Motor de Redes Neuronales

Luis Perera Pérez
Carlos Francisco Suárez Sauca
Nicolás Trujillo Estévez

Curso 2024/25

Este trabajo presenta el desarrollo de un motor de redes neuronales que implementa diferentes funciones de activación y métodos de optimización. Se utiliza el conjunto de datos Iris para clasificar tres especies de flores basadas en cuatro características. Se exploran funciones de activación como sigmoide, ReLU, tanh, Leaky ReLU, ELU, Swish y GELU, y se implementan métodos de optimización incluyendo Adam, Momentum, Nesterov, Adagrad y RMSProp. Los resultados demuestran la efectividad de cada función de activación y método de optimización en términos de precisión y pérdida durante el entrenamiento.

1 Introducción

Las redes neuronales han transformado el campo del aprendizaje automático, permitiendo resolver problemas complejos en diversas áreas, como visión por computadora y procesamiento del lenguaje natural. Este proyecto tiene como objetivo construir un motor de redes neuronales capaz de clasificar datos utilizando múltiples funciones de activación y métodos de optimización. El conjunto de datos Iris se utiliza como caso de estudio, proporcionando una base sólida para evaluar el rendimiento de diferentes configuraciones.

2 Métodos y Conjuntos de Datos

2.1 Conjunto de Datos

El conjunto de datos Iris contiene 150 muestras de flores, cada una con cuatro características (longitud y ancho de sépalo y pétalo) y una etiqueta que indica

la especie. Las etiquetas se convierten a valores numéricos para facilitar el procesamiento en la red.

2.2 Preprocesamiento

Se normalizan las características para mejorar la convergencia del modelo. La normalización se realiza restando la media y dividiendo por la desviación estándar. Las etiquetas se convierten en vectores one-hot para ser compatibles con la función de pérdida.

2.3 Funciones de Activación

Se implementan las siguientes funciones de activación:

- **Sigmoid**: Utilizada en la capa de salida para problemas de clasificación binaria, aunque también es compatible con otros problemas.
- **ReLU (Rectified Linear Unit)**: Común en capas ocultas debido a su eficiencia y su capacidad para mitigar problemas de desvanecimiento del gradiente.
- **Tanh**: Con rango de salida entre -1 y 1, útil para centrar los datos y evitar el desplazamiento de los valores de activación.
- **Leaky ReLU, ELU, Swish y GELU**: Variantes que abordan problemas de desvanecimiento del gradiente en redes profundas.

2.4 Métodos de Optimización

Se implementan varios métodos de optimización para actualizar los pesos y sesgos de la red, mejorando la convergencia:

- **Adam**: Combina las ventajas de los métodos de Momentum y RMSProp, ajustando la tasa de aprendizaje adaptativamente y con corrección de sesgo.
- **Momentum**: Ayuda a acelerar el descenso en la dirección relevante al acumular un término de velocidad de gradiente.
- **Nesterov**: Una variante de Momentum que realiza un look-ahead en la actualización de parámetros, aumentando la precisión de convergencia.
- **Adagrad y RMSProp**: Adaptan la tasa de aprendizaje para cada parámetro, lo que es útil en configuraciones donde algunos parámetros requieren pasos de actualización diferentes.

3 Detalles de Implementación

El motor de redes neuronales está estructurado en varias funciones que implementan la propagación hacia adelante y hacia atrás, junto con el cálculo de la pérdida. Las capas de la red se definen mediante matrices de pesos y sesgos, y se realiza la actualización de estos parámetros en cada época del entrenamiento.

3.1 Propagación hacia Adelante

Las funciones de propagación hacia adelante calculan la salida de la red en función de la entrada, aplicando la función de activación correspondiente en cada capa. La capa de salida utiliza la función `softmax` para obtener probabilidades de clasificación multiclase.

3.2 Retropropagación

Durante la retropropagación, se calculan los gradientes de los pesos y sesgos utilizando la derivada de la función de activación y la función de pérdida. Los gradientes calculados se utilizan para ajustar los parámetros de la red, minimizando la función de pérdida en cada iteración. La actualización de los pesos y sesgos depende del método de optimización seleccionado en el entrenamiento (por ejemplo, Adam, Momentum, etc.).

