Experiment Report For MetaRacingEnvironment

TL;DR

- Training a PPO algorithm in 2-players competitive environment with elaborate code implementation and hyper-parameters tuning.
- Design a Prioritized Fictitious Self-Playing Algorithm for algorithm to dominate the game, inspired by AlphaStar.
- Enhancing Actor-Critic algorithm by:
    - Use opponents' observations as value network's inputs.
    - Use next state prediction task as auxiliary to get better value evaluation.

Implementation

Stablize PPO

RL learning is very bumpy and can be affected by many potential factors. We carefully investigate some implementation of open-source RL libraries and papers. We find some engineering trick to boost PPO's performance in most cases[1] compared to vanilla PPO provided by TA:

- Reward scaling: I use a discounted-based scale factor to change rewards' variance. It will kind of smooth the rewards and reduce the variance of estimation.
- Return calculation: Use UPGO instead of GAE to generate advantages. This can reduce bias when model try to learn a better policy.
- Orthogonal initialization and layer scaling to initialize NNs. We can observe a more rapid rewards growth at starting stage when training.
- Learning rate warm-up and annealing: Due to unstableness of training, we use warm-up to reduce the impact of model learning a very bad data at first few batchs. Annealing helps when training in single agent environment but do not apply for multi agent game.
- Tanh Activation: More gradients can be propagated by tanh actvation than ReLU. The other adavantages in this case is that actions are bounded in [-1, 1] and tanh activation helps model's hidden states stay in a rational range.

Also we do other tricks to avoid potential inf/nan/big output.

- Action std clipping: This helps model do not generate abnormal output when sampling from unconstrained normal distribution. It can be implemented in different ways:
    - Use output as std instead of logstd. Use softplus to ensure a positive value.
    - Clip std at a max of 2.

    - Use Beta distribution instead of Normal distribution. Beta distribution is bounded in [0, 1] and can be easily scaled to action space. Some paper indicates this can siginificantly impact learning process[2].

Modify Reward

We modify the reward function of environment to make it like a zero-sum game. However, agent will be punished by other agents' rewards, but would not be rewarded by other agents' mistakes. This prevent situations like keeping crash on other vehicles to get reward from others' crash sidewalk penalty.

Fictitious Self-Play

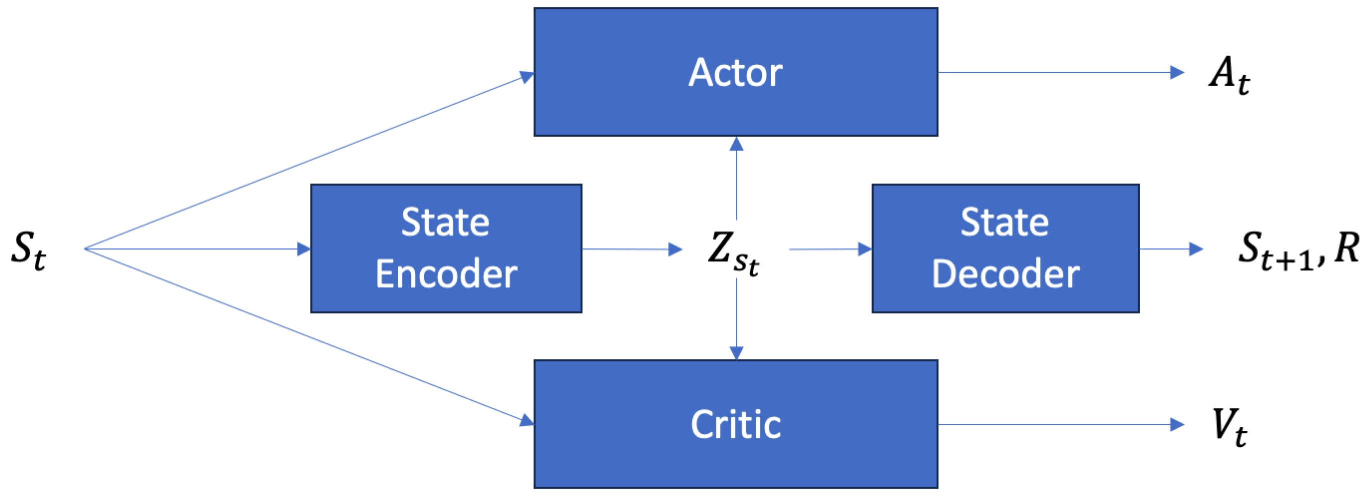
Self-play is the key to the success of AlphaGo. However, do self-play only can lead to a sub-optimal model quickly. So AlphaStar designed league train to create diverse style players to avoid sub-optimal. However, due to lack of computing resources, I simplify league train and only use its squared-p PFSP. To ensure diversity of agents, I will change some paramters when training and add them into a public pool.

