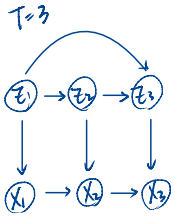
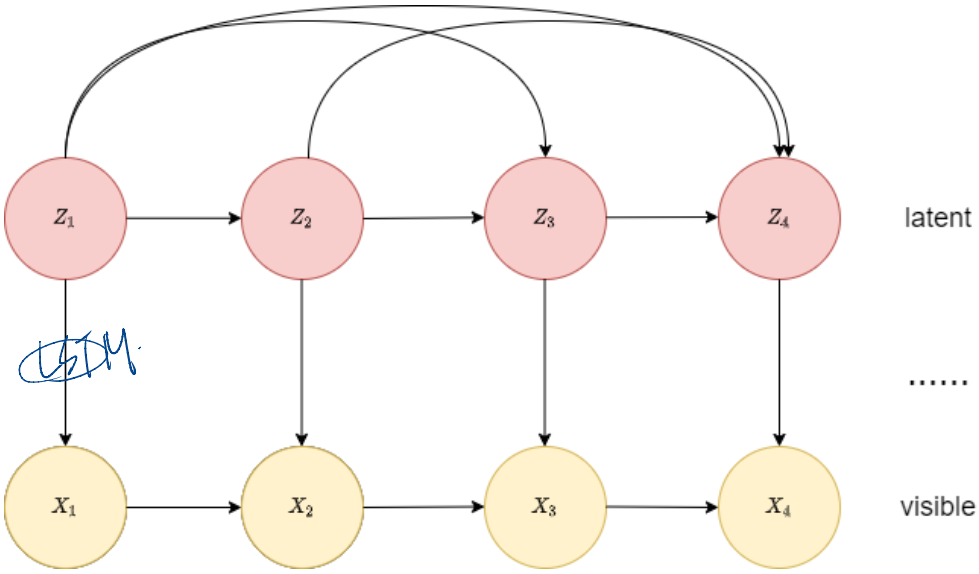
Deep Learning — Final Exam

Date: Thursday, September 1, 2022
Time: 01:20pm – 04:20pm (180 minutes)
Format: Open book
Instructions:
1) You may give your answers in Chinese or English.
2) Please give your answers in succinct phrases or point form.
3) Please write your answers clearly (with explicit denotation of labels and symbols used).

1. (15 pts) Consider the following latent factor model, where $Z_i, i = 1, 2, \ldots, T$ are latent variables and $X_i, i = 1, 2, \ldots, T$ are visible variables.





(a) (3 pts) Design an inevitable mapping from X_i's to their latent representations Z_i's using the flow model idea. Use $T = 3$ as an example. Think about how you should convert each of the X_i's into the corresponding Z_i's and what the conditioning variable should be utilized in each of these conversions. The conversions may involve coupling layers as fundamental building blocks.

(b) (3 pts) According to the graphical model, factorize the joint distribution $p(Z_{1-T})$.

(c) (3 pts) Describe how you would evaluate the probability $p(X_1, X_2, X_3)$ of one specific sample X_1, X_2, X_3 once the model is trained.

(d) (3 pts) How would you train the flow model in (a)?

(e) (3 pts) Describe how you would sample X_i's from Z_i's using the designed flow model.

(a)

(b) $p(Z_{1:T}) = p(Z_1, Z_2, \ldots, Z_T) = \prod_{k=1}^{T} p(Z_k | Z_{1:k-1})$, where $k \in \{1, 2, \ldots, T\}$

(c) $p(X_1, X_2, X_3) = p(X_1|Z_1)\, p(X_2|X_1, Z_2)\, p(X_3|X_2, Z_3)$

2. (28 pts) Following the latent factor model given in Problem 1, design a variational autoencoder to learn the distribution $p(X_1, X_2, \ldots, X_T)$.

(a) (3 pts) According to the graphical model, factorize the encoding distribution $q(Z_{1-T}|X_{1-T}) = \Pi_{t=1}^{T} q(Z_t|\text{"}\ldots\text{"})$. Apply the d-separation rule to identify the conditioning factors in "$\ldots$".

