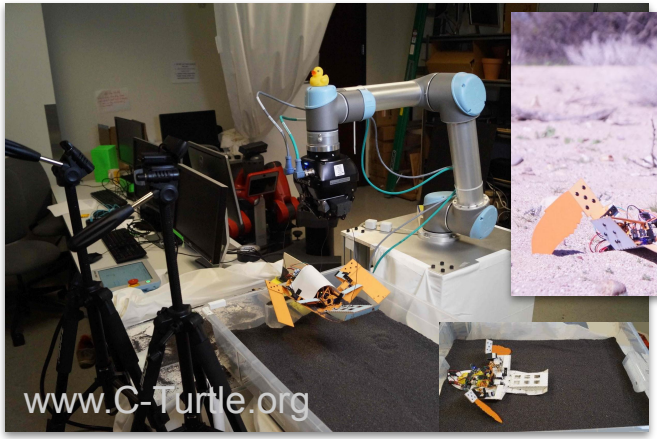
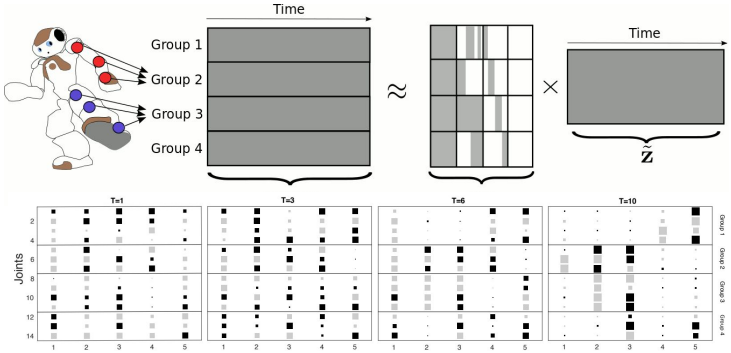
An Architectural View On Differentiable Neural Computers

Kevin Sebastian Luck (ksluck@)



Bio-inspired Robot Design Considering Load-bearing and Kinematic Ontogeny of Chelonioidea Sea Turtles
A. Jansen, K. S. Luck, J. Campbell, H. Ben Amor & D. M. Aukes
Living Machines 2017 Proceedings

From the Lab to the Desert: Fast Prototyping and Learning of Robot Locomotion
K. S. Luck, J. Campbell, A. Jansen, D. M. Aukes & H. Ben Amor
Robotics:Science&Systems 2017 Proceedings

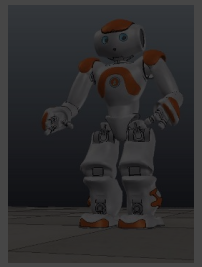


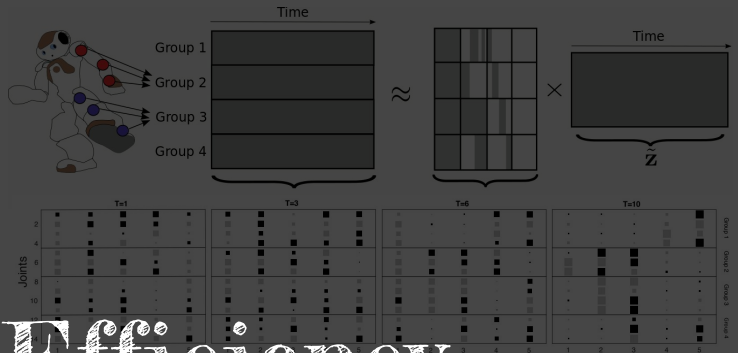
Latent space policy search for robotics
K. S. Luck, G. Neumann, E. Berger, J. Peters & H. B. Amor
IROS 2014 Proceedings

Sparse Latent Space Policy Search
K. S. Luck, J. Pajarinen, E. Berger, V. Kyrki & H. B. Amor
AAAI 2016 Proceedings

Extracting Bimanual Synergies with Reinforcement Learning
Kevin Sebastian Luck and Heni Ben Amor
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017) Proceedings

Interactive Robotics Lab





Real World Sample Efficiency for Reinforcement Learning Through Directed Exploration

Bio-inspired Robot Design Considering Load Bearing Capacity Ontogeny of Chelonioidea Sea Turtles
A. Jansen, K. S. Luck, J. Campbell, H. Ben Amor & D. M. Aukes
Living Machines 2017 Proceedings

From the Lab to the Desert: Fast Prototyping and Learning of Robot Locomotion
K. S. Luck, J. Campbell, A. Jansen, D. M. Aukes & H. Ben Amor
Robotics:Science&Systems 2017 Proceedings

Latent space policy search for robotics
K. S. Luck, G. Neumann, E. Berger, J. Peters & H. B. Amor
IROS 2014 Proceedings

Sparse Latent Space Policy Search
K. S. Luck, J. Pajarinen, E. Berger, V. Kyrki & H. B. Amor
AAAI 2016 Proceedings

Extracting Bimanual Synergies with Reinforcement Learning
Kevin Sebastian Luck and Heni Ben Amor
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017) Proceedings

Interactive Robotics Lab

An Architectural View On Differentiable Neural Computers

Kevin Sebastian Luck (ksluck@)

ARTICLE

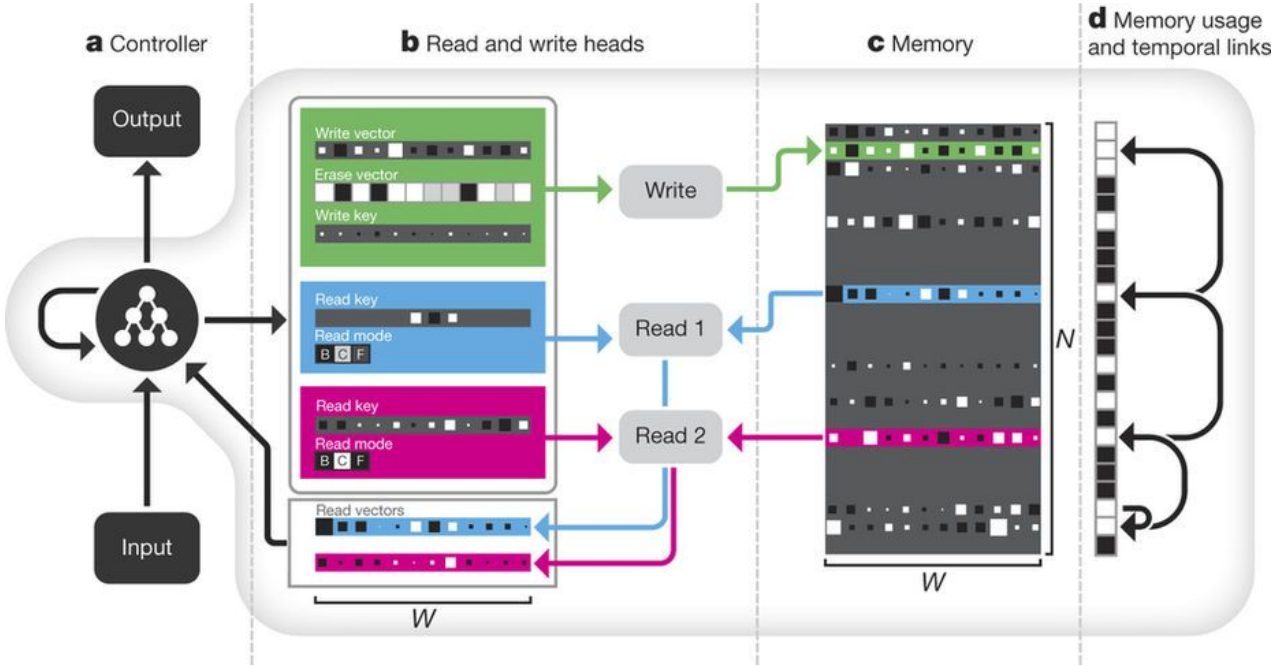
doi:10.1038/nature20101

Hybrid computing using a neural network with dynamic external memory

Alex Graves[1]*, Greg Wayne[1]*, Malcolm Reynolds[1], Tim Harley[1], Ivo Danihelka[1], Agnieszka Grabska-Barwińska[1], Sergio Gómez Colmenarejo[1], Edward Grefenstette[1], Tiago Ramalho[1], John Agapiou[1], Adrià Puigdomènech Badia[1], Karl Moritz Hermann[1], Yori Zwols[1], Georg Ostrovski[1], Adam Cain[1], Helen King[1], Christopher Summerfield[1], Phil Blunsom[1], Koray Kavukcuoglu[1] & Demis Hassabis[1]

