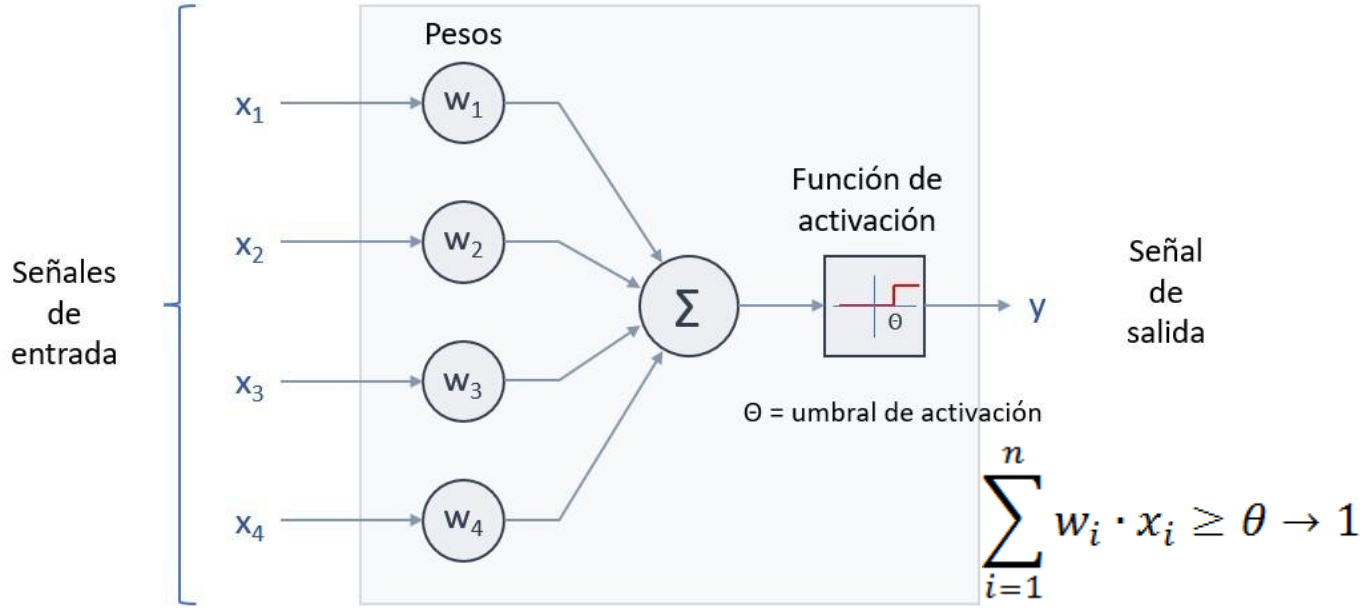
❑ DEEP LEARNING. REDES NEURONALES.

La primera neurona artificial fue desarrollada en 1943 por el neurólogo Warren Sturgis McCulloch y el matemático Walter Harry Pitts: la conocida como "Neurona de McCulloch-Pitts".

Este desarrollo fue mejorado por el psicólogo Frank Rosenblatt en 1958, siendo su neurona conocida como "Perceptrón".

❑ DEEP LEARNING. Perceptrón.



Fuente: https://interactivechaos.com

- Los pesos se fijan durante la configuración de la neurona.
- La salida es binaria.

