Baum-Welch Algorithm for Bottom Up Option Learning by Imitation

Bachelor's Project Thesis

Davide Barbieri, s3090078, d.barbieri@student.rug.nl
Supervisors: Dr M.A. Wiering

Abstract: The Baum-Welch algorithm (Jelinek, Bahl, and Mercer, 1975) was developed to fit a hidden Markov model that maximizes the likelihood of a given sequence of observations (with a given number of hidden states). Fox, Krishnan, Stoica, and Goldberg (2017) proposed to look at option learning by imitation as solving a hidden Markov model, where the observations are primitives in a given solution trajectory, and the options are hidden probability distributions over primitives generating such trajectory. This paper adapts the Baum-Welch algorithm to fit the above idea by accounting for the differences between an option hierarchy and a hidden Markov model. The results seem to suggest that the architecture is capable of learning by imitation a series of static options. It is also able to exploit them to solve previously unseen tasks, similar to the ones solved by the imitation data.

1 Introduction

Reinforcement Learning methods (Wiering and van Otterlo, 2012) are widely used to learn control policies that maximize the future expected rewards in a given task. While such algorithms can handle a wide variety of problems, the complexity of the environment or the required strategy (solution) can often slow down learning.

One possible solution, originating from Hierarchical Reinforcement Learning (Hengst, 2011), is to factorize the agent's strategy into a hierarchy of policies, having an abstraction of the agent's actions (Precup, 2000). Therefore, higher-lever (or more abstract) policies solve the problem in terms of temporally extended actions. Each temporally extended action is a sub-problem of the ongoing task which is handled by a lower-level controller. The option framework (Sutton, Precup, and Singh, 1999) is an implementation of action abstraction, and has been embedded with Reinforcement Learning algorithms (Stolle and Precup, 2002). An option wraps a policy by having a stop condition so that once it has control, it can give it to simpler options (when acting) or back to its calling option when its task is completed. Thus, option based policies present two main advantages for the agent's exploration and its learning.

The former is improved by having an option (and its descendants) keeping control until the task is completed. Therefore, to a higher-level controller, a single time-step would have passed since the call of an option until its end, since to it calling a lower-level option will correspond to performing a single action. Thus the stopping condition enforces the temporal extension of abstract actions.

The option framework can improve the learning of the agent through *transfer learning* (Pratt, 1993), that is, the possibility to reuse the (procedural) knowledge learned within an option in similar tasks. One of the issues, in this case, is to fit the option's strategy in such a way that it can be reusable. To do so, its policy needs to solve a subtask which is, generic enough to be a factor of multiple super-tasks, but also specific enough that it can be learned, otherwise the advantages of action abstraction are lost. Another issue is that the strategy to be generic should be fit as independently as possible from the current task the agent is solving (so that the option may be reused in other tasks); however, this poses a problem if we want the agent to then learn the option by itself.

A possible solution to this is given by Fox et al. (2017). In their paper, they propose to learn these

options by imitation and look at option learning as solving a Hidden Markov Model (HMM). More precisely, in HMMs, we have hidden states which can each generate observations with certain probabilities where transitions between states are also regulated by a transition probability distribution. Given a series of observations, one can compute the parameters of the model in such a way to maximize the likelihood of observing such a sequence, without knowing what the sequence of hidden states that generated the observations was.

The idea from Fox et al. (2017) is to see the actions performed by an agent as the observations and its options as hidden probability distributions over the observable actions. This is not dissimilar from an HMM as we want to compute the model's parameters to try to maximize the likelihood that a series of options would generate such a sequence of actions. Assuming that the observed agent (and thus the imitation data) is performing a (close to) optimal strategy, the goal is to fit which (hidden) options the agent was running, and then use them in our agent's strategy.

In this paper, we re-adapt the Baum-Welch algorithm (Jelinek et al., 1975), widely used to solve HMMs, to our problem, taking into account that hidden states are now options. As such, two main differences from normal HMMs are present. First, the transitions between hidden options and the probabilities to produce observations both depend on the current state (as they are policies), requiring an extra dimension. Second, the transition function between options depends on the controller of the agent and is thus constrained to follow its strategy for selecting options.

1.1 Background

In this section we will introduce formal definitions for basic concepts needed to understand this paper.

1.1.1 Reinforcement Learning

Reinforcement Learning aims at solving a Markov Decision Process (MDP). Formally, a task is described in terms of a set of states S and actions A (also called *pimitives*) which can be performed in each state. Performing an action in a state results in a transition to any state in S . The Markov property states that the probability of transitioning to a state is a function of only the current state and the currently performed action. The probability of transitioning to state s_j being in state s_i and performing action a is defined by the transition function $T : S^2 \times A \to [0,1] \ni T(s_j \mid s_i, a) = p(s_{t+1} = s_j \mid s_t = s_i, a_t = a)$. Finally performing an action in a state results in a reward for the agent specified by the reward function $R : S \times A \to \mathbb{R}$.

