공학박사 학위논문

대화처리를 위한
딥 메모리 네트워크

(Deep Memory Networks

for Natural Conversations)

2017년 8월

서울대학교 대학원

컴퓨터공학부

장 하 영

대화 처리를 위한
딥 메모리 네트워크

(Deep Memory Networks

for Natural Conversations)

지도 교수 장 병 탁

이 논문을 공학박사 학위논문으로 제출함

2017년 8월

서울대학교 대학원

컴퓨터공학부

장 하 영

장하영의 공학박사 학위논문을 인준함

2017년 7월

위 원 장 최 진 영 (인)

부위원장 장 병 탁 (인)

위 원 문 병 로 (인)

위 원 신 수 용 (인)

위 원 석 호 식 (인)

Abstract

Attention-based models are firstly proposed in the field of computer vision. And then they spread into natural language processing (NLP). The first one successfully bringing in attention mechanism from computer vision to NLP is neural machine translation. Such attention-based mechanism is motivated from that, instead of decoding based on the encoding of a whole and a fixed-length sentence during one pass of neural network-based machine translation, one can attend a specific part of the sentence. This specific part is what should currently be attended. These parts could be words or phrases.

The basic problem that the attention mechanism solves is that it allows the network to refer back to the input sequence, instead of forcing it to encode all information into one fixed-length vector. The attention mechanism is simply giving the network access to its internal memory, which is the hidden state of the encoder. In this point of view, instead of choosing what to “attend” to, the network chooses what to retrieve from memory. Unlike typical memory, the memory access mechanism here is soft, which means that the network retrieves a weighted combination of all memory locations, not a value from a single discrete location. Making the memory access soft has the benefit that we can easily train the network end-to-end using backpropagation

The trend towards more complex memory structures is now continuing. End-to-End Memory Networks allow the network to read same input sequence multiple times before making an output, updating the memory contents at each step. For

example, answering a question by making multiple reasoning steps over an input story. However, when the networks parameter weights are tied in a certain way, the memory mechanism in End-to-End Memory Networks identical to the attention mechanism presented here, only that it makes multiple hops over the memory.

In this dissertation, we propose the deep memory network with attention mechanism and word/sentence embedding for attention mechanism. Due to the external memory and attention mechanism, proposed method can handle various tasks in natural language processing, such as question and answering, machine comprehension and sentiment analysis. Usually attention mechanism requires huge computational cost. In order to solve this problem. I also propose novel word and sentence embedding methods. Previous embedding methods only use the Markov assumption. But if we consider the language structure and make use of it, it will be very helpful to reduce the computational cost. Also it does not need strong supervision which means the additional information on important sentences.

Keywords : Attention Model, Memory Network, Deep Learning, Natural Language Understanding, Machine Comprehension

Student Number : 2004-30347

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Neural network models have recently become the most effective tools for a range of hard applied natural language processing problems, including translation (Luong et al. 2015), sentiment analysis (Socher et al. 2011), and text generation (Wen et al. 2015). These models succeed in large part because they can learn and use their own continuous numeric representational systems for sentence meaning. However, their representations need not correspond in any interpretable way with the logic based representations typically used in linguistic semantics. These models' successes in learning to solve semantically difficult problems signal that they are a potentially valuable object of study for semantics, and drawing insights from semantics to improve these models could yield substantial progress across applied language understanding tasks. But there is no general method to solve the various tasks in natural language problem.

Most tasks in natural language processing can be cast into question answering (QA) problems over language input. QA is a complex natural language processing task which requires an understanding of the meaning of a text and the ability to reason over relevant facts. Most, if not all, tasks in natural language processing can be cast as a question answering problem: high level tasks like machine translation

(What is the translation into French?); sequence modeling tasks like named entity recognition (Passos et al., 2014) (NER) (What are the named entity tags in this sentence?) or part-of-speech tagging (POS) (What are the part-of-speech tags?); classification problems like sentiment analysis (Socher et al., 2013) (What is the sentiment?); even multi-sentence joint classification problems like co-reference resolution (Who does “their” refer to?).

Most higher intelligences in nature have a built-in mechanism for deciding how to apply their brainpower from moment to moment. It is called attention, and refers to management of cognitive resources. Human attention is a reasonably well studied subject within the field of psychology and known to be a key feature of human intelligence. Without attention we would constantly be overloaded with stimuli, severely affecting our ability to perform tasks, make decisions and react to the environment.

Attention-based models are firstly proposed in the field of computer vision (Mnih et al., 2014). And then they spread into natural language processing (NLP). The first one successfully bringing in attention mechanism from computer vision to NLP is neural machine translation (Bahdanau et al., 2015). Such attention-based mechanism is motivated from that, instead of decoding based on the encoding of a whole and a fixed-length sentence during one pass of neural network-based machine translation, one can attend a specific part of the sentence. This specific part is what should currently be attended. These parts could be words or phrases.

From an engineering perspective, attention can be viewed as resource optimization, enabling systems to perform tasks in complex environments while requiring insignificant amounts of resources (compared to complexity of tasks and

environments) and using existing resources only for information likely to be important or relevant. In this view, time itself can be treated as a resource.

While a general-purpose attention mechanism, applicable to any NLP, could be a goal to strive for, a perfect and complete independence from architecture has been found practically impossible, as resource management touches on too many fundamental issues in the structure and operation of an architecture to make this a theoretical possibility. The goal of the present work is therefore not to develop an attention component that can be plugged directly in to existing NLP architectures.

This work is motivated by the desire to create practical attention based model intended to perform real tasks in natural language processing rather than attempting to validate hypothesis or models relating to the functionality of the brain at any level. While clearly “biologically inspired” at a high level (by natural attention), this work is not biologically inspired in this sense: It does not target an accurate simulation or model of biological mechanisms. Where deemed useful and appropriate, inspiration from research on human attention will be taken, but it is not a goal to have the resulting components be constrained in design by what is known about the functionality of human attention.

1.2 Approach and Contributions

The basic problem that the attention mechanism solves is that it allows the network to refer back to the input sequence, instead of forcing it to encode all information

into one fixed-length vector. The attention mechanism is simply giving the network access to its internal memory, which is the hidden state of the encoder. In this point of view, instead of choosing what to “attend” to, the network chooses what to retrieve from memory. Unlike typical memory, the memory access mechanism here is soft, which means that the network retrieves a weighted combination of all memory locations, not a value from a single discrete location. Making the memory access soft has the benefit that we can easily train the network end-to-end using backpropagation

Memory Mechanisms themselves have a much longer history. The hidden state of a standard Recurrent Neural Network is itself a type of internal memory. RNNs suffer from the vanishing gradient problem that prevents them from learning long-range dependencies. LSTMs improved upon this by using a gating mechanism that allows for explicit memory deletes and updates.

The trend towards more complex memory structures is now continuing. End-to-End Memory Networks allow the network to read same input sequence multiple times before making an output, updating the memory contents at each step. For example, answering a question by making multiple reasoning steps over an input story. However, when the networks parameter weights are tied in a certain way, the memory mechanism in End-to-End Memory Networks identical to the attention mechanism presented here, only that it makes multiple hops over the memory.

In this dissertation, I propose the deep memory network with attention mechanism and word/sentence embedding for attention mechanism. Due to the external memory and attention mechanism, proposed method can handle various tasks in natural language processing, such as question and answering, machine comprehension and sentiment analysis. If we can cast the problems in natural

language processing into question answering problems, every input data can be processed via sequence modeling process. Then attention mechanism can handle it.

Disadvantage of attention mechanism is that it requires huge computational cost. In order to solve this problem. I proposed novel word and sentence embedding methods. Previous embedding methods only use the Markov assumption. But if we consider the language structure and make use of it, it will be very helpful to reduce the computational cost. Also it does not need strong supervision which means the additional information on important sentences.

1.3 Organization of the Dissertation

This dissertation is organized as follows.

In Chapter 2, we discuss memory networks and attention mechanism. We describe how attention mechanism works and the characteristics of previous memory model such as Memory Networks, End-to-End Memory Networks and Dynamic Memory Networks.

In chapter 3, we propose novel distributed representation of words. The proposed methods make use of the relationship between words in sentences. So we can use more accurate representation of words.

In chapter 4, we propose distributed representation of sentences using the co-reference. In linguistics, co-reference occurs when two or more expressions in a text have the same referent. This means that syntactic relationship exists between co-

referential expressions. These kind of information can reduce computational cost of attention mechanism dramatically.

In chapter 5, we propose the Deep Memory Network. Deep Memory Network use the syntactic relationship and structural information of language. It makes the Deep Memory Network locate the attention very efficiently and do not need strong supervision.

Finally, we summarize the dissertation and discuss contributions in Chapter 6.

Chapter 2

Related Work

2.1 Memory Networks

Memory Networks reason with inference components combined with a long-term memory component; they learn how to use these jointly (Weston et al., 2015a). The long-term memory can be read and written to, with the goal of using it for prediction. These models are investigated in the context of question answering (QA) where the long-term memory effectively acts as a (dynamic) knowledge base, and the output is a textual response.

A memory network consists of a memory m (an array of objects indexed by m_i) and four (potentially learned) components *I, G, O* and *R* as follows:

I: (input feature map) – converts the incoming input to the internal feature representation.

G: (generalization) – updates old memories given the new input. This generalization means that there is an opportunity for the network to compress and generalize its memories at this stage for some intended future use.

O: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory state.

R: (response) – converts the output into the response format desired. For

example, a textual response or an action.

Given an input *x* (e.g., an input character, word or sentence depending on the granularity chosen, an image or an audio signal) the flow of the model is as follows:

1. Convert *x* to an internal feature representation *I(x)*.
2. Update memories m_i given the new input: $m_i = G(m_i, I(x), m), \ \forall i$.
3. Compute output features o given the new input and the memory:

   $o = O(I(x), m)$.
4. Finally, decode output features o to give the final response: $r = R(o)$.

This process is applied at both train and test time, if there is a distinction between such phases, that is, memories are also stored at test time, but the model parameters of *I, G, O* and *R* are not updated. Memory Networks cover a wide class of possible implementations. The components *I, G, O* and *R* can potentially use any existing ideas from the machine learning literature, e.g., make use of your favorite models (SVMs, decision trees, etc.).

I component: Component *I* can make use of standard pre-processing, e.g., parsing, co-reference and entity resolution for text inputs. It could also encode the input into an internal feature representation, e.g., convert from text to a sparse or dense feature vector.

G component: The simplest form of *G* is to store *I(x)* in a "slot" in the memory:

$$m_{H(x)} = I(x) \tag{2.1}$$

where *H(.)* is a function selecting the slot. That is, *G* updates the index *H(x)* of m, but all other parts of the memory remain untouched. More sophisticated variants of *G* could go back and update earlier stored memories (potentially, all memories) based on the new evidence from the current input *x*. If the input is at the character or word level one could group inputs (i.e., by segmenting them into chunks) and store each chunk in a memory slot.

If the memory is huge (e.g., consider all of Freebase or Wikipedia) one needs to organize the memories. This can be achieved with the slot choosing function *H* just described: for example, it could be designed, or trained, to store memories by entity or topic. Consequently, for efficiency at scale, *G* (and *O*) need not operate on all memories: they can operate on only a retrieved subset of candidates (only operating on memories that are on the right topic).

If the memory becomes full, a procedure for "forgetting" could also be implemented by *H* as it chooses which memory is replaced, e.g., *H* could score the utility of each memory, and overwrite the least useful.

O and *R* components: The *O* component is typically responsible for reading from memory and performing inference, e.g., calculating what are the relevant memories to perform a good response. The *R* component then produces the final response given *O*. For example in a question answering setup *O* finds relevant memories, and then R produces the actual wording of the answer, e.g., *R* could be an RNN that is conditioned on the output of *O*.

An example task is given in Figure 1. In order to answer the question *x* =

"Where is the milk now?", the *O* module first scores all memories, i.e., all previously seen sentences, against *x* to retrieve the most relevant fact, m_{o_1} = "Joe left the milk" in this case. Then, it would search the memory again to find the second relevant fact given [*x*, m_{o_1}], that is m_{o_1} = "Joe travelled to the office" (the last place Joe went before dropping the milk). Finally, the *R* module would score words given [*x*, m_{o_1}, m_{o_2}] to output *r* = "office".

