**TEMA 1: Usando TensorFlow en Python**

**Introducción al tema**

Hoy en día un gran número de organizaciones, instituciones y empresas buscan automatizar sus procesos utilizando IA a través de la construcción de modelos de datos ayudando al proceso de toma de decisiones.

Por ejemplo, imagina la línea de producción dentro de una empresa que fabrica paneles solares. Un panel terminado se fotografía de manera automática y es etiquetado manualmente si es defectuoso o no. La empresa desea automatizar el proceso de etiquetado. Para esto la información dentro de cada foto debe ser útil para determinar si un panel solar terminado debe ser empaquetado para su distribución y/o venta o no.

Para que tú puedas generar un modelo de datos que resuelva este problema es de vital importancia que antes extraigas y representes la información utilizando un lenguaje de programación como Python y una biblioteca especializada en representación de datos y creación de modelos como TensorFlow. Una vez extraída la información y representada esta debe ser utilizada para entrenar un modelo de datos que para esto es de vital importancia conocer las operaciones y modos de ejecución de los cálculos dentro de TensorFlow.

En este tema vas a aprender a identificar el uso general de TensorFlow a través de la creación de tensores, sus operaciones básicas y modos de ejecución usando código en Python. Siendo la base para la construcción de modelos inteligentes con grandes volúmenes de datos.

**SUBTEMA 1: Tensores**

Dentro de tu entorno laboral es muy posible que la forma más común de visualizar, transmitir y capturar información sea a través de hojas de cálculo, formularios, listas de cotejo, entre otros.

Normalmente, la información de las empresas, organizaciones, e instituciones suele estar representada por números y texto, como, por ejemplo, los datos de contacto de un cliente en específico, las órdenes de compra, los horarios de una persona, etcétera. Cada valor asociado con este tipo de registros dentro de una base de datos puede ser manejado por Python y alguna biblioteca de análisis y manipulación de datos como Pandas. Sin embargo, muchas veces este tipo de representaciones de la información puede no ser suficiente cuando queremos almacenar la información que provenga de una imagen.

Hoy en día es muy probable que te encuentres con bases de datos que contengan información de millones de registros y por lo tanto el generar modelos de datos inteligentes sea una tarea que te consuma mucho tiempo de cómputo. En los últimos años se ha empezado a hacer uso de las unidades de procesamiento gráfico de propósito general llamadas GPUs para acelerar el proceso de generación de modelos de datos.

A close up of a computer chip

Description automatically generated

Ilustración 1 https://www.shutterstock.com/es/image-illustration/ai-circuit-board-technology-background-central-1196754286

Por lo tanto, si queremos representar imágenes y construir modelos rápidamente, surge la necesidad de utilizar una representación de un objeto en Python que nos permita representar la información que hay dentro de una imagen y además que pueda aprovechar las opciones de aceleración de los GPUs para que podamos generar modelos de datos más rápido.

Desde luego en Python existen algunas bibliotecas que nos podrían ayudar a construir modelo de datos con imágenes. Sin embargo, entre todas las bibliotecas disponibles, **TensorFlow** busca facilitar aún más este proceso y además no solamente integrar el uso de GPUs sino también de una nueva tecnología llamada unidad de procesamiento tensorial TPU.

A book with logo on it

Description automatically generated

Ilustración 2 https://www.shutterstock.com/es/image-vector/icon-books-about-programming-neural-networks-1071670085

Sabías que TensorFlow es utilizado en diversas empresas reconocidas para aplicaciones relacionadas con modelos de datos inteligentes. Te invitamos a visitar el siguiente enlace y que conozcas algunos casos de éxito y como es utilizado TensorFlow:

<https://www.tensorflow.org/about/case-studies>

Para que tu puedas utilizar esta poderosa plataforma llamada TensorFlow debes comprender primeramente el objeto con el que TensorFlow maneja la información internamente llamado **Tensor.**

Considera la siguiente información proveniente de una aerolínea que incluye tarifas de vuelos, posición de los aviones, y distancia en kilómetros por destino:

 Tabla 1

 Tabla 2

 Tabla 3

Es normal que el nombre tensor no te diga nada o incluso te tenga confundido, pero a grandes rasgos puedes ver un tensor como un objeto donde podemos poner datos. Por ejemplo, tomando la Tabla 1 como referencia podemos poner el costo unitario de cada vuelo en un tensor, es decir, 3000, 2400 y 2000 los cuales llamaremos **escalares** puesto que es solo un dato independiente por tensor. De igual forma es posible que estés buscando no solo almacenar un solo dato sino varios valores en un tensor como por ejemplo la posición de un vuelo como en la Tabla 2. Donde por identificador tiene asociado tres valores que marcan su posición actual la latitud, longitud y la altitud, este tipo de objeto que almacena más de un valor numérico le llamaremos **vector**. Finalmente, si quisieras incluso almacenar en un tensor datos distribuidos en celdas marcadas por diferentes filas y columnas como las distancias en kilómetros entre los diferentes destinos dentro de la Tabla 3, se podría hacer sin ningún problema y llamaríamos a este conjunto de valores **matrices**.

A diagram of a basic machine

Description automatically generated with medium confidence

Ilustración 3 https://www.shutterstock.com/es/image-vector/scalar-vector-matrix-tensor-1915020757

Por lo tanto, un tensor es un objeto donde puedes almacenar escalares, vectores, o matrices de dato, pero a la vez también pueden ser útiles para almacenar mucha más información. Ahora bien, es tiempo de que nos pongamos a trabajar y comiences a utilizar Python con TensorFlow.

Para utilizar TensorFlow dentro de nuestro cuaderno tenemos que agregar la siguiente línea de código: **import** **tensorflow** **as** **tf.**

Una vez que aparezca el nuevo bloque coloca el siguiente código:

escalar = tf.Variable(**1342**, shape = (), dtype = tf.int16)

vectorFila = tf.Variable([**12**, **21**, **21**, **21**], shape = (**4**,), dtype = tf.int32)

vectorCol = tf.Variable([[**12**], [**21**], [**21**], [**12**]], shape = (**4**,**1**), dtype = tf.int32)

matriz = tf.Variable([[**32**,**43**,**23**], [**23**,**44**,**53**], [**60**,**54**,**81**], [**56**,**73**,**24**]], shape = (**4**,**3**), dtype = tf.int32)

texto = tf.Variable("Este registro fue generado con la información de los clientes", shape = (), dtype= tf.string)

**print**(escalar)

**print**(vectorFila)

**print**(vectorCol)

**print**(matriz)

**print**(texto)

Te puedes dar cuenta que cada tensor es almacenado en una variable a la que llamamos según el dato o los datos que se están guardando (el nombre de la variable donde se almacena el tensor puede ser cualquiera). Cada tensor tiene un valor inicial que puede ser un escalar (**1342**) que represente, por ejemplo, el número de transacciones bancarias de una PyME, un vector fila o columna ([**12**, **21**,**21**,**21**] o ([[**12**], [**21**], [**21**], [**12**]]) que representa los días de 4 meses seguidos de cuando se hicieron pagos a algún tipo de servicio o tarjeta de crédito, una matriz ([[**32**,**43**,**23**], [**23**,**44**,**53**], [**60**,**54**,**81**], [**56**,**73**,**24**]]) que represente el número de cambios y devoluciones en cuatro tiendas diferentes durante un trimestre o una cadena de texto que represente información descriptiva y que no puede expresarse de forma numérica ("Este registro fue generado con la información de los clientes").

