سوال ۱ –

قسمت الف)

ابتدا MPN را تعریف میکنیم. اگر S یک SPN روی X باشد، MPN هه $\hat{S}$ از S ، شبکه ای با ویژگی های زیر است:

۱- هر گره distribution هه D را با یک گره maximizing distribution هه $\widehat{D}$ جایگزین میکنیم به طوری که :

$$\widehat{D} : H_{sc(D)} \to [0, \infty] \qquad , \qquad \widehat{D}(Y) \coloneqq \max_{y \in Y} D(y)$$

۲- هر گره ***sum*** هه ***S*** را با یک گره ***max*** هه $\hat{S}$ جایگزین میکنیم. یال های خروجی گره ***max*** همان وزن های یال های خروجی گره ***sum*** متناظر را دارند. یک گره ***max*** اینگونه تعریف میشود:

$$\hat{S} \coloneqq \max_{C \in ch(\hat{S})} w_{\hat{S},C}\, C$$

۳- هر گره ***product*** در ***SPN*** یک گره ***product*** در ***MPN*** است.

حال فرضیه زیر، برای ***augmented SPN*** ها ، ***MPN*** میتواند احتمال ***MPE*** را محاسبه کند:

فرضیه: اگر ***S*** یک ***augmented SPN*** روی ***X*** باشد که ***X*** از قبل ***RV*** های مدل ***SPN*** اوریجینال را تشکیل داده و همچنین ***RV*** های پنهان را که توسط ***augmentation*** بوجود آمده اند، را نیز تشکیل داده است. اگر $\hat{S}$ همان ***MPN*** متناظر آن باشد، با فرض اینکه ***N*** یک گره دلخواه در ***S*** و $\widehat{N}$ گره متناظر آن در $\hat{S}$ باشد، برای هر $X \in H_{sc(N)}$ داریم:

$$\widehat{N}(X) = \max_{x \in X} N(x)$$

یعنی *MPN* ، احتمال *augmented SPN* روی *X* را بیشینه میکند.الگوریتم آن میشود:

Algorithm 5 MPE inference in augmented SPNs

1: For all sums S, let $\mathbf{ch}'(\mathsf{S}) := \{\mathsf{C}_{\mathsf{S}}^k \mid k \in \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}]\}$
2: Evaluate $\boldsymbol{\mathcal{X}}$ in corresponding MPN $\hat{\mathcal{S}}$ (upwards pass)
3: Initialize Q as empty queue
4: $Q \leftarrow$ root node
5: **while** Q not empty **do**
6: $\quad \mathsf{N} \leftarrow Q$
7: $\quad$ **if** N is a sum node **then**
8: $\quad\quad Q \leftarrow \arg\max_{\mathsf{C} \in \mathbf{ch}'(\mathsf{N})} \left\{ w_{\hat{\mathsf{N}},\hat{\mathsf{C}}}\, \hat{\mathsf{C}}(\boldsymbol{\mathcal{X}}[\mathbf{sc}(\hat{\mathsf{C}})]) \right\}$
9: $\quad$ **else if** N is a product node **then**
10: $\quad\quad \forall \mathsf{C} \in \mathbf{ch}(\mathsf{N}) : Q \leftarrow \mathsf{C}$
11: $\quad$ **else if** N is a distribution node **then**
12: $\quad\quad \mathbf{x}^*[\mathbf{sc}(\mathsf{N})] = \arg\max_{\mathbf{x} \in \boldsymbol{\mathcal{X}}[\mathbf{sc}(\mathsf{N})]} \mathsf{N}(\mathbf{x})$
13: $\quad$ **end if**
14: **end while**
15: **return** $\mathbf{x}^*$

میبینیم که بیشینه کردن یک گره product به مسئله بیشینه کردن به صورت مستقل تمام فرزندان آن گره کاهش پیدا میکند.همچنین بیشینه کردن یک گره sum ، به مسئله یافتن فرزندی با بیشینه وزن یال خروجی و یافتن انتصاب بیشینه برای این فرزند، کاهش پیدا میکند.الگوریتم بالا با روش back-tracking حالت بیشینه augmented SPN را میابد.

حال الگوریتم بالا را کمی تغییر میدهیم به طوری که روی یک arbitrary SPN اعمال شود اما پاسخ MPE هه augmented SPN متناظر را خروجی دهد.البته کمی پیش شرط وجود دارد که بخاطر خلاصه سازی، آن هارا به صورت انگلیسی می آورم:

1. $\forall \mathsf{S} \in \mathbf{S}(\mathcal{S}) : \boldsymbol{\mathcal{X}}[Z_{\mathsf{S}}] = \mathbf{val}(Z_{\mathsf{S}})$, i.e. Z_{S} is maximized over all its possible states.
2. The states of the RV corresponding to not visited sum nodes are not assigned, i.e. steps 28–30 are not performed.
3. All $\tilde{w}_{\mathsf{S},\mathsf{C}} \equiv 1$, which is the case when all twin weights are deterministic, i.e. for each sum

