Neural Interface for Web-Scale Knowledge

Minjoon Seo

A dissertation

submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Hannaneh Hajishirzi, Co-Chair

Ali Farhadi, Co-Chair

Oren Etzioni

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

Neural Interface for Web-Scale Knowledge

Minjoon Seo

Co-Chair of the Supervisory Committee:

Assistant Professor Hannaneh Hajishirzi

Associate Professor Ali Farhadi

Computer Science and Engineering

Modern natural language tasks are increasingly dependent on external world knowledge. My PhD study has particularly focused on three challenges in this literature: making sense of unstructured knowledge, leveraging extremely large knowledge, and reasoning over the knowledge data. I will mainly discuss my approaches to tackle these challenges and how they can serve as an effective interface for interacting with the world knowledge. I will conclude with an argument that designing a seamless and universal knowledge interface is a crucial research goal that can better address knowledge-dependency problem in machine learning tasks.

Acknowledgement

I would like to first thank my PhD advisers, Hanna Hajishirzi and Ali Farhadi, for their guidance throughout my 7-year study at the University of Washington. They taught me how to become and what it means to be a scientist. I especially thank them for understanding and supporting my remote stay in Korea for the last two and half years of my PhD program due to mandatory military service. I also thank Oren Etzioni for his advising before he left UW to lead Allen Institute for AI (AI2) in 2014. With his support, I spent three spectacular summers enjoying Seattle sunlight at AI2. I thank Luke Zettlemoyer for gladly agreeing to be in my defense committee and providing me great advice for post-PhD career. I also thank Gina Levow for agreeing to serve as the GSR for my defense committee and Sebastian Riedel for agreeing to write a reference letter for my job search.

Throughout my PhD study, I have met many wonderful collaborators at different institutions. At the University of Washington, I enjoyed working on a short and fun project with Omer Levy and Eunsol Choi. I feel fortunate to mentor Sewon while she was an undergraduate intern with Hanna and me, and am happy to see her brilliant performance while continuing her PhD study with Hanna and Luke. At the Allen Institute for AI, an internship with Ani Kembhavi greatly inspired me what I should work on for my PhD. At Google, an internship with Tom Kwiatkowski, Ankur Parikh and Dipanjan Das became the cornerstone of my PhD study. At Naver, collaboration with Wonseok Hwang, Jinyeong Lim, Seunghyun Park, Sohee Yang, Jinhyuk Lee, Jaemin Cho and Jay Shin vastly helped me to grow my research and engineering mentorship.

I would like to thank my dear friends at UW CSE, especially Geunhong Park, Jeongjoon Park, Younghoon Kim, Junha Noh, Jaeyoon Jung, Seungyeop Han, Eric Gribkoff, and Mark Yatskar. Unfortunately I never had a chance to collaborate with them, but I had so much fun hanging out with them in and out of the Allen Center.

Last but definitely not least, Community Church of Seattle was a central part of my daily life at Seattle. Weekly gathering and service helped me to control the “PhD student stress”, and friends at the church gladly listened to my complaints and prayed for me. Rachel, thank you for your tremendous support.

DEDICATION

To my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Successfully completing real-world tasks requires one to be aware of the world knowledge. For instance, determining whether a movie review is positive or negative (a task called sentiment classification) requires one to acquire *linguistic knowledge*—what words are positive and negative, and how words syntactically associate with one another to shape the tone of the review. As such, when designing a machine learning model for a real-world task, an important question is how we will inform the model of the necessary knowledge to complete the task. For some tasks, it makes more sense to be formulated as creating a strict function that maps the input (e.g. a movie review) to the output (e.g. positive or negative) without any external attachment. Modern deep learning techniques train a finite-size parametric model by feeding a large number of input and output pairs, in the order of thousands to millions. A clear advantage of this method is that the knowledge required to complete the task does not need to be written down anywhere or directly informed but is rather *implied* through the training examples and is latently encoded in the parameters of the model. Indeed, such approach has proven to be extremely effective for many real-world tasks in various domains including natural language [Devlin et al., 2019] and vision [Krizhevsky et al., 2012].

Nevertheless, many other tasks cannot be stably formulated this way. Consider a question answering task, for example, that asks one to answer a question such as "When was Nicola Tesla born?". If we use a similar strategy, we would prepare a large number of input-output pairs for training, such as "When was Albert Einstein born?" and "What is the birthday of Niels Bohr?". It is very unlikely that, however, training with a large number of such examples would allow the model to answer the original question, unless we

see exactly the same question during training time. This is because individual facts are not *generalizable*; knowing when other physicians were born would not help one to know when Nikola Tesla was born. For these tasks, it is more natural to allow the model to directly access documents that contain task-relevant knowledge, such as Wikipedia for question answering.[1] A critical research question is, then, *how shall we design the interface between the model and the knowledge source?*. The strategy becomes especially important when the knowledge source is very large, because it would be computationally very inefficient to consider the entire knowledge source as a raw input to the model. Also, it becomes more tricky when the model needs to refer to (not just one paragraph but) several different paragraphs and documents in the knowledge source to complete the task.

My thesis will mainly discuss three neural-network-based design choices of the interface for the world knowledge: machine reader, knowledge memory, and neural reasoning. The rest of this introductory chapter will go through most of my work during my PhD and categorize them into the three topics. I will expand some of the core work and devote one chapter for each publication, with the retrospective discussion of the pros and cons of the approach. In Chapter 6, I will conclude with my hypothetical arguments on the future research direction of the neural knowledge interface.

1.1 Machine Reader

A machine's ability to comprehend knowledge from natural language is often evaluated through asking a question accompanied with a text segment that contains necessary information to answer the question. BiDAF [Seo et al., 2017a] is one of the first deep neural networks for this task that significantly outperformed traditional methods with hand-engineered features by taking the advantage of hierarchical multi-layer neural architecture and bidirectional attention mehcnaism between the document and the question. Its state-of-the-art performance in popular benchmarks such as Stanford Question Answering Dataset (SQuAD) and publicly-available code+demo[2] contributed towards the rapid advancement of machine reading models, which have even outperformed humans recently. I will expand on this work in Chapter 2. The work also led

[1] It is worth noting that an increasingly popular trend is to self-supervise a language model on a large knowledge corpus to latently encode the knowledge in the corpus [Raffel et al., 2019], which can be then fine-tuned on a target task without an external access to informational documents.

[2] Demo available at `https://allenai.github.io/bi-att-flow/`

to two follow-up projects.

In the first project [Min et al., 2017], I delved into the generalizability of BiDAF's language understanding ability. We observed that finetuning the model on a QA task in a different domain or even a different task (e.g. textual entailment) was very effective and even significantly outperformed previous works. Especially, it was clear that the lower levels of the hierarchical structure mostly learned generic language understanding ability that is quite easily generalizable. This was one of the earlier signs of the effectiveness of transfer learning in NLP, now mostly characterized by massive pretrained language models such as ELMo and BERT.

The second project [Levy et al., 2017] investigated on the model's ability to induce the structured form of knowledge underlying in the unstructured natural language. We created a simple template that maps a structured relation query (e.g. `(Seattle, country, ?)`) to a natural language question (e.g. "What country is Seattle in?") and obtained the answer (e.g. "USA") through BiDAF. Most notably, compared to traditional relation extraction, it was even able to generalize to unseen relations during training (i.e. zero-shot). This result especially indicated that machine reading comprehension and question answering with structured knowledge (e.g. Knowledge Graph) are closely tied and not independent problems.

1.2 Knowledge Memory

While a reading comprehension model is a powerful tool for accessing knowledge, its computational cost is heavily dependent on the length of the document(s) that the model needs to read. This becomes especially problematic when we want to use a web-scale corpus (e.g. Wikipedia) to answer open-domain questions. A widely-adopted work-around is to use a document retrieval system (search engine) to quickly obtain a few relevant documents, and then use the expensive neural model on the few documents to obtain the answer. This method, however, inherently suffers from the error propagation when the document retrieval system fails to retrieve correct documents. Furthermore, reading the retrieved documents with an expensive neural model is still often too computationally costly for many real applications that require low latency.

My recent research focused on reformulating machine reading comprehension as a phrase retrieval problem, where every answer candidate phrase in the document(s) is mapped to a high-dimensional vector space, so that each question can be answered by finding the nearest phrase vector to the question's embedding.

Nearest neighbor search is much faster than neural reader models, and it can even benefit from sublinear time inference through approximation methods such as Locality-Sensitive Hashing. I enumerated, embedded and indexed every phrase in English Wikipedia (60 Billion phrases, 1.5 TB) to serve an open-domain question answering system that actually *reads* the entire Wikipedia instead of only few retrieved documents [Seo et al., 2019]. Hence not only the system was several orders of magnitude faster than previous models, but also it could better answer long-tail or subjective questions such as "What to do when bored?"[3]. The most recent work [Lee et al., 2020] additionally proposed to learn a highly-sparse (10M+ dimensions) phrase representation to further improve the accuracy of the QA system. Altogether, these results signified the importance and the feasibility of web-scale natural language understanding in an end-to-end manner. I will expand on these two works [Seo et al., 2019; Lee et al., 2020] in Chapter 3 and 4, respectively.

1.3 Neural Reasoning

While standard machine reading models including BiDAF [Seo et al., 2017a] are capable of understanding lexical and syntactic cues in the text, they have limited ability in answering more complex questions that require multi-step reasoning. This has been and still is a challenging problem in the literature, so my previous research tackled the problem by simplifying its scope. In the first project, I explored solving high school geometry problems where the *domain* is constrained [Seo et al., 2015]. This allowed me to design a close-domain ontology with first-order logic, and formulate the problem as grounding each question onto the logical space (i.e. semantic parsing). The resulting model achieved a higher score than an average American student in a US college entrance exam (SAT).

The second project focused on learning several different kinds of reasoning (e.g. deduction, induction) when the *language* is simple and short [Seo et al., 2017b]. I proposed a neural recurrent unit that performed reasoning in vector space as it read a sequence of sentences, which could be trained end-to-end without predefined ontology and strong supervision. These results are helpful guidance for research towards general-purpose multi-step reasoning and also hint the possibility of a hybrid method that brings the advantages from both symbolic and neural approaches. I will expand on this work in Chapter 5.

[3]Demo available at `https://github.com/uwnlp/denspi`

Chapter 2

Machine Reader

2.1 Introduction

Reading and comprehending text is a fundamental skill set for interacting with the world knowledge, a significant portion of which is unstructured or semi-structured language data. As such, the tasks of machine reading comprehension (MRC) and question answering (QA) have gained significant popularity over the past few years within the natural language processing and computer vision communities. Systems trained end-to-end now achieve promising results on a variety of tasks in the text and image domains. One of the key factors to the advancement has been the use of neural attention mechanism, which enables the system to focus on a targeted area within a context paragraph (for MC) or within an image (for Visual QA), that is most relevant to answer the question [Weston et al., 2015; Antol et al., 2015; Xiong et al., 2016]. Attention mechanisms in previous works typically have one or more of the following characteristics. First, the computed attention weights are often used to extract the most relevant information from the context for answering the question by summarizing the context into a fixed-size vector. Second, in the text domain, they are often temporally dynamic, whereby the attention weights at the current time step are a function of the attended vector at the previous time step. Third, they are usually uni-directional, wherein the query attends on the context paragraph or the image.

In this chapter, I will discuss our Bi-Directional Attention Flow (BIDAF) network [Seo et al., 2017a], a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different

levels of granularity (Figure 2.1). BIDAF includes character-level, word-level, and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation. Our attention mechanism offers following improvements to the previously popular attention paradigms. First, our attention layer is not used to summarize the context paragraph into a fixed-size vector. Instead, the attention is computed for every time step, and the attended vector at each time step, along with the representations from previous layers, is allowed to *flow* through to the subsequent modeling layer. This reduces the information loss caused by early summarization. Second, we use a memory-*less* attention mechanism. That is, while we iteratively compute attention through time as in Bahdanau et al. [2015], the attention at each time step is a function of only the query and the context paragraph at the current time step and does not directly depend on the attention at the previous time step. We hypothesize that this simplification leads to the division of labor between the attention layer and the modeling layer. It forces the attention layer to focus on learning the attention between the query and the context, and enables the modeling layer to focus on learning the interaction within the query-aware context representation (the output of the attention layer). It also allows the attention at each time step to be unaffected from incorrect attendances at previous time steps. Our experiments show that memory-less attention gives a clear advantage over dynamic attention. Third, we use attention mechanisms in both directions, query-to-context and context-to-query, which provide complimentary information to each other.

Our BIDAF model[1] outperforms all previous approaches on the highly-competitive Stanford Question Answering Dataset (SQuAD) test set leaderboard at the time of submission. With a modification to only the output layer, BIDAF achieves the state-of-the-art results on the CNN/DailyMail cloze test. We also provide an in-depth ablation study of our model on the SQuAD development set, visualize the intermediate feature spaces in our model, and analyse its performance as compared to a more traditional language model for machine comprehension [Rajpurkar et al., 2016].

2.2 Model

Our machine comprehension model is a hierarchical multi-stage process and consists of six layers (Figure 2.1):

[1] Our code and interactive demo are available at: `allenai.github.io/bi-att-flow/`

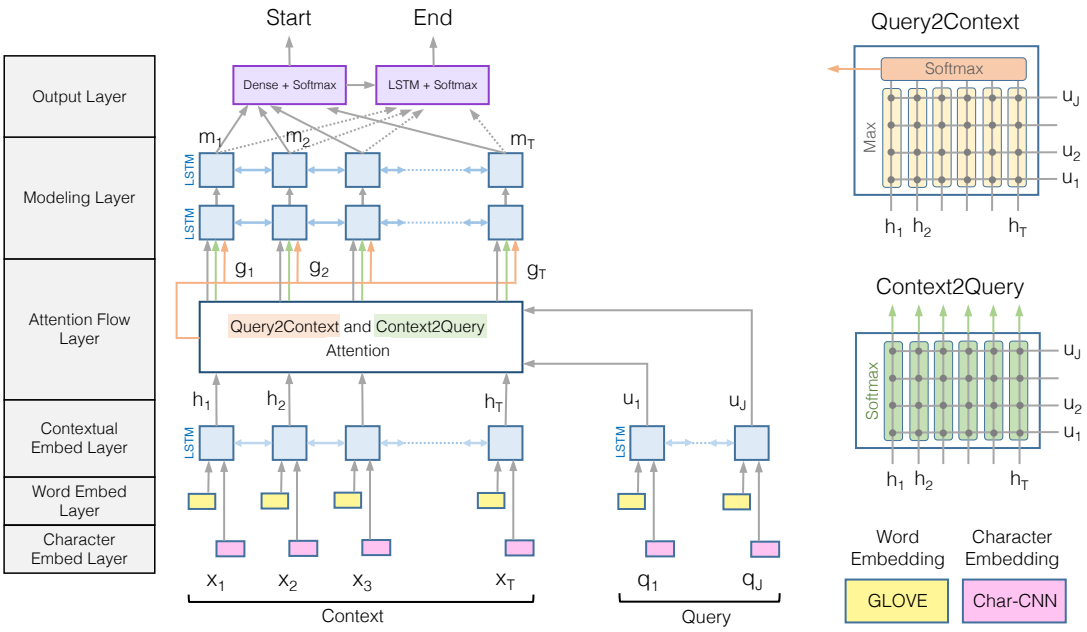


Figure 2.1: BiDirectional Attention Flow Model *(best viewed in color)*

- **Character Embedding Layer** maps each word to a vector space using character-level CNNs.
- **Word Embedding Layer** maps each word to a vector space using a pre-trained word embedding model.
- **Contextual Embedding Layer** utilizes contextual cues from surrounding words to refine the embedding of the words. These first three layers are applied to both the query and context.
- **Attention Flow Layer** couples the query and context vectors and produces a set of query-aware feature vectors for each word in the context.
- **Modeling Layer** employs a Recurrent Neural Network to scan the context.
- **Output Layer** provides an answer to the query.

1. Character Embedding Layer. Character embedding layer is responsible for mapping each word to a high-dimensional vector space. Let $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_T\}$ and $\{\boldsymbol{q}_1, ..., \boldsymbol{q}_J\}$ represent the words in the input context paragraph and query, respectively. Following Kim [2014], we obtain the character-level embedding of each word using Convolutional Neural Networks (CNN). Characters are embedded into vectors, which can be

considered as 1D inputs to the CNN, and whose size is the input channel size of the CNN. The outputs of the CNN are max-pooled over the entire width to obtain a fixed-size vector for each word.

2. Word Embedding Layer. Word embedding layer also maps each word to a high-dimensional vector space. We use pre-trained word vectors, GloVe [Pennington et al., 2014], to obtain the fixed word embedding of each word.

The concatenation of the character and word embedding vectors is passed to a two-layer Highway Network [Srivastava et al., 2015]. The outputs of the Highway Network are two sequences of d-dimensional vectors, or more conveniently, two matrices: $\mathbf{X} \in \mathbb{R}^{d \times T}$ for the context and $\mathbf{Q} \in \mathbb{R}^{d \times J}$ for the query.

3. Contextual Embedding Layer. We use a Long Short-Term Memory Network (LSTM) [Hochreiter and Schmidhuber, 1997] on top of the embeddings provided by the previous layers to model the temporal interactions between words. We place an LSTM in both directions, and concatenate the outputs of the two LSTMs. Hence we obtain $\mathbf{H} \in \mathbb{R}^{2d \times T}$ from the context word vectors $\mathbf{X}$, and $\mathbf{U} \in \mathbb{R}^{2d \times J}$ from query word vectors $\mathbf{Q}$. Note that each column vector of $\mathbf{H}$ and $\mathbf{U}$ is $2d$-dimensional because of the concatenation of the outputs of the forward and backward LSTMs, each with d-dimensional output.

It is worth noting that the first three layers of the model are computing features from the query and context at different levels of granularity, akin to the multi-stage feature computation of convolutional neural networks in the computer vision field.

4. Attention Flow Layer. Attention flow layer is responsible for linking and fusing information from the context and the query words. Unlike previously popular attention mechanisms [Weston et al., 2015; Hill et al., 2016; Sordoni et al., 2016; Shen et al., 2016], the attention flow layer is not used to summarize the query and context into single feature vectors. Instead, the attention vector at each time step, along with the embeddings from previous layers, are allowed to flow through to the subsequent modeling layer. This reduces the information loss caused by early summarization.

The inputs to the layer are contextual vector representations of the context $\mathbf{H}$ and the query $\mathbf{U}$. The outputs of the layer are the query-aware vector representations of the context words, $\mathbf{G}$, along with the contextual embeddings from the previous layer.

