Attentive Memory Networks: Efficient Machine Reading for Conversational Search

Tom Kenter
University of Amsterdam
Amsterdam, The Netherlands
tom.kenter@uva.nl

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
derijke@uva.nl

ABSTRACT

Recent advances in conversational systems have changed the search paradigm. Traditionally, a user poses a query to a search engine that returns an answer based on its index, possibly leveraging external knowledge bases and conditioning the response on earlier interactions in the search session. In a natural conversation, there is an additional source of information to take into account: utterances produced earlier in a conversation can also be referred to and a conversational IR system has to keep track of information conveyed by the user during the conversation, even if it is implicit.

We argue that the process of building a representation of the conversation can be framed as a machine reading task, where an automated system is presented with a number of statements about which it should answer questions. The questions should be answered solely by referring to the statements provided, without consulting external knowledge. The time is right for the information retrieval community to embrace this task, both as a stand-alone task and integrated in a broader conversational search setting.

In this paper, we focus on machine reading as a stand-alone task and present the Attentive Memory Network (AMN), an end-to-end trainable machine reading algorithm. Its key contribution is in efficiency, achieved by having an hierarchical input encoder, iterating over the input only once. Speed is an important requirement in the setting of conversational search, as gaps between conversational turns have a detrimental effect on naturalness. On 20 datasets commonly used for evaluating machine reading algorithms we show that the AMN achieves performance comparable to the state-of-the-art models, while using considerably fewer computations.

CCS CONCEPTS

• **Human-centered computing** → **Natural language interfaces**;

ACM Reference format:
Tom Kenter and Maarten de Rijke. 2017. Attentive Memory Networks: Efficient Machine Reading for Conversational Search. In *Proceedings of 1st International Workshop on Conversational Approaches to Information Retrieval, Tokyo, Japan, August 11, 2017 (CAIR'17),* 7 pages.

CAIR'17, August 11, 2017, Tokyo, Japan

1 INTRODUCTION

Recent advances in conversational systems [18, 19] have changed the search paradigm. In a classic setting, a search engine answers a query based on an index, possibly enriching it with information from an external knowledge base [25]. Additionally, previous interactions in the same session can be leveraged [6]. In addition to these sources, in natural language conversations, information contained in previous utterances can be referred to, even implicitly. Suppose a conversational system has to answer the query *Where are my keys?* based on a previous statement *I was home before I went to work, which is where I found out I didn't have my keys with me.* The statement conveys a lot of information, including the likely possibility that the keys are still at the speaker's house. As is clear from this example, indices or external knowledge bases are of no avail in this setting. It is crucial for a conversational system to maintain an internal state, representing the dialogue with the user so far. To address this issue, substantial work has been done in goal-oriented dialogues, tailored to specific settings such as restaurant reservations [3] and the tourist domain [13]. We argue that a generic conversational agent should be able to maintain a dialogue state without being constrained to a particular task with predetermined slots to be filled. The time has come for the Information Retrieval (IR) community to address the task of machine reading for conversational search [18].

As an important step towards generic conversational IR [15], we frame the task of conversational search as a general machine reading task [10, 11], where a number of statements is provided to an automated agent that answers questions about it. This scenario is different from the traditional question answering setting, in which questions are typically factoid in nature, and answers are based on background knowledge or external sources of knowledge. In the machine reading task, much as in a natural conversation, a number of statements is provided, and the conversational agent should be able to answer questions based on its understanding of these statements alone. In [11], for example, a single Wikipedia page is provided to a machine algorithm which has to answer questions about it. In [26] the machine reads stories abouts persons and objects and has to keep track of their whereabouts.

Memory networks have proven to be an effective architecture in machine reading tasks [22, 27]. Their key component is a memory module in which the model stores intermediate representations of input, that can be seen as multiple views on the input so far, from which a final output is computed. Speed is an important constraint in the context of conversational agents, since long pauses between turns hamper the naturalness of a conversation. We strive for an efficient architecture, and propose to use a hierarchical input encoder. Input can be large, hundreds of words, and we hypothesize

that first processing the input to get a smaller set of higher-level input representations can benefit a network in two ways: (1) the higher-level representations provide a distilled representation of the input; (2) as there are fewer higher-level representations it should be (computationally) easier for the network to focus on the relevant parts of the input. In short, in this paper we present the *Attentive Memory Network* (AMN), an end-to-end trainable memory network, with hierarchical input encoder. To test its general applicability we use 20 machine reading datasets specifically designed to highlight different aspects of natural language understanding. We show that the AMN achieves performance comparable to the state-of-the-art models, while using considerably fewer computations.

