Neural Networks for Named Entity Recognition – CS224n Final Project
By Patrick Manion and Daoying Lin

1. Introduction

For the final project, we explored using a neural network to classify which of five named entities a word represented: a location (*LOC*), an organization (*ORG*), a person (*PER*), a miscellaneous entity (*MISC*), or not any named entity (*O*).

The neural network input was a representation of all the words in a window around the target word. If we had a sentence *the boy ran* and wanted to predict *boy* with a size 3 window, the neural network would receive a vector representation of *the*, *boy*, and *ran* combined together into a single input vector. Windows that expanded beyond the start of a sentence were padded with a <s> tag, and windows beyond the end of a sentence were padded with </s> tag. Every word was represented by a 50-dimensional word vector (pre-trained in an unsupervised manner), and a special word vector UUUNKKK was included for any unknown or new words.

For training, we used a cross-entropy error function with a penalty for the size of the weight vectors. All hidden layers used a hyperbolic tangent activation function while the final output layer used a softmax activation function. The network included 5 separate output nodes, and the node with the highest output activation was used as the predicted named entity class.

2. System Design

In designing our system, we wanted to enable fast experimentation and decouple the complexity of the project as much as possible.

In order to decouple the complexity, we built numerous classes that could be separately implemented. *WindowModel* was the primary model class that wrapped up all of our tools into a train and test method. This class also prepared all of the inputs to send to the *NeuralNetwork* class for training. This allowed our neural network to only be considered with the numeric scoring and back propagation calculations and not with any domain specific information for named entity recognition.

To support these primary classes, we also created a *Document* and *DocumentSet* classes that split the large training set of *Datum* into the right granularity for our window model, which we ended up keeping at the sentence level. Then, each sentence was passed into a *WordWindow* class that handled adding the relevant start and end tags and then efficiently rolling through each window in the sentence. This class also heavily leveraged a *WordMap* class that handled the conversion between words, word ID's, and the word vectors associated with each word.

For fast experimentation, we first knew we would need a central place to set and store all settings. We created a central *Configuration* class that houses every setting that we were interested in testing from the path to the vocabulary file to the number and size of the hidden layers. This class can take a string of key-value pairs so we could directly pass command line arguments into *Configuration*.

We also recognized running tests one-by-one can be burdensome, so we created a *TestConfigReader* that allowed us to write YAML-like files that contained multiple different test configurations. We also built a *CoNLLEval* class to automatically run the CoNLL evaluation tool and output results while the model was training.

In order to leverage these classes, we also created two main methods to run the program. *Launcher* is used to run a single instance of the network and can take command line arguments to be passed to *Configuration*. We also created a *TestConfigLauncher* that can take in a test configuration file and output directory and then run every test while storing the relevant outputs.

Finally, we also implemented unit tests for the majority of our classes and methods. This allowed us to be confident that the functionality of our helper classes were all correct in isolation, which also made us confident the final network with all the components was also running correctly.

3. Gradient and Gradient Derivation

It can be shown that the expression for $\frac{\partial J(\theta)}{\partial L}$ is:

$$\frac{\partial J(\theta)}{\partial L} = W^T U^T (p_\theta - y) \odot tanh'(Wx + b^{(1)})$$

.

We also generalized the gradient expression for multiple layers of neural network, which is summarized next.

Let w_{jk}^l denote the weight for connecting the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer; b_j^l denote the bias for the j^{th} neuron in the l^{th} layer; a_j^l denote the activation of the j^{th} neuron in the l^{th} layer; z_j^l denote the weighted input to the j^{th} neuron in the l^{th} layer; $h_l(.)$ denote the activation function for the weighted input $\mathbf{z}_l$. Note that $z_j^l = \sum_i w_{ji}^l a_i^{l-1} + b_j^l$ and $a_j^l = h_l(z_j^l)$. Let's define $\delta_j^l = \frac{\partial J}{\partial z_j^l}$, the error of neuron j in layer l. Then it can be easily derived that the following four equations are true for any backpropagation system with any number of hidden layers:

$$\delta^L = \frac{\partial J}{\partial a^L} \odot h_L'(z^L) \tag{1a}$$

$$\delta^l = (W^{l+1})^T \delta^{l+1} \odot h_l'(z^l) \text{ for } l = 1, ..., L-1 \tag{1b}$$

$$\frac{\partial J}{\partial b_j^l} = \delta_j^l \text{ for } l = 1, ..., L \tag{1c}$$

$$\frac{\partial J}{\partial w_{jk}^l} = \delta_j^l (a_k^{l-1})^T \text{ for } l = 1, ..., L \tag{1d}$$

where $h'_L(z^L) = p_\theta * (1 - p_\theta)$ and $tanh'(x) = 1 - tanh^2(x)$.

