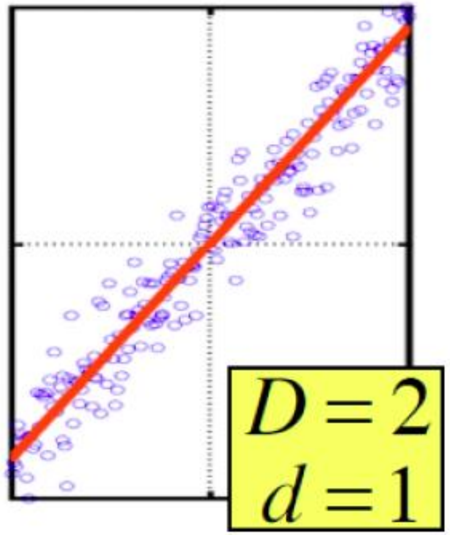
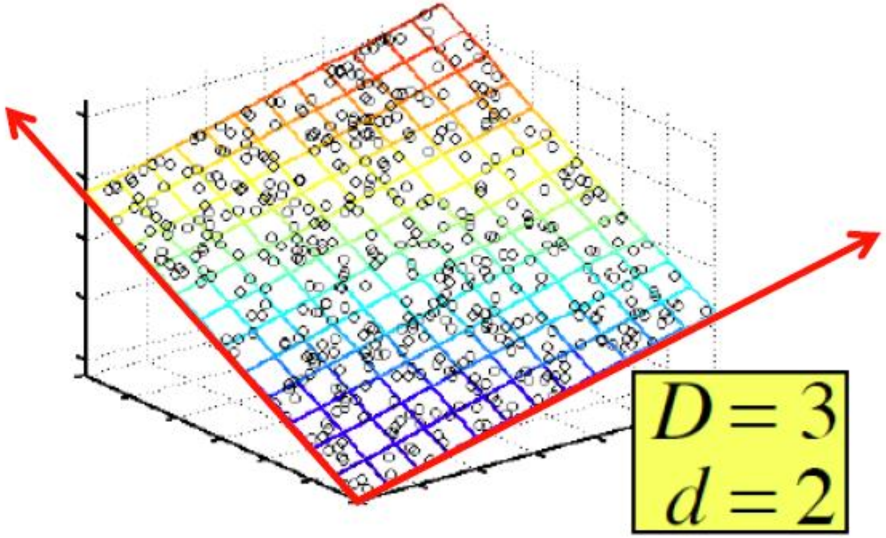
DIMENSONALITY REDUCTION WITH FEED-FORWARD NEURAL NETWORKS AND COMPARISON WITH PRINCIPLE COMPONENT ANALYSIS METHOD

Osman Furkan Kınlı – S002969

OUTLINE

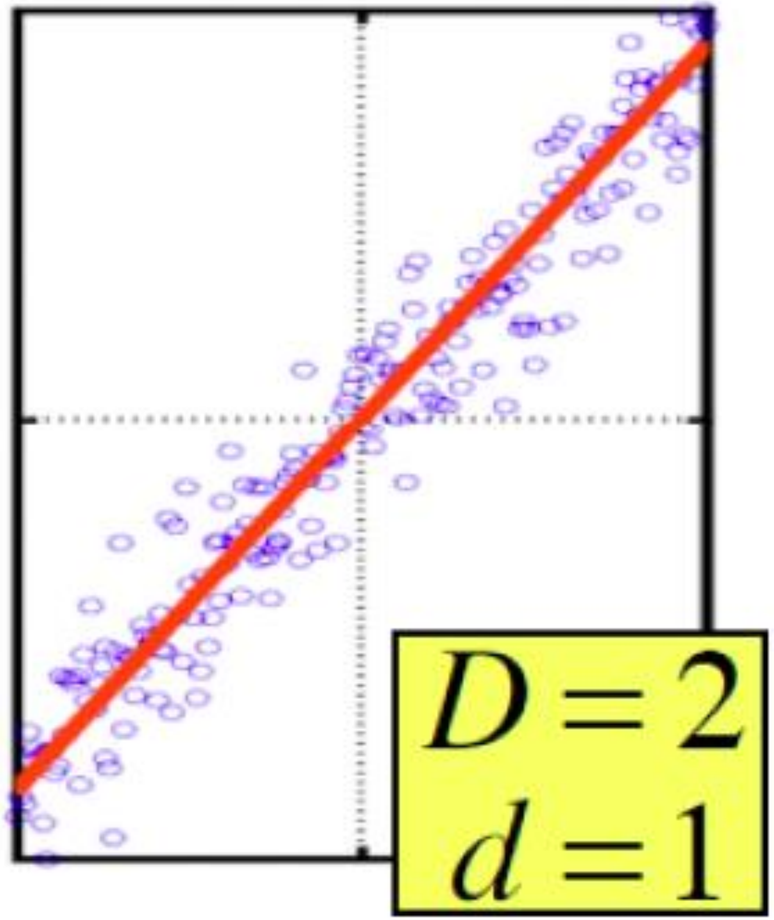
- Dimensionality Reduction
- Principle Component Analysis
- Artificial Neural Networks
- Dimensionality Reduction with ANNs
- Results
- References





