Research intern
at IBM Almaden Research Center

Younes Bouhadjar
younes.bouhadjy@gmail.com

Differentiable Working Memory

From March, 03rd to August, 30th 2018

Supervisors:
Dr. Jayram Thathachar
(jayram@us.ibm.com)
Dr. Liliana Buda-Prejbeanu
(liliana.buda@cea.fr)

Research Group:
Machine Intelligence
Manager:
Ahmet Ozcan
(asozcan@us.ibm.com)

Master Nanotech
Grenoble INP Phelma, Politecnico di Torino, EPFL
Academic Year: 2017/2018

Grenoble INP Phelma, Politecnico di Torino, EPFL
Nanotech international master

Differentiable Working Memory

By Younes Bouhadjar

Abstract

Artificial Intelligence has recently shown great success at language translation, computer vision and many other sensory perception tasks. However, it still requires further improvements to address problems that involve higher order cognitive behaviors, such as reasoning.

The human brain relies on multiple memory systems for intelligent behavior. Working memory is an essential component for high order cognitive tasks ranging from language, planning and reasoning to decision making. In this thesis, I introduce a new model called Differentiable Working Memory (DWM), which emulates the human working memory. As it shows the same functional characteristics as working memory, the model robustly learns psychology-inspired tasks and converges faster than comparable state-of-the-art models. Moreover, the DWM model successfully generalizes to sequences two orders of magnitude longer than the ones used in training. Our in-depth analysis shows that the behavior of DWM is interpretable and that it learns to have fine control over memory, allowing it to retain, ignore or forget information based on its relevance. To facilitate running the DWM model against the different WM tasks and also running other models on the same set of tasks to establish baselines, we designed a framework called MI-Prometheus, that standardizes the interface that connects together the components needed in a machine learning system: problems, models architectures, and training/testing configurations.

This work is a step towards building Artificial General Intelligence models, where different kinds of memory and reasoning centers are needed. We believe that the DWM model will be a critical part of such a cognitive architecture.

keywords: Artificial Intelligence, working memory, Memory, Differentiable Working Memory, cognitive, reasoning, psychology-inspired tasks, framework.

Grenoble INP Phelma, Politecnico di Torino, EPFL
Nanotech international master

Mémoire de Travail Différentiable

Par Younes Bouhadjar

Résumé

L'intelligence artificielle a récemment démontré de nombreux succès, dans la traduction, la vision par ordinateur et pour de nombreuses autres tâches liées à la perception sensorielle. Cependant, elle demande encore des progrès supplémentaires pour résoudre les problèmes impliquant des comportements cognitifs de plus haut niveau, tels que le raisonnement.

Le cerveau humain repose sur de multiples mémoires pour un comportement intelligent. La mémoire de travail est une composante essentielle des tâches cognitives de haut niveau, allant du langage, passant par la planification et le raisonnement, à la prise de décision. Dans cette thèse, je présente un nouveau modèle, appelé Differentiable Working Memory (DWM), qui émule la mémoire de travail humaine. Présentant les mêmes caractéristiques fonctionnelles que la mémoire de travail, le modèle apprend avec certitude des tâches inspirées par la psychologie, et converge plus rapidement que les modèles de l'état de l'art. De plus, le modèle DWM généralise à des séquences plus longues par deux ordres de grandeur que celles utilisées durant l'entraînement. Notre analyse approfondie montre que le comportement de DWM est interprétable et que ce dernier apprend à bien utiliser la mémoire, ce qui lui permet de conserver, ignorer ou oublier des informations en fonction de leur pertinence. Pour faciliter la comparaison du modèle DWM par rapport aux différentes tâches mémorielles et pour tester d'autres modèles sur le même ensemble de tâches afin d'établir des références, nous avons conçu une libraire logicielle appelée MI-Prometheus qui standardise l'interface reliant les composants nécessaires à un système d'apprentissage automatique : problèmes, architectures de modèles et configurations d'entrainement / évaluation.

Ce travail est une étape vers la construction de modèles d'intelligence artificielle générale, dans lesquels différents types de mémoire et centres de raisonnement sont nécessaires. Nous pensons que le modèle DWM constituera un élément essentiel d'une telle architecture cognitive.

Mots clés: *Intelligence artificielle, mémoire de travail, mémoire, Differentiable Working Memory, cognitif, raisonnement, libraire logicielle.*

Grenoble INP Phelma, Politecnico di Torino, EPFL
Nanotech international master

Memoria di Lavoro Differenziabile

Di Younes Bouhadjar

Sommario

L'intelligenza Artificiale ha recentemente dimostrato un grande successo nella traduzione linguistica, nella visione artificiale e in molti altri compiti di percezione sensoriale. Tuttavia, richiede ancora ulteriori miglioramenti per affrontare i problemi relativi a comportamenti cognitivi di ordine superiore, come il ragionamento.

Il cervello umano fa affidamento a molteplici sistemi di memoria per ottenere un comportamento intelligente. La memoria di lavoro è un componente essenziale per le funzioni cognitive di alto livello, come il linguaggio, la pianificazione e i ragionamenti necessari a prendere delle decisioni. In questa tesi, introduco un nuovo modello chiamato Differentiable Working Memory (DWM), il quale emula la memoria di lavoro umana. Poiché mostra le stesse caratteristiche funzionali della memoria di lavoro, il modello impara funzioni ispirate a quelle psicologiche e converge più velocemente rispetto ad altri modelli simili attualmente esistenti. Inoltre, il modello DWM generalizza con successo sequenze di due ordini di grandezza più lunghe rispetto a quelle usate nell'addestramento. La nostra analisi dettagliata mostra che il comportamento del DWM è interpretabile e che impara ad avere un controllo dettagliato sulla memoria, permettendogli di conservare, ignorare o dimenticare informazioni in base alla loro rilevanza. Per facilitare l'esecuzione del modello DWM rispetto ai diversi compiti della memoria di lavoro ed inoltre fare in modo che altri modelli siano contemporaneamente in esecuzione sullo stesso set di task per definire le condizioni iniziali, abbiamo progettato un framework chiamato MI-Prometheus, che standardizzi l'interfaccia che connette i componenti necessari in un sistema di machine learning: problemi, architetture dei modelli, e configurazioni per il training/testing.

Questo lavoro è un ulteriore passo verso la costruzione di modelli di Artificial General Intelligence, dove tipi differenti di centri di memoria e ragionamenti sono richiesti. Crediamo che il modello DWM sarà parte cruciale di tale architettura cognitiva.

Parole chiave*: Intelligenza Artificiale, memoria di lavoro, memoria, Differentiable Working Memory, cognitivo, ragionamento, struttura*

Acknowledgement

I would like first to express my special thanks of gratitude to Dr. Ahmet Ozcan the manager of the Machine intelligence group, who interviewed me and arranged the work plan for my master thesis, working under his supervision was extremely stimulating, I felt being free in realizing my work and taking decisions. I admire him for the atmosphere he manages to establish in the lab, we were all working together, sharing information, skills and knowledge.

I also want to express my deepest gratitude to Dr. Jayram Thathachar who supported me throughout this work and feel very fortunate that I could expand my horizon with his help in this area of research. He teaches me not only information about the field but also how to be a better researcher.

Many thanks to Dr. Tomasz Kornuta for helping me out during all the period of the project, for listening me and giving a hand in solving many problems. It is also thanks to him that you could feel such a friendly atmosphere in the laboratory.

I am also grateful to Mr. Ryan McAvoy who was supporting me during all the period of the thesis with his precious advices, he shed lights on many research skills that I should acquire. Also special thanks to Mr. Alexis Asseman who was a tremendous person, giving me a lot of support with the software engineering part of the project.

I would like to give a big thanks to Mr.Vincent Marois, he was an irreplaceable mate in this big adventure. I thank him for sharing everything in these months, the joy and the difficulties. Also many thanks to Mr.Vicent Albouy being supporting all the time and being a great source of motivation.

I would like also to give my great thanks to IBM Almaden Research center which offered me an opportunity as well as financial support for my research internship. I am also grateful to Dr. Luisa Bozano that helped connecting my University and IBM researchers centers, and taking care of most of my internship paperworks. I am also grateful to Dr. Winfried Wilcke the senior manager of the MI group for being a great source of support and inspiration.

Lastly, I would like to thank Dr.Youla Morfouli, Dr. Liliana Prejbeanu and Ms. Eliane Zammit the coordinators of my master, for their careful and precious guidance which were extremely valuable for my studies in the nanotech master.

Contents

List of Figures

Symbols and Abbreviations:

MI	Machine Intelligence
ARC	Almaden Research Center
VQA	Visual Question Answering
MANNs	Memory Augmented Neural Networks
AI	Artificial Intelligence
MW	Working Memory
DWM	Differentiable Working Memory
ML	Machine Learning
ANN	Artificial Neural Networks
BNN	Biological Neural Networks
RNN	Recurrent Neural Networks
DNC	Differentiable Working Memory
LSTM	Long Short Term Memory
NTM	Neural Turing Machine
GD	Gradient Descent
CNN	Convolutional Neural Network
MAC	Memory Attention Composition

1 Introduction

Over a period of six months starting from March 2018, I worked as a research intern at IBM Almaden Research Center (ARC) within the Machine Intelligence (MI) Group. This work was done within the scope of my Master thesis of the Nanotech International program.

The MI group was created initially to implement a new Machine Intelligence algorithm called Contextual Aware Learning (CAL), inspired by the current knowledge of how the mammals cerebral cortex works. Recently, the MI group focus was shifted towards deep learning [1] [17]. More specifically, the main area of interest are Visual Question Answering (VQA) [1] and Memory Augmented Neural Networks (MANNs) [18] .

1.1 IBM Research

IBM Research, is the largest industrial research organization in the world, with 12 labs on six continents. The main goal of IBM Research is to help its clients achieve both high quality and competitive products by offering a combination of Hardware, Software and Services solutions.

During my thesis, I worked at the IBM Almaden Research Center, the focus of which is mainly centered around the development of new cognitive computing softwares (Artificial Intelligence) and hardwares (neuromorphic chips) aiming to enhance machines intelligence to improve their interactions with humans. The center has other labs which work on diverse areas ranging from quantum computers and hardware design to biomedical field and nano-fabrication.

1.2 Thesis Motivation

Artificial Intelligence (AI) has made tremendous progress over the past few years and is getting involved continuously in our daily life, from allowing us to translate several languages, predicting our financial budget, to driving our cars and scheduling our meetings [37]. A great example showing AI capabilities is the famous Jeopardy game between Watson (IBM's question answering AI) and the 2 Jeopardy champions - Brad Rutter and Ken Jennings - where Watson won by a large margin [14]. Another more recent example is Google DeepMind's AlphaGo defeating the South Korean master Lee Sedol in 4 games out of 5 in 2016 [3].

One of the ultimate goals of AI research is the creation of an Artificial General Intelligence, that can do things in a way that is indistinguishable from human behavior, and can be reliable in doing our different daily tasks. What have been done so far in the field of AI, is the creation of a set of algorithms that are essentially big correlation engines that try to find any spurious pattern that describes the data being fed in. The

[1]Deep Learning a set of algorithms that are inspired by the structure of the brain called neural network (Deep refers to the use of many neural networks layers)

structure, size and nature of these diverse data often limits the performance of these models. For example, this issue clearly manifests when solving problems related to reasoning [2] which is central to human intelligence [36]. One may argue that coming closer to human intelligence requires designing models which have similarities with the respect to human brain. This is motivated by the fact that the human brain is the only proof we have of the existence of such an intelligence. Studying human cognition and its neural implementation has a vital role to play, as it can provide a window into various important aspects of higher level general intelligence [24].

Motivated by the above, we tried to come closer to neuroscience and cognitive psychology, in order to design novel models that can solve complex tasks such as reasoning. It was proven that the main building blocks of the human brain which allow such a superior cognitive behavior are the different kind of memories it possesses [26]. For instance, reasoning requires working memory, episodic memory (content in time) and associative memory (content similarity).

In this thesis, we focused specifically on studying the human working memory (WM), which is a mental workspace that processes data that is no more present in the environment. This led to designing a model called Differentiable Working Memory (DWM), which could successfully mimic the functional behavior of the working memory. This work is a step towards building Artificial General Intelligence models, where different kinds of memory and reasoning centers are needed. We believe that the DWM model can fill in one of these needs. The main contributions of this work are as follow:

[2]Reasoning is the ability to manipulate previously acquired knowledge to draw novel inferences or answer new questions

Figure 1 – IBM Almaden research center

- Develop and implement a Memory-Augmented Neural Network model, inspired by the human working memory.
- A new attention control mechanism for memory, which can deal with interference.
- Implement psychometric tests designed for human working memory and applied to our artificial model, which shows superior performance when compared to the state-of-the-art (i.e. LSTM [27], Differentiable Neural Computer [19])
- An examination of strategies developed for handling memory scarcity.

To facilitate running the DWM model against the different WM tasks and also running other models on the same set of tasks to establish baselines, we designed a framework called MI-Prometheus, that standardizes the interface that connects together the components needed in a machine learning system: problems, models architectures, and training/testing configurations.

1.3 Structure of the thesis

The thesis begins with a background of some of the Machine Learning algorithms mentioned throughout the different chapters of the thesis. In chapter 3, I introduce the DWM model, which was intentionally designed to emulate human working memory. Following, I present the results and how we could interpret the mechanism that the model can learn to solve the different tasks. In chapter 4, I detail the framework we designed to facilitate running the DWM model against the different tasks.

2 Background

2.1 Machine Learning

Machine Learning (ML) is a subgroup of AI, which can be defined as an algorithm that has the ability to learn from prior data in order to produce an answer. In other words, ML teaches machines to make decisions in situations they have never encountered. There are two main types of ML approaches:

- **Supervised Learning**: showing a model a data set of situations and telling it what the right answer is. Think of it as a teacher and student scenario. Once the model has been trained, it can be tested using previously unseen data - which is still statistically similar to the training data, hoping it can generalizes to novel situations. Figure 2 shows the MNIST dataset where supervised learning is used. During training, the model is exposed to different hand written digits and has access to the corresponding correct output (integer from 0 to 9). During the test phase (inference), new hand written digits are fed into the model, and we ask it to predict the corresponding label.
- **Unsupervised learning**: Describes the situation where training a model is done using only a set of inputs without knowing the corresponding output. Think of it as a situation where there is no teacher (compared to supervised learning). The corresponding models try to find structure or relationships between the different inputs. One of the most used unsupervised learning algorithm is clustering, which creates different clusters (groups) of inputs and predicts in which cluster the new input belongs.

We can also briefly mention Reinforcement Learning, where an agent (for instance a robot) ought to take actions (e.g. go left, right, up or down) in an environment (e.g. a maze) so as to maximize some notion of cumulative reward (e.g. exit the maze by making the lowest number of moves possible).

2.2 Neural Networks

Artificial Neural Networks (ANNs) are a subset of ML algorithms, originally invented in the mid-20th century, inspired by the human brain [23]. However at that time, the limited computational power was an obstacle that researchers in this field were facing. The interest in ANN has grown exponentially since 2012, after the development of powerful computing hardwares, which started a new era of ANN. Since then, substantial progress was done.

What helps the rapid development of the ANNs, is the belief of researchers that ANNs can potentially emulate the human brain, as they were initially designed to replicate the overall structure of the brain networks. Other Machine Learning researchers argue that overstating the connection between ANNs and the human brain might be misleading as neural networks are mostly guided by developments in engineering and mathematics, rather than biology. Nevertheless, studying the human brain and how it is related to ANNs can be a great source of inspirations.

