EINS: LONG SHORT-TERM MEMORY WITH EXTRAPOLATED INPUT NETWORK SIMPLIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper contrasts the two canonical recurrent neural networks (RNNs) of long short-term memory (LSTM) and gated recurrent unit (GRU) to propose our novel light-weight RNN of *extrapolated input for network simplification* (EINS). We treat LSTMs and GRUs as differential equations, and our analysis highlights several auxiliary components in the standard LSTM design that are secondary in importance. Guided by these insights, we present a design that abandons the LSTM redundancies, thereby introducing EINS. We test EINS against the LSTM over a carefully chosen range of tasks from language modelling and medical data imputation-prediction through a sentence-level variational autoencoder and image generation to learning to learn to optimise another neural network. Despite having both a simpler design and fewer parameters, this simplification either performs comparably, or better, than the LSTM in each task.

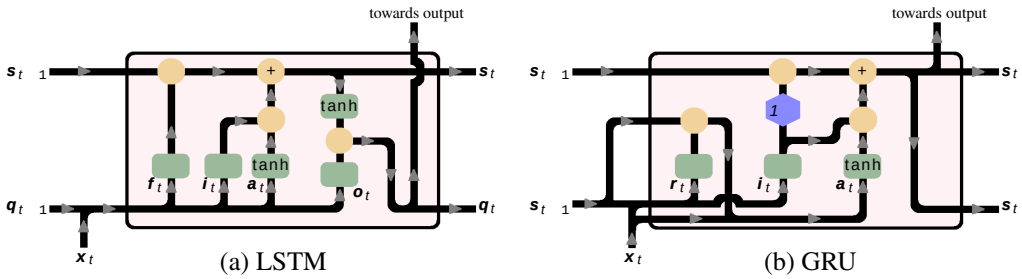
1 INTRODUCTION

Neural networks are powerful universal approximators that are difficult to interpret. This paper studies the forward pass of the two canonical *recurrent neural networks* (RNNs) of *long short-term memory* (LSTM) (Hochreiter & Schmidhuber, 1997) and *gated recurrent unit* (GRU) (Cho et al., 2014). Advancements in RNNs have pushed the state-of-the-art for a variety of natural language processing problems, including speech recognition (Sundermeyer et al., 2012), text modelling (Kim et al., 2016), and neural machine translation(Kalchbrenner & Blunsom, 2013). RNNs inspire the architectural design of the attention mechanism (Bahdanau et al., 2015) and the bidirectional design (Schuster & Paliwal, 1997); and more importantly, they are prevalent and actively studied in multiple fields of machine learning, including computer vision (Shi et al., 2015; Sun & Fu, 2019) and meta-learning (Vinyals et al., 2016; Ravi & Larochelle, 2017). Though LSTM and GRU networks have been extensively applied in a wide range of tasks, there are difficulties in justifying whether they are optimal designs for RNNs (Jozefowicz et al., 2015; Chung et al., 2014). In addition, they both have intricate designs with gated units employed to synchronise network memories, but the significance of these individual components are unclear, and understanding the network's learnt utility requires empirical explorations (Karpathy et al., 2015; Wu & King, 2016).

By treating LSTMs and GRUs as systems of differential equations, our study seeks to understand the intricacies in the updates of their network memories. These two networks behave similarly to differential equations — iteratively, they provide the network memories, the cells, with small packets of incremental updates. From our mathematical analysis, we first reveal the presence of multiple auxiliary components. Consequently, we proceed to present an RNN with a novel *extrapolated input for network simplification* (**EINS**) that has a much simpler architectural design than the LSTM and the GRU, while enjoying comparable or better generalisation ability. Furthermore, our analysis shows that gated units are analogous to timescales in differential equations and dictate speeds of variable propagation, thereby increasing network interpretability.

Our novel EINS RNN is based on the LSTM architecture. We dispense with the previously mentioned LSTM redundancies, and introduce a procedure that extrapolates the network input. EINS has a simple architectural design and, dependent on the task, can remove up to 85.4 per cent[1] of the

[1] 85.4% for image generation and 63.3% for language modelling. See **Section 5** — Experiments for details.



(a) LSTM (b) GRU

Figure 1: Architectural designs of the LSTM and the GRU RNN

parameters of the standard LSTM RNN module. We test[2] EINS against the LSTM on a wide variety of carefully chosen tasks. Our tasks include language modelling (Mikolov et al., 2010) for RNNs as generative models; medical data imputation (Che et al., 2018; Cao et al., 2018) and sentence-level autoencoders (Bowman et al., 2016) for RNNs as encoders; image generation (Gregor et al., 2015) for RNNs as decoders; and on the meta-learning task of learning to learn to optimise another neural network (Ravi & Larochelle, 2017; Andrychowicz et al., 2016) for RNNs to run online for a variable length. For all tasks, EINSs were able to yield comparable or better results than LSTMs.

The main contributions of this paper are listed as follows:

We treat LSTMs and GRUs as differential equations and reveal that gated units serve as timescales to dictate neural value propagation, thereby increasing interpretability.

The same mathematical analysis allows us to present an argument on the inferior computational power of GRUs against LSTMs under identical hyperparametric setups.

The same mathematical analysis allows us to identify removable auxiliary terms in LSTMs.

We introduce EINS, a novel RNN that removes up to 85.4 per cent[1] parameters of the LSTM, with comparable or better computational power than the LSTM.

2 LSTM AND GRU

This section introduces the canonical RNNs of LSTM and GRU. *Without loss of generality, we use LSTM-terminologies to describe GRUs*.

For every time step t, the LSTM RNN receives input x_t to compute

System (2.1): the LSTM RNN

$$\begin{aligned}
&\text{a set of gated units:} && f_t; i_t; o_t = && (\mathbf{W}_{fF;I;Og} x_t + \mathbf{W}_{RfF;I;Og} q_{t\ 1} + b_{fF;I;Og}),\\
&\text{and an internal input:} && a_t = && \tanh(\mathbf{W}_A x_t + \mathbf{W}_{RA} q_{t\ 1} + b_A),\\
&\text{to update the cell state:} && s_t = && f_t \quad s_{t\ 1} + i_t \quad a_t,\\
&\text{and the hidden state:} && q_t = && o_t \quad \tanh(s_t).
\end{aligned}$$

In a similar but slightly simpler fashion, the GRU is driven by input x_t to compute

System (2.2): the GRU RNN

$$\begin{aligned}
&\text{a set of gated units:} && i_t; r_t = && (\mathbf{W}_{fI;Rg} x_t + \mathbf{W}_{RfI;Rg} s_{t\ 1} + b_{fI;Rg}),\\
&\text{and an internal input:} && a_t = && \tanh(\mathbf{W}_A x_t + \mathbf{W}_{RA}(r_t \quad s_{t\ 1}) + b_A),\\
&\text{to update the cell:} && s_t = && (1 \quad i_t) \quad s_{t\ 1} + i_t \quad a_t.
\end{aligned}$$

Operators and tanh represent the sigmoid function and the hyperbolic tangent function respectively; whereas $\mathbf{W}$, $\mathbf{W}_R$, and b represent the forward connections, the recursive connections, and the biases. Given and as the input and the hidden dimensions, we have $\mathbf{W}$ 2 R , $\mathbf{W}_R$ 2 R , x_t 2 R ; the gated units, along s_t and q_t, and the biases, are of R . Refer to **Figure 1** for visual representations of the networks.

