Trabajo Práctico 3: Machine Learning

Grupo 4:
. Araujo Croizier, Juan I. 10649
. Linares, Ramiro 10777

Ejercicios

1. **Familiarizarse con el código de la red neuronal feedforward fully connected de 1 capa oculta explicada en clase**

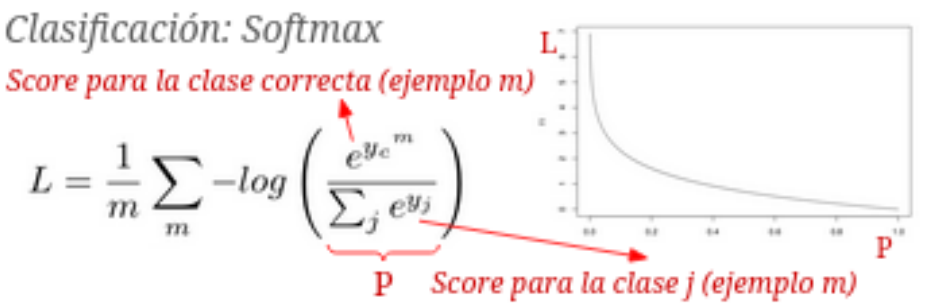
Se leyeron los apuntes de cátedra y se revieron las clases de teoría y práctica además de buscar información y videos adicionales de clustering, clasificación y perceptrones para familiarizarse con los temas.

2. **Modificar el programa para que:**

   a. **Mida la precisión de clasificación (accuracy) además del valor de Loss**

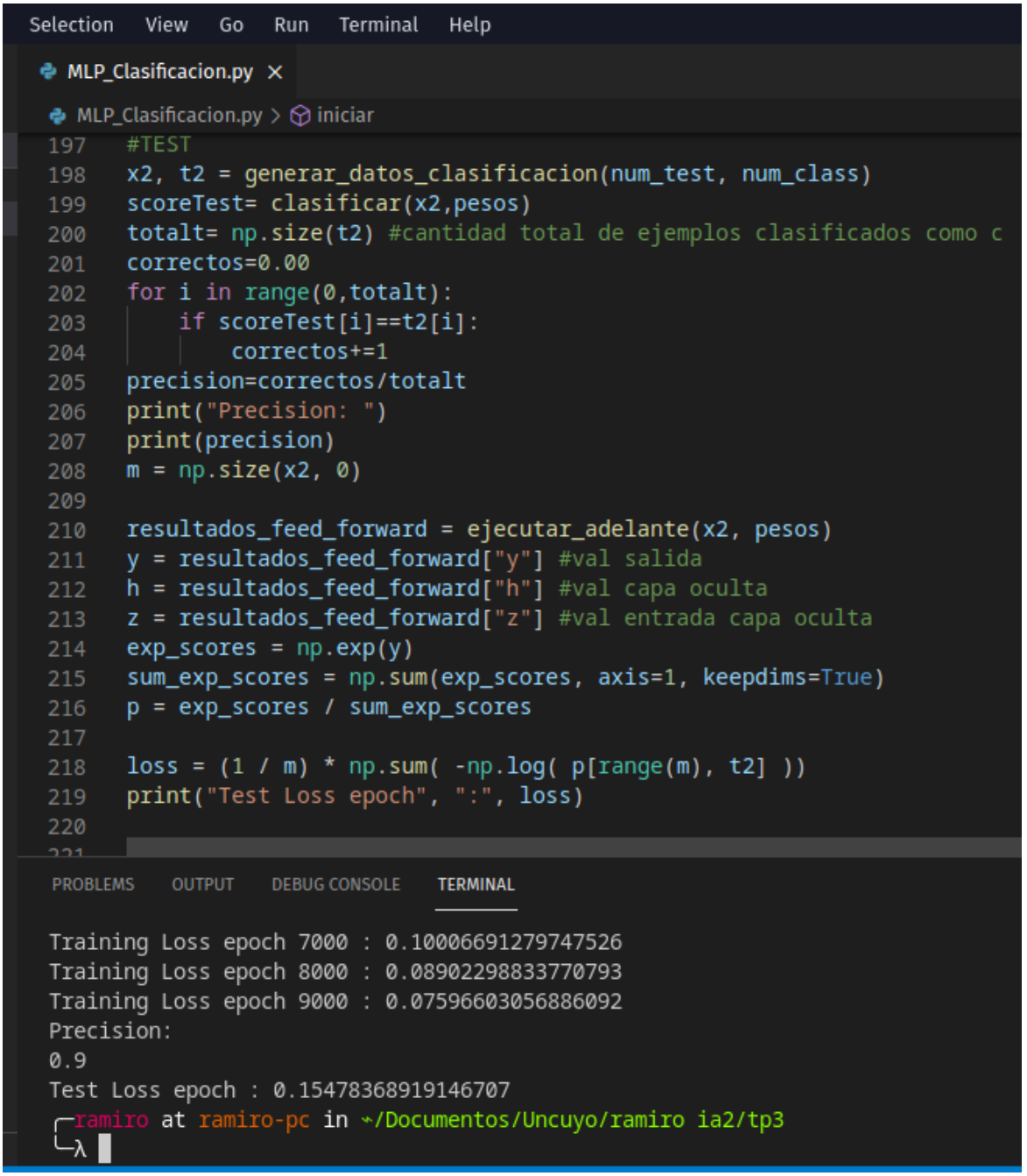
   Para este punto, se codificaron en el programa las ecuaciones vistas en teoría:

$$\text{Precision}_C = \frac{\text{Cantidad de ejemplos clasificados como C } \textbf{correctamente}}{\text{Cantidad total de ejemplos clasificados como C}} \quad \text{(ej para 2 clases: TP / (TP + FP)}$$

