SCI118: Word Embedding and Thought Vectors

Summer Research Scholarship Report

Student: Sam Clarke

Degree: BSc at University of Canterbury

Supervisor: Dr Jiamou Liu

Host Unit: Room 303S.599

University of Auckland

5th March 2017

Sam Clarke

Summer Research Scholarship Report 2016/2017

Word Embedding and Thought Vectors

1. The scholarship and my career development

The opportunity to spend 10 weeks with sustained focus on one project has furthered my career development in three ways. Firstly, it has given me the experience of having to think in a much more independent, creative and self-directed manner than in the ordinary course of my undergraduate studies. Pushing forward into new areas of human knowledge, rather than just soaking it up, having the freedom to explore in whatever direction I wanted, and having to create my own motivating goals and deadlines was a very challenging experience, but one which has taught me skills that will be indispensable in both further academic research and in industry.

Secondly, I developed the discipline and skills to sit down and read a technical paper. I also came to learn that many of the challenges I faced along the way had been encountered by other computer scientists before, and so honed my skills and confidence to go digging for answers on my own, rather than jumping to ask someone else (who, when I did, would often just google around or think in the same way that I could have).

Thirdly, I had the opportunity to further my technical and non-technical knowledge. I was exposed to the recent developments and challenges in NLP and the use of neural networks for understanding and reasoning. Non-technically, my learnings were equally as valuable: I found that I relish the challenge that comes with independent, creative and intense focus on one idea, but equally that I would thrive more in a collaborative environment than in working on a rather individual project like I did over the summer. These lessons are an invaluable guide as I consider my career direction after undergraduate study.

2. The research and its significance

If computers can better understand what we mean when we say things, the benefits would be enormous. Consider if Google knew exactly what you meant when you typed in a question: it would be able to return precisely the information you needed. Or if you could talk to Siri about complicated issues, and in the same natural way that you ask things of your friends, colleagues or teachers: this would be the end of researching by having to trawl through swathes of unhelpful articles on the net. Moreover, if a computer could make sense of all the information on the World Wide Web, it would be a tool of incredible utility, informing the decisions made by governments and companies with superhuman scope and analysis.

But the problem is, our language has evolved to be so convoluted that while it seems natural to us, linguists have been unable to identify the underlying rules that govern how we speak and write – and computers work by following rules. Therefore, this question of how to get computers to better understand our language is of great interest to computer scientists studying Artificial Intelligence. The field devoted to answering this question is called Natural Language Processing.

A key issue in this field is the question of how words, phrases or sentences should be represented. Computers work well with numbers, but if each word was just represented by any old number – its position in a dictionary, say – then so much information about that word is lost. To the computer, cat and dog would appear as unrelated as banana and seahorse.

One idea, then, is to represent each word by a string of numbers in such a way that words with similar meaning have similar number representations as each other. In some limited sense, the computer can be said to ‘understand’ a word like cat, because it knows it is similar to a dog and to a lion, but not to a banana. After all, it seems that we often understand words just by knowing other similar words to

it. Ask someone, for instance, what *benevolent* means and they will probably give you some synonyms, like kind, caring and charitable. In the words of the linguist J.R. Firth, *You shall know a word by the company it keeps.*

This idea is not a new one, but has become became popular again since 2013, when Tomas Mikolov and his team at Google developed a program to produce these number representations of words (called *word embeddings* or *vector representations*) faster than previous approaches. Their word embeddings also had some striking features that generated a lot of interest. They showed, for example, that with their approach you could take the vector representation for the word *king*, subtract the vector representation for the word *man*, and add the one for the word *woman,* and the resulting vector would be very close to the representation for *queen*. In some limited sense, they have enabled a computer to understand words and reason by analogy – a task that has been of great difficulty for traditional AI.

This is all background to our project; the rest of this article concerns how we aimed to develop and apply these concepts. The essential idea was to make a shift in the way these word embeddings are used. Instead of just using them to try and process or understand natural language, we wanted to investigate their potential to be used for modelling a 'train of thought'.

Why might this be a feasible alternative use for the embeddings, and what exactly is a 'train of thought'? When we have a conversation, or pursue a line of thought or reasoning, we often move from a topic or concept, to a related topic or concept, to another similar topic, and so on. For example, we might start out talking about the Trump Administration's latest announcement, to how the president managed to get elected in the first place, to the election system and what democracy means, to the upcoming New Zealand election, to our September holiday plans and so on. Or I might be thinking about how to solve a logic puzzle, when its similarity to another problem that I have solved before provides me with some insight which I apply to the current puzzle. Similarly, if I was rationally arguing for a case, I would start with a set of premises and follow a train of thought from these propositions to similar propositions, until I reached a conclusion.

