Deep Learning: Tarea 1

Gabriel Alejandro Aguilar Farrera *

CIMAT Unidad Monterrey

gabriel.farrera@cimat.mx

18 de septiembre de 2023

Contenido

Resumen

Deep Learning.

I. Problema 1

Considera un conjunto de n datos de entrenamiento $\{x_i, y_i\}_{i=1}^n$, con $x_i \in \mathbb{R}^d$ y $y_i \in \mathbb{R}$. Ahora, considera ajustar un modelo lineal $\hat{y}_i = \mathbf{w}^{*'} x_i$, donde $\mathbf{w}^* = (w_i, \ldots, w_d)'$ es el vector de pesos. Supón que ajustas el modelo minimizando una función de costo apropiada (como MSE) con algún método de optimización. ¿Qué condiciones debe tener la matriz de datos $\mathbf{X}$ para que exista una solución única $\mathbf{w}^*$?

Solución

Esto es un problema de regresión lineal clásico. Dicho modelo se muestra a continuación (caso general):

$$\mathbf{Y}_{n \times 1} = \mathbf{X}_{n \times (r+1)} \mathbf{W}_{(r+1) \times 1} + \boldsymbol{\epsilon}_{n \times 1} \tag{1}$$

Donde $\mathbf{X}$ es la matriz de diseño y $\mathbf{W}$ es el vector de pesos. En nuestro caso, (r + 1) = d. Luego, hay un resultado que dice lo siguiente:

Resultado: Sea $\mathbf{X}$ de rango completo (todas sus columnas son linealmente independientes) $d \leq n$. El estimador de mínimos cuadrados de $\mathbf{W}$ en (1) está dado por:

*Estudiante de Maestría en Cómputo Estadístico

$$\mathbf{W}^* = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \tag{2}$$

Es decir, el requisito necesario para que exista una solución única $\mathbf{W}^*$ es que $\mathbf{X}$ sea de rango completo.
Si $\mathbf{X}$ no es de rango completo, $(\mathbf{X}\mathbf{X}')^{-1}$ es reemplazada por $(\mathbf{X}\mathbf{X}')^{-}$, una inversa generalizada de $\mathbf{X}\mathbf{X}'$.

II. Problema 2

Considera las redes multicapa (con funciones de activación lineal) que se muestran en la Figura (1):

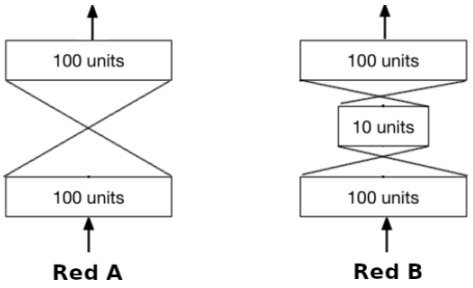


Figura 1: Esquema de dos redes neuronales.

- (a) Describe una ventaja (al menos) de la red A sobre la red B.
- (b) Describe una ventaja (al menos) de la red B sobre la red A.

Solución (a)

- No hay reudcción de la información original.
- **Simplicidad:** La red A es más simple, lo que significa menos parámetros para entrenar y menos complejidad computacional. Puede ser más rápida de entrenar y requerir menos recursos computacionales.
- **Interpretación:** Debido a su simplicidad y linealidad en todas las capas, la red A puede ser más interpretable. Es más fácil rastrear cómo las entradas se propagan a través de la red y cómo afectan a la salida.

Solución (b)

- **Mayor capacidad de representación:** La red B, al tener una capa oculta con potencial para introducir no linealidades, tiene una mayor capacidad para modelar relaciones no lineales en los datos. Puede ser más efectiva en tareas complejas.
- **Mejor generalización:** A pesar de la posibilidad de sobreajuste, la red B puede tener una mejor capacidad de generalización en comparación con la red A. Esto significa que puede adaptarse mejor a datos nuevos y desconocidos, lo que es fundamental en aplicaciones del mundo real.

III. Problema 3

¿Qué es Batch normalization? Escribe un reporte breve explicando qué es, para qué sirve y cómo se realiza. Incluye ejemplos ilustrativos (sencillos).

Solución

Comencemos por definir dos conceptos similares:

- **Normalización:** Colapsar las entradas para que tomen valores entre 0 y 1.
- **Estandarización:** Hacer que la media de los datos de entrada sea 0 y su varianza 1.

Ahora bien, Batch normalization (BatchNorm) o normalización por lotes, es una técnica utilizada en redes neuronales artificiales en donde normalizamos los valores con respecto al batch de las entradas, es decir, los valores de cada batch quedan re escalados de tal forma que ahora el valor más pequeño tome el valor 0 y el más grande tome el valor 1. En el caso de MNIST, las imágenes son en escala de grises, donde los valores de píxeles varían de 0 (negro) a 255 (blanco). Dividir por 255 escala estos valores en el rango de 0 a 1.

¿Por qué usamos Batch normalization?

- **Acelera la velocidad del entrenamiento:** E.g. tenemos dos variables (altura y edad de jugadores de la NBA), claramente la altura tiene un rango de valores más restringido (e.g. 0 a 2.5m) mientras que la edad tiene un rango más amplio (e.g. 0 a 100 años). Es claro que sin BatchNorm necesitamos ocupar un learning rate más pequeño mientras que con BatchNorm podemos usar un learning rate más grande (ver Fig. (2)) lo cual acelera la convergencia, es decir, el tiempo de entrenamiento es menor.

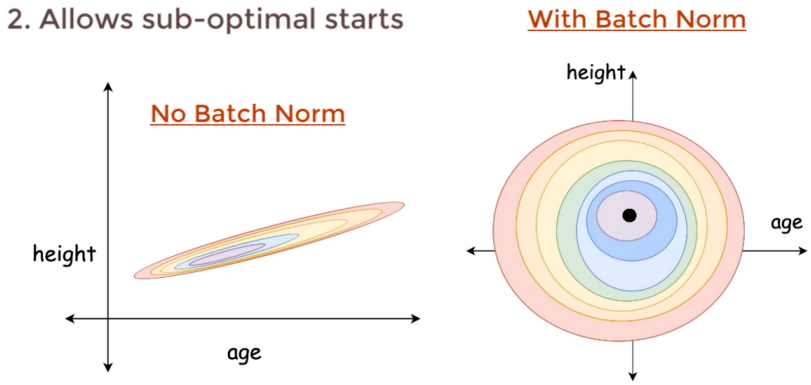


Figura 2: Función de costo: Sin batch normalization luce más alargada (izquierda) vs Con batch normalization ($\frac{x-\mu}{\sigma}$) luce más simétrica (derecha).