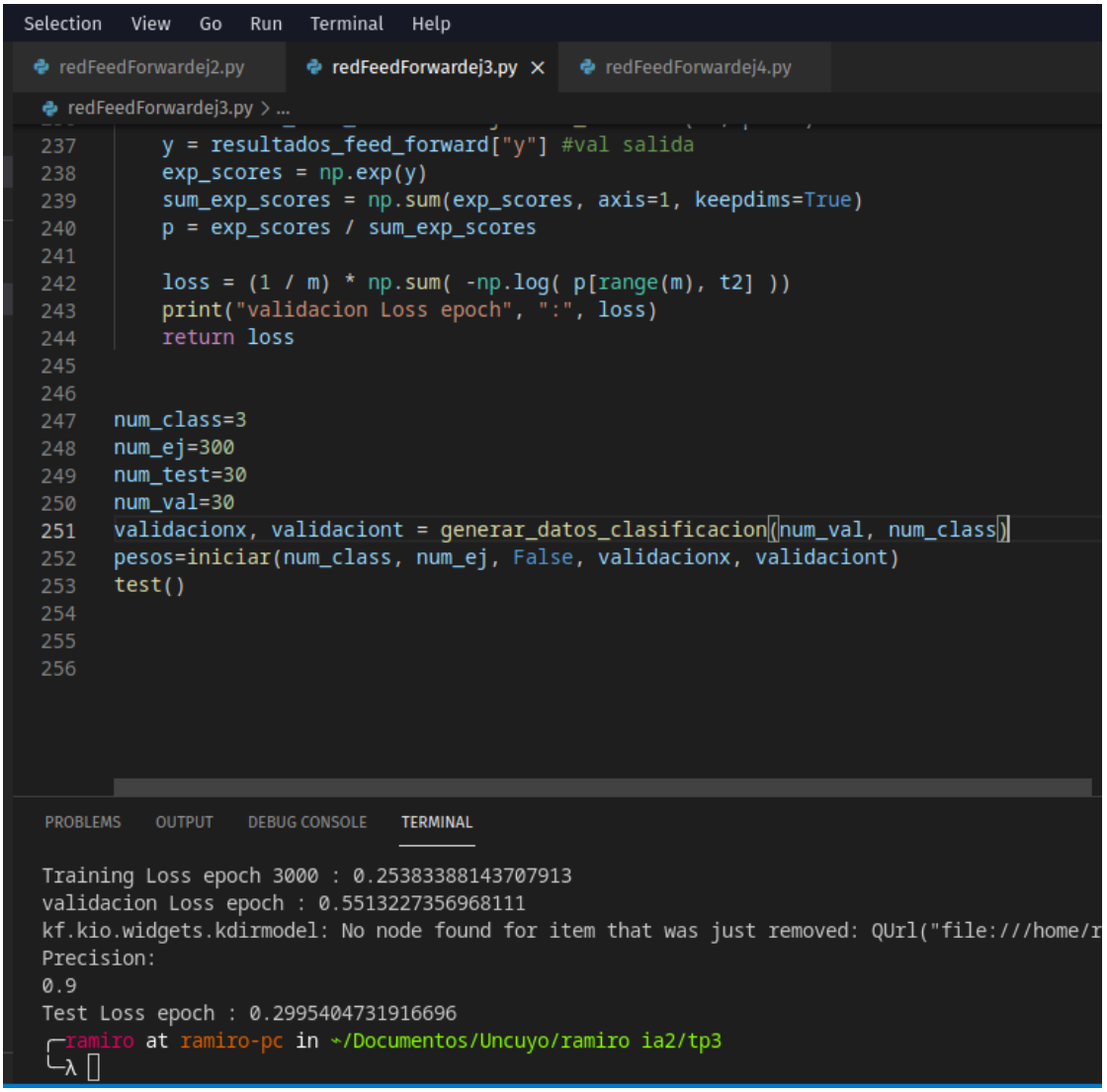


   b. **Utilice un conjunto de test independiente para realizar dicha medición (en lugar de utilizar los mismos datos de entrenamiento). Este punto requiere generar más ejemplos.**

   Una vez entrenada la Red Feedforward, se recurrió a un conjunto generado previamente de manera independiente, denominado "Test". Se expuso a la Red este conjunto, se midió entonces la precisión y el valor de Loss del Test en la Red entrenada.
   Se puede observar que el valor de precisión da 0.9 y que el valor de Loss es mayor que dicho valor obtenido para los ejemplos de Training. Esto se debe a que estos ejemplos de Test no han sido probados nunca, y probablemente la Red haya llegado a tener algo de Overfitting, al experimentar muchas veces con los mismo ejemplos de Training.



3. **Agregar parada temprana, utilizando un conjunto de validación, distinto del conjunto de entrenamiento y de test (este punto requiere generar más ejemplos). Esto es: verificar el valor de loss o de accuracy cada N epochs (donde N es un parámetro de configuración) utilizando el conjunto de validación, y detener el entrenamiento en caso de que estos valores hayan empeorado (puede incluirse una tolerancia para evitar cortar el entrenamiento por alguna oscilación propia del proceso de entrenamiento).**

