1/7/2025

یادگیری ماشین

فاز اول پروژه

Ali Sadeghian 400101464
Amirreza Tanevardi 400100898
SUT

بخش تئوری

پرسش 1

سعی کنید درباره مفهوم توابع پتانسیل در یک میدان مارکف کمی توضیح بدهید و یک مثال از یک میدان مارکف به همراه توابع پتانسیل اش بزنید. توجه کنید مثال میتواند بسیار ساده باشد اما حتما باید مقادیر عددی تعیین گردد و توضیحات هم تا حد ممکن کوتاه و هدفمند باشند.

مفهوم توابع پتانسیل در میدان مارکوف:
توابع پتانسیل توابع غیرمنفی مرتبط با گروه‌های خاص (cliques) در گراف هستند که ساختار وابستگی بین متغیرها را مشخص کرده و برای تعریف توزیع احتمالی کلی میدان مارکوف به کار می‌روند.

مثال:
فرض کنید یک میدان مارکوف با 3 گره A, B, C و یال‌های $A - B, B - C$ توزیع احتمال کلی:

$$P(A, B, C) = \frac{1}{Z}\psi_{AB}(A, B)\psi_{BC}(B, C)$$

برای $A, B, C \in \{0,1\}$ داریم:

توابع پتانسیل:

$$\psi_{AB}(A, B) = e^{A.B}, \psi_{BC}(B, C) = e^{BC}$$

ثابت نرمال‌سازی:

$$Z = 6 + 2e + e^2$$

پرسش 2

3 نوع از توابع پتانسیل را نام برده و پارامتر های آنها را مشخص کنید.

توابع پتانسیل در میدان مارکوف برای بیان تأثیر روابط بین گره‌ها در گراف استفاده می‌شوند. در اینجا سه نوع اصلی از توابع پتانسیل را توضیح می‌دهیم:

1. **پتانسیل یال:(Edge Potential)**
   این تابع، تعامل بین دو گره متصل i و j را نشان می‌دهد. برای مثال، اگر گره‌ها مقادیری مانند X_i و X_j داشته باشند، این تابع می‌تواند به شکل $\psi_{ij}(X_i, X_j) = e^{\theta_{ij} X_i X_j}$ باشد.
   - **پارامتر کلیدی:** $\boldsymbol{\theta_{ij}}$ که وزنی است برای بیان شدت رابطه بین دو گره.
2. **پتانسیل گره:(Node Potential)**
   این تابع اثر یک گره منفرد روی توزیع کلی را مدل می‌کند. برای یک گره i، می‌توان این تابع را به صورت $\psi_i(X_i) = e^{\theta_i X_i}$ نوشت.

- **پارامتر کلیدی:** $\boldsymbol{\theta_i}$ که نشان‌دهنده میزان اهمیت یا بایاس گره است.

3. **پتانسیل گروه خاص:(Clique Potential)**

این نوع تابع، تعاملات چندین گره که یک گروه خاص را تشکیل می‌دهند، توصیف می‌کند. برای گروهی از گره‌ها C، تابع به صورت $\psi_C(X_C) = \sum_{i \in C} e^{\theta_i X_i}$ تعریف می‌شود.

- **پارامترهای کلیدی**: $\boldsymbol{\theta_i}$ برای هر گره در گروه، که نقش آن‌ها در تعامل کلی را مشخص می‌کند.

این توابع در کنار هم به میدان مارکوف کمک می‌کنند تا روابط پیچیده بین متغیرها را در قالب یک مدل ساده‌تر نمایش دهد.

پرسش 3

به علت اینکه پیدا کردن آنالیزی پارامترها برای بیشینه سازی عبارت بالا به طور عمومی ممکن نیست در خیلی از مواقع از روش Gradient Ascent استفاده میشود . در کمتر از سه خط این روش را توضیح بدهید.

این روش برای بیشینه‌سازی یک تابع (مانند لگاریتم درست‌نمایی) استفاده می‌شود. ایده اصلی آن حرکت در جهت شیب مثبت تابع است:

1. **محاسبه گرادیان:** مشتق جزئی تابع هدف $L(\theta)$ نسبت به پارامترها.
2. **به‌روزرسانی پارامترها:**

$$\theta = \theta + \eta \nabla_\theta L(\theta)$$

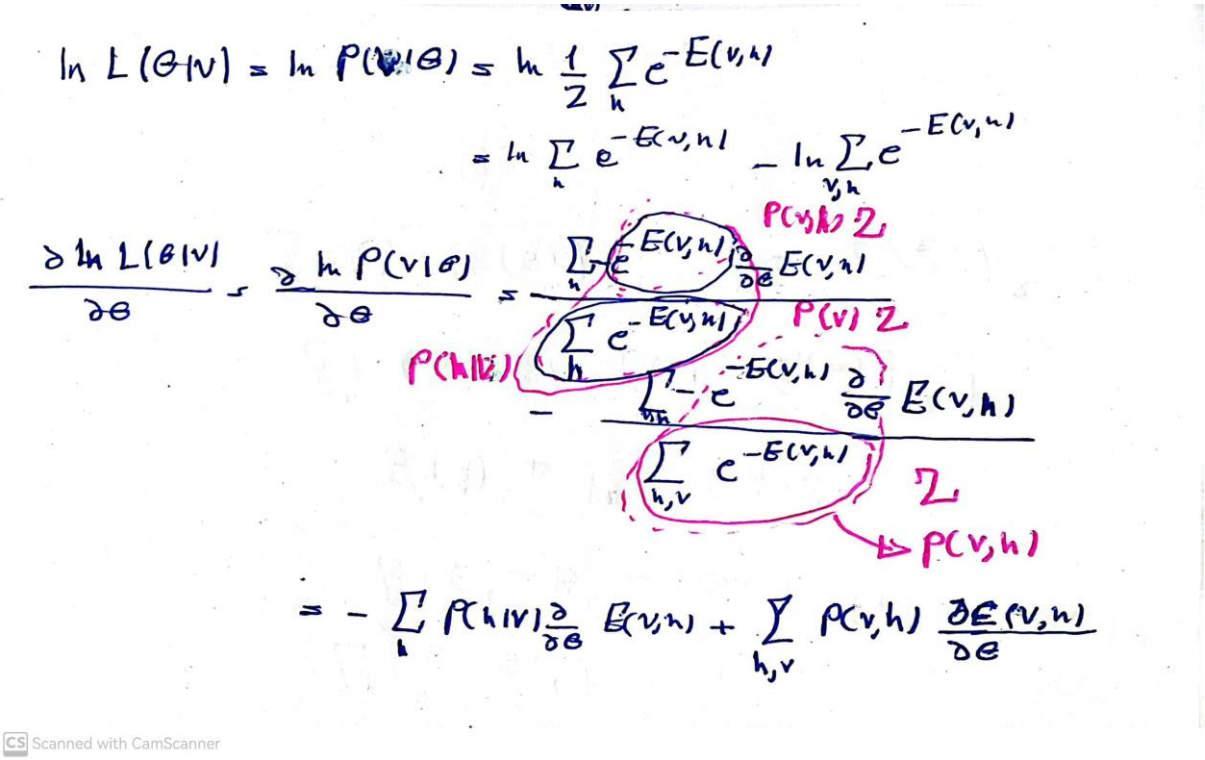
که در آن η نرخ یادگیری است.

