Robust and Scalable Differentiable Neural Computer for Question Answering

Jörg Franke, Jan Niehues, Alex Waibel
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology, Germany
joerg.franke@student.kit.edu
{jan.niehues,alex.waibel}@kit.edu

Abstract

Deep learning models are often not easily adaptable to new tasks and require task-specific adjustments. The differentiable neural computer (DNC), a memory-augmented neural network, is designed as a general problem solver which can be used in a wide range of tasks. But in reality, it is hard to apply this model to new tasks. We analyze the DNC and identify possible improvements within the application of question answering. This motivates a more robust and scalable DNC (rsDNC). The objective precondition is to keep the general character of this model intact while making its application more reliable and speeding up its required training time. The rsDNC is distinguished by a more robust training, a slim memory unit and a bidirectional architecture. We not only achieve new state-of-the-art performance on the bAbI task, but also minimize the performance variance between different initializations. Furthermore, we demonstrate the simplified applicability of the rsDNC to new tasks with passable results on the CNN RC task without adaptions.

1 Introduction

In contrast to traditional statistical models, which often require a large amount of human effort on feature engineering and task-specific adjustments, a promise of deep learning is that little task-specific knowledge and minimal adaption is required to achieve state-of-the-art performance on different tasks. But in reality, many deep learning solutions have to be adapted to a specific task to achieve good performance.

However, there are more universal approaches for example the differentiable neural computer (DNC). It is introduced by Graves et al. (2016) as a general artificial neural network (ANN) model with an external memory "to solve complex, structured tasks". It can be seen as a generic memory-augmentation framework. Unlike a vanilla ANN, it separates computation and memorization with a computational controller and a memory unit, which are independently modifiable. This allows a more accurate model design. Due to its fully differentiable design, it can be learned in a supervised fashion.

The original paper shows applications on the bAbI question answering (QA) task, graph experiments and a reinforcement learning block puzzle solver (Graves et al., 2016). But when applying this model to new QA tasks, no satisfying results are achieved. The issues of QA are the huge vocabulary, the length of the contexts and the required model complexity to find the correct answer.

In this work, we analyze the DNC in QA tasks and identify four main challenges: 1. High memory consumption makes it hard to train large models efficiently. 2. The large variance in training performance within different initializations. 3. A slow and unstable convergence requires long varying training times. 4. The unidirectional architecture makes it hard to handle variable question appearance.

This work addresses these issues while keeping the general character of the model intact. We extend the DNC to be more robust and scalable (rsDNC) with the following contributions:

1. A robust training with a strong focus on an early memory usage and normalization.
2. The usage of a slim, memory efficient, content-based memory unit for QA tasks.

Proceedings of the Workshop on Machine Reading for Question Answering, pages 47–59
Melbourne, Australia, July 19, 2018. ©2018 Association for Computational Linguistics

3. A bidirectional DNC which allows a richer encoding of the input sequences.

The rsDNC is evaluated on two datasets. On the synthetic bAbI task (Weston et al., 2016), we show performance improvements by 80% compared to the DNC. These are new state-of-the-art results within multiple runs in a joint training. We also decrease the variance by up to 90% between different random initializations. Additionally, with training-data augmentation on one task, our model solves all tasks and provides the best-recorded results to the best of our knowledge. On the CNN RC task (Hermann et al., 2015), we show the adaptability of the rsDNC and achieve passable results without task-specific adaption.

2 Related Work

This section considers the related work regarding the two used datasets.

Related to bAbI task Rae et al. (2016) provides technical enhancements with the introduction of the sparse DNC (SDNC) with sparse read and write operations. They allow to modifying only a sparse subset of interesting memory locations instead of manipulating all. This renders the memory consumption independent of memory size. A different approach provides the dynamic memory network (DMN) and its successor the DMN+ (Kumar et al., 2016; Xiong et al., 2016). They store sentence representations in an episodic memory and use attention to find the correct answer. The relation memory network (RMN) embeds sentences into a memory object and applies multiple times attention to find the answer (Yang et al., 2018). In contrast to our model, the DMN+ and the RMN are optimized for QA tasks, uses sentence representation and require a dedicated question. The recurrent entity network (EntNet) “can be viewed as a set of separate recurrent models whose hidden states store the memory slots” (Henaff et al., 2017). The memory slots or locations consist of a key vector and a content vector and have their own gated RNN as a controller. In contrast, our model has one memory matrix with no distinction between key and content.

Related to CNN RC task Hermann et al. (2015) introduce the Deep LSTM Reader and the Attentive Reader which build a document representation by direct attention. Chen et al. (2016) introduce the Stanford Attentive Reader which enhances the attentive reader and adds a bilinear term to compute the attention between document and query. The Attention-Sum (AS) Reader from Kadlec et al. (2016) also uses separate encoding for the document and the query. Its successor, the Attention-over-Attention (AoA) Reader, applies a two-way attention mechanism to find the answer (Cui et al., 2017). The ReasoNet uses iterative reasoning over a hidden representation of the document (Shen et al., 2017). The Gated-Attention (GA) Reader from Dhingra et al. (2017) uses multiple hops over the document to build an attention over the candidates to select the answer token. These models are all conceptually adapted to the QA tasks they solve. In contrast, our solution is more versatile due to a more flexible and universal design. The different parts of the DNC can be exchanged or adjusted independently which allows simpler handling of new tasks. The versatility is shown in the original paper with three different tasks (Graves et al., 2016).

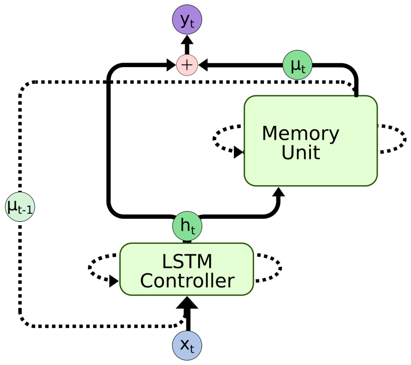


Figure 1: System overview of the DNC. The dotted lines illustrate recurrent connections.

