Práctica 4 - Aprendizaje supervisado en redes multicapa

Zablotsky, Amir Nicolás
Redes Neuronales 2022
Instituto Balseiro, UNCuyo

EJERCICIO 1

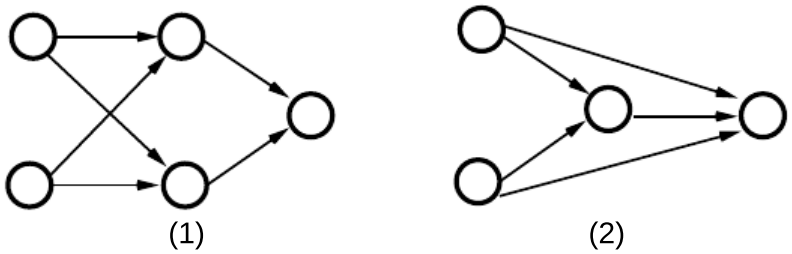

Figura 1: (Izquierda) Arquitectura 1. (Derecha) Arquitectura 2.

Se estudiaron dos redes con arquitectiuras distintas, que se observan en la Fig. 1, para aprender la regla lógica XOR. Esta regla tiene dos entradas (-1 y 1), y una salida que vale -1 si ambas entradas son distintas o 1 si son iguales.

Se implementaron ambas redes en python, y se implementó el algoritmo de retropropagación para aprender la regla XOR en ambas arquitecturas. Para llevar a cabo el aprendizaje se utilizaron los cuatro pares posibles de entradas y salidas para entrenar las redes. Se utilizó un learning rate de 0,005 y se entrenó ambas redes durante 5000 épocas. Como función de costo se utilizó el error cuadrático medio (MSE), y como función de activación se utilizó $\tanh(x)$. Para calcular la precisión, se consideró una predicción acertada si el signo de la salida corresponde con el de la salida verdadera, y errónea en caso contrario.

En la Fig. 2 se observa la evolución de la precisión y la función de costo en función de las épocas, para la primer arquitectura, y en la Fig. 3 se observa la evolución de la precisión y la función de costo para la segunda arquitectura. En ambos casos se repitió el proceso de entrenamiento 10 veces, con condiciones iniciales de los pesos y los bias al azar.

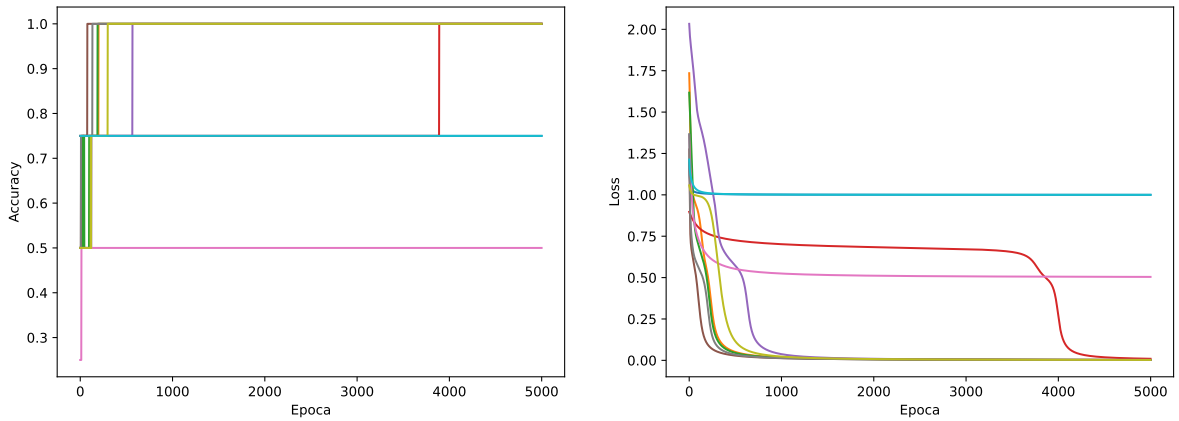

Figura 2: Evolución de la precisión y la función de costo al avanzar las épocas, en la primera arquitectura. Podemos ver como en varios de los entrenamientos, la precisión converge a 1.

