On the Memory Properties of Recurrent Neural Models

Arthur Jack Russell
Department of Computer Science
City, University of London
London EC1V 0HB, U.K.
Email: arthur.russell@city.ac.uk

Emmanouil Benetos
School of EECS
Queen Mary University of London
London E1 4NS, U.K.
Email: emmanouil.benetos@qmul.ac.uk

Artur d'Avila Garcez
Department of Computer Science
City, University of London
London EC1V 0HB, U.K.
Email: A.Garcez@city.ac.uk

***Abstract*—In this paper, we investigate the memory properties of two popular gated units: long short term memory (LSTM) and gated recurrent units (GRU), which have been used in recurrent neural networks (RNN) to achieve state-of-the-art performance on several machine learning tasks. We propose five basic tasks for isolating and examining specific capabilities relating to the implementation of memory. Results show that (i) both types of gated unit perform less reliably than standard RNN units on tasks testing fixed delay recall, (ii) the reliability of stochastic gradient descent decreases as network complexity increases, and (iii) gated units are found to perform better than standard RNNs on tasks that require values to be stored in memory and updated conditionally upon input to the network. Task performance is found to be surprisingly independent of network depth (number of layers) and connection architecture. Finally, visualisations of the solutions found by these networks are presented and explored, exposing for the first time how logic operations are implemented by individual gated cells and small groups of these cells.**

I. Introduction

Language modeling is the term used to describe the calculation of a probability distribution over the next word or character given the context of previous text [1]. These models include purely statistical n-gram models, as well as those building on some understanding of concept labels, including many variations of recurrent neural networks, which have been applied successfully to tasks such as statistical machine translation (a sequence-to-sequence task that requires a representation of meaning in one language to be successfully translated to a representation of the same meaning in another), sentiment analysis (mapping from raw text in the source language onto a *sentiment space*), and document classification [2].

Several benchmarks have been proposed to test the performance of models on each of the above tasks. However, each benchmark was simultaneously testing an integrated combination of abilities with each task requiring a machine to learn how to extract and represent the syntactic and semantic information contained in language models and apply some operation or reasoning on this information to give a useful solution to the task. In this paper, the distinct components of this challenge are isolated and investigated systematically for the first time, towards a better understanding of the fundamental memory properties of the current state-of-the-art language models based on recurrent neural networks (RNNs).

This paper proposes five basic tasks for isolating and examining specific capabilities relating to the implementation of memory in RNNs, and analyses the ability of such networks to solve these problems in order to gain insight into the fundamental intra-cell and inter-cell mechanisms that RNNs learn to employ, thus contributing to the research on the use of RNNs, and focusing particularly on work in the last two years addressing the ability of RNNs to implement memory and perform symbol grounding as well as reasoning [3], [4], [5]. Specifically, the long short term memory (LSTM) cell [3], originally developed to address the vanishing gradient problem, is evaluated in comparison with the gated recurrent unit (GRU) [4]. LSTMs have been shown capable of learning simple context-free and context-sensitive grammars [6][7]. LSTMs and GRUs have been claimed capable of learning to model grammars because their units can choose to be either linear or non-linear through multiplicative gating mechanisms [8]. Nevertheless, Karpathy et al. [9] highlight that although LSTMs have recently demonstrated exceptional results on several tasks, the source of their abilities remains poorly understood. Related work also includes the study by Boedecker et al. [10] on self-organized optimization of recurrent neural network connectivity, by White et al. [11] on short-term memory properties in orthogonal neural networks, and the study of Jaeger [12] on short term memory in echo state networks.

Our results focus not on the absolute level of performance achieved, but rather on the mechanisms by which individual or small groups of cells encode computational logic that implements memory, and the ability of different network and cell types to learn these mechanisms. We therefore do not present performance comparisons with existing works, but make largely qualitative observations based on our studies. We show that gated networks perform less reliably compared to standard RNN units on a task testing fixed delay recall, indicating that the reliability of stochastic gradient descent in finding solutions decreases as network complexity increases. At the same time, gated units are found to perform categorically better on two tasks that require conditional logic to be implemented. In particular, RNNs with gated units are found to perform better than standard RNNs on tasks that require values to be stored in memory and updated conditionally upon input to the network. Visualisations of the solutions

found by the networks are also proposed, exposing for the first time how logic operations are implemented by individual gated cells and small groups of these cells. These experiments also raise questions that may provide opportunity for further research since: task performance is found to be surprisingly independent of network depth (number of layers) and connection architecture, and significant variance in performance due to different random initialisations is found, implying that stochastic gradient descent struggles to find good minima for these tasks, as discussed in detail in Section IV. Given the small scale of these experiments, another important line of research would be to examine the relevance of these results to larger scale tasks.

