# Natural Language Processing

Assignment – 3

Viraj Kamat
112818603

## Model Implementation

**The feature extraction :** For feature extraction I followed the Chen & Manning's paper, Section 3.1. A total of 48 features were extracted and this was done within the get_configuration_features method of the Token class.

In order to perform feature extraction, methods in the class Configuration was used.

The extraction happens by picking the first 3 stack and the first 3 buffer words. This was done using the *get_stack* and the *get_buffer* methods. Next the first and second leftmost and rightmost words of the first 2 words on the stack were extracted. The configuration class provides methods such as *get_left_child* and *get_right_child* for that. This was done with the help of the index returned by the get_stack method. Lastly we also extracted the leftmost-leftmost child and the rightmost-rightmost child of the first two stack words as well. This totalled upto 18 words being extracted. Then for these 18 words, we extracted their pos_tags and for the last 12 words the labels were extracted. All methods for doing so were present in the vocabulary class, the methods for performing the extraction were *get_pos_id*, *get_label_id* and *get_word_id*.

Lastly, since we captured the ids of the pos_tags, words and labels; the actual pos_tags, words and labels were captured using the configuration class' *get_label*, *get_pos* and *get_word* methods.

In all the features totalled upto 48.

**The arc standard algorithm :** For the arc standard algorithm the function apply in the class ParsingSystem was updated, this basically involved the following steps :

a. If the transition provided was ***shift***, then ***shift*** method of the configuration method was called, this moves the element from the buffer to the stack.
b. If the transition provided started with L, then the left arc was added between the first and second words on the stack, this was done with the ***add_arc*** method of the configuration class. The arc went from the top element on stack to the second element, the second element of the stack was then popped.

c. If the transition started with R, then an arc from the second element to the top element on the stack was added and the topmost element on the stack was popped. This was also done with the ***add_arc*** method of the configuration class.

**Implementing the neural model** : As per the paper we define the input layer with the 48 features per sentence set. In the init function DependencyParser class (model file) I defined the embeddings for the input layer which was essentially of the size **[vocabulary size x size of embedding dimension]**, the hidden layer weights which were of the dimension **[ size of hidden dimension x (number of tokens * embedding size)]** and the weights of the output layer which were of the size **[ number of transitions x hidden dimension ] .**

We also defined the biases that would be applied in the hidden layer which would be of the dimension **[size of hidden dimension x 1]**

While the embeddings for the inputs were defined in the range of +0.01 to -0.01 with the tensorflow random.uniform method, the weights were defined using the truncated_normal method with a standard deviation of 0.01 and a mean of 0.

In the call method of the DependencyParser class the embeddings for the sentences were looked up using those defined in the init function, the first set of weights multiplied to these sentences (tokens) and the biases then added to them. We then take either a cube, sigmoid or a tan of the output of the hidden layer and multiply it with the second pair of weights as defined in the init function.

We then compute the loss of this output, to do so we first take a softmax on the output. A custom softmax function was defined for this to ensure that when we mask out the infeasible transitions logits to zero that do not contribute to the softmax computation. Later we mask out all the feasible but incorrect transitions, compute the sum of all the transitions, take its log and compute the mean of the entire batch of sentences passed. We then compute the l2 regularization of the weights, embeddings and the bias.

The sum of loss and regularization is returned, along with logits computed.

Depending on the parameter trainable, the embeddings could be set as frozen. Also we ensure that the weights and biases are always set to trainable.

# Experiments & Analysis

| Activation function | Pretrained Embeddings | Frozen Embeddings | UAS | UASnoPunc | LAS | LASnoPunc | UEM | UEMnoPunc | ROOT |
|---|---|---|---|---|---|---|---|---|---|
| cube | Yes | No | 87.192 | 88.984 | 84.702 | 86.172 | 33.294 | 36.176 | 86.176 |
| tanh | Yes | No | 85.472 | 87.308 | 82.882 | 84.375 | 29.470 | 32.0 | 85.823 |
| sigmoid | Yes | No | 87.411 | 89.060 | 84.921 | 86.240 | 33.470 | 36.529 | 88.882 |
| cube | No | No | 84.413 | 86.138 | 81.788 | 83.179 | 27.823 | 30.0 | 84.823 |
| cube | Yes | Yes | 14.557 | 15.534 | 0.506 | 0.562 | 0.588 | 0.647 | 3.058 |

Figure 1:

**Effects of using different activation functions :**

As shown in the table above, and as described in the Chen and Mannings analysis of using different activation functions for the hidden layer, the cube activation function performs well. However, of all the 3 activation functions( cube, sigmoid, tanh) the sigmoid function performed the best with very slight improvement over the cube activation function.

The unlabelled attachment scores were used as a parameter to identify the performance of the activation functions applied.

Label attachment scores both with and without punctuation seem to perform slightly worse than unlabelled attachment scores.

**Effects of using pretrained-embeddings :**

The pretrained embeddings provide a marginal improvement when used as compared to using randomly initialized embeddings. As noticed in the table above, with the same activation function of cube, the pretrained embeddings have a UAS score of 87.192 as compared to the score of not using pretrained embeddings which was 84.413.

**Effects of freezing embeddings (non-Wtrainable) :**

When the neural network is provided with frozen embeddings, the label attachment score is bound to drop. Embeddings that are not allowed to learn, cannot take advantage of the labels provided along with the input and learn from it. On the other hand UAS and UAS without punctuation does increase and with higher epcohs is bound to learn the dependency tree better.