دسته‌بندی اصلی روش‌های VQ:

کلاسیک ✓
- Scalar Quantization
- K-mean VQ یا LBG (Linde_Buzo_Gray)

روش‌های مبتنی بر تجزیه ✓
- product Quantization
- Optimized Product Quantization
- Residual Quantization
- Additive Quantization
- Composite Quantization

روش‌های یادگیری عمیق ✓
- Vector quantized Variational Autoencoder
- VQ_GAN
- Residual Quantized VAE

فعلاً نمی‌خواد X
- روش‌های اضافی دیگر ⟹
  - Tree_Structured VQ
  - Lattice VQ
  - learning vector Quantization

روش Scalar Quantization

- هر مولفه (بُعد) از بردار داده به طور مستقل کوانتیزه می‌شود.
- به جای کار روی کل بردار (مثل Vector Quantization) روی هر بعد جداگانه کار می‌کنیم

1 تقسیم بازه مقادیر داده

- فرض کنید داده‌ها اسکالرهای پیوسته در یک بازه مثل $[x_{min}, x_{max}]$ هستند.
- بازه به چند بخش تقسیم می‌شود.

2 تعریف بازنمایی

- برای هر بخش یک مقدار نماینده تعریف می‌کنیم

3 نگاشت ورودی به نزدیک‌ترین مقدار

- هر ورودی x به نماینده بخش متناظر نگاشت می‌شود.

4 بازسازی

- در بازسازی، فقط نماینده متناظر استفاده می‌شود نه مقدار دقیق اولیه

مثال:

داده $\Rightarrow [0, 16]$

تقسیم‌بندی به 4 قسمت $\Rightarrow$

$[0,4) \Rightarrow$ نماینده ۲

$[4,8) \Rightarrow$ نماینده ۶ — if $x = 7.2 \Rightarrow x \simeq 6$

$[8,12) \Rightarrow$ نماینده ۱۰ — if $x = 15.9 \Rightarrow \approx 14$

$[12,16] \Rightarrow$ نماینده ۱۴

انواع اصلی:

$\rightarrow$ Uniform Scalar Quantization (USQ)

$\rightarrow$ Non-uniform Scalar Quantization (NUS

۱۵۰
۸۰
۱۱

① روش کلاسیک k-means در VQ

مراحل الگوریتم

۱. انتخاب اولیه مراکز ← به صورت تصادفی یا با روش هایی مانند k-means ++

۲. مرحله تخصیص ← هر بردار ورودی به نزدیک‌ترین مرکز خوشه نسبت داده می‌شود

۳. به روز رسانی ← هر مرکز خوشه به میانگین داده‌های اختصاص یافته به آن تغییر می‌کند

۴. تکرار ← مراحل تخصیص و به روز رسانی تا همگرایی تکرار می‌شوند

② روش LBG (Linde-Buzo-Gray)

- یک شکل خاص از الگوریتم k-means است.

تفاوت‌های کلیدی با k-means

• ابتدا با یک کد واژه شروع می‌کند (میانگین کل داده‌ها)

• سپس به صورت تدریجی کدبوک را گسترش می‌دهد تا به اندازه دلخواه برسد

• در هر مرحله، روی مجموعه کد واژه‌ای فعلی الگوریتم k-means اجرا می‌شود

مراحل الگوریتم LBG:

۱. شروع اولیه ← یک کد واژه ← میانگین همه داده‌ها

۲. افزایش اندازه کدبوک ← هر کد واژه به ۲ کد واژه جدید تقسیم می‌شود

۳. اجرای k-means روی کدبوک جدید

• داده‌ها به نزدیک‌ترین کد واژه نسبت داده می‌شود.

• مراکز جدید محاسبه می‌شود

۴. تکرار تا رسیدن به تعداد مطلوب کد واژه

• مراحل ۲ و ۳ تکرار می‌شوند تا کدبوک به اندازه دلخواه برسد.

