The Differentiable Neural Computer - a neural network inspired by human memory

Quantitative Study of the BrsDNC - investigating the differences in performance on the bAbI tasks between the DNC and the BrsDNC.

Romain Dumon[1]

MEng Computer Science

John Shawe-Taylor

Submission date: 29^{th}April 2019

[1]**Disclaimer:** This report is submitted as part requirement for the MEng Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

The purpose of this project is to study the Differentiable Neural Computer (DNC) proposed by Alex Graves et al. and investigate current research on its architecture - the robust and scalable Differentiable Neural Computer (rsDNC) and the Bidirectional rsDNC (BrsDNC) for question answering proposed by Joerge Franke et al. First, this project identifies a lack of research on the impact of the four improvements proposed by the BrsDNC on the bAbI tasks results, and carries out an ablation study. The results of this study suggest that the layer normalization is the main contributor to the improvements in mean results. The results also suggest that the layer normalization is the main contributor to the improvements in convergence time. Secondly, this report also investigates the content based memory unit (CBMU) proposed by the rsDNC. The CBMU removes the temporal linkage matrix present in the original DNC. This report introduces two new tasks to test the impact of the CBMU on the rsDNC's ability to solve question answering tasks which involve remembering the order in sequences. The results of this investigation suggests that the rsDNC is better at remembering the order in sequences than the DNC, and that the purpose of the temporal linkage matrix in the DNC needs further research.

Contents

Chapter 1

Introduction

In October 2016, Alex Graves et al. introduced the Differentiable Neural Computer (DNC) : a new Memory Augmented Neural Network (MANN) capable of "answering questions about complex, structured data, including artificially generated stories, family trees, and even a map of the London Underground [23]." This architecture is highly regarded; the DNC sits at the frontier between Neuroscience and Computer Science. Alex Graves et al. use concepts from human memory to build a neural network which learns to use a memory unit. This research report studies the DNC's architecture, looks at its history and explains why it was invented. This report also takes a closer look at current research on the Differentiable Neural Computer and investigates the differences in performance on the bAbI question answering dataset between the Differentiable Neural Computer (DNC) and the Bidirectional robust and scalable Differentiable Neural Computer (BrsDNC).

1.1 Background

The Differentiable Neural Computer (DNC) is a Memory Augmented Neural Network (MANN). To understand the DNC, the Background chapter looks at the history behind Memory Augmented Neural Networks and explains the research which led to the development of these networks. The details of the Differentiable Neural Computer are then explained based on the supplementary information from Alex Graves et al.'s paper. This chapter finally takes a closer look at the relationship between the DNC and Neuroscience concepts. It makes a comparison between the human brain and the DNC taking ideas from Sam Greydanus [17]. The aim of this part of the report is not only to explain the DNC, but also understand the DNC's research context.

1.2 Research

Since October 2016, researchers have suggested many improvements to the DNC's architecture. The Research chapter focuses on the robust and scalable Differentiable Neural Computer (rsDNC) and the Bidirectional rsDNC (BrsDNC) proposed by Joerg Franke et al. These models improve the performance of the DNC on the bAbI question answering tasks [24].

Alex Graves et al.'s original DNC solves at best nineteen out of the twenty bAbI tasks, but on average only solves seven. Joerg Franke et al. argue that this is because of the model's unstable structure and propose four improvements. This report conducts an ablation study and tests each improvement on each of the twenty tasks. The report then looks at one particular improvement: the content based memory unit (CBMU). This improvement suggests to remove the temporal memory linkage of the DNC to decrease the model's memory consumption. Joerg Franke et al. also say: "the temporal memory linkage is not necessary for question answering tasks". This report hypothesizes that the rsDNC would perform worse than the DNC on question answering tasks which involve remembering the order in sequences. To test the hypothesis, two new question answering tasks in the style of the bAbI tasks are designed and are then tested on the rsDNC and the DNC. The Research chapter details the investigation of the BrsDNC: ablation study and testing of the CBMU. This chapter is written almost as a stand alone research paper but does not include the usual conclusion and future work sections. These sections are found in the Conclusion chapter of the report.

1.3 Technical Details and Conclusion

Before the Conclusion chapter, the Technical Details chapter details the different folders in the zip folder submitted with the report and also describes the planning of the experiments. This chapter also includes a reflection on the Technical Challenges encountered in this project. Finally, the Conclusion chapter summarises the background research, concludes the research project, proposes future works, and summarises the achievements looking back to the project requirements. The project requirements are detailed below in section 1.4.

1.4 Personal Motivation and Project Requirements

I decided to study the Differentiable Neural Computer out of interest for neural networks. Prior to the start of the year, I had no experience with neural networks or any machine learning concepts. I was motivated to learn more about this topic and get an understanding of state of the art research going on in this domain. I chose the DNC because of its connection to Neuroscience and its high level of complexity.

To judge the success of this project at the end of the report, I recall below the aims of the project defined in the interim report in November. The main objectives were:

1. individual explanations of the DNC and the BrsDNC architectures.
2. explanation of method used to solve the bAbI tasks and detailed explanations of experiments that will show the differences between the models.
3. analytical results of the comparison between the DNC and the BrsDNC on each task.
4. code which does the training of the two different models and reproduces the analytical results obtained in each original paper.

Chapter 2

Background

This chapter contains the background research information needed to understand the Differentiable Neural Computer. The first section hypothesizes on the importance of memory for solving human intelligence. This paragraph serves as an introduction to the section on Memory Augment Neural Networks (MANN) - neural networks with a memory. This section recalls the history behind DNC and starts by explaining Recurrent Neural Networks and Augmented Recurrent Neural Network architectures. This is followed by a detailed explanation of the Differentiable Neural Computer and a paragraph on the links between the DNC and the human brain. The aims of this chapter are: explain the history of MANNs, explain the DNC, and link the DNC to human memory.

2.1 From Memory to Intelligence

Artificial Intelligence (AI) researchers try to reproduce human thinking with algorithms. Their hope is to improve the capabilities of current computers and potentially one day create machines with the ability to think. Alan Turing defines the Turing test as the threshold between algorithms and human thinking. In *Computing Machinery and Intelligence*, he introduces the 'imitation game' and asks if it is possible for a machine to successfully imitate and be indistinguishable from other humans. The Turing test can be said to be the most famous of all artificial intelligence problems but it has yet to be solved. Memory Augmented Neural Network (MANN) researchers believe that building a machine which can imitate how a human remembers could potentially bring the field of AI one step closer to solving the Turing test.

Memory is important for two reasons. First, it enables humans to reason and think logically. Christopher Olah writes "you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence [6]." Secondly, memory is key to human identity. In her article "I remember, therefore I am [35]", Jennifer Miller extends Descartes' "Cogito ergo sum. I think therefore I am." She writes that Descartes' dictum is perhaps satisfying from a philosophical point of view but lacks the idea of 'human identity.' It seems not enough to 'think' to 'be human', being human also means being unique and different to others. Jennifer Miller believes this uniqueness comes entirely from our memories of the world we witness since the day we are born. Memory is therefore important to humans because it enables us to reason and gives us our identity. To solve the Turing Test, researchers need to build machines that identical to humans. For the two reasons highlighted above,

it is clear that replicating human memory is a major step for passing the Turing test.

2.2 Memory Augmented Neural Networks

The Differentiable Neural Computer is a Memory Augmented Neural Network (MANN): a neural network which learns to use an external memory. This section explains the history and ideas behind Memory Augmented Neural Networks starting from Recurrent Neural Networks. An overview of the Neural Turing Machine (NTM) is also given at the end.

A Recurrent Neural Network (RNN) is a type of neural network where the input is fed as a temporal sequence [6]. A RNN is fed an input at each time step in the order of the input sequence and can produce outputs during each of these time steps. This is different from a traditional feed-forward neural network which given some input produces just some output without any sequences. A RNN uses contextual information from previous time steps to produce outputs in current time steps. In the original RNN architecture, the output from time step t is concatenated to the input at time step $t + 1$. More explicitly, the hidden states from the previous time step are concatenated to the input of the current time step.

The original RNN architecture described above was quickly replaced by better RNN architectures because of the vanishing gradient problem described in 1991 by Hochreiter [14]. In 1997, Hochreiter developed the Long Short Term Memory (LSTM) to solve this problem and improve its capabilities [9]. The LSTM is similar to the original RNN architecture, i.e., hidden states from previous time steps are concatenated with inputs in next time step, but introduces an internal memory unit which is also passed along from time step to time step. To update this internal memory unit the LSTM computes values called gates responsible for updating and forgetting information in its internal memory. These gates make the LSTM a powerful architecture which can learn to extract patterns in sequences.

The LSTM was a major breakthrough and has since its invention been successfully implemented to solve a number of tasks. For example, Alex Graves et al. used the LSTM for speech recognition and achieve a new best test set error score of 17,7% on the TIMIT phoneme recognition Benchmark [7]. Another example is the new state of the arts results on language modelling tasks achieved using a LSTM by Sundermeyer et al. in 2012 [16]. The great results obtained from using LSTMs also pushed researchers to optimize its architecture. For example, in *Recurrent Nets that time and count*, Gergs and Schmidhuber introduce the LSTM with peephole connections which improve the capabilities of the internal memory state [15]. The Gated Recurrent Unit or GRU by Cho et al. in 2014 is another improvement [8]. The GRU computes different gates from those of the LSTM but uses the same exact concept of an internal memory state. The GRU gates give it more control over what information it extracts from the internal memory at a given timestep. In their paper, Cho et al. successfully apply the GRU to a machine translation task and achieve state of the art results. Overall, Recurrent Neural Network with an internal memory state reached a point where they dominated Deep Learning.

More recently, researchers have suggested that these RNNs have certain limitations. It has been shown that RNN with an internal memory state (e.g. LSTM) have problems remembering extended long term sequences [18]. Some researchers have also pointed out that the cost of training RNNs scales poorly with memory size [17]. To solve these problems, a new type of Recurrent Neural Networks were developed called

Memory Augmented Neural Networks (MANN).

Memory Augmented Neural Networks are similar to Recurrent Neural Networks in that information from the previous time step is passed to the current time step. The main difference is in the addition of an external memory. This idea comes from the functioning of the human brain. In Neuroscience, the human memory is divided into two main parts: short term storage (STS) and long term storage (LTS) [22]. This comes from the Search of Associative Memory (SAM) model introduced by Atkinson in 1968 [21]. Depending on the task it has to perform, e.g., remembering an old memory or understanding someone in a conversation, the brain interacts with these storages differently. It was mentioned earlier that the LSTM had the ability to remember well short term sequences but problems remembering long term ones. In this case, it can be said that the LSTM would be equivalent to the short term storage mechanism in the SAM model. The external memory introduced in MANNs is added to solve the problem of remembering long term sequences. The external memory of MANN becomes the long term storage of the SAM model.

The general architecture of Memory Augmented Neural Networks consists of a neural network which interacts with an external memory. This neural network is called the controller. Researchers have developed different varieties of this architecture [19, 20]. In the following paragraphs, the general architecture of a Memory Augmented Neural Network is described followed by an overview of the Neural Turing Machine (NTM).

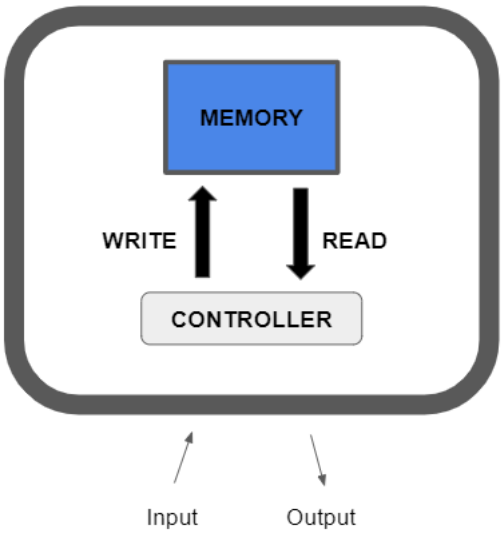

Figure 2.1: Memory Augmented Neural Network

The image above (figure 2.1) shows the basic idea behind a Memory Augmented Neural Network at a particular time step. The controller is given an input and decides what to do with this input. For instance, this input could be a question mark token '?' in the case of a question answering task and would indicate to the network that it has to perform a read at the next time step. The input could also be a particular word in a sentence, like the name of an animal, which the network has learned to write to a specific location in

the memory. In essence, a memory augmented neural network is a neural network which learns to use an external memory.

Before introducing the Differentiable Neural Computer it is worth understanding some of the concepts of the Neural Turing Machine (NTM) which the DNC builds upon. The Neural Turing Machine was built in 2014 by Alex Graves et al [11] and is one of the first examples of Memory Augmented Neural Networks.

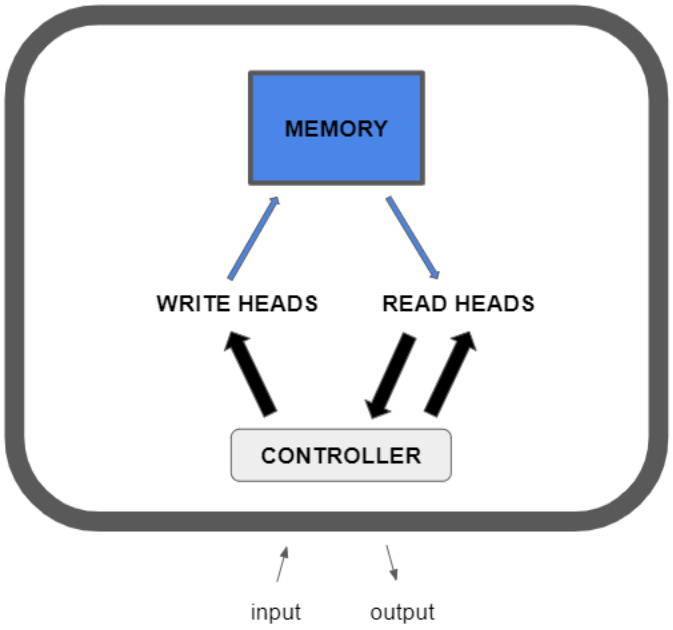

Figure 2.2: Neural Turing Machine

The NTM has two basic components as seen in figure 2.2: the memory unit and the controller unit. The memory unit is a N $\times$ M memory matrix, where N is the number of memory locations, and M is the vector size at each location. The controller is a neural network (either feed-forward or recurrent) and interacts with this memory matrix using selective write and read mechanisms. In their paper, Alex Graves et al. find the LSTM to work best as the controller unit [11].