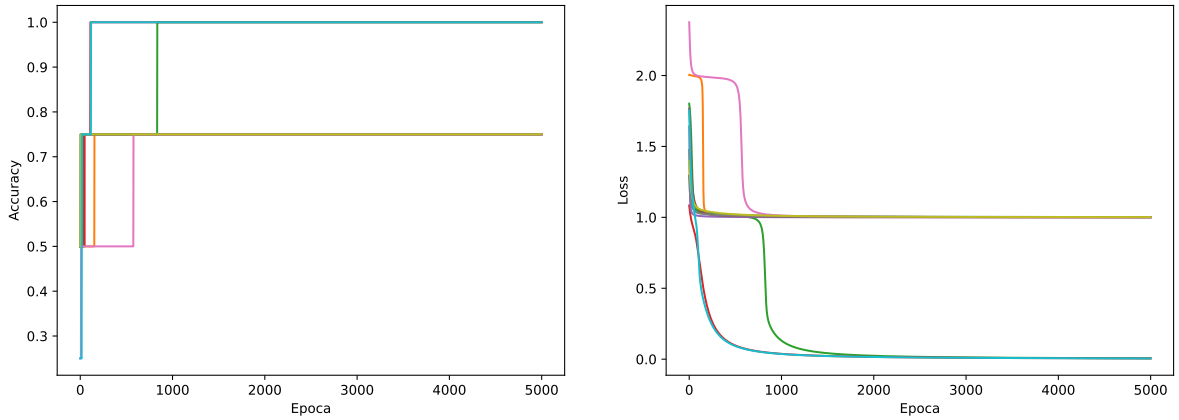

Figura 3: Evolución de la precisión y la función de costo al avanzar las épocas, en la segunda arquitectura. Podemos ver como en muchos de los entrenamientos, se alcanza un mínimo local, y no alcanzan la accuracy óptima igual a 1.

Luego estudiamos el tiempo de convergencia en ambas arquitecturas, definido como la cantidad de épocas que tarda la red en alcanzar una precisión $> 0{,}9$. En caso de que no se alcance esta precisión en las 5000 épocas de entrenamiento, decimos que no converge. Para obtener los tiempos medios de convergencia, se entrenó cada red la cantidad de veces necesaria hasta que converja 100 veces.

Si bien la primera arquitectura tiene un tiempo medio de convergencia de ~ 946 épocas, y la segunda arquitectura de ~ 819 épocas, vemos que la primer arquitectura converge el 60 % de las veces, mientras que la segunda lo hace solamente en el 35 % de los casos. Por este motivo, considero que la arquitectura número 1 resulta más apropiada para aprender la regla XOR.

EJERCICIO 2

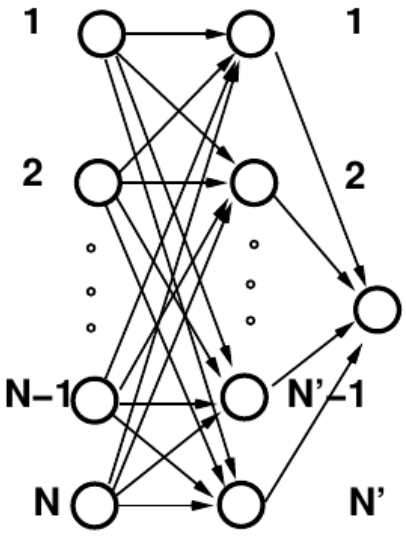

Figura 4: Arquitectura de la red propuesta para aprender el problema de pariedad.

Se utilizó nuevamente el algoritmo de retropropagación para aprender el problema de pariedad, el cual generaliza la regla XOR, en una red con la arquitectura enseñada en la Fig. 4. En este caso la entrada toma $N = 5$ valores (-1 o 1) y la salida toma el valor del producto de las entradas. Para entrenar la red se utilizaron las 32 parejas posibles de entradas y salidas, y se utilizó un learning rate de 0,005, 3000 épocas de entrenamiento, la función de activación $\tan(x)$, función de costo MSE y mismo criterio de predicción que en el Ejercicio 1 (es decir predicción 1 si la salida es positiva, y -1 si es negativa).

Se entrenaron seis redes con distintas cantidades N' de neuronas en la capa oculta $(1, 3, 5, 7, 9, 11)$, y para cada una de ellas se estudió la evolución de la precisión (evaluada sobre los mismos datos de entrenamiento) y la función de costo con el paso de las épocas (Fig. 5).

