Vector Quantized AutoEncoder を用いた

教師なしクラスタリング手法の提案

作成者　下村晏弘

作成日　8/25/2021

1．概要

ここでは、Vector Quantized AutoEncoder を用いた教師なしクラスタリング手法を提案する。2．提案 でモデル定義の大枠や距離の測り方を提案し、 3．実験 で MNIST 手書き数字データセットを使用した具体的な方法・事例を示す。またそれらの実験から考えられることを 4．考察 にて書いている。

本提案は AutoEncoder を学習しながらクラスタリングをしたいという目的から行われている。

本提案は以下の URL に実験する際のソースコードが載っている。

ソースコード： https://github.com/Geson-anko/VQ_AutoEncoder

2．提案

2．1．提案目的

この提案は、VQ_VAE（7．引用 1）の Encoder の出力を用意した重みによって量子化するという点に触発されている。

以下に提案目的を示す。

・入力データのみを用いた教師なし学習によって、入力データにクラスラベルを与えるため。

・AutoEncoder の学習と同時に行うため。またそれにより再学習を容易にするため。

・教師なし学習クラスタリングの既存手法としては K-means 法が存在するが、重心を移動させるという更新方式のため損失関数に組み込みにくいため。

・ユークリッド距離やコサイン距離など様々な距離の概念を使用可能にするため。

また、本提案は VQ_VAE を参考にして行っているため、クラス数を量子化数と呼ぶ。

2．2．手法

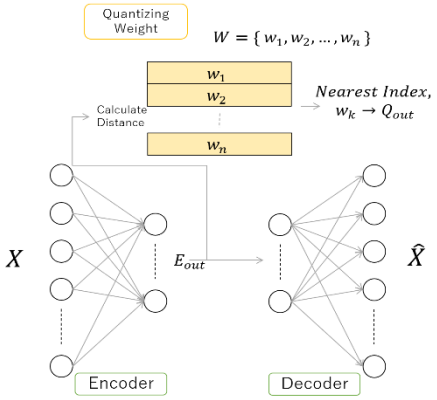

図 1 基本的なモデル

2．2．1．用語解説

X ：入力データ

E_{out} ：Encoder の出力

$\hat{X}$ ：出力データ。入力データと同形状。

W ：量子化重みのこと。E_{out}と同次元数のベクトルを n 個持つ 2 階テンソル

w_k ：Wの中の k 番目のベクトル。またこれを量子化ベクトルと呼ぶ。

Q_{out} ：E_{out}ともっとも近い量子化ベクトル。

Quantizing Weight ：量子化重みWのこと。

2．2．2．手順

１．AutoEncoder を用意。

２．量子数×Encoder の出力次元数 の量子化重みWをもつ量子化層を用意。

３．量子化層の重みの分布を Encoder の出力分布に合わせる必要があるため、実際のデータセットからランダムに取得して Encoder に通し、それを初期重みとする。

４．AutoEncoder の再構成誤差と、E_{out}とQ_{out}との距離を、最小化するように学習する。

２．２．３．量子化ベクトルQ_{out}を選ぶ例

距離の測り方によってQ_{out}の選ばれ方は変わるため、いかに２つの例を示す。

・ユークリッド距離（ソースコード：quantizing_layers.py の Quantizing クラス）

$$Q_{out} = \underset{w_i \in W}{\operatorname{argmin}}(\|E_{out} - w_i\|)$$

・コサイン距離（ソースコード：quantizing_layers.py の Quantizing_cossim クラス）

$$Q_{out} = \underset{w_i \in W}{\operatorname{argmax}} \{ CosineSimilarity(E_{out}, w_i) \}$$

$$CosineSimilarity(x, y) = \frac{x \cdot y^T}{\max (\|x\| \cdot \|y\|^T, \epsilon)} \ (\epsilon = 1 \times 10^{-8})$$

２．２．４．損失の定義

$$Loss = ReconstructionError(X, \hat{X}) + QuantizingError(SG(E_{out}), Q_{out})$$

$$ReconstructionError(X, \hat{X}) = \frac{\|X - \hat{X}\|^2}{X_{dim}}$$

X_{dim}　：入力データの次元数。

SG　：Stop Gradient の略。重み更新のための勾配計算を停止する。

$ReconstructionError$（再構成誤差）は平均二乗誤差で良いが、$QuantizingError$（量子化誤差）は量子化する際の距離の測り方によって変わる。以下に例を示す。

