TensorFlow™

IBM Innovation Lab

¿Qué es **TensorFlow**?

TensorFlow

TensorFlow es un framework utilizado en machine learning, creado por Google. Es una gran plataforma para construir y entrenar redes neuronales, que permiten detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

Componentes del **grafo**

Nodos

Representan operaciones matemáticas

Conexiones - Links

Representan los datos, comúnmente *arrays* multidimensionales (tensores)

El nombre **TensorFlow**, proviene de las operaciones realizadas por las redes neuronales, las cuales utilizan *arrays* de datos o tensores. Es esencialmente, un flujo de tensores.

Conceptos **básicos**

¿Qué es una **red neuronal**?

Redes neuronales

Como su etimología lo dice, la idea de las redes neuronales, consiste en imitar el funcionamiento de las redes neuronales de los organismos vivos.

Es decir, un conjunto de neuronas, conectadas entre sí y que trabajan en conjunto, sin tener una tarea fija para cada una. Con entrenamiento, las neuronas van creando y reforzando ciertas conexiones para “aprender”.

Redes neuronales: **Base**

Las redes neuronales se basan en una idea sencilla. Dados unos parámetros, existe una forma de combinarlos para predecir un cierto resultado.

Por ejemplo, sabiendo los píxeles de una imagen, habrá una forma de saber qué hay en esa imagen.

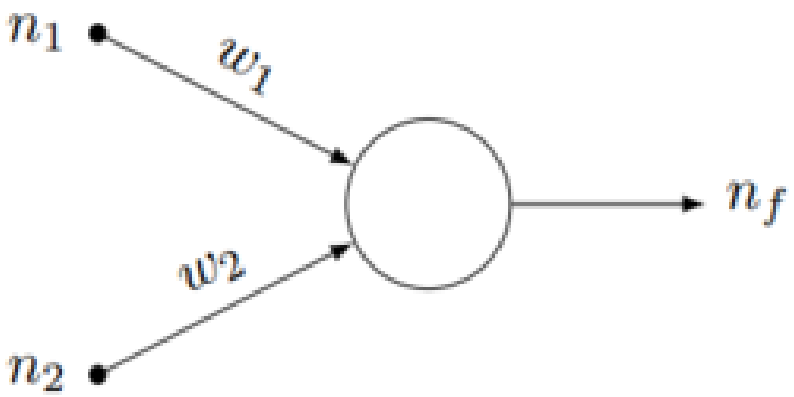
Redes neuronales: **Base**

Sin embargo, es muy difícil combinar estos parámetros.
Aquí entran las redes neuronales.

Estas nos permiten encontrar esa combinación de parámetros y aplicarla al mismo tiempo. En otras palabras, hallar la combinación que mejor se ajusta es "entrenar" la red neuronal.

Redes neuronales: **Base**

Una red ya entrenada se puede usar luego para hacer predicciones o clasificaciones, es decir, para "aplicar" la combinación.

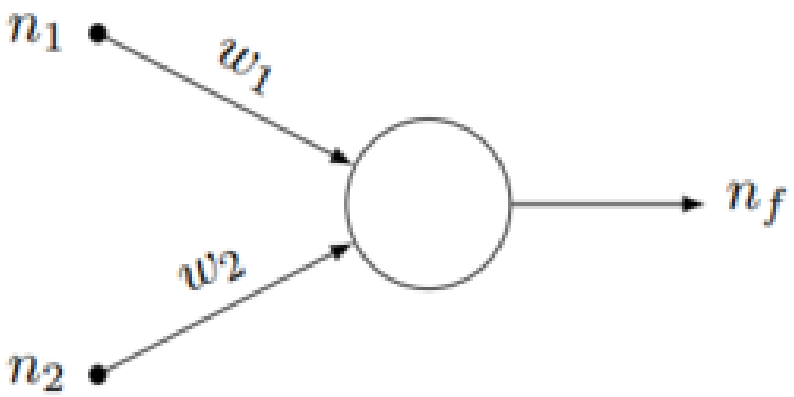


Redes neuronales: **Base**

La unidad básica de la red neuronal: el perceptrón.

Un perceptrón es un elemento que tiene varias entradas (**n1, n2**) con un cierto peso cada una (**w1, w2**).

Si la suma de esas entradas por cada peso es mayor que un determinado número, la salida (**nf**) del perceptrón es un uno. Si es menor, la salida es un cero.

