Psych 186B Final Project Write-up

Categorization of Academic Articles

By: Jason Carpenter

How can we classify an academic article if we don’t know the title or have the keywords? How can we find the top 30 most similar articles to a prototype article that we have great interest in? The aim of the present project is to answer these questions and more by implementing neural networks which can categorize academic articles into 6 pre-set subcategories of Cognitive Psychology: attention, memory, language, perception, reasoning and sleep. However, it took a lot more than simply creating neural networks to get the project to work. In the present work we will explain the entire process involved in categorization including pre-processing the articles, representing the words as features, and finally implementing the two types of neural networks, perceptron and self-organizing map.

In our pre-processing stages we took several steps to convert articles from pdf form into uniformly formatted text words with associated relative word frequencies. We began by using the online pdf to txt converter located at <http://document.online-convert.com/convert-to-txt>. Next, once we had txt files instead of pdfs, we used the porterStemmer function, provided to us by Majed. This function removes suffixes from words, converts them to lowercase, and generally makes all similar words in a uniform format, leaving only the root word behind. For example, both of the original words “attention” and “attentive” are stemmed to the root word “attent.” After we used porterStemmer on each txt file, it was time to find the core of our representation, the word frequencies. We used a function called wordcount2, created by Suri Like and Lee White, downloaded from <http://www.mathworks.com/matlabcentral/fileexchange/37768-wordcount2/content/wordcount2.m>. This function reads through every word in a txt file and returns the absolute frequency and relative frequency of the top X amount of words, with X specified by the user. We set X to 20, retrieving the top 20 most frequently occurring words in each article. It is also important to note that we edited the wordcount2 function to remove commonly occurring words such as “the,” “and,” and “of” from the list of top 20 most frequently occurring words. Now, we were finally done with our pre-processing stage, with a list of the top 20 most frequently occurring, uniformly formatted words and their associated word frequencies. We did also try to implement one other pre-processing function known as word2vec, but without success. Word2vec is a function that constructs vocabulary from training data, then computes vector representations of each word in the data, and calculates the cosine similarity of words based on meaning. This would have been beneficial to us because then we would have been able to take semantics into account instead of simply using word frequency. Regardless, we were unable to find a MATLAB implementation of word2vec and thus we continued our project without it. With our pre-processing stage done, we were now ready to move onto feature representation.

In order to represents the features of our articles in a form that neural networks could understand, we had to take these lists of the top 20 most frequently occurring words in each article and convert them into a matrix format that our neural networks could work with. In our createFeatures function, we created a dictionary of all frequently occurring words from all articles, without duplication of words, and concatenated the columns such that the rows of the matrix were unique words (i.e. our features) and the columns were the different articles. The feature type that we chose for our feature representation was relative frequency in order to control for different file lengths. This way, if a file had the stemmed word “attent” 54 times out of 1000 words it would return the same relative frequency as a file that had “attent” 540 times out of 10000 words. In Figure 1, there is a visual representation of our feature matrix. By looking at Figure 1, it is easy to tell that the first 10 articles listed are attention articles due to the high relative frequency of the root word “attent” in each of the first 10 columns.