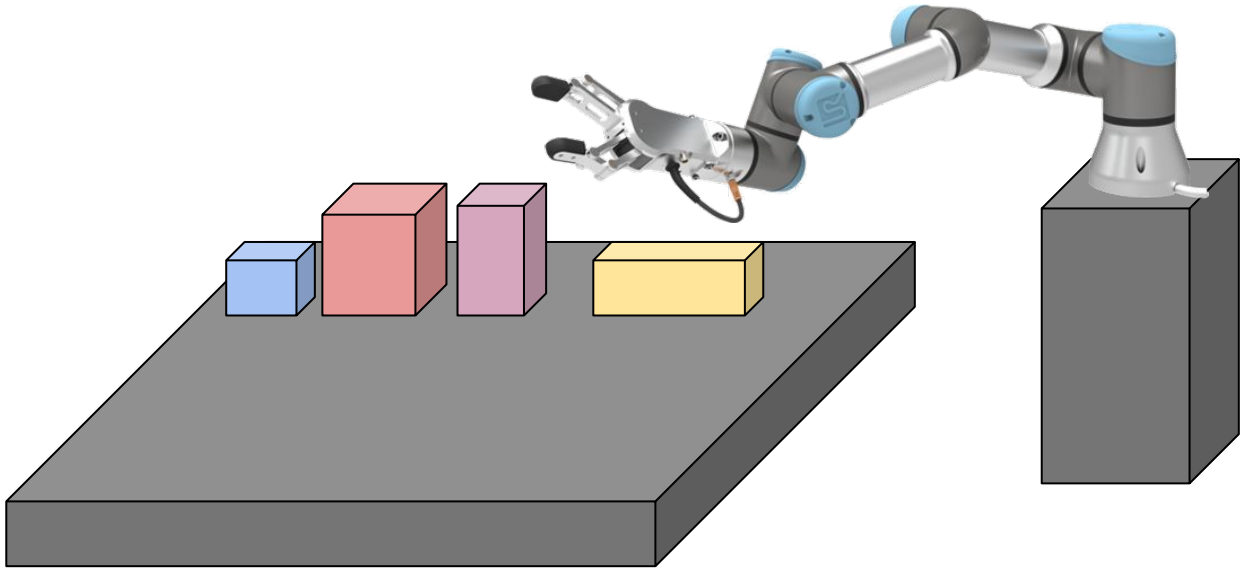
Artificial neural networks are remarkably adept at sensory processing, sequence learning and reinforcement learning, but are limited in their ability to represent variables and data structures and to store data over long timescales, owing to the lack of an external memory. Here we introduce a machine learning model called a differentiable neural computer (DNC), which consists of a neural network that can read from and write to an external memory matrix, analogous to the random-access memory in a conventional computer. Like a conventional computer, it can use its memory to represent and manipulate complex data structures, but, like a neural network, it can learn to do so from data. When trained with supervised learning, we demonstrate that a DNC can successfully answer synthetic questions designed to emulate reasoning and inference problems in natural language. We show that it can learn tasks such as finding the shortest path between specified points and inferring the missing links in randomly generated graphs, and then generalize these tasks to specific graphs such as transport networks and family trees. When trained with reinforcement learning, a DNC can complete a moving blocks puzzle in which changing goals are specified by sequences of symbols. Taken together, our results demonstrate that DNCs have the capacity to solve complex, structured tasks that are inaccessible to neural networks without external read–write memory.

DeepMind

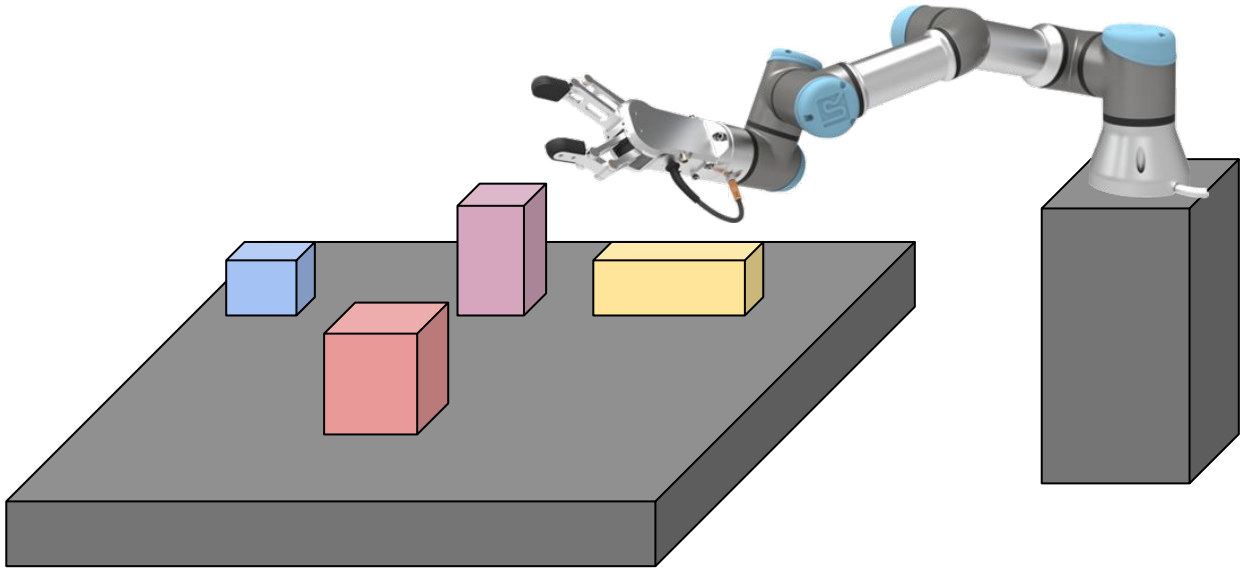

a Controller
b Read and write heads
c Memory
d Memory usage and temporal links
Output
Input
Write vector
Erase vector
Write key
Read key
Read mode
B C F
Read key
Read mode
B C F
Read vectors
W
Write
Read 1
Read 2
N
W

But why am I working on this... ?

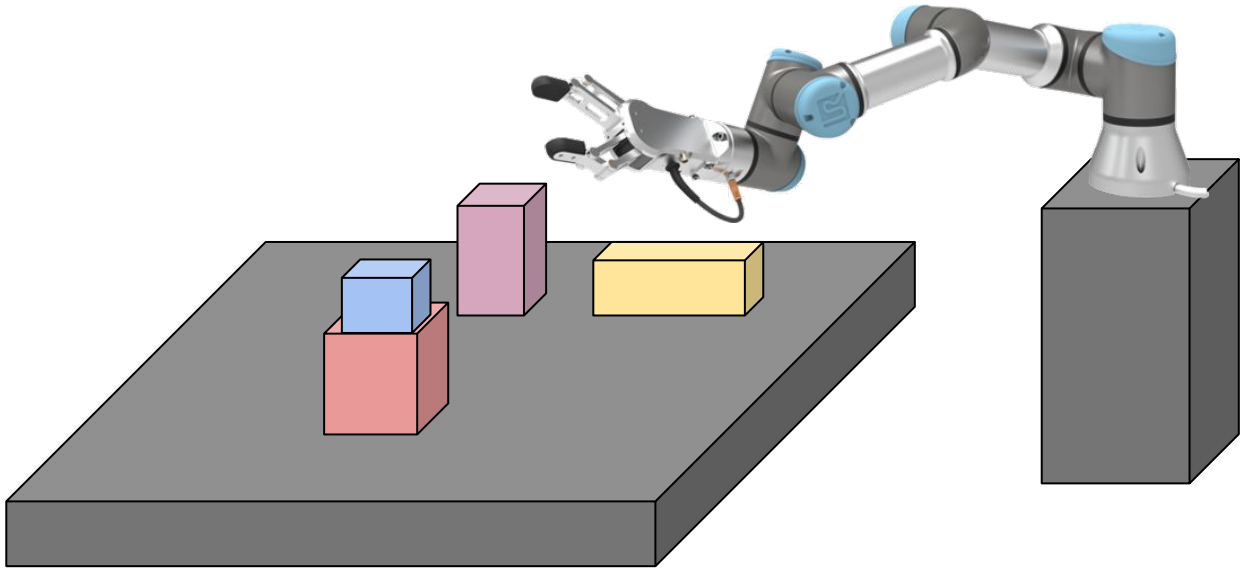
But why am I working on this... ?



