UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

INTELIGENCIA ARTIFICIAL II
UNIDAD 3: MACHINE LEARNING

Grupo Nº1

Alumnos:

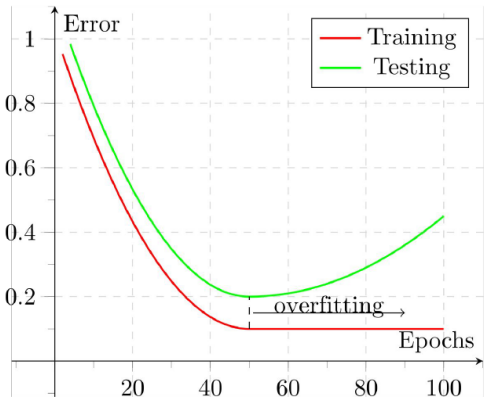
- ***Eula, Adriano German (L:12336)***
- ***Reinoso, Maximiliano Gabriel (L:11754)***
- ***Sena, Julieta (L:11367)***

Resolución de los ejercicios:
https://colab.research.google.com/drive/1TO1_IzMlAHrXcIYwFrkKW0OYbNI0wZDT#scrollTo=kxz9dk2ylGd8&uniqifier=2

Ejercicios 2,3 y 4: El tema a abordar en estos 3 primeros ejercicios planteados fueron de clasificación. Para ello se tomó como base el código visto en la clase.

Precisión de Clasificación: Cuando calculamos el valor de la precisión de la clasificación utilizamos datos del conjunto de validación provistos para realizar una prueba de clasificación de los datos. Para ello comparamos el vector de salida [y] con el vector de salida real [t] o también llamado "target". Además, hace la cuenta de cantidad de aciertos divididos por la cantidad de ejemplos y de esta manera obtenemos la precisión de la clasificación.

Parada temprana: Cuando corremos la función training luego de cierta cantidad de epochs, realizamos una prueba de validación. Esto se logra corriendo el programa con un conjunto de entradas diferentes a las utilizadas en el training y de esta manera obtenemos la precisión de validación. Vamos ir comparando estos valores a medida que se va entrenando el algoritmo para tener noción si el algoritmo va generalizando bien la clasificación de los puntos y no caer en un "overfitting". Se tiene que tener en cuenta una tolerancia por defecto debida a oscilaciones del propio algoritmo a medida que se realiza el training. En caso de que la precisión de validación caiga por debajo de dicho margen, realizamos la parada temprana del algoritmo.



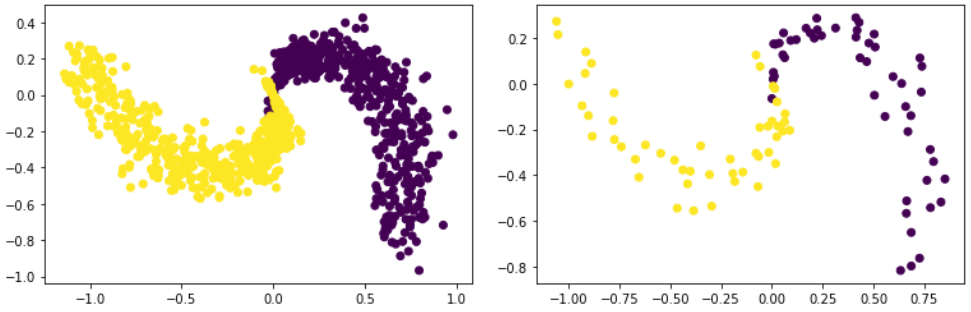
Overfitting

En este segmento del programa como primera opción analizamos la misma configuración de generación de datos propuesta en el código base de la cátedra. Para ello agregamos más clases y fuimos observando las precisiones de testeo en cada caso. Para la primera opción utilizamos la comparación para clasificación de 2,3,4 y 5 clases.

