Building the Best Idea Matching Algorithm Ever

- Conversation Feb 28, 2014
  - Jacob → Kiran
  - Kiran → Jacob
- Conversation Mar 4, 2014

Minutes Jun 22, 2014

Corpora

- Intuit Brainstorm ~?
  -
- Crunchbase
  - ~13,000
- ISIS ~3,000
  - mini demo http://ideamesh-isis2.herokuapp.com/
- Media Lab Projects ~300
- Private Equity Descriptions ~20,000
- Oxford Suggestions ~150
  - http://instadefine.com/IdeaOverflow/Outlinr-PHP/public_html/ideabox_catzjcr/public_html/index.1.7_suggestionbox_graph.php?mapid=80
  - http://ideajoin.com/
- MIT 100k
  - ideamesh-mit100k.herokuapp.com
- Hackathon Ideas ~250
  - Manually labeled
  - http://instadefine.com/IdeaOverflow/Outlinr-PHP/public_html/hackathonideas/public_html/index.1.7_suggestionbox_graph.php?mapid=169 click … > show all
- JISC
  - rhizi-sosiviewer.herokuapp.com

Method

- Cole and Lee's idea matching algorithm
  - Demo Query app+that+suggests+what+you+should+cook
  - **IdeaOverflow idea matching algorithm paper**, the basis for a platform for collaborative ideation and project sharing; Society of Mind
- (Socher?) Paraphrase detection

- Recurrent CNN

Implementation

- Pipeline
- Tools
    - NLTK
    - Stanford
    - OpenNLP
- Theano
    - Bengio’s student (Eliana)
    - Caglar
- Demo
    - D3
    - Justify

Paper

- Related Work
- Approach
- Evaluation
- Introduction

http://hackathonideas.tk/

Related Work

Idea Matching Algorithm

- Lieberman http://web.media.mit.edu/~lieber/Publications/You-Too.pdf

Relevant Work

- A Convolutional Neural Network for Modelling Sentences http://nal.co/papers/Kalchbrenner_DCNN_ACL14
- TSSP: A Reinforcement Algorithm to Find Related Papers
- Socher
    - Dynamic Pooling And Unfolding Recursive Autoencoders For Paraphrase Detection
    -

Conversation Feb 28, 2014

Jacob → Kiran

Henry Lieberman: Jacob's Adviser

Quid company: Eric Berlow and Sean Gourley: Mapping ideas worth spreading
http://www.youtube.com/watch?v=kv_uyUTx5Po

http://ideaflowplan.tk/
https://docs.google.com/document/d/113OoenkdNGJPW6XwOZdfYnhWtS1UoXx2XnrSM_GViag/edit
http://jcr.stcatz.ox.ac.uk/ideas/

http://gestaltbox.tk/

http://www.athinline.org/
http://www.media.mit.edu/~lieber/Publications/You-Too.pdf

http://people.csail.mit.edu/phw/
story understanding http://groups.csail.mit.edu/genesis/

Neuroscience: A Precis of Semantic Cognition: A Parallel Distributed Processing Approach
http://journals.cambridge.org/download.php?file=%2FBBS%2FBBS31_06%2FS0140525X0800589Xa.pdf&code=4f99012cab4ec4e5a69d566e33aeaf5e

----
if you have to implement a realtime, not-all that powerful "fuzzy search" efficiently, how would you do it? I'm investigating this tutorial...
http://www.opensourceconnections.com/2013/08/25/semantic-search-with-solr-and-python-numpy/

Inventing a Microchip

My Intel STS project originated while I was hiking during a thunderstorm. I was pondering, with some trepidation, what determines the paths of lightning bolts; I reflected that they must take some course of least resistance. Suddenly I was struck (by an idea): could an electronic hardware model of this physical phenomenon be used to solve the shortest path problem in graph theory (e.g. "What is the shortest route a car can take through a network of roads to arrive at its destination?")?

The lightning insight didn't pan out, but a week later, I found inspiration in a different natural phenomenon. While surfing, as I watched rivulets of water branching and re-fusing as they found their way down my surfboard, I realized that water molecules diffusing throughout a network could essentially function as thousands of identical-speed cars taking every possible path; the first "car" to reach the destination from the origin would have taken the shortest path. I simulated a graph with a network of paper towel strips, soaked one intersection of strips (the origin) in water, and watched the liquid diffuse through the network, marking which incident strip was the first to wet each subsequent intersection. Once water reached the destination, I could identify the path taken by the first molecules to arrive (i.e. the shortest path) simply by tracing the sequence of marked strips backwards from the destination to the origin.

This formed the basis of the parallel algorithm that I quantized, accelerated and then implemented synchronously on an FPGA microchip for the Intel STS; it ran on the order of 300 times faster than high-speed sequential approaches. Moreover, it generalized to solve the NP-complete (much harder) knapsack problem. I continue to investigate the paradigm's potential today.

My paper, entitled "Development and Synchronous FPGA Implementation of a
Parallel Accelerated Propagation-Delay Algorithm for Shortest-
Path Problem", can be found at: http://jacobsstuff.tpclubs.com/intelsts/
I would be much obliged if you would peruse it.