2 RELATED WORK

Machine reading is a much-studied domain [4, 10, 11]. It is related to question answering, the difference being that in question answering, external domain or world knowledge is typically needed to answer questions [7, 17, 29], while in machine reading answers should be inferred from a given text.

Hierarchical encoders are employed in a dialogue setting in [20] and for query suggestion in [21]. In both works, the hierarchical encoder is also trained, for every input sentence, to predict every next input sentence, a setting we did not experiment with.

We build on previous work on memory networks [22, 23, 27], in particular on dynamic memory networks [16, 28]. Memory networks are an extension of standard sequence-to-sequence architectures; their distinguishing feature is a memory module added between the encoder and decoder. As they are typically applied in question answering settings, there are two encoders, one for a question and one for a document the question is about. The decoder does not have access to the input but only to the memory module, which distills relevant information from the input, conditioned on the question. The key difference between the Attentive Memory Network we propose and the work in [16, 28], is in the defining component, the memory module. In [16, 28], to obtain every next memory, a Gated Recurrent Unit (GRU) cell iterates over the input sequence. This leads to a memory intensive and computationally expensive architecture, since multiple cells are repeatedly being unrolled over the input sequence. The number of steps an Recurrent Neural Network (RNN) is unrolled for, i.e., the number of input representations it reads, together with the hidden state size, is the main determining factor regarding computational complexity. Therefore, we propose to obtain memories by an RNN that, rather than iterating over the entire input, only applies attention over it, which is a much cheaper operation (see §3).

In [22] an attention-based memory network is presented, where the input is represented as a sequence of embeddings on which attention is computed (i.e., there is no input reader). Our Attentive Memory Network differs from this work in that we do use an input reader, a hierarchical RNN. As a consequence, our memory module has far fewer hidden states to attend over. At the output side, we use GRUs to decode answers, which is different from the softmax over a dot product between the sum of attention-weighted input and question employed in [22].

To sum up, we propose a memory network that shares its overall architecture with previous models, and that differs in how all key components are constructed, with a view to improve efficiency and, thereby, enable its usage in conversational search scenarios.

3 ATTENTIVE MEMORY NETWORKS

To facilitate the presentation of our Attentive Memory Networks, we first briefly recapitulate standard sequence-to-sequence models.

Recurrent cells

An input sequence is processed one unit per time step, where the recurrent cell computes a new state h_t as a function of an input representation x and a hidden state h_{t-1} as:

$$\mathbf{h}_t = f(\mathbf{x}, \mathbf{h}_{t-1}; \theta), \tag{1}$$

based on internal parameters θ. The function f itself can be implemented in many ways, for example as an Long Short-Term Memory (LSTM) [12] or Gated Recurrent Unit (GRU) cell [5]. The initial hidden state $\mathbf{h}_0$ is usually a 0-vector. For a given input $\mathbf{X}^{enc} = [\mathbf{x}_1^{enc}, \mathbf{x}_2^{enc}, \ldots, \mathbf{x}_n^{enc}]$—e.g., embeddings representing words in a sentence—an encoder repeatedly applies this function, which yields an $n \times d^{enc}$ matrix $\mathbf{H}^{enc} = [\mathbf{h}_1^{enc}, \mathbf{h}_2^{enc}, \ldots, \mathbf{h}_n^{enc}]$ of n hidden states of dimension d^{enc}.

A decoder generates output according to Equation 1, where the initial hidden state is the last hidden state of the encoder $\mathbf{h}_n^{enc}$. The predicted output at time step t, $\hat{o}_t$, is typically generated from the hidden state of the decoder, h_t^{dec}, by calculating a softmax over the vocabulary V:

$$\hat{o} = \arg\max_{\mathbf{v} \in \mathbf{V}} \frac{e^{h_t^{dec} \cdot \mathbf{v}}}{\sum_{\mathbf{v}' \in \mathbf{V}} e^{h_t^{dec} \cdot \mathbf{v}'}}. \tag{2}$$

Here $\mathbf{V}$ is a matrix of vector representations v, representing words in the output vocabulary. At training time, the embedding of the correct word—the word that should have been returned—is usually given as input to the recurrent cell at time step $t + 1$.