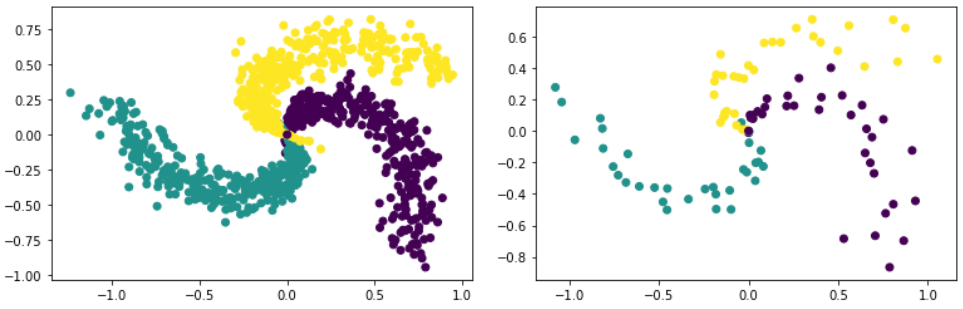
Las ilustraciones a continuación están ordenadas de izquierda a derecha con los gráficos de la distribución de datos de training y la distribución de datos de testeo

Parámetros utilizados:
EPOCHS=10000
Validación cada 1000 EPOCHS
Neuronas capa oculta=100
Learning rate=1
Cantidad ejemplos training=1000
Cantidad ejemplos validación=300
Cantidad ejemplos test=100

