EE 565 - Machine Learning I
Project 4 Report

Jorge A. Garcia

***Abstract*—In this project, the performance of a Multilayer Perceptron with varying number of layers is tested with two different data distributions. The concept of early-stopping point for training is studied. Data denoising using an Autoencoder, and dimensionality reduction using Principal Component Analysis is explored.**

***Index Terms*—Machine Learning, Data Science**

I. Introduction

Further exploring the capabilities of deep learning, the Multilayer Perceptron (MLP) is able to solve non-linear problems due to its multiple neurons and layers. This is tested in this report using the MLP to classify the Double Moon and Gaussian XOR distributions.

Neural networks can also be used for the tasks of data compression and, particularly interesting, filtering by using an Autoencoder, a network in which the desired output is the input itself. This type of network would learn the meaningful features in the data set, which ideally should result in noisy data being denoised as the noise is a "useless" feature. This idea is tested using an Autoencoder on the CIFAR-10 data set. Principal Component Analysis (PCA) is another method of dimensionality reduction which finds the basis in which the variance between the data points is maximized. Clustering using PCA is done for the "spikes.csv" data set.

II. Multilayer Perceptron: Double Moon Dataset

For the following section, the Multilayer Perceptron (MLP) function provided was used. A Double Moon distribution with $N = 300$, $r = 1$, $w = 0.6$ and $d = -0.5$ was used. Three configurations using one, two and three hidden layers were used, with a total of 10 neurons distributed across the layers. A total of 10 models using different data sets are trained for each hidden layer configuration, and the average mean squared error is reported, shown in the top plot of the figures. The learning rate used is of $\epsilon = 1 \times 10^{-4}$, a momentum term of $\alpha = 0$ and trained for a maximum of 1000 epochs. After training, the model weights due to the lowest cost found are used to predict for a testing data set generated using the same distribution parameters as before, shown in the bottom plot of the figures.

A. One Hidden Layer

The results for one hidden layer are shown in Figure 1. All models converge at about 150 epochs, with the cost slowly decreasing after that. As shown in the plot for the testing set, the non-linearity of the distribution is captured at the ends, but not as much in the middle.

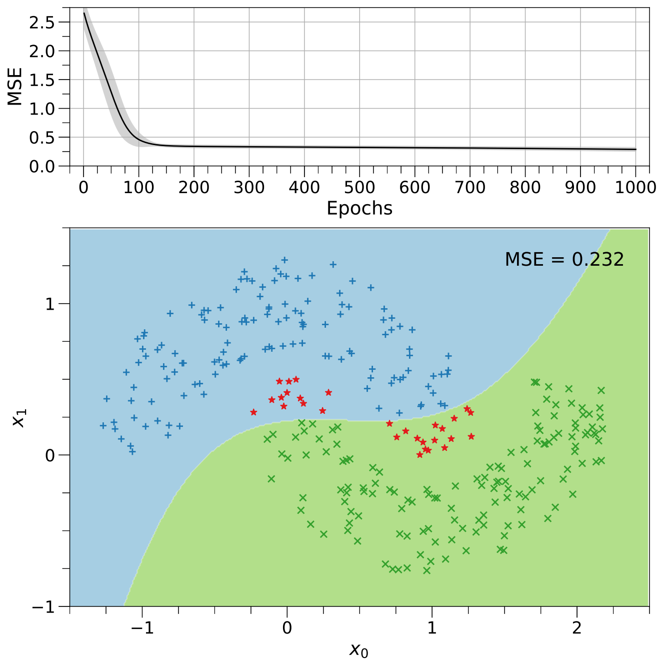

Fig. 1. Average MSE per epoch and classification using the optimal weights for the Double Moon data set using 1 hidden layer.

