Combining Policy Gradient and Q-Learning

Group10 : 吳程畯、林浩君、許承壹

Recap: Policy Gradient and Q-learning

- Policy Gradient:

We can parameterize the policy directly and attempt to improve it via gradient ascent on the objective J given by

$$\nabla_\theta J(\pi) = \mathop{\mathbf{E}}_{s,a} Q^\pi(s,a)\nabla_\theta \log \pi(s,a)$$

- Q-learning:

In value based RL, we approximate the Q-values using a function approximator(typically neural network), and follow the best actions along the maximum Q-values of each states. **Theoretically, we apply Bellman contraction operator to find the optimal Q-value.**

$$\mathbf{E}_{s,a}(\mathcal{T}^\pi Q(s,a;\theta) - Q(s,a;\theta))\nabla_\theta Q(s,a;\theta)$$

Comparison

- What are the advantages and disadvantages of Policy Gradient and Q-Learning method respectively?

	Advantages	Disadvantages
Policy Gradient	• Better convergence properties • Effective in high-dimensional or continuous action spaces • Can learn stochastic policies	• Hard for following off-policy • Sample inefficiency • Usually converge to local optimum • Hard for learning deterministic policies • Evaluating a policy is inefficient and high variance
Q-Learning	• Can apply off-policy • Sample efficiency	• Training process is relatively slow • Troubling with continuous action space

Combining These Two Family Together: PGQL(Policy Gradient Q-Learning)

- From the previous slide, we see there are many disadvantages to both two RL-algorithm families. **So, is it possible to combine them together, and eliminate each other's disadvantages and exploit both their benefits?**
- This is exactly what this paper is trying to do!
- Based on the architecture of A3C, they tries to derive a variant of Q-value function induced by current behavior policy. **This method is called PGQL(Policy Gradient Q-Learning)**

Formal Definition of PGQL

We define a variant of Q-value using the policy as

$$\tilde{Q}^{\pi}(s,a) = \alpha(\log \pi(s,a) + H^{\pi}(s)) + V(s)$$

where V is parameterized by w, i.e. we obtain this value by training a neural network. And $H^{\pi}(s) = -\sum_a \pi(s,a)\log\pi(s,a)$ denotes the entropy of policy π, and $\alpha > 0$ is the regularization penalty parameter.

In short, they tries to **estimate Q-value by current policy, and value function**, and also **add entropy regularization** to make this estimation robust.

But, Does this formula have any guarantee to reach optimal Q-value?

Theory in Behind: Bellman Residual

By the optimal Bellman r-contraction operator $\mathcal{T}^\star$, we want to prove that there exist a fixed point $\tilde{Q}^{\pi_\alpha}$,which is the optimal Q-value function, and the **Bellman residual of the induced Q-values for the PGQL updates converges to 0**. Hence .we can **guarantee that our updating process is leading us to a nice convergence**.

Note: Bellman operator $\mathcal{T}^\pi$ is defined as

$$\mathcal{T}^\pi Q(s,a) = \mathop{\mathbf{E}}_{s',r,b}(r(s,a) + \gamma Q(s',b))$$

Optimal Bellman operator $\mathcal{T}^\star$ is defined as

$$\mathcal{T}^\star Q(s,a) = \mathop{\mathbf{E}}_{s',r}(r(s,a) + \gamma \max_b Q(s',b))$$

Q^{π_α} is the same as $\tilde{Q}^\pi$ in the previous page, and $\tilde{Q}^{\pi_\alpha}$ is the fixed point we want to obtain, they somehow confuse the notations when giving proof.

Proof: Bellman Residual converges to 0
Step1: Show That We Are Contracting to a Specific Point.

- **Ulitimate Target**: Show that $\lim_{\alpha\to 0} \|\mathcal{T}^\star Q^{\pi_\alpha} - Q^{\pi_\alpha}\| = 0$,where we would have $\tilde{Q}^{\pi_\alpha}$ as our fixed point.

