Answer Extraction by Recursive Parse Tree Descent

Christopher Malon
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540
malon@nec-labs.com

Bing Bai
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540
bbai@nec-labs.com

Abstract

We develop a recursive neural network (RNN) to extract answers to arbitrary natural language questions from supporting sentences, by training on a crowdsourced data set (to be released upon presentation). The RNN defines feature representations at every node of the parse trees of questions and supporting sentences, when applied recursively, starting with token vectors from a neural probabilistic language model. In contrast to prior work, we fix neither the types of the questions nor the forms of the answers; the system classifies tokens to match a substring chosen by the question's author.

Our classifier decides to follow each parse tree node of a support sentence or not, by classifying its RNN embedding together with those of its siblings and the root node of the question, until reaching the tokens it selects as the answer. A novel co-training task for the RNN, on *subtree recognition*, boosts performance, along with a scheme to consistently handle words that are not well-represented in the language model. On our data set, we surpass an open source system epitomizing a classic "pattern bootstrapping" approach to question answering.

1 Introduction

The goal of this paper is to learn the syntax used to answer arbitrary natural language questions. If the kinds of questions were fixed but the supporting sentences were open, this would be a kind of relation extraction or slot-filling. If the questions were open but the supporting information was encoded in a database, this would be a kind of semantic parsing.

In spite of many evaluation sets, no suitable data set for *learning* to answer questions has existed before. Data sets such as TREC (Dang et al., 2008) do not identify supporting sentences or even answers unless a competing system submitted an answer and a human verified it. Exceeding the capabilities of current systems is difficult by training on such labels; any newly discovered answer is penalized as wrong. The Jeopardy Archive (Schmidt, 2013) offers more than 200,000 answer/question pairs, but no pointers to information that supports the solutions.

Believing that it is impossible to *learn* to answer questions, QA systems in TREC tended to *measure* syntactic similarity between question and candidate answer, or to map the question into an enumerated set of possible question types. For the pre-determined question types, learning could be achieved, not from the QA data itself, but from pattern bootstrapping (Brin, 1998) or distant supervision against an ontology like Freebase (Mintz et al., 2009). These techniques lose precision; Riedel et al. (2010) found the distant supervision assumption was violated on 31% of examples aligning Freebase relations to text from The New York Times.

We introduce a new, crowdsourced dataset, *TurkQA*, to enable question answering to be learned. TurkQA consists of single sentences, each with several crowdsourced questions. The answer to each question is given as a substring of the supporting sentence. For example,

> James Hervey (February 26, 1714 - December 25, 1758), English divine, was born at Hardingstone, near Northampton, and was educated at the grammar school of Northampton, and at Lincoln College, Oxford.

could have questions like "Where did James Hervey attend school as a boy?" with answers like "the grammar school of Northampton." Our approach has yielded almost 40,000 such questions, and easily scales to many more. Since the sentence containing the answer has already been located, the machine's output can be judged without worrying about missing labels elsewhere in the corpus. Token-level ground truth forces the classifier to isolate the relevant information.

To meet this challenge, we develop a classifier that recursively classifies nodes of the parse tree of a supporting sentence. The positively classified nodes are followed down the tree, and any positively classified terminal nodes become the tokens in the answer. Feature representations are dense vectors in a continuous feature space; for the terminal nodes, they are the word vectors in a neural probabilistic language model (like (Bengio and Ducharme, 2001)), and for interior nodes, they are derived from children by recursive application of an autoencoder.

Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, pages 110–118,
Sofia, Bulgaria, August 9 2013.

The contributions of this paper are: a data set for learning to answer free-form questions; a top-down supervised method using continuous word features in parse trees to find the answer; and a co-training task for training a recursive neural network that preserves deep structural information.

2 Related Work

Most submissions to TREC, TAC, and CLEF (Forner et al., 2008) QA workshops rely on a large pipeline of modules, emphasizing feature development, pattern bootstrapping, and distant supervision. However, some recent work has introduced new learning techniques for question answering.

Restricting the forms of the questions, Poon and Domingos (2009) present a question-answering system for simple questions about biomedical text, by *unsupervised semantic parsing* (USP), using Markov logic. Because of its dependence on semantic role labeling (SRL), USP can only extract limited kinds of information from a sentence's syntax. Particularly, USP has been programmed to use information from just five dependencies: `NN`, `AMOD`, `PREP_OF`, `NUM`, and `APPOS`. The system handles only questions of two forms: "What VERB OBJ?" and "What does SUBJ VERB?"

