

Motor de Redes Neuronales

Álvaro Rodríguez González, Lucía Afonso Medina,
Alejandro Alemán Alemán

Fecha: December 22, 2024

Contents

1	Introducción	3
2	Conjuntos de datos	4
3	Conjunto de Datos Iris	4
3.1	Boxplot del Conjunto de Datos Iris	6
4	Métodos y Detalles de Implementación	7
4.1	Funciones de Activación	8
4.2	Propagación hacia Adelante	9
4.3	Propagación hacia Atrás	10
4.4	Función de Costo	12
4.5	Optimizadores	13
4.5.1	Descenso por gradiente estocástico	13
4.5.2	Adam	13
4.6	Predicción	14
5	Experimentos y Resultados	15
5.1	Optimizador Estocástico	15
5.2	Optimizador Adam	17
6	Conclusiones	19
7	Trabajo Futuro	19
8	Bibliografía	20

Abstract

Este estudio presenta la aplicación de una red neuronal multicapa para la clasificación del conjunto de datos Iris, optimizada mediante técnicas de retropropagación y algoritmos de optimización como el descenso por gradiente y Adam. La red emplea funciones de activación como ReLU en las capas ocultas y Softmax en la capa de salida para producir probabilidades de clasificación. La función de costo utilizada es la entropía cruzada, que mide la discrepancia entre las predicciones del modelo y las etiquetas reales, guiando el ajuste de los parámetros para minimizar el error de clasificación.

1 Introducción

En los últimos años, las redes neuronales se han consolidado como herramientas clave en el aprendizaje automático, destacando en áreas como visión artificial y procesamiento del lenguaje natural. Este proyecto se centra en diseñar, implementar y optimizar una red neuronal para resolver el problema clásico de clasificación en el conjunto de datos Iris.

El trabajo se divide en fases que incluyen la construcción, entrenamiento y evaluación del modelo. En una primera etapa, se diseñó una red con estructura fija, mientras que en esta segunda fase se ha añadido flexibilidad para ajustar dinámicamente el número de capas, neuronas y el tamaño de los lotes. Además, se describen aspectos fundamentales del desarrollo, como el contexto del conjunto de datos Iris, las funciones de activación (ReLU y Softmax) y su impacto en el aprendizaje, y los procesos de propagación hacia adelante y retropropagación (Backtracking), con énfasis en la entropía cruzada como función de costo.

También se analizan los métodos de optimización implementados y su contribución a la minimización del costo. Finalmente, se presentan los experimentos realizados, las conclusiones del estudio y posibles mejoras futuras. Este trabajo no solo aborda una aplicación específica, sino que proporciona una base sólida para explorar desarrollos futuros en aprendizaje automático.

2 Conjuntos de datos

El conjunto de datos Iris es un clásico en el aprendizaje automático, con 150 muestras de flores de iris, cada una descrita por cuatro características: longitud y ancho del sépalo, y longitud y ancho del pétalo.

Los datos se preprocesaron dividiéndolos en conjuntos de entrenamiento, validación y prueba. Las etiquetas de destino se convirtieron en formato de codificación one-hot, lo que permite utilizar la entropía cruzada como función de pérdida durante el entrenamiento.

3 Conjunto de Datos Iris

El conjunto de datos *Iris* es un referente clásico en el ámbito del aprendizaje automático y la estadística, introducido por el estadístico y biólogo británico Ronald Fisher en 1936 [1]. Este conjunto comprende 150 muestras de flores de iris, distribuidas equitativamente entre tres especies: *Iris setosa*, *Iris versicolor* y *Iris virginica*. Cada muestra se describe mediante cuatro características métricas:

- **Longitud del sépalo** (en centímetros)
- **Ancho del sépalo** (en centímetros)
- **Longitud del pétalo** (en centímetros)
- **Ancho del pétalo** (en centímetros)

Estas características permiten analizar y clasificar las diferentes especies de iris en función de sus medidas morfológicas.

