Trabajo Práctico N° 4: Introducción a Keras

Aprendizaje Profundo y Redes Neuronales Artificiales

18 de octubre de 2020

Alumnos :

Tomás LIENDRO tomas.liendro@ib.edu.ar

Docentes :

Ariel CURIALE
Germán MATO
Luis MOYANO
Lucca DELLAZOPPA

San Carlos de Bariloche, Argentina

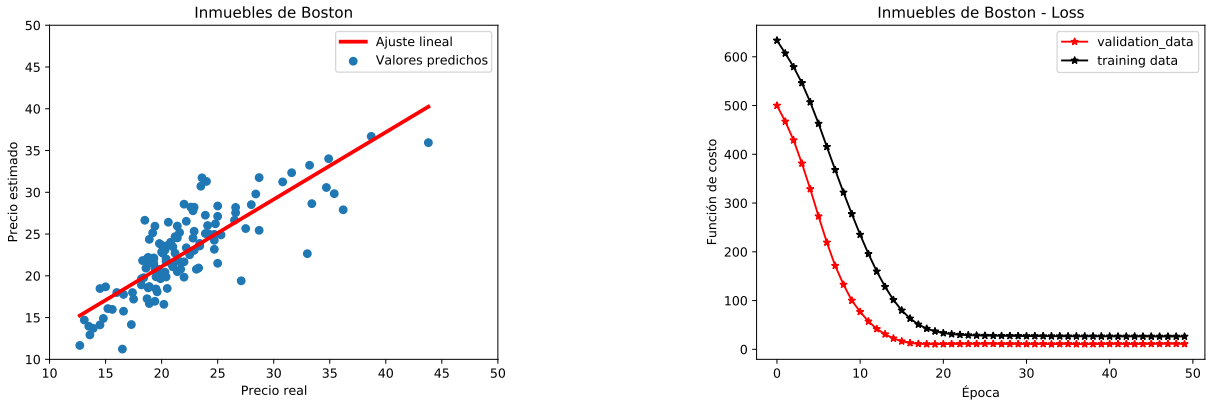
Índice

1. Ejercicio 1

En el primer ejercicio se busca entrenar un modelo para poder predecir el precio de los inmuebles en Boston por medio de una regresión lineal. Para este ejercicio utilizamos la librería *sklearn* para importar los datos y la librería *Keras* para realizar el entrenamiento de la red neuronal. En el preprocesamiento de los datos, estos se normalizaron restándoles el valor de la media y se divide por la desviación estándar. Se utilizó el 75 % de los datos para el entrenamiento y el restante para la validación.

La arquitectura de red que se propone implica dos capas densas con activación Lineal. Se elige la activación *Lineal* dado que no satura la salida de la capa y además porque el problema es lineal. La función de costo utilizada es *MSE* dado que no es un problema de clasificación sino de regresión lineal y la *MSE* permite estimar la distancia entre los datos calculados y los reales.

El resultado de entrenar la red anterior se muestra a continuación:



(a) Regresión lineal

(b) Función de Loss

Figura 1: Regresión lineal del precio de inmuebles en Boston con una arquitectura de una sola capa densa con activación Lineal

El coeficiente de correlación R^2 obtenido entre los datos reales y los datos predichos de los valores de los inmuebles es de 0,669.

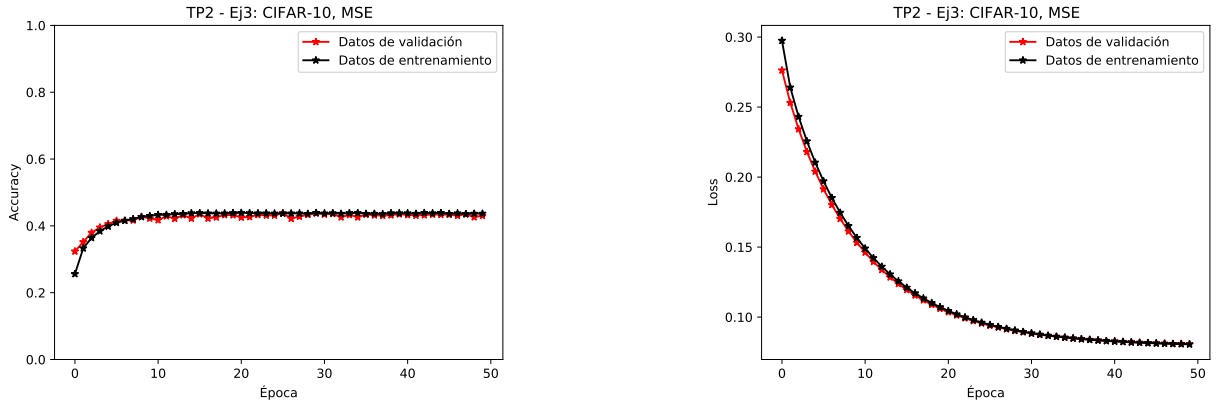
2. Ejercicio 2

En este ejercicio se implementarán los problemas 3, 4 y 6 del Trabajo Práctico 2 utilizando la librería de Keras.

2.1. Ejercicio 3 - TP2

La idea de este ejercicio es implementar una red neuronal para resolver la clasificación de CIFAR-10 utilizando la funcion de costo MSE y una regularizacion L2. Para la clasificacion se utiliza un vecotr de 10 elementos donde idealmente se tendría un 1 en la posicion correspondiente a la clasificacion real de la imagen.

La red entonces queda constituida por dos capas densas con función de activación sigmoide para la primera capa (100 neuronas) y lineal para la segunda (10 neuronas). Utilizando el optimizador SGD con learning rate de 0.1, con un coeficiente de regularizacion de 0.001 y con la funcion de costo MSE se obtienen las siguientes curvas durante el aprendizaje dela red:



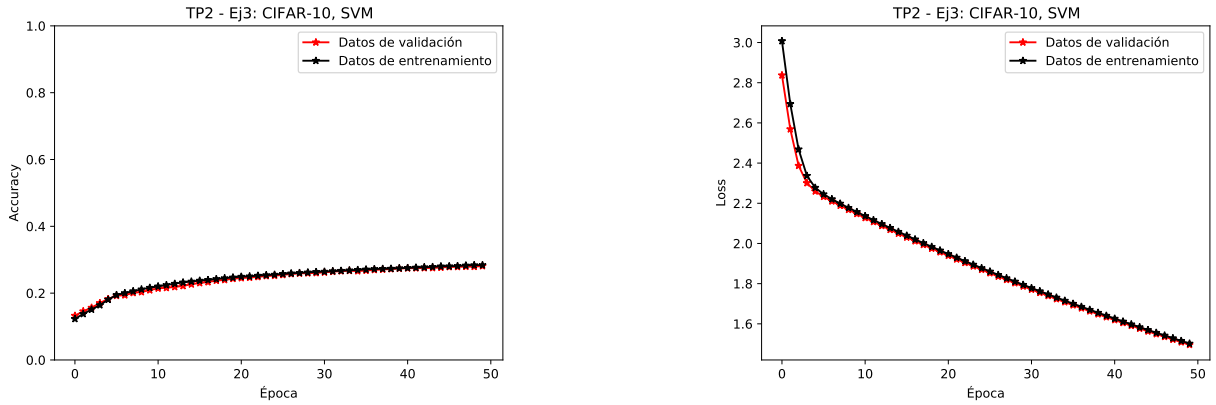
(a) Accuracy

(b) Función de Loss

Figura 2: Implementación del ejercicio 3 del TP2 utilizando la librería Keras con función de costo MSE

de donde se ve que se alcanza un accuracy de hasta 45 %.

