Challenge № 5

Valerie Parra Cortés 16 de abril de 2020

Preprocesamientos de los datos

Lo primero que se hizo al descargar el dataset es eliminar todos los datos pertenecientes a las categorías que no hacían partes del problema de clasificación, se cambió la etiquetas de las clases a 0 o 1. *Jazz & blues* fue representado por la clase 0 mientras que la clase *Soul and Reagge* fue representada con 1.

Dependencias

Las dependencias para implementar la red son las que siguen a continuación

1. Keras Versión 2.3.1
2. Numpy Versión: 1.18.1
3. Sklearn Versión 0.22.2

Perfilamiento y división de los datos

Se contaba en total con 8350 datos, el 20 % de estos datos se separó para entrenamiento y el resto se dejo para entrenar el modelo. El resumen de los datos se muestra a continuación:

Clase	Entrenamiento	Validación	Totales
1	3192	824	4016
0	3488	846	4334
Total general			8350

Cuadro 1: División entrenamiento/validación de los datos

Para dividir estos datos se utilizó sklearn, se dividieron los datos de manera aleatoria, se seleccionó el 20 % para la validación y se dividieron los datos utilizando el siguiente código.

Listing 1: Código utilizado para separar el conjunto de entrenamiento del conjunto de validación.

```
1 from numpy import genfromtxt, newaxis, savetxt
2 from sklearn.model_selection import train_test_split
3
```

```
data = genfromtxt('clean_data.csv', delimiter=';')
labels= data[:,0][:,newaxis]
features = data[:,1:]
X_train, X_test, y_train, y_test = train_test_split(features, labels, ↩
    test_size=0.2)
savetxt("Xtrain",X_train,delimiter=";")
savetxt("Ytrain",y_train,delimiter=";")
savetxt("Xtest",X_test,delimiter=";")
savetxt("ytest",y_test,delimiter=";")
```

Selección de modelo

Para seleccionar el modelo se probaron varias funciones de error, número de neuronas, funciones de activación y tasas de aprendizaje, el resumen de los valores utilizados para estos hiperparámetros se muestra en la Tabla 2.

Hiperparámetro	**Valores**
Tasa de aprendizaje	0.01, 0.001
Funciones de activación	Relu, Sigmoide
Medida del error	Mean square error, binary crossentropy
Número de neuronas	1,10,50,100,200

Cuadro 2: Resultados

Resultados

La tabla completa con todos los resultados puede consultarse en la sección de Anexos, esta tabla solo incluye los datos finales de la validación.

Función de activación: Relu

Para visualizar de mejor manera estos resultados se realizaron gráficas en un eje tridimensional donde los ejes x & y tienen el número de neuronas y la tasa de aprendizaje utilizadas, mientras que el eje z tiene ya sea el error o la exactitud lograda. En la Figura 1 vemos que en general el valor final de la función objetivo es casi independiente del número de neuronas o la tasa de aprendizaje que se utilice, en general la función de error converge al mismo punto, lo que indica que el problema no es que el algoritmo no entrene bien o no converja sino que la función de activación puede no ser la adecuada. Adicionalmente en la Figura 2 vemos la exactitud final para los mismos modelos, podemos notar que para cualquier combinación de tasa de aprendizaje, función de error y número de neuronas el accuracy es prácticamente el mismo y apenas sobrepasa el 50 % por lo que el nivel de predicción de nuestro modelo en general es muy malo.

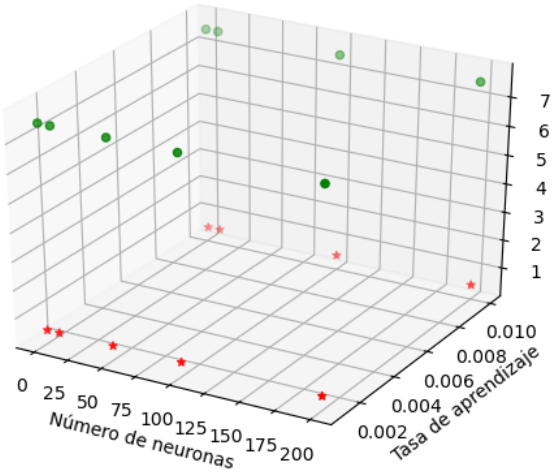


Figura 1: Resultados de para los experimentos con la función de activación *Relu*. En verde se ven los resultados usando como función de error *binary-cross-entropy* mientras que en rojo se ven los resultados utilizando *mean-square-error*, el eje z representa el valor final del error.

