Exam : Artificial Intelligence Principles and Techniques

2022 Fall

Dec.21 2022

Instruction

1. This is a **closed-book** online exam. Please do not use electronic devices other than a computer and a mobile phone during the exam. Your computer should not open applications or pages other than Tencent Meeting and PDF reader(only for reading this exam paper). Your mobile phone should only be used to keep the video for the second view. Any paper materials are not allowed either.

2. Please use your own writing papers to answer the questions and write the question number clearly. You need to scan your answers (by taking pictures or other tools) within **10 minutes** after the exam and organize them into a pdf file to upload to the web learning. **Those who submit the answers after the deadline will get 0 points.**

3. In case of midway power failure or network disconnection, or physical discomfort, report to the TAs.

4. Answer those that you believe you can solve easily first.

5. Good luck!

1 Short Questions (12 pts)

(a) (4 pts) Assume that in a single-goal graph search problem, we run *Uniform Cost Search* with action costs d_{ij} to go from node i to node j that are potentially different from the search problem's actual action costs c_{ij}. We call two search algorithms *equivalent* if and only if they expand the same nodes in the same order and return the same path. For example, the *Uniform Cost Search* algorithm with these costs $d_{ij} = 1$ is *equivalent* to Breadth-First Search.

Let $h(n)$ be the value of the heuristic function at node n (assume it is consistent). The *Uniform Cost Search* algorithm with costs $d_{ij} =?$ is *equivalent* to (i) Greedy Search (2 pts) (ii) A* Search (2 pts)? You can use expressions containing c_{ij}, $h(i)$ and $h(j)$ to indicate d_{ij} in your answer.

(b) (4 pts) *Markov blanket* of a random variable Y in a random variable set $\mathcal{S} = \{X_1, \ldots, X_n\}$ is the *minimal* subset $\mathrm{MB}(Y) \in \mathcal{S}$ conditioned on which all other variables are independent with Y. I.e., $Y \perp\!\!\!\perp \mathcal{S} \setminus \mathrm{MB}(Y) \mid \mathrm{MB}(Y)$. Consider the Bayesian network in Figure 1. For example, $\mathrm{MB}(X_2) = \{X_1, X_3\}$.

(i) (2 pts) Write down the Markov blanket of X_3.

Hint: You may first check the correctness of (ii).1 to obtain a method for computing $\mathrm{MB}(Y)$ in a Bayesian network that is faster than brute force enumerating. Actually this method is mentioned when introducing Gibbs Sampling.

(ii) (2 pts) Please check the correctness of the following statements.

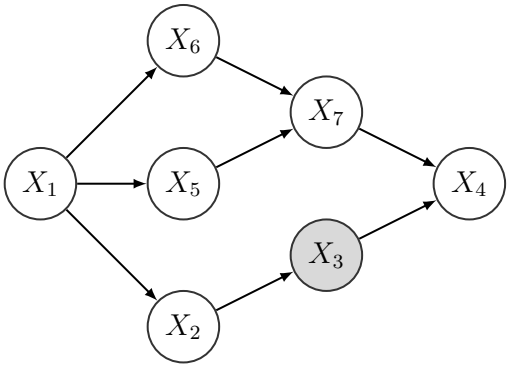

Figure 1: Bayesian Network.

1. In a Bayesian network, the Markov blanket of a node Y is the set of nodes composed of Y's parents and Y's children.
2. $\mathrm{MB}(Y)$ contains all the information one needs to infer Y, thus the variables in $\mathcal{S} \setminus \mathrm{MB}(Y)$ are redundant if all variables in $\mathrm{MB}(Y)$ are known.

(c) (4 pts) Conceptual short questions of Reinforcement Learning.

(i) (2 pts) Following a Boltzman policy $\pi(a|s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a'} \exp\left(Q(s,a')/T\right)}$, please describe how the temperature factor T affect the agent's behaviors. If we want to dynamically change T through training, should it increase or decrease over time? *Hint*: Consider how the behavior's degree of greedy (random) evolve corresponding to the change of T.

