Clasificación Binaria: Modelos Lineales vs. Redes Neuronales

Estudio docente sobre MLP y principio de parsimonia en Machine Learning

Eduardo A. Farías Reyes

Ingeniero en Informática — IP Santo Tomás
IA, Machine Learning & Ciencia de Datos
contacto@efarias.cl · https://efarias.cl

Abstract

Este documento presenta un marco teórico completo para comprender las diferencias fundamentales entre modelos lineales (Regresión Logística) y redes neuronales (Perceptrón Multicapa) en el contexto de clasificación binaria. Se enfoca específicamente en el problema de datos no-linealmente separables, usando el caso de círculos concéntricos como ejemplo pedagógico. El material está diseñado para estudiantes de nivel universitario en cursos de Minería de Datos y Machine Learning, proporcionando fundamentos matemáticos, comparaciones prácticas, y guías de evaluación de modelos.

Keywords: MLP; Redes Neuronales; Regresión Logística; Machine Learning

Introducción

Contexto y Motivación

La clasificación binaria representa uno de los problemas fundamentales en Machine Learning, donde el objetivo es asignar observaciones a una de **dos clases posibles**. Mientras que modelos lineales como la Regresión Logística han demostrado efectividad en problemas linealmente separables, la creciente complejidad de datasets reales requiere enfoques más sofisticados.

Este documento surge de la necesidad de proporcionar material educativo riguroso que explique *por qué* y *cuándo* se requieren modelos no-lineales, específicamente redes neuronales. El enfoque pedagógico se centra en demostrar las limitaciones inherentes de modelos lineales mediante un problema geométrico concreto: los círculos concéntricos.

Objetivos del Documento

Este marco teórico tiene como objetivos:

1. Establecer fundamentos matemáticos de clasificación binaria
2. Explicar en detalle la Regresión Logística y sus limitaciones
3. Introducir el Perceptrón Multicapa (MLP) y redes neuronales
4. Demostrar el rol crítico de las funciones de activación no-lineales
5. Proporcionar herramientas de evaluación y comparación de modelos
6. Contextualizar el problema de círculos concéntricos

Notación Matemática

A lo largo del documento se utilizará la siguiente notación:

- $\boldsymbol{x} \in \mathbb{R}^d$: Vector de características (features) de dimensión d
- $y \in \{0, 1\}$: Etiqueta de clase binaria
- $\boldsymbol{w} \in \mathbb{R}^d$: Vector de pesos (weights)
- $b \in \mathbb{R}$: Sesgo (bias)
- n: Número de observaciones en el dataset
- $\sigma(z)$: Función sigmoide
- $\varphi(z)$: Función de activación genérica

- $\mathcal{L}$: Función de pérdida (loss function)
- $\hat{y}$: Predicción del modelo

Clasificación Binaria

Definición Formal

📘 **Definición:** Un problema de **clasificación binaria** consiste en aprender una función $f : \mathbb{R}^d \to \{0, 1\}$ que mapea vectores de características a etiquetas binarias, basándose en un conjunto de entrenamiento $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$.

El objetivo es minimizar el error de clasificación en datos no vistos, lo que se formaliza mediante:

$$\min_{f \in \mathcal{F}} E_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\mathbb{1}_{f(\boldsymbol{x})\neq y}\right]$$

donde $\mathbb{1}$ es la función indicadora y $\mathcal{F}$ es la clase de funciones consideradas.

Frontera de Decisión

La **frontera de decisión** (decision boundary) es el conjunto de puntos donde el clasificador es indiferente entre ambas clases:

$$\{\boldsymbol{x} \in \mathbb{R}^d : P(y = 1|\boldsymbol{x}) = 0.5\}$$

📐 **Teorema (Frontera Lineal):** Para un clasificador lineal, la frontera de decisión es un hiperplano en $\mathbb{R}^d$ definido por $\boldsymbol{w}^T\boldsymbol{x} + b = 0$.