DIMENSIONALITY REDUCTION

- Assumption: Data lies on or near a low d-dimensional subspace
- Axes of the subspace are effective representation of the data



DIMENSIONALITY REDUCTION

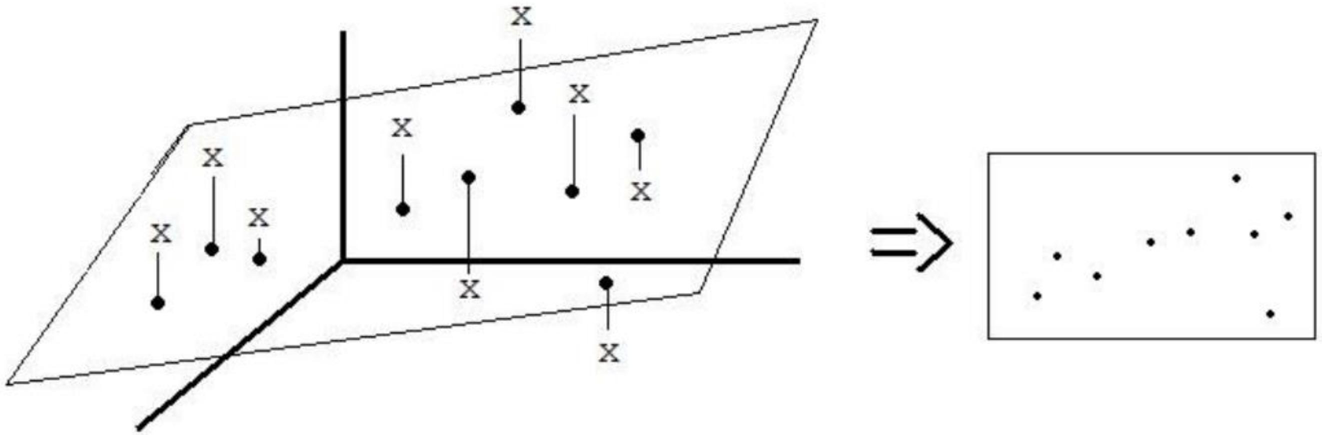
- The main goal of dimensionality reduction is to discover the representative axis of the data.
- Example: All data points can be represented with 1 coordinate corresponding to the position on the red line, rather than representing the points with 2 coordinates.

DIMENSIONALITY REDUCTION

- With the help of dimensionality reduction of the data:
  - Discover hidden correlations between features
  - Remove redundant and noisy features
  - Interpretation and visualization
  - Easy to store and process of the data

PRINCIPLE COMPONENT ANALYSIS

- PCA is one of the popular dimensionality reduction methods.
- This method provides simpler representation of the data by projecting it from higher dimensional space to a lower dimensional sub-space.
- Basically, PCA tries to achieve minimizing the error incurred by reconstructing the data in higher dimension.