A continuación, se presentan ejemplos representativos de cada especie, mostrando las medidas correspondientes:

Especie	Longitud Sépalo (cm)	Ancho Sépalo (cm)	Longitud Pétalo (cm)	Ancho Pétalo (cm)
<i>Iris setosa</i>	5.1	3.5	1.4	0.2
<i>Iris versicolor</i>	7.0	3.2	4.7	1.4
<i>Iris virginica</i>	6.3	3.3	6.0	2.5

Table 1: Ejemplos de medidas para cada especie de Iris.

Para una comprensión más visual, se incluyen imágenes de cada especie de iris:



Figure 1: Imágenes representativas de las especies de Iris: *Iris setosa*, *Iris versicolor* y *Iris virginica*.

En cuanto a las estadísticas descriptivas del conjunto de datos, se observan las siguientes medidas promedio para cada característica:

Característica	Media (cm)	Desviación Estándar (cm)	Mínimo (cm)	Máximo (cm)
Longitud Sépalo	5.84	0.83	4.3	7.9
Ancho Sépalo	3.05	0.43	2.0	4.4
Longitud Pétalo	3.76	1.76	1.0	6.9
Ancho Pétalo	1.20	0.76	0.1	2.5

Table 2: Estadísticas descriptivas de las características del conjunto de datos Iris.

Estas estadísticas proporcionan una visión general de la distribución de las medidas en el conjunto de datos, evidenciando la variabilidad y rango de las características observadas.

3.1 Boxplot del Conjunto de Datos Iris

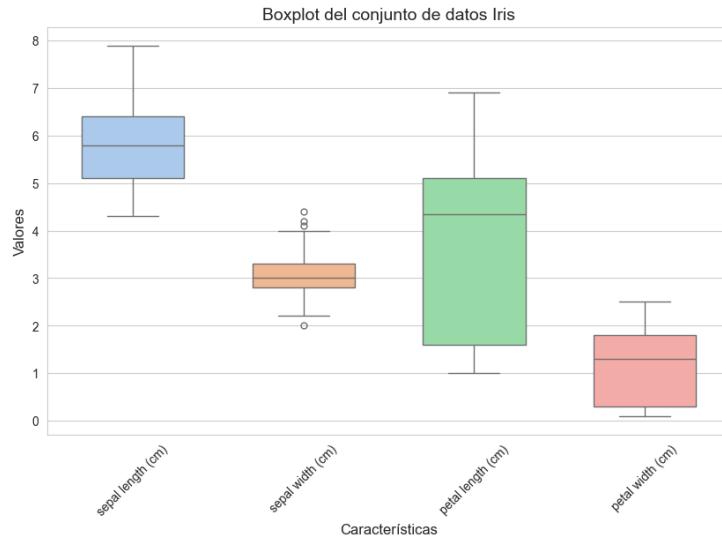


Figure 2: Boxplot del conjunto de datos Iris mostrando las cuatro características principales y los valores atípicos.

El diagrama de cajas del conjunto de datos Iris muestra la distribución de las cuatro características principales: longitud y anchura del sépalo, y longitud y anchura del pétalo. Cada caja representa el rango intercuartil, es decir, el intervalo donde se encuentran el 50% central de los datos, con una línea en el interior que marca la mediana. Las líneas que se extienden desde las cajas muestran los valores que no se consideran atípicos, mientras que los puntos fuera de estas líneas representan valores extremos o atípicos. En este caso, los valores extremos solo están presentes en la anchura del sépalo, donde se observan algunos puntos fuera del rango esperado.

La longitud del pétalo es la característica con el rango más amplio, con valores que van desde aproximadamente 1 hasta casi 7 cm, mientras que la anchura del pétalo presenta el rango más pequeño, de 0 a 2.5 cm. Por otro lado, la longitud del sépalo y la anchura del sépalo tienen distribuciones más restringidas, pero la última incluye varios valores atípicos por debajo del rango principal. Este tipo de gráfico es útil para comparar rápidamente la dispersión, la tendencia central y la presencia de valores extremos entre las diferentes características.