3 Differentiable Neural Computer

System overview The DNC model consists of two main parts, a controller and a memory unit (MU), see Figure 1. The controller is either a fully-connected ANN or a RNN. It receives at each time step a concatenation of the input signal $\boldsymbol{x}_t \in \mathbb{R}^X$ and the MU output from the last time step $\boldsymbol{\mu}_{t-1} \in \mathbb{R}^P$. C is the controller output size and P the MU output size. The controller can be considered as a closed function with a set of trainable weight parameters θ_c:

$$\boldsymbol{h}_t = \text{Controller}([\boldsymbol{x}_t, \boldsymbol{\mu}_{t-1}], \theta_c) \cdot \tag{1}$$

The purpose of the controller is to manage the

MU and additionally to help building the output signal via a weighted bypass connection. The MU receives the controller output $\boldsymbol{h}_t$ as input and contains a set of trainable weight parameters θ_μ as well:

$$\boldsymbol{\mu}_t = \text{MemoryUnit}(\boldsymbol{h}_t, \theta_\mu) \cdot \tag{2}$$

The output signal of the whole DNC $\boldsymbol{y}_t \in \mathbb{R}^Y$ is a sum of the weighted controller output and the weighted MU output.

$$\boldsymbol{y}_t = \mathbf{W}_h \boldsymbol{h}_t + \mathbf{W}_\mu \boldsymbol{\mu}_t + \boldsymbol{b}_t \cdot \tag{3}$$

Let Y be the size of the target vector $\boldsymbol{z}_t \in \mathbb{R}^Y$. $\mathbf{W}_h \in \mathbb{R}^{Y \times C}$, $\mathbf{W}_\mu \in \mathbb{R}^{Y \times P}$ and $\boldsymbol{b}_t \in \mathbb{R}^Y$ are trainable weights.

The hyperparameters of the controller network are the number of layers l and the nodes per layer C_l. The hyperparameters of the MU are the width W of the internal memory matrix and the number of locations N. There can be multiple read heads R which extract information from the memory matrix. The MU output size is then $P = R \times W$.

Memory Unit The MU contains a memory matrix $M_t \in \mathbb{R}^{N \times W}$ which stores information in form of row-wise vectors. For controlling the memory matrix, the input signal from the controller is weighted and divided into different control signals including gates, vectors or keys. They allow to write and read from the memory matrix.

For writing to or erasing the memory matrix two mechanisms are available to find the location where to manipulate M_t: By the least used memory location or content-based addressing. The least used memory location is used to find free space for new information. A *free gate* determines the freeing of used memory to allow forgetting content. Content-based addressing finds the location in M_t which has the lowest cosine-similarity to a given key $k_t \in \mathbb{R}^W$. An *allocation gate* determines which mechanism is used and a *write gate* determines the intensity of writing or erasing new information to M_t.

Two mechanisms are also available for reading from M_t: A content-based addressing similar to the writing and a temporal linkage mechanism. The temporal linkage mechanism contains a linkage matrix $LM_t \in \mathbb{R}^{N \times N}$ which stores the transition of the current write location and the previous location to repeat the sequences in forward direction or with a transposed linkage matrix in backward direction. A *read mode* determines which mode is used. Furthermore, a DNC may have multiple (R) *read heads* with such reading mechanisms. The MU's output is a concatenation of the read vectors from all *read heads* $\boldsymbol{\mu}_t \in \mathbb{R}^{W \times R}$. The DNC is described in detail in the appendix of the original paper (Graves et al., 2016).

4 Analysis of the DNC

This section describes the analysis of the DNC with regard to question answering.

Training progress We trained multiple models with same parameters but different initializations on different bAbI tasks. While some models converges, other do not. This depends only on the initialization. If a model does not converge but learns to solve the task, it is overfitting the training data.

We further analyze the influence of the memory unit. The output of the DNC is a weighted sum of the controller output and the MU output. The influence is determinable by exclusively using the memory output and comparing it to the performance when using both. While training, we found a strong correlation between a high usage of the MU and a good performance of the model. When a model does not converges, the memory unit has nearly no influence on finding the correct answer.

We assume that the direct training signal over the bypass connection between controller and output leads to a fast success in learning to use the controller exclusively. This could prevent learning to use the MU. Additionally, the MU output in the beginning is noisy which could guide the controller to ignore the MU. This could be influenced by the initialization.

Functionality The functionality of the DNC can be analyzed by observing the gates which determine the mechanisms to write or read content. We find the DNC to exclusively use content-based reading when answering a question. The reading from the the temporal linkage mechanism (in forward and backward direction) has only little impact on finding the answer. Furthermore, the usage of the different gates for freeing memory space (*free gate*), determine the write mechanism (*allocate gate*) or write intensity (*write gate*) does not seem very meaningful. Figure 3a in Appendix A shows a plot of the DNC gates during a bAbI task 1 sample.

Computing resources In comparison to a LSTM, the DNC requires over two times more memory during training. It mainly depends on the memory size of the DNC and the sequence length. A closer analysis exhibits that the linkage matrix and the temporal memory linkage mechanism are the main drivers behind memory consumption. This biggest influence is the linkage matrix of size $N \times N$. The temporal memory linkage mechanism of the DNC with the configuration of the original bAbI experiment accounts for 50% of the total memory consumption. In our implementation, the training time per sample is four times higher in comparison to a TensorFlow LSTM with the aforementioned configuration.

5 Robust and Scalable DNC

This section describes in detail the two enhancements of the robust and scalable DNC (rsDNC): An efficient memory unit and a more robust training. Additionally, it introduces a bidirectional DNC architecture.

