Assignment 2 – Policy Gradients Methods

Yonatan Amaru, 316365998
Etay Lorberboym, 314977596

How to run the code:

We have split the code to two modules under the src package:

1. Networks.py – contains all the network we have utilized.
2. Game_runners.py – contains all the run configurations we have used.

We have created an experiments.ipynb notebook to evaluate each run configuration 10 times to reduce randomness of the scores, then we saved the results and performed analysis in the results_analysis.ipynb notebook.

In order to run the code, install the dependencies in the requirements.txt and use the experiments notebook which enables to run each configuration separately.

Section 1 – Monte-Carlo Policy Gradient (REINFORCE)

In this section we focus on the REINFORCE algorithm.

> What does the value of the advantage estimate reflect?
>
> Why is it better to follow the gradient computed with the advantage estimate instead of just the return itself? (5%)

The value of the advantage estimate $\hat{A}_t = R_t - b(s_t)$ reflects how much better an action is compared to the average performance in a given state, using a baseline set as the value $V^{\pi}(s_t)$. This approach is preferred for two key reasons:

1. **Variance Reduction:** It lowers the variance in policy gradient estimates by subtracting the baseline from the returns, leading to a more stable and efficient learning process.
2. **Focused Improvement:** By measuring the quality of actions relative to the policy average, it directs learning towards actions that are not just good, but significantly so, enhancing the policy's performance more effectively.

Using the advantage leads to more stable, focused learning by emphasizing actions that significantly outperform the average, leading to faster and more precise policy improvement.

The reduction of the baseline **does not** introduce bias to the expectation of the gradient since:

$$\mathrm{E}_{\pi_\theta}[\nabla log\pi_\theta(a_t|s_t)b(s_t)] = 0$$

What is the prerequisite condition for that equation to be true? Prove the equation (10%)

The equation $\mathrm{E}_{\pi_\theta}[\nabla log\pi_\theta(a_t|s_t)b(s_t)] = 0$ states that the expected value of the product of the gradient of the log policy and the baseline equals zero. This means that introducing a baseline does not add bias into the gradient estimates for the policy optimization.

The prerequisite condition for the equation to be true is that the baseline is independent of the action a^t. The baseline can be a function of s_t but not the action chosen at state t.

Proof:

To prove $\mathrm{E}_{\pi_\theta}[\nabla log\pi_\theta(a_t|s_t)b(s_t)] = 0$, we start by expanding the expectation in the context of policy gradient methods:

$$\mathrm{E}_{\pi_\theta}[\nabla log\pi_\theta(a_t|s_t)b(s_t)] = \sum_a \pi_\theta(a \mid s_t)\nabla log\pi_\theta(a \mid s_t)b(s_t)$$

Using the log derivative trick $\nabla log\pi_\theta(a \mid s_t) = \frac{\nabla\pi_\theta(a|st)}{\pi_\theta(a|st)}$

We rewrite the expression:

$$\sum_a \pi_\theta(a \mid s_t)\frac{\nabla\pi_\theta(a \mid st)}{\pi_\theta(a \mid st)}b(s_t) = \sum_a \nabla\pi_\theta(a|s_t)b(s_t)$$

However, since we assume the $b(s_t)$ is independent of a, we factor out the differentiation:

$$b(s_t)\sum_a \nabla\pi_\theta(a|s_t)$$

The sum of the derivatives of the probability of $\nabla\pi_\theta(a|s_t)$over all actions a must sum to zero because the probabilities sum to 1, which is a constant.

$$\sum_a \nabla\pi_\theta(a|s_t) = {}^{\nabla}1 = 0$$

Therefor we get:

$$b(s_t) * 0 = 0$$

This property is crucial for ensuring that the introduction of a baseline for variance reduction does not affect the unbiased nature of the policy gradient estimate.

Section 1: Cart pole – REINFORCE with and without baseline

To incorporate the value-function approximation baseline in our code we committed the following updates:

1. **Introduce a Value Network:** We added a second NN to approximate the value function $V^{\pi}(s_t)$. This network will predict the expected return from state s_t under the current policy.
2. **Calculate Advantage Estimates:** Instead of using the total return R_t directly, we calculate the advantage estimate $A_t = R_t - V^{\pi}(s_t)$ using the predicted value from the value network as the baseline.
3. **Update both networks:** We updated the policy network using gradients calculated with the advantage estimates. The value network will be updated to minimize the difference between its predictions and the observed returns.

Compare the results before and after the change (please refer also to convergence time).

