Universidad Iberoamericana de Puebla

Ingeniería en Sistemas Computacionales

Agentes Inteligentes

Fashion MNIST y Arquitecturas de Red

Diego Pacheco Valdez

Otoño 2024

Índice

Objetivo

El objetivo de esta actividad es aplicar técnicas de redes neuronales vistas en clase para mejorar el rendimiento de un modelo entrenado con el Dataset Fashion MNIST, proponiendo y evaluando tres diferentes arquitecturas de red.

La Primer Arquitectura

Dadas las condiciones indicadas, la primera arquitectura es aquella definida durante la clase del jueves 3 de octubre. Indagaremos más en esta.

Código de la Primera Arquitectura

```
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

mnist = tf.keras.datasets.fashion_mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

class_name = ['T-
shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag'
,'Ankle boot']

x_train, x_test = x_train/255, x_test/255

x_train, x_val, y_train, y_val = train_test_split(x_train,
                                                  y_train,
                                                  test_size=0.2,
                                                  random_state=42)

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])

print(model.summary())

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model_history = model.fit(x_train,
                          y_train,
                          epochs=10,
```

```
                        batch_size=64,
                        validation_data=(x_val, y_val))

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Loss: {test_loss}')
print(f'Acc: {test_acc}')

model.save(figsize=(15,4))

plt.subplot(1,2,1)
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylbael('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')

plt.subplot(1,2,2)
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

Explicación

El tipo de modelo de redes neuronales con el que se trabajo es el modelo secuencial, esto implica que cada capa avanza hacia adelante sin ningún retroceso.

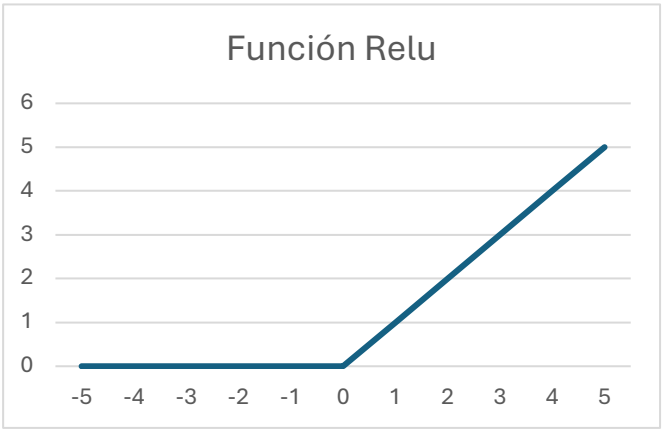
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_val,
y_val))
```

Capa de Entrada

La primera capa creada es la capa de entrada donde, por medio de la función flatten, se convierte la matriz de 28x28 en un valor del tipo array de 784 elementos. Esto permite analizar mejor los datos más adelante

Capa Oculta

La segunda capa, la capa oculta, constituye 16 neuronas las cuales trabajan con la función de activación de la Unidad Lineal Rectificada (RELU).



La función de activación para el modelo es aquella que controla la salida de nuestras neuronas. En el caso de la Capa Oculta, se usa la función de activación (RELU) donde, si el valor que recibe la neurona es menor a cero (es decir, negativo), el valor que da la neurona será cero, mientras que todo otro valor se mantiene como tal.

En pocas palabras, se trabaja exclusivamente por el universo positivo de los números reales y el cero.

Capa de Salida

Para finalizar, tenemos la tercera y última capa, esta es la capa de salida en la cual, se pasan los valores adquiridos de la capa oculta por la función de activación exponencial normalizada (SOFTMAX) con el propósito de reinterpretar los valores adquiridos como la probabilidad de que aquello que modelo está analizando sea una de las diez clases existentes dentro de la base de datos.

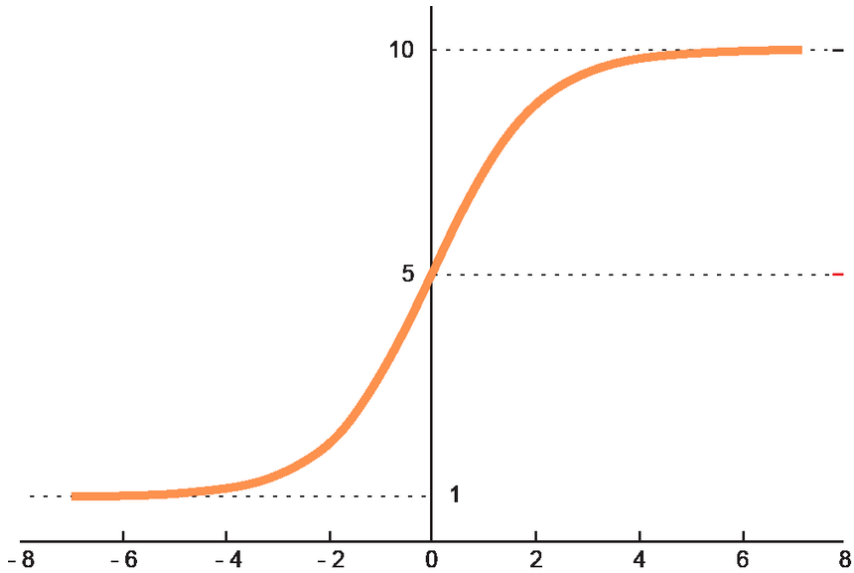


Fig.1 Grafica que representa la función de activación softmax (Es-Sabery et al., 2021)

Compilación

Una vez desarrollada la arquitectura del modelo, sigue el momento de entrenarlo con los datos de nuestra base.

Al iniciar la compilación se utiliza la función de optimación "Estimación Adaptativa de Momento" también conocida como ADAM. Esta nos ayuda a que conforme vayan pasando las épocas de entrenamiento, se vayan ajustando los pesos asignados a las distintas neuronas

> Adam utiliza una estimación del momento y de la magnitud de los gradientes anteriores para actualizar los parámetros del modelo en cada iteración. […] Adam adapta la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y de la magnitud del gradiente. Esto permite que el modelo se ajuste de manera más eficiente y efectiva a los datos de entrenamiento, lo que puede llevar a una mayor precisión de la predicción en comparación con otros métodos de optimización. (*Adam | Interactive Chaos*, s. f.)

Ahora, la función de perdida es aquella con la que se busca minimizar toda discrepancia entre los valores que va a predecir el modelo y el valor real con el que se está entrenando, por medio de esto es que se va guiando al modelo para ajustarse por su cuenta.

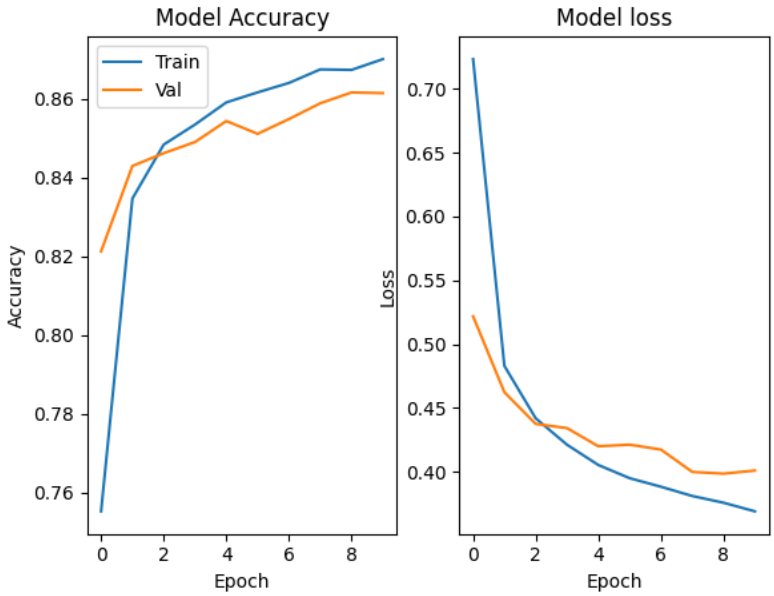
En el caso de diseño de este modelo y los siguientes, la función de perdida de entropía cruzada o función de perdida logarítmica, la cual, como nos explican Kedia y Nath (2022) "La probabilidad de cada clase predicha es comparada con la clase real y la perdida es calculada penalizando a la probabilidad dependiendo de lo alejado que este de la probabilidad real".

En el caso de nuestro modelo es la entropía cruzada escasa, la cual solo indica que las categorías son representadas con enteros.

Por último, tenemos el cálculo de la métrica accuracy, la cual, como indica su nombre en inglés, es la métrica de exactitud entre la probabilidad predicha y la probabilidad real.

Resultados

El modelo diseñado nos da como resultado una pérdida de 0.4280351996421814 y una exactitud de 0.848900020122528, las cuales en si son más que excelentes, pues no se trata ni de un caso de over ni underfitting.



Aun así, siempre se puede intentar mejorar, y eso es lo que buscaremos con los siguientes modelos.