حال الگوریتم جدید عبارت است از:

Algorithm 6 MPE inference

For all sums S, let $\mathbf{ch}'(\mathsf{S}) := \{\mathsf{C}_\mathsf{S}^k \mid k \in \boldsymbol{\mathcal{X}}[Z_\mathsf{S}]\}$

$\forall \mathsf{S} \in \mathbf{S}(\mathcal{S}) : \forall \mathsf{C} \in \mathbf{ch}(\mathsf{S}) : \; \bar{w}_{\mathsf{S},\mathsf{C}} \leftarrow 1$
for all sum nodes $\mathsf{S} \in \mathbf{S}(\mathcal{S})$ **do**
  for $\mathsf{S}^c \in \mathbf{S}^c(\mathsf{S})$ **do**
    for $\mathsf{C} \in \{\mathsf{C} \in \mathbf{ch}(\mathsf{S}^c) \mid \mathsf{S} \notin \mathbf{desc}(\mathsf{C})\}$ **do**
      $\bar{w}_{\mathsf{S}^c,\mathsf{C}} \leftarrow \bar{w}_{\mathsf{S}^c,\mathsf{C}} \times \max\limits_{k \in \boldsymbol{\mathcal{X}}[Z_\mathsf{S}]} \bar{w}_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$
    end for
  end for
end for
Equip corresponding MPN $\hat{\mathcal{S}}$ with weights $w_{\hat{\mathsf{S}},\hat{\mathsf{C}}} \leftarrow \bar{w}_{\mathsf{S},\mathsf{C}} \times w_{\mathsf{S},\mathsf{C}}$
Evaluate $\boldsymbol{\mathcal{X}}$ in MPN $\hat{\mathcal{S}}$ (upwards pass)

$\mathbf{S} \leftarrow \mathbf{S}(\mathcal{S})$
Initialize Q as empty queue
$Q \leftarrow$ root node
while Q not empty **do**
  $\mathsf{N} \leftarrow Q$
  if N is a sum node **then**
    $k^* \leftarrow \operatorname*{arg\,max}\limits_{k:\, \mathsf{C}_\mathsf{N}^k \in \mathbf{ch}'(\mathsf{N})} \left\{ w_{\hat{\mathsf{N}},\hat{\mathsf{C}}_\mathsf{S}^k} \hat{\mathsf{C}}_\mathsf{S}^k(\boldsymbol{\mathcal{X}}[\mathbf{sc}(\hat{\mathsf{C}}_\mathsf{S}^k)]) \right\}$
    $Q \leftarrow \mathsf{C}_\mathsf{N}^{k^*}$
    $z_\mathsf{S}^* = k^*$
    $\mathbf{S} \leftarrow \mathbf{S} \setminus \{\mathsf{S}\}$
  else if N is a product node **then**
    $\forall \mathsf{C} \in \mathbf{ch}(\mathsf{N}) : Q \leftarrow \mathsf{C}$
  else if N is a distribution node **then**
    $\mathbf{x}^*[\mathbf{sc}(\mathsf{N})] = \operatorname*{arg\,max}\limits_{\mathbf{x} \in \boldsymbol{\mathcal{X}}[\mathbf{sc}(\mathsf{N})]} \mathsf{N}(\mathbf{x})$
  end if
end while

for $\mathsf{S} \in \mathbf{S}$ **do**
  $z_\mathsf{S}^* = \operatorname*{arg\,max}\limits_{k \in \boldsymbol{\mathcal{X}}[Z_\mathsf{S}]} \bar{w}_{\mathsf{S},\mathsf{C}_\mathsf{S}^k}$
end for

return $\mathbf{x}^*, \mathbf{z}^*$

منبع کمکی: تز دکترا تحت عنوان Foundations of Sum-Product Networks for probabilistic modeling
نوشته Robert Peharz

قسمت ب) طبق اسلاید های درس، استنتاج marginal برای SPN خطی(tractable) و برای BN به صورت NpHard است. استنتاج conditional هم به صورت مشابه، برای SPN خطی و برای BN به صورت NPHard است. استنتاج MPE برای هر دو NPHard است.

به عنوان منبع اضافی هم برای اطمینان از مقاله learning directed acyclic graph SPNs in sub-quadratic time استفاده کردم.

