Universität Hamburg
Department Informatik
Knowledge Technology, WTM

Neuro-inspired reinforcement learning

Seminar Paper

Neural Networks

Caus Danu
Matr.Nr. 7014833
7caus@informatik.uni-hamburg.de

12.07.2018

Abstract

This paper attempts to describe the major ideas and approaches within the field of **Reinforcement Learning** (RL) and present the place this particular type of learning takes in the greater field of artificial intelligence (AI). Furthemore, it tries to emphasise and argue the biologically inspired ideas that reinforcement learning attempts to implement in computation. However, despite the biological aspects that inherently describe RL and are an intrinsic part of it, RL alone is not sufficient to design a general purpose AI and hence, more sophisticated systems are necessary that combine the major fields of supervised, unsupervised and reinforcement learning, as well as control theory.

Contents

1 Introduction

Reinforcement Learning, as the name implies, uses the principle of reinforcement in order to acquire knowledge and plan meaningful actions. Behaviour that repeatedly results in positive rewards and outcomes will be picked up by the artificial agent, since the basic learning strategy is based on positive reinforcement. On the other hand, negative rewards will encourage negative reinforcement and also result in learning by essentially pruning undesired actions. The principle is well studied within the fields of psychology and neuroscience and has relatively high success in computational models as well. In fact, psychologists have had a renewed interest in the topic based on the success stories from computer science and some of the mathematical proofs and even performance tricks that are used in artificial agents (like for instance the buffer replay trick). The research within RL is very prolific, especially with regards to model free algorithms. Some of the noteworthy results include:

- the modelling of rewards using distributions rather than expectations [1]
- and partially solving the problem of sparse rewards and huge state spaces [2]

On the other hand, the topic of model based RL is not as popular, but is becoming more and more relevant, especially in robotics, since it has the advantage of requiring significantly less training time and data. Some of the exciting outcomes here include:

- Efficient backpropagation through the model [3]
- Inverse Modeling [4]

The field of combined RL is also a hot topic since it tries to combine the best of both worlds: model based and model free. Some of the interesting novelties are:

- Using a model as a baseline [5]
- Creating a model free policy with planning capabilities [6]
- Performing model based look-ahead [7]

Although reinforcement learning is a significant type of intelligence, it is important to understand that there are other types as well. One of the other major ones is the visual intelligence. Some simple problems like finding a shortest path between two points can be easily solved by our visual system just be looking at the map, whereas an RL algorithm would require thousands of iterations, trials and errors to do so. The algorithm would wander to locations that are obviously not relevant, like to the left of the left most point, when in fact the visual system easily realizes that it should always search to the right of the left most point, and so forth. Convolutional Neural Nets (CNNs) and the domain of supervised learning are the key fields that deal with visual intelligence, attention and saliency. Reinforcement Learning on the other hand is more about the back-end logic rather than perception, although it can be viewed as an entire system of its own, capable of perception as well and as a consequence being an end to end learning method.

2 Biological Reinforcement Learning

A very important notion of RL which is biologically inspired is the concept of reward. A reward can be thought of as the teaching signal. If we make a reference to supervised learning, it is in fact the supervisor. If we make a reference to the field of expert systems and knowledge base systems (see paper [8] as an example), the reward is a way to encode the domain knowledge. In RL minor changes in reward can have a huge impact on the way the algorithm learns and the outcomes it elicits. For example, in a game, if all states give negative reward except the end state, the agent is encouraged to finish the game as quickly as possible. If some intermediate states give also pleasurable positive rewards, the agent is encouraged to wander around and hover in the vicinity of those pleasurable states. The word "pleasurable" is intentionally used to bridge the notion of reward and the processes that go on in the brain. It is a proven fact that one of the ways in which humans learn is by using the dopaminergic pathways (check figure 1) in the brain that are associated to pleasure and pleasurable excitations. In particular, this is associated with the model free type of reinforcement learning. The basic idea is that the dopamine neurons in the brain get excited or inhibited as a consequence of being exposed to a positive or negative prediction error. If an animal is trained for instance to associate a certain light stimulus with food being given a few moments later, the dopamine neurons will always start to spike whenever the animal sees the light. That is, the neurons encode a positive prediction error (see figure 2). When the food is indeed administered later on, the neurons do not spike because there is essentially no prediction error: the animal got what it expected. However, if the food does not come, the dopamine neurons get into a quiescent state, a depression state that encodes the negative prediction error: the animal did not get what it expected in the end.

