Separating Answers from Queries for Neural Reading Comprehension

Dirk Weissenborn
Language Technology Lab, DFKI
Alt-Moabit 91c
Berlin, Germany
dirk.weissenborn@dfki.de

Abstract

We present a novel neural architecture for answering queries, designed to optimally leverage explicit support in the form of query-answer memories. Our model is able to refine and update a given query while separately accumulating evidence for predicting the answer. Its architecture reflects this separation with dedicated embedding matrices and loosely connected information pathways (modules) for updating the query and accumulating evidence. This separation of responsibilities effectively decouples the search for query related support and the prediction of the answer. On recent benchmark datasets for reading comprehension, our model achieves state-of-the-art results. A qualitative analysis reveals that the model effectively accumulates weighted evidence from the query and over multiple support retrieval cycles which results in a robust answer prediction.

1 Introduction

Recent advances in many NLP tasks were achieved by utilizing neural architectures that employ some form of external memory. Making use of explicit memories enables these models to bridge long-range dependencies and solve more complex reasoning tasks that might involve multiple observations. Neural architectures equipped with explicit memories are able to achieve impressive results on a variety of NLP tasks. Memory Networks (Weston et al., 2015; Sukhbaatar et al., 2015), for example, are able to answer questions which require a higher level of reading comprehension and possibly reason over multiple observations.

The use of some form of external memory appears essential when tackling complex queries that require comprehension of a given context (support). The memory module stores explicit, contextual information of the support which either contains the correct answer or clues that can lead to it. For instance, attention-based architectures (Hermann et al., 2015) encode supporting contexts typically with (bi-directional) recurrent neural networks (RNN) into h-dimensional latent representations (*hidden states*), which jointly serve as a form of memory. End-to-end Memory Networks (Sukhbaatar et al., 2015) are similar although they split the support into individual parts that are separately encoded to form *memories*. These systems utilize the same learned query representation both for selecting memories (matching) and predicting the actual answer (prediction) which can affect the overall performance. Recent work successfully addressed this issue by directly using the retrieved hidden states (Cheng et al., 2016) or the attention weights (Kadlec et al., 2016) as pointers to the answer (Vinyals et al., 2015) for prediction.

In this work we propose a novel end-to-end neural architecture for answering queries. It explicitly separates queries from answers which is reflected in the representation of supporting knowledge as query-answer pairs and in the general architecture. In particular, we employ dedicated embedding matrices and loosely connected information pathways (modules) for updating the query and answer representation. This separation of responsibilities increases the

capabilities of the model to search through the support while selectively accumulating evidence for the answer in parallel. The representation of the support reflects the task of answering queries directly which facilitates its utilization by the model. We evaluate our approach on two reading comprehension tasks that involve answering cloze-style (Taylor, 1953) queries, namely the CNN/DailyMail QA task (Hermann et al., 2015) and the named entity (NE) subtask of the Children's Book Test (CBT) (Hill et al., 2016). These datasets provide only one document as support per query but this is not a restriction because our model can also handle multiple documents. Our contributions are the following: i) we introduce a new representation of supporting memories in form of query-answer pairs (§3.1) based on which ii) we develop a neural architecture for answering queries that leverages this representation (§3), iii) we evaluate our system on two reading comprehension benchmark datasets against other competitive systems achieving state-of-the-art results (§4.2), and iv) we give insights into the systems ability to utilize multiple support retrieval cycles for improving its reading comprehension performance (§4.3 and §4.4).

2 Related Work

Utilizing explicit memory in end-to-end differentiable neural architectures has enabled models to solve complex tasks that require learning simple algorithms, or processing and reasoning over large amounts of contextual information. Traditional architectures, such as RNNs like the LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Chung et al., 2014), are not suited for these kind of tasks due to their limited memory capacity and difficulties to learn long-range dependencies in large contexts.

Graves et al. (2014) introduced Neural Turing Machines (NTM). NTMs augment traditional RNNs with external memory that can be written to and read from. The memory is composed of a predefined number of writable slots. They are addressable via content or position shifts which allows solving simple algorithmic tasks. The capacity is also limited, but the external memory slots can carry information over long ranges more easily than traditional RNNs. NTMs inspired subsequent work on using different kinds of external memory, like queues and stacks for solving transduction tasks (Grefenstette et al., 2015) or neural theorem provers to perform first-order (Rocktäschel and Riedel, 2016) inference.

