- Independent Work Report, Spring 2017 -

Ask Me Otherwise: Synonym-Based Memory Networks for Reading Comprehension

Bharath Srivatsan
Advisor: Christiane Fellbaum

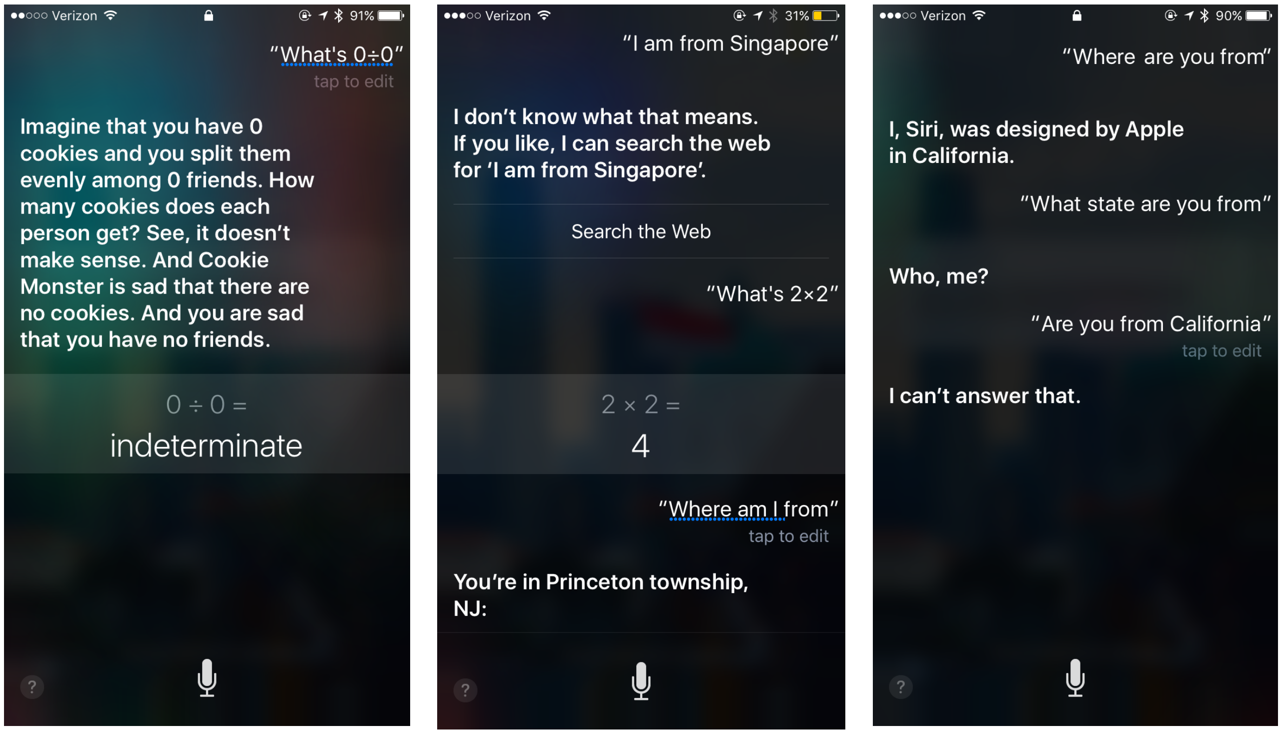
Abstract

Machine learning models built to read and comprehend text are becoming ever more complex. Driven by a desire for more accurate text understanding (whether to be used for generating conversational responses, taking actions, or even just storing knowledge), researchers have been iteratively improving on reading comprehension question-answering techniques. In this paper, I investigate one issue in this subdomain that many state-of-the-art approaches aren't resilient to: inferential jumps, in which questions being asked don't line up exactly with stored memories. I then develop a novel architecture, Synonym-Based Memory Networks (SynNets), that use WordNet and Word2Vec to better make such jumps from input documents when answering questions. In practice, applied to appropriately modified subsets of the bAbI and WikiQA datasets, I find that this model is indeed able to outperform an unmodified End-to-End Memory Network baseline.

1 Background

The past decade has seen an explosion of interest and research into natural language processing (NLP), the means by which machines interpret and interact with human-understandable language. Thanks to this relentless development, dialog agents have become increasingly advanced, tasked with doing anything from ordering groceries to telling party jokes. While heuristic or 'retrieval-based' agents once dominated this market, it is conversational machine learning that now powers millions of users' interactions with their devices via Siri, Alexa, Google, and Cortana (not to mention the wave of chatbots present on platforms from Facebook to WeChat). [1] This gradual change has been driven in large part by the complexities inherent in talking to humans in an open domain; old models that pulled pre-constructed answers from vast databases weren't resilient to the sheer variety of possible human questions. It seems that these efforts have largely paid off – today's AI assistants are *vastly* more sophisticated than those of yesteryear. [2]

All this improvement notwithstanding, end-to-end dialog agents still suffer from a host of seemingly simple problems; a plethora of pop culture references to 'Siri fails' have developed around the assistant's misunderstanding of basic questions. [3] One interesting class of such failures are related to conversational agents' inability to intelligently retain relevant information. This most often manifests itself in the form of sharply divergent answers to similar questions (see Figure 1) or as repeated requests for previously disclosed information.



"What's 0÷0"

Imagine that you have 0 cookies and you split them evenly among 0 friends. How many cookies does each person get? See, it doesn't make sense. And Cookie Monster is sad that there are no cookies. And you are sad that you have no friends.

0 ÷ 0 =
indeterminate

"I am from Singapore"

I don't know what that means. If you like, I can search the web for 'I am from Singapore'.

"What's 2×2"

2 × 2 =
4

"Where am I from"

You're in Princeton township, NJ:

"Where are you from"

I, Siri, was designed by Apple in California.

"What state are you from"

Who, me?

"Are you from California"

I can't answer that.

Figure 1: While Siri has clever, canned responses ready for some queries (left), it sometimes is seemingly forgetful (middle) or wildly inconsistent across answers to closely related questions (right).

These problems persist *despite* recent advances in storing memories for machine learning. From image processing programs to translation bots to reading comprehension agents, there has been a glut of powerful ML models with powerful recollection abilities. That said, the nature of memory utilization, especially in the realm of question-answering, is often fairly shallow, relying on direct word matches between memorized phrases and posed questions. This explains some of the difficulty that Siri has in the third example from Figure 1 above; regardless of how good its memory *storage* is, without an ability to link those three questions together and *dynamically* remember them, it will never be guaranteed to respond consistently.

This disconnect between memory *retrieval* and dynamic memory *understanding* is what I term the 'inference jump'. Humans naturally extract far more information from a given sentence than the words in the sentence themselves. For example, when I announce that *I'm hoping to chow down on some naan*, I'm simultaneously implying that *I'm hoping to eat some naan*, *I'm hoping to chow down on some Indian food*, *I'm not hoping to eat Mexican food*, and perhaps even *I like Indian food*. Some of these translations may seem excessively simple, but their value shouldn't be understated: if I subsequently ask Siri for food options, I'd hope that the agent would be able to find Indian restaurants open soon near me.

The inference jump can also manifest itself in other contexts. Conversational agents often act and sound repetitive, cycling through the same responses to given queries. An intelligent use of dynamic, inference-enabled memory would be to aid in the construction of more varied responses. Such an AI might be able to answer the same question in different ways but with the same content, thereby better approximating the response pattern of a human being.

Sample Test Question

Read and Respond: "John Doe was born on a small street in Paris. His brown hair almost completely covered his forehead. It had been many moons since he'd cut it; he feared barbers."

1: Is John Doe's hair brown?
2: Was John Doe born in France?
3: What is John Doe scared of?

Figure 2: Questions 2 and 3 demonstrate the inference jump at work – machines attempting to answer them would need to make an inferential link (however small) between Paris and France and between 'feared' and 'scared of.' Correspondingly, state-of-the-art conversational agents would likely be able to answer Question 1 with ease but subsequently struggle on 2 and 3.

Being able to more intelligently combine memory retrieval with inferences is important beyond creating a more humanoid Alexa, though. Developing a more complete 'understanding' of text (reading comprehension) relies on this ability (see Figure 2). The advantages to training an inference-enabled, memory-possessing agent on a reading comprehension dataset are immense – it would be able to generalize its learning more broadly to accurately answer a far wider array of follow up questions.

The preceding paragraphs set up in vague terms the motivation for this paper: I hope to develop a machine learning model that is better able to make small-scale inferential jumps in reasoning when answering questions. To limit the scope of this problem, I focus exclusively on question answering on reading comprehension datasets, testing my model on questions that cannot be directly tied to exact phrases in the text. There are two inferential jumps I attempt to bridge with this new architecture:

1. Synonymic Relatedness (being afraid is being scared, chowing down on is eating, etc.)
2. Conceptual Entailment (a poodle is necessarily a dog, naan is a subset of Indian food, etc.)

The latter category includes geographical entailment as well (as in the Paris example above). The remainder of this paper is organized as follows. Section §2 surveys some past work done in this domain. I discuss my approach and implementation details in Section §3 and the rationale and challenges associated with this approach in Section §4. Section §5 details the datasets I use for my work, while Section §6 outlines my results on this data. Finally, I sketch some potential avenues for future work in Section §7 and conclude in Section §8.