Figure 1: Dopamine Pathways

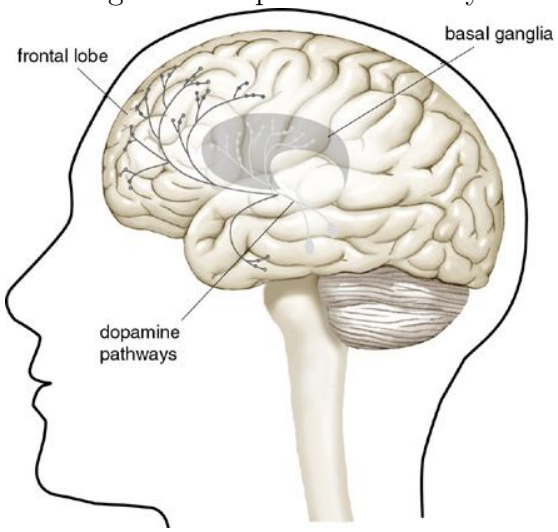
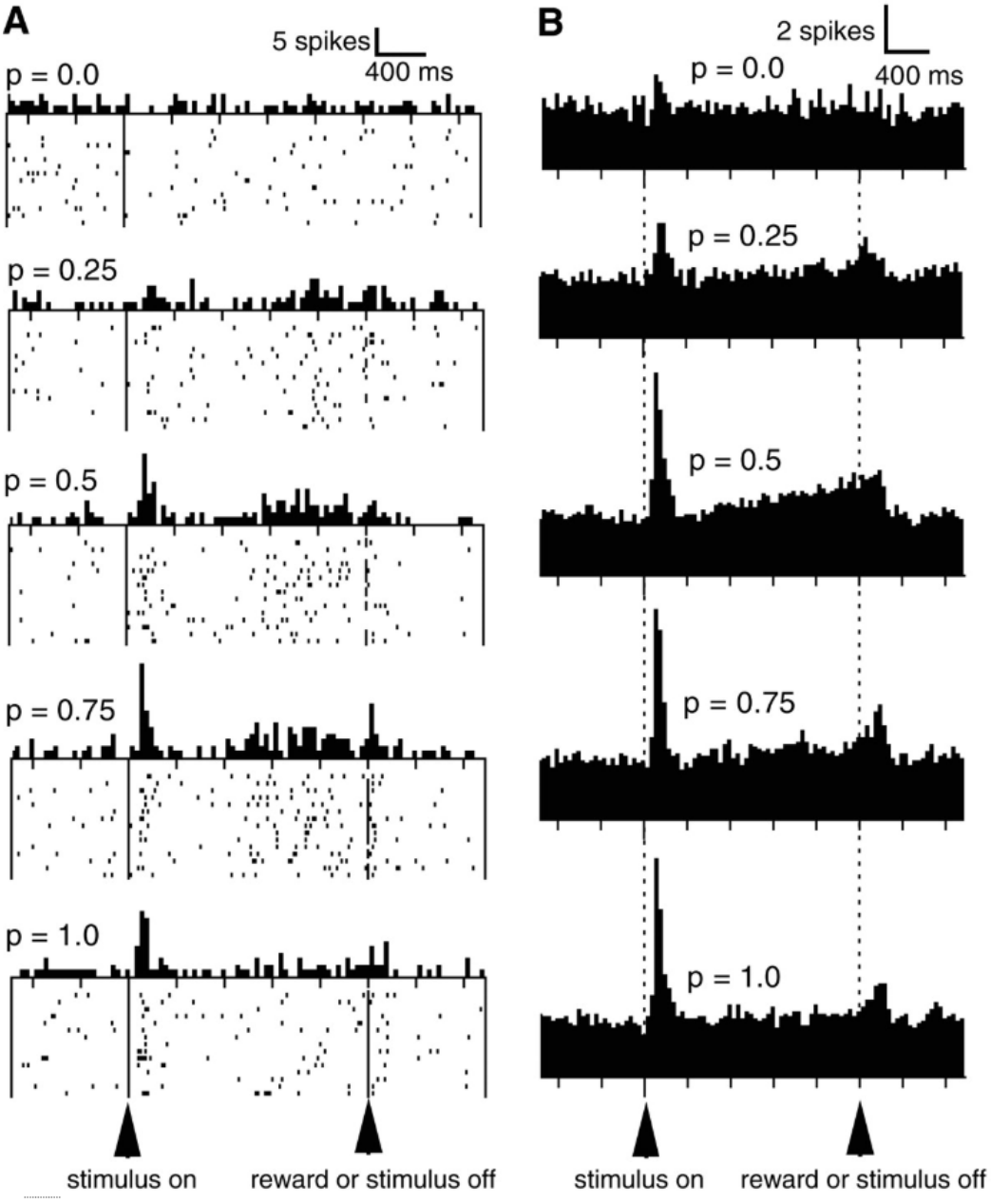


Figure 2: Peri-stimulus time histograms from a primate dopaminergic neuron in a classical conditioning experiment, reproduced from Fiorillo et al. (2003) [9]. *Zero prediction error results in a normal baseline activity of the dopamine neurons. Positive prediction errors result in dopaminergic excitations that have an intensity directly proportional to the absolute value of the prediction error.*

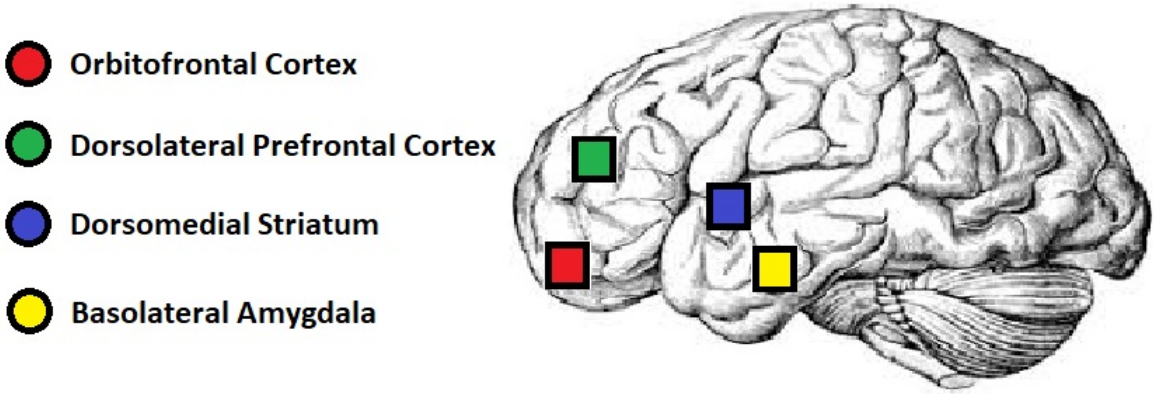


Another type of biologically inspired RL is the Model based/ tree search RL. Here we can list the Tolmanian Forward model as an example and the concepts of forward and backward tree search [10]. The regions of the brain involved in model based control (see figure 3) are typically:

- The Orbitofrontal cortex (OFC)

- The Dorsolateral prefrontal cortex (dlPFC)
- The Dorsomedial Striatum
- The basolateral amygdala (BLA)

Figure 3: Brain Regions involved in Model-Based RL



This type of learning does not necessarily involve the concept of reward, although it can, especially if the animal tries to optimize the way it acts. The important thing is that the animal encodes a certain relevant map or model for the task it tries to perform and the goal is to plan a sequence of actions that will accomplish the task by building a mental representation in the form of a decision tree.