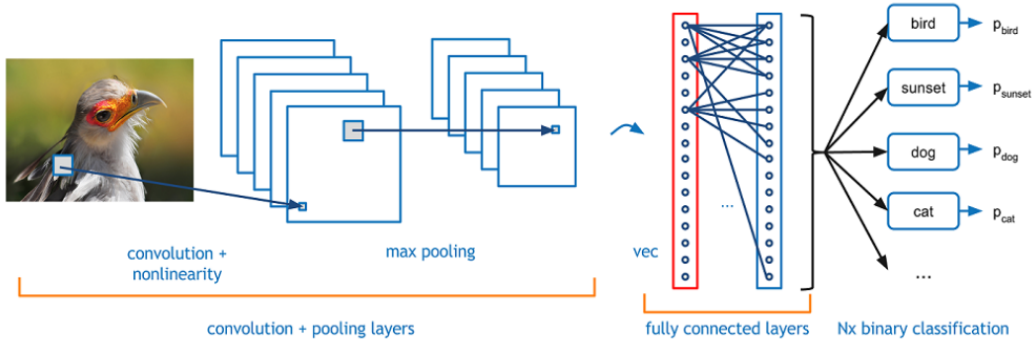


Redes neuronales: **Base**

La unidad básica de la red neuronal: el perceptrón.

Un perceptrón es un elemento que tiene varias entradas (**n1, n2**) con un cierto peso cada una (**w1, w2**).

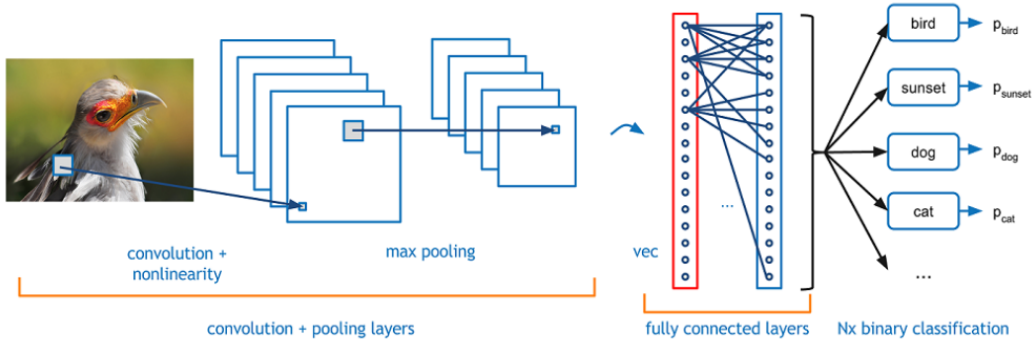
Si la suma de esas entradas por cada peso es mayor que un determinado número, la salida (**nf**) del perceptrón es un uno. Si es menor, la salida es un cero.



Redes neuronales: **Base**

Existen distintos tipos de redes neuronales, en nuestro caso, veremos un ejemplo de reconocimiento de imágenes donde se utilizan redes neuronales convulsionales.

Este tipo de redes, constan de múltiples capas diseñadas para requerir un preprocesamiento relativamente pequeño en comparación con otros algoritmos de clasificación de imágenes.



Redes neuronales: **Base**

Aprenden usando filtros y aplicándolos a las imágenes. El algoritmo toma un cuadrado pequeño y comienza a aplicarlo sobre la imagen. Cada filtro permite que la red identifique ciertos patrones en la imagen.

Las primeras capas de la red pueden detectar características simples como líneas, círculos, bordes. En cada capa, la red puede combinar estos hallazgos y aprender continuamente conceptos más complejos a medida que profundizamos en las capas de la red neuronal.

¿Qué es un **tensor**?

Tensor

Es unidad principal de datos en TensorFlow. Consiste en un conjunto de valores conformados en una matriz.

Por ejemplo, una imagen o una palabra van a ser tensores que tendrán una dimensión o un rango de tensor.

