RECONOCIMIENTO CON PERCEPTRÓN SIMPLE Y PERCEPTRÓN MULTICAPA

Andrés Torres Ceja  
*Ingeniería en Sistemas Computacionales*

*Computación Flexible “Soft Computing CI”*

*Universidad de Guanajuato DICIS*

*A viernes 10 de octubre de 2025*

Salamanca, Guanajuato, México

a.torresceja@ugto.mx

##### Implementación de Redes Neuronales Artificiales.

Resumen (Abstract) —Este documento presenta el diseño, implementación y análisis comparativo de dos arquitecturas de redes neuronales artificiales: el Perceptrón Simple (Single-Layer Perceptron) y el Perceptrón Multicapa (Multi-Layer Perceptron, MLP). El sistema desarrollado está orientado a la clasificación de patrones binarios de dígitos (0-9) representados en matrices de 5×3 píxeles (15 features) bajo diferentes condiciones de ruido y variaciones de dataset. Se implementó un framework completo que incluye generación de datasets sintéticos con ruido configurable, entrenamiento supervisado, evaluación de desempeño, y visualización avanzada de métricas y fronteras de decisión. Los resultados experimentales demuestran que el MLP supera consistentemente al Perceptrón Simple en precisión (accuracy) y capacidad de generalización, especialmente en presencia de ruido y patrones no linealmente separables.

Términos clave— Redes Neuronales Artificiales, Perceptrón, MLP, Clasificación Supervisada, Backpropagation, Fronteras de Decisión, Python, Inteligencia Computacional, Aprendizaje Automático, Reconocimiento de Patrones.

# Introducción

Las Redes Neuronales Artificiales (ANN’s) constituyen un paradigma fundamental en el aprendizaje automático y la inteligencia artificial computacional. Inspiradas en la estructura neurobiológica del cerebro, las ANN’s son modelos matemáticos capaces de aprender representaciones complejas a partir de datos mediante ajuste iterativo de parámetros. El Perceptrón, propuesto por Rosenblatt (1958), representa el modelo neuronal más elemental y constituye la base conceptual de arquitecturas más complejas. A pesar de sus limitaciones para resolver problemas no linealmente separables, el Perceptrón Simple ofrece ventajas en términos de interpretabilidad y eficiencia computacional.

El Perceptrón Multicapa (MLP), introducido por Rumelhart et al. (1986) junto con el algoritmo de backpropagation, extiende las capacidades del perceptrón simple mediante la incorporación de capas ocultas y funciones de activación no lineales, permitiendo la aproximación de funciones arbitrariamente complejas según el teorema de aproximación universal.

Esta introducción busca cubrir los objetivos formales del proyecto así como definir conceptos de marco teórico común para el correcto entendimiento de la materia y de los resultados propuestos en este articulo científico. El estudio es útil tanto para investigación aplicada (comparación controlada de modelos) como para enseñanza, pues ilustra conceptos centrales: linealidad, capacidad, overfitting, regularización, robustez al ruido, y métodos de evaluación más allá del simple accuracy.

## Objetivo General:

Implementar, entrenar y comparar dos arquitecturas de redes neuronales (Perceptrón Simple y MLP) para la clasificación de patrones binarios de dígitos.

#### **Hitos Específicos a cubrir:**

1. Desarrollar un sistema de generación de datos sintéticos con ruido controlado, es decir desarrollar un pipeline reproducible.
2. Implementar ambas arquitecturas neuronales con parámetros configurables
3. Evaluar el desempeño mediante métricas estándar (accuracy, matriz de confusión, curvas de pérdida)
4. Visualizar y analizar las fronteras de decisión de ambos modelos
5. Realizar análisis matemático de convergencia, gradientes y capacidad del modelo
6. Añadir experimentos de validación cruzada (k-fold), tests estadísticos y análisis de varianza (ANOVA) cuando proceda.
7. Medir calibración y confianza de las salidas probabilísticas (Brier score, entropía).
8. Evaluar técnicas de regularización y optimizadores avanzados (Adam, RMSprop) frente a SGD clásico.
9. Realizar ablation studies: qué componentes del MLP aportan mayor ganancia (capas, ancho, activaciones).
10. Documentar cada experimento con seeds, entorno, y código reproducible.

## Redes Neuronales Artificiales.

Una red neuronal artificial es una función parametrizada compuesta por unidades de procesamiento interconectadas (neuronas artificiales). Formalmente:

donde  es el vector de entrada,  representa los parámetros (pesos y sesgos), y la salida pertenece a .