For current system, we've three layers: input layer, hidden layer and output layer. The cost function is $J = -ylna^L$. Using the above general system, we can obtain the following:

$$\delta^3 = p_\theta - y \tag{2a}$$
$$\delta^2 = (W^3)^T\delta^3 \odot tanh'(z^2) = U^T(p_\theta - y) \odot tanh'(Wx + b^{(1)}) \tag{2b}$$
$$\delta^1 = (W^2)^T\delta^2 \odot I'(x) = W^T\delta^2 = W^TU^T\delta^3 \odot tanh'(Wx + b^{(1)}) \tag{2c}$$

And

$$\frac{\partial J}{\partial U} = \delta^3(a^2)^T = tanh(Wx + b^{(1)})(p_\theta - y) \tag{3a}$$
$$\frac{\partial J}{\partial W} = \delta^2(a^1)^T = LU^T(p_\theta - y) \odot tanh'(Wx + b^{(1)}) \tag{3b}$$
$$\frac{\partial J}{\partial L} = \delta^1 = W^TU^T(p_\theta - y) \odot tanh'(Wx + b^{(1)}) \tag{3c}$$
$$\frac{\partial J}{\partial b^{(2)}} = \delta^3 = p_\theta - y \tag{3d}$$
$$\frac{\partial J}{\partial b^{(1)}} = \delta^2 = U^T(p_\theta - y) \odot tanh'(Wx + b^{(1)}) \tag{3e}$$

4. Model Results

After building our model, we wanted to find a set of baseline parameters that gave solid performance as a starting point for our tests. We first tested a wide set of possible parameter values to understand the broad impacts of different parameters, and we then iteratively tried increasingly smaller permutations of the best performing networks.

During this process, we found that a flat small learning rate provided low performance and poor training speed, but a flat large learning rate learned quickly at first but then experienced large jumps in performance in later epochs. To balance this, we tested several different learning rate decay approaches until settling on dividing the learning rate by the current epoch number. The first epoch would use the provided learning rate, the second would be half the provided learning rate, the third would be one-third the provided learning rate, etc.

We finally settled on a model that gave an overall training F1 value of **91.37%** and a holdout set overall F1 of **82.73%**. This model used a window size of 5, a single hidden layer of 100 dimensions, an initial learning rate of 0.03 (that decayed across epochs), a lambda value of 0.001, including updating of the word vectors, and ran for 20 iterations.

After this, we ran a number of a number of controlled tests to understand the impact of each parameter on the network's performance:

a. Learning Rates and Iteration Count

The first test we conducted was experimenting with different learning rates and their performance across different training epochs. The results indicate that our 0.03 learning rate is near the optimal point.

Lower learning rates not only had worse early performance but also saw their performance near flat line at 20 epochs, which may be due to the learning rate now allowing the model to surpass the regularization. Larger learning rates learned quicker but eventually began jumping around in later epochs. The results do suggest that training beyond 20 epochs may provide further improvement in performance.

b. Regularization Values

Testing different regularization values showed a modest impact in performance. Larger values than our baseline 0.001 began to have large impacts on performance. Smaller values actually increased the performance, and the 0.0001 gave 84.03% overall F1 on the test set. However, these lower values also showed an increasing disconnect between the training and test scores, which indicates the model was increasingly overfitting to the training data.

c. Hidden Layer Size

We trained a series of single hidden layer neural networks with dimensions from 100 to 300 with a step size of 50. We observed that the performance is very flat for both training and test set. For single layer neural network, adding layers above 100 provides little incremental value.

d. Window Size

To study the effect of window size, we varied the value of window from 3 to 15 with step size 2. We observed there is a substantial performance improvement from 3 to 5. After that, F1 score increases as the window size increases on training. However, the test F1 score remains relatively flat, which suggests overfitting for larger window sizes. This may be because larger window sizes increase the model but also data sparsity,

e. Fixed Word Vectors vs. Updating Word Vectors

Moving from fixing the word vectors to updating the word vectors provided one of the biggest boosts of all the options. This is intuitive as the word vectors were created in an unsupervised manner based on word co-occurrence, which will place words that appear in similar contexts near each other in the space. However, this does not mean the words represent similar named entities, so the model benefits greatly from being able to adjust the space.

