Variable Autoencoders in the Unsupervised Data Compression and Inference from Discovery-based Quantitative Proteomic Datasets

Journal:	*Bioinformatics*
Manuscript ID	Draft
Category:	Original Paper
Date Submitted by the Author:	n/a
Complete List of Authors:	Nouri Nigjeh, Benjamin; University of Washington, Department of Medicine
Keywords:	Inference, Discovery, Quantitative, Proteomic, Datasets

Variable Autoencoders in the Unsupervised Data Compression and Inference from Discovery-based Quantitative Proteomic Datasets

Benjamin Nouri Nigjeh

Department of Medicine, University of Washington at Seattle, WA 98195

nigjeh@uw.edu

Abstract:

Data compression and inference from high-dimensional proteomic datasets is a daunting task due to the complex nature of the proteomic samples based on their origin (organs, tissues, cell-types, disease status, population diversity, therapeutic history, etc.) and variations occurring during sample collection, storage, processing, and LC/MS analysis. Hence, a solid approach is required to reduce the high dimensional complexity of the proteomic datasets to monitor the key characteristics of the samples in an easy to visualize way. In this study, *variational autoencoders* (VAEs) are utilized for nonlinear proteomic dataset compression to only two variables in the latent space layer. These variables are the mean and standard variation from a Gaussian distribution estimated by Kullback-Leibler divergence function. A fully connected VAE was opted based on its training speed and k-means silhouette score on clustering and generative quality. The impact of data quantity and quality on training datasets were measured by Euclidean distance between cluster centers generated from training partial mnist datasets, and random noise datasets with the addition of limited features. The fully connected VAE was used to nonlinear data compression of a 10-plex TMT pancreatic tissue dataset (n=54) with 7699 quantified proteins which showed successful unsupervised clustering of cancer tissue samples from healthy tissue controls. The trained algorithm was able to infer the test cancer tissue samples correctly which suggest its generalization ability and inference power. Finally, the fully connected VAE was trained to compress super SILAC breast tissue samples (n=242) with 6703 quantified proteins from three different studies. The samples were clustered based on their original study, highlighting the importance of study design on the downstream heuristic designs.

Introduction:

Proteomic samples are extraordinary complex not only by the nature of their origin (organs, tissues, cell-types, therapeutic history, and population diversities), but also by the sample collection, storage, transportation, processing, and LC/MS instrumentation [Nouri Nigjeh et. al. 2017] [Suomi et. al. 2022] [Jesse G. Meyer, 2021] [Kim et. al. 2019]. Proteomic datasets contain quantitative information underlying several thousand proteins, wherein each protein and its expression level can be dependent on numerous factors. To unravel the proteomic complexity, it is required to compress the high dimensional proteomic datasets to their essential elements. The reduction in complexity is beneficial for the characterization of the proteomic samples and can be useful for the accurate labeling of samples, data compression, data inference, and data generative purposes.

Variational autoencoders (VAEs) are gaining attention for nonlinear data compression of high dimensional datasets generated from high-throughput metabolomics and transcriptomics studies [Gronbach et. al. 2020] [Wei et. al. 2021] [Gomari et. al. 2022] [Seninge et. al. 2021] [Simidjievski et. al. 2019]. VAEs are unsupervised learner algorithms with a stochastic gradient variational Bayes algorithm in their latent space optimized for inference and generative purposes [Diederik Kingma, and Max Welling. 2014]. VAEs are highly regularized algorithms based on the Gaussian distribution assumption in their latent space, where its loss function is defined by Kullback-Leibler divergence function. The latent space of VAEs is defined with two parameters, namely, the mean value (Z_0) and standard variation (Z_1) of distribution, meaning that the complexity of the high dimensional datasets can be reduced to a single point in a two-dimensional latent space [Pratella et. al. 2021] [Franco et. al. 2021] [Wei et. al. 2021].

In this study, a fully connected variational autoencoder learner was opted to compress and to infer from the proteomic datasets. The algorithm was tested in the light of input sample data quantity and quality. Then, the algorithm was utilized in the unsupervised data compression and inference from TMT-labelled pancreatic tissue samples with quantitative information from 7699 tissue proteins [Pandit et. al. 2022]. Finally, the algorithm was used to compress a proteomic dataset with 6703 quantified tissue proteins from a super SILAC approach collected from three separate studies [Tyanova et. al. 2016] [Pozniak et. al. 2016] [Shenoy et. al. 2020].

Experimental procedure:

Hardware and Software: Deep learner algorithms were trained on an NVIDIA GeForce RTX 3050 GPU system with an Intel Core i5 CPU and 8 GB of RAM. The operating system was Windows 11. The computations on NVIDIA GPU were parallelized with the installation of CUDA/cuDNN. Coding was done on Python version 3.9.7 running on PyCharm IDE version 2022.1. Sequential and functional neural network structures were built using keras API, a high-level frontend module integrated on tensorflow library version 2.8. Data visualizations were done on GraphPad Prism version 5.

Variational autoencoder structures: The initial variational autoencoder was a model example written to reconstruct mnist dataset presented by Francios Chollet [Francis Chollet. 2021] and presented at **code S1**. The initial code architecture included two dimensional convolutional neural networks [LeCun et. al. 2015] both for the encoder and decoder parts of the algorithm. The nonlinearity activation functions were defined by "relu" for the hidden layers and "sigmoid" for the output layer. The loss function was the sum of reconstruction loss and deviation from the gaussian distribution in the latent space using Kullback–Leibler divergence. The training was performed with "adam" function, a stochastic gradient descent function enforced with a momentum mechanism to facilitate the exploration of the minima in the datasets. To monitor the performance of the code and to deliver an optimal structure for the analysis of proteomic datasets, several options were added to the initial prototype. The cluster overlap in the latent space of the VAE was evaluated with unsupervised k-mean clustering (k is equal to 10, i.e., the number of digit groups) and silhouette score from Sckitlearn library [Geron Aurelien. 2017]. Tensorboard callback was added to monitor the loss functions throughout the training cycles, and the elapsed time for training was recorded by datetime function. In addition to the training time, and cluster overlap, the performance of the algorithm with different structures were visualized by plotting the clusters in the latent space (mean value versus standard variation), and reconstructed mnist digits (see **code S1**).

