Université de Montréal

Parsimonious Reasoning in Reinforcement Learning for Better Credit Assignment

par

Michel Ma

Department of Computer Science and Operations Research
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Computer Science

August 31, 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Parsimonious Reasoning in Reinforcement Learning for Better Credit Assignment

présenté par

Michel Ma

a été évalué par un jury composé des personnes suivantes :

Irina Rish
(président-rapporteur)

Pierre-Luc Bacon
(directeur de recherche)

Yoshua Bengio
(membre du jury)

Résumé

Le contenu de cette thèse explore la question de l'attribution de crédits à long terme dans l'apprentissage par renforcement du point de vue d'un biais inductif de parcimonie. Dans ce contexte, un agent parcimonieux cherche à comprendre son environnement en utilisant le moins de variables possible. Autrement dit, si l'agent est crédité ou blâmé pour un certain comportement, la parcimonie l'oblige à attribuer ce crédit (ou blâme) à seulement quelques variables latentes sélectionnées. Avant de proposer de nouvelles méthodes d'attribution parcimonieuse de crédits, nous présentons les travaux antérieurs relatifs à l'attribution de crédits à long terme en relation avec l'idée de sparsité. Ensuite, nous développons deux nouvelles idées pour l'attribution de crédits dans l'apprentissage par renforcement qui sont motivées par un raisonnement parcimonieux : une dans le cadre sans modèle et une pour l'apprentissage basé sur un modèle. Pour ce faire, nous nous appuyons sur divers concepts liés à la parcimonie issus de la causalité, de l'apprentissage supervisé et de la simulation, et nous les appliquons dans un cadre pour la prise de décision séquentielle.

La première, appelée *évaluation contrefactuelle de la politique*, prend en compte les déviations mineures de ce qui aurait pu être compte tenu de ce qui a été. En restreignant l'espace dans lequel l'agent peut raisonner sur les alternatives, *l'évaluation contrefactuelle de la politique* présente des propriétés de variance favorables à l'évaluation des politiques. *L'évaluation contrefactuelle de la politique* offre également une nouvelle perspective sur la rétrospection, généralisant les travaux antérieurs sur l'attribution de crédits a posteriori. La deuxième contribution de cette thèse est un algorithme augmenté d'attention latente pour l'apprentissage par renforcement basé sur un modèle : Latent Sparse Attentive Value Gradients (LSAVG). En intégrant pleinement l'attention dans la structure d'optimisation de la politique, nous montrons que LSAVG est capable de résoudre des tâches de mémoire active que son homologue sans modèle a été conçu pour traiter, sans recourir à des heuristiques ou à un biais de l'estimateur original.

mots clés: attribution de crédits, apprentissage par renforcement, apprentissage basé sur un modèle, attention, contrefactuelle, rétrospection, attribution de crédits à long terme, parcimonie, troncation, Markov Decision Process, évaluation de la politique, apprentissage automatique

Abstract

The content of this thesis explores the question of long-term credit assignment in reinforcement learning from the perspective of a parsimony inductive bias. In this context, a parsimonious agent looks to understand its environment through the least amount of variables possible. Alternatively, given some credit or blame for some behavior, parsimony forces the agent to assign this credit (or blame) to only a select few latent variables. Before proposing novel methods for parsimonious credit assignment, previous work relating to long-term credit assignment is introduced in relation to the idea of sparsity. Then, we develop two new ideas for credit assignment in reinforcement learning that are motivated by parsimonious reasoning: one in the model-free setting, and one for model-based learning. To do so, we build upon various parsimony-related concepts from causality, supervised learning, and simulation, and apply them to the Markov Decision Process framework.

The first of which, called *counterfactual policy evaluation*, considers minor deviations of what could have been given what has been. By restricting the space in which the agent can reason about alternatives, *counterfactual policy evaluation* is shown to have favorable variance properties for policy evaluation. *Counterfactual policy evaluation* also offers a new perspective to hindsight, generalizing previous work in hindsight credit assignment. The second contribution of this thesis is a latent attention augmented algorithm for model-based reinforcement learning: Latent Sparse Attentive Value Gradients (LSAVG). By fully integrating attention into the structure for policy optimization, we show that LSAVG is able to solve active memory tasks that its model-free counterpart was designed to tackle, without resorting to heuristics or biasing the original estimator.

Keywords: credit assignment, reinforcement learning, model-free, model-based, attention, counterfactual, hindsight, long-term credit assignment, parsimony, sparse, discount, truncation, Markov Decision Process, on-policy evaluation, off-policy evaluation, machine learning.

Contents

List of tables

List of figures

Liste des sigles et des abréviations

MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
CA	Credit Assignment
RNN	Recurrent Neural Network
BPPT	Backpropagation through Time
TD	Temporal difference
RL	Reinforcement learning
TBPPT	Truncated backpropagation through time
HCA	Hindsight Credit Assignment
NLP	Natural language processing
TVT	Temporal Value Transport

CIS	Conditional importance sampling
SVG	Stochastic Value Gradient
AD	Automatic differentiation
T-SVG	Truncated stochastic value gradient
PMT	Passive Memory Task
AMT	Active Memory Task
LM-SVG	Latent Memory stochastic value gradient
LSTM	Long short term memory
LSAVG	Latent sparse attentive value gradient

Chapter 1

Introduction

1.1. Artificial Intelligence and Machine Learning

Many of the recent advancements in the field of computer science share a common ambition towards artificial intelligence. While the words "artificial intelligence" can be intuitively understood by most, it can become difficult to put into writing a clear definition of the supposed holy grail of computer science. A large part of this difficulty may be attributed to the vagueness of the term "intelligence", the definition of which has been long debated by psychologists and biologists alike. Perhaps a good place to start is in the Merriam-Webster dictionary which defines ***intelligence****: "the ability to learn or understand or to deal with new or trying situations".*

The ability to learn, as put in the above definition, has been a focal point of artificial intelligence in recent decades. So much so in fact, that since the term "machine learning" was coined in 1952 by Arthur Samuel [**62**], it has not only become synonymous with artificial intelligence, but in 2015, the keyword "machine learning" surpassed "artificial intelligence" in terms of popularity in online searches [**21**].

Interestingly, the ability to learn arguably comprises only one aspect of artificial intelligence. In 1961, an important article entitled *"Steps Towards Artificial Intelligence"* written by Marvin Minsky separates artificial intelligence into five different aspects: Search, Pattern-Recognition, **Learning**, Planning, and Induction. While these different aspects do not have to be mutually exclusive, the idea of machines **learning** is seemingly an important part of artificial intelligence whether you follow an official definition, popularity or Minsky's analysis. What is the difficulty of designing autonomous learning then? *The difficulty is in* ***distributing credit*** *for success of a complex strategy among the many decisions that were involved. -Minsky 1961*

1.2. Credit Assignment

As one of the earliest mention of **the credit assignment problem**, and consequently credit assignment (CA) in general, Minsky's portrayal of credit assignment was tightly associated with reinforcement learning (RL), and how credit is assigned through time. A more recent and detailed take of credit assignment was then discussed in Sutton's thesis [**65**]. The complete definition was written as: *When a learning system employs a complex decision process, it must assign credit or blame for the outcomes to each of its decisions. Where it is not possible to directly attribute an individual outcome to each decision, it is necessary to apportion credit and blame between each of the combinations of decisions that contributed to the outcome.*

While most direct references of credit assignment revolves around reinforcement learning, the discussion around credit assignment has recently become prevalent in many aspects of machine learning. For example, in the popular article [**63**], credit assignment is discussed around Schmidhuber's definition: *Which modifible components of a learning system are responsible for its success or failure? What changes to them improve performance?* In this definition, the credit assignment problem is a fundamental question of not only reinforcement learning, but machine learning and deep learning as a whole. The above formulation is concerned with understanding how changing a specific component (θ) of a system affects the overall performance (L). In other words, accurately estimating $\frac{\Delta L}{\Delta \theta}$ corresponds to credit assignment. Formally, this magic term is called the *gradient* of a system. The question then remains how to best obtain this gradient in increasingly complex learning systems. Today, the most popular method for gradient estimation in deep learning, and consequently credit assignment, is **backpropagation** [**20, 6, 38**]. However, as we will see, backpropagation is far from a perfect solution to the credit assignment problem, especially in long-term scenarios.

Another popular source for understanding credit assignment is in Richard Sutton's Thesis [**65**], where in the context of reinforcement learning, it is separated into two categories: temporal credit assignment and structural credit assignment. We will further examine the details of reinforcement learning later, but for now, the ultimate question in reinforcement learning is to properly assign credit to the internal decision making process of an agent based on a given reward signal. The two categories mentioned by Sutton splits this process into two steps: first credit needs to be assigned to the external decisions made by the agent (temporal credit assignment), then given this richer signal attributed to each individual external decision, credit can be assigned to the internal decisions of the agent that led to the external ones (structural credit assignment). Concretely, learning a better critic is often associated with temporal credit assignment, and structural credit assignment is relegated to gradient estimation techniques such as the REINFORCE estimator [**77**] or backpropagation. Where terminology starts to get confusing is when temporal information is embedded within the

structure of a learning system, such as recurrent neural networks. For this reason, it is more convenient for us to label these two credit assignment categories instead as *secondary credit assignment* (Sutton's temporal credit assignment) and *primary credit assignment* (Sutton's structural credit assignment), and reserve the term *temporal credit assignment problem* for its intuitive definition regarding the problems that may arise when assigning credit through time.

1.3. Reinforcement Learning

Until now, there has been mention of reinforcement learning a few times in the context of credit assignment. In this section, we will provide some basic fundamentals of reinforcement learning. Again let us begin our discussion with a definition: *"[Reinforcement learning] is a computational approach to learning whereby an **agent** tries to maximize the total amount of reward it receives while interacting with an [...] **environment**"* [**67**]. We can expand our definition further by fleshing out some of the key words highlighted in the above citation. In fact, Markov Decision Processes (MDP) provide us with a framework for reinforcement learning that goes well beyond words.

1.3.1. Markov Decision Processes

The agent in this context is not unlike any function mapping familiar to the machine learning literature. It takes as input some observation of the current state of the environment, and outputs an action, or a probability of actions to take in the environment. Often also called a policy, it is denoted as: $\pi(s) \rightarrow a$, where s is the current state of the environment, and a is the action chosen by the agent.