The main agent will choose their opponent as the following procedure:

- 12%, self-play:
    - The reason for 12% is higher probability of self-play will cause a very large parameters update in model (kl>1) and I still don't know the reason...
- 18%, play with the stronger players (main agent have a win rate < 0.3)
- 70%, play with all players in pool
    - if overall win rate > 0.3, use squared-p
    - else, use variance-p

State Embedding

Similar to State-action-learned-embedding(SALE)[3], I design a module to learn state embedding to enhance value network. The architecture is in Fig1.



And this embedding should be low-dimension because the original dimension is very small(161) and on-policy data collection. Or it will strongly impact training and cannot converge. We set the embedding to 32. But we actually do not use this in multi agent training because of computing cost. But we do observe slight performance elevation in single agent environment (at least this would not cause worse model).

Experiment Setup

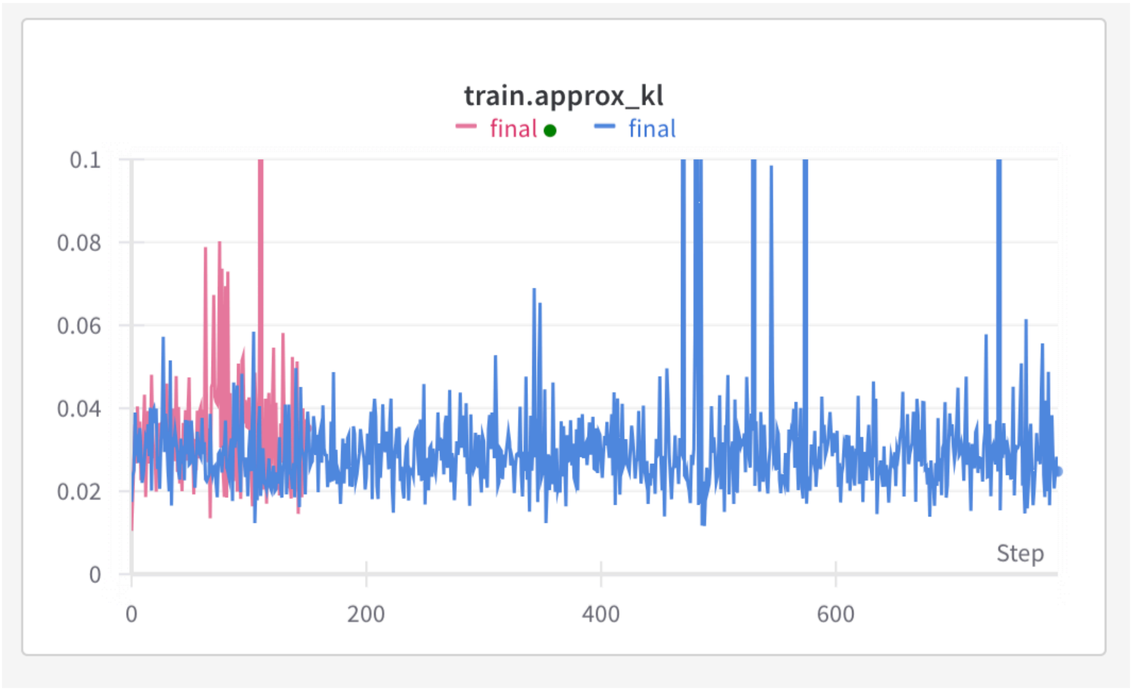
This contains all the hyper-parameters of final experiment.

```
class EnvironmentConfig:
    num_agents: int = 2
    crash_sidewalk_penalty: float = 1
    success_reward: float = 100
    speed_reward: float = 0
    out_of_road_penalty: float = 40
    crash_vehicle_penalty: float = 0 # Default is 10. Set to -10 so env will reward agent
+10 for crashing.
    idle_penalty: float = 40
    horizon: int = 2000 # Notice!

class PlayerConfig:
    # Evaluation
    num_processes_eval: int = 20
    num_episodes_to_eval: int = 10
    num_rollouts_to_save: int = 30
    num_rollouts_to_payoff: int = 10

    # Data
    num_agents: int = 2
    num_steps: int = 2000 if not debug else 100
    num_processes: int = 20 if not debug else 2
    num_epochs: int = 10 if not debug else 2
    mini_batch_size: int = 640 if not debug else 64

    # Train
    training_steps: int = 1_000_000
    epsilon: float = 1e-8
    hidden_size: int = 1024
    lr: float = 5e-5
    lr_schedule: str = "fix" # "fix" or "annealing"
    gamma: float = 0.99
    gae_lambda: float = 0.95
    clip_range: float = 0.2
    ent_coef: float = 0.
    vf_coef: float = 0.5
    max_grad_norm: float = 0.5
    calculate_return: str = "upgo" # "gae" or "upgo"
    sample_dist: str = "normal"

    # Game
    use_opponent_obs: bool = True
    use_representation_learning: bool = False
    state_embedding_size: int = 32
```

