THE UNIVERSITY OF HONG KONG

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

DASC7606 Deep Learning

Date: Wednesday, May 18, 2022 **Time: 6:30 p.m. – 8:30 p.m.**

Answer ALL questions. They are all COMPULSORY.

The mark value of each question (or part of a question) is indicated before the question (or part of the question).

Please write your answers on separate sheets of paper.

Candidates are permitted to refer to any printed/handwritten materials in the examination. Internet searching and crowdsourcing from group messages, online forums or social media, etc. are strictly forbidden.

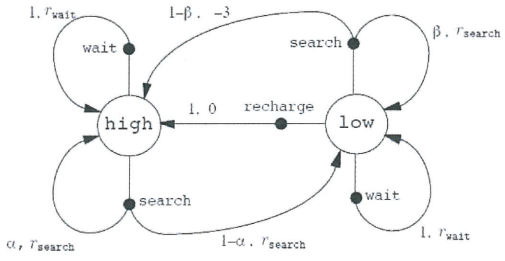
Only approved calculators as announced by the Examinations Secretary can be used in this examination. It is candidates' responsibility to ensure that their calculator operates satisfactorily, and candidates must record the name and type of the calculator used on the front page of the examination script.

(30 pts) Question 1: Reinforcement Learning

The Markov Decision Process (MDP) of a Recycling Robot with two states is given below:

- ***S***: { l = low, h = high } describing battery condition
- ***A***: { s = search, w = wait, rc = recharge }
- ***P***: transition probability distribution

s	a	s'	$p(s'\|s,a)$	$r(s,a,s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	0.



Let $Q(S, A)$ = expected cumulative reward starting from state S, following policy π, and taking action A.

(a) (4 pts) What is the update formula for $Q(S, A)$ in Q-learning?

Assuming $r_{\text{search}} = 1$, $r_{\text{wait}} = 0.25$, learning rate $\alpha = 0.5$, discount factor $\gamma = 0.5$, and the training episode comprised of (state, action) is: $(h, s) \mid (h, w) \mid (h, s) \mid (l, w) \mid (l, s) \mid (h, \ldots$

(b) (5 pts) Initially $Q(S, A) = 0$ for all entries and the start state is "high". Based on the given episode and the Q-learning algorithm, update $Q(S, A)$ after the first action "search". (Hint: From the episode, the next state is "high".)

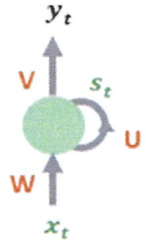
(c) (5 pts) Update $Q(S, A)$ after the second action (h, w) in the episode (or after the first two actions in the episode $(h, s) \mid (h, w)$).

(d) (10 pts) Update $Q(S, A)$ after the first 3, 4 and 5 actions in the episode $(h, s) \mid (h, w) \mid (h, s) \mid (l, w) \mid (l, s) \mid (h, \ldots$

(e) (6 pts) What is the optimal policy at this point?

(30 pts) Question 2: Binary number addition – RNN

A recurrent neural network (RNN) can be used to add two n-bit numbers, $P = p_{n-1}\, p_{n-2} \ldots p_0$ and $Q = q_{n-1}\, q_{n-2} \ldots q_0$, where $Y = P+Q = y_n\, y_{n-2} \ldots y_0$.



x_t: input at time t
s_t: state at time t
y_t: output at time t
U, V, W: weight matrices
B, b: biases

$$s_t = g(Wx_t + Us_{t-1} + B)$$
$$y_t = g(Vs_t + b)$$

The input $x_t = (p_t, q_t)$ are two binary bits and output y_t is one binary bit.

(a) (4 pts) Assuming s_t has four internal nodes $\{n_1, n_2, n_3, n_4\}$, what are the dimensions of W, U and V?

(b) (4 pts) Show the nodes and connecting lines (of the RNN at time t) between x_t and s_t, s_t and y_t, and their corresponding weights V and W.

(c) (6 pts) Let c_t be the internal node n_1 at time t which contains the "carry bit" information to be passed from s_t to s_{t+1}, where

$$c_t = 1 \text{ iff } c_{t-1} + p_t + q_t \geq 2 \text{ and } c_t = 0 \text{ otherwise}$$

With $s_t = g(Wx_t + Us_{t-1} + B)$, what are the corresponding weights in W and U and bias B for defining c_t accordingly? Give a mathematical formula for the activation function g?
(Hint: All values are binary. Let g be a step function. Choose an appropriate value for the bias so that ≥ 2 can be ensured.)

