
# I302 - Aprendizaje Automático y Aprendizaje Profundo

## Trabajo Práctico 3: Estudio de Redes Neuronales en clasificación multiclase

Ignacio Elian Gremes

12 de mayo de 2025

Ingeniería en Inteligencia Artificial

**Resumen**

En este trabajo se busco estudiar el rendimiento de una Red Neuronal para un problema de clasificación de imágenes. Se probaron distintas mejoras por separado, siendo estas, rate scheduling lineal saturado, y exponencial, uso de minibatch, optimizador ADAM en vez de GD, regularizacion L2 y early stopping. La Red Neuronal Base (M0) obtuvo una **cross entropy loss (CEL) de 1.8968** y una **accuracy (AC) de 0.6500**.Mientras la Red con mejoras (M1) obtuvo **CEL = 1.7046** y **AC = 0.6680**. Luego la Red M1 pero implementada en pytorch (M2) obtuvo las siguientes metricas **CEL = 1.8194** y **AC = 0.5740**. Ademas las metricas de M2 pero con la mejor cantidad de capas y de unidades encontradas (M3) se obtuvo **CEL = 1.4028** y **AC = 0.6840** . Por ultimo buscando overfittear el modelo simplemente modificando la cantidad de capas y de unidades (M4) se obtuvo **CEL = 1.9166** y **AC = 0.5360**

# 1. Introducción

En este trabajo se utilizo una Red Neuronal con imagenes de 28x28 de entrada y una salida que es la prediccion de que caracter chino es esa imagen, habiendo en total 49 caracteres distintos. Se arranco con una Red Neuronal basica denotada **M0**, luego se le aplicaron mejoras a esta y se la denoto **M1**. Se implemento M1 en pytorch y se la llama **M2**. Luego se modifico la cantidad de capas y de unidades para conseguir un mejor rendimiento y a esta red se la llama **M3**. Y por ultimo se busco una cantidad de capas y de unidades para overfitear la red y a esta se la llamo **M4**.

# 2. Métodos

## 1. Función de Costo: Cross-Entropy Multiclase

Para un problema de clasificación multiclase con $C$ clases, la función de costo cross-entropy se define como:

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

donde:

- $y_i$ es la etiqueta real (one-hot), es decir, $y_i = 1$ si la clase correcta es la $i$-ésima.
- $\hat{y}_i$ es la probabilidad predicha para la clase $i$, usando softmax:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

## 2. Cálculo de Accuracy

El accuracy se calcula como:

$$\text{Accuracy} = \frac{1}{N}\sum_{k=1}^{N} 1\left[\arg\max_i \hat{y}_i^{(k)} = \arg\max_i y_i^{(k)}\right]$$

## 3. Backpropagation

Sean $z^l$, $a^l$, $W^l$, $b^l$ las entradas, activaciones, pesos y sesgos de la capa $l$. La salida es $\hat{y} = a^L$.

### 3.1. Error en la capa de salida

$$\delta^L = \hat{y} - y$$

### 3.2. Propagación hacia atrás

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

### 3.3. Gradientes

$$\frac{\partial \mathcal{L}}{\partial W^l} = \delta^l (a^{l-1})^T, \quad \frac{\partial \mathcal{L}}{\partial b^l} = \delta^l$$

### 3.4. Regularización L2

Si usamos regularización L2 con parámetro $\lambda$:

$$\mathcal{L}_{\text{total}} = \mathcal{L} + \frac{\lambda}{2} \sum_l \|W^l\|^2$$

El gradiente se modifica como:

$$\frac{\partial \mathcal{L}_{\text{total}}}{\partial W^l} = \frac{\partial \mathcal{L}}{\partial W^l} + \lambda W^l$$

## 4. Actualización de Parámetros

### 4.1. Descenso por mini-batch

Se calcula el gradiente promedio sobre un conjunto pequeño de datos de tamaño $m$ (mini-batch).

### 4.2. Tasa de aprendizaje *(learning rate)*

- **Lineal con saturación**:

  $$\eta(t) = \text{máx}\left(\eta_{\text{mín}}, \eta_0 * (1 - t/t_{tot})\right)$$

  donde $\eta_0$ es la tasa inicial, $t$ la época actual y $t_{tot}$ la cantidad total de epocas.

- **Exponencial**:

  $$\eta(t) = \eta_0 \cdot k^t$$

  donde $k$ es el gamma decay.

