TECHNISCHE UNIVERSITÄT WIEN

Bachelorarbeit

Monte Carlo Policy Gradients in Discrete Event Simulations

ausgeführt zum Zwecke der Erlangung des akademischen Grades
BSc Technische Mathematik

von

Markus Peschl

MatNr: 01609416

unter der Anleitung von

Assoz. Prof. Dipl.-Ing. Dr. techn. Clemens Heitzinger

26. Juli 2019

Contents

1 Introduction

With recent advances in computing power, the field of machine learning has become increasingly popular, mainly because of its profitability and wide area of applications in industry as well as in research. While supervised learning techniques have been able to solve challenging problems in computer vision, natural language processing and related fields, these algorithms usually only deal with regression or classification tasks. However, when it comes down to programming an agent which is supposed to take actions in an environment and compare their outcomes over a set of possible actions, a new learning paradigm is needed. This is where reinforcement learning comes into play. By defining a set of possible states, actions and rewards, algorithms in this field can learn policies which maximize the rewards received in the long run. For this reason, the theory of reinforcement learning yields especially promising applications in automation.

As with all machine learning methods, an enormous amount of data is often needed during training in order to achieve a low generalization error. The data gathering process can sometimes be difficult in reinforcement learning, since one has to gather data by trying out different actions in an environment and then observe their respective outcomes, which might (depending on the environment) take especially long. This is one of many reasons why Atari 2600 games have become a standard task in the reinforcement learning community for benchmarking various algorithms. In this thesis, we will follow a similar, but more application driven approach, by letting an agent act in a discrete event simulation of a modern factory. We will apply tabular methods as well as approximate methods of reinforcement learning to a management task in factories of various complexities and analyse the effect of different value function approximators on the learning speed and convergence of policy gradient methods. Finally, we will compare natural policy gradients to the traditional (vanilla) approach to policy gradient methods.

2 Methods

2.1 Discrete Event Simulation

At its core, discrete event simulation (DES) is a tool for modeling queuing systems. Each element of the simulation is represented by a process which is either in a timed out state (i.e. waiting for other processes to time out) or currently performing an action on the environment. This forms an ongoing queue of processes, which gets processed at discrete time steps. At first glance, a queuing system might not seem like a versatile modeling tool. However, considering that the time steps can be made arbitrarily small, many problems which entail automation of a procedure can effectively be modeled by a discrete queue and therefore DES seems suitable for reinforcement learning tasks.

The task we will want our agent to learn can essentially be stated as follows: Given a set of n machines, which (linearly) work together in order to build a product, what is the best way of distributing additional resources (these could be spare machines, human workforce or just general monetary expenses) among various stages of production? A sketch of the experimental setup is shown in figure 1. In order to model this environment, we use a discrete event simulation framework, called `SimJulia`, which is a Julia implementation of `SimPy`. At each time step of the simulation k unfinished products are introduced to the assembly line and stored at the first machine. Per time step, each machine M_i moves at most s_i products from its own stash to the stash of M_{i+1}. Furthermore, each machine needs to be repaired every X_i time steps, where $X_i \sim \exp(\lambda_i)$. During an active simulation, an agent gets consulted every 5 time steps and is asked to allocate one of r spare machines to a specific point $p \in \{0, \ldots, n\}$ on the assembly line, where $p = 0$ would mean that the extra resource is not being used. For each spare machine being connected to a point, the amount of objects being processed and sent to the next station is increased by a fixed amount j. While this task could be viewed both as an episodic task (i.e. finishing products in a limited amount of time) or as a continuing task, we will constrain ourselves to the episodic case. Note that given an optimal policy for the episodic case, one might expect that this policy will perform equally well in the continuing task, since the goal of maximizing the output of the factory stays the same.

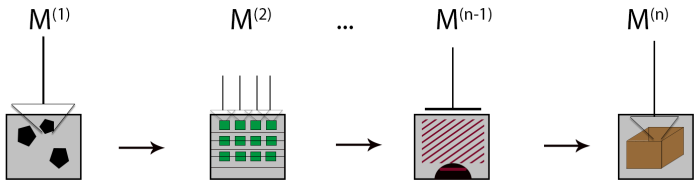

Figure 1: A linear factory consisting of n machines. Failure of one intermediate machine will cause the production to slow down.

2.2 Reinforcement Learning

Most algorithms in reinforcement learning, as well as the analysis of their convergence, assume that the interaction between agent and environment obeys the rules of a Markov Decision Process (MDP). Formally, a finite MDP consists of a set of states and actions $\mathcal{S}$ and $\mathcal{A}$ (where $|\mathcal{S}| < \infty,\ |\mathcal{A}| < \infty$) as well as transition probabilities

$$p(s', r|s, a) := \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a), \tag{2.2.1}$$

where S_t, A_t and R_t denote the state, action and received reward at a given time step t. Generally, the first state S_0 of the MDP is chosen according to a starting state distribution. Furthermore, the transition probabilities p completely characterize the dynamics of the environment, which assures that our stochastic process fulfills the Markov property. In our simulation, we will consider episodic tasks only. This means that the simulation reaches a terminal state after a certain amount of time and the goal of our agent is to maximize rewards before the end of the simulation. Alternatively, one could formulate a continuing task, which would slightly change the definitions below.

Having defined the MDP, formalizing the goal of the agent reduces to the maximization of an expectation. Let

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \tag{2.2.2}$$

denote the return at time t, where T is the final time step and $0 \leq \gamma \leq 1$ is a discount factor. With this definition, we can define a value function

$$v_\pi(s) := \mathbb{E}_\pi[G_t|S_t = s], \tag{2.2.3}$$

which measures the value of a state, given that the agent follows a fixed policy π.
Since only learning a value function does not always help in decision making tasks (i.e. a model is needed), the common goal in reinforcement learning is to learn an action value function, which is similarly defined by

$$q_\pi(s, a) := \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{2.2.4}$$

From now on, we will often deal with procedures for estimating the optimal action value function

$$q_*(s, a) := \max_\pi q_\pi(s, a) \tag{2.2.5}$$

in an episodic management task, as described in section 2.1. Theoretically, if the dynamics of the MDP were known, one could obtain an exact solution by solving the Bellman optimality equation

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')], \end{aligned} \tag{2.2.6}$$

which has a unique solution in the case of finite MDPs. For methods of this kind, see [1, Chapter 4]. In practice however, these methods are unfeasible, since the dynamics of the environment are not known.

2.3 Tabular Methods

2.3.1 Monte Carlo

One simple, yet powerful approach to solving problems where the dynamics of the MDP are not known are Monte Carlo Methods. The basic idea behind these methods is that, given enough time, an agent can abundantly sample experience by acting in its environment and then determine an approximate distribution of the returns G_t obtained in a state s, when action a was taken. (Note that the expected value of this distribution is exactly the value function (2.2.5), which we want the agent to learn.) If the action-state-space is not too large, one can explicitly save the observed returns in an array and obtain (unbiased) sample means for each state-action pair. One obvious drawback of this approach is the need to complete a whole episode before any learning can be made, which might result in slow learning. For example, if the agent made a critical mistake in our factory task, he would have to wait until the end of the episode to learn from that experience. However, it could have already been apparent during the episode that a certain move was bad, by observing that the output of the factory drastically slowed down over the course of time. For a more detailed comparison, see section 3.

On-Policy Control

By means of generalized policy iteration (GPI) we can apply Monte Carlo methods to control problems. That is, in each step we maintain approximate policies π_k and approximate value functions q_{π_k} and update the policy w.r.t. the greedy actions

$$\pi_{k+1}(A|S) := \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(S)|}, & A = \arg\max_{a \in \mathcal{A}(S)} q_{\pi_k}(S, a), \\ \frac{\epsilon}{|\mathcal{A}(S)|}, & \text{otherwise}, \end{cases} \tag{2.3.1}$$

where the approximation for $q_{\pi_k}(S, A)$ is made from the sample mean of returns previously obtained in a state-action pair (S, A). If the q-value approximations were exact, then one could obtain convergence via the policy improvement theorem. However, since this cannot be done in practice, convergence is not easy to prove. According to [1], this is still an open problem in mathematics.

Off-Policy Control

A problem that all on-policy methods share is to keep an optimal balance between exploration and exploitation. An agent behaving deterministically cannot find out previously unseen strategies, but only evaluate the efficacy of its current strategy. Therefore, the agent is forced to behave suboptimally from time to time in order to keep all possible strategies in mind. The idea of acting according to a random policy $b(a|s)$, called behavior policy, while updating a target policy is called off-policy. The difficulty in off-policy learning is due to the difference of two distributions: While the data is drawn from the distribution of the behavior policy, we actually need to update the target policy as if the data was drawn according to its state-action distribution. One way to solve this is done via importance sampling, which can essentially be stated as

$$\mathbb{E}_f[X] = \int_\Omega x f(x) dx = \int_\Omega x \frac{f(x)}{g(x)} g(x) dx = \mathbb{E}_g \left[X \frac{f(X)}{g(X)} \right] \tag{2.3.2}$$

for some probability densities f and $g \neq 0$ (almost surely). This leads to the unbiased (and consistent) importance sampling estimator

$$\mathbb{E}_f[X] \approx \frac{1}{n} \sum_{i=1}^{n} x_i \frac{f(x_i)}{g(x_i)}, \tag{2.3.3}$$

where the x_i are drawn according to g. Applying this idea to reinforcement learning tasks, we would like to estimate $q_\pi(s, a)$ for all states s and actions a. If M denotes the number of first visits to a given pair (s, a) during training and t_m the time at which these pairs occurred in their respective episodes, then

$$Q(s, a) := \frac{1}{M} \sum_{m=1}^{M} G_m w_m, \tag{2.3.4}$$

where w_m is the importance sampling ratio

$$\frac{\pi(S_{t_m+1}|A_{t_m+1})}{b(S_{t_m+1}|A_{t_m+1})} \frac{\pi(S_{t_m+2}|A_{t_m+2})}{b(S_{t_m+2}|A_{t_m+2})} \cdots \frac{\pi(S_{T_m-1}|A_{T_m-1})}{b(S_{T_m-1}|A_{T_m-1})}, \tag{2.3.5}$$

enables us to estimate the action values off-policy. An alternative importance sampler, called the weighted importance sampling estimator is defined by a weighted average

$$Q(s, a) := \frac{\frac{1}{M} \sum_{m=1}^{M} G_m w_m}{\sum_{m=1}^{M} w_m} \tag{2.3.6}$$

and has been shown to produce more accurate estimates (see [1, Section 5.5]). Ultimately, there are many ways to form estimators which are consistent. For a comprehensive comparison, see [2].