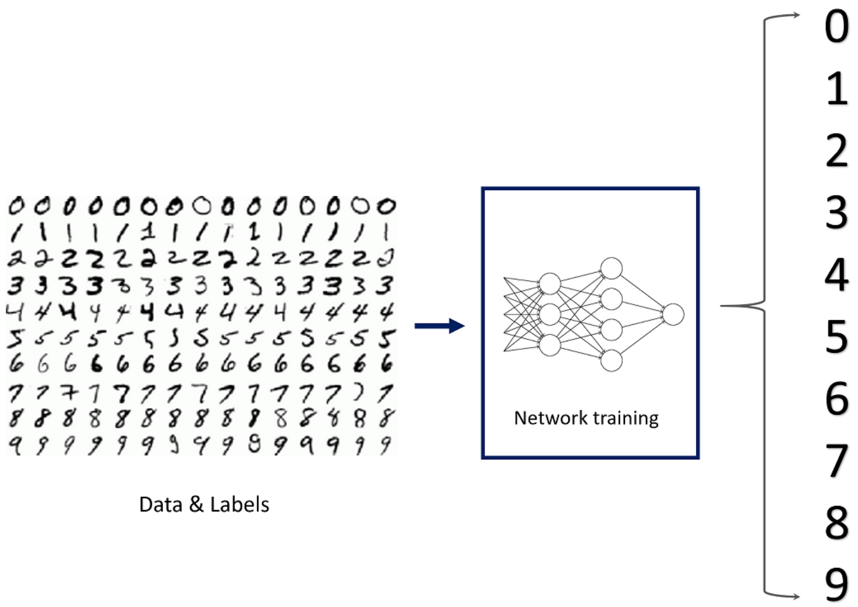
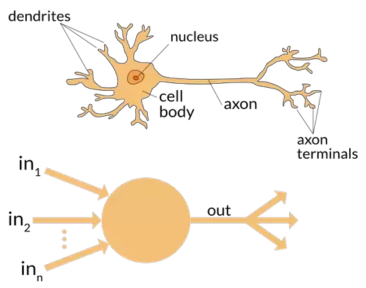


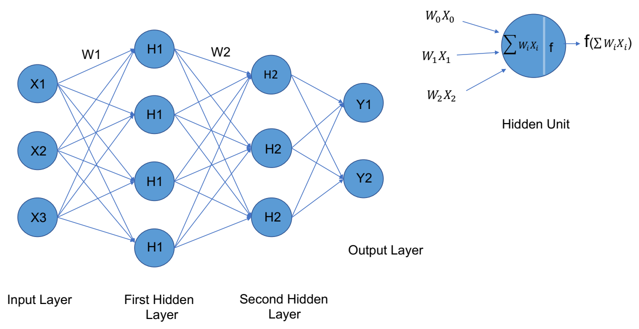
Figure 2 – a subset of MNIST dataset and a neural network trained to predict the right digit

2.2.1 Theory

The building blocks of the ANNs are the perceptrons, which is an attempt to model the biological neurons. A biological neuron is a type of cell that has several inputs called DENDRITES, which receive electrical pulses that might activate the cell to produces its own activity and sends it along its outputs called AXONS [34]. During this process, some paths which connect the different neurons in the Biological Neural Network (BNN) may be "strengthened" or "weakened", creating specialized patterns that are behind human cognition: learning new language, remembering a new word etc.



(a) Biological neuron versus artificial neuron



(b) Artificial neural network composed of two layers

In the same spirit, the artificial neuron (perceptron) receives a set of inputs x_i, multiply them by a set of modifiable weights w_i, then processes their sum with its activation function ϕ and passes the result of the activation function to the next neuron, see figure 3a. We can then form a network by connecting these artificial neurons together [23]. Usually this is done in layers - one layer's outputs are connected to the next layer's inputs. See figure 3b for illustration. People in the field of ML refer to neural networks with more than two layers as deep neural networks. A single neural network layer can be formulated as following:

$$y_i = \phi\left(\sum_i w_i x_i\right) = \phi(\mathbf{w}^T\mathbf{x})$$

The activation function ϕ determines the final output of each neuron. It is important to properly select the function in order to create an effective network. Linear systems, for instance, have some drawbacks, being unable to solve problems that are not linearly separable, such as the XOR-problem.

The following are some examples of activation functions:

- Sigmoid activation: it transforms the output of the neural network to a value between 0 and 1, similar to a probability.

$$\phi(\mathbf{w}^T\mathbf{a}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{a})}$$

- tanh activation: it forces the negative values to be zeros, and outputs the positive values as they are.

$$\phi(\mathbf{w}^T\mathbf{a}) = \tanh(\mathbf{w}^T\mathbf{a})$$

2.2.2 Training

The two training methods explained previously –supervised and unsupervised can be used to train neural networks. In the following part, I will cover the supervised one as it was used during this thesis. When training neural networks in a supervised way, we modify their weights so that they can predict the right answer for the majority of the training samples. This is an iterative process: the weights get updated every time a new input is fed until we reach convergence (defined by the minimum of a given loss function). The trained weights form a function (denoted h) that operates on input data. With a trained network, we can make predictions given unlabeled test inputs.

We can train a neural network to perform regression (output a continuous values) or classification (output discrete values).

Within the scope of the two approaches, the goal is to find the set of weights that provide the best fit to our training data. To measure how well our model fits the data, we use a cost function (error function), L. The following are examples of cost functions:

- Regression: Least square error

$$L(\mathbf{w}) = \sum_i (h(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

- Classification: Cross entropy

$$L(\mathbf{w}) = \sum_i y_i log(h(\mathbf{x}_i, \mathbf{w}))$$

In order to find the optimal set of weights, we minimize $L(\mathbf{w})$. One of the well-known method to do so is Gradient Descent (GD), in which we follow the direction of the negative gradients to find a minimum of L. Note that there exist several other optimization algorithms, which are more complex than GD and can offer better performance in term of avoiding local minima and faster convergence.

The following is the mathematical formulation of GD:

$$Repeat\ until\ convergence : \left\{ w_j \leftarrow w_j - \alpha \frac{\delta}{\delta w_j} J(w_j) \right\}$$

Neural networks being composed of many layers of neurones, we need to propagate the error from the output layer back to the input layer to update the weights of every node. A well-known technique is used called Backpropagation This is actually a simple implementation of the chain rule of derivatives, which simply states the ability to compute all required partial derivatives in linear time in terms of the graph size (while naive gradient computations would scale exponentially with depth) [25].

2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are NNs with the previous time step being fed as input of the next time step [20]. Figure 5 shows the overall design of a RNN with its structure being unfolded through time. This helps RNNs to develop an understanding of the time aspect of the input data. For instance, when translating a sentence, RNNs can grasp the specific context of the current word to be translated, as they keeps building knowledge when going through the different words in a sentence. However, RNNs are not suited for static inputs, for instance when classifying handwritten digits. Here, the model doesn't need temporal information, as there is no difference between the first element or 100th element of the dataset. Hence, a multi layer neural network should be able to solve such a problem.
The basic equations of RNNs are as following:

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\ h^{(t)} &= tanh(a^{(t)}) \\ o^{(t)} &= c + Vh^{(t)}y^{(t)} \\ y^{(t)} &= softmax(o^{(t)}) \end{aligned} \tag{1}$$

With H, W and V being neural networks and b, c are learnable biases. There are other complex RNN models like: Long Short Term Memory (LSTM) [16] and Gated Recurrent Unit (GRU) [4] which have superior performance in terms of preserving information about longer input sequences.

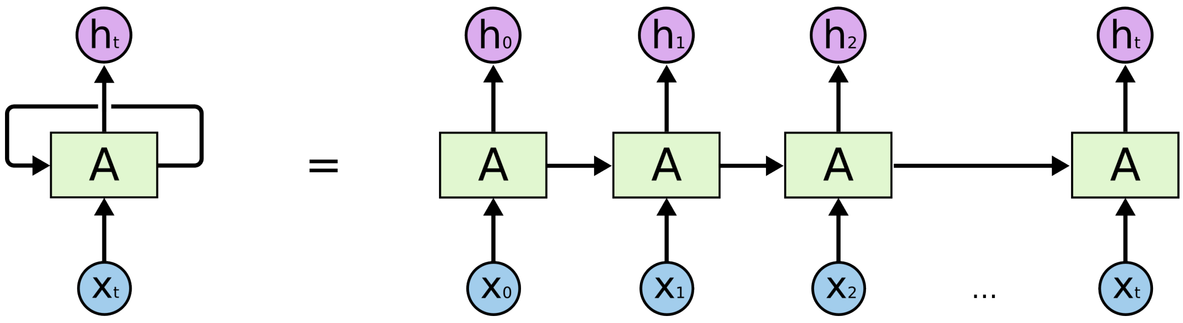


Figure 4 – Unfolded recurrent neural network through time, A is an artificial neural network

2.4 Memory Augmented Neural Networks (MANNs)

A Memory Augmented Neural Network (MANNs) is a neural network coupled to an external memory. This means that when training, the neural network learns how to read and write from / to the memory without being explicitly programmed to do so. It was proven that MANNs can learn algorithms to process tasks rather than just memorizing data [18, 19], unlike what a RNN would do. This extends the capabilities of neural networks to process longer sequences and more complex data. During this thesis, an extensive work was done around the concept of MANNs as one of the main goal was to emulate the human working memory. In the following, details about MANNs are exposed.

Being an emerging field, the terminology surrounding (MANNs) is constantly evolving. We adopt the convention of Santoro et al. [38] and view a MANN as generally consisting of the following components: a *controller*, a *memory* module, and memory-based *attention* with read/write mechanisms. We describe a template that formalizes the notion of a general MANN below and later use this template to describe our new model.

2.4.1 Controller

The controller is a module that interfaces with the external world by reading the input stream, one element at a time, and producing an output in each time step. Most importantly, it has the ability to read from and/or write to memory in each time step using *read/write heads*. The controller is *recurrent* if it has an internal state that gets transformed in each step, in analogy with a recurrent neural network (RNN).

2.4.2 Memory

The memory consists of many *words*, each storing a vector of values with fixed dimension (vector-valued memory words). Crucially, when we allow memory words to contain real numbers [3], we can also design *differentiable* transformations of the memory. The words are indexed starting from 0 and the index of each word is called its *address*. This

[3]This involves a slight abuse of notation since the term “word” is usually reserved for bit-based architectures.

way the memory can viewed as a two-dimensional array $M \times N$ of values, where the columns are indexed by addresses. Here N is the *size* and M is the *width* of memory.

2.4.3 Attention

We associate each read/write head with a *soft* attention[4]mechanism, i.e. a non-negative weight vector of dimension N whose components sum to 1. With a view towards our model, we allow additional attention mechanisms in the model for serving other roles. Given H attention vectors, we compactly represent all of them as an $H \times N$ array.

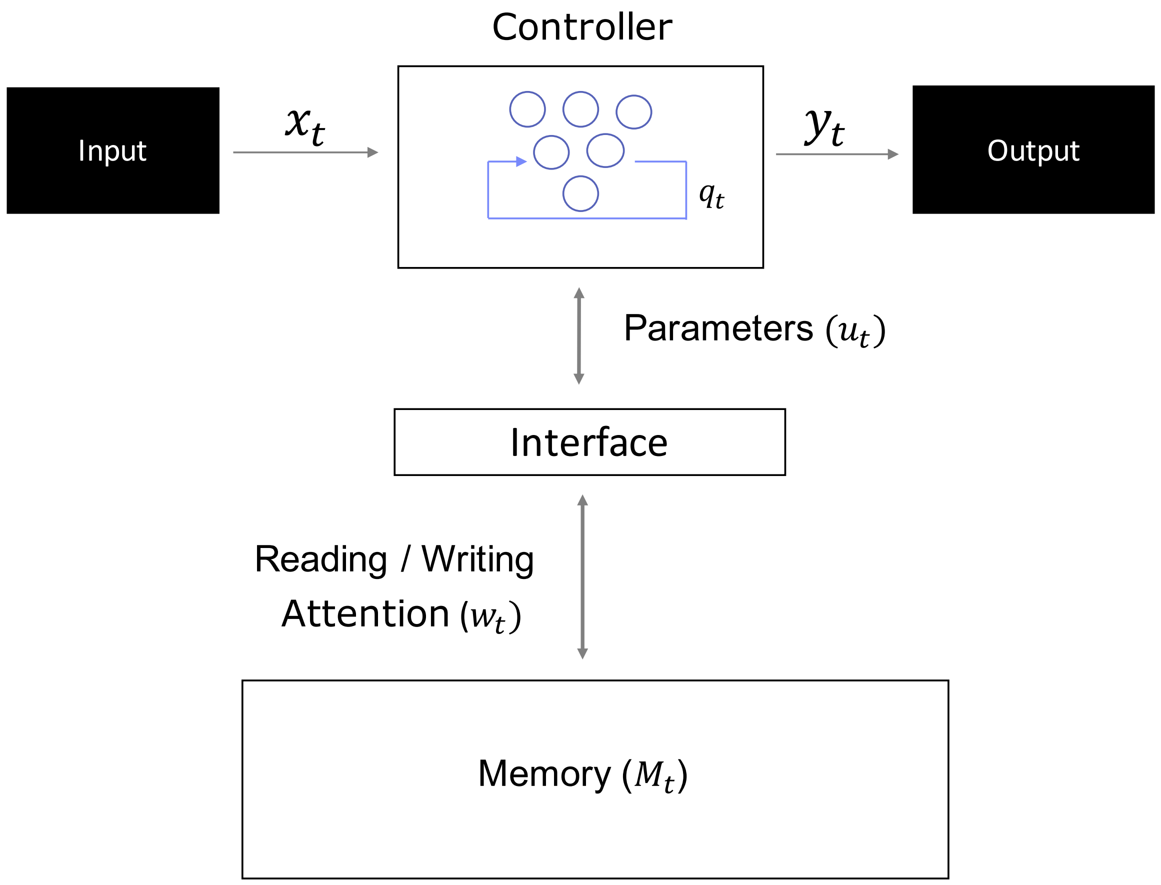


Figure 5 – The overall design of a memory augmented neural network

2.4.4 Overall design

The equations that govern the time evolution of a MANN for $t = 1, 2, \ldots$ are as follows.

(a) $d_t = \text{MEM\_READ}(\mathcal{M}_{t-1}, \mathcal{W}_{t-1})$ (b) $(q_t, y_t, u_{t-1}) = \text{CONTROLLER}(q_{t-1}, x_t, d_t)$
(c) $\mathcal{M}_t = \text{MEM\_WRITE}(\mathcal{M}_{t-1}, \mathcal{W}_{t-1}, u_t)$ (d) $\mathcal{W}_t = \text{ATTN\_UPDATE}(\mathcal{W}_{t-1}, u_t)$,

where the initial state is given by $(q_0, \mathcal{M}_0, \mathcal{W}_0)$. At the start of time t, the memory contents are given by $\mathcal{M}_{t-1}$ and the attention array equals $\mathcal{W}_{t-1}$. Using the portion of

[4]Although for intuition we will continue to use terms that would make sense strictly only for hard attention.

$\mathcal{W}_{t-1}$ associated with read heads, we read the data d_t from memory in part (a). Next in part (b) the controller uses its hidden state q_{t-1}, the current input element x_t and the data d_t to update its state, produce an output y_t and generate an update parameter u_t. Finally, u_t is used to update both the memory in part (c), using the portion of $\mathcal{W}_{t-1}$ associated with write heads, as well as the attention array in part (d). While all the four transformations above could contain trainable parameters, the art of designing MANNs involves placing suitable restrictions so that the MANN can learn to solve a wide range of problems using tractable learning procedures.

