Inteligencia Artificial II

U3: Machine Learning

Grupo N°2

Alumno:	**Legajo:**
Cantú, Maximiliano	11294
Costarelli, Agustín	10966
Lage Tejo, Joaquín	11495

2 0 2 0

Trabajo Práctico Nº 3
Machine Learning

Se dividieron las actividades en dos problemáticas: Clasificación y Regresión. Cada ejercicio aborda distintos ítems a resolver en el trabajo práctico, tanto obligatorios como opcionales.

Ejercicio de Clasificación (puntos 2, 3 y 4)

Se tomó como base el código fuente desarrollado por la cátedra. Este ejercicio, así como todo lo implementado, se encuentre en un solo archivo llamado tp3_class.py

Se unificaron todos los parámetros en la función "comenzar()", para facilitar la ejecución y modificación del mismo.

Precisión de Clasificación

Además de calcular la función de pérdida, en el caso de Clasificación por Softmax, se calcula la precisión del programa. Para ello se usa la función "validar", que a su vez hace uso de "clasificar", y que compara (resta miembro a miembro) el vector resultado (y) de la ejecución hacia adelante con el vector de resultados reales o "target" (t). Luego, cuenta la cantidad de aciertos (cantidad de 0 en el vector "diferencia") y los divide por la cantidad de ejemplos obtenemos la precisión.

Parada temprana

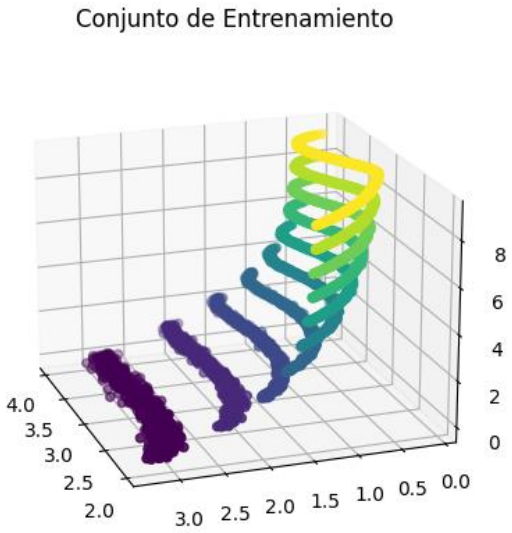
Durante el proceso de Training, cada cierto número "n" de EPOCHS, se realiza una validación (se ejecuta el programa con un conjunto de entradas diferente al usado en Training) y se obtiene la precisión de validación. Comparamos estos valores a medida que se actualizan, con el objetivo de evitar el Overfitting. Se permite una tolerancia dentro de la cual se da un aviso, y luego el programa corta la ejecución en caso de superarse un valor límite.

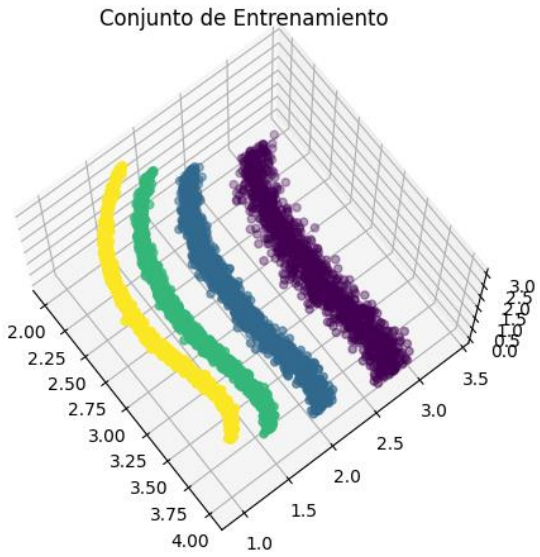
Nuevo generador de datos

Para evaluar el funcionamiento del programa con otros datos, se propuso un generador que evalúa el resultado de la función " $t=f(x,y)=(x-3)^5-(x-3)^{1.5}+y^2$ ". Los valores de "t" son las curvas de nivel de la función real, con esto adaptamos el problema a un caso de Clasificación.

