

# Deep Learning (IST, 2022-23)

## Practical 2: Perceptron

André Martins, Andreas Wichert, Luis Sá-Couto

### Question 1

Consider the following linearly separable training set:

$$\mathbf{x}^{(1)} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \mathbf{x}^{(2)} = \begin{bmatrix} 0 \\ 0.25 \end{bmatrix}, \mathbf{x}^{(3)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{x}^{(4)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$
$$y^{(1)} = -1, y^{(2)} = +1, y^{(3)} = +1, y^{(4)} = -1.$$

1. Initialize all weights to zero (including the bias). Assume  $\text{sign}(z) = +1$  iff  $z \geq 0$ , and  $-1$  if  $z < 0$ . Use a learning rate of one. Apply the perceptron learning algorithm until convergence. How many epochs does it take to converge?

**Solution:** We start by adding a constant feature of 1 to the inputs, leading to:

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \mathbf{x}^{(2)} = \begin{bmatrix} 1 \\ 0 \\ 0.25 \end{bmatrix}, \mathbf{x}^{(3)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{x}^{(4)} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

The weight vector is initialized as  $\mathbf{w} = [0, 0, 0]$ , where the first dimension corresponds to the bias (intercept) parameter.

First epoch (3 mistakes in total):

- $\mathbf{w} = [0, 0, 0]^\top$
- $\hat{y}^{(1)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(1)}) = \text{sign}(0) = +1 \neq y^{(1)}$  – **mistake!**
- Perceptron update:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(1)} \mathbf{x}^{(1)} = [0, 0, 0]^\top - [1, -1, 0]^\top = [-1, 1, 0]^\top$
- $\hat{y}^{(2)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(2)}) = \text{sign}(-1) = -1 \neq y^{(2)}$  – **mistake!**
- Perceptron update:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(2)} \mathbf{x}^{(2)} = [-1, 1, 0]^\top + [1, 0, 0.25]^\top = [0, 1, 0.25]^\top$
- $\hat{y}^{(3)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(3)}) = \text{sign}(1.25) = +1 = y^{(3)}$  – **correct, no update.**
- $\hat{y}^{(4)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(4)}) = \text{sign}(0.75) = +1 \neq y^{(4)}$  – **mistake!**
- Perceptron update:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(4)} \mathbf{x}^{(4)} = [0, 1, 0.25]^\top - [1, 1, -1]^\top = [-1, 0, 1.25]^\top$

Second epoch (1 mistake in total):

- $\mathbf{w} = [-1, 0, 1.25]^\top$

- $\hat{y}^{(1)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(1)}) = \text{sign}(-1) = -1 = y^{(1)}$  – correct, no update.
- $\hat{y}^{(2)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(2)}) = \text{sign}(-0.6875) = -1 \neq y^{(2)}$  – mistake!
- Perceptron update:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(2)} \mathbf{x}^{(2)} = [-1, 0, 1.25]^\top + [1, 0, 0.25]^\top = [0, 0, 1.5]^\top$
- $\hat{y}^{(3)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(3)}) = \text{sign}(1.5) = +1 = y^{(3)}$  – correct, no update.
- $\hat{y}^{(4)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(4)}) = \text{sign}(-1.5) = -1 = y^{(4)}$  – correct, no update.

Third epoch (2 mistakes in total):

- $\mathbf{w} = [0, 0, 1.5]^\top$
- $\hat{y}^{(1)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(1)}) = \text{sign}(0) = 1 \neq y^{(1)}$  – mistake!
- Perceptron update:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(1)} \mathbf{x}^{(1)} = [0, 0, 1.5]^\top - [1, -1, 0]^\top = [-1, 1, 1.5]^\top$
- $\hat{y}^{(2)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(2)}) = \text{sign}(-0.625) = -1 \neq y^{(2)}$  – mistake!
- Perceptron update:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(2)} \mathbf{x}^{(2)} = [-1, 1, 1.5]^\top + [1, 0, 0.25]^\top = [0, 1, 1.75]^\top$
- $\hat{y}^{(3)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(3)}) = \text{sign}(2.75) = +1 = y^{(3)}$  – correct, no update.
- $\hat{y}^{(4)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(4)}) = \text{sign}(-0.75) = -1 = y^{(4)}$  – correct, no update.

