Paraphrase Detection Using Recursive Autoencoder

Eric Huang
Stanford University
ehhuang@stanford.edu

ABSTRACT

In this paper, we tackle the paraphrase detection task. We present a novel recursive autoencoder architecture that learns representations of phrases in an unsupervised way. Using these representations, we are able to extract features for classification algorithms that allow us to outperform many results from previous works.

Keywords

paraphrase detection, computational semantics, autoencoders, deep learning

1. INTRODUCTION

Computational semantics is the study of the process of constructing meaning representations of natural language expressions. This is important in many high-level applications, such as translation, summarization, information retrieval, question answering, and communicating with computers using natural languages. Whereas statistical methods can perform satisfactorily depending on the quality the application requires, deep semantic understanding is required to have high-quality results.

Paraphrase detection is one of the difficult tasks where deep semantic understanding is required to achieve high performance. Paraphrase is defined as the restatement of a text or passages, in an alternative way. Paraphrase detection is important for applications such as summarization, information retrieval, information extraction and question answering, etc.

In this paper, We present a novel approach to learn phrasal representations using recursive neural networks. These phrasal representations are vectors in a n-dimensional semantic space, where phrases with similar meanings are close to each other. We extract features from these representations for use in this task. We use the Microsoft Paraphrase Corpus [1] for evaluating our method.

Many previous works have studied paraphrase detection. Most adopt carefully hand-engineer lexical or semantic similarity features or use heuristics. Dolan et. al. [1] uses string edit distance and a heuristic that pairs sentences from different stories in the same cluster. Islam and Inkpen [3] uses a modified version of the longest common subsequence string matching algorithm. Kozareva and Montoyo [4] evaluated three machine learning algorithms using several hand-designed lexical and semantic similarity features. Mihalcea et. al. [5] also used corpus-based and knowledge- based measures of similarity. Our approach differs in that we learn general task-independent phrasal representations in an unsupervised way, without hand-designing the representations. These representations could be easily adopted for other tasks, such as sentiment analysis. However, in this paper, we focus on paraphrase detection.

Fernando and Stevenson [2] proposes a similarity matrix approach, which inspired one of our methods for using the representations in this task. While they use similarity measures extracted from the WordNet dataset, we use the distance between our representations as the similarity measure.

Qiu et. al. [6] took a different approach that classifies paraphrases based on how dissimilar two sentences are. This is another method that could easily make use of our representations and could be explored in the future.

Collobert and Weston [7] created word feature representations using a deep neural network architecture. In our recursive autoencoder model, we bootstrap using these representations to construct phrasal representations.

2. RECURSIVE AUTOENCODER

In this section, we first present the learning problem we study. We then provide the background on autoencoders. Finally, we describe how to use recursiveness to obtain one representation for sentences of variable lengths.

2.1 Learning Problem

The learning problem we consider is as follows: Given a set of sentences of variable lengths, we want to construct a fixed n-dimensional representation for each sentence with the desired property that sentences that are closer in this n-dimensional space are more similar semantically.

2.1.1 Input Representation

Using a dictionary of size d, we represent a sentence of m words as a $d \times m$ matrix, M, where the i-th column is a d-dimensional vector with the entry corresponding to the dictionary index of the i-th word set to 1, and 0 elsewhere. Using this index matrix, we assign each word its continuous feature representation as in Collobert and Weston [7] by simply multiplying the two matrices,

$$X = LM$$

where the i-th column in L is the representation of the i-th word in the dictionary, M is the index matrix, and X is the $n \times m$ matrix representing the sentence. This X

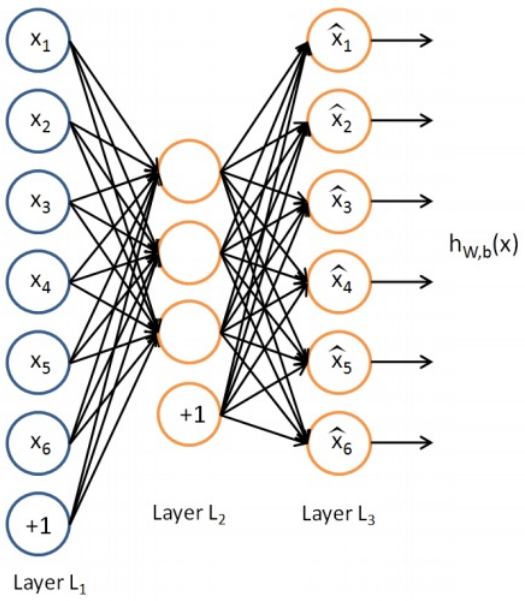

Figure 1: An autoencoder [9].