To Compare the results of REINFORCE with REINFORCE BASELINE, we tested the model with each configuration ten times, this averaging out the results and reducing the randomness.

<u>10 Runs for REINFORCE Policy and REINFORCE with Baseline Policy</u>

```
# Run the experiments for basic REINFORCE
policy_results = []
for i in range(number_of_experiments):
    policy_results.append(gr.run())

with open('result_score/policy_results.pkl', 'wb') as f:
    pickle.dump(policy_results, f)
```

```
 21%|▊         | 1070/5000 [02:49<10:21,  6.32it/s]
 10%|▌         | 486/5000 [01:11<11:08,  6.75it/s]
 21%|▊         | 1070/5000 [03:27<12:43,  5.14it/s]
 15%|▋         | 764/5000 [02:02<11:18,  6.24it/s]
 15%|▋         | 743/5000 [02:17<13:07,  5.40it/s]
 15%|▋         | 732/5000 [02:11<12:45,  5.58it/s]
 21%|▊         | 1059/5000 [04:00<14:54,  4.40it/s]
 32%|▉▉▉       | 1603/5000 [06:02<12:48,  4.42it/s]
 17%|▋         | 834/5000 [02:35<12:56,  5.37it/s]
 19%|▊         | 960/5000 [02:54<12:14,  5.50it/s]
```

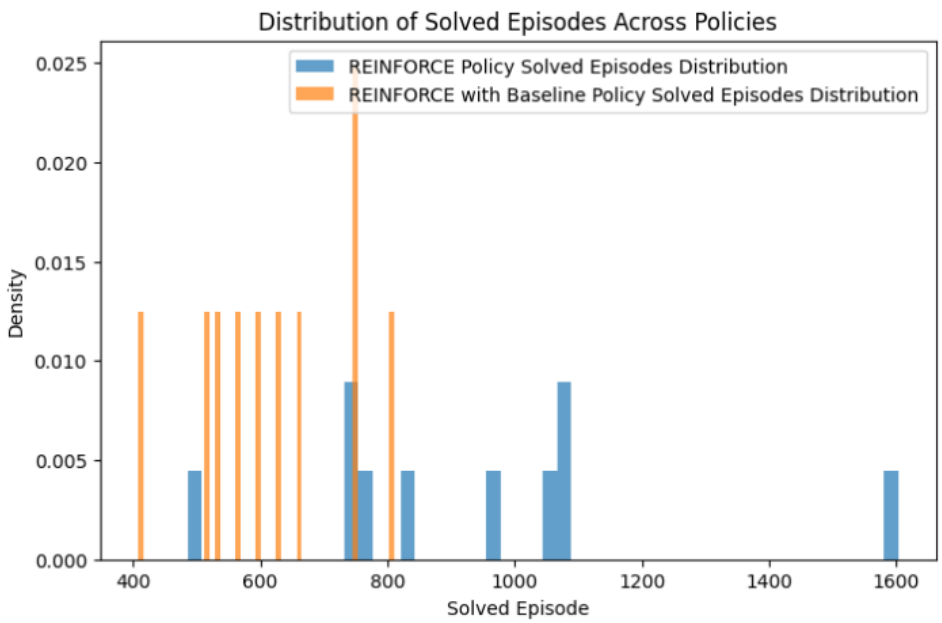
```
# Run the experiments for REINFORCE with baseline
policy_baseline_results = []
for i in range(number_of_experiments):
    policy_baseline_results.append(gr.run_with_baseline())

with open('result_score/policy_baseline_results.pkl', 'wb') as f:
    pickle.dump(policy_baseline_results, f)
Executed at 2024.02.17 15:42:11 in 7m 30s 517ms
```

```
 13%|▉         | 661/5000 [03:31<23:06,  3.13it/s]
 12%|▉         | 597/5000 [03:04<22:42,  3.23it/s]
 11%|▉         | 566/5000 [02:58<23:16,  3.17it/s]
 15%|▋         | 748/5000 [04:19<24:34,  2.88it/s]
 16%|▋         | 810/5000 [05:02<26:05,  2.68it/s]
  8%|▌         | 408/5000 [01:36<18:10,  4.21it/s]
 10%|▌         | 514/5000 [02:27<21:28,  3.48it/s]
 13%|▉         | 633/5000 [03:29<24:02,  3.03it/s]
 15%|▋         | 750/5000 [04:57<28:07,  2.52it/s]
 11%|▉         | 531/5000 [02:40<22:26,  3.32it/s]
```