Attention

An attention mechanism was proposed in [2], which gives the decoder access to the hidden states of an encoder. Instead of using Equation 1 to produce a new hidden state dependent only on the input, the computation now also depends on $\mathbf{H}^{att}$, the states to attend over, typically the states of the encoder. Following, e.g., [24], we have:

$$\begin{aligned} \mathbf{h}_t^{dec} &= g(\mathbf{x}^{dec}, \mathbf{H}^{att}, \mathbf{h}_{t-1}^{dec}) \\ &= \mathbf{W}_{proj} \cdot \mathbf{d}_t || \hat{\mathbf{h}}_t^{dec}, \end{aligned} \tag{3}$$

where $||$ is the concatenation operator, $\hat{\mathbf{h}}_t^{dec} = f(\mathbf{x}^{dec}, \mathbf{h}_{t-1}^{dec}; \theta^{dec})$ from Equation 1 and $\mathbf{d}_t$ is calculated from $\mathbf{H}^{att}$ by:

$$\begin{aligned} \mathbf{d}_t &= \sum_{i=1}^{n} a_{t,i}\, \mathbf{h}_i^{att} \\ \mathbf{a}_t &= \text{softmax}(\mathbf{u}_t) \\ \mathbf{u}_{t,i} &= v^T \tanh(\mathbf{W}_1 \mathbf{h}_i^{att} + \mathbf{W}_2 \mathbf{h}_t^{dec}), \end{aligned}$$

where $\mathbf{h}_i^{att}$ is the i-th state in $\mathbf{H}^{att}$ and $\mathbf{W}_1$ and $\mathbf{W}_2$ are extra parameters learned during training. From the hidden state produced this way, output can be generated by applying Equation 2 as usual.

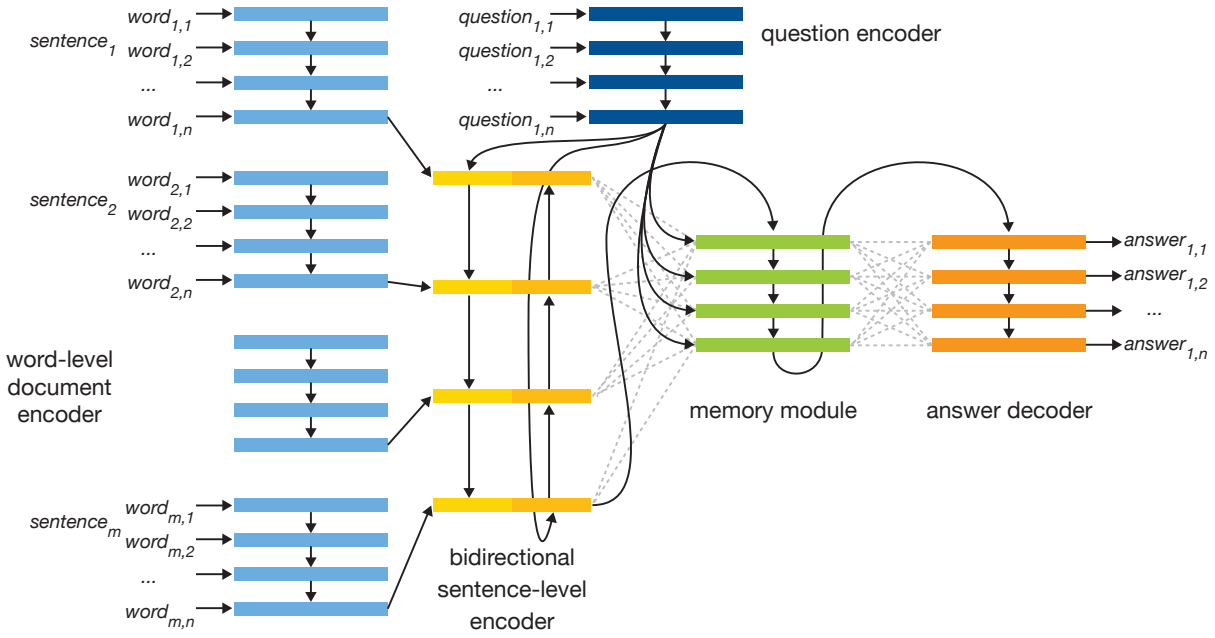


Figure 1: Attentive Memory Network. Connected blocks sharing color represent RNNs. Attention is depicted by dashed lines.

