Dimensionality Reduction through Autoencoders and subsequent Cluster Analysis of Single-Cell Sequencing Data

Raheel Ahmed
The University of Texas at Dallas
Richardson, Texas
rsa170130@utdallas.edu

Nisarg Sanjay Shah
The University of Texas at Dallas
Richardson, Texas
nss220000@utdallas.edu

Ansh Patel
The University of Texas at Dallas
Richardson, Texas
asp220000@utdallas.edu

FNU Khaja Airaj Uddin Najaf
The University of Texas at Dallas
Richardson, Texas
kxf220001@utdallas.edu

***Abstract*—Our project is at the forefront of innovation in the fields of Big Data and Machine Learning, as we propose a novel approach to analyzing single-cell sequencing data. To tackle the complexities inherent in high-dimensional genomic profiles, we leverage autoencoders—a powerful neural network architecture—to distill this information into a concise yet informative latent space representation. This reduced-dimensional output becomes the central focus for various clustering algorithms, allowing us to gain unique insights into cellular subpopulations.**

The culmination of our efforts lies in a visually compelling plot that synthesizes the outcomes generated by these clustering algorithms. By incorporating the latent vectors produced by our autoencoder model as input, this tangible representation proves invaluable for both research and clinical applications alike—providing deep insights into cellular relationships. Moreover, our implementation makes use of PySpark, MLFlow, and popular deep learning frameworks like PyTorch,Tensorflow,and Keras.

At its core, the primary objective of our project is to investigate if deep learning networks are capable of forming meaningful clusters for high-dimensional microscopic data. By pushing the boundaries of advanced techniques in this field, research aims not only to enhance understanding regarding cell heterogeneity, but also to pave ways toward precision medicine, disease characterization ,and in other related areas. This interdisciplinary exploration at the intersection of genomics and deep learning underscores the transformative potential of dimensionality reduction through autoencoders in unraveling the complexities of single-cell sequencing data, paving the way for biomedical research advancements.

***Index Terms*—Dimensionality Reduction, Autoencoders, Cluster Analysis, KMeans, PySpark, MLFlow**

I. Introduction

The innovative approach we are undertaking in our project revolutionizes the field of Big Data and Machine Learning by introducing a methodology for analyzing single-cell sequencing data. Our strategy involves leveraging autoencoders, which are powerful neural networks capable of effectively handling intricate genomic profiles. Through their operation, these remarkable autoencoders transform crucial information into concise yet meaningful representations. The significance lies in the condensed output as it becomes the focal point for various clustering techniques, granting us unique insights into cellular behavior patterns. This captivating expedition propels us beyond conventional methodologies and unlocks new prospects to comprehend complex genomic data intricacies thoroughly. Ultimately, our endeavors pave the way for discoveries within biomedical research domains and become useful in precision medical advances, drug discovery development, understanding of cellular heterogeneity. Our paper focuses on reducing the dimensionality using the PCA technique and Autoencoders. The Encoder and Decoder architecture first breaks the input in latent space representation and the reconstructs back to original form. Our Dataset has an unnormalized count matrix with dimensions 5826 x 27630 prior to dimension reduction. We preprocess the data by converting the CSV data to ANN data followed by Filtering the Matrix and Confounder Removal. Additionally, this paper provides a comprehensive explanation of the concepts used in our project, including the fundamentals of Neural Networks, Architecture of the Autoencoders and Cluster Analysis. The data preprocessing steps, model training process, and the utilization of PySpark's map-reduce framework are discussed in detail. The paper concludes by summarizing the contributions of the project and potential future directions for enhancing the model's performance, including exploring deeper architectures, transfer learning, and data augmentation techniques.