(ii) (2 pts) Recall the Cross-Entropy Method (CEM) for model-based RL (see the following implementation in Python).

```
def distribution(params: Params, t: int) -> Action: ...
def fit(elites: Iterable[Sequence[Action]]) -> Sequence[Params]: ...
def value_function(action_seq: Sequence[Action]) -> float: ...

def sample_sequence(params_seq: Sequence[Params]) ->
↪  Sequence[Action]:
    return [distribution(params) for params in params_seq]

def CEM(params_seq: Params, T: int, N: int, M: int):
    for _ in range(T):
        action_seq_samples = {sample_sequence(params_seq) for _ in
        ↪  range(N)}
        elites = heapq.nlargest(M, action_seq_samples,
        ↪  key=value_function)
        params_seq = fit(elites)
```

Suppose `T` is sufficiently large. Check the correctness of the following statements.

1. It is a closed-loop planning algorithm.
2. It is optimal in stochastic environments.

2 Parallel Search (21 pts)

In this question, let's think about what will happen if we use two or more workers on search problems. In *Parallel Search*, instead of expanding one nodes in a single time step, we can expand up to several nodes simultaneously.

(i) (3 pts) First we consider single worker on A* search problem with the search tree in the diagram in Figure 2. Each node is drawn with the state in the left and the f-value ($(f(n) = g(n) + h(n))$) in the right. G is the unique goal node. Please complete the

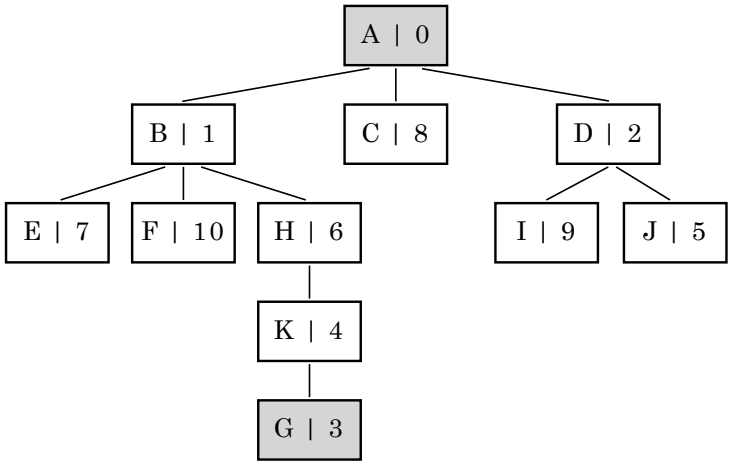

Figure 2: Search Tree.

expand-order table below.

Note: Leave the cell blank if search terminated before the step.

Step	0	1	2	3	4	5	6	7	8
Node	A	B							

(ii) (6 pts) Then we consider two and three workers on A* search problem with the search tree in the diagram in figure 2. Please complete the expand-order tables below.

Two workers:

Step	0	1	2	3	4	5	6	7	8
Node	A	B,D							

Three workers:

Step	0	1	2	3	4	5	6	7	8
Node	A	B,C,D							

(iii) (12 pts) Now we try to finish the algorithm of *Parallel A* Search*.

In A*-Parallel, we use a *master* thread which runs A*-Parallel and a set of $n > 1$ *workers*, which are separate threads that execute the function Worker-Expand which performs a

node expansion and writes results back to a shared fringe. The master thread issues non-blocking calls to Worker-Expand, which dispatches a given worker to begin expanding a particular node. The Wait function called from the master thread pauses execution (sleeps) in the master thread until at least one worker idle. The fringe for these functions is in shared memory and is always passed by reference. Assume the shared fringe object can be safely modified from multiple threads.

The pseudo-code of A*-Parallel is as follows. It first waits for some worker to be free, then (if needed) waits until the fringe is non-empty so the worker can be assigned the next node to be expanded from the fringe. If all workers have become idle while the fringe is still empty, this means no insertion in the fringe will happen anymore, which means there is no path to a goal so the search returns failure.

```
1: function A*-Parallel(problem, fringe, workers)
2:     closed ← an empty set
3:     fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
4:     loop do
5:         while All-Busy(workers) do Wait
6:         while fringe is empty do
7:             if All-Idle(workers) and fringe is empty then
8:                 return failure
9:             else Wait
10:        node ← Remove-Front(fringe)
11:        if Goal-Test(problem, State[node]) then
12:            if Should-Return(node, workers, fringe) then
13:                return node
14:        if State[node] is not in closed then
15:            add State[node] to closed
16:            Get-Idle-Worker(workers).Worker-Expand(node, problem, fringe)

17: function Worker-Expand(node, problem, fringe)
18:     child-nodes ← Expand(node, problem)
19:     fringe ← Insert-All(child-nodes, fringe)
```