Rango de un tensor: **dimensión de la matriz**

```
6.0 #Rango 0
[5.0 , 6.0] #Rango 1
[[5.0,6.0],[9.0,3.0]] #Rango 2
[[[5.0, 6.0]] , [[9.0, 3.0]]] #Rango 3
```


Es bueno saber que TensorFlow brinda APIs para Python, C++, Haskell, Java, Go, Rust y también hay un paquete de terceros para R llamado **tensorflow**

TensorFlow: ***Interactive session***

La sesión interactiva brindada por TensorFlow, permite la ejecución de grafos o parte de estos. Asigna los recursos (en una o varias máquinas) para la tarea y mantiene los valores reales de los resultados intermedios y variables. La sesión debe cerrarse para liberar recursos. Para crear una sesión debemos escribir:

`tf.Session #tf` = abreviatura tensorFlow

Veamos un ejemplo

```
# Importamos TensorFlow
import tensorflow as tf

# Inicializamos dos constantes
ten1 = tf.constant([1,2,3,4])
ten2 = tf.constant([5,6,7,8])

# Multiply - Multiplica dos variables
result = tf.multiply(ten1, ten2)

# Imprimimos el resultado
print(result)
```


¿Qué se imprime?

```
>Tensor("Mul:0", shape=(4,), dtype=int32)
```

Contrario a lo que hubiéramos esperado, el resultado de estas líneas de código, representan un tensor en el grafo de computación. Sin embargo, el resultado no fue calculado. Es decir, simplemente se definió el modelo (el grafo) pero no se calculó el resultado.

¿Y si quiero ver el resultado?

En este momento es cuando la sesión interactiva cobra importancia. Si deseamos ver el resultado, debemos correr este código dentro de una sesión interactiva.

Lo cual podemos hacer de la siguiente manera...

```python
# Importamos TensorFlow
import tensorflow as tf

# Inicializamos dos constantes
ten1 = tf.constant([1,2,3,4])
ten2 = tf.constant([5,6,7,8])

# Multiply
result = tf.multiply(ten1, ten2)

# Inicializamos la sesión
sess = tf.Session()

# Imprimimos el resultado
print(sess.run(result))

# Cerramos la sesión
sess.close()
```


Programando en **TensorFlow**

Programando en TensorFlow

- Existen conceptos matemáticos complejos
- Paradigma o modelo de programación

Programación ‘tradicional’ vs Programación en TensorFlow

¡Manos a la obra!

Primeros pasos en la práctica

- Probar TensorFlow utilizando **jupyter notebook** en IBM **Watson Studio**
- Crear nuestra primera red neuronal en 10 líneas utilizando la libería Keras (`tf.keras`)
- Aprender acerca *de machine learning* con pocos requisitos previos

Probar **TensorFlow** utilizando **jupyter notebook** en IBM **Watson Studio**

Utilizar **TensorFlow** desde IBM **Watson Studio** nos permite:

- Olvidarnos de la instalación de TensorFlow y sus dependencias dado que vienen pre-instaladas.
- Es un servicio de ***cloud*** **gratuito**, por lo que tenemos acceso desde cualquier equipo con nuestras credenciales.
- Contamos con capacidad de procesamiento en forma gratuita.

Crear nuestra primera red neuronal en **10 líneas** utilizando la libería **Keras** (**tf.keras**)

TensorFlow tiene muchas APIs. Sin embargo, una de las más completas que existe es **Keras**.

La misma está implementada dentro del mismo TensorFlow y será una de las API's que utilizaremos para crear nuestra primera red neuronal.

APIs y conceptos a utilizar

`tf.keras` **`tf.data`** **`eager execution`**

tf.keras

Keras es una API de alto nivel para crear y entrenar modelos de aprendizaje profundo (Deep learning). Se usa para prototipos rápidos, investigación avanzada y producción, con tres ventajas clave.

tf.keras – ***ventajas***

User friendly

Cuenta con una interfaz simple y consistente para usos comunes. Brinda en forma clara información sobre los errores de usuario.

Modular y componible

Los modelos en **keras** se hacen conectando bloques de construcción con pocas restricciones.

Fácil de extender

Permite escribir bloques de construcción personalizados para expresar nuevas ideas para la investigación; crear nuevas capas, funciones de pérdida (*loss function*) y desarrolle modelos de última generación.

tf.data

La API **tf.data** facilita el manejo de grandes cantidades de datos, diferentes formatos de datos y transformaciones complicadas.

Por ejemplo, imágenes seleccionadas al azar en un lote para capacitación, para facilitar esta tarea.

Eager execution

La ejecución ansiosa (*eager execution*) de TensorFlow es un entorno de programación imperativo que evalúa las operaciones de inmediato, sin construir grafos. Las operaciones devuelven valores concretos en lugar de construir un grafo computacional para ejecutarlo más tarde.

Esto hace que sea fácil comenzar con TensorFlow y depurar modelos.

CONCEPT HEAVY, code light.

Independientemente de cuán inteligente y preparado estés, aprender correctamente esta herramienta lleva su tiempo.

¡NO AFLOJES!

Cuando definimos una red neuronal con TensorFlow, vemos múltiples parámetros. Sin embargo, solo para algunos conviene pasar un tiempo analizándolos para entender su propósito.

Estos, los veremos en las siguientes diapositivas.

Nuestra primera *red neuronal*

Hello World + TensorFlow

En este ejemplo práctico, veremos el *hello world* de la computación visual utilizando TensorFlow.

Crearemos modelo de red neuronal para clasificar imágenes de ropa, como zapatillas y camisas. Veremos una descripción rápida de un programa completo de TensorFlow con los detalles explicados a medida que avanzamos.

Esta guía usa **tf.keras**

¿Y qué pasos debo seguir?

Pasos para construir una red neuronal

1. Recolectar el conjunto de datos *(mayor parte del trabajo)*
2. Construir el modelo *(algunas líneas de código)*
3. Entrenamiento *(una línea de código)*
4. Evaluar modelo *(una línea de código)*
5. Predecir *(una línea de código)*

A lo largo del dojo no veremos el código completo en detalle. Para acceder al código completo, debemos ingresar en el siguiente notebook en IBM **Watson Studio**:

LINK A NOTEBOOK

1 **Recolectar** el conjunto de datos

```
# TensorFlow y tf.keras
import tensorflow as tf
from tensorflow import keras

#librerías de ayuda
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

El MNIST de moda pretende ser un reemplazo directo del clásico conjunto de datos de MNIST, a menudo utilizado como el “Hello world" de los programas de aprendizaje automático para visión artificial. El conjunto de datos de MNIST contiene imágenes de dígitos escritos a mano (0, 1, 2, etc.) en un formato idéntico al de las prendas que usaremos aquí.

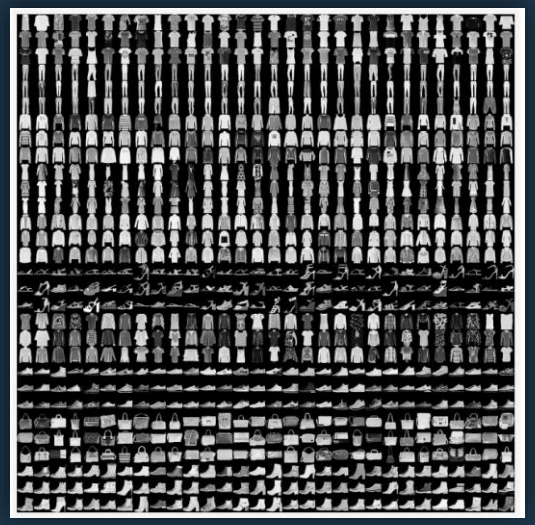
1 **Recolectar** el conjunto de datos

Realizado esto, tendremos nuestro conjunto de datos dividido en entrenamiento y *testing*. Cuando importamos un conjunto de datos, una muy buena práctica es pasar un tiempo considerable analizando los datos y realizando consultas simples.

Por ejemplo:

- Imprimir una parte del conjunto de datos
- Ver la forma del conjunto de datos
- Imprimir una imagen en particular
- Ver el tipo de dato (*float, integer, etc*)
- Cantidad de datos
- Dimensión en caso de tratarse de imágenes

1 **Recolectar** el conjunto de datos



Ejecutar las acciones mencionadas anteriormente, nos evitará grandes dolores de cabeza a futuro. En nuestro caso de estudio:

Por ejemplo:

- 70 000 imágenes
- 10 categorías
- Baja resolución 28x28 pixeles
- 60 000 imágenes para entrenamiento
- 10 000 imágenes para testing

2 **Construir** el modelo

La construcción del modelo, implica la configuración de las distintas capas de este, para luego compilar el mismo.

El bloque básico de una red neuronal es una capa.

Las capas extraen representaciones de los datos introducidos en ellas.

Y, de ser afortunados, estas representaciones son más significativas para el problema en cuestión.

2 **Construir** el modelo