The goal is to find a policy $\pi : S \times A \to [0,1] \ni \forall_{s \in S} \sum_{a \in A} \pi(a \mid s) = 1, \quad \pi(a \mid s) = P(a_t = a \mid s_t = s)$ which maximizes the expected discounted sum of all future rewards. Formally the goal is to find a control policy π^* such that:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \underset{a_t \sim \pi(s_t)}{\mathbb{E}} \Big[\sum_{t \in [0,\infty)} \gamma^t R(s_t, a_t) \Big] : \gamma \in [0,1]$$

Given the Markov property, π^* only needs to take the action that will maximize the expected discounted sum of future rewards from the current state on. We define this quantity as the value of a state:

$$U_\pi(s) = \underset{a_t \sim \pi(s_t)}{\mathbb{E}} \Big[\sum_{k \in [0,\infty)} \gamma^{t+k} R(s_{t+k}, a_{t+k}) \ \mid s_t = s \Big] \tag{1.1}$$

To relate the value of a state s_t to performing a primitive in it, we factorize U depending on the primitive performed at time t . This is done by the Q function, that gives the expected discounted sum of future rewards from the current state on, given that an arbitrary primitive is performed. Formally:

$$Q_\pi(s,a) = \underset{a_t \sim \pi(s_t)}{\mathbb{E}} \Big[U_\pi(s) \mid a_t = a \Big] \tag{1.2}$$

where $\pi^*(a \mid s) \propto Q_\pi(s,a)$ to ensure that the utility is maximized.

1.1.2 Option Framework

An option $h \in H$ is a tuple $\langle \mathcal{I}_h, \pi_h, \psi_h \rangle$ (Sutton et al., 1999), where $\mathcal{I}_h \subseteq S$ are the states in which h can be used, $\pi_h : S \times A \to [0,1]$ is the strategy used by the option, and $\psi_h : S \to [0,1]$ is the *stopping criterion*.

In this paper, the agent has a main controller to select the current running option, formally $\eta : S \times$

$H \to [0,1] \ni \eta(h \mid s) = P(h_t = h \mid s_t = s)$. Therefore, $\mathcal{I}_h$ is redundant in the option's definition and will thus be removed since, if an option h should not be used in a certain state s , then $\eta(h \mid s) \approx 0$. At each time-step t the option running in the previous step, $h_{t-1} = i$, stops with probability $\psi_i(s_t)$. If it does stop, then a new option $h_t = j$ is selected with probability $\eta(j \mid s_t)$, and then a primitive $p \in A$ is performed with probability $\pi_j(p \mid s_t)$. If i does not stop, then a primitive $p \in A$ is performed with probability $\pi_i(p \mid s_t)$.

2 Methods

As previously mentioned, bottom-up option learning is a problem comparable to finding the best fitting model of an HMM. Finding the best hierarchy of options fitting a given trajectory of observed primitives, would constitute performing imitation learning. The agent would effectively be fitting the policy that it observed another agent perform (thus assuming that the other agent's policy is close to optimal).

The hierarchy of options can be seen as an HMM. What we can observe are the primitives performed, while the agent performing them transitions through hidden options. The main goal-focused policy η (also hidden) instead determines which option to select.

Therefore the transition probability from option i to option j , with current state s is

$$\begin{aligned} a_{ij}(s) &= P(h_t = j \mid h_{t-1} = i, s_t = s) \\ &= \begin{cases} 1 - \psi_i(s) & \text{if } i = j \\ \psi_i(s)\eta(j \mid s) & otherwise \end{cases} \end{aligned} \tag{2.1}$$

where the probability of staying within the same option (a_{ii}) is the inverse of the probability of that option stopping (thus equal to the probability of that option keeping control). The probability of transitioning from an option i to a different option j is the conjunction (product) of the probability of i stopping and then of j being selected.

The probability of a hidden option i generating an observation k , in state s is simply

$$b_i(k \mid s) = P(a_t = k \mid h_t = i, s_t = s) = \pi_i(k \mid s) \tag{2.2}$$

The prior probabilities are defined by η for the initial state s_0 and are therefore given by

$$\rho(i) = P(h_0 = i) = \eta(i \mid s_0) \tag{2.3}$$

for any option i .

We propose a modified version of the Baum-Welch EM algorithm to fit our options correctly. The main changes account for the fact that the transition and observation probability distributions are functions of the current state (adding one more dimension).

After fitting the options, the higher-level policy (selecting an option every time control is given to it) keeps training with standard model-free methods from Reinforcement Learning.

2.1 Learning from imitation

We consider our imitation data a trajectory of primitive actions performed and the states at the respective time-step. Formally, a trajectory $\zeta_{0:T} = (p_0, ..., p_T) \ni p_i \in A$ and $\chi_{0:T} = (s_0, ..., s_T) \ni s_i \in S$, are primitives and states respectively observed. T is the number of time-steps observed in total in the trajectory.

During the maximization step, the algorithm will fit the prior probabilities $\rho(h)$, the observation probabilities $b_i(a \mid s)$, and the transition probabilities between options $a_{ij}(s)$.

