

TensorFlow



JUNIO DE 2020

UTP

1 CONTENIDO

1	CONTENIDO	0
2	PRESENTACIÓN	2
3	MARCO TEÓRICO.....	3
4	OBJETIVOS	4
4.1	OBJETIVO GENERAL	4
4.2	OBJETIVOS ESPECIFICOS	4
5	DISEÑO APLICACIÓN.....	5
5.1	MODELO GENERAL	5
5.2	ARQUITECTURA	6
5.3	HERRAMIENTAS.....	7
5.4	EJEMPLO CODIGO BÁSICO	¡Error! Marcador no definido.
6	EJEMPLO DE APLICACIÓN	12
6.1	PRESENTACIÓN.....	12
7	CONCLUSIONES.....	32
7.1	CONCLUSIONES DEL PROYECTO	32

2 PRESENTACIÓN

Desde el curso de Computación Blanda se ha propuesto realizar un análisis de diferentes tecnologías referentes a la asignatura. En el presente documento se hará un análisis exhaustivo a la plataforma TensorFlow, sus ventajas, documentación, alcance, herramientas y más.

AUTORES: Sebastián Molina Loaiza

Daniel Patiño Rojas

3 MARCO TEÓRICO

En pleno siglo XXI y con el apogeo de IoT, los procesos de Machine Learning se han visto ampliamente usados, investigados y mejorados. El continuo uso de la tecnología para reconocimiento de imágenes, procesos, patrones, etc., ha dado pie para que las herramientas que se disponen en el área de Machine Learning tengan un crecimiento vertiginoso y se prevé que sigan evolucionando a pasos grandes debido a que el Machine Learning se ha convertido en algo fundamental en los procesos de IoT y un foco de inversión constante proveniente de gigantes tecnológicos que ven en este campo el potencial del futuro.

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

Actualmente es utilizado tanto en la investigación como en los productos de Google. TensorFlow fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre del 2015. TensorFlow es el sistema de aprendizaje automática de segunda generación de Google Brain, liberado como software de código abierto el 9 de noviembre del 2015, mientras la implementación de referencia se ejecuta en dispositivos aislados. TensorFlow puede correr en múltiples CPU y GPU (con extensiones opcionales de CUDA para informática de propósito general en unidades de procesamiento gráfico).

TensorFlow está disponible para Linux de 64 bits, macOS y plataformas móviles que incluyan Android e iOS.

Los cálculos de TensorFlow están expresados como stateful dataflow graphs. El nombre TensorFlow deriva de las operaciones que tales redes neuronales realizan sobre arrays multidimensionales de datos. Estos arrays multidimensionales son referidos como “tensores”. En junio de 2016, Jeff Dean de Google declaró que 1500 repositorios de GitHub mencionaron a TensorFlow, de los cuales 5 repositorios eran de Google. Los tensores que se mencionan anteriormente son construcciones matemáticas fundamentales en campos como la física y la ingeniería. Históricamente, sin embargo los tensores han hecho menos avances en la informática que tradicionalmente se ha asociado más con la matemática discreta y la lógica. Este estado de cosas ha comenzado a cambiar significativamente con la llegada del aprendizaje de máquina. El aprendizaje de máquina y su fundamento en la matemática vectorial continua, este se basa en la manipulación y el cálculo de los tensores, como ejemplo para explicar lo anterior es un tensor escalar, un único valor constante

extraído de los números reales (los números reales son números decimales de precisión arbitraria, con números positivos y negativos permitidos).

Matemáticamente nosotros denotamos los números reales por \mathbb{R} . mas informalmente, llamamos a un escalar un tensor de rango 0. Si los escalares son tensores de rango 0, entonces ¿Qué constituye un tensor de rango 1? Formalmente, hablando un tensor de rango 1 es un vector, una lista de números reales.

4 OBJETIVOS

4.1 GENERAL

Realizar una profundo análisis a TensorFlow, evidenciando principalmente en que consta, cuál es su arquitectura, cuales son los diferentes usos que se le puede dar a TensorFlow y como darlos.

4.2 ESPECÍFICOS

No	Objetivo Específico
1	Implementar y ver en acción TensorFlow
2	Comprender el funcionamiento de TensorFlow
3	Explorar y entender la construcción de Modelos basados en TensorFlow
4	Evidenciar el impacto del ML TensorFlow en los campos que requieren opiniones de un experto.
5	Ver como se aplica a procesos de investigación y experimentación

5 DISEÑO APLICACIÓN

TensorFlow al ser una librería de código abierto para el cálculo numérico se puede usar la programación con grafos de flujo de datos. Los nodos en el grafo representan operaciones matemáticas mientras que las conexiones o links del grafo representan los conjuntos de datos multidimensionales o tensores.

TensorFlow permite entre otras operaciones de construir y entrenar redes neuronales para detectar correlaciones y descifrar patrones, análogos al aprendizaje y razonamiento usados por los humanos. Actualmente se utiliza TensorFlow tanto en la investigación como para la producción de Google.

TensorFlow es demasiado popular en la actualidad dado que es la mejor biblioteca de todas porque está diseñada para ser accesible por todos. La biblioteca TensorFlow incorpora diferentes API para construir a escala profunda arquitectura de aprendizaje como CNN o RNN. TensorFlow se basa en el cálculo gráfico; permite al desarrollador visualizar la construcción de la red neuronal con TensorBoard. Esta herramienta es útil para depurar el programa, finalmente, TensorFlow está diseñado para ser implementado a escala.

5.1 MODELO GENERAL

El modelo general de TensorFlow se puede clasificar en ciertos puntos tales como

- Fase de entrenamiento: Esto es cuando se realiza el entrenamiento y el procesamiento de los datos
- Fase de ejecución o fase de inferencia: Una vez finalizado el entrenamiento, TensorFlow puede ejecutar en muchas plataformas diferentes tales como
 - Escritorio de Windows, MacOS y Linux
 - Cloud como servicio web
 - Dispositivos móviles como iOS y Android

El entrenamiento se puede realizar en distintas máquinas y luego ejecutarlo en una maquina diferente una vez se tenga el modelo entrenado.

TensorFlow cuenta con dos componentes los cuales son:

- Tensor: Es un vector o matriz de n-dimensiones que representa todos los tipos de datos. Todos los valores de un tensor contienen un tipo de datos idéntico con una forma conocida (o parcialmente conocida). La forma de los datos es la dimensionalidad de la matriz.

