

## Tutorial Dream On Gym

Para la versión v0.0.7

Dream On Gym es un framework que permite implementar con pocas líneas de código el entrenamiento de un agente para tomar decisiones en un entorno de redes de fibra ópticas.

Por debajo cuenta con un simulador que representa el escenario en donde el Agente hará el uso de la red para tomar decisiones y aprender para tomar decisiones posteriormente.

La herramienta permite montar un environment el cual conecta el simulador con Gymnasium.

La tecnologías necesaria para montar DOG se necesita:

- Python 3.10
- Gymnasium 0.28
- Tensorflow 2.12

Para instalarla fácilmente se puede instalar con la aplicación pip:

```
pip install dream-on-gym-v2
```

Nota: la primera versión está basada con tecnologías obsoletas, por favor usar la versión 2.

Para ejecutar un ejemplo simple es necesario primero entender cómo funciona la herramienta.

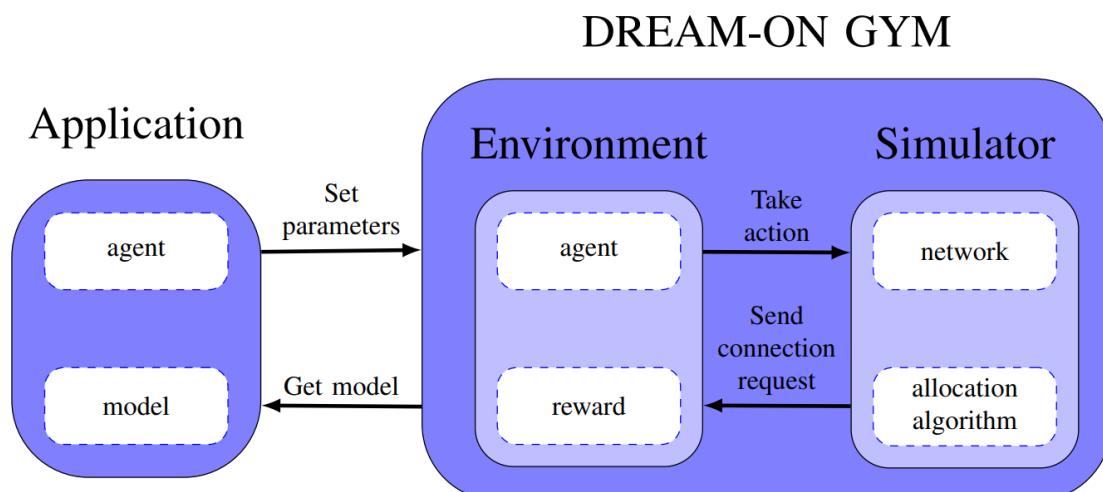


Fig. 1. DREAM-ON GYM architecture

La herramienta permite modificar distintos algoritmos esenciales para el entrenamiento:

1: Allocator: Permite realizar un algoritmo de asignación de una solicitud, sólo se debe devolver si la asignación fue asignada o no fue asignada.

2: State: Método que permite entregar información del estado que se le entregará al agente. Nota: El software permite obtener el estado acotado en base a la conexión que será asignada.

3: Reward: Permite modificar el premio que se le entrega al agente.

4: Reset:

Con esto se puede ejecutar un experimento para entrenar un agente, este soporta los especificados en [Stable-Baselines 3](#), incluidos los SB3-Contrib.

La herramienta también soporta precalentamiento de la red, haciendo que la simulación corra sin tomas decisiones del agente. Una vez terminado el precalentamiento se comienza a tomar decisiones. Esta función puede adecuarse a cada estilo y aprendizaje que se quiera lograr en el agente.

Una vez instalada la herramienta, se podrá incluir la enviroment.

```
env = gymnasium.make("rlonenv-v0")
```

Desde esta variable se modifican todas las variables y métodos disponibles.

```
def reward():
    value = env.getSimulator().lastConnectionIsAllocated()
    if (value.name == Controller.Status.Not_Allocated.name):
        value = -1
    else:
        value = 1
    return value
```

Por ejemplo, se puede definir un simple método de reward que castigue si no fue aceptada la conexión y premie si es asignada, con valores -1 y 1 respectivamente.

Para asignar este método de reward al ambiente se utiliza:

```
env.setRewardFunc(reward)
```

Otro mínimo esencial es un algoritmo de asignación, en este caso presentamos una forma de hacer un first fit.

```

def first_fit_algorithm(src: int, dst: int, b: BitRate, c: Connection, n: Network, path, action):
    numberofSlots = b.getNumberofSlots(0)
    actionSpace = len(path[src][dst])
    link_ids = path[src][dst][action]
    general_link = []
    for _ in range(n.getLink(0).getSlots()):
        general_link.append(False)
    for link in link_ids:
        link = n.getLink(link.id)
        for slot in range(link.getSlots()):
            general_link[slot] = general_link[slot] or link.getSlot(slot)
    currentNumberSlots = 0
    currentSlotIndex = 0

    for j in range(len(general_link)):
        if not general_link[j]:
            currentNumberSlots += 1
        else:
            currentNumberSlots = 0
            currentSlotIndex = j + 1
        if currentNumberSlots == numberofSlots:
            for k in link_ids:
                c.addLink(
                    k, fromSlot=currentSlotIndex,
                    toSlot=currentSlotIndex+currentNumberSlots)
            return Controller.Status.Allocated, c
    return Controller.Status.Not_Allocated, c

```

El método debe recibir siempre, aunque no se utilice:

- src int: indica el índice del nodo fuente de tipo entero.
- dst int: indica el índice del nodo destino de tipo entero.
- b BitRate: indica la tasa de bit que serán trasmisidas en esta conexión de punto a punto.
- c Connection: es el objeto que indica todos los datos de la conexión en cuestión, incluyendo la ruta por donde pasará la conexión.
- n Network: Es un objeto que tiene toda la red.
- path: Es un arreglo de arreglos que contiene una lista de rutas y en cada ruta se identifica cada enlace por la que pasa.
- action: Es un objeto/valor libre que puede definir el programador, el cual se recibe y trae lo que se necesaria en la acción tomada, puede ser un valor, listas u objetos.

Para modificar en el entorno el algoritmo se hace con el siguiente comando

```
env.getSimulator().setAllocator(first_fit_algorithm)
```

Se puede definir un entrenamiento previo de la red, para ello se configura con la siguiente opción:

```
env.getSimulator().goalConnections = 10000
```

Significa que habrá 10000 conexiones en el simulador ingresadas antes de que comience el entrenamiento. Esto permite cuando empiece aprender el agente, tome mejores decisiones.

La última afirmación es dado que una red vacía siempre tendrá asignaciones aceptadas.

También pueden cambiar las llegadas y salidas con lambda y mu respectivamente.

```
env.getSimulator().setMu(1)  
env.getSimulator().setLambda(1000)
```

Tanto el algoritmo (Allocator Algorithm), como la tasa de llegada y salida pueden modificarse antes del calentamiento de la red y antes iniciar el entrenamiento del agente.