The remainder of the paper is organised as follows. Background and related work are outlined in Section II. The proposed series of experimental methods are presented in Section III. Results are presented and discussed in Section IV. Conclusions and directions for future work are summarised in Section V.

II. Background and Related Work

A valid criticism of RNNs is that they mainly have two ways to encode and represent information: firstly in their activations which are recomputed in full and can change radically from step to step, and secondly in their weights which are learned during training but often set after training has completed and represent fixed knowledge about the statistical distribution of patterns. A third way to encode information is using various plasticity mechanisms [13], where synergetic effect of 3 different of these mechanisms leads to internal representations that are able to increase performance of recurrent neural networks. Memory cell approaches such as LSTMs and GRUs are one way to give networks the capacity for a short term memory that responds to recent context to retain information over a longer time-span.

The long short-term memory unit (LSTM) [3] was proposed as a solution to the problem of the vanishing gradients when training standard RNNs. It does this by replacing the typical neural cells that integrate their inputs and apply a non-linear activation function (typically a logistic or tanh sigmoid), with units that add a perfect integrator (referred to as cell state), between the integration of inputs and output activation, effectively allowing them to maintain a persistent memory vector alongside their hidden state vector. At each time step, the LSTM can reset or modify its memory and choose to expose or suppress communication of its contents using explicit gating mechanisms. At each time-step, the new value of the memory vector is computed by multiplying its previous state by a forget gate f, and adding new memory content g, gated by an input gate, i. The candidate hidden state that is passed onto the next layer is a tanh-squashed version of the cell contents. However this candidate output may be suppressed by the output gate o. The gated recurrent unit (GRU) was proposed in [4] as a simpler gated unit than the LSTM. The GRU cell receives an input x, computes a candidate output vector, and outputs an activation h, according to the activity of two gates, a reset gate and an update gate. The update gate interpolates between the old output and a new candidate hidden state to yield the new activation. The reset gate determines whether the previous hidden state is ignored when calculating the candidate. Unlike the LSTM, the activation h is always exposed to higher layers in the network.

In comparison to the LSTM, the update gate can be thought of as replacing the input and forget gates and imposing an additional restriction that input + forget = 1. There is no equivalent of an output gate, meaning that the GRU always communicates its state upwards to the next layer in the network. These certainly represent simplifications. However the candidate vector calculation is more complicated than for the LSTM. In the LSTM, the candidate vector (the cell input, g) is calculated as per a standard RNN, with a tanh function squashing a simple weighted sum of recurrent and feedforward inputs. In the GRU, the candidate vector input also uses a tanh activation function, but the integration of inputs is further dependent on the reset gate, which calculates a multiplicative weighting factor that is applied to the recurrent inputs alone. This gives the candidate vector a non-linear relation to the combined (feedforward + recurrent) input, and makes it harder to assign an interpretation to the activity of the reset gate.

Several other structural modifications to the standard RNN have been proposed recently. The standard recurrent network is extended in [14] by adding a hidden "context" layer that is restricted to change more slowly than the standard fast hidden layer it acts as an input to, effectively forcing them to act as an exponentially decaying bag of words representation of the input history. They demonstrate perplexity performance on the Penn Treebank and Text-8 corpuses that are comparable with LSTM nets with the same number of units (and therefore approximately four times as many parameters). This casts doubt on the desirability of using gated units to perform language modeling, given that they are more complicated, take longer to train and are harder to inspect and understand.

Another structural modification that partitions the hidden layer into separate modules each having a distinct temporal processing rate or clock speed is proposed in [15]. This fixes the activation of the slower partitions between updates, while the quicker partitions continue to process incoming data with access to the context provided by the slower partitions. The authors demonstrate that their approach outperforms RNN and LSTM networks on two audio tasks. It is perhaps unsurprising that this architecture which effectively samples an input signal at different rates performs well on audio patterns which are intrinsically amenable to frequency analysis.

A general architecture for a new class of learning models called memory networks is also introduced in [16]. Similar in inspiration to the methods presented in [8] and the Neural Turing Machines of [17], these combine the pattern recognition, inference and learning capabilities of machine learning approaches such as neural networks with a long term memory component that can be read from and written to. Several state of the art results on reasoning, language modeling, sequence modeling and sentiment analysis tasks [18], [19], [20] have

Input	0	2	2	1	2	0	0	1	0
Target output	-	-	0	2	2	1	2	0	0

Fig. 1. Example FDR pattern: input range 3 i.e. allowed values $[0, 1, 2]$ | delay $(N) = 2$ steps back. The red and blue pairs of outlined cells each illustrate a target value and the corresponding input.