Discovery-based proteomic datasets: Two large-scale proteomic datasets were used to evaluate the performance of the fully connected VAE learner in data compression and unsupervised clustering of the proteomic datasets. The first dataset was originated from 10-plex TMT labeling multiplexed result from quantification of 49 pancreatic cancer tissues, and 5 normal pancreatic tissue. One TMT channel was labeled with pooled sample to normalize the results throughout the consecutive runs. This study resulted in the quantification of 7699 tissue proteins (presented in **Table S1**) [Pandit et. al. 2022]. The second discovery proteomic dataset was originated from three separate studies published based on the quantification of breast cancer tissue proteins with super SILAC technology. The study included quantitative analysis of 6703 proteins from 242 tissue sample combined from cancer subtype analysis [Tyanova et. al. 2016], cancer staging [Pozniak et. al. 2016], and cancer proteomic response to a neoadjuvant therapy [Shenoy et. al. 2020] (presented in **Table S2**). In both of these studies, each protein level normalized to its maximum level, so that the normalized protein levels were always between 0 and 1. Missing data were replaced with the average level of each protein to minimize their impact during data compression.

Result and discussion:

Part1: Fully connected variational autoencoder (VAE) structure

The initial functional VAE algorithm consisted of convolutional neural network encoder and decoder (i.e., *CNN_CNN VAE*) optimized for the reconstruction of mnist digits (**code s1**). To enable this learner algorithm to encompass the proteomic sample complexity, the encoder was replaced by a fully connected neural network which allows full connection of all dataset input entities simultaneously to overcome the high sample complexities from proteomic experiments (i.e., *FCNN_CNN VAE*). Then, the decoder was replaced by a fully connected neural network to compare the benefits of a coherent fully connected neural network variational autoencoder (*FCNN_FCNN VAE*) to the initial convolutional structure. The codes for the fully connected decoders and encoders are presented in **code s2**. For the performance comparison purposes, these three learner algorithms were trained for 10 epochs on mnist dataset, and the total loss during training was monitored. These three learner algorithms showed a similar trend in dropping the loss functions and getting to a plateau phase (**Figure S1**). In addition, the nonlinear compression of the digit clusters in the latent layer space was evaluated by k-means algorithm (k = 10) and silhouette score, and the training time for each algorithm during the training. The latent space clusters and the reconstructed digits after each algorithm are shown in **Figure S2**. Apparently, these three learner algorithms are equally good in generalization and inference from mnist dataset. The calculated silhouette score for these three learner algorithms were 0.39, 0.37, and 0.38, respectively (*CNN_CNN*, *FCNN_CNN*, and *FCNN_FCNN*). And the time elapsed during 10 epoch training was 800, 600, and 40 seconds, respectively. Clearly, *FCNN_FCNN* learner can be trained way faster than the learner algorithms containing convolutional structures. This is a key decision-making parameter when training a large dataset or training for large training cycles. The number of trainable parameters from each layer of selected *FCNN_FCNN* structure is shown in **Figure S3**. the highest number of trainable parameters resides in the decoder structure prior to reshaping of the planar layer to a 2D output data shape. Despite the large number of trainable parameters, the fully connected variational autoencoder was successfully performing well in learning and generalization of the mnist dataset.

The performance of the *FCNN_FCNN* learner algorithm throughout deep clustering experiments were evaluated by the Euclidian distance between experiment and control cluster centers. **Code s2** presents the *FCNN_FCNN VAE* structure with the Euclidian distance performance monitoring function. The binary control and experiment set of samples were data points of 0 and 1 excised from the mnist dataset, and the training performance was monitored based on the separation of cluster centers.

Part 2: The impact of data quantity on VAE performance

There is a need to explore the impact of training dataset size on the performance of the learner algorithm. Thus, The *FCNN_FCNN VAE* learner algorithm was used to cluster mnist datasets excised for 0 and 1 digits at increasing dataset sizes from 10, 20, 50, up to 1000 samples per group. The performance of the algorithm in unsupervised clustering of these datasets were monitored by the Euclidian distance between cluster centers throughout the increasing number of training epochs from 5 to 1000 cycles for 10 training replicates. **Figure 1** shows the average Euclidean distance between cluster centers with increasing epoch for different dataset sizes. The algorithm performance results suggest that algorithm works equally well for the small quantity of samples as little as 10 samples per group. Increasing the training epoch increases the cluster center distances. The increasing distance between cluster center indicates gradual training of the algorithm from the dataset. A high epoch of 500 showed a good and reproducible learning for the studied dataset sizes. **Figure S4A** shows an example from the cluster shapes generated from 1000 sample training dataset after 500 epochs. The *FCNN_FCNN* algorithm is clearly able to separate clusters belonging to the zero and one group regardless of the number of the samples.

An imbalanced dataset containing 10 controls versus 1000 experiment samples were constructed to check the effect of an imbalanced dataset on the learning process. Here, when a data cluster is relatively abundant, it engulfs the smaller size cluster. Even though the clusters are separated (**Figure S4B**) the Euclidean distance can not be a good performance monitoring parameter. The unsupervised VAE learner is a good fit for clustering from datasets with imbalanced number of samples per each group, even though the dataset with a smaller number of samples may be engulfed by the large quantity of samples from the dominant group.

Part 3: The impact of data quality on VAE performance

The unsupervised clustering algorithm evaluates all the features and reduces the high dimensional complexity of the dataset to a two-dimensional planar data in the latent space. This apparently depends on the presence of informative features in the datasets for the data compression purposes. The *FCNN_FCNN VAE* algorithm was trained on a random noise dataset with increasing number of epochs. As shown in **Figure S5A,** the cluster centers didn't separate by increasing the learning cycles, and there was a random distribution of the presumed clusters shown at **Figure S5B**. As a true negative experiment, this highlighted that the learner doesn't train from a completely random noise dataset and the increasing number of training epochs doesn't lead to a tangible training performance. The total loss function during training from the random noise dataset initially showed a drop but then plateaued at levels higher than those mnist dataset. Given this negative control experiment, the Euclidian distance between cluster centers is a superior performance evaluation parameter than the loss functions, as there is a drop in loss function by training with the random noise dataset, while clusters don't form.

