Imitating Optimal Control

Aaron Lou

June 2018

1 Introduction

We already have methodologies for learning the model and then using optimal control to choose the actions. However, oftentimes it's better to derive a policy, as it is quicker and generalizes better.

2 Backpropogation

2.1 Problem with Backpropogation

Recall that we can technically train our policy by backpropogating into the policy. Specifically, the algorithm is given by

```
Result: The optimal policy through Model Based RL
1. run base policy $\pi_0(a_t \mid s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$.
while until convergence do
    2. learn dynamics model $f(s, a)$ through minimizing $\sum_i ||f(s_i, a_i) - s_i'||^2$.
    3. propagate through $f(s, a)$ into the policy to optimize $\pi_\theta(a_t \mid s_t)$.
    4. run $\pi_\theta(a_t \mid s_t)$ and append the visited tuples $(s, a, s')$ to $\mathcal{D}$.
end
```

Algorithm 1: Model-based Rl ver 2.0 / PILCO

But this suffers from the BPTT training problems (vanishing / exploding gradient) and, more specifically, unlike LSTM models, we can't choose simple dynamics. Intuitively, earlier actions have huge gradients.

2.2 Collocation and Constrained Trajectory Optimization

We can use collocation methods and optimize based on u_i. It is given by

$$\min_{u_1, \dots u_T} \sum_{t=1}^{T} c(x_t, u_t) \text{ s.t. } x_t = f(x_{t-1}, u_{t-1})$$

and we can also add in the extra condition $u_t = \pi_\theta(x_t)$ for our optimal policy θ. This is asking us to solve a trajectory optimization problem with constraints.

2.2.1 Dual Gradient Descent with Augmented Lagrangian

Recall that the DGD algorithm is given by

```
while until convergence do
    1. Find $x^\star \leftarrow \arg\min_x \mathcal{L}(x, \lambda)$.
    2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(x^\star, \lambda)$.
    3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$
end
```

Algorithm 2: Dual Gradient Descent (Ascent)

for our Lagracian

$$\mathcal{L}(x, \lambda) = f(x) + \lambda C(x)$$

If we augment our Lagracian to take into consideration the constraint $C(X)$, we set

$$\overline{\mathcal{L}}(x, \lambda) = f(x) + \lambda C(x) + \rho ||C(x)||^2$$

with the new algorithm being

```
while until convergence do
    1. Find $x^\star \leftarrow \arg\min_x \overline{\mathcal{L}}(x, \lambda)$.
    2. Compute $\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(x^\star, \lambda)$.
    3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$
end
```

Algorithm 3: DGD with Augmented Lagracian

and this converges to the correct solution, but and the quadratic term improves stability far from solution. This is closely related to ADMM.

2.2.2 Constrained Trajectory Optimization with DGD algorith

Our problem is given by

$$\min_{\tau, \theta} c(\tau) \text{ s.t. } u_t = \pi_\theta(x_t)$$

$$\mathcal{L}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t (\pi_\theta(x_t) - u_t)$$

$$\overline{\mathcal{L}}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t (\pi_\theta(x_t) - u_t) + \sum_{t=1}^{T} \rho_t (\pi_\theta(x_t) - u_t)^2$$

and the algorithm is given by

```
while until convergence do
    1. Find $\tau \leftarrow \arg\min_\tau \overline{\mathcal{L}}(\tau, \theta, \lambda)$. (Optimization method like iLQR)
    2. Find $\theta \leftarrow \arg\min_\theta \overline{\mathcal{L}}(\tau, \theta, \lambda)$. (Supervised learning ie SGD)
    3. $\lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda}$
end
```

Algorithm 4: Constrained Trajectory Optimization with DGD

2.3 Guided Policy Search

We notice that the above algorithm is very similar to imitation learning. This is because the second step is supervised to learning to the teacher τ. Notice that the optimal control "teacher" adapts to the learner.

```
while until convergence do
    1. Optimize $p(\tau)$ with respect to some surrogate $\tilde{c}(x_t, u_t)$.
    2. Optimize $\theta$ with respect to some supervised objective.
    3. Increment/modify dual variable $\lambda$.
end
```

Algorithm 5: General Guided Policy Search scheme

and we need to choose the form of $p(\tau)$, optimization method for 1, the surrogate $\tilde{c}$, and the supervised objective for $\pi_\theta(u_t \mid x_t)$.

2.3.1 Deterministic Case

If our algorithm is as above

$$\overline{\mathcal{L}}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t(\pi_\theta(x_t) - u_t) + \sum_{t=1}^{T} \rho_t(\pi_\theta(x_t) - u_t)^2$$

Then we optimize τ with respect to $\overline{L}(\tau, \theta, \lambda) = \tilde{c}(\tau)$, θ with the square term $\sum_{t=1}^{T} \rho_t(\pi_\theta(x_t) - u_t)^2$.