Además, se agrega un factor de aleatoriedad que se incrementa a medida que "t" se acerca a 0, es decir, para curvas de menor nivel. El conjunto de Test y Validación poseen aleatoriedades levemente mayores para probar el programa en condiciones "peores". Las

imágenes corresponden a 10 y 4 clases, del conjunto de entrenamiento, vistas de costado y de arriba, respectivamente.



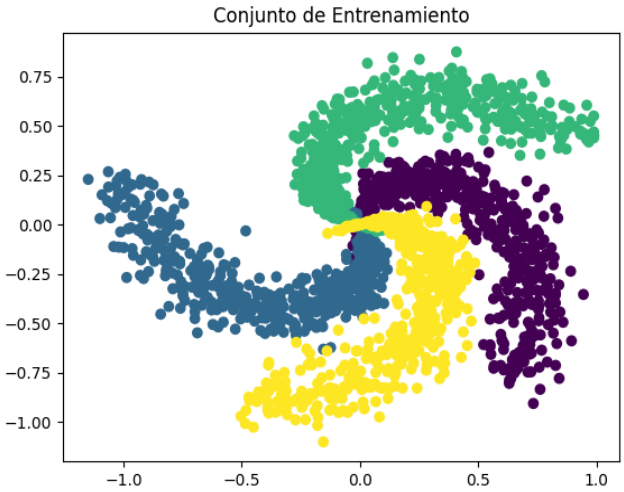


Dicha aleatoriedad es un parámetro de entrada, pero se maneja en un rango distinto al de los datos originales, por lo que hay que prestar atención a ese detalle al momento de cambiar entre los datos (datos_orig).

Variación de parámetros

Para ambos conjuntos de datos se generaron 2000 ejemplos, divididos en 4 clases, con un total de 5000 EPOCHS, y realizando una validación cada 500 EPOCHS. Además, la capa oculta posee 100 neuronas en ambos casos.

El programa responde bien (precisión en Test alrededor de 85%) a los datos originales con los siguientes parámetros:



- Aleatoriedad de Training=0.1
- Aleatoriedad de Validación=0.15
- Aleatoriedad de Test=0.2
- Learning Rate=0.3

No obstante, en el caso del polinomio de orden 5, es preferible atacar el problema con la siguiente combinación, para obtener una precisión alrededor del 77%:

- Aleatoriedad de Training=0.03
- Aleatoriedad de Validación=0.045
- Aleatoriedad de Test=0.06

- Learning Rate=0.1

Si solo cambiamos el Learning Rate a 0.3, obtenemos un resultado aproximado de 81%. Vemos que, triplicando este factor, no se observan cambios muy significativos.

Además, el polinomio es mucho más sensible a las variaciones de aleatoriedad, empeorando mucho la respuesta con pequeños incrementos. En última instancia, sin aleatoriedad en la creación de los puntos, con un Learning Rate de 0,1 obtenemos una precisión de Test de 99%. Este caso no es muy representativo porque el conjunto de Test es casi coincidente con el Training, pero sirve para mostrar la incidencia de la aleatoriedad en el comportamiento del programa.

Podemos concluir que, con el nuevo dataset, la respuesta del programa es algo peor, en gran parte porque se está intentando aproximar un polinomio de orden 5 con una sola capa oculta, y con valores de entrada con un cierto error (la aleatoriedad mencionada). Pero sin este error, la aproximación es realmente buena.

Ejercicio de Regresión (puntos 5, 6, 8 y 9)

El ejercicio de regresión se realizó en base a datos reales, y se implementaron (no todos a la vez) controles por Overfitting o Correlación, graficación de Loss (EPOCHS), barrido de parámetros, una segunda capa oculta.

