Anly 590: Lecture 1 - Loading Data

Joshuah Touyz

August 22, 2018

1 Loading Data into Python

- In what follows a review of how to load data in Python is provided
- In particular the focus will be on:
    1. Loading data manually
        - MNIST pickle
        - Loading data in folders
    2. Loading data through Keras
    3. Supplementary online sources for data

1. Outline for loading data manually
    1. How to load the MNIST (hand-written digit) dataset
    2. Loading data from a folder
    3. Memory footprint
    4. Generating data (HW1)
2. Loading data through Keras
    - Common datasets
        - MNIST
        - Fashion MNIST
        - IMDB
        - CIFAR 10
        - CIFAR 100
        - Reuters newswire articles

1.1 Transforming our data

- Data needs to be transformed so that a computer can understand it
- To do this it is stored as a matrix or tensor format
- It can then be processed for consumption by machine learning algorithms

1.2 Memory management

- In deep learning a large number of samples is required to train models
- Therefore when reading in data, memory management is important as the cost for storing the data, such as couple thousand images, can be very high

- While this does not pose a significant limitation for much of the data we'll interact with during class, production scale systems require hardware considerations
- Note: while the datasets here are considered small, they still occupy several hundred mbs or a couple of gigs on your hard drive

2 Loading data manually

2.1 MNIST Dataset

- The "hello world" of machine learning is the MNIST dataset
- It is a data set of of handwritten digits with 60,000 training and 10,000 test samples
- It's use as a benchmark for various machine learnings algorithms
- In what next follows:
  - We show you how to load and visualize this data set in python
  - The dataset can be downloaded directly from http://yann.lecun.com/exdb/mnist/, which also contains accuracy scores for different models
    * It is zipped in a binary file format which requires reshaping the images
    * See the gist here
  - Instead of using the binary filed, the pickled data set will be used, it can be download here
  - You will want to download this data set and store it locally.
    * This process will only be repeated once

2.1.1 Data Description

- The data is loaded into training, validation and test sets
- The sets are returned as tuple pairs
- The first element of the tuple is an $\{n_i \times 784; i = \text{training,validation,test}\}$ `ndarray` of values between 0 and 1 representing pixel intensity of the characters
- Each row contains 784 columns which is in reality an 28×28 image that has been flattened
- The second element of the tuple contains is an $\{n_i \times 1; i = \text{training,validation,test}\}$

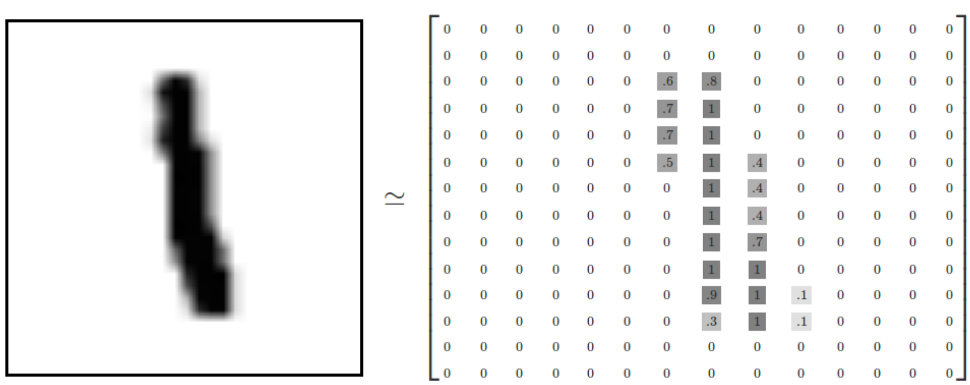
```
In [33]: # ---- Loading libraries ----
         # - Base libraries -
         import os, gzip, numpy, cPickle
         from timeit import default_timer as timer

         # - Plotting libraries -
         import matplotlib.pylab as plt
         import matplotlib.image as mpimg
         import cv2

         # - Stats libraries -
         import numpy as np
         from scipy import stats
```

```
In [11]: # ---- Load the dataset ----
         base_dir = os.getcwd()
         path_to_file = os.path.join(base_dir,'mnist.pkl.gz')
         f = gzip.open(path_to_file, 'rb')
         train_set, valid_set, test_set = cPickle.load(f)
         f.close()
```

```
In [12]: # ----- Data values ----
         len(train_set)
         print '''
         The total number of training examples is: {training_length}.
         The total number of validation examples is: {val_length}.
         The total number of test examples is: {test_length}.
         '''.format(training_length = train_set[0].shape,
                    val_length = valid_set[0].shape,
                    test_length = test_set[0].shape)
         print 'The first 10 classes of the training set are: {}'.format(train_set[1][0:10])
```

```
The total number of training examples is: (50000, 784).
The total number of validation examples is: (10000, 784).
The total number of test examples is: (10000, 784).

The first 10 classes of the training set are: [5 0 4 1 9 2 1 3 1 4]
```