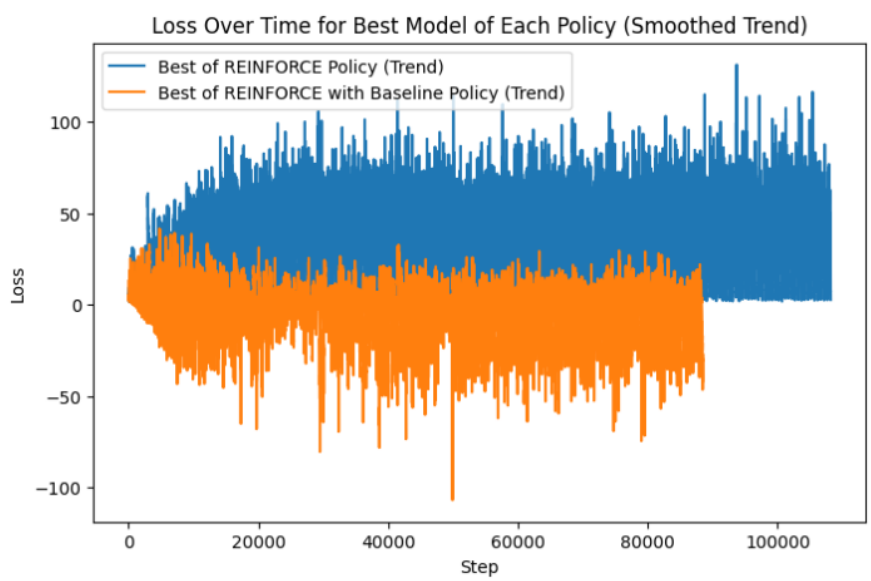
1. **How quickly does each method solve the environment?**

```
Average solved episode (REINFORCE Policy): 932.1
Average solved episode (REINFORCE with Baseline Policy): 621.8
```



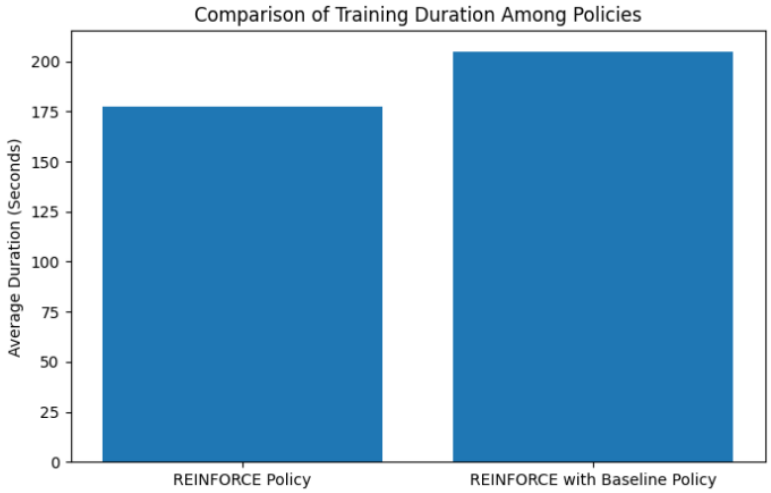
2. **How does the average loss evolve?**

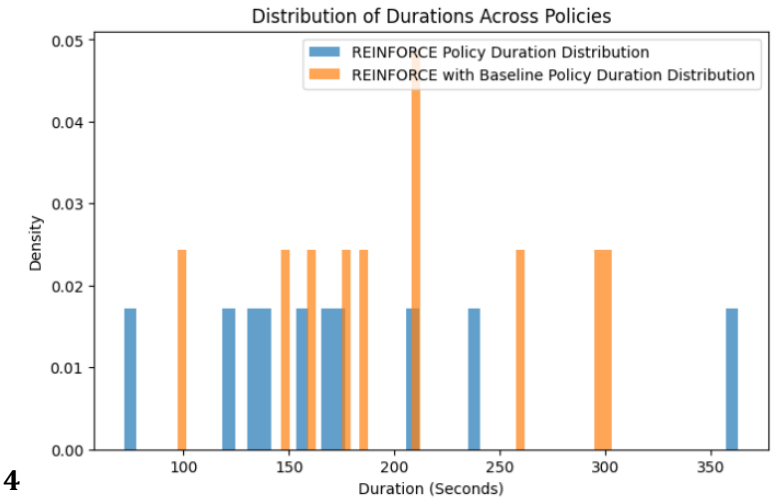
```
Average policy network loss for REINFORCE Policy: 40.6841
Average policy network loss for REINFORCE with Baseline Policy: 0.3793
Average value network loss for REINFORCE with Baseline Policy: 500.9605
```