2.3.2 Q-Learning

Besides Monte Carlo methods, which only learn once an episode has finished, temporal difference (TD) methods, such as Q-learning, enable the agent to update the action-value function at each time step being made. For the reasons mentioned above, this seems like a reasonable approach to improve learning speed. Furthermore, some neuroscientific experiments suggest that dopaminergic projections in the human brain drive behavior by constant comparison of future rewards and current expectation, in the same way that TD methods update the policy at each time step (see [3]). Although the update rule

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t(R_t + \gamma \max_{a \in \mathcal{A}(S_{t+1})} Q_t(S_{t+1}, a) - Q_t(S_t, A_t)) \tag{2.3.7}$$

introduces a bias to the estimation of the action value function (which can sometimes overestimate the true value), it can be shown that under appropriate step size requirements and correct sampling of the whole state-action space the algorithm converges to the optimal action value function [4].

2.4 Policy Gradients

When dealing with large discrete state spaces (or continuous ones), tabular methods become unfeasible as the memory required as well as the training samples needed grow linearly with the size of the state-action space. While the idea of learning an action-value function can be extended to the nontabular case by using function approximators, a different, yet recently successful approach are policy gradient methods. These directly learn a parametrized policy through gradient based optimization. Let $\theta \in \mathbb{R}^n$ be the policy parameter and $\pi(a|s,\theta)$ denote the probability of selecting an action a in state s, given θ. In the episodic case the performance measure that we want to maximize is

$$J(\theta) := v_{\pi_\theta}(s_0), \tag{2.4.1}$$

where s_0 denotes a starting state of the episode. The main challenge of policy gradient methods is to obtain a good estimate of the gradient $\nabla J(\theta)$, which needs to be estimated merely using the data obtained from the environment, when no model is used.
One of the most simple approaches are finite-difference methods. By varying the parameter θ by small increments $\Delta\theta_i$ ($i \in \{1,\dots,m\}$), one can obtain estimate changes in performance $\Delta\hat{J}_i \approx J(\theta+\Delta\theta_i) - J(\theta)$ through rollouts using the policy $\pi(\cdot|\cdot,\theta+\Delta\theta_i)$. Then a linear regression estimator

$$\nabla\hat{J}(\theta) = (\Delta\Theta^T\Delta\Theta)^{-1}\Delta\Theta^T\Delta\hat{J}, \tag{2.4.2}$$

where $\Delta\Theta = (\Delta\theta_1,\dots,\Delta\theta_m)^T$ and $\Delta\hat{J} = (\Delta\hat{J}_1,\dots,\Delta\hat{J}_m)^T$ can be used to compute the approximate derivative. According to [5], approaches of this kind have been successfully used in various robotics tasks and can be efficient for simulation optimization of deterministic systems. One obvious drawback of this procedure, however, is the need for choosing appropriate step sizes $\Delta\theta_i$ as well as the growing computational complexity for a highly dimensional parameter vector θ. A more mathematical approach would be to derive an analytic expression of the gradient. The difficulty here is that changing the parameter affects not only the actions being made during an episode, but also the distribution of how often certain states are being visited. Since the dynamics of the underlying MDP are generally unknown, one cannot compute how the state distribution changes when varying θ. Nonetheless, the policy gradient theorem, as proven in [6], provides an analytic expression of the performance gradient which can be easily estimated using samples obtained during training.

Theorem 2.1 (Policy Gradient Theorem)**.** *Let $J(\theta) := v_{\pi_\theta}(s_0)$. Then, for any MDP*

$$\nabla J(\theta) \propto \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} \nabla\pi(a|s,\theta) q_{\pi_\theta}(s,a), \tag{2.4.3}$$

where $\mu(s)$ is defined as the on-policy state distribution for a start state distribution $p_{s_0}(\cdot)$, i.e.

$$\mu(s) := \frac{\eta(s)}{\sum_{s'\in\mathcal{S}}\eta(s')}, \quad \eta(s) = p_{s_0}(s) + \sum_{s'} \eta(s') \sum_a \pi(a|s',\theta) p(s|s',a). \tag{2.4.4}$$

Proof. See the appendix. □

2.4.1 Vanilla Policy Gradients

Since the policy gradient theorem guarantees that the gradient of the performance measure $J(\theta)$ can be written as an expected value with respect to the trajectory generated when following the fixed policy $\pi(\cdot|\cdot,\theta)$, it seems feasible to form updates based on stochastic gradient descent. A short calculation yields

$$\begin{aligned}\nabla J(\theta) &\propto \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a)\nabla\pi(a|S_t,\theta)\right]\\ &= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a)\pi(a|S_t,\theta)\nabla\log\pi(a|S_t,\theta)\right]\\ &= \mathbb{E}_\pi\left[q_\pi(S_t,A_t)\nabla\log\pi(A_t|S_t,\theta)\right]\\ &= \mathbb{E}_\pi\left[G_t\nabla\log\pi(A_t|S_t,\theta)\right],\end{aligned} \tag{2.4.5}$$

where the last equality follows from $q_\pi(S_t,A_t)=\mathbb{E}_\pi[G_t|S_t,A_t]$. This allows for a stochastic gradient descent algorithm to be implemented, where the update rule reads as

$$\theta_{t+1}=\theta_t+\alpha_t\nabla\hat{J}(\theta_t)=\theta_t+\alpha_t G_t\nabla\log\pi(A_t|S_t,\theta_t). \tag{2.4.6}$$

Since the returns need to be known in order to sample from the random variable inside the expectation, a sample is only complete, once the episode has ended. Therefore, this procedure could be called a Monte Carlo method. One of the biggest disadvantages of this method is that it can suffer from very high variance, especially when episode lengths grow very large. Besides that, unlike Q-learning, one can generally only guarantee convergence to a stationary point of the objective function J. For a discrete setting, however, [7] argues that under additional assumptions global convergence could be guaranteed. Several approaches have been proposed to overcome these difficulties, one of which is adding a state-dependent baseline $b(s)$ to the returns before updating the parameter. Note that this does not change any of the convergence results for our original algorithm since

$$\sum_a b(s)\nabla\pi(a|s,\theta)=b(s)\nabla\sum_a\pi(a|s,\theta)=b(s)\nabla(1)=0 \tag{2.4.7}$$

implies that the policy gradient theorem can be generalized to

$$\nabla J(\theta)\propto\sum_s\mu(s)\sum_a(q_\pi(s,a)-b(s))\nabla\pi(a|s,\theta). \tag{2.4.8}$$

This changes the update rule (2.4.6) only slightly, by setting

$$\theta_{t+1}:=\theta_t+\alpha_t(G_t-b(S_t))\nabla\log\pi(A_t|S_t,\theta_t). \tag{2.4.9}$$

Generally, any baseline $b(s)$ can be chosen, as long as it does not depend on the actions. While optimal baselines can be derived (in the sense that they minimize variance of the gradient estimate, see [5]), this often adds unnecessary overhead, when instead a simpler baseline can be used. An efficient baseline which can be easily maintained turns out to be $b(s)=v_\pi(s)$. Of course, the exact values are unknown and also have to be estimated. In

this case, one adds a second parametrized function approximator $\hat{v}$ of the value function to the algorithm, where

$$\hat{v}(s, w) \approx v_\pi(s), \tag{2.4.10}$$

and keeps optimizing said approximator by performing a (semi-)gradient descent step on a weighted quadratic loss measure ([1, Chapter 9]). A complete pseudocode of this procedure is given below. Note that an additional γ^t term needs to be included in order to produce unbiased gradient estimates for the discounted case, due to a different definition of the on policy state distribution (for a detailed derivation, see [7]).

Algorithm 1 MC (Vanilla) Policy Gradient: Reinforce with Baseline

Initialize a parametrized policy $\pi(a|s, \theta)$
Initialize a parametrized value function approximation $\hat{v}(s, w)$
Initialize $\alpha > 0,\ \beta > 0$
Loop for each episode
    $G \leftarrow 0$
    Sample trajectory $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$ with $\pi(\cdot|\cdot, \theta)$
    Loop for $t = T - 1, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $\delta \leftarrow G - \hat{v}(S_t, w)$
        $w \leftarrow w + \beta\delta\nabla\hat{v}(S_t, w)$
        $\theta \leftarrow \theta + \alpha\gamma^t\delta\nabla \log \pi(A_t|S_t, \theta)$

While the introduction of a baseline turns out to perform better in most cases, one has to choose an appropriate function approximator $\hat{v}(s, w)$ for the state space. Unfortunately, the choice of the approximator is not trivial and depends heavily on the structure of the state space. Furthermore, a second hyperparemeter β makes the optimization of the algorithm more tedious. A different approach, which does not introduce additional hyperparameters are *Natural Policy Gradients*, the idea of which is to perform steepest gradient ascend with respect to the Fisher information metric, as opposed to the above described *Vanilla Policy Gradients*, which use the regular Eucledian distance metric in the parameter space of θ.

