Multi-objective Reinforcement learning (MORL)

Toy Tasks

The paper used 4 tasks to evaluate the proposed approach, with FTN and DST being simpler that the other two:

- Fruit tree navigation task (FTN)
- Deep sea treasure (DST)
- Dialog
- Supermario

Supermario takes a long time to run fully, about a month for 32000 episodes, and the authors had used a cluster of 2080 GPUs so I did not work on it for now. The other task, called the fruit navigation task was simpler and took about 5 hrs to run on google colab. The task is composed of a tree with depth of 5, 6 or 7 (we can choose between them when initializing the tree.). The nodes of the tree are supposed to contain the multi-objective reward for each action taken, which is moving to the right or left branch on in each node. However, in this tree the nodes contain zero reward. The only reward is on the leaves of the tree. Each node on the tree has two branch, and the direction is decided by a DNN. At every run when testing the system, the tree picks a random vector of weights. Then the tree is supposed to guide us to a leave that would result in the highest *w.r*. This means that the DNN is supposed to learn how to give us the best trajectory when it was given a random weight. Therefore, each trajectory would be dependent on the random weight. The picture of the tree is as follows (From [1] appendix):

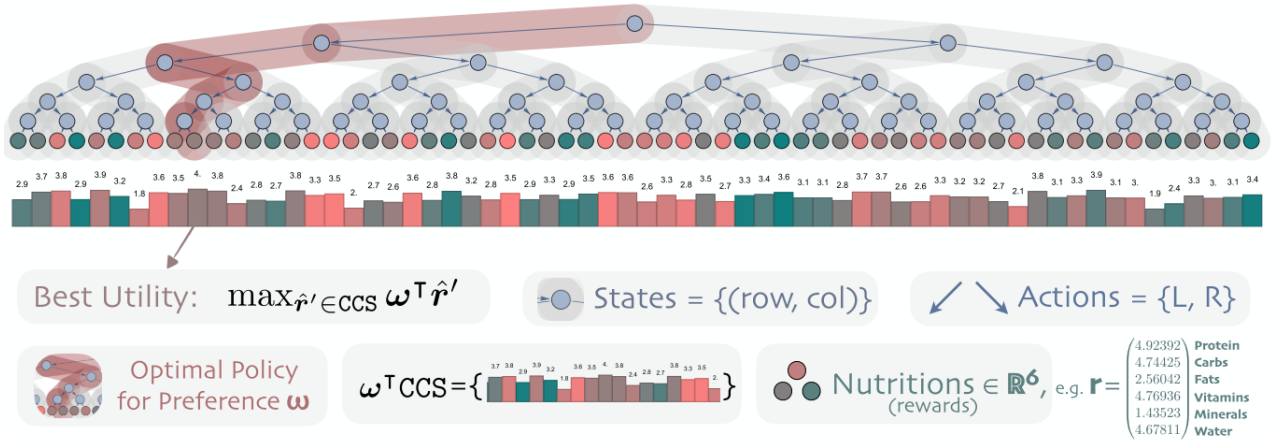

Figure 1: Fruit Tree Navigation (FTN): An agent travels from the root node to one of the leaf node to pick a fruit according to a post-assigned preference *w* on the components of nutrition, treated as different objectives. The observation of an agent is its current coordinates (row, col), and its valid actions are moving to the left or the right subtree.

The training is illustrated in Fig. 2. The process uses double Q learning, in which two models are defined. The reason for this is that, when training the Q network in the deep Q learning method, we use the bellman equation to update the model, and the bellman equation itself uses the model output. It seems like we are chasing to learn a model that is changing itself. So, two models are defined, one target model and one model. The target model only updated after certain episodes, but the model changes and learns using the loss function. The novelty in this paper is basically that, it tries to find the best trajectory regardless of the weight chosen at test time.

