
Universidad de
SanAndrés

# I302 - Aprendizaje Automático y Aprendizaje Profundo

## Trabajo Práctico 3: Redes Neuronales

Nombre Apellido

12 de mayo de 2025

Ingeniería en Inteligencia Artificial

**Resumen**

En este trabajo se buscó modelar un clasificador multiclase utilizando redes neuronales. Para ello, se implementó desde cero una red neuronal básica (modelo M0), entrenada mediante *backpropagation* y descenso por gradiente, utilizando activación *ReLU* y pérdida *cross-entropy*. Posteriormente, se incorporaron mejoras progresivas al entrenamiento y arquitectura —incluyendo *dropout*, regularización L2, *early stopping*, *rate scheduling*, mini-batch SGD y el optimizador *Adam*— y se exploraron múltiples combinaciones de hiperparámetros mediante *grid search* (modelo M1).

Luego, se reimplementó el modelo utilizando `PyTorch` (modelo M2) con los mismos hiperparámetros óptimos hallados. Sobre esa base, se exploraron arquitecturas más profundas y anchas (modelo M3), y también se forzó un escenario de sobreajuste intencional (modelo M4).

Los resultados mostraron una mejora progresiva en la métrica de *accuracy* desde el modelo base (57.4 %) hasta el mejor modelo (M3), que alcanzó un 64.2 % de *accuracy* con una *cross-entropy loss* de 0.94 sobre el conjunto de prueba. El modelo M4 logró el mayor *accuracy* (70.6 %), pero con una pérdida significativamente alta (9.92), reflejando un sobreajuste severo.

Finalmente, se utilizó el modelo M3 para predecir las probabilidades a posteriori de un conjunto oculto de muestras, generando un archivo de salida con las predicciones requeridas.

# 1. Introducción

El objetivo de este trabajo práctico fue construir un modelo de clasificación multiclase utilizando redes neuronales, partiendo desde una implementación propia y luego utilizando la biblioteca `PyTorch`.

El dataset utilizado es una variante del clásico MNIST, pero en este caso compuesto por caracteres japoneses. Cada muestra corresponde a una imagen en escala de grises de $28 \times 28$ píxeles, y existen 49 clases posibles. Por lo tanto, la entrada del algoritmo es una imagen vectorizada (de 784 dimensiones), y la salida esperada es la probabilidad a posteriori de pertenencia a cada una de las 49 clases.

Para resolver esta tarea se implementó una red neuronal completamente conectada, entrenada mediante *backpropagation* y optimización por descenso de gradiente. Inicialmente se desarrolló un modelo base (M0) con dos capas ocultas, al cual progresivamente se le incorporaron mejoras tales como *dropout*, regularización L2, *batch normalization*, *early stopping*, *learning rate scheduling*, mini-batch SGD y el optimizador *Adam*, dando lugar al modelo M1.

Posteriormente, se replicó la arquitectura óptima en `PyTorch` (modelo M2), se exploraron nuevas arquitecturas con más profundidad y capacidad (modelo M3), y finalmente se construyó un modelo deliberadamente sobreajustado (modelo M4) para estudiar su comportamiento. A lo largo del trabajo se evaluaron y compararon todas las variantes utilizando métricas estándar como *accuracy*, *cross-entropy loss* y la matriz de confusión, tanto en los conjuntos de entrenamiento, validación y prueba.

# 2. Métodos

En este trabajo se abordó la tarea de clasificación multiclase utilizando redes neuronales, comenzando por una implementación propia desde cero y culminando con experimentos en PyTorch. A continuación se describen los modelos utilizados, los algoritmos de entrenamiento implementados y las técnicas de mejora aplicadas.

## 2.1. Algoritmo de entrenamiento

El entrenamiento de la red neuronal se realizó utilizando el algoritmo de *backpropagation*, adaptado al caso de clasificación multiclase. Se utilizó activación *softmax* en la capa de salida y la *loss function* fue la cross-entropy:

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{1}$$

donde $y \in \mathbb{R}^C$ es el vector one-hot de la clase verdadera, $\hat{y} \in \mathbb{R}^C$ es el vector de probabilidades predicho por la red, y $C$ es la cantidad de clases.