Un tensor puede originarse a partir de los datos de entrada o del resultado de un cálculo. En TensorFlow, todas las operaciones se llevan a cabo dentro de un gráfico. El gráfico es un conjunto de cálculos que tienen lugar sucesivamente. Cada operación se llama un nodo op y están comunicados entre sí. Los gráficos describen las operaciones y las conexiones entre los nodos, sin embargo, no muestran los valores. El borde de los nodos es el tensor, es decir, una forma de llenar la operación con datos.

- Gráficos: TensorFlow hace uso de un marco gráfico. El gráfico recoge y describe todos los cálculos de series realizados durante el entrenamiento. El gráfico tiene muchas ventajas:
 - Se hizo para ejecutarse en múltiples CPU o GPU e incluso en el sistema operativo móvil
 - La portabilidad del gráfico permite conservar los cálculos para uso inmediato o posterior. El gráfico se puede guardar para ser ejecutado en el futuro.
 - Todos los cálculos en el gráfico se realizan conectando tensores juntos.
 - Un tensor tiene un nodo y un borde. El nodo lleva la operación matemática y produce una salida de puntos finales. Los bordes de los nodos explican las relaciones de entrada/salida entre nodos.

Adicional a estos componentes sobre cómo funciona TensorFlow, también cuenta con:

- Un búfer de protocolos de grafo
- Un tiempo de ejecución que ejecuta el grafo (distribuido)

Estos dos mencionados anteriormente son análogos al código de Python y al intérprete de Python. Tal como se implementa el intérprete de Python en múltiples plataformas de hardware para ejecutar el código de Python, TensorFlow permite ejecutar el grafo en diversas plataformas de hardware.

¿Qué API's usar? Se deben de usar el nivel de abstracción más alto que resuelva el problema. Los niveles de abstracción más altos son más fáciles de usar, pero también son menos flexibles (por su diseño). Se recomienda comenzar con una API de nivel alto y poner todo en funcionamiento. Si se necesita mas flexibilidad adicional por cuestiones de modelos especiales, se recomienda usar un nivel más bajo pero teniendo en cuenta que cada nivel de la API se crea de niveles inferiores.

5.2 ARQUITECTURA

TensorFlow se construyó pensando en el código abierto y en la facilidad de ejecución y escalabilidad. Este permite ser ejecutado en la nube pero también localmente. La idea es

que cualquiera pueda ejecutarlo. Se puede ver a personas que lo ejecutan en una sola máquina, un solo dispositivo con una sola CPU o GPU o en grandes clústeres, la disparidad es muy alta. Si realmente se desea escalar, la nube es un gran lugar para hacerlo, se puede obtener mucha automatización. Google quiere asegurarse de que todo el mundo use TensorFlow.

La arquitectura de TensorFlow funciona en tres partes:

- Procesamiento previo de los datos
- Construir el modelo
- Entrenar y estimar el modelo

Se llama TensorFlow por que toma como entrada una matriz multidimensional. Este puede construir una especie de diagrama de flujo de operaciones llamado Graph que desea realizar en esa entrada. La entrada entra en un extremo y luego fluye a través de este sistema de múltiples operaciones y sale el otro extremo como salida.

Es por esto que se llama TensorFlow, por que el tensor entra en él y fluye a través de una lista de operaciones saliendo al otro lado.

5.3 HERRAMIENTAS

TensorFlow es una plataforma informática para la compilación de modelos de aprendizaje automático. Ofrece diversos kits de herramientas que permiten crear modelos con el nivel de abstracción preferido. Se pueden usar API's de nivel inferior para compilar modelos definiendo una serie de operaciones matemáticas, O bien, se puede usar una API de nivel superior para especificar arquitecturas predefinidas, como regresores lineales o redes neuronales.

En la siguiente figura, se muestra la jerarquía actual de los kits de herramientas de TensorFlow.



- TensorFlow Estimators es una API de alto nivel orientada a objetos
- Tf.layers, tf.losses y tf.metrics son bibliotecas reutilizables para componentes comunes del modelo
- TensorFlow también proporciona operaciones para Python
- TensorFlow también proporciona operaciones para C++, uniendo esto ambos kernel's (Python y C++)
- Cada kernel se define para que funcione en una o más plataformas

A continuación se resumen los objetivos de las diferentes capas

Kits de herramientas	Descripción
Estimador (tf.estimator)	API de POO de alto nivel
tf.layers tf.losses tf.metrics	Bibliotecas de componentes comunes del modelo
TensorFlow	API de nivel inferior

5.4 EJEMPLOS DE APLICACIÓN TOMADOS DE LA LITERATURA EXISTENTE

Un ejemplo simple de TensorFlow es el siguiente:

```
import numpy as np
import tensorflow as tf

X_1 = tf.placeholder(tf.float32, name="X_1")
X_2 = tf.placeholder(tf.float32, name="X_2")

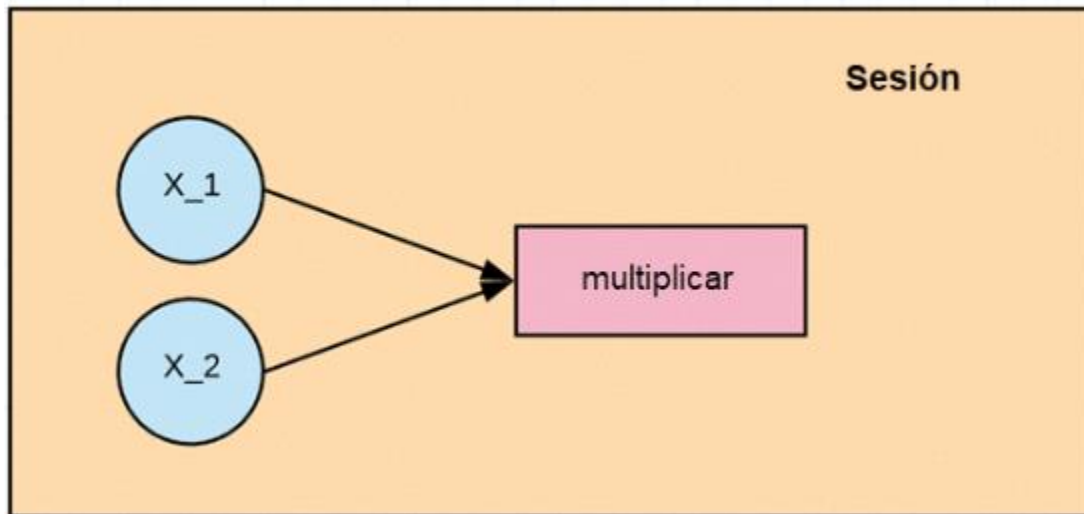
multiply = tf.multiply(X_1, X_2, name="multiply")

with tf.Session() as session:
    result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:[4,5,6]})
    print (result)
```

En las dos primeras líneas de código, se importa TensorFlow como tf. Con Python, es una práctica muy común usar un nombre corto para una biblioteca. La ventaja es evitar escribir el nombre completo de la biblioteca cuando necesitamos usarla. Por ejemplo, podemos importar TensorFlow como tf, y llamar a tf cuando queremos usar una función TensorFlow.