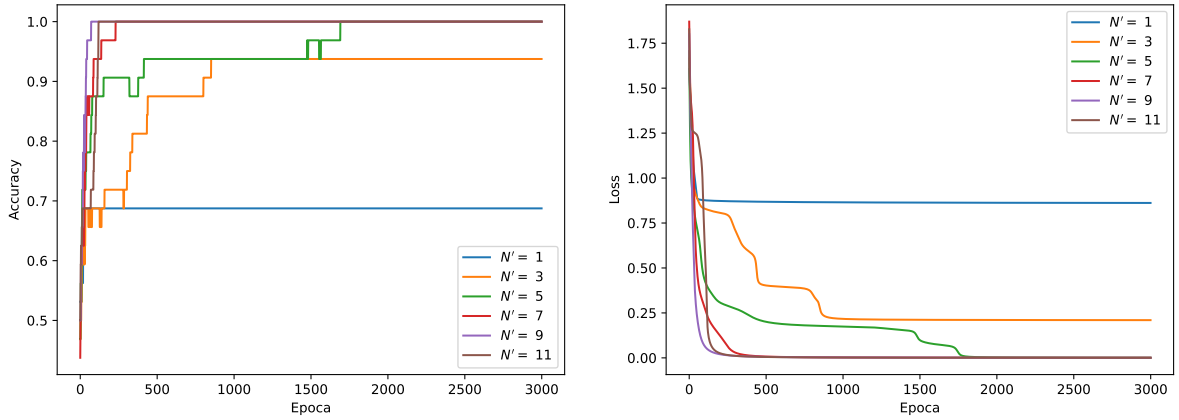

Figura 5: Evolución de la precisión y la función de costo al avanzar las épocas, para redes con distinto N'. Podemos ver como la red converge para $N' \geq 5$, pero no así en caso contrario.

Como podemos ver, para $N' < N$, la red no alcanza una buena precisión (en particular para la red con una sola neurona en la capa oculta). Esto se debe a que, para N' chico, tenemos pocos w_{ij} para ajustar correctamente una cantidad grande de datos de entrada (underfitting). Para $N' \geq 5$ vemos que la red si converge, y alcanza una precisión de 1 cada vez más rápido, siendo análogo al problema del XOR en la arquitectura 1 del Ejercicio 1.

EJERCICIO 3

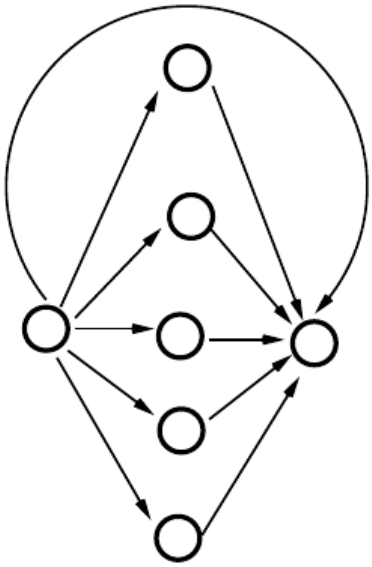

Figura 6: Arquitectura de la red propuesta para aprender el mapeo logístico.

Finalmente se utilizó el algoritmo de retropropagación para aprender el mapeo logístico, dado por la Ec. 1.

$$x(t+1) = 4x(t)\left(1 - x(t)\right) \tag{1}$$

En primer lugar se generaron iterativamente 500 pares de la forma $\big(x(t), x(t+1)\big)$, comenzando por $x(0) = 0{,}1$. Luego se entrenó la red propuesta en la Fig. 6 utilizando n pares $\big(x(t), x(t+1)\big)$ seleccionados al azar, con $n \in \{5, 10, 100\}$, y en cada uno de los casos se seleccionaron al azar otros n pares distintos a los de entrenamiento para usar como validación.

Para entrenar las redes se usó un learning rate de 0,005, 5000 épocas de entrenamiento, función de activación sigmoide(x) para las neuronas de la capa oculta, función de activación lineal para la salida, y función de costo MSE.

En la Fig. 7 se observan la precisión y función de costo vs. épocas, para la red entrenada con 5, 10 y 100 pares de datos. En cada caso, se ilustran las curvas correspondientes a los datos de train y los datos de test. Vemos que, en especial para las redes entrenadas con 5 y 10 pares de datos, la precisión no converge a 1 y la función de costo es mayor sobre los datos de validation que de test. Esta diferencia entre el error de entrenamiento y error de generalización se achica al aumentar la cantidad de datos con la que se entrena la red, siendo la red entrenada con 100 pares de datos la única que converge a precisión 1, y la más adecuada para generalizar el problema y aprender el mapeo logístico.

