# Objetivos

Si tengo una imagen que tiene texto adentro, ¿cómo tengo que hacer para que eso se convierta en un archivo de texto? Básicamente queremos crear un Mini OCR.

# ¿Qué es una Red Neuronal Artificial (RNA)?

Es un conjunto de Pedacitos chiquitos de programas que computan una función, donde cada uno de ellos está relacionado con el resto.

Una RNA trata de imitar al cerebro humano, en el sentido que todo el conocimiento que la red va a adquirir está en las conexiones entre los distintos pedacitos de cómputo que serían las neuronas.

Nosotros tenemos la capacidad de adquirir conocimiento cuando se refuerzan algunas o se debilitan otras conexiones entre neuronas.



* Dendritas = entradas a la neurona
* Núcleo = condensa energía, en algún momento se dispara por el axón la energía que tenía acumulada = es la sumatoria
* La sinapsis del cerebro humano se relaciona con los pesos sinápticos que voy a tener en una neurona artificial

# Modelo matemático de una neurona tipo Perceptrón

Tengo un montón de números que entran, que se multiplican por otro montón de números que ya tenemos (pesos sinápticos). Esa multiplicación se va a ir sumando. Opcionalmente se le va a sumar el BIAS, que es otro número más. Y todo eso me va a dar un valor que es la sumatoria que está concentrada dentro de la neurona. Este va a pasar por una función de activación, y luego se va a propagar dentro de la red.



# La Red Neuronal MLP (Multi Layer Perceptron)

Una neurona dentro de una red neuronal es cada uno de esos círculos rojos que tenemos dibujados.



Las entradas de las neuronas del medio son ya salidas de las neuronas anteriores.

La última neurona da la salida al final de la red.

El enfoque más práctico para implementar todo esto es usar notación matricial.



Tengo una matriz de pesos sinápticos.

El vector de la derecha son las entradas que tenemos.

La salida se calcula como el producto de las entradas por los pesos sinápticos.

Una vez listo, todo esto tiene que pasar por la función de activación.

# Entrenamiento de la Red MLP

Dijimos que la red neuronal va a aprender. ¿Cómo hace?

Si el conocimiento de las neuronas está en los pesos sinápticos, para que la neurona aprenda lo que tenemos que hacer es cambiar esos pesos.

Hay **dos tipos de entrenamiento: supervisado y no supervisado**.

En este caso vamos a tener un entrenamiento supervisado: tenemos una serie de patrones de datos de entrada asociados a la salida que esperamos obtener.

Una vez que ingresamos un patrón de entrada y obtenemos una salida distinta a la que esperamos obtener, tenemos que modificar el peso sináptico para que se comporte de otra forma… y repetir hasta lograr que se comporte como yo quiero.

El entrenamiento no supervisado, en cambio, hace referencia a no tener una retroalimentación de la salida.

**Entrenamiento Batch:** como tenemos varios patrones de entrenamiento, todos los patrones de entrenamiento se van a correr y se van a ir acumulando todos los errores para que, al final, cuando termine de procesar todos los patrones, calculemos un error promedio, y recién ahí corrijamos los pesos.

Este procedimiento se repite todas las veces que sea necesario, hasta que funcione como yo quiero.

El algoritmo que vamos a usar para entrenar esta red MLP se llama **Back Propagation**, “descenso por el gradiente del error”. El gradiente nos da la dirección de máximo crecimiento del error.

Calculo la salida, y como conozco la salida que esperaba obtener, puedo calcular el error. Este error se retropropaga, afectando solamente los pesos que van a hacer que yo pueda achicar mi error.

El gradiente me da la dirección máxima en la que va a crecer el error. Como nosotros no queremos que crezca el error, queremos que disminuya, vamos a ir para el otro lado (por eso el signo menos).

**Modelo de aprendizaje:**



El parámetro es la **tasa de aprendizaje**, un número entre cero y uno que me dice qué tanto tengo que aprender en este momento.