Input	3	0	1	3	2	2	0	1	0
Target output	-	3	3	-	0	1	2	2	1

Fig. 2. Example DDR pattern: input dimension 4, i.e. allowed values $[0, 1, 2, 3]$ | delay = $X + 1$ steps back. The red and blue pairs of cells each illustrate a target value and corresponding input it is derived from. The delay between each pair of cells is conditional upon the shaded cell.

recently been achieved by approaches that allow iterative interaction of these networks with their memory.

III. Methods

In this section, we present a series of experimental methods aimed to isolate and measure the important dimensions of memory for networks with various cells types and architectures, in order to help describe their fundamental memory capacity. Additional aims are to test their ability to perform simple operations on this memory which are fundamental to solving specific tasks; and to explore and characterise the apparent algorithmic and logic solutions learned by the networks, and to highlight the intra-cell and inter-cell mechanics that give rise to these solutions. The dimensions of memory the experiments are designed to explore are: 1) Length of time delay over which memory extends; 2) Number of memory items that are stored; 3) Amount of information contained in each memory item. The simple operations the experiments are designed to test are: 4) Writing to memory conditional upon current input and/or previous state; 5) Access of memory (fixed time delay or a memory "location" conditional upon input).

Fixed Delay Recall (FDR): The FDR task is designed to test the ability of a network to accurately output the input it observed a fixed number N steps previously. This is a test of memory capacity as all intervening inputs between the current input and the input N steps previous need to be stored. The difficulty of the task was modified by changing two variables: the range of possible input values, which determines the theoretical amount of memory required to store one input; and N, the number of steps in the fixed delay, which determines how many inputs need to be stored. An example pattern for the FDR task can be seen in Fig. III, with the target sequence for a network trained on the inputs in the top row.

Examination of networks that successfully complete this task should reveal both how they encode the information contained in the inputs, and how they manage the storage of this information, in order to keep track of the delays corresponding to each stored value. In particular, this task will reveal how efficiently the blocks are able to encode the input. E.g. for an input dimension of 4 (possible input values $[0, 1, 2, 3]$), the network could theoretically learn to encode this in several ways: i) as a one-hot vector requiring four units; ii) as a binary code requiring two units with on/off activations; or iii) using the space of the activation function to encode more than two values in a single unit's output potential.

As for the algorithm, theoretically a large enough single layer RNN should be able to learn to solve this task by passing the input through a series of incremental delays to successive blocks of units in its hidden layer (through a block-wise-identity recurrent connection weight matrix) with the output layer retrieving the values stored in the final delay block. This implies memory capacity will be limited by the size of the hidden layer and that it should not require more than one layer to learn the task. Neither should gating be necessary to successfully solve this task, as the computational logic required to solve it is fixed and not conditional on the input. However experimental results will be analysed to reveal whether multi-layer and/or gated networks better able to learn in practice.

Dynamic delay recall (DDR): The DDR task modifies the FDR task by making the delay conditional upon the current input. Specifically, the target output required to solve the task is the input value from $(X + 1)$ steps back, where X is the current input. The difficulty of the task was modified by adjusting the range of possible input values. This range determines both the theoretical amount of memory required to store one input and the maximum delay, which determines how many inputs need to be stored. An example pattern for the DDR task can be seen in Fig. III. This task adds an additional requirement to the FDR task: all previous inputs up to a maximum delay still need to be stored and "indexed" by delay, but the output must retrieve the correct stored value based conditionally upon the current input, as opposed to retrieving the value corresponding to the same fixed delay each time. This experiment aims to reveal whether networks using gated units learn to perform better by modifying the computation graph through the activities of their gates. It also aims to determine whether the LSTMs ability to suppress output gives it an advantage over the GRU which always passes its activation up to the next layer.

Multiswitch: This task assigns a switch or flag to each possible input value, which flips its state between 0 and 1 each time that input value is seen. The target is the new value of the switch. The difficulty of the task was modified by adjusting the range of possible input values. This range determines the number of switches that need to be maintained in memory. An example pattern for the multiswitch task can be seen in Fig. III. This task is motivated by the observation in [9] that several of the interpretable cells appear to be acting as binary flags that indicate a certain state of the sequence they are processing, e.g. "inside parentheses", "near end of line". This synthetic task isolates the ability to accurately maintain and switch a given number of such flags, and aims to reveal whether networks with gated units learn to solve the problem in different ways and whether this enables them to outperform simple RNNs.

Input	1	1	1	0	2	2	0	1	2
Target output	1	0	1	1	1	0	0	0	1

Fig. 3. Example multiswitch pattern: input range 3 i.e. allowed values $[0, 1, 2]$, corresponding to three switches. Each time a 1 is seen as input, the target output flips.

Input	1	2	1	0	2	2	0	1	0
Target	-	1	1	1	0	2	2	2	2

Fig. 4. Example SVB pattern: input range 3 i.e. allowed values [0,1,2]. In this case, every time a 2 is observed by the input, the target is refreshed with the value that immediately preceded that 2, and maintained until another 2 is observed.

