Deep Learning and Practice

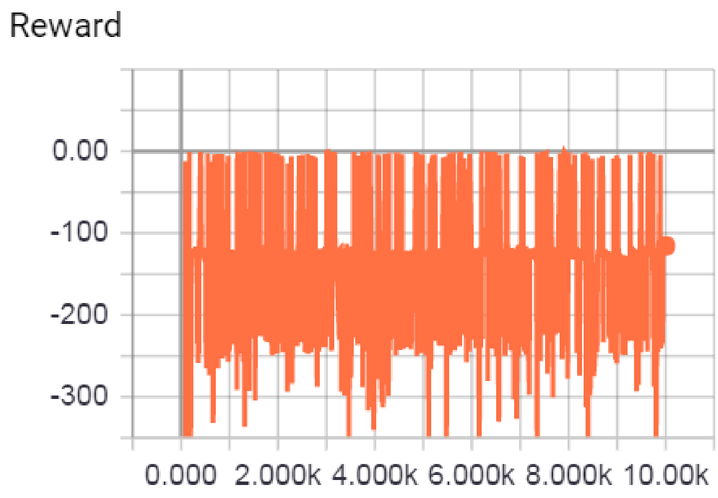
Lab 10: Deep Deterministic Policy Gradient

李韡

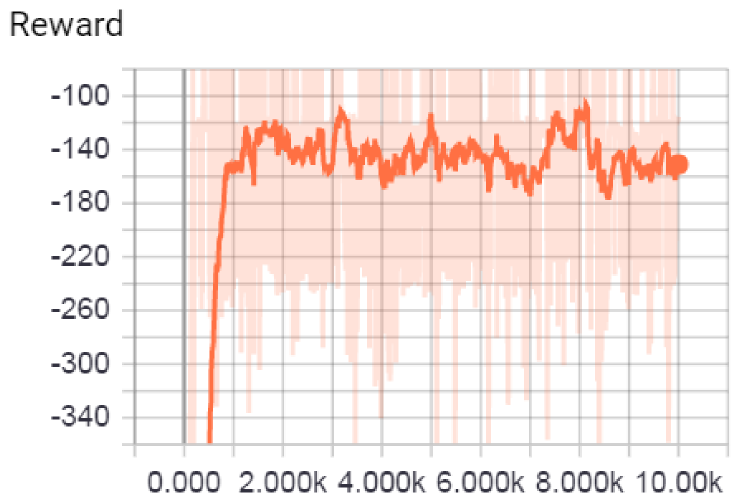
0556157

fm.bigballon@gmail.com

Episode rewards



Smoothing 0.00



Smoothing 0.95

Explain the mechanism of critic updating

```
# loss & optimize op
self.loss = tflearn.mean_square(self.target_q_value, self.out)
self.optimize = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)
```

```
def train(self, X, action, target_q_value):
    return self.sess.run([self.out, self.optimize], feed_dict={
        self.input: X,
        self.action: action,
        self.target_q_value: target_q_value
    })
```

```
# update critic
predicted_q_value, _ = behavior_critic.train(s_batch, a_batch, np.reshape(y_i, (MINIBATCH_SIZE, 1)))
```

Calculate the loss (MSE) [loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$]

Using AdamOptimizer to minimize the loss.

s_batch is the set of states,

a_batch is the set of action,

self.target_q_value is the set of y [$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$]

self.out is the predictions of critic network.

Describe the algorithm of actor updating

```
# -----------------------
action = behavior_actor.predict(s_batch)
gradient = behavior_critic.action_gradients(s_batch,action)
behavior_actor.train(s_batch,gradient[0])

# soft update target networks
sess.run([actor_update_ops, critic_update_ops])
```

1. Use behavior_actor to predict actions [param: s_batch return:action]

```
def predict(self, X):
    return self.sess.run(self.scaled_out, feed_dict={
        self.input: X
    })
```

2. Use behavior critic to compute action gradients [param: s_batch, action return:gradient]

```
def action_gradients(self, X, action):
    return self.sess.run(self.action_grads, feed_dict={self.input: X, self.action: action})
```

compute the partial derivatives of self.out with respect to self.action

```
self.action_grads = tf.gradients(self.out, self.action)
```

3. Update behavior actor

```
def train(self, X, a_gradient):
    self.sess.run(self.optimize, feed_dict={
        self.input: X,
        self.action_gradient: a_gradient
    })
```

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Using tf.gradients to combine the gradients.

```
self.actor_gradients = tf.gradients(self.scaled_out, self.network_params, -self.action_gradient)
```

Using apply_gradients to use the gradients

```
self.optimize = tf.train.AdamOptimizer(self.learning_rate).\
    apply_gradients(zip(self.actor_gradients, self.network_params))
```

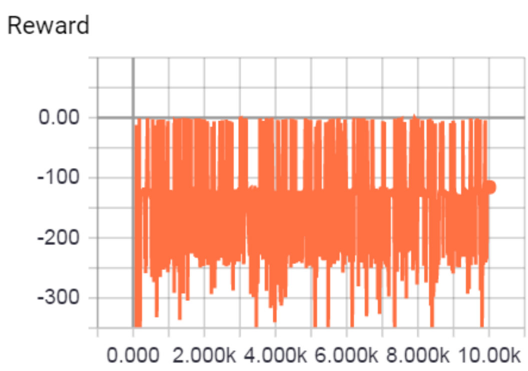
Update the target networks:

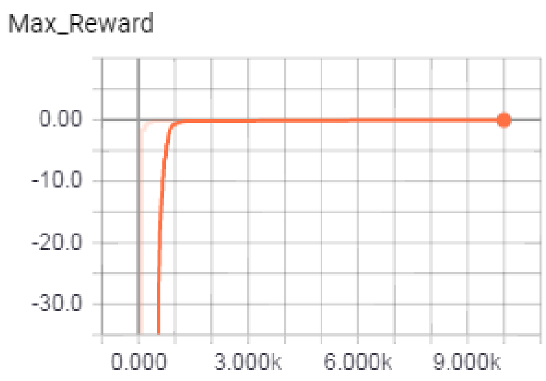
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

```
def soft_update_ops(sess, target_net, behavior_net):
    update_ops = []
    for behavior_v, target_v in zip(behavior_net.get_params(), target_net.get_params()):
        # soft update
        op = target_v.assign(TAU*behavior_v + (1.0 - TAU)*target_v)
        update_ops.append(op)

    return update_ops
```

Performance – Highest episode reward





One video during the training process

See attachments[video-1495603800.mp4].

The last one tensorflow checkpoint file

See attachments[checkpoint.zip].