# Reporte 3: Clasificador de letras

Efren Lopez, CIDETEC-IPN

## I. Introducción

Las redes neuronales o artificiales son sistemas informáticos inspirados en las redes neuronales biológicas. Estos sistemas aprenden para realizar tareas, considerando ejemplos, generalmente sin programación especificas de tareas. En reconocimiento de imágenes, se puede aprender a identificar imágenes que puedan contener algún objeto, por ejemplo, si la imagen contiene gatos, etiquetadas como "gato" y "no gato". El objetivo inicial de las redes neuronales era resolver los problemas de la misma manera que lo haría el cerebro humano. Las redes neuronales se han utilizado en variedad de tareas, incluyendo visión por computadora, reconocimiento de voz, traducción automática, filtrado de redes sociales, juegos de mesa y videojuegos, diagnósticos médicos y entre muchos otros.

### A. Componentes de una red neuronal

**Neuronas**

Una neurona con etiqueta *j* recibiendo una entrada $p_j(t)$ de las neuronas procesadas consta de los siguientes componentes:

- Una activación $a_j(t)$, dependiendo de un parámetro de tiempo discreto
- Posiblemente un umbral $\theta_t$, que permanece fijo a menos que haya cambiado por una función de aprendizaje
- Una función de activación **f** que calcula la activación en un momento dado $t+1$ de $a_j(t), \theta_j$ y la entrada neta $p_j(t)$ dando lugar a la relación $a_j(t+1) = f(a_j(t), p_j(t), \theta_t)$,
- Y una función de salida $f_{out}$ calcular la salida de activación $o_j(t) = f_{out}a_j(t)$

Una neurona de entrada no tiene predecesor, si no que sirve como interfaz de entrada para toda la red.

**Conexiones y pesos**

La red consiste en conexiones, cada una de las cuales transfiere la salida de una neurona $i$ a la entrada de una neurona $j$. En este sentido a cada conexión se le asigna un peso $w_{ij}$.

**Función de propagación** La función de propagación calcula la entrada $p_j(t)$ a la neurona $j$ de las salidas $o_i(t)$ de las neuronas predecesoras y por lo general tiene la forma:

$$p_j(t) = \sum_i = o_i(t)w_{ji}$$

**Aprendizaje supervisado** El aprendizaje supervisado un conjunto de pares de ejemplos $(x, y), x \in X, y \in Y$ y el objetivo es encontrar una función $f : X- > Y$ en la clase permitida de funciones que coinciden con los ejemplos. En otras palabras, deseamos inferir el mapeo implícitos por los datos.

jmzelectronica@gmail.com

Un costo comúnmente utilizado es el error cuadrado medio, que intenta minimizar el error cuadrático medio entre la salida de la red, $f(x)$ y el valor objetivo $y$ sobre todos los pares de ejemplo.

**Aprendizaje no supervisado** Es un método de aprendizaje automático donde un modelo es ajustado a las observaciones. Se distingue del aprendizaje supervisado por no tener un conocimiento a priori. El aprendizaje puede ser usado en conjunto con la inferencia bayesiana para producir probabilidades condiciones.

El aprendizaje no supervisado también es útil para la compresión de datos: fundamentalmente, todos los algoritmos de compresión dependen tanto explícita como implícitamente de una distribución de probabilidad sobre un conjunto de entrada.

Otra forma de aprendizaje no supervisado es la agrupación (en inglés, clustering), el cual a veces no es probabilista.

## II. Desarrollo

Para este trabajo usamos jupyter notebook en donde se encuentran los diferentes ejercicios para resolver, en primera instancia los datos que usaremos consiste en imágenes de letras de la $a$ a la letra $j$ en diferentes fondos. La meta final

Fig. 1. Datos de entrada para la red neuronal

de estos ejercicios es alcanzar una precisión de al menos 80%

- Para modificar y ejecutar los programas de los diferentes ejercicios se requiere importar algunos paquetes en python, entre ellos tensorflow; al ejecutar y compilar ese código, nos imprimirá en pantalla lo siguiente, si fue exitoso *"All modules imported."*.
- En la segunda parte se descarga el dataset notmnist, contiene 500,000 imágenes de entrenamiento, solo usamos un subconjunto de 15,000 datos, si la compilación fue exitosa nos imprime un mensaje como el siguiente: $Downloading not MNIST\_train.zip...$ $Download Finished$ $Downloading not MNIST\_test.zip...$ $Download Finished$

$All files downloaded.$

- Una vez descargado el dataset se pasa a descomprimir los archivos comprimidos y si la compilación del programa fue exitoso nos imprime un mensaje de: *All features and labels uncompressed.*