Single variable binding (SVB): The SVB task requires a network to maintain a constant memory that is only updated when a certain trigger event occurs. The trigger "event" in this case is observation of a specific possible input value, here arbitrarily chosen to be the maximum allowed value. The value to be stored is the input that immediately preceded the trigger value. The difficulty of the task was modified by adjusting the range of possible input values. An example pattern for the SVB task can be seen in Fig. III. This task is analogous to an (over-simplified) co-reference resolution task in natural language, where for example the word "she" needs to be resolved to the last female entity mentioned. In order to correctly resolve this co-reference, a memory must be maintained of the last female entity, and refreshed whenever a new female entity is mentioned. It has been argued that such a form of variable binding and addressing mechanism must be implicated in the operation of the brain [21], which further motivates this task which aims to isolate the ability of networks to bind a value to a single specific handle, and to examine whether gating mechanisms allow LSTMs and GRUs to outperform simple RNNs.

Multiple variable binding (MVB): The MVB task extends the SVB task to multiple variables. Each distinct input value now corresponds to a memory item. Two operations are required when each input is processed: firstly, the memory corresponding to that input needs to be recalled - this is the target output; secondly, the memory needs to be refreshed. As for the SVB task, the new memory to be stored is the value of the preceding input. The difficulty of the task was modified by adjusting the range of possible input values. This range determines both the theoretical amount of memory required to store one input and the total number of inputs that need to be stored. An example pattern for the MVB task can be seen in Fig. III. This task tests the ability of a network to simultaneously maintain a memory of multiple multi-bit values. This can be thought of as an extension to the SVB task to multiple variables, or as an extension of the multiswitch task to multi-bit values. Analysis and comparison of results on these three tasks should reveal which of these abilities can be learned by each network, and how they use their network and cell dynamics to implement them.

Timestep	0	1	2	3	4	5	6	7	8
Input	1	1	1	0	2	2	0	1	2
Target	-	-	1	-	-	0	1	1	2
Memory	-	1	1	1	0	2	2	0	1

Fig. 5. Example MVB pattern: input range 3 i.e. allowed values $[0, 1, 2]$. The target is calculated by looking for the previous instance of the current input value, and retrieving the memory associated with it. The blue arrows illustrate how the target value of 1 is arrived at in timestep 6. Firstly the previous instance of the input value (0) is located. The memory associated with this instance (1) is the target value to be recalled. Finally, the memory is updated with the value from the previous timestep (2; dotted line).

IV. Evaluation

A. Experimental Setup

Three main aspects of the models were varied: network architecture, the synthetic training sets used to train and test the networks, and the training regime. With a total of 17 effective hyperparameters across these three categories, the combinatorial explosion of possible experiments meant that it was critical to establish early on which of these factors led to meaningful and interesting variance in the task being examined, which were choices to be optimized for experimental performance (run-time, convergence), and which had little impact on either. Therefore "beam" or "grid" searches across a small number of parameters at a time were conducted to determine which hyperparameters could be fixed and the best values to fix them at.

In all cases, the overall network architectures tested contained a "one-hot" input layer and softmax output layer of dimension determined by the task. Between input and output layers, there were 1 to 3 hidden layers. Five different hidden layer sizes were tested (5, 10, 15, 25, and 50 units per layer). These were composed of one of five different unit types. In addition to the LSTM and GRU cells which were the primary focus of investigation, three different simple (non-gated) units were tested, with hyperbolic tangent ($\tanh(\sum_i \mathbf{W}_{ji}x_i)$), logistic sigmoid ($\sigma(\sum_i \mathbf{W}_{ji}x_i)$) and rectified linear activation function ($\max(0, \sum_i \mathbf{W}_{ji}x_i)$), where $\sigma()$ corresponds to the logistic sigmoid function.

These were chosen to explore whether performance was affected by: 1) the ability of the units to maintain negative activation, which hyperbolic tangent units can do but the other two can not; 2) the ability of the units to maintain unbounded activations, which the rectified linear units can, whilst the others are bounded; and 3) the simpler learning afforded by the rectified linear units, which have recently been shown to yield equal or better performance than hyperbolic tangent networks [22] and seem suited to the logical nature tasks given their ability to learn sparse representations with true zeros.

The output layer was connected to a "bias unit" allowing it to maintain a bias input. Early experiments showed that this increased performance, and also showed that allowing the hidden layers to maintain a bias had little impact on performance. This was surprising, given that in [23] the importance of fixing the bias of the LSTM forget gate to a

high number (effectively forcing it to remember by default) was highlighted.