(d) (6 pts) The other internal nodes $\{n_2, n_3, n_4\}$ carry information for outputting y_t. We want $y_t = 1$ iff $c_{t-1} + p_t + q_t = 1$ or 3 and $y_t = 0$ otherwise.

So let us define

node $n_2 = 1$ iff $c_{t-1} + p_t + q_t \geq 1$ and $n_2 = 0$ otherwise
node $n_3 = 1$ iff $c_{t-1} + p_t + q_t \leq 1$ and $n_3 = 0$ otherwise
node $n_4 = 1$ iff $c_{t-1} + p_t + q_t \geq 3$ and $n_4 = 0$ otherwise

What are the corresponding weights in W and U and bias B for defining these inequalities in internal nodes n_2, n_3 and n_4 accordingly?

(e) (10 pts) What are the corresponding weights in V and bias b for the output $y_t = g(Vs_t + b)$?
(Hint: Consider all cases of c_{t-1}, p_t, q_t and the corresponding values of n_2, n_3, n_4.)

(40 pts) Question 3: Generative Adversarial Network

Suppose you are given a Generative Adversarial Network (GAN) with a generator and a discriminator.

(a) (5 pts) Assume that the discriminator evaluates the real data and gives answer 0.4. Which part(s) of GAN can be trained and what is(are) the loss value(s)?

(b) (5 pts) Assume that the discriminator evaluates the data produced by the generator and gives answer 0.7. Which part(s) of GAN can be trained and what is(are) the loss value(s)?

(c) (6 pts) Referring to above, can both parts of the network be trained simultaneously by backpropagation? Justify your answer.

(d) (4 pts) In Assignment 2, loss function L1 was used for training. Suppose that

the ground truth $T_1 = \begin{bmatrix} 0.5 & 0.6 & 0.8 \\ 0.2 & 0.4 & 0.3 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$ and the generated result $G_1 = \begin{bmatrix} 0.4 & 0.7 & 0.7 \\ 0.4 & 0.5 & 0.4 \\ 0.3 & 0.2 & 0.1 \end{bmatrix}$

What is the formula for L1? What is the L1 loss for the above two matrices?

(e) (20 pts) There are five "XXX" in the following code for Pix2Pix GAN. Replace "XXX" by False or True for a valid program. Explain each of your answers.

```
class GAN():
    def __init__(self):
            xxxx

    def forward(self):
        """Run forward pass; called by both functions <optimize_parameters> and
    <test>."""
        self.fake_B = self.netG(self.real_A) # G(A)

    def backward_D(self):
        """Calculate GAN loss for the discriminator"""
        # Fake; stop backprop to the generator by detaching fake_B
        fake_AB = torch.cat((self.real_A, self.fake_B),1) # we use conditional
    GANs; we need to feed both input and output to the discriminator
        pred_fake = self.netD(fake_AB.detach) # inference the fake sample
        self.loss_D_fake = self.criterionGAN(pred_fake, XXX)
        # Real
        real_AB = torch.cat((self.real_A, self.real_B), 1)
        pred_real = self.netD(real_AB) # inference the real sample
        self.loss_D_real = self.criterionGAN(pred_real, XXX)
        # combine loss and calculate gradients
        self.loss_D = (self.loss_D_fake + self.loss_D_real) * 0.5
        self.loss_D.backward()

    def backward_G(self):
        """Calculate GAN and L1 loss for the generator"""
```

```
        # First, G(A) should fake the discriminator
        fake_AB = torch.cat((self.real_A, self.fake_B), 1)
        pred_fake = self.netD(fake_AB)
        self.loss_G_GAN = self.criterionGAN(pred_fake, XXX)
        # Second, G(A) = B
        self.loss_G_L1 = self.criterionL1(self.fake_B, self.real_B) * self.opt.
    lambda_L1
        # combine loss and calculate gradients
        self.loss_G = self.loss_G_GAN + self.loss_G_L1
        self.loss_G.backward()

    def optimize_parameters(self):
        self.forward() # compute fake images: G(A)
        # update D
        self.set_requires_grad(self.netD, XXX) # enable backprop for D
        self.optimizer_D.zero_grad() # set D's gradients to zero
        self.backward_D() # calculate gradients for D
        self.optimizer_D.step() # update D's weights
        # update G
        self.set_requires_grad(self.netD, XXX) # D requires no gradients when
    optimizing G
        self.optimizer_G.zero_grad() # set G's gradients to zero
        self.backward_G() # calculate graidents for G
        self.optimizer_G.step() # udpate G's weights

if __name__ == '__main__':
    model = GAN()
    model.optimize_parameters()
```

*** END OF EXAM ***