Práctica 1: Redes Neuronales y Aprendizaje Profundo para Visión Artificial

Evelyn G. Coronel
Aprendizaje Profundo y Redes Neuronales Artificiales
Instituto Balseiro

(21 de septiembre de 2020)

EJERCICIO 1

En este ejercicio se implementó una regresión lineal de la para un cantidad N de puntos en d dimensiones. Para realizar este ejercicio se generan los valores de x_i y $a_{m,exacto}$ de forma aleatoria entre $[-4, 4]$. se calcula el valor

$$y_i = \sum_{i=1}^{d} a_{i,esperado} x_i + a_0 + \epsilon[-1, 1] \tag{1}$$

donde se agrega un ruido uniforme que varía entre $[-1, 1]$

Una vez obtenido el conjunto de datos X, se calculan los parámetros esperados de la regresión lineal mediante la Ec. 2

$$\vec{a}_{esperado} = (X^T X)^{-1} X^T \vec{y}, \tag{2}$$

donde $\vec{a}$ representa a $\{a_i, a_0\}$, X es la representación matricial de datos e $\vec{y}$ son los valores y_i.

En las Figs. 1 y 2 se muestran los errores cuadráticos medios (MSE) en función de la cantidad de ejemplos dados para calcular los parámetros del problema así también de la salida. Se observa como el error disminuye exponencialmente con la cantidad de ejemplos.

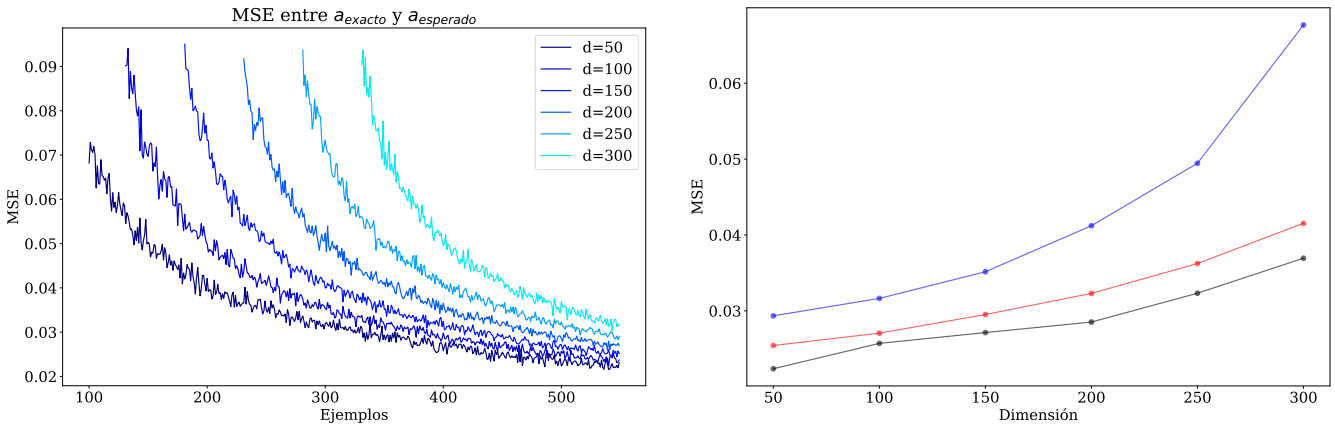

Fig. 1: Figura izquierda: MSE entre los parámetros exactos y obtenidos en función de los ejemplos de entrenamiento. Figura derecha: Se muestra en la figura el comportamiento del MSE en función de la dimensión con cantidad de ejemplos fija.

Se observa para una dimensión fija, el MSE disminuye a medida que se aumenta la cantidad de ejemplos para calcular los parámetros. En las mismas figuras se muestra como varía el valor de MSE fijando la cantidad de ejemplos, donde el error también aumenta exponencialmente con la dimensión, esto indica que a medida que voy a aumentando la dimensión del problema voy a necesitar más ejemplos para obtener un error menor.

Más aún, si consideramos un valor de MSE fijo, por ejemplo en la Fig.2 con $MSE \approx 0.5$, con $d = 80$ se necesitaron ~ 180 ejemplos para llegar a ese error,para $d = 120$ se necesitaron ~ 270 y para $d = 160$ se necesitaron ~ 380 ejemplos. Por lo que se puede decir que:

$$\Delta d \approx \Delta N \tag{3}$$

En el problema de la dimensionalidad se espera que $^{\Delta N}/_{\Delta d}$ siga una forma exponencial. En este caso sigue un comportamiento lineal debido a que el problema que se trabaja es una regresión lineal.

