Assignment 3
Natural Language Processing - CSE 538
Transition Parsing with Neural Networks

Samanvitha Reddy Panyam
112025771

I. Dependency Parsing:

1. **(a) Apply function in ParsingSystem.py**
   Implemented ARC standard algorithm for generating dependency trees for the given input sentences. It defines 3 transition operators:

- Left Arc:
    - Add an arc from the word at the top of the stack to the second topmost word on the stack
    - Remove the second top most word from the stack
- Right Arc:
    - Add an arc from the second topmost word of the stack to the top word on the stack
    - Remove word at the top of the stack
- Shift: Remove word from the head of the input buffer and push it onto the stack.

1. **(b) Feature Generation:**
   We generate 48 features in total → 18 word features + 18 POS tag features + 12 label features
    - We get word and pos tag features features of top three words from the stack and the buffer
    - We then consider the left child and right child of the top two elements on the stack and their respective word, tag and label features.
    - Having done that, we also add word, tag and label features of the left child of left child from the top of the stack and of the right child of right child from the top of the stack.

II. Experiments:

1.Analysis with number of hidden layers:

a)Two hidden layers

Activation Function	No of iterations	Results

Cube	1001	Average loss at step 1000 : 0.587039647102356 UAS: 61.2932173393 UASnoPunc: 64.9691968575 LAS: 55.4602786848 LASnoPunc: 58.6672695416 UEM: 5.52941176471 UEMnoPunc: 6.23529411765 ROOT: 37.5294117647
Cube	2001	Average loss at step 2000 : 0.2729767237603664 UAS: 76.341700526 UASnoPunc: 78.9606058893 LAS: 72.8643717127 LASnoPunc: 75.1201039959 UEM: 15.7647058824 UEMnoPunc: 16.5294117647 ROOT: 68.8823529412
Cube	5001	Average loss at step 5000 : 0.1786246557533741 UAS: 82.9747987138 UASnoPunc: 84.9318939694 LAS: 80.0907346013 LASnoPunc: 81.6735432092 UEM: 24.1176470588 UEMnoPunc: 25.5294117647 ROOT: 83.5882352941

- Adding more hidden layers has increased the accuracy rate since I ran the model for 1001, 2001, 5001 iterations using the Cube activation function and observed that the accuracy increased as the iterations increased, with accuracy values of 61.29 for 1001 iterations, 76.34 for 2001 iterations and **82.97 for 5001** iterations.
- This increase may be because the model gets more no.of iterations to update the weights more accurately.

b)Three hidden layers

Activation Function	No of iterations	Results

Relu	1001	Average loss at step 1000 : 0.3387830051779747 UAS: 62.5520352968 UASnoPunc: 66.1645848641 LAS: 58.2820250767 LASnoPunc: 61.7136720737 UEM: 4.64705882353 UEMnoPunc: 4.82352941176 ROOT: 32.3529411765
Relu	2001	Average loss at step 2000 : 0.2550507877767086 UAS: 77.9494977192 UASnoPunc: 80.2605550218 LAS: 74.686541865 LASnoPunc: 76.6235234273 UEM: 17.2941176471 UEMnoPunc: 18.3529411765 ROOT: 75.2352941176
Relu	5001	Average loss at step 5000 : 0.1865692627429962 UAS: 82.8651195254 UASnoPunc: 84.7453795286 LAS: 80.1505596131 LASnoPunc: 81.6848471147 UEM: 23.8823529412 UEMnoPunc: 25.7058823529 ROOT: 82.2941176471
Relu	9001	Average loss at step 9000 : 0.15490404941141606 UAS: 85.1309918488 UASnoPunc: 86.8394280224 LAS: 82.5535309221 LASnoPunc: 83.9258463799 UEM: 29.4705882353 UEMnoPunc: 31.2352941176 ROOT: 85.7058823529

- With 3 Hidden layers, I ran all the four activation functions Cube, Relu, Tanh and Sigmoid but observed that the results were better using Relu and hence ran the model using Relu with multiple iterations.
- Similar to the above scenario, I observed that the accuracy increased for more no.of iterations with accuracy of 62.55 using 1001 iterations, 77.94 for 2001 iterations, 82.86 for 5001 iterations and **85.13 for 9001** iterations.

2.Capturing Interactions: This experiment helps understand the different types of activation functions, effect of parallel hidden layers and also the problem of gradient explosion.

a)Cube, ReLU, Sigmoid, Tanh:
The model has been run using different activation functions to observe how they capture the interactions between the words, tags and labels. The below runs have been done using single hidden layer.