These mechanisms are called "heads". At each time step t, the controller unit emits a set of outputs to the read and write heads. Each head then produces a weighting depending on keys it receives from the controller unit output. This weighting enables the head to focus on particular memory locations. The read and write head in the Neural Turing Machine have the same weighting mechanism. This weighting consists of a focus by content and a focus by location mechanism. The head can then get/write information to a specific location in memory or not depending on what the controller tells it to do. All these mechanisms are differentiable which makes the NTM trainable with back-propagation.

At each time step, the controller activates multiple read heads and one write. First, the write weighting is calculated and then the read weighting. The write weighting is applied to the memory matrix and

re-calculated for every time step. On the other hand, the weightings from the read heads are normalized and applied to the memory matrix to read a specific location at the current time step. The read heads are then fed back to the controller and passed as inputs to the next time step along the external input. In the diagram, this is illustrated by the dark arrow going back to the controller from the read heads.

The NTM was a great innovation and was the main source of inspiration for the development of the Differentiable Neural Computer. The authors of the NTM compare its performance to that of a simple LSTM on a multitude of tasks notably the copy task and the repeat copy task. In their experiments, the NTM outperforms the LSTM in how quickly it converges to the solution and gets better test accuracy on all the tasks. In conclusion, there exists other examples of Memory Augmented Neural Networks. Each network shows interesting results and builds upon the structure described in Figure 2.1. As researchers are discovering the limits of LSTMs and other Recurrent Neural Networks, the research into Memory Augmented Neural Networks aims to fill the gaps in performance and solve problems that were not possible before. Memory Augmented Neural Networks are bringing us one step closer to fully autonomous artificial intelligence. The following section investigates in more detail another Memory Augmented Neural Network: the Differentiable Neural Computer.

2.3 Differentiable Neural Computer

The Differentiable Neural Network was introduced by Alex Graves et al. in October 2016. It builds upon the Neural Turing Machine. The NTM "used a similar architecture of neural network controller with read/write access to a memory matrix, but differed in the access mechanism used to interface with the memory [23]." This section is divided into two sub-sections: description of the DNC (section 2.3.1), analogy between the human brain and the DNC (section 2.3.2).

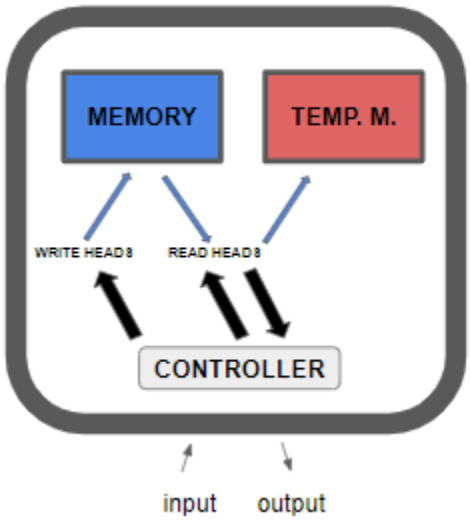

Figure 2.3: Differentiable Neural Computer

2.3.1 Model Architecture

This section describes the DNC following the supplementary information of the original paper [23]. This section first explains the controller unit, followed by the read/write heads, the memory unit, the temporal link matrix and the DNC output.

Controller Unit. The controller unit controls what is stored in the external memory. At every time step t the controller unit receives an input vector $\boldsymbol{x}_t \in \mathbb{R}^X$ from the dataset which the model is being trained on and an output vector $\boldsymbol{y}_t \in \mathbb{R}^Y$. The controller also receives a set of read vectors from the previous time step $\boldsymbol{r}^1_{t-1}, ..., \boldsymbol{r}^R_{t-1}$ where R is the number of read heads. These read vectors come from the memory matrix $M_{t-1} \in \mathbb{R}^{N \times M}$ at the previous time step. The concatenation of the input vector and the read vectors from the previous time step is called the controller unit input vecor $\boldsymbol{\chi}_t = [\boldsymbol{x}_t; \boldsymbol{r}^1_{t-1}; ...; \boldsymbol{r}^R_{t-1}]$. Any neural network (recurrent or feedforward) can be picked to be the controller unit. Alex Graves et al. use the following LSTM. They use the LSTM according to their findings on the Neural Turing Machine architecture.

$$\boldsymbol{f}^l_t = \sigma(W^l_f[\boldsymbol{\chi}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t] + \boldsymbol{b}^l_f)$$

$$\boldsymbol{i}^l_t = \sigma(W^l_i[\boldsymbol{\chi}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t] + \boldsymbol{b}^l_i)$$

$$\boldsymbol{o}^l_t = \sigma(W^l_o[\boldsymbol{\chi}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t] + \boldsymbol{b}^l_o)$$

$$\boldsymbol{s}^l_t = \boldsymbol{f}^l_t \boldsymbol{s}^l_{t-1} + \boldsymbol{i}^l_t \tanh(W^l_s[\boldsymbol{\chi}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t] + \boldsymbol{b}^l_s)$$

$$\boldsymbol{h}^l_t = o^l_t \tanh(\boldsymbol{s}^l_t)$$

where l is the layer index, σ is the logistic sigmoid function $\sigma(x) = 1/(1 + e^{-x})$, $W^l_{l,i,o,s}$ and $b^l_{l,i,o,s}$ are the learnable weights and biases. At each time step this controller unit outputs an output vector $\boldsymbol{v}_t$ and an interface vector $\boldsymbol{\xi}_t \in \mathbb{R}^{(W \times R)+3W+5R+3}$.

$$\boldsymbol{v}_t = W_y[\boldsymbol{h}^1_t; ...; \boldsymbol{h}^L_t]$$

$$\boldsymbol{\xi}_t = W_\xi[\boldsymbol{h}^1_t; ...; \boldsymbol{h}^L_t]$$

First, v_t is the controller output. It is used to calculated the DNC output vector at the end (see DNC output). The interface vector $\boldsymbol{\xi}_t$ is how the controller unit interacts with the memory at every time step. The evidence vector is composed of many sub-components which all have a role in the memory read and writes.

$$\boldsymbol{\xi}_t = [\boldsymbol{k}^{r,1}_t; ...; \boldsymbol{k}^{r,R}_t; \hat{\beta}^{r,1}_t; ...; \hat{\beta}^{r,R}_t; \boldsymbol{k}^w_t; \hat{\beta}^w_t; \hat{\mathbf{e}}_t; \boldsymbol{v}_t; \hat{f}^1_t; ...; \hat{f}^R_t; \hat{g}^a_t; \hat{g}^w_t; \hat{\pi}^1_t; ...; \hat{\pi}^R_t]$$

All the sub-components which are presented here are all detailed in Appendix A. These sub-components are constrained with certain functions so that they lie in the correct domain. This is also described in the Appendix A. The resulting evidence vector after applying the functions is:

$$\boldsymbol{\xi}_t = [\boldsymbol{k}^{r,1}_t; ...; \boldsymbol{k}^{r,R}_t; \beta^{r,1}_t; ...; \beta^{r,R}_t; \boldsymbol{k}^w_t; \beta^w_t; \mathbf{e}_t; \boldsymbol{v}_t; f^1_t; ...; f^R_t; g^a_t; g^w_t; \pi^1_t; ...; \pi^R_t]$$

Write to memory. The controller unit outputs the interface vector to the write and read heads. The DNC first writes to memory and then reads from its memory. This is similar to what is done in the Neural Turing Machine. The DNC has one write head. This paragraph explains the write weighting and then details the

write head operation on the memory.

At every time step, the write head modifies the memory of the previous time step. In order to modify according to the inputs at each time step, the write computes a write weighting. The write weighting of the DNC has a content based addressing attention mechanism. This mechanism is used to enable the DNC to identify any relationship between the input at a current timestep and what is currently in its memory. Potentially, the DNC will learn to put similar words in certain parts of its memory to retrieve them more easily during read actions in future timesteps. The write weighting also has a dynamic memory allocation mechanism. This is to allow it to keep track of what memory locations are free and can be written to. A write weighting is therefore defined as:

$$\boldsymbol{w}_t^w = g_t^w(g_t^a \boldsymbol{a}_t + (1 - g_t^a)\boldsymbol{c}_t^w)$$

where $g_t^a \in [0, 1]$ and $g_t^w \in [0, 1]$ are respectively the allocation gate and the write gate received from the interface vector outputted by the controller unit. Notice how if the write gate g_t^w is set to zero then nothing is written to memory, protecting from "unnecessary modifications". The write content weighting $\boldsymbol{c}_t^w$ is described before the allocation weighting $\boldsymbol{a}_t$.

$\boldsymbol{c}_t^w$ is the write content weighting constructed from the write key $\boldsymbol{k}_t^w$ and write strength β_t^w. Both the write key and write strength come from the interface vector outputted by the controller unit.

$$\boldsymbol{c}_t^w = C(M_{t-1}, \boldsymbol{k}_t^w, \beta_t^w)$$

C denotes the content based addressing mechanism which is the same as the one from the Neural Turing Machine. It is essentially the softmax distribution of the cosine similarities between the keys outputted by the controller and the memory location content. It is defined as:

$$C(M, \boldsymbol{k}, \beta)[i] = \frac{e^{D(\boldsymbol{k}, M[i,\cdot])}}{\sum_j e^{D(\boldsymbol{k}, M[i,\cdot])}}$$

where D is the cosine similarity between the key $\boldsymbol{k}$ and the memory location $M[i, \cdot]$. The cosine similarity is defined as:

$$D(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{|\boldsymbol{u}||\boldsymbol{v}|}$$

Coming back to the write weight equation $\boldsymbol{w}_t^w$, the last important term to understand is $\boldsymbol{a}_t$ which is the allocation weighting. The idea behind this allocation weighting is to have a "differentiable analogue of the 'free list' memory allocation scheme, whereby a list of available memory locations is maintained by adding to and removing addresses from a linked list [23]."

$$\boldsymbol{a}_t[\phi_t[j]] = (1 - \boldsymbol{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \boldsymbol{u}_t[\phi_t[i]]$$

$$\boldsymbol{u}_t = (\boldsymbol{u}_{t-1} + \boldsymbol{w}_{t-1}^w - \boldsymbol{u}_{t-1} \circ \boldsymbol{w}_{t-1}^w) \circ \boldsymbol{\psi}_t$$

$$\boldsymbol{\psi}_t = \prod_{i=1}^{R} (\mathbf{1} - f_t^i \boldsymbol{w}_{t-1}^{r,i})$$

Here, $\circ$ denotes element-wise multiplication. This paragraph first explains $\boldsymbol{\psi}_t$, then $\boldsymbol{u}_t$ and finally $\boldsymbol{a}_t$. In $\boldsymbol{\psi}_t$, the free gates f_t^i come from the interface vector outputted by the controller unit. There is one free gate for every read head. These free gates determine whether the most recently read locations can be freed. The free gates are used in the computation of the memory retention vector $\boldsymbol{\psi}_t \in [0,1]^N$. This retention vector is used to "represent by how much each location will not be freed by the free gates [23]." In other words, the retention vector identifies what memory locations will be retained from the previous time step ($\boldsymbol{w}_{t-1}^{r,i}$), e.g., $\boldsymbol{\psi}_t \approx 1$ means that the free gates are retaining everything read at the previous time step. The retention vector is then used to calculate the memory usage vector at time t, $\boldsymbol{u}_t \in [0,1]^N$. This vector is initialized to zero before the start of any unrolling of the DNC, $\boldsymbol{u}_0 = 0$. Intuitively, the usage vector records the usage of the memory. The usage vector calculates a usage value for every memory location. It takes into account what has been written to memory from the previous time step but also what has been read. For example, the DNC might want to forget something which has been read in the previous time step. Then, it would want the usage vector to record that there is a newly freed memory location. Once the usage vector is computed, a free list $\phi_t \in \mathbb{Z}^N$ is defined. This free list is computed at every time step. This free list is equal to the indices of the memory locations sorted in the ascending order of usage. The index of the least used memory location is therefore $\phi_t[1]$. This free list will be used to decide where to write to in the memory. The allocation weighting $\boldsymbol{a}_t \in \triangle_N$ uses the free list to allocate memory. The allocation weighting equation is set up in such a way that if $\boldsymbol{u}_t = 1$ then the allocation will be set to 0. Meaning that the memory is full.

The write weighting is then used to modify the memory matrix. More precisely, the write weighting is multiplied to the memory matrix.

$$M_t = M_{t-1} \circ (E - \boldsymbol{w}_t^w \boldsymbol{e}_t^T) + \boldsymbol{w}_t^w \boldsymbol{v}_t^T$$

$\boldsymbol{e}_t^T \in [0,1]^W$, $\boldsymbol{v}_t^T \in [0,1]^W$ and E denote respectively the erase vector, write vector and a $N \times M$ matrix of ones. Both the erase vector and the write vector are emitted by the controller unit and are found in the interface vector. This is how the memory is changed at every time step.

Read from Memory. Once the write operation has been applied to the memory, the read operations are applied. The DNC has multiple read heads. Like the write head, the read heads use parameters from the interface vector outputted by the controller unit to calculate read weightings. These read weightings are then applied to the memory to set read vectors. This section first explains the read weighting and then explains how the read vectors are calculated. A read weighting $\boldsymbol{w}_t^{r,i} \in \boldsymbol{S}_3$ is defined by:

$$\boldsymbol{w}_t^{r,i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{r,i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

$$\boldsymbol{c}_t^{r,i} = C(M_t, \boldsymbol{k}_t^{r,i}, \beta_t^{r,i})$$

$$\boldsymbol{b}_t^i = L_t \boldsymbol{w}_{t-1}^{r,i}$$

$$\boldsymbol{f}_t^i = L_t \boldsymbol{w}_{t-1}^{r,i}$$

The read weighting is composed of three weightings. Each weighting is parameterized by a read mode vector $\boldsymbol{\pi}_t^i \in \boldsymbol{S}_3$. There is one read mode vector for each read head. The read mode vectors are outputted by the controller unit via the interface vector. The read mode vector consists of three elements; each puts different

weight on the parameters of the read weight. If $\boldsymbol{\pi}_t^i[2]$ is higher than the other two values, the weighting will use the content based addressing during that particular read. $\boldsymbol{c}_t^{r,i}$ denotes a content weighting using key $\boldsymbol{k}_t^{r,i}$ from the controller unit interface vector. The content based addressing mechanism is the same as for writing (Write to Memory). If $\boldsymbol{\pi}_t^i[3]$ is the highest of the three read modes than the forward weighting makes up a major portion of the read weighting $\boldsymbol{w}_t^{r,i}$. The forward weighting is denoted by $\boldsymbol{f}_t^i$ and records the order in which memory locations where written to. Lastly, if $\boldsymbol{\pi}_t^i[1]$ dominates the other two read modes than the read head will iterate the iterate through the order in which memory locations are written to in the reverse order. This is because $\boldsymbol{b}_t^i$ denotes the backward weighting. The forward and backward weighting of a read weighting are calculated by taking into account the read head's previous read weights and multiplying them to the temporal linkage matrix (explained in next paragraph). Once the read weighting of each of the different read heads are calculated, the read weights are then applied to the memory matrix to retrieve the memory locations to read. The read vectors $\{\boldsymbol{r}_1^1, ..., \boldsymbol{r}_t^R\}$ are defined as:

$$\boldsymbol{r}_t^i = M_t^T \boldsymbol{w}_t^{r,i}$$

It is important to remember that these read vectors are concatenated to the DNC inputs at the next time step giving the memory access to read content.