A large part of what makes reinforcement learning separate from other fields of machine learning, such as supervised or unsupervised learning, is in the environment. So far, we have used terms like environment, state, action, and interact without properly defining them. Markov Decision Processes [**57**] provide a template to ground these terms formally. An MDP is characterized by the following tuple: $(\mathcal{A}, S, \mathcal{T},\ \mathbf{r},\ \gamma,\ \mathrm{T})$, where $\mathcal{A}$ is the set of possible actions, $\mathcal{S}$ the set of possible states, $\mathcal{T} \in \mathbb{R}^{|S|\times|\mathcal{A}|\times|\mathcal{S}|}$ the transition probabilities, $\mathbf{r} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ the reward function, $\gamma \in [0,1]$ the discount factor and $T \in \mathbb{N}$ the time horizon. Consequently, any reference to the environment can be identified by an MDP and its parameters. In the case when actions and or states are continuous, the transition dynamics can instead be written as a probability distribution $p(s_{t+1}|s_t, a_t)$.

Given the behavior of an agent specified by the stationary policy $\pi(a_t|s_t)$, the environment-agent interactions described with this MDP induces a distribution over trajectories (called episodes) as follows:

The initial state is drawn: $s_0 \sim \mu$

for $t = 0$ **to** T **:**

An action is chosen by the agent: $a_t \sim \pi(a_t|s_t)$

The agent interacts with the environment to observe the next state and reward:

$r_t \leftarrow r(s_t, a_t)$, $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$

Importantly, environments are said to be Markovian if s_t encompasses all past information needed for future predictions. If this is not the case, we typically enter the world of Partially Observable Markov Decision Processes (POMDP) [**83, 34**], where states emitted by the environment are instead called observations o_t. When this is the case, a naive way to preserve the Markovian property is to simply consider the entire history of observation-action pairs as the current state, $s_t = (o_0 a_0, \ldots, a_{t-1} o_t)$. Alternatively, it is possible to maintain a belief state $p(s_t|o_0 a_0, \ldots, a_{t-1} o_t)$ that equally encompasses all the information needed [**44**].

Under the framework of MDPs, reinforcement learning seeks to learn a policy that maximizes the cumulative reward over an episode. Value iteration and policy iteration [**67**] are clear ways to do just this, but may lack efficiency in more complex environments.

1.4. Thesis Statement

In this work, we will investigate the credit assignment problem in the context of reinforcement learning. Specifically, we investigate the effects of using parsimony as an inductive bias for better temporal credit assignment. Parsimonious, or sparse, reasoning relates to state abstraction or the options framework [**2, 69**]. As the saying goes "*less is more!*", the parsimony bias is one that argues that agents should be able to reason just as well, if not better, using only a select few variables in its latent space. The inspiration for parsimonious reasoning comes from humans, and is in line with Bengio's Consciousness Prior [**7**] and the Global Workspace Theory of conscious processing [**3**].

Chapter 2

Background and Related Work

2.1. Learning Across Time

Machine learning is built around the notion of a loss or performance objective being optimized. In the *supervised learning* setting, the metric, called a loss function $L(x,y,\theta)$, measures the performance of a model with respect to the model parameters θ, inputs x, and true values y. How can we know how to change the parameters in order to minimize the loss? Luckily, that is exactly what gradients tell us. According to he first-order optimality condition for unconstrained optimization, if θ is a minimizer f L at x,y, then $\frac{dL(x,y,\theta)}{d\theta} = 0$. Using this fundamental result, gradient descent (or ascent) is an iterative procedure which takes steps in the direction of the gradient until a local minimum (or maximum) is attained. Attaining the derivative $\frac{dL(x,y,\theta)}{d\theta}$ tells us in what direction to modify θ in order to maximize or minimize $L(x, y,\theta)$. Fortunately, there are many tools to our disposal for calculating derivatives, however with the advent of more complicated systems like artificial neural networks, more efficient means for gradient computation were necessary. Reverse mode automatic differentiation (AD) [**43, 24**], or backpropagation, serves just this purpose, and is the backbone of most modern machine learning for efficient gradient computation in said complex function approximators.

2.1.1. Backpropagation

The origins of backpropagation is somewhat disputed [**23**], but the methodology is clear. Without loss of generality, let us consider the function $f(x,\theta)$ composed of however many individual differentiable functions. For example, if the function can be written as $f(x,\theta) = f_n \circ f_{n-1} \circ \ldots \circ f_2 \circ f_1(x,\theta_1)$, where each f_i is parameterized by θ_i, the derivative $\frac{df}{d\theta_i}$ can be obtained with the chain rule:

$$\frac{df}{d\theta_i} = \frac{df_n}{df_{n-1}}\frac{df_{n-1}}{df_{n-2}}\cdots\frac{df_{i+1}}{df_i}\frac{df_i}{d\theta_i} \tag{2.1.1}$$

Notice the recursive nature of the above formulation, which can also be written as:

$$\delta_i := \frac{df_n}{df_{n-1}}\frac{df_{n-1}}{df_{n-2}}\dots\frac{df_{i+1}}{df_i}$$
$$\delta_{i-1} = \delta_i\frac{df_i}{df_{i-1}}$$
$$\frac{df}{d\theta_i} = \delta_i\frac{df_i}{d\theta_i}$$

Let us quickly examine the computational cost of the above operations. The Jacobians $\frac{df_i}{df_{i-1}}$ are of shape $N_i \times N_{i-1}$ where N_i is the output size of f_i. Recall that $f(x, \theta)$ is a loss function that takes a scalar value in a vast majority of machine learning applications, therefore $N_n = 1$. Additionally, the jacobian $\frac{df_i}{d\theta_i}$ is of shape $N_i \times |\theta_i|$ where $|\theta_i|$ is the number of parameters in f_i. Given the associative nature of matrix multiplications, there are a number of ways to go about computing $\frac{df}{d\theta_i}$. Naturally, there are two extreme opposite methods, that is going about equation 2.1.1 from right to left, and from left to right, where naively multiplying a matrix of shape $N \times M$ by one of $M \times P$ requires NMP atomic operations. For the remaining discussion regarding computation and memory cost, reference to θ_i alludes to the number of parameters in f_i, $|\theta_i|$.

Right to left corresponds finding the gradient as $\frac{df}{d\theta_i} = \frac{df_n}{df_{n-1}}(\frac{df_{n-1}}{df_{n-2}}(\dots(\frac{df_{i+1}}{df_i}\frac{df_i}{d\theta_i})))$. This involves maintaining a matrix of shape $(.) \times \theta_i$ until reaching the desired matrix of shape $N_n \times \theta_i$. The following number of operations are necessary

$$N_{i+1}N_i\theta_i + N_{i+2}N_{i+1}\theta_i + \dots + N_nN_{n-1}\theta_i \ .$$

Left to right corresponds finding the gradient as $\frac{df}{d\theta_i} = (((\frac{df_n}{df_{n-1}}\frac{df_{n-1}}{df_{n-2}})\dots)\frac{df_{i+1}}{df_i})\frac{df_i}{d\theta_i}$. This involves maintaining a matrix of size $N_n \times (.)$ until reaching the desired matrix of shape $N_N \times \theta_i$. The following number of operations are necessary

$$N_nN_{n-1}N_{n-2} + N_nN_{n-2}N_{n-3} + \dots + N_nN_i\theta_i$$

It is clear that if θ_i is much larger than N_j for all j and i, and $N_n = 1$, computing $\frac{df}{d\theta_i}$ from **left to right** following equation 2.1.1 is more computationally efficient. Moreover, the recursive nature of δ_{i-1} lets us efficiently compute $\frac{df}{d\theta_{i-1}}$ using δ_i. These two computational advantages represent the two main benefits of using what is called *reverse mode automatic*

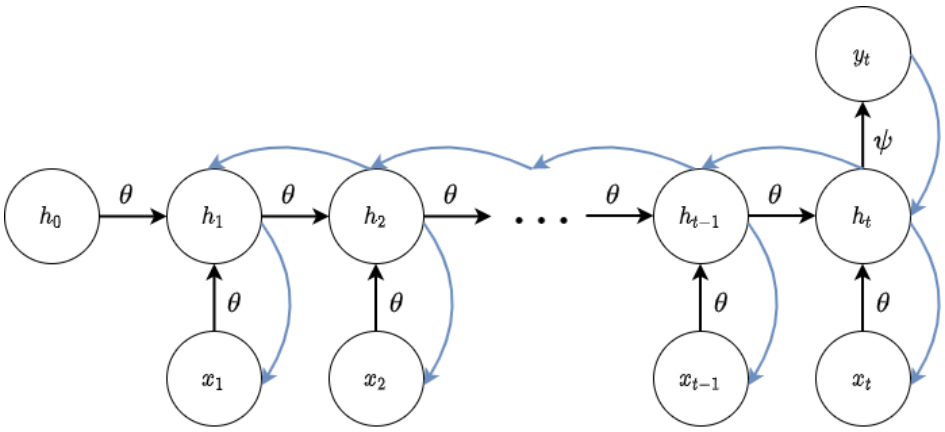

Fig. 2.1. An example of how a typical recurrent neural network computes y_t during the forward pass (represented by the black arrows), and how credit is assigned through time during backward pass (represented in blue).

differentiation [1] or *backpropagation* in machine learning. How can a similar idea be used to assign credit through time?