La Segunda Arquitectura

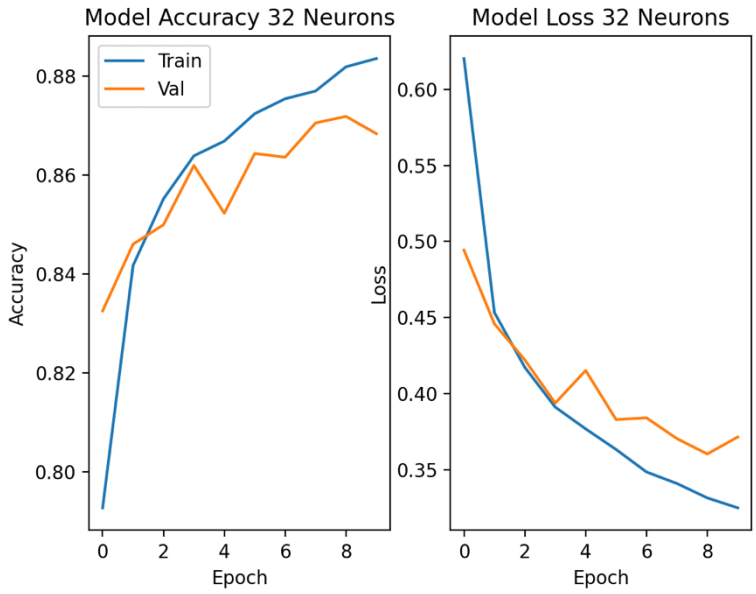
La segunda arquitectura con la que se experimentó con aumentar las neuronas en la capa oculta, este cambio, aunque no sea espectacularmente diferente, nos permitió lograr los siguientes resultados.

Experimentando con el Aumento de Neuronas

32 Neuronas

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
```

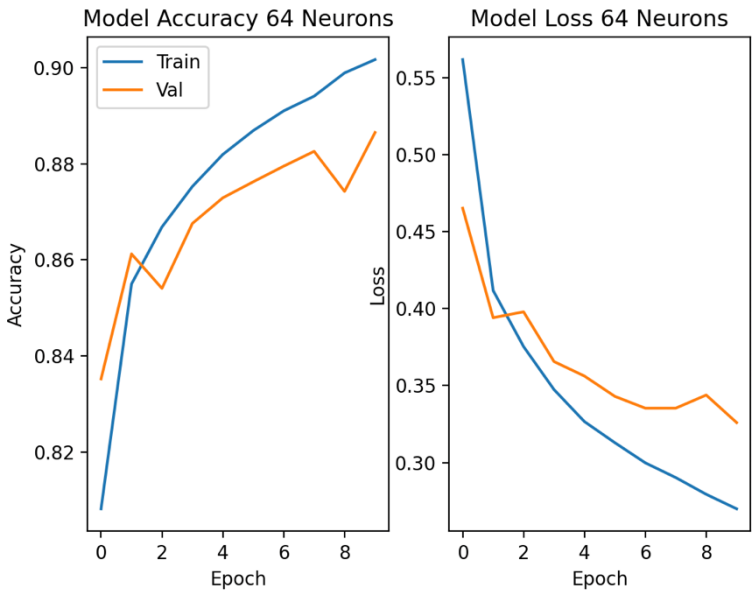
Para iniciar de forma calmada, se dobló el número de neuronas en la capa oculta, esto llevo a una exactitud de 0.8611999750137329 y una pérdida de 0.39052098989486694.



64 Neuronas

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(64, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
```

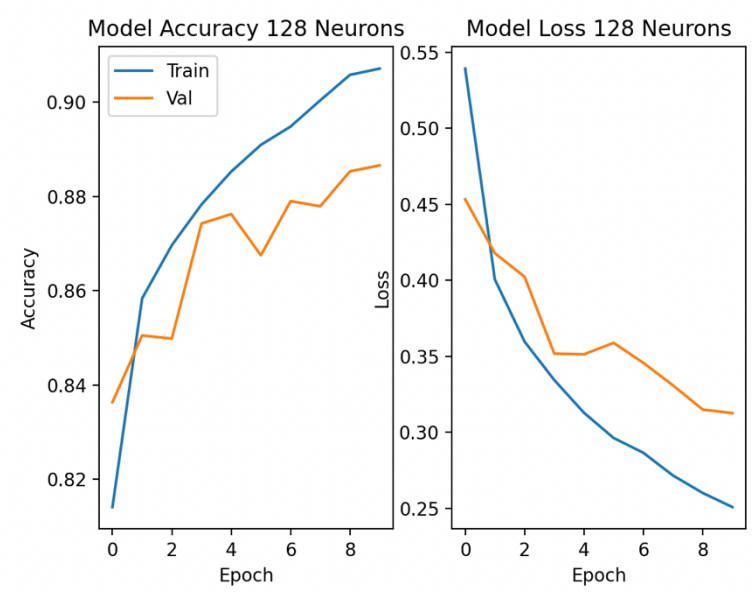
Continuando con el experimento, aumentamos al doble del intento anterior, esto nos dio como resultado una exactitud de 0.8745999932289124 y una pérdida de 0.35108333826065063