Temporal memory linkage. The last thing to look at is the temporal memory linkage matrix. The temporal linkage matrix is used to calculated the backward and forward weighting in the read weightings. This matrix stores the order in which memory locations are written to. If the DNC has to retrieve a sequence in order from memory, then the temporal memory linkage will enable it to output the sequence in the right order. $L_t \in [0,1]^{N \times N}$ represents the temporal linkage matrix.

$$L_t[i,j] = (1 - \boldsymbol{w}_t^w[i] - \boldsymbol{w}_t^w[j]) L_{t-1}[i,j] + \boldsymbol{w}_t^w[i] \boldsymbol{p}_{t-1}[j]$$

$$\boldsymbol{p}_t = (1 - \sum_i \boldsymbol{w}_t^t[i]) \boldsymbol{p}_{t-1} + \boldsymbol{w}_t^w$$

This paragraph explains $L_t[i,j]$ and then explains $\boldsymbol{p}_t$. $L_t[i,j]$ records if a memory location i was written to after memory location j. $L_t[i,j]$ values are computed for all i and j at every time step t. On time step 0, $L_t[i,j]$ is set to 0 for all i and j and at each time step t $L_t[i,j] = 0$ for all i. After initialisation, the temporal link matrix for every i and j is set to the value resulting from the equation described above. Note that it takes into account the current write weighting $\boldsymbol{w}_t^w[i]$ and the memory linkage matrix L_{t-1} from the previous time step. This is to update it according to what has been written to memory in the current time step. $L_t[i,j]$ also uses a precedence weighting which represents "the degree to which location i was the last one written to." This is to make sure that the temporal linkage matrix updates itself properly. More specifically, that it does not change the value of memory locations which have not be written to consecutively. $\boldsymbol{p}_0$ is initially set to 0. To summarize, the temporal linkage matrix is updated at every time step and keeps track how memory locations are being modified over time.

Sparsity Matrix. In the paper, the authors describe a sparse link matrix to extend their temporal linkage memory [23]. However, this sparse link matrix is not implemented in the code that they published after their paper [5]. Therefore the explanation disregards it.

DNC Output. After the read and write operations have been executed, the DNC outputs a vector for the time step $\boldsymbol{y}_t$. This output vector is composed of the output $\boldsymbol{v}_t$ from the controller unit and the read vectors. The exact equation is:

$$\boldsymbol{y}_t = \boldsymbol{v}_t + W_r[\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R]$$

$\boldsymbol{v}_t$ is the output vector outputted with the interface vector by the controller unit. W_r is a trainable weight and $[\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R]$ are the read vectors calculated for the current timestep. Finally, $\boldsymbol{y}_t$ is what is outputted by the DNC for the current timestep.

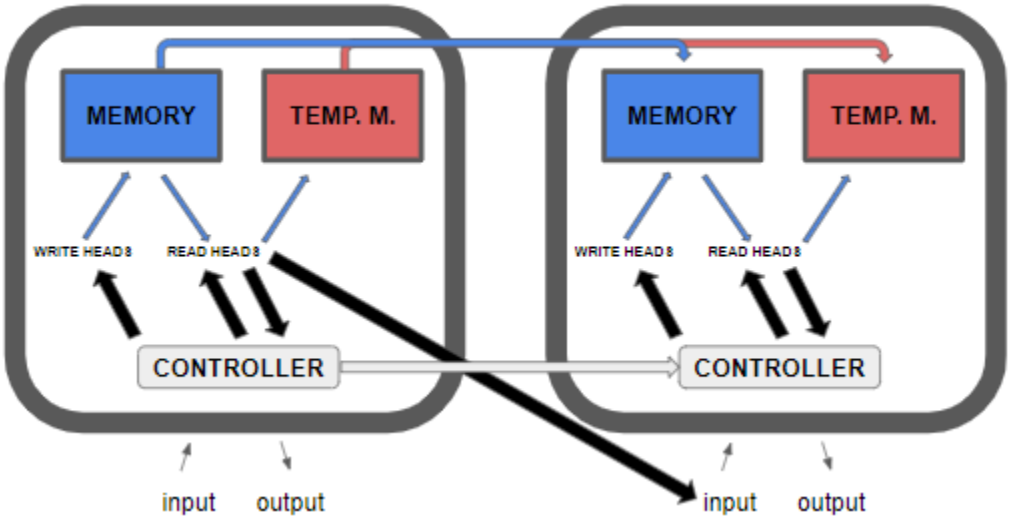

Figure 2.4: Unrolling Differentiable Neural Computer - from $t-1$ to t

Summary. The read vectors from the previous time step are concatenated to the DNC input at the current time step before being fed feed to the controller unit for the current time step. The memory matrix and the temporal matrix from the previous time step are also passed along to the following time steps. Figure 3.6 illustrates how the DNC unrolls over time. The figure illustrates how the temporal linkage and memory matrix are passed to the next time step. Note that the LSTM also passes along its hidden states to the next time step. The entire DNC described above is differentiable which makes it trainable using back-propagation.

2.3.2 Brain Analogy

An interesting analogy between the Differentiable Neural Computer and the human brain can be made [17]. This section summarises the main points from the research article by Sam Greydanus and also looks at where this analogy breaks down. The research article points out three characteristics from human memory concepts that the DNC uses. The concepts are Search of Associative Memory, Temporal Context Model and Serial Recall. The aim of this sub-section is to understand further the design choices made by the DNC's authors.

Search of Associative Memory or (SAM) is a human memory model which sees the memory as composed of a short term storage unit (STS) and a long term storage unit (LTS). The SAM model was introduced in the

Memory Augmented Neural Networks section. The important thing to note about the DNC is that it has both a STS and a LTS. The controller unit can be seen as the STS with the LSTM which has good short term memory capabilities but has been shown to failed for long tasks. The memory unit can be seen as the LST.

Temporal Context Model of memory is a concept introduced by Kahana et al. which models free recall [29]. This model is based on the idea that when a human remembers a sequence, he or she uses "retrieval of prior contextual states to drive the contextual drift [29]." The model suggests that there is a relationship between the order in which human recall thoughts from memory and their context, i.e., current thoughts are more likely to come directly after previous thoughts of similar context. For example, we would not think about cars directly after thinking about cats. Given the context, we are more likely to think about dogs or other domestic animals. In their paper, Kahana et al. introduce the Temporal Context Model and explain how they can express this idea mathematically. It is pretty obvious to see that the temporal linkage memory matrix of the DNC is similar to the temporal context model described here.

Lastly, the other two DNC mechanisms which resemble a human memory concept are the forward and backward weighting used in the calculation of the read weightings of the DNC. Recall the equation for the read weighting:

$$\boldsymbol{w}_t^{r,i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{r,i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

The forward and backward weighting are respectively $\boldsymbol{f}_t^i$ and $\boldsymbol{b}_t^i$. These weightings can be said to be similar to the free recall concepts introduced by Kahana in chapter 8 of *Foundations of Human Memory* [30]. In his book, Kahana explains associative chaining: "humans memorize series of items by creating memory links between adjacent items [17]." Free recall would essentially be similar to a doubly linked list in Computer Science theory, i.e. every node of a list points to the previous and next node. Coming back to the DNC, the read weighting $\boldsymbol{w}_t^{r,i}$ kind of imitates this doubly linked list idea with the backward and forward weightings.

Given the concepts explained above, it is clear that Alex Graves et al. used neuroscience literature to build the DNC's architecture. In the article, Greydanus performs a free recall test on the DNC and compares the result to that of a human. The results of the DNC are far worse than those of human. Although the DNC copies some human memory as seen above it lacks many other concepts which make up the human memory. The biggest of its limitations is probably that it learns through back-propagation. All of Neural Network theory relies upon back-propagation and a lot of evidence suggests that the brain does not learn with this technique. Back-propagation remains the main method used to train neural networks and is at the moment one of the biggest weaknesses of Neural Network theory.

Chapter 3

Research

This chapter reports the research which was conducted during this project. This chapter is written almost as a stand alone research paper but does not include the usual conclusion and future work sections. These sections are found in the Conclusion chapter of the report (Chapter 5). The sections found in this chapter are research motivation, background, method, results and discussion.

3.1 Introduction and Motivation

The Differentiable Neural Computer made a big impact in the domain of question answering by successfully solving nineteen of the twenty bAbI tasks. Back in 2016, the bAbI tasks were the state of the art in question answering. These tasks are made up of questions needing deductive and inductive reasoning to solve them which made them more complex them previous question answering datasets. The DNC is able to solve nineteen of the twenty bAbI tasks; the only task is cannot solve is task 16. No other model had achieved this performance before and the DNC completely outperformed the more traditional Long Short Term Memory Neural Network (LSTM).

Nevertheless, the power of the DNC does not come without limitations. First of all, the DNC takes seven days of training on GPUs to reach its state of the art results on the bAbI tasks. The computational resources and time requirements of this model are too great for researchers and data scientists. The DNC takes too much time to train and requires graphics cards which are expensive to own. More importantly, it is not guaranteed that the DNC will converge in one seven day training even by using the best learning parameters. As a result, the DNC on average only passes seven out of the twenty bAbI tasks. The Differentiable Neural Computer is thus expensive and unreliable which has made it a research artifact more than a tool for researchers and data scientist to apply on new problems. Joerg Franke et al. and others recognized these problems and have proposed solutions. This research focuses on the robust and scalable Differentiable Neural Computer (rsDNC) and the Bidirectional robust and scalable Differentiable Neural Computer (BrsDNC) for question answering [24]. These models use less GPU memory, converges with much better accuracy and take half the time to train on the bAbI tasks compared to the normal DNC. The goal of the research in Memory Augmented Neural Networks is to develop better neural networks than Recurrent Neural Networks with an internal memory (see section 2.2). The DNC shows promising results but remains highly unstable and expensive. It is clear that improving the DNCs architecture could lead to new discoveries but further

analysis needs to be done on the improvements which are proposed.

This report investigates the advantages and limitations of the rsDNC and BrsDNC on question answering tasks. This report conducts an ablation study and tests each of the individual components of the architecture to analyse which one contributes the most to the improvements from DNC to BrsDNC. This report also investigates further one of the improvements suggested in the rsDNC: the content based memory unit (CBMU). The rsDNC removes the temporal linkage memory which Alex Graves et al. propose to solve sequence ordering tasks. This report hypothesizes that the rsDNC is not as "robust" as it is advertised to be and will fail on sequence ordering tasks. This report proposes two new tasks in the style of the bAbI tasks to test this hypothesis. The overall aim of this research is thus to provide further analysis on the rsDNC/BrsDNC.

3.2 Background

This section first introduces the most recent research on Differentiable Neural Computers, then explains the robust and scalable DNC and Bidirectional DNC in details, and finishes with a description of the synthetic question answering bAbI dataset.

3.2.1 Related Work

This paragraph summarises the most recent pieces of research on the Differentiable Neural Computer in chronological order. In September 2018, Chan et al. show that the DNC is susceptible to adversarial attacks using the bAbI dataset [33]. They also perform further analysis on the DNC's memory and try to explain why the DNC fails on these attacks. The researchers conclude that the Differentiable Neural Computers architecture is not optimal and that there are opportunities to improve it further. Then, in December 2018, Csordas and Schmidhuber propose an improved version of the DNC with a relative improvement of 43% in *Improved Addressing in the Differentiable Neural Computer* [32]. They identify that the DNC has three main problems. First, it lacks a key-value separation mechanism which affects the content based addressing mechanism. Secondly, they point out how the DNC's de-allocation of memory mechanism fails under certain condition. Lastly, they write that the temporal linkage matrix accumulates noise from the write weighting over time which impacts negatively its performance. To solve these issues they propose a new improved architecture with masked content based addressing, de-allocation and content based content lookup and sharpness of temporal linkage matrix distributions. Lastly, Hung Le et al. introduce another improved version of the DNC's memory in March 2019 [34]. The authors argue that the DNC does not make effective use of the short term memory of its controller unit and hypothesize that doing so would improve the performance. As a result, they introduce Cached Uniform Writing and claim the state of the art on sequence modeling tasks like the copy and repeat copy tasks.

3.2.2 rsDNC and BrsDNC

This section describes the robust and scalable Differentiable Neural Computer (rsDNC) and the Bidirectional Differentiable Neural Computer (BrsDNC) introduced by Joerg Franke et al. [24]. The authors of the rsDNC propose their models as solutions to the poor mean performance of the DNC model. They describe the results they obtained from performing an analysis of the DNC model on the question answering bAbI

dataset. In this analysis they identify three main problems of the DNC. First, the DNC has twice the memory size requirements of a LSTM making the DNC long to train and needing expensive computational resources. This comes from the temporal linkage matrix which makes up 50% of the memory consumption of the DNC. Secondly, the average DNC results have a high variance. Lastly, they point out that the DNC does not converge correctly when it starts to learn to disregard the memory unit in its output. The changes they propose are respectively the content based memory unit (CBMU), layer normalization, and Bypass Dropout. The authors also propose to add a second controller unit to make the DNC similar to a Bidirectional Recurrent Neural Network. This section explains each of these improvements in the exact order above. The rsDNC is a DNC with CBMU, Normalization and Bypass Dropout. The BrsDNC is a rsDNC with Bidirectional architecture.

