Buenas, hoy vamos a explicar un poco en esta entrada como funciona el framework que nos ofrece herramientas para programar tareas de aprendizaje automático, crear redes neuronales, deeplearning, deep reinforcement learning, etc. En definitiva lo que suelen denominarse hoy inteligencias artificiales, se llama TensorFlow y va por la versión 2.3

TensorFlow es el sistema de aprendizaje automático de segunda generación de Google Brain, liberado como software libre en noviembre de 2015. El nombre TensorFlow deriva de las operaciones que tales redes neuronales realizan sobre arrays multidimensionales de datos. Estos arrays multidimensionales son llamados *"tensores"*

## Unidad de procesamiento del tensor (TPU)

En mayo de 2016 Google anunció su unidad de procesamiento del tensor (TPU), una construcción ASIC personalizada específicamente para aprendizaje automático y adaptada para TensorFlow. TPU es un acelerador de IA programable diseñado para proporcionar alta eficiencia aritmética de precisión baja, y orientado a utilizar modelos más que para entrenarlos

## TensorFlow

La API de TensorFlow está disponible en varios lenguajes, nosotros usaremos la de NodeJS para el ejemplo de red neuronal artificial que construiremos con el conjunto de datos MNIST

```
// Cargamos los módulos necesarios
const tfnode = require('@tensorflow/tfjs-node-gpu'),
    fs = require('fs'),
```

```
    zlib = require('zlib');

(async () => {
// Leemos el conjunto de datos (dataset) y preparamos los datos
para la entrada
    const features = (new
Uint8Array(zlib.gunzipSync(fs.readFileSync(__dirname +
'/mnist/train-images-idx3-ubyte.gz')))).slice(16),
        labels = (new
Uint8Array(zlib.gunzipSync(fs.readFileSync(__dirname +
'/mnist/train-labels-idx1-ubyte.gz')))).slice(8),
        featuresTest = (new
Uint8Array(zlib.gunzipSync(fs.readFileSync(__dirname +
'/mnist/t10k-images-idx3-ubyte.gz')))).slice(16),
        labelsTest = (new
Uint8Array(zlib.gunzipSync(fs.readFileSync(__dirname +
'/mnist/t10k-labels-idx1-ubyte.gz')))).slice(8),
        feats = tfnode.data.array((new
Float32Array(features)).map(f => f / 255.0)).batch(784),
        lbls = tfnode.data.array(labels).batch(1),
        featTests = tfnode.data.array((new
Float32Array(featuresTest)).map(f => f / 255.0)).batch(784),
        lblTests = tfnode.data.array(labelsTest).batch(1)
// Instanciación del modelo
    const model = tfnode.sequential()
// Se añaden las capas de la red neuronal
    model.add(tfnode.layers.dense({ units: 128, activation: 'relu',
inputShape: [784] }))
    model.add(tfnode.layers.dropout(0.2))
    model.add(tfnode.layers.dense({ units: 10, activation:
'softmax' }))
// Compilación del modelo
```

```
    model.compile({ optimizer: 'adam', loss:
'sparseCategoricalCrossentropy', metrics: ['acc'] })
    model.summary()
    console.log(`Training dataset with ${feats.size} features and
${lbls.size} labels...`)
// Fase de entrenamiento del modelo
    const dataTrain = tfnode.data.zip({ xs: feats, ys: lbls
}).batch(1),
        testTrain = tfnode.data.zip({ xs: featTests, ys: lblTests
}).batch(1),
        history = await model.fitDataset(dataTrain, {
            epochs: 5,
            validationData: testTrain,
            callbacks: {
                onTrainBegin: logs => console.log('Start
Training...'),
                onTrainEnd: logs => console.log('Stop Training'),
                onEpochBegin: (epoch, logs) => console.log('Begin
epoch: ' + epoch),
                onEpochEnd: (epoch, logs) => console.log('End
epoch: ' + epoch + '\n- Loss: ' + logs.loss + '\n- Accuracy: ' +
logs.acc)
            }
        }).catch(err => console.error(err))
// Se muestran los valores de salida
    Object.keys(history.history).map(key =>
console.log(`${key}=${history.history[key]}`))
})()
```

Necesitamos instalar las dependencias del módulo *@tensorflow/tfjs-node-gpu* para que funcione, o el módulo *@tensorflow/tfjs-node* en caso de no usar GPU

En el ejemplo anterior estamos leyendo el dataset y creando una red neuronal artificial con 128 neuronas, la capa de dropout permite desactivar un porcentaje de las neuronas durante el entrenamiento en nuestro caso el 20%, esto es necesario para no tener resultados por sobreentrenamiento. Finalmente tenemos 10 unidades en la capa de salida por lo que el resultado será un tensor de 10 elementos que indica la probabilidad de que la imagen sea correcta. Hacemos 5 pasadas de entrenamiento y validamos los resultados

La salida por lo tanto muestra el resumen del modelo y el entrenamiento, dando los valores de pérdida y acierto en cada entrenamiento y al final el total

```
_________________________________________________________________
Layer (type)                 Output shape              Param #
=================================================================
dense_Dense1 (Dense)         [null,128]                100480
_________________________________________________________________
dropout_Dropout1 (Dropout)   [null,128]                0
_________________________________________________________________
dense_Dense2 (Dense)         [null,10]                 1290
=================================================================
Total params: 101770
Trainable params: 101770
Non-trainable params: 0
_________________________________________________________________
Training dataset with 60000 features and 60000 labels...
Start Training...
Epoch 1 / 5
Begin epoch: 0
eta=863.4 ==>------------------------------ acc=1.00 loss=0.102
```

Una vez entrenado el modelo podremos pasarle imágenes para detectar la coincidencia en la

probabilidad de los resultados

```
// Adaptamos las imágenes a la entrada del modelo y se crea el
tensor de entrada
fs.createReadStream('/images.json.gz').pipe(zlib.createGunzip()).on
('data', async data => {
        const imagen = Float32Array.from(await
sharp(Buffer.from(data.image, 'base64')).resize(28,
28).greyscale().raw().toBuffer().catch(err => next(err))),
            buffer = tfnode.buffer([1, 784], 'float32',
imagen.map(f => f / 255.0))
        model.predict(buffer.toTensor()).print()
    })
        .on('end', () => {
            console.log('All images predicted.')
        })

Tensor
     [[0, 0, 0, 0, 0, 0.0427372, 0, 0.9571278, 0, 0.0001349],]
Tensor
     [[0.0002897, 0, 0.209211, 0.0000031, 0, 0.0015369, 0.7871032,
0.001465, 0.0003912, 0],]
Tensor
     [[0.0000626, 0, 0.4744948, 0.0397445, 0, 0.0002614, 0,
0.4854368, 0, 0],]
Tensor
     [[0.0000023, 0, 0.0093701, 0.1584364, 0, 0.3968244, 0.0428764,
0.3924459, 0.0000444, 2e-7],]
Tensor
     [[0.0000071, 0, 0.0316687, 0.0000178, 0, 0.0000988, 0.9680935,
0.0001098, 0.0000045, 0],]
```

```
...
```

Esperamos que os haya gustado, nos leemos en la próxima haciendo redes neuronales de convolución 