Homework 5 Template

Use this template to record your answers for Homework 5. Add your answers using LaTeXand then save your document as a PDF to upload to Gradescope. You are required to use this template to submit your answers. **You should not alter this template in any way** other than to insert your solutions. You must submit all 16 pages of this template to Gradescope. Do not remove the instructions page(s). Altering this template or including your solutions outside of the provided boxes can result in your assignment being graded incorrectly. You may lose points if you do not follow these instructions.

You should also save your code as a .py or .zip file and upload it to the **separate** Gradescope coding assignment. Remember to mark all teammates on **both** assignment uploads through Gradescope.

Instructions for Specific Problem Types

On this homework, you must fill in (a) blank(s) for each problem; please make sure your final answer is fully included in the given space. **Do not change the size of the box provided.** For short answer questions you should **not** include your work in your solution. Only provide an explanation or proof if specifically asked. Otherwise, your assignment may not be graded correctly, and points may be deducted from your assignment.

Fill in the blank: What is the course number?

$\boxed{\text{10-703}}$

Problem 0: Collaborators

Enter your team's names and Andrew IDs in the boxes below. If you do not do this, you may lose points on your assignment.

Name 1:	Akshay Sharma	Andrew ID 1:	akshaysh
Name 2:	Katayoon Goshvadi	Andrew ID 2:	kgoshvad
Name 3:		Andrew ID 3:	

Problem 1: Model-based Reinforcement Learning with PETS (80 pts)

1.1 Planning with Cross Entropy Method (CEM) [20 pts]

1.1.1 CEM implementation (5 pts)

Success Rate over 50 episodes CEM: 94%

1.1.2 Plan with random actions (5 pts)

Success Rate over 50 episodes Random: 76%
In CEM, we update the mean and variance of the distribution that we sample the action sequence from through iterations by using the k fittest individuals from the sampled population according to their evaluation through computing their trajectory costs. This leads to convergence closer toward more optimum distribution. Whereas in random action sequence we are just sampling a large number of population from the same distribution with no updates and select the best evaluated one. Since the distribution that we sample from stays the same and we do not perform any improvement in the distribution we sample from as before, the success rate of CEM(0.94) would be higher compared to random(0.76) and CEM would outperform random. But since we are selecting the best sample from a large sampled population in random, it gives us a relatively high success rate in the end.

1.1.3 MPC implementation; Comparison of MPC in environment w/o noise (10 pts)

a)
Pushing2D-v1: CEM+MPC: 100%
Pushing2D-v1: Random+MPC: 90%
b)
Pushing2DNoisyControl-v1: CEM : 54%
Pushing2DNoisyControl-v1: CEM+MPC: 88%

c)
In MPC, we optimize the action sequence through the trajectory by just taking the first action and re-planning again. The replanning prevents us to accumulate errors along the trajectory and gives us the chance to correct our selected action by using the actual encountered state after applying the model. This leads to better performance when we add MPC to our sequence action selection method(CEM or Random Action) on different environments which can be observed by the Success Rate calculated as above. Additionally, as mentioned before, CEM performs better compared to random action selection which can be seen by comparing the success rates on Pushing2D-v1 as above. Adding noise to the environment(Pushing2DNoisyControl-v1) leads to increasing the stochasity of the environment. Applying CEM without MPC leads to accumulating more error which leads to less success rate of 54 compared to CEM in Pushing2D-v1(0.94) and even random in Pushing2D-v1(0.76). That is, because of noise in Pushing2DNoisyControl-v1, when we plan action sequence in one shot, and sample the trajectory, because of the high stochasity of the environment the states that we end up in and we would experience are gonna be very much different and leads to high oscillation in the computed costs through trajectories. As mentioned above, MPC would increase the performance of CEM and random in any environment but one of its disadvantages is that we need to re-plan the action sequence every time that we take a step which can be computationally expensive compared to the case that we plan the action sequence base on the given state for a horizon of T at once.