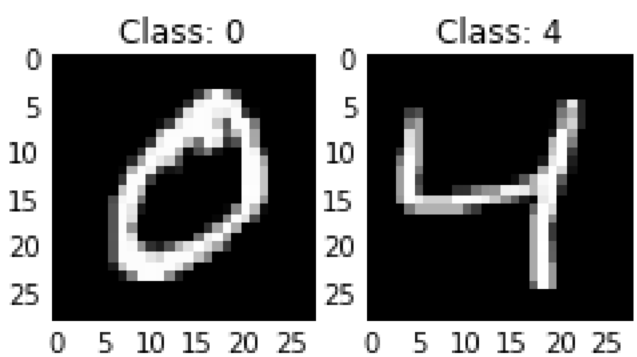
- Note each image is encoded as series of values between 0-1 representing pixel intensity



MNIST Data Set

```
In [8]: # ---- Loading images ----
        def show_images(image_array, class_array):
            """
            Renders a given set of images and their classes
            """
```

```
        def image(image_in, class_in, fig_in):
            """
            Render a single image assuming the 'figure' has been intialized
            """
            # Plotting image
            plt.imshow(image_in.reshape(28,28), interpolation='nearest', cmap='gray')
            # Fixing axes
            ax.xaxis.set_ticks_position('bottom')
            ax.yaxis.set_ticks_position('left')
            ax.set_aspect('equal')
            # Set Title
            ax.set_title('Class: {a}'.format(a=class_in))

        # -- Error handling --
        i_image, j_image = image_array.shape
        i_class = len(class_array)
        if (i_image != i_class):
            raise ValueError('Number of inputs does not equal number of ouputs')

        # Setup figure
        fig = plt.figure()

        for i in range(i_image):
            ax = fig.add_subplot(1,i_image+1,i+1)
            image(image_array[i],class_array[i],fig)
        plt.show()
    # Plotting images
    show_images(train_set[0][1:3],train_set[1][1:3])
```



- The method above provides loads data from pickeled files
- MNIST was loaded from disk, we can read in other data formats

2.1.2 Loading Images from Folders

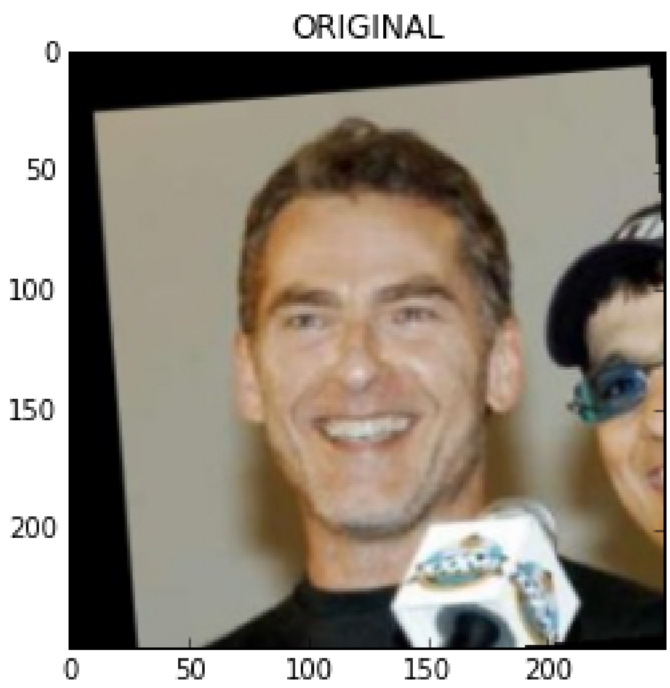
- There are different ways to read in data in python

