Implementation

Baseline:

We implemented a simple baseline by an exact string match of unambiguous named entity in train data. This gave an F1 of 62.15

Neural Network

- 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. $X = [1\ X]^T, W = [b_1\ W], U = [b_2\ U]$
- 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.
- Training Steps: We implemented the feed forward step, cost function using the negative log likelihood and regularization cost, 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 for each epoch. We used this during training to verify that the total cost was decreasing and the training accuracy was increasing.

Gradient Check

We called gradient check with window size = 3, H = 2 and λ = 0.001. For the 200K samples in the training data, gradient check failed for about 0.1% samples. When they failed, the absolute errors were between 1E-7 and 3E-7. We also used this check in our Deep NN and Mini batch implementations. There were about 1% fails for these.

Gradients with Backpropagation

We augmented the weight matrices to include the bias terms as explained in the implementation. [2:] is indexing the vector from 2nd row to the end.
$\circ$ is element wise multiplication operator on two matrices. $\delta^{(l)}$ is the error term at layer (l). X is the input vector formed by windowed sentence, augmented by 1. For a window of size, $X = [1\ x_{-1}\ x_0\ x_1]^T$. The expression for gradient w.r.t input is

$$\frac{dJ}{dX} = W^T \delta^{(1)}[2:]$$

$$\delta^{(1)} = U^T \delta^{(2)}[2:] \circ tanh'(z)$$

$$\delta^{(2)} = p - y$$

The word vectors x_{-1}, x_0, x_1 correspond to the vectors $L_{v_1}, L_{v_2}, L_{v_3}$ in L matrix of word vectors.

Results and error analysis

Base NN performance

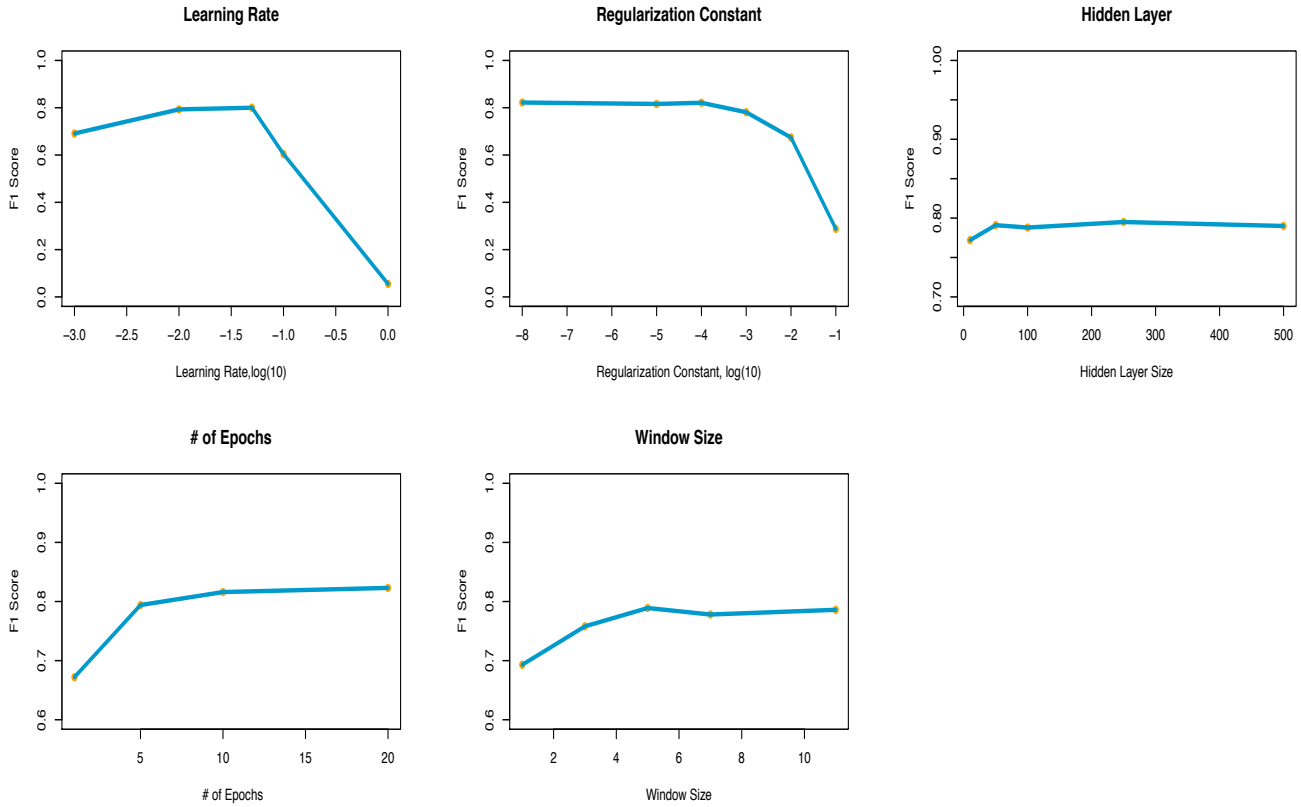
Methods	Precision	Recall	F1 Score
Overall	0.857	0.801	0.828
LOC	0.873	0.884	0.878
MISC	0.855	0.752	0.800
ORG	0.767	0.692	0.728
PER	0.904	0.837	0.869