También se vieron las curvas obtenidas cuando se utiliza la función de costo Hinge para aplicar el método Support Vector Machine (SVM) con la misma arquitectura que en el caso de la función de costo de MSE. En este caso los resultados fueron:

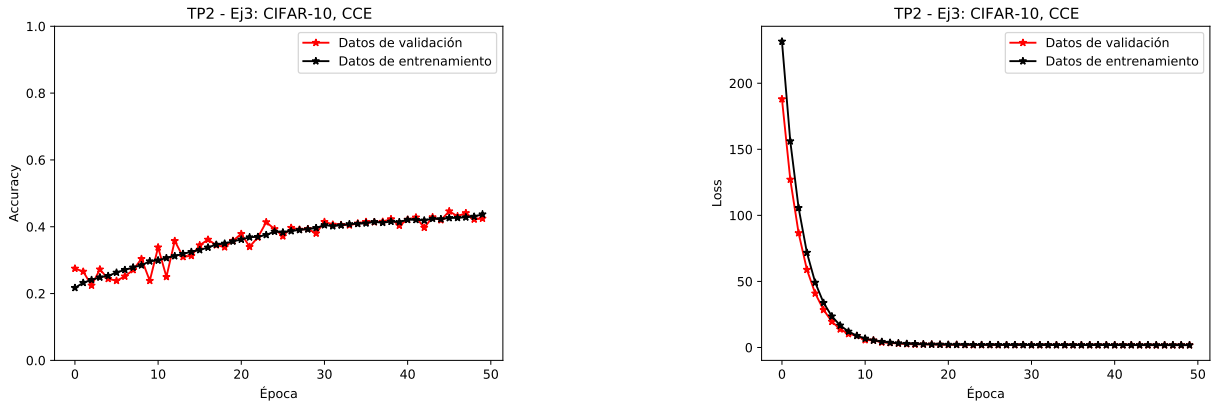


(a) Accuracy

(b) Función de Loss

Figura 3: Implementación del ejercicio 3 del TP2 utilizando la librería Keras con función de costo Hinge

Finalmente, se implementó el método de SoftMax utilizando la función de costo Categorical Cross Entropy y se obtuvieron los siguientes resultados:



(a) Accuracy

(b) Función de Loss

Figura 4: Implementación del ejercicio 3 del TP2 utilizando la librería Keras con función de costo Categorical Cross Entropy de Softmax

Analizando las tres curvas superpuestas, se ve que el método con el que se obtuvo una mayor precisión es el que utiliza la función de costo MSE que alcanza un accuracy de hasta 45 %:

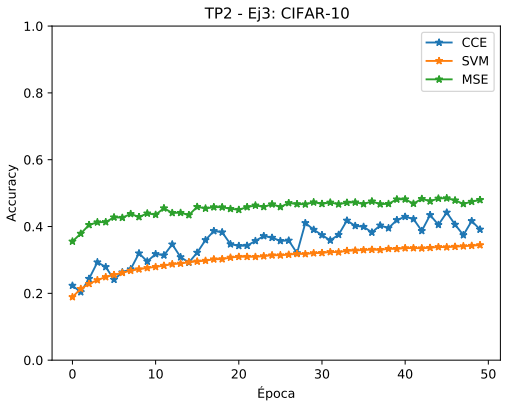
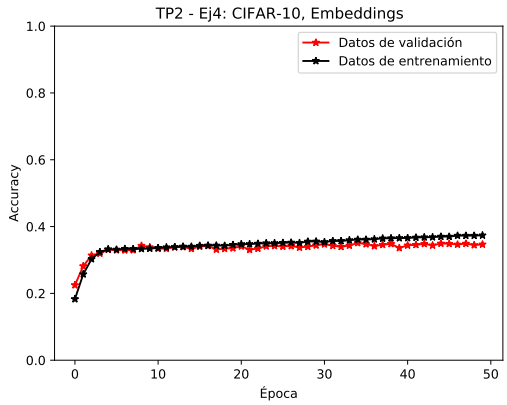


Figura 5: Comparación de las precisiones del clasificador para CIFAR-10 usando las funciones de costo CCE, MSE y Hinge

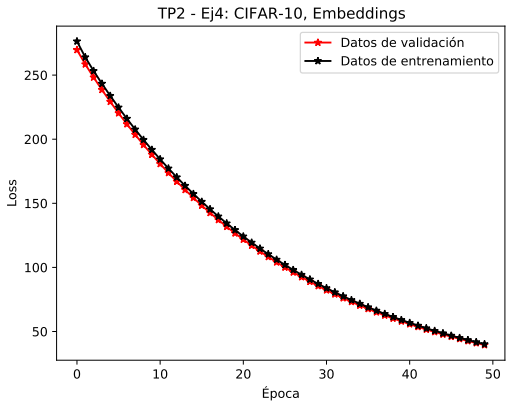
Tal como se ve en la figura anterior, el método que alcanza la mejor precisión es el MSE y el

2.2. Ejercicio 4 - TP2

En el ejercicio 4 del TP2 se solicitaba implementar el mismo clasificador que en el caso anterior utilizando en este caso la función de costo Categorical Cross Entropy de SoftMax. En este caso se utilizó un learning rate de 0.1 y un coeficiente de regularizacion de 0.0001 con regularización de tipo L2. El optimizador usado fue SGD. Los resultados obtenidos fueron:



(a) Accuracy



(b) Función de Loss

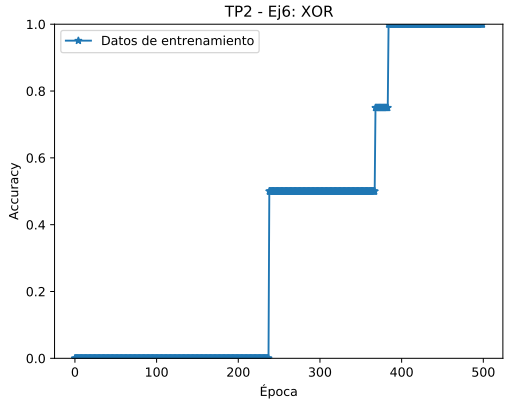
Figura 6: Implementación del ejercicio 4 del TP2 utilizando la librería Keras con función de costo MSE

de donde se ve que se alcanza un accuracy máximo de 40 %.

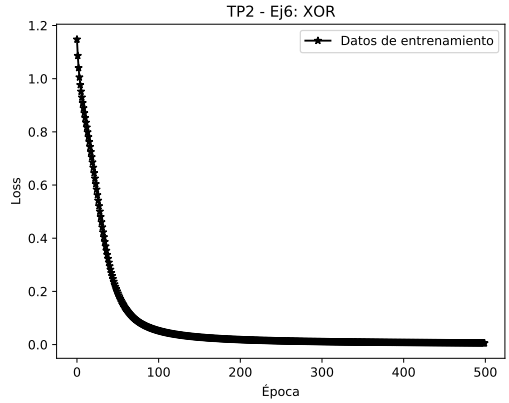
2.3. Ejercicio 6 - TP2

En el ejercicio 6 se solicitaba implementar una red neuronal que aprenda la regla XOR para una entrada de dos bits. La arquitectura utilizada consiste en dos capas densas con 2 y 1 neuronas respectivamente. Las funciones de activación fueron la tangente hiperbólica para ambas capas. El optimizador utilizado es SGD con un learning rate de 0.1 y la función de costo es MSE. Además, se implementó un método para calcular el accuracy mediante un threshold donde se supone que la clasificación es correcta si se supera cierto umbral.

