Cristian David Estupiñán Ojeda
Christian Brito Ramos

PRÁCTICA 1

En primer lugar, para la primera parte de la práctica 1, al realizar el método de ramificación y acotación hay que tener en cuenta que no se necesita lista cerrada debido a que siempre la peor ruta con el path_cost más largo estará siempre al final. Por tanto podemos utilizar el tree_search y un método FIFO con una modificación:

```
def ramificacion_acotacion(problem):
    return tree_search(problem,FIFOQueueRAMACO())
```

Al realizar la modificación tengo en cuenta lo siguiente, al extender la lista abierta, me los deberá ordenar en función del path_cost, para que el menor siempre salga primero y el peor esté siempre al final:

```
class FIFOQueueRAMACO(Queue):

    def __init__(self):
        self.A = []
        self.start = 0

    def append(self, item):
        self.A.append(item)

    def __len__(self):
        return len(self.A) - self.start

    def extend(self, items):
        self.A.extend(items)
        self.A.sort(key=lambda n: n.path_cost)

    def pop(self):
        e = self.A[self.start]
        self.start += 1
        if self.start > 5 and self.start > len(self.A) / 2:
            self.A = self.A[self.start:]
            self.start = 0
        return e
```

Para la segunda parte de la práctica, vuelvo a utilizar el método del tree_search por el mismo motivo que el de la primera parte, y esta vez puedo utilizar también el método ya implementado PriorityQueue debido a que si le paso por parámetro una función, me devolverá el el elemento mínimo primero. Para este caso con subestimación le sumo la distancia euclídea:

```
def ramificacion_acotacion_subestimacion(problem):
    return tree_search(problem,PriorityQueue(f=lambda x: x.path_cost +
problem.h(x)))
```

Tenemos en cuenta que los métodos de anchura y profundidad son métodos de búsqueda de fuerza bruta sin una base fundamentada para encontrar el camino óptimo, es por ello que ramificación y acotación con subestimación obtiene un número de nodos expandidos menor casi siempre que estos métodos, además de asegurarse que es una mejor opción. Si la heurística no fuera consistente, seguiríamos expandiendo nodos en el árbol y tendríamos una lista más larga, se puede observar cuando hacemos búsquedas sin subestimación.

Cristian David Estupiñán Ojeda
Christian Brito Ramos

PRÁCTICA 2

En primer lugar, se obtienen los datos y se utiliza la función one_hot para codificar los resultados en Unos y Ceros.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Translate a list of labels into an array of 0's and one 1.
# i.e.: 4 -> [0,0,0,0,1,0,0,0,0,0]
def one_hot(x, n):
    """
    :param x: label (int)
    :param n: number of bits
    :return: one hot code
    """
    if type(x) == list:
        x = np.array(x)
    x = x.flatten()
    o_h = np.zeros((len(x), n))
    o_h[np.arange(len(x)), x] = 1
    return o_h

data = np.genfromtxt('iris.data', delimiter=",")  # iris.data file loading
np.random.shuffle(data)  # we shuffle the data
x_data = data[:, 0:4].astype('f4')  # the samples are the four first rows of
data
y_data = one_hot(data[:, 4].astype(int), 3)  # the labels are in the last row.
Then we encode them in one hot code
```

Después, guardamos un espacio para x e y. X son las entradas e Y las salidas que se deben dar, las etiquetas. Definimos una capa oculta de 5 neuronas a las que se le pasa las 4 entradas del problema, y las salidas de esas neuronas con 3 neuronas para obtener los resultados:

```
x = tf.placeholder("float", [None, 4])  # samples
y_ = tf.placeholder("float", [None, 3])  # labels

W1 = tf.Variable(np.float32(np.random.rand(4, 5)) * 0.1)
b1 = tf.Variable(np.float32(np.random.rand(5)) * 0.1)

W2 = tf.Variable(np.float32(np.random.rand(5, 3)) * 0.1)
b2 = tf.Variable(np.float32(np.random.rand(3)) * 0.1)
```

En el siguiente paso, h da un resultado según la función sigmoide entre unos y ceros. Y utiliza la función Softmax para utilizarlo como una función probabilidad y convertir el conjunto de evidencias en una probabilidad de que las entradas son de un cierto tipo:

```
h = tf.nn.sigmoid(tf.matmul(x, W1) + b1)
# h = tf.matmul(x, W1) + b1  # Try this!
y = tf.nn.softmax(tf.matmul(h, W2) + b2)
```

Cristian David Estupiñán Ojeda
Christian Brito Ramos

En loss obtenemos la pérdida y con la función del gradiente lo minimizamos:

```
loss = tf.reduce_sum(tf.square(y_ - y))

train = tf.train.GradientDescentOptimizer(0.01).minimize(loss)  # learning
rate: 0.01
```

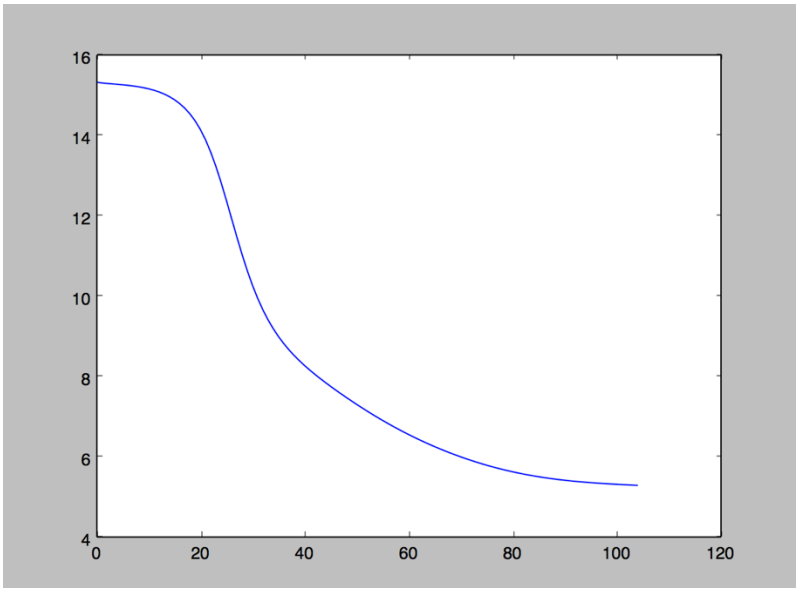
Para finalizar, obtenemos los tamaños de los diferentes conjuntos de entrenamiento, test y validación. Tenemos en cuenta una condición de parada en caso de que el error no mejore en más de 10 veces seguidas y le damos un mínimo de 30 épocas para ejecutar sí o sí. Primero entrenamos cada lote, se pasa la función loss y obtenemos el error para saber si mejora o no, y comprobamos los resultados con el conjunto test:

```
batch_size = 20
tamanoEntrenamiento = int(len(x_data)*0.7)
tamanoTest = int(len(x_data)*0.15)
tamanoValidacion = len(x_data) - tamanoEntrenamiento - tamanoTest

errorAnterior = 0
contadorEstabilizacion = 0
vectorErrores = []

epoch = 0

while (contadorEstabilizacion < 15) or (epoch <= 30):
    for jj in xrange(int(tamanoEntrenamiento/batch_size)):
        minimo = min(batch_size, tamanoEntrenamiento - jj * batch_size)
        batch_xs = x_data[jj * batch_size: jj * batch_size + minimo]
        batch_ys = y_data[jj * batch_size: jj * batch_size + minimo]
        sess.run(train, feed_dict={x: batch_xs, y_: batch_ys})

    batch_xv = x_data[jj * batch_size + minimo: jj * batch_size + minimo +
tamanoValidacion]
    batch_yv = y_data[jj * batch_size + minimo: jj * batch_size + minimo +
tamanoValidacion]
    error = sess.run(loss, feed_dict={x: batch_xv, y_: batch_yv})
    vectorErrores.append(error)

    #Comprobacion de la estabilizacion

    if (epoch == 0):
        errorAnterior = error
    elif (error >= errorAnterior * 0.95):
        contadorEstabilizacion += 1
    else:
        contadorEstabilizacion = 0
        errorAnterior = error

    print "Estabilizacion: %d" % contadorEstabilizacion


    print "Epoch #:", epoch, "Error: ", error
    result = sess.run(y, feed_dict={x: batch_xv})
    for b, r in zip(batch_yv, result):
        print b, "-->", r
    print "----------------------------------------------------------------------
----------"
    epoch += 1

batch_xt = x_data[len(x_data) - tamanoTest:]
batch_yt = y_data[len(y_data) - tamanoTest:]