II. TECHNIQUE/ALGORITHM-RELATED WORK

A. Dimensionality Reduction

Dimensionality reduction is a process in machine learning and statistics where data with a high number of dimensions is re-represented as a lower dimension dataset; in essence, the number of variables under consideration is reduced. To do this in a meaningful way, the lower dimensional data must retain critical information present in the higher dimension, original

data. Dimensionality reduction is very useful for simplifying models, reducing computational costs, and in dealing with the curse of dimensionality, which can hamper the performance of machine learning models. Dimensionality reduction can be achieved through various techniques

B. Dimensionality Reduction Methods:

- Principal Component Analysis (PCA): Transforms data to a new coordinate system consisted of principle components, focusing on directions with maximum variance.
- Linear Discriminant Analysis (LDA): Aims to find a linear combination of features that separates two or more classes.
- t-Distributed Stochastic Neighbor Embedding (t-SNE): A non-linear technique for reducing dimensionality of data while preserving local relationships between data points.
- Autoencoders: Neural networks that learn efficient codings of data by compressing and reconstructing the input.
- Singular Value Decomposition (SVD): A matrix factorization method used in image compression and recommender systems.
- Feature Selection Techniques: Methods like backward elimination and forward selection to choose a subset of relevant feature.

C. Building Autoencoders in Keras

Autoencoders are a type of neural network characterized by a compression of larger dimension vector into a latent space then decompression to the same larger dimension vector; it is particularly effective in tasks like dimensionality reduction and feature learning. Here's a short guide on how to build them using Keras, a high-level neural networks API:

1) Defining the Architecture: An autoencoder typically consists of two main parts: an encoder and a decoder, as seen in [Fig. 1]

Encoder: This part of the network compresses the input into a latent-space representation. It usually consists of a series of layers that reduce in size, funneling the input data into a more compressed form.

Decoder: This part attempts to reconstruct the input data from the compressed latent-space representation. The layers in the decoder typically increase in size and aim to replicate the original input data as closely as possible.

2) Choosing Layers and Activation Functions: Layers: Common choices include Dense (fully connected) layers, as shown in [Fig. 2] or Convolutional layers for image data

Activation Functions: Functions like ReLU (Rectified Linear Unit) are commonly used. For the final layer, an activation function that will yield values similar to the original data will be used. Sigmoid and softmax activation might be used in probabilistic outputs whereas linear activation functions and their variants can be used for regression tasks.

3) Setting Up the Loss Function and Optimizer: The loss function measures how well the decoder is able to reconstruct the input data.

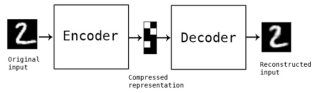

Fig. 1. General workflow and architecture of an autoencoder

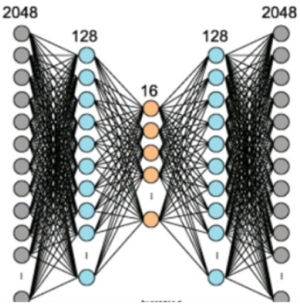

Fig. 2. Example of a fully connected Autoencoder with specified widths for all layers.

Optimizers like Adam or SGD (Stochastic Gradient Descent) are used to minimize the loss function during training.

Learning rate is also critical for proper training; a balance is needed to avoid slow training and to still achieve a low loss.

4) Training the Model: The autoencoder is trained using input data, with the target data being the input data as well. Parameters like the number of epochs and batch size are set according to the dataset and computational resources.

5) Testing and Using the Autoencoder: After training, the autoencoder can be used for tasks like dimensionality reduction by utilizing the encoder to produce a latent-space representation of data.

D. Dimensionality Reduction through Autoencoders

Autoencoders provide a robust solution by compressing high-dimensional data into a lower-dimensional space, which simplifies the data structure without losing critical information. This process not only aids in data visualization and interpretation but also enhances the effectiveness of subsequent analytical techniques like cluster analysis.

Autoencoder Architecture and Function

The Autoencoder architecture used in our study consists of several key components:

1. Input Layer: Receives the high-dimensional data, typically representing a large set of genes or features (in our case,max-random-projection = 2048).

2. Hidden Layers: These intermediate layers (with hidden-dims = (128, 16) neurons) use non-linear transformations to process the data. They play a crucial role in learning the data's inherent structure.

