Teaching a robot to perform a basketball shot using EM-based reinforcement learning methods

Tobias Michels
TU Darmstadt

Aaron Hochländer
TU Darmstadt

Abstract

In this paper we experiment with reinforcement learning methods using a basketball throwing task for a simulated robot as simulation environment. We show the results of the implementation of expectation maximization algorithms to learn the parameters of dynamic movement primitives that generate a trajectory for the arm of the robot. The first EM algorithm we used led to good, but not perfect results. We tried to improve it by inserting prior knowledge to the method, but for our problem it did not work out.

1 Introduction

Reinforcement learning algorithms are a very interesting research topic. They allow us to create virtual behaviors that are able to adapt to new circumstances autonomously. This is a huge improvement over traditional approaches in the field of artificial intelligence, where agents usually simply follow a set of predefined static rules to solve a given problem. Therefore reinforcement learning will be crucial if someday researchers want to be able to achieve human-level artificial intelligence. A common application domain for these methods is robotics. However, einforcement learning algorithms usually require a lot of trials until they converge. This consumes a lot of time when applied on actual robots, thus tasks are often simulated before experimenting with the robot itself.

A central problem for a lot of robotics applications is learning a certain movement, for example an arm motion. This is required in most robotics tasks, so our goal was to implement a learning method for one task that can be generalized for various utilizations.

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume 22 of JMLR: W&CP 22.

The task we chose is making a simulated robot arm learn to throw a ball into a cup, similar to a basketball shot. We used weighted dynamic movement primitives (DMP) to compute a trajectory for the arm and expectation maximization (EM) algorithms with and without prior knowledge to learn good weights for the DMPs.

DMPs are a nice way for generating and adjusting trajectories since they always depend on a fixed number of parameters that is chosen by the developer. This makes learning fairly easy in comparison to other methods which have to take several actions into account in every state of the robot. We chose EM methods to learn the weights of the DMPs because they do not require a lot of parameter tuning in order to produce good results. Still adding prior knowledge to the EM approach can improve the behavior of the robot even further by making sure that the algorithm does not converge to a bad local maximum.

1.1 Related Work

Our work is part of a regular practical project at TU Darmstadt where students can come up with diverse tasks for a simulated robot and learning methods that they want to implement. We based our basketball task on a beerpong task which was previously introduced to the project. The throwing motion in the drinking game beerpong bears a resemblance to a simplified basketball shot, thus it can be solved in a similar way (Wagner and Schmitt 2013).

Kober and Peters (2008) furthermore showed that when using DMPs in combination with policy search algorithms robots can fulfill very complex tasks like the cup-and-ball game. In that game a ball hangs on a string from a cup and the player holds the cup in his hands while he performs movements to get the ball up in the air and inside the cup.

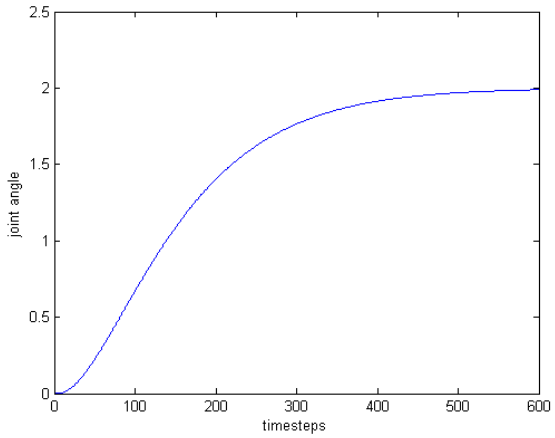

Figure 1: trajectory generated by DMPs with goal g = 2 and the forcing function being zero

1.2 Outline

In the next chapter we will give a short overview about the backgrounds of the methods we used. Afterwards our simulation setup and the results of our experiments are presented and ultimately we summarize our work and give suggestions for possible follow-up projects.

2 Approach

In this section we are going to present the methods that we used in order to solve the given robot learning problem.