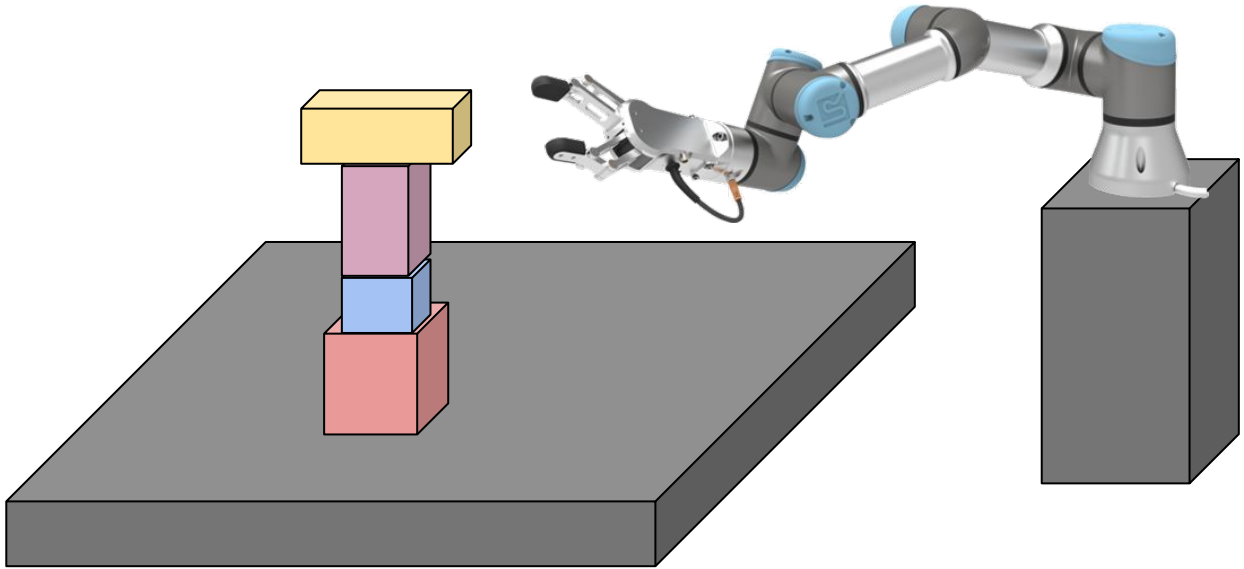
But why am I working on this... ?



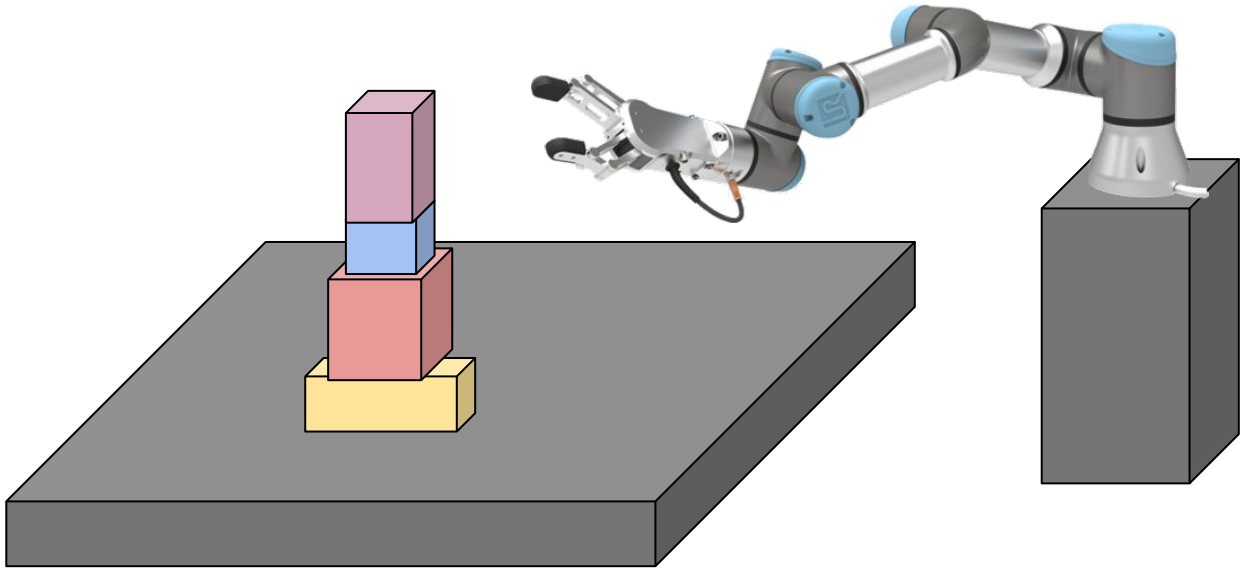
But why am I working on this... ?



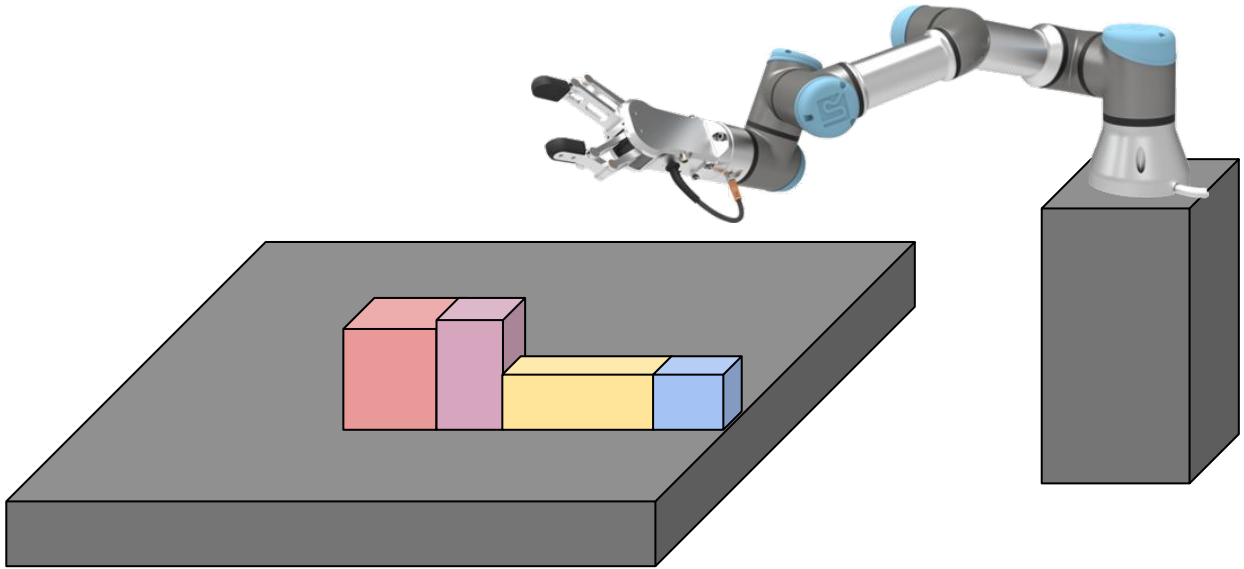
But why am I working on this... ?



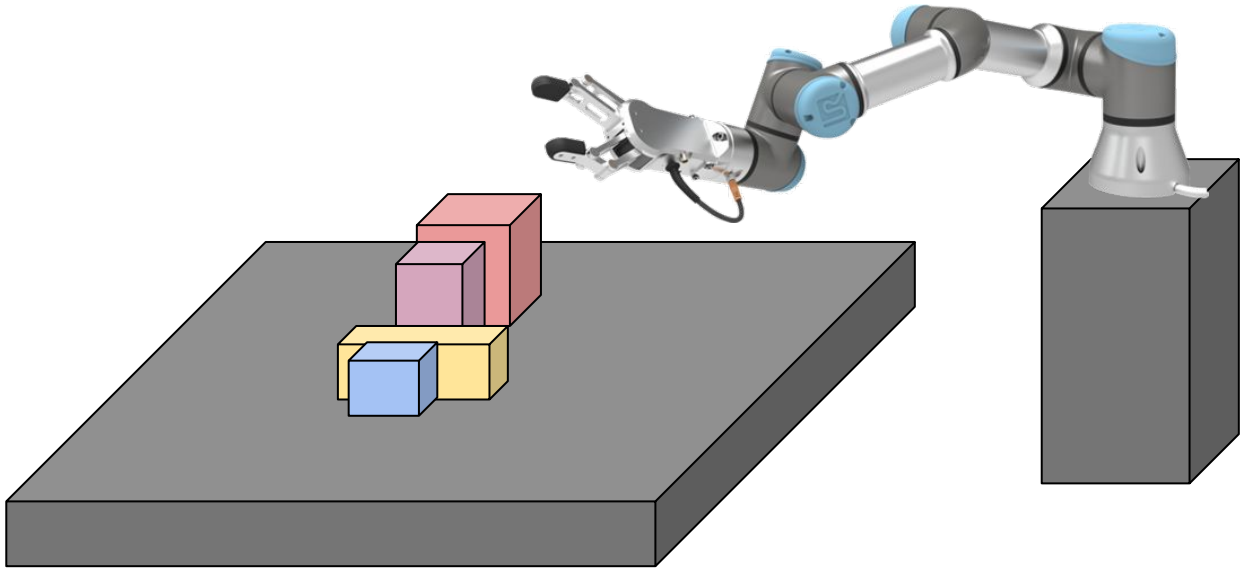
But why am I working on this... ?



