Overview

Natural Language Processing

NLP is the subdomain of artificial intelligence concerned with the "word", ranging from voice input through to speech output.

All of it is deeply hard research stuff and the core is a focus on text words, ranging from sentences to paragraphs to documents. The analysis of the word ranges from determining frequencies of word use and collocations of words (e.g., for document content classification, document similarity comparison, and sentiment analysis) to pattern matching, word stemming, and part-of-speech tagging and parsing (for trying to access part of the meaning of text for information retrieval systems, machine summarization, machine translation, and natural language command and control over programs and devices). Keyword search (e.g., Google or Amazon) is a simplified use of NLP which does not involve meaning to find information or products.

Given its historical research component, it has been primarily developed by academics for academics. Major NLP tools have been written in LISP, Java, C++, Python, and Perl and run on either Windows or Linux (Mac is a Linux) or both. There are rarely any performance requirements in cpu time or memory and they don't run on mobile devices. The only area of NLP that actually has real-time requirements and are primarily engineered by the business world are chatbots, which either answer questions (e.g. Siri) or converse on either their own life story (entertainment) or on corporate information (customer help agents). They need to reply within seconds of the user's input.

Video Game Overview

Video games are a completely unrelated industry. The traditional AAA game takes several years to develop using a team of 40 people and costs millions to make. Game development has to manage a production pipeline for gazillions of animation and sound assets and deliver them to the screen at a minimum of 30 frames per second (33 milliseconds per frame).

Cpu performance and memory use are always of paramount importance. So are testing and bug reproducability, since millions of customers will be outraged if they play a game for hours and then it freezes or crashes on them. Given those requirements, game engines are not programmed in whatever language pleases the developers. They are programed in C++ for maximum performance.

And game levels are written by less skilled programmers called designers, who write in scripting languages. These are less complete programming languages built on top of the engine, which focus on the common elements needed for writing games. Scripting languages let you write less to achieve more.

ChatScript

So how do these two industries intersect? They intersect in ChatScript because I come to the NLP industry from the videogame industry. I started out in NLP on the chatbot side, where issues of content authoring and real-time performance abound. I have applied all of the concerns and characteristics of the video game industry to creating an engine that can perform in real time and perform on mobile devices.

You probably think of ChatScript as a system just for writing chatbots - for conducting some kind of prescripted conversation. This view is incomplete. ChatScript is also a broad industrial-strength system for a wide range of English NLP tasks. ChatScript is radically different from other NLP tools because it was architected with many considerations in mind: compact authoring expressiveness, speed, size, capability, and maintainability.

Real World Opinions of ChatScript

I know a guy who has worked for over a decade with different commerical customer help agent software systems for business and education, which are a kind of chatbot. He was tired of the industry and was exiting it when ChatScript arrived. Now he has switched over to ChatScript and is reinvigorated. Its performance and capabilities exceed all the systems he is familiar with and he knows all the major players.

The Virtual Patient project at OSU used AIML chatbot technology. They were dying because they had hundreds of thousands of rules that were killing their server's performance and were unmaintainable. Every new rule added tended to break something elsewhere. They switched over to ChatScript and rewrote their system. It takes less than 2,000 rules and they figure their final system will not exceed 5000. Adding new rules without breaking things is easy and the system automatically tells them when they do create conflicts. Their server has no load at all and can run multiple different patient simulations simultaneously.

An example of a high-quality ChatScript bot, Tom Loves Angela that took about 5 person-months to create 16,000 script lines for iOS/Android . It stayed in the top 10 of Apple's AppStore paid Entertainment category for 6 months. It was still in the top 25 iPad apps 18 months after release, even though the extension to 26,000 lines of Talking Angela was also released.

The Holy Grail - Meaning

Understanding meaning is the holy grail of NLP. If you've ever used Google translate on a Japanese website, you'll know that even Google doesn't understand meaning. Most tools have to settle for focusing on words without any meaning. Parser tools get closest, but only when they can successfully parse the sentence.