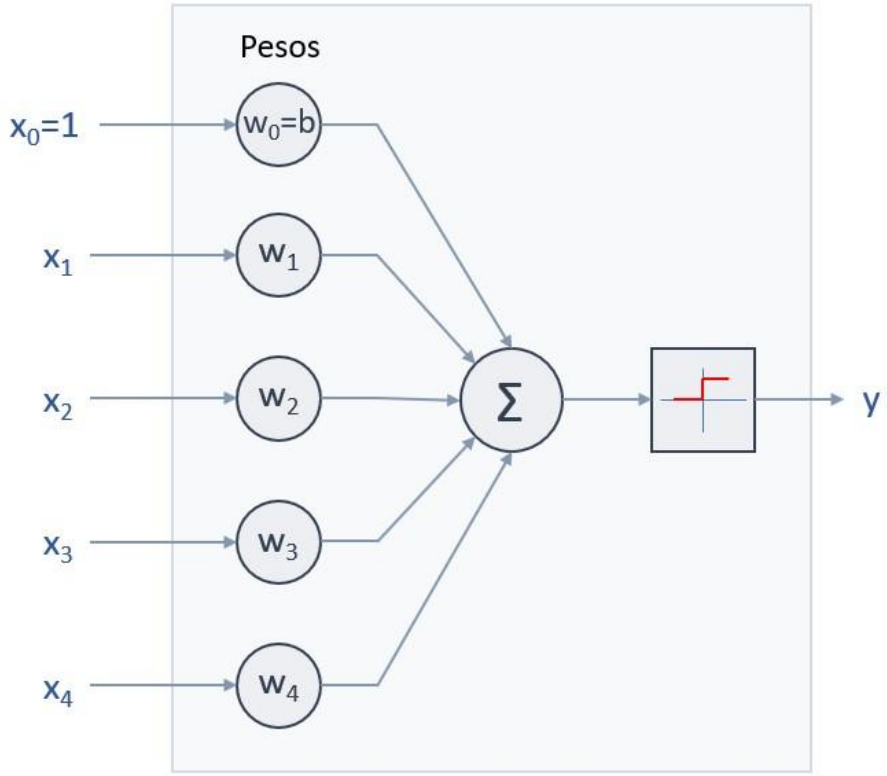
❑ DEEP LEARNING. Perceptrón.

➢ **Por simplificar, se pasa el umbral a la izquierda, y se nombra '-umbral' como 'b', frecuentemente denominado *bias*.**

$$Salida = \begin{cases} 1 \; si \; \sum_{i=1}^{n} w_i \cdot x_i + b \geq 0 \\ 0 \; si \; \sum_{i=1}^{n} w_i \cdot x_i + b < 0 \end{cases}$$

❑ DEEP LEARNING. Perceptrón.

➢ **Si definimos el bias como X_0*W_0, siendo X_0=1 y W_0=b, tenemos:**



Fuente: https://interactivechaos.com

Obsérvese que la función de activación (la función escalón) está centrada en el valor 0.

❑ DEEP LEARNING. Perceptrón.

➢ El Perceptrón como clasificador:

Se devolverá una clase (0 o 1) en función de la entrada x.

Ejercicio: si tenemos dos señales de entrada, con pesos asociados de -0.8 y 1.1, con un bias de 2, tendremos:

$$z = W_0 * X_0 + W_1 * X_1 + W_2 * X_2 = 2 - 0.8 * X_1 + 1.1 * X_2$$

Dibujar con 'contourf' las clases que asignaría Perceptrón a cada pareja de valores (x1, x2). Usar código propio.

Utilizar 'linspace(-3, 3, 1000)' para ambos ejes del 'meshgrid'.

❑ DEEP LEARNING. Perceptrón.

➢ Ejercicio. Solución.