2.4.2 Natural Policy Gradients

Natural policy gradients share many of the advantages that Vanilla Policy gradients have, while eliminating some of their disadvantages. The basic idea can be stated as follows: Let $G(\theta)$ denote the Fisher information matrix of the parametrized policy $\pi(\cdot|\cdot, \theta)$, that is

$$G(\theta) := \mathbb{E}_\pi[\nabla \log \pi(A|S, \theta)\nabla \log \pi(A|S, \theta)^T]. \tag{2.4.11}$$

For a general proof, why this equals the Fisher matrix of θ, see [8]. (Basically, the dynamics of the environment do not depend on θ and therefore vanish when forming the gradient. The second moment of the score function then reduces to include only the derivative of $\log \pi(A|S, \theta)$.) Then, the goal is to perform a gradient ascent step on the performance measure $J(\theta)$, by using the natural gradient

$$\nabla\hat{J}(\theta) = G^{-1}(\theta)\nabla J(\theta). \tag{2.4.12}$$

For the next algorithm, we will need to prove some theoretical results first. Recall that the expression derived in the policy gradient theorem involved a sum over action values $q_\pi(s,a)$. In [6] it has been shown that the policy gradient theorem holds, even when replacing $q_\pi(s,a)$ with an appropriate function approximator $\hat{q}_\pi(s,a,w)$:

Theorem 2.2. *Let $\hat{q}_\pi(s,a,w)$ be a compatible action-value function approximator, that is*

$$\nabla_w \hat{q}_\pi(s,a,w) = \nabla_\theta \log \pi(a|s,\theta). \tag{2.4.13}$$

Then, if additionally $\hat{q}_\pi(s,a,w)$ minimizes the mean squared error

$$\sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \pi(a|s,\theta)[q_\pi(s,a) - \hat{q}_\pi(s,a,w)]^2 \tag{2.4.14}$$

the policy gradient theorem holds, in the sense that

$$\nabla J(\theta) = \sum_s \eta(s) \sum_a \nabla_\theta \pi(a|s,\theta) \hat{q}_\pi(s,a,w). \tag{2.4.15}$$

Proof. Since $\hat{q}_\pi(s,a,w)$ is assumed to minimize (2.4.14), its gradient must be zero:

$$\sum_s \eta(s) \sum_a \pi(a|s,\theta)[q_\pi(s,a) - \hat{q}_\pi(s,a,w)] \nabla_w \hat{q}_\pi(s,a,w) = 0. \tag{2.4.16}$$

Combining this with (2.4.13) gives

$$\sum_s \eta(s) \sum_a \nabla_\theta \pi(a|s,\theta)[q_\pi(s,a) - \hat{q}_\pi(s,a,w)] = 0. \tag{2.4.17}$$

Subtracting this quantity from the original expression of the policy gradient theorem yields

$$\begin{aligned}
\nabla J(\theta) &= \sum_s \eta(s) \sum_a \nabla_\theta \pi(a|s,\theta) q_\pi(s,a) - \sum_s \eta(s) \sum_a \nabla_\theta \pi(a|s,\theta)[q_\pi(s,a) - \hat{q}_\pi(s,a,w)] \\
&= \sum_s \eta(s) \sum_a \nabla_\theta \pi(a|s,\theta)[q_\pi(s,a) - q_\pi(s,a) + \hat{q}_\pi(s,a,w)] \\
&= \sum_s \eta(s) \sum_a \nabla_\theta \pi(a|s,\theta) \hat{q}_\pi(s,a,w).
\end{aligned}$$

□

From now on, we will assume that the approximation $\hat{q}_\pi(s,a,w)$ is linear with respect to w, so that the compatibility condition (2.4.13) is met, i.e.

$$\hat{q}_\pi(s,a,w) := w^T \nabla \log \pi(a|s,\theta). \tag{2.4.18}$$

This corresponds to a linear function in the feature space of the eligibility vectors $\nabla \log \pi(a|s,\theta)$. Since the eligibility vectors give the direction in the parameter space of θ, in which the probability of taking action a in state s increases the most, action values $\hat{q}_\pi(s,a,w)$ are higher, if w points in that same direction. Under this assumption, using Theorem 2.2, we can rewrite the policy gradient in the alternative form

$$\begin{aligned}
\nabla J(\theta) &= \sum_s \eta(s) \sum_a \big(\nabla_\theta \pi(a|s,\theta)\big) \big(w^T \nabla_\theta \log \pi(a|s,\theta)\big) \\
&= \sum_s \eta(s) \sum_a \big(\pi(a|s,\theta) \nabla_\theta \log \pi(a|s,\theta)\big) \big(w^T \nabla_\theta \log \pi(a|s,\theta)\big)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_s \eta(s) \sum_a \big(\pi(a|s,\theta)\nabla_\theta \log \pi(a|s,\theta)\big)\big(\nabla_\theta \log \pi(a|s,\theta)^T w\big) \\
&= \bigg(\sum_s \eta(s) \sum_a \pi(a|s,\theta)\nabla_\theta \log \pi(a|s,\theta)\nabla_\theta \log \pi(a|s,\theta)^T\bigg) w \\
&=: F(\theta)w,
\end{aligned} \tag{2.4.19}
$$

where the last equation denotes the definition of the matrix $F(\theta)$. It is notable that Theorem 2.2 also allows for the insertion of a baseline, for the same reasons as in (2.4.7). However, this does not change the results derived in (2.4.19), since the baseline integrates out. Therefore, the choice of an appropriate baseline is eliminated and not needed anymore. When including a baseline, [6] states that $\hat{q}_\pi(s,a,w)$ should rather be thought of an approxmiation of the advantage function

$$
A_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.4.20}
$$

by noting that (2.4.18) implies that $\hat{q}(s,a,w)$ has zero mean for each state, i.e.

$$
\mathbb{E}_{A\sim\pi_\theta}[\hat{q}_\pi(s,A,w)] = \sum_a \pi(a|s,\theta)\hat{q}_\pi(s,a,w) = w^T \sum_a \nabla_\theta \pi(a|s,\theta) = 0 \tag{2.4.21}
$$

(since $\sum_a \nabla_\theta \pi(a|s,\theta) = \nabla_\theta \sum_a \pi(a|s,\theta) = \nabla_\theta(1) = 0$), which is a trait that the advantage function $A(s,a)$ also exhibits.

Now, observing that the Fisher matrix $G(\theta)$ equals the just derived matrix $F(\theta)$, (again, see [8]) our natural gradient update becomes

$$
\begin{aligned}
\nabla \hat{J}(\theta) &= G^{-1}(\theta)\nabla J(\theta) \\
&\overset{(2.4.19)}{=} G^{-1}(\theta)F(\theta)w \\
&= w.
\end{aligned} \tag{2.4.22}
$$

This greatly reduces the amount of data needed to produce good estimates, since only w needs to be estimated, as opposed to the matrix $G(\theta)$. As a result, the update simplifies to $\theta_{t+1} := \theta_t + \alpha w$, where α_t again denotes a learning rate. Unfortunately, it is impossible to learn the function approximator of the advantage function $A_\pi(s,a)$ with a traditional temporal difference bootstrapping method if the values $v_\pi(s)$ are not known, because TD makes use of comparing $v_\pi(s)$ for two adjacent states. However, this value is missing, since it gets subtracted in the advantage function. Another obvious way of learning $\hat{q}(\cdot,\cdot,w)$ would be to obtain unbiased estimates of the action values $q_\pi(s,a)$ and then to apply linear regression to minimize the mean squared error. The biggest drawback of this is that the parametrization of $\hat{q}(\cdot,\cdot,w)$ would have to be sufficiently complex, in order to prevent underfitting the training data. This can, however, be prevented to a certain extent. First, we observe that the Bellman equation yields

$$
q_\pi(s,a) = A_\pi(s,a) + v_\pi(s) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)v_\pi(s'), \tag{2.4.23}
$$

where $r(s,a)$ is a scalar deterministic reward received upon action a in state s. Now, replacing $A_\pi(s,a)$ and $v_\pi(s)$ by appropriate linear estimates, with $\hat{v}_\pi(s,v) := v^T\phi(s)$ for basis functions $\phi(s)$ and $\hat{q}(s,a,w)$ as before, we get

$$q_\pi(S_t,A_t) \approx w^T\nabla_\theta \log\pi(A_t|S_t,\theta) + v^T\phi(S_t) \approx R_{t+1} + \gamma v^T\phi(S_{t+1}). \tag{2.4.24}$$

At first glance, the introduction of the basis functions $\phi(s)$ seems overly complicated, especially since the unbiasedness of the natural gradient estimate cannot be guaranteed anymore. In the episodic case, the choice of appropriate features is fortunately not needed. By summing up (2.4.23) along a sample trajectory, we obtain

$$\sum_{t=0}^{T-1}\gamma^t A_\pi(S_t,A_t) = \sum_{t=0}^{T-1}\gamma^t R_t + \gamma^T v_\pi(S_T) - v_\pi(S_0). \tag{2.4.25}$$

Now, since S_T is terminal, we have $v_\pi(S_T) = 0$. Inserting function approximators yields

$$\sum_{t=0}^{T-1}\gamma^t w^T\nabla_\theta \log\pi(A_t|S_t,\theta) + v^T\phi(S_0) = \sum_{t=0}^{T-1}\gamma^t R_t, \tag{2.4.26}$$

where one rollout would yield one equation. If we assume that there is only a single start state, we only need to estimate $v_\pi(S_0)$. Therefore, the choice of basis functions $\phi(S)$ is not needed, besides a constant $\phi(S_0) = 1$. This reduces the problem to the regression task

$$\sum_{t=0}^{T-1}\gamma^t w^T\nabla_\theta \log\pi(A_t|S_t,\theta) + x = \sum_{t=0}^{T-1}\gamma^t R_t \tag{2.4.27}$$

for the variables $(w,x) \in \mathbb{R}^d \times \mathbb{R}$. An algorithm for estimating w this way in an episodic task was termed *Episodic Natural Actor Critic* (eNAC) and first published in [8]. A complete pseudocode of this algorithm is given in the box below.

