Hongxi (Jeremy) Huang

CSE 584 Homework #2

Code reference:

https://github.com/dmksjfl/Job_Shop_Scheduling_Problem_with_Reinforcement_Learning

1. The project is trying to solve the Job Shop Scheduling Problem (JSSP) using reinforcement learning. It focuses on allocating jobs across multiple machines as efficiently as possible. By implementing an actor-critic framework, the system learns to make better scheduling decisions with each iteration, reducing the overall time it takes to complete jobs. It starts with setting up a JSSP environment where the reinforcement learning agent is trained on the dataset. As it goes through different states, the agent updated its scheduling strategy and becoming more efficient.

2. The choose_action function below is part of the ActorCritic class located in RL_brain.py. I think it is one of the core section of the reinforcement learning implementation because it helps the agent decide actions based on the current state. It calculates action probabilities, picks one, and provides information such as log probabilities, which are used to update the model. This process allows the agent to explore different strategies while reinforcing those that work well in order to improve its scheduling efficiency. (The content in " " is a comment.)

```
def choose_action(self,inputs,action_dim):
    # unpack the input state s and the hidden state (hx, cx)
    s, (hx, cx) = inputs
    # get the value, the action logit, and the updated hidden state
    value, logit, (hx, cx) = self.forward((s.unsqueeze(0),(hx, cx)))
    # convert the logit to the probability
    prob = F.softmax(logit, dim=-1)
    # calculate the log probability
    log_prob = F.log_softmax(logit, dim=-1)
    # calculate entropy to encourage exploration
    entropy = -(log_prob * prob).sum(1, keepdim=True)

    #action = prob.multinomial(num_samples=action_dim).detach()
    # create an empty list to store the action
```

```
action=[]
# for each action
for i in range(action_dim):
# sample an action based on the probability distribution
    action.append(prob.multinomial(num_samples=1).detach()[0])
# convert the action to tensor format
action = torch.from_numpy(np.array(action,dtype=np.int64).reshape(1,133))
# return the action, the log probability, the entropy, and the value
return action, log_prob, entropy, value
```