مثالی از الگوریتم LBG:

$X = \{2, 3, 3, 4, 5, 6, 8, 9, 12, 13, 14\}$ -

$N = 11$

$$M = \frac{79}{11} = 7.18$$

$\varepsilon = ۰/۰۱$

- گام ۱: شروع و تقسیم اولیه $K=2$

$C_1 = (1-\varepsilon) * M = ۰/۹۹ \times ۷/۱۸ = ۷/۱۱$

$C_2 = (1+\varepsilon) * M = ۱/۰۱ \times ۷/۱۸ = ۷/۲۵$

- تخصیص داده‌ها به نزدیکترین مرکز (مرحلهٔ تخصیص)

- برای هر $x \in X$ فاصله به C_1 و C_2 را محاسبه می‌کنیم و نزدیکتر را انتخاب می‌کنیم

$2 \Rightarrow C_1$, $3 \Rightarrow C_1$, $6 \Rightarrow C_1$, $4 \Rightarrow C_1$, $5 \Rightarrow C_1$

$8 \Rightarrow C_2$, $9 \Rightarrow C_2$, $12 \Rightarrow C_2$, $13 \Rightarrow C_2$, $14 \Rightarrow C_2$

دو خوشه $\Longrightarrow \{2, 3, 3, 4, 5, 6\}$; $\{8, 9, 12, 13, 14\}$

- به‌روزرسانی مراکز

$$C'_1 = \frac{2+3+3+4+5+6}{6} = \frac{23}{6} = 3.83$$

$$C'_2 = \frac{۸+۹+۱۲+۱۳+۱۴}{۵} = \frac{۵۶}{۵} = ۱۱.۲$$

$\Longrightarrow C = \{۱۱,۲ , ۳,۸۳\}$

- گام ۲ - تقسیم هر کدوم از مراکز برای رسیدن به $K=4$;

$$\begin{cases} (\varepsilon - 1) \times 3.83 = 3.79 \\ (\varepsilon + 1) \times 3.83 = 3.87 \\ (1-\varepsilon) \times ۱۱,۲ = ۱۱,۰۸ \\ (1+\varepsilon) \times ۱۱,۲ = ۱۱,۳۱ \end{cases} \xrightarrow[K=4]{\text{مراکز اولیه}} \{۱۱.۳۱ , ۱۱,۰۸ , ۳,۸۷ , ۳,۷۹\}$$

گام ۳- بروزرسانی روی این ۴ مرکز

2, 3, 3 ⟹ 3.79   8, 9 ⟹ ۱۱,۰۸

6, 5, 4 ⟹ 3.87   12, 13, 14 ⟹ 11.31

بروزرسانی مراکز

خوشه {2, 3, 3} ⟹ ۸/۳ = ۲,۶۶

خوشه {6, 5, 4} ⟹ 5

خوشه {8, 9} ⟹ 8.5

خوشه {12, 13, 14} ⟹ 13

⟹ $C = \{13, 8.5, 5, 2.66\}$

دیگر ابعادی مراکز تغییر نمی‌کنند پس همگرا شده‌ایم

کدبوک ⟹ $C = \{13, 8.5, 5, 2.66\}$

بنابراین مقادیر کوانتیزه به صورت زیر خواهند بود:

$2 \to 2.66$ $3 \to 2.66$ $3 \to 2.66$ $4 \to 5$ $5 \to 5$ $6 \to 5$

$8 \to 8.5$ $9 \to 8.5$ $12 \to 13$ $13 \to 13$ $14 \to 13$

بنابراین خطای میانگین مربعات:

$MSE \approx 146$

بررسی روش Product Quantization :

◊ مشکل اصلی در VQ این است که اگر داده‌ها در یک فضای با ابعاد بالا (مثلاً 128 یا 512) باشند (که معمولاً در مسائل ما هر دو دسته روش‌ها با ابعاد بالا است) ← ساختن کدبوک با تعداد کافی کدواژه برای پوشش دقیق این فضا بسیار پرهزینه است (هم از نظر حافظه و هم از نظر محاسبات)

* PQ برای حل این مشکل، به جای کوانتیزه کردن کل بردار یکجا، آن را به چند زیربردار مستقل (sub-vectors) تقسیم می‌کند و هر بخش را جداگانه کوانتیزه می‌کند.

