Universidad de O'Higgins

Tarea N° #2:
Informe #2

Sebastián Garrido

COM4402 – Introducción a Inteligencia Artificial
Escuela de Ingeniería, Universidad de O'Higgins
27, Octubre, 2023

I. Resumen

En este trabajo, se busca mediante el uso de redes neuronales, resolver un problema de clasificación de dígitos, analizando en aquellas redes su precisión, matriz de confusión, pérdida de datos, fidelidad, entre otros parámetros que ayuden a escoger la red con mejor validación.

II. Introduction

En este informe, se hará presente la ejecución, procedimiento, análisis de resultados, estudios, definiciones y conclusiones relacionados al trabajo de redes neuronales que fue realizado. Su lectura se basará en brindar un marco teórico que defina conceptos claves para esta temática, la metodología que explique lo realizado de forma programática, la revisión de resultados, el análisis del procedimiento y lo que se observó, las conclusiones generales de lo que se extrae a partir de lo realizado, su respectivo resumen, y finalmente la exposición de la bibliografía utilizada para hacer este informe.

III. Marco teórico

A. *Redes neuronales: "Una **red neuronal** es un modelo simplificado que emula el modo en que el cerebro humano procesa la información: Funciona simultaneando un número elevado de unidades de procesamiento interconectadas que parecen versiones abstractas de neuronas", esto también se puede apreciar cuando las redes neuronales son usadas para generar respuestas, cálculos, entre otras actividades que simulen el pensamiento humano mediante un proceso de aprendizaje profundo.*

B. *Overfiting: Un modelo neuronal en el que exista la presencia de Overfitting, será aquel donde se obtiene un error de entrenamiento relativamente bajo, y un error de validación relativamente alto.*

C. *Underfitting: Un modelo neuronal que tenga Underfitting como característica de sus datos, será aquel cuyos "errores tanto de entrenamiento como de validación son similares y relativamente altos"*

D. *Deep Learning: También conocido como aprendizaje profundo, corresponde a una forma de aprendizaje automático, "donde una máquina intenta imitar al cerebro humano utilizando redes neuronales artificiales con más de tres capas que le permiten hacer predicciones con una gran precision"*

E. *Perceptrón: Un perceptron resulta ser una neurona artificial, la cual efectúa cálculos para detectar tendencias o características en los datos de entrada. Estos se usan en gran medida para "clasificar datos, o como algoritmo para supervisar capacidades de aprendizaje de clasificadores binarios".*

F. *Función de activación: Tiene la labor de imponer un límite o modificar el valor resultado para poder proseguir a otra neurona. En otras palabras," es una función que transmite la información generada por la combinación lineal de los pesos y las entradas , es decir son la manera de transmitir la información por las conexiones de salida"*

G. *Back Propagation: "Es un algoritmo que se emplea para entrenar redes neuronales artificiales con el objetivo de minimizar los errores en el proceso de aprendizaje automático de una máquina"*

H. *Pérdida (loss): La función de pérdida evalúa la desviación entre las predicciones y cálculos realizados por la red neuronal, y los valores reales de las observaciones utilizadas durante el aprendizaje." Cuanto menor es el resultado de esta función, más eficiente es la red neuronal."*

I. *Matriz de confusion: Tiene la función de "valorar cómo de bueno es un modelo de clasificación basado en aprendizaje automático", caracterizándose principalmente en mostrar explícitamente cuando una clase se confunde con otra, permitiendo trabajar de forma separada con diferentes tipos de errores.*

J. *Accuracy: corresponde al porcentaje de clasificaciones correctas que un modelo de aprendizaje entrenado logra, dentro de todas las que ejecuta.*

K. *Epocs (épocas): "Es el número total de iteraciones de todos los datos de entrenamiento en un ciclo para entrenar el modelo de aprendizaje automático".*

IV. METODOLOGÍA

Considerando el Desarrollo de la tarea, cabe detallar en el sentido y explicación del código.

