Principles and Applications of Differential Neural Computers

Student Name: Alastair Breeze

Supervisor Name: Dr Stephen McGough, Dr Noura Al Moubayed

Submitted as part of the degree of Computer Science MEng to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

02.05.2017

Abstract —

Context / Background — Memory is a core mechanism that facilitates intelligence in a highly contextual universe, allowing the entity to draw on past experiences to influence the subsequent. Sequence problems deal with ordered data as seen everywhere in the natural world. In artificial intelligence research there have been advances in machine learning incorporating memory into trainable models that have been applied to real world recurrent problems ranging from natural language processing to healthcare.

Aims — The project will explore memory rich machine learning architecture to review both historical approaches against recent. Current state of the art models have become the definitive for recurrent problems until recently where an architecture has been proposed to bridge the gap between classic Turing Machine architectures with neural computing models to train programmable models. The project will analyse, evaluate and compare these upcoming memory heavy architectures and discuss their capability in solving problems from data structure tasks to real world strategic, computer vision and natural language processing problems.

Method — Our system poses state of the art recurrent architectures against the new Differential Neural Computer, a Turing machine inspired architecture that performs actions on memory using read heads and is the basis for most modern computation. We optimised and improved the architectures, applying modern recurrent neural network regularisations and optimisations maximising potential of our architectures. We applied revisions to the models through deeper exploration of the mechanisms through visualisations.

Results — We evaluated the power of the models gaining insight into the difficulties of training. Our methods prove the scalability and how powerful the memory mechanisms compare. The concluded architecture are tested and insights applied into realised problems. The use of various common visualisation techniques and more concept ideas will allow us to validate our findings.

Conclusions — The project exhibited the power of the state of the art memory architectures, and demonstrated cases highlighting the benefits of the differential neural computer architecture. Through the understanding of the bottlenecks of the architecture, we identified shortcomings of the models and applied revisions to improve the architecture.

Keywords — Artificial Intelligence, Computer Vision, Machine Learning, Natural Language Processing, Recurrent Neural Network, Regularisation, Turing Machine

I INTRODUCTION

Artificial intelligence (AI) research constantly strives to create truly intelligent systems, something classically modelled on human intelligence. This does not come without difficulty, where the universe itself can be visualised as a giant time sequential masterpiece of patterns. Examples range from video, audio, speech and so understanding ordered patterns is key. Turing (1950) elaborated on the difficulties of intelligent machines and led to discussing 'The Imitation Game', a test that uses recurrent data. We won't directly attempt the challenge, but natural language processing (NLP) and strategy are core to the problem.

In recent years' advances have been made in sequential problems thanks to advancements in machine learning (ML) that led to algorithms that recognise patterns in high dimensional sequence data (Karpathy, 2015). Feed-forward and convolutional neural network (CNN) research in competitions like the ImageNet[1] drove research and were extended into recurrent neural networks (RNNs). Regularisations and visualisations from such fields will be drawn to benefit our models.

RNN architectures can recur previous state/memory through the model as we chronologically process the sequence. More recently research has taken a radical approach to neural network design in the form of Differential Neural Computer (DNC) that inflates memory at the disposal of the models (Graves et al., 2016). It crosses a Turing Machine (TM) (Turing, 1937), the basis of modern programmable computation, using neural networks to train a program. The models stem close to classic computer architectures, something Turing (1950) later discussed as 'Digital Computers' made up of core parts: Store, Execution Unit and Control as portrayed recurrently in Figure 1. In our research we will elaborate on this idea to acumen future research and push model capabilities.

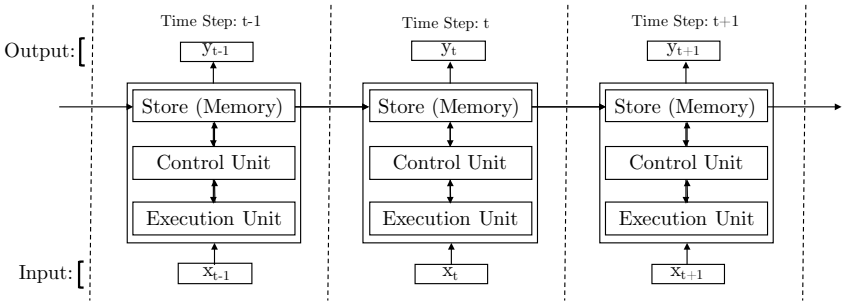


Figure 1: Turing's 'digital computer' model unravelled over time-steps for recurrent problems.

A Potential Applications

NLP interfaces human-computer communications and, due to the highly contextual nature of communication, the problem area lends itself to memory models. Examples include automatic summarisation, sentiment analysis, translation speech recognition and text-to-speech problems.

[1]ImageNet - `www.image-net.org/` (accessed February 2017)

Healthcare also draws benefits from recurrent problems, feeding clinical measurements to predict early signs of disease like Lipton et al. (2015) attempting to classify 128 diagnoses.

B Project Aims and Deliverables

The project aims to explore state of art against the DNC architecture on a variety of problems. We answer: 'How do concept recurrent architectures compare to state of the art in memory, speed, complexity and their potential?' The aims are broken into components:

1. **Validate** state of art and concept recurrent architectures against research problems.
2. **Optimise** the architectures to push capabilities and limits of the models.
3. **Visualise** the methods to understand and verify their inner workings.
4. **Explore** adjustments to improve the models.

We realise these aims through the deliverables listed in Table 1.

Deliverable	**Class**
Implementation of a classic RNN models: SRN/RNN, LSTM, GRU.	Basic
Implementation of a modular testing system.	Basic
Implementation of a problem interface with basic problems.	Basic
Implementation of the DNC neural architecture.	Intermediate
Validation of DNC against original paper claims.	Intermediate
Optimisation of the system for SIMD architectures.	Intermediate
Investigate regularisations and optimisations over the architectures.	Intermediate
Visualisation system for understanding the inner workings of the models.	Intermediate
Implementation of more complex problems to challenge model characteristics.	Intermediate
Implementation of real world recurrent problems to the models.	Advanced
Implementation of distributed variant of the algorithms.	Advanced
Improvement to mechanisms of the DNC architecture.	Advanced

Table 1: Project Deliverables.

II RELATED WORK

In this section we review the history behind recurrent problems and recurrent architectures, exploring optimisations and visualisations that could enhance our models.

A Architectures

RNNs are ML architectures that tackle sequence data problems. In this sub-section we survey key milestones in recurrent architectural research history.

A.1 Simple Recurrent Network (SRN)

Back-propagation by Rumelhart et al. (1986) is the base of learning algorithms, and was extended for sequence data by Werbos (1990) to create the back-propagation through time (BPTT) algorithm. Elman (1990) used this to coin the simple recurrent network (SRN). Elman applied the architecture to the next letter prediction problem and next word prediction in sentences, an

elementary NLP problem demonstrating early uses RNNs in the field.