A further set of permutations concerns the inter-layer connection graph. Four configurations are tested, with a) skip connections to all hidden layers from the inputs, b) skip connections to the outputs from all hidden layers; c) both sets of skip connections; and d) no skip connections (as per standard RNN). This is motivated by a claim [24] that skip connections mitigate vanishing gradients by reducing the number of processing steps between the bottom and top of a network. All layer-layer connections were fully connected where present.

A GRU layer was implemented as an extension to the popular PyBrain library[1] and PyBrain's RPropTrainer[2] was used for training. This applies full batch training using the RProp algorithm [25], adjusting the learning rates for each parameter individually according to whether the error gradient with respect to that parameter has changed sign compared to the previous training epoch (in which case the learning rate is reduced) or has stayed the same (in which case it is increased). It requires the factors determining learning rate increases (etaplus) and decreases (etaminus) to be specified, as well as the initial learning rate (delta0). Factors of 1.2 and 0.5 for etaplus and etaminus were shown to work effectively in [26], and these were adopted here. Values of $\{0.1,0.03,0.01\}$ were used for delta0.

By default, PyBrain initialises weight parameters by sampling from the standard normal distribution. A grid search over a range of values showed that GRU networks converged best when initialised with lower absolute weights - networks initialised with higher weights often exhibited unstable behaviour with performance getting worse, whereas LSTM and RNN networks performed better when their weights were initialised with larger absolute values. An adequate balance for comparison was achieved by initializing weights with a zero-mean Gaussian of variance 0.3, and this was adopted for the main experimental work.

All the tasks were tested using datasets containing synthetically generated sequences. It was expected that larger sets should be less prone to overfitting but would take longer to train on. Another experimental choice was the length of sequences. Given that all tasks had deterministic outcomes, and networks were fully unfolded for error backpropagation, it was unclear how this would affect performance. Initial experiments found that performance on datasets containing 64 sequences of length 20 was similar to performance on larger sets (in both dimension). Therefore this size of dataset was adopted for the main experiments. The training set consisted of 36 sequences, 12 were used as a validation set for early stopping, and 16 were used as a test set. The seed used by random number generators was set for both dataset generation and model training scripts, to ensure that all results were repeatable.

[1]https://github.com/pybrain/pybrain/pull/176

[2]http://pybrain.org/

B. Experimental Results

A summary of best and mean task performance, alongside standard deviation (std) by network type is shown in Table I, as measured by prediction error on the test set. The minimum error demonstrates the best performance that was possible across a range of network configurations and several random weight initialisations for each configuration. This indicates the best practically findable solution within the time and resource constraints of the experiment, and in particular shows whether any network was able to perfectly learn the logic required to solve the problem. The mean and std errors are useful for understanding the typical performance of networks built with each cell type relative to other types. Note that divergence between mean and minimum performance is driven in this case by a conflation of: i) the variety of network sizes and configurations present in the average; and ii) the variance of performance across different random instantiations of each configuration; and should therefore not be interpreted as meaningful.

It can be seen that the best performing cell types vary by task. For the fixed delay recall task, simple recurrent networks containing $tanh$ units provided the best performing networks for all permutations of input dimension and memory span. On this measure, standard recurrent networks (RNNs) comprised of rectified linear units were second best in all cases except one. RNNs with $tanh$ units were also the most consistently performing networks (with lowest mean error). RNNs with logistic sigmoid units were the worst performing on both measures, with the two gated cell types giving intermediate performance. This advantage disappears when the recall task is made conditional on the input (dynamic delay).

For the single variable binding task, the results for harder versions (with greater input dimension) indicate that the clear performance advantage of tanh simple recurrent networks (SRNs) on the FDR task is not apparent for this task where the target value is conditional on the input. The gated units perform slightly better than simple recurrent networks as the input dimension is increased to 4 and 5, however while they close the performance gap on this task, they are not able to learn to perform categorically better than the SRNs. This may indicate that their ability to dynamically modify their computation graph through the activities of their gates either is not relevant for this task, or that a solution utilising this ability was not practically findable by the stochastic gradient descender used here. There is also no evidence to show that the LSTM's ability to suppress output gives it an advantage over the GRU.

Results for the multiswitch task show that gated cell networks are able to solve the 2 switch problem perfectly, whereas SRNs are not. Note that the average results here cover all permutations of skip connectivity, with several different random initialisations for each permutation. No specific skip architecture was found to perform consistently better than the others. The discrepancy between minimum and mean error therefore indicates that the ability of these networks to find