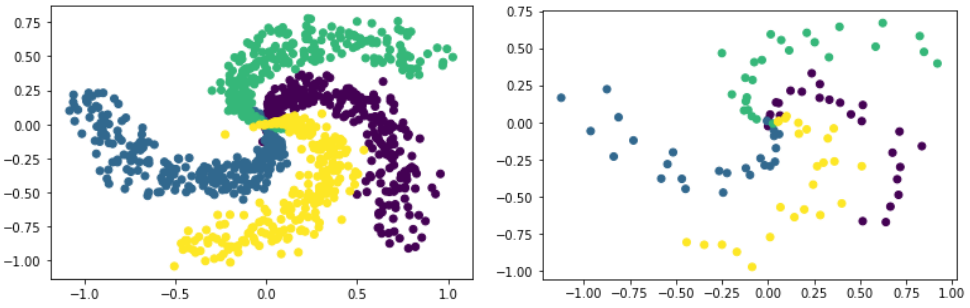
Clases = 2: *Precisión de testeo 98%*



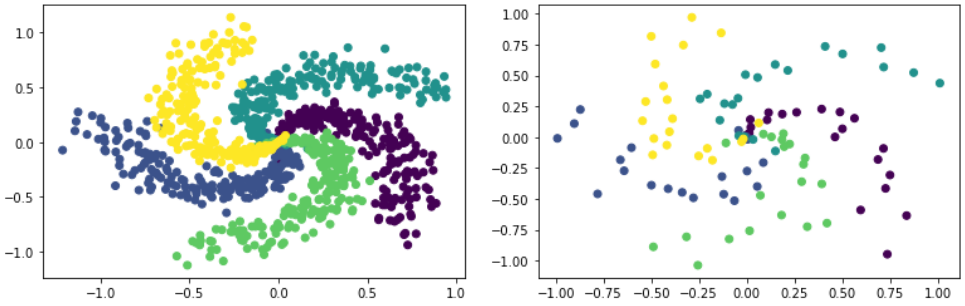
Clases = 3: *Precisión de testeo 95%*



Clases = 4: *Precisión de testeo 97%*



Clases = 5: *Precisión de testeo 92%*

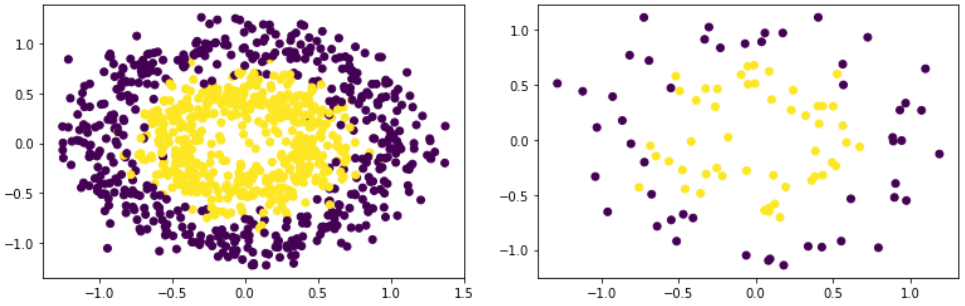


Como conclusión podemos observar que a medida que aumentamos la cantidad de clases la red pierde precisión. Como vemos en el caso de la clase 5, que contiene 5 clases, la precisión es del 92%. Esto puede deberse a que al tener mayor cantidad de clases y, por ende, estar cercanas unas de otras, los hiperparametros de la red neuronal no son los suficientes para lograr una mejor clasificación, esto podría mejorar si aumentamos el número de neuronas de la capa oculta o aumentando el número de capas.

Otra observación que podemos hacer a esta distribución es que después de correr el código un par de veces notamos que no había una disminución lineal de la precisión en el caso de la clase 3 posee menos precisión que para la clase 4. Pero en otras ocasiones se respetaba la linealidad hasta cuatro clases y cuando calculamos para cinco clases, esta última poseía mejor precisión de testeo que para cuatro clases. Establecimos que esto se puede deber a la aleatoriedad de los datos generados..

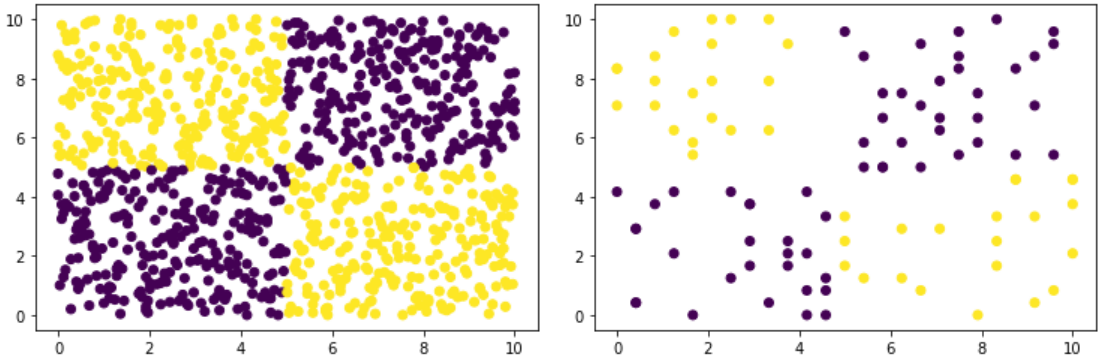
Como segunda opción la distribución de la dispersión de datos está establecida en forma de dos circunferencias concéntricas con un cierto nivel de ruido (dos clases).

Clases = 2: *Precisión de validación 95,66%* | *Precisión de testeo 96%*



Luego, como tercera opción, configuramos una distribución de la dispersión de datos en cuadrantes. En este caso también para dos clases en la cual observamos una disminución abrupta de la precisión.

Clases = 2: *Precisión de validación 68%* | *Precisión de testeo 62%*



Como conclusión final de estos puntos realizados, constatamos dados los resultados obtenidos que el algoritmo funciona mejor para la distribución de datos en forma espiralada que en las demás distribuciones de datos propuestas en la opción 2 y 3, ya que presentan menor precisión que la primera.

Como aporte general podemos decir que a fines prácticos tal vez para la visualización y posterior comprensión de los datos en algunos casos puede ser conveniente otra forma de representación, generalmente, esto ocurre al tener más de 3 características en los datos (dimensiones).

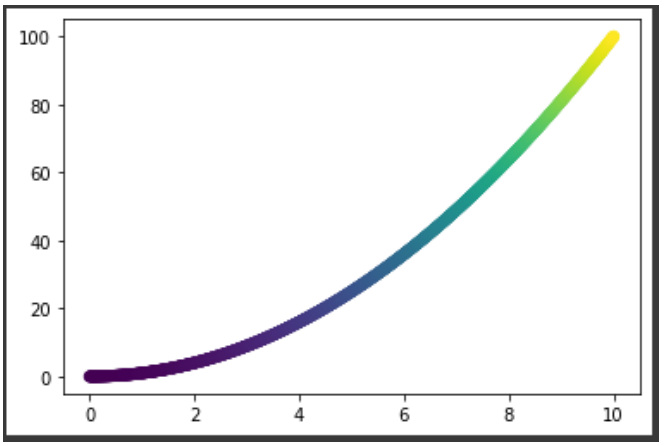
Otra observación para agregar, es que el algoritmo no se comportó de la misma manera durante varias compilaciones para el mismo conjunto de hiperparametros, es una realidad que tal vez la configuración más eficiente y precisa para una ejecución nos dé una precisión más baja en una ejecución siguiente debido a la aleatoriedad de los datos.

Teniendo en cuenta además el resultado para la última distribución propuesta, esto nos está diciendo que para esta red quizás debemos cambiar parámetros internos, esto pueden verse expresado en cambios de un aumento de neuronas, aumento en la cantidad de capas o el ajuste del learning rate.
Es por ello que los datos expresados anteriormente son a fines ilustrativos para mostrar el comportamiento de la red con las distintas configuraciones.

Ejercicio 5:

En este ejercicio procedemos realizando el cambio, en primer lugar, de la función de costo “Softmax” por “MSE” (Error Cuadrático Medio). Luego también hacemos el cambio de nuestra función de activación “ReLU” por la función “Sigmoide”.

Para el generador de datos creamos 2 conjuntos de valores (X e Y) de entrada a la red entre 0 y 10, y la salida de la función es igual al producto de dichos valores, componente a componente.



Como configuración de la red, dejamos 2 neuronas en la capa de entrada, 100 en la capa oculta y 1 en la capa de salida.

Una vez realizadas las implementaciones correspondientes, procedimos a ejecutar el algoritmo, el cual nos presentó un error en las dimensiones de las matrices en la etapa del "Backpropagation". Al momento de la entrega del presente informe dicho error no ha podido ser solucionado, aunque intentaremos de tener el problema resuelto para el momento del coloquio correspondiente. Por otro lado, consideramos la opción de realizar la implementación mediante las librerías "Keras" de Tensor Flow, las cuales nos simplificarían todo el trabajo matemático.

Ejercicio 6:

Parámetros utilizados:
EPOCHS=5000
Validación cada 500 EPOCHS
Cantidad ejemplos training=1000
Cantidad ejemplos validación=300
Cantidad ejemplos test=300

En este ejercicio realizamos la prueba de la red neuronal para distintos hiperparametros a fin de buscar la combinación más óptima:

Utilizando la función de activación "ReLU":

En primer lugar, fijamos un número de neuronas en la capa oculta en 100 y fuimos variando el learning rate:

Ejemplo 1:

LR	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Accuracy	93.0	93.66	94.66	92.66	94.66	94.0	94

Ejemplo 2:

LR	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Accuracy	95.33	95.33	95.0	95.0	95.33	95.33	95.33

Ejemplo 3:

LR	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Accuracy	91.33	89.66	88.66	63.33	63.66	63.0	63.33

En segundo lugar, fijamos el learning rate en 1.0 y fuimos variando el número de neuronas de la capa oculta:

Ejemplo 1:

Neuronas	50	100	150	200	250	300	400
Accuracy	91.0	93.0	94.0	93.66	93.33	93.66	94.66

Ejemplo 2:

Neuronas	50	100	150	200	250	300	400
Accuracy	95.0	95.33	95.33	95.33	95.33	95.33	95.33

Ejemplo 3:

Neuronas	50	100	150	200	250	300	400
Accuracy	63.33	88.33	63.33	81.66	63.66	50.33	63.33

Utilizando la función de activación "Sigmoide":

En primer lugar, fijamos un número de neuronas en la capa oculta en 100 y fuimos variando el learning rate:

Ejemplo 1:

LR	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Accuracy	20.0	20.0	24.66	15.33	23.33	20.0	20.0

Ejemplo 2:

LR	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Accuracy	51.66	50.0	59.66	54.33	56.66	50.0	50.0

Ejemplo 3:

LR	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Accuracy	52.66	47.33	52.66	47.66	52.66	52.66	47.33