Liang et al. (2011) offers a compositional focus on semantic parsing. He has implemented a question-answering system for geography and job databases, learning to transform natural language questions into SQL queries. Liang is able to take many words as verbatim analogues of table columns (*e.g.* "city" triggers a search on a *city* column), but our task requires learning such associations in natural language ("city" to a *place* named entity), and less attention to Boolean compositional semantics.

We have not seen recursive neural networks (RNN) applied to QA yet, but Socher has developed applications to paraphrase (Socher et al., 2011a) and sentiment analysis (Socher et al., 2011b). Relying either on dynamic pooling or the root feature alone, these methods do not use the full information of the input graphs.

3 TurkQA: a scalable, crowdsourced data set

The TurkQA data set consists of 13,424 problem sets. Each problem set consists of the first sentence of a Wikipedia article, which we call a *support sentence*, and four questions, written by workers from Amazon Mechanical Turk.[1] (Occasionally, due to a faulty heuristic, two or three consecutive sentences at the beginning of the article are taken.) Each of the four questions is answered by a phrase in the support sentence, or yes/no. At least two short answer questions must exist in each problem set, and their answers are selected by their authors as contiguous, non-overlapping substrings of the support sentence.

Over 600 workers contributed. The quality of the questions was ensured by rigorous constraints on the input: no pronouns could be used; all words from the question had to be in the dictionary or the support sentence; the same phrase could not be used as the answer for multiple questions. We requested that anyone who understood English should be able to understand the question just by reading the support sentence, without any background knowledge. As we took the support sentences from the start of an article, references to prior text should not occur.

At first we reviewed submissions by hand, but as we found that 96% of the problem sets were acceptable (and a higher percentage of the questions), we approved most submissions automatically. Thus we expect the data acquisition technique to be scalable with a budget.

A possible drawback of our data acquisition is so-called *back-formulation*: a tendency of question writers to closely match the syntax of the supporting sentence when writing questions. This drawback was observed in TREC 8, and caused TREC organizers to change data set construction for later conferences by starting with questions input to a search engine, and then localize supporting sentences, rather than starting with the support (Voorhees, 2000). In actuality, many TurkQA question writers introduced their own wording and asked questions with more qualifications than a typical search engine query. They even asked 100 "why" questions.

4 Recursive neural networks

In their traditional form (Pollack, 1990), autoencoders consist of two neural networks: an encoder E to compress multiple input vectors into a single output vector, and a decoder D to restore the inputs from the compressed vector. Through recursion, autoencoders allow single vectors to represent variable length data structures. Supposing each terminal node t of a rooted tree T has been assigned a feature vector $\vec{x}(t) \in \mathbb{R}^n$, the encoder E is used to define n-dimensional feature vectors at all remaining nodes. Assuming for simplicity that T is a binary tree, the encoder E takes the form $E : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$. Given children c_1 and c_2 of a node p, the encoder assigns the representation $\vec{x}(p) = E(\vec{x}(c_1), \vec{x}(c_2))$. Applying this rule recursively defines vectors at every node of the tree.

The decoder and encoder may be trained together to minimize reconstruction error, typically Euclidean distance. Applied to a set of trees $\mathcal{T}$ with features already assigned at their terminal nodes, autoencoder training minimizes:

$$L_{ae} = \sum_{t \in \mathcal{T}} \sum_{p \in N(t)} \sum_{c_i \in C(p)} ||\vec{x}'(c_i) - \vec{x}(c_i)||, \qquad (1)$$

where $N(t)$ is the set of non-terminal nodes of tree

[1] https://www.mturk.com

Algorithm 1: Auto-encoders co-trained for subtree recognition by stochastic gradient descent

