Solving SAT Reading Comprehension Questions with Memory Networks

Saahil Madge
Adviser: Professor Christiane Fellbaum

1. Background and Motivation

Machine question-answering (QA) has been a popular subject in recent research. It is a very interesting field from an academic perspective, requiring knowledge and techniques across the areas of artificial intelligence, machine learning, natural language processing, linguistics, and mathematics.

QA has applications to many aspects of daily life. A famous use of question-answering techniques is in Apple's Siri program. Siri combines speech recognition software with question-answering techniques to create a virtual iPhone assistant. Another example is IBM's Watson, which answered questions in many subjects and performed quite well as a contestant on the game show *Jeopardy!* A less spectacular but more commonly-used example can be seen by typing in a simple question such as *How many calories are in an apple?* on Google. The search results provide a detailed nutritional breakdown of an apple.

The general term QA covers a broad array of subfields. Information retrieval, factual question-answering, reading comprehension, and inference are all part of the greater question-answering area. QA is a rapidly evolving field, but as the examples above indicate, recent research has focused extensively on answering spoken questions, and factual questions.

However, the subfield of machine comprehension has not been studied nearly as well, though it has become quite popular in the past few years. Machine comprehension involves training programs and systems to read text and answer questions based on the newly acquired information. In most other QA tasks, the system has a large database of facts (knowledge base) and must interpret the question and provide the relevant factual answer. Machine comprehension tasks, on the other hand,

require the system to parse both the informational text as well as the question, and then provide the correct answer. As there is possibility for error within the knowledge base, machine comprehension tasks are some of the most difficult QA tasks.

Machine comprehension has a wide range of applications. Reading comprehension tests are a natural choice, but fields like document retrieval also use machine comprehension techniques. Financial firms that use news analysis to make trading decisions heavily rely on these techniques as well. Many long-term goals of AI such as dialogue and humanoid robots cannot be achieved without significant advance in this area [20]. Another incentive to study machine comprehension is that new research here can be applied to drive research in many other areas of artificial intelligence and natural language processing.

As a consequence of machine comprehension tasks being so difficult, most research with reading comprehension tests has focused on elementary school level tests. The most common dataset is MCTest[16], which we discuss in more detail in 3. This dataset is a set of passages and reading comprehension questions created by crowdsourcing. The stories are "carefully limited to those a young child would understand"[16]. Another common dataset is Facebook's bAbi dataset[20], which has a collection of short toy problems that can be used to train machine comprehension systems. These problems could certainly be solved by a human elementary school student. Berant et al.[2] train a system to answer questions based on a paragraph describing biological processes, which is certainly an advanced topic, but the system is specialized for biological tasks and not applicable as is to general reading comprehension.

In this paper we focus on SAT reading questions. The major reason is that they are far more complex than MCTest questions and other commonly-used machine comprehension datasets. The sentence structures, vocabulary, and topics covered in SAT passages are all far more varied than in the other benchmarks. The questions are much harder as well, and simply understanding the question and the individual choices is difficult by itself. SAT questions often test underlying themes and broad ideas rather than particular factual details ("Why" or "How" questions). In contrast, MCTest questions are often of a "Who", "When", or "What" form, which have traditionally been

easier for programs to answer.

Another aspect that makes SAT questions so difficult is that they require inference to answer, as opposed to just matching words or syntax. MCTest and similar questions can mostly be answered through word-matching (discussed in 2.1.1) or via syntactic matching. Word-matching means the system will try to find which sentences in the passage have the most words in common with the query, and use that to choose an answer. Syntactic matching is the same principle, but aims to match syntactic structure. SAT questions, however, almost never repeat the answer words in the question itself, and the query structure is not correlated with passage structure. To solve these questions, our system has to really understand the high-level concepts and information that is implied but not directly on the surface. This is a significantly harder problem than has been tackled before, but it is necessary to solve to continue moving forwad in the field of machine comprehension. Essentially, by trying to answer SAT questions we can try to solve problems that will be present in real-world applications but have not yet been tackled by existing research.

Due to the difficulty of interpreting the text as-is, we represent the text using relational logic instead. Essentially we dynamically create a *Knowledge Graph* (covered in more detail in 2.1) as we parse through the text and try to frame each question as a query on this knowledge graph. The knowledge graph representation is typically used for factual data that has already been collected in entity-relation triple form. As far as we know, no previous research in this space has tried to dynamically create a knowledge graph on which we can answer questions.

Traditionally most research on relational learning has used either neural networks or tensor decomposition to answer queries on the graph. In this work we present an improved version of both methods. For tensor factorization we enhance a method known as RESCAL[1][14] with a technique known as *Semantically Smooth Embedding*[8]. RESCAL trains a factorization of the knowledge graph using Alternating Least Squares, and this factorization can then be used to answer queries.

The neural network approaches such as TransE[4] and TransH[19] use neural networks to train an embedding method for the relations on the knowledge graph, and use the trained model to answer queries. However, none of these previous approaches are designed for question-answering.

We enhance the Memory Networks architecture proposed by Weston et al.[21] and improved by Sukhbaatar et al.[18] to take knowledge graph triples and queries as input, and provide a scoring of the answer choices as output. We discuss memory networks and alternatives in much greater depth in 3 and 4.

Our goal with this research is to present a way to combine ideas that were previously used in isolation for QA or other learning tasks into a system which can parse and interpret a complex text, and answer difficult high-level questions about the text. The text, questions, and techniques we use are more ambitious than those studied in previous research. Our results should be interpreted as merely proof-of-concept. We hope that this paper will encourage others to tackle problems of equal or greater ambition, and drive progress in the development of safe artificial intelligence.

2. Technical Review

In this section we review some of the techniques that are used in previous research and in this paper.

2.1. Representation Techniques

We begin by reviewing various methods for embedding text and performing queries on the embedded representations.