To fit a better model, we need to know the marginal likelihoods (of the occurrence and co-occurrence of hidden options). However, to compute the marginal likelihoods, the model must be known. As with the standard Baum-Welch algorithm this problem is solved by iteratively computing the marginal likelihoods from the current model (**E-step**) and use them to fit a better model on the observation (**M-step**).

2.1.1 M-step

During the M-step we want to estimate the model parameters. Formally, we want to estimate $a_{ij}(s)$, $b_i(p \mid s)$, and $\rho(i)$ for every i , j , s , and p . Given that, $a_{ij}(s)$ is a composition of $\psi_i(s)$ and $\eta(j \mid s)$, to enforce the constrain in formula *2.1*, $\psi_i(s)$ and $\eta(j \mid s)$ can be instead estimated and $a_{ij}(s)$ computed from them.

By definition $\psi_i(s)$ is the probability of i stopping with state s . Therefore we can define it as

$$\begin{aligned}\psi_i(s) &= P(h_t \neq i \mid h_{t-1} = i, s_t = s, \zeta, \chi) \\ &= \frac{P(h_t \neq i, h_{t-1} = i, s_t = s \mid \zeta, \chi)}{P(h_{t-1} = i, s_t = s \mid \zeta, \chi)}\end{aligned} \tag{2.4}$$

at any time-step t . First we define

$$\gamma_t(i) = P(h_t = i \mid \zeta, \chi) \tag{2.5}$$

as the occurrence probability of a hidden option i at time t (given what we observed), and

$$\xi_t(i,j) = P(h_t = j, h_{t-1} = i \mid \zeta, \chi) \tag{2.6}$$

as the co-occurrence probability of hidden options i and j , being in succession at time-steps $t-1$ and t . These two probabilities represent the expected number of times we can find a single option or two consecutive ones (respectively) running at a certain point in time. Making them independent of the time-step would yield the likelihoods needed to compute the transition probabilities' update.

From formulas *2.5* and *2.6* we derive:

$$P(h_{t-1} = i, s_t = s \mid \zeta, \chi) = \sum_t \mathbb{I}[s_t = s]\gamma_{t-1}(i) \tag{2.7}$$

where $\mathbb{I}[]$ are the Iverson brackets. We are thus removing the temporal dimension by considering the disjunction (sum) of the probabilities of a hidden option i running at any point in time. However, only the time-steps in which the subsequent state is s are taken into account. We are thus looking for the expected number of occurrences of i in which $s_t = s$ also occurs.

Similarly, from the co-occurence probability we compute $P(h_t = j, h_{t-1} = i, s_t = s \mid \zeta, \chi)$ for any time-step t as follows:

$$P(h_t = j, h_{t-1} = i, s_t = s \mid \zeta, \chi) = \sum_t \mathbb{I}[s_t = s]\xi_t(i,j)$$

from which we compute the nominator of formula *2.4* as :

$$\begin{aligned}&P(h_t \neq i, h_{t-1} = i, s_t = s \mid \zeta, \chi) \\ &= \sum_j \mathbb{I}[j \neq i] P(h_t = j, h_{t-1} = i, s_t = s \mid \zeta, \chi) \\ &= \sum_j \mathbb{I}[j \neq i] \sum_t \mathbb{I}[s_t = s]\xi_t(i,j)\end{aligned} \tag{2.8}$$

thus the probability of an option changing is the sum of all the co-occurrence probabilities with a different option, in the state s .

Therefore formula *2.4* can be rewritten as

$$\begin{aligned}\psi_i(s) &= P(h_t \neq i \mid h_{t-1} = i, s_t = s, \zeta, \chi) \\ &= \frac{\sum_j \mathbb{I}[j \neq i] \sum_t \mathbb{I}[s_t = s]\xi_t(i,j)}{\sum_t \mathbb{I}[s_t = s]\gamma_{t-1}(i)}\end{aligned} \tag{2.9}$$

which constitutes the update rule for the stop condition matrix (given the occurrence and co-occurrence probabilities).

Similarly $\eta(j \mid s)$ can be expressed as the probability of switching to hidden option j after a different option was running before

$$\begin{aligned}\eta(j \mid s) &= P(h_t = j \mid h_{t-1} \neq j, s_t = s, \zeta, \chi) \\ &= \frac{P(h_t = j, h_{t-1} \neq j, s_t = s \mid \zeta, \chi)}{P(h_{t-1} \neq j, s_t = s \mid \zeta, \chi)}\end{aligned} \tag{2.10}$$

The nominator is similar to that of formula *2.4* with the difference that it is the expected number of transitions to j rather than from i . For the denominator we again consider the probability of something being different as the disjunction of the probability of that option at that point in time being an option different from j . Formally

$$\begin{aligned}\eta(j \mid s) &= P(h_t = j \mid h_{t-1} \neq j, s_t = s, \zeta, \chi) \\ &= \frac{\sum_i \mathbb{I}[j \neq i] \sum_t \mathbb{I}[s_t = s]\xi_t(i,j)}{\sum_i \mathbb{I}[i \neq j] \sum_t \mathbb{I}[s_t = s]\gamma_{t-1}(i)}\end{aligned} \tag{2.11}$$