Durante el **forward pass**, en cada capa $l$ se realiza:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{2}$$

$$a^{(l)} = \phi(z^{(l)}) \tag{3}$$

donde $a^{(l-1)}$ es la activación de la capa anterior, $W^{(l)}$ y $b^{(l)}$ son los pesos y sesgos de la capa $l$, y $\phi$ es la *activation function* (ReLU para capas ocultas, softmax en la capa final).

Durante el **backward pass**, se computan los gradientes usando la regla de la cadena. En la capa de salida, gracias a la combinación de softmax con cross-entropy, el gradiente se simplifica:

$$\delta^{(L)} = \hat{y} - y \tag{4}$$

Para las capas ocultas se propaga el error hacia atrás:

$$\delta^{(l)} = (W^{(l+1)})^\top \delta^{(l+1)} \circ \phi'(z^{(l)}) \tag{5}$$

donde $\circ$ denota el producto elemento a elemento, y $\phi'$ es la derivada de la función de activación. En el caso de ReLU:

$$\phi'(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases} \tag{6}$$

Finalmente, los gradientes se usan para actualizar los parámetros mediante *gradient descent*:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(l)}} \tag{7}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(l)}} \tag{8}$$

donde $\eta$ es el *learning rate*.

Este proceso se repite por cada *mini-batch* durante todas las *epochs* del entrenamiento.

## 2.2. Técnicas de mejora y regularización

Para mejorar la performance del modelo se implementaron diversas técnicas:

- **Rate scheduling**: se exploró la modificación dinámica del learning rate a lo largo del entrenamiento. Se utilizó un scheduler lineal con saturación, y otro exponencial. Las funciones de actualización fueron:

$$\text{Lineal: } \eta_t = \eta_0 - t \cdot \frac{\eta_0 - \eta_f}{T} \quad (t < T) \tag{9}$$

$$\text{Exponencial: } \eta_t = \eta_0 \cdot \gamma^t \tag{10}$$

- **Mini-batch SGD**: se implementó una versión de gradiente descendente estocástico por mini-batchs, lo que acelera el entrenamiento y mejora la generalización.

- **Optimizador ADAM**: se incorporó el algoritmo ADAM, que combina momentum (promedio exponencial del gradiente) y RMSProp (promedio del cuadrado del gradiente), con las siguientes actualizaciones por parámetro:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{12}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{13}$$

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{14}$$

- **Regularización L2**: se añadió un término $\frac{\lambda}{2m} \sum_l \|W^{(l)}\|^2$ a la función de costo para penalizar pesos grandes.

- **Dropout**: durante el entrenamiento, se apagó aleatoriamente un porcentaje de unidades en cada capa oculta, lo que promueve la robustez del modelo.

- **Early stopping**: se monitoreó la pérdida de validación y se interrumpió el entrenamiento si no mejoraba tras $p$ épocas consecutivas.

## 2.3. Datos y preprocesamiento

El conjunto de datos original fue dividido en tres particiones: 70 % para entrenamiento, 15 % para validación y 15 % para prueba.

En cuanto al preprocesamiento, los valores de entrada fueron normalizados dividiendo todos los píxeles por 255, de modo que los valores quedaran en el rango $[0, 1]$.

Adicionalmente, se realizó un análisis exploratorio sobre el dataset en busca de valores faltantes o desbalanceo de clases, sin encontrar problemas significativos. Por lo tanto, no fue necesario realizar imputaciones ni filtrado adicional.

Se utilizó one-hot encoding para las etiquetas con el fin de facilitar el cálculo de la Cross Entropy y su derivada. Para el entrenamiento con PyTorch, las matrices fueron convertidas a tensores flotantes.