### 4.3. Optimización con ADAM

ADAM adapta la tasa de aprendizaje individual por parámetro usando momentos:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla \mathcal{L}_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla \mathcal{L}_t)^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

donde:

- $\theta_t$ son los parámetros,
- $\beta_1, \beta_2$ son factores de decaimiento (por ejemplo, 0.9 y 0.999),
- $\epsilon$ es una constante pequeña para evitar división por cero.

## 5. Early Stopping

Durante el entrenamiento se monitorea la pérdida en el conjunto de validación. Se detiene el entrenamiento si no hay mejora durante $p$ épocas consecutivas (paciencia).

Esto previene el sobreajuste al conjunto de entrenamiento y mejora la generalización.

### 2.1. Conjunto de datos

Se utilizo un dataset de 5000 muestras , se dividio un 80 % en train, 10 % en val y 10 % en test. Y el valor mas alto que puede tener un pixel es 255, entonces se divdio todo por 255 para tener valores entre 0 y 1 por pixel.

### 2.2. Estructura de las Redes Neuronales

Para la red neuronal **M0** se utilizo una capa de entrad de 784 neuronas, 2 capas ocultas con activacion ReLU con 100 y 80 neuronas y una capa de salida con activacion softmax y 49 neuronas. Se utilizo un learn rate de 0.1 y 1500 epocas. Para **M1** se agrego una capa de 400 neuronas con activacion ReLU entre la capa de entrada y la de 100 neuronas. Se utilizo un learn rate de 0.001, rate schedule exponencial de gamma 0.98, optimizador ADAM con mini batch de 64, regularizaion L2 con decay 0.01, early stopping con paciencia 10 y 300 epocas. **M2** es igual a **M1** pero implementada con pytorch. En **M3** lo que cambia es la cantidad de capas y la cantidad de neuronas, teniendo esta 2 capas ocultas, una de 2000 neuronas y la otra de 1000 neuronas, todas con activación ReLU. Por ultimo **M4** tiene 3 capas ocultas de 1000 neuronas cada una con activación ReLU.

# 3. Resultados

## 3.1. Mejoras de una Red Neuronal

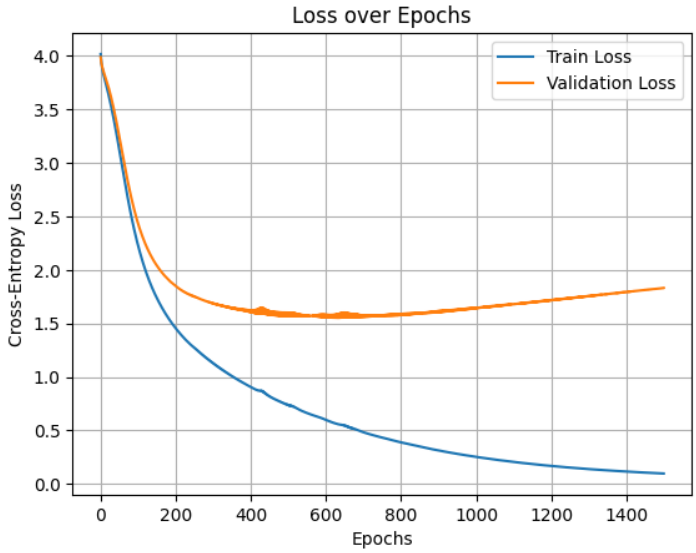



Figura 1: Cross Entropy Loss vs Epocas de la red **M0**.

Con esta Red se obtuvo un **CEL = 1.8312 y AC = 0.6220**. Esta va ser la red contra la que vamos a comparar las mejoras siguientes. Nos va a importar reducir la **CLE**.

