Abstract

There isn't a human alive who can triumph over Deepmind's AlphaGo Zero in the game of Go, however this does not make the AI super-intelligent in that a child may beat it at draughts. Obtaining a more general intelligence requires 'lifelong learning', i.e. an AI that can remember and learn from all of its life experiences to master more than a single problem.

In reinforcement learning (Sutton & Barto, 1998), the ***policy*** is considered to be the probability over the set of possible actions given the state of an environment. Existing techniques for continuous reinforcement learning (CRL) such as Composition Value Functions of James et. al (2018) and CAPS of Li et al. (2018) scale poorly as the number of tasks for an agent grow due to the need to store many learned policies.

This thesis concerns the ***Kullback-Leibler Policy Chain*** method which helps an AI to generalise over several tasks in a scalable manner. Properties of the Kullback-Leibler divergence are exploited across specific polices to create a dependence between them. The need to store historic policies as some existing methods do is then greatly reduced with the introduction of a pseudo running total or a *chain* of these polices. Within this policy chain, we can say that the youngest link (i.e. policy for most recent task) will inherit the prevailing memories from older policies throughout the chain.

Introduction

The environment used in this experiment will be a GridWorld of varying sizes. Both the agent's starting location and the goal will be at random locations and the agent will continue to search until the negative reward limit is reached. A Policy Gradient approach will be taken to solve for the optimal policy (see Background). To facilitate an environment of CRL, the goal within the GridWorld will be changed after the policy for that goal is considered optimal. The agent will then be required to learn or re-learn the optimal route to the new goal location from anywhere in the grid. The aim of this experiment is to appropriately select an optimal prior policy for when this occurs, without needing to store a large dictionary of previously learned policies to consult.

Catastrophic forgetting (REF) is an event that can occur when we attempt to solve a multitask environment with traditional policy gradient methods. If we consider each task as a unique goal location within the grid, catastrophic forgetting can take the following form in a RL setting:

1. Optimise policy for task 1.
2. Change location of goal (i.e. begin task 2).

3. Optimise policy for task 2.
4. Agent can no longer perform task 1, and may find it impossible to relearn.
5. Catastrophic forgetting has occurred.

Inspiration for this thesis was originally taken from the research into entropy-regularized policy gradient methods of Mnih et al. (2016) which attempted to address catastrophic forgetting. This experiment regularised the posterior policy of an agent such that the policy π_a of an agent must remain similar to some other policy π_0. The aim of this thesis is to create a cost function that is optimised by the optimal prior policy, such that this optimal policy is highly susceptible to learning useful re-used traits of previous policies. Utilised correctly, a chain of these policies could quickly lead to a prior policy that directly inherits the most common traits of historic policies, without having to store the policies themselves. In the context of the GridWorld environment, it means that the prior policy π_i for goal location i will already know not to leave the grid.

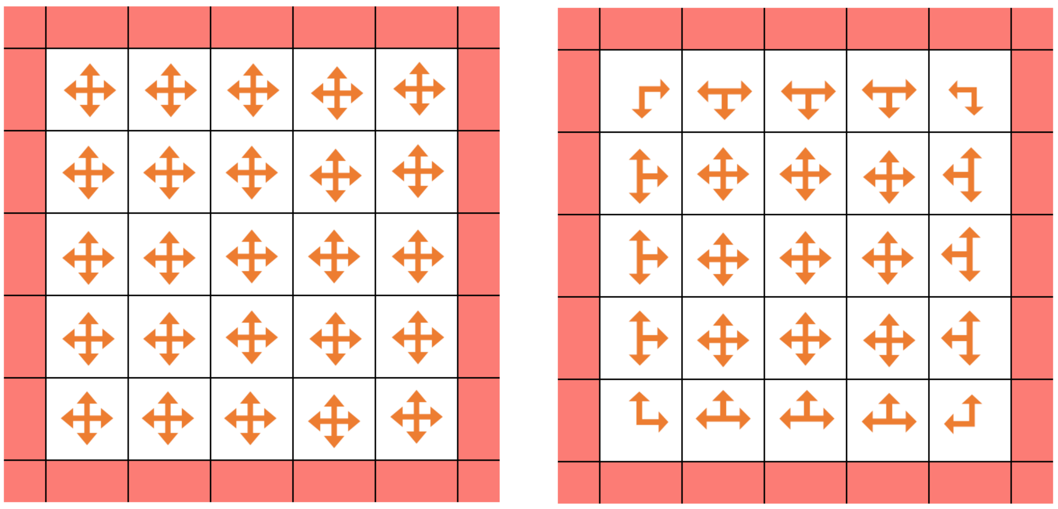
Fig. 1: Uniform prior policy versus worst-case optimal prior policy

