# Experimental Analysis

## *1.Hidden Layers Results:*

Below reported are the experimental results with increasing number of hidden layers and changing different activation functions for default configuration with std-dev - 0.1 ,steps - 1001:

1 Hidden Layer: *Activation Function - Cube*

| UAS | LAS | ROOT |
|---|---|---|
| 68.9358 | 64.9674 | 54.3529 |

*Results of one hidden layer with different activation functions reported below in Section 3 while analyzing activation functions*

2 Hidden Layers: *Activation Functions : Layer 1 - Cube ,Layer 2 - Cube*

| UAS | LAS | ROOT |
|---|---|---|
| 68.8735 | 66.9615 | 59.1176 |

2 Hidden Layers: *Activation Functions : Layer 1 - Tanh ,Layer 2 - Tanh*

| UAS | LAS | ROOT |
|---|---|---|
| 56.6792 | 49.4653 | 37.9411 |

2 Hidden Layers: *Activation Functions : Layer 1 - ReLU ,Layer 2 - ReLU*

| UAS | LAS | ROOT |
|---|---|---|
| 48.5280 | 41.7807 | 18.2352 |

2 Hidden Layers: *Activation Functions : Layer 1 - Cube ,Layer 2 - ReLU*

| UAS | LAS | ROOT |
|---|---|---|
| 57.9821 | 52.2378 | 23.5882 |

3 Hidden Layers: *Activation Functions : Layer 1 - ReLU ,Layer 2 - ReLU, Layer 3 - ReLU*

| UAS | LAS | ROOT |
|---|---|---|
| 59.8123 | 52.1672 | 31.5981 |

3 Hidden Layers:*Activation Functions : Layer 1 - Tanh ,Layer 2 - Tanh ,Layer 3 - Tanh*

| UAS | LAS | ROOT |
|---|---|---|
| 67.6514 | 62.4119 | 52.4782 |

*Results of three hidden layers with cubic activation functions reported below in Section 6*

3 Hidden Layers:*Activation Functions : Layer 1 - Sigmoid ,Layer 2 - Tanh ,Layer 3 - ReLU*

| UAS | LAS | ROOT |
|---|---|---|
| 45.6988 | 36.0345 | 13.6470 |

Analysis: For two hidden layers, neural network with cubic activation function performs almost similar to that of a single hidden layer. Two hidden layers with other activation functions results in less accuracy. Further increasing hidden layers will improve accuracy for larger datasets, as more layers with less stride factor will extract more features for your input data. Consider the below explanation:

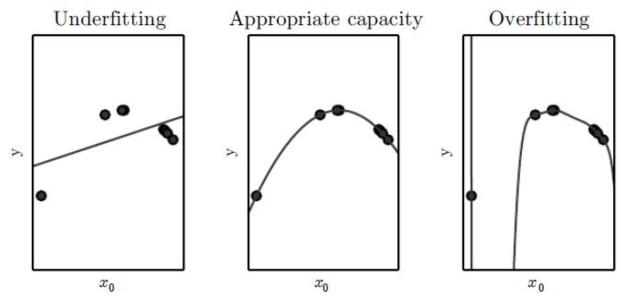


*Fig 1* *Fig 2* *Fig 3*

Fig 1 represents to fit a linear function to the data. This function is not complex enough to correctly represent the data, and it suffers from a bias(underfitting) problem.Fig 2 model has the appropriate complexity to accurately represent the data and to generalize, since it has learned the trend that this data follows (Assuming the data has an inverted parabola shape).

Fig 3 model fits to the data, but it overfits to it, it hasn't learnt the trend and thus it is not able to generalize to new data.  As we are not increasing training data and without a large training set, adding second and third hidden layer is ineffective as an increasingly large network is likely to overfit, and in turn reduces accuracy on the training data. Thus, as we increase the hidden layers, backpropagation algorithm becomes less effective and accuracy decreases as we are not increasing the training data. Since we are providing one more layer of random values we should provide enough training data to make those random values to detect some pattern in the data.

## *2.Parallel Hidden Layers:*

| UAS | LAS | ROOT |
|---|---|---|
| 65.7631 | 62.1427 | 57.6723 |

Making the hidden layers parallel, gives quite a good accuracy. This is explained from the fact mentioned in the research paper that little gain is obtained from label embeddings when the POS embeddings are present as because the POS tags of two tokens already capture most of the label information between them. Hence, avoiding connections that go across words embeddings, pos and labels and having three separate parallel hidden layers for each of them results in decent accuracy.