Bengio et al. (1994) explored the difficulty of training due to the added complexity that grew with sequence length and issues like gradients blowing up or vanishing on long term dependencies. Elman used truncated BPTT of Williams and Peng (1990) to counter the limitation, however this is unable to yield long term dependencies further than the truncation weakening the model.

A.2 Long Short-Term Memory (LSTM) & Gated Recurrent Unit (GRU)

Issues in the SRN led to the work of Hochreiter and Schmidhuber (1997) developing the long short-term memory (LSTM) architecture. The technique, often described as a SRN regularisation, was designed to maintain long term dependencies through gating mechanisms that avoid vanishing or exploding gradients and assist efficient gradient flow through the models.

LSTMs are still prominent today including the PixelRNN by van den Oord et al. (2016), an implementation that demonstrates recurrent depths of 12 LSTM units to yield high level data abstractions on hard problems. Other research includes naive regularisation heuristic Pascanu et al. (2012) to the gradient explosion problem that clips gradients at a given threshold.

The Gated Recurrent Unit (GRU) proposed by Bahdanau et al. (2014) is an LSTM simplification that attempts to accelerate learning. Merits of the GRU over the LSTM have been subject to deliberation, with Chung et al. (2014) concluding deficient evidence and that further problems should be explored. In our research we investigated this open problem.

A.3 Neural Turing Machine (NTM) & Differentiable Neural Computer (DNC)

Much like that of a human the LSTM uses neurons for storage, but what if we were able to mount accessible memory like a modern computer? Recent research probed such radical ideas leading to the Neural Turing Machines (NTM), a simulated Turing Machine (TM) controlled by a neural network (Graves et al., 2014). The model abstracts similarities with the TM Turing (1937) using read/write heads to access and modify a memory matrix.

DeepMind[2] improved their memory mechanisms forming the Differential Neural Computer (DNC) (Graves et al., 2016). The architecture constrains the mechanics of the NTM into a more trainable system through introduction of usage vectors and link matrices to improve memory lookup. There has been mild success replicating the DNC, once in October 2016 where a small system was made[3]. Shortly after, Samir implemented the model[4] in a more common library that we will build upon, optimising and analysing the DNC mechanisms on a variety of problem areas to cross validate against a more standard LSTM/GRU model.

B Problem Areas

Recurrent problems span a wide variety of fields, something we explore in following sub-sections.

[2]DeepMind - `deepmind.com/` (accessed 04/2017)

[3]yos1up/DNC: Differentiable Neural Computers `github.com/yos1up/DNC` (accessed 03/2017)

[4]Mostafa-Samir/DNC-tensorflow: A TensorFlow implementation of DeepMind's Differential Neural Computers (DNC) `github.com/Mostafa-Samir/DNC-tensorflow` (accessed 03/2017)

B.1 Data Structure Manipulation

Data structures are the foundation to problem solving and provide good baseline benchmarks for the inner workings of models. It will also prove useful when evaluating the algorithmic power behind the DNC. Examples of this include Henaff et al. (2016) performing similar analysis.

B.2 Strategic Problems

Strategic problems involve designing a policy to solve a given task. Headlines were reached by the DNC being applied to navigate the London Underground[5], an application of graph theory, an interesting achievement in reasons the press did not realise. The press focused on the real nature to the problem, yet overlooked the ability to form strategy on problems like block puzzles.

A Rubik's cube problem sustains roughly $43,252,003,274,489,856,000$ permutations, and the difficulty has been heavily researched by projects like Rokicki et al. (2010). The work of Cube 20 made it possible confirm 'God's Number' bounding maximum diameter of the $3 \times 3 \times 3$ Rubik's cube state space. Irpan (2016) explored boosted LSTM but concluded the extensive difficulty of the Rubik's cube, something we too wish to investigate on various cube sizes.

B.3 Computer Vision

Classifying handwritten digits is standardised by LeCun and Cortes (2010) as MNIST. The computer vision problem evolved into a baseline ML benchmark as one of the original neural network applications. Although not classically a recurrent problem, it is possible to formulate recurrently.

B.4 Natural Language Processing (NLP)

A powerful application of RNNs is in NLP thanks to its sequence modelling abilities. Early recurrent models like the SRN Elman (1990) were applied to some elementary word sequencing problems. The paper also discussed vector representation of words later evaluated by Řehůřek and Sojka (2010), more recently Word2Vec of Mikolov et al. (2013) has became the state of the art.

Question Answering is the ability to understand and reason with text, a key problem for human computer communication. Weston et al. (2015) standardised a set of problems known as bAbi. The DNC was applied to the problem by Graves et al. (2016) and later investigated by Samir[4] concluding trainability but limited improvement.

Sentiment Analysis focuses on classifying opinions towards text and has became a benchmark in RNN as the IMDb movies database (Maas et al., 2011). The problem classifies movie reviews from IMDb[6] as positive or negative. Another example is classifying different argument sentiment types Oraby et al. (2016) including sarcasm, a popular application for filtering product reviews and has further led to datasets on `Amazon.com` reviews (Filatova, 2012).

[5] DeepMind's AI learned to ride the London Underground using human-like reasoning and memory - `www.wired.co.uk/article/deepmind-ai-tube-london-underground` (accessed 2017)

[6] 'IMDb - Movies, TV and Celebrities - IMDb' - IMDb.com Inc. - `www.imdb.com` (accessed 03/2017)

C Optimisations

Efficiently trainable models are vital for effective systems given the high complexity and parametrisation of the models. In this sub-section we explore RNN regularisations and optimisations.

C.1 Regularisation

Over-fitting is when we fit the training set too closely over the general case. The regularisation work of Zaremba et al. (2014) can help avoid over-fitting, like dropout that removes dependence on certain neurons through random neurons used during training popular with feed-forward networks and was mutated for RNNs. L1/L2 regularisation ensures network weights remain small, discussed by Pascanu et al. (2012) for recurrent models.

C.2 Scalability

Hardware parallelisation was a key breakthrough in ML that came power contained by general purpose graphics processing units (GPGPU). Thanks to the high vectorisation and parallelisation potential of ML functions, we scale training over these many-core GPGPUs as single input multiple data (SIMD) programming. Future ML specific hardware like Tensor Processing Unit (TPU)[7] aim to take this further. Unfortunately the youth of TPUs mean we won't be able to test the claims, but we will investigate GPGPU advantages.

Distributed learning was a core part of the Tensorflow library (Abadi et al., 2015, 2016). Thanks to the serialisation it is possible to distribute over different hardware options demonstrated by projects like AlphaGo that utilised 1920 CPUs and 280 GPUs (Silver et al., 2016). We lack this scale of resources, however will investigate speed up on a lesser scale.