1.2 Probabilistic Ensemble and Trajectory Sampling (PETS) [60 pts]

1.2.1 Derive the Loss of a Probabilistic Model (5 pts)

Given the state transition pairs (s_t, a_t, s_{t+1}) the negative log likelihood prediction probability loss can be written as

$$L(\theta) = -\sum_{n=1}^{N} \log(f_\theta(s_{n+1}|s_n, a_n))$$

For a Gaussian distribution $\mathbf{N}(\mu_\theta(s_n, a_n), \Sigma_\theta(s_n, a_n))$ as an output we can write,

$$f_\theta(s_{n+1}|s_n, a_n) = \frac{1}{\sqrt{(2\pi)^k \det(\Sigma_\theta)}} \exp\big(-\frac{1}{2}(\mu_\theta - s_{n+1})^T \Sigma_\theta^{-1}(\mu_\theta - s_{n+1})\big)$$

$$\log(f_\theta(s_{n+1}|s_n, a_n)) = -\frac{k}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Sigma_\theta)) - \frac{1}{2}(\mu_\theta - s_{n+1})^T \Sigma_\theta^{-1}(\mu_\theta - s_{n+1})$$

So we can write the loss as,

$$L(\theta) = \sum_{n=1}^{N} -\frac{k}{2}\log(2\pi) + \frac{1}{2}\log(\det(\Sigma_\theta)) + \frac{1}{2}(\mu_\theta - s_{n+1})^T \Sigma_\theta^{-1}(\mu_\theta - s_{n+1})$$

The first term inside the summation is a constant so we can ignore that for the optimization procedure and can write the final loss as,

$$L(\theta) = \sum_{n=1}^{N} \frac{1}{2}\log(\det(\Sigma_\theta)) + \frac{1}{2}(\mu_\theta - s_{n+1})^T \Sigma_\theta^{-1}(\mu_\theta - s_{n+1})$$

