A Parallel-Hierarchical Model for Machine Comprehension on Sparse Data

Adam Trischler* adam.trischler
Zheng Ye* jeff.ye
Xingdi Yuan eric.yuan
Jing He jing.he
Philip Bachman phil.bachman
Kaheer Suleman k.suleman@maluuba.com
Maluuba Research
Montreal, Québec, Canada

Abstract

Understanding unstructured text is a major goal within natural language processing. Comprehension tests pose questions based on short text passages to evaluate such understanding. In this work, we investigate machine comprehension on the challenging *MCTest* benchmark. Partly because of its limited size, prior work on *MCTest* has focused mainly on engineering better features. We tackle the dataset with a neural approach, harnessing simple neural networks arranged in a parallel hierarchy. The parallel hierarchy enables our model to compare the passage, question, and answer from a variety of trainable perspectives, as opposed to using a manually designed, rigid feature set. Perspectives range from the word level to sentence fragments to sequences of sentences; the networks operate only on word-embedding representations of text. When trained with a methodology designed to help cope with limited training data, our Parallel-Hierarchical model sets a new state of the art for *MCTest*, outperforming previous feature-engineered approaches slightly and previous neural approaches by a significant margin (over 15 percentage points).

1 Introduction

Humans learn in a variety of ways—by communication with each other and by study, the reading of text. Comprehension of unstructured text by machines, at a near-human level, is a major goal for natural language processing. It has garnered significant attention from the machine learning research community in recent years.

Machine comprehension (MC) is evaluated by posing a set of questions based on a text passage (akin to the reading tests we all took in school). Such tests are objectively gradable and can be used to assess a range of abilities, from basic understanding to causal reasoning to inference (Richardson et al., 2013). Given a text passage and a question about its content, a system is tested on its ability to determine the correct answer (Sachan et al., 2015). In this work, we focus on *MCTest*, a complex but data-limited comprehension benchmark, whose multiple-choice questions require not only extraction but also inference and limited reasoning (Richardson et al., 2013). Inference and reasoning are important human skills that apply broadly, beyond language.

We present a parallel-hierarchical approach to machine comprehension designed to work well in a data-limited setting. There are many use-cases in which comprehension over limited data would be handy: for example, user manuals, internal documentation, legal contracts, and so on. Moreover, work towards more efficient learning from any quantity of data is important in its own right, for bringing machines more in line with the way humans learn. Typically, artificial neural networks require numerous parameters to capture complex patterns, and the more parameters, the more training data is required to tune them. Likewise, deep models learn to extract their own features, but this is a data-intensive process. Our model learns to comprehend at a high level even when data is sparse.

The key to our model is that it compares the question and answer candidates to the text using several distinct *perspectives*. We refer to a question combined with one of its answer candidates as a *hypothesis* (to be detailed below). The *seman-*

*A. Trischler and Z. Ye contributed equally to this work.

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 432–441,
Berlin, Germany, August 7-12, 2016.

tic perspective compares the hypothesis to sentences in the text viewed as single, self-contained thoughts; these are represented using a sum and transformation of word embedding vectors, similarly to Weston et al. (2014). The *word-by-word* perspective focuses on similarity matches between individual words from hypothesis and text, at various scales. As in the semantic perspective, we consider matches over complete sentences. We also use a sliding window acting on a subsentential scale (inspired by the work of Hill et al. (2015)), which implicitly considers the linear distance between matched words. Finally, this word-level sliding window operates on two different views of story sentences: the *sequential* view, where words appear in their natural order, and the *dependency* view, where words are reordered based on a linearization of the sentence's dependency graph. Words are represented throughout by embedding vectors (Bengio et al., 2000; Mikolov et al., 2013). These distinct perspectives naturally form a hierarchy that we depict in Figure 1. Language is hierarchical, so it makes sense that comprehension relies on hierarchical levels of understanding.

The perspectives of our model can be considered a type of feature. However, they are implemented by parametric differentiable functions. This is in contrast to most previous efforts on *MCTest*, whose numerous hand-engineered features cannot be trained. Our model, significantly, can be trained end-to-end with backpropagation. To facilitate learning with limited data, we also develop a unique training scheme. We initialize the model's neural networks to perform specific heuristic functions that yield decent (though not impressive) performance on the dataset. Thus, the training scheme gives the model a safe, reasonable baseline from which to start learning. We call this technique *training wheels*.

