CS6700: Reinforcement Learning (Jan-May 2024) Programming Assignment 2

Vivek Sivaramakrishnan NS24Z170 Pooja Pandey NS24Z172

Table of contents

1 DDQN

We implement a `ReplayBuffer` and an agent following an ϵ-greedy exploration strategy for our DQN architecture. A 4 layer neural network of the following dimension is used:

$$\text{state\_size} \times 128 \times 64 \times (\text{no\_of\_actions}, 1) \times \text{no\_of\_actions}$$

The $(\text{no\_of\_actions}, 1)$ corresponds to the advantage and value streams, in accordance to the Dueling DQN scaffolding. These are further combined (fixed combination) as dictated by the types mentioned in the assignment.

1.1 DDQN Type-1 Implementation

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|A|}\sum_{a' \in A} A(s,a';\theta)\right)$$

From `agent_ddqn.py`, lines 148-182:

~~~
```{python}
class DuelingQNetwork1(nn.Module):
    def __init__(self, state_size...
    ...
    def forward(self, state):
        """Build a network that maps state -> action values.
        Uses Type-1 Dueling Architecture"""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))

        advantage = self.advantage(x)
        value = self.value(x)

        return value + advantage - advantage.mean()
```
~~~

1.2 DDQN Type-2 Implementation

$$Q(s,a,\theta) = V(s,\theta) + \left(A(s,a,\theta) - \max_{a' \in A} A(s,a',\theta)\right)$$

We reinherit the same DuelingQNetwork class and update the `forward` function (from `.mean()` to `.max()`)

From `agent_ddqn.py`, lines 184-195:

~~~~
```{python}
class DuelingQNetwork2(DuelingQNetwork1):

  def forward(self, state):
      """Build a network that maps state -> action values.
      Uses Type-2 Dueling Architecture"""
      x = F.relu(self.fc1(state))
      x = F.relu(self.fc2(x))

      advantage = self.advantage(x)
      value = self.value(x)

      return value + advantage - advantage.max()
```
~~~~

1.3 Hyperparameter Tuning and Results

We use the Optuna framework to automate our hyperparameter search. You can use optuna-dashboard to view a report of all the runs (stored in `hpt-results.db`).

Note: The nature of the environments `CartPole-v1` and `Acrobot-v1` is such that the optimal policy will always achieve a constant return. Hence minimizing the regret of an instance is equivalent to maximizing its mean episodic reward:

~~~~
```{python}
episode_rewards = run_experiment(...)
regret = -1 * episode_rewards

return regret.mean()
```
~~~~

We train our instance for 100 episodes in the tuning step. The best hyperparameter setting is picked after evaluating and ranking the average regret over 100 candidate model instances.

1.3.1 CartPole-v1

The following hyperparameters were tuned:

Name	Type-1 Optimal	Type-2 Optimal
`BATCH_SIZE`	128	64
`OPTIMIZER_NAME`	`RMSprop`	`RMSprop`
`LR`	0.002673	0.003958
`UPDATE_EVERY`	25	28
`EPSILON`	0.094805	0.084795
`MAX_GRAD_NORM`	0.984049	0.483248