1.2.2 Loss and RMSE of a single dynamic model (5 pts)

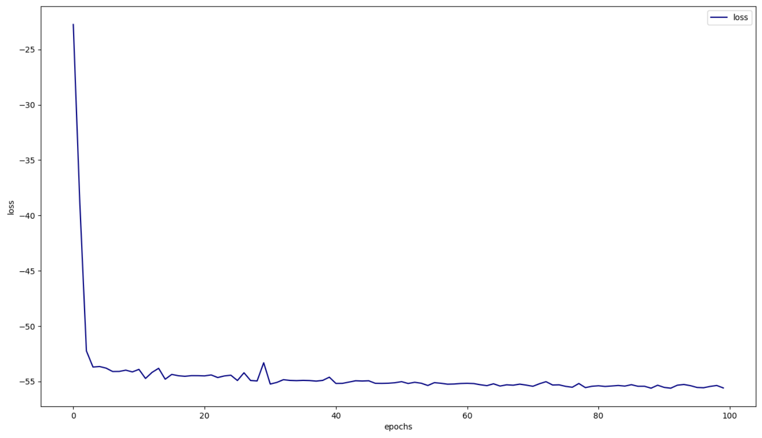


Figure 1: Gaussian Loss vs epochs

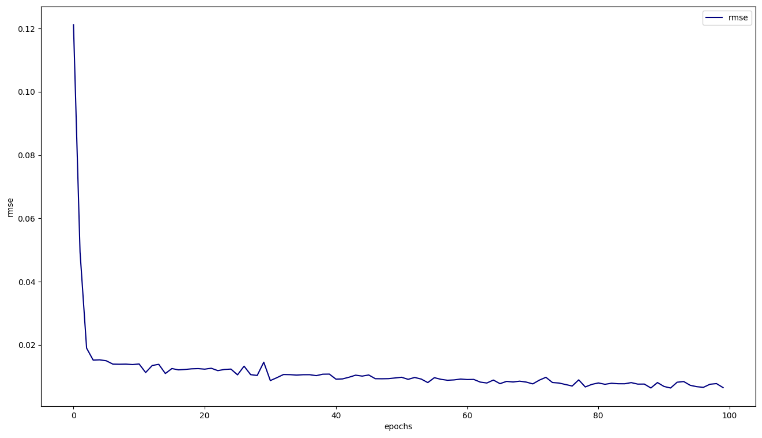


Figure 2: RMSE vs epochs

1.2.3 Planning on single model with random actions + MPC (5 pts)

Success Rate over 50 episodes random actions + MPC = 64%

1.2.4 Planning on single model with CEM+MPC (10 pts)

Success Rate over 50 episodes CEM + MPC = 76%

1.2.5 Discussion on the comparison between CEM and random actions (5 pts)

CEM(0.76) performs better than random(0.64) as mentioned before. Since we don't have an accurate model of the dynamics of the environment, the policy we learn according to this poor estimate of the dynamic of the environment would not be able to solve the environment leading to lower success rates compared to the case using ground truth. In general MPC performs better than without using MPC for both CEM and random policies. As we are using just one network to model the environment and have trained it for limited time, the model itself is not accurate. On top of that the CEM and random policies also can make wrong prediction about the future actions, which if followed will lead to error accumulation in every step. MPC makes the policies replan the action sequences at every step which helps prevent error accumulation due to inaccurate future planning.
During testing, when the agent is sampling trajectories if it ends up experiencing states similar to what the model has been trained on the agent most probably would be able to perform better and would have a higher chance of success.

1.2.6 Description of the implementation details (5 pts)

We have used the same hyper parameters as suggested in the pdf for both training and network design. We define CEM function in mpc file which is responsible for outputting a sequence of action with the size of our horizon given that particular state from a normal distribution which we update its mean and variance through time. The initial selected mean value is 0 and std is 0.5I. As we iterate through time, we sample M number of population from this distribution and generate trajectories using these action populations and the given state as input. We use the given cost function in order to evaluate different samples and select the K fittest of samples to update the mean and std of the normal distribution that we sample the actions from for next time. In CEM, for the cases that we are using probabilistic dynamics model and not the ground truth, we sample p trajectories with the similar action sequence and initial state and use the average cost for evaluation process.
For random sequence of action, we have defined a function named Random in mpc file which returns a sequence of actions with the size of horizon as output given a particular state in which we sample a population size of M*I from a normal distribution with zero mean and std of 0.5I. We evaluate all of samples using by computing the cost of states when generating trajectories with action sequence and with the given input state and select the best one.
When using MPC, we use CEM or Random to compute the sequence of actions but we only return the first action given the current state. We interact with the model to get the next state by executing the computed action and use the next state to replan. When sampling trajectories, for the first part we use the ground truth dynamics of the environment, for the second part we use a single time trained probabilistic model as estimate of the dynamics of the environment for sampling trajectories. For the third part, we use TS1 for sampling the trajectories in which an ensemble of probabilistic dynamics model includes several dynamics model defined as neural networks in model class that through at each time step we randomly select between one of the dynamics models(networks) to predict the next state given current state and action. When optimizing the action sequences using CEM or Random in PETS, we use TS1 for trajectory sampling and simultaneously update the networks and optimize the actions. During training of the Model, a buffer of (state, action, next state) is sent to the train function in mpc which calls the train function in model to train all neural networks to learn the dynamics of the environment. We use batch size of 128 when training the model and buffer size is increasing as we sample more trajectories. We use this buffer in order to learn also based on our past experience and replay them for better convergence and sample efficiency. According to size of the buffer and batch size, we compute number of batches to train on which can include repetitive data. After every 50 epochs, we evaluate our policies (CEM and Random) by computing their success rate on 20 test cased.

