Computational Humor with Word Representations in Vector Space

Kavan Patel, SEAS-Ahmedabad University

***Abstract*—Computational humor generation and recognizing are both aspects of Natural Language Processing and understanding. There has been some satisfactory work done for humor recognition but very limited research has been done for humor generation given that computational humor is a tough nut to crack. We as humans understand jokes because we have uncertain brains that are able to process the information and information that turns out to be joke because of a surprise element. Computers can't do such tasks because it follows a program written by programmer that he intentionally programmed it to do which won't work for humor generation. Some research has been done for humor generation which falls under a limited domain like "Knock-Knock jokes" which generates humor by word-play, "that's what she said", "I like my ... like my..." etc . Humor comes in many forms like joke, song parody, Internet memes (that we found on facebook and twitter). Till date no work is done to computationally generate Internet memes. Memes is having its own advantage and disadvantage if we consider generating them computationally. Memes span over different types and each type is having its own fashion of sentence structure. Thus generating memes is kind of a very complex task. But the positive point is part of humor is generated with the photo in background in meme. Moreover each type of meme follows different type of rules and structure. If we think how to generate such fashioned sentences computationally then there is a hope to expand the domain of computational humor.**

Keywords*—*Computational Humor, computer generated jokes, computer generated memes, word representation in vector space

I. Introduction

Computational humor is a branch of computational linguistics and intelligence that makes computer able to generate humor, be it in any form like a joke, pun, song parody etc. In 1843 Lady Ada Lovelace and English mathematician which is considered to be world's first computer programmer said that machines could not have human like intelligence as long as machines only did what humans intentionally programmed it to do. According to Lovelace a machine should be able to create original ideas if it is considered to be intelligent. Since many years people have been trying to develop computer creativity by originality and artistic outcomes. This includes poetry, sculptures, music, paintings and more. Humor is also one such element.

Humor comes with many types and is spread across varied cultural styles and languages. Lets say an English joke might not be understandable to a French guy though he knows English just because joke could be about Queen Elizabeth about which the French guy is unknown about. Due to such huge span of humor, humor generation becomes a tough task to do. Developing computers that are able to create humor is an essential task to do given that humor is needed in many human communication and interaction tasks.

It can be thought that computers in general have no need to generate jokes or any form of humor. But humor plays an important role in cases of steady supply of jokes where quantity is more important than quality. Another obvious, yet remote, direction is automated joke appreciation[1]. It is known that humans interact with computers in ways similar to interacting with other humans that may be described in terms of personality, politeness, flattery, and in-group favoritism. Therefore the role of humor in human-computer interaction is being investigated thereby making communication with computer easier and more intuitive.

Internet memes are in great boom nowadays and the more you go through these memes the more you understand about them. Internet memes have their origin in 21st century and are of varied types ranging from Good guy Steve, Thinking T-rex, Civil War, Crying Peter Parker, Facepalm, Matrix Morpheus etc. Each meme has its own structure of sentences like for Matrix Morpheus sentence begins with "What if I told you". Here the example of one such meme can be "What if I told you you've lost every game of tetris you have played" as shown in Fig. 1. There are lots of different types of memes floating around on the Internet. Each meme follows certain protocol according to which background and the text are related as random pun can't be placed over random meme background. This report shows and effective way to construct Civil War meme which is based on the theme of upcoming movie Captain America : Civil War. Though there are some false positives produced, but it does produce some acceptable memes that can be classifed as true positive and quite funny.

II. Related Work

Some work is done under the domain of computational humor generation that does give funny jokes. Taylor and Mazlack [2] which successfully frames Knock-Knock jokes by overlapping n-grams of words to produce humorous phrases. Knock-Knock jokes are certain fashioned jokes which produce humor by wordplays of two words by which combining gives a new word.

A Stanford project of humor recognition by Jim Cai and Nicolas Ehrhardt [3] constructed a Neural Network of some fixed size of input layer size which gives output 1 to 0

Kavan Patel is final year student of School of Engineering and Applied Science (SEAS, formerly Institute of Engineering and Technology (IET)) - Ahmedabad University pursuing Bachelor's in ICT (email : kavan33@gmail.com)

Fig. 1. Matrix Morpheus Meme[5]

Fig. 2. Civil War Meme[5]

for given input sentence of fixed length, 1 if it is pun else 0. Beauty of this Neural Network is that it ultimately gives the probability of a particular sentence being a pun. But on the other side it has to be of fixed length which can be considered as a constraint. The project successfully produced a model of joke recognition with satisfactory precision and recall.