A. *Parte 1*

En esta parte del código (cedido con anterioridad para permitir el desarrollo de la tarea), se ejecutan todas las librerías cedidas y añadidas para realizar lo pedido, además de la lectura de datasets, dígitos, creación de datasets de entrenamiento, validación y prueba, su normalización de datos, y la inclusión de variables device y class_labels para la creación futura de modelos neuronales.

```
import pandas as pd
import torch
import torch.nn as nn
import numpy as np
import time
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score
```

B. *Parte 2*

La parte 2 de esta tarea consta en gran medida de crear las redes neuronales solicitadas, con la cantidad de capas ocultas, funciones de activación, números de épocas y cantidad de neuronas que se piden en cada modelo, a la vez que se cumplen con las indicaciones en cuestión.

En primer lugar, se empieza por crear el modelo, junto con la variable de early stopping para evitar resultados ambiguos en el entrenamiento.

```
model_a = nn.Sequential(
          nn.Linear(64,10),
          nn.ReLU(),
          nn.Linear(10,10)      #entrada vs datos de salida, 10 vs 10
        )
model_a = model_a.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_a.parameters(), lr=1e-3)


# Definir parámetros de early stopping
patience = 10
best_val_loss = float('inf')
no_improvement_count = 0
```

Luego, se procede a crear datasets y data loaders para optimizar y tener en cuenta lo necesario para parámetros de entrenamiento, validación y prueba.

```
[13] # Crear datasets
     feats_train = df_train.to_numpy()[:,0:64].astype(np.float32)
     labels_train = df_train.to_numpy()[:,64].astype(int)
     dataset_train = [ {"features":feats_train[i,:], "labels":labels_train[i]} for i in range(feats_train.shape[0]) ]

     feats_val = df_val.to_numpy()[:,0:64].astype(np.float32)
     labels_val = df_val.to_numpy()[:,64].astype(int)
     dataset_val = [ {"features":feats_val[i,:], "labels":labels_val[i]} for i in range(feats_val.shape[0]) ]

     feats_test = df_test.to_numpy()[:,0:64].astype(np.float32)
     labels_test = df_test.to_numpy()[:,64].astype(int)
     dataset_test = [ {"features":feats_test[i,:], "labels":labels_test[i]} for i in range(feats_test.shape[0]) ]

     # Crear dataloaders
     dataloader_train = torch.utils.data.DataLoader(dataset_train, batch_size=128, shuffle=True, num_workers=0)
     dataloader_val = torch.utils.data.DataLoader(dataset_val, batch_size=128, shuffle=True, num_workers=0)
     dataloader_test = torch.utils.data.DataLoader(dataset_test, batch_size=128, shuffle=True, num_workers=0)
```

