Algorithms for Reinforcement Learning

Introduction:

The essay gives insights into what Reinforcement Learning is, how it works, highlights its various problems, and highlights the problem of sparse reward setting and how it is overcome using latest research strategies, and specifically touches upon Evolution Strategy algorithm and its advantages when applied to Reinforcement Learning problems.

Understanding Reinforcement Learning:

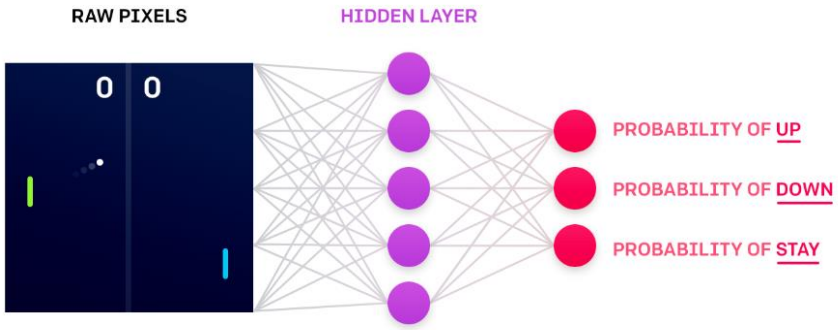
A core topic in machine learning is what we call Sequential decision-making. What action to perform in an uncertain environment in order to achieve some goals based on the prior experience? How does an artificial agent learn by interacting with its environment, similarly to a biological agent? In Supervised learning, input is fed to the neural network, and knowing the output the model should produce, one can train the network to produce intended result using gradient descent via backpropagation; however, there are some downsides to supervised learning - creating large data sets to train the model is a challenge, a model trained to simply imitate human actions can never be better than the human. For the agent to make decisions entirely by itself, we use Reinforcement Learning.

Working of RL:

The input frame is run through Neural network model in order to produce an output action. The agent doesn't know the target label, so it will not know in any situation which action to execute because there is no data set to train on. The agent explores different actions in the environment and these actions return values which are used to update the weights of the network.

The network that transforms input frames to output actions is called 'Policy Network'. One of the simplest ways to train a policy network is a method called Policy Gradients where we define a policy function, which computes how the agent should act in any given situation. A typical policy function might have about 1,000,000 parameters, so our task comes down to finding the precise setting of these parameters such that the policy plays well.

In Policy Gradients approach, we start off with a completely random network, feed the network with a frame from the control engine in order to produce a random output action. The produced action is sent back to the control engine and the control engine produces the next frame. This is how the loop continues. The network could be a full connected network or convolutions network. The output is usually something like the probability of different actions or a score. While training, sample from the distribution so that the same actions are not repeated. This will allow the agent to explore the environment a bit randomly and discover better rewards and better behavior. Since the agent must learn entirely learn by itself, the feedback is given in the form of scores comprising of rewards and penalties. If the agent manages to hit the target, it will receive a reward and if it misses the target, it gets a penalty.



The goal of the agent is to optimize its policy to receive as much reward as possible. To train the policy network, collect a bunch of experience, run the whole bunch of input frames through the network, select random actions, feed them back into the engine and create a whole bunch of input output combinations. Though the agent doesn't learn anything here, sometimes it might select a whole sequence of actions that lead to hitting the target. In this case, the agent receives a reward. For every episode, regardless of positive or negative reward, we can compute the gradients that would make the actions that the agent has chosen more likely in the future. Whenever it is a positive reward, the normal gradients increase the probability of those actions in the future. Whenever it's a negative reward, the normal gradients decrease the probability of those actions in the future. This means we can then use backpropagation to compute a small update on the network's parameters that makes the actions that leads to negative rewards slowly filtered out and the actions that leads to positive rewards become more and more likely. Here's a blog 'Pong from Pixels' which helps understanding the RL process better.

Some of the downsides of Policy Gradients:

- **Reduce the likelihood of actions in the future** - Policy gradients assume that since an episode is lost, all the actions associated with that episode are bad actions and it reduces the likelihood of taking those actions in the future. For most part of episode, the performance has been good, and therefore wouldn't want to decrease the likelihood of those actions.
- **Sparse reward setting** - Instead of getting a reward for every single action, the agent only gets a reward after the entire episode. The agent needs to figure out from sparse feedback what parts of its action sequence enables it to get a positive reward. In some cases, the sparse reward setting fails completely wherein the agent doesn't get to see a single reward due to the random actions. The sequence of actions that the agent needs to take to get a reward is too complicated and the agent never gets there due to random actions. Policy gradient wouldn't know what to do as it never sees a positive reward.