Vamos a practicar el flujo de trabajo elemental de TensorFlow como un ejemplo simple. Vamos a crear un gráfico computacional que multiplica dos números juntos.

Durante el ejemplo, multiplicaremos X_1 y X_2 juntos. TensorFlow creará un nodo para conectar la operación. En el ejemplo, se llama multiplicar. Cuando se determina el gráfico, los motores computacionales de TensorFlow se multiplicarán juntos X_1 y X_2.



Finalmente, ejecutaremos una sesión de TensorFlow que ejecutara el gráfico computacional con los valores de X_1 y X_2 e imprimirá el resultado de la multiplicación.

Vamos a definir los nodos de entrada X_1 y X_2 . Cuando creamos un nodo en TensorFlow tenemos que elegir qué tipo de nodo crear. Los nodos X_1 y X_2 serán un nodo marcador de posición. El marcador de posición asigna un nuevo valor cada vez que hacemos un cálculo. Vamos a crearlos como un nodo de marcador de posición de punto TF. Esto se realiza en las líneas 4 y 5.

Cuando creamos un nodo marcador de posición, tenemos que pasar el tipo de datos será la adición de números aquí para que podamos utilizar un tipo de datos de punto flotante, vamos a utilizar `tf.float32`. También necesitamos darle un nombre a este nodo. Este nombre aparecerá cuando veamos las visualizaciones gráficas de nuestro modelo. Vamos a nombrar este nodo X_1 pasando un parámetro llamado `nombre` con un valor X_1 y ahora vamos a definir X_2 de la misma manera.

Ahora podemos definir el nodo que hace la operación multiplicación. En TensorFlow podemos hacer esto creando un nodo `tf.multiply`.

Pasaremos los nodos X_1 y X_2 al nodo de multiplicación. Le dice a TensorFlow que vincule esos nodos en el gráfico computacional, por lo que le estamos pidiendo que extraiga los valores de X , E , Y y multiplique el resultado. También vamos a dar al nodo de multiplicación el nombre de `multiplicar`. Es la definición completa de nuestro gráfico computacional simple.

Ahora podemos ejecutar la aplicación. Para ejecutar operaciones en el gráfico, tenemos que crear una sesión. En TensorFlow, lo hace `tf.session()`. Ahora que tenemos una sesión

podemos pedirle a la sesión que ejecute operaciones en nuestro gráfico computacional llamado sesión. Para ejecutar el cálculo necesitamos usar `run`.

Cuando se ejecuta la operación de adición, va a ver que necesita tomar los valores de los nodos `X_1` y `X_2`, por lo que también necesitamos alimentar valores para `X_1` y `X_2`. Podemos hacerlo suministrando el parámetro llamado `feed_dict`. Pasamos el valor 1,2,3 para `X_1` y 4,5,6 para `X_2`.

Imprimimos los resultados con `print`. Deberíamos de ver 4, 10 y 18.



6 RESULTADOS

6.1 PRESENTACIÓN

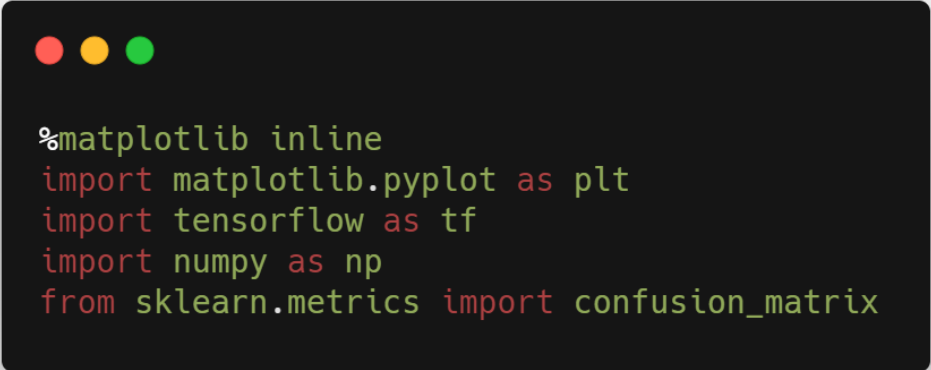
Reconocimiento de dígitos escrito a mano

En este primer tutorial mostraremos el flujo de trabajo básico al usar TensorFlow con un modelo lineal simple. El ejemplo que seguiremos para ello es desarrollar una aplicación que reconozca los dígitos escritos a mano.

Comenzaremos por implementar el modelo más sencillo posible. En este caso, haremos una regresión lineal como primer modelo para el reconocimiento de los dígitos tratados como imágenes.

Primero procederemos a cargar un conjunto de imágenes de los dígitos escritos a mano del conjunto de datos MNIST. Luego procederemos a definir y optimizar un modelo matemático de regresión lineal en TensorFlow.

Primero cargaremos algunas librerías.



```
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from sklearn.metrics import confusion_matrix
```

Descargamos los datos, el conjunto de datos MNIST es de aproximadamente 12MB y se descargará automáticamente si no se encuentra en la ruta dada.

Posterior a la descarga verificamos los datos.

```
# Load Data....  
from tensorflow.examples.tutorials.mnist import input_data  
data = input_data.read_data_sets("data/MNIST/", one_hot=True)
```

```
print("Size of:")  
print("- Training-set:\t\t{}".format(len(data.train.labels)))  
print("- Test-set:\t\t{}".format(len(data.test.labels)))  
print("- Validation-set:\t{}".format(len(data.validation.labels)))
```

Una vez cargado el conjunto de datos con la codificación One-Hot. Esto significa que las etiquetas se han convertido de un solo número a un vector cuya longitud es igual a la cantidad de clases posibles. Todos los elementos del vector son cero excepto el elemento i ésimo que toma el valor uno; y significa que la clase es i .