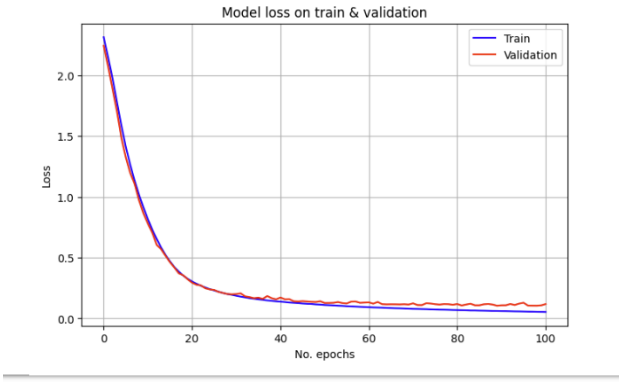
Posteriormente a aquello, se ejecuta una celda grande que entrena el modelo neuronal, definiendo librerías que almacenen datos de pérdida, épocas (definidas en 1000), y la misma función que ejecute el entrenamiento del modelo, y luego la validación de este, añadiendo además que incluye el procedimiento de early stopping dependiente de la variable pacience para evitar una desviación brusca de las variables de entrenamiento y validación.

```
start = time.time()

# Guardar resultados del loss y epocas que duró el entrenamiento
loss_train = []
loss_val = []
epochs = []
best_val_loss = float('inf')
best_epoch = 0
# Entrenamiento de la red por n epocas
for epoch in range(1000):                #Se definen épocas = 1000

  # Guardar loss de cada batch para cada parte de los datos
  loss_train_batches = []
  loss_val_batches = []


  # Entrenamiento ----------------------------------------------------------
  model_a.train() #se define para que empiece el entrenamiento
  # Se establece un for para recorrer cada batch en cada época y parte del conjunto de datos.
  for i, data in enumerate(dataloader_train, 0):
    # Procesar batch actual
    inputs = data["features"].to(device) # Características
    labels = data["labels"].to(device)   # Clases
    optimizer.zero_grad() #descenso del gradiente
    outputs = model_a(inputs)            # Predicciones de entrenamiento que obtuvo el modelo, se calcula la pérdida
    loss = criterion(outputs, labels) # Loss o pérdida de entrenamiento
    loss.backward()                   # Backpropagation
    optimizer.step()

    # Se guarda la pérdida de entrenamiento en el batch actual
    loss_train_batches.append(loss.item()) #sacar número específico a la pérdida de un 1 solo batch

  # se guarda el loss de entrenamiento de la época actual
  loss_train.append(np.mean(loss_train_batches)) # Loss promedio de los batches

  # Predicción en conjunto de validación -----------------------------------
  model_a.eval() #se le pasan los datos de validación para medir que tan bueno es
  #es similar al código superior, pero con datos de variación en vez de entrenamiento, sacando así predicción, probando la red, en vez de entrenar.
  with torch.no_grad():
    # Se itera dataloader_val para evaluar el modelo en los datos de validación
    for i, data in enumerate(dataloader_val, 0):
      # Procesar batch actual
      inputs = data["features"].to(device) # Características
      labels = data["labels"].to(device)   # Clases

      outputs = model_a(inputs)            # Obtenemos predicciones

      # Se guarda la pérdida de validación en el batch actual
      loss = criterion(outputs, labels)
      loss_val_batches.append(loss.item())

  # Se guarda el Loss de validación de la época actual y la época
  loss_val.append(np.mean(loss_val_batches)) # Loss promedio de los batches
  epochs.append(epoch)

  # Imprimir la pérdida de entrenamiento/validación en la época actual
  print(("Epoch: %d, train loss: %.4f, val loss: %.4f"  %(epoch, loss_train[epoch], loss_val[epoch])))

  # Se define early stopping en base al loss de entrenamiento y validación
  if loss_val[-1] < best_val_loss:
        best_val_loss = loss_val[-1]
        best_epoch = epoch
  elif epoch - best_epoch > patience:  # Detener si el loss de validación no mejora en relación de la variable patience
      break
end = time.time()
print('Finished Training, total time %f seconds' % (end - start))
```

A esta fase le sigue la visualización gráfica de los resultados del entrenamiento, en el que se le pone enfoque a la estabilidad de los resultados relacionando al número de épocas, con las pérdidas, haciendo enfoque en las variables de entrenamiento y validación.

```
import matplotlib.pyplot as plt

# Graficar loss de entrenamiento Y validación
plt.figure(figsize = (8, 5))
plt.title('Model loss on train & validation')
plt.xlabel('No. epochs')
plt.ylabel('Loss')
plt.plot(epochs, loss_train, 'b', label = 'Train')
plt.plot(epochs, loss_val, 'r', label = 'Validation')
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c9f4093ea40>
```