The impact of data quality with the presence of meaningful features in the datasets, useful for clustering, was tested by generation of increasing number of features in a random noise dataset with constant signal to noise ratio of six. **Figure 2A** shows the increase of the Euclidian distance of the cluster centers for dataset sizes of 100 controls versus 100 experiments after 500 epochs with increasing number of features in the random noise dataset for 10 analysis replicates. It is evident that the unsupervised learner algorithm requires more than 14 features with signal to noise ratio of 6 to distinguish the clusters when the size of dataset is 200 samples in total. The same experiment was repeated when then size of dataset increased to 2000 (**Figure 2B**). Here, learner algorithm can distinguish control and experiment samples from merely more than 2 features. This experiment highlights the importance of training from large datasets when the datasets are poor in feature quality and suffer from background noise. Thus, training from big datasets circumvents the poor quality of the data.

Part 4: Training VAE learner algorithm from discovery-based quantitative proteomic datasets

The Fully connected learner algorithm was used for clustering of pancreatic biopsy tissue samples collected from 49 patients versus 5 healthy objects (n=54). From 49 patient samples 5 of them were separated as the test dataset to assess the inference ability of the trained algorithm. 10-plex TMT global quantification of these samples resulted in the quantification of 7699 tissue proteins (see experimental section). The dataset was imbalanced and contained far less control tissue samples (44 versus 5). The tabular quantification data was given to the learner algorithm without sample labels or highlighting any feature. Since the number of entities (quantified proteins) per sample were increased by a factor of 9.8-folds (in compared to mnist dataset), the number of trainable parameters were increased by the same ratio. Thus, there was a need to tighten the regularization elements to avoid overfitting the data, and to improve the inference power of the algorithm. Therefore, the Kullback-Leibler divergence loss function was given a factor of 10 to enhance the generalization and to improve the learning process. The algorithm was trained at increasing number of epochs and a learning curve was observed by increasing epoch number (**Figure 3A**). The unsupervised learner algorithm clearly separated cancer samples from healthy controls after 500 epochs (**Figure 3B**). From these unsupervised experiments it is evident that the algorithm is learning from the dataset and can compress the dataset.

Using the trained wights of the encoder part of this learner algorithm, it is possible to infer the unseen test dataset to check if they belong to the existing clusters or they appear at different space. Therefore, the algorithm was trained on the training dataset, and the weights were saved. After loading the saved weights, the encoder predicted the latent space coordination of the test dataset. **Figure 3C** shows the coordination of the test dataset predicted by the encoder. The test dataset is shown to reside in the same latent space as the cancer samples.

The experiments from clustering of TMT-labelled pancreatic tissue samples highlights the successful nonlinear compression of proteomic datasets, and its utilization for the inference purposes. The trained algorithms from large proteomic datasets can be useful further for data generative purposes and discovery of vectors in hidden latent space corresponding to the key biological pathways. But this will require training from large quantity of samples to shed light upon the latent space, thoroughly. Yet, having a large proteomic dataset is particularly challenging as the sizes of analyzed samples are limited and analyses are divided into separate studies with a variety of sample processing and data acquisition instrumentations. Here, we used results from a multisite breast cancer study that are linked together with super SILAC global internal standard technology to generate a larger proteomic dataset.

Super SILAC quantification of 242 breast tissue samples with 6703 proteins were combined from three separate studies to evaluate the proteomic differences based on the cancer subtype (n= 40), cancer stage (n= 88), and response to a neoadjuvant therapy (n= 114). The tabular dataset, after normalization, was given to the fully connected VAE learner algorithm for 500 training epochs. The shape of the clusters generated in the latent space layer is presented at **Figure 4**. The unsupervised clustering of the proteomic dataset apparently shows clustering based on the belonging of proteomic samples to the original study. This suggests the profound effect of proteomic study design on the characteristics of the analyzed samples. In this example, the learner algorithm successfully trained from the proteomic dataset collected from separate studies, and highlighted the importance of study design on the downstream heuristic purposes.

Conclusions:

A fully connected variational autoencoder learner algorithm was opted based on its nonlinear data compression and inference power and learning speed from mnist dataset. Then, the training ability of the learner algorithm from datasets with increasing number of data quantity and quality was examined. The learner algorithm can be trained from datasets with small size data, but for low quality data the quantity of dataset is important for a successful training process. The fully connected learner algorithm was used for data compression and inference from 10-plex TMT quantified pancreatic tissue samples with > 7000 quantified proteins from disease versus control healthy samples. The learner algorithm showed success in the unbiased clustering of the disease versus control samples using the whole proteomic dataset. The trained algorithm was successful in the inference from the test cancer dataset. Finally, the learner algorithm was used for mapping 242 breast tissue sample quantified using super SILAC technology containing quantification information from 6703 proteins originated from three separate studies. The unsupervised learner algorithm formed three distinct clusters that each contained samples originated from each of the three studies. This showed the importance of study design on the proteomic characteristics of the samples. In addition to nonlinear data compression and inference power, fully connected variational autoencoders can be useful for data generative purposes, and discovery of key biological pathways if the algorithm is trained by a sufficiently big proteomic dataset.

