Lucy

1. Introduction

Lucy is an always on cognitive agent observing its environment. She collects data from various sources in its environment and learns to take autonomous actions by finding patterns in data. The environment in this case is a typical home consisting of different rooms. There are sensors attached to various items in the household and they are constantly communicating their state to Lucy . An example communication : a force sensor sending its state to Lucy and she, depending on the time of the day and the location of the sensor, switches on a light. She makes this decision based on past usage patterns. The accompanying code for the project assumes a person waking up from bed in the morning, stepping on a force sensitive mat next to his bed, making a trip to the bathroom and turning on the connected light in the bathroom. Lucy collects this usage data over time and switches on the light on behalf of the user depending on the time of the mat-force-sensor activation.

2. Motivation

After the internet revolution of the 1990's and 2000's when computers and hand-held devices became connected, connectedness of “things” (think cars, hospital beds, library seats etc.) is a logical next step. If the hype created by buzzwords like “Internet of Things (IoT)” and market research estimates are to be believed, IoT is the next BIG thing (same category as King Kong and Hulk). Various market research firms like IDC and Gartner estimate the number of devices that will be internet connected to be between 20 billion and 50 billion by 2020. There isn't much to gain when all these devices are by their own but when they start communicating with each other, interesting things can be made possible. Arguably, there needs to be some sort of operating system to facilitate this communication between devices and it is not possible to imagine the infinite possibilities such an interplay of devices can bring forth. So, the operating system cannot be pre-programmed. It needs to be cognitive. Hence Lucy.

3. Method

Lucy's brain is designed as a form of recurrent neural network (RNN) called a Long Short Term Memory (LSTM). The reason for picking this type of neural network is because it learns fast and outperforms other type of RNNs in time series prediction with long time lags [1]. An open source Java implementation of LSTM [2] & [3] is used for building and training Lucy.

The scenario Lucy is working with is this : A person has a morning routine on weekdays. He wakes up at a specific time, uses the bathroom, makes breakfast and leaves for work. Lucy tries to model his behavior based on his routine. The input files (train.txt and trainFloat.txt) contain this daily routine data as sensed by various sensors and reported to Lucy. Each line in the input file corresponds to data for a single day. There are two types of input files : one contains the times of activations of various sensors represented as floats (e.g: 6.47 for 6:47 AM) and the other as strings (e.g: 6:47 for 6:47 AM). Why? Because, how data is represented has direct impact on Lucy's predictions. There are two Lucy's (Sr and Jr) per input data type, each one with a different brain wiring (the type and number of memory units, training regime etc., see the attached pdfs for visual representations). So, in total there are four different Lucy's each with her own models for the person's behavior.

After observing the person for a while, Lucy begins to take actions on behalf of him. For example, when Lucy receives an activation from the mat at 6:45 AM, she will turn on the bathroom light automatically after about 15 minutes, as this is what the person has done in the past (this hypothetical person happens to be my roommate who works with Epic, I creeped him out for a couple of weeks to understand his morning routine. I could have used my own routine but it didn't exist during the winter break – thanks to PopcornTime. If you are wondering what it is, it is a video streaming app, the usage of which may or may not be legal/ethical/moral. Netflix – who?)

4. Running the code

You can find the source code, visualizations, input files, the executable jar file (lucy.jar) and this pdf all packaged into an archive. You need a JRE 1.8 installed. You can run Lucy like this and follow the onscreen instructions:

- `java -jar lucy.jar` // Default. Trains using the float input data and uses LucyFloatingSr
  // or LucyFloatingJr (whovever has higher accuracy)
- `java -jar lucy.jar ft` // Same as default. Here **ft** is the cmd line argument telling Lucy to
  // use **float** input and use a Lucy who performed better on
  // the **test** data. **f** in **ft** represents **float** and **t** represents **test**
- `java -jar lucy.jar st` // Here **st** is the cmd line argument telling Lucy to
  // use **string** input and use a Lucy who performed better on
  // the **test** data. **s** in **st** represents **string** and **t** represents **test**
- `java -jar lucy.jar s` // Here **s** is the cmd line argument telling Lucy to
  // use **string** input. Both Lucy' are used for prediction.
- `java -jar lucy.jar f` // Here **f** is the cmd line argument telling Lucy to
  // use **float** input. Both Lucy' are used for prediction.
- `java -jar lucy.jar t` // Same as ft
- `java -jar lucy.jar sft` // Same as ft

5. Results and Discussion

In the current stage, although the input files have different types of sensors and their activation times, Lucy can only predict the time when the bathroom light comes on when inputted the mat-force-sensor activation time. The accuracy of any type of Lucy is at best 50%. This requires extensive investigation as to how the accuracy can be improved. Evidently, building a neural net is no easy task. There are simply too many knobs to fine tune : building weight matrices, connecting different layers, infinite choices of training algorithms - should I use a BPTT, EA, RTRL, invent something new, somehow adapt Beethoven's symphonies and make an algorithm out of it, dissect an animal brain and see how it is wired and replicate it in code or use a different type of network altogether [4], [5] & [6]. Also, data representation dictates many things when building a system like this. For instance, the input files used have data from different sources : e.g. pSensor-6:45. This open source implementation of LSTM takes only floating point numbers as input for training and updates weights of matrices connecting different layers of the network. So the output layer outputs some prediction based on these weights. Hence correctly encoding pSensor-6:45 into a float and passing it to the training algorithm is absolutely crucial. Voila, another knob to turn. And the biggest problem of them all is having enough data. How much is enough? As someone said, a bad algorithm with lots of data will most definitely outperform a good algorithm with less data. Makes me wonder, maybe hard coding rules or pre-programming a system might be a better option. Or maybe, ditch trying to teach computers, hypnotize a monkey and make him learn certain behaviors and deploy him in a production scenario (may or may not invite the wrath of PETA).

6. References

[1] Sepp Hochreiter & Jurgen Schmidhuber, Neural Computation 9(8):1735-1780, 1997

[2] A generalized LSTM-like training algorithm forsecond-order recurrent neural networks

[3] Open source Java implementation of LSTM

[4] Echo State Networks

[5] Prediction Recurrent Artificial NeuralNetwork (PRANN)

[6] RNN trained with Extended KalmanFiltering (EKF) multistream training