Both networks synchronise the update of the network memory s_t with gated units. For an LSTM, the forget gate f_t releases partial memory of the previous instance $s_{t\ 1}$, the input gate i_t controls the influx of the newly formulated information a_t, and the hidden state q_t is the transformed memory with controlled exposure from the output gate o_t. In comparison, a GRU couples its forget gate and input gate, utilises its raw cell as its hidden state, and employs a reset gate r_t to control the strength of recurrent connections for generating the internal input a_t.

[2]Implementation of our algorithm will be available at https://github.com/anonymous_author/EINS_RNN/

3 LSTMS AND GRUS AS DIFFERENTIAL EQUATIONS

In this section, we study LSTMs and GRUs as systems of dynamic Hopfield equations. We discuss why the cells should be modelled as differential equations; and why the Hopfield network is a suitable starting point. From there, we introduce more mathematical concepts and *eventually derive the GRU formulation*.

3.1 CELL STATES AS DIFFERENTIAL EQUATIONS FOR CONTINUOUS UPDATES

As defined in **Systems (2.1)** and **(2.2)**, the cell state s_t receives cumulative incremental updates from a_t over a series of instances. That is, similar to variables of differential equations, *their future values are additively updated from their past values*. For this reason, we define the update of the cell as

$$s_t' = a_t; \tag{1}$$

which is equivalent (in its discrete form) to

$$s_t = s_{t-1} + h a_t; \tag{2}$$

where a_t is an arbitrary package of update, with a small positive constant timescale h.

After asserting the intrinsic differential nature of the cell states, we revise the appropriateness of **Equation (1)**. That is, we question "*What kind of differential equation should be considered as a natural starting point?*"

3.2 HOPFIELD NETWORKS FOR ROBUST PLASTIC MEMORY

In the present work, we model the cell states as Hopfield networks (Hopfield, 1988). Dynamic Hopfield networks are well-studied in theoretical physics (Sompolinsky & Zippelius, 1982; Amit et al., 1985; Crisanti et al., 1986; Sompolinsky et al., 1988) and have two desirable qualities. First, there exist network equilibria — autonomous convergent configurations among the neurons. This corresponds to the capability to develop stable robust memories. Second, the network has phase transitions — parametric transitional boundaries for internal degrees of freedom to successively fall out of equilibrium. This corresponds to the plasticity of the memories.

Dynamic Hopfield networks have also been extensively studied in statistical machine learning. In the field of reservoir computing (Lukoševičius & Jaeger, 2009), they serve as the foundations of the powerful class of RNN known as the echo state network (Jaeger, 2001; 2002; Sussillo & F Abbott, 2009). For all of the reasons above, we model s_t as the dynamic Hopfield network

$$s_t' = s_{t-1} + a_t: \tag{3}$$

3.3 DERIVING THE GRU FORMULATION

Similar to how **Equation (2)** was rewritten from **Equation (1)**, we now rewrite **Equation (3)** into

$$s_t = s_{t-1} + h(s_{t-1} + a_t); \tag{4}$$

$$= (1-h)s_{t-1} + h a_t: \tag{5}$$

The readers should find clear resemblances between **Equation (5)** and the cell state of the GRU RNN, where h corresponds to the input gate i_t.

Hence from **Equation (5)**, we understand that the gated units correspond to the timescales for recurrent neural variables. A natural question that follows is "*Is it sensible, for timescales of an RNN to behave as variables, instead of constants, for generating deterministic outputs?*"

Here, we draw inspiration from the Newton-Raphson method (Ypma, 1995), an iterative method for finding roots of an arbitrary continuous differentiable function $g(s)$. The method derives better approximated root-values of s_t from the previous approximations of s_{t-1} via

$$s_t = s_{t-1} - \frac{g(s_{t-1})}{g'(s_{t-1})}; \tag{6}$$

which is graphically equivalent[3] to satisfying $(s; y) = (s_t; 0)$ with $h = g'$ for

$$y = g(s_t) + h(s_t)(s - s_t): \tag{7}$$

[3] Refer to Ypma (1995, p. 535, Section 3, Figure 2).

In **Equation (7)**, the term $h(s_t)$ serves the same functional purpose as the term h in **Equation (5)**. Thus, *it is sensible to include variable timescales in recursive systems for deterministic outcomes*.

In accordance with this analysis, if we were to define

$$h = h(s_t) = i_t = (\mathbf{W}_I x_t + \mathbf{W}_{RI} s_{t-1} + b_I); \quad \text{and} \tag{8}$$

$$a_t = \tanh(\mathbf{W}_A x_t + \mathbf{W}_{RA}(r_t \odot s_{t-1}) + b_A); \quad \text{where} \tag{9}$$

$$r_t = (\mathbf{W}_R x_t + \mathbf{W}_{RR} s_{t-1} + b_r); \tag{10}$$

and substitute **Equations (8)–(10)** into **Equation (5)**, we derive the GRU of **System (2.2)**. That is, our mathematical analysis therein this section provides an illuminating perspective on viewing existing RNNs as dynamic Hopfield model (**Equation (3)**) with variable timescales (**Equation (7)**).

Here, we emphasise that **Equations (8)–(10)** *present simply one of the many possible network designs*. That is, *we may choose our network architecture as long as they fit under the dynamical mathematical framework*. In **Section 4**, we present a well-justified step-by-step network simplification for the LSTM for a more optimal architectural design for RNNs.

3.4 Theoretical disparity of the computational power of LSTMs and GRUs

Our claim begins with an important remark stated in a paper by Greff et al. (2017) that addressed an in-depth lesion study on trained LSTMs across a wide variety of tasks. For all tasks, the removal of the output gate o_t severely impaired LSTM performances. The authors of that paper thus conjectured that o_t is needed to prevent unbounded cells from propagating through the network and destabilise learning. Our analysis in **Subsection 3.3**, as elaborated below, is in line with their conjecture.

In **Equation (5)**, we saw that $1 - h$ serves as the timescale for s_{t-1} and h as that of a_t; furthermore, we saw that h can be defined as the input gate i_t via **Equation (8)**. Hence, the GRU does not need a forget gate f_t given the presence of i_t. Under the same mathematical narrative, if an extra f_t were to be introduced to replace $1 - i_t$, numeric overflow will occur when $f_t < 1 - i_t$. That is, when the removal rate of old memory s_{t-1} is lower than the replenishment rate of new information a_t. Thus, LSTMs require o_t to prevent the regurgitation of uncontrollably growing memories.

However, the training stability of the GRU comes with the detrimental price of a simplified solution space. For GRUs, the cell is solely conditioned on i_t alone; whereas cells of LSTMs are conditioned on the superposition of $(f_t; i_t)$. That is, the update of the LSTM cell explores for an additional subspace of $\mathbb{R}$ for every marginal dimension that it is conditioned on i_t. Dimensionality is the network hidden dimension, see **Section 2** for details. In other words, LSTMs have the potential to have more equilibrium states, or more complicated periodic equilibrium expressions, than the GRUs. This quality makes the LSTM memory considerably more versatile than that of the GRU.

This theoretical disparity in memorisational computational power is also observed in practice. The empirical sequential modelling study of Chung et al. (2014) showed that GRUs perform comparably to LSTMs when the two networks had similar amounts of parameters[4] — *for when the GRUs had larger hidden dimensionalities than the LSTMs*. In other words, GRUs are less powerful than LSTMs for the same number of hyper-parameters.

For reasons listed in this subsection, EINS — our novel light-weight RNN — to be introduced in the next section, will be a simplification from the LSTM RNN, instead of the GRU RNN. Furthermore, to ensure all experiments are conducted under identical hyper-parameter settings, EINS will be benchmarked against LSTMs, instead of GRUs, in the experimental section of this paper.