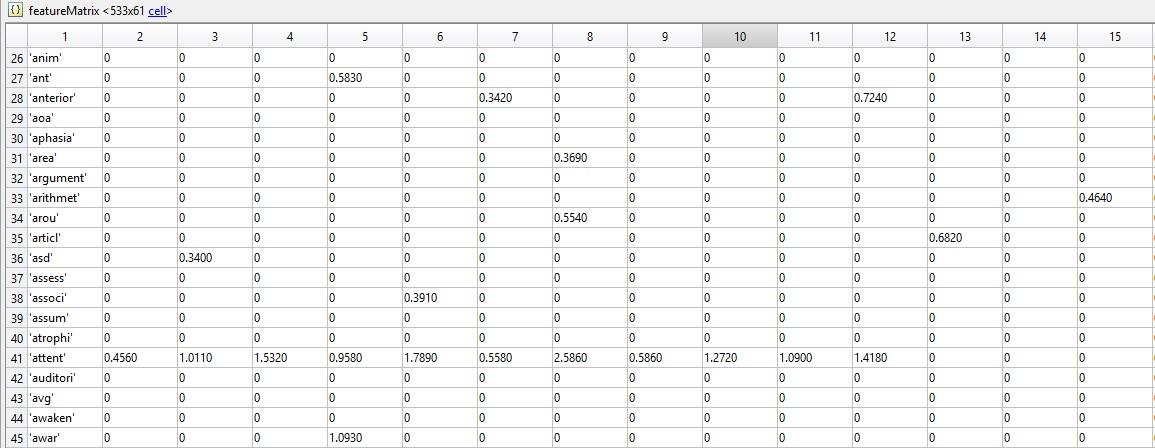


Figure . The feature matrix, with each row a unique word and each column a unique article. The values are relative frequency of that word in that article. The first 10 articles listed here are clearly attention papers, given the high relative frequency of the root word “attent.”

However, one other thing becomes blatantly clear by looking at Figure 1: there is a lot of zeros! Our feature matrix was very sparse and contained a lot of elements which did not add any information to the overall network. It is for this reason that we used a technique known as Principle Component Analysis (PCA). PCA is a method which is used to emphasize variation and bring out the patterns in the dataset. PCA reduces the dimensionality of a matrix with minimal loss of information. Starting out with a 812(words) x 120(articles) feature matrix with tons of zeros, applying PCA reduced the dimensionality to a 125(principle components) x 120(articles) PCA matrix with little to no zeros. This new PCA matrix was finally in a form that we could use with our neural networks.

The first neural network that we implemented for classification of articles was the perceptron with backpropagation. In our perceptron, we used 3 layers: 1) input layer of dimensionality = number of features (125 units), 2) hidden layer of dimensionality = half the number of features (62 units), and 3) output layer of dimensionality = number of categories (6 units). For our error reduction criteria, we required that the sum of squared errors (SSE) of the training set be reduced to 0.01 in less than 1000 epochs (it never reached 1000 epochs, but usually converged around approximately 300 epochs). The final classification of each article depended on the output unit with the highest activity. So even if the activity of an output unit was only 0.2, but it had the highest activity of any output unit, then the article was classified into that category. We will analyze the results from three different variations of training vs testing set sizes: 1) 90% training, 10% testing, 2) 50% training, 50% testing, and 3) 25% training, 75% testing. We ran the 90% training, 10% testing set 10 times so as to test all 120 articles in our database. Figure 2 shows example outputs for 2 articles per category (2x6=12, 12/120=10%).

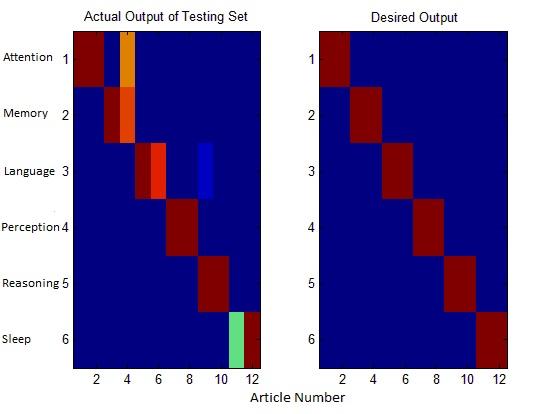


Figure . Actual output compared to desired output for 12 articles (2 per category x 6 categories) of the perceptron when trained on 90% and tested on 10% (12/120).

