classification using deep rl code documentation

swakshar.sd

March 2020

1 Introduction

The documentation follows the paper "Classification with costly feature using deep reinforcement learning" [1].
This is a documentation on the code of classification with costly feature using deep reinforcement learning. I forked the code from this Github code
The code contains many modules, so I thought of writing documentation to ease the understanding of the code.

2 Parameters

There are some parameters which I find important:
DATA FILE = training file location
DATA VAL FILE = validation file location
META FILE = meta file location
CLASSES = number of classes = must start from 0
FEATURE DIM = number of features
ACTION DIM = feature dimension + number of classes
AGENTS = number of samples collected in one step
MAX MASK CONST = used for not considering any repeated action = 10^6

3 Some visualization

Before starting the function class, let's introduce ourself with some of the important variables.
Action space : classes dimension + feature dimension

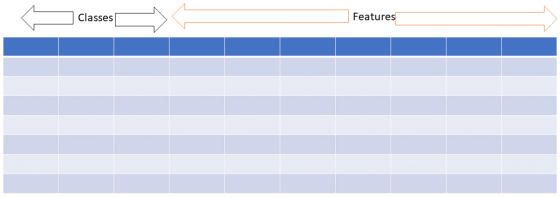

Figure 1: Action space

State space : feature dimension + mask dimension(mask dimension is same as feature dimension)

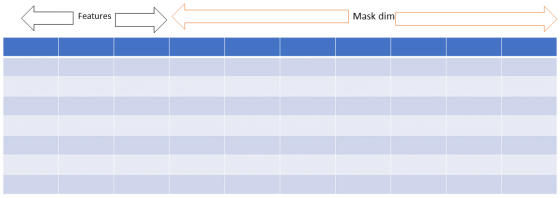

Figure 2: State space

Mask: use to keep track of whether a feature is present or not. Since we are replacing NaN features with zero, we should keep track of whether a feature value is zero or it is Nan.

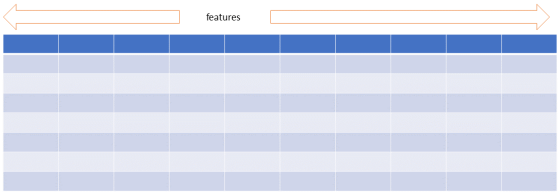

Figure 3: Mask

4 Functions

4.1 Main Functions

main.py = main code section for the whole program. Run this file to train the agent

4.2 Modules

4.2.1 Agent

store: Convert state, action, next state, reward into tensor and store them to replay memory via ***put****(fucntion inside pool class)*.
act: Return the Q-values into numpy array via ***`predict_np`****(function inside brain class)*.
step: Call the ***store****(function inside agent class)*.
`update_epsilon`: update epsilon.

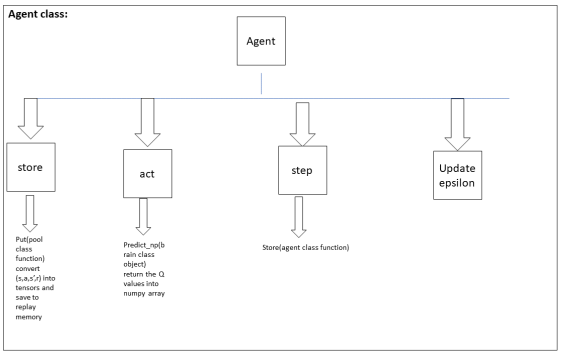

Figure 4: Agent class

4.2.2 Pool

First Initialize state, action, next state, reward tensors.
Put: store state, action, next state, reward into replay memory.
sample: sample random index from memory.
cuda: store state, action, next state, reward into GPU .