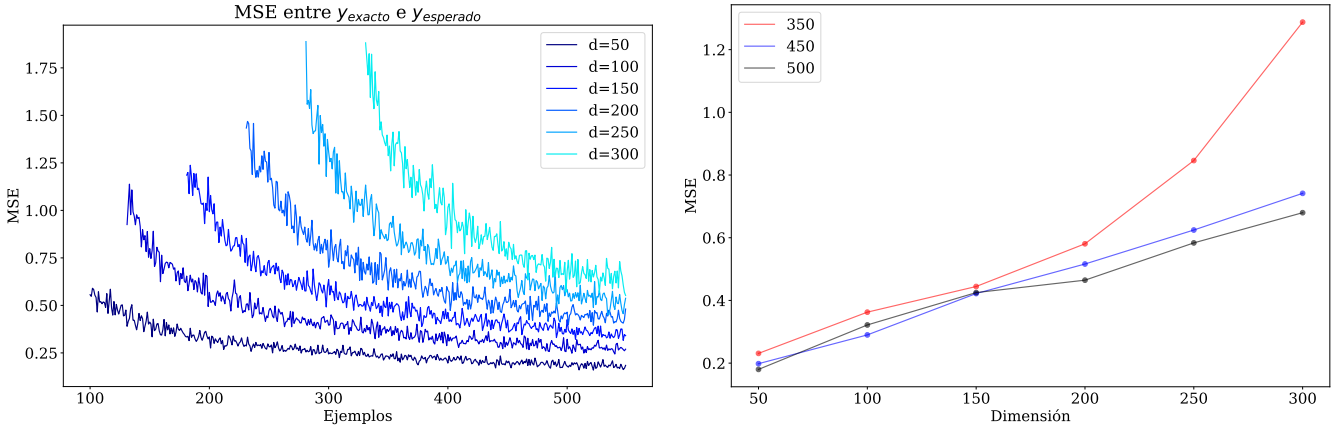

Fig. 2: Figura izquierda: MSE entre la salida y_i exacta y la obtenida en función de los ejemplos de entrenamiento. Figura derecha: MSE en función de la dimensión con cantidad de ejemplos fija.

EJERCICIO 2

En este ejercicio se generó un conjunto de datos en dimensión $d = 7$ distribuidos en $p = 4$ distribuciones normales con media y desviación estándar aleatorias entre $[-3, 3]$ y $[0.3, 1.3]$ respectivamente. Sobre este conjunto se implemente el algoritmo de *k-means* para clasificar a los puntos en 4 grupos.

El algoritmo usa la media del grupo, o centroide, para definir si un punto pertenece o no al grupo. Para empezar el algoritmo se inicializan los centroides con puntos aleatorios del conjunto de datos. A medidas que se realizan las iteraciones, los centroides tienden a las medias que se utilizaron para la inicialización de los datos, esto es propio a la forma con la que se generan los puntos para clasificar.

Dependiendo del conjunto de datos y de los centroides iniciales, el algoritmo puede converger a una solución coherente o no. En la Fig. 3 se muestra dos estados de la clasificación, la imagen izquierda es el estado iniciales de los puntos, con la clasificación y centroides aleatorios, mientras que en la imagen derecha se observa el estado final de las iteraciones donde los centroides calculados por el algoritmo convergen a la media de la distribuciones normales del conjunto.

