Jan Ramos

ID: 903758793

Githash: b8f6d736fb17fe1aea04665f789934a1e4e862a8

Football

The aim of this project is to outperform three baseline multi-agent reinforcement algorithms by selecting and tuning an optimal algorithm. We will be conducting our training on a modified version of the "Football" multi-agent environment. developed by Google Research. Thus far in the course, we have focused primarily on single agent algorithms, the most notable being the DQN algorithm we used in the Lunar Lander environment from our previous project. In this paper, we extend these principles onto our multi-agent environment by examining two policy gradient methods.

Multi-agent reinforcement learning algorithms focus on training multiple learning agents in the same environment. In the case of the "Football" environment, the shared environment is principled on collaboration as our agents are playing to win as a team. Like in single-agent reinforcement learning, multi-agent problems are expressed using Markov decision processes. The key distinction between single-agent learning and multi-agent learning is that the goal of single-agent learning is to maximize the reward structure for one agent. Since multi-agent learning involves multiple agents, there are underlying sociological behaviors at play. Multi-agent environments can be grouped into three settings: competitive, collaborative, and mixed. The "Football" environment falls under a mixed strategy as we have collaboration within our team and our facing an opposing team in competition/training. In other words, our environment is a zero-sum game, where one team's victory is another's loss.

As stated, we are training our multi-agent reinforcement learning algorithms using a modified version of the "Football" environment. Google Research Football was developed by Google in 2019 as a reinforcement learning training environment. The goal of this environment is to train our team to maximize its chances of scoring and minimize its chances of conceding a goal. Our modified environment consists of a stochastic 3v3 game, where the teams' field two outfielders and a goalkeeper, respectively. The game has three terminating conditions: 1. Goal is scored, 2. Ball goes out of bounds and 3. Maximum time-steps reached. For our episodes we are capping the time-steps at five-hundred steps and the offsides rule is disabled. The two agents that are trained are the two outfielders. The goalkeeper and opposition players are controlled by the Game AI, which consists of some hardcoded logic which pertains to the native environment. The football environment's observation space is high-dimensional and continuous. Our modified environment consists of 43-dimensional vectors broken down as: 6 (x,y) coordinates and directions for each team (24 total), 3 (x,y,z) coordinates for both ball position and direction, 3 one-hot encodings of ball possession and active player and finally 7 game mode encodings. Our actions space is discrete consisting of 19 actions. Within these 19 actions, we have 8 directional actions, 4 passing/shooting actions and finally 7 player actions that can toggle sprinting, sliding, dribbling etc. Similarly to real life football, our reward function is based on scoring and conceding goals. Teams receive +1 for scoring and the opposing team receives -1 for concerning and vice versa. However, since scoring and conceding is rather sparse in our stochastic environment, reward shaping is implemented in the reward structure to provide frequent metric feedback. This reward is based on possession and grants a +0.1 for reaching 10 designated checkpoints in the game. Scoring grants the points for the remaining unlocked checkpoints.

We will be comparing our multi-agent algorithms with three baseline teams. These teams were trained using the Proximal Policy Optimization (PPO) algorithm, each using a unique configuration. PPO is a form of policy gradient methods, which is the family of algorithms explored in this paper. Policy gradient methods are a type of reinforcement learning algorithms that optimize policies with gradient descent. It is a model-free learning algorithm, which learns directly from experience. As such, policy gradient methods directly learn the optimal policy. Below is a generalized framework of policy gradient algorithms.