Computational models that comprehend (insofar as they perform well on MC datasets) have been developed contemporaneously in several research groups (Weston et al., 2014; Sukhbaatar et al., 2015; Hill et al., 2015; Hermann et al., 2015; Kumar et al., 2015). Models designed specifically for *MCTest* include those of Richardson et al. (2013), and more recently Sachan et al. (2015), Wang et al. (2015), and Yin et al. (2016). In experiments, our *Parallel-Hierarchical* model achieves state-of-the-art accuracy on *MCTest*, outperforming these existing methods.

Below we describe related work, the mathematical details of our model, and our experiments, then analyze our results.

2 The Problem

In this section, we borrow from Sachan et al. (2015), who laid out the MC problem nicely. Machine comprehension requires machines to answer questions based on unstructured text. This can be viewed as selecting the best answer from a set of candidates. In the multiple-choice case, candidate answers are predefined, but candidate answers may also be undefined yet restricted (*e.g.*, to *yes*, *no*, or any noun phrase in the text) (Sachan et al., 2015).

For each question q, let T be the unstructured text and $A = \{a_i\}$ the set of candidate answers to q. The machine comprehension task reduces to selecting the answer that has the highest evidence given T. As in Sachan et al. (2015), we combine an answer and a question into a *hypothesis*, $h_i = f(q, a_i)$. To facilitate comparisons of the text with the hypotheses, we also break down the passage into sentences t_j, $T = \{t_j\}$. In our setting, q, a_i, and t_j each represent a sequence of embedding vectors, one for each word and punctuation mark in the respective item.

3 Related Work

Machine comprehension is currently a hot topic within the machine learning community. In this section we will focus on the best-performing models applied specifically to *MCTest*, since it is somewhat unique among MC datasets (see Section 5). Generally, models can be divided into two categories: those that use fixed, engineered features, and neural models. The bulk of the work on *MCTest* falls into the former category.

Manually engineered features often require significant effort on the part of a designer, and/or various auxiliary tools to extract them, and they cannot be modified by training. On the other hand, neural models can be trained end-to-end and typically harness only a single feature: vector-representations of words. Word embeddings are fed into a complex and possibly deep neural network which processes and compares text to question and answer. Among deep models, mechanisms of attention and working memory are common, as in Weston et al. (2014) and Hermann et al. (2015).

3.1 Feature-engineering models

Sachan et al. (2015) treated *MCTest* as a structured prediction problem, searching for a latent *answer-entailing* structure connecting question, answer, and text. This structure corresponds to the best latent alignment of a hypothesis with appropriate snippets of the text. The process of (latently) selecting text snippets is related to the attention mechanisms typically used in deep networks designed for MC and machine translation (Bahdanau et al., 2014; Weston et al., 2014; Hill et al., 2015; Hermann et al., 2015). The model uses event and entity coreference links across sentences along with a host of other features. These include specifically trained word vectors for synonymy; antonymy and class-inclusion relations from external database sources; dependencies and semantic role labels. The model is trained using a latent structural SVM extended to a multitask setting, so that questions are first classified using a pretrained top-level classifier. This enables the system to use different processing strategies for different question categories. The model also combines question and answer into a well-formed statement using the rules of Cucerzan and Agichtein (2005).

Our model is simpler than that of Sachan et al. (2015) in terms of the features it takes in, the training procedure (stochastic gradient descent *vs.* alternating minimization), question classification (we use none), and question-answer combination (simple concatenation or mean *vs.* a set of rules).

Wang et al. (2015) augmented the baseline feature set from Richardson et al. (2013) with features for syntax, frame semantics, coreference chains, and word embeddings. They combined features using a linear latent-variable classifier trained to minimize a max-margin loss function. As in Sachan et al. (2015), questions and answers are combined using a set of manually written rules. The method of Wang et al. (2015) achieved the previous state of the art, but has significant complexity in terms of the feature set.

Space does not permit a full description of all models in this category, but we refer the reader to the contributions of Smith et al. (2015) and Narasimhan and Barzilay (2015) as well.