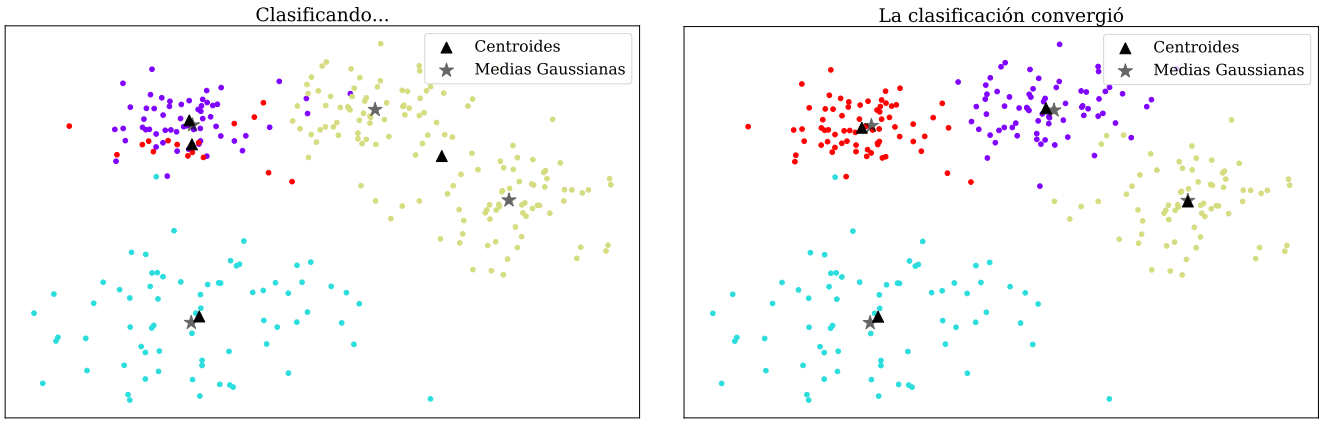

Fig. 3: Estados inicial y final de la clasificación del *k-means*. La figura izquierda muestra los valores iniciales de los centroides y la clasificación inicial de los puntos. En la figura derecha se observa que la clasificación convergió y los centroides convergen a puntos cercanos a las medias de las gaussianas usadas para generar los puntos.

En la figura 4 se observa un ejemplo de cuando el algoritmo no converge a una solución coherente con el conjunto de

datos generado.

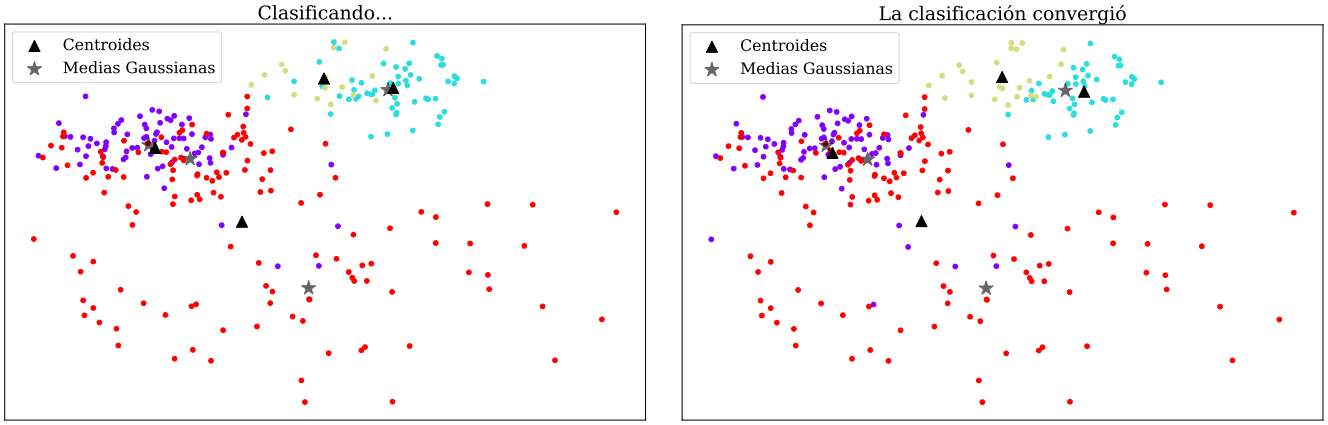

Fig. 4: Estados inicial y final de la clasificación del *k-means*. En esta ejecución del programa, los centroides no convergen a las medias y en general la clasificación es pobre y erronea.

EJERCICIO 3

Para este ejercicio se utilizó el módulo `datasets` de la librería `Keras` para cargar los datos de entrenamiento y validación del MNIST y CIFAR-10, para utilizar el algoritmo de *KNN* para la clasificación.