5.1 Efficient Memory Unit for QA

For an efficient usage of the model and the scalability to large-scale tasks, less memory consumption is relevant. This allows dealing with larger sequences and bigger batch sizes for faster iterations needed e.g. for feasible hyperparameter tuning. The memory consumption of the DNC is very high compared to other recurrent models. As the analysis in Section 4 shows, the main cause for memory consumption is the temporal memory linkage mechanism. But the analysis also shows that the DNC does not use them in the bAbI task. This makes sense since restoring sequences is barely necessary for finding the correct answer in a QA task.

To allow a more efficient usage in QA we used a slim, only content-based memory unit (CBMU). The CBMU has the same features as the DNC's MU but without the temporal memory linkage mechanism. Consequently, the linkage matrix and all related components are removed. The read weightings are only based on the content-based addressing. The write head, memory update and the actual memory reading stay the same.

The drop in memory consumption depends on the hyperparameters and the sequence length but in this paper, it is between 30% and 70%. The computation time is also reduced by 10% to 50%.

5.2 Robust DNC Training

A more robust training improves the large variance in the training progress within different initializations as well as the slow and unstable convergence behaviour. This makes the training repeatable and reduces the training time. To achieve this, we apply normalization to the DNC and present Bypass Dropout to force a faster memory usage.

DNC Normalization Analysis of the DNC shows a high variance in performance between different runs. We approach this issue with a normalization technique to enforce a robust and smooth convergence behaviour. In recent years especially layer normalization (LN) shows performance improvements in ANNs on several applications (Ba et al., 2016; Klambauer et al., 2017). In the DNC setup, it can be applied to the controller as well as the MU. Let μ_t be the mean of a vector x_t and σ_t^2 the variance of it. Then the normalization

$$\mathrm{LN}(\boldsymbol{x}_t) = \boldsymbol{g} \circ \frac{\boldsymbol{x}_t - \mu_t}{\sqrt{\sigma_t^2 + \epsilon}} + \boldsymbol{b} \tag{4}$$

is applied before each activation function. The recurrent and current input signals are computed jointly. Each LN has trainable variables, called bias $\boldsymbol{b}$ and gain $\boldsymbol{g}$ for scaling the normalization.

In the MU we applied LN to the gates, the vectors and the keys separately but this gave no performance increase compared to a joint normalization of all signals. Thus, we apply it after the weighting of the controller output h_t

$$\xi_t = \mathrm{LN}(\boldsymbol{h}_t W_\xi) \tag{5}$$

and before the vector ξ_t is split into the different control signals. It is applied during training and test times. The drawbacks are increased computation time and memory need due to momentum calculation and the additional gain and bias variables.

Bypass Dropout The analysis in Section 4 shows that convergence behaviour of the DNC depends on the usage of the MU. If the MU strongly impacts the system output, the model achieves a good performance. This insight demands the explicit force of MU usage during training to reach convergence faster and obtain better performance.

To force the MU's influence, the impact of the controller to the output via the bypass connection can be limited. This can be achieved by reducing the connectivity between the controller and the output. A reduction of the connectivity by weighting or lower the feature space would be permanent and not adjustable. By using dropout in the bypass connection between controller and output is a controllable lowering of the connectivity possible. Dropout is a regularization technique introduced 2014 in Srivastava et al. (2014) to prevent ANNs from overfitting. In our use-case, dropout allows a adjustable reduction of the bypass connectivity. It can be controlled with the keep probability and is only used during training. His helps to enforce a faster usage of the MU. We call this technique Bypass Dropout (BD).

The dropout probability p regularizes the signal flow exactly without permanent declining the controller's functionality. Bypass Dropout is applied to the DNC setup by multiplying a Bernoulli vector $\boldsymbol{r}_t$ to the bypass connection:

$$\begin{aligned} \boldsymbol{r}_t &\sim \text{Bernoulli}(p) \\ \boldsymbol{y}_t &= \mathbf{W}_h(\boldsymbol{h}_t \circ \boldsymbol{r}_t) + \mathbf{W}_\mu \boldsymbol{\mu}_t \cdot \end{aligned} \tag{6}$$

5.3 Bidirectional DNC

The unidirectional architecture of the DNC makes it hard to handle variable input where for example the question appears in the middle of a text and the full text is relevant. It also prevents a rich information extraction in forward and backward direction in any sequential task.

Therefore, this work introduces a bidirectional setup to provide complete availability of the input sequence to the model. No more distinction between context and question is necessary. The model is able to use information from a later point in the input sequence to determine what to store. In the bidirectional DNC (BDNC) and rsDNC (BrsDNC) an additional RNN in backward linkage provides a sequential comprehension in both directions.

Due to the recurrent connection from MU to controller, an encapsulated bidirectional controller is not possible, since such a model is not unfoldable in time. The solution presented in this work applies an independent backward-directed recurrent controller which provides an additional input signal to the MU and the output layer, illustrated in Figure 2. Hence, the BrsDNC has two controllers, a forward controller and a backward controller

$$\begin{aligned} \boldsymbol{h}_t^{fw} &= \text{ForwardController}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}^{fw}, \boldsymbol{\mu}_{t-1}], \theta_{c^{fw}}) \\ \boldsymbol{h}_t^{bw} &= \text{BackwardController}([\boldsymbol{x}_t, \boldsymbol{h}_{t+1}^{bw}], \theta_{c^{bw}}) \end{aligned} \tag{7}$$