・ユークリッド距離

$$QuantizingError(E_{out}, Q_{out}) = \ \| SG(E_{out}) - Q_{out}\|$$

・コサイン距離（最小距離を 0 にしている）

$$QuantizingError(E_{out}, Q_{out}) = - CosineSimilarity(SG(E_{out}), Q_{out}) + 1$$

$$CosineSimilarity(x, y) = \frac{x \cdot y^T}{\max(\|x\| \cdot \|y\|^T, \epsilon)} \ (\epsilon = 1 \times 10^{-8})$$

3．実験

3．1．データセット

本実験では、MNIST 手書き数字データセット（7．引用2）の Validation データセット 10000 枚を使用した。またモデルに入力する際にはランダムに順番を並び替えながらミニバッチを生成した。

3．2．モデルの作成

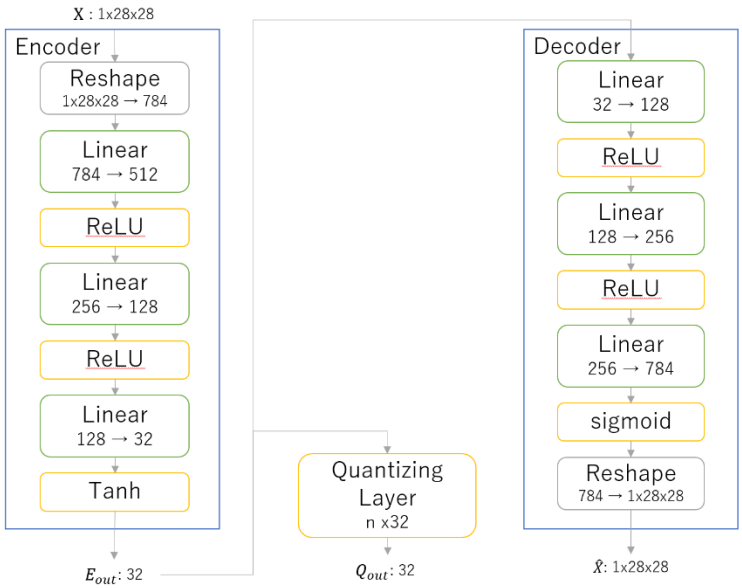

図 2　モデル

Quantizing Layer の n は量子化数。

入力：X，出力：$E_{out}, Q_{out}, \hat{X}$

ソースコード：MNIST_model.py を参照

図 2 に示す通り、3 層の全結合層を持つ Encoder と Decoder を用いて AutoEncoder のモデルを作成した。また Encoder の出力E_{out}は量子化層（Quantizing Layer）にも通され、量子化ベクトルQ_{out}が出力されるようにした。

3．3．実験手順

本実験は Python3.8.10、PyTorch1.9 を使い行った。また適宜ライブラリなどを使用して効率化を図った。

すべての実験において最適化アルゴリズムは Adam を使用し学習率は 0.001 にした。学習率以外は Pytorch のデフォルトの値を使用した。バッチサイズは 1024 とし、データセットから取り出した際にバッチサイズに満たない場合は学習に使わなかった。(DataLoader の drop_last=True)

すべての実験において以下の項目を記録した。

・1 STEP ごとに記録したもの（MNIST_model.py の VQ_AutoEncoder.training_step を参照）

損失（$Loss$）、 再構成誤差、量子化誤差

再構成量子化誤差　　：入力Xと、Q_{out}を Decoder に通した出力との平均二乗誤差

・10EPOCH ごとに記録したもの（MNIST_model.py の VQ_AutoEncoder.on_epoch_end を参照）

入力画像、再構成画像、

量子化再構成画像　　：Q_{out}を Decoder に通した出力画像

量子化ヒストグラム　　：量子化ベクトルそれぞれのデータ割り当て数をカウントしたヒストグラム

・学習ごとに記録したもの
VQ_AutoEncoder 全体の重みパラメータ

3．3．1．実験1　量子化層の重みの初期化法について

1．量子化ベクトルと損失の取り方を、2．2．で提案したユークリッド距離にした。
・未学習初期化（Encoder の重みは乱数で初期化されている）
2．データセットからランダムに取得し、Encoder に通したものを初期量子化重みとした。
3．学習世代数(EPOCH)を 500 にして学習させ結果を TensorBoard に記録した。