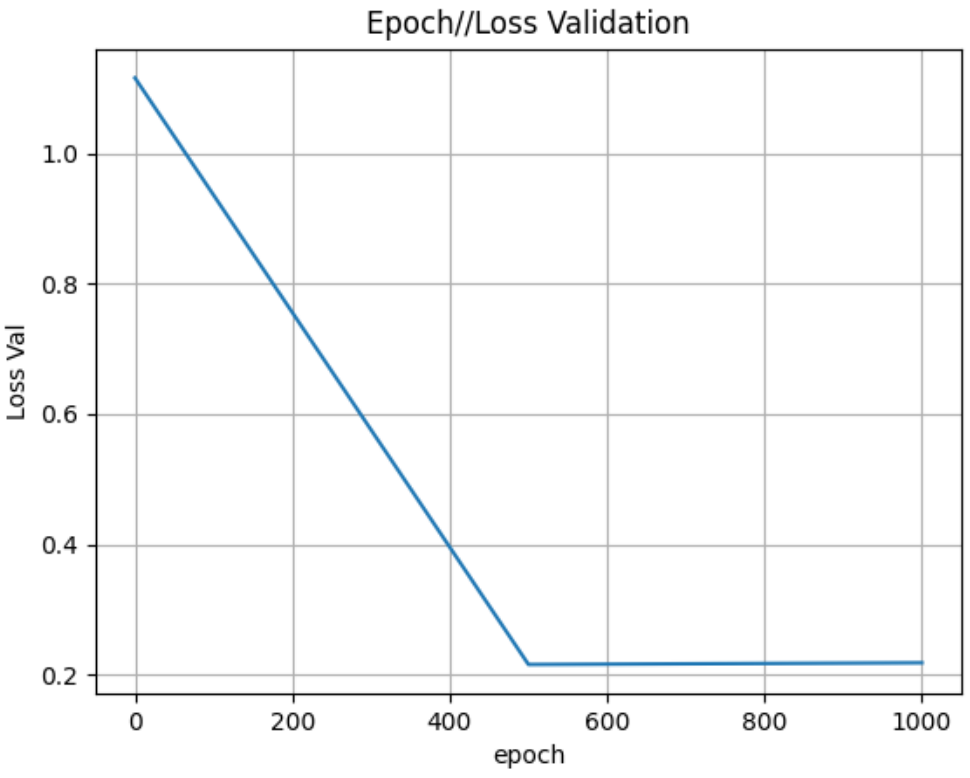


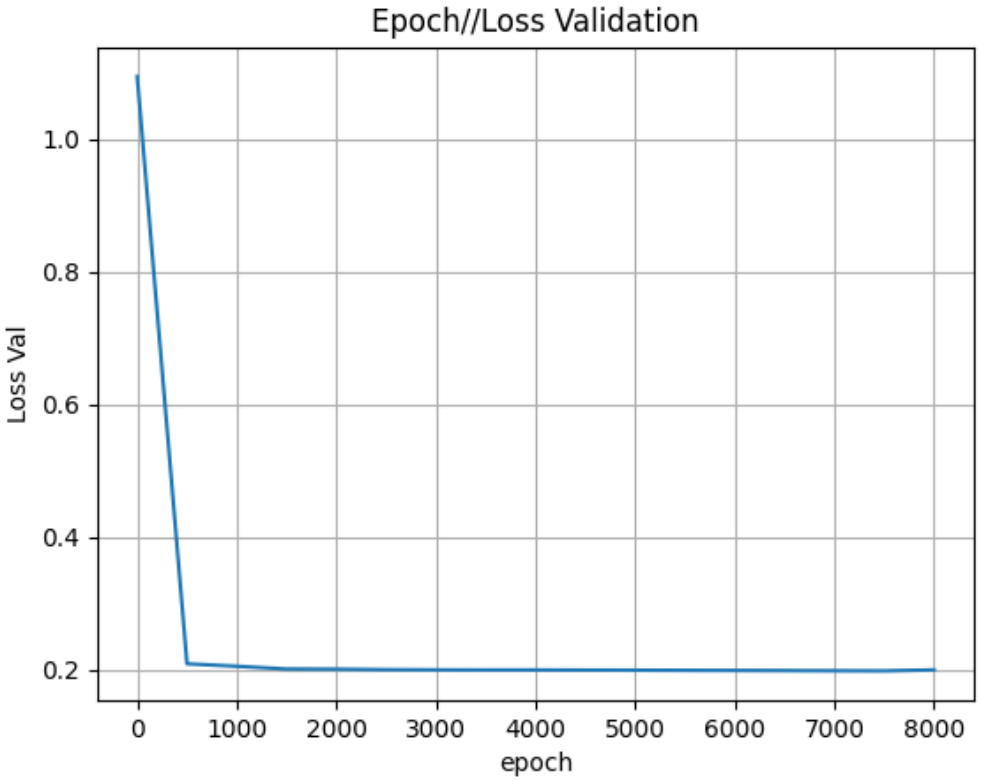
Antes de entrenar o generar los datos de entrenamiento, se generó un grupo de datos de validación distinto del conjunto de Training y de Test.

Cada 500 epochs, se comprobará el valor de Loss del conjunto de validación, en caso de que el Loss empeore con respecto al valor anterior, se cortará el entrenamiento.

```
Selection  View  Go  Run  Terminal  Help
redFeedForwardej2.py | redFeedForwardej3.py | redFeedForwardej4.py
redFeedForwardej3.py > train
123             # d. Calculo de la funcion de perdida global. Solo se usa la probabilidad de la clase corre
124             #    que tomamos del array t ("target")
125             loss = (1 / m) * np.sum( -np.log( p[range(m), t] ))
126             # Mostramos solo cada 1000 epochs
127             if i %500 == 0:
128                 print("Training Loss epoch", i, ":", loss)
129                 lossVal=validacion(validacionx, validaciont,pesos)
130                 epochGraf.append(i)
131                 lossValGraf.append(lossVal)
132                 if (lossVal>lossValAnt):
133                     condValidacion=False
134                     fig, ax = plt.subplots()
135                     ax.plot(epochGraf, lossValGraf)
136
137                     ax.set(xlabel='epoch', ylabel='Loss Val', title='Epoch//Loss Validation')
138                     ax.grid()
139                     plt.show()
140                     break
141                 else:
142                     lossValAnt=lossVal
143
144             #medir accuracy,  aciertos/cant eje; loss medir con ejemplos test. no estan ni en training
145             # Extraemos los pesos a variables locales
146             w1 = pesos["w1"]
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                                    fish
Training Loss epoch 3000 : 0.25383388143707913
validacion Loss epoch : 0.5513227356968111
kf.kio.widgets.kdirmodel: No node found for item that was just removed: QUrl("file:///home/ramiro/Escritorio/.direct
Precision:
0.9
Test Loss epoch : 0.2995404731916696
ramiro at ramiro-pc in ~/Documentos/Uncuyo/ramiro ia2/tp3
λ
```

