Ask Me Anything: Dynamic Memory Networks for Natural Language Processing

Ankit Kumar Ozan Irsoy Peter Ondruska
Mohit Iyyer James Bradbury Ishaan Gulrajani Richard Socher
Corresponding author: richard@metamind.io
MetaMind
Palo Alto, CA

Abstract

Most tasks in natural language processing can be cast into question answering (QA) problems over language input. We introduce the dynamic memory network (DMN), a unified neural network framework which processes input sequences and questions, forms semantic and episodic memories, and generates relevant answers. Questions trigger an iterative attention process which allows the model to condition its attention on the result of previous iterations. These results are then reasoned over in a hierarchical recurrent sequence model to generate answers. The DMN can be trained end-to-end and obtains state of the art results on several types of tasks and datasets: question answering (Facebook's bAbI dataset), sequence modeling for part of speech tagging (WSJ-PTB), coreference resolution (Quizbowl dataset) and text classification for sentiment analysis (Stanford Sentiment Treebank). The model relies exclusively on trained word vector representations and requires no string matching or manually engineered features.

This paper is a shortened version of the original paper on Arxiv [1], which includes complete references and more details on experiments.

1 Introduction

Question answering (QA) is a complex natural language processing task which requires an understanding of the meaning of a text and the ability to reason over relevant facts. Most, if not all, tasks in natural language processing can be cast as a question answering problem: high level tasks like machine translation (*What is the translation into French?*); sequence modeling tasks like named entity recognition [2] (NER) (*What are the named entity tags in this sentence?*) or part of speech tagging (POS) (*What are the part of speech tags?*); classification problems like sentiment analysis [3] (*What is the sentiment?*); even multi-sentence joint classification problems like coreference resolution (*Who does "their" refer to?*).

2 The Dynamic Memory Network Framework

The DMN is a general modeling framework for asking questions over inputs. The goal of the DMN is to first compute a vector representation of an input, given a question, and to then generate the correct answer. It contains the following modules:

2.1 Input Module

The input module computes a useful representation of the inputs such that relevant facts can be retrieved later. Generally, the input module may be thought of as computing the intermediate steps of

I:	Jane went to the hallway.	I:	Peter's sister is called Isabelle.
I:	Mary walked to the bathroom.	Q:	What are the mentions?
I:	Sandra went to the garden.	A:	[[Peter] 's sister] is called [Isabelle] .
I:	Daniel went back to the garden.	I:	Peter's sister is called [Isabelle].
I:	Sandra took the milk there.	Q:	Is it coreferent with: [Peter's sister] is called Isabelle?
Q:	Where is the milk?	A:	yes
A:	garden	Q:	Is it coreferent with: [Peter] 's sister is called Isabelle?
I:	Everybody is happy.	A:	no
Q:	What's the sentiment?	I:	The answer is far from obvious.
A:	positive	Q:	In French?
Q:	What are the POS tags?	A:	La réponse est loin d'être évidente.
A:	NN VBZ JJ .		

Figure 1: Example inputs and questions together with answers generated by dynamic memory networks trained on these questions. Note that a similar learning algorithm is used but different weights are triggered for different types of questions.

a function that eventually returns a final vector representation. The input module sends these intermediate values to the episodic memory module, which will complete the computation, conditioned on the question, by way of its attention mechanism. We refer to the representations that the input module provides to the episodic memory as *facts* to be reasoned over by the episodic memory.

In natural language processing, we have a sequence of T_I words $w_1^I, \ldots, w_{T_I}^I$. The input module computes the hidden states of a recurrent sequence model [4].

We use Glove [5] vectors to capture context-independent representations, and initalize the embeddings of the DMN with these values. Word embeddings are given as inputs to the recurrent network to compute hidden fact states: $c_t = \texttt{SEQ\_MODEL}(L[w_t^I], h_{t-1})$, where L is the embedding matrix and w_t^I is the tth word of the input sequence. In particular, we use a gated recurrent network (GRU) [6, 7].