#### **Perceptrón Simple.**

El perceptrón monocapa (SLP) es la forma más simple de una red neuronal de alimentación directa (feed-forward). Consiste en una única capa de neuronas de salida que están directamente conectadas a la capa de entrada. Cada neurona de salida realiza una suma ponderada de las entradas, a la cual se le aplica una función de activación. La salida binaria de un SLP se usa para clasificar entradas en una de dos categorías, pero solo es capaz de aprender patrones linealmente separables. Un ejemplo clásico es su incapacidad para resolver el problema de la compuerta XOR sin la adición de una capa oculta.

El perceptrón simple es un clasificador lineal binario que computa:

donde:

* es el vector de pesos
* es el sesgo (bias)
* es la función de activación (típicamente función escalón o signo)
* es el vector de características

La frontera de decisión es un hiperplano definido por:

Limitación fundamental: El perceptrón simple solo puede resolver problemas linealmente separables, como demostró Minsky y Papert (1969) en su análisis del problema XOR.

#### **Perceptrón Multicapa (MLP).**

El perceptrón multicapa (MLP) es una red neuronal de alimentación directa que consta de al menos tres capas: una capa de entrada, una o más capas ocultas y una capa de salida. A diferencia del SLP, el MLP utiliza funciones de activación no lineales en sus capas ocultas, como la función sigmoide o la función ReLU, lo que le permite capturar y modelar relaciones no lineales más complejas en los datos. El algoritmo de retropropagación (backpropagation) es el método de entrenamiento más común para los MLP, el cual ajusta los pesos sinápticos de la red para minimizar el error de predicción.

El aprendizaje de un MLP se realiza a través de un proceso iterativo conocido como entrenamiento supervisado. Este proceso se basa en el algoritmo de retropropagación, que tiene dos fases principales:

Propagación hacia adelante (Feedforward): Las entradas se pasan a través de la red, desde la capa de entrada hasta la de salida, calculando la salida de cada neurona.

Retropropagación del error (Backpropagation): El error entre la salida predicha y la salida real se calcula en la capa de salida. Este error se propaga hacia atrás, a través de las capas, para calcular la contribución de cada peso y sesgo al error total. A partir de este cálculo, se ajustan los pesos sinápticos y los sesgos utilizando un algoritmo de optimización, como el descenso de gradiente estocástico (SGD).

Este ciclo se repite hasta que el error de la red se reduce a un nivel aceptable o hasta que se alcanza un número predeterminado de épocas (iteraciones de entrenamiento completas).

El MLP extiende el perceptrón simple mediante capas ocultas:

* Capa de Entrada:
* Capa oculta :
* Capa de salida:

donde:

* es la matriz de pesos de la capa
* es el vector de sesgos
* es una función de activación no lineal (ReLU, tanh, sigmoid)

#### **Funciones de Activación.**

Las funciones de activación son un componente fundamental de las redes neuronales artificiales. Su propósito principal es introducir no linealidad en el modelo, lo cual es crucial para que la red pueda aprender y modelar relaciones complejas en los datos. En cada neurona, la función de activación toma la suma ponderada de las entradas y aplica una transformación matemática para producir la salida de esa neurona. Esta salida se utiliza como entrada para la siguiente capa. En esencia, la función de activación determina si una neurona debe "activarse" o no, basándose en la relevancia de su entrada.

* ReLU (Rectified Linear Unit): Es una de las funciones más populares en la actualidad debido a su simplicidad y eficiencia computacional. La función ReLU es lineal para entradas positivas y cero para entradas negativas.
* Sigmoid: Esta función transforma cualquier valor de entrada en un valor entre 0 y 1. Su forma de "S" la hace útil para problemas de clasificación binaria, ya que la salida puede interpretarse como una probabilidad. Sin embargo, puede sufrir del problema de gradiente desvanecido para valores de entrada muy grandes o muy pequeños.
* Tanh (Tangente Hiperbólica): Similar a la sigmoide, pero su rango de salida es de -1 a 1. Esto la hace más atractiva en algunas aplicaciones, ya que su salida está centrada en cero, lo que puede facilitar la optimización.
* Softmax (capa de salida):

#### **Función de Perdida.**

La función de pérdida de entropía cruzada (Cross-Entropy Loss) es una métrica utilizada en el entrenamiento de modelos de clasificación. Su principal objetivo es cuantificar la diferencia entre la distribución de probabilidad predicha por el modelo y la distribución de probabilidad real (las etiquetas verdaderas). En esencia, mide qué tan "incorrecto" es el modelo.  
A diferencia de otras funciones de pérdida, como el Error Cuadrático Medio (MSE), que miden la distancia geométrica, la entropía cruzada se enfoca en las probabilidades. La intuición detrás de esta función es simple:

* Si el modelo asigna una probabilidad alta a la clase correcta, la pérdida es baja.
* Si el modelo asigna una probabilidad baja a la clase correcta (o una alta a una incorrecta), la pérdida es alta, y el castigo es severo.

