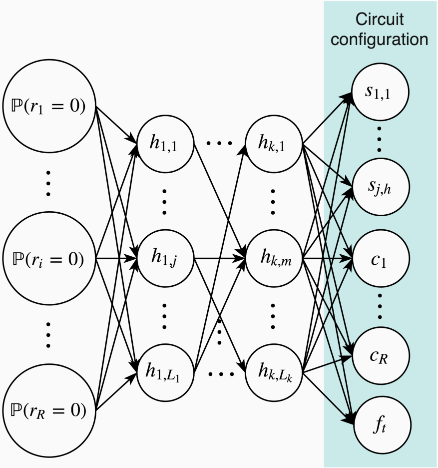
The memory $\mathcal{M} \in \mathbb{R}^{M \times M}$ is a support of M cells, where each value is represented by a probability distribution in $\mathbb{R}^M$ over the set $\{0, \ldots, M-1\}$.

The NRAM interacts with the memory through:

- **Read**: takes a pointer and returns the pointed value of the memory.
- **Write**: takes a pointer and a value, modifies the memory.


Controller
Registers
Modules
Read
Zero
One
Two
Inc
Add
Sub
Dec
Less-Than
Less-Equal Than
Equal-Than
Min
Max
Write
Memory

- It is a neural network
- It takes as input $\mathbb{P}(r_i = 0)$, for $i = 1, \ldots, R$
- It emits configurations for fuzzy circuits (how the registers and modules are connected together)
- From an high view, each $s_{i,j}$ and c_h is a probability distribution generated with the **softmax** function



NRAM $\rightarrow$ Termination of NRAM

The NRAM terminates its execution in two ways:

- Reaching the **last timestep** T
- Through an **internal criterion**:
  - In each timestep, NRAM emits the willingness of terminate the execution $f_t = \sigma(x_i)$
  - The execution stops if $f_t = 1.0$.

Let $\mathcal{M} \in \mathbb{R}^{M \times M}$ the output memory and $\mathbf{y} \in \{0, \ldots, M-1\}^M$ the expected memory, the **cost function** is the **expected negative log-likelihood**

$$-\sum_{i=1}^{T} \left(p_t \cdot \sum_{i=1}^{M} log\left(\mathcal{M}_{i,\mathbf{y}_i}^{(t)} \right) \right)$$

where p_t is computed as

$$p_t = f_t \cdot \prod_{i=1}^{t-1} (1 - f_i)$$

Neural Network optimization & DENN

NN optimization & DENN → Gradient Descent & back-propagation

Gradient Descent and back-propagation optimization is divided in **computing of the gradient** and **updating of the network parameters**.

We used **ADAM** (Adaptive Moment estimation) [Kingma and Ba, 2014] as in [Kurach et al., 2015] as gradient-based optimization algorithm. Uses the concept of **momentum** to regularize the descent of the gradient.

Differential Evolution

- Meta-heuristic
- Searches a solution through the parallel evolution of a set of candidate solutions
- Candidate solutions set called **population**
  - Composed by **N** D-dimensional numerical vectors, called **Individuals**

Its functioning is iterative

1. **Mutation**: driven by a constant *F* creates a new population called **donor set** (several methods, e.g. **DEGL** and **Current-to-pbest**)
2. **Crossover**: driven by a constant *CR* creates a new population called **trial set** (several methods, e.g. **bin**)
3. **Selection**: generates the new population for the next generation comparing one-by-one the trial vectors with the corresponding target vectors.

Exist various self-adaptive variants of Differential Evolution which alleviate the problem dependence of *F* and *CR*:

- JADE
- SHADE
- L-SHADE

NN optimization & DENN → DENN (Differential Evolution for Neural Network)

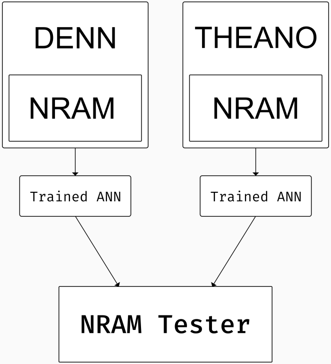
DENN [Baioletti et al., 2018] is a **framework** which implements the concepts of the Differential Evolution to train ANN:

- Created by Gabriele Di Bari and Mirco Tracolli
- It is written in C++ based on the library Eigen
- Each individual is composed by the weights and biases of the network
- The mutation and crossover operators are applied in a component-wise way

Implementation & Faced problems

Three applications

- **NRAM-DENN**: training with Differential Evolution
- **NRAM-Theano**: training with Gradient Descent-ADAM
- **NRAM-Tester**: execution and generalization testing



Implementation & Faced problems → Additional used techniques

NRAM-Theano:

- **Gradient clipping**: gradient clipped in $[C_1, C_2]$ to avoid the common problem of the Gradient Explosion;
- **Noise**: added a noise to the computed gradient to enhance the exploration;

NRAM-DENN:

- **Curriculum Learning**: it is used to enhance the training of the ANN through increasing difficulties of the datasets.