Algorithm 1 Vanilla Policy Gradient Algorithm

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, ...$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Estimate policy gradient as
           $\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$
7:     Compute policy update, either using standard gradient ascent,
           $\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$
       or via another gradient ascent algorithm like Adam.
8:     Fit value function by regression on mean-squared error:
           $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
       typically via some gradient descent algorithm.
9: end for
```

For every episode: we collect our metrics after executing a policy, then for every episode and timestep we compute the expected return and advantage function. We then refit our base model by minimizing the squared differences of our value function and return function. Afterwards, we update the policy using the gradient estimate of our advantage function. The disadvantages of policy gradient methods are that they suffer from high variance and low convergence. However, policy gradient methods offer simple implementation and can learn stochastically. As such, policy gradient algorithms have shown to be useful when implementing the "Football" environment. More specifically, with implementing Actor-Critic methods. The two policy gradient methods discussed in this paper are also Actor-Critic methods, which are a variant of policy gradient. Actor-Critic methods, learn the value function in addition to the policy. The method is composed of two components: the Critic that updates the value function and the Actor that updates the policy. Here the critic serves to evaluate the actor's updates through the value function. This is an important feature of the following two algorithms.

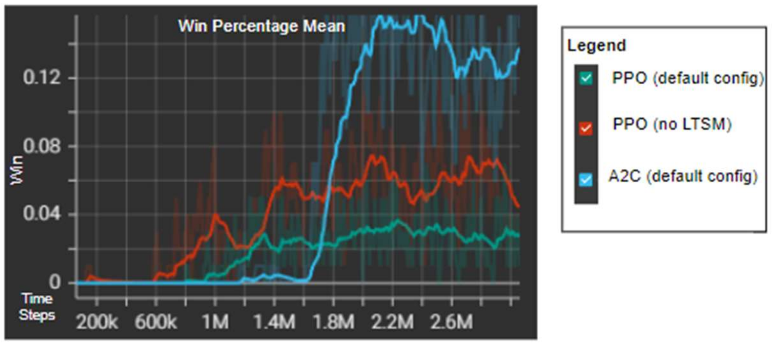
As stated, our baselines teams were trained using the Proximal Policy Optimization (PPO) algorithm. The PPO algorithm improves stability during training by reducing the effect of large parameter updates onto the policy. This helps against the high variance disadvantage of policy gradient methods. To do so, this algorithm implements a Kullback-Leibler divergence constraint onto the size of the updates. More specifically, PPO implements a constrained probability ratio to dampen the update effects. Below is the pseudo-code for the PPO framework:

Algorithm 1 PPO, Actor-Critic Style

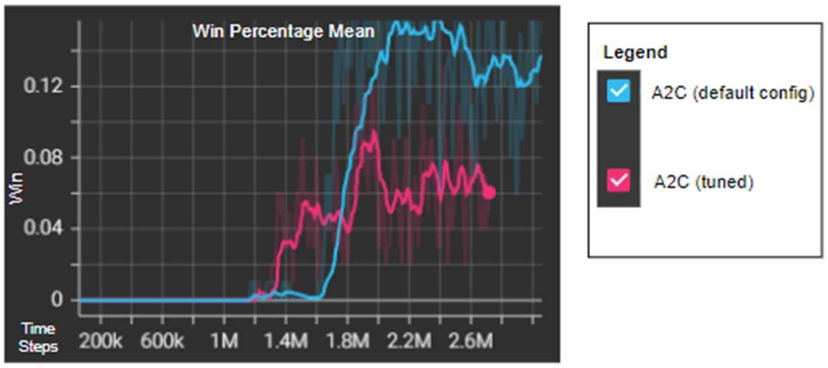
```
for iteration=$1, 2, \ldots$ do
    for actor=$1, 2, \ldots, N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

The PPO algorithm is considered a state-of-the-art algorithm with respect to multi-agent reinforcement learning due to its quick computational time and stability in results. The algorithm I tested against PPO was the Advantage Actor Critic (A2C) algorithm. A2C and PPO are similar algorithms except PPO has a clipped update step that stabilizes the high variance of the model. It prevents our updated policy from steering too far from our prior policy by applying the KL constraint explained above. A2C lacks this clipping feature which can be both advantageous and disadvantageous during training. Since we do not dampen the effect of our advantage, A2C can be influenced heavily by stochasticity and training, as an actor's preferred actions could be influential to the rest of the training process. However, in the context of the Football environment we want our agents to push advantages earlier and further.

To get an understanding of the algorithms and environment, I did four trial runs using two PPO algorithms (one using default config and the other using no LTSM and ReLU activation) and one A2C algorithm (default config). I compared their win rate across 5M time-steps to get an understanding of their performance in comparison. Below are the results:

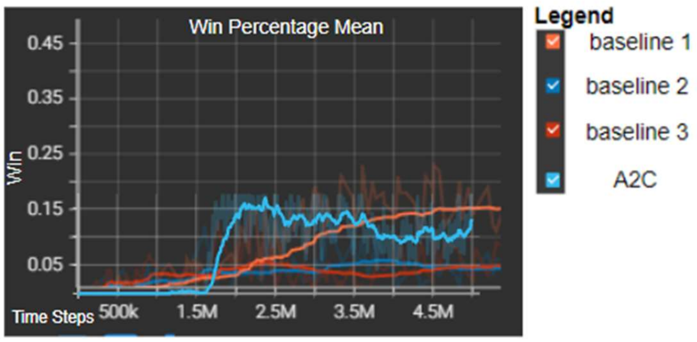


As seen, A2C had a higher win-rate percentage when compared to the two PPO algorithms. It has a sharper spike almost doubling in win-rate percentage. This spike could have resulted from the lack of clipping in the A2C algorithm. As stated, PPO clips large policy changes, which gives it stable learning increments as seen with our two algorithms. After analyzing the results, I then opted to further test the A2C algorithm. I trained one A2C model on the default configuration and another using my tuned hyperparameters. The key difference between the two was in the learning rate applied. The tuned algorithm had a slightly reduced learning rate. One of the disadvantages of A2C is its sensitivity to hyperparameters so I wanted to see the extent of the effect. Below are the win rate results:

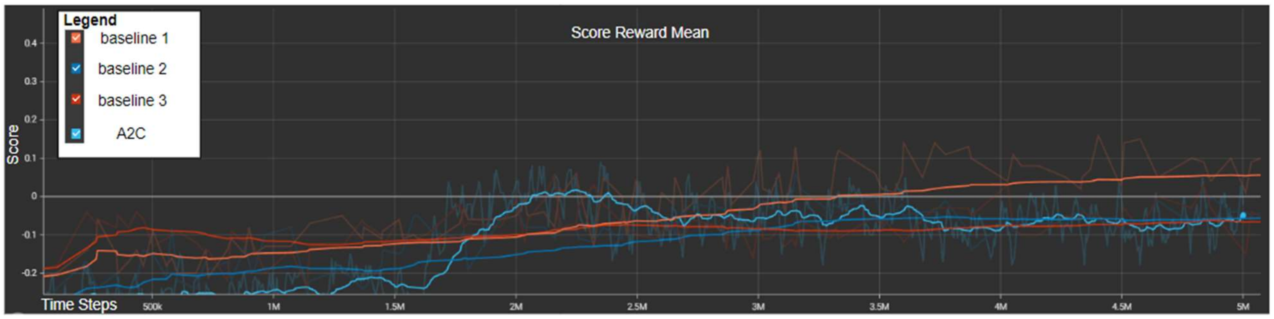


When comparing my two algorithms, the default configuration performed quicker and better than the tuned A2C algorithm. One thing that was peculiar was how much more computationally expensive it was to run my tuned algorithm. I initially planned on tuning to 5M steps however the tuned algorithm slowed down drastically past 1.8M time-steps. This resulted in taking twice the time to produce half of the results of the default configuration. Due to time and resource constraints I opted to evaluate the default A2C configuration against the three baseline models.

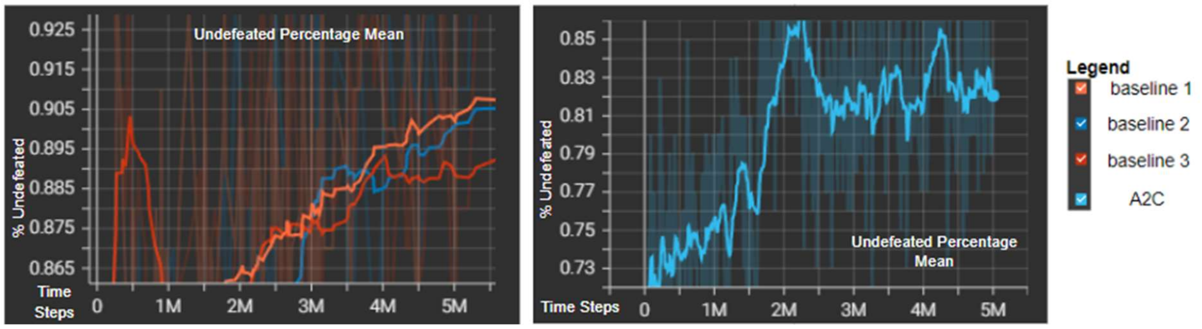
As stated, I trained my agent on the A2C algorithm and compared my results to the three baseline models. The following represents the win-rates of the three baselines PPO algorithms to my best A2C training run. Due to the computational expense of running the algorithms I trained them for 5M time steps and compared. Below are the results:



As seen, PPO and A2C do not show substantive results until about 1.5M time-steps. The A2C run spikes dramatically at 1.5M time steps and steadily declines until it reaches a steady state above baseline models two and three. One thing to note is that since the A2C's policy improvement is less stable when compared to the baseline models. Again, this could be because of the lack of the advantage dampener used in PPO. I then evaluated the scoring and undefeated percentages for each respective algorithm. Scoring is the obvious desired reward, as such I am hoping by scoring a lot early the A2C critic can prioritize those actions. I trained the A2C algorithm for 5M steps and below are the scoring results:



As seen A2C has significant spike at around 2M time-steps. This is in-line with the spike increase in win-percentage that was seen earlier. This is important to note as the spike in scoring must have led to the spike in win-rate. However, after the initial spike our A2C algorithm eventually regresses and converges near the same level as baselines 2 and 3. Baseline 1 performed the best overall, which is interesting as it uses the default hyper parameter configuration. I then looked at the undefeated percentage and compared. Below are the results:



As we can see, all baselines had a higher undefeated percentage. Since higher scoring requires players to be higher up on the pitch it exposes the goalkeeper to counter attacks. This is perhaps why the A2C algorithm had a lower undefeated percentage. Although it wins more games, it loses more as well. Additionally, since PPO reduces huge advantages, riskier moves that lead to wins have less of an affect on our training. This is the central drawback of A2C as the training trajectory can skew our learning. With additional time and computational power, I would run my A2C algorithm for longer and see how the results would compare.

To finalize this paper examined how policy gradient methods can be used in multi-agent environments. As highlighted, A2C can lead to quicker and better results, at the cost of learning stability. PPO faired better during longer training sessions when compared to A2C. As stated, given additional computational power and time, I would train the A2C for a longer period to see how it compares long term with PPO.

References

Mnih, Volodymyr, et al. "Asynchronous Methods for Deep Reinforcement Learning." *ArXiv.org*, 16 June 2016, https://arxiv.org/abs/1602.01783.

Schulman, John, et al. "Proximal Policy Optimization Algorithms." *ArXiv.org*, OpenAI, 28 Aug. 2017, https://arxiv.org/abs/1707.06347.

"Vanilla Policy Gradient." *Vanilla Policy Gradient - Spinning Up Documentation*, https://spinningup.openai.com/en/latest/algorithms/vpg.html#vanilla-policy-gradient.

Yu, Chao, et al. "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games." *OpenReview*, 4 June 2022, https://openreview.net/forum?id=YVXaxB6L2Pl.

Zhang, Kaiqing, et al. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms." *ArXiv.org*, 28 Apr. 2021, https://arxiv.org/abs/1911.10635.