will be the final input to our model. Note that sentences of different lengths, m and m', would have matrices of different dimensions, $n \times m$ and $n \times m'$. Our model needs to construct a continuous n-dimensional representation from an $n \times m$ matrix for any positive-valued m.

2.2 Background: Autoencoder

The autoencoder learning algorithm is one approach to automatically extract features from inputs in an unsupervised way [9]. Similar to typical neural networks, it has layers of nodes and represents a hypothesis $h_\theta(x) = y$, where x is the input, y is the output, and θ are all the weights of the connections. The output is obtained by the feedforward process as in neural networks,

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)} \quad (1)$$

$$a^{(l+1)} = f(z^{(l+1)}) \quad (2)$$

The $a^{(l+1)}$ is a vector whose values are the activations of hidden nodes at layer $(l+1)$. The matrix $W^{(l+1)}$ are the weights of connections between layer l nodes and layer $(l+1)$ nodes. The vector $b^{(l)}$ is the bias term for layer l. f is an element- wise sigmoid-like function such as logit or tanh.

While typical neural networks are often used in supervised learning, an autoencoder is a neural network with a specific architecture that learns in an unsupervised way, by aiming to set output values to be equal to input values of each training instance i, i.e. $y^{(i)} = x^{(i)}$. In other words, the autoencoder learning algorithm tries to learn an identity function to reproduce its inputs. Specifically, an autoencoder has three layers: input layer, hidden layer, and output layer, where the output layer has the same number of nodes as the input layer (see Figure 1). The idea is that by using a smaller number of hidden nodes, as the network will learn to "encode" the inputs with the hidden nodes, then "decode" using the hidden nodes to reconstruct the inputs. If the number of hidden nodes are smaller than the number of input nodes, the activations of the hidden nodes would try to capture most of the information from the input nodes.

2.3 Recursive Autoencoder

One problem with computing representations for sentences is that sentences have variable lengths. The model needs to produce fixed-dimensional representations from inputs of different dimensions. Our approach to the problem is to apply autoencoders recursively to the input. Specifically, we use an autoencoder with $2n$ input nodes, n hidden nodes, and $2n$ output nodes. The autoencoder collapses the sentence by first taking two neighboring words, concatenating the two n-dimensional vector representations to form one $2n$-dimensional input vector to the autoencoder. After applying the encoding process, the activations at the hidden layer will be an n-dimensional vector representing the two words jointly. Then, we replace the two words with this joint representation and repeat until there is only one representation for the entire sentence (see Figure 2). Note that because the representations of the sub-phrases constructed in this process have same dimensions as the single word representations, the autoencoder can join two words, one word and a phrase, or two phrases together. We leave the discussion of the different ways by which we make the decisions for the order the words and phrases are joined until a later section.

2.4 Learning

Let T_i be the set of all non-leaf nodes in the tree structure for sentence i. We define the cost function for each sentence i to be the sum of reconstruction errors at each node of its tree structure:

$$J^{(i)}(W, b) = \sum_{d \in T^{(i)}} \frac{1}{2} \| [c_1^{(i)}; c_2^{(i)}]^{(d)} - h_{W,b}([c_1^{(i)}; c_2^{(i)}]^{(d)}) \|^2$$