3 Working Memory

3.1 Introduction

Keeping information in mind after it is no longer present in the environment is critical for all higher cognitive behaviors. *Working memory* (WM) is the term used for this ability, which is distinct from the storage of vast amount of information in long-term memory [2]. The two main distinguishing characteristics of WM are the limited capacity (3-5 items) [7] and temporary retention (secs-minutes). Hence, WM is not a storage per se, but a mental workspace utilized during planning, reasoning and solving problems. Most psychologists differentiate WM from “short-term” memory because it can involve the manipulation of information rather than being a passive storage [9]. Along the same lines, Engle et al. [13] argued that WM is *all about the capacity for controlled, sustained attention in the face of interference or distraction.* Attention-control is a fundamental component of the WM system and probably the main limiting factor for capacity [8, 12]. Consequently, the inability to effectively parallel process two-attention demanding tasks limits our multitasking performance severely.

Over the past several decades psychologists have developed tests to measure the individual differences in WM capacity and better understand the underlying mechanisms. These tests have been carefully crafted to focus on the specific aspects of WM such as task-driven attention control, interference and capacity limits. The best known and successfully applied class of tasks for measuring WM capacity is the “complex span” paradigm. The challenge presented by complex span tasks is recalling the list of items, despite being distracted by the processing task. Studies show that individuals with high WM capacity are less likely to store irrelevant distractors [41] and they are better at retaining task-relevant information [32]. Developing task-driven strategies for cognitive control are essential for the effective use of WM.

The power of maintaining information over time has also been recognized by the AI community. Starting with the basic recurrent neural network architectures [11, 28] followed by the introduction of gating mechanisms [27], the research has recently moved onto more complex architectures with memories [18, 29, 42, 19, 38, 21]. These models are typically applied to tasks (e.g. associative recall, bAbI QA [43]) that require a complex mixture of long-term memory (episodic and semantic) and working memory. In the human brain, these kinds of memory systems are distinct: working memory is instantiated in multiple interconnected areas with the prefrontal cortex playing a major role [6], whereas for episodic memory the hippocampus is the critical structure [15]. Studying these mechanisms separately is necessary to disentangle the contributions of each memory system and develop a detailed understanding of human intelligence.

In this work, we introduce a Differentiable Working Memory (DWM) model that more closely mimics the functional behavior of human WM. We also present a battery of tasks adopted from the cognitive psychology literature that allow us to elucidate the working memory behavior of a neural network directly. In contrast to prior work on neural nets with external memory, our starting point is the functional attributes of WM and the tasks that primarily tests those.

3.2 Methods

3.3 Psychometric tasks for working memory

Over last several decades cognitive psychologists have developed many tasks to measure the performance of human WM (See figure 6 for examples). These tasks are mainly sequential and typically divided into verbal and visuospatial domains. One of the fundamental goals of these psychometric tests is to measure individual differences and correlate to performance in reasoning and fluid intelligence. In this regard, WM capacity, retention and the ability to switch between tasks are the key predictors.

Given the large number of WM tasks in the psychology literature and various categorizations by different researchers, we built a taxonomy of tasks (7) and carefully selected tasks that seem to be the most representative for a given category. First order categorization is based on the number and complexity of tasks. For simple tasks, the presence of data manipulation is the next level sub-category. Serial recall is a prime example for a simple task without manipulation. Other WM tasks may require the manipulation of the memory content, which could be divided in spatial and temporal domains. The complex WM tasks involve multiple sequential inputs or sub-tasks but not necessarily implies "multi-tasking". We follow the framework developed by Clapp and Gazzeley [5] to distinguish the sources of goal interference, i.e. Distraction (to-be-ignored) and Interruption (i.e. multi-tasking). For example, Operation Span (6) is a dual task because the subjects must attend and process the summation (Interruption) even though they do not need to recall the results afterwards. In Reading Span (1980 version by [10]) subjects read sentences and need to recall the last word of each one.

3.4 Differentiable Working Memory (DWM)

Inspired by human working memory, we designed a Differentiable Working Memory model with the appropriate functional characteristics. We illustrate its operation on 10. As a neural network with external memory, the DWM has three main components: a

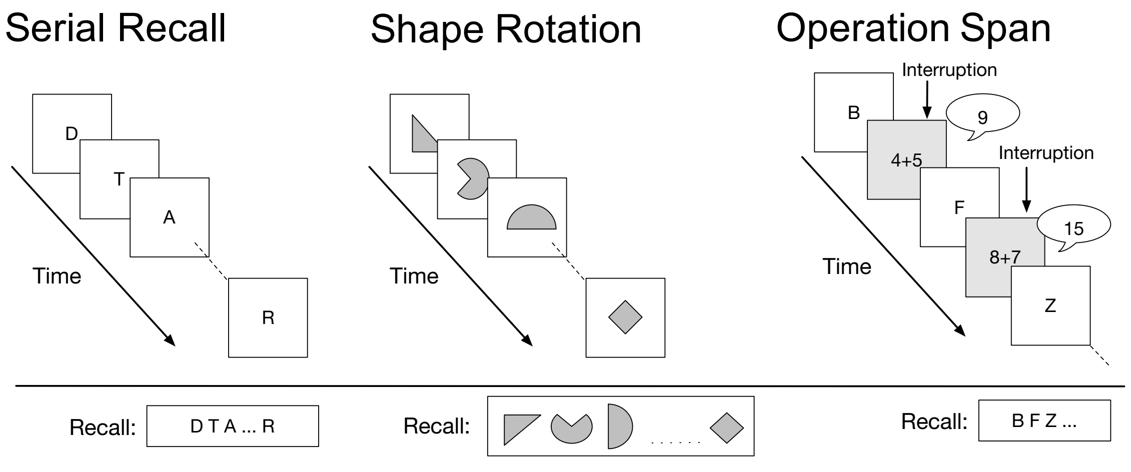


Figure 6 – Exemplary tasks for testing the performance of human working memory

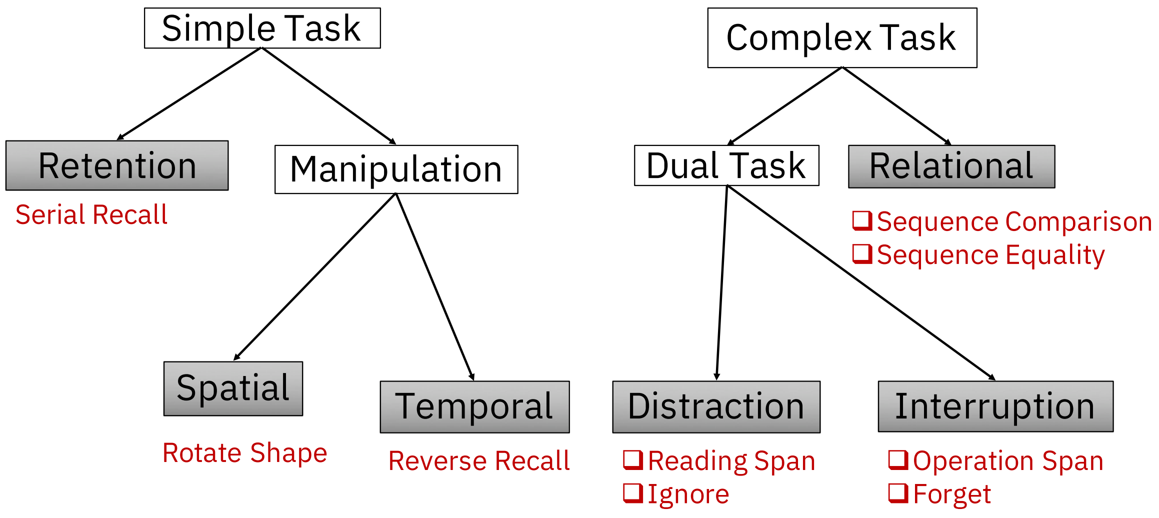


Figure 7 – Taxonomy of working memory tasks

controller, an *external memory* and an *interface* between the two [44, 45]. The controller generates parameters that allow the interface to pay *attention* to specific locations in memory and then read and write to them. The procedure is sketched in Algorithm 1. In the following, the detailed equations of DWM are exposed:

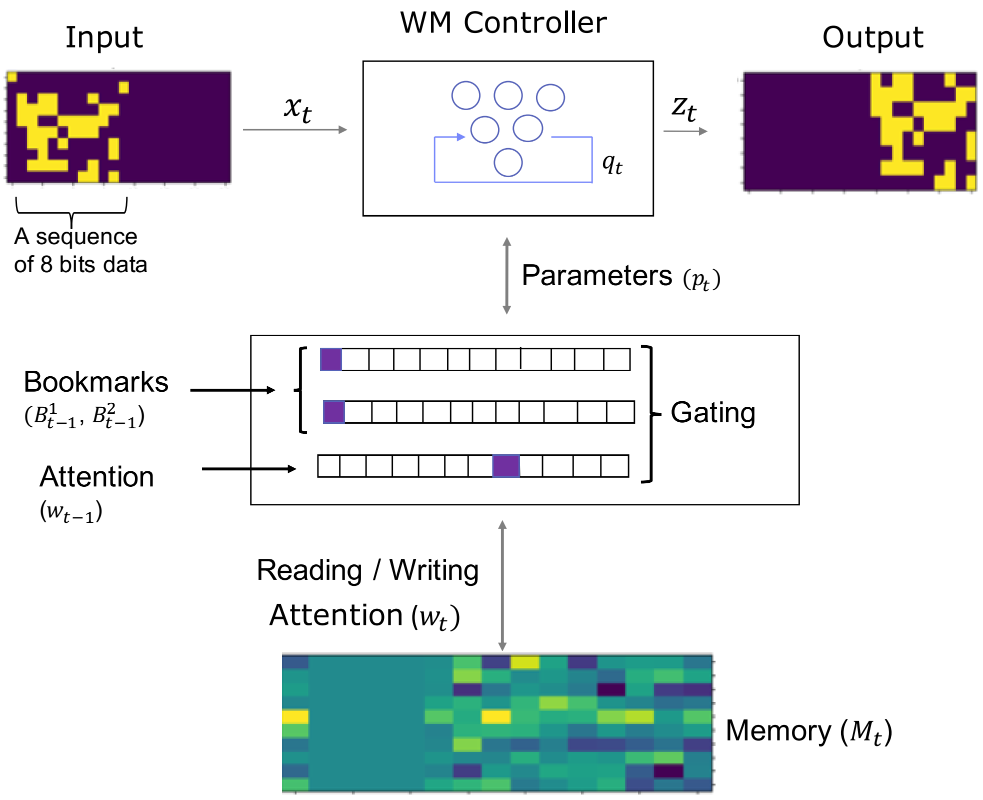


Figure 8 – Illustration of the operation of the DWM Model

3.4.1 Read from the memory

Differentiable models with addressable external memory read from the memory M with N_M addresses through with soft attention using the formula (e.g. [42, 22]):

$$r_t = M_t w_t, \tag{2}$$

where r_t is a vector read from memory.

3.4.2 Memory update

There are several schemes for memory update, e.g. using least recently used access [38, 22]. In DWM we decided to use the simple erase–add scheme derived from NTM [18]:

$$M_t = M_{t-1} \circ (E - w_t \otimes e_t) + w_t \otimes a_t \tag{3}$$

where E is a matrix of all ones, e_t and a_t are vectors of content to be erased and added to memory, respectively. e_t, a_t and w_t are emitted by interface mechanisms directed by a controller network.

3.4.3 Controller

Controller processes inputs in order to produce outputs and interface parameters. In DWM we use a single-layer recurrent neural network controller:

$$h_t = \sigma(W_h[x_t, h_{t-1}, r_{t-1}]) \tag{4}$$

where x_t denotes the current input and h_{t-1} and r_{t-1} are the hidden state and vector read from memory in the previous time step, respectively. To prevent the controller from acting as a separate working memory, the hidden state size is chosen to be smaller than that of a single input vector (in all of our experiments it was set to 5). The output logits, y_t and interface vector P_t are produced similarly as:

$$y_t = W_y[x_t, h_{t-1}, r_{t-1}] \tag{5}$$

$$P_t = W_P[x_t, h_{t-1}, r_{t-1}] \tag{6}$$

W_h, W_y and W_P are the only trainable parameters of our DWM model. The interface vector P_t contains all of the parameters that control reading, writing, and the attention mechanisms. Denoting the unprocessed parameters from the interface with a hat, the full list of parameters is as follows:

- The write vector $a_t \in \mathbb{R}^{N_M}$
- The erase vector $e_t = \sigma(\hat{e}_t) \in [0,1]^{N_M}$
- The shift vector $s_t = \text{softmax}(\text{softplus}(\hat{s})) \in [0,1]^3$
- The bookmark update gates $g_t^i = \sigma(\hat{g}_t^i) \in [0,1]^{N_B - 1}$
- The attention update gate $\delta_t^i = \text{softmax}(\hat{\delta}_t^i) \in [0,1]^{N_B + 1}$
- The sharpening parameter $\gamma = 1 + \text{softplus}(\hat{\gamma}) \in [1, \infty]$

Algorithm 1 Operation of the Differentiable Working Memory

```
1: Initialize the hidden state $h_0$ and memory array $M_0$
2: Initialize the attention vector $w_0$ and bookmark vectors $\{B_0^i : i = 1, 2, \ldots, N_B\}$
3: for $t \in \{1, 2, \ldots, T\}$ do
4:     Read from the memory: $r_{t-1} \leftarrow M_{t-1}w_{t-1}$
5:     Compute a new controller hidden state and parameters: $h_t, P_t \leftarrow \phi(x_t, r_{t-1}, h_{t-1})$
6:     Write and erase from memory: $M_t \leftarrow \text{edit}(w_{t-1}, P_t, M_{t-1})$
7:     Update attention and bookmarks: $w_t, \{B_t^i\} = \text{attn\_control}(w_{t-1}, \{B_{t-1}^i\}, P_t)$
8:     Shift attention linearly: $w_t = \text{convolution}(w_t^g, P_t)$
```

3.4.4 Attention control

For our model to act like human working memory, it must have the same characteristics, the most important of which is that working memory is accessed and written to sequentially [39]. One condition is that it has to use circular convolution shifting enabling it to shift attention over memory. For that purpose we use a mechanism similar to the one used by Neural Turing Machine (NTM) [18]:

$$w_t = \mathit{convolution}(w_t^g, s_t) \tag{7}$$

where w_t^g and w_t are the vectors of attention weights over cells in memory at time t before and after shifting, and s_t is a shift vector outputted by the controller. The other condition that sequentiality suggests is that the read and write operations should jointly share a single attention so that all access happens in sequential order. As is standard with the linear shifting in memory enhanced neural networks, we also apply a weight sharpening step as detailed in Graves *et al.* [18].

Working memory also involves some limited model of when information was attended to in the past [39]. This characteristic suggests that human working memory could contain some limited internal record of its past attention. In DWM, this is accomplished by storing *bookmarks* of system's attention at previous time steps. This is recorded in N_B bookmark vectors $\{B_t^i : i = 1, 2, \ldots, N_B\}$ at time t. At each time step, the DWM must decide whether to remember its previous attention w_{t-1} by recording it in a bookmark, as:

$$B_t^i = g_t^i w_{t-1} + (1 - g_t^i)B_{t-1}^i, \tag{8}$$

where the gating parameter g_t^i is emitted by the controller. Additionally, the DWM keeps one bookmark fixed to the initial attention at time $t = 0$ so the model maintains a fixed reference frame. As discussed below, we found even when limiting the bookmark memory to only two bookmarks that we could still solve all tasks.