3. **تکرار :** این فرآیند تا زمانی ادامه می‌یابد که مقدار تابع همگرا شود.

پرسش 4

نشان دهید:

$$\frac{\partial \ln(L(\theta \mid v)}{\partial \theta} = -\sum_{v} p(h|v) \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} p(v,h) \frac{\partial E(v,h)}{\partial \theta}$$



پرسش 5

با توجه به توضیحات داده شده در مورد نحوه عملکرد ماشین بولتزمن، این ساختار در چه بخش هایی از یادگیری ماشین می تواند استفاده شود؟

۱ .سیستم‌های توصیه‌گر:(Recommendation Systems)

ماشین بولتزمن محدود (RBM) در سیستم‌های توصیه‌گر برای پیشنهاد محصولات یا خدمات به کاربران استفاده می‌شود.

جزئیات:

- در سیستم‌های توصیه‌گر، ماتریسی از تعاملات کاربران و آیتم‌ها داریم که برخی ورودی‌ها خالی هستند (مثلاً یک کاربر به برخی فیلم‌ها امتیاز نداده است).
- RBMبا یادگیری روابط پنهان میان کاربران و آیتم‌ها می‌تواند این مقادیر گم‌شده را پیش‌بینی کند.
- در اینجا، نورون‌های لایه نمایان (Visible) نشان‌دهنده کاربران یا آیتم‌ها هستند و نورون‌های لایه پنهان (Hidden)وابستگی‌های غیرمستقیم میان آن‌ها را کشف می‌کنند.
- **مثال عملی :**در سیستم توصیه فیلم، مدل می‌تواند با استفاده از داده‌های قبلی، امتیاز یک کاربر به فیلم‌های ندیده را پیش‌بینی کرده و پیشنهاد دهد.

۲ .پیش‌آموزش شبکه‌های عمیق:(Pretraining in Deep Learning)

RBMبه عنوان یکی از اجزای اصلی در شبکه‌های باور عمیق (Deep Belief Networks - DBN) به کار می‌رود.

جزئیات:

- در یادگیری عمیق، RBMها می‌توانند به صورت لایه‌به‌لایه برای پیش‌آموزش وزن‌ها استفاده شوند.
- ابتدا هر RBM وزن‌های یک لایه را یاد می‌گیرد و سپس لایه بعدی روی خروجی لایه قبلی آموزش داده می‌شود.
- این پیش‌آموزش باعث می‌شود مدل به یک نقطه بهینه نزدیک‌تر شود و بهبود عملکرد در مسائل پیچیده مانند دسته‌بندی تصاویر یا پردازش زبان طبیعی را تضمین کند.
- **مثال عملی :**در دسته‌بندی تصاویر مجموعه دادهMNIST ، از DBN که از چندین RBM تشکیل شده برای پیش‌آموزش وزن‌ها و بهبود دقت دسته‌بندی استفاده می‌شود.

۳ .بازسازی داده‌های گم‌شده:(Data Imputation)

RBMمی‌تواند مقادیر گم‌شده در داده‌ها را با استفاده از وابستگی‌های یادگرفته‌شده بین متغیرها تخمین بزند.

جزئیات:

- در مجموعه داده‌های واقعی، اغلب برخی مقادیر در ستون‌های مختلف ناقص هستند.
- RBMبا مدل‌سازی احتمال مشترک بین متغیرها، مقادیر از دست رفته را پیش‌بینی می‌کند.
- فرآیند یادگیری شامل به‌روزرسانی وزن‌ها با استفاده از داده‌های کامل و سپس تخمین مقادیر ناقص است.
- **مثال عملی :**در پایگاه داده بیماران، اگر اطلاعاتی مانند قد، وزن یا فشار خون برخی بیماران ناقص باشد، RBM می‌تواند این مقادیر را تخمین زده و پایگاه داده را کامل کند.

پرسش 6

ثابت کنید:

$$P(H_i = 1|v) = \sigma(\sum_{j=1}^{m} \omega_{ij} v_j + c_i)$$

$$P(V_i = 1|vh) = \sigma(\sum_{i=1}^{n} \omega_{ij} h_i + b_j)$$

$$P(H_i=1|v) = \sigma\left(\sum_{j=1}^{m} w_{ij} v_j + c_i\right)$$

$$P(v_j=1|h) = \sigma\left(\sum_{i=1}^{n} w_{ij} h_i + b_j\right)$$

$$P(H_i=1|v) = \sum_{h'} \frac{P(H_i=1, H', v)}{P(v)}, \quad H' = \{h_k\}_{i \neq k}$$

$$= \frac{\sum_{h'} P(h_i=1, h', v)}{\sum_{h} P(v,h)} = \frac{\sum_{h'} P(v, h_i=1, h')}{\sum_{h'} P(v, h_i=1, h') + P(v, h_i=0, h')}$$

$$= \frac{\sum_{h'} e^{\sum_{k \neq i}\sum_{j} w_{kj} h_k v_j + \sum b_j v_j + \sum_{k \neq i} c_k h_k + \sum_j w_{ij} v_j + c_i}}{\sum_{h'} e^{\sum_{i \neq k}\sum_j w_{kj} h_k v_j + \sum b_j v_j + \sum_{k \neq i} c_k h_k} + e^{\sum_{i \neq k}\sum_j w_{kj} v_j h_k + \sum b_j v_j + \sum_{k \neq i} c_k h_k + \sum_j w_{ij} v_j + c_i}}$$

$$= \frac{\sum_{h'} \alpha(h')\, e^{\sum_j w_{ij} v_j + c_i}}{\sum_{h'} \alpha(h') + \sum \alpha(h')\, e^{\sum_j w_{ij} v_j + c_i}} \times \frac{1}{\sum_{h'} \alpha(h')} = \frac{e^{\beta}}{1+e^{\beta}} = \frac{1}{1+e^{-\beta}}$$

$$= \sigma(+\beta) = \sigma\left(\sum w_{ij} v_j + c_i\right)$$

$$P(v_j=1|h) = \sum_{v'} \frac{P(v_j=1, v', h)}{\sum_{v'} P(v_j=1, v', h) + P(v_j=1, v', h)}, \quad v' = \{v_L\}_{L \neq j}$$