2.1 Dynamic Movement Primitives

First of all we need a simple way to represent a trajectory for a joint of the robot arm, so that we are able to perform learning algorithms with a reasonable amount of parameters. Our choice fell on the approach called Dynamic Movement Primitives (Schaal et al. 2003). With DMPs it is possible to formulate a point attractive system which will always approach a given goal state. The dynamics for this system are based on a spring-damper system which is given by

$$\ddot{y} = (\alpha_y(\beta_y(g-y) - (\dot{y}/\tau)) + af(x))\tau^2,$$

where g is the goal state, y is the state, $\dot{y}$ is the velocity, $\ddot{y}$ is the acceleration, a is an amplitude, τ is a time constant, α_y as well as β_y are constants that define how critically damped the system is. The forcing function f is defined by

$$f(x) = \frac{\sum_{i=1}^{n} \psi_i(x) w_i x}{\sum_{i=1}^{n} \psi_i(x)},$$

where w_i are weights and

$$\psi_i(x) = \exp(-h_i(\frac{x}{g} - c_i)^2)$$

are radial basis functions with c_i and h_i being their centers and widths. x is the canonical state of the system with the initial state set to 1, and the canonical system is given as

$$\dot{x} = -\alpha_x x \tau,$$

where α_x is another constant.

A typical trajectory that is generated by DMPs with all the weights being zero is shown in figure 1. The addition of forcing function, whose centers are placed at different points of the trajectory, now give us the possibility to adjust the movement of the robot arm by learning good weights.

2.2 Expectation Maximization

The method which we used to learn the weights of the DMPs is a specialized EM algorithm which uses Gaussian distributions to explore the parameter space, similar to the PoWER algorithm as described by Kober and Peters (2008). First of all we introduce a Gaussian distribution $p(w|\theta)$ for every weight of the DMPs which represents the probability to draw a value w for the weight given a mean μ and a variance σ^2. These sets of values $\theta = [\mu, \sigma^2]$ are the parameters of our algorithm that we want to optimize. The means of the Gaussians are initially set to the original values of the weights and the initial variances express the exploration range when searching for better values for the weights. Subsequently we iteratively perform expectation and maximization steps to update the parameters.

In the expectation step we draw n samples w_i from the Gaussian distributions of the weights and generate new trajectories using DMPs with the sampled weights. Then the new trajectories are applied to the simulated robot and we collect the rewards r_i of these actions.

Thereupon new means and variances are computed using a weighted maximum likelihood method:

$$\mu = \frac{\sum_{i=1}^{n} p(R|w_i) w_i}{\sum_{i=1}^{n} p(R|w_i)},$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} p(R|w_i)(w_i - \mu)^2}{\sum_{i=1}^{n} p(R|w_i)},$$

where $p(R|w_i) = \exp(\beta r_i)$ are the exponentially shaped reward weightings with a weighting factor β.

The outcome of this approach is obviously probabilistic. It depends on the initial choices of the means and

variances as well as on the amount of samples that are drawn every iteration and the weighting factor β. Because of that it is not guaranteed to find an optimal solution and instead some experimenting with all of these parameters is still necessary in order to achieve good results. Even then best solution can not always be found because convergence to local maxima is hard to deal with using this approach.

On the other hand the problem of finding a good step size, which is common in gradient descent approaches for example, does not occur since the variances which represent the exploration range are always updated according to the latest results.

2.3 Expectation Maximization with a prior

Formally the previously described maximum likelihood approach looks like this:

$$\max_{\theta} \int p(R|w)p(w|\theta)dw,$$

We hope that using prior information about our parameters prevents us from converging to bad local optima during our EM algorithm. In the resulting approach we leave the expectation step the way it was before, but we add prior knowledge $p(\theta)$ into our computation in the maximization step and thus calculate the maximum a posteriori probability

$$\max_{\theta} \int p(R|w)p(w|\theta)p(\theta)dw.$$

The problem that has to be solved boils down to

$$\max_{\theta} \sum_{i=1}^{n} p(R|w_i) \log(p(w_i|\theta)) + \log p(\theta),$$

with $p(\theta) = \mathcal{N}(\mu|\mu_0, (\beta\lambda^{-1}))Gam(\lambda|a,b)$ and $p(w|\theta) = \mathcal{N}(w|\mu, \lambda^{-1})$. Gam is a gamma distribution over the inverse variance λ and a, b, β, λ, μ_0 are parameters.