References:

Francois Chollet. Deep Learning with Python. MANNING. 2021. 2nd Edition. ISBN:1617296864

Franco et. al. Performance Comparison of Deep Learning Autoencoders for Cancer Subtype Detection Using Multi-Omics Data. Cancers. 2021, 13 (2013).
doi.org/10.3390/cancers13092013

Aurélien Géron. Hands-On Machine Learning with Scikit-Learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'REILY. 2017. 3rd Edition. ISBN:1491962291

Gomari et. al. Variational autoencoders learn transferrable representations of metabolomics data. Communications Biology. 2022, 5(645).
doi.org/10.1038/s42003-022-03579-3

Grønbech et. al. scVAE: variational auto-encoders for single-cell gene expression data. Bioinformatics. 2020. 36(16): 4415–4422
doi.org/10.1093/bioinformatics/btaa293

Kim et. al. Clinically Applicable Deep Learning Algorithm Using Quantitative Proteomic Data. Journal of Proteome Research. 2019, 18(8): 3195–3202.
doi.org/10.1021/acs.jproteome.9b00268

Diederik P. Kingma, and Max Welling. Auto-Encoding Variational Bayes. Computing Research Repository, abs. 2014. 1312(6114).
http://arxiv.org/abs/1312.6114

LeCun et. al. Deep learning. Nature. 2015, 521(7553): 436-44.
doi.org/10.1038/nature14539

Jesse G. Meyer. Deep learning neural network tools for proteomics. Cell Reports Methods. 2021, 1(2): 100003
doi.org/10.1016/j.crmeth.2021.100003

Nouri Nigjeh et al. Quantitative proteomics based on optimized data-independent acquisition in plasma analysis. Journal of Proteome Research. 2017, 16(2): 665-676.
doi.org/10.1021/acs.jproteome.6b00727

Pandit et. al. Proteome Analysis of Pancreatic Tumors Implicates Extracellular Matrix in Patient Outcome. Cancer Research Communications. 2022, 2(6): 434-446.
doi.org/10.1158/2767-9764.CRC-21-0100

Pozniak et. al. System-wide Clinical Proteomics of Breast Cancer Reveals Global Remodeling of Tissue Homeostasis. Cell Systems. 2016, 2(3): 172-184.
doi.org/10.1016/j.cels.2016.02.001

Pratella et. al. A Survey of Autoencoder Algorithms to Pave the Diagnosis of Rare Diseases. International Journal of Molecular Sciences. 2021. 22(19):10891.
doi.org/10.3390/ijms221910891

Seninge et. al. VEGA is an interpretable generative model for inferring biological network activity in single-cell transcriptomics. Nature Communications. 2021. 12(5684).
doi.org/10.1038/s41467-021-26017

Shenoy et. al. Proteomic patterns associated with response to breast cancer neoadjuvant treatment. Molecular Systems Biology. 2020, 16(9): e9443
doi.org/10.15252/msb.20209443

Simidjievski et. al. Variational autoencoders for cancer data integration: Design principles and computational practice. Frontiers Genetics. 2019. 10(1205).
doi.org/10.3389/fgene.2019.01205

Tomi Suomi, and Laura L. Elo. Statistical and machine learning methods to study human CD4+ T cell proteome profiles. Immunology Letters. 2022. 245: 8-17.
doi.org/10.1016/j.imlet.2022.03.006.

Tyanova et. al. Proteomic maps of breast cancer subtypes. Nature Communications. 2016, 7(10259): 1-11.
doi.org/10.1038/ncomms10259

Ruoqi Wei and Ausif Mahmood. Recent Advances in Variational Autoencoders with Representation Learning for Biomedical Informatics: A Survey. *IEEE Access*. 2021. 9: 4939-4956.
doi.org/10.1109/ACCESS.2020.3048309.

Figures:

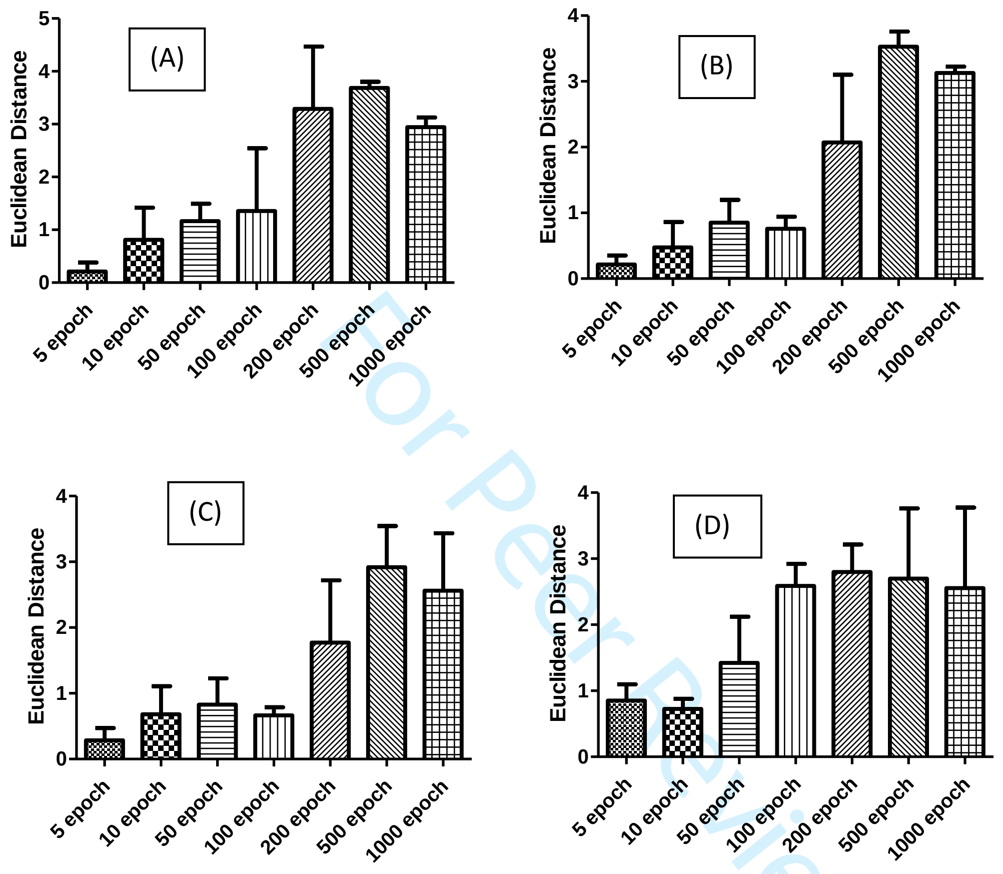


Figure 1. Training of fully connected VAE in clustering two groups of controls versus experiments by increasing size of dataset form 10, 20, 50, up to 1000 per group (A to D) with epoch measuring the Euclidean distance between the cluster centers (10 analysis replicates)

