Deep Learning and Practice

Lab 9: Deep Q Network

李韡

0556157

fm.bigballon@gmail.com

Episode rewards

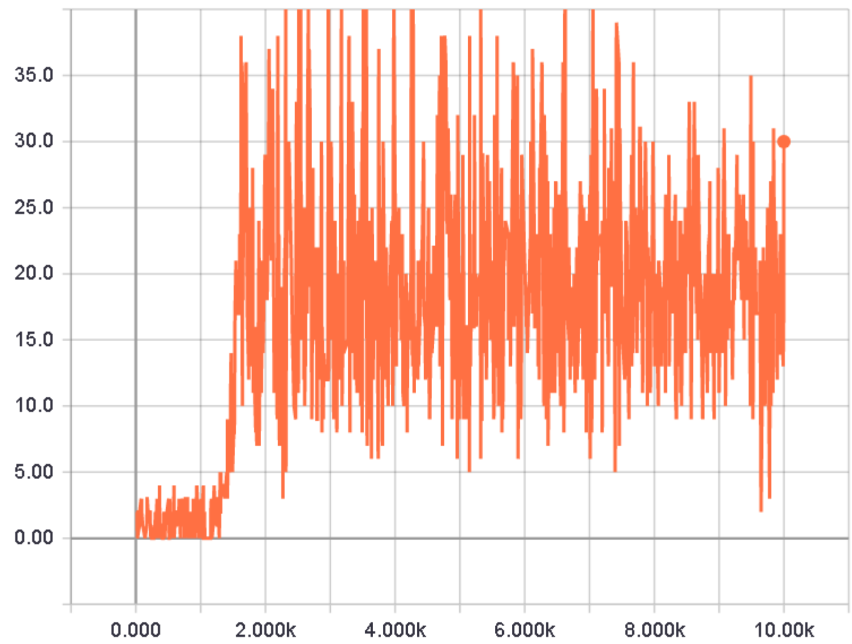
It cost about 30 hours to train the model.

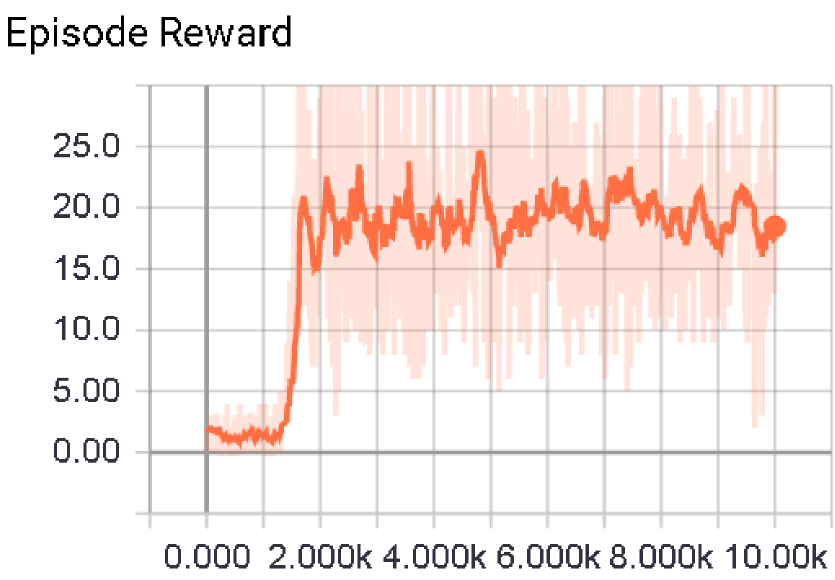
The first time the highest episode reward during training is 78, but I forget to save the plot and logs.

The following figures is the second time's plot:

The highest episode reward during training is 60.

And the average episode reward is about 23.





Deep Q network implementation

■ Network Structure

According to the Implementation Details, we need to set up 3 convolution layers.

