Ensemble Learning For Machine Comprehension: Bidirectional Attention Flow Models

William Du, willadu@stanford.edu
Stephen Ou, sdou@stanford.edu
Divya Saini, sainid@stanford.edu

March 22, 2017

Abstract

In this paper, we will explore machine comprehension in Stanford Question and Answering Dataset using ensembled deep recurrent neural networks with bi-directional attention flow. Given a context paragraph, we attempt to answer a query related to the context paragraph. This requires use to not only generate knowledge representation for each question and paragraph, but also create mechanisms that explore attention between the questions and paragraphs. In this paper, we use bi-directional attention flow networks that use bi-directional long short-term memory recurrent neural networks to help represent the context, the questions and their interactions at multiple levels of granularity. Our best ensemble model achieves 63.748 F1 and 52.507 EM scores on the development set and 64.41 F1 and 53.498 EM scores on the test set, all published on CodaLab.

1 Introduction

Machine Comprehension is a studied task in both natural language processing and artificial intelligence research. Intuitively, strong reading comprehension involves both the interpretation of a text and the interpretation of a related question in a way that enables a machine to make complex inferences regarding a relevant answer.

In this task, we set out to build an end-to-end neural network that has the ability to, fairly accurately and precisely, conquer the question-answering task on the Stanford Question and Answering Dataset (SQuAD). We define the act of answering a question as the process of predicting an answer span within the given context paragraph for the relevantly provided question.

Overall, our goal for the project was to: predict a span of words within the context paragraph that represents an answer to the provided question. Our span is indicated by a prediction of a start word and an end word that delineate the spanned answer, a subset of the full context paragraph.

1.1 Dataset

SQuAD is comprised of around 100K question-answer pairs, along with a context paragraph. The context paragraphs were extracted from a set of articles from Wikipedia. Humans generated questions using that paragraph as a context, and selected a span from the same paragraph as the target answer. In our model we train using 8132 samples and validate using 4284 samples. We preprocess questions to trim and pad them to lengths of 30, and do the same with contexts to length 300. Lengths of 30 and 300 were chosen based on the distribution of question and context lengths (Figure 1).

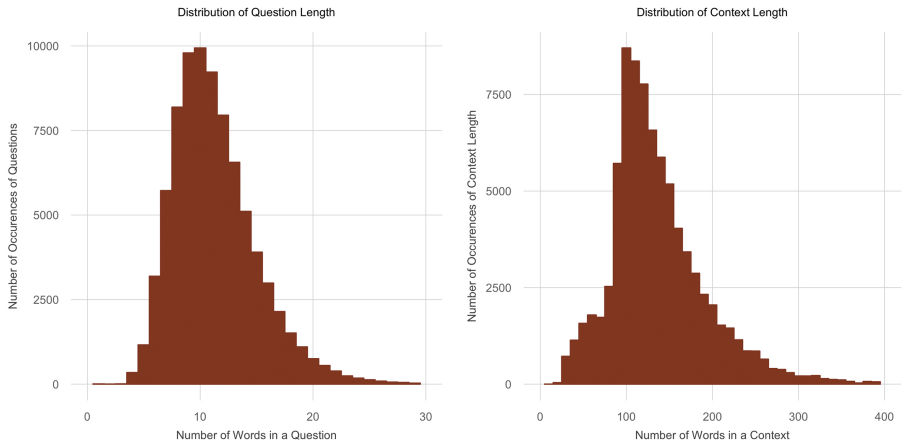

Figure 1: Distribution of question and context lengths on the train dataset. This distribution is used to inform the question cutoff length of 30, and the context cutoff length of 300.

2 Background and Related Work

The initial baseline model, a simplified version of the Match LSTM model described by Wang and Jiang in "Machine Comprehension Using Match-LSTM And Answer Pointer" was used to get acquainted with the dataset and the problem. The simplified description of the model is as follows: a Bi-LSTM was run over the question, and the two end hidden vectors were concatenated to create a representation of the question. This was also done over the context paragraph, conditioned on this question representation. For the attention step, we calculate a vector over the context paragraph representation based on the question representation and conglomerate it with each context paragraph position to derive a new vector for each position.