2.1.1. Bag-of-words Bag-of-words is a type of analysis in which we treat each sentence as simply a collection of words, without paying attention to their order or the dependencies between individual words. If we have some vocabulary of size V, we can encode each sentence as a vector of size V where each vector element corresponds to some word in the vocabulary. If the word appears in the sentence we store how many times it appears, and if it does not appear we store it as a 0. This is known as *one-hot* vector encoding. Most designs encode each input sentence and the query as one-hot vectors. These vectors are not used as-is, but are first embedded into some dimension d. The learning algorithm is run on the embedded vectors, and we can output a response in a specified format. Two common methods are to return a probability vector of size V where each element represents how likely that word is to be the output, or to pick the most likely word from

that distribution and return only that word.

2.1.2. Dependency Parsing Dependency parsing is an important technique used to understand the structural information in a sentence. Essentially, dependency parsing creates a graph of the words in the sentence and describes how each word relates to other words. A sample graph is shown in Figure **??**. Each dependency in the graph has a *governor* and *dependent*. The collection of dependencies contains all the structural relationships in the sentence. Each dependency is labelled to show the exact relationship between governor and dependent. For example, in the graph in Figure **??** the label *nsubj* highlights the nominal subject of a particular clause in the sentence[7]. The specific parser we use is the Stanford High-Performance Dependency Parser by Chen and Manning[6].

Once the dependency graph is created, we can analyze it to extract the relevant information in the sentence. The exact details are discussed more in Section 4.1.2. The basic approach is to find the main subject of the sentence and either the direct object it acts on or some description of it. Then it can be turned into an entity-relation triple on the knowledge graph.

2.1.3. word2vec The *word2vec* embedding technique introduced by Mikolov et al. [12] is perhaps the most common embedding technique used in NLP research. It is actually an umbrella term for various methods, such as *skip-gram* or even bag-of-words embedding. The word2vec techinque produces a vector representation of words, often with several hundred elements. These representations can then be manipulated as vectors in the higher-dimensional space, allowing for interesting operations on words. For example, **vector("king")** - **vector("man")** + **vector("woman")** produces a vector that is very close to **vector("queen")**. The reason word2vec is able to embed so successfully is generally thought to be the fact that the process of embedding gives similar words a similar embedding (which is measured by cosine similarity).

2.1.4. Knowledge Graph A knowledge graph generally refers to a set of *entity-relation triples*. These triples take the form $\langle e_1, r, e_2 \rangle$. where e_1 and e_2 are entities and r describes the relation between these two entities. Entities and relations can be repeated across triples. The total set of

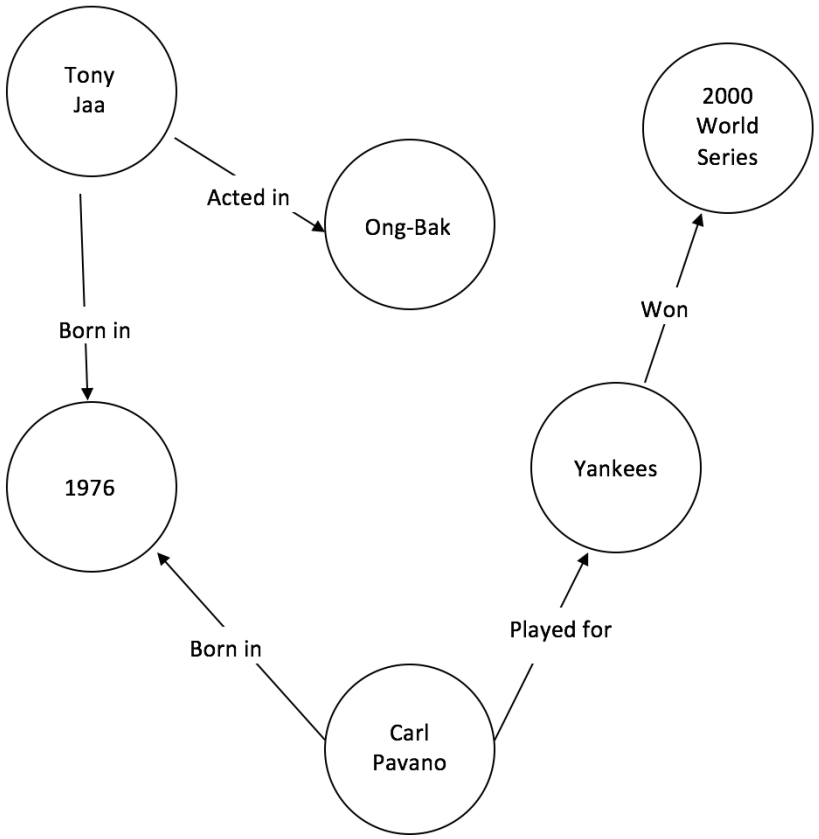

Figure 1: A small example knowledge graph

triples denotes all the knowledge in our "Knowledge Base".

We can represent these triples as a directed graph, with each entity as a vertex and the relations as edge types between them. Figure 1 shows an example of a very small knowledge graph. Note that there is only one node per entity. This allows the graph to store multiple pieces of information regarding the same entity together, to allow for easy processing by the program while also making it simple for humans to understand. The mathematical details of our knowledge graph model can be found in 4.

Once a knowledge graph is built, we can pose queries about the graph. For example, we can provide two entities e_1, e_2 and ask which relation r links the two on the graph. We can also provide an entity e and the relation r, and ask for which other entity e' do the triples $\langle e, r, e' \rangle$ or $\langle e', r, e \rangle$ exist on the graph. Put another way, the queries are either *link prediction* queries or *entity prediction* queries.

2.2. Neural Networks

As discussed in 1, recent research in machine comprehension has extensively used neural nets (NNs) and their variants. Here we provide a quick review of neural network design. These topics can be covered in far more depth in [3] and [15].

The most basic model is known as a *feed-forward* neural net. These have several layers composed of units (nodes). There is an initial input layer, a final output layer, and *hidden layers* in the middle. A unit in layer l takes as input the output of nodes in the previous layer (or the actual input if this is the input layer). It multiplies these by some weight matrix W^l, transforms it using an activation function g, and sends its output to the next layer (or as the final output if this is the output layer). Common activation functions are the *tanh* and *sigmoid* functions. The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. For now we will assume that our activation function is sigmoid.