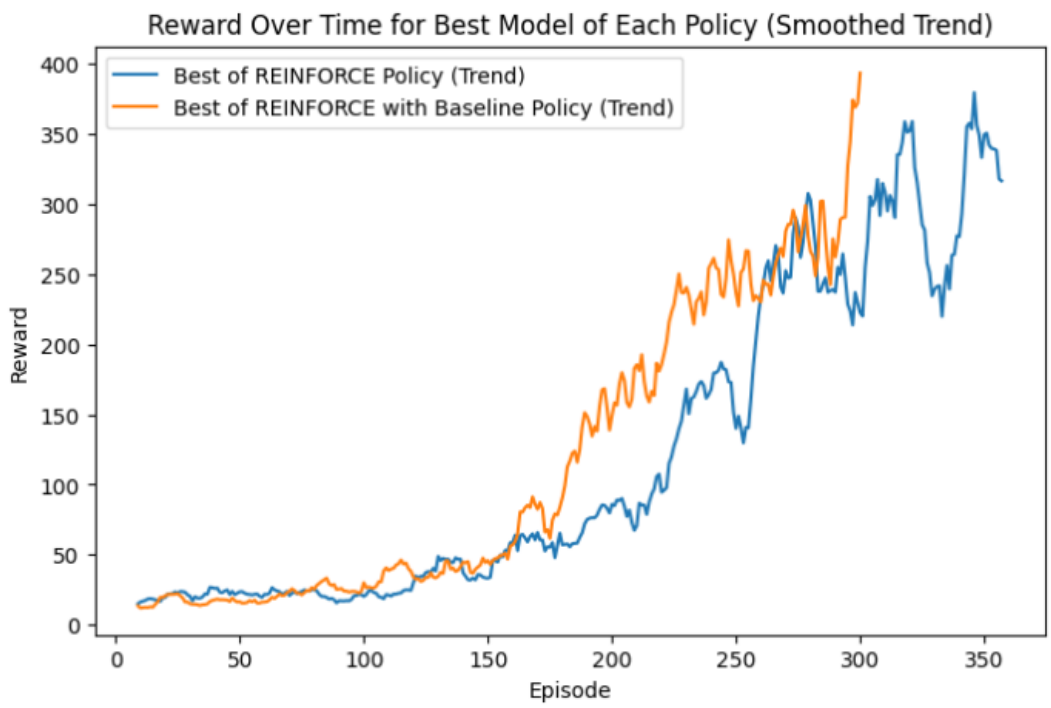
3. How efficient is the learning process in terms of duration?

```
Average training duration for REINFORCE Policy: 177.47 seconds
Average training duration for REINFORCE with Baseline Policy: 205.01 seconds
```





4 Time for the "Best" Model of Each Policy?



Key Insights:

- Incorporating the baseline significantly reduced the number of steps it took to learn a good policy.
- We ran the model on a CPU. Sense matrix multiplication is an expensive operation on the CPU, despite converging faster stepwise, on average the model took a longer time duration to train.
- The policy loss is much more stable with a baseline.
- For harder problem, this added stability can be a significant factor.
- We expect the time difference to be minimized when running on a GPU in an optimized setting.
- The graph shows that while both policies improve over time, the "Best of REINFORCE with Baseline Policy" consistently achieves higher rewards compared to the "Best of REINFORCE Policy", particularly evident after around 150 episodes.
- We observed that although the "REINFORCE with Baseline" policy typically solves episodes earlier, this does not correlate with shorter durations. In fact, the duration for the "REINFORCE with Baseline" was longer across most models.

Section 2 - Advantage Actor-Critic

In this section we will focused on the Advantage Actor-Critic evolvement of policy gradients.

> Why is using the TD-error of the value function for the update of the policy network parameters is practically the same as using the advantage estimate (of equation 1)? Prove. (5%)

In the Advantage Actor-Critic method, the advantage estimate is given by

$\hat{A}_t = Q_w(s_t, a_t) - V_v(s_t)$. However, in practice, the Temporal Difference (TD) error δ_t is used for updating the policy network parameters. The TD error in the context the Actor-Critic methods is defined as:

$$\delta_t = R_{t+1} + \gamma V_v(S_{t+1}) - V_v(S_t)$$

Why is using δ_t practically the same as using the advantage estimate $\hat{A}_t$ for updating the policy parameters?