(همانند قبل داریم)

$$= \frac{\sum_{v'} e^{\sum_i \sum_{L \neq j} w_{iL} v_L h_i + \sum_{L \neq j} b_L v_L + \sum_i c_i h_i + \sum_i w_{ij} h_i + b_j}}{\sum_{v'} \alpha(v') + \sum_{v'} \alpha(v')\, e^{\sum_i w_{ij} h_i + b_j}}$$

$$= \sigma\left(\sum_{i}^{m} w_{ij} h_i + b_j\right)$$

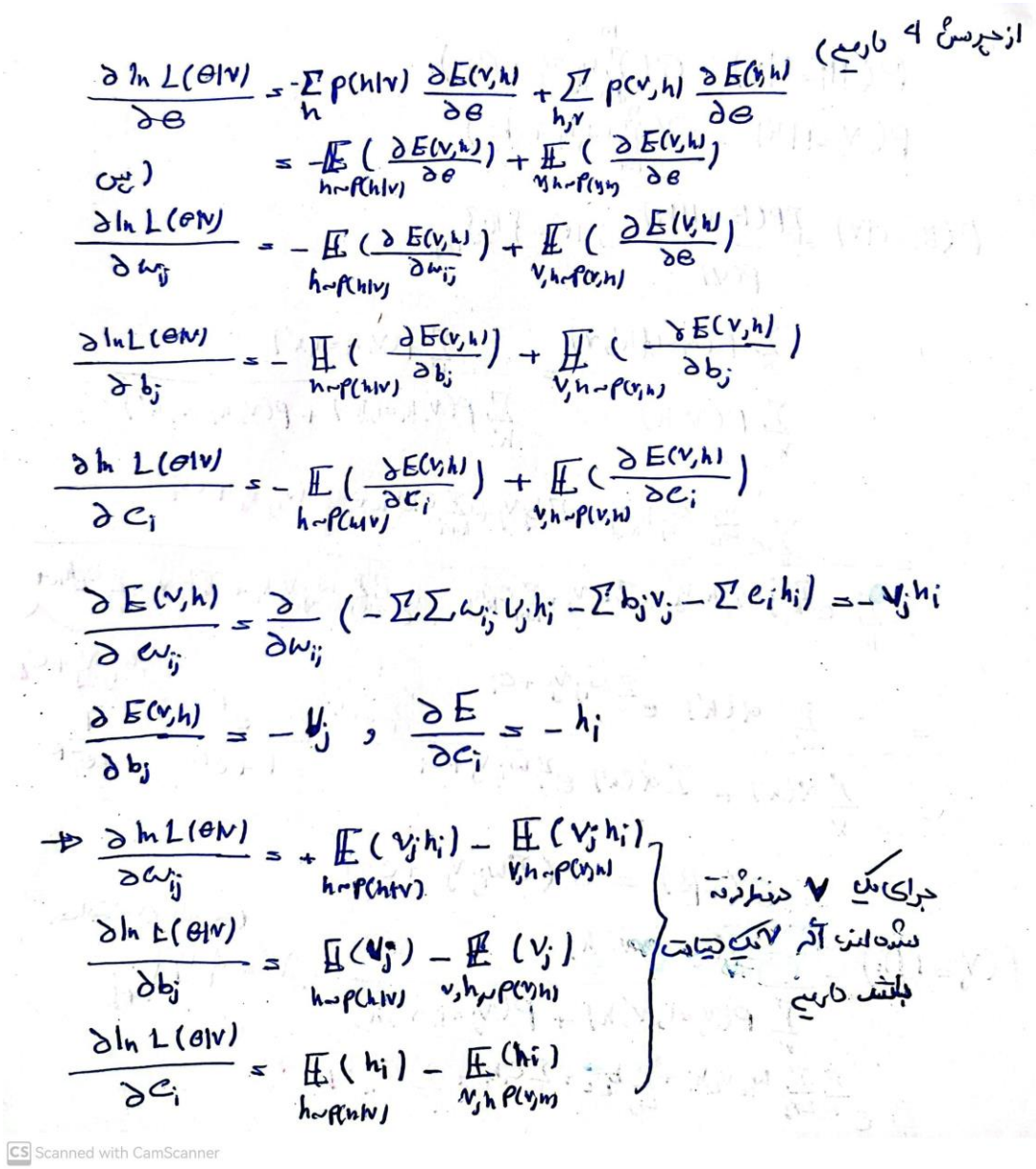
پرسش 7,8

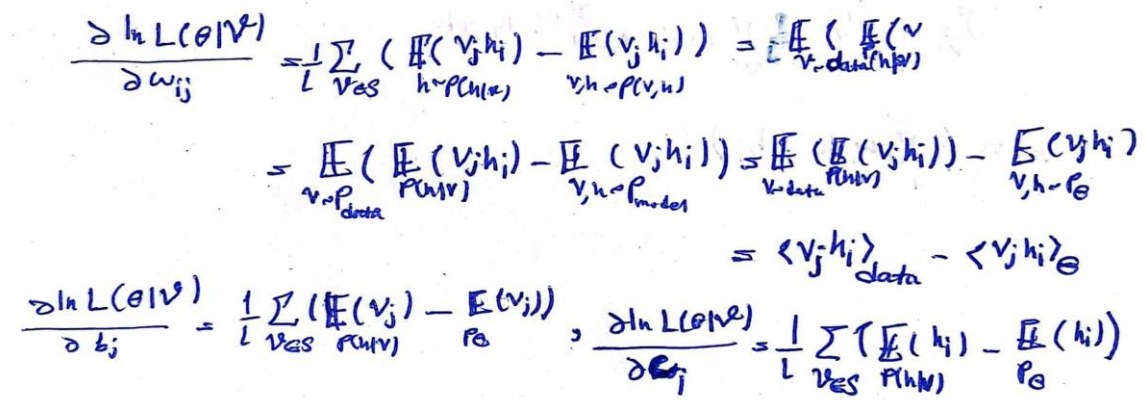
برای به روز رسانی پارامتر ها، نیاز داریم مشتق های جزئی تابع هدف را نسبت به پرامتر ها حساب کنیم .مشتق های جزئی زیر را بدست آورده و تا حد امکان آنها را ساده کنید.

$$\frac{\partial \ln L(\theta|v)}{\partial \omega_{ij}}, \frac{\partial \ln L(\theta|v)}{\partial b_j}, \frac{\partial \ln L(\theta|v)}{\partial c_i}$$

نشان دهید پاسخ شما به صورت زیر میتواند نوشته شود:

$$\frac{\partial \ln L(\theta|v)}{\partial \omega_{ij}} = \frac{1}{l}\sum_{v\in S}\left[\mathrm{E}_{p(h|v)}(v_j h_i) - \mathrm{E}_{p(v,h)}(v_j h_i)\right] = < v_j h_i >_{data} - < v_j h_i >_{model}$$





پرسش 9

نشان دهید که بهینه کردن $CD_k(\theta, v_0)$ معادل با بهینه‌سازی $D_{KL}(p_{data}||p_{theta}) - D_{KL}(p^k_{theta} - p_{theta})$ است.