First can write $\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha} = \eta(\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha})$ derived by this paper, hence, we can show

$$\begin{aligned}
\|\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| &= \eta\|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| \\
&= \eta\|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha} + \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| \\
&\leq \eta(\|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha}\| + \|\mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}Q^{\pi_\alpha}\|) \\
&\leq \eta(\|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha}\| + \gamma\|\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\|) \\
&\leq \eta/(1-\eta\gamma)\|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha}\|,
\end{aligned}$$

Therefore, we firstly prove that as $\alpha \to 0$, $\|\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| \to 0$, **this means that we are going to approach a point** $\tilde{Q}^{\pi_\alpha}$ if we keep updating Q^{π_α} , **but we still don't know whether** $\tilde{Q}^{\pi_\alpha}$ **is a fixed point or not.**

Step2: Show This Specific point Is a Fixed Point

By the previous proof(**A fixed point $\tilde{Q}^{\pi_\alpha}$ exists**), we can deduce that

$$\begin{aligned}
\|\mathcal{T}^\star\tilde{Q}^{\pi_\alpha} - \tilde{Q}^{\pi_\alpha}\| &= \|\mathcal{T}^\star\tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha} + \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha} + Q^{\pi_\alpha} - \tilde{Q}^{\pi_\alpha}\| \\
&\leq \|\mathcal{T}^\star\tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha}\| + \|\mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}Q^{\pi_\alpha}\| + \|Q^{\pi_\alpha} - \tilde{Q}^{\pi_\alpha}\| \\
&\leq \|\mathcal{T}^\star\tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha}\| + (1+\gamma)\|\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| \\
&< 3/(1-\eta\gamma)\|\mathcal{T}^\star\tilde{Q}^{\pi_\alpha} - \mathcal{T}^{\pi_\alpha}\tilde{Q}^{\pi_\alpha}\|,
\end{aligned}$$

which therefore also converges to 0 in the limit.

This means that $\tilde{Q}^{\pi_\alpha}$ is actually a fixed point. We cannot contract anymore.

Step3: Bellman Residual Approaches 0

So, By the previous two steps, we obtain:

$$\begin{aligned}
\|\mathcal{T}^\star Q^{\pi_\alpha} - Q^{\pi_\alpha}\| &= \|\mathcal{T}^\star Q^{\pi_\alpha} - \mathcal{T}^\star \tilde{Q}^{\pi_\alpha} + \mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \tilde{Q}^{\pi_\alpha} + \tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| \\
&\leq \|\mathcal{T}^\star Q^{\pi_\alpha} - \mathcal{T}^\star \tilde{Q}^{\pi_\alpha}\| + \|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \tilde{Q}^{\pi_\alpha}\| + \|\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| \\
&\leq (1+\gamma)\|\tilde{Q}^{\pi_\alpha} - Q^{\pi_\alpha}\| + \|\mathcal{T}^\star \tilde{Q}^{\pi_\alpha} - \tilde{Q}^{\pi_\alpha}\|,
\end{aligned}$$

As it implies that $\lim_{\alpha\to 0}\|\mathcal{T}^\star Q^{\pi_\alpha} - Q^{\pi_\alpha}\| = 0$.

So, we know that we have a nice convergence as we keep updating the variant of Q-value function Q^{π_α}.