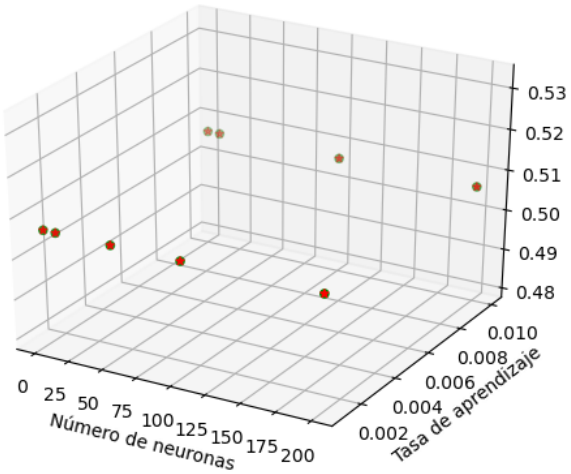


Figura 2: Resultados de para los experimentos con la función de activación *Relu*. En verde se ven los resultados usando como función de error *binary-cross-entropy* mientras que en rojo se ven los resultados utilizando *mean-square-error,* el eje z representa el accuracy en los datos de validación

Función de activación: Sigmoid

Para la función de activación sigmode se obtuvieron muchos mejores resultados que con la función *Relu*, como vemos en la figura 3. En general hay muy buenos modelos, podemos ver que el modelo con la menor tasa de aprendizaje 0.01 y 200 neuronas tiene el mejor resultado obteniendo exactitudes por encima del 75 % independiente de la función de error que se utilice, sin embargo hay un resultado un poco mejor para mean square error. En cuando a la

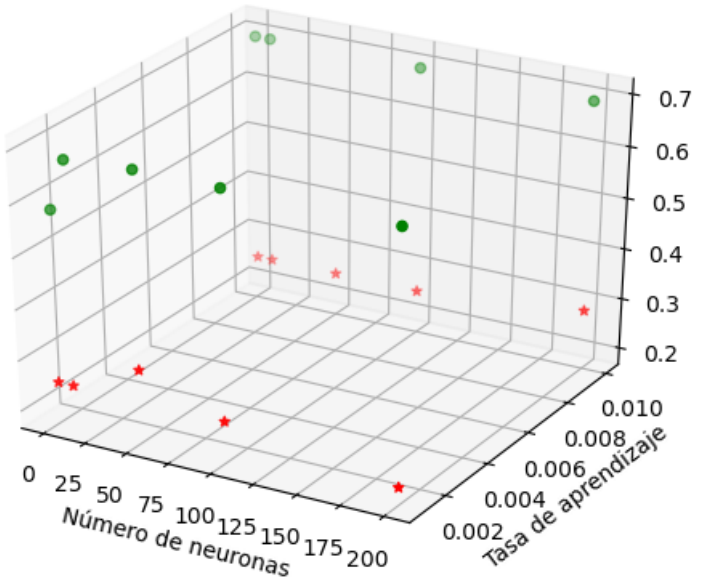


Figura 4: Resultados de para los experimentos con la función de activación *sigmoide*. En verde se ven los resultados usando como función de error binary-cross-entropy mientras que en rojo se ven los resultados utilizando mean square error, el eje z representa el valor final del error.

convergencia de los método en general se alcanza un menor valor de la función objetivo en los modelos que mayor exactitud tiene. Algo interesante es que para una neurona el valor de la función de error es menor tanto para *binary-cross-entropy* como para *mean-square-error* error. Finalmente se volverán a validar los modelos de esta función de activación con 200 neuronas y 0.01 de tasa de aprendizaje.