نشان دهید که با افزایش k مقدار بهینه CD معادل با تخمینگر بیشینه درست نمایی است.

$$\nabla \ln P_\theta(v') = -\sum_h P(h|v)\frac{\partial E(v,h)}{\partial \theta} + \sum_{v',h'} \frac{P(v',h')\,\partial E(v',h')}{\partial \theta}$$ (از جزوه 4 داریم)

$$\nabla_\theta \left(D(P_{data} \| P_\theta) - D(P_\theta^t \| P_\theta) \right)$$

$$\nabla_\theta D(P_{data}\|P_\theta) = \mathop{\mathbb{E}}_{v\sim P_{data}}\left(\nabla_\theta \log \frac{P_{data}}{P_\theta}\right) = -\mathop{\mathbb{E}}_{v\sim P_{data}}\left(\nabla_\theta \log P_\theta(v)\right)$$

$$= \mathop{\mathbb{E}}_{v\sim P_{data}}\left(\sum_h P(h|v)\frac{\partial E(v,h)}{\partial\theta} - \sum_{v',h'} P(v',h')\frac{\partial E(v',h')}{\partial\theta}\right)$$

$$= \mathop{\mathbb{E}}_{v\sim P_{data}}\left(\sum_h P(h|v)\frac{\partial E(v,h)}{\partial\theta}\right) - \sum_{v',h'} P(v',h')\frac{\partial E(v',h')}{\partial\theta}$$

(همینطور داریم:

$$\nabla_\theta D(P_\theta^t\|P_\theta) = \nabla_\theta \int \overbrace{P_\theta^t(v)\log\frac{P_\theta^t(v)}{P_\theta(v)}}^{A(P_\theta^t,P_\theta)}\,dv$$

$$= \int \frac{\partial A(P_\theta^t,P_\theta)}{\partial P_\theta^t}\frac{\partial P_\theta^t}{\partial\theta}\,dv + \int \frac{\partial A(P_\theta^t,P_\theta)}{\partial P_\theta}\frac{\partial P_\theta}{\partial\theta}\,dv$$

$\left(\frac{\partial}{\partial x}\,a\log\frac{a}{x} = -\frac{a}{x}\right)$

$$= \int \frac{\partial A(P_\theta^t,P_\theta)}{\partial P_\theta^t}\frac{\partial P_\theta^t}{\partial\theta}\,dv - \int \frac{P_\theta^t(v)}{P_\theta(v)}\nabla_\theta P_\theta(v)\,dv$$

$$= \int \text{—} - \int P_\theta^t(v)\,\nabla\log P_\theta(v)\,dv$$

$$= \int \text{—} + \mathop{\mathbb{E}}_{v\sim P_\theta^t}\left(\sum_h P(h|v)\frac{\partial E(v,h)}{\partial\theta}\right) - \sum_{v',h'} P(v',h')\frac{\partial E(v',h')}{\partial\theta}$$

$$\Rightarrow \nabla_\theta\left(D(P_{data}\|P_\theta) - D(P_\theta^t\|P_\theta)\right) = \mathop{\mathbb{E}}_{v\sim P_{data}}\left(\sum_h (P(h|v))\right) - \sum_{v',h'} P(v',h')\frac{\partial E(v',h')}{\partial\theta}$$

معمولا صرف نظر می شود

$$- \int \text{—} - \mathop{\mathbb{E}}_{v\sim P_\theta^t}\left(\sum_h (P(h|v))\right) + \sum_{v',h'} P(v',h')\frac{\partial E(v',h')}{\partial\theta}$$

$$= \mathop{\mathbb{E}}_{v\sim P_{data}}\left(\sum_h (P(h|v))\right) - \mathop{\mathbb{E}}_{v\sim P_\theta^t}\left(\sum_h P(h|v)\right)$$

اگر k به سمت بی‌نهایت میل کند، به هم، P_θ^k با P_θ برابر خواهند شد و $D(P_\theta^k \| P_\theta)$ برابر 0 خواهد شد.

همچنین مینیمم کردن $D(P_{data}\|P_\theta)$ با مینیمم کردن منفی [illegible] (برابر است)

$$\nabla_\theta D(P_{data}\|P_\theta) = \nabla_\theta \mathop{\mathbb{E}}_{P_{data}}\left(\log\frac{P_{data}}{P_\theta}\right) = -\mathbb{E}(\nabla_\theta \log P_\theta)$$

پرسش 10

یافته های قسمت های قبل خود را، با الگوریتم divergence contrastive step-k توجیح کنید، در مورد پیچیدگی این الگوریتم، و انتظار شما از خروجی مدل بعد از اعمال یادگیری با استفاده از این الگوریتم بحث کنید.

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, training batch S
Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

1 init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, m$
2 **forall the** $\boldsymbol{v} \in S$ **do**
3 $\boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$
4 **for** $t = 0, \ldots, k-1$ **do**
5 **for** $i = 1, \ldots, n$ **do** sample $h_i^{(t)} \sim p(h_i \mid \boldsymbol{v}^{(t)})$
6 **for** $j = 1, \ldots, m$ **do** sample $v_j^{(t+1)} \sim p(v_j \mid \boldsymbol{h}^{(t)})$
7 **for** $i = 1, \ldots, n$, $j = 1, \ldots, m$ **do**
8 $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \mid \boldsymbol{v}^{(0)}) \cdot v_j^{(0)} - p(H_i = 1 \mid \boldsymbol{v}^{(k)}) \cdot v_j^{(k)}$
9 $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$
10 $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \mid \boldsymbol{v}^{(0)}) - p(H_i = 1 \mid \boldsymbol{v}^{(k)})$

همانطور که مشاهده می کنیم درحلقه اول سعی داریم از مدل نمونه برداری کنیم(p_{theta}^{t}) در حلقه دوم با استفاده از رابطه قسمت 9 و قسمت 7، وزن‌ها را بروز می کنیم. اگر از بخش 7 بیاد بیاوریم:

$$\frac{\partial \ln L(\theta|V)}{\partial w_{ij}} = + \mathbb{E}_{h \sim p(h|v)}(v_j h_i) - \mathbb{E}_{v,h \sim p(v,h)}(v_j h_i)$$

$$\frac{\partial \ln L(\theta|V)}{\partial b_j} = \mathbb{E}_{h \sim p(h|v)}(v_j) - \mathbb{E}_{v,h \sim p(v,h)}(v_j)$$

$$\frac{\partial \ln L(\theta|V)}{\partial c_i} = \mathbb{E}_{h \sim p(h|v)}(h_i) - \mathbb{E}_{v,h \sim p(v,h)}(h_i)$$

h_i چون بین 0 و 1 هستند، احتمال آنکه برابر 1 شوند، برابر امید آنهاست، با ضرب آنها در v_i می توان وزن‌ها را آپدیت نمود. همچنین با جایگذاری p_{theta}^{t} با p_{theta} می توان به روابط بالا دست یافت.

پیچیدگی زمانی

- **در هر مرحله از یادگیری :**

نمونه‌گیری از واحدهای پنهان و مشاهده‌شده به ترتیب دارای پیچیدگی $O(n)$ و $O(m)$ است، که n تعداد واحدهای پنهان و m تعداد واحدهای مشاهده‌شده است.

به‌روزرسانی وزن‌ها و بایاس‌ها نیز دارای پیچیدگی $O(n \cdot m)$ است.

- **برای کل الگوریتم :**

با توجه به اینکه این به‌روزرسانی‌ها برای k-مرحله و برای تمام نمونه‌های مجموعه آموزشی $|S|$ تکرار می‌شود، پیچیدگی کلی الگوریتم برابر است با $O(|S| \cdot k \cdot (m \cdot n + m + n))$

انتظارات از خروجی مدل

1. **یادگیری بازنمایی فشرده:**

بعد از آموزش، مدل باید ویژگی‌های مهم داده‌های ورودی را استخراج کند و این ویژگی‌ها را در واحدهای پنهان خود ذخیره کند. این موضوع می‌تواند در کاربردهایی مانند خوشه‌بندی یا کاهش ابعاد داده مؤثر باشد.