De igual forma cada tensor tiene asociado una forma (shape) que es la que establece la cantidad de datos y como se organizan dentro del tensor. Por ejemplo, en la línea 1 y 5 del código anterior donde almacenamos un escalar y texto la forma se declaró como shape = () debido a que solamente almacenamos un solo dato como el número de transacciones de una Pyme o información descriptiva de un registro. Si en tu caso necesitaras almacenar más datos debemos proporcionar más información. Ejemplos de elementos con más datos pueden ser vectores que podrían representar toda la información relacionada con un producto de tu inventario o matrices que bien podrían ser hojas de cálculo completas que almacenen reportes de ventas por mes y producto. En el caso de los vectores tenemos dos alternativas, una puede ser un vector general o fila y solo debemos especificar el primer valor indicando el número de elementos. Por ejemplo, en la línea 2 estamos almacenando una lista de 4 valores que representa los días de pago de una tarjeta de crédito y la forma la especificamos como shape = (**4**,) donde el primer número dentro del paréntesis representa la cantidad de valores que almacenaremos dentro del tensor. Si necesitaras almacenar más valores podrías especificar la cantidad modificando este parámetro en la construcción del tensor. De igual forma es importante reconocer que tenemos vectores columna que a diferencia de los vectores fila si los vemos de forma gráfica están en una posición vertical. Para utilizar los vectores columna debes especificar su forma incluyendo el número de filas y especificar que al menos se tendrá una columna. Por ejemplo, la línea 3 especifica que el vector columna tendrá 4 filas y una columna shape = (**4**,**1**). Es muy importante que te des cuenta de que la forma de inicializar el vector columna es diferente que el vector fila. La matriz al igual que el vector columna necesita especificar en su forma tanto el número de fila como el de columnas el cual puede ser diferente de 1. Por ejemplo, la matriz de la línea 4 representando el número de cambios y devoluciones en cuatro tiendas diferentes durante un trimestre almacena 4 filas de 3 columnas shape = (**4**,**3**) donde cada lista dentro de la lista principal del valor inicial representa una fila de la matriz.

Otro elemento importante de los tensores es el tipo de dato que van a almacenar que se especifica manualmente en dtype, en nuestro código anterior tenemos: tf.int16, tf.int32, tf.string. El tipo de dato nos ayudará a especificar dentro de TensorFlow la cantidad de bits que representará nuestro número. El tipo de dato tienen una gran relevancia porque te ayudará a ahorrar espacio en memoria, por ejemplo, si quieres guardar días del mes bastaría utilizar un tf.int8, sin embargo, si vas a guardar cantidades más grandes como el número de transacciones anuales dentro de una empresa por mes podrías utilizar un tf.int32. En ocasiones el número de bits se va a colocar explícitamente seguido del tipo de dato como en tf.int16 y tf.int32 donde mientras mayor sea el número podemos representar números con más grandes. Las últimas líneas de código del bloque son para imprimir la información dentro de los tensores.

A screen shot of a computer code

Description automatically generated

Como observas, la información que recibimos es relacionada con los tensores que acabamos de crear. Podemos observar primeramente información interna de TensorFlow sobre que los tensores que creamos son del tipo “Variable”, también podemos ver los atributos de forma (shape), y el tipo de dato (dtype).

Finalmente, TensorFlow en ocasiones se apoya del uso de numpy una biblioteca para el manejo de vectores, matrices y sus operaciones. Es por esa razón que al imprimir nuestros tensores al final del código anterior existe un atributo llamado “numpy” que especifica la representación de los datos que ingresamos utilizando esta biblioteca.

Si ya dominas los atributos de los tensores también puedes omitir la forma y el tipo de dato y dejar que sea TensorFlow el que tome estas decisiones como sucede en el siguiente código:

escalarFlotante = tf.Variable(**23.43**)

vectorFila2 = tf.Variable([**3.4**, **4.2**, **5**])

matriz2 = tf.Variable([[**2**,**34**,**5**,**3**],[**43**,**23**,**532**,**23**]])

**print**(escalarFlotante)

**print**(vectorFila2)

**print**(matriz2)

A computer code with text

Description automatically generated with medium confidence

Aquí podrás observar información similar al bloque de código anterior con la diferencia de que dejamos que TensorFlow asignará la forma y el tipo de dato para almacenar la información. Puedes notar que introdujimos otro tipo de dato tf.float32 así como también es importante que si dentro de un escalar, vector o matriz hay un dato con punto decimal este será un tensor de tipo tf.float32.

Así como hemos visto los atributos internos de un tensor, también existen varios tipos de ellos.