If we subsample the output of the recurrent network, the input module will return just the hidden states c_t that correspond to end-of-sentence markers in the original story. Otherwise, the input module returns hidden states c_t for all words. For notational convenience, in future modules we will refer to either of these states as c_t, and say in the experiments section whether or not the subsampling was used.

2.2 Semantic Memory Module

The semantic memory consists of (i) stored word concepts and (ii) facts about them. We initialize embeddings to Glove vectors as described above. This module could include gazeteers or other forms of explicit knowledge bases, but in this work we do not use them.

2.3 Question Module

This module maps a question into a representation that can then be used for querying specific facts from the input module. We have questions that consist of sequences of T_Q words w_t^Q. We compute a hidden state for each via $q_t = GRU(L[w_t^Q], q_{t-1})$, where the GRU and embedding weights are shared with the input module. The final question vector is defined as $q = q_{T_Q}$.

2.4 Episodic Memory Module

The episodic memory module retrieves facts from the input module conditioned on the question. It then reasons over those facts to produce a final representation that the answer module will use to generate an answer. We refer to this representation as a *memory*. Importantly, we allow our module to take multiple passes over the facts, focusing attention on different facts at each pass. Each pass produces an *episode*, and these episodes are then summarized into the memory. Endowing our module with this episodic component allows its attention mechanism to attend more selectively to specific facts on each pass, as it can attend to other important facts at a later pass. It also allows for a type of transitive inference, since the first pass may uncover the need to retrieve additional facts.

For instance, in the example in Fig. 2, we are asked *Where is the football?* In the first iteration, the model ought attend to sentence 7 (*John put down the football.*), as the question asks about the

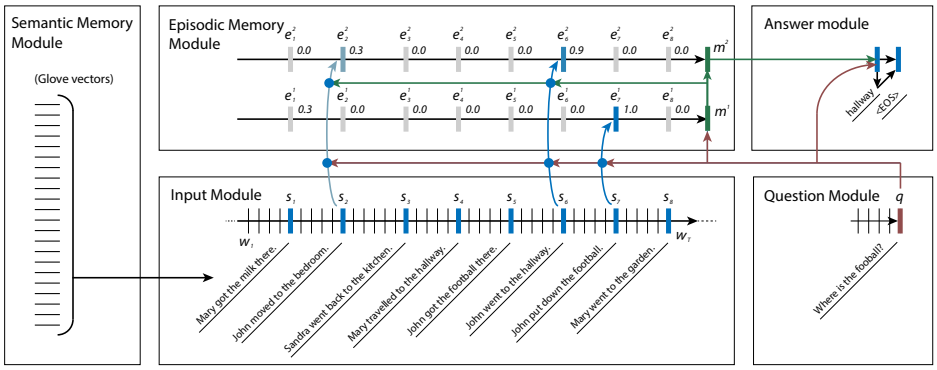

Figure 2: Real example of an input sentence sequence and the attention gates that are triggered by a specific question. Gate values g_t^i are shown above the corresponding vectors. The gates change with each search over inputs. We do not draw connections for gates that are close to zero. See Section 4.1 for details on the dataset that this example comes from.

football. Only once the model sees that John is relevant can it reason the second iteration should retrieve where John was. In this example, taken from a true test question on Facebook's bAbI task, this behavior is indeed seen. Note that the second iteration has wrongly placed some weight in sentence 2, which makes some intuitive sense, as sentence 2 is another place John had been.

In its general form, the episodic memory module is characterized by an attention mechanism, a function which returns an episode given the output of the attention mechanism and the facts from the input module, and a function that summarizes the episodes into a memory.