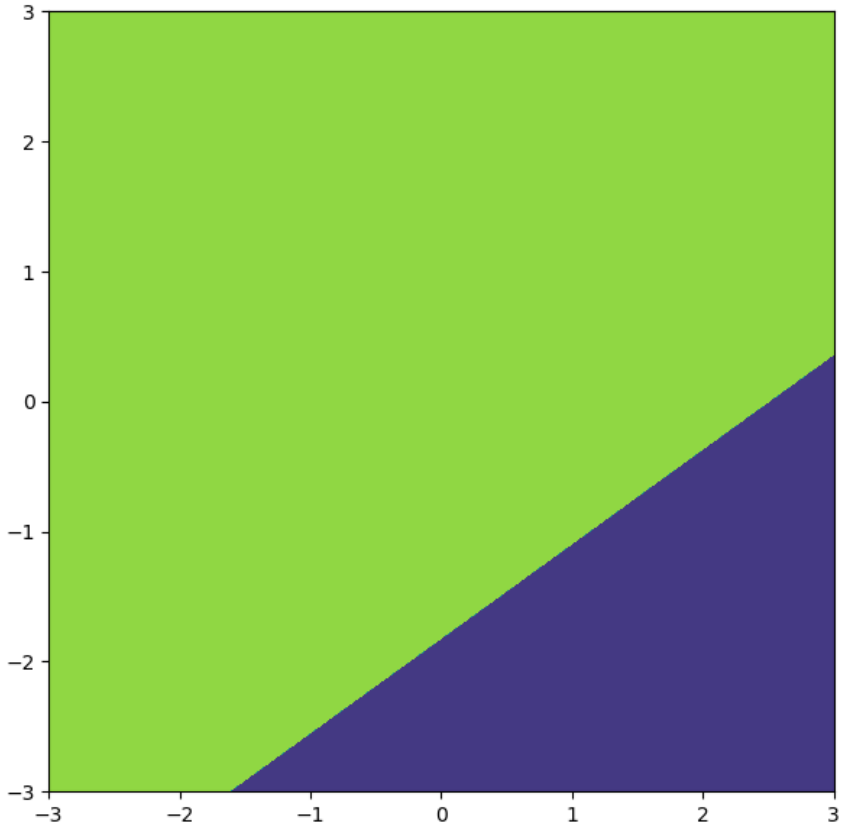
```
import numpy as np
import matplotlib.pyplot as plt

coord = np.linspace(-3, 3, 1000)

x1, x2 = np.meshgrid(coord, coord)
#print(x1)
#print(x2)
z =  2 - 0.8*x1 + 1.1*x2
#print(z)
i=0
for zi in z:
    j=0
    for zj in zi:
        if zj >=0:
            z[i][j]=1
        else:
            z[i][j]=0
        j=j+1
    i=i+1
#print(z)
fig, ax = plt.subplots(figsize = (7, 7))
ax.set_aspect("auto")
ax.contourf(x1, x2, z, levels = 2,
        zorder = 0
    );
```

❑ DEEP LEARNING. Perceptrón.

➢ Solución.



Todos los puntos del plano que provocarían la activación de la neurona son los mostrados en el área con color verde.
Los puntos que no activarían la neurona se muestran en el área con color morado.

❑ DEEP LEARNING. Configuración del Perceptrón.

Los pesos y el bias son valores constantes que se definen durante la configuración de la neurona, ¿cómo?

Entrenamiento, regla de aprendizaje:

✓ Se inicializan los pesos con ceros o con valores aleatorios pequeños.

✓ A continuación, se van pasando las muestras una por una por la neurona. Para cada muestra:

- **Se obtiene la predicción correspondiente ŷ.**
- **Se obtiene la diferencia entre la clase real (y) y la predicción (ŷ), y se modifican los pesos wi según un múltiplo de esta diferencia y del valor xi.**

❑ DEEP LEARNING. Configuración del Perceptrón.

Por tanto, si llamamos Δwi al incremento a aplicar a cada peso, tendremos:

$$\Delta wi = \eta * (y - \hat{y}) * xi$$

$$wi = wi + \Delta wi$$

Donde el parámetro η es la tasa de aprendizaje o Learning rate. Su objetivo es evitar que las variaciones de los pesos sean demasiado grandes.

Durante el entrenamiento del Perceptrón se van a ajustar sus parámetros internos (pesos y bias).

hay otros valores que también van a influir en el comportamiento del modelo resultante. Por ejemplo, la tasa de aprendizaje.

❑ DEEP LEARNING. TASA DE APRENDIZAJE.

- Tasa de aprendizaje, Su valor dependerá de la elección que hagamos nosotros.

- Podemos escoger una tase de aprendizaje de 0.01, o de 0.001, o una tasa que varíe a lo largo del entrenamiento. Este tipo de "parámetros" del algoritmo que dependen del analista reciben el nombre de hiperparámetros del algoritmo.

- Los hiperparámetros deben ser validados, comparando el comportamiento del modelo obtenido con el de otros modelos configurados con otros hiperparámetros.

❑ DEEP LEARNING.

➢ **Ejemplo de entrenamiento:**

X1	X2	y
2	1	1
3	1	0
1	1	1

Sumatorio $z = W_0 * X_0 + W_1 * X_1 + W_2 * X_2$

Si inicializamos por ejemplo los valores de W_0=0, W_1=-1, W_2=1 y una tasa de aprendizaje de 0.1, y pasamos la primera muestra X=(2,1):

$z = 0*1 + (-1)*2 + 1*1 = 0 - 2 + 1 = -1$ (<0, la neurona no se activa)

La neurona devuelve 0 cuando el valor real es 1.