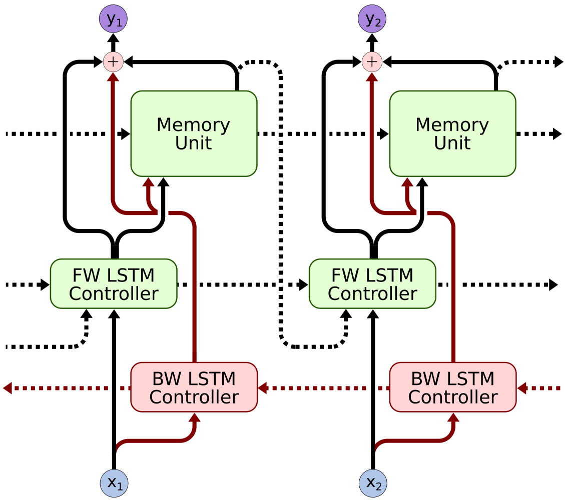


Figure 2: The bidirectional DNC architecture unfolded in time.

with independent weights θ and recurrent connections. The MU receives a concatenation of the two controller outputs

$$\boldsymbol{\mu}_t = \text{MemoryUnit}([\boldsymbol{h}_t^{fw}, \boldsymbol{h}_t^{bw}], \theta_{mu}) \cdot \tag{8}$$

The output of the BrsDNC system is the sum of the weighted memory output, the weighted backward controller output and the weighted forward controller output:

$$\boldsymbol{y}_t = \mathbf{W}_\mu \boldsymbol{\mu}_t + \mathbf{W}_{fwh} \boldsymbol{h}_t^{fw} + \mathbf{W}_{bwh} \boldsymbol{h}_t^{bw} \cdot \tag{9}$$

This architecture allows first an independent unfolding of the backward controller and second an unfolding of the forward controller and MU.

6 Experiments

The rsDNC and the BrsDNC are evaluated on two datasets, the bAbI task and the CNN RC task.[1]

6.1 bAbI Task

Task description The bAbI task is a set of 20 synthetic QA tasks for testing text understanding and logical reasoning (Weston et al., 2016). Each task has several stories and each story contains a context, one or more questions and the correct answers. There are different sets available but this work only uses the 10k set in English. Each story is pre-processed by removing numbers, transforming all words to lower case and splitting the sequences into word tokens. The whole set contains

[1] All experiments are implemented in TensorFlow (Abadi et al., 2015) and are trained on a single K80 GPU within two to nine days. The source code is available at https://github.com/joergfranke/ADNC.

156 unique words and three symbols: '?', '!', and '-'. The '-' symbol in an input sequence symbolizes that an answer is requested. From each task 10% of the samples are used as a validation set. The test set is separate and has 1k questions per task. Inconsistencies in the training set, like a question asked twice, are deleted. The loss metric for the bAbI task is the word error rate (WER), the fraction of incorrectly answered words to all requested words.

Task 16 augmentation Many related models struggle with task 16 in the bAbI dataset and only achieve a WER of roughly 50% (Graves et al., 2016; Sukhbaatar et al., 2015; Xiong et al., 2016). The task is to find a colour given a name and name-animal-colour constellations. In the dataset are four different colours, four animals and five names but not equally distributed in the samples. In many samples one or more colours or animals are present multiple times, see training sample of task 16:

Greg is a lion. Julius is a swan. Julius is yellow. Greg is yellow. Brian is a swan. Bernhard is a frog. Brian is white. Lily is a frog. Bernhard is yellow. What colour is Lily? yellow

There are 9k samples in the training set, but in 4181 cases the correct word appears only once in the context. In all other cases, the correct word appears two, three or four times in the context. If the model learns only to count the colour words and to answer the colour that appears multiple times, then it is correct in 3449 cases and in 1370 additional cases with a $50/50$ probability. In 4181 cases it is able to guess and has a $1/4$ probability to be correct. This leads to a mean probability over 50% for a correct answer by guessing and counting words. This is a strong local optimum and makes it hard to find a better solution strategy.

We provide an augmentation of task 16 so that each sample contains different colours and different animals. We pretrain a model on augmented task 16 plus all other tasks and then fine-tune it without any augmentation to learn the original distribution of the data. This model is marked as *+aug16*. The augmentation also could be helpful in related models but it is only evaluated with this work.

Training details Different enhanced DNC models are evaluated on the bAbI task as well as the rsDNC with Bypass Dropout, DNC Normalization and the CBMU, the bidirectional rsDNC (BrsDNC) and the BrsDNC with augmented task 16 (BrsDNC +aug16). All tasks are trained jointly on all bAbI tasks without any additional information.

For a direct comparison, the hyperparameters are based on the original paper (Graves et al., 2016). The unidirectional controller has one LSTM layer and 256 hidden units and the bidirectional has 172 hidden units in each direction. Thus both models have about 891k parameters. The MU of all models has 192 locations, a width of 64 and 4 read heads. All models have a dropout probability of 10%.

Each word of the bAbI task is encoded as a one-hot vector with the size of 159. An additional sequence mask is used to generate only training signals when an answer is requested, so the outputs of all other time steps are ignored. The output is a vector of size 159 and normalized with a softmax function. For training, the cross-entropy loss between the prediction vector and the target one-hot vector is minimized.

Each training uses mini-batches with a batch size of 32. Different-length sequences are padded. The maximum sequence length during training is limited to 800 words. The optimizer is found using a grid search, the underlined options are used: optimization algorithm [SGD, Adam, RMSprop] with a learning rate of $[1, \underline{3}, 6] * 10^{-[3,4,\underline{5}]}$ and a momentum of 0.9 (Kingma and Ba, 2015; Tieleman and Hinton, 2012). Each model runs five experiments with different initializations.

Model	Mean WER
DNC*	16.7 ± 7.6
EntNet*	9.7 ± 2.6
SDNC*	6.4 ± 2.5
DNC +CBMU	17.6 ± 1.6
DNC +CBMU+Norm	13.6 ± 3.3
DNC +CBMU+BD	9.7 ± 2.9
rsDNC (CBMU+Norm+BD)	6.3 ± 2.7
BrsDNC (bidirectional rsDNC)	3.2 ± 0.5
BrsDNC +aug16	0.4 ± 0.0

Table 1: Mean word error rate of different models and our enhancements to the DNC in the bAbI task. All models are trained jointly on all 20 bAbI tasks at once without information about the actual task. *Result from (Graves et al., 2016; Henaff et al., 2017; Rae et al., 2016)

Results Table 1 shows the mean word error rate and variance on the test set from all tasks and all models of this work in comparison to the original paper (DNC) (Graves et al., 2016) and the EntNet, the best model with reported mean results as far as we know (Henaff et al., 2017).