In Figure 2, though we did not get a perfect classification (where each appropriate cell is completely dark red and the rest dark blue), in each column (article), the row (category) with the highest output (most close to dark red) is the correct row when compared to our desired output. The average SSE across all 10 iterations of running the 90% training x 10% testing perceptron was AVG SSE = 1.9006. The overall accuracy across all 10 iterations was an astounding 115/120 = 95.8333% correct. This high accuracy was to be expected given that the training set was so large. Next, we looked at how the accuracy would fare when we trained on 50% and tested on 50%. We ran the 50% training, 50% testing set twice so as to test all 120 articles in our database. Figure 3 shows example outputs for 10 articles per category (10x6=60, 60/120=50%).

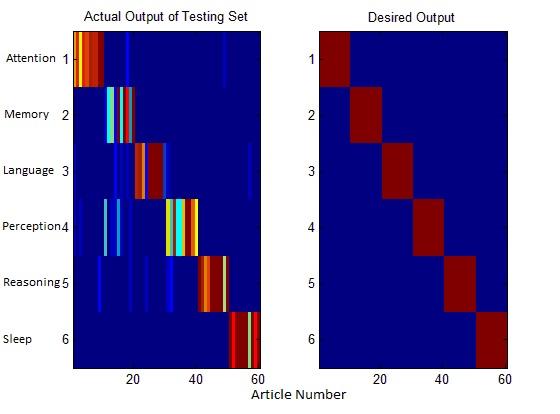


Figure . Actual output compared to desired output for 60 articles (10 per category x 6 categories) of the perceptron when trained on 50% and tested on 50% (60/120).

Once again, looking at Figure 3, we see a failure to perfectly classify each article into the appropriate category, but most of the time, the category with the highest output unit activity was the correct category as compared to the desired output. The average SSE across both iterations of running the 50% training x 50% testing perceptron was AVG SSE = 8.694, increased from the 90% training x 10% testing due to the larger number of articles in the testing set. The overall accuracy across both iterations was once again an amazing 115/120 = 95.8333% correct. This high accuracy proved the robustness of our perceptron and its ability to correctly classify novel articles given a limited amount of training information. Finally, we attempted to break our neural network down by training the perceptron on 25% of the articles and testing of 75%. We ran this for 4 iterations so as to evenly test all papers in the database. Figure 4 shows example outputs for 15 articles per category (15x6=90, 90/120=75%).

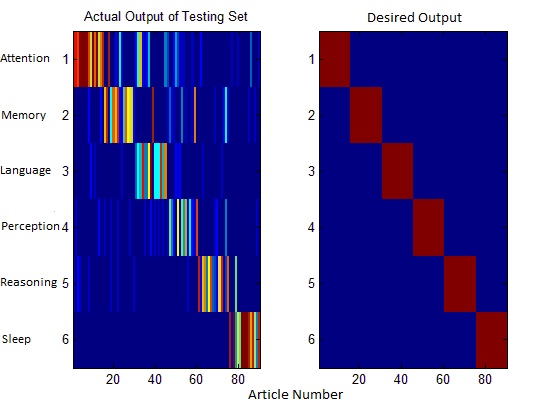


Figure . Actual output compared to desired output for 90 articles (15 per category x 6 categories) of the perceptron when trained on 25% and tested on 75% (90/120).

Based on looking at Figure 4, it appears as though our system is finally beginning to break down. However, when we look more closely we realize that once again, very often the correct category is the one with the highest output unit activity. Though the highest output unit activity is probably very low for many of the light blue cells, most of the time it is still higher output unit activity than the other categories for that article. The average SSE across all four iterations of running the 25% training x 75% testing perceptron was AVG SSE = 30.825, increased greatly from the 50% training x 50% testing due to the larger number of articles in the testing set and the high error behind actual and desired outputs even though categorization didn’t decrease so greatly. In fact, the overall accuracy of the 25% training x 75% testing perceptron was 310/360 = 86.1111% correct. Though the accuracy did decrease significantly from the 90x10 and 50x50 iterations, this is still an incredibly high accuracy for such a small training set. Ultimately, our perceptron proved highly robust, despite our best attempts to break it down. Another interesting point to note is that the errors that our perceptron did make were usually representative of the subject matter of the paper that was incorrectly classified. A key example of this is language4.txt, which was classified as a memory article by every variation of our perceptron. One might think that our perceptron incorrectly classified the article and thus failed to do its job. However, we find it to be more likely that we incorrectly labeled language4.txt as a language paper because the title is “Verbal declarative memory impairments in specific language impairment are related to working memory deficits.” We will return to language4.txt during our discussion of the self-organizing map.