The DWM must also decide before moving sequentially whether it wishes to return to a previous bookmark. For this purpose we once again use a gating mechanism, this time in a slightly more sophisticated form:

$$w_t^g = \delta_t^0 w_{t-1} + \sum_{i=1}^{N_B} \delta_t^i B_{t-1}^i, \tag{9}$$

where $\delta_t^i, i = 0 \ldots N_B$ are gating parameters emitted by the controller. These gating parameters are scalars, normalized using a softmax function.

The DWM attention control incorporates the presented mechanisms using the combination of equations (9),(8),(7) in order.

3.5 Formal description of tasks

In order to investigate the capabilities of working memory (WM) such as retention, forgetting and ignoring we introduce a battery of tasks presented in 1. These tasks are motivated by, and have a direct correspondence to the categorization shown in 7, but modified so that they are agnostic to audio/visual processing. In addition to the classical psychometric tasks, we introduce additional tasks that will test the effectiveness of attention control in memory (Ignore, Forget and Scratch Pad).

The input to every WM task is a time-indexed stream of items. At a higher level, we view the input as a *concatenation* of various subsequences that represent different functional units of processing. Additionally we use a constant-sized set of special items (called command markers) to both mark the beginning of a subsequence as well as indicate its functional type. Important note is that the system does not know a priori what kind of operation is associated with a given type of marker and must learn that from data. We ignored markers in Table 1 to keep the description simple. Also, note that such markers are also commonly employed in the psychometric tests (e.g. see [33]).

For all Simple tasks, there is only one type of subsequence, and the output will be reproduced from the memory with or without manipulation. The | sign indicates the delay between input and output of the primary subsequence(s). In our notation, $x^{\circlearrowleft}$ represents doing a circular shift of the bit representation of element x by half the number of bits. The Complex tasks may involve a secondary set of subsequences, which may or may not require immediate output as indicated in the Forget and Operation Span tasks.

3.6 Definition of input sequences

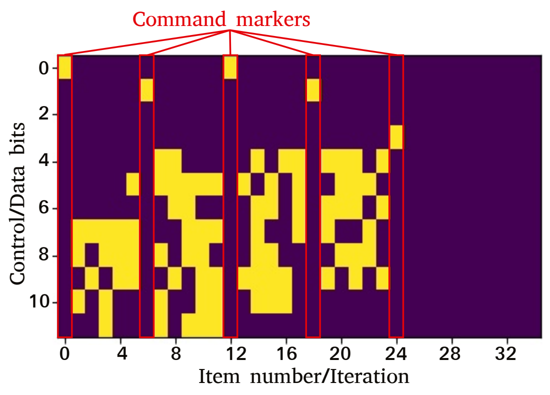
The input to every WM task is a time-indexed stream of items (Fig. 9). At a higher level, we view the input as a *concatenation* of various subsequences that represent different functional units of processing. For all "Simple" tasks, there is only one type of subsequence, and the output will be reproduced from the memory with or without manipulation. The "Complex" tasks may involve a secondary set of subsequences, which may or may not require immediate output as indicated in the "Forget and "Operation Span" tasks.

In the actual encoding of the input, we use a constant-sized set of special items (called Command Markers, please refer to Fig. 9a) to both mark the beginning of a subsequence as well as indicate its functional type. Important note is that the system does not know a priori what kind of operation is associated with a given type of marker and must learn that from data. We ignored markers in Table 1 to keep the description simple.

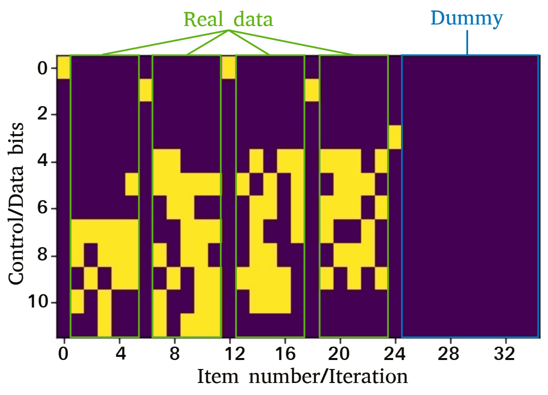
Each subsequence is either real data or dummy (Fig. 9b). The dummy subsequences represent elements in the input processing where a suitable target of the *same* length needs to be output. In 1 we denote this by the ␣ symbol. Introduction of dummies

	Task		(I)input/(O)output sequences	Notes
Simple	Serial Recall	I:	$x_1x_2\dots x_n \mid \sqcup\sqcup\dots\sqcup$	Store the input and recall it
		O:	$\sqcup\sqcup\dots\sqcup \mid x_1x_2\dots x_n$	in the same order
	Reverse Recall	I:	$x_1x_2\dots x_n \mid \sqcup\sqcup\dots\sqcup$	Store the input and recall it
		O:	$\sqcup\sqcup\dots\sqcup \mid x_nx_{n-1}\dots x_1$	in the reversed order
	Rotate Shape	I:	$x_1x_2\dots x_n \mid \sqcup\sqcup\dots\sqcup$	Rotate each element of sequence
		O:	$\sqcup\sqcup\dots\sqcup \mid x_1^{\circlearrowleft}x_2^{\circlearrowleft}\dots x_n^{\circlearrowleft}$	
Complex	Reading Span	I:	$\mathbf{x}_1\dots\mathbf{x}_k \mid \sqcup\dots\sqcup$	Recall z_i being the last
		O:	$\sqcup\dots\sqcup \mid z_1\dots z_k$	element of $\mathbf{x}_i$
	Forget	I:	$\mathbf{x}_1\mathbf{y}_1\sqcup\dots\mathbf{x}_k\mathbf{y}_k\sqcup \mid \sqcup\dots\sqcup$	Recall every $\mathbf{y}_i$ immediately,
		O:	$\sqcup\sqcup\mathbf{y}_1\dots\sqcup\sqcup\mathbf{y}_k \mid \mathbf{x}_1\dots\mathbf{x}_k$	recall all $\mathbf{x}_i$ in the same order
	Operation Span	I:	$\mathbf{x}_1y_1\sqcup\dots\mathbf{x}_ky_k\sqcup \mid \sqcup\dots\sqcup$	Rotate y_i immediately,
		O:	$\sqcup\sqcup y_1^{\circlearrowleft}\dots\sqcup\sqcup y_k^{\circlearrowleft} \mid \mathbf{x}_1\dots\mathbf{x}_k$	recall all $\mathbf{x}_i$ in the same order
	Scratch Pad	I:	$\mathbf{x}_1\dots\mathbf{x}_k \mid \sqcup$	Return only the last $\mathbf{x}_k$
		O:	$\sqcup\dots\sqcup \mid \mathbf{x}_k$	
	Ignore	I:	$\mathbf{x}_1\mathbf{y}_1\dots\mathbf{x}_k\mathbf{y}_k \mid \sqcup\dots\sqcup$	Ignore $\mathbf{y}_i$, recall $\mathbf{x}_i$
		O:	$\sqcup\sqcup\dots\sqcup\sqcup \mid \mathbf{x}_1\dots\mathbf{x}_k$	

Table 1 – The Working Memory Tasks used in our experiments



(a) Command Markers



(b) Real data and dummy subsequences

Figure 9 – Input sequence example

enables *delays* in the input processing for capturing memory retention and other aspects of WM tasks.

Throughout the experiments, we fixed the input item size to be 8 bits supplemented by additional 2–4 control bits, depending on the task. In order to ensure that the sequences returned by our data generators for training, validation and testing are distinctive, we followed the conditions for length and number of subsequences presented in 2. For the memory based models, the memory size was chosen dynamically for each episode to equal the (common) length of the input sequences within the batch.

	Simple tasks			Complex tasks		
	Training	Validation	Testing	Training	Validation	Testing
Subsequence Length	1–10	100	1000	1–6	20	20
Number of subsequences	1	1	1	1–3	5	50

Table 2 – Parameters used during data generation

3.7 Experimental results

We evaluated the performance of DWM on the proposed tasks and compared it to two models: LSTM (Long Short-Term Memory) [27], considered as a classical baseline for sequential problems, and DNC (Differentiable Neural Computer) [19] being the state-of-the-art memory augmented neural model. In our implementation we used the Pytorch framework [5] [35]. During training we used the Adam (Adaptive Momentum) optimizer [31] and (average) binary cross-entropy as the loss function. We apply early stopping based on validation loss (10^{-4}). Additionally we terminate training when the number of training episodes reach 100,000, where a single episode involves processing a batch of sequences. The size of batch was a hyper-parameter that was tuned for each model along with training rate using validation loss. The exact parameters describing each tasks can be found in the Supplemental Information.

It is well-known that neural networks augmented by external memory learn algorithms to process tasks rather than memorizing data [18, 19]. Therefore, to determine the robustness of our models we train them on sequences of lengths of order 10 and then validate and test them on sequences of size 100 and 1000, respectively. As seen in 3, the DWM, the LSTM and the DNC are all capable of learning each of the tasks for sequences of order 10. To train on these tasks, the DWM required on average the fewest number of sequences with only Forget and Operation Span requiring more than 10000 batches of size 16 to converge 10. The DNC required the most number of sequences with only Serial Recall, Rotate Shape and Ignore converging with fewer than 20000 batches of size 16 and the rest requiring between 20000 to 90000 batches.

Only the DWM was able to successfully generalize to sequences of size 1000 even with only 1066 trainable parameters. The DNC with its 4,792 trainable parameters generalized to sequences of length 100 just on Serial Recall, Scratch Pad and Rotate Shape and did not fully generalize on sequences of length 1000 for any of the tasks. Despite that the LSTM possessed over 5 million trainable parameters it was only able to generalize to sequences of length 100 on Serial Recall.

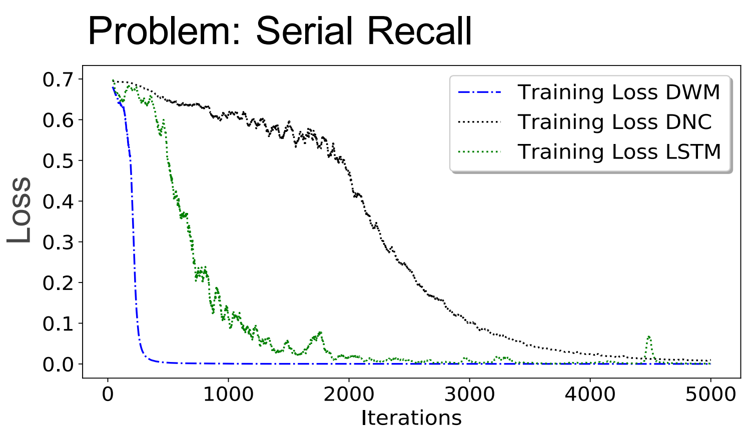
The DWM performance on these tasks indicates that it is actually acting as a form of artificial working memory. Human working memory's operation is straightforward compared to episodic memory and therefore it is more effective at solving the relatively simple working memory tasks. We hypothesize that the DNC, which was explicitly designed to mimic an episodic memory [19], did not learn these simple working memory tasks as effectively because the complexity required for true episodic remembrance

[5]Pytorch is a scientific computing framework that offers wide support for machine learning algorithms, it was built at the top python

Task	Best Train Accuracy Seq. Size 10 [%]			Validation Accuracy Seq. Size 100 [%]			Test Accuracy Seq. Size 1000 [%]		
	LSTM	DNC	DWM	LSTM	DNC	DWM	LSTM	DNC	DWM
Serial Recall	100	100	100	100	100	100	50.20	64.64	100
Reverse Recall	100	100	100	52.96	50.62	100	50.38	50.15	99.76
Rotate Shape	100	100	100	52.17	100	100	50.20	60..91	100
Reading Span	100	100	100	50.90	53.36	100	50.44	49.04	91.88
Forget	100	100	100	55.90	69.36	98.92	50.45	49.94	94.11
Operation Span	100	100	100	58.16	79.22	99.95	51.26	53.61	99.64
Scratch Pad	100	100	100	71.28	100	100	70.02	74.97	100
Ignore	100	100	100	56.13	69.32	100	50.89	49.99	90.05

Table 3 – Summary of experimental results. The first column is the average of the best accuracy achieved during training for each run. The second column is the average of the best validation accuracy for each run. The third column is the average of the test accuracy for the model parameters with the best validation accuracy. For the DNC and LSTM, the validation loss did not reached threshold for some tasks (i.e. training was stopped at 100,000 episodes), in which cases we decided to include the best scores of single best (but diverged) models

interfered with learning tasks that do not require episodic memory.



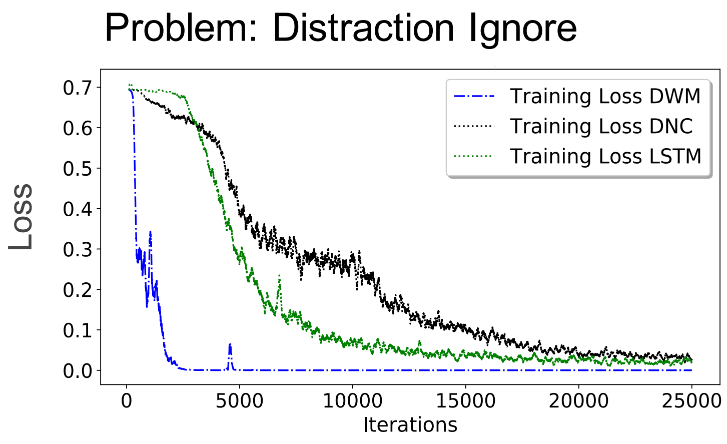


Figure 10 – Training Loss

3.8 Analysis of strategies for solving tasks

In contrast to long-term memory, working memory has an extremely small capacity. Therefore, dealing with interference (e.g. distractions) is a major challenge for memory capacity and attention control. As mentioned earlier, ignoring distractions without encoding them in the memory is arguably the best strategy to minimize memory consumption. On the other hand, for a complex task with an interruption (i.e. multi-tasking), the secondary task cannot be ignored and may require extensive memory usage.

In this case, the best strategy might be to forget (e.g. erase or overwrite) the secondary information as soon as possible in order to maintain sufficient memory capacity for the main task. Ignoring and forgetting may be good strategies for the efficient use of capacity but they also complicate the attention control and addressing during writing in the memory.

During the training and testing of all of the tasks reported in Table 3, we provided sufficient memory size, so that the system could store all the encoded input items in the memory (if it has chosen to). However, limitation of the memory size can force the system to develop more memory efficient strategies, thus we decided to investigate that issue further.

3.8.1 Strategies for the Scratch Pad task

The goal of the Scratch Pad task is to recall only the last input subsequence. Given the DWM mechanisms, we expect two possible strategies for the model to learn in order to solve this task.

The "Expand" strategy exploits the fact that memory can be used in a similar way to a circular buffer, storing each consecutive subsequences one after the other in the memory. In this case the model should write each subsequence, then place the dynamic bookmark (i.e. the one that does not have fixed attention) at the start of given subsequence, and then update the bookmark position to the beginning of the next subsequence. Finally, when the model receives a command marker indicating it needs to recall, it should recall the attention associated with that dynamic bookmark and then retrieve consecutive items one by one using circular convolution.