A similar case can be made for the observation probability matrix, given that:

$$\begin{aligned}b_i(a \mid s) &= P(a_t = k \mid h_t = i, s_t = s \mid \zeta, \chi) \\ &= \frac{P(a_t = k, h_t = i, s_t = s \mid \zeta, \chi)}{P(h_t = i, s_t = s \mid \zeta, \chi)}\end{aligned} \tag{2.12}$$

where the same process used to derive formulas *2.7* and *2.8* is used to express formula *2.12* in terms of the marginal likelihoods γ and ξ . Formally:

$$b_i(a \mid s) = \frac{\sum_t \mathbb{I}[a_t = a \wedge s_t = s]\gamma_t(i)}{\sum_t \mathbb{I}[s_t = s]\gamma_t(i)} \tag{2.13}$$

Finally the prior probabilities don't need to be fit, as formula *2.3* expresses ρ in terms of η which is fit using *2.11*.

2.1.2 E-step

In the expectation step, the marginal likelihoods γ and ξ are computed. To do so, we assume that the model is given; we thus use the model's current fit to estimate them. Moreover, we can define them in terms of α and β recursion.

α and β recursions

$\alpha_t(i)$ is defined as the forward probability, which is the probability of observing all primitives up to time-step t and hidden option i being performed at time t . Formally:

$$\alpha_t(i) = P(p_0, ..., p_t, h_t = i \mid \chi) \tag{2.14}$$

While $\beta_t(i)$ is the backward recursion which expresses the probability of observing the sequence of primitives from time $t+1$ until the end of the imitation trajectory (T), given that hidden option i was performed at time-step t . That is:

$$\beta_t(i) = P(p_{t+1}, ..., p_T \mid h_t = i, \chi) \tag{2.15}$$

Knowing $\alpha_t(i)$ for every option i we can compute $\alpha_{t+1}(i)$ as follows. By definition $\alpha_{t+1}(i) = P(p_0, ..., p_{t+1}, h_{t+1} = i \mid \chi)$, according to the Bayesian rule this is equivalent to:

$$P(p_{t+1} \mid h_{t+1} = i, \chi)P(h_{t+1} = i, p_0, ..., p_t \mid \chi)$$

Since the observation at time-step $t+1$ depends solely on h_{t+1} , we have that:

$$\alpha_{t+1}(i) = b_i(p_{t+1} \mid s_{t+1})P(h_{t+1} = i, \zeta_{0:t} \mid \chi) \tag{2.16}$$

Since options are mutually exclusive, we can further rewrite $P(h_{t+1} = i, p_0, ..., p_t \mid \chi)$ as:

$$\sum_j P(h_{t+1} = i, h_t = j, \zeta_{0:t} \mid \chi)$$

which using Bayes again is equivalent to:

$$\sum_j P(h_{t+1} = i \mid h_t = j, \chi)P(\zeta_{0:t}, h_t = j \mid \chi)$$

which using formulas *2.1* and *2.14* can be rewritten as $\sum_j a_{ji}(s_{t+1})\alpha_t(j)$. This can be substituted in formula *2.16* to have the recursive definition:

$$\alpha_{t+1}(i) = b_i(p_{t+1} \mid s_{t+1}) \sum_j a_{ji}(s_{t+1})\alpha_t(j) \tag{2.17}$$

therefore for each time-step the α recursion keeps track of all possible paths from a preceding hidden option to the current one (i), weighted by the probability that such an option generated the observation at that particular time-step.

The initial values $\alpha_0(i) = P(p_0, h_0 = i \mid \chi)$ is the conjucntion (product) of the prior probability $P(h_0 = i \mid \chi) = \rho(i)$ and the probability of hidden option i generating the observation at time-step 0 (thus $\pi_i(p_0 \mid \chi_0)$).

β_t can be also recursively defined in terms of β_{t+1} . By the definition given in formula *2.15* it holds that:

$$\beta_t(i) = \sum_j P(\zeta_{t+1:T}, h_{t+1} = j \mid h_t = i, \chi)$$
$$= \sum_j b_j(p_{t+1} \mid s_{t+1})P(\zeta_{t+2:T}, h_{t+1} = j \mid h_t = i, \chi) \tag{2.18}$$

by using the Bayesian rule and given that the probability of observing p_{t+1} is independent of the previous observations. Using Bayes a second time $P(\zeta_{t+2:T}, h_{t+1} = j \mid h_t = i, \chi)$ can be rewritten as the product of $P(\zeta_{t+2:T} \mid h_{t+1} = j, h_t = i, \chi)$ and $P(h_{t+1} = j \mid h_t = i, \chi)$. Given that the probability of observing $\zeta_{t+2:T}$ is independent from h_t (as it follows the Markov property) we have that:

$$P(\zeta_{t+2:T}, h_{t+1} = j \mid h_t = i, \chi) = a_{ij}(s_{t+1})\beta_{t+1}(j) \tag{2.19}$$

which can be substituted in formula *2.18*, finally obtaining the recursive definition:

$$\beta_t(i) = \sum_j b_j(p_{t+1} \mid s_{t+1}) a_{ij}(s_{t+1})\beta_{t+1}(j) \tag{2.20}$$

The initial values $\beta_T(i) = P(p_{T+1} \mid h_T = i, \chi)$ is set to 1 as p_{T+1} is unknown.