Activation Function	No of iterations	Results
Cube	1001	Average loss at step 1000 : 0.38936768531799315 UAS: 68.8536032106 UASnoPunc: 71.7317583225 LAS: 64.4165814991 LASnoPunc: 66.7947775957 UEM: 9.0 UEMnoPunc: 9.47058823529 ROOT: 64.1176470588
Cube	2001	Average loss at step 2000 : 0.28041369274258615 UAS: 76.9848194032 UASnoPunc: 79.4268919912 LAS: 73.7019218785 LASnoPunc: 75.7729045385 UEM: 15.7647058824 UEMnoPunc: 16.7647058824 ROOT: 74.0
Cube	3001	Average loss at step 3000 : 0.23536388516426088 UAS: 78.7297155819 UASnoPunc: 80.9472672808 LAS: 75.6387566368 LASnoPunc: 77.4713163398 UEM: 18.1764705882 UEMnoPunc: 19.6470588235 ROOT: 77.5294117647

Relu	1001	Average loss at step 1000 : 0.708461326956749 UAS: 56.1682079916 UASnoPunc: 58.9809529192 LAS: 47.7578084104 LASnoPunc: 49.8304414175 UEM: 3.64705882353 UEMnoPunc: 3.82352941176 ROOT: 46.4705882353
Relu	2001	Average loss at step 2000 : 0.4439378589391708 UAS: 66.2786349926 UASnoPunc: 69.5274967501 LAS: 61.038961039 LASnoPunc: 63.9348895043 UEM: 7.52941176471 UEMnoPunc: 8.0 ROOT: 51.7647058824
Tanh or Hyperbolic Tangent Activation Function	1001	Average loss at step 1000 : 0.7118461692333221 UAS: 50.8462746467 UASnoPunc: 54.0439721924 LAS: 42.7798688835 LASnoPunc: 45.5632170915 UEM: 2.58823529412 UEMnoPunc: 2.70588235294 ROOT: 25.5294117647
Tanh or Hyperbolic Tangent Activation Function	2001	Average loss at step 2000 : 0.45943921893835066 UAS: 67.3305581175 UASnoPunc: 70.4487650483 LAS: 61.9812049754 LASnoPunc: 64.7176849602 UEM: 8.76470588235 UEMnoPunc: 9.41176470588 ROOT: 56.1176470588

Sigmoid	1001	Average loss at step 1000 : 1.4514522159099579 UAS: 31.5726499988 UASnoPunc: 33.8128073249 LAS: 18.56818805 LASnoPunc: 20.8189679534 UEM: 1.0 UEMnoPunc: 1.0 ROOT: 3.94117647059
Sigmoid	2001	Average loss at step 2000 : 1.008610758781433 UAS: 46.1375476731 UASnoPunc: 49.4150228904 LAS: 35.6457362216 LASnoPunc: 38.2467642571 UEM: 2.0 UEMnoPunc: 2.0 ROOT: 18.5294117647

- I ran the model for the four types of activation functions using 1001, 2001 iterations and observed that the loss and accuracy better in the case of Cube function.
- The reason for this is because the other activation functions like Sigmoid, Tanh and ReLU do not take into account the interaction between words, tags and labels.
- Obtaining the best result using Cube, I ran the model using this activation function for 1001, 2001, and 3001 iterations and observed the same pattern as described above wherein the accuracy increases with increase in the no.of iterations.

Cube Non-Linearity with three separate hidden layers:
This experiment observes the effect of using three separate hidden layers, one each for words, tags and labels after which the three layers are added together.

Activation Function	No of iterations	Results

Cube	1001	Average loss at step 1000 : 2.625013530254364 UAS: 51.2700351472 UASnoPunc: 54.6798168767 LAS: 45.2650995837 LASnoPunc: 48.7113547731 UEM: 3.23529411765 UEMnoPunc: 3.23529411765 ROOT: 19.8823529412

Effect of fixing Word, POS and Dep Embeddings

This is done by setting the parameter trainable of tf.Variable to False for embeddings.

I.e. `self.embeddings = tf.Variable(embedding_array, dtype=tf.float32),`

This means the embeddings will not get trained. The values of embeddings will remain constant throughout the training process. This results in low accuracy.

Activation Function	No of iterations	Results
Cube	1001	Average loss at step 1000 : 1.5763098466396332 UAS: 22.7335044993 UASnoPunc: 23.9416718476 LAS: 11.6185158412 LASnoPunc: 12.4455999548 UEM: 0.647058823529 UEMnoPunc: 0.705882352941 ROOT: 6.05882352941

Best Configuration:

Number of hidden layers: 3
Activation function : ReLU
Average loss at step 9000 : 0.15490404941141606
UAS: 85.1309918488
UASnoPunc: 86.8394280224
LAS: 82.5535309221
LASnoPunc: 83.9258463799
UEM: 29.4705882353
UEMnoPunc: 31.2352941176
ROOT: 85.7058823529

Gradient Clipping:

- While training, there can be large updates in the neural network model weights which is caused when large error gradients accumulate and hence we can say that exploding gradients are a problem.
- Due to this the weights become too large and result in NaN values.
- In the assignment after removing gradient clipping and running with single layer it resulted in NaN at 100 iterations. This is because the cube function is increasing the values to potentially large ones. There is no threshold in the form of gradient clipping.

References:

[1]. [Chen and Manning, 2014] Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 740–750. [Nivre, 2004] [2]. Nivre, J. (2004). Incrementality in deterministic dependency parsing. In Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together, pages 50–57. Association for Computational Linguistics.