1.3.1.1 Results plot and inferences

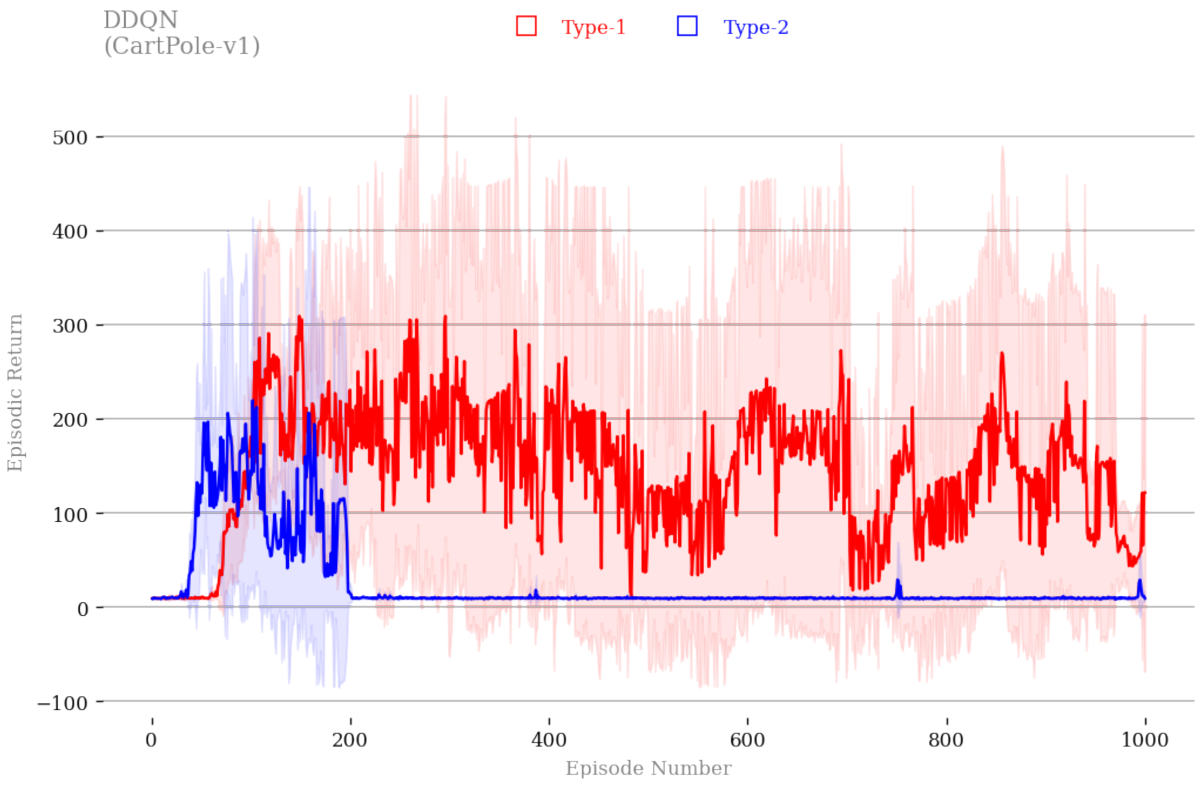

Figure 1: DDQN on CartPole-v1

Even though both algortihms solve CartPole-v1 (achieve returns > 194), the algorithms, after 200 episodes, undergo forgetting, becoming worser over time. It is unclear as to why this is happening (perhaps due to the 100-episode hyperparameter tuning). We see that Type-1 is able to re-learn after a brief period of forgetting, but Type-2 is never able to. In the constrained $[0, 200]$ episode window, we see that Type-1 has a quicker rate of learning and therefore higher sample-efficiency, in contrast to Type-2.

1.3.2 Acrobot-v1

Name	Type-1 Optimal	Type-2 Optimal
BATCH_SIZE	128	64
OPTIMIZER_NAME	Adam	RMSprop
LR	0.000565	0.000665
UPDATE_EVERY	24	23
EPSILON	0.057214	0.001148
MAX_GRAD_NORM	3.589517	4.944265