Occurence and Co-Occurence likelihoods

ξ can be expressed as a function of the α and β recursions (and the model itself). Formally, according to formula *2.6* and applying Bayes we have:

$$\xi_t(i,j) = c \cdot P(\zeta \mid h_t = j, h_{t-1} = i, \chi) P(h_t = j, h_{t-1} = i \mid \chi)$$

where $c^{-1} = P(\zeta \mid \chi) = \sum_j \alpha_t(j)\beta_t(j)$, since $\alpha_t(j)\beta_t(j) = P(\zeta, h_t = i \mid \chi)$.

The probability of observing the trajectory of primitives ζ is split in the conjunction (product) of the probabilities of: observing the primitives from time-step 0 to time-step $t-1$ (independent from $h_t = j$), then the primitive at time-step t , and finally from $t+1$ up to T . Since the last two are independent from $h_{t-1} = i$, it can be observed that they are respectively equal to $b_j(p_t \mid s_t)$ and $\beta_t(j)$. Using Bayes, $P(h_t = j, h_{t-1} = i \mid \chi)$ can be split in $P(h_t = j \mid h_{t-1} = i, \chi)P(h_{t-1} = i \mid \chi)$, where $P(h_t = j \mid h_{t-1} = i, \chi) = a_{ij}(s_t)$. Multiplying $P(h_{t-1} = i \mid \chi)$ with the previously mentioned $P(\zeta_{0:t-1} \mid h_{t-1} = i, \chi)$ yields $\alpha_{t-1}(i)$. Applying all these equivalences to the previous definition of $\xi_t(i,j)$ results in:

$$\xi_t(i,j) = c \cdot \alpha_{t-1}(i) b_j(p_t \mid s_t)\beta_t(j) a_{ij}(s_t) \tag{2.21}$$

It is noticeable that the term c is cancelled out in formulas *2.9*, *2.11*, and *2.13*

γ can be instead computed from ξ . Since in formulas *2.9* and *2.11* the term γ_{t-1} is used, it is easier then to express $\gamma_{t-1}(i) = P(h_{t-1} = i \mid \zeta, \chi) = \sum_j P(h_t = j, h_{t-1} = i \mid \zeta, \chi) = \sum_j \xi_t(i,j)$

2.2 Online learning

Once the options are learned, the agent can keep improving it's main policy η as it acts based on the rewards the agent receives. To do so the Q values can be found so that the resulting policy would be η . We need to define how to compute a policy (such as η) from its Q_η function. Several methods (Tijsma, Drugan, and Wiering, 2016) can be used that try to balance exploration (gaining new experience) and exploitation (acting in a smart manner). In this paper, we use *Softmax exploration*, that is

$$\eta(h \mid s) = \frac{e^{Q_\eta(s,h)/T}}{\sum_{h' \in H} e^{Q_\eta(s,h')/T}} \tag{2.22}$$

where T is the temperature with determines how skewed the probability distribution is. The reason why we select this exploration policy is because, first, it generates a probability distribution with non-zero probabilities, which makes it more easily invertible as all Q values influence all action's probabilities. Moreover, with a low enough temperature we have that $\eta(\underset{h'}{\text{argmax}}\, Q_\eta(s,h') \mid s) >> 0$ while $\eta(h \mid s) << 1 \ni h \neq \underset{h'}{\text{argmax}}\, Q_\eta(s,h')$. In other words, with a low temperature the policy can still be greedy, since the imitation data is assumed to be coming from a (close to) optimal strategy, not much exploration is required and thus a more greedy policy can be less noisy. One disadvantage is that there are infinite solutions (infinite sets of values that result in the same probability distribution). Thus when inverting the softmax function (going from the policy fitted on the imitation data to the Q values), we need to take into account the scale of rewards in the current task. The inverse of the softmax function is in fact

$$Q_\eta(s,h) = ln(\eta(h \mid s)) + C_1 \tag{2.23}$$

with C_1 being the unknown quantity $ln(\sum_{h' \in H} e^{Q_\eta(s,h')/T})$. C_1 can be thus set to account for the reward range in the current task.