Setting the partial derivatives of this term to zero leads to the new optimal solutions for μ and σ^2:

$$\mu = \frac{\mu_0\beta + \sum_{i=1}^{n} p(R|w_i)w_i}{\beta + \sum_{i=1}^{n} p(R|w_i)},$$

$$\sigma^2 = \frac{(\mu - \mu_0)^2\beta + 2b + \sum_{i=1}^{n} p(R|w_i)(w_i - \mu)^2}{2a - 1 + \sum_{i=1}^{n} p(w|u_i)}.$$

This approach gives us several new parameters to choose. The value for μ_0 is the initial guess of μ and should be the same as the initial value of μ if we do not have any prior knowledge about the means of the Gaussians. The constant β expresses the certainty about μ_0 and a as well as b are used to regulate how fast the variance converges. Thus in theory if the maximum likelihood method converges to a local maximum instead of the global optimum, using the a posteriori approach with high values for a and b should prevent this from happening. Still it is a challenging task to choose all the parameters correctly.

3 Experiment

In this section we will describe our setup and the influences of different values of the parameters on the outcome of the experiment.

3.1 Setup

For our experiment we used the SL-simulator. The machine learning algorithms were executed in Matlab and sent to SL using shared memory communication. The robot arm is hanging from the ceiling and possesses seven degrees of freedom but currently we are using only two of them (the wrist and the elbow). The ball is attached on the end effector and released when the arm passes a specific point. The basket is placed slightly higher than the arm and in front of a board so the ball can bounce of the board into the basket if the board is hit correctly.

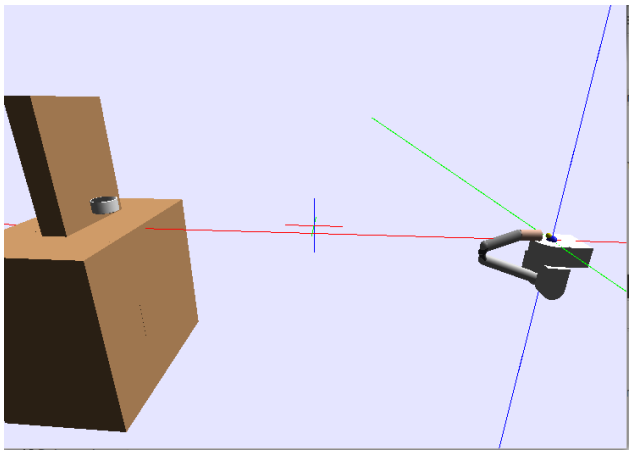

Figure 2: the initial position of the robot

3.2 Reward function

Basically the reward function consists of the Euclidean distance between the basket and the ball and some penalties for undesirable actions. One of the main

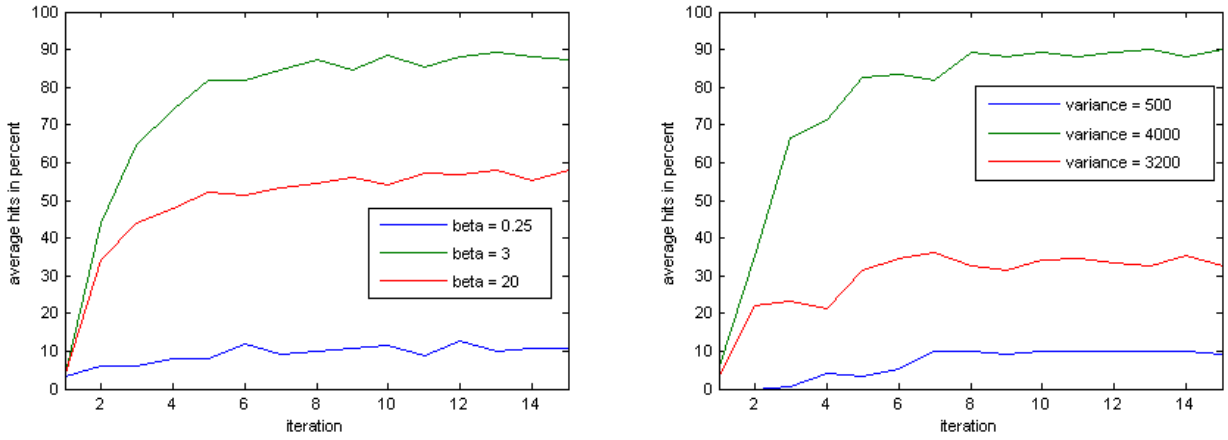

Figure 3: The results for different parameters of the EM algorithm.

tasks of the reward function is to ensure that the ball is thrown high enough, so that there is even the possibility of throwing into the basket. To achieve this a rectangle is spanned on a level with the opening of the basket. If the rectangle is passed during a shot the Euclidean distance between the point where the ball passes the rectangle the second time and the basket is computed. When the rectangle is not passed during a throw a high penalty (-200) is given.