1.2.7 Loss and RMSE of the probabilistic ensemble model (10 pts)

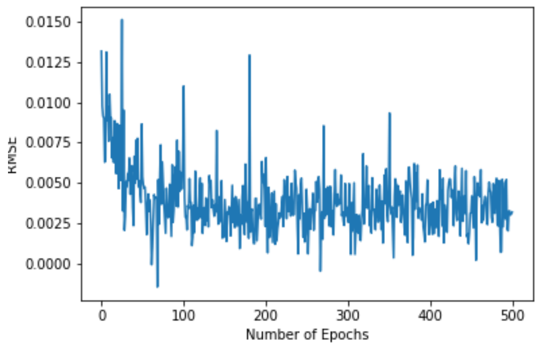


Figure 3: RMSE vs epochs

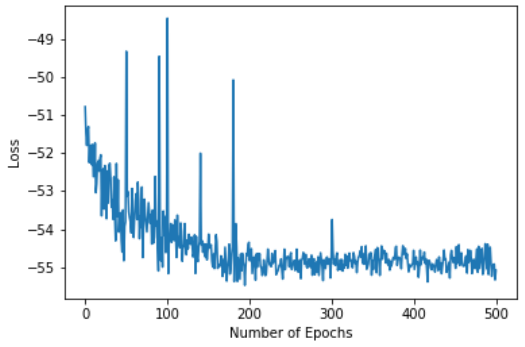


Figure 4: Gaussian Loss vs epochs

1.2.8 Success percentage of CEM+MPC and random actions + MPC (10 pts)

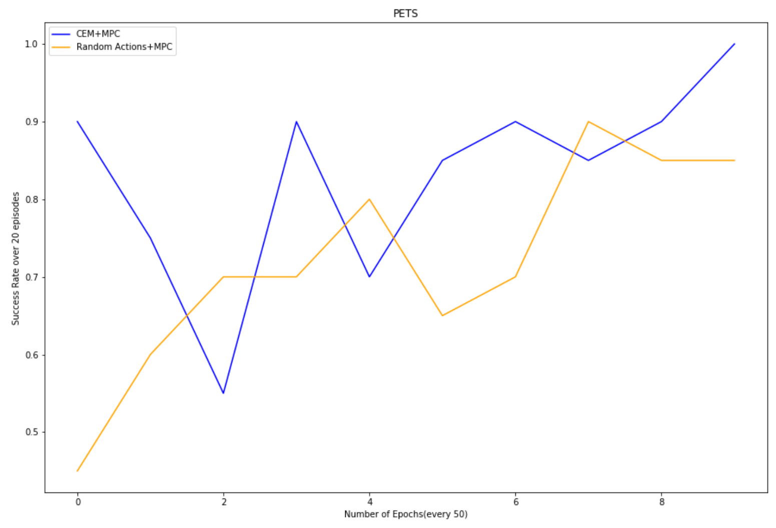


Figure 5: Success Rate(PETS) vs epochs

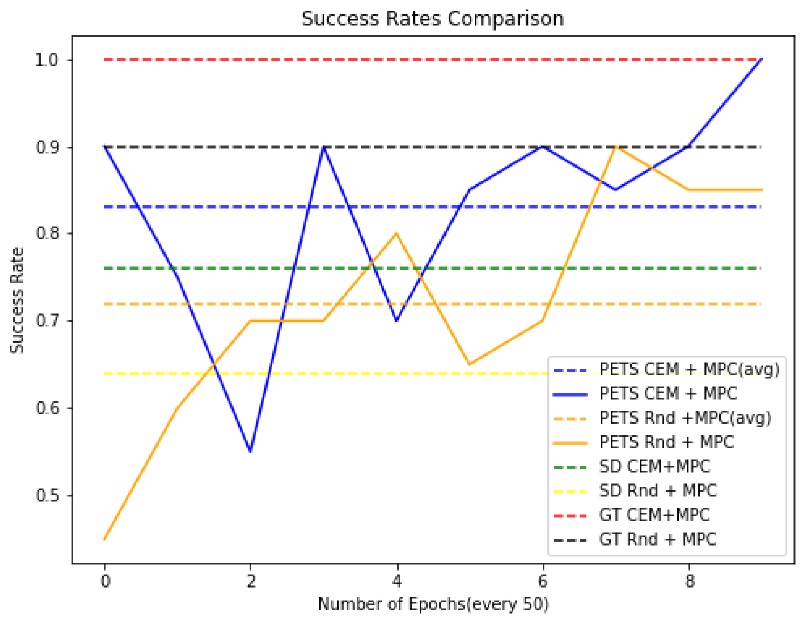


Figure 6: Success Rate Comparisions

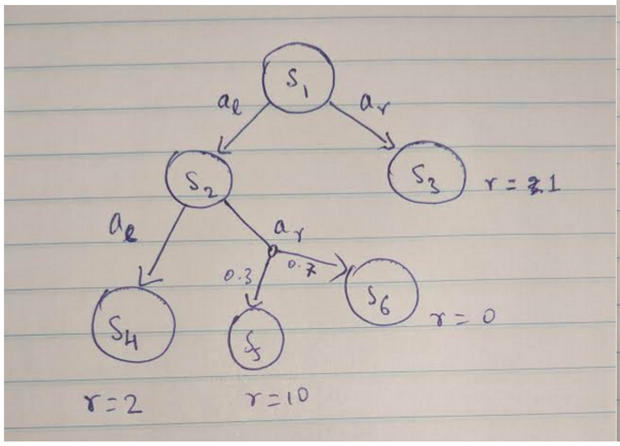
According to above graph(Fig 3), CEM+MPC performs better than Random+MPC on average as before. As the model and policy are both trained more through epochs, the success rate increases which is expected because we get a more accurate model of the dynamics of the environment and try to optimize the action sequences based on that. According to above graph(Fig 4), whatever model that we have used, CEM always outperforms Random. By comparing single dynamics model and ground truth dynamics we can see that the success rate of ground truth dynamics is higher for both CEM and random which is expected. Since in single dynamics model, the trained model of the environment is not that accurate which results in modeling the dynamics of environment poorly compared to ground truth dynamics which we have the real dynamics of the environment and we are just trying to optimize the action sequence. By comparing the ground truth dynamics and PETS, we can observe as we train model of the environment in