## *3.Exploring Activation Functions:*

I have explored the following activation functions on one hidden layer with default configuration to count for interactions between different combinations of token features.

## Sigmoid:

| UAS | LAS | ROOT |
|---|---|---|
| 41.9497 | 28.5664 | 6.0 |

*Fig1: Results*

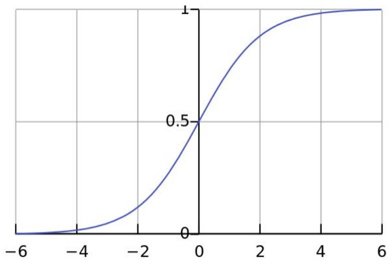


*Fig 2: Sigmoid Function*

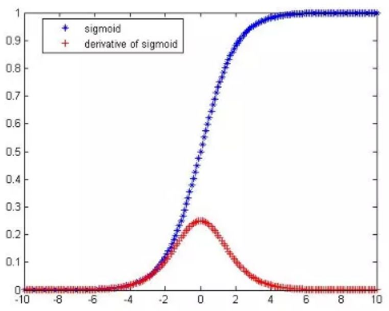


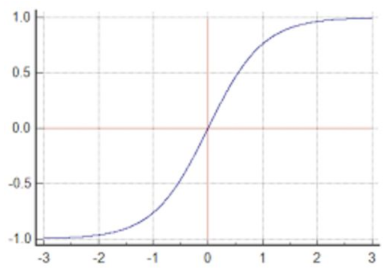
*Fig 3: Sigmoid and its Derivative*

Analysis: Sigmoid is a nonlinear function and brings non-linearity to the Neural Network. Sigmoid function has properties like nonlinearity, differentiability and the (0,1) range gives us a probability of return values.
In BackPropagation, the derivative from our output is propagated back to our first weights. For this, we should be able to derive our layers and update the weights. In case of Sigmoid, the max value of derivative is 0.25 which means a small fraction of error will be passed to previous layers thus causing neural network to learn slow. Thus, sigmoid function has this drawback.

## Tanh:

| UAS | LAS | ROOT |
|---|---|---|
| 55.3730 | 48.8246 | 33.2352 |

*Results*



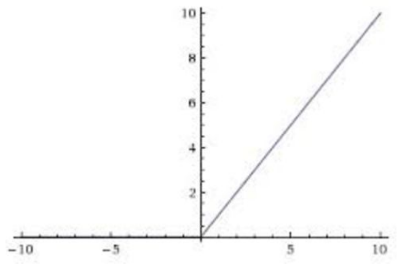
*Tanh function*

Analysis: Tanh is a nonlinear and differentiable function with its output for the hidden layers being in the range of (-1,1). It performs better than sigmoid as it has a maximum derivative of 1. Hence the error is propagated better through the layers.

## ReLU:

| UAS | LAS | ROOT |
|---|---|---|
| 58.0078 | 51.0606 | 39.1176 |

*Results*



Analysis:
ReLU is a nonlinear function whose range is between 0 - Infinity. We can pass the maximum amount of error through the network during backpropagation as the gradient is always equal to 1. One activation function may be better than other in many cases, but will be worse in some other cases. Also, I have observed that using different activations function only affects fast our network can learn rather than what it learns. With RELU(z) vanishing gradients are generally not a problem as the gradient is 0 for negative (and zero) inputs and 1 for positive inputs.

Cubic:

| UAS | LAS | ROOT |
|---|---|---|
| 68.9358 | 64.9674 | 54.3529 |

Thus, out of the above three activation functions Cubic performs the best for this neural network.

## *4.Effect of fixed Word Embeddings:*

Below are the results which were observed on fixing word embeddings and thus unmodifiable via backpropagation:

| UAS | LAS | ROOT |
|---|---|---|
| 22.6537 | 12.2217 | 5.5294 |

Since word embeddings can be pre-trained in an unsupervised way on large amounts of text, they are important in a Neural Network. As a result, neural network learns continuous representation of words in a space where words with similar meanings are close to each other. These continuous word representations are very easy for the Neural Network to work with since activation function is also continuous (sigmoid or cubic). These representations also improve generalization to unknown words. If we do not allow the word representations to learn, the LAS(labeled attachment score) obviously decreases as the percentage of words having correct head is very low since the embeddings are fixed and the neural network cannot learn them as per the activation function.

## *5.Best Configuration:*

max_iterations = 7001 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1 display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1, cubic activation function

| UAS | LAS | ROOT |
|---|---|---|
| 83.3910 | 80.5568 | 84.0588 |

On increasing the number of steps on single hidden layer, it is observed that the accuracy increases along with higher UAS and LAS scores.