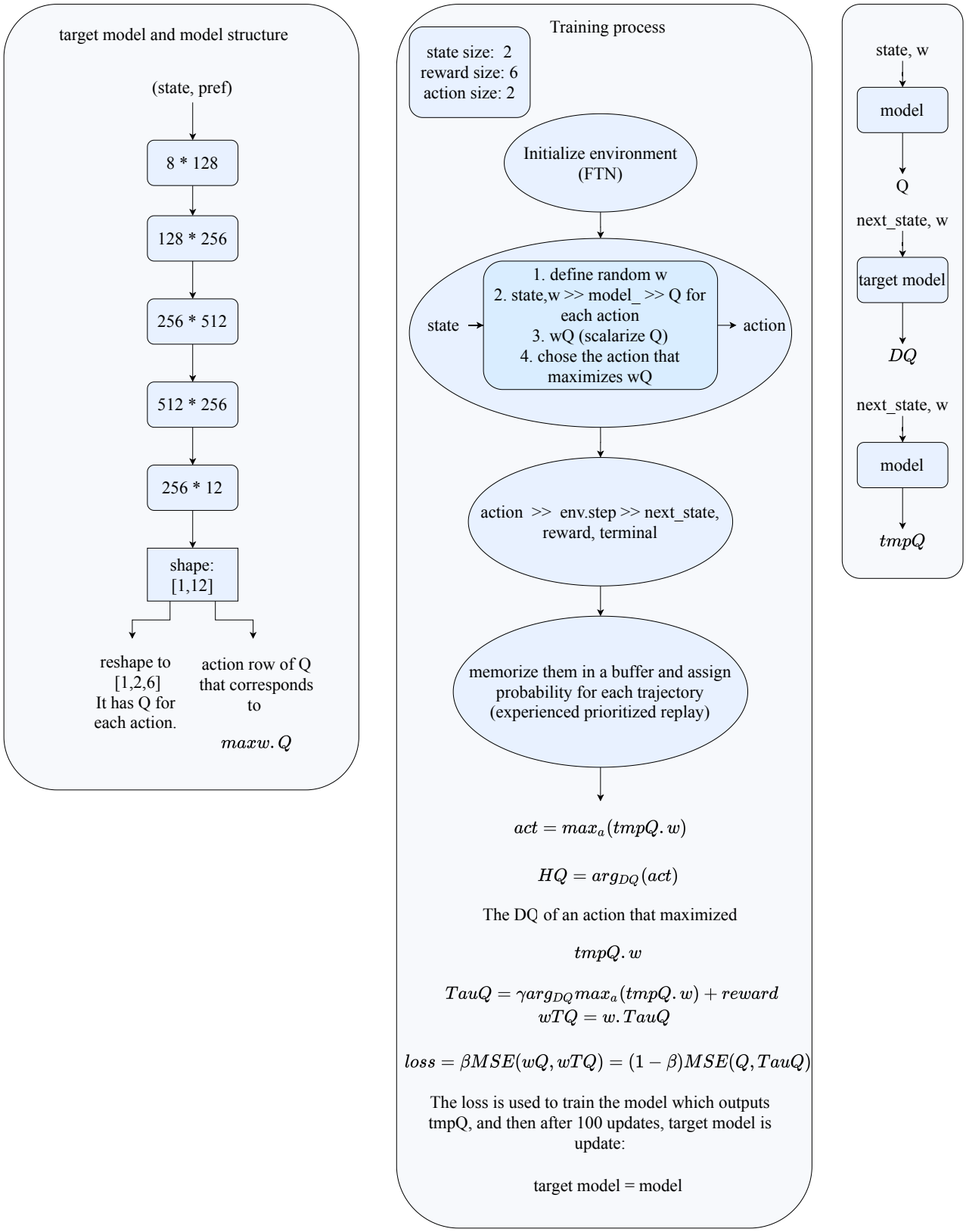

Figure 2: The whole process of training MORL.

IMPLEMENTING THE PROPOSED APPROACH

As we talked about this in our last meeting, it might be beneficial to let the weights change with a trend like the approach taken in [2]. In this paper, there are coefficients for each segment of the loss function.

$$\alpha_k^i = \frac{e^{\beta s_k^i}}{\sum_{l=1}^{n} e^{\beta s_k^i}} \tag{1}$$

where s_k^i is the difference between the loss component in the iterations i and $i-1$. when $\beta > 0$ the algorithm focuses on improving the worst component of the loss (the part of the loss function that is increasing.) and vice versa. But here, the advantage of using random preferences is that it is random. and it gives the whole process sort of a freedom to chose any weight and get the best result with that weight. However, we can change the mean of the each component of the random weight. In the implementation of the paper, the weights are generated with 0 mean and unit variance normal random function. But if we save the loss and make a list of losses in an inner loop of the process, we can modify the function to define the mean of the random function that is generating the weights. So for example, if the β is negative, the random function that is generating the weights for each objective would be likely to assign a bigger number to the objective that had the least reward. Then the mean is saved after training to be used at test time. In the main approach of the paper, a weight is generated with zero mean but here we use the mean from the last episode of training. The proposed approach is explained in Fig. 3 more thoroughly.The red text is the added parts to the approach in the paper.

EXPERIMENTS

The paper uses two evaluation metric. Coverage ration (F1) and Adaptive error. F1 measure how well the trajectories are spread. For instance, when changing the weights, do we get another leaf in the tree or do we end up in a few popular leaves? Adaptive error also finds the error between the *w.r* of the leaf that was chosen in the test process and the highest *w.r* from the tree. So it tries to find out if we got to the leaf that would result in the largest value after multiplying the weight in the multi-objective reward.

