CS 224N Project 4: Neural Networks for Named Entity Recognition

Addison Weiler (alweiler, 05716994) &
Lindsay Willmore (linzwill, 05726306)

December 7, 2014

1 Baseline System

Our baseline system uses simple string matching for NER classification. Iterating over each word in our training data, we update a counter map ($Map < String, Map < String, Integer >>$) for every instance of a word matched to a tag. The counter map is referenced in testing as follows: when a test word is found in the map, it is given the tag that corresponds to the greatest count of training instances seen. If a word was not seen in training, it is given the label 'O'. Results for the baseline are shown in Table 1. To run baseline, uncomment the corresponding lines in NER.java. This simply calls baseLineTrain() and baseLineTest() as opposed to the normal neural network equivalents.

Data Set	Accuracy	Precision	Recall	F1
Dev	0.953	0.892	0.688	0.777
Test	0.9316	0.835	0.571	0.678

Table 1: Results from baseline NER classifier.

2 Gradient with Respect to Work Vectors L

We derived the cost function J with respect to the word vectors L as the following:

$$\frac{\partial J}{\partial L} = W^T[diag(U^T(y - p_\theta(x_i))(1 - tanh^2(Wx + b_1))] \quad (1)$$

$$= W^T[diag(U^T\delta_2)(1 - tanh^2(\delta_1))] \quad (2)$$

3 System Implementation and Design

Our neural network was implemented in WindowModel.java, and was broken up into two methods—Train and Test—that each analyzed datasets created by processing files through FeatureFactory.java. When processing these files, we took into account the Window Size of the neural network specified in NER.java, and padded each sentence in the dataset with the correct amount of start and end tokens. For example, if the Window Size was 5, then two start ($< s >$) and two end ($< /s >$) tokens would be added to sentences. Aside from Window Size, the other parameters in our system can be specified in NER.java.

We used the following parameters for our baseline neural network based on suggestions from the assignment handout and Collobert et al.:

$$\text{Regularization Constant } (\lambda) = 0.0 \text{ (No regularization applied)}$$
$$\text{Learning Rate } (\alpha) = 0.001$$
$$\text{Hidden Layer Size } (H) = 100$$
$$\text{Iterations through the Dataset } (e) = 1$$
$$\text{Window Size } (W) = 3$$

Results for this baseline neural network are shown in Table 2.

Data Set	Accuracy	Precision	Recall	F1
Training	0.364	0.848	0.824	0.836
Dev	0.949	0.803	0.751	0.776
Test	0.937	0.757	0.696	0.725

Table 2: Results from the baseline neural network NER classifier, using word vector file for L initialization.

Random versus Given Initialization of L:

One variation from our basic model was to initialize the word matrix L with randomly generated values instead of PA4-provided vectors. Results from our random initialization are shown in Table 3. To recreate our results, change the corresponding lines in NER.java.

Data Set	Accuracy	Precision	Recall	F1
Training	0.963	0.857	0.808	0.832
Dev	0.950	0.928	0.725	0.773
Test	0.937	0.761	0.679	0.718

Table 3: Results from the baseline neural network NER classifier, using randomly initialized L.

A comparison of F1 scores shows that the networks initialized with the vector file did slightly better than random initialization of L. These differences were small, however, which leads us to believe that the model is robust enough to converge on values despite random initialization . However, seeding the model with "hint" values will allow it to converge to a more optimal solution. Therefore, we will use the word vector file for our baseline tests in the future.

4 Gradient Check

Our gradient checks can be implemented by setting the GRADIENT_CHECK_ON flag in WindowModel.java to True. When set to true, gradient checks will be run on the gradients: $\frac{\partial J}{\partial U}$, $\frac{\partial J}{\partial b_2}$, $\frac{\partial J}{\partial W}$, $\frac{\partial J}{\partial b_1}$, and $\frac{\partial J}{\partial L}$. The gradient checks will analyze each inputted matrix, and then output the number of "Good" and "Bad" elements: A good element is defined to be an element with a gradient-check difference of under 10^{-7}, while a bad element is defined to be an element with a gradient-check difference of over 10^{-7}. If our gradients are correct, the number of bad elements should be 0.

