Football Learning With DQN, APE-X, A3C, and PPO

CS 7642 – Reinforcement Learning and Decision Making

Hash: add80ab8a6de6f310847a8551a3a7ab7a7e7c5bc

Abstract—**In this project, I implement DQN, Ape-X, A3C and Reward Shaping by using Ray's Rllib library; and compare and contrast them with the already implement 3 PPO agents.**

I. Introduction (*Heading 1*)

So far, we have studied reinforcement learning problems where only one agent is moving, and the environment is fixed. With multiagent reinforcement learning, however, several agents are trying to maximize their rewards. For example, a football game, consists of two teams where both teams are trying to win in a minimax environment, which is similar to game theory. Further, within the team, all the team players are collaborating for the team to win. Therefore, the state space needs to include all these agents' locations.

There are many algorithms to tackle this problem. However, in this project, we will only look at Deep Q-Networks (DQN), Distributed Prioritized Experience Replay (APE-X), Proximal Policy Optimization (PPO), Actor-Critic models (A3C) [1].

A. Deep Q-Networks

DQN's, as discussed in Project 2, enabled combining reinforcement learning with function approximation. DQN minimizes the following objective function by using any gradient descent method like ADAM or RMSProp.

$$J = \mathrm{E}\left[\left(Q_\pi(\boldsymbol{s},\boldsymbol{a}) - \hat{Q}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{w})\right)^2\right] \quad (1)$$

Before, neural networks were not suggested for reinforcement learning because of stability issues and convergence was not guaranteed. However, Mnih *et. al.* [2,3] introduced the following two techniques to stabilize DQN's. First, replay buffer replay is used in order to avoid divergence. Replay buffer stores possibly 100000's of previous experiences and the trained experiences are picked randomly from the replay buffer. This way, we do not focus only on the last experiences. Second, Q_π is the fixed policy and $\hat{Q}$ is the target policy. Fixed policy has no derivatives with respect to weight. The fixed policy is updated every few steps to the target policy. These two novelties bring some stability to DQN's.

B. Actor-Critic Models

Mnih *et. al.* [4] argue that replay buffers can only be implemented to off-policy algorithms such as Q-learning and they are expensive in terms of memory and computation. They propose another method where they "asynchronously execute different agents" at different instances of the environment. This approach stabilizes the algorithm without taking too much memory space such as the replay buffer. They note that this method can also be implemented to on-policy learning algorithms such as Sarsa.

While DQN is a value-based model-free algorithm, there are policy based algorithms which use the policy itself in the optimization function. Actor critic models have both the value function term and the policy term in the optimization function. A3C [4] which is an actor critic model uses the following objective function.

$$J = log\pi(\boldsymbol{a}_t|\boldsymbol{s}_t;\boldsymbol{\theta}')\left(R_t - b_t(\boldsymbol{s}_t) + \beta H\left(\pi(\boldsymbol{s}_t;\boldsymbol{\theta}')\right)\right) \quad (2)$$

where π is the policy. R_t is an estimate of $Q^\pi(a_t, s_t)$ and b_t is an estimate of $V^\pi(\boldsymbol{s}_t)$, β is the entropy coefficient and H is the entropy regularization term. Here, policy itself would be the actor and b_t would be the critic. While adding the stability term improves the stability, entropy regularization term enhances exploration.

C. Proximal Policy Optimization

PPO [5] is a policy-based model-free function approximation algorithm which is uses the following objective function:

$$J = E[L_t^{CLIP}(\boldsymbol{\theta}) - c_1 L_t^{VF}(\boldsymbol{\theta}) + \beta H(\pi, \boldsymbol{s}_t)] \quad (3)$$

where $L_t^{VF}(\boldsymbol{\theta})$ is Eq (1) for one experience only, c_1 is a coefficient, and $L_t^{CLIP}(\boldsymbol{\theta})$ is another objective function from another method called clipped surrogate objective. The authors note that this method is stable, simple and fast.

