Introducción a redes neuronales con TensorFlow

Pedro Bueno Aldrey
`pedro.bueno@rai.usc.es`

Aula Profesional USC

12 de Febrero 2016

Índice

Elementos necesarios

- **La tarea** T: transformación de un ejemplo: $x \epsilon \mathbb{R}^n$

Ejemplo: Función de clasificación en k categorías

$$f : \mathbb{R}^n \to \{1, \ldots, k\} \tag{1}$$

- **Experiencia** E: Datos de los que aprende la red.

Ejemplo: Experiencia en clasificación de imágenes

Conjunto de pares de vectores $v \epsilon \mathbb{R}^n$ y categorías $k \epsilon \{1, \ldots, k\}$
A cada vector v_i le corresponde una categoría k_i

- **Medida de rendimiento** P: comparamos los valores **procesados** con valores que deberíamos obtener. Obteniendo un ratio de ***precisión***.

Ejemplo aprendizaje supervisado

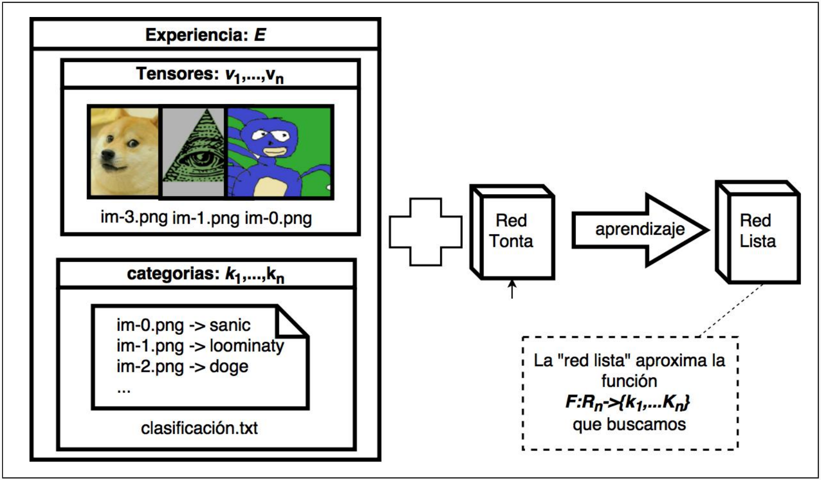

Figura: Fase de Aprendizaje

Ejemplo aprendizaje supervisado

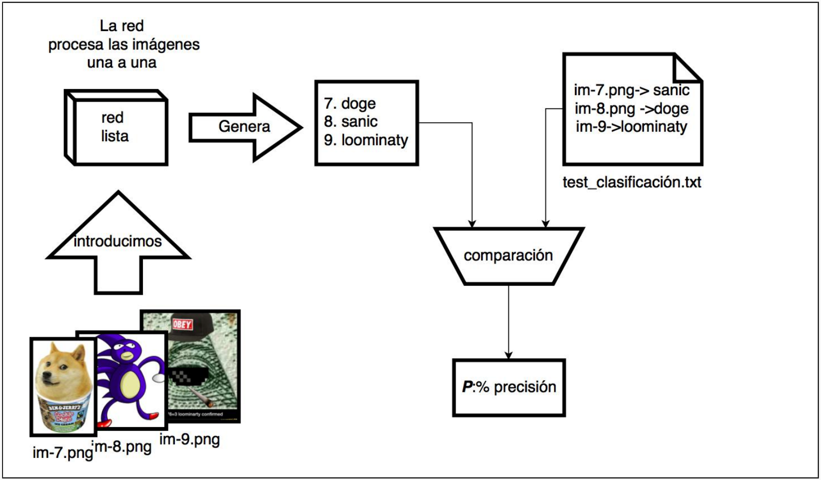

Figura: Fase de Testeo

Modelo simplificado de una neurona

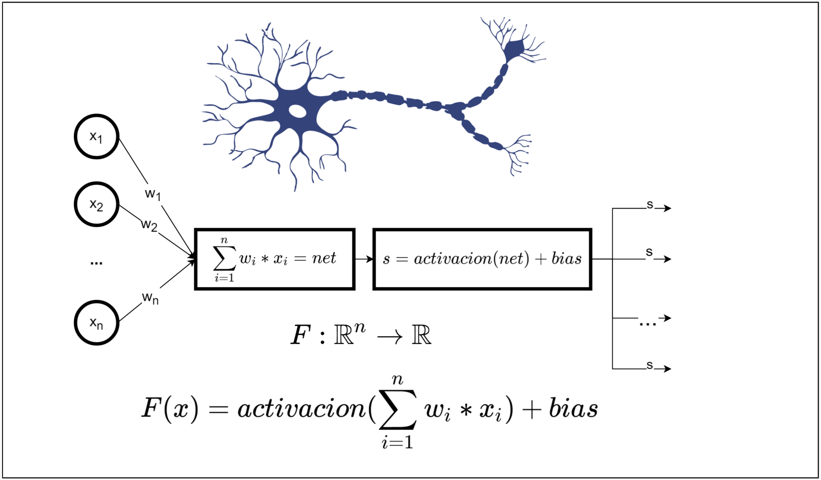

Figura: Multiplicamos las entradas por los pesos, los sumamos y los pasamos por la función de activación