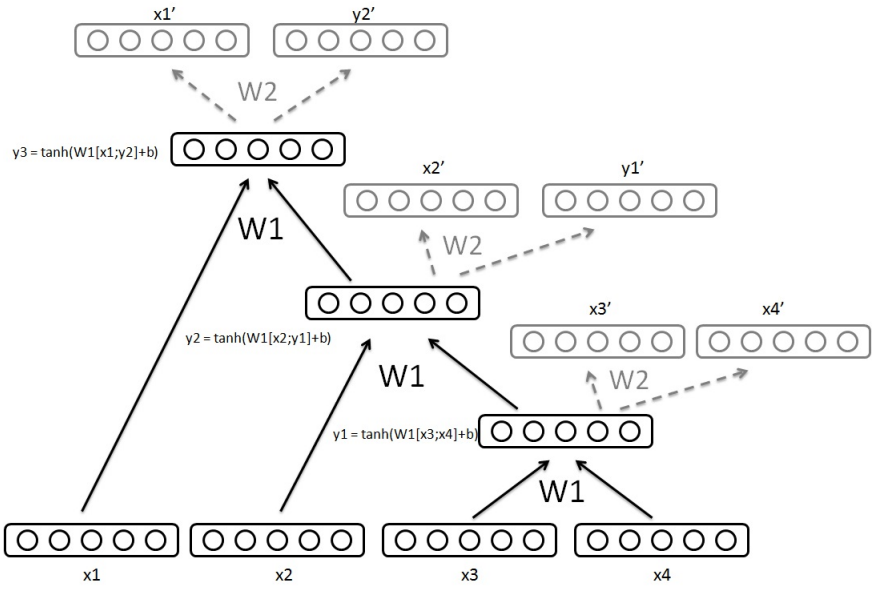

Figure 2: An example RAE tree of a four-word sentences. Autoencoders are stacked on top of one another where the hidden node activations computed by the lower ones are used as inputs to higher autoencoders.

where $c_1^{(i)}$ is the representation of the left child, and $c_2^{(i)}$ is that of the right child. $[c_1^{(i)}; c_2^{(i)}]^{(d)}$ is the concatenation of the representations of the two children and is the input to the autoencoder at node d. $h_{W,b}([c_1^{(i)}; c_2^{(i)}]^{(d)})$ is the values at the output layer.

Our training objective is to minimize the sum of the costs for every training sentence i plus the regularization term:

$$\min_{W,b} \sum_i J^{(i)}(W,b) + \lambda \frac{1}{2}\|W\|^2$$

In order to minimize this objective, we efficiently compute the gradients using backpropagation through structure. For updating W_2, we derived the following equations:

$$\begin{aligned}
\delta_r^{(d,i)} &= -([c_1; c_2]^{(d,i)} - a^{(3)(d,i)}) \bullet f'(z^{(3)(d,i)}) \\
\nabla_{W_2} J^{(d,i)} &= \delta_r^{(d,i)} (a^{(2)(d,i)})^T \\
\nabla_{b_2} J^{(d,i)} &= \delta_r^{(d,i)}
\end{aligned}$$

where the superscript (d,i) denote that the value is for node d in sentence i. The above is simply backpropagating the reconstruction error for which W_2 is only responsible. For W_1, we need to backpropagate errors coming from the node's parent, as well as the reconstruction error at the node because W_1 directly affects those errors. We have the following equations:

$$\begin{aligned}
\delta^{(2)(d,i)} &= (W_2^T \delta_r^{(d,i)}) \bullet f'(z^{(2)(d,i)}) + [\delta^{(p(d),i)}]_{1:n} \\
\nabla_{W_1} J^{(d,i)} &= \delta^{(2)(d,i)} ([c_1; c_2]^{(d,i)})^T \\
\nabla_{b_1} J^{(d,i)} &= \delta^{(2)(d,i)} \\
\delta^{(d,i)} &= (W_1^T \delta^{(2)(d,i)} + \\
&\quad ([c_1; c_2]^{(d,i)} - a^{(3)(d,i)})) \bullet f'([c_1; c_2]^{(d,i)})
\end{aligned}$$

where $[c_1; c_2] \in \mathbb{R}^{2n \times 1}$ is the input to the autoencoder at node d. $[\cdot]_{1:n}$ denotes the first n elements of a vector. $p(d)$ denotes the parent node immediately above node d. In the above equations, we consider d to be the left child of its parent. For the right child, we use $n+1:2n$, instead of $1:n$. The gradients are then summed over all non-leaf nodes and over all sentences plus the derivative of the regularization term:

$$\begin{aligned}
\nabla_{W_j} J &:= \sum_i \sum_{d \in T} \nabla_{W_j} J^{(i)} + \lambda W_j \\
\nabla_{b_j} J &:= \sum_i \sum_{d \in T} \nabla_{b_j} J^{(i)}
\end{aligned}$$

With these gradients computed, we use the L-BFGS algorithm to perform the optimization.