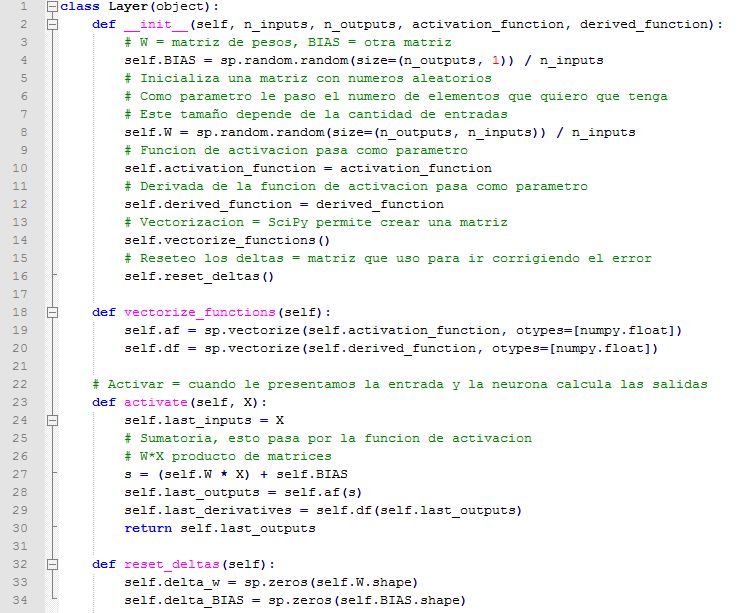
# Implementación de la Red Neuronal en Python

La red multicapa está modelada mediante dos clases, una para cada capa y otra que comprende toda la red. A su vez hay una tercera clase que se encarga del entrenamiento de la red. Estas clases están definidas en el archivo **<mlp.py>**

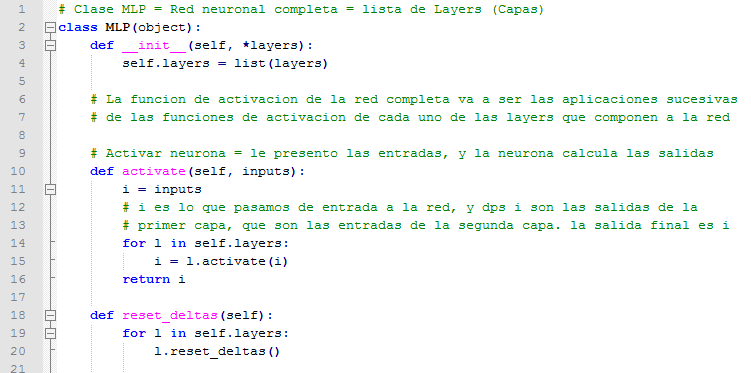
Las clases son:

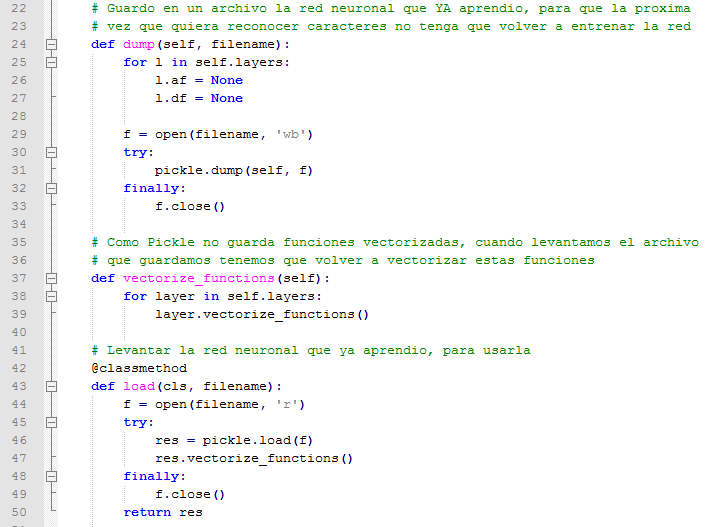
* **class Layer**, modela cada capa
* **class MLP**, modela la red neuronal MLP
* **class BackPropagationTrainer**, modela el algoritmo de entrenamiento