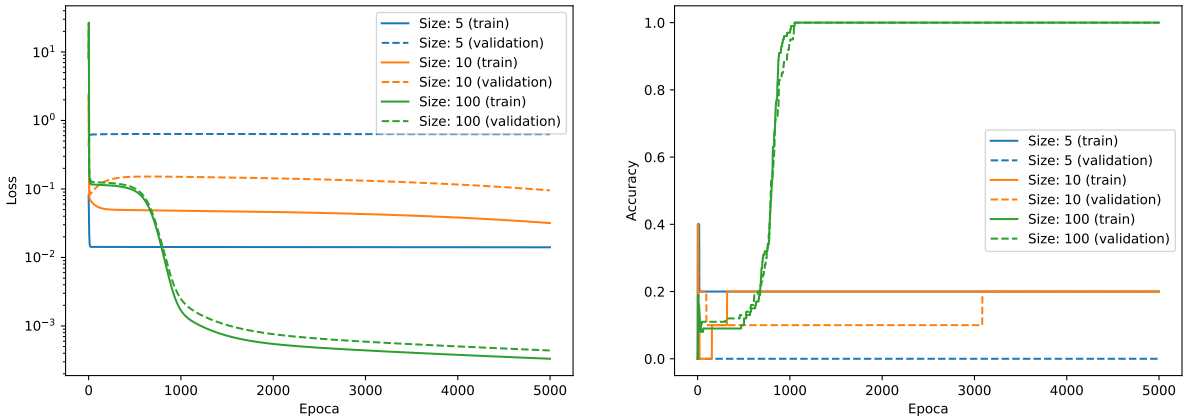

Figura 7: Evolución de la precisión y la función de costo al avanzar las épocas, para los datos de entrenamiento y de validación (para cada uno de los tres casos de entrenamiento). Podemos ver como la red aprende el mapeo logístico de manera correcta al entrenarla con 100 pares $\big(x(t), x(t+1)\big)$.

APÉNDICE

Código Ejercicio 1

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

def fit1(X, Y, n_epochs, lr):
    W1 = np.random.randn(2,2)
    b1 = np.random.randn(2)
    W2 = np.random.randn(2,1)
    b2 = np.random.randn(1)
    loss = np.zeros(n_epochs)
    acc = np.zeros(n_epochs)
    t_conv=-1

    for i in range(n_epochs):
        Z1 = np.dot(X,W1) + b1
        A1 = np.tanh(Z1)
        Z2 = np.dot(A1,W2) + b2
        A2 = np.tanh(Z2)

        loss[i] = np.mean((A2 - Y)**2)
        acc[i] = np.mean(np.sign(A2) == np.sign(Y))

        if loss[i]<0.1 and acc[i]>0.9 and t_conv==-1:
            t_conv=i

        dZ2 = 2*(A2 - Y)*(1 - np.tanh(Z2)**2)
        dW2 = np.dot(A1.T, dZ2)
        db2 = np.sum(dZ2, axis=0)
        dZ1 = np.dot(dZ2, W2.T) * (1 - np.tanh(Z1)**2)
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0)

        W1 -= lr * dW1
        b1 -= lr * db1
        W2 -= lr * dW2
        b2 -= lr * db2
    return loss, acc, t_conv

def fit2(X, Y, n_epochs, lr):
    W1 = np.random.randn(2,1)
    b1 = np.random.randn(1)
    W2 = np.random.randn(3,1)
    b2 = np.random.randn(1)
    t_conv=-1

    loss = np.zeros(n_epochs)
    acc = np.zeros(n_epochs)

    for i in range(n_epochs):
        Z1 = np.dot(X,W1) + b1
        A1 = np.tanh(Z1)
        Z2 = np.dot(np.hstack((X,A1)),W2) + b2
        A2 = np.tanh(Z2)
```

```
        loss[i] = np.mean((A2 - Y)**2)
        acc[i] = np.mean(np.sign(A2) == np.sign(Y))

        if loss[i]<0.1 and acc[i]>0.9 and t_conv==-1:
            t_conv=i

        dZ2 = 2*(A2 - Y)*(1 - np.tanh(Z2)**2)
        dW2 = np.dot(np.hstack((X,A1)).T, dZ2)
        db2 = np.sum(dZ2, axis=0)
        dZ1 = dZ2 * (1 - np.tanh(Z1)**2) * W2[2]
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0)