- If all the files are located in a single folder then they can be read in using a generator and stored in an array
- Below is a general approach for iterating through folders, it can be adapted for files stored in a single folder
- We use the Labeled Faces in the Wild Home
  - Contains 13233, 250×250 color images of 5760 famous people
  - We'll compare two different methods for reading in files matplotlib and openCV
  - OpenCV is commercial grade software and reads the images about 20% faster than matplotlib

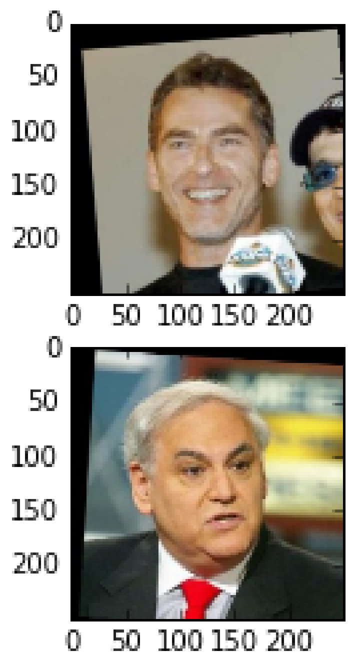
In [34]:
```
# --- Setting base dir ---
# Assuming your image folders is located in the base directory
imgs = []
base_path = os.getcwd()
base_folder_path = os.path.join(base_path,'lfw_funneled')
image_folders = map(lambda x: os.path.join(base_folder_path, x), os.listdir(path))
print 'There are {} folders containg images of people in the base folders'.format(len(i
```

```
There are 5760 folders containg images of people in the base folders
```

In [35]:
```
# --- Visualizing one picture ----
check = image_folders[0]
image_file_to_be_read = os.path.join(check, os.listdir(check)[0])
img=mpimg.imread(image_file_to_be_read)
plt.imshow(img),plt.title('ORIGINAL')
plt.show()
```



In [36]:
```python
# ---- Function for plotting multiple images ----
def plot_images(location_of_images):
    l = len(location_of_images)
    for i, image_i in enumerate(location_of_images):
        plt.subplot(l, 1, i+1)
        image_file_to_be_read = os.path.join(image_i, os.listdir(image_i)[0])
        img=mpimg.imread(image_file_to_be_read)
        plt.imshow(img)
    plt.show()
plot_images(image_folders[0:2])
```



In [37]:
```python
# ---- Crawler for images ----
images = [os.path.join(folder,file_in_folder)
          for folder in image_folders
          if os.path.isdir(folder) # check to ensure folder is a folder
          for file_in_folder in os.listdir(folder)
          ]
print 'There are {} unique images in this data set'.format(len(images))
```

```
There are 13233 unique images in this data set
```

```
In [39]: # --- Compare load times for 1000 images ---
         def opencv_load_image(images_to_be_loaded, n = 1000):
             start = timer()
             for i in range(n):
                  cv2.imread(images_to_be_loaded[n], cv2.IMREAD_COLOR)
             end = timer()
             return round(end - start,4)

         def matplotlib_load_image(images_to_be_loaded, n = 1000):
             start = timer()
             for i in range(n):
                  mpimg.imread(images_to_be_loaded[n])
             end = timer()
             return round(end - start,4)

         n=1000

         opencv_time = opencv_load_image(images, n = n)
         matplotlib_time = matplotlib_load_image(images, n = n)

         print 'OpenCV: Total elapse time: {} seconds for {} loads'.format(opencv_time,n)
         print 'Matplotlib: Total elapse time: {} seconds for {} loads'.format(matplotlib_time,n
         print 'OpenCV is approximately {}% faster than MatplotLib'.format(round(matplotlib_time
```

```
OpenCV: Total elapse time: 0.6955 seconds for 1000 loads
Matplotlib: Total elapse time: 0.8475 seconds for 1000 loads
OpenCV is approximately 21.9% faster than MatplotLib
```