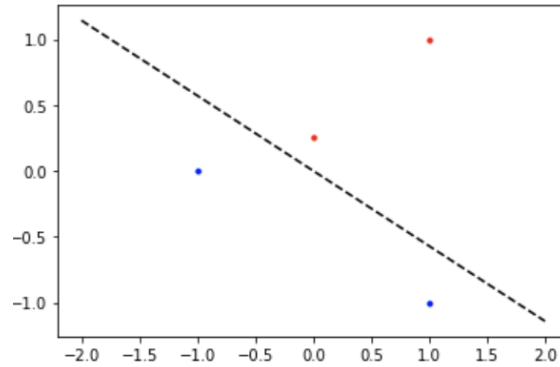
Fourth epoch (0 mistakes in total):

- $\mathbf{w} = [0, 1, 1.75]^\top$
- $\hat{y}^{(1)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(1)}) = \text{sign}(-1) = -1 = y^{(1)}$  – correct, no update.
- $\hat{y}^{(2)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(2)}) = \text{sign}(0.4375) = 1 = y^{(2)}$  – correct, no update.
- $\hat{y}^{(3)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(3)}) = \text{sign}(2.75) = +1 = y^{(3)}$  – correct, no update.
- $\hat{y}^{(4)} = \text{sign}(\mathbf{w}^\top \mathbf{x}^{(4)}) = \text{sign}(-0.75) = -1 = y^{(4)}$  – correct, no update.

Since there were no updates for a full epoch, no updates would happen for subsequent epochs, i.e., the perceptron algorithm converged in 3 epochs with 6 mistakes in total. The trained weight vector is  $\mathbf{w} = [0, 1, 1.75]^\top$ .

2. Draw the separation hyperplane.

**Solution:** The trained weight vector is  $\mathbf{w} = [0, 1, 1.75]^\top$ , resulting in the following separating boundary, which corresponds to the line with equation  $w_0 + w_1x_1 + w_2x_2 = 0$ :

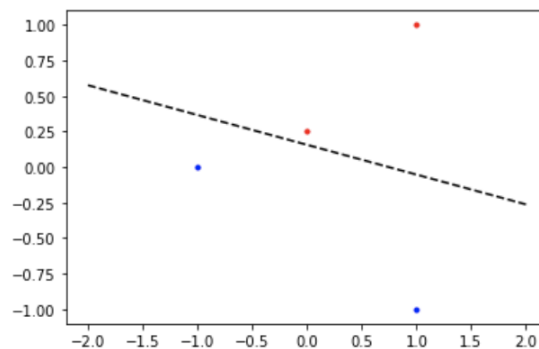


3. What is the perceptron output for the query point  $[0 \ 1]^\top$ ?

**Solution:** We have  $\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x}_{\text{new}}) = \text{sign}([0, 1, 1.75]^\top [1, 0, 1]) = \text{sign}(1.75) = +1$ .

4. Change the initialization of weights and biases to be random with a standard normal distribution  $\mathcal{N}(0, 1)$ . Try multiple times. Does it always converge?

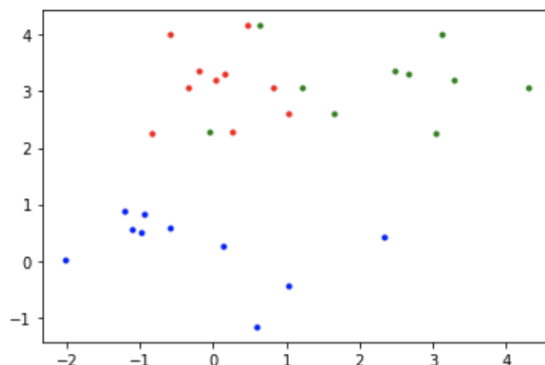
**Solution:** The data is separable, hence, due to the mistake bound, after a sufficient number of epochs, the perceptron will always converge. However, the separating hyperplane may be different. Here is an example –  $\mathbf{w} = [-0.20824551, 0.27894765, 1.33227645]^\top$ :



## Question 2

1. Generate a balanced dataset with 30 examples in  $\mathbb{R}^2$  and 3 classes. Assume each of the 10 inputs associated to class  $k \in \{0, 1, 2\}$  is generated as  $x \sim \mathcal{N}(\mu_k, \sigma_k^2 I)$ , with  $\sigma_0 = \sigma_1 = \sigma_2 = 1$ ,  $\mu_0 = [0, 0]^\top$ ,  $\mu_1 = [0, 3]^\top$ , and  $\mu_2 = [2, 2]^\top$ . Plot the data.