PRINCIPLE COMPONENT ANALYSIS

- Interpretation:
  - Maximize the variance of the projection data along each principle component
  - Minimize the reconstruction error
- Example: Projecting 3-dimensional data points into 2-dimensional sub-space

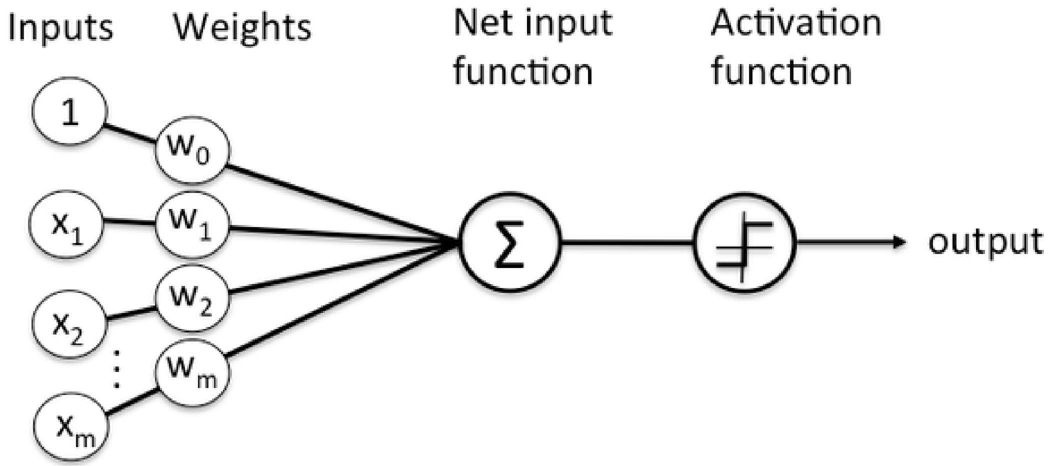
PRINCIPLE COMPONENT ANALYSIS

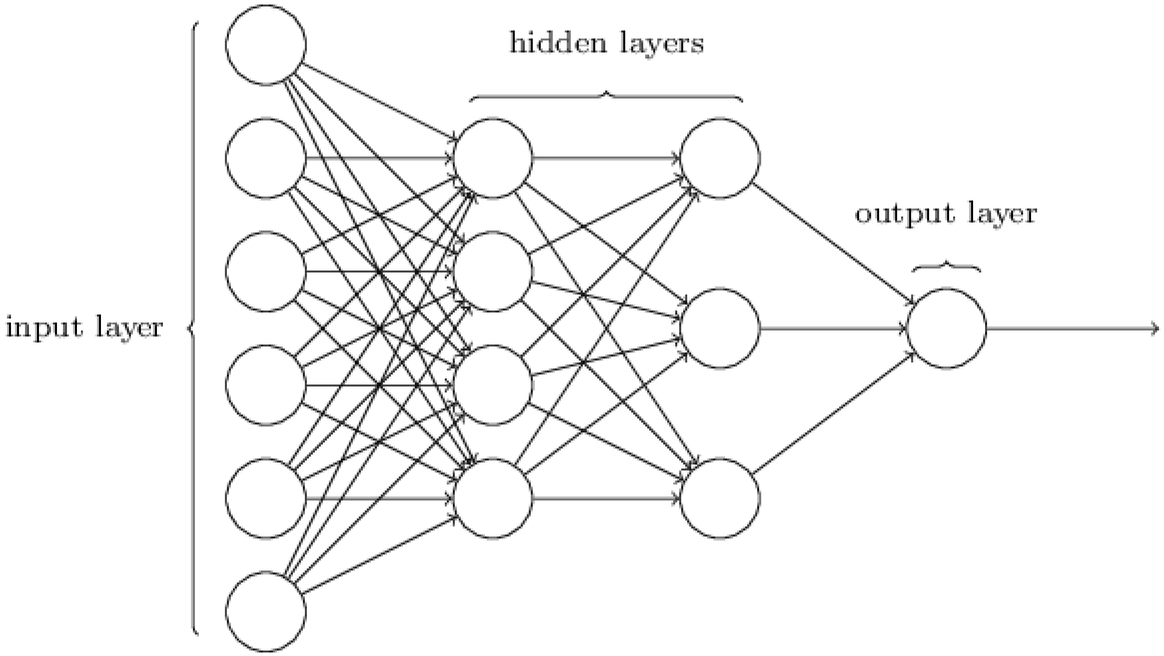
- The algorithm of PCA as follows:
  - Step-1: Subtract the mean from each data point
  - Step-2: Calculate covariance matrix of zero-centered data
  - Step-3: Calculate eigens of the covariance matrix
  - Step-4: Pick the components to project the data