It also shows the performance impact of the different enhancements: DNC Normalization, Bypass Dropout (BD) and the content-based memory unit (CBMU). The use of CBMU leads to a drop in performance but it also reduces the required memory consumption and lowers the standard deviation within different initializations. Both DNC Normalization and BD show a clear performance gain and together they achieve the best performance.

Experiments on a single bAbI task with stepwise validation provides a closer insight on these results. The usage of the CBMU affects the convergence behaviour only marginally. With use of DNC normalization, the time until convergence is reduced by more than 50%. Additional use of BD lowers the mean convergence time further and increases the reliability of convergence. A bidirectional controller speeds up convergence even more and allowing improvements of up to 75% less required iterations. Figure 4 in Appendix B shows the impacts in bAbI task 1.

In Appendix C, Table 3 shows the mean word error rate of all tasks and Table 4 contains the results of the best runs with additional comparison to the RMN (Yang et al., 2018), the best-reported model in literature as far as we know.

The rsDNC outperforms the DNC from the original paper as well as the jointly trained EntNet. This shows the impact of the normalization and the BD which improves the model performance even without the temporal memory linkage mechanism. This could imply that this mechanism is not important for QA tasks. It also leads to a significant drop in variance which demonstrates that our models are more robust.

The additional model complexity through the bidirectional design shows clear improvements without requiring more parameters. It outperforms the DNC in terms of mean error as well as variance. The lower variance indicates a very robust model for different random initializations. But through the bidirectional controller, the model has access to information about the question while reading the context similar to the RMN or DMN+. Particularly the performance on task 3, 17 and 19 reaches a new quality in the mean results without any failed tasks.

The BrsDNC with augmentation of task 16 leads to the best-reported overall results as far as we know. The modifications allow to learn the task correctly and solves it completely. Even when ignoring the task 16 in the results, the performance of this model is better than previous results. This indicates that the corrected task has a positive cross-effect on the learning of the other tasks.

But the advancements of the rsDNC not only improve the overall performance. The functionality of the MU, as analyzed in Section 4, shows more meaningful control signals, see Figure 3b in appendix A. The *allocate gate* chooses the writing by the least used location when it writes new information and uses content-based writing when it adds supplementary information to existing entries. The *write gate* prevents writing of uninformative words like articles or prepositions. The *free gate* prevents freeing memory when adding new information to not overwrite stored information. The memory influence is maximal when a answer is requested. All of this shows that our advancements probably increase the fundamental functionality of the DNC.

The training time of our implementation is 40 min per epoch for the rsDNC and 45 min per epoch for the BrsDNC. An epoch of training a DNC takes double the time due to smaller batch sizes and additional computation for the temporal linkage mechanism.

6.2 CNN Reading Comprehension Task

Task description The CNN reading comprehension (RC) task is introduced by Hermann et al. (2015) and is based on crawling the online news articles published on the CNN website from April 2007 to April 2015. The dataset contains samples with news article as context and short article summaries as query statements. The articles and query statements are anonymized by replacing all name entities with tokens. This is required since the articles contain name entities, for example celebrities, and the task aims to exploit general relationships between anonymized entities rather than common knowledge. Each article is the source for four queries on average and each query is the finding of a token which is replaced by a placeholder in the query statement.

The task is divided into training, validation and test sets on a monthly base. The training set contains 90,266 articles and 380,298 samples with a mean length of 775 words. The used preprocessed dataset has all words in lower case and reduced inconsistencies. The vocabulary size of 118,497 is limited to 50k by replacing rarely-seen words with a 'zero' token. The task contains 408 name entity tokens. The loss metric for the CNN is the accuracy, the fraction of all samples with correct words to the number of samples in total.

Training details The query and the article are concatenated (the query first) as an input sequence. The sequence is fed to the model word by word and each word is represented as a word vector with a size of 100 and GloVe initialization (Pennington et al., 2014). The word vector is optimized during training. The target is the correct word represented as the index of a one-hot vector with the size of all name entity tokens. A candidate mask is created which masks out all name entity tokens which are not present in the sample. The last model output predicts the word and is normalized with a softmax function. The cross-entropy loss between the prediction output vector and the target one-hot vector is minimized.

We use a batch size of 32; different-length sequences are padded starting from the beginning. Two models are evaluated on this task, rsDNC and BrsDNC. All hyperparameters are chosen inspired by related work. The controller is a LSTM with one hidden layer and a layer size of 512 in the unidirectional setup and 384 each in the bidirectional setup. Both models have a memory matrix with 256 locations, a width of 128 and four read heads. Bypass Dropout is applied with a dropout rate of 10%. The maximum sequence length during training is limited to 1400 words. The model is optimized with RMSprop with fixed learning rate of 3e-05 and momentum of 0.9.

Results The results on the CNN RC task are shown in Table 2. Both models achieve passable results compared to task-specific state-of-the-art models. But when focusing on the fact that our model is trained without any adaption to the task like sentence representations or word-windows, without hyperparameter tuning or any other optimization it is a satisfactory result. The bidirectional model outperforms the unidirectional rsDNC, but only on the test set. The training time of our implementation is 14/16 hours per epoch on the rsDNC/BrsDNC. A DNC/BDNC requires up to 35/41h per epoch. The overall training time is 7 epochs on average. The memory savings of the CBMU result in 4 training days instead of 12. This shows the model's adaptability to perform other tasks without any task-specific adjustment and its scalability due to a drastic reduction of training time.

