Búsqueda de Soluciones e Inferencia Bayesiana
Optimización de Pesos en Red Neuronal para Clasificación de Imágenes

8 de abril de 2025

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS
Av. Universidad 1001, Chamípa, Cuernavaca, Morelos México. CP 62210.
Tel: 777 329 7000

Optimización de Pesos en Red Neuronal para Clasificación de Imágenes

Autor: Anibal Medina Cabrera
Materia: Búsqueda de Soluciones e Inferencia Bayesiana

Índice

1. Descripción del proyecto

1.1. Contexto

Este proyecto se enmarca en el campo de la Inteligencia Artificial, específicamente en el aprendizaje automático mediante redes neuronales. La optimización de pesos es fundamental para mejorar el rendimiento de redes neuronales en tareas de clasificación de imágenes. El problema adquiere relevancia científica por su aplicación en sistemas de reconocimiento de patrones y relevancia práctica en aplicaciones como visión por computadora.

1.2. Problema

Dado el conjunto de imágenes MNIST (entrada: matrices 28x28 normalizadas) y sus etiquetas (salida: dígitos 0-9), el problema consiste en encontrar los parámetros óptimos (pesos y sesgos) de una red neuronal de dos capas que minimicen la función de pérdida de entropía cruzada con regularización L2. Formalmente:

$$\text{Minimizar } \mathcal{L}(W_1, b_1, W_2, b_2) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{10} y_{ij}\log(\hat{y}_{ij}) + \lambda(\|W_1\|^2 + \|W_2\|^2)$$

2. Planteamiento del problema

2.1. Definición de estado

Un estado E se define como una 4-tupla de parámetros:

$$E = (W_1 \in R^{10\times 784}, b_1 \in R^{10\times 1}, W_2 \in R^{10\times 10}, b_2 \in R^{10\times 1})$$

donde cada componente representa:

- W_1: Matriz de pesos capa oculta
- b_1: Vector de sesgos capa oculta
- W_2: Matriz de pesos capa de salida
- b_2: Vector de sesgos capa de salida

2.2. Representación para optimización

Los parámetros se aplanan en un vector continuo $\mathbf{v} \in R^{7850+100+10}$ mediante concatenación. Esta representación permite operaciones genéticas:

- Cruce: Recombinación de bloques de parámetros
- Mutación: Perturbaciones gaussianas

2.3. Función de aptitud

La función objetivo a minimizar combina pérdida y regularización:

$$F(E) = \underbrace{-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{10} y_{ij}\log(\hat{y}_{ij})}_{\text{Entropía Cruzada}} + \lambda \underbrace{\left(\|W_1\|_F^2 + \|W_2\|_F^2\right)}_{\text{Regularización L2}}$$

con $\lambda = 0{,}0001$ y $\|\cdot\|_F$ = norma de Frobenius.

Validación de estados:

$$\text{Estado válido} \iff \begin{cases} 1.\ \hat{y}_{ij} \in (0,1)\ \forall i,j \\ 2.\ \|W_k\|_\infty \leq 2\ \forall k \\ 3.\ \text{Precisión} \geq 0{,}5 \end{cases}$$

3. Metodología

3.1. Algoritmo Genético Implementado

El proceso de optimización sigue un esquema evolutivo con las siguientes etapas:

1. **Inicialización:**
   - Generación de 200 individuos (configuraciónes de pesos y sesgos) con parámetros aleatorios
   - Estrategias de inicialización:
     - Capa oculta: Inicialización He para activaciones ReLU

       $$W_1 \sim \mathcal{N}(0, \sqrt{2/784})$$

     - Capa de salida: Inicialización Xavier para softmax

       $$W_2 \sim \mathcal{N}(0, \sqrt{1/10})$$

- $b = \mathcal{N}(0,1) \cdot 0{,}01$, de tamaño 10×1, son los sesgos de cada capa.