Mathematically, we define the input to the current layer as x^{l-1}, where x_{ij}^{l-1} means that it is the input from unit j in the previous layer to unit i in the current layer. W_{ij}^l is the weight on this connection, defined as 0 if node i does not depend on node j. The output of node j is $\sigma(W^l x_j^{l-1})$. Typically the quantity Wx_j^{l-1} is known as z_j. We can also define bias vectors b such that $z_j = W^l x_j^{l-1} + b_j^l$. To summarize, we can write the output of some layer l in vector notation as

$$o^l = \sigma(W^l x^{l-1} + b^l)$$

We can propagate this forward starting at the input layer, through the hidden layers, and finally at our output layer. We can then train our network using the backpropagation algorithm. We have some cost function C that is a measure of the error of our output. We find the error at the output layer, and then change the weights at each previous layer, working backwards to the input layer. The key to backpropagation relies on updating the weights based on their partial derivatives with relation to C. In-depth derivations are provided in [3] and [15].

2.2.1. Recurrent Neural Networks Feed-forward networks are good tools, but do not have any form of state (memory). Here we review recurrent neural networks (RNNs) which have a hidden state h_t.

At step t in a recurrent neural network, we have input x_t and a hidden state h_{t-1}. We also have weight matrices U and W. We can define the current state $h_t = \sigma(Ux_t + Wh_{t-1})$. We have another weight matrix V for the output. The output $y_t = softmax(h_t)$, where $softmax(z_i) = \frac{e_{z_i}}{\sum_j e^{z_j}}$. The softmax function output is a probability distribution over the vector.

Using the softmax function has several advantages. Because it is a smooth function, we can take its derivative, making it easier to calculate the backpropagation equations. Additionally, most RNNs and variants use a loss function known as cross-entropy loss. If the model output is $\hat{y}$ and the real (target) is y, then cross-entropy loss is defined as $\sum_i y_i \log(\hat{y}_i)$. When we pair a softmax output with cross-entropy loss, the error at the output node i is simply $\hat{y}_i - y_i$. This is very convenient because it can be calculated rapidly, so it allows for fast training of the RNN. Training is normally a bottleneck, and by combining a robust output function and robust loss function we get a very elegant output error. As this is a very important result, we have included it in 6.1.

3. Related Work

The first major research in machine comprehension was conducted by Hirschman, et al.[9]. Their system "Deep Read" takes a story and a series of questions as input, and proposes answers by using bag-of-words techniques with other methods such as stemming and pronoun resolution layered on top. On a collection of remedial reading questions for grades 3-6, Deep Read answered about 40% of the questions correctly.

Most recent research has used the MCTest[16], released by Richardson, et al. at Microsoft research. There are a total of 500 passages, each with 400 questions, created at the reading level of a young child. The stories are fictitious, which means the answers can only be found in the story itself. Richardson et al. also provide two baseline implementations to serve as a benchmark. The first is a simple bag-of-words model with a sliding window. It scores about 51% accuracy on the

MC500 questions. The second implementation adds a distance-based scoring metric and reaches 56% accuracy. Both implementations score significantly higher on questions where the answer is contained in just one sentence than on questions where the answer requires information across multiple sentences.

Narasimhan and Barzilay[13] also work on the MCTest tasks. Their main insight is to create a task-specific discourse parser to capture specific discourse relations in the MCTest passages, while the prevailing method had been to use generalized off-the-shelf discourse parsers. They create three models of increasing complexity. The first assumes each question can be answered using just one sentence, and estimates a joint probability distribution across the question, query, and answer. The second model adds in the joint distribution across a second sentence as well, to handle the case in which the answer needs two sentences to answer. The third model incorporates discourse relations between the two sentences to better understand the relationship between information in each sentence. The relations are defined as "Causal", "Temporal", "Explanation", and "Other". Most systems have performed the worst on these types of questions. By focusing specifically on modeling these explanatory relations, the third model easily outscores the other two, and the two MCTest baseline implementations, achieving almost 64% accuracy.

Berant et al.[2] also focus on analyzing inter-sentence relations, with application specifically to passages and questions about biological processes. These passages generally describe a chemical reaction or other process in which there are various starting entities which interact with each other and form a new output. Understanding how the inputs interact and tracing the flow of the process is crucial to answering the question. To solve this, Berant et al. define events (or non-events) as "triggers", and try to find relationships between events. The events can be thought of as nodes on a graph, with an edge defining some relation between the two. There are eight possible relations, including "cause", "enable", and "prevent". They first create events and then predict relations between the events. The queries are also formulated as a graph. They are categorized as dependency questions, temporal questions, or true-false questions. This model scores almost 67% on the dataset, over 6% better than the next-best model.

Most recent machine comprehension research has focused on using neural nets as the core system. Neural nets are a very generalizable technique and in the past decade have become very popular. In fact, advances in neural net techniques have greatly contributed to the recent surge in machine comprehension research.

Weston et al.[21] introduced memory networks. These are a type of neural network which simulate a long-term memory. We discussed in 2.2.1 how RNNs have a "vanishing gradient" problem, and are not able to take advantage of states from more than a few steps prior. Memory networks have a memory module which stores old memories and then updates them given new inputs. When given a query, the memory network finds relevant memories, then finds memories that are relevant given those memories, and so on. Finally, it provides an answer to the query. Sukhbaatar et al.[18] improved this model by creating a model that can be trained end to end, while in the original design each module needed to be trained independently. This is the model that is the basis for our design, so we discuss it in more detail in 4. Kumar et al.[11] create "Dynamic Memory Networks". Their design uses two memory modules. The semantic memory module stores general knowledge, while the episodic memory module iteratively finds memories relevant to the query.

Knowledge Graphs are also a popular research area. There are two main approaches that are used for solving knowledge graph embedding and query problems. The first is the neural network embedding style, popularized by Bordes et al.[4] and improved by Wang et al.[19]. Bordes' TransE and Wang's TransH both train a neural network to recognize triple embeddings in some higher dimension. The triples are embedded in such a way that triples that are "similar" according to some metric are embedded near each other. Queries can be answered more easily using the embedded vectors.