2 Related Work

Broadly, there are three relevant areas of background research pertinent to my work. First, existing Question-Answering models were informative in guiding the possible approaches I wished to develop for this problem. Second, memory networks, learning models developed to solve various NLP-related tasks, served as the underlying architecture for my project, and represented the baselines that I wished to improve upon. Finally, I built upon other work done to improve the inferential reasoning capacity of machines, even if not directly applied to the NLP space.

2.1 Q-A Models

The basic task that Q-A machine learning models are faced with involves being fed information and told to answer follow-up questions on the content of these conversations. There are a variety of ways that researchers have attacked this problem, but the central intuition behind many proposed models is the need for a ‘storage mechanism’ and (sometimes) an ‘attention mechanism.’ The former is the way by which agents retain memories to be accessed when constructing answers, while the latter is the way by which agents identify the *particular* pieces of memory most relevant to a given question.

Most storage mechanisms consist of some encoding of textual inputs, whether separated into sentences, ‘windows’ (sliding groups of words), or some other form. Some memory structures are static, constructed during the initial document input, while others are dynamically modified through their use. For example, Joulin and Mikolov [4] build a Recurrent Neural Network (RNN) that can begin to keep track of past memories in a stack, but Xiong et al. [5] outperform that setup with what they call the Dynamic Coattention Network, an architecture that refines a question and input document simultaneously (and dynamically) using Long Short Term Memory networks (LSTMs).[1]

The ‘attention mechanism’ domain has been the focus of a vast amount of research in recent years. In the machine translation space[2], Bahdanau et al. [7] build a system called RNNSearch.

[1] For the purposes of this paper, I do not explore RNN and LSTM constructions in much depth, though it is worth mentioning that these neural network architectures are very commonly used in the NLP space (especially in conjunction with other modules). For more, see [6]

[2] Broadly, machine translation problems involve converting text in one format to another (often language to language). Techniques vary wildly, but many approaches use networks that recursively convert individual entities from the input. See §11.1 for more

This neural machine translation approach involves building a single RNN that is jointly tuned to refine both an encoder and decoder for relevant memories. In doing so, their key contribution is to move from a fixed-length source vector model to one that can soft-search for relevant pieces of source text, thereby allowing more nuanced retrieval of memories to pay attention to. Within the conversational ML space, Miao et al. [8] apply a similar technique to help LSTMs attend to particular input memories in their Neural Answer Selection Model. Yao et al. [9] do the same with a triple-nested RNN.

Finally, a few state-of-the-art approaches to conversational ML attempt to use reinforcement learning (RL) to improve. Li et al. (2016c) [10] and Li et al. (2017) [11] allow dialog agents to improve their own communications via reciprocal interactions with human 'teachers.'

2.2 Memory Networks

Memory networks (MemNNs) are a relatively novel class of methods used to "incorporate reasoning with attention over memory (RAM)." [12] By focusing on a select number of 'relevant' memories from a potentially massive input set, these networks can more effectively answer comprehension questions. The model, developed by Weston et al. [13], operates by training and running four component networks – an input feature map to convert incoming data, a generalization layer to incorporate the new inputs into memory, an output network to find the most relevant memories, and a response layer to convert the output to a textual output. When temporal information was relevant, Weston et al. modified the model to include features that tracked the relative time of each input fact's occurrence. Furthermore, baseline resilience to words in the question that hadn't appeared in the input (background/memory) text was built into the model by using a "bag-of-words" approach and by training with dropouts, or pretending that words were new *d*% of the time. With these modifications, memory networks were able to vastly outperform both RNNs and LSTMs from past papers [13].

The output layer, tasked with identifying *k* memories to pay attention to, was originally trained in a fully supervised setting. This meant that the model required datasets that not only gave answers to questions (to train the response layer), but that also preselected the most relevant memories for each answer. This was a level of training that could not be incorporated into RNNs or LSTMs, but that few datasets met. Sukhbaatar et al. [14] removed the need for hard attention, requiring supervision only at the final response, in their End-to-End Memory Network construction (MemN2N). While performance for this model is marginally worse than the standard memory network, this is vastly outweighed by the benefits of no longer needing full supervision. There are many additional styles of memory networks built upon these baseline models and used to solve various other types of problems – I describe some of them in section §**Error! Reference source not found.**.

The true success of these models on real-world datasets is (to some extent) in dispute. Bordes et al. [15] achieve "excellent performance" using memory networks trained on Freebase (a massive knowledgebase) to answer SimpleQuestions[3], and are even able to perform well on the Reverb question set *without* retraining their model. However, Kapashi and Shah [16] find that the memory network approach performs poorly on other real-world datasets, in fact underperforming a baseline LSTM model on the MCTest set.

2.3 Inferential Reasoning

There are two broad areas of inferential reasoning that are particularly relevant to my work. One is the subdomain of paraphrasing, which essentially focuses on training machines to identify or

[3] A popularly used dataset in this space

construct sentences that are paraphrases of one another. Berant and Liang [17] build two paraphrasing models: an associative approach and a vector-space approach, and train them on question-answer pairs. Meanwhile, Fader et al. [18] essentially take the converse approach by resolving multiple paraphrased versions of the same sentence into a global form.

The second relevant subdomain is logical inference, determining whether a candidate hypothesis can logically be inferred from a given statement. While the potential approaches to this problem are vast (including natural logic based architectures), most were out of the scope of what was achievable in my project timeframe. [19]

3 Approach: SynNets

To tackle the inferential jump problem, I develop a new machine learning model: the Synonym-based Memory Network (SynNet). SynNets are modified versions of End-to-End Memory Networks that incorporate different word vector representations to improve resilience to reading comprehension questions with inferential jumps. In this section, I describe the underlying MemNN and MemN2N models in further detail, and then outline the modifications I made when designing and implementing SynNets.

3.1 Memory Networks

As mentioned in §2.2, Memory Networks allow agents to intelligently read and write from a dynamic memory bank. The central intuition behind their creation is "to combine the successful learning strategies developed in the machine learning literature for inference with a memory component that can be read and written to." [13]

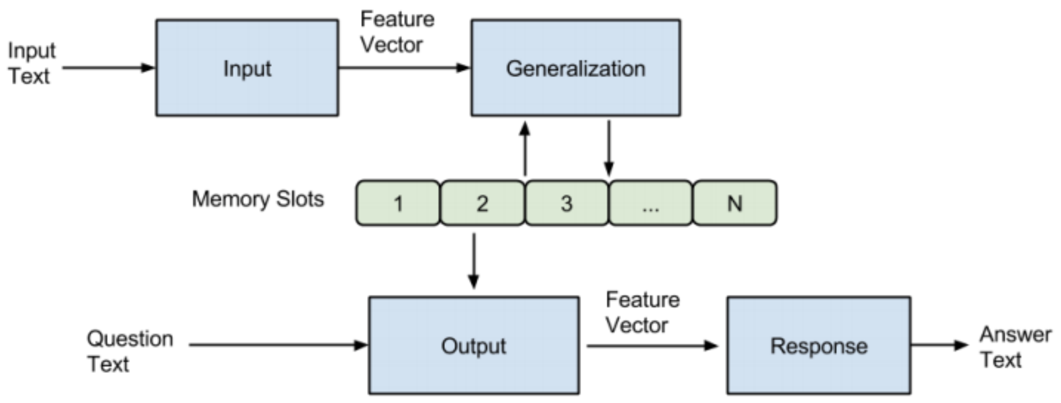


Figure 3: A top-level design diagram for the components that make up a Memory Network [20]

The networks themselves consist of four component layers – Input, Generalization, Output, and Response. In greater detail:

1. Input Feature Map: Converts the input text (ie. the background reading) into the internal feature representation. This typically involves preprocessing (tagging, parsing, etc.) the sentences and translating the inputs to vector form
2. Generalization: Update the memory storage based on the input provided. This could be as simple as storing the converted input in a slot, or could involve more complex organization/periodic forgetfulness measures
3. Output Feature Map: Given a question and the state of the model's memory, produces an internal representation of the output. This is typically done in two stages: first, iterating through the memories to find the most relevant ones to the input question, and second, generating an output vector based on this subset of memories and the input question
4. Response: Converts the output feature representation into the desired response format. This could be as simple as an RNN that is trained to build a textual answer from a given vector

3.2 End-to-End Memory Networks

To remove the need for high-granularity datasets (see §2.2), Sukhbaatar et al. built a version of the baseline memory network architecture that is only trained at the response stage, the End-to-End Memory Network. [14] Vanilla memory networks used 'hard' functions like *argmax* that weren't differentiable, meaning that the model as a whole could not back-propagate feedback from output to input. This new architecture, by contrast, reads from memory with *soft* attention, allowing for such feedback transmission. Further, unlike the regular memory network architecture, the end-to-end model 'hops' through the memories when searching for relevance, incorporating previously selected memories when picking out new ones.