Para preparar los datos de cara al entrenamiento del modelo, se realizaron los siguientes pasos de preprocesamiento:

1. **División del conjunto de datos:** Se separaron las 150 muestras en conjuntos de entrenamiento, validación y prueba, asegurando una representación equilibrada de las tres especies en ambos conjuntos.
2. **Codificación One-Hot:** Las etiquetas de las especies se transformaron mediante codificación one-hot, convirtiendo las etiquetas categóricas en vectores binarios. Este formato es esencial para utilizar la entropía cruzada como función de pérdida durante el entrenamiento, facilitando la optimización del modelo en tareas de clasificación multiclas.

4 Métodos y Detalles de Implementación

En esta entrega, se implementó una red neuronal diseñada para ofrecer flexibilidad en su configuración, permitiendo ajustar dinámicamente el número de capas y neuronas en cada capa. Este enfoque parametrizable facilita la exploración de diferentes arquitecturas y optimizaciones para adaptarse a las necesidades del problema y los datos disponibles.

Además, se introdujo el uso de **lotes** durante el entrenamiento, dividiendo el conjunto de datos en pequeños subconjuntos procesados de manera iterativa. Esta estrategia ofrece varias ventajas significativas:

- **Mejor uso de la memoria:** Al procesar solo un subconjunto de datos a la vez, se reduce la cantidad de memoria necesaria, lo que resulta particularmente útil en problemas con grandes volúmenes de datos.
- **Mayor estabilidad en la optimización:** La actualización de los pesos basada en un lote promedio reduce la variabilidad inherente al procesar ejemplos individuales, proporcionando un equilibrio entre la estabilidad del descenso por gradiente completo y la rapidez del descenso por gradiente estocástico.

Las funciones de activación empleadas incluyen **ReLU**, que introduce no linealidad en las capas ocultas, y **Softmax** en la capa de salida para calcular las probabilidades normalizadas en problemas de clasificación multiclas.

Esta mejora en la flexibilidad, junto con la incorporación de lotes, representa un avance significativo frente a versiones anteriores, ofreciendo una base robusta y adaptable para realizar experimentos y evaluar cómo las configuraciones arquitectónicas y técnicas influyen en el rendimiento del modelo.

4.1 Funciones de Activación

- **ReLU (Rectified Linear Unit):** ReLU es una función de activación ampliamente utilizada, definida matemáticamente como:

$$f(x) = \max(0, x)$$

Es ideal para este proyecto debido a su capacidad para introducir no linearidad, lo cual permite que la red neuronal capture relaciones complejas en los datos. Además, su simplicidad computacional es un beneficio clave, ya que solo requiere una operación comparativa para determinar la salida. Esto lo hace especialmente útil para las capas ocultas, donde se necesitan funciones rápidas y efectivas. Sin embargo, ReLU no está exenta de limitaciones. Una posible complicación es el fenómeno conocido como *neuronas moribundas*, que ocurre cuando las salidas negativas se fijan constantemente en cero, impidiendo que las neuronas contribuyan al aprendizaje. En este proyecto, hemos implementado estrategias para mitigar este problema, como el ajuste cuidadoso de los hiperparámetros y la inicialización adecuada de los pesos.

- **Softmax:** Softmax es una función de activación utilizada en la capa de salida para problemas de clasificación multiclas. Matemáticamente, se define como:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

donde x_i representa la salida de la red para la clase i . Esta función transforma las salidas en una distribución de probabilidad, lo que es fundamental para nuestro proyecto, ya que permite interpretar las predicciones como probabilidades asignadas a cada clase. Este enfoque facilita la evaluación del modelo, al comparar directamente las predicciones con las etiquetas reales.

Una de las principales ventajas de Softmax es su capacidad para amplificar las diferencias entre clases probables e improbables, haciendo más evidente cuál es la clase predicha. Sin embargo, también puede presentar problemas si las salidas de la red tienen valores extremos, ya que esto puede conducir a distribuciones altamente sesgadas.

4.2 Propagación hacia Adelante

La propagación hacia adelante es el proceso mediante el cual los datos de entrada pasan por la red neuronal para generar predicciones. Este mecanismo constituye la base del aprendizaje supervisado, ya que permite al modelo calcular las salidas en función de los pesos actuales y los datos de entrada.

