Collective Superintelligence : Recursion, Iteration, AI Join Groups to Improve NLP Interpretations.

Poondru Prithvinath Reddy

May 11, 2020

Collective Superintelligence : Recursion, Iteration, AI Join Groups to Improve NLP Interpretations.

Poondru Prithvinath Reddy

ABSTRACT

If research produces sufficiently intelligent software, it would be able to reprogram and improve itself, and could continue doing so leading to a superintelligence. A superintelligence may be an emulated human with a human-like reasoner requiring long strings of actions. This raises the possibility of collective superintelligence : a large number of separate reasoning systems could act in aggregate with far greater capabilities than any sub-agent. In this paper, we have implemented collective superintelligence which is an ideal future in which Recursion & Iteration models work along side AI for combining the intelligence of different models along with AI to better NLP interpretations for determining the most likely parse tree for a sentence. When we used other methods along with AI which allowed multiple models to work together for achieving improved interpretations beating those of individual models alone. The results from our study shows superior performance of a combined machine and AI augmented model compared to either AI or machine alone. The combined approach shows the feasibility of our method and could harness the best of machine intelligence & artificial intelligence to create a collective superintelligence.

INTRODUCTION

If research into strong AI produced sufficiently intelligent software, it would be able to reprogram and improve itself. It would then be even better at improving itself, and could continue doing so in a rapidly increasing cycle, leading to a superintelligence. This scenario is known as an intelligence explosion. Such an intelligence would not have the limitations of human intellect, and may be able to invent or discover almost anything.

Thus, the simplest example of a superintelligence may be an emulated human mind that's run on much faster hardware than the brain. A human-like reasoner that could think millions of times faster than current humans would have a dominant advantage in most reasoning tasks, particularly ones that require haste or long strings of actions. This also raises the possibility of collective superintelligence : a large enough number of separate reasoning systems, if they communicated and coordinated well enough, could act in aggregate with far greater capabilities than any sub-agent.

The technological singularity – is a hypothetical future point in time at which technological growth becomes called intelligence explosion, an upgradable intelligent agent (such as a computer running software-based artificial general intelligence) will eventually enter a "runaway reaction" of self-improvement cycles, with each new and more intelligent generation appearing more and more rapidly, causing an "explosion" in intelligence and resulting in a powerful superintelligence that qualitatively far surpasses all human intelligence. Chalmers' proportionality thesis hypothesizes that an increase in the capability of creating future systems proportionally increases the intelligence of the resulting system. With this hypothesis, he shows if a process iteratively generates a greater intelligent system using the current system, then this process leads to a phenomenon many refer to as super-intelligence.

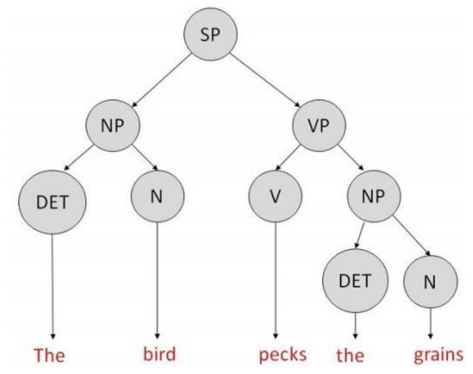
METHODOLOGY

PARSING

Context-Free Grammar

A parser processes input sentences according to the productions of a grammar, and builds one or more constituent structures that conform to the grammar. A grammar is a declarative specification of well-formedness -- it is actually just a string, not a program. A parser is a procedural interpretation of the grammar. It searches through the space of trees licensed by a grammar to find one that has the required sentence along its fringe.

The **parse tree** breaks down the sentence into structured parts so that the computer can easily understand and process it. In order for the **parsing** algorithm to construct this **parse tree**, a set of rewrite rules, which describe what tree structures are legal, need to be constructed.



A parse tree is a really just a "diagrammed" form of a sentence; that sentence could be written in any language, which means that it could adhere to any set of grammatical rules. Also a parse tree is an illustrated, pictorial version of the grammatical structure of a sentence and gives a concrete idea of the syntax of that particular sentence.

Sentence diagramming involves breaking up a single sentence into its smallest, most distinct parts. The diagramming sentences is depending on the grammar and language of a sentence and a parse tree could really be constructed in a multitude of different ways.