Attention-based architectures store information, typically in form of hidden RNN states, dynamically for each time-step while processing a given context. These states are retrieved through an attention mechanism that softly selects a state that matches a given query state. This can be viewed as keeping the encoded context in memory. Such architectures achieved impressive results on tasks that involve larger contexts such as Reading Comprehension (Hermann et al., 2015; Kadlec et al., 2016; Chen et al., 2016), Machine Translation (Bahdanau et al., 2015; Luong et al., 2015) or recognizing textual entailment (Rocktäschel et al., 2016; Cheng et al., 2016).

Based on ideas of the attention mechanism, End-to-end Memory Networks (Sukhbaatar et al., 2015) select explicit memories for query answering. Memories are encoded into two representations: i) the input representation for query matching and ii) the output representation for subsequent utilization. This distinction is important, because the representation that is used to match the original query has a different responsibility than the representation that is used to answer or update the query. Thus, attention-based approaches for answering queries using supporting documents can be considered a special case of Memory Networks where hidden states form both the input- and output representation of the individual memories which are jointly encoded. Variants of Memory Networks have achieved very good results in various NLP tasks, such as language modeling (Sukhbaatar et al., 2015), reading comprehension (Hill et al., 2016) and question answering (Sukhbaatar et al., 2015; Kumar et al., 2015; Miller et al., 2016).

One important contribution of Memory Networks is the idea of refining or updating the query (Sukhbaatar et al., 2015) or memories (Kumar et al., 2015) for multiple memory retrieval cycles before answering the query. This idea lead to significant improvements for architectures that employ the attention mechanism iteratively for reading comprehension tasks (Sordoni et al., 2016).

Recently, other forms of explicit memory have

been suggested for neural architectures. For instance, associative memory can be used to effectively compress multiple memories into redundant copies of a single memory array. It has shown very promising results, e.g., for language modeling (Danihelka et al., 2016) or recognizing textual entailment (Weissenborn, 2016), and might therefore be suited to compress large amounts of external memories when used in conjunction with our model.

3 Query-Answer Neural Network

Our query-answer neural network utilizes supporting knowledge in the form of explicit query-answer pairs (z, y) to predict the answer a to a given query q. Answers from support (y) are weighted via matching scores between their corresponding support query z and the actual query q. Once a weighted query-answer pair $(\tilde{z}, \tilde{y})$ has been retrieved, it is used to update the current query and the predicted answer representation for a subsequent support retrieval cycle (hop). This process can be repeated for a specified number of hops (T). Finally, we use the predicted answer representation after T hops as input for answer classification given a set of possible answer candidates. Note that this approach does not require supporting answers to correspond to the answer candidates.

3.1 Supporting Knowledge

Our model stores supporting knowledge as pairs of queries (z) and answers (y). Given a set of supporting documents, (z, y)-pairs are formed by i) detecting task-specific *spans-of-interest* (SOIs), ii) forming (z, y)-pairs for each SOI. In this work we consider cloze-style (z, y)-pairs. Thus, given a (z, y)-pair z corresponds to an entire document with a gap at a particular SOI (cloze-text) and y to the filler of this gap. Consider the following example:

Query:
q: Schweinsteiger plays for the national team of __
Support Document 1:
D: Schweinsteiger scored against Ukraine
Support Document 2:
D: Germany played against Ukraine

From this example we extract the following supporting query-answer pairs, if we identified all countries (underlined) as SOIs:

Support QA 1:
z: Schweinsteiger scored against __
y: Ukraine
Support QA 2:
z: __ played against Ukraine
y: Germany
Support QA 3:
z: Germany played against __
y: Ukraine

Note that spans-of-interest can cover almost anything, e.g., entire sentences or only single words. Defining SOIs and their respective answers can be adapted to the needs of the task at hand.

3.2 Encoding Queries

Given a (supporting) document $D = (x_1, ..., x_N)$ of symbols and spans-of-interest (SOIs) $P = \{(l_s, l_e) \mid l_s, l_e \in [1, N]\}$, at first all symbols x_l are embedded by an embedding matrix E_i. Next, the entire document D is encoded by a bi-directional RNN resulting in representations $\boldsymbol{h}_l^f \in \mathbb{R}^h$ of the forward-RNN and $\boldsymbol{h}_l^b \in \mathbb{R}^h$ of the backward-RNN for each document position $l \in [1, N]$. Afterwards, we form the following query representation for each $(l_s, l_e) \in P$:

$$\boldsymbol{z}^l = W_q \begin{bmatrix} \boldsymbol{h}_{l_s-1}^f \\ \boldsymbol{h}_{l_e+1}^b \end{bmatrix} \quad W_q \in \mathbb{R}^{h \times 2h} \tag{1}$$

The trainable parameter-matrix W_q is initialized with $[I^n; I^n]$ and additional random noise, where I^n is the identity matrix. Thus, initially $\boldsymbol{z}^l$ corresponds roughly to the sum of the forward state $\boldsymbol{h}_{l_s-1}^f$ of the left context and the backward state $\boldsymbol{h}_{l_e+1}^b$ of the right context. In order to ensure that the query representation only considers the outer context of the respective SOI it is required that $l_s \leq l_e$.[1] Encoding supporting queries in this way has the advantage, that the entire context is encoded in contrast to restricting

[1] It is also possible to define $l_s > l_e$, s.t. the span-of-interest becomes part of the query which might be important for some tasks. However, we are not considering these types of tasks in this work.

