Analytical Component:

Programming Component:

Neural Network Dependency Parsing (100 pts):

Part 1 – Obtaining the Vocabulary (0 pts):

Part 2 – Extracting Input/Output matrices for training (35 pts):

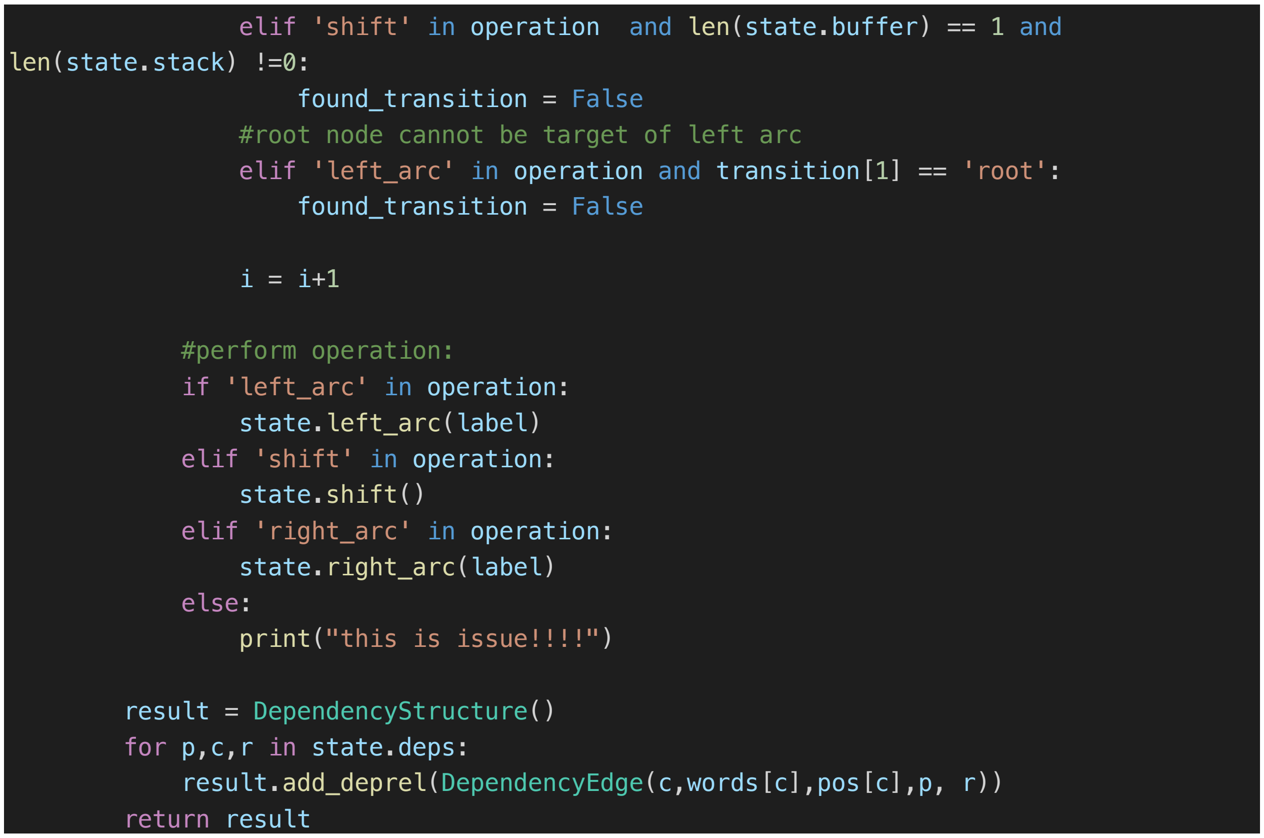
```
def get_input_representation(self, words, pos, state):
        special_ch_dict = defaultdict()
        special_ch_dict['CD'] = 0
        special_ch_dict['NNP'] = 1
        special_ch_dict['UNK'] = 2
        special_ch_dict[None] = 3
        special_ch_dict['NULL'] = 4

        list_rep_0 = state.stack[-1] if len(state.stack) > 0 else 'NULL'
        list_rep_1 = state.stack[-2] if len(state.stack) > 1 else 'NULL'
        list_rep_2 = state.stack[-3] if len(state.stack) > 2 else 'NULL'
        list_rep_3 = state.buffer[-1] if len(state.buffer) > 0 else 'NULL'
        list_rep_4 = state.buffer[-2] if len(state.buffer) > 1 else 'NULL'
        list_rep_5 = state.buffer[-3] if len(state.buffer) > 2 else 'NULL'
        list_rep = [list_rep_0, list_rep_1, list_rep_2, list_rep_3, list_rep_4,
list_rep_5]

        input_rep = np.zeros(6)

        for i in range(6):
            list_curr = list_rep[i]
            top_pos = pos[list_curr] if list_curr != 'NULL' else 'NULL'
            if top_pos in special_ch_dict:
                input_rep[i] = special_ch_dict[top_pos]
            else:
                word = words[list_curr].lower()
                if word in self.word_vocab:
                    input_rep[i] = self.word_vocab[word]
                else:
                    input_rep[i] = special_ch_dict['UNK']

        as_np_array = np.array(input_rep)
        return as_np_array

    def get_output_representation(self, output_pair):
        transition = output_pair
        index = self.output_labels[transition]
        output_matrix = np.zeros(91)
        output_matrix[index] = 1
        return output_matrix
```