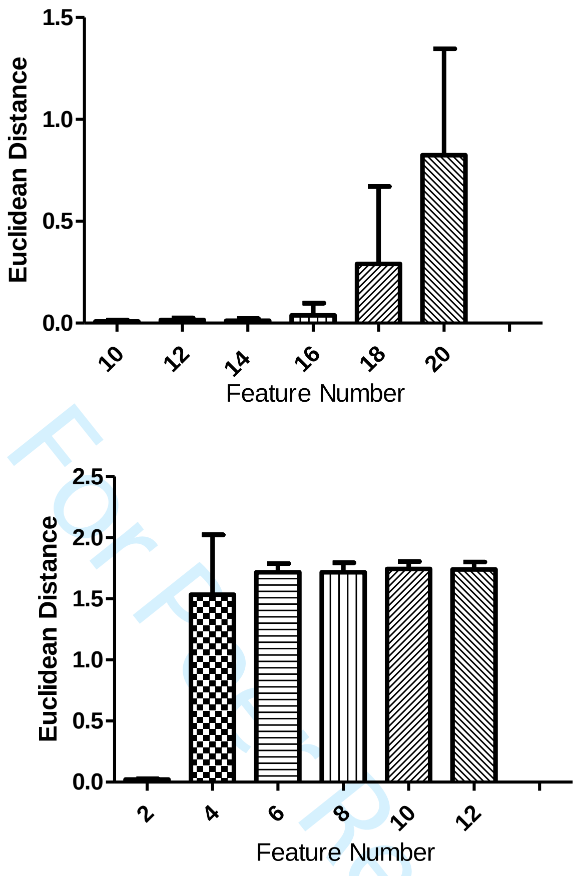


Figure 2. Training of fully connected variational autoencoder from noise datasets with increasing numbers of features with signal to noise ratio of 6 in the case of 100 controls versus 100 experiments (top), and 1000 controls versus 1000 experiments (bottom). The epoch number was 500 and each analysis repeated 10-times.

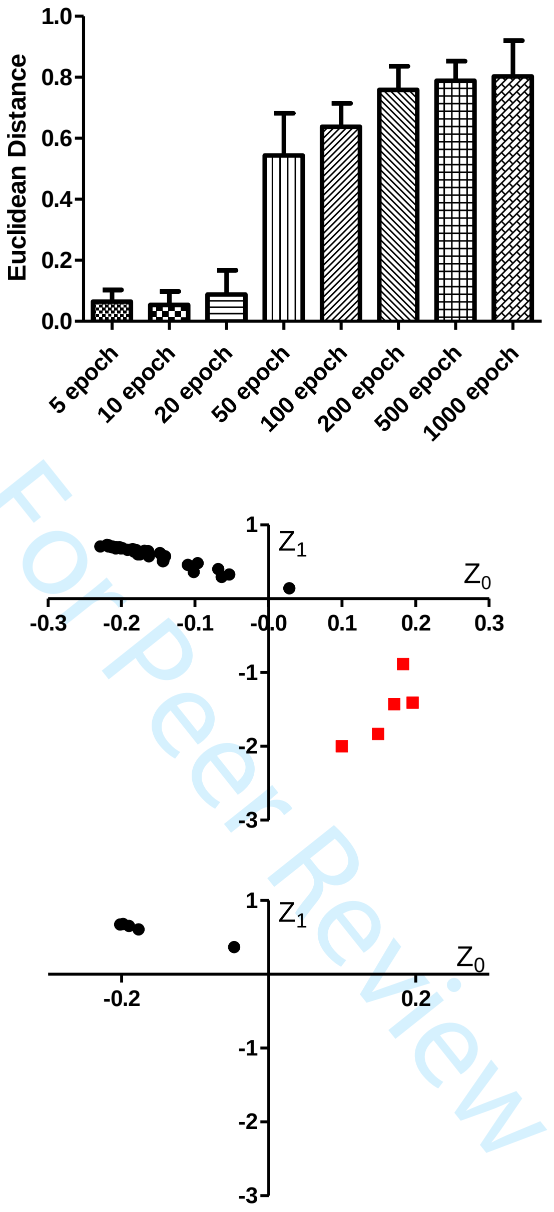


Figure 3. Training of the fully connected variational autoencoder from TMT-labeled proteomic dataset by increasing number of epochs, 10 analysis replicates (top), the latent space separation of the healthy versus control samples (middle), and inference of 5 cancer samples from the test dataset using the encoder weights (bottom).