Finally, the computed Q_η function will return η once softmax is applied to it, with the advantage that it can be updated on-line using standard Q-learning (Watkins and Dayan, 1992). From formula *1.1* and *1.2* it is noticeable that given a

transition $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle \ni s_t, s_{t+1} \in S, \quad a_t \in A, \quad r_t = R(s_t, a_t), \quad T(s_{t+1} \mid s_t, a_t) \neq 0$, the Q_η function can be approximated as

$$Q_\eta(s_t, a_t) \overset{\alpha}{\leftarrow} r_t + \gamma U_\eta(s_{t+1}) \tag{2.24}$$

where $\alpha \in [0, 1]$ is the learning rate. Given that the starting policy is fit on a given trajectory assuming to be a (close to) optimal one, a simple off-policy method can be used to update the Q_η , such as standard Q-learning (Watkins and Dayan, 1992), thus $U_\eta(s) = \max_a Q_\eta(s, a)$. This because the assumption of off-policy methods that the agent is acting optimally is possibly met in imitation learning.

The options are not trained while acting, and thus their policies (and stop conditions) remain unchanged. This also removes the need to compute the respective Q function of the options' policies.

2.3 Base Line Algorithm

The baseline algorithm is a simple single policy agent. To have a fair comparison between the option based and the baseline agents, the latter will be also trained on the imitation data. To do so the given trajectory will be composed not only of $\zeta_{0:T} = (p_0, ..., p_T) \ni p_i \in A$ and $\chi_{0:T} = (s_0, ..., s_T) \ni s_i \in S$, but also of $\phi_{0:T} = (R(\chi_0, \zeta_0), ..., R(\chi_T, \zeta_T))$.

To train the agent on the imitation data we use an on-policy Q-Learning method, that is SARSA (Rummery and Niranjan, 1994). This method has the same updated rule for the Q_π function (formula *2.24*) but $U_{pi}(s_{t+1}) = Q(s_{t+1}, a_{t+1})$. This takes into account the current policy which, in this case, is the optimal one observed from the imitation data, since transitions are extracted from the given trajectory. With SARSA a transition is defined as a tuple $e_t = \langle s_t, a_t, r_t, s_{t+1}, a_{t+1} \rangle \ni s_t, s_{t+1} \in S, \quad a_t, a_{t+1} \in A, \quad r_t = R(s_t, a_t), \quad T(s_{t+1} \mid s_t, a_t) \neq 0$. Given the imitation data $\langle \zeta, \chi, \phi \rangle$ then $e_t = \langle \chi_t, \zeta_t, \phi_t, \chi_{t+1}, \zeta_{t+1} \rangle$.

Experience replay (Lin, 1992) is used to train the model for multiple iterations over the given transitions e_t . This is done to remove some temporal dependencies in the presented trajectory that may be specific of the tasks solved in the imitation data. While the proposed architecture is able to factorize the data into smaller trajectories, by assuming that a series of options are generating sub-trajectories of primitives, the baseline algorithm has not such ability. If the trajectory can be factorized into options, such options (and thus sub-trajectories) could be performed in different order for similar tasks. Thus we decided to have the baseline algorithm get rid of these chronological dependencies overall, effectively factorizing the imitation trajectories in single transitions, so to give to the baseline algorithm a better chance to adapt to new unseen tasks, as the proposed algorithm is expected to do as well.

3 Results

In this section we will introduce the setup of the experiment used to test the proposed algorithm, together with the results collected from it.

3.1 Experimental Setup

To test the proposed architecture we implemented a driving simulation. The agent is represented by a circle (25px radius). The agent can accelerate forward or backward and turn left or right while accelerating (or not turn and thus go straight). The car cannot rotate on its place instead it can only turn in arcs (much like real cars).

A 500×500px window is then divided in 100×100px tiles (forming a 5×5 grid). A randomly closed loop of tiles is selected (where two consecutive tiles in the loop are either horizontally or vertically aligned). These tiles are delimited by walls and therefore this loop is the track in which the car will have to drive in. If the agent touches a wall it bounces back in the opposite direction (and it cannot go through any wall). An example of such a setup is show in figure 3.1.

The car position in the environment is continuous. The agent is positioned in an arbitrary starting tile and is oriented towards the next tile in the loop. The goal is to move forward in the same direction (clockwise or anti-clockwise arbitrarily chosen) in the track. This goal is encoded by giving the agent a reward of -10 for bumping in a wall, -100 for moving from a tile to the one preceding it (according to the chosen direction of the loop), 100 for moving from a tile to the one following it, -1 otherwise. The agent is thus rewarded for going forward, and punished for going backward, bumping, or staying still. We gave more importance to

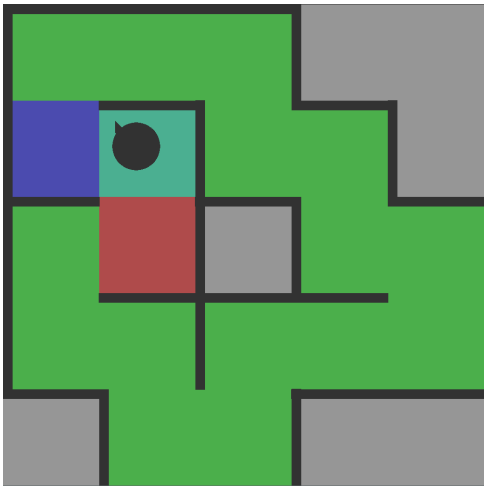

Figure 3.1: *An example of the environment setup used to test the algorithm. The black circle is the agent. The path the agent can go through is colored in* **green**. *The tile following the one the agent is currently on is colored in* **blue**, *while the one preceding it in* **red**. **gray** *tiles can't be crossed by the agent (outside of the track).*

not going backward rather than not bumping (since the punishment is ten times bigger) because we preferred the agent to go forward even if occasionally bumping, instead of not bumping but going backward sometimes.