Algorithm 2 Episodic Natural Actor Critic

Initialize a parametrized policy $\pi(a|s,\theta)$
Initialize $\alpha > 0$, $w \in \mathbb{R}^{\dim\theta}$
Loop forever
    Loop for each episode $e = 1, 2, \ldots$
        Sample trajectory $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$ with $\pi(\cdot|\cdot,\theta)$
        Calculate basis function $\phi(e) = \left[\sum_{t=0}^{T-1}\gamma^t\nabla_\theta \log\pi(a|s,\theta), 1\right]^T$ and
        reward statistic $R(e) = \sum_{t=0}^{T-1}\gamma^t R_t$
        Solve linear system of equations $\Phi\begin{pmatrix} w_e \\ x_e \end{pmatrix} = R$, where
        $R = [R(1), R(2), \ldots, R(e)]^T$ and $\Phi = [\phi(1), \phi(2), \ldots, \phi(e)]^T$
    Until natural gradient estimate w_e converged
    Update policy parameter $\theta \leftarrow \theta + \alpha w_e$

2.5 Convergence Analysis of Stochastic Gradient Descent

Since there are little to no assumptions we can make on the performance function $J(\theta)$, finding efficient optimization algorithms turns out to be cumbersome. While stochastic gradient descent is easy to implement, it is impossible to guarantee convergence to a global optimum in most cases. In the worst case scenario, even finding a local optimum turns out to be an NP-hard problem [9]. However, one can proof convergence to a stationary point, given appropriate requirements on the step sizes and objective function. The general theorem, as proven in [10], reads as follows.

Theorem 2.3. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ denote a differentiable objective function and x_t be a sequence recursively defined by*

$$x_{t+1} := x_t + \alpha_t(s_t + w_t), \tag{2.5.1}$$

where $\alpha_t > 0$ denotes the step size and s_t the descent direction, which is disturbed by the noise w_t. Furthermore, let $\mathcal{F}_t$ be an increasing sequence of σ-fields. Now, assume that the following conditions hold:

- *x_t and s_t are $\mathcal{F}_t$ measurable.*
- *There exist constants $c_1 > 0$ and $c_2 > 0$ with*

$$c_1\|\nabla f(x_t)\|^2 \leq -\nabla f(x_t)'s_t, \qquad \|s_t\| \leq c_2(1 + \|\nabla f(x_t)\|). \tag{2.5.2}$$

- *For $c_3 > 0$ and for all t it holds with probability one that*

$$\mathbb{E}\left[w_t|\mathcal{F}_t\right] = 0, \qquad \mathbb{E}\left[\|w_t\|^2|\mathcal{F}_t\right] \leq c_3\left(1 + \|\nabla f(x_t)\|^2\right). \tag{2.5.3}$$

- *The step sizes fulfill*

$$\sum_{t=0}^{\infty} \alpha_t = 0, \qquad \sum_{t=0}^{\infty} \alpha_t^2 = \infty. \tag{2.5.4}$$

- *∇f is Lipschitz continuous with Lipschitz constant L, i.e.*

$$\|\nabla f(x) - \nabla f(x')\| \leq L\|x - x'\|. \tag{2.5.5}$$

Then, either $f(x_t) \to -\infty$ or x_t converges to a stationary point, i.e. $\lim_{t\to\infty} \nabla f(x_t) = 0$ and $f(x_t)$ converges to a finite value (almost surely).

Proof. For a sketch of the proof, see the appendix. □

The theorem proofs convergence of stochastic gradient descent for a more general case, than is actually needed for most policy gradient methods. The second assumption is automatically fulfilled, if the descent direction is chosen to be the steepest descent, i.e. $s_t = -\nabla f(x_t)$, since then $\|\nabla f(x_t)\|^2 = -\nabla f(x_t)'s_t$ and $\|s_t\| = \|\nabla f(x_t)\|$. The measurability constraints are only needed, since $\mathcal{F}_t$ is at first assumed to be an arbitrary increasing σ-field sequence. Thinking of $\mathcal{F}_t$ as the trajectory of events $x_0, s_0, w_0 \ldots, w_{t-1}, x_t, s_t$ explains the first assumption. The third assumption ensures that the updates match the

descent direction in expectation. By the policy gradient theorem, this is guaranteed for the Monte Carlo Reinforce procedure (Algorithm 1), since each episode a batch update is performed in expected direction of the steepest descent of $J(\theta)$, given that the data was sampled under the past weight $x_t := \theta_t$. Obviously, one has to assume that the gradient $\nabla J(\theta)$ is Lipschitz continuous as well as the boundedness of the variances $\mathbb{E}\left[\|w_t\|^2|\mathcal{F}_t\right]$, since this cannot be shown to hold for any arbitrary MDP. Fortunately, given these assumptions are true, the convergence of gradient descent using natural policy gradients follows from the same result, since it is guaranteed that the angle between the natural and the vanilla policy gradient is below 90 degrees (as needed in (2.5.2)). This follows from the positive definiteness of the Fisher information matrix $G(\theta)$ (2.4.11) and therefore the positive definiteness of $G(\theta)^{-1}$. Using the angle formula in $\mathbb{R}^n$, we then obtain that

$$0 < x^T G(\theta)^{-1} x = \|x\| \|G(\theta)^{-1} x\| \cos(\eta),$$

where η denotes the angle between the vector $x \in \mathbb{R}^n$ and $G(\theta)^{-1}x$. By the definition of the natural policy gradient (2.4.12), its angle must be less than 90 degrees from the vanilla policy gradient.

2.6 Experience Replay and Value-Function Approximators

As with most reinforcement learning algorithms, policy gradient methods make use of other areas in machine learning, which deal with function approximation and therefore also share difficulties such as under- and overfitting data as well as hyperparameter optimization. While most algorithms are short to formulate, many details have to be considered when implementing such procedures. One of the most important ones for our purposes is obviously the choice of policy parametrization $\pi(\cdot|\cdot, \theta)$. For a discrete setting, coding schemes like coarse- or tile-coding are unfeasible, as well as a traditional basis function approach, which either requires handcrafted features or, for high dimensional environments, extremely many automatically selected features in order to capture the structure of the training data. A common approach is to use a neural network parametrization followed by a final softmax layer

$$\sigma(z)_k = \frac{e^{z_k}}{\sum_{i=1}^{|\mathcal{A}|} e^{z_i}} \tag{2.6.1}$$

to convert computed values into a vector which provides a probability of selecting the k-th action. We did this using networks with a single (dense) hidden layer of variable size, ranging from 10 to 50 neurons, depending on the complexity of the environment. The implementations of the networks were mainly built in the Julia machine learning framework `Flux` [11]. To optimize the speed of gradient descent convergence, SGD with momentum was sometimes used alongside classical SGD. Batch normalization, as described in [12], did not seem to increase the performance of the neural network by any significant amount. In case of additionally approximating a value function, as is common in actor-critic algorithms and Reinforce with baseline, two different approaches have been tested. On the one hand, memory based methods like k-nearest neighbor regression (KNN) seem promising due to its high accuracy for previously often visited states. On the other hand, a second neural network for approximating the state space is a more memory efficient solution, which comes at the cost of having to optimize a second learning rate. As it

Algorithm 3 Reinforce with Baseline and Experience Replay

Initialize a parametrized policy $\pi(a|s, \theta)$
Initialize $\alpha > 0$, $\beta > 0$
Initialize empty array of (circular) buffers experience(t) of size $B \in \mathbb{N}$
Loop for each episode
    $G \leftarrow 0$
    Sample trajectory $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$ with $\pi(\cdot|\cdot, \theta)$
    Loop for $t = T - 1, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append the tuple (S_t, A_t, G) to experience(t)
        Sample $(\hat{S}, \hat{A}, \hat{G})$ uniformly from experience(t)
        $\delta \leftarrow \hat{G} - \hat{v}(\hat{S}, w)$
        $w \leftarrow w + \beta\delta\nabla\hat{v}(\hat{S}, w)$
        $\theta \leftarrow \alpha\gamma^t\hat{G}\nabla \log \pi(\hat{A}|\hat{S}, \theta)$

turns out, both approaches are effective overall and seem to scale well with increasingly large state spaces, see section 3.

Despite the wide success of artificial neural networks in the past decade, the correct choice of a learning rate for gradient descent is still troublesome, especially in reinforcement learning. While multiple approaches, such as the ADAM optimizer [13] have been proposed to efficiently choose adaptive learning rates, the problem of needing multiple passes through the network in order to achieve notable learning progress persists. Furthermore, subsequent gradient descent steps might work against each other and make the network forget past information by overwriting the weights. One way to deal with this problem, was introduced by [14] in the context of learning to play Atari 2600 games. The paper proposed a Q-learning agent with a neural network action value function and was trained on the images of the games. Since the states of these games are often highly correlated and only little parts of the picture change each frame, they introduced a mechanism called *experience replay*, which in essence means keeping a buffer of state-action-reward tuples and sampling from it during training, instead of directly using the just received state from the environment to perform a gradient descent step. Besides its novel name, such ideas are quite common in reinforcement learning and similar procedures have been proposed in the past for model based methods, such as the Dyna architecture [1, Chapter 8]. One of the biggest advantages of experience replay is, that it is easy to implement and does not add much computational complexity to the algorithms. Furthermore, it works very well for highly stochastic environments, since resampling from old seen data reduces the variance of the gradients to a certain extent. Unfortunately, policy gradient methods like Reinforce need to sample from the on policy distribution in order to perform unbiased gradient estimates, which makes the usage of large experience buffers mathematically incorrect. However, when keeping the experience buffer small (like short term memory), it still seems to work well and greatly improve the learning speed. Pseudocode of an implementation can be found in Algorithm 3.

