Alphago 实验报告

周韧哲 (181220076、zhourz@smail.nju.edu.cn)

(南京大学 计算机科学与技术系，南京 210093)

摘 要: alphago 复现

关键词: 强化学习，蒙特卡洛树搜索

1 引言

在这次的实验中，通过阅读 paper 与博客掌握了 alphago 的一些实现细节并将其部分实现。

2 具体实现

围棋环境中的 MCTS 主要由以下几个部件组成：

1、 Select: 从根节点开始，在树内选择 Q+U 值最大的孩子节点直到叶节点，u 依赖于节点的先验概率 p 和节点的访问次数 N 和节点的父节点的访问次数 N'，先验概率 P 从策略网络中获得，Q 由值函数网络初始化，在搜索过程中不断迭代而来。

2、 Expand: 如果孩子节点的访问次数超过了一个给定的阈值，则将孩子节点扩展，即获取节点的可执行动作执行后生成一些新的子节点。

3、 Rollout: 从孩子节点出发，通过 rollout 网络不断选择动作在围棋环境中执行直到游戏结束，得到游戏结果 reward

4、 Backup：将 rollout 得到的 reward，从孩子节点出发沿着 select 的路径反向传播，更新路径上节点的访问次数和 Q 值

所以在围棋环境中实现 MCTS 的思路就是：在和对手博弈时，当对手落子后，将当前棋盘状态传入 MCTS 作为根节点，然后不断从根节点出发进行 select、(expand)、rollout、backup，直到搜索的时间超出了给定的限度，最后返回访问次数最多的节点的动作即可。

我的 MCTS 框架参考了 https://github.com/HardcoreJosh/JoshieGo 上的实现。将 Node 类与 MCTS 类写在了 agent/agent.py 中，具体如下：

```
class Node(object):
    def __init__(self, parent, env,timestep, action, prior,Q):
        self.parent=parent
        self.env=copy.deepcopy(env)
        self.state=timestep
        self.last_action = action
        self.current_player=self.state.observations["current_player"]
        self.legal_actions = self.state.observations["legal_actions"][self.current_player]
        self.children = []
        self.Q = Q
        self.N = 0
        self.prior=prior
```

Node 类代表搜索树中的一个节点，它有 parent、env(围棋环境)、state、last_action、current_player、legal_actions、children、Q、N、prior 这些属性。

```
class MCTS(object):
    def __init__(self, root,policy_net,value_fn,rollout_fn, env,time_limit=20):
        self.time_limit = time_limit
        self.root = root
        self.policy_fn = policy_net.policy_fn
        self.value_fn=value_fn
        self.rollout_fn=rollout_fn
    def start(self):
        start = time.time()
        while True:
            self.search()
            if time.time() - start > self.time_limit:
                self.root.children.sort(key=lambda c: -c.N)
                return self.root.children[0].last_action
```

MCTS 类有 root、time_limit、policy_fn、value_fn、rollout_fn 属性，_fn 代表的是网络的函数。Start 函数在时间限制范围内搜索，若超出时间限制范围，则返回 root 节点下 MCTS 搜索出的应该执行的动作。

```
    def search(self,expand_limit=1):
        node=self.root
        while len(node.children)!=0:
            node=self.select(node)
        if node.N>expand_limit or node.parent==None:
            self.expand(node)
            node=self.select(node)
        reward=self.rollout(node)
        self.backup(node,reward)
```

search 函数从 root 开始搜索，不断选择直到孩子节点，expand(如果有)后进行 rollout 得到 reward，然后将其反向更新。

```
    def select(self,node):
        if node.current_player == 0:
            node.children.sort(key=lambda child: child.Q + (child.prior+0.1)* np.sqrt(node.N) / (1 + c
hild.N))
        else:
            node.children.sort(key=lambda child: -
child.Q + (child.prior+0.1)* np.sqrt(node.N) / (1 + child.N))
        return node.children[-1]
```