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Figure 2.1. Example "story" statements, questions and answers generated by a simple simulation. Answering the question about the location of the milk requires comprehension of the actions "picked up" and "left". The questions also require comprehension of the time elements of the story, e.g., to answer "where was Joe before the office?".

2.2 End-to-End Memory Networks

End-to-End Memory Network (Sukhbaatar el al., 2015) is a form of Memory Network (Weston et al., 2015a) but unlike the model in that work, it is trained end-to-end, and hence requires significantly less supervision during training, making it more generally applicable in realistic settings. It can also be seen as an extension of RNNsearch (Bahdanau et al., 2015) to the case where multiple computational steps (hops) are performed per output symbol. The flexibility of the model allows to apply

it to tasks as diverse as (synthetic) question answering (Weston et al., 2015b) and to language modeling. For the former it is competitive with Memory Networks, but with less supervision. For the latter, on some datasets it demonstrates comparable performance to RNNs and LSTMs. In both cases the key concept of multiple computational hops yields improved results.

It takes a discrete set of inputs $x_1, ..., x_n$ that are to be stored in the memory, a query q, and outputs an answer a. Each of the x_i, q, and a contains symbols coming from a dictionary with V words. The model writes all x to the memory up to a fixed buffer size, and then finds a continuous representation for the x and q. The continuous representation is then processed via multiple hops to output a. This allows backpropagation of the error signal through multiple memory accesses back to the input during training.

Input memory representation: Suppose an input set $x_1, ..., x_i$ are stored in memory. The entire set of $\{x_i\}$ are converted into memory vectors $\{m_i\}$ of dimension d computed by embedding each x_i in a continuous space, in the simplest case, using an embedding matrix A (of size $d \times V$). The query q is also embedded (again, in the simplest case via another embedding matrix B with the same dimensions as A) to obtain an internal state u. In the embedding space, we compute the match between u and each memory m_i by taking the inner product followed by a softmax:

$$p_i = \text{Softmax}(u^t m_i) \tag{2.2}$$

where $\text{Softmax}(z_i) = e^{z_i}/\sum_j e^{z_j}$. Defined in this way p is a probability vector over the inputs.

Output memory representation: Each x_i has a corresponding output vector c_i (given in the simplest case by another embedding matrix C). The response vector from the memory o is then a sum over the transformed inputs c_i, weighted by the probability vector from the input:

$$o = \sum_i p_i c_i \tag{2.3}$$

Because the function from input to output is smooth, we can easily compute gradients and backpropagate through it. Other recently proposed forms of memory or attention take this approach (Bahdanau et al., 2015; Graves et al., 2014; Gregor et al., 2015).

Generating the final prediction: In the single layer case, the sum of the output vector o and the input embedding u is then passed through a final weight matrix W (of size $V \times d$) and a softmax to produce the predicted label:

$$\hat{a} = \text{Softmax}(W(o + u)) \tag{2.4}$$

The overall model is shown in Figure 2.2. During training, all three embedding matrices A, B and C, as well as W are jointly learned by minimizing a standard cross-entropy loss between $\hat{a}$ and the true label $\hat{a}$. Training is performed using stochastic

gradient descent.

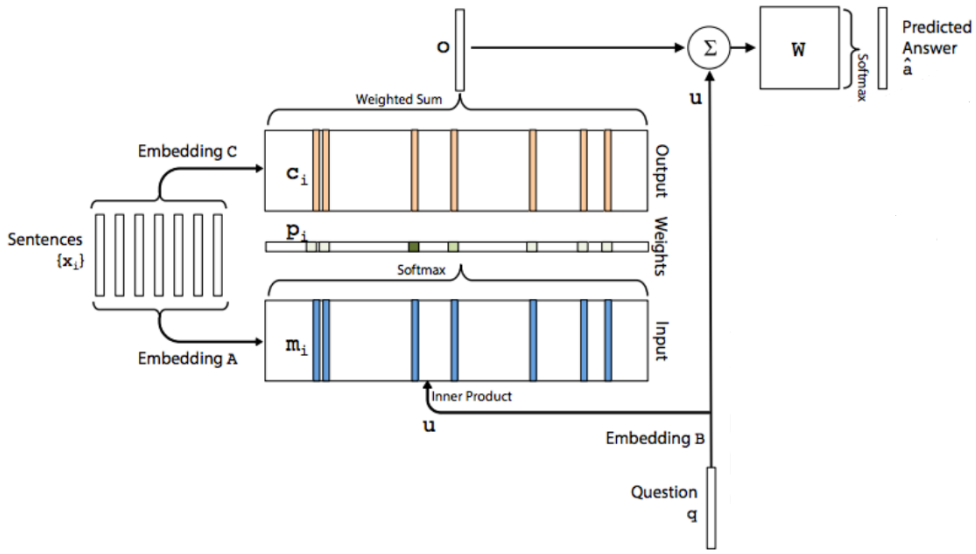

Figure 2.2. End-to-End Memory Network

2.3 Dynamic Memory Networks

The Dynamic Memory Network is a general architecture for question answering (QA) (Kumar et al., 2016). It is composed of four modules which are input module, question module, episodic memory module and answer module. Each of modules allow different aspects such as input representations or memory components to be analyzed and improved independently.

Input module: In natural language processing problems, the input is a sequence

of T_I words $w_i, \ldots, w_{T_I}$. One way to encode the input sequence is via a recurrent neural network (Elman, 1991). Word embeddings are given as inputs to the recurrent network. At each time step *t*, the network updates its hidden state h_t = $RNN(L[w_t], h_{t-1})$, where L is the embedding matrix and w_t is the word index of the *t*th word of the input sequence.

In cases where the input sequence is a single sentence, the input module outputs the hidden states of the recurrent network. In cases where the input sequence is a list of sentences, the sentences are concatenated into a long list of word tokens, inserting after each sentence an end-of-sentence token. The hidden states at each of the end-of-sentence tokens are then the final representations of the input module. In subsequent sections, the output of the input module is denoted as the sequence of T_C fact representations c, whereby c_t denotes the *t*th element in the output sequence of the input module. Note that in the case where the input is a single sentence, $T_C = T_I$. That is, the number of output representations is equal to the number of words in the sentence. In the case where the input is a list of sentences, T_C is equal the number of sentences.

In order to model the input sequences, a gated recurrent network (GRU) (Cho et al., 2014; Chung et al., 2014) is used. Assume each time step *t* has an input x_t and a hidden state h_t. The internal mechanics of the GRU is defined as:

$$z_t = \sigma(W^{(z)}x_t + U^z h_{t-1} + b^{(z)}) \tag{2.5}$$

$$r_t = \sigma(W^{(r)}x_t + U^r h_{t-1} + b^{(r)}) \tag{2.6}$$

$$\tilde{h}_t = \tanh(Wx_t + r_t \circ Uh_{t-1} + b^{(h)}) \quad (2.7)$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t \quad (2.8)$$

where $\circ$ is an element-wise product, $W^{(z)}, W^{(r)}, W \in \mathbb{R}^{n_H \times n_I}$ and $U^{(z)}, U^{(r)}, U \in \mathbb{R}^{n_H \times n_H}$. The dimensions n are hyperparameters. The above computation is abbreviated with $h_t = GRU(x_t, h_{t-1})$.

Question Module: Similar to the input sequence, the question is also most commonly given as a sequence of words in natural language processing problems. As before, the question is encoded via a recurrent neural network. Given a question of T_Q words, hidden states for the question encoder at time *t* is given by $q_t = GRU(L[w_t^Q], q_{t-1})$, L represents the word embedding matrix as in the previous section and w_t^Q represents the word index of the *t*th word in the question. The word embedding matrix can be shared across the input module and the question module. Unlike the input module, the question module produces as output the final hidden state of the recurrent network encoder: $q = q_{T_Q}$.

Episodic Memory Module: The episodic memory module iterates over representations outputted by the input module, while updating its internal episodic memory. In its general form, the episodic memory module is comprised of an attention mechanism as well as a recurrent network with which it updates

its memory. During each iteration, the attention mechanism attends over the fact representations c while taking into consideration the question representation q and the previous memory m^{i-1} to produce an episode e^i.

The episode is then used, alongside the previous memories m^{i-1}, to update the episodic memory $m^i = GRU(e^i, m^{i-1})$. The initial state of this GRU is initialized to the question vector itself: $m^0 = q$. For some tasks, it is beneficial for episodic memory module to take multiple passes over the input. After T_M passes, the final memory m^{T_M} is given to the answer module.

The iterative nature of this module allows it to attend to different inputs during each pass. It also allows for a type of transitive inference, since the first pass may uncover the need to retrieve additional facts. For instance, in the example in Figure 2.3, the question is “Where is the football?” In the first iteration, the model ought to attend to sentence 7 (John put down the football.), as the question asks about the football. Only once the model sees that John is relevant can it reason that the second iteration should retrieve where John was. Similarly, a second pass may help for sentiment analysis.

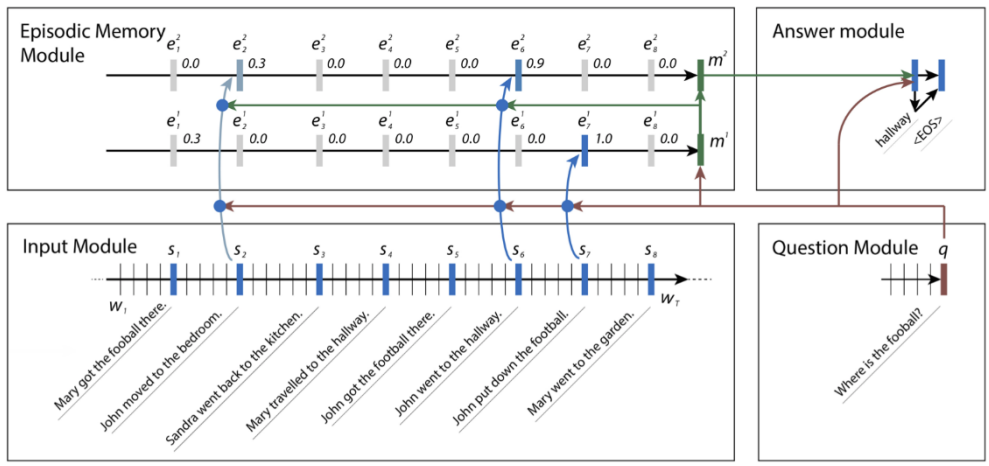

Figure 2.3. Example of an input list of sentences to the Dynamic Memory Network

In the Dynamic Memory Network, the gating function is used as attention mechanism. For each pass i, the mechanism takes as input a candidate fact c_t, a previous memory m^{i-1}, and the question q to compute a gate: $g_t^i = G(c_t, m^{i-1}, q)$

The scoring function G takes as input the feature set $z(c, m, q)$ and produces a scalar score. It is defined as a large feature vector that captures a variety of similarities between input, memory and question vectors:

$$\mathrm{z(c,m,q)} = \left[c, m, q, c \circ q, c \circ m, |c - q|, |c - m|, c^T W^{(b)} q, c^T W^{(b)} m\right] \quad (2.9)$$

where $\circ$ is an element-wise product. The function G is a simple two-layer feed forward neural network.

$$\mathrm{G(c,m,q)} = \sigma(W^{(2)} \tanh\left(W^{(1)} z(c, m, q) + b^{(1)}\right) + b^{(2)}) \quad (2.10)$$

To compute the episode for pass i, a modified GRU over the sequence of the inputs $c_1, \dots, c_{T_c}$, weighted by the gates g_i is used. The episode vector that is given to the answer module is the final state of the GRU. The equation to update the hidden states of the GRU at time t and the equation to compute the episode are, respectively:

$$h_t^i = g_t^i GRU\left(c_t h_{t-1}^i\right) + (1 - g_t^i) h_{t-1}^i \tag{2.11}$$

$$e^i = h_{T_c}^i \tag{2.12}$$

Answer Module: The answer module generates an answer given a vector. Depending on the type of task, the answer module is either triggered once at the end of the episodic memory or at each time step.