Because there are workers acting in parallel it is not a simple task to determine when a goal can be returned: perhaps one of the busy workers was just about to add a really good goal node into the fringe. We look at how to finish the implementation of the Should-Return function called before returning a goal node in A*-Parallel. Assume that the state space is finite, and the heuristic used is consistent. Here are four implementations.

```
I    function Should-Return(node, workers, fringe)
         return true

II   function Should-Return(node, workers, fringe)
         return All-Idle(workers)

III  function Should-Return(node, workers, fringe)
         fringe ← Insert(node, fringe)
         return All-Idle(workers)

IV   function Should-Return(node, workers, fringe)
         while not All-Idle(workers) do Wait
         fringe ← Insert(node, fringe)
         return F-Cost[node] == F-Cost[Get-Front(fringe)]
```

For each of the implementations, please indicate whether it results in a *complete* search algorithm, and whether it results in an *optimal* search algorithm. And give a brief justification for your answer.

complete implementation(s)(2 pts):
Justify your answer(4 pts):

optimal implementation(s)(2 pts):
Justify your answer(4 pts):

3 Initialization of Neural Networks (25 pts)

In this question we will discuss what kind of initialization is reasonable for training neural networks. Consider a deep feed-forward network with ReLU activation. Note that we *do not* use bias, so it can be writen as

$$x_i^l = \sum_{j=1}^{N_{l-1}} W_{ij}^l f(x_j^{l-1}), \quad 1 \le i \le N_l, 1 \le l \le D, \tag{1}$$

where $\boldsymbol{x}^0 = (x_i^0)_{i=1}^{N_0}$ denotes the input, $\boldsymbol{x}^D$ denotes the output, and $f(x) = \max(x, 0)$ is the ReLU function. The weights $\boldsymbol{W} = \{W_{ij}^l\}$ are initialized with independent normal distributions:

$$p_{\text{init}}(\boldsymbol{W}) = \prod_{l,i,j} \mathcal{N}(W_{ij}^l | 0, \sigma_l^2). \tag{2}$$

(i) (15 pts)Under a reasonable initialization scheme, the variance of the network output $\text{Var}_{\text{init}}[\boldsymbol{x}^D] \overset{\text{def}}{=} \text{Var}_{\boldsymbol{W} \sim p_{\text{init}}, \boldsymbol{x_0}}[\boldsymbol{x}^D]$ should neither explode nor vanish when the network depth or width grow unbounded. A sufficient condition for this is:

$$\text{Var}_{\text{init}}[x_i^l] = \text{Var}_{\text{init}}[x_j^{l-1}], \quad \forall l > 1, i, j. \tag{3}$$

Suppose that $\{x_i^0\}$ are i.i.d. and symmetric (i.e., $P(x_i^0 > a) = P(x_i^0 < -a)$), $\{W_{ij}^l\}, l = 1, \ldots, D$ and $\{x_i^0\}$ are independent. Further suppose $\sigma_l = \frac{\alpha}{\sqrt{N_{l-1}}}$. Find α such that Eq. 3 holds.

Note: The PDF of x_i^l might not be well-defined. Even if we assume x_j^0 follow continuous probability distribution, e.g. $\mathcal{U}(-1, 1)$, due to the ReLU, x_i^1 are not continuous random variables since $p(x_i^1 = 0) \ge \frac{1}{2^{N_0}}$.

Hint: We only take i.i.d and symmetric assumptions of the input $\boldsymbol{x}^0$. Please show that these properties also hold for $l \ge 1$. Then prove that $\mathbb{E}[f(x_i^l)^2] = \frac{1}{2} Var(x_i^l)$.

(ii) (10 pts) Another condition the initialization scheme should satisfy is that, as the network depth or width grows unbounded, the back-propagated gradient signal,

$$\text{Var}_{\text{init}} \left[\frac{\partial \text{Loss}(\boldsymbol{W})}{\partial x_i^l} \right] \tag{4}$$

should neither explode nor vanish (so that the gradient w.r.t. weights will not explode or vanish).