All configurations			Minimum test set error					Mean test set error					Std test set error				
Task	**Steps back**	**Input dimension**	**gru**	**lstm**	**sig**	**relu**	**tanh**	**gru**	**lstm**	**sig**	**relu**	**tanh**	**gru**	**lstm**	**sig**	**relu**	**tanh**
Fixed Delay Recall	3	3	0.0	0.3	3.8	0.0	0.0	15.0	15.1	41.8	10.1	0.2	15.6	14.2	16.8	15.6	0.3
		4	5.3	16.3	23.8	6.3	0.0	26.8	27.5	53.8	32.1	10.2	11.1	8.6	12.8	22.2	15.5
		5	20.0	14.7	32.2	3.8	0.0	35.5	39.6	54.3	26.9	5.4	10.2	10.7	13.2	16.7	7.6
	4	3	10.6	13.8	48.8	3.4	0.0	29.0	34.3	52.8	30.0	11.6	10.0	9.8	1.6	16.1	16.0
		4	28.8	32.2	55.6	11.3	0.3	40.0	42.3	58.4	36.5	18.1	7.9	6.7	1.8	13.9	20.6
		5	39.1	38.1	57.8	27.2	1.3	46.7	48.5	63.2	44.8	26.9	5.4	7.3	2.5	9.7	24.2
Dynamic Delay Recall	dynamic	3	5.0	3.8	17.8	5.3	2.5	11.6	14.2	29.0	23.1	17.9	4.5	5.7	9.1	13.3	14.4
		4	33.4	36.6	41.6	40.3	40.6	39.8	40.9	53.0	49.9	48.0	3.7	2.7	5.5	8.4	4.2
		5	42.5	43.4	50.3	50.3	51.9	48.1	51.2	61.7	58.1	56.7	3.4	5.3	6.9	4.9	5.0
Multiswitch	dynamic	2	0.0	0.0	10.0	14.4	15.6	21.3	20.4	38.6	37.3	44.3	14.5	15.8	9.0	11.4	6.6
		3	14.1	2.5	34.1	30.3	39.1	42.0	42.4	46.7	43.8	45.7	8.1	8.8	3.3	4.5	2.7
		4	32.2	32.5	41.3	32.2	40.9	44.6	44.6	44.7	44.8	45.7	4.0	3.7	1.6	1.9	2.8
Single Variable Binding	dynamic	3	0.0	0.0	0.0	0.0	0.0	0.4	1.7	8.2	16.3	6.3	0.8	1.7	9.0	12.0	6.1
		4	0.0	2.8	5.3	15.9	11.3	6.3	20.9	38.8	37.5	40.5	5.9	12.8	12.5	11.2	10.6
		5	0.0	10.3	27.2	30.9	27.5	15.8	30.5	46.1	50.5	48.7	10.7	12.0	9.0	8.9	8.4
		6	0.3	24.4	48.4	47.5	38.4	33.6	51.0	60.9	61.8	59.5	14.5	11.6	5.8	7.0	5.9
		7	28.1	47.8	52.5	54.1	54.7	48.0	62.2	63.2	66.1	63.6	11.4	4.6	3.6	5.1	4.2
Multiple Variable Binding	dynamic	2	0.0	0.0	10.0	0.9	0.6	5.0	6.9	23.4	12.0	3.2	6.7	8.0	9.2	7.0	2.2
		3	24.4	29.7	39.7	37.8	30.9	38.9	40.5	53.5	48.1	44.2	8.7	7.9	9.2	7.0	8.3
		4	44.7	45.9	53.4	55.9	49.1	52.2	53.4	62.4	61.5	57.9	3.2	3.6	3.0	2.7	4.1

TABLE I
SUMMARY OF PERFORMANCE FOR EACH NETWORK TYPE ON THE FIVE TASKS TESTED.

