Part 2

```
def get_input_representation(self, words, pos, state):
    # TODO: Write this method for Part 2
    # if stack and/or buffer has less than 3 elements, the default manipulation is to assign NULL to
    # to input, which is 4 in words_vocab.
    buffer = [4]*3
    stack = [4]*3
    for i in range(1, min(len(state.buffer), 3)+1):
        # CD
        if pos[state.buffer[-i]] == 'CD':
            buffer[i-1] = 0
        # NNP
        elif pos[state.buffer[-i]] == 'NNP':
            buffer[i-1] = 1
        # Root
        elif words[state.buffer[-i]] is None:
            buffer[i-1] = 3
        # Valid
        elif words[state.buffer[-i]].lower() in self.word_vocab:
            buffer[i-1] = self.word_vocab[words[state.buffer[-i]].lower()]
            # UNK
        else:
            buffer[i-1] = 2
    for i in range(1, min(len(state.stack), 3)+1):
        # CD
        if pos[state.stack[-i]] == 'CD':
            stack[i-1] = 0
        # NNP
        elif pos[state.stack[-i]] == 'NNP':
            stack[i-1] = 1
        # Root
        elif words[state.stack[-i]] is None:
            stack[i-1] = 3
        # Valid
        elif words[state.stack[-i]].lower() in self.word_vocab:
            stack[i-1] = self.word_vocab[words[state.stack[-i]].lower()]
        # UNK
```

```
        else:
            stack[i-1] = 2
    # print(np.array(stack+buffer))
    return np.array(stack+buffer)
```

```
def get_output_representation(self, output_pair):
    # TODO: Write this method for Part 2
    return keras.utils.to_categorical(self.output_labels[output_pair], num_classes=91)
```

Part 3

```
def build_model(word_types, pos_types, outputs):
    # TODO: Write this function for part 3
    model = Sequential()
    model.add(Embedding(word_types, 32, input_length=6))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(10, activation='relu'))

    model.add(Dense(outputs, activation='softmax'))

    model.compile(keras.optimizers.Adam(lr=0.01), loss="categorical_crossentropy")
    return model
```

Part 4

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

    while state.buffer:
```

```
    # TODO: Write the body of this loop for part 4
    input_vec = self.extractor.get_input_representation(words, pos, state)
    output_vec = self.model.predict(input_vec.reshape((1, 6)))[0]

    sortedIdx_by_possibility = np.argsort(output_vec)[::-1]
    permitted_idx = 0
    permitted_action, rel = self.output_labels[sortedIdx_by_possibility[permitted_idx]]
    while (len(state.stack) == 0 and permitted_action in {'left_arc','right_arc'}) \
            or (len(state.buffer) == 1 and permitted_action=='shift' and len(state.stack) > 0) \
            or (len(state.stack) > 0 and state.stack[-1] == 0 and permitted_action == 'left_arc'):
        permitted_idx += 1
        permitted_action, rel = self.output_labels[sortedIdx_by_possibility[permitted_idx]]

    if permitted_action == 'shift':
        state.shift()
    elif permitted_action == 'left_arc':
        state.left_arc(rel)
    elif permitted_action == 'right_arc':
        state.right_arc(rel)

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