But why am I working on this... ?



But why am I working on this... ?



Neural Turing Machines

Alex Graves gravesa@google.com
Greg Wayne gregwayne@google.com
Ivo Danihelka danihelka@google.com

Google DeepMind, London, UK

Abstract

We extend the capabilities of neural networks by coupling them to external memory resources, which they can interact with by attentional processes. The combined system is analogous to a Turing Machine or Von Neumann architecture but is differentiable end-to-end, allowing it to be efficiently trained with gradient descent. Preliminary results demonstrate that *Neural Turing Machines* can infer simple algorithms such as copying, sorting, and associative recall from input and output examples.

1 Introduction

Computer programs make use of three fundamental mechanisms: elementary operations (e.g., arithmetic operations), logical flow control (branching), and external memory, which can be written to and read from in the course of computation (Von Neumann, 1945). Despite its wide-ranging success in modelling complicated data, modern machine learning has largely neglected the use of logical flow control and external memory.

Recurrent neural networks (RNNs) stand out from other machine learning methods for their ability to learn and carry out complicated transformations of data over extended periods of time. Moreover, it is known that RNNs are Turing-Complete (Siegelmann and Sontag, 1995), and therefore have the capacity to simulate arbitrary procedures, *if* properly wired. Yet what is possible in principle is not always what is simple in practice. We therefore enrich the capabilities of standard recurrent networks to simplify the solution of algorithmic tasks. This enrichment is primarily via a large, addressable memory, so, by analogy to Turing's enrichment of finite-state machines by an infinite memory tape, we

Thus each DLA organ has now a number $\mu = 0, 1, \ldots, 255$ (or 8-digit binary), and each minor cycle in it has a number $\rho = 0, 1, \ldots, 31$ (or 5-digit binary). A minor cycle is completely defined within M by specifying both numbers μ, ρ. Due to these relationships we propose to call a DLA organ a *major cycle*.

Fourth: As the contents of a minor cycle make their transit across a DLA organ, i.e. a major cycle, the minor cycles number ρ clearly remains the same. When it reaches the output and is then cycled back into the input of a major cycle the number ρ is still not changed (since it will reach the output again after 1,024 periods τ, and we have synchronism in all DLA organs, and a 1,024 τ periodicity, cf. above), but μ changes to the number of the new major cycle. For individual cycling, the arrangement of Figure 19 (a), this means that μ, too, remains unchanged. For serial cycling, the arrangement of Figure 19 (b), this means that μ usually increases by 1, except that at the end of such a series of, say, s major cycles it decreases by $s - 1$.

These observations about the fate of a minor cycle after it has appeared at the output of its major cycle apply as such when that major cycle is undisturbed, i.e. when it is off in the sense of 13.2. When it is on, in the same sense, but in the first case of 13.3, then our observations are obviously still valid—i.e. they hold as long as the minor cycle is not being cleared. When it is being cleared, i.e. in the second case of 13.3, then those observations apply to the minor cycle which replaces the one that has been cleared.

15.0 THE CODE

15.1 The considerations of 14.0 provide the basis for a complete classification of the contents of M, i.e. they enumerate a system of successive disjunctions which give together this classification. This classification will put us into the position to formulate the code which effects the logical control of CC, and hence of the entire device.

Let us therefore restate the pertinent definitions and disjunctions.

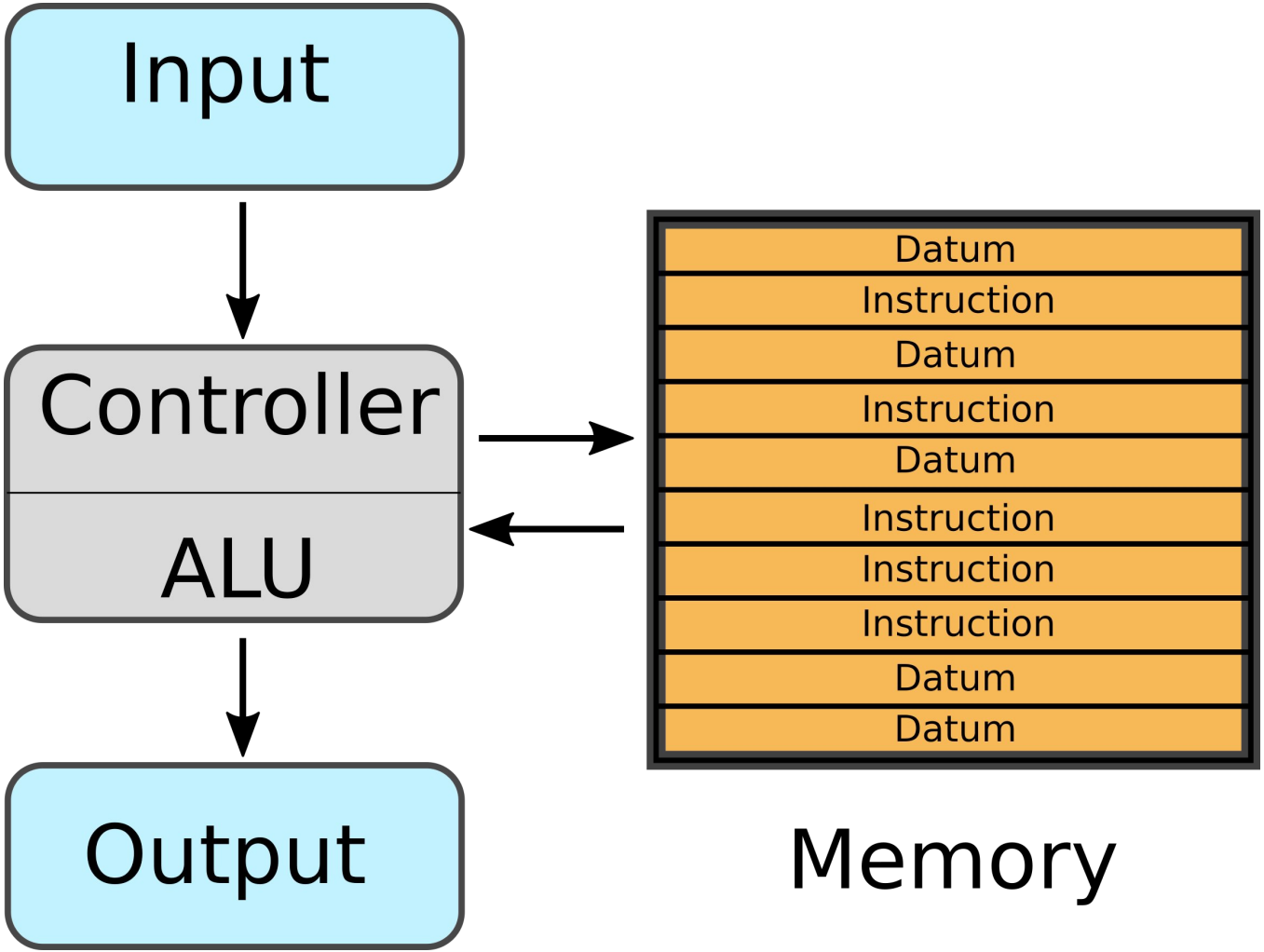
The contents of M are the memory units, each one being characterized by the presence or absence of a stimulus. It can be used to represent accordingly the binary digit 1 or 0, and we will at any rate designate its content by the binary digit $i = 1$ or 0 to which it corresponds in this manner. (cf. 12.2 and 12.5 with 7.6.) These units are grouped together to form 32-unit minor cycles, and these minor cycles are the entities which will acquire direct significance in the code which we will introduce. (cf. 12.2.) We denote the binary digits which make up the 32 units of a minor cycle, in their natural temporal sequence, by $i_0, i_1, i_2, \ldots, i_{31}$. The minor cycles with these units may be written $I = (i_0, i_1, i_2, \ldots, i_{31}) = (i_v)$.