2.5 Determining Tree Structure

Here we discuss several ways for determining the tree structure for the recursive autoencoder.

2.5.1 *Greedy*

In the greedy approach, we first compute the reconstruction error of the autoencoder for each adjacent pair of words in the sentence. We choose the pair with the lowest reconstruction error, then replace those two words with the computed representation. We repeat the process until we have one representation. This method is greedy because it chooses the pair with the lowest reconstruction error first. The idea is that if we combine words for which we can most confidently represent, we would lose little information as we collapse the sentence.

2.5.2 *Global*

Another approach we implement uses the CKY algorithm to find the globally optimal tree structure for combining words. However, we have a very general grammar, $X \rightarrow XX$ where X can be any string. Essentially, we utilize the

bottom-up dynamic-programming approach to compute all possible tree structures and pick the one whose sum of reconstruction errors at all nodes is minimal. However, because the CKY is very computationally intensive, especially with the general grammar, we only keep one node with the lowest reconstruction error at each entry in the CKY table, instead of keeping all possible splits at that entry. This gives us a near-optimal tree structure, which is better than the greedy approach.

2.5.3 Parsed Tree

Because we are interested in capturing the semantics of the sentences rather than syntax, it makes sense to separate semantics and syntax by using parse trees of the sentences, determined by either humans or any off-the-shelf parsers. The intuition is that syntax often captures information blocks in a sentence, so it makes sense for the RAE to reconstruct phrase-by-phrase from the bottom up according to the parse tree. By relieving the responsibility to decide tree structures from the RAE, it can focus better on reconstructing the meaning of the sentence.

2.6 Cost Weighting

In the above section on learning, we define the cost function at each node to be the sum of the reconstruction errors of each input word/phrase. However, we may want to weight the errors of each input differently because often the two inputs do not carry the same weight in terms of the meaning of the combined phrase.

2.6.1 Weighting 1: Number of Leaf Nodes Below

In Weighting 0, we weight the error of the input based on how many leaf nodes are below it. Let n_1, n_2 be the number of leaf nodes under the left and right input respectively, and $recErr_1$, $recErr_2$ be the reconstruction error of the left and right input respectively. Cost at the node is weighted as:

$$\frac{n_1}{n_1+n_2}recErr_1 + \frac{n_2}{n_1+n_2}recErr_2$$

The idea is that the input representing more words should be more important and so the autoencoder should try harder in reconstructing that input versus the other.

2.6.2 Weighting 2: Average Counts of Leaf Nodes

One problem we might face with training is that words in the corpus follow a power-law distribution. Stop words such as 'the', 'a', etc. appear far more often than other words in training sentences. This presents a problem because in the training objective the reconstruction errors each word contributes is proportional to the number of occurrences in the corpus. That is, the more a word appears, the more times the RAE tries to reconstruct the word, the more error terms the word contributes to the objective. Thus, to minimize the objective, the optimization might make the RAE very good at reconstructing these more frequent words in order to make the overall cost lower.

Weighting 2 responds to this problem by weighting the reconstruction error of each input by the average of the counts of the words under the input. Let $c_1, ..., c_n$ be the counts of the n words under the left child, and $c'_1, ..., c'_m$ be the counts of the m words under the right child. The cost at each node is thus:

$$\frac{f_2}{f_1+f_2}recErr_1 + \frac{f_1}{f_1+f_2}recErr_2$$

where $f_1 = \frac{\sum_j c_j}{n}$ and $f_2 = \frac{\sum_j c'_j}{m}$. This weighting says that the reconstruction error of the input whose words are more frequent will be weighted less.