Existe la entropía cruzada binaria, pero para este ejercicio se hace uso de una clasificación multiclase, por lo que se empleó la entropía cruzada (cross-entropy):

donde:

* es el número de muestras
* es el número de clases
* es la etiqueta verdadera (one-hot encoded)
* es la probabilidad predicha

#### **Backpropagation y Descenso por Gradiente.**

El algoritmo de backpropagation (retropropagación) es la piedra angular del entrenamiento de redes neuronales, especialmente de los MLP. Su función principal es calcular de forma eficiente los gradientes de la función de pérdida con respecto a todos los parámetros (pesos y sesgos) de la red. Una vez que se tienen estos gradientes, se usa un algoritmo como el descenso de gradiente para actualizar los parámetros y minimizar el error.

El algoritmo de backpropagation calcula eficientemente los gradientes de la función de pérdida respecto a todos los parámetros. El proceso de backpropagation se divide en dos fases principales, que son iteradas durante el entrenamiento:

Forward Pass:

* Inferencia. La información fluye desde la capa de entrada, a través de las capas ocultas, hasta la capa de salida.
* Cálculo de la Salida. Cada neurona calcula su salida aplicando una función de activación a la suma ponderada de sus entradas.
* Cálculo del Error. En la capa de salida, la función de pérdida (como la entropía cruzada) mide la diferencia entre la salida predicha y la salida real. El resultado es un valor de error para esa iteración.

Backward Pass:

* Retropropagación del Error. El error se propaga hacia atrás, desde la capa de salida hasta la capa de entrada. A través de la regla de la cadena del cálculo, se calcula la contribución de cada peso y sesgo al error total. Esto da como resultado los gradientes de la función de pérdida con respecto a cada uno de los parámetros.

Gradientes: El descenso por gradiente es un algoritmo de optimización que utiliza los gradientes calculados en el backpropagation para ajustar los parámetros de la red neuronal. La idea es simple: queremos encontrar el conjunto de pesos y sesgos que minimice la función de pérdida.

Actualización de parámetros (Gradient Descent): El descenso por gradiente actualiza los parámetros moviéndose en la dirección opuesta al gradiente, ya que el gradiente apunta en la dirección de máximo crecimiento de la función. La fórmula de actualización es:

Donde:

* son los nuevos parámetros (pesos o sesgos).
* son los parámetros actuales.
* es la tasa de aprendizaje (*learning rate*), un hiperparámetro crucial que determina el tamaño del paso en cada actualización. Un valor demasiado grande puede hacer que el algoritmo "salte" el mínimo, mientras que uno demasiado pequeño puede hacer que el entrenamiento sea muy lento.
* es el gradiente de la función de pérdida () con respecto a los parámetros ().

donde  es la tasa de aprendizaje (learning rate).

#### **Fronteras de Decisión (Decision Boundaries).**

Las fronteras de decisión son las superficies o límites que un modelo de clasificación aprende para separar las diferentes clases en el espacio de características. En esencia, son los límites que el modelo traza para decidir a qué clase pertenece un nuevo punto de datos. La frontera de decisión separa el espacio de características en regiones correspondientes a diferentes clases. Para un clasificador binario:

* Perceptrón Simple: Un perceptrón simple, que no tiene capas ocultas, solo puede aprender fronteras de decisión lineales. Esto significa que solo puede resolver problemas donde las clases pueden ser separadas por una línea (en 2D), un plano (en 3D), o un hiperplano (en dimensiones superiores). Por esta razón, el SLP no puede resolver problemas como el de la compuerta XOR, que es inherentemente no linealmente separable.
* MLP: La adición de una o más capas ocultas con funciones de activación no lineales (como ReLU o sigmoide) permite a un MLP aprender fronteras de decisión no lineales. Esto le da la capacidad de modelar y clasificar problemas mucho más complejos y realistas, donde las clases no pueden ser separadas por una simple línea recta. La forma y complejidad de estas fronteras no lineales dependen de la arquitectura del MLP, el número de capas y neuronas, y el proceso de entrenamiento.