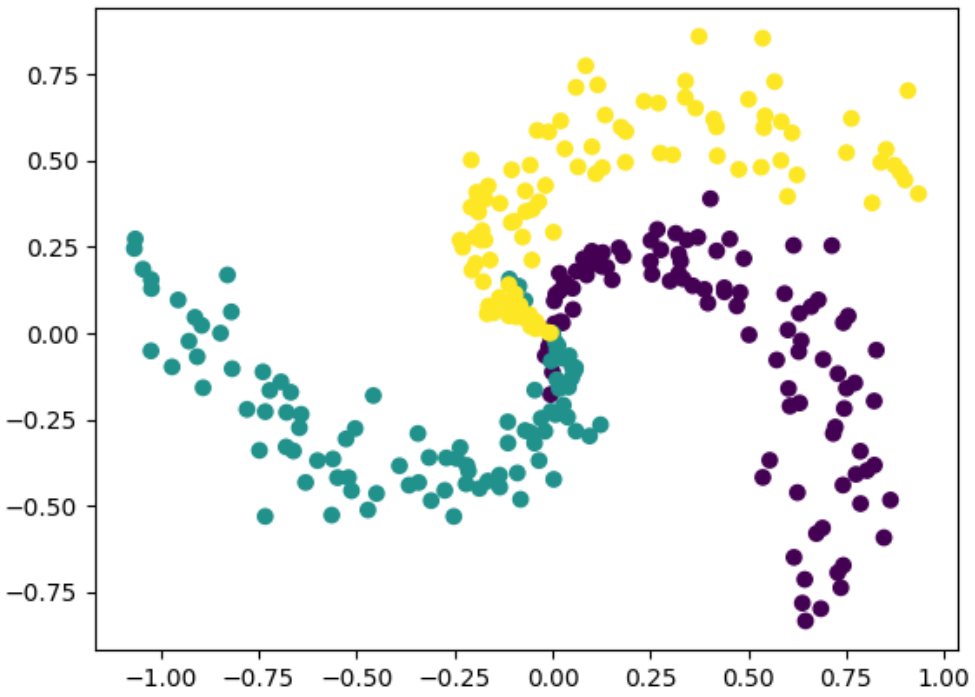
A continuación, se muestran 2 ejemplos:



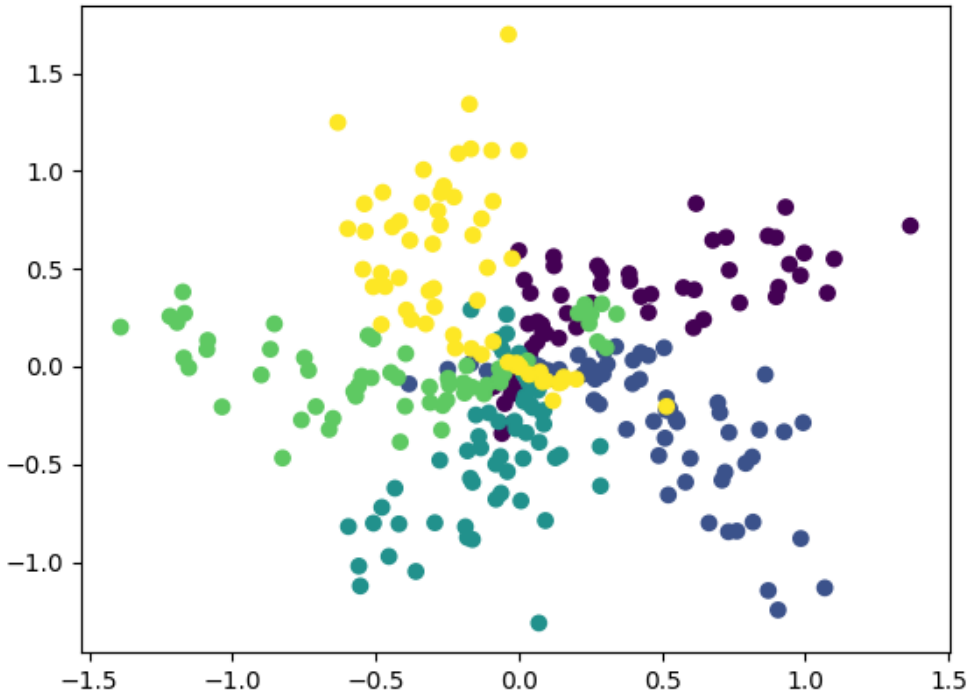


4. **Experimentar con distintos parámetros de configuración del generador de datos para generar sets de datos más complejos (con clases más solapadas, o con más clases). Alternativamente, experimentar con otro generador de datos distinto (desarrollado por usted). Evaluar el comportamiento de la red ante estos cambios.**