En segundo lugar, fijamos el learning rate en 1.0 y fuimos variando el número de neuronas de la capa oculta:

Ejemplo 1:

Neuronas	50	100	150	200	250	300	400
Accuracy	31.33	20.0	20.0	20.0	20.0	20.0	20.0

Ejemplo 2:

Neuronas	50	100	150	200	250	300	400
Accuracy	55.66	50.0	55.66	50.0	50.0	53.0	50.0

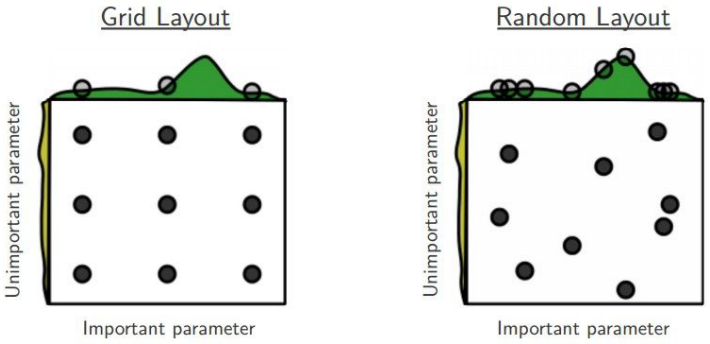
Ejemplo 3:

Neuronas	50	100	150	200	250	300	400
Accuracy	52.66	47.33	52.0	52.66	47.33	47.33	47.33

Analizando los resultados obtenidos vemos que, en rasgos generales, el uso de la función de activación “ReLU” nos da un mejor desempeño para estos conjuntos de datos respecto de la función Sigmoide.

Por otro lado, analizando la variación de los parámetros realizadas con la función “ReLU” podemos ver que para cada uno de los ejemplos o conjuntos de datos los parámetros óptimos no son los mismos para todos los casos, esto se debe a que para cada tarea o desarrollo particular se debe realizar la optimización de hiperparametros que generen el mejor desempeño. Esto puede realizarse de manera manual “a prueba y error” o con ayuda de algún algoritmo de optimización de parámetros o búsqueda local, como por ejemplo: “Temple Simulado”.

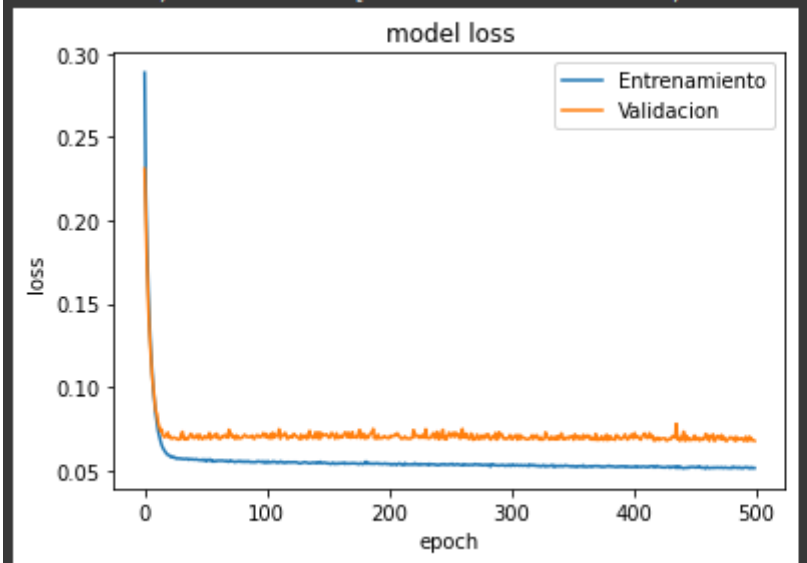
La idea de utilizar algoritmos de búsqueda es la de acelerar el proceso y hacerlo de manera más rápida, llegando a un buen conjunto de parámetros en menor tiempo.