Oliviero Stock and Carlo Strapparava modeled HAHAcronym[4] which gives funny acronym of any set of letters given to it. Model is having an inbuilt dictionary of many words which mostly focuses of sex, religion and food. It was stated that such types of humorous models help in making Human Computer Interaction (HCI) more intuitive and funny thereby making it easy for beginners.

Craig McDonough implemented the Mnemonic Sentence Generator, which converts passwords into humorous sentences. Basing on the incongruity theory of humor, it is suggested that the resulting meaningless but funny sentences are easier to remember. For example, the password AjQA3Jtv is converted into ”Arafat joined Quayle’s Ant, while TARAR Jeopardized thurmond’s vase”[1].

III. Vector representation of words

As humans we can read and analyse words but a computer can’t do such tasks. There has to be a method or protocol of words representations which can be handled by machines. There are 2 methods to represent words with vectors and this vectors are known as word vectors. They are : One hot encoding and Bag of words model.

A. One Hot Encoding

In this method each word is represented with a vector where size of vector is the number of unique words. In each word vector, 1 is represented with presence of a particular word and 0 as absence of other words. Example in sentence ”I like running” word vector of ’I’ is [1,0,0]; ’like’ is [0,1,0] and ’running’ as [0,0,1].

B. Bag of Words model

This representation is quite similar to above model, only difference is here vector represents a line/document as whole rather than single word with length of vector equals to dimension. Every digit of vector indicates the frequency of that particular word in that line. Example for ”I like running. I like ice cream”. Unique words are ”I, like, runnning, ice, cream” so its vector is [2,2,1,1,1]

C. Problems with such representations

Here, there is no semantic relationship expressed in such type of models. Like if we talk about ’library’, we get idea about books, authors, plays etc. Other problem is that if we talk about ’happy’ and ’cheerful’, both these words mean the same but their vector representations will be very different. Another example is if we take two sentences ”River bank of Ganga” and ”HDFC bank”, here word ’bank’ both mean different. Moreover, number of dimensions will be equal to number of words in entire vocabulary. So, there is a need of some effective model which solves above issues thereby making easy for machines to capture semantic relationships. Next sections focuses on addressing above problems.

IV. Co-occurrence vectos

The problem is to capture the semantics. To address the problem one possible solution can be to construct a co-occurrence matrix. Co-occurrence matrix is a matrix that have rows which contain information about the number of occurrence of words adjacent to target words. Here window size if fixed for a particular matrix. Ideal size ranges from 5-10. Window is fixed length(size) and words are taken into consideration if they fall in window. For example for sentence ”I like to eat ice cream.” the window size of 3 contains following [I,like,to], [like,to,eat], [to,eat,ice] and [eat,ice,cream].

To construct co-occurrence matrix, each word is selected as target word, two words from left and two words from right (for window size of 5) are considered and its frequency is noted. For example corpus:

- I like ice cream

counts	I	like	ice	cream	running	enjoy	flying
I	0	2	0	0	0	1	0
like	2	0	1	0	1	0	0
ice	0	1	0	1	0	0	0
cream	0	0	1	0	0	0	0
running	0	1	0	0	0	0	0
enjoy	1	0	0	0	0	0	1
flying	0	0	0	0	0	1	0

Fig. 3. Co-occurrence Matrix

- I like running
- I enjoy flying

For the above corpus Fig.3 is the co-occurrence matrix. Each element in row represents a word and its frequency in window of size 5 in corresponding word falling in row. The sparse of matrix represents the document as whole.

But here too there are few problems with co-occurrence matrix:

- Size of matrix increases with increase in size of vocalubary
- With increase in size of sparse matrix required storage increases.

Solution to above problem is to reduce the dimensionality of matrix by performing SVD (Singular Value Decomposition). For a matrix A, its SVD can be formulated as

$$A = U\Sigma V^T$$

Here the sparse (energy)of matrix A is expressed in U matrix. Extracting first 2 columns will demonstrate maximum sparse in space. Hence, by mapping rows of U to unique words will give the word vectors in 2D space. Here one thing to be noted is that words having same (not all of the same) adjacent words will have vectors quite close to eachother, because they share the adjacent words and co-occurrence counts as well. Fig. 4 represents the word vectors in vector space. As shown in Fig. 4, each word is represented as vector and lies somewhere in vector space. If we observe the plot, word *enjoy* is closest to *like*, and it can also be inferred that according to available corpus of above 3 sentences these two word most likely mean the same. But here it is also to be noted that *like* is not closest word to *enjoy*. The false positives are due to shortage of corpus, this can be addressed by increasing the corpus size.