3. Encoded Layer: This is the heart of the autoencoder, where data is compressed into a lower-dimensional space (encoded-dim = 2). This layer captures the most salient features of the data.

4. Decoding Layers: Post the encoded layer, the network attempts to reconstruct the original data from the compressed form, maintaining as much original information as possible.

5. Output Layer: Produces a reconstructed version of the input data, ideally retaining the key characteristics of the original dataset.

E. Autoencoder Training

1) Parameter Setting:

Input Dimensions (max-random-projection = 2048): This parameter likely represents the number of features in your input data, which, in the context of scRNA-seq typically corresponds to gene expression levels across different genes.

Latent Space Dimensionality (encoded-dim = 2): The dimensionality of the latent space, where the data is compressed. This size indicates how much the data is reduced, aiming to retain essential information in a more compact form.

Hidden Layer Neurons (hidden-dims = 128, 16): The number of neurons in each hidden layer of the autoencoder. This value suggests a model complexity capable of learning intricate patterns in the data.

Learning Rate (0.001), Batch Size (32), Epochs (100): These are hyperparameters for training the neural network. The learning rate controls the step size at each iteration of the optimization, the batch size dictates how many data points are used in one update of the model parameters and the epochs define how many times the entire dataset is passed through the neural network

F. Model Architecture

The architecture consists of an input layer that takes data of shape max-random-projection, followed by hidden layers with ReLU activation,leading to an encoded layer. The data is then decoded back to the original dimensionality in the output layer.

1) Encoder Information Extraction:

Encoder Model: The encoder part of the autoencoder is defined, which takes the input data and compresses it to the encoded-dim latent space.

Purpose: This model is used to transform high-dimensional data into a more manageable lower-dimensional space.

Decomposition of snRNA/scRNA Data: Data Transformation: The high-dimensional snRNA data is fed into the encoder, which compresses it into a 2-dimensional latent space.

Retention of Key Features: Despite the reduction in dimensions, the process is designed to retain the most critical aspects of the data, essential for biological analysis.

G. Data Types and Handling

Initially, we are given datasets concerning two different types of plant groups: Wild-Type and Short Root. These plant datasets have count matrices encoded in CSV files where the rows are the individual cell samples and the columns are features concerning expression of that specific gene.

Ideally, we would be able to combine these two datasets, but we first need to ensure that the type of plant group is not a reasonable differentiator of cellular and genetic expression because that would be another source of variability. Our hope is that only the expression level is an indicator of variability in the type of cell and, therefore, the clustering. If the plant groups do not impact this cell clustering and assignment of cell type, it is acceptable to merge these two datasets and conduct a more sufficient analysis.

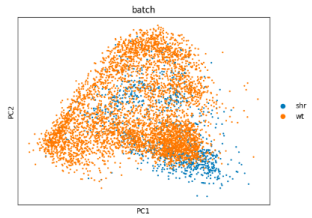

Fig. 3. Plot of data points in 2D space consisting of the first two principle components separated by plant group.

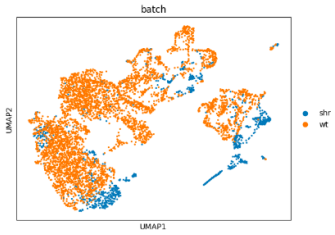

Fig. 4. Plot of data points in 2D space consisting of UMAP1 and UMAP2 separated by plant group.

To do so, we looked at a lower dimensional representation via UMAP and PCA and depicted which type of plant group was present per point. As you can see in [Fig.3, Fig. 4], there's no substantial separation between these two groups, so we can concatenate the matrices and conduct a full analysis.

The matrices are then concatenated and placed into an AnnData structure, where qualities on a per-observation can be recorded, such as cluster values, and the ScanPy module we plan on using can make plots in context to those observations. Further information about the structure and attributes of AnnData can be found in the documentation.