・学習済初期化（Encoder の重みは既に訓練されている）
4．2 で 500EPOCH 学習させた Encoder の重みを使い、1 と同様に量子化重みを初期化した。
5．500EPOCH 学習させ、結果を TensorBoard に記録した。

6．未学習と学習済のそれぞれの初期重みの平均と標準偏差を、3 回初期化して計測し記録した。
7．未学習初期化と学習済初期化の 2 つの結果を比較した。

3．3．2．実験2　量子化数の変化

1．量子化ベクトルと損失の取り方を、2．2．で提案したコサイン距離にした。
2．実験 1 の 2 で 500EPOCH 学習させた Encoder、Decoder の重みを使用した。
3．データセットからランダムに取得し、Encoder に通したものを初期の量子化重みとした。
4．量子化数を 8 にし、100EPOCH 学習させた。
5．使われている量子化ベクトルの数と学習結果を TensorBoard に記録した。
6．量子化数を 32, 128, 512, 2048, 8096 に変えて、2~5 を繰り返した。
7．それぞれの結果を比較した。

3．3．3．実験3　コサイン距離とユークリッド距離の比較

1．量子化ベクトルと損失の取り方を、2．2．で提案したコサイン距離にした。
2．実験 1 の 2 で 500EPOCH 学習させた Encoder、Decoder の重みを使用した。
3．データセットからランダムに取得し、Encoder に通したものを初期の量子化重みとした。
4．量子化数を 32 にし、100EPOCH 学習させた。
5．使われている量子化ベクトルの数と学習結果を TensorBoard に記録した。
6．量子化数を 128, 512 にして 2~5 を繰り返した。
7．実験 2 の量子化数が 32, 128, 512 の場合と比較した。

3．4．結果

3．4．1．実験1　量子化層の重みの初期化法について

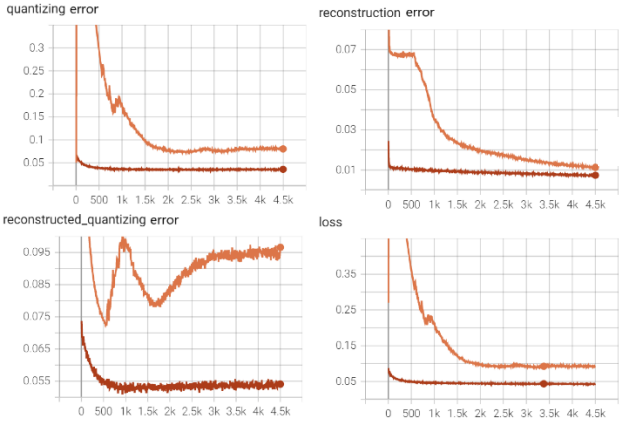

図 3　誤差・損失グラフ

未学習（橙色）、学習済（茶色）

横軸：STEP 数、縦軸：値

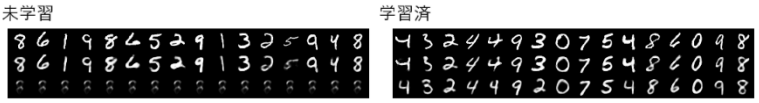

図 4 出力画像

上から、入力画像X、再構成画像$\hat{X}$、量子化再構成画像。

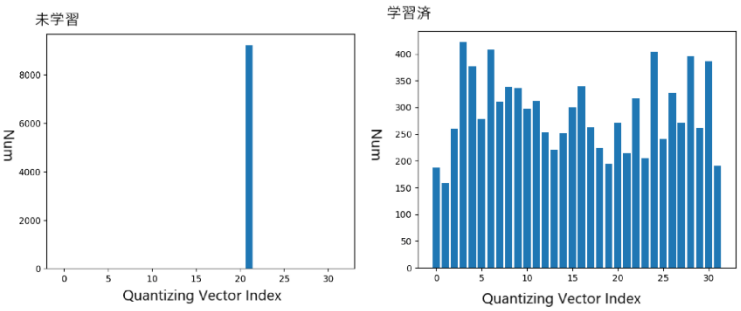

図 5 量子化ヒストグラム

表 1　最終誤差・損失　すべての値は小さい方が良い。

ケース　＼　値	量子化誤差	再構成誤差	再構成量子化誤差	損失
未学習	0.08034	0.01134	0.09655	0.09168
学習済	0.03612	0.007367	0.05409	0.04349