        W1 -= lr * dW1
        b1 -= lr * db1
        W2 -= lr * dW2
        b2 -= lr * db2
    return loss, acc, t_conv

X_train=np.array([[-1,-1],[-1,1],[1,-1],[1,1]]).astype(float)
y_train=np.array([[1],[-1],[-1],[1]]).astype(float)

fig, ax = plt.subplots(1,2, figsize=(15,5))
for i in range(10):
    loss, accuracy, t_conv = fit1(X_train, y_train, 5000, 0.005)
    ax[1].plot(loss)
    ax[0].plot(accuracy)
ax[1].set_xlabel('Epoca')
ax[1].set_ylabel('Loss')
ax[0].set_xlabel('Epoca')
ax[0].set_ylabel('Accuracy')
plt.show()

iters=0
iters_conv=0
epoch_conv=[]
while len(epoch_conv)<100:
    loss, accuracy, t_conv = fit1(X_train, y_train, 5000, 0.005)
    iters+=1
    if t_conv!=-1:
        epoch_conv.append(t_conv)
        iters_conv+=1
print("La arquitectura 1 converge en promedio en ",np.mean(epoch_conv)," epocas")
print("La arqutectura 1 converge el ",iters_conv/iters*100,"% de las veces")

fig, ax = plt.subplots(1,2, figsize=(15,5))
for i in range(10):
    loss, accuracy, t_conv = fit2(X_train, y_train, 5000, 0.005)
    ax[1].plot(loss)
    ax[0].plot(accuracy)
ax[1].set_xlabel('Epoca')
ax[1].set_ylabel('Loss')
ax[0].set_xlabel('Epoca')
ax[0].set_ylabel('Accuracy')
plt.show()

iters=0
```

```
iters_conv=0
epoch_conv=[]
while len(epoch_conv)<100:
    loss, accuracy, t_conv = fit2(X_train, y_train, 5000, 0.005)
    iters+=1
    if t_conv!=-1:
        epoch_conv.append(t_conv)
        iters_conv+=1
print("La arquitectura 2 converge en promedio en ",np.mean(epoch_conv)," epocas")
print("La arqutectura 2 converge el ",iters_conv/iters*100,"% de las veces")
```

Código Ejercicio 2

```
def fitN(X, Y, n_epochs, lr,N):
    W1 = np.random.randn(5,N)
    b1 = np.random.randn(N)
    W2 = np.random.randn(N,1)
    b2 = np.random.randn(1)
    loss = np.zeros(n_epochs)
    acc = np.zeros(n_epochs)
    t_conv=-1

    for i in range(n_epochs):
        Z1 = np.dot(X,W1) + b1
        A1 = np.tanh(Z1)
        Z2 = np.dot(A1,W2) + b2
        A2 = np.tanh(Z2)

        loss[i] = np.mean((A2 - Y)**2)
        acc[i] = np.mean(np.sign(A2) == np.sign(Y))

        if loss[i]<0.1 and acc[i]>0.9 and t_conv==-1:
            t_conv=i

        dZ2 = 2*(A2 - Y)*(1 - np.tanh(Z2)**2)
        dW2 = np.dot(A1.T, dZ2)
        db2 = np.sum(dZ2, axis=0)
        dZ1 = np.dot(dZ2, W2.T) * (1 - np.tanh(Z1)**2)
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0)

        W1 -= lr * dW1
        b1 -= lr * db1
        W2 -= lr * dW2
        b2 -= lr * db2
    return loss, acc, t_conv

X_train=np.array(list(itertools.product([-1,1], repeat=5))).astype(float)
y_train=np.array([np.prod(x) for x in X_train]).astype(float).reshape(-1,1)

fig, ax = plt.subplots(1,2, figsize=(15,5))
Ns=[1,3,5,7,9,11]
for N in Ns:
    loss, accuracy, t_conv = fitN(X_train, y_train, 3000, 0.005,N)
    ax[1].plot(loss,label="$N'=$ "+str(N))
    ax[0].plot(accuracy,label="$N'=$ "+str(N))
```

```
ax[1].set_xlabel('Epoca')
ax[1].set_ylabel('Loss')
ax[0].set_xlabel('Epoca')
ax[0].set_ylabel('Accuracy')
ax[1].legend()
ax[0].legend()
plt.show()
```

Código Ejercicio 3