A. Costly Algorithm

Lets say we have a huge text of 1 billion words, then size of sparse matrix will me 1 billion x 1 billion. Calculating SVD of such a huge matrix is very time costly. Time complexity of SVD calculation of m x n matrix is

$$O(m^2n + n^3)$$

V. Part of Speech (POS) tagging

Part of speech tagging is a complex problem in Natural Language Processing. It is a process of tagging words in text, like noun, adjective, named entity, pronoun etc. Viterbi algorithm is used in the process of POS. It is supervised learning algorithm where huge data of words and its tagged pairs are feed into algorithm. For wach word, the probability of a particular tag associated with that word is calculated and also the trigram model, for three consecutive tags is calculated. When a new word is to be tagged, it goes through analysing the probability of that particular word having a particular tag and sequence of two tags before it.

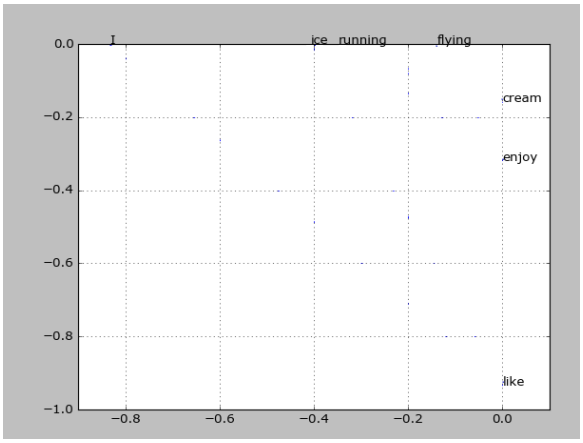

Fig. 4. Word vectors in vector space

A. Generative models for tagging

Let's say we have set of x^i and y^i for i = 1...m for words x^i and tags y^i. We have to map a function f that maps inputs (words) x to output (tags) y. Now we have

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

B. Trigram hidden Markov models

Let's say that probability for any sentence $x1...xn$ and tag sequence $y_1...y_n$ is $p(x_1, ..., x_n, y_1, ..., y_n)$.

$$p(x_1, ..., x_n, y_1, ..., y_n) = \Pi_{i=1}^{n+1} p(y_i|y_{i-2}, y_{i-1}) \Pi_{i=1}^{n} p(x_i|y_i)$$

VI. Word Representation in Vector Space with Neural Network

Beauty of Neural Networks is that it can train any function. Any function with appropriately normalized data can be fed into network and trained with promising accuracy but it does take much time to train. Neural Networks work on the algorithm called Gradient Descent. One cannot directly just feed the words into network and train them. They are to be converted into vectors which we call it as one-hot encoding as described in above section. Each word get a vector, which contains all the zeroes except 1 and the position of 1 is at unique place which corresponds to a unique word.

A. Activation Function

Sigmoid function is used as Activation function, because

- It is non-linear activation function, and non-linear function is used because there was need to map input and output by non-linear decision boundary.
- Back propagation calculates derivative of activation function. And calculating derivative of sigmoid function is easy. If $f(x)$ is sigmoid function, then its derivative is $f(x) * (f1 - (x))$ where $f(X)$ is

$$f(x) = \frac{1}{1 + e^{-x}}$$

These word vectors are fed into network in either of two ways:

- Continuous Bag of Words(CBOW)
- Skip-Gram model

B. Continuous Bag of Words (CBOW)

In this model, for an each word of corpus, network is trained in such a way that for context words (words of left and right of $target$ word) are input and output as $target$ word. The input is one hot encoded vector, which means that dimensions of every vector is equal to the size of vocabulary. Here N is the number of hidden layers and V is size of vocabulary. The weight matrix W of first layer would be of V x N, then

$$h = X^T W = V_{WI}$$

Now V_{WI} is passed to second Weight matrix W' of N x V. Using this matrix we can compute score of each word as

$$u_j = {v'_{wj}}^T . h$$

where v'_{wj} is j^{th} column of W'.