2. **مدل‌سازی توزیع داده‌ها:**

مدل باید بتواند توزیع احتمالی داده‌های آموزشی را به‌خوبی تقریب بزند. این ویژگی کمک می‌کند تا داده‌های مشابه با داده‌های آموزشی با احتمال بیشتری بازتولید شوند.

3. **سرعت یادگیری مناسب:**

الگوریتم با مقدار k کوچک، مانند $k=1$، سرعت یادگیری بالایی دارد، اما ممکن است دقت گرادیان کافی را نداشته باشد. با این حال، این تنظیم معمولاً برای داده‌های با پیچیدگی کمتر مناسب است.

4. **تولید داده‌های جدید:**

یکی از قابلیت‌های مهم RBM ، توانایی تولید داده‌هایی است که شباهت زیادی به داده‌های اصلی دارند. این ویژگی در کاربردهایی مانند بازسازی تصویر یا تولید داده‌های مصنوعی بسیار مفید است.

پرسش 11

همانطور که تا الان مشاهده کردید، ماشین بولتزمن مناسب کار کردن با داده های باینری است، v,h هر دو متغیر های باینری هستند، آیا ممکن است این مدل ها را برای داده های پیوسته استفاده کرد؟ پیشنهاد شما چیست، برای $v, h \in \mathbb{R}$ و همچنین $v, h \in [0,1]$ آیا با الگوریتم های مشابه با الگوریتم قبل میتوان این مدل ها را آموزش داد؟

۱ .داده‌های پیوسته

برای کار با داده‌های پیوسته، استفاده از **Gaussian-Bernoulli RBM** یا **Gaussian RBM** پیشنهاد می‌شود. در این مدل، فرض بر این است که داده‌های قابل مشاهده (v) از توزیع گاوسی پیروی می‌کنند. بنابراین، واحدهای قابل مشاهده پیوسته بوده و واحدهای پنهان همچنان باینری باقی می‌مانند. برای این مدل، تابع انرژی به‌صورت زیر تعریف می‌شود:

$$E(v,h) = \sum_{i=1}^{n} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\omega_{ij} v_i h_j}{\sigma_i} - \sum_{j=1}^{m} c_j h_h$$

در این رابطه:

- b_i, c_j بایاس واحدهای قابل مشاهده است.
- σ_i انحراف معیار داده‌های پیوسته را نشان می‌دهد.

- ω_{ij} وزن اتصال بین واحدهای قابل مشاهده و واحدهای پنهان است.

برای آموزش این مدل، لازم است که علاوه بر وزن‌ها و بایاس‌ها ، مقادیر نیز به‌درستی تخمین زده شوند.

۲ .داده‌های در بازه [0,1]

برای داده‌هایی که در بازه $[0,1]$ قرار دارند (مانند داده‌های نرمال‌سازی‌شده یا احتمالات)، می‌توان از همان مدل Bernoulli RBMاستفاده کرد، اما با این فرض که مقادیر داده‌ها نشان‌دهنده احتمال هستند. در این حالت:

- نمونه‌گیری از داده‌ها با استفاده از مقادیر احتمالی در نظر گرفته می‌شود.
- همچنان می‌توان از فرآیند استاندارد Gibbs Sampling برای آموزش استفاده کرد.

۳ .استفاده از الگوریتمContrastive Divergence

الگوریتم Contrastive Divergence (CD) که معمولاً برای آموزش RBM استفاده می‌شود، برای داده‌های پیوسته یا در بازه $[0,1]$ نیز قابل استفاده است، اما تغییرات زیر لازم است:

- درGaussian RBM ، هنگام نمونه‌گیری از $v^{(t)}$، به جای استفاده از توزیع برنولی، باید از توزیع گاوسی استفاده شود.
- سایر بخش‌های الگوریتم مانند محاسبه گرادیان وزن‌ها و به‌روزرسانی بایاس‌ها مشابه نسخه استاندارد باقی می‌مانند.

بخش عملی

در این بخش از گزارش، کد های زده شده را بررسی می کنیم.

پیش پردازش داده ها

```
transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: (x > 0.5).float())  # Binarize the data
        #transforms.Normalize((0.5,), (0.5,))
    ])

# Load the MNIST dataset
def load_mnist(batch_size=64):
        train_dataset = datasets.MNIST(
            root="./data",
            train=True,
            transform=transform,
            download=True
        )

        test_dataset = datasets.MNIST(
            root="./data",
            train=False,
            transform=transform,
            download=True
        )

        train_loader = DataLoader(
            train_dataset,
            batch_size=batch_size,
            shuffle=True
        )

        test_loader = DataLoader(
            test_dataset,
            batch_size=batch_size,
            shuffle=False
        )

        return train_loader, test_loader

train_loader, test_loader = load_mnist()
```

کد بالا برای لود کردن دیتاست MNIST و انجام بعضی پیش‌پردازش ها بر روی آن نوشته شده است.

در این کد ابتدا یک transform برای انجام بعضی تبدیلات بر روی دیتاست تعریف می‌کنیم. این تبدیلات شامل تغییر داده

به ساختار tensor و باینری کردن عکس ها است. باینری کردن به این صورت اتفاق می افتاد که تمام پیکسل هایی که مقدارشان از 0.5 بالاتر است را به عنوان پیکسل با مقدار 1 و بقیه پیکسل ها را به عنوان پیکسل با مقدار 0 در نظر می‌گیریم.

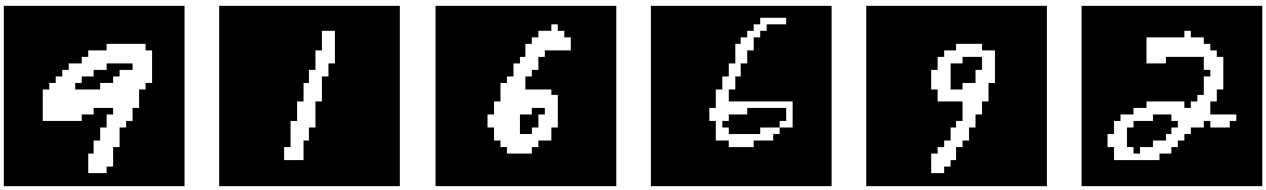
سپس تابع load_mnist برای لود کردن دیتاست های train و test (با اعمال تبدیلات گفته شده) و سپس ساخت DataLoader با استفاده از این دیتاست ها نوشته شده است. batch_size انتخابی برای DataLoader ها 64 است.