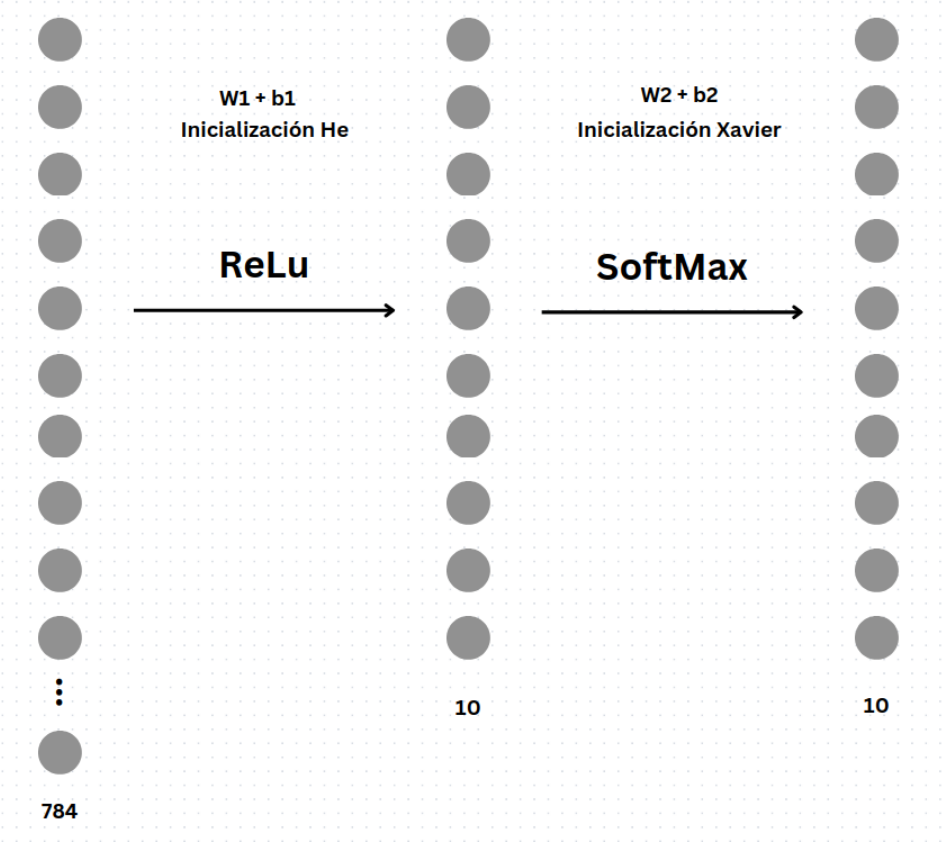

Figura 1: Arquitectura de la red neuronal (784-10-10)

2. **Evaluación:**

   - Cálculo de aptitud mediante función de pérdida:

$$\mathcal{L} = \text{Entropía Cruzada} + 0{,}0001(\|W_1\|^2 + \|W_2\|^2)$$

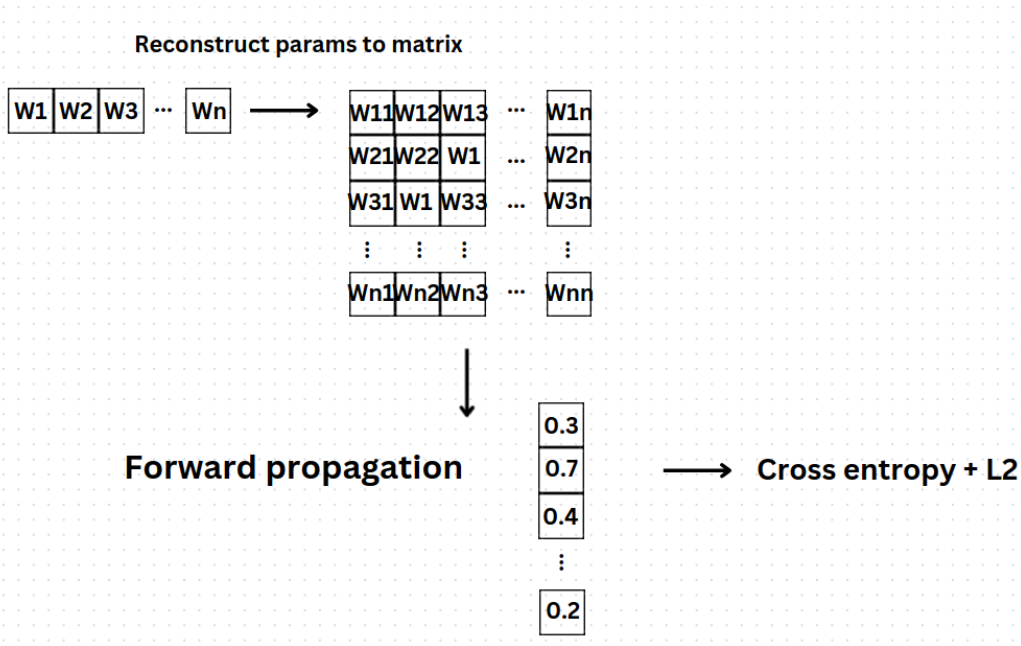

Figura 2: Proceso de evaluación de la población

3. **Selección:**

   - Torneo probabilístico de tamaño 3
   - 20 % de la población seleccionada (40 individuos)

```python
# Emparejar individuos con sus pérdidas
paired_population = list(zip(population, fitness_scores))
max_loss = max(fitness_scores)
probabilities = [(max_loss - score) for _, score in paired_population]  # Invertir la escala
total_probability = sum(probabilities)
probabilities = [p / total_probability for p in probabilities]  # Normalizar

selected_population = []
for _ in range(num_selected):
    tournament = random.choices(paired_population, weights=probabilities, k=3)
    winner = min(tournament, key=lambda x: x[1])  # Seleccionar el de menor pérdida
    selected_population.append(winner[0])  # Agregar solo el individuo

return selected_population
```

Figura 3: Código del proceso de selección por torneo

4. **Cruce:**
    - Recombinación de 3 padres por bloque de parámetros
    - Tamaño de bloque: 10 parámetros
    - Probabilidad de herencia: 33 % por padre

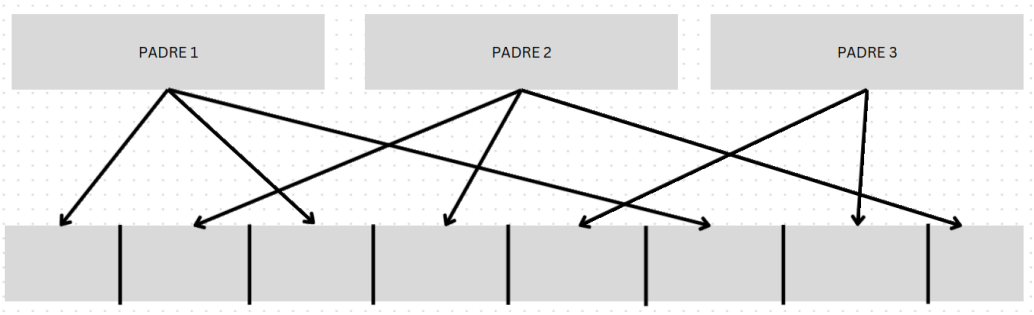

Figura 4: Mecanismo de cruce por bloques neuronales

5. **Mutación:**
    - Probabilidad por gen: $1\,\%$
    - Distribución de mutación: $\Delta w \sim \mathcal{N}(0, 0{,}1)$

```
mutated_individual = individual.copy()
for i in range(len(mutated_individual)):
    if random.random() < mutation_rate:
        mutated_individual[i] += np.random.normal(0, mutation_strength)
return mutated_individual
```

Figura 5: Efecto de la operación de mutación

4. Resultados

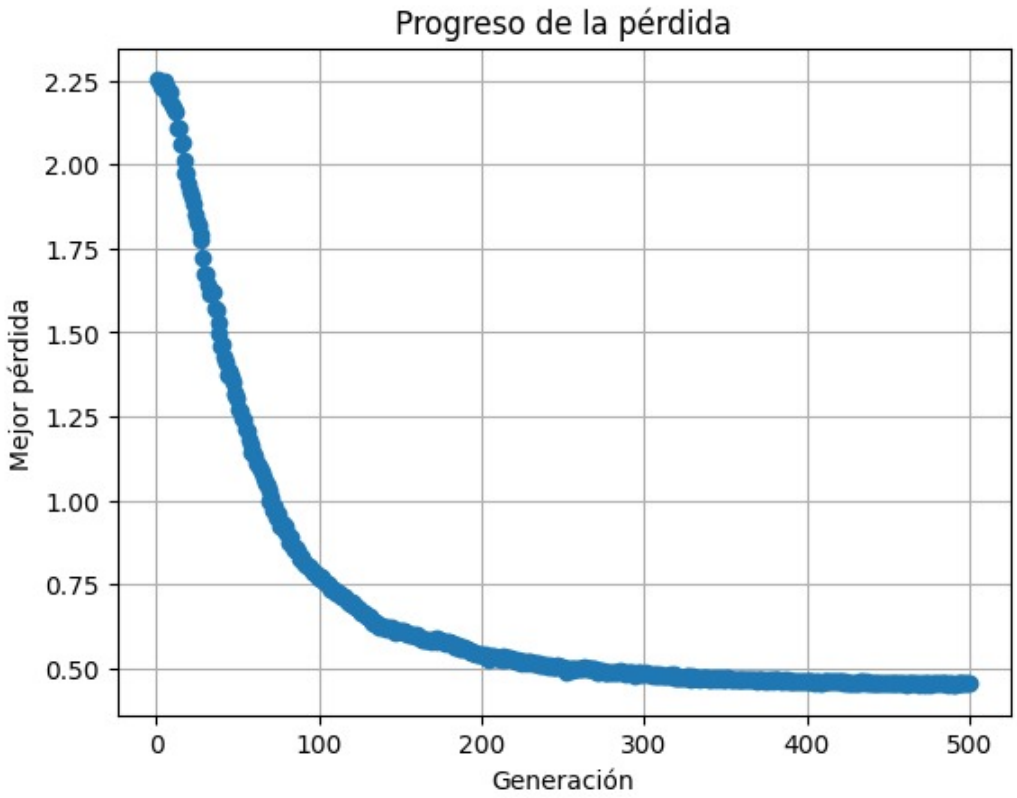

Figura 6: Evolución de la pérdida durante 500 generaciones

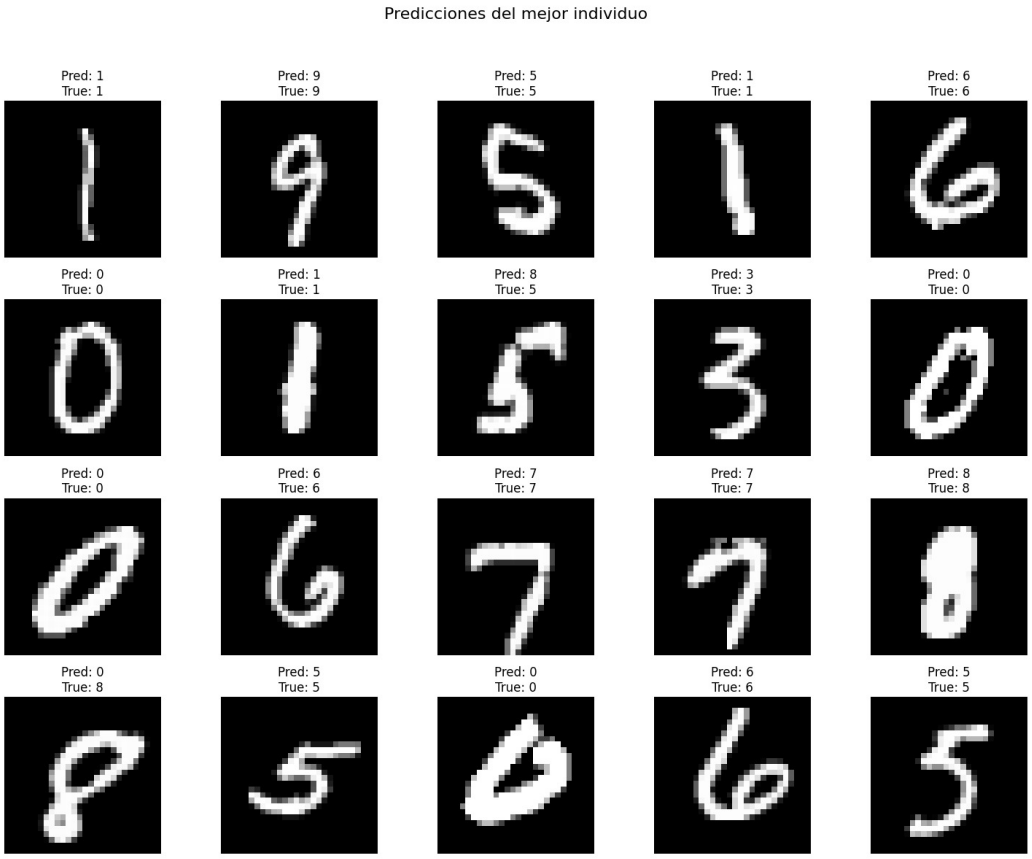

Figura 7: Ejemplos de predicciones en conjunto de prueba

Resultado final:

- Mejor pérdida en entrenamiento: 0.4577
- Exactitud en prueba: 89.73 %
- Tiempo de ejecución: 11h 15m en colab

5. Conclusiones

El algoritmo genético demostró ser efectivo para optimizar pesos en redes pequeñas, alcanzando exactitud normal en MNIST. La regularización L2 evitó sobreajuste. Sin embargo el tiempo de ejecución fue demasiado largo. Como mejora futura, se propone implementar una estructura de red que permita ajustar un modelo más complejo.

6. Bibliografía

- Keras Documentation. *MNIST Dataset*. 2023
- NumPy Documentation. *Array Broadcasting*. 2023
- Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, 1998