Lastly, a third type of biological RL is the so called Pavlovian Control. If we compare this to the act of learning how to play chess, this would be the list of opening strategies that all players learn in order to set up the game more comfortably. Pavlovian Control is closely related in this sense to the concept of "situational memory". As an example of brain structure that evolved for this type of RL is the "Periaqueductal gray" which encodes evolutionary decision making defensive strategies, for instance: the fight or flight response. A very eloquent example of this is presented in [11] by the Hershberger chicken experiment. The chicken has to acquire food from a feeder that would move away at twice the speed if the chicken ran towards it. However, if the chicken runs away and not towards the feeder, this moves the feeder closer to the chicken at twice the speed. What is interesting is that the chicken will always try to move towards the food even if this action does not bring the desired outcome, which is a behaviour that is definitely not optimal, but is somehow encoded and hardcoded as a Pavlovian response. Therefore, there is an evolutionary circuitry in the chicken's brain that tells it that moving towards the food source is always better.

3 Computational Reinforcement Learning

Disclaimer: *This chapter uses equations taken from [12] for explanation purposes.*

Reinforcement Learning can be viewed as an approach that stands in-between the supervised learning and the unsupervised learning techniques. All of these three fields essentially deal with the problem of prediction and hence solving or filling in the blanks for an expression of the sort:

$$y = f(x)$$

The supervised learning can be thought of as a function approximation problem. We are given the outputs, or the y and the inputs, or the x, and we need to come up with a function f that successfully maps from x to y. In other words, we need to learn f. With more intuitive terms, supervised learning can be described as a top-down information precessing, especially if we consider the topic of visual saliency detection, where the image is decomposed into small meaningful feature maps at each layer of the convolutional network and the entire process of decomposition is supervised by essentially knowing the ground truth outcome and being able to backpropagate and correct errors.

The unsupervised learning problem is essentially a clustering and description problem. The algorithm is simply given an input x and needs to come up with the function f that would somehow make sense of the fed arguments, and cluster them in different categories. Note that there is no correct output y that the algorithm takes into account in order to solve the problem. Unsupervised learning can be seen as a bottom-up approach, with some of the most important algorithms being:

- Principal Component Analysis
- Single Value Decomposition
- K-Means Clustering

Finally, reinforcement learning is a peculiar problem, where the algorithm is given the input x and another helper variable r, which will be referred to as the reward in later sections, and is expected to produce the output y and learn the function f that produces this y.

Reinforcement Learning is one way to do decision making and therefore, it is closely related to the field of Markov Decision Processes (MDP).

In order to understand what a Markov Decision Process is, the best way is to decompose it into the four main components:

- States: s
- Model (or the so called Transition Model) $T(s, a, s\prime) \sim Pr(s\prime|s, a)$
- Actions $A(s)$

- Rewards $R(s), R(s,a), R(s,a,s\prime)$

One of the most important Markov properties can be stated as: only the present counts and nothing else matters. In other words, we need not care about the past states in order to reason about what the next state might be. Only the current state matters for this purpose. This might seem like an unrealistic and possibly even restrictive assumption at first glance. It is conceivable for instance that we would need to consider N previous states in order to come up with the next one. However, in the Markov world, you can always embed these N states into one state and consider this as your current state. Somewhat of a similar argument can be applied to the fact that we represent rewards as $R(s)$, $R(s,a)$ or $R(s,a,s\prime)$. Strictly from a technical point of view, these expressions are not equivalent, the first one signifying the reward an agent gets in state **s**, the second meaning the reward of getting in a state **s** and performing an action **a**, and the third adding the notion of ending up in a state $s\prime$ after performing an action **a** in state **s**. However, we can also think of all three as being identical if we carefully craft our state space such that actions are part of states, just as we did before.

Another important assumption which is many times made, but need not be is the fact that the model and actions of an MDP are stationary. In other words, the physics or dynamics of the world does not change, which is a reasonable assumption to make in many cases. For instance, the Newtonian physics always applies in the daily world as we know it (i.e. in the macro-world). It can only be modified or disregarded at the microlevel, where quantum mechanics takes over and describes a new kind of physics.

The solution of a Markov Decision Process is referred to as a policy $\pi(s)$. It is basically a function that takes in a state of the world **s** and outputs the next action **a** that should be performed in this state. Furthermore, we can talk about an optimal policy $\pi^*(s)$, which is the best policy that maximizes the long term expected reward.