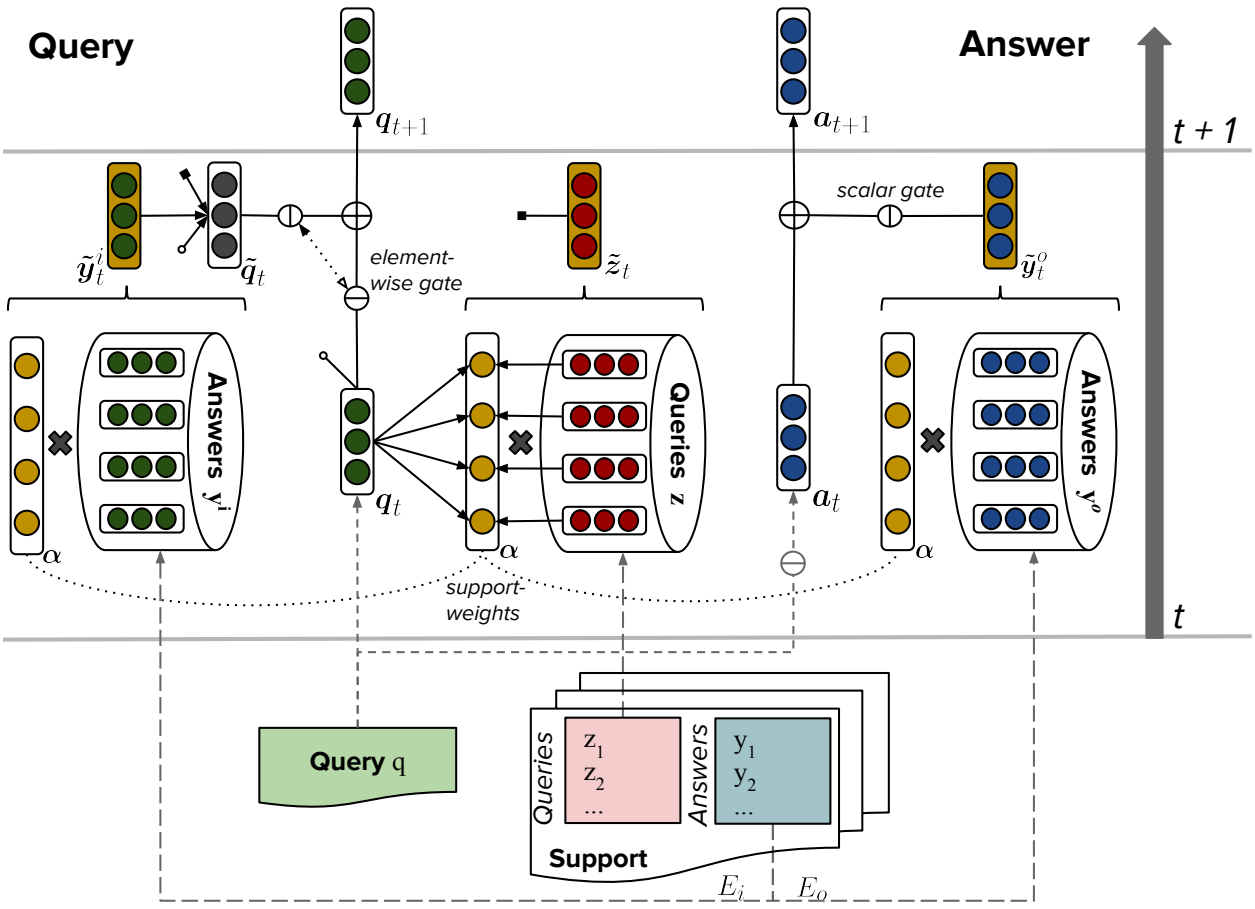


Figure 1: Illustration of our architecture which demonstrates an support retrieval cycle (hop) along with its corresponding update of the query and answer utilizing supporting queries and answers ((z, y)-pairs). The query representation is initialized ($t = 0$) by encoding the query string q. The initial answer representation is computed based on the initial query representation.

the context to a fixed-size context window or sentence. Furthermore, word-order and positional information is captured naturally by employing RNNs. Encoding the actual query q is identical to the encoding supporting queries z.