قسمت ج) تبدیل SPN به bayes net ممکن است: ابتدا SPN را به فرم نرمال در می آوریم. اگر SPN نرمال شده ما S باشد و P یک گره product در این SPN با L فرزند باشد، اگر فرض کنیم k گره sum از ریشه این شبکه تا گره product گفته شده با نام های v1,v2,...,vk باشند، داریم:

$$P_S\left(X_{scope(p)}\middle|H_{v1}=v_1^*,\dots.,H_{vk}=v_k^*\right)=\prod_{i=1}^{L}P_S(x_{scope(p)}|H_{v1}=v_1^*,\dots,H_{vk}=v_k^*)$$

حال الگوریتم زیر، نحوه تبدیل *bayes net* را به *SPN* نشان میدهد:

Algorithm 3 Build BN Structure

Input: normal SPN $\mathcal{S}$
Output: BN $\mathcal{B}=(\mathcal{B}_V,\mathcal{B}_E)$

```
R ← root of S
if R is a terminal node over variable X then
    Create an observable variable X
    B_V ← B_V ∪ {X}
else
    for each child R_i of R do
        if BN has not been built for S_{R_i} then
            Recursively build BN Structure for S_{R_i}
        end if
    end for
    if R is a sum node then
        Create a hidden variable H_R associated with R
        B_V ← B_V ∪ {H_R}
        for each observable variable X ∈ S_R do
            B_E ← B_E ∪ {(H_R, X)}
        end for
    end if
end if
```

همچنین تبدیل *bayes net* به*SPN* نیز امکان پذیر است: ابتدا باید *bayes net* را به یک درخت *junction* تبدیل کرد و از روی آن به *SPN* متناظر برسیم.بخاطر اندازه نمایی *SPN* تولید شده، اگر *bayes net* عرض زیادی داشته باشد، *SPN* تولید

شده مشکلاتی خواهد داشت(قدرت کم در نمایش *CPD*های محلی). همچنین میتوان یک *bayes net* با *ADDs* را به یک *AC* کامپایل کنیم و از روی آن *SPN* را بدست آوریم. برای اینکار ابتدا باید عملیات های معمول استفاده شده در *VE* را بدست آوریم.

دو عملیات معمول، اول ضرب دو *symbolic ADDs* و دومی *summing-out* یک متغیر پنهان است. لذا داریم:

Algorithm 6 Multiplication of two symbolic ADDs, $\otimes$

Input: Symbolic ADD $\mathcal{A}_{X_1}$, $\mathcal{A}_{X_2}$
Output: Symbolic ADD $\mathcal{A}_{X_1,X_2} = \mathcal{A}_{X_1} \otimes \mathcal{A}_{X_2}$

$R_1 \leftarrow$ root of $\mathcal{A}_{X_1}$, $R_2 \leftarrow$ root of $\mathcal{A}_{X_2}$
if R_1 and R_2 are both variable nodes **then**
  if $R_1 = R_2$ **then**
    Create a node $R = R_1$ into $\mathcal{A}_{X_1,X_2}$
    for each $r \in dom(R)$ **do**
      $\mathcal{A}^r_{X_1} \leftarrow Ch(R_1)|_r$
      $\mathcal{A}^r_{X_2} \leftarrow Ch(R_2)|_r$
      $\mathcal{A}^r_{X_1,X_2} \leftarrow \mathcal{A}^r_{X_1} \otimes \mathcal{A}^r_{X_2}$
      Link $\mathcal{A}^r_{X_1,X_2}$ to the rth child of R in $\mathcal{A}_{X_1,X_2}$
    end for
  else
    $\mathcal{A}_{X_1,X_2} \leftarrow$ create a symbolic node $\otimes$
    Link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as two children of $\otimes$
  end if
else if R_1 is a variable node and R_2 is $\otimes$ **then**
  if R_1 appears as a child of R_2 **then**
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2}$
    $\mathcal{A}^{R_1}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1} \otimes \mathcal{A}^{R_1}_{X_2}$
  else
    Link $\mathcal{A}_{X_1}$ as a new child of R_2
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2}$
  end if
else if R_1 is $\otimes$ and R_2 is a variable node **then**
  if R_2 appears as a child of R_1 **then**
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1}$
    $\mathcal{A}^{R_2}_{X_1,X_2} \leftarrow \mathcal{A}_{X_2} \otimes \mathcal{A}^{R_2}_{X_1}$
  else
    Link $\mathcal{A}_{X_2}$ as a new child of R_1
    $\mathcal{A}_{X_1,X_2} \leftarrow \mathcal{A}_{X_1}$
  end if
else
  $\mathcal{A}_{X_1,X_2} \leftarrow$ create a symbolic node $\otimes$
  Link $\mathcal{A}_{X_1}$ and $\mathcal{A}_{X_2}$ as two children of $\otimes$
end if
Merge connected product nodes in $\mathcal{A}_{X_1,X_2}$

Algorithm 7 Summing-out a hidden variable H from $\mathcal{A}$ using $\mathcal{A}_H$, $\oplus$

Input: Symbolic ADDs $\mathcal{A}$ and $\mathcal{A}_H$
Output: Symbolic ADD with H summed out

if H appears in $\mathcal{A}$ **then**
  Label each edge emanating from H with weights obtained from $\mathcal{A}_H$
  Replace H by a symbolic $\oplus$ node
end if