- **La importancia de los pesos iniciales decrece:** No importa mucho desde que punto comencemos, vamos a llegar al óptimo en un número similar de iteraciones (ver Fig. (2) derecha).
- **Regulariza un poco el modelo:** Al aplicar BatchNorm a todas las salidas de todas las capas de nuestra red (Ver Fig.(3)) ya no es necesario utilizar técnicas de regularización ya que BatchNorm ya lidia con este problema.

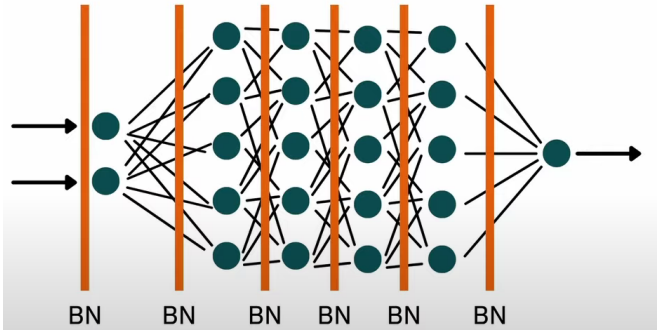


Figura 3: Batch Normalization en una red neuronal.

IV. Problema 4

Considera la regularización l_1 de una función de costo L que asumimos es continuamente diferenciable:

$$\tilde{L}(\mathbf{w};\mathbf{X},\mathbf{y}) = L(\mathbf{w};\mathbf{X},\mathbf{y}) + \alpha \sum_i |w_i|$$

1. Como en clase, considera una aproximación de segundo orden alrededor de $\mathbf{w}^*$, y muestra que la aproximación regularizada de $\tilde{L}$ es

$$\hat{L} = L(\mathbf{w}^*) + \sum_i \left(\frac{1}{2} H_{ii}(w_i - w_i^*)^2 + \alpha |w_i| \right),$$

   donde se asume que los datos están decorrelacionados (blanqueados), tal que $\mathbf{H}$ es una matriz diagonal con $H_{ii} > 0$.

2. Muestra que $\hat{L}$ se minimiza en

$$w_i = \text{signo}(w_i^*)\text{máx}\left\{ |w_i^*| - \frac{\alpha}{H_{ii}}, 0 \right\}$$

   ¿En qué casos tendremos soluciones *sparse* (donde $w_i = 0$)?

Solución 1.
La expansión por series de Taylor de segundo orden para aproximar una función de costo $L(\mathbf{w})$ al rededor de $\mathbf{w}^*$ tiene la siguiente forma:

$$L(\mathbf{w}) \approx L(\mathbf{w}^*) + (\mathbf{w} - \mathbf{w}^*)^T \nabla_{\mathbf{w}} L(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*) \tag{3}$$

Donde $\mathbf{H}$ es el Hessiano de L con respecto a $\mathbf{w}$ evaluado en $\mathbf{w}^*$.
Ahora bien, la aproximación regularizada de $\tilde{L}$ (usando la ecuación 3) y al final sumando el término de regularización se calcula como sigue:

$$\hat{L}(\mathbf{w}) = \tilde{L}(\mathbf{w}^*) + (\mathbf{w} - \mathbf{w}^*)^T \nabla_{\mathbf{w}} \tilde{L}(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*) + \alpha \sum_i |w_i| \tag{4}$$

Luego, como **H** es una matriz diagonal, es decir, todas sus entradas son cero a excepción de la diagonal principal, tenemos que:

$$\frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^T\mathbf{H}(\mathbf{w}-\mathbf{w}^*) = \frac{1}{2}\sum_i H_{ii}(w_i - w_i^*)^2 \tag{5}$$

Por otro lado, sabemos que

$$\nabla_{\mathbf{w}}\tilde{L}(\mathbf{w}) = \mathbf{H}(\mathbf{w}-\mathbf{w}^*) \tag{6}$$

Pero en la ecuación 4 tenemos $\nabla_{\mathbf{w}}\tilde{L}(\mathbf{w}^*)$, por lo que la ecuación anterior nos queda así:

$$\nabla_{\mathbf{w}}\tilde{L}(\mathbf{w}^*) = \mathbf{H}(\mathbf{w}^*-\mathbf{w}^*) = 0 \tag{7}$$

Finalmente, sustituyendo las ecs. (5) y (7) en (4) tenemos:

$$\begin{aligned}\hat{L}(\mathbf{w}) &= \tilde{L}(\mathbf{w}^*) + 0 + \frac{1}{2}\sum_i H_{ii}(w_i - w_i^*)^2 + \alpha\sum_i |w_i| \\ &= \tilde{L}(\mathbf{w}^*) + \sum_i \left(\frac{1}{2}H_{ii}(w_i - w_i^*)^2 + \alpha|w_i|\right)\end{aligned}$$

Solución 2. Tenemos que

$$\hat{L}(\mathbf{w}) = \tilde{L}(\mathbf{w}^*) + \sum_i \left(\frac{1}{2}H_{ii}(w_i - w_i^*)^2 + \alpha|w_i|\right) \tag{8}$$

Entonces bien, tomando la derivada de (8) con respecto a w_i, es decir, la derivada en cada dimensión i; tenemos que

$$\begin{aligned}\frac{\partial\hat{L}(\mathbf{w})}{\partial w_i} &= \frac{\partial}{\partial w_i}\tilde{L}(\mathbf{w}^*) + \frac{1}{2}\frac{\partial}{\partial w_i}\sum_i H_{ii}(w_i - w_i^*)^2 + \alpha\frac{\partial}{\partial w_i}\sum_i |w_i| \\ &= 0 + \frac{1}{2} * 2 * H_{ii}(w_i - w_i^*) + \alpha\ \text{signo}(w_i)\end{aligned}$$

Igualamos a cero:

$$\begin{aligned}H_{ii}(w_i - w_i^*) + \alpha\ \text{signo}(w_i) &= 0 \\ \Rightarrow H_{ii}(w_i - w_i^*) &= -\alpha\ \text{signo}(w_i) \\ \Rightarrow w_i - w_i^* &= -\frac{\alpha}{H_{ii}}\text{signo}(w_i) \\ \Rightarrow w_i &= w_i^* - \frac{\alpha}{H_{ii}}\text{signo}(w_i)\end{aligned}$$

Ahora consideremos los dos casos de signo (w_i):

- Si signo(w_i) = 1 (w_i es positivo):

$$w_i = w_i^* - \frac{\alpha}{H_{ii}} \tag{9}$$

- Si signo(w_i) = -1 (w_i es negativo):

$$w_i = w_i^* + \frac{\alpha}{H_{ii}} \tag{10}$$

Entonces la solución completa es

$$w_i = \text{signo}(w_i^*)\text{máx}\left\{|w_i^*| - \frac{\alpha}{H_{ii}}, 0\right\}$$

La solución *sparse* (w_i = 0) se obtiene cuando consideramos que w_i^* >0 y teniendo que $w_i^* \leq \frac{\alpha}{H_{ii}}$. Aquí tenemos que w_i = 0.