3.3 Entrenamiento del Modelo

El entrenamiento se realiza mediante la función `train_model`, que recibe el conjunto de entrenamiento y el conjunto de prueba, la función de activación y el método de optimización. En cada época, se calcula la precisión y la pérdida en el conjunto de entrenamiento y de prueba, almacenándose para su análisis posterior.

Previo a esto se realiza un proceso opcional de validación que devuelve de manera similar resultados sobre la precisión del modelo para diferentes combinaciones de hiperparámetros, a saber el "learning rate" y el número de neuronas de la capa oculta. Estos resultados son muy interesantes ya que durante el transcurso de este trabajo hemos experimentado problemas con valores anómalos de la precisión para las distintas combinaciones de métodos de optimización y funciones de activación que en la teoría deberían de dar valores muy buenos mientras que, en la práctica daban valores muy bajos. Este problema se debía al valor del learning rate que era demasiado bajo, pues si este baja de la décima (<0.1) empieza a dar valores muy heterogéneos y en

su mayoría malos; y también se debía al número de neuronas de la capa oculta ya que si este disminuye de 30, también disminuye la precision anómalamente. Es un proceso de validación poco ortodoxo pues printea la precisión para las distintas combinaciones de hiperparámetros, la mayoría devuelven valores correctos por lo que cualquier combinación es válida pero si se descomenta la línea de graficación se puede analizar con qué rapidez (número de epochs) se alcanza los valores de precision altos y así uno se puede decidir por una combinación específica e incluso extrapolar un número máximo de epochs necesario dependiendo de cada método de retropropagación. Como tónica general un "learning rate" de 0.2 y más de 30 neuronas en la capa oculta es una combinación sólida para cualquier método.

3.4 Evaluación del Modelo

La función `predict` permite evaluar el modelo usando la función de activación seleccionada en la capa oculta. Devuelve las etiquetas predichas, las cuales se comparan con las etiquetas reales para calcular la precisión del modelo en el conjunto de prueba.

4 Experimentos y Resultados

4.1 Entrenamiento

Se entrenaron redes neuronales con diferentes configuraciones de funciones de activación y métodos de optimización. A continuación, se presentan los resultados de precisión obtenidos en el conjunto de prueba.

5 Entrenamiento

Se entrenaron redes neuronales utilizando distintas configuraciones de funciones de activación y métodos de optimización. Como ejemplo, se incluye la gráfica correspondiente al método de optimización Momentum. Sin embargo, en el código están disponibles todas las gráficas para cada combinación de método de optimización y función de activación utilizada.

5.1 Resultados para Momentum utilizando Sigmoid

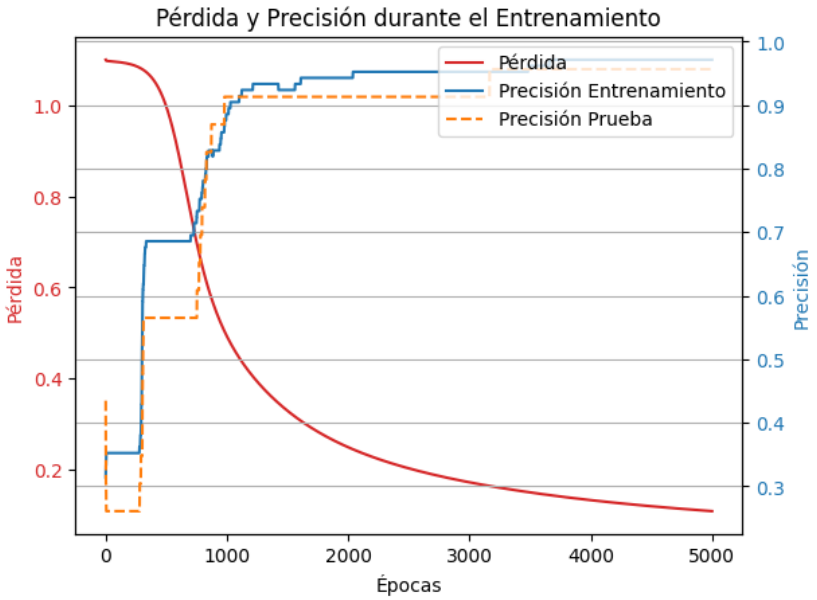

Figure 1: Gráfica de precisión y pérdida durante el entrenamiento para Momentum con función de activación Sigmoid