ApplyingPGQL to Update NN Parameters

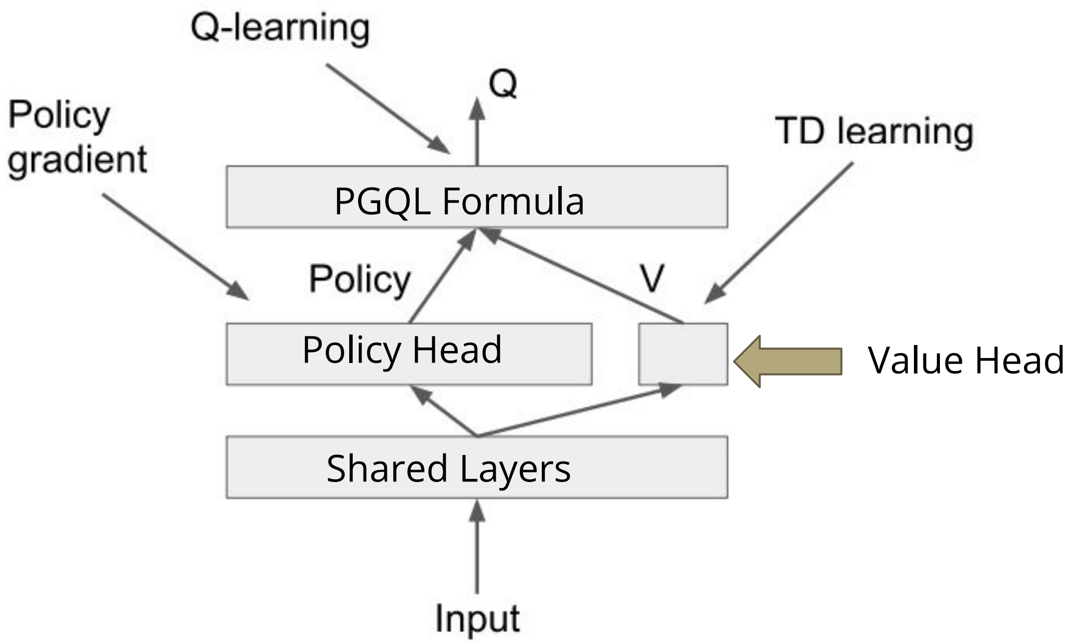
Since we know that at the fixed point the Bellman residual will be small for small α, we can consider updating the parameters to reduce the Bellman residual in a fashion similar to Q-learning, i.e.,

$$\Delta\theta \propto \mathop{\mathbf{E}}_{s,a}(\mathcal{T}^\star\tilde{Q}^\pi(s,a) - \tilde{Q}^\pi(s,a))\nabla_\theta \log \pi(s,a), \quad \Delta w \propto \mathop{\mathbf{E}}_{s,a}(\mathcal{T}^\star\tilde{Q}^\pi(s,a) - \tilde{Q}^\pi(s,a))\nabla_w V(s).$$

This is exactly Q-learning applied to a particular form of the Q-values, and can also be interpreted as an actor-critic algorithm with an optimizing critic.

In practice, we are leveraging between Q-learning and A3C algorithm.

Implementation Detail(1):Network Architecture

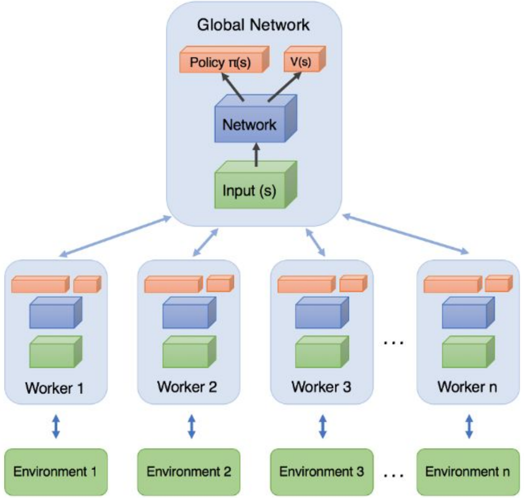


- We use **only one network** with shared layers, and one head for giving current **policy**, another head for giving **value**. Lastly, **The last layer is simply parameterized by the PGQL formula, without any network weights**.

Implementation Detail(2): Using The A3C Architecture

This is the **detail they didn't emphasize**, and hence, **we did have a hard time doing research** on **what neural network they exactly implemented**.