En cada capa de la red, los datos se transforman siguiendo tres pasos fundamentales:

1. **Transformación Lineal:** Se realiza una combinación lineal de las entradas mediante la ecuación $Z = W \cdot X + b$, donde W representan los pesos de la capa, X las entradas y b el sesgo. Esta operación proyecta las características de entrada a un nuevo espacio dimensional definido por los pesos.
2. **Aplicación de Función de Activación:** Los valores calculados en Z se transforman utilizando una función de activación. En las capas ocultas, se emplea ReLU para introducir no linealidad, permitiendo al modelo aprender patrones complejos. En la capa de salida, se aplica Softmax para convertir los valores en una distribución de probabilidad que representa la predicción del modelo.
3. **Producción de Salida:** La última capa de la red devuelve una salida interpretada como probabilidades asignadas a cada clase. Estas probabilidades permiten determinar la predicción más probable según el modelo.

Este proceso refleja la capacidad de las redes neuronales para modelar relaciones complejas y no lineales entre las características de entrada y las etiquetas de salida. En el contexto de nuestro proyecto, la propagación hacia adelante es crucial para evaluar el rendimiento del modelo en la tarea de clasificación multiclas.

4.3 Propagación hacia Atrás

La propagación hacia atrás, conocida como *backpropagation*, es un algoritmo esencial en el entrenamiento de redes neuronales que permite ajustar los parámetros del modelo (pesos y sesgos) para minimizar el error en las predicciones.

El objetivo principal de la propagación hacia atrás es calcular los gradientes de la función de pérdida con respecto a cada uno de los parámetros del modelo. Estos gradientes se utilizan para actualizar los pesos y sesgos de la red mediante un algoritmo de optimización, como el SGD o el Adam en nuestro caso. La propagación hacia atrás se basa en el **principio de la cadena**, una regla matemática que permite descomponer las derivadas de funciones compuestas, lo que facilita el cálculo eficiente de los gradientes en redes con múltiples capas. [2]

El proceso de *backpropagation* consta de las siguientes etapas:

- **Cálculo del error en la capa de salida:** Se mide la discrepancia entre la salida del modelo (predicción) y la etiqueta real, utilizando, nuestro caso, la función de pérdida de entropía cruzada. Este error proporciona la base para retropropagar las señales hacia las capas anteriores.

$$dz^{(L)} = a^{(L)} - y$$

donde:

- $dz^{(L)}$: Error en la capa de salida.
- $a^{(L)}$: Activación (salida) de la última capa L .
- y : Etiqueta real.

Los gradientes para los pesos y sesgos de la capa de salida son:

$$\begin{aligned} dW^{(L)} &= \frac{1}{m} \cdot a^{(L-1)^T} \cdot dz^{(L)} \\ db^{(L)} &= \frac{1}{m} \sum dz^{(L)} \end{aligned}$$

- **Propagación de los errores hacia atrás:** Utilizando el principio de la cadena, se calcula cómo las neuronas en capas anteriores contribuyen al error total. Este cálculo se realiza de manera recursiva, capa por capa, desde la salida hasta la entrada.

$$\begin{aligned} da^{(l)} &= dz^{(l+1)} \cdot W^{(l+1)^T} \\ dz^{(l)} &= da^{(l)} \times RELU'(z^{(l)}) \end{aligned}$$

Los gradientes para los pesos y sesgos de la capa l son:

$$dW^{(l)} = \frac{1}{m} \cdot a^{(l-1)^T} \cdot dz^{(l)}$$

$$db^{(l)} = \frac{1}{m} \sum dz^{(l)}$$

- **Actualización de parámetros:** Los gradientes calculados se emplean para ajustar los pesos y sesgos de cada capa. Este paso asegura que el modelo mejore su rendimiento al reducir la función de pérdida en cada iteración.

$$\begin{aligned} W^{(l)} &:= W^{(l)} - \alpha \cdot dW^{(l)} \\ b^{(l)} &:= b^{(l)} - \alpha \cdot db^{(l)} \end{aligned}$$

donde:

- α : Tasa de aprendizaje.
- $dW^{(l)}$: Gradiente de los pesos en la capa l .
- $db^{(l)}$: Gradiente de los sesgos en la capa l .