contadorFallos = 0
```

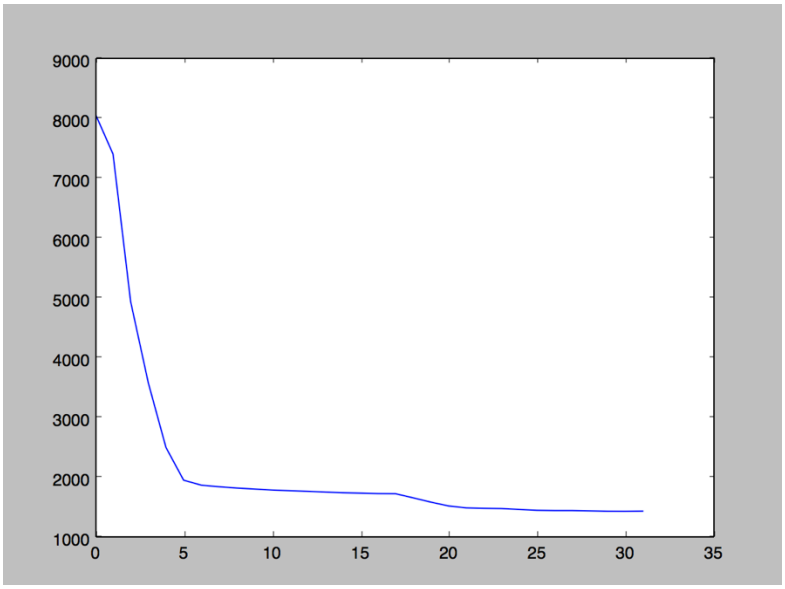
Cristian David Estupiñán Ojeda
Christian Brito Ramos

```
result = sess.run(y, feed_dict={x: batch_xt})
for b, r in zip(batch_yt, result):
    if np.argmax(b) != np.argmax(r):
        contadorFallos += 1


print "El numero de fallos obtenido es de: %d" % contadorFallos
print "El porcentaje es de: %f" % float(contadorFallos/tamanoTest)

plt.figure()
vectorErrores = np.array(vectorErrores)
plt.plot(vectorErrores)
plt.show()
```

Estos son los resultados de los errores frente a las épocas del iris:



Para mnist se sigue el mismo procedimiento que para iris pero el número de épocas mínimo pasa a ser 20 y el número de neuronas pasa a ser 10 en la capa intermedia:



Cristian David Estupiñán Ojeda
Christian Brito Ramos

PRÁCTICA 3

En primer lugar, creo un diccionario inverso al que se encuentra en la práctica para que en función del índice del valor máximo del estado y la acción que deba tomar, realice dicha acción.

```
actions_list_inverso = {0: "UP",
                        1: "RIGHT",
                        2: "DOWN",
                        3: "LEFT",
                        }
```

Cuento el número de acciones promedio aumentando la variable después de ir a cada estado. Y lo muestro dividiéndolo por el número de episodios que se han ejecutado.

```
# CALCULO PROMEDIO EXPLORACION

print "Exploracion: "

contadorExploracion = 0
numeroEpisodios = 1000

# Episodes
for i in xrange(numeroEpisodios):
    state = getRndState()
    while state != final_state:
        action = getRndAction(state)
        y = getStateCoord(state)[0] + actions_vectors[action][0]
        x = getStateCoord(state)[1] + actions_vectors[action][1]
        new_state = getState(y, x)
        qlearning(state, actions_list[action], new_state)
        state = new_state
        contadorExploracion = contadorExploracion + 1

print "Calculo promedio:"
print contadorExploracion/numeroEpisodios
```

Para la explotación, en caso de que la lista del estado actual tenga algún valor distinto de 0, se cogerá de la lista el índice del valor más grande y esa será la acción que deberá realizar. Vuelvo además a tomar el número de cálculos promedio e imprimirlo por pantalla.

```


# CALCULO PROMEDIO GREEDY

print "Greedy: "

Q = np.zeros((height * width, num_actions))
contadorGreedy = 0
numeroEpisodios = 1000


# Episodes
for i in xrange(numeroEpisodios):
    state = getRndState()
    while state != final_state:
        if max(Q[state]) != 0:
            indice = np.argmax(Q[state])
```