3.3 Encoding Answers

In this work, we consider answers to be individual symbols. However, this approach can be extended to sequences of symbols as well.

Answer Candidates Answer candidates $c \in A_q$ for a query q are embedded ($\boldsymbol{c}$) by a second embedding matrix E_o.

Supporting Answers For cloze-style queries supporting answers (y) correspond to the symbols at SOIs within the support.[2] There are two encodings of y with different applications: i) its corresponding output embedding $\boldsymbol{y}^o$ from E_o, and ii) its corresponding input embedding $\boldsymbol{y}^i$ from E_i. $\boldsymbol{y}^o$ is used to update the current answer of the model and $\boldsymbol{y}^i$ is used to update the query.

The intuition behind using $\boldsymbol{y}^i$ for updating the query representation is that we want to use the answer as if it was a word of the original query and would thus be embedded by E_i. $\boldsymbol{y}^o$ corresponds to the embeddings of E_o that are only used for answer prediction.

[2]Note, supporting answers do not necessarily have to correspond to an actual answer candidate.

3.4 Supporting Answer Retrieval

A supporting answer $\tilde{y}$ is selected softly from all (M) supporting (z, y)-pairs by $\mathrm{softmax}$-weights based on similarity scores between all $\boldsymbol{z}_k$ and the query $\boldsymbol{q}$. These support weights can be viewed as attention weights over the respective supporting (z, y)-pairs.

$$\alpha_k = \frac{\exp(\boldsymbol{q} \cdot \boldsymbol{z}_k)}{\sum_{k'=1}^{M} \exp(\boldsymbol{q} \cdot \boldsymbol{z}_{k'})} \tag{2}$$

$$\begin{bmatrix} \tilde{\boldsymbol{y}}^i \\ \tilde{\boldsymbol{y}}^o \\ \tilde{\boldsymbol{z}} \end{bmatrix} = \sum_{k=1}^{M} \alpha_k \begin{bmatrix} \boldsymbol{y}_k^i \\ \boldsymbol{y}_k^o \\ \boldsymbol{z}_k \end{bmatrix} \tag{3}$$

3.5 Query & Answer Update

The query representation $\boldsymbol{q}$ and predicted answer representation $\boldsymbol{a}$ are consecutively updated by using supporting (z, y)-pairs realizing multi-hop support retrieval. For instance, in the example of §3.1 the model might find *Support QA 1* to fit the original query best and retrieve the (wrong) answer (*Ukraine*). It is reasonable to update the original query with *Support QA 1* which includes the answer *Ukraine*. The subsequent, updated query eventually leads to the correct answer *Germany* of *Support QA 2*. Figure 1 illustrates this process.

We utilize the weighted support-queries $\tilde{\boldsymbol{z}}_t$ and their corresponding answer input-representations $\boldsymbol{y}_t^i$ (Eq. 3) to update the current query $\boldsymbol{q}_t$ by an element-wise weighted addition (Eq. 4), where $\boldsymbol{q}_0 = \boldsymbol{q}$ (the encoded query).

$$\begin{aligned} \tilde{\boldsymbol{q}}_t &= \tanh\left(U_c^q \begin{bmatrix} \boldsymbol{q}_t \\ \tilde{\boldsymbol{y}}_t^i \\ \tilde{\boldsymbol{z}}_t \end{bmatrix}\right) \\ \boldsymbol{g}_t^q &= \mathrm{sigmoid}\left(U_g^q \begin{bmatrix} \boldsymbol{q}_t \\ \tilde{\boldsymbol{z}}_t \end{bmatrix} + \boldsymbol{b}_g^q\right) \\ \boldsymbol{q}_{t+1} &= \boldsymbol{g}_t^q \odot \boldsymbol{q}_t + (1 - \boldsymbol{g}_t^q) \odot \tilde{\boldsymbol{q}}_t \end{aligned} \tag{4}$$

The answer is initialized by a gated linear transformation of the initial query $\boldsymbol{q}_0$ (Eq. 5). The query-answer-gate g_q^a decides whether the query itself can be utilized to infer the answer for a specific task or not. The answer representation at hop t represented by $\boldsymbol{a}_t$ is updated to $\boldsymbol{a}_{t+1}$ by adding the gated, retrieved answer $\tilde{\boldsymbol{y}}_t^o$ (Eq. 6). The scalar answer accumulation gate g_t^a (Eq. 7) depends on: i) the similarity between the current query $\boldsymbol{q}_t$ and the weighted support queries $\tilde{\boldsymbol{z}}_t$, ii) the similarity of the original query encoded as answer $\boldsymbol{a}_0$ and the weighted support answer representation $\tilde{\boldsymbol{y}}_t^o$ retrieved from support and iii) η_t which measures the highest answer candidate probability if $\tilde{\boldsymbol{y}}_t^o$ was the final answer representation (Eq. 8 which refers to §3.6).