Another GRU whose initial state is initialized to the last memory $a_0 = m^{T_M}$ is used in answer module. At each time step, it takes as input the question q, last hidden state a_{t-1}, as well as the previously predicted output y_{t-1}.

$$y_t = \text{softmax}(W^{(a)} a_t) \tag{2.13}$$

$$a_t = GRU([y_{t-1}, q] a_{t-1}) \tag{2.13}$$

where we concatenate the last generated word and the question vector as the input at each time step. The output is trained with the cross-entropy error

classification of the correct sequence appended with a special end-of-sequence token.

In the sequence modeling task, we wish to label each word in the original sequence. To this end, the Dynamic Memory Network is run in the same way as above over the input words. For word t, Equation 2.12 is replaced with $e^i = h_t^i$.

Chapter 3

Conceptual Word Embedding

3.1 Related Work

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train N-grams on virtually all available data (Brants et al., 2007).

However, the simple techniques are at their limits in many tasks. For example, the amount of relevant in-domain data for automatic speech recognition is limited - the performance is usually dominated by the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques.

With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. Probably the most successful concept is to use

distributed representations of words (Hinton et al., 1986). For example, neural network based language models significantly outperform N-gram models (Bengio et al., 2003; Schwenk, 2007; Mikolov et al., 2011).

Neural network language model can be successfully trained in two steps: first, continuous word vectors are learned using simple model, and then the N-gram feedforward neural net language model is trained on top of these distributed representations of words.

Skim-gram tries to maximize classification of a word based on another word in the same sentence. More precisely, each current word is used as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. Increasing the range improves quality of the resulting word vectors, but it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, less weight are given to the distant words by sampling less from those words in training examples.

The training complexity of this architecture is proportional to

$$Q = C \times (D + D \times \log_2(V)) \tag{3.1}$$

where C is the maximum distance of the words. Thus, if we choose $C = 5$, for each training word we will select randomly a number R in range $< 1; C >$, and then use R words from history and R words from the future of the current word as correct labels. This will require us to do $R \times 2$ word classifications, with the current word as input,

and each of the R + R words as output.

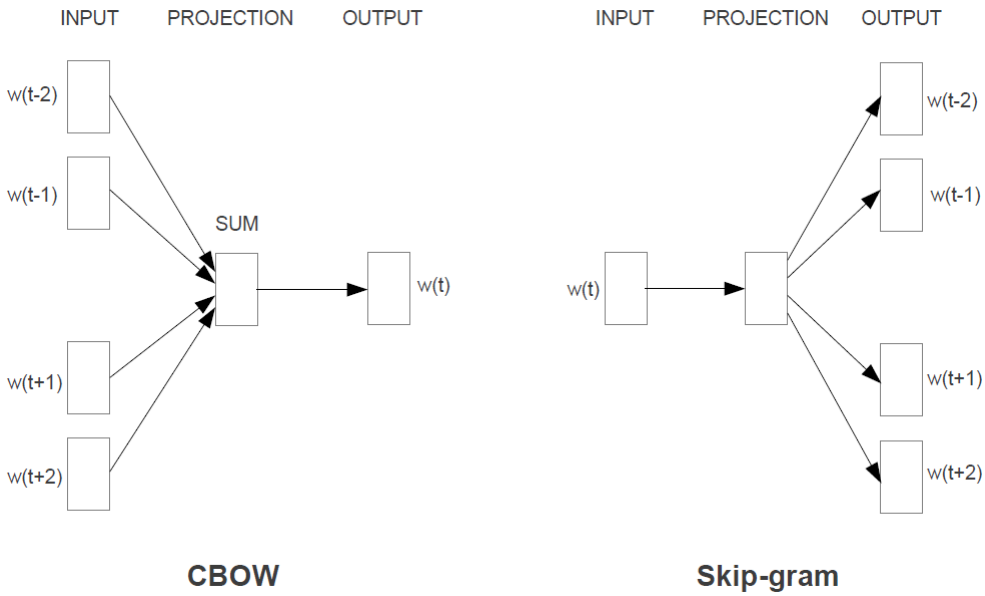

Figure 3.1. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

3.2 Dependency-Gram

Word embedding, also known as word representation, plays an increasingly vital role in building continuous word vectors based on their contexts in a large corpus. Word embedding captures both semantic and syntactic information of words, and can be used to measure word similarities, which are widely used in various IR and NLP tasks.

Most word embedding methods assume each word preserves a single vector, which is problematic due to homonymy and polysemy. Multi-prototype vector space

models (Reisinger and Mooney 2010) were proposed to cluster contexts of a word into groups, then generate a distinct prototype vector for each cluster. Following this idea, (Huang et al. 2012) proposed multi-prototype word embeddings based on neural language models (Bengio et al. 2003). Despite of their usefulness, multi-prototype word embeddings face several challenges: (1) These models generate multi-prototype vectors for each word in isolation, ignoring complicated correlations among words as well as their contexts. (2) In multi-prototype setting, contexts of a word are divided into clusters with no overlaps. In reality, a word's several senses may correlate with each other, and there is not clear semantic boundary between them.

In this dissertation, I propose a more flexible and powerful framework for multi-prototype word embeddings, namely Dependency-gram, in which dependency refers to a word taking a specific context. The basic idea of Dependency-gram is that, we allow each word to have different embeddings under different context. For example, the word apple indicates a fruit under the topic food, and indicates an IT company under the topic information technology (IT).

I use the dependency parser to obtain context, and perform collapsed Gibbs sampling (Griffiths and Steyvers2004) to iteratively assign latent topics for each word token. In this way, given a sequence of words $D = \{w_1, \ldots, w_M\}$, each word token w_i will be discriminated into a specific topic z_i, forming a word-context pair $\langle w_i, z_i \rangle$, which can be used to learn conceptual word embeddings. As shown in Figure 3.2, where the window size is 1, and w_{i-1} and w_{i+1} are conceptual words of w_i.

.

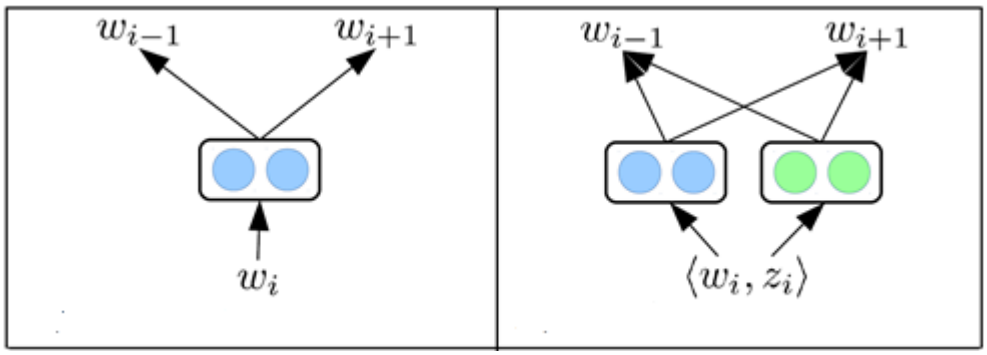

Figure 3.2. Skip-Gram and TWE models. Blue circles indicate word embeddings and green circles indicate context embeddings.

Dependency-gram aims to learn vector representations for words and contexts separately and simultaneously. For each target word with its topic $\langle w_i, z_i \rangle$, I propose Dependency-gram as follows. The objective of dependency-gram is defined to maximize the following average log probability

$$\mathcal{L}(D) = \frac{1}{M} \sum_{i=1}^{M} \sum_{-k \le c \le k, c \neq 0} \log \Pr(w_{i+c}|w_i) + \log \Pr(w_{i+c}|z_i) \qquad (3.2)$$

Compared with only using the target word w_i to predict context words in Skip-Gram, Dependency-gram also uses the context z_i of target word to predict context words. The basic idea of Dependency-gram is to regard each context as a pseudo word that appears in all positions of words assigned with this context. Hence, the vector of a context will represent the collective semantics of words under this context. In Dependency-gram, we get topical word embedding of a word w in context z by concatenating the embedding of w and z, i.e., $w^z = w \oplus z$, where $\oplus$ is the

concatenation operation, and the length of w^z is double of w or z.

TWE-1 can be used for conceptual word embedding. For each word w with its document c, TWE-1 will first infer the topic distribution $\Pr(z|w,c)$ by regarding c as a document, namely $\Pr(z|w,c) \propto \Pr(w|z)\Pr(z|c)$. With the distribution, we can further obtain the conceptual word embedding of w in c as

$$\mathbf{w}^c = \sum_{z \in T} \Pr(z|w,c)\mathbf{w}^z \tag{3.3}$$

where w_z is the embedding of word w under context z, obtained by concatenating word vector w and context vector z.

conceptual word embedding will be used for computing conceptual word similarity. Given a pair of words with their contexts, namely (w_i, c_i) and (w_j, c_j), conceptual word similarity aims to measure the similarity between the two words, which can be formalized as follows $\mathrm{S}(w_i, c_i, w_j, c_j) = (W_i^{c_i} W_j^{c_j})$, which can also be rewritten as

$$\sum_{z \in T} \sum_{z' \in T} \Pr(z|w_i, c_i) \Pr(z'|w_j, c_j) S(\mathbf{w}^z, \mathbf{w}^{z'}) \tag{3.4}$$

where $\mathrm{S}(W^z, W^{z'})$ is the similarity between W^z and $W^{z'}$.

3.3 Experimental Results

Multi-class text classification is well studied problem in NLP and IR. In this dissertation, I run the experiments on the dataset 20NewsGroup. 20NewsGroup consists of about 20,000 documents from 20 different newsgroups. I report macro-averaging precision, recall and F-measure for comparison.

I learn topical word embeddings using the training set, then generate document embeddings for both training set and test set. Afterwards, I regard document embedding vectors as document features and train a linear classifier us-ng Liblinear (Fan et al. 2008). I set the dimensions of both word and dependency embeddings as K = 400.

I consider the following baselines, bag-of-words (BOW) model, LDA, Skip-Gram, and Paragraph Vector (PV) models (Le and Mikolov 2014). The BOW model represents each document as a bag of words and the weighting scheme is TFIDF. For the TFIDF method, I select top 50,000 words according to TFIDF scores as features. LDA represents each document as its inferred topic distribution. In Skip-Gram, I build the embedding vector of a document by simply averaging over all word embedding vectors in this document. The dimension of word embeddings in Skip-Gram is also K = 400. Paragraph Vector models are document embedding models proposed most recently, including the distributed memory model (PV-DM) and the distributed bag-of-words model (PV-DBOW). PV models are reported to achieve the

state-of-the-art performance on sentiment classification (Le and Mikolov 2014).

Table 3.1 shows the evaluation results of text classification on 20NewsGroup. I can observe that Dependency-gram outperforms all baselines significantly, especially for topic models and embedding models. This indicates that our model can capture more precise semantic information of documents as compared to topic models and embedding models. Moreover, as compared to the BOW model, the Dependency-gram models manage to reduce the document feature space by 99.2 percent in this case.

Table 3.1. Evaluation results of multi-class text classification

Model	Accuracy	Precision	Recall	F-measure
BOW	79.7	79.5	79.0	79.0
LDA	72.2	70.8	70.7	70.0
Skip-Gram	75.4	75.1	74.3	74.2
PV-DM	72.4	72.1	71.5	71.5
PV-DBOW	75.4	74.9	74.3	74.3
Dependency-Gram	**80.6**	**81.0**	**80.9**	**80.1**

In order to demonstrate the characteristics of Dependency-gram, I selected several example words and used Dependency-gram to find the most similar words of these words in different topics. For comparison, I also used Skip-Gram to find similar words of these example words.

In Table 3.2, I show the most similar words of three example words, *bank*, *left* and *apple*, which are typical polysemous words. For each example word *w*, I first show the result obtained from Skip-Gram, i.e., the first line of each example word;

then I list the results under another representative context of the example word obtained from Dependency-gram, denoted as w#.