1.3.2.1 Results plot and inferences

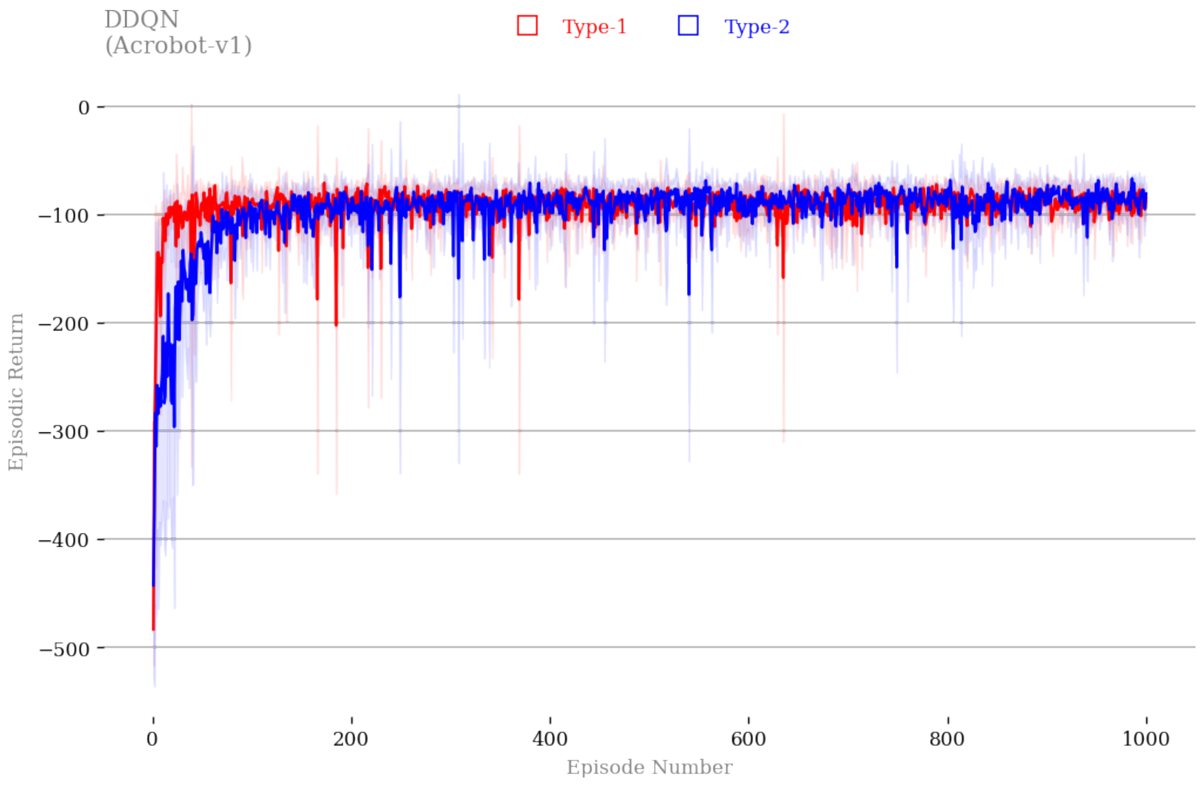

Figure 2: DDQN on Acrobot-v1

We observe that Type-1 converges to the optimal policy (returns around −100) faster than Type-2. Both types show relatively low variance. After around 100 episodes, both tend to face slight deviations, but quickly return back to the −100 *position*.

2 REINFORCE

A 3 layer neural network of the following dimension is used for **REINFORCE Type-1**:

$$\text{state\_size} \times 128 \times \text{no\_of\_actions}$$

The output layer is a probability distribution over the action space.

For **Type-2**, an additional head is added to estimate the state-value function used as a baseline (shared-network AC). One-step td error (TD(0)) is used as its respective loss function:

$$\text{state\_size} \times 128 \times (\text{no\_of\_actions}, 1)$$

2.1 REINFORCE Type-1 Update

$$\theta = \theta + \alpha G_t \frac{\nabla_\pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}$$

From `reinforce.py`, lines 50-55:

~~~
```{python}
for log_prob, R in zip(policy.saved_log_probs, returns):
    policy_loss.append(-log_prob * R)
optimizer.zero_grad()
policy_loss = torch.cat(policy_loss).sum()
policy_loss.backward()
optimizer.step()
```
~~~

2.2 REINFORCE Type-2 Update

From `actorCritic.py`, lines 58-66:

$$\theta = \theta + \alpha(G_t - V(S_t;\Phi))\frac{\nabla_\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

~~~
```{python}
for (log_prob, value), R in zip(saved_actions, returns):
    advantage = R - value.item()
    policy_losses.append(-log_prob * advantage)
    value_losses.append(F.smooth_l1_loss(value, torch.tensor([R])))

optimizer.zero_grad()
loss = torch.stack(policy_losses).sum() + torch.stack(value_losses).sum()
loss.backward()
optimizer.step()
```
~~~

2.3 Hyperparameter Tuning and Results

We train our instance for 500 episodes in the tuning step. The best hyperparameter setting is picked after evaluating and ranking the average regret over 70 candidate model instances.