Los resultados del entrenamiento de la red neuronal luego de 500 epocas se muestra a continuación:



(a) Accuracy



(b) Función de Loss

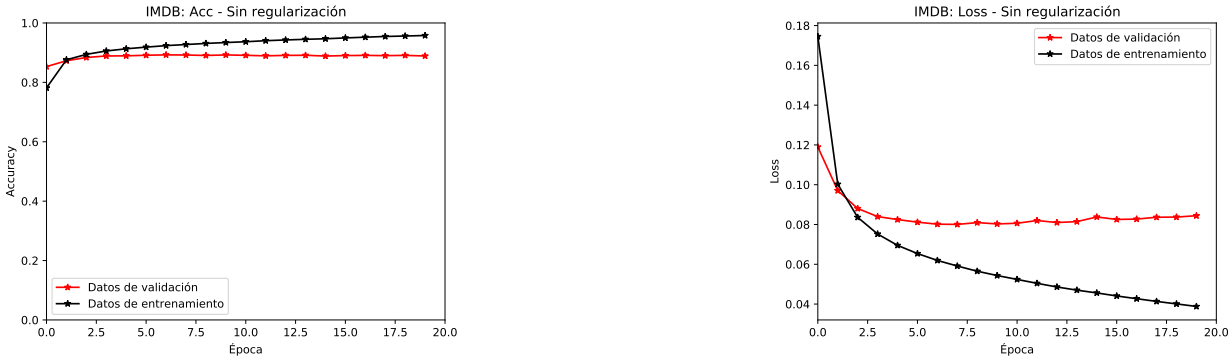
Figura 7: Implementación del ejercicio 6 del TP2 utilizando la librería Keras con función de costo MSE

3. Ejercicio 3

En este ejercicio se busca implementar una red neuronal que permita clasificar las revisiones realizadas en IMDB. Para ello se importa de la librería *Keras* el dataset correspondiente a *imdb* y se importan las 10000 palabras más frecuentes. Además se evalúa el rendimiento de la red propuesta y se ve cómo se el impacto del los regularizadores L2, BN y Dropout en caso de overfitting.

La red propuesta consta de 2 capas densas ocultas con activación *Relu* con 100 y 50 neuronas y una capa densa de salida con activación *Sigmoide* con una neurona de salida. El motivo por el que se usa *Relu* es debido a la no linealidad de los datos del problema, y se utiliza Sigmoide a la salida debido a que la clasificación es binaria y puede tomar los valores 0, para reseñas negativas, o 1 para las positivas. En todos los casos se utilizó Adadelta como optimizador con un learning rate de 0,1.

Inicialmente no se utiliza ningún regularizador, y luego de 20 épocas se obtienen los siguientes resultados:



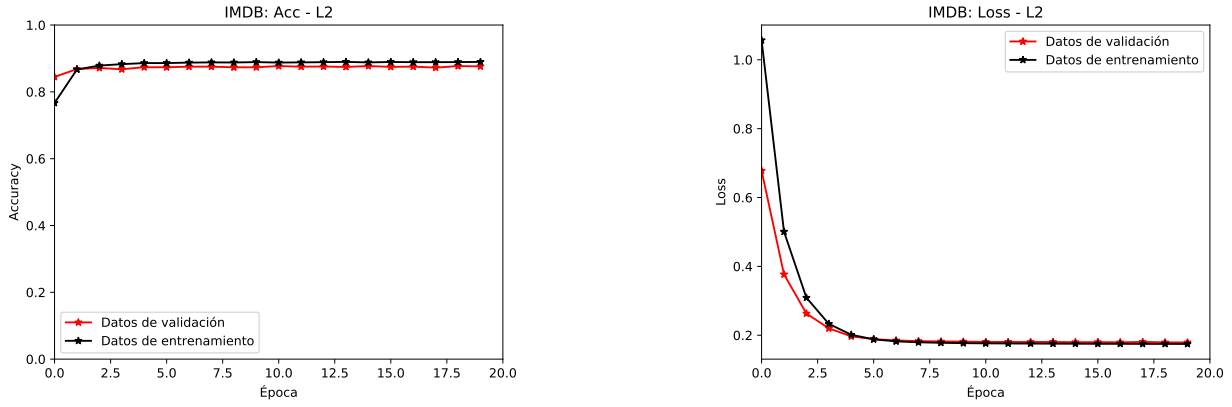
(a) Accuracy

(b) Función de costo

Figura 8: Resultados del clasificador de reseñas de IMDB sin regularizador

De esta figura se ve que ocurre overfitting a partir de la época número 2. Por lo que para corregir esto, se propone utilizar distintos regularizadores.Los regularizadores penalizan las complejidades del modelo mediante distintas estrategias que se mencionarán a continuación.

En primer lugar, para hallar una solución al overfitting se propone utilizar el regularizador L2 con un coeficiente de regularización de $\lambda = 0{,}01$. El resultado obtenido es el siguiente:

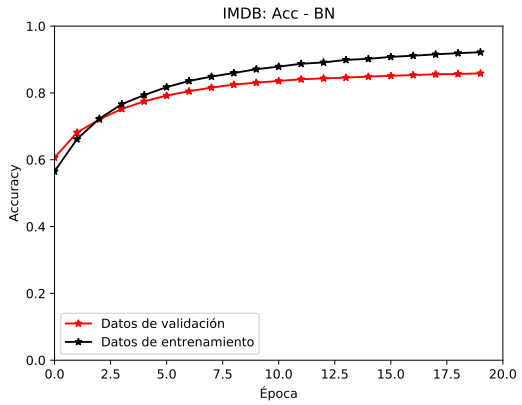


(a) Regresión lineal

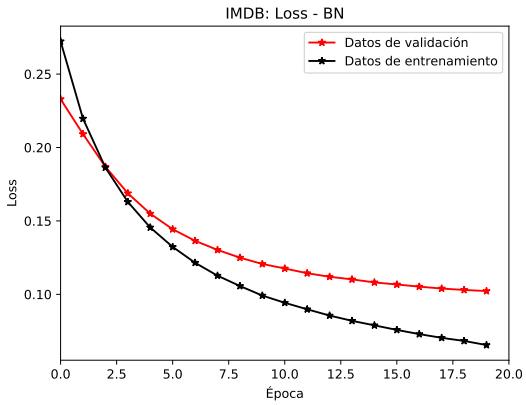
(b) Función de Loss

Figura 9: Resultados del clasificador de reseñas de IMDB con el regularizador L2

Luego, se plantea el uso del regularizador conocido como *Batch Normalization* el cual toma los datos y hace una normalización por batches. La normalización consiste en tomar el conjunto de datos de un batch y restarle el valor medio y dividirlos por la desviación estándar para que ahora tengan media en 0 y desviación estándar igual a uno. Tras aplicar este método de regularización se obtiene:



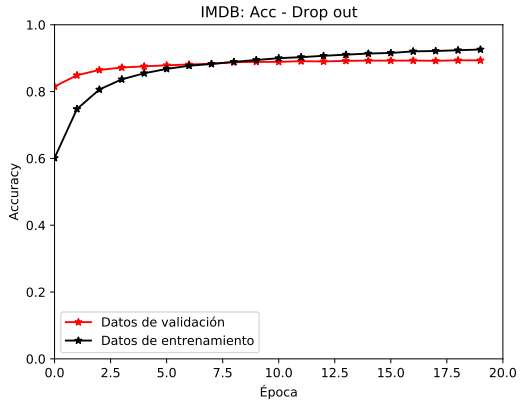
(a) Regresión lineal



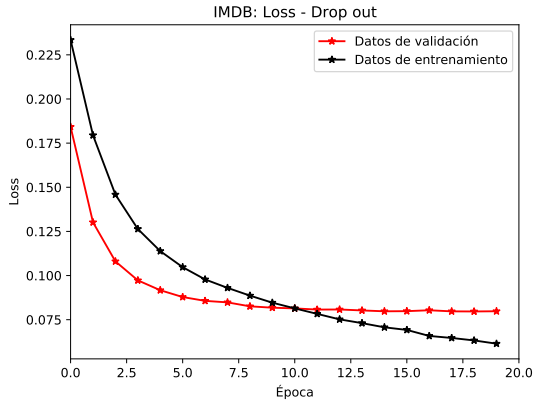
(b) Función de Loss

Figura 10: Regresión lineal del precio de inmuebles en Boston con una arquitectura de una sola capa densa con activación Relu

Finalmente, el regularizador *Dropout* es un método de regularización que en cada iteración saltea una capa aleatoriamente elegida de esta manera las capas a entrenarse ven un conjunto de datos distintos en cada iteración. Luego de aplicar este método, se obtienen los siguientes resultados:



(a) Regresión lineal



(b) Función de Loss

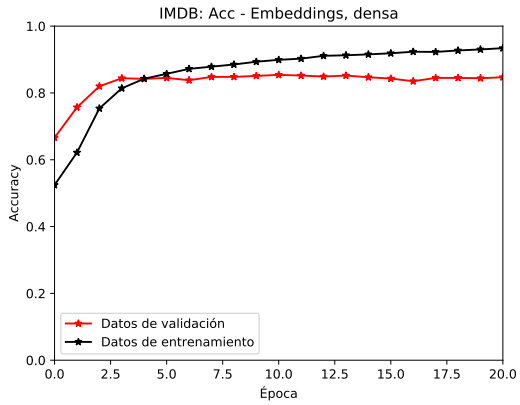
Figura 11: Regresión lineal del precio de inmuebles en Boston con una arquitectura de una sola capa densa con activación Relu

En conclusión, el regularizador L2 resultó ser el que mejor resolvía el problema de overfitting para este problema.

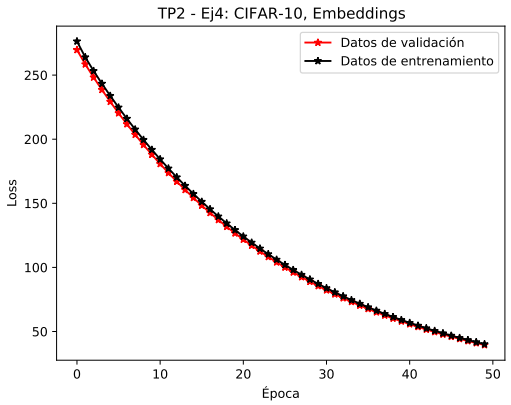
4. Ejercicio 4

Para este ejercicio se solicita resolver el problema anterior utilizando Embeddings y luego proponer una nueva arquitectura utilizando capas convolucionales 1D en lugar de capas densas.

Para la primera arquitectura se utilizó una capa de Embedding con 15 dimensiones de salida y 100 palabras por reseña. Luego se aplica una capa densa con 200 neuronas y con función de activación Relu dada la no linealidad de los datos y con Embeddings. Se utilizó el mismo código del ejercicio 3 con el regularizador de Dropout con un factor de 0.5, *learning rate* de 1, el optimizador Adadelta y la función de costo *Categorical Cross Entropy*, dado que es un problema de clasificación. Los resultados fueron:



(a) Accuracy

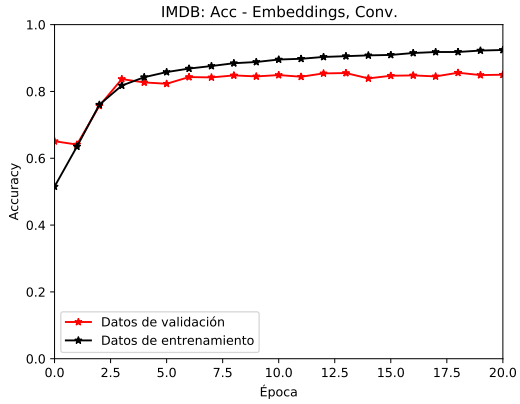


(b) Función de Loss

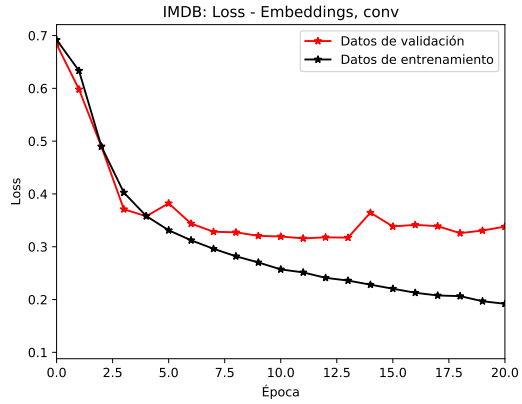
Figura 12: Problema de clasificacion de reseñas de IMDB utilizando Embeddings y capas densas

Del resultado anterior se ve que la capa Embedding ayuda a corregir el overfitting aunque el accuracy pasó de 85 % en el caso incial a 82 % para la arquitecura con Embedding.

La arquitectura que contiene capas convolucionales 1D tiene una capa convolucional 1D con un *kernel_size* de 2 y 128 filtros. Se aplica regularización por Dropout con factor de 0.5 y una capa de Pooling de *pool_size=2*. Finalmente se aplica un *flatten* y la salida es una capa densa de 1 neurona con activación sigmoidal. Se utiliza activacion sigmoidal dado que los valores de la salida pueden ser 0 o 1 que están en el rango de la salida de la función sigmoide. Los resultados de Accuracy y Loss para esta arquitectura se muestra en la siguiente figura:



(a) Accuracy



(b) Función de Loss

Figura 13: Problema de clasificacion de reseñas de IMDB utilizando Embeddings y capas densas

Ambos métodos permitieron conseguir valores de accuracy similares del orden de 84 %, pero la mayor diferencia se ve en el overfitting que, al menos para la arquitectura propuesta, la que utiliza capas convolucionales empeora el overfitting.

5. Ejercicio 5

Mediante este ejercicio se pretende implementar la arquitectura de la figura 15 para resolver el mapeo logístico que responde a la ecuación

$$x(t+1) = 4x(t)(1-x(t)), \tag{5.1}$$

para $x \in [0, 1]$. La función de activación de la neurona de salida es lineal.