(b) (3 pts) By observing $q(Z_t|\text{"}\ldots\text{"})$, choose a neural network to implement it. You must specify (a) the network type, e.g. CNN or RNN, and the rationale; (b) indicate explicitly its input and output, and whether there is any hidden connection and/or output recurrence; and (c) specify the assumption about the distribution.

(c) (3 pts) According to the graphical model, factorize the decoding distribution $p(X_{1-T}|Z_{1-T}) = \Pi_{t=1}^{T} p(X_t|\text{"}\ldots\text{"})$. Apply the d-separation rule to identify the conditioning factors in "$\ldots$".

(d) (3 pts) By observing $p(X_t|\text{"}\ldots\text{"})$, choose a neural network to implement it. You must specify (a) the network type, e.g. CNN or RNN, and the rationale; (b) indicate explicitly its input and output, and whether there is any hidden connection and/or output recurrence; and (c) specify the assumption about the distribution.

(e) (3 pts) Choose a neural network to implement the prior distribution $P(Z_1, Z_2, \ldots, Z_T)$. You must specify (a) the network type, e.g. CNN or RNN, and the rationale; (b) indicate explicitly its input and output, and whether there is any hidden connection and/or output recurrence; and (c) specify the assumption about the distribution.

(f) (10 pts) Draw a diagram to connect all these networks together for end-to-end training and specify explicitly the loss function. If re-parameterization is needed, indicate where it should be used.

(g) (3 pts) Describe how you would sample X_i's once the model is trained.

3. (20 pts) Convolution, dropout, activation function, and CNN

   (a) (2 pts) Describe the reasons we don't use a full image for Transformer and how ViT addresses the issues.

   (b) (3 pts) Explain the idea of dropout.

   (c) (3 pts) What is the size of the output feature map for an 256 x 256 input image after convolution with kernel (3,3), padding (2,2), and stride (2,2)?

   (d) (6 pts) What distinguishes CNN, RNN, and an attention module? What benefit does the attention module provide?

   (e) (6 pts) What may cause gradient vanish problem and how to solve it? Explain your answer.

(a) When we using Tranformer for image data,

using the full image as input will cost a lot of computation and memory.

Vision Transformer (ViT) solve this issue by dividing the image into smaller patches.

Each patches is treated as a token, similar to a word in NLP tasks.

↓

it use self-attension to capture relationships between different parches.

(b) Dropout 通过在训练过程随机丢弃一定比例的神经元，使神经网络不会过度依赖部分神经元

降低了 overfitting 的可能性

(c) The output size = $\lfloor$ (input size + 2× padding − ~~kernel size~~ (dilation × kernel size)) / stride $\rfloor$ + 1

= $\lfloor (256 + 2\times2 - 3) / 2 \rfloor + 1$

= 128 + 1

= 129

↓

3 + 2n = 256 + 2×2

kernel stride input padding

↓

n = 128.5 取 n = 128

↓

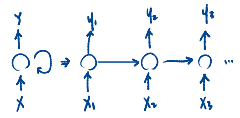
output size = 129 × 129

(d) CNN：多用於影像任務上，由多個卷積層和池化層所構成

利用這兩種網路層取得影像中的特徵及模式

RNN：多用於處理序列資料 (ex. text, time-series data)