In this project, three already trained PPO agents were given. The hyperparameters used for the agents can be found at Table 1. These agents as well as the ones I have implemented are implemented on a 3×3 football environment developed by google research. The purpose of this project is to implement reinforcement learning agents as well as reward shaping to train the football environment and beat the PPO baselines that were given.

D. Distributed Prioritized Experience Play (Ape-X)

Horgan *et. al.* [9] combine multi-step bootstrapping, double Q-learning and DQN and call it Ape-X DQN. They estimate the value of a state as

$$G_t = \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n Q\left(S_{t+n}, argmax_a Q\left(S_{t+n,} a, \theta\right), \theta^-\right) \quad (4)$$

where the summation term is n-step bootstrapping. Please note Q-learning only uses one-step bootstrapping. However, this method increases the number of steps for the information to be carried faster. The second term uses a double-Q-learning term where first Q value after n-steps is calculated for all action scenarios by using the target Q-network. The action value that gives the largest Q value is taken as the action. That action is taken into the fixed Q-network along with S_{t+n}. Estimated G value is fed into the following loss function.

$$L_t = \frac{1}{2}\left(G_t - Q(S_t, a, \theta)\right)^2 \tag{5}$$

The authors also improve parallelization and random selection from buffer replay. This results in improvement in computational time and convergence behavior.

II. ENVIROMENT

The environment itself is a 3×3 wrapper of the google football problem [6,7]. Typically, there are 11 players from each team, but the game is simplified to 3 players from each team. For each team, one player is the goalkeeper who is controlled by the internal AI agent. Therefore, there are four controls: two from our team, and two from the other team. Our goal is to maximize the goals of the team on the left.

While the original environment has 115 continuous and discrete states, the simplified version has 43 states which are the x, y locations of 6 separate players (12 continuous states), 12 x, y rotations for 6 players, 3-dimensional ball position and ball direction, ball possession (discrete), active player (discrete), 7 game modes such as penalty. The game ends when one side wins, within 500 timesteps or when the ball goes out of bounds.

The difference between this football environment and the other football environments is that this one focuses on the high-level actions such as kicking the ball or running. There are 19 actions which are running in 8 different directions, 4 difference passing/shooting options, sprinting, resetting player direction, stopping sprinting, sliding, dribbling, and stopping dribbling. The action space is discrete and stochastic.

The authors [6] implement PPO, IMPALA and Ape-X DQN algorithms. IMPALA with 500,000,000 episodes performs the best. I tried to implement IMPALA. However, I think rllib's IMPALA agent is not plugging well with the google football environment because it was not able to recognize the policy that I set. It raised the following error: "PolicyID 'default_policy' not found in this PolicyMap!"

III. DIFFICULTIES

I had many difficulties in setting up the project that were related to my lack of knowledge of Docker, Linux and Google Cloud. I initially tried the Docker setup. I had to setup Docker at the least two times because the first time I ran out of storage. I rebooted the laptop to create space. I learned Docker and successfully setup the environment. However, the run failed right at the beginning because of lack of memory. As my memory was only 8 GB, I decided to try Google Cloud while waiting for a new 16 GB ram order.

I set up the google cloud environment with the help of the fellow student Jacob Williams. Creating a Google Cloud environment includes opening an account, creating a VM instance, creating a Docker inside the VM that can also forward port which helps to see Tensorboard on http://localhost:6006/. Setting up the environment in Google Cloud was a great idea as I took advantage of their large memory and storage.

I was never able to setup a gpu. I initially started with 8 cpu's. My initial run of the baseline 0 completed in almost two days. Later on, I deleted the VM and created a new one with 16 cpus. The problem is userid and groupid of the new VM were different. Therefore, it took quite a while to set up the Docker. Eventually, I added another VM with 60 cpu's.

I wanted to be able to upload and download data between the google cloud and my local computer. It took about half a day to figure that out. I tried to connect Pycharm to Google Cloud which was not available. It is possible to connect Vscode to Google Cloud. However, I was not able to overcome the permission issues. I decided to use ipdb based on community suggestion for debugging purposes. I figured ipdb is not compatible with some of the python libraries and hence I switched to pdb which worked very well. Vim was very handy in modifying the files in the VM.

I tried running IMPALA, QMIX and A3C at first. However, I had a lot of errors. Some of those errors had to do with missing or unrecognized attributes because each attribute is agent specific. Other errors were bugs in rllib.

I have identified a bug in A3C. One object in rollout_worker.py was checking an attribute that was not defined, and it was throwing an error based on that. I made the following correction at /opt/conda/lib/python3.7/site-packages/ray/rllib/evaluation/rollout_worker.py. The correction is that I defined an attribute self.tf_sess and initialized it to None. This way A3C started to run. I also took the course in this reference [8] in order to get familiar with rllib.

IV. IMPLEMENTATION

A. *PPO*

PPO was already implemented and the hyperparameters were available at params.json. Three baselines were given as shown in Table 1.

I would like to touch on some of the parameters shown in the table. Batch size is 2800 for each baseline with rollout fragment length 100. A rollout worker collects 100 episodes of the football environment 28 times. This makes a batch. The batch is trained with the stochastic gradient descent by using minibatches of 128. While PPO uses minibatches, A3C and DQN do not. The entropy term is an additional term in the objective function that enhances exploration. Therefore, a higher entropy coefficient enhances exploration. The framework for PPO was torch, but Tensorflow option is available as well. Gamma is a term between 0 and 1 that punishes future rewards for being too far away. The lower the gamma, the lower the value of the reward that is too many steps away. Horizon finishes the episode at maximum 500 environment steps. Clip parameter determines how far the new policy can go from the old. KL is Kullback-Leibler divergence that shows the surprise when old and new policies are compared. Learning rate is the alpha value that multiplies the step to avoid big changes in theta values. VF loss coefficient is c_1 at Eq (3).

The above parameters were related to the calculation of the objective function or the step size. We also need to determine a model. Unless otherwise stated, the default model is dense layer model with two layers with 256 neurons each. It is also possible to add an LSTM layer which serves the purpose of memory. Activation function between the layers can be tanh or relu. However, the final activation function is relu and the output is one-hot vector of size 19.

Policy maps the states into actions. Policy consists of state vectors. Policy can be shared or separate for each control.

Table 1. Hyperparameters for the PPO baselines.

Baseline #	1	2	3
batch_mode	truncate episodes	truncate episodes	truncate episodes
clip_param	0.2	0.21	0.19
entropy coeff	0.004	0.0056	0.008
framework	torch	torch	torch
gamma	0.99	0.993	0.996
horizon	500	500	500
ignore worker failures	false	false	false
kl_coeff	1	0.98	1
kl_target	0.01	0.035	0.018
lambda	0.95	0.93	0.94
learning rate	0.0002	0.0001	0.0001
layers	2	2	2
neurons	256+256	128+256	256+128
activation function	tanh	relu	relu
lstm cell size	256	256	-
vf_clip_param	1.49	0.54	2.36
vf_loss_coeff	0.35	0.73	0.98
policy	separate	separate	separate

B. *Reward Shaping*

After failing with hyperparameter tuning for several agents such as A3C, I decided to modify the reward system. Originally, the player in possession of the ball is rewarded 0.1 for every one of the checkpoints. Therefore, it is possible to gain 1 point by competing all the checkpoints. The additional goal will complete the rewards to 2 points independent of how many checkpoints are passed.

I have modified the system such that there are 50 checkpoints and every checkpoint is worth 0.02 points. Therefore, the ultimate reward is still 2 points, but the rewards are more continuous. This would teach the agent to learn easier as the rewards are distributed more uniformly.

I have made the changes at wrappers.py which can be found at:

/opt/conda/lib/python3.7/site-packages/gfootball/env/

The changes are:

```
self._num_checkpoints = 50
self._checkpoint_reward = 1./self._num_checkpoints
```

Figure 1 shows the reward mean curve for baseline 1 before and after reward shaping. PPO+RS shows the reward mean curve after reward shaping and it shows better training performance.

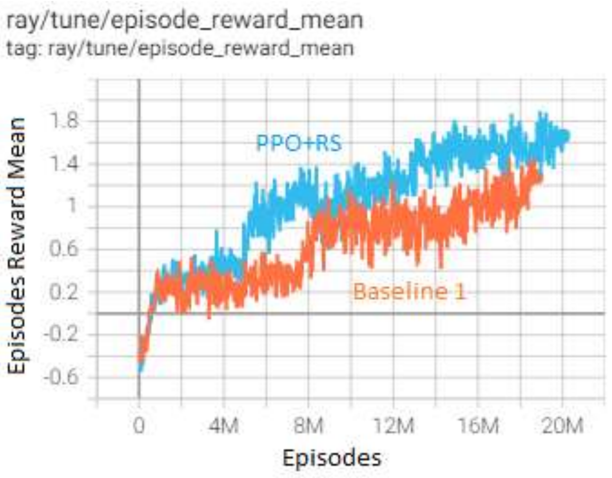

Figure 1. Episodes vs Reward Mean for Baseline 1 training and Baseline 1 training after Reward Shaping. Both of them are using PPO.

C. *A3C*

I spent several days trying to identify hyperparameters that will beat the baseline for the A3C agent. Based on experimentation, I decided A3C doesn't work well with LSTM. I used the rllib tune option to optimize the hyperparameters. Configurations for the tuning are as follows: Learning rate varies between [0.00005,0.0005], gamma varies between [0.99,0.999], lambda is randomly picked from a normal distribution with 0.95 mean, 0.02 std and ±0.05 clipping, value function loss coefficient varies between [0.1,1.0], entropy coefficient varies between [0, 0.05], gradient clip varies between [10, 100], the policy can be shared or separate. Model layers are dense only and vary between 2-3 layers and 128-256 neurons. Also, I used the Tensorflow framework instead of Torch as Torch was failing at the beginning.

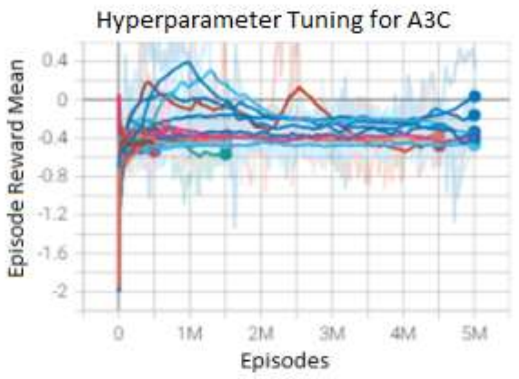

Figure 2. Episodes vs Episode Reward Mean for 100 A3C agents.

Figure 2 shows the episode reward for 100 different configurations many of which were terminated by the tuner as they were not promising. Most of the reward mean seems to be around -0.4. It means we are losing more than we win. A3C runs were much faster than PPO or DQN. One reason for that is they don't use replay buffers. However, unfortunately, I was not able to get a configuration that can learn the game.

D. DQN

As I am more familiar with DQN, I decided to tune DQN, I created a tune experiment with 50 runs. The configuration for the tuned hyperparameters is very similar to that of A3C except number of layers vary between 1-3, and number of neurons vary between 64-512. There are only dense layers. Activation function is tanh.

Figure 3 shows episode reward mean for 50+ DQN runs during training. Similar to A3C, DQN is not able to teach players to learn football.

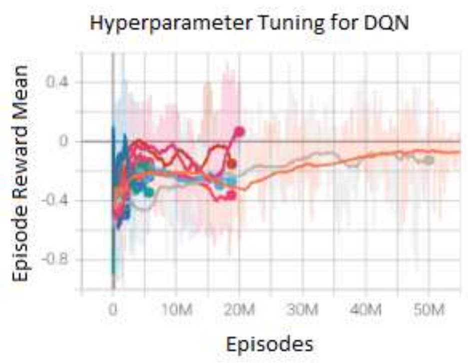

Figure 3. Episodes vs Episode Reward Mean for 50+ DQN runs.

E. APE-X

I tried about 20 runs with FIFO scheduler where each run completes 18,500,000 episodes. Ape-X performs better than DQN and A3C. However, it still does not reach the performance of PPO. Figure 4 shows that one of the Ape-X configurations is able to reach 0.6 mean reward which is an improvement compared to DQN and A3C.

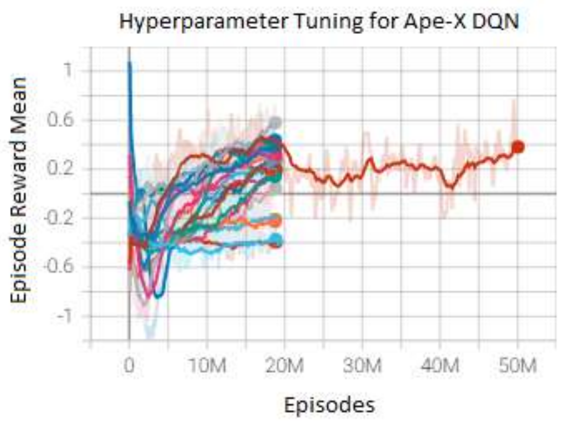

Figure 4. Episodes vs Episode Reward Mean for 20+ Ape-X DQN runs.

V. Results

I carry out 10 simulations for every of the 7 agents including Baseline 1, Baseline 2, Baseline 3, Reward Shaping, Ape-X, DQN and A3C. Some of the performance plots are shown in Figure 5.

PPO+RS (Reward Shaping) shows similar performance to the baselines, which are PPO themselves. PPO+RS is not defeated for the 10 simulations same as the baselines. I have

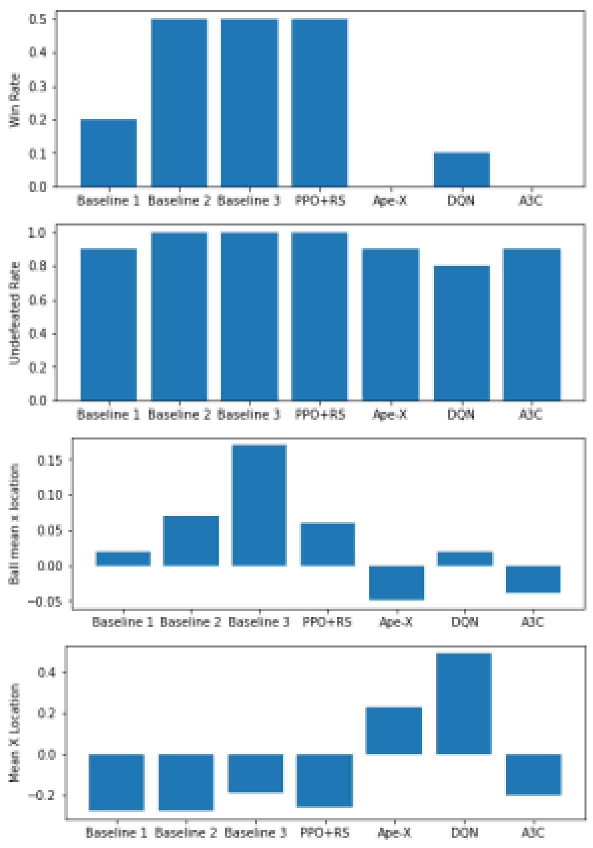

Figure 5. Seven Agents vs Win rate, Undefeated Rate, Mean Ball X Location and Mean Player X Location.

noticed that left players are able to score a goal if the ball is initially on the left team. However, if the ball is initially on the other side, they are not able to figure out what to do. I guess a reward shaping based on which team owns the ball would be helpful for training purposes. Further, PPO+RS showed better training episode rewards. However, this did not show an improvement for the win rate during testing. This is probably because players are able to collect points when they run to the right even when they shouldn't.