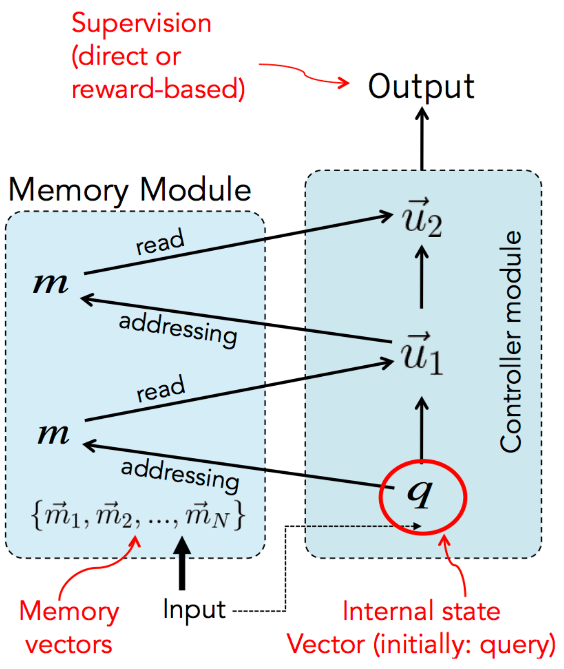

Supervision (direct or reward-based)
Output
Memory Module
read
m
addressing
$\vec{u}_2$
$\vec{u}_1$
read
m
addressing
q
Controller module
$\{\vec{m}_1, \vec{m}_2, ..., \vec{m}_N\}$
Memory vectors
Input
Internal state Vector (initially: query)

Figure 4: A design overview for the end-to-end memory network architecture. Note that supervision is only performed at the output end. [21]

In Figure 4 above, we see an overview of this process. The model makes two 'hops', addressing and reading from memory twice, before converting the final vector $\overrightarrow{u_2}$ to the returned output. Figure 5, below, demonstrates an individual hop in more detail, outlining the specific modules and operations used.

The Synonym-based Memory Network (SynNet) is built from the end-to-end memory network architecture as opposed to the vanilla memory network, given an obvious desire to allow for applicability to a wider range of datasets by minimizing required granularity. I proceed to outline the components of this network and my implementation of this construction in more detail.[4]

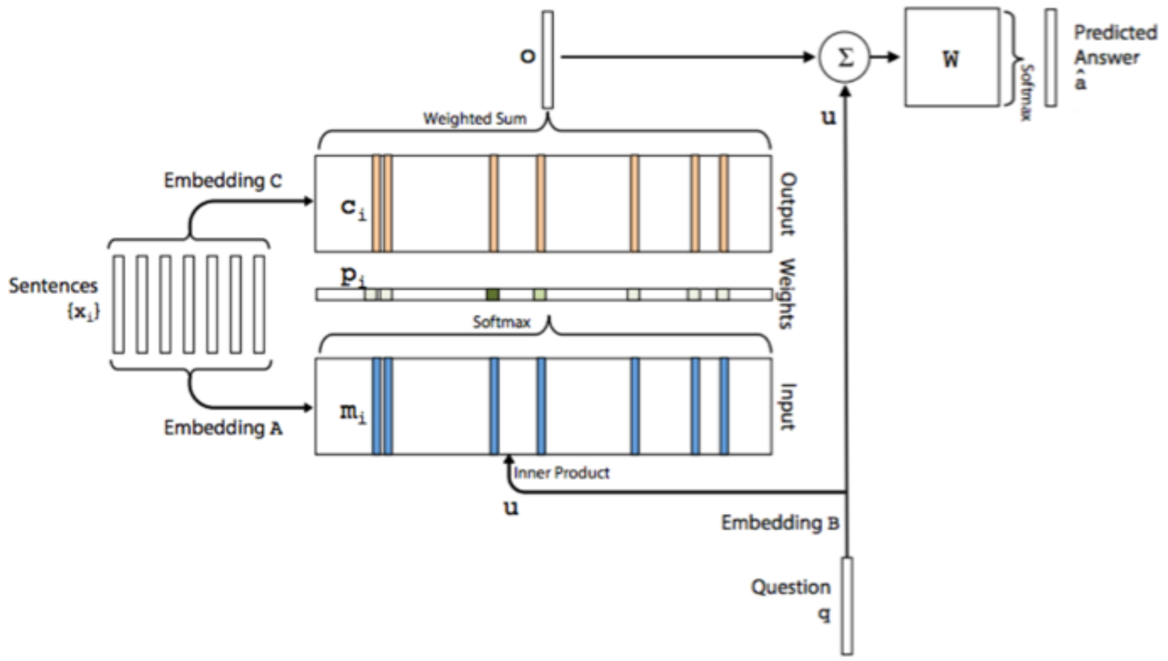


Figure 5: A more intricate picture of the internal workings of the architecture. This picture represents a 'one hop' approach; Figure 4 essentially is an abstraction of three such modules rolled together. [14]

[4] As mentioned in section §4.2.1, this construction is based off of the Matlab code open sourced by the Facebook AI Research team as well as Python scripts written by user Vinh Khuc

For any given task[5] we begin with: input (background) text represented as a series of sentences $\{x_1, x_2, \dots, x_i\}$, a question q, and an answer a that can be another sentence, a word, or a reference to one of the input sentences.[6] These inputs are preprocessed: sentences are converted to lower case, tokenized, and iterated through to build a dictionary of the D words involved in the stories.

3.2.1 Input Feature Map

This stage stores all of the inputs not as text that is uninterpretable by the machine but as a series of operable vectors called memories. We represent the memory vectors internally using a vector embedding approach. To this end, two embedding matrices are built and trained: $A_{\{v \times D\}}$ and $C_{\{v \times D\}}$ (see Figure 5 above). These matrices, each with dimensions $v \times D$, are used to convert the input sentences $\{x_i\}$ into input and output memory vectors $\{m_i\}$ and $\{c_i\}$, each of dimension v. These memory matrices are stored as the input and output embedding in Figure 5. The specific vector embedding I use is called a 'one-hot' matrix, with 1s in the positions of hit words and 0s otherwise; memories are simply one-hot vectors collected over a sentence of at most 20 words.

<u>Input and Output Embedding:</u>

$$[m_1, m_2, \dots, m_i] = A_{\{v \times D\}} \times [x_1, x_2, \dots, x_i] \tag{1}$$

$$[c_1, c_2, \dots, c_i] = C_{\{v \times D\}} \times [x_1, x_2, \dots, x_i] \tag{2}$$

3.2.2 Generalization

Similarly, matrix $B_{\{v \times D\}}$ is used to convert the question sentence q into an input vector $\vec{u}$.[7]

[5] My approach involved training on thousands of such tasks and questions. Each independent story had its own background (input), but the model was trained jointly across all tasks
[6] The choice of answer format is dependent on the dataset instructions
[7] In subsequent hops, the question q is replaced with the internal representation of the output from the previous layer, $u_{\{k-1\}}$. See Figure 13 in Section §11.2 for more details on layering hops

Question Embedding:

$$\vec{u} = B_{\{v \times D\}} \times q \tag{3}$$

This input vector is then multiplied with the input embedding to form a probability vector (denoted by 'Weights' in Figure 5). This essentially represents using the question to establish which memories are most probabilistically likely to be relevant; there will ultimately be a probability assigned to each 'slot' of memory. For this operation, we use a *softmax* function that tracks the relative weights of each linear combination of memory and question.

Probability Vector Construction:

$$p_i = \text{Softmax}(u^T m_i) \tag{4}$$

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5}$$

3.2.3 Output Feature Map

To build the output vector $\vec{o}$, the model takes the sum of the output embeddings weighted by the probability vector for the current question. Here we essentially use the probability vector to identify which pieces of background will be most relevant in constructing the *output*.

Output Vector Construction:

$$\vec{o} = \sum_i p_i c_i \tag{6}$$

3.2.4 Response

Finally, to build a response, we use a different weight matrix $W_{\{v \times D\}}$ in conjunction with the input and output vectors $\vec{u}$ and $\vec{o}$. We sum the input and output vectors in order to essentially represent both the question and the memory generated by our hop(s), and weight/softmax this sum to once again establish the particular parts of the vector (ie. words) that are uniquely highlighted.

Response Generation:

$$\hat{a} = \text{Softmax}(W(\vec{u} + \vec{o})) \tag{7}$$

This output label (or sentence, etc., depending on the nature of the dataset) can then be compared to the actual answer a, and we can attempt to minimize a cross-entropy loss function between the two labels. Given a set of input stories, questions, and answers, my end-to-end architecture trains the four embedding/weight matrices: $A, B, C,$ and $W.$ Because these functions are smooth from input to output (ie. there aren't any hard max functions, etc.), backpropagation is possible through the model to iteratively improve the values of these matrices. Once the matrices are finalized, testing is simple: stories are embedded and questions are pushed through the model to yield predicted answers that can be cross checked for accuracy.

3.3 SynNets

Having discussed my implementation of the baseline MemN2N architecture, I now turn to the SynNet modifications. Broadly, the intuition behind these changes lies in the hope that by including information on word similarities, the model will be able to make small-scale inferential jumps. To do so, I implement two approaches, one that relies on WordNet and another that relies on Word2Vec.

3.3.1 WordNet

The first implementation of the SynNet architecture relies on WordNet, a "large lexical database of English," in which "nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept." [22] Synsets themselves can be linked to other synsets via hyponymy relationships, which connect more general synsets with more specific ones. For example, the synset for furniture and the synset for bed would be linked in this way. [22]

To incorporate this information into the baseline MemN2N architecture, I began to embed questions along with the synsets of their component words. When encoding the one-hot vectors for questions, I generated a list of words present within the current synset and its parent synset for each hit word and mapped these into the vector as well. I ignored words from synsets not already present in the dictionary D (as that would mean they were certainly not in the input memories). The implementation of WordNet I used was rolled into the Natural Language Toolkit (NLTK) package, provided for Python.

3.3.2 Word2Vec

The second implementation I incorporated involves Word2Vec, a model that represents words based on the terms they co-occur with. One of the most successful intuitions developed in NLP in the past few decades is the idea that a word's meaning can often be inferred from the company it keeps: to some extent, word co-occurrence can be a good predictor for word similarity. Word2Vec is a predictive model that is particularly effective and efficient at learning word vector embeddings that can represent such co-occurrence information in a dense form. [23], [24]

To include such information into my SynNet architecture, I removed the one-hot vector encoding of words from the models, replacing it with word vector representations returned by a pre-trained Word2Vec implementation. This way, SynNets would be able to more naturally see similarities between input questions and memories without having to rely on direct word matches.

4 Rationale and Challenges

In this section, I walk through various design decisions I made while building these networks. I begin by describing my rationale for choosing End-to-End Memory Networks as the foundational model for my project, then outline some of the challenges I ran into while implementing them.

4.1 Architecture Choice

I was spoiled for choice when deciding on an adequate baseline model for the SynNet architecture. From the r-nets and BiDAF ensembles that dominate the Stanford Question Answering Dataset (SQuAD) accuracy leaderboard to more traditional LSTM and RNN-based systems, there exist a huge variety of question answering models, each optimized for a huge variety of datasets. To choose a useful baseline, I developed a set of four criteria that were particularly relevant for my design and implementation process:

1. Accuracy – The model must demonstrate state-of-the-art performance on its datasets, whether measured in F1 scores, response accuracy, or some other metric
2. Efficiency – Due to limitations in my ability to execute expensive code, especially on large datasets, the module must be relatively efficient in both space and memory
3. Relevance – The model must be runnable on reading comprehension question-answering datasets. Ideally, it would be resilient enough to run on others as well. This was intended to filter out architectures that may be revolutionary in theory but have low accuracy within my specific sub-domain
4. Availability – At the very least, the model must be published about and discussed extensively online; ideally, an open-source baseline implementation in Python exists, as this would vastly simplify my implementation process

The results of my analysis are outlined in Figure 6.

Architecture	Accuracy	Efficiency	Relevance	Availability
LSTMs [6][25] : Long-Short Term Memory models are a series of chained components that each iteratively modify a 'memory vector'	**Poor:** Lower performance than most other models	**Neutral**: Largely contingent on implementation details	**Good:** There exist a huge variety of potential	**Good:** Python impls exist, including for Q-A problems specifically
RNNSearch [7]: This architecture consists of a bidirectional RNN encoder and a gated RNN decoder used for machine translation	**Neutral:** Better than base LSTMs, worse than MemNets	**Good:** Runnable given my constraints	**Poor:** Applied to machine translation, not reading comp.	**Poor:** implementation in original form not published
Neural Turing Machines [26]: Uses a read/write-able memory that is accessed by both content and address	**Neutral:** Untested here; memory too small(?)	**Good:** Runnable given my constraints	**Poor:** Not applied specifically to QA subdomain	**Good:** Implementations in various languages exist
Reaso Nets [27] : Uses RL and a multi-turn approach to outperform on machine comprehension benchmarks	**Good:** Outperforms all other attempts for SQuAD challenge	**Neutral:** Largely contingent on implementation details	**Good:** Works well on SQuAD problems	**Poor:** Code not found openly available
Memory Networks [13], [28] : Discussed at length in §3.1	**Good:** High accuracy on baseline bAbI tasks	**Neutral:** Runnable given my constraints; few datasets	**Good:** Directly applied to reading comprehension	**Neutral:** Original impl in Torch, but 3rd party in Python
End-to-End Memory Networks [14]: Discussed at length in §3.2	**Good:** High accuracy, but underperforms Memory Nets	**Good:** Runnable given my constraints	**Good:** Directly applied to reading comprehension	**Neutral:** Original impl in Matlab, but 3rd party in Python
Key-Value Memory Networks [29], [30]: To be discussed in §4.2.2	**Good:** High accuracy, especially from real documents	**Poor:** Unable to run on large datasets on local clusters	**Good:** Directly applied to reading comprehension	**Neutral:** Original impl in Torch, but 3rd party in Python

Figure 6: Assessment of potential baseline architectures for SynNets

I ultimately settled on using End-to-End Memory Networks for their decent performance on all four metrics. Though they marginally underperformed models like the regular Memory Network or Reaso Nets, they more than made up for this in terms of implementation ease – both because existing implementations were widely available and because they were runnable on less heavily annotated datasets.

4.2 Challenges

While developing this model, I ran into a variety of challenges that often derailed my work for days or even weeks at a time. Below, I outline some of the most salient examples, along with explanations of how I attempted to resolve them.

4.2.1 Interoperability of SynNet Components

To build SynNets, I needed access to various third-party packages like WordNet and Word2Vec. In practice, this meant that my SynNet models had to be implemented in Python; WordNet especially is distributed primarily for mainstream languages (Java, Python, etc.) as opposed to languages like Torch or Matlab. This was a potentially serious issue since the candidate baseline architectures – including all of the memory network distributions – were written in the latter category of languages.

I evaded this problem by seeking out and using third-party implementations of these architectures. Obviously, third-party code is more susceptible to transcription and implementation errors; I attempted to reduce this possibility by locally benchmarking these versions against the official open-sourced packages, and by poring over the code to identify any unusual modifications. The final MemN2N implementation that I used both ran effectively and accurately and seemed to stick to the implementation details outlined in [14].[8]

4.2.2 Key-Value Memory Network Efficiency

The initial approach I investigated for this project relied on modifying Key-Value Memory Networks (KV-MemNNs) instead of MemN2Ns. KV-MemNNs differ from end to end memory

[8] See §11.3 for a comparison between this version and the official Matlab implementation

networks in that they explicitly separate the input and output memory embeddings (see Figure 5 above) into 'keys' and associated 'values.' The authors of the paper stored bags-of-words and sentences as keys (just as they are in the input memory structure of the MemN2N), but then store the 'central word' from those bags as values, or entities. The rationale behind this change is that by storing only the central or most important words as the (output) values, the model would better be able to utilize memory information across hops. [29]

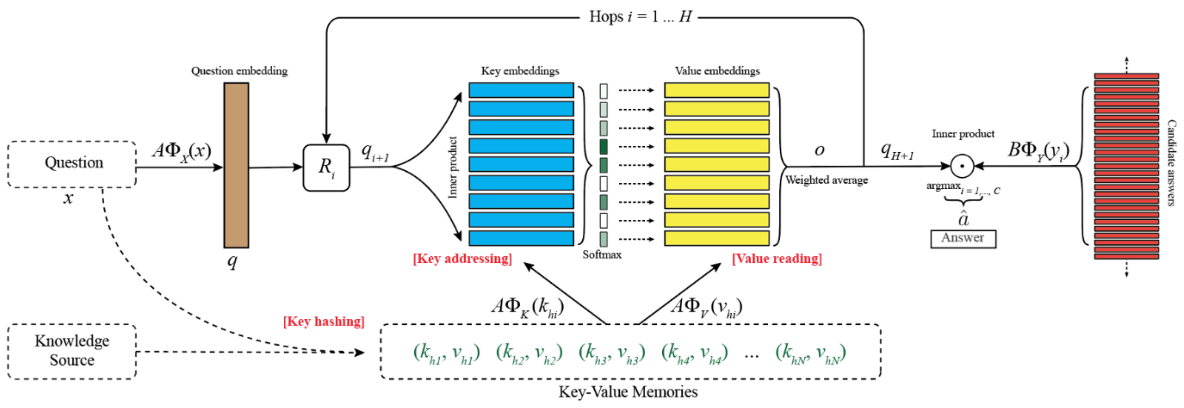


Figure 7: An illustration of the inner workings of the KV-MemNN architecture. Note that when the keys and values are the same form of representation of the knowledge source, the KV-MemNN reduces to a MemN2N [29]

In order to take advantage of the KV-MemNN's greater resilience to messy real-world documents (in the hopes that modifying from this architecture would allow SynNets to perform well on a wider array of datasets), I initially attempted to use KV-MemNNs as my baseline architecture. This approach, though, ended up failing for efficiency reasons – the third-party KV-MemNN code was unable to store the amount of data necessary to be able to run on large new datasets (like WikiQA or SQuAD). While this took up a large chunk of valuable time, I was able to quickly apply the insights I'd picked up while modifying this architecture when pivoting to adapt the MemN2N model instead.

4.2.3 Dataset Applicability

The final challenge related to developing the data for this project. Most reading comprehension datasets intend to test models on questions that are explicitly answerable, ie. both the answers and formulations of the questions are present within the input memories. For my task, though, I needed questions that were answerable from the text but not directly present in it. Rather than building a brand-new dataset using Amazon Mechanical Turk or the like, I decided to modify questions from existing Q-A datasets (see section §5). In order to reduce the likelihood that the method of modification would confound the success rates,[9] I made these changes manually, adapting patterns in the text to replace words in the vocabulary. This, though, drastically slowed my ability to modify the data and generate large scale testing sets (especially for a dataset as open ended as WikiQA).