### A. Problema 1

Para el problema uno se pide normalizar las características de sus datos de entrenamiento y prueba. Para ello se pide implementar la escala Min-Max en el $normalize()$ otorgándonos los siguientes valores para la ecuación:

$$X' = a + \frac{(X - X_{min}) * (b - a)}{X_{max} - X_{min}}$$

En donde:
$a = 0.1$
$b = 0.9$
$x_{min} = 0$
$x_{max} = 255$
Aplicando la formula de la escala Min-Max, queda lo siguiente:

```
def normalize_grayscale(image_data):
"""
Normalize the image data with Min-Max
 scaling to a range of [0.1, 0.9]:
 param image_data: The image data
 to be normalized:
 return: Normalized image data
"""
a = 0.1
b = 0.9
x_min = 0
x_max = 255

xprima = a + (((image_data-x_min)*(b-a))/
(x_max-x_min))
return xprima
```

Si no hay errores de sintaxis y la solución es correcta después de la compilación nos imprime el siguiente mensaje: *Tests Passed!*

### B. Problema 2

Para este problema se requiere entrenar la red neuronal con los datos y necesitamos tensores $float32$.
Para ello acudimos a la pagina de ayuda de Tensorflow, en donde nos dice que cada elemento en el tensor tiene el mismo tiempo de dato, algunos tipos de tensores son especiales, algunos de los mas importantes son:

- tf.Variable
- tf.Constant
- tf.Placeholder
- tf.Sparsetensor

Con la excepción de la variable tf.Variable el valor de un tensor es inmutable, es decir que en el contexto de una sola ejecución solo tienen un solo valor, sin embargo, evaluando el mismo tensor dos veces, nos puede retornar diferente valores, por ejemplo que el tensor puede estar leyendo datos del disco, o generando números aleatorios.
Para el **features** y **labels** usamos el tensor **tf.holder**, el cual se requiere de los siguientes argumentos.

- **dtype**: El tipo de elementos en el tensor tiene que ser alimentado,$float32, int32$
- **shape**: La forma de el tensor tiene que ser alimentado (opcional)
- **name**: Un nombre por la operación (opcional)

Dicho tensor nos regresa un tensor que puede ser utilizado como ”handle” para alimentar un valor, pero no evaluado directamente.
Para encontrar los pesos(weight) y el sesgo(bias), requerimos usar el tensor **tf.truncated_normal**, el cual nos dice que emite valores aleatorios de una distribución normal truncada.
Los valores generados siguen una distribución normal con un valor especifico y una desviacion estandard, excepto que los valores cuya magnitud es más de 2 desviaciones estándar respecto a la media se descartan y se vuelven a seleccionar.
**Argumentos**

- **Shape**: Un tensor entero de 1-D o un arreglo de Python. El ”shape” de la salida del tensor
- **mean**:Un tensor de 0-D o un valor de python tipo **dtype, (float32, int32)**
- **dtype**: El tipo de la salida
- **seed**: Un entero en Python.
- **name**: un nombre para la operación(opcional)

Sabiendo esto se realizó la siguiente solución:

```
features_count = 784
labels_count = 10

# TODO: Set the features and labels tensors
features = tf.placeholder(tf.float32)
labels = tf.placeholder(tf.float32)

# TODO: Set the weights and biases tensors
weights = tf.Variable(tf.truncated_normal
((features_count, labels_count)))
biases = tf.Variable(tf.zeros(labels_count
```

### C. Problema 3

Para el problema 3 necesitamos configurar 3 parámetros para entrenar la red neuronal. En donde cada configuracion, uno de los parametros tiene diferentes opciones, podemos elegir el que de la mejor precision, para mi caso, probé todas las configuraciones y no me dio muy buena presicion, asi que tuve que combinar algunas configuraciones, dando como resultado los siguientes valores:

```
# TODO: Find the best parameters for each
    configuration
epochs = 5
batch_size = 100
learning_rate = 0.5
```

Y como mejor precisión fue la siguiente: Y para el test dió el siguiente accuracy.

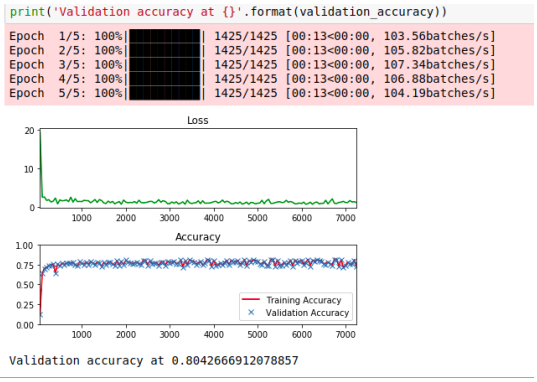


Fig. 2. Alcance de accuracy con parametros de configuración

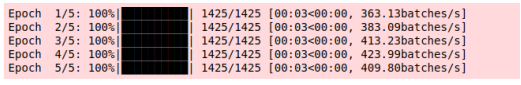


Fig. 3. Accuracy test

## III. Conclusión

En estos ejercicios se observa el proceso de entrenamiento de una red neuronal, y determinar si la red construida puede aprender las características de cada una de las imagenes.

## Referencias

www.TensorFlow.com/api_docs/python/tf/
Deep Learning-Udacity