Las etiquetas codificadas de One-Hot para las primeras 5 imágenes en el conjunto de prueba son:

```
data.test.labels[0:5, :]
```

Se obtiene:

```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]])
```

Como se observa tenemos cinco vectores donde cada componente tiene valores cero excepto en la posición de la componente que identifica la clase. Cuyo valor es 1.

Como necesitamos las clases como números únicos para las comparaciones y medidas de rendimiento, procedemos a convertir estos vectores codificados como One-Hot a un solo número tomando el índice del elemento más alto. Tenga en cuenta que la palabra 'clase' es una palabra clave utilizada en Python, por lo que necesitamos usar el nombre 'cls' en su lugar.

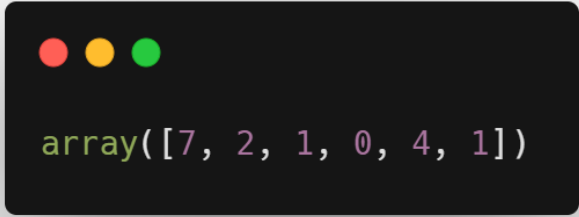
Para codificar estos vectores a números:

```
data.test.cls = np.array([label.argmax() for label in data.test.labels])
```

Ahora podemos ver la clase para las primeras cinco imágenes en el conjunto de pruebas

```
print (data.test.cls[0:5])
```

Se obtiene

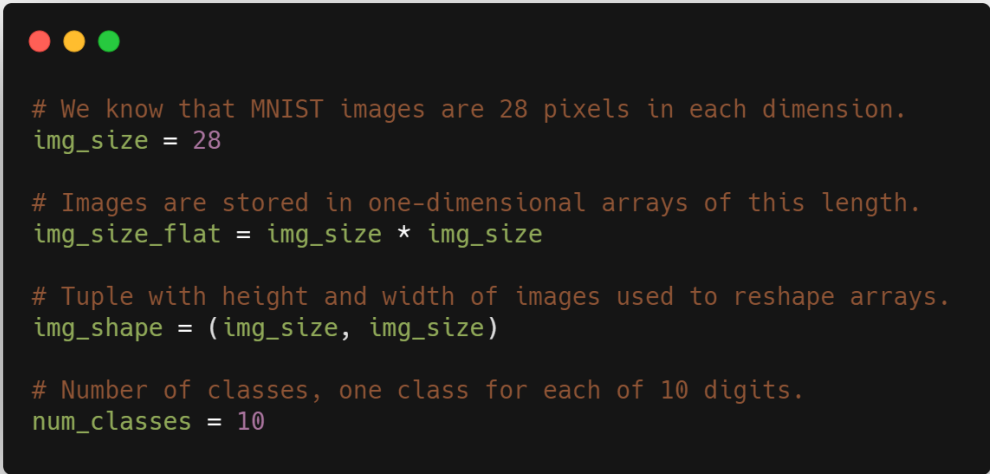


```
array([7, 2, 1, 0, 4, 1])
```

Comparemos estos con los vectores codificados One-Hot de arriba. Por ejemplo, la clase para la primera imagen es 7, que corresponde a un vector codificado One-Hot donde todos los elementos son cero excepto el elemento con índice 7.

El siguiente paso es definir algunas variables que se usaran en el código. Estas variables y sus valores constantes nos permitirán tener un código más limpio y fácil de leer.

Las definimos de la siguiente manera:



```
# We know that MNIST images are 28 pixels in each dimension.  
img_size = 28  
  
# Images are stored in one-dimensional arrays of this length.  
img_size_flat = img_size * img_size  
  
# Tuple with height and width of images used to reshape arrays.  
img_shape = (img_size, img_size)  
  
# Number of classes, one class for each of 10 digits.  
num_classes = 10
```

Ahora crearemos una función que es utilizada para trazar 9 imágenes en una cuadrícula de 3x3 y escribir las clases verdaderas y predichas debajo de cada imagen.


```
def plot_images(images, cls_true, cls_pred=None):
    assert len(images) == len(cls_true) == 9

    # Create figure with 3x3 sub-plots.
    fig, axes = plt.subplots(3, 3)
    fig.subplots_adjust(hspace=0.5, wspace=0.5)

    for i, ax in enumerate(axes.flat):
        # Plot image.
        ax.imshow(images[i].reshape(img_shape), cmap='binary')

        # Show true and predicted classes.
        if cls_pred is None:
            xlabel = "True: {0}".format(cls_true[i])
        else:
            xlabel = "True: {0}, Pred: {1}".format(cls_true[i], cls_pred[i])

        ax.set_xlabel(xlabel)

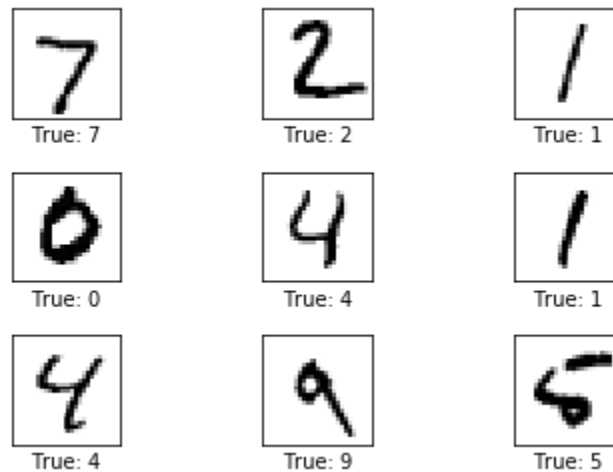
        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])
```

Dibujemos algunas imágenes para ver si los datos son correctos

```
# Get the first images from the test-set.
images = data.test.images[0:9]

# Get the true classes for those images.
cls_true = data.test.cls[0:9]

# Plot the images and labels using our helper-function above.
plot_images(images=images, cls_true=cls_true)
```



El propósito de usar la librería de TensorFlow es generar un grafo computacional que se puede ejecutar de manera mucho más eficiente. TensorFlow puede ser más eficiente que NumPy (en algunos casos), puesto que TensorFlow conoce todo el grafo de cálculo y su flujo de datos que debe ejecutarse, mientras que NumPy solo conoce el cálculo de la operación matemática que se esté ejecutando en un determinado momento.