Despite its relative lack of features, the Parallel-Hierarchical model improves upon the feature-engineered state of the art for *MCTest* by a small amount (about 1% absolute) as detailed in Section 5.

3.2 Neural models

Neural models have, to date, performed relatively poorly on *MCTest*. This is because the dataset is sparse and complex.

Yin et al. (2016) investigated deep-learning approaches concurrently with the present work. They measured the performance of the Attentive Reader (Hermann et al., 2015) and the Neural Reasoner (Peng et al., 2015), both deep, end-to-end recurrent models with attention mechanisms, and also developed an attention-based convolutional network, the HABCNN. Their network operates on a hierarchy similar to our own, providing further evidence of the promise of hierarchical perspectives. Specifically, the HABCNN processes text at the sentence level and the *snippet* level, where the latter combines adjacent sentences (as we do through an n-gram input). Embedding vectors for the question and the answer candidates are combined and encoded by a convolutional network. This encoding modulates attention over sentence and snippet encodings, followed by max-pooling to determine the best matches between question, answer, and text. As in the present work, matching scores are given by cosine similarity. The HABCNN also makes use of a question classifier.

Despite the conceptual overlap between the HABCNN and our approach, the Parallel-Hierarchical model performs significantly better on *MCTest* (more than 15% absolute) as detailed in Section 5. Other neural models tested in Yin et al. (2016) fare even worse.

4 The Parallel-Hierarchical Model

Let us now define our machine comprehension model in full. We first describe each of the perspectives separately, then describe how they are combined. Below, we use subscripts to index elements of sequences, like word vectors, and superscripts to indicate whether elements come from the text, question, or answer. In particular, we use the subscripts k, m, n, p to index sequences from the text, question, answer, and hypothesis, respectively, and superscripts t, q, a, h. We depict the model schematically in Figure 1.

4.1 Semantic Perspective

The semantic perspective is similar to the Memory Networks approach for embedding inputs into memory space (Weston et al., 2014). Each sen-

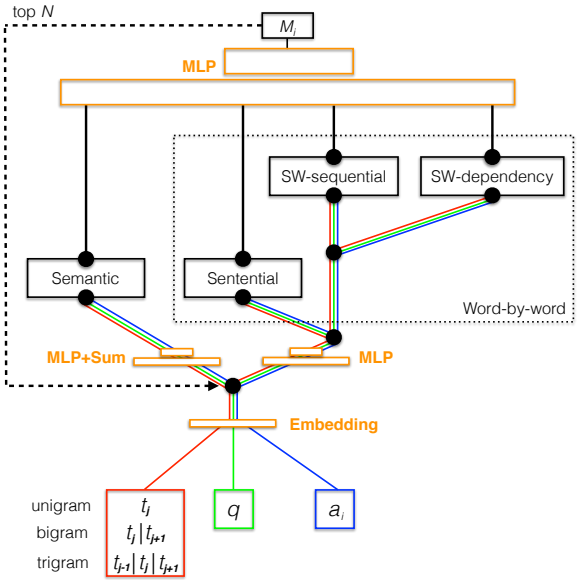

Figure 1: Schematic of the Parallel-Hierarchical model. SW stands for "sliding window." MLP represents a fully connected neural network.

tence of the text is a sequence of d-dimensional word vectors: $t_j = \{\mathbf{t}_k\}$, $\mathbf{t}_k \in \mathbb{R}^d$. The semantic vector $\mathbf{s}^t$ is computed by embedding the word vectors into a D-dimensional space using a two-layer network that implements weighted sum followed by an affine tranformation and a nonlinearity; *i.e.*,

$$\mathbf{s}^t = f\left(\mathbf{A}^t \sum_k \omega_k \mathbf{t}_k + \mathbf{b}_A^t\right). \quad (1)$$

The matrix $\mathbf{A}^t \in \mathbb{R}^{D\times d}$, the bias vector $\mathbf{b}_A^t \in \mathbb{R}^D$, and for f we use the *leaky ReLU* function. The scalar ω_k is a trainable weight associated with each word in the vocabulary. These scalar weights implement a kind of *exogenous* or bottom-up attention that depends only on the input stimulus (Mayer et al., 2004). They can, for example, learn to perform the function of stopword lists in a soft, trainable way, to nullify the contribution of unimportant filler words.