The problem of sparse reward setting is overcome by using **Reward Shaping**, where in a reward function is manually designed in order to guide the policy to some desired behavior. But there are some downsides to Reward shaping as well:

- Reward shaping is a custom process that needs to be redone for every new environment we want to train a policy in. This is not scalable.

- Reward shaping suffers from alignment problem. In many cases the agent gets lot of rewards but doesn't accomplish the intended task. The policy is overfitting to that specific reward function that is designed and not generalizing to the intended behavior.
- Reward shaping is also not optimal in cases where it constrains the policy to the behavior of human.

Some solutions to overcome the problem of sparse reward setting:

- **Augment the sparse extrinsic reward signals** with additional dense feedback reward signals that need to help the learning of the agent. The dense additional signals should be in some ways related to the task that we want the agent to solve. Whenever agent succeeds in those tasks, it's probably also going to get knowledge or get feature extractors that can be useful in the sparse tasks that we really care about.
- **Add additional learning goals** - In most of the reinforcement learning settings, our agent is presented with raw input data like sequences of images. The agent will then apply feature extraction pipeline in order to extract the useful information from those raw input images. The policy network will use those extracted features in order to perform an intended task. The problem in reinforcement learning is that the feedback signal can be so sparse that our agent never succeeds in extracting useful features from the input frames. One successful approach in this case is to add additional learning goals to our agent that leverage the strengths of supervised learning to come up with very useful feature extractors on those images.
- **Injecting noise into agent's actions** - we can introduce exploration into the learning process by injecting noise into the agent's actions, which we do by sampling from the action distribution at each time step.

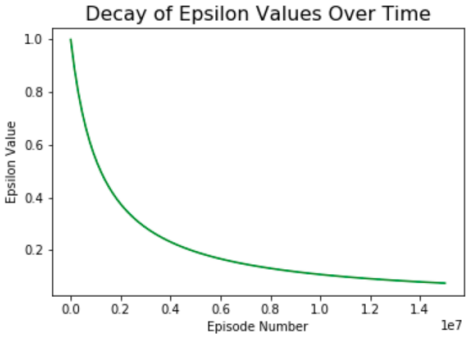
Highlights from some of the key research papers to drive exploration by agents:

***Reinforcement learning with unsupervised Auxiliary tasks*:** Here, the standard sparse reward signal is that agent is walking around in a 3D maze and it needs to find specific objects. When ever it encounters one of those objects, it gets a reward. In this paper, the sparse reward signal is augmented with 3 additional reward signals.

- In the first auxiliary task, the agent learns to do pixel control. So given a frame from the environment, it uses the feature extraction pipeline to learn a separate policy to maximally change the pixel intensities in some parts of the input images. In the suggested implementation, the input frame is divided into small number of grids and a visual change score is computed for each grid. The policy is then trained to maximize the total visual change in all the grids. This will force the feature extractor to become sensitive to the general dynamics in the environment.
- Second auxiliary task is reward prediction: Here agent is given 3 recent frames from the episode sequence and it's tasked to predict the reward that it will get in the next step. Again, we are adding additional learning goal to the agent that is only used to optimize the feature extraction pipeline that we think is generally useful to the eventful goal we care about.
- Third auxiliary task is value function replay which basically estimates the value of being in the current state by predicting the total future reward that the agent is going to get from this point onwards.

By adding these additional learning goals to the agent, we can significantly increase the sample efficiency of our learning agent.

Curiosity driven exploration: Incentivize your agent to learn about new things that it discovers in its environment. We use Epsilon Greedy Exploration where the agent selects the best possible move based on its current policy but with a small probability of epsilon, the agent is going to take a random action. This epsilon starts out at 100% in the beginning of training so it's completely doing random things and as we progress in training, it's going to start declining the epsilon value and, in the end, the agent completely follows its current policy. The idea is to make the agent explore its environment. The agent quickly learns some type of very simple behavior that earns a recurring low amount of rewards. But if the environment is hard to explore, a simple agent will never be able to explore the full environment in search of better policies. Then incentivize your agent to explore unseen regions using 'Forward Model'.