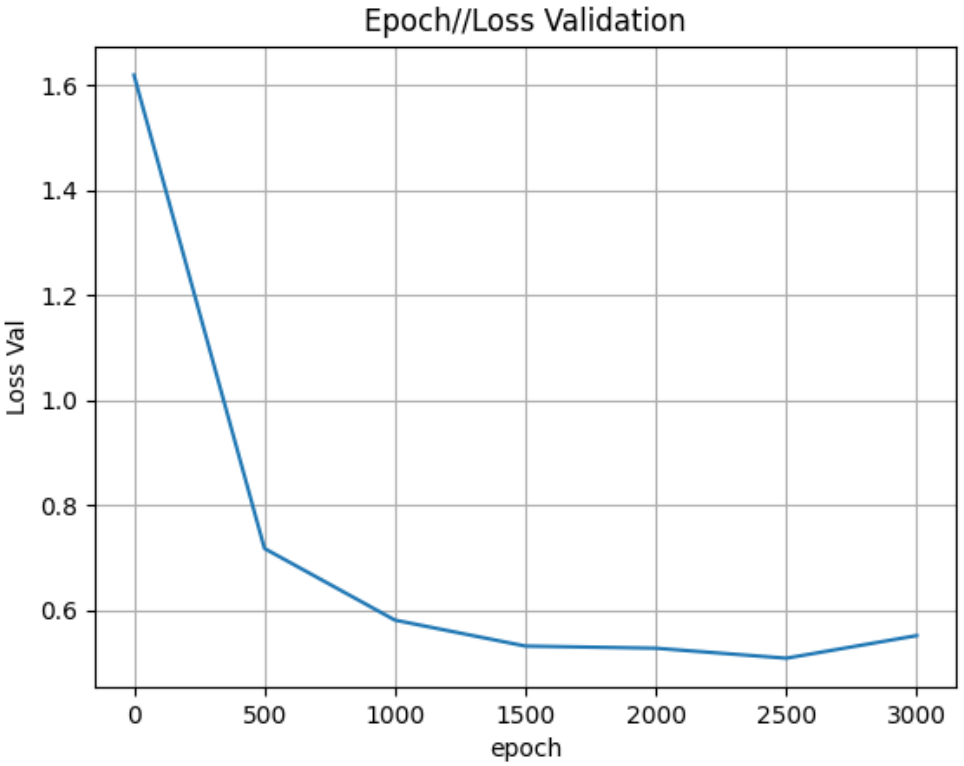
Los datos originales eran:



Aumentamos la cantidad de clases a 5, disminuimos el factor de ángulo y aumentamos en gran medida el valor de aleatoriedad, por lo que generamos clases más complejas (pero del mismo tipo).