In this layer, we compute attentions in two directions: from context to query as well as from query to context. Both of these attentions, which will be discussed below, are derived from a shared similarity matrix, $\mathbf{S} \in \mathbb{R}^{T \times J}$, between the contextual embeddings of the context ($\mathbf{H}$) and the query ($\mathbf{U}$), where $\mathbf{S}_{tj}$ indicates the similarity between t-th context word and j-th query word. The similarity matrix is computed by

$$\mathbf{S}_{tj} = \alpha(\mathbf{H}_{:t}, \mathbf{U}_{:j}) \in \mathbb{R} \tag{2.1}$$

where α is a trainable scalar function that encodes the similarity between its two input vectors, $\mathbf{H}_{:t}$ is t-th column vector of $\mathbf{H}$, and $\mathbf{U}_{:j}$ is j-th column vector of $\mathbf{U}$, We choose $\alpha(\mathbf{h}, \mathbf{u}) = \mathbf{w}_{(\mathbf{S})}^{\top}[\mathbf{h}; \mathbf{u}; \mathbf{h} \circ \mathbf{u}]$, where $\mathbf{w}_{(\mathbf{S})} \in \mathbb{R}^{6d}$ is a trainable weight vector, $\circ$ is elementwise multiplication, $[;]$ is vector concatenation across row, and implicit multiplication is matrix multiplication. Now we use $\mathbf{S}$ to obtain the attentions and the attended vectors in both directions.

Context-to-query Attention. Context-to-query (C2Q) attention signifies which query words are most relevant to each context word. Let $\mathbf{a}_t \in \mathbb{R}^J$ represent the attention weights on the query words by t-th context word, $\sum \mathbf{a}_{tj} = 1$ for all t. The attention weight is computed by $\mathbf{a}_t = \mathrm{softmax}(\mathbf{S}_{t:}) \in \mathbb{R}^J$, and subsequently each attended query vector is $\tilde{\mathbf{U}}_{:t} = \sum_j \mathbf{a}_{tj}\mathbf{U}_{:j}$. Hence $\tilde{\mathbf{U}}$ is a $2d$-by-T matrix containing the attended query vectors for the entire context.

Query-to-context Attention. Query-to-context (Q2C) attention signifies which context words have the closest similarity to one of the query words and are hence critical for answering the query. We obtain the attention weights on the context words by $\mathbf{b} = \mathrm{softmax}(\max_{col}(\mathbf{S})) \in \mathbb{R}^T$, where the maximum function ($\max_{col}$) is performed across the column. Then the attended context vector is $\tilde{\mathbf{h}} = \sum_t \mathbf{b}_t \mathbf{H}_{:t} \in \mathbb{R}^{2d}$. This vector indicates the weighted sum of the most important words in the context with respect to the query. $\tilde{\mathbf{h}}$ is tiled T times across the column, thus giving $\tilde{\mathbf{H}} \in \mathbb{R}^{2d \times T}$.

Finally, the contextual embeddings and the attention vectors are combined together to yield $\mathbf{G}$, where each column vector can be considered as the query-aware representation of each context word. We define $\mathbf{G}$ by

$$\mathbf{G}_{:t} = \boldsymbol{\beta}(\mathbf{H}_{:t}, \tilde{\mathbf{U}}_{:t}, \tilde{\mathbf{H}}_{:t}) \in \mathbb{R}^{d_{\mathbf{G}}} \tag{2.2}$$

where $\mathbf{G}_{:t}$ is the t-th column vector (corresponding to t-th context word), $\boldsymbol{\beta}$ is a trainable vector function that

fuses its (three) input vectors, and $d_{\mathbf{G}}$ is the output dimension of the $\boldsymbol{\beta}$ function. While the $\boldsymbol{\beta}$ function can be an arbitrary trainable neural network, such as multi-layer perceptron, a simple concatenation as following still shows good performance in our experiments: $\boldsymbol{\beta}(\mathbf{h}, \tilde{\mathbf{u}}, \tilde{\mathbf{h}}) = [\mathbf{h}; \tilde{\mathbf{u}}; \mathbf{h} \circ \tilde{\mathbf{u}}; \mathbf{h} \circ \tilde{\mathbf{h}}] \in \mathbb{R}^{8d \times T}$ (i.e., $d_{\mathbf{G}} = 8d$).

5. Modeling Layer. The input to the modeling layer is $\mathbf{G}$, which encodes the query-aware representations of context words. The output of the modeling layer captures the interaction among the context words conditioned on the query. This is different from the contextual embedding layer, which captures the interaction among context words independent of the query. We use two layers of bi-directional LSTM, with the output size of d for each direction. Hence we obtain a matrix $\mathbf{M} \in \mathbb{R}^{2d \times T}$, which is passed onto the output layer to predict the answer. Each column vector of $\mathbf{M}$ is expected to contain contextual information about the word with respect to the entire context paragraph and the query.

6. Output Layer. The output layer is application-specific. The modular nature of BIDAF allows us to easily swap out the output layer based on the task, with the rest of the architecture remaining exactly the same. Here, we describe the output layer for the QA task. In section 2.5, we use a slight modification of this output layer for cloze-style comprehension.

The QA task requires the model to find a sub-phrase of the paragraph to answer the query. The phrase is derived by predicting the start and the end indices of the phrase in the paragraph. We obtain the probability distribution of the start index over the entire paragraph by

$$\mathbf{p}^1 = \mathrm{softmax}(\mathbf{w}_{(\mathbf{p}^1)}^\top [\mathbf{G}; \mathbf{M}]), \tag{2.3}$$

where $\mathbf{w}_{(\mathbf{p}^1)} \in \mathbb{R}^{10d}$ is a trainable weight vector. For the end index of the answer phrase, we pass $\mathbf{M}$ to another bidirectional LSTM layer and obtain $\mathbf{M}^2 \in \mathbb{R}^{2d \times T}$. Then we use $\mathbf{M}^2$ to obtain the probability distribution of the end index in a similar manner:

$$\mathbf{p}^2 = \mathrm{softmax}(\mathbf{w}_{(\mathbf{p}^2)}^\top [\mathbf{G}; \mathbf{M}^2]) \tag{2.4}$$

Training. We define the training loss (to be minimized) as the sum of the negative log probabilities of

the true start and end indices by the predicted distributions, averaged over all examples:

$$L(\theta) = -\frac{1}{N}\sum_{i}^{N} \log(\mathbf{p}^1_{y_i^1}) + \log(\mathbf{p}^2_{y_i^2}) \tag{2.5}$$

where θ is the set of all trainable weights in the model (the weights and biases of CNN filters and LSTM cells, $\mathbf{w}_{(\mathbf{S})}$, $\mathbf{w}_{(\mathbf{p}^1)}$ and $\mathbf{w}_{(\mathbf{p}^2)}$), N is the number of examples in the dataset, y_i^1 and y_i^2 are the true start and end indices of the i-th example, respectively, and $\mathbf{p}_k$ indicates the k-th value of the vector $\mathbf{p}$.

Test. The answer span (k, l) where $k \leq l$ with the maximum value of $\mathbf{p}^1_k \mathbf{p}^2_l$ is chosen, which can be computed in linear time with dynamic programming.

2.3 Related Work

Machine comprehension. A significant contributor to the advancement of MC models has been the availability of large datasets. Early datasets such as MCTest [Richardson et al., 2013a] were too small to train end-to-end neural models. Massive cloze test datasets (CNN/DailyMail by Hermann et al. [2015] and Childrens Book Test by [Hill et al., 2016]), enabled the application of deep neural architectures to this task. More recently, Rajpurkar et al. [2016] released the Stanford Question Answering (SQuAD) dataset with over 100,000 questions. We evaluate the performance of our comprehension system on both SQuAD and CNN/DailyMail datasets.

Previous works in end-to-end machine comprehension use attention mechanisms in three distinct ways. The first group (largely inspired by [Bahdanau et al., 2015]) uses a dynamic attention mechanism, in which the attention weights are updated dynamically given the query and the context as well as the previous attention. [Hermann et al., 2015] argue that the dynamic attention model performs better than using a single fixed query vector to attend on context words on CNN & DailyMail datasets. [Chen et al., 2016] show that simply using bilinear term for computing the attention weights in the same model drastically improves the accuracy. [Wang and Jiang, 2017] reverse the direction of the attention (attending on query words as the context RNN progresses) for SQuAD. In contrast to these models, BIDAF uses a memory-less attention mechanism.

The second group computes the attention weights once, which are then fed into an output layer for final

prediction (e.g., [Kadlec et al., 2016]). Attention-over-attention model [Cui et al., 2016] uses a 2D similarity matrix between the query and context words (similar to Equation 2.1) to compute the weighted average of query-to-context attention. In contrast to these models, BIDAF does not summarize the two modalities in the attention layer and instead lets the attention vectors flow into the modeling (RNN) layer.

The third group (considered as variants of Memory Network [Weston et al., 2015]) repeats computing an attention vector between the query and the context through multiple layers, typically referred to as *multi-hop* [Sordoni et al., 2016; Dhingra et al., 2017]. [Shen et al., 2016] combine Memory Networks with Reinforcement Learning in order to dynamically control the number of hops. One can also extend our BIDAF model to incorporate multiple hops.

Visual question answering. The task of question answering has also gained a lot of interest in the computer vision community. Early works on visual question answering (VQA) involved encoding the question using an RNN, encoding the image using a CNN and combining them to answer the question [Antol et al., 2015; Malinowski et al., 2015]. Attention mechanisms have also been successfully employed for the VQA task and can be broadly clustered based on the granularity of their attention and the approach to construct the attention matrix. At the coarse level of granularity, the question attends to different patches in the image [Zhu et al., 2016; Xiong et al., 2016]. At a finer level, each question word attends to each image patch and the highest attention value for each spatial location [Xu and Saenko, 2016] is adopted. A hybrid approach is to combine questions representations at multiple levels of granularity (unigrams, bigrams, trigrams) [Yang et al., 2015b]. Several approaches to constructing the attention matrix have been used including element-wise product, element-wise sum, concatenation and Multimodal Compact Bilinear Pooling [Fukui et al., 2016].

Lu et al. [2016] have recently shown that in addition to attending from the question to image patches, attending from the image back to the question words provides an improvement on the VQA task. This finding in the visual domain is consistent with our finding in the language domain, where our bi-directional attention between the query and context provides improved results. Their model, however, uses the attention weights directly in the output layer and does not take advantage of the attention flow to the modeling layer.

2.4 Question Answering Experiments

In this section, we evaluate our model on the task of question answering using the recently released SQuAD [Rajpurkar et al., 2016], which has gained a huge attention over a few months. In the next section, we evaluate our model on the task of cloze-style reading comprehension.

Dataset. SQuAD is a machine comprehension dataset on a large set of Wikipedia articles, with more than 100,000 questions. The answer to each question is always a span in the context. The model is given a credit if its answer matches one of the human written answers. Two metrics are used to evaluate models: Exact Match (EM) and a softer metric, F1 score, which measures the weighted average of the precision and recall rate at character level. The dataset consists of 90k/10k train/dev question-context tuples with a large hidden test set. It is one of the largest available MC datasets with human-written questions and serves as a great test bed for our model.

Model Details. The model architecture used for this task is depicted in Figure 2.1. Each paragraph and question are tokenized by a regular-expression-based word tokenizer (PTB Tokenizer) and fed into the model. We use 100 1D filters for CNN char embedding, each with a width of 5. The hidden state size (d) of the model is 100. The model has about 2.6 million parameters. We use the AdaDelta [Zeiler, 2012] optimizer, with a minibatch size of 60 and an initial learning rate of 0.5, for 12 epochs. A dropout [Srivastava et al., 2014] rate of 0.2 is used for the CNN, all LSTM layers, and the linear transformation before the softmax for the answers. During training, the moving averages of all weights of the model are maintained with the exponential decay rate of 0.999. At test time, the moving averages instead of the raw weights are used. The training process takes roughly 20 hours on a single Titan X GPU. We also train an ensemble model consisting of 12 training runs with the identical architecture and hyper-parameters. At test time, we choose the answer with the highest sum of confidence scores amongst the 12 runs for each question.

Results. The results of our model and competing approaches on the hidden test are summarized in Table 2.1a. BIDAF (ensemble) achieves an EM score of 73.3 and an F1 score of 81.1, outperforming all previous approaches.

	Single Model		Ensemble	
	EM	F1	EM	F1
Logistic Regression Baseline[a]	40.4	51.0	-	-
Dynamic Chunk Reader[b]	62.5	71.0	-	-
Fine-Grained Gating[c]	62.5	73.3	-	-
Match-LSTM[d]	64.7	73.7	67.9	77.0
Multi-Perspective Matching[e]	65.5	75.1	68.2	77.2
Dynamic Coattention Networks[f]	66.2	75.9	71.6	80.4
R-Net[g]	**68.4**	**77.5**	72.1	79.7
BIDAF (Ours)	68.0	77.3	**73.3**	**81.1**

(a) Results on the SQuAD test set

	EM	F1
No char embedding	65.0	75.4
No word embedding	55.5	66.8
No C2Q attention	57.2	67.7
No Q2C attention	63.6	73.7
Dynamic attention	63.5	73.6
BIDAF (single)	67.7	77.3
BIDAF (ensemble)	72.6	80.7

(b) Ablations on the SQuAD dev set

Table 2.1: (2.1a) The performance of our model BIDAF and competing approaches by [Rajpurkar et al., 2016][a], [Yu et al., 2016][b], [Yang et al., 2017][c], [Wang and Jiang, 2017][d], IBM Watson[e] (unpublished), [Xiong et al., 2017][f], and Microsoft Research Asia[g] (unpublished) on the SQuAD test set. A concurrent work by [Lee et al., 2016] does not report the test scores. All results shown here reflect the SQuAD leaderboard (`stanford-qa.com`) as of 6 Dec 2016, 12pm PST. (2.1b) The performance of our model and its ablations on the SQuAD dev set. Ablation results are presented only for single runs.

Ablations. Table 2.1b shows the performance of our model and its ablations on the SQuAD dev set. Both char-level and word-level embeddings contribute towards the model's performance. We conjecture that word-level embedding is better at representing the semantics of each word as a whole, while char-level embedding can better handle out-of-vocab (OOV) or rare words. To evaluate bi-directional attention, we remove C2Q and Q2C attentions. For ablating C2Q attention, we replace the attended question vector $\tilde{\mathbf{U}}$ with the average of the output vectors of the question's contextual embedding layer (LSTM). C2Q attention proves to be critical with a drop of more than 10 points on both metrics. For ablating Q2C attention, the output of the attention layer, $\mathbf{G}$, does not include terms that have the attended Q2C vectors, $\tilde{\mathbf{H}}$. To evaluate the attention flow, we study a dynamic attention model, where the attention is dynamically computed within the modeling layer's LSTM, following previous work [Bahdanau et al., 2015; Wang and Jiang, 2017]. This is in contrast with our approach, where the attention is pre-computed before flowing to the modeling layer. Despite being a simpler attention mechanism, our proposed static attention outperforms the dynamically computed attention by more than 3 points. We conjecture that separating out the attention layer results in a richer set of features computed in the first 4 layers which are then incorporated by the modeling layer. We also show the performance of BIDAF with several different definitions of α and $\boldsymbol{\beta}$ functions.

Layer	Query	Closest words in the Context using cosine similarity
Word	When	when, When, After, after, He, he, But, but, before, Before
Contextual	When	When, when, 1945, 1991, 1971, 1967, 1990, 1972, 1965, 1953
Word	Where	Where, where, It, IT, it, they, They, that, That, city
Contextual	Where	where, Where, Rotterdam, area, Nearby, location, outside, Area, across, locations
Word	Who	Who, who, He, he, had, have, she, She, They, they
Contextual	Who	who, whose, whom, Guiscard, person, John, Thomas, families, Elway, Louis
Word	city	City, city, town, Town, Capital, capital, district, cities, province, Downtown
Contextual	city	city, City, Angeles, Paris, Prague, Chicago, Port, Pittsburgh, London, Manhattan
Word	January	July, December, June, October, January, September, February, April, November, March
Contextual	January	January, March, December, August, December, July, July, July, March, December
Word	Seahawks	Seahawks, Broncos, 49ers, Ravens, Chargers, Steelers, quarterback, Vikings, Colts, NFL
Contextual	Seahawks	Seahawks, Broncos, Panthers, Vikings, Packers, Ravens, Patriots, Falcons, Steelers, Chargers
Word	date	date, dates, until, Until, June, July, Year, year, December, deadline
Contextual	date	date, dates, December, July, January, October, June, November, March, February

Table 2.2: Closest context words to a given query word, using a cosine similarity metric computed in the Word Embedding feature space and the Phrase Embedding feature space.

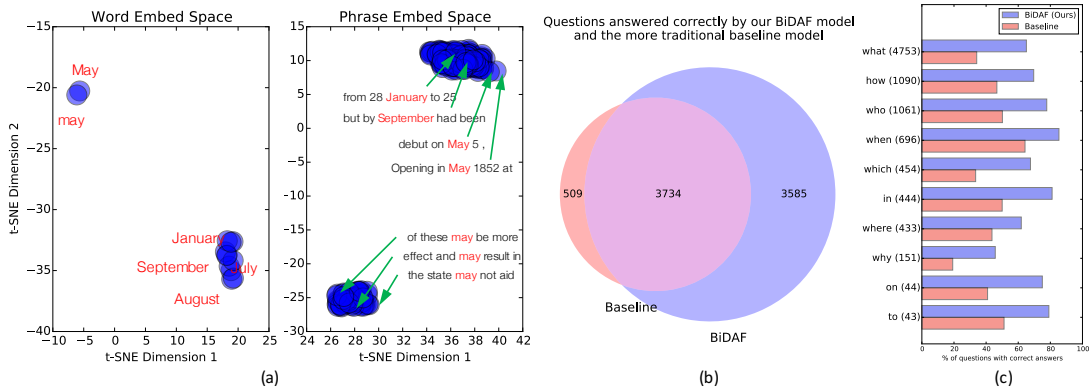


Figure 2.2: (a) t-SNE visualizations of the *months* names embedded in the two feature spaces. The contextual embedding layer is able to distinguish the two usages of the word *May* using context from the surrounding text. (b) Venn diagram of the questions answered correctly by our model and the *more traditional* baseline [Rajpurkar et al., 2016]. (c) Correctly answered questions broken down by the 10 most frequent first words in the question.

Visualizations. We now provide a qualitative analysis of our model on the SQuAD dev set. First, we visualize the feature spaces after the word and contextual embedding layers. These two layers are responsible for aligning the embeddings between the query and context words which are the inputs to the subsequent attention layer. To visualize the embeddings, we choose a few frequent query words in the dev data and look at the context words that have the highest cosine similarity to the query words (Table 2.2). At the word embedding layer, query words such as *When*, *Where* and *Who* are not well aligned to possible answers in the context, but this dramatically changes in the contextual embedding layer which has access to context from surrounding words and is just 1 layer below the attention layer. *When* begins to match years, *Where* matches locations, and *Who* matches names.