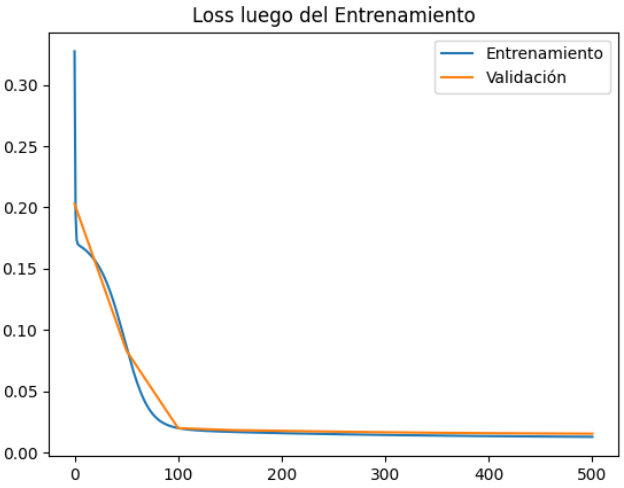
Existen 4 códigos para este ejercicio.: tp3_central.py trabaja con una sola capa oculta y sin barrido de parámetros, tp3_central_2ocultas.py utiliza una segunda capa oculta, tp3_central_barrido.py reimplementa el primer código sumando una optimización de hiperparámetros.

Datos utilizados

El dataset contiene 9500 puntos de datos recopilados de una central eléctrica de ciclo combinado durante 6 años (2006-2011). Las características consisten en las variables ambientales promedio por hora: Temperatura (T), Vacío de escape (V), Presión ambiente (AP) y Humedad relativa (RH), para predecir la producción neta de energía eléctrica (EP) por hora de la planta. Se utiliza un archivo de 9500 aislado, pero el original incorpora 5 hojas de 9565 muestras.

Los datos son, previo a comenzar el proceso, centrados y normalizados, quedando en el rango [-1,1].

Se obtuvieron del Machine Learning Repository de la Universidad de California en Irvine[1].



Función de activación

El programa puede elegir entre dos funciones de activación para la/s capa/s oculta/s: Sigmoide o ReLU. Esto se consigue solo "comentando" y "descomentando" unas pocas líneas en la sección de "train" y en la de "ejec_forw".

Los resultados, manteniendo los parámetros invariables, muestran un mejor rendimiento utilizando ReLU, quizás debido a una posible saturación de las entradas en los extremos de la función Sigmoide.

Función de pérdida

En todos los casos se utilizó la función MSE (error cuadrático medio) para medir la precisión de cada EPOCH.

Parada temprana y error de correlación

El programa controla si la pérdida de cada validación crece o decrece (Overfitting), y si se aleja significativamente de los valores de training (Correlación).

Con distintas tolerancias, se emite un AVISO si detecta que puede haber un error, o bien un ERROR si se supera el límite máximo, deteniendo la ejecución del código.

Gráfica y Underfitting

Se grafica la progresión de la pérdida tanto en Training como en Validación, con la intención de compararlos luego de todo el entrenamiento, y así corroborar un posible Underfitting. Hemos encontrado que 500 EPOCHS es un buen número para evitar este problema.

Barrido de parámetros

Este proceso se puede ver en el código tp3_central_barrido.py.

Al código correspondiente a los datos de la central para una capa oculta se le incorporo la función barrido. La cual está compuesta de dos bucles for que se encargan de ir