```
model = keras.Sequential([ #indicamos que utilizaremos Sequential API
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation=tf.nn.relu),#Layer 1
keras.layers.Dense(10, activation=tf.nn.softmax)#Layer 2
])
```

`Keras.layers.Flatten`, transforma las imágenes de un array de 2 dimensiones (28x28 pixeles) a un array de una dimensión de 28x28 = 784 pixeles.

Esta capa no tiene parámetro de aprendizaje, simplemente modifica el formato de la información.

2 **Construir** el modelo

```
model = keras.Sequential([ #indicamos que utilizaremos Sequential API
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation=tf.nn.relu),#Layer 1
keras.layers.Dense(10, activation=tf.nn.softmax)#Layer 2
])
```

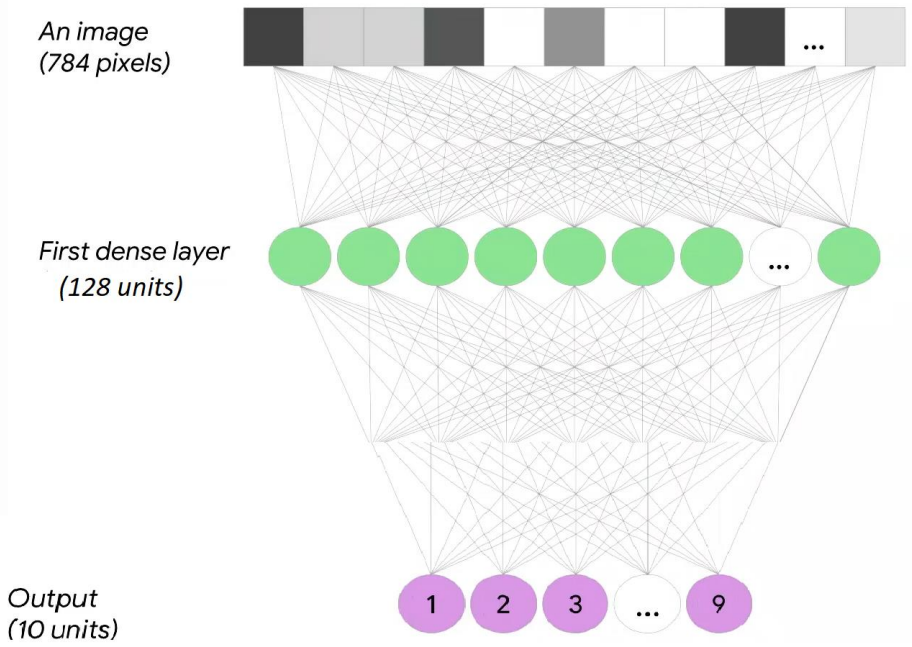
Luego de formateados, la red consiste en una secuencia de dos capaz tf.keras.layers.Dense. Estas son capas totalmente conectadas de la red. La primer capa tiene 128 nodos (o neuronas), mientras que la segunda y ultima, es una capa de 10 nodos softmax, es decir, cuenta con un array de 10 probabilidades que suman 1.

Cada nodo contiene un score que indica la probabilidad de que la imagen analizándose pertenezca a una clase.

2

CONSTRUIR EL MODELO

Representación de ***nuestra red***




```
keras.layers.Flatten(input_shape=(28, 28))
```

```
keras.layers.Dense(128, activation=tf.nn.relu)
```

```
keras.layers.Dense(10, activation=tf.nn.softmax)
```

Es esperable, que si agregamos, en la forma correcta, más capas y mayor cantidad de neuronas por capa a nuestra red, tenga un mejor desempeño y por tanto mayor capacidad de identificar distintos patrones.

2

CONSTRUIR EL MODELO

Compilado ***del modelo***

Antes de que el modelo este listo para ser entrenado, requiere algunas otras configuraciones. Los siguientes puntos son agregados durante la compilación del modelo:

Loss function

Mide que tan preciso es el modelo durante el entrenamiento. Trataremos de minimizar esta función, con el objetivo de "dirigir" el modelo en la dirección correcta.

2

CONSTRUIR EL MODELO

Compilado ***del modelo***

Optimizer

Esto indica como el modelo se actualiza basado en los datos que ve y en la función de costo (loss function)

Metrics

Son utilizadas para monitorear los pasos de entrenamiento y testing. En este ejemplo usaremos exactitud, que es la fracción de las imágenes que están clasificadas correctamente.

2

CONSTRUIR EL MODELO

Compilado *del modelo*

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy’,
              Metrics =['accuracy'])
```

'optimizer’, este es el método con el que entrenaremos la red neuronal. Existen una gran cantidad de optimizadores que podemos utilizar, y generalmente dependiendo el problema existen los estándares. Es decir, para tal problema usamos tal optimizador. Es por esto que no entraremos mucho en detalle en este punto.