Figure 2.3: Attention matrices for question-context tuples. The left palette shows the context paragraph (correct answer in red and underlined), the middle palette shows the attention matrix (each row is a question word, each column is a context word), and the right palette shows the top attention points for each question word, above a threshold.

We also visualize these two feature spaces using t-SNE in Figure 2.2. t-SNE is performed on a large fraction of dev data but we only plot data points corresponding to the months of the year. An interesting pattern emerges in the Word space, where *May* is separated from the rest of the months because *May* has multiple meanings in the English language. The contextual embedding layer uses contextual cues from surrounding words and is able to separate the usages of the word *May*. Finally we visualize the attention matrices for some question-context tuples in the dev data in Figure 2.3. In the first example, *Where* matches locations and in the second example, *many* matches quantities and numerical symbols. Also, entities in the question typically attend to the same entities in the context, thus providing a feature for the model to localize possible answers.

Discussions. We analyse the performance of our our model with a traditional language-feature-based baseline [Rajpurkar et al., 2016]. Figure 2.2b shows a Venn diagram of the dev set questions correctly answered by the models. Our model is able to answer more than 86% of the questions correctly answered by the baseline. The 14% that are incorrectly answered does not have a clear pattern. This suggests that neural architectures are able to exploit much of the information captured by the language features. We also break this comparison down by the first words in the questions (Figure 2.2c). Our model outperforms the traditional

baseline comfortably in every category.

Error Analysis. We randomly select 50 incorrect questions (based on EM) and categorize them into 6 classes. 50% of errors are due to the imprecise boundaries of the answers, 28% involve syntactic complications and ambiguities, 14% are paraphrase problems, 4% require external knowledge, 2% need multiple sentences to answer, and 2% are due to mistakes during tokenization.

2.5 Cloze Test Experiments

We also evaluate our model on the task of cloze-style reading comprehension using the CNN and Daily Mail datasets [Hermann et al., 2015].

Dataset. In a cloze test, the reader is asked to fill in words that have been removed from a passage, for measuring one's ability to comprehend text. Hermann et al. [2015] have recently compiled a massive Cloze-style comprehension dataset, consisting of 300k/4k/3k and 879k/65k/53k (train/dev/test) examples from CNN and DailyMail news articles, respectively. Each example has a news article and an incomplete sentence extracted from the human-written summary of the article. To distinguish this task from language modeling and force one to refer to the article to predict the correct missing word, the missing word is always a named entity, anonymized with a random ID. Also, the IDs must be shuffled constantly during test, which is also critical for full anonymization.

Model Details. The model architecture used for this task is very similar to that for SQuAD (Section 2.4) with only a few small changes to adapt it to the cloze test. Since each answer in the CNN/DailyMail datasets is always a single word (entity), we only need to predict the start index ($\mathbf{p}^1$); the prediction for the end index ($\mathbf{p}^2$) is omitted from the loss function. Also, we mask out all non-entity words in the final classification layer so that they are forced to be excluded from possible answers. Another important difference from SQuAD is that the answer entity might appear more than once in the context paragraph. To address this, we follow a similar strategy from Kadlec et al. [2016]. During training, after we obtain $\mathbf{p}^1$, we sum all probability values of the entity instances in the context that correspond to the correct answer. Then the loss function is computed from the summed probability. We use a minibatch size of 48 and train for 8 epochs, with early

stop when the accuracy on validation data starts to drop. Inspired by the window-based method [Hill et al., 2016], we split each article into short sentences where each sentence is a 19-word window around each entity (hence the same word might appear in multiple sentences). The RNNs in BIDAF are not feed-forwarded or back-propagated across sentences, which speed up the training process by parallelization. The entire training process takes roughly 60 hours on eight Titan X GPUs. The other hyper-parameters are identical to the model described in Section 2.4.

Results. The results of our single-run models and competing approaches on the CNN/DailyMail datasets are summarized in Table 2.3. * indicates ensemble methods. BIDAF outperforms previous single-run models on both datasets for both val and test data. On the DailyMail test, our single-run model even outperforms the best ensemble method.

	CNN		DailyMail	
	val	test	val	test
Attentive Reader [Hermann et al., 2015]	61.6	63.0	70.5	69.0
MemNN [Hill et al., 2016]	63.4	6.8	-	-
AS Reader [Kadlec et al., 2016]	68.6	69.5	75.0	73.9
DER Network [Kobayashi et al., 2016]	71.3	72.9	-	-
Iterative Attention [Sordoni et al., 2016]	72.6	73.3	-	-
EpiReader [Trischler et al., 2016]	73.4	74.0	-	-
Stanford AR [Chen et al., 2016]	73.8	73.6	77.6	76.6
GAReader [Dhingra et al., 2017]	73.0	73.8	76.7	75.7
AoA Reader [Cui et al., 2016]	73.1	74.4	-	-
ReasoNet [Shen et al., 2016]	72.9	74.7	77.6	76.6
BIDAF (Ours)	**76.3**	**76.9**	**80.3**	**79.6**
MemNN* [Hill et al., 2016]	66.2	69.4	-	-
ASReader* [Kadlec et al., 2016]	73.9	75.4	78.7	77.7
Iterative Attention* [Sordoni et al., 2016]	74.5	75.7	-	-
GA Reader* [Dhingra et al., 2017]	76.4	77.4	79.1	78.1
Stanford AR* [Chen et al., 2016]	77.2	77.6	80.2	79.2

Table 2.3: Results on CNN/DailyMail datasets. We also include the results of previous ensemble methods (marked with *) for completeness.

2.6 Conclusion

In this chapter, I discussed the concept of machine reading and introduced BIDAF [Seo et al., 2017a], a multi-stage hierarchical process that represents the context at different levels of granularity and uses a bi-

directional attention flow mechanism to achieve a query-aware context representation without early summarization. The experimental evaluations show that our model achieves the state-of-the-art results in Stanford Question Answering Dataset (SQuAD) and CNN/DailyMail cloze test. The ablation analyses demonstrate the importance of each component in our model. The visualizations and discussions show that our model is learning a suitable representation for MC and is capable of answering complex questions by attending to correct locations in the given paragraph.

Chapter 3

Neural Knowledge Memory

3.1 Introduction

Extractive open-domain question answering (QA) is usually referred to the task of answering an arbitrary factoid question (such as "Where was Barack Obama born?") by reading a general web text (such as Wikipedia). This is an extension of the reading comprehension task of selecting an answer phrase to a question given an evidence document as discussed in Chapter 2. To make a scalable open-domain QA system, One can leverage a search engine to filter the web-scale evidence to a few documents, in which the answer span can be extracted using a reading comprehension model [Chen et al., 2017]. However, the accuracy of the final QA system is bounded by the performance of the search engine due to the pipeline

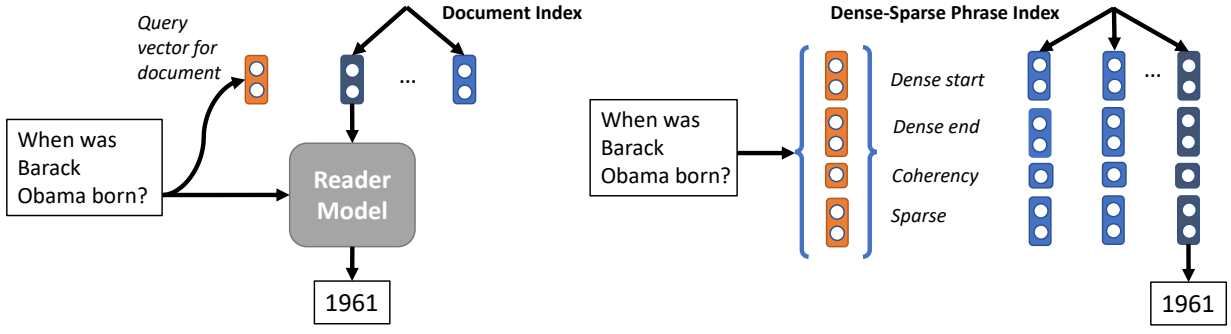


Figure 3.1: An illustrative comparison between a pipelined QA system, e.g. DrQA [Chen et al., 2017] (left) and our proposed Dense-Sparse Phrase Index (right) for open-domain QA, best viewed in color. Dark blue vectors indicate the retrieved items from the index by the query.

nature of the search process. What is more, running a neural reading comprehension model [Seo et al., 2017a] on a few documents is still computationally costly, since it needs to process the evidence document for every new question at inference time. This often requires multi-GPU-seconds or tens to hundreds of CPU-seconds—BERT [Devlin et al., 2019] can process only a few thousand words per second on an Nvidia P40 GPU.

In this chapter, I will discuss Dense-Sparse Phrase Index[Seo et al., 2019], an *indexable* query-agnostic phrase representation model for real-time open-domain QA. The phrase representations are indexed offline using time- and memory-efficient training and storage. During inference time, the input question is mapped to the same representation space, and the phrase with maximum inner product semearch is retrieved. This phrase index can be considered as a *neural knowledge memory* that encodes the world knowledge in a memory format with neural (distributed) key representations.

Our phrase representation combines both dense and sparse vectors. Dense vectors are effective for encoding local syntactic and semantic cues leveraging recent advances in contextualized text encoding [Devlin et al., 2019], while sparse vectors are superior at encoding precise lexical information. The independent encoding of the document phrases and the question enables real-time inference; there is no need to re-encode documents for every question. Encoding phrases as a function of their start and end tokens facilitates indexable representations with under 2TB for up to 60 billion phrases in Wikipedia. Moreover, approximate nearest neighbor search on indexable representations allows fast and direct retrieval in a web-scale environment.

Experiments on SQuAD-Open [Chen et al., 2017] show that DENSPI is comparable or better than most state-of-the-art open-domain QA systems on Wikipedia with up to 6000x reduced computational cost on RAM. In our end-to-end benchmark, this translates into at least 58x faster query inference including disk access time.

At the web scale, every detail of the training, indexing, and inference needs to be carefully designed. For reproducibility under an academic setting, we discuss optimization strategies for reducing time and memory usage during each stage in Section 4.3.3. This enables us to start from scratch and fully deploy the model with a 4-GPU, 64GB memory, 2 TB SATA[1] SSD server in a week.

[1]Further speed up is expected when taking a full advantage of a faster interface such as PCIe.

3.2 Related Work

Open-domain question answering Creating a system that can answer an open-domain factoid question has been a significant interest to both academic and industrial communities. The problem is largely approached from two subfields: knowledge base (KB) and text (document) retrieval. Earlier work in large-scale question answering [Berant et al., 2013] has focused on answering questions from a structured KB such as Freebase [Bollacker et al., 2008]. These approaches usually achieve a high precision, but their scope is limited to the ontology of the knowledge graph. While KB QA is undoubtedly an important part of open-domain QA, we mainly discuss literature in text-based QA, which is most relevant to our work.

Sentence-level QA has been studied since early 2000s, some of the most notable datasets being TrecQA [Voorhees and Tice, 2000] and WikiQA [Yang et al., 2015a]. See Prager et al. [2007] for a comprehensive overview of early work. With the advancement of deep neural networks and the availability of massive QA datasets such as SQuAD [Rajpurkar et al., 2016], open-domain phrase-level question answering has gained a great popularity [Shen et al., 2017; Raiman and Miller, 2017; Min et al., 2018; Raison et al., 2018; Das et al., 2019], where a few (5-10) documents relevant to the question are retrieved and then a deep neural model finds the answer in the document. Most previous work on open-domain QA has focused on mitigating error propagation of retriever models in a pipelined setting [Chu-Carroll et al., 2012]. For instance, retrieved documents could be re-ranked using reinforcement learning [Wang et al., 2018a], distant supervision [Lin et al., 2018], or multi-task learning [Nishida et al., 2018]. Several studies have also shown that answer aggregation modules could improve performance of the pipelined models [Wang et al., 2018b; Lee et al., 2018].

Our work is motivated by Seo et al. [2018] and adopts the concept and the advantage of using phrase index for large-scale question answering, though they only experiment in a close-domain (vanilla SQuAD) setup.

Approximate similarity search Sublinear-time search for the nearest neighbor from a large collection of vectors is a significant interest to the information retrieval community [Deerwester et al., 1990; Blei et al., 2003]. In metric space (L1 or L2), one of the most classic search algorithms is Locality-Sensitive Hashing (LSH) [Gionis et al., 1999], which uses a data-independent hashing function to map nearby vectors to the same cell. Stronger empirical performance has been observed with a data-dependent hashing function [An-

doni and Razenshteyn, 2015] or k-means clustering for defining the cells. More recently, graph-based search algorithms [Malkov and Yashunin, 2018] have gained popularity as well. In non-metric space such as inner product, asymmetric Locality Sensitive Hashing (aLSH) [Shrivastava and Li, 2014] is considered, where maximizing inner product search can be transformed into minimizing L2 distance by appending a single dimension to the vectors. While these methods are widely used for dense vectors, for extremely sparse data (such as document tf-idf with stop words), it is often more efficient to construct an inverted index and only look up items that have common hot dimensions with the query.

Generative question answering Mapping the phrases in a document to a common vector space to that of the questions can be viewed as an exhaustive enumeration of all possible questions that can be asked on the document in the vector space, but without a surface-form decoder. It is worth noting that generative question answering [Lewis and Fan, 2019] has the opposite property; while it has a surface-form decoder by definition, it cannot easily enumerate a compact list of all possible semantically-unique questions.

Memory networks One can view the phrase index as an external memory [Weston et al., 2015; Sukhbaatar et al., 2015] where the key is the phrase vector and the value is the corresponding answer phrase span.

3.3 Overview

In this section, we formally define "open-domain question answering" and provide an overview of our proposed model.

3.3.1 Problem Definition

In this paper, we are interested in the task of answering factoid questions from a large collection of web documents in real-time. This is often referred to as open-domain question answering (QA). We formally formulate the task as follows. We are given a fixed set of (Wikipedia) documents $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^K$ (where K is the number of documents, often on the order of millions), and each document $\boldsymbol{x}^k$ has N_k words, $\boldsymbol{x}^k_1, \ldots, \boldsymbol{x}^k_{N_k}$. The task is to find the answer $\boldsymbol{a}$ to the question $\boldsymbol{q} = \boldsymbol{q}_1, \ldots, \boldsymbol{q}_S$. Then an open-domain QA model is a scoring function F for each candidate phrase span $\boldsymbol{x}^k_{i:j}$ such that $\boldsymbol{a} = \arg\max_{k,i,j} F(\boldsymbol{x}^k_{i:j}, \boldsymbol{q})$.

Scalability challenge While the formulation is straightforward, argmax-ing over the entire corpus is computationally prohibitive, especially if F is a complex neural model. To avoid the computational bottleneck, previous open-domain QA models adopt pipeline-based methods; that is, as illustrated in Figure 3.1 left, a fast retrieval-based model is used (e.g. tf-idf) to obtain a few relevant documents to the question, and then a neural QA model is used to extract the exact answer from the documents [Chen et al., 2017]. However, the method is not *efficient* enough for real-time usage because the neural QA needs to re-encode all the documents for every new question, which is computationally expensive even with modern GPUs, and not suitable for low-latency applications.

3.3.2 Encoding and Indexing Phrases

Motivated by Seo et al. [2018], our model encodes query-agnostic representations of text spans in Wikipedia offline and obtains the answer in real-time by performing nearest neighbor search at inference time. We represent each phrase span in the corpus (Wikipedia) with a dense vector and a sparse vector. The dense vector is effective for encoding syntactic and semantic cues, while the sparse vector is good at encoding precise lexical information. That is, the embedding of each span (i, j) in the document $\boldsymbol{x}^k$ is represented with

$$\mathbf{x}^k_{i:j} = [\mathbf{d}^k_{i:j}, \mathbf{s}^k_{i:j}] \in \mathbb{R}^{d^{\text{d}}+d^{\text{s}}} \tag{3.1}$$

where $\mathbf{d}^k_{i:j} \in \mathbb{R}^{d^{\text{d}}}$ is the dense vector and $\mathbf{s}^k_{i:j} \in \mathbb{R}^{d^{\text{s}}}$ is the sparse vector for span (i, j) in the k-th document. Note that $d^{\text{d}} \ll d^{\text{s}}$. This is also illustrated in Figure 3.1 right. Text span embeddings ($\mathbf{x}^k_{i:j}$) for all possible i, j, k pairs with $j - i < J$, where J is maximum span length (i.e. all possible spans from all documents in Wikipedia), are pre-computed and stored as a *phrase index*. Then at inference time, we embed each question into the same vector space, $\mathbf{q} = [\mathbf{d}', \mathbf{s}'] \in \mathbb{R}^{d^{\text{d}}+d^{\text{s}}}$. Finally, the answer to the question is obtained by finding the maximum inner product between $\mathbf{q}$ and $\mathbf{x}^k_{i:j}$,

$$k^*, i^*, j^* = \arg\max_{k,i,j} \mathbf{q} \cdot \mathbf{x}^k_{i:j}. \tag{3.2}$$

Needlessly to say, designing a good phrase representation model is crucial, which will be discussed in Section 5.2.1. Also, while inner product search is much more efficient than re-encoding documents, the

search space is still quite large, such that exact search on the entire corpus is still undesirable. We discuss how we perform inner product search efficiently in Section 4.3.3.

3.4 Phrase and Question Embedding

In this section, we first explain the embedding model for the dense vector in Section 3.4.1. Then we describe the embedding model for the sparse vector in Section 3.4.2. Lastly, we describe the corresponding question embedding model to be queried on the phrase index in Section 3.4.3. For the brevity of the notations, we omit the superscript k in this section since we do not learn cross-document relationships.