```
            action = actions_list_inverso[indice]
        else:
            action = getRndAction(state)
        y = getStateCoord(state)[0] + actions_vectors[action][0]
        x = getStateCoord(state)[1] + actions_vectors[action][1]
        new_state = getState(y, x)
        qlearning(state, actions_list[action], new_state)
        state = new_state
        contadorGreedy += 1

print "Calculo promedio:"
print contadorGreedy/numeroEpisodios
```

Para el caso del e-greedy es similar al de la explotación, con la diferencia de que si existe algún número random que sea superior a un porcentaje determinado, que tome una acción aleatoria aun teniendo algún valor distinto de 0 en la lista de acciones del estado en el que se encuentre. Le di 3 valores distintos: 0,9;0,85;0,8.

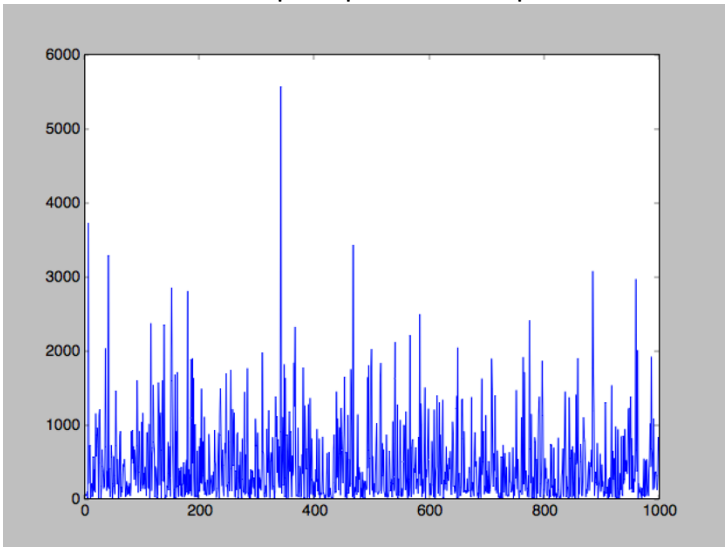
```
# CALCULO PROMEDIO E-GREEDY e=0.9

print "E-Greedy e=0.9: "

Q = np.zeros((height * width, num_actions))
contadorEGreedy09 = 0
numeroEpisodios = 1000
e = 0.9


# Episodes
for i in xrange(numeroEpisodios):
    state = getRndState()
    while state != final_state:
        if max(Q[state]) != 0:
            if (random.random() > e):
                action = getRndAction(state)
            else:
                indice = np.argmax(Q[state])
                action = actions_list_inverso[indice]
        else:
            action = getRndAction(state)
        y = getStateCoord(state)[0] + actions_vectors[action][0]
        x = getStateCoord(state)[1] + actions_vectors[action][1]
        new_state = getState(y, x)
        qlearning(state, actions_list[action], new_state)
        state = new_state
        contadorEGreedy09 = contadorEGreedy09 + 1

print "Calculo promedio:"
print contadorEGreedy09/numeroEpisodios

# CALCULO PROMEDIO E-GREEDY e=0.85

print "E-Greedy e=0.85: "

Q = np.zeros((height * width, num_actions))
contadorEGreedy085 = 0
numeroEpisodios = 1000
e = 0.85


# Episodes
for i in xrange(numeroEpisodios):
```

Cristian David Estupiñán Ojeda
Christian Brito Ramos

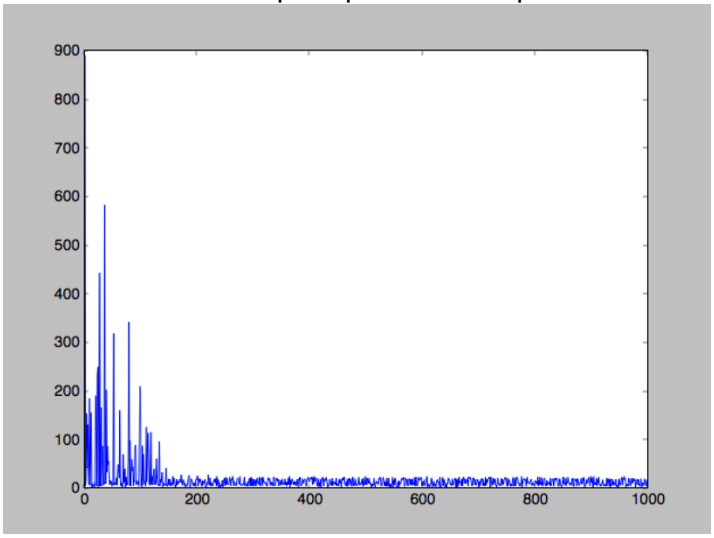
```
    state = getRndState()
    while state != final_state:
        if max(Q[state]) != 0:
            if (random.random() > e):
                action = getRndAction(state)
            else:
                indice = np.argmax(Q[state])
                action = actions_list_inverso[indice]
        else:
            action = getRndAction(state)
        y = getStateCoord(state)[0] + actions_vectors[action][0]
        x = getStateCoord(state)[1] + actions_vectors[action][1]
        new_state = getState(y, x)
        qlearning(state, actions_list[action], new_state)
        state = new_state
        contadorEGreedy085 = contadorEGreedy085 + 1

print "Calculo promedio:"
print contadorEGreedy085/numeroEpisodios

# CALCULO PROMEDIO E-GREEDY e=0.8

print "E-Greedy e=0.8: "

