Assignment 9

Reinforcement Learning – Hanging Around: The Inverted Pendulum Problem

Authors

Mangesh Sakordekar
Minati Alphonso

Goal

Use reinforcement learning to create a policy for the inverted pendulum problem.

Basic Policy

In this policy, we try to stabilize the pendulum by applying torque in the opposite direction of the angular velocity when the pole is close to a 0 degree angle. Otherwise a force proportional to the angle is applied in the same direction of the angular velocity to move the pole closer towards the top. The implementation can be seen below:

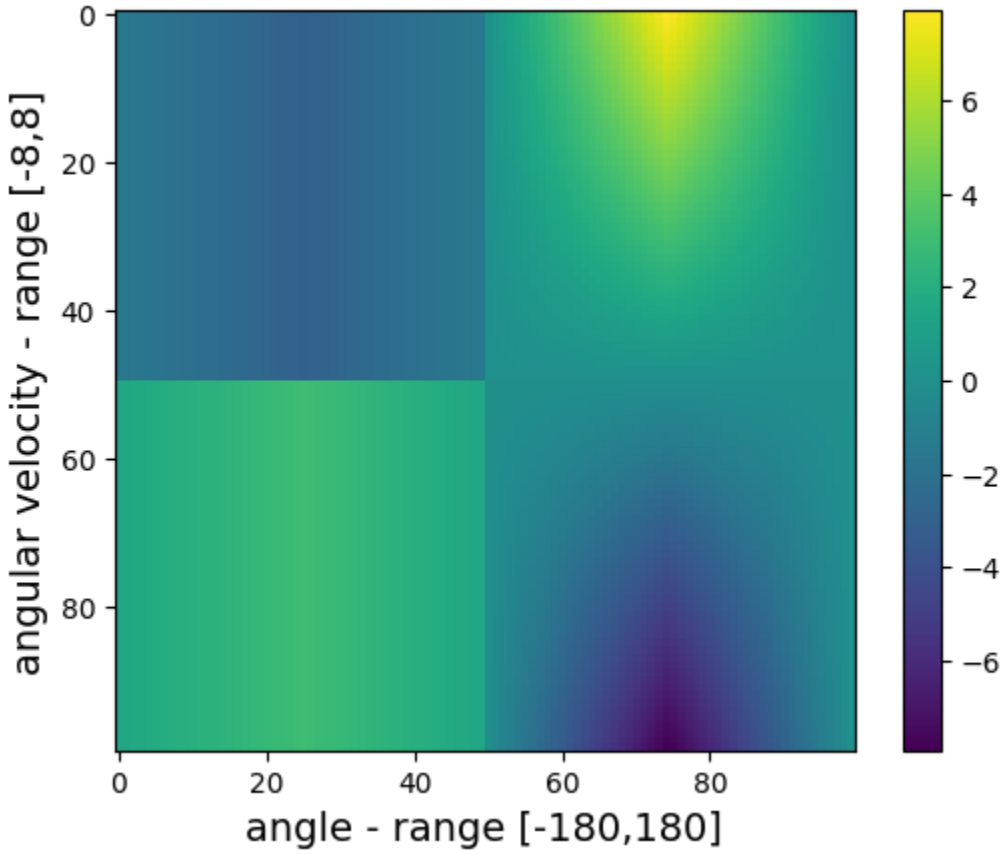
```
def basic_policy(obs):
  angle = -180/np.pi*np.arctan2(obs[0],obs[1]) + 90
  if angle > 180: angle -= 360
  ang_vel = 180/np.pi*obs[2]
  # if within 45 degrees of top exert proportional control towards top
  if angle == 0:
    torque = -ang_vel
  elif abs(angle) <= 90: # in top half
    torque = -(90-abs(angle))*ang_vel/90
  else:
    torque = 1.5 * (ang_vel/abs(ang_vel)) * abs(angle)/90

  action = np.array((torque,))

  return action
```

Since the environment clips the values in the range of [-2,2], we did not clip the torque values to avoid a complicated if-else tree.

Using the above policy, a gradient plot was made in the range [-180, 180] degrees along the X axis and [-8, 8] angular velocity along the Y axis.



Neural Network

For the Neural nets model, we adapted the code from the chapter 18 notebook from the textbook. We updated the code to use the rewards to update the model variables.

```
def play_one_step(env, obs, model, loss_fn):
    theta = np.arctan2(obs[0],obs[1])
    theta_dt = obs[2]
    with tf.GradientTape() as tape:
        action = 2 * model(obs[np.newaxis])
        reward = -(theta**2 + 0.1 * theta_dt**2 + 0.001 * action[0]**2)


    grads = tape.gradient(reward, model.trainable_variables)
    obs, reward, done, info = env.step(action)
    return obs, reward, done, grads
```

We used a simple model with tanh as the activation function. We opted for tanh to preserve the signs of the angular velocity.

```
# extra code – let's create the neural net and reset the environment, for
#              reproducibility

tf.random.set_seed(42)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(4, activation="tanh"),
    tf.keras.layers.Dense(1, activation="tanh"),
])

obs, info = env.reset(seed=42)
```

We trained the model for 5 epochs which yielded a mean rewards value of -1188.41. The highest reward value observed in all the training runs was around -950.

```
for iteration in range(n_iterations):
    all_rewards, all_grads = play_multiple_episodes(env, n_episodes_per_update, n_max_steps, model, loss_fn)

    # extra code – displays some debug info during training
    total_rewards = sum(map(sum, all_rewards))
    print(f"\rIteration: {iteration + 1}/{n_iterations},",
          f" mean rewards: {total_rewards / n_episodes_per_update}", end="")

    all_final_rewards = discount_and_normalize_rewards(all_rewards,
                                                       discount_factor)
    all_mean_grads = []
    for var_index in range(len(model.trainable_variables)):
        mean_grads = tf.reduce_mean(
            [final_reward * all_grads[episode_index][step][var_index]
             for episode_index, final_rewards in enumerate(all_final_rewards)
                 for step, final_reward in enumerate(final_rewards)], axis=0)
        all_mean_grads.append(mean_grads)

    optimizer.apply_gradients(zip(all_mean_grads, model.trainable_variables))
```

```
Iteration: 5/5,  mean rewards: [-1188.4125]
```

This model did not perform well as the pendulum was never close to being in a balanced state. The pendulum tries to move towards the target - 0 degree angle, but once it reaches there, it starts to spiral around continuously with no signs of stopping.

Using the neural nets policy, a gradient plot was made in the range [-180, 180] degrees along the X axis and [-8, 8] angular velocity along the Y axis.