4 EINS, Extrapolated Inputs for Network Simplification

In **Subsection 3.3**, we chose **Equations (8)–(10)** for recovering the GRU formulation; but now we question "*Are GRUs and LSTMs optimal architectural designs for RNNs?*" In this section, we present arguments of the existences of redundant terms, which we now identify and remove to propose our novel light-weight RNN of EINS. EINS is simplified from the LSTM instead of the GRU, refer to **Subsection 3.4** for justifications.

[4] Refer to Chung et al. (2014, p. 6, Section 4.2,Table 1).

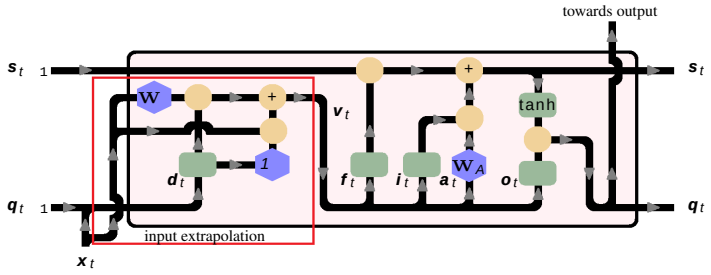


Figure 2: Architectural design of EINS

4.1 Removing unnecessary terms in the package update

Equation (3) of $s_t^0 = s_{t-1} + a_t$ dictates the cell s_t to receive incremental package updates from a_t. However, no mathematical constraints were imposed on the update to undergo nonlinear transformation. Thus, we simplify the statistical engineering effort of a_t as

$$a_t = \mathbf{W}_A x_t; \tag{11}$$

4.2 Removing unnecessary synaptic connections

As defined in **System (2.1)**, the LSTM gated units take the form of

$$f_t; i_t; o_t = (\mathbf{W}_{fF;I;Og} x_t + \mathbf{W}_{RfF;I;Og} q_{t-1} + b_{fF;I;Og}).$$

However, there was no indication in **Equation (8)** for the need of recurrent connections. For this reason, we simplify the gated units as

$$f_t; i_t; o_t = (\mathbf{W}_{fF;I;Og} x_t); \tag{12}$$

4.3 Input Extrapolation

In **Subsection 4.2**, we remove the recurrent connections because of their non-necessity under the lens of the dynamic mathematical study presented in **Subsection 3.3**. However, the same mathematical analysis has not provided insights towards *why the original LSTM design functions smoothly with the inclusion of the redundant terms*. In this paper, we propose to interpret the recurrent inputs of $\mathbf{W}_R q_{t-1}$ as regularisations similar to batch normalisation (Ioffe & Szegedy, 2015). That is, they ease the training of the feedforward input in activation functions.

With this asserted interpretation, we can then simplify the LSTM regularisation mechanism via regularising the feedforward pre-activated neurons of $\mathbf{W}x_t$ prior to their projections under $\mathbf{W}$. That is, for the standard LSTM gated units shown in **Equation (12)**, we substitute the original input of x_t with the newly regulated input of v_t, $x_t \quad v_t$, where

$$v_t = (1 \quad d_t) \quad x_t + d_t \quad \mathbf{W} \; x_t; \quad \text{with} \tag{13}$$

$$d_t = (\mathbf{W}_D x_t + \mathbf{W} \quad q_{t-1} + b_D); \tag{14}$$

To summarise, the EINS RNN encompasses

System (4.1): the EINS RNN

the self-diagnosis gate:	$d_t = (\mathbf{W}_D x_t + \mathbf{W} \quad q_{t-1} + b_D),$
the extrapolated input:	$v_t = (1 \quad d_t) \quad x_t + d_t \quad \mathbf{W} \; x_t,$
the orthodox gated units:	$f_t; i_t; o_t = (\mathbf{W}_{fF;I;Og} v_t),$
the internal input:	$a_t = \mathbf{W}_A v_t,$
the update of the cell state:	$s_t = f_t \quad s_{t-1} + i_t \quad a_t,$
and the hidden state:	$q_t = o_t \quad \tanh(s_t).$

The newly introduced components have dimensionalities $d_t; v_t; b_D \; 2 \; \mathrm{R}$, $\mathbf{W} \; ; \mathbf{W}_D \; 2 \; \mathrm{R}$, and $\mathbf{W} \quad 2 \; \mathrm{R}$; those remaining ones follow the dimensionalities listed in **Section 2**. We stress that $\mathbf{W}_D$ has fewer parameters than the orthodox feedforward synapses of $\mathbf{W}$, and similarly $\mathbf{W}$ has fewer parameters than $\mathbf{W}_R$. EINSs have significantly less parameters than LSTMs through the removal of recurrent synaptic connections from the gated units. When implemented in PyTorch (Paszke et al., 2017)[5], EINSs have $2^2 + 5 \quad + 2$ parameters in contrast to LSTMs' $4 \quad + 4^2 + 8$.

[5]The PyTorch implementation for LSTM have 2 biases each gate, refer to their source code for details.

5 EXPERIMENTS

Our tasks test RNNs as generative models, as encoders, as decoders, and for their abilities to run for variable lengths of time. Generative capabilities test the ability to represent and manipulate high-dimensional probability distributions (Goodfellow, 2016). Encoding powers test the efficiency in leveraging statistical information to produce a vector space with meaningful sub-structure (Pennington et al., 2014). Decoding powers are similar to generative capabilities but additionally test for the reconstruction powers given outsourced encoded features. Finally, the ability to run for variable lengths tests the robustness against the regurgitation of compounded prediction errors conditioned on unseen sequential context at test time, from those seen during training (Goyal et al., 2016).

We include tasks of a variety of different machine learning disciplines, including natural language processing, computer vision, and meta-learning. GRUs are not included for reasons previously justified in **Subsection 3.4**. Due to space constraint, we refer readers to the Appendicies for training details and additional results. Last, we stress that experiments are included to demonstrate LSTM-like performances exhibited by EINS; our goal is ***not*** to compete for the state-of-the-art results.

5.1 RESULTS AS GENERATIVE MODELS

Language models (LMs) employ RNNs to construct probabilistic predictions of the next word given preceding ones. Following Merity et al. (2018), we trained 3-layer RNN-LMs with 400-word embedding dimension and 1150-hidden dimension on word-level *Penn TreeBank* (PTB) (Mikolov et al., 2010). Results in **Table 1** show that EINS removed up to 63.3% parameters of an LSTM module, and that EINSs and LSTMs yielded respective perplexity of 57.44 and 58.32. Refer to **Appendix A** for additional experiments, statistical analysis, and for architectural and training details.

5.2 RESULTS AS ENCODERS

Following Bowman et al. (2016), we employed one-layer RNN *variational autoencoders* (VAEs) as extensions to LMs to prevent inferences over unknown words for *sentence imputation*. The RNN-VAEs had 300-word embedding dimension with 256-hidden dimension to encode Gaussian prior distributions on cells[6] of one-layer GRU-LMs for word-level PTB. **Table 2** presents some imputed words for original sentences with their final four tokens removed. Refer to **Appendix B** for additional experiments, length analysis, NLL analysis, and for training details.