Ejemplo de Perceptron de 2 capas

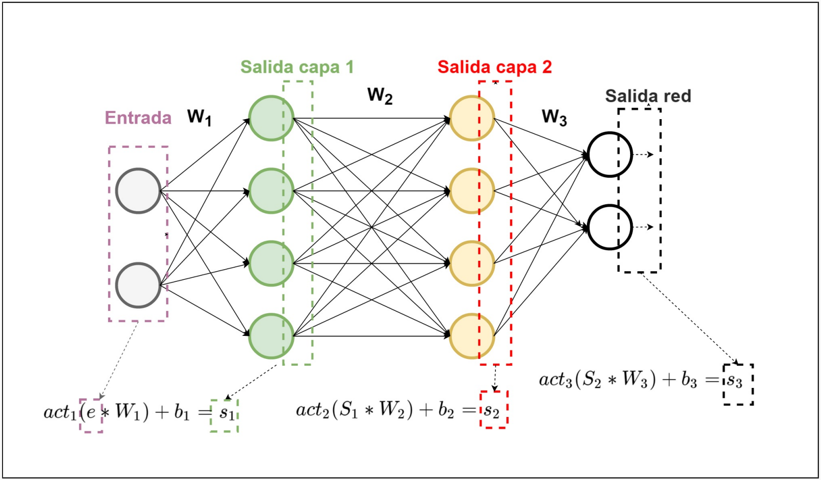

Figura: Realizamos las operaciones en cadena

Propagación hacia atrás

Es el algoritmo mas común utilizado para el aprendizaje supervisado, consiste en lo siguiente:

- Pasamos las entradas por la red, obteniendo una salidas reales.
- Generamos una **función de error** a partir de la diferencia entre la salidas reales y las salidas esperadas.
  En ella los **pesos** son las **variables**.
- **Minimizamos** la función de error, podemos utilizar técnicas como el descenso por gradiente.

Definición de la función de error

Dada la siguiente nomenclatura:

- y: función de la red
- t: función de salida esperada
- E: función de error

Ejemplo de función de error sobre una entrada

$$E(y,t) = \frac{1}{2}(t-y)^2 \tag{2}$$

Ejemplo de función de error de n entradas

$$E_t = \frac{1}{n} * \sum_x E_x \tag{3}$$

Calculo del Gradiente del error

Calculamos el gradiente de la función de error. Para ello utilizamos:

Función de activación (debe ser diferenciable):

$$\varphi(z) = \frac{1}{1+e^{-z}} \tag{4}$$

Derivada de la función de activación

$$\frac{d\varphi}{dz}(z) = \varphi(z)(1-\varphi(z)) \tag{5}$$

Salida de una neurona j:

$$o_j = \varphi(\text{net}_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right) \tag{6}$$

Cálculo del Gradiente del error

Aplicamos la regla de la cadena para obtener la derivada parcial de un peso w_{ij}:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net_j}} \frac{\partial \mathrm{net_j}}{\partial w_{ij}} \tag{7}$$

$$\frac{\partial \mathrm{net_j}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} o_k \right) = o_i \tag{8}$$

$$\frac{\partial o_j}{\partial \mathrm{net_j}} = \frac{\partial}{\partial \mathrm{net_j}} \varphi(\mathrm{net_j}) = \varphi(\mathrm{net_j})(1 - \varphi(\mathrm{net_j})) \tag{9}$$

Si es la última capa

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2} (t - y)^2 = y - t \tag{10}$$

Cálculo del Gradiente del error

Si es una capa oculta consideramos la función de error con respecto a las entradas de la siguiente capa

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\mathrm{net}_u, \mathrm{net}_v, \ldots, \mathrm{net}_w)}{\partial o_j} \tag{11}$$

$$\frac{\partial E}{\partial o_j} = \sum_{l \in L} \left(\frac{\partial E}{\partial \mathrm{net}_l} \frac{\partial \mathrm{net}_l}{\partial o_j} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial \mathrm{net}_l} w_{jl} \right) \tag{12}$$

Derivada parcial con respecto del peso w_{ij}:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net_j}} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{si es capa de salida} \\ (\sum_{l \in L} \delta_l w_{jl}) o_j (1 - o_j) & \text{si es capa interna} \end{cases} \tag{13}$$