Data: $E : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ a neural network (encoder)
Data: $S : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^2$ a neural network for binary classification (subtree or not)
Data: $D : \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$ a neural network (decoder)
Data: $\mathcal{T}$ a set of trees T with features $\vec{x}(t)$ assigned to terminal nodes $t \in T$
Result: Weights of E and D trained to minimize a combination of reconstruction and subtree recognition error

```
begin
    while stopping criterion not satisfied do
        Randomly choose T ∈ 𝒯
        for p in a postorder depth first traversal of T do
            if p is not terminal then
                Let c_1, c_2 be the children of p
                Compute x⃗(p) = E(x⃗(c_1), x⃗(c_2))
                Let (x⃗'(c_1), x⃗'(c_2)) = D(x⃗(p))
                Compute reconstruction loss L_R = ||x⃗'(c_1) − x⃗(c_1)||_2 + ||x⃗'(c_2) − x⃗(c_2)||_2
                Choose a random q ∈ T such that q is a descendant of p
                Let c_1^q, c_2^q be the children of q, if they exist
                Compute S(x⃗(p), x⃗(q)) = S(E(x⃗(c_1), x⃗(c_2)), E(x⃗(c_1^q), x⃗(c_2^q)))
                Compute cross-entropy loss L_1 = h(S(x⃗(p), x⃗(q)), 1)
                if p is not the root of T then
                    Choose a random r ∈ T such that r is not a descendant of p
                    Let c_1^r, c_2^r be the children of r, if they exist
                    Compute cross-entropy loss L_2 = h(S(x⃗(p), x⃗(r)), 0)
                else
                    Let L_2 = 0
                Compute gradients of L_R + L_1 + L_2 with respect to weights of E, D, and S, fixing
                x⃗(c_2), x⃗(c_1^q), x⃗(c_2^q), x⃗(c_1^r), and x⃗(c_2^r).
                Update parameters of E, D, and S by backpropagation
```

t, $C(p) = c_1, c_2$ is the set of children of node p, and $(\vec{x}'(c_1), \vec{x}'(c_2)) = D(E(\vec{x}(c_1), \vec{x}(c_2)))$. This loss can be trained with stochastic gradient descent (Bottou, 2004).

However, there have been some perennial concerns about autoencoders:

1. Is information lost after repeated recursion?
2. Does low reconstruction error actually keep the information needed for classification?

Socher attempted to address the first of these concerns in his work on paraphrase with *deep unfolding recursive autoencoders* (Socher et al., 2011a), where each node is penalized for reconstruction errors many levels down an input tree, not just the reconstruction of its immediate descendants. Beyond five levels, Socher observed many word-choice errors on decoding input sentences. Socher's work on sentiment analysis (Socher et al., 2011b) focused on the second concern, by co-training on desired sentence classification, along with the usual reconstruction objective, at every level down to the terminal nodes. Of course, this had the side effect of imputing sentence-level sentiment labels to words where it was not really relevant.

As an alternative, we propose *subtree recognition* as a semi-supervised co-training task for any recurrent neural network on tree structures. This task can be defined just as generally as reconstruction error. While accepting that some information will be lost as we go up the tree, the co-training objective encourages the encoder to produce representations that can answer basic questions about the presence or absence of descendants far below.

Subtree recognition is a binary classification problem concerning two nodes x and y of a tree T; we train a neural network S to predict whether y is a descendant of x. The neural network S should produce two outputs, corresponding to log probabilities that the descendant relation is satisfied. In our experiments, we take S (as we do E and D) to have one hidden layer. We train the outputs $S(x, y) = (z_0, z_1)$ to minimize the cross-entropy function

$$h((z_0, z_1), j) = -\log\left(\frac{e^{z_j}}{e^{z_0} + e^{z_1}}\right) \text{ for } j = 0, 1. \tag{2}$$

so that z_0 and z_1 estimate log likelihoods that the descendant relation is satisfied.

Our algorithm for training the subtree classifier is presented in Algorithm 1. We use SENNA software (Collobert et al., 2011) to compute parse trees for sentences. Training on a corpus of 64,421 Wikipedia sentences and testing on 20,160, we achieve a test error

rate of 3.2% on pairs of parse tree nodes that are subtrees, for 6.9% on pairs that are not subtrees ($F1 = .95$), with .02 mean squared reconstruction error.

5 Features for question and answer data

Application of the recursive neural network begins with features from the terminal nodes (the tokens). These features come from the language model of SENNA (Collobert et al., 2011), the Semantic Extraction Neural Network Architecture. Originally, neural probabilistic language models associated words with learned feature vectors so that a neural network could predict the joint probability function of word sequences (Bengio and Ducharme, 2001). SENNA's language model is co-trained on many syntactic tagging tasks, with a semi-supervised task in which valid sentences are to be ranked above sentences with random word replacements. Through the ranking and tagging tasks, this model learned embeddings of each word in a 50-dimensional space. Besides this learned representations, we encode capitalization and SENNA's predictions of named entity and part of speech tags with random vectors associated to each possible tag, as shown in Figure 1. The dimensionality of these vectors is chosen roughly as the logarithm of the number of possible tags. Thus every terminal node obtains a 61-dimensional feature vector.