CBMU. The authors of the rsDNC and BrsDNC propose to remove the backward and forward weighting in the read weighting calculations and remove the temporal linkage matrix. They find in their experiments that the DNC does not use the temporal linkage matrix when it solves the bAbI tasks. They say the DNC does not need the temporal linkage matrix for question answering and decide to improve the memory consumption of their model by removing it. They call this new memory unit the content based memory unit (CBMU). The read weighting of the CBMU is then defined as:

$$\boldsymbol{w}_t^{r,i} = \boldsymbol{\pi}_t^i \boldsymbol{c}_t^{r,i}$$

DNC Normalization. The second improvement is the introduction of a normalization technique. This is to solve the high variance in results on the bAbI tasks between different runs. This technique is called Layer Normalization (LN) and it is applied to the memory unit and the controller unit. For the controller, the layer normalization is applied after the weighting of the controller output $\boldsymbol{v}_t$. For the memory unit, LN is applied to the gates, vectors and the keys separately. Layer normalization is defined as:

$$LN(\boldsymbol{x}_t) = \boldsymbol{g} \circ \frac{\boldsymbol{x}_t - \mu_t}{\sqrt{\sigma_t^2 + \epsilon}} + \boldsymbol{b}$$

where $\boldsymbol{g}$ and $\boldsymbol{b}$ are trainable weights, and μ_t and σ_t are the mean and variance of $\boldsymbol{x}_t$ i.e. the gates, vectors, and keys. The word "Norm" is used in the report to refer to this mechanism.

Bypass Dropout. The rsDNC introduces a bypass dropout mechanism. This is to force the model to learn to use its memory more than the controller unit. Below is the original, DNC output vector at time step t which adds outputs from the controller unit and the read heads:

$$\boldsymbol{y}_t = \boldsymbol{v}_t + W_r[\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R]$$

As described in section 1.5.1, $\boldsymbol{y}_t$ is composed of the controller output $\boldsymbol{v}_t$ and memory read output from the read heads $\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R$. The bypass dropout mechanism then works as follows:

$$\boldsymbol{b}_t \sim \text{Bernoulli}(p)$$

$$\boldsymbol{y}_t = \boldsymbol{v}_t \circ \boldsymbol{b}_t + W_r[\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R]$$

p is the dropout probability and is set to a default value of $p = 0.2$ by the authors of the BrsDNC. $\boldsymbol{b}_t \in \{0, 1\}$ is then a Bernoulli vector which regularizes the signal flow coming from the controller unit. The $\boldsymbol{b}_t$ is multiplied to $\boldsymbol{v}_t$. This mechanism is only used during training and the DNC output reverts back to its normal form during testing. The word "Bypass" is used in the report to refer to this mechanism.

Bidirectional DNC. The original DNC has one controller unit which extracts information from the previous time step and concatenates it to input from the current timestep. In the BrsDNC, the authors propose a second controller unit which extracts information from the next time step. This idea comes from the Bidirectional Recurrent Neural Networks (BRNN) introduced by Schuster and Paliwal. The Bidirectional architecture allows for a better extraction of context [31]. The original controller unit and the new controller unit are respectively called forward (fw) and backward (bw) controller units. Both controller units are LSTMs and both have the same structure as the LSTM described in 1.5.1 which outputs an output vector $\boldsymbol{v}_t$ and interface vector $\boldsymbol{\xi}_t$. The difference is that the backward unit at time step t concatenates its inputs with the output from the backward unit in time step $t + 1$. Additionally, the backward unit inputs do not include reads vectors. From 1.5.1, the forward controller unit is:

$$\boldsymbol{h}_t^{fw} = LSTM([\boldsymbol{\chi}_t, \boldsymbol{h}_{t-1}^{fw}])$$

$$\boldsymbol{v}_t^{fw} = W_y^{fw}[\boldsymbol{h}_t^{fw,1}; ...; \boldsymbol{h}_t^{fw,L}]$$

$$\boldsymbol{\xi}_t^{fw} = W_x^{fw}[\boldsymbol{h}_t^{fw,1}; ...; \boldsymbol{h}_t^{fw,L}]$$

where $\boldsymbol{\chi}_t = [\boldsymbol{x}_t; \boldsymbol{r}_{t-1}^1; ...; \boldsymbol{r}_{t-1}^R]$. The backward controller unit is defined by:

$$\boldsymbol{h}_t^{bw} = LSTM([\boldsymbol{x}_t, \boldsymbol{h}_{t+1}^{bw}])$$

$$\boldsymbol{v}_t^{bw} = W_y^{bw}[\boldsymbol{h}_t^{bw,1}; ...; \boldsymbol{h}_t^{bw,L}]$$

$$\boldsymbol{\xi}_t^{bw} = W_x^{bw}[\boldsymbol{h}_t^{bw,1}; ...; \boldsymbol{h}_t^{bw,L}]$$

The memory unit know receives a concatenation of the $\boldsymbol{\xi}_t^{fw}$ and $\boldsymbol{\xi}_t^{bw}$. The output of the DNC is also changed to incorporate the backward controller output.

$$\boldsymbol{y}_t = \boldsymbol{v}_t^{bw} + \boldsymbol{v}_t^{fw} + W_r[\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R]$$

To allow the backward controller to compute output of future time steps, the architecture first independently unfolds the backward controller unit on the inputs and then unfolds the forward controller unit and the memory unit together which takes into account the calculated backward controller outputs at each time step.

3.2.3 bAbI Task

The bAbI dataset consists of twenty synthetic question answering tasks [12]. The idea behind the dataset is that "an intelligent dialogue agent" should have a set of basic tasks it needs to answer correctly in order to be considered worthy of being called intelligent. Through a set of simple induction, deduction, reasoning

tasks and more, the authors argue that this dataset puts together the simplest question answering capabilities that any dialogue agent should possess. A human would make zero mistakes solving these tasks. They also propose this dataset to enable researchers to benchmark their neural network architectures against each other and identify their failures.

This dataset consists in total of 20 tasks. Two tasks are never the same in terms of complexity, context and type of logic. The tasks are written in the form of stories where the agent can be asked any number of questions in the middle or at the end. Solving a task is considered to achieve less than 5% error rate per tasks when solving the test dataset. An example of a one question story is figure 3.1. It seems obvious to a

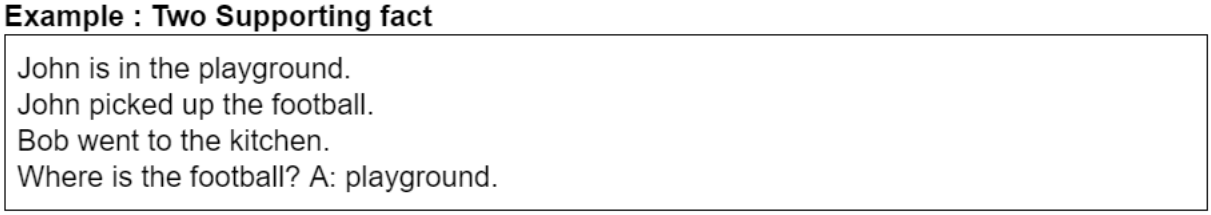

Figure 3.1: Example bAbI task 2

human how to solve this task. If a human where to ask this question to an intelligent dialogue system, the human would expect it to answer correctly. Prior to the invention of this dataset, there was no other dataset with similar complexity and ideology. When this dataset was released, many existing learning systems could not solve them. In the following years, researchers developed models like the Differentiable Neural Network (DNC) or the Recurrent Entity Network (EntNet) [13] which achieved new state of the art results and solve the majority of the tasks.

3.3 Method

The aim of this research is to better understand the BrsDNC and identify its strengths and weaknesses. The hypotheses sub-section introduces two hypotheses and the method used to test them. This is followed by an explanation of the dataset used and how bAbI tasks where presented as inputs to the DNC, a description of the two new tasks developed in this project and finally an overview of the technological frameworks used.

3.3.1 Hypotheses

This report makes two hypotheses.

1. Each component of the rsDNC and BrsDNC improves the mean bAbI tasks results in different ways.

In the original paper, the authors tested certain combinations of their architecture on the bAbI dataset and reported the changes in the mean word error rate on all the tasks combined. The combinations tested in the paper are: DNC with content-based memory unit, DNC with content-based memory unit and normalization, DNC with content-based memory unit and bypass dropout. The authors do not include the mean results of these different architectures on each task of the 20 bAbI task. This hypothesis is put forward this hypothesis because it is not clear how certain components work better on certain tasks and worst on some others.

2. The rsDNC with its content based memory unit cannot solve question answering tasks which involve remembering the order in sequences. The DNC which uses a temporal linkage matrix can.

The authors of the rsDNC suggest that the bAbI dataset results they obtained with the model imply that the temporal memory linkage mechanism is not important for QA tasks and name their paper with the following title *Robust and Scalable Differentiable Neural Computer for Question Answering.* In the original DeepMind paper, the authors of the Differential Neural Computer write that the temporal memory linkage mechanism would be useful when a sequence of instructions must be recorded and retrieved in order. The bAbI dataset does not feature any tasks where the answer is found by having the ability to remember sequences in order. As result, the claim made by the BrsDNC authors is not justified and correctly tested. This hypothesis is put forward because the temporal linkage matrix seems to be an important part of the DNC's functioning (section 2.3).

To test the first hypothesis, an ablation study is done on the rsDNC and BrsDNC by testing each of the individual components on top of the DNC architecture. Each of these tests are done on a full training over the entire bAbI dataset. The exact tested models are: DNC with normalization, DNC with bypass dropout, DNC with bidirectional controllers, and DNC with content-based memory unit. Each model is trained five times on the entire bAbI dataset. Each run is done over five days. From these experiments, the mean test WER on all tasks of each model over the five runs, the evolution of the mean validation WER on all tasks of each model, and the average WER on each task of each model are reported. Following the method from the bAbI paper [12], a model is considered to have passed a task if it gets a word error rate per task lower than 5%.

To test the second hypothesis, two new QA tasks are created. These task are explained in the New Task sub-section below. Then, four rsDNC and four DNC are trained on the new extended dataset to evaluate and compare the mean performance of each model on the new tasks. Each run is done over three days and twelve hours. For each model, two metrics are reported: 1) the mean evolution of the word error rate on the new tasks' validation sets at every training epoch 2) the mean word error rate on each of the new task's test set at the end of the training.

3.3.2 Dataset and Problem Specification

The bAbI dataset used was collected from The Sperm Whale [1]. The version used for training is the english 10k version. This dataset is also the one used in the DNC and BrsDNC papers. The dataset comes partitioned into training and test sets for each task. Each training set contains a total a 10k questions and each test set contains 1k questions. The bAbI tasks, as described before, are stories and may contain multiple questions for one story. There are 159 unique words when all the stories of all the 20 tasks are combined including the punctuation symbols '.', '?' and '-'. Each word is represented as a 159 one-hot vector. The X vector fed to the network is a one-hot encoded tokenized version of the story with every answer to questions replaced by the symbol '-'. For example, a one-hot encoded version of the following story (figure 3.2) would be fed to the network.

The Y vector which the networks is given in order to compute the loss is an equivalent sized vector with the symbol '-' everywhere except for when an answer is required. The Y vector of the previous example (figure 3.2) is figure 3.3.

Example : X vector fed to network

```
Sandra travelled to the office.
Fred is no longer in the office.
Is Fred in the office? -
Is Sandra in the office? -
```

Figure 3.2: X vector

Example : Y vector fed to network to compute the loss

```
-  -  -  -  -  -
-  -  -  -  -  -  -  -
-  -  -  -  -  -  no
-  -  -  -  -  -  yes
```

Figure 3.3: Y Vector

The network is trained to minimize the cross entropy of the softmax distribution of the output at the time steps where an answer is required. The most probable word for each answer is selected as the answer. The combined loss for a story, which is the sum of the losses for each of the answers of a story, is obtained by applying a mask to the total output of the network. This mask is built from the Y vector of every story.

Similar to the rsDNC paper [4], the networks are evaluated using the per-task 'word error rate' or WER (the fraction of incorrectly answered words). During training, ten percent of the training set is used as a validation set. The pipeline described above is similar to what is done in the original paper.

An adjusted dataset was used to train the models. The DNC and rsDNC use a lot of GPU memory and take a significant amount of days to train even on a GPU. Indeed, in order to carry out their experiments in both papers, the two sets of authors used a Tesla K80 [2] and train their models for up to 9 days in order to achieve their results. The available GPUs for this project had a lot less memory and so a max story sequence length of the training/testing data had to established. The max sequence here refers the max story length fed to the network. Through trial and error, the max sequence length of 450 words was picked as the limit for one story. This adjustment removes 1553 stories from the original 62991 stories in the dataset. This change is minor and does not impacts the generality of the results that were found. The exact specification of the dataset with this max sequence length limit can be found in Appendix B.

3.3.3 New Task

This paper proposes two new tasks to test the second hypothesis. To answer these tasks the person or algorithm answering needs to remember the order in which words are said in the sentence to successfully answer the question. Another design requirement is that the addition of these tasks should not significantly increase the vocabulary length. This requirement is needed because of the limited computing resources available for this project. As a result, the tasks built have a similar style to the original 20 bAbI tasks. The

Example Task 21: Listening to one specific person

Focus on Yann.
Yann is saying that Daniel went back to the kitchen.
Winona is saying that Mary and Sandra journeyed to the garden.
John is saying that Daniel and Mary went back to the garden.
Brian is saying that then he moved to the bedroom.
What did Yann say ? Daniel went back to the kitchen

Figure 3.4: New task number 1

first task is based on a real life scenario where multiple people are talking one at a time and the person or algorithm is asked to focus only on one person. Figure 3.4 shows an example story. To get the answer right the respondent needs to repeat every word that the person said after the keywords: "saying that". Additionally, the repeated words need to be in the same order as they were originally. This task is different than any other bAbI task in the original dataset because the answer is longer than the previous maximum of two words. The maximum answer length for this task is eight words.

The vocabulary that this task introduces is very little compared to task 22. The only additions are the words: focus, saying and that. This is because the phrases that the actors in the scenario are saying come from the original 20 tasks of the bAbI dataset. This limits the increase in vocabulary size.