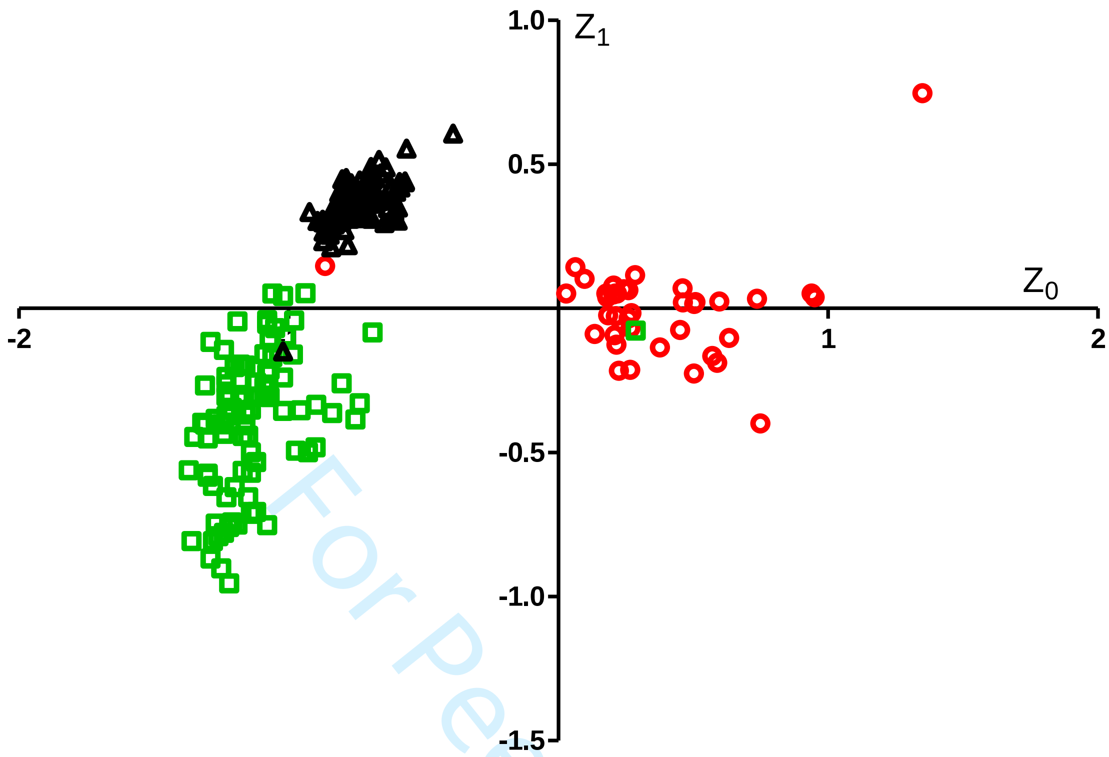


Figure 4. Training of the fully connected variational autoencoder from super SILAC proteomic dataset (n=242) collected from three separate studies (500 epoch). Each color represents the study origin.

Supplementary Figures:

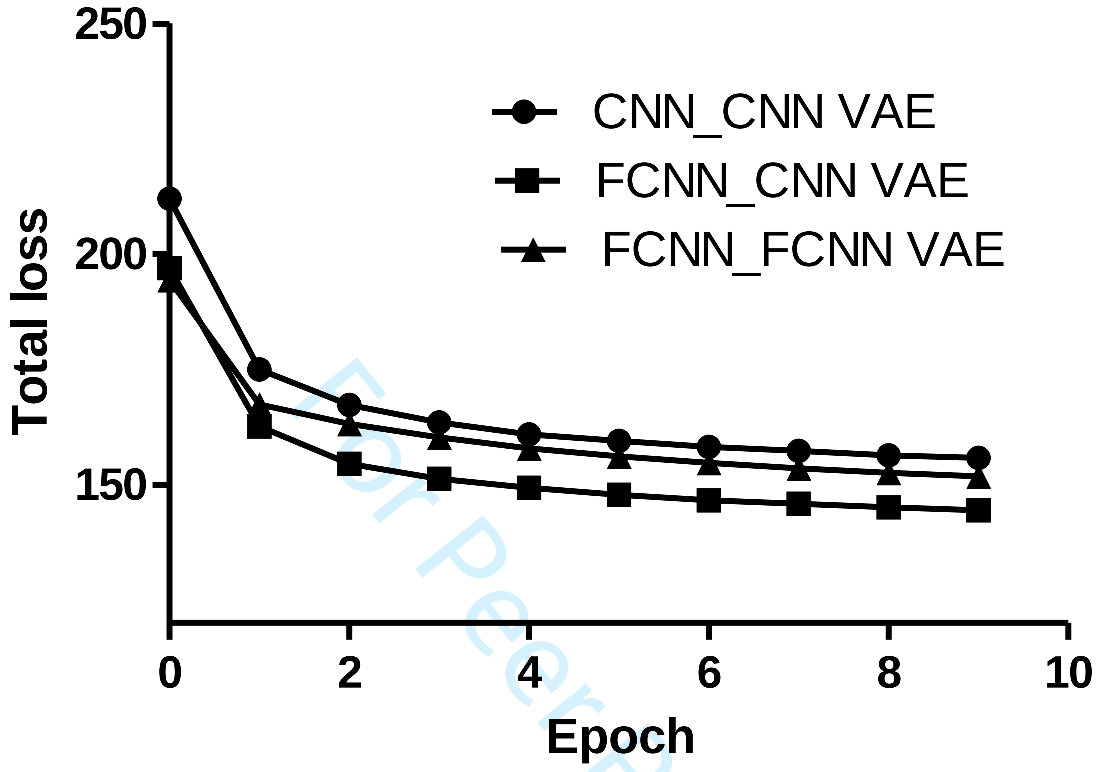


Figure S1. Total loss function drops during training of three different VAE learner algorithms on mnist dataset

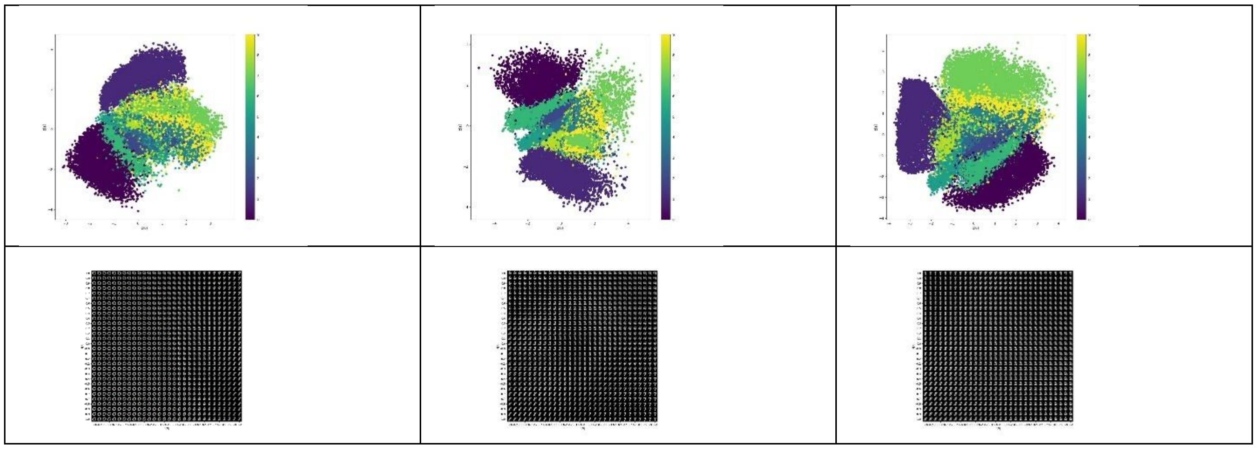
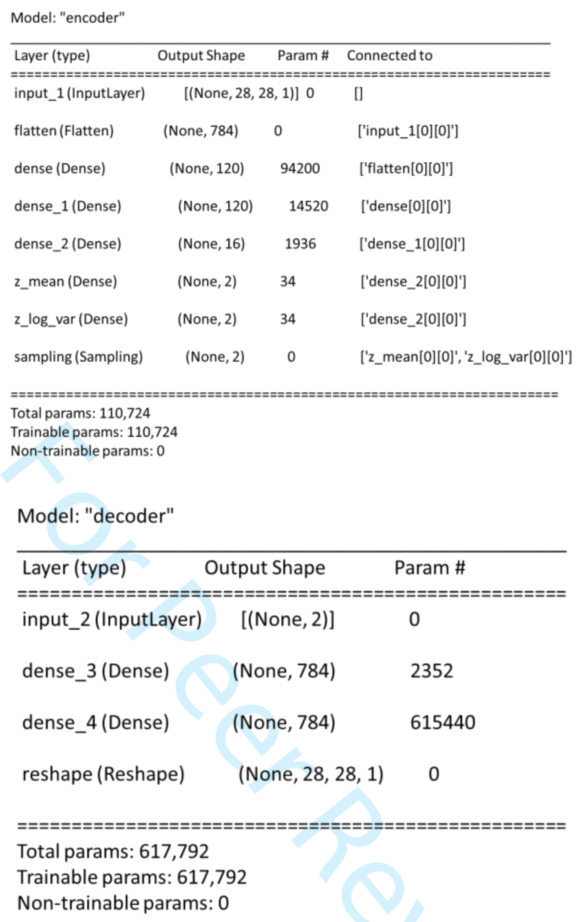