3.1 Attentive Memory Network architecture

We now present the Attentive Memory Network (AMN) architecture. AMNs, like traditional sequence-to-sequence networks, are composed of recurrent neural networks. Their key part is a memory module, which is a recurrent network itself. It stores memories by attending over the input document, conditioned on the question. As can be seen from Equation 3, the computational complexity of the attention mechanism is primarily dependent on the size of $\mathbf{H}^{att}$, the states to attend over. To keep this matrix small, a hierarchical approach is taken, where the input is first read by a word-level document encoder, which reads word embeddings—also trained by the model—per sentence to compute sentence representations. A sentence-level encoder iterates over these sentence embeddings to get a final document encoding. The memory module only has access to the sentence embeddings produced by the sentence-level encoder. For example, if the input consists of 20 sentences of 12 words each, the memory module of the AMN attends over 20 sentence representations, rather than over 240 representations, had a non-hierarchical word-level approach been taken.

Figure 1 shows a graphical overview of the network layout. There are two input encoders, a question encoder and a word-level document encoder. The memory module, the green block in Figure 1, attends over the sentence embeddings to extract relevant parts of the input, conditioned on the question. Lastly, the answer decoder attends over the memory states, to produce the final output. Let us turn to the details.

Question encoder. For encoding the question we use a single Recurrent Neural Network (RNN). For a question $Q \in \{q_1, q_2, \ldots, q_{|Q|}\}$ it produces a final state $\mathbf{h}^{que}_{|Q|}$, a vector of dimension d^{que}, that is used as a distributed representation of the question.

Document encoder. To encode the document we use a hierarchical approach. First, a word-level RNN is used to encode sentences. The word-level encoder is applied for every sentence individually. The unroll length is the maximum sentence length in words. For sentences $S \in \{s_1, s_2, \ldots, s_{|S|}\}$ the word-level encoder yields $\mathbf{H}^{wrd}$, an $|S| \times d^{wrd}$ matrix.

The sentence representations in $\mathbf{H}^{wrd}$ are read as a sequence by a sentence-level encoder. Following, e.g., [28], we use a bidirectional RNN for the sentence-level encoder, which for $|S|$ sentences and a hidden state size d^{sen} yields $\mathbf{H}^{sen}$, an $|S| \times d^{sen}$ matrix. The final state of the question encoder, $\mathbf{h}^{que}_{|Q|}$, is used as initial value of the hidden states of the sentence-level encoder.

Memory module. The memory module consists of a single recurrent cell that produces $\mathbf{M}$, a matrix of m memory representations of dimension d^{mem}. The i-th memory $\mathbf{m}_i$ is computed conditioned on the question representation and the sentence representations, analogous to Equation 3, as:

$$\mathbf{m}_i = g\left(\mathbf{h}^{que}_{|Q|}, \mathbf{H}^{sen}, \mathbf{m}_{i-1}\right). \tag{4}$$

That is, the final representation of the question encoder $\mathbf{h}^{que}_{|Q|}$ is repeatedly provided as input to a recurrent cell, whose hidden state is computed from the memory it produced previously, $\mathbf{m}_{i-1}$, while attending over the hidden states of the sentence-level encoder $\mathbf{H}^{sen}$.

The final representation of the sentence-level document encoder $\mathbf{h}^{sen}_{|S|}$ is used to initialize the hidden state of the memory cell, $\mathbf{m}_0$.

Answer decoder. Finally, the decoder produces an answer using Equation 2, where $\mathbf{h}^{dec}_t$ is computed by attending over the memory states:

$$\mathbf{h}^{dec}_t = g(\mathbf{x}^{dec}_t, \mathbf{M}, \mathbf{h}^{dec}_{t-1}).$$

3.2 Efficiency

As can be seen from Equation 4, the memory module is a recurrent cell itself. In previous memory networks, the memory module passes over the input multiple times, updating memory after each pass [16, 28]. The key difference in our approach is that AMNs iterate over the input only once, but *attend* over it multiple times. This is more efficient, as the attention mechanism (Equation 3) has far less paramaters than an LSTM or GRU recurrent cell, which update multiple gates and an internal state at every time step. The attention mechanism calculates a softmax over the input encodings, the number of which in our case is reduced to number of input sentences, rather than words, by the hierarchical encoder.

Additionally, the AMN needs relatively few iterations to learn. Details per evaluation set are provided in §5.2.