$$\boldsymbol{a}_0 = \mathrm{sigmoid}(g_q^a)\, U_q^a \boldsymbol{q} \tag{5}$$

$$\boldsymbol{a}_{t+1} = \boldsymbol{a}_t + g_t^a \tilde{\boldsymbol{y}}_t^o \tag{6}$$

$$g_t^a = \mathrm{sigmoid}\left(\boldsymbol{u}_g^a \begin{bmatrix} \boldsymbol{q}_t \odot \tilde{\boldsymbol{z}}_t \\ \boldsymbol{a}_0 \odot \tilde{\boldsymbol{y}}_t^o \\ \eta_t \end{bmatrix} + b_a\right) \tag{7}$$

$$\eta_t = \max_{c \in A_q} p_q(\boldsymbol{c} | \tilde{\boldsymbol{y}}_t^o) \tag{8}$$

The trainable parameters of this module have the following dimensions: $U_c^q \in \mathbb{R}^{h \times 3h}$; $U_g^q \in \mathbb{R}^{h \times 2h}$; $U_a \in \mathbb{R}^{h \times h}$; $\boldsymbol{b}_g^{\cdot}, \boldsymbol{u}_g^a \in \mathbb{R}^h$; $g_q^a, u_a, b_a \in \mathbb{R}$.

3.6 Answer Scoring & Training

After a maximum number of hops T, scores $s_{q,c}$ of all answer candidates $c \in A_q$ are calculated using the inner product between their respective embeddings $\boldsymbol{c}$ and the final answer representation $\boldsymbol{a}_T$:

$$\begin{aligned} \forall c \in A_q : s_q(\boldsymbol{a}, \boldsymbol{c}) &= \boldsymbol{a} \cdot \boldsymbol{c} \\ p_q(\boldsymbol{c} | \boldsymbol{a}) &= \frac{e^{s_q(\boldsymbol{a}, \boldsymbol{c})}}{\sum_{c' \in A_q} e^{s_q(\boldsymbol{a}, \boldsymbol{c}')}} \end{aligned} \tag{9}$$

Finally, the model is trained by minimizing the cross-entropy loss using the $\mathrm{softmax}$-weights (Eq. 9) of candidate scores as the predicted probabilities.

4 Experiments

4.1 Setup

Dataset We evaluate our architecture on two recently proposed benchmarks for reading comprehension. Both benchmarks require a system to answer a cloze-style query solely based on a single supporting document. Hermann et al. (2015) created two datasets (CNN, DailyMail) from news articles. For each article, queries were created from

their respective summaries by removing a named entity from the summary sentence that has to be predicted. All articles in the dataset are pre-processed by named entity recognition, co-reference resolution and entity anonymization. Similar in mind, Hill et al. (2016) created the *Children's Book Test* (CBT). For this dataset, passages of children's books of 21 sentences were extracted. Within the last sentence of the passage a word is removed that has to be predicted. The dataset is split into subtasks depending on the part-of-speech tag of the word that has to be predicted. We evaluate our model on the named entity (NE) subtask because it is the most challenging subtask for traditional language models.

Input Presentation & Encoding The input to the model consists of the context document and the query. The actual query is the cloze-text for the position of the removed named entity, which is replaced by a placeholder symbol. The entire input (document + query) is encoded by a bi-directional GRU from which the query and answer representations are computed as described in §3.2 and §3.3. Supporting (z, y)-pairs are extracted at occurrences of an answer candidate (all entities for CNN/DailyMail, given for CBT) in the context document in form of cloze-text (query) and its corresponding filler (answer).