具有遞迴結構，能依據資料的順序關係依次輸入，並使用前一次的隱藏層結果當作次的計算



Attention module：利用 Q、K、V 三元組，代表注意中相關，

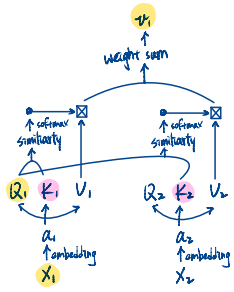
通過權重分配，使模型能專注於當前任務最相關的輸入部分

其目的是想替代 RNN 原本帶的的事情，只是不用一個接一個地處理

↳ 使用 attention module 的好處有以下

① 藉由 self-attention，使 CNN 在輸入資料維度較大時能大幅減少所需時間

② 不用像 RNN 按序處理資料，可平行化進行



(e) 在反向傳遞時，微分的連乘可能小於 1，

當層數越多，這些數字連乘就可能導致梯度趨近 0 (消失) 的問題

① 可以利用 Residual Connection 來解決這個問題

↓ 殘差連接

通過添加在某些層的上跳連接，可以保留梯度傳播，有助於訓練更深層的神經網路

↓

我們應盡所用的連接數

② 可以在反向傳遞過程中設置界值將梯度限制在一範圍中，以防止梯度爆炸或消失的問題

4. (14 pts)

(a) RL3-P.42 (a) (7 pts) Prove the following theorem: For any ε-greedy policy π, the ε-greedy policy π' with respect to q_π is an improvement, that is $v_{\pi'}(s) \geq v_\pi(s)$.

(b) (7 pts) Show how ε-greedy policy improvement will converge to the optimal.

(b) The ε-greedy policy improvement can converge to the optimal policy by decreasing ε over time.

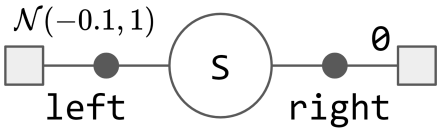
As ε decreases, the policy becomes more greedy, which means it selecting action with higher action values more frequently.

① When ε is initially high (close to 1), the policy is exploratory, as it selects random actions with equal probability. This exploration allows the agent to gather the information about the environment and learn the values.

② As the training process, ε will be decreasing by decay mechanism. This reduction in ε let the policy from exploration to choosing higher action values gradually.

③ As ε close to zero, the policy becomes purely greedy, and always selecting the action with highest value. At this point, the policy has converge to the optimal policy, as it consistently selects the actions that maximize the expected cumulative reward.

5. (18 pts) Consider the undiscounted MDP shown below.



Episodes start in S with a choice between two actions, left and right. The right action transitions immediately to the terminal state with a reward of zero. The left action also transitions to the terminal state but with a reward drawn from a normal distribution with mean -0.1 and variance 1.0.

(a) (2 pts) What is the optimal state-action value $Q^*(S, \text{left})$ and $Q^*(S, \text{right})$?

(b) (2 pts) What is the probability of taking the left action using the ε-greedy policy ($\varepsilon = 0.1$) with respect to Q^*?

(c) (14 pts) What problem will you encounter when training with Deep Q-learning (DQN)? In order to avoid the problem, what method can we apply?

$a = \begin{cases} \underset{a}{\text{argmax}}\ Q(s,a), & \text{prob.} = 1-\varepsilon \\ \text{random}, & \text{prob.} = \varepsilon \end{cases}$

(a) $Q^*(s, \text{left}) = \mathbb{E}[\mathbb{E}_{\sim \mathcal{N}(-0.1, 1)}] = -0.1$ (expected reward)

$Q^*(s, \text{right}) = 0$ *

(b) The ε-greedy policy involves choosing the action with the highest state-action value with a prob. of $1-\varepsilon$ (exploitation), and choosing a random action with a prob. of ε.

$\Rightarrow a = \begin{cases} \underset{a}{\text{argmax}}\ Q^*(s,a) = Q^*(s, \text{right}), & \text{prob.} = 1-\varepsilon = 0.9 \\ \text{random}, & \text{prob.} = \varepsilon = 0.1 \end{cases}$

Therefore, we obtain the prob. of taking the left action using the ε-greedy policy is 0.1 *

(c) When training with DQN, one problem that can be encountered is the issue of overestimation of state-action value.

(First, DQN uses an approximation function (typically a neural network) to estimate the state-action value, so the neural network may tends to overestimate the values for certain actions, and the overestimation will lead to suboptimal policies and slower convergence.)

To address the issue, we can use DDQN, which uses two approximation functions that are trained on different samples, one for selecting the best action and the other for calculating the value of this action, since these two functions seen two different samples, it is unlikely that they overestimate the same action.