B. Two Hidden Layers

The results for two hidden layers are shown in Figure 2. There are 6 neurons in the first hidden layer, and 4 in the second one. The models seem to converge at about 200 epochs, and the overall uncertainty is larger than in the previous architecture. This could indicate that the architecture captures much more distinct features that are specific to each data set. Peculiarly, as seen in the predictions for the testing set, the best weights have a linear decision at the edges of the distribution and captures the non-linear part in the middle which is in contrast to the previous one-layer architecture.

C. Three Hidden Layers

The results for three hidden layers are shown in Figure 3. The hidden layer layout is that of 5, 3 and 2 neurons accordingly. This architecture takes longer to train, converging at about 250 epochs, and an even larger uncertainty for the cost at later stages. This increased uncertainty indicates that this architecture can learn even better representations of the data than the previous ones. This is further shown in the prediction plot for the testing set, as the curvature of the positive and negative moons is very well captured by the model.

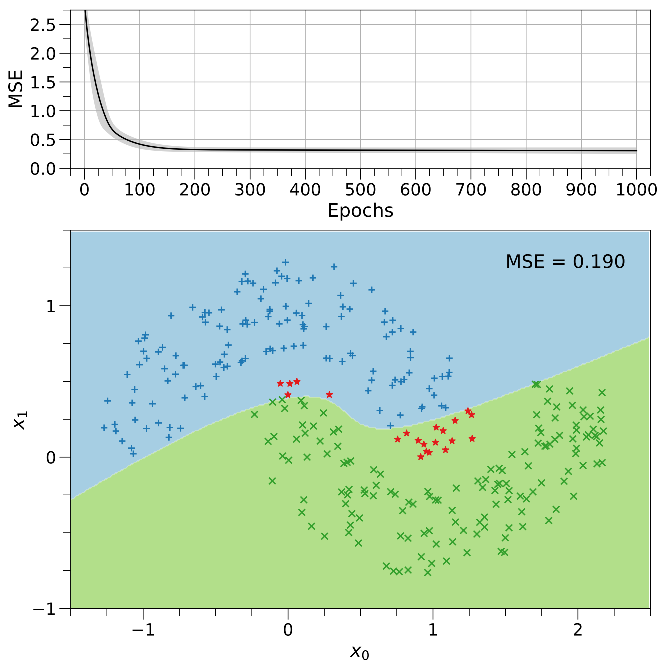

Fig. 2. Average MSE per epoch and classification using the optimal weights for the Double Moon data set using 2 hidden layers.

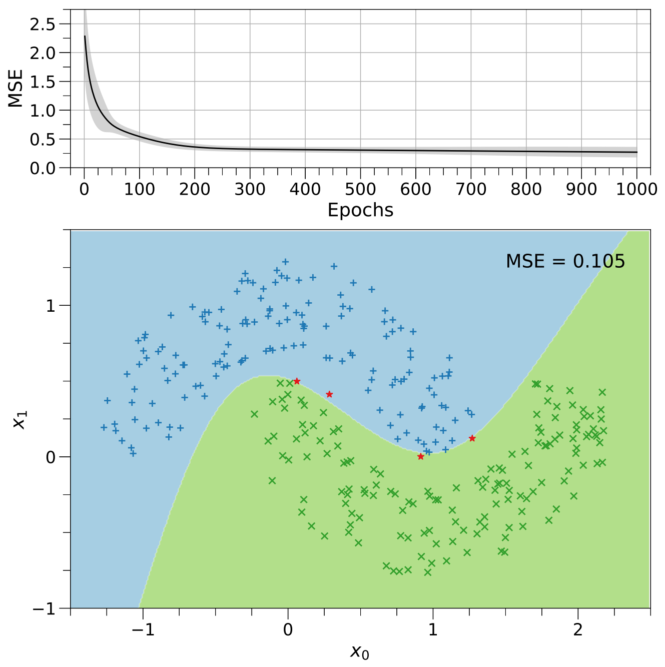

Fig. 3. Average MSE per epoch and classification using the optimal weights for the Double Moon data set using 3 hidden layers.