مثال عددی

هر بردار 4 بعدی است و کلاً 6 بردار داریم
(برای سادگی 4 بعدی در نظر گرفتیم)

$x_1 = (2.0, 3.0, 6.0, 6.0)$

$x_2 = (2.2, 3.5, 5.8, 6.1)$

$x_3 = (1.9, 2.8, 6.2, 5.9)$

$x_4 = (4.1, 5.0, 8.1, 7.9)$

$x_5 = (4.3, 5.1, 7.9, 8.2)$

$x_6 = (3.8, 4.6, 8.2, 8.1)$

مرحله تقسیم به زیربردارها (با روش تقسیم متوالی ساده) $M = 2$ (هر زیربردار 2 بعدی)

- زیرفضای 1 (ابعاد 1 و 2)

$(2,3), (2.2, 3.5), (1.9, 2.8), (4.1, 5), (4.3, 5.1), (3.8, 4.6)$

- زیرفضای 2 (ابعاد 3 و 4)

$(6,6), (5.8, 6.1), (6.2, 5.9), (8.1, 7.9), (7.9, 8.2), (8.2, 8.1)$

مرحله ۲ - آموزش کدبوک‌ها با k-means با $k=2$ برای هر زیرفضا:

* زیرفضای ۱ ← مراکز دو خوشه:

خوشه ۱:

$$[(1.9, 2.8), (2.2, 3.5), (2.0, 3.0)]$$

$$\text{مرکز} \Rightarrow c_1^{(1)} = (2.03, 3.1)$$

خوشه ۲:

$$[(3.8, 4.6), (4.3, 5.1), (4.1, 5.0)]$$

$$\text{مرکز} \Rightarrow c_2^{(1)} = (4.06, 4.9)$$

* زیرفضای ۲ ← مراکز دو خوشه:

خوشه ۱:

$$[(6.0, 6.0), (5.8, 6.1), (6.2, 5.9)]$$

$$\text{مرکز} \quad c_1^{(2)} = (6, 6)$$

خوشه ۲:

$$[(8.2, 8.1), (7.9, 8.2), (8.1, 7.9)]$$

$$\text{مرکز} \quad c_2^{(2)} = (8.06, 8.06)$$

مرحله ۳ ← کدگذاری بردار جدید (هنگام تست - فقط)

$$x_{new} = (2.1, 3.4, 5.7, 6.2)$$

$$\text{زیربردار} \Rightarrow q^{(1)} = (2.1, 3.4)$$

$$q^{(2)} = (5.7, 6.2)$$

مرحله ۳: کدگذاری

برای هر زیرفضای تا، نزدیک‌ترین کدواژه آن زیرفضا را پیدا می‌کنیم.

زیرفضای 1 ← مقایسه $q^{(1)}$ با 2 کدواژه

$$\|3.1 - 3.4\|^2 = 13.7 \Rightarrow c_0^{(1)}$$

$$\|4.9 - 3.4\|^2 = 247 \Rightarrow c_1^{(1)}$$

$$\Longrightarrow d(c_1^{(1)}) > d(c_0^{(1)})$$

$\Downarrow$

برای زیرفضای 1 نزدیک‌ترین کدواژه $c_0^{(1)}$ است. ← اندیس زیرفضای 1 ⇐ 0

زیرفضای 2 ← مقایسه $q^{(2)}$ با 2 کدواژه

$$\|6.0 - 5.7\|^2 = .136 \Rightarrow c_0^{(2)}$$

$$\|8.06 - 5.7\|^2 = 3.01 \Rightarrow c_1^{(2)}$$

$$\Longrightarrow c_0^{(2)}$$ نزدیک است

$\Downarrow$

اندیس زیرفضای 2 ⇐ 0

پس کد PQ بردار query ← $code(x) = (i_1, i_2) = (0, 0)$

مرحله ٤: بازسازی ⇐ $\hat{x} = (2.03, 3.1, 6.0, 6.0)$

$$x_{test} \Rightarrow (2.1, 3.4, 5.7, 6.2)$$

$$\|\hat{x} - x_{test}\| = 147$$

- روش Optimized Product Quantization که یکی از پیشرفته‌ترین و کاربردی‌ترین نسخه‌های PQ است
(OPQ)

◊ در PQ ما بردار را به زیرفضاها تقسیم می‌کنیم و در هر زیرفضا یک کدبوک مستقل آموزش می‌دهیم.