When the GRADIENT_CHECK_ON flag is activated and gradient checks are performed, our system passes with 0 bad elements both with and without using the regularization term. This leads us to believe that our gradients are correct for both the un-regularized and regularized cost functions with respect to each of the learned parameters. (Note: We set $\lambda = 10^{-7}$ to check our gradients with the regularized cost function).

5 Tuning Analysis

We tuned our parameters to try and achieve a higher F1 score when trained on the training set and run on the development set. See Figure 1 for results.

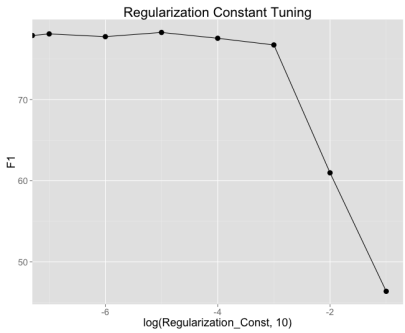
5.1 Tuning Details

Regularization Constant:

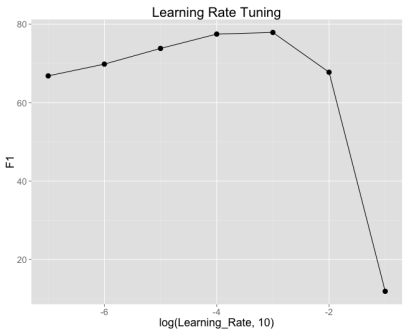
Results showed that F1 scores decreased as the regularization parameter was varied by powers of 10 from 10^{-7} to 10^{-1}. We found that some regularization helped with our F1 score in comparison to no regularization (On the test set, F1_Regularized($\lambda = 10^{-4}$) = 0.726, F1_Unregularized = 0.725), but overall, the gains were marginal at best. Based on this information, we decided to keep regularization with $\lambda = 10^{-7}$. The minimal gains that we see when using our regularized cost function show us that our model does not significantly overfit the training data, when all other parameters are held constant.

Learning Rate:

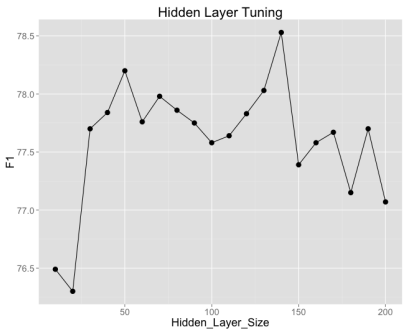
Results show that by varying learning rate from 10^{-7} to 10^{-1} by factors of 10, we were able to achieve a peak F1 score of 0.779 at a learning rate of 10^{-3}. Our data also show that as the learning rate increases, we gradually see an increase in F1 score, until it drops off at learning rates of 10^{-2} and 10^{-1}. At these higher rates,



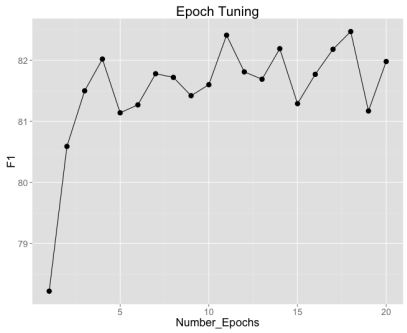
(a) Regularization Constant



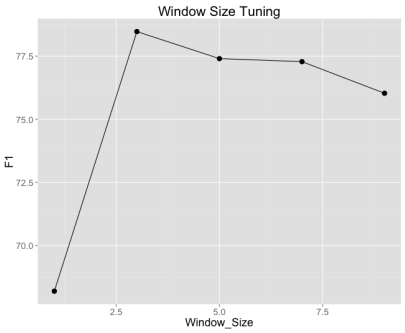
(b) Learning Rate



(c) Hidden Layer Size



(d) Number of Epochs



(e) Window Size

Figure 1: Tuning plots: F1 score versus parameter modulation.

we believe that the values of our matrices are not converging, as the learning rate is too high to allow for fine-tuning of our parameters. Based on our results, we opted to use 10^{-3} as our final learning rate for the test set data.