Q = np.zeros((height * width, num_actions))
contadorEGreedy08 = 0
numeroEpisodios = 1000
e = 0.8


# Episodes
for i in xrange(numeroEpisodios):
    state = getRndState()
    while state != final_state:
        if max(Q[state]) != 0:
            if (random.random() > e):
                action = getRndAction(state)
            else:
                indice = np.argmax(Q[state])
                action = actions_list_inverso[indice]
        else:
            action = getRndAction(state)
        y = getStateCoord(state)[0] + actions_vectors[action][0]
        x = getStateCoord(state)[1] + actions_vectors[action][1]
        new_state = getState(y, x)
        qlearning(state, actions_list[action], new_state)
        state = new_state
        contadorEGreedy08 = contadorEGreedy08 + 1

print "Calculo promedio:"
print contadorEGreedy08/numeroEpisodios
```

Cristian David Estupiñán Ojeda
Christian Brito Ramos

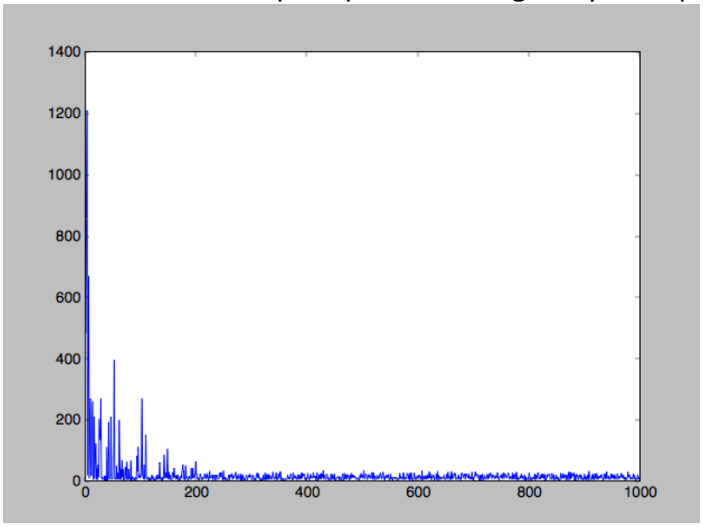
Número de acciones por episodio de exploración:



Número de acciones por episodio de explotación:

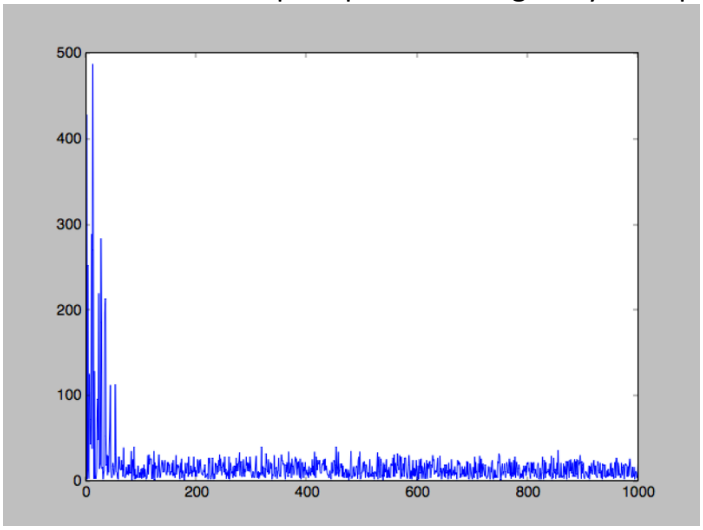


Número de acciones por episodio de E-greedy con épsilon = 0.9:



Cristian David Estupiñán Ojeda
Christian Brito Ramos

Número de acciones por episodio de E-greedy con épsilon = 0.85:



Número de acciones por episodio de E-greedy con épsilon = 0.8: