Table 3.2. Tried hyperparameters and their values.

Hyperparameter	Values
optimizer	adadelta, adam, adagrad
dropout ratio	0.1, 0.3, 0.5, 0.7, 0.8, 0.9
# of DNN layers	1, 2, 3, 5, 7, 10
# of nrns in DNN	8, 16, 32, 64, 128, 256, 512, 1024, 2048, 3072, 4096
learning rate	1e-3, 1e-4, 1e-5
# of nrns in TBL	50, 100, 200, 300, 500, 1000, 1500
use of batchnorm	1, 0
rank of input	w, h
# of epochs	500

nrns, DNN and TBL indicate neurons, Dense Neural Network and Tree-Bidirectional-LSTM, respectively.

3.4. Attentive Recursive Tree Model

An input sentence S of N words is represented as $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$, where $\mathbf{x_i}$ is a word embedding vector in the D_x-dimensions. An *Attentive Recursive Tree* (AR-Tree) is constructed basicly as a binary tree for each phrase, with R and T standing for the root and the tree's itself, respectively. Each node $t \in T$ has two children marked by $t.left \in T$ and $t.right \in T$ (nil for missing cases) and one word denoted by $t.index$ ($t.index = i$ means the i-th word of input sentence). The $in - order$ traversal of T corresponding to S (i.e., the index of each node in the left subtree of t must be smaller than $t.index$) is ensured to preserve the crucial sequential information. The AR-Tree's most notable characteristic is that words with more task-specific information are located closer to the root.

In order to accomplish the property, a scoring function that evaluates the relative significance of words and top-down recursively selects the word with the highest score is created. A modified Tree-LSTM is used to embed the nodes bottom-up, or from leaf to root, in order to produce the sentence embedding. The downstream tasks use the resulting sentence embedding. Abstract of the model can be seen in **Figure 3.2**.

3.4.1. Top-Down AR-Tree Formation

A bidirectional LSTM is used to process the input phrase and produce a context-sensitive hidden vector for each word:

$$\overrightarrow{\mathbf{h_i}}, \overrightarrow{\mathbf{c_i}} = \overrightarrow{\mathbf{LSTM}}(\mathbf{x_i}, \overrightarrow{\mathbf{h_{i-1}}}, \overrightarrow{\mathbf{c_{i-1}}}) \tag{3.1}$$

$$\overleftarrow{\mathbf{h_i}}, \overleftarrow{\mathbf{c_i}} = \overleftarrow{\mathbf{LSTM}}(\mathbf{x_i}, \overleftarrow{\mathbf{h_{i+1}}} \overleftarrow{\mathbf{c_{i+1}}}), \tag{3.2}$$

$$\mathbf{h_i} = [\overrightarrow{\mathbf{h_i}}; \overleftarrow{\mathbf{h_i}}] \tag{3.3}$$

