PONG (Policy Gradients)

Basado en el código de Karpathy https://gist.github.com/karpathy/a4166c7fe253700972fcbc77e4ea32c5 (https://gist.github.com/karpathy/a4166c7fe253700972fcbc77e4ea32c5)

Con Policy Gradients las recompensas se asignan de manera diferente en comparación a DQN. En este caso, se recompensan las acciones que han llevado a obtener una recompensa positiva. Por ejemplo, si obtenemos una recompensa de +1 en el momento 200, la recompensa del momento 199 sería de +0.99, la recompensa del momento 198 de +0.98 y así sucesivamente. Por lo tanto está recompensando todos los movimientos que le han llevado a hacer un punto.

· Librerias

In [1]:

```
# Librerias
import numpy as np
import os
import gym
import moviepy.editor as mp

import matplotlib.pyplot as plt
from matplotlib import animation

from keras.layers import Dense, Convolution2D, Activation, Flatten
from keras.models import Sequential
```

```
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html (htt
ps://www.pygame.org/contribute.html)

Using TensorFlow backend.
```

· Funciones

In [2]:

```
# Guardar los frames como un gif
def save_frames_as_gif(frames, filename=None):
    # Save a list of frames as a gif
    patch = plt.imshow(frames[0])
    plt.axis('off')
    def animate(i):
        patch.set_data(frames[i])
    anim = animation.FuncAnimation(plt.gcf(), animate, frames = len(frames), interval=50)
    if filename:
        anim.save(filename, dpi=72, writer='imagemagick')

# Cargar pesos existentes (si existen)
def cargarpesos(name_weights):
    if (os.path.isfile(name_weights)):
        model.load_weights(name_weights)
        print('Pesos cargados')
    else:
        print('No se encuentran pesos con este nombre')

# Resultado de un episodio
def resultado_episodio(reward_list,n):
    agente = reward_list.count(1.0)
    maquina = reward_list.count(-1.0)
    reward = agente-maquina
    if agente > maquina:
        res = 'WIN'
    else:
        res = 'LOSE'
    print('Episodio {} -> {} ({}-{}), Reward: {}'.format(n,res,maquina,agente, reward))
    return reward

# Resultado de n episodios
def resultados_n_episodios(reward_list_episodes):
    list_reward = []
    for i in range(len(reward_list_episodes)):
        list_reward.append(resultado_episodio(reward_list_episodes[i],i+1))
    print('-> Reward mean: ', np.mean(list_reward),'\n')
```

· Entorno

In [3]:

```
# Environment
env = gym.make("Pong-v0")
observation = env.reset()
prev_input = None

# Macros
UP_ACTION = 2
DOWN_ACTION = 3
```

· Preprocesamiento de las observaciones (imágenes) y Asignación Recompensas

In [4]:

```
# Preprocesamiento para las observaciones
def prepro(I):
    #prepro 210x160x3 uint8 frame into 6400 (80x80) 1D float vector
    I = I[35:195] # crop
    I = I[::2,::2,0] # downsample by factor of 2
    I[I == 144] = 0 # erase background (background type 1)
    I[I == 109] = 0 # erase background (background type 2)
    I[I != 0] = 1 # everything else (paddles, ball) just set to 1
    return I.astype(np.float).ravel()

# Discount reward
def discount_rewards(r, gamma):
    #take 1D float array of rewards and compute discounted reward
    r = np.array(r)
    discounted_r = np.zeros_like(r)
    running_add = 0
    # we go from last reward to first one so we don't have to do exponentiations
    for t in reversed(range(0, r.size)):
        if r[t] != 0:
            running_add = 0 # if the game ended (in Pong), reset the reward sum
            running_add = running_add * gamma + r[t] # the point here is to use Horner's me
            discounted_r[t] = running_add
    discounted_r -= np.mean(discounted_r) #normalizing the result
    discounted_r /= np.std(discounted_r) #idem
    return discounted_r
```

· Modelo de red neuronal

In [5]:

```
# Modelo - red densa

model = Sequential()

model.add(Dense(units=200,input_dim=80*80, activation='relu', kernel_initializer='glorot_un
model.add(Dense(units=1, activation='sigmoid', kernel_initializer='RandomNormal'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
```

```
WARNING:tensorflow:From C:\Users\Ocin\Anaconda3\lib\site-packages\tensorflow
\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tens
orflow.python.ops.array_ops) is deprecated and will be removed in a future v
ersion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 200)               1280200
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 201
=================================================================
Total params: 1,280,401
Trainable params: 1,280,401
Non-trainable params: 0
_________________________________________________________________
```