The semantic representation of a hypothesis is formed analogously, except that we concatenate the question word vectors $\mathbf{q}_m$ and answer word vectors $\mathbf{a}_n$ as a single sequence $\{\mathbf{h}_p\} = \{\mathbf{q}_m, \mathbf{a}_n\}$. For semantic vector $\mathbf{s}^h$ of the hypothesis, we use a unique transformation matrix $\mathbf{A}^h \in \mathbb{R}^{D\times d}$ and bias vector $\mathbf{b}_A^h \in \mathbb{R}^D$.

These transformations map a text sentence and a hypothesis into a common space where they can be compared. We compute the semantic match between text sentence and hypothesis using the cosine similarity,

$$M^{\text{sem}} = \cos(\mathbf{s}^t, \mathbf{s}^h). \quad (2)$$

4.2 Word-by-Word Perspective

The first step in building the word-by-word perspective is to transform word vectors from a text sentence, question, and answer through respective neural functions. For the text, $\tilde{\mathbf{t}}_k = f\left(\mathbf{B}^t\mathbf{t}_k + \mathbf{b}_B^t\right)$, where $\mathbf{B}^t \in \mathbb{R}^{D\times d}$, $\mathbf{b}_B^t \in \mathbb{R}^D$ and f is again the *leaky ReLU*. We transform the question and the answer to $\tilde{\mathbf{q}}_m$ and $\tilde{\mathbf{a}}_n$ analogously using distinct matrices and bias vectors. In contrast to the semantic perspective, we keep the question and answer candidates separate in the word-by-word perspective. This is because matches to answer words are inherently more important than matches to question words, and we want our model to learn to use this property.

4.2.1 Sentential

Inspired by the work of Wang and Jiang (2015) in paraphrase detection, we compute matches between hypotheses and text sentences at the word level. This computation uses the cosine similarity as before:

$$c_{km}^q = \cos(\tilde{\mathbf{t}}_k, \tilde{\mathbf{q}}_m), \quad (3)$$
$$c_{kn}^a = \cos(\tilde{\mathbf{t}}_k, \tilde{\mathbf{a}}_n). \quad (4)$$

The word-by-word match between a text sentence and question is determined by taking the maximum over k (finding the text word that best matches each question word) and then taking a weighted mean over m (finding the average match over the full question):

$$M^q = \frac{1}{Z}\sum_m \omega_m \max_k c_{km}^q. \quad (5)$$

Here, ω_m is the word weight for the question word and Z normalizes these weights to sum to one over the question. We define the match between a sentence and answer candidate, M^a, analogously. Finally, we combine the matches to question and answer according to

$$M^{\text{word}} = \alpha_1 M^q + \alpha_2 M^a + \alpha_3 M^q M^a. \quad (6)$$

Here, the α are trainable parameters that control the relative importance of the terms.

4.2.2 Sequential Sliding Window

The sequential sliding window is related to the original *MCTest* baseline by Richardson et al. (2013). Our sliding window decays from its focus word according to a Gaussian distribution, which we extend by assigning a trainable weight to each location in the window. This modification enables the window to use information about the distance between word matches; the original baseline (Richardson et al., 2013) used distance information through a predefined function.

The sliding window scans over the words of the text as one continuous sequence, without sentence breaks. Each window is treated like a sentence in the previous subsection, but we include a location-based weight $\lambda(k)$. This weight is based on a word's position in the window, which, given a window, depends on its global position k. The cosine similarity is adapted as

$$s^q_{km} = \lambda(k) \cos(\tilde{\mathbf{t}}_k, \tilde{\mathbf{q}}_m), \tag{7}$$

for the question and analogously for the answer. We initialize the location weights with a Gaussian and fine-tune them during training. The final matching score, denoted as M^{sws}, is computed as in (5) and (6) with s^q_{km} replacing c^q_{km}.

4.2.3 Dependency Sliding Window

The dependency sliding window operates identically to the linear sliding window, but on a different view of the text passage. The output of this component is M^{swd} and is formed analogously to M^{sws}.