A modo de verificación del código desarrollado, realizamos la implementación de la misma red neuronal haciendo uso de las librerías de “Keras” de Tensor Flow, lo cual nos permite una rápida implementación y posteriormente una rápida variación de características de la red. Obtuvimos los siguientes resultados, los cuales se correlacionan con los obtenidos para nuestro algoritmo de manera aproximada en cuanto al valor de la precisión:

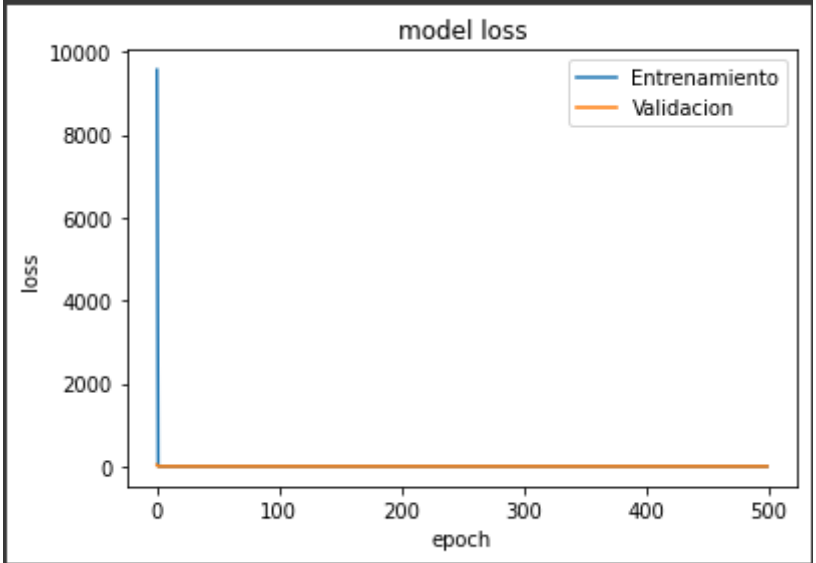
LR = 0.6 | N = 400 | epochs = 500

```
test loss, test acc: [0.07020746916532516, 0.9333333373069763]
```




```
Accuracy ReLU:  0.9333333373069763
```

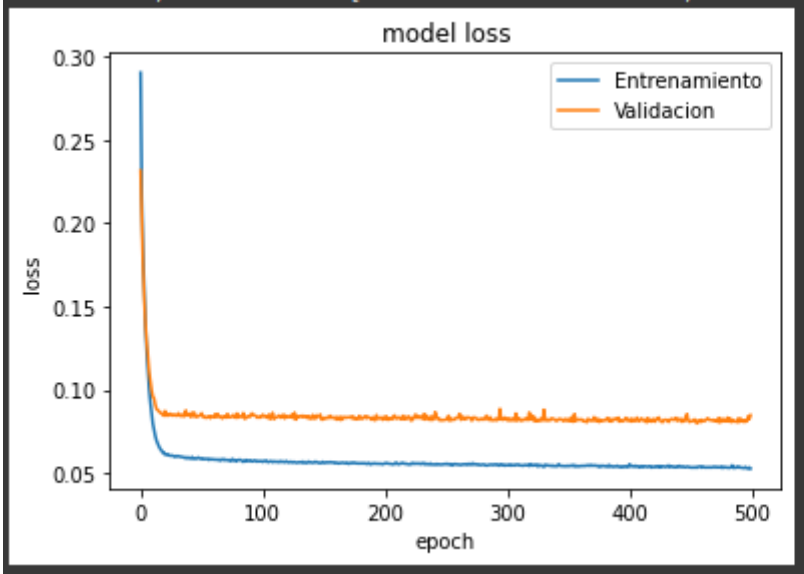
```
test loss, test acc: [0.25000789761543274, 0.5]
```




```
Accuracy Sigmoide:  0.5
```