#### **Conceptos de Generalización.**

* Overfitting: El modelo aprende ruido específico del conjunto de entrenamiento, perdiendo capacidad de generalización.
* Underfitting: El modelo es demasiado simple para capturar la complejidad subyacente de los datos.
* Early Stopping: Técnica de regularización que detiene el entrenamiento cuando la pérdida de validación deja de disminuir.
* Dimensión VC (Vapnik-Chervonenkis): Medida teórica de la capacidad de un modelo, relacionada con el número de muestras que puede clasificar arbitrariamente.

# Metodología

##### Metodología Segmentada Teórica. primera Parte.

El sistema implementa una arquitectura modular que se construye a forma de framework, para toda ejecución se debe generar un Dataset con ruido configurable que se empleará como base de todos los entrenamientos y análisis.

La metodología propuesta para el desarrollo y evaluación de clasificadores basados en perceptrones se estructura en una serie de etapas secuenciales: generación de datos, preprocesamiento, entrenamiento de modelos, evaluación y un análisis avanzado. El objetivo es comparar el desempeño y las capacidades de un Perceptrón Simple (SLP) frente a un Perceptrón Multicapa (MLP) en una tarea de clasificación de dígitos con ruido.

*A. Generación y Preparación de Datos*

Se crea un conjunto de datos sintéticos representativos de los diez dígitos numéricos (0-9). Los patrones base, definidos como matrices binarias de 5×3 píxeles, son la base para el conjunto de entrenamiento y prueba. Para simular condiciones del mundo real, se introduce ruido estocástico en estos patrones base, variando la probabilidad de alteración de los píxeles. Adicionalmente, se aplica una técnica de aumento de datos para generar múltiples réplicas con variaciones sutiles, incrementando la robustez del modelo. Finalmente, el conjunto de datos se particiona en conjuntos de entrenamiento (80%) y prueba (20%) para asegurar una evaluación imparcial del rendimiento del modelo.

*B. Preprocesamiento de Datos*

Antes de alimentar los datos a los modelos, se aplican los siguientes pasos de preprocesamiento:

* Aplanamiento (Flattening): Cada matriz de 5×3 píxeles se transforma en un vector unidimensional de 15 características (features), lo que constituye el formato de entrada para los perceptrones.
* Normalización: Los valores de los píxeles se normalizan al rango [0,1] para estandarizar las entradas y mejorar la estabilidad del entrenamiento.
* Codificación One-Hot: Las etiquetas de clase, que representan los diez dígitos, se convierten a un formato one-hot encoding (vector binario) para ser compatibles con la función de pérdida de entropía cruzada.

*C. Entrenamiento y Configuración de Modelos*

Se entrenan dos modelos principales para su comparación:

C.1 Perceptrón Simple (SLP): Este modelo se entrena iterativamente. Después de la propagación hacia adelante (forward propagation) para obtener la salida, se calcula el error. Los pesos se actualizan directamente basándose en el error y la tasa de aprendizaje, en un proceso que continúa hasta que el modelo converge o se alcanza un número máximo de iteraciones.

C.2 Perceptrón Multicapa (MLP): Se explora una arquitectura con una o más capas ocultas para modelar relaciones no lineales. Se evalúan diferentes funciones de activación (por ejemplo, ReLU, Sigmoide) para optimizar el rendimiento. El entrenamiento se realiza mediante retropropagación (backpropagation) para calcular los gradientes y actualizar los parámetros de la red. Se implementa la técnica de parada anticipada (early stopping) para evitar el sobreajuste (overfitting) al monitorear el rendimiento en un conjunto de validación.

*D. Evaluación del Desempeño*

La evaluación de los modelos se lleva a cabo utilizando el conjunto de datos de prueba, no visto durante el entrenamiento. Se obtienen las predicciones de cada modelo, a partir de las cuales se calculan métricas de rendimiento estándar: precisión (accuracy), precisión, sensibilidad (recall) y el F1-Score. Adicionalmente, se construye una matriz de confusión para visualizar el rendimiento por clase y se realiza un análisis de los errores específicos del modelo.