Kiran → Jacob

the key thing is that your representation of concepts/ideas needs to be very very good. I think that is the first step that needs to be taken in order to represent the relationships between objects.

So the focus should be on mathematically representing concepts in ways that are amenable to composition (we want to represent relationships in a way such that they are composable)

-- so memeome is potentially a good method of representing concepts

Named Entity Recognition → identify nouns in question

Professor Christiane Fellbaum -- WordNet, my advisor

(isRed) o (isSolid) o (canEat) o … = { set of objects with these properties } = S_p

ideally, you want to build such a composable language with the property that the number of components it has is minimal and at the same time such that | S_p | = 0 or 1

similar to semantic vector possibly? except the components of the vectors are supposed to be more like functions that output relations (some scale) when applied to a concept

The problem with this approach is that it is too finite
similar to semantic vector space stuff
MODEL 1:
each type i is an aspect of a general definition of a concept
when representing the relationship between v1 and v2 i want to define
how we translate from concept 1 to concept 2 along aspect i (of the general definition)

v = < a1, a2, a3, … > (represents a concept) (say a_i is type i)
f_v1v2 = < f1, f2, f3, … > (represents the way you compose v1 with v2)

then type (f_i) is (i -> i)

f_v1v2 (v) -> v' = <f1(v1), f2(v1), … > -- these are concept relationships
define what a concept is

m = f_i_j (matrix of functions f) → (rows are the basis vectors of v, columns are also the basis vectors of v) -> we want to generate a basis for the space M, where m in M. → the space M defines relations between all objects v in V.

I like the idea of defining relationships between their ideas in terms of "what they appear to be" -- duck typing -- but in a more tight sense -- you can start out by doing this, then I want to do some kind of dimension reduction so that you have essentially what the components appear to be for any kind of relationship in your graph

I want to be able to pick two things at random, put them together, and generate something new

David Blei:
http://www.cs.princeton.edu/~blei/publications.html

from Angela:
http://www.princeton.edu/~achaney/tmve/wiki100k/browse/topic-presence.html

http://en.wikipedia.org/wiki/Line_graph

Conversation Mar 4, 2014

MODEL 2:

each concept is labeled c1, c2, c3, …

process has input type (can have multiple wires coming in -- talk about it as a tuple -- tensor product)

output type is tensor product of output wires (tuple)

concept is a special type of process:
{I (type i1 x i2 x i3 x ..), O (type j1 x j2 x j3 x …), T (transformation: {} -> c, where c is the label) }

now how do we represent relationships between two concepts.

Let c1 = {I1, O1, T1:{} -> c1 }

Let c2 = {I2, O2, T2:{} -> c2 }

note that labels are almost arbitrary as an interface to the real world -- structure is topology of the graph

app that {suggests what you should cook}
node “app”
http://en.wikipedia.org/wiki/Relative_pronoun (that)
process “that” (is an operation that means “instanceOf”)
http://en.wikipedia.org/wiki/Relative_clause

end up with concept graph -- that is a bunch of processes -- draw a virtual membrane around it

-- using graphs of concepts to represent new concepts
-- find graphs of similar structure to compare -- but we still need to define comparison functions

howSimilarAre is different from howRelatedAre

similar = they share properties -> how do they check properties → however many match.
related = how connected in the graph, period

I feel like ConceptNet is a really brute force way of doing things. → but there is a lot of intelligence in it

Jiannan Notes

model grammars: pregroups (Lambek #book: “from word to sentence”)
monoid + left / right adjoint → compact closed category

example of pregroups

```
dog          chases              cat
np_l      (np_r)*s*(np_l)          np_r          ------->    s
subj         v            o
```

n_l * n_r = I

so we can do this for every single word in the english language and give unique types, distinguishing words spelt the same but with different semantic meanings by numbering.
bark1 ---> np_(l || r) (tree bark)
bark2 ---> (np_r)*s (here you have to use r) (dog bark)

dog	barks
(np_l)	(np_r)*s

model meaning: vector space (distributional model)
→ also compact closed category

dog		chases		cat
[v1...vn]	inner prod	cube	inner prod	[u1....um]

vector → sentence

cube: 3 components (but not in the same vector space) (takes left, right, and sentence)

(([v1...vn]   inner prod   cube) inner prod [u1....um]) -> sentence

sentence 1 * sentence 2 -- if normalize, it's the angle of difference in the vector space

-- we can also define other ways to compare sentences.

NLP/ML Tools:

if we will ever involve building distributional model, here is a great tool
http://radimrehurek.com/gensim/

three higher level libraries for machine learning:

http://scikit-learn.org/stable/
http://pybrain.org/
http://luispedro.org/software/milk

concept is a minimal idea (think of a single node being a graph)

```
Let c1 = {I1, O1, T1:{} -> c1 }
Let c2 = {I2, O2, T2:{} -> c2 }
----------------------------------------------------

#Our entire project is to build this method
#Args: idea1, idea2
# this will be model2 -- with processes and graphs that contain only edges
def Process relate(Process idea1, Process idea2):
        #todo
        return


class Process:
        Process source,target
        MathematicalStructure relation #function ? no


type Structure = I | O | T (* where I is input of type (i1 * i2 * i3 * i4 * ..), O is
(o1 * o2 * o3 * o4 * ...), T : None -> Label
```