select 函数选择 Q+U 值最大的孩子节点，常数项是为了防止 0 的出现。

```
    def expand(self, node):
        for action in node.legal_actions:
            envs=copy.deepcopy(node.env)
            timestep=envs.step(action)
            prior=self.evaluate_P(timestep)
```

```
            p=prior[action]
            Q=self.evaluate_Q(timestep)
            child = Node(node,envs,timestep,action,p,Q)
            node.children.append(child)
    def evaluate_P(self, timestep):
        return self.policy_fn(timestep,timestep.observations["current_player"])
    def evaluate_Q(self, timestep):
        return self.value_fn(timestep,timestep.observations["current_player"])
```

expand 函数首先获取 node 的可执行动作，然后将 node 的围棋环境复制一份后执行动作得到 timestep，将 timestep 送入 policy net 得到先验概率，送入 value net 得到 Q 值，然后将子节点添加到搜索树中。

```
    def rollout(self,node):
        timestep=node.state
        env=copy.deepcopy(node.env)
        legal_actions=node.legal_actions
        while not timestep.last():
            c_player=timestep.observations["current_player"]
            action=self.rollout_fn(timestep,c_player)
            timestep=env.step(action)
        return timestep.rewards[node.current_player]
```

rollout 函数根据 rollout 的策略选择一个动作并在环境 env 中执行，直到游戏结束，返回玩家的 rewards

```
    def backup(self,node,reward):
        while node.parent!=None:
            node.N += 1
            node.Q = (float(node.N-1) * node.Q + reward) / node.N
            node=node.parent
```

backup 函数沿着叶子节点到根节点的路径更新 N 和 Q 值。

Dqn 算法中添加了 policy_fn 函数用作接口给出动作的先验概率。

```
     def policy_fn(self,time_step, player_id):
        player_id = time_step.observations["current_player"]
        info_state = time_step.observations["info_state"][player_id]
        legal_actions = time_step.observations["legal_actions"][player_id]
        epsilon = self._get_epsilon(is_evaluation=True)
        _ , probs = self._epsilon_greedy(info_state, legal_actions, epsilon)
        return probs
```

在我的(残疾版)实现中，我用随机下棋的 Agent 和用 MCTS 结合 dqn policy net 与随机 rollout net 与随机 value net 的 Agent 对弈：

```
def main(unused_argv):
    begin = time.time()
    env = Go()
    info_state_size = env.state_size
    num_actions = env.action_size
    agentR=agent.RandomAgent(0)
    hidden_layers_sizes = [int(l) for l in FLAGS.hidden_layers_sizes]
```

```
kwargs = {
    "replay_buffer_capacity": FLAGS.replay_buffer_capacity,
    "epsilon_decay_duration": int(0.6*FLAGS.num_train_episodes),
    "epsilon_start": 0.8,
    "epsilon_end": 0.001,
    "learning_rate": 1e-3,
    "learn_every": FLAGS.learn_every,
    "batch_size": 128,
    "max_global_gradient_norm": 10,
}
ret = [0]
max_len = 2000
with tf.Session() as sess:
    dqn=DQN(sess, 0, info_state_size,num_actions, hidden_layers_sizes, **kwargs)
    dqn.restore("saved_model/10000")
    for ep in range(10):
        print("start mcts train ep"+str(ep))
        time_step = env.reset()
        while not time_step.last():
            player_id = time_step.observations["current_player"]
            if player_id == 0:  #用 MCTS
                root=Node(None,env,time_step,None,0,0)
                mcts=MCTS(root,dqn,random_value_net,random_rollout_net,env,time_limit=5)
                action_list=mcts.start()
            else:
                agent_output = agentR.step(time_step).action
                action_list = agent_output
            time_step = env.step(action_list)
        print(time_step.rewards)
print('Time elapsed:', time.time()-begin)
```

胜率在 70%左右：

```
Use standard file APIs to check for files with this prefix.
I0115 20:49:13.441445  4356 saver.py:1280] Restoring parameters from s
[-1, 1]
[1, -1]
[-1, 1]
[1, -1]
[1, -1]
[-1, 1]
[1, -1]
[1, -1]
[1, -1]
[1, -1]
Time elapsed: 824.1105604171753
```

限于时间和个人能力所限，我没能进一步地探索作业的其他内容，希望在寒假的剩余时间里可以更深入地学习 tensorflow 和 Alphago。

3 致谢

感谢 xzh 大佬的开源代码，让我弄懂了不少实现过程。