C. Skip-Gram model

In this model, the network is trained in such a way that $target$ word is input and its $context$ word are output. Here also, input and output are one-hot vectors, having dimensions equal to size of vocabulary.

Here weight matrix W is associated with first and second layer, and input being just 1 vector, with 1 at k^{th} position and rest 0,h would be just k^{th} row of W and the definition of h is

$$h = W_{k:} = V_{WI}$$

It is then passed to second layer and weight matrix W'. Using this matrix we can compute score of each word as

$$u_j = {v'_{wj}}^T . h$$

where v'_{wj} is j^{th} column of W'.

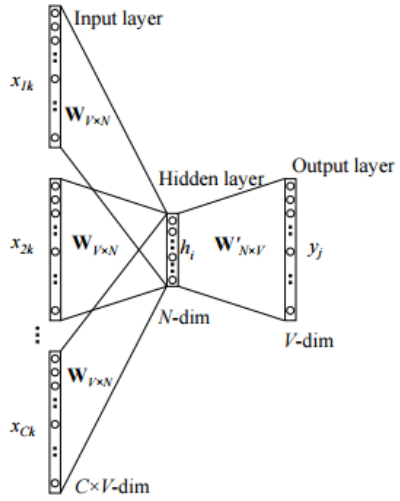

Fig. 5. Continuous Bag of Words model[6]

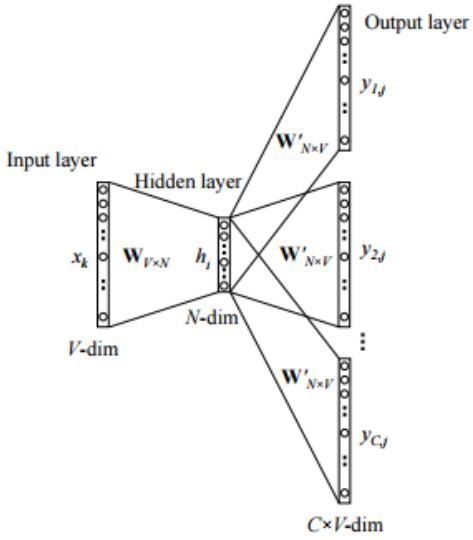

Fig. 6. Skip-Gram model model[6]

VII. Meme Generation

Civil War Memes are based on theme of upcoming movie Captain America : Civil War. The plot of the movie is all about the disagreement between Iron Man and Captain America. Meme describes the same thing, any thing that is said by Captain America is disagreed by Iron Man and that results into Civil War. Fig. 2 describes one such meme. Among football fans $Ronaldo$ and $Messi$ are top debating football players for who is best? Hence, Captain America says $Ronaldo$ which Iron Man disagrees and chooses to go for $Messi$. Thus every meme describes one such scenario of debate.

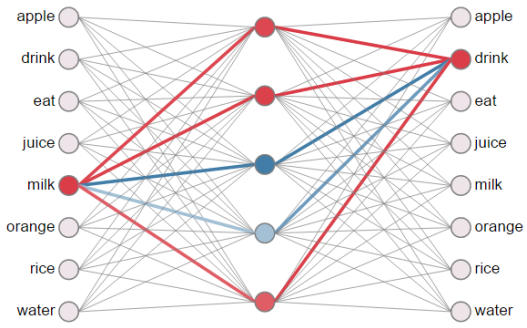

Fig. 7. Trained network of CBOW model

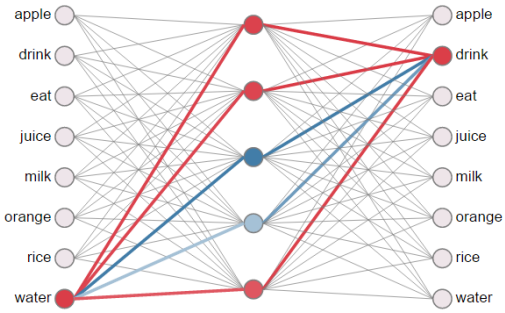

Fig. 8. Trained network of CBOW model

A. Magic of Meme generation

It seems to be a small magic behind generation of such memes. But the secret is behind in the Neural Network and the way they are trained. Each vector contains only one entry of 1 and rest zeros, which is unique for each word in vocabulary. This project utilizes Continuous Bag of Words (CBOW) model. $Context$ words are fed into input of network and $target$ as output and weights are set accordingly to train network till least error. In this model, the weight matrix corresponding to a particular word synapse and hidden layers is considered to be word vector. Here there might be some words that have same context words, if not all but some of those from four words (2 left and 2 right) can be same. Words having same or almost same context words will tend to have same weight matrix from word synapse to hidden layer. As hoswn in Fig .7 adn Fig .8, words $milk$ and $water$ tend to have same weights because they belong to $context$ words that have same output. Here one, thing is kept in mind that input and output words should haven noun tag. Output words that don't have noun tag get filtered out.