TensorFlow también puede calcular automáticamente los gradientes que se necesitan para optimizar las variables del grafo a fin de que el modelo funcione mejor. Esto se debe a que el grafo es una combinación de expresiones matemáticas simples, por lo que el gradiente de todo el grafo se puede calcular utilizando la regla de cadena para el cálculo de las derivadas al optimizar la función de coste.

Un grafo de TensorFlow de manera general consta de las siguientes partes:

- Las variables de marcador de posición (Placeholder variables) utilizadas para cambiar las entradas (datos) al grafo (links entre los nodos).
- Las variables del modelo.
- El modelo, que es esencialmente una función matemática que calcula los resultados dada la entrada en las variables del marcador de posición (Placeholder variables) y las variables del modelo (recuerde que desde el punto de vista de TensorFlow las operaciones matemáticas son tratadas como nodos del grafo).
- Una medida de costo que se puede usar para guiar la optimización de las variables.
- Un método de optimización que actualiza las variables del modelo.


Además, el grafo TensorFlow también puede contener varias declaraciones de depuración, por ejemplo para que los datos de registro se muestren utilizando TensorBoard, que no se trata en este tutorial.

Las variables de marcador de posición (Placeholder variables) sirven de entrada al grafo, y que podemos cambiar a medida que ejecutamos operaciones sobre el grafo.

Pasemos a definir las variables de marcador de posición para las imágenes de entrada (Placeholder variables), a la que llamaremos *x*. Hacer esto nos permite cambiar las imágenes que se ingresan al grafo de TensorFlow.

El tipo de dato que se introducen en el grafo, son vectores o matrices multidimensionales (denotados tensores). Estos tensores son arrays multidimensionales, cuya forma es `[None, img_size_flat]`, donde `None` significa que el tensor puede contener un número arbitrario de imágenes, siendo cada imagen un vector de longitud `img_size_flat`.

Definimos:



```
x = tf.placeholder(tf.float32, [None, img_size_flat])
```


Note que en la función `tf.placeholder`, hemos de definir el tipo de dato, que en éste caso es un `float32`.

A continuación, definimos la variable de marcador de posición para las etiquetas verdaderas asociadas con las imágenes que se ingresaron en la variable de marcador de posición *x*. La forma de esta variable de marcador de posición es `[None, num_classes]`, lo que significa que puede contener una cantidad arbitraria de etiquetas y cada etiqueta es un vector de longitud de `num_classes`, que es 10 en nuestro caso.



```
y_true = tf.placeholder(tf.float32, [None, num_classes])
```

Finalmente, tenemos la variable de marcador de posición para la clase verdadera de cada imagen en la variable de marcador de posición x . Estos son enteros y la dimensionalidad de esta variable de marcador de posición se pre-define como `[None]`, lo que significa que la variable marcador de posición es un vector unidimensional de longitud arbitraria.



```
y_true_cls = tf.placeholder(tf.int64, [None])
```

Como hemos indicado anteriormente, en éste ejemplo vamos a emplear un simple modelo matemático de regresión lineal, es decir, que vamos a definir una función lineal donde se multiplica las imágenes en la variable de marcador de posición x por una variable w que llamaremos pesos y luego agrega un sesgos (bias) que llamaremos b .

Por tanto:


$$\text{Logist} = wx + b$$

El resultado es una matriz de forma `[num_images, num_classes]`, dado que x tiene la forma `[num_images, img_size_flat]` y la matriz de pesos w tienen la forma `[img_size_flat, num_classes]`, por lo que la multiplicación de éstas dos matrices es una matriz con cuya forma es `[num_images, num_classes]`. Luego el vector de sesgos b se agrega a cada fila de esa matriz resultante.

Nota hemos usado el nombre logits para respetar la terminología típica de TensorFlow, pero se puede llamar a la variable de otra manera.


Esta operación en TensorFlow la definimos de la siguiente manera:

- Primero declaramos la variable w (tensor de pesos) y lo inicializamos con cero




```
w = tf.Variable(tf.zeros([img_size_flat, num_classes]))
```

- Luego declaramos la variable `b` (tensor de sesgo) y lo inicializamos con cero



```
b = tf.Variable(tf.zeros([ num_classes]))
```

- Definimos nuestro modelo lineal




```
logits = tf.matmul(x, w) + b
```

En la definición del modelo logits, hemos usado la función `tf.matmul`. Esta función nos devuelve el valor de multiplicar el tensor `x` por el tensor `w`.


El modelo logits es una matriz con filas `num_images` y columnas `num_classes`, donde el elemento de la fila `i` enésima y la columna `j` enésima es una estimación de la probabilidad de que la imagen de entrada `i` enésima sea de la `j` enésima clase.

Sin embargo, estas estimaciones son un poco difíciles de interpretar, dado que los números que se obtienen pueden ser muy pequeños o muy grandes. El siguiente paso entonces sería normalizar los valores de tal modo que, para que cada fila de la matriz logits todos sus valores sumen uno, así el valor de cada elemento de la matriz restringido entre cero y uno. Con TensorFlow esto se calcula utilizando la función llamada `softmax` y el resultado se almacena en una nueva variable `y_pred`.



```
y_pred = tf.nn.softmax(logits)
```

Finalmente la clase predicha puede calcularse a partir de la matriz `y_pred` tomando el índice del elemento más grande en cada fila.



```
y_pred_cls = tf.argmax(y_pred, dimension=1)
```

Como hemos indicado anteriormente, el modelo de clasificación y reconocimiento de los dígitos escrito a mano que hemos implementado, es un modelo matemático de regresión lineal $\text{logits} = wx + b$. La calidad de predicción del modelo va depender de los valores óptimos de las variables w (tensor de pesos) y b (tensor de sesgos) dada una entrada x (tensor de imágenes). Por tanto, optimizar nuestro clasificador para la tarea del reconocimiento de los dígitos consiste en hacer un ajuste de nuestro modelo de tal forma que podamos encontrar los valores óptimos en el tensor w y b . Este proceso de optimización en estas variables es lo que se conoce como el proceso de entrenamiento o aprendizaje del modelo.