In each of these cases, it seems like a train or flow of thought has something to do with moving from concept to similar concept. If we wanted to model this on a computer, therefore, the key would be to get some representation of concepts such that similar concepts were clustered together. This sounds a lot like what Mikolov and his team achieved with words, except that we would need to try and capture some notion of 'thought' or 'concept' or 'proposition', and convert these to vector representations, as opposed to just words. But if it could be done, then the hope is that a model of a train of thought would simply be a path from one vector representation of a concept, to another related concept (one with a similar vector representation), and so on.

But pinning down what the essence of a concept is, to then try to convert that essence to a vector representation, is an age-old and complicated task. After some futile attempts to try to use standard word embeddings to model a train of thought, we narrowed down the domain to informal, propositional logic, which consists of statements like 'This is making me fall asleep' and 'If I have to read any more, then I'm going to have to shoot someone'. Propositions like these certainly do not capture the full complexity of what human thought is, but if they could be successfully converted to vector representations, it would be an interesting start.

Mikolov's original approach to word embedding worked by using what is called Machine Learning, which relies on a vast quantity of input data. As the program processes more data, where the words are used in different contexts, it slowly improves the vector representations of those words. Accordingly, in order to use the same approach to convert simple statements to vectors, we would need a huge bank of simple statements. If we want similar statements that should follow from each other in a train of thought to be clustered together, then this bank of statements actually needs to be

a sequence that humans consider follow from each other in a train of thought or logical argument. A small extract from this sequence might look something like:

It is not the case that (it is sunny and I will not go to the market)
Either it is not sunny or I will go to the market
If it is sunny, I will go to the market

Each of these statements are equivalent, and the reasoning has simplified a complicated logical statement to a short one. The only difference between this example and the examples I used to train the model were that statements like *it is sunny* and *I will go to the market* are represented in my examples by variables like X and Y, to maintain their generality. It is the reasoning process that we want to be able to model, without regard for the particular context of the reasoning.

The trouble is though, no one has had reason to set about producing a massive bank of simple statements that have been logically reasoned on, at least in a form a computer can process. My first task, therefore, was to produce this, using a common technique in Artificial Intelligence called *search*. I then moved onto applying Mikolov's techniques to try to obtain the vector representations of these statements.

The time I had left to investigate the Word2vec style models of informal logic reasoning, after they had been 'trained', was very limited. Nonetheless, amongst a lot of garbage, there were a few promising looking results, where the trained model was fed statements which it then simplified a little, a task amounting to 'reasoning' in some limited, rudimentary sense. Given the greater accuracy and efficiency of existing algorithms that can simplify informal logic statements, this is hardly a ground breaking result. However, what is different here is that the model was not explicitly given any information about systems of informal logic, logical laws, or when to apply them, but that, *despite this*, we saw examples where statements were successfully manipulated and simplified.

Looking forward, I want to find time to investigate the reliability of such a model. It would be very worthwhile to look more into the circumstances in which the reliable results were produced, and how altering the parameters of the model and the examples of reasoning it is given affect the quality of the results. Furthermore, Mikolov's Word2vec is not the only program developed to create vector representations, and it would be very interesting to see how results from other models compared.

3. Abstract

The group of neural networks called 'Word2vec' were initially developed to further the ideas and tools used in natural language processing. The central idea is to exploit the fact that words with similar meaning have similar contexts. Therefore, a neural network which creates vectors to represent words, and trains the vectors representing words that occur many times with similar words surrounding them to have similar values, will cluster words with similar meaning together in vector space. We investigated the application of Word2vec style algorithms to model a train of thought. If thoughts can be represented as vectors, and the vectors representing thoughts with similar content are clustered together in vector space, then a 'train of thought' would simply be a path through this vector space, from thought to similar thought. To explore this idea, we narrowed down the definition of 'thought' to 'informal logic statement form', and 'train of thought' to the train of reasoning that is produced by using intuition to simplify a long and complicated logical statement down to a simple one (as is necessary, for example, to prove a form of argument in informal logic is valid). We found some initially promising results, where vectors representing statements that follow in good logical reasoning can be produced by vector addition, subtraction and calculations of cosine similarity between vectors. However, many of the results were also garbage, and much more investigation needs to be done on the reliability of such models, how it is affected by hyperparameters and the training examples, and the use different neural network architectures for this purpose.