The Q-value for state-action pair (s_t, a_t) can be estimated using the bellman equation as:

$$Q_w(s_t, a_t) = R_{t+1} + \gamma V_v(S_{t+1})$$

Substituting this into the advantage estimate, we get:

$$\hat{A}_t = R_{t+1} + \gamma V_v(S_{t+1}) - V_v(s_t) = \delta_t$$

Therefore, the TD error δ_t is an unbiased estimate of the advantage function $\hat{A}_t$ because it is derived directly from the definition of the advantage, which is the difference between the estimated Q-value and the value function $V_v(s_t)$.

This allows us to simplify the computation of Actor-Critic by using the TD error instead of explicitly computing the advantage estimate at each step, which would require maintaining an additional set of weights for the action-value function.

Explain which function is the *actor* and which is the *critic* of the model and what is the role of each one. (3%)

The Actor:

the actor represents the policy function. Its role is to select actions from the environment. The actor is responsible for the actual decision-making process and is updated based on the policy gradient, which is influenced by the TD error provided by the critic. This update is intended to increase the probability of good actions that lead to higher rewards and decrease the probability of bad actions.

The Critic:

The critic represents the value function. Its role is to evaluate the actions taken by the actor by estimating the value of the current policy for the given state. The critic computes the TD error, which serves to estimate the advantage function and provides the actor with feedback on the quality of the actions it has taken. The critic helps the actor to understand which actions are better or worse than what it typically expects to receive in return from the state.

Interaction:

The actor and critic work together to improve the policy. The interaction creates a feedback loop where the policy is continuously being adjusted based on the critic's assessments, which in turn are based on the actor's actions.

Section 2: Cart Pole – Actor Critic Agent

In this section we Implement an Actor-Critic algorithm. Using policy_gradients.py, we committed the following updates:

1. **Introduce a Value Network:** We added a second NN to approximate the value function. This network will predict the expected return from state s_t under the current policy.
2. **Update the Actor:** Update the policy network (actor) using the TD error from the critic instead of the total discounted return.
3. **Update the Critic:** Update the value network (critic) using the TD error.

Compare the results with the previous two models.

Upon evaluating the outcomes of REINFORCE against actor-critic, as well as REINFORCE and REINFORCE with BASELINE, it was observed that the actor-critic variation of REINFORCE yielded less favorable results for the Value Network utilized in the REINFORCE with BASELINE setup, which exhibited superior performance. Consequently, a larger network architecture comprising two hidden layers was developed. This new configuration was subjected to testing across 10 training iterations (10 models for each setup), consistent with the methodology applied in the preceding section.

<u>10 Runs for REINFORCE with Actor Critic Policy and REINFORCE with Actor Critic and bigger Value Network Policy</u>

```python
# Run the experiments for actor critic with smaller net
policy_actor_critic_results = []
for i in range(number_of_experiments):
    policy_actor_critic_results.append(gr.run_actor_critic())

with open('result_score/policy_actor_critic_results.pkl', 'wb') as f:
    pickle.dump(policy_actor_critic_results, f)
```

```
 41%|████      | 2052/5000 [04:51<06:59,  7.03it/s]
 37%|███       | 1851/5000 [04:19<07:21,  7.14it/s]
 58%|└└└└└└    | 2907/5000 [07:57<05:43,  6.09it/s]
 47%|└└└└      | 2363/5000 [06:45<07:32,  5.83it/s]
 50%|└└└└      | 2509/5000 [05:13<05:11,  8.01it/s]
 74%|└└└└└└└   | 3698/5000 [11:36<04:05,  5.31it/s]
 59%|└└└└└└    | 2964/5000 [07:08<04:54,  6.91it/s]
 52%|└└└└      | 2605/5000 [05:04<04:39,  8.56it/s]
 62%|└└└└└└    | 3111/5000 [07:23<04:29,  7.02it/s]
 43%|└└└└      | 2150/5000 [05:38<07:28,  6.35it/s]
```