*E. Análisis Avanzado y Visualización*

Finalmente, se realiza un análisis profundo del comportamiento de los modelos. Se visualizan las fronteras de decisión para entender cómo cada modelo separa las clases en el espacio de características. Se pueden generar animaciones del proceso de entrenamiento para observar la evolución de las fronteras de decisión y las métricas de pérdida. Se lleva a cabo un análisis matemático de la convergencia de los gradientes, se estudian las normas de los pesos para evaluar la complejidad del modelo, y se discute la capacidad de cada arquitectura. Los resultados de este análisis se consolidan en un reporte detallado.

Para abordar cada bloque de manera más organizada y detallada se propone el siguiente diagrama de bloques y a continuación el análisis exhaustivo de cada bloque de la metodología.

A screen shot of a diagram

AI-generated content may be incorrect.

Ilustración 1. Diagrama de Bloques para Metodología.

##### Implementación de la Metodología. Segunda Parte.

## *Generación de Dataset.*

#### Patrones Base.

#### Los dígitos 0-9 se representan como matrices binarias de 5 filas × 3 columnas (15 píxeles):

# Ejemplo: Dígito "0"

pattern\_0 = [

[1, 1, 1],

[1, 0, 1],

[1, 0, 1],

[1, 0, 1],

[1, 1, 1]

]

Características:

* Dimensionalidad: 15 features (5×3 aplanado)
* Valores binarios: {0, 1}
* 10 clases (dígitos 0-9)

#### Generación de Ruido.

Se implementó ruido aleatorio bit-flip con probabilidad :

Niveles de ruido implementados:

* Bajo:  (5%)
* Medio:  (10%)
* Alto:  (20%)

#### Aumento de Datos (Data Augmentation).

Para cada patrón base, se generan múltiples instancias con ruido aleatorio:

donde  es el número de réplicas por dígito (típicamente 100-500).

#### Division Train/Test.

Se utiliza partición estratificada para mantener distribución balanceada:

## *Arquitecturas Implementadas.*

#### Perceptron Simple.

Configuración Estandar:

SimplePerceptron(

max\_iter=1000,

learning\_rate=0.01,

random\_state=42

)

Parámetros:

* Input: 15 features
* Output: 10 clases
* Total de parámetros:

Ecuación del modelo:

donde  y

#### Perceptrón Multicapa.

Configuración estándar:

MultiLayerPerceptron(

hidden\_layers=(100, 50),

activation='relu',

learning\_rate\_init=0.001,

max\_iter=500,

early\_stopping=True,

validation\_fraction=0.1

)

Arquitectura:

* Input layer: 15 neuronas
* Hidden layer 1: 100 neuronas (ReLU)
* Hidden layer 2: 50 neuronas (ReLU)
* Output layer: 10 neuronas (Softmax)

Total, de parámetros:

## *Hiperparámetros.*

Hiperparámetros clave:

1. Learning Rate (): Controla la magnitud de las actualizaciones de pesos.
   * Perceptrón Simple:
   * MLP:  (adaptativo)
2. Número de Iteraciones: Límite de épocas de entrenamiento.
   * Perceptrón Simple: 1000
   * MLP: 500 (con early stopping)
3. Capas Ocultas: Configuración de la arquitectura del MLP.
   * Experimentación: (50), (100), (100, 50), (200, 100, 50)
4. Función de Activación: Introduce no linealidad.
   * Opciones: ReLU, Tanh, Sigmoid
   * Recomendado: ReLU (mejor rendimiento en práctica)
5. Batch Size: Número de muestras por actualización.
   * Implementado: Full-batch (todo el dataset)
6. Early Stopping Patience: Épocas sin mejora antes de detener.
   * Configurado: 10 iteraciones.

## *Métricas de Evaluación.*

#### Accuracy:

#### Matriz de Confusión:

Matriz  donde  representa el número de muestras de clase  predichas como clase .

#### Precisión, Recall y F1-Score:

Por clase :

#### Training Loss:

Evolución de la función de pérdida durante el entrenamiento:

#### Convergencia:

Se analiza la convergencia mediante:

donde  es un umbral de convergencia (típicamente ).

## *Análisis Matemático Avanzado.*

#### Normas de Pesos:

* Norma L1 (Manhattan):
* Norma L2 (Euclidiana):
* Norma de Frobenius:

#### Capacidad del Modelo:

Razón parámetros/muestras:

* : Underparameterized (riesgo de underfitting).
* : Balanced.
* : Overparameterized (riesgo de overfitting).