4. Aims

A. To understand how the 'Word2vec' algorithms of Mikolov et al. produce vector representations of words
B. To understand why these vector representations have the properties they do
C. To review some of the literature in Cognitive Psychology that attempts to provide a model for how trains of thought flow in the human brain
D. To investigate the potential for a similar algorithm to produce embeddings for 'thoughts' or 'concepts', and for the use of these vectorised thoughts to model a 'train of thought'

5. Methods

Several online papers, tutorials and discussion forums helped me understand how and why Word2vec works. I had to fill in various holes in my knowledge of more foundational concepts in neural networks and NLP, to then understand Word2vec. These included backpropagation, logistic and softmax functions, cross-entropy cost function, cosine similarity and clustering. Online resources were my main method for getting up to speed with these concepts as well. As with all my resources, the most important and useful ones are listed in Section 8 of this report. A method that helped me to achieve the first two aims was to implement my own (naïve) version of Word2vec in Python. While it lacks the optimisations of the various online implementations, and so is too inefficient to train on a large dataset, it was a good exercise to ensure I fully understood the algorithm. I experimented with trained Word2vec models and trained my own, using the Gensim implementation of Word2vec in Python.

For the development of ideas in the later part of the project, I read papers and watched videos of seminars about Geoffrey Hinton's notion of Thought Vectors. We were keen to draw inspiration from ideas in Cognitive Psychology about spreading activation and train of thought, so I read into the literature on this as well. Discussion with my supervisor was an invaluable resource. All algorithm implementation was in Python.

6. Results and Discussion

For each project aim, details are given of my results, followed by some discussion.

(1) To understand how the 'Word2vec' algorithms of Mikolov et al. produce vector representations of words

Word2vec is a group of different neural network architectures, and I was primarily concerned with the skip-gram algorithm. It is a fully connected, feedforward shallow neural network with one hidden layer. It is trained to predict the words that fall inside the context window of each word in the corpus. The input layer has V nodes, where V is the size of the vocabulary in the training corpus. The hidden layer has N nodes, where N is the chosen dimensionality of the word vectors. The values of the nodes of the input layer are a one-hot vector for the particular word in the vocab that we are trying to predict the context of. As for the hidden layer, there is no activation function so the value of each hidden node during feedforward is just the weight from the input node representing the current input word to the hidden node. These input-hidden layer weights are known as the word vector for the input word. The output later has V nodes, and softmax is applied to the weighted sum. The value of each word's output node represents the estimated probability that the word is in the context window of the input word, or more precisely, that if a word is randomly selected in the window of the input word, that word is the output word.

The backpropagation equations for updating the weights of the neural network are

$$w_{ij}'^{(new)} = w_{ij}'^{(old)} - \eta \cdot \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot h_i$$

$$w_{\mathbf{ki}}^{(new)} = w_{\mathbf{ki}}^{(old)} - \eta \cdot \sum_{j=1}^{V} \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot w'_{ij} \cdot x_{\mathbf{k}}$$

where w is the input-hidden weight matrix, w' is the hidden-output weight matrix, η is the learning rate, C is the context window size, x, h and y are the input, hidden and output node values respectively after feedforward, and t is the desired target output node value (either 1 if the word is in the context window, else 0). On the most basic implementation of skip-gram, every weight is updated for every word in the corpus and for every epoch that the model is trained for, but in practice, negative sampling is used to reduce the number of weights updated. Also, hierarchical softmax, a more computationally efficient version of softmax, is applied to the output nodes.

These equations are derived to minimise E, where

$$E = -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,C} | w_I)$$

w_{Oi} is the i^{th} word in the input word's context window and w_I is the input word. This means that the network will be trained to output high probabilities for words which appear in the context window of the input word.

(2) To understand why these vector representations have the properties they do

The basic intuition behind why skip-gram produces interesting vector representations of words is that words with similar meanings have similar contexts. The network is trained to output the probabilities of other words in the vocab appearing in the input word's context. Therefore, words which often have similar contexts should give similar output, and so words with similar meaning should give similar output. Now, the essential feature of the network that will make different input words give similar output is the input-hidden layer weights. The nodes representing the different words must have similar weights connecting them to the hidden layer if they are to given similar output. The reason for this is that, since the network is fully connected, the same function will be applied to all words between the hidden-output layers, so changing these weights alone is not sufficient to ensure different input words will give the same output. It is the input-hidden layer weights that must be similar if two different words are to give similar output. Therefore, words with similar meaning will have similar input-hidden weights, and these weights are interpreted as the word vectors for each word. In this way, the simple, shallow neural network assigns similar word vectors to words with similar meaning. As Goldberg and Levy point out in their 2014 paper, this intuition is not very precise. There is an opportunity for this explanation to be formalised. This also fails to explain the model's performance in analogical reasoning tasks, like the famous vector('King') – vector('Man') + vector('Woman') = vector('Queen').