In order to understand more the nature of reinforcement learning, it is very important to grasp the concept of utility, which is somewhat different than that of a reward. A reward basically describes immediate gratification, while utility describes the long term return or reward of a sequence of states. In other terms:

$$U(s_0, s_1, s_2, ...) = \sum_{t=0}^{\infty} R(s_t)$$

If we consider only positive rewards, this expression basically describes the "Immortality Dilemma", which states roughly that if a human leaves forever, it really doesn't matter what he/she does at any point in time, because he/she can always get to a better place later. At any rate, the rewards will sum up to infinity. This dilemma is an incentive to create negative rewards and to also discount rewards over time such that utility becomes:

$$U(s_0, s_1, s_2, ...) = \sum_{t=0}^{\infty} \gamma^t * R(s_t)$$

Now, considering the theory of geometric series, discounted rewards will allow us to have a definite utility attached to any state, even if the number of states and rewards is infinite:

$$\sum_{t=0}^{\infty} \gamma^t * R(s_t) \leq \frac{MaxReward}{1-\gamma}$$

This is related to the futuristic concept of **singularity**, that states that one day computers will be powerful enough to produce new generations of computers in half the time, and therefore we will have infinite number of successors or computational power in finite time. The gist of this idea was also presented in the past in a different context as one of **Zeno's paradoxes**: "Achilles and the tortoise", that portrays how Achilles can never reach the turtle because although the distance to it is finite, Achilles has to overcome an infinite number of steps.

As a result, the optimal policy has the following expression:

$$\pi^*(t) = argmax_\pi E[\sum_{t=0}^{\infty} \gamma^t * R(s_t)|\pi]$$

and the utility of a state if we follow policy π looks as follows:

$$U^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t * R(s_t)|\pi, s_0 = s]$$

In more simple terms, utility is the immediate reward plus the average or expected reward that the agent would get from the current point and going on into the (infinite) future. Utility is more comprehensive and important than the simple reward scalar because it is tied to the notion of delayed rewards and temporal learning which is a key aspect of reinforcement learning. For example, we can do short term negative actions that cause suffering if in the long run they result in positive feedback and outcomes. We can connect actions temporally and judge about cause and effects in the long term by using the concept of utility.

4 Most common RL

Disclaimer: *This chapter uses equations taken from [12] for explanation purposes.*

Using the information presented in the background chapter, we can compile one of the most important equations of the RL field, namely: the Bellman Equation

$$U(s) = R(s) + \gamma * max_a \sum_{s\prime} T(s, a, s\prime) * U(s\prime)$$

If we are able to solve this expression, it effectively means we have a policy, since we can obtain the utility of any state and make the appropriate planning that would maximize the long term reward. However, as it can be observed, things are not so simple to solve analytically. Even though we have the number of equations equal

to the number of unknowns (say we have **n** states and **n** such bellman equations), the overall system is not linear because of the **max** operator, which is a very tricky mathematical non-linearity. Therefore, there should be special algorithms created in order to solve the system of equations in spite of its non-linearity. One such algorithm is called the "Value Iteration Algorithm" [12] It has the following steps:

- Start with arbitrary utilities
- Update utilities based on neighbours
- Repeat the updates until convergence to the true utility values

It is somewhat incredible that this algorithm works at all and to give a small intuition about why it does: it basically all boils down to the fact that the Bellman Equation has a ground truth term in it, namely the immediate reward: R(s). Over time this ground truth gets cumulated and the discounted rewards from other states converge also to a true value because those states themselves have this ground truth reward term in their equations and so forth. This is basically a recurrent system that converges because at the base level it has a truth factor attached to it.

By knowing the utilities we can effectively derive a policy to do planing. However, it is not necessarily the case that we need to know the exact utility values in order to do this. Just like in the case of Support Vector Machines in the field of supervised machine learning we can use the kernel functions without specifically deriving the mapping functions that map data points from one space to another (see [13] for details), the same kind of thing can be done here. Not all computations need to be performed exhaustively till the end and a rough estimate of utility values often suffices to derive good plans and sequences.

Another more practical way to find a policy is by doing what is called a "policy iteration" [12]:

- Start with policy π_0 as a guess
- Given a policy π_t calculate $U_t = U^{\pi_t}$ where

$$U_t(s) = R(s) + \gamma \sum_{s\prime} T(s, \pi_t(s), s\prime) U(s\prime)$$

  Note that the max operator is gone here.

- Improve the policy: $\pi_{t+1} = argmax_a \sum_{s\prime} T(s, a, s\prime) U(s\prime)$

As it turns out, most of the things that were discussed so far are actually very much related to the field of Markov Decision Processes, since they assume the fact that we know the transition probabilities $T(s, a, s\prime)$ In real life RL we actually do not have them and we only have access to a set of observations of the form: $< s, a, r, s\prime >$ Because of this, the value iteration algorithm is modified under a new name: Q-Learning. Q-Learning makes use of the Q function:

$$Q(s, a) = R(s) + \gamma \sum_{s\prime} T(s, a, s\prime) maxQ(s\prime, a\prime)$$

The Q value can be interpreted as the value for arriving in state **s**, leaving it via action **a** and proceeding optimally afterwards. What is useful about this is that we can represent the utility U(s) as: $U(s) = max_a Q(s,a)$ and the policy π as: $\pi(s) = argmax_a Q(s,a)$ If only we can have an algorithm that finds Q efficiently, we would be on the right track. The term Q-Learning is omnipresent in the RL literature, but it is important to understand that actually it stands for a whole family of learning algorithms. But just for the sake of "completion", one of the most intuitive algorithm is summarized by the following Q-Learning Update Equation:

$$\hat{Q}(s,a) = (1-\alpha)\hat{Q}(s,a) + \alpha[r + \gamma max_a \hat{Q}(s\prime, a\prime)]$$

, where α signifies the learning rate. In other more simple words, we basically start with some estimated Q values $\hat{Q}$ and we apply the update formula from above many times, such that if **s, a** pairs are visited infinitely often, than $\hat{Q}$ will converge to a true value of Q, solving the Bellman equation, and we will derive the transition probabilities for the world: $s\prime \sim T(s,a,s\prime)$ and the ultimate rewards: $r \sim R(s)$. What is also insightful is that the sum of learning rates: $\sum_t \alpha_t = \infty$ while $\sum_t \alpha_t^2 < \infty$, which basically suggests that we can always learn something (just like in life); there is technically no end to learning, however we can expect to understand things, connect the knowledge dots and converge to a finite knowledge base in a limited amount of time.

