A Constituent-Centric Neural Architecture for Reading Comprehension

Pengtao Xie[*†] and Eric P. Xing[†]
[*]Machine Learning Department, Carnegie Mellon University
[†]Petuum Inc.
pengtaox@cs.cmu.edu, eric.xing@petuum.com

Abstract

Reading comprehension (RC), aiming to understand natural texts and answer questions therein, is a challenging task. In this paper, we study the RC problem on the Stanford Question Answering Dataset (SQuAD). Observing from the training set that most correct answers are centered around constituents in the parse tree, we design a constituent-centric neural architecture where the generation of candidate answers and their representation learning are both based on constituents and guided by the parse tree. Under this architecture, the search space of candidate answers can be greatly reduced without sacrificing the coverage of correct answers and the syntactic, hierarchical and compositional structure among constituents can be well captured, which contributes to better representation learning of the candidate answers. On SQuAD, our method achieves the state of the art performance and the ablation study corroborates the effectiveness of individual modules.

1 Introduction

Reading comprehension (RC) aims to answer questions by understanding texts, which is a challenge task in natural language processing. Various RC tasks and datasets have been developed, including Machine Comprehension Test (Richardson et al., 2013) for multiple-choice question answering (QA) (Sachan et al., 2015; Wang and McAllester, 2015), Algebra (Hosseini et al., 2014) and Science (Clark and Etzioni, 2016) for passing standardized tests (Clark et al., 2016), CNN/Daily Mail (Hermann et al., 2015) and Children's Book Test (Hill et al., 2015) for cloze-style QA (Chen et al., 2016; Shen et al., 2016), WikiQA (Yang et al., 2015), Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) and Microsoft Machine Reading Comprehension (Nguyen et al., 2016) for open domain QA. In this paper, we are specifically interested in solving the SQuAD QA task (Figure 1 shows an example), in light of its following features: (1) large scale: 107,785 questions, 23,215 paragraphs; (2) non-synthetic: questions are generated by crowdworkers; (3) large search space of candidate answers.

> The most authoritative account at the time came from the medical faculty in Paris in a report to the king of France that blamed the heavens. This report became the first and most widely circulated of a series of plague tracts that sought to give advice to sufferers. That the plague was caused by bad air became the most widely accepted theory. Today, this is known as the Miasma theory.
>
> 1. Who was the medical report written for?
> the king of France
>
> 2. What is the newer, more widely accepted theory behind the spread of the plague?
> bad air
>
> 3. What is the bad air theory officially known as?
> Miasma theory

Figure 1: An example of the SQuAD QA task

We study two major problems: (1) how to generate candidate answers? Unlike in multiple-choice QA and cloze-style QA where a small amount of answer choices are given, an answer in SQuAD could be any span in the text, resulting in a large search space with size $O(n^2)$ (Rajpurkar et al., 2016), where n is the number of words in the sentence. This would incur a lot of noise, ambiguity and uncertainty, making it highly difficult to pick up the correct answer. (2) how to effectively represent the candidate answers? First, long-range semantics spanning multiple sentences need to be captured. As noted in (Rajpurkar et al., 2016), the answering of many questions requires multiple-sentence reasoning. For instance, in Figure 1, the last two sentences in the passages are needed to answer the third question. Second, local syntactic structure needs to be incorporated into representation learning. The study by (Rajpurkar et al., 2016) shows that syntax plays an important role in SQuAD QA: there are a wide range of syntactic divergence between a question and the sentence containing the answer; the answering of 64.1% questions needs to deal with syntactic variation; experiments show that syntactic features are the major contributing factors to good performance.

To tackle the first problem, motivated by the observation in (Rajpurkar et al., 2016) that the correct answers picked up by human are not arbitrary spans, but rather centered around constituents in the parse tree, we generate candidate answers based upon constituents, which significantly reduces the search space. Different from (Rajpurkar et al., 2016) who only consider exact constituents, we adopt a constituent expansion mechanism which greatly improves the coverage of correct answers.

For the representation learning of candidate answers which are sequences of constituents, we first encode individual constituents using a *chain-of-trees LSTM* (CT-LSTM) and tree-guided attention mechanism, then feed these encodings into a chain LSTM (Hochreiter and Schmidhuber, 1997) to generate representations for the constituent sequences. The CT-LSTM seamlessly integrates intra-sentence tree LSTMs (Tai et al., 2015) which capture the local syntactic properties of constituents and an inter-sentence chain LSTM which glues together the sequence of tree LSTMs such that the semantics of each sentence can be propagated to others. The tree-guided attention leverages the hierarchical relations among constituents to learn question-aware representations.

Putting these pieces together, we design a constituent-centric neural network (CCNN), which contains four layers: a chain-of-trees LSTM encoding layer, a tree-guided attention layer and a candidate-answer generation layer, a prediction layer. Evaluation on SQuAD demonstrates the effectiveness of CCNN.