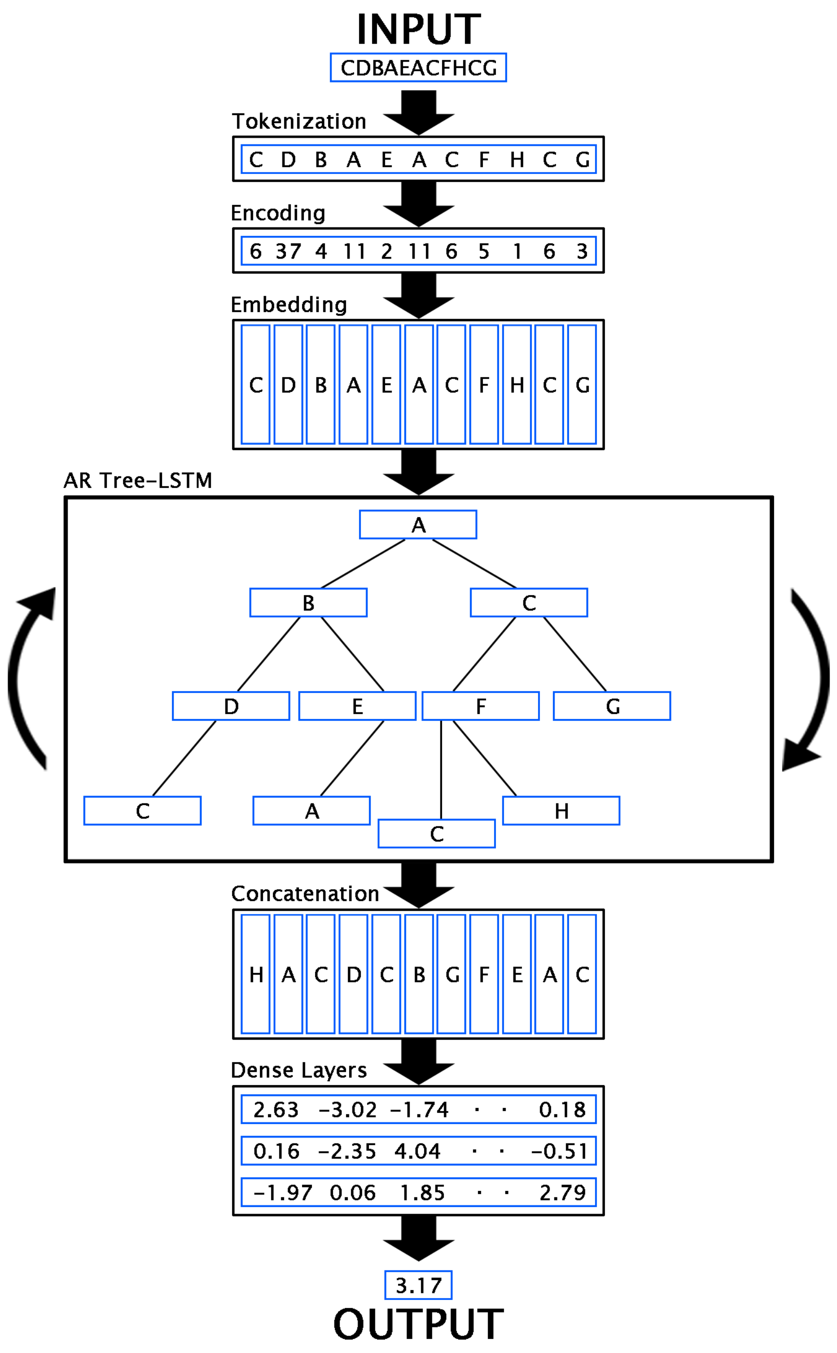
$$\mathbf{c_i} = [\overrightarrow{\mathbf{c_i}}; \overleftarrow{\mathbf{c_i}}] \tag{3.4}$$

where **h** and **c** indicate the hidden states and the cell states, respectively. **h** is used to score and leave $S = \{\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_N}\}$ alone. A trainable scoring function is created based on these context-aware word embeddings to account for the significance of each word:

$$Score(\mathbf{h_i}) = \mathbf{MLP}(\mathbf{h_i}; \theta) \tag{3.5}$$

where MLP is any multi-layer perceptron that has been parameterized by θ. A 2-layer MLP with 128 hidden units and ReLU activation are employed in particular. Traditional Term Frequency - Inverse Document Frequency (TF-IDF) is a straightforward and obvious way to express the value of words, but it is not intended for certain jobs. It will serve as the starting point.

Figure 3.2. Abstract of Attentive Recursive Tree.



AR indicates Attentive Recursive. Blue rectangulars represent layers or vectors as realistic as possible. **h** vectors of LSTM layers are concatenated in the "Concatenation" box and LSTMs are bidirectional. Output can be a classification or regression task value, here regression is prefered.

Figure 3.3. Pseudo-code of Attentive Recursive Tree Architecture.