It is also interesting to note that this task is similar to the Copy Task introduced by Alex Graves et al. in the Neural Turing Machines paper. The original purpose of the Copy Task was to test if the Neural Turing Machine they developed in their paper could "store and recall a long sequence of arbitrary information [11]." The main difference between this new bAbI task and their particular task is that in their task the network is fed an "input sequence of random binary vectors followed by a delimiter flag" and has to repeat the entire input sequence, where in task 22, the network should only repeat a particular part of the input sequence which is fed to it. In order to solve the task, the network needs to understand that the third token in the input sequence is the name of the person that it needs to focus on. Additionally, the network needs to understand that it should repeat every word between the delimiter tokens 'saying that' and '.'. Therefore, it is clear that this task introduces some additional complexity compared the original Copy task. Although the Differentiable Neural Network has already been shown to solve the Copy Task there is not guarantee that it can solve this task. This task pushes the limit of what has been solved by Neural Networks in the domain of question answering.

Example Task 22: Location Reasoning

Greg travelled to Madrid , Rome , Sarajevo , Dublin and London.
Emily travelled to Bern , Berlin , Kiev , Prague and Paris.
Daniel travelled to Lisbon , Amsterdam , Belgrade , London and Copenhagen.
Where did Emily travel to before Prague? Kiev

Figure 3.5: New task number 2

Task 22 (figure 3.5) is different to task 21 in that it is more complex and adds a lot more vocabulary words. The idea of this new task is to remember what city an individual travelled to just before travelling to a particular city. The only way to answer this question correctly is to remember all the cities in order that each person travelled to. In some cases, when the question asks where the person travelled to before the first city he travelled to, the answer is always 'none'. Task 22 introduces some new vocabulary: 16 city names and the comma token i.e. ','. The names and all the other tokens (verbs and connectives) where already present in the vocabulary.

In order to test the second hypothesis, the rsDNC and the DNC are trained on the entire bAbI dataset plus these additional two new tasks. This extended bAbI dataset has a vocabulary size of 185 words including the punctuation words (more information in section 4.4).

3.3.4 Framework

In the experiment section, the original model code developed by each paper was used. These models are implemented in Tensorflow[3] and can be found on the following github repositories: DNC[5] , BrsDNC[4]. The input pipeline to train the DNC on the bAbI tasks for this experiment was entirely built from scratch (see section 4.3). The entire pipeline is built in Python.

To test the first and second hypothesis, the model implementation from the rsDNC paper was used as a base. This implementation is a different from the original implementation by Deepmind in that it includes a batch implementation. The batch implementation is kept for this project and a batch size of 32 is used. From their model implementation, entire sections were added to the code to obtain the results to test the two hypotheses (see section 4.3). All the additional code written is also built using Tensorflow and Python.

3.4 Experiments

Experiments were conducted to try to reproduce the original results obtained by each paper on the bAbI tasks. Two BrsDNC and two DNC where trained for five days on the full bAbI dataset. Below, the best mean word error rate of each network on the full dataset is reported. The mean word error rate is obtained by averaging the word error rate of task together.

Model	Mean Best WER
DNC	12.01
BrsDNC	3.10
DNC paper(5)	3.8
BrsDNC paper(4)	2.8

Figure 3.6: Experiment WER results

The best result obtained for the BrsDNC is close to the BrsDNC papers best results. On the contrary, the

best DNC result obtained is far from the best results obtained by the authors of the paper. The interesting insight from these results is that the BrsDNC seems to converge a lot better like the authors of the paper are claiming it to. In just two runs, BrsDNC obtained passable results. These are the expected results given the four improvements suggested in the paper. The conclusion of this experiment for the DNC is similar to Joerg Franke and al., it seems that the architecture of the DNC is such that it takes too much time to converge and rarely converges to states which give passable results on the bAbI dataset.

3.5 Results

This section reports the results for each hypothesis.

3.5.1 Hypothesis 1

This sub-section reports the mean WER results of the different models on the entire bAbi dataset, shows the evolution of the WER of each model over the number of training iterations and finally identifies which component of the architecture of the BrsDNC contributes the most to the improvements from the DNC to the BrsDNC. The model tested are DNC with normalization (DNC + Norm), DNC with content-based memory unit (DNC + Cbmu), DNC with bypass dropout (DNC + Bypass) and DNC with bidirectional architecture (DNC + Bidirect).

1. Mean word error rate improvement of each model over five runs on all tasks.

The mean WER for each model is calculated at the end of five days on the training set. The score is the average WER of the models on all the tasks in the bAbI test set.

Model	Mean WER
DNC* (23)	16.7 ± 7.6
BrsDNC* (24)	3.2 ± 0.5
DNC + Bidirect	16.2 ± 4.7
DNC + Cbmu	9.8 ± 4.8
DNC + Norm	7.6 ± 2.9
DNC + Bypass	14.4 ± 7.4

WER on testing set

Figure 3.7: WER on testing set

The DNC with normalisation model has the better improvement in mean results for the entire task combined with a percentage word error rate of 7.6 %. It is also the model with the lowest standard error in the average word error rate with a standard error of 2.9. The second best model is the DNC with the content-based memory unit. The model averages 9.8% in word per error rate. The standard error remains high however with a value of 4.8 which makes it similar to the st. error in word error rate of the DNC + Bidirect.

The models which improve the WER results the least are the DNC with the bidirectional LSTM as its controller unit and the DNC with bypass dropout. For the DNC + Bidirect, the mean result is 16.2 % and the standard error is 4.7. The DNC + Bypass obtained a 14.4 % average WER. The standard error is much higher than all the other models with a value of 7.4. Considering the standard errors, these results are not significantly different from the mean results of the original DNC. The mean results of the DNC reported by Alex Grave are 16.7% with a standard error of 7.6.

2. Mean Evolution of each models Word Error Rate on bAbI validation set.

The validation set used to calculate the word error rate here contains all the 20 tasks. The validation set WER is calculated for each model and for every training iterations. Each curve is an average over five training runs of each model. Each training run takes five days. The individual curves of the five runs for each model can be found in Appendix C. The following paragraphs point out the important results which this graph gives (figure 3.8).

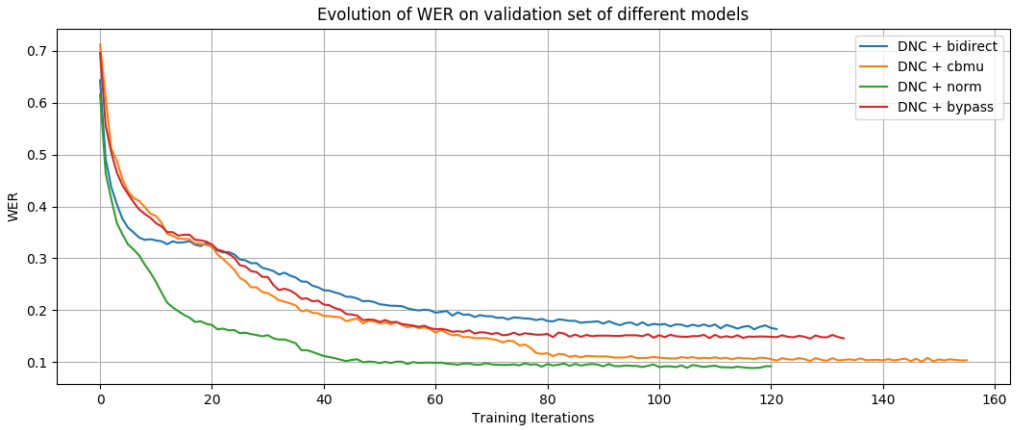

Figure 3.8: Evolution of WER

DNC + Norm. The WER over training iterations of the DNC with normalization has a much smoother validation score evolution over time on average than the curves of the other models. The DNC with normalization also reaches on average a WER of 10% in around 42 training iterations. After passing the 10% WER mark, the curve decreases at an extremely low rate and ends at around 9% after 120 training iterations.

DNC + Bidirect. The average WER curves of the DNC with cbmu, bidirectional LSTM controllers, and bypass dropout all plateau around the 20th iteration when reaching a WER on the validation set of approximately 35%. On the contrary, the average WER of the DNC with normalization does not plateau at 35 % on the 20th iteration but has an average value of 18%. Additionally, before the 20th training iteration, the DNC with a bidirectional LSTM controller unit converges on average faster than the DNC with cbmu and DNC with bypass dropout to the WER on the validation set of 35%. The DNC + Bidirect curve follows the same evolution initially as that of the DNC with normalization in the first five training iterations. After its first plateau at 35% the curve drops again to reach 17% on average at the end of the five days after 120 training iterations.

DNC + Cbmu / DNC + Bypass. The DNC with content-based memory unit and DNC with bypass dropout have similar curves until the 60th training iteration. The validation scores initially decrease at the same rate and then perform more or less the same on average until training epoch 60 where they reach on average a WER of 18%. The average WER on the validation of the DNC with cbmu then then drops further to end at a value of 11% at the end of the five days of training, where that of the DNC with bypass dropout plateaus at 16%. The DNC with cbmu and DNC with bypass dropout on average go through more training iterations than the other two models for the same amount of training time: five days. The DNC with bypass dropout goes through 130 training iterations while the DNC with cbmu does on average 155 training iterations.

3. Average WER on each task of each model.

The table in Figure 3.9 reports the average word error rate of each model on each task of the bAbI dataset. The average WER on each task calculated by the authors of the DNC [23] and the authors of the BrsDNC [24] are represented in the respectively in the left and right most columns. Respecting the method created by the authors of the bAbI tasks [12], a task is solved if the word error rate for a given task achieved by the model is below 5%. The colour coding scheme of the table in Figure 3.9 to differentiate between the WER of each model on each task:

1. Red - the model on average does not pass the task.
2. Green - the model passes the tasks on average
3. **Bold** - the word error rate is the best out of the 4 models (excludes DNC and rsDNC).

On average, the DNC + Norm and DNC + Cbmu give the best tasks results. The worst performing models are the DNC with a bidirectional LSTM for controller unit and DNC with bypass dropout. According to the previous results [23,24], the DNC passes seven tasks, namely task 4, 5, 11, 12, 13, 18, and 20, while the BrsDNC passes on average all tasks except task 16. In the paragraphs below, the general results for each model are summarised in the order of: DNC with normalization, DNC with content-based memory unit, DNC with bypass dropout and DNC with a bidirectional LSTM.

DNC Norm. The DNC with normalization passes on average sixteen out of the twenty tasks. Particularly, it passes on average task 2, 7, 14, and 15 which none of the other models pass. For eleven out of the sixteen tasks that it passes, DNC with normalization achieves the best mean WER out of the four models tested in the experiment. These tasks are tasks 2, 4, 6, 7, 8, 12, 13, 14, 15, 18 and 20. The four that it fails are 3,16, 17 and 19. For these four tasks, none of the other three architectures tested in the experiment seems to solve them on average. The DNC also does not solve them on average. Lastly, for all the tasks except task 16, the DNC with normalization brings improvements in the mean results compared to the DNC.

DNC with cbmu. The DNC with content based memory unit passes on average eleven out of the twenty tasks. For six out of those eleven tasks, the DNC with cbmu obtains the best average WER result. These tasks are tasks 1, 5, 9, 10, 11 and 20. The tasks that it fails are tasks 2, 3, 7, 14, 15, 16, 17, 18 and 19. For seventeen out of all the tasks, the DNC with cbmu improves the average results of the DNC.The tasks that it does not improve on are 16, 17 and 18.

DNC Bypass. The DNC with bypass dropout passes eight out of the twenty task. These tasks are

Tasks *Run*	DNC* *7 days*	DNC + Bidirect *5 days*	DNC + Cbmu *5 days*	DNC + Norm *5 days*	DNC + Bypass *5 days*	BrsDNC* *5 days*
Task 1	9.0 ± 12.6	1.8 ± 1.1	**0.1 ± 0.3**	0.3 ± 0.4	6.9 ± 15.4	0.1 ± 0.1
Task 2	39.2 ± 20.5	50 ± 12.5	15.7 ± 18.9	**4.5 ± 4.1**	19.9 ± 22.9	0.8 ± 0.2
Task 3	39.6 ± 16.4	48.4 ± 5	28 ± 12.4	**27.2 ± 12**	35.8 ± 14.6	2.4 ± 0.6
Task 4	0.4 ± 0.7	0.9 ± 0.7	0.2 ± 0.3	**0.2 ± 0.2**	**0.2 ± 0.2**	0.0 ± 0.0
Task 5	1.5 ± 1.0	1.8 ± 0.6	**1.2 ± 0.5**	1.4 ± 0.3	3.2 ± 4.1	0.7 ± 0.1
Task 6	6.9 ± 7.5	8 ± 5.3	0.5 ± 0.4	**0.4 ± 0.4**	4.4 ± 8.5	0.0 ± 0.0
Task 7	9.8 ± 7.0	18.1 ± 3.8	5.9 ± 3.6	**2.8 ± 1.1**	11.1 ± 6	1.0 ± 0.5
Task 8	5.5 ± 5.9	13.5 ± 7.4	1.5 ± 1.6	**1.2 ± 0.9**	7.3 ± 8.4	0.5 ± 0.3
Task 9	7.7 ± 8.3	7.3 ± 4.1	**0.3 ± 0.2**	0.6 ± 0.7	3.9 ± 8	0.1 ± 0.2
Task 10	9.6 ± 11.4	7.6 ± 4.4	**0.6 ± 0.3**	0.7 ± 0.5	2.1 ± 3	0.0 ± 0.0
Task 11	3.3 ± 5.7	0.3 ± 0.3	**0 ± 0**	0.2 ± 0.3	3.6 ± 8	0.0 ± 0.0
Task 12	5.0 ± 6.3	0.7 ± 0.5	0.6 ± 0.9	**0.1 ± 0.2**	5.4 ± 11.1	0.0 ± 0.1
Task 13	3.1 ± 3.6	0.4 ± 0.4	0.3 ± 0.4	**0.1 ± 0.2**	2.2 ± 4.6	0.0 ± 0.0
Task 14	11.0 ± 7.5	17.5 ± 7.3	6.3 ± 5.5	**3.1 ± 1.7**	8.9 ± 6.2	0.8 ± 0.7
Task 15	27.2 ± 20.1	23.8 ± 16.3	8.8 ± 19.5	**0 ± 0**	45.2 ± 1.7	0.1 ± 0.1
Task 16	53.6 ± 1.9	53.8 ± 1.6	54.1 ± 0.8	**53.7 ± 2.4**	54.8 ± 0.9	52.6 ± 1.6
Task 17	32.4 ± 8.0	29.8 ± 9.4	35 ± 2.1	**12 ± 3.9**	35 ± 3.8	4.8 ± 4.8
Task 18	4.2 ± 1.8	3.9 ± 1.3	5.1 ± 1.4	**2.1 ± 1.2**	5.8 ± 1.1	0.4 ± 0.4
Task 19	64.6 ± 37.4	35.4 ± 27.5	32.3 ± 37.9	41.1 ± 36.8	**32.2 ± 37.6**	0.0 ± 0.0
Task 20	0.0 ± 0.1	1 ± 0.9	**0 ± 0**	**0 ± 0**	**0 ± 0**	0.1 ± 0.1

Figure 3.9: Mean WER on each task for each model

tasks 4, 5, 6, 9, 10, 11, 13 and 20. For task 4 and task 20, it obtains the best results out of the four models tested in this experiment. The tasks that it fails are tasks 1, 2, 4, 7, 8, 12, 14, 15, 16, 17, 18 and 19. Compared to the other models, it is the only model which fails to pass task 1 and 12. For eleven out of the twenty tasks, the DNC with bypass dropout improves the results of the DNC. The tasks it does not improve are 5, 7, 8, 11, 12, 15, 16, 17 and 18.