The agent's state is given by a series of 6 distance sensors, each $60°$ with respect to the car's orientation. The simulated distance sensor measures the distance from the center of the car to the closest wall in its direction. The sensor can only have one of 3 values. If the distance is less than 100px then the distance is considered short, if it is between 100px and 200px pixel then it's medium, if it is greater than 200px then it is a large distance. With 6 distance sensors (and 3 values per sensor) there are $3^6 = 729$ different states. These are enumerated and the corresponding value is the agent's state.

The simulation is a separated thread from the running A.I. The car keeps performing the last action the A.I. communicated while the A.I. computes the next action to perform. The environment will communicate the state to the A.I. when the agent requests it. The A.I. can also request the environment to communicate its reward, in which case the accumulated reward since the last request is sent to the agent.

3.1.1 Parameters

Both the baseline and the tested agents use a learning rate α of 0.75 when learning online, while the baseline agent has been pre-trained on the imitation data with a learning rate of 0.5 . Both agents use a discount factor of 0.99 (with the baseline agent using the same value during pre-training as well). Both softmax functions were used with a temperature value of 10 which proved to be the lowest value resulting in the best performance (on average). The Q values for the main policy of the option based agents were computed using a value of 50 for C_1 in formula *2.23*.

During pre-training of the tested architecture, the imitation data was fitted using 10 hidden options. This value was selected as the agent in the environment described above can perform 8 different turns (as each curve can be performed clockwise or anti-clockwise) And can also move on a straight line forward or backward. The baseline agent was trained on imitation data containing $77,000$ transitions. Stochastic experience replay was performed for $10,000$ iterations on the baseline agent. The options were trained with a trajectory of $1,500$ time-steps ($\approx 2\%$ of the data used for the baseline). 100 iterations of the modified Baum-Welch algorithm (each composed of an M-step and an E-step) were performed.

3.2 Experimental Results

Both the baseline and the proposed algorithm were left to run for 500 time-steps in 100 randomly generated tracks each. The tracks were consecutive, meaning that both agents would keep learning between tracks.

The number of tiles the agent managed to move forward was measured. This was done by adding 1 whenever the agent would move to the subsequent tile (from its current one), -1 when the agent would move to the tile preceding its current one, and 0 otherwise. The results can be observed in figure 3.2

It can be seen that both agents move forward at a linear rate (constant velocity), with the option based agent having a speed which is $\approx 28.57\%$ of that of the baseline agent.

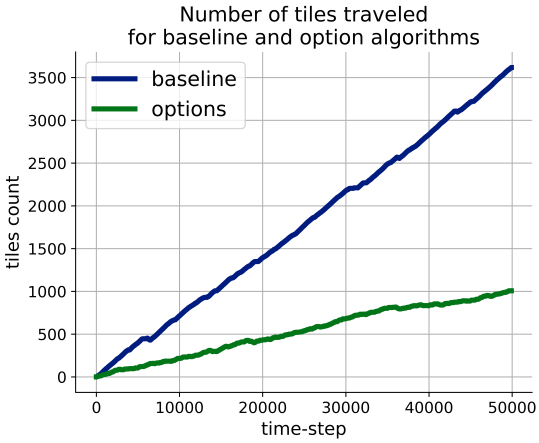

Figure 3.2: *Number of tiles that each agent traveled (in the forward direction) with respect to each time-step, for both the baseline and option based agent.*

4 Discussion

The results in figure 3.2 show that the agent was able to fit a set of suitable options to complete the task, as it was able to move forward in the given environment from the start. However, the data also shows that the baseline algorithm has superior performance than the tested one.

Looking at the experimental runs the reason for the performance disparity becomes clear. While both agents can move in the path in a forward direction from the start (once pre-trained on the imitation data), the option based agent is afflicted by an architectural problem that makes it engage in undesired behavior.

To be more precise, as specified above, the option based agent will select an option to run whose policy will decide which primitive actions to perform given a state. The selected option will be in control until a stochastic stop-condition is met. By keeping track of the option in control of the agent at each time-step, it seems evident that the same option(s) get control when performing the same turn (in the same direction). However, given the stochastic nature of the stop condition, the aforementioned option(s) may stop prematurely or later than desired. This leads to the agent performing a partial turn or turning too much and thus bumping in a wall. Moreover, the main policy η , while having a high temperature (relative to the Q values) for its softmax function, is still stochastic in the selection of the option. Thus an option performing the wrong turn in the wrong direction may be given control. Both cases make the turn of the agent a noisy process and thus prone to errors. Some of these errors lead the agent to bump in the corner of a turn. In such cases, the agent is in a state in which the distance sensor hold little information to what is the forward direction of the curve. This is coupled with the fact that the agent has no memory of past states or actions, and therefore has no way of knowing from which direction it came. Since each curve may be required to be driven clockwise or anti-clockwise, when the option based agent is in the situation mentioned above, it may go backward in the path (and do so at the same speed it was going forward). When this happens, the agent receives a series of severe punishments that do not give an incentive to the agent to turn in the opposite direction (as that would require planning), but instead, the agent learns to stay still when it is found against a corner.