The weight matrix can be visualized that the red synapse is high and $blue$ is low. With this convention it can be easily visualized that word vector of $milk$ and $water$ are same.

B. Determining the similarity of word vectors

Each word is having word vectors of fixed dimension. To determine similarity among this vectors the easiest way is to get dot product. Beauty of this approach is that it gives the cosine angle between two vectors, and it ranges from 0 to 1, with 0 being most unlikely to be similar and 1 being most likely to be similar. It can be inferred that it gives the probability of two words being similar.

VIII. Remarks

This project uses arrays that are very huge in size and can't be fit in RAM. So a python library $Dask$ is used with which one can store huge arrays on hard-disk. But the catch is, when ever a function is applied to that array, a "task graph" is constructed and it is stored in memory (RAM). This task graph takes upto 100B to 1kB, and whenever a $loop$ is applied to such an array, each iteration just dumps millions of task graphs into task graph which in the end overwhelms machine.

In order to overcome above problem, another library $h5py$ is used which uses hierarchical data format to store arrays into hard-disk. But again problem with such arrays is that, data writing and reading takes more time, which in the end makes function call more time hungry.

So, $dask$ arrays are used along with $hdf5$ whenever a function is to be applied over a $h5py$ array, it is written into $dask$ array and then applied a function over it and output is written back into $h5py$ array. Thereby avoiding building of task graphs into task graphs and also making data writing , reading faster.

IX. Results

Fig .9,10 and 11 are some results including true positive and false positives. Input word is $Batman$ to which the disagreement leads as said by Iron Man which is shown in images.

X. Future Work

- This project produces only one-word memes. The project can be further extended to produce memes having sentences.
- This project focuses only on Civil War meme. Other types can be included with different methodology and structure.

XI. Conclusion

Humor is something difficult (or impossible) to make computers understand and even more difficult to produce. But Turing tests have shown us that it really doesn't matters if whether or not a machine understands like a human, so long as it can convince a human that it does. Here too, machine is able to give some results and some of them might be false positive while some can be true positive. The thing to be noted is that it does produce an element of humor, though it doesn't understands what humor is. I guess this can be considered to be passing Lovelace test to some extent but not as a full. The jury

Fig. 9. Civil War meme : False positive

Fig. 10. Civil War meme : False positive

Fig. 11. Civil War meme : True positive

may still be out as to whether it's fair to call this process of meme creation by machine humorous or not. But if a piece of work can make someone laugh or smile, does it really matter who created it?[17]

REFERENCES

[1] www.everything.explained.today
[2] Julia M. Taylor and Lawrence J. Mazlack, Humorous Wordplay Recognition (2004)
[3] Jim Cai and Nicolas Ehrhardt, Is This A Joke? (2013)
[4] Oliviero Stock and Carlo Strapparava, Laughing with HAHAcronym, a Computational Humor System (2006)
[5] www.memegenerator.com
[6] Xin Rong, word2vec Parameter Learning Explained (2014)
[7] Yoav Goldberg and Omer Levy, word2vec Explained: Deriving Mikolov et al.s Negative-Sampling Word-Embedding Method (2014)
[8] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean,Efficient Estimation of Word Representations in Vector Space (2013)
[9] Yilun Wang, Understanding Personality through Social Media (2015)
[10] http://www.ai-junkie.com/ann/evolved/nnt1.html
[11] Darrell Whitley, A Genetic Algorithm Tutorial
[12] http://deeplearning4j.org/word2vec
[13] https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html
[14] Mo Yu and Mark Dredze, Learning Composition Models for Phrase Embeddings (2015)
[15] https://ronxin.github.io/wevi.html
[16] https://en.wikipedia.org/wiki/Part-of-speech_tagging
[17] https://www.youtube.com/watch?v=Rh9vBczqMk0