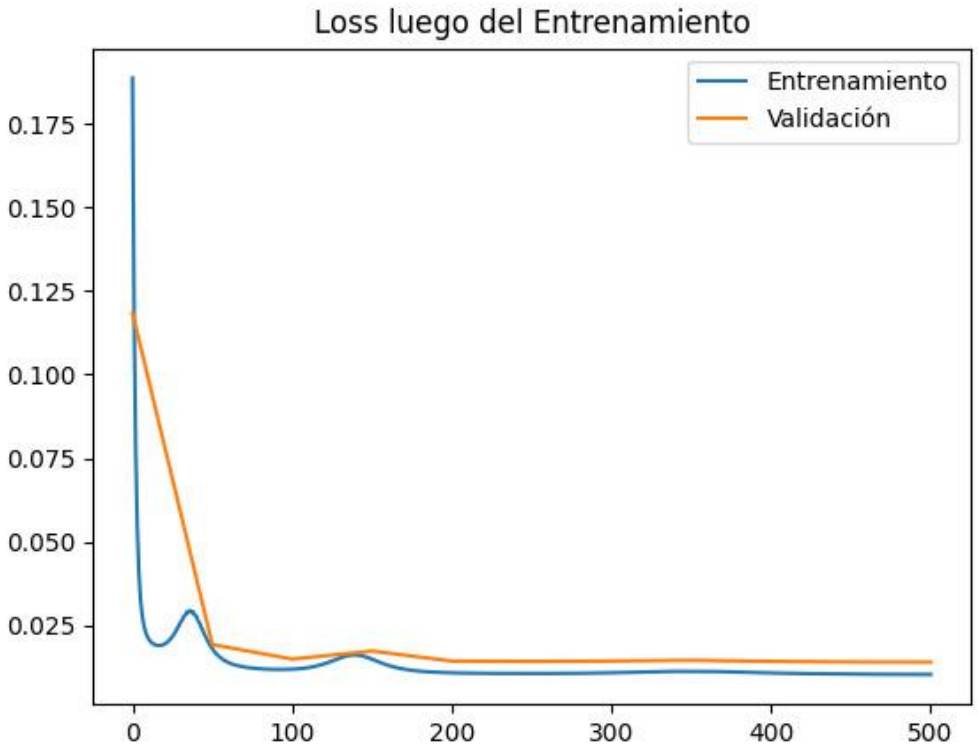
1 https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant

modificando los valores de learning rate y cantidad de neuronas en la capa oculta. Dentro de cada for se ejecutó el entrenamiento de la red para 500epoch. Se guardó el menor valor de loss y se registraron los valores de learning rate y número de neuronas en capa oculta que lo producían. Luego se realizó un segundo barrido que partiendo de los hiperparámetros que daban el menor loss, este segundo barrido se hacía en un intervalo reducido en torno a estos puntos óptimos y con un menor paso buscando con mayor precisión valores que dieran un menor loss. Al final del segundo barrido y con los parámetros óptimos se grafica el valor de loss de training y validación de la red neuronal con estos valores.

Este barrido se realizó tanto para la función sigmoide y para la función ReLU. Estos fueron los resultados. Dado que el tiempo de ejecución del barrido era superior a 4 horas no se pudieron realizar muchas comparaciones buscando variación del paso e intervalo de barrido de los hiperparámetros.

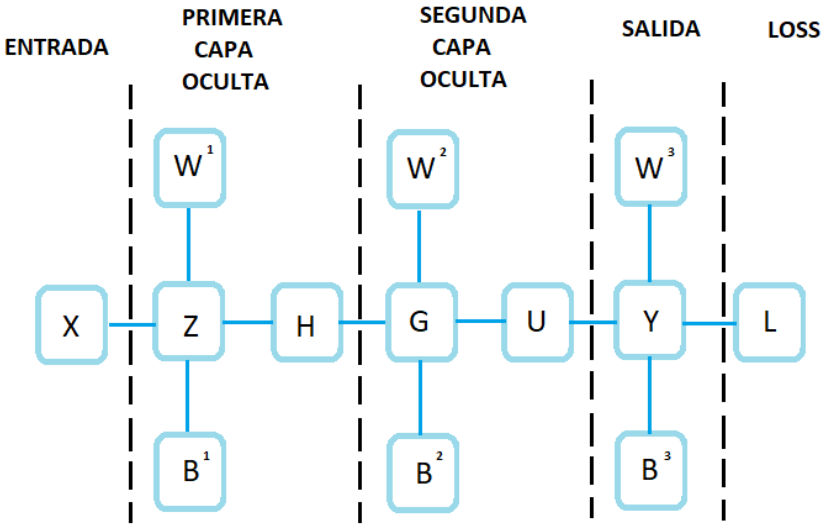
Para el caso de la función sigmoide se obtuvieron como parámetros óptimos un learning rate de 0.2019, 11 neuronas de capa oculta y un valor de loss de 0.03318. Su gráfica es la siguiente.