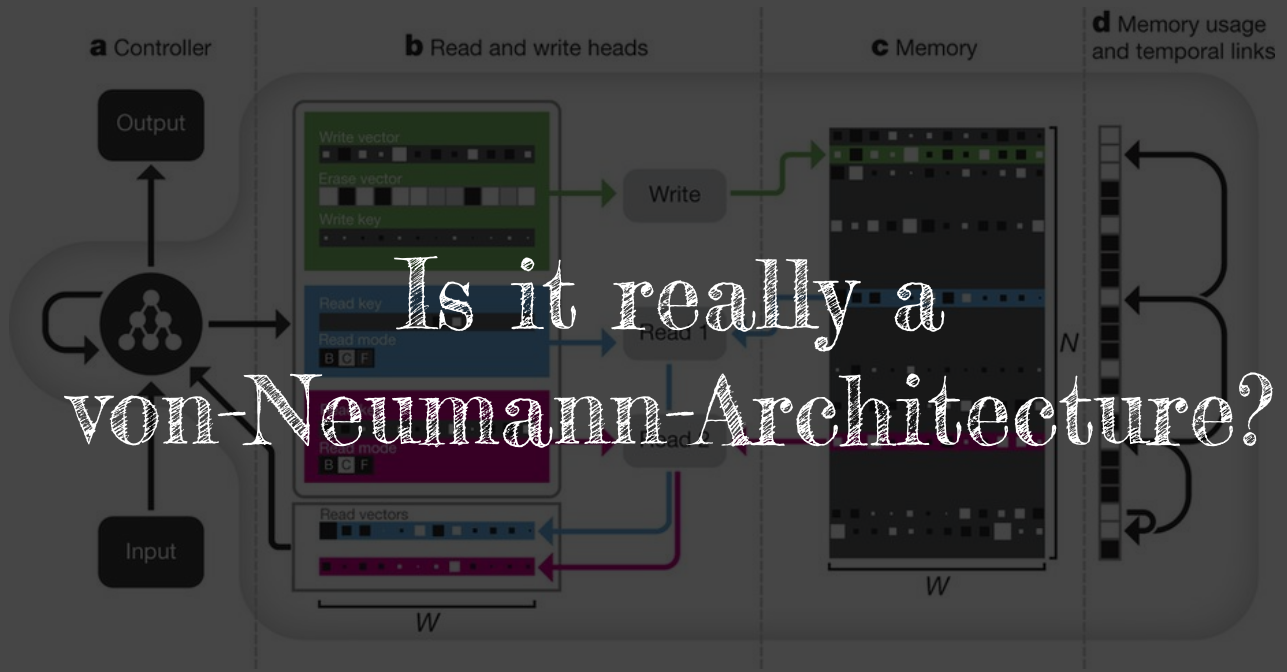
Minor cycles fall into two classes: *Standard numbers* and *orders*. (cf. 12.2 and 14.1.) These two categories should be distinguished from each other by their respective first units (cf. 12.2) i.e. by the value of i_0. We agree accordingly that $i_0 = 0$ is to designate a standard number, and $i_0 = 1$ an order.

Minor cycles fall into two classes: *Standard numbers* and *orders*. (cf. 12.2 and 14.1.) These two categories should be distinguished from each other by their respective first units (cf. 12.2) i.e. by the value of i_0. We agree accordingly that $i_0 = 0$ is to designate a standard number, and $i_0 = 1$ an order.

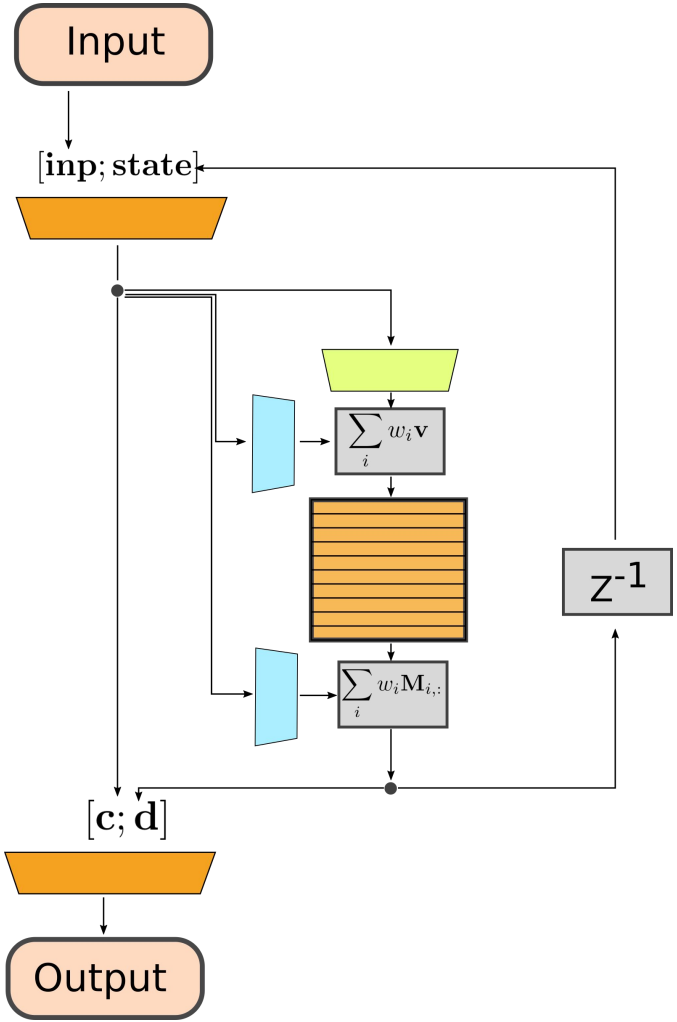
15.2 The remaining 31 units of a standard number express its binary digits and its sign. It is in the nature of all arithmetical operations, specifically because of the role of carry digits, that the binary digits of the numbers which enter into them must be fed in from right to left, i.e. those with the lowest positional values first. (This is so because the digits appear in a temporal succession and not simultaneously, cf. 7.1. The details are most simply evident in the discussion of the adder in 7.2.) The sign plays the role of the digit farthest left, i.e. of the highest positional value (cf. 8.1). Hence it comes last, i.e. $i_{31} = 0$ designates the + sign and $i_{31} = 1$ the − sign. Finally by 9.2 the binary point follows immediately after the sign digit, and the number ξ thus represented must be moved mod 2 into the interval -1, 1. That is $\xi = i_{31} i_{30} i_{29} \cdots i_1 = \sum_{v=1}^{31} i_v 2^{v-31} \pmod 2$, $-1 \leq \xi < 1$.

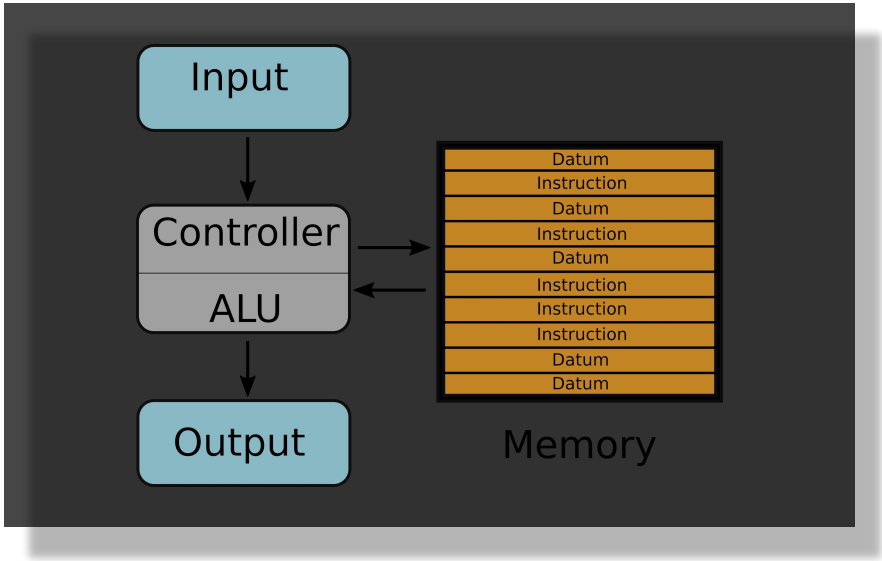
15.3 The remaining 31 units of an order, on the other hand, must express the nature of this order. The orders were classified in 14.1 into four classes (a)–(d), and these were subdivided further as follows: (a) in 11.4, (b) in 14.2, (b) and (c) in 14.3, 14.4, and the second remark in 14.5. Accordingly,