2.1.2. Recurrent Networks

Originally introduced in [**61**], recurrent networks are an extension of traditional multilayer perceptrons [**58**] to the problem of predicting a sequence of inputs. Given a sequence of data $s_t = x_1 x_1 \dots x_t$, a basic recurrent neural network (RNN) is parameterized as $g(s_t, \theta, \psi) = f_\psi \circ f(x_t, \theta) \circ f(x_{t-1}, \theta) \circ \dots \circ f(x_1, \theta)$, where $f(.)$ behaves similarly to $f(x,\theta)$ in the previous section. An RNN iteratively computes:

$$\begin{aligned} h_0 &:= \mathbf{0} \\ h_t &:= f(x_t, h_{t-1}, \theta) \\ y_t &:= f_\psi(h_t, \psi) \end{aligned} \tag{2.1.2}$$

Even though f is a function of itself through time, the general procedure for backpropagation can still be applied for gradient calculations. Again, we are interested in finding $\frac{df}{d\theta}$

[1] Why is it called *reverse mode automatic differentiation* if it appears to compute equation 2.1.1 in the standard left to right direction, and what would be the alternative *forward mode automatic differentiation*? Recall that in order to calculate the gradients, a forward pass through the function needs to be done to obtain the values $f_n, f_{n-1}, f_{n-1}, \dots, f_{i+1}, f_i$. Due to the compositional nature of f_n, all previous intermediate values $f_{j<n}$ need to be computed before doing the first matrix multiplication $\frac{df_n}{df_{n-1}} \frac{df_{n-1}}{d_f n-2}$, however the intermediate results $f_{j<n}$ will be used later during backpropagation. Therefore, the intermediate values must be saved during the forward pass, resulting in the use of a memory of size $N_i + \dots + N_{n-1}$ for reverse mode autodiff. The alternative, *forward mode automatic differentiation*, which computes equation 2.1.1 from right to left, aligns its gradient computation with the forward pass, which means intermediate results $f_{j<n}$ can be discarded as you go, saving you any memory hassle, hence the terms *forward* and *backward* differentiation. The (general) computational benefits of backpropagation are therefore not free, and comes with a corresponding memory tradeoff.

for a given sequence of data s_t. We can unroll the forward computations, and apply the chain rule to see what $\frac{df}{d\theta}$ is really composed of:

$$\begin{aligned}\frac{dh_0}{d\theta} &:= \mathbf{0} \\ \frac{dh_t}{d\theta} &:= \frac{df(x_t, h_{t-1}, \theta)}{d\theta} \\ \frac{dh_t}{d\theta} &= \frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{\partial h_{t-1}} \frac{dh_{t-1}}{d\theta}\end{aligned} \tag{2.1.3}$$

The gradient of interest for backpropagation through time can also be written recursively as

$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial \theta} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_i}{\partial \theta} \ . \tag{2.1.4}$$

Similarly to our previous discussion of backpropagation, we can compute equation 2.1.4 using *reverse mode automatic differentiation* for computational benefits[2]. Recurrent networks are built using the ideas of standard neural networks with a time component. Consequently, backpropagation through time (BPPT) naturally extends credit assignment into the temporal dimension.

2.2. Assigning Credit Through Experience

As discussed earlier, the agent-environment interactions of reinforcement learning lends itself to a slightly different problem class than the above credit assignment paradigms. Supervised learning is interested in changing some parameters θ that only emit a structural dependency with respect to the loss to minimize. Conversely, the parameters of RL emit a distributional dependency with respect to the performance measure to maximize [**60**].

Definition 2.2.1 (Expected Value)**.** *The expected value of a continuous random variable X with a probability density function $f(x)$ is*

$$\mathbb{E}[X] = \int x f(x) dx \ .$$

It then follows that if the probability density function of X is parameterized by θ as $p_\theta(x)$, and let $h(X)$ be any function of X, then

$$\mathbb{E}_\theta[h(X)] = \int h(x) p_\theta(x) dx \ .$$

[2]There is a forward mode equivalency, called Real-time recurrent learning which inherits all the benefits and drawbacks of forward mode differentiation as discussed in the previous section. However, because the situations in which t gets very large in recurrent networks tend to be much more common and unavoidable compared to the cases when n is very large in standard neural networks, there has been a lot more work discussing the potential uses of RTRL [**78, 48**]

In general, credit assignment in reinforcement learning involves sampling experiences, X, from the environment with the agent, paramterized by θ, and maximizing an expected performance measure $h(X)$ with respect to the sampled experiences. The interest is therefore in estimating the following gradient[3]:

$$\nabla_\theta \mathbb{E}_\theta \left[h(X)\right] = \nabla_\theta \int h(x) p_\theta(x) dx \ . \tag{2.2.1}$$

For practical reasons, we would the gradient in equation 2.2.1 to be inside the integrand such that $\nabla_\theta \mathbb{E}_\theta \left[h(X)\right]$ can be estimated from samples of X. The different tricks used to estimate this gradient has subsequently given rise to two main categories of reinforcement learning; model-free learning, and model-based learning.

Under the MDP framework, X represents the sequence of states and actions, the performance measure is the accumulated reward, and $p_\theta(X)$ is the probability distribution of the sequence of state-action pairs induced by the current policy and the dynamics:

$$\begin{aligned} X &:= S_0, A_0, S_1, A_1, \ldots, S_T, A_T \ , \\ G_T &:= \sum_{t=0}^{T} r(S_t, A_t) \\ h(X) &:= \sum_{t=0}^{T} r(S_t, A_t) \ , \\ p_\theta(X) &:= \mu(S_0) \prod_{t=0}^{T} \pi_\theta(A_t|S_t) p(S_{t+1}|S_t, A_t) \ , \end{aligned} \tag{2.2.2}$$

where capital letters denote random variables of the environment. In this case, S_i is the state observed at time step i, A_i is the action observed at time step i and G_T is the accumulated reward.

2.2.1. Model-free Learning

Model-free learning estimates equation 2.2.1 without the need to know the environments dynamics $p(s_{t+1}|s_t, a_t)$. The basic building blocks of most model-free learning approaches follow from the REINFORCE estimator derived in [**77**] and the policy gradient theorem first shown in [**68, 5, 37**]. We will show here a derivation of the REINFORCE estimator starting from (2.2.1) using likelihood ratios and score functions.

Likelihood ratios are most commonly used in the simulation community for importance sampling. At its core, likelihood ratios can be used in importance sampling [**60**] for a *change of measure* [**12**], that is we can measure $\mathbb{E}_\theta \left[h(X)\right]$ under a different probability density

[3]The discussion so far is working under the assumption of continuous state spaces. For simplicity, we omit the discrete counterpart.

function $q(X)$ through a simple multiplication:

$$\mathbb{E}_\theta\left[h(X)\right] = \mathbb{E}_q\left[h(X)\frac{p_\theta(X)}{q(X)}\right] \tag{2.2.3}$$

$$\int h(x)p_\theta(x)dx = \int h(x)\frac{p_\theta(x)}{q(x)}q(x)dx \ . \tag{2.2.4}$$

The resulting ratio $\frac{p_\theta(x)}{q(x)}$ is called a likelihood ratio. Importantly, $h(x)p_\theta(x)$ must be *dominated* by the new distribution $q(x)$. That is, $q(x) = 0 \implies h(x)p_\theta(x) = 0$.

Score functions are defined as the gradient of the log likelihood of a probability distribution. Often used in statistics for its various properties, it intuitively measures how a small change of θ will affect the the overall probability distribution relative to the original distribution [**64**]. This can be seen through the following identity given by calculus:

$$\frac{\partial}{\partial x}\log f(x) = \frac{\nabla_x f(x)}{f(x)} \ . \tag{2.2.5}$$

Equipped with these concepts, we are now ready to see how to evaluate $\nabla_\theta\mathbb{E}_\theta\left[h(X)\right]$ (equation 2.2.1) purely through experience. First, we use the likelihood ratio to rewrite the aforementioned gradient with respect to a new distribution $q(x)$, $\nabla_\theta\mathbb{E}_\theta\left[h(X)\right] = \nabla_\theta\mathbb{E}_q\left[h(X)\frac{p_\theta(X)}{q(X)}\right]$. Second, the gradient operator can now be put inside the expectation, since it no longer depends on θ, $\nabla_\theta\mathbb{E}_\theta\left[h(X)\right] = \mathbb{E}_q\left[h(X)\frac{\nabla_\theta p_\theta(X)}{q(X)}\right]$. Finally, let us use the score function of $p_\theta(x)$, that is let $q(x) = p(x)$:

$$\nabla_\theta\mathbb{E}_\theta\left[h(X)\right] = \mathbb{E}_\theta\left[h(X)\nabla_\theta\log p_\theta(X)\right] \ . \tag{2.2.6}$$

Aside from notation, this is how REINFORCE estimates the gradient of the performance measure through experience, or in other words, how credit can be assigned through experience. For completeness, we rewrite the above equation using the MDP framework defined in equations 2.2.2. Particularly, it is worth addressing how the score function induced by $p_\theta(X) = \mu(S_0)\prod_{t=0}^{T}\pi_\theta(A_t|S_t)p(S_{t+1}|S_t, A_t)$ can be reduced:

$$\frac{\nabla_\theta p_\theta(X)}{p_\theta(X)} = \frac{\nabla_\theta\mu(S_0)\prod_{t=0}^{T}\pi_\theta(A_t|S_t)p(S_{t+1}|S_t, A_t)}{\mu(S_0)\prod_{t=0}^{T}\pi_\theta(A_t|S_t)p(S_{t+1}|S_t, A_t)} \ .$$

The initial distribution $\mu(S_0)$ and transition probabilities $p(S_{t+1}|S_t, A_t)$ can be canceled out because they are independent of θ.

$$\frac{\nabla_\theta p_\theta(X)}{p_\theta(X)} = \frac{\nabla_\theta\prod_{t=0}^{T}\pi_\theta(A_t|S_t)}{\prod_{t=0}^{T}\pi_\theta(A_t|S_t)}$$

Then, we use the identity in (2.2.5) to rewrite it in terms of the log probabilities, which lets us change the product into a summation,

$$\frac{\nabla_\theta p_\theta(X)}{p_\theta(X)} = \sum_{t=0}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t) \ .$$

Now, putting everything together recovers the REINFORCE estimator in (Williams 1992).

$$\nabla_\theta \mathbb{E}_\theta \left[\sum_{t=0}^{T} r(S_t, A_t) \right] = \mathbb{E}_\theta \left[\sum_{t=0}^{T} r(S_t, A_t) \sum_{k=0}^{t} \nabla_\theta \log \pi_\theta(A_k|S_k) \right] \tag{2.2.7}$$

$$= \mathbb{E}_\theta \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) \sum_{k=t}^{T} r(S_k, A_k) \right] \tag{2.2.8}$$

Note the switch in summation in equation 2.2.8 with which REINFORCE was first derived with, and the form most current algorithms are based on. Many popular model-free reinforcement learning algorithms, such as DQN[**50**], DDPG[**42**], TD3 [**18**], SARSA [**67**], are derived using this formulation as a starting point, often replacing the cumulative reward for value functions [4].

2.2.2. Model-based Learning

Model-based learning takes another approach to credit assignment. As the name implies, using a known model of the dynamics, we can similarly assign credit with experience even if the agent or environment may be stochastic. Using what many in the machine learning community now call the **reparameterization trick**, we can move the stochasticity of the original estimator elsewhere such that the gradient can be pushed inside the expectation.

The **reparameterization trick**[5] recently popularized through variational auto-encoders [**36**] is based on a *change of variable* instead of the *change of measure* used in model-free learning. The goal of reparameterizing is to write a random variable as a deterministic function with another random variable as input. In doing so, the new expectation is instead conditioned on the new random variable. Take for example a random variable drawn from a normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$. The random variable X can instead be written as a deterministic function of the new random variable $\xi \sim \mathcal{N}(\mu, \sigma^2)$: $x = f(\xi) = \mu + \sigma\xi$.