```
def g(x):
    return 1/(1+np.exp(-x))
def g_p(x):
    return np.exp(-x)/((1+np.exp(-x))**2)
def lineal(x):
    return x
def lineal_p(x):
    return 1

def fitLogistic(X, Y,X_val,Y_val, n_epochs, lr):
    W1 = np.random.randn(1,5)
    b1 = np.random.randn(5)
    W2 = np.random.randn(6,1)
    b2 = np.random.randn(1)
    t_conv=-1

    loss = np.zeros(n_epochs)
    acc = np.zeros(n_epochs)
    loss_val=np.zeros(n_epochs)
    acc_val=np.zeros(n_epochs)

    for i in range(n_epochs):
        Z1 = np.dot(X,W1) + b1
        A1 = g(Z1)
        Z2 = np.dot(np.hstack((X,A1)),W2) + b2
        A2 = lineal(Z2)

        loss[i] = np.mean((A2 - Y)**2)
        acc[i] = np.mean(np.abs(A2-Y) < 0.05)

        Z1_val = np.dot(X_val,W1) + b1
        A1_val = g(Z1_val)
        Z2_val = np.dot(np.hstack((X_val,A1_val)),W2) + b2
        A2_val = lineal(Z2_val)

        loss_val[i] = np.mean((A2_val - Y_val)**2)
        acc_val[i] = np.mean(np.abs(A2_val-Y_val) < 0.05)

        if loss[i]<0.1 and acc[i]>0.9 and t_conv==-1:
            t_conv=i

        dZ2 = 2*(A2 - Y)*lineal_p(Z2)
        dW2 = np.dot(np.hstack((X,A1)).T, dZ2)
        db2 = np.sum(dZ2, axis=0)
        dZ1 = dZ2 * g_p(Z1) * W2[1:,:].T
        dW1 = np.dot(X.T, dZ1)
```

```
        db1 = np.sum(dZ1, axis=0)

        W1 -= lr * dW1
        b1 -= lr * db1
        W2 -= lr * dW2
        b2 -= lr * db2
    return loss, acc, loss_val, acc_val, t_conv, W1, b1, W2, b2

def predict(X,Y,W1,b1,W2,b2):
    Z1 = np.dot(X,W1) + b1
    A1 = g(Z1)
    Z2 = np.dot(np.hstack((X,A1)),W2) + b2
    A2 = lineal(Z2)
    acc = np.mean(np.abs(A2-Y) < 0.05)
    return acc

def logistic_map(x,r=4):
    return r*x*(1-x)

x0=0.1
pares=[(x0,logistic_map(x0))]
for i in range(499):
    pares.append((pares[-1][1],logistic_map(pares[-1][1])))

train_size=[5,10,100]
fig, ax = plt.subplots(1,2, figsize=(15,5))
colors=["tab:blue","tab:orange","tab:green"]
ind=0

for size in train_size:
    indices_train=[]

    while len(indices_train)!=size:
        id=np.random.randint(0,500)
        if id not in indices_train:
            indices_train.append(id)
    x_train=np.array([pares[i][0] for i in indices_train]).reshape(-1,1)
    y_train=np.array([pares[i][1] for i in indices_train]).reshape(-1,1)

    indices_test=[]
    while len(indices_test)!=size:
        id=np.random.randint(0,500)
        if id not in indices_test and id not in indices_train:
            indices_test.append(id)
    x_test=np.array([pares[i][0] for i in indices_test]).reshape(-1,1)
    y_test=np.array([pares[i][1] for i in indices_test]).reshape(-1,1)

    loss, accuracy, loss_val, acc_val, t_conv, W1, b1, W2, b2 = fitLogistic(x_train,
        y_train, x_test, y_test, 5000, 0.005)
    accuracy_test=predict(x_test,y_test,W1,b1,W2,b2)
    print("Accuracy test: ",accuracy_test)

    ax[0].plot(loss,label="Size: "+str(size)+" (train)",c=colors[ind])
    ax[0].plot(loss_val,ls="--",label="Size: "+str(size)+" (validation)",c=colors[ind])
    ax[1].plot(accuracy,label="Size: "+str(size)+" (train)",c=colors[ind])
    ax[1].plot(acc_val,ls="--",label="Size: "+str(size)+" (validation)",c=colors[ind])
    ind+=1
ax[0].set_xlabel('Epoca')
```

```
ax[0].set_ylabel('Loss')
ax[0].set_yscale('log')
ax[1].set_xlabel('Epoca')
ax[1].set_ylabel('Accuracy')
ax[0].legend()
ax[1].legend()

plt.show()
```