V. Problema 5

Para los datos de dígitos MNIST, realiza lo siguiente.

1. Usando **Pytorch**, implementa un baseline basado en regresión logística (multiclase) para la clasificación de dígitos. Reporta su desempeño con las métricas que creas conveniente.

2. Nuevamente, usando **Pytorch**, implementa redes neuronales con 1, 2 y 3 capas ocultas. ¿Qué mejoras obtienes en cada caso respecto al baseline? Reporta todos tus hallazgos y los parámetros que uses, incluyendo el número de unidades ocultas en la capa, el optimizador (uno solo para todos los modelos), el número de datos de entrenamiento, validación, tamaño del mini-batch, capas de regularización, etcétera.

3. Repite el inciso anterior usando al menos 2 métodos de optimización. Reporta las diferencias que encuentres.

Puedes usar el siguiente el Código (4) para crear los data loaders para entrenamiento y prueba según lo vimos en clase.

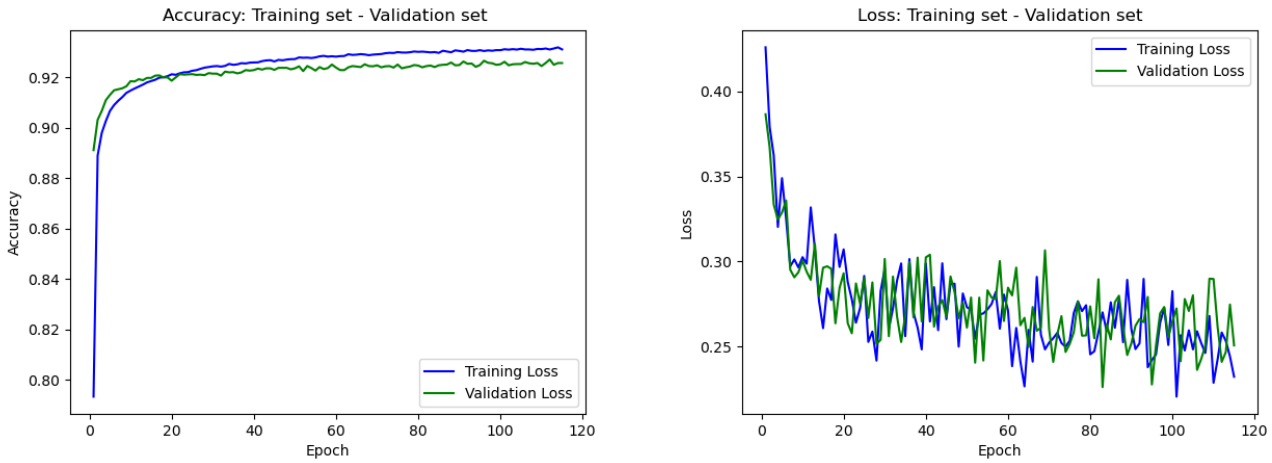
```python
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable

args={}
kwargs={}
# puedes poner estos parametros por separado, y cambiarles su valor...
args['batch_size']=1000
args['test_batch_size']=1000

#load the data
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args['batch_size'], shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=False, transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                       ])),
    batch_size=args['test_batch_size'], shuffle=True, **kwargs)

# Verificamos los datos del  dataloader
import matplotlib.pyplot as plt
_, (example_datas, labels) = next(enumerate(train_loader))
sample = example_datas[0][0]
# grafica
plt.imshow(sample, cmap='gray', interpolation='none')
print("Label: "+ str(labels[0]))
```

Figura 4: Código MNIST: Creación de data loaders para los datos MNIST.

Solución 1

Para este ejercicio se usó el optimizador SGD (Stochastic Gradient Descent), se usaron 60000 datos de entrenamiento, 10000 datos de validación y mini batch de 2000. A continuación se muestran los resultados usando un learning rate **lr = 0.1**. Por simplicidad a este modelo lo llamaremos "Modelo 0".

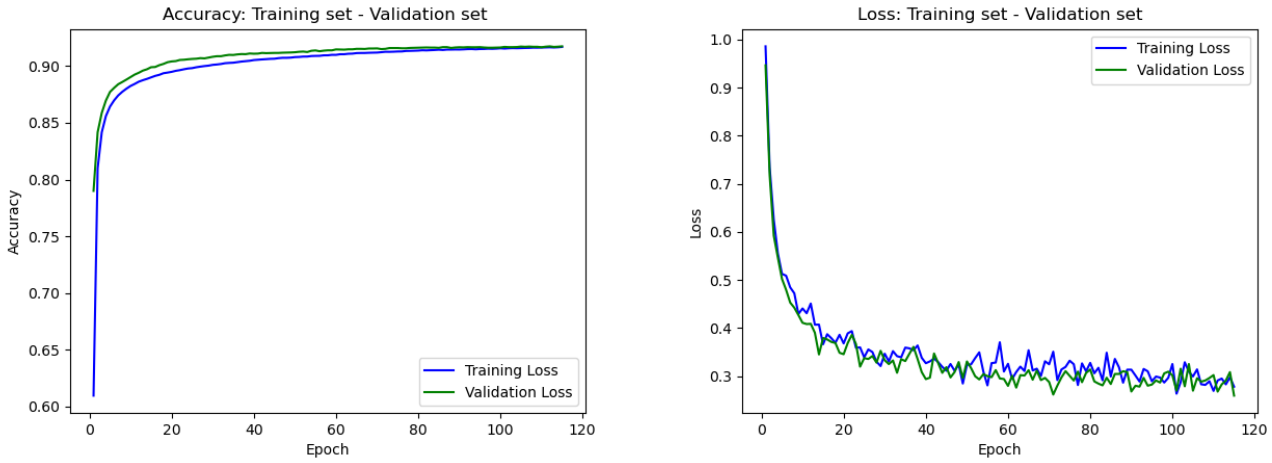
En la Fig. 5 (a) podemos observar que el accuracy en Training set es mayor que en Validation set. Otro aspecto importante a mencionar es que se ha alcanzado convergencia a partir de la época 40 (aprox.) y el accuracy en ambos conjuntos es bueno (mayor a 0.90). Por otro lado, en la Fig. 5 (b) observamos que la pérdida no es estable, después de la época 30 (aprox.) la pérdida en ambos conjuntos oscila entre 0.30 y 0.25 (aprox.), esto puede deberse a que el valor del learning rate no es el adecuado.



(a) **Accuracy:** (b) **Loss:**

Figura 5: Curvas de pérdida y accuracy obtenidas en **Modelo 0** con un **lr = 0.1**.

