Supplementary material for Imitating Unknown Policies via Exploration

Nathan Schneider Gavenski[1]
nathan.gavenski@edu.pucrs.br
Juarez Monteiro[1]
juarez.santos@acad.pucrs.br
Roger Granada
roger.granada@acad.pucrs.br
Felipe Meneguzzi
felipe.meneguzzi@pucrs.br
Rodrigo C. Barros
rodrigo.barros@pucrs.br

Machine Learning Theory and Applications Research Group (MALTA), School of Technology, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, RS, Brazil

1 Training flow

In this section, we present the training flow for IUPE. As illustrated in Figure 1, the process of training starts by collecting pairs of states (s_t, s_{t+1}) from a given environment, followed by a random action a_t that was responsible for generating the transition between these two states. Once the data is collected, we start to train the Inverse Dynamics Model, IDM, as shown in the yellow box of Figure 1. The IDM is responsible for predicting the action that caused the transition between two consecutive states. Here, the IDM uses our proposed exploration strategy, which consists of choosing the action predicted by sampling from the $softmax$ distribution given the probabilities generated by the IDM.

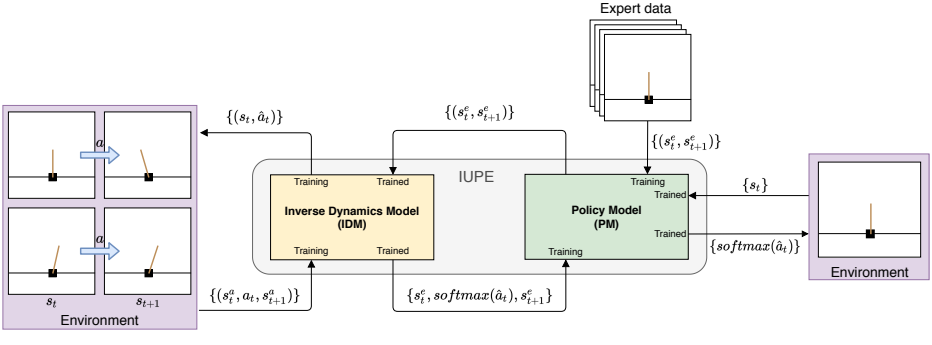

Figure 1: Pipeline representing the training flow for IUPE.

[1]These authors contributed equally to the work.

The next step is to train the Policy Model, PM, as shown in the green box of Figure 1. In this part of the flow, we start by generating a label for the data provided by the expert. For each pair of states (s_t^e, s_{t+1}^e) provided by the expert, the trained IDM will predict the action $(\hat{a}_t)$ that caused such transition. After generating the new expert dataset, now containing a predicted action, we start to train the Policy Model. The PM is trained by passing as input a given state (s_t), and the model needs to predict which action the expert will choose. The error is calculated with the previous prediction that comes from the IDM. In the end, the PM follows the same exploration strategy from the IDM. It samples from the $softmax$ distribution of the predictions from the PM to generate which action the agent needs to take.

2 IUPE Models

In this section, we present the Inverse Dynamics Model and the Policy Model implementations that are used in this work. In Figure 2, we divide the implementations in image-based and vector-based models given the environment they are on. Considering that the state representation varies in size, we do not present the input sizes. For the sake of simplicity, we show a simplified representation of the ResNet network.

In Figure 3, we present the self-attention modules that we add within the ResNet architecture. The overall architecture has been kept the same. We introduce both self-attention modules after the first and second ResBlocks from the encoder. The rest of the original implementation is untouched and follows the *PyTorch*'s framework.