سپس از روی *BN* با الگوریتم زیر، *SPN* را بازیابی میکنیم:

Algorithm 8 Variable Elimination for BN with ADDs

Input: BN $\mathcal{B}$ with ADDs for all observable variables and hidden variables

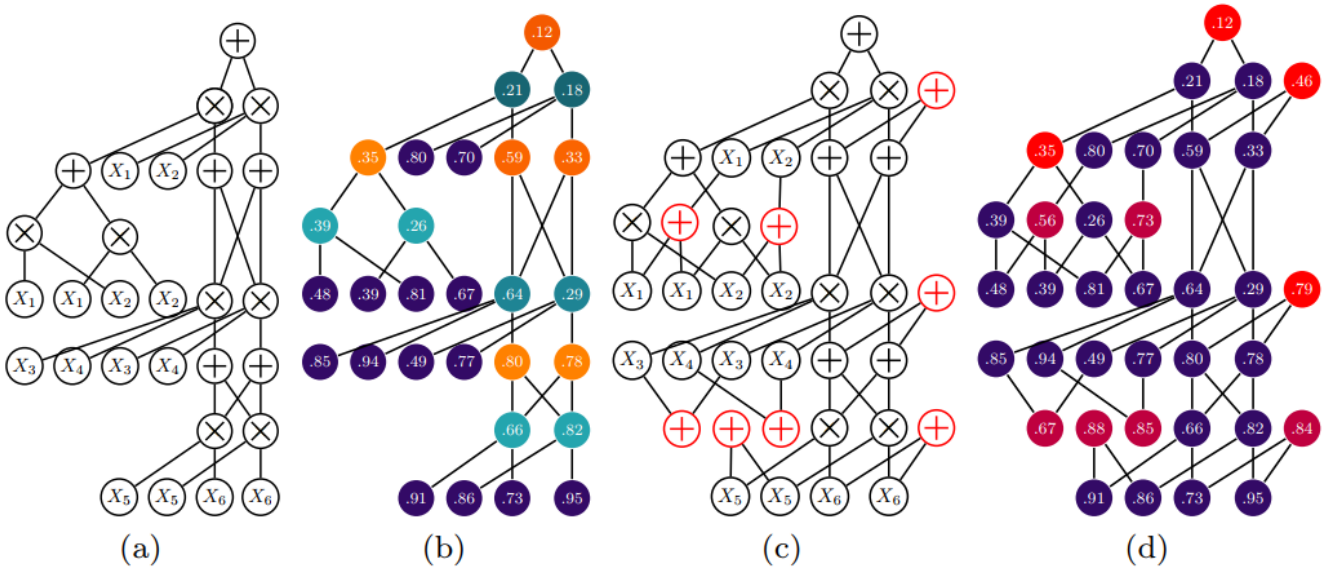
Output: Original SPN $\mathcal{S}$

1: $\pi \leftarrow$ the inverse topological ordering of all the hidden variables present in the ADDs
2: $\Phi \leftarrow \{\mathcal{A}_X \mid X \text{ is an observable variable}\}$
3: **for** each hidden variable H in π **do**
4: $\quad P \leftarrow \{\mathcal{A}_X \mid H \text{ appears in } \mathcal{A}_X\}$
5: $\quad \Phi \leftarrow \Phi \backslash P \cup \{\oplus_H \otimes_{\mathcal{A} \in P} \mathcal{A}\}$
6: **end for**
7: **return** Φ

منبع کمکی: مقاله تحت عنوان *on the relationship between sum-product networks and Bayesian networks*

قسمت د) دو مقاله راهکار هایی برای هندل کردن *SPN* های پیوسته پیدا کردم. در مقاله اول تحت عنوان *on the latent variable interpretation in sum-product networks* یک الگوریتم *EM* برای یادگیری پارامتر های اینچنین *SPN* هایی معرفی شده است. همچنین در مقاله *online algorithms for sum-product networks with continuous variables* الگوریتم های *online Bayesian moment matching* برای *SPN* های با توزیع *gaussian leaf* تعریف شده که میتواند این *SPN* هارا هندل کند. این الگوریتم ها و محاسبات ریاضی شان بسیار گسترده و خارج سرفصل بوده و به همین دلیل آن هارا در پاسخ ام نیاوردم.

قسمت ه) ابتدا یک نمونه Xi به SPN خود وارد میکنیم(a) و activation های گره های آن را جمع آوری میکنیم(b).سپس فیلتر های مختلفی را اعمال کرده و embedding تشکیل میدهیم برای مثال جمع آوری activation های تمام گره های داخلی(e) یا فیلتر کردن بر اساس نوع گره مثلا جمع آوری activation های گره های sum (f) یا گره های product (g) یا فیلتر کردن گره ها با معیار طول scope کوچک (h) یا متوسط (i) یا بزرگ(j) یا aggregate کردن گره های با scope برابر (k) (همان گره های sum قرمز رنگ در c و ارزیابی شده در d) و یا در آخر با استفاده از فقط گره های داخلی(l) :



در نهایت با این روش ها ***embedding*** ایجاد شده و به وسیله آن ها ، بازنمایی های مختلف نمایش داده میشود.