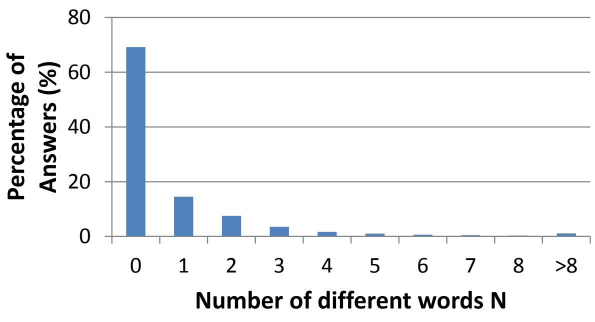
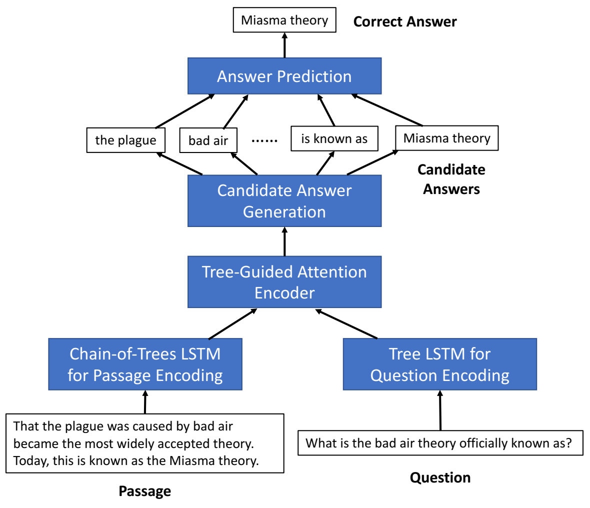


Figure 2: Percentage of answers that differ from their closest constituents by N words



Miasma theory
Correct Answer
Answer Prediction
the plague
bad air
......
is known as
Miasma theory
Candidate Answers
Candidate Answer Generation
Tree-Guided Attention Encoder
Chain-of-Trees LSTM for Passage Encoding
Tree LSTM for Question Encoding
That the plague was caused by bad air became the most widely accepted theory. Today, this is known as the Miasma theory.
Passage
What is the bad air theory officially known as?
Question

Figure 3: Constituent-centric neural network.

2 Constituent-Centric Neural Network for Reading Comprehension

2.1 Overall Architecture

As observed in (Rajpurkar et al., 2016), almost all correct answers are centered around the constituents. To formally confirm this, we compare the correct answers in the training set with constituents generated by the Stanford parser (Manning et al., 2014): for each correct answer, we find its "closest" constituent – the longest constituent that is a substring of the answer, and count how many words they differ from (let N denote this number). Figure 2 shows the percentage of answers whose N equals to $0, \cdots, 8$ and $N > 8$. As can be seen, $\sim$70% answers are exactly constituents ($N = 0$) and $\sim$97% answers differ from the closest constituents by less equal to 4 words. This observation motivates us to approach the reading comprehension problem in a constituent-centric manner, where the generation of candidate answers and their representation learning are both based upon constituents.

Specifically, we design a Constituent-Centric Neural Network (CCNN) to perform end-to-end reading comprehension, where the inputs are the passage and question, and the output is a span in the passage that is mostly suitable to answer this question. As shown in Figure 3, the CCNN contains four layers. In the *encoding* layer, the chain-of-trees LSTM and tree LSTM encode the constituents in the passage and question respectively. The encodings are fed to the tree-guided *attention layer* to learn question-aware representations, which are passed to the *candidate-answer generation layer* to produce and encode the candidate answers based on constituent expansion. Finally, the *prediction layer* picks up the best answer from the candidates using a feed-forward network.

2.2 Encoding

Given the passages and questions, we first use the Stanford parser to parse them into constituent parse trees, then the encoding layer of CCNN learns representations for constituents in questions and passages, using tree LSTM (Tai et al., 2015) and chain-of-trees LSTM respectively. These LSTM encoders are able to capture the syntactic properties of constituents and long-range semantics across multiple sentences, which are crucial for SQuAD QA.

2.2.1 Tree LSTM for Question Encoding

Each question is a single sentence, having one constituent parse tree. Internal nodes in the tree represent constituents having more than one word and leaf nodes represent single-word constituent. Inspired by (Tai et al., 2015; Teng and Zhang, 2016), we build a bi-directional tree LSTM which consists of a bottom-up LSTM and a top-down LSTM, to encode these constituents (as shown in Figure 4). Each node (constituent) has two hidden states: $\mathbf{h}_\uparrow$ produced by the LSTM in bottom-up direction and $\mathbf{h}_\downarrow$ produced by the LSTM in top-down direction. Let T denote the maximum number of children an internal node could have. For each particular node, let L ($0 \leq L \leq T$) be the number of children it has, $\mathbf{h}_\uparrow^{(l)}$ and $\mathbf{c}_\uparrow^{(l)}$ be the bottom-up hidden state and memory cell of the l-th ($1 \leq l \leq L$) child (if any) respectively and $\mathbf{h}_\downarrow^{(p)}$