D Visualisation

Understanding hidden logic of a neural network can further enhance and optimise models (Karpathy et al., 2015). Advancements in tools like Tensorflow make it possible to visualise the models live:

- **Tensor Visualisations —** Verifying neuron distributions provides insight into model learning and helps deciding the need for regularising through methods like L1/L2.
- **Word Embeddings —** High dimensional word embeddings can be visualised using a non-linear dimensionality reduction techniques like t-SNE van der Maaten and Hinton (2008) and are be powerful when applied to word-embeddings like by Mikolov et al. (2013).
- **Model Visualisations —** As we create self-programming TMs, visualising the workings of the mechanisms on the memory could prove useful.

III SOLUTION

Progressively defining our models, optimisations, regularisations and modifications with hypotheses are discussed in this section followed by the targeted problems areas.

[7] Google Cloud Platform Blog: Google supercharges machine learning tasks with TPU custom chip - Google Cloud Platform Blog `cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html` (accessed 03/2017)

A Model

In this sub-section we evolve our model from a SRN all the way to our most advanced DNC.

A.1 Simple Recurrent Network (SRN)

We begin with the SRN model of Elman (1990) by defining hidden variable h at time-step t, with weight matrices W_*, bias vectors b_*, input x_t, activation functions σ_* to form predicted output $\hat{y}_t$:

$$h_t = \sigma_h(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

$$\hat{y}_t = \sigma_y(W_{hy}h_t + b_y) \tag{2}$$

Unfolding our network over time t, we form a trainable neural network (Rumelhart et al., 1986).

A.2 Long Short Term Memory (LSTM) & Gated Recurrent Unit (GRU)

We regularise our SRN into the **LSTM** described by Chung et al. (2014). The unit maintains both an input and forget gate i, f at each time-step t:

$$i_t = \textsc{Sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3}$$

$$f_t = \textsc{Sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{4}$$

Updated cell contents c_t is updated partially forgetting previous and adding the new content:

$$c_t = (f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)) \tag{5}$$

Our output gate o modulates the memory exposure finally combined as output h:

$$o_t = \textsc{Sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{6}$$

$$h_t = (o_t \tanh(c_t)) \tag{7}$$

The **GRU** similarly defined by Chung, simplifies this using update gate z and reset gate r:

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{8}$$

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{9}$$

That are combined to create candidate activation $\tilde{h}$ where $\odot$ is the Hadamard product:

$$\tilde{h}_t = \tanh(W_{x\tilde{h}}x_t + W_{rh\tilde{h}}(r_t \odot h_{t-1}) + b_{\tilde{h}}) \tag{10}$$

A linear interpolation between previous activation h_{t-1} and candidate $\tilde{h}$ forms our cell output:

$$h_t = ((1 - z_t)h_{t-1} + z_t\tilde{h}_t) \tag{11}$$

The simplicity of the GRU is obvious, hence has potential to train faster. Both models are able to moderate their update unlike SRNs. Another core GRU difference is the inability to control both the new and current memory separately and instead exposes the full memory.

Layering recurrent units form higher levels of data abstractions to improve pattern recognition performance where the output of one layer becomes the input to the next. Depth was exhibited by van den Oord et al. (2016) reaching depths of 12, a potential merit we wish to explore.

A.3 Neural Turing Machine (NTM) & Differential Neural Computer (DNC)

The NTM was built around basic functions of a TM, controlled by a neural network. The DNC improves the NTM by re-engineering the restrictive head mechanisms. We focus on the DNC, forming definitions from Graves et al. (2016) breaking each time-step t into four distinct stages.

Stage 1: Controller Parsing — Firstly we define the controller, a single neural network that takes as input the current input vector x_t concatenated with previous time-step read vectors from each read head i: r^i_{t-1} later defined. The controller can be any neural network including feed-forward and RNNs. Using the scheme in Table 2 we parse controller output vector c_t. We define constants R - the amount of read heads; N memory rows of width W.

Field Name	Notation	Description	Domain	Activation
Lookup mask	l_t	Mask for key-value lookup	$\mathbb{R}^{\mathrm{W}}$	SIGMOID
Write key	k^w_t	Lookup key for writing	$\mathbb{R}^{\mathrm{W}}$	SIGMOID
Write strength	β^w	Write key strength	$\mathbb{R}$	SOFTPLUS
Free gates	g^f_t	Free or retain read heads locations	$\mathbb{R}^{\mathrm{R}}$	SIGMOID
Allocation gate	g^a_t	Allocate or replace memory	$\mathbb{R}$	SIGMOID
Write vector	v_t	Vector to be written	$\mathbb{R}^{\mathrm{W}}$	SIGMOID
Erase vector	e_t	Vector to be erased	$\mathbb{R}^{\mathrm{W}}$	SIGMOID
Write gate	g^w_t	Perform write	$\mathbb{R}$	SIGMOID
Read keys	$k^{r,i}_t$	Lookup keys for reading	$\mathbb{R}^{\mathrm{R}} \times \mathbb{R}^{\mathrm{W}}$	SIGMOID
Read strengths	$\beta^{r,i}_t$	Read key strengths	$\mathbb{R}^{\mathrm{R}}$	SOFTPLUS
Read modes	π^i_t	Read modes	$\mathbb{R}^{\mathrm{R}} \times \mathbb{R}^3$	SOFTMAX

Table 2: DNC controller output c_t parsed at time-step t, R read heads, N width W memory rows.

State Field	Notation	Description	Domain
Memory Matrix	M_t	Matrix of rows representing memory.	$\mathbb{R}^{\mathrm{N}} \times \mathbb{R}^{\mathrm{W}}$
Usage Vector	u_t	Vector showing memory row usages.	$\mathbb{R}^{\mathrm{N}}$
Precedence Vector	p_t	Previous write vector.	$\mathbb{R}^{\mathrm{N}}$
Link Matrix	L_t	Temporal links between write locations. $L_t[i,j]$ represents if row j was written after i.	$\mathbb{R}^{\mathrm{N}} \times \mathbb{R}^{\mathrm{N}}$

Table 3: Internal state of a DNC at time-step t with N width W memory rows.

Stage 2: Writing — Writing is performed on DNC state defined in Table 3. Memory accesses are performed using weightings $w \in R^{\mathrm{N}}$ that represent a weighting over each row of the memory matrix. To choose our final write weighting w^w_t, we consider two weightings: lookup weighting w^l_t and allocation weighting w^a_t, modulated between using allocation gate g^a_t. To form our lookup weighting, we use a content based lookup LOOKUPBYVALUE from Equation 12, a method that calculates cosine similarity between a vector and each memory matrix row forming a weighting.

$$\text{LOOKUPBYVALUE}(M_t, v)[i] = \frac{M[i] \cdot v}{|M[i]||v|} \tag{12}$$

LOOKUPBYVALUE relies upon knowing some value which seems counter-intuitive as if we knew the value, we wouldn't need to perform the lookup. The controller encodes this however we investigate a modification that masks the vector on lookup allowing a key lookup in Equation 13.

