CS 285, Lecture Notes

Deep Reinforcement Learning

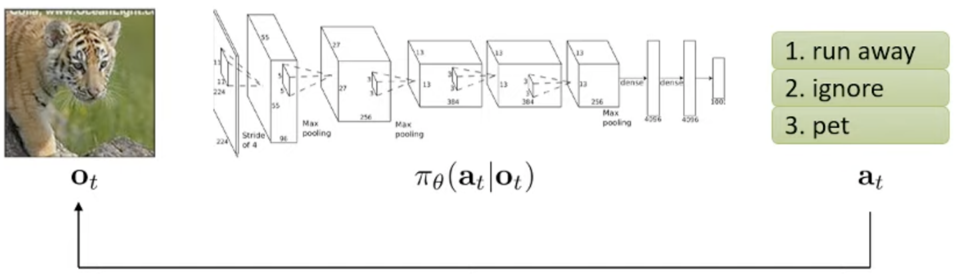
Vishal Raman

Contents

§1 Supervised Learning of Behaviors

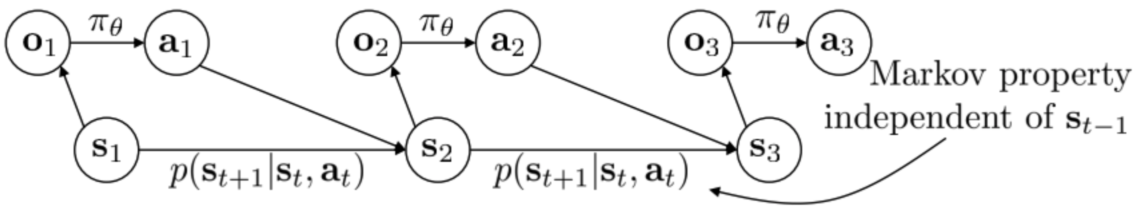
§1.1 Terminology and Notation

In a normal supervised learning problem, say object recognition, we may have a object o_t that goes through a DNN and the target is a label a_t from a policy $\pi_\theta(a_t|o_t)$, where θ represents the parameter of the policy. The subscripts correspond to a timestep, since we usually assume we are given data in a sequential format. Unlike usual supervised learning, the action a_t influences o_{t+1}. This is summarized in the diagram below:



Sometimes, we see the policy written as $\pi_\theta(a_t|s_t)$, where the states s_t are usually assumed to be Markovian while the observations o_t is an observation that results from s_t.

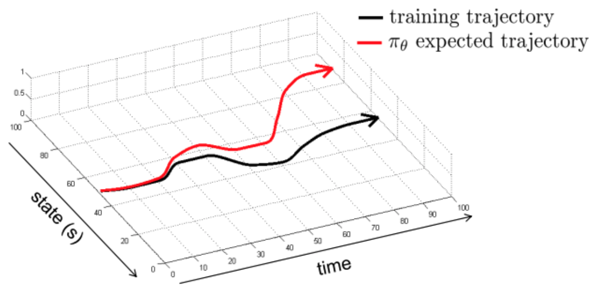
This is illustrated in the graphical model below:



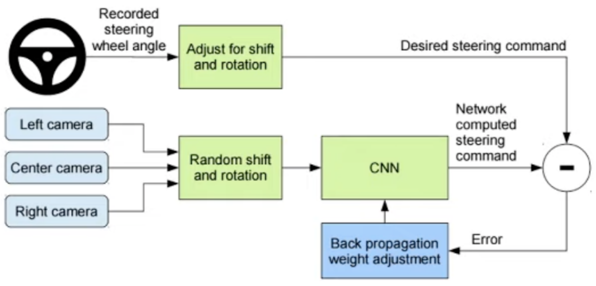
§1.2 Imitation Learning

We can take an example of self-driving cars. We start with observations which are images from the car camera and the actions are how we steer the car. First, we get an image from an expert driving a car and their action. Then, we record the training data and use supervised learning to map from observations to actions. This is a type of **imitation learning** and is sometimes referred to as behavioral cloning, since we are cloning the behavior of the expert driver. The original deep imitation learning system was proposed in 1989, called ALVINN: Autonomous Land Vehicle In a Neural Network.

We do not expect this to work in theory because if we deviate from the training trajectories(which is essentially inevitable), the expected trajectory will make larger and larger mistakes since they are different from all the training trajectories.



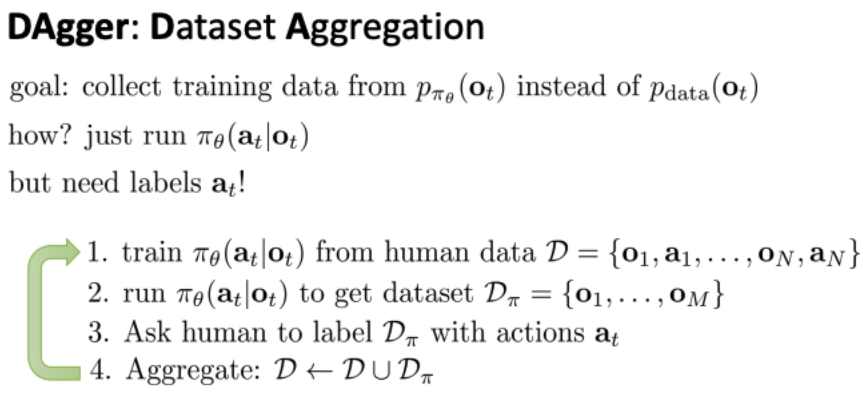
In practice, however, it seems to work well. Bojarski et al. '16, NVIDIA presented the following algorithm:


Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error

Essentially, the left and right cameras enabled the vehicle to learn how to correct small errors. This is a special case of a more general principle, where we learn to train a stable optimal feedback controller, and use this controller to learn a stable policy.