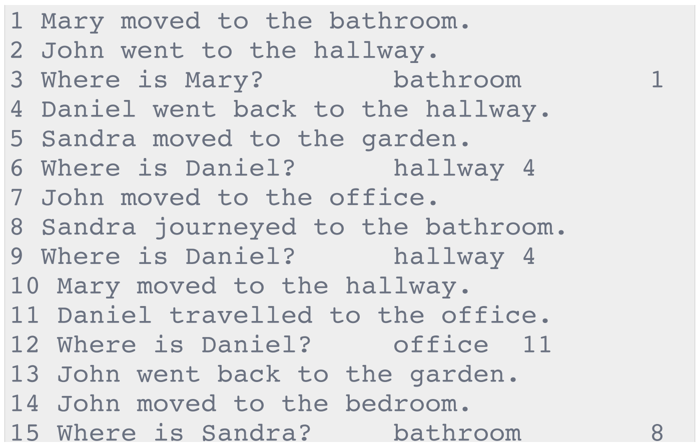
5 Data

In section §11.5, I outline various datasets often used in the domain of reading comprehension question answering. For the purposes of this project, I focused on the bAbI and WikiQA sets of questions.

5.1 The bAbI Tasks

The bAbI tasks, open sourced by the Facebook AI Research lab along with their work on memory networks, is a series of computer-generated 'stories' that are intermittently sprinkled with questions (see Figure 8). These stories are split into 20 tasks, each attempting to test the model on a different style of question (yes/no questions, temporal questions, positional questions, etc.). For each task, the bAbI dataset includes 1000 training questions and 1000 testing questions. [31]

[9] For example, if I had used a script to replace the words with elements from their synsets, I would trivially have been setting up the WordNet model to succeed.



```
1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary?        bathroom        1
4 Daniel went back to the hallway.
5 Sandra moved to the garden.
6 Where is Daniel?      hallway 4
7 John moved to the office.
8 Sandra journeyed to the bathroom.
9 Where is Daniel?      hallway 4
10 Mary moved to the hallway.
11 Daniel travelled to the office.
12 Where is Daniel?     office  11
13 John went back to the garden.
14 John moved to the bedroom.
15 Where is Sandra?     bathroom        8
```

Figure 8: An example of a story from the bAbI dataset. Notice the interspersed questions.

For the purposes of this project, I focused on tasks 2 (Two-Supporting Facts) and 6 (Yes/No Questions). I chose these tasks because their question styles contained common nouns that could be replaced in the testing set as needed. For each task, I modified the testing set by replacing common nouns in questions with parent or sibling nouns. For example, "*Is John in the hallway?"* became "*Is John in the corridor?", Is Mary in the garden?* changed to "*Is Mary outside?",* and *"Where is the apple?",* became *"Where is the fruit?"* I attempted to make modifications that fit both criteria described in §1: synonymic relatedness and conceptual entailment.

These changes had a drastic effect on the error rates of the traditional MemN2N: task 2 saw the mean error go from 8.17% (original) to 21.07% (modified), while task 6 saw errors jump from 9.88% (original) to 46.67% (modified).[10]

5.2 WikiQA

The WikiQA dataset is constructed from the real world – it involves the Wikipedia summary paragraphs associated with over 3,000 Bing user queries. These summary paragraphs are then

[10] Note that these spiking error rates also in a way validate the motivation for this paper; clearly, these state-of-the-art models have trouble dealing with even basic inferential jumps

broken up into sentences and labeled with 1s and 0s based on whether they answer the question at hand. [32] For example:

```
Q1      how are glacier caves formed?  D1      Glacier cave   D1-0   A partly submerged
glacier cave on Perito Moreno Glacier .       0

Q1      how are glacier caves formed?  D1      Glacier cave   D1-1   The ice facade is
approximately 60 m high        0

Q1      how are glacier caves formed?  D1      Glacier cave   D1-2   Ice formations in
the Titlis glacier cave        0

Q1      how are glacier caves formed?  D1      Glacier cave   D1-3   A glacier cave is a
cave formed within the ice of a glacier .     1

Q1      how are glacier caves formed?  D1      Glacier cave   D1-4   Glacier caves are
often called ice caves , but this term is properly used to describe bedrock caves that
contain year-round ice.        0
```

Figure 9: An example of a question with possible Wikipedia summary answers, annotated by success/failure.

This dataset required a vast amount of labor-intensive modification. For one, I needed to make further modifications to the SynNet architecture to be able to parse this new format. More pressingly, though, I needed to take further care in changing up the test set questions, as these were far more complex queries. All in all, I was able to modify around 150 of the questions, done in the same way as outlined in section §5.1.[11]

5.3 Hypotheses

I predict that my models will generally be more resilient to the modified test sets than the original MemN2N approach. By including synset and Word2Vec information, I hope that SynNets will be able to resolve the kinds of changes made (especially in the bAbI set). While I expect error rates on the bAbI tasks to be lower (based on its constructed nature and limited word set), I am curious to see which approach will be more successful. The WordNet setup is simpler and so may be less

[11] Note that since the training set is untouched by these modifications, the model training process shouldn't have been adversely affected by the smaller scale of the test data

resilient to modifications that aren't in the same synsets, but the Word2Vec setup may adversely affect the way by which the model is able to retrieve memories in the first place, thereby increasing error rates.

6 Results

I ran both SynNet modifications and the original MemN2N implementation on these three datasets, noting the final error rates for each. Each run consisted of 120 epochs (with a 20-epoch linear start) on a 3-hop MemN2N/SynNet. My results are presented in the following table, which compares each model's performance on these tasks. SynNet + WN represents the WordNet construction, while SynNet + WV represents the Word2Vec instantiation.

	MemN2N	SynNet + WN	SynNet + WV
bAbI			
Task 2	21.07%	12.03%	53.35%
Task 6	46.67%	25.65%	51.84%
WikiQA			
Subset of Questions	43.46%	32.21%	26.76%

Figure 10: Error rates for the different implementations on the modified datasets.

The data highlight some interesting conclusions (especially relative to my hypotheses). For one, the Word2Vec SynNet approach was inconsistent, achieving benchmark rates on the WikiQA subset but performing poorly on both bAbI tasks. There are a variety of potential explanations for this unpredictable performance; as I mentioned in §5.3, the drastic modifications made to the memory storage structure may have reduced retrieval efficiency. The WordNet construction was more consistent, outperforming the MemN2N implementation on each task. As expected, error rates were generally lower on the bAbI tasks than on the wikiQA subset.

7 Future Work

There are a variety of avenues for future work in this space. Most immediately, the intuitions behind SynNets could be applied to other reading comprehension models like Key-Value Memory Networks, Entity Networks, or deep LSTMs. By incorporating Word2Vec or WordNet similarity metrics into other architectures, we could better understand the value-add inherent to such modifications and continue to iteratively improve the state of the art on various Q-A tasks.

Future work should ideally also investigate exploring new styles of data. For one, running the model (with proper parallelization built in) on a modified version of the SQuAD dataset could give valuable insight as to whether SynNets are a viable solution to the inferential-jump problem in a much larger, real-world set. Similarly, SynNets could be applied to sentence completion benchmarks and the like, to establish whether their built-in resilience to word modifications are useful beyond the question answering subdomain. More broadly, researchers could attempt to use these insights to more directly take on the 'consistency in conversation' problems that were described in the Background section.

Finally, the true scope of SynNets needn't be limited to exploring resilience to questions that are but one 'word-similarity step' away. By incorporating information from knowledge bases like Freebase, or by training on more complicated inferential jumps, the baseline SynNet architecture could potentially be able to explore and tackle far broader domains of questions. Even more ambitiously, SynNets could be combined with state of the art models in the conversational ML space and be trained to only request specific information from knowledge bases when such inferential jumps are necessary. [33]

8 Conclusion

In this paper, I presented a novel machine learning model, the SynNet, that could help resolve the problem of inferential jumps in reading comprehension question answering. This approach was demonstrably effective relative to an End-to-End Memory Network baseline, but to differing degrees on the datasets they were tested on. I recommend that future work be done to refine this model further and apply it to even more complex domains.

Through this process, I received invaluable help from Professor Fellbaum, who guided me throughout and consistently pointed me in the right directions. I'd like to thank her for all of her support over the past semester.

9 Honor Code

This paper represents my own work in accordance with University Regulations.

Signed: Bharath Srivatsan

10 References

[1] J. Koetsier, "Alexa, Google, Siri, Cortana: 24.5M Voice-first Devices Will Ship This Year," *Forbes*. [Online]. Available: http://www.forbes.com/sites/johnkoetsier/2017/01/26/alexa-google-siri-cortana-24-5m-voice-first-devices-will-ship-this-year/. [Accessed: 01-May-2017].

[2] "Why Deep Learning Is Suddenly Changing Your Life," *Fortune*. .

[3] "/r/SiriFail," *reddit*. [Online]. Available: https://www.reddit.com/r/SiriFail/. [Accessed: 01-May-2017].

[4] A. Joulin and T. Mikolov, "Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets," *ArXiv150301007 Cs*, Mar. 2015.

[5] C. Xiong, V. Zhong, and R. Socher, "Dynamic Coattention Networks For Question Answering," *ArXiv161101604 Cs*, Nov. 2016.

[6] C. Olah, "Understanding LSTM Networks," 27-Aug-2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 28-Feb-2017].

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv Prepr. ArXiv14090473*, 2016.

[8] Y. Miao, L. Yu, and P. Blunsom, "Neural variational inference for text processing," in *Proc. ICML*, 2016.

[9] K. Yao, G. Zweig, and B. Peng, "Attention with intention for a neural network conversation model," *ArXiv Prepr. ArXiv151008565*, 2015.