- Entrenamiento

In [6]:

```
# Hyperparameters
gamma = 0.99
episodes_train = 1000

# initialization of variables used in the main loop
x_train, y_train, rewards = [],[],[]
reward_sum = 0
episode_nb = 0

# initialize variables
epochs_before_saving = 10
```

In [7]:

```
# Cargar pesos existentes (si existen)
load_weights = False # True para cargar pesos

if load_weights:
    cargarpesos('trained_weights/weights_trained.h5')
```

In [8]:

```
# Entrenamiento
train = False # True para entrenar

ep = 0

if train:

    while (True):

        # preprocess the observation, set input as difference between images
        cur_input = prepro(observation)
        x = cur_input - prev_input if prev_input is not None else np.zeros(80 * 80)
        prev_input = cur_input

        # forward the policy network and sample action according to the proba distribution
        proba = model.predict(np.expand_dims(x, axis=1).T)
        action = UP_ACTION if np.random.uniform() < proba else DOWN_ACTION
        y = 1 if action == 2 else 0 # 0 and 1 are our labels

        # log the input and label to train later
        x_train.append(x)
        y_train.append(y)

        # visualizar durante el entrenamiento
        env.render()

        # do one step in our environment
        observation, reward, done, info = env.step(action)
        rewards.append(reward)
        reward_sum += reward

        # end of an episode
        if done:
            print('At the end of episode', episode_nb, 'the total reward was :', reward_sum

            # increment episode number
            episode_nb += 1

            # training
            model.fit(x=np.vstack(x_train), y=np.vstack(y_train), verbose=2, sample_weight=

            # Saving the weights used by our model
            if episode_nb % epochs_before_saving == 0:
                model.save_weights('my_model_weights.h5')

            # Reinitialization
            x_train, y_train, rewards = [],[],[]
            observation = env.reset()
            reward_sum = 0
            prev_input = None

            # End
            if ep >= episodes_train:
                break
            else:
                ep += 1
```

- Test

In [9]:

```
# Test
test = False # True para testear
episodes_test = 3 # Nº de episodios test


ep = 1
frames = []
frames_episodes = []
reward_list = []
reward_list_episodes = []

if test:

    # Cargar pesos entrenados
    cargarpesos('trained_weights/weights_trained.h5')

    while(True):

        cur_input = prepro(observation)
        x = cur_input - prev_input if prev_input is not None else np.zeros(80 * 80)
        prev_input = cur_input
        proba = model.predict(np.expand_dims(x, axis=1).T)
        action = UP_ACTION if np.random.uniform() < proba else DOWN_ACTION

        # visualizar durante el test
        env.render()

        observation, reward, done, info = env.step(action)
        frames.append(observation) # collecting observation
        reward_list.append(reward) # collecting reward

        # if episode is over, reset to beginning
        if done:
            frames_episodes.append(frames)
            reward_list_episodes.append(reward_list)
            observation = env.reset()

            frames = []
            reward_list = []

            # End
            if ep >= episodes_test:
                break
            else:
                ep += 1

    # Resultado
    resultados_n_episodios(reward_list_episodes)
```

Resultados Test

Con un entrenamiento de 1300 episodios.

```
Episodio 1 -> WIN (19-21), Reward: 2
Episodio 2 -> WIN (17-21), Reward: 4
Episodio 3 -> LOSE (21-20), Reward: -1
-> Reward mean:  1.6666666666666667
```

Video (guardar y visualizar un episodio del test)

In [10]:

```
# Crear el gif a partir de los frames (del test)
crear_gif = False # True para crear gif

if crear_gif:
    save_frames_as_gif(frames_episodes[1], filename='pong.gif')
```

In [11]:

```
# Convertir el gif a video
convert_mp4 = False # True para convertir gif a mp4

if convert_mp4:
    clip = mp.VideoFileClip("pong.gif")
    clip.write_videofile("pong.mp4")
```

In [12]:

```
%%HTML
<video width="432" height="288" controls>
<source src="partida_gif_mp4/pong.mp4" type="video/mp4">
</video>
```

Conclusiones

Como se puede apreciar los resultados no son tan buenos como en el algoritmo DQN, pero cabe decir que el entrenamiento en policygradients es de 1300 episodios y que la red neuronal no es convolucional sinó densa.