Precisely, they used the **A3C architecture**, which will have **a central shared model**, and use threads to **launch plenty of workers** to do the training process, and finally **update the weights from the workers back to the central shared model.**

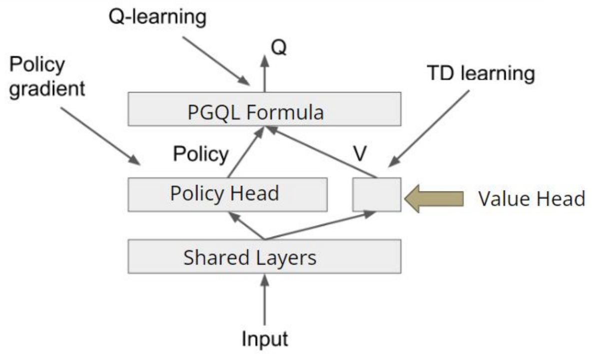


Code of Our implemented Neural Network

```python
class NNPolicy(nn.Module): # an actor-critic neural network
    def __init__(self, channels, memsize, num_actions):
        super(NNPolicy, self).__init__()
        self.conv1 = nn.Conv2d(channels, 32, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.conv4 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.gru = nn.GRUCell(32 * 5 * 5, memsize)
        self.critic_linear, self.actor_linear = nn.Linear(memsize, 1), nn.Linear(memsize, num_actions)

    def forward(self, inputs, train=True, hard=False):
        inputs, hx = inputs
        x = F.elu(self.conv1(inputs))
        x = F.elu(self.conv2(x))
        x = F.elu(self.conv3(x))
        x = F.elu(self.conv4(x))
        hx = self.gru(x.view(-1, 32 * 5 * 5), (hx))
        return self.critic_linear(hx), self.actor_linear(hx), hx

    def try_load(self, save_dir):
        paths = glob.glob(save_dir + '*.tar') ; step = 0
        if len(paths) > 0:
            ckpts = [int(s.split('.')[-2]) for s in paths]
            ix = np.argmax(ckpts) ; step = ckpts[ix]
            self.load_state_dict(torch.load(paths[ix]))
        print("\tno saved models") if step == 0 else print("\tloaded model: {}".format(paths[ix]))
        return step
```



Implementation Detail(3): Calculate Q from the policy

- This part correspond to the formula $\tilde{Q}^{\pi}(s,a) = \alpha(\log \pi(s,a) + H^{\pi}(s)) + V(s)$

```python
def calculate_q_value(value, policy, action, args):
    entropy = -torch.sum(policy * torch.log(policy + 1e-8), 1).reshape(args.batch_size, 1)
    pi = policy.gather(1, action.long())
    return args.alpha * (torch.log(pi + 1e-8) + entropy) + value
```

- We obtain the first and second input parameters(value,policy) by the outputs of the neural network, then we simply calculate the policy's entropy and try to build the formula.
- This is exactly the Q-value that we use to update the network.

Implementation Detail(4): PGQL Update

- This part corresponds to how we exactly update the network parameters.
- Here we will use **replay buffer** to help us train the network.

```python
def q_update(shared_model, shared_optimizer, model, args, memory):
    state, action, reward, next_state, done, hx = memory.sample(args.batch_size)

    value, logit, nhx = model((state, hx))
    policy = torch.exp(F.log_softmax(logit, dim=-1))
    q_value = calculate_q_value(value, policy, action, args)

    with torch.no_grad():
        value_next, logit_next, _ = model((next_state, nhx))
        policy_next = torch.exp(F.log_softmax(logit_next, dim=-1))
        action_next = torch.argmax(policy_next, 1).reshape(args.batch_size, 1)
        q_next = calculate_q_value(value_next, policy_next, action_next, args)
        q_target = reward + args.gamma * q_next * (1 - done)

    mse_criterion = nn.MSELoss()
    loss = mse_criterion(q_value, q_target)

    update_shared_model(shared_model, shared_optimizer, model, loss)
```