**Solution:** Here is an example of a plot (this may vary, since data is generated randomly):



2. Implement the multi-class perceptron algorithm and run 100 iterations. Initialize all the weights to zero and use a learning rate of one. What is the training accuracy (fraction of points that are correctly classified)?

**Solution:** With the data above, we get a training accuracy of 86.7%. Of course, depending on the generated dataset, the problem can be easier or harder.

### Question 3

The perceptron can learn a relatively large number of functions. In this exercise, we focus on simple logical functions.

1. Show graphically that a perceptron can learn the logical NOT function. Give an example with specific weights.

**Solution:** The NOT function receives as input a logical value  $x \in \{-1, +1\}$  and outputs its logical negation  $y \in \{-1, +1\}$ . We can enumerate all possible inputs and their outputs:

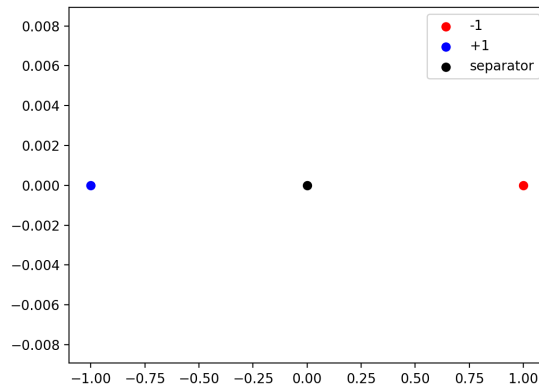
- For  $x = -1$  the output is  $y = +1$
- For  $x = +1$  the output is  $y = -1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output (+1) can be separated from all points that require a negative output by an hyperplane.

Since we are working with 1-dimensional inputs, an hyperplane is, in this case, a point.

So, is there a point that accurately separates the points? Yes, any point between  $-1$  and  $+1$  will achieve this.

An example hyperplane is given by the weight  $-1$  and bias  $0$ , leading to  $\hat{y} = -x$ .



2. Show graphically that a perceptron can learn the logical AND function for two inputs. Give an example with specific weights.

**Solution:** The AND function receives as input a pair of logical values  $\mathbf{x} \in \mathbb{R}^2$  and outputs its logical conjunction  $y \in \{-1, +1\}$ . We can enumerate all possible inputs and their outputs:

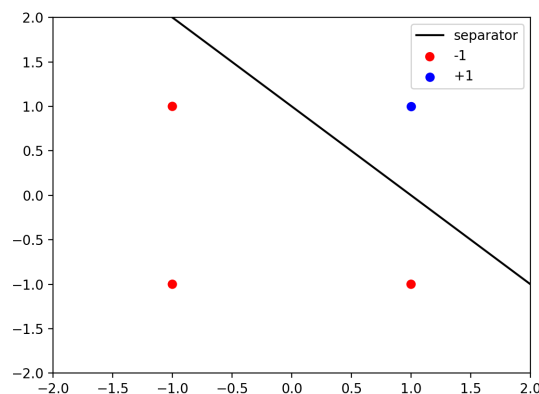
- For  $\mathbf{x} = \begin{bmatrix} -1 & -1 \end{bmatrix}^\top$  the output is  $y = -1$
- For  $\mathbf{x} = \begin{bmatrix} -1 & +1 \end{bmatrix}^\top$  the output is  $y = -1$
- For  $\mathbf{x} = \begin{bmatrix} +1 & -1 \end{bmatrix}^\top$  the output is  $y = -1$
- For  $\mathbf{x} = \begin{bmatrix} +1 & +1 \end{bmatrix}^\top$  the output is  $y = +1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output (+1) can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2-dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

Yes, for instance  $\mathbf{w} = \begin{bmatrix} -1 & 1 & 1 \end{bmatrix}^\top$  achieves:



3. Show graphically that a perceptron can learn the logical OR function for two inputs. Give an example with specific weights.