As previously mentioned, Q-Learning comprises many different algorithms. They can vary based on:

- How we initialize the first estimates $\hat{Q}$
- How we decay the learning rates α_t
- How we make action choices

One aspect that deserves special attention is the action choice topic. If for instance the agent always chooses an action **a** and no other whatsoever, it basically means that it will never learn. A more popular saying would apply here: You can't know what you've never tried, be it good or bad. But also from a mathematical point of view, we've mentioned that in order for Q learning to converge, all the (state, action) pairs need to be visited infinitely often and in this case they are not visited even once, so we can't talk about any convergence. Another bad idea would be to always choose randomly among the different actions. This basically means that the agent never takes advantage of the knowledge that it acquires over time. It makes no sense to even learn anything if at any given point in time an action will be picked randomly, that is: irrationally or not according to any policy, be it the optimal policy or an intermediate policy.

The reinforcement learning can also suffer from local minima just like any neural net based learning algorithms. Therefore, the agent can get stuck into picking the same action over and over again because the other alternatives simply seem too bad (computer equivalent of obsessive compulsive disorder). This can of course be avoided using the standard technique of restarting the algorithm with a new

initialization. However, this will take a lot of time because it essentially means performing a new training session from scratch. A better alternative would be to take some random actions once in a while that would "shake" the agent from "getting stuck". This can be intuitively portrayed as injecting some noise in the system that is a little destabilizing, and prevents the agent from getting stably stuck in the point of minimum. This last idea deserves special attention since it results in the state of the art way in which most RL algorithms are implemented in practice nowadays. The empirical trick is named ϵ greedy policy, which under other circumstances, especially in the field of shortest path algorithms is called "simulated annealing" [14]. Simply put, the agent behaves as follows: it takes the optimal action according to the current best policy with a probability $1 - \epsilon$, but it also takes random action with probability ϵ in order to learn new things and improve the policy. The simulated annealing idea is therefore a modified problem in RL that goes under the name of Exploration-Exploitation trade-off.

5 Model-Based vs Model-Free RL

In chapter 2, we have seen that biologically, animals have evolved three types of learning systems:

- Model based/ Tree-Search system
- Model free/ Habitual system
- Pavlovian Control / Evolutionary mechanisms

In chapter 4 we've also seen that the most well known and empirical RL is the model free approach. However, this approach is very statistically inefficient because it requires lots of trials and errors in order to compute a good evaluation look up table. Model based approach however is unfortunately an under-researched area of RL with important benefits. For example, it is very statistically efficient, as opposed to model free approach. Training can be done at the order of minutes with very few samples because the agent takes advantage of the task model encoded in its memory. Model free and model based methods have another important trade-off, namely the fact that model based is more computationally intensive at test time. In order to make a plan, a model based agent has to derive a decision tree based on the model, which can be potentially very big (ex: check Deep Blue chess engine that defeated Kasparov). A model free agent does not need to compute any tree, it just needs to look up the (Q)value of the state it's in currently. It's almost like the agent has a feeling/natural talent for what is good to do in a certain state, whereas the model based agent actually has to think a lot in order to know what to do. A good analogy would be that of a vocation talented person (model free) and an overachiever that actually has to put a lot of effort into a task (model based). Obviously, the appearances are misleading, because the talented person has to train a lot in order to have a natural feeling for the task and be quick at execution, whereas the overachiever has to train a lot less but be slow at execution or planning.

A common ambiguity might be that model free has resemblance with model based RL because the lookup table with values is essentially also a model. This is indeed true, the table can be thought of as a model. However, it is somewhat of a learned by heart model that is not general enough. If something in the environment changes because it might be a very dynamic world, the model free agent has to recompute the entire Q table just because a single Q value changed in a certain cell. A model based AI agent knows the more general model of the environment and hence changes in one part of the world do not trigger any heavy computations in order to relearn the dynamics of the world. It is in essence like knowing the analytical differential equation of a process. It will work with any random variable X because you can solve the equation analytically simply by plugging in the current instance of X. So to clear up the possible confusion and summarize, a model free RL knows the model "by heart" at a certain particular point in time and only partially and often needs to recompute it to keep it relevant over time.

Taking into consideration the above mentioned trade-offs we can see that model based and model free approaches are quite opposite. They are complementary in fact. Considering also the biological evidence that animals and humans exhibit a combination of both, it is no surprise that the best performing artificial learning systems are the hybrids that concatenate them together.

To summarize, give an intuition and support for the above claims, some relevant results were gathered from well-regarded and important papers, along with the tasks their authors were trying to solve:

Figure 4: RL Summary

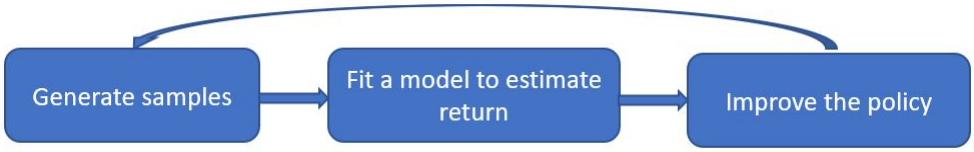


Figure 5: Model Free Training Efficiency (taken from [15]). *Model Free methods require hundreds of millions of steps and hundreds of thousands of episodes to train properly. Time-wise this implies days of training, sometimes even weeks or fortnights. The higher dimensionality tasks include: humanoid, cheetah, walker and fish*