The "Overwrite" strategy for the Scratch Pad relies on the fact that when a new subsequence appears, the elements from the previous one can be discarded. The model could exploit this by learning to recall attention stored in the static bookmark (pointing at address 0) every time it processes a command marker denoting the next subsequence, which will result in overwriting the previous subsequences until the system is told to recall. This strategy is clearly more memory efficient, as the system reuses the same addresses and overwrites the memory repeatedly.

To our (initial) surprise, the model *always* developed the "Overwrite" strategy, irrespective of the memory size (i.e. as long as the memory size was sufficient to fit all the encoded items of a single subsequence). A typical example run of an early training episode is presented in Fig. 11. Please note that memory addresses 1 and 2 remain unchanged and the model stores consecutive items of subsequences $\mathbf{x}_1$ to $\mathbf{x}_5$ in the same addresses 3-7. After analyzing several runs, we hypothesize that overwriting was simpler to learn for this task because: a) both for storing and recalling the command markers, the model had to learn exactly the same behavior: recalling the attention stored in the static bookmark, b) for every other input item (data and dummy) it had to shift by one address location with the circular convolution. As a result, it could converge rapidly by disregarding the control (update, recalling) of the dynamic bookmark (in the last training episodes the dynamic bookmark was typically "following" the current attention, despite it wasn't recalled at all).

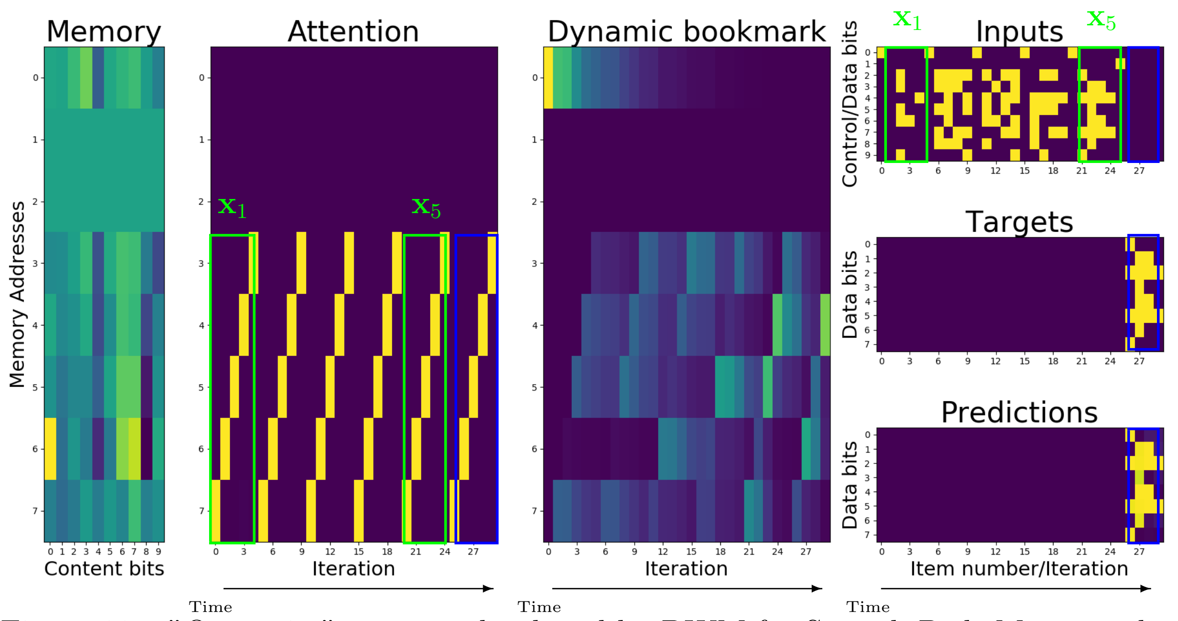


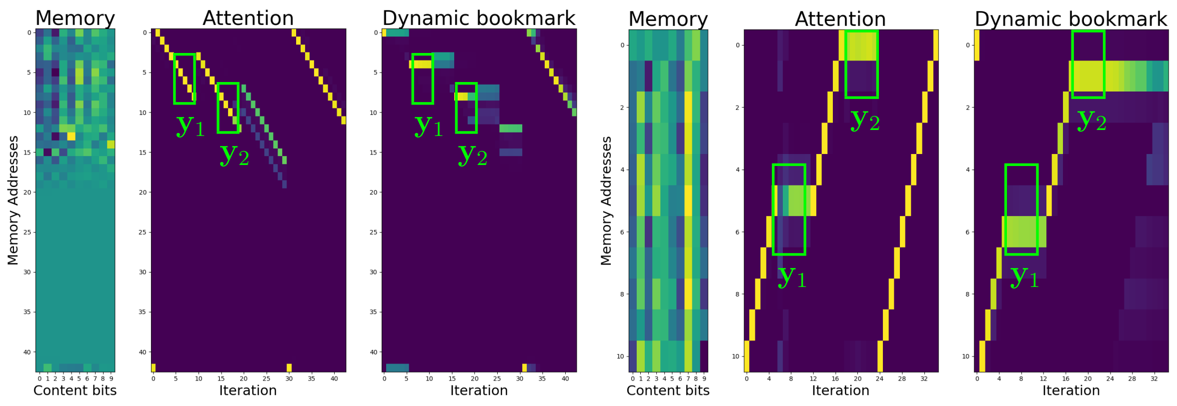
Figure 11 – "Overwrite" strategy developed by DWM for Scratch Pad. Memory plot contains a snapshot of the memory content from the last iteration, whereas the other ones present concatenation of states from consecutive iterations (evolution in time)

3.8.2 Strategies for the Ignore task

The main goal of the Ignore task is to test the retention capabilities of the system in the presence of distractors. For this task the input consists of two types of subsequences $\mathbf{x}$ and $\mathbf{y}$, where the system is supposed to ignore all $\mathbf{y}_i$ and at the end recall $\mathbf{x}_i$ one by one in the order of their appearance. The task can be solved with two strategies which we call "Overwrite" and "Skip".

The "Overwrite" strategy involves overwriting of the distractors, as we have already observed during the Scratch Pad task. It assumes that system will store the consecutive items in memory and use the bookmark for moving its attention to the first address containing $\mathbf{y}$ to be overwritten. The difference is, however, that in this case the model must learn to use the dynamic bookmark for that purpose. Our experiments with sufficient memory have shown that the system can learn this strategy. An example plot from one of the final training episodes (Fig. 12a) shows that the dynamic bookmark retains its attention while processing items from $\mathbf{y}_1$ and $\mathbf{y}_2$. As soon as the command marker indicating $\mathbf{x}$ appears, the model jumps back its attention to the dynamic bookmark and starts to overwrite. Finally, when the recall marker appears, it recalls the attention stored in the static bookmark.

The "Skip" strategy to solve this task would be to ignore elements within the $\mathbf{y}$ subsequences and *skip* writing these into the memory. Our experiments with limited memory have shown that the model could also learn this strategy. Example plots from the final episode from one of the training runs is presented in Fig. 12b. Please notice that in this case the model has learned to keep its attention focused on a single address for all items of $\mathbf{y}_i$ and shift attention only for items belonging to $\mathbf{x}_i$.



(a) "Overwrite" strategy (sufficient memory) (b) "Skip" strategy (insufficient memory)

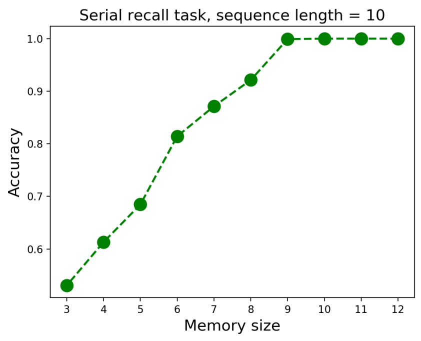
Figure 12 – Strategies developed by DWM for solving the Ignore task (two different training runs)

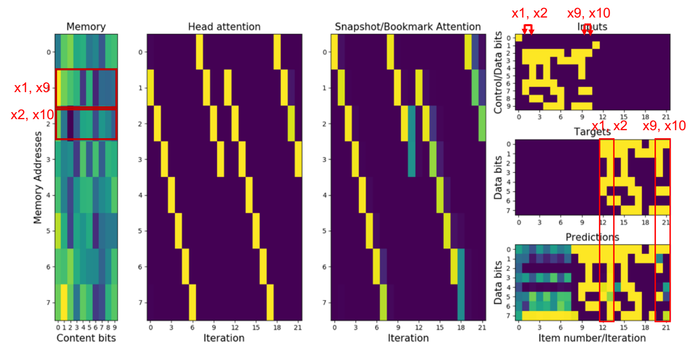
3.9 Ablation study

One of the essential properties of human working memory is its limited capacity to store information, it can range from 3 to 5 elements depending on the complexity of the task [40]. With respect to DWM, the memory size is an adjustable parameter that the user can set. In this study, we were interested in analyzing the case where this parameter is set below the required memory size (insufficient memory), this was motivated by the fact that most of real datasets are very large and may exceed DWM memory. As it can be seen in figure 13a using the serial recall task with a sequence length equal to 10 and varying the memory size from 13 cells to 3 cells, the DWM similarly to the human working memory can remember a number of elements that correspond to its capacity for example if a memory size equal to 8 and a sequence length equal 10 the measured accuracy is 90%.

The strategy developed by the DWM to solve the serial recall task is presented in figure 13b, it shows that the DWM can store and recall the six middle elements effectively and recall the first and last two elements with some deficiency, this is different from what would a human working memory do, which is recalling the last eight elements if it is characterized by the same capacity. This difference is resulting essentially from the fact the elements stored in the human working memory are retained for a short period of time –retention time, which makes the human working memory remembers only the last elements. This is not the case for DWM as the elements can be theoretically retained for an infinite time, this highlights one issue of the DWM design that needs to be addressed in a future implementation.

The strategy developed by the DWM is still interesting, as it can be seen in figure 13b, although the elements x_1 and x_2 are overwritten by x_9 and x_{10}, the DWM can still remember these two element with slight deficiency exploiting the similarity in the structure of these four elements. This opens new research axis on whether we can design a learnable mechanism that compresses the data before storing it.





(a) Ablation study, varying the memory size between 13 and 3.

(b) Plot showing the strategy developed by the DWM to deal wit the ablation.

3.10 Summary

We decided to study human working memory in order to improve artificial learning models. Our analysis allowed us to distill a novel attention control mechanism (bookmarks), which is quantitatively more robust and more data-efficient than previously published work (i.e. DNC, LSTM) based on the tasks tested in this study. Moreover, our model showed generalization to sequences two orders of magnitude longer than the training regime. We also studied the behavior of our model during learning and found out that for complex tasks, it has developed efficient strategies to control attention and use its memory resources.

4 MI-Prometheus

4.1 Introduction

Machine Learning models, especially deep Neural Networks, have made tremendous progress in recent years, growing continuously in different areas such as Computer Vision, Text Comprehension, Question Answering etc. These neural networks require a training setup and procedure that may need a considerable amount of engineering work to ensure consistency, results repeatability and precision. These requirements often lead to an implementation designed for a specific pair of model and problem, making it hard to run new experiments and compare the results. This problem clearly emerged when trying to test the DWM model against different kind of problems – such as the algorithmic tasks, but also when testing other models on the same problems to establish baselines results.

To address these issues, we started to design *MI-Prometheus*, which is a framework based on PyTorch [30] - an open source Machine Learning Python library - that standardizes the interface that connects together the components needed in a deep learning system: problems, models architectures, training/testing configurations etc. This unifies the training and testing paradigms, enabling experiments reproducibility and making easy the run of several models on the same dataset, or applying the same model to several datasets.

MI-Prometheus training and testing mechanisms are no longer pinned to a specific model or dataset. When trying to test a new model, the main change will impact the body of the model architecture, and not the overall mechanism. We believe that designing such a framework presents several advantages:

- A centralized code base lowers the probability of mistakes,
- New models and problems can be added with minimal changes to the code,
- A training run of a given model on a (compatible) given dataset is defined in a configuration file. These configurations files are both human and machine readable, easy to edit and easy to generate (we are using the YAML markup language).
- Automation of training/validation [6] /testing pipelines.

4.2 Overall design

The framework is composed of a set of problem classes, model classes and workers. The worker retrieves the set of parameters from the configuration file, which specifies the model and problem to use, see figure 14. This configuration file also contains information about the training (e.g. batch size, number of epochs, optimizer, learning rate etc), validation and testing setups. The worker hands batches of the dataset (generated by the problem class) to the model, which in turn outputs its predictions and hands them

[6]Validation is an inference step which is done during the training to check if the model is overfitting the training data

back to the problem class to compute the loss. The worker then can call the optimizer to update the model weights, and eventually display visualization windows.

The workers are independent from the problem and model classes, meaning that they can handle different dataset structures, thus making the training and testing setups uniform across the problem and model hierarchies.

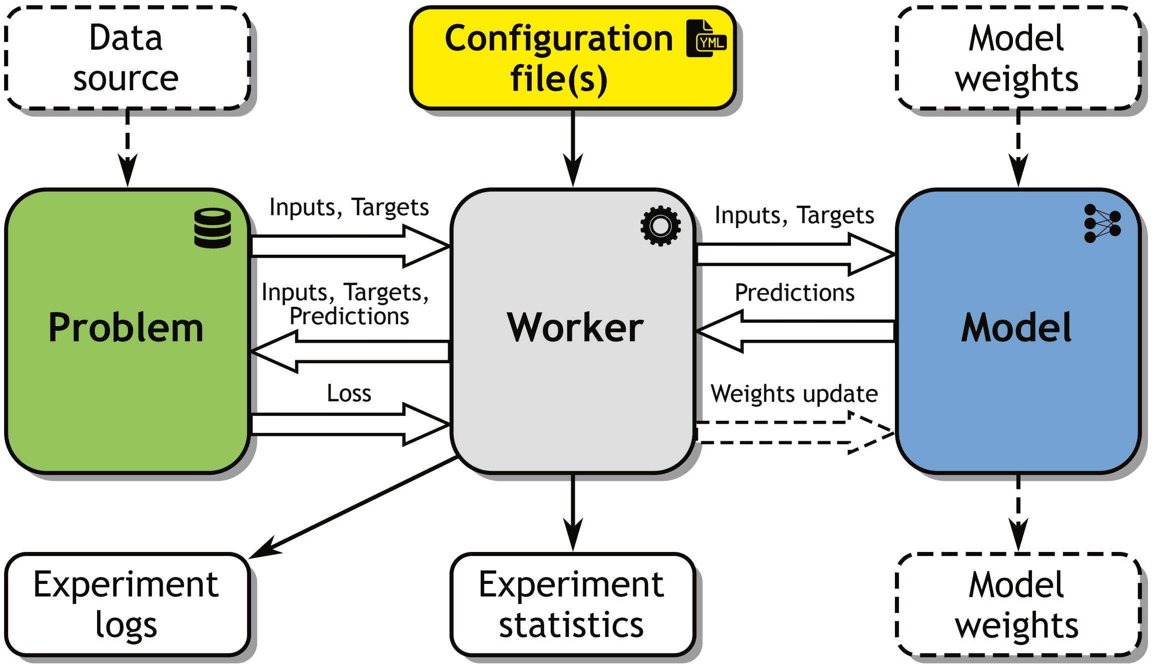


Figure 14 – Core concept of MI-Prometheus [source: MI group]

4.3 Main components

There are four important building blocks of the framework:

4.3.1 Problems

A problem class defines a particular dataset. Its main role is to collect the data (e.g. download them or load them from disk), handle the batches generation, and compute metrics such as the loss and accuracy, which are dependent on the type of problem at hand.