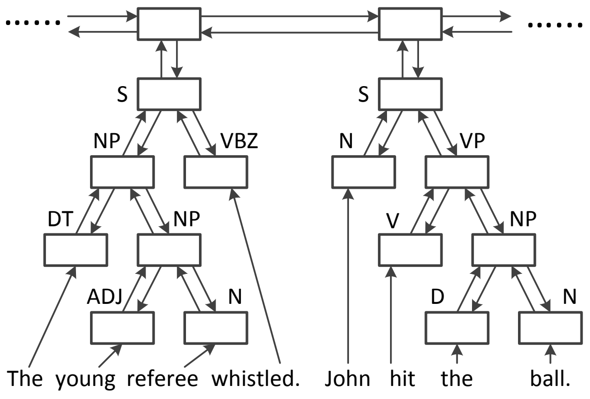


Figure 4: Chain-of-trees LSTM

and $\mathbf{c}_\downarrow^{(p)}$ be the top-down hidden state and memory cell of the parent.

In the bottom-up LSTM, each node has an input gate $\mathbf{i}_\uparrow$, L forget gates $\{\mathbf{f}_\uparrow^{(l)}\}_{l=1}^{L}$ corresponding to different children, an output gate $\mathbf{o}_\uparrow$ and a memory cell $\mathbf{c}_\uparrow$. For an internal node, the inputs are the hidden states and memory cells of its children and the transition equations are defined as:

$$\begin{aligned}
&\mathbf{i}_\uparrow = \sigma(\textstyle\sum_{l=1}^{L} \mathbf{W}_\uparrow^{(i,l)}\mathbf{h}_\uparrow^{(l)} + \mathbf{b}_\uparrow^{(i)}) \\
&\forall l, \mathbf{f}_\uparrow^{(l)} = \sigma(\mathbf{W}_\uparrow^{(f,l)}\mathbf{h}_\uparrow^{(l)} + \mathbf{b}_\uparrow^{(f,l)}) \\
&\mathbf{o}_\uparrow = \sigma(\textstyle\sum_{l=1}^{L} \mathbf{W}_\uparrow^{(o,l)}\mathbf{h}_\uparrow^{(l)} + \mathbf{b}_\uparrow^{(o)}) \\
&\mathbf{u}_\uparrow = \tanh(\textstyle\sum_{l=1}^{L} \mathbf{W}_\uparrow^{(u,l)}\mathbf{h}_\uparrow^{(l)} + \mathbf{b}_\uparrow^{(u)}) \\
&\mathbf{c}_\uparrow = \mathbf{i}_\uparrow \odot \mathbf{u}_\uparrow + \textstyle\sum_{l=1}^{L} \mathbf{f}_\uparrow^{(l)} \odot \mathbf{c}_\uparrow^{(l)} \\
&\mathbf{h}_\uparrow = \mathbf{o}_\uparrow \odot \tanh(\mathbf{c}_\uparrow)
\end{aligned} \tag{1}$$

where the weight parameters $\mathbf{W}$ and bias parameters $\mathbf{b}$ with superscript l such as $\mathbf{W}_\uparrow^{(i,l)}$ are specific to the l-th child. For a leaf node which represents a single word, it has no forget gate and the input is the wording embedding (Pennington et al., 2014) of this word.

In the top-down direction, the gates, memory cell and hidden state are defined in a similar fashion as the bottom-up direction (Eq.(1)). For an internal node except the root, the inputs are the hidden state $\mathbf{h}_\downarrow^{(p)}$ and memory cell $\mathbf{c}_\downarrow^{(p)}$ of its parents. For a leaf node, in addition to $\mathbf{h}_\downarrow^{(p)}$ and $\mathbf{c}_\downarrow^{(p)}$, the inputs also contain the word embedding. For the root node, the top-down hidden state $\mathbf{h}_\downarrow^{(r)}$ is set to its bottom-up hidden state $\mathbf{h}_\uparrow^{(r)}$. $\mathbf{h}_\uparrow^{(r)}$ captures the semantics of all constituents, which is then replicated as $\mathbf{h}_\downarrow^{r}$ and propagated downwards to each individual constituent.

Concatenating the hidden states of two directions, we obtain the LSTM encoding for each node

$\mathbf{h} = [\mathbf{h}_{\uparrow}; \mathbf{h}_{\downarrow}]$ which will be the input of the attention layer. The bottom-up hidden state $\mathbf{h}_{\uparrow}$ composes the semantics of sub-constituents contained in this constituent and the top-down hidden state $\mathbf{h}_{\downarrow}$ captures the contextual semantics manifested in the entire sentence.

2.2.2 Chain-of-Trees LSTM for Passage Encoding

To encode the passage which contains multiple sentences, we design a chain-of-trees LSTM (Figure 4). A bi-directional tree LSTM is built for each sentence to capture the local syntactic structure and these tree LSTMs are glued together via a bi-directional chain LSTM (Graves et al., 2013) to capture long-range semantics spanning multiple sentences. The hidden states generated by the bottom-up tree LSTM serves as the input of the chain LSTM. Likewise, the chain LSTM states are fed to the top-down tree LSTM. This enables the encoding of every constituent to be propagated to all other constituents in the passage.