5.3 RESULTS AS SEMI-ENCODER DISCRIMINATIVE MODELS

Che et al. (2018) presented the *GRU-D* architecture as an extension to GRUs for *medical data imputation-prediction*. It was test on PhysioNet (Silva et al., 2012), a challenging real-world clinical task with 78% missing rate, for mortality prediction. The network first imputes for missing values and then makes predictions. Cao et al. (2018) extended on this concept and proposed the LSTM-based *bidirectional recurrent imputation for time series* (BRITS). We tested one-layer RNN-BRITS with 70-input dimension and 100-hidden dimension on PhysioNet, and reported the *area under the ROC curve* (AUC) (Bradley, 1997), *mean absolute error* (MAE), and *mean relative error* (MRE). **Table 3** shows that EINS removed 34.4% parameters of the LSTM, and that LSTMs and EINSs scored respective AUC/MAE/MRE of 0.866/0.270/0.382 and 0.850/0.261/0.370. Refer to **Appendix C** for results on different BRITS variants, and for architectural and training details.

5.4 RESULTS AS DECODERS

Gregor et al. (2015) proposed the *deep recurrent attention writer* (DRAW) architecture to *generate images* of photographs of house numbers (Goodfellow et al., 2014). DRAW employs single-layer LSTMs with 100-input dimension and 800-hidden dimension to decode pixel information in latent distributions to reassess previously generated images. We followed their setup and **Figure 3** shows that DRAW-EINS generated images of similar quality to those of DRAW-LSTM. The LSTMs had 2.88 million parameters, while the EINSs had only 0.42 million, removing 85.4% parameters. Refer to **Appendix D** for examples of sequential image generation, and architectural and training details.

[6] As mentioned in **Section 2**, we use LSTM-terminologies to describe GRUs.

Table 1: Language modelling on word-level PTB

Model	Test Perplexity / Equivalent Loss	Dimensionality	# Parameters
Merity et al. (2018)			
3-Layer LSTM-LM	57.3 / 4.0483	identical as the cell below	
our implementation			
3-Layer LSTM-LM	58.32 / 4.0695	L1: (,) = (400, 1150)	7.13M
		L2: (,) = (1150, 1150)	10.58M
		L3: (,) = (1150, 400)	7.13M
our proposed model			
3-Layer EINS-LM	57.44 / 4.0507	L1: (,) = (400, 1150)	2.62M (63.3% less)
		L2: (,) = (1150, 1150)	9.26M (12.5% less)
		L3: (,) = (1150, 400)	2.62M (63.3% less)
	z lower, better		z lower, better

Table 2: RNN-VAE for sentence imputation

Example sentence 1	
from n to n n billion kent cigarettes with the filters were sold _ _ _ _ _	**True**: the company said .
LSTM-VAE	Dimensionality (,) = (300, 256) # Parameters 570.37K
Sample 1: *by toyota moto co. .*	**Sample 2**: *to $ n billion from n billion .*
EINS-VAE	Dimensionality (,) = (300, 256) # Parameters 564.30K (1.06% less)
Sample 1: *through underwriters led by goldman sachs & co. and salomon brother inc .*	**Sample 2**: *at the end of the year's n million shares outstanding .*
Example sentence 2	
but you have to recognize that these events took place _ _ _ _ _	**True**: n years ago .
LSTM-VAE	
Sample 1: *and i'm not going to do .*	**Sample 2**: *in the u . s . .*
EINS-VAE	
Sample 1: *in the next few days .*	**Sample 2**: *on the street .*

*: With style of presentation adopted from Bowman et al. (2016, p. 6, Table 3).

Table 3: BRITS for PhysioNet

Model	AUC	MAE(MRE)	
Cao et al. (2018)			
LSTM-BRITS	0.850	0.281(0.391)	identical as the cell below
our implementation			Dimensionality (,) = (70, 100)
LSTM-BRITS	0.866	0.270(0.382)	# Parameters 136.80K
our proposed model			Dimensionality (,) = (70, 100)
EINS-BRITS	0.850	0.261(0.370)	# Parameters 89.74K (34.4% less)
	z higher, better	z lower, better	

5.5 Run Online for Variable Length — Learning to Learn

Learning to learn was selected to test EINS's ability to run online for a variable length of instances. Andrychowicz et al. (2016) trained optimisee networks with LSTM-optimisers, and found that the optimisees were trained faster than traditional optimisation methods such as SGD (Robbins & Monro, 1951) and ADAM (Kingma & Ba, 2015), and also yielded lower losses in test time. For classifying the CIFAR-10 dataset (Krizhevsky, 2009), Andrychowicz et al. (2016) selected an optimisee network with the architecture of three convolutional layers with max pooling followed by a fully-connected layer. Their LSTM-optimiser was trained to update their optimisee network for 100 steps; and during testing, was set to run freely to optimise the optimisee for 1000 steps. We selected to train the much deeper ResNet-16 (He et al., 2016), and updated weights of the ResNets according to Ravi & Larochelle (2017). This was because we found this method allowed the RNN-optimisers to train the ResNets faster than the method in Andrychowicz et al. (2016). The average result over training 50 ResNet-16s are shown in **Figure 4** (**Appendix E**). Both RNN-optimisers

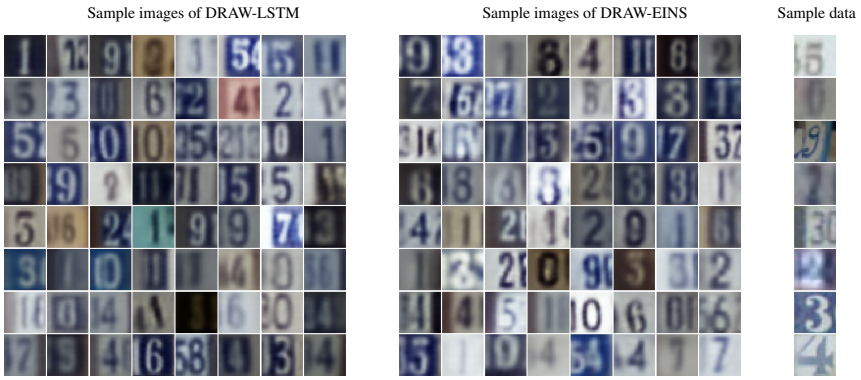


Figure 3: DRAW for generating street-view-house-number images.
The style of presentation adopted from Gregor et al. (2015, p.7, Figure 9).

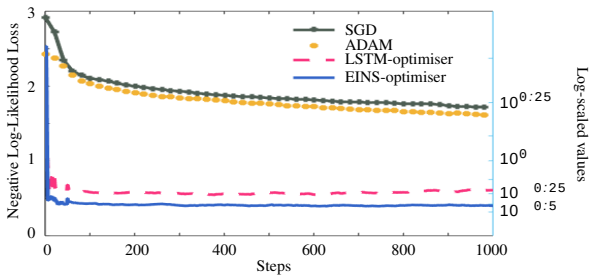


Figure 4: Optimising ResNet-16 on Cifar10

significantly outperformed the hand-crafted optimisers of SGD and ADAM. The EINS-optimiser allowed ResNets to converge to lower losses than the LSTM-optimiser. With 20-input dimension and 20-hidden dimension, the LSTM-optimiser had 3.28 thousand parameters, while the EINS-optimiser had 2.82 thousand, removing 14.0% of parameters.

6 Discussion

This paper modelled cell states of LSTMs and GRUs as dynamic Hopfield networks. We interpreted gated units as time scales of network variables, and highlighted redundant network components. Based on our insights, we presented the novel light-weight RNN of EINS and evaluated it on various tasks for a wide range of practical RNN aspects. Despite having both a simpler design and fewer parameters, this simplification either performed comparably, or better, than the LSTM in each task.