Ahora bien, los resultados obtenidos con un learning rate **lr = 0.01** son los siguientes (ver Fig. (6)). Aquí se puede observar que el accuracy converge rápidamente (al rededor de la época 40), pero con la diferencia que ahora el accuracy en Training y Validation es practicamente el mismo (aprox. desde la época 80 en adelante). La pérdida sigue siendo algo ruidosa, es probable que con al menos una capa oculta y un mayor número de neuronas la pérdida se logre estabilizar.



(a) **Accuracy:** (b) **Loss:**

Figura 6: Curvas de pérdida y accuracy obtenidas en **Modelo 0** con un **lr = 0.01**.

Solución 2

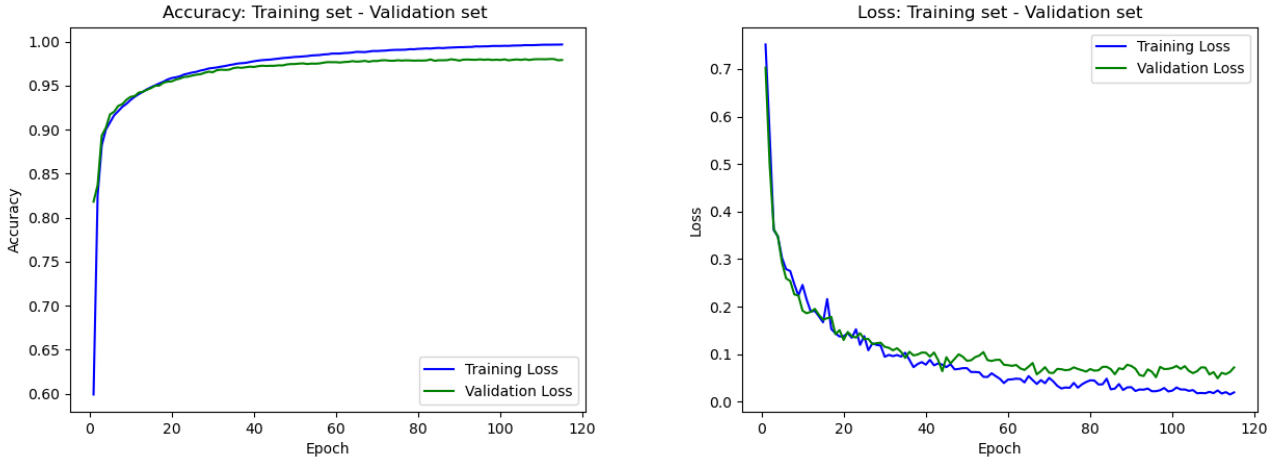
Por simplicidad vamos a llamar "Modelo 0" al pipeline basado en regresión logística (multi-clase), "Modelo 1" a la red neuronal con una capa oculta, "Modelo 2" a la red neuronal con dos capas ocultas y "Modelo 3" a la red neuronal con tres capas ocultas. En todos los casos, las funciones de activación que se utilizaron fueron funciones ReLU. A continuación se muestra una tabla con los respectivos parámetros ocupados en cada modelo.

	No. hidden layer	Optimizer	No. training set	No. validation set	mini-batch
Modelo 0	0	SGD	60000	10000	2000
Modelo 1	1	SGD	60000	10000	2000
Modelo 2	2	SGD	60000	10000	2000
Modelo 3	3	SGD	60000	10000	2000

Tabla 1: Parámetros usados durante el entrenamiento y en la etapa de validación.

A continuación se muestran las curvas de pérdida y accuracy obtenidas en los Modelos 1, 2 y 3. Primero mostramos los resultados obtenidos con un learning rate **lr = 0.1**.

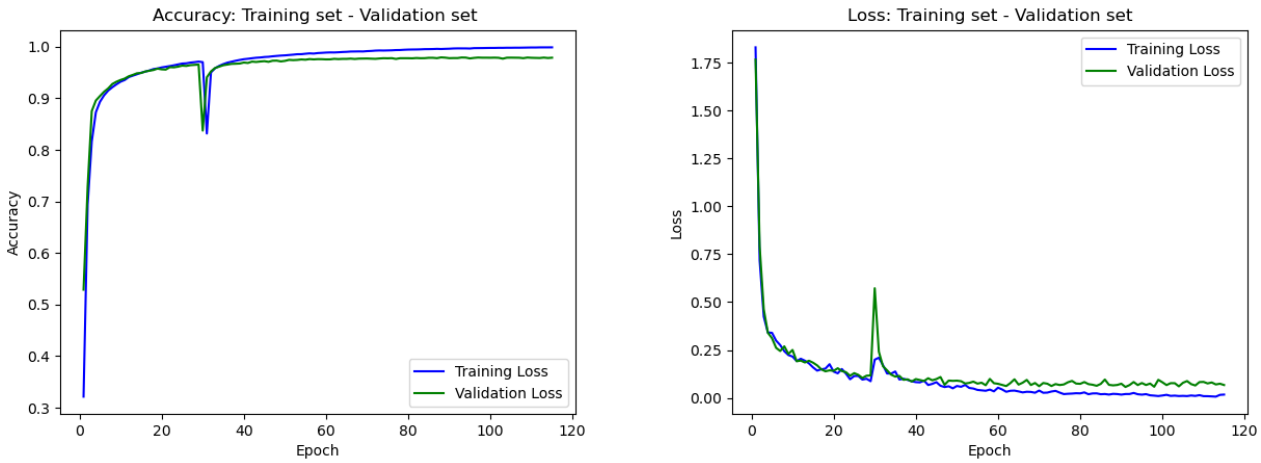
Modelo 1: En la Figura (7) se puede observar que el accuracy en Training y Validation está por encima de 0.95. Por otro lado, la pérdida se ve menos ruidosa que en el Modelo 0 (ver Figuras 5 y 6).



(a) **Accuracy:** (b) **Loss:**

Figura 7: Curvas de pérdida y accuracy obtenidas en **Modelo 1** con un **lr = 0.1**.