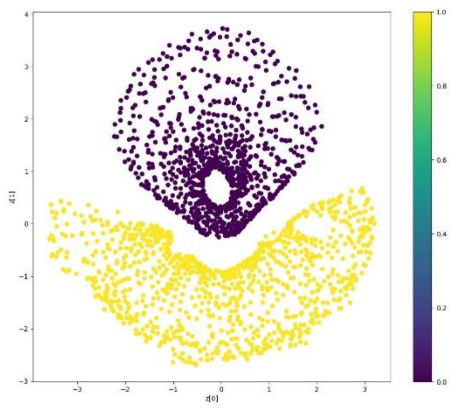
Figure S2. Clustering of mnist dataset in the latent space and corresponding generated digits from the trained algorithms, left: CNN_CNN VAE, middle: FCNN_CNN VAE, and right: FCNN_FCNN VAE.



```
Model: "encoder"
_______________________________________________________________________
Layer (type)              Output Shape          Param #     Connected to
=======================================================================
input_1 (InputLayer)      [(None, 28, 28, 1)]   0           []

flatten (Flatten)         (None, 784)           0           ['input_1[0][0]']

dense (Dense)             (None, 120)           94200       ['flatten[0][0]']

dense_1 (Dense)           (None, 120)           14520       ['dense[0][0]']

dense_2 (Dense)           (None, 16)            1936        ['dense_1[0][0]']

z_mean (Dense)            (None, 2)             34          ['dense_2[0][0]']

z_log_var (Dense)         (None, 2)             34          ['dense_2[0][0]']

sampling (Sampling)       (None, 2)             0           ['z_mean[0][0]', 'z_log_var[0][0]']

=======================================================================
Total params: 110,724
Trainable params: 110,724
Non-trainable params: 0
```

```
Model: "decoder"
_____________________________________________________
Layer (type)              Output Shape          Param #
=====================================================
input_2 (InputLayer)      [(None, 2)]           0

dense_3 (Dense)           (None, 784)           2352

dense_4 (Dense)           (None, 784)           615440

reshape (Reshape)         (None, 28, 28, 1)     0

=====================================================
Total params: 617,792
Trainable params: 617,792
Non-trainable params: 0
```

Figure S3. Architecture of the fully connected variational autoencoder from encoder and decoder structures: the node numbers, layer types, and trainable parameters



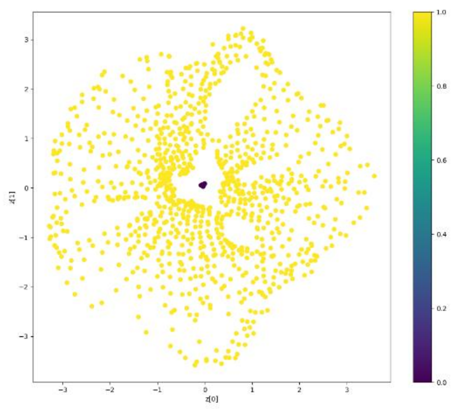
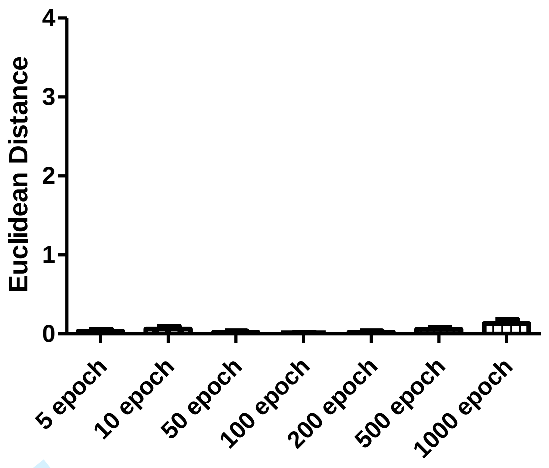


Figure S4. Clustering shape after training of 1000 controls versus 1000 experiments, a balanced dataset (top) and after training of 10 controls versus 1000 experiments, an imbalanced dataset (bottom) for 500 epochs



