Facultad de Ciencias, UNAM
Redes Neuronales
Tarea 2

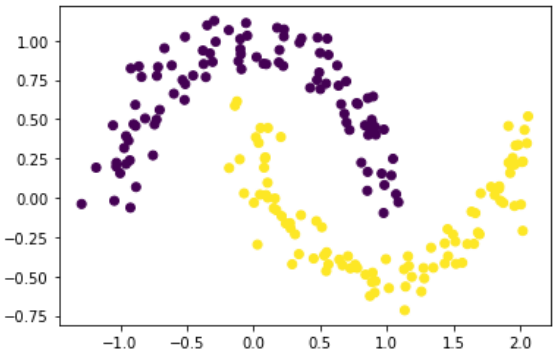
Rubí Rojas Tania Michelle

13 de abril de 202

1. Usando *sklearn.datasets.make_moons* genera un conjunto de datos de la siguiente forma:

```
In [1]: C1, C2 = moons(random state=123, n samples=200, noise=0.1)
```

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

C1, C2 = make_moons(random_state = 123, n_samples=200, noise=0.1)
plt.scatter(C1[:,0], C1[:,1], c = C2)
plt.show()

```



a) Implementa la regresión logística usando el descenso gradiente para clasificar C_1 y C_2.
Solución:

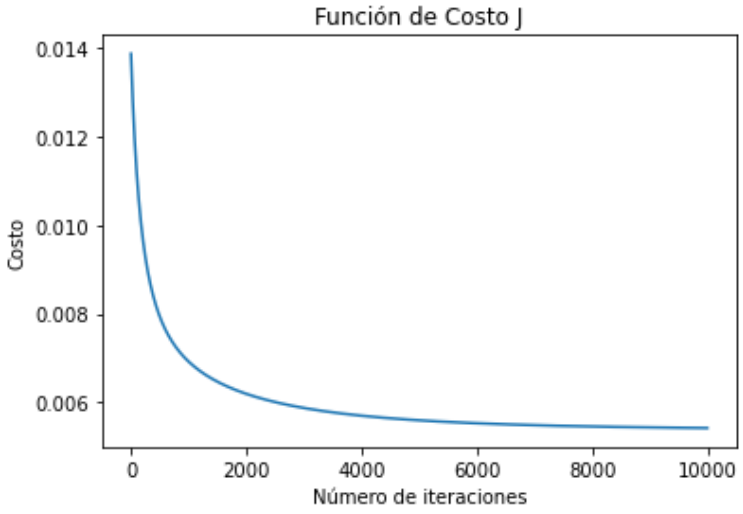
```
        import numpy as np

        '''
        La funcion sigmoide es de la forma S(z) = 1/(1 + e^{-z}), donde
        z = x * w^{T} + b.
        '''
        def sigmoide(z):
            return 1 / (1 + np.exp(-z))

        def de_coste(y, y_hat):
            j = - (y.dot(np.log(y_hat)) + (1 - y).dot((np.log(1 - y_hat))))
            return j

```

```
        def descenso_gradiente(x, y, alpha, theta, num_iteraciones):
            costos = []
            for each_iter in range (num_iteraciones):
                z = np.dot(x, theta[1:]) + theta[0]
                y_hat = sigmoide(z)
                error = y_hat - y
                gradiente = x.T.dot(error)
                theta[0] -= alpha * error.sum()
                theta[1:] -= alpha * gradiente
                costos.append((1 / num_iteraciones) * (de_coste(y, y_hat)))
            return costos