3 Results

3.1 The Tabular Case

We will start with the tabular case, which allows for a wider class of algorithms to be applied. Due to the high dimensionality of the state space, however, we are constrained to small factories. The encoding of the state contains binary information about the status of each of the n machines, current positions of the r spare machines and workload variables $z_k \in \{0, 1, 2, 3\},\ k \in 1 \dots n$. In total, this gives us

$$|\mathcal{S}| = 2^n \cdot 4^n \cdot (n+1)^r = 8^n (n+1)^r, \tag{3.1.1}$$

which increases exponentially, making the use of tabular methods unfeasible for $n > 6$. The following figures show online training performance of various algorithms. To enhance readability, the curves have been smoothed using the `geom_smooth` function of the `ggplot2` library, as implemented in the Julia package `Gadfly`. In figure 2 we plotted Monte Carlo methods as well as Q-learning and Reinforce against each other, where Reinforce used an ANN softmax policy and an ANN baseline for value estimation. The ϵ-greediness of the tabular algorithms was set to $\epsilon = 0.1$, where no discounting was used ($\gamma = 1$). Overall, Reinforce without baseline and Monte Carlo off-policy perform the worst. For Reinforce, this is due to a small learning rate to counteract the high variance of the gradients.

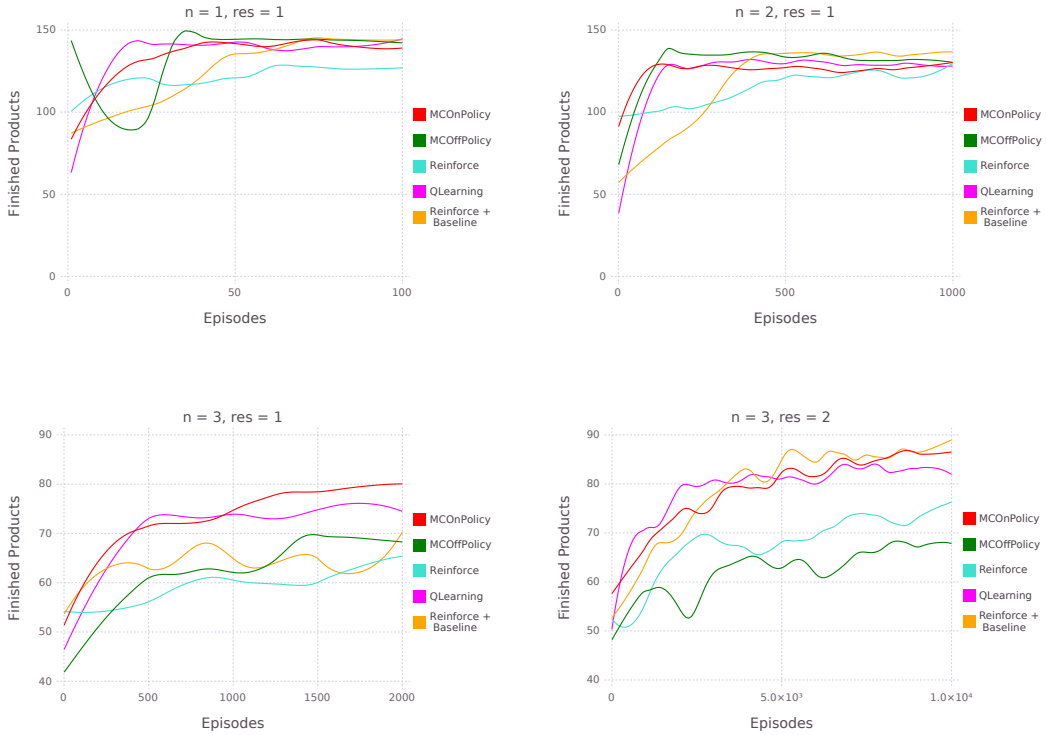

Figure 2: Comparison of training performance for increasingly complex environments (n denoting the amount of machines and res denoting the amount of spare machines).

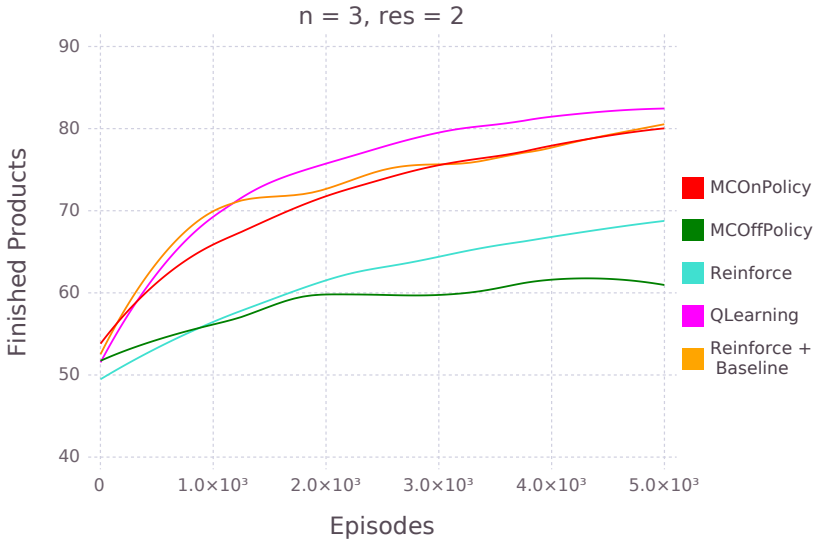

Figure 3: Average training performance over 25 runs, with 5000 episodes each.

By tuning the hyperparameters correctly, Reinforce with baseline has the capabilities of matching the performance of Q-learning and Monte Carlo on-policy, but stays closer to that of MC on-policy. This becomes apparent when averaging training performance over multiple runs, see figure 3. Unfortunately, Monte Carlo off-policy did not improve when using per decision sampling, so it seems like due to the nature of MC off-policy control only learning from the ends of the episodes, the algorithm struggles with backpropagating information to the start of the episode, which is crucial for saving the first machine from overloading and getting the assembly line stuck.

3.2 Complex Environments

In more complex situations, learning can get difficult due to high variance in the estimates of the policy gradient. As in the tabular case, using an appropriate baseline can speed up the convergence of Reinforce, see figure 4. While a neural network with a softmax output layer is an appropriate choice for parametrizing a policy in a highly dimensional discrete space, it is not clear a-priori which function approximator to use for a value function baseline. In any case, the use of a small experience replay buffer helps training performance, despite not being completely mathematically sound for Monte Carlo policy gradients (see figure 5). Figures 6 and 7 compare the performance of the underlying baseline approximator. At first glance, approximation of the value function through a neural network might seem to yield the best results. However, the introduction of an additional hyperparameter makes optimization more tedious than it is for a memory based approach, which does not need any additional parameters to be tuned. This makes KNN approximation overall more robust, but comes with the disadvantage of consuming considerably more memory, which might become problematic for very large discrete state spaces. Overall, when perfectly optimizing a neural network approximator, it will often slightly outperform the KNN approxmiator for large state spaces.

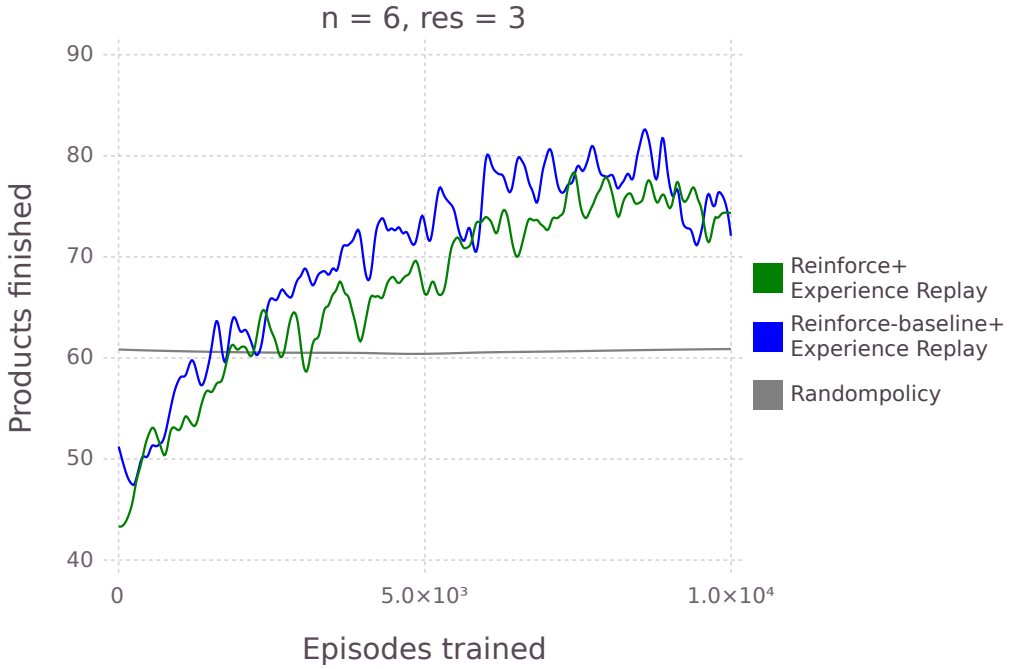

Figure 4: Single run training performance of Reinforce versus adding a time-variant value baseline through neural network approximation.

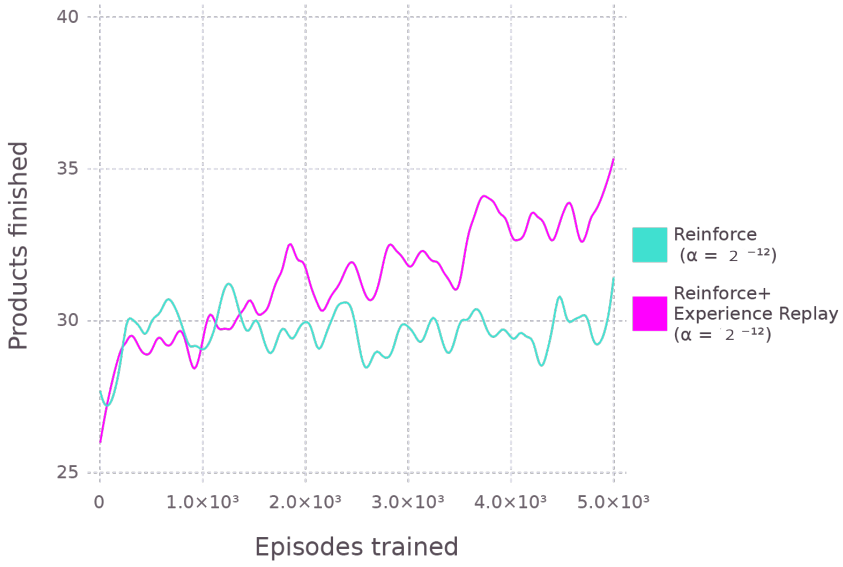

Figure 5: Adding a small experience buffer (100 samples per time step were used for this figure) helps with overriding past experiences. Without it, almost no learning performance can be observed.