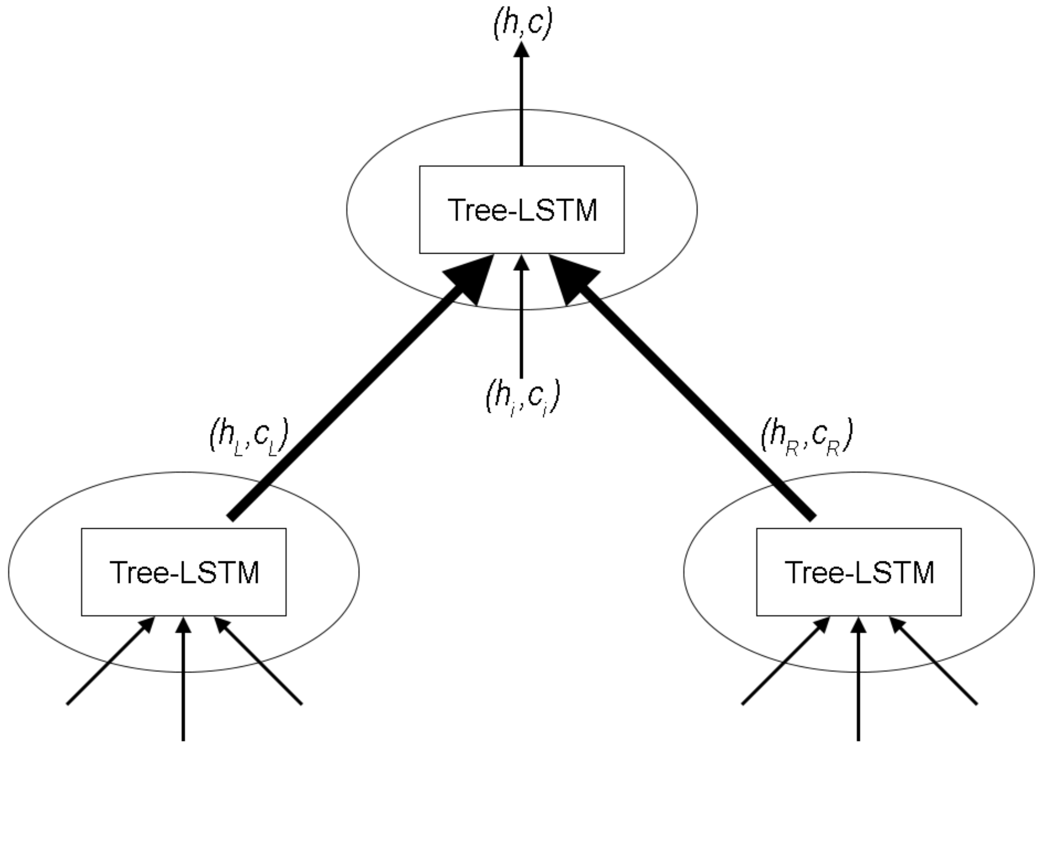
```
Input: Sentence hidden vectors $S = \{\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_N}\}$, beginning index $b$ and
ending index $e$
Output: root node $R_{S[b:e]}$ of sequence $S[b:e]$
  procedure BUILD($S, b, e$)
    $R \leftarrow nil$
    if $e = b$ then
      $R \leftarrow$ new Node
      $R.index \leftarrow b$
      $R.left, R.right \leftarrow nil, nil$
    else if $e > b$ then
      $R \leftarrow$ new Node
      $R.index \leftarrow argmax_{i=b}^{e} Score(\mathbf{h_i})$
      $R.left \leftarrow$ BUILD($S, b, R.index - 1$)
      $R.right \leftarrow$ BUILD($S, R.index + 1, e$)
    end if
    return $R$
  end procedure
```

To build the AR-Tree, a recursive top-down attention-first method is used. Given an input phrase S and the scores for each word, The word with the highest score is chosen as the root R. Then, using recursion, the two subsequences that come before and after the R is used to get the two offspring of the root. The general algorithm for creating the AR-Tree for the sequence $S[b:e] = \{\mathbf{h_b}, \mathbf{h_{b+1}}, \ldots, \mathbf{h_e}\}$ is provided in **Figure 3.3**. By invoking $R = \text{BUILD}(S, 1, N)$, the whole sentence's AR-Tree and T can be gotten by traversing all of the nodes. Each node in the parsed AR-Tree is the most insightful among its rooted subtree. Because of the fact that any additional data is not utilized in the creation, AR-Tree is applicable to any tasks requiring sentence embedding.