Let us see how equation 2.2.1 can be estimated using this trick. First, assume a deterministic function $f(\theta, \xi)$, such that the random variable X becomes has the deterministic relationship $x = f(\theta, \xi)$. Moreover, ξ is sampled from a known fixed distribution $\xi \sim p(\xi)$. For example, it could the uniform distribution $p(\xi) = \mathcal{U}(0,1)$. The desired gradient then becomes:

$$\begin{aligned} \nabla_\theta \mathbb{E}_\theta \left[h(X) \right] &= \nabla_\theta \int h(x) p_\theta(x) dx \\ &= \nabla_\theta \int h(f(\theta, \xi)) p(\xi) d\xi \\ &= \int \nabla_\theta h(f(\theta, \xi)) p(\xi) d\xi \qquad = \mathbb{E}_{\xi \sim p(\xi)} \left[\nabla_\theta h(f(\theta, \xi)) \right] \end{aligned} \tag{2.2.9}$$

[4] Learning value functions, or policy evaluation, will be addressed in a later section.
[5] Also known as *infinitesmal perturbation analysis* in the simulation community [**39**]

A necessary first to estimating the above gradient is have access to $f(\theta, \xi)$. In the context of reinforcement learning, this amounts to having the reparameterized dynamics of the model $s_t = f(s_{t-1}, a_{t-1}, \xi)$ and the reparamterized policy $a_t = \pi(s_t, \xi_\pi)$. In this case, equation 2.2.9 can be written as:

$$\mathbb{E}_{p(\xi)}\left[\sum_{t=0}^{T} \nabla_\theta r(s_t,a_t)\right] \text{ , where} \tag{2.2.10}$$
$$s_t = f_\psi(s_{t-1}, a_{t-1}, \xi)$$
$$a_t = \pi_\theta(s_t, \xi)$$

Given that $r(s_t, a_t)$, $f_\psi(s_{t-1}, a_{t-1}, \xi)$ and $\pi_\theta(s_t, \xi)$ are all deterministic thanks to the reparameterization trick, it is straight forward to derive the gradient $\nabla_\theta r(s_t,a_t)$ using backpropagation through time. The interesting parallel between recurrent networks and model-based learning will be discussed later in this work.

Most of the derivation in this section for model-based learning is based on stochastic value gradients (SVG) [**30**], and does not generalize all model-based learning algorithms. The focus of this section is to introduce background for what are known as **value gradient** methods [**30, 15**]. Other model-based methods exist based on planning [**66, 25, 26**] may also make use of the reparameterization trick, but do not necessarily backpropagate through the learned model for credit assignment. For the purposes of this thesis, any reference to model-based learning references value gradient based methods as shown in this section.

2.3. Long-term Credit Assignment

How well do the above methods for gradient estimation scale over long horizons? For reasons that will become clear, the basic approaches previously highlighted are not ideal when assigning credit through a long period of time. We will see how the horizon T of an environment or problem affects the behaviors of computing gradients in both supervised learning and reinforcement learning.

2.3.1. Supervised Learning

Here, we discuss the challenges of applying backpropagation in the supervised learning setting given long-time scales or deep networks. The main challenge, infamously known as *the vanishing/exploding gradient problem* [**20, 53**] is particularly problematic in recurrent neural networks given their depth.

Recall the problem statement given in equation 2.1.2. For this discussion, let us simplify h_t to admit only a recurrent linear dependency :

$$\begin{aligned} h_0 &:= \mathbf{0} \ , \\ h_t &:= \mathbf{W}^T h_{t-1} \ , \\ \textbf{find} \ & \frac{dh_t}{d\mathbf{W}} \ , \end{aligned} \tag{2.3.1}$$

where $\mathbf{W} \in \mathbb{R}^{N \times N}$ and $h_0 \in \mathbb{R}^N$. Given the iterative unrolling of $\frac{dh_t}{d\mathbf{W}}$ shown in equation 2.1.4 or the recurrent relationship in (2.1.3), the effect of changing the parameters in the long term, say at time step $i < t$, are captured by :

$$\begin{aligned} \frac{\partial h_t}{\partial \mathbf{W}_i} &= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \mathbf{W}_i} \\ &= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{i+1}}{\partial h_i} \frac{\partial h_i}{\partial \mathbf{W}_i} \ . \end{aligned}$$

Following the relationship established in (2.3.1), the terms $\frac{\partial h_k}{\partial h_{k-1}}$ for all $i < k \leq t$ can be reduced to

$$\frac{\partial h_k}{\partial h_{k-1}} = \mathbf{W}^T \ .$$

The long term effect of changing the parameters at time step $i < t$ can therefore be found to be

$$\frac{\partial h_t}{\partial \mathbf{W_i}} = (W^T)^{(t-i)} \frac{\partial h_i}{\partial \mathbf{W}_i} \tag{2.3.2}$$

Before we are ready to formally show *the vanishing/exploding gradient problem*, a couple definitions are necessary.

Definition 2.3.1 (Eigenvalues and Eigenvectors)**.** *Let $A \in \mathbb{R}^{m \times m}$ be a square matrix. For any $\mathbf{v} \neq \mathbf{0}$ such that $A\mathbf{v} = \lambda \mathbf{v}$ for some $\lambda \in \mathbb{C}$, $\mathbf{v}$ is called an **eigenvector** corresponding to the **eigenvalue** λ.*

Definition 2.3.2 (Diagonalizable matrices, and Eigendecompositions)**.** *A matrix $A \in \mathbb{R}^{m \times m}$ is **diagonalizable** if and only if there exists a basis $v_1, v_2, \ldots, v_m \in \mathbb{R}^m$ of $\mathbb{R}^m$ consisting of eigenvectors of A. In which case, A has the **eigendecompositon**:*

$$A = Q \Lambda Q^{-1} \ ,$$

where $Q \in \mathbb{R}^{m \times m}$ consists of the eigenvectors of A and $\Lambda \in \mathbb{R}^m$ consists of the corresponding eigenvalues.

Theorem 2.3.3 (The vanishing/exploding gradients problem)**.** *Given a recurrent neural network defined in* (2.3.1)*, the long term effects of changing the parameters in $i < t$ exhibit an exponential relationship with respect to the eigenvalues of W and $t - i$.*

Proof. Given (2.3.1) and let $\mathbf{W}$ be diagonalizable, then

$$\begin{aligned}\frac{\partial h_t}{\partial \mathbf{W}_i} &= (\mathbf{W}^T)^{t-i}\frac{\partial h_i}{\partial \mathbf{W}_i} \\ &= (Q^{-1}\Lambda Q)^{t-i}\frac{\partial h_i}{\partial \mathbf{W}_i} \\ &= Q^{-1}\Lambda^{t-i}Q\frac{\partial h_i}{\partial \mathbf{W}_i}\end{aligned}$$

□

The implication of theorem 2.3.3 being that unless the eigenvalues of $\mathbf{W}$ are very close to 1, any long term effects captured through BPPT quickly reduce to 0 if the eigenvalues are less than 1, and explode if the eigenvalues are more than 1. Long term credit assignment in vanilla recurrent networks suffer from this, and it still remains a hot topic of research to this day [**32, 53, 3, 20**]. Some common solutions to this include specially crafted gated RNNs [**32**], gradient clipping [**20**] and, as we will see, gradient truncation.

2.3.2. Actor-Critic Methods

There are two main components to the REINFORCE estimator shown in equation 2.2.8. The first of which is the score function estimator, and the second component evaluates the performance of this policy with respect to the action taken. Put simply, the first part tells us how changing the internal decisions θ affects the external decision $\pi_\theta(a_t|s_t)$. This is the primary credit assignment problem (Sutton's structural credit assignment) introduced in the introduction. The second part is the secondary credit assignment problem (Sutton's temporal credit assignment), and is often called a *policy evaluation* step. Policy evaluation looks to estimate the following values:

$$V_t^\theta(s) := \mathbb{E}_\theta\left[\sum_{t=t}^{T} r(S_t, A_t)\middle| S_t = s\right] \quad , \tag{2.3.3}$$

$$Q_t^\theta(s,a) := \mathbb{E}_\theta\left[\sum_{t=t}^{T} r(S_t, A_t)\middle| S_t = s, A_t = a\right] \quad . \tag{2.3.4}$$

Particularly, $Q_\theta(s,a)$, called the action-value function [**67**], can be used within the REINFORCE estimator to provide an estimate of the expected return along the state-action pairs of a trajectory:

$$\nabla_\theta \mathbb{E}_\theta\left[\sum_{t=0}^{T} r(S_t, A_t)\right] = \mathbb{E}_\theta\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) Q_t^\theta(S_t, A_t)\right]$$

In its simplest form, the one used in (2.2.8), Q is estimated using a one-sample crude Monte Carlo estimate. The variance of this estimate is therefore very high, especially when T is very large. In the case of $T = \infty$, the variance can even become unbounded.

There are various ways to address this issue, the most popular of which is to learn a function to approximate Q, called a critic. Still, true values need to be approximated for the critic to ground itself. Temporal difference learning (TD) [**65**] serves just this purpose, offering a middle ground between full Monte Carlo estimates and dynamic programming. A detailed account of TD predictions can be found in [**65**] and [**67**]. In summary, TD learning trades variance for bias by bootstrapping the value function. However temporal difference predictions are also affected by the long-term credit assignment problem, as the bias of TD also increases with the horizon. Another popular practice for long-term credit assignment is to employ a recency heuristic [**5, 65, 67**], and discount future rewards. We will address this approach in more detail in the next section.

2.3.3. Model-based Reinforcement Learning

Model-based RL does not suffer the same variance issue as model-free RL, due to its implicit modeling of dependencies. However, since the model is typically learned during training, any error in the model is propagated through time. Since the model is used at every step of credit assignment, an error e in the dynamics model is accumulated through time. Again, as T grows, errors in the model are accentuated for long-term credit assignment, and become increasingly unreliable. Similarly to model-free learning, one can make use of a critic to prevent the learned transition model to propagate its errors through time. However, just like model-free learning, policy evaluation comes with its own issues for long-term credit assignment.

We will also see that pure model-based reinforcement learning with no critic makes extensive use of backpropagation through time. Consequently, the *vanishing/exploding gradient problem* also plagues model-based RL.

2.4. Parsimony for Better Reasoning

Seeing the challenges of long-term credit assignment, let us explore further some of the solutions in the current literature, and how they might relate to parsimonious reasoning. We have already motivated the use of parsimony for credit assignment for biological and psychological reasons, but this remains a heuristic. Generally, sparse reasoning can also be seen as a way to artificially reduce the horizon T of our credit assignment problem. Whether by changing time scales, or by ignoring a certain portion of the time steps, lowering the value of T addresses most of the challenges associated with long-term credit assignment discussed in the previous subsection at the cost of introducing bias.