**Clase Layer:**

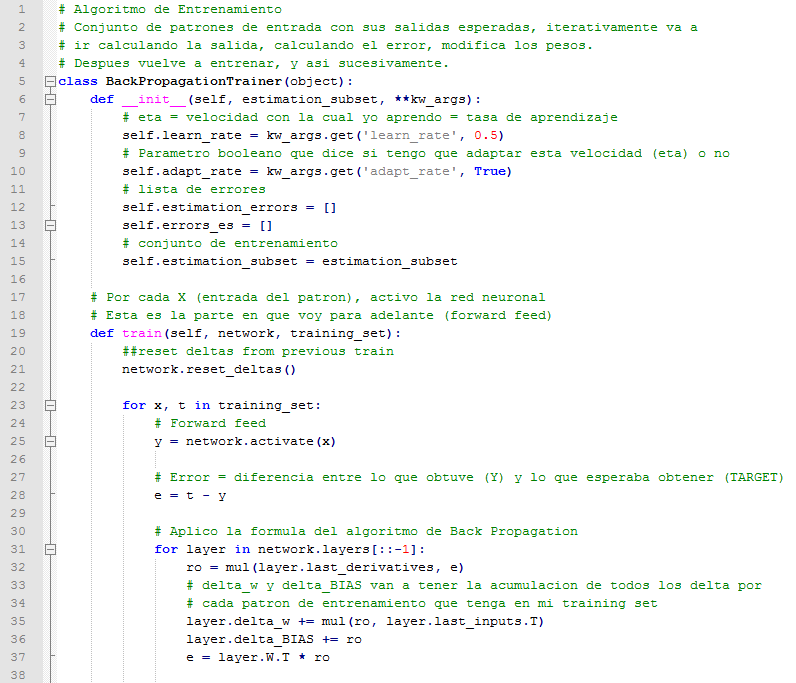


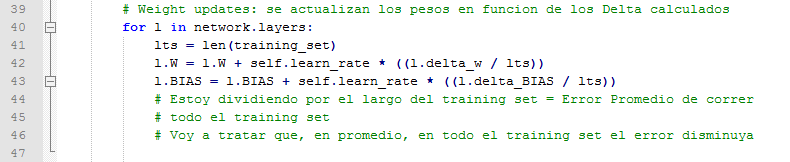
**Clase MLP:**



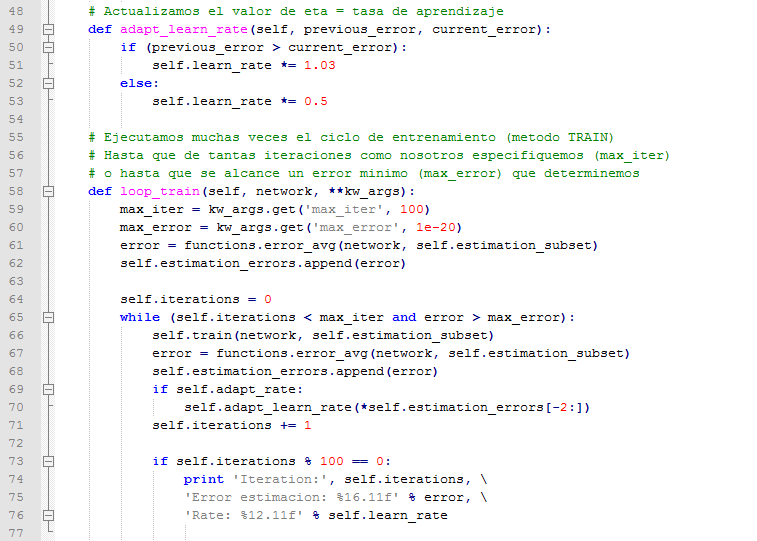


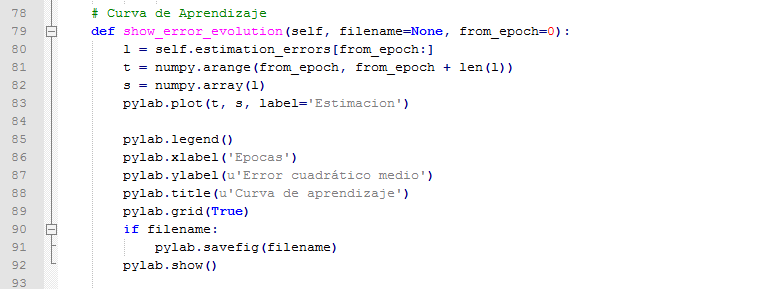
**Clase BackPropagationTrainer:**





**Clase BackPropagationTrainer (continuación):**

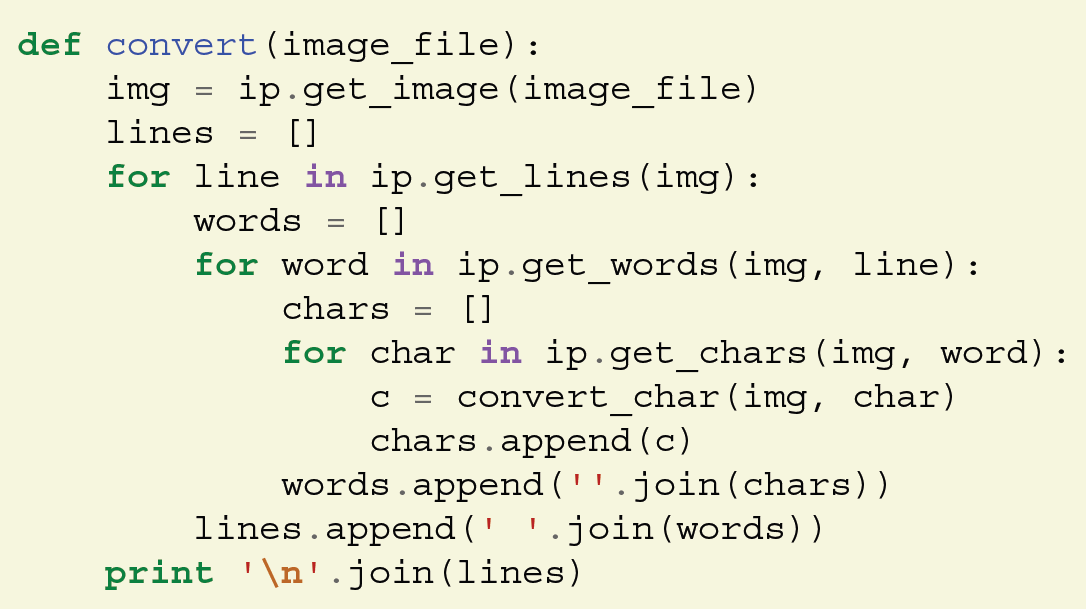




# ¿Cómo usamos todas estas clases para reconocer caracteres?

La idea es entrenar una red neuronal para que dada una imagen de un solo carácter, nos devuelva ese carácter.

La estructura de nuestro OCR sería la siguiente: (archivo **<ocr.py>**)



En este código ya estamos en condiciones de identificar el papel de la red neuronal:



En sí el código es bastante simple :)

Obviamente el problema es cómo implementamos **get\_image()**, **get\_line()**, etc...