III. Multilayer Perceptron: Gaussian XOR

For the following section, the MLP function provided is used again. A Gaussian XOR distribution with $N = 300$, $\mu = 0$ and $\sigma = 1$ was used. As before, three architectures with one, two and three hidden layers with a total of 15 neurons distributed across were used. A total of 10 models using different data sets are trained for each hidden layer configuration, and the average mean squared error is reported, shown in the top plot of the figures. The learning rate used is of $\epsilon = 1 \times 10^{-4}$, a momentum term of $\alpha = 0$ and trained for a maximum of 1000 epochs. After training, the model weights due to the lowest cost found are used to predict for a testing data set generated using the same distribution parameters as before, shown in the bottom plot of the figures.

A. One Hidden Layer

The results for one hidden layer are shown in Figure 4. Training of the weights is highly variable at first, and reaches a stable point at about 225 epochs. As seen in the testing set predictions, the decision surface does an overall good job at classifying the testing set, with some exaggerated curved features likely due to data point density of one class compared to the other in certain regions. Something of note that is pervasive throughout the three architectures is that the middle part of the distribution will always be modeled as a continuous section for one of the classes while in reality it should be part of the decision boundary.

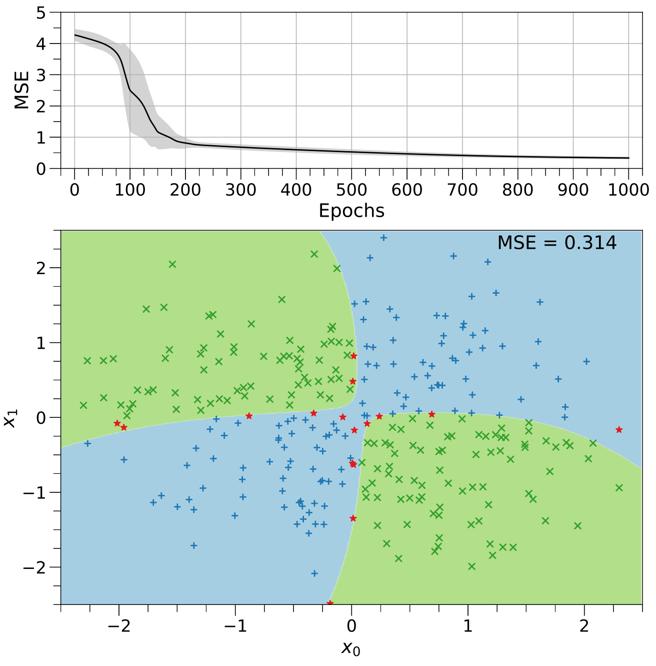

Fig. 4. Average MSE per epoch and classification using the optimal weights for the Gaussian XOR data set using 1 hidden layer.

B. Two Hidden Layers

The results for two hidden layers are shown in Figure 5. The neurons are distributed having 10 neurons in the first layer and 5 neurons in the second. As expected from there being more layers, the models take longer to train and converge near 300 epochs. This architecture does a better job at keeping the straight lines that divide the classes than the previous one, but still has difficulty at learning the center of the data set.