As the baseline of our more advanced model, we started with an implementation of the BiDAF methodology described by Seo et al. in "Bi-Directional Attention Flow For Machine Comprehension"[1]. We implemented the word em-

bedding layer, context embedding layer, and attention flow as described. We did not, however, choose to implement the character-level embedding of the design.

Instead of character-level embeddings, we focused efforts on exploring in a scope beyond the BiDAF model. These explorations are further described below, but it is important to note that the filtering exploration was partially inspired by Wang et al. and their model described in "Multi-Perspective Context Matching for Machine Comprehension"[3]. The high-level intuition behind the filtering layer we were inspired by the group to implement, is that, often, only a small piece of the passage is needed to answer the question and it could make sense to define an additional layer to filter out redundant information from the passage.

3 Approach

Our question-answering model is called the bidirectional attention flow model. The five layers are word embedding layer, encoding layer, attention flow layer, modeling layer, and decoding layer.

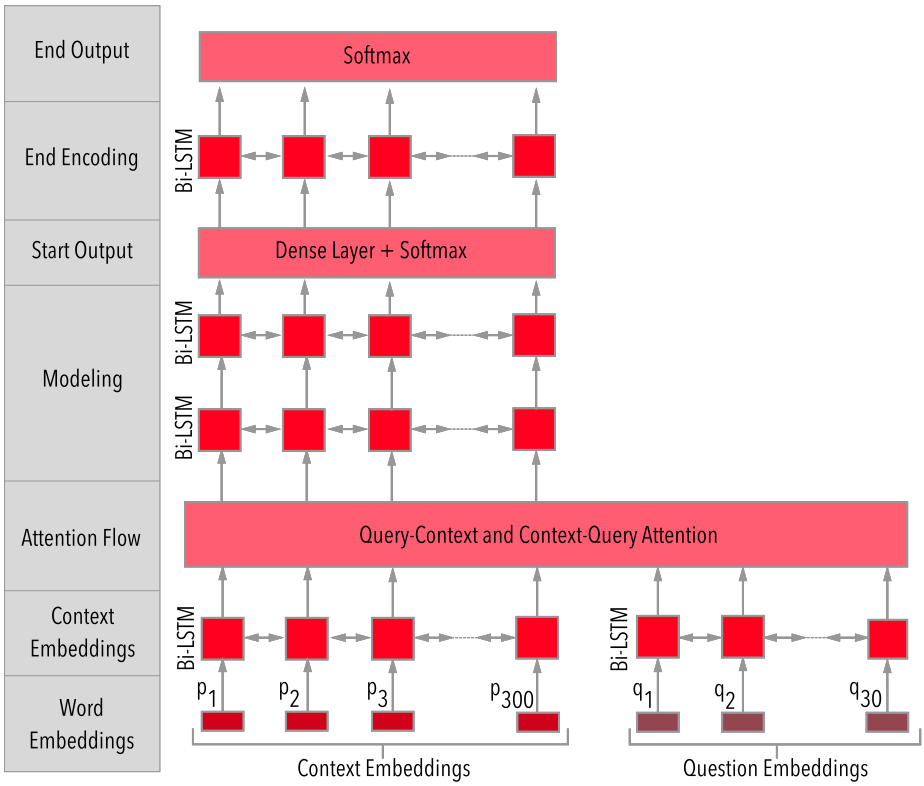

Figure 2: Graph of the machine comprehension model

Before we describe each layer, let's establish notations. Let p be the number of words in a paragraph. Let q be the number of words in a question. Let h be the number of hidden states. Let d be the dimension of the word embeddings.

3.1 Word Embedding Layer

The goal of this word embedding layer is to obtain high-dimensional vector representation of the meaning of the words in the questions and the paragraphs. For example, two words with similar meanings, such as dog and puppy, will have similar vector mapping.

In order to get accurate vector representation of words, we use GLoVe[4], which is an industry standard of pretrained word embeddings. In particular, these pretrained word embeddings are available in many dimensions - 50, 100, 200, and 300. We end up using $d = 300$ dimensions for the pretrained word embeddings.

As a result, we obtain two matrices, $Q \in \mathbb{R}^{d \times q}$ representing the question embeddings, and $P \in \mathbb{R}^{d \times p}$ representing the paragraph embeddings.