Model	valid	test
Deep LSTM Reader	55.0	57.0
Attentive Reader	61.6	63.0
rsDNC	67.5	69.0
AS Reader	68.6	69.5
BrsDNC	67.1	69.8
Stanford AR	72.2	72.4
AoA Reader	73.1	74.4
ReasoNet	72.9	74.7
GA Reader	77.9	77.9

Table 2: The validation and test accuracy (%) of the rsDNC/BrsDNC and others on CNN dataset.

7 Conclusion

This work introduces the robust and scalable DNC (rsDNC), an improved DNC applied in QA tasks. It achieves state-of-the-art results on the bAbI task and passable results on the CNN RC task without any task-specific model adaption.

We show that the rsDNC is more robust and usable in contrast to the vanilla DNC due to a faster and more stable training behaviour as well as less dependence on the initialization. This is caused by the introduced Bypass Dropout and DNC Normalization. These advancements make the DNC easier applicable to other tasks. Furthermore, we demonstrate the scalability of the rsDNC to a large-scale QA task by introducing a contend-based memory unit. It lowers memory consumption and training time accompanied by only a minimal loss of performance. A novel bidirectional architecture improves the contextual accessibility and allows questions at every position in the input sequence. Additionally, we provide a training augmentation for one of the bAbI tasks to solve all of them.

In further work, the rsDNC architecture could be extended to a sequence-to-sequence model which enables the usage in other NLP tasks.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. https://www.tensorflow.org/.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *CoRR* abs/1607.06450.

Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 2358–2367.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. 2017. Attention-over-attention neural networks for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 593–602.

Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. 2017. Gated-attention readers for text comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 1832–1846.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538(7626):471–476.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2017. Tracking the world state with recurrent entity networks. *ICLR* .

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*. pages 1693–1701.

Rudolf Kadlec, Martin Schmid, Ondřej Bajgar, and Jan Kleindienst. 2016. Text understanding with the attention sum reader network. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 908–918.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*. pages 972–981.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*. pages 1378–1387.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Tim Lillicrap. 2016. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances In Neural Information Processing Systems*. pages 3621–3629.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pages 1047–1055.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* 15(1):1929–1958.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*. pages 2440–2448.

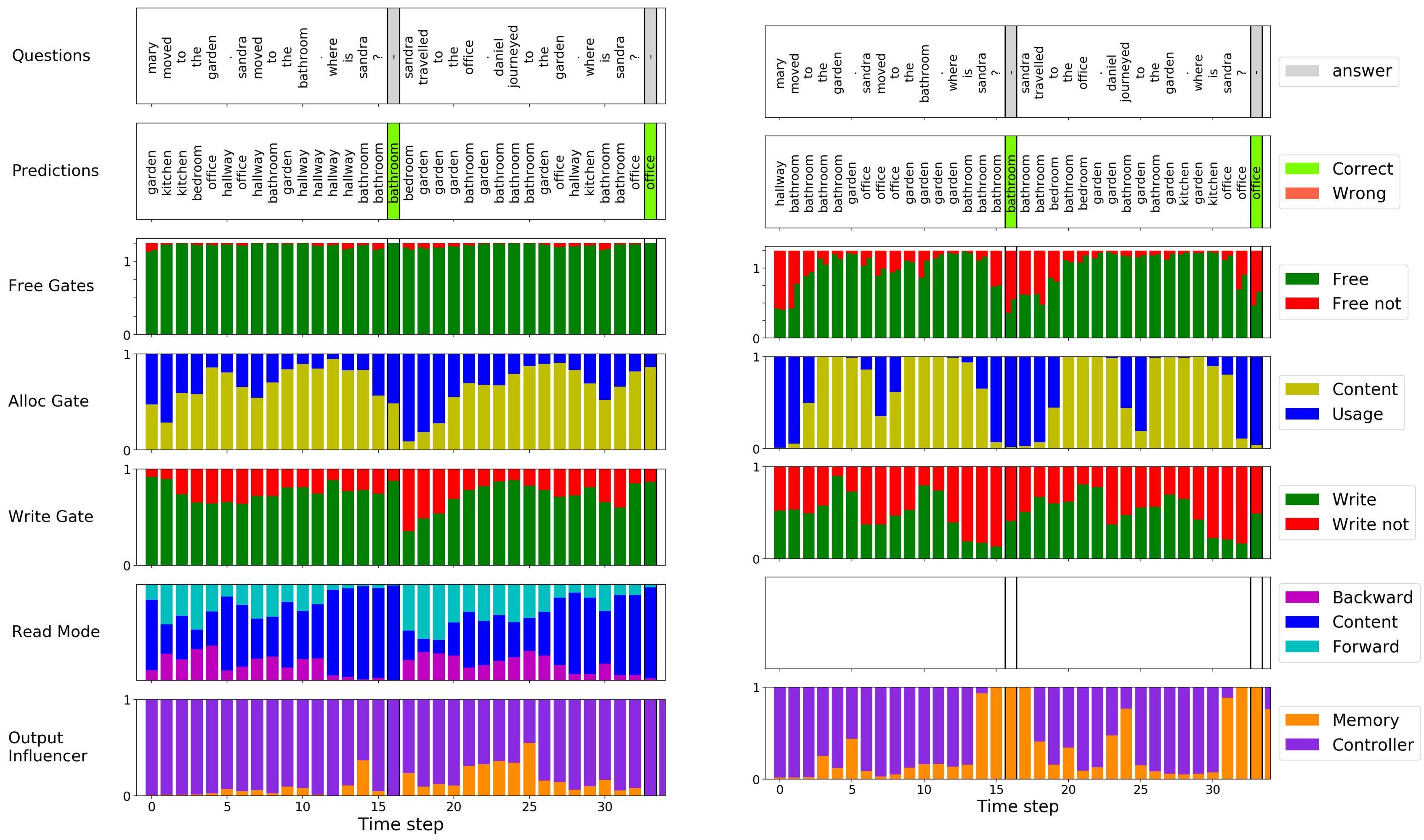
Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2):26–31.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. *ICLR* .

Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *International Conference on Machine Learning*. pages 2397–2406.

Hyochang Yang, Sungzoon Cho, et al. 2018. Finding remo (related memory object): A simple neural architecture for text based reasoning. *arXiv preprint arXiv:1801.08459* .

A DNC functionality on the bAbI task 1



(a) Functionality DNC

(b) Functionality rsDNC

Figure 3: The functionality of the DNC's (a) and the rsDNC's (b) gates and the influences in a bAbI task 1 sample. The activity of the gates and read modes are the actual gating values between 0 and 1. Influences are normalized between 0% and 100%. The gate usage of the rsDNC (b) seems to be more meaningful and the memory influence is higher when answering a question.

B Convergence behaviour on the bAbI task 1

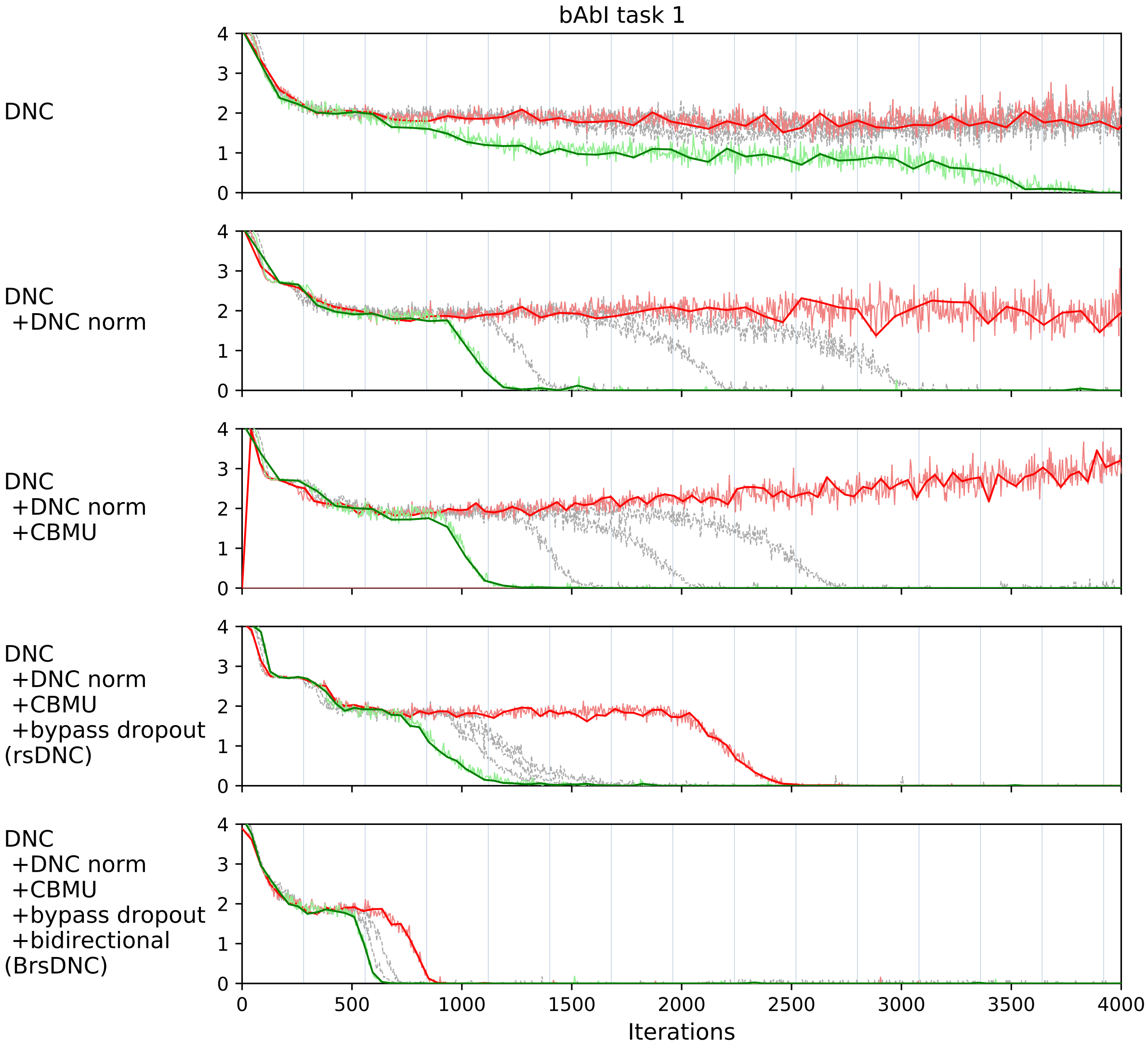


Figure 4: The convergence behaviour of the DNC with different improvements on the bAbI task 1. The plot shows the validation loss for 5 model with the same parameters but different initializations, the red line is the worst model and the green line the best.