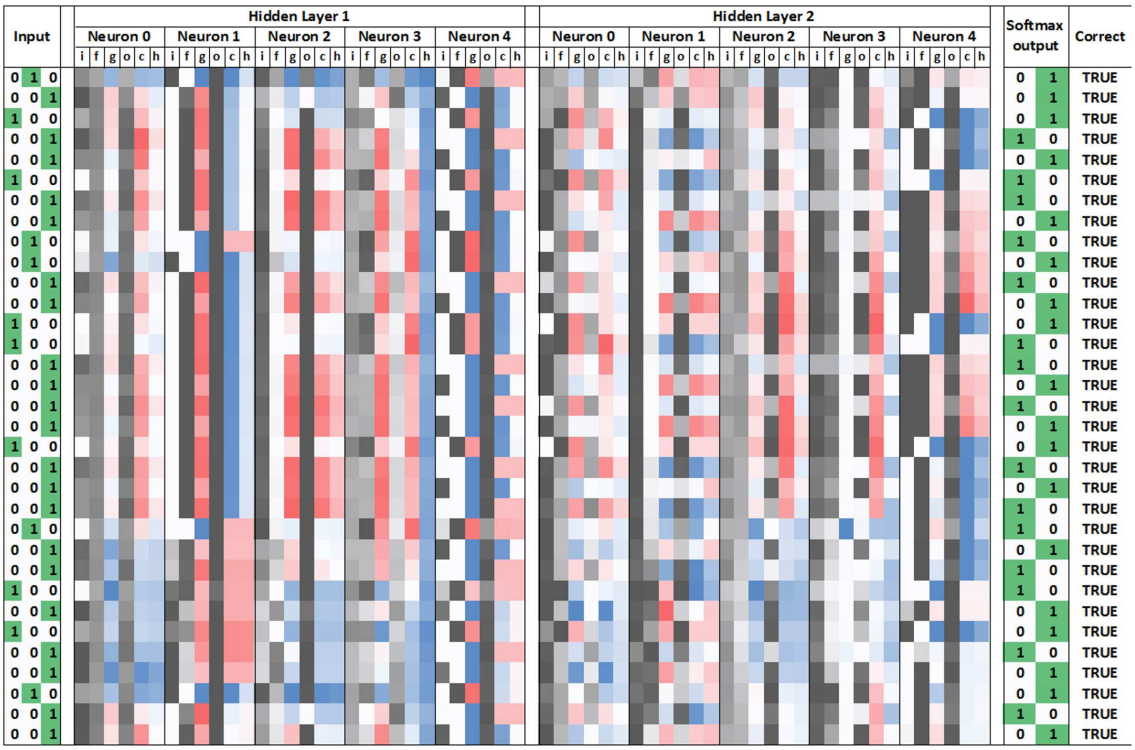

Fig. 6. Visualisation of the 2-layer LSTM net on the multiswitch task, showing the activity of the LSTM gates (shaded columns labeled i, f and o - darker shading represent activities closer to 1, lighter shading activity closer to 0), the candidate and resultant cell states and hidden activations for each hidden unit (g, c and h respectively; red and blue represent negative and positive values of activation relative to the column average respectively), and the activation of the softmax output layer (argmax = green), which correctly predicts the target in every case in this example.

a solution is sensitive to the random initialisation of weight parameters. Further investigation into the topography of the loss function and associated gradient descent mechanics is required to understand this sensitivity. No network was able to perfectly solve the 3 switch problem, although an LSTM network with 2 layers of five neurons and full skip connections managed to achieve 2.5% error. However the networks with gated units performed consistently better than SRNs on this task. Performance on the 4 switch task was worse again, with no network achieving an error rate below 30%. The differential between gated and non-gated cells decreased, driven by a decrease in performance of the gated-cell networks (the SRNs performed similarly on both 3 and 4 switch tasks with error rates between 40-50%).

For the SVB task, gated cell networks performed better than SRNs. There was no consistent difference in performance between the different types of non-gated unit in the SRNs. As for the gated units, GRUs outperformed LSTM networks, and were the only networks able to perfectly solve the tasks with input dimension 4 and 5. Finally, for the multiple variable binding task, perfect solutions to the 2 variable task were found by GRU and LSTM networks. However the tanh RNN came very close to a perfect solution and had a lower mean error that gated cell networks for smaller hidden layer sizes. Given the rapid decline in performance of all networks on the multiswitch and SVB tasks, it is perhaps unsurprising that no network was able to solve the 3 or 4 variable task.

As a way to visualise network performance, Figure IV-B details the weights learned by the network that achieved an error rate of 2.5% on the multiswitch task (3 switches). This network had 2 layers of five LSTM cells and skip connections between both the input and second hidden layer, and between the first hidden layer and the output layer. The figure illustrates how this network processes an example input sequence, showing the activity of the LSTM gates, the resultant cell states and hidden activations for each hidden unit, and finally the activation of the softmax output layer. Inspection of the cell state of neuron 1 in hidden layer 1 reveals that this cell is tracking the status of the second switch, flipping from higher (blue) to lower (red in this case representing zero) activation and vice-versa whenever the input [0,1,0] is observed. For the LSTM network that solved the multiswitch task, the input and forget gates of one cell for each switch worked together to compute an effective XOR over their previous state and the current input, allowing the correct state of the switches to be maintained. The output gate was not found to be important to solving the task. For the GRU network that solved the single variable binding task, the interaction between update gate and candidate inputs was found to be the key mechanism by which information was either maintained or updated according to the input. The role of the reset gate was less clear. These results provide experimental support for the hypothesis that gated units employ fundamentally different mechanisms to simple RNNs in order to implement logic that is conditional based on the input, and indicate which components are crucial to their ability to do so, shedding light into the mechanisms at play behind the success of these network models at performing language modelling. As such, we do not present performance comparisons with existing works here, rather we hope that these results suggest a method to interrogate the micro-level basis of performance gains made by recent works, and hope that future research can bring together our understanding of the micro-level and macro-level behaviour of recurrent neural models.