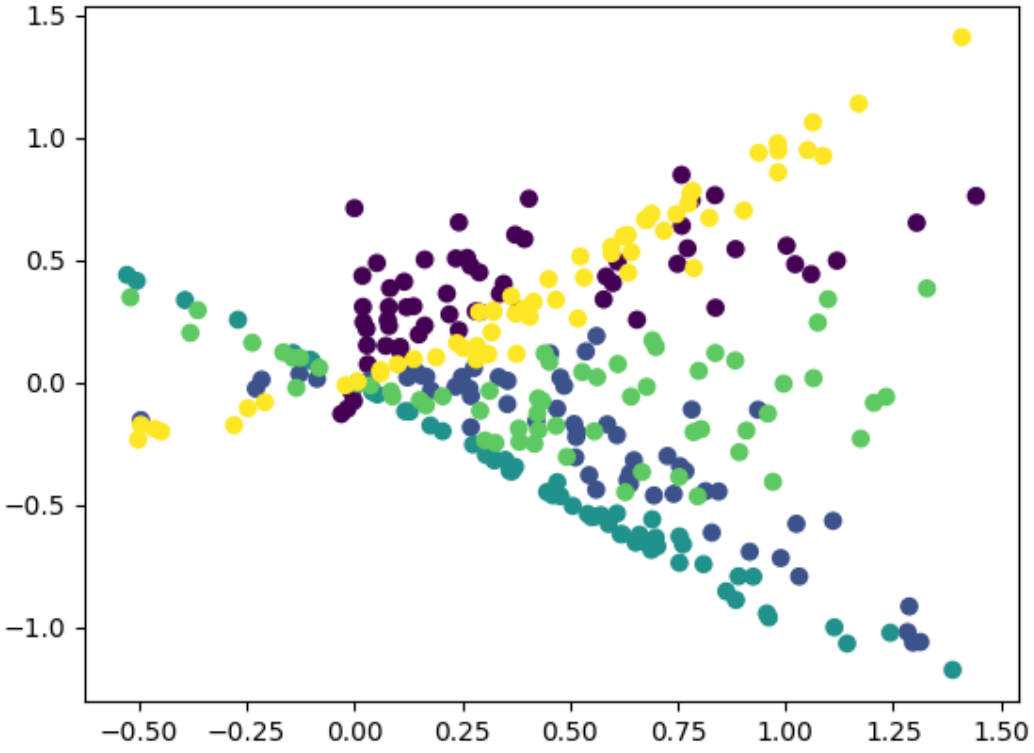


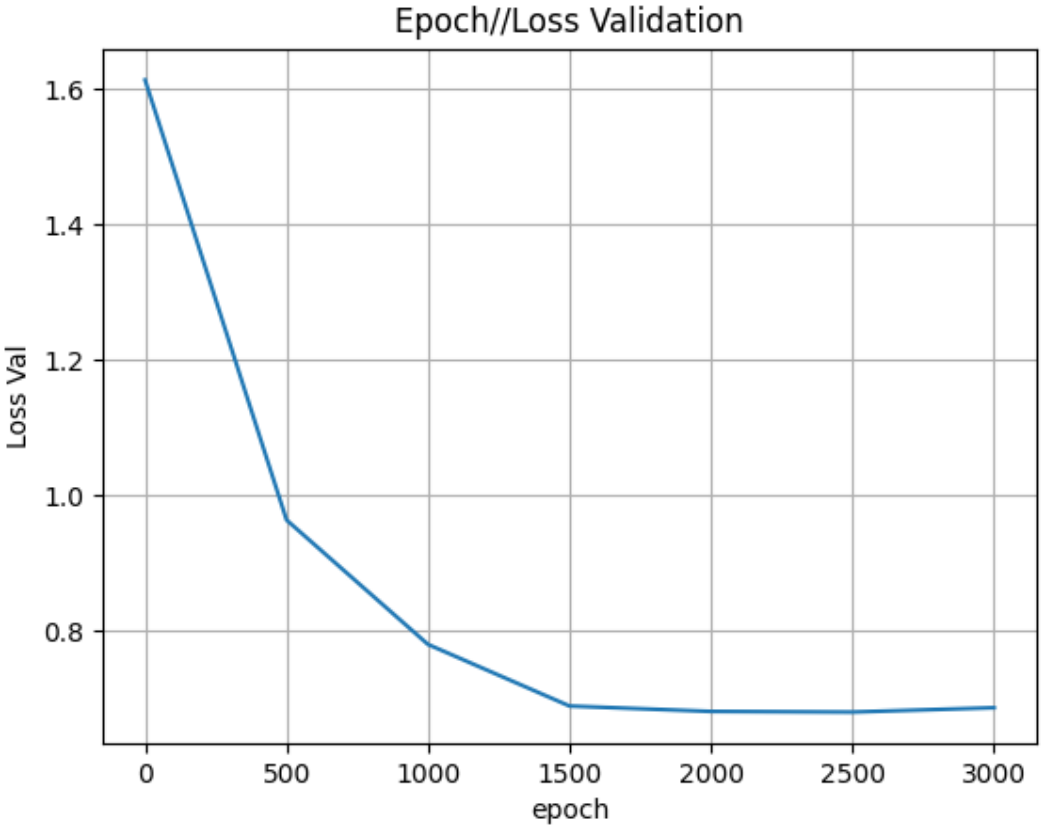
El rendimiento en este caso del valor de Loss del conjunto de validación fue:



Como se esperaba, al generar clases más complejas la red se comporta de peor manera ante estos cambios (mayor valor de loss).

En el caso de cambiar el tipo de datos (ligeramente)





En comparación al cambio de parámetros anterior, se puede observar que el valor de Loss es menor, se debe a que las clases amarillas y azules son más lineales que las anteriores (y menos complejas en comparación)

5. Modificar el programa para que funcione para resolver problemas de regresión

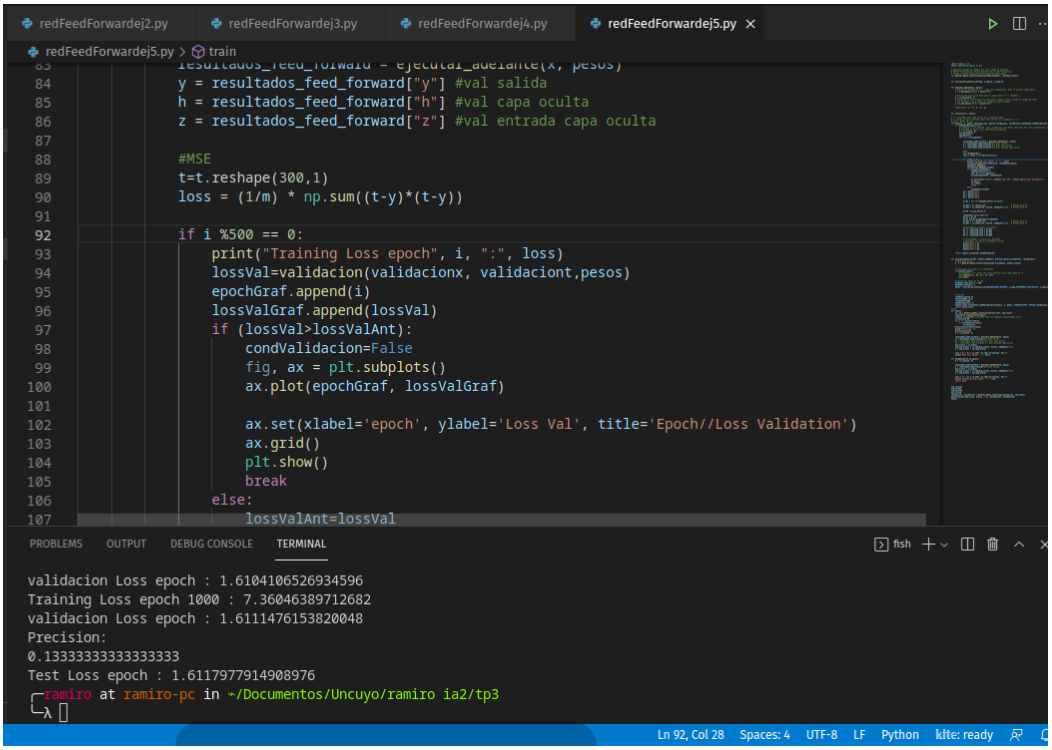
a. Debe modificarse la función de pérdida y sus derivadas, utilizando por ejemplo MSE