La ejecución del programa se imprime lo siguiente en la salida estándar:

```
Con el MNIST:
Dimensiones del set de entrenamiento:
(60000, 28, 28)

60000 ejemplos de entrenamiento
10000 ejemplos para probar

100.0% probando con 20 ejemplos


Con el CIFAR-10:
Dimensiones del set de entrenamiento:
(50000, 32, 32, 3)

50000 ejemplos de entrenamiento
10000 ejemplos para probar

30.0% probando con 20 ejemplos
```

EJERCICIO 4

Usando los datos generado para el ejercicio 2 como datos de entrenamiento y validación y se implementó el algoritmo *KNN* para clasificar 375 puntos, distribuidos en 5 distribuciones normales con media y dispersión aleatorias, en dos dimensiones para 1, 3 y 7 vecinos.

Dependiendo de la clasificación de *k-means*, el algoritmo de *KNN* puede funcionar o no. En el caso presentado en la Fig. 5 se observa que el *k-means* separó a los conjuntos de datos coherentemente pero el algoritmo de *KNN* convergió a una solución errónea, se presentaron 125 ejemplos de validación de los cuales el 36 % fueron clasificados correctamente.

En la Fig.6 se observa que el algoritmo converge a una solución, dando 99.2 de aciertos en los 125 ejemplos de validación.

Para una cantidad de vecinos $k = 3$, las simulaciones y clasificaciones en promedio daban un porcentaje de aciertos menor que $k = 1$. En el caso de la Fig. 7 se obtuvo un 94.4 % probando con 125 ejemplos.

Para una cantidad de vecinos $k = 7$, el rendimiento de las simulaciones y clasificaciones en promedio eran peor que los casos anterior. En el caso de la Fig. 8 se obtuvo un 72.0 % probando con 125 ejemplos. Para $k = 7$, la solución a la que converge el algoritmo no tiene la misma generalización que para $k = 1$ y $k = 3$.

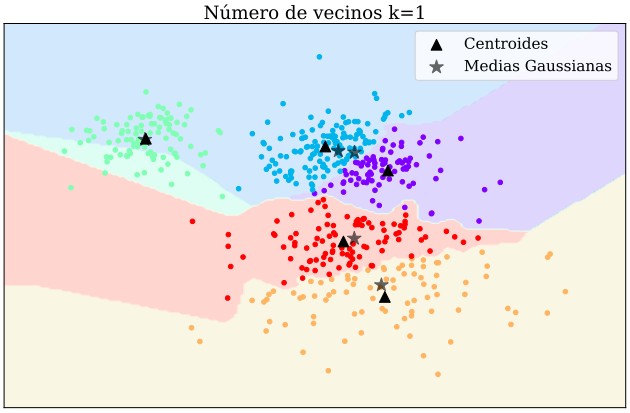

Fig. 5: Estado final de la clasificación del algoritmo de *KNN* con $K = 1$, en la validación con 125 datos solo se pudo calificar correctamente el 36.0 %. En este caso el algoritmo de *KNN* convergió a una solución errónea

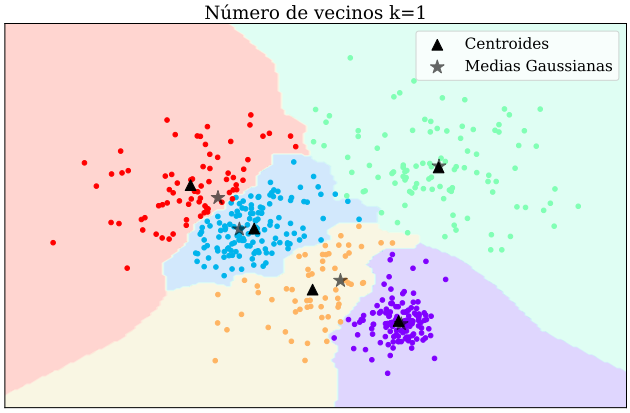

Fig. 6: Estado final con $K = 1$. Se observa que la clasificación se corresponde con el entrenamiento y en la validación se obtuve un 99.2 % de acierto con 125 datos. El algoritmo convergió a una solución correcta.