2.4.1. Discounting and Truncation

The easiest method for handling long-term credit assignment is to just heuristically cut the horizon to a lower value. In fact, truncating or discounting the future (or past) is a common practice in both recurrent networks and reinforcement learning [**14, 79, 67**]. So much so, that the reinforcement learning literature has started considering the *discount factor* as part of the environment parameters, rather than a hyper-parameter used to control the horizon. In any case, the horizon is reduced following a recency heuristic, operating under the hypothesis that events which are temporally near each other are more likely causally correlated. While this form of local credit assignment is logically sound, it makes learning long-term dependencies difficult, and biases the overall gradient estimates.

Consider again the performance measure $h(X)$. As defined in reinforcement learning in equation 2.2.2, the discounted-MDP takes an additional parameter, the discount factor $\gamma \in [0,1)$, and includes it in $h(X)$ as

$$h(X) := \sum_{t=0}^{T} \gamma^t r(S_0, A_0) \ .$$

The advantages are particularly meaningful in infinite horizon problems, effectively turning those into finite horizon ones all the while conserving convergence guarantees (1995 Bertsekas). Practically, we define the *effective horizon* of a discounted-MDP as $\frac{1}{1-\gamma}$. The motivation for this definition follows from the closed form solution to a geometric series.

Theorem 2.4.1 (Geometric Series)**.** *Given $a \in \mathbb{R}$ and $-1 < \gamma < 1$, then:*

$$\sum_{t=0}^{\infty} a\gamma^t = \frac{a}{1-\gamma}$$

Proof. The proof follows by calling $S := \sum_{t=0}^{\infty} a\gamma^t$, multiplying both sides with γ, and subtracting S from both sides. □

Additionally, the discount factor in REINFORCE-like estimators is separately introduced in GPOMDP [**5**] to account for memory and variance issues in the estimator for infinite horizon problems. In their work, they show a natural bias-variance tradeoff with respect to the choice of γ, where the variance roughly grows with $\frac{1}{1-\gamma}$ and the bias decreases as a function of $\frac{1}{1-\gamma}$.

The effects of a discount factor in the model-based setting is not as well understood. However, one can make the logical connection that if γ effectively reduces the horizon of a problem, the issues of long-term credit assignment in model-based learning discussed in the previous subsection should be mitigated. Instead, a common practice in computing value gradients is to truncate the gradients [**30**], similarly to gradient truncation in recurrent neural networks. Truncated backpropagation through time (TBPPT), first introduced around 1990 [**14, 79**], allows for a practical use of BPPT in excessively long recurrent relationships.

TBPPT works by removing any recurrent relationships that are n steps away from the current time step. In other words, even though forward computation remains the same, the backward pass pretends that h_{t-n} is an independent constant. Concretely, the recursive relationship for the gradients in TBPPT can written as:

$$\begin{aligned} \frac{dh_{t-n}}{d\theta} &:= \mathbf{0} \ , \\ \frac{dh_t}{d\theta} &:= \frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{h_{t-1}} \frac{dh_{t-1}}{d\theta} \ . \end{aligned} \tag{2.4.1}$$

The memory and stability benefits are clear, since we are effectively replacing the horizon T with $n < T$, and both the memory costs and stability issues associated with the vanishing/exploding gradients problem are related to the size of T. Of course, this comes at the large cost of being unable to assign credit to any time step further than n steps away.

Both reward discounting, and gradient truncation are crude examples of a parsimony bias widely adopted in the current literature to address some of the issues regarding long-term credit assignment. However, such practices are far from ideal, and present more of a practical way to avoid long-term credit assignment, rather than a solution to it.

2.4.2. Counterfactuals

The term *counterfactual* has a rich history within causal theory. In this world, counterfactuals adheres to a strict definition useful for causal discovery and inference. There are recent accounts of using strict causal counterfactuals in reinforcement learning [**16, 4, 17, 82**], but this series of work remain closer to causal theory than to reinforcement learning. Alternatively, another line of work in counterfactual, or hindsight, reinforcement learning [**28, 11, 49**] has also gained traction recently that is much more familiar to the RL literature. A larger focus will be put on the latter, but we will give a brief overview of counterfactuals in causality to gain a better understanding of its intuitive benefits and motivation.

One of the first detailed accounts of counterfactuals for causation was in [**41**], where Lewis says : "*We think of a cause as something that makes a difference, and the difference it makes must be a difference from what would have happened without it. Had it been absent, its effects [...] would have been absent as well*". Later, Judea Pearl would introduce a mathematical framework for causal reasoning that would change the way many interact with causal discovery. The framework, introduced in [**54**], called Do-calculus, would also provide insight on a hierarchy of causal information: Pearl's causal hierarchy [**?**]. In this theory, causal inference would be separated into three layers: observational, interventional, and counterfactual. These layers emit a natural ordering of information, with counterfactual being the strongest form of causal inference, and observational being the weakest. The three layers can be better understood using Table 2.1 and Figure 2.2. In general, counterfactuals of this form can be evaluated in a three step process:

Layer	Notation	Question	Example
Observational	$P(Y = y\|X = x)$	How does seeing x affect my belief in y	I saw someone take an aspirin, will his headache be cured?
Interventional	$P(Y = y\|do(X = x))$	How will doing x affect y	If I take an aspirin, will my headache be cured?
Counterfactual	$P(Y_x = y\|X = x', Y = y')$	Given that doing x' led to y', how would doing x instead have affected y?	Not taking an aspirin did not cure my headache, keeping all things equal, would taking an aspirin have cured my headache?

Table 2.1. The different layers of causal inference, their mathematical notation, and examples. Note that a subscript denotes a *do* operation. Therefore $P(Y_x = y) = P(Y = y|do(X = x)$.

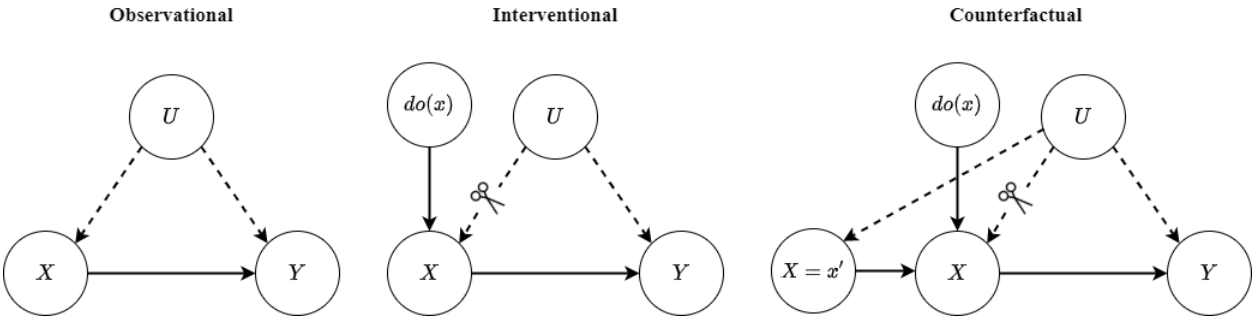

Fig. 2.2. Visualisation of the structural causal model of the various layers of PCH. In the observational setting, exogenous variables U may counfound the two variables in question. In the interventional setting, we cut any parents of X, therefore avoiding any potential confounders. In the Counterfactual setting, *abduction* lets us make an inference over U, thus truly understanding if $do(x)$ causes Y, as opposed to a possible exogenous variable affecting it.

(1) *Abduction:* Update the probability $P(u)$ conditioned on e: $P(u|e)$, where u are any exogenous variables.
(2) *Action:* Intervene, performing a $do(x)$ operation to the causal model
(3) *Prediction:* Use the modified intervened model and better understanding of u to predict $P(Y_x = y|e)$.

The benefit of counterfactuals over the interventional layer is in being able to keep any exogenous variables the same, and reasoning only on the effects of X on Y. In other words, if I observed $Y = y'|do(X = x')$, counterfactuals would be able to answer what would happen to Y if i did something else $do(X = x)$ in the exact same scenario. Following Pearl's causal hierarchy, this form of parsimonious reasoning is the ultimate form of causal inference.

In reinforcement learning, the idea of predicting the outcome of other actions beyond the one actually taken [**56, 44**], vaguely resembling the form counterfactuals take in causality. Off-policy methods [**56**] using importance sampling [**60**] is a popular method addressing just this question. However, such off-policy methods have actually been shown to be an instance of **interventional** inference from Table 2.1. Another approach to **counterfactual** inference in RL is to use **hindsight** [**28, 49**].

The relationship of counterfactuals with hindsight is one that has been exploited in recent work regarding hindsight credit assignment [**28, 49, 11**] for reinforcement learning. At the core of these works is the novelty of conditioning some of the familiar reinforcement learning estimators with the future. Specifically, work in hindsight credit assignment postulates that the value functions, in the form of equation 2.3.3 or 2.3.4 can benefit from being conditioned on some future statistic ϕ_t. Counterfactuals then arise when considering an alternative course of events based on new information retrospectively or in hindsight. The future conditioned value functions can be written as:

$$V_t^\theta(s|\phi_t) := \mathbb{E}_\theta \left[G_\pi \middle| S_t = s, \phi_t \right] \quad , \tag{2.4.2}$$

$$Q_t^\theta(s,a|\phi_t) := \mathbb{E}_\theta \left[G_\pi \middle| S_t = s, A_t = a, \phi_t \right] \quad , \tag{2.4.3}$$

where G_π denotes the return following the policy π. The work in hindsight credit assignemnt (HCA) [**28**] show convincing empirical evidence supporting the use of hindsight conditioned policy evaluation, and even some theoretical evidence with regards to the variance benefits. We will show in this work a deeper treatment of hindsight credit assignment algorithms, offering a new perspective into the role of hindsight in credit assignment.

2.4.3. Attention

Ever since its introduction in 2014, the attention model [**3**] has had a lasting impact onto many fields of deep learning including natural language processing (NLP) and computer vision [**80, 72**]. Dot product key-value attention [**71**] is used in Transformer models for language processing. This attention architecture acts as a filter over a set of values, and appropriately performs a weighted sum of the values based on a given query. The formula given in [**71**] for this filtering process takes as input a query $Q \in \mathbb{R}^{1\times n}$, a set of m values represented as the matrix $V \in \mathbb{R}^{m\times h}$, and the associated m keys $K \in \mathbb{R}^{m\times n}$:

$$\text{Attention}(Q,K,V) = \text{softmax}(QK^T)V \quad . \tag{2.4.4}$$

This process therefore performs a weighted average over V, functionally compressing all the values in V into one vector. Effectively densely feeding V into whatever model that may need V for prediction, attention assumes the same parsimony inductive bias that motivates

this thesis, filtering out unnecessary information in V, and simplifying the prediction task downstream. For example, in NLP, V may represent the sequence of m words in a sentence, and computer vision might use V the different convolutions of an image.