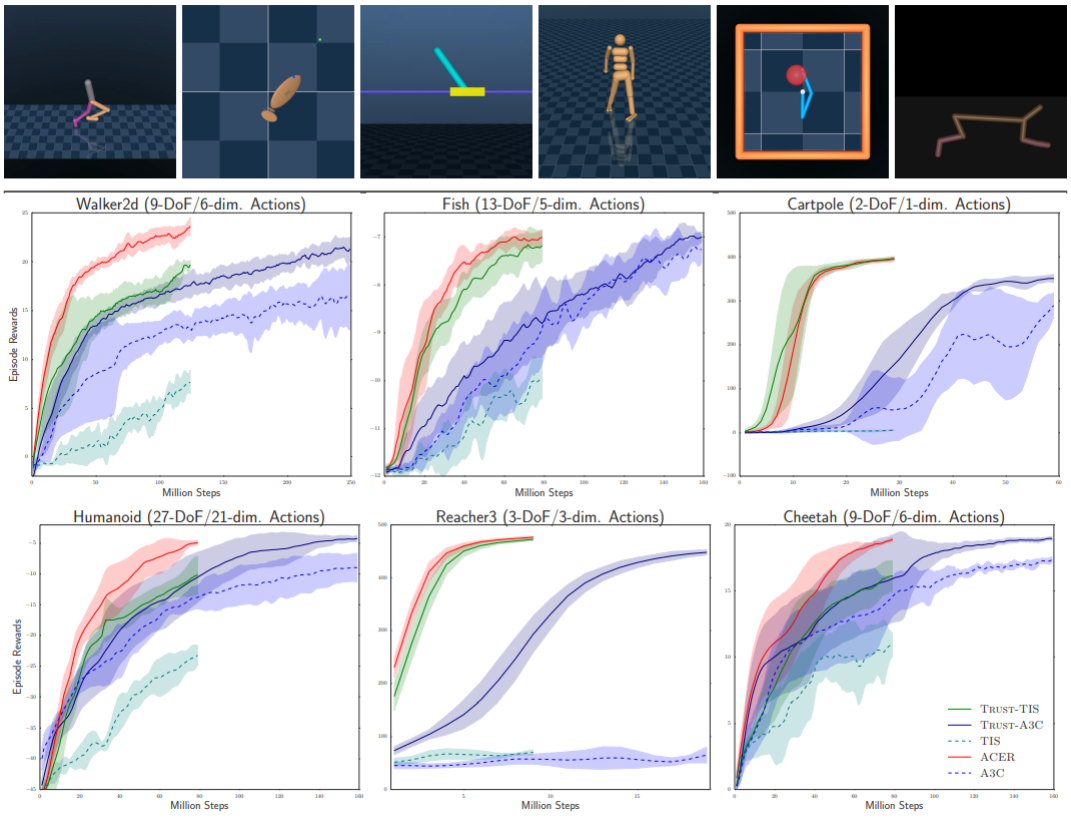


Figure 6: Model Based systems training efficiency (taken from [16]). *The task is to learn how to attach lego blocks, grasp various objects reliably, put a ring on a peg, etc.*

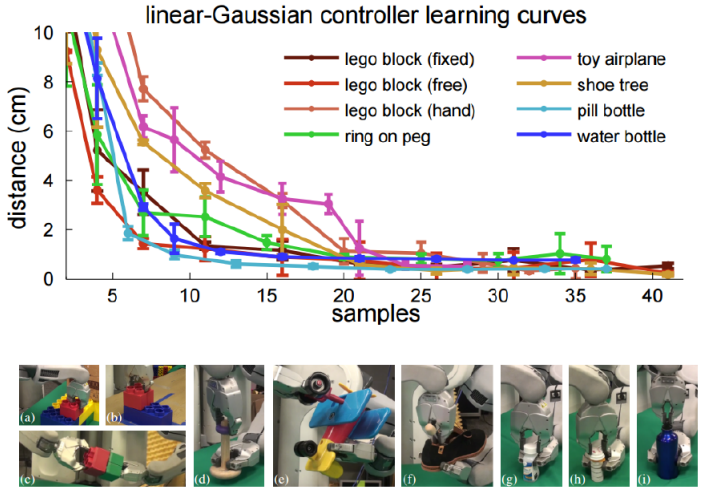
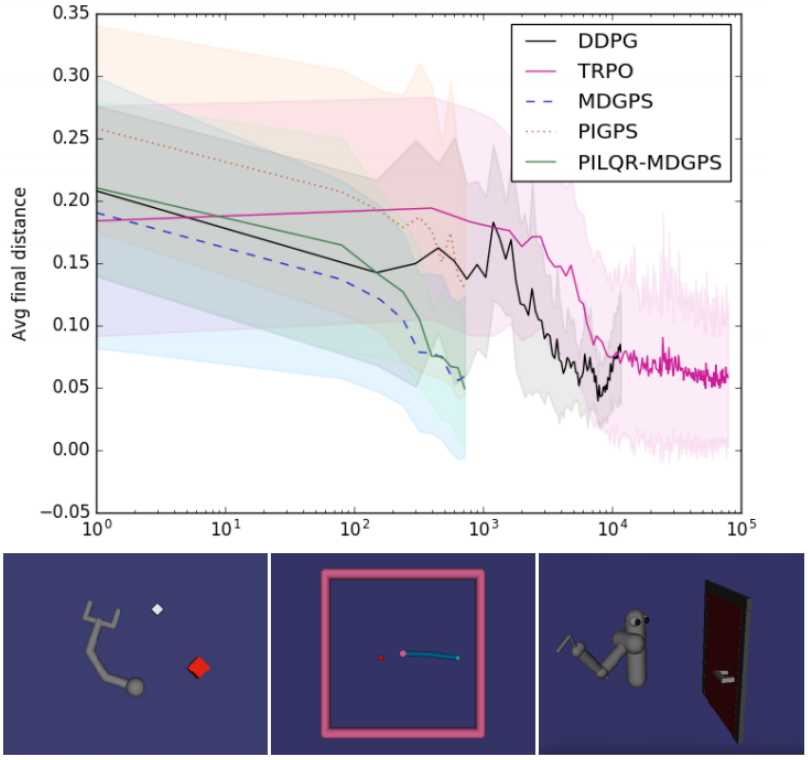