DNC Bidirectional. The DNC with a bidirectional controller unit passes on average eight out of the twenty tasks. These tasks are tasks 1, 4, 5, 11, 12, 13, 18, 20. In the other twelve tasks that it fails, it fails task 6, 9, and 10 which none of the other models fail. For half the tasks it performs better than the DNC. For tasks 2, 3, 4, 5, 6, 7, 8, 14, 16 and 18, it does not improve the results and performs worse than the DNC.

3.5.2 Hypothesis 2

This section reports the result of the experiment on task 21 and then reports the results on task 22. A model is considered to have solved a task when it achieves lower than 5% WER.

1. Task 21: Listening to one specific person

DNC. On average, for the DNC models trained on the whole dataset, none were able to solve task 21 (less

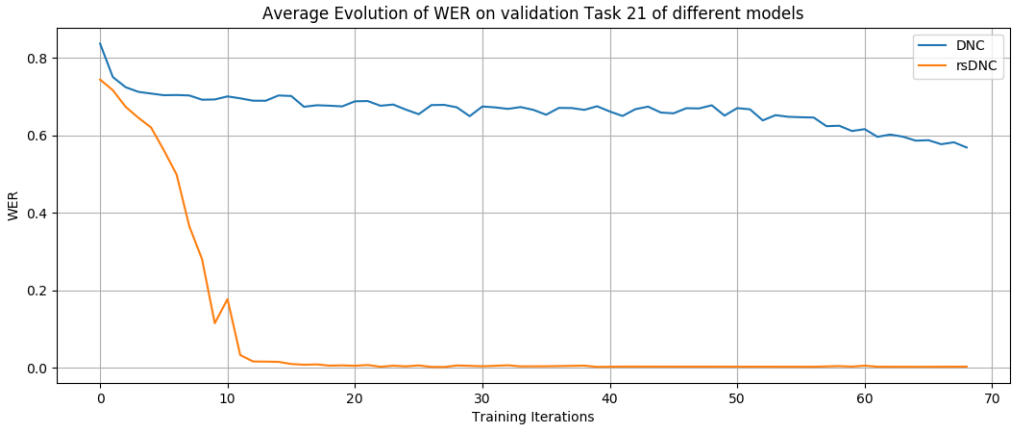

Figure 3.10: Mean WER Evolution of DNC vs rsDNC on task 21

than 5% WER) after the 68 training iteration (figure 3.10). The average progression of the WER on the validation task 21 can be said to be linear with a negative slope. The mean WER reached after the three and a half days of training is 58.8%. Thus, the DNC does not solve this task on average for this time frame.

rsDNC. On the other hand, the four rsDNC trained were all able to solve this task with less than 5% WER each time when tested on the test set after training for three days and a half (figure 3.10). After 12 training iterations, the rsDNC is able to solve the task by reaching 5% WER.

2. Task 22: Location Reasoning

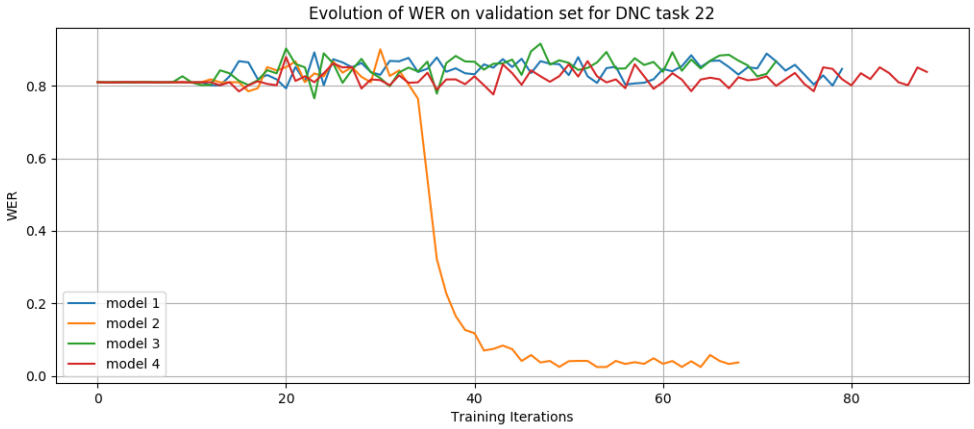

Figure 3.11: Mean WER Evolution of DNC

DNC. On average, for the DNC models trained on the whole dataset, one was able to solve task 22 with 4.8% WER after the 68 training iteration (figure 3.11). This is model 2 showed in the graph above, The mean WER reached after the three and a half days of training is 65.5%. Thus, the DNC does not solve this task on average for this time frame. However, the DNC can solve this task.

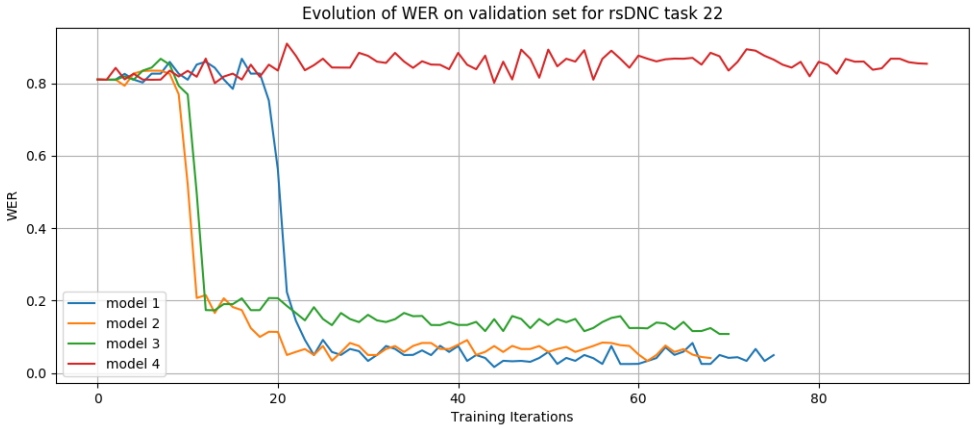

Figure 3.12: Mean WER Evolution of rsDNC

rsDNC. Out of the four rsDNC trained, none were able to solve the task (figure 3.12). Model 1, Model 2, Model 3 and Model 4 achieve respectively 6.8%, 10.8%, 88.1% and 5.5%. On average the rsDNC cannot solve this task with a average WER of 27.8%.

3.6 Discussion

This section discusses the results, interpretations and also weaknesses in the findings.

DNC Normalization. The results obtained show that the DNC with just normalization brings a lot of improvements to the mean results of all tasks compared to the DNC with the other improvements proposed by the BrsDNC. The normalization brings improvements to all the tasks except task 16 where the others all improve smaller sets of tasks. Furthermore, the normalization also drastically improves number of training iterations needed for a faster convergence to passable results as seen with the results on the evolution of the mean word error rate on the validation set (figure 3.8). In the paper *Self-Normalizing Neural Networks* [25], which introduces the concept of normalization for neural networks, the authors suggest that adding normalization to neural network architectures works best on "very deep networks". A possible interpretation of the results is that the DNC is a "very deep network" which would explain why normalization is so powerful.

Bidirectional DNC. The bidirectional LSTM does not lead to significantly better results than the DNC with a simple forward controller. The only difference is that DNC+Bidirect passes task 1 that the DNC with simple LSTM controller does not pass. According to research on the Bidirectional Recurrent Neural Network, this architecture gives better results than an unidirectional architecture on tasks which are resolved with contextual information [26]. Task 1 is the single supporting fact question and the easiest question out of the three supporting facts questions (1-3). It is a good example of a question which can be answered by extracting simple contextual information which could explain why the Bidirectional DNC does better here. The other tasks which the DNC and Birectional DNC do not pass on average possibly need more context and the answer is less obvious than the others.

Improving Memory Usage. The CBMU and the Bypass dropout are introduced to improve the memory

use of the DNC (24). First of all, this paper finds contradicting results to the results of the original paper for the DNC with CBMU. The original paper finds an average WER of 17.6 $\pm$ 1.6 while here an average WER of 9.8 $\pm$ 4.8 is found. This is can be viewed as one of the weakness of this research project and their research. More runs for the average should be done to improve the results instead of just five. Nevertheless, the difference in terms of training iterations per training time that the CBMU brings is confirmed by figure 3.8. It seems that the CBMU allows for more training iterations in the same amount of time and thus allows the DNC with CBMU to be exposed to more training data and learn from more training questions. Another interesting finding is that the CBMU and the Bypass Dropout combined with the DNC are able to pass on average task 6, 9 and 10 which the DNC alone does not pass on average. It could suggest that these tasks are solved on average with a better memory usage.

BrsDNC improvements. From the results in figure 3.9 it clear that the normalization of the parameters of the BrsDNC is the improvement which has the biggest impact on results. However, the experiments do not show conclusive results on which architecture is responsible for the improvements in the results for tasks 3, 17 and 19. For these tasks, it is probable that a specific combination of the architectures is the main reason for the improvement.

Task 22 disparity in WER. The results obtained on this task by the rsDNC and the DNC are particularly odd. Either the models seem to get a WER of around 80% or a WER of 10% and the transition from the two WER happens in one training iteration. A possible interpretation is to think about how a human would solve this task. The most basic way a human would solve this task is by first finding the set of cities which the person featured in the question travelled to. Then the person would find the city mentioned in the question and at this point, look at the city just before. If there is a city the person answering the question would answer that city and otherwise would answer none. When the DNC is learning this task, it is possible that the DNC has a problem learning by how much it should go back once it has matched the city and the person in the question. The DNC will thus always get it wrong until it is able to understand that it needs to fetch memory location x. At this point it will get most of the answer right, suddenly decreasing the WER rate. Nevertheless, this remains a hypothesis.

rsDNC CBMU limitations. The goal of hypothesis two was to test the CBMU of the rsDNC for question answering and if it cannot solve certain tasks because it does not have the temporal linkage memory. The results on task 21 clearly goes against the hypothesis and show that the rsDNC is more robust than the DNC even though the rsDNC has no temporal link memory matrix. The results on task 22 again show that the rsDNC is more robust on average than the DNC by getting better mean results on new tasks. Nevertheless, no rsDNC model trained on the bAbI dataset with the two new tasks was able to pass task 22 while one DNC passed the task. The last results mentioned here possibly identify some limitations in the rsDNC compared to the DNC which could come from some limitations of the rsDNCs CBMU.

Weakness in method. A major problem about the DNC is that it takes so much time to train on the bAbI dataset before achieving passable results. This is the problem that the authors of the BrsDNC have tackled and found a solution to by building their better version of the model. The experiments were constrained by the training time needed to train each model. Each run was done over five days and considering the limited computing resources available limited the amount of runs per model. Thus, only 5 runs for each model were

used to calculate the mean results shown in the results of hypothesis one. There is no correct number to do the averaging but running more of each models on the dataset could have obtained better results. Another weakness of the research is the a word error rate lower than 5% used to consider a model to have solved a task. This came from the original method described in the bAbI paper [12]. It is unclear from there method why 5% was picked and not 4% or any other relatively low number. Considering that a human would make no mistake on these tasks, a better judging criteria would be to consider 0% word error, the task solved rate.

Limitations of bAbI dataset. The goal of hypothesis 2 was to test the CBMU of the rsDNC for Question Answering. This study focused on the bAbI dataset and the tasks developed are the same style to the bAbI dataset. However, it is important to consider that synthetic tasks have limitations and are potentially not the best representation of the general question answering domain. The main limitation is that these tasks are written in a ideal way and do not ready include the possible nuance questions that general question answering could involve. The bAbI tasks do not show that an architecture is good at question answering but that an architecture is able to solve multi-hop problems. Synthetic tasks best use is to unit test an architecture for certain capabilities but not to prove it is good at question answering. Therefore, the two new tasks proposed in this report should be considered as unit tests for testing the ability of a network to remember the order in sequences.

Chapter 4

Technical Details

This chapter explains the zip folder containing the code files used during this project. This section shows where to find the code for the first sets of experiments which were done before building intuition for the project, then gives some details on GPU usage, followed by the research experiments, some details on the code files for the new tasks and a reflection on the technical challenges. In general, this section does not go into too much details about how the code actually works but explain the purpose the files. All the code files are adequately commented and the comments are written in a clear and concise way.

4.1 First Experiments

Many initial experiments were first conducted to get some intuition on what was feasible during the time limitations of this project. In the zip folder, all the experiments can be found in the *experiments/initial_experiments/ folder*. The files in this folder are the following:

1. *experiment_1.py*, . . . ,*experiment_6.py*: These are all the experiments that were run to test the different scores of the DNC on task 1, 4 and 9. The results and outcome of each experiment is outline at the top of each file. tasks/ : These files take care of parsing, cleaning, and encoding the dataset.
2. *dnc_tester.py*: class used to test trained DNC models.
3. *dnc_trainer.py*: class which trains the DNC model defined by Deepmind on any dataset. This class also enables the user to choose which GPU he can run it on.
4. *dnc_trainer_v2.py*: Optimized version of the *dnc_trainer.py*

Another purpose of these experiments was to get a better understanding of Tensorflow and the model implementation built by Alex Graves et al.[5] and Joerg Franke et al. [4]. These experiments were carried out until the end of January before the interim report was due. The insight obtained from these experiments was that the DNC takes a lot of time to train to get good results. The first tests conducted were on just on tasks 1, 4 and 9. This was because doing it on more tasks needed a GPU with a larger memory.