Assume that $N_l = N_{l+1}$, $\{\frac{\partial \text{Loss}(\boldsymbol{W})}{\partial x_j^{l+1}}\}$ are i.i.d. with zero mean, $\{\frac{\partial \text{Loss}(\boldsymbol{W})}{\partial x_j^{l+1}}\}$, $\{W_{ji}^{l+1}\}$ and $\{x_i^l\}$ are independent, and $\{x_i^l\} \sim \mathcal{U}(-1, 1)$.

Above assumption might be implausible, but you can safely use it.

Show that the initialization scheme determined in the previous problem also satisfies this condition, whose sufficient condition is:

$$\mathrm{Var}_{\mathrm{init}}\left[\frac{\partial \mathrm{Loss}(\boldsymbol{W})}{\partial x_i^l}\right] = \mathrm{Var}_{\mathrm{init}}\left[\frac{\partial \mathrm{Loss}(\boldsymbol{W})}{\partial x_j^{l+1}}\right]$$

4 MDP: Blackjack (20 pts)

We will treat the Blackjack game as an MDP. The game has states 0, 1, ...,8, corresponding to dollar amounts, and a *Done* state where the game ends. The player starts with \$2, i.e., at state 2. The player has two actions: Stop and Roll, and is forced to take the Stop action at states 0, 1, and 8.

When the player takes the Stop action, they transition to the *Done* state and receive a reward equal to the number of dollars of the state they transitioned from: e.g., taking the stop action at state 3 gives the player \$3. The game ends when the player transitions to *Done*.

The Roll action is available from states 2-7. The player rolls a biased 6-sided die that will land on 1, 2, 3, or 4 with 1/8 probability each and 5 or 6 with probability 1/4 each.

If the player Rolls from state s and the die lands on outcome o, the player transitions to state $s + o - 2$, as long as $s + o - 2 \leq 8$. (s is the number of dollars of the current state, o is the amount rolled, and the negative 2 is the price to roll). If $s + o - 2 > 8$, the player busts, i.e., transitions to *Done* and does NOT receive a reward.

(i) (10 pts) We consider using policy iteration. Our initial policy $\pi^0(s)$ is in the table below. Evaluate the policy at each state with $\gamma = 1$. (4 pts) Then please write down the *optimal* policy $\pi^*(s)$. (4 pts) Based on the policy iteration algorithm, use one or two sentences to describe the evidence supporting your proposed $\pi^*(s)$ is optimal. (2 pts) If two actions are equally good, you should write both Roll and Stop. Note that the action at states 0, 1, and 8 is fixed in the rule, so we will not consider those states in the update.

You can write down your answer directly without a detailed computing process.

State	2	3	4	5	6	7
$\pi^0(s)$	Roll	Stop	Roll	Stop	Stop	Stop
$V^{\pi^0}(s)$						
$\pi^*(s)$						

(ii) (6 pts) Suppose we do not have access to the transition ground truth, i.e., we can not see the die. Now we alternatively turn to Q-Learning. The initial Q-table is the following:

$a \setminus s$	2	3	4	5	6	7
Roll	0	0	5	3	4	2
Stop	2	3	4	5	6	7

Given an observed trajectory as follows:

$$(s = 2, a = \text{Roll}, s' = 4, r = 0),$$
$$(s = 4, a = \text{Roll}, s' = 7, r = 0),$$
$$(s = 7, a = \text{Stop}, s' = \textit{Done}, r = 7).$$

What will the Q-table be like after running one pass of Q-learning over the given trajectory? (6 pts) Suppose discount rate $\gamma = 1$, and learning rate $\alpha = 0.5$.

(iii) (4 pts) We will perform Deep Q-Learning in this part. Suppose the actions are assigned as $a = 1$ for Roll and $a = 0$ for Stop. We use a linear approximation of the Q-values. I.e., $Q_{\mathbf{w}}(s, a) = w_0 + w_1 s + w_2 a$, where $\mathbf{w} = [w_0, w_1, w_2]^T$ is learnable weight. The objective is the following:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\Big[Q_{\mathbf{w}}(s, a) - \text{sg}\big(r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a')\big)\Big]^2,$$

where r is reward and $\text{sg}(\cdot)$ means stop gradient, i.e., we treat its operand as constant and not backpropagate gradients. With learning rate α, please calculate the gradient and describe how we update $\mathbf{w}$ (2 pts).