◊ ضعف اصلی PQ ⇐ تقسیم‌بندی ابعاد به زیرفضاها به صورت ساده و ثابت انجام می‌شود اما توزیع داده‌ها ممکن است این تقسیم‌بندی را بهینه نکند چون

- ابعاد مختلف ممکن است همبستگی داشته باشند
- قرار دادن ابعاد همبسته در زیرفضای جدا می‌تواند خطای کوانتیزاسیون را زیاد کند

◊ ایده OPQ ← قبل از تقسیم بردارها به زیرفضاها یک تبدیل خطی (معمولاً یک ماتریس چرخش ارتوگونال R) روی داده‌ها اعمال می‌شود.

$y = x \cdot R$

سپس بردار y به زیرفضاها تقسیم می‌شود روی آن PQ اعمال می‌شود.

- انتخاب R به گونه‌ای که خطای کوانتیزاسیون کمینه شود

مراحل الگوریتم OPQ

1. شروع با یک ماتریس R اولیه (مثلاً ماتریس واحد I)
2. اعمال تبدیل ← برای همه بردارها $y = x \cdot R$
3. اعمال PQ روی y
- تقسیم به زیرفضاها
- آموزش کدبوک‌ها (kmeans)
- کدگذاری بردارها

1

۴- بهینه سازی R:

• بعد از کوانتیزاسیون، بازسازی $\hat{y}$ داریم.

• خطای $\|y-\hat{y}\|^2$

• با روش های مانند SVD یا gradient descent ماتریس R را آپدیت می کنیم تا خطا کمتر شود

۵- تکرار گام های ۲-۴ تا همگرایی

مثال عددی

- ۸ بردار ۴ بعدی

- انتخاب $M=2$ زیرفضای متوالی (هر زیرفضا 2 بعد)، $K=2$ کد واژه در هر زیرفضا - مقدار کوچک

$$X = ([2, 3, 6, 6], [2.2, 3.5, 5.8, 6.1], [1.9, 2.8, 6.2, 5.9], [4.1, 5, 8.1, 7.9], [4.3, 5.1, 7.9, 8.2], [3.8, 4.6, 8.2, 8.1], [2.5, 3.2, 6.5, 6.3], [3, 3.8, 7, 6.9])$$

در هر تکرار: ۱- داده ها را با R چرخش می دهیم $Y=RX$

۲- برای هر زیرفضای m با K-means (در مثال $k=2$) کدبوک محلی می سازیم و هر نقطه را به نزدیک ترین کد واژه اختصاص می دهیم

۳- بازسازی نقاط در فضای چرخیده را داریم ← $\hat{Y}$

۴- برای پیدا کردن R جدید مسئله پروکراستس ارتوگونال (؟؟) حل می شود.
کمینه $\sum \|\hat{y}_i - Rx_i\|^2$ با قید $I = R^T R$

• حل بسته (orthogonal Procrustes) به کمک $X^T \hat{Y} = A$، $A = U^T \Sigma V$

طبق SVD → $R = U^T V$

۵- تکرار تا همگرایی

* روش Residual Quantization ← یکی از رویکردهای مهم برای کاهش خطای کوانتیزاسیون در الگوریتم (RQ)

با ابعاد بالا است. (نسبت به PQ پیچیدگی اش بیشتر است)

□ ایده اصلی RQ :

در RQ به جای اینکه بردار را مستقیماً در زیر فضا تقسیم کنیم (مثل PQ) بردار را مرحله به مرحله تقریب می‌زنیم

1 ابتدا یک کدبوک اولیه آموزش داده می‌شود و نزدیکترین کد واژه به بردار اصلی پیدا می‌شود

2 خطای بازسازی یا Residual محاسبه می‌شود:

$$x - c_1 = r_1$$

بردار اصلی

نزدیک ترین کدواژه از کدبوک اول

3 سپس روی این residual دوباره یک کدبوک جدید یاد میگیریم و نزدیکترین کدواژه c_2 پیدا می‌شود.

4 Residual بعدی:

$$c_2 - r_1 = r_2$$

(این بار تر را ابتدا مشخص می‌کنیم)

5 این کار تا M مرحله ادامه می‌یابد و در نهایت بردار اصلی تقریب زده می‌شود (به صورت جمع کدواژه‌ها)

$$c_1 + c_2 + \cdots + c_M = \hat{x}$$

* مثال عددی

$$X = \{x_1 = [2,3],\ x_2 = [3, 3.5],\ x_3 = [4,5],\ x_4 = [5, 4.5]\}$$

هدف ← دو مرحله residual با 2 کدواژه در هر مرحله (K=2)

مرحله 1 ← یادگیری کدبوک اول

1 انجام K-means روی داده‌های اصلی X با k=2 کدواژه.