V. Conclusions and Future Work

We have proposed 5 basic tasks for isolating and examining specific capabilities relating to the implementation of memory for two popular gated units used in RNNs. Results show that gated networks perform less reliably compared to standard RNN units on a task testing fixed delay recall. Gated units are found to perform better than standard RNNs on tasks that require values to be stored in memory and updated conditionally upon input to the network. Task performance is found to be surprisingly independent of network depth and connection architecture, with significant variance in performance. Finally, visualisations of the solutions found by these networks were proposed, exposing for the first time how logic operations are implemented by individual gated cells and small groups of these cells.

As future work, we propose the use of the above five basic tasks to investigate fundamental performance properties of other alternative recurrent network models, and would regard as particularly interesting the study of the properties of memory networks and neural Turing machines, as a result of their recent success at language modelling and deep reinforcement learning. We also suggest to study the applicability of the proposed measures in practice, and their potential relevance to large-scale problems, by carrying out a systematic comparative evaluation of results on benchmark datasets.

Acknowledgment

EB is supported by a UK Royal Academy of Engineering Research Fellowship (RF/128).

References

[1] A. Bordes, N. Usunier, R. Collobert, and J. Weston, "Towards understanding situated natural language," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 65–72.

[2] D. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *ArXiv e-prints*, 2014, arXiv:1406.1078 [cs.CL]. [Online]. Available: http://arxiv.org/abs/1406.1078

[5] T. Besold, K.-U. Kuehnberger, A. Garcez, A. Saffiotti, M. Fischer, and A. Bundy, *Anchoring Knowledge in Interaction: Towards a harmonic subsymbolic/symbolic framework and architecture of computational cognition*, ser. Lecture Notes in Computer Science – Artificial General Intelligence. Springer International Publishing, 2015, pp. 35–45.

[6] F. Gers and J. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, Nov 2001.

[7] W. Zaremba and I. Sutskever, "Learning to execute," *ArXiv e-prints*, 2014, arXiv:1410.4615 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1410.4615

[8] A. Joulin and T. Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," *ArXiv e-prints*, 2015, arXiv:1503.01007 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1503.01007

[9] A. Karpathy, J. Johnson, and F.-F. Li, "Visualizing and understanding recurrent networks," *ArXiv e-prints*, Nov. 2015, arXiv:1506.02078 [cs.LG].

[10] J. Boedecker, O. Obst, N. M. Mayer, and M. Asada, "Initialization and self-organized optimization of recurrent neural network connectivity," *HFSP Journal*, vol. 3, no. 5, 2009.

[11] O. L. White, D. D. Lee, and H. Sompolinsky, "Short-term memory in orthogonal neural networks," *Physical Review Letters*, vol. 92, no. 14, 2004.

[12] H. Jaeger, "Short term memory in echo state networks," German National Research Center for Information Technology, Tech. Rep., 2001.

[13] A. Lazar, G. Pipa, and J. Triesch, "SORN: A self-organizing recurrent neural network," *Frontiers in Computational Neuroscience*, vol. 3, no. 23, 2009.

[14] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, "Learning longer memory in recurrent neural networks," *ArXiv e-prints*, 2015, arXiv:1412.7753 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1412.7753

[15] J. Koutnk, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork RNN," *ArXiv e-prints*, 2014, arXiv:1402.3511 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1402.3511

[16] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merrinboer, A. Joulin, and T. Mikolov, "Towards AI-complete question answering: a set of prerequisite toy tasks," *ArXiv e-prints*, 2015, arXiv:1502.05698 [cs.AI]. [Online]. Available: http://arxiv.org/abs/1502.05698

[17] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *ArXiv e-prints*, 2015, arXiv:1410.5401 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1410.5401

[18] B. Peng, Z. Lu, H. Li, and K.-F. Wong, "Towards neural network-based reasoning," *ArXiv e-prints*, 2015, arXiv:1508.05508 [cs.AI]. [Online]. Available: http://arxiv.org/abs/1508.05508

[19] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *ArXiv e-prints*, 2015, arXiv:1506.07285 [cs.CL]. [Online]. Available: http://arxiv.org/abs/1506.07285

[20] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *ArXiv e-prints*, 2015, arXiv:1503.08895 [cs.NE]. [Online]. Available: http://arxiv.org/abs/1503.08895

[21] G. F. Marcus, *The algebraic mind: Integrating connectionism and cognitive science*. MIT Press, 2003.

[22] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, vol. 15, 2011, pp. 315–323.

[23] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 2342–2350.

[24] A. Graves, *Supervised sequence labelling with recurrent neural networks*. Springer-Verlag, 2012.

[25] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *IEEE International Conference on Neural Networks*, 1993.

[26] C. Igel and M. Hüsken, "Empirical evaluation of the improved Rprop learning algorithms," *Neurocomputing*, vol. 50, pp. 105–123, 2003.