La propagación hacia atrás es particularmente poderosa debido a su capacidad para manejar redes profundas con múltiples capas no lineales. Sin embargo, también presenta desafíos, como el *problema del gradiente desvanecido*, que puede dificultar el aprendizaje en redes muy profundas. En este proyecto hemos implementado funciones de activación como ReLU para mitigar estos problemas, al mantener gradientes más estables durante el entrenamiento.

4.4 Función de Costo

La función de costo es un componente esencial en el entrenamiento de redes neuronales. Su propósito principal es cuantificar la discrepancia entre las predicciones del modelo y las etiquetas reales, proporcionando una medida numérica del error cometido por el modelo en sus estimaciones. Esta cuantificación es fundamental, ya que guía el proceso de ajuste de los parámetros internos de la red (pesos y sesgos) para mejorar su precisión y capacidad de generalización.

La función de costo utilizada en este modelo es la **entropía cruzada**, una métrica común en tareas de clasificación que mide la distancia entre las predicciones del modelo y las etiquetas reales. La entropía cruzada ayuda a optimizar el modelo penalizando grandes diferencias entre la probabilidad predicha por el modelo y la probabilidad real (es decir, si la clase es correcta o no).

Esta función penaliza con mayor severidad las predicciones que asignan alta probabilidad a clases incorrectas, incentivando al modelo a ajustar sus parámetros para aumentar la probabilidad asignada a las clases verdaderas. Durante el entrenamiento, el objetivo es minimizar esta función de costo, lo que se traduce en una mejora en la precisión del modelo. [3]

La función de costo se define como:

$$\text{cost} = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^C y_{i,j} \log(a_{i,j} + \epsilon)$$

donde:

- m es el número de ejemplos en el conjunto de datos.
- C es el número de clases.
- $y_{i,j}$ es la etiqueta verdadera en formato *one-hot encoding*, que toma el valor de 1 si el ejemplo i pertenece a la clase j , y 0 en caso contrario.
- $a_{i,j}$ es la probabilidad predicha por el modelo de que el ejemplo i pertenezca a la clase j .
- ϵ es un valor muy pequeño (e.g., 10^{-8}) añadido para evitar problemas numéricos al calcular el logaritmo de 0.

4.5 Optimizadores

En redes neuronales, el objetivo fundamental del proceso de optimización es ajustar los parámetros de la red (pesos y sesgos) de manera que se minimice la función de costo, una medida que refleja la discrepancia entre las predicciones del modelo y los valores reales. Este ajuste de parámetros permite a la red aprender patrones en los datos y mejorar su precisión en la predicción.

4.5.1 Descenso por gradiente estocástico

El algoritmo de actualización de parámetros con descenso de gradiente es un método utilizado en redes neuronales para ajustar los pesos y sesgos de cada capa en función de los gradientes de la función de costo. En cada iteración, el algoritmo recorre cada capa de la red y modifica sus parámetros en la dirección opuesta al gradiente, multiplicado por una tasa de aprendizaje. Esta tasa controla el tamaño de los pasos de ajuste: una tasa alta permite cambios grandes y rápidos, mientras que una tasa baja permite cambios más precisos pero más lentos. Este proceso reduce gradualmente el error de predicción del modelo, permitiendo que la red aprenda patrones en los datos de entrenamiento y, así, mejore su capacidad de hacer predicciones precisas.

Algorithm 1 Actualización de parámetros con descenso de gradiente

Require: `parametros, gradientes, α`
 $L \leftarrow$ Número de capas en la red neuronal
for $l = 1$ to L **do**
 $W^{[l]} \leftarrow W^{[l]} - \alpha \cdot dW^{[l]}$
 $b^{[l]} \leftarrow b^{[l]} - \alpha \cdot db^{[l]}$
end for
return `parametros`

4.5.2 Adam

El algoritmo de actualización de parámetros con el método Adam es una técnica avanzada para optimizar redes neuronales, que combina las ideas de momentum y ajuste adaptativo de la tasa de aprendizaje. En cada iteración, Adam utiliza dos estimaciones de momentos (promedio de gradientes y promedio de gradientes al cuadrado) que se ajustan para corregir el sesgo en sus valores iniciales. Posteriormente, estos momentos se emplean para actualizar los pesos y sesgos de cada capa. Los hiperparámetros β_1 y β_2 determinan la contribución de los gradientes previos en los promedios móviles, mientras que ϵ evita divisiones por cero durante la actualización. Esta combinación permite a Adam adaptarse a la escala de los gradientes, lo que facilita una convergencia más estable y eficiente.