LSTMs and GRUs have previously been studied with different approaches. Empirical studies in text (Karpathy et al., 2015) found LSTM hidden states acting as quotation machines; and in speech (Wu & King, 2016), they were found to correlate to the Mel-Cepsrtal coefficient. Search studies of Jozefowicz et al. (2015) found no particular combinatory sets of variables consistently outperformed LSTMs in all experimental conditions.

Our mathematical analysis aligned with the lesion study of Greff et al. (2017). We found that, as in the GRU, the forget gate f_t is not required given the presence of an input gate i_t. Based on this analysis, we deduced that the LSTM output gate o_t prevent numerical overflow from the deviated values among f_t and $(1 \quad i_t)$. This aligned with the observations in Greff et al. (2017), where they found that the removal of o_t severely impacted LSTM performances across multiple tasks. Furthermore, we provided a theoretical argument that the collaborative effort of LSTM f_t, i_t, and o_t create a more powerful and more versatile cell than that of the GRU.

Dynamical mathematical analysis in RNN have also been previously reported in Tallec & Ollivier (2018). The authors of that paper based their work on Jaeger (2002) and modelled hidden variables of simple recurrent networks as leaky differential networks. In their paper, they interpreted the gated units as autonomous heuristic derivatives. A similar mathematical approach can be found in echo state networks (Jaeger, 2001) and in theoretical physics (Sompolinsky et al., 1988).

REFERENCES

Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. Spin-glass models of neural networks. *Phys. Rev. A*, 32:1007–1018, Aug 1985. doi: 10.1103/PhysRevA.32.1007. URL https://link.aps.org/doi/10.1103/PhysRevA.32.1007.

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3988–3996, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL http://dl.acm.org/citation.cfm?id=3157382.3157543.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *Computing Research Repository (CoRR)*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1002. URL https://www.aclweb.org/anthology/K16-1002.

Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):1145–1159, July 1997. ISSN 0031-3203. doi: 10.1016/S0031-3203(96)00142-2. URL http://dx.doi.org/10.1016/S0031-3203(96)00142-2.

Wei Cao, Dong Wang, Jian Li, Hao Zhou, Yitan Li, and Lei Li. Brits: Bidirectional recurrent imputation for time series. In *Proceedings of The 32nd Neural Information Processing Systems (NeurIPS)*, 2018. URL https://arxiv.org/abs/1805.10572.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018. URL https://www.nature.com/articles/s41598-018-24271-9.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning, December 2014*, 2014. URL https://nyuscholars.nyu.edu/en/publications/empirical-evaluation-of-gated-recurrent-neural-networks-on-\sequen9.

A. Crisanti, D. J. Amit, and H. Gutfreund. Saturation level of the Hopfield model for neural network. *EPL (Europhysics Letters)*, 2:337, August 1986. doi: 10.1209/0295-5075/2/4/012.

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL http://dx.doi.org/10.1162/089976600300015015.

Ian J. Goodfellow. Generative adversarial networks. *Tutorial of Advances in Neural Information Processing Systems (NIPS); and ArXiv*, abs/1701.00160, 2016. URL https://arxiv.org/abs/1701.00160.

Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6082.

Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4601–4609, 2016. URL http://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent\-networks.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Trans. Neural Netw. Learning Syst.*, 28(10):2222–2232, 2017. doi: 10.1109/TNNLS.2016.2582924. URL https://doi.org/10.1109/TNNLS.2016.2582924.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A recurrent neural network for image generation. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1462–1471, 2015. URL http://proceedings.mlr.press/v37/gregor15.html.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In James A. Anderson and Edward Rosenfeld (eds.), *Neurocomputing: Foundations of Research*, pp. 457–464. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104422.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045167.

H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001. URL http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf.

Herbert Jaeger. *Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the "Echo State Network" Approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2342–2350. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045367.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1176.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *Computing Research Repository (CoRR)*, abs/1506.02078, 2015. URL http://arxiv.org/abs/1506.02078.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2741–2749. AAAI Press, 2016. URL http://dl.acm.org/citation.cfm?id=3016100.3016285.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Technical report, University of Toronto 1 (4), 7, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Mantas Lukoševičius and Herbert Jaeger. Survey: Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3(3):127–149, August 2009. ISSN 1574-0137. doi: 10.1016/j.cosrev.2009.03.005. URL http://dx.doi.org/10.1016/j.cosrev.2009.03.005.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=SyyGPP0TZ.

Tomas Mikolov, Martin Karafit, Luks Burget, Jan Cernock, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura (eds.), *INTERSPEECH*, pp. 1045–1048. ISCA, 2010. URL http://dblp.uni-trier.de/db/conf/interspeech/interspeech2010.html#MikolovKBCK10.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Workshop on Deep Learning and Unsupervised Feature Learning of Advances in Neural Information Processing Systems (NIPS)*, 2011. URL https://ai.google/research/pubs/pub37648.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *Conference on Neural Information Processing Systems (NIPS) 2017 workshop*, 2017.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://www.aclweb.org/anthology/D14-1162.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, July 1992. ISSN 0363-0129. doi: 10.1137/0330046. URL http://dx.doi.org/10.1137/0330046.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=rJY0-Kcll.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951. ISSN 00034851. URL http://www.jstor.org/stable/2236626.

M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11): 2673–2681, November 1997. ISSN 1053-587X. doi: 10.1109/78.650093. URL http://dx.doi.org/10.1109/78.650093.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 802–810, 2015. URL http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach\-for-precipitation-nowcasting.

Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*, pp. 245–248. IEEE, 2012.

H. Sompolinsky and Annette Zippelius. Relaxational dynamics of the edwards-anderson model and the mean-field theory of spin-glasses. *Phys. Rev. B*, 25:6860–6875, Jun 1982. doi: 10.1103/PhysRevB.25.6860. URL https://link.aps.org/doi/10.1103/PhysRevB.25.6860.

H. Sompolinsky, A. Crisanti, and H. J. Sommers. Chaos in random neural networks. *Phys. Rev. Lett.*, 61:259–262, Jul 1988. doi: 10.1103/PhysRevLett.61.259. URL https://link.aps.org/doi/10.1103/PhysRevLett.61.259.

Qiang Sun and Yanwei Fu. Stacked self-attention networks for visual question answering. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval, ICMR 2019, Ottawa, ON, Canada, June 10-13, 2019.*, pp. 207–211, 2019. doi: 10.1145/3323873.3325044. URL https://doi.org/10.1145/3323873.3325044.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *INTERSPEECH, Thirteenth annual conference of the international speech communication association*, 2012. URL https://www.isca-speech.org/archive/interspeech_2012/i12_0194.html.

David Sussillo and L F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–57, 09 2009. doi: 10.1016/j.neuron.2009.07.018. URL https://www.cell.com/neuron/supplemental/S0896-6273(09)00547-9.

Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=SJcKhk-Ab.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3630–3638, 2016. URL http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning.

Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1058–1066, 2013. URL http://proceedings.mlr.press/v28/wan13.html.

Zhizheng Wu and Simon King. Investigating gated recurrent networks for speech synthesis. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5140–5144, 2016. URL https://ieeexplore.ieee.org/document/7472657.

Tjalling J. Ypma. Historical development of the newton-raphson method. *SIAM Rev.*, 37(4):531–551, December 1995. ISSN 0036-1445. doi: 10.1137/1037125. URL http://dx.doi.org/10.1137/1037125.

A APPENDIX - LANGUAGE MODELLING

This appendix provides details for the RNN-LMs of **Subsection 5.1**. We will address training details of as well as additional experimental results.