A cube with numbers in it

Description automatically generated with medium confidence

Ilustración 5 https://www.shutterstock.com/es/image-illustration/mathematics-tensor-multidimensional-dataset-array-3d-1145539652

Los principales son:

* Variable: el valor asignado al tensor puede cambiar en cualquier momento, no así su forma o tipo de dato. Son utilizados normalmente para almacenar resultados de operaciones o datos que cambiarán en el tiempo. Por ejemplo, los 5 mejores tiempos de un corredor que son datos que podrían actualizarse en un futuro pueden almacenarse en un tensor de tipo variable:

variable = tf.Variable([**12.11**, **12.23**, **12.44**, **12.51**, **12.54**])

**print**(variable)

Salida:



* Constant: el valor o los valores asignados al tensor no pueden ser modificados una vez creado. Pueden utilizarse como elementos fijos para calcular valores en operaciones en el proceso de evaluación. Por ejemplo, el reporte de ventas de 3 productos en un cuatrimestre previo al actual, ya no se actualizará por lo que puede permanecer constante una vez ingresado a un tensor:

constante = tf.constant([[**124**,**100**,**154**, **100**], [**30**, **44**, **23**, **50**], [**55**, **98**, **100**, **87**]])

**print**(constante)

Salida:

A black text on a white background

Description automatically generated

También, existen otro tipo de tensores menos comunes para otros propósitos específicos:

* SparseTensor: el valor o los valores asignados al tensor no pueden ser modificados una vez creado. Son tensores que se utilizan para representar matrices dispersas que son aquellas que contienen valores numéricos pero la mayoría son ceros solo una pequeña cantidad de valores son diferentes a cero. Ejemplo:

tf.sparse.SparseTensor(indices=[[**1**, **1**], [**2**, **3**]], values=[**20**, **44**], dense\_shape=[**3**, **4**])

Salida: 

Si bien no podemos visualizar la información debido a que esta almacenada de forma optimizada, la matriz asociada al SparseTensor es de la siguiente forma:



Donde el atributo indices especifica el número de fila y columna empezando desde la fila 0 y columna 0 de las posiciones de la matriz que tendrán un valor diferente de 0. Es por esa razón que el valor 20 se encuentra en la celda que coincide con la fila 1 y la columna 1. Y para el caso del 44 está en la celda de intersección de la fila 2 con la columna 3.

* Placeholder: son tensores que pueden utilizarse para alimentar datos a modelos y no necesariamente deben tener un valor inicial fijo, pero no pueden ser evaluados para realizar operaciones. De igual forma es necesario utilizar sesiones para utilizarlos y no son compatibles con el concepto “eager execution” que nos permite evaluar el tensor de forma inmediata. Ejemplo:

tf.compat.v1.disable\_eager\_execution()

x = tf.compat.v1.placeholder(tf.float32, shape=(**1024**, **1024**))

**print**(x)

Salida:



Te darás cuenta de que el uso de tensores se extenderá aún más adelante, pero esta breve introducción te ayudará a entender de mejor forma los datos que puedas representar en tensores y explotarlos con TensorFlow.

**SUBTEMA 2: Operaciones básicas con tensores**

Una vez que has logrado representar la información que viene de hojas de cálculo, reportes, cuestionarios, entre otros a través de una biblioteca como TensorFlow que te permita su manipulación es tiempo de realizar operaciones con esos datos. En muchas ocasiones dentro de una organización se busca constantemente transformar todos esos datos en información que nos permita sacarles provecho, por ejemplo, en nuestro caso en particular utilizarlos para la creación de modelos de datos. Estos modelos pueden ayudarnos al proceso de tomas de decisiones en ambientes sociales, económicos, industriales, etcétera.

Pero, recuerda que para llegar a los modelos de datos es de vital importancia que conozcas las operaciones y la forma de evaluación de los procesos dentro de TensorFlow.

Además de que los tensores te ayuden a almacenar información a la vez **pueden ser generados por el resultado de una operación**. Es decir, los tensores te ayudarán a representar cálculos parciales de operaciones definidas que producirán un valor. Esto es muy importante porque TensorFlow contiene una serie de operaciones que trabajarán con tensores y producirán tensores.





Intenta con el código:

**print**(tf.add([**345**, **200**, **123**, **145**, **180**, **200**], [**33**, **24**, **47**, **65**, **120**, **90**]))

**print**(tf.subtract([**1430**, **242**, **53**, **234**, **353**,**200**], **34**))

La línea 1 de nuestro código representa la suma semestral por mes de las ventas de dos productos diferentes para sacar el total de ventas mensuales y la línea dos es la corrección del inventario por 34 unidades de 6 productos. Los resultados que debes obtener son los siguientes:



Si observas detenidamente, la operación tf.add toma dos vectores del mismo tamaño y forma sumando elemento a elemento lo cual te puede servir para sacar el total de eventos en un período de tiempo. De igual forma el resultado de dicha operación es colocado en un tensor que tiene una forma y un tipo de dato. El segundo resultado que obtendrás será por una operación de resta entre un vector y un escalar en este caso el escalar se le va a restar a cada elemento del vector lo cual te puede servir para hacer ajustes en todos los elementos del vector como la corrección de inventarios.