f. Randomly Initialized vs. Pre-Trained Word Vectors

We also compared the performance of using randomly initialized word vectors and pre-trained word vectors. Pre-training improved the performance dramatically on both training and test data. Does pre-training always help? Intuitively, if words were pertained on relevant corpus this should be the case. But what if words were pre- trained on Wall Street Journal and the problem is to do

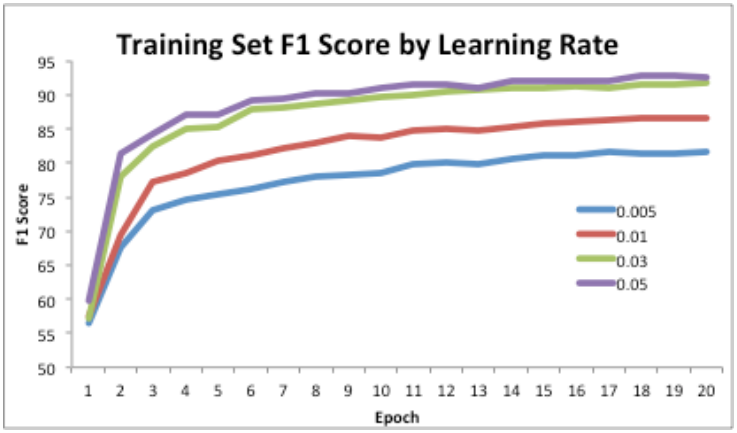
NER on corpus related to Football or some other not relevant field? Without more experiments we can't make a general conclusion.

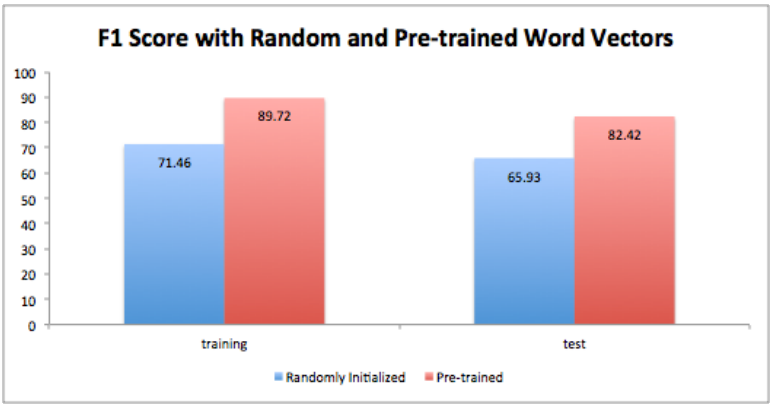
5. Error Analysis

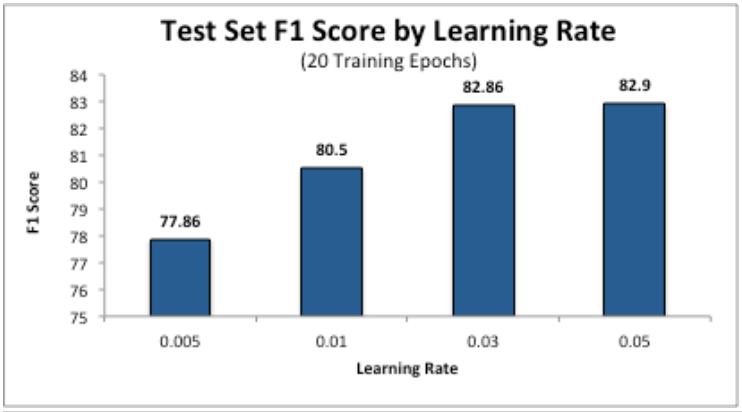
Using our benchmark model above, we also explored the reason for errors in each entity types:

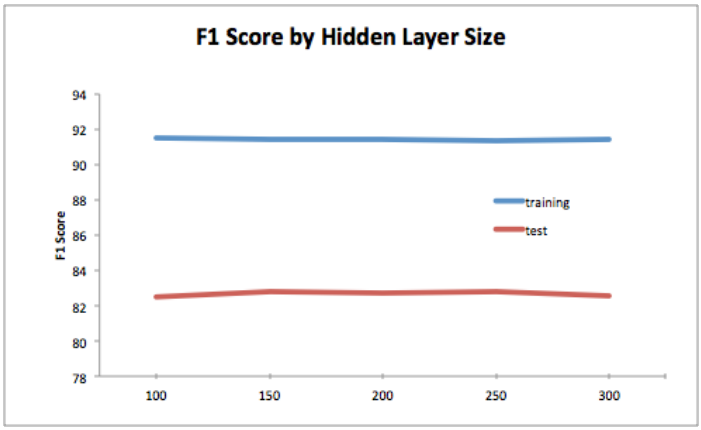
a. Location

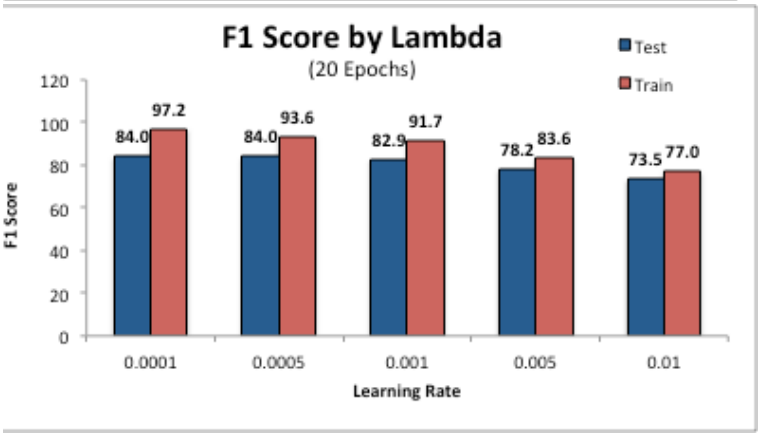
dfdfd

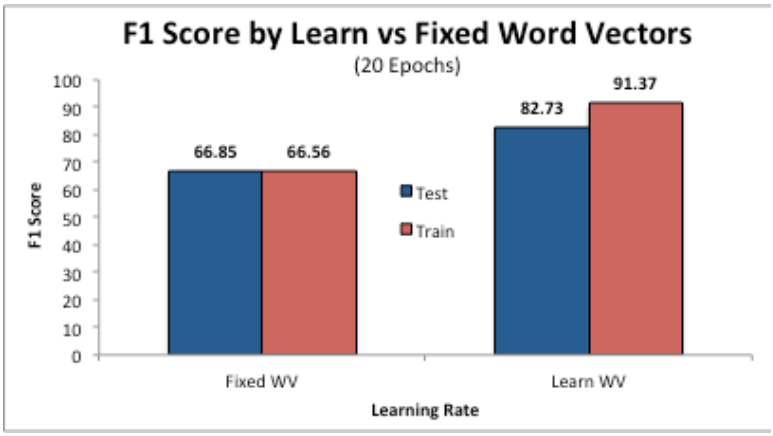












b. Organization

c. Person

• False Positive There are 129 cases where O is misclassified as PER. Among them, mostly are numbers (1, 2, 3, 53.98, 1,627, 1988, etc) and compound adjectives (ex- rebel, Lieutenant-Colonel, newly-signed, over-allotment, soft-spoken, etc.). These misclassification can be avoided if we could enforce some rules. For example, usually numbers won't be a person.
There are 69 cases where ORG is misclassified as PER. Some examples are: Fed, Duke, Ford, Johnson, Kent, Jones, Lola, Magna. A lot of them are actually orga- nization's that are named after their founders. These would be really hard for the algorithm to distinguish since those words could be either cases.
There are 38 cases where LOC is misclassified as PER. PATRICK.
There are 25 cases where MISC is misclassified as PER. One very interesting example is the words "Michael" and "Collins". These two are actually human names. The reason the algorithm misclassified them is because there is a movie named "Michael Collins". For cases like this, it's going to be really hard to make the right prediction.
• False Negative The false negative is dominated by misclassifying PER as O, which are mostly non-standard English name. For example: Hondo, Inzamam-ul-Haq, Capelli, Djorkaeff, Hun, Wang, Donghai, Xiao, Sihanouk, etc. This type of error can be re- duced by pre-training words on more general text that contains non-standard English name.

d. Miscellaneous

6. Extra Credit

We also explored several different extra credit opportunities to further test the performance of our network and better understand how the network was learning.

a. Deeper Networks

<DAOYING>

b. Visualization of Word Vector Training

<PATRICK>