The dependency perspective uses the Stanford Dependency Parser (Chen and Manning, 2014) as an auxiliary tool. Thus, the dependency graph can be considered a fixed feature. Moreover, linearization of the dependency graph, because it relies on an eigendecomposition, is not differentiable. However, we handle the linearization in data preprocessing so that the model sees only reordered word-vector inputs.

Specifically, we run the Stanford Dependency Parser on each text sentence to build a dependency graph. This graph has n_w vertices, one for each word in the sentence. From the dependency graph we form the Laplacian matrix $\mathbf{L} \in \mathbb{R}^{n_w \times n_w}$ and determine its eigenvectors. The second eigenvector $\mathbf{u}_2$ of the Laplacian is known as the *Fiedler* vector. It is the solution to the minimization

$$\underset{g}{\text{minimize}} \sum_{i,j=1}^{N} \eta_{ij} (g(v_i) - g(v_j))^2, \tag{8}$$

where v_i are the vertices of the graph and η_{ij} is the weight of the edge from vertex i to vertex j (Golub and Van Loan, 2012). The Fiedler vector maps a weighted graph onto a line such that connected nodes stay close, modulated by the connection weights.[1] This enables us to reorder the words of a sentence based on their proximity in the dependency graph. The reordering of the words is given by the ordered index set

$$I = \arg\text{sort}(\mathbf{u}_2). \tag{9}$$

To give an example of how this works, consider the following sentence from *MCTest* and its dependency-based reordering:

> *Jenny, Mrs. Mustard 's helper, called the police.*
> *the police, called Jenny helper, Mrs. 's Mustard.*

Sliding-window-based matching on the original sentence will answer the question *Who called the police?* with *Mrs. Mustard*. The dependency reordering enables the window to determine the correct answer, *Jenny*.

4.3 Combining Distributed Evidence

It is important in comprehension to synthesize information found throughout a document. *MCTest* was explicitly designed to ensure that it could not be solved by lexical techniques alone, but would instead require some form of inference or limited reasoning (Richardson et al., 2013). It therefore includes questions where the evidence for an answer spans several sentences.

To perform synthesis, our model also takes in n-grams of sentences, *i.e.*, sentence pairs and triples strung together. The model treats these exactly as it treats single sentences, applying all functions detailed above. A later pooling operation combines scores across all n-grams (including the single-sentence input). This is described in the next subsection.

[1] We experimented with assigning unique edge weights to unique relation types in the dependency graph. However, this had negligible effect. We hypothesize that this is because dependency graphs are trees, which do not have cycles.

With n-grams, the model can combine information distributed across contiguous sentences. In some cases, however, the required evidence is spread across distant sentences. To give our model some capacity to deal with this scenario, we take the top N sentences as scored by all the preceding functions, and then repeat the scoring computations, viewing these top N as a single sentence.

The reasoning behind these approaches can be explained well in a probabilistic setting. If we consider our similarity scores to model the likelihood of a text sentence given a hypothesis, $p(t_j \mid h_i)$, then the n-gram and top N approaches model a joint probability $p(t_{j_1}, t_{j_2}, \ldots, t_{j_k} \mid h_i)$. We cannot model the joint probability as a product of individual terms (score values) because distributed pieces of evidence are likely not independent.

4.4 Combining Perspectives

We use a multilayer perceptron (MLP) to combine M^{sem}, M^{word}, M^{swd}, and M^{sws}, as well as the scores for separate n-grams, as a final matching score M_i for each answer candidate. This MLP has multiple layers of staged input, because the distinct scores have different dimensionality: there is one M^{sem} and one M^{word} for each story sentence, and one M^{swd} and one M^{sws} for each application of the sliding window. The MLP's activation function is linear.

Our overall training objective is to minimize the ranking loss

$$\mathcal{L}(T, q, A) = \max(0, \mu + \max_i M_{i \neq i^*} - M_{i^*}), \tag{10}$$

where μ is a constant margin, i^* indexes the correct answer. We take the maximum over i so that we are ranking the correct answer over the best-ranked incorrect answer (of which there are three). This approach worked better than comparing the correct answer to the incorrect answers individually as in Wang et al. (2015).

