Auto-Dialog Systems: Implementing Automatic Conversational Man-Machine Agents by Using Artificial Intelligence & Neural Networks

Annapurna Bala, T.Padmaja, Dr.Guru Kesava Das. Gopisettry
[1,2,]Dept. Of Computer Science, [3]HOD, Dept. Of CSE
[1]Ch.S.D.St Theresas Autonomous College for Women, Eluru, Andhra Pradesh, India,
[2,3]Eluru collegeof Engineering, Eluru, Andhra Pradesh, India.
E-mail: [1]annapurnagandrety@gmail.com

Abstract:
Many companies are hoping to develop bots to have natural conversations indistinguishable from human ones, and many are claiming to be using Neuro-Linguistic Programming and Deep Learning techniques to make this possible. Microsoft is making big bets on chat bots, and so are companies like Facebook (M), Apple (Siri), Google, We Chat, and Slack. There is a new wave of start ups trying to change how consumers interact with services by building consumer apps like Operator or x.ai, bot platforms like Chat fuel, and bot libraries like Howdy's Botkit. Microsoft recently released their own bot developer framework.

Keywords: **chatbots, NLP, DLP, neural networks, AI**

1. Introduction

Conversational Agents use a dialog system to have a conversation with a human. There are a few steps a Conversational Agents goes through to process human information: The first step is converting human input into an understandable context for the Conversational Agents. This is done through input recognizers and decoders, which can analyze speech, text and even gestures. The next step is applying Natural Language Processing to analyze the plain text and search for semantics. All the while the input is managed and processed by a dialog manager to ensure a correct flow of information from and to the participant. The dialog manager also makes sure that questions or issues are assigned to the right task manager and solved. After the tasks are solved the output manager translates the solution into "human like output". This is done through a natural language generator to mimic human speech. The output rendered will then regulate how the output is communicated, e.g. through audio, voice and in a visual format as well. Depending on the status of advancement of the natural language processing and its engines for processing the information, the Conversational Agents is thus able to mimic human language and interpret and communicate efficiently.

2. Machine learning

NLP (Natural language processing) and Machine Learning are both fields in computer science related to AI (Artificial Intelligence). Machine learning can be applied in many different fields. NLP takes care of "understanding" the natural language of the human that the program (e.g. chatbot) is trying to communicate with. This understanding enables the program e.g. chatbot) to both interpret input and produce output in the form of human language. The machine "learns" and uses its algorithms through supervised and unsupervised learning. Supervised learning means to train the machine to translate the input data into a desired output value. In other words, it assigns an inferred function to the data so that newer examples of data will give the same output for that "learned" interpretation. Unsupervised learning means discovering new patterns in the data without any prior information and training. The machine itself assigns an inferred function to the data through careful analysis and extrapolation of patterns from raw data. The layers are for analysing the data in a hierarchical way. This is to extract, with hidden layers, the feature through supervised or unsupervised learning. Hidden layers are part of the data processing layers in a neural network.

3. Existing model, Seq2Seq

Sequence To Sequence model introduced in Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation has since then, become the Go-To model for Dialogue Systems and Machine Translation. It consists of two RNNs (Recurrent Neural Network) : An Encoder and a Decoder. The encoder takes a sequence(sentence) as input and processes one symbol(word) at each timestep. Its objective is to convert a sequence of symbols into a fixed size feature vector that encodes only the important information in the sequence while losing the unnecessary information. You can visualize data flow in the encoder along the time axis, as the flow of local information from one end of the sequence to another. Each hidden state influences the next hidden state and the final hidden state can be seen as the summary of the sequence. This state is called the context or thought vector, as it represents the intention of the sequence. From the context, the decoder generates another sequence, one symbol(word) at a time. Here, at each time step, the decoder is influenced by the context and the previously generated symbols.

Challenges:

The most disturbing one is that the model cannot handle variable length sequences. It is disturbing because almost all the sequence-to-sequence applications, involve variable length sequences. The next one is the vocabulary size. The decoder has to run softmax over a large vocabulary of say 20,000 words, for each word in the output. That is going to slow down the training process, even if your hardware is capable of handling it. Representation of words is of great importance. How do you represent the words in the sequence? Use of one-hot vectors means we need to deal with large sparse vectors due to large vocabulary and there is no semantic meaning to words encoded into one hot vectors.

Proposed Model, Hill-climbing algorithm

Hill climbing is an anytime algorithm: it can return a valid solution even if it's interrupted at any time before it ends. In numerical analysis, hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

4. Hill-climbing algorithm

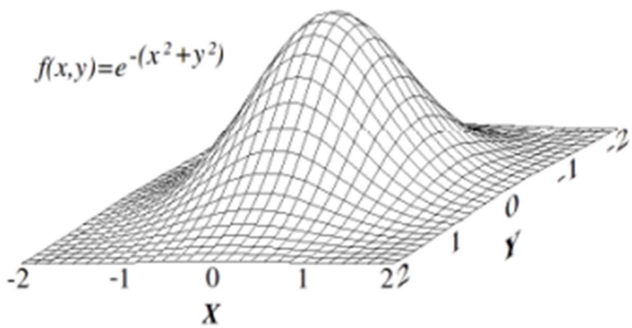
The relative simplicity of the algorithm makes it a popular first choice amongst optimizing algorithms. It is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms. Although more advanced algorithms such as simulated annealing or tabu search may give better results, in some situations hill climbing works just as well. Hill climbing can often produce a better result than other algorithms when the amount of time available to perform a search is limited, such as with real-time systems, so long as a small number of increments typically converges on a good solution (the optimal solution or a close approximation). At the other extreme, bubble sort can be viewed as a hill climbing algorithm (every adjacent element exchange decreases the number of disordered element pairs), yet this approach is far from efficient for even modest N, as the number of exchanges required grows quadratically