表 2 平均と標準偏差

項目　＼　回数	1回目	2回目	3回目	平均値
未学習：平均	0.0130	-0.0021	0.0023	0.0058
学習済：平均	-0.0102	-0.0220	-0.0349	-0.0224
未学習：標準偏差	0.0729	0.0648	0.0611	0.0663
学習済：標準偏差	0.7000	0.7011	0.7009	0.7007

表1より、すべてにおいて学習済の方の値が小さかった。

図3より、未学習の量子化誤差と再構成量子化誤差は大きなバタつきがあった。

図4より、未学習はQ_{out}を Decoder に通しても数字のような画像は生成されなかったが、学習済は数字と認識可能なものを生成した。

図5より、未学習は一つの量子化ベクトルにクラスタリングされていたが、学習済は全ての量子化ベクトルに一様にクラスタリングされた。

表2より、未学習と学習済の平均と標準偏差は大きく離れていた。

3．4．2．実験２　量子化数の変化

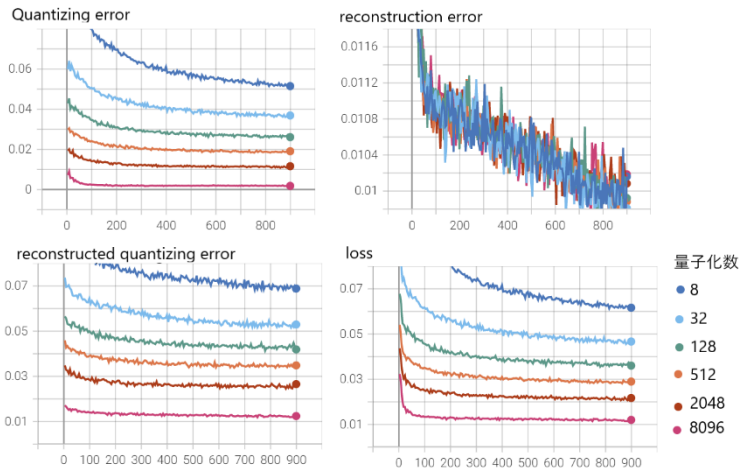

図 6　誤差・損失のグラフ
横軸：STEP 数，　縦軸：値

図 7　出力画像
上から、入力画像X、再構成画像$\hat{X}$、量子化再構成画像

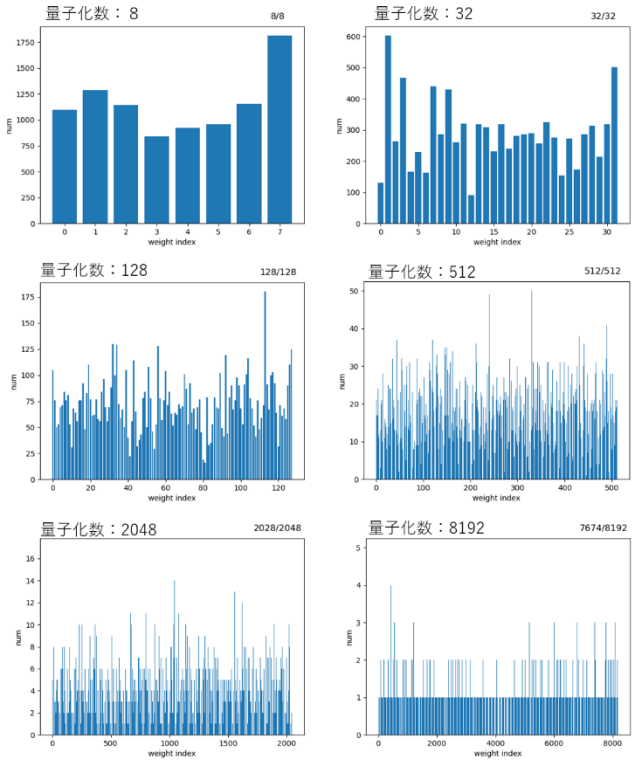

図 8　量子化ヒストグラム
Weight Index は Quantizing Vector Index

表 3　最終誤差・損失　誤差・損失は小さい方が良い。

量子化数＼値	量子化誤差	再構成誤差	再構成量子化誤差	損失	使用数／量子化数
8	0.0514	0.0102	0.0688	0.0617	8/8
32	0.0369	0.00981	0.0529	0.0467	32/32
128	0.0261	0.00992	0.0419	0.0360	128/128
512	0.0191	0.00988	0.0349	0.0290	512/512
2048	0.0116	0.0101	0.0266	0.0217	2028/2048
8192	0.00183	0.0101	0.0124	0.0120	7674/8192