        def prediccion(data):
            z = np.dot(data, theta[1:]) + theta[0]
            return np.where(sigmoide(z)>= 0.5, 0, 1)

```

Asignando valores a θ, α y al número de iteraciones, obtenemos

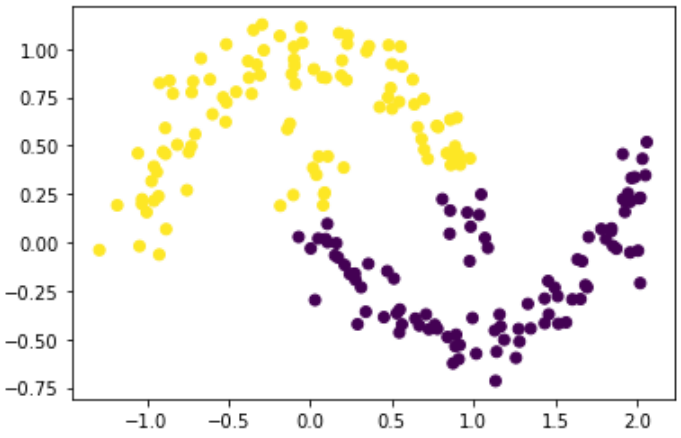
```
        k, l = C1.shape
        theta = np.zeros(l+1)
        costo = descenso_gradiente(C1, C2, 0.0001, theta, 10000)

        print ('Interseccion:', theta[0])
        print ('Coeficientes estimados:', theta[1:])

        # Visualizamos la funcion de costo.
        plt.title('Funcion de Costo J')
        plt.xlabel('Numero de iteraciones')
        plt.ylabel('Costo')
        plt.plot(costo)
        plt.show()

```

```
Intersección: 0.5894707477910425
Coeficientes estimados: [ 1.15656526 -4.85276922]
```



Finalmente, la clasificación quedaría de la siguiente forma:

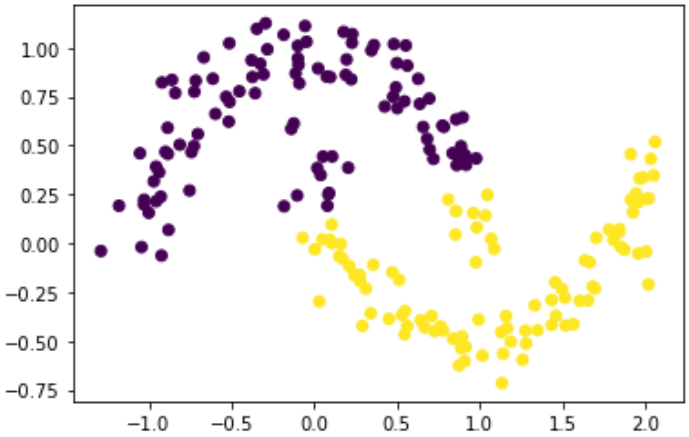
```
        plt.scatter(C1[:,0], C1[:,1], c = prediccion(C1))
        plt.show()

```



Utilizando **sklearn** para comprar los resultados obtenemos

```
        from sklearn.linear_model import LogisticRegression

        regresion_logistica = LogisticRegression()
        modelo = regresion_logistica.fit(C1, C2)

        print ("Interseccion: ", modelo.intercept_)
        print ("Coeficientes estimados: ", modelo.coef_)