Figure 6: Comparison of using various SGD step sizes, where α denotes the step size used for the neural network policy and β denotes the step size used for performing the semi-gradient descent step on the value function baseline. The yellow curve was produced using a KNN approximation for the baseline and thus has only one hyperparameter α.

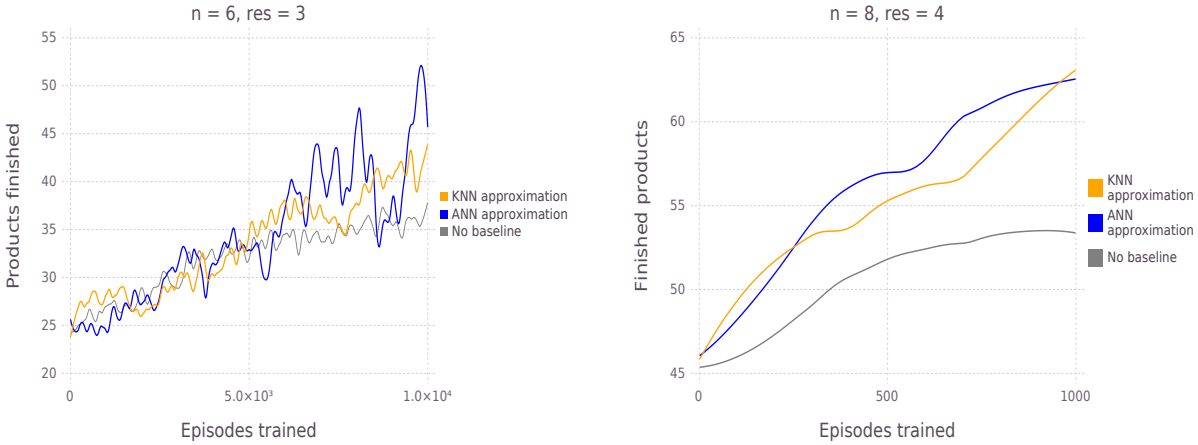

Figure 7: Comparison of KNN versus ANN baseline approximators. Left: Smoothed training performance using optimal hyperparameters (optimized to the next best power of 2). Right: Average training performance over 10 runs (again using optimized hyperparameters).

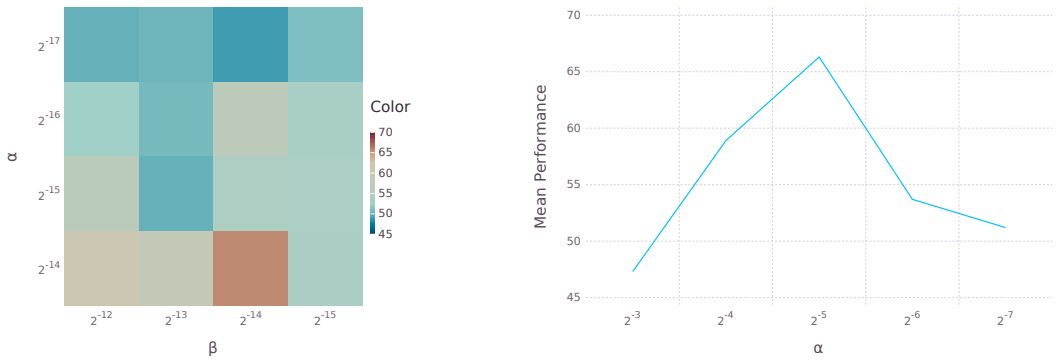

Figure 8: Left: Average training performance of the hyperparameters α and β for Reinforce with an ANN baseline. Right: Average training performance of eNAC with varying learning rates.

3.3 Vanilla versus Natural Policy Gradients

While Reinforce with baseline can perform quite well, the choice of appropriate function approximators and learning rates is critical for achieving good performance. As already mentioned, the natural policy gradient algorithm eNAC only has one hyperparameter and does not need an explicit approximation for the baseline to be included. This greatly simplifies the parameter tuning stage of implementation. Figure 8 illustrates the addi-

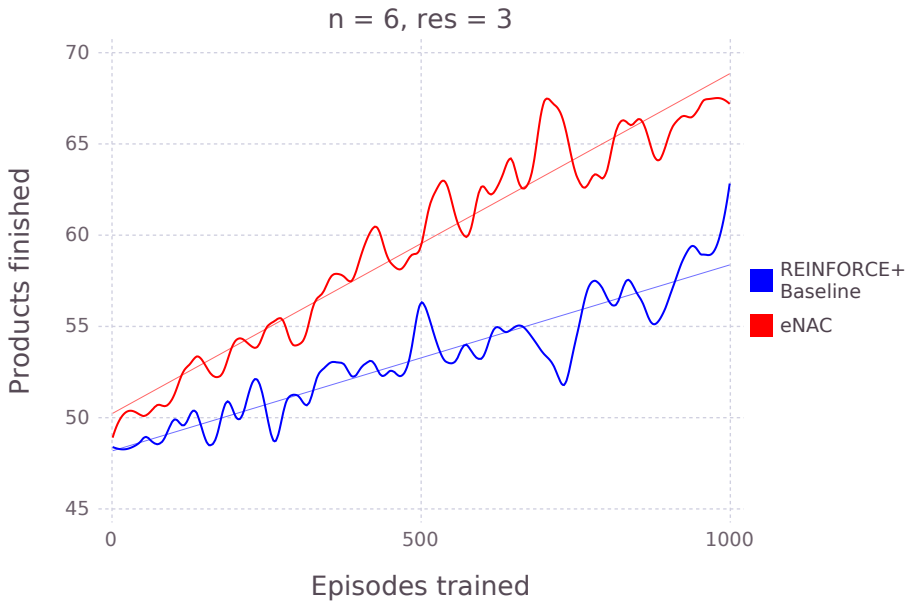

Figure 9: Average online training performance of eNAC versus Reinforce with a ANN baseline after 10 runs of 1000 episodes. Linear regression lines were added to illustrate learning speed. The respective learning rates were optimized as illustrated in Figure 8.

tional complexity when needing to optimize hyperparameters on a two dimensional grid, as opposed to a one dimensional optimization problem for eNAC. The biggest advantage of Reinforce over eNAC is that it does generally need less computation time, since the linear regression in eNAC involves matrix inversion and can be quite costly for high dimensional parameters θ. Overall, however, eNAC does exhibit slightly better learning speeds, when comparing the amount of episodes needed to learn. This makes sense, since natural policy gradients are designed to avoid getting stuck in local plateaus of the objective function by averaging out the influence of the stochastic policy as well as the baseline of the function approximator. Furthermore the natural policy gradient is actually covariant, which means it is independent of the coordinatization of $\pi(\cdot|\cdot,\theta)$ [8]. Despite the performance boost of natural policy gradients, the eNAC algorithm comes with the disadvantage of consuming more memory. This can be seen in figure 10. It shows that for growing size of the intermediate network layer of the policy parametrization, as well as for growing complexity of the underlying environment, eNAC grows exponentially. On the other hand, Reinforce with an ANN baseline does only grow by a negligible amount, whereas Reinforce with a KNN baseline only needs more memory for more complex environments. Note that this analysis was done under fixed episode lengths. If the episode lengths were unbounded, then eventually the KNN baseline would overtake eNAC in memory consumption, assuming that all memory points are being saved.

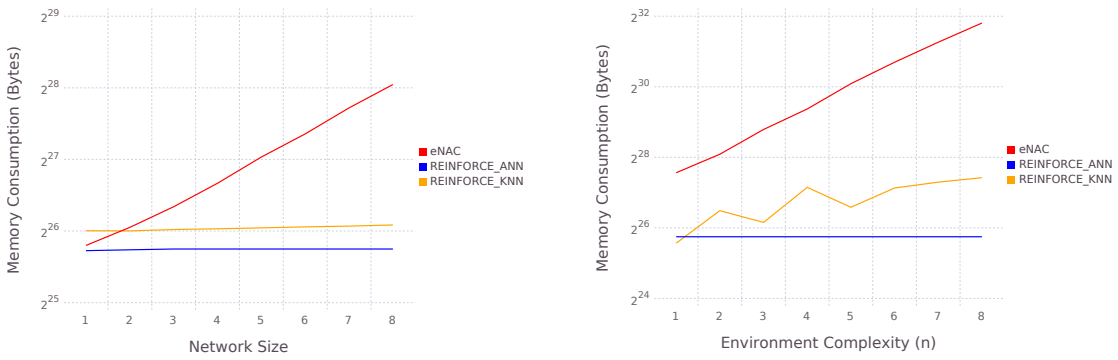

Figure 10: Left: Comparison of memory consumption for varying size of the intermediate ANN layer and varying environments, keeping one of both components fixed respectively.

4 Conclusion

Overall, discrete event simulation is a suitable tool for formulating automation tasks, which copes well with the reinforcement learning framework. The ability to generate arbitrarily large amounts of data without any repercussions is an essential key to making approximative learning algorithms work well, given enough computation time.

While there still is a lack of mathematical convergence proofs for tabular Monte Carlo algorithms, they seem to converge to an optimal policy, just like Q-learning, whose convergence has indeed been proven. However, bootstrapping action values seems to benefit the learning speed enough, in order for Q-learning to converge faster than any of the other tested methods. Reinforce with baseline has a performance comparable to that of Monte Carlo on-policy in small environments, but cannot beat Q-learning, which makes sense, since as a Monte Carlo procedure it does not make use of any value bootstrapping and waits until the end of an episode to perform an update.