2.6.3 Weighting 3: Frequency of n-gram

In dealing with the same frequency problem described above, Weighting 3 weights each reconstruction error based on the count of the n-gram. Let c_1 be the count of the n-word sequence represented by the left- input, and c_2 the m-word sequence represented by the right-input. The cost at each node is:

$$\frac{1}{c_1}recErr_1 + \frac{1}{c_2}recErr_2$$

Because each n-gram will be reconstructed c times, this weighting will make each n-gram contribute to the overall cost with exactly one term, which is the average reconstruction error over all the errors for each reconstruction. Thus, the RAE tries equally hard at reconstructing each n-gram that appears in the corpus. To conserve memory, we only keep the counts for phrases with fewer than five words, and assign a count of one to longer n-grams.

3. DATASETS

In this section, we describe the two datasets that are used in our experiments.

3.1 Microsoft Research Paraphrase Corpus

We use the Microsoft Research Paraphrase Corpus (Dolan et al., 2004), which is a standard resource for the paraphrase detection task. There are 5801 sentence pair in the dataset, which are hand labeled by human judges whether the sentence pair are paraphrases of one another. The sentence pairs are selected from Web news sources specifically for evaluating paraphrase detection algorithms. Thus, many of the negative pairs are not completely unrelated sentences, but rather only exhibit subtle distances. Below is a positive pair that is clearly semantically equivalent,

- Amrozi accused his brother, whom he called the witness, of deliberately distorting his evidence.
- Referring to him as only the witness, Amrozi accused his brother of deliberately distorting his evidence.

One sentence in a positive example could contain extra information that's not in the other sentence:

- They had published an advertisement on the Internet on June 10, offering the cargo for sale, he added.
- On June 10, the ship 's owners had published an advertisement on the Internet, offering the explosives for sale.

Below is one negative pair example:

- Gyorgy Heizler, head of the local disaster unit, said the coach was carrying 38 passengers.
- The head of the local disaster unit, Gyorgy Heizler, said the coach driver had failed to heed red stop lights.

These examples illustrate the difficulty of this dataset and suggest that simple methods such as counting word overlap between sentences would not work well.

The data is unbalanced with 67% of the pairs being positive examples and 33% being negative examples. The dataset is split into the training set with 4,076 sentence pairs, and the test set with 1,725 sentence pairs. This partition is used by all previous published work on this data. We further split the training set into the training set we use in this paper with 3,261 sentence pairs, and the validation set with 815 sentence pairs, which is used to tune model parameters.

3.2 English Gigaword

In order to train the RAE to learn phrasal representations well, we use 152,487 sentences from the English Gigaword Corpus, in addition to the 6,522 sentences from the 3,261 training sentence pairs in the MSRP corpus. In the combined set of 159,009 sentences, there are a total of 50,002 words that can be found in the vocabulary used by Collobert and Weston [7]. Other words are mapped to the UNKNOWN token.

4. DETAILS OF TRAINING THE RAE

We choose to use *tanh* as the sigmoid-like activation function f in the autoencoder because its range includes both positive and negative numbers, which are both found in the word representations. We use the 100-dimensional word representations learned by Collobert and Weston [7] and we pre-processed this lookup table of word representations by applying *tanh* to each value, making each value to be in the range of -1 and 1, consistent with the values of the nodes in the autoencoder. To parse the sentences in the corpus for use in the parsed tree approach for obtaining the tree structure, we used the Stanford Parser [8].

Because the RAE is very computationally intensive and the training set contains a large number of examples, in order to speed up convergence, we train the RAE using batches of the training sentences. Specifically, we split the training set into batches of 5,000 sentences each. We train on the first batch of sentences using L-BFGS for twenty iterations. Then, we move on to use the next batch of 5,000 sentences. We repeat this process in a round-robin fashion until each batch has been trained on twice, equivalent to a total of forty iterations on each batch. As we train the RAE, we examined the reconstruction error of the validation set and observed that the value indeed decreases in number of iterations until convergence, suggesting that this batch training method is approximately minimizing the objective.