In the chain LSTM, each sentence t is treated as a unit. The input of this unit is generated by the tree LSTM of sentence t, which is the bottom-up hidden state $\mathbf{h}_{\uparrow t}$ at the root. Sentence t is associated with a forward hidden state $\overrightarrow{\mathbf{h}}_t$ and a backward state $\overleftarrow{\mathbf{h}}_t$. In the forward direction, the transition equations among the input gate $\overrightarrow{\mathbf{i}}_t$, forget gate $\overrightarrow{\mathbf{f}}_t$, output gate $\overrightarrow{\mathbf{o}}_t$ and memory cell $\overrightarrow{\mathbf{c}}_t$ are:

$$\begin{aligned}
\overrightarrow{\mathbf{i}}_t &= \sigma(\overrightarrow{\mathbf{W}}^{(i)}\mathbf{h}_{\uparrow t} + \overrightarrow{\mathbf{U}}^{(i)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}^{(i)}) \\
\overrightarrow{\mathbf{f}}_t &= \sigma(\overrightarrow{\mathbf{W}}^{(f)}\mathbf{h}_{\uparrow t} + \overrightarrow{\mathbf{U}}^{(f)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}^{(f)}) \\
\overrightarrow{\mathbf{o}}_t &= \sigma(\overrightarrow{\mathbf{W}}^{(o)}\mathbf{h}_{\uparrow t} + \overrightarrow{\mathbf{U}}^{(o)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}^{(o)}) \\
\overrightarrow{\mathbf{u}}_t &= \tanh(\overrightarrow{\mathbf{W}}^{(u)}\mathbf{h}_{\uparrow t} + \overrightarrow{\mathbf{U}}^{(u)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{b}}^{(u)}) \\
\overrightarrow{\mathbf{c}}_t &= \overrightarrow{\mathbf{i}}_t \odot \overrightarrow{\mathbf{u}}_t + \overrightarrow{\mathbf{f}}_t \odot \overrightarrow{\mathbf{c}}_{t-1} \\
\overrightarrow{\mathbf{h}}_t &= \overrightarrow{\mathbf{o}}_t \odot \tanh(\overrightarrow{\mathbf{c}}_t)
\end{aligned} \tag{2}$$

The backward LSTM is defined in a similar way. Subsequently, $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$, which encapsulate the semantics of all sentences, are inputted to the root of the top-down tree LSTM and propagated to all the constituents in sentence t.

To sum up, the CT-LSTM encodes a passage in the following way: (1) the bottom-up tree LSTMs compute hidden states $\mathbf{h}_{\uparrow}$ for each sentence and feed $\mathbf{h}_{\uparrow}$ of the root node into the chain LSTM; (2) the chain LSTM computes forward and backward states and feed them into the root of the top-down tree LSTMs; (3) the top-down tree LSTMs compute hidden states $\mathbf{h}_{\downarrow}$. At each constituent C, the bottom-up state $\mathbf{h}_{\uparrow}$ captures the semantics of sub-constituents in C and the top-down state $\mathbf{h}_{\downarrow}$ captures the semantics of the entire passage.

2.3 Tree-Guided Attention Mechanism

We propose a tree-guided attention (TGA) mechanism to learn a question-aware representation for each constituent in the passage, which consists of three ingredients: (1) constituent-level attention score computation; (2) tree-guided local normalization; (3) tree-guided attentional summarization. Given a constituent $\mathbf{h}^{(p)}$ in the passage, for each constituent $\mathbf{h}^{(q)}$ in the question, an unnormalized attention weight score a is computed as $a = \mathbf{h}^{(p)} \cdot \mathbf{h}^{(q)}$ which measures the similarity between the two constituents. Then we perform a tree-guided local normalization of these scores. At each internal node in the parse tree, where the unnormalized attention scores of its L children are $\{a_l\}_{l=1}^{L}$, a local normalization is performed using a softmax operation $\widetilde{a}_l = \exp(a_l)/\sum_{m=1}^{L}\exp(a_m)$ which maps these scores into a probabilistic simplex. This normalization scheme stands in contrast with the global normalization adopted in word-based attention (Wang and Jiang, 2016; Wang et al., 2016), where a single softmax is globally applied to the attention scores of all the words in the question.

Given these locally normalized attention scores, we merge the LSTM encodings of constituents in the question into an attentional representation in a recursive and bottom-up way. At each internal node, let $\mathbf{h}$ be its LSTM encoding, a and $\{a_l\}_{l=1}^{L}$ be the normalized attention scores of this node and its L children, and $\{\mathbf{b}_l\}_{l=1}^{L}$ be the attentional representations (which we will define later) generated at the children, then the attentional representation $\mathbf{b}$ of this node is defined as:

$$\mathbf{b} = a(\mathbf{h} + \sum_{l=1}^{L} a_l \mathbf{b}_l) \tag{3}$$

which takes the weighted representation $\sum_{l=1}^{L} a_l \mathbf{b}_l$ contributed from its children, adds in its own encoding $\mathbf{h}$, then performs a re-weighting using the attention score a. The attentional representation $\mathbf{b}^{(r)}$ at the root node acts as the final summarization of constituents in the question. We concatenate it to the LSTM encoding $\mathbf{h}^{(p)}$ of the passage constituent and obtain a *concatenated representation* $\mathbf{z} = [\mathbf{h}^{(p)}; \mathbf{b}^{(r)}]$ which will be the input of the candidate answer generation layer.