In larger environments, where tabular methods become unfeasible, policy gradient methods form a class of algorithms which have convenient theoretical properties, because convergence results of stochastic gradient descent are applicable. Furthermore, they exhibit good performance when using appropriate function approximators. Artificial neural networks form a versatile method of parametrizing the policy as well as additional value functions, when coupled with experience buffers that ensure long lasting updates on the parameters. To counteract high variances during learning, k-nearest neighbor value approximation yields promising results at the cost of using more memory. Nonetheless, gradients greatly vary in size in highly stochastic environments, which limits one to using small learning rates or clipping gradients. This makes vanilla policy gradients very prone to getting stuck in local plateaus of the objective function.

Natural policy gradients, although having the same theoretical convergence properties as vanilla policy gradients, tend to learn significantly faster in all environments considered. Whether or not the additional memory requirements of the episodic Natural Actor Critic framework become critically large, depends heavily on the complexity of the environment as well as on the size of the neural network policy.

Appendix

Proof of theorem 2.1 (excerpted from [6]):
Let $\pi(\cdot|\cdot,\theta)$ be a parametrized policy with parameter $\theta \in \mathbb{R}^n$. From now on, differentiation will be carried out only with respect to θ. We start out by noting that, by the law of total expectation, the value function can be written as

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a) \qquad \forall s \in \mathcal{S}. \tag{1}$$

Taking the gradient on both sides and applying the product rule as well as applying the Bellman equation to the action value function yields

$$\begin{aligned}
\nabla v_\pi(s) &= \sum_a \nabla\pi(a|s) q_\pi(s,a) + \pi(a|s)\nabla q_\pi(s,a) && (2)\\
&= \sum_a \left(\nabla\pi(a|s) q_\pi(s,a) + \pi(a|s)\nabla \sum_{s',r} p(s',r|s,a)(r+\gamma v_\pi(s')) \right) && (3)\\
&= \sum_a \left(\nabla\pi(a|s) q_\pi(s,a) + \pi(a|s)\gamma \sum_{s',r} p(s',r|s,a)\nabla v_\pi(s') \right). && (4)
\end{aligned}$$

Now, recursively inserting the just derived equality for $\nabla v_\pi(s)$ gives us the following equality

$$\nabla v_\pi(s) = \sum_{s'\in\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(s \to s', k, \pi) \sum_{a\in\mathcal{A}} \nabla\pi(a|s') q_\pi(s',a), \tag{5}$$

where $\mathbb{P}(s \to s', k, \pi)$ denotes the probability of transitioning from a state s to another state s' in k steps while following $\pi(\cdot|\cdot,\theta)$. After plugging in the definition of the performance measure and noting that the expected number $\eta(s)$ of visits to a certain state s during an episode is exactly the sum of probabilities of reaching s from the start in a finite amount of steps, we get

$$\begin{aligned}
\nabla J(\theta) &= \nabla v_\pi(s_0) && (6)\\
&\overset{(5)}{=} \sum_{s\in\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(s_0 \to s, k, \pi) \sum_{a\in\mathcal{A}} \nabla\pi(a|s) q_\pi(s,a) && (7)\\
&= \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \nabla\pi(a|s) q_\pi(s,a) && (8)\\
&= \left(\sum_{s'\in\mathcal{S}} \frac{\eta(s')}{\sum_{s'\in\mathcal{S}} \eta(s')} \right) \sum_{s\in\mathcal{S}} \eta(s) \sum_{a\in\mathcal{A}} \nabla\pi(a|s) q_\pi(s,a) && (9)\\
&= \sum_{s'\in\mathcal{S}} \eta(s') \sum_{s\in\mathcal{S}} \frac{\eta(s)}{\sum_{s'\in\mathcal{S}} \eta(s')} \sum_{a\in\mathcal{A}} \nabla\pi(a|s) q_\pi(s,a) && (10)\\
&\propto \sum_{s\in\mathcal{S}} \frac{\eta(s)}{\sum_{s'\in\mathcal{S}} \eta(s')} \sum_{a\in\mathcal{A}} \nabla\pi(a|s) q_\pi(s,a). && (11)
\end{aligned}$$

■

Proof sketch of theorem 2.3. The following proof is a summary of the proof given in [10]. We will start out by establishing an upper bound for $f(x_{t+1})$, given the previous value $f(x_t)$.

Lemma 4.1. *Under the assumptions of theorem 2.3 and for t large enough that $\alpha_t 2Lc_2^2 \leq \frac{c_1}{2}$ it holds that*

$$f(x_{t+1}) \leq f(x_t) - \alpha_t \frac{c_1}{2}\|\nabla f(x_t)\|^2 + \alpha_t \nabla f(x_t)' w_t + \alpha_t^2 2Lc_2^2 + \alpha_t L\|w_t\|^2. \tag{12}$$

Proof. A standard trick in analysis yields for two fixed vectors x and z and $g\colon \mathbb{R} \to \mathbb{R}$, $g(\xi) = f(x + \xi z)$ that

$$\begin{aligned} f(x+z) - f(x) &= g(1) - g(0) \\ &= \int_0^1 g'(\xi) d\xi \\ &= \int_0^1 z' \nabla f(x + \xi z) d\xi \\ &\leq \int_0^1 z' \nabla f(x) d\xi + \left| \int_0^1 z'(\nabla f(x + \xi z) - \nabla f(x)) d\xi \right| \\ &\leq z' \nabla f(x) + \int_0^1 \|z\| \cdot \|\nabla f(x + \xi z) - \nabla f(x)\| d\xi \\ &\leq z' \nabla f(x) + \|z\| \int_0^1 L\xi \|z\| d\xi \\ &= z' \nabla f(x) + \frac{L\|z\|^2}{2}. \end{aligned} \tag{13}$$

Inserting the update rule 2.5.1 and applying the inequalities for s_t, we get

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) + \alpha_t \nabla f(x_t)'(s_t + w_t) + \alpha_t^2 \frac{L}{2}\|s_t + w_t\|^2 \\ &\leq f(x_t) - \alpha_t c_1 \|\nabla f(x_t)\|^2 + \alpha_t \nabla f(x_t)' w_t + \alpha_t^2 2Lc_2^2 \\ &\quad + \alpha_t^2 2Lc_2^2 \|\nabla f(x_t)\|^2 + \alpha_t^2 L\|w_t\|^2. \end{aligned} \tag{14}$$

Then, for t large enough, (12) follows. □

To proof the theorem, one partitions the time steps t into sets where the gradients are small to a certain extent and into intervals $I_k = \{\tau_k, \ldots, \tau_{k+1}\}$ where the gradient stays above a constant. Let $\delta > 0$ be an arbitrary constant. Then the intervals I_k are recursively defined by

$$\tau_k = \min\{t > \tau'_{k-1} | \|\nabla f(x_t)\| \geq \delta\} \tag{15}$$

as long as such a t exists. Furthermore, let

$$\tau'_k = \max\left\{ t \geq \tau_k \;\middle|\; \sum_{i=\tau_k}^{t} \alpha_i \leq \eta, \quad \frac{\|\nabla f(x_{\tau_k})\|}{2} \leq \|\nabla f(x_t)\| \leq 2\|\nabla f(x_{\tau_k})\| \right\}, \tag{16}$$

where η is defined such that $\eta c_2(1/\delta + 2) + \eta = \frac{1}{2L}$. The first constraint ensures that the intervals don't get too large and we say that an interval I_k is full, if it cannot get any larger in the sense that $\sum_{t=\tau_k}^{\tau_k'+1} \alpha_t > \eta$. Additionally, let $S = \mathbb{N} \setminus \bigcup_k I_k$ and

$$G_t := \begin{cases} \delta, & t \in S, \\ \|\nabla f(x_{\tau_k})\| =: H_k, & t \in I_k. \end{cases} \tag{17}$$

To ensure convergence of various sequences, we need the following lemma.

Lemma 4.2. *Let r_t be a sequence of random variables where r_t is $\mathcal{F}_{t+1}$measurable and $\mathbb{E}[r_t|\mathcal{F}_t] = 0,\ \mathbb{E}[\|r_t\|^2|\mathcal{F}_t] \leq B$ for $B \geq 0$. Then, with probability 1 the inequalities*

$$\sum_{t=0}^{\infty} \alpha_t r_t < \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 \|r_t\|^2 < \infty \tag{18}$$

hold. As a consequence, the convergence of the sequences

- $\sum_{t=0}^{T} \mathbb{1}_S(t)\alpha_t \nabla f(x_t)' w_t$,
- $\sum_{t=0}^{T} \alpha_t \frac{w_t}{G_t}$,
- $\sum_{t=0}^{T} \alpha_t \frac{\nabla f(x_t)' w_t}{G_t^2}$,
- $\sum_{t=0}^{T} \alpha_t^2 \frac{\|w_t\|^2}{G_t^2}$,
- $\sum_{t=0}^{T} \alpha_t^2 \chi_t \|w_t\|^2$,

can be guaranteed.

The proof follows from the martingale convergence theorem and from (2.5.4) (for details, see [10]). We now introduce a final constant ϵ, for which all of the above sequences are in an ϵ ball around their respective limits for $T \geq t_0$. Furthermore w.l.o.g. we can assume

$$\epsilon \leq \eta, \qquad 2\epsilon + 2L\epsilon \leq \frac{c_1\eta}{48}, \qquad 4Lc_2^2\epsilon \leq \frac{c_1\delta^2\eta}{48}. \tag{19}$$

We can now prove the following lemma.