5.2 Resultados para Momentum utilizando ReLU

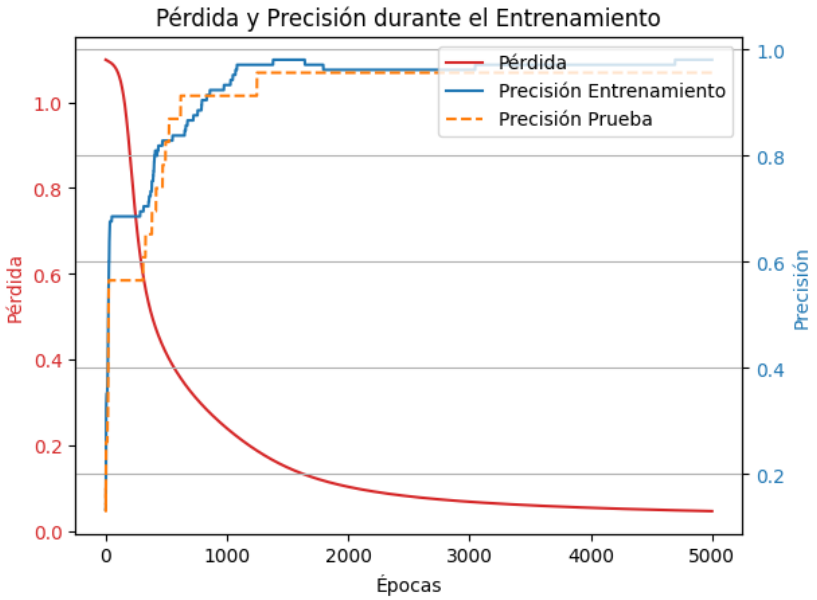

Figure 2: Gráfica de precisión y pérdida durante el entrenamiento para Momentum con función de activación ReLU

5.3 Resultados para Momentum utilizando Leaky ReLU

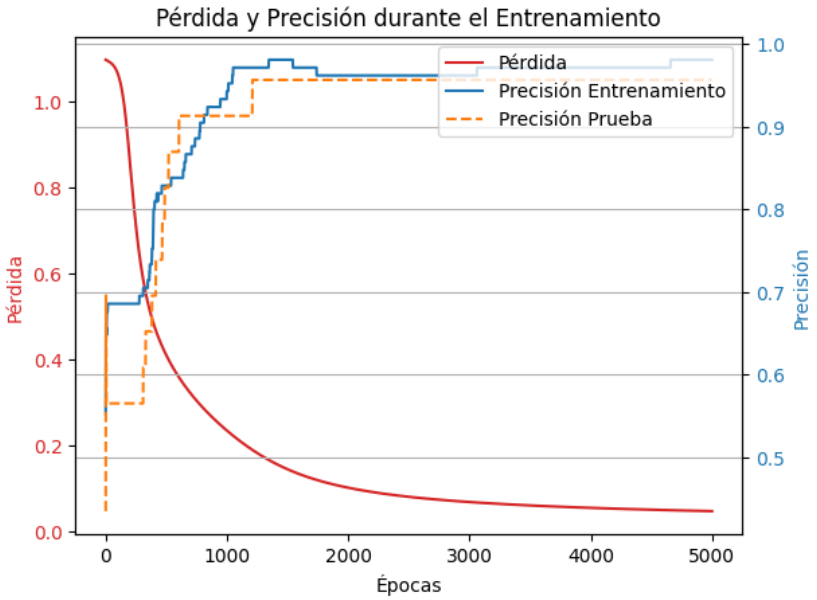

Figure 3: Gráfica de precisión y pérdida durante el entrenamiento para Momentum con función de activación Leaky ReLU