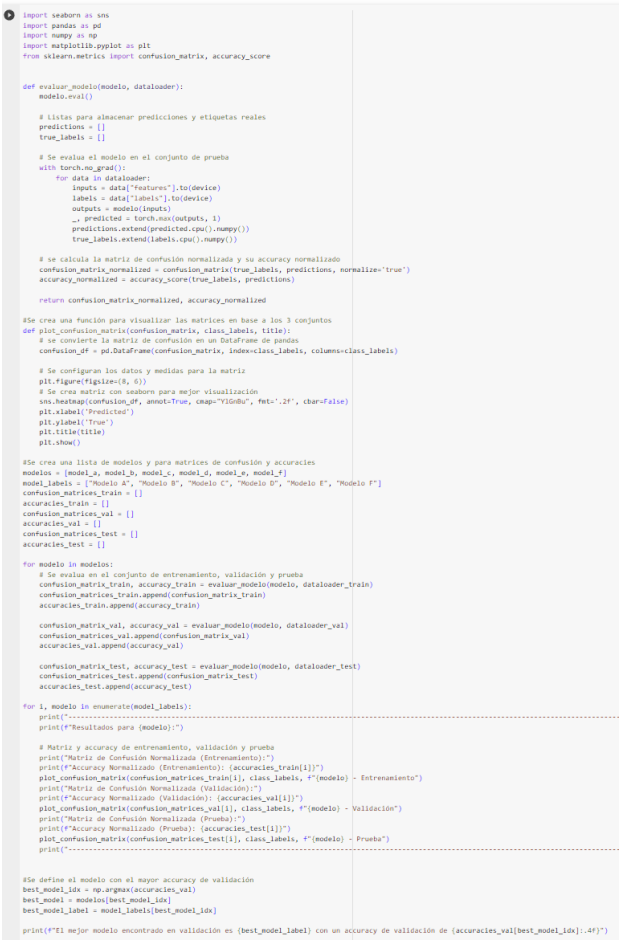


Finalmente, se ejecuta la última celda de código que guarda los resultados del modelo entrenado, y que en resumidas cuentas exhibe la matriz de confusión y los parámetros de accuracy tanto para el dataset de entrenamiento, como el de validación, arrojando 2 matrices y 2 valores de precisión respectivamente.

Esto se repite para los 6 modelos neuronales que se piden.

C. *Parte 3*

En esta parte, y con tal de hacer más argumentado los resultados pedidos, se ejecutan matrices y parámetros de accuracy para los 3 conjuntos por los cuales se rige lo solicitado, adjuntando al final la mejor red encontrada en la validación. (Con el detalle de hacer lo solicitado usando el conjunto de prueba para todos los modelos en vez de al mejor, siendo en todas las ocasiones probadas el modelo B).



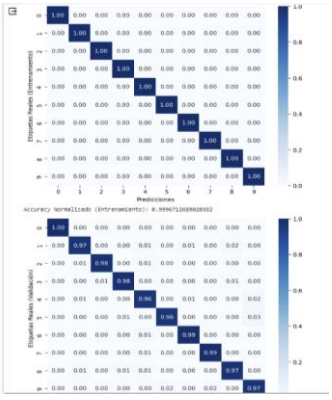
En adición, se añade este código un par de funciones for que evalúen los mejores resultados en los tres conjuntos de entrenamiento, validación y prueba respectivamente para facilitar la visualización de resultados

```python
# Se crea una lista de modelos y para matrices de confusión y accuracies
modelos = [model_a, model_b, model_c, model_d, model_e, model_f]
model_labels = ["Modelo A", "Modelo B", "Modelo C", "Modelo D", "Modelo E", "Modelo F"]
confusion_matrices_train = []
accuracies_train = []
confusion_matrices_val = []
accuracies_val = []
confusion_matrices_test = []
accuracies_test = []

# Se crean listas para el mejor modelo y su accuracy en cada conjunto
best_models_train = []
best_accuracies_train = []
best_models_val = []
best_accuracies_val = []
best_models_test = []
best_accuracies_test = []

for modelo in modelos:
    # Se evalua en el conjunto de entrenamiento, validación y prueba
    confusion_matrix_train, accuracy_train = evaluar_modelo(modelo, dataloader_train)
    confusion_matrices_train.append(confusion_matrix_train)
    accuracies_train.append(accuracy_train)

    confusion_matrix_val, accuracy_val = evaluar_modelo(modelo, dataloader_val)
    confusion_matrices_val.append(confusion_matrix_val)
    accuracies_val.append(accuracy_val)

    confusion_matrix_test, accuracy_test = evaluar_modelo(modelo, dataloader_test)
    confusion_matrices_test.append(confusion_matrix_test)
    accuracies_test.append(accuracy_test)

# Se almacena el mejor modelo y su accuracy en cada conjunto
for i, modelo in enumerate(model_labels):
    # Para el conjunto de entrenamiento
    if len(best_accuracies_train) == 0 or accuracies_train[i] > max(best_accuracies_train):
        best_models_train = [modelo]
        best_accuracies_train = [accuracies_train[i]]
    elif accuracies_train[i] == max(best_accuracies_train):
        best_models_train.append(modelo)
        best_accuracies_train.append(accuracies_train[i])

    # Para el conjunto de validación
    if len(best_accuracies_val) == 0 or accuracies_val[i] > max(best_accuracies_val):
        best_models_val = [modelo]
        best_accuracies_val = [accuracies_val[i]]
    elif accuracies_val[i] == max(best_accuracies_val):
        best_models_val.append(modelo)
        best_accuracies_val.append(accuracies_val[i])

    # Para el conjunto de prueba
    if len(best_accuracies_test) == 0 or accuracies_test[i] > max(best_accuracies_test):
        best_models_test = [modelo]
        best_accuracies_test = [accuracies_test[i]]
    elif accuracies_test[i] == max(best_accuracies_test):
        best_models_test.append(modelo)
        best_accuracies_test.append(accuracies_test[i])

# Se imprime o utiliza los resultados
print("Mejores modelos en el conjunto de entrenamiento:")
for i, modelo in enumerate(best_models_train):
    print(f"Modelo {modelo} con un accuracy en entrenamiento de {best_accuracies_train[i]:.4f}")

print("\nMejores modelos en el conjunto de validación:")
for i, modelo in enumerate(best_models_val):
    print(f"Modelo {modelo} con un accuracy en validación de {best_accuracies_val[i]:.4f}")

print("\nMejores modelos en el conjunto de prueba:")
for i, modelo in enumerate(best_models_test):
    print(f"Modelo {modelo} con un accuracy en prueba de {best_accuracies_test[i]:.4f}")
```