Our implementation of the Parallel-Hierarchical model, built in *Theano* (Bergstra et al., 2010) using the *Keras* framework (Chollet, 2015), is available on `Github`.[2]

4.5 Training Wheels

Before training, we initialized the neural-network components of our model to perform sensible heuristic functions. Training did not converge on the small *MCTest* without this vital approach.

[2]https://github.com/Maluuba/mctest-model

Empirically, we found that we could achieve above 50% accuracy on *MCTest* using a simple sum of word vectors followed by a dot product between the story-sentence sum and the hypothesis sum. Therefore, we initialized the network for the semantic perspective to perform this sum, by initializing $\mathbf{A}^x$ as the *identity* matrix and $\mathbf{b}_A^x$ as the zero vector, $x \in \{t, h\}$. Recall that the activation function is a $ReLU$ so that positive outputs are unchanged.

We also found basic word-matching scores to be helpful, so we initialized the word-by-word networks likewise. The network for perspective-combination was initialized to perform a sum of individual scores, using a *zero* bias-vector and a weight matrix of *ones*, since we found that each perspective contributed positively to the overall result.

This *training wheels* approach is related to other techniques from the literature. For instance, Socher et al. (2013) proposed the identity-matrix initialization in the context of parsing, and Le et al. (2015) proposed it in the context of recurrent neural networks (to preserve the error signal through backpropagation). In residual networks (He et al., 2015), shortcut connections bypass certain layers in the network so that a simpler function can be trained in conjunction with the full model.

5 Experiments

5.1 The Dataset

MCTest is a collection of 660 elementary-level children's stories and associated questions, written by human subjects. The stories are fictional, ensuring that the answer must be found in the text itself, and carefully limited to what a young child can understand (Richardson et al., 2013).

The more challenging variant consists of 500 stories with four multiple-choice questions each. Despite the elementary level, stories and questions are more natural and more complex than those found in synthetic MC datasets like *bAbI* (Weston et al., 2014) and *CNN* (Hermann et al., 2015).

MCTest is challenging because it is both complicated and small. As per Hill et al. (2015), "it is very difficult to train statistical models only on *MCTest*." Its size limits the number of parameters that can be trained, and prevents learning any complex language modeling simultaneously with the capacity to answer questions.

5.2 Training and Model Details

In this section we describe important details of the training procedure and model setup. For a complete list of hyperparameter settings, our stopword list, and other minutiæ, we refer interested readers to our `Github` repository.

For word vectors we use Google's publicly available embeddings, trained with `word2vec` on the 100-billion-word *News corpus* (Mikolov et al., 2013). These vectors are kept fixed throughout training, since we found that training them was not helpful (likely because of *MCTest*'s size). The vectors are 300-dimensional ($d = 300$).

We do not use a stopword list for the text passage, instead relying on the trainable word weights to ascribe global importance ratings to words. These weights are initialized with the inverse document frequency (IDF) statistic computed over the *MCTest* corpus.[3] However, we do use a short stopword list for questions. This list nullifies query words such as {*who, what, when, where, how*}, along with conjugations of the verbs *to do* and *to be*.

Following earlier methods, we use a heuristic to improve performance on negation questions (Sachan et al., 2015; Wang et al., 2015). When a question contains the words *which* and *not*, we negate the hypothesis ranking scores so that the minimum becomes the maximum. This heuristic leads to an improvement around 6% on the validation set.

The most important technique for training the model was the *training wheels* approach. Without this, training was not effective at all (see the ablation study in Table 2). The identity initialization requires that the network weight matrices are square ($d = D$).

We found *dropout* (Srivastava et al., 2014) to be particularly effective at improving generalization from the training to the test set, and used 0.5 as the dropout probability. Dropout occurs after all neural-network transformations, if those transformations are allowed to change with training. Our best performing model held networks at the word-by-word level fixed.

For combining distributed evidence, we used up to trigrams over sentences and our best-performing model reiterated over the top two sentences ($N = 2$).

[3]We override the IDF initialization for words like *not*, which are frequent but highly informative.