We modify the basic RNN construction of Section 4 to obtain features for interior nodes. Since interior tree nodes are tagged with a node type, we encode the possible node types in a six-dimensional vector and make E and D work on triples (ParentType, Child 1, Child 2), instead of pairs (Child 1, Child 2).

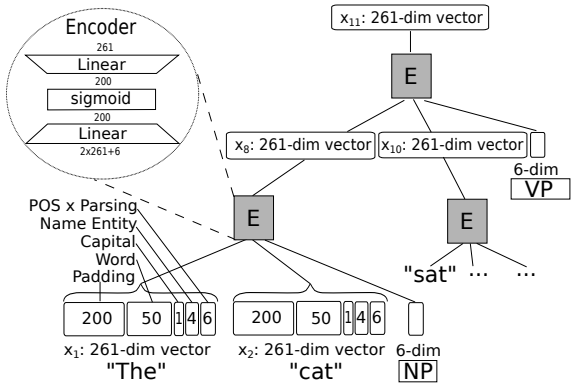

Figure 1: Recursive autoencoder to assign features to nodes of the parse tree of, "The cat sat on the mat." Note that the node types (e.g. "NP" or "VP") of internal nodes, and not just the children, are encoded.

Also, parse trees are not necessarily binary, so we binarize by right-factoring. Newly created internal nodes are labeled as "SPLIT" nodes. For example, a node with children c_1, c_2, c_3 is replaced by a new node with the same label, with left child c_1 and newly created right child, labeled "SPLIT," with children c_2 and c_3.

Vectors from terminal nodes are padded with 200 zeros before they are input to the autoencoder. We do this so that interior parse tree nodes have more room to encode the information about their children, as the original 61 dimensions may already be filled with information about just one word.

The feature construction is identical for the question and the support sentence.

5.1 Modeling unknown words

Many QA systems derive powerful features from exact word matches. In our approach, we trust that the classifier will be able to match information from autoencoder features of related parse tree branches, if it needs to. But our neural language probabilistic language model is at a great disadvantage if its features cannot characterize words outside its original training set.

Since Wikipedia is an encyclopedia, it is common for support sentences to introduce entities that do not appear in the dictionary of 100,000 most common words for which our language model has learned features. In the support sentence

> Jean-Bedel Georges Bokassa, Crown Prince of Central Africa was born on the 2nd November 1975 the son of Emperor Bokassa I of the Central African Empire and his wife Catherine Denguiade, who became Empress on Bokassa's accession to the throne.

both *Bokassa* and *Denguiade* are uncommon, and do not have learned language model embeddings. SENNA typically replaces these words with a fixed vector associated with all unknown words, and this works fine for syntactic tagging; the classifier learns to use the context around the unknown word. However, in a question-answering setting, we may need to read *Denguiade* from a question and be able to match it with *Denguiade*, not *Bokassa*, in the support.

Thus we extend the language model vectors with a random vector associated to each distinct word. The random vectors are fixed for all the words in the original language model, but a new one is generated the first time any unknown word is read. For known words, the original 50 dimensions give useful syntactic and semantic information. For unknown words, the newly introduced dimensions facilitate word matching without disrupting predictions based on the original 50.

6 Convolutions inside trees

We extract answers from support sentences by classifying each token as a word to be included in the answer or not. Essentially, this decision is a tagging problem on the support sentence, with additional features required from the question.

Convolutional neural networks efficiently classify sequential (or multi-dimensional) data, with the ability to reuse computations within a sliding frame tracking