Tested problems:

- **Access**: accessing of a value of an input sequence
- **Increment**: incrementing by one of a input sequence
- **Copy**: copying an input sequence to a part of the memory
- **Reverse**: copying an input sequence in a reverse order to a part of the memory

Implementation & Faced problems → Datasets

The datasets are generated at runtime; each batch of samples is composed by:

- **initial memory content**
- **expected memory content**
- **cost mask**
- **error rate mask**

Initial memory content								
5	5	1	4	7	<u>0</u>	0	0	0

Expected memory content								
5	5	1	4	7	7	4	1	5

Cost mask								
0	1	1	1	1	1	1	1	1

Error mask								
0	0	0	0	0	1	1	1	1

Table 1: Reverse task.

Results, Conclusions & Future works

Task	Train complexity		Reached cost 0		Train error		Generalization	
	No CL	CL	No CL	CL	No CL	CL	No CL	CL
Access	—	$\mathrm{len}(A) \leq 20$	×	✓	—	0	×	Perfect
Increment	—	$\mathrm{len}(A) \leq 15$	×	✓	—	0	×	Perfect
Copy	—	$\mathrm{len}(A) \leq 15$	×	✓	—	0	×	Perfect
Reverse	—	$\mathrm{len}(A) \leq 15$	×	✓	—	0	×	Perfect

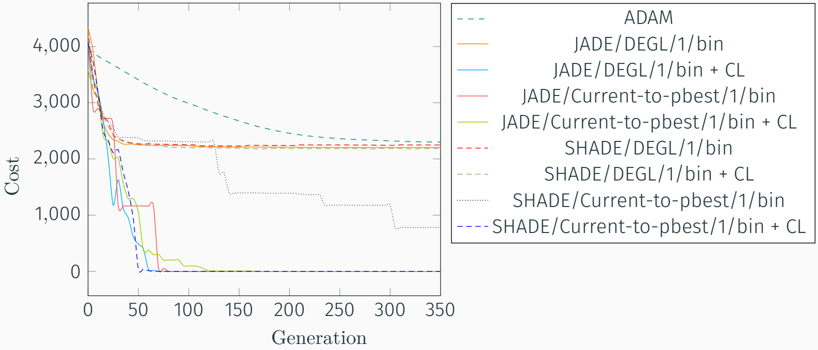
Table 2: Results of the tests with ADAM in [Kurach et al., 2015].

Task	Train complexity		Reached cost 0		Train error		Generalization	
	No CL	CL	No CL	CL	No CL	CL	No CL	CL
Access	$\mathrm{len}(A) = 8$ $t = 5$	$\mathrm{len}(A) \leq 10$	✓	✓	0	0	Perfect	Perfect
Increment	$\mathrm{len}(A) = 9$ $t = 4$	$\mathrm{len}(A) \leq 10$	✓	✓	0	0	Perfect	Perfect
Copy	$\mathrm{len}(A) = 5$ $t = 11$	$\mathrm{len}(A) \leq 9$	×	×	—	—	—	—
Reverse	$\mathrm{len}(A) = 4$ $t = 9$	$\mathrm{len}(A) \leq 8$	✓	✓	0	0	Perfect	Perfect

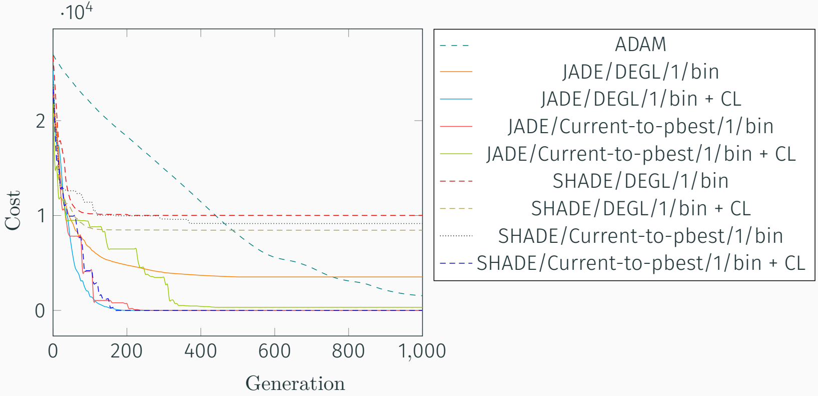
Table 3: Results of the tests with Differential Evolution.