3.4.2. Bottom-Up Tree-LSTM Embedding

After building the AR-Tree, Tree-LSTM [50, 51] is utilized as the composition function to calculate the parent representation from its children and corresponding word in a bottom-up fashion in **Figure 3.4**. Tree-LSTM inserts cell state into Tree-RNNs to promote improved information flow. Tree-LSTM units may use both the sequential and the structural information to compose semantics since the original word sequence is maintained throughout the in-order traversal of the AR-Tree.

Figure 3.4. Bottom-Up Embedding.



The following describes the whole Tree-LSTM composition function in the model:

$$\begin{bmatrix} \mathbf{ig} \\ \mathbf{f_L} \\ \mathbf{f_R} \\ \mathbf{f_i} \\ \mathbf{o} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} \left(\mathbf{W_c} \begin{bmatrix} \mathbf{h_L} \\ \mathbf{h_R} \\ \mathbf{h_i} \end{bmatrix} + \mathbf{b_c} \right) \tag{3.6}$$

$$\mathbf{c} = (\mathbf{f_L} \odot \mathbf{c_L}) + (\mathbf{f_R} \odot \mathbf{c_R}) + (\mathbf{f_i} \odot \mathbf{c_i}) + (\mathbf{i} \odot \mathbf{g}) \tag{3.7}$$

$$\mathbf{h} = \mathbf{o} \odot tanh(\mathbf{c}) \tag{3.8}$$

$\mathbf{ig}$, $\mathbf{f_L}$, $\mathbf{f_R}$, $\mathbf{f_i}$, $\mathbf{o}$, $\mathbf{g}$, σ and $tanh$ indicate the input gate, the gate of the left child node, the gate of the right child node, the gate of the parent node, the output gate, the candidate vector, the sigmoid function and the hyperbolic tangent function, respectively.

While $\mathbf{f_L}$ and $\mathbf{f_R}$ gates control the cell state of the left and right child nodes, $\mathbf{f_i}$ is responsible for adding new information to the current node's cell state. $\mathbf{ig}$ determines the extent to which the cell state will be updated when adding new information. $\mathbf{o}$ converts the embedded representation obtained from the cell state into an output representation. $\mathbf{g}$ is used to update the cell state. σ squeezes values between 0 and 1 which is an activation function commonly used in neural networks. And $tanh$ squeezes values between -1 and 1 which is also another activation function commonly used in neural networks.

To create the node embedding $(\mathbf{h}, \mathbf{c})$, the Tree-LSTM unit combines the semantics of the current word $(\mathbf{h_i}, \mathbf{c_i})$, the right child $(\mathbf{h_R}, \mathbf{c_R})$, and the left child $(\mathbf{h_L}, \mathbf{c_L})$. Zeros are substituted for the missing inputs for nodes that lack certain inputs, such as leaf nodes or nodes with just one child.

Finally, the phrase S is fed onto tasks farther down the line using the embedding $\mathbf{h}$ of the R. Because they are closer to the R and their semantics is naturally highlighted, the sentence embedding will concentrate on those informative terms.

3.5. Creating Dynamic Fragment Dictionary

Firstly, all the possible subtree formations (fragments) should be found in all molecule trees through the corresponding datasets. It is important to find relatively large-sized fragments rather than small fragments (i.e., the ones that consist only 2 or 3 atoms except hydrogen atoms). Because larger-sized fragments are more interpretable and can be more chemically meaningful.

The only possible way of this is to form the fragments from leafs to root or bottom-up. Although root is more discriminative than other nodes, atoms of the subtree structures those are formed from root to leafs cannot be found side-by-side in SMILES representations of the corresponding molecules as it can be seen in **Figure 3.5**. So basicly, this structures are chemically not valid since the atoms of the chemical fragments which are presented as nodes are not connected to each other. That applies to the subtree structures those are formed between leafs and root as well. That's why, the only way to obtain chemically valid subtree structures (fragments) is to form fragments from leafs to root. The formation procedure is explained digestedly in **Figure 3.6**.

With obtained fragments, a dynamic fragment dictionary (DFD) is constituted which is specifically dependant to the its corresponding dataset. Then, some of the