Training For all experiments, we use a hidden dimension (and embedding-size) of $h = 256$. We train models with and without pre-trained word vectors. The input embedding matrix E_i is partially initialized with 100-dimensional `Glove`-embeddings (Pennington et al., 2014) and randomly for the rest (156 dimensions) when using pre-trained word vectors. In general, embeddings are initialized with a Gaussian of 0-mean and 0.1-stddev, matrices as described in Glorot and Bengio (2010) and biases with 0, except for the encoder GRU update-gate bias which is initialized with 1. Dropout is applied with a rate of 0.2 to the embedded input words for regularization. We train our system using mini-batch SGD with ADAM (Kingma and Ba, 2015) for optimization using an initial learning-rate of 0.001 that is halved whenever the accuracy on the development set drops between checkpoints and the the first entire epoch has passed. If the accuracy drops between entire epochs training is stopped. The mini-batch sizes/respective checkpoint iterations are 128/500 for the DailyMail and CNN datasets, and 32/1000 for the CBT NE dataset. Note, that similar to Chen et al. (2016), we do not consider all words but only entities as answer candidates for the CNN/DailyMail dataset. Our models are trained with 4 and 8 consecutive support retrieval cycles (hops) from the support, which performed better than using only 1 or 2. All models were implemented with TensorFlow (Abadi et al., 2015).

4.2 Empirical Results

The results of our model (QANN) on the two benchmarks are presented in Table 1a. They show that our model outperforms current state-of-the-art results for single models on the CBT NE, CNN and DailyMail datasets.[3] An important observation is that our model outperforms the Memory Networks by a large margin. Even with self-supervision, which explicitly introduces a training objective for selecting the correct memory at each hop, Memory Networks are clearly outperformed by our system. We attribute this to the query-answer representation of our supporting memory and the related architectural changes that separate end-to-end Memory Networks from QANNs.

For our models we observe that using 4 instead of 8 support retrieval cycles (hops) makes a difference only if `Glove` is not used for initialization. Using `Glove` to (partially) initialize embeddings gives a boost in performance on all datasets.

We would like to point out that the only systems with comparable results to ours use either the attention-weights over context-words (Iterative Attentive Reader) or the retrieved hidden state (Stanford Attentive Reader) to predict the final answer. Both works follow a similar idea as this work which separates the answer that is used for prediction from the query.

The Iterative Attentive Reader (Sordoni et al., 2016) alternates between attention on specific parts of the query and attention on the context document. They found that attending over the query is very useful, however, the attention is usually set on the placeholder symbol which is similar in our approach.

[3] We excluded results obtained with model ensembles for a fair comparison.

Model	CBT NE	CNN	DailyMail
Attentive Reader†	-	63.0	69.0
Stanford Attentive Reader (Glove)♦	-	72.4	75.8
Multi-hop Systems			
Impatient Reader†	-	63.8	68.0
MemNNs (window)✸	49.3	60.6	-
MemNNs (window + self-sup.)✸	66.6	66.8	-
Iterative Attentive Reader (8 hops)*	68.6	73.3	-
QANN (4 Hops)	69.4	72.6	76.6
QANN (8 Hops)	70.3	73.4	76.4
QANN (4 Hops, Glove)	**70.6**	**73.7**	76.9
QANN (8 Hops, Glove)	**70.6**	73.6	**77.2**

(a)

Hops	CBT NE	CNN
1	60.4	67.0
2	67.0	72.0
3	70.0	73.3
4	70.6	73.7
5	**71.2**	73.3
6	71.0	**73.8**
7	70.7	73.2

(b)

Table 1: (a) Accuracies of different models on 3 benchmark test datasets for reading comprehension. Hermann et al. (2015) †, Chen et al. (2016) ♦, Hill et al. (2016) ✸, Sordoni et al. (2016) *. (b) Accuracies of QANNs on the CBT NE and CNN testsets when employing a model trained with 4 support retrieval cycles, but applied on a varying number of retrieval cycles (hops).

They achieve similar results on the CNN dataset but are outperformed by our models on the CBT NE dataset. We attribute this improvement to our answer update mechanism (§3.5) which accumulates dedicated answer embeddings from support over multiple hops and from the original query through gates. Thus, the final answer prediction depends not only on the attention (support) weights, but also on the query itself and the answer embeddings. This advantage cannot be exploited as much in the CNN and DailyMail datasets because entities and thus all answers are anonymized. As illustration, Figure 2c provides an example of a correctly predicted answer that does not align with the computed support weights (see §4.4 for more details).