Task	JADE/DEGL/1		JADE/c-to-pb/1		SHADE/DEGL/1		SHADE/c-to-pb/1	
	No CL	CL	No CL	CL	No CL	CL	No CL	CL
Access	×	✓	✓	✓	×	×	×	✓
Increment	×	×	✓	×	×	×	×	✓
Copy	×	×	×	×	×	×	×	×
Reverse	×	✓	✓	×	✓	✓	×	✓

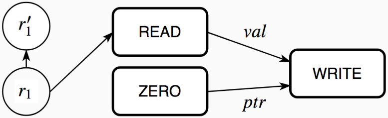
Table 4: Results of DE variants



4,000
3,000
2,000
1,000
0
Cost
0
50
100
150
200
250
300
350
Generation
ADAM
JADE/DEGL/1/bin
JADE/DEGL/1/bin + CL
JADE/Current-to-pbest/1/bin
JADE/Current-to-pbest/1/bin + CL
SHADE/DEGL/1/bin
SHADE/DEGL/1/bin + CL
SHADE/Current-to-pbest/1/bin
SHADE/Current-to-pbest/1/bin + CL

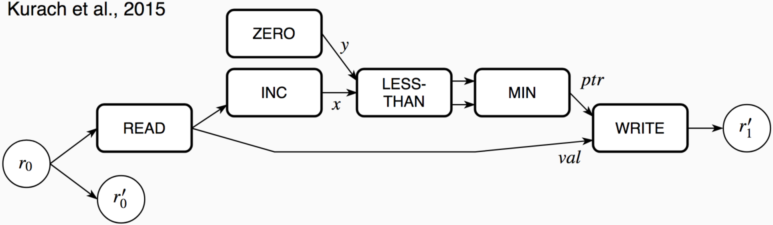

$\cdot 10^4$
2
1
0
Cost
0
200
400
600
800
1,000
Generation
ADAM
JADE/DEGL/1/bin
JADE/DEGL/1/bin + CL
JADE/Current-to-pbest/1/bin
JADE/Current-to-pbest/1/bin + CL
SHADE/DEGL/1/bin
SHADE/DEGL/1/bin + CL
SHADE/Current-to-pbest/1/bin
SHADE/Current-to-pbest/1/bin + CL

DENN with Curriculum Learning

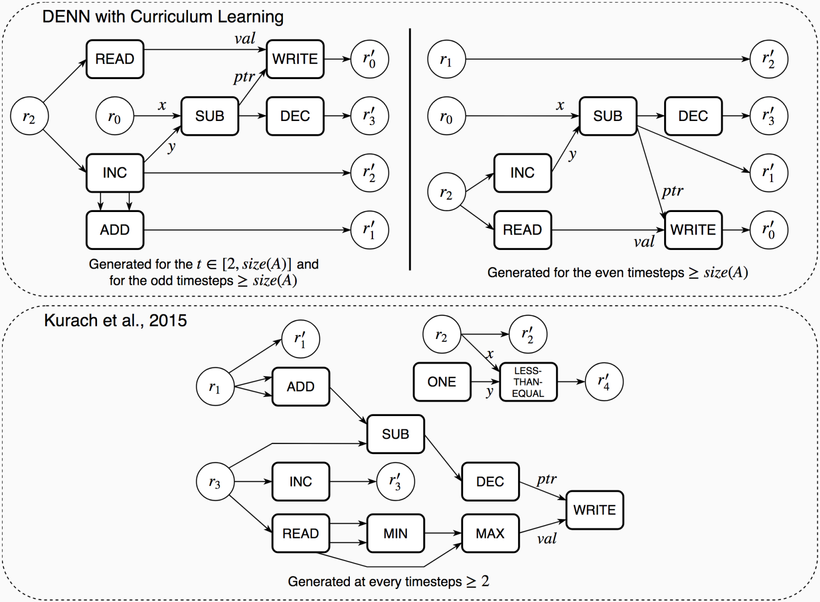


Generated for every timesteps ≥ 2.

Kurach et al., 2015



Generated for every timesteps ≥ 2.


DENN with Curriculum Learning
READ
val
WRITE
r'_0
ptr
r_2
r_0
x
SUB
DEC
r'_3
y
INC
r'_2
ADD
r'_1
Generated for the $t \in [2, size(A)]$ and
for the odd timesteps $\geq size(A)$
r_1
r'_2
r_0
x
SUB
DEC
r'_3
INC
y
r'_1
r_2
ptr
READ
val
WRITE
r'_0
Generated for the even timesteps $\geq size(A)$
Kurach et al., 2015
r'_1
r_1
ADD
r_2
r'_2
x
ONE
y
LESS-THAN-EQUAL
r'_4
SUB
r_3
INC
r'_3
DEC
ptr
WRITE
READ
MIN
MAX
val
Generated at every timesteps ≥ 2

DENN behave well in this type of model:

- Differential Evolution behave differently with these problems w.r.t. ADAM producing different results
- Best performing variants are those with **Current-to-pbest**
- Found controllers generate simpler circuits w.r.t. in [Kurach et al., 2015]

Create a new dynamic system for the gate selection.

Write an enhanced NRAM model that does not require the differentiability condition.

Thanks for the attention!