## 2.4. Métricas utilizadas

Las métricas de evaluación utilizadas para comparar los modelos fueron:

- **Accuracy**: mide la proporción de predicciones correctas sobre el total de muestras. Matemáticamente, se define como:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i) \tag{15}$$

  donde $N$ es la cantidad de muestras, $y_i$ es la clase verdadera de la $i$-ésima muestra, $\hat{y}_i$ es la clase predicha, y $\mathbf{1}$ es la función indicadora que vale 1 si el predicho es correcto y 0 en caso contrario.

- **Cross-Entropy promedio**: mide la distancia entre la distribución verdadera (one-hot) y la distribución de probabilidades predicha por el modelo. Para un problema de clasificación multiclase con $C$ clases, se define como:

$$\text{CrossEntropy}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \tag{16}$$

  donde $y_{i,c}$ es 1 si la muestra $i$ pertenece a la clase $c$ (0 en caso contrario), y $\hat{y}_{i,c}$ es la probabilidad predicha de que la muestra $i$ pertenezca a la clase $c$.

- **Matriz de Confusión**: es una matriz de tamaño $C \times C$, donde la entrada $(i, j)$ representa la cantidad de muestras cuya clase verdadera es $i$ pero fueron clasificadas como $j$. Permite observar errores específicos entre pares de clases, identificar confusiones frecuentes y evaluar el rendimiento por clase.

Estas métricas fueron calculadas para los conjuntos de entrenamiento, validación y prueba, y se utilizaron para comparar los modelos desarrollados (M0–M4).

## 2.5. Búsqueda de hiperparámetros

Para seleccionar los valores óptimos de hiperparámetros, se realizó una búsqueda mediante *grid search*, evaluando múltiples combinaciones de arquitectura (cantidad y tamaño de capas ocultas) y técnicas de entrenamiento (dropout, regularización L2, scheduler, uso de ADAM, etc.).

Dado que el espacio total de combinaciones posibles es extremadamente grande, se adoptó una estrategia *greedy*: a medida que se observaban mejoras significativas con ciertas técnicas (por ejemplo, el uso de ADAM respecto a SGD estándar), se fijaban dichas decisiones para los experimentos posteriores, reduciendo así la complejidad de la búsqueda. Esto permitió enfocar el análisis en los hiperparámetros más prometedores, manteniendo la coherencia entre configuraciones comparables y limitando el costo computacional, que crece rápidamente con la cantidad de combinaciones a evaluar.

El espacio de búsqueda considerado se resume en la Tabla 1. Incluye arquitecturas de redes de diferentes profundidades y tamaños, distintos valores de tasa de aprendizaje, regularización y tamaño de batch, así como combinaciones de técnicas de optimización y control de overfitting.

Para cada configuración evaluada, el entrenamiento se realizó sobre el conjunto de entrenamiento y la selección del mejor modelo se basó en la métrica de Cross Entropy promedio sobre el conjunto de validación. Es decir, el mejor conjunto de hiperparámetros fue aquel que minimizó la pérdida de validación, y también se utilizó la accuracy como referencia adicional.

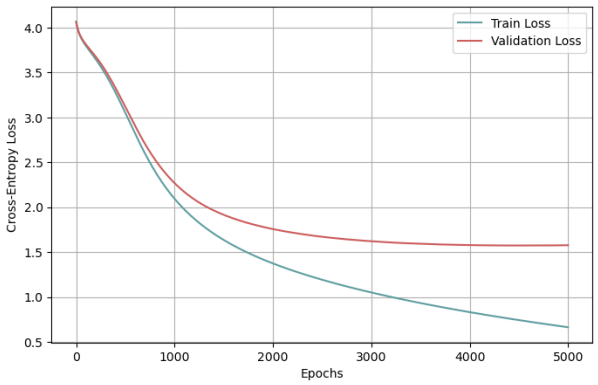
| Hiperparámetro | Valores explorados |
|---|---|
| Capas ocultas | [32], [64], [128]<br>[64, 32], [128, 64], [256, 128]<br>[128,128,64], [256,128,64], [512,256,128]<br>[64,64,64], [128,128,128], [256,256,256] |
| Learning Rate | 0.001, 0.01, 0.05, 0.1 |
| Regularización L2 | 0.001, 0.01 |
| Dropout | 0.1, 0.2 |
| Scheduler | `linear`, `exponential` |
| Optimizador ADAM | `True` |
| Batch size | 16, 32, 64, 128, 256 |
| Early stopping | `True` |
| Patience | 10 |

Cuadro 1: Espacio de hiperparámetros que se usa para explorar en grid search. Ver salidas del notebook para ver los resultados de cada uno de los casos.