[10] J. Li, A. H. Miller, S. Chopra, M. Ranzato, and J. Weston, “Learning Through Dialogue Interactions,” *ArXiv Prepr. ArXiv161204936*, 2016.
[11] J. Li, A. H. Miller, S. Chopra, M. Ranzato, and J. Weston, “Dialogue Learning With Human-In-The-Loop,” *ArXiv Prepr. ArXiv161109823*, 2017.
[12] CS224D, *CS224D Guest Lecture: Jason Weston - Lectures from 2015*. .
[13] J. Weston, S. Chopra, and A. Bordes, “Memory networks,” *ArXiv Prepr. ArXiv14103916*, 2014.
[14] S. Sukhbaatar, J. Weston, R. Fergus, and others, “End-to-end memory networks,” in *Advances in neural information processing systems*, 2015, pp. 2440–2448.
[15] A. Bordes, N. Usunier, S. Chopra, and J. Weston, “Large-scale simple question answering with memory networks,” *ArXiv Prepr. ArXiv150602075*, 2015.
[16] D. Kapashi and P. Shah, “Answering Reading Comprehension Using Memory Networks.”
[17] J. Berant and P. Liang, “Semantic Parsing via Paraphrasing,” *Assoc. Comput. Linguist. ACL*, 2014.
[18] A. Fader, L. Zettlemoyer, and O. Etzioni, “Paraphrase-Driven Learning for Open Question Answering,” *Assoc. Comput. Linguist. ACL*, 2013.
[19] B. MacCartney and C. D. Manning, “Natural Logic for Textual Inference,” in *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Stroudsburg, PA, USA, 2007, pp. 193–200.
[20] A. Coyler, “Memory Networks,” *the morning paper*, 10-Mar-2016. .
[21] “Memory Networks for Language Understanding, ICML Tutorial 2016.” [Online]. Available: http://www.thespermwhale.com/jaseweston/icml2016/. [Accessed: 03-May-2017].
[22] “About WordNet - WordNet - About WordNet.” [Online]. Available: https://wordnet.princeton.edu/. [Accessed: 02-Apr-2017].
[23] “Vector Representations of Words,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/word2vec. [Accessed: 05-May-2017].
[24] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” *ArXiv13104546 Cs Stat*, Oct. 2013.
[25] K. M. Hermann *et al.*, “Teaching Machines to Read and Comprehend,” *ArXiv150603340 Cs*, Jun. 2015.
[26] A. Graves, G. Wayne, and I. Danihelka, “Neural turing machines,” *ArXiv Prepr. ArXiv14105401*, 2014.
[27] Y. Shen, P.-S. Huang, J. Gao, and W. Chen, “ReasoNet: Learning to Stop Reading in Machine Comprehension,” *ArXiv160905284 Cs*, Sep. 2016.
[28] “facebook/MemNN,” *GitHub*. [Online]. Available: https://github.com/facebook/MemNN. [Accessed: 13-Feb-2017].
[29] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston, “Key-value memory networks for directly reading documents,” *ArXiv Prepr. ArXiv160603126*, 2016.
[30] “siyuanzhao/key-value-memory-networks,” *GitHub*. [Online]. Available: https://github.com/siyuanzhao/key-value-memory-networks. [Accessed: 03-May-2017].
[31] J. Weston *et al.*, “Towards ai-complete question answering: A set of prerequisite toy tasks,” *ArXiv Prepr. ArXiv150205698*, 2015.
[32] Y. Yang, S. W. Yih, and C. Meek, “WikiQA: A Challenge Dataset for Open-Domain Question Answering,” *Microsoft Res.*, Sep. 2015.

[33] J. E. Weston, “Dialog-based language learning,” in *Advances in Neural Information Processing Systems*, 2016, pp. 829–837.
[34] A. Bordes, Y.-L. Boureau, and J. Weston, “Learning End-to-End Goal-Oriented Dialog,” *ArXiv160507683 Cs*, May 2016.
[35] T.-H. Wen *et al.*, “A network-based end-to-end trainable task-oriented dialogue system,” *ArXiv Prepr. ArXiv160404562*, 2016.
[36] P.-H. Su *et al.*, “Continuously learning neural dialogue management,” *ArXiv Prepr. ArXiv160602689*, 2016.
[37] L. Shang, Z. Lu, and H. Li, “Neural responding machine for short-text conversation,” *ArXiv Prepr. ArXiv150302364*, 2015.
[38] A. Ritter, C. Cherry, and W. B. Dolan, “Data-driven Response Generation in Social Media,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Stroudsburg, PA, USA, 2011, pp. 583–593.
[39] O. Vinyals and Q. Le, “A neural conversational model,” *ArXiv Prepr. ArXiv150605869*, 2015.
[40] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau, “Building end-to-end dialogue systems using generative hierarchical neural network models,” *ArXiv Prepr. ArXiv150704808*, 2015.
[41] T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young, “Semantically conditioned lstm-based natural language generation for spoken dialogue systems,” *ArXiv Prepr. ArXiv150801745*, 2015.
[42] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan, “A diversity-promoting objective function for neural conversation models,” *ArXiv Prepr. ArXiv151003055*, 2016.
[43] A. Sordoni *et al.*, “A Neural Network Approach to Context-Sensitive Generation of Conversational Responses,” *ArXiv150606714 Cs*, Jun. 2015.
[44] J. Li, M. Galley, C. Brockett, G. P. Spithourakis, J. Gao, and B. Dolan, “A persona-based neural conversation model,” *ArXiv Prepr. ArXiv160306155*, 2016.
[45] S. Wang and J. Jiang, “Machine comprehension using match-lstm and answer pointer,” *ArXiv Prepr. ArXiv160807905*, 2016.
[46] C. N. dos Santos, M. Tan, B. Xiang, and B. Zhou, “Attentive pooling networks,” *CoRR Abs160203609*, 2016.
[47] M. Richardson, C. J. Burges, and E. Renshaw, “MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text.,” in *EMNLP*, 2013, vol. 3, p. 4.
[48] “karthikncode/nlp-datasets,” *GitHub*. [Online]. Available: https://github.com/karthikncode/nlp-datasets. [Accessed: 05-May-2017].
[49] T. Nguyen *et al.*, “MS MARCO: A Human Generated MAchine Reading COmprehension Dataset,” *ArXiv161109268 Cs*, Nov. 2016.
[50] D. Paperno *et al.*, “The LAMBADA dataset: Word prediction requiring a broad discourse context,” *ArXiv160606031 Cs*, Jun. 2016.
[51] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuAD: 100,000+ Questions for Machine Comprehension of Text,” *ArXiv160605250 Cs*, Jun. 2016.
[52] “QANTA: A Deep Question Answering Model.” [Online]. Available: https://cs.umd.edu/~miyyer/qblearn/. [Accessed: 05-May-2017].

11 Appendices

11.1 Literature Review: Conversational Machine Learning

In section §**Error! Reference source not found.**, I described approaches to Q-A problems specifically, performing an abridged literature review of some recent work in that subdomain. Here, I take a broader approach, outlining some of the most important papers in the conversational ML domain. I proceed as follows: first, I outline the three kinds of conversations that these systems attempt to model, describing seminal papers and relevant techniques in each. Then, I'll list some supplementary papers that work on the Q-A space.

11.1.1 Subdomains in Conversational ML

We can subdivide the domain of conversation-based machine learning tasks into three main categories: goal-oriented tasks, "chit chat" tasks, and Q-A tasks. Note, though, that these categories can often bleed into one another; chit-chat bots may also be expected to answer questions, and part of a goal-oriented conversational system might be engaging in chit chat. Also, I focus in the below discussion on machine learning-based approaches to these tasks, largely leaving aside purely heuristic or rule based systems.

Goal-Oriented tasks involve structuring a conversational assistant to help a user complete a set of jobs. For example, a chatbot that allows users to book movies or find recommendations for restaurants is "goal-oriented" insofar as it is aimed at helping fulfill a user goal. A single goal-oriented dialogue system can perform multiple functions (ie. order an Uber, make a reservation, etc.), but typically has a closed domain (a limited total set of functions that it can perform).

Many goal-oriented chatbots use a slot-filling approach to fulfilling tasks – they predefine a set of "slots" (location, cuisine, time, etc.) necessary for the relevant API calls (to Yelp, for example), and attempt to map parts of input sentences to these slots. This approach has its shortcomings, most centrally that user dialog may not follow this set of predefined patterns [34]. One approach used to avoid this problem uses a series of purposed neural networks. Wen et al. [35] train four neural networks in their dialog system: an LSTM for mapping vectorized input strings to intents, an RNN with a CNN feature extractor to build beliefs (strict mappings to be used in the database query) from intents, a policy network to construct database queries, and an LSTM to generate a conversational response based on the database results. Bordes et al. [34] use memory networks for this task instead, leveraging their ability to represent pieces of past conversation history to outperform a slot-filling baseline. Another approach to building goal-oriented taskbots incorporates reinforcement learning. RL in this context is sometimes used (to mixed success) in generating response dialogue, but more recently has been applied as a supplement to supervised learning of goal-fulfillment (in the style of AlphaGo's training) by Su et al. [36].