V. *Resultados*

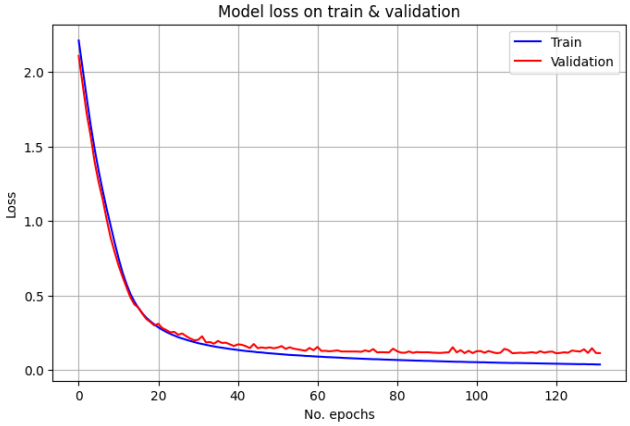
Considerando la metodología anteriormente explicada, ahora consta exhibir los resultados:

Respondiendo la pregunta 3, el mejor modelo encontrado en validación es Modelo B con un accuracy del conjunto de validación de 0.9785



Mejores modelos en el conjunto de prueba:
Modelo Modelo B con un accuracy en prueba de 0.9819
Respecto a la pregunta 4 se concluye lo siguiente:

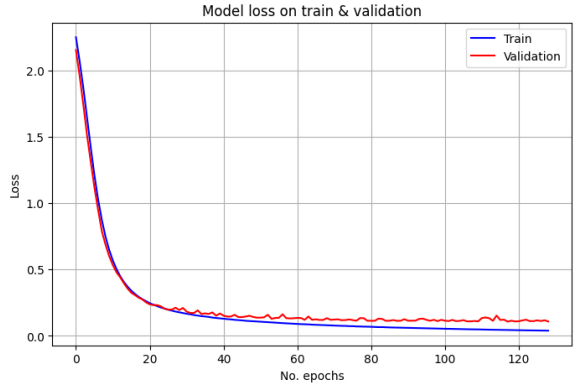
a) Los efectos de variar la cantidad de neuronas en la capa oculta puede causar que la red sea capaz de modelar relaciones más complejas en los datos de entrenamiento, lo que puede robustecer los resultados y la capacidad de ajuste a lo que se someten los datos. No obstante, a mayor cantidad, más riesgo hay de exponerse a fallos y variaciones no deseadas en el entrenamiento, produciéndose un sobreajuste de los datos, caso contrario, a menor cantidad, más riesgo hay de presentar subajuste, el cual empobrezca el rendimiento del proceso. Para evitar aquello, se introducen variables de early stopping y pacience como un estándar de tolerancia al error, los cuales detienen el entrenamiento para conservar su fidelidad. Como ejemplo, he aquí los resultados del modelo a, ordenado con 1000 épocas:



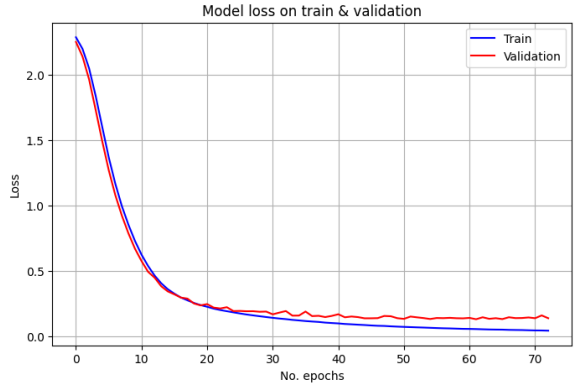
Como se puede observar, el entrenamiento se detuvo alrededor de las 130 iteraciones (131 exactamente), ya que se estaba presenciando un sobreajuste que excedía la tolerancia definida.

b) Los efectos de variar la cantidad de capas ocultas tienen injerencia en la capacidad del modelo para interpretar los datos de maneras más complejas o profundas. De tener más capas ocultas, se garantiza una resolución detallada y más exacta de problemas con datos complejos, mientras que el tener menos capas ocultas puede exhibir un subajuste en problemas con cierto grado de complejidad, al no capacitar al modelo de poder aprender con cierto nivel de profundidad/complejidad.
Para exhibir los efectos en el modelo, he aquí los gráficos de a y e:

Este es el caso de a



Este es el caso de e



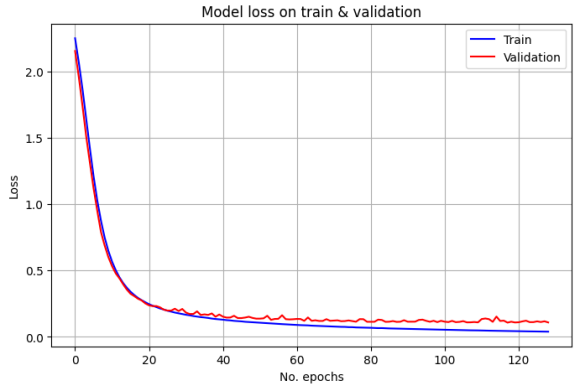
Como se puede apreciar, en el modelo de e, el entrenamiento y la validación difieren más tarde que en a, pero exceden el límite de error más temprano. No obstante, eso no quita que en a, se exija una menor complejidad a la hora de procesar los datos.

c) *El efecto de la función de activación radica en cómo la neurona responde a la suma ponderada de sus entradas, considerando dentro de las funciones de activación comunes a ReLU, Sigmoide y Tangente.*

En este caso particular, se observó el uso de la función de activación Relu, la cual se mueve en valores de entrada mayores o iguales a 0, y es eficaz para aprender patrones no lineales y se usan en redes neuronales profundas, y la función Tangente, que transofmra los valores entre -1 y 1, siendo útil cuando se necesita que las salidas se centren alrededor de 0.
Afecta al desempeño de la red en su capacidad de aprender y modelar modelos complejos, siendo ReLU lo más común considerando su efectividad.

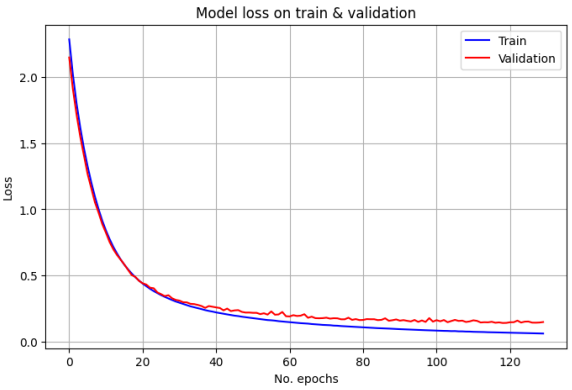
Como ejemplo, he aquí las matrices de los modelos a y c:

Caso a



Caso c

Como se puede apreciar, en el caso de la función de activación Relu, se observa una mayor robustez que