- Next we load the full data set into python
- The array will be pre-allocated
- It's best practice to do this first with a small subset then scale up

```
In [41]: # -- Load locations of 100 images --
         images_subset = images[0:100]

         # -- Preallocate 24 bit array --
         data = np.empty((len(images_subset), 3, 250, 250), dtype=np.uint8)

         # -- Iterate through locations and store in array --
         for i, fpath in enumerate(images_subset):
           img = cv2.imread(fpath, cv2.IMREAD_COLOR)
           # transpose is necessary for correct colorization
           data[i, ...] = img.transpose(2, 0, 1)
```

2.1.3 Memory footprint

- Below is a utility function to assess the size of your data set is
- It is helpful to understand the amount of data loaded into python

- In general values are stored in memory for fast access
    - For best performance you'll want to make sure the loaded data do not exceed total RAM
    - When the amount of data loaded in RAM exceeds available RAM it gets written to disk
    - Reading and writing from disk is much slower than in-memory and will slow processing

In [42]:
```
import math

def convert_size(object_in):
    size_bytes = os.sys.getsizeof(object_in)
    if (size_bytes == 0):
        return '0B'
    size_name = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB")
    i = int(math.floor(math.log(size_bytes, 1024)))
    p = math.pow(1024, i)
    s = round(size_bytes/p, 2)
    return '%s %s' % (s, size_name[i])

print('For 100 images the total memory footprint is:' + str(convert_size(data)))
```

Out[42]:
```
'17.88 MB'
```

3 Loading data through Keras

- Keras is a high-level API
- It comes bundled with tensorflow
- Several common data sets can be loaded through it's API, these include:
    - **MNIST**: 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images
    - **Fashion MNIST**: 60,000 28x28 grayscale images of 10 fashion categories, along with a test set of 10,000 images
    - **IMDB**: 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative)
    - **CIFAR 10**: 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images
    - **CIFAR 100**: 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images
    - **Reuters newswire articles**: 11,228 newswires from Reuters, labeled over 46 topics
    - **Boston Housing Data**: 13 attributes of houses at different locations around the Boston suburbs in the late 1970s
- For a full description see the documentation here
- Notes:
    - This api is bundled with tensorflow
    - All data is stored in `~/.keras/datasets/ + path`
    - Do not use this api until lecture 5 where we discuss deep learning tools

* Installing tensorflow properly (with gpu support) isn't simple
* Installing base tensorflow (with cpu support) has much worse performance

In [ ]:
```
# --- Loading data sets through keras ----
from keras.datasets import mnist, fashion_mnist, boston_housing
from keras.dataset import cifar10, cifar100
from keras.datasets import imdb, reuters

# MNIST, Fashion MNIST, Boston Housing Data sets
(x_train, y_train), (x_test, y_test) = mnist.pkl.gz.load_data()
(x_train, y_train), (x_test, y_test) = fashion_mnist.pkl.gz.load_data()
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
(x_train, y_train), (x_test, y_test) = cifar10.load_data(label_mode='fine')
(x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='fine')

# Reuters data set
(x_train, y_train), (x_test, y_test) = reuters.load_data(path="reuters.npz",
                                                         num_words=None,
                                                         skip_top=0,
                                                         maxlen=None,
                                                         test_split=0.2,
                                                         seed=113,
                                                         start_char=1,
                                                         oov_char=2,
                                                         index_from=3)
word_index = reuters.get_word_index(path="reuters_word_index.json")

# Imdb data set
(x_train, y_train), (x_test, y_test) = imdb.load_data(path="imdb.npz",
                                                      num_words=None,
                                                      skip_top=0,
                                                      maxlen=None,
                                                      seed=113,
                                                      start_char=1,
                                                      oov_char=2,
                                                      index_from=3)
```