## *6.Hyperparameter Tuning:*

Below are the results reported after changing a few hyper parameters:

max_iterations = 1001 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.05
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1

| UAS | LAS | ROOT |
|---|---|---|
| 54.1432 | 47.8214 | 21.3781 |

Decreasing learning rate for single layer decreases accuracy value along with lower UAS and LAS scores.

max_iterations = 1001 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.2
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1

| UAS | LAS | ROOT |
|---|---|---|
| 74.2467 | 70.4921 | 69.8721 |

Increasing learning rate for single layer increases accuracy value along with lower UAS and LAS scores.

max_iterations = 501 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1

| UAS | LAS | ROOT |
|---|---|---|
| 57.7892 | 51.6120 | 47.8935 |

Decreasing steps single layer decreases accuracy value along with lower UAS and LAS scores.

max_iterations = 5001 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1,cubic activation function

| UAS | LAS | ROOT |
|---|---|---|
| 82.0973 | 79.2207 | 81.5294 |

On further increasing number of steps for single layer increases accuracy value along with higher UAS and LAS scores.

max_iterations = 1001 , batch_size = 20000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1

| UAS | LAS | ROOT |
|---|---|---|
| 70.6290 | 66.7014 | 64.9871 |

Increasing batch size for single layer was almost as accurate as a batch size of 10000 along with almost similar UAS and LAS scores.

max_iterations = 1001 , batch_size = 5000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 1

| UAS | LAS | ROOT |
|---|---|---|
| 72.3901 | 68.2475 | 63.5213 |

Decreasing batch size for single layer improved the accuracy, LAS and UAS scores. Not too small batch sizes and not too large improve accuracy of a neural network.

max_iterations = 6001 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 2, cubic activation functions

| UAS | LAS | ROOT |
|---|---|---|
| 83.4858 | 80.6815 | 84.1176 |

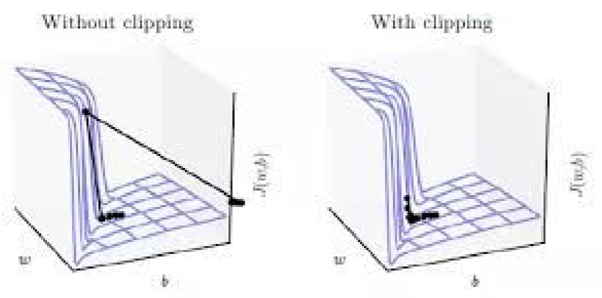
On further increasing the number of steps and number of hidden layers to 2, it is observed that the accuracy increases along with higher UAS and LAS scores.

max_iterations = 5001 , batch_size = 10000 , hidden_size = 200, embedding_size = 50, learning_rate = 0.1
display_step = 100 , validation_step = 200, n_Tokens = 48, lam = 1e-8, number of hidden layers = 3, cubic activation functions, std dev=0.14

| UAS | LAS | ROOT |
|---|---|---|
| 79.5797 | 77.9262 | 76.9411 |

For three hidden layers and with 1001 steps, the accuracy score does not improve with a constant loss value of 4.5111. On further increasing the standard deviation by 0.04 and increasing the number of steps to 5001, the gradient converges and gives the above UAS, LAS and ROOT values.

## *7.Gradient Clipping:*



When we are back propagating in Neural Networks, few gradients in the front layers can get too small due to their multiplication with small numbers.  This is called the vanishing gradient problem. Similarly, there can arise a problem of exploding gradients where gradients are getting bigger due to multiplication by number greater than one. This is called as the Gradient Explosion Problem. To solve this problem, gradient clipping helps to keep the gradient values with in the bounds. Basically, it will clip the gradients between two numbers to prevent them from getting too large. Here, we set a threshold value, and if a chosen function of a gradient is larger than this threshold, we set it to another value. When I tried to remove the gradient clipping the loss was coming out to be NaN after 3-4 iterations because gradient started exploding due of continuous multiplication of numbers greater than 1.

References:
[1]https://towardsdatascience.com/exploring-activation-functions-for-neural-networks-73498da59b02
[2]https://stats.stackexchange.com/questions/137509/do-word-embeddings-impact-neural-network-performance
[3]https://hackernoon.com/gradient-clipping-57f04f0adae
[4]https://stats.stackexchange.com/questions/338255/what-is-effect-of-increasing-number-of-hidden-layers-in-a-feed-forward-nn