In its basic form, attention is not required to be sparse. Instead, empirical evidence suggests that attention excels at directing the focus of downstream tasks onto important parts of V. Sparse attentive backtracking (SAB) demonstrates that the explicit use of sparse attention over a recurrent network helps in learning long-term dependencies [**35**]. Moreover, they note the importance of sparsity for long time scales. Other works, such as Neural Production Systems [**22**] also show the value of sparsity in attention for structural credit assignment.

Even though attention has seen much practical success in deep learning, it has sparingly made its way into reinforcement learning. Most notable uses of attention in reinforcement learning [**22, 70**] do not leverage attention directly for temporal credit assignment, instead using it on an auxiliary task to enrich representation. One example of the use of attention for temporal credit assignment in reinforcement learning is in the work on Temporal Value Transport (TVT) [**33**]. Based on on the attention weights learned for state representation in a POMDP, value is transported heuristically from the future to moments when attention drew focus on. Although the authors of TVT note that true credit assignment in humans is probably model-based [**55, 29**], TVT is a model-free algorithm. Moreover, the attention weights are re-used heuristically, and do not fully leverage the rich structure an attention-augmented recurrent network might offer.

Chapter 3

Counterfactuals for Efficient Credit Assignment

Counterfactuals rely on conditioning on what has already happened to infer what could have happened. We formally build upon this idea to develop a framework for model-free counterfactual policy evaluation that has favorable variance properties. This is desirable for long-term credit assignment as was pointed out in section 2.3.2. As it turns out, the family of *Hindsight Credit Assignment* algorithms in [**28**], of which [**11, 49**] follow, and the tree backup algorithm proposed in [**56**] directly follow from this framework, shedding light onto some theoretical benefits of HCA and tree backup. For the remainder of this chapter, we consider the discounted-MDP setting.

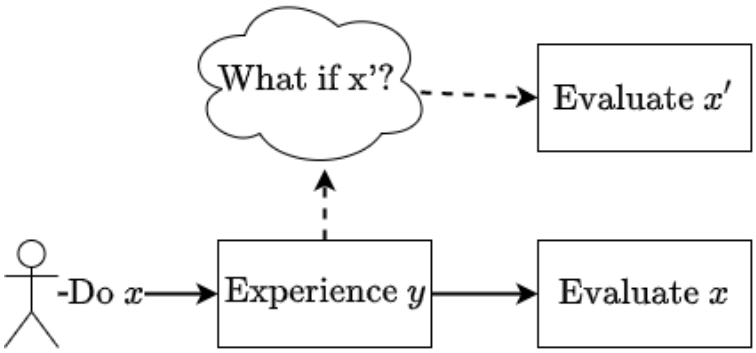

Fig. 3.1. A high level overview of using counterfactuals for better policy evaluation. The agent is able to reason about possible alternative x' based on experience from a different behavior x.

3.1. Conditional Importance Sampling

Conditional importance sampling (CIS) is the backbone of our derivation for counterfactual policy evaluation. Originally introduced in [**46**], CIS leverages a variance reduction

technique known as the (extended) conditional Monte Carlo method [**8**], which arises naturally from the law of total variance.

3.1.1. Importance Sampling

Importance sampling [**31, 10, 59**] is a popular tool in reinforcement learning for off-policy evaluation. That is, estimating the value function of a target policy π under samples taken from some other behavior policy μ. The term importance sampling originates from the simulation community [**8**], where it is meant to be used as a variance reduction method. In the context of RL, importance sampling simply refers to using *likelihood ratios* for a change of measure as seen in equation 2.2.3. Concretely,

$$\mathbb{E}_\pi\left[G_T\right] = \mathbb{E}_\mu\left[G_T \frac{\mathbb{P}_\pi(\tau)}{\mathbb{P}_\mu(\tau)}\right] = \mathbb{E}_\mu\left[G_T \prod_{k=0}^{T} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}\right] := \mathbb{E}_\mu\left[G_T \mathcal{P}_{0:T}^{\pi,\mu}\right] \quad , \tag{3.1.1}$$

where τ is a shorthand for the trajectory of random variables $S_0, A_0, S_1, A_1, \ldots, S_T, A_T$ and $\mathcal{P}_{i:j}^{\pi,\mu}$ is the importance sampling ratio $\prod_{k=i}^{j} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$. It is well-known that importance sampling is prone to high variance in high dimensional settings [**60, 8, 40**], which is especially pronounced in reinforcement learning when learning over long horizon [**45, 46**]. To build upon this intuition, note that for long horizons, $\mathcal{P}_{0:T}^{\pi,\mu}$ takes values close to 0 with a high probability, but may also be very large when rare trajectories are sampled. However, the expectation of $\mathcal{P}_{0:T}^{\pi,\mu}$ is always 1, no matter the horizon, which means the variance only grows with T.

To reduce the variance of importance sampling, marginalized estimators [**81, 27, 45, 19**] have been suggested. The underlying mechanism of these marginalized estimators stems from the conditional Monte Carlo Method as shown in [**46**], giving rise to conditional importance sampling for reinforcement learning.

3.1.2. The Conditional Monte Carlo Method

The conditional Monte Carlo estimator is a variance reduction technique that naturally follows from the law of total variance.

Theorem 3.1.1 (Law of Total Variance)**.** *Given the random variables Y and X, the variance of Y can be decomposed as*

$$\mathit{Var}(Y) = \mathbb{E}\left[\mathit{Var}(Y|X)\right] + \mathit{Var}(\mathbb{E}\left[Y|X\right]) \tag{3.1.2}$$

Proof. The proof follows from the law of total expectation [**75**]. □

Applied to the importance sampling estimator in (3.1.1), let ϕ_T be some statistic of the entire trajectory $\tau_{0:T}$, we have:

$$V^\pi(s) = \mathbb{E}_\mu \left[G_T \mathcal{P}_{0:T}^{\pi,\mu} \middle| S_0 = s \right] = \mathbb{E}_\mu \left[\mathbb{E}_\mu \left[G_T \mathcal{P}_{0:T}^{\pi,\mu} \middle| S_0 = s, \phi_T \right] \middle| S_0 = s \right] \quad . \tag{3.1.3}$$

By the law of total variance,

$$\mathrm{Var}\left(\mathbb{E}_\mu \left[G_T \mathcal{P}_{0:T}^{\pi,\mu} \middle| S_0 = s, \phi_T \right] \right) = \mathrm{Var}\left(G_T \mathcal{P}_{0:T}^{\pi,\mu} \right) - \mathbb{E}_\mu \left[\mathrm{Var}\left(G_T \mathcal{P}_{0:T}^{\pi,\mu} \right) \middle| S_0 = s, \phi_T \right] \quad .$$

Since the expected variance of a random variable is always non-negative, we have that the variance of the conditional importance sampling estimator in (3.1.3) is always equal or lower than the variance of the crude importance sampling estimator in (3.1.1):

$$\mathrm{Var}\left(\mathbb{E}_\mu \left[G_T \mathcal{P}_{0:T}^{\pi,\mu} \middle| S_0 = s, \phi_T \right] \right) \leq \mathrm{Var}\left(G_T \mathcal{P}_{0:T}^{\pi,\mu} \right) \quad .$$

There is also an extension of the conditional Monte Carlo estimator, aptly named the extended conditional Monte Carlo estimator [**8**] that allows us to condition instead on a stage-dependent variable ϕ_t instead of ϕ_T:

$$V^\pi(s) = \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \mathcal{P}_{0:T}^{\pi,\mu} \middle| S_0 = s \right] = \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t \mathbb{E}_\mu \left[R_t \mathcal{P}_{0:T}^{\pi,\mu} \middle| S_0 = s, \phi_t \right] \middle| S_0 = s \right] \quad . \tag{3.1.4}$$

Unforunately, unlike CMC, ECM does not guarantee the variance remain unchanged or is reduced [**8**]. While we can similarly show that the variance of each term in the summation is individually reduced, the variance of a sum of random variables is not the sum of the variance of the random variables:

$$\mathrm{Var}(\sum_{t=0}^{T} X_t) = \sum_{t=0} \mathrm{Var}(X_t) + \sum_{k \neq t}^{T} \mathrm{Cov}(X_k, X_t) \quad .$$

Bratley [**8**] suggests that we need to simply hope for the covariance terms to play in our favor when using the extended Monte Carlo estimator[1].

Equations 3.1.3 and 3.1.4 provide the necessary framework for conditional importance sampling. In what follows, we will see how counterfactual policy evaluation corresponds to an application of the above concepts, and consequently the family of *hindsight credit assignment* algorithms and tree backup.

3.2. Counterfactual Policy Evaluation

Importance sampling, and consequently conditional importance sampling are methods for off-policy evaluation, while HCA pertains to the on-policy setting. We first argue that on-policy evaluation of equation 2.3.4 can be seen as a special case of off-policy evaluation

[1] [**46**] further develops some sufficient conditions for variance reduction for extended conditional importance sampling.

of equation 2.3.3. From this perspective we can derive a method for counterfactual policy evaluation using conditional importance sampling that amounts to HCA.