A.1 ARCHITECTURAL DESIGN AND TRAINING SETUP

The RNN-LM of Merity et al. (2018) consists a word embedding layer following 3 layers of LSTM modules and then a softmax layer. For our experiments, we incorporated all regularisation methods detailed in Section 4 of that paper. However, since EINS-LMs do not have recurrent synaptic connections in its gated units, we performed a 10% WeightDrop (Wan et al., 2013) on feedforward synaptic connections for the EINS gated units instead.

We trained our RNN-LMs using a mixed procedure of the SGD optimiser and the ASGD optimiser (Polyak & Juditsky, 1992) as according to Merity et al. (2018). However, we disabled the feature of non-monotonically triggered ASGD (NT-ASGD) as described in Section 3 of that paper. This was done for two reasons. First, there is yet a study to provide theoretical guidelines for when to switch from SGD to ASGD, and this was also noted by Merity et al. (2018). Second, we wanted to ensure that the LSTM-LMs and the EINS-LMs were optimised under the same training environment. For these two reasons, we trained the RNN-LMs with SGD for 30 epochs, and followed with 470 epochs of training on ASGD. That is, with 500 epochs of training in total. While using SGD, the learning rate was set as 30; and as for ASGD, the first 100 epochs were trained with the learning rate of 30, and that the remaining 370 were trained with 50.

A.2 IMPORTANT SOFTWARE ISSUES AND METRIC ISSUES

The PyTorch (Paszke et al., 2017) Github repository of Merity et al. (2018) of
https://github.com/salesforce/awd-lstm-lm
noted that the reported 57.3 perplexity of their paper were achieved under PyTorch version 0.4 on seed 141. The authors of this manuscript found that, although similar performances can be reproduced under PyTorch version 0.4, such is not the case under the latest PyTorch version 1.0 . Under our experimentation, we got a worse perplexity of 61.4 . After huge difficulties in inspecting potential bugs, the first author of this manuscript concluded that, this performance discrepancy should not be attributed to Merity et al. (2018), but instead to the updated random initialisation of PyTorch version 1.0 , and to the update of the "flatten_parameters()" function therein the "RNNCell" module.

The following comments are provided specifically to readers who are less familiar with language modelling. Perplexity is the exponentiated version of the loss value (which Merity et al. (2018) chose the cross entropy loss). That is, though a perplexity of 61.4 is worse than that of 57.3 with an relative increase of 7.16%, their original losses are actually very similar. Perplexity 61.4 has an equivalent loss of 4.1174 while that of perplexity 57.3 is 4.0483. That is, the difference in loss is merely 0.0691. Our reported results in **Subsection 5.1** were conducted with PyTorch version 0.4, and we deliberately chose this version of PyTorch in order to reproduce similar perplexity results to the baseline 57.3. However, while using PyTorch version 1.0 with the exact same code, we yielded 63.89 for LSTM-LM and 63.50 for EINS-LM. That is, we yielded equivalent loss of 4.1572 and 4.1510 for LSTM-LM and EINS-LM, respectively; but they were close to perplexity 57.3, with equivalent loss of 4.0483, reported in the original paper of Merity et al. (2018).

A.3 ADDITIONAL EXPERIMENTS FOR MODELS THAT ARE LESS OVERFIT

As shown in **Subsection 5.1**, each layer of the LSTM-LM consisted more than 7M parameters and it was highly likely that the RNN-LMs (though used as standard practice) were overfitting severely. For this reason, we now present additional experiments with 2-layer RNN-LMs trained for mere 50 epochs. The RNNs had 400-embedding dimension with 1150-hidden dimension; and again, the first 30 epochs were trained with SGD with learning rate 30 and that the later 20 were with ASGD with learning rate 30. Across 50 random initialisations, the shallow LSTM-LMs and shallow EINS-LMs achieved perplexities of 67.76 0.22 and 67.77 0.13 respectively. Thus, with the significant reduction of 63.3% parameters in each layer modules, shallow EINS-LMs achieved performances with no practical significant difference to that of LSTM-LMs.

B APPENDIX - SENTENCE VAE

This appendix provides details for the RNN-VAEs of **Subsection 5.2**. We will address training details of the RNN-VAE as well as additional experimental results.

B.1 ARCHITECTURAL DESIGN

For sentence imputation, we implemented the network architectural design according to Bowman et al. (2016). The network consists of two components, an RNN-VAE and an LM. The RNN-VAE is added as an extensions to the language model, which, for this task, is a GRU-LM.

The purpose of the RNN-VAE is to provide the LM with distributed latent representations of entire sentences. Both the GRU-LM and the RNN-VAE were one-layer deep and had 300-word embedding dimension with 256-hidden dimension.

B.2 TRAINING SETUP

The network was trained on the ADAM optimiser with learning rate 0.001 for 10 epochs. During training, we applied 0.5 embedding dropout and, both the RNN-VAE as well as the GRU-LM, had recurrent synaptic connection dropout (Wan et al., 2013) with rate 0.25.

B.3 ADDITIONAL EXPERIMENTS

The original paper of Bowman et al. (2016) presented negative log-likelihood (NLL) as a measure of the typicality of the entire generated sentence. The lower the NLL, the better the score. However, this metric may not be the most suitable indicator for two reasons. First, the generated outputs are of various lengths. That means, due to the additive and non-negative nature of NLL, longer sequences will usually result in higher scores; longer sentences are not necessarily worse sentences. Second, using NLLs implicitly means that we trust the machine had learnt human-like probabilistic inferences. This cannot be gaureenteed unless a separate in-depth empirical study, like that of Karpathy et al. (2015), is conducted.

In this appendix, we provide two additional indicators to compare EINS-VAE against LSTM-VAE. First, we aggregate the amount of unknown <*unk*> tokens generated during sentence imputation. This provides us a glimpse of the output quality. Second, we provide lengths of the generated sentences. Ideally, we would like to see sentences of moderate lengths or of diverse. Short sentences imply the network is incapable of generating rich context, while extremely long sentences imply malfunctioning networks.

Our additional experimental results can be found in **Table 4** and **Figures 5**, **6**. We considered sentences of two different lengths, one short and one long, for testing RNN-VAEs' abilities in encoding latent information over different lengths of sequences. Furthermore, we sampled sentences with different NLLs to show that subjectively good quality sentences can occur at different NLLs.

Over 20 imputed sequences of the short input, the LM with LSTM-VAE yielded 9 sentences with <*unk*>, whereas the LM with EINS-VAE yielded 2 sentences with <*unk*>. The LM with EINS-VAE was more capable of generating longer sentences than the LM with LSTM-VAE, and had generated sentences with a larger variety of NLLs. However, both the NLLs and the sentence lengths had overlaying interquartile ranges on the side-by-side box plots in the Figures. Thus there are no significant differences between the performances of the two RNN-VAEs.

Over 20 imputed sequences of the long input, the LM with LSTM-VAE yielded 0 sentences with <*unk*>, whereas the LM with EINS-VAE yielded 1 sentence with <*unk*>. Overall, the LM with LSTM-VAE generated much shorter sentences than the LM with EINS-VAE, however both RNN-VAEs generated sentences with similar NLLs. Similar to the generated sentences of the short input, side-by-side box plots in the Figures showed overlaying interquartile ranges, indicating no significant differences in the NLLs nor the sentence lengths.