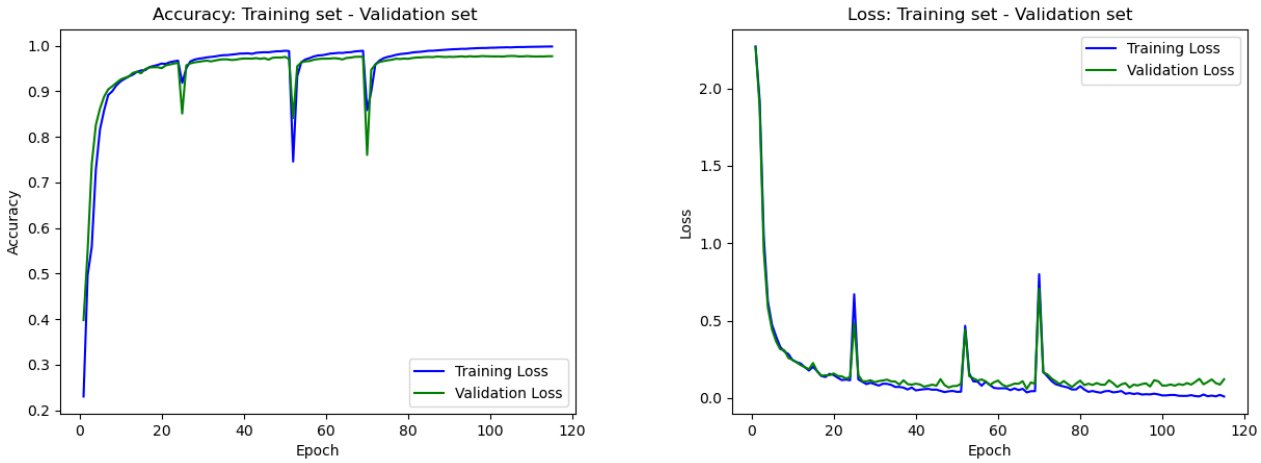
Modelo 2: En la Figura 8 (a) se puede ver que entre la época 20 y 40 el accuracy tuvo un "overfitting momentary drop" es decir, momentáneamente disminuyo su precisión, lo cual a veces es indicio de sobreajuste. Sin embargo, después el modelo se recupera y tanto el accuracy como la pérdida se vuelven a estabilizar. Es importante mencionar que la precisión en este caso es muy buena, es decir, siendo prácticamente 1 en Training y muy cercano a 1 en Validation.



(a) **Accuracy:** (b) **Loss:**

Figura 8: Curvas de pérdida y accuracy obtenidas en **Modelo 2** con un **lr = 0.1**.

Modelo 3: En la Figura 9 (a) se puede ver que entre la época 10 (aprox.) y 80 el accuracy tuvo tres casos de "overfitting momentary drop" es decir, momentáneamente disminuyo su precisión, lo cual a veces es indicio de sobreajuste. Sin embargo, después el modelo se recupera y tanto el accuracy como la pérdida se vuelven a estabilizar. Debido a que ahora el modelo tuvo 3 ocasiones de "momentary drop" en intervalos de tiempo cercanos quizá una mejor práctica sea ajustar un modelo más sencillo o variar el parámetro de learning rate, entre otras opciones.



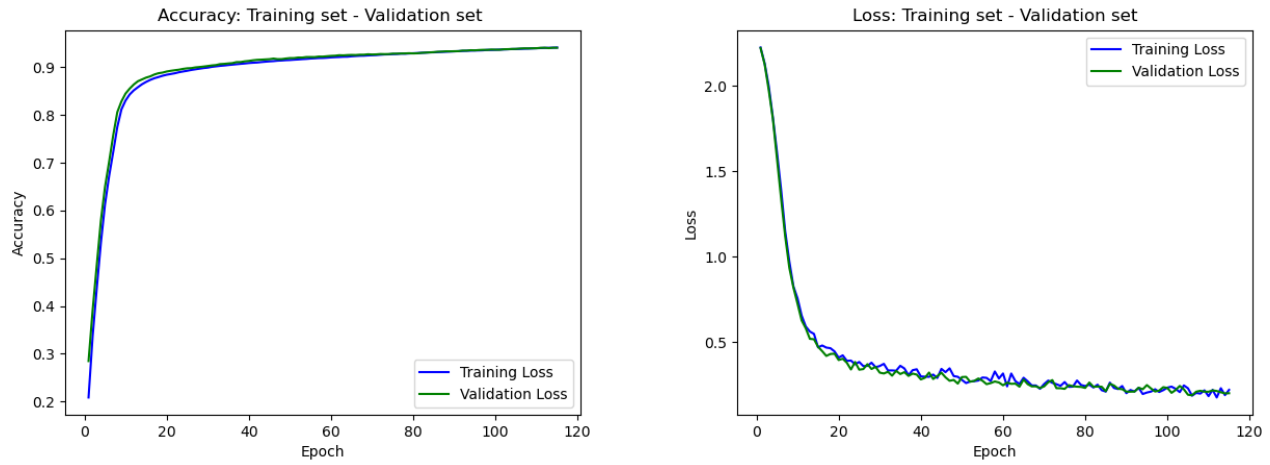
(a) **Accuracy:** (b) **Loss:**

Figura 9: Curvas de pérdida y accuracy obtenidas en **Modelo 3** con un **lr = 0.1**.

Ahora bien, ahora mostramos los resultados obtenidos con un learning rate **lr = 0.01**

Modelo 1: En la Fig. (10) se observan las curvas de accuracy y loss del modelo 1 con un learning rate de 0.01. Mientras que en la Fig. (7) se observan las curvas de accuracy y loss del modelo 1 con un learning rate de 0.1. Al hacer una comparación entre ambos modelos se puede concluir que es preferible el modelo 1 con lr de 0.01 ya que las curvas de accuracy en

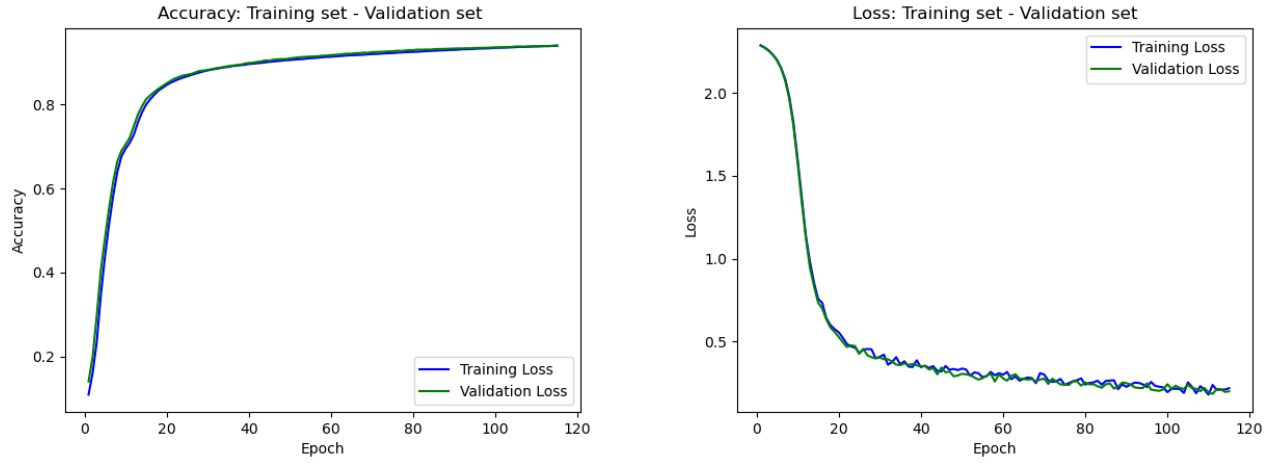
Training y Validation están más empalmadas, lo que quiere decir, que al disminuir el learning rate la capacidad de generalizar del modelo aumenta.



