

# 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

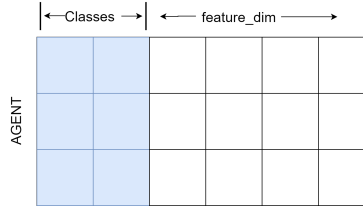


Figure 1: Action space

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

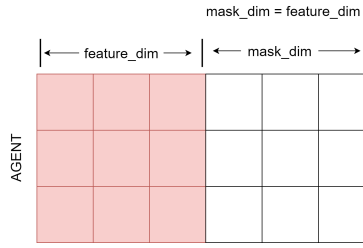


Figure 2: State space

**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.

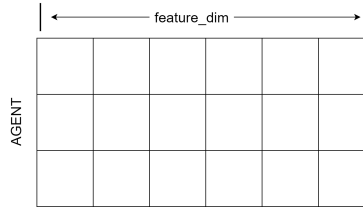


Figure 3: Mask

## 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

In agent class, we reset the state(initialize it to zero) and make environment and Brain object.

**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.

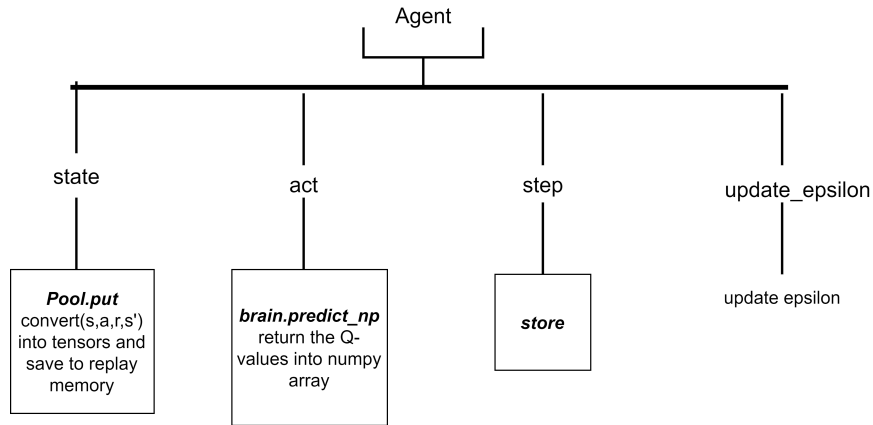


Figure 4: Agent class

### 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 .

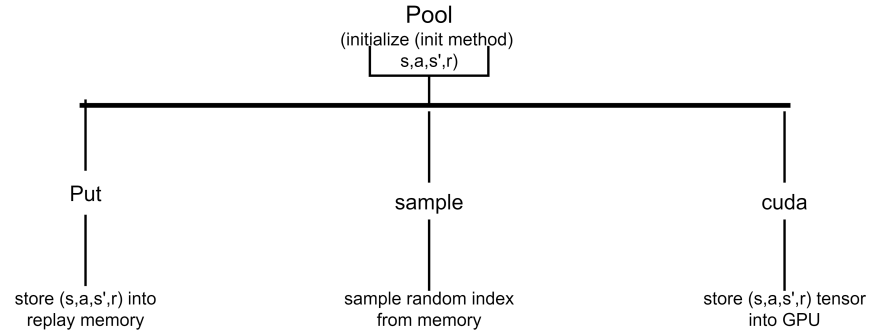


Figure 5: Pool class

### 4.2.3 Net

Build the neural network, calculate loss and backpropagate the loss.

**copy\_weight:** soft update of target network.

**train\_network:** Calculate loss and backpropagate.

**set\_lr:** learning rate scheduler.

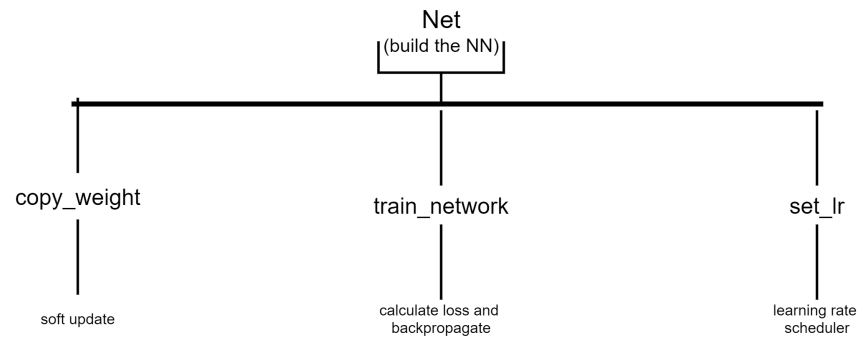


Figure 6: Net class

#### 4.2.4 Brain

Initialize the target and local network.