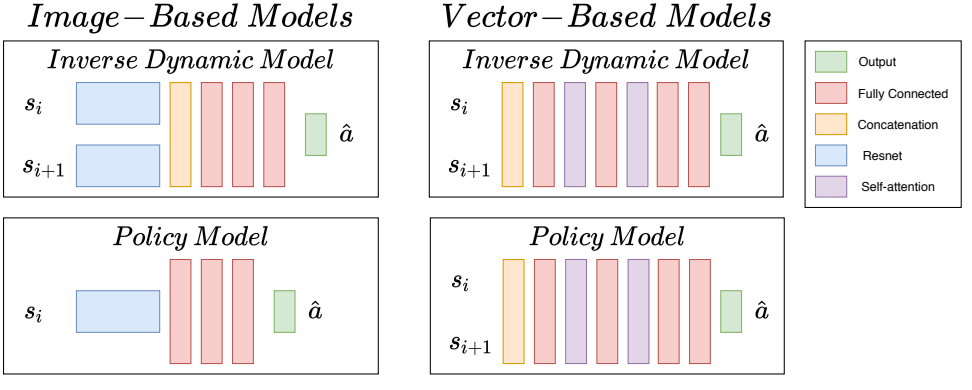

Figure 2: The Inverse Dynamics Model and the Policy Model representations for the image-based and vector-based environments. The concatenation layer for the image-based environments concatenates the visual embeddings from the ResNet encoder. For the vector-based environments, it concatenates the state representation.

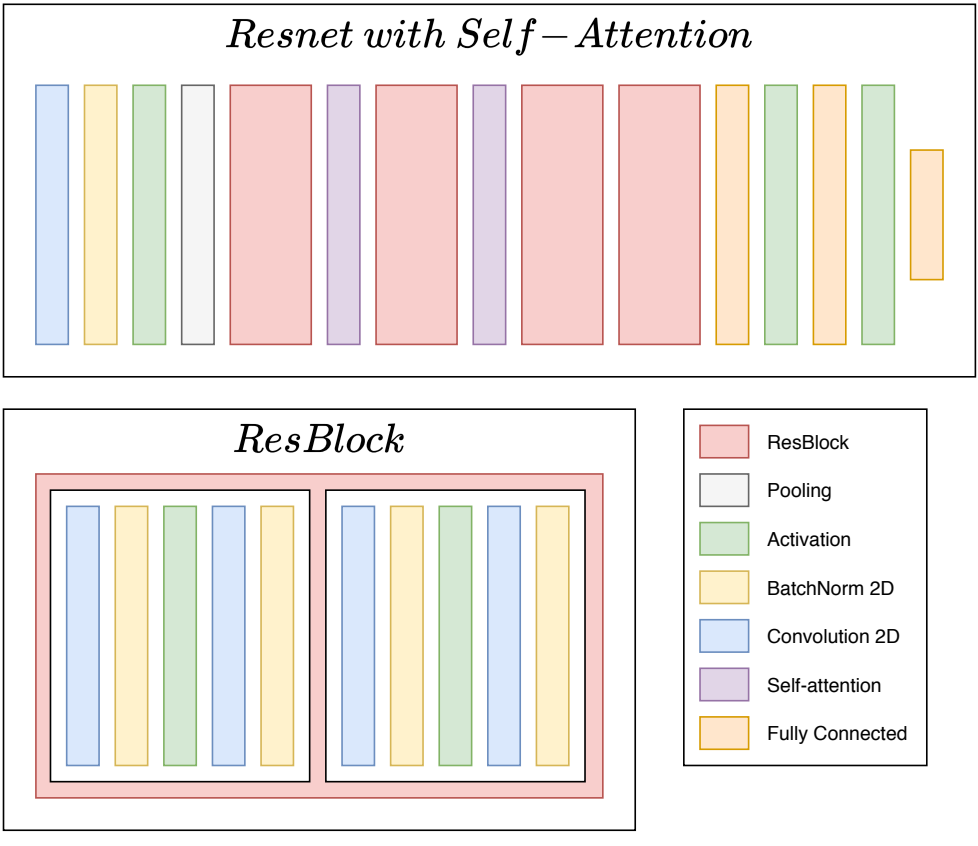

Figure 3: The ResNet model with self-attention modules. This implementation follows *PyTorch*'s version with four ResBlocks. We add two self-attention modules to capture global features from the image state representations.