Demostración: Sea $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$ donde σ es monótona. Entonces $P(y = 1|\boldsymbol{x}) = 0.5$ si y solo si $\sigma(z) = 0.5$, lo cual ocurre cuando $z = 0$ para la función sigmoide. Por lo tanto, $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ define un hiperplano. □

Separabilidad Lineal

📘 **Definición:** Un dataset es **linealmente separable** si existe un hiperplano que separa perfectamente las dos clases, es decir:

$$\exists \boldsymbol{w}, b : \forall i,\ y_i = 1 \Rightarrow \boldsymbol{w}^T\boldsymbol{x}_i + b > 0 \text{ y } y_i = 0 \Rightarrow \boldsymbol{w}^T\boldsymbol{x}_i + b < 0$$

La mayoría de problemas reales no son linealmente separables, lo que motiva el uso de modelos no-lineales.

Regresión Logística

Fundamentos Matemáticos

La Regresión Logística modela la probabilidad condicional de la clase positiva mediante:

$$P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

Función Sigmoide

La función sigmoide transforma la salida lineal en probabilidades:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Proposición: La función sigmoide satisface:
1. $\sigma(0) = 0.5$
2. $\lim_{z\to\infty} \sigma(z) = 1$
3. $\lim_{z\to-\infty} \sigma(z) = 0$
4. $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

La propiedad (4) es crucial para el cálculo eficiente de gradientes durante el entrenamiento.

Función de Pérdida

Para entrenar el modelo, se minimiza la **entropía cruzada binaria**:

$$\mathcal{L}(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

donde $\hat{y}_i = \sigma(\boldsymbol{w}^T\boldsymbol{x}_i + b)$.

📐 **Teorema (Convexidad):** La función de pérdida de entropía cruzada binaria es convexa en $\boldsymbol{w}$ cuando se usa con la función sigmoide.

Esta propiedad garantiza que el descenso de gradiente converge al mínimo global.

Limitaciones Fundamentales

📐 **Teorema (Limitación de Modelos Lineales):** Sea $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$ un clasificador de Regresión Logística. Entonces, la frontera de decisión de f es necesariamente un hiperplano en $\mathbb{R}^d$.

📌 **Corolario:** La Regresión Logística no puede clasificar correctamente datasets no-linealmente separables con accuracy superior al azar en el caso general.

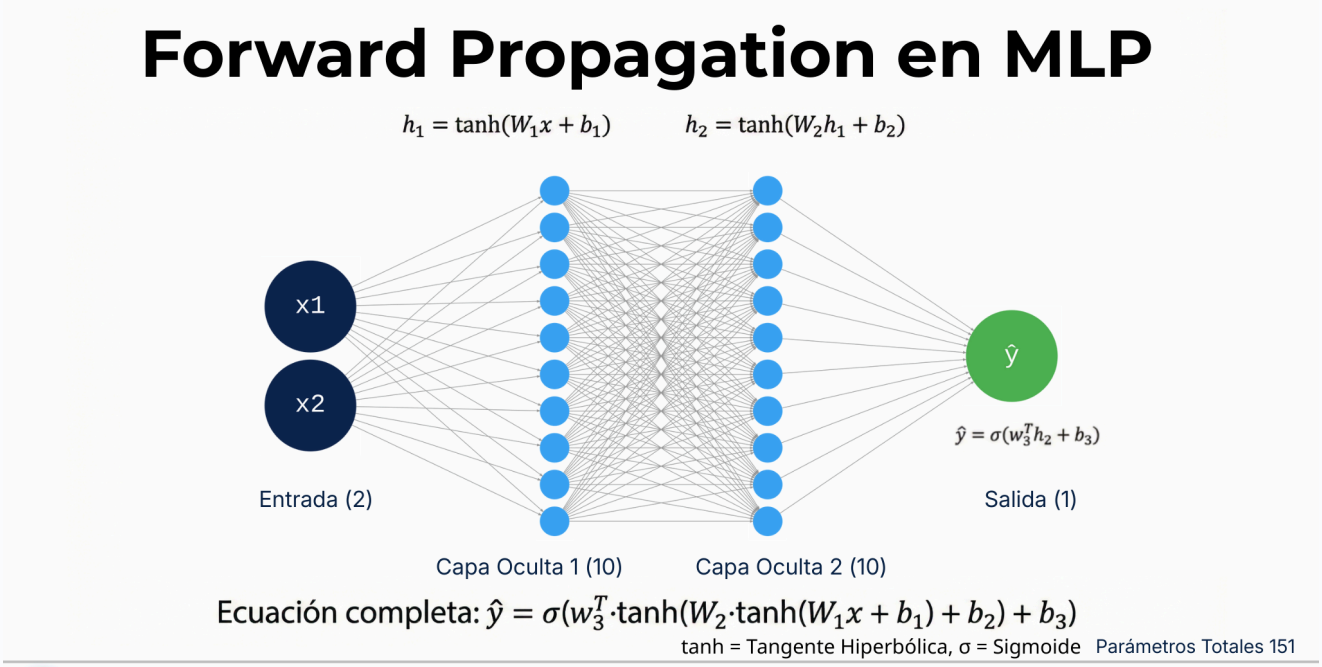

Figura 1: Esquema conceptual de un perceptrón multicapa.

Esta limitación fundamental motiva el desarrollo de modelos no-lineales como las redes neuronales.

Perceptrón Multicapa (MLP)

Arquitectura de Red Neuronal

Un Perceptrón Multicapa es una red neuronal feedforward compuesta por:

1. **Capa de entrada**: Recibe el vector $\boldsymbol{x} \in \mathbb{R}^d$
2. **Capas ocultas**: L capas que transforman los datos
3. **Capa de salida**: Produce la predicción $\hat{y}$

Una arquitectura se denota como d-n_1-n_2-...-n_L-c, donde n_l es el número de neuronas en la capa l y c es el número de clases (1 para clasificación binaria).

Transformación en una Neurona

Cada neurona en la capa l realiza:

$$a_j^{(l)} = \varphi\left(z_j^{(l)}\right) = \varphi\left(\sum_{i=1}^{n_{l-1}} w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}\right)$$

donde:

- $w_{ji}^{(l)}$: Peso de la conexión entre neurona i en capa $l-1$ y neurona j en capa l
- $b_j^{(l)}$: Sesgo de la neurona j en capa l
- φ: Función de activación no-lineal
- $a_j^{(l)}$: Activación de la neurona j en capa l

En notación matricial:

$$\boldsymbol{a}^{(l)} = \varphi\left(\boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}\right)$$

Funciones de Activación

Las funciones de activación introducen no-linealidad en el modelo, permitiendo aproximar funciones complejas.

Tangente Hiperbólica (Tanh)

$$\varphi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Propiedades:

- Rango: $[-1, 1]$
- Centrada en cero: $\tanh(0) = 0$
- Antisimétrica: $\tanh(-z) = -\tanh(z)$
- Derivada: $\tanh'(z) = 1 - \tanh^2(z)$

ReLU (Rectified Linear Unit)

$$\varphi(z) = \max(0, z) = \begin{cases} z & \text{si } z > 0 \\ 0 & \text{si } z \leq 0 \end{cases}$$

Propiedades:

- Rango: $[0, \infty)$
- No saturante para $z > 0$
- Computacionalmente eficiente
- Derivada: $\varphi'(z) = \mathbb{1}_{z>0}$

Función Sigmoide

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

Propiedades:

- Rango: $(0, 1)$
- Útil para capa de salida en clasificación binaria
- Puede sufrir "vanishing gradient" en capas profundas

Teorema de Aproximación Universal

📐 **Teorema (Teorema de Aproximación Universal - Cybenko 1989):** Sea φ una función de activación continua acotada no-constante. Entonces, para cualquier función continua $f : [0,1]^d \to \mathbb{R}$ y $\varepsilon > 0$, existe una red neuronal de una capa oculta con N neuronas tal que:

$$\sup_{\boldsymbol{x}\in[0,1]^d} |f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})| < \varepsilon$$

Este teorema establece que las redes neuronales con una sola capa oculta pueden aproximar cualquier función continua con precisión arbitraria, dado suficientes neuronas.

ℹ **Nota:** Aunque una capa es teóricamente suficiente, en la práctica redes más profundas aprenden representaciones más eficientes y requieren menos neuronas totales.

Propagación hacia Adelante

Para una red con L capas ocultas, la predicción se calcula mediante:

$$\boldsymbol{a}^{(0)} = \boldsymbol{x}$$
$$\boldsymbol{a}^{(l)} = \varphi\big(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}\big), \quad l = 1, ..., L$$
$$\hat{y} = \sigma\big(\boldsymbol{w}^{(L+1)T}\boldsymbol{a}^{(L)} + b^{(L+1)}\big)$$

Este proceso se conoce como **forward propagation** (propagación hacia adelante).

Número de Parámetros

El número total de parámetros en una red $d - n_1 - n_2 - \ldots - n_L - 1$ es:

$$N_{\text{params}} = \sum_{l=1}^{L+1}(n_{l-1} \times n_l + n_l)$$

donde $n_0 = d$ y $n_{L+1} = 1$.

💡 **Ejemplo:** Para la arquitectura 2-10-10-1 (nuestro experimento):

- Capa 1: $(2 \times 10) + 10 = 30$
- Capa 2: $(10 \times 10) + 10 = 110$
- Capa 3: $(10 \times 1) + 1 = 11$
- **Total: 151 parámetros**

Entrenamiento de Modelos

Función de Pérdida

Para clasificación binaria con MLP, también se usa entropía cruzada binaria:

$$\mathcal{L}(\boldsymbol{\Theta}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

donde $\boldsymbol{\Theta}$ representa todos los parámetros de la red.

Descenso de Gradiente

El objetivo es minimizar $\mathcal{L}(\boldsymbol{\Theta})$ mediante:

$$\boldsymbol{\Theta}^{(t+1)} = \boldsymbol{\Theta}^{(t)} - \eta\nabla_{\boldsymbol{\Theta}}\mathcal{L}\big(\boldsymbol{\Theta}^{(t)}\big)$$

donde $\eta > 0$ es el **learning rate** (tasa de aprendizaje).

Variantes de Descenso de Gradiente

1. Batch Gradient Descent: Usa todo el dataset

$$\nabla_{\boldsymbol{\Theta}}\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\Theta}}\mathcal{L}_i$$

2. Stochastic Gradient Descent (SGD): Usa una observación

$$\nabla_{\boldsymbol{\Theta}}\mathcal{L} \approx \nabla_{\boldsymbol{\Theta}}\mathcal{L}_i$$

3. Mini-batch Gradient Descent: Usa subconjuntos (típico: 16-256)

$$\nabla_{\boldsymbol{\Theta}}\mathcal{L} \approx \frac{1}{m}\sum_{i\in\mathcal{B}}\nabla_{\boldsymbol{\Theta}}\mathcal{L}_i$$

Backpropagation

El algoritmo de **backpropagation** (retropropagación) calcula eficientemente los gradientes en redes neuronales mediante la regla de la cadena.

Algoritmo

Algoritmo: Backpropagation

Entrada: Dataset $\mathcal{D}$, red con parámetros $\boldsymbol{\Theta}$, learning rate η

Para cada época:

1. **Forward pass:** Calcular $\hat{y}_i$ para cada $\boldsymbol{x}_i$
2. **Calcular pérdida:** $\mathcal{L} = -\frac{1}{n}\sum_i[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$
3. **Backward pass:** Para $l = L + 1, ..., 1$:
   - Calcular $\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{(l)}}$
   - Calcular $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L} = \delta^{(l)}\big(\boldsymbol{a}^{(l-1)}\big)^T$
   - Calcular $\nabla_{\boldsymbol{b}^{(l)}}\mathcal{L} = \delta^{(l)}$
4. **Actualizar:** $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta\nabla_{\boldsymbol{\Theta}}\mathcal{L}$

Retornar: Parámetros optimizados $\boldsymbol{\Theta}$

Cálculo del Error en Capa de Salida

Para la capa de salida con sigmoide:

$$\delta^{(L+1)} = \hat{y} - y$$

Esta forma simple resulta de la combinación de entropía cruzada con sigmoide.

Propagación del Error hacia Atrás

Para capas ocultas:

$$\delta^{(l)} = \left(\left(\boldsymbol{W}^{(l+1)} \right)^T \delta^{(l+1)} \right) \odot \varphi' \left(\boldsymbol{z}^{(l)} \right)$$

donde $\odot$ denota producto elemento-wise (Hadamard).

> **i Nota:** La complejidad temporal de backpropagation es $O\left(\sum_{l=1}^{L+1} n_{l-1} \times n_l\right)$, proporcional al número de conexiones en la red.

Métricas de Evaluación

Matriz de Confusión

Para clasificación binaria, la matriz de confusión es:

	Predicho 0	Predicho 1
Real 0	TN	FP
Real 1	FN	TP

Figura 1: Matriz de Confusión para Clasificación Binaria

donde:

- **TP** (True Positive): Correctamente clasificados como positivos
- **TN** (True Negative): Correctamente clasificados como negativos
- **FP** (False Positive): Error Tipo I
- **FN** (False Negative): Error Tipo II

Métricas Derivadas

Accuracy (Exactitud)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Proporción de predicciones correctas. Útil cuando las clases están balanceadas.

Precision (Precisión)

$$\text{Precision} = \frac{TP}{TP + FP}$$

De todos los predichos como positivos, qué proporción son correctos.

Recall (Sensibilidad)

$$\text{Recall} = \frac{TP}{TP + FN}$$

De todos los positivos reales, qué proporción detectamos.

F1-Score

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}$$

Media armónica de precision y recall.

Curva ROC y AUC

La **curva ROC** (Receiver Operating Characteristic) grafica TPR vs. FPR para diferentes umbrales.

El **AUC** (Area Under the Curve) cuantifica el rendimiento:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}) \, \mathrm{d}(\text{FPR})$$

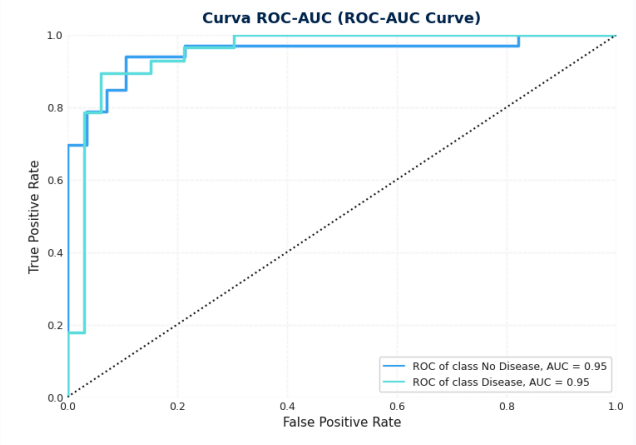

Figura 2: Ejemplo de curva ROC. Fuente: Elaboración propia.

> **Proposición: Interpretación del AUC:**
> - $\text{AUC} = 0.5$: Clasificación aleatoria
> - $\text{AUC} = 1.0$: Clasificación perfecta
> - $\text{AUC} > 0.8$: Generalmente considerado buen modelo

Comparación: Regresión Logística vs. MLP

Análisis Comparativo

Aspecto	Reg. Logística	MLP
Complejidad	Baja	Alta
Parámetros	$d + 1$	Cientos-Miles
Frontera	Lineal	No-lineal
Interpretabilidad	Alta	Baja
Velocidad train	ms	segundos
Velocidad inference	Muy rápida	Rápida
Requisitos datos	Pocos	Muchos
Overfitting	Menos propenso	Más propenso

Figura 2: Comparación entre Regresión Logística y MLP

El Problema de Círculos Concéntricos

Definición del Problema

Sean $\boldsymbol{x} \in \mathbb{R}^2$ puntos en el plano. El dataset de círculos concéntricos se define como:

$$y = \begin{cases} 1 & \text{si } \|\boldsymbol{x}\| < r_1 \\ 0 & \text{si } r_1 \leq \|\boldsymbol{x}\| \leq r_2 \end{cases}$$

donde $r_1 < r_2$ son los radios interior y exterior.

Por Qué Regresión Logística Falla

📐 **Teorema (Imposibilidad de Separación Lineal):** No existe $\boldsymbol{w} \in \mathbb{R}^2$ y $b \in \mathbb{R}$ tal que la frontera lineal $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ separe correctamente el dataset de círculos concéntricos.