**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.

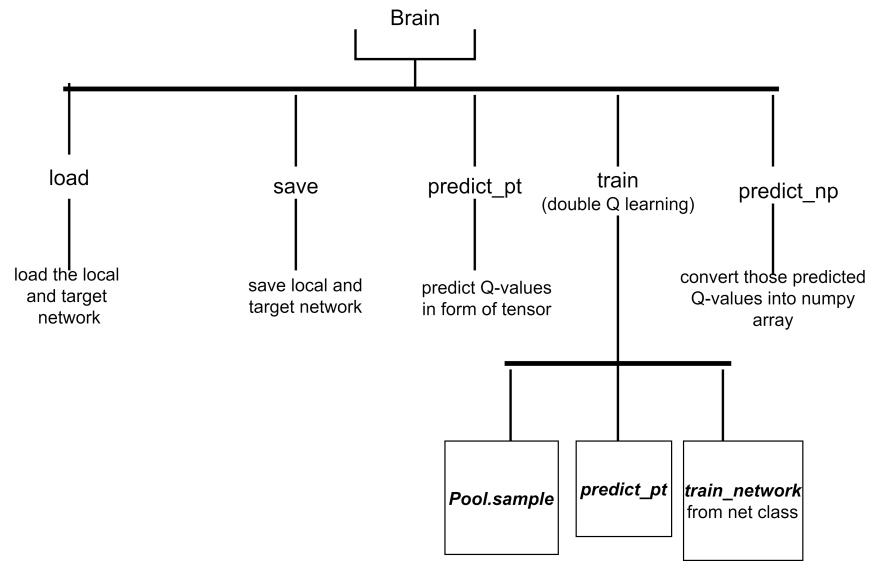


Figure 7: Brian class

#### 4.2.5 Environment

Inside Environment class we first perform those things:

- Get the whole dataset containing features and labels
- Set the cost of each feature and make an cost array
- Initialize the mask to zero
- Initialize an array that contain a batch of features

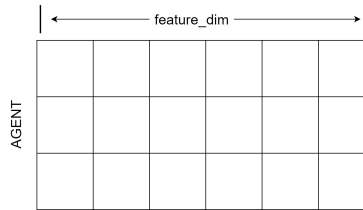


Figure 8: contain a batch of features

- Initialize an array that contain a batch of labels for those features.

**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.

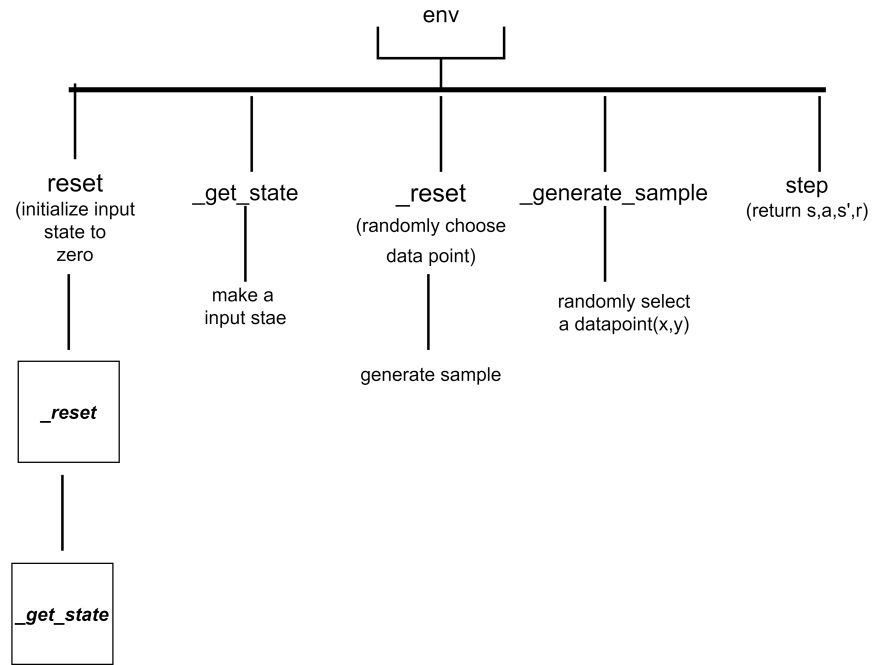


Figure 9: Environment class

### 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.

## 5 Usage:

- store the dataset into data/raw direactory.

- use tools *conv\*.py* to prepare datasets (tips: if the dataset is csv file, copy paste the *conv\_stroke.py* and change the file name. If it is .data file, copy paste the *conv\_miniboone.py* and change the file name). After running this *conv\*.py* file train, test, validation, meta data will be created inside data folder.
- select a dataset to use and copy corresponding file from consts-template to *const.py* (tips: if dataset is .csv copy paste the *const – stroke.py* file and modify it as you like. If dataset is .data copy paste the *const – mb.py* file and modify it as you like.
- run *main.py*
- the run will create multiple log files
- you can use octave or matlab to analyze them with tools *debug.m*
- you can also evaluate the agent on the test set with tools *eval\*.py*

## References

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