سوال ۲ –

قسمت الف) در ***Dual supervised learning*** ما با هر معماری مناسب برای یادگیری با نظارت، میتوانیم یادگیری را انجام دهیم.لذا معماری خاصی مدنظر نیست.در ***DSL*** ، اگر فرض کنیم پارامتر های حالت ***primal*** برابر θ_{XY} و پارامتر های حالت ***dual*** اش برابر θ_{YX} باشد، باید خطای آموزش آن هارا به حداقل برسانیم یعنی:

$$\min_{\theta_{XY}} -\frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \log P(y|x;\theta_{XY});$$

$$\min_{\theta_{YX}} -\frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \log P(x|y;\theta_{YX}).$$

متعاقبا، میتوانیم دو تابع پیش بینی زیر را برای فرآیند ***primal*** و ***dual*** ارائه دهیم:

$$f(x;\theta_{XY}) \triangleq \arg\max_{y' \in \mathcal{Y}} P(y'|x;\theta_{XY}),$$

$$g(y;\theta_{YX}) \triangleq \arg\max_{x' \in \mathcal{X}} P(x'|y;\theta_{YX}).$$

در *DSL* ، به صورت مشترک آموزش هم برای *primal* هم برای*dual* انجام میشود که بهینه سازی زیر نتیجه میشود:

$$\text{objective 1: } \min_{\theta_{XY}} \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \ell_1(f(x;\theta_{XY}), y),$$

$$\text{objective 2: } \min_{\theta_{YX}} \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \ell_2(g(y;\theta_{YX}), x),$$

$$\text{s.t. } P(x)P(y|x;\theta_{XY}) = P(y)P(x|y;\theta_{YX}), \forall (x,y) \in \mathcal{D},$$

اگر فرموله سازی لاگرانژ آن را بنویسیم و هر دو *objective function* ها را تلفیق کنیم، داریم:

$$\ell_{\text{dsl}} = (\log \hat{P}(x) + \log P(y|x;\theta_{XY}) - \log \hat{P}(y) - \log P(x|y;\theta_{YX}))^2.$$

الگوریتم کلی *DSL* برابر است با:

Algorithm 1 Dual supervise learning algorithm

Require: : Marginal distributions $\hat{P}(x)$ and $\hat{P}(y)$; Lagrange parameters λ_{XY} and λ_{YX}; optimizers Opt_1 and Opt_2;

repeat

Sample a minibatch of m pairs $\{(x_j, y_j)\}_{j=1}^m$;

Calculate the gradients as follows:

$$G_f = \nabla_{\theta_{XY}} (1/m) \sum_{j=1}^m \left[\ell_1(f(x_j;\theta_{XY}), y_j) + \lambda_{XY}\ell_{\text{dsl}}(x_j, y_j;\theta_{XY},\theta_{YX})\right];$$

$$G_g = \nabla_{\theta_{YX}} (1/m) \sum_{j=1}^m \left[\ell_2(g(y_j;\theta_{YX}), x_j) + \lambda_{YX}\ell_{\text{dsl}}(x_j, y_j;\theta_{XY},\theta_{YX})\right];$$

Update the parameters of f and g:

$$\theta_{XY} \leftarrow Opt_1(\theta_{XY}, G_f), \theta_{YX} \leftarrow Opt_2(\theta_{YX}, G_g).$$