Table 3. 2. Nearest neighbor words by Skip-Gram and Dependency-gram.

Words	Similar Words
bank	citibank, investment, river
bank#	insurance, stock, investor
left	right, leave, quit
left#	moved, arrived, leave
apple	macintosh, ios, juice
apple#	moved, arrived, leave

From Table 3.2, I can observe that, similar words returned by Skip-Gram contain similar words of multiple senses of example words. This indicates that Skip-Gram combines multiple senses of a polysemous word into a unique embedding vector. In contrast, with Dependency-gram models, we can successfully discriminate word senses into multiple topics by conceptual word embeddings.

Figure 3.3. t-SNE visualization of the 500 most frequent words learned by Dependency-gram

3.4 Discussion and Summary

The success of IR and NLP tasks crucially depend on text representation, of which word representation is the foundation. Conventionally, NLP tasks usually take one-hot word representation, with each word being represented as a *W*-length vector with only one non-zero entry. The one-hot representation is simple and has been widely used as the basis of bag-of-words (BOW) document models. However, it suffers from several challenges, the most critical one of which is it cannot take the relationship between words into consideration, while in fact many words share high semantic or syntactic relations. Word embeddings have been successfully used in language models (Bengio et al., 2006; Mnih and Hinton, 2008). Word embeddings

are useful because they can encode both syntactic and semantic information of words into continuous vectors and similar words are close in vector space. Previous word embedding models are time consuming due to high computational complexity. Recently, (Mikolov et al., 2013) proposed two efficient models, Skip-Gram and continuous bag-of-words model (CBOW), to learn word embeddings from a large-scale text corpus. The training objective of CBOW is to combine the embeddings of context words to predict the target word; while Skip-Gram is to use the embedding of each target word to predict its context words (Mikolov et al. 2013). In this dissertation, I base on Skip-Gram to extend our models. In most previous word embedding models, one word owns a unique vector, which is problematic because many words have multiple senses. Hence, researchers propose multi prototype models. (Reisinger and Mooney 2010) proposed a multi-prototype vector space model, which cluster contexts of each target word into groups, and build context vectors for each cluster. Following this idea, (Huang et al. 2012) also clustered contexts, and each cluster generated a distinct prototype embedding. Besides, probabilistic models (Tian et al. 2014), bilingual resources (Guo et al. 2014) and nonparametric models (Neelakantan et al. 2014) have been explored for multi-prototype word embeddings. Most of these methods perform multi-prototype modeling for each word in isolation. On the contrary, Dependency-gram use dependency as context to discriminate word senses by considering all words and their contexts together. Dependency-gram also applicable for document embeddings. Moreover, multi-prototype models can be incorporated in Dependency-gram easily, which will be left as future work.

Chapter 4

Sentence Embedding using Context

4.1 Related Work

Developing learning algorithms for distributed compositional semantics of words has been a longstanding open problem at the intersection of language understanding and machine learning. In recent years, several approaches have been developed for learning composition operators that map word vectors to sentence vectors including recursive networks (Socher et al., 2013), recurrent networks (Hochreiter and Schmidhuber, 1997), convolutional networks (Kalchbrenner et al., 2014; Kim, 2014) and recursive-convolutional methods (Cho et al., 2014; Zhao et al., 2015) among others. All of these methods produce sentence representations that are passed to a supervised task and depend on a class label in order to backpropagate through the composition weights. Consequently, these methods learn high quality sentence representations but are tuned only for their respective task. The paragraph vector of (Le et al., 2014) is an alternative to the above models in that it can learn unsupervised sentence representations by introducing a distributed sentence indicator as part of a neural language model. The downside is at test time, inference needs to be performed to compute a new vector.

Skip-thought is a model for learning high-quality sentence vectors without a particular supervised task in mind (Kiros et al., 2015). Using word vector learning

as inspiration, it adopts an objective function that abstracts the skip-gram model of (Mikolov et al., 2013)) to the sentence level. That is, instead of using a word to predict its surrounding context, we instead encode a sentence to predict the sentences around it. Thus, any composition operator can be substituted as a sentence encoder and only the objective function becomes modified. Figure 4.1 illustrates the model.

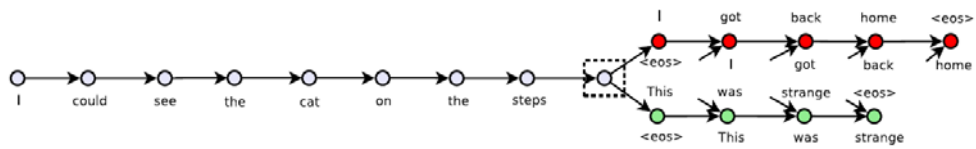

Figure 4.1 The skip-thoughts model

Skip-thoughts is a kind of encoder-decoder models. That is, an encoder maps words to a sentence vector and a decoder is used to generate the surrounding sentences. Encoder-decoder models have gained a lot of traction for neural machine translation. In this setting, an encoder is used to map e.g. an English sentence into a vector. The decoder then conditions on this vector to generate a translation for the source English sentence. The source sentence representation can also dynamically change through the use of an attention mechanism (Bahdanau et al., 2015) to take into account only the relevant words for translation at any given time. Skip-thought model use an RNN encoder with GRU (Chung et al., 2014) activations and an RNN decoder with a conditional GRU.

Given sentence tuple (s_{i-1}, s_i, s_{i+1}), let w_i^t denote the t-th word for sentence s_i and let x_i^t denote its word embedding. The model can be described in three parts: the encoder, decoder and objective function.

Encoder: Let $w_i^1, \ldots, w_i^N$ be the words in sentence s_i where N is the number of words in the sentence. At each time step, the encoder produces a hidden state h_i^t which can be interpreted as the representation of the sequence $w_i^1, \ldots, w_i^t$. The hidden state h_i^t thus represents the full sentence. To encode a sentence, we iterate the following sequence of equations (dropping the subscript *i*):

$$\mathbf{r}^t = \sigma(\mathbf{W}_r \mathbf{x}^t + \mathbf{U}_r \mathbf{h}^{t-1}) \quad (4.1)$$

$$\mathbf{z}^t = \sigma(\mathbf{W}_z \mathbf{x}^t + \mathbf{U}_z \mathbf{h}^{t-1}) \quad (4.2)$$

$$\bar{\mathbf{h}}^t = \tanh(\mathbf{W}\mathbf{x}^t + \mathbf{U}(\mathbf{r}^t \odot \mathbf{h}^{t-1})) \quad (4.3)$$

$$\mathbf{h}^t = (1 - \mathbf{z}^t) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \bar{\mathbf{h}}^t \quad (4.4)$$

where $\bar{h}^t$ is the proposed state update at time *t*, z^t is the update gate, r_t is the reset gate ($\odot$) denotes a component-wise product. Both update gates takes values between zero and one.

Decoder: The decoder is a neural language model which conditions on the encoder output $\mathbf{h}_i$. The computation is similar to that of the encoder except we introduce matrices $\mathbf{C}_z$, $\mathbf{C}_r$ and $\mathbf{C}$ that are used to bias the update gate, reset gate and hidden state computation by the sentence vector. One decoder is used for

the next sentence s_{i+1} while a second decoder is used for the previous sentence s_{i-1}. Separate parameters are used for each decoder with the exception of the vocabulary matrix **V**, which is the weight matrix connecting the decoder's hidden state for computing a distribution over words. In what follows we describe the decoder for the next sentence s_{i+1} although an analogous computation is used for the previous sentence s_{i-1}. Let h_{i+1}^t denote the hidden state of the decoder at time *t*. Decoding involves iterating through the following sequence of equations (dropping the subscript *i+1*):

$$\mathbf{r}^t = \sigma(\mathbf{W}_r^d \mathbf{x}^{t-1} + \mathbf{U}_r^d \mathbf{h}^{t-1} + \mathbf{C}_r \mathbf{h}_i) \tag{4.5}$$

$$\mathbf{z}^t = \sigma(\mathbf{W}_z^d \mathbf{x}^{t-1} + \mathbf{U}_z^d \mathbf{h}^{t-1} + \mathbf{C}_z \mathbf{h}_i) \tag{4.6}$$

$$\bar{\mathbf{h}}^t = \tanh(\mathbf{W}^d \mathbf{x}^{t-1} + \mathbf{U}^d (\mathbf{r}^t \odot \mathbf{h}^{t-1}) + \mathbf{C}\mathbf{h}_i) \tag{4.7}$$

$$\mathbf{h}_{i+1}^t = (1 - \mathbf{z}^t) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \bar{\mathbf{h}}^t \tag{4.8}$$

Given h_{i+1}^t the probability of word w_{i+1}^t given the previous *t-1* words and the encoder vector is

$$P(w_{i+1}^t | w_{i+1}^{<t}, \mathbf{h}_i) \propto \exp(\mathbf{v}_{w_{i+1}^t} \mathbf{h}_{i+1}^t) \tag{4.9}$$

where $\mathbf{v}_{w_{i+1}^t}$ denotes the row of **V** corresponding to the word w_{i+1}^t. An

analogous computation is performed for the previous sentence s_{i-1}.

Objective: Given a tuple (s_{i-1}, s_i, s_{i+1}), the objective optimized is the sum of the log-probabilities for the forward and backward sentences conditioned on the encoder representation:

$$\sum_t \log P(w_{i+1}^t | w_{i+1}^{<t}, \mathbf{h}_i) + \sum_t \log P(w_{i-1}^t | w_{i-1}^{<t}, \mathbf{h}_i) \quad (4.10)$$

The total objective is the above summed over all such training tuples.

4.2 CR-Gram

Natural language is intrinsically ambiguous, learning one vector for each word may not cover all the senses of the word. In the case of a multi-sense word, the learned vector will be around the average of all the senses of the word in the embedding space, and therefore may not be a good representation of any of the sentences. A possible solution is sentence embedding which trains a vector for each sense of a word. There are two key steps in training sense embeddings. In order to do, we need to train embedding vectors for word senses according to their contexts

Recently, sense embedding methods based on complete probabilistic models and well-defined learning objective functions (Tian et al., 2014; Jauhar et al., 2015) become more popular. These methods regard the choice of senses of the words in a

sentence as hidden variables. Learning is therefore done with expectation maximization style algorithms, which alternate between inferring word sense choices in the training corpus and learning sense embeddings.

A common problem with these methods is that they model the sense embedding of each center word dependent on the word embeddings of its context words. As I explained in chapter 3, word embedding of a polysemous word is not a good representation and may negatively influence the quality of inference and learning. Furthermore, these methods choose the sense of each word in a sentence independently, ignoring the dependency that may exist between the neighboring words. I argue that such dependency is important in word sense disambiguation and therefore helpful in learning sentence embeddings. For example, consider the sentence “He cashed a check at the bank”. Both “check” and “bank” are ambiguous here. Although the two words hint at banking related senses, the hint is not decisive (as an alternative interpretation, they may represent a check mark at a river bank). Fortunately, “cashed” is not ambiguous and it can help disambiguate “check”. However, if we consider a small context window in sense embedding, then “cashed” cannot directly help disambiguate bank”. We need to rely on the dependency between the sense choices of “check” and “bank” to disambiguate “bank”.

In this dissertation, I propose a novel probabilistic model for sentence embedding that takes into account the dependency between sense choices of neighboring words. We do not learn any word embeddings in our model and hence avoid the problem with embedding polysemous words discussed above. It contains a sequence of observable words and latent sentences and models the dependency between each word-sentence pair and between neighboring sentences in the

sequence. The energy of neighboring sentences can be modeled using existing word embedding approaches such as CBOW and Skip-gram (Mikolov et al., 2013).