2.3.1 CartPole-v1

The following hyperparameters were tuned:

Name	Type-1 Optimal	Type-2 Optimal
`OPTIMIZER_NAME`	`Adam`	`RMSprop`
`LR`	`0.009158`	`0.003654`

2.3.1.1 Results plot and inferences

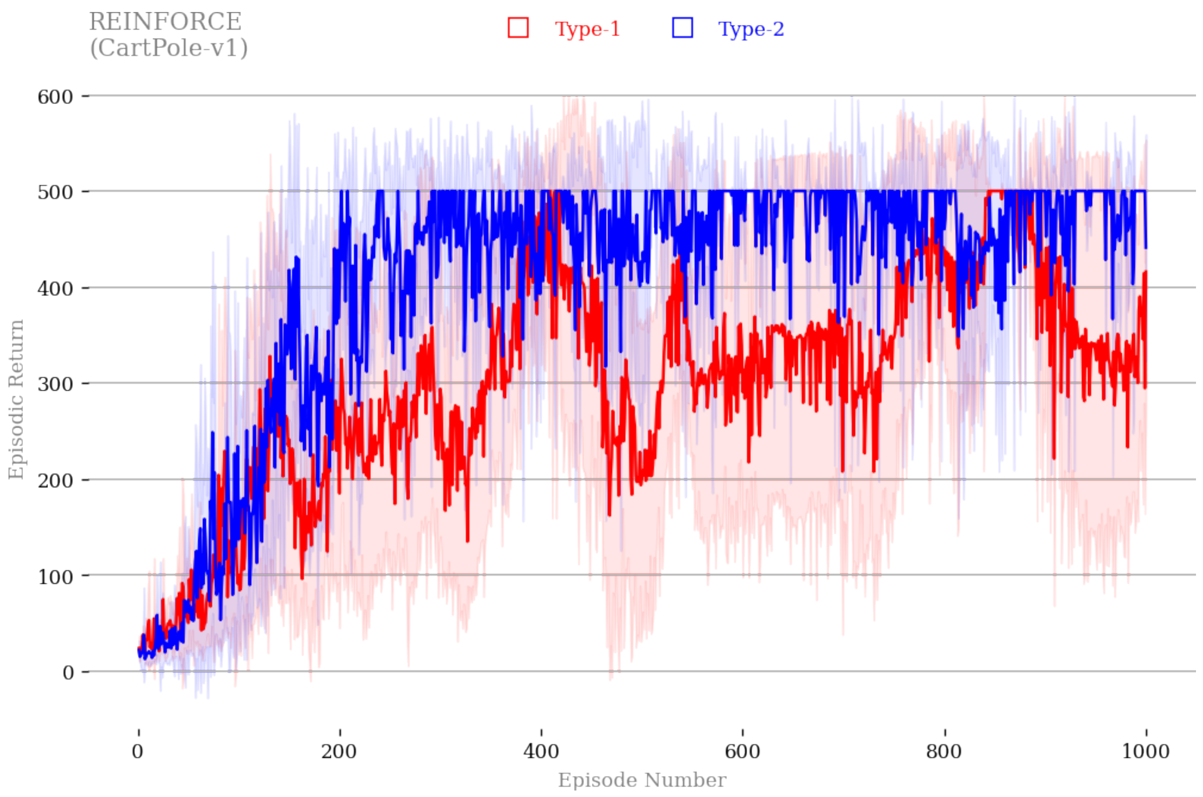

Figure 3: REINFORCE on CartPole-v1

REINFORCE w/ Baseline (Type-2) touches 500 (maxiumum reward in CartPole-v1) after around 200 episodes, where as REINFORCE w/o Baseline (Type-1) touches a little after 400. Both algorithms experience catastrophic forgetting, but Type-1 does to a greater level; Type-2 takes around 20 more episodes to comeback/re-learn, but Type-1 around 400 more. The presence of the baseline therefore could be playing an important factor in considerably reducing the variance in the policy gradients of the model instances.