With the above parametrization of Q, can we express the optimal policy π^* we obtain in question (i)? (2 pts) Here, we greedily take action based on Q-functions. I.e., $\pi(s) = \arg\max_a Q(s, a)$.

5 Diffusion Probabilistic Models (22 pts)

Diffusion Probabilistic Models (DPMs) have attracted lots of attention recently due to their impressive data-generating performance. Popular image generation models like Dall-E 2 and StableDiffusion are all built upon DPMs. In this part, you will be asked to investigate some of their properties based on the knowledge of probabilistic models. It is OK that you are unfamiliar with DPMs. We will only focus on its probabilistic formulation and ignore sophisticated implementation details. Meanwhile, the background required will be provided in the context.

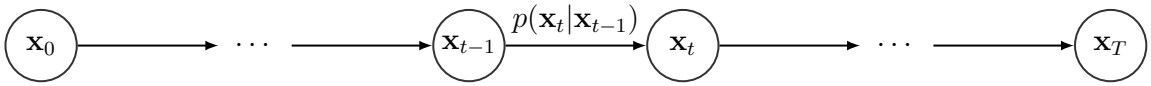

Figure 3: Diffusion process.

A DPM consists of a forward process $\mathbf{x}_0 \to \mathbf{x}_T$ (or diffusion process), in which a datum (generally an image) is progressively noised, and a reverse process $\mathbf{x}_T \to \mathbf{x}_0$ (or reverse diffusion process), in which noise is transformed back into a sample from the target distribution. DPMs follow Markov assumptions. Figure 3 depicts the forward process.

We first look at the forward process. At every time step, we gradually corrupt $\mathbf{x}_{t-1}$ by additive Gaussian noise and get $\mathbf{x}_t$. In particular, each step of the forward process adds Gaussian noise according to some variance schedule given by $\beta_t \in \mathbb{R}$:

$$p_{\mathbf{x}_t|\mathbf{x}_{t-1}}(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

Hereafter we omit subscripts of probabilities if there is no ambiguity, e.g., $p_{\mathbf{x}_t|\mathbf{x}_{t-1}}(\mathbf{x}_t|\mathbf{x}_{t-1})$ will be denoted by $p(\mathbf{x}_t|\mathbf{x}_{t-1})$.

(i) (8 pts) **Forward process.** Please write $\mathbf{x}_t$ in the form of a linear combination of $\mathbf{x}_{t-1}$ and a standard Gaussian noise $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. I.e., $\mathbf{x}_t = a\mathbf{x}_{t-1} + b\epsilon_t$, where you should derive parameter a and b. (2 pts) Then please apply this recursively to derive $\mathbf{x}_t$ in the form of a linear combination of $\mathbf{x}_0$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. (4 pts) Finally, show that $p(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$, where $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_s^t \alpha_t$. (2 pts).

(ii) (6 pts) **Reverse process.** From the last part, we know that with a proper setting of β_t and sufficiently large steps T, we will ultimately have $\mathbf{x}_T = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. Thus for the reverse process, we start at the standard Gaussian noise.

Then we wish to denoise $\mathbf{x}$ step by step to reach the target distribution from which we sample realistic data. A straightforward method is to sample from the forward process posteriors. However, these posteriors are only tractable when conditioned on $\mathbf{x}_0$.

Please show that $p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is Gaussian. (4 pts)

Moreover, previous literature has proved that if $\beta_t \to 0$, $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$ will be Gaussian as well. Inspired by this, we use neural networks to parameterize the reverse process as a Markov chain with learned Gaussian transitions:

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t; \theta) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t)).$$

This can be interpreted as we predict $\mathbf{x}_0$ from $\mathbf{x}_t$ and t using neural networks.

After training, we learn a reasonable θ. Given the learned θ, we are ready for sampling.