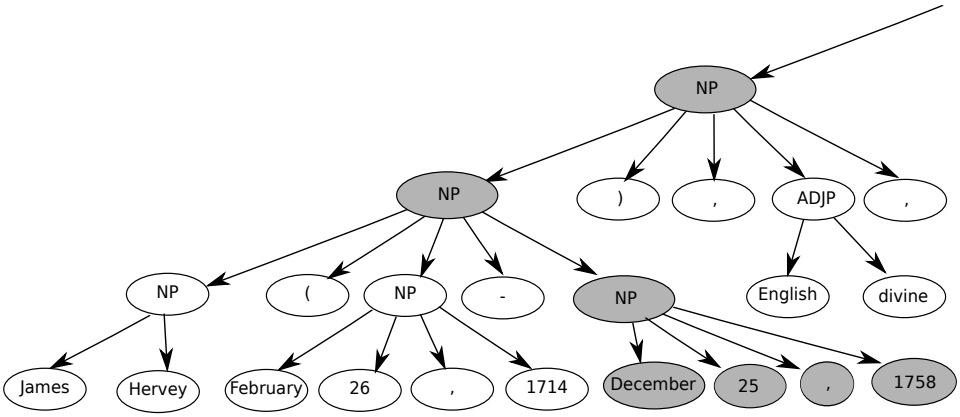

Figure 2: Labeling nodes to be followed to select "December 25, 1758."

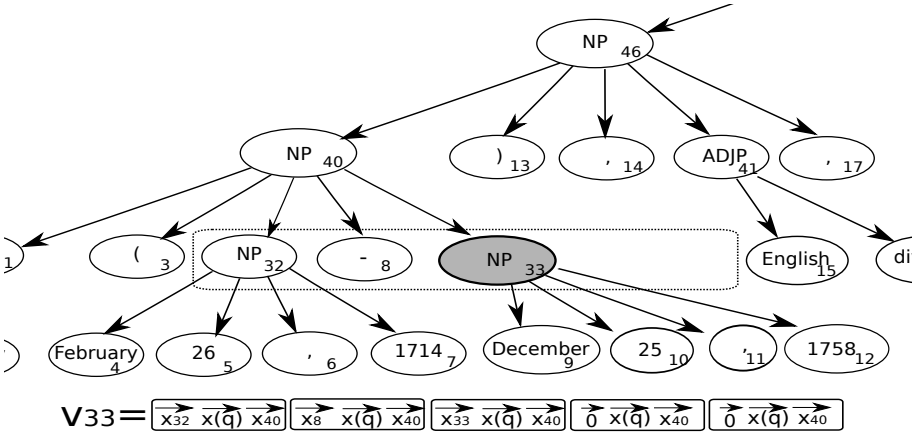

Figure 3: Assembling features from the question, the parent, and siblings, to decide whether to follow node 33

the item to be classified (Waibel et al., 1989). Convolving over token sequences has achieved state-of-the-art performance in part-of-speech tagging, named entity recognition, and chunking, and competitive performance in semantic role labeling and parsing, using one basic architecture (Collobert et al., 2011). Moreover, at classification time, the approach is 200 times faster at POS tagging than next-best systems such as (Shen et al., 2007) and 100 times faster at semantic role labeling (Koomen et al., 2005).

Classifying tokens to answer questions involves not only information from nearby tokens, but long range syntactic dependencies. In most work utilizing parse trees as input, a systematic description of the whole parse tree has not been used. Some state-of-the-art semantic role labeling systems require multiple parse trees (alternative candidates for parsing the same sentence) as input, but they measure many ad-hoc features describing path lengths, head words of prepositional phrases, clause-based path features, *etc.*, encoded in a sparse feature vector (Pradhan et al., 2005).

By using feature representations from our RNN and performing convolutions across siblings inside the tree, instead of token sequences in the text, we can utilize the parse tree information in a more principled way. We start at the root of the parse tree and select branches to follow, working down. At each step, the entire question is visible, via the representation at its root, and we decide whether or not to follow each branch of the support sentence. Ideally, irrelevant information will be cut at the point where syntactic information indicates it is no longer needed. The point at which we reach a terminal node may be too late to cut out the corresponding word; the context that indicates it is the wrong answer may have been visible only at a higher level in the parse tree. The classifier must cut words out earlier, though we do not specify exactly where.