The problem classes were designed such that they benefit from inheritance see figure: 15. We have designed base classes that contains features used by every problem class, like the loss computation and logging. Then, we are able to classify the problems based on their type: algorithmic tasks, sequence-to-sequence (e.g. like Language Translation) problems, Image Classification (e.g. MNIST, CIFAR and ImageNet), Visual Question Answering (VQA, datasets: CLEVR) etc. This speeds up the integration of a new problem class as similar problem classes share characteristics (same loss function, same metrics, same format of data etc.). Hence, integrating it correctly in the problem classes hierarchy limits the amount of code to write and thus speeds up the workflow.

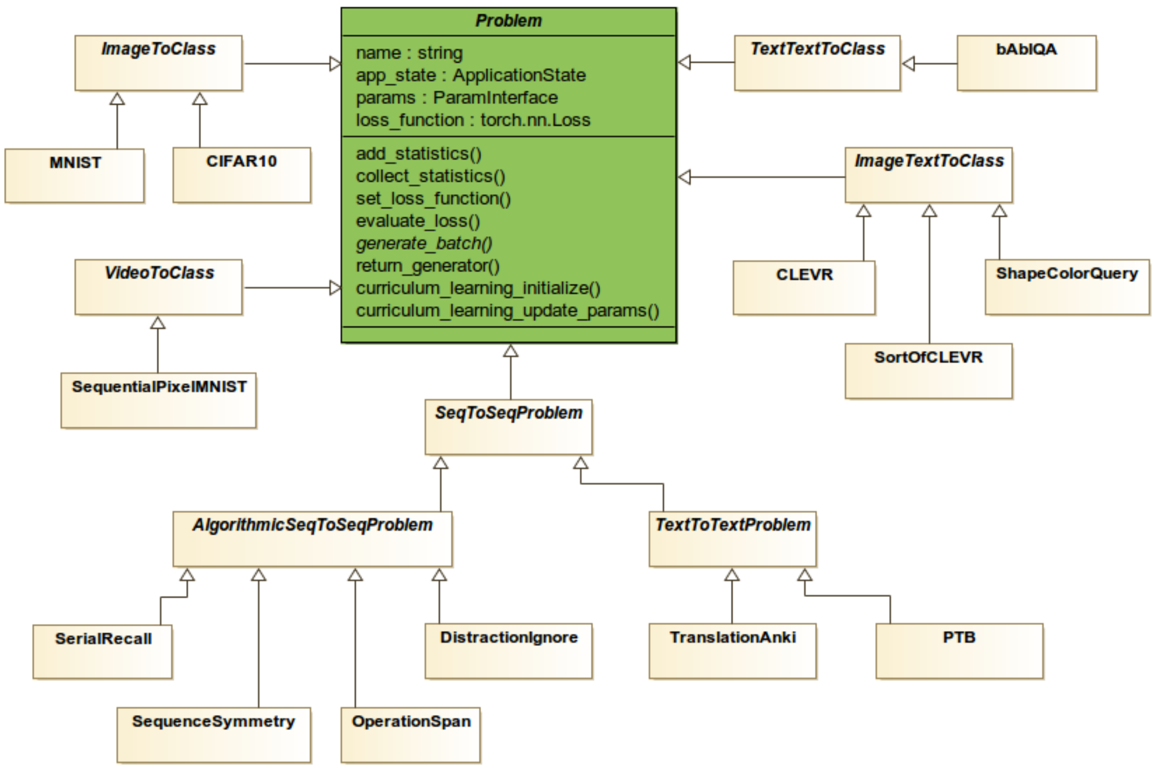


Figure 15 – Problem class hierarchy [source: MI group]

Here is a short description of the problem types we have implemented so far:

- Image to Class: Label a given image (examples of datasets: MNIST, CIFAR)
- Image and Text to Class: Answer a given question about an image (example of dataset: CLEVR)
- Sequence to Sequence: Where both the input and output are a sequence of items (example of dataset: Neural Machine Translation), usually used with a Recurrent Neural Net.
- Video to Class: Questioning about certain events that occurred during a movie (example: Sequential MNIST).

4.3.2 Model

A Model represents here a (deep) neural network (i.e. a set of differentiable mathematical equations). Similar to the problem classes, the model classes are organized following a hierarchy, to maximize code reusability, see figure 16. All model perform the same: they take a tensor of data as input, perform mathematical operations on it and hand back a tensor of prediction that will be matched against the ground truth. We try to keep the models independent of the problem classes, as we initially wanted to test DWM against several algorithmic tasks. Naturally, not all models will work with all problems:

A Convolutional Neural Net is not designed for an algorithmic or sequence-to-sequence problem. The following models are present in the current version of the framework:

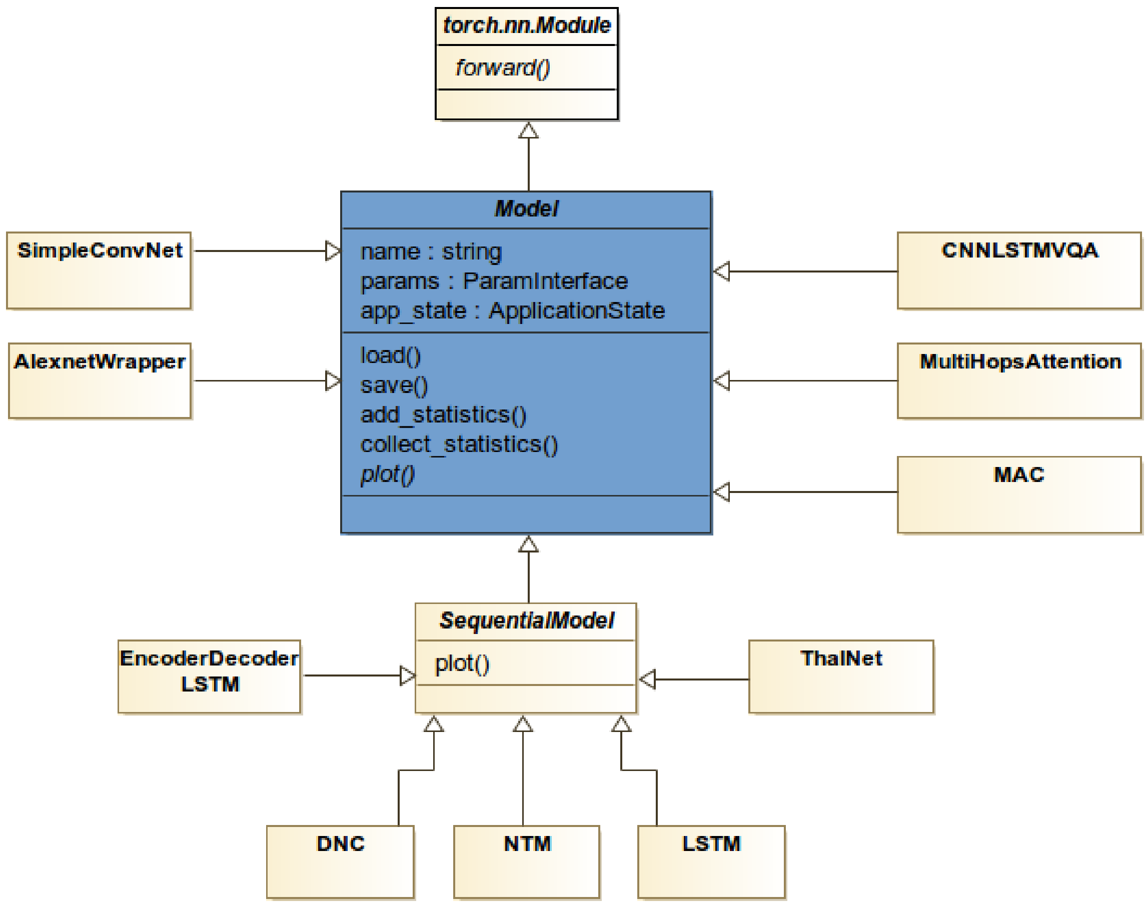


Figure 16 – Model class hierarchy [source: MI group]

- Memory-Augmented Neural Networks: Differentiable Working Memory, Neural Turing Machine, Differentiable Neural Computer,
- Recurrent Neural Networks: LSTM, Memory Attention Composition (MAC),
- Models for VQA: Relational Network, CNN + LSTM, Stacked Attention Networks[7]

4.3.3 Workers: Trainer and Tester

A worker is a script that will execute a certain task given a model and a problem class. They are related to either the training or the testing procedure. The trainer is the main script of the framework. After reading the configuration file, it instantiates the problem and model classes, and loop over the specified number of epochs, feeding every time a batch of inputs to the model, collecting the predictions and performing an optimization step (i.e. updating the model weights using back-propagation). Several options are available for this script, to indicate whether we activate visualization or

[7] is RN really working? besides shouldn't model names be coherent with class names?

not, whether we log the metrics using TensorBoard etc. Computations on GPUs are supported (using CUDA). The trainer also has the ability to validate the model at a given frequency, or at the end of the training. All events are logged to an experiment folder, which contains among others the trained model. We have also implemented the ability to run several experiments simultaneously on several GPUs. The tester mainly runs an inference step using a trained model and a problem class. As its name indicates, it allows testing the model against previously unseen samples, in order to assess the generalization capabilities of the model.

It is planned to add support for several validation schemes (e.g. K-fold cross validation), hyper-parameter search etc.

4.3.4 Configuration management

An important aspect of using Machine Learning software to develop a new model is configuration management. This is critical as we need results repeatability to ensure statistical validation, experiments definition (it should be easy to change the value of a given parameter), and models comparison (limiting the number of varying factors is important when benchmarking models on the same problem). To organize the parameters setup within the framework, we implemented a ParameterRegistry singleton* that stores all parameters across the whole framework. These parameters are organized into a tree (i.e. sections for training / testing / model ...), and it is possible to create multiple views of this tree, such that it eases the overwriting of one specific parameter (leaf within the tree) to run a new experiment for instance. We consider 2 kinds of

- Default values: parameters that usually don't change from one training run to another, or when the user is unaware of their use-cases.
- Custom values: Can overwrite an existing value.

4.4 Summary

MI-Prometheus has helped us accelerate the several experiments we ran with the DWM. By also helping integrate other models from different domains of Deep Learning into one unique standardized code base, it enables research ideas to be quickly tested and shared. In future work, we want to extend *MI-Prometheus* capacities to handle more models and problems and further optimize its different tools.

5 Conclusion

The work performed at IBM Almaden Research Center within the Machine Intelligence group, on the Differentiable Working Memory, along with a theoretical background has been presented. The main goal was to extend the capabilities of artificial intelligence models to more than just remembering patterns in data, but also to understand the meaning of structures hidden within it. Motivated by the role of the different memories in the human brain and how they are correlated with higher order human cognitive behaviors, we specifically target the working memory which was proven to be responsible of many domains of cognition such as planning, solving problems, language comprehension and understand relations in visual scenes. Using the rich literature in cognitive psychology about working memories, which explains their properties and underlying mechanisms, we could design the DWM model, which successfully imitates the functional characteristics of the working memory, such as retention and processing of information stored in memory. This was verified using a battery of working memory tasks inspired by the psychology literature. The work on DWM led to inventing a novel attention control mechanism (bookmarks), which is a simple mechanism that retains relevant attention in past. Moreover, our model showed generalization to sequences two orders of magnitude longer than the training regime. Our analysis in depth showed that for complex tasks our model developed efficient strategies to control attention and use its memory resources. In the ablation study, we analyze the performance of DWM when it is given insufficient memory, this study showed some deficiencies of the DWM model that need to be addressed in a future implementation and opens new research areas about data compression.

Another piece of work that was developed within the scope of this thesis is the *MI-Prometheus* framework, which unifies the training and testing paradigms of machine learning models, enabling experiments reproducibility and making easy to apply the DWM model on different algorithmic tasks, or run different models on the same tasks to establish baselines.

In our future work we plan to use our working memory model as one component of more complex system and combine it with long-term memory and expand the capabilities to solve tasks that require both working and episodic memory.

References

[1] Stanislaw Antol et al. "Vqa: Visual question answering". In: *Proceedings of the IEEE international conference on computer vision.* 2015, pp. 2425–2433.

[2] Alan Baddeley. "Working memory: Looking back and looking forward". In: *Nature Reviews Neuroscience* 4.10 (2003), pp. 829–839. ISSN: 14710048. DOI: `10.1038/nrn1201`. eprint: `arXiv:1408.1149`.

[3] Hyeong Soo Chang et al. "Google Deep Mind's AlphaGo". In: *OR/MS Today* 43.5 (2016), pp. 24–29.

[4] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[5] Wesley C Clapp, Michael T Rubens, and Adam Gazzaley. "Mechanisms of working memory disruption by external interference". In: *Cerebral Cortex* 20.4 (2009), pp. 859–872.

[6] Christos Constantinidis and Torkel Klingberg. "The neuroscience of working memory capacity and training". In: *Nature Reviews Neuroscience* 17 (May 2016), p. 438. URL: `http://dx.doi.org/10.1038/nrn.2016.43%20http://10.0.4.14/nrn.2016.43%20https://www.nature.com/articles/nrn.2016.43%7B%5C#%7Dsupplementary-information`.

[7] Andrew R A Conway et al. "Working memory span tasks: A methodological review and user's guide". In: *Psychonomic bulletin & review* 12.5 (2005), pp. 769–786. ISSN: 1069-9384. DOI: `10.3758/BF03196772`. URL: `http://www.springerlink.com/index/Q13137313L395011.pdf`.

[8] Andrew RA Conway and Randall W Engle. "Working memory and retrieval: A resource-dependent inhibition model." In: *Journal of Experimental Psychology: General* 123.4 (1994), p. 354.

[9] Nelson Cowan. "The many faces of working memory and short-term storage". In: *Psychonomic Bulletin and Review* 24.4 (2017), pp. 1158–1170. ISSN: 15315320.

[10] Meredyth Daneman and Patricia A Carpenter. "Individual differences in working memory and reading". In: *Journal of verbal learning and verbal behavior* 19.4 (1980), pp. 450–466.

[11] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[12] Randall W Engle and Michael J Kane. "Executive attention, working memory capacity, and a two-factor theory of cognitive control". In: *Psychology of learning and motivation* 44 (2004), pp. 145–200.

[13] Randall W Engle et al. "Working memory, short-term memory, and general fluid intelligence: a latent-variable approach." In: *Journal of experimental psychology: General* 128.3 (1999), p. 309.

[14] David Ferrucci et al. "Building Watson: An overview of the DeepQA project". In: *AI magazine* 31.3 (2010), pp. 59–79.

[15] Norbert J Fortin, Kara L Agster, and Howard B Eichenbaum. "Critical role of the hippocampus in memory for sequences of events". In: *Nature Neuroscience* 5 (Mar. 2002), p. 458. URL: `http://dx.doi.org/10.1038/nn834%20http://10.0.4.14/nn834`.

[16] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[17] Ian Goodfellow et al. *Deep learning.* Vol. 1. MIT press Cambridge, 2016.

[18] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: *arXiv preprint arXiv:1410.5401* (2014).

[19] Alex Graves et al. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626 (2016), p. 471.

[20] Stephen Grossberg. "Recurrent neural networks". In: *Scholarpedia* 8.2 (2013), p. 1888.

[21] Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. "Memory augmented neural networks with wormhole connections". In: *arXiv preprint arXiv:1701.08718* (2017).

[22] Caglar Gulcehre et al. "Dynamic neural turing machine with soft and hard addressing schemes". In: *arXiv preprint arXiv:1607.00036* (2016).