Experiment Results(1): Paper Part

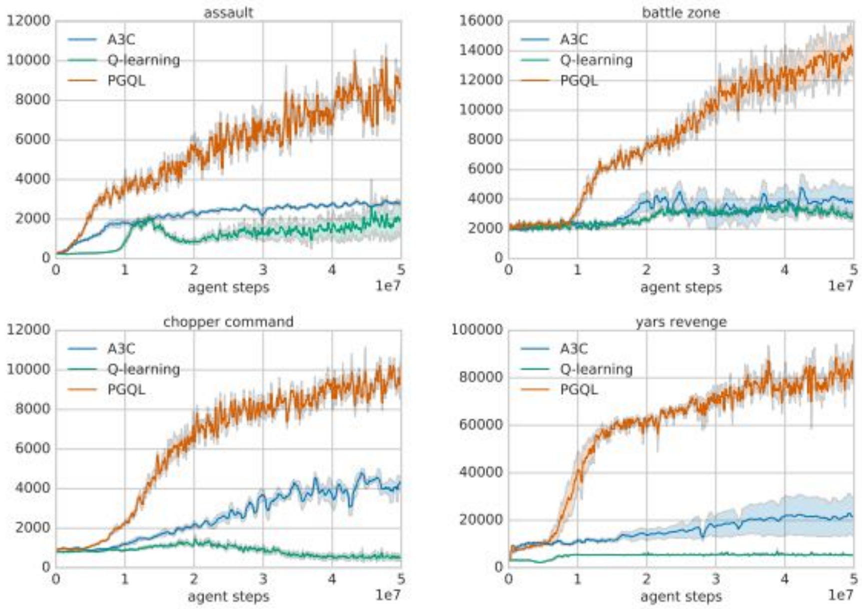
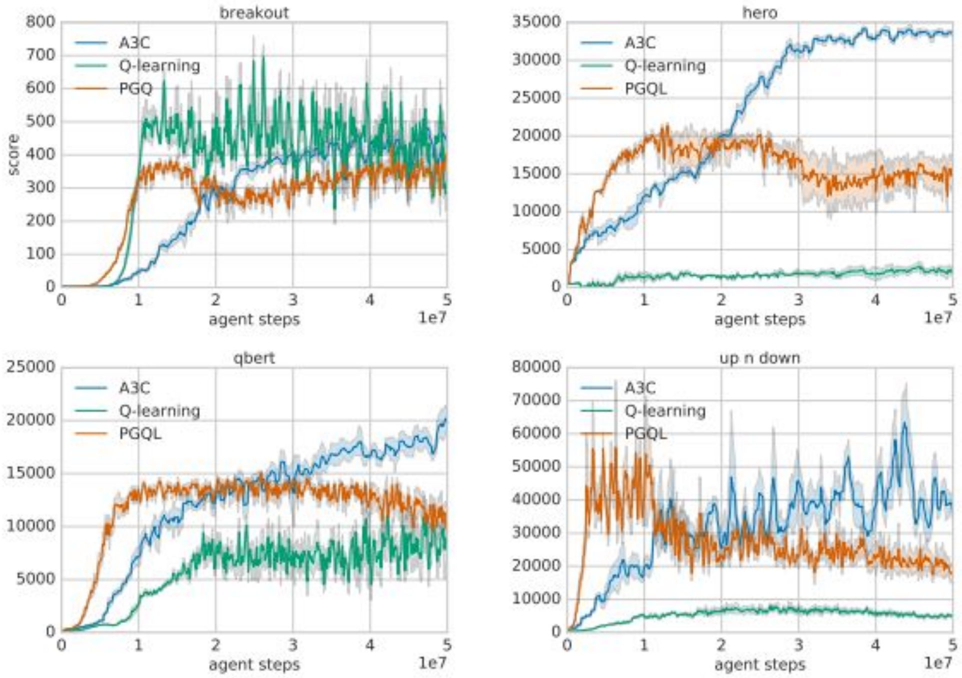


Figure 3: Some Atari runs where PGQL performed well.

- **A3C**: only do actor-critic update.
- **Q-learning**: only update by Q-value
- **PGQL**: do update by both A3C method and Q value.
- **Big Note**: The baseline Q-learning is not the typical DQN or else. **It is the Q-learning with Q as their provided variant.**
- PGQL **outperforms** A3C and Q-learning in these Atari games.

Experiment Results(2): Paper Part



- But in some other Atari games, PGQL didn't have a good performance.

Experiment Results(2): Replication Part

- We only choose two environments **“Assault”** and **“Breakout”** to conduct the experiments, since it is time-consuming to train models.
- Surprisingly, we see that the performance of PGQL on Assault is similar to A3C, and the performance of PGQL on Breakout is much better than the others, **This is different from what we see in the paper results.**