The second approach focuses on representing the knowledge graph as a tensor. Nickel et al.[14] proposed RESCAL, a model which converts the knowledge graph into a tensor of adjacency matrices and trains a latent rank-r factorization of the tensor using a technique called ASALSAN[1]. Queries about the knowledge graph can be easily converted into searching over a relevant subset of elements in the factorization. Chang et al.[5] created TRESCAL, which is an optimized version of RESCAL.

4. Model

This research presents three fundamental insights.

1. We treat the input text as a knowledge base and dynamically convert it into a knowledge graph. We can then interpret the questions as queries on the graph, requiring us to find an entity or relation that represents the answer to the question.

2. We use tensor decomposition to embed the knowledge graph and questions. Specifically, we use an enhanced version of RESCAL[14]. RESCAL by itself is a fine model, but we make it more accurate by adding "Semantically Smooth Embedding"(SSE), as proposed by Guo et al.[8]. In their original paper they propose adding this type of embedding to tensor decomposition as future work, so to our knowledge we are the first to perform tensor decomposition with SSE.

3. We embed the knowledge graph and questions into input vectors, and pass them into a Memory Network. Given a set of embedded inputs and queries, this type of neural network takes advantage of "memory" to find new facts that are relevant given the current set of relevant facts, but may not be relevant solely based on the query.

These techniques were first introduced in previous research. However, all of them were applied to separate domains. Neural nets have been used for machine comprehension on MCTest, bAbi, and questions of similar difficulty. Their embedding model is quite simple, however (often just bag-of-words). Knowledge graphs and tensor decomposition have traditionally been used only for relational learning when a knowledge graph or set of triples already exists. As our problem is more complex than that tackled by these original models, and we must combine these original models, we have to significantly modify their conceptual and mathematical basis.

4.1. Knowledge Graph Representation

The first part of our design is to dynamically convert the input passage into a knowledge graph and the questions as queries on that graph.

4.1.1. Advantages of Knowledge Graph Representation There are several advantages to using a knowledge graph representation. The main benefit is that it actually preserves the meaning of the text. Whereas bag-of-words loses the ordering of words and word2vec keeps the latent properties but no others, a knowledge graph representation almost completely stores the intended meaning of the original text. Most sentences in English can be broken down into Subject-Verb-Object triples, which require little if any processing to convert into the $\langle entity_1, relation, entity_2 \rangle$ format. The knowledge graph itself is simply a list of these triples. If we can parse the input text and extract the Subject-Verb-Object relationship, we can convert each sentence into a triple of the form

$$\langle subject, verb, object \rangle \Rightarrow \langle e_1, r, e_2 \rangle$$

How to actually perform this conversion is covered in 4.1.2. *****ADD FIGURES HERE SHOWING EXAMPLE TRANSFORMATIONS******

Even more importantly, the knowledge graph inherently has *memory*. What this means is that it automatically pieces together information about the same concept, even if that information occurs across different sentences or even different paragraphs. *****ADD FIGURE HERE SHOWING EXAMPLE******* As the figure shows, even if in one paragraph the text says that Harry was born in 1980 and several paragraphs later we learn that he has a friend named James, these relations both link to the single Harry entity node. By analyzing the outbound and inbound relations of the Harry entity node, we can easily find all the information the text has stated about Harry. Any question that is asked about Harry can be answered using this information. While such a representation for memory may seem natural, virtually all previous research in the machine comprehension space has parsed by sentence. As a result, the knowledge graph representation is the best way for our system

to easily find all information about an entity that was given in the original text.

Entities also do not have to be concrete things; they can just as easily be abstract concepts. For example, Figure 1 shows entities like Carl Pavano (a person), but it also shows an entity node for the year 1976. To roughly generalize, any noun can be an entity node. The phrase "abstract concept" itself could be an entity node. Entity-relation triples can all be analyzed similarly, so abstract reasoning is as simple as concrete reasoning. Not only does a knowledge graph allow us to perform abstract reasoning, but it is arguably the only way to perform reasoning of this form.

A good way to understand the knowledge graph representation is to think of it as a *concept map*. Every sentence in the text introduces a new concept (concrete or abstract) and/or provides more information about a previously introduced concept. All information is provided in such a way that the concept is either the actor or the receiver of some action. We update our concept map with this new information by adding entity nodes as needed and the relation provided in the current sentence. When we've finished parsing the input text, we have the full concept map, which gives complete information on each concept and the information we know about it.

Another benefit is that there are already methods and theory of analysis for knowledge graphs. As mentioned earlier, passages as complex as SAT reading comprehension tests have rarely been analyzed semantically. Moreover, the fact that information comes in a variety of sentence structures across several paragraphs means that traditional semantic analysis will not suffice. Traditional analysis often parses one sentence or one paragraph at a time, which is not enough for our particular use case. The original input simply cannot be analyzed as it is using existing techniques. Since this problem cannot be solved in its original form, we convert it to a form which we know how to analyze. Not only have knowledge graphs been studied extensively in the field of relational learning, but they can also be analyzed using techniques from graph theory. Essentially, we turn the problem of analyzing complex text into a problem of analyzing a knowledge graph.

At the same time, the knowledge graph representation also preserves human readability. A major problem with text input is that it is easily interpretable by humans but requires much preprocessing before it can be interpreted by a program. It is not necessary for our processed text to be in a

format humans can interpret, but it certainly helps us reason about the problem and come up with interesting solutions. The entity-relation triple format is almost as intuitive to humans as the original textual input.

Of course, even the knowledge graph representation has some drawbacks. Because we build the knowledge graph using the statements in the original text, we can only perform analysis on the textual information. We can trace relations through the graph but there is no way to perform higher-level reasoning. For example, questions that ask about classification of new information or broad themes of the passage cannot be answered. Nevertheless, these are flaws in any form of representation. The knowledge graph representation provides the most advantages and fewest drawbacks of any representation.