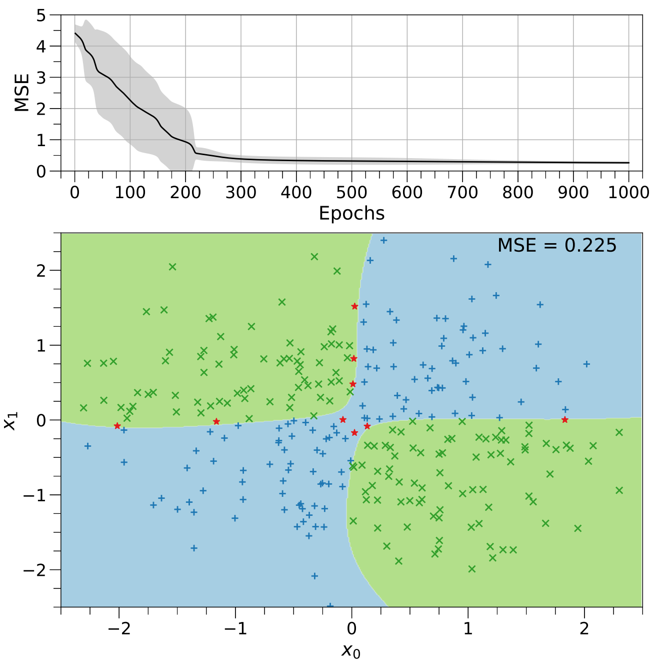

Fig. 5. Average MSE per epoch and classification using the optimal weights for the Gaussian XOR data set using 2 hidden layers.

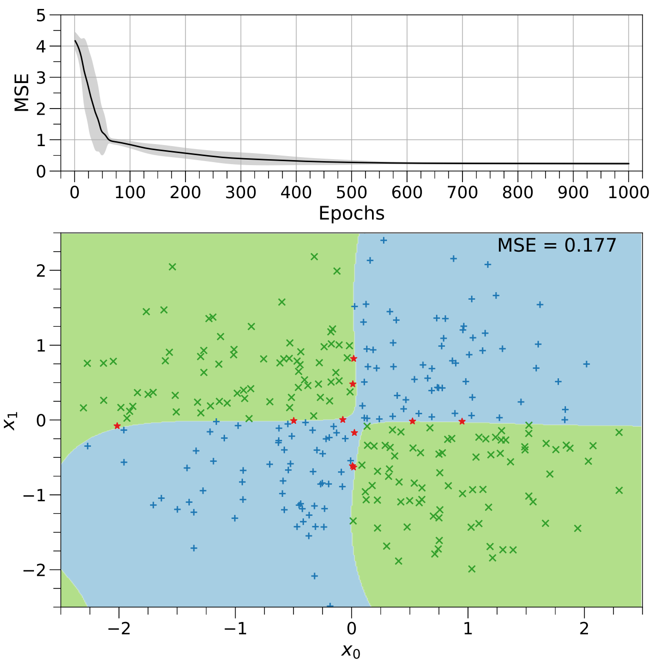

Fig. 6. Average MSE per epoch and classification using the optimal weights for the Gaussian XOR data set using 3 hidden layers.

C. Three Hidden Layers

The results for three hidden layers are shown in Figure 6. The neurons are distributed as 8, 4 and 3 neurons in the first, second and third hidden layer accordingly. Training stabilizes at about 500 epochs. This model was able to effectively keep the linear boundary on one quadrant of the distribution and tighten the center part of the distribution, something the previous architectures would struggle with.

IV. Model Validation

For the following section, the MLP function and modified such that a validation set could be included as part of the input, and the mean squared error of this validation set could be obtained as an output. The Double Moon and Gaussian XOR distributions were used to generate training and validation data, with a very large validation set being used to mimic the conditions to observe model overfitting. Both cases use a MLP with a single hidden layer consisting of 5 neurons in order to obtain a model that is less able to properly learn the distribution; even with this in mind, certain challenges were met. A learning rate of $\epsilon = 1.25 \times 10^{-3}$ and a momentum term of $\alpha = 0$ were used.

A. Double Moon Distribution

The Double Moon distribution was set to have parameters $r = 1$, $w = 0.6$ and $d = -0.5$. The training set consisted of $N = 300$ points and the validation set of $N = 3000$ points. Training was done for 125000 epochs, a number much larger than expected. The early-stopping point was found to be at 47852 epochs, with the testing accuracy slowly starting to diverge after that. This large number of epochs until divergence is likely due to the number of points sampled for the training set are enough to capture the overall shape of the distribution. This would require plenty of epochs in order for the model to start overfitting the distribution. The final accuracies were 100% for the training set and 98.8% for the validation set.

