**DataPredict: A General Machine, Deep, Reinforcement Learning Library For Lua-Native Games**

**Aqwam Harish Aiman**

**Abstract**

Machine learning, deep learning and reinforcement learning frameworks such as scikit-learn, TensorFlow, PyTorch and Stable Baselines allow researchers to train models using real-world datasets. However, there is a noticeable gap for game-related frameworks. This has led to difficulties in producing game-related machine learning, deep learning and reinforcement learning research as researchers are forced to integrate frameworks whose programming languages are incompatible with the game engines. Aiming at mimicking scikit-learn design, DataPredict offers a way of implementing machine learning, deep learning and reinforcement learning into Lua-native game engines like Roblox Studio.

**Introduction**

Machine learning, deep learning and reinforcement learning usage have become increasingly common in general real-world applications. Many frameworks, like scikit-learn and PyTorch, speed up the implementation of these models for general real-world applications. However, this focus has led to neglect in simulated worlds that mimic real-world interactions, such as games.

The gaming industry is estimated to be worth over $184 billion. This presents a significant opportunity for commercialising such models on a large scale. In addition, gaming platforms like Roblox that support user-generated content provide a significant amount of data that could be used to train these models. Although model training on this user-generated content has been done, very few have taken advantage of game-generated data to build in-game models that adapt to players in real-time. If done correctly, the in-game models could allow games to retain players much longer, earn higher revenue and have higher player-returning-power.

**Model Selection Criteria**

Currently, DataPredict offers over 50 machine learning, deep learning and reinforcement learning models. This includes unsupervised learning, deep reinforcement learning and so on. However, choosing the most appropriate models that could be used in games is difficult. This is because games produce incomplete data in real-time and may never produce a full dataset. Additionally, games generally require substantial computational resources to simulate physics and graphics. Heavy use of these models could result in a significant decrease in game performance and negatively impact players' experiences. As such, the models are generally selected based on these groups:

* Ability to perform online or incremental training.
* Sample efficiency
* Use cases
* Computational complexity

**General API Design**

DataPredict heavily relies on object-oriented programming architecture and uses both inheritance and compositional structure. This allows the sharing of extensions and utilities between different models, as well as having different configurations of these extensions and utilities by a single model. This flexibility allows the researchers and practitioners to adapt these models to their desired environments.

For example, DataPredict offers a way of having multiple deep reinforcement learning agents in different environments while having the same model using CategoricalPolicy and DiagonalGaussian quick setup objects. These allow the storing of individual environment states, actions and rewards for each agent without requiring this information to be stored in the deep reinforcement learning models themselves. This flexibility is further extended by having the deep reinforcement learning models act as a wrapper for a neural network model in the form of composition. This allows a single neural network to be shared with multiple deep reinforcement learning models without requiring the researchers and practitioners to write boilerplate code to access these deep reinforcement learning models’ functionalities. These deep reinforcement learning models mainly extend the neural network model’s update functions through the deep reinforcement learning models’ categoricalUpdate, diagonalGaussianUpdate and episodeUpdate functions that will be used by the CategoricalPolicy and DiagonalGaussian quick setup objects.

In addition, the library took extra precautions in performing inheritance. We ensured that when a class is inherited, we are confident that all modifications made in the extended class must affect the configuration of the inherited class. Another criterion for inheritance is that we want to ensure that the users of this library write minimal boilerplate code. One such example can be seen from the NeuralNetwork class, where it inherits from the IterativeMethodBaseModel class which in turn inherits from BaseModel class. In here, the BaseModel is responsible for handling model parameters initialisation and cost tracking. Meanwhile, the IterativeMethodBaseModel is responsible for tracking the number of iterations. Because the BaseModel’s cost tracking relies on IterativeMethodBaseModel, the IterativeMethodBaseModel must inherit the BaseModel as there is only one configuration possible from the BaseModel. As for the second criterion, one could agree that performing composition just to use NeuralNetwork model’s functionalities can end up producing too much boilerplate code without any clear benefits.