(3) To review some of the literature in Cognitive Psychology that attempts to provide a model for how trains of thought flow in the human brain

John Anderson makes a well-known attempt to create a simplified computational model to explain some of the features of our psychology. One feature of this 'ACT-R' architecture is the concept of *spreading activation*. Under the model, memories are retrieved when they become 'activated' by similar memories. Activation spreads according to (1) association strength and (2) the baseline activation of the memory. Anderson illustrates the explanatory power of the model with the results of 'free association' tasks, where memories or thoughts are given to a subject who is then asked to say the first memory or thought that comes to mind. 'Bible' alone might activate the thought of 'Jesus' or 'Moses', because these are strongly associated with the prime word and have high baseline activation. But if 'Bible' and 'Arc' and 'Flood' were the prime words, then 'Noah' would become

activated, because even though it has lower baseline activation than 'Jesus' (it is less frequently recalled), it has strong associations to all three prime words. The activation equation he gives is:

$$A_i = B_i + \sum S_{ij} A_j$$

where A_j is the activation of a concept in memory, A_i is the activation of a potential next concept, B_i is the baseline activation of that next concept, and S_{ij} is the strength of the association between A_i and A_j. The sum is over all the concepts currently activated. The concept in memory with the highest activation A_i is the memory or concept that comes to mind, or the next thought in a train of thought from A_j.

Applied to problem solving, this model suggests that if we can build strong associations between the underlying form of problems and a solution strategy, then we will be able to recall that strategy when faced with an unfamiliar problem (if the strategy is relevant). The ACT-R explanation would be that activation in the part of our brain representing the underlying problem form will spread to the internal representation of the solution strategy (if there is a strong association between the two), and this strategy will come to mind. We might call this a chain of reasoning. The strength of the associations increases with practice on different problems.

As we shall see, there is a similarity between this model and the use of vectorised thoughts to model a train of thought. In both, the strength of association affects where the direction of the train of thought. In the brain, this strength of association is a function of how frequently we make the association (a theory summarised by Hebb's Rule: "neurons that fire together, wire together"). For vectorised thoughts, association strength is given by the cosine similarity of the vector representation of the thoughts.

There are two obvious differences between the ACT-R model of train of thought and modelling a train of thought by a path through vector space. First, vectorised thoughts have no baseline activation; this might be something to add in to an activation function, based on the frequency that the word occurs in the corpus. Secondly, the brain does not work in this highly abstracted way: there is a lot of redundancy in the brain when concepts activate similar concepts. A precise path between word vectors does not capture the loose and poorly understood notion of a train of thought being activated in the brain. This will be discussed more below.

(4) To investigate the potential for a similar algorithm to produce embeddings for 'thoughts' or 'concepts', and for the use of these vectorised thoughts to model a 'train of thought'

In his 2015 speech to the Royal Society in London, Geoff Hinton introduced the notion of a 'thought vector'.

> *"…if you can convert each sentence in a document into a vector, then you can take that sequence of vectors and [try to model] natural reasoning… If we can read every English document on the web, and turn each sentence into a thought vector, you've got plenty of data for training a system that can reason like people do." (Hinton, 2015)*

Word2vec is trained to generate a word's context, and in doing so, words with similar meanings are mapped to similar vectors. Following this, one interpretation of what Hinton is saying here is that if a neural net was trained to generate the context of a thought, then similar thoughts would be clustered together. Then, a train of thought would simply be a path through the vector space, from similar thought to similar thought. In exploring this idea, I found there to be two essential problems with the idea of mapping thoughts to vectors.

Firstly, there is no precise notion of what a thought is. Hinton argues that "a sentence is a thought, because for any sentence X, I can say 'I think X'. That makes it a thought." He seems to suggest that being a sentence is a sufficient condition to be a thought, but it is unclear whether he thinks it is necessary that thoughts be sentences. In any case, thoughts combine proposition, sensation, sometimes emotion and are far more complex than just sentences. Conversely, sentences mingle thoughts, and some sentences have more thought content than others. But this does not mean that the notion of a thought as a sentence is not a good place to start the investigation.

Secondly, and more problematically, thoughts are combinatorial. As the thoughts or sentences to be vectorised become longer, the length of the set of all possible thoughts increases exponentially. Word2Vec style algorithms rely on each unique word having a unique index in the vocabulary, and moreover on each word being present in the training corpus many times, to allow the model to learn the word's context in many situations. But in a corpus of many sentences, the set of unique sentences would become infeasibly large from a computational point of view, and the vectors will not be trained well if no one sentence occurs more than once (which is likely).

Nonetheless, if these questions can be resolved, the notion of a thought vector is promising. Thus, our project was concerned with these questions, and answering them in some way that allowed us to create and investigate some rudimentary model of a train of thought. To do so, I needed to decide (1) What thought-like entities will be converted to vectors, (2) What corpus of these entities will be used to train the model, (3) What neural network architecture will be used for training, and (4) How the next thought in a sequence will be generated once the model is trained.