ARTIFICIAL NEURAL NETWORKS

- Information processing paradigm
- Inspired by biological nervous systems
- Composed of a large number of highly interconnected processing elements (neuron)

- Motivation: Minimizing the loss function by learning some information from examples



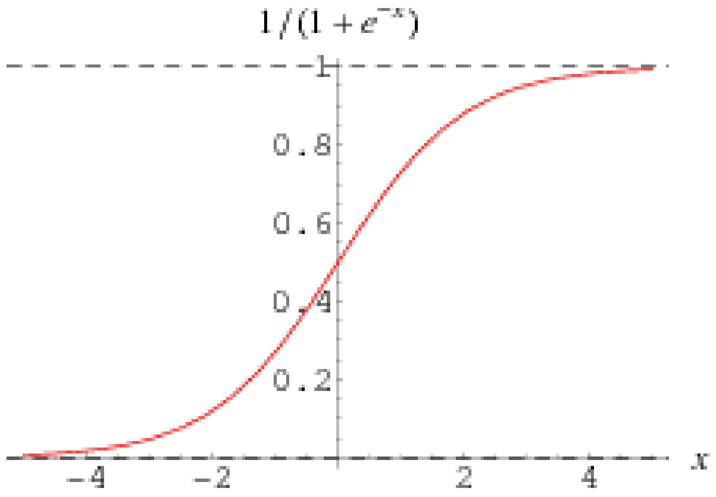


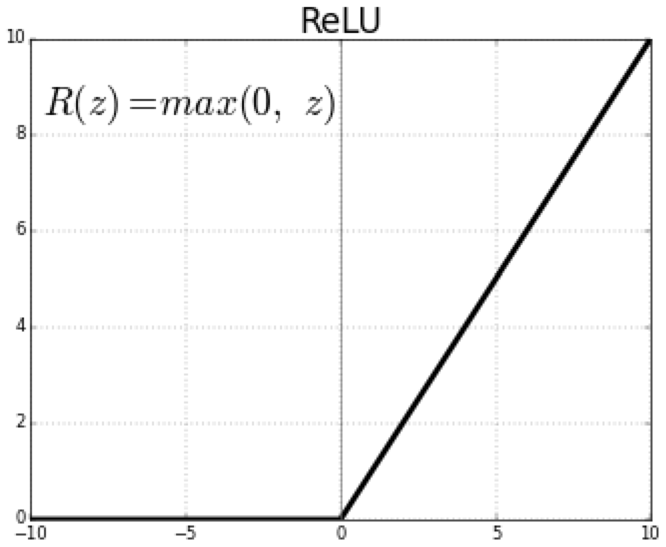
FEED-FORWARD NEURAL NETWORKS

- A single neuron will pass the information to the next neuron with the sum of weighted input signals.
- Net input values will be activated with such a function by mapping the values into desired range.

FEED-FORWARD NEURAL NETWORKS

Activation function: The function (Sigmoid, ReLU etc.) which maps the sum of weighted input signals (can be any number) into a certain range.





FEED-FORWARD NEURAL NETWORKS

Cost function: The function (MSE, Cross-entropy etc.) which represents the difference between actual output and predicted output throughout the network.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

FEED-FORWARD NEURAL NETWORKS

Iterative & Gradient-based optimization: FFNNs are usually trained by using iterative, gradient-based optimizers (SGD, Momentum etc.) that only drive the cost function to very low value.