Algorithm 2 Optimización con el método Adam

Require: *parametros, gradientes, t, α, β₁, β₂, ε*

```
vdWt ← 0  
vdbt ← 0  
sdWt ← 0  
sdbt ← 0  
1: for l do  
    vdWt ← β1 · vdWt + (1 - β1) · dWt  
    vdbt ← β1 · vdbt + (1 - β1) · dbt  
     $\hat{v}_{dW_t} \leftarrow \frac{v_{dW_t}}{1-\beta_1^t}$   
     $\hat{v}_{db_t} \leftarrow \frac{v_{db_t}}{1-\beta_1^t}$   
    sdWt ← β2 · sdWt + (1 - β2) · (dWt)2  
    sdbt ← β2 · sdbt + (1 - β2) · (dbt)2  
     $\hat{s}_{dW_t} \leftarrow \frac{s_{dW_t}}{1-\beta_2^t}$   
     $\hat{s}_{db_t} \leftarrow \frac{s_{db_t}}{1-\beta_2^t}$   
    Wt ← Wt - α ·  $\frac{\hat{v}_{dW_t}}{\sqrt{\hat{s}_{dW_t}}+\epsilon}$   
    bt ← bt - α ·  $\frac{\hat{v}_{db_t}}{\sqrt{\hat{s}_{db_t}}+\epsilon}$ 
```

2: **end for**

3: **return** *parameters*

4.6 Predicción

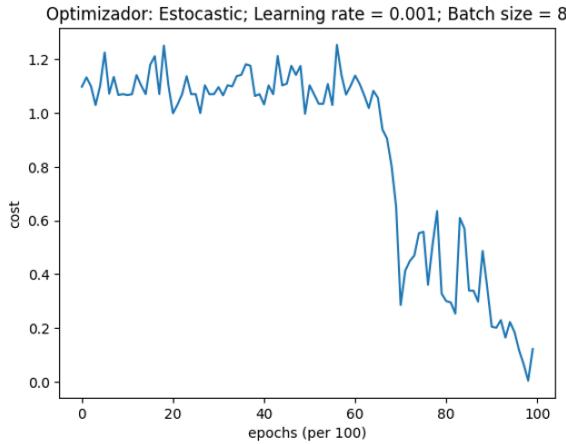
La función de predicción en una red neuronal toma los datos de entrada y utiliza los parámetros entrenados (pesos y sesgos) para generar las predicciones de clase. Este proceso permite aplicar el modelo a datos nuevos y evaluar su rendimiento. Los pasos principales para realizar una predicción son:

1. Propagación hacia adelante: A partir de los datos de entrada X, se realiza una propagación hacia adelante utilizando los parámetros entrenados de la red. Esto genera una salida de la última capa, que consiste en las probabilidades asignadas a cada clase.
2. Conversión de probabilidades a etiquetas de clase: La salida de la última capa contiene las probabilidades predichas para cada clase. Para obtener la clase predicha, se selecciona el índice de la clase con la mayor probabilidad.