❑ DEEP LEARNING.

➢ **Ejemplo de entrenamiento. Los incrementos en cada caso serían:**

$\Delta w0 = \eta*(y - \hat{y}) * x0 = 0.1*(1-0)*1=0.1$

$\Delta w1 = \eta*(y - \hat{y}) * x1 = 0.1*(1-0)*2=0.2$

$\Delta w2 = \eta*(y - \hat{y}) * x2 = 0.1*(1-0)*1=0.1$

X1	X2	y
2	1	1
3	1	0
1	1	1

Los pesos quedarían:

$w0 = w0 + \Delta w0 = 0 + 0.1 = 0.1$

$w1 = w1 + \Delta w1 = -1 + 0.2 = 0.8$

$w2 = w2 + \Delta w2 = 1 + 0.1 = 1.1$

❑ DEEP LEARNING.

➢ Epochs

Este proceso continúa con las demás muestras, pasando una por una y modificándose los pesos en cada caso si las predicciones realizadas por la neurona no coinciden con la etiqueta correspondiente a la muestra considerada.

El paso de todas las muestras por la neurona es lo que se conoce como un epoch.

Es posible que, tras un epoch, la neurona todavía no haya convergido (que los pesos no hayan tomado todavía sus valores óptimos).

La convergencia solo está asegurada si las muestras del conjunto de entrenamiento son linealmente separables.

❑ DEEP LEARNING. Perceptrón.

➢ Abrir un nuevo Notebook llamado Perceptron.

```
import seaborn as sb
iris = sb.load_dataset("iris")
iris.sample(5, random_state = 0)
```

	sepal_length	sepal_width	petal_length	petal_width	species
114	5.8	2.8	5.1	2.4	virginica
62	6.0	2.2	4.0	1.0	versicolor
33	5.5	4.2	1.4	0.2	setosa

❑ DEEP LEARNING. Perceptrón.

```
iris.shape
#vemos que el dataset tiene 150 registros
```

```
(150, 5)
```

```
#codificamos la columna 'species'
iris["label"] = iris.species.astype("category").cat.codes
iris.sample(5, random_state = 0)
```

	sepal_length	sepal_width	petal_length	petal_width	species	label
114	5.8	2.8	5.1	2.4	virginica	2
62	6.0	2.2	4.0	1.0	versicolor	1

❑ DEEP LEARNING. Perceptrón.

El Perceptrón de Scikit-learn

```
from sklearn.linear_model import Perceptron
```

Hemos visto que el Perceptrón es un clasificador binario (puede trabajar con dos clases/salidas), así que vamos a filtrar el dataset para quedarnos con las muestras de las especies setosa y versicolor (habíamos visto en clases anteriores que son linealmente separables).

❑ DEEP LEARNING. Perceptrón.

```
iris = iris[iris.species.isin(["setosa", "versicolor"])]
iris.shape # el dataset se ha reducido a 100 muestras
```

```
(100, 6)
```

Para poder visualizar los resultados en el plano, vamos a seleccionar dos características predictivas. La variable objetivo será 'label‘.

```
X = iris[["sepal_length", "sepal_width"]]
y = iris["label"]
```

❑ DEEP LEARNING. Perceptrón.

Ahora instanciamos la clase Perceptron con sus parámetros por defecto, y entrenamos el modelo.

```
model = Perceptron(random_state = 0)
model.fit(X.values, y.values)
```

Si visualizáramos los datos con los que estamos trabajando, veríamos que para una muestra con una longitud de sépalo de 5.5 cm y un ancho de sépalo de 4.0 cm la case a asignar debería ser setosa (la clase 0). Veamos qué predice nuestro modelo.

❑ DEEP LEARNING. Perceptrón.

```
model.predict(np.array([5.4, 4]).reshape(1, -1))
#vemos que la predicción es correcta.
```

```
array([0], dtype=int8)
```

Ejercicio: dibujar las fronteras de decisión de nuestro clasificador "model" entrenado sobre el dataset Iris, utilizando la librería que creamos en clase para ello.