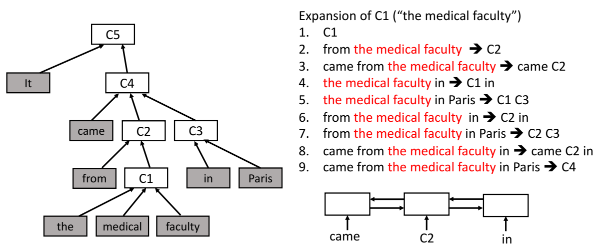

C5
It
C4
came
C2
C3
from
C1
in
Paris
the
medical
faculty
Expansion of C1 ("the medical faculty")
1. C1
2. from the medical faculty ➔ C2
3. came from the medical faculty ➔ came C2
4. the medical faculty in ➔ C1 in
5. the medical faculty in Paris ➔ C1 C3
6. from the medical faculty in ➔ C2 in
7. from the medical faculty in Paris ➔ C2 C3
8. came from the medical faculty in ➔ came C2 in
9. came from the medical faculty in Paris ➔ C4
came
C2
in

Figure 5: Constituent expansion. (Left) Parse tree of a sentence in the passage. (Top Right) Expansions of constituent C1 and their reductions (denoted by arrow). (Bottom Right) Learning the representation of an expansion using bidirectional chain-LSTM.

Unlike the word-based flat-structure attention mechanism (Wang and Jiang, 2016; Wang et al., 2016) where the attention scores are computed between words and normalized using a single global softmax, and the attentional summary is computed in a flat manner, the tree-guided attention calculates attention scores between constituents, normalizes them locally at each node in the parse tree and computes the attentional summary in a hierarchical way. Tailored to the parse tree, TGA is able to capture the syntactic, hierarchical and compositional structures among constituents and arguably generate better attentional representations, as we will validate in the experiments.

2.4 Candidate Answer Generation

As shown in Figure 2, while most correct answers in the training set are exactly constituents, some of them are not the case. To cover the non-constituent answers, we propose to expand each constituent by appending words adjacent to it. Let C denote a constituent and $S =$ "$\cdots w_{i-1} w_i C w_j w_{j+1} \cdots$" be the sentence containing C. We expand C by appending words preceding C (such as w_{i-1} and w_i) and words succeeding C (such as w_j and w_{j+1}) to C. We define an (l, r)-expansion of a constituent C as follows: append l words preceding C in the sentence to C; append r words succeeding C to C. Let M be the maximum expansion number that $l \leq M$ and $r \leq M$. Figure 5 shows an example. On the left is the constituent parse tree of the sentence "it came from the medical faculty in Paris". On the upper right are the expansions of the constituent C1 – "the medical faculty". To expand this constituent, we trace it back to the sentence and look up the M (M=2 in this case) words preceding C1 (which are "came" and "from") and succeeding C1 (which are "in" and "Paris"). Then combinations of C1 and the preceding/succeeding words are taken to generate constituent expansions. On both the left and right side of C1, we have three choices of expansion: expanding 0,1,2 words. Taking combination of these cases, we obtain 9 expansions, including C1 itself ($(0, 0)$-expansion).

The next step is to perform reduction of constituent expansions. Two things need to be reduced. First, while expanding the current constituent, new constituents may come into being. For instance, in the expansion "came from C1 in Paris", "in" and "Paris" form a constituent C3; "from" and C1 form a constituent C2; "came", C2 and C3 form a constituent C4. Eventually, this expansion is reduced to C4. Second, the expansions generated from different constituents may have overlap and the duplicated expansions need to be removed. For example, the $(2, 1)$-expansion of C1 – "came from the medical faculty in" – can be reduced to "came C2 in", which is the $(1, 1)$-expansion of C2. After reduction, each expansion is a sequence of constituents.

Next we encode these candidate answers and the encodings will be utilized in the prediction layer. In light of the fact that each expansion is a constituent sequence, we build a bi-directional chain LSTM (Figure 5, bottom right) to synthesize the representations of individual constituents therein. Let $E = C_1 \cdots C_n$ be an expansion consisting of n constituents. In the chain LSTM, the input of unit i is the combined representation of C_i. We concatenate the forward hidden state at C_n and backward state at C_1 as the final representation of E.

2.5 Answer Prediction and Parameter Learning

Given the representation of candidate answers, we use a feed-forward network $f : \mathbb{R}^d \to \mathbb{R}$ to predict the correct answer. The input of the network is the feature vector of a candidate answer and the output is a confidence score. The one with the largest score is chosen as the the correct answer.