3.2 Encoding Layer

We apply a bidirectional Long Short-Term Memory Network (LSTM) for the question embeddings. At each time step t, we feed in the the word embedding for question position t. Similarly, we apply the same LSTM on the paragraph embeddings. It's important to note that we share the weights between the question bidirectional LSTM and the paragraph bidirectional LSTM. In addition, we feed in the last hidden state of the question bidirectional LSTM as the initial hidden state of the paragraph bidirectional LSTM.

The goal of this encoding layer is to capture the relationship between each word and its surrounding words, for both the question and paragraph. The LSTM we choose is bidirectional because we want to reflect the interaction of neighboring words that come both before and after the current word.

As a result, we obtain two matrices, $Q' \in \mathbb{R}^{2h \times q}$ representing the question meaning, and $P' \in \mathbb{R}^{2h \times p}$ representing the paragraph meaning.

3.3 Attention Flow Layer

The purpose of the attention flow layer is to obtain two attention matrices, query-context attention and context-query attention. In query-context attention, we want to capture which question word are the most relevant to each paragraph word. In context-query attention, we want to capture which paragraph word are the most relevant to each question word. Both of these attention matrices are crucial at figuring out which paragraph word and which question word should the model pay higher attention to.

The input of the attention flow layer is Q' and P', and the output of the attention flow layer is $G \in \mathbb{R}^{8h \times p}$.

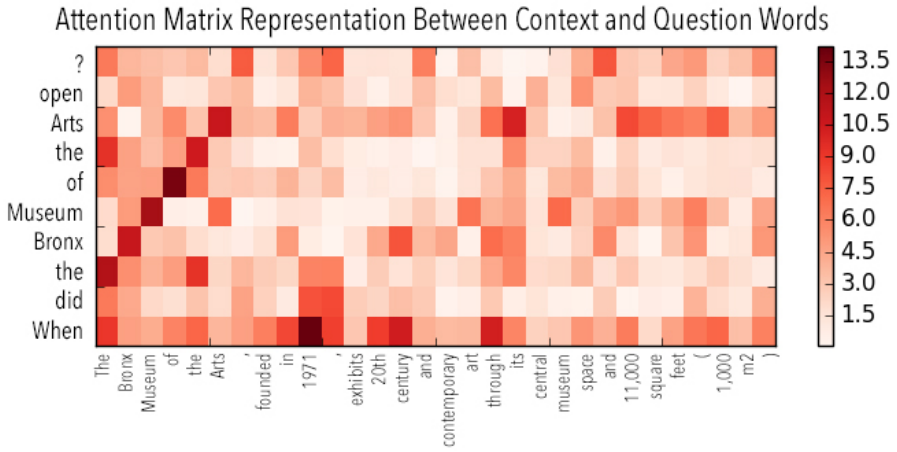

Figure 3: Attention matrix representation for a sample question, answer pair. The heatmap below depicts the attention matrix for each question word, as a representation of the words emphasized in the paragraph for each word in the question. Each row is a question word and each column in a paragraph word.

3.4 Modeling Layer

In the modeling layer, we take our representation of the paragraph words that incorporated attention and output the interaction among the paragraph words for all the questions. Implementation wise, we use a two-layer stacked bidirectional LSTM. The input for the modeling layer is the output from the attention layer A. And the output of the modeling layer is $M \in \mathbb{R}^{2d \times p}$.

3.5 Decoding Layer

In the decoding layer, we apply the linear function $W_{p_s}^T[A; M]$ with a trainable weight W_{p_s} and then apply the softmax function to it in order to get a probability distribution of each paragraph position being the start index.

Before we decode for the end index, we apply another layer of bidirectional LSTM for M to obtain M'. This helps condition the end based on the start predictions. Now, just like before, we apply the linear function $W_{p_e}^T[A; M']$ with a trainable weight W_{p_e} and then apply the softmax function to it in order to get a probability distribution of each paragraph position being the end index.

3.6 Training with Parameter Searching

For training, we use cross entropy as our loss function. In particular, we get the predicted probability for the start and end index, and take the cross entropy loss with the one-hot vector where the correct index has a probability of 1 and the incorrect indices have a probability of 0.

The optimizer we use is stochastic gradient descent. The learning rate we use is 0.5. The initializer we use is Xavier initialization with normal distribution. The question size is capped at 60. The paragraph size is capped at 60. The answer size is capped at 30. We did not include dropout in the model because the performance was worse when we incorporated dropout.