The other neural network that we implemented was the self-organizing map (SOM). A SOM is an unsupervised machine learning technique incorporating a network of neurons (1D or 2D), initialized with random weights on the range [0,1] such that the size of each weight vector is the number of features (125 principle components). For each epoch, the input vector is compared to the weight vectors to find the most similar neuron and then the weight vectors of the winning neuron and nearby neighbors are updated (with a neighborhood size that decreases with every epoch). This procedure is repeated with all input vectors until the map stabilizes, producing a topographical map with clusters representative of different categories. Figure 5 shows an example of the topographical map with 6 clusters, each representing a different subcategory of Cognitive Psychology, once the map has stabilized. Each cluster (category) appears in a different region of the overall map and is formed solely based on the 125 principle component input units for each article.

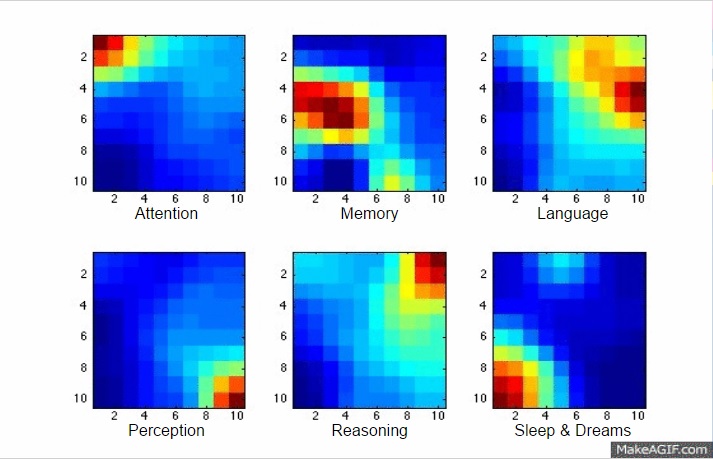


Figure . The final state of the self-organizing topographical map once it has stabilized into 6 distinct categories, each representing a different subcategory of Cognitive Psychology: attention, memory, language, perception, reasoning, and sleep.

In our self-organizing map, we used two primary sizes for our lattice of neurons (10x10, 15x15). The weights matrix was the number of principle components features by the number of neurons (125x100 and 125x225, respectively). The learning rate began with eta0 = 0.1 and decreased exponentially with a rate etaN = eta0\*exp(-i/1000). Our input vector was the PCA matrix (125x120), and we ran the SOM for 1000 epochs before assuming that it had converged to a stable map. Our output was in the form of a topographical map, a result of the input matrix transposed multiplied by the weight matrix. We adapted our SOM code from Jason Yu-Tseh Chi, accessed from the internet at <https://chi3x10.wordpress.com/2008/05/08/som-self-organizing-map-code-in-matlab/>.

In order to classify the SOM outputs into the 6 pre-set categories we used an exemplar technique, in which we create exemplars for each category by taking the average output for all papers in that category. Then, we calculate the difference vectors between each article and each exemplar and chose the category with the smallest difference between exemplar and article. Using this exemplar technique, we reached a very high accuracy of 114/120 = 95% correct for both our 10x10 and 15x15 lattices of neurons. Figure 6 shows the SOM topographical map outputs for 10 articles per category for all categories.