The Syntactic analysis examines the structure of a sentence and performs detailed analysis of the sentence. In English language, a sentence S is made up of a noun phrase (NP) and a verb phrase (VP), i.e. S=NP+VP The given noun phrase (NP) normally can have an article or delimiter(D) or an adjective(ADJ) and the noun(N), i.e. NP=D+ADJ+N. Also a noun phrase may have a prepositional phrase(PP) which has a preposition(P), a delimiter (D) and the noun(N),i.e. PP=D+P+N The verb phrase(VP) has a verb(V) and the object of the verb. The object of the verb may be a noun(N) and its determiner, i.e. VP=V+N+D These are some of the rules of the English grammar that helps us to construct a parser for NLP.

In the case of the English language, the smallest "part" of every sentence is a word; words can be combined into phrases, like noun phrases or verb phrases, which can, in turn, be joined with other phrases to create a sentence expression.

Ultimately, the *grammar* and *syntax* of a language — including the way the sentences of that language are structured — become the rules that define the language. Parsing a sentence involves the same mental steps as diagramming a sentence or building a parse tree.

The methodology essentially consist of the following :

- Implementation of Recursive descent parsing
- Implementation of Iterative shift-reduce parsing
- Artificial Intelligence(Recursive Neural Network) implementation
- Building the combined yield using probabilistic model.

ARCHITECTURE

RECURSION

Recursive Descent Parsing

The simplest kind of parser interprets a grammar as a specification of how to break a high-level goal into several lower-level subgoals. The top-level goal is to find an S. The S → NP VP production permits the parser to replace this goal with two subgoals: find an NP, then find a VP. Each of these subgoals can be replaced in turn by sub-sub-goals, using productions that have NP and VP on their left-hand side. Eventually, this expansion process leads to subgoals such as: find the word telescope. Such subgoals can be directly compared against the input sequence, and succeed if the next word is matched. If there is no match the parser must back up and try a different alternative.

The recursive descent parser builds a parse tree during the above process. With the initial goal (find an S), the S root node is created. As the above process recursively expands its goals using the productions of the grammar, the parse tree is extended downwards (hence the name *recursive descent*). We can see this in action using the graphical six stage execution of this parser are shown as below.

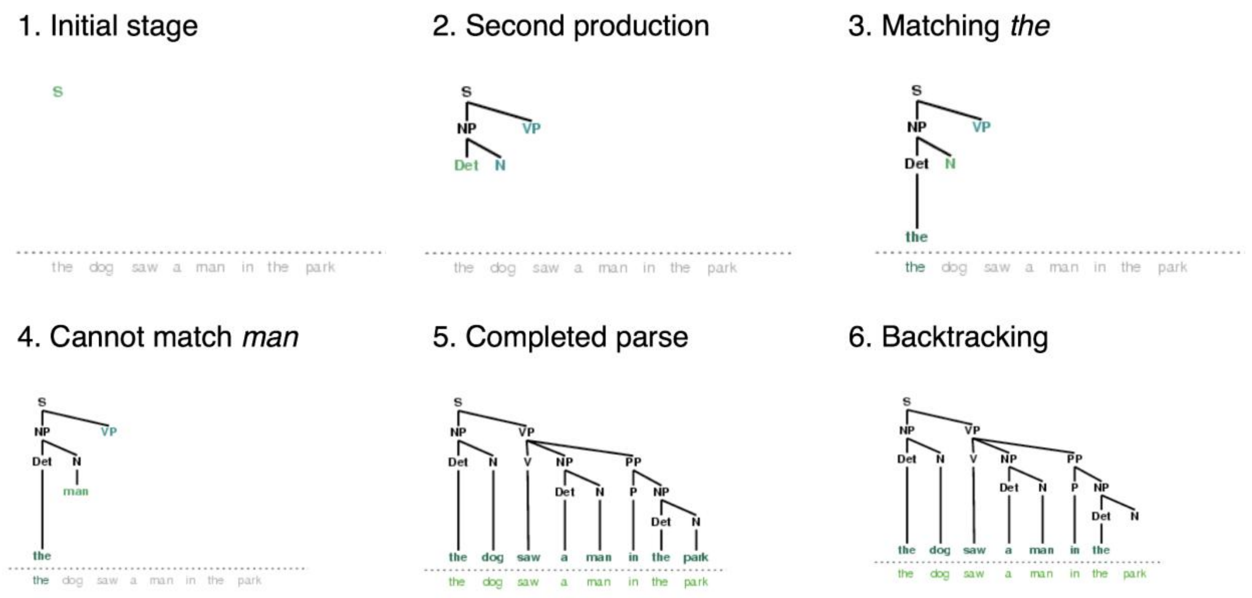


Figure 1: Six Stages of a Recursive Descent Parser: the parser begins with a tree consisting of the node `S`; at each stage it consults the grammar to find a production that can be used to enlarge the tree; when a lexical production is encountered, its word is compared against the input; after a complete parse has been found, the parser backtracks to look for more parses.

During this process, the parser is often forced to choose between several possible productions. For example, in going from step 3 to step 4, it tries to find productions with N on the left-hand side. The first of these is N → man. When this does not work it backtracks, and tries other N productions in order, until it gets to N → dog, which matches the next word in the input sentence. Much later, as shown in step 5, it finds a complete parse. This is a tree that covers the entire sentence, without any dangling edges. Once a parse has been found, we can get the parser to look for additional parses. Again it will backtrack and explore other choices of production in case any of them result in a parse.

We have used Python library NLTK (Natural Language Toolkit) for doing text analysis in English Language. The Natural language toolkit (NLTK) is a collection of Python libraries designed especially for identifying and tag parts of speech found in the text of natural language like English.

The Recursive parser has been implemented in Python using NLTK as it provides a recursive descent parser.

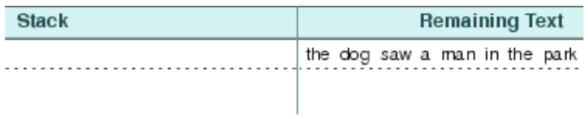
ITERATION(Nonrecursive)

Shift-Reduce Parsing

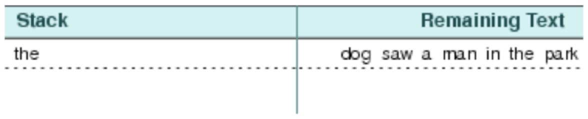
A simple kind of bottom-up parser is the shift-reduce parser. In common with all bottom-up parsers, a shift-reduce parser tries to find sequences of words and phrases that correspond to the *right hand* side of a grammar production, and replace them with the left-hand side, until the whole sentence is reduced to an S.

The shift-reduce parser repeatedly pushes the next input word onto a stack ; this is the shift operation. If the top *n* items on the stack match the *n* items on the right hand side of some production, then they are all popped off the stack, and the item on the left-hand side of the production is pushed on the stack. This replacement of the top *n* items with a single item is the reduce operation. This operation may only be applied to the top of the stack; reducing items lower in the stack must be done before later items are pushed onto the stack. The parser finishes when all the input is consumed and there is only one item remaining on the stack, a parse tree with an S node as its root. The shift-reduce parser builds a parse tree during the above process. Each time it pops *n* items off the stack it combines them into a partial parse tree, and pushes this back on the stack. We can see the shift-reduce parsing algorithm in action using the graphical six stage execution of this parser as shown below.

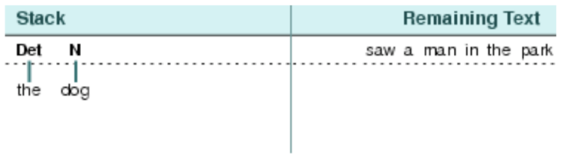
1. Initial state



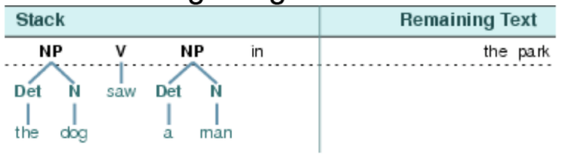
2. After one shift



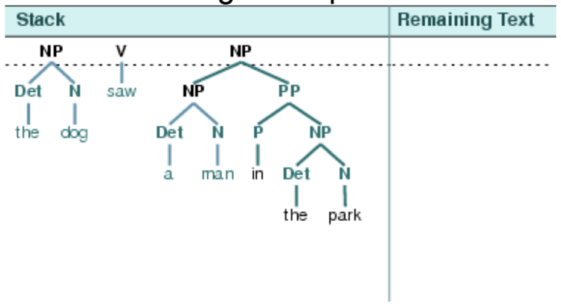
3. After reduce shift reduce



4. After recognizing the second NP



5. After building a complex NP



6. Built a complete parse tree

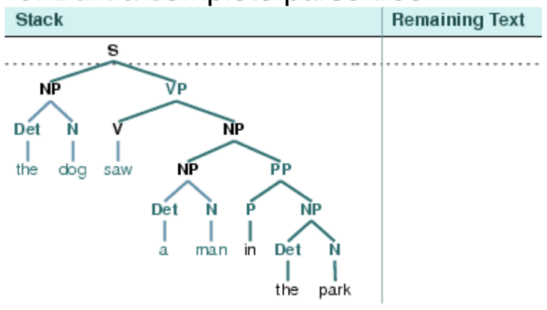


Figure 2: Six Stages of a Shift-Reduce Parser: the parser begins by shifting the first input word onto its stack; once the top items on the stack match the right hand side of a grammar production, they can be replaced with the left hand side of that production; the parser succeeds once all input is consumed and one S item remains on the stack.

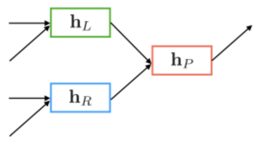
We have used Python library NLTK (Natural Language Toolkit) for doing text analysis in English Language. The Natural language toolkit (NLTK) is a collection of Python libraries designed especially for identifying and tag parts of speech found in the text of natural language like English.

The Nonrecursive parser has been implemented in Python using NLTK as it provides ShiftReduceParser(), an implementation of a shift-reduce parser. This parser does not implement any backtracking, so it is not guaranteed to find a parse for a text, even if one exists. Furthermore, it will only find at most one parse, even if more parses exist.

ARTIFICIAL INTELLIGENCE(Recursive Neural Network)

Recursive Neural Network is one of Recurrent Neural Networks that extended to a tree structure. As both networks are often written as RNN, but in natural language processing it sometimes refers to the Recursive Neural Network. Recursive Neural Network uses a tree structure with a fixed number of branches. In the case of a binary tree, the hidden state vector of the current node is computed from the hidden state vectors of the left and right child nodes, as follows:

$$\mathbf{h}_P = a\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_L\\ \mathbf{h}_R\end{bmatrix} + \mathbf{b}\right)$$



Recursive Neural Networks are natural mechanisms to model sequential data. This is so because language could be seen as a recursive structure where words and sub-phrases compose other higher-level phrases in a hierarchy. In such structure, a non-terminal node is represented by the representation of all its children nodes. The figure 3 below illustrates a simple recursive neural network .

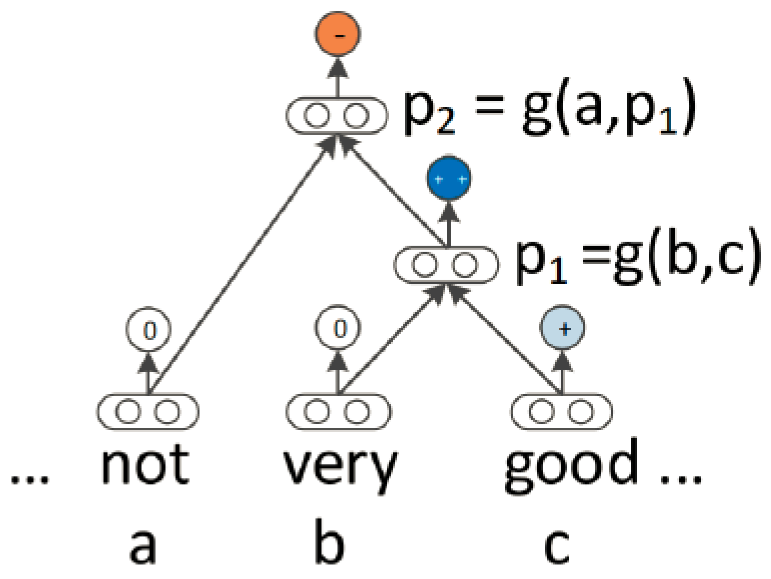


Figure 3: Recursive Neural Network

In the basic recursive neural network form, a compositional function (i.e., network) combines constituents in a bottom-up approach to compute the representation of higher-level phrases (see figure above).

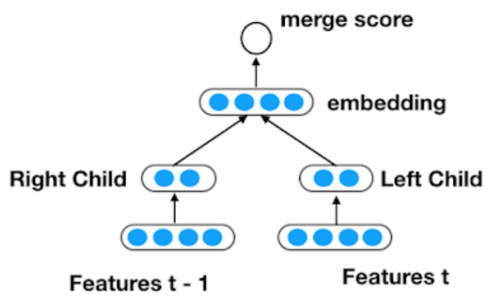
In the standard recursive neural network, It is first determined which parent already has all its children computed. In the above figure, *p1* has its two children's vectors since both are words. RNNs use the following equations to compute the parent vectors:

$p_1 = f(W\begin{bmatrix} b \\ c \end{bmatrix})$, $p_2 = f(W\begin{bmatrix} a \\ p1 \end{bmatrix})$,

where f = tanh is a standard element wise nonlinearity, $\mathbf{W \in R^{dx2d}}$ is the main parameter to learn and we omit the bias for simplicity. The parent vectors must be of the same dimensionality to be recursively compatible and be used as input to the next composition. Each parent vector *pi*, is given to the same softmax classifier of Eq.(y^a = softmax($_{Ws}a$), where $Ws \in R^{5xd}$ is the sentiment classification matrix) to compute its label probabilities.

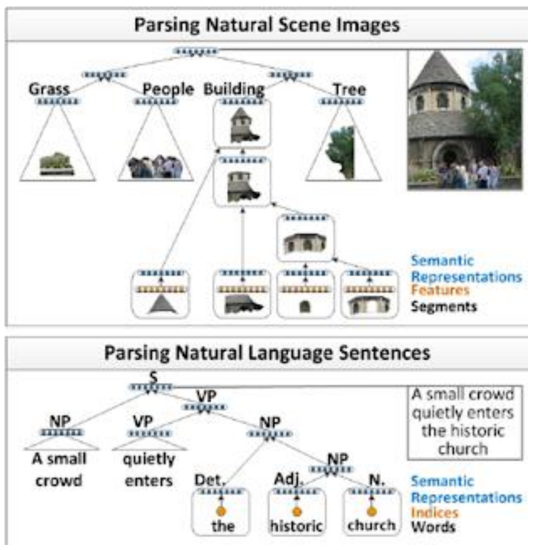
Recursive neural networks are used for various applications and in this paper, we use them for Parsing.

Recursive Neural Networks with merging score is given as below :



A Recursive Neural Network

Recursively Neural Networks can learn to structure data such as text or images hierarchically. The network can be seen to work similar to a context free grammar and the output is a parse t. The figure below shows examples of parse trees for an image (top) and a sentence (bottom).



A recursive neural network for sequences always merges two consecutive input vectors and predicts if these two vectors can be merged. If so, we replace the two vectors with best merging score with the hidden vector responsible for the prediction and this is done in a recursive manner by constructing a parse tree. We have used a recursive neural network which is replicated for each pair of possible input vectors. We predict parse trees and compute their scores with RNN, and this network is a different RNN formulation in that it predicts a score for being a correct merging decision.

Formally we construct the network as follows :

Given an input sequence $x = x_1 \ldots x_T,\ x_i \in \mathbb{R}^D$ then for two neighboring inputs x_t, x_{t+1} we predict a hidden representation:

$$h_t = \sigma(x_t W_l + b_l)$$
$$h_{t+1} = \sigma(x_{t+1} W_r + b_r)$$

The hidden layer used for replacement is computed from the concatenation of the two previous hidden layers: $h = \sigma([h_t, h_{t+1}] W_h + b_h)$. Finally, the prediction is simply another layer predicting the merging score.

As a part of the implementation, we first defined node class which also handles the parsing, by greedily merging up to a tree. A node in the tree holds it's representation as well as a left and a right child. In other words, each node represents a merging decision with the two children being the nodes merged and the parent representation being the hidden layer in the merger grammar. Parsing a tree involves merging the vectors with the highest score and replacing the nodes with their parent node. Also as a part of the implementation, we defined the neural network with the merging decision. Given a node with the parent representations (for leaf nodes that are the input vectors) we build the hidden representations and predict the label from the hidden representation, and then compute the merging score.

We have used a text file containing sentences for training the recursive neural network and during learning, we collected the scores of the subtrees and used labels to decide if a merge was correct or not. Further, training is done with an aim to increase scores of segment pairs with the same label and decrease scores of pairs with different labels, unless no more pairs with the same labels are left.

The Recursive Neural Network has been implemented in Python using PyTorch.

BUILDING A COMBINED YIELD

We need to build a combined yield network for achieving Collective Superintelligence.

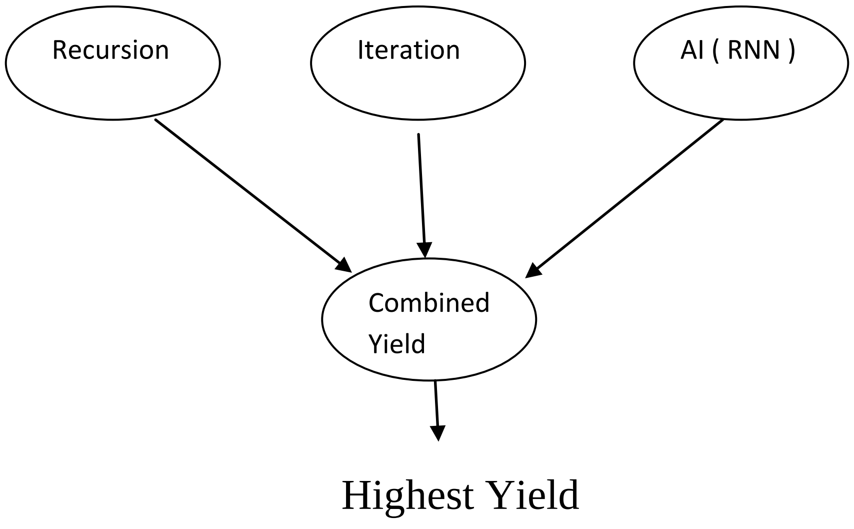
For this, we need to identify variables i.e. which nodes to represent, what values can they take and in which state can they be ?

Let us consider nodes, with only probable values and the variable must take on exactly one of those values at a time.

The types of probable node are :-

- Recursion – explores options that will not lead to a full parse, but can explore many options that never connect to actual sentence. Given a sentence, recursion (Top Down Parsing) explores 7 to 8 different options and one of which may connect to the actual sentence. Hence a probability of about 0.15 for getting the actual sentence.
- Iteration – explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse. Given a sentence, iteration (Bottom Up Parsing) explores 7 to 8 different options and one of which may connect to the actual sentence. Hence a probability of about 0.15 for getting the actual sentence.
- AI (Recursive Neural Network) – since there are no options and there is only method which leads to a full parse with an accuracy of 20%, hence a probability of 0.20.

The topology of the combined network should capture relationships between nodes and variables. For example, what are the different methods of parsing a sentence i.e. Recursion, Iteration & AI, and then add arcs from nodes to combined yield node to get highest outcome.



The set of parent nodes of a node X is given by Parent(X). The Combined-Yield node has three parents (Recursion, Iteration & AI) and node Recursion is an ancestor of node combined-yield and successor of nodes Iteration and AI.

We used statistical parsing which uses a probability model of syntax in order to assign probabilities to each parse tree and provide approach to resolving syntactic ambiguity. A PCPG (Probabilistic Context Free Grammer) is a probabilistic version of a CFG where each production has a probability. The probability of a sentence is the sum of the probabilities of all of its derivations i.e.

$$\begin{aligned} P(\text{Sentence}) &= P(\text{Rec.}) + P(\text{Ite.}) + P(\text{AI}) \\ &= 0.15 + 0.15 + 0.20 \\ &= 0.50 \end{aligned}$$

RESULTS

The recursion i.e. Top Down approach is not highly precise as parser takes it as a correct sentence even if it makes no sense. Similarly Top Down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.

The Iteration i.e. Bottom Up is also not highly precise as it can be used to parse natural language but produce many spurious parses. In addition, Bottom Up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.

The initial experiments showed that the AI (Recursive Neural Network) model worked with lower accuracy(20%) at the sentence level. This is due to the fact that the learning was performed with smaller dataset and also performance varied at larger or smaller phrases and batch sizes.

The Combined Yield Network with overall outcome of 50% which uses a probability model by assigning probabilities to each parse tree of all of its derivatives and this yielded highest combined performance to either Recursion or Iteration alone.

CONCLUSION

A superintelligence may be an emulated human with a human – like reasoner requiring long strings of actions. This raises the possibility of collective superintelligence ; a large number of separate reasoning systems act in aggregate with far greater capabilities than any sub-agent. We have implemented collective superintelligence in which Recursion & Iteration models work along side AI to better NLP interpretations. Results show superior performance of a combined machine and AI augmented model compared to either AI or machine alone, and the combined approach could harness the best of machine intelligence and artificial intelligence to create a collective superintelligence.

REFERENCES

1. https://www.nltk.org/book/ch08.html

2. Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank; Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng and Christopher Potts; 2013; Stanford University.
3. Parsing Natural Scenes and Natural Language with Recursive Neural Networks; Richard Socherrichard, Cliff Chiung-Yu Linchiungyu, Andrew Y. Ngang, Christopher D. Manningmanning; Stanford University.