(a) **Accuracy:** (b) **Loss:**

Figura 10: Curvas de pérdida y accuracy obtenidas en **Modelo 1** con un **lr = 0.01**.

Modelo 2: En la Fig. (11) se observan las curvas de accuracy y loss del modelo 2 con un learning rate de 0.01. Mientras que en la Fig. (8) se observan las curvas de accuracy y loss del modelo 2 con un learning rate de 0.1. Al hacer una comparación entre ambos modelos se puede concluir que es preferible el modelo 2 con lr de 0.01 ya que las curvas de accuracy en Training y Validation están más empalmadas, lo que quiere decir, que al disminuir el learning rate la capacidad de generalizar del modelo aumenta. Otro aspecto importante es que al disminuir el learning rate ya no tenemos la presencia de "overfitting momentary drop".

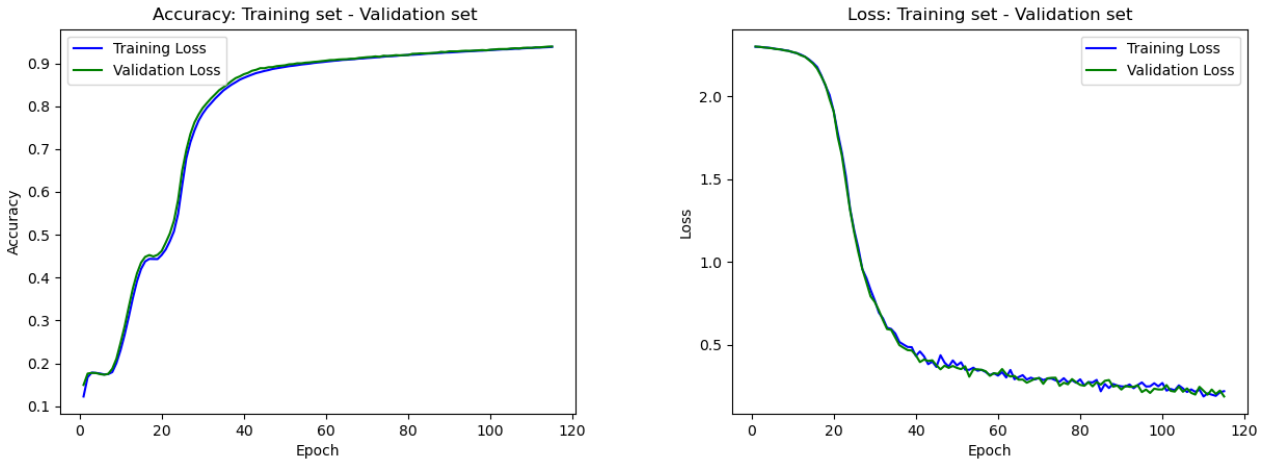


(a) **Accuracy:** (b) **Loss:**

Figura 11: Curvas de pérdida y accuracy obtenidas en **Modelo 2** con un **lr = 0.01**.

Modelo 3: En la Fig. (12) se observan las curvas de accuracy y loss del modelo 3 con un learning rate de 0.01. Mientras que en la Fig. (9) se observan las curvas de accuracy y loss del modelo 3 con un learning rate de 0.1. Al hacer una comparación entre ambos modelos se puede

concluir que es preferible el modelo 3 con lr de 0.01 ya que las curvas de accuracy en Training y Validation están más empalmadas, lo que quiere decir, que al disminuir el learning rate la capacidad de generalizar del modelo aumenta. Otro aspecto importante es que al disminuir el learning rate ya no tenemos la presencia de "overfitting momentary drop".

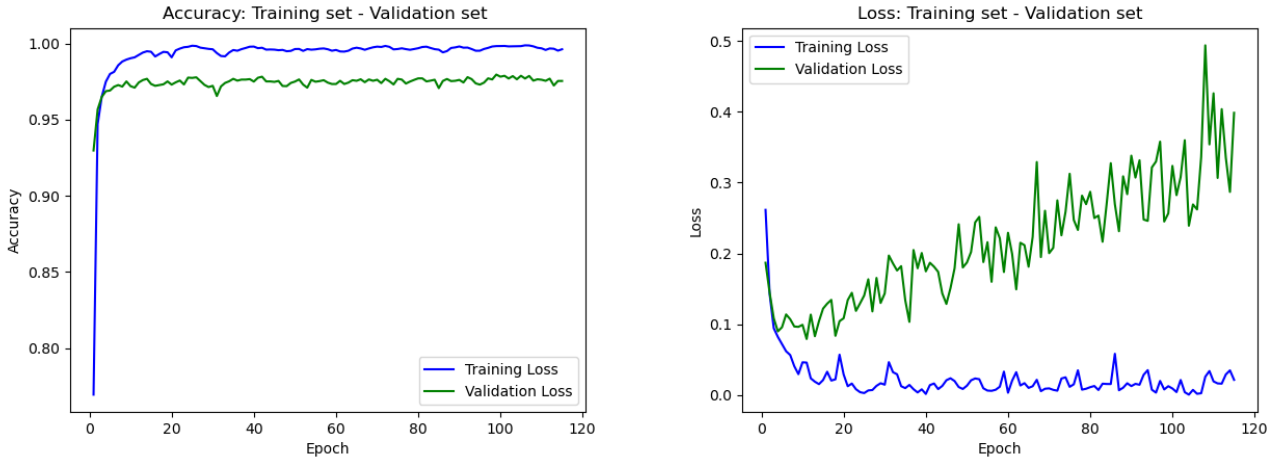


(a) **Accuracy:** (b) **Loss:**

Figura 12: Curvas de pérdida y accuracy obtenidas en **Modelo 3** con un **lr = 0.01**.

Solución 3
En el problema anterior se uso Stochastic Gradient Descent (SGD) como algoritmo de optimización. Ahora bien, vamos a considerar 2 algoritmos de optimización más: Adam y Adagrad. Vamos a evaluar el rendimiento y la velocidad de convergencia de ambos algoritmos de optimización. Los parámetros que se utilizaron son los mismos que los presentados en la Tabla (1). Del mismo modo, como se obtuvieron mejores resultados usando un learning rate de 0.01, entonces las soluciones aquí presentadas también se hicieron considerando este mismo valor de 0.01.