2.3.2 Learning with Multiple Trajectories

Often times we wish to learn on multiple trajectories, as this allows our policy to generalize better. Our updated function is given by

$$\min_{\tau_1, \ldots, \tau_n, \theta} \sum_{i=1}^{N} c(\tau_i) \text{ s.t. } u_{t,i} = \pi_\theta(x_{t,i})$$

as we have multiple trajectories and have them all be given by one policy. The updated algorithm for this case is

```
while until convergence do
    1. Optimize each $\tau_i$ in parallel with respect to $\tilde{c}(\tau_i)$.
    2. Optimize $\theta$ with respect to a supervised objective.
    3. Increment/modify $\lambda$.
end
```

Algorithm 6: GPS with multiple trajectories

2.3.3 Stochastic GPS

Our problem (in the stochastic case) is given by

$$\min_{p,\theta} E_{\tau\sim p(\tau)}[c(\tau)] \text{ s.t. } p(u_t \mid x_t) = \pi_\theta(u_t \mid x_t)$$
$$p(u_t \mid x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t)$$

and our algorithm is given by

```
while until convergence do
    1. Optimize $p(\tau)$ with respect to $\tilde{c}(x_t, u_t)$.
    2. Optimize $\theta$ with respect to supervised objective.
    3. Increment/modify $\lambda$
end
```

Algorithm 7: Stochastic GPS

In particular we have already encountered this when fitting local linear models with KL divergence as

$$\min_p \sum_{t=1}^{T} E_{p(x_t,u_t)}[\tilde{c}(x_t, u_t)] \text{ s.t. } D_{KL}(p(\tau) \mid\mid \bar{p}(\tau)) \le \epsilon$$

In particular, we run p to get trajectories and use them to train out network π_θ, which will help us improve p.

It can be shown that, at training time, we pass in full states x_t, but for testing time we can just pass in observations o_t. This is because we can train our neural network on observations, but our p uses x_t.

3 Imitating optimal control with DAgger

We note that DAgger has two problems. Firstly, it's difficult to have humans label the data. More importantly, running our policy can cause damage (ie cost is really high, damaging robotics).

We can fix this problem using the following algorithm (PLATO):
And this gives us a model predictive control combination with DAgger since we are replannign at each time step (since KL divergence changes each step).

```
while until convergence do
    1. train $\pi_\theta(u_t \mid o_t)$ from human data $\mathcal{D} = \{u_1, o_1, \dots u_n, o_n\}$.
    2. run $\hat{\pi}(u_t \mid o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, \dots o_M\}$.
    3. Ask computer to label $D_\pi$ with actions $u_t$.
    4. Aggregate $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$.
end
```

$$\hat{\pi}(u_t \mid x_t) = \mathcal{N}(K_t x_t + k_t, \Sigma_{u_t})$$

$$\hat{\pi}(u_t \mid x_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} E_{\hat{\pi}}[c(x_{t'}, u_{t'})] + \lambda D_{KL}(\hat{\pi}(u_t \mid x_t) \mid\mid \pi_\theta(u_t \mid o_t))$$

Algorithm 8: PLATO algorithm

3.1 RL reasoning

3.1.1 DAgger vs GPS

In general DAgger doesn't require an adaptive expert. Therefore, any expert will do and we need to assume that it is possible to match the expert (not always true).

GPS adapts the expert's behavior, which means we don't require a bounded loss on the initial expert (since it will change).

3.1.2 Why Imitate Optimal Control

It is relatively stable (both supervised learning and Optimal control are). Furthermore, through the input remapping trick, we can train our policy on the raw inputs in stead of the whole state. Lastly, it is efficient and gives us a policy.

3.1.3 Limitations of Model-Based RL

However, Model-Based RL suffers from the fact that we obviously need a model. Oftentimes this is not available or harder to learn than just the policy.

Another common problem is that our expressive approximators are slow. Furthermore, we often assume too much (ie local linearity and smoothness) when we use Model-Based RL.

3.1.4 Overview of our learned methods

Our methods actually divide nicely into different orders of magnitude for training time.

1. Gradient Free Methods (NES, CMA) are the slowest.

2. Fully Online Algorithms (A3C actor critic)

3. Policy Gradient (TRPO)
4. Replay Buffer Value Estimation (Q-learning, DDPG, NAF).
5. Model Based RL (GPS)
6. Model Based "shallow" RL (PILCO)

and the times range from half a month to 20 minutes. In general, we should consider the following breakdown of when to use which algorithm

- Model-Based for fast learning without a simulator.
- Q-learning for slow learning without a simulator or for small simulations (not expensive).
- Policy Gradient/ Actor Critic if our simulation is computationally expensive.