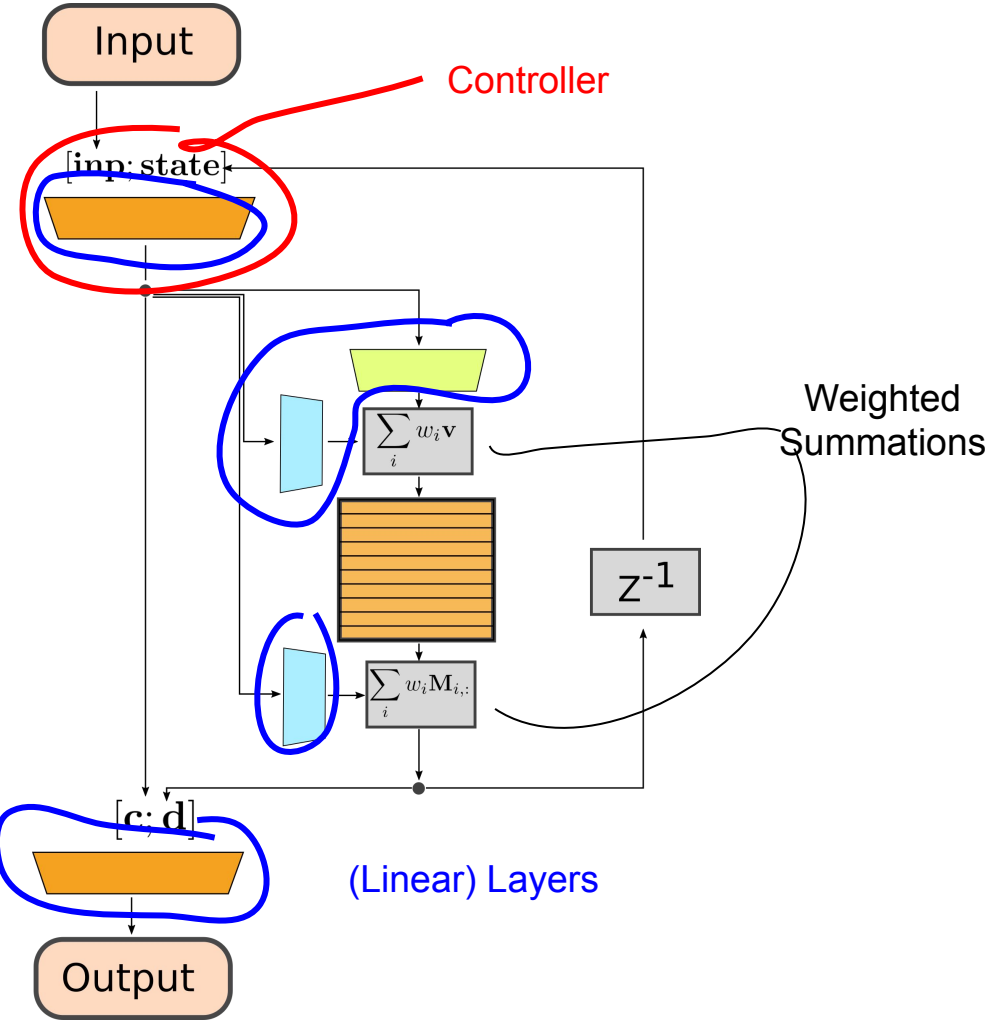

Input
Controller
ALU
Output
Datum
Instruction
Datum
Instruction
Datum
Instruction
Instruction
Instruction
Datum
Datum
Memory

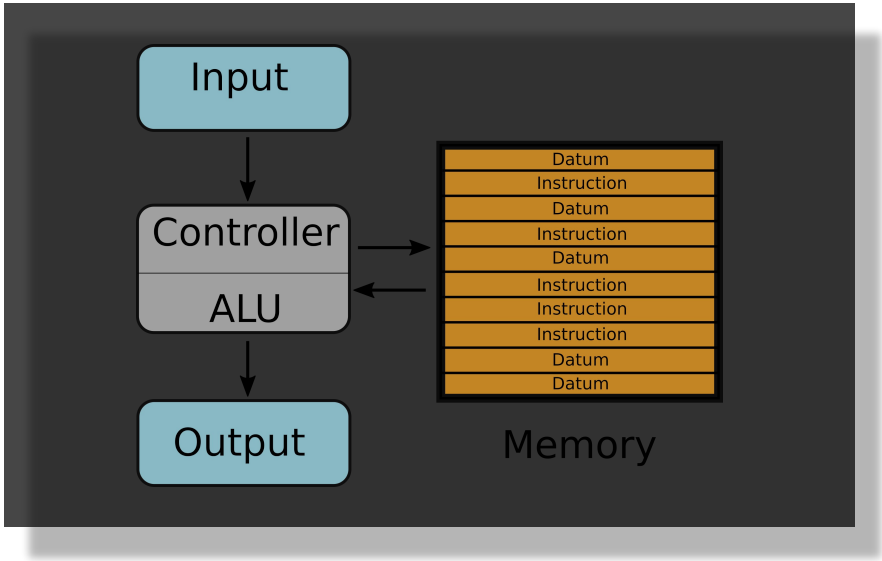


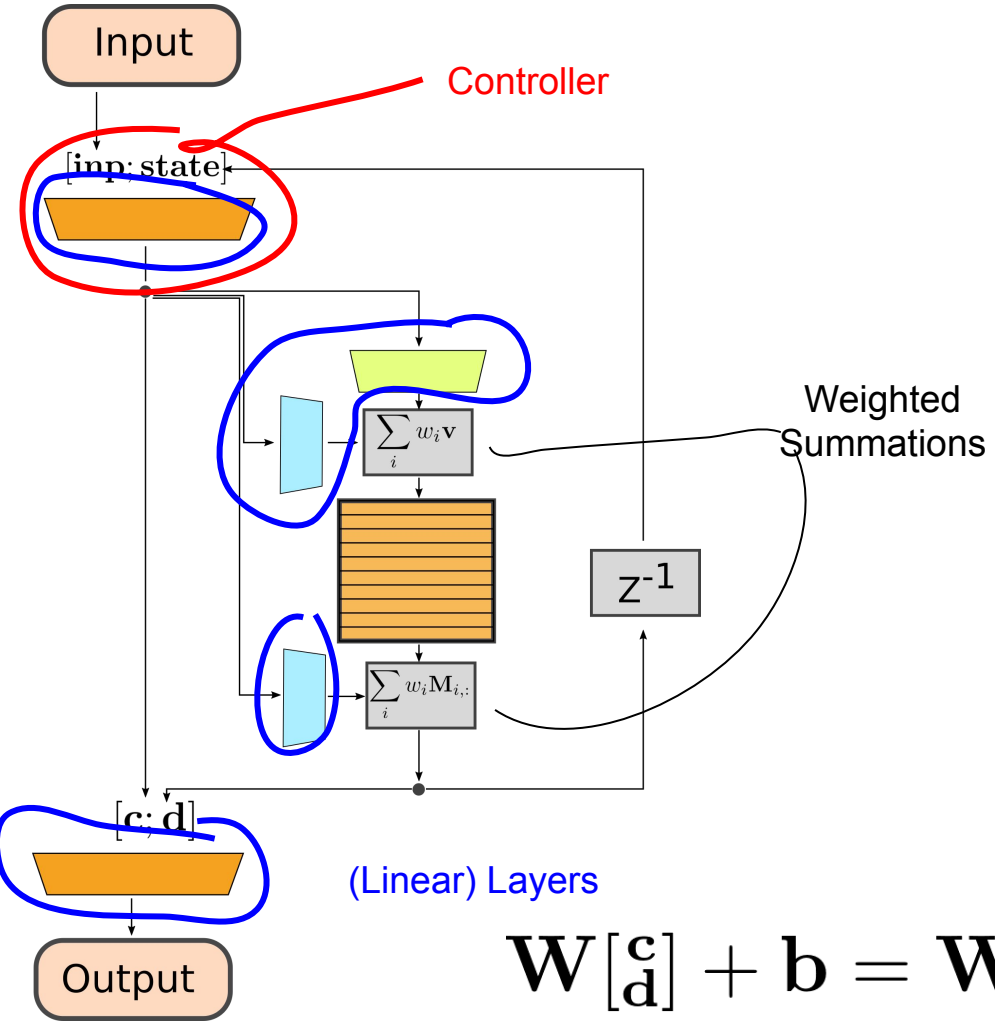
Is it really a von-Neumann-Architecture?