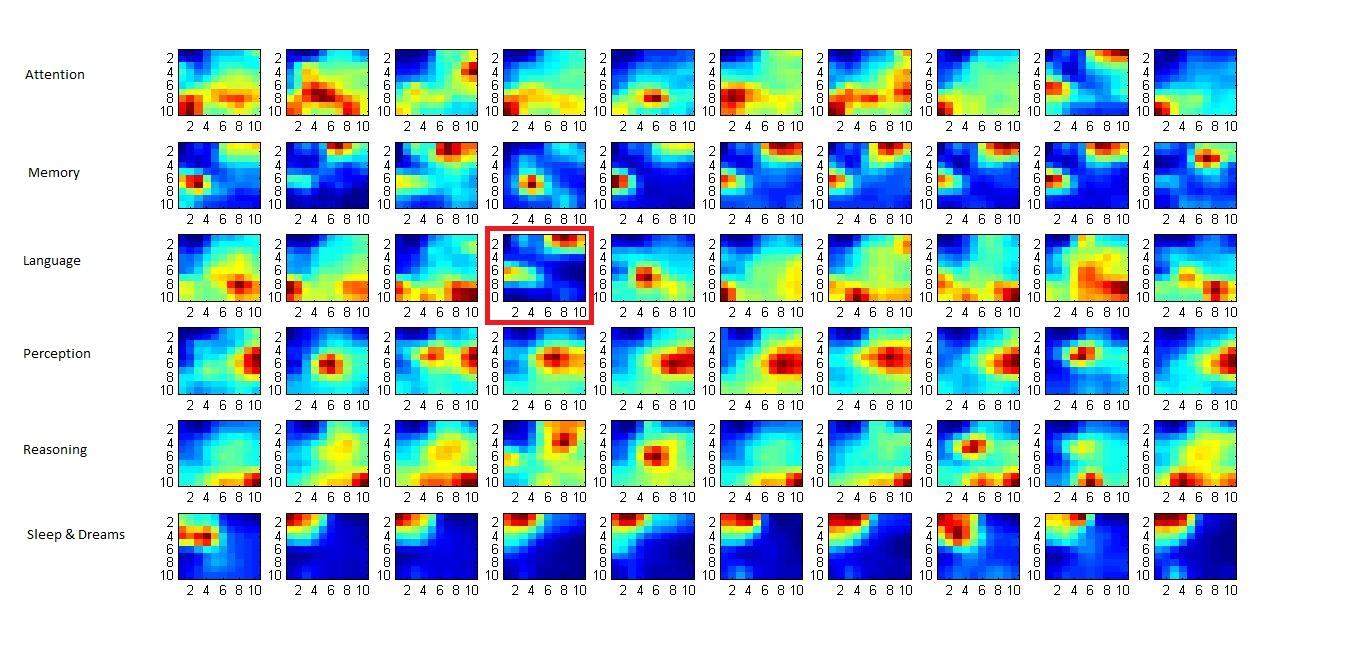


Figure . The SOM topographical map output for 10 articles per category for all 6 categories. The box highlighted in red is language4.txt, which appears to have output more similar to a memory paper than a language paper.

Looking closely at the box highlighted in red, we return to our misclassified culprit, language4.txt. Once again, we see that language4.txt is more representative of a memory paper (the outputs of the second row) than a language paper (the outputs of the third row). It is very interesting that our self-organizing map took the features of language4.txt and using unsupervised machine learning, clustered it in with the memory articles. This once again adds evidence to support the likelihood that we mislabeled language4.txt as a language paper when in fact it should’ve been a memory paper. Thus, even the “errors” of the SOM and perceptron are sometimes more representative of the actual subject matter of the article than even human labelling, simply by using the relative word frequencies in the paper. Another way to view the topographical map of the SOM is by looking at maximum activity plot, as seen in Figure 7, which displays the neuron indices at which the highest output activity was found for each article.