4 Additional DL Datasets

4.1 Specific datasets

1. Webhose.io - News Free Datasets
2. Yelp dataset
3. Uber TLC FOIL Response
4. Google Open Images Dataset
5. Maluuba NewsQA Dataset
6. Datascience Bowl 2017- Kaggle

4.2 Directories

1. MILA Lab datasets
2. Skymind Open Datasets
3. Awesome Deep Learning (scroll down to dataset section)
4. Kaggle Image datasets

5 Three Problems with Logistic Regression

- To motivate a neural network architecture we start with logistic regression
- In particular we'll see how logistic units form the foundation of more sophisticated and expressive architectures
- Ahead of jumping into the construction of networks though we discuss three common problems associated with "shallow" machine learning models
- With standard logistic regression three common problems are the:

  1. Donut problem,
  2. XOr problem,
  3. Multiclass classification

- Each of these can be solved with a clever trick however require human input
- Introducing them here motivates a discussion of the utility of NNs as generalizable models where special knowledge about the features or data structure is not necessary

5.1 XOr problem

- The XOr problem is based on the logic gate which has the following truth table

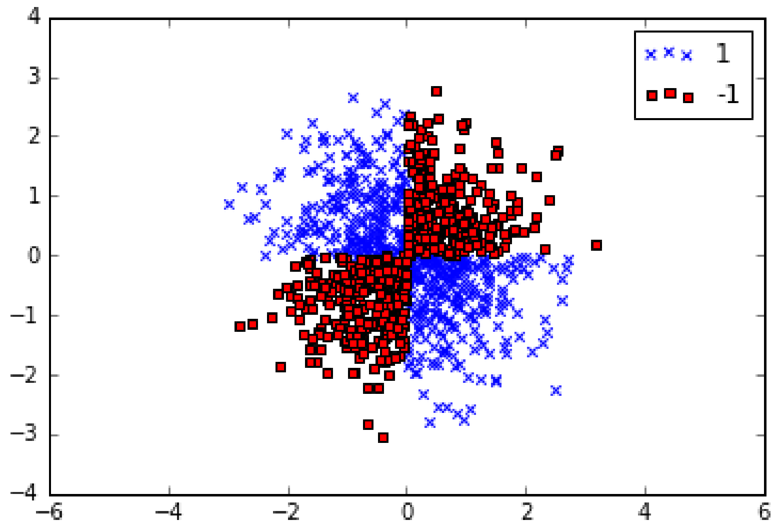
```
|a|b|a XOR b|
--------------
|1|1|   0   |
|0|1|   1   |
|1|0|   1   |
|0|0|   0   |
```

- This arises in a situation when you have exactly 1 value that is true (see plot below)

- A variant of the problem generates random samples within the different quadrants

In [2]:
```
# --- Loading libraries ---
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# --- Generating rvs for X-OR problem -
np.random.seed(0)
X_xor = np.random.randn(1000, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0, X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
plt.scatter(X_xor[y_xor==1, 0], X_xor[y_xor==1, 1],
```

```
c='b', marker='x', label='1')
plt.scatter(X_xor[y_xor==-1, 0], X_xor[y_xor==-1, 1],
c='r', marker='s', label='-1')
plt.ylim(-3.0)
plt.legend()
plt.axis('equal')
plt.show()
```



- Notice that with a logistic model there is no linear decision boundary that segments the quadrants

5.2 Donut Problem

- In the donut problem two concentric circles of varying radius and different classes are drawn
- One circle is contained in the other
- To generate a sample (x, y) coordinate pairs are sampled from 2 concentric circles with different bounding inner and outer radii
- A uniform random sampler is applied twice:

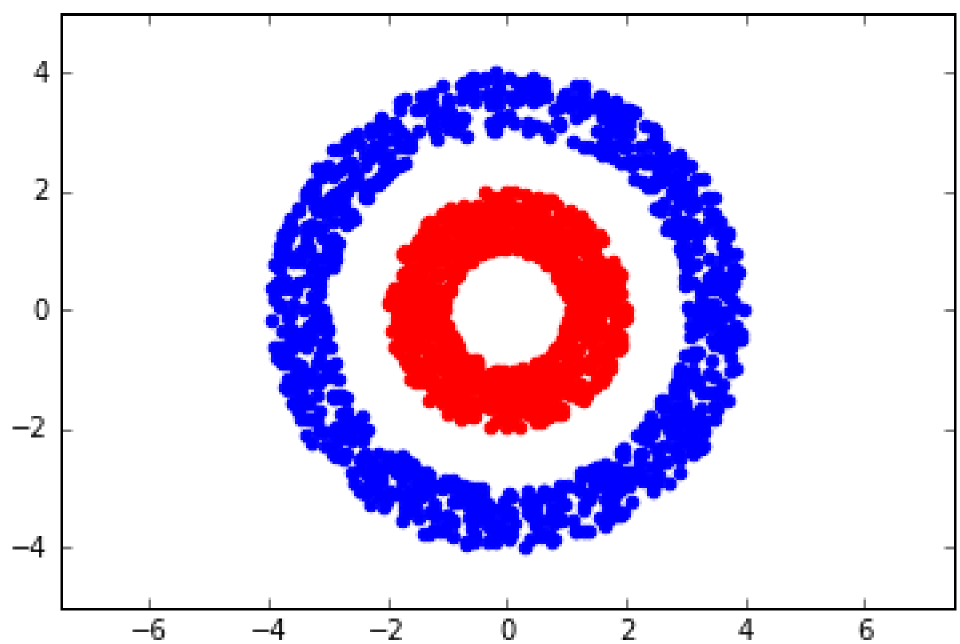
1. To generate a number between an inner and outer radius
2. To generate a random angle along with random

- Here the donuts are centered at (0,0)

```
In [3]: # --- Importing libraries ---
        import random
        import math
        import numpy as np
        import matplotlib.pylab as plt
```

```
# --- Function for generating a donut with inner_radius and outer_radius ---
def gen_random_points_for_a_donut(inner_radius=1, outer_radius=2, n=1000):
    random_points_in_donut = []
    for i in range(n):
        # Random angle
        alpha = 2 * math.pi * random.random()
        # Random point in between inner and outer radius
        r = random.uniform(inner_radius,outer_radius)
        # calculating coordinates
        x = r * math.cos(alpha)
        y = r * math.sin(alpha)
        random_points_in_donut += [(x,y)]
    return np.array(random_points_in_donut)
```

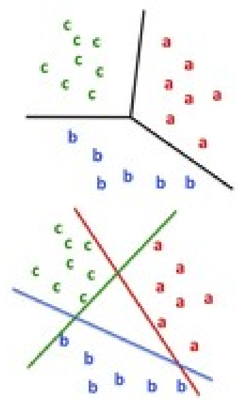
In [4]:
```
# --- Plotting the donut problem ---
donut_1 = gen_random_points_for_a_donut(inner_radius=3, outer_radius=4)
donut_2 = gen_random_points_for_a_donut(inner_radius=1,outer_radius=2)
plt.scatter(donut_1[:,0],donut_1[:,1], color ='blue')
plt.scatter(donut_2[:,0],donut_2[:,1], color='red')
plt.axis('equal')

plt.show()
```



- The donut problem like the XOr has 2 clearly divided classes however their is no linearly separable decision boundary that correctly classify red and blue points
- Logistic units have limited capacity for expressing more complex decision boundaries

5.3 Multiclass classification

- Logistic units are binary classifiers
- Multi-class classification with logistic units is limited by the linearity of their decision boundaries (see image below)



