Redes Neuronales y Aprendizaje Profundo para Visión Artificial

Practica I: Introducción a las técnicas de ML

Enrique Nicanor Mariotti (mariottien@gmail.com)

Octubre 2019

1. Ejercicio I

Se implemento una regresión lineal n-dimensional para un conjunto de i puntos. La solución del modelo se realiza mediante la aproximación de mínimos cuadrados ordinarios. Bajo esta aproximación, la solución es cerrada y esta dada por:

$$\boldsymbol{\beta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y} \tag{1}$$

Aquí, $\mathbf{X}$ es una matriz que contiene los i ejemplos n-dimensionales almacenados en sus filas. La matriz $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}$ es a veces llamada matriz de *Moore-Penrose*.

Finalmente, $\mathbf{Y}$ es el vector objetivo para los los i ejemplos y $\boldsymbol{\beta}$ es el vector de pesos de la aproximación lineal n-dimensional.

En el presente ejemplo, los datos se obtuvieron sumando ruido gaussiano de media $\mu = 0$ y $\sigma = 3$ a una función lineal n-dimensional de pesos aleatorios.

Para ejemplo unidimensional con $i = 100$ puntos, ejecútese:

```
python pr-1-ej-1.py test
```

La imagen 1 ilustra este ejemplo.

Para obtener el vector de pesos $\boldsymbol{\beta}$ para i ejemplos n-dimensionales, ejecútese:

```
python pr-1-ej-1.py fit [-n DIMENSIÓN] [-i PUNTOS]
```

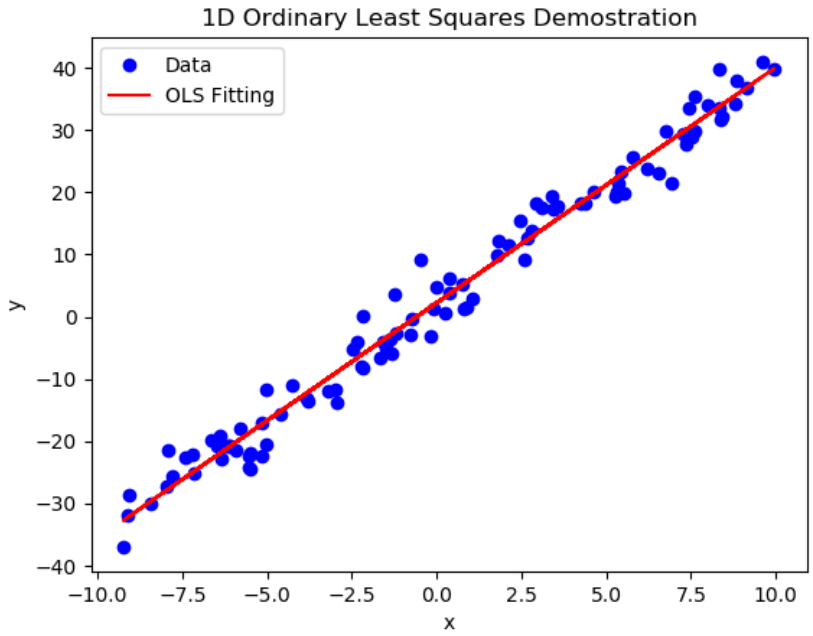

Figura 1: Ejemplo unidimensional de regresión lineal calculado mediante la Ec. 1 de mínimos cuadrados ordinarios.

2. Ejercicio II

Aquí se propone implementar un algoritmo de *clustering* no supervisado mediante el método de *K-Means*.

La secuencia del algoritmo se resume de la siguiente manera:

1. **Inicialización**: Previamente debe elegirse un numero k de *clusters* para clasificar los datos. Seguidamente se establecen k centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente.

2. **Asignación de objetos a los centroides**: Cada elemento de los datos es asignado a su centroide más cercano según su distancia *Euclídea*.

3. **Actualización de centroides**: Se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los ejemplos pertenecientes a dicho grupo.

El criterio de convergencia se fija en una tolerancia $0.1\,\%$ de movimiento relativo de los centroides respecto a su posición en la iteración anterior. Además, se fija un máximo de 400 iteraciones.

En el presente ejemplo, los i datos se obtuvieron a partir de p distribuciones gaussianas con μ y σ aleatorios en el intervalo $[0, 10]$.

El hiperparametro k que indica el numero de *clusters* que se debe de definir de antemano por el el usuario, por un método del tipo **ELBOW**, o similar.

Para un ejemplo generado por $p = 3$ distribuciones gaussianas de de $i = 100$ puntos bidimensionales puede ejecutarse:

```
python pr-1-ej-2.py test
```

La imagen 1 ilustra este ejemplo para un numero de *clusters* fijado en $k = 3$. Para correr el caso mas general, ejecútese:

```
python pr-1-ej-2.py fit [-i PUNTOS] [-n DIMENSIÓN] [-p
DISTRIBUCIONES] [-k CLUSTERS]
```

Figura 2: Ejemplo bidimensional del algoritmo *K-Means* con $p = k = 3$.

3. Ejercicios III & IV

Aquí se propone implementar un algoritmo de clasificación supervisado *K-Nearest Neighbors*.

La secuencia del algoritmo se resume de la siguiente manera:

1. **Distancia hacia los ejemplos**: Previamente debe elegirse un numero k vecinos cercanos para clasificar los datos. Seguidamente se calcula la distancia *Euclídea* desde el elemento a clasificar hacia cada uno de los ejemplos de entrenamiento.

2. **Tabla de cercanía**: Las distancias hacia el elemento a clasificar y los índices correspondiente a las clases de los ejemplos de entrenamiento se almacenan en una colección ordenada en orden ascendiente en distancia.

3. **Conteo de k-vecinos**: Seleccionados los k vecinos mas cercanos al elemento a clasificar, el índice de dicho elemento corresponde con el índice de mayor incidencia en la colección de k vecinos cercanos.

La ejecución del código para $k \in \{1, 3, 7\}$ se realiza mediante:

```
python pr-1-ej-3-4.py test
```

Aquí, se generaron $i = 200$ puntos bidimensionales gaussianos, correspondientes $p = 5$ clases diferentes con medias en el intervalo $\mu \in [0, 9]$. Los resultados pueden observarse en la Figs. 3, 4 & 5.

A su vez, el subespacio comprendido en $x \in [0, 10] \bigwedge y \in [0, 10]$ se discretizó con una resolución espacial de $\epsilon = 0.1$ unidades para su clasificación.

Observando las fronteras de decisión generadas es posible concluir que al reducirse el numero de vecinos cercanos, hay una tendencia clara del clasificador a sobre-ajustar los ejemplos de entrenamiento.

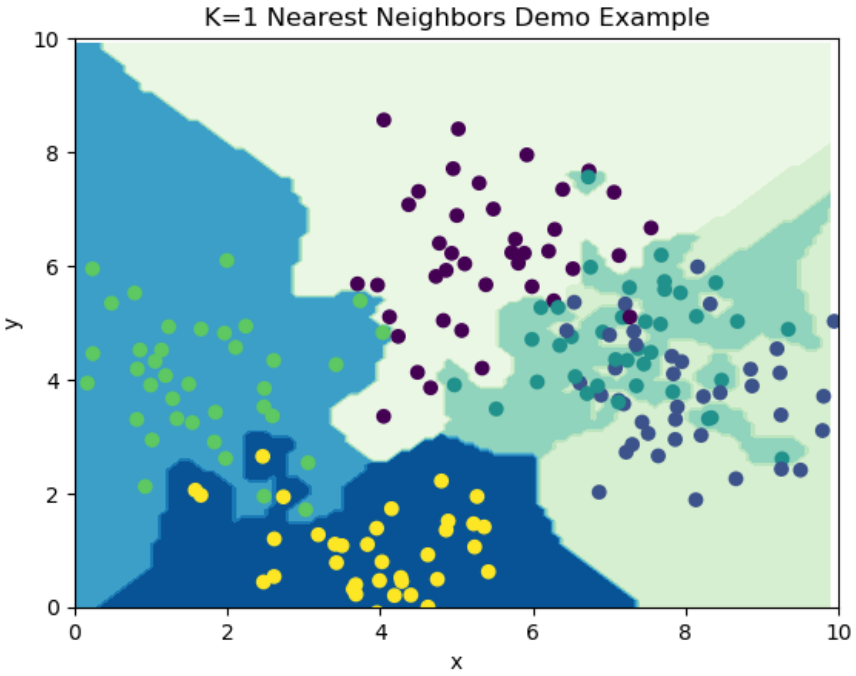

Figura 3: Ejemplo bidimensional del algoritmo *K-Nearest Neighbors* con $k = 1$.

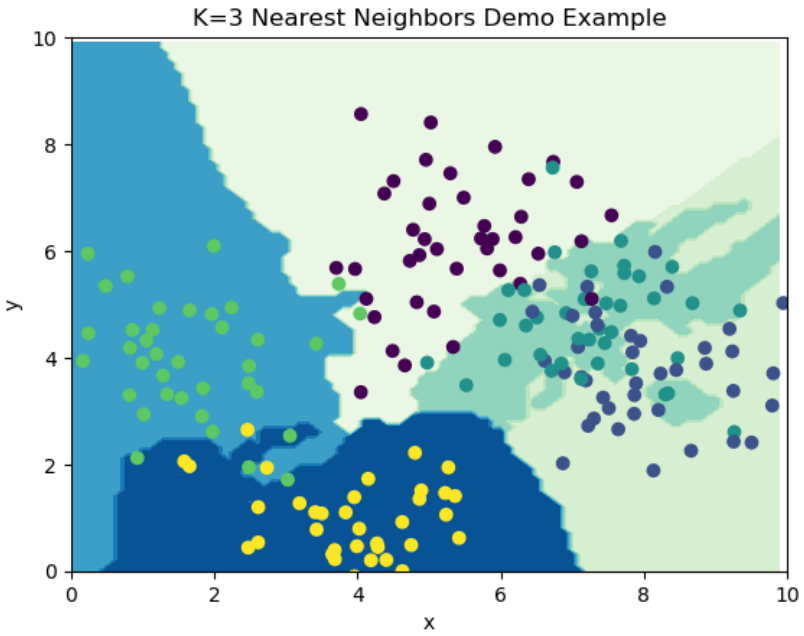

Figura 4: Ejemplo bidimensional del algoritmo *K-Nearest Neighbors* con $k = 3$.

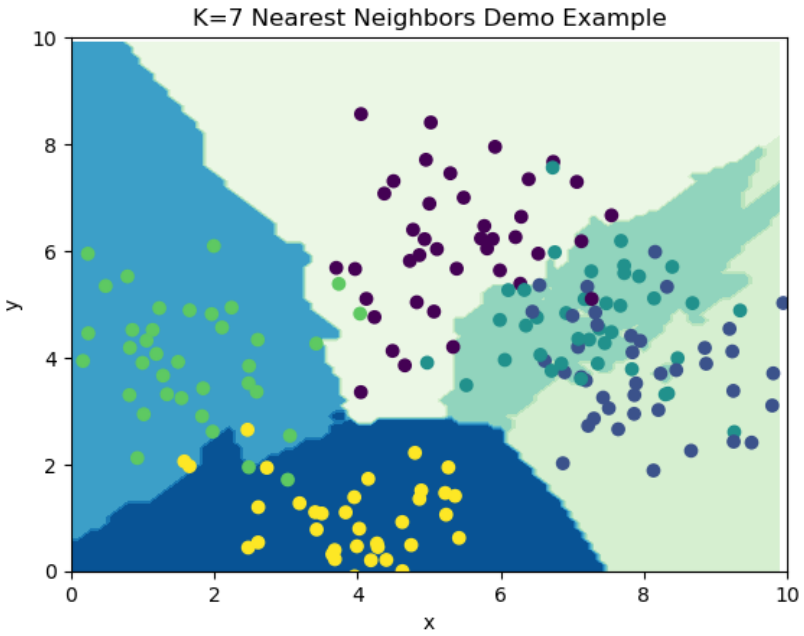

Figura 5: Ejemplo bidimensional del algoritmo *K-Nearest Neighbors* con $k = 7$.

El mismo procedimiento es susceptible a ser aplicado a imágenes, llevando las mismas a formato vectorial. Para ello se emplearon las bases de datos **MNIST**

y **CIFAR10**, ejemplificadas en la Fig. 6 y en la Fig. 7, respectivamente.

Se emplearon los conjuntos completos de entrenamiento, consistiendo de 50000 ejemplos para **CIFAR10** y 60000 ejemplos para **MNIST**. Los conjuntos de pruebas se limitaron a 10 ejemplos en ambos casos

La ejecución de :

```
python pr-1-ej-3-4.py predict [-k K-VECINOS] -d {cifar10, mnist}
```

Devuelve el valor medio de la precision sobre el conjunto de prueba. Para $k = 1$ mediante el método de *K-Nearest Neighbors*:

$$meanAcc^{CIFAR10} \approx 0.4$$
$$meanAcc^{MNIST} \approx 1.0$$

Si bien el algoritmo es capas de resolver problemas no-linealmente separables la dimensionalidad del problema **CIFAR10** lo vuelve impracticable.

Figura 6: Segmento de los datos de entrenamiento del conjunto **MNIST**.

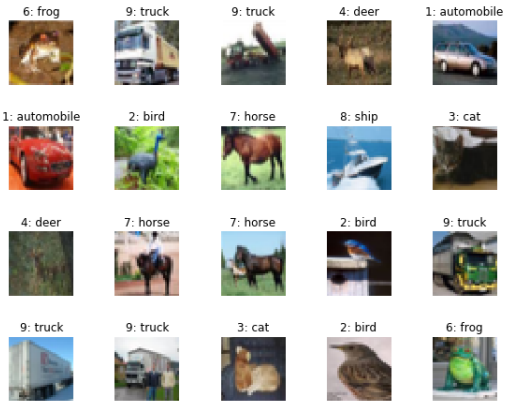

Figura 7: Segmento de los datos de entrenamiento del conjunto **CIFAR10**.

4. Ejercicios V & VI

Aquí se trata de resolver el problema de clasificación de **CIFAR10** y **MNIST** mediante el empleo de 2 clasificadores lineales, a saber, *Support Vector Machine* y *Softmax*.

En ambos casos se intenta solucionar el problema lineal:

$$f(x_i, \mathbf{W}, b) = \mathbf{W}x_i + b \tag{2}$$

Donde la función $f(x_i, \mathbf{W}, b)$ mapea los vectores x_i de los ejemplos a un índice de clasificación de clases.

Para encontrar los valores óptimos de la matriz $\mathbf{W}$ se emplea el método del gradiente descendente estocástico (SGD) por mini-batches. La actualización iterativa de los pesos se realiza en proporcional al gradiente de una función de costo a definir según el método.

La función de costo multiclase de *Support Vector Machine* se define como:

$$loss_{svm} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq y_i}^{c} \max\left(0, f_j - f_{y_i} + \Delta\right) + \lambda\|\mathbf{W}\|^2 \tag{3}$$

Donde el subíndice j designa clases y el subíndice i designa los ejemplos. Aquí, y_i es la clase correcta y f_i el *score* para el ejemplo i resultado de la transformación lineal de la Ec. 2. Por otra parte, el hiperparámetro Δ designa el margen de clasificación del hiperplano de decisión del presente método.

La Ec. 3 incluye una regularización del tipo *L2* que se suma a a función de costo, como mecanismo para disminuir el sobre-ajuste del problema.

Por otro lado, la función de costo de *Softmax* se define como:

$$loss_{softmax} = \frac{1}{n}\sum_{i=1}^{n} -log\left(\frac{e^{f_{y_i}}}{\sum_j^c e^{f_j}}\right) + \lambda\|\mathbf{W}\|^2 \tag{4}$$

Nuevamente, el subíndice j designa clases y el subíndice i designa los ejemplos. Aquí, y_i es la clase correcta y f_i el *score* para el ejemplo i resultado de la transformación lineal 2.

Al igual que la Ec. 3, la Ec. 4 incluye una regularización del tipo *L2* que se suma a la función de costo.

Una vez calculada la función de costo y sus respectivos gradientes. Los pesos son actualizados según:

$$\mathbf{W} = \mathbf{W} - \nabla_{\mathbf{W}} loss \tag{5}$$

Para esta implementación, el numero de *epochs* totales se fijo en 100. El tamaño de los mini-batches para el método del *SGD* se tomo en 16 ejemplos. El *learning rate* se mantiene fijo en $lr = 0.07$ para el caso de la función *Softmax* y como $lr = 0.09$ para el caso de *Support Vector Machine* multiclase. La regularización de la función de costo se realiza con un coeficiente de $\lambda = 0.01$. Los pesos $\mathbf{W}$ se inicializan en valores aleatorios gaussianos normalizados ($\mu = 0$ y $\sigma = 1$) escalados al orden de 1×10^{-4}.

En principio, el conjunto de ecuaciones anteriores solo es útil para la resolución de problemas linealmente separables, es decir, problemas de optimización plenamente convexos.

Como caso de aplicación se emplearon las bases de datos **MNIST** y **CIFAR10**, ejemplificadas en la Fig. 6 y en la Fig. 7, respectivamente. Se usan aquí los conjuntos completos de entrenamiento y de prueba.

A su vez, también es posible generar y clasificar un problema de 1000 ejemplos correspondientes a 2 *clusters* gaussianos linealmente separables en $\Re^2$.

La llamada al clasificador lineal se realiza mediante:

```
python pr-1-ej-5-6.py [svm | softmax] [-e EPOCHS] [-lr LEARNING
  RATE] [-bs BATCH SIZE] [-r REG STRENGTH] -d {cifar10, mnist,
                                blobs}
```

Para el caso de **MNIST** tanto el método de *Support Vector Machine* como de *Softmax* logran resolver el problema con relativa rapidez, como se muestra en la Fig. 9 y la Fig. 8.

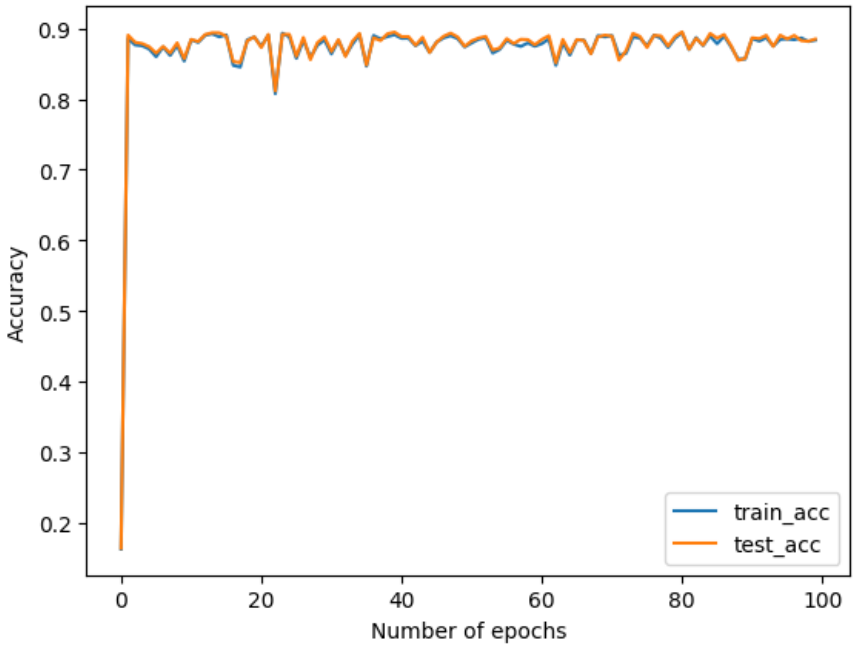

Figura 8: **SVM:** Precisión media en función de las *epochs* para el problema **MNIST**.

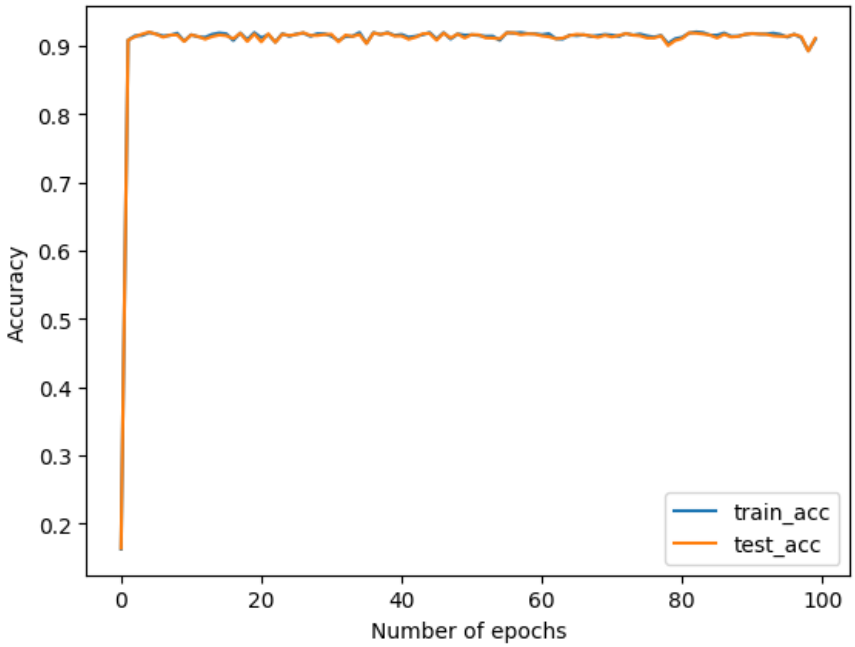

Figura 9: **Softmax:** Precisión media en función de las *epochs* para el problema **MNIST**.

Por el contrario, como se muestra en la Fig. 11 y la Fig. 10, para el caso de **CIFAR10** tanto el método de *Support Vector Machine* como de *Softmax* fallan en resolver el problema, de la misma forma que en la resolución de la Sección 3. No solo el costo computacional es elevado para problemas de alta dimension, si no que en el caso de **CIFAR10** no hay garantías de que se trate de un problema linealmente separable.

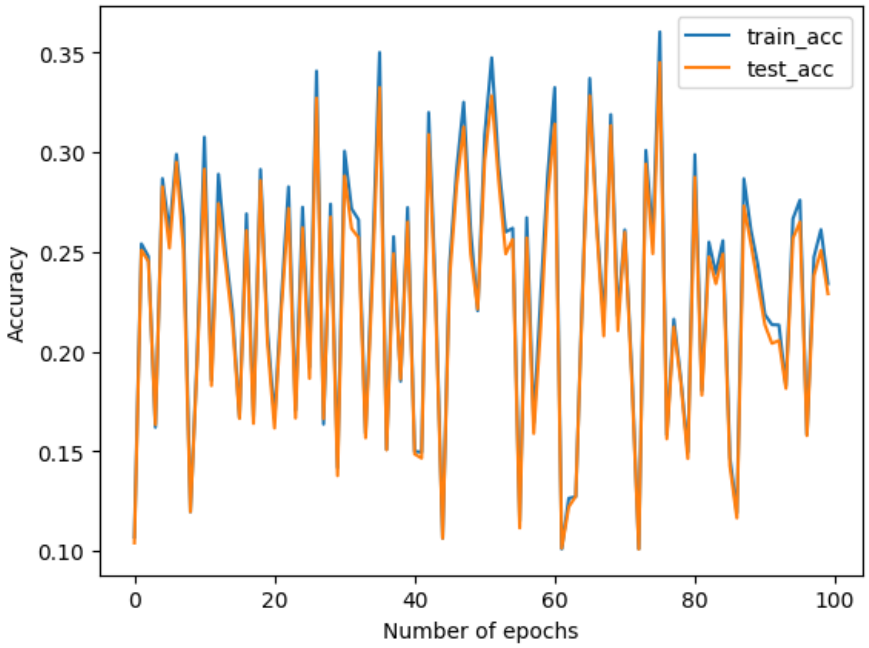

Figura 10: **SVM:** Precisión media en función de las *epochs* para el problema **CIFAR10**.

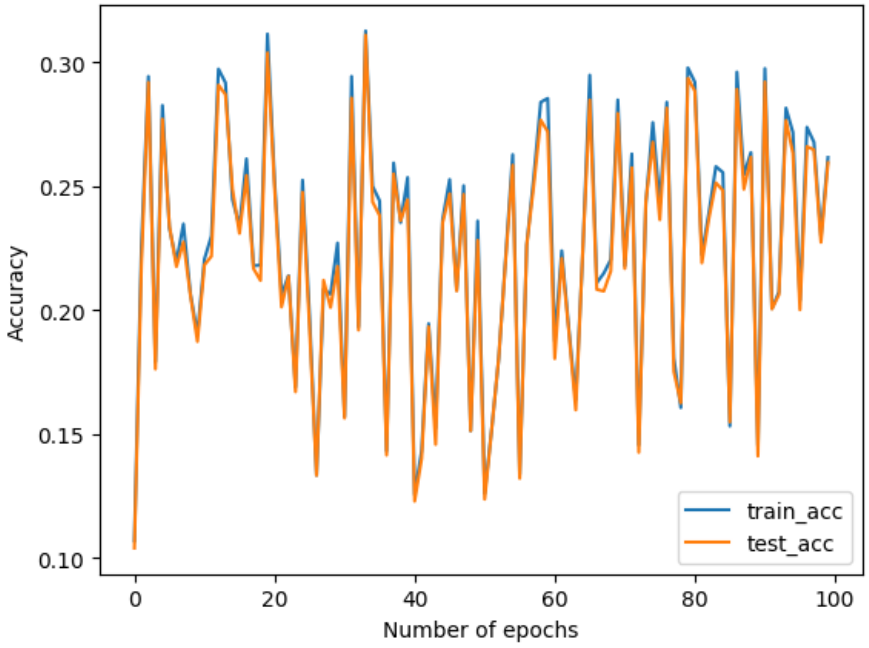

Figura 11: **Softmax:** Precisión media en función de las *epochs* para el problema **CIFAR10**.