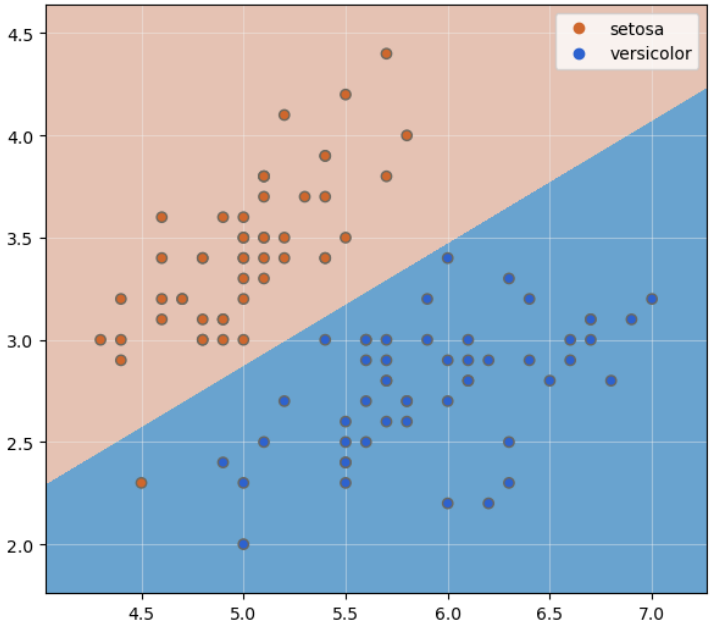
❑ DEEP LEARNING. Perceptrón.

Ejercicio:

```
import fronteras as fr
```

solución.

```
fr.mostrar_fronteras(model,X.values,None,y.values,None,
                     iris.species.unique())
```



Comprobamos que el entrenamiento ha terminado antes de que el algoritmo haya convergido en una recta que divida de forma perfecta los datos.

❑ DEEP LEARNING. Perceptrón.

```
model.score(X.values, y.values)
```

```
0.99
```

El score es del 99%. Eran 100 muestras, por lo que ha quedado una muestra mal clasificada, tal y como se veía en la imagen previa.

Vamos a ver el nº de veces que los datos han pasado por la neurona antes de que el algoritmo haya parado el entrenamiento (el número de epochs). Para ello usamos el atributo '.n_iter_'

❑ DEEP LEARNING. Perceptrón.

```
model.n_iter_
```

```
8
```

Vamos a ver la configuración del algoritmo con el método '.get_params()' del modelo.

```
model.get_params()
```

```
{'alpha': 0.0001,
 'class_weight': None,
 'early_stopping': False,
 'eta0': 1.0,
 'fit_intercept': True,
 'l1_ratio': 0.15,
 'max_iter': 1000,
 'n_iter_no_change': 5,
 'n_jobs': None,
 'penalty': None,
 'random_state': 0,
 'shuffle': True,
 'tol': 0.001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

❑ DEEP LEARNING. Perceptrón.

Vamos a ajustar el modelo. Por ejemplo, vamos a modificar el número de epochs sin mejoras antes de terminar el entrenamiento 'n_iter_no_change‘.

```
model = Perceptron(n_iter_no_change = 300, random_state = 0)
model.fit(X.values, y.values)
score = model.score(X.values, y.values)
print(model.n_iter_)
print(score)
```

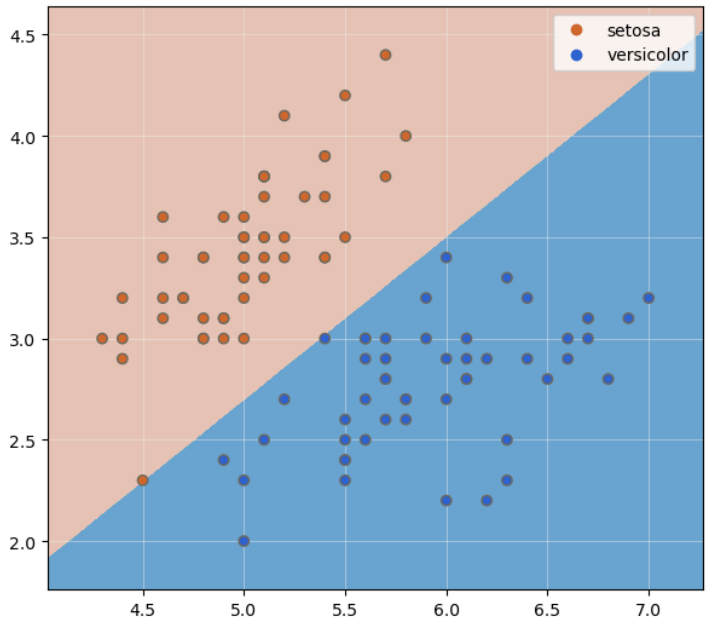
```
837
1.0
```

Vemos que ahora, tras 837 iteraciones, el modelo clasifica los datos perfectamente.

❑ DEEP LEARNING. Perceptrón.

Veamos ahora las fronteras de decisión.