$$\text{MASKEDLOOKUPBYVALUE}(M, v, l)[i] = \frac{(l \odot M[i]) \cdot (l \odot v)}{|l \odot M[i]||l \odot v|} \tag{13}$$

$$\text{Lookup Write Weighting: } w_t^l = \text{LookupByValue}(M, k^w, l) \tag{14}$$

We also consider allocation weighting w_t^a, calculated through usage vector u_t and free list ϕ_t:

$$\text{Usage Vector: } u_t = (u_{t-1} + w_{t-1}^w - u_{t-1} \odot w_{t-1}^w) \odot \prod_{i=1}^{R} (1 - g_{t,i}^f w_{t-1}^{r,i}) \tag{15}$$

$$\text{Free List: } \phi_t = \text{SORTINDICESASCENDING}(u_t) \tag{16}$$

$$\text{Allocation Weight: } w_t^a[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]] \tag{17}$$

The final write weighting is then modulated by gates g_t^w, g_t^a:

$$\text{Write Weighting: } w_t^w = g_t^w (g_t^a w_t^l + (1 - g_t^a) w_t^l) \tag{18}$$

Finally, the state updates are performed using write vector v and erase vector e:

$$\text{Memory Matrix: } M_t = M_{t-1} \odot (1 - w^w e^T) + w^w v^T \tag{19}$$

$$\text{Precedence Vector: } p_t = (1 - \sum_i w_t^w[i]) p_{t-1} + w_t^w \tag{20}$$

$$\text{Link Matrix: } L_t[i, j] = (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i, j] + w_t^w[i] p_{t-1}[j] \tag{21}$$

Stage 3: Reading — We read using our R read heads independently on the state. Each head performs lookup using one of three options by either forward or backward links from the last head read location or an independent lookup by value:

$$\text{Forward Weightings: } w_t^{f,i} = L_t w_{t-1}^{r,i} \tag{22}$$

$$\text{Backward Weightings: } w_t^{b,i} = L_t^T w_{t-1}^{r,i} \tag{23}$$

$$\text{Read Weightings: } w_t^{r,i} = \pi_t^i[w_t^{f,i}, \text{LOOKUPBYVALUE}(M_t, k_t^{r,i}), w_t^{b,i}] \tag{24}$$

$$\text{Read Vectors: } r_t^i = \beta^{r,i} M_t^T w_t^{r,i} \tag{25}$$

Stage 4: Output — Final output for the given time-step t is calculated using interface vector c_t and read vectors r_t^i regressed through a feed-forward neural network. The step allows the DNC to access read vectors or bypass the DNC and use the controller to directly process the input.

$$\text{Predicted Output: } \hat{y}_t = \sigma(W_{vy} c_t + W_{ry} r_t + b_y) \tag{26}$$

A.4 Training

We evaluate our output with a binary cross entropy loss of actual y against predicted $\hat{y}$ shown in Equation 27. Full pipeline differentiability enables our loss to be minimised through differentiation with respect to the network parameters, using stochastic gradient descent (SGD). Dynamic learning rate progressions of SGD include root mean squared back-propagation (RMSProp) that works well in on-line and non-stationary settings by maintaining a moving average of the squared gradient for each weight and dividing the gradient by the root (Tieleman and Hinton, 2012). Similarly, Adaptive Moment Estimation (Adam) by Kingma and Ba (2014) also keeps exponentially decaying average of past gradients, boasting advantages like parameter updates being invariant to gradient rescaling, working with sparse gradients or its ability to naturally performs a form of step size annealing.

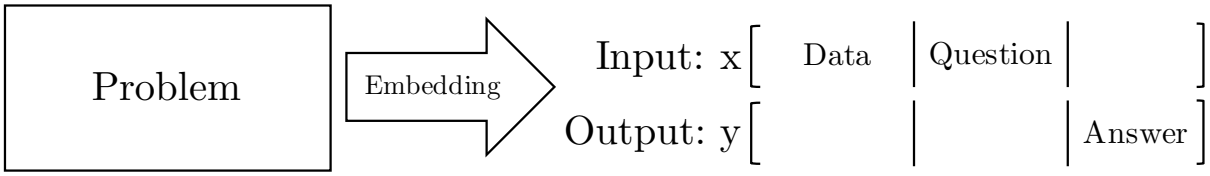
$$\textsc{CrossEntropy}(y, \hat{y}) = -\frac{1}{n}\sum_{t=1}^{n}(\hat{y}_t \cdot log(y_t) + (1 - \hat{y}_t) \cdot log(1 - y_t)) \tag{27}$$

A.5 Regularisation

Due to the nature of the problems such as the data structure challenges, we are able to efficiently generate training examples and so minimise risk of over-fitting. For tasks like the sentiment analysis however, we have limited datasets and thus regularisation is a requirement. Dropout will be investigated to ensure we are not too reliant on particular neurons (Zaremba et al., 2014).

B Sequence Modelling

Similar to the way logic interpreters are fed, we pass a sequence of facts that defines the problem followed by a flag or query to initiate the output:



(28)

To embed a Rubik's cube state, we follow a similar naive approach to Irpan (2016) by feeding each face colour to the model as one hot vectors. By using the Rokicki et al. (2010) hardest cube position sequences, we are able to form optimal move sequences. Due to symmetry and sub-structure optimality we are able to form a huge dataset to train on, regressing the next move.

For sentiment problems, we must embed words as fixed dimension vectors. An approach would to represent each word by unique one-hot vectors. This method is however limited as the size input vectors scales with the amount of words in the dataset. Mikolov et al. (2013) constructs the state of the art Word2Vec embedding that attempts to place similar words like synonyms nearby. We compare both approaches as pre-processing using the Gensim library (Řehůřek and Sojka, 2010).

C *Performance Optimisation*

The solution is optimised for our high performance computing (HPC) GPGPU cluster[8]. To efficiently maximise use of the cluster we choose a modern ML library enables us to quickly scale to hardware resources like GPGPUs. An issue with the DNC, is the sort algorithm in the allocation mechanism performed on CPU due to the difficulty of implementing such algorithms on SIMD architectures and we investigate hardware transfer bottlenecks.

Scaling over multiple nodes is core to the Tensorflow library (Abadi et al., 2015, 2016). Figure 2 displays our master-worker topology that forms a worker for each GPGPU. Our chief worker, contrary to the Tensorflow documentation, maintains visualisations and I/O on a CPU node maximising GPGPU throughput of other workers by not performing training. There are two approaches for distributed training:

- **Synchronous —** Batches are processed at the same time, averaging the update.
- **Asynchronous —** Each worker will update the model when it has finished.

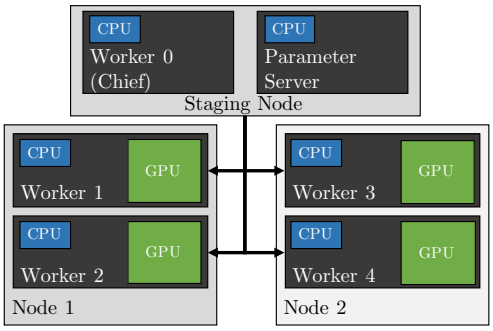


Figure 2: Distributed training master-worker network topology.