[23] Martin T Hagan et al. *Neural network design.* Vol. 20. Pws Pub. Boston, 1996.

[24] Demis Hassabis et al. "Neuroscience-inspired artificial intelligence". In: *Neuron* 95.2 (2017), pp. 245–258.

[25] Robert Hecht-Nielsen. "Theory of the backpropagation neural network". In: *Neural networks for perception.* Elsevier, 1992, pp. 65–93.

[26] Evan Heit, Caren M Rotello, and Brett K Hayes. "Relations between memory and reasoning". In: *Psychology of learning and motivation.* Vol. 57. Elsevier, 2012, pp. 57–101.

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] John J Hopfield and David W Tank. "Computing with neural circuits: A model". In: *Science* 233.4764 (1986), pp. 625–633.

[29] Armand Joulin and Tomas Mikolov. "Inferring algorithmic patterns with stack-augmented recurrent nets". In: *Advances in neural information processing systems.* 2015, pp. 190–198.

[30] Nikhil Ketkar. "Introduction to pytorch". In: *Deep Learning with Python.* Springer, 2017, pp. 195–208.

[31] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[32] Ashleigh M Maxcey-Richard and Andrew Hollingworth. "The strategic retention of task-relevant objects in visual working memory." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 39.3 (2013), p. 760.

[33] Fiona McNab and Torkel Klingberg. "Prefrontal cortex and basal ganglia control access to working memory". In: *Nature Neuroscience* 11.1 (2008), pp. 103–107. ISSN: 10976256. DOI: 10.1038/nn2024.

[34] Erkki Oja. "Simplified neuron model as a principal component analyzer". In: *Journal of mathematical biology* 15.3 (1982), pp. 267–273.

[35] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[36] Matthew Ricci, Junkyung Kim, and Thomas Serre. "Not-So-CLEVR: Visual Relations Strain Feedforward Neural Networks". In: *arXiv preprint arXiv:1802.03390* (2018).

[37] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited, 2016.

[38] Adam Santoro et al. "Meta-learning with memory-augmented neural networks". In: *International conference on machine learning.* 2016, pp. 1842–1850.

[39] Inder Singh, Zoran Tiganj, and Marc W Howard. “Is working memory stored along a logarithmic timeline? Converging evidence from neuroscience, behavior and models”. In: *Neurobiology of learning and memory* (2018).

[40] Marilyn L Turner and Randall W Engle. “Working memory capacity”. In: *Proceedings of the Human Factors Society Annual Meeting*. Vol. 30. 13. SAGE Publications Sage CA: Los Angeles, CA. 1986, pp. 1273–1277.

[41] Edward K Vogel, Andrew W McCollough, and Maro G Machizawa. “Neural measures reveal individual differences in controlling access to working memory”. In: *Nature* 438.7067 (2005), p. 500.

[42] Jason Weston, Sumit Chopra, and Antoine Bordes. “Memory networks”. In: *International Conference on Learning Representations (ICLR)*. 2015.

[43] Jason Weston et al. “Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks”. In: *CoRR* abs/1502.05698 (2015). arXiv: `1502.05698`. URL: `http://arxiv.org/abs/1502.05698`.

[44] Wojciech Zaremba. “Learning Algorithms from Data”. PhD thesis. New York University, 2016.

[45] Wojciech Zaremba et al. “Learning simple algorithms from examples”. In: *International Conference on Machine Learning*. 2016, pp. 421–429.

Appendices

A Experiments

A.1 Number of parameters per model

During our experiments with LSTM model we have used stacked LSTM with 3 layers and 512 hidden units in each of them. DNC model used an single-layer LSTM controller with 20 hidden units. Our DWM model used RNN with sigmoid activation function and 5 hidden units. In 4 we report number of trainable parameters for each of our models.

	Models		
	LSTM	DNC	DWM
Number of model parameters	5,279,752	4,792	1,066
Learning Rate	$5 \cdot 10^{-3}$	$5 \cdot 10^{-5}$	$1 \cdot 10^{-2}$
Optimizer	Adam	Adam	Adam

Table 4 – Number of parameters of the used models

A.2 Number of runs and converged models

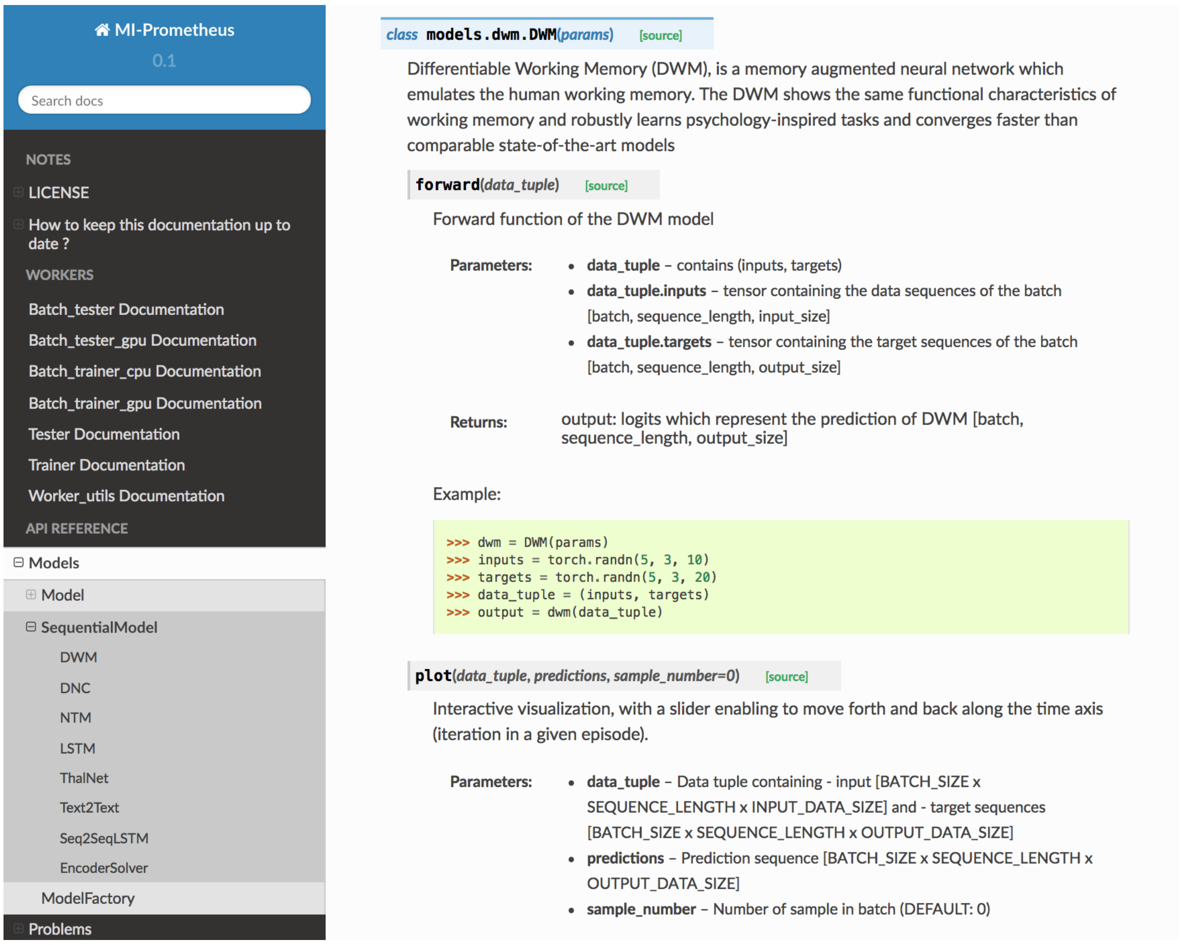
For each model and task pair we performed 10 runs. In 5 we report number of models that converged, i.e. the validation loss went below the 1e-4 threshold.

Task	Number of Successful Runs		
	LSTM	DNC	DWM
Serial Recall	0/10	10/10	10/10
Reverse Recall	0/10	0/10	7/10
Rotate Shape	0/10	9/10	10/10
Reading Span	0/10	0/10	8/10
Forget	0/10	0/10	5/10
Operation Span	0/10	0/10	3/10
Scratch Pad	0/10	1/10	9/10
Ignore	0/10	0/10	8/10

Table 5 – Success criterion: validation loss $< 10^{-4}$

B DWM documentation

The following shows the documentation of the DWM model, it was done using *Sphinx*



class **models.dwm.DWM**(*params*) [source]

Differentiable Working Memory (DWM), is a memory augmented neural network which emulates the human working memory. The DWM shows the same functional characteristics of working memory and robustly learns psychology-inspired tasks and converges faster than comparable state-of-the-art models

forward(*data_tuple*) [source]

Forward function of the DWM model

Parameters:

- **data_tuple** – contains (inputs, targets)
- **data_tuple.inputs** – tensor containing the data sequences of the batch [batch, sequence_length, input_size]
- **data_tuple.targets** – tensor containing the target sequences of the batch [batch, sequence_length, output_size]

Returns: output: logits which represent the prediction of DWM [batch, sequence_length, output_size]

Example:

```
>>> dwm = DWM(params)
>>> inputs = torch.randn(5, 3, 10)
>>> targets = torch.randn(5, 3, 20)
>>> data_tuple = (inputs, targets)
>>> output = dwm(data_tuple)
```

plot(*data_tuple, predictions, sample_number=0*) [source]

Interactive visualization, with a slider enabling to move forth and back along the time axis (iteration in a given episode).

Parameters:

- **data_tuple** – Data tuple containing - input [BATCH_SIZE x SEQUENCE_LENGTH x INPUT_DATA_SIZE] and - target sequences [BATCH_SIZE x SEQUENCE_LENGTH x OUTPUT_DATA_SIZE]
- **predictions** – Prediction sequence [BATCH_SIZE x SEQUENCE_LENGTH x OUTPUT_DATA_SIZE]
- **sample_number** – Number of sample in batch (DEFAULT: 0)

Figure 17 – Documentation of the DWM model

C MI-Prometheus

The following shows the different software layers of *MI-Prometheus*

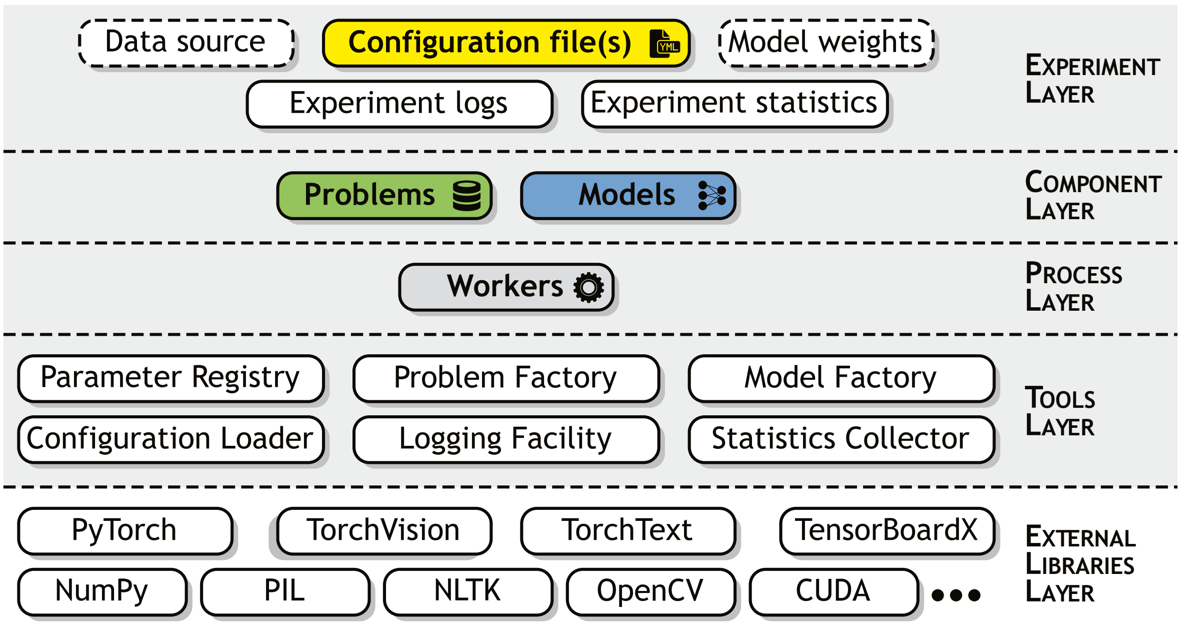


Figure 18 – Software Layers of MI-Prometheus

D Convergence plots

The following plots present convergence of the best models for each of the tasks, expressed in training loss.