The loss functions for both approaches are depicted in figure Fig. 4 and Fig. 5. The loss is noticeably lower in our proposed approach in the last episodes. However, there is a dip in the loss value around episode number 3000. I will run this again for 5 times to average the results and see if it happens again. The loss value is also highly unstable at the beginning when looking at the prposed approach but it gets better at the end compared to MORL.

REFERENCES

[1] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 636–14 647.

[2] A. A. Heydari, C. A. Thompson, and A. Mehmood, "Softadapt: Techniques for adaptive loss weighting of neural networks with multi-part loss functions," *arXiv preprint arXiv:1912.12355*, 2019.

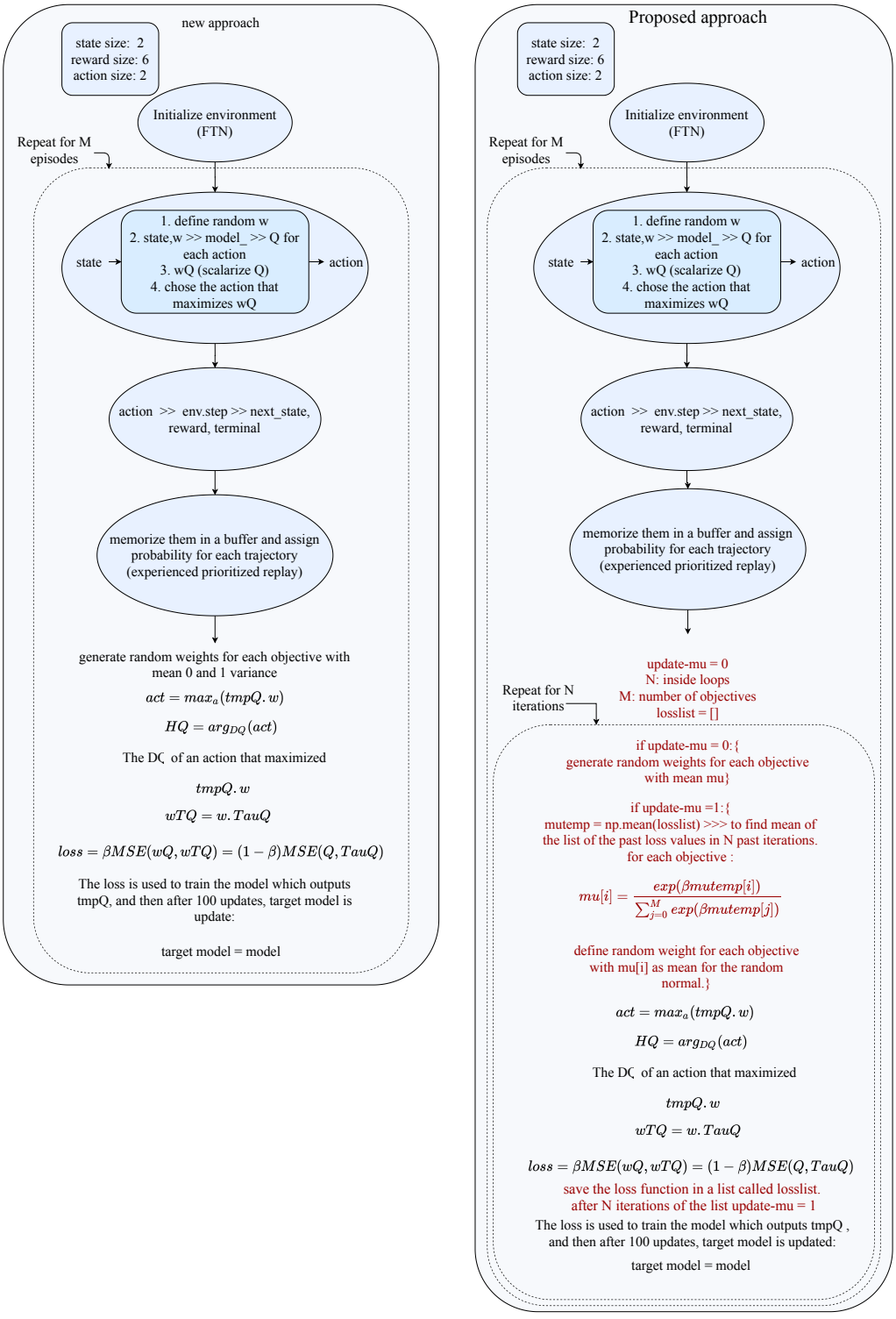

Figure 3: The whole process of training MORL.