Anteriormente se calculaba la función de Loss por la clasificación SoftMax, modificaremos la función de pérdida utilizando el error medio cuadrático dado por

$$L_i(W) = (t_i - y_i)^2$$

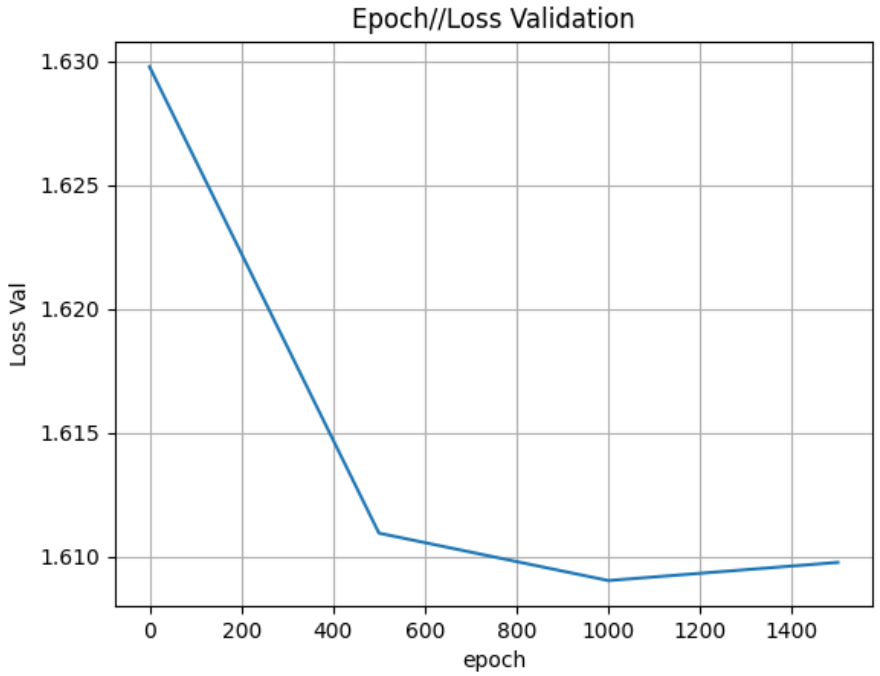
$$L(W) = \frac{1}{m} \sum_i L_i(W)$$

Cambiamos SoftMax y sus derivadas utilizando MSE:

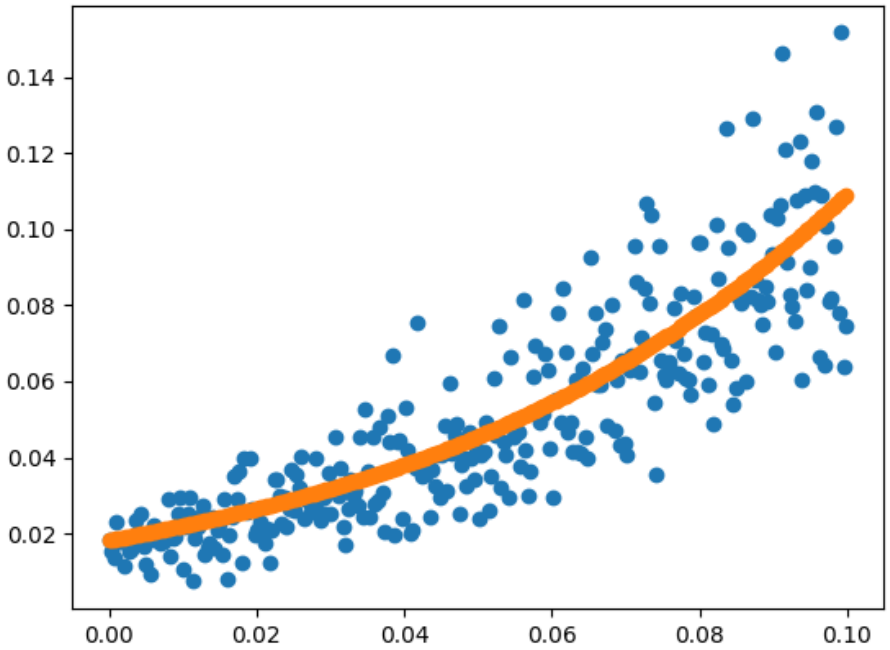


```
redFeedForwardej2.py  redFeedForwardej3.py  redFeedForwardej4.py  redFeedForwardej5.py
redFeedForwardej5.py > train
110         w2 = pesos["w2"]
111         b2 = pesos["b2"]
112
113         dL_dy = -2* (t.reshape((len(t),1))-y)/m
114
115         dL_dw2 = h.T.dot(dL_dy)                          # Ajuste para w2
116         dL_db2 = np.sum(dL_dy, axis=0, keepdims=True)    # Ajuste para b2
117
118         dL_dh = dL_dy.dot(w2.T)
119
120         sigmoide=1/(1+np.exp(-z))
121         dL_dz = dL_dh
122         dL_dz =dL_dz *sigmoide*(1-sigmoide)
123         dL_dw1 = x.T.dot(dL_dz)                          # Ajuste para w1
124         dL_db1 = np.sum(dL_dz, axis=0, keepdims=True)    # Ajuste para b1
125
126         # Aplicamos el ajuste a los pesos
127         w1 += -learning_rate * dL_dw1
128         b1 += -learning_rate * dL_db1
129         w2 += -learning_rate * dL_dw2
130         b2 += -learning_rate * dL_db2
131
132         # Actualizamos la estructura de pesos
133         # Extraemos los pesos a variables locales

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                   fish

validacion Loss epoch : 1.6104106526934596
Training Loss epoch 1000 : 7.36046389712682
validacion Loss epoch : 1.6111476153820048
Precision:
0.13333333333333333
Test Loss epoch : 1.6117977914908976
ramiro at ramiro-pc in ~/Documentos/Uncuyo/ramiro ia2/tp3
λ

Ln 121, Col 33  Spaces: 4  UTF-8  LF  Python  klite: ready
```