4 EXPERIMENTAL SETUP

To the best of our knowledge, there is currently no conversational search data set (consisting of sequences of utterances plus questions about these utterances) on which we could evaluate AMN. Instead we evaluate AMN on a broad collection of more traditional machine reading datasets. Specifically, we evaluate AMN on the 20 datasets provided by the bAbi tasks [26], of which we use the 10k sets, version 1.2. The sets consist of stories, 2 to over 100 sentences in length, and questions about these stories. The 20 sets are designed to highlight different aspects of natural language understanding like counting, deduction, induction and spatial reasoning. As argued by Kumar et al. [16], while showing the ability to solve one of the bAbi tasks is not sufficient to conclude a model would succeed at the same task on real world text data —such as conversational search data— it is a necessary condition.

Every dataset in the bAbi collection comes as a training set of 10,000 examples and a test set of 1,000 examples. We split the 10,000 training examples of each dataset into a training set—the first 9,000 examples—and a validation set—the remaining 1,000 examples—on which we tune the hyperparameters. All text is lowercased.

We use GRU cells [5] for all recurrent cells. To restrict the number of hyperparameters to tune, the same value is used for all embedding sizes, and for the state sizes of all recurrent cells. I.e., for an embedding size e, we have $e = d^{que} = d^{wrd} = d^{sen} = d^{mem}$, which is either 32 or 64. The weights of the question encoder and document word-level encoder are tied. GRU cells can be stacked and we experiment with 1 to 3 level deep encoder, memory, and decoder cells, the depths of which always match (i.e., if, for example, 3-level encoder cells are used, 3-level decoder cells are used). We use a single embedding matrix for the words in the question, document and answer. The number of memories to generate, m, is chosen from {1, 2, 3}. Dropout is applied at every recurrent cell, the dropout probability being either 0.0 (no dropout), 0.1 or 0.2. We optimize cross entropy loss between actual and predicted answers, using Adam [14] as optimization algorithm and set the initial learning rate to one of {0.1, 0.5, 1.0}. We measure performance every 1000 training examples. If the loss does not improve or performance on the validation set decreases for three times in a row, the learning rate is annealed by dividing it by 2. The maximum norm for gradients is either 1 or 5. The batch size is set to 50.

Table 1: Results in terms of error rate on the bAbi 10k tasks. For comparison, results of previous work are copied from [22, MemN2N], [8, DNC], [28, DMN+], and [9, EntNet].

Dataset	MemN2N	DNC	DMN+	EntNet	AMN
single supporting fact	0.0	0.0	0.0	0.0	0.0
two supporting facts	0.3	0.4	0.3	0.1	4.1
three supporting facts	2.1	1.8	1.1	4.1	29.1
two arg relations	0.0	0.0	0.0	0.0	0.0
three arg relations	0.8	0.8	0.5	0.3	0.7
yes-no questions	0.1	0.0	0.0	0.2	0.2
counting	2.0	0.6	2.4	0.0	3.1
lists sets	0.9	0.3	0.0	0.5	0.3
simple negation	0.3	0.2	0.0	0.1	0.0
indefinite knowledge	0.0	0.2	0.0	0.6	0.1
basic coreference	0.1	0.0	0.0	0.3	0.0
conjunction	0.0	0.0	0.0	0.0	0.0
compound coreference	0.0	0.1	0.0	1.3	0.0
time reasoning	0.1	0.4	0.2	0.0	3.6
basic deduction	0.0	0.0	0.0	0.0	0.0
basic induction	51.8	55.1	45.3	0.2	45.4
positional reasoning	18.6	12.0	4.2	0.5	1.6
size reasoning	5.3	0.8	2.1	0.3	0.9
path finding	2.3	3.9	0.0	2.3	0.3
agents motivations	0.0	0.0	0.0	0.0	0.0
number of tasks solved	18	18	19	20	18

Table 2: Hyperparameter values for the minimal AMNs that were fastest in achieving best performance on the validation set. The size refers to both size of embeddings and hidden states. The last column lists the number of batches needed.