I tried three different approaches to these questions. The first was to use the original Word2vec models I generated, with the hope that words would turn out to be closer to 'thoughts' than it might appear. I attempted to model a train of thought by finding the word represented by the word vector with the highest cosine similarity to a starting word's vector. Perhaps this word, and then the next one generated in the same way, will be like thoughts that might follow in a train of thought.

Call the method in which the next word is generated from the current word the 'activation function'. The problem with the above idea of using the most similar word as the activation function is that the train of thought converges to oscillations between two words. If $word_2$ is the most similar word to $word_1$, then $word_1$ will likely be the most similar word to $word_2$, and if not, then it will be after making a few moves to the next most similar word. The activation function needs to incorporate some non-determinism, not just selecting the most similar word. Getting stuck in a loop of words for a while might not necessarily be a bad thing (this might be analogous to humans conversing for a while about an interesting topic), but it needs to move on at some point.

I developed an activation function with these features. It was not allowed to loop too much, incorporated baseline activation based on word frequency and had a probabilistic selection of the next word from the list of words with highest cosine similarity. This would sometimes generate plausible looking chains of thought, like *'Barack_Obama', 'McCain', 'Clinton', 'Illinois_senator', 'Biden', 'Bush', 'White_House'* (this model was trained on a corpus of Google news articles). However, an essential problem remains that, really, this is just a chain of related words. It seems plausible as a chain of thought mostly because of the embellishments we attribute to it when we read it. We can imagine these words being the changing subject of a conversation if we try, but in reality they are just related words.

We also considered using a clustering algorithm to cluster words together that have a high cosine similarity. Then, each cluster of words might represent more of a detailed 'concept'. This would both stop the activation function jumping to trivially similar concepts, and reduce the computational complexity of the function, for there would be fewer cluster prototypes than words to which the current cluster's cosine similarity must to calculated. Because of the massive number of vectors in the model, however, my attempts to use the sklearn.cluster module to achieve this were fraught with

memory issues. Further, it was hard to get a feel for the embeddings and therefore to decide what sort of clustering algorithm to use. With more time, this might be something to take further.

The final two approaches involved a narrowing of the scope of the 'thoughts'. Instead of using embedding techniques to try to model chain of natural language sentences, I attempted to model a 'chain of reasoning' in one well-defined domain. After considering algebra, calculus, geometry, logic, balancing chemical equations and basic computer science problems, I decided to try modelling algebraic reasoning as a proof of concept, then moved onto the domain of informal statement logic.

Before describing my strategy and results, I will comment on some insights from cognitive psychology on how humans reason in well-defined domains. When novices are learning how to reason in a domain (solving quadratic equations, for example) they tend to go through rote learned strategies in their heads, examining whether any of them lead to a solution. They have explicit knowledge of rules to follow, and they try applying each one. Experts, however, 'just know' what strategy to use. Instead of working backwards from rules to solutions, they reason by their ability to 'see' the right way forward, using intuition built up over hours of practice. When asked why they applied a certain rule, they may have lost the ability to explain declaratively why they chose that one, instead applying it simply because it 'feels right'. This phenomenon is documented in many domains, including physics, maths, computer programming and board games like chess and Go.

This distinction has striking similarities to the distinction between good old fashioned computational approaches to solving problems in well-defined domains, and what my project proposes as an alternative way to model a chain of reasoning in a domain. Algorithms that solve reasoning problems in these well-defined domains typically employ mechanical techniques like a novice would. For example, an algorithm to show two logical propositions are logically equivalent may follow a rule-like procedure to convert both propositions into a form called *disjunctive normal form,* and then show that those are equivalent. This is a mechanical but not a very intuitive or elegant way to prove the logical equivalence of two statements.

By contrast, it is the more intuitive reasoning that a neural network may be able to model. There are three reasons why this might be desirable. Firstly, the chain of reasoning that the model produces might be more intuitive, simple or direct. Secondly, this sort of model might be able to continue to solve problems that become computationally too expensive to solve mechanically as they become more complex. Thirdly, since the neural net does not need to be given a procedure or set of rules in order to reason towards a solution, these techniques may be able to be applied to domains where no procedure or set of rules have been established.