Para este problema se generó una base de datos que responde a la ecuación 5.1 y se tomaron 1000 puntos al azar para conformar los datos de training y de testing. El 90 % de los datos totales se usaron para el entrenamiento y el 10 % restante para la validación. El conjunto de datos es:

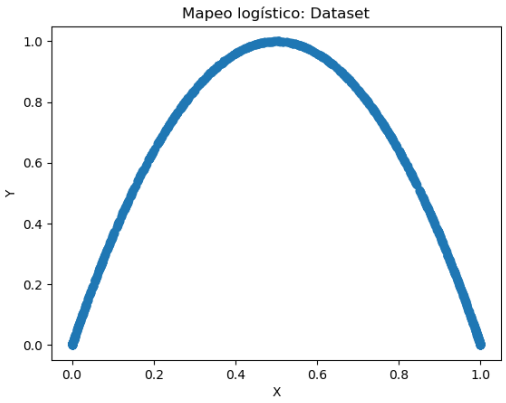


Figura 14: Dataset del problema 5: mapeo logístico

La arquitectura propuesta para este problema es una red con una capa densa oculta de 5 neuronas con función de activación **tanh**, una capa que concatena la entrada de la red con la salida de la primera capa oculta y finalmente la capa densa de salida con una neurona y con activación lineal (propuesto por el enunciado). La métrica utilizada fue MSE dado que nos permite ver directamente el error del entrenamiento y de los datos de validación. El optimizador usado fue Adadelta con un *learning rate* de 0.1. La regularización que se usó fue L2 con un coeficiente de regularización de 0,01.

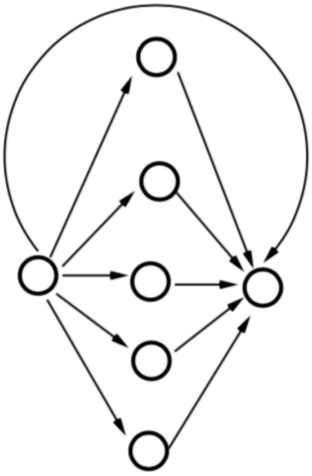


Figura 15: Arquitectura propuesta para el problema 5

Luego de entrenar por 50 épocas esta red, se obtienen el siguiente resultado:

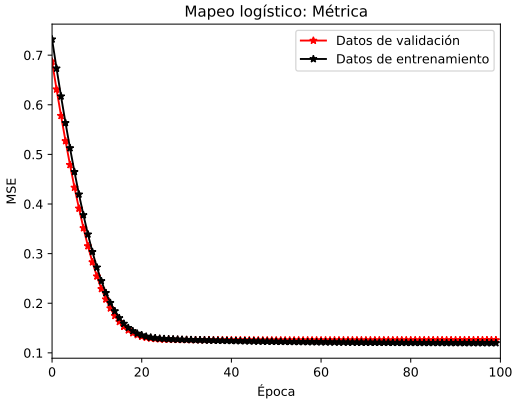


Figura 16: Error de entrenamiento de la red para el problema del mapeo logístico

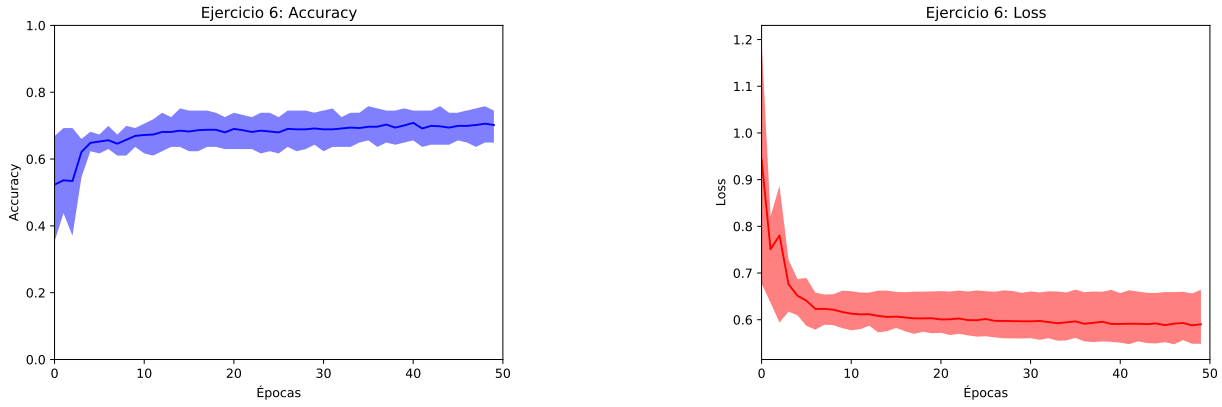
de donde se ve que el error disminuye a medida que se entrena la red y no se produce overfitting.

6. Ejercicio 6

En este ejercicio se trabaja con el dataset **pima-indians-diabetes** y se pretende diseñar una red para predecir la probabilidad de que un paciente tenga diabetes o no en función de un conjunto de factores del paciente como la cantidad de embarazos, concentración de glucosa plasmática, entre otros.

Se plantea utilizar el esquema de K-Folds con K igual a 5 para obtener una medida más precisa sobre la generalización del método propuesto. El método K-Folds, permite iterar K veces el entrenamiento de la red tomando en cada iteración un conjunto de datos distintos para testing y para training, La arquitectura de la red que se utilizó cuenta con cinco capas ocultas densas con 250, 100, 50, 50 y 50 neuronas con activación Relu y una capa densa de salida con una sola neurona y con activación sigmoidal. Se utiliza la función de activación Relu por la no linealidad del problema y la función de activación sigmoidal a la salida porque solo se puede clasificar en dos clases: con o sin probabilidad de tener diabetes. El optimizador es Adam con *learning rate* de 0,0001 y la función de costo es *Categorical Cross Entropy* dado que se trata de una clasificación en dos grupos. Además, se utilizó una regularización de tipo L2 con coeficiente de regularización $\lambda = 0{,}01$.

El resultado obtenido del Accuracy y de la Loss después de 50 épocas es el que se muestra a continuación:

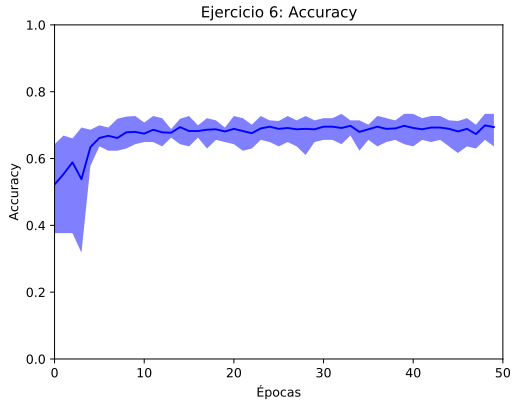


(a) Accuracy

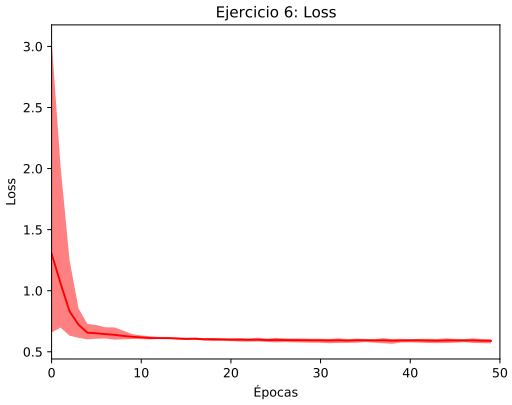
(b) Función de Loss

Figura 17: Predicción de que una persona desarrolle diabetes en función de un conjunto de parámetros obtenidos del dataset pima-indians-diabetes

Viendo los datos, se ve que hay faltante de algunos datos, los cuales son completados como 0.0 cuando en realidad deberían tener un valor distinto de 0. Por este motivo, se propone que si hay un dato faltante, entonces se complete este valor suponiendo el promedio de los datos del resto de los pacientes (excepto para el valor relacionado con la cantidad de embarazos). Esto puede considerarse una buena estimación y contribuye a que la red se entrene en función de datos que son más aproximados a la realidad.