PETS and optimize the action sequence accordingly, the average success rate becomes higher through time and success rates become relatively close to ground truth dynamics results. As for comparing PETS and Single Dynamics since we are using a more accurate model of the environment for optimizing the actions in PETS, the performance of PETS for both CEM and Random is higher.

1.2.9 Limitation of MBRL (5 pts)

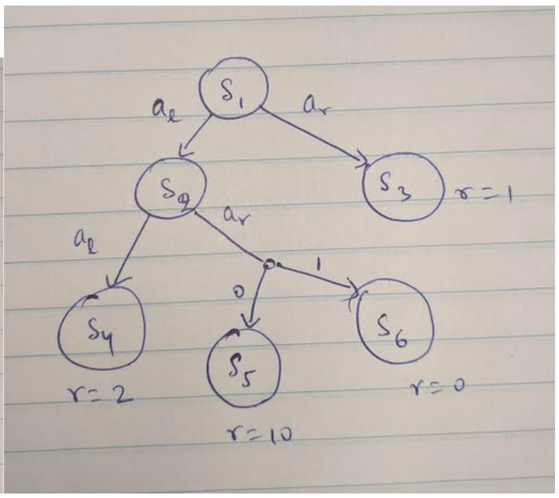
Limitations:
If the learned model is not good enough then training any policy on such a model could lead to bad results. As all learned models have some inherent assumptions about the actual dynamics of the environment, they could go wrong easily.
As can be seen in the current homework, learning the dynamics model itself is more computationally complex than policy gradients or other model free RL methods.
Scenarios where MBRL is should be preffered:
As MBRL is generally more sample efficient, it could lead to faster training of RL based agents, so it could be used as pre-training step for most model free methods.
As we are directly learning from a model of the environment, we can train policies for multiple tasks at once, so it could be used for multi-task learning.
As learned models allow us bypass interacting with the actual environment, MBRL can be really helpful in scenarios where interacting with the actual environment is expensive (though the model should be good enough to actually approximate the environment).