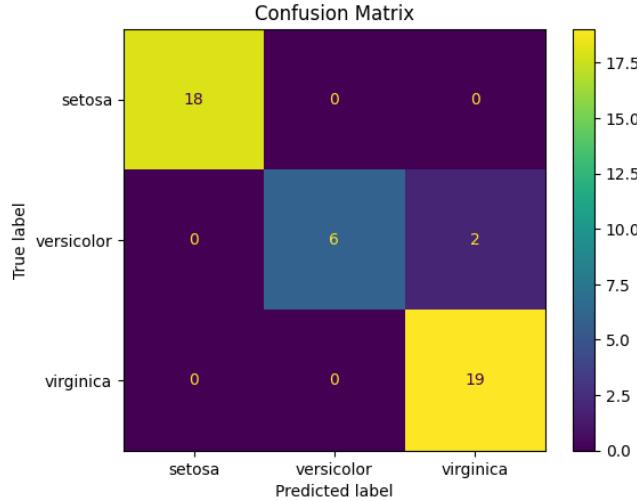
5 Experimentos y Resultados

Para evaluar el rendimiento de la red neuronal desarrollada, realizamos una serie de experimentos utilizando el conjunto de datos Iris. Estas pruebas se llevaron a cabo con un conjunto reducido del total, dividiendo las muestras para validación y posteriormente para testeo. Entrenamos el modelo variando diferentes tasas de aprendizaje, estructuras de red neuronal con distinto número de capas y neuronas y número de lotes. Observamos el comportamiento de las funciones de costo a lo largo de los epochs, exactitud de validación y seleccionamos el mejor conjunto de hiperparámetros para cada optimizador, mostrando los resultados obtenidos con las configuraciones que mejor rendimiento ofrecieron.

5.1 Optimizador Estocástico



La gráfica muestra cómo la función de costo evoluciona durante el entrenamiento de un modelo utilizando un optimizador estocástico con un tamaño de lote de 8 y una tasa de aprendizaje de 0.001. Al principio (épocas 0-60), el costo fluctúa debido a la naturaleza estocástica del optimizador, permaneciendo en un rango elevado. A partir de la época 60, el costo disminuye de manera pronunciada, indicando que el modelo está aprendiendo efectivamente. En las últimas épocas (80-100), el costo se estabiliza en valores bajos con pequeñas fluctuaciones, reflejando un ajuste fino del modelo y un progreso positivo en la minimización del error. Con este conjunto de hiperparámetros se consiguió una exactitud de 100% en validación.



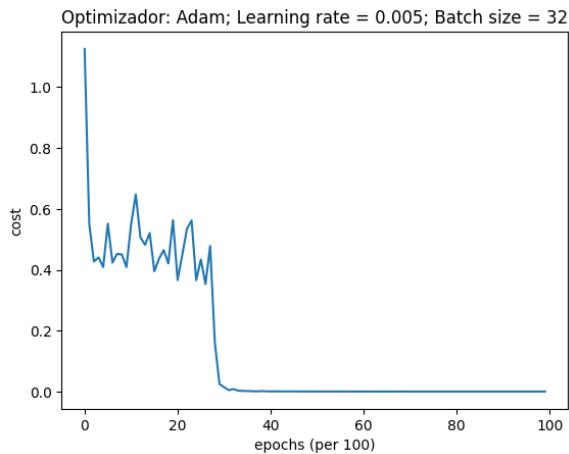
La matriz de confusión muestra que el modelo tiene un desempeño excelente en la clasificación de la clase "setosa", ya que todas las 18 instancias fueron clasificadas correctamente. Esto sugiere que las características de esta clase son muy distintivas, lo que permite al modelo identificarla sin errores.

En el caso de la clase "versicolor", de las 8 instancias totales, 6 fueron clasificadas correctamente, mientras que 2 fueron erróneamente clasificadas como "virginica". Esto indica que existe cierta similitud entre las características de estas dos clases, lo que genera confusión en el modelo. Esta es la única fuente de error en la clasificación general.

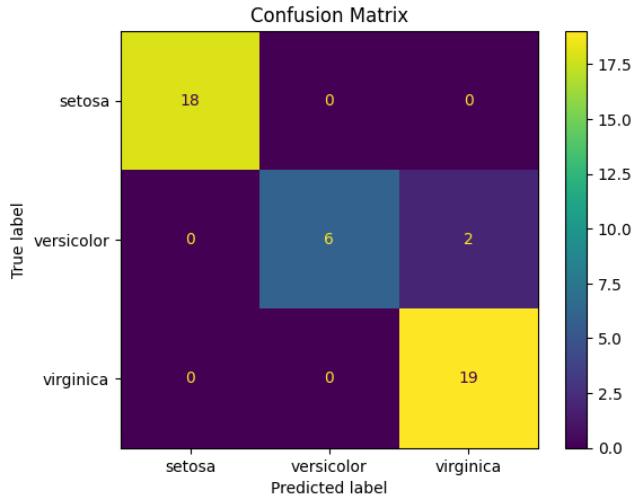
Para la clase "virginica", el modelo logró clasificar correctamente todas las 19 instancias, reflejando un alto grado de precisión. Esto sugiere que el modelo puede capturar las características principales de esta clase sin dificultad.