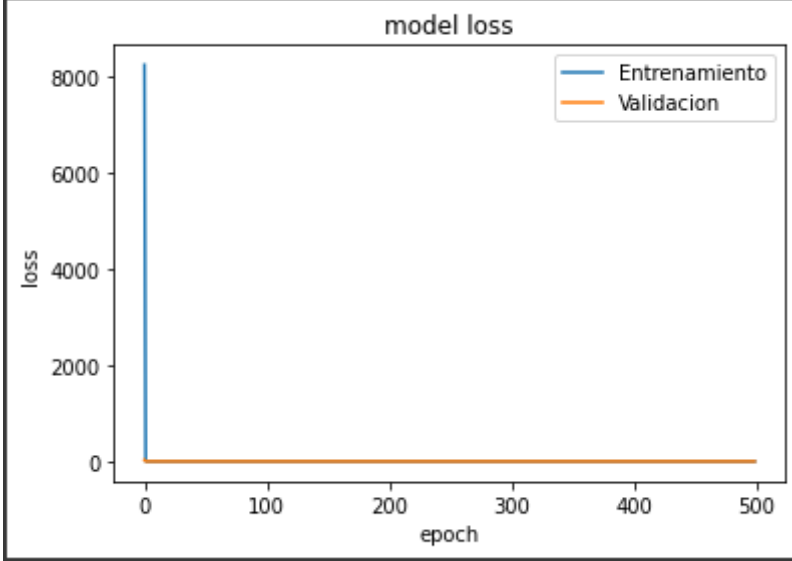
LR = 0.75 | N = 400 | epochs = 500

```
test loss, test acc: [0.062361858785152435, 0.9399999976158142]
```




```
Accuracy ReLU:  0.9399999976158142
```

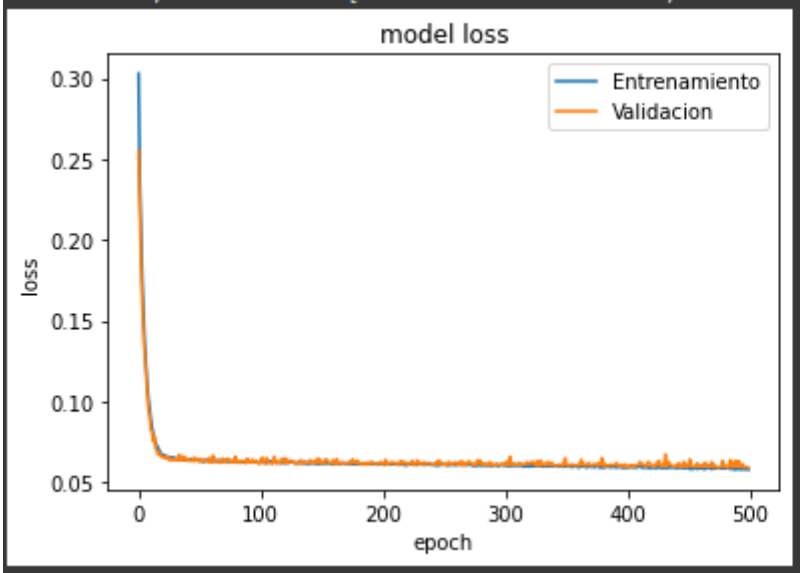
```
test loss, test acc: [0.25022637844085693, 0.5]
```




```
Accuracy Sigmoide:  0.5
```

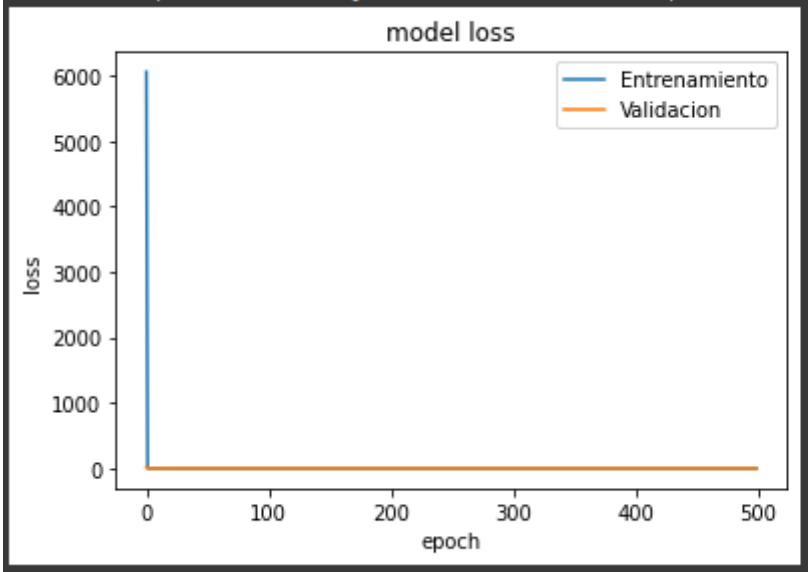
LR = 1.0 | N = 400 | epochs = 500

```
test loss, test acc: [0.06479250639677048, 0.949999988079071]
```




```
Accuracy ReLU:  0.949999988079071
```