Our classifier uses three pieces of information to decide whether to follow a node in the support sentence or not, given that its parent was followed:

1. The representation of the question at its root
2. The representation of the support sentence at the parent of the current node
3. The representations of the current node and a frame of k of its siblings on each side, in the order induced by the order of words in the sentence

Algorithm 2: Training the convolutional neural network for question answering

Data: Ξ, a set of triples (Q, S, T), with Q a parse tree of a question, S a parse tree of a support sentence, and $T \subset \mathcal{W}(S)$ a ground truth answer substring, and parse tree features $\vec{x}(p)$ attached by the recursive autoencoder for all $p \in Q$ or $p \in S$

Let $n = \dim \vec{x}(p)$

Let h be the cross-entropy loss (equation (2))

Data: $\Phi : \left(\mathbb{R}^{3n}\right)^{2k+1} \to \mathbb{R}^2$ a convolutional neural network over frames of size $2k+1$, with parameters to be trained for question-answering

Result: Parameters of Φ trained

begin

  while *stopping criterion not satisfied* **do**

    Randomly choose $(Q, S, T) \in \Xi$

    Let $q, r = root(Q), root(S)$

    Let $X = \{r\}$ (the set of nodes to follow)

    Let $\mathcal{A}(T) \subset S$ be the set of ancestor nodes of T in S

    while $X \neq \emptyset$ **do**

      Pop an element p from X

      if p *is not terminal* **then**

        Let $c_1, \ldots, c_m$ be the children of p

        Let $\vec{x_j} = \vec{x}(c_j)$ for $j \in \{1, \ldots, m\}$

        Let $\vec{x_j} = \vec{0}$ for $j \notin \{1, \ldots, m\}$

        for *i=1, ... m* **do**

          Let $t = 1$ if $c_i \in \mathcal{A}(T)$, or 0 otherwise

          Let $v_{c_i} = \oplus_{j=i-k}^{i+k} \left(\vec{x}_j \oplus \vec{x}(p) \oplus \vec{x}(q)\right)$

          Compute the cross-entropy loss $h\left(\Phi\left(v_{c_i}\right), t\right)$

          if $\exp(-h\left(\Phi\left(v_{c_i}\right), 1\right)) > \frac{1}{2}$ **then**

            Let $X = X \cup \{c_i\}$ (the network predicts c_i should be followed)

          Update parameters of Φ by backpropagation

Each of these representations is n-dimensional. The convolutional neural network concatenates them together (denoted by $\oplus$) as a $3n$-dimensional feature at each node position, and considers a frame enclosing k siblings on each side of the current node. The CNN consists of a convolutional layer mapping the $3n$ inputs to an r-dimensional space, a sigmoid function (such as $\tanh$), a linear layer mapping the r-dimensional space to two outputs, and another sigmoid. We take $k = 2$ and $r = 30$ in the experiments.

Application of the CNN begins with the children of the root, and proceeds in breadth first order through the children of the followed nodes. Sliding the CNN's frame across siblings allows it to decide whether to follow adjacent siblings faster than a non-convolutional classifier, where the decisions would be computed without exploiting the overlapping features. A followed terminal node becomes part of the short answer of the system.

The training of the question-answering convolutional neural network is detailed in Algorithm 2. Only visited nodes, as predicted by the classifier, are used for training. For ground truth, we say that a node should be followed if it is the ancestor of some token that is part of the desired answer. For example, to select the death date "December 25, 1758" from the support sentence (displayed on page one) about James Hervey, nodes of the tree would be attached ground truth values according to the coloring in Figure 2. At classification time, some unnecessary (negatively labeled) nodes may be followed without mistaking the final answer.

For example, when deciding whether to follow the "NP" in node 33 on the third row of Figure 3, the classifier would see features of node 32 (NP) and node 8 ('-') on its left, its own features, and nothing on its right. Since there are no siblings to the right of node 33, zero vectors, used for padding, would be placed in the two empty slots. To each of these feature vectors, features from the parent and the question root would be concatenated.

The combination of recursive autoencoders with convolutions inside the tree affords flexibility and generality. The ordering of children would be immeasurable by a classifier relying on path-based features alone. For instance, our classifier may consider a branch of a parse tree as in Figure 2, in which the birth date and death date have isomorphic connections to the rest of the parse tree. It can distinguish them by the ordering of nodes in a parenthetical expression (see examples in Table 2).

System	Short Answer Precision	Short Answer Recall	Short Answer F1	MC Precision	MC Recall	MC F1
Main	58.9%	27.0%	.370	79.9%	38.8%	.523
No subtree recognition	48.9%	18.6%	.269	71.7%	27.8%	.400
No unknown word embeddings	66.8%	19.7%	.305	84.2%	29.0%	.431
Smaller training (1,333 questions)	40.6%	16.7%	.236	65.5%	20.4%	.311
OpenEphyra	52.2%	13.1%	.209	73.6%	32.0 %	.446

Table 1: Performance on TurkQA test set, for short answer and multiple choice (MC) evaluations.