While no software can understand the meaning of what people may unexpectedly

say, ChatScript - better than any other software in the world - allows you to hunt in the input for a particular meaning, whether the input is parseable or not . For example, if you said *I love dogs except on days of the week starting with a letter* ChatScript would not understand that you don't like dogs (unless we had explicitly written script to hunt for that meaning). But we could write a single brief rule in a pets topic which hunts for the meaning that you own an animal. The user could say I own two poodles or i have a dog or my parrot eats seeds and the system could record what kind of animal you owned and react to it with appropriate chat. And it would not be confused by *I don't own a cat or I would never have a dog* or *I have no pet but my friend owns a dog.*

ChatScript Architecture Objectives

Compact authoring

Because everything is scripted, it means every chatbot is all hand-authored. One could not create a major chatbot automatically. It's a consistent skilled creation. The more script you have to type, the longer it takes to code. ChatScript is both extremely clear and extremely concise. Our chatbots of 12K to 16K ChatScript rules outperform chatbots with hundreds of thousands of their rules. They take years to write theirs; we take months.

Speed

A low-grade server running our most advanced chatbot can generate a response in a few milliseconds; handling thousands of simultaneous users at once. And you can run ChatScript on each core of a machine simultaneously, servering the same port, with near linear scaling, so an 8-core machine can handle well over 10K simultaneous conversations.

As for parsing English sentences, Stanford's parser is the gold standard for such things. But it runs over much slower than we do. Stanford's online server parses at a rate of 28 tokens per second (periods and commas are also tokens). Parsing a serious document would take it a long time. On my basic laptop, ChatScript can scan and parse a document at 30,000 tokens per second. We parse user input and chatbot output as part of the several milliseconds response time merely to automatically handle pronoun resolution for chat.

Memory

Our largest chatbot with Chatscript, parsing, etc, runs on an iPhone in 18Mb of memory. It doesn't matter whether the input sentence is 5 words or 250 words. For Stanford's parser, you can't fit it on a phone. 20 word sentences take 50MB. 50 word sentences take 125MB. And their FAQ says "We have parsed sentences as long as 234 words, but you need lots of RAM and patience."

Maintenance

Because ChatScript is concise, maintaining tens of thousands of rules is tons easier than maintaining hundreds of thousands (which is what other bots have to do). But ChatScript builds in explicit mechanisms to support maintenance. First, you can author things by topics, which are independent of each other and provide logical structured areas for rules on areas of knowledge. This makes it easy for multiple authors to work on a bot simultaneously.

Second, with every rule you associate a sample input- something the user might say that is supposed to trigger that rule. ChatScript can automatically test that the rule would respond to that input and that no other rule anywhere else in the system would collide and try to take control instead. It can generate reports that tell you of problems with your rules. Third, you can hold an ideal conversation with the system and ask it to turn that into a regression test.

You can then run that test anytime in the future to see if the system has degraded. Even if you insert other rules or alter the code of a rule, it can usually tell you whether you are still executing correctly even though the literal output has changed.

Fourth, ChatScript has analytic tools to peruse user logs. It can find user inputs that mention a particular word or words, tell you how many users entered a topic, how often, how deep they got into it, how many times a rule was executed and what users said just before or just after that are likely related to it. You can use these tools to find popular topics to expand, extend reactions to user inputs, and learn what things a topic fails to cover that seem to be frequent with users.

ChatScript can also print out abstracts of the system (what a user might say and what the system would say in return) so that customers can bless or request adjustment of the content without them having to know any ChatScript syntax or having to try out random inputs by themselves (they would never be able to test it all). And it can spell check the outputs. Or if you specify an age level, it can tell you where in the output you wrote there are words that violate vocabulary guidelines for that age group.

ChatScript can also trace execution of the script to controllable levels of detail when you are trying to debug why your script failed. You can trace a kind of activity (like function calls), a topic, etc

ChatScript Capabilities

Finally, it's all about what ChatScript can do easily and what you can do with ChatScript.

Conversation

First thing that comes to mind is controlling conversation. ChatScript is good at finding the rule you wrote to address a specific meaning, so it can appropriately answer users' questions. It is also good at conversational management, taking charge in a topic and leading the user on a conversational journey in a subject. Or taking conversational requests to speak on a subject or drop some subject.

Emotion

ChatScript can classify expressed and implied emotions in user input and react accordingly. We have written bots that get huffy when insulted, respond with put-downs for sexual innuendos, detect when you seem bored with the discussion (based on expressed input or imputed behavior like how long your responses are) and either change topic or directly ask you if you are bored. You can use ChatScript to control gestures of an avatar to express bot emotions simultaneous with verbal output.