SGD

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

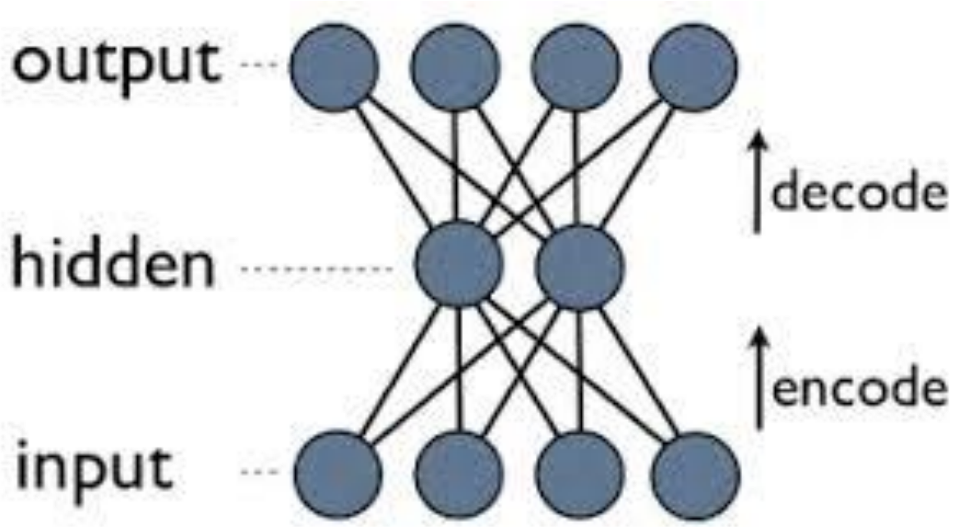
SGD with Momentum Term

$$\nu_j \leftarrow \boxed{\eta * \nu_j} - \alpha * \nabla_w \sum_{1}^{m} L_m(w)$$

$$\omega_j \leftarrow \nu_j + \omega_j$$

$$\boxed{\underbrace{\eta}_{\textit{Coefficient of Momentum}} * \underbrace{\nu_j}_{\textit{Retained Gradient}}}$$

DIMENSIONALITY REDUCTION WITH ANN



- In generative perspective, ANNs can learn to mimic any distribution of data.
- Generative models model the distribution of individual classes.
- Inspired by the structure of auto-encoders.
- While encoder part is responsible to project the data to lower dimensional sub-space, decoder part re-constructs the data in higher dimension.

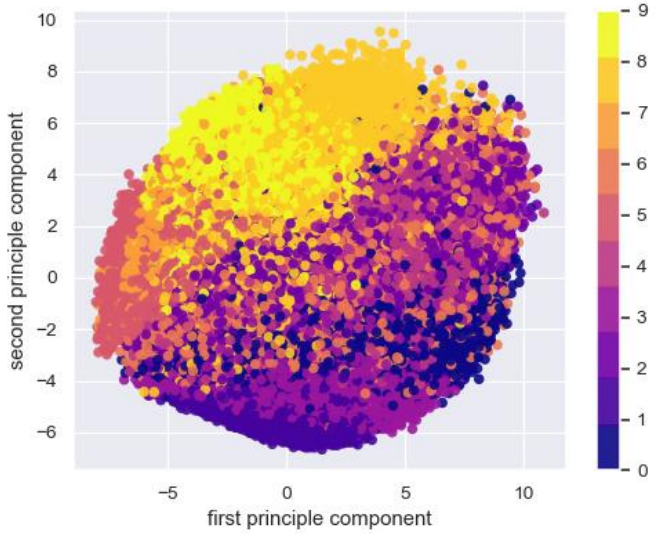
DATASET (FASHION-MNIST)

- FashionMNIST:
  - contains 60.000 gray-scaled fashion images
  - with 10 category labels
  - Dimensionality (h, w, c): 28x28x1