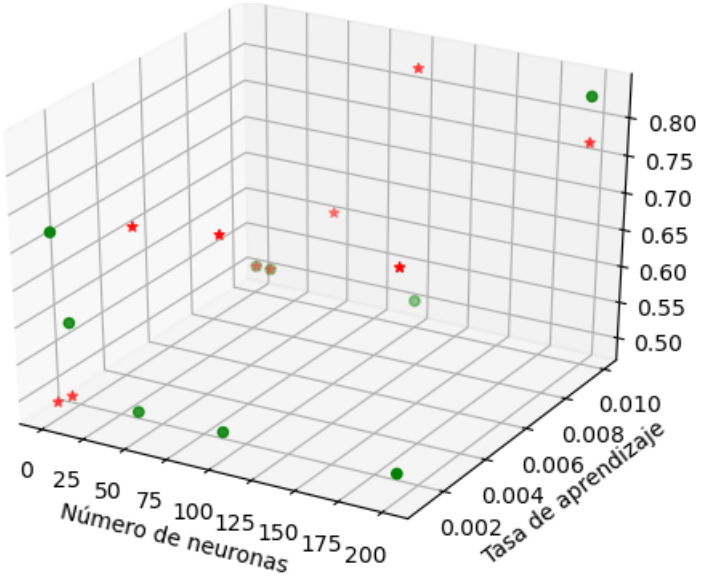


Figura 3: Resultados de para los experimentos con la función de activación *Sigmoid*. En verde se ven los resultados usando como función de error *binary-cross-entropy* mientras que en rojo se ven los resultados utilizando *mean-square-error*, el eje z representa el accuracy

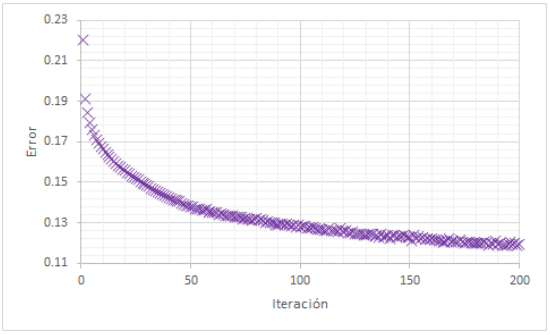
Estadísticas finales para modelos

Los modelos final a los cuales se les evaluó su comportamiento fueron:

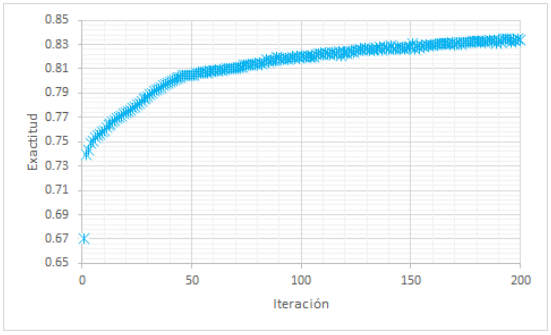
1. Función de activación: Relu
2. Tasa de aprendizaje: 0.01
3. Número de neuronas: 200
4. Función de error: binary cross entropy y mean square error

Utilizando entropía cruzada

Miramos el avance del primer modelo, la exactitud final de este modelo fue de 83 % para los datos de entrenamiento y 80 % para los datos de validación. En la Figura 6a vemos como la función de error no parece tener problemas para converger y en la Figura 6b vemos que la entropía se comporta de manera opuesta al error aumenta con el número de iteraciones.



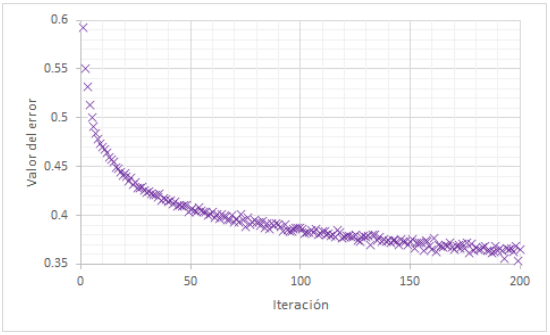
(a) Avance de la función de error con las iteraciones