5. QUALITY OF LEARNED PHRASAL REPRESENTATIONS

Before we dive into evaluating our model on the paraphrase detection task, we would like to first examine qualitatively what representations the RAE can produce, both as a sanity check that the RAE has desirable properties and so that we can better understand the model. To do so, we train the RAE as described above. With the trained parameters, we construct representations for each phrase as determined by the parse tree in the training set we used. We compute the euclidean distance between each pair of the representations. Below are some phrases and their nearest neighbors listed in order:

- the U.S.
  - Weighting 0: the French, the Swiss, the Japanese
  - Weighting 1: the French, the Swiss, the Chinese
  - Weighting 2: a U.S., the second biggest U.S., the most experienced U.S.
  - Weighting 3: a U.S., a 36-year-old U.S., a recent U.S.

  We see that Weighting 0 and 1, which do not take frequency of words into account, matches "the U.S." with "the <country>". From the single-word representations, the RAE knows the countries are closer together in the semantic space; however, it tries too hard to reconstruct "the", which really does not carry much meaning. With Weighting 2 and 3, which do take frequency of words into account, the RAE realizes that "U.S." is the more important word than "the". Thus, the nearing neighbors it finds are "<adjective> U.S.".

- the late
  - Weighting 0: the early, the next, the last
  - Weighting 1: the early, the next, the last
  - Weighting 2: a late, ... in a joint statement late, the day 's late
  - Weighting 3: a late, a 26-point lead late, a wicket late

  Similarly, for "the late", the frequency weightings place more weight on "late", which is the more significant word than "the" in the phrase.

- executive director
  - Weighting 0: executive editor, executive secretary, council director
  - Weighting 1: executive secretary, executive editor, council director
  - Weighting 2: council director, general director, assistant director
  - Weighting 3: council director, assistant director, Managing director

  In this example, we observe that Weighting 0 and 1 finds nearest neighbors that match the first word, "executive", while nearest neighbors found by Weighting 2 and 3 match the second word, "director". This might be an interesting artifact of the frequency weighting: because stop words like "the" and "a" are often in the beginning of a phrase, the RAEs with frequency weighting learn to weight the second input more in the combined representation.

- began Wednesday
  - Weighting 0: began Monday, began Friday, began Tuesday
  - Weighting 1: began Friday, began Monday, began Tuesday
  - Weighting 2: began Friday, began Monday, began Tuesday

- Weighting 3: arrived Wednesday, *UNKNOWN* p.m. Wednesday, claimed Wednesday

Nearest neighbors using Weighting 2 and 3 are mostly the same except in some examples. The above example shows that Weighting 3 chooses nearest neighbors based on the day in the week, while Weighting 2 chooses by the action.

- Fourteen people were wounded when a war plane fired a missile at the town Wednesday
  - Weighting 0: Fourteen people were killed and 150 wounded at the mosque and in unrest that spilled over to other parts of the city / Fourteen people were killed and 150 wounded in the violence / Fourteen people were killed and over 150 wounded in the worst internal violence since self-rule began in May
  - Weighting 1: Fourteen people were killed and 150 wounded at the mosque and in unrest that spilled over to other parts of the city / Fourteen people were killed and 150 wounded in the violence / Fourteen people were killed and over 150 wounded in the worst internal violence since self-rule began in May
  - Weighting 2: Fourteen people were killed and 150 wounded at the mosque and in unrest that spilled over to other parts of the city / Fourteen people were killed and over 150 wounded in the worst internal violence since self-rule began in May / Seventeen people were killed , including a prominent political figure , in the latest wave of sniper shootings in Pakistan 's troubled business capital
  - Weighting 3: Fourteen people were killed and 150 wounded in the violence / Fourteen people were killed and 150 wounded at the mosque and in unrest that spilled over to other parts of the city / Fourteen people were killed and over 150 wounded in the worst internal violence since self-rule began in May

This examples shows that our models can capture meaning even for longer phrases. However, since it is clearly more difficult to obtain good representations for longer phrases, many of the nearest neighbors do not look quite as good as those for the bigrams and trigrams.

6. CLASSIFICATION APPROACHES

Using the RAE, not only do we have the representation for the entire sentence, but we also obtain representations for each of the subphrase in the tree structure. The question now is how we can use these representations for paraphrase detection. Again, since the sentences in each sentence pair have variable lengths, we need some ways to extract the same number of features for different sentence pairs. We do so by two approaches.

