Seminario de Machine Learning Estadístico: Proyecto Final

DNN & CNN para clasificación de la base de datos MINST

Daniel Robles Leong Samir Santoscoy Mario Rául Gúzman Gutierréz

Introducción: La base de datos MNIST (*Modified National Institute of Standards and Technology*) es una versión modificada de NIST, donde cada observación es un dígito del 0 al 9 escrito a mano (Figura 1) por los empleados del *American Census Bureau* y posteriormente el conjunto de prueba por estudiantes de secundaria y preparatoria. La nueva versión ya remuestreada y pre-procesada es ampliamente utilizada para medir por primera vez el desempeño de modelos de clasificación.[1]

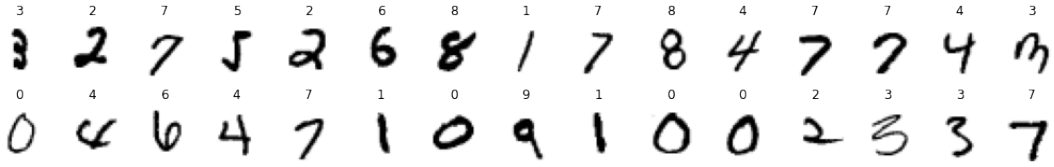

Figura 1: Ejemplo de la base de datos MNIST

Nuestro propóstio es utilizar las arquitecturas *Multilayer Perceptron* y *Convolutional Neural Network* para llevar a cabo la mejor predicción posible, usando como punto de comparación una Regresión Logística Multinominal.

Regresión Logística Multinominal (RLM): En nuestro primer acercamiento para la clasificación de datos consideramos una RLM con coding *softmax*, es decir, no hay categoría de referencia. [2]

Model	Loss	Accuracy	Error	Val Loss	Val Accuracy	Val Error
RLM	0.62	0.89	0.10	0.64	0.89	0.10

Cuadro 1: Errores aparentes y de prueba de la regresión logística

La aproximación tuvo como resultado la siguiente matriz de confusión:

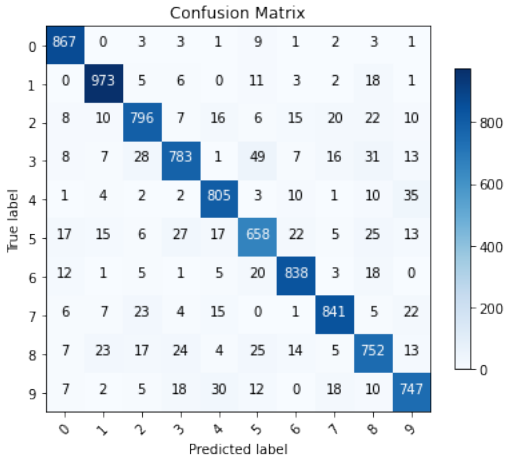

Figura 2: Matriz de confusión para la RLM

A pesar de haber predicho con exactitud un 89 % de nuestros datos, la aproximación queda muy por debajo incluso de nuestros primeros acercamientos por DNN, como veremos mas adelante. Como observación interpretaremos los resultados obtenidos.

Observemos como en la Figura 2, los errores de predicción están aparentemente distribuidos por cada una de nuestras categororías. Esto nos da la impresión que existe un error general en la clasificación, más allá de la complejidad de los datos a clasificar, como puede observarse en la Figura 1 hay dígitos facilmente reconocibles, mientras que otros presentan una orientación, escala o trazos exagerados que dificultan su semejanza al resto de elementos en su clase.

Multilayer Perceptron (MLP): Existen gran variedad de parámetros a modificar en la estructura *MLP*, un acercamiento que decidimos considerar, fue el elegir con cuidado nuestras funciones de activación, ya que modelan distintas estructuras a la hora de clasificar (Ver ANEXO 1).

Primero utilizando la misma arquitectura de red neronal [100, 50, 25] neuronas en cada capa oculta, variamos la funciones de activación, se consideraron 4 posibles candidatas: "*ReLu*", "*SeLu*", "*Tanh*" y "*Sigmoidal*" (Ver ANEXO 2), adicionalmente consideramos los métodos *Drop-out* para evitar un sobre ajuste y *Early-Stop* para detener el entrenamiento una vez que los valores ya no se reduzcan, considerando tambíen la cantidad de epocas que tardo en converger la arquitectura (Ver ANEXOS 3 y 4).