En términos generales, el modelo alcanzó una exactitud del 95.5%, lo que demuestra un desempeño sólido y efectivo.

5.2 Optimizador Adam



La gráfica muestra cómo la función de costo evoluciona durante el entrenamiento de un modelo utilizando el optimizador Adam, con una tasa de aprendizaje de 0.005 y un tamaño de batch de 32. En las primeras épocas (0-20), el costo disminuye de forma irregular, reflejando las fluctuaciones inherentes al optimizador y la inicialización del modelo. Entre las épocas 20 y 40, se observa una disminución significativa y consistente del costo, lo que indica un aprendizaje efectivo del modelo. Finalmente, a partir de la época 40, el costo se estabiliza en valores cercanos a 0, con fluctuaciones mínimas, lo que sugiere que el modelo ha alcanzado un ajuste óptimo y está minimizando eficazmente el error. Este comportamiento es indicativo de un entrenamiento exitoso y un buen ajuste de los hiperparámetros seleccionados que lleva a tener una exactitud del 95,24% en validación.



La matriz de confusión refleja un desempeño sobresaliente del modelo al clasificar las diferentes clases del conjunto de prueba.

En la clase "setosa", las 18 instancias fueron clasificadas correctamente, lo que indica que esta clase posee características muy distintivas que el modelo puede identificar sin errores.

Para la clase "versicolor", de las 8 instancias totales, 6 fueron clasificadas correctamente, mientras que 2 se confundieron con la clase "virginica". Esto evidencia cierta superposición en las características entre estas dos clases, lo que provoca confusión en el modelo.

Por otro lado, en la clase "virginica", las 19 instancias fueron clasificadas correctamente, reflejando un alto nivel de precisión y la capacidad del modelo para capturar las características representativas de esta clase.

En general, el modelo alcanzó una exactitud del 95.56%, lo que demuestra su efectividad y capacidad para clasificar correctamente la mayoría de las instancias del conjunto de prueba.

6 Conclusiones

En conclusión, hemos implementado con éxito una red neuronal multicapa para la clasificación de datos utilizando el conjunto de datos Iris. La red emplea funciones de activación ReLU en las capas ocultas y Softmax en la capa de salida. Durante el desarrollo, incorporamos el uso de lotes (batches), lo que optimizó el entrenamiento al procesar subconjuntos de datos en cada iteración. Además, modularizamos el código para garantizar una estructura clara, flexible y reutilizable, facilitando su mantenimiento y escalabilidad.

Para optimizar el modelo, experimentamos con dos métodos: Descenso por Gradiente Estocástico y Adam. Ambos demostraron ser efectivos, alcanzando una precisión cercana al 95.5% en el conjunto de prueba, aunque Adam presentó una convergencia más rápida y estable debido a su ajuste adaptativo de la tasa de aprendizaje. La visualización de la curva de costo y la matriz de confusión valida que el modelo clasifica correctamente la mayoría de las muestras y converge de manera adecuada.

7 Trabajo Futuro

Como trabajo futuro, se podría evaluar el modelo en conjuntos de datos más complejos y con más clases para probar su capacidad de generalización. También sería útil explorar arquitecturas de red más profundas y técnicas avanzadas para reducir la confusión entre clases similares, como ajustes más precisos de hiperparámetros, regularización o aumento de datos. Además, integrar métodos de validación cruzada más exhaustivos y probar otros optimizadores como RMSprop o AdaGrad permitiría analizar su impacto en la velocidad de convergencia y precisión, fortaleciendo el modelo para aplicaciones más avanzadas.

8 Bibliografía

- Christopher M. Bishop y Hugh Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024.
- Iris Dataset, Wikipedia. *Iris flower data set*
- Ian Goodfellow, Yoshua Bengio y Aaron Courville, *Deep Learning*
- Entropía Cruzada, *Cross Entropy Loss*