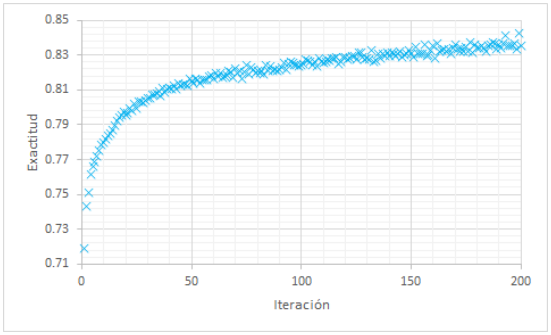
(b) Avance de la exactitud en los datos de entrenamiento

Utilizando Mean Square error

Con el error cuadrático tenemos resultados muy parecido que el al usar Binary cross entropy, tenemos los mismos valores en las exactitudes finales (83,4 % para validación y 80,0 % para los datos de validación). En cuanto a la convergencia de la función de error, converge bien a un valor pequeño, por lo que podemos decir que la arquitectura en general independiente de este hiperparámetro (para las funciones de error utilizadas, es decir tocaría probar otras funciones de activación como lineal, hard simoid, etc para poder hacer la generalización) y son los otros hiperparámetros los que permiten una alta exactitud en el modelo.



(a) Avance de la función de error con las iteraciones



(b) Avance de la exactitud en los datos de entrenamiento

Conclusiones

La validación cruzada nos permite encontrar el mejor modelo, en esta caso en particular vimos que hay hiperparámetros al cual el problema es más sensible, por ejemplo la función de activación. Cuando utilizabamos Relu, independientemente de los demás hiperparámetros, los resultados eran malos, por lo que está función no es adecuada para este problema. Por otro lado, la función sigmoide tenía resultados muy variados para las arquitecturas probadas, teniendo su mejor desempeño en 200 neuronas obteniendo una exactitud final del 80 % para ambas funciones de error utilizadas. Comparando estos resultados obtenidos en el Challenge 2 tenemos una ligera mejora (del menos de 2 %) pero a consta de mucho más poder computacional, este es un modelo alrededor de 200 veces más complejo. Esto es coherente con el error final logrado en la red con una neurona y función de activaciόm sigmoide, qué es lo que más se asemejaría a una regresión logística que fue lo que se impmplemento en el reto 2. Por estas razones para este problema en particular quizás sería mejor implementar una regresión logistíca a una red neuronal.

Anexos

Implementación

Para implementación la validación de los modelos se utilizaron las librerías ya mencionadas en la sección de dependencias. La implementación completa se muestra a continuación:

Listing 2: Código utilizado para separar el conjunto de entrenamiento del conjunto de validación.

```
1 from numpy import genfromtxt, newaxis, savetxt
2 from sklearn.model_selection import train_test_split
3 from keras.models import Sequential
4 from keras.layers import Dense
5 from keras import optimizers
6 from sklearn.metrics import confusion_matrix
```

```
import os
import glob
import csv
from numpy import genfromtxt
X_train= genfromtxt('Xtrain.txt', delimiter=';')
X_test= genfromtxt('Xtest.txt', delimiter=';')
y_train= genfromtxt('Ytrain.txt', delimiter=';')
y_test= genfromtxt('ytest.txt', delimiter=';')

# define the keras model
model = Sequential()
summary = open("summarySigmoid.csv", "w")
summary.write("af,loss,lr,nn,loss,aca \n")

epochNumber= 200
activationFunctions= ['sigmoid']
learningRates=[0.001,0.01]
loss=['binary_crossentropy','mean_squared_error']
nn= [1,10,50,100,200]
for activation in activationFunctions:
    for lr in learningRates:
        for l in loss:
            for n in nn:
                model.add(Dense(n, input_dim=30, activation=activation))
                model.add(Dense(1, activation=activation))
                pathTrainLoss="./results/train/af/"+activation+"/loss/"+l+"/lr/"+"0.00001"+"/"+str(n)+"_loss.csv"
                pathTrainAccuracy="./results/train/af/"+activation+"/loss/"+l+"/lr/"+"0.00001"+"/"+str(n)+"_accuracy.csv"
                pathAccuracy="./results/validation/af/"+activation+"/loss/"+l+"/lr/"+"0.00001"+"/"+str(n)+".csv"
                sgd = optimizers.SGD(lr=lr)
                model.compile(sgd, loss=l, metrics=['accuracy'])
                history_callback= model.fit(X_train, y_train, epochs=epochNumber, verbose=0, steps_per_epoch=50)
                accuracy_history = history_callback.history["accuracy"]
                loss_history = history_callback.history["loss"]
                predictions = model.predict_classes(X_test)
                accuracy = model.evaluate(X_test, y_test)
                print(activation+","+l+","+str(lr)+","+str(n)+","+str(accuracy[0])+","+str(accuracy[1]))
                summary.write(activation+","+l+","+str(lr)+","+str(n)+","+str(accuracy[0])+"\n")

summary.close()
```