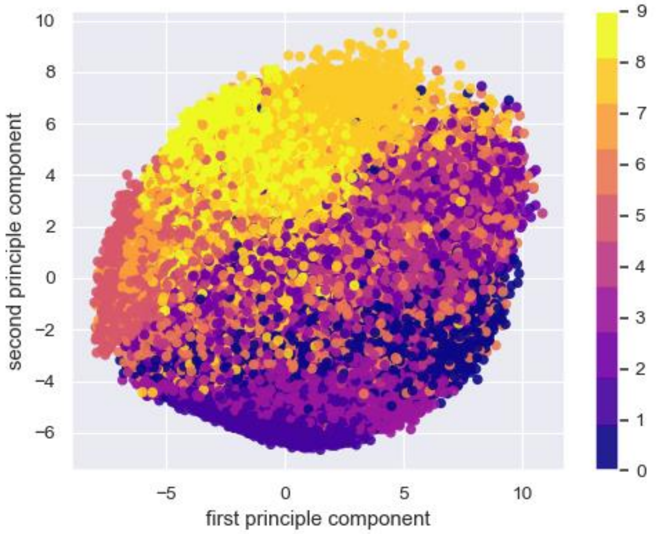
MODEL ARCHITECTURE

- Input layer: (None, 784)
- 1^{st} hidden layer: Size(784, 256) – Activation(ReLU)
- 2^{nd} hidden layer: Size(256, 64) – Activation(ReLU)
- Projection layer (3^{rd}): Size(64, 3)
- Activation layer: Activation(ReLU)
- 4^{th} hidden layer: Size(3, 64) – Activation(ReLU)
- 5^{th} hidden layer: Size(64, 256) – Activation(ReLU)
- Output layer: Size(256, 784)

- Loss function: Mean squared error (MSE)
- Optimizer: Adam



Baseline PCA
(from scikit-learn
library)

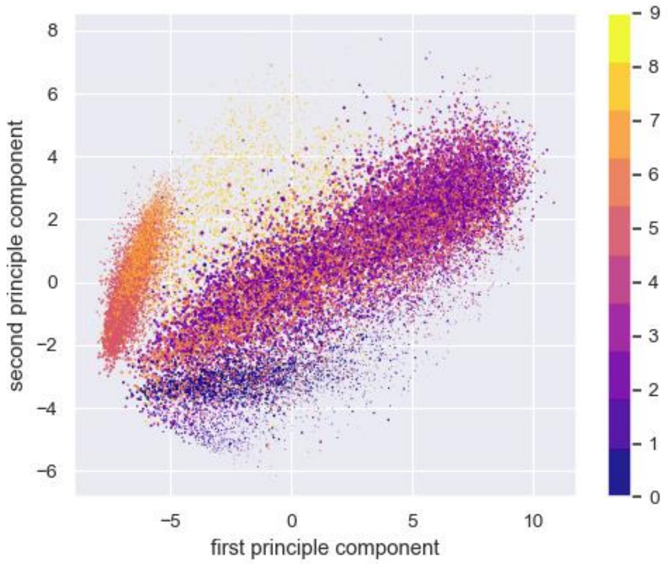


My PCA
Implementation

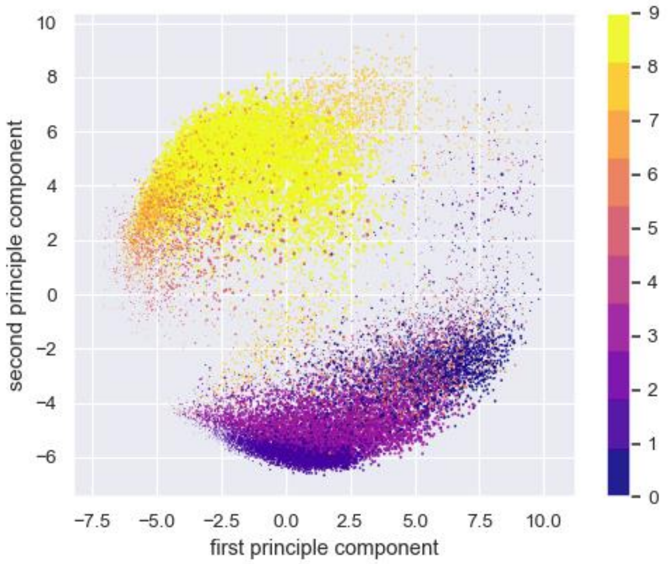
RESULTS

- Left: Built-in function of scikit-learn library with 2 components
- Right: My PCA implementation with 2 components
- In both images, we can see that most individual classes are separated in 2-dimensional sub-space with (of-course) some noise.