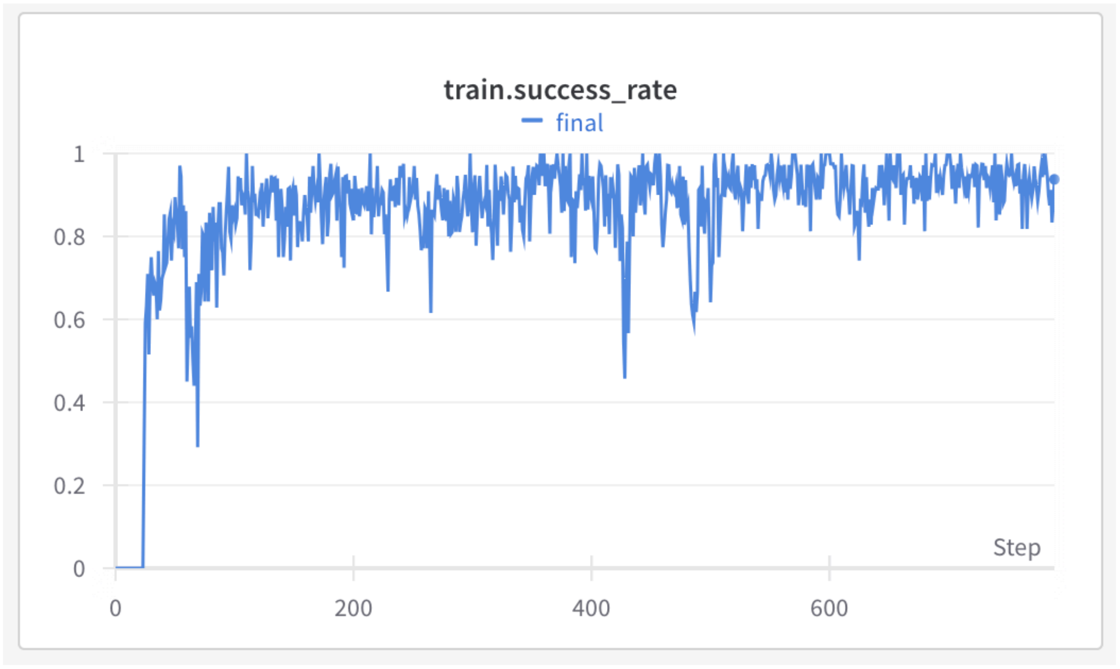
Results and Discussion

Training Curve

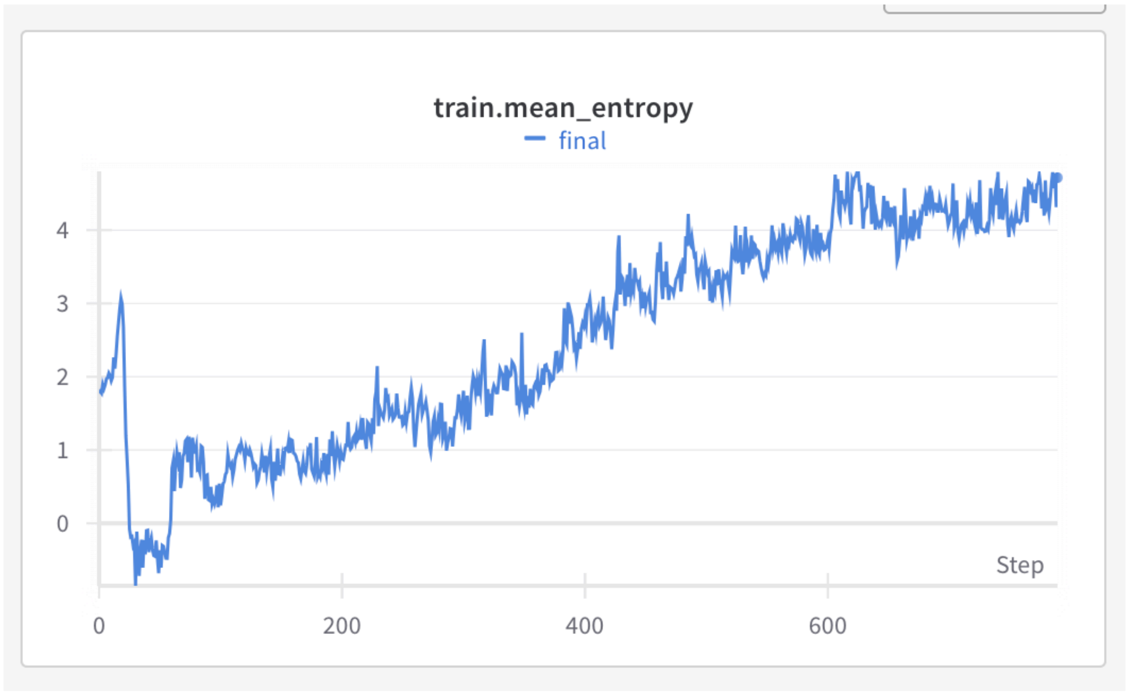
We use an adaptive clip-range, so we control kl in [0.02, 0.04] to make sure model would not update too fast.



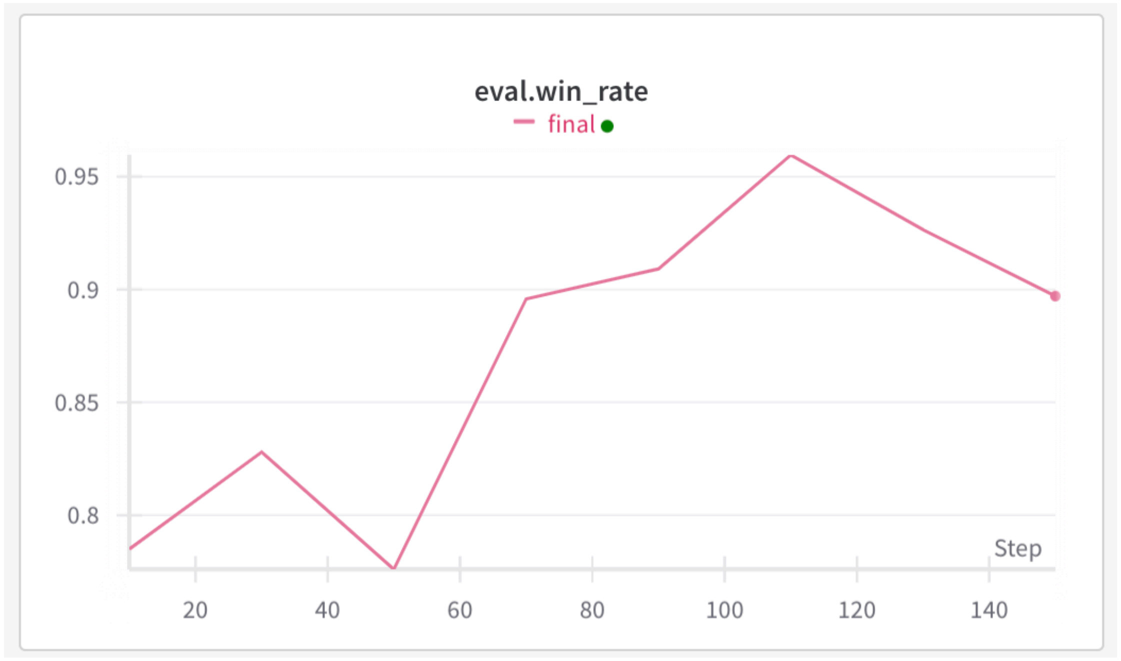
The success rate of agent keep around 0.9.



I notice that the entropy of model output keeps rising. This may indicates that a deterministic policy cannot handle this game. You should use random policy to make your opponent unable to predict your actions sometimes.



And finally, the main agent can beat most of its ancestors.



Strange Problem

- MetaDrive environment seems to cause memory leaking. Fixed by del and remake.
- PPO training sometimes causes inf ratio. Fixed by constrain action_logstd and reduce probability of self-play.

Reference

[1] https://arxiv.org/abs/2005.12729

[2] https://www.ri.cmu.edu/app/uploads/2017/06/thesis-Chou.pdf

[3] https://arxiv.org/abs/2306.02451