4.1.2. Converting to Knowledge Graph Representation ******FILL THIS IN******

4.2. Tensor Decomposition Embedding

First we discuss the tensor decomposition approach to answering knowledge graph queries. The other main way to solve relational learning queries is with tensor analysis. The advantage of using tensor decomposition is that it is based on well-understood mathematics, rather than simply empirical validation as with the neural network.

Our model is an improved version of RESCAL, proposed by Nickel et al.[14] and used by Chang et al.[5]. RESCAL itself takes the general tensor decomposition model proposed by Bader et al.[1] and modifies it to work for relational learning.

4.2.1. Knowledge Base Tensor The tensor decomposition approach treats the knowledge graph as a directed graph. A tensor is simply a multi-dimensional array. In our case, we use a three-dimensional array $\mathcal{X}$ to represent the knowledge graph. If we have N entities $e_1, e_2, ..., e_N$ and M relations $k_1, k_2, ..., k_M$ then we can consider $\mathcal{X}$ as M layers of $N \times N$ matrices. Each layer $\mathcal{X}_k$

tells which entities are linked by relation k. If we denote our knowledge base as KB, then $\mathcal{X}$ is formulated as

$$\begin{cases} \mathcal{X}_{ijk} = 1 & \langle e_i, k, e_j \rangle \in KB \\ \mathcal{X}_{ijk} = 0 & \text{otherwise} \end{cases}$$

*********INCLUDE FIGURE HERE********. Another way to think of this is to see the matrix $\mathcal{X}_k$ as an adjacency matrix of the knowledge graph, but only for the relation type k.

4.2.2. Tensor Factorization Our approach aims to factorize the tensor such that we take advantage of the inherent structure of the graph. To that end, we train a rank-r factorization of each slice, as suggested in [1][14].

$$X_k \approx AR_kA^T \text{ for } k \in [1, m]$$

Here, A is an $n \times r$ matrix which contains the latent-component representation of the entities in the knowledge graph. We can think of this as an r-dimensional embedding of the entities, where we claim that there are r hidden features of each entity. This is conceptually similar to the word2vec embedding, where each feature of a word vector does not necessarily mean something intuitive but is a hidden component found through training.

R_k is a an $r \times r$ matrix which represents how these hidden features interact with each other for the k-th relation. R_k is asymmetric, which means that the hidden feature interactions are one-way. It is not necessarily true that $R_{ijk} = R_{jik}$, although it could be. Note that R itself is also a three-dimensional tensor of shape $r \times r \times k$.

The reason we factorize the tensor is that we can now answer queries using the factorization, which has the hidden components of the graph and is thus more generalizable, as opposed to using the original tensor which is simply an adjacency tensor of the knowledge graph.

We will frame the problem of finding A and R_k as an optmiziation problem, where we aim to

minimize the difference between the original matrix $\mathcal{X}_k$ and its factorization AR_kA^T. Specifically, define the loss function f as the squared frobenius norm of this difference.

$$f(A,R_k) = \frac{1}{2}\sum_k \|\mathcal{X}_k - AR_kA^T\|_F^2 \tag{1}$$

where the frobenius norm of some $a \times b$ matrix X is defined as

$$\|X\|_F = \sqrt{\sum_{i=1}^{a}\sum_{j=1}^{b}|X_{ij}|^2}$$

To find the optimal A and R_k, we just have to minimize 1, along with some regularization function $g(A,R_k)$. Formally,

$$\min_{A,R_k} f(A,R_k) + \lambda g(A,R_k) \tag{2}$$

In [14][5] the regularization term is simply defined as $g(A,R_k) = \frac{1}{2}\lambda(\|A\|_F + \sum_k \|R_k\|_F)$. This is a standard regularization function where λ is a hyperparameter. The aim is to prevent overfitting when we try to solve the optimization problem. We enhance their approach by using a stronger regularization function which adds more constraints on the factorization.

4.2.3. Semantically Smooth Embedding Semantically Smooth Embedding (SSE) was proposed by Guo et al.[8] as a way to improve the TransE[4] embedding model. Whereas the traditional models simply embed each individual fact in isolation, SSE imposes geometric constraints on the whole model.

Specifically, we modify the *Locally Linear Embedding* (LLE) constraint. This constraint states that each node should be roughly a linear combination of its neighbors[17]. Guo et al. created a version which applies to the neural net embeddings done by TransE and other models, but does not apply to the tensor decomposition method. Here we review their version first and then present our new version.

In the original implementation each entity was labeled with a category and then embedded into a vector representation. All nodes in the same category as e_i were defined as its neighbors and this set was denoted $\mathcal{N}(e_i)$. Then they defined

$$w_{ij} = \begin{cases} 1 & \text{if } e_j \in \mathcal{N}(e_i) \\ 0 & \text{otherwise} \end{cases}$$

$$R = \sum_{i=1}^{n} \|\mathbf{e_i} - \sum_{e_j \in \mathcal{N}(e_i)} w_{ij}\mathbf{e_j}\|_2^2$$

where R is the regularization term added to the loss function. The logic here is that we suffer more penalty in our loss function when entities that are close to each other in the original set (i.e. are neighbors) have embedded vectors that are far from each other. We cannot use this equation for our purposes since our entities do not have categories and we do not have embedded vectors, but we can apply similar logic.

To that end we propose defining the neighbor set of each entity e_i in our graph as all other entities for which there is a directed edge from e_i to e_j in the knowledge graph. The full neighbor set matrix is denoted W.

$$w_{ij} = \begin{cases} 1, & \sum_k \mathcal{X}_{ijk} \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$e_j \in \mathcal{N}(e_i) \text{ iff } w_{ij} = 1 \tag{4}$$

That is to say, $w_{ij} = 1$ if and only if we have some relation r for which the triple $\langle e_i, r, e_j \rangle$ exists in the knowledge base. We do not consider entities which are related to e_i in such a way that e_i is the recipient of the relation. Note that this means the neighbor relation goes one way. $e_j \in \mathcal{N}(e_i)$ does not imply that $e_i \in \mathcal{N}(e_j)$.