until models converge

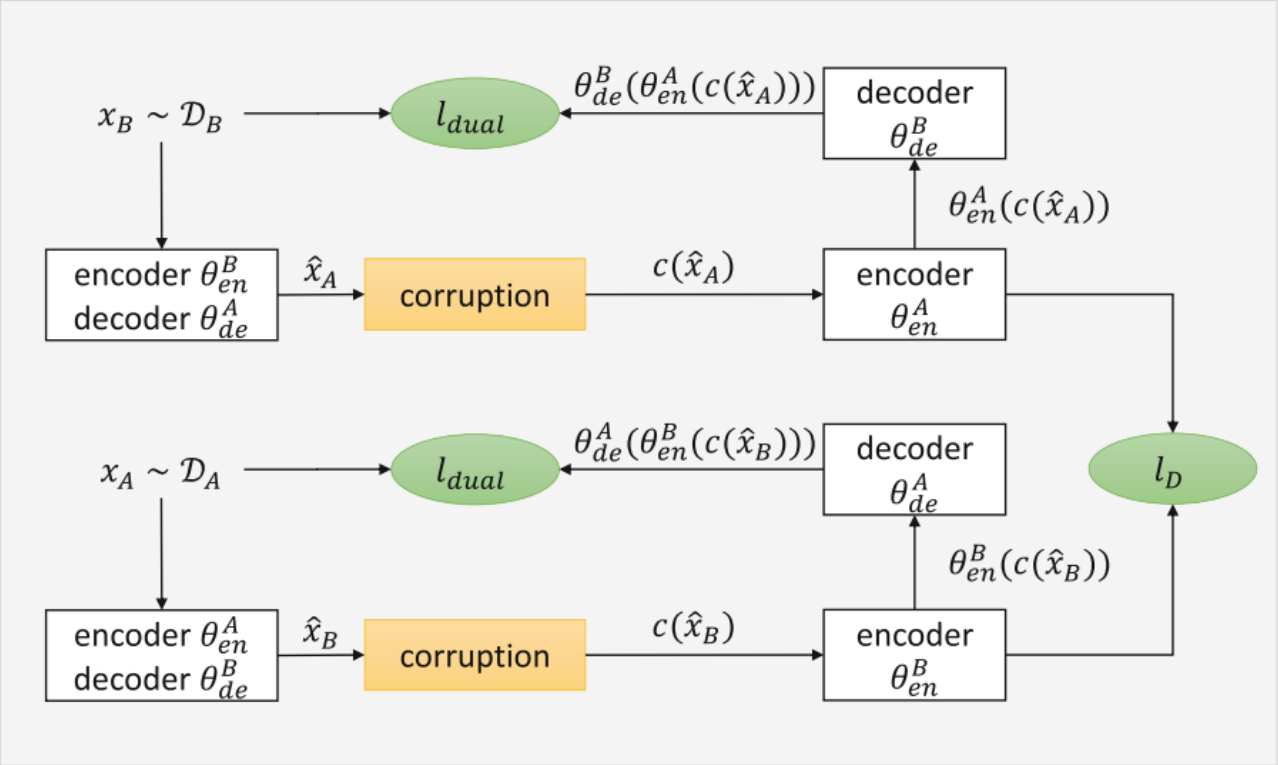
در *Dual unsupervised learning* معماری *encoder-decoder* لازم است زیرا برچسب داده ها یا جمله های دوزبانه وجود ندارد و باید برای مثال از زبان مبدا، جمله . برای مثال در کاربرد ترجمه ماشینی، این نوع یادگیری بسیار پرکاربرد است.معماری آن به این صورت است که نسخه نویزی جمله مبدا توسط *encoder* به یک *embedding* ثابت تبدیل شده و توسط *decoder* به جمله مقصد تبدیل میشود.لذا تابع هزینه برابر است با :

$$\ell_{dae}(\theta^l_{en}, \theta^l_{de}) = \frac{1}{|\mathcal{M}_l|} \sum_{x \in \mathcal{M}_l} \Delta(x, \theta^l_{de}(\theta^l_{en}(c(x)))),$$

لذا در فرآیند *dual* آن تابع هزینه میشود:

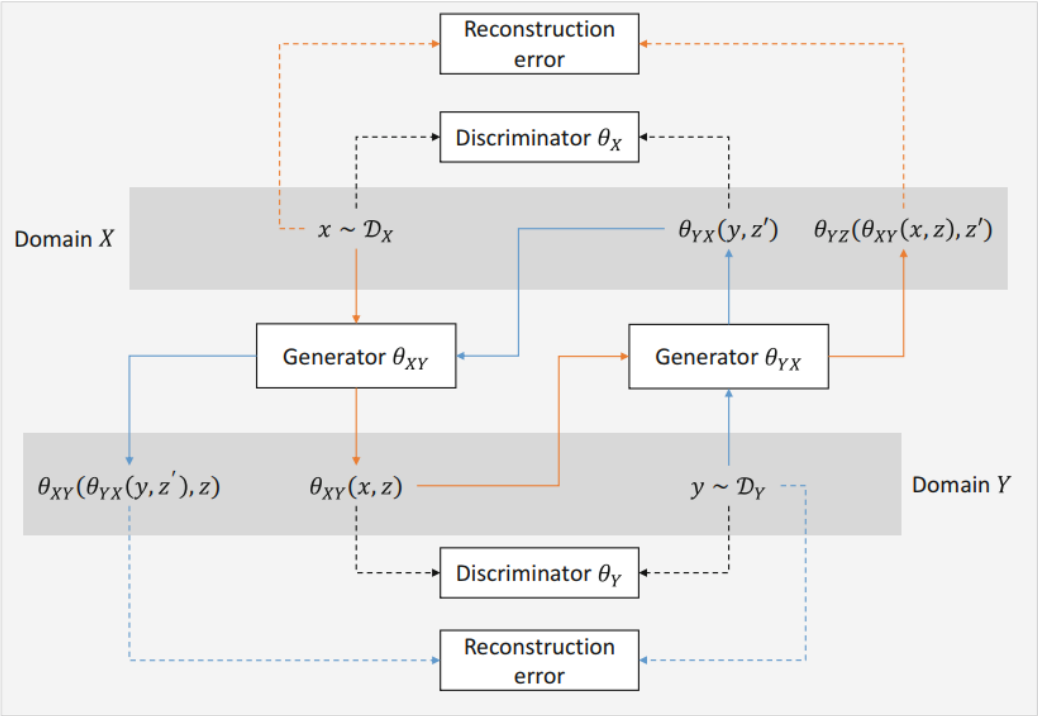
$$\ell_{dual}(\theta^l_{en}, \theta^l_{de}) = \frac{1}{|\mathcal{M}_A|} \sum_{x \in \mathcal{M}_A} \Delta(x, \theta^A_{de}(\theta^B_{en}(c(\hat{y})))) + \frac{1}{|\mathcal{M}_B|} \sum_{x \in \mathcal{M}_B} \Delta(x, \theta^B_{de}(\theta^A_{en}(c(\hat{y})))),$$