Number of Hidden Layer Neurons:

The accuracy by number of hidden layer nodes graph shows that small hidden layers produce much weaker performance, whereas once the size of the hidden layer exceeds 140 neurons, the gains decrease. Within the 80 to 140 neuron range, there were marginal differences seen; however, with a peak at a hidden layer size of 140, we decided to make our final model using 140 hidden layer neurons.

Number of Epochs:

By varying the number of passes over the training set, we were able to achieve a sizeable increase in F1 score, which peaked at 4 epochs (F1 = .820). However, beyond 4 epochs, we found that there were minimal variations in the F1 score, which we believed were due to the random initialization of our matrices rather than more passes helping to improve our neural network. We believe this happened because after 4 passes over the data, the neural network had learned the training set, making any more than 4 passes unnecessary. Based on these results, 4 epochs were chosen for the final test set run.

Window Size:

Results from varying Window Size show that the peak F1 score occurred at a Window Size of 3 (F1 = 0.785). At a Window Size of 5, the score was very close as well; over other tests, we have found that F1 scores for the two window sizes are comparable. However, at all other Window Sizes, we saw a substantial drop in the F1 score. We believe that this occurred for a few reasons: at a Window Size of 1, we hypothesize that there was not enough information to make an accurate update to the word vector in L, as we were only looking at one word at a time. For the Window Sizes of 7 and 9, in contrast, we believe there was too much noise resulting from the extra words being analyzed. This makes sense intuitively as well because when trying to predict a word's meaning, the meaning can be derived from the word's immediate context rather than the larger sentence as a whole. Based on these results, we chose a final Window Size of 3.

Fixed versus Learned Word Vectors:

We tested our model on fixed word vectors, which means that L was not changed during backpropagation. The results from this experiment were markedly lower (> 5% lower F1 score) than our results when learning L. For this reason, we continue to use learned word vectors in our final model.

Capitalization:

Both our baseline and first passes at the neural network NER classifiers disregarded capitalization. All words were changed to lower case before being used for either training or testing. We wanted to ask if keeping the capitalized property for words in the middle of the sentence (only reducing words at the beginning of a sentence to lower case) improved our accuracy. The results, however, were lower with this capitalization policy (again $> 5\%$ difference), so we elected not to selectively modify the capitalization of words. Our final model converts all words to lower case before training and testing.

5.2 Final Model

We used the following parameters for our final neural network model based upon the analysis done above:

$$\begin{aligned} \text{Regularization Constant } (\lambda) &= 10^{-7} \\ \text{Learning Rate } (\alpha) &= 0.001 \\ \text{Hidden Layer Size } (H) &= 140 \\ \text{Iterations through the Dataset } (e) &= 4 \\ \text{Window Size } (W) &= 3 \end{aligned}$$

Results for this final neural network are shown in Table 4.

Data Set	Accuracy	Precision	Recall	F1
Training	0.979	0.925	0.908	0.916
Dev	0.953	0.866	0.783	0.822
Test	0.935	0.802	0.726	0.762

Table 4: Results from the final neural network NER classifier with all parameters tuned.

6 Error Analysis

Our final results show that we achieved an F1 score of 0.762 on the test set, which is a large improvement over the baseline of 0.678. We saw similar improvements for both the training set and the development set as well: the training went from 0.836 to 0.916, and the development went from 0.776 to 0.822. The breakdown of the test set tags are shown in Table 5.

Data Set	Precision	Recall	F1
LOC	0.854	0.796	0.824
MISC	0.716	0.700	0.708
ORG	0.795	0.585	0.675
PER	0.801	0.811	0.806

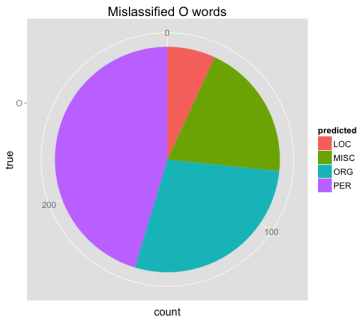
Table 5: Results from the final neural network NER classifier broken down by label (Test set).

