Implementation:

* The model class takes in the window length and word size as parameters, to be able to test different network configurations.
* We derived the training and back propogation equations in matrix-vectorial form. This helped us to use EJML library api to implement all the steps. All weight matrices were extended by one column, input vector and intermediate vectors were extended by one row to accommodate bias terms within the matrices.
* Initialization: We initialized the weights matrices (W and U) using the formula provided in the handout. The faniIn value is the number of columns and fanOut is the number of rows of the weight matrices. We then zeroed out the first row of these matrices, to initialize the bias part to zeros.
* Inputs: The input words were grouped into sentences and then padded based on the window length. These padded sentences were then broken into training samples and then randomly shuffled.
* Feed forward: We implemented the feed forward step with matrix multiplications to get z and tanh nonlinear transformation to get a.
* Cost Function: We implemented the negative log likelihood and regularization cost to keep track of the cost during training.
* Back propagation: Next we implemented the error term calculations and the gradient calculation for the weights and the input vector.
* Gradient Check: We implemented the ObjectiveFunction interface and used the provided gradient check method to verify the gradient calculations.
* We also implemented a method to calculate accuracy and other metrics on the training data. We used this during training to verify that the total cost was decreasing and the training accuracy was increasing with each iteration.

Implementation (Extra Credit)

* Multi layer
* Mini Batch

Visualization of word vector. (Extra Credit)

We picked the first 6000 unique words from the training sample and used them as samples for the t-SNE tool to visualize this 50 dimensional data. T-SNE maps this high dimensional data to a 2-dimensional space by preserving local similarity between the samples. We used Python scikit-learn’s implementation of t-SNE

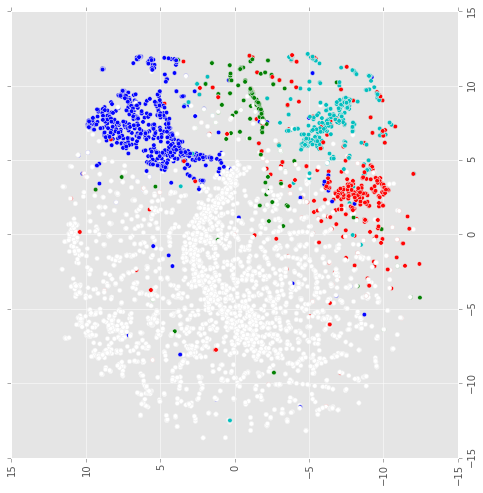
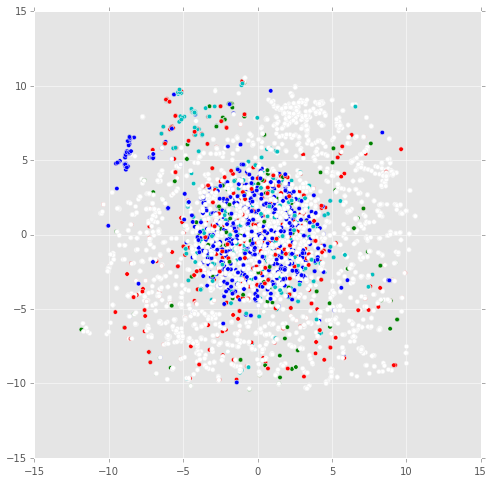


Figure : Before Training

Figure : After training

The class colors are **O, PER, MISC, ORG, LOC**

In figure 1), we have t-SNE plot of word vectors before training. The scatter plot is colored based on the gold labels in the training data. We see that there is not much separation between different classes and most of them are clustered in the center. Figure 2) is the t-SNE plot after training. Clearly, the words are separated into 5 different clusters according to their classes. We do see some of the red points (ORG) closer to green (MISC) and cyan (LOC) points, telling that there is still some ambiguity in those ORG words.

In figure 2) We looked at the words in the lower left quadrant. This region is predominantly white (O). We see a few points belonging to other classes in there and the netword was not able to learn a good representation for them. These words are as follows, color coded by their classes.

And, in, for, as, across, said, man, mother, jan, ally, sterling

The words trained from random initialization were also able to learn good representation. Here is the tsne plot for the words trained on random vector.

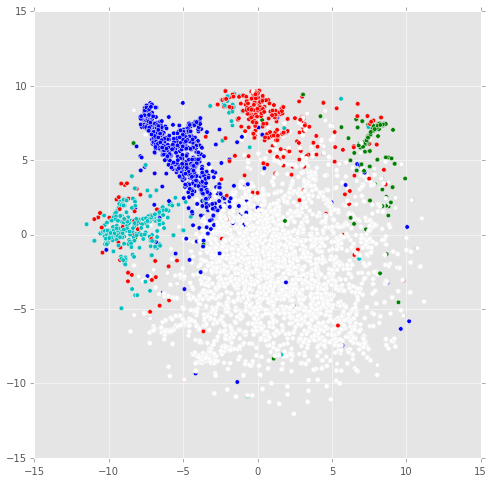


Figure : After training words with random initialization