Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

Classification with a single-layer perceptron

Nathan Ansel, Stefano Bosoppi, and Kusumastuti Cahyaningrum

January 23, 2025

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to implement and apply single-layer perceptrons, using either the classical perceptron or Widrow-Hoff delta rule learning algorithms, for classification problems.
- to configure and monitor the behavior of the aforementioned learning algorithms.
- to identify the limitations of single-layer perceptrons, their learning methods, and learning strategies (sequential and batch learning).

In this laboratory, we implemented and studied the properties of single-layer perceptrons, which employed the classical perceptron and Widrow-Hoff delta rule learning algorithms, for classification problems. We assumed a binary classification context with two-dimensional data to ease visualization.

2 Methods

We implemented and analyzed the perceptrons using Python scripts and Jupyter Notebooks. The former was used mainly to develop the perceptrons and data-generation functions, while the latter was used for visualization and performance measurement.

Since we developed the perceptrons from scratch, we used only the Python standard library and Numpy. This last library facilitates matrix operations, which are fundamental for implementing efficient versions of the models studied. As for visualization, we relied on the Matplotlib library, which provides extensive tools for visualizing data and generating plots. The specific implementations are available at `https://github.com/nathananse128/DD2437/tree/main/lab1a`.

We started by implementing the functions for generating the two data clusters, consisting of two-dimensional data, our models needed to classify. We chose (4.0, -2.0) and (-2.0, 3.0) as the reference points for the linearly separable clusters and set the standard deviations to 3.5 and 5, respectively. On the other hand, for the non-linearly separable clusters, we selected (2.0, -2.0) and (2.0, -1.0) with standard deviations of 4.5 and 5.5, respectively. The random seed for the point generation was fixed in both cases to ensure applicability.

After this, we compared the convergence speed, in number of epochs, of the perceptrons using the classical perceptron and the delta rule learning algorithms, trained with sequential learning, under different learning rate values. Then, we analyzed how sequential and batch learning, under different learning rates, impacts the convergence speed of a perceptron using the delta rule as its learning algorithm. By changing the seed in the random initialization of the weights, we also studied how different starting points affect the speed of the optimization process. In addition, we tested our hypothesis that perceptron learning without bias would converge only if the data is separable by a hyperplane, in this setting a line, passing through the origin. We did so by changing the reference points mentioned in the previous paragraph.

Finally, we considered the performance of the learning algorithms on non-linearly separable data. We trained the classic perceptron using sequential learning and the delta rule perceptron using batch learning, which have respectively proven to be the best training approaches for the two models. We first used the previously described non-linearly separable clusters and studied their performance. For this, we both plotted the decision boundary and measured the accuracy, sensitivity, and specificity of the two models. After this, we generated the clusters according to the instructions and repeated the measurements from the previous step.

3 Results and discussion

3.1 Classification with a single-layer perceptron

We considered the convergence speed of the two learning algorithms under study, trained with sequential learning, under different learning rates. In particular, we used the training error to determine when, and how rapidly, a specific algorithm converged. We represented the two error trends in figure 1.

Even though sequential learning is not the best training approach for this model, the delta rule converges fastest for each learning rate. We attributed this behavior to the delta rule using gradient descent to update weights, which minimizes the error function for faster convergence. Unlike perceptron learning, it adjusts weights for every sample, not just the misclassified ones, ensuring that all contribute to error reduction. In addition, weight updates are proportional to the size of the error, allowing larger corrections for larger misclassifications.

We also noticed that the error abruptly drops to zero when the perceptron converges. As a matter of fact, in figure 1 the convergence happens from one epoch to the next. This is likely because the classic perceptron algorithm updates weights based on misclassifications, correcting them for each

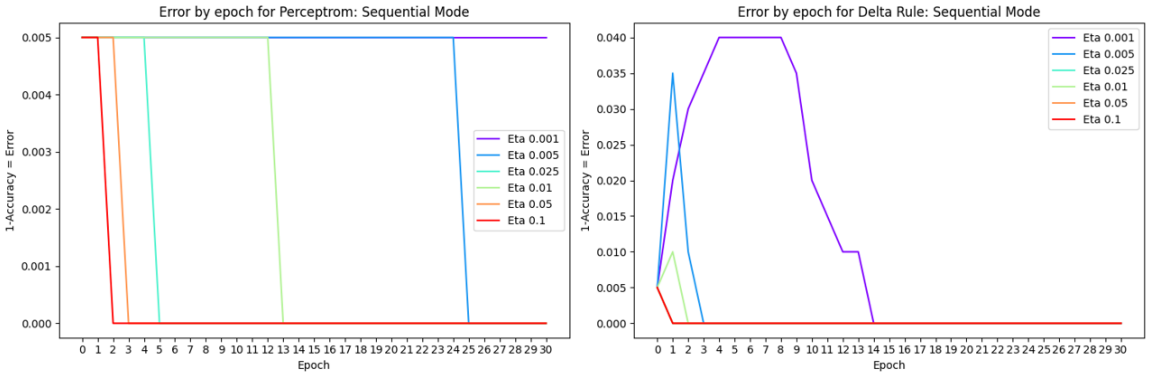

Figure 1: Comparison of the convergence speed, in number of epochs, of perceptrons using the classical perceptron and delta rule learning algorithms. Each line corresponds to a different learning rate value.

misclassified sample. Unlike methods that minimize a continuous error function, it adjusts weights to create a decision boundary that separates the data.