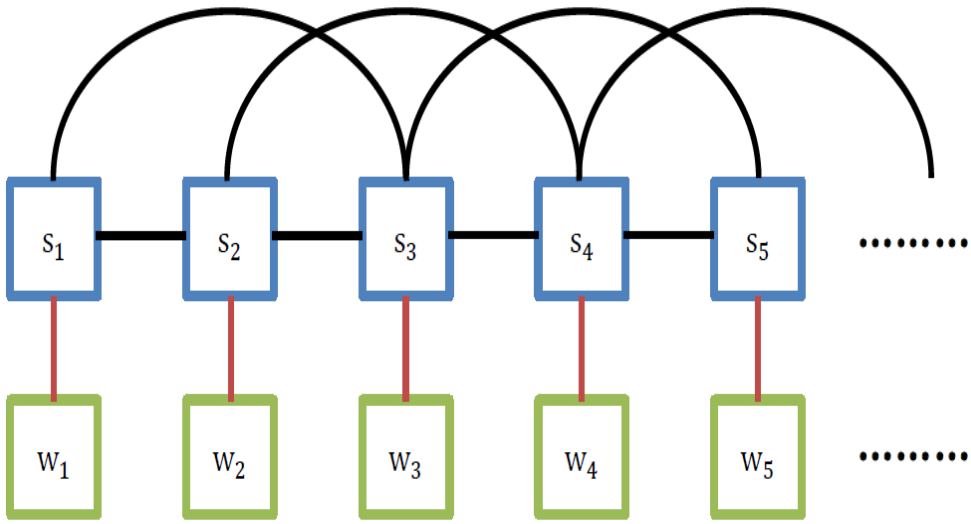

Figure 4.2 Sentence embedding model using co-reference.

In a sentence, let w_i be the *i*th word of the sentence and s_i be the latent sentence of the *i*th word. *S(w)* denotes the set of all the sentences of word *w*. Our model can be represented as a Markov network shown in Figure 1. It is similar to a high order hidden Markov model. The model contains a sequence of observable words (*w1, w2,* …) and latent senses (*s1, s2,* …). It models the dependency between each word-sentence pair and between neighboring sentences in the document. The energy function is formulated as follows:

$$E(\mathbf{w},\mathbf{s}) = \sum_i \Big(E_1(w_i, s_i) + E_2(s_{i-k}, \ldots, s_{i+k}) \Big) \qquad (4.11)$$

Here $\mathbf{w} = \{w_i | 1 \leq i \leq l\}$ is the set of words in a sentence with length l and $\mathbf{s} = \{s_i | 1 \leq i \leq l\}$ is the set of sentences. The function E_1 models the dependency between a word-sentence pair. If I assume that the sets of sentences of different words do not overlap, we can formulate E1 as follows:

$$E_1(w_i, s_i) = \begin{cases} 0 & s_i \in S(w_i) \\ +\infty & s_i \notin S(w_i) \end{cases} \tag{4.12}$$

Here we assume that all the matched word-sentence pairs have the same energy, but it would also be interesting to model the degrees of matching with different energy values in E_1. In Equation 4.11, the function E_2 models the compatibility of neighboring senses in a context window with fixed size k. Existing embedding approaches like CBOW and Skip-gram (Mikolov et al., 2013) can be used here to define E_2. The formulation using Skip-gram is as follows:

$$E_2(s_{i-k}, \ldots, s_{i+k}) = \\ - \sum_{i-k \leq j \leq i+k, j \neq i} \sigma\left(V^T(s_j) V'(s_i)\right) \tag{4.13}$$

Here $V(s)$ and $V'(s)$ are the input and output embedding vectors of sentence s. The function σ is an activation function.

Given the model and a sentence **w**, we want to infer the most likely values of the hidden variables (i.e. the optimal sense sequence of the sentence) that minimize the energy function in Equation 4.11:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} E(\mathbf{w}, \mathbf{s}) \quad) \tag{4.14}$$

We use dynamic programming to do inference which is similar to the Viterbi algorithm of the hidden Markov model.

$$\begin{aligned} m(A_{i-2k+1}, \ldots, A_i) = \min_{A_{i-2k}} \Big(& m(A_{i-2k}, \ldots, A_{i-1}) \\ & + E_1(w_i, A_i) + E_2(A_{i-2k}, \ldots, A_i) \Big) \end{aligned} \tag{4.15}$$

Once we finish the forward process, we can retrieve the best sentence sequence with a backward process. The time complexity of the algorithm is $O(n^{4k}l)$ where n is the maximal number of sentences of a word. Because most words in a typical sentence have either a single sense or far less than n sentences, the actual running time of the algorithm is very fast.

We want to learn all the input and output sense embedding vectors that optimize the following max-margin objective function:

$$\begin{aligned} \Theta^* = \arg\min_{\Theta} \sum_{\mathbf{w} \in C} \min_{\mathbf{s}} \sum_{i=1}^{\|\mathbf{w}\|} \sum_{s_{neg} \in S_{neg}(w_i)} \\ \max\Big(1 + E_1(w_i, s_i) + E_2(s_{i-k}, \ldots, s_{i+k}) - \\ E_2(s_{i-k}, \ldots, s_{i-1}, s_{neg}, s_{i+1}, \ldots, s_{i+k}), 0 \Big) \end{aligned} \tag{4.16}$$

Here Θ is the set of all the parameters including $\boldsymbol{V}$ and $\boldsymbol{V'}$ for all the sentences.

C is the set of training sentences. Our learning objective is similar to the negative sampling and max-margin objective proposed for word embedding (Collobert and Weston, 2008). $S_{neg}(w_i)$ denotes the set of negative samples of sentences of word w_i which is defined with the following strategy. For a polysemous word w_i, $S_{neg}(w_i) = S(w_i)\backslash\{s_i\}$. For the other words with a single sentence, $S_{neg}(w_i)$ is a set of randomly selected sentences of a fixed size. The objective in Equation 4.16 can be optimized by coordinate descent which in our case is equivalent to the hard Expectation-Maximization algorithm. In the hard E step, we run the inference algorithm using the current model parameters to get the optimal sense sequences of the training sentences. In the M step, with the sentences sequences s of all the sentences fixed, we learn sentence embedding vectors. Assume we use the Skip-gram model for E_2 (Equation 4.13), then the M-step objective function is as follows:

$$\begin{aligned}\Theta^* = \arg\min_{\Theta} \sum_{\mathbf{w}\in C} \sum_{i=1}^{\|\mathbf{w}\|} \sum_{i-k\le j\le i+k, j\neq i} \sum_{s_{neg}\in S_{neg}(w_i)} \\ \max\Big(1 - \sigma(V(s_j)^T V'(s_i)) \\ + \sigma(V(s_j)^T V'(s_{neg})), 0\Big)\end{aligned} \tag{4.17}$$

Here E_1 is omitted because the sense sequences produced from the E-step always have zero E_1 value.

We optimize the M-step objective function using stochastic gradient descent.

4.3 Experimental Results

I used a large collection of novels, namely the BookCorpus dataset (Zhu et al., 2015) for training our models. These are free books written by yet unpublished authors. The dataset has books in 16 different genres, e.g., Romance (2,865 books), Fantasy (1,479), Science fiction (786), Teen (430), etc. Table 4.1 highlights the summary statistics of the book corpus. Along with narratives, books contain dialogue, emotion and a wide range of interaction between characters. Furthermore, with a large enough collection the training set is not biased towards any particular domain or application.

Table 4.1. Summary statistics of the BookCorpus dataset

# of books	# of sentences	# of words	# of unique words	mean # of words per sentence
11,038	74,004,228	984,846,357	1,316,420	13

For the quantitative experiments, we report results on several classification benchmarks which are commonly used for evaluating sentence representation learning methods. We use 5 datasets: movie review sentiment (MR), customer product reviews (CR), subjectivity/objectivity classification (SUBJ), opinion polarity (MPQA) and question-type classification (TREC). 10-fold cross-validation is used for evaluation on the first 4 datasets, while TREC has a pre-defined train/test split. On these tasks, properly tuned bag-of-words models have been shown to perform exceptionally well. In particular, the NB-SVM of [37] is a fast and robust performer on these tasks. Skip-thought vectors potentially give an alternative to these

baselines being just as fast and easy to use.

Table 6 presents the results. On most tasks, CR-gram performs about as well as the bag-of-words baselines but fails to improve over methods whose sentence representations are learned directly for the task at hand. This indicates that for tasks like sentiment classification, tuning the representations, even on small datasets, are likely to perform better than learning a generic unsupervised sentence vector on much bigger datasets. Finally, we observe that the skip-thoughts-NB combination is effective, particularly on MR.

Table 4.2. Classification accuracies on several standard benchmarks.

Method	MR	CR	SUBJ	MPQA	TREC
NB-SVM	79.4	81.8	93.2	86.3	
MNB	79.0	80.0	93.6	86.3	
cBoW	77.2	79.9	91.3	86.4	87.3
GrConv	76.3	81.3	89.5	84.5	88.4
RNN	77.2	82.3	93.7	90.1	90.2
BRNN	82.3	82.6	94.2	90.3	91.0
CNN	81.5	85.0	93.4	89.6	**93.6**
AdaSent	83.1	**86.3**	**95.5**	**93.3**	92.4
Pagraph-vector	74.8	78.1	90.5	74.2	91.8
Skip-thought	76.5	80.1	93.6	87.1	92.2
CR-gram	**84.1**	83.4	91.2	90.9	92.1

As a final experiment, I applied t-SNE to skip-thought vectors extracted from BookCorpus and the visualizations are shown in Figure 4.3. Each point corresponds to a sentence. Each color corresponds to a topic.

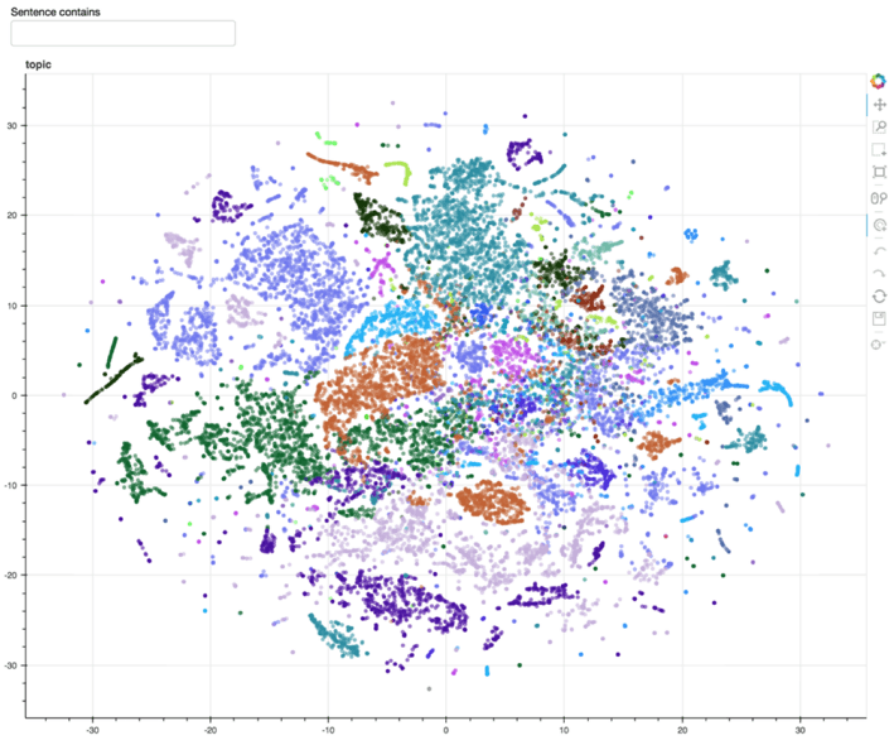

Figure 4.3. Sentences grouped based on predicted topics.

4.4 Discussion and Summary

LEARNING a good representation (or features) of input data is an important task in machine learning. In text and language processing, one such problem is learning of an embedding vector for a sentence; that is, to train a model that can automatically transform a sentence to a vector that encodes the semantic meaning of the sentence. While word embedding is learned using a loss function defined on word pairs, sentence embedding is learned using a loss function defined on sentence pairs. In the sentence embedding usually the relationship among words in the sentence, i.e.,

the context information, is taken into consideration. Therefore, sentence embedding is more suitable for tasks that require computing semantic similarities between text strings. By mapping texts into a unified semantic representation, the embedding vector can be further used for different language processing applications, such as machine translation, sentiment analysis, and information retrieval.