In our work, we use a gating function as our attention mechanism. It takes as input, for each pass i, a candidate fact c_t, a previous state m^{i-1}, and the question q to compute a gate: $g_t^i = G(c_t, m^{i-1}, q)$. The state is updated by way of a GRU: $m^i = GRU(e^i, m^{i-1})$, where e^i is the computed episode at pass i. The state may be initialized randomly, but in practice we have found that initializing it to the question vector itself helps; e.g, $m^0 = q$. The function G returns a single scalar and is defined as follows:

$$z(c, m, q) = [c, m, q, c \circ q, c \circ m, |c - q|, |c - m|, c^T W^{(b)} q, c^T W^{(b)} m] \quad (1)$$

$$G(c, m, q) = \sigma\left(W^{(2)} \tanh\left(W^{(1)} z(c, m, q) + b^{(1)}\right) + b^{(2)}\right) \quad (2)$$

To compute the episode for pass i, we employ a modified GRU over the sequence of T_C facts c_t, endowed with our gates. The episode is the final state of the GRU:

$$h_t^i = g_t^i GRU(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i, \quad e^i = h_{T_C}^i \quad (3)$$

Finally, to summarize the T_P episodes e^i into a memory, we use the same GRU that updates the attention mechanism's state: $m^i = GRU(e^i, m^{i-1})$, and we set the memory m as $m = m^{T_P}$. This is equivalent to setting the memory to simply the attention mechanism's final state, but we have described it here as its own computation to highlight the potential modularity of these subcomponents.

For datasets that mark which facts are important for a given question, such as Facebook's bAbI dataset, the gates of Eq. 2 can be trained supervised with a standard cross entropy classification error function. We also append a special end-of-passes representation to the facts, and stop the iterative attention process if this representation is chosen by the gate function. Otherwise, for datasets without explicit supervision, we set a maximum number of passes. The whole module is end-to-end differentiable.

Sequence Modeling

It is straightforward to apply the DMN to sequence modeling. In the sequence modeling task, we wish to label each word in the original sequence. Therefore, we desire one vector representation for each word. To this end, we run the DMN in the same way as above, once for each word. When running the DMN for word t, instead of setting the episode for pass i to the final state of our modified

GRU, we take the tth: e.g, for word t, we replace Eq. 3 with $e^i = h^i_t$. Note that the gates for the first pass will be the same for each word, as the question is the same. This allows for speed-up in implementation by computing these gates only once. However, gates for subsequent passes will be different, as the episodes are different.

The final output of the episodic memory module is the memory m, which goes to the answer module. In the sequence modeling case, each word's unique m is sent independently to the answer module.

2.5 Answer Sequence

The answer sequence module decodes the memory into a sequence of words representing the answer.

We use a GRU, and set the initial hidden state to the memory $a_0 = m$. The subsequent hidden states take as input the last hidden state and the previously predicted output y_{t-1}, as well as the question:

$$a_t \quad = \quad GRU([y_{t-1}, q], a_{t-1}), \qquad y_t = softmax(W^{(a)} a_t) \tag{4}$$

where $W^{(a)}$ is a standard softmax layer. The output is trained with the cross entropy error classification of the correct sequence appended with a special end-of-sequence token.. At test time, we generate words until an end-of-sequence is generated.

2.6 Training

Training is cast as a supervised classification problem to minimize cross entropy error of the answer sequence. For datasets with gate supervision, such as bAbI, we also include the cross entropy error of the gates into the overall cost.

2.7 Experiments

The DMN learning algorithm obtains state of the art results on question answering (babI), part of speech tagging (WSJ-PTB), sentiment analysis (Stanford Sentiment Treebank) and Coreference resolution (Quizbowl). Details are listed in the full version of this paper [1].

3 Conclusion

We believe the DMN is a potentially general model for a variety of NLP applications. The entire model can be trained end-to-end with one, albeit complex, objective function. The model uses some ideas from neuroscience such as semantic and episodic memories known to be required for complex types of reasoning.

References

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

[2] A. Passos, V. Kumar, and A. McCallum. Lexicon infused phrase embeddings for named entity resolution. In *Conference on Computational Natural Language Learning*. Association for Computational Linguistics, June 2014.

[3] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[4] J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2-3):195–225, 1991.

[5] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[6] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

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