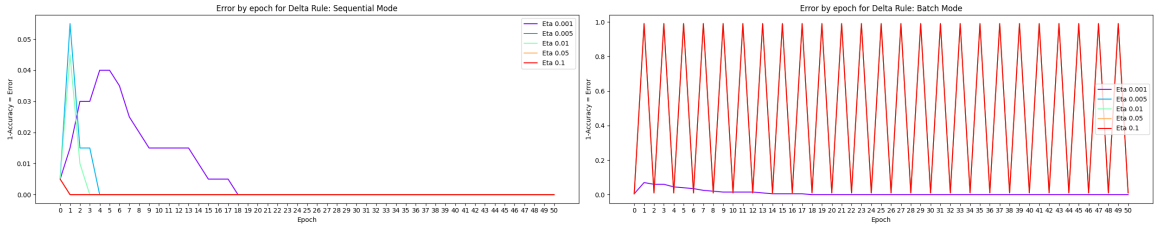

Figure 2: Comparison of the convergence speed, in number of epochs, of delta rule learning using sequential mode vs batch mode. Each line corresponds to a different learning rate value.

With different learning rates, the delta rule converges faster initially in sequential mode (online learning), as shown on the left side of Figure 2, because weights are updated after each sample, allowing quick adaptation. However, it is sensitive to noise and data order, leading to potential oscillations and requiring more iterations. Batch mode, on the other hand, updates weights after processing the entire dataset in each epoch, offering more stable and consistent convergence with fewer epochs but slower initial progress. When the learning rate is too high, it may show divergence, with oscillations in learning preventing convergence, as illustrated on the right side of Figure 2.

In delta rule learning, both sequential and batch mode converges almost at the same time with different random weight initializations, as illustrated in Figure 3. Unlike sequential mode, where weights are updated after each example, batch mode waits until all examples in the dataset are processed before applying a single update. This delay in adjustment means that the large initial error persists across the first epoch, contributing to the noticeable spike. This typically leads to a significant reduction in error in subsequent epochs, smoothing out the learning curve. This rapid decrease is due to the model correcting the initial misalignment of weights with the target values.

The bias term, as shown in Figure 4, is crucial in neural networks as it allows the model to handle

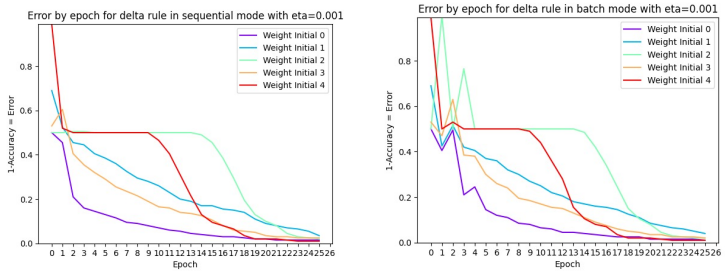

Figure 3: Convergence on different random weight initializations

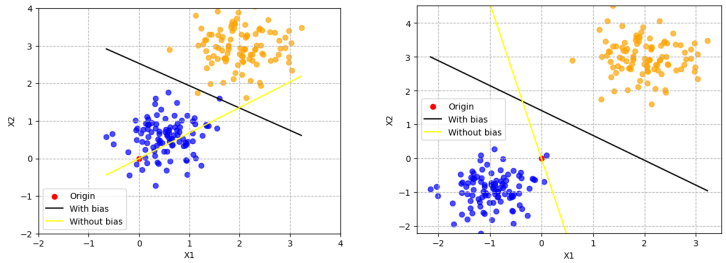

Figure 4: Effects of removing bias in delta rule learning in batch mode

data distributions that are not symmetrically centered around the origin. For two labels on the same side of the origin, the bias shifts the decision boundary away from the origin, enhancing the network's ability to classify them accurately. Without a bias term, the network can still classify the labels if they are linearly separable, but the decision boundary is constrained to pass through the origin, limiting the model's flexibility. Despite this limitation, the model may still perform adequately if the data distribution aligns well with this restriction.

3.2 Classification of data that are not linearly

3.2.1 1st XOR Example

In the first XOR example, we generated a non-linearly separable dataset and trained a linear neural network using the perceptron learning rule and delta learning rule (Fig. 5).
It can be seen that both methods fail to solve the XOR problem effectively as a linear neural network is unfit for separating non-linearly separated data samples. This is reflected from the relatively low accuracy, sensitivity and specificity rates in Table 1, but also visually seen from the plot in Fig. 5.