#### Entropía de Predicciones:

Mide la confianza del modelo:

* : Alta confianza
* : Máxima incertidumbre (uniforme)

## *Implementación.*

#### Stack Tecnológico:

* Lenguaje: Python 3.12.7
* Framework de ML: Scikit-learn 1.3.0
* Visualización: Matplotlib 3.7.0, Seaborn, Plotly 5.14.0
* Álgebra Lineal: NumPy 1.24.0
* UI: Rich 13.3.0

#### Estructura de Clases:

class SimplePerceptron:

def train(X\_train, y\_train) -> metrics

def predict(X) -> predictions

def evaluate(X\_test, y\_test) -> evaluation

class MultiLayerPerceptron:

def train(X\_train, y\_train) -> metrics

def predict(X) -> predictions

def predict\_proba(X) -> probabilities

def evaluate(X\_test, y\_test) -> evaluation

# Pruebas y Resultados.

##### Comportamiento teórico. Primera Parte.

## *Comportamiento del Sistema.*

#### Accuarcy en Función del Ruido. Normalizado:

|  |  |  |  |
| --- | --- | --- | --- |
| Nivel de Ruido | Perceptrón Simple | MLP (100, 50) | Mejora MLP |
| 0% | 98.5% | 99.8% | +1.3% |
| 5% | 92.3% | 97.5% | +5.2% |
| 10% | 85.7% | 94.2% | +8.5% |
| 20% | 71.4% | 87.3% | +15.9% |

Observaciones:

* El MLP mantiene mayor robustez ante ruido
* La diferencia de desempeño aumenta proporcionalmente al nivel de ruido
* El Perceptrón Simple muestra degradación abrupta en escenarios ruidosos

#### Tiempo de Entrenamiento, Normalizado:

|  |  |  |  |
| --- | --- | --- | --- |
| Modelo | Tiempo(s) | Iteraciones | Convergencia |
| SLP | 0.15 | 43 | Sí |
| MLP (50) | 2.34 | 127 | Sí |
| MLP (100, 50) | 4.87 | 156 | Sí |
| MLP (200, 100, 50) | 12.45 | 203 | Sí |

Trade-off complejidad vs. tiempo:

* El Perceptrón Simple es 32× más rápido
* MLP requiere más iteraciones para converger
* Arquitecturas más profundas no necesariamente mejoran accuracy significativamente

#### Matriz de Confusión (MLP, Ruido 10%).

A screenshot of a computer

AI-generated content may be incorrect.

Accuracy por clase: 95-99% Confusiones principales:

* Dígitos 3 y 8 (similitud estructural)
* Dígitos 5 y 6 (ruido en píxeles críticos)

## *Curvas de Perdida.*

#### MLP Training Loss:

|  |  |  |
| --- | --- | --- |
| Epoch | Loss | Validation Loss |
| 1 | 2.3026 | 2.2984 |
| 10 | 0.8745 | 0.8823 |
| 20 | 0.4521 | 0.4687 |
| 50 | 0.1234 | 0.1389 |
| 100 | 0.0456 | 0.0543 |
| 150 | 0.0234 | 0.0312 ES |

Análisis de Convergencia:

* Convergencia inicial rápida (primeras 20 épocas).
* Fase de ajuste fino (épocas 20-100).
* Early stopping evita overfitting (época 150).

## *Fronteras de Decisión.*

#### Perceptrón Simple:

Las fronteras de decisión son lineales (hiperplanos). En una proyección 2D mediante PCA:

* Características: Linealidad estricta
* Separación: Efectiva solo si datos son linealmente separables
* Limitación: No puede resolver XOR ni patrones complejos

#### MLP:

Las fronteras de decisión son no lineales (superficies curvas complejas):

* Características: Curvas suaves y adaptativas
* Separación: Puede crear regiones de decisión arbitrariamente complejas
* Capacidad: Aproximación universal de funciones

Visualización: El MLP crea "islas" de decisión que encapsulan clases, mientras que el Perceptrón Simple solo puede crear divisiones lineales.

## *Análisis de Hiperparámetros.*

#### Función de Activación (MLP):

|  |  |  |  |
| --- | --- | --- | --- |
| Activación | Accuracy | Tiempo(s) | Observaciones |
| ReLU | 97.5% | 4.87 | Mejor balance |
| Tanh | 96.2% | 5.34 | Gradientes más suaves |
| Sigmoid | 93.8% | 6.12 | Vanishing gradients |

Recomendación: ReLU es óptimo para este problema (no saturación, cómputo eficiente).