        # Visualizamos la clasificacion.
        plt.scatter(C1[:,0], C1[:,1], c = regresion_logistica.predict(C1))
        plt.show()

```

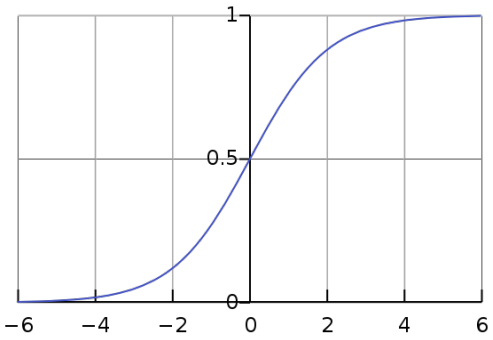
```
Intersección:  [0.36023346]
Coeficientes estimados:  [[ 1.09045248 -3.69780292]]
```



b) ¿Qué transformación de los datos ocupaste para poder hacer la correcta clasificación?
SOLUCIÓN: Sabemos que la función *sigmoide*

$$S(z) = \frac{1}{1+e^{-z}} \qquad z = w^T x + b$$

nunca llega a tocar el eje de las y a 0 o 1, y se mueve entre estos dos valores.



Lo cual la hace perfecta para expresar una propabilidad para nuestra decisión binaria (clasificar).
Ahora bien, debemos encontrar los valores correctos de w y b que nos permitan encontrar el valor correcto de la función S. El vector de pesos w se inicializa de manera arbitraria (aunque en nuestra implementación lo inicializamos como el vector cero).
Para saber qué tan bien o qué tan mal logra clasificar nuestra función, requerimos una función de error o de coste. Y teniéndo en cuenta que nuestro problema no es convexo, entonces sería de la forma

$$L(y', y) = -(y \log y' + (1-y)\log(1-y'))$$

Esta función únicamente se usará sobre una entrada del conjunto de datos. Ahora, para obtener una imagen general sobre el error de los parámetros con respecto al total de los datos, usaremos la función de costo, que es definida como:

$$\begin{aligned} J(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L(y'^i, y^i) \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^i \log y'^i + (1-y^i)\log(1-y'^i)] \end{aligned}$$

El objetivo es minimizar el costo total del modelo, para así acercarnos lo más posible a nuestras clases dadas y. Esta minimización la hacemos utilizando el *descenso gradiente*.
Por lo tanto, la función *sigmoide* usa los datos de entrada y los valores del modelo θ para minimizar la función de coste con ayuda del algoritmo del desceso. Ésta es la transformación de los datos que se va realizando, y se puede ver en la gráfica de función de coste $1a$.

2. Calcula la derivada de la tangente hiperbólica tanh.

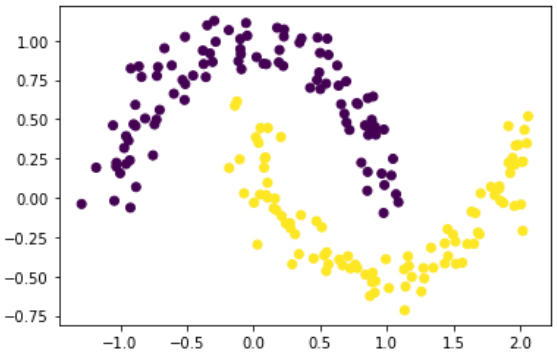
   Solución:

$$\begin{aligned}
\frac{d}{dx}\tanh(x) &= \frac{d}{dx}\left(\frac{\sinh(x)}{\cosh(x)}\right) && \text{definición de } \tanh(x)\\
&= \frac{(\sinh'(x)\cdot\cosh(x)) - (\cosh'(x)\cdot\sinh(x))}{\cosh^2(x)} && \text{derivative quotient rule}\\
&= \frac{(\cosh(x)\cdot\cosh(x)) - (\sinh(x)\cdot\sinh(x))}{\cosh^2(x)} && \sinh'(x)=\cosh(x) \text{ y } \cosh'(x)=\sinh(x)\\
&= \frac{\cosh^2(x) - \sinh^2(x)}{\cosh^2(x)} && \text{aritmética}\\
&= \frac{1}{\cosh^2(x)} && \cosh^2(x)-\sinh^2(x)=1\\
&= sech^2(x) && \frac{1}{\cosh^2} = sech^2(x)
\end{aligned}$$

3. Usando el perceptrón multicapa visto en clase, clasifica a C_1 y C_2. ¿Qué parámetros ocupaste?

   Solución: Primero, generamos un conjunto de datos.

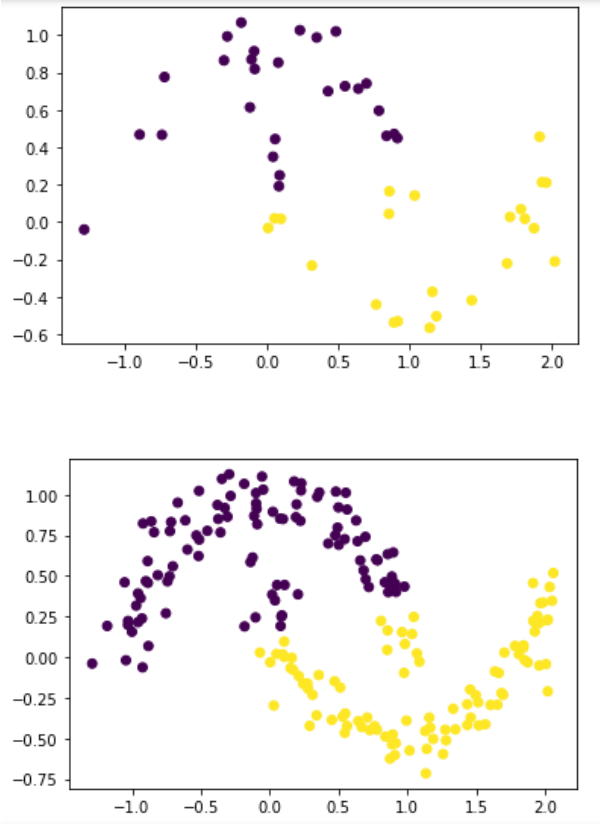
```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