§1.3 DAgger: Dataset Aggregation

Another question we could ask is what is the mathematical principle behind the drift? We are simulating from $\pi_\theta(a_t|o_t)$, where we sample from $p_{data}(o_t)$, a distribution of training data. But, when we run the policy, the observation over the policy is different - $p_{\pi_\theta}(o_t) \neq p_{data}(o_t)$. This is a classic distribution shift problem.



DAgger: Dataset Aggregation

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

The issue with DAgger is that asking a human to label $\mathcal{D}_\pi$ with actions a_t is not easy, because humans use feedback control - watch the effect of the actions and see the observations.

§1.4 Deep Imitation In Practice

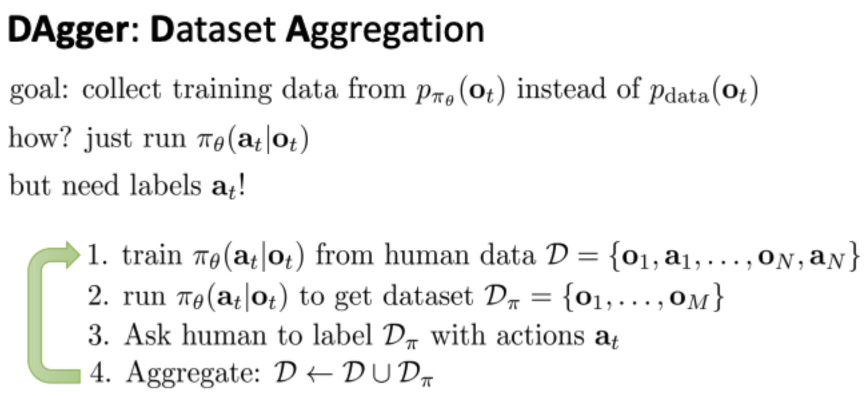
Can we make it work without more data? DAgger addresses distributional drift, but what if our model is so good that it doesn't drift? To do this, we need to mimic expert behavior accurately without overfitting.

Why might we fail to fit the expert?

- Even if the observation is fully Markovian, the human behavior might not be Markovian, i. e. depend on additional previous observations.
- If we have continuous actions, the demonstrator behavior might inconsistently select from multiple modes in the distribution

§1.4.1 Non-Markovian Behavior

To address the Markovian problem, we can use a neural network to encode the images into an RNN state, and use the sequential RNN states to output an action. We could also use an LSTM cell, which performs better in practice. However, we have the issue of **causal confusion**.



DAgger: Dataset Aggregation

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

but need labels $\mathbf{a}_t$!

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Adding a breaklight in the above example makes it difficult to determine whether the breaking is determined by the light or by the pedestrian in the board. Does DAgger mitigate causal confusion?

§1.4.2 Multimodal Behavior

When we have multimodal distributions, it is difficult to use the mean and the variance to understand the high probability possibilities: for example, if we want to move left and right with high probability but to the middle with low probability, the mean is the middle option, which should be low probability.

We have a couple of methods for addressing this:

- Output a mixture of Gaussians. These are sometimes called Mixture Density Networks. The idea is we output $w_1, \mu_1, \Sigma_1, \ldots, w_N, \mu_N, \sigma_N$ and output $\pi(a|o) = \sum_i w_i \mathcal{N}(\mu_i, \Sigma_i)$. Some problems with this is that we need more output parameters, and the number of mixture elements we need grows exponentially with the number of dimensions.
- A more sophisticated option is a Latent variable model. In this, we output a Gaussian, but we also input a latent variable $\xi \sim \mathcal{N}(0, I)$. We can show in theory that such a model can represent arbitrary distributions. Some examples are the

conditional variational autoencoder, normalizing flows, or Stein variational gradient descent.

- If we have discrete options, multimodal distributions are not an issue since the number of bins we need for discretizing a continuous action space grows exponentially with the dimension. Autoregressive discretization discretizes one dimension at a time.
  1. First, we discretize the first dimension. Then, we sample from the softmax and obtain a value of the first dimension.
  2. Then, we feed the value into another neural network that outputs a distribution over the second dimension.
  3. Repeat the process over all the dimensions.

§1.5 Cost/Reward Functions

In imitation learning, the humans need to provide data, which is typically finite. Deep learning methods work well when there is a large amount of data. Furthermore, humans are not good at providing some kinds of actions.

We will try to get our machines to learn autonomously, like humans do. Namely, we try to minimize the objective

$$\min_\theta E_{s_{1:T},a_{1:T}}\left[\sum_t c(s_t, a_t)\right]$$

where $c(s_t, a_t)$ is a cost function. We could also replace this with the maximization of a reward $r(s_t, a_t)$.