Para el caso de la función ReLU se obtuvieron como parámetros óptimos un learning rate 0.4426, 91 neuronas de capa oculta y un valor de loss de 0.009836. Su gráfica es la siguiente



Segunda capa oculta

La segunda capa se implementa en el programa tp3_central_2ocultas.py. Estas son las derivadas finales que se dedujeron analíticamente, con ambas funciones de activación (ReLU y Sigmoide):



$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial W^3} \qquad \frac{\partial L}{\partial B^3} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial B^3}$$

$$\frac{\partial L}{\partial Y} = -2.\frac{T-Y}{m} \qquad \frac{\partial Y}{\partial W^3} = U \qquad \frac{\partial Y}{\partial B^3} = 1$$

$$\frac{\partial L}{\partial W^3} = U^T.\left(-2.\frac{T-Y}{m}\right) \qquad \frac{\partial L}{\partial B^3} = -2.\frac{T-Y}{m}$$

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial U} \cdot \frac{\partial U}{\partial G} \cdot \frac{\partial G}{\partial W^2} \qquad \frac{\partial L}{\partial B^2} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial U} \cdot \frac{\partial U}{\partial G} \cdot \frac{\partial G}{\partial B^2}$$

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial G} \cdot \frac{\partial G}{\partial W^2} \qquad \frac{\partial L}{\partial B^2} = \frac{\partial L}{\partial G} \cdot \frac{\partial G}{\partial B^2}$$

$$\frac{\partial L}{\partial G} = \frac{\partial L}{\partial U} \cdot \frac{\partial U}{\partial G} \qquad \frac{\partial G}{\partial W^2} = H^T \qquad \frac{\partial G}{\partial B^2} = 1$$

$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial U} = \frac{\partial L}{\partial Y} \cdot (W^3)^T \qquad \frac{\partial U}{\partial G} = \sigma(x).\left(1-\sigma(x)\right)$$

$$\text{ó} \qquad 1(z>0)$$

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial U} \cdot \frac{\partial U}{\partial G} \cdot \frac{\partial G}{\partial H} \cdot \frac{\partial H}{\partial Z} \cdot \frac{\partial Z}{\partial W^1}$$

$$\frac{\partial L}{\partial B^1} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial U} \cdot \frac{\partial U}{\partial G} \cdot \frac{\partial G}{\partial H} \cdot \frac{\partial H}{\partial Z} \cdot \frac{\partial Z}{\partial B^1}$$

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial W^1} \qquad \frac{\partial L}{\partial B^1} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial B^1}$$

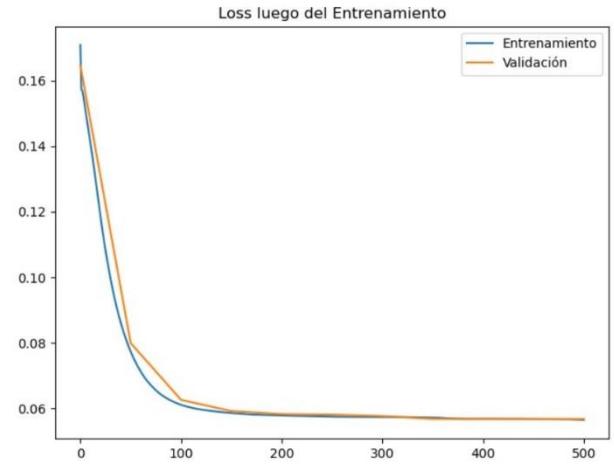
$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial G} \cdot \frac{\partial G}{\partial H} \cdot \frac{\partial H}{\partial Z} \qquad \frac{\partial Z}{\partial W^1} = X^T \qquad \frac{\partial Z}{\partial B^1} = 1$$

$$\frac{\partial G}{\partial H} = (W^2)^T$$

$$\frac{\partial H}{\partial Z} = \sigma(x).\big(1 - \sigma(x)\big) \quad \text{ó} \quad 1(z > 0)$$