Demostración: Supongamos que existe tal hiperplano. Entonces puntos en el círculo interior satisfacen $\boldsymbol{w}^T\boldsymbol{x} + b > 0$. Pero el círculo interior está completamente rodeado por el exterior, por lo que cualquier línea que pase por el origen cruzará ambas clases. Por simetría radial, no existe orientación de línea que separe las clases. □

Resultados Experimentales

En nuestro experimento con círculos concéntricos:

- **Dataset:** 1000 puntos, noise = 0.05, factor = 0.5
- **Arquitectura MLP:** 2-10-10-1 con activación tanh
- **Resultados:**
  - Regresión Logística: 47.5% accuracy
  - MLP: 100.0% accuracy
  - Diferencia: 52.5 puntos porcentuales

ℹ **Nota:** La accuracy de 47.5% para Regresión Logística está cerca del azar (50%), confirmando que el modelo no puede aprender el patrón. El MLP alcanza clasificación perfecta, demostrando su capacidad para fronteras no-lineales.

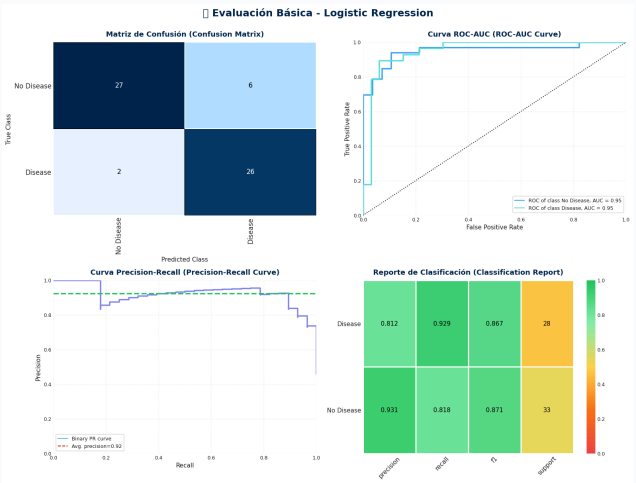

Figura 3: Ejemplo de curva métricas de modelo. Fuente: Elaboración propia.

Arquitecturas de Redes Neuronales

Profundidad vs. Amplitud

Redes Profundas (Deep Networks):

- Muchas capas con pocas neuronas cada una
- Ejemplo: 2-8-8-8-8-1
- Aprenden jerarquías de características

Redes Anchas (Wide Networks):

- Pocas capas con muchas neuronas
- Ejemplo: 2-50-1
- Más fáciles de entrenar

Regularización

Para prevenir overfitting:

L2 Regularization (Weight Decay):

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_{l=1}^{L} \|\boldsymbol{W}^{(l)}\|_F^2$$

Dropout:

Durante entrenamiento, desactivar aleatoriamente neuronas con probabilidad p.

Conclusiones

Resumen de Conceptos Clave

Este documento ha presentado un marco teórico completo para comprender las diferencias fundamentales entre modelos lineales y redes neuronales en clasificación binaria:

1. La **Regresión Logística** es un modelo lineal eficiente pero limitado a problemas linealmente separables.
2. El **MLP** introduce no-linealidad mediante funciones de activación, permitiendo aproximar funciones arbitrariamente complejas.
3. El **Teorema de Aproximación Universal** garantiza que redes neuronales pueden representar cualquier función continua.
4. El algoritmo de **Backpropagation** permite entrenar redes eficientemente mediante cálculo de gradientes.
5. El problema de **círculos concéntricos** demuestra claramente las limitaciones de modelos lineales y la necesidad de no-linealidad.