معماری کلی *dual reconstruction* برابر است با:



قسمت ب) *DualGan* نوعی استفاده از *Dual reconstruction* و *Gan* است.کاربرد آن *img2img translation* است.

معماری آن برابر است با:



هر دو مدل ترجمه تصویری(*primal,dual*) برای کمینه کردن خطای بازسازی تصویر اوریجینال، آموزش داده میشوند.*DualGan* از *L1 distance* استفاده میکند زیرا *L2 distance* باعث تار شدن تصویر بازسازی شده میشود. تابع هزینه آن برابر است با:

$$\begin{aligned}\ell(x, y; \theta_{YX}, \theta_{XY}) &= \lambda_X \|x - \theta_{YX}(\theta_{XY}(x, z), z')\| \\ &+\lambda_Y \|y - \theta_{XY}(\theta_{YX}(y, z'), z)\| \\ &-\theta_Y(\theta_{XY}(x, z)) - \theta_X(\theta_{YX}(y, z')).\end{aligned}$$

و همچنین الگوریتم آموزش این شبکه برابر است با:

Algorithm 1 DualGAN training procedure

Require: Two image collections $\mathcal{D}_X$ and $\mathcal{D}_Y$, clipping parameter c, batch size m, and d

Randomly initialize θ_{YX}, θ_{XY}, θ_X, and θ_Y
repeat
  for $t = 1, \ldots, d$ **do**
    Sample images $\{x^{(k)}\}_{k=1}^m \subseteq \mathcal{D}_X$, $\{y^{(k)}\}_{k=1}^m \subseteq \mathcal{D}_Y$
    Update θ_X to minimize $\frac{1}{m}\sum_{k=1}^m \ell(x^{(k)}, y^{(k)}; \theta_X)$
    Update θ_Y to minimize $\frac{1}{m}\sum_{k=1}^m \ell(x^{(k)}, y^{(k)}; \theta_Y)$
    $clip(\theta_X, -c, c)$, $clip(\theta_Y, -c, c)$
  end for
  Sample images $\{x^{(k)}\}_{k=1}^m \subseteq \mathcal{D}_X$, $\{y^{(k)}\}_{k=1}^m \subseteq \mathcal{D}_Y$
  Update θ_{YX} and θ_{XY} to minimize $\frac{1}{m}\sum_{k=1}^m \ell(x^{(k)}, y^{(k)}; \theta_{YX}, \theta_{XY})$
until convergence

منبع کمکی هر دو قسمت: کتاب *Dual learning* نوشته *Tao Qin*

سوال ۳ –

قسمت الف) در *Q-learning* ، *updated policy* با *behavior policy* متفاوت است. لذا میتوانیم برای مثال ابتدا اپیزود هایی از محیط را اجرا کرده و مدلی بدست آوردیم و سپس در آینده این مدل را استفاده کنیم(طبق گفته های کلاس دکتر رهبان درس یادگیری ماشین). از طرفی در *Q-learning* ، برای عمل های آینده ما یک پاداش تخمین میزنیم و مقدار جدیدی به حالت های جدید میدهیم بدون اینکه از *greedy policy* استفاده کرده باشیم. زیرا به این دلایل، *off policy* است زیرا *off policy* یعنی یادگیرنده از این نوع، مقادیر *optimal policy* را به صورت مستقل از عمل های عامل یاد میگیرد.

یکی از مزایای *off policy* این است که *updated policy* و *behavior policy* میتوانند یکی *deterministic* باشد و دیگری نباشد. یعنی مستقل بودن و جدا بودن این دو این کمک را به ما میکند.از طرف دیگر اگر عامل در محیطی *ofline*

بخواهد یادگیری انجام دهد و *explore* زیادی نداشته باشد، *off policy* بهتر عمل میکند.در دنیای واقعی هم از لحاظ هزینه، به صرفه تر است.