Asynchronism suits our varied hardware that process batches at different rates as in synchronous settings, the training would be limited by the slowest worker.

IV RESULTS

In this section we evaluate DNC configurations and modifications, validating against classic RNNs over a variety of problem areas.

A *Architecture*

We begin by refining our models for consistency, progressively explored in this sub-section.

A.1 Controllers

At the heart of a DNC lies a controller, but what makes a good controller? In Figure 5 we yield similar learning performance between DNC LSTM and GRU. We therefore focus on the LSTM for the controller and as our comparison RNN due to the established trust of the model.

SGD fails to improve above 83% accuracy shown in Figure 3 but the embedding from Equation 28 also measures pre-answer 0s thus easily achieving $\frac{n+2}{2n}\% > 50\%$. Dynamic rate descent algorithms like Adam performs more gradually but RMSProp is able to accelerate the process.

[8] *Node 1:* $2\times$ *Intel Xeon CPU E5-2650 v3 @ 2.30GHz; 64GB DDR3 RAM;* $2\times$ *Nvidia Tesla K40c GPUs, 12GB GDDR5 each, 2880 CUDA cores each*
Node 2: $2\times$ *Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz; 64GB DDR3 RAM;* $2\times$ *Nvidia Titan X (Pascal) GPUs, 12 GB G5X each, 3584 CUDA cores each*
Staging: $4\times$ *Intel(R) Xeon(R) CPU E5620 @ 2.40GHz; 16GB DDR3 RAM*

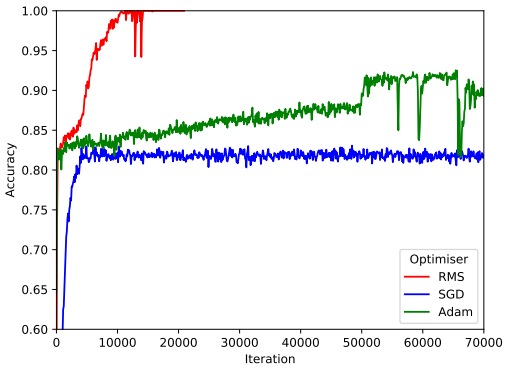


Figure 3: Training accuracy of optimisation algorithms on a DNC Feed-forward (1) against the copy 9 problem.

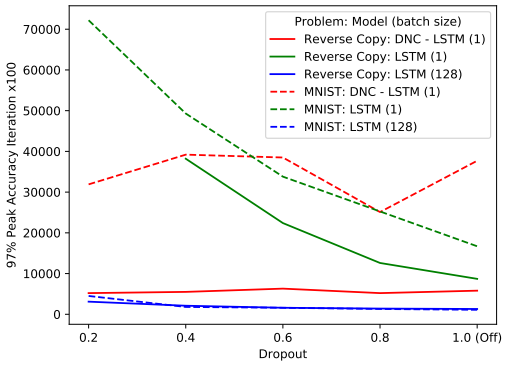


Figure 4: Effect of dropout on reaching 97% peak accuracy against reverse copy and MNIST problems.

A.2 Regularisation

Dropout between recurrent layers is applied to prevent over-fitting our data. Figure 4 displays the effect on the DNC and LSTM, witnessing significant improvement reaching high accuracy at $p = 1.0$ dropout (Off). The LSTM was unable to train in low dropout cases. MNIST yielded similar conclusions implying self-regularising properties of the models are sufficient.

B Problem Applications

Numerous problems can be embedded recurrently. In this subsection, we will explore a variety of problem areas to investigate model characteristics, optimisations and regularisations.

B.1 Data Structure Manipulation

Evaluating the models against basic data structure challenges will give us an insight into the DNCs ability to learn to utilise its memory mechanisms.

Copy, Reverse and Repeat-Copy — Lists are at the heart of most computational problems and so recall tasks are an effective way to measure memory stored in a network.

Mini-batching helps reduce variance between gradient updates, ensuring a gradual and consistent descent. In Figure 5 there is clear difference between the high batch size LSTM and the more erratic learning of a low batch size DNC, yet this allows the DNC to learn the algorithm. Around step $13,000$ for DNC Feed-forward notices an effect we call the penny drop, stemming from the idiom referring to sudden gained understanding like the algorithm reaching a breakthrough. We quantify the penny drop as the iteration where 99.95% accuracy is achieved. The distributions are visualised in Figure 6 showing an LSTM has less variance with higher batch size and thus learning earlier and more consistently. The DNC with LSTM controller performs better than feed-forward, unsurprising with the additional memory of a RNN, however suffers more variance. Extreme cases also occur more often proving confusion between memory sources.

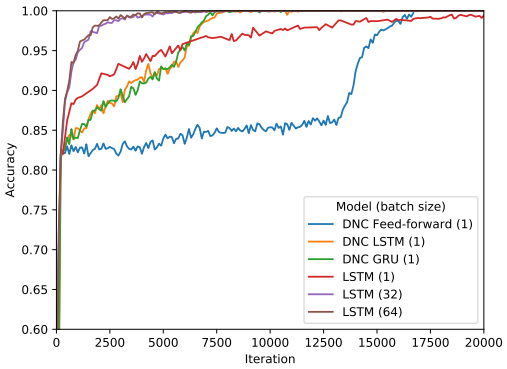


Figure 5: Accuracy of different models against the copy task of length 9.

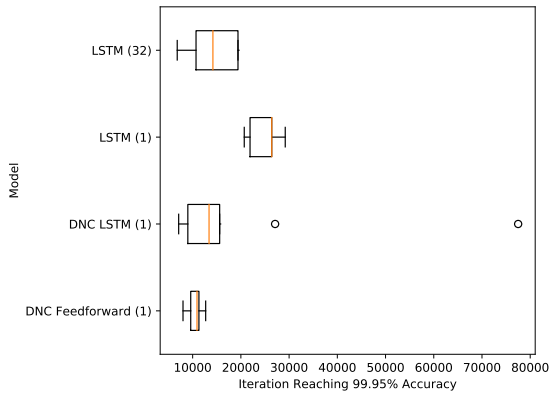


Figure 6: Variance of 9 runs on each model reaching 99.95% accuracy.

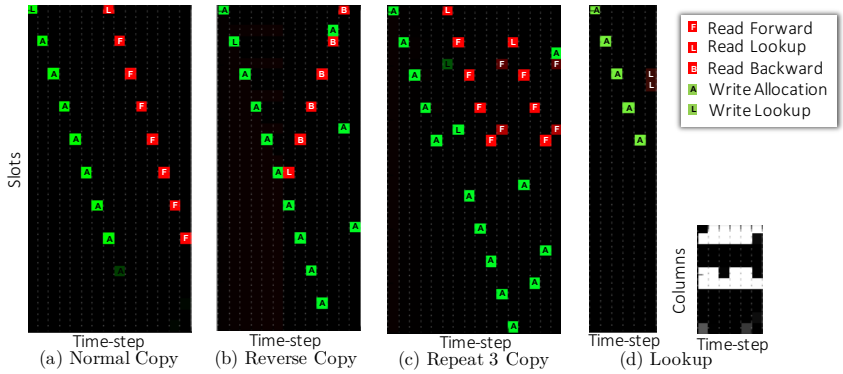


Figure 7: Visualisation of the read and write heads of a trained DNC.