2.3.2 Acrobot-v1

Name	Type-1 Optimal	Type-2 Optimal
`OPTIMIZER_NAME`	`Adam`	`RMSprop`
`LR`	`0.054455`	`0.005877`

2.3.2.1 Results plot and inferences

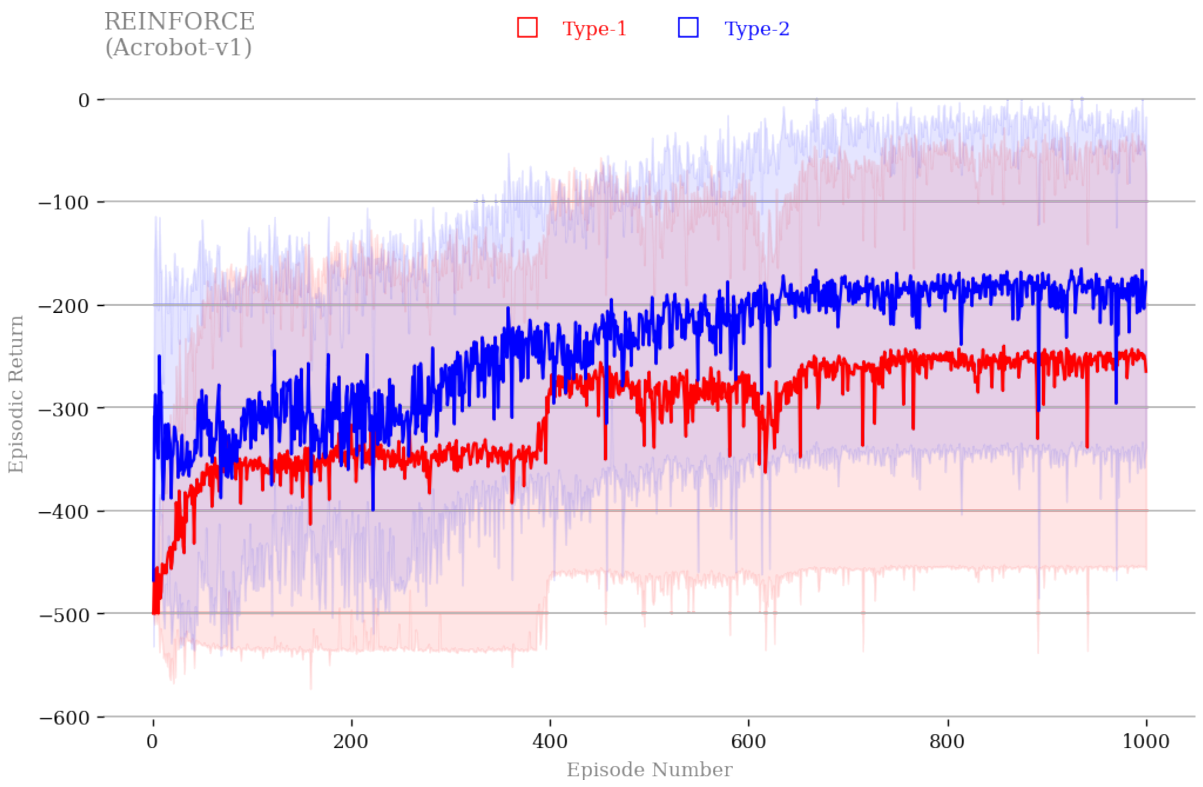

Figure 4: REINFORCE on Acrobot-v1

The model performance of REINFORCE is quite noisy as shown by the huge fill-in area (width $\pm 1\sigma$). We see that REINFORCE w/ Baseline (Type-2) gradually climbs up inching towards the optimal policy, with reducing variance. In contrast, REINFORCE w/o Baseline (Type-1) has a *jerky* step-up learning which slows down its learning process, and also has considerable variance. The plot therefore *reinforces* the importance of including a good (state-dependent) baseline, and how it assists in decreasing this variance.

3 Github Repo Link

Find Here