(a) Accuracy



(b) Función de Loss

Figura 18: Predicción de que una persona desarrolle diabetes en función de un conjunto de parámetros obtenidos del dataset pima-indians-diabetes, cambiando los valores de los datos faltantes por el promedio de los datos de ese parámetro

Comparando ambos resultados, se ve que corrigiendo los datos faltantes con el promedio del resto de los pacientes la precisión del método mejora de 68 % a 72&.

7. Ejercicio 7

En este problema se busca implementar un autoencoder que permita eliminar el ruido a las imágenes de la base de datos de MNIST. Inicialmente debe realizarse un preprocesado de los datos dividiéndolos por 255 para que estos queden en el rango de 0 a 1. A partir de los resultados de la normalización anterior, se genera un conjunto de datos **Noisy** que surgen de sumar una señal de ruido gaussiana al conjunto de datos normalizados. Los parámetros de la gaussiana para generar la imagen ruidosa son: $\mu = 0$ y $\sigma = 0{,}5$. Al conjunto de datos **Noisy** se le aplica la función *clip* de Numpy para que los valores se encuentren entre 0 y 1.

La arquitectura para la red neuronal propuesta consiste en 5 capas densas ocultas con 1500, 750, 250, 750 y 1500 neuronas, con funciones de activación *Relu* y con regularizador L2 con un coeficiente de regularización de 0,001. La última capa es una capa densa de 784 neuronas y con la función de activación sigmoide, dado que la salida será una imagen normalizada en el rango de 0 a 1. El optimizador utilizado fue Adagrad con un *learning rate* de 1 y la función de costo usada fue MSE.

En la siguiente figura se muestran los resultados de aplicar el filtrado para eliminar el ruido de las imágenes. En la primera fila se muestran los datos originales, en la segunda el conjunto *Noisy* obtenido cuando se les aplicó el ruido gaussiano y en la última fila se muestran los resultados filtrados.

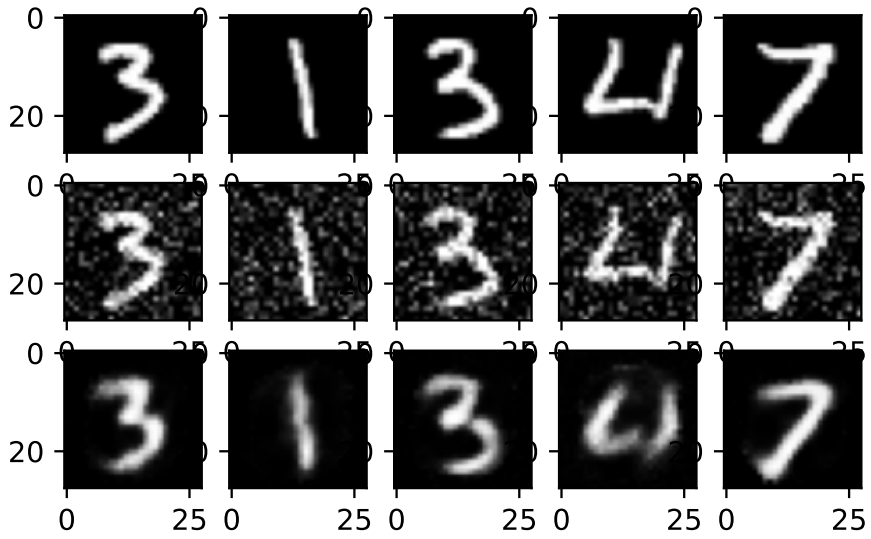


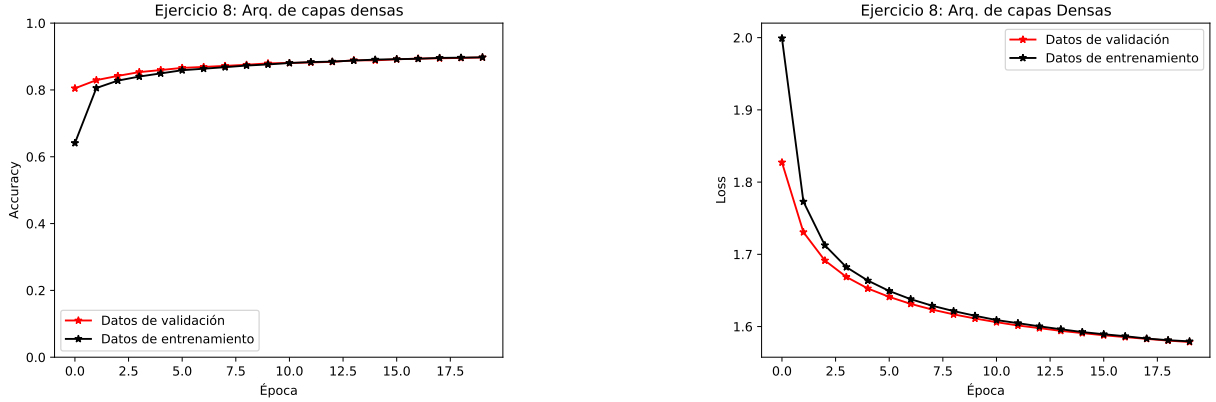
Figura 19: Resultados del autoencoder

8. Ejercicio 8

La primera parte de este ejercicio implica proponer e implementar una red neuronal con capas densas para resolver el problema de clasificación del set de datos MNIST. Para esto, lo primero que se hizo es preprocesar los datos restándole la media y dividiéndolos por la desviación estándar del conjunto de datos.

La arquitectura densa que se propone cuenta con 1 capa oculta de 100 neuronas con función de activación Relu y con regularización de Dropout con un factor de 0.15. La capa de salida tiene 10 neuronas (una por cada clase del conjunto de datos de MNIST) y con activación sigmoidal. Se utiliza la activación sigmoidal dado que los valores de la salida toman valores entre 0 y 1. El optimizador que se usó es Adagrad con *learning rate* de 0.01 y la función de costo que se usó fue *categorical cross entropy* dado que se trata de un proceso de clasificación con muchas clases.

Los resultados del Accuracy y de la función de costo para la arquitectura con capas densas fueron los siguientes:

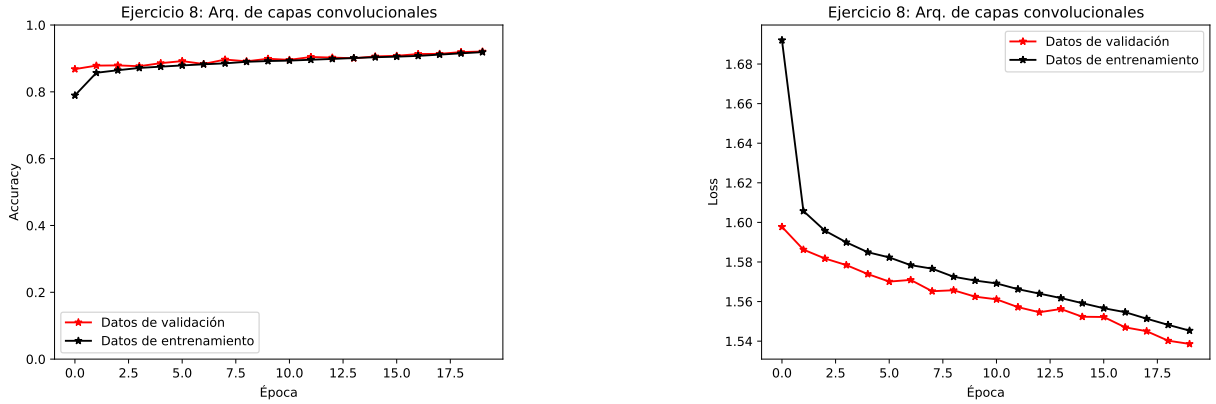


(a) Accuracy

(b) Función de Loss

Figura 20: Clasificación del conjunto de datos MNIST a partir de una arquitectura con capas densas