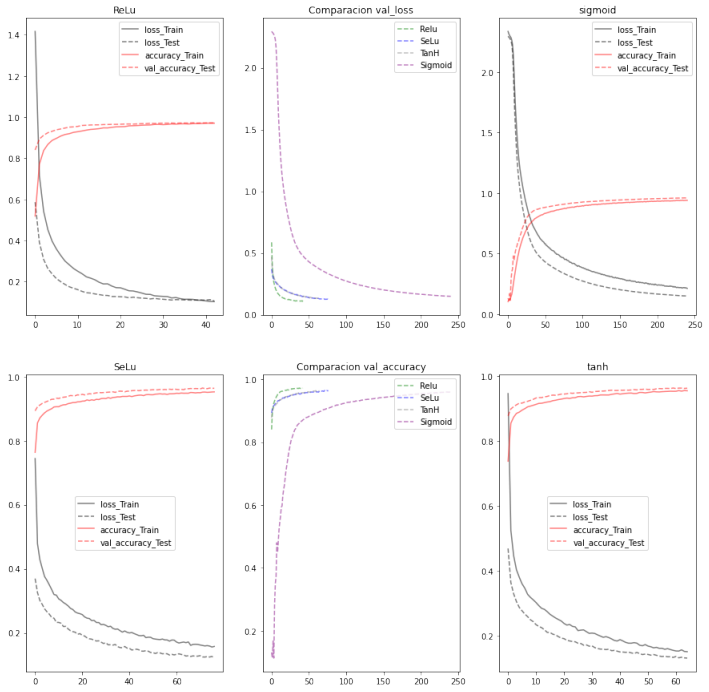

Figura 3: Comparación en funciones de activación

Model	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy	Epochs
MLP_{relu}	0.1121	0.9678	0.1126	0.9707	36
MLP_{selu}	0.1579	0.9522	0.1215	0.9637	77
MLP_{tanh}	0.1505	0.9552	0.1307	0.9632	65
$MLP_{sigmoid}$	0.2121	0.9391	0.1492	0.9594	241

Cuadro 2: Datos de convergencia por función de activación

Como podemos observar en ambos casos (Train / Validation) parece ser que la arquitectura con ReLu tiene costo minimo y exactitud máxima, dentro del conjunto de candidatos. Además su convergencia es significantemente más rapida como nos muestra el Cuadro 2, también presenta el menor sobre ajuste como puede observarse en su respectiva gráfica en la Figura 3

Name	T. Parameters	Dense L.	Nodes per Layer	Activation Functions	Dropout	Epochs
DNN1	182,660	3	[200, 100, 50]	[relu, relu, relu]	[0.4,0.4,0.4]	46
DNN2	168,000	4	[200, 40, 40, 40]	[relu, relu, selu, tanh]	[0.4,0.4,0.4,0.4]	31
DNN3	103,310	3	[120,50,50]	[relu, selu, relu]	[0.45,0.35,0.35]	22
DNN4	156,430	3	[180,60,60]	[relu, tanh, relu]	[0.45,0.35,0.25]	15
DNN5	142,960	4	[170,40,40,20]	[relu, relu, relu, sigmoid]	[0.45,0.20,0.20,0.15]	27
DNN6	162,750	3	[190,60,30]	[relu, selu, sigmoid]	[0.40,0.25,0.20]	22
DNN7	93,015	2	[110,55]	[relu, relu]	[0.40,0.35]	20
DNN8	183,685	4	[200,100,50,25]	[relu, relu, relu, tanh]	[0.40,0.35,0.30,0.30]	22

Cuadro 3: Estructuras de las DNN

Las arquitecturas fueron seleccionadas basandonos en el primer analisis sobre las funciones de activación, las capas con mas neuronas les fueron asociadas funciones de activación ReLu, las siguientes son variaciones de selu y tanh, por último consideramos sigmoidales, solo para la última capa escondida, los valores del Dropout tambíen se hizo considerando la cantidad de neuronas, en cada capa, y se considero una paciencia de 4 epocas para el EarlyStopping, las arquitecturas se presentan en el cuadro 3

Los resultados fueron los siguientes, ordenados de menor a mayor exactitud, en el conjunto de validación:

Model	Test loss	Test accuracy	Test error	Loss	Accuracy	Error
MLP8	0.113	0.975	0.024	0.030	0.992	0.007
MLP6	0.103	0.974	0.025	0.024	0.992	0.007
MLP1	0.112	0.972	0.027	0.029	0.992	0.007
MLP4	0.105	0.972	0.027	0.035	0.989	0.010
MLP5	0.112	0.972	0.027	0.033	0.990	0.009
MLP2	0.131	0.971	0.028	0.051	0.987	0.012
MLP3	0.121	0.970	0.029	0.039	0.988	0.011
MLP7	0.113	0.969	0.030	0.031	0.990	0.009

Cuadro 4: Tasas de error DNN

Parece ser que MLP8 es nuestro mejor modelo, observemos nuestros errores en la Figura 4 para realizar nuestros comentarios finales respecto a los MLP.