در بخش بعدی، برای نمایش بعضی از نمونه های دیتاست تابع visualize_samples نوشته شده است.
این تابع یک DataLoader را به عنوان ورودی گرفته و سپس با لود کردن یک batch ، نمونه داده را نمایش می‌دهد.
همانطور که معلوم است، عکس ها باینری شده‌اند و تمام پیکسل ها یا 1 و یا 0 هستند.

```
# Visualize samples from the dataset
def visualize_samples(data_loader):
        data_iter = iter(data_loader)
        images, _ = next(data_iter)
        images = images[:6]  # Select 6 samples

        fig, axes = plt.subplots(1, 6, figsize=(15, 3))
        for i, img in enumerate(images):
            axes[i].imshow(img.view(28, 28), cmap="gray")
            axes[i].axis("off")
        plt.show()


print("Visualizing samples from the dataset:")
visualize_samples(train_loader)
```



پیاده سازی الگوریتم و یادگیری

در این بخش، کلاس مدل اصلی Restricted Boltzmann Machine نوشته شده است. متد ها و بخش های مختلف این کلاس توضیح داده می‌شود.

```
class RBM:
def __init__(self, input=None, n_visible=784, n_hidden=128, W=None,
hbias=None, vbias=None, numpy_rng=None):
self.n_visible = n_visible
self.n_hidden = n_hidden

if numpy_rng is None:
numpy_rng = np.random.RandomState(1234)

if W is None:
a = 1. / n_visible
W = numpy_rng.uniform(-a, a, size=(n_visible, n_hidden))

if hbias is None:
hbias = np.zeros(n_hidden)

if vbias is None:
vbias = np.zeros(n_visible)

self.W = W
self.hbias = hbias
self.vbias = vbias
self.numpy_rng = numpy_rng

def sigmoid(self, x):
return 1 / (1 + np.exp(-x))

def sample_h_given_v(self, v):
h_mean = self.sigmoid(np.dot(v, self.W) + self.hbias)
h_sample = self.numpy_rng.binomial(size=h_mean.shape, n=1, p=h_mean)
return h_mean, h_sample

def sample_v_given_h(self, h):
v_mean = self.sigmoid(np.dot(h, self.W.T) + self.vbias)
v_sample = self.numpy_rng.binomial(size=v_mean.shape, n=1, p=v_mean)
return v_mean, v_sample

def gibbs_hvh(self, h0_sample):
v_mean, v_sample = self.sample_v_given_h(h0_sample)
h_mean, h_sample = self.sample_h_given_v(v_sample)
return v_mean, v_sample, h_mean, h_sample

def contrastive_divergence(self, data, k=1, lr=0.1):
```

```
ph_mean, ph_sample = self.sample_h_given_v(data)

chain_start = ph_sample
for step in range(k):
if step == 0:
nv_mean, nv_sample, nh_mean, nh_sample = self.gibbs_hvh(chain_start)
else:
nv_mean, nv_sample, nh_mean, nh_sample = self.gibbs_hvh(nh_sample)

self.W += lr * (np.dot(data.T, ph_sample) - np.dot(nv_sample.T, nh_mean)) /
len(data)
self.vbias += lr * np.mean(data - nv_sample, axis=0)
self.hbias += lr * np.mean(ph_sample - nh_mean, axis=0)

def reconstruct(self, v):
h = self.sigmoid(np.dot(v, self.W) + self.hbias)
reconstructed_v = self.sigmoid(np.dot(h, self.W.T) + self.vbias)
return reconstructed_v

def save_model(self, file_path):
with open(file_path, 'wb') as f:
pickle.dump(self, f)


@staticmethod
def load_model(file_path):
with open(file_path, 'rb') as f:
return pickle.load(f)
```

1. تابع سازنده(__init__)

این تابع هنگام ایجاد یک نمونه از کلاس RBM فراخوانی می‌شود و پارامترها و وزن‌های اولیه مدل را تنظیم می‌کند:

- n_visible: تعداد نرون‌های لایه ورودی (پیش‌فرض 784 برای تصاویر 28×28).
- n_hidden: تعداد نرون‌های لایه مخفی (پیش‌فرض 128).
- W: ماتریس وزن بین لایه‌های ورودی و مخفی.
- hbias: بایاس لایه مخفی.
- vbias: بایاس لایه ورودی.
- numpy_rng: برای ایجاد اعداد تصادفی استفاده می‌شود.

نکات:

- اگر W تعریف نشده باشد، وزن‌ها به صورت تصادفی مقداردهی می‌شوند.

- بایاس‌ها (hbias و vbias) به‌صورت پیش‌فرض صفر تنظیم می‌شوند.

---

2. تابع sigmoid

این تابع، تابع فعال‌سازی سیگموید را پیاده‌سازی می‌کند:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

برای محاسبه احتمال‌ها در مدل استفاده می‌شود.

---

3. تابع sample_h_given_v

این تابع نورون‌های لایه مخفی (h) را با توجه به نرون‌های لایه ورودی (v) نمونه‌گیری می‌کند:

- ابتدا مقادیر متوسط (میانگین احتمال) نرون‌های مخفی با استفاده از وزن‌ها و بایاس محاسبه می‌شود.
- سپس مقادیر باینری 0) یا 1)برای هر نرون مخفی، بر اساس توزیع برنولی نمونه‌گیری می‌شوند.

---

4. تابع sample_v_given_h

این تابع مشابه تابع قبلی است، اما نرون‌های لایه ورودی (v) را با توجه به نرون‌های لایه مخفی (h) نمونه‌گیری می‌کند.

---

5. تابع gibbs_hvh

این تابع یک گام نمونه‌گیری گیبز (Gibbs Sampling) را انجام می‌دهد:

- ابتدا با استفاده از نرون‌های مخفی(h) ، لایه ورودی (v) را نمونه‌گیری می‌کند.
- سپس با استفاده از لایه ورودی نمونه‌گیری‌شده، لایه مخفی را نمونه‌گیری می‌کند .این گام در زنجیره مارکوف (Markov Chain)استفاده می‌شود.

---

6. تابع contrastive_divergence

الگوریتم **Contrastive Divergence (CD)** را پیاده‌سازی می‌کند که ابزار اصلی آموزش و به روز کردن وزن های مدل است. ورودی های تابع به این شکل هستند:

- data: داده‌های ورودی برای آموزش.

- k: تعداد گام‌های نمونه‌گیری گیبز (پیش‌فرض 1).
- lr: نرخ یادگیری (پیش‌فرض 0.1).

هدف : به‌روزرسانی وزن‌ها(W) ، بایاس ورودی (vbias) و بایاس مخفی (hbias) با استفاده از تفاوت بین داده‌های واقعی و داده‌های بازسازی‌شده توسط مدل

محاسبه مقادیر اولیه لایه مخفی

تابع ابتدا احتمال‌های فعال شدن نرون‌های مخفی را با توجه به داده‌های واقعی محاسبه می‌کند:

```
ph_mean, ph_sample = self.sample_h_given_v(data)
```

- ph_mean: میانگین احتمال فعال شدن نرون‌های مخفی.
- ph_sample: نمونه‌های باینری نرون‌های مخفی که از توزیع برنولی تولید شده‌اند.

این مقادیر نشان‌دهنده "نمایش اولیه (Initial Representation) "داده‌های ورودی در فضای لایه مخفی هستند.