Aplicación de descenso de gradiente

Recordemos que el **gradiente** apunta a la dirección de **máximo crecimiento** de la función.
Debemos restarle al **vector de pesos** el vector gradiente en el

punto actual multiplicado por una **constante** α

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} = \begin{cases} -\alpha o_i (o_j - t_j) o_j (1 - o_j) & \text{Capa interna} \\ -\alpha o_i (\sum_{l \in L} \delta_l w_{jl}) o_j (1 - o_j) & \text{Capa de salida} \end{cases} \tag{14}$$

Elementos básicos de TensorFlow

- TensorFlow computa el modelo de red creando un **grafo** que se ejecuta **fuera de python**, aumentando la eficiencia.
- Los grafos se ejecutan dentro de sesiones
- Los datos se representan mediante **Tensores**, que pueden ser:
  - Variables
  - Constantes
  - Placeholders
- Aplicando operaciones sobre las variables definimos el modelo

Ejemplo del Tutorial

- Entrada: las imágenes 24x24 se pasan como un vector de tamaño 784, los introducimos en lotes de 55000
- Salida: Sacamos las categorías como un vector, donde cada componente representa una categoría

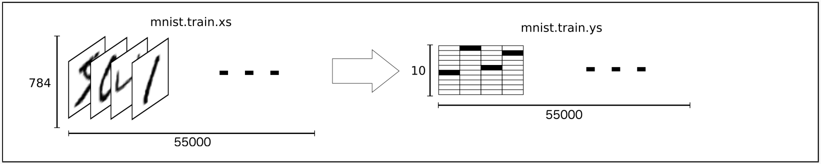

Figura: Entrada y salida de la red del tutorial

Tensores

- Declaración de variables

```
a = tf.Variable(tf.zeros([784, 10]))
```

- Declaración de constantes

```
shape=[32]
a = tf.constant(0.1, shape=shape)
```

- Declaración de placeholders

```
a = tf.placeholder(tf.float32, [None, 10])
```

Operaciones

- Construcción del modelo

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

- Construcción de la función de error

```
V=tf.nn.softmax_cross_entropy_with_logits(y_,y)
cross_entropy=tf.reduce_mean(V)
```

- Aplicación del algoritmo de optimización

```
train_step = tf.train
                .GradientDescentOptimizer(0.5)
                .minimize(cross_entropy)
```

Ejecutar el modelo

- Inicialización de la sesión

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

- Iteración de aprendizaje

```
sess.run(
    train_step,
    feed_dict={x: batch_xs, y_: batch_ys}
    )
```

Prueba del Modelo

- Convertimos el vector resultado en una categoría

```
cy=tf.argmax(y, 1)
cy_=tf.argmax(y_, 1)
```

- Comparamos las categorías dadas con las esperadas

```
cPred = tf.cast(tf.equal(cy,cy_), tf.float32)
accuracy = tf.reduce_mean(cPred)
```

- Probamos el modelo con los datos de ejemplo

```
print(sess.run(
        accuracy,
        feed_dict={
            x: mnist.test.images,
            y_: mnist.test.labels
        }))
```

Ejercicios

- **Ejercicio 1**: Completar con éxito el Tutorial para novatos de la página de TensorFlow `www.tensorflow.org/tutorials/mnist/beginners/`
- **Ejercicio 2:** Transformar el Ejercicio 1 en una red multicapa.
- **Ejercicio 3:** ¿Quién puede conseguir el valor más alto de precisión?

Redes convolucionales

Introducción :

- Asumimos que vamos a operar sobre imágenes, por lo tanto su **disposición espacial** nos aporta información.
- La entrada deja de ser un vector y pasa a ser un **Tensor de rango 3**. Por ejemplo una imagen .jpg sería de dimensión **[32,32,3]**
- Las sucesivas capas continúan teniendo como salida un Tensor de rango 3; a excepción de la ultima capa, que es de clasificación normal.
- Se suelen aplicar **diferentes canales** de capas CONV. Cada una se puede ver como un **filtro**.

