UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

Práctica 4

ROBÓTICA INTELIGENTE APLICADA

Grado en Inteligencia Artificial - 4^{Or} curso

Motivación

- Práctica optativa que permita:
  - Subir hasta +2 la nota promedio de las prácticas 1 a 3.
  - Eximir del examen (22/01/2026)*

Motivación

- 3 opciones:
  - Uso de una redescripción perceptual obtenida automáticamente (cont. Práctica 3).
  - Utilización de una arquitectura cognitiva clásica simbólica.
  - Utilización de una arquitectura en CDR (*cognitive developmental robotics).*

Opción 1: SRL

SRL

- Objetivo: utilizar un espacio de estados obtenido automáticamente en vez de definirlo a mano.

- Pasos:

  - Seleccionar un método de SRL (no es necesario reentrenar aunque puede ser útil) que proporcione una salida similar ante el objeto deseado tanto en simulación como en robot real => CNN preentrenada sin capas de clasificación?

  - Realizar el entrenamiento de la práctica 1 con el espacio de estados aprendido.

  - Probar en el robot real de forma similar a la práctica 3.

SRL: Ejemplo sencillo con CNN ad-hoc

```python
import gymnasium as gym
import torch as th
from stable_baselines3 import PPO
from stable_baselines3.common.torch_layers import BaseFeaturesExtractor

class SimpleVAEExtractor(BaseFeaturesExtractor):
    """Extraer features con VAE simplificado"""
    def __init__(self, observation_space, features_dim=64):
        super().__init__(observation_space, features_dim)
        # CNN encoder
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 32, 8, stride=4), nn.ReLU(),
            nn.Conv2d(32, 64, 4, stride=2), nn.ReLU(),
            nn.Conv2d(64, 64, 3, stride=1), nn.ReLU(),
            nn.Flatten()
        )
        # Linear mapping a latent space
        self.linear = nn.Linear(64*7*7, features_dim)

    def forward(self, observations):
        return self.linear(self.cnn(observations))

# Uso
env = gym.make("CartPole-v1")
policy_kwargs = dict(features_extractor_class=SimpleVAEExtractor)
model = PPO("CnnPolicy", env, policy_kwargs=policy_kwargs)
model.learn(10000)
```

SRL: Ejemplo sencillo con ResNet50

```python
import torch
from torchvision.models import resnet50
from torchvision.models.feature_extraction import create_feature_extractor

# Extraer features de capas intermedias
model = resnet50(pretrained=True)
feature_extractor = create_feature_extractor(
    model,
    return_nodes={"layer4": "features"}
)

# Usar en SB3
class TorchvisionExtractor(BaseFeaturesExtractor):
    def __init__(self, observation_space, features_dim=2048):
        super().__init__(observation_space, features_dim)
        self.feature_extractor = feature_extractor

    def forward(self, observations):
        return self.feature_extractor(observations)["features"]
```

SRL: algunas opciones

- CNN:
  - **MobileNet**
  - EfficientNet
  - **ResNet**
  - VGG
  - Inception
- Vision Transformers:
  - **ViT**

SRL: enlaces útiles

- https://docs.pytorch.org/vision/stable/index.html
- https://github.com/huggingface/pytorch-image-models
- https://huggingface.co/docs/timm/feature_extraction
  - Modelos adicionales.
  - Método `forward_features()`.

Opción 2: SOAR

SOAR

- Objetivo: utilizar una arquitectura cognitiva clásica.
- Pasos:
  - Recordar explicación de SOAR del año pasado.
  - Descargar https://soar.eecs.umich.edu/
  - Tutorial => https://github.com/SoarGroup/Engineers-Guide-to-Soar/
  - Modificar ejemplo?

Opción 3: e-MDB

e-MDB

- Objetivo: utilizar una arquitectura que siga el paradigma CDR.
- Pasos:
  - No dormir en la explicación de clase :-)
  - Descargar https://github.com/GII/emdb
  - Instalar ROS Humble.
  - Ver ejemplo en https://docs.pillar-robots.eu/projects/emdb_experiments_gii/en/latest/experiments/oscar_experiment.html
  - Reemplazar la implementación con ANN de los P-nodes por una en PyTorch y jugar con la arquitectura de la red y los hiperparámetros del entrenamiento para conseguir que este sea lo más rápido posible.