C_1 کدبوک = $\{c_{11} = [2.5, 3.25],\ c_{12} = [4.5, 4.75]\}$

این مقادیر، میانگین خوشه‌ها هستند

۲ انتساب هر بردار به نزدیکترین کد واژه:

$x_1 = [2,3] \Rightarrow c_{11} = [2.5, 3.25]$ | $r_1 \Rightarrow [-0.5, -0.25]$

$x_2 = [3, 3.5] \Rightarrow c_{11} = \sim\sim$ | $r_1 \Rightarrow [0.25, 0.5]$

$x_3 = [4,5] \Rightarrow c_{12} = [4.5, 4.75]$ | $r_1 \Rightarrow [-0.25, 0.5]$

$x_4 = [5, 4.5] \Rightarrow c_{12} = \sim\sim$ | $r_1 \Rightarrow [0.5, -0.25]$

۳ محاسبه residual مرحله ۱

مرحله ۲ ← یادگیری کدبوک دوم روی residual:

۱ انجام k-mean روی residual ($K=2$)

• کدبوک دوم $C_2 = \{c_{21} = [0.5, 0.25],\ c_{22} = [-0.5, -0.25]\}$

۲ انتساب residual به نزدیکترین کد واژه:

مرحله ۳ ← بازسازی بردارها $\hat{x} = c_1 + c_2$

روش additive quantization

- در روش VQ وقتی ابعاد بزرگ است، دیکشنری باید بسیار بزرگ باشد تا حجم کم شود و این مشکل حافظه و محاسبات ایجاد می‌کند

- ایده Additive Quantization ← یک روش برای کاهش این مشکل است. ایده این است که به جای استفاده از یک کد از دیکشنری، بردار را به صورت جمع چند کد کوچک تقریب می‌زنیم:

$$x \approx c_1 + c_2 + \dots + \underbrace{c_M}_{\text{تعداد کدبوک های کوچک}}$$

عضو دیکشنری

- این روش در واقع تعمیم Product Quantization است اما AQ اجازه می‌دهد که کدها به صورت جمعی و مستقل کمینه شوند و نه به صورت بخش های غیرهمپوشان

* آموزش AQ ← شامل 2 مرحله است:

الف - ساخت کدبوک ها

• هدف ← پیدا کردن M دیکشنری $c_1, \dots, c_M$ است که جمع عناصر آنها بیشترین نزدیکی را به داده داشته باشد.

• این کار معمولاً با روش های تکراری انجام می‌شود:

1. فرض می‌کنیم دیکشنری ها اولیه اند.

2. برای هر بردار x، ترکیب بهینه $c_1 + c_2 + \dots + c_M$ پیدا می‌شود که خطای بازسازی $\|x - (c_1 + c_2 + \dots + c_M)\|^2$ کمینه شود.

3. سپس هر کدبوک بروز می‌شود تا مجموع خطا کمینه شود.

4. این مراحل تا همگرایی ادامه می‌یابد

ب) انکدینگ بردار جدید

• برای بردار x باید $c_1, \dots, c_M$ را از دیکشنری‌ها انتخاب کنیم تا

$x \approx c_1 + \dots + c_M$ شود

• این کار می‌تواند greedy انجام شود:

1. نزدیک‌ترین کد c_1 به x را از C_1 انتخاب می‌کنیم.

2. باقیمانده $r_1 = x - c_1$ را محاسبه می‌کنیم.

3. نزدیک‌ترین کد c_2 به باقی‌مانده را از C_2 انتخاب می‌کنیم و ادامه می‌دهیم.