```
fr.mostrar_fronteras(model,X.values,None,y.values,None,
                     iris.species.unique())
```



❑ DEEP LEARNING. Perceptrón.

Aunque el Perceptrón originariamente era un clasificador binario, la implementación de Scikit-Learn soporta la clasificación multiclase aplicando una estrategia denominada one vs. rest (OvR), también conocida como one vs. all.

En la estrategia OvR se entrena un clasificador distinto para cada clase y, en cada clasificador se intenta separar dicha clase de todas las demás. Por último, se combinan los resultados obtenidos de todos los clasificadores.

❑ DEEP LEARNING. Perceptrón.

Ejercicio: volver a cargar el dataset para contemplar las tres especies, dividir la muestra en datos de entrenamiento-validación y hacer el grafo de las fronteras de decisión con las tres especies.

❑ DEEP LEARNING. Perceptrón.

Ejercicio: solución.

```
iris = sb.load_dataset("Iris")
iris["label"] = iris.species.astype("category").cat.codes
```

```
X = iris[["sepal_length", "sepal_width"]]
y = iris["label"]
```

```
from sklearn.model_selection import train_test_split
```

❑ DEEP LEARNING. Perceptrón.

Ejercicio: solución.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y,
                                                    test_size=0.2, random_state = 0)
```

Con el parámetro stratify estamos pidiendo a la función que asegure que la distribución de los valores de la etiqueta 'y' sea equivalente en los bloques generados (es decir, que el porcentaje de flores setosa, versicolor y virginica sea equivalente en los bloques de entrenamiento y de validación).

❑ DEEP LEARNING. Perceptrón.

Ejercicio: solución.

Podemos modificar la escala de los datos para evitar que el Perceptrón requiera aplicar incrementos demasiado grandes a los pesos, mejorando el rendimiento del entrenamiento.

Por ejemplo, la clase 'StandardScaler' elimina el valor medio de los datos y los escala de forma que su desviación estándar sea 1 (medida de la dispersión del conjunto de datos, que vimos en clase).

❑ DEEP LEARNING. Perceptrón.

Ejercicio: solución.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
```

Importante: el método '.fit_transform()' entrena el escalador a partir de los datos que se le pasan como argumento, y transforma éstos devolviendo un array NumPy con los datos transformados. El método '.transform()' simplemente aplica el escalador (ya entrenado con otros datos), devolviendo los nuevos datos escalados en un array NumPy.

❑ DEEP LEARNING. Perceptrón.

Ejercicio: solución.

Entrenamiento y validación del modelo: la tasa de aprendizaje por defecto en el Perceptrón de Scikit-Learn es de 1, valor demasiado grande si tenemos en cuenta que hemos escalado los datos (con desviación estándar = 1).

Vamos a poner una tasa de aprendizaje de 0.1.

```
model = Perceptron(eta0 = 0.1, random_state = 10)
model.fit(X_train_std, y_train)
```

❑ DEEP LEARNING. Perceptrón.

Ejercicio: solución.