6.1 Aggregate Features

The first method is to aggregate the representations into one single feature. For each sentence, we consider taking the average, maximum or minimum of each element in a group of representations. For example, given a group of n representations, $\{X^{(1)}, X^{(2)}, ..., X^{(n)}\}$, the average feature would be $Y = [y_1, y_2, ..., y_d]$ where $y_j = \frac{x_j^{(1)} + x_j^{(2)} + ... + x_j^{(n)}}{n}$.

There are four groupings we consider. The first is only using the representation of the top node, which is the representation for the entire sentence after the RAE collapses it. The second group is using all leaf nodes, which are the single-word representations we get from Collobert and Weston. The third is using all non-leaf nodes. These are the subphrase representations computed by the RAE in the process of collapsing the sentence. The final grouping is using representations of all nodes in the tree structure.

6.2 Similarity Matrix

The other method we use is adopted from Fernando et. al. [2]. We use a similarity matrix to calculate the distance for pairs of representations between words and phrases in a sentence pair. Given sentences $s^{(1)}$ and $s^{(2)}$ in a sentence pair, we compute representations of words and subphrases for each sentence to get $\{{s_1}^{(1)}, {s_2}^{(1)}, ..., {s_{2n-1}}^{(1)}\}$ and $\{{s_1}^{(2)}, {s_2}^{(2)}, ..., {s_{2m-1}}^{(2)}\}$, where n and m are the lengths of $s^{(1)}$ and $s^{(2)}$ respectively. We then remove the words that appear over some threshold of times in the corpus, because these words do not carry much meaning. We found the threshold with best performance to correspond to the ten most frequent words in the dictionary, which includes "the", "a", etc. We then construct a matrix W where W_{ij} is the Euclidean distance between the representations of ${s_i}^{(1)}$ and ${s_j}^{(2)}$.

Because we want similar words or phrases to have a number close to 1 in the matrix and close to 0 for dissimilar words or phrases, we transform each entry by applying the function $f(n) = 2sigmoid(-n)$ so large n approaches 0 and small n approaches 1. For pairs that are really dissimilar, these are words or phrases that have little relation to each other, so we do not want them to contribute to the final similarity score for the sentence pair. Therefore, we apply another threshold that if the value for a pair is below the threshold we set it to 0. We try 200 threshold values for the cutoff, so as to retain as much information as possible. Finally, we sum up the entries in the matrix and normalize by the number of words in the sentence pair. In the end, we have 200 similarity numbers that are used as features to classification algorithms.

7. RESULTS

For the aggregate features, we used SVMs as the classifier. For similarity matrix, we used logistic regressions. For all settings, we used Weighting 3, which we judge to have best quality nearest neighbors. We cross-validated using the validation set that was split from the original training set to choose the best-performing parameters and features. The best-performing aggregate feature reported is the average of all nodes in the tree structure. Then, we train on both the training set and validation set, and evaluated the performance on the test set. Table 1 summarizes our results.

We compare our models against three baselines. The first is the naive baseline which simply predicts every sentence pair to be paraphrases. This achieves 66.49% accuracy because the test set is unbalanced. The second baseline uses the aggregate feature of the average of only the leaf nodes, which achieves 68.06%. This suggests that using the single-word representations alone provides signal for this task. This

	Accuracy
Baseline (all positive)	66.49%
Leaf Node only - Aggregate	68.06%
Random RAE - Aggregate	68.12%
Greedy - Aggregate	68.57%
CKY - Aggregate	68.75%
Parsed - Aggregate	70.55%
Leaf Node Only- SimMatrix	73.04%
Parsed - SimMatrix	73.33%

Table 1: Testing accuracies.

	Accuracy
Mihalcea et al. (2006)	70.30%
Rus et al. (2008)	70.60%
Qiu et al. (2006)	72.00%
Islam and Inkpen (2007)	72.60%
Fernando and Stevenson (2008)	74.10%
Wan et al. (2006)	75.60%
Kozareva and Montoyo (2006)	76.60%
Parsed - Aggregate	70.55%
Parsed - SimMatrix	73.33%

Table 2: Previous results.

is similar to counting the number of word overlaps in the sentence pair, but more flexible because it uses the distance between the words instead of a binary match. The third baseline is a RAE with randomly initialized parameters, using the average of all nodes aggregate feature. This performs slightly better than using only leaf nodes, but could be just due to chance.