2

CONSTRUIR EL MODELO

Compilado *del modelo*

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy’,
              Metrics=['accuracy'])
```

'sparse_categorical_crossentropy’, a pesar de su complejo nombre, simplemente significa que se contrastará lo que la red predice con lo que se desea predecir. Esto es, dado un número a la red, esta nos ‘dirá’ de que número se trata con determinada probabilidad de acierto. Por ejemplo, con 5% de probabilidad es 0, con 10% de probabilidad es 1 y así sucesivamente.

2

CONSTRUIR EL MODELO

Compilado *del modelo*

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy’,
              Metrics=['accuracy'])
```

'sparse_categorical_crossentropy’, a pesar de su complejo nombre, simplemente significa que se contrastará lo que la red predice con lo que se desea predecir. Esto es, dado un número a la red, esta nos ‘dirá’ de que número se trata con determinada probabilidad de acierto. Por ejemplo, con 5% de probabilidad es 0, con 10% de probabilidad es 1 y así sucesivamente.

3 *Entrenamiento*

En general, la manera en que se entrena las redes neuronales consiste en el uso de gradiente descendiente. Este método permite encontrar el mínimo de una función de múltiples variables.

Es por esto, que se utiliza dicho método para minimizar la función de costo o *loss function*. Si se minimiza la función de costo, el error de nuestro estimador es menor y por tanto, este es más preciso.

3

Entrenamiento

Supongamos que queremos resolver una regresión lineal donde tenemos una ecuación del tipo `y = mx + n`. Lo que buscamos es encontrar m y n tal que nuestra recta se ajuste de la mejor forma a nuestros datos.

Para esto se utiliza gradiente descendente.

Las redes neuronales, funcionan en una forma similar, con la diferencia de que en lugar de contar con un solo parámetro, contamos con cientos de variables X1, X2,...,Xn los cuales tratamos de aprender.

3

Entrenamiento

En pocas palabras, lo que el método hace es:

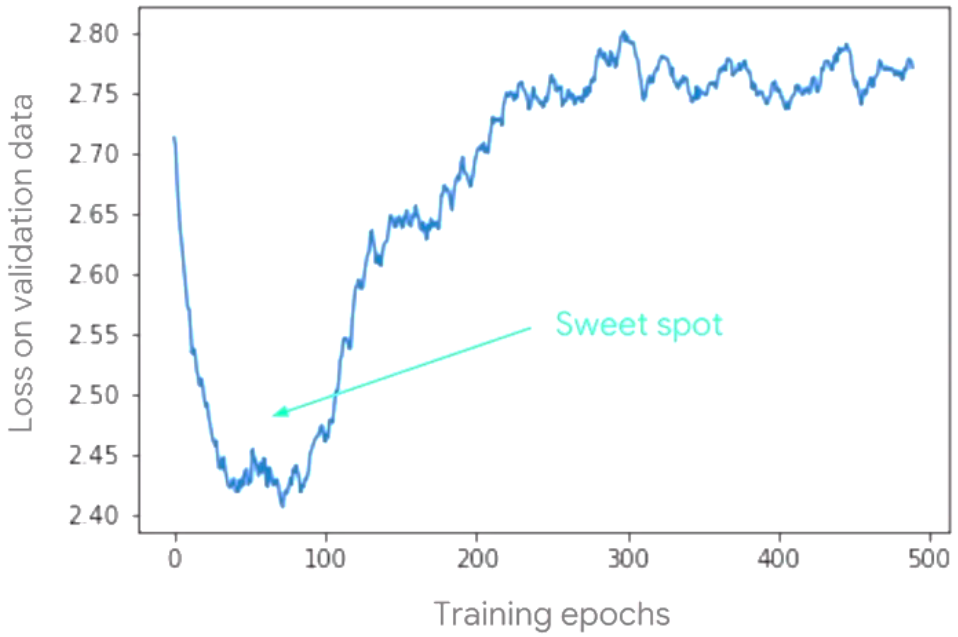
Comenzar con unos valores aleatorios, los cuales va ajustando luego de iterar durante un tiempo hasta que la red neuronal se ajusta mejor al reconocimiento de dígitos.