**Solution:** The OR function receives as input a pair of logical values  $\mathbf{x} \in \mathbb{R}^2$  and outputs its logical conjunction  $y \in \{-1, +1\}$ . We can enumerate all possible inputs and their outputs:

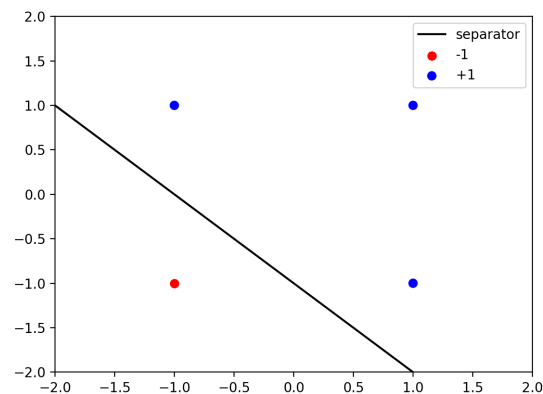
- For  $\mathbf{x} = \begin{bmatrix} -1 & -1 \end{bmatrix}^\top$  the output is  $y = -1$
- For  $\mathbf{x} = \begin{bmatrix} -1 & +1 \end{bmatrix}^\top$  the output is  $y = +1$
- For  $\mathbf{x} = \begin{bmatrix} +1 & -1 \end{bmatrix}^\top$  the output is  $y = +1$
- For  $\mathbf{x} = \begin{bmatrix} +1 & +1 \end{bmatrix}^\top$  the output is  $y = +1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output (+1) can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2-dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

Yes, for instance  $\mathbf{w} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^\top$  achieves:



4. Show graphically that a perceptron can not learn the logical XOR function for two inputs.

**Solution:** The XOR function receives as input a pair of logical values  $\mathbf{x} \in \mathbb{R}^2$  and outputs its logical conjunction  $y \in \{-1, +1\}$ . We can enumerate all possible inputs and their outputs:

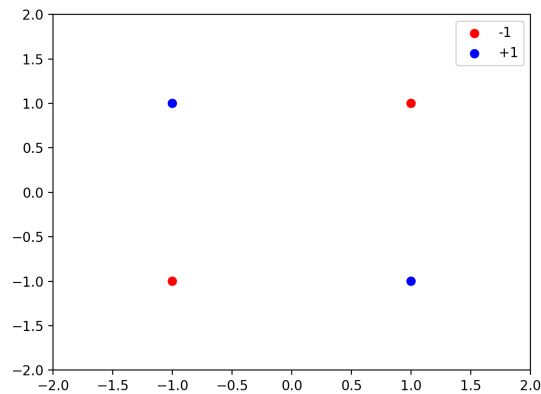
- For  $\mathbf{x} = \begin{bmatrix} -1 & -1 \end{bmatrix}^\top$  the output is  $y = -1$
- For  $\mathbf{x} = \begin{bmatrix} -1 & +1 \end{bmatrix}^\top$  the output is  $y = +1$
- For  $\mathbf{x} = \begin{bmatrix} +1 & -1 \end{bmatrix}^\top$  the output is  $y = +1$
- For  $\mathbf{x} = \begin{bmatrix} +1 & +1 \end{bmatrix}^\top$  the output is  $y = -1$

To show that a perceptron can learn a given function we just need to show that all points that require a positive output (+1) can be separated from all points that require a negative output by an hyperplane.

Since we are working with 2-dimensional inputs, an hyperplane is, in this case, a line.

So, is there a weight vector that defines a boundary line that accurately separates the points?

No... If we plot the points we see right away that no line can separate the positive instances from the negative ones:



## Question 4

Now it's time to try the perceptron on real data and see what happens.

1. Load the UCI handwritten digits dataset using `scikit-learn`:

```
from sklearn.datasets import load_digits
data = load_digits()
```

This is a dataset containing 1797 8x8 input images of digits, each corresponding to one out of 10 output classes. You can print the dataset description and visualize some input examples with:

```
print(data.DESCR)

import matplotlib.pyplot as plt
plt.gray()
for i in range(10):
    plt.matshow(data.images[i])
plt.show()
```

Randomly split this data into training (80%) and test (20%) partitions. This can be done with:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

2. Run your implementation of the multi-class perceptron algorithm on this dataset. Measure the training and test accuracy.

**Solution:** We got a training accuracy of 100% and a test accuracy of 95.3%. But results may differ depending on the random split.

3. Use `scikit-learn`'s implementation of the perceptron algorithm. This can be done with

```
from sklearn.linear_model import Perceptron
clf = Perceptron(fit_intercept=False, shuffle=False)
clf.fit(X_train, y_train)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))
```

Compare the resulting accuracies.

**Solution:** Results were slightly different, with a training accuracy of 97.4% and a test accuracy of 95.8%, possibly due to small differences in the implementation or data shuffling.