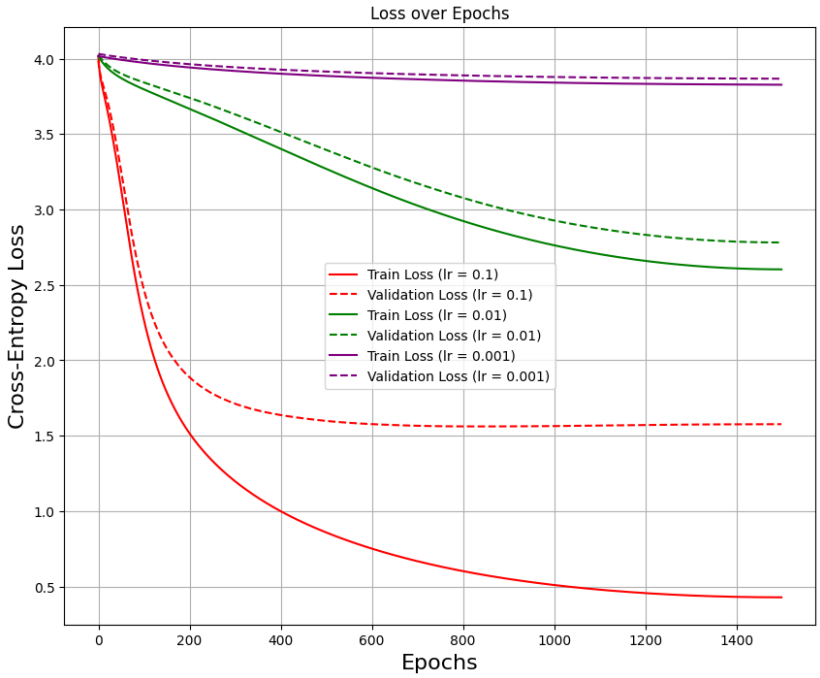
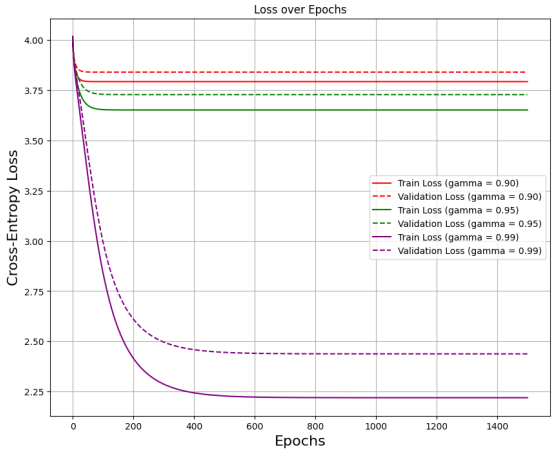
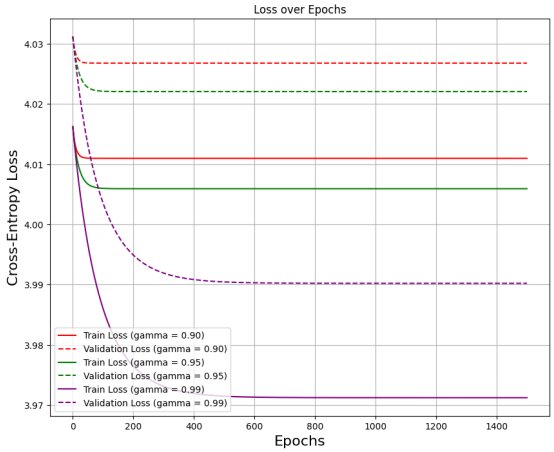



Figura 2: Cross Entropy Loss vs Epocas de la red **M0** con mejora de rate scheduling lineal .

Como se puede ver en la Figura 2 usar un lr = 0.1 es mas beneficioso para esta mejora, pero no hay una mejora notable en la CEL ya que el minimo de ambos esta cerca de 1.5. Lo que ayuda esta mejora es que reduce el overfit del set de entrenamiento a costo de necesitar una mayor cantidad de epocas para alcanzar el minimo CEL.




(a) Cross Entropy Loss vs Epocas de la red **M0** con mejora de rate scheduling exponencial con lr = 0.1 .

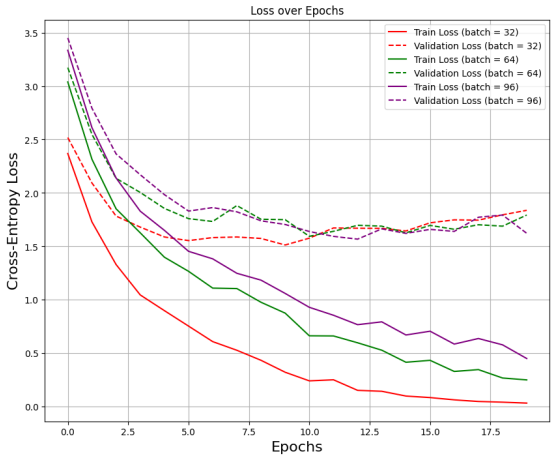



(b) Cross Entropy Loss vs Epocas de la red **M0** con mejora de rate scheduling exponencial con lr = 0.001 .