Para mejorar el modelo al clasificar las imágenes de entrada, de alguna manera debemos encontrar un método para cambiar el valor de las variables para las ponderaciones (w) y los sesgos (b). Para hacer esto, primero necesitamos saber qué tan bien funciona actualmente el modelo al comparar el resultado predicho del modelo `y_pred` con el resultado deseado `true`. La función de rendimiento que mide el error entre la salida real del sistema que se pretende modelar y la salida del núcleo estimador (el modelo), es lo que se conoce como función de coste. Diferentes funciones de coste se pueden definir.

La entropía cruzada (cross-entropy) es una medida de rendimiento utilizada en la clasificación. La entropía cruzada es una función continua que siempre es positiva y si la salida predicha del modelo coincide exactamente con la salida deseada, entonces la entropía cruzada es igual a cero. Por lo tanto, el objetivo de la optimización es minimizar la entropía cruzada para que se acerque lo más posible a cero cambiando los pesos w y los sesgos b del modelo.

TensorFlow tiene una función incorporada para calcular la entropía cruzada. Note que se usan los valores de los logits puesto que esta función del TensorFlow calcula el softmax internamente.

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y_true)
```

Una vez que hemos calculado la entropía cruzada para cada una de las clasificaciones de imágenes, tenemos una medida de qué tan bien se comporta el modelo en cada imagen individualmente. Pero para usar la entropía cruzada para guiar la optimización de las variables del modelo, necesitamos tener un solo valor escalar, así que simplemente tomamos el promedio de la entropía cruzada para todas las clasificaciones de imágenes.

Para esto:

```
cost = tf.reduce_mean(cross_entropy)
```

Ahora que ya tenemos una medida de costo que debe minimizarse, podemos crear un optimizador. En este caso usaremos uno de los métodos más utilizados conocido como Gradient, donde el tamaño de paso para el ajuste de las variables lo prefijamos en 0.5.

Tenga en cuenta que la optimización no se realiza en este momento. De hecho, nada se calcula en absoluto, simplemente agregamos el objeto optimizador al gráfico TensorFlow para su posterior ejecución.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5).minimize(cost)
```

Necesitamos algunas medidas de rendimiento más para mostrar el progreso al usuario. Creamos un vector de booleanos, donde verificamos si la clase predicha es igual a la clase verdadera de cada imagen.

```
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
```

Esto calcula la precisión (accuracy) de la clasificación y transforma los booleanos a floats, de modo que False se convierte en 0 y True se convierte en 1. Luego calculamos el promedio de estos números.

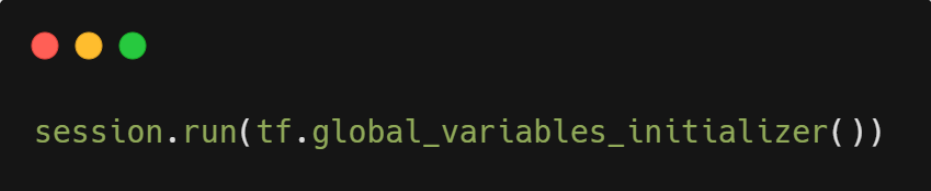
```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Una vez especificados todos los elementos de nuestro modelo, podemos ya crear el grafo. Para ello tenemos que crear una sesión para ejecutar luego el grafo:



```
session = tf.Session()
```

Inicializar variables: Las variables para pesos y sesgos deben inicializarse antes de comenzar a optimizarlas.



```
session.run(tf.global_variables_initializer())
```

Al tener 50.000 imágenes en el conjunto de entrenamiento, puede llevar mucho tiempo calcular el gradiente del modelo usando todas estas imágenes a durante el proceso de optimización. Por lo tanto, usamos un Stochastic Gradient Descent que solo usa un lote (batch) de imágenes en seleccionada aleatoriamente cada iteración del optimizador. Esto permite que el proceso de aprendizaje sea más rápido.

Creamos una función para realizar varias iteraciones de optimización para mejorar gradualmente los pesos w y los sesgos b del modelo. En cada iteración, se selecciona un nuevo lote de datos del conjunto de entrenamiento y luego TensorFlow ejecuta el optimizador utilizando esas muestras de entrenamiento. Fijamos ese lote de imágenes en 100 (`batch_size = 100`).

```
# batch of images
batch_size = 100

def optimize(num_iterations):
    for i in range(num_iterations):
        # Get a batch of training examples.
        # x_batch now holds a batch of images and
        # y_true_batch are the true labels for those images.
        x_batch, y_true_batch = data.train.next_batch(batch_size)

        # Put the batch into a dict with the proper names
        # for placeholder variables in the TensorFlow graph.
        # Note that the placeholder for y_true_cls is not set
        # because it is not used during training.
        feed_dict_train = {x: x_batch,
                           y_true: y_true_batch}

        # Run the optimizer using this batch of training data.
        # TensorFlow assigns the variables in feed_dict_train
        # to the placeholder variables and then runs the optimizer.
        session.run(optimizer, feed_dict=feed_dict_train)
```

Vamos a crear un conjunto de funciones que nos ayudaran a monitorizar el rendimiento de nuestro clasificador. Primero creamos un diccionario con los datos del conjunto de prueba que se utilizarán como entrada al grafo de TensorFlow.

```
feed_dict_test = {x: data.test.images,
                  y_true: data.test.labels,
                  y_true_cls: data.test.cls}
```

Función para imprimir la precisión (accuracy) de clasificación en el conjunto de prueba.

```
def print_accuracy():
    # Use TensorFlow to compute the accuracy.
    acc = session.run(accuracy, feed_dict=feed_dict_test)

    # Print the accuracy.
    print("Accuracy on test-set: {0:.1%}".format(acc))
```

Función para imprimir y trazar la matriz de confusión usando scikit-learn.

```
def print_confusion_matrix():
    # Get the true classifications for the test-set.
    cls_true = data.test.cls

    # Get the predicted classifications for the test-set.
    cls_pred = session.run(y_pred_cls, feed_dict=feed_dict_test)

    # Get the confusion matrix using sklearn.
    cm = confusion_matrix(y_true=cls_true,
                          y_pred=cls_pred)

    # Print the confusion matrix as text.
    print(cm)

    # Plot the confusion matrix as an image.
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

    # Make various adjustments to the plot.
    plt.tight_layout()
    plt.colorbar()
    tick_marks = np.arange(num_classes)
    plt.xticks(tick_marks, range(num_classes))
    plt.yticks(tick_marks, range(num_classes))
    plt.xlabel('Predicted')
    plt.ylabel('True')
```

Función para trazar los pesos del modelo. Se trazan 10 imágenes, una para cada dígito en el que el modelo está entrenado para reconocerlo.