RESULTS

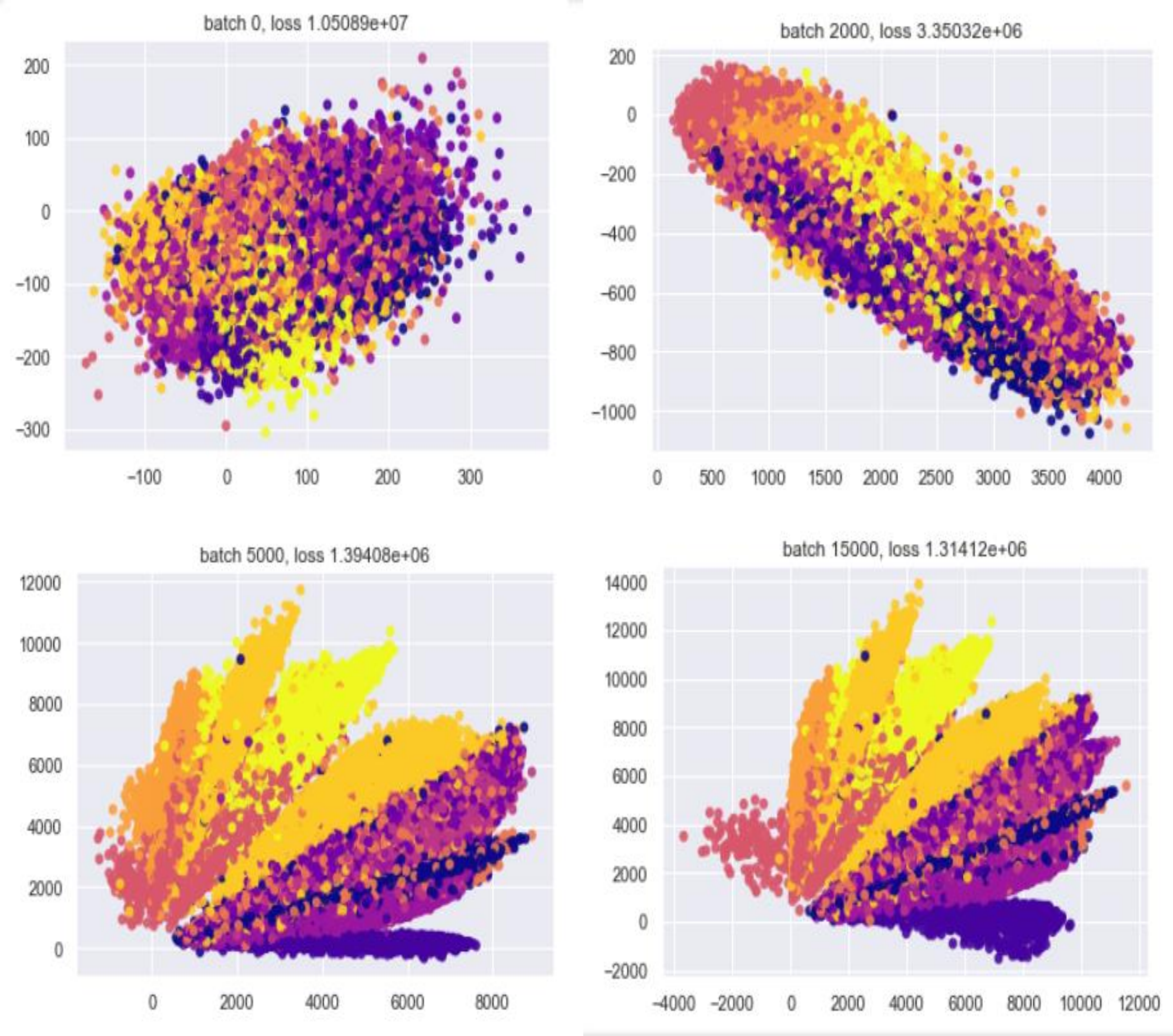


Baseline PCA
(from scikit-learn
library)



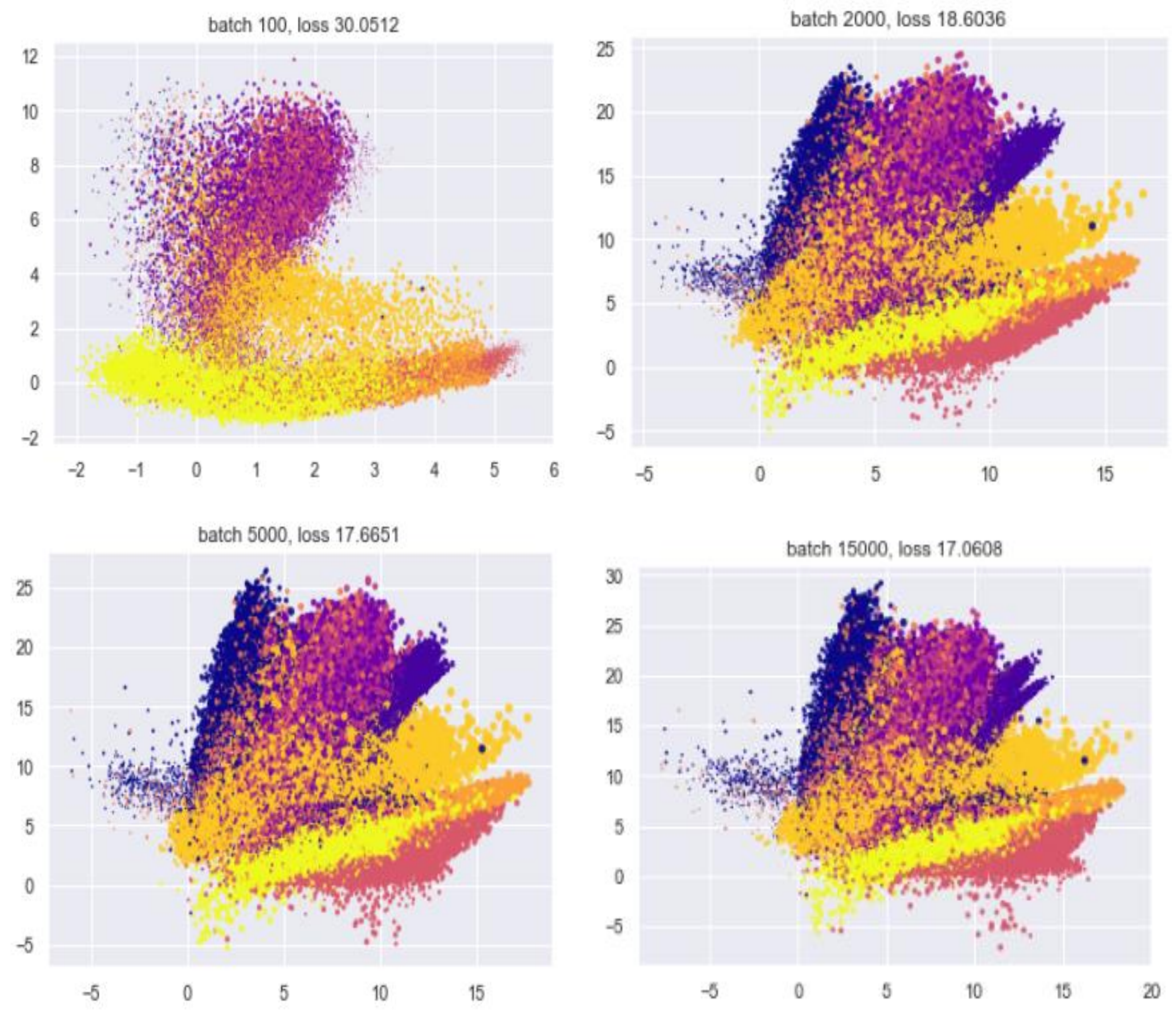
My PCA
Implementation

- Left: Built-in function of scikit-learn library with 3 components
- Right: My PCA implementation with 3 components
- In both images, we can see that most individual classes are separated in 3-dimensional sub-space with (of-course) some noise. (shown in 2-dim)



RESULTS

- Here is the projection of the data on 0., 2000., 5000. and 15000. steps in 2-dimensional sub-space.



RESULTS

- Here is the projection of the data on 100., 2000., 5000. and 15000. steps in 3-dimensional sub-space. (shown in 2-dim)

THANK YOU!

Osman Furkan Kınlı – S002969