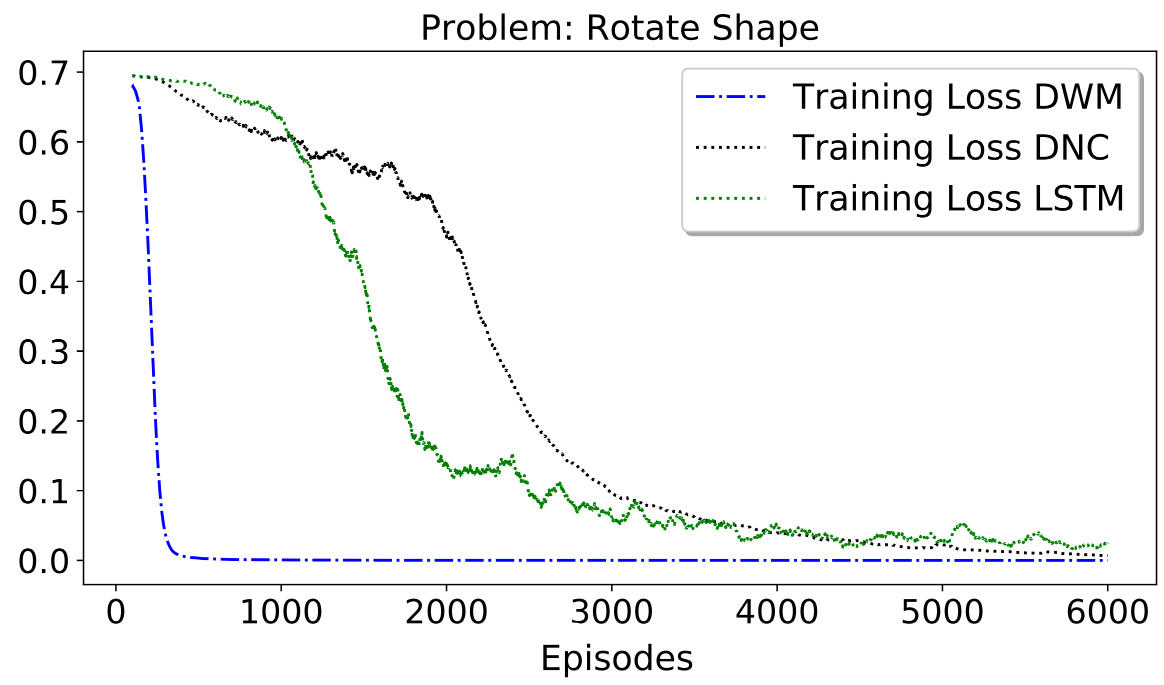


Figure 19 – Training Loss of the best models on the Rotate Shape task

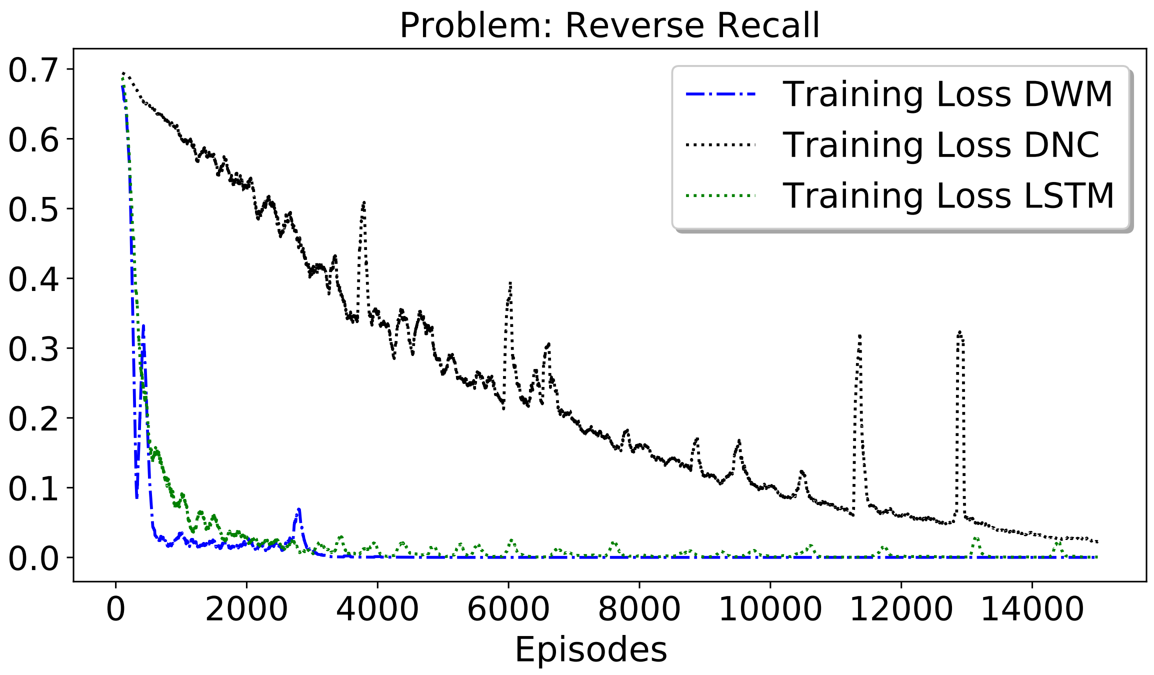


Figure 20 – Training Loss of the best models on the Reverse Recall task

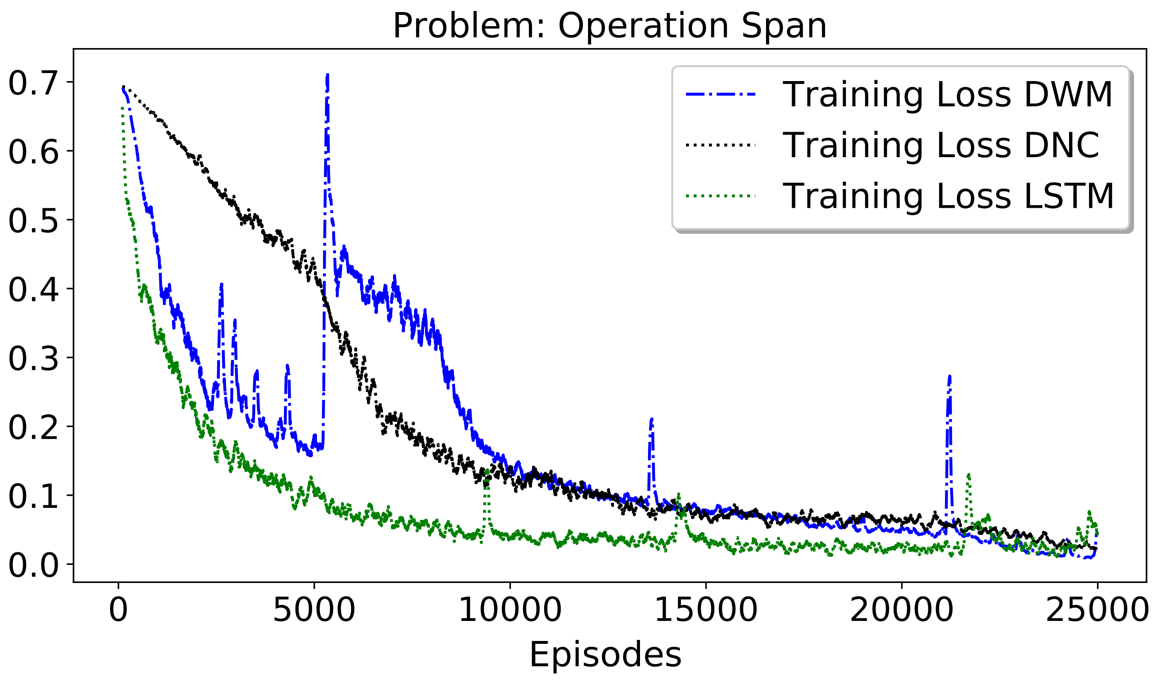


Figure 21 – Training Loss of the best models on Operation Span task

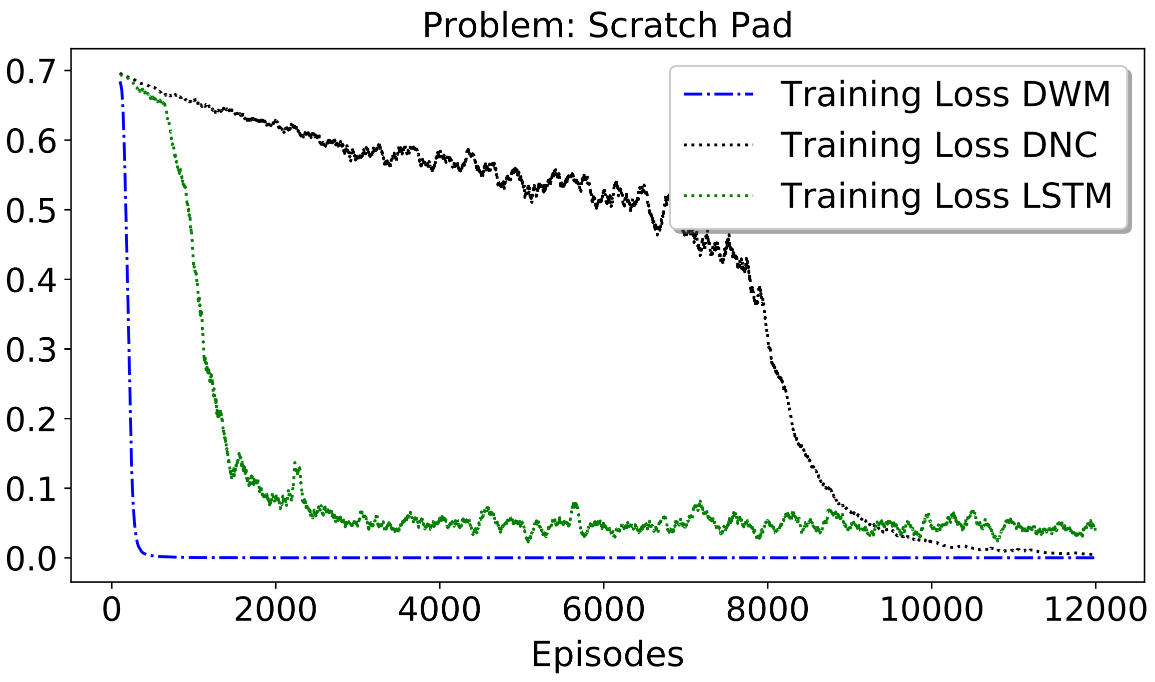


Figure 22 – Training Loss of the best models on Scratch Pad task

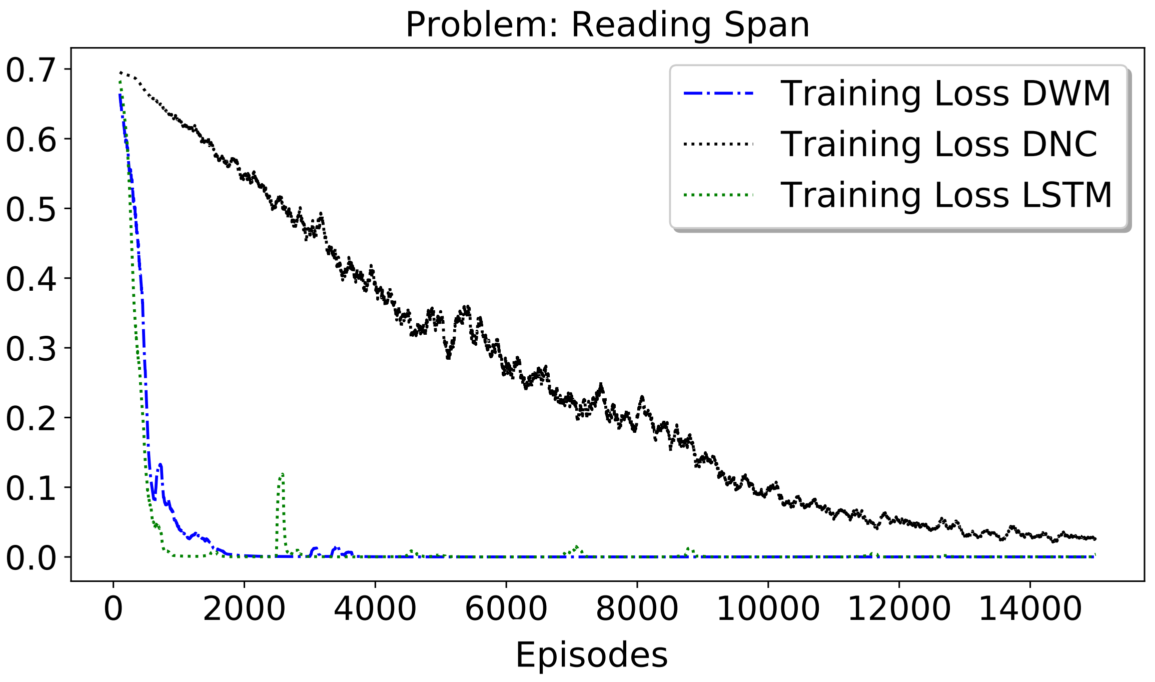


Figure 23 – Training Loss of the best models on Reading Span task

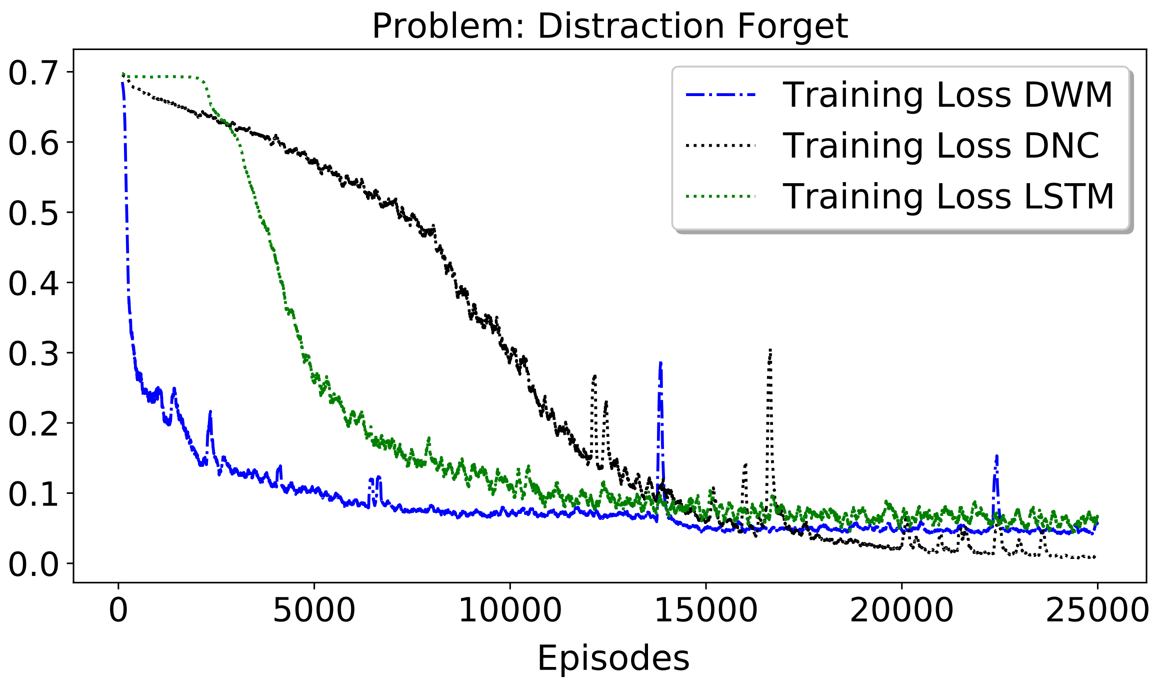


Figure 24 – Training Loss of the best models on Forget task

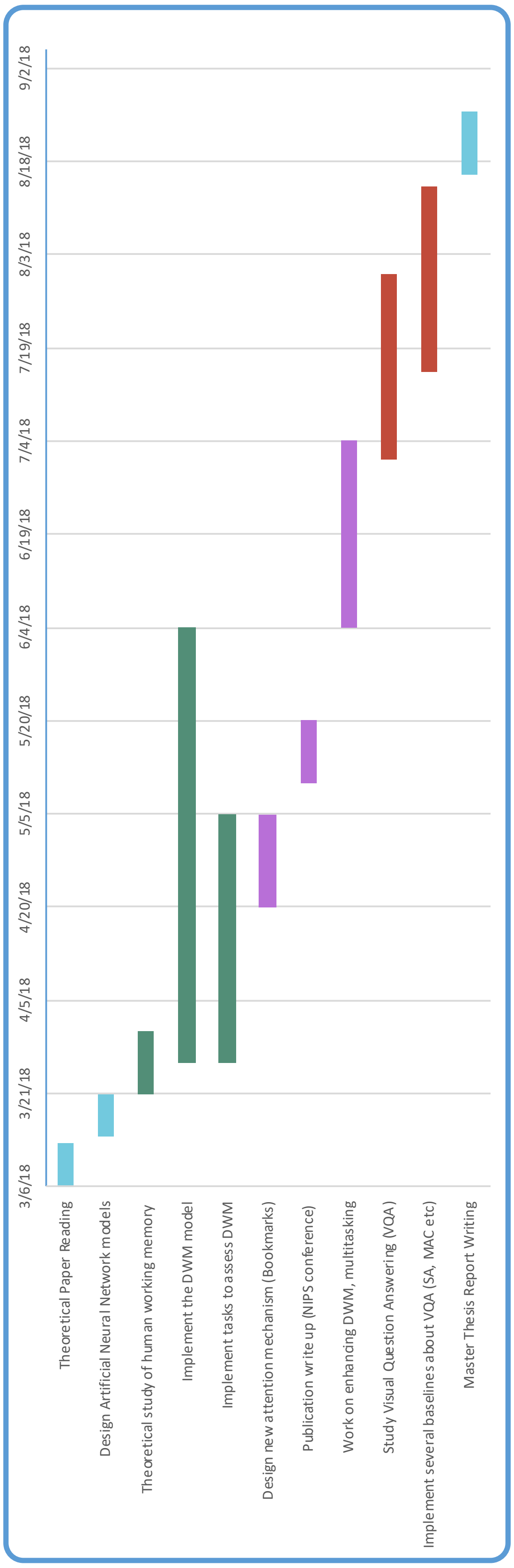


Figure 25 – Gantt Diagram for the master thesis