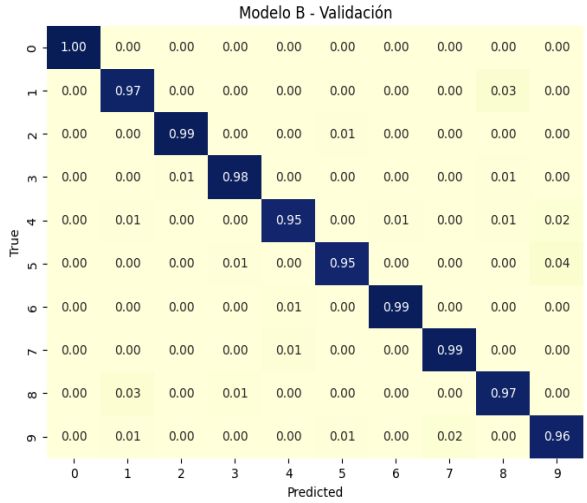


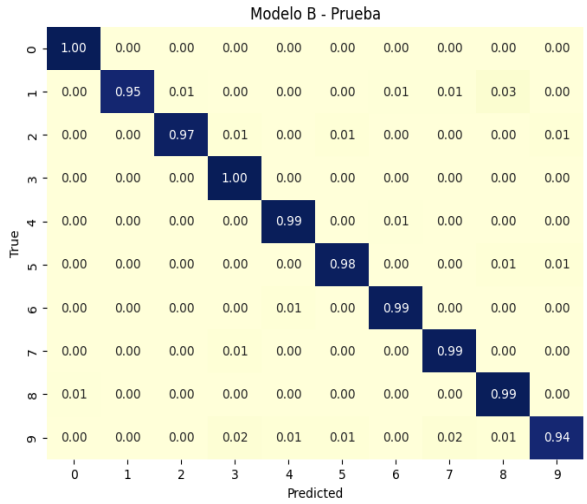
en el caso del uso de tangente, aunque las iteraciones finalizan en el mismo ciclo.

d y e) Considerando los siguientes datos arrojados por la ejecución del código:

Accuracy:
a validación: 0.967816091954023
prueba: 0.9685534591194969
b validación: 0.9762452107279693
prueba: 0.9819
c validación: 0.957088122605364
prueba: 0.9591194968553459
d validación: 0.9762452107279693
prueba: 0.9772012578616353
e validación: 0.9555555555555556
prueba:0.9567610062893082
f validación: 0.9739463601532568
prueba: 0.9772012578616353

Matriz de b en conjunto de validación y prueba:





Podemos determinar en el caso de la validación que, en todos los casos, tiene menos accuracy que en el caso de prueba, además, las matrices en el caso del modelo de prueba tienen una mayor robustez en su distribución.

VI. *Análisis*

Tomando en cuenta los resultados arrojados por la ejecución del código, se puede determinar lo siguiente

De todos los accuracy´s calculados, en el conjunto de validación el modelo B suele estar en todos los resultados presente, en otros, se suma el modelo D con un valor aproximado de 0.976.

El mejor modelo de prueba en cada una de las ejecuciones hechas resulta ser el modelo B, con un accuracy de 0.98 aproximadamente, o el cual suele estar sobre ese valor en la totalidad de los casos.

En otro aspecto, se puede rescatar que el uso de 2 capas ocultas con 40 neuronas y función ReLu como se puede observar en el modelo F, causa que sus iteraciones terminen habitualmente antes que el resto de modelos. Por otro lado, el mismo caso con menos neuronas como lo es el modelo E permite un incremento no significativo de épocas. Luego, podemos observar que el aumento de neuronas en la capa oculta, independiente de la función de activación (cuya influencia es mínima respecto del diferir de los valores de entrenamiento y validación) en el caso de los modelos B y D, aumenta la cifra de épocas sin causar tanta inestabilidad hasta su interrupción. Finalmente, en los modelos A y C se puede apreciar el mejor desempeño posible respecto al número de épocas realizadas, las cuales en todos los casos observados superan las 100 iteraciones.