7 Experiments

From the TurkQA data, we disregard the yes/no questions, and obtain 12,916 problem sets with 38,083 short answer questions for training, and 508 problem sets with 1,488 short answer questions for testing.

Because there may be several ways of stating a short answer, short answer questions in other data sets are typically judged by humans. In TurkQA, because answers must be extracted as substrings, we can approximate the machine's correctness by considering the token classification error against the substring originally chosen by the Turk worker. Of course, answer validation strategies could be used to clean up the short answers—for instance, require that they are contiguous substrings (as is guaranteed by the task)—but we did not employ them here, so as not to obfuscate the performance of the substring extraction system itself. Inevitably, some token misclassification will occur because question writers choose more or less complete answers ("in Nigeria" or just "Nigeria").

Table 6 shows the performance of our main algorithm, evaluated both as short-answer and as multiple choice. The short answer results describe the main setting, which formulates answer extraction as token classification. The multiple choice results come from considering all the short answers in the problem sets as alternative choices, and comparing the classifier's outputs averaged over the words in each response to select the best, or skip if no average is positive. (Thus, all questions in a single problem set have the same set of choices.) Although the multiple choice setting is less challenging, it helps us see how much of the short answer error may be due to finding poor answer boundaries as opposed to the classifier being totally misled. On more than half of the 1,488 test questions, no answer at all is selected, so that multiple choice precision remains high even with low recall.

As one baseline method, we took the OpenEphyra question answering system, an open source project led by Carnegie Mellon University, which evolved out of submissions to TREC question answering contests (Ko et al., 2007), bypassing its retrieval module to simply use our support sentence. In contrast to our system, OpenEphyra's question analysis module is trained to map questions to one of a fixed number of answer types, such as `PERCENTAGE`, or `PROPER_NAME.PERSON.FIRST_NAME`, and utilizes a large database of answer patterns for these types. In spite of OpenEphyra's laborious pattern coding, our system performs 17% better on a multiple choice basis, and 77% better on short answers, the latter likely because OpenEphyra's answer types cover shorter strings than the Turks' answers.

The results show the impact of several of our algorithmic contributions. If the autoencoder is trained only on reconstruction error and not subtree recognition, the F1 score for token classification drops from .370 (58.9% precision, 27.0% recall) to .269 (48.9% precision, 18.6% recall). Without extended embeddings to differentiate unknown words, F1 is only .305 (66.8% precision, 19.7% recall). We are encouraged that increasing the amount of data contributes 50% to the F1 score (from only F1=.236 training on 1,333 questions), as it suggests that the power of our algorithms is not saturated while picking up the simplest features.

Table 2 gives examples of questions in the test set, together with the classifier's selection from the support sentence.

8 Discussion

We have developed a recursive neural network architecture capable of using learned representations of words and syntax in a parse tree structure to answer free form questions about natural language text. Using meaning representations of the question and supporting sentences, our approach buys us freedom from explicit rules, question and answer types, and exact string matching.

Certainly retrieval is important in a full-fledged question answering system, whereas our classifier performs deep analysis after candidate supporting sentences have been identified. Also, multi-sentence documents would require information to be linked among coreferent entities. Despite these challenges, we present our system in the belief that strong QA technologies should begin with a mastery of the syntax of single sentences. A computer cannot be said to have a complete knowledge representation of a sentence until it can answer all the questions a human can ask about that sentence.