In machine translation, the recurrent neural networks (RNN) with Long Short-Term Memory (LSTM) cells, or the LSTM-RNN, is used to encode an English sentence into a vector, which contains the semantic meaning of the input sentence, and then another LSTM-RNN is used to generate a French (or another target language) sentence from the vector. The model is trained to best predict the output sentence. In (Le et al., 2014) a paragraph vector is learned in an unsupervised manner as a distributed representation of sentences and documents, which are then used for sentiment analysis. Sentence embedding can also be applied to information retrieval, where the contextual information are properly represented by the vectors in the same space for fuzzy text matching (Huang et al., 2013).

Inspired by the word embedding method (Mikolov et al., 2013a;, Mikolov et al., 2013b) the authors in (Le et al., 2014) proposed an unsupervised learning method to learn a paragraph vector as a distributed representation of sentences and documents, which are then used for sentiment analysis with superior performance. However, the model is not designed to capture the fine-grained sentence structure. In (Kiros et al., 2015), an unsupervised sentence embedding method is proposed with great performance on large corpus of contiguous text corpus, e.g., the BookCorpus (Zhu et al., 2015). The main idea is to encode the sentence *s(t)* and then decode previous and next sentences, i.e., *s(t-1)* and *s(t+1)*, using two separate decoders. The encoder

and decoders are RNNs with Gated Recurrent Unit (GRU) (Chung et al., 2014). However, this sentence embedding method is not designed for document retrieval task having a supervision among queries and clicked and unclicked documents. In (Socher et al., 2011), a Semi-Supervised Recursive Autoencoder (RAE) is proposed and used for sentiment prediction. Similar to our proposed method, it does not need any language specific sentiment parsers. A greedy approximation method is proposed to construct a tree structure for the input sentence. It assigns a vector per word. It can become practically problematic for large vocabularies. It also works both on unlabeled data and supervised sentiment data.

In this dissertation I propose a novel probabilistic model for learning sentence embeddings. Unlike previous work, proposed model do not learn sentence embeddings dependent on word embeddings and hence avoid the problem with inaccurate embeddings of polysemous words. Furthermore, I model the dependency between sentences of neighboring words which can help us disambiguate multiple ambiguous words in a sentence. Based on CR-gram, I derive a dynamic programming inference algorithm and an EM-style unsupervised learning algorithm which do not rely on external knowledge from any knowledge base or lexicon except that I determine the number of senses of polysemous words according to an existing sense inventory.

For the future work, I plan to try learning our model with soft EM. Besides, I plan to use dependency information in our model to improve the generality of our model. Finally, I plan to evaluate our model with more NLP tasks.

Chapter 5

Deep Memory Networks

5.1 Related Work

A number of recent efforts have explored ways to capture long-term structure within sequences using RNNs or LSTM-based models (Chung et al., 2014; Graves, 2013; Koutnık et al., 2014; Mikolov et al., 2014; Hochreiter et al., 1997). The memory in these models is the state of the network, which is latent and inherently unstable over long timescales. The LSTM-based models address this through local memory cells which lock in the network state from the past. In practice, the performance gains over carefully trained RNNs are modest.

Some of the very early work on neural networks by (Steinbuch and Piske, 1963) and (Taylor, 1959) considered a memory that performed nearest-neighbor operations on stored input vectors and then fit parametric models to the retrieved sets. This has similarities to a single layer version of our model.

The earliest recent work with a memory component that is applied to language processing is that of memory networks (Weston et al., 2015a) which adds a memory component for question answering over simple facts. Their input module computes sentence representations independently and hence cannot easily be used for other tasks such as sequence labeling. This memory network requires that supporting facts are labeled during QA training. End-to-end memory networks (Sukhbaatar et al.,

2015) do not have this limitation. In contrast to previous memory models with a variety of different functions for memory attention retrieval and representations, dynamic memory networks (Kumar et al., 2015) have shown that neural sequence models can be used for input representation, attention and response mechanisms. Sequence models naturally capture position and temporality of both the inputs and transitive reasoning steps.

Attention mechanisms allow neural network models to use a question to selectively pay attention to specific inputs. They can benefit image classification (Stollenga et al., 2014), generating captions for images (Xu et al., 2015), among others mentioned below, and machine translation (Cho et al., 2014; Bahdanau et al., 2015; Luong et al., 2015). Other recent neural architectures with memory or attention which have proposed include neural Turing machines (Graves et al., 2014), neural GPUs (Kaiser and Sutskever, 2015) and stack-augmented RNNs (Joulin and Mikolov, 2015).

Question answering involving natural language can be solved in a variety of ways to which we cannot all do justice. If the potential input is a large text corpus, QA becomes a combination of information retrieval and extraction (Yates et al., 2007). Neural approaches can include reasoning over knowledge bases, (Bordes et al., 2012; Socher et al., 2013b) or directly via sentences for trivia competitions (Iyyer et al., 2014).

5.2 Deep Memory Networks

Deep Memory Network is a memory model based on attention mechanism. It is composed of four modules which are input module, question module, episodic memory module and answer module. Each of modules allow different aspects such as input representations or memory components to be analyzed and improved independently.

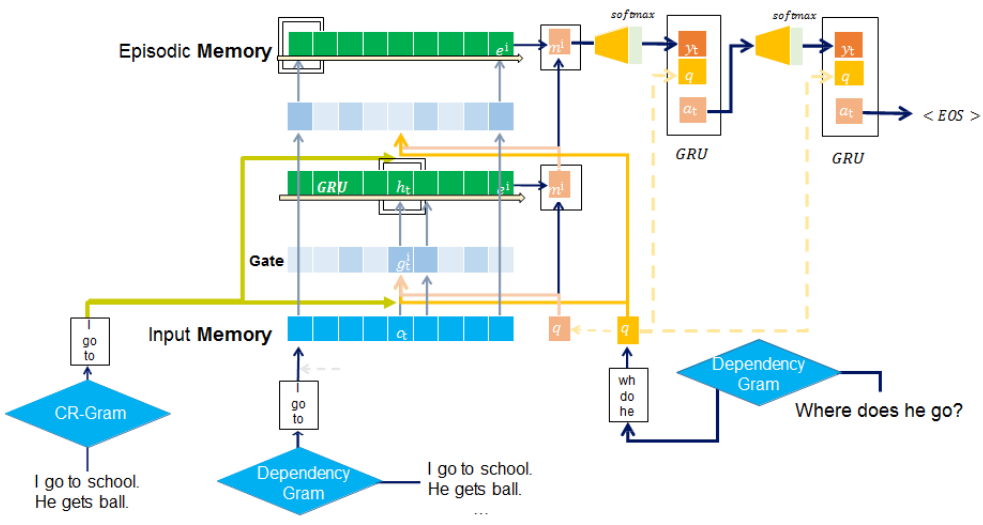

Figure 5.1 Deep Memory Network

Input module: In natural language processing problems, the input is a sequence of T_I words $w_i, \ldots, w_{T_I}$. One way to encode the input sequence is via a recurrent neural network (Elman, 1991). Word embeddings are given as inputs to the recurrent network. At each time step t, the network updates its hidden state h_t = $RNN(L[w_t], h_{t-1})$, where L is the embedding matrix and w_t is the word index of

the *t*th word of the input sequence.

In cases where the input sequence is a single sentence, the input module outputs the hidden states of the recurrent network. In cases where the input sequence is a list of sentences, the sentences are concatenated into a long list of word tokens, inserting after each sentence an end-of-sentence token. The hidden states at each of the end-of-sentence tokens are then the final representations of the input module. In subsequent sections, the output of the input module is denoted as the sequence of T_C fact representations c, whereby c_t denotes the tth element in the output sequence of the input module. Note that in the case where the input is a single sentence, $T_C = T_I$. That is, the number of output representations is equal to the number of words in the sentence. In the case where the input is a list of sentences, T_C is equal the number of sentences.

In order to model the input sequences, a gated recurrent network (GRU) (Cho et al., 2014; Chung et al., 2014) is used. Assume each time step t has an input x_t and a hidden state h_t. The internal mechanics of the GRU is defined as:

$$z_t = \sigma(W^{(z)}x_t + U^z h_{t-1} + b^{(z)}) \tag{5.1}$$

$$r_t = \sigma(W^{(r)}x_t + U^r h_{t-1} + b^{(r)}) \tag{5.2}$$

$$\tilde{h}_t = \tanh(Wx_t + r_t \circ Uh_{t-1} + b^{(h)}) \tag{5.3}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t \tag{5.4}$$

where $\circ$ is an element-wise product, $W^{(z)}, W^{(r)}, W \in \mathbb{R}^{n_H \times n_I}$ and $U^{(z)}, U^{(r)}, U \in \mathbb{R}^{n_H \times n_H}$. The dimensions n are hyperparameters. The above computation is abbreviated with $h_t = GRU(x_t, h_{t-1})$.

Sentence Module: The output of the sentence module is denoted as the sequence of T_C fact representations *s*, whereby s_t denotes the *t*th element in the output sequence of the sentence module. In order to model the input sequences, CR-gram is used. Sentence pair and between neighboring sentences in the document. The energy function is formulated as follows:

$$E(\mathbf{w}, \mathbf{s}) = \sum_i \left(E_1(w_i, s_i) + E_2(s_{i-k}, \ldots, s_{i+k}) \right) \tag{5.5}$$

Here $\mathbf{w} = \{w_i | 1 \leq i \leq l\}$ is the set of words in a sentence with length *l* and $\mathbf{s} = \{s_i | 1 \leq i \leq l\}$ is the set of sentences. The function E_1 models the dependency between a word-sentence pair. If I assume that the sets of sentences of different words do not overlap, we can formulate E_1 as follows:

$$E_1(w_i, s_i) = \begin{cases} 0 & s_i \in S(w_i) \\ +\infty & s_i \notin S(w_i) \end{cases} \tag{5.6}$$

Here we assume that all the matched word-sentence pairs have the same energy, but it would also be interesting to model the degrees of matching with different energy values in E_1. In Equation 5.5, the function E_2 models the compatibility of neighboring senses in a context window with fixed size k. Existing

embedding approaches like CBOW and Skip-gram (Mikolov et al., 2013) can be used here to define E_2. The formulation using Skip-gram is as follows:

$$E_2(s_{i-k}, \ldots, s_{i+k}) = - \sum_{i-k \leq j \leq i+k, j \neq i} \sigma\left(V^T(s_j) V'(s_i)\right) \tag{5.7}$$

Here $V(s)$ and $V'(s)$ are the input and output embedding vectors of sentence s. The function σ is an activation function. Inference is similar to the Viterbi algorithm of the hidden Markov model.

Question Module: Similar to the input sequence, the question is also most commonly given as a sequence of words in natural language processing problems. As before, the question is encoded via a recurrent neural network. Given a question of T_Q words, hidden states for the question encoder at time t is given by $q_t = GRU(L[w_t^Q], q_{t-1})$, L represents the word embedding matrix as in the previous section and w_t^Q represents the word index of the tth word in the question. The word embedding matrix can be shared across the input module and the question module. Unlike the input module, the question module produces as output the final hidden state of the recurrent network encoder: $q = q_{T_Q}$.

Episodic Memory Module: The episodic memory module iterates over representations outputted by the input module, while updating its internal episodic memory. In its general form, the episodic memory module is comprised

of an attention mechanism as well as a recurrent network with which it updates its memory. During each iteration, the attention mechanism attends over the fact representations *c* and *s* while taking into consideration the question representation *q* and the previous memory m^{i-1} to produce an episode e^i.

The episode is then used, alongside the previous memories m^{i-1}, to update the episodic memory $m^i = GRU(e^i, m^{i-1})$. The initial state of this GRU is initialized to the question vector itself: $m^0 = q$. For some tasks, it is beneficial for episodic memory module to take multiple passes over the input. After T_M passes, the final memory m^{T_M} is given to the answer module.

The iterative nature of this module allows it to attend to different inputs during each pass. It also allows for a type of transitive inference, since the first pass may uncover the need to retrieve additional facts.