Additionally, we do notice through evaluation of the full dataset's metrics that some preprocessing will be needed. Initially, there is already the problem of having a high dimensional dataset, but on top of that, some features of the data are inherently imbalanced. There's a lot of variance in the number of genes and number of cells by count, as we can see in [Fig. 5] and based on the scattering of data points in [Fig. 6]

The preprocessing we conduct involves filtering cells with an imposed restriction of having expression of at least 100 genes. Another filtration we do is to restrict the gene, or variable, information by imposing a restriction of being present in at least 2 cells. Finally, we filter out any observations where the number of genes by count is greater than 12,000 and the

n_genes_by_counts	n_cells_by_counts
1860	915
6564	2041
5483	551
733	616
3710	4271
...	...
3880	0
3784	0
5600	0
5664	0
3582	52

Fig. 5. Tabular representation of the number of genes and cells by counts of various observations

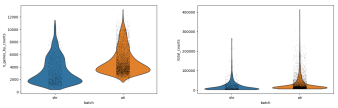

Fig. 6. Violin plot of the data points in terms of number of genes and cells by counts

total count is greater than 120,000.

For our next preprocessing steps, we examine the sparse matrix representation in our AnnData. We decide to normalize counts per cell, logarithmize the data matrix, regress out unwanted sources of variation, and scale the data with a maximum value of 10. At this point, we have our final, full preprocessed data.

III. Results and Evaluation

To understand what a reasonable clustering of these data points should look like in a two-dimensional space, we can observe a plot of our data dimensionally reduced via PCA and UMAP with labelling of clusters through Louvain and Leiden Clustering on the original data, as seen in [Fig. 7, 8].

Distinct globular clusters can be seen in a reasonable quantity. According to our references on cellular types for this data set, these are very reasonable clustering.

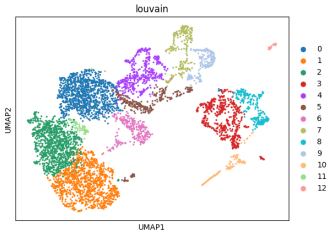

Fig. 7. Louvain Clustered points on a UMAP graph.

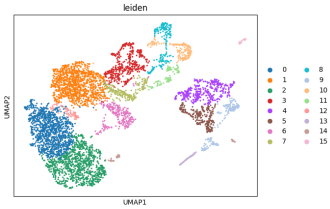

Fig. 8. Leiden Clustered points on a UMAP graph.

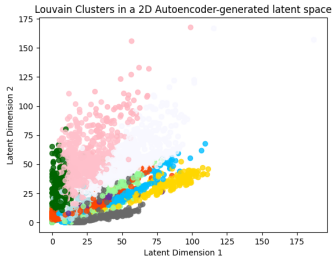

Fig. 9. Louvain Clustered points on the Autoencoder-generated 2D Latent space graph.

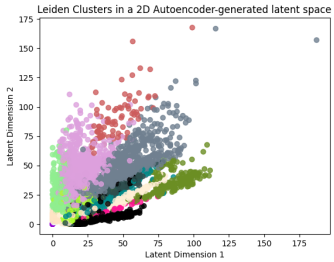

Fig. 10. Leiden Clustered points on the Autoencoder-generated 2D Latent space graph.

Our first goal for our Autoencoder architecture was to produce a latent space that, when plotted, would distinguish between data points that had been properly clustered, showing good separation between cluster members. We decided to fit our Autoencoder to a subset of our data, extract the encoder model to predict the positioning of the data points, and plot the data points in context to our two-dimensional latent space. We decided to use Louvain and Leiden labels as they provided a reasonable clustering in context to our other two-dimensional plot of UMAP, as seen in [Fig. 9, 10].

From these figures, we can see that the Autoencoder does produce a two-dimensional representation of our original data that has enough variability to distinguish between reasonable clusters, an important quality in dimension reduction. However, we also notice that the data points that should be clustered together produce linear structures rather than the globular structures present in UMAP. Immediately, our inference was that these linear structures would be challenging to cluster properly with solely the latent vector as input.