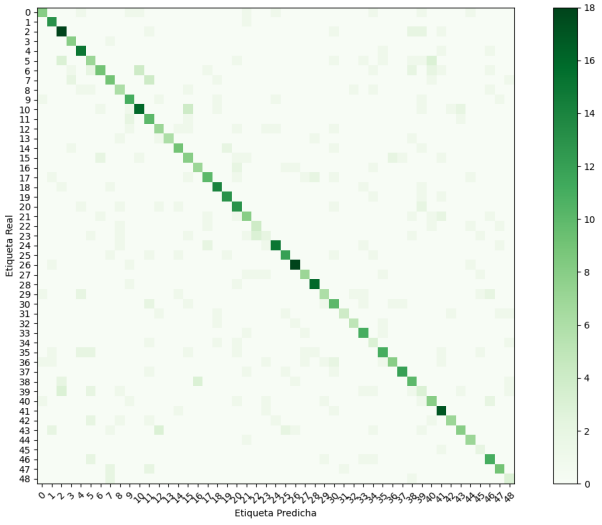
# 3. Resultados

## 3.1. Análisis del modelo M0

El modelo base (M0) consistió en una red neuronal completamente conectada con dos capas ocultas de 100 y 80 unidades respectivamente, y funciones de activación ReLU. Fue entrenado utilizando gradiente descendente estándar con tasa de aprendizaje fija $\eta = 0{,}01$ y sin técnicas de regularización.




(a) Evolución de la función de costo (Cross-Entropy).




(b) Matriz de confusión sobre validación.

Figura 1: Desempeño del modelo M0.

En la Figura 1a se muestra la evolución de la función de costo (Cross Entropy) durante el entrenamiento. Se observa una disminución progresiva tanto en el conjunto de entrenamiento como en el de validación, estabilizándose recién alrededor de las 5000 épocas. Esto sugiere que el modelo necesitó un largo entrenamiento para converger, en parte debido a la simplicidad del algoritmo de

optimización utilizado. Además, en la Figura 1b se presenta la matriz de confusión del modelo. Si bien el desempeño global no es óptimo, se observa que se tiende a predecir correctamente una buena parte de las clases, ya que la mayor concentración de valores se encuentra sobre la diagonal principal. Esto indica que, aunque hubo errores, el modelo logró capturar algunas relaciones significativas entre los datos de entrada y sus respectivas clases.

Una vez alcanzada la convergencia, el modelo obtuvo una Accuracy de 0.605 y una Cross Entropy Loss de 1.576. Estos resultados marcan una línea base sobre la cual se evaluarán las mejoras implementadas en modelos posteriores (M1 a M4).

### 3.2. Análisis del modelo M1

El modelo M1 corresponde a una red neuronal avanzada, en la cual se incorporaron progresivamente distintas técnicas de optimización y regularización, las cuales se explican en la sección de Técnicas de mejora y regularización 2.2

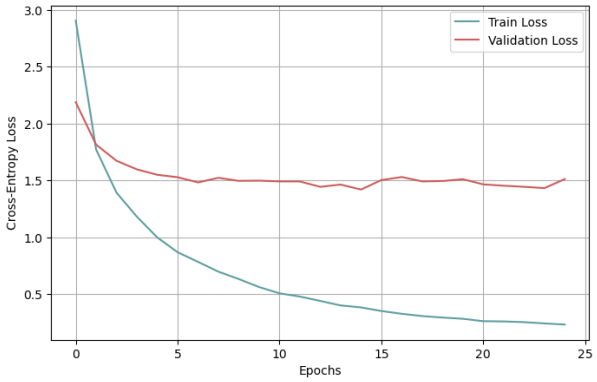
Cada mejora fue incorporada de manera incremental, y se evaluó individualmente su impacto sobre el desempeño y el tiempo de entrenamiento. Una vez fijadas las técnicas más prometedoras, se realizó un *grid search* sobre los hiperparámetros restantes. El espacio de búsqueda se ve en la tabla 1

| **Hiperparámetro** | **Mejor valor** |
|---|---|
| Capas ocultas | [128] |
| Learning rate | 0.001 |
| Regularización L2 | 0.01 |
| Dropout | 0.1 |
| Scheduler | Exponential |
| Optimizador | ADAM |
| Mini-batch size | 16 |
| Early stopping | Activado |
| Patience | 10 |

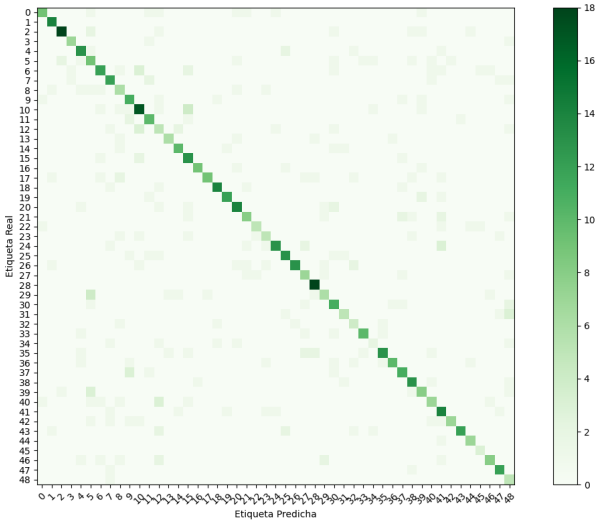
Cuadro 2: Mejor configuración obtenida tras evaluar 1920 combinaciones.

La configuración final se muestra en la tabla 2 y es la que se utilizó para entrenar el modelo M1. Es interesante destacar que la mejor arquitectura encontrada fue sorprendentemente simple: una sola capa oculta con 128 neuronas. Esto sugiere que el problema no requería una red profunda para obtener buenos resultados. Además, el mejor desempeño se logró con un batch size pequeño (16), lo que podría haber contribuido a una mayor variabilidad en los gradientes, favoreciendo una mejor generalización.

En la Figura 2a puede observarse una convergencia mucho más rápida en comparación con el modelo M0, debido tanto al optimizador ADAM como al uso del scheduler, además del early stopping, que interrumpió automaticamente el entrenamiento en la epoch 24. En validación se observa una mejora en el mínimo alcanzado por la función de costo, ya que se obtuvo una Accuracy de 0.640 y una Cross Entropy Loss de 1.501 sobre el conjunto de validación. La matriz de confusión (Figura 2b) no muestra una mejora drástica respecto al modelo M0, ya que la accuracy sobre validación aumentó solo un 3,5 %. Sin embargo, la principal ventaja del modelo M1 fue la eficiencia en el entrenamiento: logró un rendimiento mejor en tan solo 24 epochs, en contraste con las 5000 necesarias en el modelo base.




(a) Evolución de la función de costo (Cross-Entropy).




(b) Matriz de confusión del modelo M1 sobre validación.

Figura 2: Desempeño del modelo M1.

### 3.3. Análisis del modelo M2

El modelo M2 corresponde a la implementación en PyTorch de la misma arquitectura y conjunto de hiperparámetros hallados previamente en el grid search aplicado al modelo M1. El objetivo fue verificar si los resultados obtenidos se mantenían consistentes al utilizar una plataforma de entrenamiento más robusta como PyTorch.