Función activación	Función error	Tasa aprendizaje	Número de neuronas	Valor final del error	Exactitud
relu	binary_crossentropy	0.001	1	7.61087277	0.5065868
relu	binary_crossentropy	0.001	10	7.61087277	0.5065868
relu	binary_crossentropy	0.001	100	7.61087277	0.5065868
relu	binary_crossentropy	0.001	200	7.61087277	0.5065868
relu	binary_crossentropy	0.001	50	7.61087277	0.5065868
relu	binary_crossentropy	0.01	1	7.61087277	0.5065868
relu	binary_crossentropy	0.01	10	7.61087277	0.5065868
relu	binary_crossentropy	0.01	100	7.61087277	0.5065868
relu	binary_crossentropy	0.01	200	7.61087277	0.5065868
relu	binary_crossentropy	0.01	50	7.61087277	0.5065868
relu	mean_squared_error	0.001	1	0.49341317	0.5065868
relu	mean_squared_error	0.001	10	0.49341317	0.5065868
relu	mean_squared_error	0.001	100	0.49341317	0.5065868
relu	mean_squared_error	0.001	200	0.49341317	0.5065868
relu	mean_squared_error	0.001	50	0.49341317	0.5065868
relu	mean_squared_error	0.01	1	0.49341317	0.5065868
relu	mean_squared_error	0.01	10	0.49341317	0.5065868
relu	mean_squared_error	0.01	100	0.49341317	0.5065868
relu	mean_squared_error	0.01	200	0.49341317	0.5065868
relu	mean_squared_error	0.01	50	0.49341317	0.5065868
sigmoid	binary_crossentropy	0.001	1	0.58556088	0.7227545
sigmoid	binary_crossentropy	0.001	10	0.68484106	0.6065868
sigmoid	binary_crossentropy	0.001	100	0.69354571	0.5065868
sigmoid	binary_crossentropy	0.001	200	0.69354582	0.5065868
sigmoid	binary_crossentropy	0.001	50	0.69354581	0.5065868
sigmoid	binary_crossentropy	0.01	1	0.69354589	0.5065868
sigmoid	binary_crossentropy	0.01	10	0.69354571	0.5065868
sigmoid	binary_crossentropy	0.01	100	0.6935458	0.5065868
sigmoid	binary_crossentropy	0.01	200	0.69354581	0.8323739
sigmoid	binary_crossentropy	0.01	50	0.69354576	0.7431284
sigmoid	mean_squared_error	0.001	1	0.25052293	0.4934132
sigmoid	mean_squared_error	0.001	10	0.25016979	0.5065868
sigmoid	mean_squared_error	0.001	100	0.25019902	0.7674929
sigmoid	mean_squared_error	0.001	200	0.20183473	0.7753748
sigmoid	mean_squared_error	0.001	50	0.3123123	0.7535287
sigmoid	mean_squared_error	0.01	1	0.25019899	0.5065868
sigmoid	mean_squared_error	0.01	10	0.25019898	0.5065868
sigmoid	mean_squared_error	0.01	100	0.25019899	0.8257593
sigmoid	mean_squared_error	0.01	200	0.283713	0.7717264
sigmoid	mean_squared_error	0.01	50	0.25019901	0.6065868

Figura 7: Tabla con todos los resultados de la validación cruzada