• دقت بالاتر نسبت به PQ • قابل استفاده برای جستجوی سریع در دیتابیس‌های بزرگ

مثال عددی:

1. فرض کنید ۲ بردار آموزشی ۲ بعدی داریم: $x_1 = [5, 7]$; $x_2 = [4, 6]$

و می‌خواهیم ۲ کدبوک $(M=2)$ با ۲ بردار هر کدبوک بسازیم:

$C_1 = \{c_{11}, c_{12}\}, C_2 = \{c_{21}, c_{22}\}$

2. مقداردهی اولیه به صورت تصادفی یا نزدیک به داده:

$$C_1^{(0)} = \{[2,3], [1,2]\}$$

$$C_2^{(0)} = \{[3,4], [2,3]\}$$

3. مرحله Encoding ←

$x_1 = [5, 7]$

مرحله 1 ← کد از C_1:

$$[5,7] - [2,3] = \|[3,4]\|^2 = 25$$

$$[5,7] - [1,2] = \|[4,5]\|^2 = 41$$

پس انتخاب می‌کنیم $c_{11} = [2,3]$

$$r_1 = [5,7] - [2,3] = [3,4]$$

• باقی مانده

• انتخاب کد از C_2 نزدیک ـ باقی مانده

$$\begin{cases} [3,4] - [3,4] = 0 \Rightarrow 0 \\ [3,4] - [2,3] = [1,1] \Rightarrow 2 \end{cases}$$

$$\Rightarrow C_{21} = [3,4]$$

برای $x_2 = [4,6]$

از C_1 انتخاب می‌کنیم

$$\begin{cases} [4,6] - [2,3] = [2,3] \Rightarrow 13 \\ [4,6] - [1,2] = [3,4] \Rightarrow 25 \end{cases}$$

پس $C_{12} = [2,3]$

• باقی مانده

$$r_1 = [4,6] - [2,3] = [2,3]$$

از C_2 انتخاب می‌کنیم:

$$\begin{cases} [2,3] - [3,4] = [-1,-1] \Rightarrow 2 \\ [2,3] - [2,3] = [0,0] \Rightarrow 0 \end{cases}$$

پس $C_{22} = [2,3]$

☐ مرور مراکز کدبوک:

برای C_1 ← میانگین بردارهایی که انتخاب شدند:

$$C_{11}^{new} = \frac{[2,3] + [2,3]}{2} = [2,3]$$

$$C_{12}^{new} = C_{12} \quad \text{تغییر نمی‌کند}$$

برای C_2 ← چون برای هر یک فقط یک داده داریم پس نیازی به میانگین‌گیری نیست

$$C_{21}^{new} = [3,4] \quad ; \quad C_{22}^{new} = [2,3]$$

☐ بازسازی بردار:

$$\begin{cases} x_1 \approx C_{11} + C_{12} = [2,3] + [3,4] = [5,7] \\ x_2 \approx C_{21} + C_{22} = [2,3] + [2,3] = [4,6] \end{cases}$$

روش Composite Quantization

ایده ← Composite Quantization دقیقاً مثل AQ است به جای یک کدبوک بزرگ چند کدبوک کوچک داریم:

$$x \approx c_1 + c_2 + \cdots c_m$$

تفاوت با AQ:

در AQ { فقط این‌ها را جمع کن $c_1 + \cdots + c_m$ که شبیه x باشد
برای انتخاب کدبوک‌ها باید جستجوی پیچیده یا الگوریتم ترکیبی انجام داد

در CQ علاوه بر ترکیب ← $\sum c_m \approx x$ یک قید ریاضی هم اضافه می‌شود:

$$c_i^T c_j = \text{ثابت}$$

آموزش مشابه AQ ولی با یک قید اضافه

مزیت ← سرعت بیشتر (مخصوصاً جستجو در دیتابیس بزرگ) اما دقت کمتر

مثال عددی

$x_1 = [5,7]$; $x_2 = [4,6]$; $x_3 = [6,8]$

2 کدبوک داریم، هر کدبوک 2 بردار:

$C_1 = \{c_{11} = [2,3],\ c_{12} = [1,4]\}$ $\quad$ $C_2 = \{c_{21} = [3,4],\ c_{22} = [4,3]\}$

حالا بردارها را انتخاب می‌کنیم که ضرب داخلی همه ترکیب‌ها برابر ۱۸ بشود:

$$c_{21}^T c_{11} = 2*3 + 3*4 = 18$$

$c_{22}^T c_{12} = 1\times 4 + 4\times 3 = 4+12 = 16 \Rightarrow$ کمترین تغییر $c_{22} = [3.5, 3]$

۵ درصد (۷۵ - ۷۸)