3.4.1 Dense Model

The dense vector is responsible for encoding syntactic or semantic information of the phrase with respect to its context. We decompose the dense vector $\mathbf{d}^k_{i:j}$ (Equation 3.1) into three components: a vector that corresponds to the start position of the phrase, a vector that corresponds to the end position, and a scalar value that measures the coherency between the start and the end vectors. Representing phrases as a function of start and end vectors allows us to efficiently compute and store the vectors instead of enumerating all possible phrases (discussed in Section 3.5.2).[2]

The coherency scalar allows us to avoid non-constituent phrases during inference. For instance, consider a sentence such as "Barack Obama was the 44th President of the US. He was also a lawyer." and when a question "What was Barack Obama's job?" is asked. Since both answers "44th President of the US" and "lawyer" are technically correct, we might end up with the answer that spans from "44th" to "lawyer" if we model start and end vectors independently. The coherency scalar helps us avoid this by modeling it as a function of the start position and the end position. Formally, after phrase vector decomposition into dense and sparse, we can expand the dense vector into

$$\mathbf{d}_{i:j} = [\mathbf{a}_i, \mathbf{b}_j, c_{i,j}] \in \mathbb{R}^{2d^\flat+1} \tag{3.3}$$

where $\mathbf{a}_i, \mathbf{b}_j \in \mathbb{R}^{d^\flat}$ are the start and end vectors for the i-th and j-th words of the document, respectively;

[2] Our phrase encoding is analogous to how existing QA systems obtain the answer by predicting its start and the end positions.

and $c_{i,j} \in \mathbb{R}$ is the phrasal coherency scalar between i-th and j-th positions (hence $d^{\text{d}} = 2d^{\text{b}} + 1$).

To obtain these components of the dense vector, we leverage available contextualized word representations, in particular BERT-large [Devlin et al., 2019], which is pretrained on a large corpus (Wikipedia and BookCorpus) and has proved to be very powerful in numerous natural language tasks. BERT maps a sequence of the document tokens $\boldsymbol{x} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ to a sequence of corresponding vectors (i.e. a matrix) $\mathbf{H} = [\mathbf{h}_1; \ldots; \mathbf{h}_N] \in \mathbb{R}^{N \times d}$, where N is the length of the input sequence, d is the hidden state size, and $[;]$ is vertical concatenation. We obtain the three components of the dense vector from these contextualized word representations.

We fine-tune BERT to learn a d-dimensional vector $\mathbf{h}_i$ for encoding each token $\boldsymbol{x}_i$. Every token encoding is split into four vectors $\mathbf{h}_i = [\mathbf{h}_i^1, \mathbf{h}_i^2, \mathbf{h}_i^3, \mathbf{h}_i^4] \in \mathbb{R}^d$, where $[,]$ is a column-wise concatenation. Then we obtain the dense start vector $\mathbf{a}_i$ from $\mathbf{h}_i^1$ and dense end vector $\mathbf{b}_j$ from $\mathbf{h}_j^2$. Lastly, we obtain the coherency scalar $c_{i,j}^k$ from the inner product of $\mathbf{h}_i^3$ and $\mathbf{h}_j^4$. The inner product allows more coherent phrases to have more similar start and end encodings. That is,

$$\mathbf{d}_{i:j} = [\mathbf{h}_i^1, \mathbf{h}_j^2, \mathbf{h}_i^3 \cdot \mathbf{h}_j^4] \in \mathbb{R}^{2d^{\text{b}}+1} \tag{3.4}$$

where $\cdot$ indicates inner product operation and $\mathbf{h}_i^1, \mathbf{h}_j^2 \in \mathbb{R}^{d^{\text{b}}}$ and $\mathbf{h}_i^3, \mathbf{h}_j^4 \in \mathbb{R}^{d^{\text{c}}}$ (hence $2d^{\text{b}} + 2d^{\text{c}} = d$).

3.4.2 Sparse Model

We use term-frequency-based encoding to obtain the sparse embedding $\mathbf{s}_{i:j}^k$ for each phrase. Specifically, we largely follow DrQA [Chen et al., 2017] to construct 2-gram-based tf-idf, resulting in a highly sparse representation ($d^{\text{d}} \approx$16M) for each document. The sparse vectors are normalized so that the inner product effectively becomes cosine similarity. We also compute a paragraph-level sparse vector in a similar way and add it to each document sparse vector for a higher sensitivity to local information. Note that, however, unlike DrQA where the sparse vector is merely used to retrieve a few (5-10) documents, we concatenate the sparse vector to the dense vector to form a standalone single phrase vector as in Equation 3.1.

3.4.3 Question Embedding Model

At inference, the question is encoded as $\mathbf{q} = [\mathbf{d}', \mathbf{s}'] = [\mathbf{a}', \mathbf{b}', c', \mathbf{s}']$ with the same number of components as the phrase index. To obtain the dense query vector $\mathbf{d}' = [\mathbf{a}', \mathbf{b}', c']$, we use a special token (`[CLS]` for BERT) which is appended to the front of the question words (i.e. input question words are $\boldsymbol{q} = \texttt{[CLS]}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_S$). This allows us to model the dense query embedding differently from the dense embedding in the phrase index while sharing all parameters of the BERT encoder. That is, given the contextualized word representations of the question, we obtain the the dense query vector by

$$\mathbf{p}' = [\mathbf{h}_1'^1, \mathbf{h}_1'^2, \mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4], \tag{3.5}$$

where $\mathbf{h}_1'^1$ is the encoding corresponding to the (first) special token and we obtain the others in a similar way. To obtain the sparse query vector $\mathbf{s}'$, we use the same tf-idf embedding model (Section 3.4.2) on the entire query.

3.5 Training, Indexing & Search

Open-domain QA is a web-scale experiment, dealing with billions of words in Wikipedia while aiming for real-time inference. Hence (1) training the models, (2) indexing the embeddings, and (3) performing inner product search at inference time are non-trivial for both (a) computational time and (b) memory efficiency. In particular, we carry out this section assuming that we have a constrained hardware environment of 4 P40 GPUs, 128 GB RAM, 16 cores and 2 TB of SATA SSD storage, to promote reproducibility of our experiments under academic setting.[3]

3.5.1 Training

As discussed in Section 3.4.2, the sparse embedding model is trained in an unsupervised manner. For training the dense embedding model, instead of directly optimizing for Equation 3.2 on entire Wikipedia, which is computationally prohibitive, we provide the golden paragraph to each question during training (i.e. SQuAD v1.1 setting).

[3]Training takes 16 hours (64-GPU hours) and indexing takes 5 days (500 GPU-hours).

Given the dense phrase and question embeddings, we first expand Equation 3.2 by substituting Equation 3.4 and Equation 3.5 (omitting document terms):

$$\begin{aligned} i^*, j^* &= \arg\max_{i,j} \mathbf{d}' \cdot \mathbf{d}_{i:j} \\ &= \arg\max_{i,j} \mathbf{h}_1'^1 \cdot \mathbf{h}_i^1 + \mathbf{h}_1'^2 \cdot \mathbf{h}_j^2 + \\ &(\mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4)(\mathbf{h}_i^3 \cdot \mathbf{h}_j^4) \end{aligned}$$

From now on we let $l_i^1 = \mathbf{h}_1'^1 \cdot \mathbf{h}_i^1$ (phrase start logits), $l_j^2 = \mathbf{h}_1'^2 \cdot \mathbf{h}_j^2$ (phrase end logits), and $l_{i,j} = l_i^1 + l_j^2 + (\mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4)(\mathbf{h}_i^3 \cdot \mathbf{h}_j'^4)$ i.e. the value that is being maximized in the above equation.

One straightforward way to define the loss is to define it as the negative log probability of the correct answer where $\Pr(i, j) \propto \exp(l_{i,j})$. In other words,

$$L = -l_{i^*,j^*} + \log \sum_{i,j} \exp(l_{i,j}) \tag{3.6}$$

where L is the loss to minimize. Note that explicitly enumerating all possible phrases (enumerating all (i, j) pairs) during training time would be memory-intensive. Instead, we can efficiently obtain the loss by:

$$\begin{aligned} \mathbf{l}^1 &= [l_1^1, \ldots, l_T^1] = \mathbf{q}^1 {\mathbf{H}^1}^\top \\ \mathbf{l}^2 &= \mathbf{q}^2 {\mathbf{H}^2}^\top \\ \mathbf{L} &= {\mathbf{l}^1}^\top + \mathbf{l}^2 + (\mathbf{h}_1'^3 \cdot \mathbf{h}_1'^4)\mathbf{H}^3 {\mathbf{H}^4}^\top \end{aligned}$$

where $\mathbf{H}^m = [\mathbf{h}_1^m, \ldots, \mathbf{h}_T^m]$ for $m = 1, 2, 3, 4$, $+$ is with broadcasting and (i, j)-th element of $\mathbf{L}$ is $l_{i,j}$. Note that L can be entirely computed from $\mathbf{L}$.

While the loss function is clearly unbiased with respect to $\Pr(i, j) \propto \exp(l_{i,j})$, the summation in Equation 3.6 is computed over T^2 terms which is quite large and causes small gradient. To aid training, we define an auxilary loss L^1 corresponding to the start logits,

$$L^1 = -l_{i^*}^1 + \log \sum_i \exp(\frac{1}{T} \sum_j l_{i,j}) \tag{3.7}$$

and L^2 for the end logits in a similar way. By early summation (taking the mean), we reduce the number of exponential terms and allow larger gradients. We average between the true and aux loss for the final loss: $\frac{L}{2} + \frac{L^1+L^2}{4}$.

No-Answer Bias During training SQuAD (v1.1), we never observe negative examples (i.e. an unanswerable question in the paragraph). Following Levy et al. [2017], we introduce a trainable no-answer bias when computing softmax. For each paragraph, we create two negative examples by bringing one question from another article and one question from the same article but different paragraphs. Instead of randomly sampling, we bring the question with the highest inner product (i.e. most similar) with a randomly-picked positive question in the current paragraph, using a question embedding model trained on SQuAD v1.1. We jointly train the positive examples with the negative examples.

3.5.2 Indexing

Wikipedia consists of approximately 3 billion tokens, so enumerating all phrases with length ≤ 20 will result in about 60 billion phrases. With 961D of `float32` per phrase, one needs 240 TB of storage (60 billion times 961 dimensions times 4 bytes per dimension). While not impossible in industry scale, the size is clearly out of reach for independent or academic researchers and critically unfriendly for open research. We discuss three techniques we employee to reduce the size of the index to 1.2 TB without sacrificing much accuracy, which becomes much more manageable for everyone. In practice, additional 300-500GB will be needed to store auxiliary information for efficient indexing, which still sums up to less than 2TB.

1. Pointer Since each phrase vector is the concatenation of $\mathbf{a}_i$ and $\mathbf{b}_j$ (and a scalar $c_{i,j}$ but it takes very little space), many phrases share the same start or end vectors. Hence we store a single list of the start and the end vectors independently and just store pointers to those vectors for the phrase representation. This effectively reduces the memory footprint from 240 TB to 12 TB.

2. Filtering We train a simple single-layer binary classifier on top of each of the start and end vectors, supervised with the actual answer (without observing the question). This allows us to not store vectors that are unlikely to be a potential start or end position of the answer phrase, further reducing the memory

footprint from 12 TB to 5 TB.

3. Quantization We reduce the size of each vector by scalar quantization (SQ). That is, we convert each `float32` value to `int8` with appropriate offset and scaling. This allows us to reduce the size by one-fourth. Hence the final memory consumption is 1.2 TB. In future, more advanced methods such as Product Quantization (PQ) [Jegou et al., 2011] can be considered, though we note that in our experiment setup we could not find a good configuration that does not drop the accuracy significantly.

3.5.3 Search

While it would be ideal to (and possible to) directly approximate argmax in Equation 3.2 by using sparse maximum inner product search algorithm (some discussed in Section 3.2), we could not find a good open-source implementation that can scale up to billions of vectors and handle the dense and the sparse part of the phrase vector at the same time. We instead consider three approximation strategies.

First, *sparse-first search* (SFS) approximates the argmax by retrieving top-k^{s} documents with the sparse similarity search and then performing exact search (including sparse inner product scores) over all the phrases in retrieved documents. This is analogous to most pipeline-based QA systems, although our model can still yield much higher speed because it only needs to perform inner product once the documents are retrieved. Since the number of sparse document vectors is relatively small (5 million), we directly perform exact search using `scipy`, which has under 0.2s latency per query.

Second, *dense-first search* (DFS) approximates the argmax by doing search on the dense part first to retrieve top-k^{d} vectors and then reranking them by accessing the corresponding sparse vectors. Note that this implies a widely different behavior from SFS, as described in Section 3.6.2. We use `faiss` [Johnson et al., 2017], open-sourced and large-scale-friendly similarity search package for dense vectors.

Lastly, we consider a *hybrid* approach by independently performing both search strategies and reranking the appended list of the results.

Also, instead of directly searching on the dense vector $\mathbf{d}_{i:j}$ (concatenation of start, end, and coherency), we first search on the start vector $\mathbf{a}_i$ and obtain the best end position for each retrieved start position by computing the rest. We found that this allows us to save memory and time without sacrificing much accuracy, since the start vectors alone seem to contain sufficiently rich syntactic and semantic information already that

	Model	EM	F1	W/s
Original	DrQA	69.5	78.8	4.8K
	BERT-Large	84.1	90.9	51
Query-Agnostic	LSTM+SA	49.0	59.8	-
	LSTM+SA+ELMo	52.7	62.7	-
	DENSPI (dense only)	73.6	81.7	28.7M
	+ Linear layer	66.9	76.4	-
	+ Indep. encoders	65.4	75.1	-
	− Coherency scalar	71.5	81.5	-

Table 3.1: Results on SQuAD v1.1. 'W/s' indicates number of words the model can process (read) per second on a CPU in a batch mode (multiple queries at a time). DrQA [Chen et al., 2017] and BERT [Devlin et al., 2019] are from SQuAD leaderboard, and LSTM+SA and LSTM+SA+ELMo are query-agnostic baselines from Seo et al. [2018].

makes the search possible even in a large scale.

3.6 Experiments

Experiment section is divided into two parts. First, we report results on SQuAD [Rajpurkar et al., 2016]. This can be considered as a small-scale prerequisite to the open-domain experiment. It also allows a convenient comparison to state-of-the-art models in SQuAD, especially on the speed of the model. Under a fully controlled environment and batch-query scenario, our model processes words nearly 6,000 times faster than DrQA [Chen et al., 2017]. Second, we report results on open-domain SQuAD (called SQuAD-Open), following the same setup as in DrQA. We show that our model achieves up to 6.4% better accuracy and up to 58 times faster end-to-end inference time than DrQA while exploring nearly 200 times more unique documents. All experiments are CPU-only benchmark.

3.6.1 SQuAD v1.1 Experiments

In the SQuAD v1.1 setup, our model effectively uses only the dense vector since every sparse (document tf-idf) vector will be identical in the same paragraph. While this is a much easier problem than open-domain, it can serve as a reliable and fast indicator of how well the model would do in the open-domain setup.

Model details We use BERT-large ($d = 1024$) for the text encoders, which is pretrained on a large text corpus (Wikipedia dump and Book Corpus). We refer readers to the original paper by Devlin et al. [2019] for details; we mostly use the default settings described there. We use $d^{\text{b}} = 480$, resulting in phrase size of $2d^{\text{b}} + 1 = 961$, and $d^{\text{c}} = 32$. We train with a batch size of 12 (on four P40 GPUs) for 3 epochs.

Baselines We compare the performance of our system DENSPI with a few baselines in terms of accuracy and efficiency. The first group are among the models that are submitted to SQuAD v1.1 Leaderboard, specifically DrQA [Chen et al., 2017] and BERT [Devlin et al., 2019] (current state of the art). These models encode the evidence document given the question, but they suffer from the disadvantage that the evidence document needs to be re-encoded for every new question at the inference time, and they are strictly linear time in that they cannot utilize approximate search algorithms. The second group of baselines are introduced by Seo et al. [2018], specifically LSTM+SA and LSTM+SA+ELMo that also encode phrases independent of the question using LSTM, Self-Attention, and ELMo [Peters et al., 2018] encodings.

Results Table 3.1 compares the performance of our system with different baselines in terms of efficiency and accuracy. We note the following observations from the result table. (1) DENSPI outperforms the query-agnostic baseline [Seo et al., 2018] by a large margin, 20.1% EM and 18.5% F1. This is largely credited towards the usage of BERT encoder with an effective phrase embedding mechanism on the top. (2) DENSPI outperforms DrQA by 3.3% EM. This signifies that phrase-indexed models can now outperform early (unconstrained) state-of-the-art models in SQuAD. (3) DENSPI is 9.2% below the current state of the art. The difference, which we call *decomposability gap*[4], is now within 10% and future work will involve further closing the gap. (4) Query-agnostic models can process (read) words much faster than query-dependent representation models. In a controlled environment where all information is in memory and the documents are pre-indexed, DENSPI can process 28.7 million words per second, which is 6,000 times faster than DrQA and 563,000 times faster than BERT without any approximation.

Ablations Ablations are also shown at the bottom of Table 3.1. The first ablation adds a linear layer on top of the BERT encoder for the phrase embeddings, which is more analogous to how BERT handles other

[4]The gap is due to constraining the scoring function to be decomposable into question encoder and context encoder.

	F1	EM	s/Q	#D/Q
DrQA	-	29.8	35	5
R^3	37.5	-	-	-
Paragraph ranker	-	30.2	-	20
Multi-step reasoner	39.2	31.9	-	-
MINIMAL	42.5	34.7	-	10
BERTserini	46.1	38.6	115	-
Weaver	-	42.3	-	25
DENSPI-SFS	42.5	33.3	0.60	5
DENSPI-DFS	35.9	28.5	0.51	815
–sparse scale=0	16.3	11.2	0.40	815
DENSPI-Hybrid	44.4	36.2	0.81	817

Table 3.2: Results on SQuAD-Open. Top rows are previous models that re-encode documents for every question. The bottom rows are our proposed model. 's/Q' is seconds per query on a CPU and '#D/Q' is the number of documents visited per query.

language tasks. We see a huge drop in performance. We also try independent BERT encoders (i.e. unshared parameters) between phrase and question embedding models, and we also see a large drop as well. These seem to indicate that a careful design consideration for even small details are crucial when finetuning BERT. Our ablation that excludes coherency scalar decreases DENSPI's EM score by 2% and F1 by 0.2%. This agrees with our intuition that the coherency scalar is useful for precisely defining valid phrase constituents.

3.6.2 Open-domain Experiments

In this subsection, we evaluate our model's performance (accuracy and speed) on Open-domain SQuAD (SQuAD-Open), which is an extension of SQuAD [Rajpurkar et al., 2016] by Chen et al. [2017]. In this setup, the evidence is the entire English Wikipedia, and the golden paragraphs are not provided for questions.

Model details For the dense vector, we adopt the same setup from Section 3.6.1 except that we train with no-answer questions (Section 3.5.1) and an increased batch size of 18. For the sparse vector of each phrase, we use the identical 2-gram tf-idf vector used by Chen et al. [2017], whose vocabulary size is approximately 17 million, of the document that contains the phrase. Since the sparse vector and the dense vector are independently obtained, we tune the linear scale between the sparse and the dense vectors and found that 0.05 (multiplied on the sparse vector) gives the best performance. As discussed in Section 3.5.3,

we consider three search strategies. For sparse-first search (SFS), we retrieve top-5 documents. For dense-first search (DFS), we use an HNSW-based [Malkov and Yashunin, 2018] coarse quantizer with 2^{20} (1M) clusters (obtained with k-means) and nprobe=64 (number of clusters to visit). We retrieve top 1000 dense (start) vectors. The 'Hybrid' setup adopts the same configurations from both.