For parameter learning, we normalize the confidence scores into a probabilistic simplex using softmax and define a cross entropy loss thereupon. Let J_k be the number of candidate answers produced from the k-th passage-question pair and

$\{\mathbf{z}_j^{(k)}\}_{j=1}^{J_k}$ be their representations. Let t_k be the index of the correct answer. Then the cross entropy loss of K pairs is defined as

$$\sum_{k=1}^{K}(-f(\mathbf{z}_{t_k}) + \log \sum_{j=1}^{J_k} \exp(f(\mathbf{z}_j^{(k)}))) \quad (4)$$

Model parameters are learned by minimizing this loss using stochastic gradient descent.

3 Experiments

3.1 Experimental Setup

The experiments are conducted on the Stanford Question Answering Dataset (SQuAD) v1.1, which contains 107,785 questions and 23,215 passages coming from 536 Wikipedia articles. The data was randomly partitioned into a training set (80%), a development set (10%) and an unreleased test set (10%). Rajpurkar et al. (2016) build a leaderboard to evaluate and publish results on the test set. Due to software copyright issues, we did not participate this online evaluation. Instead, we use the development set (which is untouched during model training) as test set. In training, if the correct answer is not in the candidate-answer set, we use the shortest candidate containing the correct answer as the target.

The Stanford parser is utilized to obtain the constituent parse trees for questions and passages. In the parse tree, any internal node which has one child is merged together with its child. For instance, in "(NP (NNS sufferers))", the parent "NP" has only one child "(NNS sufferers)", we merge them into "(NP sufferers)". We use 300-dimensional word embeddings from GloVe (Pennington et al., 2014) to initialize the model. Words not found in GloVe are initialized as zero vectors.

We use a feed-forward network with 2 hidden layers (both having the same amount of units) for answer prediction. The activation function is set to rectified linear. Hyperparameters in CCNN are tuned via 5-fold cross validation (CV) on the training set, summarized in Table 1. We use the ADAM (Kingma and Ba, 2014) optimizer to train the model with an initial learning rate 0.001 and a mini-batch size 100. An ensemble model is also trained, consisting of 10 training runs using the same hyperparameters. The performance is evaluated by two metrics (Rajpurkar et al., 2016): (1) exact match (EM) which measures the percentage of predictions that match any one of the ground truth answers exactly; (2) F1 score which measures the average overlap between the prediction and ground truth answer. In the development set each question has about three ground truth answers. F1 scores with the best matching answers are used to compute the average F1 score.

3.2 Results

Table 2 shows the performance of our model and previous approaches on the development set. CCNN (single model) achieves an EM score of 69.3% and an F1 score of 78.5%, significantly outperforming all previous approaches (single model). Through ensembling, the performance of CCNN is further improved and outperforms the baseline ensemble methods. The key difference between our method and previous approaches is that CCNN is constituent-centric where the generation and encoding of candidate answers are both based on constituents while the baseline approaches are mostly word-based where the candidate answer is an arbitrary span of words and the encoding is performed over individual words rather than at the constituent level. The constituent-centric model-design enjoys two major benefits. First, restricting the candidate answers from arbitrary spans to neighborhoods around the constituents greatly reduces the search space, which mitigates the ambiguity and uncertainty in picking up the correct answer. Second, the tree LSTMs and tree-guided attention mechanism encapsulate the syntactic, hierarchical and compositional structure among constituents, which leads to better representation learning of the candidate answers. We conjecture these are the primary reasons that CCNN outperforms the baselines and provide a validation in the next section.

3.3 Ablation Study

To further understand the individual modules in CCNN, we perform an ablation study. The results are shown in Table 2.

Tree LSTM To evaluate the effectiveness of tree LSTM in learning syntax-aware representations, we replace it with a syntax-agnostic chain LSTM. We build a bi-directional chain LSTM (denoted by A) over the entire passage to encode the individual words. Given a constituent $C = w_i \cdots w_j$, we build another bi-directional chain LSTM (denoted by B) over C where the inputs are the encodings of words $w_i, \cdots, w_j$ generated by LSTM

Parameter	Tuning Range	Best Choice
Maximum expansion number M in constituent expansion	$0, 1, 2, 3, 4, 5$	2
Size of hidden state in all LSTMs	$50, 100, 150, 200, 250, 300$	100
Size of hidden state in prediction network	100, 200, 300, 400, 500	400