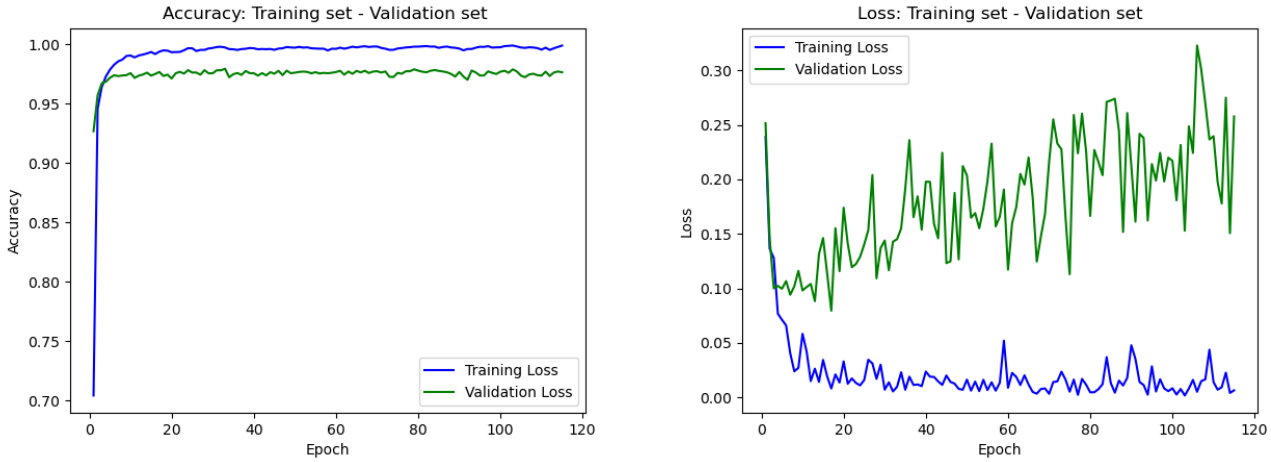
A continuación se muestran los resultados obtenidos con el algoritmo **Adam**.
Modelo 1: En la Fig. (13) podemos observar que la pérdida en el conjunto de validación no se estabiliza, de hecho va en aumento, sino que oscila en un rango de 0.1 a 0.5 aprox. Sin embargo, el accuracy en el conjunto de validación parece ser bueno (por encima de 0.95).



(a) **Accuracy:** (b) **Loss:**

Figura 13: Curvas de pérdida y accuracy obtenidas en **Modelo 1** con un **lr = 0.01** y **Adam** como algoritmo de optimización.

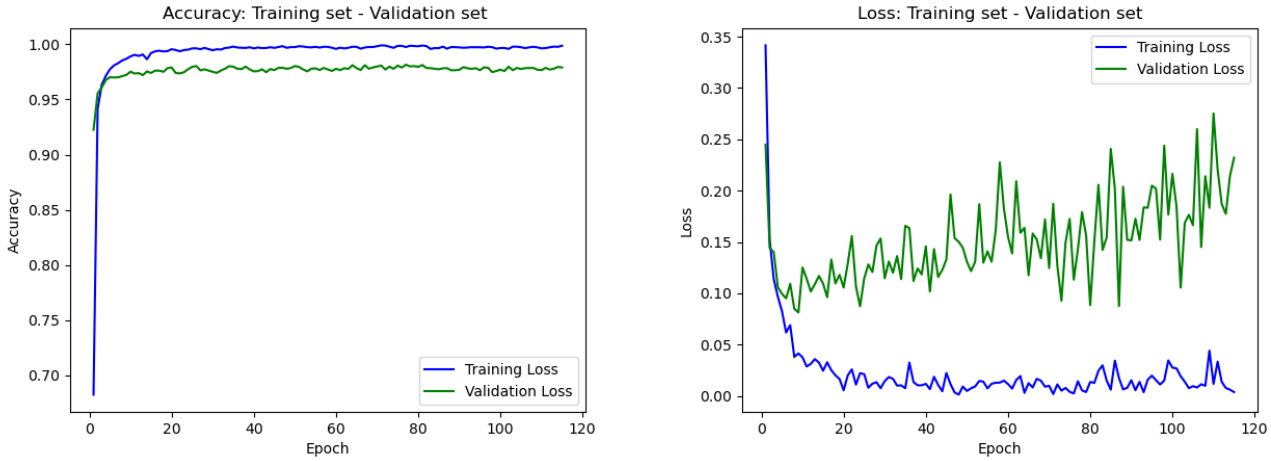
Modelo 2: En la Fig. (14) podemos observar que la pérdida en el conjunto de validación no se estabiliza (pero es menor que la pérdida considerado una sola capa oculta. Ver Fig. 13), sino que oscila en un rango de 0.10 a 0,30 aprox. Sin embargo, el accuracy en el conjunto de validación parece ser bueno (por encima de 0.95).



(a) **Accuracy:** (b) **Loss:**

Figura 14: Curvas de pérdida y accuracy obtenidas en **Modelo 2** con un **lr = 0.01** y **Adam** como algoritmo de optimización.

Modelo 3: En la Fig. (15) podemos observar que la pérdida en el conjunto de validación no se estabiliza (pero es menor que la pérdida considerado una sola capa oculta. Ver Fig. 14), sino que oscila en un rango de 0.10 a 0.25 aprox. Sin embargo, el accuracy en el conjunto de validación parece ser bueno (por encima de 0.95).



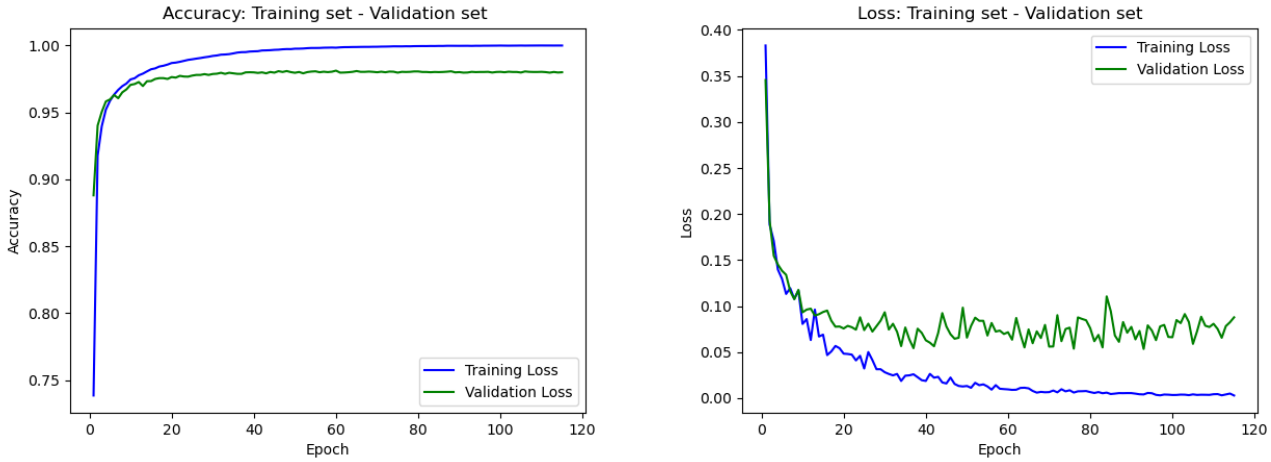
(a) **Accuracy:** (b) **Loss:**

Figura 15: Curvas de pérdida y accuracy obtenidas en **Modelo 3** con un **lr = 0.01** y **Adam** como algoritmo de optimización.