I will now describe how I used a word embedding neural network to try to model reasoning in the domain of informal statement logic. In answer to the above questions, (1) informal logic statement forms, such as *'A implies B'* or *'not(D) or not(B and not(A))'* will be converted to vectors. Statement variables are limited to A, B, C or D, and the allowed predicates are conjunction, disjunction, implication and negation. (2) For the neural net to learn 'proposition embeddings' that will allow a sequence of quality reasoning to be generated after training, a corpus of many examples of quality logical reasoning is needed. Reasoning in informal logic can performed by applying the logical laws, which are a set of standard logical equivalences and implications, to a statement form. A table of these is given in Appendix A. This method can be demonstrated to be a sound and adequate method of proving that two statement forms are logically equivalent or that one logically implies the other. Demonstrating logical equivalence or implication by a successive application of laws can be considered 'reasoning' in this domain, not only because it is a task that requires what we would call 'reasoning', but also because this is a method of proving whether an argument form – quite literally a chain of reasoning – is valid. The quality of reasoning in informal logic is measured by how much the applied laws shrink down size of the statement form, in much the same way that applying good operations to both sides of a mathematical equation makes it more simple.

Because there are no published corpuses of endless examples of logical reasoning in a form that could easily be fed to a neural network for training, I needed to produce these examples. For this, I used 'search', a common technique in Artificial Intelligence. The idea was to apply logical laws at random to randomly generated statements (of depth no more than four levels of nesting), and searching for a path of successive law applications that shrinks down the size of the statement form. I used a heuristic to cut off the search at a depth of 4, selecting the most promising statement form and not continuing to search with the remaining ones, because doing an exhaustive search is computationally too expensive. The heuristic was the mean average of the maximum depth/nesting of the expression tree representing the statement form and the total number of predicates and variables in the statement form.

With the training examples, I tackled the questions of (3) what neural network to train and (4) how to generate the next thought in the sequence after training. Word2vec was my first choice for a model, because I was already familiar with it. After training, it was hoped that statement forms that should be reasoned on in similar ways (i.e. the application of the same laws shrinks them down) will be clustered together in vector space, because statement forms that should be reasoned on in similar ways have similar contexts.

Recall, however, that skip-gram training only clusters words with similar meanings together because in a large corpus of, say, news articles, there will many sentences of the form 'X met with other world leaders at the conference' where X is very often a world leader. This means that vectors representing world leaders get clustered together because of their consistently similar contexts. But suppose that there is a chain of reasoning from statement forms A_1 to A_2 to A_3, and a similar chain of reasoning from B_1 to B_2 to B_3. Because all six propositions are different, A_i has no context in common with B_j for any i and j. Therefore, then no similarity will be learned between A_i and B_j.

I had two methods of getting around this issue. Firstly, I trained some models with the examples of statement forms that had been "cut off" at various depths. For example, 'And(Or(A,B), Not(Or(B,C)), A)' cut off at a depth of 2 would give 'And(A, Not(), Or())'. When the model is trained with cut off statements, there will be many more identical statements in the examples. In the above example, statements A_2 and B_2 might be identical if both are cut off at a certain depth. Thus, A_1 and B_1 will now share some context and get clustered together.

Secondly, I trained some models with *(statement, logical law)* pairs for the input and output of Word2vec, rather than the regular *(statement, adjacent statement)* pairs. *Logical law* is either the law that produced the *statement* in the example reasoning, or that is applied to *statement* to produce the next one. This will mean that statements that are consistently manipulated by the same logical laws will be clustered together. Also, propositions that follow from each other in the example reasoning will be nudged closer, because they share a *logical law.*

Then, there are several possible methods for obtaining the next statement that should follow in good logical reasoning, given an arbitrary statement. If the current statement occurs in vocab, the next statement might be the one with highest cosine similarity, or a little less than the highest. Perhaps a sort of analogical reasoning could be used instead: if the current statement, S_0, has a high cosine similarity to another statement T_0, and statement T_1 logically followed from T_0 in the original corpus of reasoning, then the vector addition of S_0 to T_1 - T_0 might give the statement that should follow from S_0. If the statement is not in vocabulary, take top d = N - 1 levels of statement (where N is the original statement depth), and see if that is in the vocab for the model trained on statements that were "cut off" at depth d. If this is not in the vocab, then take the top N - 2 levels of the statement, and so on.

I trained 36 models: one for each of six cut off depths (including no cut off), with both adjacent statements and logical laws as the output that the neural net was trained to predict, and for four different Word2vec models. These models were the standard Gensim implementation which includes reduced window and negative sampling optimisations, one without these optimisations, one that

trained the net to predict the *current* statement for each statement, as well as its context, and one which trained only to predict the logical law, not any adjacent statements. Chosen hyperparameters for training were 150 epochs, 50 dimension vectors, learning rate of 0.1, context window size of 1 and a minimum statement count of two for it to be included the model. The skip-gram algorithm was used. These hyperparameters were chosen by trial and error. (Mostly just error, rather than trial. I needed more time to investigate with these further!)