C1, C2 = make_moons(random_state = 123, n_samples=200, noise=0.1)
plt.scatter(C1[:,0], C1[:,1], c = C2)
plt.show()

```



   Apoyándonos del **MLPClassifier** para clasificar el conjunto de datos, obtenemos

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection  import train_test_split
import matplotlib.pyplot as plt

# Dividimos al conjunto de datos para poder manejarlo.
X_train, X_test, y_train, y_test = train_test_split(C1, C2)

# Aqui podemos observar los parametros que utilizamos.
modelo = MLPClassifier(activation = 'tanh', max_iter = 1000,
                       learning_rate_init = 0.001, hidden_layer_sizes = (10,4))

modelo.fit(X_train, y_train)


```

```
    # Visualizamos la clasificacion con los datos de prueba.
    plt.scatter(X_test[:,0], X_test[:,1], c = modelo.predict(X_test))
    plt.show()

    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    # Visualizamos la clasificacion con los datos de prueba y entrenamiento.
    ax1.scatter(X_test[:,0], X_test[:,1], c = modelo.predict(X_test))
    ax1.scatter(X_train[:,0], X_train[:,1], c = modelo.predict(X_train))
    plt.show()

```



4. Con la red neuronal, vista en clase, que hace la clasificación multiclase usando la función *softmax*, realiza los siguiente ejercicios:

   a) Encuentra la mejor arquitectura para el conjunto de Iris. Justifica tu respuesta de por qué es la mejor.

   b) Usa las funciones $\tan h$ y γ en la capa intermedia. ¿Cuál funciona mejor?

   c) Clasifica los siguientes estímulos y reporta a qué clase pertenece cada uno:

   - 5.97 4.20 1.23 0.25
   - 6.80 5.00 1.25 1.20
   - 12.50 9.20 40.32 21.55

   d) ¿Te parecen correctas todas las clasificaciones? En caso de que alguna no, ¿por qué? ¿cómo corregirías este error?

5. ¿Qué es y cómo funciona la función de activación *Radial Basis Function (RBF)*?

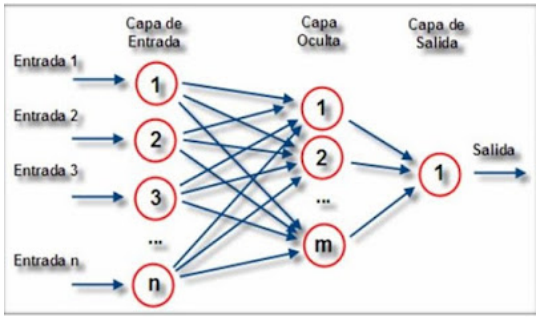
SOLUCIÓN: Una *función de base radial* $\phi(x)$ es una función que calcula la distancia euclideana de un vector de entrada x respecto de un centro c, de tal manera que resulta la siguiente función:

$$f(x) = (\|x - c_i\|)$$

A cada neurona de la capa de entrada le corresponde una función de base radial $\Phi(x)$ y un peso de salida w_i. El patrón de salida ingresa a una neurona de salida que suma las entradas y da como resultado una salida. La función de una red RBF final resulta:

$$F(x) = \sum_{i=1}^{N} w_i \Phi(\|x - c_i\|)$$

Las redes RBF tienen una construcción rígida de tres capas:



- **Capa de Entrada**. Transmiten las señales de entrada a las neuronas ocultas sin realizar procesamiento, es decir, las conexiones de la capa de entrada a la capa oculta no llevan pesos asociados.
- **Capa Oculta**. Realizan una transformación local y no lineal de dichas señales.