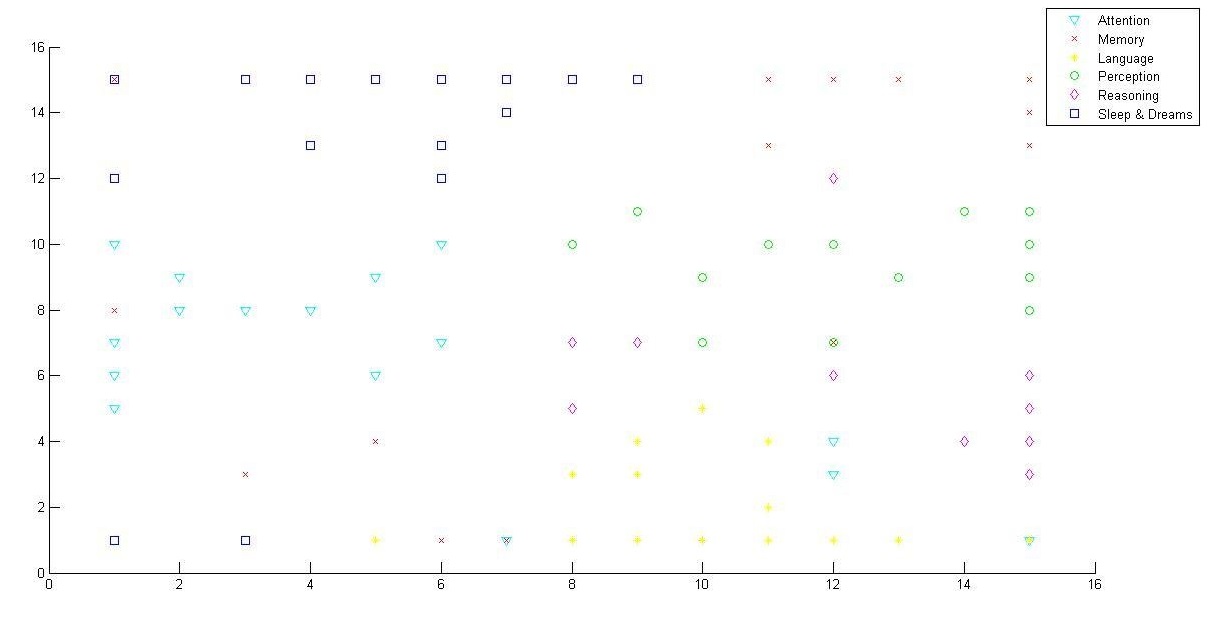


Figure . The maximum activity plot for a 15x15 lattice of neurons, displaying the neuron indices at which the highest output unit activity was found for each article.

In Figure 7, we can see that for the highest output unit activity, the SOM placed the clusters in different locations, with little overlap. The SOM effectively clustered all 120 articles into their own categorized regions of the overall topographical map, without knowing what category each article was in and using solely the principle components (based on relative word frequency) as input. We also found another use for the SOM based on the way each output is clustered into an area that represents the features of the article.

Instead of classifying articles into categories based on knowledge of what categories they already exist within, we decided to try to take a novel article and find the top 30 most similar articles to that article within the database based on the SOM representation of the article. The example article that we used was Sleep5.txt, an article about sleep and episodic memory. By calculating the minimum difference between the SOM output matrices between every article and Sleep5.txt, we found the top 30 most relevant articles within our database, pictured in Figure 8.



Figure . Top 30 most relevant articles to Sleep5.txt based on calculating the minimum difference between the SOM outputs for each article and Sleep5.txt.

Interestingly enough, the top 20 most relevant articles were all sleep papers, the category of the original paper, and the top 21-30 most relevant articles were all memory papers (or language4.txt, a paper that probably should be classified as a memory paper), the category which contains the second most important keywords. Thus, if we had any article that we really liked for purposes of a research project and wanted to find the most similar articles to that article from within the database, we could simply choose which one to run and the SOM would help us find exactly what we’re looking for, based solely on the frequency of words in the articles. This technique greatly increases the utility of our SOM and further supports the effectiveness of our neural network. However, just like for the perceptron, we had to see if we could break down our SOM.