Unfortunately, because of time constraints on the project, I could not complete a thorough investigation of the models and the different ways of producing the next statement that should logically follow in a sequence of reasoning. In the future, I will look more into this, and train models on corpuses of different reasoning quality, with different hyperparameters. However, I did obtain a few promising looking results. For example:

```
>>> t0 = model['Implies(Not(B), D)'] # t0 is the 'statement vector' representing to Not(B) → B
>>> t1 = model['Not(And(Not(), Not()))'] # t1 is the 'statement vector' representing Not(Not(*any statement*) & Not(*any statement*))
>>> s0 = model['Implies(A, B)'] # s0 is the 'statement vector' representing A → B
>>> model.most_similar(positive=[s0, t1], negative=[t0]) # 'analogical reasoning': what is the most similar statement to s0 + t1 – t0?
[…, ('Not(And(A, Not()))', 0.857), …] # the statement 'Not(And(A, Not())) has a cosine similarity of 0.857 to the vector s0 + t1 – t0
```

To explain: (positive=[s0, t1], negative=[t0]) represents the difference between the vector for Not(B) → D and the vector for Not(Not(*any statement*) & Not(*any statement*)), (representing an application of the implication law) added to the vector representing A → B. Querying the model for the most similar statements to this resultant vector, s0 + t1 – t0, gives a vector which is similar to Not(A & Not(*any statement*)). This statement would indeed be produced by the application of the implication law to t0. There were other statements with slightly higher cosine similarity to the resultant vector, but nonetheless the result I have shown was one of the most similar statements with the chosen combination of hyperparameters. While this small instance of 'reasoning' is nothing that a mechanical algorithm could not produce, what is important here is that the model was not fed any explicit logical laws, let alone information about when to apply them, yet a well-chosen law has been applied nonetheless.

However, my investigation was far from conclusive. Most of the results produced in this way could not be described as 'good logical reasoning'. Looking forward, perhaps a different neural network architecture would be a more promising way to model of train of thought. Recurrent neural networks (RNNs), for example, can also be used to convert 'thoughts' to vectors. Some proposed RNN architectures for this purpose function as encoder-decoders. The input to the encoder is a sequence of words, thoughts or statements, which get encoded as a sequence of vectors. This sequence is supposed to be an abstract representation of the content of the sentence or thought. The decoder is then trained to convert from word vectors to a sequence of words, thoughts or statements that follow from the input. Some promising results have been seen using this sort of technique in machine translation, where a sentence of one language is encoded to vectors, representing the content of the sentence, then the decoder part of the network produces sentences in the target language from those vectors. In the future, I am keen to pursue these trains of thought further. The work I put into writing an algorithm to produce a large corpus of quality informal logic reasoning will be able to be used to experiment with the training of different neural networks in the future – this was a successful and concluded part of my project. While I did not take the ideas as far as I wanted to, the project has been an outstanding introduction, and given me the motivation and some essential skills needed to extend my study of neural networks and natural language processing.

7. Appendices

A. Standard logical equivalences and implications

Table: Logical laws

$\mathcal{A}$	$\Leftrightarrow$	$\neg\neg\mathcal{A}$	Double negation
$(\mathcal{A} \wedge \mathcal{B})$	$\Leftrightarrow$	$(\mathcal{B} \wedge \mathcal{A})$	Commutative law
$(\mathcal{A} \vee \mathcal{B})$	$\Leftrightarrow$	$(\mathcal{B} \vee \mathcal{A})$	Commutative law
$(\mathcal{A} \wedge (\mathcal{B} \wedge \mathcal{C}))$	$\Leftrightarrow$	$((\mathcal{A} \wedge \mathcal{B}) \wedge \mathcal{C})$	Associative law
$(\mathcal{A} \vee (\mathcal{B} \vee \mathcal{C}))$	$\Leftrightarrow$	$((\mathcal{A} \vee \mathcal{B}) \vee \mathcal{C})$	Associative law
$(\mathcal{A} \wedge (\mathcal{B} \vee \mathcal{C}))$	$\Leftrightarrow$	$((\mathcal{A} \wedge \mathcal{B}) \vee (\mathcal{A} \wedge \mathcal{C}))$	Distributive law
$(\mathcal{A} \vee (\mathcal{B} \wedge \mathcal{C}))$	$\Leftrightarrow$	$((\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C}))$	Distributive law
$\mathcal{A}$	$\Leftrightarrow$	$(\mathcal{A} \wedge \mathcal{A})$	Idempotent law
$\mathcal{A}$	$\Leftrightarrow$	$(\mathcal{A} \vee \mathcal{A})$	Idempotent law
$\neg(\mathcal{A} \wedge \mathcal{B})$	$\Leftrightarrow$	$(\neg\mathcal{A} \vee \neg\mathcal{B})$	De Morgan's law
$\neg(\mathcal{A} \vee \mathcal{B})$	$\Leftrightarrow$	$(\neg\mathcal{A} \wedge \neg\mathcal{B})$	De Morgan's law
$(\mathcal{A} \rightarrow \mathcal{B})$	$\Leftrightarrow$	$(\neg\mathcal{A} \vee \mathcal{B})$	Implication law
$(\mathcal{A} \rightarrow \mathcal{B})$	$\Leftrightarrow$	$\neg(\mathcal{A} \wedge \neg\mathcal{B})$	Implication law
$\mathcal{A}$	$\Leftrightarrow$	$(\mathcal{A} \vee (\mathcal{B} \wedge \neg\mathcal{B}))$	Contradiction law
$\mathcal{A}$	$\Leftrightarrow$	$(\mathcal{A} \wedge (\mathcal{A} \vee \mathcal{B}))$	Absorption law
$\mathcal{A}$	$\Leftrightarrow$	$(\mathcal{A} \vee (\mathcal{A} \wedge \mathcal{B}))$	Absorption law
$(\mathcal{A} \leftrightarrow \mathcal{B})$	$\Leftrightarrow$	$((\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A}))$	Equivalence law
$(\mathcal{A} \leftrightarrow \mathcal{B})$	$\Leftrightarrow$	$((\mathcal{A} \wedge \mathcal{B}) \vee (\neg\mathcal{A} \wedge \neg\mathcal{B}))$	Equivalence law