Table 1: Hyperparameter Tuning

	Exact Match (EM,%)	F1 (%)
Single model		
Logistic Regression (Rajpurkar et al., 2016)	40.0	51.0
Fine Grained Gating (Yang et al., 2016)	60.0	71.3
Dynamic Chunk Reader (Yu et al., 2016)	62.5	71.2
Match-LSTM with Answer Pointer (Wang and Jiang, 2016)	64.1	73.9
Dynamic Coattentation Network (Xiong et al., 2016)	65.4	75.6
Multi-Perspective Context Matching (Wang et al., 2016)	66.1	75.8
Recurrent Span Representations (Lee et al., 2016)	66.4	74.9
Bi-Directional Attention Flow (Seo et al., 2016)	68.0	77.3
Ensemble		
Fine Grained Gating (Yang et al., 2016)	62.4	73.4
Match-LSTM with Answer Pointer (Wang and Jiang, 2016)	67.6	76.8
Recurrent Span Representations (Lee et al., 2016)	68.2	76.7
Multi-Perspective Context Matching (Wang et al., 2016)	69.4	78.6
Dynamic Coattentation Network (Xiong et al., 2016)	70.3	79.4
Bi-Directional Attention Flow (Seo et al., 2016)	73.3	81.1
CCNN Ablation (single model)		
Replacing tree LSTM with chain LSTM	63.5	73.9
Replacing chain-of-trees LSTM with independent tree LSTMs	64.8	75.2
Removing the attention layer	63.9	74.3
Replacing tree-guided attention with flat attention	65.6	75.9
CCNN (single model)	**69.3**	**78.5**
CCNN (ensemble)	**74.1**	**82.6**

Table 2: Results on the development set

A. In LSTM B, the forward hidden state of w_j and backward state of w_i are concatenated to represent C. Note that the attention mechanism remains intact, which is still guided by the parse tree. This replacement cause 5.8% and 4.6% drop of the EM and F1 scores respectively, which demonstrates the necessity of incorporating syntactic structure (via tree LSTM) into representation learning.

Chain-of-Trees LSTM (CT-LSTM) We evaluate the effectiveness of CT-LSTM by comparing it with a bag of tree LSTMs: instead of using a chain LSTM to glue the tree LSTMs, we treat them as independent. Keeping the other modules intact and replacing CT-LSTM with a bag of independent tree LSTMs, the EM and F1 score drop 4.5% and 3.3% respectively. The advantage of CT-LSTM is that it enables the semantics of one sentence to be propagated to others, which makes multiple-sentence reasoning possible.

Tree-Guided Attention (TGA) Mechanism To evaluate the effectiveness of TGA, we performed two studies. First, we take it off from the architecture. Then constituents in the passage are solely represented by the chain-of-trees LSTM encodings and the question sentence is represented by the tree LSTM encoding at the root of the parse tree. At test time, we concatenate the encodings of a candidate answer and the question as inputs of the prediction network. Removing the attention layer decreases the EM and F1 by 5.4% and 4.2% respectively, demonstrating the effectiveness of attention mechanism for question-aware representation learning.

Second, we compare the tree-structured mech-

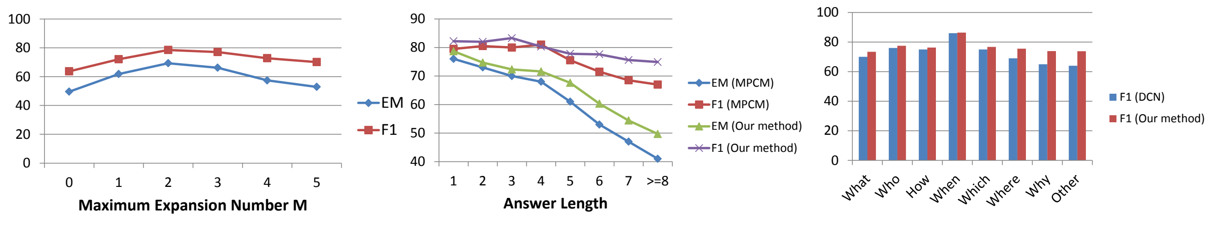


Figure 6: Performance for different (a) M (expansion number), (b) answer length, (c) question type.

anism in TGA with a flat-structure mechanism. For each constituent $\mathbf{h}_i^{(p)}$ in the passage, we compute its unnormalized score $a_{ij} = \mathbf{h}_i^{(p)} \cdot \mathbf{h}_j^{(q)}$ with every constituent $\mathbf{h}_j^{(q)}$ in the question (which has R constituents). Then a global softmax operation is applied to these scores, $\{\tilde{a}_{ij}\}_{j=1}^{R} = softmax(\{a_{ij}\}_{j=1}^{R})$, to project them into a probabilistic simplex. Finally, a flat summarization $\sum_{j=1}^{R} \tilde{a}_{ij}\mathbf{h}_j^{(q)}$ is computed and appended to $\mathbf{h}_i^{(p)}$. Replacing TGA with flat-structure attention causes the EM and F1 to drop 3.7% and 2.6% respectively, which demonstrates the advantage of the tree-guided mechanism.

Constituent Expansion We study how the maximum expansion number M affects performance. If M is too small, many correct answers are not contained in the candidate set, which results in low recall. If M is too large, excessive candidates are generated, making it harder to pick up the correct one. Figure 6(a) shows how EM and F1 vary as M increases, from which we can see a value of M in the middle ground achieves the best tradeoff.