Baselines We compare our system with previous state-of-the-art models for open-domain question answering. The baselines include DrQA [Chen et al., 2017], MINIMAL [Min et al., 2018], multi-step-reasoner [Das et al., 2019], Paragraph Ranker [Lee et al., 2018], R^3 [Wang et al., 2018a], BERTserini [Yang et al., 2019], and Weaver [Raison et al., 2018]. We do not experiment with Seo et al. [2018] due to its poor performance with respect to DENSPI as demonstrated in Table 3.1.

Results Table 3.2 shows the results of our system and previous models on SQuAD-Open. We note following observations: (1) DENSPI-Hybrid outperforms DrQA by 6.4% while achieving 43 times faster inference speed. (2) DENSPI-Hybrid is 6.1% EM behind Weaver, which co-encodes top 25 documents (retrieved by tf-idf) for every new question. As mentioned in Section 3.6.1, the difference between ours and Weaver can be considered as the *decomposability gap* arising from the constraint of query-agnostic phrase representations. We note, however, that the gap is smaller now in open-domain, and the speed-up is expected to be much larger[5] since Weaver has higher computational complexity than DrQA and reads top 25 documents. (3) We also report the number of documents that our model computes exact search on and compare it to that of DrQA, as indicated by '#D/Q' in the table. Top-1000 dense search in DENSPI-Hybrid results in 817 unique documents on average, which is much more diverse than the 5 documents that DrQA considers. The benefit of this diversity is better illustrated in the upcoming qualitative analysis (Table 3.3).

Ablations Table 3.2 (bottom) shows the effect of different search strategies (SFS vs DFS vs Hybrid) and the importance of the sparse vector.

SFS vs DFS vs Hybrid: We first see that DENSPI-SFS and DENSPI-DFS have comparable inference speed while DENSPI-SFS has 6.6% higher F1. While this demonstrates the effectiveness of sparse search, it is important to note that this might be due to the high word overlap between the question and the context

[5]Weaver is not open-sourced so we could not benchmark it.

Q: What can hurt a teacher's mental and physical health? A: occupational stress	
DrQA	[*Mental health*] ... and poor mental health can lead to problems such as **substance abuse**.
DENSPI	[*Teacher*] Teachers face several occupational hazards in their line of work, including **occupational stress**, ...
Q: Who was Kennedy's science adviser that opposed manned spacecraft flights? A: Jerome Wiesner	
DrQA	[*Apollo program*] Kennedy's science advisor **Jerome Wiesner**, (...) his opposition to manned spaceflight ... [*Apollo program*] ... and the sun by NASA manager **Abe Silverstein**, who later said that ... [*Apollo program*] Although Grumman wanted a second unmanned test, **George Low** decided (...) be manned.
DENSPI	[*Apollo program*] Kennedy's science advisor **Jerome Wiesner**, ... his opposition to manned spaceflight ... [*Space Race*] **Jerome Wiesner** of MIT, who served as a (...) advisor to (...) Kennedy, (...) opponent of manned ... [*John F. Kennedy*] ... science advisor **Jerome Wiesner** (...) strongly opposed to manned space exploration, ...
Q: What to do when you're bored?	
DrQA	[*Bored to Death (song)*] I'm nearly bored to **death** [*Waterview Connection*] The twin tunnels were bored by (...) tunnel **boring** machine (TBM) ... [*Bored to Death (song)*] It's easier to say you're bored, or to be angry, than it is to be **sad**.
DENSPI	[*Big Brother 2*] When bored, she enjoys **drawing**. [*Angry Kid*] Angry Kid is (...) bored of long car journeys, so Dad suggests he just **close his eyes and sleep**. [*Pearls Before Swine*] In law school, he became so bored during classes, he started to **doodle a rat**, ...

Table 3.3: Prediction samples from DrQA and DENSPI in open-domain (English Wikipedia). Each sample shows [*document title*], context, and **predicted answer**.

Q: What was the main radio network in the 1940s in America? A: NBC Red Network	
DENSPI	[*American Broadcasting Company*] In the 1930s, radio in the United States was dominated by (...): the **Columbia Broadcasting System**, the Mutual Broadcasting (...).
Q: Which city is the fifth-largest city in California? A: Fresno	
DENSPI	[*Oakland, California*] **Oakland** is the largest city and the county seat of (...), California, United States.

Table 3.4: Wrong prediction samples from DENSPI in open-domain (English Wikipedia). Each sample shows [*document title*], context, and **predicted answer**.

in SQuAD. Furthermore, we see that Hybrid achieves the highest accuracy in both F1 and EM, implying that the two strategies are complimentary.

Sparse vector: DENSPI-DFS with 'sparse scale=0' implies that we entirely remove the sparse vector, i.e. $\mathbf{x}_{i:j} = \mathbf{d}_{i:j}$ in Equation 3.1. While this wouldn't have any effect in SQuAD v.1.1, we see a significant drop (-19.6% F1), indicating the importance of the sparse vector in open-domain for distinguishing semantically close but lexically distinct entities.

Qualitative Analysis Table 3.3 contrasts between the results from DrQA and DENSPI-Hybrid. In the top example, we note that DrQA fails to retrieve the right document, whereas DENSPI finds the correct answer. This happens exactly because the document retrieval model would not precisely know what kind of content is in the document, while dense search allows it to consider the content directly through phrase-level retrieval. In the second example, while both obtain the correct top-1, DENSPI also obtains the same answer from three different documents. The last example (not from SQuAD) does not have a noun entity, in which a term-frequency-based search engine often performs poorly. We indeed see that DrQA fails because wrong documents are retrieved. On the other hand, DENSPI is able to obtain good answers from several different documents. These results also reinforce the importance of exploring diverse documents ('#D/Q' in Table 3.2).

Error Analysis Table 3.4 shows wrong predictions from DENSPI. In the first example, the model seems to fail to distinguish '1940s' from '1930s'. In the second example, the model seems to focus more on the

word ‘largest’ than the word ‘fifth-’ in the question.

3.7 Conclusion

We introduce a model for real-time open-domain question answering by learning indexable phrase representations independent of the query, which leverage both dense and sparse vectors to capture lexical, semantic, and syntactic information. On SQuAD-Open, our experiments show that our model can read words 6,000 times faster under a controlled environment and 43 times faster in a real setup than DrQA while achieving 6.4% higher EM. We believe that even further speedup and larger coverage of documents can be done with a dedicated similarity search package for dense+sparse vectors. We note that, however, the gap due to query-agnostic constraint still exists and is at least 6.1% EM. Hence, more effort on designing a better phrase representation model is needed to close the gap. In Chapter 4, I discuss a follow-up work for this effort.

Chapter 4

Sparse Knowledge Memory

4.1 Introduction

Open-domain question answering (QA) is the task of answering generic factoid questions by looking up a large knowledge source, typically unstructured text corpora such as Wikipedia, and finding the answer text segment [Chen et al., 2017]. One widely adopted strategy to handle such large corpus is to use an efficient document (or paragraph) retrieval technique to obtain a few relevant documents, and then use an accurate (yet expensive) machine reading model to *read* the retrieved documents and find the answer [Chen et al., 2017; Wang et al., 2018a; Das et al., 2019; Yang et al., 2019], as also discussed in Chapter 2.

In Chapter 3, I discussed an alternative approach that formulates the task as an end-to-end phrase retrieval problem by encoding and indexing every possible text span in a dense vector offline [Seo et al., 2018]. The approach promises a massive speed advantage with several orders of magnitude lower time complexity, but it performs poorly on entity-centric questions, often unable to disambiguate similar but different entities such as "*1991*" and "*2001*" in dense vector space. To alleviate this issue, Seo et al. [2019] concatenate a term-frequency-based sparse vector with the dense vector to capture lexical information. However, such sparse vector is identical across the document (or paragraph), which means every word's importance is equally considered regardless of its context (Figure 4.1).

In this paper, we introduce a method to learn a Contextualized Sparse Representation (SPARC) for each phrase and show its effectiveness in open-domain QA under phrase retrieval setup. Related previous work

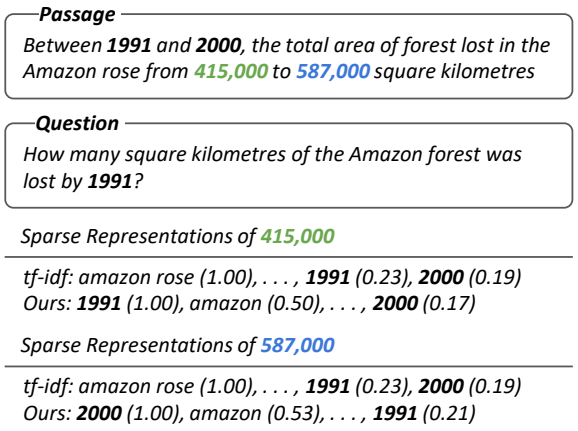


Figure 4.1: An example of sparse vectors given a context from SQuAD. While *tf-idf* has high weights on infrequent n-grams, our contextualized sparse representation (SPARC) focuses on sematically related n-grams.

(for a different task) often directly maps dense vectors to a sparse vector space [Faruqui et al., 2015; Subramanian et al., 2018], which can be at most only a few thousand dimensions due to computational cost and small gradients. We instead leverage rectified self-attention weights on the neighboring n-grams to scale up its cardinality to n-gram vocabulary space (billions), allowing us to encode rich lexical information in each sparse vector. We *kernelize*[1] the inner product space during training to avoid explicit mapping and obtain memory- and computational efficiency.

SPARC improves the previous phrase retrieval model, DenSPI [Seo et al., 2019] (by augmenting its phrase embedding), by more than 4% in both CuratedTREC and SQuAD-Open. In fact, our CuratedTREC result achieves the new state of the art even when compared to previous retrieve & read approaches, with at least 45x faster speed.

4.2 Background

We focus on open-domain QA on unstructured text where the answer is a text span in a textual corpus (e.g., Wikipedia). Formally, given a set of K documents $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^K$ and a question $\boldsymbol{q}$, the task is to design a model that obtains the answer $\hat{\boldsymbol{a}}$ by $\hat{\boldsymbol{a}} = \arg\max_{\boldsymbol{x}^k_{i:j}} F(\boldsymbol{x}^k_{i:j}, \boldsymbol{q})$, where F is the score model to learn and $\boldsymbol{x}^k_{i:j}$ is a phrase consisting of words from the i-th to the j-th word in the k-th document. Pipeline-based

[1] Our method is inspired by the kernel method in SVMs [Cortes and Vapnik, 1995].

methods [Chen et al., 2017; Lin et al., 2018; Wang et al., 2019] typically leverage a document retriever to reduce the number of documents to read, but they suffer from error propagation when wrong documents are retrieved and can be still slow due to the heavy reader model.

Phrase-Indexed Open-domain QA As an alternative, Seo et al. [2018, 2019] introduce an end-to-end, real-time open-domain QA approach to directly encode all phrases in documents agnostic of the question, and then perform similarity search on the encoded phrases. This is feasible by decomposing the scoring function F into two functions,

$$\hat{a} = \arg\max_{\boldsymbol{x}^k_{i:j}} H_{\boldsymbol{x}}(\boldsymbol{x}^k_{i:j}) \cdot H_{\boldsymbol{q}}(\boldsymbol{q})$$

where $H_{\boldsymbol{x}}$ is the query-agnostic phrase encoding, and $H_{\boldsymbol{q}}$ is the question encoding, and $\cdot$ denotes a fast inner product operation.

Seo et al. [2019] propose to encode each phrase (and question) with the concatenation of a dense vector obtained via a deep contextualized word representation model [Devlin et al., 2019] and a sparse vector obtained via computing the tf-idf of the document (paragraph) that the phrase belongs to. We argue that the inherent characteristics of tf-idf, which is not learned and identical across the same document, has limited representational power. Our goal in this paper is to propose a better and learned sparse representation model that can further improve the QA accuracy in the phrase retrieval setup.

4.3 Sparse Encoding of Phrases

Our sparse model, unlike pre-computed sparse embeddings such as tf-idf, *dynamically* computes the weight of each n-gram that depends on the context.

4.3.1 Why do we need sparse representations?

To answer the question in Figure 4.1, the model should know that the target answer (***415,000***) corresponds to the year *1991* while the (confusing) phrase *587,000* corresponds to the year *2000*. The dense phrase encoding is likely to have difficulty in precisely differentiating between *1991* and *2000* since it needs to also

encode several different kinds of information. Window-based tf-idf would not help because the year *2000* is closer (in word distance) to ***415,000***. This example illustrates the strong need to create an n-gram-based sparse encoding that is highly syntax- and context-aware.

4.3.2 Contextualized Sparse Representations

The sparse representation of each phrase is obtained as the concatenation of its start word's and end word's sparse embedding, i.e. $\mathbf{s}_{i:j} = [\mathbf{s}_i^{\text{start}}, \mathbf{s}_j^{\text{end}}]$. This way, similarly to how the dense phrase embedding is obtained in Seo et al. [2019], we can efficiently compute them without explicitly enumerating all possible phrases.

We obtain each (start/end) sparse embedding in the same way (with unshared parameters), so we just describe how we obtain the start sparse embedding here and omit the superscript 'start'. Given the contextualized encoding of each document $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N] \in \mathbb{R}^{N \times d}$, we obtain its (start/end) sparse encoding $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_N] \in \mathbb{R}^{N \times F}$ by

$$\mathbf{S} = \text{ReLU}\big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\big)\mathbf{F} \in \mathbb{R}^{N \times F} \tag{4.1}$$

where $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{N \times d}$ are query, key matrices obtained by applying a (different) linear transformation on $\mathbf{H}$ (i.e., using $W_{\mathbf{Q}}, W_{\mathbf{K}} : \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^{N \times d}$), and $\mathbf{F} \in \mathbb{R}^{N \times F}$ is an one-hot n-gram feature representation of the input document $\boldsymbol{x}$. That is, for instance, if we want to encode unigram (1-gram) features, $\mathbf{F}_i$ will be a one-hot representation of the word x_i, and F will be equivalent to the vocabulary size. Intuitively, $\mathbf{s}_i$ contains a weighted bag-of-ngram representation where each n-gram is weighted by its relative importance on each start or end word of a phrase. Note that F will be very large, so it should always exist as an efficient sparse matrix format (e.g., csc), and one should not explicitly create its dense form. Since we want to handle several different sizes of n-grams, we create the sparse encoding $\mathbf{S}$ for each n-gram and concatenate the resulting sparse encodings. In practice, we experimentally find that unigram and bigram are sufficient for most use cases.

We compute sparse encodings on the question side ($\mathbf{s}' \in \mathbb{R}^F$) in a similar way to the document side, with the only difference that we use the `[CLS]` token instead of start and end words to represent the entire question. We share the same BERT and linear transformation weights used for the phrase encoding.

4.3.3 Training

As training phrase encoders on the whole Wikipedia is computationally prohibitive, we use training examples from an extractive question answering dataset (SQuAD) to train our encoders. We also use an improved negative sampling method which makes both dense and sparse representations more robust to noisy texts.

Kernel Function Given a pair of question $\boldsymbol{q}$ and a golden document $\boldsymbol{x}$ (a paragraph in the case of SQuAD), we first compute the dense logit of each phrase $\boldsymbol{x}_{i:j}$ by $l_{i,j} = \mathbf{h}_{i:j} \cdot \mathbf{h}'$. Each phrase's sparse embedding is trained, so it needs to be considered in the loss function. We define the sparse logit for phrase $\boldsymbol{x}_{i:j}$ as $l^{\text{sparse}}_{i,j} = \mathbf{s}_{i:j} \cdot \mathbf{s}'_{\texttt{[CLS]}} = \mathbf{s}^{\text{start}}_i \cdot \mathbf{s}'^{\text{start}}_{\texttt{[CLS]}} + \mathbf{s}^{\text{end}}_j \cdot \mathbf{s}'^{\text{end}}_{\texttt{[CLS]}}$. For brevity, we describe how we compute the first term $\mathbf{s}^{\text{start}}_i \cdot \mathbf{s}'^{\text{start}}_{\texttt{[CLS]}}$ corresponding to the start word (and dropping the superscript 'start'); the second term can be computed in the same way.

$$\mathbf{s}^{\text{start}}_i \cdot \mathbf{s}'^{\text{start}}_{\texttt{[CLS]}} = \text{ReLU}\Big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\Big)_i \mathbf{F} \quad \left(\text{ReLU}\Big(\frac{\mathbf{Q}'\mathbf{K}'^\top}{\sqrt{d}}\Big)_{\texttt{[CLS]}} \mathbf{F}'\right)^\top \tag{4.2}$$

where $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{M \times d}, \mathbf{F}' \in \mathbb{R}^{M \times F}$ denote the question side query, key, and n-gram feature matrices, respectively. The output size of $\mathbf{F}$ is prohibitively large, but we efficiently compute the loss by precomputing $\mathbf{F}\mathbf{F}'^\top \in \mathbb{R}^{N \times M}$. Note that $\mathbf{F}\mathbf{F}'^\top$ can be considered as applying a kernel function, i.e. $K(\mathbf{F}, \mathbf{F}') = \mathbf{F}\mathbf{F}'^\top$ where its (i, j)-th entry is 1 if and only if the n-gram at the i-th position of the context is equivalent to the j-th n-gram of the question, which can be efficiently computed as well. One can also think of this as *kernel trick* (in the literature of SVM [Cortes and Vapnik, 1995]) that allows us to compute the loss without explicit mapping.

The final loss to minimize is computed from the negative log likelihood over the sum of the dense and sparse logits:

$$L = -(l_{i^*,j^*} + l^{\text{sparse}}_{i^*,j^*}) + \log \sum_{i,j} \exp(l_{i,j} + l^{\text{sparse}}_{i,j})$$

where i^*, j^* denote the true start and end positions of the answer phrase. As we don't want to sacrifice the quality of dense representations which is also very critical in dense-first search explained in Section 4.4.1,

we add dense-only loss that omits the sparse logits (i.e. original loss in Seo et al. [2019]) to the final loss, in which case we find that we obtain higher-quality dense phrase representations.

Negative Sampling To learn robust phrase representations, we concatenate negative paragraphs to the original SQuAD paragraphs. To each paragraph $\boldsymbol{x}$, we concatenate the paragraph $\boldsymbol{x}_{\text{neg}}$ which was paired with the question whose dense representation $\mathbf{h}'_{\text{neg}}$ is most similar to the original dense question representation $\mathbf{h}'$, following Seo et al. [2019]. We find that adding tf-idf matching scores on the word-level logits of the negative paragraphs further improves the quality of sparse representations.