3.7 Testing with Answer Extraction

Once we obtain the probability of each word in the paragraph being the predicted start and end index, we will extract the actual answer. We perform a search of all pairs of start index a_s and end index a_e such that $a_s \leq a_e$ and $a_e - a_s \leq \alpha$, where α is a threshold for maximum answer size, and we pick the pair (a_s, a_e) with the highest probability product $a_s \cdot a_e$.

3.8 Ensemble Learning

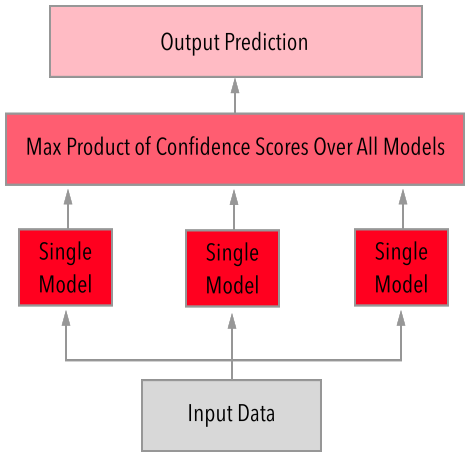

Figure 4: Graph of ensemble combination of single models

Additionally, we use ensemble learning on top of our bidirectional attention flow model. During training, we optimize each of our bidirectional attention flow model with different randomized initialization. During testing, we feed the input into all our trained models. Besides outputting the predicted start and end index, our model also returns a confidence score that is the product of the probability of the start index and the probability of the end index. Then, we will return the answer with the highest confidence score as the overall answer.

4 Experiments

In our process, we balanced both exploration and refinement of our top models. First, we will explain the different models that were explored that were not included in the final model. These explorations are both in intrinsic model structure but also hyperparameter space searching (Figure 1).

As previously discussed, before implementing coattention, we created the baseline model, simplified from the Match-LSTM paper. Then after implementing the bi-directional attention flow model, we explored the use of both stochastic gradient descent (SGD) and ADAM as optimizers, ultimately with SGD outperforming ADAM. We propose that SGD outperformed ADAM because it allowed the model to escape quickly from local minima and allowed it to converge on the global minima. Then, we implemented filtering from "Multi-Perspective Context Matching for Machine Comprehension,"[3] as a prepossessing step before feeding the word embeddings into our first contextual embedding bi-LSTM. We believed this would improve model performance by increasing the weight of highly relevant words. After filtering proved to be less successful than the baseline bidirectional attention flow model, we explored the hyperparameter space searching for optimal learning rates and dropout rates.

Table 1: Exploration Models: Models trained and tested on validation dataset.

MODEL	**LEARN RATE**	**OPTIMIZER**	**F1**	**EM**
Attention Baseline	0.01	ADAM	12.03	7.42
BiDAF	0.001	ADAM	21.51	11.16
BiDAF Filtering	0.5	SGD	47.80	32.20
BiDAF Dropout	0.5	SGD	53.88	37.59
BiDAF	2.0	SGD	52.07	36.40

Next we will discuss the results from our model with the 6B GLoVe[4] vectors. Below in table 2 are the F1 and EM scores on the validation set for the 3 single models at their best 2 epochs. These 6 models were then used to build an ensemble model that was our first submission to codalab. These single models were the bi-directional attention flow model described above, trained with stochastic gradient descent, with no dropout, and with a learning rate of 0.5 and vocabulary of word embeddings from the 6B Wikipedia GLoVe dataset[4] trimmed to the vocabulary of our train and validation set. For this model, the dev set performance for the ensemble model was 62.37, F1, and 50.67, EM. **The test set performance for the ensemble model was 63.17, F1, and 52.73, EM.** These results were published on codalab under the username `stephenou`.

Table 2: F1 and EM scores for best Bi-DAF model with 6B GLoVe[4].