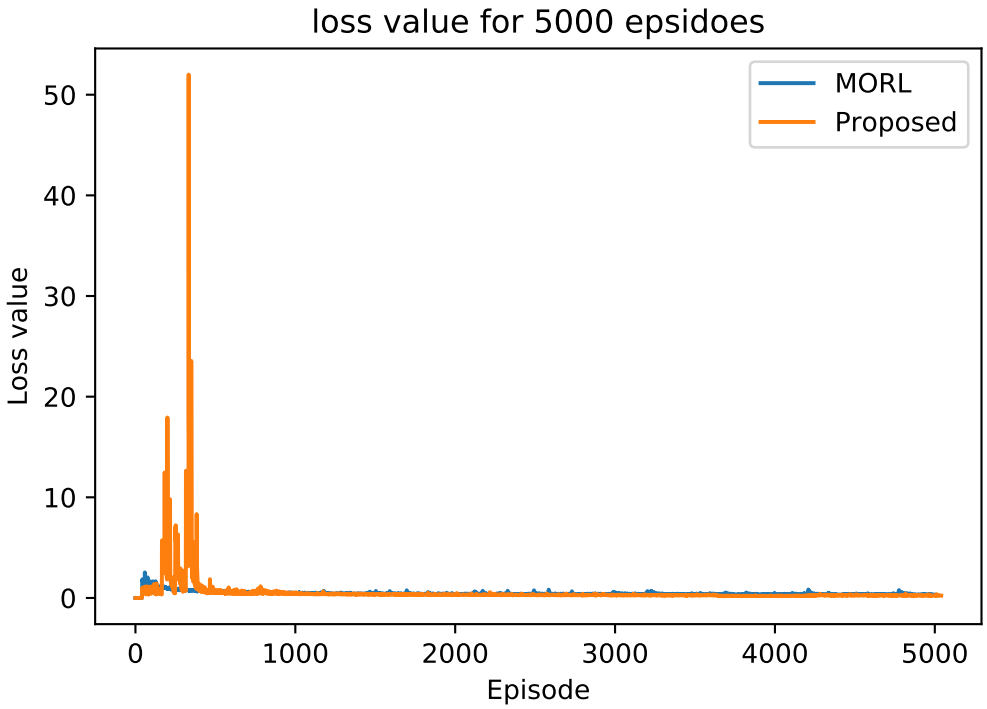

Figure 4: Loss value of the MORL and proposed approach in 5000 episodes of training.

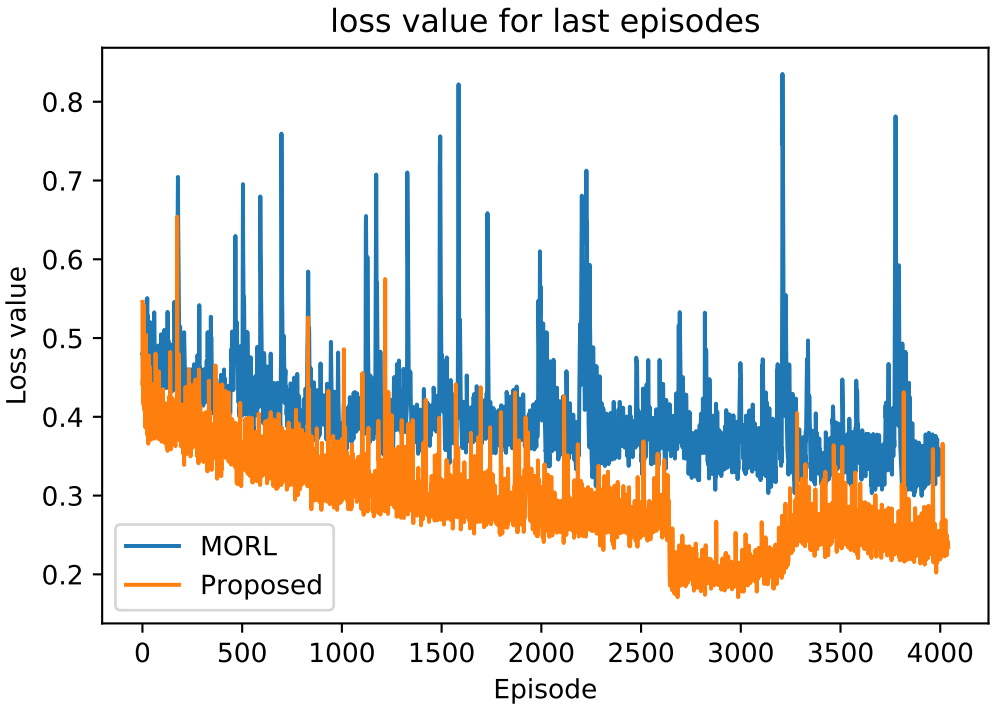

Figure 5: Loss value of the MORL and proposed approach in the last 1000 episodes of training.