Base NN achieved F1 score of 82.8% (84% with decay as mentioned later) with window size of 5, hidden layer size of 100, learning rate of 0.03 and regulation constant of 0.0001.
Among each NER type, PER and LOC has a high F1 scores of around 87%, followed by MISC of 80%. ORG has the lowest F1 of 73%.
In general the model performs well with PER and LOC. For example "Neil Jordan" and "Roberts" are correctly tagged as PER, and "United States" and "Algeria" are identified as LOC.
Compared to PER and LOC, ORG can be hard to differentiate from other NER labels and requires a lot of context to identify correctly. This is also shown in the word vector visualization later, where the ORG words are spread out closer to LOC and PER words. One example is "**the Dhaka Stock Exchange**", in which **Dhaka** is taken as LOC instead of ORG mistakenly. The word **Warner** in "**a statement released by Warner Bros**" is another example, which is misclassified as PER.

Similar to ORG, MISC tags can be tagged as O or other types, which may explain why it has a pretty low recall rate. For example, "Venice Film Festival" which should be tagged as MISC is identified as O for each word instead. Another example is the sentence '**He never lost more sleep over a film than over "Michael Collins"**'. **Michael Collins**, name of a movie, should be tagged as MISC, but it can be easily taken as PER unless the whole context is taken into consideration.

Hyper parameters tuning:

We tested the base model with different values of the hyperparameters to optimize F1 score on the Dev data. The impacts of the hyperparameters on the F1 score are shown in the graphs below.



Learning rate: Increasing learning rate makes it harder for the network to converge, while decreasing learning rate too much makes it hard to learn.
Regularization constant: Smaller regularization constant gives us higher F1 score. This may be due to a relative simple model structure with one layer and no severe overfitting issue. We would expect more strict regularization for deeper network.
Hidden layer size: The performance is not very sensitive to the increase of the hidden layer size. This may be due to the regularization.
Epochs: Running more iterations improves the performance, while the improvement is minimal after 10 epochs.
Window Size: Increasing the window size from 3 to 7 increased the overall F1, as shown below. This is because the model was able to use the context of the word better. Further increase in window size from 7 reduced the F1 a little. Some examples of a bigger window size leading to a better classification,
In the phrase "**US Ship in the Gulf for a total of 20**", the model with window size 3 (the, Gulf, for) said the word **Gulf** as O**.** But with window size 5, (in, the, Gulf, for, a) it had enough context to correctly get LOC.
In the phrase "**the two girls, Rachel Legeard**", window size 3 tagged **Rachel** as ORG but window size 5 got it right as PER

Model exploration

We also investigated different issues of the model, including using random/fixed word vector, and adding capitalization indicator to the model. The performance of these methods is compared to the base NN in the table below.

Methods	Accuracy	Precision	Recall	F1 Score
Base NN - SGD	0.954	0.857	0.801	0.828
Random Word Vector	0.951	0.895	0.742	0.811
Fixed Word Vector	0.904	0.725	0.522	0.607
With Capitalization Indicator	0.958	0.900	0.784	0.838
Decaying Learning rate	0.957	0.866	0.817	0.841

Random Word Vectors: The model performed equally well with the word vectors initialized with random numbers. This is because the given word vectors were trained on a corpus that is not related to NER task and is not an ideal representation for NER. In both cases, the model learnt a representation that was suited for NER.

Fixed Word Vector: The model performed poorly when the word vectors were fixed. For example, in the phrase "**as Leicestershire beat Somerset by an innings**", the base model learnt **Leicestershire** and **Somerset** as "ORG" from the training set which has this tag in a few sentences. However, with fixed words, this was not learnt and it classified them as PER. Similarly in phrase "**as Essex reached 372**", **Essex** was learnt by base model as "ORG" but fixed word model called it "O".

Capitalization: To deal with capitalization information, we added an extra row to the word vector, that indicated whether the center word in the window was capitalized or not. This helped in correctly classifying some cases that the base model did not. Examples:
The phrase "**Rochester Fire Department**", base model tagged **Fire** as "O" while the capitalized word model classified it correctly as "ORG" because of capital F.
The phrase "**monitoring Burundi closely**", base model tagged **Burundi** as "O" while the capitalized word model classified it correctly as "LOC" because of capital B.

Other things we tried --- Decaying Learning Rate
We also tried to decay the learning rate by 0.9 per epoch instead of using the constant learning rate. This way base NN was able to **achieve F1 of 84.1%.** Decaying learning rate especially in the later epochs can help reduce variance and reach stable convergence.

Extra credits

We explored two topics from the extra credits: visualization and deeper networks.

Visualization of word vector.