# Buscando líneas, palabras y caracteres

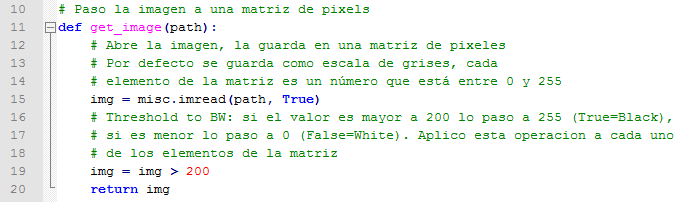
Los pasos serían:

1. Abrir la imagen y convertir cada pixel a blanco y negro
2. Filtrar un poco la imagen para eliminar ruidos (si la calidad no es del todo buena y falta algún que otro pixel, consideremos como si estuviera ahí)
3. Enumerar los grupos de pixeles que están juntos para tratar de identificar porciones de la imagen que contengan “algo” (una línea, una palabra, un carácter)
4. Buscar los slices que encierran a estos grupos, es decir, desde que posición de la imagen hasta que otra posición de la imagen está ese objeto que acabo de enumerar

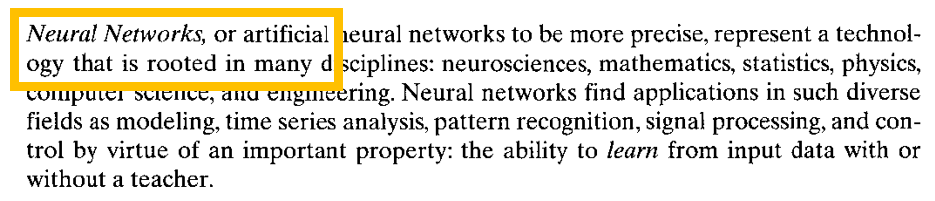
Para esto vamos a usar **ndimage**, un paquete de SciPy.

A partir de ahora empezamos a trabajar en el archivo **<image\_processing.py>**

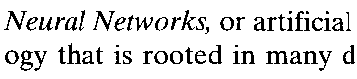
**Método get\_image():**



El ejemplo que vamos a usar es una porción de un párrafo escaneado del libro “Neural Networks, A Comprehensive Foundation” de Simon Haykin, 1999:

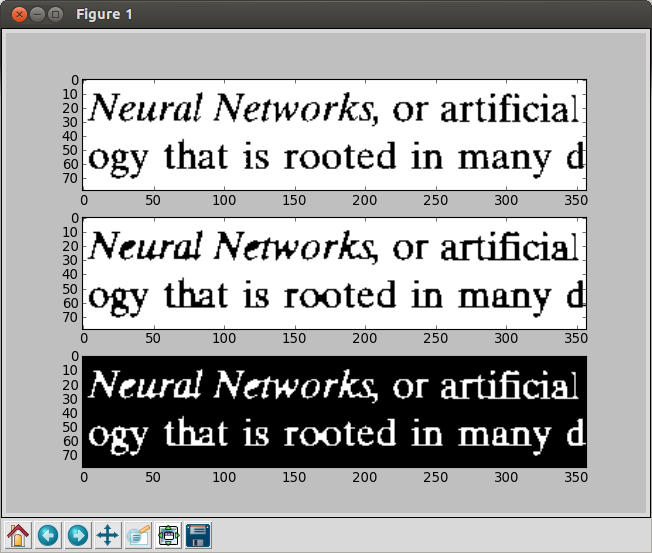


Vamos a trabajar con la siguiente imagen:

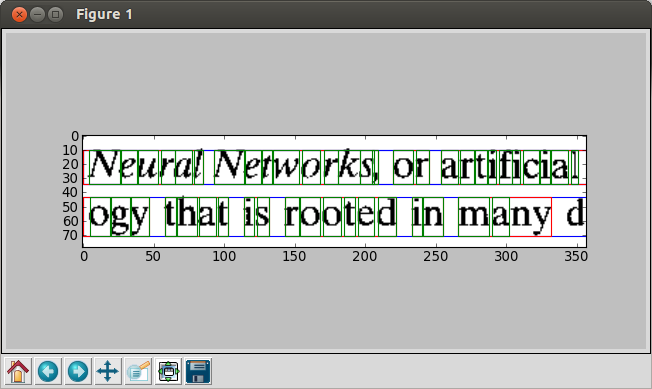


Vamos a aplicar dos filtros: el primero para

Este es el resultado de aplicar los **filtros** a la imagen:



Esto corresponde a los pasos 3 y 4:



# El Modelo Elegido

En el objetivo dijimos que la entrada era una imagen y la salida un carácter. Pero en realidad tanto la entrada como la salida son números. Entonces... **¿Cuáles son las entradas y cuáles las salidas?**

* **Entradas de la red:** ¿Pixeles o características? Características
* **Salidas de la red:** 8 binarias. Vamos a obtener el código ASCII del carácter identificado.