Now consider the nodes in $\mathcal{N}(e_i)$. Since e_i is related to all of these, and because we want to

enforce some geometric structure on our representation (as done by Guo) we claim that the latent properties of each node e_i, as provided in the matrix A, should be some linear combination of its neighbor set.

$$A_i \approx \sum_{i=1}^{n} w_{ij} A_j = \sum_{i=1} W_i A$$

We can penalize more for an A in which this distance is large. The reason we made the neighbor relation only one way is that we do not expect a large number of disconnected components of our knowledge graph, and a two-way neighbor set would push every node to have the same latent representation.

4.2.4. Solving the Optimization Problem The new, semantically smooth regularization function is

$$g(A, R_k) = \sum_{i=1}^{n} \left\| A_i - \sum_{j=1}^{n} w_{ij} A_j \right\|_F^2 = \sum_{i=1}^{n} \|A_i - W_i A\|_F^2 \tag{5}$$

We are now ready to actually solve for A and R_k. We want to minimize $f(A, R_k) + \lambda g(A, R_k)$. This is a nonlinear, non-convex optimization problem which would require complicated techniques. Calculating f and g is expensive, so we use the Alternating Least Squares(ALS) method[10]. Specifically, we use the ASALSAN factorization method[1].

ALS turns the non-convex optimization problem into a convex problem by fixing one of the unknowns. Then the problem can be solved with ordinary least-squares optimization, so we use the normal method of least-squares to lower the variable unknown. Then we alternate by fixing this unknown and changing the other one. The ordinary least-squares can be solved relatively fast (although calculating inverse matrices is expensive) and since each unknown is calculated independently (keeping one fixed), the ALS algorithm can be easily parallelized. These two qualities make it very computationally fast, a quality necessary for our purposes.

We have

$$\bar{\mathcal{X}} = A\bar{R}(\mathbf{I}_{2m} \otimes A^T)$$

where $\otimes$ is the Kronecker product and

$$\bar{\mathcal{X}} = \left(\mathcal{X}_1 \mathcal{X}_1^T ... \mathcal{X}_m \mathcal{X}_m^T\right)$$

$$\bar{R} = \left(R_1 R_1^T ... R_m R_m^T\right)$$

Then by ASALSAN we derive the following update rules. We first update A while keeping R_k fixed.

$$A \leftarrow \left[\sum_k \mathcal{X}_k A R_k^T + \mathcal{X}_k^T A R_k\right]\left[B_k + C_k + \lambda \mathbf{I}\right]^{-1}$$

where

$$B_k = R_k A^T A R_k^T, \quad C_k = R_k^T A^T A R_k$$

To update R_k, we first fix A. Then we notice that if we vectorize $\mathcal{X}$ and R_k, we can rewrite 2 as

$$f(R_k) = \|\mathbf{vec}\left(\mathcal{X}_k\right) - \left(A \otimes A\right)\mathbf{vec}\left(R_k\right)\|$$

where we have removed an extra term $\lambda \|\mathbf{vec}(R_k)\|$ because R_k is no longer in our regularization term. This rewritten f is actually a linear regression problem whose solution is

$$R_k \leftarrow \left(\left(A \otimes A\right)^T \left(A \otimes A\right) + \lambda \mathbf{I}\right)^{-1} \left(A \otimes A\right)\mathbf{vec}\left(\mathcal{X}_k\right)$$

We alternate updating A and R_k until we have performed some maximum number of iterations or until $\frac{f(A,R_k)}{\|\mathcal{X}_k\|_F^2}$ is below some tolerance ε. The final factorization $\hat{\mathcal{X}}$ can now be used to answer queries.

4.2.5. Answering Queries Link-prediction queries are relatively simple to answer. Given e_i, e_j the existence of a link between the two can be found by seeing whether for any relation k we have that

$\hat{\mathcal{X}}_{ijk} > 0$. Given an entity e_i and a relation k, we can see whether some entity e_j completes the triple by seeing whether $\hat{\mathcal{X}}_{ijk} > \theta$, where θ is some threshold of confidence.

4.3. Memory Networks

The second way we propose answering queries is with neural networks. Specifically, an enhanced version of Memory Networks. The advantage of the neural network approach is that it is very intuitive, computationally feasible, and empirically proven. However, it does lack the strong theory of the mathematical approach. We choose the Memory Networks architecture as the basis for our model because it has strong performance on the bAbi dataset and the foundational architecture itself scales easily to more complex problems as long as the input is in a valid format. First we review the architecture as proposed by [18]. Then we discuss how to adapt it to this specific problem.

4.3.1. Memory Network Architecture First we will review the architecture for one layer, as the extension to several layers is straightforward. The basic model takes in a set of input vectors $x_1, ..., x_n$ and a query vector q, and outputs an answer a. Memory Networks require the input text and the query to be vectorized in some form for easy computation. These embedding techniques are covered in detail in 2.1. Typically the embedding is done with bag-of-words embedding. This embedding technique relies on the same word being used in the answer choices as well as in the original text, which is true for MCTest and bAbi. The vocabulary comes from a dictionary of fixed size V.

The input vectors are converted into a set of memory vectors m_i using a matrix A of size $d \times V$, where d is a pre-determined dimension. q is also embedded into a state u using a different matrix B of size $d \times V$. To find which memory vectors are most relevant to the initial query, we take the dot product of each vector with u.

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

where $\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$. Since the softmax function returns a value between 0 and 1, each

score p_i is a scalar that can be considered a probability of how relevant each memory vector is to the query. The more relevant it is, the more likely it will help find the answer.