Question:	What is the name of the British charity based in Haringey in north London?
Support:	Based in Haringey in North London, Exposure is a British charity which enables children and young people from all backgrounds, including disadvantaged groups and those from areas of deprivation, to participate and achieve their fullest potential in the media.
Selection:	Exposure
Correct Answer:	Exposure
Question:	What was Robert Person's profession?
Support:	Robert Person was a professional baseball pitcher who played 9 seasons in Major League Baseball: two for the New York Mets, two and a half for the Toronto Blue Jays, three and a half for the Philadelphia Phillies, and only pitched 7 games for the Boston Red Sox in the last year of his career.
Selection:	baseball pitcher
Correct Answer:	baseball pitcher
Question:	How many seasons did Robert Person play in the Major League?
Support:	Robert Person was a professional baseball pitcher who played 9 seasons in Major League Baseball: two for the New York Mets, two and a half for the Toronto Blue Jays, three and a half for the Philadelphia Phillies, and only pitched 7 games for the Boston Red Sox in the last year of his career.
Selection:	9
Correct Answer:	9 seasons
Question:	What sea does Mukka have a shore on?
Support:	Mukka is suburb of Mangalore city on the shore of Arabian sea . It is located to north of NITK, Surathkal campus on National Highway 17 . There is a beach in Mukka which has escaped public attention.
Selection:	Arabian sea
Correct Answer:	Arabian sea
Question:	What genre was Lights Out?
Support:	Lights Out was an extremely popular American old-time radio program, an early example of a network series devoted mostly to horror and the supernatural, predating Suspense and Inner Sanctum.
Selection:	horror supernatural
Correct Answer:	horror and the supernatural
Question:	Where is the Arwa Group?
Support:	The Arwa Group is a set of three Himalayan peaks, named Arwa Tower, Arwa Crest, and Arwa Spire, situated in the Chamoli district of Uttarakhand state, in northern India.
Selection:	the Chamoli district Uttarakhand state northern India
Correct Answer:	the Chamoli district of Uttarakhand state
Question:	What year did Juan Bautista Segura die in?
Support:	Juan Bautista Quiros Segura (1853 - 1934) was president of Costa Rica for two weeks, from August 20 to September 2, 1919, following the resignation of Federico Tinoco.
Selection:	1934
Correct Answer:	1934
Question:	What state is the Oregon School Activities Association in?
Support:	The Oregon School Activities Association, or OSAA, is a non-profit, board-governed organization that regulates high school athletics and competitive activities in Oregon, providing equitable competition amongst its members, both public and private.
Selection:	OSAA
Correct Answer:	Oregon

Table 2: Example results of main classifier on TurkQA test set.

References

Y. Bengio and R. Ducharme. 2001. A neural probabilistic language model. In *Advances in NIPS*, volume 13.

L. Bottou. 2004. Stochastic learning. In O. Bousquet and U. von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer.

S. Brin. 1998. Extracting patterns and relations from the World Wide Web. In *Proceedings World Wide Web and Databases International Workshop (LNCS 1590)*, pages 172–183. Springer.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

H. T. Dang, D. Kelly, and J. Lin. 2008. Overview of the TREC 2007 question answering track. In *NIST Special Pub. 500-274: The Sixteenth Text Retrieval Conference (TREC 2007)*.

P. Forner, A. Penas, E. Agirre, I. Alegria, C. Forascu, N. Moreau, P. Osenova, P. Prokopidis, P. Rocha, B. Sacaleanu, R. Sutcliffe, and E. T. K. Sang. 2008. Overview of the Clef 2008 Multilingual Question Answering Track. In *CLEF*.

J. Ko, E. Nyberg, and L. Luo Si. 2007. A probabilistic graphical model for joint answer ranking in question answering. In *Proceedings of the 30th ACM SIGIR Conference*.

P. Koomen, V. Punyakanok, D. Roth, and W. Yih. 2005. Generalized inference with multiple semantic role labeling systems. In *Proceedings of CoNLL*.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *ACL*.

M. Mintz, S. Bills, R. Snow, and D. Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of ACL-IJCNLP*, pages 1003–1011.

J. B. Pollack. 1990. Recursive distributed representations. *Artficial Intelligence*, 46.

H. Poon and P. Domingos. 2009. Unsupervised semantic parsing. In *Proceedings of ACL*.

S. Pradhan, K. Hacioglu, W. Ward, J. H. Martin, and D. Jurafsky. 2005. Semantic role chunking combining complementary syntactic views. In *Conference on Computational Natural Language Learning (CoNLL)*, pages 217–220.

S. Riedel, L. Yao, and A. McCallum. 2010. Modeling relations and their mentions without labeled text. In *Proceedings of ECML-PKDD*.

R. Schmidt. 2013. The fan-created archive of Jeopardy! games and players. `http://www.j-archive.com`. Accessed: April 26, 2013.

L. Shen, G. Satta, and A. K. Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of ACL*.

R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning. 2011a. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in NIPS*.

R. Socher, J. Pennington, E. Huang, A. Y. Ng, and C. D. Manning. 2011b. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of EMNLP*.

E. M. Voorhees. 2000. Overview of the TREC-9 Question Answering track. In *NIST Special Pub. 500-249: The Ninth Text Retrieval Conference (TREC 9)*, pages 71–79.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. 1989. Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 37(3):328–339.