La segunda arquitectura que se propone para el proceso de clasificación de las imágenes de MNIST utiliza 2 capas convolucionales 2D con 16 filtros y un tamaño de kernel de (3×3) con regularización de Dropout con un factor de 0.5 y el optimizador de Adagrad con un *learning rate* de 0.001. Luego de 20 épocas se obtienen los siguientes resultados:



(a) Accuracy

(b) Función de Loss

Figura 21: Clasificación del conjunto de datos MNIST a partir de una arquitectura con capas convolucionales

Se ve que para los esquemas propuestos, la arquitectura con capas convolucionales permite alcanzar un accuracy ligeramente superior (93 %) al esquema con capas densas (90 %).

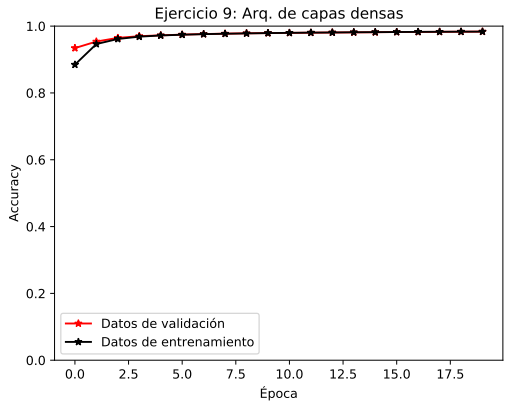
Las capas convolucionales permiten alcanzar una precisión un poco mayor que el esquema que tiene capas densas dado que las convolucionales se entrenan tomando solo una característica de la imagen, por ejemplo líneas horizontales. El uso de una arquitectura con capas densas tiene la ventaja de que para generar la información en una neurona de una capa siguiente se utiliza toda la información de la capa actual, lo que a su vez implica una gran desventaja cuando se tienen imágenes muy grandes. Las capas convolucionales en general agilizan el entrenamiento al utilizar solo una sección de la imagen de entrada a la capa para calcular la información en la neurona de la capa siguiente. A su vez esto se traduce en que a veces el entrenamiento no es adecuado por recortar demasiado la información.

Con respecto a la cantidad de parámetros a entrenar, en el caso de la arquitectura con capas densas se tienen 79510 parámetros entrenables y la arquitectura con capas convolucionales tiene 94650 parámetros.

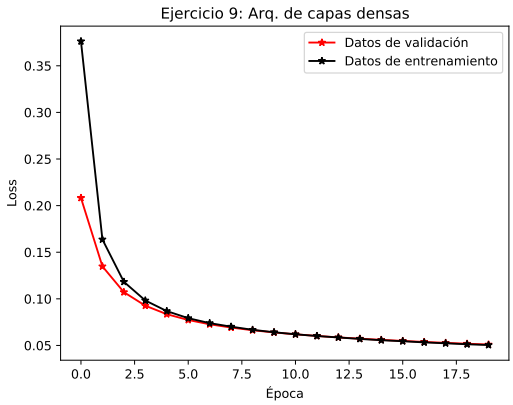
9. Ejercicio 9

Para este ejercicio se vuelve a utilizar el código implementado en la Sección 8 con la diferencia que durante el preprocesado de los datos se hacen permutaciones espaciales para el caso de una arquitectura de capas densas y una de capas convolucionales.

Para el caso de las capas densas el resultado fue el siguiente:



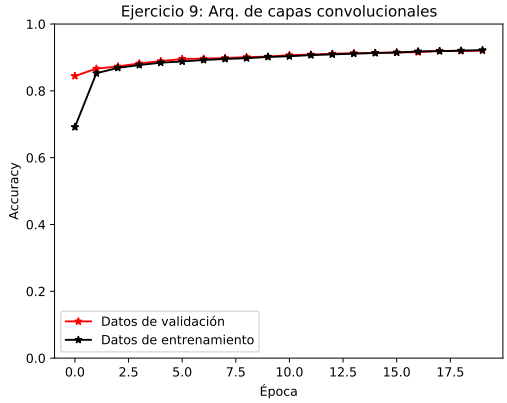
(a) Accuracy



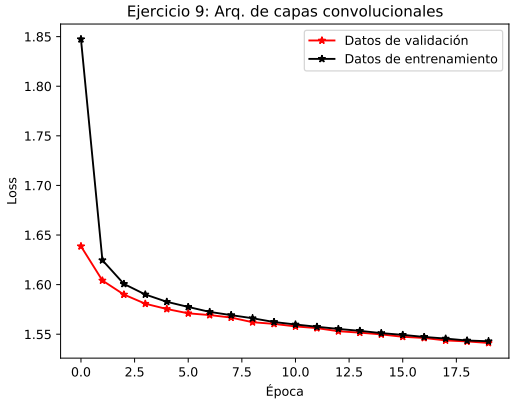
(b) Función de Loss

Figura 22: Clasificación del conjunto de datos MNIST a partir de una arquitectura con capas densas y permutaciones espaciales

Para la arquitectura que involucra capas convolucionales los resultados fueron:



(a) Accuracy



(b) Función de Loss

Figura 23: Clasificación del conjunto de datos MNIST a partir de una arquitectura con capas convolucionales y permutaciones espaciales

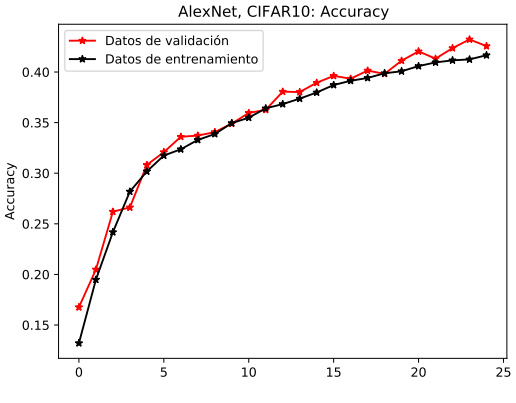
En este caso, a diferencia del anterior, se ve que la arquitectura con capas densas resulta alcanzar un mayor accuracy (98 %) en comparación con la arquitectura que involucra capas convolucionales la cual alcanza un 92 %. Es preciso notar que las permutaciones espaciales parecerían tener un efecto sobre la precisión de la red cuando se utilizan capas densas, sin embargo, la arquitectura con capas convolucionales no parece verse beneficiada ni perjudicada por las permutaciones espaciales.

10. Ejercicio 10

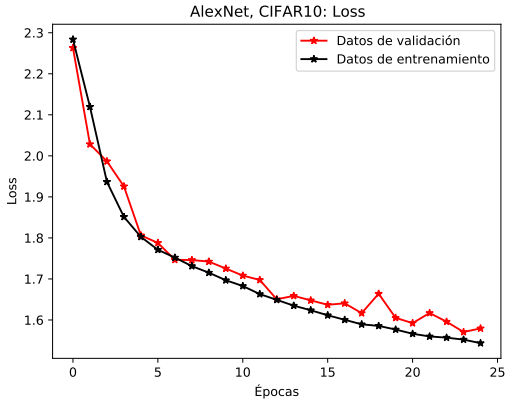
El último ejercicio consiste en la implementación de dos redes neuronales famosas llamadas **Alexnet** y **VGG16** para resolver el problema de clasificación de los dataset CIFAR-10 y CIFAR-100 utilizando aumentación de datos.