The input vectors are now sent through another matrix C to produce a set of transformed inputs c_i. Each transformed vector c_i is weighted by its probability c_i and the sum is taken as the output o of that layer.

$$o = \sum_i p_i c_i$$

This output can then be fed as an input to the next layer, but in the single layer case it is added to u and sent through one final matrix W of size $V \times d$. Lastly, a softmax is taken on the result to yield a probability distribution over the vocabulary space. This is designed to pick one word or set of words as the answer.

$$\hat{a} = \text{Softmax}\left(W\left(o + u\right)\right)$$

Several strategies can be used to extend the Memory Network to multiple layers. We use the *Recurrent Neural Network* strategy as it makes most sense for our purposes. This strategy is the simplest to understand as well as to train. Each layer acts the same as in the one-layer case. The only difference is that for all layers $k > 1$ the input $u^{k+1} = u^k + o^k$, and the matrix W is only applied to o^{final}. Each layer uses the same A, B, C matrices. By using the same matrices at each level and passing the state along, this architecture acts like a Recurrent Neural Network.

The initial layer calculates how relevant each memory vector is to the initial query. That output is combined with the initial query and passed as u to the next layer. That layer finds how relevant each memory vector is to the new state u. This means that memories which were not very relevant to the initial query may be relevant to the memories which were relevant to the initial query. Each layer finds new memories that can help answer the question.

4.3.2. Memory Network Enhancements As discussed, the original Memory Networks model uses simple bag-of-words embedding to vectorize the text into the input vectors x_i. This cannot work for SAT question purposes because each layer finds what is most relevant based on which words are shared between query and input. To answer SAT questions the text must be vectorized in such a way that the knowledge graph relationships are captured. One possible way would be to use word2vec embedding, but it still does not make use of the inherent structure of the original text, nor is it compatible with the knowledge graph. We need to change the inputs while preserving the overall architecture so that the memories chosen as relevant actually match the concept in the query, as opposed to simply matching the words.

We propose stacking horizontally each slice of the tensor $\mathcal{X}$ as defined in 4.2.1 and using each column as an input vector. In total the input matrix $\mathcal{X}'$ will be of shape $n \times (nm)$. The query is more complicated to vectorize. The key is that we can parse the question itself in the same way we parsed the original text and obtain an entity-relation triple. However, in this case either an entity or the relation is missing. Finding the right answer can be turned into a problem of finding the missing entity or relation. If the query has an entity e_i and a relation k, we pass in the column vector $\mathcal{X}_{ik}^T$ as the query. If instead it is a missing link problem, we create a m-length vector where element k is equal to $\mathcal{X}_{ijk}$ and pad the rest so that it becomes an n-length vector q.

The input vectors and query vectors already take advantage of the knowledge graph representation, so we do not need to embed them again using A, B matrices. $m_i = x_i$ and $u = q$. We do, however, use a separate matrix C to transform the input vectors again. In a one-layer case, the architecture calculates

$$p_i = \text{Softmax}\left(u^T m_i\right)$$

$$o = \sum_i p_i c_i$$

$$\hat{a} = \text{Softmax}\left(W\left(o + u\right)\right)$$

In the multiple-layer case, the approach is the same as in the original model where the matrix C

stays constant at each layer, and $u^{k+1} = u^k + o^k$.

5. Data

For preliminary testing, we use both the MCTest [16] dataset as well as the Facebook bAbi dataset [20]. The MCTest baseline algorithms provide a benchmark to test our preliminary bag-of-words implementations against. The bAbi dataset is used to evaluate [18]. We use the same model so our preliminary neural net implementation is also evaluated on the bAbi dataset. However, we do not use as many training optimizations. These comparisons are meant to be benchmark comparisons, rather than trying to perform better.

The crux of our project relies on SAT reading comprehension tests for both training and testing. As there are no publicly available datasets, we tried contacting ETS to obtain a research dataset. As we have not yet heard a response, we also collected practice SAT tests from prep books. As of now we have 8 tests from a CollegeBoard prep book, and 11 from a Princeton Review prep book (10 full practice tests and 1 practice PSAT test). We already owned these prep books.

Each practice test contains 3 reading sections. Each reading section has approximately 18 comprehension questions. About 4-6 of these are from short passages (100 words), and the remaining are from longer passages (450-500 words). There are 2 short passages and 1-2 long passages per section, each followed by comprehension questions. Hereafter we use "test" to refer to just the 3 reading sections of a test, and "reading section" to refer to just the reading comprehension passages and accompanying questions of each reading section. Math and writing sections, as well as the vocabulary questions in the reading section, are ignored.

Our practice tests are in paper format, so we must put them in an electronic format. Amazon Mechanical Turk was used for this task. Each test was scanned, and we approximated that typing up one test requires an hour. We requested Master Turkers, 2 per test, and paid $8.00 for each task. The total expenditure was $304.00.

References

[1] B. Bader, R. a. Harshman, and T. G. Kolda, "Temporal Analysis of Semantic Graphs Using ASALSAN," *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 33–42, 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4470227

[2] J. Berant and P. Clark, "Modeling Biological Processes for Reading Comprehension," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1499–1510, 2014. [Online]. Available: http://allenai.org/content/publications/berant-srikumar-manning-emnlp14.pdf

[3] C. M. Bishop, *Neural networks for pattern recognition*, 1995, vol. 92.

[4] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems*, 2013, pp. 2787–2795.

[5] K.-W. Chang, W.-t. Yih, B. Yang, and C. Meek, "Typed tensor decomposition of knowledge bases for relation extraction," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1568–1579, 2014.

[6] D. Chen and C. D. Manning, "A Fast and Accurate Dependency Parser using Neural Networks," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, no. i, pp. 740–750, 2014. [Online]. Available: https://cs.stanford.edu/{~}danqi/papers/emnlp2014.pdf

[7] M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning, "Universal stanford dependencies: A cross-linguistic typology." in *LREC*, vol. 14, 2014, pp. 4585–4592.

[8] S. Guo, Q. Wang, B. Wang, L. Wang, and L. Guo, "Semantically smooth knowledge graph embedding," in *Proceedings of ACL*, 2015, pp. 84–94.

[9] L. Hirschman, M. Light, E. Breck, and J. D. Burger, "{D}eep {R}ead: {A} Reading Comprehension System," *Proceedings of ACL*, pp. 325–332, 1999.

[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *Computer*, vol. 42, no. 8, pp. 42–49, 2009.