De igual forma existen diversos tipos de operaciones que pueden aprovechar el uso de los GPUs y TPUs dentro de TensorFlow como, por ejemplo:

* tf.matmul: realiza una multiplicación de matrices
* tf.reduce\_max: obtiene el máximo valor entre los elementos numéricos que conforman un tensor.
* tf.reduce\_mean: obtiene el valor promedio entre los elementos numéricos que conforman un tensor.
* tf.linalg.inv: calcula la inversa de una matriz
* tf.linalg.matrix\_transpose: funciona para transponer una matriz

Ahora intenta

matriz4x3 = tf.constant([**3**, **4**, **3**, **4**, **5**, **2**, **6**, **6**, **3**, **2**, **1**, **3**], shape=[**4**, **3**])

**print**(matriz4x3)

matriz3x2 = tf.constant([**1**, **2**, **2**, **1**, **1**, **2**], shape=[**3**, **2**])

**print**(matriz3x2)

**print**(tf.matmul(matriz4x3,matriz3x2))

ventasxvend = tf.constant([**13**, **24**, **23**, **28**, **34**, **45**, **20**, **14**, **35**, **50**, **35**])

**print**(tf.reduce\_max(ventasxvend))

**print**(tf.reduce\_mean(ventasxvend))

Ya habrás identificado que la línea 1 y 3 contienen la declaración de dos tensores constantes. Pareciera que ambos tensores tienen como valor inicial un vector, sin embargo, es importante que notes que la forma shape nos puede ayudar a acomodar un vector como matriz. Esto será posible, siempre y cuando el número de elementos del vector se igual al número total de elementos dentro de la matriz de acuerdo con sus dimensiones. Por ejemplo, la primera matriz es de 4 filas por 3 columnas lo que nos da un total de 12 elementos dentro de la matriz. En el caso de la segunda matriz la forma indica que son 3 filas por 2 columnas lo que nos da un total de 6 elementos. Es muy importante que tengas presente como se especifica la forma de los tensores dado que te será de mucha utilidad para especificar el alto y ancho del número de datos que contiene una imagen. **En otras palabras, utilizaremos matrices dentro de tensores para representar las imágenes en TensorFlow**. La línea 5 del código realiza una multiplicación de matrices con la matriz de 4 x 3 y la matriz de 3 x 2. Si te das cuenta estamos consumiendo tensores ya declarados a diferencia de nuestro bloque de código anterior donde colocamos directamente los valores con los que se realizarían las operaciones. Ambas maneras son totalmente válidas para realizar operaciones en TensorFlow. Como ya sabrás el resultado de dicha operación será un tensor.

De igual forma puedes utilizar otras operaciones complementarias para conocer información relevante de los datos que ingresas. Por ejemplo, en la línea 6 declaramos un tensor que tiene las ventas mensuales por vendedor de una tienda departamental en el departamento de electrónica.



Si quisieras saber cuál es el máximo número de ventas o el promedio de ventas mensual por empleado puedes utilizar operaciones como tf.reduce\_max y tf.reduce\_mean en las líneas 7 y 8 respectivamente.

Texto

Descripción generada automáticamente con confianza media

Es muy posible que en este momento te preguntes la utilidad directa de operaciones como la multiplicación de matrices, la inversa y transpuesta puesto que mayormente están orientadas a las matrices. Por esa razón te decimos que utilizaras estas y otras operaciones para lograr la construcción de modelos inteligentes.