128 Neuronas

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(128, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
```

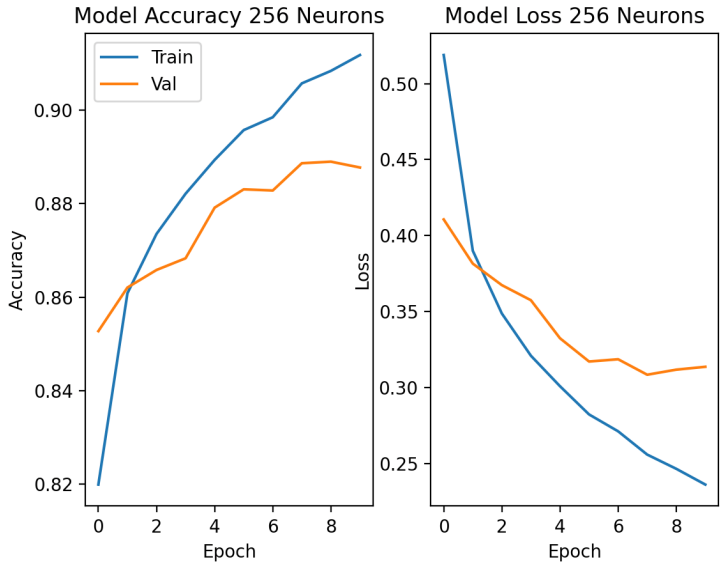
Ahora tenemos una exactitud de 0.8799999952316284 y una pérdida de 0.33720943331718445.



256 Neuronas

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(256, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
```

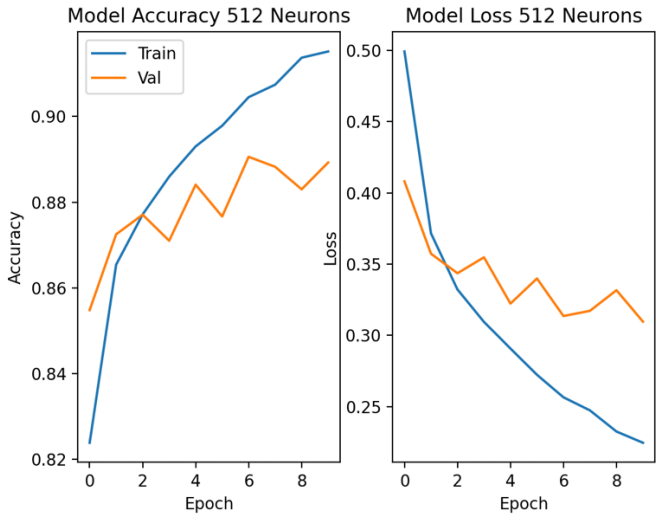
Los valores resultantes nuevamente vuelven a subir, mostrando una exactitud de 0.8835999965667725y una precisión de 0.337973028421402.



512 Neuronas

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(512, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
```

Por último, agregamos 512 neuronas con las cuales tenemos una exactitud de 0.8838000297546387 y una pérdida de 0.3298753798007965.



Interpretación de Resultados

Aunque los resultados obtenidos indican correctamente que el aumento de las neuronas se ve relacionado con un aumento en la exactitud del modelo, también queda claro que el aumento en este no es lo suficientemente alto para poder justificar el costo de cómputo para realizar el entrenamiento con más neuronas.

Sin embargo, de este experimento se podría indicar al modelo con 64 neuronas en su capa oculta como el idóneo, pues su aumento a 0.8745999932289124 en exactitud comparado con el 0.848900020122528 del modelo original es suficientemente mayor sin costar exponencialmente más que el diseño original.

El Tercer Modelo

En el caso del tercer modelo, se opta por experimentar por la creación de más capas ocultas, buscando acercarnos a una mayor exactitud sin hacer uso de demasiadas neuronas. En estos experimentos se tendrán ejemplos donde cada capa tendrá el mismo número de neuronas y ejemplos donde este no será el caso.

Experimentando con el Aumento de Varias Capas Ocultas

Dos Capas Ocultas

Iniciaremos agregando una sola capa oculta.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.4096366763114929
Acc: 0.8587999939918518
```

El resultado es significativamente mejor que el primer modelo, pero sigue siendo menor a los resultados obtenidos al usar una sola capa con 32 Neuronas. Se experimentará ahora con valores distintos entre neuronas

Creciente.