Lemma 4.3. *For t_0 defined as above, it holds that for all $\tau_k > t_0$, the interval I_k is full.*

Proof. Let $t \in I_k$. Per definition $G_t = H_k = \|\nabla f(x_{\tau_k})\| \geq \delta$ and $\|s_t\| \leq c_2(1 + 2H_k)$. Combining this with the definition of η, we get

$$\begin{aligned} \|x_{\tau_k'+1} - x_{\tau_k}\| &= \left\| \sum_{t=\tau_k}^{\tau_k'} \alpha_t (s_t + w_t) \right\| \\ &\leq \sum_{t=\tau_k}^{\tau_k'} \alpha_t \|s_t\| + \left\| \sum_{t=\tau_k}^{\tau_k'} \alpha_t w_t \right\| \end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=\tau_k}^{\tau_k'} \alpha_t \|s_t\| + H_k \left\| \sum_{t=\tau_k}^{\tau_k'} \alpha_t \frac{w_t}{G_t} \right\| \\
&\leq \eta c_2 (1 + 2H_k) + H_k \epsilon \\
&\leq \eta c_2 H_k (1/\delta + 2) + \eta H_k \\
&= \frac{H_k}{2L}.
\end{aligned} \tag{20}
$$

Ultimately,

$$
\|\nabla f(x_{\tau_k'+1}) - \nabla f(x_{\tau_k})\| \leq L \|x_{\tau_k'+1} - x_{\tau_k}\| \leq \frac{H_k}{2} = \frac{\nabla f(x_{\tau_k})}{2}, \tag{21}
$$

so the boundedness of the gradients is ensured, i.e.

$$
\frac{1}{2} \|\nabla f(x_{\tau_k})\| \leq \|\nabla f(x_{\tau_k'+1})\| \leq 2 \|\nabla f(x_{\tau_k})\|. \tag{22}
$$

Therefore, the first condition in the definition of I_k has to fail, otherwise $\tau_k' + 1 \in I_k$. □

This lemma is very useful, since the gradients are large enough during a full interval, one can show a guaranteed decrease in the value of the objective function:

Lemma 4.4. *For $\tau_k > t_0$, the inequality*

$$
f(x_{\tau_k'+1}) \leq f(x_{\tau_k}) - h(\delta) \tag{23}
$$

holds, where $h(\delta) > 0$. Note that δ has been assumed to be constant since the start of the proof, so h can be seen as a constant.

Proof. By Lemma 4.1 we have

$$
f(x_{t+1}) - f(x_t) \leq -\alpha_t \frac{c_1}{2} \|\nabla f(x_t)\|^2 + \alpha_t \|\nabla f(x_t)\|^2 + \alpha_t \nabla f(x_t)' w_t + \alpha_t^2 2 L c_2^2 + \alpha_t^2 L \|w_t\|^2. \tag{24}
$$

Summing the above quantity over the interval I_k, we obtain upper bounds for each part of the right hand side. Since I_k is a full interval (Lemma 4.3), for t_0 large enough the inequalities

$$
\frac{\eta}{2} \leq \eta - \alpha_{\tau_k'+1} < \sum_{t=\tau_k}^{\tau_k'} \alpha_t \leq \eta \tag{25}
$$

hold. Furthermore, if $t \in I_k$, by definition $\|\nabla f(x_t)\| \geq \frac{H_k}{2}$. This ensures

$$
-\sum_{t=\tau_k}^{\tau_k'} \alpha_t \frac{c_1}{2} \|\nabla f(x_t)\|^2 \leq -\frac{c_1 H_k^2}{8} \sum_{t=\tau_k}^{\tau_k'} \alpha_t \leq -\frac{c_1 H_k^2 \eta}{16}. \tag{26}
$$

For the second quantity, we note that

$$
\sum_{t=\tau_k}^{\tau_k'} \alpha_t \frac{\nabla f(x_t)' w_t}{G_t^2} = \frac{1}{H_k} \sum_{t=\tau_k}^{\tau_k'} \alpha_t \nabla f(x_t)' w_t. \tag{27}
$$

Since for $T > t_0$ we have that $\sum_{t=0}^{T} \alpha_t \frac{\nabla f(x_t)' w_t}{G_t^2}$ is within ϵ of its limit (see above), summing from τ_k to τ_k' cannot change the value for more than ϵ, and must therefore be smaller than ϵ, so overall

$$\sum_{t=\tau_k}^{\tau_k'} \alpha_t \nabla f(x_t)' w_t \leq 2H_k^2 \epsilon. \tag{28}$$

In a similar manner, we get the bound

$$\sum_{t=\tau_k}^{\tau_k'} L\alpha_t^2 \|w_t\|^2 \leq 2LH_k^2 \epsilon \tag{29}$$

on the last quantity, whereas the second last quantity can be bounded by the definition

$$2Lc_2^2 \sum_{t=\tau_k}^{\tau_k'} \alpha_t^2 \leq 4Lc_2^2 \epsilon \tag{30}$$

of ϵ. Inserting and noting that $\delta \leq H_K$ as well as using the bounds for ϵ as defined in (19) gives

$$f(x_{\tau_k'+1}) \leq f(x_{\tau_k}) - \frac{c_1 \eta \delta^2}{48}. \tag{31}$$

□

At this point, the proof can be finished for two different cases. In the first case, we assume that there are only finitely many intervals I_k where the gradients are large. In this case, it is by definition guaranteed that $\limsup_{t\to\infty} \|\nabla f(x_t)\| \leq \delta$, where δ was arbitrarily small. The convergence of $f(x_t)$ to a finite value can be shown by using the convergence of the series defined in Lemma 4.2. For the case that there are infinitely many I_k, we can show hat $f(x_t)$ diverges to $-\infty$. This makes sense intuitively, considering Lemma 4.4. First, note that the following statement holds.

Lemma 4.5. *Let Y_t, W_t and Z_t be three sequences, where $W_t \geq 0$ for all t, $\sum_{t=0}^{\infty} Z_t < \infty$ and*

$$Y_{t+1} \leq Y_t - W_t + Z_t. \tag{32}$$

Then either $Y_t \to -\infty$ or $Y_t \to y \in \mathbb{R}$ and $\sum_{t=0}^{\infty} W_t < \infty$.

Then, consider the subsequence $\{f(x_t)\}_{t\in\mathcal{T}}$ along the set $\mathcal{T} = S \cup \{\tau_1, \tau_2, \dots\}$. When $t \in S$, the inequality (12) can be written as

$$f(x_{t+1}) - f(x_t) \leq \alpha_t \mathbb{1}_S(t) \nabla f(x_t)' w_t + \alpha_t^2 2Lc_2^2 + \mathbb{1}_S(t) \alpha_t^2 L \|w_t\|^2, \tag{33}$$

since $\mathbb{1}_S(t) = 1$. The sum of the terms on the right hand side converges, because of Lemma 4.2. For $t \in \mathcal{T} \setminus S$ we have that $f(x_t)$ decreases strictly inside one interval, due to Lemma 4.4. Setting $W_t = h(\delta)$ for $t \in \{\tau_1, \tau_2, \dots\}$ and 0 otherwise, as well as setting Z_t to the right hand side of (33), we can apply Lemma 4.5 to the subsequence $\{f(x_t)\}_{t\in\mathcal{T}}$ and get in fact divergence to $-\infty$, since $\sum_{t=0}^{\infty} W_t = \infty$. To get convergence

for the whole sequence, one has to show that during the intervals I_k the fluctuations do not get arbitrarily large and can be bounded by a constant. If there is no $t \in S$ between two intervals I_k and I_{k-1}, one can show, similarly to Lemma 4.4 that $f(x_t) \leq f(x_{\tau_k-1})$ for all $t \in I_k$. This gives the convergence to $-\infty$ of $f(x_t),\ t \in I_k$. Otherwise, if there is a $t' \in S$ between intervals I_k and I_{k-1}, one can proceed in a similar manner as in (20) to find a bound for $x_t - x_{\tau_k-1}$ for $t \in I_k$ and since $\|\nabla f(x_{\tau_k-1})\| \leq \delta$ also a bound for $f(x_t) - f(x_{\tau_k-1})$ can be obtained, which guarantees convergence to $-\infty$ for $f(x_t)$ outside the subsequence. ■

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second Edition. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[2] D. Precup, R. Sutton, and S. Singh, "Eligibility traces for off-policy policy evaluation," in *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 759–766, Jun. 2000.

[3] P. Montague, P. Dayan, and T. Sejnowski, "A framework for mesencephalic dopamine systems based on predictive Hebbian learning," *The Journal of Neuroscience*, vol. 16, pp. 1936–47, Apr. 1996. DOI: 10.1523/JNEUROSCI.16-05-01936.1996.

[4] C. J. C. H. Watkins, "Learning from delayed rewards," PhD thesis, King's College, Cambridge, UK, Jun. 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[5] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006, pp. 2219–2225. DOI: 10.1109/IROS.2006.282564.

[6] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, Feb. 2000.

[7] P. Thomas, "Bias in natural actor-critic algorithms," Proceedings of Machine Learning Research, vol. 32, pp. 441–448, Jun. 2014. [Online]. Available: http://proceedings.mlr.press/v32/thomas14.html.

[8] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, Mar. 2008. DOI: 10.1016/j.neucom.2007.11.026. [Online]. Available: https://doi.org/10.1016/j.neucom.2007.11.026.

[9] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, no. 2, pp. 117–129, Jun. 1987. DOI: 10.1007/bf02592948.

[10] D. P. Bertsekas and J. N. Tsitsiklis, "Gradient convergence in gradient methods with errors," *SIAM Journal on Optimization*, vol. 10, no. 3, pp. 627–642, Jan. 2000. DOI: 10.1137/s1052623497331063.

[11] M. Innes, "Flux: Elegant machine learning with julia," *Journal of Open Source Software*, 2018. DOI: 10.21105/joss.00602.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. eprint: arXiv:1412.6980.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. [Online]. Available: http://dx.doi.org/10.1038/nature14236.

[15] S. Robinson, *Simulation – The Practice of Model Development and Use*. New York: Wiley, 2004, ISBN: 978-0-470-84772-5.