C Results on the bAbI task

Task	DNC	EntNet	SDNC	rsDNC	BrsDNC	BrsDNC +aug16
1: 1 supporting fact	9.0 ± 12.6	0.0 ± 0.1	**0.0 ± 0.0**	0.1 ± 0.0	0.1 ± 0.1	0.1 ± 0.0
2: 2 supporting facts	39.2 ± 20.5	15.3 ± 15.7	7.1 ± 14.6	0.8 ± 0.5	0.8 ± 0.2	**0.5 ± 0.2**
3: 3 supporting facts	39.6 ± 16.4	29.3 ± 26.3	9.4 ± 16.7	6.5 ± 4.6	2.4 ± 0.6	**1.6 ± 0.8**
4: 2 argument relations	0.4 ± 0.7	0.1 ± 0.1	0.1 ± 0.1	0.0 ± 0.0	**0.0 ± 0.0**	**0.0 ± 0.0**
5: 3 argument relations	1.5 ± 1.0	**0.4 ± 0.3**	0.9 ± 0.3	1.0 ± 0.4	0.7 ± 0.1	0.8 ± 0.4
6: yes/no questions	6.9 ± 7.5	0.6 ± 0.8	0.1 ± 0.2	0.0 ± 0.1	**0.0 ± 0.0**	**0.0 ± 0.0**
7: counting	9.8 ± 7.0	1.8 ± 1.1	1.6 ± 0.9	**1.0 ± 0.7**	1.0 ± 0.5	1.0 ± 0.7
8: lists/sets	5.5 ± 5.9	1.5 ± 1.2	0.5 ± 0.4	**0.2 ± 0.2**	0.5 ± 0.3	0.6 ± 0.3
9: simple negation	7.7 ± 8.3	0.0 ± 0.1	0.0 ± 0.1	**0.0 ± 0.0**	0.1 ± 0.2	**0.0 ± 0.0**
10: indefinite knowledge	9.6 ± 11.4	0.1 ± 0.2	0.3 ± 0.2	0.1 ± 0.2	**0.0 ± 0.0**	0.0 ± 0.1
11: basic coreference	3.3 ± 5.7	0.2 ± 0.2	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**
12: conjunction	5 ± 6.3	**0.0 ± 0.0**	0.2 ± 0.3	**0.0 ± 0.0**	0.0 ± 0.1	**0.0 ± 0.0**
13: compound coreference	3.1 ± 3.6	0.0 ± 0.1	0.1 ± 0.1	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**
14: time reasoning	11 ± 7.5	7.3 ± 4.5	5.6 ± 2.9	**0.2 ± 0.1**	0.8 ± 0.7	0.3 ± 0.1
15: basic deduction	27.2 ± 20.1	3.6 ± 8.1	3.6 ± 10.3	**0.1 ± 0.1**	**0.1 ± 0.1**	**0.1 ± 0.1**
16: basic induction	53.6 ± 1.9	53.3 ± 1.2	53.0 ± 1.3	52.1 ± 0.9	52.6 ± 1.6	**0.0 ± 0.0**
17: positional reasoning	32.4 ± 8	8.8 ± 3.8	12.4 ± 5.9	18.5 ± 8.8	4.8 ± 4.8	**1.5 ± 1.8**
18: size reasoning	4.2 ± 1.8	1.3 ± 0.9	1.6 ± 1.1	1.1 ± 0.5	**0.4 ± 0.4**	0.9 ± 0.5
19: path finding	64.6 ± 37.4	70.4 ± 6.1	30.8 ± 24.2	43.3 ± 36.7	**0.0 ± 0.0**	0.1 ± 0.1
20: agents motivation	0.0 ± 0.1	**0.0 ± 0.0**	**0.0 ± 0.0**	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1
Mean WER:	16.7 ± 7.6	9.7 ± 2.6	6.4 ± 2.5	6.3 ± 2.7	3.2 ± 0.5	**0.4 ± 0.3**
Failed Tasks (>5%):	11.2 ± 5.4	5.0 ± 1.2	4.1 ± 1.6	3.2 ± 0.8	1.4 ± 0.5	**0.0 ± 0.0**

Table 3: The mean word error rate (WER) of the different models on the 20 bAbI tasks . All models are trained jointly on all 20 bAbI tasks at once without information about the actual task. Best results in bold.

Task	DNC	EntNet	EntNet †	DMN+ †	SDNC	RMN	rsDNC	BrsDNC	BrsDNC +aug16
1: 1 supporting fact	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**	0.1	0.1	0.1
2: 2 supporting facts	0.4	2.8	**0.1**	0.3	0.6	0.5	0.8	0.5	0.6
3: 3 supporting facts	1.8	10.6	4.1	1.1	**0.7**	14.7	2.5	2.5	1.6
4: 2 argument relations	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
5: 3 argument relations	0.8	0.4	**0.3**	0.5	**0.3**	0.4	1.6	0.7	0.4
6: yes/no questions	**0.0**	0.3	0.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
7: counting	0.6	0.8	**0.0**	2.4	0.2	0.5	1.5	0.3	0.6
8: lists/sets	0.3	0.1	0.5	**0.0**	0.2	0.3	0.1	0.4	0.6
9: simple negation	0.2	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
10: indefinite knowledge	0.2	**0.0**	0.6	**0.0**	0.2	**0.0**	**0.0**	**0.0**	**0.0**
11: basic coreference	**0.0**	**0.0**	0.3	**0.0**	**0.0**	0.5	**0.0**	**0.0**	**0.0**
12: conjunction	**0.0**	**0.0**	**0.0**	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**
13: compound coreference	**0.0**	**0.0**	1.3	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**
14: time reasoning	0.4	3.6	**0.0**	0.2	0.1	**0.0**	0.1	0.1	0.5
15: basic deduction	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	0.2	**0.0**
16: basic induction	55.1	52.1	**0.2**	45.3	54.1	0.9	52.0	49.9	**0.0**
17: positional reasoning	12.0	11.7	0.5	4.2	0.3	0.3	11.1	0.8	**0.2**
18: size reasoning	0.8	2.1	0.3	2.1	**0.1**	2.3	1.6	1.0	0.9
19: path finding	3.9	63.0	2.3	0.0	1.2	2.9	0.8	**0.0**	0.3
20: agents motivation	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.2	0.1	**0.0**
Mean WER:	3.8	7.4	0.5	2.8	2.9	1.2	3.6	2.8	**0.3**
Failed Tasks (>5%):	2	4	0	1	1	1	2	1	**0**

Table 4: The word error rate (WER) of the best runs on the bAbI 20 task. Best results per row in bold. Models tagged with † are trained on each task individually, the other are trained on all tasks jointly.