[11] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing," *arXiv*, pp. 1–10, 2015.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *Nips*, pp. 1–9, 2013.

[13] K. Narasimhan and R. Barzilay, "Machine Comprehension with Discourse Relations," *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1253–1262, 2015. [Online]. Available: http://www.aclweb.org/anthology/P15-1121

[14] M. Nickel, V. Tresp, and H.-P. Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data," *28th International Conference on Machine Learning*, pp. 809—-816, 2011.

[15] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/index.html

[16] M. Richardson, C. J. C. Burges, and E. Renshaw, "MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text," *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, no. October, pp. 193–203, 2013.

[17] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[18] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-To-End Memory Networks," pp. 1–11, 2015. [Online]. Available: http://arxiv.org/abs/1503.08895

[19] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes." in *AAAI*. Citeseer, 2014, pp. 1112–1119.

[20] J. Weston, A. Bordes, S. Chopra, T. Mikolov, and A. M. Rush, "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks," 2015. [Online]. Available: http://arxiv.org/abs/1502.05698

[21] J. Weston, S. Chopra, and A. Bordes, "Memory Networks," *International Conference on Learning Representations*, pp. 1–14, 2015. [Online]. Available: http://arxiv.org/abs/1410.3916

6. Appendix

6.1. Derivation of Output Layer Error

In this section we present the derivation for the output layer error of an RNN with cross-entropy loss and softmax at the output layer. Let us consider an output layer with n nodes. Define z_i as the quantity before activation function at the output layer, and h_i as the final output of node i.

$$h_i = \text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} = \frac{e^{z_i}}{e^{z_i} + \sum_{j \neq i} e^{z_j}} \tag{6}$$

The partial derivatives yield

$$\begin{aligned}
\frac{\partial h_i}{\partial z_i} &= \frac{\sum_{j=1}^{n} e^{z_j}\left(e^{z_i}\right) - e^{z_i}\left(e^{z_i}\right)}{\left(\sum_{j=1}^{n} e^{z_j}\right)^2} = \frac{e^{z_i}\sum_{j=1}^{n} e^{z_j}}{\left(\sum_{j=1}^{n} e^{z_j}\right)^2} - \left(\frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}\right)^2 \\
&= h_i - h_i^2 = h_i\left(1 - h_i\right) \\
\frac{\partial h_i}{\partial z_j} &= \frac{\sum_{j=1}^{n} e^{z_j} * 0 - e^{z_i}\left(e^{z_i}\right)}{\left(\sum_{j=1}^{n} e^{z_j}\right)^2} = \frac{-e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \frac{e^{z_j}}{\sum_{j=1}^{n} e^{z_j}} \\
&= -h_i h_j
\end{aligned} \tag{7}$$

******WORK IN PROGRESS*******

6.2. Full Evaluation Data

Here we present the full evaluation data on three practice tests.

Table 1: Evaluation on Test 1

	Number	Tensor w/o Reg	Tensor w/ Reg	1-layer MemN2N	3-layer MemN2N	Correct Answer	Difficulty
Section 1							
Small	9	A	A	-	-	E	E
	10	A	D	-	-	B	M
	11	D	D	-	-	D	E
	12	E	E	-	-	D	E
Long	13	C	D	train	train	E	M
	14	D	D	train	train	D	E
	15	E	B	train	train	B	M
	16	E	A	train	train	A	H
	17	A	A	train	train	A	H
	18	E	D	train	train	C	E
	19	D	D	train	train	A	M
	20	D	A	train	train	B	M
	21	E	E	A	A	B	M
	22	A	A	A	A	C	M
	23	D	D	A	A	B	M
	24	A	B	A	A	A	E
Section 2							
Medium	13	B	B	train	train	C	M
	14	C	D	train	train	B	E
	15	E	E	train	train	C	M
	16	B	B	train	train	E	E
	17	A	A	C	C	D	E
	18	A	A	C	D	C	M
Medium	19	B	E	train	train	E	M
	20	E	E	train	train	C	E
	21	D	B	train	train	D	M
	22	E	E	train	train	E	M
	23	D	E	E	C	A	M
	24	B	B	E	C	B	M
Section 3							
Long	7	A	A	train	train	B	M
	8	C	C	train	train	A	M
	9	E	E	train	train	D	M
	10	D	D	train	train	E	M
	11	B	B	train	train	A	M
	12	C	C	train	train	A	E
	13	E	B	train	train	D	E
	14	B	B	train	train	D	M
	15	A	A	D	D	E	M
	16	A	A	D	D	E	M
	17	D	D	D	D	C	M
	18	D	D	D	A	A	M
	19	C	C	D	D	E	M
Total Questions		41	41	13	13		
Total Correct		7	8	2	2		

Table 2: Evaluation on Test 2

	Number	Tensor w/ Reg	3-layer MemN2N	Correct Answer	Difficulty
Section 1					
Long	13	C	train	A	M
	14	A	train	C	M
	15	D	train	E	M
	16	D	train	D	M
	17	B	train	D	M
	18	E	train	D	M
	19	E	train	E	M
	20	A	train	C	M
	21	B	E	A	M
	22	A	B	B	M
	23	B	B	B	H
	24	B	D	A	M
Section 2					
Medium	10	B	train	A	M
	11	B	train	A	M
	12	A	train	D	M
	13	C	train	C	E
	14	B	D	B	M
	15	E	A	A	M
Medium	16	A	train	E	M
	17	B	train	D	E
	18	A	train	A	E
	19	E	train	C	M
	20	B	train	E	H
	21	A	train	A	M
	22	C	E	C	M
	23	E	E	B	M
	24	B	E	B	M
Section 3					
Long	7	C	train	B	M
	8	E	train	B	E
	9	A	train	D	M
	10	C	train	D	M
	11	E	train	C	E
	12	E	train	A	M
	13	A	train	B	M
	14	E	train	E	H
	15	A	B	B	M
	16	E	E	E	M
	17	C	D	D	M
	18	A	B	B	M
	19	C	C	A	M
Total Questions		40	14		
Total Correct		11	7		