表 3 より、量子化誤差と再構成量子化誤差は量子化数を大きくすると単調に小さくなった。

表 3 より、量子化数を 2048、8192 にした場合は使用数が量子化数に満たなかったため、使われない量子化ベクトルが存在した。

表 3 より、量子化数を大きくしていくと、再構成量子化誤差は再構成誤差に近づいていった。

図 7 より、量子化数が 8 の時の量子化再構成画像は 0 のような画像が 5 になっていた。

図 7 より、量子化数を 512 より大きくすると、入力画像とほとんど変わらない量子化再構成画像が出力された。

3．4．3．実験3　コサイン距離とユークリッド距離の比較

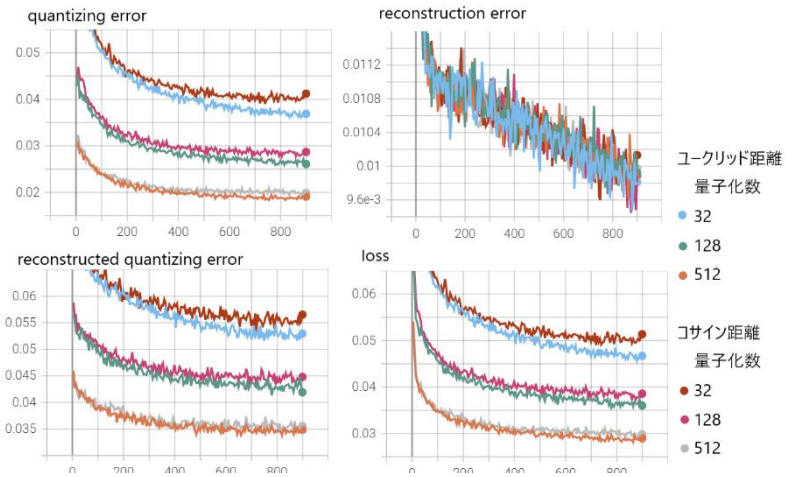

図 9　誤差・損失のグラフ

横軸：STEP 数，　縦軸：値

図 10　出力画像

上から、入力画像X、再構成画像$\hat{X}$、量子化再構成画像

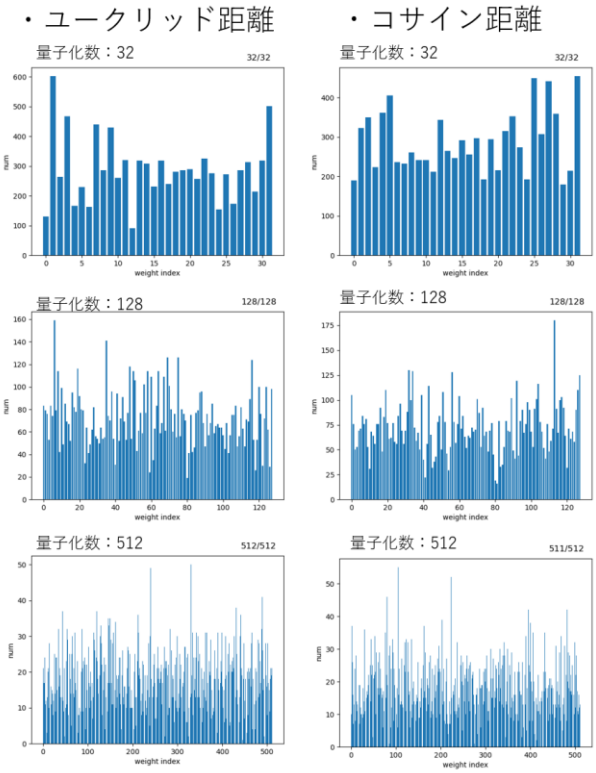

図 11　量子化ヒストグラム

Weight Index は Quantizing Vector Index

表 4　最終誤差・損失　誤差・損失は小さい方が良い。

量子化数＼値	量子化誤差	再構成誤差	再構成量子化誤差	損失	使用数／量子化数
ユークリッド距離					
32	0.0369	0.00981	0.0530	0.0467	32/32
128	0.0261	0.00992	0.0419	0.0360	128/128
512	0.0191	0.00988	0.0486	0.0290	512/512
コサイン距離					
32	0.0412	0.0101	0.0565	0.0514	32/32
128	0.0287	0.00989	0.0441	0.0386	128/128
512	0.0199	0.00997	0.0356	0.0299	512/512

※量子化誤差と損失は距離の定義が違うため比較することができない。

表 4 より、ユークリッド距離とコサイン距離との違いによる、誤差や損失の値に大きな変化は見られなかった。

図 10 より、コサイン距離の量子化数が 512 の場合において、入力画像が 7 なのに対し、量子化再構成画像は 0 となっていた箇所があった。

4．考察

4．1．実験1　量子化層の重みの初期化法について