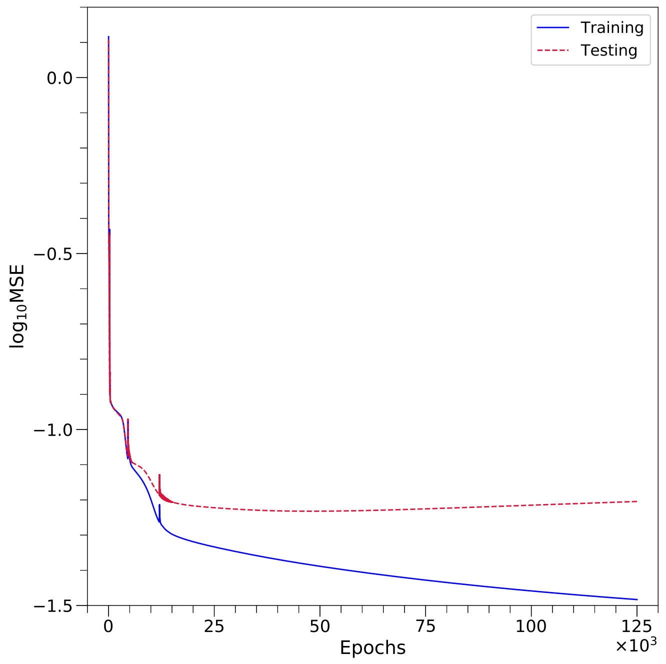

Fig. 7. Log cost as a function of epochs for the Double Moon data set. The early-stopping point is at 47,852 epochs.

B. *Gaussian XOR Distribution*

The Gaussian XOR distribution was set to have parameters $\mu = 0$ and $\sigma = 1$. The training set consisted of $N = 300$ points and the validation set of $N = 3000$ points. Training was done for 1000 epochs. The early-stopping point was found at 341 epochs. The final accuracies were 98.33% for the training set and 93.03% for the validation set.

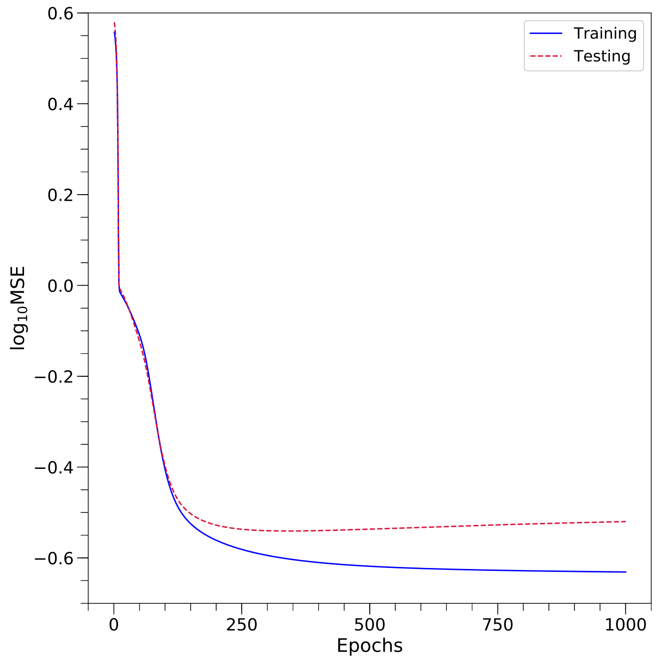

Fig. 8. Log cost as a function of epochs for the Gaussian XOR data set. The early-stopping point is at 341 epochs.

V. AE-based Image Compression/Filtering

For this section, an Autoencoder was implemented using the Keras library in Python. The first 50 images of the CIFAR-10 data set are used for training, and the data is normalized by 255 to lie in the range [0,1].

A. *Tool Used: Keras*

The Python library used to solve this problem was the Keras library [1], which is a wrapper around Tensorflow for quick implementation of deep-learning models. Objects exist to initialize models, various types of layers and optimizers and can easily be linked together through functions within the objects.