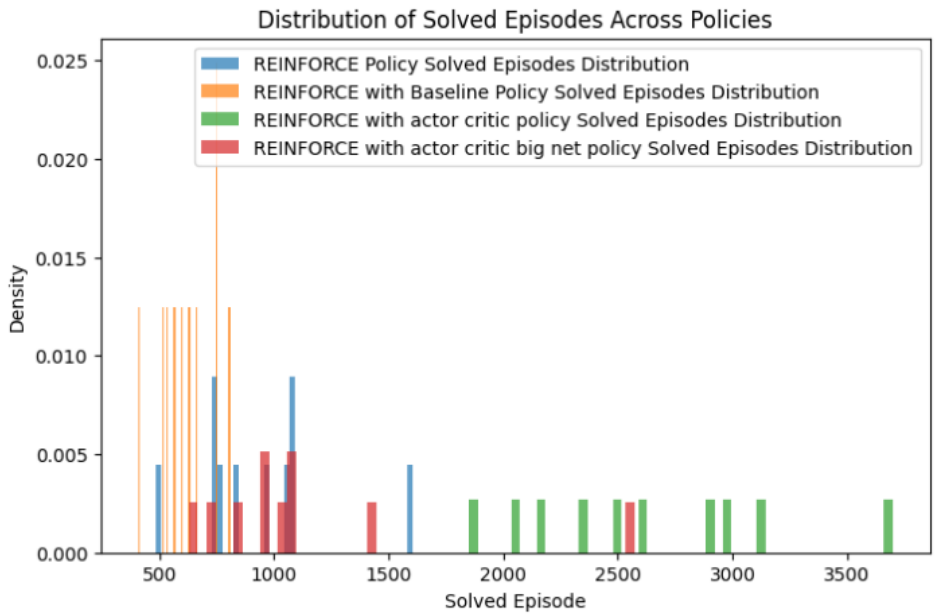
```python
# Run the experiments for actor critic with bigger net
policy_actor_critic_results_big_net = []
for i in range(number_of_experiments):
    policy_actor_critic_results_big_net.append(gr.run_actor_critic(is_big=True))

with open('result_score/policy_actor_critic_results_big_net.pkl', 'wb') as f:
    pickle.dump(policy_actor_critic_results_big_net, f)
```

```
2024-02-21 18:36:50.301825: W tensorflow/core/common_runtime/gpu/gpu_device.cc:
libraries mentioned above are installed properly if you would like to use GPU.
download and setup the required libraries for your platform.
Skipping registering GPU devices...
2024-02-21 18:36:50.308270: I tensorflow/compiler/mlir/mlir_graph_optimization_
 13%|└         | 629/5000 [02:34<17:55,  4.06it/s]
 21%|└└        | 1074/5000 [06:15<22:50,  2.86it/s]
 15%|└         | 733/5000 [03:08<18:16,  3.89it/s]
 20%|└└        | 976/5000 [04:58<20:32,  3.26it/s]
 22%|└└        | 1075/5000 [05:48<21:13,  3.08it/s]
 51%|└└└└└     | 2571/5000 [22:37<21:22,  1.89it/s]
 20%|└└        | 977/5000 [04:17<17:39,  3.80it/s]
 17%|└└        | 831/5000 [05:06<25:37,  2.71it/s]
 21%|└└        | 1039/5000 [05:04<19:20,  3.41it/s]
```

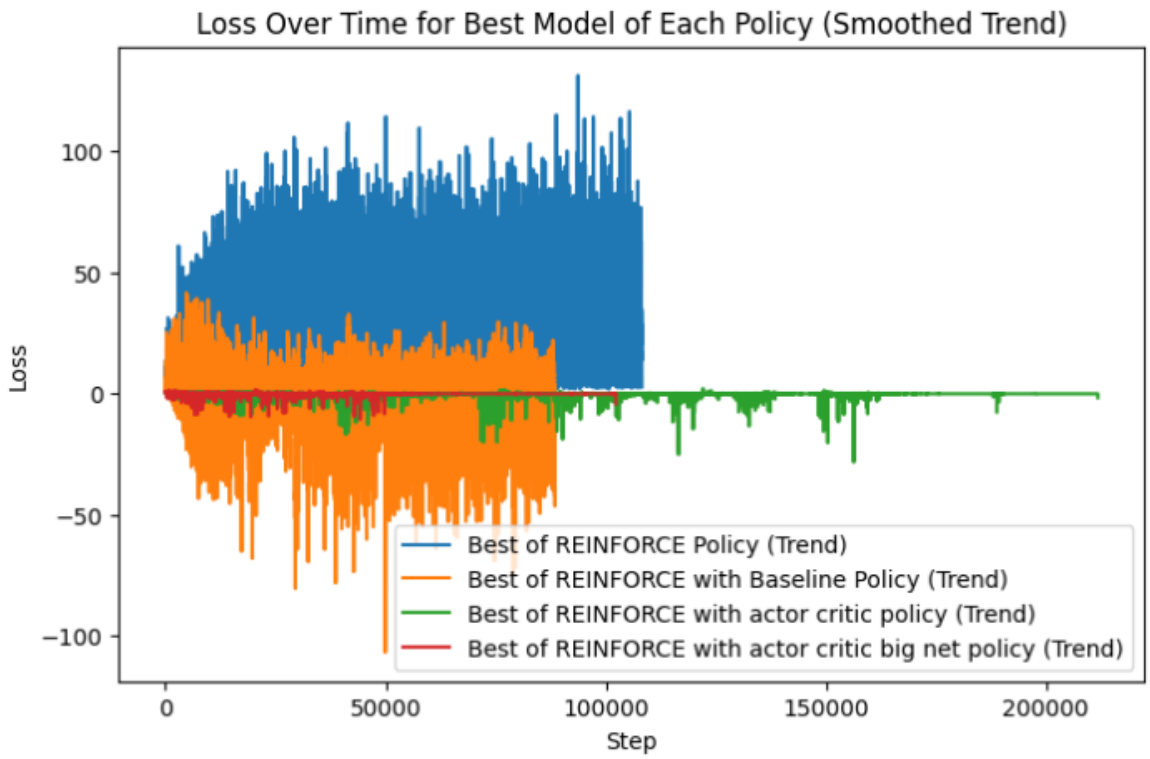
1. **How quickly does each method solve the environment?**

```
Average solved episode (REINFORCE Policy): 932.1
Average solved episode (REINFORCE with Baseline Policy): 621.8
Average solved episode (REINFORCE with actor critic policy): 2621.0
Average solved episode (REINFORCE with actor critic big net policy): 1132.7
```