4.4 Experiments

4.4.1 Experimental Setup

Datasets SQuAD-Open is the open-domain version of SQuAD [Rajpurkar et al., 2016]. We use 87,599 examples with the golden evidence paragraph to train our encoders and use 10,570 examples from dev set to test our model, as suggested by Chen et al. [2017]. CURATEDTREC consists of question-answer pairs from TREC QA [Voorhees et al., 1999] curated by Baudiš and Šedivỳ [2015]. We use 694 test set QA pairs for testing our model. We only train on SQuAD and test on both SQuAD-Open and CuratedTREC, relying on the generalization ability of our model (zero-shot) for CuratedTREC.

Implementation Details We use and finetune BERT-Large for our encoders. We use BERT vocabulary which has 30,522 unique tokens based on byte pair encodings. As a result, we have $F \approx 1B$ when using both uni-/bigram features. We do not finetune the word embedding during training. We pre-compute and store all encoded phrase representations of all documents in Wikipedia (more than 5 million documents). It takes 600 GPU hours to index all phrases in Wikipedia. We use the same storage reduction and search techniques by Seo et al. [2019]. For search, we perform dense search first and then rerank with sparse scores (DFS) or perform sparse search first and rerank with dense scores (SFS), or a combination of both (Hybrid).

Comparisons For models using dedicated search engines, we show performances of DrQA [Chen et al., 2017], R^3 [Wang et al., 2018a], Paragraph Ranker [Lee et al., 2018], Multi-Step-Reasoner [Das et al., 2019],

Model	C.TREC	SQuAD-Open		
	EM	EM	F1	s/Q
Models with Dedicated Search Engines				
DrQA	25.4*	29.8**	-	35
R^3	28.4*	29.1	37.5	-
Paragraph Ranker	**35.4***	30.2	-	161
Multi-Step-Reasoner	-	31.9	39.2	-
BERTserini	-	38.6	46.1	115
Multi-passage BERT	-	**53.0**	**60.9**	84
End-to-End Models				
ORQA	30.1	20.2	-	8.0
DENSPI	31.6†	36.2	44.4	0.71
DENSPI + SPARC (Ours)	**35.7**†	**40.7**	**49.0**	0.78

* Trained on distantly supervised training data.
** Trained on multiple datasets
† No supervision using target training data.

Table 4.1: Results on two open-domain QA datasets.

BERTserini [Yang et al., 2019], and Multi-passage BERT [Wang et al., 2019]. For end-to-end models that do not rely on search engine results, DENSPI [Seo et al., 2019], ORQA [Lee et al., 2019], and DENSPI + SPARC (Ours) are evaluated. For DENSPI and ours, 'Hybrid' search strategy is used.

4.4.2 Results

Open-Domain QA Experiments Table 4.1 shows experimental results on two open-domain question answering datasets, comparing our method with previous pipeline and end-to-end approaches. On both datasets, our model with contextualized sparse representations (DENSPI + SPARC) largely improves the performance of the phrase-indexing baseline model (DENSPI) by more than 4%. Also, our method runs significantly faster than other models that need to run heavy QA models during the inference. On CuratedTREC, which is constructed from real user queries, our model achieves state-of-the-art performance at the time of submission. Even though our model is only trained on SQuAD (i.e., zero-shot), it outperforms all other models which are either distant- or semi-supervised with at least 45x faster inference.

On SQuAD-Open, our model outperforms BERT-based pipeline approaches such as BERTserini [Yang et al., 2019] while being more than two orders of magnitude faster. Multi-passage BERT, which utilizes

Model	$\text{SQuAD}_{1/100}$	$\text{SQuAD}_{1/10}$
Ours	60.0	51.6
− SPARC	55.9 (−4.1)	48.4 (−3.2)
− Doc./Para. tf-idf	58.1 (−1.9)	50.9 (−0.7)
+ Trigram SPARC	58.0 (−2.0)	49.8 (−1.8)

Table 4.2: Ablations of our model. We show effects of different sparse representations.

	Model	EM	F1
Original	DrQA [Chen et al., 2017]	69.5	78.8
	BERT [Devlin et al., 2019]	84.1	90.9
Query-Agnostic	LSTM + SA + ELMo	52.7	62.7
	DENSPI	73.6	81.7
	DENSPI + SPARC	76.4	84.8

Table 4.3: Results on the SQuAD development set. LSTM+SA+ELMo is a query-agnostic baseline from Seo et al. [2018].

a dedicated document retriever, outperforms all end-to-end models with a large margin in SQuAD-Open. While our main contribution is on the improvement in end-to-end, we also note that retrieving correct documents in SQuAD-Open is known to be often easily exploitable [Lee et al., 2019], so we should use more open-domain-appropriate test datasets (such as CuratedTREC) for a more fair comparison.

Ablation Study Table 4.2 shows the effect of contextualized sparse representations by comparing different variants of our method on SQuAD-Open. We use a subset of Wikipedia dump (1/100 and 1/10). Interestingly, adding trigram features in SPARC is worse than using uni-/bigram representations only, calling for a stronger regularization for high-order n-gram features.

Closing the Decomposability Gap Table 3.1 shows the performance of DenSPI + SPARC in the SQuAD v1.1 development set, where a single paragraph that contains an answer is provided in each sample. While BERT-Large that jointly encodes a passage and a question still has a higher performance than ours, we have closed the gap to 6.1 F1 score in a query-agnostic setting.

Qualitative Analysis Table 4.4 shows the outputs of three OpenQA models: DrQA [Chen et al., 2017], DENSPI [Seo et al., 2019], and DENSPI + SPARC (ours). Our model is able to retrieve various correct

Q: When is Independence Day?	
DrQA	[*Independence Day (1996 film)*] Independence Day is a **1996** American science fiction ...
DENSPI + SPARC	[*Independence Day (India)*] ... is annually observed on **15 August** as a national holiday in India.
Q: What was the GDP of South Korea in 1950?	
DRQA	[*Economy of South Korea*] In 1980, the South Korean GDP per capita was **$2,300**.
DENSPI	[*Economy of South Korea*] In 1980, the South Korean GDP per capita was **$2,300**.
DENSPI + SPARC	[*Developmental State*] South Korea's GDP grew from **$876** in 1950 to $22,151 in 2010.

Table 4.4: Prediction samples from DrQA, DENSPI, and DENSPI + SPARC. Each sample shows [*document title*], context, and **predicted answer**.

answers from different documents, and it often correctly answers questions with specific dates or numbers compared to DENSPI showing the effectiveness of learned sparse representations.

4.5 Conclusion

In this paper, we demonstrate the effectiveness of contextualized sparse representations, DENSPI, for encoding phrase with rich lexical information in open-domain question answering. We efficiently train our sparse representations by kernelizing the sparse inner product space. Experimental results show that our fast open-domain QA model that augments DENSPI with DENSPI outperforms previous open-domain QA models, including recent BERT-based pipeline models, with two orders of magnitude faster inference time.

Chapter 5

Neural Reasoning

5.1 Introduction

In this chapter, I address the problem of reasoning over multiple pieces of factual knowledge. For example, consider we know that `Frogs eat insects` and `Flies are insects`. Then answering `Do frogs eat flies?` requires reasoning over both of the above facts. Question answering, more specifically context-based QA, has been extensively studied in machine comprehension tasks [Richardson et al., 2013b; Hermann et al., 2015; Hill et al., 2016; Rajpurkar et al., 2016]. However, most of the datasets are primarily focused on lexical and syntactic understanding, and hardly concentrate on inference over multiple facts. Recently, several datasets aimed for testing multi-hop reasoning have emerged; among them are story-based QA [Weston et al., 2016] and the dialog task [Bordes and Weston, 2017].

Recurrent Neural Network (RNN) and its variants, such as Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Unit (GRU) [Cho et al., 2014], are popular choices for modeling natural language. However, when used for multi-hop reasoning in question answering, purely RNN-based models have shown to perform poorly [Weston et al., 2016]. This is largely due to the fact that RNN's internal memory is inherently unstable over a long term. For this reason, most recent approaches in the literature have mainly relied on global attention mechanism and shared external memory [Sukhbaatar et al., 2015; Peng et al., 2015; Xiong et al., 2016; Graves et al., 2016]. The attention mechanism allows these models to focus on a single sentence in each layer. They can sequentially read multiple relevant sentences

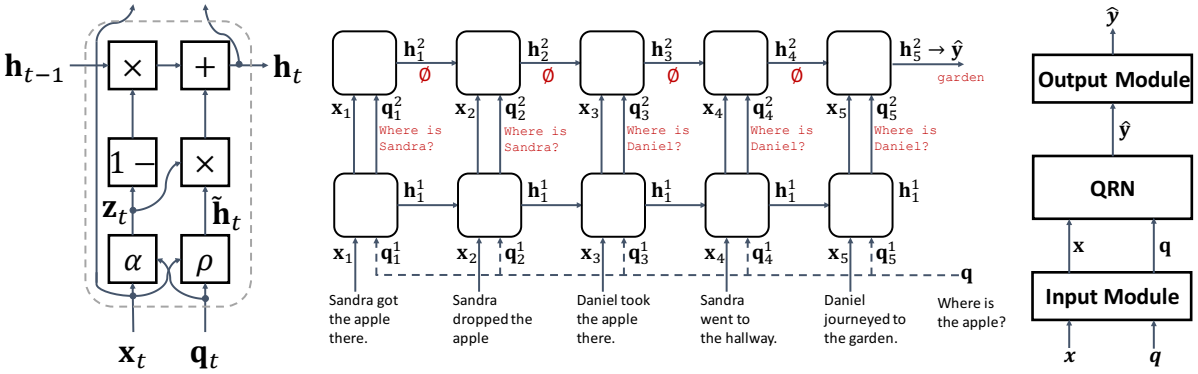
from the memory with multiple layers to perform multi-hop reasoning. However, one major drawback of these standard attention mechanisms is that they are insensitive to the time step (memory address) of the sentences when accessing them.

Our proposed model, Query-Reduction Network[1](QRN), is a single recurrent unit that addresses the long-term dependency problem of most RNN-based models by simplifying the recurrent update, while taking the advantage of RNN's capability to model sequential data (Figure 5.1). QRN considers the context sentences as a sequence of state-changing triggers, and transforms (*reduces*) the original query to a more informed query as it observes each trigger through time. For instance in Figure 5.1b, the original question, `Where is the apple?`, cannot be directly answered by any single sentence from the story. After observing the first sentence, `Sandra got the apple there`, QRN transforms the original question to a reduced query `Where is Sandra?`, which is presumably easier to answer than the original question given the context provided by the first sentence.[2] Unlike RNN-based models, QRN's candidate state ($\tilde{\mathbf{h}}_t$ in Figure 5.1a) does not depend on the previous hidden state ($\mathbf{h}_{t-1}$). Compared to memory-based approaches [Sukhbaatar et al., 2015; Peng et al., 2015; Kumar et al., 2016; Xiong et al., 2016], QRN can better encodes *locality* information because it does not use a global memory access controller (circle nodes in Figure 5.2), and the query updates are performed locally.

In short, the main contribution of QRN is threefold. First, QRN is a simple variant of RNN that *reduces* the query given the context sentences in a differentiable manner. Second, QRN is situated between the attention mechanism and RNN, effectively handling time dependency and long-term dependency problems of each technique, respectively. Hence it is well-suited for sequential data with both local and global interactions (note that QRN is *not* the replacement of RNN, which is arguably better for modeling complex local interactions). Third, unlike most RNN-based models, QRN can be parallelized over time by computing candidate reduced queries ($\tilde{\mathbf{h}}_t$) directly from local input queries ($\mathbf{q}_t$) and context sentence vectors ($\mathbf{x}_t$). In fact, the parallelizability of QRN implies that QRN does not suffer from the vanishing gradient problem of RNN, hence effectively addressing the long-term dependency. We experimentally demonstrate these contributions by achieving the state-of-the-art results on story-based QA and interactive dialog datasets.

[1]Code is publicly available at: `seominjoon.github.io/qrn/`

[2]This mechanism is akin to logic regression in situation calculus [Reiter, 2001].



(a) QRN unit **(b)** 2-layer QRN **(c)** Overview

Figure 5.1: (5.1a) QRN unit, (5.1b) 2-layer QRN on 5-sentence story, and (5.1c) entire QA system (QRN and input / output modules). $\boldsymbol{x}, \boldsymbol{q}, \hat{\boldsymbol{y}}$ are the story, question and predicted answer in natural language, respectively. $\mathbf{x} = \langle \mathbf{x}_1, \ldots, \mathbf{x}_T \rangle, \mathbf{q}, \hat{\mathbf{y}}$ are their corresponding vector representations (upright font). α and ρ are update gate and reduce functions, respectively. $\hat{\mathbf{y}}$ is assigned to be $\mathbf{h}_5^2$, the local query at the last time step in the last layer. Also, red-colored text is the inferred meanings of the vectors (see 'Interpretations' of Section 5.5.3).

5.2 Model

In story-based QA (or dialog dataset), the input is the *context* as a sequence of sentences (story or past conversations) and a *question* in natural language (equivalent to the user's last utterance in the dialog). The output is the predicted answer to the question in natural language (the system's next utterance in the dialog). The only supervision provided during training is the answer to the question.

In this paper we particularly focus on end-to-end solutions, i.e., the only supervision comes from questions and answers, and we restrain from using manually defined rules or external language resources, such as lexicon or dependency parser. Let $\langle \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \rangle$ denote the sequence of sentences, where T is the number of sentences in the story, and let $\boldsymbol{q}$ denote the question. Let $\hat{\boldsymbol{y}}$ denote the predicted answer, and $\boldsymbol{y}$ denote the true answer. Our proposed system for end-to-end QA task is divided into three modules (Figure 4.1): input module, QRN layers, and output module.

Input module. Input module maps each sentence $\boldsymbol{x}_t$ and the question $\boldsymbol{q}$ to d-dimensional vector space, $\mathbf{x}_t \in \mathbb{R}^d$ and $\mathbf{q}_t \in \mathbb{R}^d$. We adopt a previous solution for the input module (details in Section 5.5).

QRN layers. QRN layers use the sentence vectors and the question vector from the input module to obtain the predicted answer in vector space, $\hat{\mathbf{y}} \in \mathbb{R}^d$. A QRN layer refers to the recurrent application of a QRN

unit, which can be considered as a variant of RNN with two inputs, two outputs, and a hidden state (reduced query), all of which operate in vector space. The details of the QRN module is explained throughout this section (5.2.1, 5.2.2).

Output module. Output module maps $\hat{\mathbf{y}}$ obtained from QRN to a natural language answer $\hat{\boldsymbol{y}}$. Similar to the input module, we adopt a standard solution for the output module (details in Section 5.5).

We first formally define the base model of a QRN unit, and then we explain how we connect the input and output modules to it (Section 5.2.1). We also present a few extensions to the network that can improve QRN's performance (Section 5.2.2). Finally, we show that QRN can be parallelized over time, giving computational advantage over most RNN-based models by one order of magnitude (Section 5.3).

5.2.1 QRN Unit

As an RNN-based model, QRN is a single recurrent unit that updates its hidden state (reduced query) through time and layers. Figure 5.1a depicts the schematic structure of a QRN unit, and Figure 5.1b demonstrates how layers are stacked. A QRN unit accepts two inputs (*local* query vector $\mathbf{q}_t \in \mathbb{R}^d$ and sentence vector $\mathbf{x}_t \in \mathbb{R}^d$), and two outputs (reduced query vector $\mathbf{h}_t \in \mathbb{R}^d$, which is similar to the hidden state in RNN, and the sentence vector $\mathbf{x}_t$ from the input without modification). The local query vector is not necessarily identical to the original query (question) vector $\mathbf{q}$. In order to compute the outputs, we use *update gate* function $\alpha : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ and *reduce* function $\boldsymbol{\rho} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. Intuitively, the update gate function measures the relevance between the sentence and the local query and is used to update the hidden state. The reduce function transforms the local query input to a candidate state which is a new reduced (easier) query given the sentence. The outputs are calculated with the following equations:

$$z_t = \alpha(\mathbf{x}_t, \mathbf{q}_t) = \sigma(\mathbf{W}^{(z)}(\mathbf{x}_t \circ \mathbf{q}_t) + b^{(z)}) \tag{5.1}$$

$$\tilde{\mathbf{h}}_t = \boldsymbol{\rho}(\mathbf{x}_t, \mathbf{q}_t) = \tanh(\mathbf{W}^{(\mathbf{h})}[\mathbf{x}_t; \mathbf{q}_t] + \mathbf{b}^{(\mathbf{h})}) \tag{5.2}$$

$$\mathbf{h}_t = z_t\tilde{\mathbf{h}}_t + (1 - z_t)\mathbf{h}_{t-1} \tag{5.3}$$

where z_t is the scalar update gate, $\tilde{\mathbf{h}}_t$ is the candidate reduced query, and $\mathbf{h}_t$ is the final reduced query at time step t, $\sigma(\cdot)$ is sigmoid activation, $\tanh(\cdot)$ is hyperboolic tangent activation (applied element-wise),

$\mathbf{W}^{(z)} \in \mathbb{R}^{1\times d}$, $\mathbf{W}^{(\mathbf{h})} \in \mathbb{R}^{d\times 2d}$ are weight matrices, $b^{(z)} \in \mathbb{R}$, $\mathbf{b}^{(\mathbf{h})} \in \mathbb{R}^d$ are bias terms, $\circ$ is element-wise vector multiplication, and $[;]$ is vector concatenation along the row. As a base case, $\mathbf{h}_0 = \mathbf{0}$. Here we have explicitly defined α and $\boldsymbol{\rho}$, but they can be any reasonable differentiable functions.

The update gate is similar to the global attention mechanism [Sukhbaatar et al., 2015; Xiong et al., 2016] in that it measures the similarity between the sentence (a memory slot) and the query. However, a significant difference is that the update gate is computed using sigmoid (σ) function on the current memory slot only (hence internally embedded within the unit), whereas the global attention is computed using $\mathrm{softmax}$ function over the entire memory (hence globally defined). The update gate can be rather considered as *local sigmoid* attention.

Stacking layers We just showed the single-layer case of QRN, but QRN with multiple layers is able to perform reasoning over multiple facts more effectively, as shown in the example of Figure 5.1b. In order to stack several layers of QRN, the outputs of the current layer are used as the inputs to the next layer. That is, using superscript k to denote the current layer's index (assuming 1-based indexing), we let $\mathbf{q}_t^{k+1} = \mathbf{h}_t^k$. Note that $\mathbf{x}_t$ is passed to the next layer without any modification, so we do not put a layer index on it.