شروع زنجیره گیبز

نمونه‌گیری از زنجیره گیبز با استفاده از نمونه‌های لایه مخفی (ph_sample) آغاز می‌شود:

```
chain_start = ph_sample
```

اجرای گام‌های گیبز(Gibbs Sampling)

برای بهبود بازسازی داده‌ها، نمونه‌گیری‌های متوالی با استفاده از زنجیره مارکوف انجام می‌شود. تعداد این نمونه‌گیری‌ها با پارامتر k مشخص می‌شود:

```
for step in range(k):
   if step == 0:
      nv_mean, nv_sample, nh_mean, nh_sample =
self.gibbs_hvh(chain_start)
   else:
       nv_mean, nv_sample, nh_mean, nh_sample =
self.gibbs_hvh(nh_sample)
```

- **مرحله ۱ (Sample v):** ابتدا داده‌های ورودی بازسازی می‌شوند.
- **مرحله ۲ (Sample h):** سپس داده‌های مخفی دوباره نمونه‌گیری می‌شوند.

به‌روزرسانی وزن‌ها و بایاس‌ها

پس از انجام نمونه‌گیری، وزن‌ها و بایاس‌ها با استفاده از تفاوت بین داده‌های واقعی و بازسازی‌شده تنظیم می‌شوند:

```
self.W += lr * (np.dot(data.T, ph_sample) - np.dot(nv_sample.T, nh_mean)) / len(data)
```

- **به‌روزرسانی W**
  - `np.dot(data.T, ph_sample)`: ضرب داخلی بین داده‌های واقعی و مقادیر اولیه لایه مخفی.
  - `np.dot(nv_sample.T, nh_mean)`: ضرب داخلی بین داده‌های بازسازی‌شده و مقادیر بازسازی‌شده لایه مخفی.
  - تفاوت این دو، گرادیان وزن‌ها را نشان می‌دهد.

```
self.vbias += lr * np.mean(data - nv_sample, axis=0)
self.hbias += lr * np.mean(ph_sample - nh_mean, axis=0)
```

- **به‌روزرسانی vbias**: میانگین تفاوت بین داده‌های واقعی و بازسازی‌شده.
- **به‌روزرسانی hbias**: میانگین تفاوت بین مقادیر اولیه و بازسازی‌شده لایه مخفی.

Contrastive Divergence به دنبال کاهش تفاوت (واگرایی) بین توزیع داده‌های واقعی و توزیع داده‌های بازسازی‌شده توسط مدل است. این کاهش واگرایی منجر به یادگیری بهتر روابط بین داده‌ها می‌شود.

---

7. تابع reconstruct

این تابع یک ورودی را دریافت می‌کند و نسخه بازسازی‌شده آن را برمی‌گرداند:

1. ابتدا ورودی به فضای ویژگی‌های مخفی نگاشت می‌شود.
2. سپس از ویژگی‌های مخفی برای بازسازی ورودی استفاده می‌شود.

---

8. تابع save_model

مدل آموزش‌داده‌شده را در یک فایل ذخیره می‌کند. از کتابخانه pickle برای سریال‌سازی استفاده می‌شود.

---

9. تابع load_model

مدل ذخیره‌شده را از یک فایل بارگذاری می‌کند و برای استفاده آماده می‌کند.

حال برای آموزش مدل تابع زیر نوشته شده است. در این تابع بر اساس hyperparameter های داده شده، ابتدا یک مدل rbmساخته می‌شود سپس این مدل با استفاده از DataLoader ای که در اختیارش گذاشته شده و متد contrastive_divergenceمدل rbm آموزش داده می‌شود .
همچنین در هر epoch خطای بازسازی روی داده های test محاسبه شده و print می‌شود. این کار برای نظارت بر روند آموزش صورت گرفته است.

```
def train_rbm(train_loader, test_loader, n_visible, n_hidden, learning_rate,
epochs, k):
      rbm = RBM(n_visible=n_visible, n_hidden=n_hidden)

      for epoch in range(epochs):
            for batch, _ in train_loader:
                batch = batch.view(-1, n_visible).numpy()
                rbm.contrastive_divergence(batch, k=k, lr=learning_rate)

            test_data = next(iter(test_loader))[0].view(-1, n_visible).numpy()
            reconstruction = rbm.reconstruct(test_data)
            loss = np.mean((test_data - reconstruction) ** 2)
           print(f"Epoch {epoch + 1}, k={k}: Reconstruction Loss = {loss:.4f}")

      return rbm
```

در این بخش، hyperparameter ها مشخص شده، مدل ها آموزش داده می‌شوند و سپس در فایل های pkl. ذخیره می‌شوند.

```
# Parameters
num_visible = 28 * 28  # Number of visible units
num_hidden = 256       # Number of hidden units
learning_rate = 0.1
epochs = 15
print("Training RBM with k=1")
rbm_k1 = train_rbm(train_loader, test_loader, num_visible, num_hidden,
learning_rate, epochs, k=1)
rbm_k1.save_model("rbm_k1.pkl")

print("Training RBM with k=5")
rbm_k5 = train_rbm(train_loader, test_loader, num_visible, num_hidden,
learning_rate, epochs, k=5)
rbm_k5.save_model("rbm_k5.pkl")

print("Training RBM with k=10")
rbm_k10 = train_rbm(train_loader, test_loader, num_visible, num_hidden,
learning_rate, epochs, k=10)
rbm_k10.save_model("rbm_k10.pkl")
```

توابع زیر به ترتیب برای ساخت نمونه با استفاده از مدل و نمایش نمونه های ساخته شده، نوشته شده‌اند. نحوه ساخت نمونه به این ترتیب است که یک ماتریس تصادفی را به عنوان لایه ورودی به مدل می‌دهیم سپس با استفاده از نمونه گیری گیبز به تعداد num_gibbs_steps خروجی را می‌سازیم.

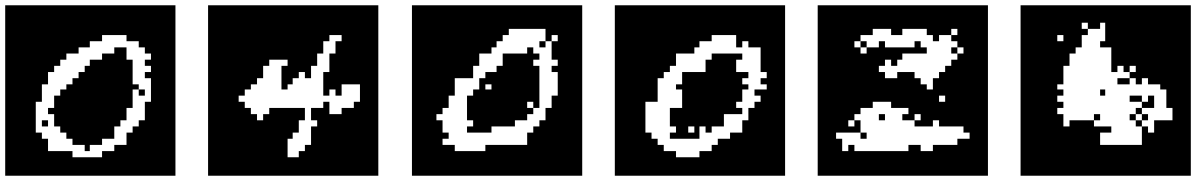
```
# Generate samples from a trained RBM
def generate_samples(rbm, num_samples=6, num_gibbs_steps=100):
      random_visible = np.random.rand(num_samples, rbm.n_visible) > 0.5
      for _ in range(num_gibbs_steps):
          _, random_visible = rbm.sample_h_given_v(random_visible)
          _, random_visible = rbm.sample_v_given_h(random_visible)
      return random_visible

def visualize_generated_samples(generated_samples):
      fig, axes = plt.subplots(1, len(generated_samples), figsize=(15, 3))
      for i, sample in enumerate(generated_samples):
          axes[i].imshow(sample.reshape(28, 28), cmap="gray")
          axes[i].axis("off")
      plt.show()
```

خروجی های مدل ها به این شکل هستند:

Generating samples from RBM with k=1

Generating samples from RBM with k=5



Generating samples from RBM with k=10

نمایش روند نمونه سازی

در این بخش با استفاده از تابع زیر یک ویدیو از تمام حالت های لایه ورودی (visible) در حین ساخت نمونه می‌سازیم.
ویدیو ساخته شده در فایل ارسالی وجود دارد.

```
# Animate the MCMC improvement process
def animate_mcmc(rbm, num_samples=1, num_gibbs_steps=800,
output_file="mcmc_animation.mp4"):
        random_visible = (np.random.rand(num_samples, rbm.n_visible) >
0.5).astype(float)
        fig, ax = plt.subplots(figsize=(5, 5))

        img = ax.imshow(random_visible[0].reshape(28, 28), cmap="gray",
animated=True)
        ax.axis("off")

        def update(frame):
            nonlocal random_visible
            _, random_visible = rbm.sample_h_given_v(random_visible)
            _, random_visible = rbm.sample_v_given_h(random_visible)
            img.set_array((random_visible[0].reshape(28, 28) >
0.5).astype(float))
            return [img]

        writer = FFMpegWriter(fps=10, metadata=dict(artist="RBM Animation"),
bitrate=1800)
        with writer.saving(fig, output_file, dpi=100):
            for frame in range(num_gibbs_steps):
                update(frame)
                writer.grab_frame()

print("Animating the MCMC process for RBM with k=5")
animate_mcmc(rbm_k5, output_file="mcmc_animation_k5.mp4")
```

کنترل روی نمونه های تولیدی

برای کنترل کردن نمونه های تولیدی مدل می‌توان از Conditional Restricted Boltzmann Machine استفاده کرد.

پیشنهاد ما برای ایجاد این کنترل اضافه کردن 10 نورون به لایه ورودی به عنوان نشان‌دهنده لیبل ورودی ها و نمونه های ساخته شده است. به این صورت که هنگام آموزش علاوه بر خود عکس ورودی، لیبل عکس نیز به عنوان یک بردار one-hotبه لایه ورودی داده می‌شود.

و همگام تولید نمونه نیز میتوان 10 نورون لایه ورودی را برابر با بردار one-hot لیبل دلخواه قرار داد و بقیه لایه را با مقادیر تصادفی پر کرد. در حین نمونه‌گیری گیبز نورون های مربوط به لیبل تغییر پیدا نمی‌کنند ولی در ساخت مقادیر لایه پنهان تاثیرگذارند.

یک پیاده‌سازی اولیه از این مدل به این صورت انجام شده‌است. اما نمونه های تولیدی به طور کامل کنترل نمی‌شوند. یک دلیل محتمل برای این امر، کم بودن نورون های لیبل نسبت به نورون های تصویر و در نتیجه کم بودن تاثیر آنهاست.

```
class ConditionalRBM:

    def __init__(self, n_visible=784, n_hidden=256, n_labels=10,
numpy_rng=None):

        self.n_visible = n_visible
        self.n_hidden = n_hidden
        self.n_labels = n_labels

        if numpy_rng is None:
            numpy_rng = np.random.RandomState(1234)
        self.numpy_rng = numpy_rng

        # Weight matrix, and biases for hidden and visible units
        # W shape = (n_visible + n_labels, n_hidden)
        self.W = numpy_rng.uniform(
            low=-0.1, high=0.1, size=(n_visible + n_labels, n_hidden)
        )
        self.hbias = np.zeros(n_hidden, dtype=np.float32)
        self.vbias = np.zeros(n_visible + n_labels, dtype=np.float32)

    @staticmethod
    def sigmoid(x):

        return 1.0 / (1.0 + np.exp(-x))

    def sample_h_given_v(self, v):

        h_mean = self.sigmoid(np.dot(v, self.W) + self.hbias)
        h_sample = self.numpy_rng.binomial(n=1, p=h_mean, size=h_mean.shape)
        return h_mean, h_sample
```

```
    def sample_v_given_h(self, h, fixed_label=None):

        v_mean = self.sigmoid(np.dot(h, self.W.T) + self.vbias)
        # If we want to fix the label portion, override the last n_labels
entries
        if fixed_label is not None:
            v_mean[:, -self.n_labels:] = fixed_label
        v_sample = self.numpy_rng.binomial(n=1, p=v_mean, size=v_mean.shape)
        return v_mean, v_sample

    def gibbs_hvh(self, h_sample_start, fixed_label=None):

        v_mean, v_sample = self.sample_v_given_h(h_sample_start,
fixed_label=fixed_label)
        h_mean, h_sample = self.sample_h_given_v(v_sample)
        return v_mean, v_sample, h_mean, h_sample

    def contrastive_divergence(self, data, labels, k=1, lr=0.1):
        # Combine data and labels into a single visible vector
        v0 = np.hstack((data, labels))
        # Positive phase: sample hidden given v0
        ph_mean, ph_sample = self.sample_h_given_v(v0)

        # Start the chain from the positive hidden sample
        h_sample = ph_sample

        # Gibbs sampling (negative phase)
        for step in range(k):
            v_mean, v_sample, nh_mean, nh_sample = self.gibbs_hvh(
                h_sample, fixed_label=labels
            )
            h_sample = nh_sample  # continue chaining


        self.W += lr * (
            np.dot(v0.T, ph_sample) - np.dot(v_sample.T, nh_mean)
        ) / len(data)

        self.vbias += lr * np.mean((v0 - v_sample), axis=0)
        self.hbias += lr * np.mean((ph_sample - nh_mean), axis=0)

    def reconstruct(self, v):
        h = self.sigmoid(np.dot(v, self.W) + self.hbias)
        return self.sigmoid(np.dot(h, self.W.T) + self.vbias)

    def generate(self, label, num_samples=1, num_gibbs_steps=100):
        label_one_hot = np.zeros((num_samples, self.n_labels))
```

```
        label = np.array(label, ndmin=1)
        label_one_hot[np.arange(num_samples), label] = 1

        # Start with random visible units, and attach the chosen labels
        random_vis = self.numpy_rng.rand(num_samples, self.n_visible)
        v = np.hstack((random_vis, label_one_hot))


        for _ in range(num_gibbs_steps):
            _, h_sample = self.sample_h_given_v(v)
            _, v = self.sample_v_given_h(h_sample, fixed_label=label_one_hot)


        return v[:, :-self.n_labels]

    def save_model(self, file_path):

        with open(file_path, 'wb') as f:
            pickle.dump(self, f)

    @staticmethod
    def load_model(file_path):
        with open(file_path, 'rb') as f:
            return pickle.load(f)
```

بقیه اجزای کد مانند قبل است. نمونه های تولیدی توسط این مدل را میبینیم:

```
# Generate samples for label '6'
num_gen = 6
generated_images = crbm.generate(label=[6] * num_gen, num_samples=num_gen,
num_gibbs_steps=800)

# Visualize generated samples
fig, axes = plt.subplots(1, num_gen, figsize=(15, 3))
for i, sample in enumerate(generated_images):
     axes[i].imshow(sample.reshape(28, 28), cmap="gray")
     axes[i].axis("off")
plt.show()
```