Chit-chat tasks require conversational agents to generate "natural" conversations with users, whether in a closed or an open domain. Due to the massive scope of possible use cases, techniques used in this sphere vary widely but can generally be split into retrieval-based, Statistical Machine Translation (SMT), and generative models. The first class refers to systems that 'retrieve' and modify appropriate responses from existing corpora. These are less effective in open domains that can include a huge variety of appropriate responses and that require context-sensitivity (ie. in response style) [37]. SMT models attempt to 'translate' input text to parallel forms of output text, for example by mapping "I'm slowly making this soup… and it smell gorgeous" to the response "I'll bet it looks delicious too!" [38]. Ritter, Cherry, and Dolan apply an SMT model on Twitter conversations, taking care to correct for the fact that responses don't *have* to be semantically equivalent to inputs and that the most statistically similar response phrases are often identical phrases or synonyms. This model outperforms a benchmark retrieval system [38].

The final class of chit chat models use neural training to build responses. While technically just extensions of SMT approaches, generative models typically incorporate contextual information, or otherwise move further away from direct translations. Shang et al. [37] build a Neural Responding Machine (NRM), training and evaluating it on a microblogging service reminiscent of Ritter's. By converting input strings to vectors and by using a probabilistic model to develop output vector strings, the NRM can generate responses that are relevant but that have completely different semantic structures and content from the inputs, something that Ritter's SMT model is unable to do. Vinyals and Le [39] build upon this approach, and propose an end-to-end solution that incorporates the sequence to sequence framework (seq2seq), which predicts each token in the output based on input tokens. Serban et al. [40] use a similar model – a Hierarchical Recurrent Encoder-Decoder (HRED) – to generate speech from past occurrences, applied on a movie dialogue dataset. Finally, Wen et al. [41] use an LSTM for this purpose instead. These methods perform substantially better than the baseline-STM and retrieval-based models.

In particular, three papers in this space bear highlighting for their relevance to my work. Li et al. (2016a) [42] recognize that neural conversational models often prioritize bland, generic responses like "I don't know" based on their frequency in training corpora. To combat this, her team abandons the seq2seq paradigm for a Maximum Mutual Information (MMI) objective function. This objective function uses pairwise likelihood as opposed to source-to-output likelihood as the basis for its predictions, but more broadly, shows the importance of choosing appropriate objective functions for these problems [42]. Sordoni et al. [43] extend the reach of these models backwards, by incorporating context beyond single-line inputs into a continuous context vector fed to RNN Language Models (RLMs). In this way, responses can be sensitive to a larger amount of relevant past information. Finally, Li et al. (2016b) [44] try to enforce consistency in a conversational agent's responses by embedding and incorporating a speaker/speaker-addressee 'style vector' in their seq2seq LSTM.[12]

11.1.2 Question-Answering Models:

The third style of task is the one that I focused on in this paper: *Q-A Models*. As mentioned previously, in this set of challenges, conversational agents are told to answer questions based on varying amounts of input information. I've included only *supplementary* literature below (taking care not to repeat information mentioned in §2.1).

Beyond Joulin and Mikolov's [4] stack-augmented net, Wang and Jiang [45] use a match-LSTM model to answer reading comprehension questions based on a given ('memorized') chunk of text. This approach is quite effective, since answers are often paraphrased pieces from the input text, and these models simply match pieces of relevant input premises to outputs. Abstracting their approach away from any specific implementation (making their mechanism work on both CNNs and RNNs), dos Santos et al. [46] propose a two-way attention system called Attentive Pooling (AP). AP allows paired inputs to be represented together regardless of respective lengths, allowing attention vectors to be computed for specific questions. This approach allows their team to outperform CNNs and bidirectional LSTMs trained without attention.

[12] To avoid the problem described in Li et al. (2016a), the team ranks potential responses using a scoring function that works from target to source

11.2 Additional Implementation Diagrams

I've included below other helpful figures and explanatory diagrams relevant for this paper.

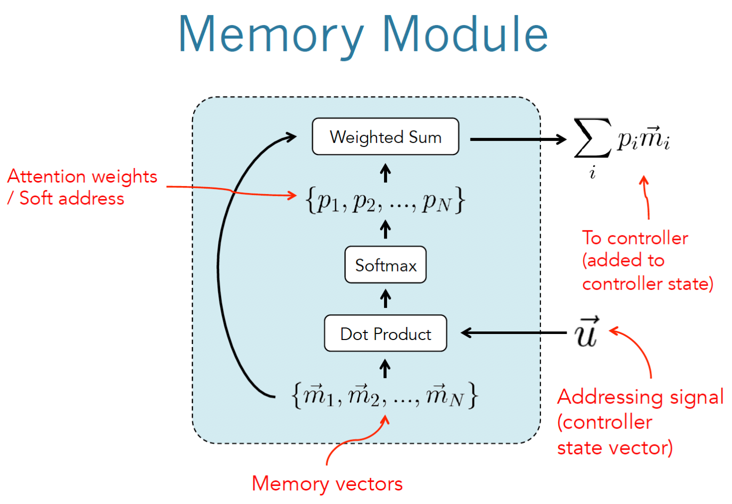

Memory Module
Weighted Sum
$\sum_i p_i \vec{m}_i$
Attention weights / Soft address
$\{p_1, p_2, ..., p_N\}$
Softmax
To controller (added to controller state)
Dot Product
$\vec{u}$
Addressing signal (controller state vector)
$\{\vec{m}_1, \vec{m}_2, ..., \vec{m}_N\}$
Memory vectors

Figure 11: This is a simplified representation of Figure 5, and more cleanly displays the process by which a given hop utilizes the memory module in answering questions. [21]

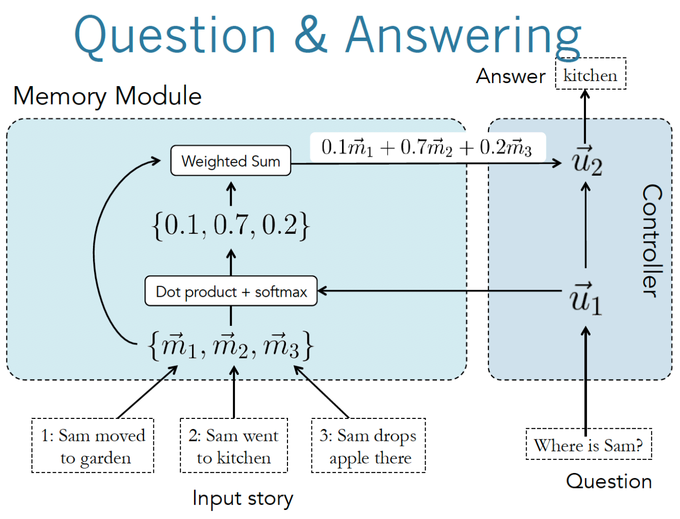


Figure 12: This is a replication of Figure 11, but with sample inputs and weights to further contextualize the process. [21]

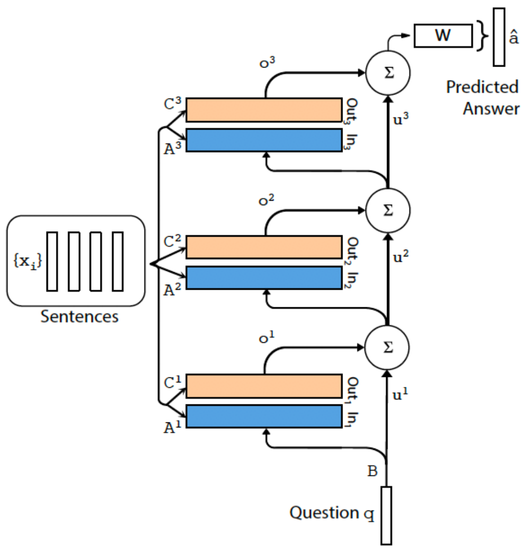


Figure 13: This is a clearer picture for the layering process of the end-to-end memory network architecture. As shown in the diagram, input and output vectors are summed to form the input vectors of the layers above. [14]

11.3 Implementation Comparison

As mentioned in Section §4.2.1, my implementation of the baseline end-to-end memory network was modified from an existing open source python repository (created by Vinh Khuc). Apart from Vinh's code, though, there also exists another widely-used implementation. This one, written by Dominique Luna, leverages TensorFlow in Python. Below, I've included a comparison of the self-reported task-by-task error rates for these two models and the original implementation, trained jointly on the bAbI tasks (lower is better). For each task, I've bolded the better error rate between the two third-party implementations.