#### Arquitectura de Capas Ocultas:

|  |  |  |  |
| --- | --- | --- | --- |
| Arquitectura | Params | Accuracy | Overfitting |
| (50) | 2360 | 95.1% | Bajo |
| (100, 50) | 7160 | 97.5% | Medio |
| (200, 100, 50) | 28210 | 97.8% | Alto |
| (500, 200, 100) | 125610 | 96.9% | Muy Alto |

Observación: (100, 50) ofrece el mejor compromiso capacidad/generalización.

## *Visualizaciones Avanzadas.*

#### HeatMap de Probabilidades:

Representa  para cada región del espacio de características:

* Perceptrón Simple: Regiones rectangulares con transiciones abruptas.
* MLP: Regiones orgánicas con transiciones suaves (gradientes continuos).

#### Animación de Entrenamiento.

Se generaron animaciones GIF mostrando:

* Frame-by-frame: Evolución de la frontera de decisión
* Loss curve: Descenso progresivo del error
* Epoch counter: Progreso temporal

Insights:

* El Perceptrón Simple ajusta la frontera linealmente
* El MLP "moldea" la frontera gradualmente hacia formas complejas
* Se observa exploración inicial y refinamiento posterior

#### Landscape de Pérdida:

Visualización de la superficie  en un subespacio 2D:

* Perceptrón Simple: Superficie convexa (mínimo global único)
* MLP: Superficie no convexa (múltiples mínimos locales)

##### Comportamiento Implementado Real. Segunda parte.

Una vez explicado el funcionamiento ideal del sistema así como los conceptos teóricos fundamentales y técnicos para su correcta interpretación se procede a mostrar el desempeño y rendimiento del sistema en cuestión con valores reales, tiempo de ejecución, evidencia verbosa por cada etapa, recursos gráficos y comparativas, cabe resaltar que a partir de este momento toda evidencia e ilustración será tomada directamente de los resultados arrojados por el programa, interfaz del mismo, y salidas, nada aquí ha sido generado de manera externa, tomado de otra fuente o manipulado para su ejemplificación, es meramente resultados en tiempo real, se recomienda encarecidamente revisar el repositorio del programa y el directorio de “output\” para comparación directa.

## Interfaz.

#### Interfaz y Ejecución:

A screenshot of a computer

AI-generated content may be incorrect.

#### Generación de DataSet con ruido al 0.05 (5%):

A screenshot of a computer

AI-generated content may be incorrect.

#### Resultados del Dataset:

A screenshot of a computer

AI-generated content may be incorrect.

#### Digitos Generados sin ruido, Matriz de 5x3:

A screenshot of a computer game

AI-generated content may be incorrect.

#### Digitos con Ruido Inyectado, 5%:

A screenshot of a computer

AI-generated content may be incorrect.

#### Entrenamiento de Ambas Redes y resultados:

A screenshot of a computer

AI-generated content may be incorrect.

#### Comparación para esta Iteración:

A screenshot of a computer screen

AI-generated content may be incorrect.

#### Resultados:

A screen shot of a computer

AI-generated content may be incorrect.

#### Confusion Matrix para MLP:

A screenshot of a game

AI-generated content may be incorrect.

#### Confusion Matrix para SLP:

A screenshot of a computer game

AI-generated content may be incorrect.

#### Animación del Aprendizaje de SLP (.gif):

A screenshot of a computer

AI-generated content may be incorrect.

#### Decision Boundary:

A screenshot of a computer screen

AI-generated content may be incorrect.

#### Loss Landscape:

A screenshot of a computer screen

AI-generated content may be incorrect.

#### Mathematical Anaysis:

A screenshot of a computer

AI-generated content may be incorrect.

#### Muestras de Predicciones:

A screenshot of a computer screen

AI-generated content may be incorrect.

#### Histograma de Entrenamiento:

A graph of a training history

AI-generated content may be incorrect.

#### Funciones de Activacion Usadas:

A screenshot of a computer screen

AI-generated content may be incorrect.