A feed-forward model can be initialized through the Sequential class. One can then add the hidden layer of an Autoencoder by initializing an instance of the Dense class, taking as arguments the number of neurons desired in the layer and activation function. Other options such as method of weight initialization or whether to include a regularization term are optional and can be defined if desired. The default weight initialization method uses the Glorot Uniform distribution. Input shape of layers does not have to be specified when building the architecture, as this is inferred by the code. Output would then consist of another Dense layer with a number of neurons equal to the total number of pixels for the images.

B. *Trained Model*

The final architecture chosen consisted of a hidden layer with 50 neurons and an output layer with 3072 neurons, both using the logistic activation function. The cost function chosen was mean squared error, and the Adam optimizer with a learning rate of $\epsilon = 1 \times 10^{-3}$ was used. The final mean squared error across the 50 images was $E^2 = 0.0034$ after training.

C. *Image Denoising*

From the 50 images in the data set, 3 of them are randomly chosen. Gaussian noise with a mean $\mu = 0$ and a standard deviation $\sigma = 25$ is added to the data un-normalized data, and then put through the model to see if the denoised image is predicted. The mean squared errors between the noisy and original data, and the denoised and original data for each image are shown in Table I. The original, noisy and denoised images are shown in Figure 9. The car and frog images were very well reconstructed by the model, from a visual perspective, whereas the image of the duck is a lot more blurry and fuzzy in comparison.

ID	Noisy-Original MSE	Denoised-Original MSE
44	0.0091	0.0040
47	0.0096	0.0036
0	0.0095	0.0020

TABLE I
Mean squared error between the original image and its noisy and denoised versions.

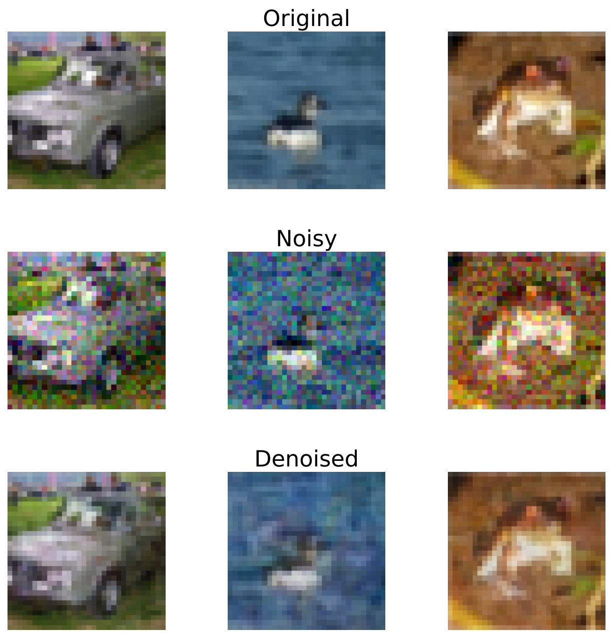

Fig. 9. Original, noisy and reconstructed versions of three random images from the CIFAR-10 data set.

VI. Clustering in PCA Space

For this section, the PCA class from the Scikit Learn library in Python was used to do Principal Component Analysis on the “spikes.csv” data set.

A. Tool Used: Scikit Learn

The Python library used to solve this problem was the Scikit Learn library [2]. It is the most popular library for machine learning routines in Python, having functions and classes for supervised and unsupervised learning, as well as for data preprocessing and model evaluation.

The specific function used was the PCA function, which is a part of the decomposition routines of the library. the most important options to consider are that of the number of components to output and the type of SVD solver to use to find the principal components. Other optional inputs are only considered depending on the type of solver used, and this is constrained to the more sophisticated solvers available. For this section, exact full SVD is used to find the principal components, and only the first three components are output.

B. PCA Projections: 1D, 2D and 3D

The top plot in Figure 10 shows the 200 neural traces projected onto the first principal component. With 1 principal component, 37.87% of the variance in the data is captured. Upon visual inspection, one can see 3 clusters of points, with the left-most cluster being the most separated, but an argument could be made for 2 clusters as well.