Task	Original (Matlab)	Vinh Khuc (Python)	Dominique Luna (Python)
1	0	0.1	0.1
2	13.1	16.6	**15.1**
3	23.4	**26.3**	28.5

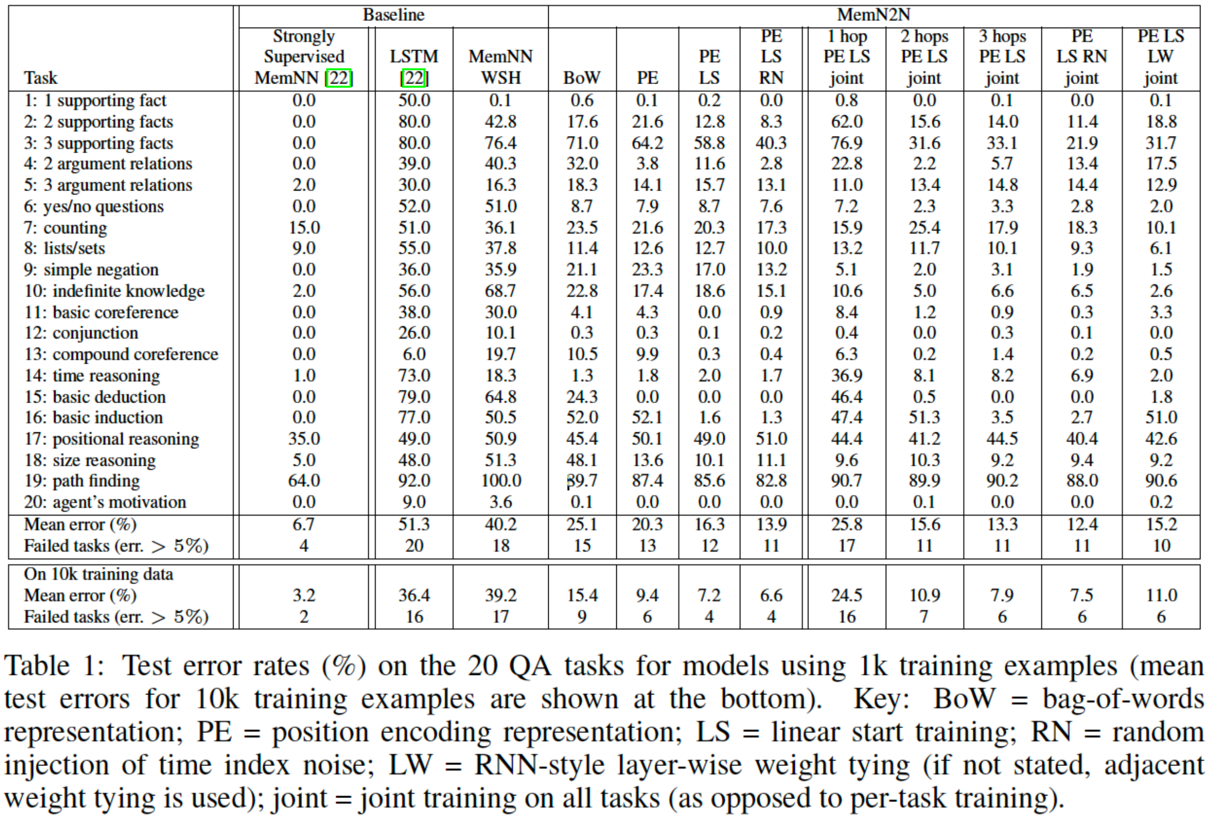
4	5.9	**<u>11.3</u>**	14.9
5	12.9	14.4	**<u>13.5</u>**
6	3.7	**<u>2.8</u>**	3.6
7	22.9	16	**<u>14.9</u>**
8	9.1	**<u>10.1</u>**	10.2
9	2.7	**<u>2.3</u>**	4
10	7.2	**<u>6.5</u>**	7.2
11	0.8	**<u>1.2</u>**	7
12	0.1	**<u>0.2</u>**	1.8
13	0.1	**<u>0.5</u>**	2.4
14	4.2	**<u>5.5</u>**	12.3
15	0	**<u>0.3</u>**	1.7
16	1.2	**<u>2.1</u>**	56
17	43.6	**<u>42.6</u>**	45.3
18	10.5	**<u>9</u>**	41.4
19	86.7	90.2	**<u>89.6</u>**
20	0	**<u>0.2</u>**	0.4
Mean	12.4	**<u>12.9</u>**	18.5

Figure 14: A comparison of baseline models

The Vinh Khuc implementation was preferred on almost all tasks, and has a significantly lower mean error rate. It approximates the MemN2N model accuracy rates on almost all tasks.

11.4 End-to-End Memory Network Accuracy Tradeoff

The following table, taken from [14], shows the accuracy tradeoffs inherent in the vanilla memory network versus end-to-end memory network comparison. As mentioned briefly earlier, the MemN2N model slightly underperforms the baseline MemNN implementation, though this can be overlooked for the massive reduction in dataset complexity needed.



	Baseline			MemN2N								
Task	Strongly Supervised MemNN [22]	LSTM [22]	MemNN WSH	BoW	PE	PE LS	PE LS RN	1 hop PE LS joint	2 hops PE LS joint	3 hops PE LS joint	PE LS RN joint	PE LS LW joint
1: 1 supporting fact	0.0	50.0	0.1	0.6	0.1	0.2	0.0	0.8	0.0	0.1	0.0	0.1
2: 2 supporting facts	0.0	80.0	42.8	17.6	21.6	12.8	8.3	62.0	15.6	14.0	11.4	18.8
3: 3 supporting facts	0.0	80.0	76.4	71.0	64.2	58.8	40.3	76.9	31.6	33.1	21.9	31.7
4: 2 argument relations	0.0	39.0	40.3	32.0	3.8	11.6	2.8	22.8	2.2	5.7	13.4	17.5
5: 3 argument relations	2.0	30.0	16.3	18.3	14.1	15.7	13.1	11.0	13.4	14.8	14.4	12.9
6: yes/no questions	0.0	52.0	51.0	8.7	7.9	8.7	7.6	7.2	2.3	3.3	2.8	2.0
7: counting	15.0	51.0	36.1	23.5	21.6	20.3	17.3	15.9	25.4	17.9	18.3	10.1
8: lists/sets	9.0	55.0	37.8	11.4	12.6	12.7	10.0	13.2	11.7	10.1	9.3	6.1
9: simple negation	0.0	36.0	35.9	21.1	23.3	17.0	13.2	5.1	2.0	3.1	1.9	1.5
10: indefinite knowledge	2.0	56.0	68.7	22.8	17.4	18.6	15.1	10.6	5.0	6.6	6.5	2.6
11: basic coreference	0.0	38.0	30.0	4.1	4.3	0.0	0.9	8.4	1.2	0.9	0.3	3.3
12: conjunction	0.0	26.0	10.1	0.3	0.3	0.1	0.2	0.4	0.0	0.3	0.1	0.0
13: compound coreference	0.0	6.0	19.7	10.5	9.9	0.3	0.4	6.3	0.2	1.4	0.2	0.5
14: time reasoning	1.0	73.0	18.3	1.3	1.8	2.0	1.7	36.9	8.1	8.2	6.9	2.0
15: basic deduction	0.0	79.0	64.8	24.3	0.0	0.0	0.0	46.4	0.5	0.0	0.0	1.8
16: basic induction	0.0	77.0	50.5	52.0	52.1	1.6	1.3	47.4	51.3	3.5	2.7	51.0
17: positional reasoning	35.0	49.0	50.9	45.4	50.1	49.0	51.0	44.4	41.2	44.5	40.4	42.6
18: size reasoning	5.0	48.0	51.3	48.1	13.6	10.1	11.1	9.6	10.3	9.2	9.4	9.2
19: path finding	64.0	92.0	100.0	89.7	87.4	85.6	82.8	90.7	89.9	90.2	88.0	90.6
20: agent's motivation	0.0	9.0	3.6	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.2
Mean error (%)	6.7	51.3	40.2	25.1	20.3	16.3	13.9	25.8	15.6	13.3	12.4	15.2
Failed tasks (err. $> 5\%$)	4	20	18	15	13	12	11	17	11	11	11	10
On 10k training data												
Mean error (%)	3.2	36.4	39.2	15.4	9.4	7.2	6.6	24.5	10.9	7.9	7.5	11.0
Failed tasks (err. $> 5\%$)	2	16	17	9	6	4	4	16	7	6	6	6

Table 1: Test error rates (%) on the 20 QA tasks for models using 1k training examples (mean test errors for 10k training examples are shown at the bottom). Key: BoW = bag-of-words representation; PE = position encoding representation; LS = linear start training; RN = random injection of time index noise; LW = RNN-style layer-wise weight tying (if not stated, adjacent weight tying is used); joint = joint training on all tasks (as opposed to per-task training).

Figure 15: Accuracy rates for MemN2N and MemNN models

11.5 Dataset Comparison

Reading comprehension datasets generally come in one of two flavors. *Cloze* sets expect an agent to fill in a blank word or sentence based on a collection of words that came prior. Q-A sets, by contrast, ask models to formulate a specific response (yes/no, a word, a sentence, etc.) to an explicit question asked about information that came prior. In this section, I provide a rough outline of freely available, simple, open-ended[13], and high-quality reading comprehension Q-A datasets.[14] Obviously, though, Q-A sets can be converted to cloze sets (and vice versa) by representing questions as the final sentence in the set upon which to generate new content (or by converting the final sentence to a question).

Dataset	Description	Source	Notes for Project
bAbI	20 tasks for testing text understanding and reasoning	Computer-generated	Used in project (discussed above)
WikiQA	A Challenge Dataset for Open-Domain Question Answering	User Logs	Dataset hard to modify for project scope; involved mappings that were incredibly time consuming to do manually. Subset used in project
SQuAD	100,000+ Questions for Machine Comprehension of Text	Human-generated	Dataset quite large, in a format that was difficult to parse for project given variance in sentence length, etc.
QuizBowl	A Neural Network for Factoid Question Answering over Paragraphs	Human-generated	Involved a lot of extraneous detail, like position of buzzing, etc., that was out of my project's scope

Figure 16: Dataset details

Information used in this table: [32], [49]–[52]

[13] This condition excluded multiple-choice datasets like MCTest and factoid creation datasets like SimpleQuestions [15], [47]

[14] For cloze datasets and datasets outside of the reading comprehension subdomain, see [48]