Problem 2: Theoretical Questions (20 pts)

2.1 Deterministic vs Stochastic Model (10 pts)



(a) Stochastic model



(b) Deterministic model

Assume an MDP with 6 possible states as shown above, with two possible start states s_1, and s_2 and four terminal states s_3, s_4, s_5, s_6. The possible actions in any state except the terminal ones are a_l, and a_r which takes them to the corresponding next states as shown in the diagram. Any agent following this MDP gets rewards only when it reaches the terminal states. The rewards are, $r(s_3) = 1$, $r(s_4) = 2$, $r(s_5) = 10$, and $r(s_6) = 0$. Also the discount factor is $\gamma = 1$. Assuming we have a stochastic model for the MDP such that the transition from state s_2 on taking action a_r gives the next state as s_5 with probability 0.3 and s_6 with probability 0.7

Now consider we fit a deterministic model to the given MDP such that for the state, action pair (s_2, a_r), we always land up in state s_6.

The optimal policy for the stochastic model is:

$$\pi_p^* = \{(a_l|s_1), (a_r, s_2)\}$$

with average return = 3

Whereas the optimal policy for the deterministic model is:

$$\pi_{\hat{p}} = \{(a_l|s_1), (a_l, s_2)\}$$

with average return = 2

where $(a_i|s_k)$ means select action a_i in state s_k, Clearly $\pi_{\hat{p}} \neq \pi_p^*$

2.2 Aleatoric vs Epistemic Uncertainty (5 pts)

Aleatoric uncertainty can be measured by taking an average of the variance output of the ensemble of networks. Each probabilistic network outputs a variance indicating how uncertain it is about the next state in the transition. So an average of the variance of the ensemble we can get an estimate of how uncertain the whole ensemble is about the next state, which represents the inherent stochasticity of the actual environment.

If all the networks in the ensemble have fitted to the seen distribution of data, they all will more or less agree about what the next state of a transition should be. In case of unseen part of the environment, or the part on which the networks did not fit properly they will have a lot of disagreement on the outputs. So we can measure epistemic uncertainty by calculating some sort of similarity metric (eg. dot product between the output vectors) between the next states predicted by each network. One measure for Epistemic Uncertainty could also be the variance of the means of the output which indicates how uncertain different networks are about the next predicted state.

2.3 Failure Modes without Considering Uncertainty (5 pts)

Failure mode for aleatoric uncertainty:
If we do not consider aleatoric uncertainty, we are basically trying to fit a deterministic model to a stochastic environment. As seen in the Q2.1 an optimal policy learned for such a model might not be optimal for the actual environment. This could easily lead to an agent following this policy to land in bad states, which the deterministic model never even considered.
Failure mode for epistemic uncertainty:
Epistemic uncertainty mainly arises from not seeing enough data during training. Not just because that data is appears with low probability but could be just because of limited data. The model will give wrong predictions for those data points during testing or training an agent. This could lead potentially bad states and could lead to bad results.

Feedback (1pts): You can help the course staff improve the course for future semesters by providing feedback. You will receive a point if you provide actionable feedback **for each of the following categories**.

What advice would you give to future students working on this homework?

As this assignment involves MBRL which is very different from the previous ones, reading the referred paper properly before starting the assignment will help a lot. Going directly for the assignment is not a good strategy.
One thing for the course staff: All the other assignments were nicely structured in terms of skeleton code, and it actually made the assignment easier to understand. But for this one the skeleton code just added to the confusion, so we would recommend to change the skeleton code for future offerings of the course.

Time Spent (1pt): How many hours did you spend working on this assignment? Your answer will not affect your grade.

Alone	15
With teammates	25
With other classmates	0
At office hours	3