5.4 Resultados para Momentum utilizando Swish

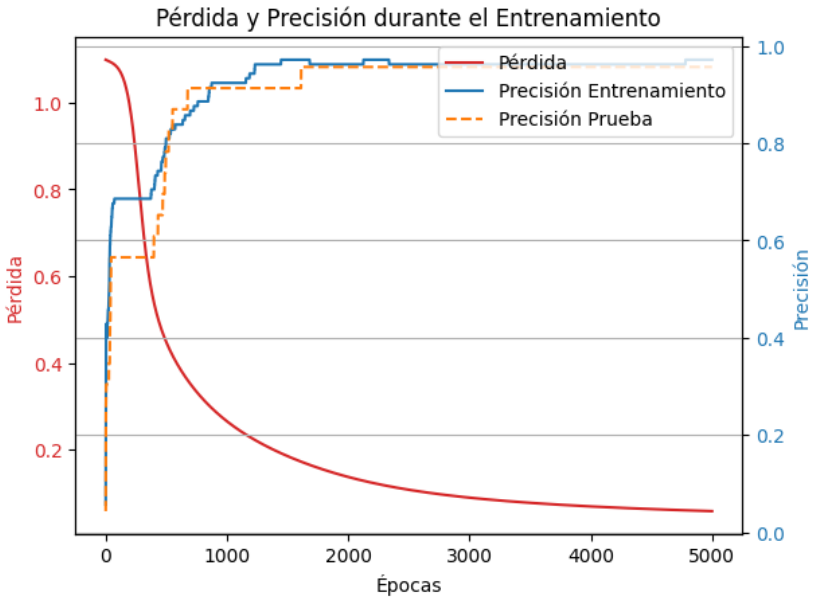

Figure 4: Gráfica de precisión y pérdida durante el entrenamiento para Momentum con función de activación Swish

5.5 Resultados para Momentum utilizando GELU

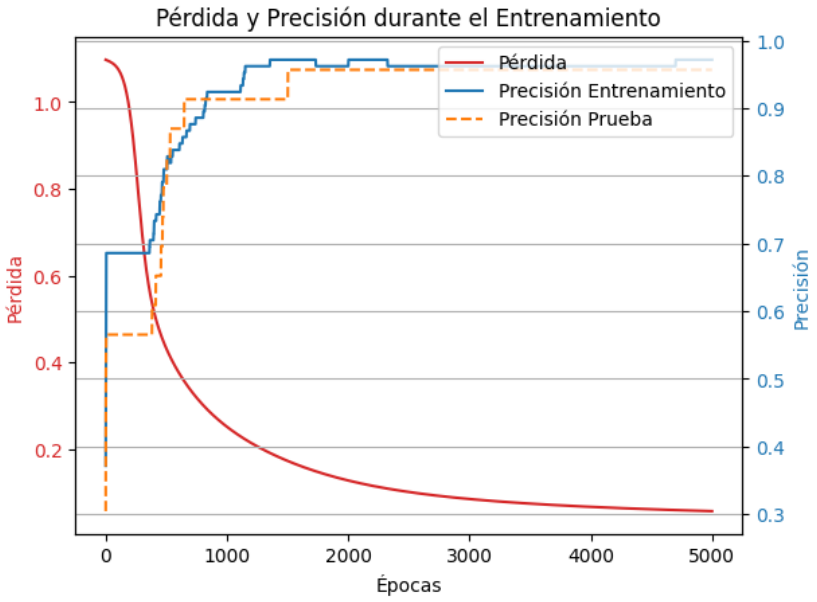

Figure 5: Gráfica de precisión y pérdida durante el entrenamiento para Momentum con función de activación GELU

6 Conclusiones

En conclusión, consideramos que el método de Momentum es el más eficaz y robusto. Este enfoque muestra un rendimiento excelente con todas las funciones de activación y presenta una notable estabilidad, ya que no experimenta grandes variaciones al recalcular el gradiente. Además, su uso de un término de inercia, que incorpora información de gradientes anteriores, contribuye significativamente a su desempeño.

7 Trabajo Futuro

Para futuras investigaciones, se sugiere investigar arquitecturas de redes neuronales avanzadas, como redes neuronales convolucionales (CNN) o recurrentes (RNN), que puedan abordar problemas en diferentes dominios, como procesamiento de imágenes y texto. También se podría explorar el uso de técnicas de regularización para evitar el sobreajuste en modelos más complejos y mejorar la generalización del modelo en datos desconocidos.

8 Repositorio de GitHub

Todo el código fuente, los conjuntos de datos utilizados y la documentación detallada de este proyecto están disponibles en el repositorio de GitHub. Este repositorio proporciona acceso completo a los archivos necesarios para replicar el experimento, junto con instrucciones para su uso.

`https://github.com/LuisPereraPerez/OH_preoyecto`

En el repositorio se incluyen:

- **Código fuente**: Implementación del motor de redes neuronales con diferentes funciones de activación y métodos de optimización.
- **Dataset**: El conjunto de datos Iris utilizado para la clasificación de especies de flores.
- **Documentación**: Archivos y notas explicativas adicionales, que proporcionan una guía para el uso del código y los experimentos realizados.