Las arquitecturas de las redes Alexnet y VGG16 pueden consultarse en la guía de trabajos prácticos 4 de la materia y los códigos pueden consultarse en [6]. Los resultados obtenidos se muestran a continuación para CIFAR-10 con optimizador Adagrad, *learning rate* de 0.01 y *batch size* de 128:

Inicialmente tenemos

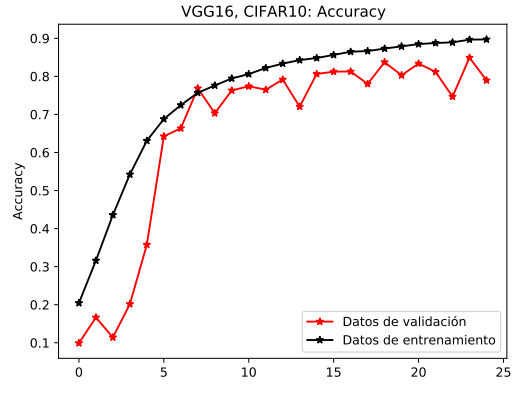


(a) Accuracy

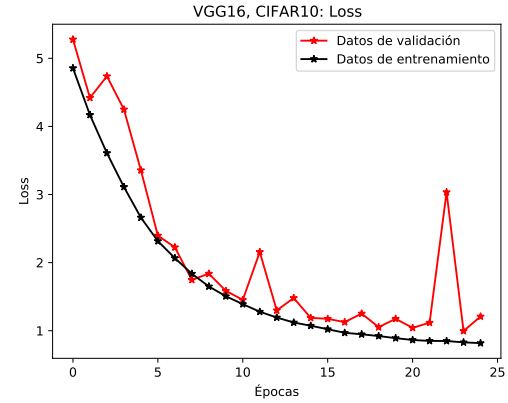


(b) Función de Loss

Figura 24: Clasificación del conjunto de datos CIFAR 10 con la arquitectura de Alexnet



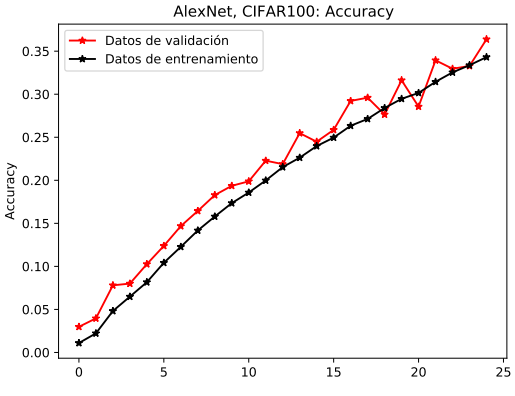
(a) Accuracy



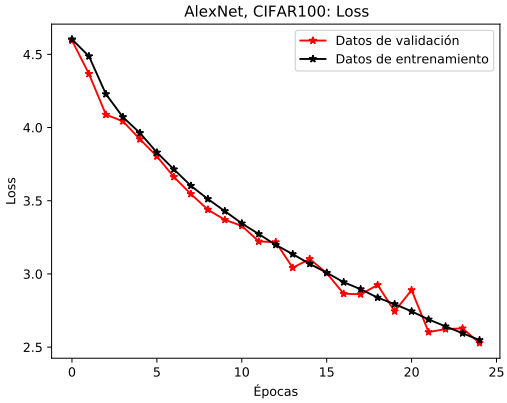
(b) Función de Loss

Figura 25: Clasificación del conjunto de datos CIFAR 10 con la arquitectura de VGG16

De los resultados anteriores se tiene que la arquitectura de VGG16 es muy superior para la clasificación del conjunto de datos de CIFAR-10, alcanzando un accuracy de hasta 75 % para el conjunto de datos de testing. Por su parte, la arquitectura Alexnet permite alcanzar un accuracy de solo 40 % para el conjunto de datos de testing.

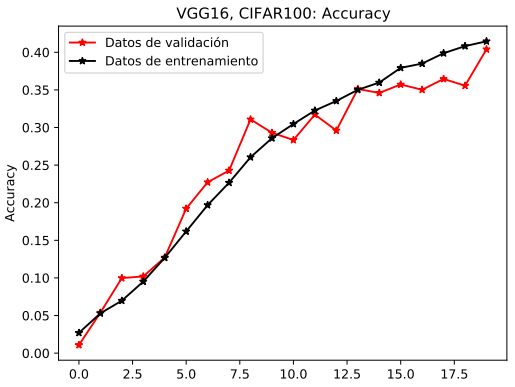


(a) Accuracy

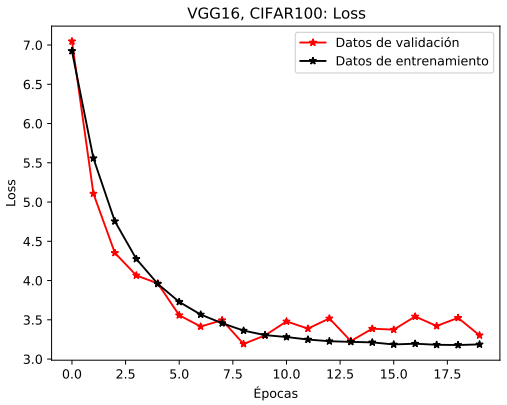


(b) Función de Loss

Figura 26: Clasificación del conjunto de datos CIFAR 100 con la arquitectura de Alexnet



(a) Accuracy



(b) Función de Loss

Figura 27: Clasificación del conjunto de datos CIFAR 100 con la arquitectura de VGG16

Para el caso de CIFAR-100 ocurre algo similar a lo concluido anteriormente. Para VGG16, el acuracy alcanzado despues de 20 épocas es de 40 % para el conjunto de datos de testing., mientras que la arquitectura Alexnet permite alcanzar un accuracy de solo 25 % para el conjunto de datos de testing luego de 20 épocas.

La conclusión directa de lo anterior es que el esquema de VGG16 es muy superior al de Alexnet, aunque en términos de costo computacional es 2 o 3 veces más costoso.

Referencias

[1] Julien Jacques, Linear Regression in High Dimension and/or for Correlated Inputs. Article in EAS Publications Series ů January 2015. https://www.researchgate.net/publication/276249836

[2] Ariel Curiale, German Matos, *Notas de la cátedra: Aprendizaje Profundo y Redes Neuronales Artificiales*. https://classroom.google.com/u/2/c/MTQxNDcxMTM5NjEx

[3] Fei-Fei Li, *Convolutional Neural Networks for Visual Recognition*. http://cs231n.stanford.edu/2017/

[4] Códigos del TP1: https://github.com/TomasLiendro/Deep_Learning/blob/master/TP1_Introduction%20to%20ML/

[5] Códigos del TP2: https://github.com/TomasLiendro/Deep_Learning/tree/master/TP2_Introduction%20to%20Neural%20Networks

[6] Códigos del TP4: https://github.com/TomasLiendro/Deep_Learning/tree/master/TP4_Convolutional%20Neural%20Networks