Figure 7: Hybrid systems (model based + model free) training efficiency (taken from [5]) *The authors argue that this hybrid training takes roughly 20 minutes of experience on a real robot. They evaluate on a set of simulated robotic manipulation tasks with varying difficulty. Left to right, the tasks involve pushing a block, reaching for a target, and opening a door in 3D*



As it can be observed, the model based method requires very few samples, which makes it ideal for it to run on real hardware directly. Model free is very inefficient, which makes it suitable only for a simulated environment. However, to achieve optimal performance, a hybrid system is the best. It has an in-between efficiency, requiring more samples than the purely model based approach, but it can perform arguably more complicated tasks, is more biologically plausible and runs in an acceptable amount of time on a real hardware platform (approximately anywhere from 20 minutes to 3-5 hours usually)

6 Discussions

Model based RL is starting to become more and more popular in the field of robotics with no virtual simulations because of the very quick training time and inherent goal of "planning" specific to the field of robotics. However, since model free RL is omnipresent nowadays in the research community, this chapter considers this type of RL in particular.

As it can be observed, reinforcement learning is indeed a mathematically sound, as well as biologically inspired approach to learning. However, it does not have the same success as other machine learning approaches. Nowadays, the most prominent thing people mean when referring to AI is actually: supervised learning. Reinforcement learning however also has a niche success in the field of gaming. And this is not surprising considering the fact that it needs a huge amount of examples in order to do any kind of training, much more than the supervised and unsupervised approaches. It is simply not economically feasible to for example destroy a robot arm 50000 times in order to make it do anything remotely meaningful. Hence, the gaming environments are the obvious candidates where all these simulations are possible and very cheap financially and even time-wise with the improvements in computer performance. Another bad thing is that, at least for now, they do not generalize well. If you train an agent to play a game of tennis, it will learn to be outstanding, but at the same time it will not know how to use the acquired knowledge to play a game of badminton. It will have to start all over again with the training process as if the two games are totally unrelated.

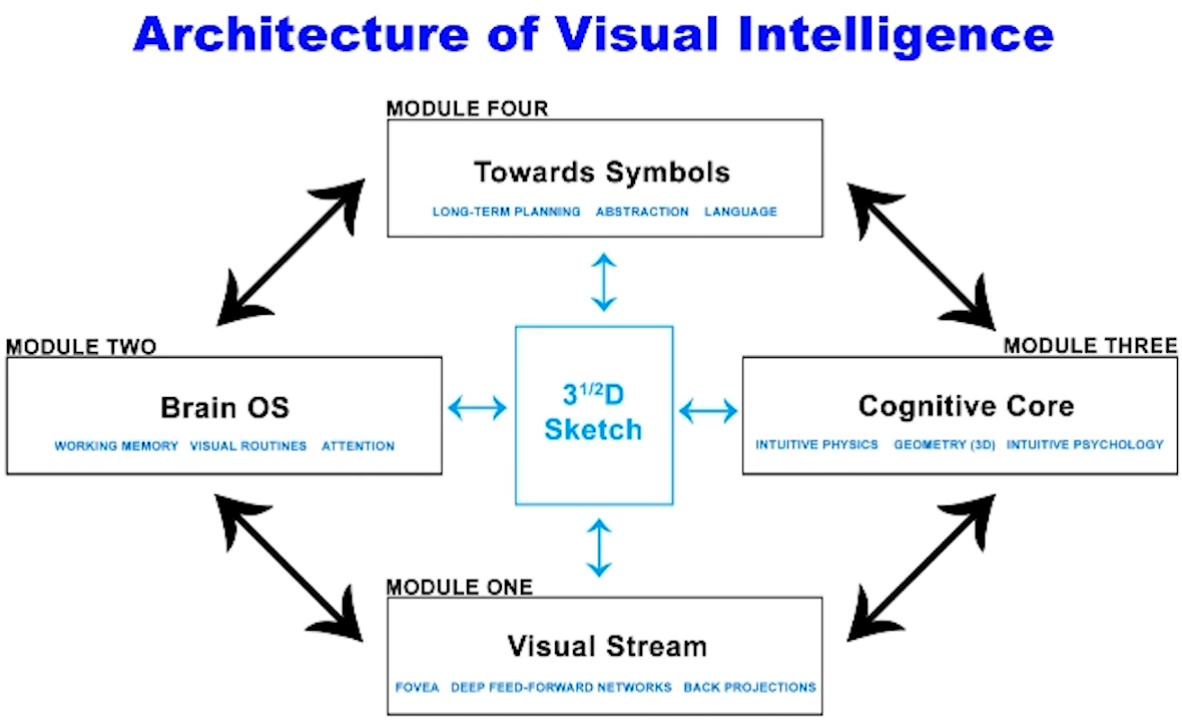
The good part about RL is that it scales well with the developing technology. The algorithms are suitable for distributed systems and can take advantage of more and more processing power. Considering that the brain has at least 100 billion neurons just in the cortex and even more so in the cerebellum and other areas, we need to have algorithms that are able to run on billions of processing units. Another good thing about reinforcement learning is that, even if we cannot train them on real physical agents, the field encourages and drives the development of simulators, just like the gaming industry encouraged the development of graphics cards that are ultimately very useful for AI. You can always train for instance an autopilot agent in a simulator and then port it to a real physical air-plane. In the end this is also related to the data generation problem, that is an important issue in many areas. For instance in biology, you can never have enough data to test

the effect of certain drugs, or how cells biochemically behave in certain conditions. Very good simulators that emulate the real world in different domains would be very useful in this context of data generation, as well as training. Finally, what is also promising about RL is that it is one of those approaches that can work end to end, throughout the following pipeline:

- Acquire Data
- Extract Features
- Represent Features
- Do machine learning
- Acquire knowledge
- Reason
- Plan
- Act

Typically other forms of AI only cover one or a combination of these items and must be used in combination with other blocks to form a full system. For example, feature extraction and feature representation is the task of supervised learning techniques like Convolutional Neural Nets that will create the perception block in the overall system. Machine Learning is generally what an unsupervised learning algorithm would do, or a supervised learning algorithm like a linear gradient descent or logistic regression algorithm. Planning and Reasoning is the task of Symbolic reasoning, logic and scheduling (see the ICAPS community) and so forth. RL has the potential to do all of it, but at the expense of many trials and very large amounts of data. Other researches within AI also fancy this idea of such a monolithic approach, but not necessarily RL inspired. For example Yann LeCun speaks about a dynamic neural net that would be able to restructure itself based on the task that needs to be learned. Andrej Karpathy also has similar ideas that he stated will be disclosed in future papers. However this does not seem to be the only main-stream hypothesis. Josh Tennenbaum for instance proposes the view of a distributed system in many papers, one of which being [17] (see also figure 8 from one of his talks in relation to this paper). The main argument is that the brain seems to be modularized in many aspects, although the algorithms can be very similar throughout all modules and very flexible. Therefore he argues we should aim at building and emulating a sort of brain operating system with special dedicated modules for different tasks. The same idea is promoted by Marc Raibert, the CEO of the well established robot company "Boston Dynamics", after years of trials and errors in the field of humanoid robotics amongst other things.

Figure 8: Visual Brain OS. Taken from Jeff Tennebaum's 2017 talk at MIT and in relation to paper [17] *Author tries to convey the idea that in order to solve intelligence, we need to aim for a modularized approach rather than an all encompassing monolithic system*



7 Conclusion

In conclusion, Reinforcement Learning is a promising AI technique with bio-inspired aspects. However, at least in its current form, it does not seem to be the answer for a totally general AI, so called AGI. What it does offer, is the proof of concept that AGI can be thought of as a monolithic type of system, a sort of black box if you will, that does many things and abstracts many details, from perception to reasoning. So it is powerful in this sense, but at the same time works in artificial setups like game environments and simulators that give scalar scores and quantify rewards by numbers. It is still an open question as to how we would quantify rewards in general real world tasks and how does the brain actually do it. It is also very unlikely that the brain operates only based on rewards of the dopamine system to learn new things. This is definitely one of the key elements that explains motivation and how people have incentives to learn new things and just perform daily activities. However, it does not seem to for example explain the visual intelligence that is one of the major components of the overall human intellect. Topics like visual attention and saliency, top-down and bottom-up models are not really grasped by RL. They can be helpful in combination with RL, to reduce the state space and aid the algorithm by emphasizing important states and features of the

search space. For instance, the large state space of the Montezuma's Revenge game described in [2], can also be potentially minimized by using saliency mechanisms as a helper technique for the new RL algorithm described in the paper to be able to pay more attention to salient features in the game frames.

All in all, reinforcement learning is worth mentioning as one of the most important AI implementations nowadays and has the potential to play a higher role more so in the future, even with the development of modularized AGI models.

References

[1] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *arXiv preprint arXiv:1707.06887*, 2017.

[2] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

[3] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[4] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, pp. 5074–5082, 2016.

[5] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," *arXiv preprint arXiv:1703.03078*, 2017.

[6] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

[7] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Advances in neural information processing systems*, pp. 3338–3346, 2014.

[8] V. Hung, A. Gonzalez, and R. DeMara, "Towards a context-based dialog management layer for expert systems," in *Information, Process, and Knowledge Management, 2009. eKNOW'09. International Conference on*, pp. 60–65, IEEE, 2009.

[9] C. D. Fiorillo, P. N. Tobler, and W. Schultz, "Discrete coding of reward probability and uncertainty by dopamine neurons," *Science*, vol. 299, no. 5614, pp. 1898–1902, 2003.

[10] A. M. Wikenheiser and A. D. Redish, "The balance of forward and backward hippocampal sequences shifts across behavioral states," *Hippocampus*, vol. 23, no. 1, pp. 22–29, 2013.

[11] W. A. Hershberger, "An approach through the looking-glass," *Animal Learning & Behavior*, vol. 14, no. 4, pp. 443–451, 1986.

[12] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[13] A. Patle and D. S. Chouhan, "Svm kernel functions for classification," in *Advances in Technology and Engineering (ICATE), 2013 International Conference on*, pp. 1–9, IEEE, 2013.

[14] Y. Wang, D. Tian, and Y. Li, "An improved simulated annealing algorithm for traveling salesman problem," in *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*, pp. 525–532, Springer, 2013.

[15] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[16] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 156–163, IEEE, 2015.

[17] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, 2017.