In the Deep Memory Network, the gating function is used as attention mechanism. For each pass *i*, the mechanism takes as input a candidate fact c_t, sentence fact s_t, a previous memory m^{i-1}, and the question *q* to compute a gate:

$$g_t^i = G(c_t, s_t, m^{i-1}, q)$$

The scoring function *G* takes as input the feature set *z(c, s, m, q)* and produces a scalar score. It is defined as a large feature vector that captures a variety of similarities between input, memory and question vectors:

$$\mathrm{z(c,s,m,q)} = \left[c, s, m, q, c \circ q, c \circ m, s \circ q, s \circ m, |c-q|, |c-m|, |s-q|, |s-m|, c^T W^{(b)} q, c^T W^{(b)} m, s^T W^{(b)} q, s^T W^{(b)} m\right] \quad (5.8)$$

where $\circ$ is an element-wise product. The function G is a simple two-layer feed forward neural network.

$$\mathrm{G}(\mathrm{c},\mathrm{s},\mathrm{m},\mathrm{q}) = \sigma(W^{(2)}\tanh\left(W^{(1)}z(c,s,m,q) + b^{(1)}\right) + b^{(2)}) \quad (5.9)$$

To compute the episode for pass *i*, a modified GRU over the sequence of the inputs $c_1, \ldots, c_{T_c}$, weighted by the gates g_i is used. The episode vector that is given to the answer module is the final state of the GRU. The equation to update the hidden states of the GRU at time *t* and the equation to compute the episode are, respectively:

$$h_t^i = g_t^i GRU\left(c_t h_{t-1}^i\right) + (1 - g_t^i)h_{t-1}^i \quad (5.10)$$

$$e^i = h_{T_c}^i \quad (5.11)$$

Answer Module: The answer module generates an answer given a vector. Depending on the type of task, the answer module is either triggered once at the end of the episodic memory or at each time step.

Another GRU whose initial state is initialized to the last memory $a_0 = m^{T_M}$ is used in answer module. At each time step, it takes as input the question *q*, last hidden state a_{t-1}, as well as the previously predicted output y_{t-1}.

$$y_t = \mathrm{softmax}(W^{(a)}a_t) \quad (5.12)$$

$$a_t = GRU([y_{t-1}, q]a_{t-1}) \tag{5.13}$$

where we concatenate the last generated word and the question vector as the input at each time step. The output is trained with the cross-entropy error classification of the correct sequence appended with a special end-of-sequence token.

In the sequence modeling task, we wish to label each word in the original sequence. To this end, the Deep Memory Network is run in the same way as above over the input words. For word t, Equation 5.12 is replaced with $e^i = h_t^i$.

5.3 Experimental Results

5.3.1 bAbI Dataset

The Facebook bAbI dataset (Weston et al., 2015b) is a synthetic dataset for testing a model's ability to retrieve facts and reason over them. Each task tests a different skill that a question answering model ought to have, such as co-reference resolution, deduction, and induction. Showing an ability exists here is not sufficient to conclude a model would also exhibit it on real world text data. It is, however, a necessary condition.

Table 5.1 Sample statements and questions from bAbI tasks 1 to 10.

Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Task 2: Two Supporting Facts
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

Task 4: Two Argument Relations
The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
What is north of the bedroom? A: office
What is the bedroom north of? A: bathroom

Task 5: Three Argument Relations
Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Who gave the cake to Fred? A: Mary
Who did Fred give the cake to? A: Bill

Task 6: Yes/No Questions
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

Task 7: Counting
Daniel picked up the football.
Daniel dropped the football.
Daniel got the milk.
Daniel took the apple.
How many objects is Daniel holding? A: two

Task 8: Lists/Sets
Daniel picks up the football.
Daniel drops the newspaper.
Daniel picks up the milk.
John took the apple.
What is Daniel holding? milk, football

Task 9: Simple Negation
Sandra travelled to the office.
Fred is no longer in the office.
Is Fred in the office? A:no
Is Sandra in the office? A:yes

Task 10: Indefinite Knowledge
John is either in the classroom or the playground.
Sandra is in the garden.
Is John in the classroom? A:maybe
Is John in the office? A:no

Each task provides a set of training and test data, with the intention that a successful model performs well on test data. The supervision in the training set is given by the true answers to questions, and the set of relevant statements for answering a given question, which may or may not be used by the learner. Correct answers are limited to a single word (Q: Where is Mark? A: bathroom), or else a list of words (Q: What is Mark holding?) as evaluation is then clear-cut, and is measured simply as right or wrong.

All of the tasks are noiseless and a human able to read that language can potentially achieve 100% accuracy. We list the results in Table 5.2

Table 5.2 Results on bAbI dataset. Strong supervision is the additional information on important sentences to answer the question.

Task	MemNN with Strong Supervision (Kumar et al., 2015)	DMN with Strong Supervision (Kumar et al., 2015)	DeepMN without Strong Supervison
1: Single Supporting Fact	100	100	**100**
2: Two Supporting Facts	100	98.2	98
3: Three Supporting Facts	100	95.2	97
4: Two Argument Relations	100	100	**100**
5: Three Argument Relations	98	99.3	98.5
6: Yes/No Questions	100	100	**100**
7: Counting	85	96.9	**96**
8: Lists/Sets	91	96.5	**96.7**
9: Simple Negation	100	100	100
10: Indefinite Knowledge	98	97.5	97.9
11: Basic Co-reference	100	99.9	99
12: Conjunction	100	100	**100**
13: Compound Co-reference	100	99.8	98.2
14: Time Reasoning	99	100	**100**
15: Basic Deduction	100	100	**100**
16: Basic Induction	100	99.4	99
17: Positional Reasoning	65	59.6	63
18: Size Reasoning	95	95.3	94.2
19: Path Finding	36	34.5	35.1
20: Agent's Motivations	100	100	**100**
Mean Accuracy (%)	93.3	93.6	**93.6**

The Deep Memory Network shows as good as Dynamic Memory Network (DMN) and Memory Network (MemNN). But Deep Memory Network does worse than the Memory Network, which we refer to from here on as MemNN, on tasks 2 and 3, both tasks with long input sequences. I guess that this is due to the recurrent input sequence model having trouble modeling very long inputs. The MemNN does not suffer from this problem as it views each sentence separately. But both DMN and MemNN needs additional information about the important sentence which tells the answer. Without strong supervision, mean accuracy of MemNN decreases to 59.8% (Sukhbaatar et al., 2015).

In tasks 7 and 8, both tasks require the model to iteratively retrieve facts and store them in a representation that slowly incorporates more of the relevant information of the input sequence. In this situation, MemNN is worse than Deep Memory Network or DMN.

5.3.2 Stanford Sentiment Treebank

The Stanford Sentiment Treebank (SST) (Socher et al., 2013) is a popular dataset for sentiment classification. It provides phrase-level fine-grained labels, and comes with a train/development/test split. The original dataset includes 10,662 sentences, half of which were considered positive and the other half negative. Each label is extracted from a longer movie review and reflects the writer's overall intention for this review.

We present results on two formats: fine-grained root prediction, where all full sentences (root nodes) of the test set are to be classified as either very negative, negative, neutral, positive, or very positive, and binary root prediction, where all non-neutral full sentences of the test set are to be classified as either positive or negative. To train the model, we use all full sentences as well as subsample 50% of phrase-level labels every epoch. During evaluation, the model is only evaluated on the full sentences (root setup). In binary classification, neutral phrases are removed from the dataset. The Deep Memory Network achieves state-of-the-art accuracy on the binary classification task, as well as on the fine-grained classification task. Table 5.4 shows the results.

Table 5.4 Test accuracies for sentiment analysis on the Stanford Sentiment Treebank.

Method	Binary	Fine-grained
RNN	82.4	43.2
RNTN	82.9	44.4
TreeLSTM	85.4	45.7
DRNN	87.8	48.7
DCNN	86.8	48.5
DMN	88.6	52.1
DeepMN	**89.5**	**52.4**

5.3.3 SQuAD Dataset

SQuAD is composed of 100,000+ questions posed by crowd workers on 536 Wikipedia articles. The dataset is randomly partitioned into a training set (80%), a development set (10%), and a test set (10%). The answer to every question is a segment of the corresponding passage.

Two metrics are utilized to evaluate model performance of SQuAD: Exact Match (EM) and F1 score. EM measures the percentage of the prediction that matches one of the ground truth answers exactly. F1 measures the overlap between the prediction and ground truth answers which takes the maximum F1 over all of the ground truth answers.

A couple of preprocessing steps is in place to ensure that the deep neural models get the correct input. We segmented context and questions into sentences by using NLTK's Punkt sentence segmenter. Words in the sentences were then converted into symbols by using PTB Tokenizer. Syntactic information including POS tags and syntactic trees were acquired by Stanford CoreNLP utilities (Manning et al., 2014).

For the parser, we collected constituent relations and dependency relations for each word by using tree annotation and enhanced dependencies annotation respectively. To generate syntactic sequence, we removed sequences whose first node is a punctuation (“$”, “:”, “#”, “.”, “ ” ”, “ “ ”, “,”). To use dependency labels, we removed all the subcategories (e.g., “nmod:poss” ⇒ “nmod”).

Table 5.5 shows exact match and F1 scores on the dev and test set of our model and competing approaches. As we can see, our method clearly outperforms the baseline and several strong state-of-the-art systems.

Table 5.5 Performance comparison on the SQuAD test set.

Method	Dev EM	Dev F1
LR Baseline	40.0	51.0
Dynamic Chunk Reader	62.5	71.0
Match-LSTM with Ans-Ptr	64.1	73.9
Dynamic Coattention Networks	65.4	75.6
BiDAF	68.0	77.3
R-NET	71.1	79.5
Deep Memory Network	**67.2**	**76.6**

To take a closer look at how syntactic sequences affect the performance, we removed the word and sentence embedding from our model and conducted experiments based on the syntactic input along. In particular, we are interested in two aspects related to syntactic sequences. We compared the performance of the models using syntactic information along with the models without syntactic information. The predictive results in terms of EM and F1 metrics are reported in Table 5.6. From the table we see that both the word and sentence embedding are important for the models to work properly.

Table 5.6 Performance comparisons of models with the models which do not use dependency-gram or CR-gram using the SQuAD Dev set.

Method	EM	F1
DeepMN with Skip-gram	60.4	69.3
DeepMN with Skip-thought	56.4	65.1
Deep Memory Network	**67.2**	**76.6**

5.4 Discussion and Summary

Representations of Texts and Words. One of the main issues in reading comprehension is to identify the latent representations of texts and words (Chen et al., 2016; Lee et al., 2016; Wang et al., 2016; Xiong et al., 2016; Yu et al., 2016). Many pre-trained libraries such as word2vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014) have been widely used to map words into a high dimensional embedding space. Another approach is to generate embeddings by using neural networks models such as Character Embedding (Kim, 2014) and Tree-LSTM (Tai et al., 2015). One thing that worth mentioning is that although Tree-LSTM does utilize syntactic information, it targets at the phrases or sentences level embedding other than the word level embedding. Many machine comprehension models include both pre-trained embeddings and variable embeddings that can be changed through a training stage (Seo et al., 2016; Yang et al., 2016).

Deep Memory Network is a memory model based on attention mechanism. It is composed of four modules which are input module, question module, episodic memory module and answer module. Each of modules allow different aspects such as input representations or memory components to be analyzed and improved

independently. The Deep Memory Network is a potentially general architecture for a variety of NLP applications, including classification, question answering and sequence modeling. A single architecture is a first step towards a single joint model for multiple NLP problems. The Deep Memory Network is trained end-to-end with one, albeit complex, objective function. Future work will explore additional tasks, larger multi-task models and multimodal inputs and questions.

Chapter 6

Concluding Remarks

6.1 Summary and Discussion

In this dissertation, we propose the deep memory network with attention mechanism and word/sentence embedding for attention mechanism. Due to the external memory and attention mechanism, proposed method can handle various tasks in natural language processing, such as question and answering, machine comprehension and sentiment analysis. If we can cast the problems in natural language processing into question answering problems, every input data can be processed via sequence modeling process. Then attention mechanism can handle it.