```
scor = model.score(X_test_std, y_test)
print(scor)
print(model.n_iter_)
```

```
0.7333333333333333
12
```

❑ DEEP LEARNING. Perceptrón.

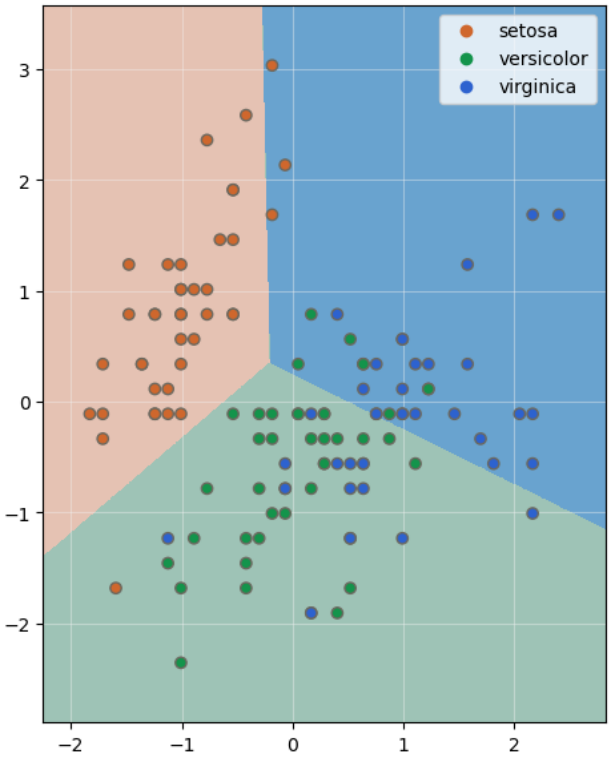
Ejercicio. Teniendo en cuenta la nueva escala de datos:

- **Hacer un grafo de las fronteras de decisión situando los datos de entrenamiento.**
- **Hacer un grafo de las fronteras de decisión situando los datos de validación.**
- **Hacer un grafo de las fronteras de decisión situando los datos de entrenamiento y validación.**

❑ DEEP LEARNING. Perceptrón.

Ejercicio. Solución.

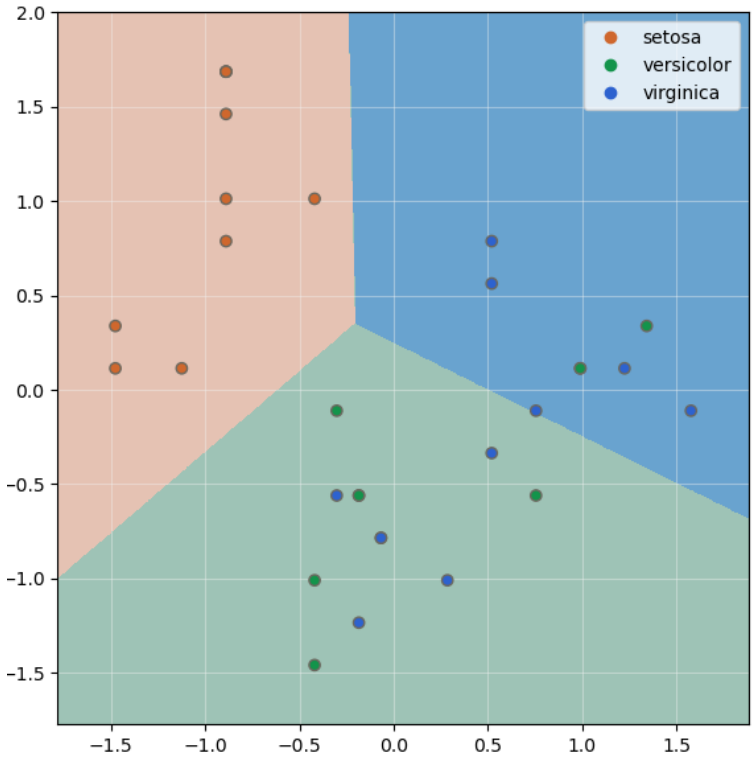
```
fr.mostrar_fronteras(model,X_train_std,None,y_train,None,iris.species.unique())
```



❑ DEEP LEARNING. Perceptrón.

Ejercicio. Solución.

```
fr.mostrar_fronteras(model,X_test_std,None,y_test,None,iris.species.unique())
```



❑ DEEP LEARNING. Perceptrón.

Ejercicio. Solución.

```
fr.mostrar_fronteras(model, X_train_std, X_test_std, y_train, y_test, iris.species.unique())
```