Table 4: Additional experiments for RNN-VAE on sentence imputation

Short input *there is no asbestos in _ _ _ _*	**True**: our products now .
LSTM-VAE **Sample 1** (**NLL:** 42.78) **Sample 2** (**NLL:** 33.74) **Sample 3** (**NLL:** 25.29) **Sample 4** (**NLL:** 13.89)	 *the next few weeks he says .* *the u . s . and europe .* *n to n .* *<unk> .*
EINS-VAE **Sample 1** (**NLL:** 51.19) **Sample 2** (**NLL:** 32.17) **Sample 3** (**NLL:** 21.33) **Sample 4** (**NLL:** 9.30)	 *the u . s . r . s . r . s . r . a . president and chief executive officer of the u . s . ambassador to the u . s . embassy .* *any case he said .* *new york stock exchange composite trading .* *n .*
Long input *united illuminating is based in new haven conn . and northeast is based _ _ _ _*	**True**: in hartford conn .
LSTM-VAE **Sample 1** (**NLL:** 40.04) **Sample 2** (**NLL:** 33.62) **Sample 3** (**NLL:** 21.26) **Sample 4** (**NLL:** 17.14)	 *on cable networks .* *on the tapes .* *the san francisco area .* *.*
EINS-VAE **Sample 1** (**NLL:** 39.29) **Sample 2** (**NLL:** 34.35) **Sample 3** (**NLL:** 28.33) **Sample 4** (**NLL:** 13.99)	 *in sunnyvale calif . s president and chief executive officer of <unk> inc . and puerto rico .* *on the u . s . attorney general foods corp . a new york state department spokesman said .* *in palo alto calif . A unit of texas air corp .* *on revenue of $ n billion .*

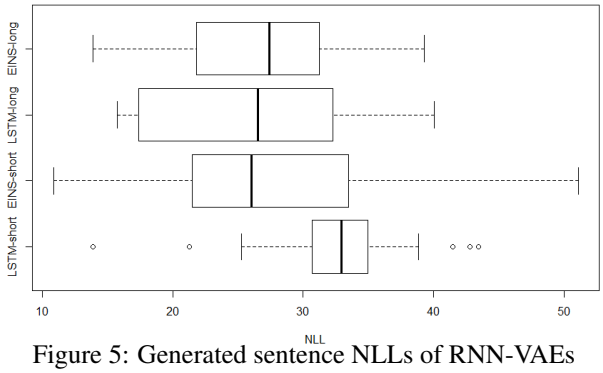


Figure 5: Generated sentence NLLs of RNN-VAEs

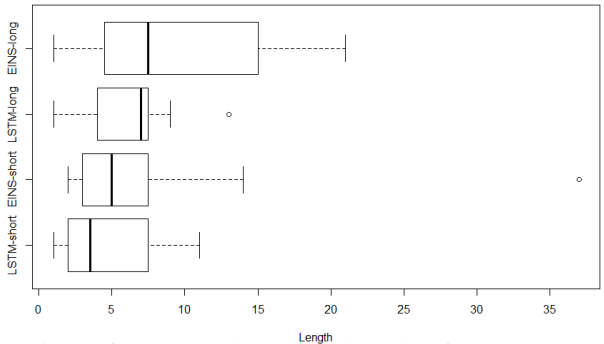


Figure 6: Generated sentence lengths of RNN-VAEs

C APPENDIX - BRTIS

This appendix provides details for the RNN-BRITS of **Subsection 5.3**. We will address training details of the BRTIS as well as additional experimental results.

For medical data imputation-prediction, we implemented the network architectural design according to Cao et al. (2018). The BRITS is an extended architecture of the GRU-D (Che et al., 2018); we will first address the GRU-D before describing the BRITS.

C.1 THE GRU-D ARCHITECTURAL DESIGN

The GRU-D network consists of trainable decays to impute missing values for predictions. The imputation is a two-step procedure. The network first masks existing values, and then applies a trainable decay to modulate impacts of existing values on their missing counterparts. Values of the hidden states are also depreciated in a similar fashion. Iteratively, over the non-equally discretised time interval of $\ _t$, the network utilises the last observed input $x_{t'}$ and emperical mean $\bar{x}$ to

System (C.1): the GRU-D

$$
\begin{aligned}
&\text{mask existing values:} && m_t = \begin{cases} 1; & \text{if } x_t \text{ is observed} \\ 0; & \text{otherwise} \end{cases}, \\
&\text{and accounts time delay:} && \ _t = \begin{cases} t \quad t\ 1 + \ t\ 1; & t > 1; m_{t\ 1} = 0 \\ t \quad t\ 1; & m_{t\ 1} = 1 \\ 0; & t = 1 \end{cases}, \\
&\text{to create scaling factors:} && {}^x_t = \exp\{-\max(0; \mathbf{W}_{x\ t} + b_{x})\}; \\
& && {}^s_t = \exp\{-\max(0; \mathbf{W}_{s\ t} + b_{s})\} \\
&\text{for imputing inputs:} && \hat{x}_t = m_t x_t + (1 \quad m_t)({}^x_t x_{t'} + (1 \quad {}^x_t)\bar{x}), \\
&\text{and the cached cell:} && \hat{s}_t = {}^s_t \quad s_{t\ 1}, \\
&\text{with a set of gated units:} && i_t; r_t = \quad (\mathbf{W}_{fl;Rg}\hat{x}_t + \mathbf{W}_{Rfl;Rg}\hat{s}_{t\ 1} + \mathbf{V}_{fl;Rg}m_t + b_{fl;Rg}), \\
&\text{and an internal input:} && a_t = \tanh(\mathbf{W}_A\hat{x}_t + \mathbf{W}_{RA}(r_t \quad \hat{s}_{t\ 1}) + \mathbf{V}_{fl;Rg}m_t + b_A), \\
&\text{to update the cell:} && s_t = (1 \quad i_t) \quad \hat{s}_{t\ 1} + i_t \quad a_t.
\end{aligned}
$$

Remember, as mentioned in **Section 2**, this paper uses LSTM-terminologies to describe GRUs.

C.2 THE BRITS ARCHITECTURAL DESIGN

The BRITS design is near identical to **System (C.1)**. The only difference is that the internal input of a_t is alternatively modelled by a bidirectional LSTM. When a_t is modelled by a unidirectional LSTM, the resulting variant of BRITS is called RITS. For both the RITS and the BRITS architecture, we used RNN modules with 70-input dimension and 100-hidden dimension.

Aside from the small architectural modification, another small contribution of the BRITS paper to the GRU-D is the reformulation of the loss function. Interested readers can find the reformulation documented in Section 4.3 "Correlated Recurrent Imputation" of page 6 of Cao et al. (2018).

C.3 TRAINING SETUP

The network is trained on the ADAM optimiser with learning rate 0.001 for 150 epochs. For training, we follow the hyperparametric setting used in the official repository of Cao et al. (2018) of https://github.com/caow13/BRITS . During our experiments, we found that the not-very-well-explained weights of loss heavily influenced the quality of the final network. Interested readers can find the these setups provided in https://github.com/caow13/BRITS/hyper_param.py .

C.4 ADDITIONAL EXPERIMENTS

In this appendix, we provide additional experiments on the alternative network of RITS. Results are shown in **Table 5** overleaf.

Table 5: RITS for PhysioNet

Model	AUC	MAE(MRE)	
Cao et al. (2018)			
LSTM-RITS	0.840	0.300(0.419)	
our implementation			Dimensionality (,) = (70, 100)
LSTM-RITS	0.853	0.299(0.414)	# Parameters 68.4K
our proposed model			Dimensionality (,) = (70, 100)
EINS-RITS	0.855	0.286(0.405)	# Parameters 44.9K (34.4% less)
	z higher, better	z lower, better	

By combining the above results and those in **Table 3**, we found that the best result was achieved by LSTM-BRITS at an AUC score of 0.866 with 136.80K parameters, and the second best result was achieved by EINS-RITS at an AUC score of 0.855 with 44.9K parameters. In addition, LSTM-RITS was also able of achieving the competitive result of an AUC score of 0.853 with 68.4K parameters. Thus we conclude that it is unnecessary to go for the bidirectional design for modelling the internal input of a_t.