6 Biological Inspiration: Neural Networks and the Brain

6.1 Neural networks expressiveness?

- In the previous section three canonic logistic challenges were introduced
- Historically approaches to such problems have involved feature engineering, subject matter expertise or more sophisticated algorithms
- Most of historic solution likely use **shallow models** in that they prescribe a structure to the problem rather than letting the **model** learn the structure

6.2 Solution to the Donut problem

- Only the (x, y) coordinates are given
- Two approaches to solving the donut problem include:

  1. Feature engineering
     - A combination of the x and y coordinates are used to calculate to extend the linear activation function to include a quadratic term that calcultes the circle, i.e.

$$a(x, y) = \beta_0 + \beta_x x + \beta_y y + \beta_r \sqrt{x^2 + y^2}$$

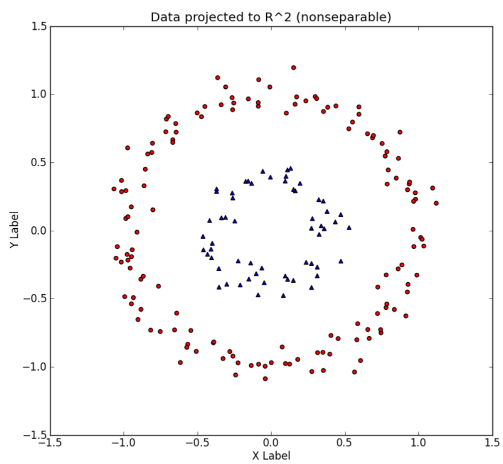
     - Once the parameter β_r is calculated it is sufficient to determine whether a point lies in a the inner or outer donut.
     - By using the combination of (x,y) problems dimensionality reduces from 2 to 1.
     - Notice that θ the angle does not matter.
  2. Support vector machines - selecting kernels
     - With support vector machines we select a kernel then minimize a cost function
     - The "kernel trick" is used to ensure we have a linear representation of features

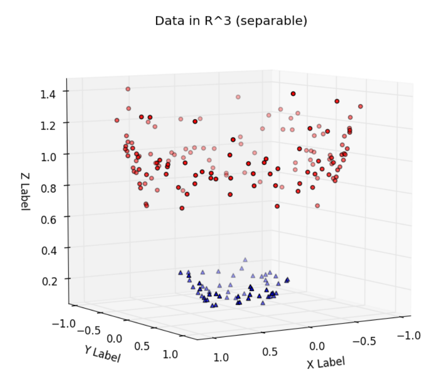
- This means support vector machines rely on transformations of the input space into some other such that we can find a linearly seperable hyperplane (or at least one this has a minimal marginal distance cost)
- A canonic kernel for the donut problem is to use a 3 dimensions mapping such that:

$$\phi : (x_1, x_2) \to (x_1^2, \sqrt{2x_1x_2}, x_2^2) \text{ where } f(\mathbf{x}) = \mathbf{w}^T \cdot \phi(\mathbf{x}) + b$$

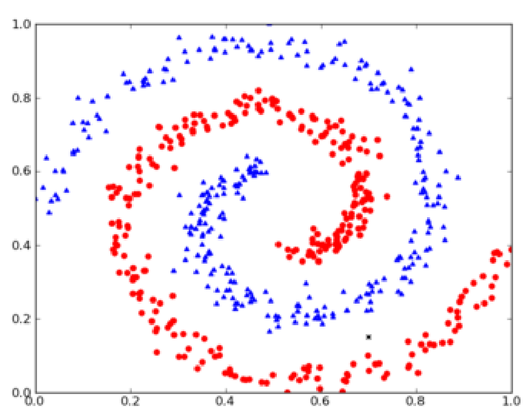
where **w** is minimized with respect to the maximum margin minimum classifier:

$$\min_{\mathbf{w}\in\mathbb{R}} \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(w \cdot x_i + b)\right) \right] + \lambda ||w||^2$$