- conv1
- conv2
- conv3

Then we need two fully connected layers

- fc1
- predictions

```python
def _build_network(self):
    self.X_pl = tf.placeholder(shape=[None, 84, 84, 4], dtype=tf.uint8, name="X")
    self.y_pl = tf.placeholder(shape=[None], dtype=tf.float32, name="y")
    self.actions_pl = tf.placeholder(shape=[None], dtype=tf.int32, name="actions")

    X = tf.to_float(self.X_pl) / 255.0

    conv1 = tf.contrib.layers.conv2d(X, 32, 8, 4, activation_fn=tf.nn.relu)
    conv2 = tf.contrib.layers.conv2d(conv1, 64, 4, 2, activation_fn=tf.nn.relu)
    conv3 = tf.contrib.layers.conv2d(conv2, 64, 3, 1, activation_fn=tf.nn.relu)
```

3 conv

```python
    # Fully connected layers
    flattened = tf.contrib.layers.flatten(conv3)
    fc1 = tf.contrib.layers.fully_connected(flattened, 512)
    self.predictions = tf.contrib.layers.fully_connected(fc1, len(VALID_ACTIONS))
```

2 fc

```python
    # Get the predictions for the chosen actions only
    batch_size = tf.shape(self.X_pl)[0]
    gather_indices = tf.range(batch_size) * tf.shape(self.predictions)[1] + self.actions_pl
    self.action_predictions = tf.gather(tf.reshape(self.predictions, [-1]), gather_indices)

    # Calcualte the loss
    self.losses = tf.squared_difference(self.y_pl, self.action_predictions)
    self.loss = tf.reduce_mean(self.losses)

    # Optimizer Parameters from original paper
    self.optimizer = tf.train.RMSPropOptimizer(0.00025, 0.99, 0.0, 1e-6)
    self.train_op = self.optimizer.minimize(self.loss, global_step=tf.contrib.framework.get_or_create_global_step())

    # Summaries for Tensorboard
    self.summaries = tf.summary.merge([
        tf.summary.scalar("loss", self.loss),
        tf.summary.histogram("loss_hist", self.losses),
        tf.summary.histogram("q_values_hist", self.predictions),
        tf.summary.scalar("max_q_value", tf.reduce_max(self.predictions))
    ])
```

■ Loss function

1. calculate the [action_predictions]
2. then calculate the squared_difference [losses]of(y and action_predictions)
3. calculate the average of losses [loss]
4. using RMSPropOptimizer to minimize

- **Implement update_target_network()**

```python
def update_target_network(sess, behavior_Q, target_Q):

    e1_params = [t for t in tf.trainable_variables() if t.name.startswith(behavior_Q.scope)]
    e1_params = sorted(e1_params, key=lambda v: v.name)
    e2_params = [t for t in tf.trainable_variables() if t.name.startswith(target_Q.scope)]
    e2_params = sorted(e2_params, key=lambda v: v.name)

    update_ops = []
    for e1_v, e2_v in zip(e1_params, e2_params):
        op = e2_v.assign(e1_v)
        update_ops.append(op)

    sess.run(update_ops)
```

1. using [tf.trainable_variables]to returns all variables created with trainable=True,
2. then using a list update_ops to save all assign operators,
3. let sess run the update_ops to finish assigned between behavior_Q and target_Q

- **Explain how you implement the training process of deep Q learning**

- **Populate replay memory**

```python
action_probs = policy(sess, state, epsilons[min(total_tmp, EXPLORE_STPES-1)])
action = np.random.choice(np.arange(len(action_probs)), p=action_probs)
```

Using linspace to generate anneal epsilon,
Using epsilon_greedy_policy to select action,
Playing the game util replay_memory less than INIT_REPLAY_MEMORY_SIZE

- **Select actions**

With probability ε select a random action a_t
otherwise select $a_t = \mathrm{argmax}_a Q(\varphi(s_t), a; \theta)$

- **Update Epsilon**

```python
epsilons = np.linspace(INITIAL_EPSILON, FINAL_EPSILON, EXPLORE_STPES)
```

When epsilon increase ,

```python
epsilon = epsilons[min(total_t, EXPLORE_STPES-1)]
```

■ **Prepare minibatch for network update**

```
samples = random.sample(replay_memory, BATCH_SIZE)
states_batch, action_batch, reward_batch, next_states_batch, done_batch = map(np.array, zip(*samples))
```

Using random.sample to sample a minibatch from the replay memory (BATCH_SIZE = 32)

Using map to mapping samples to all informations(states actions rewards next_states and done)

Performance – Highest episode reward during training

Training 1: the highest episode reward during training is 78.

Training 2: the highest episode reward during training is 60.