En este caso, se duplica el número de neuronas al pasar a la segunda capa oculta.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.4024789035320282
Acc: 0.8568999767303467
```

El resultado resulta similar al anterior.

Decreciente.

Ahora, el valor de neuronas de la segunda capa será la mitad de las neuronas existentes en la primera capa oculta.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(8, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.47201353311538696
Acc: 0.8337000012397766
```

Ahora la exactitud del modelo disminuye.

Tres Capas Ocultas.

A continuación, Se experimenta con el uso de 3 capas ocultas.

Decreciente.

En este primer ejemplo, la primera capa oculta tendrá el valor original de neuronas, la segunda capa la mitad y la tercera la mitad de la cantidad de neuronas de la segunda.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(8, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(4, activation='relu', name="Hidden_Layer_No3"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.5257018208503723
Acc: 0.8187000155448914
```

Intermedio.

En este intento, el numero original de neuronas será puesto en la segunda capa, con la primera y tercera correspondiendo a la mitad de las neuronas de la segunda capa.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(8, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(8, activation='relu', name="Hidden_Layer_No3"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.4647700786590576
Acc: 0.8375999927520752
```

Creciente.

Por último, el numero original de neuronas será otorgado con la tercera capa, la segunda siendo asignada la mitad de la tercera y la primera capa siendo asignada la mitad de la segunda.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(8, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(4, activation='relu', name="Hidden_Layer_No3"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.47355738282203674
Acc: 0.8378000259399414
```

Interpretación de Resultados

Los resultados en cada uno de estos cambios son increíblemente decepcionantes, sin embargo, me hace cuestionar que pasaría si el número de neuronas original fuese 64 en vez de 16.

Dos Capas Ocultas

Creciente.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(64, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.3715358376502991
Acc: 0.868399977684021
```

Decreciente.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(64, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.3586394488811493
Acc: 0.8733000159263611
```

Tres Capas Ocultas

Creciente.

```
model = tf.keras.models.Sequential([
```

```
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(64, activation='relu', name="Hidden_Layer_No3"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.383510023355484
Acc: 0.8628000020980835
```

Decreciente.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(64, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(16, activation='relu', name="Hidden_Layer_No3"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.35138845443725586
Acc: 0.8762999773025513
```

Intermedio.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28), name="Input_Layer"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No1"),
    tf.keras.layers.Dense(64, activation='relu', name="Hidden_Layer_No2"),
    tf.keras.layers.Dense(32, activation='relu', name="Hidden_Layer_No3"),
    tf.keras.layers.Dense(10, activation='softmax', name="Output_Layer")
])
Loss: 0.3695790767669678
Acc: 0.8666999936103821
```

Como se puede apreciar, el uso de la conclusión adquirida en el modelo anterior junto con diversas capas ocultas nos permite acercarnos más a un mejor nivel de exactitud.

Conclusiones.

Esta actividad nos mostró que la experimentación es una parte clave del análisis a la hora de crear modelos, pues conforme se fueron creando más capas, las decisiones anteriores formaron parte vital en la toma de nuevas decisiones.

Por último, queda aun así abierta la duda de, al iniciar la creación de un nuevo modelo seguimos sin identificar cuáles son aquellas pistas que seguir para poder empezar a modelar.

Referencias

Adam | Interactive Chaos. (s. f.). https://interactivechaos.com/es/manual/tutorial-de-machine-learning/adam

Es-Sabery, F., Hair, A., Qadir, J., Sainz-De-Abajo, B., Garcia-Zapirain, B., & Torre-Diez, I. (2021). Sentence-Level classification using parallel Fuzzy Deep learning classifier. IEEE Access, 9, 17943-17985. https://doi.org/10.1109/access.2021.3053917

Kedia, S., & Nath, P. (2022, 31 agosto). Here is what you need to know about Sparse Categorical Cross Entropy in nutshell. Machine Learning Diaries. https://vevesta.substack.com/p/here-is-what-you-need-to-know-about