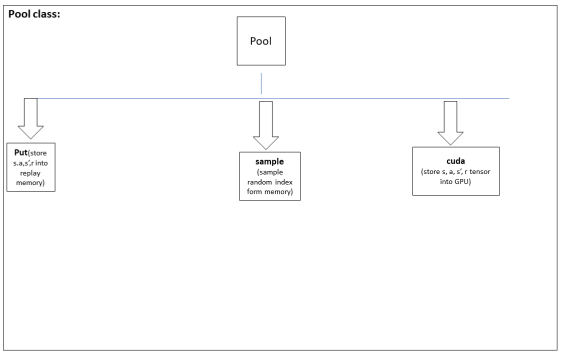

Figure 5: Pool class

4.2.3 Net

`copy_weight:` soft update of target network.
`train_network:` Calculate loss and backpropagate.
`set_lr:` learning rate scheduler.

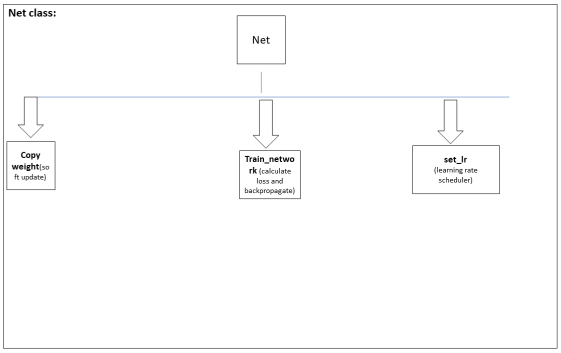

Figure 6: Net class

4.2.4 Brain

load: load the local and target network.
save: save the local and target network.
`predict_pt:` predict Q-values in form of tensor.
`predict_np:` convert predicted Q-values into numpy.
train: We use double Q learning to train the agent.

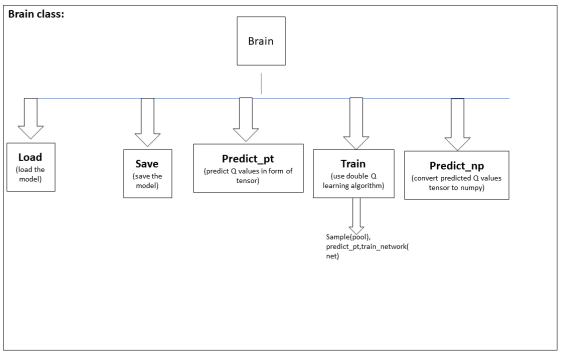

Figure 7: Brian class

4.2.5 Environment

reset: Initialize the input state to zeros.
`_reset:` randomly chose a data point.

`_generate_sample:` randomly chose a data point.
`_get_state:` Make an input state for the network.
step: Return next state and reward.

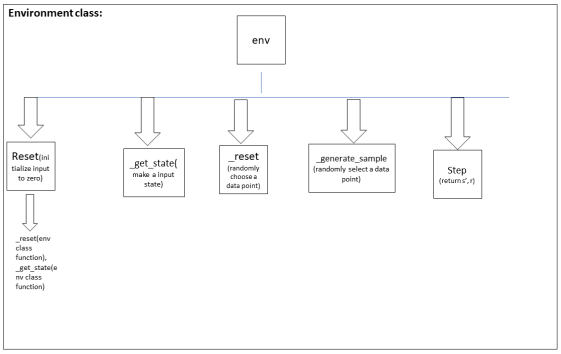

Figure 8: Net class

4.3 Some extra functions

`conv_filename.py`

- Load the dataset.
- Make train, validation, test file, and store those files.
- Make a 'meta' file. Meta file consist of mean, standard deviation, and cost of each feature.
- Store this meta file.

`eval_filename.py`

- For the testing purpose of the model.

Note that: In `'eval_filename.py'` file `'Predicted_labels'` variable contain all the labels predicted by the agent. `'data_y'` variable contain all the true labels.

References

[1] Jaromír Janisch, Tomáš Pevnỳ, and Viliam Lisỳ. Classification with costly features using deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3959–3966, 2019.