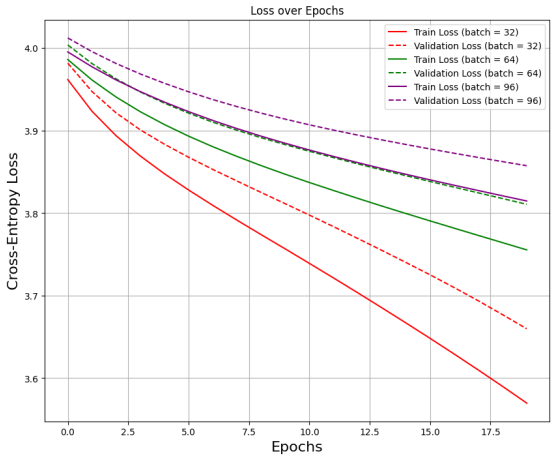
Figura 3

Como se puede ver el rate scheduling exponencial funciona mejor con un lr = 0.1. Se puede

notar una gran diferencia de CEL entre gamma = 0.95 y gamma = 0.99, por lo tanto para este problema conviene una gamma entre 0.95 y 0.99, idealmente mas cercano a 0.99.




(a) Cross Entropy Loss vs Epocas de la red **M0** con mejora de mini batch con lr = 0.1 .




(b) Cross Entropy Loss vs Epocas de la red **M0** con mejora de mini batch con lr = 0.001 .

Figura 4

Como se puede ver, mini batch con lr = 0.1 tiene menor CEL que con lr = 0.001. En la Figura 4a se puede ver que en validation todos los distintos batch size tiene un rendimiento parecido, salvo por el batch size 32 que tiene un mínimo menor a los otros. Otra cosa que se puede notar es el ruido que tiene la Figura 4a con respecto a la 4b. El beneficio principal de esta mejora es la velocidad que se gana en el entrenamiento, ya que se puede ver en la Figura 4a que se obtiene una CEL similar o incluso un poco menor a la de M0, pero esta se obtiene en tan solo 10 epocas. Una gran diferencia en comparación con las 500 épocas que le tomaba a M0 llegar a su mínimo de CEL.

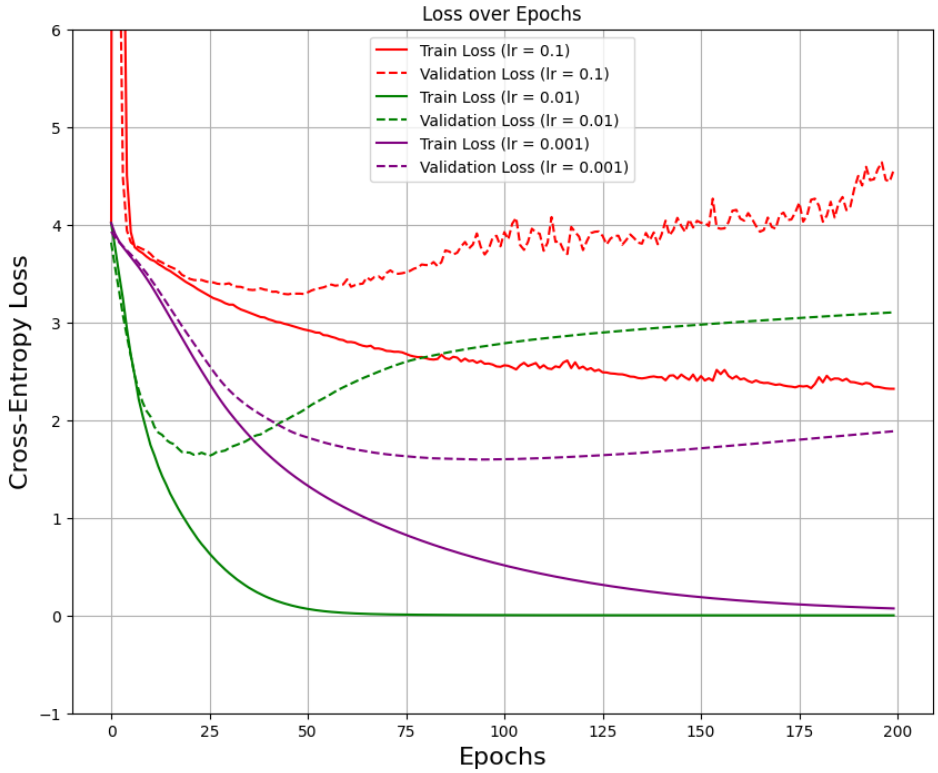
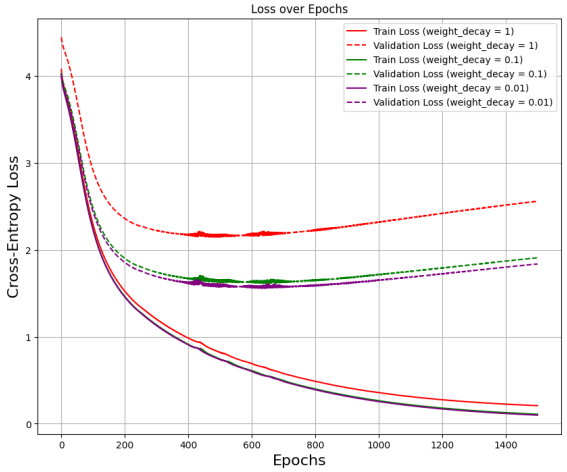
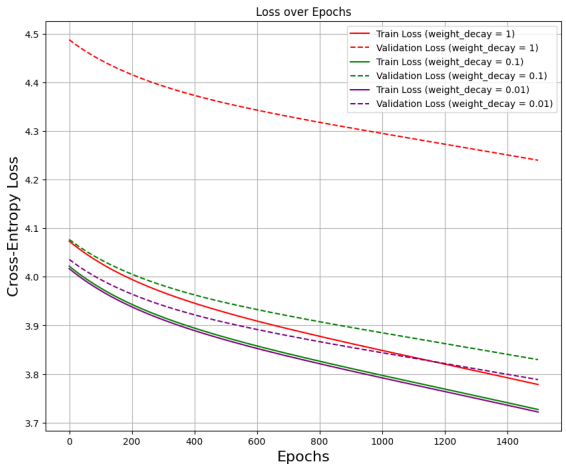



