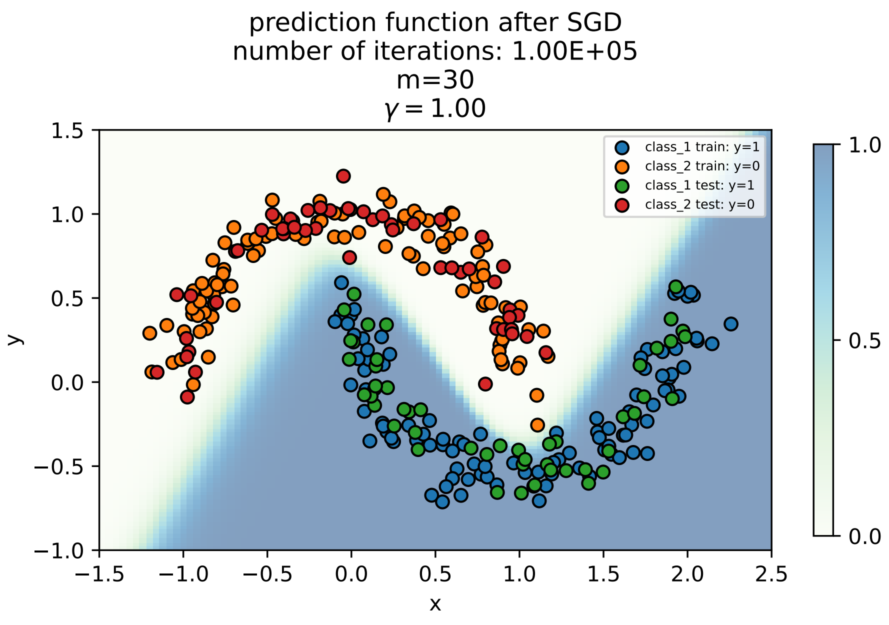
# PTML 6: 02/06/2022



## TABLE DES MATIÈRES

# 1 INFLUENCE ON DATA SCALING ON THE CONVERGENCE OF SGD

## 1.1 Introduction

We would like to perform a binary clasification on the following 3 datasets (figures 1, 2, 3). Each one has a different difficulty for a linear classifier, such as a SVM. We also note that the scales different for some axes.