We used the *Adam* optimizer with the standard settings (Kingma and Ba, 2014) and a learning rate of 0.003. To determine the best hyperparameters we performed a search over 150 settings based on validation-set accuracy. *MCTest*'s original validation set is too small for reliable hyperparameter tuning, so, following Wang et al. (2015), we merged the training and validation sets of *MCTest*-160 and *MCTest*-500, then split them randomly into a 250-story training set and a 200-story validation set. This repartition of the data did not affect overall performance *per se*; rather, the larger validation set made it easier to choose hyperparameters because validation results were more consistent.

5.3 Results

Table 1 presents the performance of feature-engineered and neural methods on the *MCTest* test set. Accuracy scores are divided among questions whose evidence lies in a single sentence (*single*) and across multiple sentences (*multi*), and among the two variants. Clearly, *MCTest*-160 is easier.

The first three rows represent feature-engineered methods. Richardson et al. (2013) + RTE is the best-performing variant of the original baseline published along with *MCTest*. It uses a lexical sliding window and distance-based measure, augmented with rules for recognizing textual entailment. We described the methods of Sachan et al. (2015) and Wang et al. (2015) in Section 3. On *MCTest*-500, the Parallel Hierarchical model significantly outperforms these methods on *single* questions ($> 2\%$) and slightly outperforms the latter two on *multi* questions ($\approx 0.3\%$) and overall ($\approx 1\%$). The method of Wang et al. (2015) achieves the best overall result on *MCTest*-160. We suspect this is because our neural method suffered from the relative lack of training data.

The last four rows in Table 1 are neural methods that we discussed in Section 3. Performance measures are taken from Yin et al. (2016). Here we see our model outperforming the alternatives by a large margin across the board ($> 15\%$). The Neural Reasoner and the Attentive Reader are large, deep models with hundreds of thousands of parameters, so it is unsurprising that they performed poorly on *MCTest*. The specifically-designed HABCNN fared better, its convolutional architecture cutting down on the parameter count. Because there are similarities between our model and the

Method	*MCTest*-160 accuracy (%)			*MCTest*-500 accuracy (%)		
	Single (112)	Multiple (128)	All	Single (272)	Multiple (328)	All
Richardson et al. (2013) + RTE	76.78	62.50	69.16	68.01	59.45	63.33
Sachan et al. (2015)	-	-	-	67.65	67.99	67.83
Wang et al. (2015)	**84.22**	67.85	**75.27**	72.05	67.94	69.94
Attentive Reader	48.1	44.7	46.3	44.4	39.5	41.9
Neural Reasoner	48.4	46.8	47.6	45.7	45.6	45.6
HABCNN-TE	63.3	62.9	63.1	54.2	51.7	52.9
Parallel-Hierarchical	79.46	**70.31**	74.58	**74.26**	**68.29**	**71.00**

Table 1: Experimental results on *MCTest*.

Ablated component	Validation accuracy (%)
-	**70.13**
n-gram	66.25
Top N	66.63
Sentential	65.00
SW-sequential	68.88
SW-dependency	69.75
Word weights	66.88
Trainable embeddings	63.50
Training wheels	34.75

Table 2: Ablation study on *MCTest*-500 (all).

HABCNN, we hypothesize that the performance difference is attributable to the greater simplicity of our model and our *training wheels* methodology.

6 Analysis and Discussion

We measure the contribution of each component of the model by ablating it. Results on the validation set are given in Table 2. Not surprisingly, the n-gram functionality is important, contributing almost 4% accuracy improvement. Without this, the model has almost no means for synthesizing distributed evidence. The top N function contributes similarly to the overall performance, suggesting that there is a nonnegligible number of *multi* questions that have their evidence distributed across noncontiguous sentences. Ablating the sentential component made a significant difference, reducing performance by about 5%. Simple word-by-word matching is obviously useful on *MCTest*. The sequential sliding window contributes about 1.3%, suggesting that word-distance measures are not overly important. Similarly, the dependency-based sliding window makes a very minor contribution. We found this surprising. It may be that linearization of the dependency graph removes too much of its information. The exogenous word weights make a significant contribution of over 3%. Allowing the embeddings to change with training reduced performance fairly significantly, almost 8%. As discussed, this is a case of having too many parameters for the available training data. Finally, we see that the training wheels methodology had enormous impact. Without heuristic-based initialization of the model's various weight matrices, accuracy goes down to about 35%, which is only ten points over random chance.