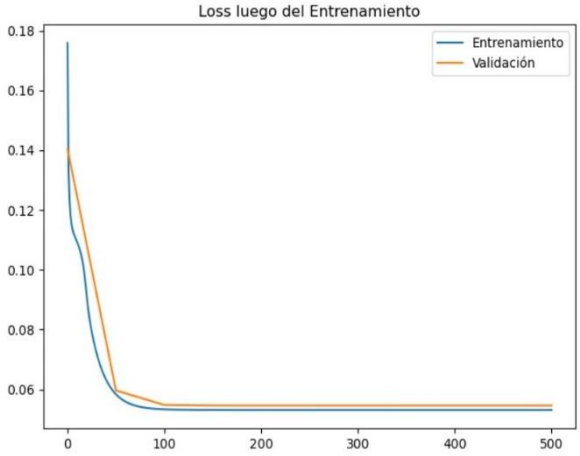
Para lograr la correcta implementación de la segunda capa oculta, se hicieron modificaciones en 3 secciones distintas del código. Primero en “inic_pesos” se generan los nuevos valores de los pesos sinápticos y bias (w3 y b3 respectivamente) de forma aleatoria, al igual que se generaban los de w2, b2, w1 y b1; y dependen de los valores ingresados como número de neuronas de entrada y número de neuronas de cada capa oculta (N_ENTRADA y N_OCULTAS, respectivamente). En segundo lugar, se modificó la sección “ejec_forw” agregando la función “g” y la función “u”, las cuales son la función de entrada para la segunda capa oculta y la función de activación de la misma capa, respectivamente. Como se explicó previamente, es posible modificar las funciones de activación de ambas capas ocultas para trabajar con función Sigmoide o ReLU y así poder comprobar y analizar con cual se obtienen mejores resultados. Por último, en la sección “train” se agregaron las ecuaciones derivadas ya mencionadas, para ajustar los valores de los parámetros de cada capa (pesos sinápticos y bias) de acuerdo a la función de perdida y a la función de activación utilizada, y así lograr el aprendizaje deseado mediante Backpropagation.

Para comprobar el correcto funcionamiento del programa con 2 capas ocultas, se ejecutó el programa reiteradas veces modificando el número de neuronas en las capas ocultas y el learning rate, y utilizando la función Sigmoide:



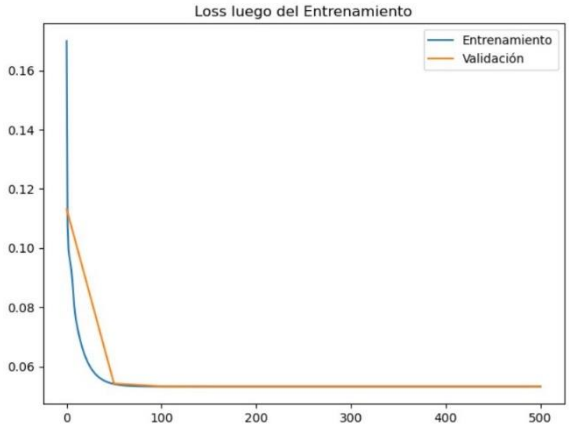
- Neuronas en capas ocultas = 1
- Learning rate=0.05

```
Loss en Test:  [0.05526255]
```



- Neuronas en capas ocultas = 5
- Learning rate=0.05

```
Loss en Test:  [0.05227859]
```



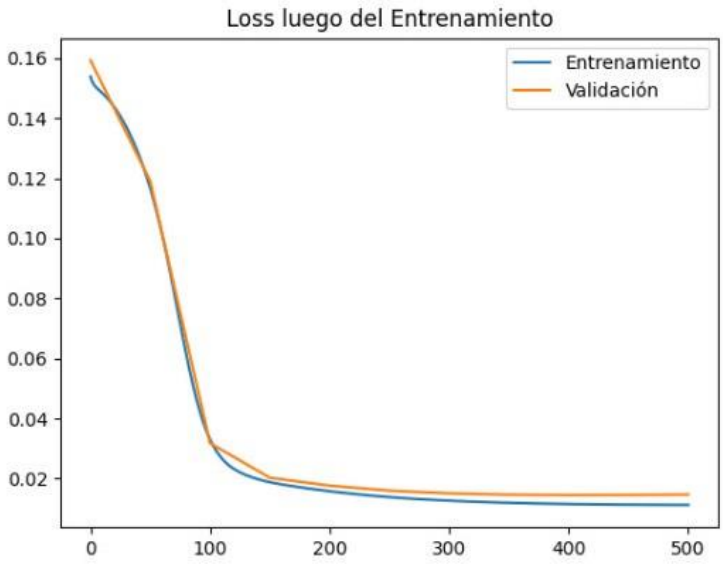
- Neuronas en capas ocultas=10
- Learning rate=0.05

```
Loss en Test:  [0.05096499]
```