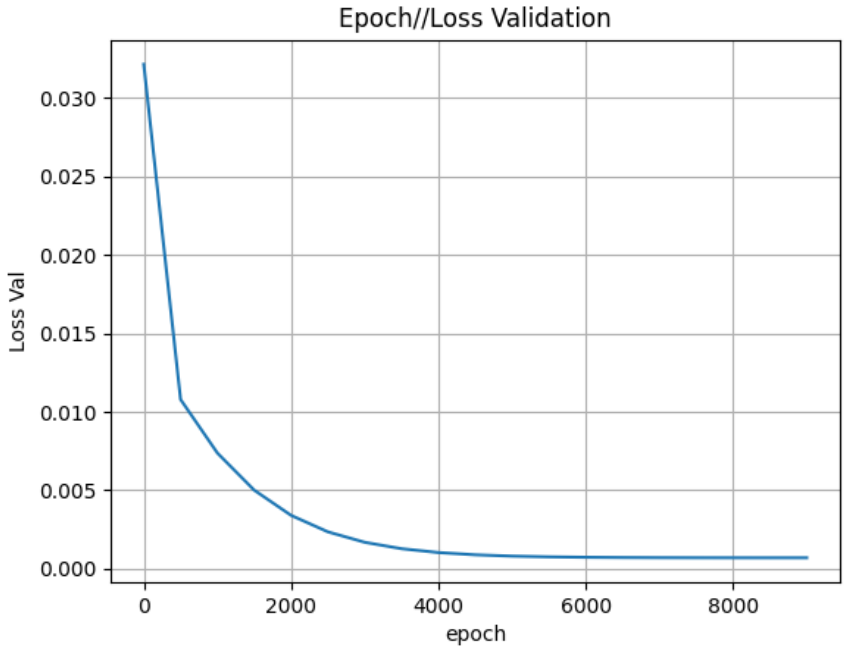
Los resultados fueron:



b. Debe crearse un generador de datos nuevo para que genere datos continuos (pueden mantenerse igualmente 2 entradas; en caso de usar más entradas puede requerirse más capas en la red neuronal
Cambiamos la clase generar_datos

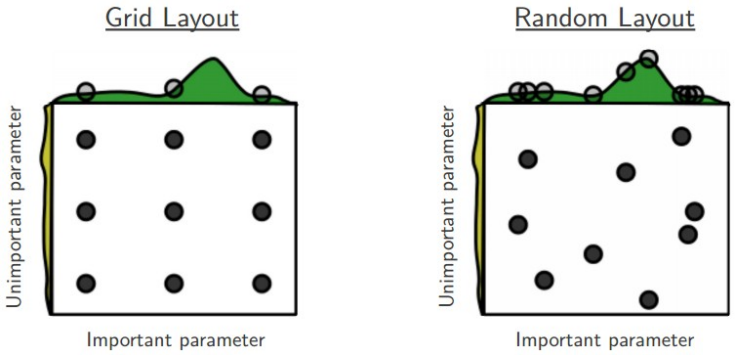


Generamos una función cuadrática continua, se normalizaron los datos, y se predijeron los valores esperados con la condición de parada definida anteriormente que era siempre y cuando el valor de loss del conjunto de validación no empeorará (con MSE)



6. Realizar un barrido de parámetros (learning rate, cantidad de neuronas en la capa oculta, comparación de ReLU con Sigmoide)

Tenemos dos opciones de barrido de parámetros



Los valores que modificaremos serán learning rate y cantidad de neuronas en la capa oculta primero con sigmoide,y luego con relu para comparar resultados. Utilizaremos solo el valor de precisión dado por el conjunto test para simplificar resultados

MSE

LearningRate/ NeuronasOcultas	0.1	0.25	0.5	0.75	0.9
50	**0.1666**	**0.133**	**0.2**	**0.233**	**0.2**
100	**0.1333**	**0.3**	**0.233**	**0.166**	**0.2**
250	**0.1**	**0.2**	**0.2**	**0.2**	**0.233**
500	**0.3**	**0.166**	**0.133**	**0.2**	**0.2**

SoftMax

LearningRate/ NeuronasOcultas	0.1	0.25	0.5	0.75	0.9
50	**0.9**	**0.933**	**0.8**	**0.866**	**0.966**
100	**0.966**	**0.833**	**0.966**	**0.9**	**0.9**
250	**0.9**	**0.933**	**0.966**	**0.933**	**0.98**
500	**0.966**	**0.966**	**0.9**	**0.966**	**0.9**

Pudimos observar que el learning rate no afecta la velocidad del algoritmo, mientras que las neuronas ocultas afectaba la velocidad de ejecución considerablemente.
Debemos tener en cuenta que estos valores no son del todo representativo por dos razones, una se hizo un barrido de parámetros en grilla (cuando es recomendable que sea aleatorio), y segundo la generación de datos misma es aleatoria por lo que ejecutar el algoritmo dos veces no daría exactamente el mismo resultado (se debería ejecutar varias veces por cada modificación de hiperparametros y promediar los resultados)

Solamente con estos datos, podríamos concluir que los valores recomendables para nuestros hiperparámetros serían tener un learning rate de 0.5 y un cantidad de neuronas en la capa oculta de 100 ya que en ambos casos nos dieron unos resultados aceptables, y el tiempo de ejecución del algoritmo es aceptable también, mientras que en los otros casos el tiempo de ejecución se extendía en gran medida.