Analysis reveals that most of our system's test failures occur on questions about quantity (*e.g.*, *How many...?*) and temporal order (*e.g.*, *Who was invited last?*). Quantity questions make up 9.5% of our errors on the validation set, while order questions make up 10.3%. This weakness is not unexpected, since our architecture lacks any capacity for counting or tracking temporal order. Incorporating mechanisms for these forms of reasoning is a priority for future work (in contrast, the Memory Network model (Weston et al., 2014) is quite good at temporal reasoning).

The Parallel-Hierarchical model is simple. It does no complex language or sequence modeling. Its simplicity is a response to the limited data of *MCTest*. Nevertheless, the model achieves state-of-the-art results on the *multi* questions, which (putatively) require some limited reasoning. Our model is able to handle them reasonably well just by stringing important sentences together. Thus, the model imitates reasoning with a heuristic. This suggests that, to learn true reasoning abilities, *MCTest* is too simple a dataset—and it is almost certainly too small for this goal.

However, it may be that human language processing can be factored into separate processes of *comprehension* and *reasoning*. If so, the Parallel-Hierarchical model is a good start on the former. Indeed, if we train the method exclusively on *single* questions then its results become even more impressive: we can achieve a test accuracy of **79.1%** on *MCTest*-500. Note that this boost in performance comes from training on only about *half* the data. The 'single' questions can be con-

sidered a close analogue of the RTE task, at which our model becomes very adept even with less data.

Incorporating the various views of our model amounts to encoding prior knowledge about the problem structure. This is similar to the purpose of feature engineering, except that the views can be fully trained. Encoding problem structure into the structure of neural networks is not new: as another example, the convolutional architecture has led to large gains in vision tasks.

7 Conclusion

We have presented the novel Parallel-Hierarchical model for machine comprehension, and evaluated it on the small but complex *MCTest*. Our model achieves state-of-the-art results, outperforming several feature-engineered and neural approaches.

Working with our model has emphasized to us the following (not necessarily novel) concepts, which we record here to promote further empirical validation.

- Good comprehension of language is supported by hierarchical levels of understanding (*cf.* Hill et al. (2015)).
- Exogenous attention (the trainable word weights) may be broadly helpful for NLP.
- The *training wheels* approach, that is, initializing neural networks to perform sensible heuristics, appears helpful for small datasets.
- *Reasoning* over language is challenging, but easily simulated in some cases.

References

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

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, pages 932–938.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *In Proc. of SciPy*.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

François Chollet. 2015. keras. `https://github.com/fchollet/keras`.

Silviu Cucerzan and Eugene Agichtein. 2005. Factoid question answering over unstructured and structured web content. In *TREC*, volume 72, page 90.

Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*, volume 3. JHU Press.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

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 1684–1692.

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*.

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

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Andrew R Mayer, Jill M Dorflinger, Stephen M Rao, and Michael Seidenberg. 2004. Neural networks underlying endogenous and exogenous visual–spatial orienting. *Neuroimage*, 23(2):534–541.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Karthik Narasimhan and Regina Barzilay. 2015. Machine comprehension with discourse relations. In *53rd Annual Meeting of the Association for Computational Linguistics*.

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

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 1, page 2.

Mrinmaya Sachan, Avinava Dubey, Eric P Xing, and Matthew Richardson. 2015. Learning answerentailing structures for machine comprehension. In *Proceedings of ACL*.

Ellery Smith, Nicola Greco, Matko Bosnjak, and Andreas Vlachos. 2015. A strong lexical matching method for the machine comprehension test. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1693–1698, Lisbon, Portugal, September. Association for Computational Linguistics.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439.

Shuohang Wang and Jing Jiang. 2015. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*.

Hai Wang, Mohit Bansal, Kevin Gimpel, and David McAllester. 2015. Machine comprehension with syntax, frames, and semantics. In *Proceedings of ACL, Volume 2: Short Papers*, page 700.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916*.

Wenpeng Yin, Sebastian Ebert, and Hinrich Schütze. 2016. Attention-based convolutional neural network for machine comprehension. *arXiv preprint arXiv:1602.04341*.