Figura 5: Cross Entropy Loss vs Epocas de la red **M0** con optimizador ADAM.

Como se ve en la Figura 5 el optimizador ADAM funciona mejor con lr = 0.001, aunque lr = 0.01 no es una mala opcion ya que tiene un minimo casi igual al lr = 0.001, por mas que a medida que aumentan la epocas, este empeore. Es muy claro que el funcionamiento de ADAM con lr = 0.1 es pesimo y tiene un monton de ruido. ADAM ayuda tambien a reducir el tiempo de entrenamiento sin sacrificar performance, por lo menos al utlizar lr = 0.001.




(a) Cross Entropy Loss vs Epocas de la red **M0** con regularizacion L2 con lr = 0.1 .




(b) Cross Entropy Loss vs Epocas de la red **M0** con regularizacion L2 con lr = 0.001 .

Figura 6

Como se puede notar en ambas Figuras 6a y 6b se obtiene mejores resultados al utilizar un weight decay de 0.01. Sin embargo no se consiguió una mejora notable en cuanto a la velocidad de entrenamiento ni en la perfomance de la red.

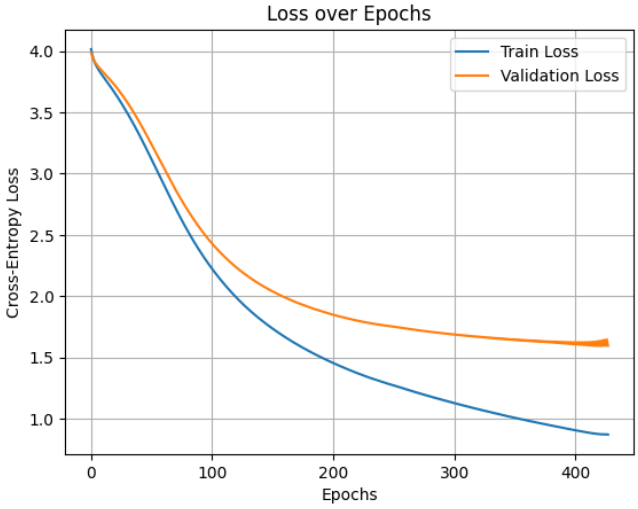



Figura 7: Cross Entropy Loss vs Epocas de la red **M0** con early stopping con paciencia 10.

Con esta mejora, se puede ver en la Figura 7 que la red deja de entrenar en cuanto la perfomance del validation empieza a empeorar, de esta manera se evita que el modelo empeore por entrenar de mas, como se puede ver en la Figura 1, la red alcanza su mejor rendimiento alrededor de la epoca 600, pero como sigue entrenando, su CEL empeora. En cambio con esta mejora se logra que el entrenamiento para cerca de la epoca 400 y logrando de esta forma obtener un mejor modelo que si entrenasemos las 1500 epocas, pero no alcanza su minimo

global que esta mas cerca de las 600 epocas, esto es algo que hay que saber calibrar con la paciencia de esta mejora.

