CSE 519 assignment 3 - Report

Arc standard algorithm [2] implementation:

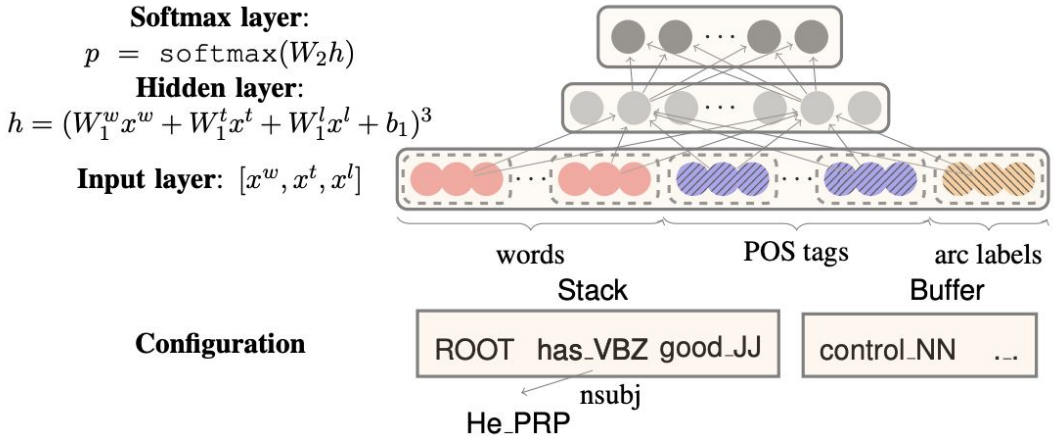
- The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.
- It is very similar to shift–reduce parsing as it is known for context-free grammar.
- A configuration for a sentence w = w1 … wn consists of three components:
    - a buffer containing words of w
    - a stack containing words of w
    - the dependency graph constructed so far
- Initial configuration:
    - All words are in the buffer.
    - The stack is empty.
    - The dependency graph is empty
- Terminal configuration:
    - The buffer is empty.
    - The stack contains a single word
- Possible transitions
    - shift (sh): push the next word in the buffer onto the stack
    - left-arc (la): add an arc from the topmost word on the stack, s1, to the second-topmost word, s2, and pop s2
    - right-arc (ra): add an arc from the second-topmost word on the stack, s2, to the topmost word, s1, and pop s1
- Configurations and transitions
    - Initial configuration: ([],[0,...,n],[])
    - Terminal configuration: ([0],[],A)
    - shift (sh): (σ,[i|β],A) => ([σ|i],β,A)
    - left-arc (la): ([σ|i|j],B,A) => ([σ|j],B,A∪{j,l,i}) only if i ≠ 0
    - right-arc (ra): ([σ|i|j],B,A) => ([σ|i],B,A∪{i,l,j})

Feature extraction implementation:

- Get top 3 stack and top 3 buffer words.
- After getting the top 3 words, extract the word ids for top 3 stack and buffer words.
- Then get word ids for 1st and 2nd left child of top 2 stack words.
- Followed by this word ids for 1st and 2nd right child of top 2 stack words.
- Then get the word ids for left of left child of top two stack words.
- Similarly get the word ids for right of right child of top two stack words.
- Then extract pos ids for top 3 stack and buffer words.
- As done above for word ids, get position ids for 1st and 2nd left child of top 2 stack words.
- Similarly get position ids for right of right child of top two stack words.
- After getting the word and position Ids, get label ids for 1st and 2nd left child of top 2 stack words.
- Similarly get label ids for 1st and 2nd right child of top 2 stack words.

- Then get label ids for left of left child of top two stack words.
- Finally get label Ids for right of right child of top two stack words.
- This makes it a total of 48 features.

The neural network architecture including activation function implementation [1]:



The neural network architecture used in the paper

- For the word embeddings, each word is represented as a d-dimensional vector.
- Then map POS tags and arc labels to a d-dimensional vector space.
- Then we choose a set of elements based on the stack / buffer positions for each type of information.
- Following this we build a standard neural network with one hidden layer, where the corresponding embeddings of the elements chosen in the above step will be added to the input layer.
- Similarly, we will add the POS tag features and arc label features x to the input layer as shown in the figure.
- To map the input layer to a hidden layer with the nodes a cube activation function is used. This is the basic mode. Experiments have been carried out with other activation functions as well like ‘tanh’ and ‘sigmoid’.
- Finally, a softmax layer is finally added on the top of the hidden layer for modeling multi-class probabilities.
- ***Cube activation function***: As stated above, a cube activation function defined as g(x) = x^3 has been used in this model. The cubic function helps model the product terms for any three different elements at the input layer directly.

Loss function implementation:

- I implemented the loss function as described in the paper [*A Fast and Accurate Dependency Parser using Neural Networks*].
- It says that the final training objective is to minimize the cross-entropy loss, plus a l2-regularization term.
- However, it had a slight variation, that is, we compute the softmax probabilities only among the feasible transitions in practice.
- In order to do so, I masked the labels tensor using tensor.nn.relu

- Following that I used '*tensor.nn.softmax'* to get the softmax probabilities for the logits tensor.
- To get the L2 regularization terms, I used the '*tf.nn.l2_loss'* function
- Finally, utilised the '*tf.keras.losses.CategoricalCrossentropy'* loss function to get the cross entropy loss.

Experiments and Analysis:

- **Activation functions**
    - *Cubic/Basic:*
      UAS: 86.45212752698357
      UASnoPunc: 88.20437461142825
      LAS: 83.82979784131416
      LASnoPunc: 85.2314474650992

      UEM: 30.529411764705884
      UEMnoPunc: 33.05882352941177
      ROOT: 87.23529411764706

    - *Tanh*:
      UAS: 86.50696712117058
      UASnoPunc: 88.32589159554627
      LAS: 84.07408330632899
      LASnoPunc: 85.57339060645452

      UEM: 31.235294117647058
      UEMnoPunc: 33.88235294117647
      ROOT: 86.47058823529412

    - *Sigmoid*:
      UAS: 85.25562729017624
      UASnoPunc: 87.20115299836094
      LAS: 82.79781638706783
      LASnoPunc: 84.417566269146

      UEM: 28.176470588235293
      UEMnoPunc: 30.41176470588235
      ROOT: 84.6470588235294

- **Without Pretrained embeddings**
      UAS: 26.323005209761448
      UASnoPunc: 29.370372463686202
      LAS: 0.019941670613455642
      LASnoPunc: 0.022607810998700052

      UEM: 0.47058823529411764
      UEMnoPunc: 0.47058823529411764

ROOT: 0.47058823529411764

- **Without tunable embeddings (by setting trainable=False in tf.Variable)**

  UAS: 83.96440411795498
  UASnoPunc: 85.76838297631832
  LAS: 81.09778896727073
  LASnoPunc: 82.52981405075454

  UEM: 26.88235294117647
  UEMnoPunc: 28.941176470588236
  ROOT: 82.94117647058823

From the above results we can get the following observations:

- Cubic activation function outperforms the other activation functions. Thus, confirming what the paper [*A Fast and Accurate Dependency Parser using Neural Networks*] claims.
- We need pre trained embeddings for our model to perform well, because from the above results we can see that without pre trained embeddings the accuracy reduces to half of what it was originally with trained embeddings
- Also, if we perform our analysis without tunable embeddings then we see a significant decline in our performance. Thus, training a parser with fixed embeddings is not a very viable solution.

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.*

2. *[Nivre, 2004] 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*.