4.2 GPU

The Differentiable Neural Computer takes a lot of time to train which using a GPU to accelerate the process is essential. A big part of this project was to get access to enough GPUs to carry out the experiments and also to design a good plan to carry out the experiments. The first experiments (section 4.1) were all carried out on the UCL server Blaze from which only has one GPU. The problem was that more GPUs were needed to perform five runs of each model since each run needs one GPU for five days because of the huge GPU memory requirement of one run. Doing this on one GPU from Blaze meant to perform the experiments in 5*4*5 or 100 days. Furthermore, if at any point the server crashed it would add an additional 5 days and could completely jeopardize the project. In the end, with access to the UCL GPU CS cluster with approximately 16 GPUs on the 15th of february it was possible to carry out some experiments. Only four GPUs were used at a time though for the experiments to leave some for other students and professors also sharing the cluster. The exact planning was the following:

Experiment	Runs	Outcome
16/02-20/02	4 * DNC+Norm	Success
21/02-25/02	4 * DNC+Bidirect	Failed (server crashed)
26/02-02/03	4 * DNC+CBMU	Success
03/03-07/03	4 * DNC+Bypass	Success
08/03-12/03	1 of each model above	Success
13/03-17/03	4 * DNC+Bidirect	Success
18/03-22/03	4 * rsDNC new tasks	Success
23/03-27/03	4 * DNC new tasks	Success

Figure 4.1: Experiment Plan

Most of the experiments went smoothly except for the server which crashed on the 23$^{\text{rd}}$ of February. This crash was only discovered on the 25$^{\text{th}}$ unfortunately. A total of seven days was lost because of this crash; 2 days from the 23$^{\text{rd}}$ to the 25$^{\text{th}}$ and 5 days because the experiments were relaunches during the week of the 13$^{\text{th}}$ of March until the 17$^{\text{th}}$. To anticipate any other servers crashes after this error, the code was adapted to send update messages from the server to a mobile phone app every 5 hours to give on update on the status of the runs. The messaging library used to write this up is PushSafer [28]. The Python Class called *notification.py* was built on top of the API provided by PushSafer. The class is instantiated at the beginning of every training run and then sends updates every five hours. This class features in the *lib/utils* folder.

4.3 Training Models

This secion clarifies all the code present in *experiments/research_experiments/* folder. This project used the code model implementation from the Alex Graves et al.[5] and Joerg Franke et al. [4] Github repositories as a base for the experiments given the complexity of the task. The most important steps were to build the training pipeline for the models and to modify the BrsDNC source code to fit the experiments. Table C.4 shows what code each experiments used. It also points out where modifications where needed to fit the experiments. The details of all the modification to the BrsDNC source code can be found the *memory_augmented_networks* folder.

The entire training pipeline can be found in the *experiments/research_ experiments/training_pipeline* folder.

	Experiments	Tensorflow implementation used	Modifications
1	DNC (section 2.4)	Deepmind	Training Pipeline
2	BrsDNC (section 2.4)	BrsDNC	None
3	DNC + CBMU, DNC+Norm, DNC+CBMU, DNC+Bypass	BrsDNC	Training Pipeline and Modification of the source code to enable testing of individual additions to the DNC.
4	DNC and rsDNC on the new dataset with additional tasks	BrsDNC	Re-wrote the entire training pipeline. Modification of source code.

Figure 4.2: Source code usage

In summary, the training pipeline goes through the entire bAbI dataset, builds a dictionary for the entire vocabulary, dissects all the tasks into the respective stories and finally encodes the stories. For more details, see the code. All the tensorflow trained models with the learnt weights can be found in the *experiments/research_experiments/training_pipeline* folder. All the BrsDNC source code modifications are found in the *memory_augmented_networks/rsdnc/* folder.

4.4 New Tasks

The code files for the two new tasks developed in this project are found inside the *new_tasks* folder. To generate task 21 (Listening to one specific person) and task 22 (Location Reasoning) the files are respectively *generate_listening_one_person.py* and *generate_location_reasoning.py*. A complete training and testing sets are also included in the folder for potential future research.

4.5 Reflection on Technical Challenges

This project was a great opportunity to get first hand experience with the machine learning library Tensorflow and also learn about planning scientific experiments. First of all, I had no experience of neural networks before the start of this project. This also meant that I had no experience with Tensorflow. Tensorflow has a steep learning curve. The problem I encountered using Tensorflow was that I did not expect the great amount of background knowledge about Neural Networks and Machine Learning that the library assumes. Tensorflow creates shortcuts for many operations, like its shortcuts for calculating the cross entropy between distributions, which make the programming easier only once you know what these operations mean and are used for in the Neural Network training pipeline. Additionally, debugging Tensorflow code is a challenge of its own. One of the big challenges in the project was planning five day long training runs for each model. What happened multiple times was that I would get unexpected errors from Tensorflow code after two days of training. Since Tensorflow errors are not clear and do not give enough details, I lost a lot of time fixing these errors. In summary, I overestimated my capacity to learn this library quickly and thought I could dive directly into the problem. A better approach would have been to solve easier problems first before the one in this project.

The other important skill this project taught me was time management. Running five day long experiments on remote servers was more challenging than expected. First of all, it took a lot of time to get access

to servers with enough computing power to perform my experiments. Second of all, it was difficult to plan my experiments in time in order to have all my results in time. The reason for this was because the training time for each run was extremely long compared to more basic machine learning tasks. I lost a lot of time waiting for results before moving on to new experiments. Looking back to the start of the project, I should have requested access to the GPU cluster immediately and not lose any time trying to optimize my code to improve the training time. For a long time, I thought that my problem was coming from using Tensorflow incorrectly and I re-wrote the training pipeline multiple times when in fact the problem was simply that the DNC is a model which takes a lot of time to train. If I had thought about these issues at the beginning of the project I could have done things differently, although it would have been hard to foresee all of these challenges.

Chapter 5

Conclusion

This chapter summarises the main ideas of the report. The first section discusses the main points made in the Background chapter. The Research section reports the conclusion and future works of the Research chapter. Finally, the epilogue reflects broadly on Memory Augmented Neural Network research and concludes the report.

5.1 Background

The Background chapter introduced the importance of solving human memory in order to one day pass the Turing test (section 2.1). Memory enables humans to think logically and memory gives humans a sense of identity. This introduction was followed by an explanation of the history Memory Augmented Neural Network which led to the development of the Differentiable Neural Computer (DNC). Recurrent Neural Network architectures were first introduced as an alternative to feed-forward neural networks but soon became a major research tool with LSTMs. This chapter then introduced the general Memory Augmented Neural Network (MANN) architecture and gave an explanation of the concepts behind the Neural Turing Machine - the base architecture for the DNC. LSTM and other Recurrent Neural Networks with internal memories have problems remembering long term dependencies. MANNs which use an external memory do not have this problem. Finally, this chapter ended with a detailed description of the Differentiable Neural Computer's mechanisms and the human memory concepts it uses. Every detail of the DNC's architecture is given and an analogy is made between the human brain and the DNC. Overall, this chapter served as the backbone research needed to understand the research in chapter 3.

5.2 Research

This section contains two mains parts: conclusion of the research, and future works. This follows from the research conducted in chapter 3.

5.2.1 Research Conclusion

The research project in this paper analyses each individual part of the BrsDNC architecture to understand the improvements in mean results from the DNC to the BrsDNC on each bAbI task. The results of the research suggests that the main component responsible for the BrsDNC improvements is the normalization

of the memory and controller unit layers, followed by the content based memory unit, the bypass dropout and finally the bidirectional architecture. The normalization gives by far the best improvements out of the four and leads to improvements in mean results on nineteen of the twenty tasks and shortens the convergence time.

The paper also proposed that the rsDNC which has a content based memory unit (CBMU) could perform worse on tasks which involved remembering the order in sequences compared to the DNC which has a temporal linkage matrix. To test limits of the rsDNC's CBMU compared to the DNC with temporal linkage matrix, two new synthetic question answering tasks are created to test the sequence ordering capabilities of each network. No other synthetic tasks seem to test this particular capability of neural networks at this level of complexity. These tasks can be used by future researchers to test the capability of their neural networks to remember the order in sequences. The results on both tasks confirm the findings by the authors of the rsDNC in that the rsDNC seems to be more robust than the DNC although it does not have the temporal linkage memory matrix. The mean results of the rsDNC on both tasks is on average much lower than that of the rsDNC. Nevertheless, the impact of removing the temporal linkage memory unit in the rsDNC's content based memory unit remains unclear. On task 22, no rsDNC obtained less than 5% word error rate on the test set while one DNC was able to get lower than 5% word error rate on this task. This result could be coming from the temporal memory linkage matrix in the DNC. It suggests that the impact of removing the temporal linkage matrix should be studied further.

5.2.2 Future Works

This sub-section identifies multiple possible research directions: research on rsDNC, research on the DNC and research on synthetic tasks for question answering. Each research direction is evaluated based on its possible results and its limitations.

The rsDNC does not just improve the DNC, more importantly, it points out certain limitations in its design. When Alex Graves et al. published their paper introducing the Differentiable Neural Computer they do not really justify the temporal memory matrix. The only thing presented is a graph showing the convergence behaviour of the DNC with temporal linkage memory unit compared to that of the DNC without temporal linkage memory unit on a repeat copy task with 100 tokens with no explanations of the experiments carried out. Through this diagram, they seem to suggest that the temporal linkage memory unit is key to the DNC's architecture. However, considering the research which led to the rsDNC and the research in this paper, its is clear that the temporal linkage memory unit has limitations. It seems that the temporal linkage matrix does not enable the DNC to remember the order of small sequences. Task 22's biggest sequence to remember is of size 8. In contrast, Deepmind's experiment tests sequences with lengths of size 100. Thus, it is possible that there is a threshold between sequences of size 8 and 100 where the temporal linkage matrix would be needed. This threshold might have a relationship with the size of the memory as well. A possible research direction is to identify this threshold and identify any relationship between memory size and sequence size. Research in this area would also explain better why the rsDNC performs better than the DNC on the bAbI tasks. Another interesting area of research would be to tune the Bypass Dropout probability introduced by the rsDNC. This could potentially lead to improvements in the scalability of the rsDNC. However, considering that the DNC with Bypass Dropout did not seem to show major improvements on average on the bAbI tasks compared to the Normalization or the CBMU, this research direction might not lead to significant results.

More research could also be conducted on the DNC in general. It is clear that the DNC takes too much time to train before being able to achieve passable results. Research could be done to try and improve the training time of the DNC other than adding normalization and removing the temporal linkage matrix. This would enable the wider community to use the DNC on more tasks rather than leaving the use of this architecture for the few researchers which have accessed to a lot of computing power. Another interesting possible direction of research is the addition of more than one external memory. This idea would be similar to the idea of a cache in modern computers which enables a computer to access memory in a faster way through seeing memory as multiple memory units with different hierarchies rather than just one memory block. This could potentially improve the training time and also inference time of the DNC, and in general, Memory Augmented Neural Networks.

In section 3.6, the relevance of solving synthetic tasks as a step towards solving question answering in general is challenged. The bAbI tasks have limitations and are more unit tests showing the capability of a network to learn logical connections in sentences rather than tasks proving the ability of networks to answer questions in general. There is a lack of research into the complexity of question answering tasks. It would be interesting to create rules to rank all possible question answering tasks imaginable into complexity categories. For example, task 1 of the bAbI tasks could be considered to be of complexity 1 because it only requires a network to learn one logical dependency in a sentence while task 16 could be said to have complexity 10 because four logical connections between words are required to be able to answer questions. This idea results from the problems encountered when designing the new bAbI tasks. It was hard to judge the difficulty of answering the new tasks developed compared to solving the original tasks. Having a set of rules to identify the complexity of every question answering tasks within a limited range would allow for a better and more general understanding of the limitations of the current natural language processing neural networks. Nevertheless, this might be extremely difficult to do or not be doable at all. Another research direction for synthetic tasks is to see if they could be added to transfer learning pipelines. Synthetic tasks are elementary and knowledge extracted from them by pre-training models on these tasks could enable models to then generalize better to other question answering task. Transfer learning is an exciting area of research which has seen interesting results in recent years [27]. Synthetic tasks could potentially be tasks which would work well for pre-training question answering models.

5.3 Epilogue

Looking back to section 1.4, all the objectives of this project were met. The report includes clear descriptions of the BrsDNC and the DNC as well as an analysis on the limitations and the advantages of each of these models. To summarize, the BrsDNC does not have the temporal linkage memory matrix used in the DNC to keep the order in which memory locations are written to. This is called the content based memory unit (CBMU). The BrsDNC also introduces a bidirectional architecture, a normalization technique and a bypass dropout training mechanism. This report identifies the limitations of the DNC to be its training time and the computational resources needed to train it while its advantages are that it surpasses the performance of the LSTM and obtains state of the arts results in many tasks. This is equivalent to the conclusion reached in the DNC paper. To reduce the training time and computation resources, the BrsDNC is introduced. The BrsDNC takes half the time to train and converges with a much better accuracy to state of the art results compared to the DNC. This report finds that these new improvements are mainly due to the normalization

technique that the BrsDNC introduces. Additionally, this paper concludes that it still unclear what the effects of removing the temporal linkage matrix are as it does not appear to have an impact on solving tasks which require remembering the order in sequences.

This project was a great opportunity to discover the current state of the art research in artificial intelligence. The Differentiable Neural Computer can successfully solve the bAbI tasks (Chapter 3) with a neural architecture somewhat resembling that of the human brain (Sections 2.5.1-2.5.2). The DNC is an interesting architecture to study but remains far from the neural network which will solve human intelligence. "Human-like learning and thinking machines will have to reach beyond current engineering trends in both what they learn and how they learn it [...] to one day think like people [36];" this means first of all finding better alternatives to LSTMs. Considering the history of Recurrent Neural Networks it is possible that the DNC will be the architecture which replaces LSTM, although a lot more work is needed. Research similar to that carried out in this report pave the way for the developments of better neural networks.

Bibliography

[1] J.Weston. The Sperm Whale. *bAbI tasks*. 12-04-2015.

[2] NVIDIA Tesla K80. *https://www.nvidia.com/en-gb/data-center/tesla-k80/*.

[3] Tensorflow. *https://www.tensorflow.org/*.

[4] Joerg Franke. Advanced Differentiable Neural Computer (ADNC) with application to bAbI task and CNN RC task. *https://github.com/JoergFranke/ADNC*, 12-04-2019.

[5] Google Deepmind. Differentiable Neural Computer. *https://github.com/deepmind/dnc*, 12-04-2019.

[6] Christopher Olah. Understanding LSTM Networks. *Recurrent Neural Networks*, 12-04-2019.