Please complete Algorithm 1. (2 pts)

Here, $x_T \in \mathbb{R}^d$ is a vector sample from the distribution of random variable $\mathbf{x}_T$, i.e., $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Note that μ_θ and $\mathbf{\Sigma}_\theta$ are both given as trained neural networks. For $x_t \in \mathbb{R}^d$, the output of the neural networks are $\mu_\theta(x_t, t) \in \mathbb{R}^d$ and $\mathbf{\Sigma}_\theta(x_t, t) \in \mathbb{R}^{d \times d}$.

Algorithm 1 Sampling

$x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  Your line ①
  Your line ② ▷ You may use less or more than 2 lines
end for
return x_0

(iii) (8 pts) **Classifier guidance sampling.** We have already completed the sampling process. However, it is *unconditioned*. The generated data is realistic but may not be what we want. Hereafter we will investigate classifier guidance sampling to drive the reverse process to generate data with desired class labels.

The conditioned forward process's probabilistic graph is $y \to \mathbf{x}_0 \to \cdots \to \mathbf{x}_T$, where y is the class label.

Please show the following equality holds. (3 pts)

$$p(\mathbf{x}_t|\mathbf{x}_{t+1}, y) = \frac{p(\mathbf{x}_t|\mathbf{x}_{t+1})p(y|\mathbf{x}_t)}{p(y|\mathbf{x}_{t+1})}. \tag{5}$$

Hint: You may first prove that y and $\mathbf{x}_{t+1}$ are independent conditioned on $\mathbf{x}_t$.

Provided with Equation 5, we can sample from

$$p(\mathbf{x}_t|\mathbf{x}_{t+1}, y) = Zp(\mathbf{x}_t|\mathbf{x}_{t+1})p(y|\mathbf{x}_t)$$

where Z is a normalizing constant. This is because when sampling $\mathbf{x}_t$, $p_{y|\mathbf{x}_{t+1}}(y|\mathbf{x}_{t+1};\phi)$ is given as constant (see the for-loop of Algorithm 1). Now we can derive classifier guidance sampling.

In practice, we use a trained neural network classifier $p_{y|\mathbf{x}_t}(y|\mathbf{x}_t;\phi)$ as $p(y|\mathbf{x}_t)$. Thus we assume $p_{y|\mathbf{x}_t}(y|\cdot;\phi)$ is smooth enough to have first-order Taylor expansion with Lagrange's form of remainder, and the norm of its Hessian matrix has an upper bound $k||\mathbf{\Sigma}^{-1}||$, since $||\mathbf{\Sigma}|| \to 0$ when T is sufficiently large. The definition of matrix norms and vector norms satisfies $||A|| = \sup_{\mathbf{x}\neq 0} \frac{||A\mathbf{x}||}{||\mathbf{x}||}$ and $|u^T v| \leq ||u|| \cdot ||v||$.

Please show that there exists $\delta > 0$ such that $|e| \leq k||\mathbf{\Sigma}^{-1}|| \cdot ||\mathbf{x} - \mu||^2$ for all $\mathbf{x}$ satisfying $||\mathbf{x} - \mu|| < \delta$ in the following equation(5 pts)

$$\log(p_{\mathbf{x}_t|\mathbf{x}_{t+1}}(\mathbf{x}|\mathbf{x}_{t+1};\theta)p_{y|\mathbf{x}_t}(y|\mathbf{x};\phi)) = \log \mathcal{N}(\mathbf{x};\mu + g\mathbf{\Sigma},\mathbf{\Sigma}) + C + e, \tag{6}$$

where $\mu = \mu_\theta(\mathbf{x}_{t+1}, t+1)$, $\mathbf{\Sigma} = \mathbf{\Sigma}_\theta(\mathbf{x}_{t+1}, t+1)$, $g = \nabla_{\mathbf{x}} \log p_{y|\mathbf{x}_t}(y|\mathbf{x})|_{\mathbf{x}=\mu}$, and C is a constant w.r.t. $\mathbf{x}$.

Hint: It might be useful to take logarithms over Gaussian PDF:

$$\log p_{\mathbf{x}_t|\mathbf{x}_{t+1}}(\mathbf{x}|\mathbf{x}_{t+1}) = -\frac{1}{2}(\mathbf{x} - \mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \mu) + C_1.$$