3.2.1. Off-policy Value Evaluation

Policy evaluation takes two forms as shown in equations 2.3.3 and 2.3.4. The first of which corresponds to the expected value of a policy conditioned only on some initial state s. The second form evaluates the expected value of a policy additionally conditioned on some initial action a. Let us consider instead a perturbed policy denoted π^a to be identical to π, except that π^a performs action a as its first action every time instead. Specifically,

$$\begin{aligned} \pi^a(a_t|s_t) &:= \pi(a_t|s_t) && for\ \ t > 0 \\ \pi^a(a_0|s_0) &:= 1 && for\ \ a_0 = a \\ \pi^a(a_0|s_0) &:= 0 && for\ \ a_0 \neq a \end{aligned}$$

The on-policy evaluation of the action-value function Q in equation 2.3.4 can also be seen as the off-policy evaluation of the value function of π^a under the behavior policy π:

$$Q^\pi(s,a) = V^{\pi^a}(s) = \mathbb{E}_\pi \left[\mathcal{P}_{0:T}^{\pi^a,\pi} \sum_{t=0}^{T} \gamma^t R_t \middle| S_0 = s \right] \ . \tag{3.2.1}$$

Naively, the above formulation reduces to the classical crude Monte Carlo estimate for $Q^\pi(s,a)$, since $\mathcal{P}_{0:T}^{\pi^a,\pi}$ just filters out all samples where $A_0 \neq a$:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\mathcal{P}_{0:T}^{\pi^a,\pi} \sum_{t=0}^{T} \gamma^t R_t \middle| S_0 = s \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_t \middle| S_0 = s, A_0 = a \right].$$

3.2.2. The Counterfactual Policy Estimator

Recall the idea of counterfactuals is to condition on what has or will happen to to improve credit assignment. Let us use CMC 3.1.3 or ECMC 3.1.4 on 3.2.1 by conditioning the off policy estimator with some future statistic ϕ_T or ϕ_t to get the following *counterfactual policy evaluation estimators*:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\mathbb{E}_\pi \left[\mathcal{P}_{0:T}^{\pi^a,\pi} G_T | S_0 = s, \phi_T \right] \middle| S_0 = s \right], \tag{3.2.2}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t \mathbb{E}_\pi \left[\ R_t \mathcal{P}_{0:T}^{\pi^a,\pi} | S_0 = s, \phi_t \right] \middle| S_0 = s \right]. \tag{3.2.3}$$

These estimators generalise the family of *Hindsight Credit Assignment* algorithms, which take two possible forms in [**28**]:

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_0 \middle| S_0 = s, A_0 = a\right] + \mathbb{E}_{\pi}\left[\sum_{t>0}^{T} \gamma^t \frac{h_{\pi}(A_0 = a|S_0, S_t)}{\pi(a|S_0)} R_t \middle| S_0 = s\right] \tag{3.2.4}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_T \frac{h_{\pi}(A_0 = a|S_0, G_T)}{\pi(a|S_0)} \middle| S_0 = s\right] \quad , \tag{3.2.5}$$

where $h_{\pi}(A_0 = a|S_0, G_T) = \mathbb{P}_{\pi}(A_0 = a|S_0, G_T)$ and $h_{\pi}(A_0 = a|S_0, S_t) = \mathbb{P}_{\pi}(A_0 = a|S_0, S_t)$. Particularly, the *counterfactual policy evaluation estimator*'s connection to the hindsight estimators can be formally stated as the following two theorems.

Lemma 3.2.1. *For any state s, action a, and policy π for which $\pi(a_t|s_t) > 0$ for all t, we have*

$$\mathbb{E}_{\pi}\left[\mathcal{P}_{0:T}^{\pi^a,\pi} \middle| S_0 = s, S_0, A_0\right] = \frac{\mathbb{P}_{\pi}(A_0 = a|A_0)}{\pi(a|s)}$$

$$\mathbb{E}_{\pi}\left[\mathcal{P}_{0:T}^{\pi^a,\pi} \middle| S_0 = s, S_t, A_t\right] = \frac{\mathbb{P}_{\pi}(A_0 = a|S_0 = s, S_t)}{\pi(a|s)} \;\; \textit{for } t > 0$$

Proof.

$$\mathbb{E}_{\pi}\left[\mathcal{P}_{0:T}^{\pi^a,\pi} \middle| S_0 = s, S_t, A_t\right]$$

$$= \sum_{s_0, a_0, \ldots, s_T, a_T} \mathcal{P}_{0:T}^{\pi^a,\pi} \mathbb{P}_{\pi}(s_0, a_0, \ldots, s_T, a_T | S_0 = s, S_t, A_t)$$

$$= \sum_{s_0 \in \mathcal{S}} \sum_{a_0 \in \mathcal{A}} \frac{\pi^a(a_0|s_0)}{\pi(a_0|s_0)} \sum_{s_1, a_1, \ldots, s_T, a_T} \mathcal{P}_{1:T}^{\pi^a,\pi} \mathbb{P}_{\pi}(s_0, a_0 | S_0 = s, S_t, A_t) \mathbb{P}_{\pi}(\tau_{1:T} | S_0 = s, S_t, A_t, s_0, a_0)$$

Because $\pi^a(a_t|s_t) = \pi(a_t|s_t)$ for $t > 0$

$$= \sum_{s_0 \in \mathcal{S}} \sum_{a_0 \in \mathcal{A}} \frac{\pi^a(a_0|s_0)}{\pi(a_0|s_0)} \mathbb{P}_{\pi}(s_0, a_0 | S_0 = s, S_t, A_t) \sum_{s_1, a_1, \ldots, s_T, a_T} \mathbb{P}_{\pi}(s_1, a_1, \ldots, s_T, a_T | S_0 = s, S_t, A_t, s_0, a_0)$$

$$= \sum_{s_0 \in \mathcal{S}} \sum_{a_0 \in \mathcal{A}} \frac{\pi^a(a_0|s_0)}{\pi(a_0|s_0)} \mathbb{P}_{\pi}(s_0, a_0 | S_0 = s, S_t, A_t)$$

$$= \sum_{a_0 \in \mathcal{A}} \frac{\pi^a(a_0|s)}{\pi(a_0|s)} \mathbb{P}_{\pi}(a_0 | S_0 = s, S_t, A_t)$$

Because $\pi^a(a_0|s) = 1$ only when $a_0 = a$, and 0 otherwise

$$= \frac{\mathbb{P}_{\pi}(A_0 = a|S_0 = s, S_t, A_t)}{\pi(a|s)}$$

Now consider when $t = 0$, because A_0 is independent of S_0 given A_0:

$$\frac{\mathbb{P}_{\pi}(A_0 = a|S_0 = s, S_0, A_0)}{\pi(a|s)} = \frac{\mathbb{P}_{\pi}(A_0 = a|A_0)}{\pi(a|s)}$$

And when $t > 0$, because A_0 is independent of A_t given S_t for $t > 0$:

$$\frac{\mathbb{P}_\pi(A_0 = a|S_0 = s, S_t, A_t)}{\pi(a|s)} = \frac{\mathbb{P}_\pi(A_0 = a|S_0 = s, S_t)}{\pi(a|s)}$$

□

Theorem 3.2.2 (State Conditioned Hindsight Estimator)**.** *Given the counterfactual policy evaluation estimator 3.2.2 that makes use of the extended conditional Monte Carlo method, and setting $\phi_t = (S_t, A_t, R_t)$, for any state s, action a, and policy π for which $\pi(a_t|s_t) > 0$ for all t, we obtain the state conditioned hindsight estimator in 3.2.4.*

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t\mathbb{E}_\pi\left[R_t\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0 = s, S_t, A_t, R_t\right]\middle|S_0 = s\right] \\ &= \mathbb{E}\left[R_0\middle|S_0 = s, A_0 = a\right] + \mathbb{E}_\pi\left[\sum_{t>0}^{T}\gamma^t\frac{\mathbb{P}_\pi(A_0 = a|S_0, S_t)}{\pi(a|S_0)}R_t\middle|S_0 = s\right] \quad . \end{aligned}$$

Proof.

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t R_t\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0 = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t\mathbb{E}_\pi\left[R_t\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0 = s, S_t, A_t, R_t\right]\middle|S_0 = s\right] \text{ by (3.2.3)} \\ &\text{Since } \mathcal{P}_{0:T}^{\pi^a,\pi} \text{ is conditionally independent of } R_t \text{ given } S_t, A_t \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t R_t\mathbb{E}_\pi\left[\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0 = s, S_t, A_t\right]\middle|S_0 = s\right] \\ &= \mathbb{E}_\pi\left[R_0\frac{\mathbb{P}_\pi(A_0 = a|A_0)}{\pi(a|s)}\middle|S_0 = s\right] + \mathbb{E}_\pi\left[\sum_{t>0}^{T}\gamma^t R_t\frac{\mathbb{P}_\pi(A_0 = a|S_0, S_t)}{\pi(a|S_0)}\middle|S_0 = s\right] \text{ by lemma 3.2.1} \\ &= \mathbb{E}_\pi\left[R_0\middle|S_0 = s, A_0 = a\right] + \mathbb{E}_\pi\left[\sum_{t>0}^{T}\gamma^t R_t\frac{\mathbb{P}_\pi(A_0 = a|S_0, S_t)}{\pi(a|S_0)}\middle|S_0 = s\right] \end{aligned}$$

□

Lemma 3.2.3. *For any state s, action a, and policy π for which $\pi(a_t|s_t) > 0$ for all t, we have*

$$\mathbb{E}_\pi\left[\mathcal{P}_{0:T}^{\pi^a,\pi}|S_0 = s, G_T\right] = \frac{\mathbb{P}_\pi(A_0 = a|S_0 = s, G_T)}{\pi(a|s)}$$

Proof.

$$\mathbb{E}_\pi\left[\mathcal{P}_{0:T}^{\pi^a,\pi}|S_0=s,G_T\right]$$
$$=\sum_{s_0,a_0,\ldots,s_T,a_T}\mathcal{P}_{0:T}^{\pi^a,\pi}\mathbb{P}_\pi(s_0,a_0,\ldots,s_T,a_T|S_0=s,G_T)$$

Because $\pi^a(a_t|s_t)=\pi(a_t|s_t)$ for $t>0$

$$=\sum_{s_0\in\mathcal{S}}\sum_{a_0\in\mathcal{A}}\frac{\pi^a(a_0|s_0)}{\pi(a_0|s_0)}\sum_{s_1,a_1,\ldots,s_T,a_T}\mathbb{P}_\pi(s_0,a_0|S_0=s,G_T)\mathbb{P}_\pi(s_1,a_1,\ldots,s_T,a_T|S_0=s,G_T,s_0,a_0)$$
$$=\sum_{s_0\in\mathcal{S}}\sum_{a_0\in\mathcal{A}}\frac{\pi^a(a_0|s_0)}{\pi(a_0|s_0)}\mathbb{P}_\pi(s_0,a_0|S_0=s,G_T)\sum_{s_1,a_1,\ldots,s_T,a_T}\mathbb{P}_\pi(s_1,a_1,\ldots,s_T,a_T|S_0=s,G_T,s_0,a_0)$$
$$=\sum_{s_0\in\mathcal{S}}\sum_{a_0\in\mathcal{A}}\frac{\pi^a(a_0|s_0)}{\pi(a_0|s_0)}\mathbb{P}_\pi(s_0,a_0|S_0=s,G_T)$$
$$=\sum_{a_0\in\mathcal{A}}\frac{\pi^a(a_0|s)}{\pi(a_0|s)}\mathbb{P}_\pi(a_0|S_0=s,G_T)$$