Furthermore there is a small punishment (-2) if the ball hits the outer rim of the cup so the robot learns to throw through the top of the cup and does not try to throw through the rim of the cup. Another purpose of this small punishment is to make local optima very close to our global optimum and less attractive for the EM algorithm.

3.3 Results for EM

The results for the experiment with EM are greatly influenced by the β parameter of EM. β influences the speed of convergence of the variance. For $\beta = 0.25$ the variance does not converge, thus the throws are just randomly thrown by taking the samples without any learning at all. With $\beta = 3$ only in the first iteration the samples are taken completely random. After the first iteration (in most cases after the ball hits the cup the first time) there is already a great improvement visible. In the following iterations the probability of hitting the cup increases to approximately 80% and stays on that level. For $\beta = 20$ the variance converges too fast so in slightly more than half of the attempts the EM finds only a local maximum and not the global one.

Since the variance determines the radius from where the samples are taken it is important to take a good initial variance. For $\sigma^2 = 500$ the area from where the samples are taken is too small and converges too fast so that the basket is barely hit. With $\sigma^2 = 4000$ the exploration range is big enough to find samples that hit the cup and the variance does not converge too fast so that the probability to find only a local maximum is rather low. When taking $\sigma^2 = 32000$ as initial variance the exploration range is big enough but it does not converge in an appropriate time so that it will take a long time until the basket is hit constantly.

3.4 Results for EM with prior

We tried to implement the EM with prior as well but unfortunately it did not work out for our experiment. We had to choose $0 < a < 1$ to make the variance converge. But since our reward r is negative, the exponentially shaped reward $p(R|u_i) = \exp(\beta r_i)$ becomes in most cases very close to 0 so that σ^2 becomes negative because the denominator of σ^2: $2a - 1 + \sum_{i=1}^{n} p(R|u_i)$ becomes negative. Thus $\sqrt{\sigma^2}$ in $\mathcal{N} \in C$.

4 Conclusion

Finally we want so sum up our work and give an impulse for future work in this area.

4.1 Summary

We showed that the task of throwing a ball in a cup similar to the motion of a basketball shot can be solved relatively well using DMPs to generate a trajectory for the elbow joint of a robot arm and EM methods to learn good weights for these DMPs. The first EM method did not produce perfect results but with good parameters a hit ratio of at least 80% could be achieved already after about five iterations. Tweaking the re-

ward function might improve this even further.

Instead we added a prior to the EM algorithm to prevent the algorithm from converging without getting to a hit ratio of values closer to 100%. Unfortunately this approach did not work out well because of unexpected problems with the prior.

4.2 Future work

There is a vast field of possible extensions to our task that we could not examine further since we ran short on time in the end. We only give some of the suggestions here that we had in mind.

First of all there is the problem with the EM with prior we would like to fix because we spent a lot of time on it.

During our experiment we also got the impression that it is not sufficient to perform the learning process just for the elbow joint, contrary to earlier expectations. So another possible future work would be to add the shoulder and especially the wrist joint to the learning process.

Finally there is the possibility to replace the Gaussians with only one multi-variant Gaussian in the EM algorithm.

References

F. Wagner and F. Schmitt (2013). Robot Beerpong: Model-Based Learning for Shifting Targets, *Proceedings of Projektpraktikum Robot Learning.*

J. Kober and J. Peters (2008). Policy Search for Motor Primitives in Robotics, *Machine Learning* 84(1-2), 171-203.

S. Schaal, J. Peters, J. Nakanishi and A. Ijspeert (2003). Control, planning, learning, and imitation with dynamic movement primitives, *Proceedings of the Workshop on Bilateral Paradigms on Humans and Humanoids, IEEE 2003 International Conference on Intelligent Robots and Systems (IROS)*, 27-31.

J. Kober and J. Peters (2009). Learning motor primitives for robotics, *Robotics and Automation, 2009. ICRA'09. IEEE International Conference*, 2112-2118.