All of PPO+RS, Ape-X, DQN and A3C use reward shaping which means the players should be more likely to run to the right. Mean X location shows the mean of the X location of the two left team players throughout the episode for 10 episodes. It can be seen that Ape-X and DQN football players are more likely to be on the right half of the stadium. Unfortunately, this is not correlated with the ball location. I also plot the mean ball X location throughout the episode for 10 episodes. Basically, I am averaging the X location of the ball for all the observations. Ball X location is not correlated with the Player X location. However, ball X location is positively correlated with the win rates. Baselines and reward shaping agents keep the ball mostly on the right side.

Even though Ape-X, DQN and A3C have very low to zero win rates, they have quite high rates of undefeated rates. I think for these cases, the players run to the right even if they don't have the ball and they are not able to score i.e. they do not capture the required action to score a goal.

Further, even though the neural network model and some other parameters are quite similar between these agents, the objective function and sampling replay are very important to teach them the rules of the game.

Even though Ape-X had shown better performance than DQN and A3C during training, it turns its win rate is the worst.

Even though A3C is very fast as it doesn't have buffer replay, unfortunately it not able to learn the rules of the game.

VI. Conclusion

I have implemented reward shaping (RS), Ape-X, DQN and A3C.

Reward shaping is able to better teach the players to run to the right. PPO+RS shows similar to better win rate compared to the baselines. Other agents such as Ape-X, DQN and A3C do not perform as good as the baselines.

Reward shaping teaches all the players on the left team to run to the right even if they do not have the ball. If the ball starts with the left team at the beginning of the game, baselines and PPO+RS agent score a goal. Otherwise, they are not able to get the ball from the other team.

A3C is the fastest for training because it does not have any replay buffer. Ape-X performs best in training compared to DQN and A3C. However, it performs the worst in testing.

I noticed that the football environment of Google Research is not very compatible with Rllib. Therefore, I spent quite a bit of time with debugging.

In the future, I should probably pick more advanced RL agents in order to improve performance. I have selected Ape-X as this was used by the Google Research team itself and they had succeeded with this algorithm. I also saw online A3C is used for training other football environments. The reason I chose DQN is because I am already familiar with it from Project 2. I was not able to run IMPALA as I think there were bugs in Rllib. I also tried MARWIL but it was not able to learn the game.

REFERENCES

[1] https://docs.ray.io/en/releases-1.6.0/rllib-algorithms.html?highlight=a3c#impala

[2] Mnih, V., Kavukcuoglu, K. et. al. Playing Atari with Deep Reinforcement Learning. Deepmind Technologies, 2013.

[3] Mnih, V., Kavukcuoglu, K. et. al. Human-Level Control Through Reinforcement Learning. Nature vol. 518, pp. 529-533, 2015.

[4] Mnih, V., Badia, A. P. et. al. Asyncronous Methods for Deep Reinforcement Learning. Proceedings of The 33rd International Conference on Machine Learning. 2016.

[5] Schulman, J., Wolski, F. et. al. Proximal Policy Optimization Algorithms. Machine Learning. arXiv:1707.06347. 2017.

[6] Kurach, K., Raichuk, A. et. al. Google Research Football: A Novel Reinforcement Learning Environment. Proceeding of the Thirty-Fourth AAAI Conference on Artificial Intelligence. 2020.

[7] https://github.com/google-research/football

[8] https://courses.dibya.online/p/fastdeeprl

[9] Horgan, D., Quan J. et. al. Distributed Prioritized Experience Play. Proceedings of the Sixth International Conference on Learning Representations. 2018.