### 3.2. Distintas estructuras de Redes Neuronales

En esta seccion se presentaran los resultados de CEL y de AC en las redes **M0**, **M1**, **M2**, **M3** y **M4** en el conjunto de test. Siendo **M0** la red basica sin mejoras, **M1** la red con mejoras seleccionadas en base al analisis de mejoras hecho en la seccion anterior, se eligio optimizador ADAM con lr = 0.001 debido a que se vio que es el que tiene mejores resultados como optimizador ya que ayuda a la perfomance y a la velocidad de entrenamiento, mini batch de tamaño 64, ya que no se noto una diferencia de performance notoria entre tamaño 32 y 64 para esta red, rate shceduling exponencial de gamma = 0.98 ya que se había visto que lo ideal es un valor entre 0.95 y 0.99, en este caso 0.98 tuvo un rendimiento mejor que 0.95 y 0.99. También se utilizo regularización L2 con decay = 0.01, que se había visto que era el que tenia mejor rendimiento y también se agrego early stopping con paciencia 10 para prevenir empeoramiento de la red por sobre entrenamiento y por ulitmo se agrego una capa de 400 neuronas. Por otro lado la red **M2** es identica a **M1** pero fue implementada en pytorch. En cambio **M3** tiene todas las mejoras mencionadas en **M1**, también fue implementada en pytorch, pero la cantidad de capas y de neuronas fue modificada. Inicialmente se fueron probando distintas cantidades de capas, y de las 3 capas ocultas (400,100,80) se agrego una de 700 antes de la de 400, pero el rendimiento empeoro drasticamente, por lo tanto se penso en disminuir la cantidad de capas y se probo (100,80) y se obtuvieron mejores resultados, con esto nos dimos cuenta que nuestra asunción que tuvimos en la red **M1** de que mas capas era mejor, fue errónea. Entonce se probo aumentar la cantidad de neuronas a (200,100) y mejoro, luego se probo una capa sola de 400 neuronas y mejoro, con esto teniamos la hipotesis de que menores capas y mas neuronas era mejor, entonces se utilizo una capa de 2000 neuronas y la mejora fue sustancial, pero luego se probo (2000,1000) y también mejoro. Con esto se concluyo que lo ideal para este problema son 2 capas, que mientras mas neuronas tengan mejor. Por cuestiones computacionales **M3** se utilizo con dos capas ocultas de 2000 y 1000 neuronas. Por ultimo para **M4**, que se busco overfittear el train, se utilizo 3 capas de 1000 neuronas cada una, se supuso que una mayor cantidad de capas ayudaria a overfitear, pero si se utilizan muchas neuronas, el rendimiento es muy malo y el train no llega ni a overfittear. Con esta estructura lo maximo a que se llego es a un **CEL = 0.6 y AC = 0.9** en el set de train, no esta completamente overfitteado pero es lo mejor que se obtuvo luego de muchas estructuras distintas

| | M0 | M1 | M2 | M3 | M4 |
|---|---|---|---|---|---|
| AC | 0.6500 | 0.6680 | 0.5740 | 0.6840 | 0.5360 |
| CEL | 1.8968 | 1.7046 | 1.8194 | 1.4028 | 1.9166 |

Cuadro 1: Resultados de accuracy (AC) y cross entropy loss (CEL) para diferentes modelos M0 a M4 en set de test

Como se esperaba el modelo con menor **CEL** termino siendo **M3**. Esto debido a que se implemento las mejoras y la mejor estructura encontrada, en este modelo. Por otro lado en cuanto comparamos **M1** con **M2**, a pesar de que ambos tienen la mismas mejoras y misma estrucutra, en pytorch, esta red performa peor, y esto era de esperar ya que pytorch al ser

una libreria profesional, tiene varias mas optimizaciones que permiten obtener un resultado mas realista de la red. Tambien se ve que el rendimiento de **M1** y **M2** es peor que el de **M3**, y esto tiene sentido teniendo en cuenta que no tienen la mejor estructura en cuanto a cantidad de capas y de neuronas. Por ultimo podemos ver que **M0** tiene peor rendimiento que los modelos anteriores, pero interesante mente, **M4** tiene menor **CEL**, a pesar de tener todas las mejoras mencionadas. Esto demuestra la importancia de no overfittear un modelo ya que puede empeorar drásticamente el rendimiento de una Red Neuronal.