Forward model: The agent sees a specific input frame and encodes the input using a feature extractor into latent representation. The forward model that predicts the same latent representation for the next frame in the environment. The agent learns the dynamics of the environment based on what it is seeing now and then predicts what's going to happen next. The assumption is that if the agent is in place where it has been many times before, these predictions will be accurate, but if the agent is in a new situation it has never encountered before, then its forward model will probably not be that accurate. We can then use these prediction errors as an additional feedback signal on top of the sparse rewards to incentivize the agent to explore the unseen areas of the state space.

Example of Intrinsic curiosity model - Imagine a scenario where the agent is observing the scenario of movement of tree leaves in a breeze. It's hard to exactly model the breeze predicting pixel changes for each leaf. This means that the prediction error in the pixel space always remain high and the agent will be forever curious about the leaves you need. The agent is not aware that there are some parts of the environment that it simply cannot control or predict.

The difference between actual action and the predicted action is fed back into the agent in addition to the sparse reward signal. Since the motion of the leaves cannot be controlled by the action of the agent, there is no incentive for the feature encoders to model the behavior of those tree leaves because in the inverse model, those features will never be useful for actually predicting the action the agent took. Therefore, the resulting features from our extraction pipeline will be completely unaffected by irrelevant aspects of our environment. And we will get a much more robust exploration strategy. Getting the agent to efficiently and intrinsically explore the environment is a very crucial part of learning.

Hindsight Exploration Replay (HER): The general idea behind HER is that they want to learn from all episodes even if an episode was not successful for the actual tasks we want to learn. Say an agent has to push the object to position A, but the agent moved the object to position B since the policy is not very good yet. Instead of telling the agent it will get a reward of 0, pretend that going to position B is the intended action the agent has to perform. Agent now learns how to move to position B. This is creating a dense reward setting from a sparse reward problem. Initialize all the parameters and start playing the episodes. Given a goal position, using the current policy, we get a trajectory and we get a final position where the object ended up in.

After the episode is completed, we store all the transitions in the replay buffer with the goal that was selected for the policy. We also sample a set of modified additional goals and swap those out in the state transitions and store everything in the replay buffer. To move the object to a new location, we don't have to retrain the policy, change the goal vector and policy would do he right thing.

Other problems in Reinforcement Learning:

- The training data that is generated is itself dependent on the current policy because the agent is generating its own training data by interacting with the environment rather than relying on static data set as in the case of the supervised learning. This means that the data distributions of the observations and rewards are constantly changing as the agent learns which is the major cause of instability in the training process
- RL also suffers from high sensitivity to hyper parameter tuning (for example initialization). If the agent's learning rate is large, there could be a policy update that pushes the policy network into a region of parameter space. In parameter space, the policy network collects the next batch of data under a poor policy causing it to never recover again.

Evolution Strategy (ES) and Genetic Algorithm

In Reinforcement Learning, the weights of the network are updated based on the values returned by various actions when agents explore different actions in the environment. This can be referred to as searching in action space. ES is simply a class of black-box stochastic optimization techniques that searches for the best weights in the network and returns a score on the performance of any set of parameters. This is known as searching the parameter space. For all the input parameters that goes in, one output total reward comes out. The goal of ES is to find the best setting of all the input parameters. It is as good as optimizing a function f(w) with respect to the input vector w (the parameters / weights of the network). ES uses gradient descent and estimates the gradient of the expected reward in the parameter space using finite differences. ES is invariant to action frequency and delayed rewards, tolerant of extremely long horizons, and does not need temporal discounting or value function approximation

We start with a parameter vector w and generate say 100 different parameter vectors. Evaluate each of the 100 vectors independently by running the corresponding policy network and add up all the rewards in each case. The updated parameter vector then becomes the weighted sum of the 100 vectors, where each weight is proportional to the total reward. The most successful candidates will have higher weights. RL injects noise in the action space and uses backpropagation to compute the parameter updates, while ES injects noise directly in the parameter space.