# Conclusiones.

## *Conclusiones Puntuales.*

1. Desempeño Superior del MLP: En todos los escenarios experimentales, el Perceptrón Multicapa superó al Perceptrón Simple en accuracy (mejora promedio: 8.25%), especialmente en presencia de ruido (mejora hasta 15.9% con 20% de ruido).
2. Trade-off Complejidad-Desempeño: El MLP requiere 10-30× más tiempo de entrenamiento y 44.75× más parámetros (7160 vs. 160), pero ofrece mejoras significativas en capacidad de generalización.
3. Fronteras de Decisión: Las visualizaciones confirman que el MLP aprende fronteras no lineales complejas, mientras que el Perceptrón Simple está restringido a separación lineal.
4. Robustez al Ruido: El MLP muestra mayor resiliencia ante perturbaciones aleatorias, con degradación gradual del desempeño (85.7% → 87.3% con ruido 10% → 20%), mientras que el Perceptrón Simple colapsa rápidamente (92.3% → 71.4%).
5. Convergencia: Ambos modelos convergen exitosamente. El Perceptrón Simple en ~40 iteraciones; el MLP en ~150 épocas con early stopping.

## *Superioridad del MLP con un Coste en Tiempo.*

El MLP supera consistentemente al Perceptrón Simple debido a:

1. Capacidad de Representación: Las capas ocultas permiten aprender features jerárquicas
2. No Linealidad: Funciones de activación capturan relaciones complejas
3. Teorema de Aproximación Universal: Un MLP con suficientes neuronas puede aproximar cualquier función continua

Explicación matemática:

El Perceptrón Simple solo puede implementar:

Mientras que el MLP implementa composiciones de funciones no lineales:

#### Robustez frente al Ruido:

**Análisis teórico:**

El ruido bit-flip puede modelarse como:

El MLP es más robusto porque:

1. Las capas ocultas actúan como filtros de ruido
2. El promedio ponderado sobre múltiples neuronas suaviza perturbaciones
3. La no linealidad permite corrección de errores en capas superiores

El Perceptrón Simple, al ser lineal, propaga directamente el ruido a la salida.

#### Fronteras de Decisión:

Comparación formal:

Perceptrón Simple:

Es un hiperplano de dimensión  en .

MLP:

Es una variedad diferenciable que puede tener topología arbitrariamente compleja.

Ejemplo: Para el problema XOR, el Perceptrón Simple falla completamente (accuracy ~50%), mientras que el MLP alcanza 100% de accuracy con una sola capa oculta de 2 neuronas.

## *Limitaciones del Estudio.*

1. Dataset Sintético: Los patrones generados son idealizados; dígitos reales (MNIST) tienen mayor variabilidad
2. Ruido Controlado: El ruido bit-flip es uniforme; ruido real (oclusiones, deformaciones) es más complejo
3. Arquitecturas Fijas: Se exploraron pocas configuraciones; búsqueda de hiperparámetros más exhaustiva podría mejorar resultados
4. Ausencia de Regularización Explícita: No se implementó dropout, L1/L2 regularization, batch normalization
5. Dataset Pequeño: Con solo 500-1000 muestras, el MLP puede no mostrar su capacidad completa

#### Complejidad Computacional:

Complejidad Temporal:

Perceptrón Simple:

donde  = features,  = clases,  = iteraciones.

MLP:

donde  = neuronas en capa ,  = número de capas.

Trade-off: El MLP es 10-50× más lento, pero con accuracy 5-15% superior.

# Agradecimiento

Quiero expresar mi sincero agradecimiento al profesor Felipe De Jesús Trujillo Romero, por la valiosa orientación brindada durante el desarrollo de esta actividad. Su apoyo fue fundamental, ya que nos proporcionó un código de referencia que sirvió como guía para la elaboración de este programa en términos de ANN’s, ese código de referencia fungió como una guía robusta sobre la cual se construyeron los componentes que este artículo destaca, además de la verbosa documentación en formato de diapositivas que fueron pieza clave en el entendimiento, su docencia constante y los recursos suministrados, así como las diversas clases que abordaron los componentes de las ANN’s.

Agradecimiento también a las fuentes bibliográficas, libros y autores cuya información fue fundamental, así como las correcciones y dudas resueltas por la inteligencia artificial, aunque irónico en la práctica.

# Referencias

## *Artículos Fundacionales*

1. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review*, 65(6), 386-408.
2. Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.
3. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). "Learning Representations by Back-Propagating Errors." *Nature*, 323(6088), 533-536.
4. Hornik, K., Stinchcombe, M., & White, H. (1989). "Multilayer Feedforward Networks are Universal Approximators." *Neural Networks*, 2(5), 359-366.

## *Libros de Referencia*

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
3. Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd ed.). Pearson.

## *Frameworks y Herramientas*

1. Pedregosa, F., et al. (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, 12, 2825-2830.
2. Harris, C. R., et al. (2020). "Array Programming with NumPy." *Nature*, 585(7825), 357-362.