Bi-direction. So far we have assumed that QRN only needs to look at past sentences, whereas often times, query answers can depend on future sentences. For instance, consider a sentence "John dropped the football." at time t. Then, even if there is no mention about the "football" in the past (at time $i < t$), it can be implied that "John" has the "football" at the current time t. In order to incorporate the future dependency, we obtain $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ in both forward and backward directions, respectively, using Equation 5.3. We then add them together to get $\mathbf{q}_t$ for the next layer. That is,

$$\mathbf{q}_t^{k+1} = \overrightarrow{\mathbf{h}}_t^k + \overleftarrow{\mathbf{h}}_t^k \tag{5.4}$$

for layer indices $1 \le k \le K-1$. Note that the variables $\mathbf{W}^{(z)}, b^{(z)}, \mathbf{W}^{(\mathbf{h})}, \mathbf{b}^{(\mathbf{h})}$ are shared between the two directions.

Connecting input and output modules. Figure 4.1 depicts how QRN is connected with the input and output modules. In the first layer of QRN, $\mathbf{q}_t^1 = \mathbf{q}$ for all t, where $\mathbf{q}$ is obtained from the input module by

processing the natural language question input $\boldsymbol{q}$. $\mathbf{x}_t$ is also obtained from $\boldsymbol{x}_t$ by the same input module. The output at the last time step in the last layer is passed to the output module. That is, $\hat{\mathbf{y}} = \mathbf{h}_t^K$ where K represent the number of layers in the network. Then the output module gives the predicted answer $\hat{\boldsymbol{y}}$ in natural language.

5.2.2 Extensions

Here we introduce a few extensions of QRN, and later in our experiments, we test QRN's performance with and without each of these extensions.

Reset gate. Inspired by GRU [Cho et al., 2014], we found that it is useful to allow the QRN unit to reset (nullify) the candidate reduced query (i.e., $\tilde{\mathbf{h}}_t$) when necessary. For this we use a *reset gate* function $\beta : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [0, 1]$, which can be defined similarly to the update gate function:

$$r_t = \beta(\mathbf{x}_t, \mathbf{q}_t) = \sigma(\mathbf{W}^{(r)}(\mathbf{x}_t \circ \mathbf{q}_t) + b^{(r)}) \tag{5.5}$$

where $\mathbf{W}^{(r)} \in \mathbb{R}^{1\times d}$ is a weight matrix, and $b^{(r)} \in \mathbb{R}$ is a bias term. Equation 5.3 is rewritten as

$$\mathbf{h}_t = z_t r_t \tilde{\mathbf{h}}_t + (1 - z_t)\mathbf{h}_{t-1}. \tag{5.6}$$

Note that we do not use the reset gate in the last layer.

Vector gates. As in LSTM and GRU, update and reset gates can be vectors instead of scalar values for fine-controlled gating. For vector gates, we modify the row dimension of weights and biases in Equation 5.1 and 5.5 from 1 to d. Then we obtain $\mathbf{z}_t, \mathbf{r}_t \in \mathbb{R}^d$ (instead of $z_t, r_t \in \mathbb{R}$), and these can be element-wise multiplied ($\circ$) instead of being broadcasted in Equation 5.3 and 5.6.

5.3 Parallelization

An important advantage of QRN is that the recurrent updates in Equation 5.3 and 5.5 can be computed in parallel across time. This is in contrast with most RNN-based models that cannot be parallelized, where

computing the candidate hidden state at time t explicitly requires the previous hidden state. In QRN, the final reduced queries ($\mathbf{h}_t$) can be decomposed into computing over candidate reduced queries ($\tilde{\mathbf{h}}_t$), without looking at the previous reduced query. Here we primarily show that the query update in Equation 5.3 can be parallelized by rewriting the equation with matrix operations. The extension to Equation 5.5 is straightforward. The recursive definition of Equation 5.3 can be explicitly written as

$$\mathbf{h}_t = \sum_{i=1}^{t} \left[\prod_{j=i+1}^{t} 1 - z_j \right] z_i \tilde{\mathbf{h}}_i = \sum_{i=1}^{t} \exp \left\{ \sum_{j=i+1}^{t} \log \left(1 - z_j\right) \right\} z_i \tilde{\mathbf{h}}_i. \tag{5.7}$$

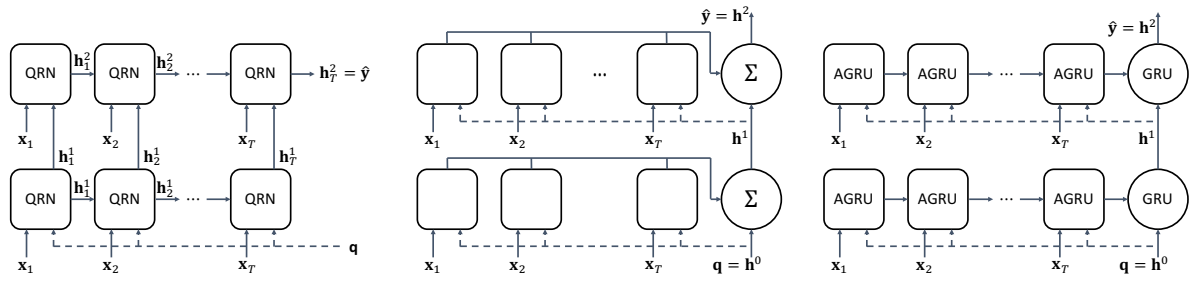
Let $b_i = \log(1 - z_i)$ for brevity. Then we can rewrite Equation 5.7 as the following equation:

$$\begin{pmatrix} \mathbf{h}_1^\top \\ \mathbf{h}_2^\top \\ \mathbf{h}_3^\top \\ \vdots \\ \mathbf{h}_T^\top \end{pmatrix} = \left[\exp \left\{ \begin{pmatrix} 0 & -\infty & -\infty & \dots & -\infty \\ b_2 & 0 & -\infty & \dots & -\infty \\ b_2 + b_3 & b_3 & 0 & \dots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{j=2}^{T} b_j & \sum_{j=3}^{T} b_j & \sum_{j=4}^{T} b_j & \dots & 0 \end{pmatrix} \right\} \right] \begin{pmatrix} z_1 \tilde{\mathbf{h}}_1^\top \\ z_2 \tilde{\mathbf{h}}_2^\top \\ z_3 \tilde{\mathbf{h}}_3^\top \\ \vdots \\ z_T \tilde{\mathbf{h}}_T^\top \end{pmatrix} \tag{5.8}$$

Let $\mathbf{H} = [\mathbf{h}_1^\top; \dots; \mathbf{h}_T^\top]$ be a T-by-d matrix where the transposes ($\top$) of the column vectors $\mathbf{h}_t$ are concatenated across row. We similarly define $\tilde{\mathbf{H}}$ from $\tilde{\mathbf{h}}_t$. Also, let $\mathbf{z} = [z_1; \dots; z_T]$ and $\mathbf{b} = [0; b_2; \dots; b_T]$ be column vectors (note that we use 0 instead of b_1). Then Equation 5.8 is:

$$\mathbf{H} = \left[\mathbf{L} \circ \exp \left(\mathbf{L} \left[\mathbf{B} \circ \mathbf{L}' \right] \right) \right] \left[\mathbf{Z} \circ \tilde{\mathbf{H}} \right] \tag{5.9}$$

where $\mathbf{L}, \mathbf{L}' \in \mathbb{R}^{T \times T}$ are lower and *strictly* lower triangular matrices of 1's, respectively, $\circ$ is element-wise multiplication, and $\mathbf{B}$ is a matrix where T $\mathbf{b}$'s are tiled across the column, i.e. $\mathbf{B} = [\mathbf{b}, \dots, \mathbf{b}] \in \mathbb{R}^{T \times T}$, and similarly $\mathbf{Z} = [\mathbf{z}, \dots, \mathbf{z}] \in \mathbb{R}^{T \times d}$. All implicit operations are matrix multiplications. With reasonable N (batch size), d and T (e.g. $N, d, T = 100$), matrix operations in Equation 5.9 can be comfortably computed in most modern GPUs.



(a) QRN **(b)** N2N [Sukhbaatar et al., 2015] **(c)** DMN+ [Xiong et al., 2016]

Figure 5.2: The schematics of QRN and the two state-of-the-art models, End-to-End Memory Networks (N2N) and Improved Dynamic Memory Networks (DMN+), simplified to emphasize the differences among the models. AGRU is a variant of GRU where the update gate is replaced with soft attention, proposed by Kumar et al. [2016]. For QRN and DMN+, only forward direction arrows are shown.

5.4 Related Work

QRN is inspired by RNN-based models with gating mechanism, such as LSTM [Hochreiter and Schmidhuber, 1997] and GRU [Cho et al., 2014]. While GRU and LSTM use the previous hidden state and the current input to obtain the candidate hidden state, QRN only uses the current two inputs to obtain the candidate reduced query (equivalent to candidate hidden state). We conjecture that this not only gives computational advantage via parallelization, but also makes training easier, i.e., avoiding vanishing gradient (which is critical for long-term dependency), overfitting (by simplifying the model), and converging to local minima.

The idea of structurally simplifying (constraining) RNNs for learning longer-term patterns has been explored in recent previous work, such as Structurally Constrained Recurrent Network [Mikolov et al., 2015] and Strongly-Typed Recurrent Neural Network (STRNN) [Balduzzi and Ghifary, 2016]. QRN is similar to STRNN in that both architectures use gating mechanism, and the gates and the candidate hidden states do not depend on the previous hidden states, which simplifies the recurrent relation. However, QRN can be distinguished from STRNN in three ways. First, QRN's update gate simulates attention mechanism, measuring the relevance between the input sentence and query. On the other hand, the gates in STRNN can be considered as the simplification of LSTM/GRU by removing their dependency on previous hidden state. Second, QRN is an RNN that is natively compatible with context-based QA tasks, where the QRN unit accepts two inputs, i.e. each context sentence and query. This is distinct from STRNN which has only one input. Third, we show that QRN is timewise-parallelizable on GPUs. Our parallelization algorithm is also applicable to STRNN.

End-to-end Memory Network (N2N) [Sukhbaatar et al., 2015] uses external memory with multi-layer attention mechanism to focus on sentences that are relevant to the question. There are two key differences between N2N and our QRN. First, N2N summarizes the entire memory in each layer to control the attention in the next layer (circle nodes in Figure 5.2b). Instead, QRN does not have any controller node (Figure 5.2a) and is able to focus on relevant sentences through the update gate that is internally embodied within its unit. Second, N2N adds time-dependent trainable weights to the sentence representations to model the time dependency of the sentences (as discussed in Section 5.1). QRN does not need such additional weights as its inherent RNN architecture allows QRN to effectively model the time dependency. Neural Reasoner [Peng et al., 2015] and Gated End-to-end Memory Network [Perez and Liu, 2016]) are variants of MemN2N that share its fundamental characteristics.

Improved Dynamic Memory Network (DMN+) [Xiong et al., 2016] uses the hybrid of the attention mechanism and the RNN architecture to model the sequence of sentences. It consists of two distinct GRUs, one for the time axis (rectangle nodes in Figure 5.2c) and one for the layer axis (circle nodes in Figure 5.2c). Note that the update gate of the GRU for the time axis is replaced with external softmax attention weights. DMN+ uses the time-axis GRU to summarizes the entire memory in each layer, and then the layer-axis GRU controls the attention weights in each layer. In contrast, QRN is simply a single recurrent unit without any controller node.

5.5 Experiments

5.5.1 Data

bAbI story-based QA dataset bAbI story-based QA dataset [Weston et al., 2016] is composed of 20 different tasks, each of which has 1,000 (1k) synthetically-generated story-question pair. A story can be as short as two sentences and as long as 200+ sentences. A system is evaluated on the accuracy of getting the correct answers to the questions. The answers are single words or lists (e.g. “football, apple”). Answering questions in each task requires selecting a set of relevant sentences and applying different kinds of logical reasoning over them. The dataset also includes 10k training data (for each task), which allows training more complex models. Note that DMN+ [Xiong et al., 2016] only reports on the 10k dataset.

bAbI dialog dataset bAbI dialog dataset [Bordes and Weston, 2017] consists of 5 different tasks, each of which has 1k synthetically-generated goal-oriented dialogs between a user and the system in the domain of restaurant reservation. Each dialog is as long as 96 utterances and comes with external knowledge base (KB) providing information of each restaurant. The authors also provide Out-Of-Vocabulary (OOV) version of the dataset, where many of the words and KB keywords in test data are not seen during training. A system is evaluated on the accuracy of its response to each utterance of the user, choosing from up to 2500 possible candidate responses. A system is required not only to understand the user's request but also refer to previous conversations in order to obtain the context information of the current conversation.

DSTC2 (Task 6) dialog dataset Bordes and Weston [2017] transformed the Second Dialog State Tracking Challenge (DSTC2) dataset [Henderson et al., 2014] into the same format as the bAbI dialog dataset, for the measurement of performance on a real dataset. Each dialog can be as long as 800+ utterances, and a system needs to choose from 2407 possible candidate responses for each utterance of the user. Note that the evaluation metric of the original DSTC2 is different from that of the transformed DSTC2, so previous work on the original DSTC2 should not be directly compared to our work. We will refer to this transformed DSTC2 dataset by "Task 6" of dialog dataset.

5.5.2 Model Details

Input Module. In the input module, we are given sentences (previous conversations in dialog) $\boldsymbol{x}_t$ and a question (most recent user utterance) $\boldsymbol{q}$, and we want to obtain their vector representations, $\mathbf{x}_t, \mathbf{q} \in \mathbb{R}^d$. We use a trainable embedding matrix $\mathbf{A} \in \mathbb{R}^{d \times V}$ to encode the one-hot vector of each word $\boldsymbol{x}_{tj}$ in each sentence $\boldsymbol{x}_t$ into a d-dimensional vector $\mathbf{x}_{tj} \in \mathbb{R}^d$. Then the sentence representation $\mathbf{x}_t$ is obtained by Position Encoder [Sukhbaatar et al., 2015]. The same encoder with the same embedding matrix is also used to obtain the question vector $\mathbf{q}$ from $\boldsymbol{q}$.

Output Module for story-based QA. In the output module, we are given the vector representation of the predicted answer $\hat{\mathbf{y}}$ and we want to obtain the natural language form of the answer, $\hat{\boldsymbol{y}}$. We use a V-way single-layer softmax classifier to map $\hat{\mathbf{y}}$ to a V-dimensional sparse vector, $\hat{\mathbf{v}} = \mathrm{softmax}\left(\mathbf{W}^{(y)}\hat{\mathbf{y}}\right) \in \mathbb{R}^V$, where $\mathbf{W}^{(y)} \in \mathbb{R}^{V \times d}$ is a weight matrix. Then the final answer $\hat{\boldsymbol{y}}$ is simply the argmax word in $\hat{\mathbf{v}}$. To handle

questions with multiple-word answers, we consider each of them as a single word that contains punctuations such as space and comma, and put it in the vocabulary.

Output Module for dialog. We use a fixed number single-layer softmax classifiers, each of which is similar to that of the sotry-based QA model, to sequentially output each word of the system's response. While it is similar in spirit to the RNN decoder [Cho et al., 2014], our output module does not have a recurrent hidden state or gating mechanism. Instead, it solely uses the final ouptut of the QRN, $\hat{\mathbf{y}}$, and the current word output to influence the prediction of the next word among possible candidates.

Training. We withhold 10% of the training for development. We use the hidden state size of 50 by deafult. Batch sizes of 32 for bAbI story-based QA 1k, bAbI dialog and DSTC2 dialog, and 128 for bAbI QA 10k are used. The weights in the input and output modules are initialized with zero mean and the standard deviation of $1/\sqrt{d}$. Weights in the QRN unit are initialized using techniques by Glorot and Bengio [2010], and are tied across the layers. Forget bias of 2.5 is used for update gates (no bias for reset gates). L2 weight decay of 0.001 (0.0005 for QA 10k) is used for all weights. The loss function is the cross entropy between $\hat{\mathbf{v}}$ and the one-hot vector of the true answer. The loss is minimized by stochastic gradient descent for maximally 500 epochs, but training is early stopped if the loss on the development data does not decrease for 50 epochs. The learning rate is controlled by AdaGrad [Duchi et al., 2011] with the initial learning rate of 0.5 (0.1 for QA 10k). Since the model is sensitive to the weight initialization, we repeat each training procedure 10 times (50 times for 10k) with the new random initialization of the weights and report the result on the test data with the lowest loss on the development data.

5.5.3 Results.

We compare our model with baselines and previous state-of-the-art models on story-based and dialog tasks (Table 5.1). These include LSTM [Hochreiter and Schmidhuber, 1997], End-to-end Memory Networks (N2N) [Sukhbaatar et al., 2015], Dynamic Memory Networks (DMN+) [Xiong et al., 2016], Gated End-to-end Memory Networks (GMemN2N) [Perez and Liu, 2016], and Differentiable Neural Computer (DNC) [Graves et al., 2016].

Task	1k						10k				
	Previous works				QRN		Previous works				QRN
	LSTM	N2N	DMN+†	GMemN2N	2r	3r	N2N	DMN+	GMemN2N	DNC	6r200
# Failed	20	10	16	10	7	**5**	3	1	3	2	**0**
Average error rates	51.3	15.2	33.2	12.7	**9.9**	11.3	4.2	2.8	3.7	3.8	**0.3**

Task	Plain				With Match		
	Previous works		QRN		Previous works		QRN
	N2N	GMemN2N	2r	2r100	N2N+	GMemN2N+	2r+
bAbI dialog Average error rates	13.9	14.3	**5.5**	**5.5**	6.7	5.4	**1.5**
bAbI dialog (OOV) Average error rates	30.3	27.9	**11.1**	**11.1**	11.2	10.3	**2.3**
DSTC2 dialog Average error rates	58.9	52.6	49.5	**48.9**	59.0	51.3	**49.3**

Table 5.1: (top) bAbI QA dataset [Weston et al., 2016]: number of failed tasks and average error rates (%). † is obtained from `github.com/therne/dmn-tensorflow`. (bottom) bAbI dialog and DSTC2 dialog dataset [Bordes and Weston, 2017] average error rates (%) of QRN and previous work (LSTM, N2N, DMN+, GMemN2N, and DNC). For QRN, the first number (1, 2, 3) indicates the number of layers, 'r' means the reset gate is used, and the last number (100, 200), if exists, indicates the dimension of the hidden state, where the default value is 50. '+' indicates that 'match' (See Appendix for details) is used. See Section 5.5.3 for details.