Genetic Algorithm(GA) is an evolutionary based algorithm that uses Novelty Search (NS) technique to encourage exploration in tasks with deceptive or sparse rewards. When applied to the problem of 'Image Hard Maze', it outperformed other gradient and evolutionary based algorithms that optimize solely for reward. GAs do not follow the gradient, instead they use exploration through mutation and exploitation through selection to evolve the individuals of a population. It saves computation because there is no gradient calculation. Evolutionary computing algorithms in general and GAs specifically have had much empirical success on a variety of difficult design and optimization problems. They start with a randomly initialized population of candidate solution typically encoded in a string (chromosome). A selection operator focuses search on promising areas of the search space while crossover and mutation operators generate new candidate solutions.

Advantages of ES:

- No need for backpropagation - It's less susceptible to confusion for tasks involving long time horizons or delayed rewards which is a problem for reinforcement learning with backpropagation.
- Highly parallelizable - It's easy to parallelize and can therefore be much faster.
- High robustness - Several hyperparameters that are difficult to set in RL implementations are side-stepped in ES
- Structured Exploration – Due to random policies of RL algorithms, agents do not perform consistent action for a while. ES does not suffer from these problems because we can use deterministic policies and achieve consistent exploration.
- Overcomes Credit Assignment problem
- The genetic algorithm searches through the space of parameter values for values that maximize task performance and minimize the number of training epochs

Downside of ES:

One core problem is that for ES to work, adding noise in parameters must lead to different outcomes to obtain some gradient signal. Further work on effectively parameterizing neural networks to have variable behaviors as a function of noise is necessary

Conclusion:

Though each of the algorithms do better than the other for specific Reinforcement problems, combination of algorithms based on (deep) reinforcement learning promise great value to people and society. For example, when we use a genetic algorithm (GA) to find the values of parameters used in Deep Deterministic Policy Gradient (DDPG) combined with Hindsight Experience Replay (HER), it helps speed up the learning agent. This can significantly impact the overall learning process. Heuristic search as performed by genetic and other similar evolutionary computing algorithms are a viable computational tool for optimizing reinforcement learning performance in multiple domains.

The RL algorithms have the potential to enhance the quality of life by automating tedious and exhausting tasks with robots, improve quality of education by providing adaptive content, improve public health, provide robust solutions to self-driving car challenges and many more applications.

My Future Interest:

We need to careful that deep RL algorithms are safe, reliable and predictable. We design a reward function based on what we want the agent to do. Many times, the reward functions are designed arbitrarily. Sometimes, the arbitrary design produces unexpected, catastrophic behaviors. All aspects related to safe exploration are also potential concerns in the hypothesis that deep RL algorithms are deployed in real-life settings. In order to obtain maximum cumulative reward, the agent may execute actions which is not the intended behavior. I would like to do some research around AI safety. The following links could be a starting point for my exploration.

https://futureoflife.org/ai-safety-research/

https://www.fhi.ox.ac.uk/wp-content/uploads/Interruptibility.pdf

References:

Sehgal, Adarsh, et al. "Deep Reinforcement Learning Using Genetic Algorithm for Parameter Optimization." *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, doi:10.1109/irc.2019.00121.

Sewak, Mohit. "Introduction to Deep Learning." *Deep Reinforcement Learning*, 2019, pp. 75–88., doi:10.1007/978-981-13-8285-7_6.

Stanley, Kenneth O., et al. "Welcoming the Era of Deep Neuroevolution." *Uber Engineering Blog*, 2 Oct. 2018, eng.uber.com/deep-neuroevolution/.

"Using Evolutionary AutoML to Discover Neural Network Architectures." *Google AI Blog*, 15 Mar. 2018, ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html.

"Using Machine Learning to Explore Neural Network Architecture." *Google AI Blog*, 17 May 2017, ai.googleblog.com/2017/05/using-machine-learning-to-explore.html.

Jaafra, Yesmina, et al. "Reinforcement Learning for Neural Architecture Search: A Review." *Image and Vision Computing*, vol. 89, 2019, pp. 57–66.

Kamei, Keiji, and Masumi Ishikawa. "Reduction of Computational Cost in Optimization of Parameter Values in Reinforcement Learning by a Genetic Algorithm." *International Congress Series*, vol. 1291, 2006, pp. 185–188.