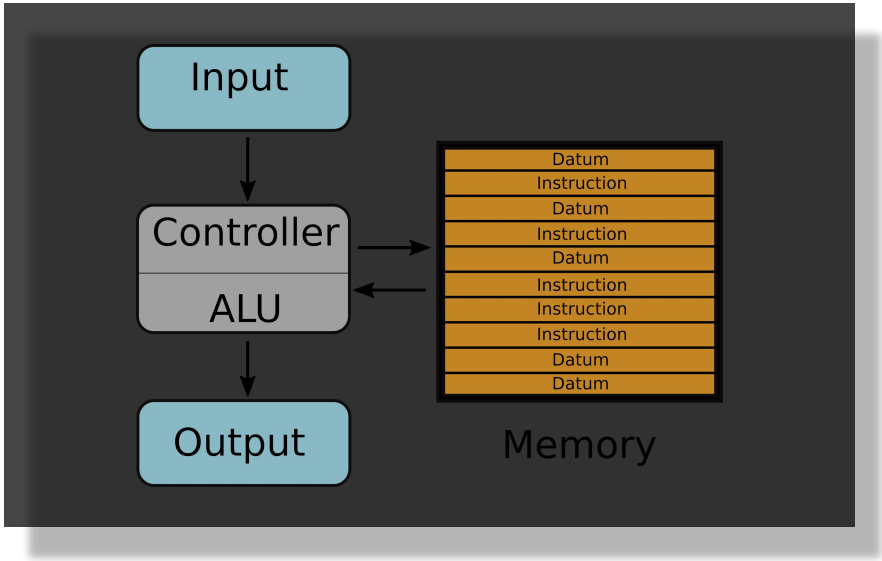

Input
[inp; state]
$\sum_i w_i \mathbf{v}$
z^{-1}
$\sum_i w_i \mathbf{M}_{i,:}$
[c; d]
Output


Input
Controller
ALU
Output
Datum
Instruction
Datum
Instruction
Datum
Instruction
Instruction
Instruction
Datum
Datum
Memory


Input
Controller
[inp; state]
$\sum_i w_i \mathbf{v}$
Weighted
Summations
z^{-1}
$\sum_i w_i \mathbf{M}_{i,:}$
[c; d]
(Linear) Layers
Output

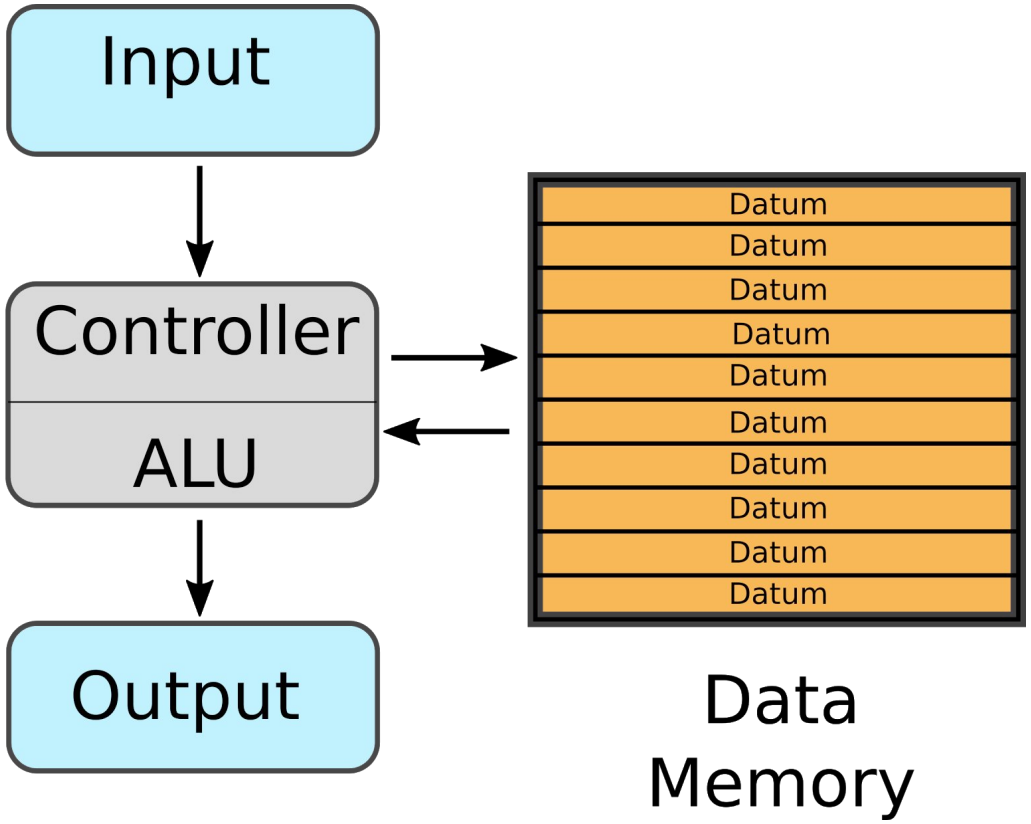

Input
Controller
ALU
Output
Datum
Instruction
Datum
Instruction
Datum
Instruction
Instruction
Instruction
Datum
Datum
Memory