In an attempt to break down the SOM neural network, we measured the accuracy of its classification when varying the lattice dimension size. In order to report a classified object as correct, we once again compared the object to the exemplars made up of the average of all the papers in that category. Figure 9 shows a plot of the accuracy of the SOM classifications at different lattice sizes.

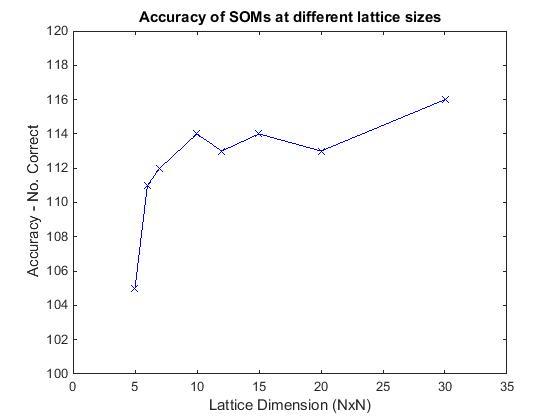


Figure . Accuracy of SOM classifications are different lattice dimension sizes (5x5 – 30x30)

The standard lattice sizes (10x10 and 15x15) performed pretty well, reporting 114/120 = 95% correct classifications. However, when we went to the ends of the spectrum is when we saw the interesting results. At 5x5 lattice dimension sizes we really saw a breaking down of the system, at only 105/120 = 87.5% correct. Though this number is much lower than our standard lattice size SOM accuracy at 95%, 87.5% is actually not that bad and once again supports the robustness of our neural network. A 30x30 lattice dimension size actually proved to be the most accurate, at 116/120 = 96.666% correct. This was a surprise as we thought that having too many dimensions would lead to inaccuracies due to the small neighborhood size. Next, we will discuss the shortcomings of our models.

The limitations of our models are inherent as a result of our methods. First, by using frequency of words we failed to take advantage of the semantic meaning behind the words and thus limited ourselves to the number of times a word appeared. This means that we could replace the very common word “attent” with the word “x123” and if it continued to appear in the same papers then our system would still classify them together, even though the word has no relevance to the topic. Additionally, we had to manually change the wordcount2 code to remove common, unimportant words such as “the” and “of,” which means we could’ve missed some common unimportant words that we failed to remove. Also, by choosing only the top 20 most frequent words in each article, we could be failing to incorporate some important, but less commonly used words that occurred frequently across articles of the same category. Second, in order to categorize outputs we were required to know which papers belong to which category beforehand. While this isn’t a big deal for the perceptron as that involves supervised learning, it is a huge shortcoming for the SOM as it takes a beautiful unsupervised learning process and turns it into a semi-supervised learning process. We tried to use K-means clustering (a method for partitioning all our articles into 6 clusters where each article belongs to the cluster with the nearest mean) to classify our articles once we had the topographical map outputs, but it was not effective, so we had to stick with the semi-supervised method. Third and finally, in our SOM we represented differences between articles and exemplars with a single value, losing topographical information. While this still represents how different the articles and exemplars are, it loses information about in what way the articles and exemplars differ.

Over a period of 3 weeks, our group successfully produced two working neural networks for classification of academic articles: a perceptron and a self-organizing map. Lots of pre-processing steps were involved, but once we got the data in the form that we wanted, we were ready to rock ‘n’ roll. The perceptron proved to be very accurate, and highly robust to using a small training set. The SOM came out with nearly equally high accuracy and a robustness to high and low lattice dimension sizes. Overall, we are very proud of our accomplishments and owe it all to our Professor, Zili Liu and our TA, Majed Samad.