学習済の方が未学習に比べて量子化誤差が小さく量子化ヒストグラムも一様に分布しているため、ある程度同じデータセットで学習させた Encoder、Decoder の重みを使用して量子化重みを初期化した方が良いと考えた。

また未学習の量子化ヒストグラムが偏った理由としては、学習初期の急激な重みの更新によって、Encoder の出力分布が初期の量子化重みの分布と大きく離れたことが原因だと考えた。

学習済の方がうまく量子化ベクトルを獲得し、クラスタリングできたと考えた。

4．2．実験2　量子化数の変化

量子化数を大きくすると使用数が量子化数に満たなくなるため、量子化数はデータ全体に対して適切な値が存在すると考えた。

量子化数を大きくすると量子化ベクトル 1 つに対して近いデータが 1 つや 2 つ程度になるため、量子化ベクトルがデータそのもののベクトルを表すと考えた。よって再構成量子化誤差が再構成誤差に近づいたと考えた。量子化誤差が量子化数を増やすと単調に減少した理由も同様だと考えた。

量子化数が 8 のときに 0 の画像の量子化再構成画像が 5 の画像に近かった理由として、10 個の数字クラスに対し 8 クラスと満たないため、距離が近いデータは同じクラスに丸め込まれたためと考えた。

4．3．実験3　コサイン距離とユークリッド距離の比較

結果に大きな違いは存在しないため、コサイン距離に変えてもクラスタリングとして使うことが可能だと考えた。

5．まとめ

本提案では、Vector Quantized AutoEncoder による教師なしクラスタリング手法を提案した。（2．）その過程で、重みの初期化法（3．3．1．）、量子化数の変化（3．3．2．）、距離の定義の違いについて（3．3．3．）それぞれ実験し、考察した。

実験を通し、本手法はデータセットの傾向からクラスタリングすることができると考えた。

提案目的のほかにも、AutoEncoder を作成しつつデータに対し傾向から ID を割り当てたいといったタスクに有効だと考えた。

6．用語解説

EPOCH　　：学習世代数。学習に用いられるデータセットすべてを使った回数。
STEP　　：更新回数。重みを更新した回数。
量子化　　：連続値を離散値に変えること。本提案では n 個のベクトルから 1 つを割り当てること。

7．引用

1：VQ_VAE　　https://arxiv.org/abs/1711.00937
2：MNIST 手書き数字データセット　https://pytorch.org/vision/stable/datasets.html#mnist