حالا:

$$c_{22}^T c_{12} = 1\times 3.5 + 4\times 3 = 3.5 + 12 = 15.5 \approx 18$$

۳- بازسازی نمونه‌ها:

$$\begin{cases} x_1 = [5,7] \approx c_{11} + c_{21} = [2,3] + [3,4] = [5,7] \\ x_2 = [4,6] \approx c_{12} + c_{22} = [1,4] + [3.5,3] = [4.5,7] \\ x_3 = [6,8] \approx c_{11} + c_{22} = [2,3] + [3.5,3] = [5.5,6] \end{cases}$$

۴- فاصله با پرس و جو:

$$q = [7,10]$$

مدل CQ برای فاصله:

$$\|q-(c_1+c_2)\|^2 = \|q\|^2 + \|c_1\|^2 + \|c_2\|^2 + 2c_1^T c_2 - 2q^T c_1 - 2q^T c_2$$

$\|q\|^2 = 7^2 + 10^2 = 149$; $\|c_{11}\|^2 = 2^2 + 3^2 = 13$

$\|c_{12}\|^2 = 1^2 + 4^2 = 17$; $\|c_{21}\|^2 = 3^2 + 4^2 = 25$; $\|c_{22}\|^2 = 3.5^2 + 3^2 = 21.25$

$q^T c_{11} = 7\times 2 + 10\times 3 = 44$; $q^T c_{12} = 7\times 1 + 10\times 4 = 47$

$q^T c_{21} = 7\times 3 + 10\times 4 = 61$; $q^T c_{22} = 7\times 3.5 + 10\times 3 = 54.5$

ضرب داخلی ها تقریبا ثابت $c_1^T c_2 \approx 18$

□ محاسبه فاصله ها با CQ

۱- ترکیب $c_{11} + c_{21} = [5,7]$

$$\|q - [5,7]\|^2 = 149 + 13 + 25 + 2\times 18 - 2(44+61) = -34 \approx 13$$

۲- ترکیب $c_{21} + c_{22} = [4.5,7]$

$$d = 149 + 17 + 21.25 + 2\times 18 - 2(47 + 54.5) = 17.25$$

تمرین

Vector Quantized Variational Autoencoder (VQ-VAE)

- The encoder produces a continuous latent vector
- Quantizer maps it to the nearest codebook vector.
- The decoder reconstructs the input from this discrete vector
- Loss has three parts: reconstruction, codebook update, and commitment
- This structure allows discrete latent representation while maintaining good reconstruction quality.

Example: Input : $x = [1,5, 2,7]$; $\text{codebook} = \{e_1 = [1,2], e_۲ = [2,3]\}$

Encoder output: $z_e(x) = [1,5, 2,7]$

(درعمل)

$$\begin{cases} \text{nearest codebook vector } e_۲ = [2,3] \\ \beta = 0.25 \end{cases}$$

VQ Loss $\Rightarrow$ $L_{vq} = \| sg[z_e(x)] - e_2 \|^2 = \| [-0.5, -0.3] \|^2 = 134$

Commitment Loss $\Rightarrow$ $L_{commit} = \beta \| z_e(x) - sg[e_۲] \|^2 = /25 \times 134 = ۰/۰۸۵$

VQ-GAN مرور

• VQ-GAN architecture has four main components

1. Encoder E_θ ⟹ Maps input x to a latent vector $z_e(x)$
2. Vector Quantizer ⟹ Maps $z_e(x)$ to the nearest codebook vector $e_k \rightarrow z_q(x)$
3. Decoder D_Φ ⟹ Reconstructs $\hat{x} = D_\Phi(z_q(x))$
4. Discriminator D ⟹ Tries to distinguish real images from reconstructed/generated images

VQ-GAN combines three types of losses:

1. Reconstruction loss (L_1 / L_2)

$$L_{recon} = \|x - \hat{x}\|_1 \quad \text{or} \quad \|x - \hat{x}\|_2^2$$

2. VQ-VAE loss (codebook + commitment)

$$L_{VQ} = \|sg[z_e(x)] - e_k\|^2 + \beta \|z_e(x) - sg[e_k]\|^2$$

3. Adversarial loss (GAN loss)

For discriminator D: $L_D = -E[\log D(x)] - E[\log(1 - D(\hat{x}))]$

For generator (decoder): $L_{GAN} = -E[\log D(\hat{x})]$

Total generator loss: $L_G = L_{recon} + L_{VQ} + \lambda L_{GAN}$

Residual Quantized VAE

رزی

. Standard VQ-VAE uses a single discrete latent vector (from a codebook) to approximate the continuous latent.