D APPENDIX - DRAW

This appendix provides details for DRAW of **Subsection 5.4**.

D.1 ARCHITECTURAL DESIGN

We implemented the DRAW architecture for image generation according to Gregor et al. (2015). DRAW employs a VAE network to compress information of images x to prepare for a latent vector of Gaussian information z, which is then fed to a decoder network to infer for pixel information on an image canvas c. At its core, the network can be summarised as

System (D.1): the core of the DRAW architecture

$$\begin{aligned} &\text{prior distribution tuning:} && l^{ENC} = && \text{ENCODER}(x; c)\ , \\ &\text{latent information sampling:} && z && N(z|l^{ENC}), \\ &\text{for canvas pixel inference:} && c = && \text{DECODER}(z). \end{aligned}$$

When an RNN-decoder is employed, we have

$$\begin{aligned} &\text{canvas pixel inference:} && c_t = && c_{t\ 1} + \\ & && && \text{Recurrent-DECODER}([q_{t\ 1}; c_{t\ 1}; z]) \end{aligned}$$

where $q_{t\ 1}$ is the hidden state of the RNN module of the recurrent-decoder, and that $c_0 = \tilde{0}$.

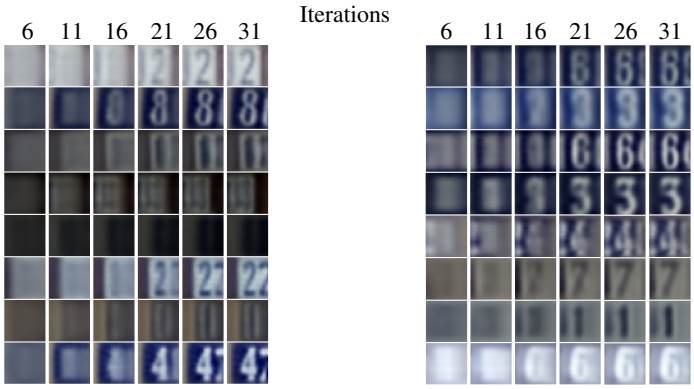
D.2 TRAINING SETUP

The networks were trained on the ADAM optimiser with learning rate 0.001 for 100 epochs. The RNN modules of the decoder network had 100-input dimension with 800-hidden dimension. Latent vectors are of R^{100} and the canvas was reassessed for 32 iterations as according to the paper. An extremely important detail to the RNN-Decoders is that, the forget gate needs to be initialised with a positive bias of $+1$ to allow for successful training, a recommended practice in Gers et al. (2000).

One minor difference in our implementation lies in the format of the dataset. According to Section 4.4 "Street View House Number Generation" on page 7, Gregor et al. (2015) used the pre-processing of Goodfellow et al. (2014). This means that they have used the **Format 1** dataset of http://ufldl.stanford.edu/housenumbers/ (Netzer et al., 2011), and then cropped for relevant number images. In contrast, we used the pre-cropped **Format 2** dataset provided by the same data suppliers. As a consequence, our input images are of pixel sizes 32 32, while theirs are of size 54 54.

D.3 ADDITIONAL EXPERIMENTS

Figures 7 in below are additional experiments for the iterative reassessments of generated images. The NLL metric for the generated images are not provided, for similar reasons previously elaborated in **Subsection B.3**



(a) LSTM **(b)** EINS

Figure 7: Sequentially updating generated images with DRAW

E APPENDIX - LEARNING TO LEARN

This appendix provides details for the RNN-optimisers of **Subsection 5.5**. We will address different variants of updates for the parameters of the optimisee network as well as training setups for the task.

E.1 EXPERIMENTAL SETUP

Three important components are involved in this task — an optimisee network, an RNN-optimiser, and an optimiser for the RNN-optimiser. Our optimisee networks followed the ResNet-16 (He et al., 2016) architecture and were trained to learn to classify images of the CIFAR-10 dataset (Krizhevsky, 2009). The RNN-optimiser was trained by a handcrafted optimiser to learn to update the parameters of an optimisee network.

We followed Andrychowicz et al. (2016) for general practices in the learning to learn task. The RNN-optimisers had 20-input dimension with 20-hidden dimension and were updated for 200 epochs. For each epoch, the RNN-optimiser updated 10 randomly initialised ResNet optimisees. Each optimisee network is set to run for 100 steps. For each step, the optimisees provided information of their gradients as well as their parameters to the RNN-optimisers for parametric updates.

The RNN-optimisers were updated for every 10 steps ran by the optimisee network. This was referred to as *unrolling* in Andrychowicz et al. (2016). The hidden states and the cell states of the RNN-optimisers were removed from the chained differential computational graph after each unrolling. This was done to prevent leakage of previously computed gradient information. For each unrolling step, the RNN-optimisers were updated by the ADAM optimiser with learning rate 0.001.

E.2 UPDATING PARAMETERS OF THE OPTIMISEE NETWORK

The RNN-optimiser of Andrychowicz et al. (2016) is a vanilla single-layer LSTM module, which updates the optimisee weights of $\mathbf{M}(\ )$ as a function of its own parameters of $\ $. That is, given the gradients of the optimisee r_t, the RNN-optimiser infers for a package of update g_t to $\mathbf{M}$, such that

$$\mathbf{M}_t = \mathbf{M}_{t\ 1} + g_t; \text{ where} \tag{15}$$

$$\begin{matrix} g_t \\ q_t \end{matrix} = \text{RNN-optimiser}(r_t; q_{t\ 1}j\ ) \tag{16}$$

and where q_t is the hidden state of the RNN-optimiser.

This method of update differs to handcrafted updates that usually take the form of

$$\mathbf{M} = (1 \qquad)\mathbf{M} \qquad \frac{@E}{@\mathbf{M}} \tag{17}$$

where $\ $ is the learning rate, and that $\ $ is the weight decay. For this reason, we also employed a vanilla single-layer LSTM to train the optimisee via

$$\mathbf{M}_t = (1 \qquad {}_t\ {}_t)\mathbf{M}_{t\ 1} \qquad {}_t r_t; \text{ where} \tag{18}$$

$$\begin{matrix} \\ {}_t \\ {}_t \\ q_t \end{matrix} = \text{RNN-optimiser}(r_t; q_{t\ 1}j\ ): \tag{19}$$

However, we found that this method did not yield ResNet-optimisees with lower losses, nor did it allow the RNN-optimisers to train faster.

As a follow-up experiment, we implemented the RNN-optimiser described in Ravi & Larochelle (2017). Their RNN-optimiser consists of a single layer of vanilla LSTM module with an extra meta-update layer. Layer Normalisation (Ba et al., 2016) is applied to all synaptic connections within the LSTM module. The meta-update layer formulates the parametric update of the optimisee as the propagation of an LSTM cell, where

$$\mathbf{M}_t = \mathrm{f}_t\mathbf{M}_{t\ 1} + \mathrm{i}_t g_t \tag{20}$$

such that f_t, i_t, and g_t are recurrent units that takes the hidden state of the first vanila LSTM layer as input. During our experiments, we found this method updated the RNN-optimiser the fastest, and also yielded ResNet-optimisees with the least loss.