```
def plot_weights():
    # Get the values for the weights from the TensorFlow variable.
    wi = session.run(w)

    # Get the lowest and highest values for the weights.
    # This is used to correct the colour intensity across
    # the images so they can be compared with each other.
    w_min = np.min(wi)
    w_max = np.max(wi)

    # Create figure with 3x4 sub-plots,
    # where the last 2 sub-plots are unused.
    fig, axes = plt.subplots(3, 4)
    fig.subplots_adjust(hspace=0.3, wspace=0.3)

    for i, ax in enumerate(axes.flat):
        # Only use the weights for the first 10 sub-plots.
        if i < 10:
            # Get the weights for the i'th digit and reshape it.
            # Note that w.shape == (img_size_flat, 10)
            image = wi[:, i].reshape(img_shape)

            # Set the label for the sub-plot.
            ax.set_xlabel("Weights: {0}".format(i))

            # Plot the image.
            ax.imshow(image, vmin=w_min, vmax=w_max, cmap='seismic')

        # Remove ticks from each sub-plot.
        ax.set_xticks([])
        ax.set_yticks([])
```

Ahora que ya tenemos todo lo necesario, vamos a ejecutar el clasificador y hacer algunas pruebas de rendimiento.

Como ya hemos inicializado las variables, lo primero que vamos a mirar es ver el nivel de precisión que éste tiene antes de ejecutar cualquier optimización.

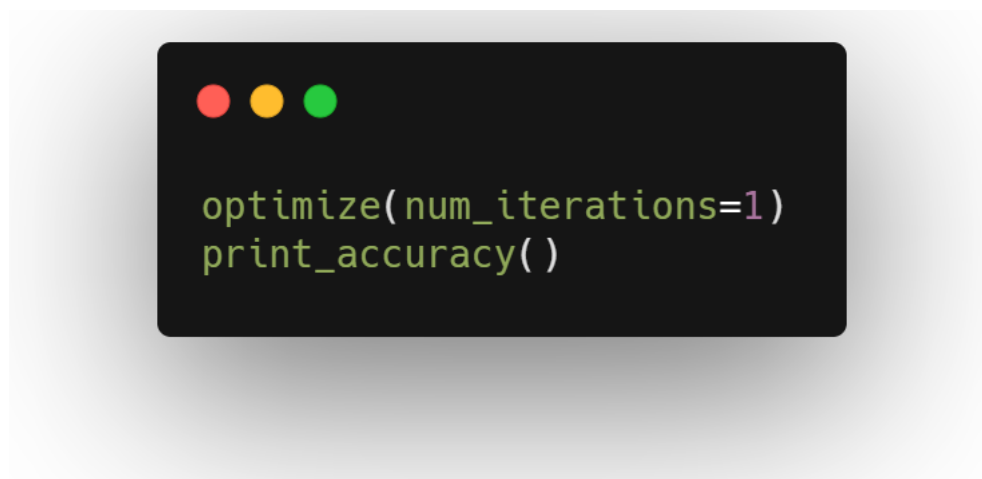
Ejecutemos la función accuracy que hemos creado:



Obtenemos: Accuracy on test-set: 9.8%.

La precisión en el conjunto de prueba es 9.8%. Esto se debe a que el modelo solo se ha inicializado y no se ha optimizado en absoluto.

Vamos a usar la función de optimización que hemos creado a una interacción:



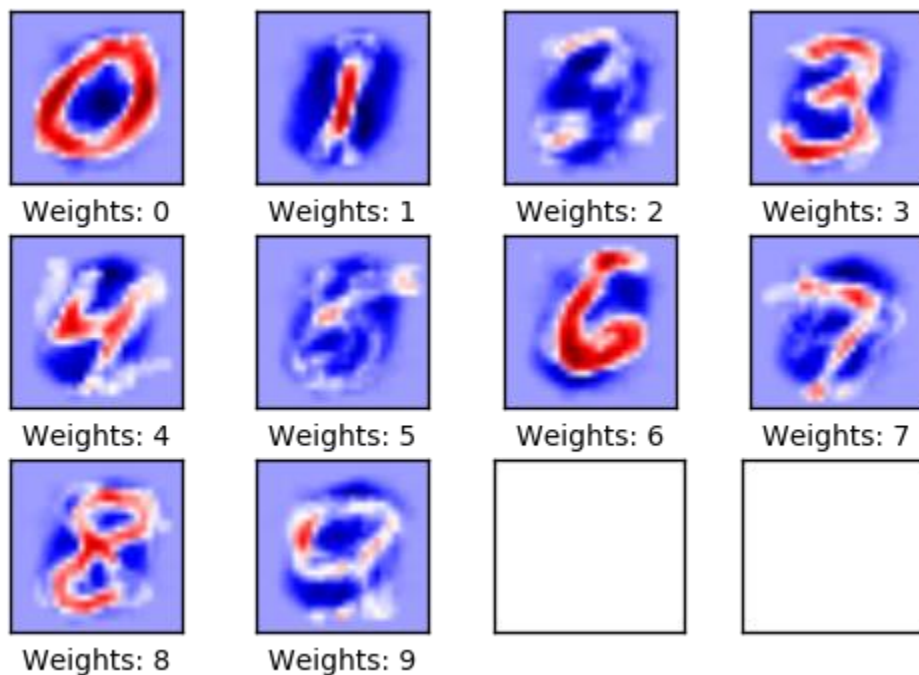
Resultado: Accuracy on test-set 40.9%.

Como se aprecia después de una única iteración de optimización, el modelo ha incrementado su precisión en el conjunto de prueba a 40.7%. Esto significa que clasifica incorrectamente las imágenes aproximadamente 6 de cada 10 veces.

Los pesos del tensor `w` también pueden trazarse como se muestra a continuación. Los pesos positivos toman los tonos rojos y los pesos negativos los tonos azules. Estos pesos pueden entenderse intuitivamente como filtros de imagen.

Usemos la función `plot_weights()`

```
plot_weights()
```



Por ejemplo, los pesos utilizados para determinar si una imagen muestra un dígito cero tienen una reacción positiva (roja) como la imagen de un círculo y tienen una reacción negativa (azul) a las imágenes con contenido en el centro del círculo.

De manera similar, los pesos utilizados para determinar si una imagen muestra el dígito 1, éste reacciona positivamente (rojo) a una línea vertical en el centro de la imagen, y reacciona negativamente (azul) a las imágenes con el contenido que rodea esa línea.

En estas imágenes, los pesos en su mayoría se parecen a los dígitos que se supone deben reconocer. Esto se debe a que solo se ha realizado una iteración de optimización, por lo que los pesos solo se entrenan en 100 imágenes. Después de entrenar en miles de

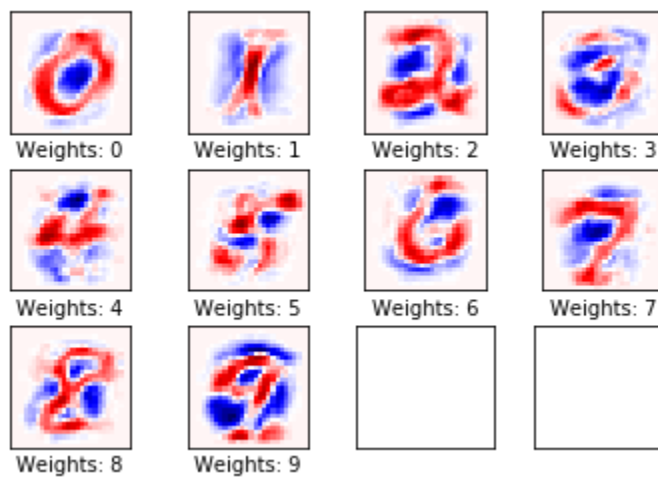
imágenes, los pesos se vuelven más difíciles de interpretar porque tienen que reconocer muchas variaciones de cómo se pueden escribir los dígitos.

Rendimiento después de 10 iteraciones

```
# We have already performed 1 iteration.
optimize(num_iterations=9)
print_accuracy()
```

El resultado: Accuracy on test-set 78.2%

```
plot_weights()
```

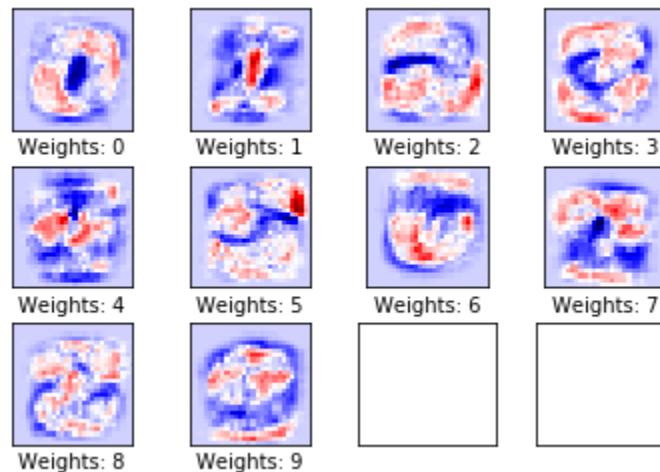


Rendimiento después de 1000 iteraciones de optimización

```
# We have already performed 1000 iteration.
optimize(num_iterations=999)
print_accuracy()
plot_weights()
```

Obtenemos: Accuracy on test-set 92.1%.

Los pesos serian



Después de 1000 iteraciones de optimización, el modelo solo clasifica erróneamente una de cada diez imágenes. Este modelo simple no puede alcanzar un rendimiento mucho mejor y, por lo tanto, se necesitan modelos más complejos. En los subsiguientes tutoriales crearemos un modelo más complejo usando redes neuronales que nos ayudará mejorar el rendimiento del nuestro clasificador.

Finalmente para tener una visión global de los errores cometidos por nuestro clasificador, vamos a analizar la matriz de confusión (confusion matrix). Usamos nuestra función `print_confusion_matrix()`.

```
print_confusion_matrix()

[[ 961    0    0    3    0    7    3    4    2    0]
 [    0 1097    2    4    0    2    4    2   24    0]
 [   12    8  898   23    5    4   12   12   49    9]
 [    2    0   10  927    0   28    2    9   24    8]
 [    2    1    2    2  895    0   13    4   10   53]
 [   10    1    1   37    6  774   17    4   36    6]
 [   13    3    4    2    8   19  902    3    4    0]
 [    3    6   21   12    5    1    0  936    2   42]
 [    6    3    6   18    8   26   10    5  883    9]
 [   11    5    0    7   17   11    0   16    9  933]]
```

7 CONCLUSIONES

7.1 CONCLUSIONES PROYECTO

No	Conclusiones
1	Google ha sido pionero desde su nacimiento en el mundo de la Inteligencia Artificial, impulsando la investigación y el desarrollo en este campo. Con TensorFlow han dado sin duda un pasito más, impulsando como siempre la innovación y abriendo el conocimiento a multitud de empresas, universidades, ingenieros y científicos que basándose en tecnologías abiertas como TensorFlow conseguirán logros fascinantes en los próximos años.
2	TensorFlow es una herramienta increíble que nos ofrece un marco de trabajo muy potente, pero que nos simplifica enormemente la complejidad interna que supone el manejo de algoritmos de aprendizaje profundo.
3	TensorFlow facilita la creación y uso de redes neuronales

- | | |
|---|--|
| 4 | Es una herramienta de fácil uso, escalable desde plataformas móviles a workstations. |
|---|--|

8 BIBLIOGRAFÍA

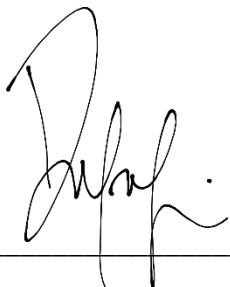
- <https://www.apsl.net/blog/2017/12/05/tensor-flow-para-principiantes-i/>
- <https://www.bbva.com/es/redes-neuronales-distribuidas-tensorflow-conclusiones/>
- <https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit?hl=es-419>
- <https://www.tensorflow.org/>

9 FIRMA DEL DOCUMENTO



NOMBRE: Sebastián Molina Loaiza

CÓDIGO: 1.088.311.096



NOMBRE: Daniel Patiño Rojas

CÓDIGO: 1.088.349.562