Dicho esto, es correcto mencionar que, a pesar de producirse más iteraciones, el poseer más neuronas otorga un mayor accuracy en los modelos b, d y f

VII. *Conclusiones Generales*

En conclusion, es possible concluir que, incluyendo comandos que eviten overfitting o ajustes contraproducentes, el uso de capas ocultas suele ser contraproducente si se quiere obtener un resultado que tenga múltiples iteraciones respecto a las solicitadas. En otro aspecto, incluir más neuronas en la creación del modelo ayudará a tener un mejor parámetro de accuracy. Por ende, si bien se recomienda el uso de múltiples capas ocultas para cálculos muy complejos, es bueno probar con una sola capa oculta y un adecuado número de neuronas (mayor) para buscar resultados generalmente óptimos.

VIII. *Resumen*

En resumen, se puede observar en el trabajo realizado una rápida ejecución del modelamiento, para cualquiera de los 6 casos, aún así, se observa una ligera demora mayor en aquellos que tengan que lidiar con una mayor cantidad de capas ocultas, no así de más neuronas. Posteriormente, se requirió hacer todos los cálculos posibles respecto a los 3 conjuntos existentes para tener los resultados más fieles a lo solicitado, incluyendo matriz de confusión, y sus respectivas cifras de accuracy, dando como resultado al modelo b como el de mejor desempeño en la totalidad de las situaciones estudiadas.

IX. *Bibliografía*

1. ***[1]*** *IBM. "Redes neuronales (SPSS Modeler)". [En línea]. Disponible en: https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model. Accedido el 23 de octubre de 2023.*
2. ***[2]*** *CodificandoBits. "Underfitting y Overfitting: Conceptos Esenciales". [En línea]. Disponible en: https://www.codificandobits.com/blog/underfitting-y-overfitting/#:~:text=Un%20modelo%20con%20underfitting%20es,uno%20de%20validaci%C3%B3n%20relativamente%20alto.. Accedido el 23 de octubre de 2023.*
3. ***[3]*** *Datademia. "¿Qué es el Deep Learning y qué es una Red Neuronal?". [En línea]. Disponible en: https://datademia.es/blog/que-es-deep-learning-y-que-es-una-red-neuronal. Accedido el 23 de octubre de 2023.*
4. ***[4]*** *DataScientest. "Perceptrón: qué es y para qué sirve". [En línea]. Disponible en: https://datascientest.com/es/perceptron-que-es-y-para-que-sirve. Accedido el 23 de octubre de 2023.*
5. ***[5]*** *Telefonica Tech. "Función de Activación en Redes Neuronales". [En línea]. Disponible en: https://aiofthings.telefonicatech.com/recursos/datapedia/funcion-activacion#:~:text=Una%20funci%C3%B3n%20de%20activaci%C3%B3n%20es,por%20las%20conexiones%20de%20salida.. Accedido el 23 de octubre de 2023.*
6. ***[6]*** *Universidad Internacional de la Rioja. "Backpropagation". [En línea]. Disponible en: https://www.unir.net/ingenieria/revista/backpropagation/. Accedido el 23 de octubre de 2023.*
7. ***[7]*** *Telefonica Tech. "Cómo interpretar la matriz de confusión: Ejemplo Práctico". [En línea]. Disponible en: https://telefonicatech.com/blog/como-interpretar-la-matriz-de-confusion-ejemplo-practico. Accedido el 23 de octubre de 2023.*
8. ***[8]*** *Ediciones ENI. "Inteligencia Artificial Fácil: Machine Learning y Deep Learning Prácticos". [En línea]. Disponible en: https://www.ediciones-eni.com/libro/inteligencia-artificial-facil-machine-learning-y-deep-learning-practicos-9782409025327/la-prediccion-con-neuronas. Accedido el 23 de octubre de 2023.*
9. ***[9]*** *Iguazio. "Model Accuracy in Machine Learning". [En línea]. Disponible en: https://www.iguazio.com/glossary/model-accuracy-in-ml/#:~:text=AI%20accuracy%20is%20the%20percentage,is%20often%20abbreviated%20as%20ACC.. Accedido el 23 de octubre de 2023.*
10. ***[10]*** *Huawei Enterprise. "¿Qué es Epoch en Machine Learning?". [En línea]. Disponible en: https://forum.huawei.com/enterprise/es/%C2%BFQu%C3%A9-es-Epoch-en-Machine-Learning/thread/667232453749784577-667212895009779712. Accedido el 23 de octubre de 2023.*