Learning Rule	Accuracy	Sensitivity	Specificity
Perceptron	0.70	0.79	0.62
Delta	0.72	0.68	0.77

Table 1: Metrics for the 1st XOR Problem

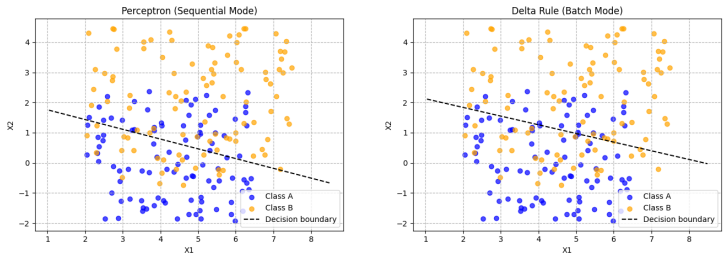

Figure 5: Decision Boundaries using Perceptron and Delta Rules

3.2.2 2nd XOR Example

Here, we generated another non-linearly separable dataset as can be seen in Fig. 6.

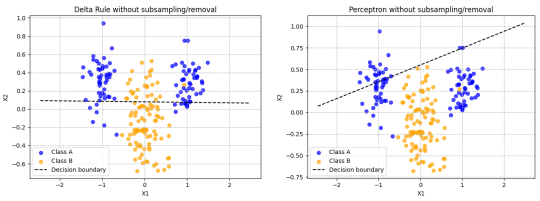

Figure 6: Decision boundaries using Delta and Perceptron Rules

We then performed the subsampling manipulations as stipulated in the instructions, obtaining the decision boundaries (as seen in Fig. 7) and the accuracy rates (as seen in Table 2).

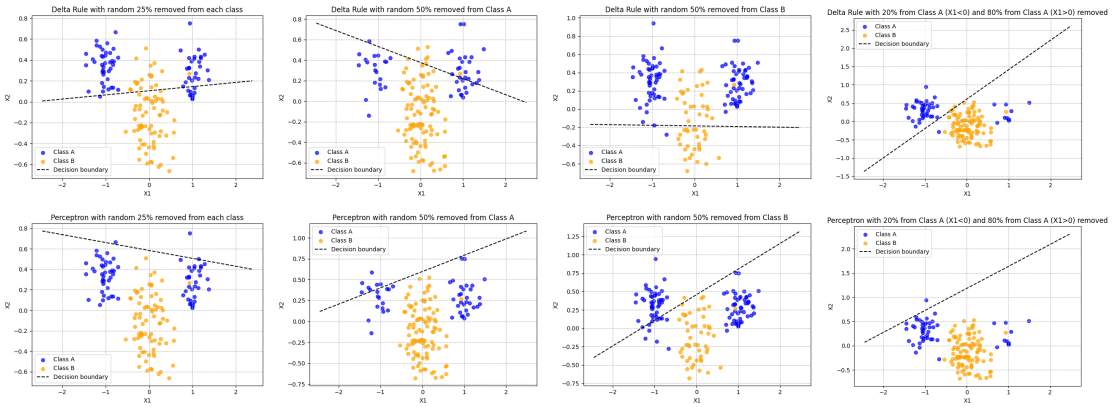

Figure 7: Decision boundaries using Delta and Perceptron Rules for the Subsampled Datasets

There are a few conclusions from this experiment. Firstly, subsampling does not significantly improve the accuracy of the network; the networks are still unfit in solving the XOR problem. Secondly, subsampling may introduce bias towards a specific class—which in turn heavily affect the decision boundaries—as seen especially in cases 2 and 3. Finally, it is evident that the delta rule performs better than the perceptron rule in these scenarios thanks to gradient descent. Due to gradient descent, the delta rule yielded a relatively high accuracy without getting stuck in a local minimum (even though the dataset is non-linearly separable). The perceptron rule on the other hand was

	Perceptron					**Delta**				
	Original	1	2	3	4	Original	1	2	3	4
Accuracy	0.80	0.81	0.76	0.83	0.92	0.62	0.53	0.71	0.60	0.67
Sensitivity	0.74	0.77	0.94	0.50	0.99	1.00	1.00	1.00	0.98	1.00
Specificity	0.86	0.86	0.40	0.99	0.78	0.25	0.04	0.14	0.41	0.02

Table 2: Accuracy, Sensitivity, and Specificity of the Datasets using Perceptron and Delta Rules. Subsample *1* refers to the one with 25% of observations removed from each class. Subsample *2* is the one with 50% of samples removed from class A. Similarly, *3* is for the one with 50% less data points for class B. Finally, *4* refers to the subsample where we removed 20% from the data points with $X1 < 0$ and 80% from the data points with $X1 > 0$.

unable to find a good solution as it was stuck in a local minimum due to its "discrete" way of learning.

4 Final remarks

This lab assignment helped us understand how single-layer perceptrons work. It was also a great opportunity to gain a practical understanding of the inner workings of a perceptron. Through this hands-on experience, we were able to deepen our understanding of concepts such as delta and perceptron learning rules, sequential and batch learning, as well as the strengths and limitations of single-layer perceptrons.

In addition, we found it very useful to experiment on our own with the effects of certain conditions (such as weight initialization, learning rate value, or the presence/absence of bias) on these learning algorithms. It helped us solidify the observations we made in class, grounding them with practical examples.