We attempted this, nevertheless, with a custom K-means algorithm using PySpark and MLLib data structures, with the number of centroids being 6 in our example shown in [Fig. 11].

As expected, the compressed dimension data set had not retained enough information to produce a cluster that could infer the linear pattern of the data in respect to the latent space generated.

We tried the same with HDBScan's Robust Single Linkage Clustering Algorithm, with the same number of centroids in our example shown in [Fig. 12].

Again, we ran into the same issue of the compressed

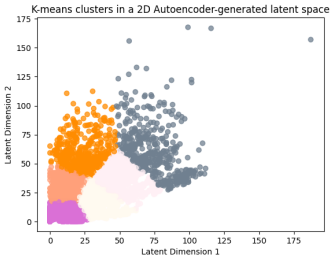

Fig. 11. K-means Clustered points on the Autoencoder-generated 2D Latent space graph.

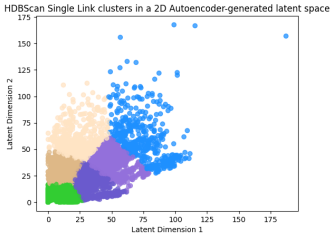

Fig. 12. HDBScan Clustered points on the Autoencoder-generated 2D Latent space graph.

dimension data set not retaining enough information from the original data matrix.

IV. Conclusion

We believe that this does not mean that this approach of using an Autoencoder to generate a reduced dimension dataset is incapable of creating a clustering that can retain these patterns practically as well as the Louvain and Leiden clustering on the original data. Rather, we feel that a less extreme restriction could be done; a latent vector of greater dimension could still offer clustering in accordance to those patterns while being potentially less computationally intensive. Further testing would be ideal, with different architectures and the potential use of ensembles being the primary avenues of investigation.

With proper understanding of the features of our data and a well designed Autoencoder, a better clustering can be produced and can even be visualized in context to UMAP or other state-of-the-art reduced dimension plots, especially if the Autoencoder can't be used to make a latent-space representation with the size of the new latent vector.

References

[1] M. Tekman, B. Serrano-Solano, C. Gallardo, and P. Videm, "Analysis of plant scRNA-Seq Data with Scanpy," Galaxy Training Network, Nov. 03, 2023. https://training.galaxyproject.org/training-material/topics/single-cell/tutorials/scrna-plant/tutorial.html (accessed Dec. 03, 2023).

[2] Geddes, T., Kim, T., Nan, L. et al. Autoencoder-based cluster ensembles for single-cell RNA-seq data analysis. BMC Bioinformatics 20 (Suppl 19), 660 (2019). https://doi.org/10.1186/s12859-019-3179-5

[3] "scCCESS/r_autoencoder.R at master · gedcom/scCCESS," GitHub. https://github.com/gedcom/scCCESS/blob/master/r_autoencoder.R (accessed Dec. 03, 2023).

[4] "Overview-celloracle 0.13.1 documentation," morris-lab.github.io. https://morris-lab.github.io/CellOracle.documentation/notebooks/03_scRNA-seq_data_preprocessing/scanpy_preprocessing_with_Paul_etal_2015_data.html (accessed Dec. 03, 2023).

[5] "Building Autoencoders in Keras," Keras.io, 2016. https://blog.keras.io/building-autoencoders-in-keras.html

[6] Stathis Kamperis, "The encoder-decoder model as a dimensionality reduction technique," *Let's talk about science!*, Jan. 21, 2021. https://ekamperi.github.io/machine%20learning/2021/01/21/encoder-decoder-model.html

[7] Feng X, Fang F, Long H, Zeng R and Yao Y (2022) Single-cell RNA-seq data analysis using graph autoencoders and graph attention networks. Front. Genet. 13:1003711. doi: 10.3389/fgene.2022.1003711

[8] "notebook," Umich.edu, 2023. https://socr.umich.edu/HTML5/ABIDE_Autoencoder (accessed Dec. 03, 2023).