Ya para finalizar este subtema es necesario que puedas reconocer dos tipos de ejecución que se pueden dar en TensorFlow. La primera y también la que hemos venido usando en nuestros bloques de código anteriores es la ejecución ávida o en inglés “eager execution”. Este tipo de ejecución evalúa las operaciones de inmediato es decir podemos visualizar el resultado dentro de un tensor una vez ejecutando el código. Te recomendamos este tipo de ejecución para facilitar la experimentación y depuración de modelos y datos pequeños. Es decir, si tienes un proyecto que quieras llevar rápidamente a TensorFlow para probar tus ideas es la mejor forma de ejecutarlo. Es importante saber que la ejecución ávida es compatible con la mayoría de las operaciones en TensorFlow y la aceleración con GPU.

La segunda forma de ejecución llamada ejecución gráfica o en inglés “graph execution” se refiere que los cálculos de nuestros tensores se ejecutarán como un grafo de TensorFlow tf.Graph. Pero la pregunta aquí sería ¿qué beneficios te traería usar este tipo de ejecución? Imagina que necesitas exportar y ejecutar fuera de Python tu modelo de datos ya sea colocarlo en un sistema embebido que forme parte de la línea de producción de una empresa, una aplicación web que ayude a tus clientes a optimizar sus procesos o en un dispositivo móvil que te ayude al reconocimiento de objetos específicos entonces la construcción del gráfico te va a ayudar mucho. A continuación, te mostramos un ejemplo simple del uso de la ejecución gráfica en TensorFlow. Considera que quieres obtener el total de comisión que tendrás que pagar a un empleado en función de sus ventas teniendo el costo unitario de los productos que vendió y las cantidades.



La comisión es el 7% de las ventas totales por lo que el grafo para realizar esta operación en TensorFlow quedaría de la siguiente manera:



Donde los dos primeros elementos son tensores constantes que nos ayudarán a alimentar el grafo con nuestros datos. Si observas los tensores será ingresados como matrices de 1x3 y 3x1 para simplificar las operaciones y solo utilizando una multiplicación de matrices podamos obtener el total de dinero. El resultado va a generar un solo valor por lo que puede considerarse como escalar y ya para finalizar el proceso realizaremos una multiplicación por el porcentaje de comisión para obtener la comisión total que se le dará al vendedor. Los grafos son estructuras de datos que como lo vimos en el ejemplo anterior tienen un conjunto de objetos tf.Operation que representan unidades de cálculo como tf.matmul y tf.multiply . Así como también, tensores que representan unidades de datos que fluyen entre las operaciones.

Copia el siguiente código en un bloque nuevo dentro de Colab:

**def** **funcion\_ejemplo\_python**(num\_ventas, precios):

total = tf.matmul(num\_ventas, precios)

comision = tf.multiply(tf.cast(total,tf.float32), .**07**)

**return** comision

funcion\_grafo = tf.function(funcion\_ejemplo\_python)

n\_ventas = tf.constant([**3**, **5**, **9**],shape = [**1**,**3**])

preciosxprod = tf.constant([**1800**, **759**, **599**], shape = [**3**,**1**])

**print**(funcion\_grafo(n\_ventas,preciosxprod))

La forma más sencilla de establecer un grafo es creando una función en Python con las operaciones de nuestro grafo como en la línea 1. Las entradas que utilizamos son el número de ventas por producto y su precio unitario por producto. Después, realizamos la multiplicación de matrices y posteriormente el cálculo del total de la comisión tal y como lo marca el grafo anterior. Nota que dentro de la multiplicación convertimos el total a un tipo de dato tf.float32 esto debido a que no podemos multiplicar datos de diferente tipo sin haber hecho su conversión explícitamente.

Una vez generada la función en Python debemos convertirla a una función de Tensor Flow tf.function como en la línea 5 con el objetivo de cambiar de una ejecución ávida a gráfica. Las líneas 6 y 7 son para declarar tensores constantes que tienen vectores fila 1x3 y columna 3x1 que también pueden verse como matrices. Recuerda que las dimensiones son muy importantes para la multiplicación de matrices de lo contrario la operación de no podría realizar. La regla elemental es que el número de columnas de la primera matriz debe ser igual al numeró de filas de la segunda matriz y el resultado será una matriz del número de filas de la primera matriz por el número de columnas de la segunda. La línea 8 muestra el resultado de la ejecución del grafo el cual es un escalar con la comisión ganada por el vendedor.



Si observas bien el escalar viene dentro de una matriz de 1x1 debido a que durante nuestro grafo estuvimos operando con matrices.