It is important to note that the above figures are not truly representative of what occurs in a Kullback-Leibler policy chain (KLPC) and this will be addressed further in the thesis. EDIT: ‘Uniform’ policies in the inner squares will likely have different probabilities of each direction, i.e. P(Left | Inner square) is not necessarily 0.25 (so are not actually uniform)

Background

Reinforcement Learning

Each reinforcement learning setting can be described in terms of tuples of a Markov Decision Process (MDP). A tuple is defined as $\{S, \mathcal{A}, p_s, r\}$ whereby at each time step t an agent observes the state $s_t \in S$, takes an action $a_t \in \mathcal{A}$, resulting in a reward $r(a_t, s_t)$and a transition to the next state s_{t+1} with probability $p_s(s_{t+1}|\, s_t, a_t)$. The goal of the agent is to find a policy, defined by a probability distribution over actions given the state $\pi(a_t, s_t)$, that maximises its expected sum of discounted future rewards:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_\pi[\, r(s_t, a_t)\,]$$

where $\mathbb{E}_\pi$ is the expectation under the reward distribution defined by policy π (Kaplanis et. al, 2018).

Policy Gradient Methods

Policy gradient methods are a technique employed to solve directly for the optimal policy for any given task, given a large set of policies Π such that:

$$\Pi = \{\pi_\theta, \theta \ \in \mathbb{R}^m\}$$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi_\theta\right]$$

The optimal policy is then defined as:

$$\theta^* = \arg\max_\theta J(\theta)$$

REINFORCE Algorithm

Let's say that our agent follows a certain trajectory τ through the MDP such that $\tau = \ (s_0, a_0, r_0, s_1, \dots)$. Now let $r(\tau)$ be the reward for this trajectory. We can now express a policy's value as an expectation of the reward that we can get from any policy π_θ:

$$J(\theta) = \mathbb{E}_{\tau\sim p(\tau;\theta)}[\, r(\tau)\,]$$

$$= \int_{\tau} r(\tau) p(\tau;\theta) d\tau$$

The desired outcome of this algorithm is to use gradient ascent on $J(\theta)$ to solve for the optimal parameter values such that:

$$\theta_{\tau+1} \leftarrow \theta_{\tau} + \alpha \nabla_{\theta} J(\theta)$$

However, the gradient of $J(\theta)$ is an intractable integral:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \int_{\tau} r(\tau) p(\tau;\theta) d\tau$$

$$= \int_{\tau} r(\tau) \nabla_{\theta} p(\tau;\theta) d\tau \ldots intractable$$

We therefore use a trick (Williams, 1992) to express $p(\tau;\theta)$ in terms of the log function which allows us to utilise Monte Carlo sampling for :

$$\nabla_{\theta} p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_{\theta} p(\tau;\theta)}{p(\tau;\theta)}$$

$$= p(\tau;\theta) \nabla_{\theta} \log p(\tau;\theta)$$

$$\therefore \nabla_{\theta} J(\theta) = \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau;\theta)) p(\tau;\theta) d\tau$$

$$= \mathbb{E}_{\tau \sim p(\tau;\theta)} [\, r(\tau) \nabla_{\theta} \log p(\tau;\theta) \,]$$

ASK IVANA ABOUT CODE IMPLEMENTATION OF ACTOR CRITIC

So use this to find policy which will be used to create the next link in the chain

(Li et. al, 2017)

Kullback-Leibler Divergence

To Do

1. Create script to create figures to show how this works
2. Need Polcies first – i.e. dependent on actor critic working.

Experimental Theory

Take large portions from earlier article draft