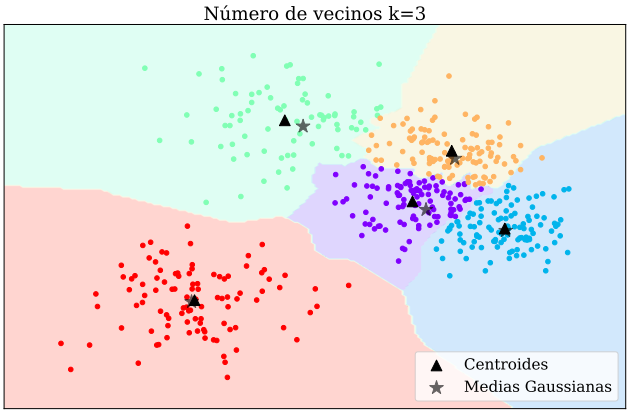

Fig. 7: Estado final con $k = 3$. Se observa que la clasificación se corresponde con el entrenamiento y en la validación se obtuve un 99.2 % de acierto con 125 datos.

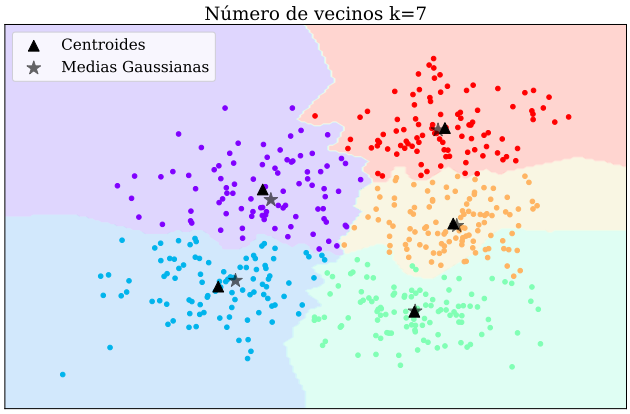

Fig. 8: Estado final con $k = 7$. El algoritmo pierde generalización y en la validación se obtiene un 72.0 % de acierto con 125 datos.

EJERCICIO 5

En este ejercicio se implementaron los clasificadores lineales *Support Vector Machine* y *SoftMax* para clasificación de los conjuntos de datos del *MNIST* y *CIFAR-10*, almacenados en el módulo `datasets` de la librería `keras`. módulo ya proporciona los datos de entrenamiento y validación para ambos casos. En la función de pérdida se agregó una función de regularización del tipo L_2.

Si implementó la clase `LinearClassifier` donde se definieron las funciones de:

- Inicialización:

En la inicialización se definen los parámetros de tasa de aprendizaje η, la cantidad de épocas `epochs`, el parámetro de la normalización λ_{L2}, el tamaño del batch de datos para el *gradiente estocástico por batch*, y también se define si se implementa el uso de bias o no.

- `fit`:

  Esta función recibe el conjunto de datos de entrenamiento y validación. Transforma las imágenes en vector unidimensionales, dependiendo si utiliza el bias, se agregan el uno al inicio de cada imagen.

  También en esta función se inicializa la matriz de peso de forma uniforme entre $[-0.1, 0.1]$, como también se normaliza a las imágenes con el valor máximo de cada elemento que estas dos bases de datos es 255.

  Esta función recorre el conjunto de datos durante la cantidad de épocas definida en la inicialización. También realiza el entrenamiento por batch y la validación. Los valores de precisión y pérdida por época se almacenan en vectores inicializados en esta función.

- `predict`

  Esta función devuelve la clasificación predicha por la matriz de pesos dada un época.

- `loss_gradient`

  Esta función es propia de cada método de clasificación lineal. En la definición de las subclases `SVM` y `SMC` se detallan la funciones de pérdida y el gradiente de la matriz de pesos.

La precisión y la pérdida en función para los datos son calculados con las siguientes parámetros:

	η	Épocas	Batch	λ_{L2}	Bias
MNIST	0.003	50	400	0.001	Sí
CIFAR-10	0.0001	50	400	0.001	Sí

Los mismos se utilizaron para ambos clasificadores.

MNIST

La Fig. 9 se muestran los valores de precisión y pérdida en función de la época para CIFAR-10. Los resultados obtenidos con los algoritmos de *VSM* y *SoftMax* dan una precisión del $\sim 92\,\%$ en los datos de validación, por lo que para esta base de datos no se observan diferencias en el rendimiento de ambos algoritmos.