Hasta aquí podemos observar que conforme vamos aumentando el número de capas ocultas, la pérdida va oscilando en rangos cada vez más pequeños. También, notemos que para cualquiera de estos el accuracy es bueno (≥ 0.95) y la velocidad de convergencia es alta ya que para estos 3 casos a partir de la época 20 en adelante los cambios (en accuracy) ya no son significantes.

A continuación se muestran los resultados obtenidos con el algoritmo **Adagrad**.

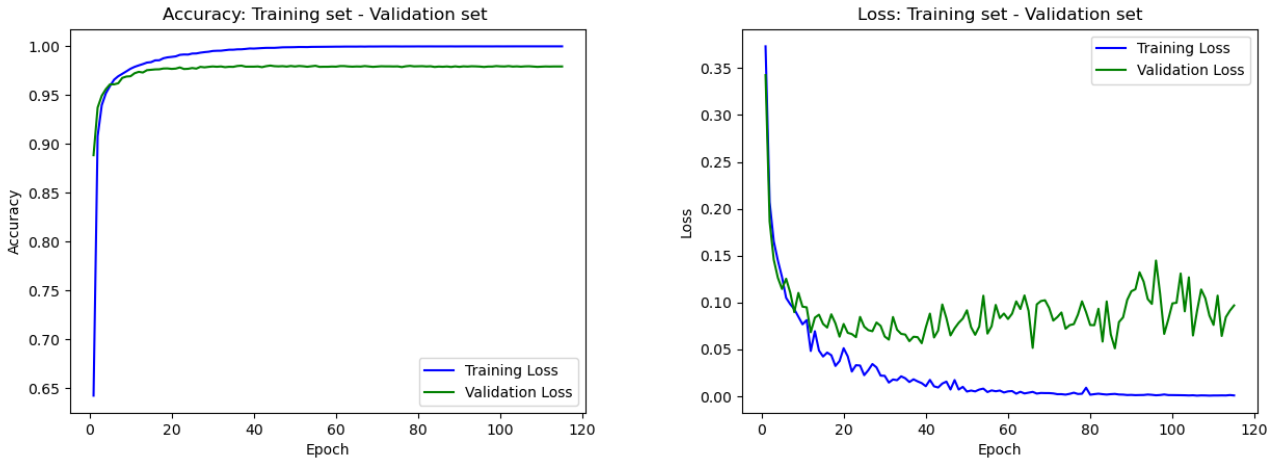
Modelo 1: En la Fig. (16) podemos observar que la pérdida en el conjunto de validación oscila entre valores muy pequeños (0.05 y 0.10 aprox). Por otro lado, el accuracy en el conjunto de validación parece ser bueno (por encima de 0.95).



(a) **Accuracy:** (b) **Loss:**

Figura 16: Curvas de pérdida y accuracy obtenidas en **Modelo 1** con un **lr = 0.01** y **Adagrad** como algoritmo de optimización.

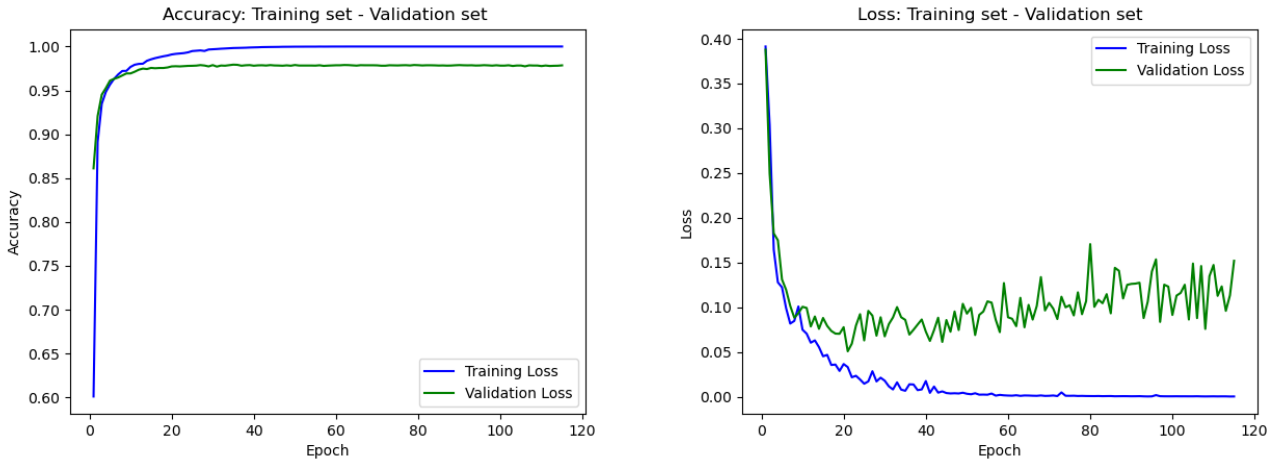
Modelo 2: En la Fig. (17) podemos observar que la pérdida en el conjunto de validación oscila entre valores pequeños (0.05 y 0.15 aprox). Por otro lado, el accuracy en el conjunto de validación parece ser bueno (por encima de 0.95).



(a) **Accuracy:** (b) **Loss:**

Figura 17: Curvas de pérdida y accuracy obtenidas en **Modelo 2** con un **lr = 0.01** y **Adagrad** como algoritmo de optimización.

Modelo 3: En la Fig. (18) podemos observar que la pérdida en el conjunto de validación oscila entre valores pequeños (0.05 y 0.15 aprox). Por otro lado, el accuracy en el conjunto de validación parece ser bueno (por encima de 0.95).



(a) **Accuracy:** (b) **Loss:**

Figura 18: Curvas de pérdida y accuracy obtenidas en **Modelo 3** con un **lr = 0.01** y **Adagrad** como algoritmo de optimización.

Conclusión final: Basándonos en criterios como velocidad de convergencia y capacidad de generalización a datos nunca antes vistos se prefiere el **Modelo 2** o incluso el **Modelo 3** con **lr = 0.01** y **Adagrad** como algoritmo de optimización. Los motivos son los siguientes:

- Las curvas de accuracy en Training y Validation observadas en la Fig. (18) y (17) son las más "**limpias**" es decir, no tienen prácticamente nada de oscilación en contraposición a las observadas en las Figs. (15) y (14), la cual corresponde al Modelo 3 con algoritomo Adam y lr de 0.01.
- La velocidad de covergencia es alta, a partir de la época 20 el accuracy ya no presenta cambios significativos.
- Buena capacidad de generelización: La diferencia entre el accuracy de Training y Validation es pequeña.