Allocating memory is essential when copying, followed by reading these slots using links validated by Figure 7(a). Extensions to copy include reverse copy that works similar to normal copy but using backwards lookup capability in Figure 7(b). Repeat copy assures the system is not loosing data on lookup in Figure 7(c). These problems demonstrate the power of the link matrix but as previously discussed, this is an expensive mechanism. We replaced the mechanism with a sparse matrix, by next and previous links in the form of a value rather than a weighting for us to use content lookup on. This brings the link matrix into space complexity $\mathcal{O}(NW)$. Upon implementation the DNC circumvents this using lookup by value, likely due to increased complexity that the controller avoids.

Lookup Tables — Table storage is a prevalent programming mechanism, practical for testing the LOOKUPBYVALUE method of the model. First, we visualise the memory matrix in Figure 9 over a copy task, demonstrating the distribution of memory to facilitate lookup by value. Figure 7(d) validates looking up the relevant query, but how does MASKEDLOOKUPBYVALUE compare? The mask is also visualised showing a sparse use of the matrix used for the lookup keys. As the mask is chosen by the controller neural network, there must be an advantage to using the mechanism yet it demonstrated experimentally no influence on training speed.

The pair lookup problem extends lookup where data can come as a bijection, using both sides of the pair to query the other. The task tests the ability to query a single cell of data depending on a bit in the query vector. The DNC shows no struggle to learning this method, however the mask resembles a ones vector and so is querying the full memory row.

Figure 8 displays an interesting phenomena of DNC learning. Feed-forward sees that the model reaches high accuracy earlier for the larger sequences, likely due to the more forceful use of training mechanisms. The LSTM witness early achievements on small sequences due to using the LSTMs built in memory. Upon growing the sequences, this detriments the model as it becomes a battle of mechanisms until we reach length 8 where it has no choice but to use the memory mechanisms properly.

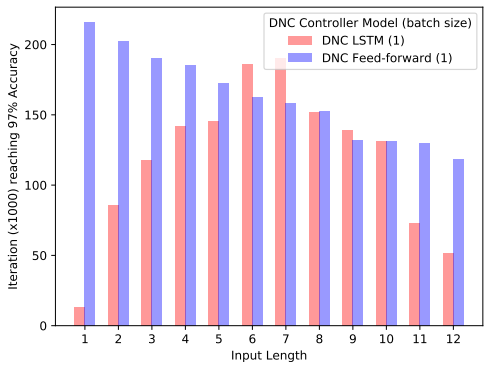


Figure 8: Iteration reaching 97% accuracy on lookup task for sequence lengths comparing DNC controller models.

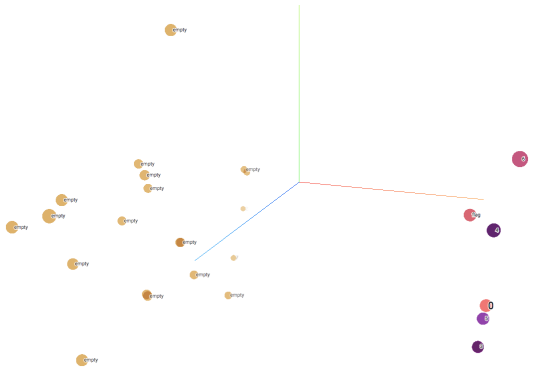


Figure 9: A t-SNE 3D visualisation of the DNC memory matrix over after copy task.

B.2 Computer Vision

Machine learning benchmarks include the MNIST problem by LeCun and Cortes (2010) and we confirmed a DNC is capable of training and most utilising the memory and not simply bypassing to use the controller's neural network. Our naive implementation embed the problem as a row by row approach as opposed to pixel by pixel. This is because it reduces the sequence lengths by 28 times and thus ensures our models aren't affected by length with results shown in Table 4.

B.3 Rubik's Cube

Training the Rubik's cube is possible to an extent and there is potential for the memory heavy architectures to complement the problem. However we discovered a flaw to our logic: We could penalise the system for perfectly valid equivalent moves. In doing so we are expecting the system to perfectly overfit the problem rather than solve it. We need to be able to evaluate moves with respect to how far it becomes from the goal, but with a state space as large as it is this is infeasible without much further research. Unless we reach near perfect accuracy, the model will be useless in most cases.

Problem	Model (batch size)			
	DNC Feed-forward (1)	DNC LSTM (1)	LSTM (1)	LSTM (64)
MNIST	89.24%	95.44%	97.81%	99.06%
IMDb	65.63%	64.32%	65.22%	66.17%
Sarcasm	53.54%	54.28%	57.48%	59.17%

Table 4: Peak test accuracy of the various models.

B.4 Sentiment Analysis

Classifying movie reviews as positive or negative we reach best around 62%, an accuracy level much less than the work of Maas et al. (2011) who managed to achieve 88.89%. This is likely due to the one-hot word embeddings used in addition that limiting our dictionary to 1000 words.

Extending upon basic positive/negative regression, we consider the sarcasm dataset from Filatova (2012) of Amazon reviews. This time, we pre-process the data using Word2Vec, trained on word occurrence distances. This allows us to extract vector space embedding as shown in Figure 10[9]. Words like 'wonderful' have similar semantic meaning to 'fantastic' and so are a good embedding, yet 'boring' could be considered the opposite. This highlights the erratic nature of sarcasm, hinting at the difficulty of the problem.

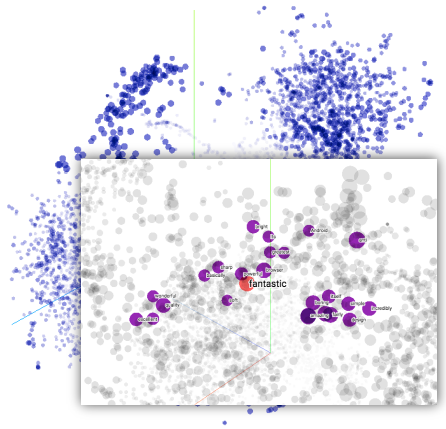


Figure 10: A Word2Vec word embedding of the sarcasm corpus visualised using t-SNE (dim 50, perplexity 30, learning rate 1, 1k iterations)

C Performance Analysis

Ensuring efficient training is a vital component in ML for quick, effective and scalable systems. In this subsection we evaluate the performance bottlenecks and their optimisations.