B. Extract from corpus of informal logical reasoning

I have included a small extract from one of the corpuses of 'informal logic reasoning' that my search algorithm produced. The lines in between the informal logic statements indicate the logical law that was applied to get from one to the next. 'disj' and 'conj' indicate whether the 'disjunctive' or 'conjunctive' version of the logical law was applied. 'F' and 'R' indicate whether the law was applied in the forwards or reverse direction. Two examples are shown; the training corpus included thousands of such examples.

Implies(Not(B), Not(Not(D)))
double_negation_R
Implies(Not(B), D)
implication_conj_F
Not(And(Not(B), Not(D)))
implication_conj_R
Implies(Not(B), D)
implication_disj_F
Or(D, Not(Not(B)))
double_negation_R
Or(B, D)

Implies(Or(B, Not(And(A, B))), Implies(And(A, Or(A, B)), B))
absorption_conj
Implies(Or(B, Not(And(A, B))), Implies(A, B))
de_morgan_conj_F
Implies(Or(B, Not(A), Not(B)), Implies(A, B))
implication_disj_R

Implies(Implies(A, Or(B, Not(B))), Implies(A, B))
implication_disj_R
Implies(A, B)

8. References

http://www.deeplearningbook.org/
https://www.tensorflow.org/versions/r0.11/tutorials/word2vec/index.html
https://rare-technologies.com/making-sense-of-word2vec/
https://arxiv.org/pdf/1402.3722v1.pdf
https://rare-technologies.com/word2vec-tutorial/
https://arxiv.org/pdf/1301.3781v3.pdf
https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
http://cs.stanford.edu/~quocle/paragraph_vector.pdf
http://neuralnetworksanddeeplearning.com/
http://neuralnetworksanddeeplearning.com/chap2.html#the_four_fundamental_equations_behind_backpropagation
https://arxiv.org/pdf/1405.4053v2.pdf
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/rvecs.pdf
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.501.1570&rep=rep1&type=pdf
https://github.com/sherjilozair/char-rnn-tensorflow
http://mccormickml.com/2014/03/04/gradient-descent-derivation/
https://www.utwente.nl/cw/theorieenoverzicht/Theory%20clusters/Interpersonal%20Communication%20and%20Relations/ACT_theory/
http://edutechwiki.unige.ch/en/Adaptive_control_of_thought_theory
http://act-r.psy.cmu.edu/wordpress/wp-content/uploads/2012/12/526FSQUERY.pdf
https://deeplearning4j.org/thoughtvectors
MATHS 315 Mathematical Logic course book, Prof David Gauld and Prof Andre Nies (2015)
http://www.numpy.org/
https://radimrehurek.com/gensim/
http://www.sympy.org/en/index.html
http://scikit-learn.org/
Cognitive Psychology and Its Implications, John R. Anderson (2015)
https://www.youtube.com/watch?v=izrG86jycck&list=PLg7f-TkW11iX3JlGjgbM2s8E1jKSXUTsG&index=1
https://www.youtube.com/watch?v=RNQK4MZ2_lE
http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec11.pdf
https://www.python.org/