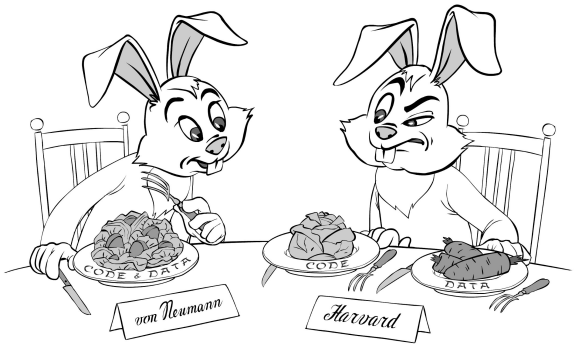


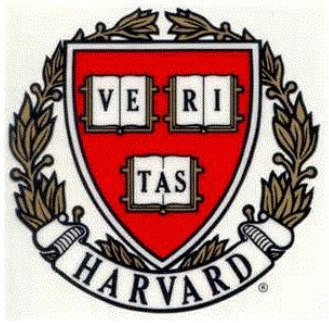


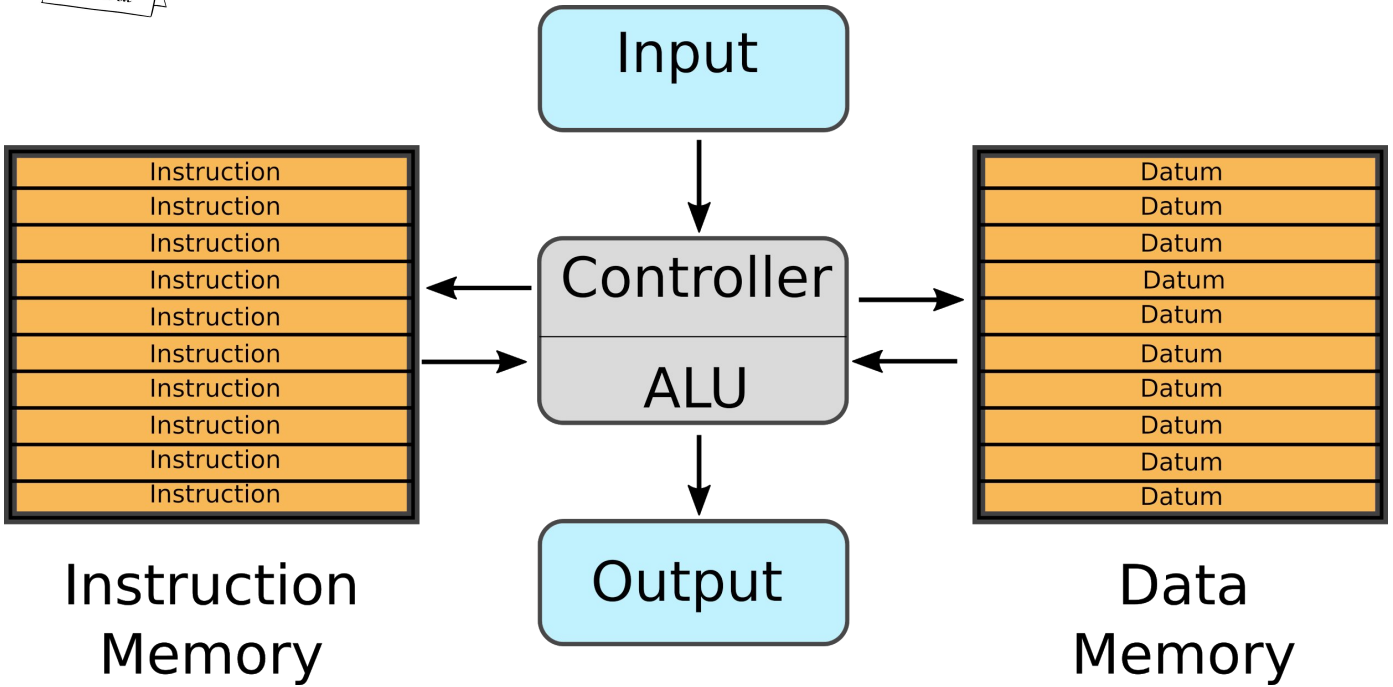
(Linear) Layers

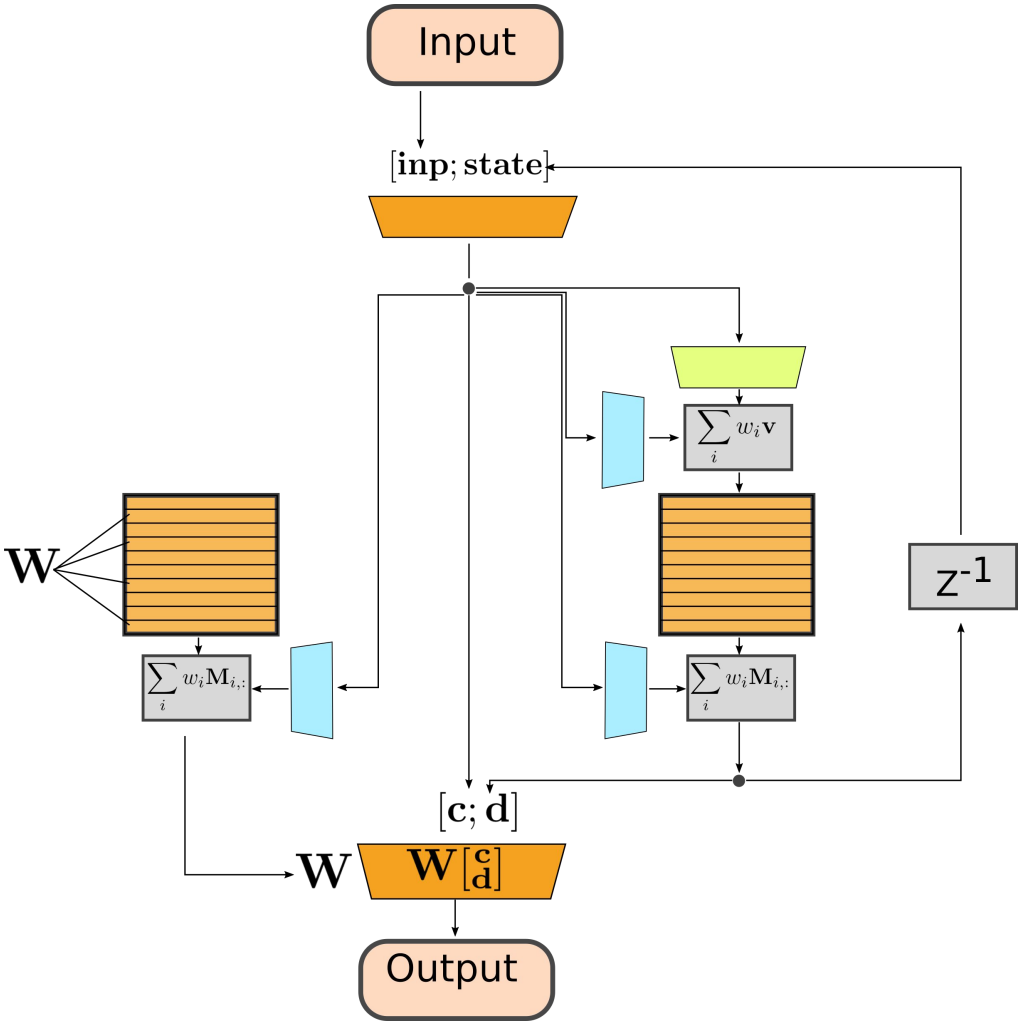
$$\mathbf{W}\begin{bmatrix}\mathbf{c}\\ \mathbf{d}\end{bmatrix} + \mathbf{b} = \mathbf{W}_{:,c}\mathbf{c} + \mathbf{W}_{:,d}\mathbf{d} + \mathbf{b}$$


Input
Controller
ALU
Output
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Data Memory


CODE & DATA
CODE
DATA
von Neumann
Harvard

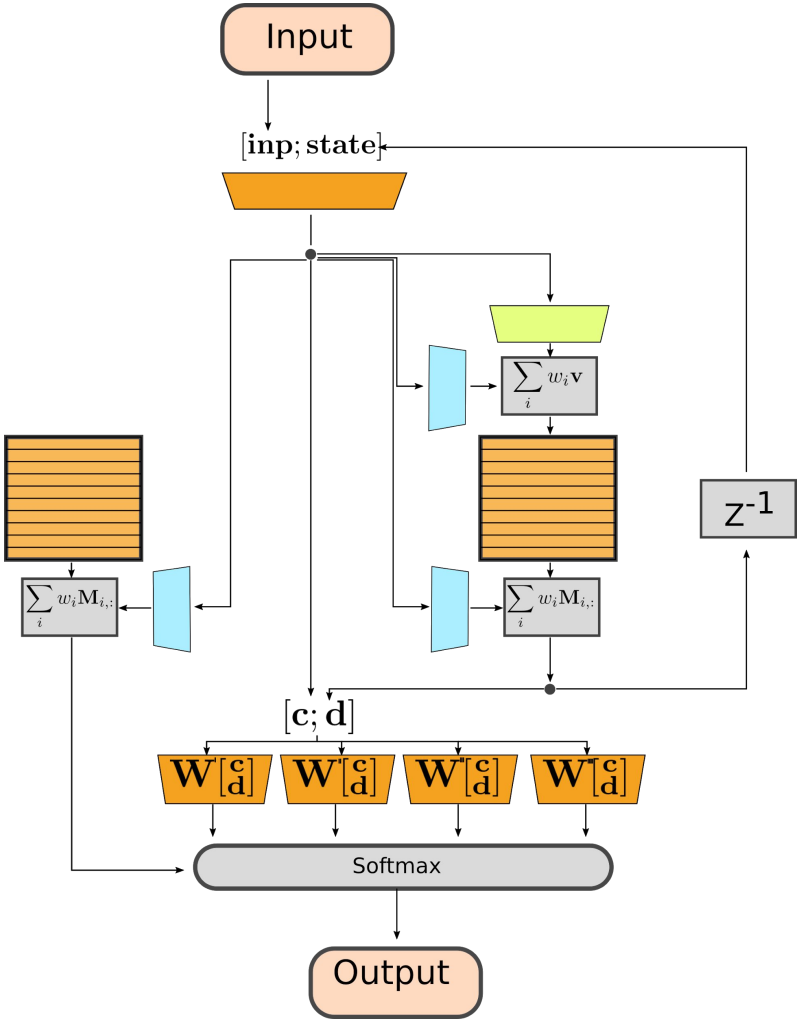

VE
RI
TAS
HARVARD


Input
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Controller
ALU
Output
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Datum
Instruction Memory
Data Memory



1. Idea

Add Instruction Memory



2. Idea

Add
Instruction Memory
&
ALU-Layers

But.... what is the benefit?

But.... what is the benefit?

Most DNCs are trained on one task.

Maybe we can find that the proposed architectures has an advantage for multi-task scenarios?

But.... what is the benefit?

Most DNCs are trained on one task.

Maybe we can find that the proposed architectures has an advantage for multi-task scenarios?

(Conservative) Hypothesis

Instruction memory or ALU-Layers allow faster convergence for Multi-Task scenarios

But.... what is the benefit?

Most DNCs are trained on one task.

Maybe we can find that the proposed architectures has an advantage for multi-task scenarios?

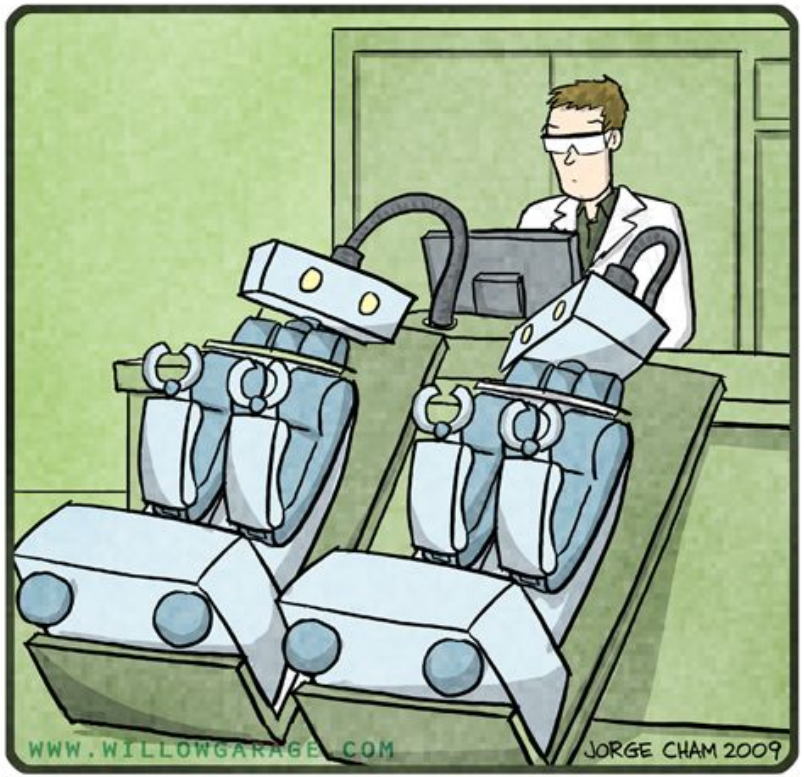
Hypothesis

Instruction memory or ALU-Layers make the DNC.... BETTER!

Thanks!

ksluck@asu.edu

R.O.B.O.T. Comics



"Do you ever feel like you're in The Matrix?"