Las gráficas solo son los resultados obtenidos con algunos de los valores ingresados, ya que el análisis se realizó para muchos más valores, pero sintetizando, podemos observar

que a medida que aumentamos la cantidad de neuronas, se obtuvieron respuestas más rápidas y con menores valores de perdida en test. Cabe destacar que a medida que aumentábamos el número de neuronas de las capas ocultas y ejecutábamos repetidas veces con los mismos valores, disminuía la probabilidad de que la respuesta convergiera. En cambio, con una sola neurona en cada capa oculta, resultaba más probable que obtener respuestas satisfactorias y convergentes, pero también se obtenían divergencias.

Finalmente, al no obtener resultados muy convincentes, con Sigmoide, se procedió a analizar el programa con ReLU. En principios, con valores de neuronas entre 1 y 10, y con un learning rate de 0.05 y 0.1, las gráficas marcaban una gran correlación entre el loss de entrenamiento y el de validación, pero al aumentar el número de neuronas ocultas a 15 y con un learning rate de 0.1, se obtuvieron respuestas mucho más satisfactorias y convergentes que las obtenidas con la función Sigmoide, y un loss de test de 0.0127 aproximadamente.

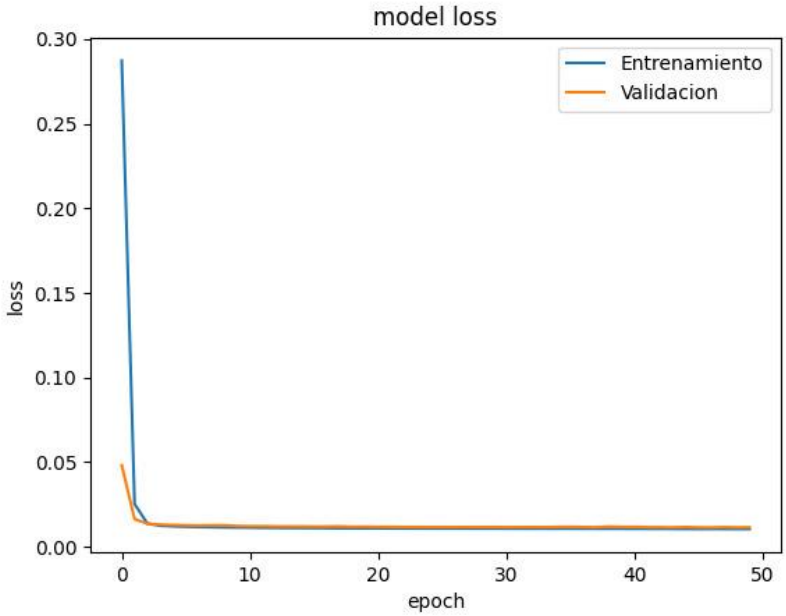


Punto 11

Haciendo uso de la biblioteca de Keras se implementó la misma red neuronal del dataset de la central con una capa oculta. En el código “implementacion_keras.py” se puede apreciar la enorme simplificación del problema y el gran poder que tiene la biblioteca de Keras para machine learning, es muy fácil agregar capas ocultas a nuestra red como también definir funciones de activación y actualizadores de parámetros. Los resultados del loss para la función sigmoide y ReLU se pueden apreciar en las siguientes gráficas.

En ambos casos se utilizó una red neuronal de una sola capa oculta con 8 neuronas y 50 epoch. Para comparar con lo obtenido en los códigos anteriores.

Para la función de activación ReLU el resultado de loss fue 0,0107



Para la función de activación sigmoide el resultado de loss fue 0,0106