C.1 Scale Up

Scaling up involves optimising for parallelisation and vectorisation on specialised hardware. Figure 11 compares the iteration rates of various recurrent models, unsurprisingly showing that the DNC is slower than classic RNNs due to the overheads from the mechanisms.

Batch size is another important factor for running on a GPGPU, something further investigated in Figure 12 demonstrating the power of a multi-core CPU on low batch sizes overtaken by the SIMD architectures on higher batch sizes. The high speed up is possible due to the overheads in training and optimisations of the library.

[9] Sarcasm Corpus Visualisation - `projector.tensorflow.org/?config=https://community.dur.ac.uk/a.p.breeze/alastm/config.php` (accessed 03/2017)

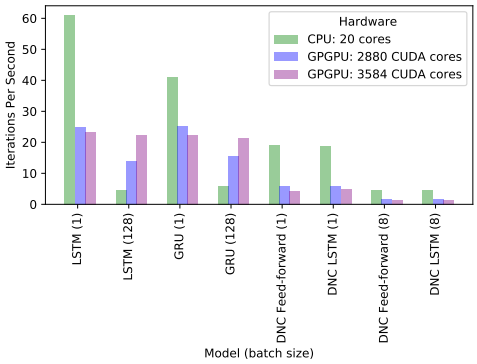


Figure 11: Comparing iteration rates of various models and batch sizes on a copy 9.

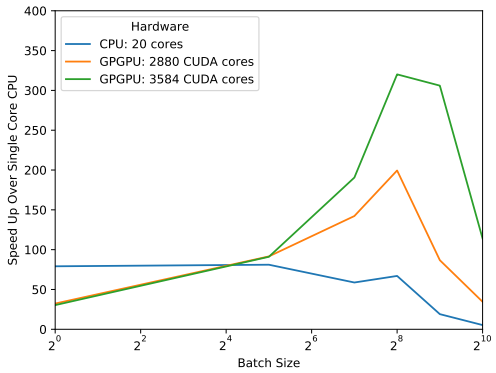


Figure 12: LSTM speed up over single-core CPU on various hardware.

The DNC allocation mechanism was hypothesized as unsuited to a GPGPU architecture, in particular the sorting mechanism. Speed difference after removing the allocation mechanism is shown in Figure 14 where large improvement on GPGPU confirmed our hypothesis. Training suffered due to only allocation by lookup on a blank matrix and led to initialising the matrix:

- **Zeros** — Training was possible, however the system was using multiple memory slots per step and so training was not as algorithmically structured as before.
- **Binary Numbers** — Initialising each memory slot as the binary representation of the id. But cosine similarity can still have the same distance to various binary vectors.
- **Consistently Random** — Initialised a subset of each row to to a uniformly random vector, but by fixing the seed of the random generator it is trainable.

The above techniques helped bypass sorting, however the system was less predictable implying stability added by the implementation. Forcing single slots helped, however again further complicated the model.

C.2 Distributed Speed Up

Different speed up over various GPUs achieve various batch sizes visible in Figure 12, consolidating the asynchronous distributed architecture decision to maximise utilisation. Maximising the workers is vital, however upon analysis of the chief worker we found that the chief was only achieving 65.5% of the performance of other workers. This is due to the overheads of a chief including initialization, checkpoints, visualisations, and recovery. Separating the chief onto a separate CPU process balances worker load and maximised GPGPU usage.

Distributed speed up was witnessed Figure 13, achieving near linear improvements. Even a single worker over the serial approach saw great speed up due to separating some work to a chief, maximising our hardware utilisation.

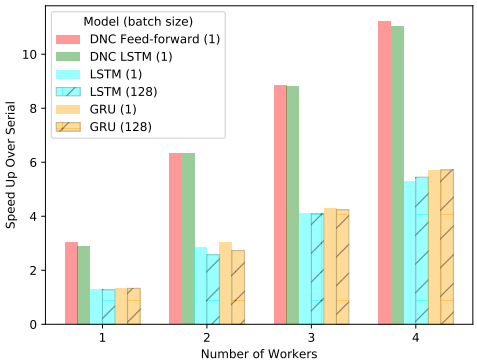


Figure 13: Distributed speed up over serial execution on the copy task.

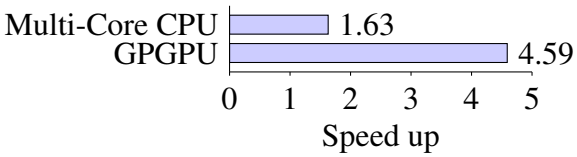


Figure 14: DNC speed up when allocation mechanism is removed.

V EVALUATION

In this section, we draw on the strengths and shortcomings of the models in addition to analysing our research methodology. We reflect back to the original question: 'How do concept recurrent architectures compare to state of the art in memory, speed, complexity and their potential?'

A Strengths

A highly versatile system facilitated testing of the architectures over a span of different problem areas. The DNC's algorithmic ability showed promise for neural computational models where the DNC was able to solve problems, where classic RNNs could only approach this limit. Identifying concepts like the penny-drop effect that are key for analysing these kinds of models. Applying different learning algorithms, regularisations and hyper-parameter optimisation provided valuable insight and optimism for future research in the field.

Profiling quantified the merits to scaling over a distributed system demonstrating significant speed-up with our own topology. Our re-engineered algorithms like the lookup mechanism proven to be used by the neural network. Altogether this resulted in our highly optimised state of the art implementation capable of a range of problem areas.

B Limitations

A DNC is an unorthodox architecture to train and optimise. Examples include batch size gaining better training over classic RNNs on low batch sizes, but on higher batch size training became detrimentally slow. We showed the DNC as capable at learning algorithms but not always beneficial over classic RNNs with some cases bypassing the memory mechanisms to instead use the controller. Scalability was another concern to the DNC, where some state size was in $\mathcal{O}(n^2)$. We explored ideas to store links sparsely demonstrating the difficulty training these mechanisms.

SIMD parallelisation has become popular in ML and we were able to successfully witness speed up with GPGPU architectures. Performance analysis did however identify bottlenecks like the memory allocation that we were unable to re-engineer, justifying original approaches

complexity. Libraries were capable of our architectures, however huge computational graphs do not scale well. Our research provides insight for future improvement to the libraries.

C Approach

Due to the youth of the research, a dynamic and agile methodology was necessary. As the primary research had not released the code, our base model stemmed from open source projects and their insights. Our modularised testing and visualisation system facilitated many problems and optimisations. Visualisations enabled efficient understanding of our models through a live web portal allowed us to easily manage the extensive runs in real time.

Upon repeating the project, we would focus more on classic NLP problems rather than the Rubik's cube that was harder than anticipated, requiring large amounts of discrete maths research.

VI CONCLUSIONS

Our research evaluated and experimented with the cutting edge DNC neural architecture, studying the characteristics against a range of sequential problem against classic recurrent models like the LSTM. We successfully ratify our original project aims:

1. We **validated** state of art and concept recurrent architectures against research problems.
2. Our **optimisations** to the architectures have pushed the power of the models.
3. By **visualising** the models in classic and more abstract ways, we were able to explore the inner workings to understand and verify the models.
4. Our research **explored** adjustments to the models, investigating alterations to the methods.