Dataset	size	# layers	# mem	# batches
single supporting fact	32	1	1	1,000
two supporting facts	64	2	3	12,200
three supporting facts	64	2	3	14,000
two arg relations	32	1	1	1,200
three arg relations	32	1	2	3,000
yes-no questions	32	1	1	3,800
counting	32	1	3	5,000
lists sets	32	1	1	4,400
simple negation	32	1	2	3,200
indefinite knowledge	32	1	1	3,800
basic coreference	32	1	2	1,400
conjunction	32	1	1	1,200
comp coreference	32	1	1	10,000
time reasoning	64	2	1	6,000
basic deduction	32	1	1	2,200
basic induction	64	1	2	10,200
positional reasoning	32	1	3	6,200
size reasoning	32	1	3	2,400
path finding	64	1	1	13,000
agents motivations	32	1	3	3,600

We implemented the AMN in Tensorflow [1]. The implementation is released under an open source license and is available at https://bitbucket.org/TomKenter/attentive-memory-networks-code.

5 RESULTS AND ANALYSIS

We present the results of the experiments described in §4 and provide an analysis of the results.

5.1 Main results

Table 1 lists the results of our Attentive Memory Network (AMN) on the 20 bAbi 10k datasets, together with results of previous approaches. Following [26], we consider a dataset solved if the error rate is less than 5%.

As can be seen from the Table 1, AMN solves 18 of the 20 datasets. This is particularly noteworthy given the fact that it is a general framework, not catered towards tracking entities (as in [9]). Moreover, the AMN needs an order of magnitude fewer computation steps than previous memory network architectures used for these tasks [16, 28] as it only reads the input once.

There are two tasks the AMN does not solve. The *basic induction* set proves to be hard for the AMN, as it does for most other networks. More interestingly, the *three supporting facts* sets is problematic as well. This dataset has the longest documents, sometimes over 100 sentences long. Analysis of the results, see below for examples, shows that the probability mass of the attention vectors of the memory module is much more spread out across sentences then it is in other sets. That is, the network struggles to keep its attention focused.

The results in Table 1 show that the AMN can solve a wide variety of machine reading tasks and that it behaves different from other memory networks.

5.2 Analysis

We analyze the hyperparameter settings used to produce the results in Table 1 and provide examples of the inner workings of the attention mechanism of the memory module.

Hyperparameters and speed of convergence. Table 2 lists the hyperparameter values for the smallest AMNs that achieve the best performance on the validation set, with fewest training examples. Here, *smallest network* refers to the size of the network in terms of embedding size and number of memories. The last column lists the number of batches needed. As can be seen from Table 2, AMNs can learn fast. As an example, it needs only 5 epochs to solve the first dataset: there are 10k examples—1,000 batches of 50 examples = 50k examples = 5 epochs. This is in contrast to the 100 epochs reported in [22] and 256 epochs listed as a maximum in [16].

Interestingly, adding depth to a network by stacking GRU cells was helpful in only 3 out of 20 cases.

Result analysis. Figure 2 shows a visualization of the attention vectors of the memory module. The attention is visualized per memory step. Although some stories in the dataset are over 100 sentences in length, short examples were picked here, for reasons of brevity. Every column represents a memory step, and the values per memory step add up to 1 (barring rounding errors).

Figure 2a shows an example where one memory step is used. The attention focuses on the last time Daniel, the person the question is about, is mentioned. Interestingly, the second sentence also gets some attention, presumably because the bedroom, which features in the question, is being referred to. A particularly striking detail is that—correctly—nearly no attention is paid to the fifth sentence, although it is almost identical to the question.

In Figure 2b, attention is highest for sentences in which the person being asked about is referred to. This is especially noteworthy, as the reference is only by a personal pronoun, which moreover refers to two people.

For the *size reasoning* dataset, three memory steps were needed (see Table 2). An example is shown in Figure 2c. The first memory step mistakenly focuses on the sixth sentence about the chest. Gradually, however, the memory module recovers from this error, and attention shifts to the fourth sentence about the suitcase.

Figure 2d shows the ability of the network to focus only on relevant parts. Although the seventh and tenth sentence are nearly identical, it is the last sentence that matters, and it is this sentence the network attends to almost solely. Curiously, the two memory steps attend to the same sentences, which is consistently the case for this dataset. This might indicate that a single memory step could suffice too. Indeed, experiments show that on some datasets networks with fewer memory steps achieve the same or nearly the same performance as bigger networks, but take longer to reach it. The extra memory steps might serve as extra training material.