Another problem that contributed to the slower speed of the option based agent is the conflict in the option selection. This problem is also caused by the stochasticity of the selection and stopping of an option. If an option terminates early, it may need to be called more times (which increases the chances of selecting the wrong option). If the wrong option is selected, then its stop condition will terminate it after a few time-steps. This, however, slows down the agent which is more indecisive regarding which actions to take to move forward.

Both these problems are some of the cons of the exploratory advantage of options mentioned above. The environment (and the task) do not reward exploration given that the imitation data has already a close to optimum solution.

However, one noticeable feature is that the option based agent was trained on $\approx 2\%$ of the data used for the baseline. This is because the overall likelihood of the data (given the model) as well as the performance, was not much improved by adding more data. As previously explained, the main pitfalls of the agent come from the option architecture itself, and not their training. Moreover, the agent was trained on simple short tracks with at most two examples for each type of curve the agent could be performing. During testing, however, there were no such constraints. The agent was still able to move forward in more convoluted (more turns)

and longer tracks. Most importantly, the agent was able to run over consecutive curves in combinations not observed in the imitation data. This show the ability of the algorithm to use the imitation data to solve a more complex problem than the one shown to it in the imitation data.

5 Conclusion

The algorithm resulted in a worse performance than the single policy agent. However, the algorithm seems to fit working options on little data and training. Moreover, it seems to be able to adapt quickly to new environments not presented before in the imitation data.

Since most of the performance issues seem to derive from the option architecture itself, improvements may be made to the algorithm so to use a more suitable hierarchical architecture (Hengst, 2011). When explaining the results, the option architecture and its stochastic seem to justify the reason behind the inferior performance of the proposed architecture. However, merely making the stop condition deterministic or removing the option framework itself may only accentuate the issue. The latter would result in a less flexible agent, while the former in a more noisy behavior (as at each timestep a controller is selected, and thus there are higher chances that a wrong one is selected multiple times).

Future research should be focused on the transfer learning (Pratt, 1993) capabilities of the architecture, which seem to be promising given the architecture's ability to adapt to unseen track formation, using options trained on a small data sample (where options' policies are fit only during pre-trained and remain static during the acting phase). Moreover, a training procedure for the options online should also be a topic of interest in the future, to make the agent more flexible when acting in new tasks, as it cannot now tweak its options as needed.

In the paper from Fox et al. (2017), online learning seems to play a more significant role in the algorithm's performance. More precisely, during the online learning phase, the architecture improves its performance over the baseline algorithm. In this paper, however, the pretraining part seems to weight more on the agent's performance, both for the baseline and the tested algorithm. This was expected, however, as Fox et al. (2017) don't have static options. In their algorithm, the agent is mainly trained on introspective data, that is, it produced a trajectory to imitate (running for an arbitrary number of time-steps) and then fits the options on its trajectory. The advantage of the architecture proposed in this paper is that assuming that the option, states, and actions are discrete, it presents a tailored algorithm for discrete (or discretized) Markov Decision Process. This is reflected by the greater use that the architecture proposed in this paper makes of the pre-train imitation data, with respect to the algorithm proposed by Fox et al. (2017).

Overall the algorithm seems to learn options as expected; however, the architecture exploiting the learned procedural knowledge presents some setbacks in the task at hand.

References

R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-Level Discovery of Deep Options. *arXiv e-prints*, art. arXiv:1703.08294, Mar 2017.

B. Hengst. *Hierarchical Approaches*. 2011. doi: 10.1007/978-3-642-27645-3_9.

F. Jelinek, L. Bahl, and R. Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, 21(3):250–256, May 1975. ISSN 0018-9448. doi: 10.1109/TIT.1975.1055384.

L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699. URL `https://doi.org/10.1007/BF00992699`.

L.Y. Pratt. Discriminability-based transfer between neural networks. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 204–211, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-274-7.

D. Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, 2000. AAI9978540.

A.G. Rummery and M. Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

M. Stolle and D. Precup. Learning options in reinforcement learning. In Sven Koenig and Robert C. Holte, editors, *Abstraction, Reformulation, and Approximation*, pages 212–223, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45622-3.

R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181 – 211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1.

A. D. Tijsma, M. M. Drugan, and M. A. Wiering. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Dec 2016. doi: 10.1109/SSCI.2016.7849366.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698.

M. Wiering and M. van Otterlo. *Reinforcement Learning: State-Of-The-Art*, volume 12. 2012. doi: 10.1007/978-3-642-27645-3.