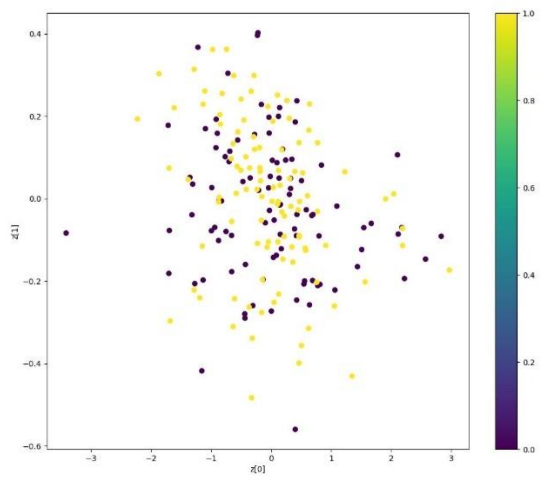


Figure S5. Training of the random noise dataset with the increasing number of epoch (top) and the shape of the presumed control versus experiment clusters after 500 epochs (bottom)

Code s1: Unsupervised data compression and inference using CNN_CNN variational autoencoder empowered with tensorboard callback and silhouette score

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from keras.callbacks import TensorBoard
from datetime import datetime
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score


NAME = "Unsupervised_clustering"
tensorboard = TensorBoard(
    log_dir='///logs/{}'.format(NAME))


(x_train, y_train), _ = keras.datasets.mnist.load_data()
x_train = np.expand_dims(x_train, -1).astype("float32") / 255


class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon

latent_dim = 2

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation="relu", strides=2,
padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()


latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
x = layers.Reshape((7, 7, 64))(x)
x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2,
padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2,
padding="same")(x)
decoder_outputs = layers.Conv2DTranspose(1, 3, activation="sigmoid",
padding="same")(x)
```

```
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()

class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction),
axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) -
tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }


vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam())
tstart = datetime.now()
vae.fit(x_train, epochs=10, batch_size=128, callbacks=[tensorboard])
tend = datetime.now()
elapsed_time = tend - tstart
print("time to fit the model is", elapsed_time)
def plot_latent_space(vae, n=30, figsize=15):
    digit_size = 28
```

```
        scale = 1.0
        figure = np.zeros((digit_size * n, digit_size * n))
        grid_x = np.linspace(-scale, scale, n)
        grid_y = np.linspace(-scale, scale, n)[::-1]

        for i, yi in enumerate(grid_y):
            for j, xi in enumerate(grid_x):
                z_sample = np.array([[xi, yi]])
                x_decoded = vae.decoder.predict(z_sample)
                digit = x_decoded[0].reshape(digit_size, digit_size)
                figure[
                    i * digit_size : (i + 1) * digit_size,
                    j * digit_size : (j + 1) * digit_size,
                ] = digit

        plt.figure(figsize=(figsize, figsize))
        start_range = digit_size // 2
        end_range = n * digit_size + start_range
        pixel_range = np.arange(start_range, end_range, digit_size)
        sample_range_x = np.round(grid_x, 1)
        sample_range_y = np.round(grid_y, 1)
        plt.xticks(pixel_range, sample_range_x)
        plt.yticks(pixel_range, sample_range_y)
        plt.xlabel("z[0]")
        plt.ylabel("z[1]")
        plt.imshow(figure, cmap="Greys_r")
        plt.show()
    plot_latent_space(vae)

    def plot_label_clusters(vae, data, labels):
        z_mean, _, _ = vae.encoder.predict(data)
        plt.figure(figsize=(12, 10))
        plt.scatter(z_mean[:, 0], z_mean[:, 1], c=labels)
        plt.colorbar()
        plt.xlabel("z[0]")
        plt.ylabel("z[1]")
        plt.show()
    plot_label_clusters(vae, x_train, y_train)

    z_mean, _, _ = vae.encoder.predict(x_train)
    z_mean = np.array(z_mean)
    clusterer = KMeans(n_clusters=10, random_state=10)
    cluster_labels = clusterer.fit_predict(z_mean)
    silhouette_avg = silhouette_score(z_mean, cluster_labels)
    print("The average silhouette_score is :", silhouette_avg)
```

Code s2: Fully connected variational autoencoder training from tabular CSV files and estimation of performance with Euclidean distance

```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from keras.callbacks import TensorBoard

NAME = "Unsupervised_clustering"
tensorboard = TensorBoard(
    log_dir='///logs/{}'.format(NAME))

n_dim = 88

file_path = '///unnamed.csv'
df = pd.read_csv(file_path)
X = df.copy()
Y = X.pop("target")
X_np = np.array(X)
x_train = np.reshape(X_np,(-1, n_dim, n_dim, 1))
y_train = np.array(Y)

class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon

latent_dim = 2

encoder_inputs = keras.Input(shape=(n_dim, n_dim, 1))
x = layers.Flatten()(encoder_inputs)
x = layers.Dense(120 , activation="relu")(x)
x = layers.Dense(120 , activation="relu")(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()

latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(n_dim * n_dim * 1, activation="relu")(latent_inputs)
x = layers.Dense(n_dim * n_dim * 1, activation="sigmoid")(x)
decoder_outputs = layers.Reshape((n_dim, n_dim, 1))(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()

class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction),
axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) -
tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = 0.1*reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }


vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam())
vae.fit(x_train, epochs=10, batch_size=500, callbacks=[tensorboard])

def plot_label_clusters(vae, data, labels):
    z_mean, _, _ = vae.encoder.predict(data)
    plt.figure(figsize=(12, 10))
    plt.scatter(z_mean[:, 0], z_mean[:, 1], c=labels)
    plt.colorbar()
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.show()
plot_label_clusters(vae, x_train, y_train)
```

```
data = []
length = len(y_train)

for i in range(0, length, 1):
    b = np.reshape(x_train[i], (1, n_dim, n_dim, 1))
    z_mean, _, _ = vae.encoder.predict(b)
    c = np.array([[z_mean[:, 0], z_mean[:, 1]]])
    c = np.reshape(c, (2, 1))
    c = np.transpose(c)
    data.append(c)

data = np.array(data)
data = np.reshape(data, (length, 2))
y_train_label = np.reshape(y_train, (length, 1))
data = np.concatenate((data, y_train_label), axis=1)
means_cluster_1 = data[0:int(length/2), :2].mean(axis=0)
means_cluster_2 = data[int(length/2):length+1, :2].mean(axis=0)
dist = np.linalg.norm(means_cluster_1-means_cluster_2)
print("Euclidian distance is ", dist)
```