Story-based QA. Table 5.1(top) reports the summary of results of our model (QRN) and previous work on bAbI QA. In 1k data, QRN's '2r' (2 layers + reset gate + $d = 50$) outperforms all other models by a large margin (2.8+%). In 10k dataset, the average accuracy of QRN's '6r200' (6 layers + reset gate + $d = 200$) model outperforms all previous models by a large margin (2.5+%), achieving a nearly perfect score of 99.7%.

Dialog. Table 5.1(bottom) reports the summary of the results of our model (QRN) and previous work on bAbI dialog and Task 6 dialog. As done in previous work [Bordes and Weston, 2017; Perez and Liu, 2016], we also report results when we use 'Match' for dialogs. 'Match' is the extension to the model which additionally takes as input whether each answer candidate matches with context (more details on Appendix). QRN outperforms previous work by a large margin (2.0+%) in every comparison.

Ablations. We test four types of ablations (also discussed in Section 5.2.2): number of layers (1, 2, 3, or 6), reset gate (r), and gate vectorization (v) and the dimension of the hidden vector (50, 100). We show a subset of combinations of the ablations for bAbI QA in Table 5.1; other combinations performed poorly and/or did not give interesting observations. According to the ablation results, we infer that: **(a)** When the number of layers is only one, the model lacks reasoning capability. In the case of 1k dataset, when there are too many layers (6), it seems correctly training the model becomes increasingly difficult. In the case of 10k

	Layer 1			Layer 2
Task 2: Two Supporting Facts	z^1	$\overrightarrow{r}^1$	$\overleftarrow{r}^1$	z^2
Sandra picked up the apple there.	0.95	0.89	0.98	0.00
Sandra dropped the apple.	0.83	0.05	0.92	0.01
Daniel grabbed the apple there.	0.88	0.93	0.98	0.00
Sandra travelled to the bathroom.	0.01	0.18	0.63	0.02
Daniel went to the hallway.	0.01	0.24	0.62	0.83
Where is the apple?	hallway			

	Layer 1			Layer 2
Task 15: Deduction	z^1	$\overrightarrow{r}^1$	$\overleftarrow{r}^1$	z^2
Mice are afraid of wolves.	0.11	0.99	0.13	0.78
Gertrude is a mouse.	0.77	0.99	0.96	0.00
Cats are afraid of sheep.	0.01	0.99	0.07	0.03
Winona is a mouse.	0.14	0.85	0.77	0.05
Sheep are afraid of wolves.	0.02	0.98	0.27	0.05
What is Gertrude afraid of?	wolf			

	Layer 1			Layer 2
Task 3: Displaying options	z^1	$\overrightarrow{r}^1$	$\overleftarrow{r}^1$	z^2
resto-paris-expen-frech-8stars?	0.00	1.00	0.96	0.91
Do you have something else?	0.41	0.99	0.00	0.00
Sure let me find another option.	1.00	0.00	0.00	0.12
resto-paris-expen-frech-5stars?	0.00	1.00	0.96	0.91
No this does not work for me.	0.00	0.00	0.14	0.00
Sure let me find an other option.	1.00	0.00	0.00	0.12
What do you think of this?	resto-paris-expen-french-4stars			

	Layer 1			Layer 2
Task 6: DSTC2 dialog	z^1	$\overrightarrow{r}^1$	$\overleftarrow{r}^1$	z^2
Spanish food.	0.84	0.07	0.00	0.82
You are lookng for a spanish restaurant right?	0.98	0.02	0.49	0.75
Yes.	0.01	1.00	0.33	0.13
What part of town do you have in mind?	0.20	0.73	0.41	0.11
I don't care.	0.00	1.00	0.02	0.00
What price range would you like?	0.72	0.46	0.52	0.72
I don't care.	API CALL spanish R-location R-price			

Figure 5.3: (top) bAbI QA dataset [Weston et al., 2016] visualization of update and reset gates in QRN '2r' model (bottom two) bAbI dialog and DSTC2 dialog dataset [Bordes and Weston, 2017] visualization of update and reset gates in QRN '2r' model. Note that the stories can have as many as 800+ sentences; we only show part of them here.

dataset, many layers (6) and hidden dimensions (200) helps reasoning, most notably in difficult task such as task 16. **(b)** Adding the reset gate helps. **(c)** Including vector gates hurts in 1k datasets, as the model either overfits to the training data or converges to local minima. On the other hand, vector gates in bAbI story-based QA 10k dataset sometimes help. **(d)** Increasing the dimension of the hidden state to 100 in the dialog's Task 6 (DSTC2) helps, while there is not much improvement in the dialog's Task 1-5. It can be hypothesized that a larger hidden state is required for real data.

Parallelization. We implement QRN with and without parallelization in TensorFlow [Abadi et al., 2016] on a single Titan X GPU to qunaitify the computational gain of the parallelization. For QRN without parallelization, we use the RNN library provided by TensorFlow. QRN with parallelization gives 6.2 times faster training and inference than QRN without parallelization on average. We expect that the speedup can be even higher for datasets with larger context.

Interpretations. An advantage of QRN is that the intermediate query updates are interpretable. Figure 5.1 shows intermediate local queries ($\mathbf{q}_t^k$) interpreted in natural language, such as "Where is Sandra?". In order to obtain these, we place a decoder on the input question embedding $\mathbf{q}$ and add its loss for recovering the

question to the classification loss (similarly to Peng et al. [2015]). We then use the same decoder to decode the intermediate queries. This helps us understand the flow of information in the networks. In Figure 5.1, the question `Where is apple?` is transformed into `Where is Sandra?` at $t = 1$. At $t = 2$, as `Sandra dropped the apple`, the apple is no more relevant to Sandra. We obtain `Where is Daniel?` at time $t = 3$, and it is propagated until $t = 5$, where we observe a sentence (fact) that can be used to answer the query.

Visualization. Figure 5.3 shows vizualization of the (scalar) magnitudes of update and reset gates on story sentences and dialog utterances. In Figure 5.3, we observe high values on facts that provide information to answer question (the system's next utterance for dialog). In QA Task 2 example (top left), we observe high update gate values in the first layer on facts that state who has the `apple`, and in the second layer, the high update gate values are on those that inform where that person went to. We also observe that the forward reset gate at $t = 2$ in the first layer ($\overrightarrow{r}_2^1$) is low, which is signifying that `apple` no more belongs to `Sandra`. In dialog Task 3 (bottom left), the model is able to infer that three restaurants are already recommended so that it can recommend another one. In dialog Task 6 (bottom), the model focuses on the sentences containing `Spanish`, and does not concentrate much on other facts such as `I don't care`.

5.6 Conclusion

In this chapter, I introduced Query-Reduction Network (QRN) [Seo et al., 2017b] to answer context-based questions and carry out conversations with users that require multi-hop reasoning. We show the state-of-the-art results in the three datasets of story-based QA and dialog. We model a story or a dialog as a sequence of state-changing triggers and compute the final answer to the question or the system's next utterance by recurrently updating (or *reducing*) the query. QRN is situated between the attention mechanism and RNN, effectively handling time dependency and long-term dependency problems of each technique, respectively. It addresses the long-term dependency problem of most RNNs by simplifying the recurrent update, in which the candidate hidden state (reduced query) does not depend on the previous state. Moreover, QRN can be parallelized and can address the well-known problem of RNN's vanishing gradients.

Chapter 6

Conclusion

General-purpose language representation learning models such as Transformer (and BERT) have recently shown to be very effective for natural language tasks that are less dependent on factual knowledge (e.g. classification, textual entailment, machine translation). In many tasks, the models have already reached the upper bound and outperformed human accuracy. Nevertheless, more challenges still remain for the tasks that are more explicitly dependent on the world knowledge, which is often too large to be considered as a raw input to the parametric model. In this thesis, I tackled these challenges by introducing three different methods for interacting with the world knowledge in the form of natural language data: machine reader, knowledge memory, and neural reasoning.

A promising future research direction is to design a convenient neural interface for accessing the world knowledge that can be attached to the parametric model as a *plug-and-play* module, so that the parametric model can be *knowledge-aware* during training and inference for various knowledge-dependent tasks. These not only include direct applications such as open-domain QA and multi-step reasoning (as discussed in 5), but also may include non-obvious cases such as language models and machine translation where the access to the vast amount of world knowledge can indirectly boost the performance. For the rest of this chapter, I delve into four possible future research topics that are closely related to this challenge.

6.1 Unified knowledge representation in vector space

A vast amount of world knowledge is available on the web in many different forms. Structured data such as Knowledge Graph and databases is effective for fast and precise access to specific knowledge, and is also better able to handle complex queries by leveraging its formal ontology [Seo et al., 2015; Hwang et al., 2019]. On the other hand, unstructured data has several orders of magnitude more information and its scope is not limited by the predefined ontology [Seo et al., 2019]. Semi-structured and multimodal data such as tables, images and lists embodied in text also contain important information. For instance, consider a factoid question "How large is Seattle?". Information necessary to answer the question might be in Knowledge Graph ((Seattle, size, 217km^2)), in a Wiki document ("Seattle has an area of 217km^2.") or in a semi-structured table listing city sizes. In order to access such diverse forms of information with a common interface that is not tied to any manually built ontology, developing a unified, distributed (vector) knowledge representation model is crucial. Embedding even a single piece of information in vector space is known to be a difficult task, but its feasibility is also explored and demonstrated in my previous work [Seo et al., 2019; Lee et al., 2020]. In the process of realising knowledge embodied in the data, it is also important to be sufficiently aware of the context, including not only document-wide information but also various kinds of metadata (e.g. time stamp when the document was written) and images and figures [Kembhavi et al., 2017]. This will require us to consider more different kinds of information than the current scope of large pretrained language models (ELMo, BERT) that only look at raw words.

6.2 Large-scale neural memory

Unified, distributed knowledge representations will allow us to build a key-value memory architecture that houses the world knowledge in vector space, but two significant challenges exist. First, effective key representation model will be needed. Poor key vectors will hinder a precise access to the requested information. This could mean that we need a better distance function (than typical ones such as L1, L2, inner product), and/or we need multiple key vectors per knowledge piece. Second, making a web-scale neural memory is a computationally challenging problem. We will need a tighter integration and collaboration with lower-level technology stacks in Computer Science, including distributed systems and (physical) memory architecture

(for low random-access latency). Furthermore, computing the exact gradient over a large memory is inefficient, so we will need a faster method for approximating the gradient in order to attach it to and train it with a task-specific parametric model. We can also benefit from developing a dedicated similarity search library for the neural memory, especially if we want to handle extremely sparse key vectors [Seo et al., 2019; Lee et al., 2020].

6.3 Combining knowledge in vector space

While the neural memory is meant to contain a quite comprehensive collection of world knowledge, many of complex queries still cannot be directly answered by the memory (with a single access) and require combining knowledge from different places to infer a new conclusion. The existence of the neural memory, however, will allow the task-specific parametric model to minimize effort on finding and processing knowledge and focus on learning to reason. My previous work [Seo et al., 2017b] showed the feasibility of vector-space reasoning when the language structure is simple enough; better knowledge representation models (#A) could extend it to more complex language structures. I believe that such plug-and-play neural interface for world knowledge will make model development for researchers easier and more effective. An exciting direction is to observe if such knowledge interface can benefit not only direct applications such as open-domain QA&dialog and commonsense and multi-step reasoning, but also non-obvious problems such as machine translation and language models where the task-specific model might find a clever usage of the world knowledge to improve the output quality. I also plan to explore how we can gradually augment the neural memory with new knowledge by combining existing knowledge in the memory.

6.4 Parametric knowledge encoding

A very recent and increasingly popular approach is to self-supervise a large model (tens of billions of parameters) with an even larger text corpus of the world knowledge (such as Wikipedia) by asking the model to generate the next sentence. Then the model is finetuned on the target task [Raffel et al., 2019].[1] Surprisingly, experiments have shown that the model is able to latently encode (i.e. memorize) most of the

[1]This methodology is also often called "closed-book QA", as opposed to "open-book QA" that would allow the model to 'look up' Wikipedia.

facts in the text corpus and effectively utilize them for the target task. While a promising paradigm, it also exhibits some non-trivial drawbacks. The model is very large and training it is computationally expensive that it is almost impossible to experiment with it or make a use of it at most places except for companies like Google that have enough computing resources. Furthermore, the implicitness of the knowledge make it hard to control the knowledge—that is, if we want to delete or update a fact, or if we want to ensure that a certain piece of knowledge is encoded in the model. One might view it to be similar to squeezing a lot of information into a human brain. Humans forget, memory gets distorted, and it has a finite capacity. Therefore, for a precise access to information, an on-demand interface for interacting with the world knowledge, stored in explicit formats such as natural language, image, or memory slots, is still essential. Latent knowledge (encoded in the model) and explicit knowledge have pros and cons that compliment each other, and we should actively conduct research on both directions and also consider a hybrid model that can combine the best of the two worlds.

Bibliography

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *ICCV*.

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

David Balduzzi and Muhammad Ghifary. 2016. Strongly-typed recurrent neural networks. In *ICML*.

Petr Baudiš and Jan Šedivỳ. 2015. Modeling of the question answering task in the yodaqa system. In *CLEF*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*.

Antoine Bordes and Jason Weston. 2017. Learning end-to-end goal-oriented dialog. In *ICLR*.

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

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *ACL*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*.

Jennifer Chu-Carroll, James Fan, BK Boguraev, David Carmel, Dafna Sheinwald, and Chris Welty. 2012. Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. 2016. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. 2019. Multi-step retriever-reader interaction for scalable open-domain question answering. In *ICLR*.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Bhuwan Dhingra, Hanxiao Liu, William W Cohen, and Ruslan Salakhutdinov. 2017. Gated-attention readers for text comprehension. In *ACL*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A Smith. 2015. Sparse overcomplete word vector representations. In *ACL*.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. 2016. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *EMNLP*.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *JMLR*.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*.

Matthew Henderson, Blaise Thomson, and Jason Williams. 2014. The second dialog state tracking challenge. In *SIGdial*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*.

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

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. In *NeurIPS Workshop on KR2ML*.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *TPAMI*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

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

Aniruddha Kembhavi, Minjoon Seo, Dustin Schwenk, Jonghyun Choi, Hannaneh Hajishirzi, and Ali Farhadi. 2017. Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension. In *CVPR*.

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

Sosuke Kobayashi, Ran Tian, Naoaki Okazaki, and Kentaro Inui. 2016. Dynamic entity representation with max-pooling improves machine reading. In *NAACL*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*.

Jinhyuk Lee, Minjoon Seo, Hannaneh Hajishirzi, and Jaewoo Kang. 2020. Contextualized sparse representations for real-time open-domain question answering. In *ACL*.

Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. 2018. Ranking paragraphs for improving answer recall in open-domain question answering. In *EMNLP*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *ACL*.

Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. 2016. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *CoNLL*.

Mike Lewis and Angela Fan. 2019. Generative question answering: Learning to answer the whole question. In *ICLR*.

Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. 2018. Denoising distantly supervised open-domain question answering. In *ACL*.

Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. 2016. Hierarchical question-image co-attention for visual question answering. In *NIPS*.

Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. 2015. Ask your neurons: A neural-based approach to answering questions about images. In *ICCV*.

Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *TPAMI*.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. 2015. Learning longer memory in recurrent neural networks. In *ICLR 2015 Workshop*.

Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. 2017. Question answering through transfer learning from fine-grained supervision data. In *ACL*.

Sewon Min, Victor Zhong, Richard Socher, and Caiming Xiong. 2018. Efficient and robust question answering from minimal context over documents. In *ACL*.

Kyosuke Nishida, Itsumi Saito, Atsushi Otsuka, Hisako Asano, and Junji Tomita. 2018. Retrieve-and-read: Multi-task learning of information retrieval and reading comprehension. In *CIKM*.

Baolin Peng, Zhengdong Lu, Hang Li, and Kam-Fai Wong. 2015. Towards neural network-based reasoning. *arXiv preprint arXiv:1508.05508*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Julien Perez and Fei Liu. 2016. Gated end-to-end memory networks. *arXiv preprint arXiv:1610.04211*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

John Prager et al. 2007. Open-domain question–answering. *Foundations and Trends® in Information Retrieval*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Jonathan Raiman and John Miller. 2017. Globally normalized reader. In *EMNLP*.

Martin Raison, Pierre-Emmanuel Mazaré, Rajarshi Das, and Antoine Bordes. 2018. Weaver: Deep co-encoding of questions and documents for machine reading. *arXiv preprint arXiv:1804.10490*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Raymond Reiter. 2001. *Knowledge in Action*, 1st edition. MIT Press.

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013a. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*.

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013b. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: combining text and diagram interpretation. In *EMNLP*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017a. Bidirectional attention flow for machine comprehension. In *ICLR*.

Minjoon Seo, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Phrase-indexed question answering: A new challenge for scalable document comprehension. In *EMNLP*.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. In *ACL*.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2017b. Query-reduction networks for question answering. In *ICLR*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *KDD*.

Anshumali Shrivastava and Ping Li. 2014. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*.

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

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *arXiv preprint arXiv:1505.00387*.

Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. 2018. Spine: Sparse interpretable neural embeddings. In *AAAI*.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *NIPS*.

Adam Trischler, Zheng Ye, Xingdi Yuan, and Kaheer Suleman. 2016. Natural language comprehension with the epireader. In *EMNLP*.

Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *SIGIR*.

Ellen M Voorhees et al. 1999. The trec-8 question answering track report. In *Trec*.

Shuohang Wang and Jing Jiang. 2017. Machine comprehension using match-lstm and answer pointer. In *ICLR*.

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerry Tesauro, Bowen Zhou, and Jing Jiang. 2018a. R 3: Reinforced ranker-reader for open-domain question answering. In *AAAI*.

Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. 2018b. Evidence aggregation for answer re-ranking in open-domain question answering. In *ICLR*.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage bert: A globally normalized bert model for open-domain question answering. In *EMNLP*.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. In *ICLR*.

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

Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *ICML*.

Caiming Xiong, Victor Zhong, and Richard Socher. 2017. Dynamic coattention networks for question answering. In *ICLR*.

Huijuan Xu and Kate Saenko. 2016. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *ECCV*.

Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015a. Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP*.

Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. 2017. Words or characters? fine-grained gating for reading comprehension. In *ICLR*.

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. 2015b. Stacked attention networks for image question answering. *arXiv preprint arXiv:1511.02274*.

Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. 2016. End-to-end reading comprehension with dynamic answer chunk ranking. *arXiv preprint arXiv:1610.09996*.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Yuke Zhu, Oliver Groth, Michael S. Bernstein, and Li Fei-Fei. 2016. Visual7w: Grounded question answering in images. In *CVPR*.