Using the RAE to create the tree structures for sentences with the greedy and CKY approaches produces better results than all three baselines. However, when examining the trained parameters, we noticed that the encoding and the decoding matrices are heavily skewed. That is, either the right half or the left half has much bigger norms than the other side for the encoding matrix, and the top half or the bottom has bigger norms than the other in the decoding matrix. This causes the RAE to produce entirely left-to-right or right-to-left trees, depending on the initialization of the parameters. This might be because these approaches allow the RAE to freely choose the tree structure, the optimization causes the RAE to focus on always reconstructing one of the inputs really well in order to get a low overall reconstruction error.

Observing this problem, we use the Stanford Parser to parse the dataset and fix the tree structure of each sentence, so that the RAE can focus more on the semantics instead of the syntax. With this pre-parsing, we get a boost in performance to 70.55%.

Using the similarity matrix method, we see large improvements on accuracy up to 73.33% because it is able to retain more information in the representations that the RAE produces. The aggregate feature approach, which simply take the average of the representations, could potentially lose a lot of information, as we can imagine two very far representations losing its characteristics when we take the average. On the other hand, the similarity matrix compares the representations of each word/phrase pair and is able to keep more information.

Note that when we use features that include the representations of the non-leaf nodes, the performance beats when we only use the representations of the leaf nodes for features. This suggests that the phrase representations produced by the RAE model does provide signal for the paraphrase detection task.

Finally, Table 2 compares our results with previous works'. Although we are not able to achieve state-of-the-art performance on this dataset, our method beats many of results from previous works.

8. CONCLUSION

In this paper, we presented a novel method for learning phrasal representations in semantic space in an unsupervised way. We show that these representations are decent by examining their nearest neighbors. We extract features from these representations for the paraphrase detection task and outperform several previous works' methods.

8.1 Future Work

There are many possible directions for future work. First, the phrasal representations for longer phrases are still not as good as we hope them to be. We would like to try other tweaks, such as other weighting schemes or training objectives. Second, we would like to investigate more effective ways to extract features from the representations to achieve higher accuracy. Finally, we would like to explore other tasks where these representations could help.

9. COLLABORATION STATEMENT

This is joint work with the CS294 project. This work is done in collaboration with Jeffrey Pennington, along with

valuables advice from Professor Andrew Ng and Richard Socher. The original idea of the RAE comes from Richard, while Jeffrey and I jointly implemented the framework for the RAE. Jeffrey went on to focus on sentiment analysis using the RAE, while I focused on the paraphrase detection.

10. REFERENCES

[1] Dolan, B., Quirk, C., and Brockett, C. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources Proceedings of the 20th international conference on Computational Linguistics (COLING 2004), Geneva, Switzerland, pp. 350-356.

[2] Fernando, S., and Stevenson, M. A semantic similarity approach to paraphrase detection Computational Linguistics UK (CLUK 2008) 11th Annual Research Colloquium.

[3] Islam, A., and Inkpen, D. Semantic similarity of short texts Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2007), Borovets, Bulgaria, pp. 291-297.

[4] Kozareva, Z., and Montoyo, A. Paraphrase identification on the basis of supervised machine learning techniques Advances in Natural Language Processing: 5th International Conference on NLP (FinTAL 2006), Turku, Finland, 524-533.

[5] Mihalcea, R., Corley, C., and Strapparava, C. Corpus-based and knowledge-based measures of text semantic similarity Proceedings of the National Conference on Artificial Intelligence (AAAI 2006), Boston, Massachusetts, pp. 775-780.

[6] Qiu, L. and Kan, M.Y. and Chua, T.S. Paraphrase recognition via dissimilarity significance classification Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 2006), pp. 18-26.

[7] Collobert, R. and Weston, J. A unified architecture for natural language processing: deep neural networks with multitask learning In ICML, pages 160-167, 2008.

[8] Klein, D. and Manning, C. Accurate unlexicalized parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

[9] Andrew Ng. Sparse autoencoder (CS294A Lecture notes).