- In both cases, a solution to the donut problem required both knowing how to augment the feature set, select the machine learning method and then identify the right parameter or functional inputs
- The point is that neither approaches are sufficiently **generalizable**
- That is, without some sort of subject matter expertise, they cannot be applied to other problems with a similar flavor like the twin spiral problem

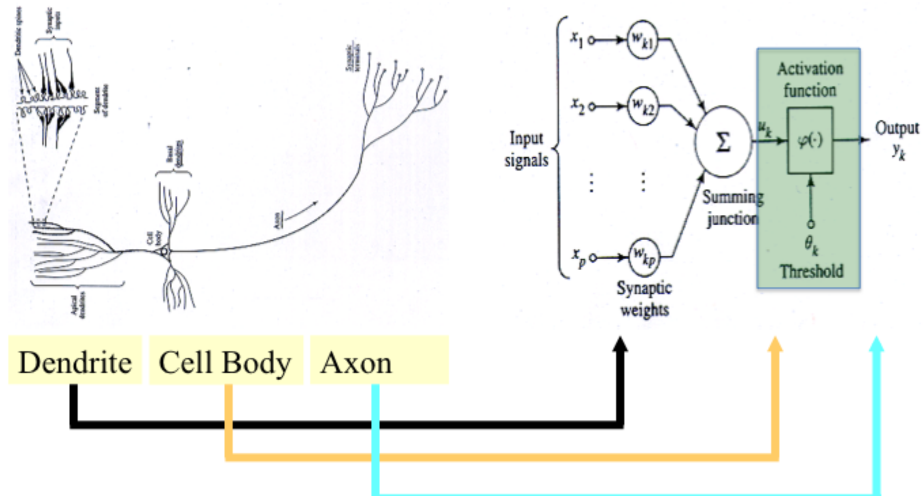


- One of the goals of neural networks is to setup architectures that allow networks to generalize on similar learning tasks

- So whether a donut, XOr or spiral problem is passed into a neural network it should be able to classify the different points without additional input
- Tensorflow playground is a cool tool that shows you how NNs can be trained to address the problems above (see here)

6.3 Biological inspiration

- The inspiration for an artificial neuron arose in the 1950s and is usually credited to Rosenblatt's perceptron
- Artificial neurons "mimic" the way a biological neuron works
- A biological neuron is modulated by the signals it receives from other neurons
- Once their is sufficient positive charge then the neuron `activates` and send a signal down it's axon to other neurons
- The signal is a 0-1 potentiated value that connects with a particular strength to other neurons
- In a simplified world the a biological neuron can be mapped to the mathematical artificial construct:



6.3.1 Mathematics of an Artificial Neuron

- A biological neuron can be modeled assuming the following:
  - The signal inputs are from other neurons $\mathbf{x}$,
  - The connective weights $\mathbf{w}$ modulate the strength of the input neurons,
  - Inputs are additive and
  - Once the activation threshold is reached then a signal is sent.
- Based on the above the probability of the neuron activating is:

$$a(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{1}$$

$$\sigma : a(\mathbf{w}, \mathbf{x}) \rightarrow y \in [0, 1] \tag{2}$$

where:

- b is the "resting polarization rate" (bias),
- $a(\cdot, \cdot)$ is the activation function,
- σ is a smoothed continuous polarization curve and

- y is a continuous value between 0 and 1 indicating the probability of polarization given $\mathbf{w}, \mathbf{x}$.

- In general $\sigma(\cdot)$ is the logistic unit, because
  - It is continuous,
  - Differentiable and
  - Characterizes the 0-1 potentiated signals for a linear activation input
- Some code is included below for constructing and visualizing your first artificial neuron

In [5]:
```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(123)
x = np.arange(-10,10,0.1)
y = 1/(1+np.exp(-x))
plt.plot(x,y)
plt.axhline(y=.5, xmin=-10, xmax=10, linewidth=2, color = 'k')
plt.ylabel('Activation Probability (y)')
plt.xlabel('Input (x)')
plt.title('Logisitic Regression Function')
plt.show()
```