Otros aspectos del modelo que hay que determinar:

* **Cantidad de neuronas en la capa oculta:** proporcional a la cantidad de caracteres distintos a reconocer.
* **Funciones de activación:** Tangente hiperbólica en la capa oculta, Linear en la capa de salida.

Se eligió esta configuración porque de esta forma la red funciona, aunque tranquilamente se podría encontrar otra configuración distinta que también funcione.

# Paquetes y Dependencias

Para que todo esto funcione, debemos tener instalado Python >= 2.7, SciPy, NumPy, MatPlotLib y PIL (Python Image Library).

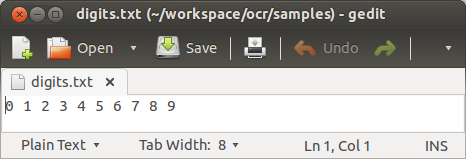
Desde Ubuntu o cualquier Debian-based distro podemos instalarlo así:

**sudo apt-get install python-numpy python-scipy python-matplotlib python-imaging**

# Definiendo Conjuntos de Entrenamiento

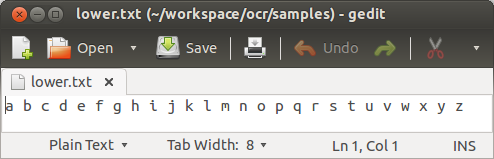
Vamos a armar dos conjuntos de entrenamiento, uno con letras y otro con números. Para eso tenemos una imagen con los números del 0 al 9, uno al lado del otro, y un archivo de texto que contiene la misma información que la imagen, sólo que en modo texto.



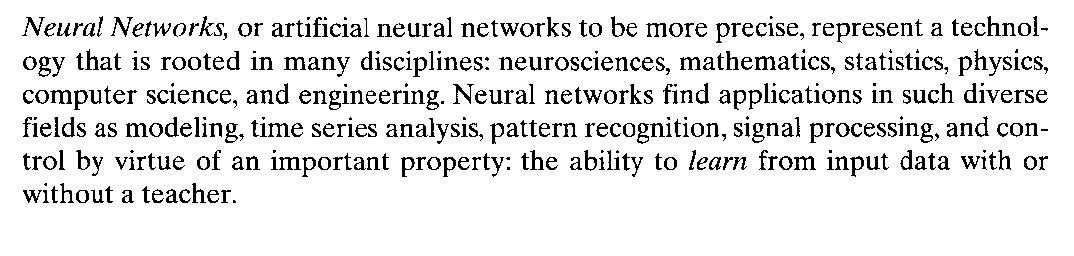


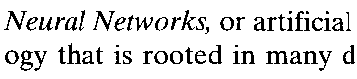
Con las letras (minúsculas) pasa lo mismo:





# Ejemplo de Funcionamiento





# Bibliografía

Este documento es una adaptación de la charla **“Neuronas Pythonicas que Reconocen Caracteres”** dada por **Ariel Rossanigo** (PyAr/UCSE) en el PyDay Rafaela 2012 (<http://www.pyday.com.ar/rafaela2012>) el sábado 20 de octubre de 2012.

* **Video:** <http://blip.tv/pyday-rafaela/neuronas-6432642>
* **Slides:** <http://goo.gl/cJA7d>
* **Código:** <https://bitbucket.org/arielrossanigo/neuronaspythonicas_rafaela2012>

--

Otro ejemplo interesante de redes neuronales: <https://www.youtube.com/watch?v=0Str0Rdkxxo>