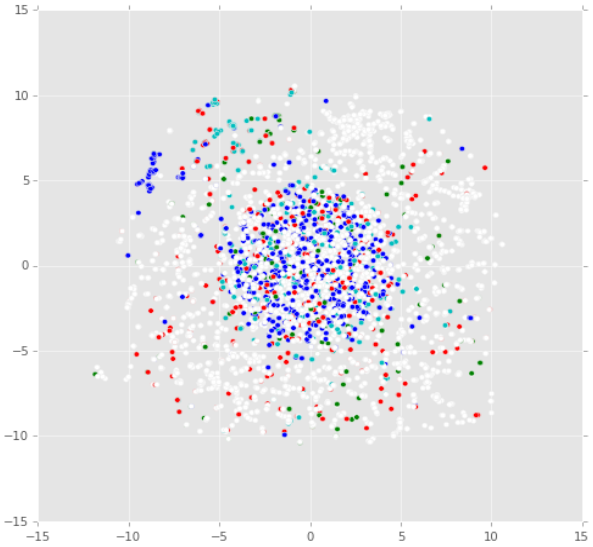


Figure 1: Before Training

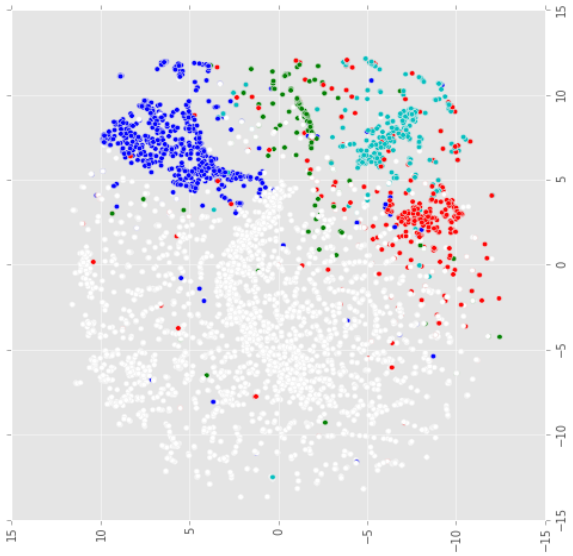


Figure 2: After training

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 and network learnt a better word vector representation for NER. 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 words belonging to other classes in this area. These words failed to be learnt properly. They are as follows, color coded by their classes, and, in, for, as, across, 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.

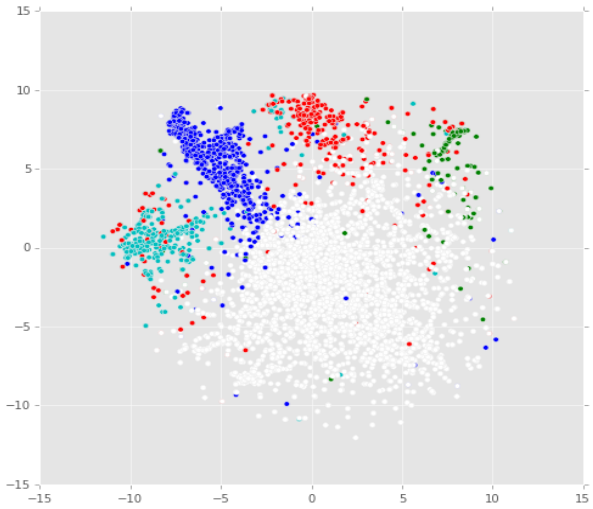


Figure 3: After training words with random initialization

Deep Neural Network and Mini Batch Implementation

Deep NN: We extended the base model to include a 2nd hidden layer. We added the new weights, and new gradient calculations for the 2nd hidden layer.
The model is implemented with Back Propagation with the gradient input similar to the base NN with just repeating $\delta^{(1)}$ step. The same logic can be generalized to K hidden layers.

$$\frac{dJ}{dX} = W^{(1)T}\delta^{(1)}[2:]$$

$$\delta^{(1)} = W^{(2)T}\delta^{(2)}[2:] \circ tanh'(z^{(1)})$$

$$\delta^{(2)} = U^{T}\delta^{(3)}[2:] \circ tanh'(z^{(2)})$$

$$\delta^{(3)} = p - y$$

Mini Batch: We also implemented a mini batch based gradient update. Here the gradients are averaged over a batch of training samples and then used to update the weights.

Methods	Accuracy	Precision	Recall	F1 Score
Base NN - SGD	0.954	0.857	0.801	0.828
Deeper NN	0.959	0.884	0.81	0.845
Base NN with MiniBatch	0.958	0.896	0.798	0.844

It appears that both deeper networks and Mini Batch improve the performance compared to base NN. For example, in "**Chittagong Cement were expected to rise**", base NN said both "**Chittagong**" and "**Cement**" as O, while deeper NN correctly tagged them as ORG. For the phrase "**Jordan said**", base NN tagged as LOC, while MiniBatch correctly tagged it as PER.
We also tuned parameters for the two methods. For Deeper NN, we used 70 for the size of the extra hidden layer and increased regularization constant from 0.0001 to 0.0003 to better take care of overfitting due to a more complicated structure. For MiniBatch, the learning rate is increased from 0.01 to 0.1 for a batch of 500 word windows.