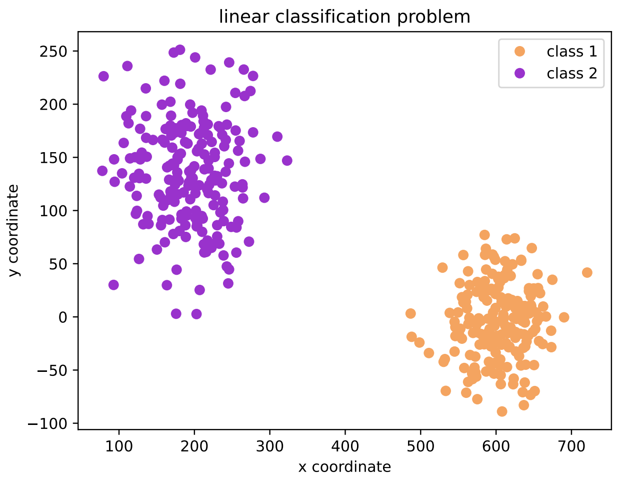


Figure 1 – Dataset 1

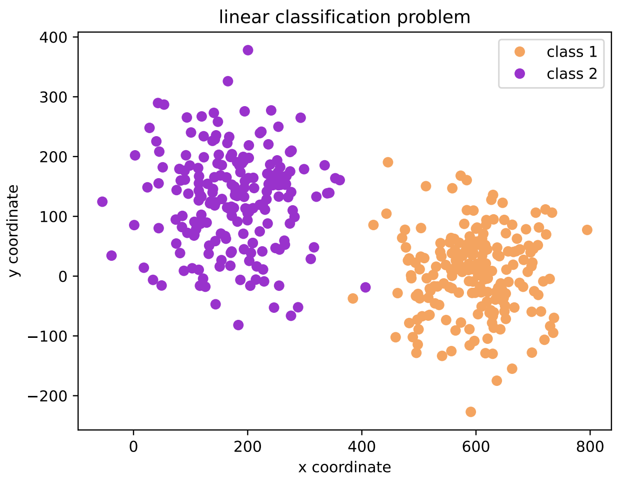


Figure 2 – Dataset 2

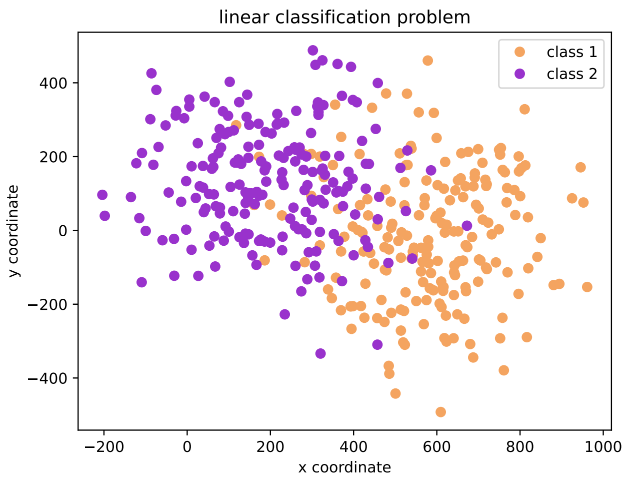


FIGURE 3 – Dataset 3

The data are located in **TP6/data_svm/**.

### 1.2 Exercise

Data standardization consists in transforming the data so that each feature (each column) is centered (zero mean) and has a variance equal to 1.
https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler.

It is experimentally often noticed that algorithms trained by SGD give a better performance (generalization error) when the data are standardized. Intuitively, a bad case where the data are not standardized is when one of the features is orders of magnitude larger than the other features. A optimizer might then only concentraite on optimizing this feature, although this might only come from the fact that this feature was measured in a different unit than others, and in fact does not carry more information on the problem than other features.

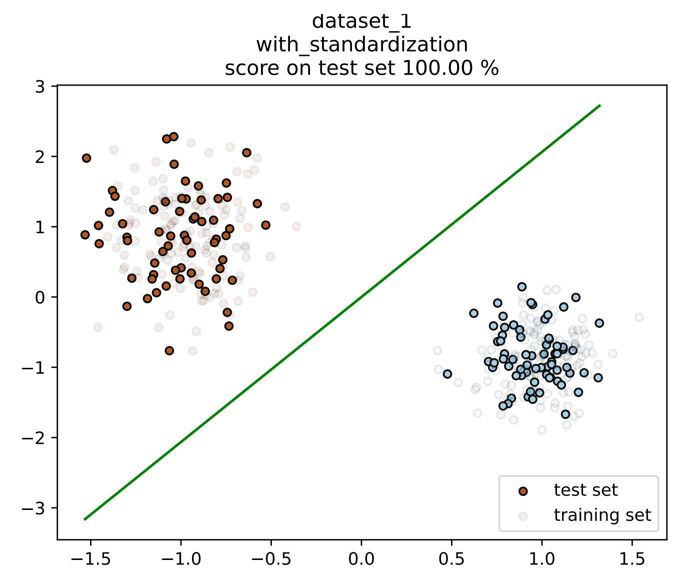
Exercice 1 : Perform a linear classification on these data, optimized by SGD, and compare quality of the results with and without preprocessing the data by standardizing them. You may use scikit-lean to do it, in particular :

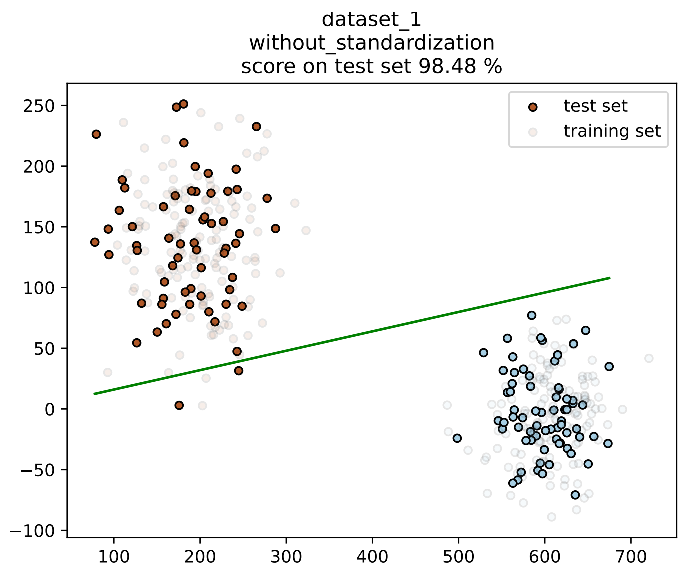
— https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html. By default, this class trains a liner SVM (no kernel).

