**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