Se utilizó una red neuronal con una sola capa oculta de 128 unidades, activación ReLU, regularización L2 con $\lambda = 0{,}01$, dropout de 0.1, optimizador ADAM, tasa de aprendizaje de 0,001, scheduler exponencial, batch size de 16 y early stopping con patience 10.

Tras el entrenamiento, el modelo alcanzó un desempeño muy similar al de M1, confirmando la solidez de nuestro modelo. El modelo M2 consiguio un Accuracy de 0.65 y una Cross Entropy Loss de 1.741 con un early stoing en la epoch 75.

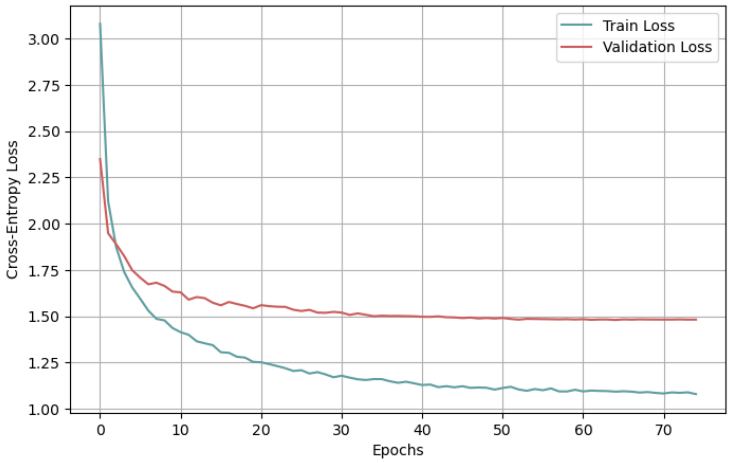



Figura 3: Evolución de la función de costo (Cross-Entropy) para el modelo M2.

La Figura 3 muestra la evolución de la pérdida durante el entrenamiento. Al igual que en M1, se observa una rápida convergencia. La similitud en el desempeño reafirma la consistencia entre ambas implementaciones.

### 3.4. Análisis del modelo M3

En el modelo M3 se mantuvieron fijos los hiperparámetros que habían demostrado buen rendimiento en modelos anteriores (learning rate, regularización L2, dropout, uso de ADAM, scheduler y early stopping) y se realizó un nuevo *grid search* centrado exclusivamente en la arquitectura de la red, es decir, la cantidad de capas ocultas y unidades por capa.

Esto permitió ampliar considerablemente el espacio de búsqueda respecto al modelo M1, evaluando desde arquitecturas simples con una sola capa hasta redes muy profundas y anchas. La exploración incluyó múltiples arquitecturas, organizadas en:

- Capas únicas: [32], [64], [128], [256], [512], [1024], [2048], [4096], [5000]
- Dos capas (creciente y decreciente): [64, 32], [128, 64], ..., [1024, 2048]
- Tres capas (decreciente): [512, 256, 128], ..., [5000, 2048, 1024]
- Tres capas (creciente): [128, 256, 512], ..., [32, 64, 128]
- Simétricas: [64, 64, 64], ..., [1024, 1024, 1024]
- Cuatro capas: [256, 128, 64, 32], ..., [1024, 512, 256, 128]
- Arquitecturas de muy alta capacidad: [5000, 5000], ..., [4096, 4096, 4096]

La arquitectura que obtuvo el mejor desempeño fue sorprendentemente una de las más grandes: **[5000, 5000]**, alcanzando una Accuracy de 0.682 y Cross Entropy Loss de 0.614, frenando en la epoch 58.

Este resultado es llamativo, ya que el modelo cuenta con más capas ocultas que cantidad de muestras disponibles en el dataset. En principio, esto podría sugerir un riesgo alto de sobreajuste. Sin embargo, no se observó overfitting gracias a la incorporación de técnicas de regularización como dropout, penalización L2, y sobre todo el uso de *early stopping*, que interrumpió el entrenamiento antes de que el modelo comenzara a memorizar los datos.