3 Entrenamiento

Para entrenar el modelo, bastará con la siguiente línea:

```
model.fit(train_images, train_labels, epochs=5)
```

Epochs significa un único barrido sobre todo el conjunto de datos. Esto se realiza en un conjunto de *batches* (lotes) más pequeños. El número 5 representa la cantidad de lotes.



Una forma de hallar el mejor número de épocas es utilizando la gráfica mostrada.

4 ***Evaluar*** modelo

VALORES >>> **CLASIFICACIÓN** >>> **OBSERVACIÓN**

Evaluar el modelo consiste en tomar nuevos valores, clasificarlos utilizando mi red, y observar que tan precisa es esta clasificación contrastándola con otras predicciones realizadas para poder determinar si existe o no algún inconveniente como puede ser el sobreajuste o *overfitting*.

El ***overfitting*** se da cuando un modelo de aprendizaje automático tiene un peor rendimiento en los datos nuevos que en sus datos de entrenamiento. Se suele decir que el modelo se ajusta mucho al conjunto de datos en particular, dificultándosele la predicción de nuevas entradas.

4 *Evaluar* modelo

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

En estás líneas simplemente utilizamos el método `evaluate` con las imágenes de *testing* e imprimimos el acierto que obtenemos para poder analizar el resultado.

5

Predecir

Realizado los pasos anteriores solo bastará con utilizar nuestro modelo para predecir según nuestro conjunto de datos de *testing*. Realizamos esto en la siguiente línea:

```
predictions = model.predict(test_images)
```

Una predicción (`prediction`) es un array de 10 números. Cada número indica un valor que corresponde a que tanto el modelo considera esa imagen, se asemeja con cada uno de los 10 artículos de ropa.

5 *Predecir*

Podremos ver que etiqueta tiene el valor más alto en la entrada 0:

```
np.argmax(predictions[0])
```

Dentro de **IBM Watson Studio** tendremos otras verificaciones y muestras del desempeño de nuestro modelo las cuales podremos ejecutar para visualizar.

En TensorFlow existen muchas formas de hacer las cosas. Siempre debemos tratar de hacer las cosas de la forma más sencilla que se adecúe a nuestro problema.

Cuando comienzas con ML, no te preocupes tanto por la performance, ¡reduce la complejidad!

¿Y qué sucede si manejo **una gran cantidad de datos**?

tf.data

Manejo de datos con *tf.data*

Cuando escribimos nuestro *hello world*, utilizamos la base de datos fashion-MNIST, en formato Numpy (vectores).

Sin embargo, si trabajamos con un conjunto de datos mayor, si estamos leyendo imágenes del disco o si estamos bajando los datos de la *cloud*, hay varios detalles y es donde la *performance* comienza a importar.

Manejo de datos con *tf.data*

En esto puede ayudar **tf.data**.

Cuenta con muchas funcionalidades de ordenamiento que permitirán solucionar los problemas antes mencionados.

¡Veamos un ejemplo!

Manejo de datos con *tf.data*

```
import tensorflow as tf
tf.enable_eager_execution() #1) Habilitamos eager execution

#2) Descargamos un conjunto de datos
(train_images,train_labels) , (test_images, test_labels)
= tf.keras.datasets.mnist.load_data()

#3) Creamos el dataset
dataset = tf.data.Dataset.from_tensor_slices((images, labels))
          .shuffle(1000)
          .batch(32)

#4) Iteramos a través del dataset
for image, label in dataset:
print(image)
```

¿Notas algo raro en el **código anterior**?

¿Recuerdas la idea de ***sesión***?

Manejo de datos con *tf.data*

Si miramos el script anterior, notamos que en ningún momento nos referimos a una sesión o grafo. Incluso, si miramos el punto 4 del script, vemos que la forma de iteración es la misma que usando Python normalmente

¿Cómo es esto posible?

Eager execution

A partir de la versión 1.8 de TensorFlow, simplemente escribiendo la segunda línea del script, podremos utilizar `tensorflow`, desarrollando tal cual lo hacemos en Python.

Hasta aquí llega el **Dojo de TensorFlow**.

Gracias.

Esperamos que haya sido de
tu agrado y beneficio.

Host:
Juan Fajardo
IBM Innovation Lab Uruguay
Juan.Fajardo@ibm.com

¿Preguntas?

</ilab>

IBM