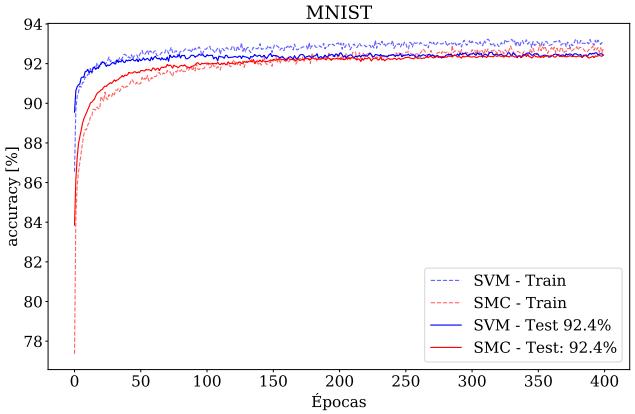

Fig. 9: Precisión de los algoritmos sobre el MNIST

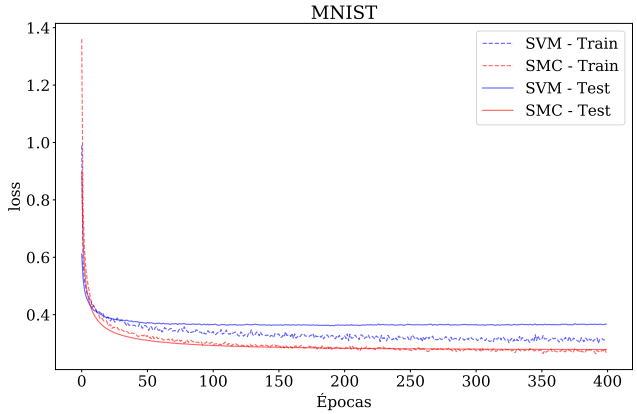

Fig. 10: Pérdida de los algoritmos sobre el MNIST

CIFAR-10

Las Figs. 11 y 12 se muestran los valores de precisión y pérdida en función de la época para CIFAR-10. Los resultados obtenidos tienen una presición menor que con la base de datos del MNIST, ya que la presición no supera el 40 % para

el algoritmo *SoftMax* y 35 % para *VSM* . Esto se debe a que el CIFAR-10 tiene imagenes con distintas propiedades, como color, forma, fondo, mientras que el MNIST solo tiene la información de una componente de color y son imagenes de números en un fondo blanco, el problema es menos complejo que CIFAR-10. En este caso, tambien la precisión de *SoftMax* es mejor que el clasificador *VSM*.

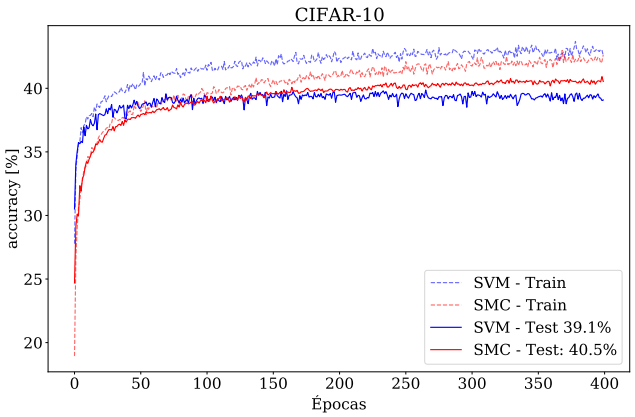

Fig. 11: Precisión en función de las épocas de los algoritmos sobre el CIFAR-10

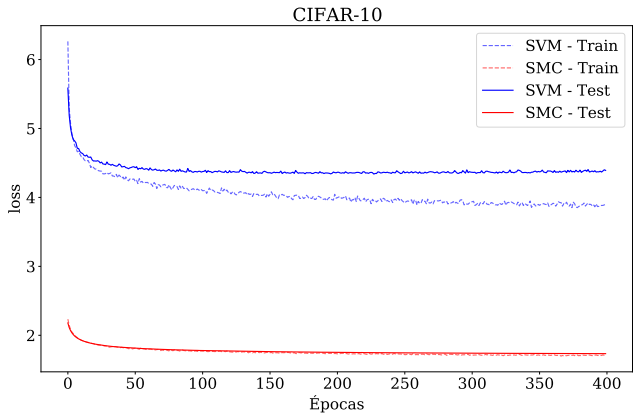

Fig. 12: Pérdida en función de las épocas de los algoritmos sobre el CIFAR-10