3.4 Analysis

In this section, we study how CCNN behaves across different answer length (number of words in the answer) and question types, which are shown in Figure 6(b) and (c). In Figure 6(b), we compare with the MPCM method (Wang et al., 2016). As answer length increases, the performance of both methods decreases. This is because for longer answers, it is more difficult to pinpoint the precise boundaries. The decreasing of F1 is slower than EM, because F1 is more elastic to small mismatches. Our method achieves larger improvement over MPCM at longer answers. We conjecture the reason is: longer answers have more complicated syntactic structure, which can be better captured by the tree LSTMs and tree-guided attention mechanism in our method. MPCM is built upon individual words and is syntax-agnostic.

In Figure 6(c), we compare with DCN (Xiong et al., 2016) on 8 question types. Our method achieves significant improvement over DCN on four types: “what”, “where”, “why” and “other”. The answers of questions in these types are typically longer and have more complicated syntactic structure than the other four types where the answers are mostly entities (person, numeric, time, etc.). The syntax-aware nature of our method makes it outperform DCN whose model design does not explicitly consider syntactic structures.

4 Related Works

Several neural network based approaches have been proposed to solve the SQuAD QA problem, which we briefly review from three aspects: candidate answer generation, representation learning and attention mechanism.

Two ways were investigated for candidate answer generation: (1) chunking: candidates are preselected based on lexical and syntactic analysis, such as constituent parsing (Rajpurkar et al., 2016) and part-of-speech pattern (Yu et al., 2016); (2) directly predicting the start and end position of the answer span, using feed-forward neural network (Wang et al., 2016), LSTM (Seo et al., 2016), pointer network (Vinyals et al., 2015; Wang and Jiang, 2016), dynamic pointer decoder (Xiong et al., 2016).

The representation learning in previous approaches is conducted over individual words using the following encoders: LSTM in (Wang et al., 2016; Xiong et al., 2016); bi-directional gated recurrent unit (Chung et al., 2014) in (Yu et al., 2016); match-LSTM in (Wang and Jiang, 2016); bi-directional LSTM in (Seo et al., 2016).

In previous approaches, the attention (Bahdanau et al., 2014; Xu et al., 2015) mechanism is mostly word-based and flat-structured (Kadlec et al., 2016; Sordoni et al., 2016; Wang and Jiang, 2016; Wang et al., 2016; Yu et al., 2016): the attention scores are computed between individual words, are normalized globally and are used to summarize word-level encodings in a flat manner. Cui et al. (2016); Xiong et al. (2016) explored a coattention mechanism to learn question-to-passage and passage-to-question summaries. Seo et al. (2016) proposed to directly use the attention weights as augmented features instead of applying them for early summarization.

5 Conclusions and Future Work

To solve the SQuAD question answering problem, we design a constituent centric neural network (CCNN), where the generation and representation learning of candidate answers are both based on constituents. We use a constituent expansion mechanism to produce candidate answers, which can greatly reduce the search space without losing the recall of hitting the correct answer. To represent these candidate answers, we propose a chain-of-trees LSTM to encode constituents and a tree-guided attention mechanism to learn question-aware representations. Evaluations on the SQuAD dataset demonstrate the effectiveness of the constituent-centric neural architecture.

For future work, we will investigate the wider applicability of chain-of-trees LSTM as a general text encoder that can simultaneously capture local syntactic structure and long-range semantic dependency. It can be applied to named entity recognition, sentiment analysis, dialogue generation, to name a few. We will also apply the tree-guided attention mechanism to NLP tasks that need syntax-aware attention, such as machine translation, sentence summarization, textual entailment, etc. Another direction to explore is joint learning of syntactic parser and chain-of-trees LSTM. Currently, the two are separated, which may lead to suboptimal performance.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858* .

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* .

Peter Clark and Oren Etzioni. 2016. My computer is an honor student-but how intelligent is it? standardized tests as a measure of ai. *AI Magazine* 37(1):5–12.

Peter Clark, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter D Turney, and Daniel Khashabi. 2016. Combining retrieval, statistics, and inference to answer elementary science questions. In *AAAI*. pages 2580–2586.

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* .

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, pages 273–278.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*. pages 1693–1701.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2015. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301* .

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

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*. pages 523–533.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. 2016. Text understanding with the attention sum reader network. *arXiv preprint arXiv:1603.01547* .

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

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

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268* .

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* .

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*. volume 3, page 4.

Mrinmaya Sachan, Kumar Dubey, Eric P Xing, and Matthew Richardson. 2015. Learning answer-entailing structures for machine comprehension. In *ACL (1)*. pages 239–249.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603* .

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

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

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075* .

Zhiyang Teng and Yue Zhang. 2016. Bidirectional tree-structured lstm with head lexicalization. *arXiv preprint arXiv:1611.06788* .

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. pages 2692–2700.

Hai Wang and Mohit Bansal Kevin Gimpel David McAllester. 2015. Machine comprehension with syntax, frames, and semantics .

Shuohang Wang and Jing Jiang. 2016. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905* .

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. 2016. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211* .

Caiming Xiong, Victor Zhong, and Richard Socher. 2016. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604* .

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP*. Citeseer, pages 2013–2018.

Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. 2016. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724* .

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