  Cada elemento de procesado i de la capa oculta tiene asociada una función de base radial de tal manera que representa una clase o categoría, donde dicha clase viene dada por (C_i, d_i). C_i representa un centro de cluster (pesos asociados a cada neurona i) y d_i representa la desviación, anchura o dilatación de la función de base radial asociada a dicho elemento.

  La salida de cada elemento de la capa oculta $z_i(n)$ se calcula como ls distancia que existe entre el patrón de entrada $X(n)$ al centro del clouster C_i ponderada inversamente por d_i, y aplicando después a ese valor una función de base radial.

$$z_i(n) = \Phi\left(\frac{\left(\sum_{j=1}^{p}(x_j(n) - c_{ji})^2\right)^{\frac{1}{2}}}{d_i}\right) \qquad i \in \{1, 2, ..., m\}$$

  donde Φ es una función de base radial, dentro de éstas la más utilizada es la función Gaussiana

$$\Phi(r) = e^{\frac{-r^2}{2}}$$

- **Capa de Salida**. Realiza una combinación de las actividades de las neuronas ocultas. Tiene la responsabilidad en la red de activación de patrones aplicados en la capa de entrada.

  Cada elemento de procesado calcula su valor neto como una combinación lineal de las salidas de los elementos de procesado de la capa oculta. La función de activación y transferencia es lineal,

por lo que para un patrón n, $X(n) = (x_1(n), x_2(n), ..., x_p(n))$, la salida de la red asociada a cada elemento k de la capa de salida se obtiene de la forma

$$y_k(n) = \sum_{i=1}^{m} w_{ik} z_i(n) + \mu_k \qquad k \in \{1, 2, ..., r\}$$

donde los w_{ik} son los pesos asociados al elemento k de la capa oculta y el elemento i de la capa oculta, que ponderan cada uno las salidas $z_i(n)$ del elemento de procesado de la capa oculta correspondiente.

El término μ_k es un término denominado umbral y está asociado a cada uno de los elementos de procesado de la capa de salida.

El aprendizaje consiste en la determinación de centros, desviaciones y pesos de la capa oculta a la capa de salida. Como las capas de la red realizan diferentes tareas, se separaán los parámetros de la capa oculta de la capa de salida para optimizar el proceso. De esta forma, los centros y las desviaciones siguen un proceso guiado por una optimización en el espacio de entrada, mientras que los pesos siguen una optimización sobre la base de las salidas que se desean obtener.

Los métodos de aprendizaje más utilizados son el *método híbrido* y el *método totalmente supervisado*.

Bibliografía

- https://medium.com/@dtellogaete/regresi%C3%B3n-log%C3%ADstica-en-python-y-r-machine-learning-
- http://www.eenube.com/index.php/ldp/machine-learning/121-programar-una-red-neuronal-para-clas
- https://www.interactivechaos.com/manual/tutorial-de-machine-learning/perceptron-multicapa
- https://analisisydecision.es/machine-learnig-analisis-grafico-del-funcionamiento-de-algunos-a
- https://www.analyticslane.com/2020/04/20/entrenamiento-validacion-y-test-con-scikit-learn/
- https://blog.crespo.org.ve/2018/02/
- https://www.wikiwand.com/es/RNA_de_base_radial#/Referencias
- http://ele.aut.ac.ir/~abdollahi/Lec_3_NN11.pdf
- http://dianainteligenciaartificial.blogspot.com/2015/07/redes-de-neuronas-de-base-radial.html
- http://www.varpa.org/~mgpenedo/cursos/scx/archivospdf/Tema5-6.pdf