. This can limit representation capacity for complex data, because a single codebook vector may not capture all fine details.

. Solutions: Residual Quantization

. Instead of a single codebook, use multiple codebooks in sequence, where each codebook encodes the residual error left by the previous quantization.

. This is inspired by Additive Quantization (AQ) in vector quantization.

Idea => Approximate latent as a sum of discrete vectors from multiple codebooks

$$Z_q \approx e_1 + e_2 + \dots + e_M$$

e_2 → from the second

e_1 → comes from the first codebook

. Each codebook encodes what the previous codebooks failed to represent (the residual).

مراحل الگوریتم =>

1 Encoder => Maps input x to a latent vector $Z_e(x)$.

2 Residual Quantizer

. step1: find nearest vector in first codebook: $e_1 = \arg\min \| Z_e - e_1^k \|$

. step2: Compute residual: $r_1 = Z_e - e_1$

. step3: Quantize residual with second codebook: $e_2 = \arg\min \| r_1 - e_2^k \|$

. Repeat for M codebooks. => $Z_q = e_1 + e_2 + \dots + e_M$

3 Decoder

- Reconstructs input from z_q: $\Rightarrow \hat{x} = D_\Phi(z_q)$

Loss Function

Similar to VQ-VAE but applied across all residual codebooks:

$$L = L_{recon} + \sum_{i=1}^{M} (\| sg[r_{i-1}] - e_i \|^2 + \beta \| r_{i-1} - sg[e_i] \|$$

- $r_0 = z_e$
- $r_i = r_{i-1} - e_i$
- Each codebook has its codebook loss and commitment loss

Benefits:

- Increased representation capacity with multiple codebooks
- Approximation error decreases with more codebooks

Toy Example

- Encoder output $z_e(x) = [2.3, 3.7]$
- 2 codebooks $(M=2) \Rightarrow C_1 = \{ [2,3], [3,4] \}$, ~~$C_2 = \{ [0,2$~~

$C_2 = \{ [.2, .5], [.5, .7] \}$

- compute distances:

$\| [2.3, 3.7] - [2,3] \|^2 = (0.3)^2 + (0.7)^2 = 0.09 + .49 = .58$

$\| [2.3, 3.7] - [3,4] \|^2 = (-.7)^2 + (-0.3)^2 = .49 + .09 = .58$

- Pick $e_1 = [2,3]$
- Residual: $r_1 = z_e - e_1 = [0.3, 0.7]$

Step 2: Second codebook quantization

- Quantize residual r_1 with C_2:

$$\| [0.3, 0.7] - [0.2, 0.5] \|^2 = (0.1)^2 + (0.2)^2 = 0.05$$

$$\| [0.3, 0.7] - [0.5, 0.7] \|^2 = (-0.2)^2 + 0^2 = 0.04$$

- Pick $e_2 = [.5, .7]$

- Residual after second codebook $\Rightarrow r_2 = r_1 - e_2 = [0.3, 0.7] - [0.5, 0.7]$
$= [-0.2, 0]$

Step3: Quantized latent

$$Z_q = e_1 + e_2 = [2, 3] + [0.5, 0.7] = [2.5, 3.7]$$

- True latent $z_e = [2.3, 3.7]$
- Approximation error: $Z_e - Z_q = [-0.2, 0] \Rightarrow$ much smaller than using only one codebook.

Step4: Losses

$L_1 \leftarrow$ codebook 1 loss $\Rightarrow \| sg[z_e] - e_1 \|^2 = \| [2.3, 3.7] - [2, 3] \|^2$
$= 0.58$

$L_2 \leftarrow$ Commitment 1: $\beta * 0.58$

$L_3 \leftarrow$ Codebook 2 loss: $\| sg[r_1] - e_2 \|^2 = \| [0.3, 0.7] - [0.5, 0.7] \|^2$
$= 0.04$

$L_4 \leftarrow$ commitment 2: $\beta * 0.64$

$$\text{Total loss} = L_1 + L_2 + L_3 + L_4$$