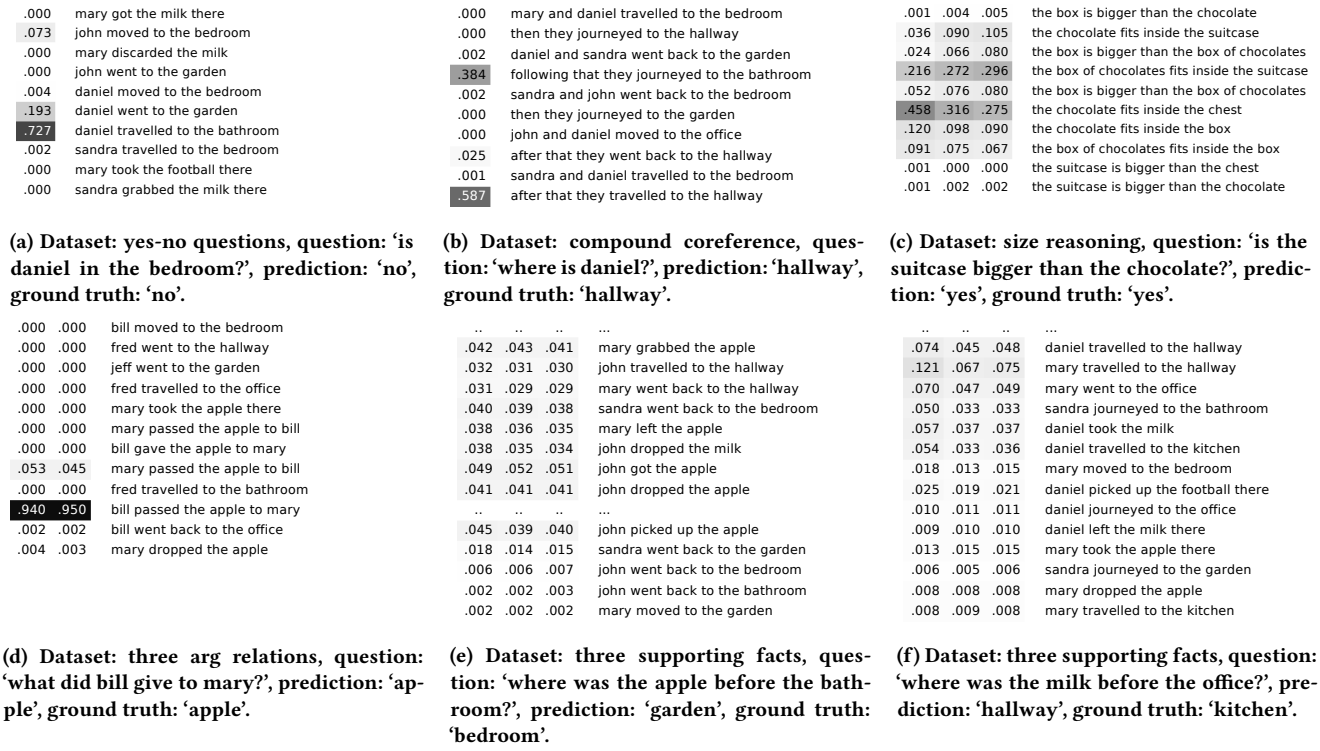
The last two cases, Figure 2e and 2f, are from the *three supporting facts* dataset that the model could not solve. What stands out immediately is the fact that the attention is much more spread out than in other cases. This is the case throughout the entire dataset. It shows that the model is confused and fails to learn what is relevant. In Figure 2e just reading the last five sentences would have been enough. The model does seem to capture that John picked up the apple, but only very weakly so. The crucial sentence, third form the end, is the sentence the model pays least attention to. Figure 2e shows the model being even more confused. It starts out by attending mostly to Mary, who has nothing to do with the story. The sentences that do matter, again, get very little attention.

Overall, these examples indicate that, when the AMN learns to solve a task, its memory module is very decisive in paying attention to the relevant parts of the input and ignoring the rest.

6 CONCLUSION

As search becomes more conversational, the machine reading task, where a system is able to answer questions against prior utterances in a conversation, becomes a highly relevant task for Information Retrieval (IR). We introduced Attentive Memory Networks (AMNs), efficient end-to-end trainable memory networks with a hierarchical input encoder. AMNs perform nearly as well as existing machine reading algorithms, with less computation. Analysis shows they typically need only a few epochs to achieve optimal performance, making them ideally suited for IR's high efficiency settings. Our findings indicate that a straightforward architecture like the AMN is sufficient for solving a wide variety of machine reading tasks.

The bAbi datasets provide an ideal test bed for machine reading algorithms as the tasks and evaluation are well-defined. However, it would also be interesting to test the performance of AMNs on bigger datasets, with more varied and noisier problems, especially ones that are directly derived from conversational search scenarios.