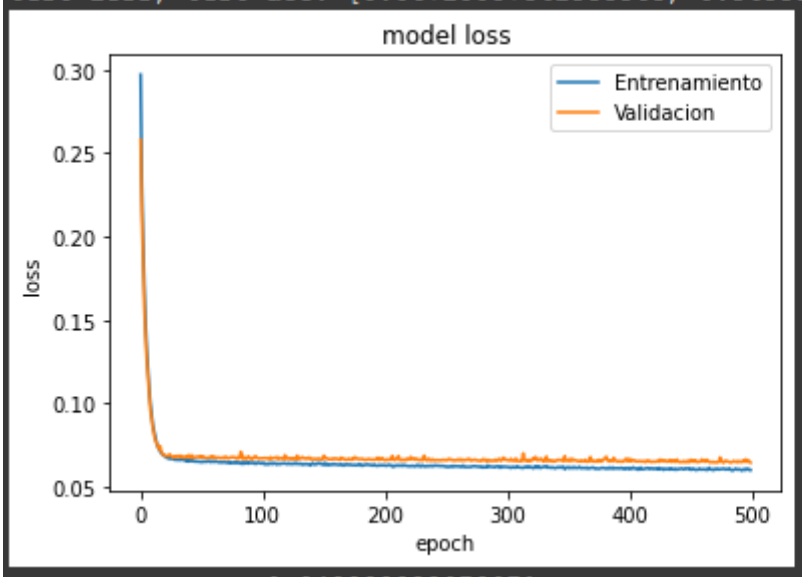
```
test loss, test acc: [0.24988135695457458, 0.5733333230018616]
```




```
Accuracy Sigmoide:  0.5733333230018616
```

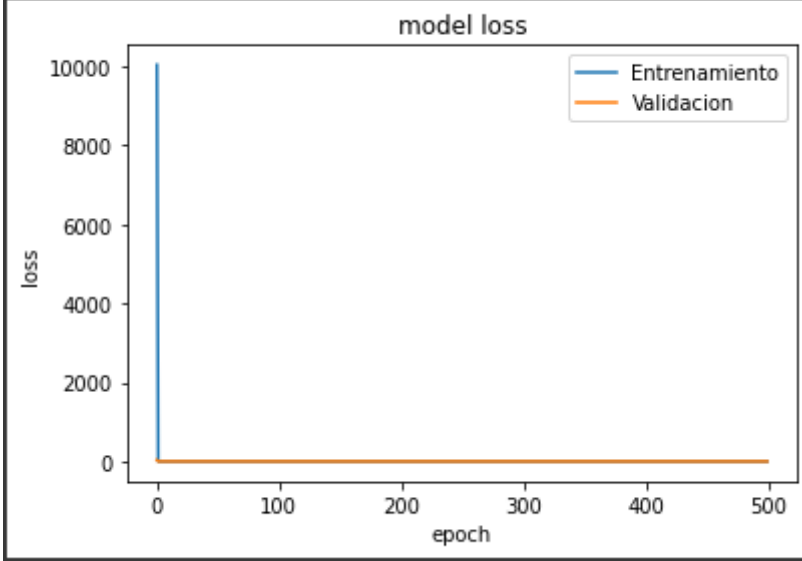
LR = 1.25 | N = 400 | epochs = 500

```
test loss, test acc: [0.06725037842988968, 0.949999988079071]
```




```
Accuracy ReLU:  0.949999988079071
```

```
test loss, test acc: [0.2503798007965088, 0.5]
```




```
Accuracy Sigmoide:  0.5
```