قسمت ب) در Q-learning ما Q-table داریم اما در DQN ، به جای آن، یک شبکه عصبی عمیق داریم.در Q-learning ما علاقه داریم Q-function را بیابیم اما .از طرفی با بزرگ شدن اعمال و فضای مسئله، سایز Q-table بزرگ شده و آموزش با Q-learning تقریبا غیر ممکن میشود که در این موقعیت، DQN بخاطر اینکه با شبکه عصبی کار میکند، مشکل را حل میکند. از طرفی Q-learning با حالت های پیوسته کار نمیکند اما DQN مشکلی ندارد.اما Q-learning همیشه همگرا میشود اما DQN ممکن است همگرا نشود.از طرفی Q-learning نوعی الگوریتم online است اما DQN اینطور نیست.

قسمت ج)

استفاده از experience replay buffer : آموزش روی داده های ترتیبی مرتبط(correlated) میتواند منجر به ناپایداری مدل شود. لذا از این بافر برای ذخیره نمونه برداری(transition tuple) ها استفاده میشود و سپس به صورت رندوم از این بافر نمونه برداری میکنیم به صورت minibatch و اینگونه ناپایداری کاهش میابد.

استفاده از تکنیک freeze the target network : در Q-learning ما با استفاده از یک حدس، حدس بعدی را میزنیم. لذا اینکار باعث correlation مضری میتواند بشود و شبکه عصبی نمیتواند بین دو چیز خیلی شبیه به هم تمایز قائل شود.وقتی ما میخواهیم Q value را برای یک حالت با آپدیت کردن پارامتر های شبکه، بهینه کنیم، ناخودآگاه به مقادیر حالت های دیگر و نزدیک به آن هم دست میزنیم و این ناپایداری ایجاد میکند.برای حل این مشکل یک کپی از کل شبکه عصبی گرفته و آن را target network مینامیم. از Q value پیش بینی شده توسط target network، برای backpropagation و یادگیری شبکه اصلی و اولیه، استفاده میشود.توجه شود پارامتر های target network آموزش داده نمیشوند یعنی freeze هستند اما به صورت متناوب و هر چند مدت یک بار، با شبکه اصلی sync میشوند.

قسمت د) در DDQN ، از دو مدل شبکه عصبی یکسان استفاده میکند.اولی هنگام experience replay یادگیری انجام میدهد و دیگری یک کپی از آخرین اپیزود اولین مدل است که Q value با این شبکه محاسبه میشود. در DQN ، Q value محاسبه شده برابر است با بیشترین Q value حالت بعدی به اضافه پاداش کسب شده. لذا اگر هر دفعه Q value برای یک حالت خاص مقدار زیادی باشد، مقداری که خروجی شبکه عصبی به ازای آن حالت خاص به ما میدهد، بزرگ و بزرگ تر میشود. حال اگر برای یک حالتی، مقدار عمل a از عمل b بیشتر باشد، پس همیشه a انتخاب میشود و اگر برای بعضی از memory experience ها b بهتر از a باشد، شبکه نمیتواند b را انتخاب کند زیرا اختلاف بین مقادیر خروجی زیاد است. لذا باید این

اختلاف را کم کنیم.در DDQN در شبکه دوم، اختلاف خروجی ها پایین است و به همین دلیل از این شبکه برای بدست آوردن Q value استفاده میکنیم. لذا در DQN ما تخمین بالاتر پاداش واقعی را داریم که در DDQN با جدا کردن action selection و action evaluation ، این مشکل را حل میکنیم.

تفاوت شبکه دوم در DDQN با target network این است که target network کپی کامل از شبکه اصلی است و برای محاسبه Q value از آن استفاده نمیشود به صورت مستقیم اما در DDQN شبکه دوم، کپی آخرین اپیزود از شبکه اول است و برای محاسبهQ value استفاده میشود.

سوال ۴ – در نوتبوک آورده شده است.

سوال ۵ –

قسمت الف) الگوریتم DDPG ترکیبی از الگوریتم های DPG و DQN است. به این صورت که از دو شبکه target استفاده میکند که با یکی یک سری target را تخمین زده و با دیگری از target های تخمین زده شده، یادگیری را انجام میدهیم. در این الگوریتم از experience replay و slow-learning ای که در DQN بود نیز استفاده میشود. برای آپدیت شبکه عامل، از sampled policy gradient استفاده میشود به این صورت:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

قسمت ب) برای حل مشکل exploration از یک فرآیند نویز به صورت correlated (بخاطر explore خوب در محیط فیزیکی) استفاده میشود. برای مثال میتوان از فرآیند Ornstein-uhlenbeck استفاده کرد.این فرآیند velocity را در حرکت براونی مدل میکند. از طرف دیگر الگوریتم ما off-policy است یعنی بحث exploration از الگوریتم یادگیری مستقل است.لذا یک exploration policy به شکل زیر تعریف میکنیم که حاصل جمع actor policy و همان نویز گفته شده است:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

سوال ۶ – در نوتبوک آورده شده است.