[7] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton. Neural and Evolutionary Computing. *Speech Recognition with Deep Recurrent Neural Networks*, 22-03-2013 .

[8] Kyunghyun Cho et al. Neural and Evolutionary Computing. *Learning Phrase Representaions using RNN Encoder-Decoder for Statiscal Machine Translation*, 03-09-2014 .

[9] Sepp Hochreiter, Jurgen Schmidhuber. Neural Computation. *Long Short-Term Memory*, 1997.

[10] Andrej Karpathy. Andrek Karpathy blog. *The Unreasonable Effectiveness of Recurrent Neural Networks*, 21-05-2015.

[11] Alex Graves, Greg Wayne, Ivo Danihelka. Neural and Evolutionary Computing. *Neural Turing Machines*, 10-12-2014.

[12] Jason Weston et al. Artificial Intelligence. *Towards AI-Complete Question Answering: a set of prerequisite toy tasks*, 31-12-2015.

[13] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes and Yann LeCun. Computation and Language. *Tracking the World State with Recurrent Entity Networks*, 10-05-2017.

[14] Hochreiter, J. Diploma thesis. *Untersuchungen zu dynamischen neuronalen Netzen*, 1991.

[15] F.A. Gers, Schmidhuber. IJCNN. *Recurrent nets that time and count*, 2000.

[16] Martin Sundermeyer, Ralf Schluter, Hermann Ney ISCA. *LSTM Neural Networks for Language Modeling*, 09-09-2012.

[17] Sam Greydanus Sam's Research Blog. *Differentiable memory and the brain*, 27-02-2017.

[18] Eugenio Culurciello. Medium. *The fall of RNN/LSTM*, 13-04-2018.

[19] Adam Santoro, Sergey Bartuno, Matthew Botvinick, Daan Wierstra, Timothy Lillicrap. Google Deepming. *Meta-Learning with Memory-Augmented Neural Networks*, 2016.

[20] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus. Neural and Evolutionary Computing. *End-To-End Memory Networks*, 31-03-2015.

[21] Michael Jacob Kahana. Cognitive Psychology. *Foundations of Human Memory*, 05-07-2012.

[22] Jeroen G.W.Raaijmakers, Richard M.Shiffrin. Psychology of Learning and Motivation. *SAM: A Theory of Probabilistic Search of Associative Memory*, 1980.

[23] Alex Graves et al. Nature. *Hybrid computing using a neural network with dynamic external memory*, 27-10-2016.

[24] Joerg Franke, Jan Niehues, Alex Waibel. Computation and Language. *Robust and Scalable Differentiable Neural Computer for Question Answering*, 07-07-2018.

[25] Gnter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter. Machine Learning. *Self-Normalizing Neural Networks*, 07-09-2017.

[26] Alex Graves, Jurgen Schmidhuber. Neural Networks. *Framewise phoneme classification with bidirectional LSTM and other neural network architectures*, 2005.

[27] Lisa Torrey and Jude Shavlik. Machine Learning Applications and Trends. *Transfer Learning*, 2010.

[28] Pusher Safer. *https://www.pushsafer.com/*, 12-04-2019.

[29] Marc W.Howard, Michael J.Kahana. Journal of Mathematical Psychology. *A Distributed Representation of Temporal Context*, 01-06-2002.

[30] Michael J.Kahana. Medical. *Foundations of Human Memory*, 18-05-2012.

[31] Mike Schuster and Kuldip K. Paliwal. IEEE Transactions on Signal Processing. *Bidirectional Recurrent Neural Networks*, 1997.

[32] Robert Csordas and J. Schmidhuber. Neural Information Processing Systems. *Improved Addressing in the Differentiable Neural Computer*, 2018.

[33] Alvin Chan, Lei Ma, Felix Juefei-Xu, Xiaofei Xie, Yang Liu, Yew Soon Ong. Machine Learning. *Metamorphic Relation Based Adversarial Attacks on Differentiable Neural Computer*, 07-09-2018.

[34] Hung Le, Truyen Tran, Svetha Venkatesh. Machine Learning. *Learning to Remember More with Less Memorization*, 05-01-2019.

[35] Jennifer Miller. The Cresset: A review of literature, the arts, and public affairs. *I Remember, Therefore I Am*, 2013.

[36] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum and Samuel J. Gershman. Behavioral and Brain Sciences. *Building machines that learn and think like people*, 24-11-2016.

Appendix A

Interface Values

The individual components are processed with the logistic sigmoid, 'oneplus' and softmax function so that they lie in the correct domain. The logistic sigmoid is already described in the report. The 'oneplus' is to contrain to $[1, \infty)$:

$$\text{oneplus}(x) = 1 + \log(1 + \mathrm{e}^x)$$

The softmax is used to constrain to the vectors S_n, the $N-1$ dimensional unit simplex

$$S_n = \{\boldsymbol{\alpha} \in \mathbb{R}^N : \alpha_i \in [0,1], \sum_{i=1}^{N} \boldsymbol{\alpha}_i = 1\}$$

- R read keys $\{\boldsymbol{k}_t^{r,i} \in \mathbb{R}^W; 1 \leq i \leq R\}$;
- R read strengths $\{\beta_t^{r,i} = \text{oneplus}(\hat{\beta}_t^{r,i}) \in [1, \infty); 1 \leq i \leq R\}$;
- $\boldsymbol{k}_t^w \in \mathbb{R}^W$, the write key;
- $\beta_t^w = \text{oneplus}(\hat{\beta}_t^{r,i}) \in [1, \infty)$, the write strength;
- $\boldsymbol{e}_t = \sigma(\hat{\boldsymbol{e}}_t) \in [0,1]^W$, the erase vector;
- $\boldsymbol{v}_t^w \in \mathbb{R}^W$, the write vector;
- R free gates $\{f_t^{r,i} = \sigma(\hat{f}_t^{r,i}) \in [0,1); 1 \leq i \leq R\}$;
- $g_t^a = \sigma(\hat{g}_t^a) \in [0,1]$, the allocation gate;
- $g_t^w = \sigma(\hat{g}_t^w) \in [0,1]$, the write gate;
- R read modes $\{\boldsymbol{\pi}_t^i = \text{softmax}(\hat{\boldsymbol{\pi}}_t^{r,i}) \in S_3; 1 \leq i \leq R\}$;

Appendix B

Dataset Specs

Task	Name	Samples	Vocab Size	Min Length	Max Length
1	Single Supporting Fact	2000	22	85	93
2	Two Supporting Facts	2000	36	89	452
3	Three Supporting Facts	2000	37	130	1920
4	Two Arg. Relations	10000	17	24	24
5	Three Arg. Relations	2000	42	94	820
6	Yes/No Questions	2000	38	93	191
7	Counting	2000	46	97	361
8	Lists/Sets	2000	38	87	346
9	Simple Negation	2000	26	95	109
10	Indefinite Knowledge	2000	27	95	124
11	Basic Coreference	2000	29	90	100
12	Conjunction	2000	23	105	112
13	Compound Coref	2000	29	100	111
14	Time Reasoning	2000	28	116	156
15	Basic Deduction	2500	20	72	72
16	Basic Induction	10000	20	47	47
17	Positional Reasoning	1250	21	92	128
18	Size Reasoning	1978	21	80	249
19	Path Finding	10000	26	53	53
20	Agent's Motivations	933	38	45	161
all		62991	159	24	1920

Figure B.1: Original Dataset

Task	Name	Samples	Vocab Size	Min Length	Max Length
1	Single Supporting Fact	2000	22	85	93
2	Two Supporting Facts	1989	36	89	450
3	Three Supporting Facts	985	37	130	450
4	Two Arg. Relations	10000	17	24	24
5	Three Arg. Relations	1803	42	94	450
6	Yes/No Questions	2000	38	93	191
7	Counting	2000	46	97	361
8	Lists/Sets	2000	38	87	346
9	Simple Negation	2000	26	95	109
10	Indefinite Knowledge	2000	27	95	124
11	Basic Coreference	2000	29	90	100
12	Conjunction	2000	23	105	112
13	Compound Coref	2000	29	100	111
14	Time Reasoning	2000	28	116	156
15	Basic Deduction	2500	20	72	72
16	Basic Induction	10000	20	47	47
17	Positional Reasoning	1250	21	92	128
18	Size Reasoning	1978	21	80	249
19	Path Finding	10000	26	53	53
20	Agent's Motivations	933	38	45	161
all		61438	159	24	450

Figure B.2: Modified Dataset

Appendix C

Extra Hypothesis 1

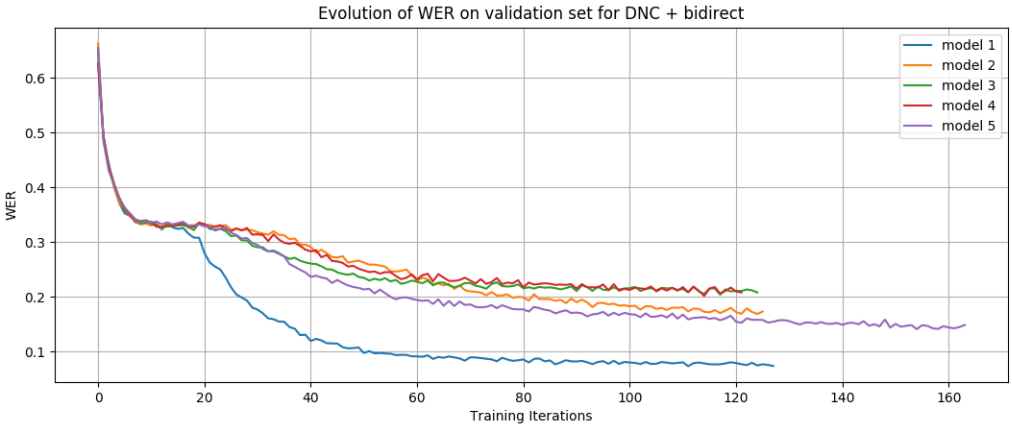

Figure C.1: Bidirectional DNC

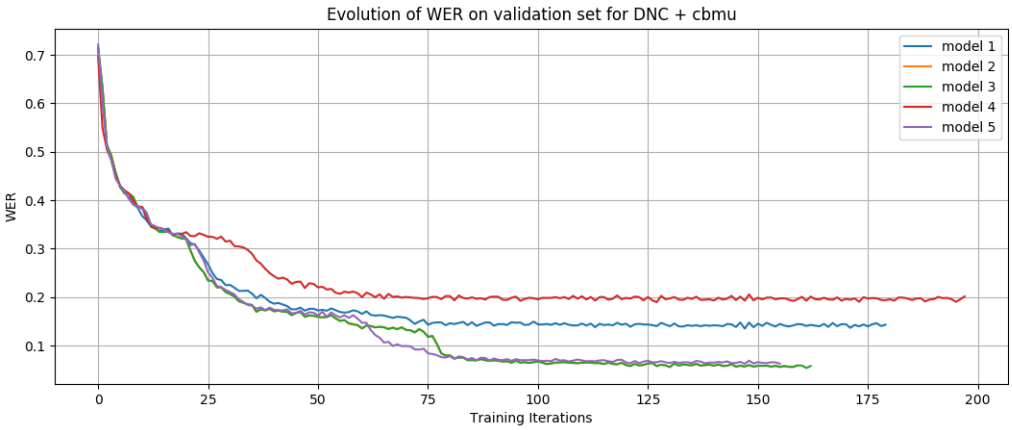

Figure C.2: DNC with CBMU

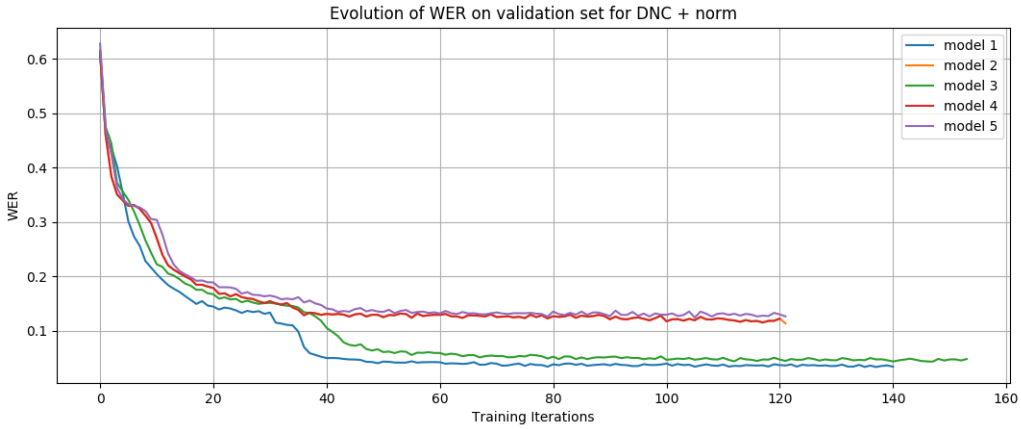

Figure C.3: DNC with Norm

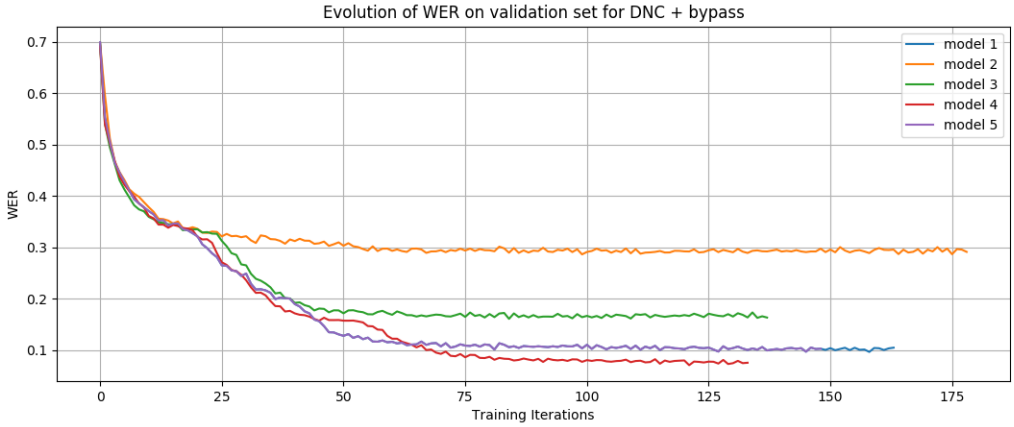

Figure C.4: DNC with Bypass