MODEL	DATASET	EPOCH	F1	EM
BiDAF 1	VAL	11	60.5	43.6
BiDAF 1	VAL	12	60.2	44.7
BiDAF 2	VAL	13	58.9	43.5
BiDAF 2	VAL	14	59.3	44.4
BiDAF 3	VAL	13	58.7	42.8
BiDAF 3	VAL	14	58	42.7
BiDAF Ensemble	DEV	-	62.3	50.6
BiDAF Ensemble	TEST	-	63.17	52.73

Finally, to squeeze the most performance out of our model, we ran the same process and model using the 840B Common Crawl GLoVe vectors[4] trimmed to the vocabulary of our validation and train sets. This increases the number of word embeddings available, and decreases the number of out of vocabulary words. Below in table 3 are the F1 and EM scores on the validation set for the 2 single models at their best 2 epochs. These 4 models were then used to build an ensemble model that was our second and final submission to codalab. These single models were the bi-directional attention flow model described above, trained with stochastic gradient descent, with no dropout, and with a learning rate of 0.5 and vocabulary of word embeddings from the 840B Common Crawl GLoVe[4] dataset trimmed to the vocabulary of our train and validation set. For this model, the dev set performance for the ensemble model was 63.748 F1, and 52.507 EM. **The test set performance for the ensemble model was 64.41, F1, and 53.498, EM.** These results were published on codalab under the username `stephenou`.

Table 3: F1 and EM scores for best Bi-DAF model with 840B GLoVe[4].

MODEL	DATASET	EPOCH	F1	EM
BiDAF 1	VAL	15	60.08	45.30
BiDAF 1	VAL	16	60.02	44.03
BiDAF 2	VAL	15	61.51	46.15
BiDAF 2	VAL	16	60.82	46.05
BiDAF Ensemble	DEV	-	63.748	52.507
BiDAF Ensemble	TEST	-	64.41	53.498

5 Analysis

Overall we see we achieved reasonably good performance on using our best BiDAF model with the 840B GLoVe vectors. Next we will perform some analyses to understand how our model performed for different types of questions and for different lengths of predictions.

As depicted by Figure 5, our model had a much easier time correctly predicting answers to questions that lend themselves to more quantitative answers. This includes questions in the format of "what date" and "what percentage", which performed at 97.8 and 95.6 F1 scores, respectively, in our validation set. Questions that lend themselves to more qualitative explanation-based answers, such as "what happens", "what was", and "what may", tended to have lower F1 and EM scores.

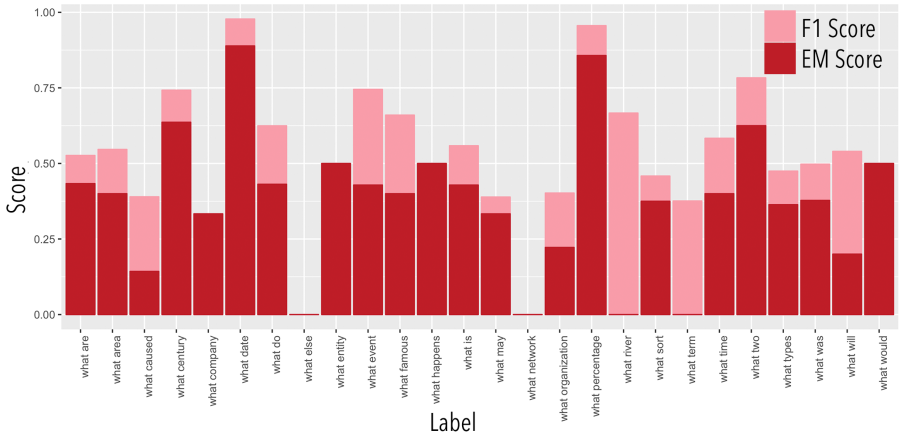

Figure 5: Average performance for different question types. In the development set, what questions that request specific quantitative data as answers, i.e. numbers perform the best, with the highest relative F1 and EM scores.

As depicted by Figure 6A, our model has the highest proportion of exact matches on questions that lend themselves to shorter answer lengths. As the answer lengths become longer, our model becomes less effective at perfectly predicting the answer span. Of all the answer lengths in our validation set, the ones at length one or two see the highest 'exactly matched' proportion, of around .48, while the longest answer lengths around 33 words see 'exactly matched' proportions close to 0.