The bottom plot in Figure 10 shows the 200 neural traces projected onto the first 2 principal components. With 2 principal components, 57.30% of the variance in the data is captured. Visually, 3 distinct clusters with a high density of points can be seen, with a region between them have points largely spread out between them, which could seem like a fourth cluster.

Figure 11 shows the 200 neural traces projected onto the first 3 principal components. With 3 principal components, 69.52% of the variance in the data is captured. Visualization of the three clusters is much more clear in this case, with 2 of them being seen at the foreground and the other one in the background.

C. PCA Clustering

Using the K-Means algorithm from Project 1, the spikes are clustered using their first two principal components. Clusters ranging from 2 to 5 are tested by taking the average performance due to number of clusters from 100 trials. Performance is measured through the Silhouette score, which is a measure of cluster cohesion and separation [3]; the closer the metric is to equaling 1, the better defined the clusters are. The average Silhouette score across the 100 trials is plotted for the range of clusters in the top plot of Figure 12. The largest average score, as well as the largest potential given its uncertainty, is that of three clusters. The bottom plot in Figure 12 shows the clustering using the centroids with the highest silhouette score from the 100 trials.

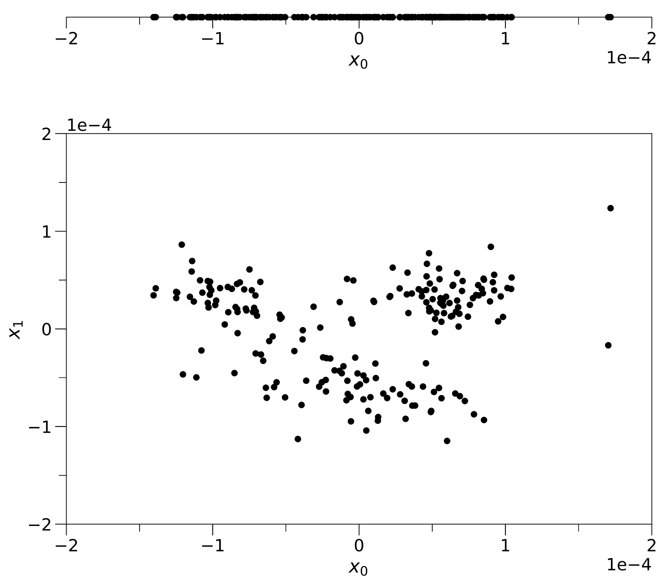

Fig. 10. Line plot for the first principal component (top) and scatter plot of the first two principal components (bottom) of the “spikes.csv” data set.

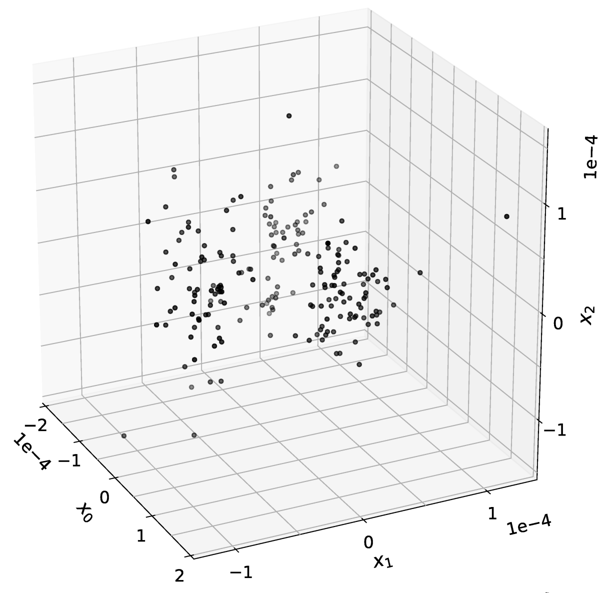

Fig. 11. 3D scatter plot for the first three principal components of the “spikes.csv” data set.