Esto demuestra que, si se aplican correctamente las estrategias de regularización, es posible entrenar redes grandes sin perder capacidad de generalización. Además, evidencia que la arquitectura **[128]** seleccionada en M1 no era subóptima por sí misma, sino que reflejaba las limitaciones del espacio explorado en ese momento, donde también se buscaban combinaciones de técnicas y no se evaluaron arquitecturas de alta capacidad.

Cabe mencionar que no se descarta la posibilidad de obtener aún mejores resultados con arquitecturas más grandes que las evaluadas, ya que el modelo no mostró signos de saturación. Sin embargo, por cuestiones prácticas y de costo computacional, se decidió frenar la exploración en ese punto.

En la Figura 4 se observa que la evolución de la función de costo para M3 presenta una ligera mejora respecto a los modelos anteriores, especialmente en las primeras épocas, alcanzando una pérdida de validación más baja de forma más estable.

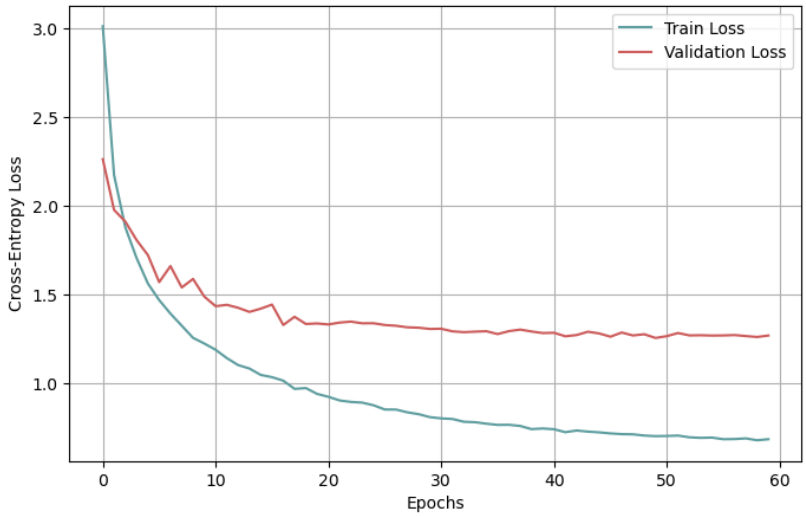



Figura 4: Evolución de la función de costo (Cross-Entropy) para el modelo M3.

### 3.5. Análisis del modelo M4

El objetivo del modelo M4 fue inducir un sobreajuste para ver sus efectos. Para ello, se utilizó una red neuronal de gran capacidad, compuesta por una capa oculta de 5000 neuronas, y se desactivaron todas las técnicas principales de regularización: no se aplicó *early stopping*, ni regularización L2, ni *dropout*. El modelo fue entrenado durante 200 épocas completas, utilizando un *batch size* de 64, el optimizador *ADAM* y una tasa de aprendizaje fija de 0.001.

El resultado luego de 200 epochs fue un caso claro de sobreajuste:

- **Train Accuracy**: 1.000 **Train Cross-Entropy**: 0.0005
- **Validation Accuracy**: 0.718 **Validation Cross-Entropy**: 8.61

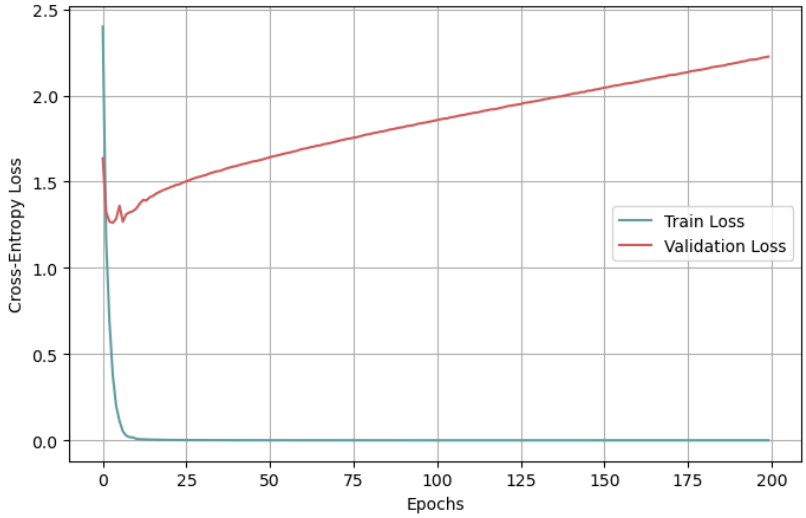



Figura 5: Evolución de la función de costo (Cross-Entropy) para el modelo M4.

En la Figura 5 se ve como el modelo logró memorizar perfectamente los datos de entrenamiento, alcanzando una pérdida prácticamente nula y una precisión del 100 %. Sin embargo, la pérdida sobre el conjunto de validación fue extremadamente alta, a pesar de que el *accuracy* se mantuvo elevado.

Este comportamiento se explica porque, si bien el modelo acertó muchas predicciones en validación, cuando se equivocó lo hizo con una seguridad muy alta. Es decir, asignó una probabilidad cercana a 1 a una clase incorrecta, lo cual la función de pérdida *cross-entropy* penaliza fuertemente. La *accuracy*, al solo contar aciertos y no tener en cuenta la confianza de las predicciones, no refleja este problema.

Este experimento demuestra que un *accuracy* elevado no garantiza que un modelo esté generalizando correctamente, y que es necesario considerar métricas complementarias como la *cross-entropy* para evaluar el comportamiento real del modelo.

### 3.6. Resultados en el Set de Test

| **Modelo** | **Accuracy (Test)** | **Cross-Entropy (Test)** |
|---|---|---|
| M0 | 0.574 | 1.78 |
| M1 | 0.600 | 1.76 |
| M2 | 0.606 | 1.75 |
| M3 | 0.642 | 0.89 |
| M4 | 0.704 | 9.92 |

Cuadro 3: Resultados en el conjunto de test para los distintos modelos entrenados.

En la Tabla 3 se observa una progresiva mejora en la precisión a medida que se introducen mejoras en la arquitectura y en el entrenamiento de los modelos. El modelo M0, como era de esperarse, presenta la peor performance en *accuracy*, al tratarse de una red básica sin mejoras de entrenamiento ni regularización, aunque cabe destacar que definitivamente la desventaja de este modelo son las 5000 epochs que necesita. Los modelos M1 y M2 incorporan optimizaciones como regularización, *dropout* y uso de *ADAM*, logrando una mejora mas que nada sobre el tiempo de entrenamiento, aunque limitada en resultados, principalmente por la simplicidad de su arquitectura. Esta simplicidad se debió al alto costo computacional asociado a realizar una búsqueda simultánea sobre hiperparámetros y arquitectura.

Por otro lado, el modelo M3, resultado de un *grid search* más exhaustivo sobre arquitecturas utilizando PyTorch y con los hiperparámetros previamente optimizados, logra un equilibrio destacable entre Accuracy (64.2 %) y baja loss de Cross Entropy (0.89), lo cual indica predicciones confiables y generalización adecuada. Finalmente, el modelo M4 alcanza la mayor *accuracy* (70.4 %), pero con una pérdida extremadamente alta (9.92), lo que evidencia un fuerte sobreajuste: el modelo memoriza los datos y, cuando se equivoca, lo hace con excesiva confianza. Esto implica que no generaliza correctamente fuera del conjunto de entrenamiento, a pesar del alto porcentaje de aciertos.

Como etapa final del trabajo, se empleó el mejor modelo obtenido (M3) para generar las probabilidades a-posteriori sobre un conjunto adicional de muestras provistas en `X_COMP.npy`. Cada fila del archivo resultante contiene la distribución de probabilidad estimada por el modelo para una muestra, reflejando su grado de certeza respecto a cada una de las 49 clases posibles. Este tipo de salida es útil en aplicaciones donde no solo importa la predicción más probable, sino también la confianza del modelo en su decisión.