Advanced Model Learning

Aaron Lou

July 2018

1 Recap

Previously it has been shown that DQN works with images and Convolutional Neural Networks. However, we would like to see if this also applies to Model Based RL. As a quick recap, model based RL is given by

```
**Result:** A model and a way to pick through this model
1. Run base policy $\pi_0(u_t \mid x_t)$ (ex: random policy) to collect
   $\mathcal{D} = \{(x, u, x')\}$.
while until convergence do
    2. learn dynamics model $f(x, u)$ to minimize $\sum_i ||f(x_i, u_i) - x'_i||^2$.
    3. backpropogate through $f(x, u)$ to choose actions (iLQR).
    4. execute those actions and add to $\mathcal{D}$.
end
```

Algorithm 1: Model Based RL algorithm 1

If we have a POMDP (partially observed MDP) then we wish to find x, the state, given an observation o. Normally, o has high dimension, but x has lower dimension.

2 Models in Latent Space

The key idea is to learn an embedding $g(o_t)$ to our latent space x_t, and have this be our state for model based rl.

The basic idea is to collect data with exploratory policy. Then, learn low dimensional embedding of image. Finally, run q-learning with function approximation embedding.

For example, we can use an autoencoder and use the bottleneck as our feature space. We can use regular, sparse, and variational autoencoder. Note that the variational ae tries to minimize the reconstruction cost with the KL divergence between $q(z \mid 0)$ and some normal distribution of z. Normally, the loss is just

$$\sum_i \frac{1}{2}||o^{(i)} - \hat{o}^{(i)}||^2$$

A model based learning is given below.

Collect data with exploratory policy
Learn smooth, structured embedding of image
Learn local-linear model with embedding
Run iLQG to learn to reach images of goals and goal gripper pose.

In the given case, we take an activation layer (final conv layer) which we denote a_{cij}, where c is the channel and ij are coordinates. Taking a spatial softmax over ij we have a distribution and just sum up these values to get the expected feature

$$f_c = (\sum_i i s_{cij}, \sum_j j s_{cij})$$

To get the reward, we provide the agent with a r̈eward imageänd measure the distance between the embedding between this and the actual image. The agent learns the autoencoder as well as the correct path.

The last method is as follows

1. Collect data
2. Learn embedding of image and dynamics model (jointly)
3. run iLQG to reach image of goal

In particular, we learn the embedding $g : o_t \rightarrow z$ and the decoder $d : z \rightarrow o_t$. Learn a mapping $f : z_t \rightarrow \hat{z}_{t+1}$ or the expected next latent space. We wish to make sure that $\hat{z}_{t+1}$ to match $g(o_{t+1})$.

Note that we can't learn the embedding and the models of the embedding space since we might eventually just learn a constant latent space.

3 Models directly in Image Space

In this case, we need to learn a video-predicative model conditioned on our action. There are two models.

The first was a feed-forward convolution enconder with an action applied and attempts to reconstruct the next state.

The second was a recurrent encoder. There is an lstm module that recalls previous frames before reconstructing the next state. This allows for multistep prediction based on initial observation and a sequence of actions.

The model structure is an encoder CNN, the next step prediction model with an action, and the decoder Dilated CNN. We also learn curriculum learning/schedule sampling.

We can also use this for an informed explorer (instead of a random explorer). The algorithm is

1. Store most recent d frames.
2. For every valid action, predict 1 fram into the future.
3. Take action corresponding to frame least like past d frames. Similarity is given by Gaussian Kernel Similarity

$$n_D(x^{(n)}) = \sum_{i=1}^{d} k(x^{(a)}, x^{(i)}) \quad k(x,y) = \exp(-\sum_j \min(\max((x_j - y_j)^2 - \delta, 0), 1/\sigma)$$

Note that this normally works for DQN/pretrained replay buffer.

This is stable and useful for control, but it is restricted to synthetic images and is hard to plan on.

However, this doesn't really work for real, more complex images. Instead, we use something called **action-conditioned** multiframe video prediction using **flow prediction**. This predicts how the objects will move between frames.

The model given is a Stacked ConvLSTM to predict convolution kernels, apply them to the image, and predict which parts of the image correspond to that image. We can feedback images to get multistep prediction, and we train it with l2 loss.

We plan using MPC.

1. Sample N potential action futures
2. Predict the future for each action sequence
3. Pick best situation and repeat.

Specify goal by selecting where the pixels should move. Select future with maximal probability of pixels reaching their desired goals.
This methodology is useful on real images with limited human involvement, and is more efficient that single task model-free learning. However, it is still limited and can't handle complex skills like model free learning. Furthermore, it is very computing heavy.

4 Inverse Models

We can try to circumvent the reconstruction of images by learning an inverse model $f(o_t, o_{t+1}) = u_t$ ie find the action to get a new state.

One possible way is a siamese CNN. Another is a greedy planner that feed in a goal and current image and predict the action to get from current to goal.

This doesn't require human involvement, and it doesn't have to reconstruct the image (which is hard). However, you can't plan into the future, and the objective only cares about the action.

5 Predict Alternate Quantities

Instead of predicting the next action, this asks to predict another quantity. For example, it predicts collisions or game character health.

This only needs to predict task-relevant values, but we must be able to actually observe them.

6 Overall Summary

Learning the right features is important. Furthermore, learning reward/objective is important when using models of observations. Some pros vs cons are given below

Models:

+ Easy to collect data
+ Possible transfer between tasks
+ Typically requires a smaller quantity of supervised data.
- Models don't optimize
- Sometimes harder to learn than policy.
- Often needs assumptions to learn complex skills (continuity/reset)

Model-Free:

+ Makes little assumptions besides beyond reward function.
+ Effective for learning complex policies.
- Requires a lot of experience (slower)
- Not transferable between tasks.