Having evaluated the DNC (Graves et al., 2016) against the popular LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Bahdanau et al., 2014), we discovered merits and shortcomings. We applied the models against a core data structure tasks like copy and lookup tables (Henaff et al., 2016). DNC analysis coined the term 'Penny Drop' that defines a training effect we have identified as key in training of such models.

Benchmarks against real world problems like MNIST problem (LeCun and Cortes, 2010), in addition to IMDb sentiment analysis problem (Maas et al., 2011) demonstrated versatility to more than just algorithmic tasks. We applied the models to sentiment analysis to classify shopping reviews as sarcastic (Filatova, 2012), applying state of the art word embeddings validated by visualisations (Mikolov et al., 2013; van der Maaten and Hinton, 2008). The conclusions of Irpan (2016) were consolidated through justifying the difficulty of the Rubik's cube problem.

Improvements applied to the DNC including regularisations concluded the self regularisation properties of the models (Pascanu et al., 2012). Distributed learning was explored to form our topology that maximises utilisation (Abadi et al., 2015, 2016). Profiling and visualisations identified potential that inspired improvements to the model.

The DNC shows potential in an LSTM dominated field, and we demonstrated solid step towards bridging modern day computation and neural models. Our research concludes high potential and demonstrated an improvement that forms a powerful state of the art system.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org. **URL:** *http://tensorflow.org/*

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A. et al. (2016), Tensorflow: A system for large-scale machine learning, *in* '12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)', USENIX Assoc., GA, pp. 265–283.

Bahdanau, D., Cho, K. and Bengio, Y. (2014), 'Neural machine translation by jointly learning to align and translate', *CoRR* **abs/1409.0473**.

Bengio, Y., Simard, P. and Frasconi, P. (1994), 'Learning long-term dependencies with gradient descent is difficult', *Trans. Neur. Netw.* **5**(2), 157–166.

Chung, J., Gülçehre, Ç., Cho, K. and Bengio, Y. (2014), 'Empirical evaluation of gated recurrent neural networks on sequence modeling', *CoRR* **abs/1412.3555**.

Elman, J. L. (1990), 'Finding structure in time', *Cognitive Science* **14**(2), 179–211.

Filatova, E. (2012), Irony and sarcasm: Corpus generation and analysis using crowdsourcing, *in* 'In Proc. Int. Conf. Language Resources and Evaluation', ELRA.

Graves, A., Wayne, G. and Danihelka, I. (2014), 'Neural turing machines', *CoRR* **abs/1410.5401**.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I. et al. (2016), 'Hybrid computing using a neural network with dynamic external memory', *Nature* .

Henaff, M., Szlam, A. and LeCun, Y. (2016), 'Orthogonal rnns and long-memory tasks', *CoRR* **abs/1602.06662**.

Hochreiter, S. and Schmidhuber, J. (1997), 'Long short-term memory', *Neural Comput.* **9**(8), 1735–1780.

Irpan, A. (2016), 'Exploring boosted neural nets for rubik's cube solving'.

Karpathy, A. (2015), 'The unreasonable effectiveness of recurrent neural networks'. **URL:** *karpathy.github.io/2015/05/21/rnn-effectiveness/*

Karpathy, A., Johnson, J. and Li, F. (2015), 'Visualizing and understanding recurrent networks', *CoRR* **abs/1506.02078**.

Kingma, D. P. and Ba, J. (2014), 'Adam: A method for stochastic optimization', *CoRR* **abs/1412.6980**.

LeCun, Y. and Cortes, C. (2010), 'MNIST handwritten digit database'. **URL:** *http://yann.lecun.com/exdb/mnist/*

Lipton, Z. C., Kale, D. C., Elkan, C. and Wetzel, R. C. (2015), 'Learning to diagnose with LSTM recurrent neural networks', *CoRR* **abs/1511.03677**.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y. et al. (2011), Learning word vectors for sentiment analysis, *in* 'In Proc. Meet. of Assoc. for Comp. Linguistics: Human Language Technologies', Assoc. for Comp. Linguistics, pp. 142–150.

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013), 'Efficient estimation of word representations in vector space', *CoRR* **abs/1301.3781**.

Oraby, S., Harrison, V., Reed, L., Hernandez, E., Riloff, E. and Walker, M. (2016), Creating and characterizing a diverse corpus of sarcasm in dialogue, *in* 'Proc. of the Special Interest Group on Discourse and Dialogue', Assoc. for Comp. Linguistics, pp. 31–41.

Pascanu, R., Mikolov, T. and Bengio, Y. (2012), 'Understanding the exploding gradient problem', *CoRR* **abs/1211.5063**.

Řehůřek, R. and Sojka, P. (2010), Software Framework for Topic Modelling with Large Corpora, *in* 'Proc. LREC Workshop on New Challenges for NLP Frameworks', ELRA, pp. 45–50.

Rokicki, T., Kociemba, H., Davidson, M., and Dethridge, J. (2010), 'God's number is 20.'. **URL:** *http://www.cube20.org/*

Rumelhart, D., Hinton, G. and Williams, R. (1986), 'Learning representations by back-propagating errors', *Nature* **323**(6088), 533–536.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L. et al. (2016), 'Mastering the game of go with deep neural networks and tree search', *Nature* **529**(7587), 484–489.

Tieleman, T. and Hinton, G. (2012), 'Lecture 6.5-rmsprop, coursera: Neural networks for machine learning', *University of Toronto, Tech. Rep* .

Turing, A. (1950), 'Computing machinery and intelligence', *Mind* **LIX**(236), 433.

Turing, A. M. (1937), 'On computable numbers, with an application to the entscheidungsproblem', *In Proc. London Mathematical Society* **2**(1), 230–265.

van den Oord, A., Kalchbrenner, N. and Kavukcuoglu, K. (2016), 'Pixel recurrent neural networks', *CoRR* **abs/1601.06759**.

van der Maaten, L. and Hinton, G. E. (2008), 'Visualizing high-dimensional data using t-sne', *Journal of Machine Learning Research* **9**, 2579–2605.

Werbos, P. (1990), 'Backpropagation through time: what it does and how to do it', *Proc. of the IEEE* **78**(10), 1550–1560.

Weston, J., Bordes, A., Chopra, S. and Mikolov, T. (2015), 'Towards ai-complete question answering: A set of prerequisite toy tasks', *CoRR* **abs/1502.05698**.

Williams, R. and Peng, J. (1990), 'An efficient gradient-based algorithm for on-line training of recurrent network trajectories', *Neural Computation* **2**, 490–501.

Zaremba, W., Sutskever, I. and Vinyals, O. (2014), 'Recurrent neural network regularization', *CoRR* **abs/1409.2329**.