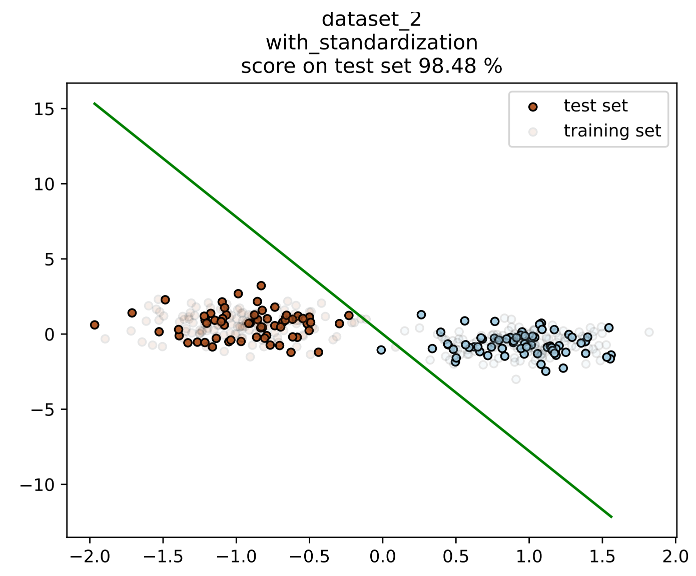
— https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

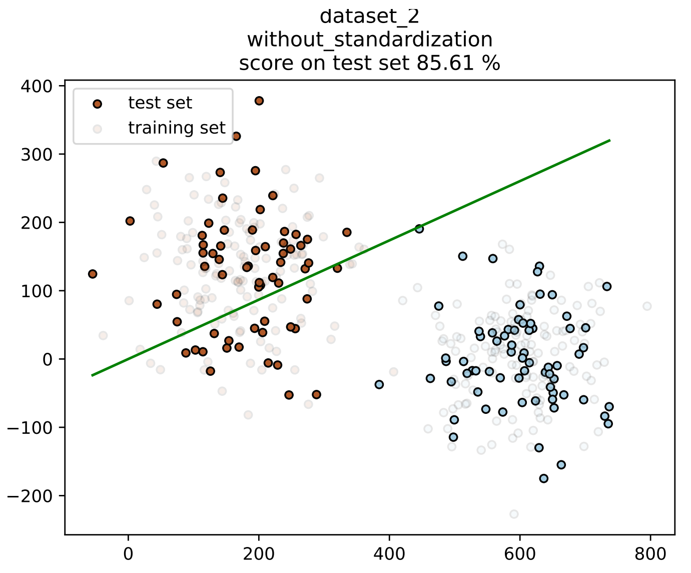
— https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

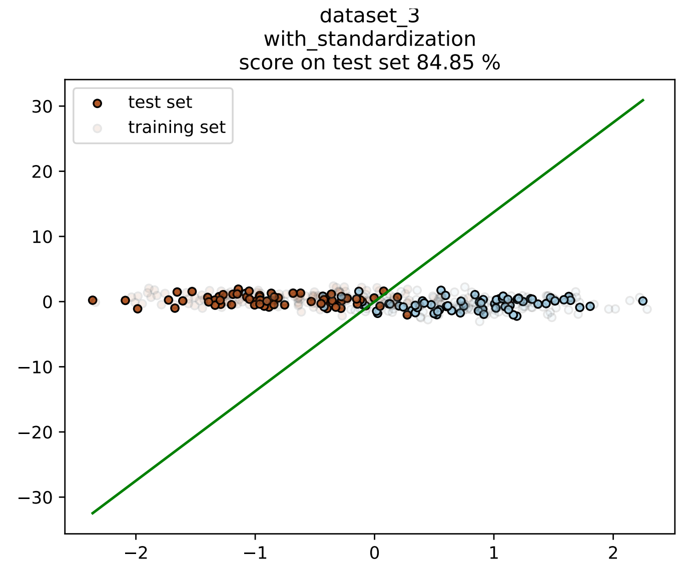
You should obtain results like the folling figures.

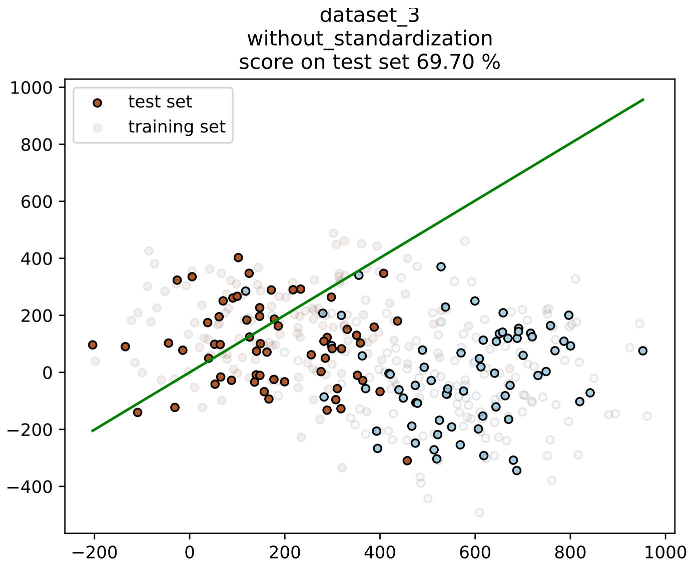

dataset_1
with_standardization
score on test set 100.00 %
test set
training set


dataset_1
without_standardization
score on test set 98.48 %
test set
training set


dataset_2
with_standardization
score on test set 98.48 %
test set
training set


dataset_2
without_standardization
score on test set 85.61 %
test set
training set
400
300
200
100
0
−100
−200
0
200
400
600
800


dataset_3
with_standardization
score on test set 84.85 %
test set
training set
30
20
10
0
−10
−20
−30
−2
−1
0
1
2


dataset_3
without_standardization
score on test set 69.70 %
test set
training set
1000
800
600
400
200
0
−200
−400
−200
0
200
400
600
800
1000

### 1.3 Conclusion

In this example, we saw that data standardization may give an improved performance, on a linear SVM trained by SGD.
You can do the same exercise by using a dataset in higher dimension.

## 2 ONE HIDDEN LAYER NEURAL NETWORK

### 2.1 Introduction

The goal of this exercise is to implement a simple neural network manually, without using third-party libraries (but for loading the dataset). The setting is a binary classification problem, on a dataset that is not linearly separable. Hence, a linear classification method such a vanilla logistic regression or support vector machine would not work well.

The file to use is **TP_6_NN.py**. The dataset is loaded at the beginning of the file.

— Section 2.2 decribes the architecture of the neural network.

— Section 2.3 presents the computation of the gradients for backpropagation

— Section 2.5 contains the steps of the exercise.

Many calculations are detailed but you can use the results in order to implement the algorithm. However, to fully understand the method, you are encouraged to do the calculations yourselves.

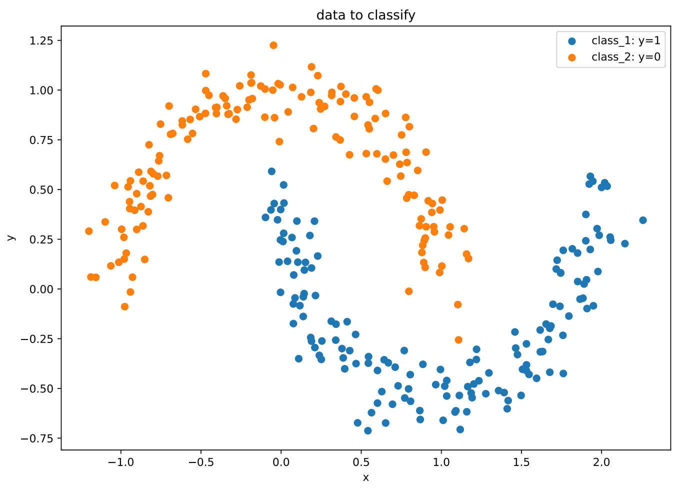


FIGURE 4 – Data to classify. Note that they are not linearly separable.

### 2.2 Neural network description

We will use a setting similar (but not exactly similar) to that of lecture 10, which is as follows :

— $\mathcal{X} = \mathbb{R}^d$.

— $\mathcal{Y} = \{0, 1\}$.

- In order to add an intercept (a constant term added to the linear combinations inside the sigmoids), we can add a component to the inputs $x \in \mathbb{R}^d$ and build a vector $(x, 1) \in \mathbb{R}^{d+1}$. The same applies to the hidden layer.
- The hidden layer contains $m$ neurons. Thus, the matrix $w_h$ containing the weigths between the input layer and the hidden layer can be either in $\mathbb{R}^{m,d+1}$, either in $\mathbb{R}^{d+1,m}$, depending on the convention. In this exercice we will consider it to be in $\mathbb{R}^{m,d+1}$. Remember that the $m+1$ and the $d+1$ come from the previous point. This is practical to have lighter notations.
- the activation function used is the sigmoid, defined as :

$$\forall x \in \mathbb{R}, \sigma(x) = \frac{1}{1+e^{-x}} \tag{1}$$

  We have seen that its derivative verifies :

$$\forall z \in \mathbb{R}, \sigma'(z) = \sigma(z)\sigma(-z) \tag{2}$$

- The output of the network is a single real number, hence the matrix $\theta$ containing the weights between the hidden layer and the output layer is just a vector in $\mathbb{R}^{m+1}$.
- We use the squared loss in order to compare our prediction $\hat{y_i} \in \mathbb{R}$ for input $x_i$ to the label $y_i \in \{0, 1\}$ for the same input.

$$l(\hat{y_i}, y_i) = (\hat{y_i} - y_i)^2 \tag{3}$$

  **Remark.** *Although we have not underlined it during the course, using the squared loss for classification is a valid option, even if it is often used for regression.*
- Let $\theta_j \in \mathbb{R}$ be the component $j$ of $\theta$ and $L_j \in \mathbb{R}^{d+1}$ be the line $j$ of $w_h$. The prediction ( **forward pass**) of the neural network for input $x \in \mathbb{R}^{d+1}$ is then :

$$\hat{y} = f(x) = \sigma\Big(\sum_{j=1}^{m} \theta_j \sigma(\langle L_j, x\rangle) + \theta_{m+1}\Big) \tag{4}$$

  Equivalently, if $h \in \mathbb{R}^{m+1}$ is the vector representing the input layer computed from $x$,

$$\hat{y} = f(x) = \sigma(\langle \theta, h\rangle) \tag{5}$$

## 2.3 Gradients and backpropagation

The neural network will be trained by SGD. Thus, we need to compute the gradient of the loss $l_i$ for each sample $(x_i, y_i)$.

$$l_i = \frac{1}{2}(f(x_i) - y_i)^2 = \frac{1}{2}(\hat{y_i} - y_i)^2 \tag{6}$$

There will be a gradient with respect to $\theta$, noted $\nabla_\theta l_i$, and a gradient with respect to $w_h$, noted $\nabla_{w_h} l_i$.

We drop the $i$ index for simplicity, so the calculation is performed for a given input $x \in \mathbb{R}^d$, that outputs a prediction $\hat{y} \in \mathbb{R}$, with a hidden layer $h \in \mathbb{R}^m$.

### 2.3.1 *Gradient with respect to* $\theta$

By the composition of

$$\begin{cases} \mathbb{R} \to \mathbb{R} \\ \hat{y} \mapsto \frac{1}{2}(\hat{y} - y)^2 \end{cases}$$

and

$$\begin{cases} \mathbb{R}^{m+1} \to \mathbb{R} \\ \theta \mapsto \sigma(\langle \theta, h \rangle) \end{cases}$$

we get that

$$\nabla_\theta l(\theta, w_h) = (y - \hat{y})\sigma'(\langle \theta, h \rangle) h \in \mathbb{R}^{m+1} \tag{7}$$

### 2.3.2 *Gradient with respect to $w_h$*

We can compute the gradient with respect to each $L_j \in \mathbb{R}^{d+1}$.

We first need to compute the gradient of each component $h_j \in \mathbb{R}$ of the hidden layer $h$ with respect to $L_j$. We have that

$$\nabla_{L_j} h_j(\theta, w_h) = \sigma'(\langle L_j, x \rangle) x \tag{8}$$

Remember that the gradient is the transpose of the jacobian in the special case where the function is real valued.

To obtain $\hat{y}$, we must compose two applications. The first one is :

$$\begin{cases} \mathbb{R}^{d+1} \to \mathbb{R}^{m+1} \\ L_j \mapsto h = \begin{pmatrix} \sigma(\langle L_1, x \rangle) \\ \dots \\ \sigma(\langle L_j, x \rangle) \\ \dots \\ \sigma(\langle L_m, x \rangle) \\ 1 \end{pmatrix} \end{cases}$$

Its jacobian is

$$\begin{pmatrix} \nabla^T_{L_j}\left(\sigma(\langle L_1, x \rangle)\right) \\ \dots \\ \nabla^T_{L_j}\left(\sigma(\langle L_j, x \rangle)\right) \\ \dots \\ \nabla^T_{L_j}\left(\sigma(\langle L_m, x \rangle)\right) \\ \nabla^T_{L_j} 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ \nabla^T_{L_j}\left(\sigma(\langle L_j, x \rangle)\right) \\ \dots \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ \sigma'(\langle L_j, x \rangle) x^T \\ \dots \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

The second application is

$$\begin{cases} \mathbb{R}^{m+1} \to \mathbb{R} \\ h \mapsto \sigma(\langle \theta, h \rangle) \end{cases}$$

Its gradient is $\sigma'(\langle \theta, h \rangle)\theta$. Hence, here the jacobian is

$$\sigma'(\langle \theta, h \rangle)\theta^T \tag{10}$$

Finally, the jacobian of $L_j \mapsto \hat{y}$ is the product of the two previous jacobians :

$$\sigma'(\langle \theta, h \rangle)\theta^T \begin{pmatrix} 0 \\ \dots \\ \sigma'(\langle L_j, x \rangle) x^T \\ \dots \\ 0 \\ 0 \end{pmatrix} = \sigma'(\langle \theta, h \rangle)\theta_j \sigma'(\langle L_j, x \rangle) x^T \tag{11}$$

and

$$\nabla_{L_j} \hat{y}(\theta, w_h) = \sigma'(\langle \theta, h \rangle)\theta_j \sigma'(\langle L_j, x \rangle) x \in \mathbb{R}^{d+1} \tag{12}$$

To conclude, we need to compose by

$$\begin{cases} \mathbb{R} \to \mathbb{R} \\ \hat{y} \mapsto \frac{1}{2}(\hat{y} - y)^2 \end{cases}$$

obtain that

$$\nabla_{L_j} l(\theta, w_h) = (\hat{y} - y)\sigma'(\langle \theta, h\rangle)\theta_j \sigma'(\langle L_j, x\rangle)x \in \mathbb{R}^{d+1} \tag{13}$$

## 2.4 The chain rule

The chain rule is a formal way of writing a product of jacobians in order to compute a gradient or a jacobian of a composition. For instance, if

— $\frac{\partial \hat{y}}{\partial h}$ denotes the jacobian of $h \mapsto \hat{y}$ ($\in \mathbb{R}^{(1,m)}$)

— $\frac{\partial l}{\partial \hat{y}}$ denotes the jacobian of $\hat{y} \mapsto l$ (which is just a derivative $\in \mathbb{R}$)

— $\frac{\partial l}{\partial h}$ denotes the jacobian of $h \mapsto l$ ($\in \mathbb{R}^{(1,m)}$)

Then

$$\frac{\partial l}{\partial h} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \tag{14}$$

and this product makes sense as a product of **matrices**. This formal way is also useful to decompose the calculation of the gradient. It is equivalent to the previous calculation and is often encountered.

## 2.5 Implementation

As you can see from equations 4, 5, 7, and 13, some quantities that are computed during the forward pass are useful in the backpropagation, such as $\langle \theta, h\rangle$. Thus, it is a good idea, when computing the forward pass, to return also these intermediate calculations.

For a given input $x \in \mathbb{R}^{d+1}$, we can consider the following intermediate variables :

— $\text{pre}_h = w_h x \in \mathbb{R}^{m,d+1}$. The vector from which the hidden layer $h$ is computed, by appliying $\sigma$.

— $h = \sigma(\text{pre}_h) \in \mathbb{R}^m$, where $\sigma$ is applied elementwise.

— $\text{pre}_y = \langle \theta, h\rangle \in \mathbb{R}$. The scalar from which the output $\hat{y}$ is computed by applying $\sigma$. Remember that this is computed after adding a $1$ to the last component of $h$, which size becomes $m+1$.

— $\hat{y} = \sigma(\text{pre}_y)$ : output of the forward pass.

We can write $\text{pre}_h$ explicitely.

$$\begin{pmatrix} \langle L_1, x\rangle \\ \dots \\ \langle L_j, x\rangle \\ \dots \\ \langle L_m, x\rangle \end{pmatrix} \tag{15}$$

Now, we can express the gradient with respect to $w_h$ directly, by reshaping $\nabla_{L_j} \hat{y}(\theta, w_h)$ to its transpose that is of shape $(1, d+1)$. If $\odot$ notes the elementwise product of two vectors of the same size, and $\theta'$ the $m$ first components of $\theta$.

$$\theta' \odot \text{pre}_h \in \mathbb{R}^m \tag{16}$$

$$\nabla_{w_h} l = (\hat{y} - y)\sigma'(\langle \theta, h\rangle)(\theta' \odot \text{pre}_h)x^T \in \mathbb{R}^{m \times (d+1)} \tag{17}$$

Equivalently, with the chain rule, we can also write :

$$\frac{\partial l}{\partial \theta} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \text{pre}_y} \frac{\partial \text{pre}_y}{\partial \theta} \tag{18}$$

and

$$\frac{\partial l}{\partial w_h} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial pre_y} \frac{\partial pre_y}{\partial h} \frac{\partial h}{\partial pre_h} \frac{\partial pre_h}{\partial w_h} \tag{19}$$

## 2.6 Simulation

In our example, $d = 2$. You can start with $m = 10$.
Perform the following steps :

— Exercice 2 : Implement the forward pass of the neural network (function **forward_pass.py**).

— Exercice 3 : Implement the computation of the gradient with respect to $w_h$ and $\theta$ for one given sample $(x_i, y_i)$.

— Exercice 4 : Implement an SGD on the training set and plot the train and test error as a function of the number of iterations. To save some time, it is not necessary to store and plot them at each iteration. As the objective function is not convex, we can not use our usual criteria for setting the learning rate $\gamma$, and must experiment to find a working value.

— Exercice 5 : Display the area that are assigned to each label by the predictor you obtained. The delimitation between the two zones will depend on the number of iterations.

You should observe results like in the following figures.

### 2.6.1 *Learning curves*

### 2.6.2 *Prediction functions*

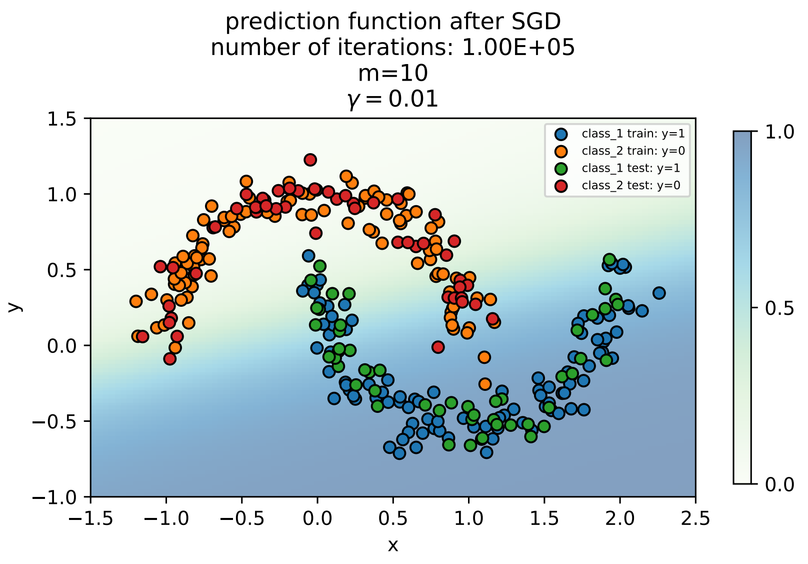


FIGURE 5 – Prediction function of the learned estimator after $10^4$ SGD iterations.

## 2.7 Conclusion

This simple neural network is able to solve a non linear classification problem.
You can explore the influence of several parameters, like $m$, and the learning on different datasets, on this page :

https://playground.tensorflow.org/

More information on the implementation of backpropagation and on the optimization of neural networks can be found for instance in [LeCun et al., 1998], where some important tricks and aspects of the model are discussed, such as : http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

- advantages and disadvantages of SGD over GD
- emphasizing schemes
- input normalization / standardization
- weight initialization
- learning rates

## RÉFÉRENCES

[LeCun et al., 1998] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient BackProp. pages 9–48.