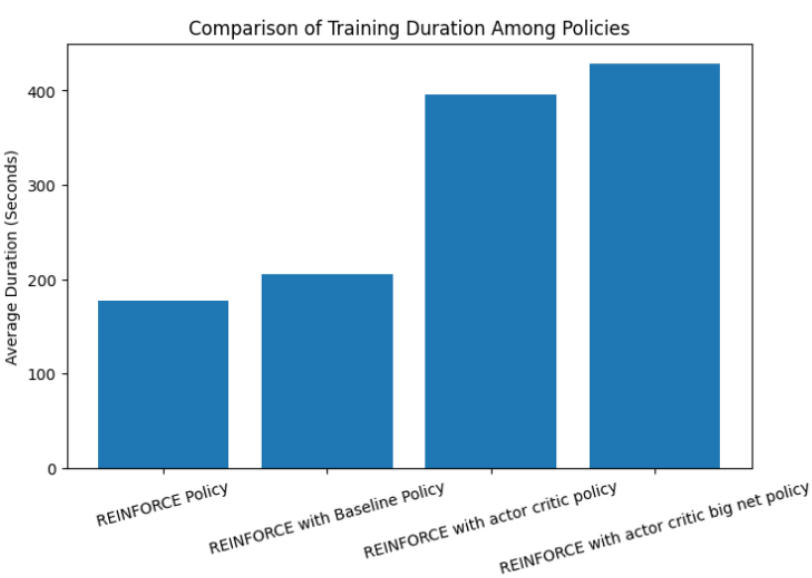
2. **How does the average loss evolve?**

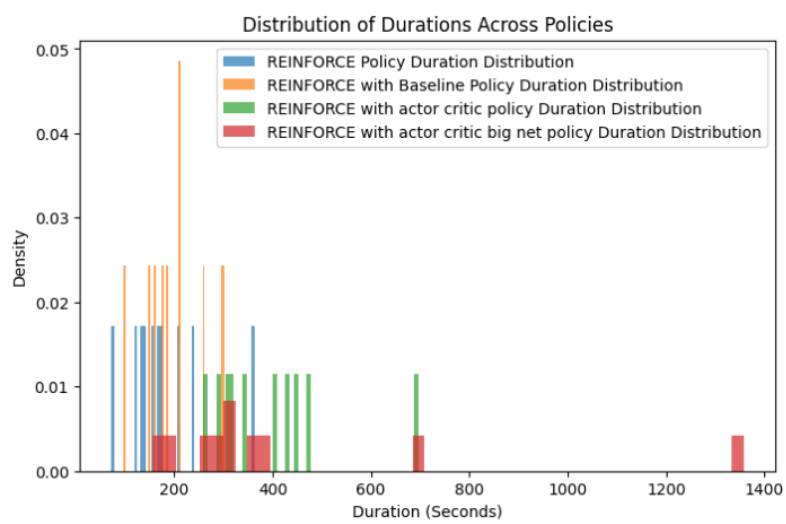
```
Average policy network loss for REINFORCE Policy: 40.6841
Average policy network loss for REINFORCE with Baseline Policy: 0.3793
Average value network loss for REINFORCE with Baseline Policy: 500.9605
Average policy network loss for REINFORCE with actor critic policy: -0.0362
Average value network loss for REINFORCE with actor critic policy: 13.4166
Average policy network loss for REINFORCE with actor critic big net policy: -0.0149
Average value network loss for REINFORCE with actor critic big net policy: 2.1839
```



