Advances in QA Models II: Machine Reading Comprehension and Multi-hop Reasoning

Xisen Jin
University of Southern California
xisenjin@usc.edu

Abstract

Machine reading comprehension (MRC) answers a query about a given context, which usually requires modeling complex interactions between the context and the query. Some harder questions require explicit modeling of multi-hop reasoning process. In this lecture, we will discuss advances in machine reading comprehension and multi-hop reasoning.

1 Introduction

Task formulation. Given a context $\mathbf{c}$, and a question $\mathbf{q}$, machine reading comprehension models output an answer $\mathbf{a}$. The format of the answer is specific to applications. For example,

- cloze style, where the model need to fill in a blank in the context.
- multiple choice. The model selects an answer from given choices.
- span based, where the model usually predicts the start and the end of the answer span in the given context.

2 Background

We first introduce some popular models that are usually selected as baselines for state of the art reading comprehension models.

Background: Attention mechanism. To model the interaction between the contexts and questions, one of the basic building blocks is *attention mechanism*. Attention mechanism generates attention weight matrix $\alpha(Q, K) \in \mathbb{R}^{n_Q \times n_K}$ to measure the similarities between each pair of key vectors $K \in \mathbb{R}^{n_K \times d_K}$ and query vectors $Q \in \mathbb{R}^{n_Q \times d_Q}$. For example, in scaled-dot product attention (Vaswani et al., 2017), the attention weight $\alpha(Q, K)$ is calculated as,

$$\alpha(Q, K) = softmax(\frac{QK^T}{\sqrt{d_k}}) \tag{1}$$

When given a set of value vectors, we can compute a weighted sum of values as $\alpha(Q, K)V$.

Bidirectional Attention Flow (BiDAF). BiDAF is a well-known neural MRC model. The basic idea is to bulid query aware context representation with attention mechanism. Figure 2 shows an overview of the model architecture. It first employ character-level and word-level encoder to encode contexts and queries. Then it computes the the attention matrix, normalized over the context dimension (Query2Context) and the query dimension (Context2Query) respectively. It then employs LSTM to scan the context to get query-aware context representation. Finally, it predicts the start and the end of the answer span.

CSCI 699 Lecture notes. October 29, 2019.

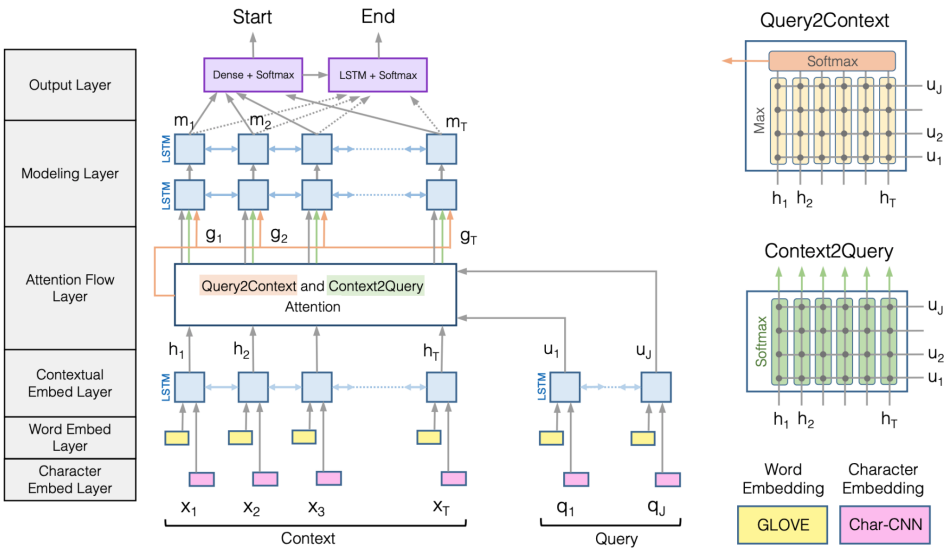


Figure 1: BiDAF model architecture

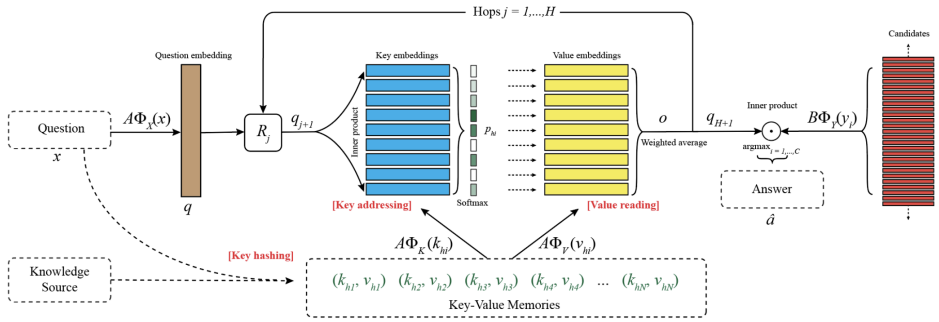


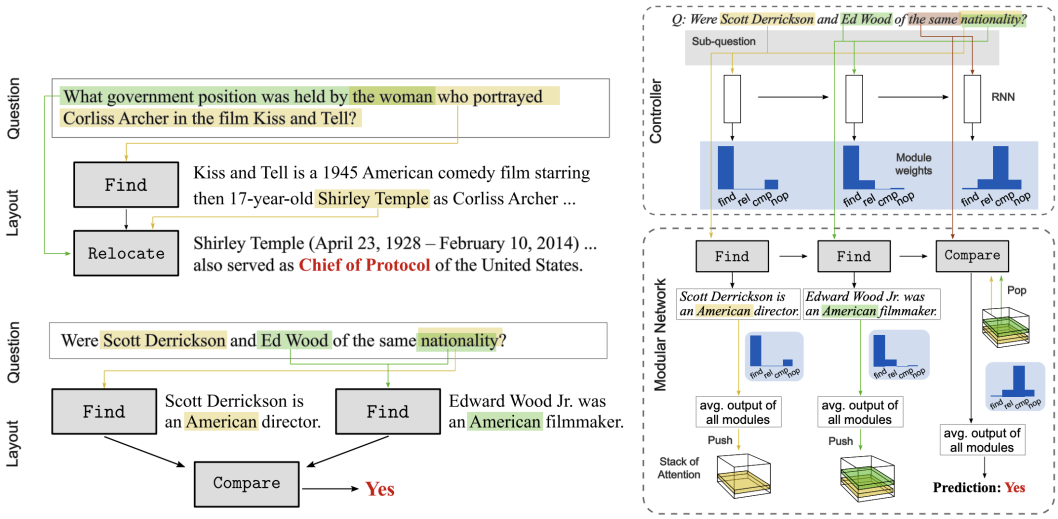
Figure 2: Key value memory networks.

Some reading comprehension problems require multi-hop reasoning, i.e., they require a chain of evidences to reach predictions.

Memory networks. Memory networks can deal with multi-hop reasoning. We exemplify with key-value memory networks (KVMN) (Miller et al., 2016). KVMN treats the context representations as both key vectors K and value vectors V in the memory. At each hop, the query vector q reads the memory with the attention mechanism to get an output o. To perform multi-hop reasoning, we update the query with the output o, for example, $q_2 = R_1(q + o)$, where R_1 is a trainable matrix.

3 Recent Advances

Compared to single-hop QA, multi-hop QA is more challenging. Unlike single-hop QA, which emphasize the role of matching the information between the question and the the context, multi-hop QA requires the model to collect information from multiple sentences or paragraphs, and build up a chain of evidence to reach the prediction. While memory networks are capable for multi-hop reasoning, it neither achieves satifying performance, nor have strong interpretability. Recent work study the problem in a variety of ways; among all the algorithms, we will discuss two recrent papers that tackle the problem in novel approaches. The achieve the goal of (1) more interpretable and modular reasoning process (2) intergrating first-order logic into reasoning respectively.



(a) (b)

Figure 3: Illustration on how neural modular networks solve HotpotQA problems.

3.1 Self-Assembling Modular Networks for Interpretable Multi-Hop Reasoning, EMNLP 2019

Consider such a question: *Were A and B of the same nationality?* To solve the question, we may first find out the nationality of A and B, then compare them. To model such human reasoning process, Jiang & Bansal (2019) porpose an interpretable, controller based Self-Assembling Neural Modular Network for multi-hop reasoning, with four applicable modules (Find, Relocate, Compare, NoOp). Based on a question, the layout controller RNN dynamically infers a series of reasoning modules to construct the entire network. Each module has different architectures and solve different sub-tasks.

Background: Neural modular networks. Neural Modular Network Neural Modular Network (NMN) is a class of models that are composed of a number of sub-modules, where each sub-module is capable of performing a specific subtask. Submodules are identified either with a parser or a layout policy that turns the question into a module layout. Then the module layout is executed with a neural module network. Overall, given an input question, the layout policy learns what sub-tasks to perform, and the neural modules learn how to perform each individual sub-task.

3.1.1 Model Layout Controller.

The controller reads the question and predicts a series modules (from Find, Relocate, Compare, NoOp) that could be executed in order to answer the given question. For multi-hop QA, each module represents a specific 1-hop reasoning behavior and the controller needs to deduce a chain of reasoning behavior that uncovers the evidence necessary to infer the answer.

The model use an encoder to get a fix-sized query representation q. Note that the controller is implemented as a RNN to keep track of previously selected modules. The probability for selecting a module at time step t is modeled with a MLP, which takes the query representation and the previous hidden state c_{t-1} for the controller RNN as input.

$$p_{t,i} = Softmax(MLP(c_{t-1}, q)) \tag{2}$$

The controller also provide a *sub-question* vector c_t. Sub-question notes for the set of context words that a module should particularly focus on. To get the representation, the controller first calculates attention over all question words and then computes the weighted average of all the word representations, noted as c_t, which also acts as the hidden state of the controller.

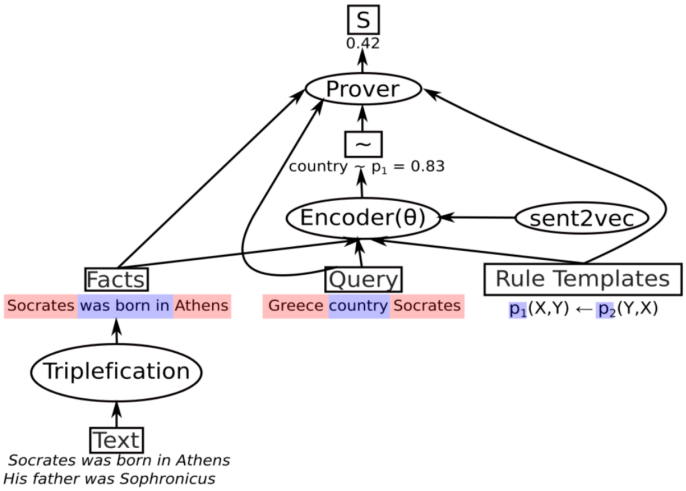


Figure 4: Overview of NLProlog. All components are depicted as ellipses, while inputs and outputs are drawn as squares. Phrases with red background are entities and blue ones are predicates.

3.1.2 Four NMN modules

All the modules take the question representation $\mathbf{u} \in \mathbb{R}^{J \times d}$, context representation $\mathbf{h} \in \mathbb{R}^{S \times d}$, and sub-question vector $c_t \in \mathbb{R}^d$ as input.

Find module. It performs the bi-attention operation between queries $\mathbf{u}$ and contexts $\mathbf{h}$.

$$\begin{aligned} \mathbf{h}' &= \mathbf{h} \odot c_t \\ M_{j,s} &= \mathrm{v}_1 \mathbf{u}_j + \mathrm{v}_2 \mathbf{h}'_s + \mathrm{v}_3 \left(\mathbf{u}_j \odot \mathbf{h}'_s\right) \end{aligned} \tag{3}$$

where M is the bi-attention matrix. By normalizing M over different dimension and concatenate them, we get a question-aware context representation, noted as $\tilde{\mathbf{h}}$. The module also push the attention weights into the stack.

Relocate module. It use the result from the previous reasoning step to perform multi-hop reasoning. The Relocate module first pops an attention map from the stack and computes the bridge entity's representation b as the weighted average over context representation $\mathbf{h}$ using the popped attention. It calculates bridge-entity aware context representation $\mathbf{c}_b$ as $\mathbf{h} \odot b$. It then executes Find($\mathbf{u}, \mathbf{h}_b, c_{t-1}$)

Compare module. It pops two context representations att_1 and att_2 from the stack and computes two weighted averages over $\mathbf{h}$ using the attention maps, noted as $\mathbf{h}_{s1}, \mathbf{h}_{s2}$ respectively. It together with the sub-question representation c_t are fed into an MLP to get an memory output m.

NoOP module. It does nothing and can be seen as a skip-command.

3.1.3 Prediction

To predict a span from the context as the answer, the algorithm pops the top question-aware context representation from the stack, apply self-attention and project it down to get the span's start index and end index. To predict yes/no, they take the memory output m and project it down to a 2-way classifier. They concatenate the question vector q and memory m and then project down to a 2-way classifier to decide whether to output a span or yes/no.

3.2 NLProlog: Reasoning with Weak Unification for Question Answering in Natural Language, ACL 2019

NLProlog (Weber et al., 2019) combine neural networks with logic programming for solving multi-hop reasoning tasks over natural language. It uses a prolog prover and utilize sa similarity function over pretrained sentence encoders to allow reasoning with soft rules.

Background. NLProlog consider Prolog programs in the form of horn clauses.

$$h(f_1^h, ..., f_n^h) \Leftarrow p_1(f_1^1, ..., f_m^1) \wedge ... \wedge p_B(f_1^B, ..., f_m^B) \tag{4}$$

Here, h, p_i are termed predicates, f_j^i are functions or variables. $h(f_1^h, ..., f_n^h)$ is the head, and $p_1(f_1^1, ..., f_m^1) \wedge ... \wedge p_B(f_1^B, ..., f_m^B)$ is the body of the rule. B is the body size of the rule, an a rule with body size of 0 is a fact.

Prolog use backward chaining for proving assertions. Given a goal atom g, it first checks whether g is stated in a KB. If not, the algorithm finds subgoals to be proved next. It is achieved by the *unification* operator: given two atoms, it tries to find variable substitutions that make both atoms syntactically equal.

NLProlog. NLProlog follows the pipeline in Prolog on how it proves assertions. It also use unificiation and backward chaining algorithms. However, in NLProlog, all the "matchings" are soft, with the similarity measure modeled with neural networks. It also learns the embedding of rule predicates (in the form of Eq. 4) in an end-to-end manner.

Triplet construction The algorithm first transform the support document to a set of triples. The triplets have the form *(ENTITY 1, surface text, ENTITY 2)*, where the entities are extracted with NER models. For example, given a sentence *Socrates was born in Greece*, it generates a triple *(Socrates, ENT1 was born in ENT2, Greece)*. The triplet notes for an atomic formula *pred(Socrates, Greece)*, where the predicate of the atom is *ENT1 was born in ENT2*.

Rule learning. Now, we have mapped every sentence in the support document to a triplets. These triplets serves as a knowledge base. However, up to now, the model cannot perform multi-hop reasoning, as there are no rules relating these atoms, such as *born_in(X,Z)* $\Leftarrow$ *born_in(X,Y)* $\wedge$ *located_in(Y,Z)*. To learn these rules, NLProlog need a predefined set of rule templates in the form such as $p_1(X, Z) \Leftarrow p_2(X, Y) \wedge p_3(Y, Z)$. The embeddings of the rules are trained as follows.

Training Model Parameters by Backpropgation. The parameters (rule embedding, sentence encoders) in NLProlog are trained with backpropagation using a learning from entailment setting, in which the model should decide whether a Prolog program R entails the truth of a candidate triple (answer) $c \in C$, where C is the set of candidate triples. The model is trained to assign high probabilities $p(c|R; \theta)$ to true candidate triples, and low probabilities to false triples. $p(c|R; \theta)$ is the maximum proof score of all possible proof up to a given depth D. The *proof scores* are given by the aggregation (e.g. product, min) of weak unification scores, which are in turn computed via a similarity function determined by model parameters. Therefore, simply by backpropagation, we may train the model parameters.

References

Yichen Jiang and Mohit Bansal. Self-assembling modular networks for interpretable multi-hop reasoning. *arXiv preprint arXiv:1909.05803*, 2019.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Leon Weber, Pasquale Minervini, Jannes Münchmeyer, Ulf Leser, and Tim Rocktäschel. Nlprolog: Reasoning with weak unification for question answering in natural language. *arXiv preprint arXiv:1906.06187*, 2019.