(a) Dataset: yes-no questions, question: 'is daniel in the bedroom?', prediction: 'no', ground truth: 'no'.

(b) Dataset: compound coreference, question: 'where is daniel?', prediction: 'hallway', ground truth: 'hallway'.

(c) Dataset: size reasoning, question: 'is the suitcase bigger than the chocolate?', prediction: 'yes', ground truth: 'yes'.

(d) Dataset: three arg relations, question: 'what did bill give to mary?', prediction: 'apple', ground truth: 'apple'.

(e) Dataset: three supporting facts, question: 'where was the apple before the bathroom?', prediction: 'garden', ground truth: 'bedroom'.

(f) Dataset: three supporting facts, question: 'where was the milk before the office?', prediction: 'hallway', ground truth: 'kitchen'.

Figure 2: Attention visualizations. The attention is visualized per memory step. Every column represents a memory step, and adds up to 1 (allowing for rounding errors), except in the last two examples where some (irrelevant) sentences were left out. Although some stories in the dataset are over 100 sentences in length, short examples were picked here, for brevity.

Memory networks have also been applied in settings where external knowledge is available, in particular in the form of key-value pairs [17]. Although this setting is different from the machine reading setting, it would be interesting to see how AMNs could be applied here. Finally, in a conversational setting involving multiple actors, it would be challenging for the memory module to attend to the utterances of the right actor at the right time. A richer attention-like mechanism seems to be needed. One that allows a decoder to attend to specific parts of the input, including the utterances produced by the system itself, conditioned on whose utterances are being referred to.

ACKNOWLEDGMENTS

We would like to thank Nikos Voskarides of the University of Amsterdam for valuable feedback on an earlier version of the manuscript, and Llion Jones and Daniel Hewlett of Google Research for many inspiring discussions on topics related to the work in this paper.

This research was supported by Ahold Delhaize, Amsterdam Data Science, the Bloomberg Research Grant program, the Criteo Faculty Research Award program, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the Microsoft Research Ph.D. program, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 612.001.116, HOR-11-10, CI-14-25, 652.002.001, 612.001.551, 652.001.003, and Yandex. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2015).
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *ICLR*.
[3] Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683* (2016).
[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *EMNLP*.
[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.
[6] Carsten Eickhoff, Jaime Teevan, Ryen W. White, and Susan T. Dumais. 2014. Lessons from the journey: a query log analysis of within-session learning. In *WSDM*.
[7] Anthony Fader, Luke S. Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *ACL*.
[8] 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 (2016), 471–476.
[9] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2016. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969* (2016).
[10] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*.

[11] Daniel Hewlett, Alexandre Lacoste, Llion Jones, Illia Polosukhin, Andrew Fandrianto, Jay Han, Matthew Kelcey, and David Berthelot. 2016. WIKIREADING: A novel large-scale language understanding task over Wikipedia. In *ACL*.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[13] Seokhwan Kim, Luis Fernando D'Haro, Rafael E Banchs, Jason D Williams, and Matthew Henderson. 2017. The fourth dialog state tracking challenge. In *Dialogues with Social Robots*. Springer, 435–449.

[14] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

[15] Julia Kiseleva and Maarten de Rijke. 2017. Evaluating personal assistants on mobile devices. In *1st International Workshop on Conversational Approaches to Information Retrieval (CAIR'17)*. ACM.

[16] 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 *ICML*.

[17] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *EMNLP*.

[18] Filip Radlinski and Nick Craswell. 2017. A theoretical framework for conversational search. In *CHIIR*. ACM, 117–126.

[19] Iulian Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*.

[20] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. A hierarchical latent variable encoder-decoder model for generating dialogues. *arXiv preprint arXiv:1605.06069* (2016).

[21] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *CIKM*.

[22] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *NIPS*.

[23] Ke Tran, Arianna Bisazza, and Christof Monz. 2016. Recurrent memory networks for language modeling. In *NAACL-HLT*.

[24] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *NIPS 2015*.

[25] Zhongyuan Wang, Kejun Zhao, Haixun Wang, Xiaofeng Meng, and Ji-Rong Wen. 2015. Query understanding through knowledge-based conceptualization. In *IJCAI*.

[26] 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. In *ICLR*.

[27] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *ICLR*.

[28] Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *ICML*.

[29] Yi Yang, Wen tau Yih, and Christopher Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *EMNLP*.