The question becomes, is there some component in our model that helps increase accuracies on these certain types of questions and decrease on others? The reason our shorter answers were more often perfectly predicted is likely grounded in an understanding of our attention layer's influence. Because we take time to calculate the relevance of each context word to the question words, it is likely that most of the positive relevance is placed on one or a few words for

shorter answers versus across many more words for longer answer spans. This is because for longer answers there are naturally going to be more keywords with potential relevance to question words. Thus, with a greater spread of relevance on context words, it becomes more difficult for the model to as accurately predict the start and end tokens whereas when the relevance is mostly placed on one or a few words, as in shorter answers, there is a greater chance of being able to accurately predict the start and end tokens based on the more concentrated relevance.

The reason quantitative answers were more accurately predicted is likely because quantitative answers are, by nature, shorter than qualitative answers. By the above reasoning, this would increase our model's ability to correctly predict them. The question becomes how we can modify our model to overcome the deficiency in those more convoluted, longer, qualitative answer predictions. We describe thoughts briefly in the next section.

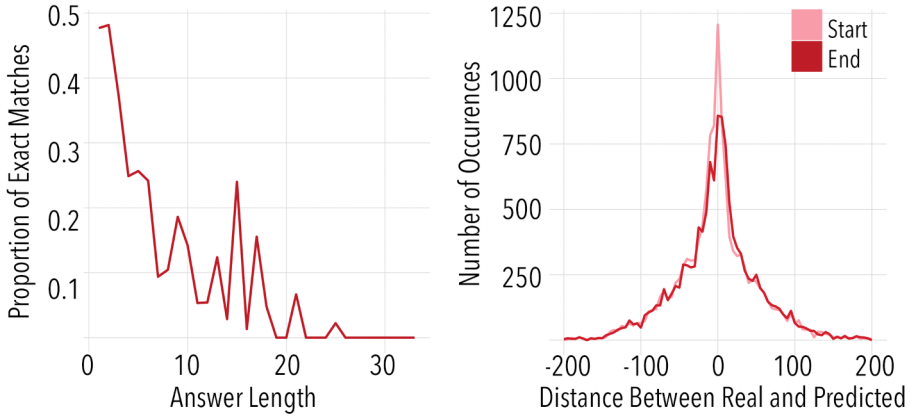

Figure 6: (A) On the left, we have the proportion of perfect answers matches for each length of answer. A perfect answer is defined as the correct start and end tokens identified by the model. (B) On the right, this is the distribution showing the occurrences of distances between the correct index and the predicted index for all the incorrect predictions.

Finally in figure 6B, we note that even when we do not predict a perfect match in start or end indices, the model tends to predict indices that are relatively close to the real value. This indicates perhaps with additional data, hyper-parameter tuning, or additions to the model, we could further improve our EM and F1 scores.

6 Conclusion and Future Directions

In the future, we want to incorporate syntactic structure of the question and the paragraph into our model. One possibility is using dependency pars-

ing to generate range of answers that makes sense grammatically. We could also incorporate character embedding, suggested by the original BiDAF paper. Finally, since the model performs better on quantitative questions than qualitative questions and shorter answers rather than longer answers, we could also try changing the output of the model. Rather than generating the probability of start and end indices, we could generate probabilities for each individual word in regards to whether they are in the answer. Using these probabilities we could extract the longest positively classified contiguous sequence with some stop word exceptions to generate our answers. We believe this would allow the machine comprehension model to better answer more complex answers.

With an F1 score of 64.41 and an EM score of 53.498, we were able to achieve reasonable performance using our bidirectional attention flow model. The performance is only 20% to 25% lower than current state of the art models and within a few percentage points of early models for SQuAD, in particular, the Match-LSTM model. Through our research, we have no only created a reasonably good model and explored potential extensions to the existing bi-directional attention flow model, we have also outlined weaknesses with possibility of improvements that we believe will help improve the model to be better comparable in performance with the current state of the art models.

7 References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi, *Bi-Directional Attention Flow for Machine Comprehension.*
[2] Minjoon Seo, *Bi-directional Attention Flow for Machine Comprehension,* `http://github.com/allenai/bi-att-flow`.
[3] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. *Multi-Perspective Context Matching for Machine Comprehension.*
[4] Jeffrey Pennington, Richard Socher, and Christopher D Manning. *GLoVe: Global vectors for word representation.*
[5] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. *Squad: 100,000+ questions for machine comprehension of text.*