Despite out high F1 scores, our system does have some error. Because of the super-high F1 score on the training set, we hypothesize that we might be overfitting our neural network due to the 4 iterations over the training data. However, this does not seem to have a huge effect, as our system still is able to achieve high F1 scores on both the development and test sets. Despite these improvements, some consistent failures are observed. Here are a few examples of those systematic errors:

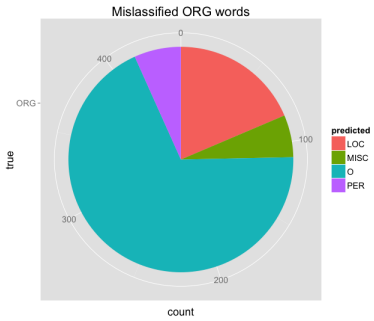
- Lots of name "misclassification": The names are correct to a human reading through them, but are not marked as PER on the gold set. This could have the effect of incorrectly lowering the F1 score from its true value.
- Low misc. precision is probably owing to the fact that misc. named entities do not follow regular patterns of grammar or preposition coincidence. In addition, the number of "MISC" entities is much lower—about 1/2 the size of any other entity—in the training set. Furthermore, we notice that many of the "MISC"-labeled words in the set are actually groups of people united by titles: "Hindu", "Swiss", "Real" (as part of Real Madrid), etc. These entities were falsely categorized as "ORG" or "O", but these errors represent those which even humans might make. A human might be inclined to call the Hindu religion an organization or "Real" without the "Madrid" an unnamed entity.

- There is a significant overlap between organization and location labelings. For example, “Milan,” “Verona,” etc. are classified as ORG rather than LOC, while “Philadelphia ” is classified as LOC rather than ORG (Philadelphia Eagles). Figures 2b and 2e support this symmetrical error pattern. In the figures, we see that after “O”, the category in which misclassified organizations fall in is “LOC” and the category in which misclassified locations fall in is “ORG”. This is most likely due to the semantic similarity between how location and organization words are used. One is “inside” a group (“ORG”) like one is “inside” a country (“LOC”).

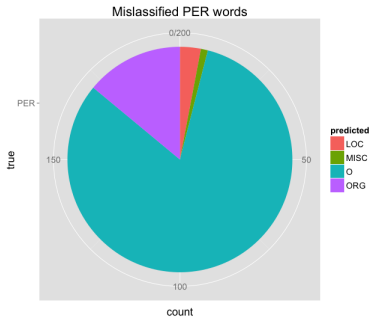
- The majority of misclassified “O” words were labeled as people. Examples of “O” words labeled as “PER” include “9.4-0-43-1”, “fourth-placed”, “attack-oriented”, and “vast”. As you can see, many of these words were numbers or contained characters such as hyphens. The model may have picked up on hyphens as a characteristic of names (hyphenated last names might be common in the training set), and this might be one example of a feature that was overlearned by our model. This could be fixed by having more hyphenated number examples of “O” words in our training set. The final word in our list of examples, “vast”, shows how some words might not even appear to fall under any named entity. If a human were to categorize a word like “vast”, which is ordinarily not a noun, that human’s first guess would probably be a person’s last name. As, such it is reasonable for our model to make the same mistake.



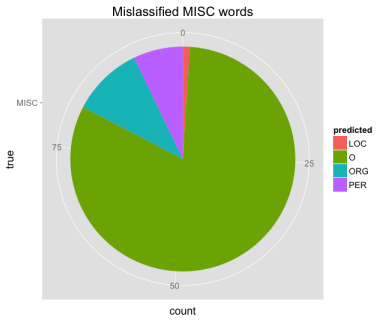
(a) Misclassified “Other” Words



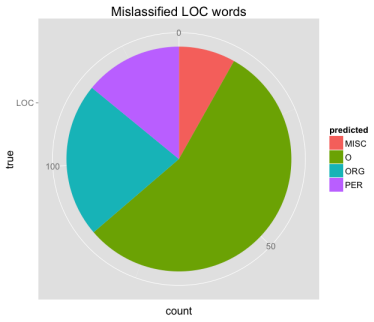
(b) Misclassified “Organization” Words



(c) Misclassified “Person” Words



(d) Misclassified “Misc.” Words



(e) Misclassified “Location” Words

Figure 2: Plot of errors by Named Entity Label.