Given the groupings previously found, the spikes are plotted as time series with their respective group colors in Figure 13. There are three distinct set of spikes it seems: The blue spikes have a deep dip in signal on the left side of the peak, the green spikes have a deep dip on the right of the peak, and the red spikes have no dip on either side of the peak.

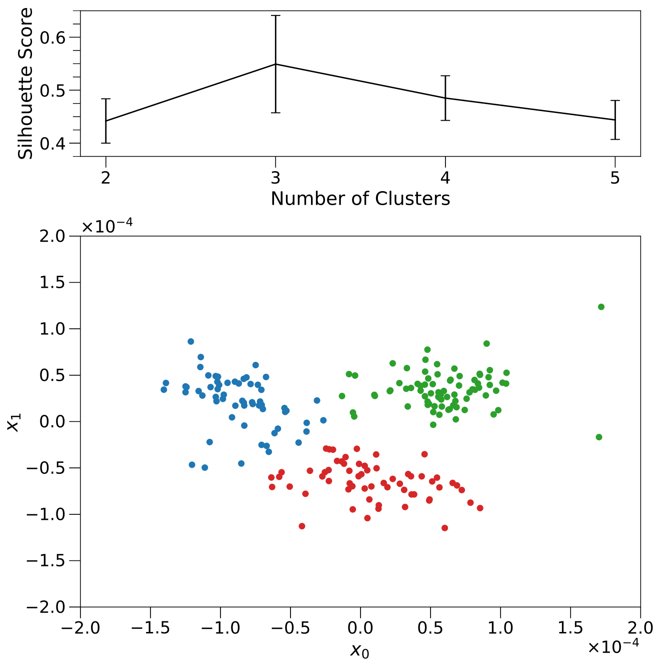

Fig. 12. Average silhouette score for different cluster sizes (top) and labeled scattered plot using the optimal cluster size of three (bottom) with the first two principal components of the "spikes.csv" data set.

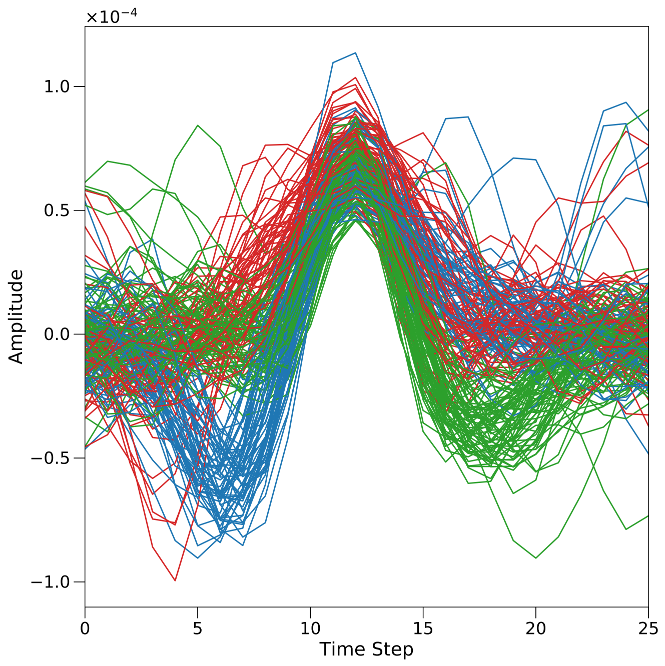

Fig. 13. Labeled plot using three clusters of the time series data from the "spikes.csv" data set.

VII. Conclusion

In this report, the Multilayer Perceptron was used to classify the Double Moon and Gaussian XOR data sets. The same distributions were used to find the early-stopping point for training of the MLP. Image denoising of the CIFAR-10 data set was done using an Autoencoder implemented through Keras. Using PCA, clustering of the time series data in the "spikes.csv" was done.

References

[1] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson, 2018.