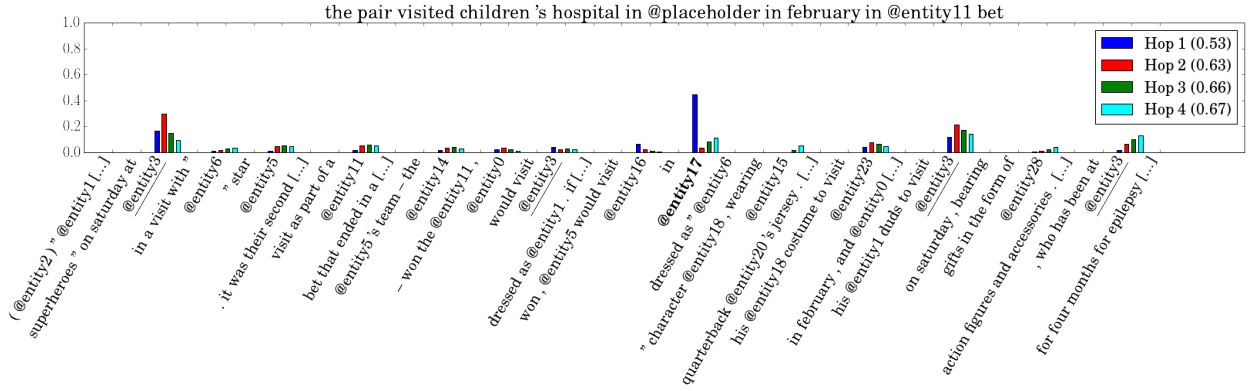
4.3 Impact of Multiple Answer Retrievals

We trained our models with different numbers of support retrieval cycles (hops). We found that using at least 4 hops leads to a significantly better performance than using only 1 or 2 hops. This indicates that multiple consecutive support retrievals and respective query updates are important for a robust performance on the reading comprehension tasks.

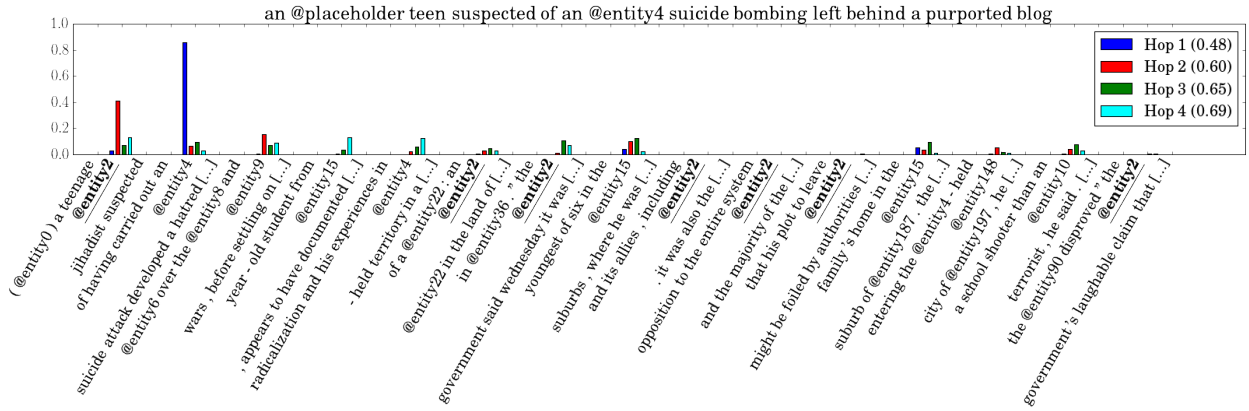
In addition, we evaluate the differences in performance when varying the number of hops when using our model trained on 4 hops. This evaluation gives insights into accuracy gains and the stability of answer prediction when increasing the number of hops. The results presented in Table 1b demonstrate that the model gains most until 3 hops, after which results are quite stable. The most pronounced difference occurs between using only 1 and 2 hops. The relative stability of performance between 3 to 7 hops indicates that the system learns to utilize the gating mechanisms which decide to keep or update the current query and accumulate answers successfully. Even though our model was trained with 4 hops, the best results for the CBT NE and CNN testsets were achieved when utilizing the model with additional hops (5 for the CBT NE and 6 for CNN). Surprisingly, for CBT we found a rather large improvement of about 0.6 percentage points in accuracy.

4.4 Analysis

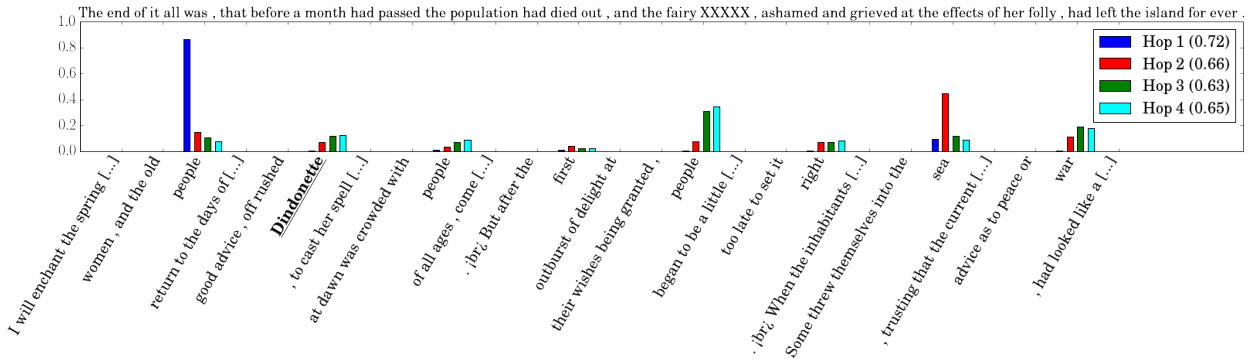
In a qualitative analysis of our system on sampled documents from the CNN and CBT NE dataset, we found that the correct answer is retrieved already after the first hop and kept until prediction in most cases (64% on 1000 sampled CNN examples). However, there are interesting exceptions to this rule that are displayed in Figure 2. It shows example doc-