Ejemplo 1

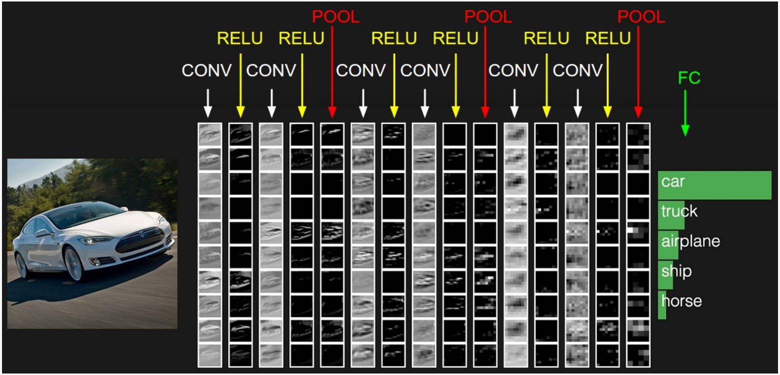

Figura: Utilizada en una imagen de un coche

Ejemplo 2

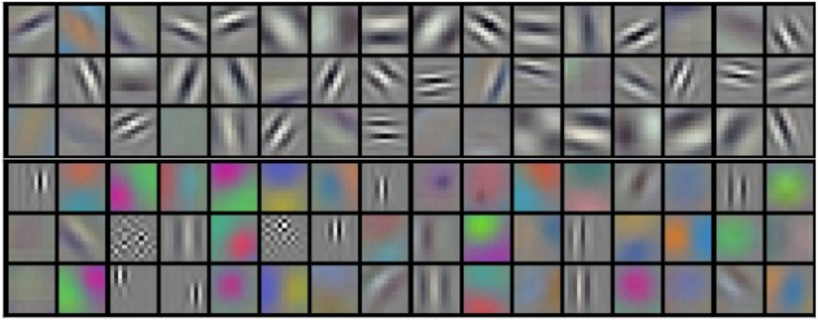

Figura: Filtros aplicados

Ejemplo 3

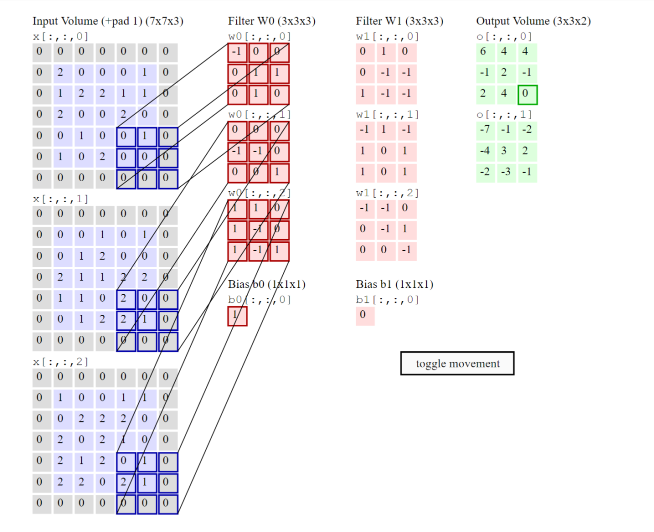

Figura: Capa CONV con matrices

Variables capa CONV

- Dimensión de entrada W: [ancho, alto, profundidad]
- Dimensión de salida: [ancho, alto, profundidad, canales]
- Zero-padding P: número de ceros que se ponen alrededor de la imagen
- Stride S: determina el salto que el filtro hace en cada desplazamiento
- Field F: rango que percibe la capa de cada vez
- Número de pesos de la red N: viene determinado por:

$$N = (W - F + 2P)/S + 1 \tag{15}$$

Completando el modelo

Ampliamos el modelo con dos capas fijas(que no aprenden)

- RELU: Ejecuta sobre cada elemento la función $max(0, x)$ a cada elemento
- POOL: Realiza un **rescalado** utilizando la operación $max()$

De esta forma podríamos tener una arquitectura como la siguiente para un conjunto de imágenes [32,32,3] a clasificar en 10 categorías con correspondientes **salidas**:

1. INPUT [32,32,3]
2. CONV: [32,32,12]
3. RELU: [32,32,12]
4. POOL: [16,16,12]
5. FC: [1,1,10], capa de clasificación “fully-connected”

Implementación en TensorFlow

- Capa CONV:

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, data_format=None, name=None)
```

- Capa RELU:

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, data_format=None, name=None)
```

- Capa POOL:

```
tf.nn.relu(features, name=None)
```

Proyecto final

Consiste en la realización de una red de clasificación de imágenes, tomando como modelo la del tutorial para expertos:
`https://www.tensorflow.org/tutorials/mnist/pros/`
Se deben utilizar imágenes que no estén en los ejemplos de

TensorFlow, las instrucciones para cargar archivos se encuentran en:
`https://www.tensorflow.org/how_tos/reading_data/`
Ejemplo del proyecto a realizar (se pueden utilizar partes del

mismo)
`https://github.com/petrusboniatus/cursoRN`

Bibliografía

Libro: Deep Learning

- Fecha de publicación original: 2016
- Autores: Yoshua Bengio, Ian Goodfellow

Backpropagation:

- `https://en.wikipedia.org/wiki/Backpropagation`

Convultional networks:

- `http://cs231n.github.io/convolutional-networks/`

TensorFlow

- `https://www.tensorflow.org/`