5. Mathematical description

Hill climbing attempts to maximize (or minimize) a target function {\displaystyle f(\mathbf {x})} f(\mathbf {x}), where {\displaystyle \mathbf {x} } \mathbf {x} is a vector of continuous and/or discrete values. At each iteration, hill climbing will adjust a single element in {\displaystyle \mathbf {x} } \mathbf {x} and determine whether the change improves the value of {\displaystyle f(\mathbf {x})} f(\mathbf {x}). (Note that this differs from gradient descent methods, which adjust all of the values in {\displaystyle \mathbf {x} } \mathbf {x} at each iteration according to the gradient of the hill.) With hill climbing, any change that improves {\displaystyle f(\mathbf {x})} f(\mathbf {x}) is accepted, and the process continues until no change can be found to improve the value of {\displaystyle f(\mathbf {x})} f(\mathbf {x}). Then {\displaystyle \mathbf {x} } \mathbf {x} is said to be "locally optimal". In discrete vector spaces, each possible value for {\displaystyle \mathbf {x} } \mathbf {x} may be visualized as a vertex in a graph. Hill climbing will follow the graph from vertex to vertex, always locally increasing (or decreasing) the value of {\displaystyle f(\mathbf {x})} f(\mathbf {x}), until a local maximum (or local minimum) {\displaystyle x_{m}} x_{m} is reached.

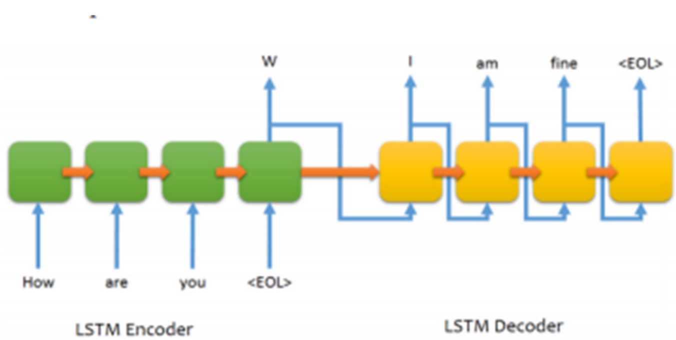
6. Variants

In simple hill climbing, the first closer node is chosen, whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one. Stochastic hill climbing does not examine all neighbors before deciding how to move. Rather, it selects a neighbor at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another. Coordinate descent does a line search along one coordinate direction at the current point in each iteration. Some versions of coordinate descent randomly pick a different coordinate direction each iteration. Random-restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. It is also known as Shotgun hill climbing. It iteratively does hillclimbing, each time with a random initial condition {\displaystyle x_{0}} x_{0}. The best {\displaystyle x_{m}} x_{m} is kept: if a new run of hill climbing produces a better {\displaystyle x_{m}} x_{m} than the stored state, it replaces the stored state. Random-restart hill climbing is a surprisingly effective algorithm in many cases. It turns out that it is often better to spend CPU time exploring the space, than carefully optimizing from an initial condition.



Benefits:

Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management. It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function. It requires much less conditions than other search techniques.



Acknowledgement

Dr.Guru Kesava Das. Gopisettry is the Professor & HOD, in CSE Department, Eluru college of Engineering Eluru. He obtained Doctorol Degree in Engineering Ph.D(CSE) from Acharya Nagarjuna University, in the year 2014. He received his Master's Degree in Engineering, M.E (CSE) from Anna University, Chennai in the year 2008. He completed his Bachelor's Degree in Engineering, B.E (CSE) from Anna University, Chennai. He has a rich experience of 17 years which includes Teaching, Research and Administration. He served various reputed engineering colleges as a HOD for several years. He is a multi-tasking personality and ago getter. His relentless efforts and commitment in Institutional Administration lead few colleges to reach greater heights. He published several technical papers in National and International Journals of repute. He is a certified professional of IBM Rational.

References

[1] Chatbots: An Introduction And Easy Guide To Making Your Own, by Oisin Muldowney
[2] Beginner's guide to build chatbot using api.ai, by Amruta Mohite
[3] https://chatbotsmagazine.com/machine-learning-neuralnetworks-and-algorithms- c0711eb8f9a
[4] https://itnext.io/chatbots-a-bright-future-in-iotf9f6cc8e12a9
[5] https://www.hutoma.ai/
[6] https://chatbotsmagazine.com/the-complete-beginner-sguide-to-chatbots-8280b7b906ca
[7] Applied Artificial Intelligence: Neural networks and deep learning with Python and TensorFlow by Wolfgang Beer
[8] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, "A Neural Network Approach to ContextSensitive Generation of Conversational Responses", NAACL-HLT 2015.
[9] Kaisheng Yao, Geoffrey Zweig, Baolin Peng, "Attention with Intention for a Neural Network Conversation Model", NIPS Workshop on Machine Learning for Spoken Language Understanding and Interaction. 2015.
[10] Sepp Hochreiter; Jürgen Schmidhuber, "Long shortterm memory". Neural Computation. 9 (8): 1735–1780, 1997.