Part 3 – Designing and Training the network (30 pts)

```
def build_model(word_types, pos_types, outputs):
    # TODO: Write this function for part 3
    model = tensorflow.keras.Sequential(
        [
            tensorflow.keras.layers.Embedding(input_dim=word_types, output_dim=32,
input_length=6),
            tensorflow.keras.layers.Flatten(),
            tensorflow.keras.layers.Dense(100, activation="relu", name="layer2"),
            tensorflow.keras.layers.Dense(10, activation="relu", name="layer3"),
            tensorflow.keras.layers.Dense(outputs, activation="softmax")
        ]
    )
    model.compile(tensorflow.keras.optimizers.Adam(lr=0.01),
loss="categorical_crossentropy")
    return model
```

Part 4 – Greedy Parsing Algorithm – Building and Evaluating the Parser (35pts)

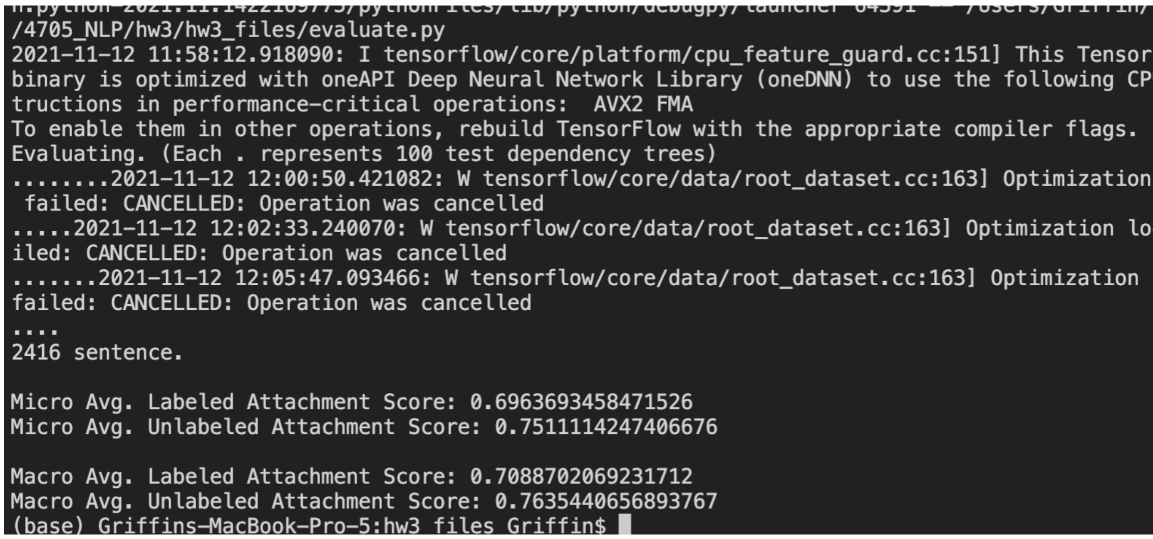
```
def parse_sentence(self, words, pos):
        state = State(range(1,len(words)))
        state.stack.append(0)

        while state.buffer:
            features = self.extractor.get_input_representation(words, pos, state)
            features = features.reshape(1,-1)
            transition_scores = self.model.predict_on_batch(features)
            found_transition = False
            i = 1
            sorted = np.argsort(transition_scores)
            while not found_transition:
                found_transition = True
                index_highest = sorted[-1][-i]
                transition = self.output_labels[index_highest]
                operation = transition[0]
                label = transition[1]

                #arc-left/arc-right not allow if stack is empty
                if 'arc' in operation and len(state.stack) == 0:
                    found_transition = False
                #shifting the only word out of buffer is also illegal, unless the
stack is empty
```



```
                elif 'shift' in operation  and len(state.buffer) == 1 and
len(state.stack) !=0:
                    found_transition = False
                #root node cannot be target of left arc
                elif 'left_arc' in operation and transition[1] == 'root':
                    found_transition = False

                i = i+1

            #perform operation:
            if 'left_arc' in operation:
                state.left_arc(label)
            elif 'shift' in operation:
                state.shift()
            elif 'right_arc' in operation:
                state.right_arc(label)
            else:
                print("this is issue!!!!")

        result = DependencyStructure()
        for p,c,r in state.deps:
            result.add_deprel(DependencyEdge(c,words[c],pos[c],p, r))
        return result
```

Test Scores:
Micro LAS: 69.6%
Micro UAS: 75.1%

Macro LAS: 70.8%
Macro UAS: 76.3%



```
n.python-2021.11.1422169773/pythonFiles/lib/python/debugpy/launcher 64391 -- /Users/Griffin/
/4705_NLP/hw3/hw3_files/evaluate.py
2021-11-12 11:58:12.918090: I tensorflow/core/platform/cpu_feature_guard.cc:151] This Tensor
binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CP
tructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Evaluating. (Each . represents 100 test dependency trees)
........2021-11-12 12:00:50.421082: W tensorflow/core/data/root_dataset.cc:163] Optimization
 failed: CANCELLED: Operation was cancelled
.....2021-11-12 12:02:33.240070: W tensorflow/core/data/root_dataset.cc:163] Optimization lo
iled: CANCELLED: Operation was cancelled
.......2021-11-12 12:05:47.093466: W tensorflow/core/data/root_dataset.cc:163] Optimization
failed: CANCELLED: Operation was cancelled
....
2416 sentence.

Micro Avg. Labeled Attachment Score: 0.6963693458471526
Micro Avg. Unlabeled Attachment Score: 0.7511114247406676

Macro Avg. Labeled Attachment Score: 0.7088702069231712
Macro Avg. Unlabeled Attachment Score: 0.7635440656893767
(base) Griffins-MacBook-Pro-5:hw3_files Griffin$ 
```