User Tracking

ChatScript keeps track of conversations with each user; can record where it is in a conversational flow and what facts it has learned about a user (you have to tell it what facts to try to learn). You can optionally keep logs of the conversations (either on a ChatScript server or locally on a freestanding device).

Multiple Bots

ChatScript can support multiple chatbots at the same time, either all acting independently or all coordinating in something like a theatrical production where multiple bots chat with each other and with the user. If you asked one chatbot a question about his family you'd get one answer, and if you asked a different one the same question you'd get a different answer. And if you asked one a fact question about some topic the other knows, usually the other would answer it instead.

Planning

Apart from mere conversation and avatar gesture control, you can write scripts of how to do things and ChatScript has built-in planner code that can combine them to achieve a goal. You could ask your robot for a ham sandwich and it could decide on a plan to go into the kitchen and make one, including going thru closed or open doors, returning for the key if the door is locked.

Big Data Access

ChatScript can interact with the open source Postgres database. We can store gigabytes of information and access it quickly using NLP. One person used

ChatScript to query a Dun & BradStreet database. We are working on using it to store digested wikipedia information so we can query it instantly.

Document reading

ChatScript has a built in pos-tagger/parser. It can be used to read a document and try to extract information from it. There is even a wikipedia to plain text converter in ChatScript, so you can datamine Wikipedia with appropriate script. We are hoping to make the document reader automatically preliminarily grade papers in massive on-line courses.

Unlike normal NLP tools which are all trained on proper Wall Street Journal input (proper casing, proper speech), ChatScript aims to handle text which is not grammatically correct and which may be in all lower case or all upper case.

These capabilities support a more reliable sentiment analysis.This is just another area of hunting for meaning. Most sentiment analysis merely tallies keywords to decide positive or negative. So *it's not a good movie* is likely to be considered positive.

A bunch of literature talks about using sentence analysis (parsing) to improve sentiment, being able to take into account the not for example. Or a sentence like *I liked it in spite of its horrible acting, bad pacing, and stupid plot.* But there are several issues here. First, something like the Stanford parser is slow. Second, it is hard to work with its output parse tree. Third, being based on statistical probability, it has no clue if it's parse is correct or not. While it's partof-speech tagging of words is 97.3% accurate, its parse is wrong almost half the time.

ChatScript's parser is fast, integrated with ChatScript for easy use, and can generally know if it has found a parse or not because it is based on grammar rules. So if it can't parse something, it can know that and fall back on its more general pattern matching abilities.

Connectivity to other data

ChatScript can interact with the local machine (e.g. robotics control) or the Internet. You can from script request arbitrary programs be run locally and process returned data. Similarly you can use HTTP to request information (current temperature in Singapore, what is the size of the moon) or whatever via the Internet, postprocessing the HTML you get back to pull out the answers.

Improved Search

ChatScript supports defining and manipulating pattern matches on collections of words. This is part of how rules can be so concise. But it also means it can pull apart information in unusual ways.

We can outperform Amazon search. A typical Amazon product keyword search from a user is a series of words. To Amazon the words have no inherent order,

but to the user they do, because they represent a meaning of what they want. If the user says *teal kitchen timer made of aluminum not made in China, costing around $50*, we can decompose that into all the intents the user expressed (including that teal is a hue of the color blue). Amazon search can't handle that.

It can't even handle *teal kitchen timer*. Only the 10th of the ten items found by them is in fact a timer. Since it recognizes it found way too few items, it also displays searches trying to reduce your input. So you see three items from teal kitchen none of which are timers. Then it gives you three items from kitchen timer, which are in fact kitchen timers though none of them are teal or even blue. Then it gives you three items from teal timer, but none of them are kitchen timers.

We wrote a program that took the first page of Amazon search results and reordered the items based on how closely they matched what the user wanted. So we would put that timer first in the list. And when Amazon returns too few matching items, we could tell it to re-search using *blue kitchen timer*, and if that was no good we could have then done *kitchen timer*. We can even handle context-sensitive distinctions like *chocolate pecans* (meaning chocolate flavor) vs *chocolate shirt* (meaning chocolate color).

Conclusion

ChatScript is an ever evolving system, gaining new capabilities every month. You could think of ChatScript as being like the Siri engine on steroids. With enough scripting, one could create a Siri that really could help you with every aspect of your online life. Eventually ChatScript, because is faster, more capable, and takes less memory, will overtake all existing NLP tools and become the world's NLP standard.