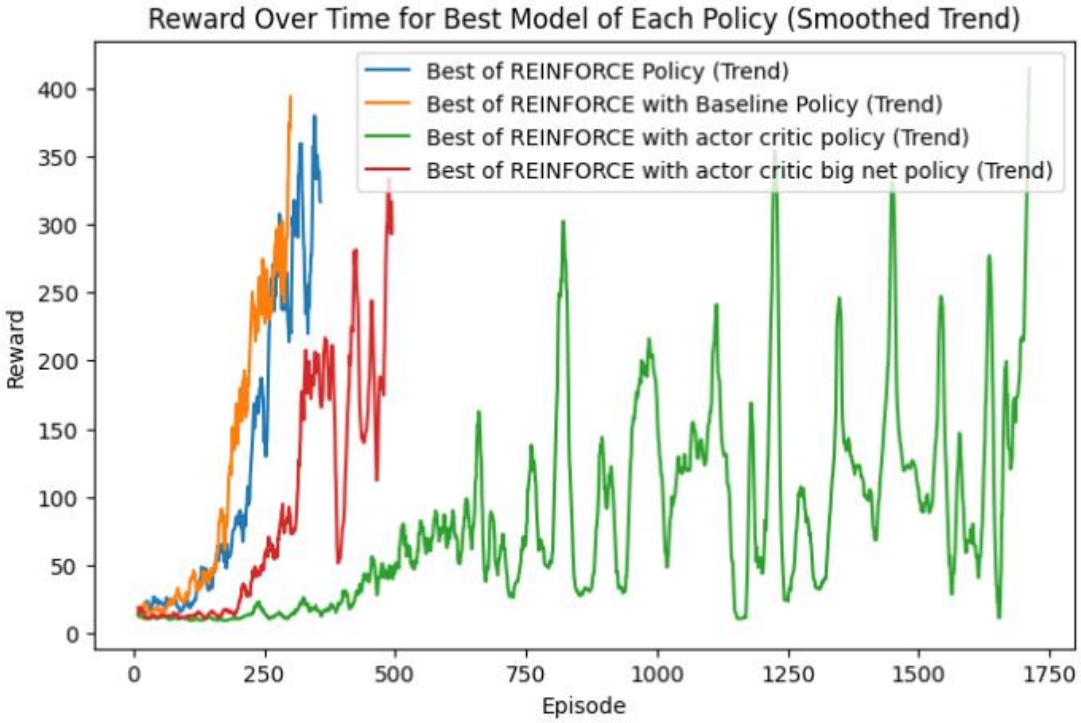
3. How efficient is the learning process in terms of duration?

```
Average training duration for REINFORCE Policy: 177.47 seconds
Average training duration for REINFORCE with Baseline Policy: 205.01 seconds
Average training duration for REINFORCE with actor critic policy: 396.09 seconds
Average training duration for REINFORCE with actor critic big net policy: 428.69 seconds
```





4. **How Are Rewards Distributed Over Time for the "Best" Model of Each Policy?**



Key

Insights:

- Implementing the actor critic policy with the initial value network resulted in nearly four times more episodes required to complete the task compared to the REINFORCE with baseline. However, expanding the value network reduced the average episodes needed by half of that required for the REINFORCE with baseline.
- The duration distribution for the actor critic with the initial value network showed greater variance than that of the larger network, and both actor critic variants recorded longer durations overall.
- The actor critic policy exhibited more stable policy network losses over time compared to those of the REINFORCE and REINFORCE with baseline. Moreover, a larger value network size resulted in reduced losses compared to the standard value network.
- Integrating the actor critic policy increased the average time taken to solve the game. Enlarging the network size further extended the duration, though this was inversely related to the number of episodes needed to solve. The distribution analysis revealed that the variance is much higher for the actor critic with the larger network than with the smaller one.
- The reward trend for the "Best of REINFORCE with actor critic policy" shown in the graph is marked by considerable volatility and fluctuations, likely contributing to its lower performance, while the other policies demonstrate steadier and more consistent reward trajectories over time.