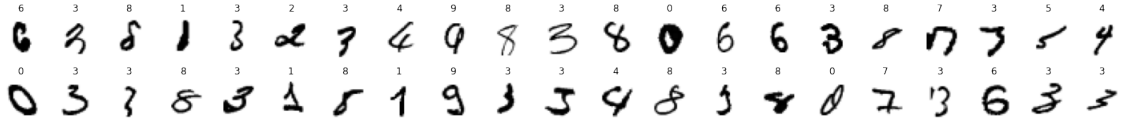

Figura 4: Errors in DNN8

Podmeos observar la complejidad de identificar estos elementos, es necesario realizar un análsis por secciones mas que una perspectiva general del digito, las MLP no pueden desarrollar este proceso, es tiempo de pasar a las CNN.

Convolutional Deep Neural Network (CNN): A diferencia de la "visión"que poseen las redes MLP, las CNN poseen capas especialmente diseñadas para el procesamiento de imágenes, que funcionan a través de operadores matemáticos que permiten segmentar la información y posteriormente comprimirla y aplanarla.

Sin embargo existe otra herramienta para el procesamiento de imágenes llamada *Maxpooling*, la cual se encarga de elegir el máximo en un arreglo de dimensiones específicas. Para la estructura de nuestros datos se usará *MaxPool2D*. (Ver ANEXO 5), se consideran las siguientes arquitecturas:

Name	T. Param.	Dens. L.	Conv. L.	Nodes per Layer	Activation Functions	Dropout	Epochs
CNN1	54,410	1	1	32	[relu]	[0]	18
CNN2	108,810	1	1	64	[relu]	[0]	10
CNN3	34,826	1	2	[32,64]	[relu, relu]	[0]	14
CNN4	34,826	1	2	[32, 64]	[relu, relu]	[0,0.5]	20
CNN5	34,826	1	4	[32,32,64,64]	[relu (x4)]	[0,0,0,0.5]	13
CNN6	143,978	1	5	[32,32,64,64,128]	[relu (x5)]	[0,0.2,0,0.3,0.5]	14
CNN7	241,546	1	4	[32,64,128,128]	[relu (x5)]	[0.2,0.3,0.3,0.5]	14

Cuadro 5: Estructuras de las CNN

Como podemos apreciar en el Cuadro 6 las CNN tienen un poder predictivo eficaz para este problema, nuestro mejor modelo CNN6 predijo con certeza el 99.1 % del conjunto de prueba y el 99.6 % el de entrenamiento, aun así, inclusive los algoritmos mas simples obtuvieron mejores resultados que nuestros MLP's.

Model	Test loss	Test accuracy	Test error	Loss	Accuracy	Error
CNN6	0.029	0.991	0.008	0.010	0.996	0.003
CNN5	0.029	0.991	0.008	0.012	0.995	0.004
CNN7	0.035	0.990	0.009	0.012	0.996	0.003
CNN4	0.041	0.988	0.011	0.018	0.994	0.005
CNN3	0.054	0.984	0.015	0.015	0.996	0.003
CNN1	0.072	0.980	0.019	0.015	0.996	0.003
CNN2	0.071	0.980	0.019	0.024	0.993	0.006

Cuadro 6: Tasas de error CNN

Como observación final, podemos apreciar en la Figura 5 los dígitos que nuestro mejor modelo (CNN6) no pudo predecir, cabe resaltar que ninguno es sencillamente identificable ya que bien podrían pertenecer a dos clases.

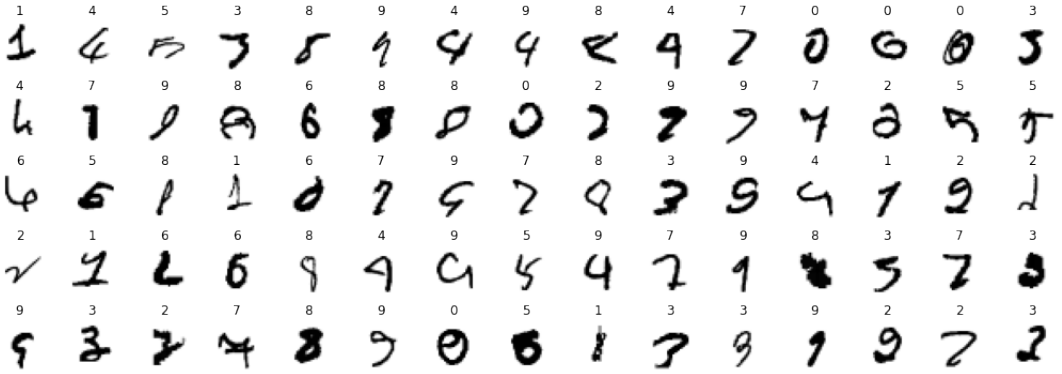

Figura 5: Errores de la CNN5

Estrategia A lo largo de toda la modelación, tanto para MLP como para CNN, la principal estrategia fue através de ensayo/error, una vez familiarizados con el comportamiento de las redes y los datos se optó por fijar una arquitectura, para el caso de las MLP, e ir testeando diversas funciones de activación. Por otro lado se fue variando el optimizer de "sgd" a "adam", concluyendo que en promedio "adam" minimizaba los errores. Para CNN la estrategia fue partir de la arquitectura de la API de Keras, entenderla y experimentar con las capas convolucionales y el Pooling nosotros mismos. De nuevo a través de ensayo/error se seleccionaron los mejores modelos.

Model	Test loss	Test accuracy	Test error	Loss	Accuracy	Error
CNN6	0.029	0.991	0.008	0.010	0.996	0.003
MLP8	0.113	0.975	0.024	0.030	0.992	0.007
RLM	0.62	0.89	0.10	0.64	0.89	0.10

Cuadro 7: Tasas de error CNN

Dígito	0	1	2	3	4	5	6	7	8	9
CNN6	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
MLP8	0.99	0.99	0.97	0.97	0.97	0.96	0.99	0.98	0.95	0.97
RLM	0.93	0.93	0.89	0.89	0.90	0.83	0.92	0.92	0.84	0.87

Cuadro 8: Errores por categoría en los modelos seleccionados

Conclusiones Los aprendizajes fueron bastante diversos para todos los miembros del equipo, empezando por el manejo del software mismo y hasta conceptos teóricos de las redes. Por ejemplo, nos dimos cuenta que ajustar más de 3 capas o meter más de 200 neuronas sobreajustaba el modelo, incluso correr un número elevado de épocas no significará una mejor generalización, aspectos como la aproximación que dan distintas funciones de activación, en general las DNN presentan aproximaciones interesantes a como percibimos y aprendemos, hay mucho que se puede reflexionar al respecto y aplicar a otros problemas similares a la experiencia sensorial humana.

ANEXO

1. Variación en la función de activación Las funciones de activación modelan distintas estructuras en su forma de clasificar los datos, como puede observarse en la Figura 6

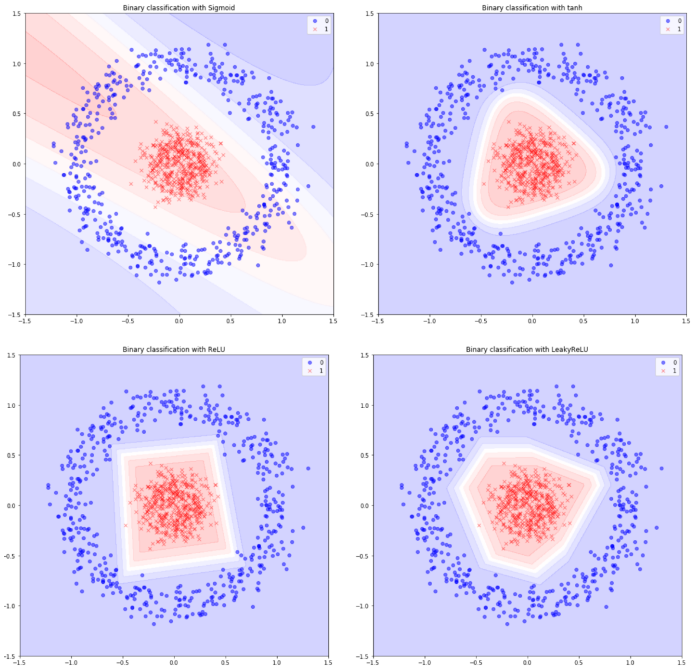

Figura 6: Variación de la clasificación binaria por función de activación

2. Funciones de Activación a considerar

- **Rectified Linear Units** (ReLu)

$$ReLu(x) = \max_{x}(0, x)$$

- **Scaled Exponential Linear Unit** (SELU)

$$SeLu(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

  donde: $\alpha = 1.67326324$ y $\lambda = 1.05070098$ por default.[3]

- **Hyperbolic Tangent** (TanH)

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

- **Sigmoid or Logistic Activation Function** (sigmoid)

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

3. Drop-out Method La capa Dropout [4] establece aleatoriamente las unidades de entrada en 0 con una frecuencia de velocidad en cada paso durante el tiempo de entrenamiento, lo que ayuda a evitar el sobreajuste. Las entradas no establecidas en 0 se escalan en $\frac{1}{1-tasa}$ de modo que la suma de todas las entradas no cambia.

4. EarlyStopping [5]
Asumiendo que el objetivo de un entrenamiento es minimizar la pérdida. Con esto, la métrica a monitorear sería 'loss', y el modo sería 'min'. Un bucle de entrenamiento model.fit() verificará al final de cada época si la pérdida ya no está disminuyendo, considerando el min_{delta} y la *patience*. Una vez que se descubre que ya no disminuye, $model.stop_training$ marca $True$ y el entrenamiento termina.

5. Maxpooling [6]
Reduce la muestra de la entrada a lo largo de sus dimensiones espaciales (altura y anchura) tomando el valor máximo sobre una ventana de entrada (de tamaño definido por $pool_size$) para cada canal de la entrada. La ventana se desplaza por zancadas a lo largo de cada dimensión.

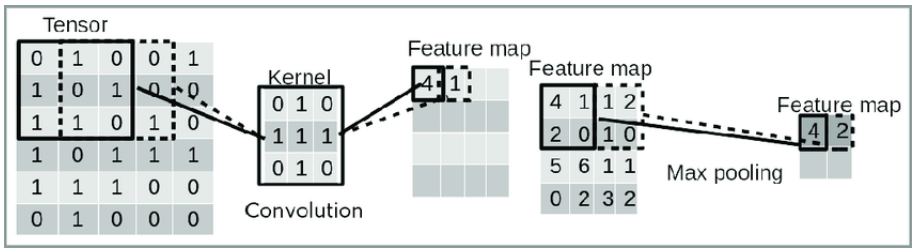

Figura 7: Maxpool y Convolucion, tomada de [7]

6. Tiempos de ejecución y optimizers A continuación se muestran los tiempos de ejecución que obtuvimos através de los servidores de Google Colab con CPU como entorno de ejecución.

Model	Execution time (seg)	Optimizer	Model	Execution time (seg)	Optimizer
RegLog	104	rmsprop	DNN6	85	adam
MLP_relu	199	sgd	DNN7	129	adam
MLP_selu	314	sgd	DNN8	95	adam
MLP_tahn	220	sgd	CNN1	321	adam
MLP_sig	791	sgd	CNN2	251	adam
DNN1	181	sgd	CNN3	359	adam
DNN2	165	sgd	CNN4	32	adam
DNN3	77	adam	CNN5	53	adam
DNN4	68	adam	CNN6	1174	adam
DNN5	103	adam	CNN7	494	adam

Cuadro 9: Tiempos de ejecución y optimizers para todos los modelos ajustados

Bibliografía

[1] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[2] J. Gareth, W. Daniela, H. Trevor, and T. Robert, *An introduction to statistical learning: with applications in R.* Springer, 2013.

[3] TensorFlow, "Scaled exponential linear unit (selu)." `https://www.tensorflow.org/api_docs/python/tf/keras/activations/selu`, May 2022.

[4] Keras, "Dropout layer." `https://keras.io/api/layers/regularization_layers/dropout/`, July 2022.

[5] Keras, "Earlystopping." `https://keras.io/api/callbacks/early_stopping/`, July 2022.

[6] Keras, "Maxpooling2d layer." `https://keras.io/api/layers/pooling_layers/max_pooling2d/`, July 2022.

[7] J. Schäfer, P. Schmitt, M. W. Hlawitschka, and H.-J. Bart, "Measuring particle size distributions in multiphase flows using a convolutional neural network," *Chemie Ingenieur Technik*, vol. 91, no. 11, pp. 1688–1695, 2019.