Implicaciones Pedagógicas

Este marco teórico está diseñado para:

- Proporcionar fundamentos matemáticos rigurosos
- Mantener claridad didáctica mediante ejemplos
- Contextualizar conceptos abstractos en problemas concretos
- Facilitar la transición de teoría a práctica

Los estudiantes que dominen estos conceptos estarán preparados para:

- Seleccionar modelos apropiados para problemas específicos
- Diseñar arquitecturas de redes neuronales
- Evaluar y comparar modelos sistemáticamente
- Comprender literatura avanzada de Deep Learning

Mensaje Final

La elección entre Regresión Logística y MLP no se trata de cuál es "mejor" en términos absolutos, sino de cuál es **apropiado** para el problema específico. Como se demostró con círculos concéntricos:

"*La herramienta correcta depende de la geometría de los datos. Un modelo simple puede ser suficiente para problemas linealmente separables, pero la complejidad inherente de algunos problemas requiere la capacidad expresiva de redes neuronales.*"

Este principio se extiende más allá de Machine Learning: en ingeniería y ciencia, la sofisticación de la solución debe coincidir con la complejidad del problema.

Referencias

1. **Goodfellow, I., Bengio, Y., & Courville, A.** (2016). *Deep Learning*. MIT Press.
2. **Bishop, C. M.** (2006). *Pattern Recognition and Machine Learning*. Springer.
3. **Géron, A.** (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.
4. **Cybenko, G.** (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2(4), 303-314.
5. **Hornik, K., Stinchcombe, M., & White, H.** (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5), 359-366.
6. **Rumelhart, D. E., Hinton, G. E., & Williams, R. J.** (1986). Learning Representations by Back-Propagating Errors. *Nature*, 323(6088), 533-536.
7. **LeCun, Y., Bengio, Y., & Hinton, G.** (2015). Deep Learning. *Nature*, 521(7553), 436-444.
8. **Kingma, D. P., & Ba, J.** (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.
9. **Pedregosa, F., et al.** (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

Apéndice A: Preguntas de Autoevaluación

Nivel Básico

1. ¿Qué es una frontera de decisión y por qué es importante en clasificación?
2. Explique por qué la Regresión Logística solo puede crear fronteras lineales.
3. ¿Cuál es el rol de las funciones de activación en redes neuronales?
4. Defina accuracy, precision y recall. ¿Cuándo cada métrica es más relevante?
5. ¿Qué significa que un dataset sea "linealmente separable"?

Nivel Intermedio

6. Demuestre que un MLP sin funciones de activación no-lineales colapsa a un modelo lineal.
7. Compare redes profundas vs. redes anchas. ¿Cuáles son las ventajas de cada enfoque?
8. Interprete una learning curve que muestra gran brecha entre training y validation accuracy.
9. Calcule el número de parámetros para una arquitectura 5-20-15-10-3.
10. Explique el concepto de vanishing gradient y cómo afecta el entrenamiento.

Nivel Avanzado

11. Diseñe una arquitectura de red neuronal para un problema con 100 features y 5 clases. Justifique sus decisiones de diseño.
12. Derive la regla de actualización de backpropagation para una capa con activación ReLU.
13. Analice el trade-off entre capacidad del modelo y riesgo de overfitting. ¿Cómo regularización mitiga este problema?
14. Proponga un experimento para determinar si un problema requiere un modelo no-lineal.
15. Discuta las implicaciones del Teorema de Aproximación Universal para el diseño de arquitecturas en la práctica.

Apéndice B: Glosario de Términos

Backpropagation: Algoritmo para calcular gradientes en redes neuronales mediante regla de la cadena.

Batch: Subconjunto de datos usado en cada iteración de entrenamiento.

Bias (Sesgo): Término independiente en transformaciones lineales.

Decision Boundary: Frontera que separa regiones de diferentes clases en el espacio de features.

Epoch (Época): Una pasada completa por el dataset de entrenamiento.

Feature: Variable de entrada o atributo de las observaciones.

Forward Propagation: Cálculo de predicción desde input hasta output.

Gradient Descent: Algoritmo de optimización que sigue la dirección opuesta al gradiente.

Hidden Layer: Capa intermedia en red neuronal entre input y output.

Learning Rate: Parámetro que controla el tamaño del paso en optimización.

Loss Function: Función que cuantifica el error del modelo.

Overfitting: Modelo que memoriza training data pero falla en generalizar.

Regularization: Técnicas para prevenir overfitting (L1, L2, dropout).

Underfitting: Modelo demasiado simple que no captura el patrón de los datos.

Weight (Peso): Parámetro multiplicativo en transformaciones lineales.