Usually attention mechanism requires huge computational cost. In order to solve this problem. I proposed novel word and sentence embedding methods. Previous embedding methods only use the Markov assumption. But if we consider the language structure and make use of it, it will be very helpful to reduce the computational cost. Also it does not need strong supervision which means the additional information on important sentences.

In Chapter 3, we propose a more flexible and powerful framework for multi-prototype word embeddings, namely Dependency-gram, in which dependency refers to a word taking a specific context. The basic idea of Dependency-gram is that, we allow each word to have different embeddings under different context. For example,

the word apple indicates a fruit under the topic food, and indicates an IT company under the topic information technology (IT). We use the dependency parser to obtain context, and perform collapsed Gibbs sampling to iteratively assign latent topics for each word token.

In chapter 4, we propose a novel probabilistic model for sentence embedding that takes into account the dependency between sense choices of neighboring words. We do not learn any word embeddings in our model and hence avoid the problem with embedding polysemous words discussed above. It contains a sequence of observable words and latent sentences and models the dependency between each word-sentence pair and between neighboring sentences in the sequence. The energy of neighboring sentences can be modeled using existing word embedding approaches such as CBOW and Skip-gram.

In chapter 5, we propose the Deep Memory Network. Deep Memory Network use the syntactic relationship and structural information of language. It makes the Deep Memory Network locate the attention very efficiently and do not need strong supervision. The Deep Memory Network is a memory model based on attention mechanism. It is composed of four modules which are input module, question module, episodic memory module and answer module. Each of modules allow different aspects such as input representations or memory components to be analyzed and improved independently. It is a potentially general architecture for a variety of NLP applications, including classification, question answering and sequence modeling. A single architecture is a first step towards a single joint model for multiple NLP problems. The Deep Memory Network is trained end-to-end with one, albeit complex, objective function.

6.2 Future Work

Our work in this dissertation has demonstrated that memory based model with attention mechanism can be effectively used in natural language problems to learn representations in language. In the following, we discuss future work in several directions:

Towards multitask learning and general natural language understanding.

Multitask learning in NLP has been of interest in previous work. Proposed model shows some possibility to multitask learning. Since common concepts in language would apply to individual tasks, it is intuitive to share information across tasks. This could be seen as the first step towards a general, task-independent natural language understanding model. Even though there has been interest in multitask learning specifically with neural models, improvements remain relatively small and the best mechanism for knowledge sharing across tasks is unclear.

Exploring “less greedy” methods.

We think that one of the strengths of Deep Memory Network resides in its greedy nature which provided by CR-gram. Nonetheless, less greedy methods considering multiple decoding paths during training would be worth exploring. For this purpose, global scores for sentences would be required in order to discriminate between different solutions.

References

Mnih, Volodymyr, Nicolas Heess, and Alex Graves. Recurrent models of visual attention. *Advances in neural information processing systems*. 2014.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Weston, Jason, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015a.

Weston, Jason, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint: 1502.05698*, 2015b.

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

Gregor, Karol, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, Daan Wierstra. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Elman, Jeffrey L. Distributed representations, simple recurrent networks, and

grammatical structure. Machine learning 7.2-: 195-225, 1991.

Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS Deep Learning Workshop*, 2014.

Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus. End-to-end memory networks. *Advances in neural information processing systems*. 2015.

Kumar, Ankit, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *International Conference on Machine Learning*. 2016.

Brants, Thorsten, Ashok C. Popat, Peng Xu, Franz J. Och, Jeffrey Dean. Large language models in machine translation. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Language Learning*, 2007.

Hinton, Geoffrey E., James L. Mcclelland, and David E. Rumelhart. Distributed

representations. In: Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations,
MIT Press, 1986.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137-1155, 2003.

Schwenk, Holger, Continuous space language models. *Computer Speech and Language*, vol. 21, 2007.

T. Mikolov, A. Deoras, S. Kombrink, L. Burget, J. C ernocky´. Empirical Evaluation and Combination of Advanced Language Modeling Techniques, In: *Proceedings of Interspeech*, 2011.

Reisinger, Joseph, and Raymond J. Mooney. Multi-prototype vector-space models of word meaning. In *Proceedings of HLT-NAACL*, 109–117, 2010.

Huang, Eric H., Richard Socher, Christopher D. Manning, Andrew Y. Ng, Improving word representations via global context and multiple word prototypes. In *Proceedings of ACL*, 873–882, 2012.

Griffiths, Thomas L., and Mark Steyvers. M. Finding scientific topics. *PNAS* 101:5228–5235, 2004.

Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; and Lin, C.-J. Liblinear: A library for large linear classification. JMLR 9:1871–1874, 2008.

Le, Q. V., and Mikolov, T. Distributed representations of sentences and documents. 873–882, 2014.

Bengio, Y.; Schwenk, H.; Sen´ecal, J.-S.; Morin, F.; and Gauvain, J.-L. Neural probabilistic language models. In Innovations in Machine Learning. 137–186, 2006.

Mnih, A., and Hinton, G. E. A scalable hierarchical distributed language model. In Proceedings of NIPS, 1081–1088, 2008.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of NIPS, 3111–3119, 2013.

Socher, Richard, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

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

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *ACL*, 2014.

Kim, Yoon, Convolutional neural networks for sentence classification. *EMNLP*, 2014.

Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *SSST-8*, 2014.

Zhao, Han, Zhengdong Lu, and Pascal Poupart. Self-adaptive hierarchical sentence model. *IJCAI*, 2015.

Le, Quoc V. and Tomas Mikolov. Distributed representations of sentences and documents. *ICML*, 2014.

Kiros, Ryan, Yukun Zhu, Ruslan R. Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, Sanja Fidler, Skip-thought vectors. *Advances in neural information processing systems*. 2015.

Tian, Fei, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tie-Yan Liu. A probabilistic model for learning multi-prototype word embeddings.
In *COLING*, pages 151–160, 2014.

Jauhar, Sujay Kumar, Chris Dyer, and Eduard Hovy. Ontologically grounded multi-sense representation learning for semantic vector space models. In *Proc. NAACL*, pages 683–693. 2015.

Collobert, Ronan and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM. 2008.

Duchi, John, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7):257–269, 2010.

Zhu, Yukun, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, 2015.

Luong, Thang, Ilya Sutskever, Quoc Le, Oriol Vinyals & Wojciech Zaremba. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd annual meeting of the Association for Computational Linguistics and the 7th international joint conference on natural language processing (ACL-IJCNLP)*, volume 1: Long papers, 11–19. Beijing, China: Association for Computational Linguistics, 2015.

Socher, Richard, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng & Christopher

D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 conference on empirical methods in natural language processing (EMNLP)*, 151–161. Edinburgh, UK: Association for Computational Linguistics, 2011.

Wen, Tsung-Hsien, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke & Steve Young. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 conference on empirical methods in natural language processing (EMNLP)*, 1711–1721. Lisbon, Portugal: Association for Computational Linguistics, 2015.

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

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, Distributed representations of words and phrases and their compositionality, in *Proceedings of Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013a.

T. Mikolov, K. Chen, G. Corrado, and J. Dean, Efficient estimation of word representations in vector space, *arXiv preprint arXiv*:1301.3781, 2013b.

Q. V. Le and T. Mikolov, Distributed representations of sentences and documents, *Proceedings of the 31st International Conference on Machine Learning*, pp. 1188–

1196, 2014.

R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, Skip-thought vectors, *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, *arXiv preprint arXiv:1506.06724*, 2015.

P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, Learning deep structured semantic models for web search using clickthrough data, in Proceedings of the 22Nd ACM International Conference on Conference on Information, Knowledge Management, ser. CIKM '13. ACM, pp. 2333–2338, 2013.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

Joulin, A. and Mikolov, T. Inferring algorithmic patterns with stack-augmented recurrent nets. In *NIPS*, 2015.

Kaiser, L. and Sutskever, I. Neural GPUs Learn Algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Socher, R., Chen, D., Manning, C. D., and Ng, A. Y. Reasoning With Neural Tensor Networks For Knowledge Base Completion. In *NIPS*, 2013b.

Yates, A., Banko, M., Broadhead, M., Cafarella, M. J., Etzioni, O., and Soderland, S. Textrunner: Open information extraction on the web. In *HLT-NAACL* (Demonstrations), 2007.

Bordes, A., Glorot, X., Weston, J., and Bengio, Y. Joint Learning of Words and Meaning Representations for Open-Text Semantic Parsing. *AISTATS*, 2012.

Iyyer, M., Boyd-Graber, J., Claudino, L., Socher, R., and Daume III, H. A Neural Network for Factoid Question Answering over Paragraphs. In *EMNLP*, 2014.

A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint: 1308.0850*, 2013.

J. Koutnık, K. Greff, F. J. Gomez, and J. Schmidhuber. A clockwork RNN. In *ICML*, 2014.

T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint: 1412.7753*, 2014.

K. Steinbuch and U. Piske. Learning matrices and their applications. *IEEE*

Transactions on Electronic Computers, 12:846–862, 1963.

W. K. Taylor. Pattern recognition by means of automatic analogue apparatus. *Proceedings of The Institution of Electrical Engineers*, 106:198–209, 1959.

M. Richardson, C. J. Burges, and E. Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 193–203, 2013.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu. Attentionover-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*, 2016.

K. Lee, T. Kwiatkowski, A. Parikh, and D. Das. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*, 2016

Z. Wang, H. Mi, W. Hamza, and R. Florian. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*, 2016.

C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

Y. Yu, W. Zhang, K. Hasan, M. Yu, B. Xiang, and B. Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*, 2016.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543. Association for Computational Linguistics, October 2014.

K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In IN *PROC. ACL*, 2015.

M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

Z. Yang, B. Dhingra, Y. Yuan, J. Hu, W. W. Cohen, and R. Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*, 2016.

초 록

어텐션 기반 모델은 컴퓨터 비전 분야에서 먼저 이용되기 시작해서 최근에는 자연언어처리 문제에까지 널리 적용되고 있다. 이러한 시도들은 신경망 기반 기계번역에서 가장 먼저 적용이 되었다. 한 언어에서 다른 언어로의 번역 문제를 전체 문장을 신경망을 이용하여 인코딩한 후에 다른 언어로 디코딩하는 것이 아니라 지금 번역해야 하는 문장의 일부에 집중하여 인코딩과 디코딩을 수행함으로써 보다 효율적인 번역이 가능하다.

어텐션 기반 모델로 해결할 수 있는 기본적인 문제는 신경망이 모든 정보를 하나의 고정 길이 벡터로 인코딩하도록 강제하지 않고 입력 시퀀스를 다시 참조 할 수 있게 한다는 것이다. 어텐션 기반 모델은 히든 노드에 대한 접근을 통해서 단순히 신경망의 어느 부분에 대해서 관심을 가져야 할지 결정하는 것이 아니라 신경망에서 어느 정보를 가져올지 결정하는 것이 가능해 진다. 이러한 과정은 개별 노드가 아닌 모든 노드의 가중치 조합에 대한 검색을 통해서 구현 되기 때문에 역전파를 통한 신경망의 학습이 보다 빨라 진다는 이점이 있다.

본 논문에서는 어텐션 기반의 딥 메모리 네트워크와 이를 위한 단어와 문장의 벡터 표현 방법을 제안하였다. 제안된 방법은 외부 메모리를 이용한 어텐션 기발 모델로 인하여 질문/답변(Q&A), 기계 이해(Machine Comprehentions) 및 정서 분석과 같은 자연 언어 처리의

다양한 작업을 처리 할 수 있다. 기존의 어텐션 기반 모델들이 많은 양의 메모리와 계산량을 요구하는 문제를 해결하기 위해서 새로운 단어와 문장의 벡터 표현 방법을 제안하였다. 기존의 방법들이 마코프 가정(Markov assumption)만을 사용한 반면에 제안한 방법은 언어의 구조적 특징을 이용함으로써 어텐션 기반의 모델에서 계산 비용을 크게 줄일수 있었고, 기존의 모델들과 달리 해답문장에 대한 정보를 제공하는 강한 감독학습(strong supervision) 없이도 높은 성능을 얻을 수 있었다.

주요어 : 주의 모델, 메모리 네트워크, 딥러닝, 자연언어처리, 기계독해

학번 : 2004-30347