(a) CNN: Correct answer retrieved after first hop but wrong prediction after subsequent hops.



(b) CNN: Wrong answer retrieved after first hop but corrected in subsequent hops.



(c) CBT NE: Retrieved answers are wrong yet system predicts answer correctly.

Figure 2: Examples of support (attention) weights for each hop for models trained with fixed 4 hops. The legends show the activity of the respective answer gates for each hop in brackets. The predicted answer is underlined and the correct answer is displayed in bold-face.

uments with respective attention weights for supporting spans-of-interest (positions of answer candidates) in each hop. A general observation is that the support weights are very pronounced for the first hop and spread over an increasing number of positions with each additional hop. We find that highly weighted positions can vary significantly between hops which shows that the query is updated. As we have shown empirically in §4.3 this can have a positive effect (8.4%), see for example Figure 2b. However, sometimes this can also result in an incorrect prediction, although the answer was correctly found in the first hop (2.7%) as demonstrated in Figure 2a.

A very interesting example from the CBT NE validation set is displayed in Figure 2c. It shows that the system puts high support weights on different positions in the document, but never on the correct answer. Nevertheless, to our surprise the model predicts the answer correctly anyway. One explanation might be that the model has learned that general words like "people" or "sea" are not good answers for the CBT NE dataset (e.g., through answer embeddings with small norm). Another explanation is that the query itself (which is also used to form the final answer representation) puts a strong bias on the final answer. To test the latter hypothesis we set the query-gate of Eq. 5 to 0 which effectively removes the query representation from the predicted answer representation. We found that the prediction changed to "people" which can be explained by the support weights. This finding confirms our premise that the query is able to put a bias on the final answer, and that the use of the query itself maybe be beneficial for answer prediction.

5 Conclusion

We have presented a new type of neural network architecture for answering queries. It is end-to-end trainable and learns to utilize knowledge in the form of supporting query-answer pairs to infer the answer to a given query. It explicitly separates the query representation used for selecting support from the answer representation used for predicting the answer. Results on recently proposed benchmark datasets for the task of reading comprehension show that our model achieves better results compared to existing single-model systems. This shows that the idea of explicitly separating query and answer is important for tasks that involve answering queries.

Future work involves the extension of this architecture to be able to properly handle other kinds of queries, e.g., list-queries or queries expecting generated answers. We also believe that our architecture can be applied successfully to a variety of tasks in the area of information extraction.

Acknowledgments

We thank Thomas Demeester, Thomas Werkmeister, Sebastian Krause, Tim Rocktäschel and Sebastian Riedel for their comments on an early draft of this work. This research was supported by the German Federal Ministry of Education and Research (BMBF) through the projects ALL SIDES (01IW14002), BBDC (01IS14013E), and Software Campus (01IS12050, sub-project GeNIE).

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, Dan 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.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. In *ACL*.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. 2016. Associative long short-term memory. *arXiv preprint arXiv:1602.03032*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *NIPS*, pages 1819–1827.

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*, pages 1693–1701.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2016. The goldilocks principle: Reading children's books with explicit memory representations. In *ICLR*, volume abs/1511.02301.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. 2016. Text understanding with the attention sum reader network. *ACL*.

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

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Tim Rocktäschel and Sebastian Riedel. 2016. Learning knowledge base inference with neural theorem provers. In *AKBC (NAACL)*.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. 2016. Reasoning about entailment with neural attention. *ICLR*.

Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. 2016. Iterative alternating neural attention for machine reading. *arXiv preprint arXiv:1606.02245*.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *NIPS*, pages 2431–2439.

Wilson L Taylor. 1953. Cloze procedure: a new tool for measuring readability. *Journalism and Mass Communication Quarterly*, 30(4):415.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *NIPS*, pages 2692–2700.

Dirk Weissenborn. 2016. Neural associative memory for dual-sequence modeling. In *RepL4NLP (ACL)*.

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