Because $\pi^a(a_0|s)=1$ only when $a_0=a$, and 0 otherwise

$$=\frac{\mathbb{P}_\pi(A_0=a|S_0=s,G_T)}{\pi(a|s)}$$

□

Theorem 3.2.4 (Return Conditioned Hindsight Estimator)**.** *Given the counterfactual policy evaluation estimator* (3.2.3) *that makes use of the conditional Monte Carlo method, and setting $\phi_T=G_T$, for any state s, action a, and policy π for which $\pi(a_t|s_t)>0$ for all t, we obtain the state conditioned hindsight estimator in* (3.2.5)*.*

$$Q^\pi(s,a)=\mathbb{E}_\pi\left[\mathbb{E}_\pi\left[\mathcal{P}_{0:T}^{\pi^a,\pi}G_T\middle|S_0=s,G_T\right]\middle|S_0=s\right]$$
$$=\mathbb{E}_\pi\left[G_T\frac{\mathbb{P}_\pi(A_0=a|S_0,G_T)}{\pi(a|S_0)}\middle|S_0=s\right]\quad.$$

Proof.

$$Q^\pi(s,a)=\mathbb{E}_\pi\left[G_T\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0=s\right]$$
$$=\mathbb{E}_\pi\left[\mathbb{E}_\pi\left[G_T\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0=s,G_T\right]\middle|S_0=s\right]\text{ by (3.2.2)}$$

Since $\mathcal{P}_{0:T}^{\pi^a,\pi}$ is conditionally independent of G_T given G_T

$$=\mathbb{E}_\pi\left[G_T\mathbb{E}_\pi\left[\mathcal{P}_{0:T}^{\pi^a,\pi}\middle|S_0=s,G_T\right]\middle|S_0=s\right]$$
$$=\mathbb{E}_\pi\left[G_T\frac{\mathbb{P}_\pi(A_0=a|S_0,G_T)}{\pi(a|S_0)}\middle|S_0=s\right]\text{ by lemma 3.2.3}$$

□

In light of theorems 3.2.2 and 3.2.4, we can therefore distill HCA into three main ingredients:

(1) Importance sampling using a *perturbed* policy as the target
(2) The use of the Conditional Monte Carlo Method over the importance sampling weights.
(3) A particular choice of conditioning statistics ϕ_t, ϕ_T

3.3. Tree Backup

The tree backup algorithm originally derived in [**56**] is an approach for off-policy evaluation that does not involve explicit importance sampling ratios. In fact, the absence of the behavior policy at the denominator means that off-policy evaluation can be achieved without knowledge of the behavior policy. Consider the off-policy evaluation problem of equation 2.3.4, where Q^π is estimated using samples from an unknown behavior policy μ. The full Monte Carlo version of n-step tree backup[2], where $n = T$ is

$$Q^\pi(s,a) = \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t \prod_{k=1}^{t} \pi(A_k|S_k)(R_t + \gamma \sum_{\alpha \neq A_{t+1}} \pi(\alpha|S_{t+1}) Q^\pi(S_{t+1}, \alpha)) \middle| S_0 = s, A_0 = a \right] . \tag{3.3.1}$$

By convention when $k > j$, $\prod_{k=i}^{j} X_k = 1$. The algorithm intuitively bootstraps a sample trajectory with estimated Q-values for all unseen paths to estimate Q^π. Like HCA, tree backup also extrapolates from all the actions that could have been taken along a trajectory of the behavior policy. However, instead of assuming that the same policy would be followed in those alternatives, it can use a different policy as a target.

Using the same conditional importance sampling framework discussed previously, we can re-derive the tree-backup algorithm, and explain for the first time the statistical concepts at play without having recourse to eligibility traces which they have always been associated with [**56, 51, 47**].

Theorem 3.3.1 (Tree Backup and Conditional Importance)**.** *Given the extended conditional importance sampling estimator in* (3.1.4)*, if* $\phi_t = (A_1, A_2, \ldots, A_t)$*, the off-policy action-value estimator reduces to the tree backup algorithm in* (3.3.1)

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t \mathbb{E}_\mu \left[R^t \mathcal{P}_{1:t}^{\pi,\mu} \middle| S_0 = s, A_0 = a, A_1, A_2, \ldots, A_t \right] \middle| S_0 = s, A_0 = a \right] \\ &= \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t \prod_{k=1}^{t} \pi(A_k|S_k)(R_t + \gamma \sum_{\alpha \neq A_{t+1}} \pi(\alpha|S_{t+1}) Q^\pi(S_{t+1}, \alpha)) \middle| S_0 = s, A_0 = a \right] . \end{aligned}$$

[2]The results in this section can be extended to any other values of n.

Proof. The proof can be found in Appendix A.1 □

From Theorem 3.3.1, we can achieve a better understanding of the real benefits of the tree backup algorithm. We leave any theoretical analysis of this theorem for future work, but this provides a good starting point for explaining the empirical results of tree backup [**56**].

Chapter 4

Sparse Attentive Value Gradients: A Model-Based Method for Better Credit Assignment

We have seen in Chapter 2 how credit is assigned in model-based RL, and some of the shortcomings of crude value gradient estimators for long-term credit assignment. Based on the parallel between recurrent networks and value gradients, we devise new solutions to address the problem of long-term credit assignment. In this chapter, we explore this connection and propose a new method for long-term credit assignment in partially observable environments using attention and gradient truncation.

4.1. Value Gradients and Recurrent Networks

The connection between value gradient learning and recurrent networks has been established previously [**15, 30**]. More precisely, the recurrent process $f_\theta(x_t, h_{t-1})$ in an RNN is replaced with a composition of a transition model $f_\psi(s_{t-1}, a_{t-1}, \xi)$ and a policy $\pi_\theta(s_t, \xi)$. While the popularized SVG algorithm [**30**] claims this relationship, it is not immediately clear from the value gradient formulation derived in their work that it shares the same form as BPPT in equation 2.1.3. For this reason, we will re-derive value gradients for reinforcement learning following a similar process as the derivation of equation 2.1.4, and show its equivalence with the value gradient derived for SVG(∞) [**30**].

Recall the problem statement, where we are looking to maximize $\mathbb{E}_\theta\left[h(X)\right]$. Without loss of generality, let us assume for the remainder of this section that X is deterministic. Therefore, we are trying to maximize $h(x) = \sum_{t=0}^{T} r(s_t,a_t)$ as written in 2.2.2. Moreover, assume an initial state s_0, then we are equivalently interested in maximizing the value function in

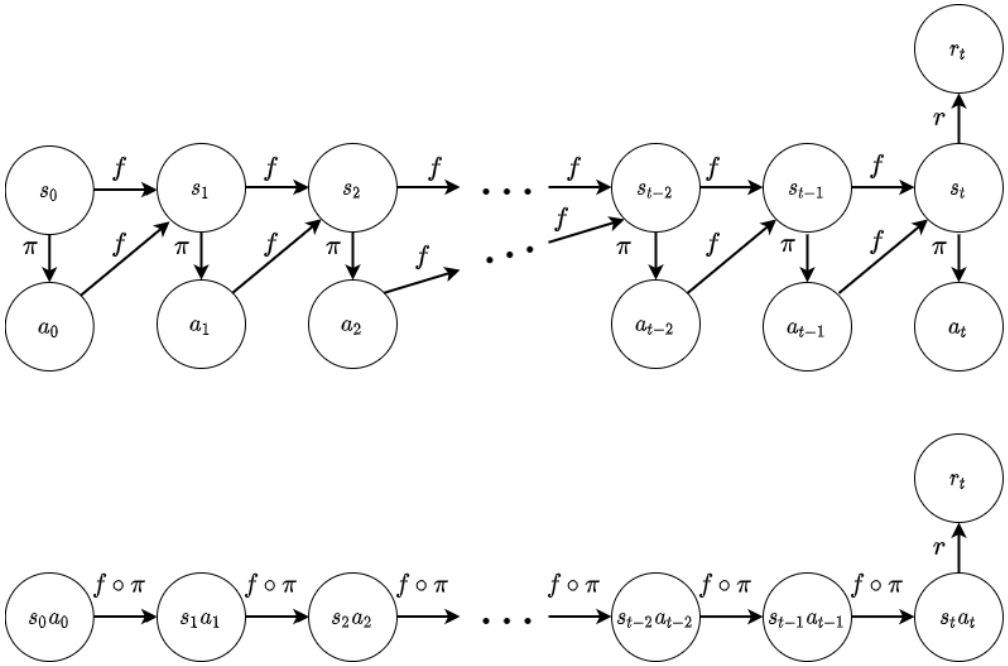

Fig. 4.1. Visualisation of the computation graph for computing the reward at time step t. If f, r and π are known and differentiable, the similarities with Figure 2.1 imply an identical form of backpropagation for credit assignment.

equation 2.3.3, $\max_\theta V^0(s_0;\theta)$ where

$$\begin{aligned} V^0 &:= \sum_{t=0}^{T} r_t \ , & s_t &:= f(s_{t-1}, a_{t-1}) \ \text{ for } t > 0 \ , \\ r_t &:= r(s_t, a_t) \ , & a_t &:= \pi_\theta(s_t) \ . \end{aligned} \tag{4.1.1}$$

If f, r and π_θ are all differentiable with respect to their arguments, then we can maximize $V^0(s_0;\theta)$ by gradient ascent using the gradient $\nabla_\theta V^0(s_0;\theta)$.

$$\begin{aligned} \nabla_\theta V^0(s_0;\theta) &= \sum_{t=0}^{T} \frac{dr_t}{d\theta} \\ \frac{dr_t}{d\theta} &= \frac{\partial r_t}{\partial s_t}\frac{ds_t}{d\theta} + \frac{\partial r_t}{\partial a_t}\frac{da_t}{d\theta} \\ \frac{ds_t}{d\theta} &= \frac{\partial s_t}{\partial s_{t-1}}\frac{ds_{t-1}}{d\theta} + \frac{\partial s_t}{\partial a_{t-1}}\frac{da_{t-1}}{d\theta} \\ \frac{da_t}{d\theta} &= \frac{\partial a_t}{\partial \theta} + \frac{\partial a_t}{\partial s_t}\frac{ds_t}{d\theta} \\ \frac{ds_0}{d\theta} &:= \mathbf{0} \end{aligned}$$

Cleaning up the above equations, we can identify the recurrent relationship:

$$\begin{aligned}
\nabla_\theta V^0(s_0;\theta) &= \sum_{t=0}^{T} \frac{dr_t}{d\theta} \\
\frac{dr_t}{d\theta} &= \left(\frac{\partial r_t}{\partial s_t} + \frac{\partial r_t}{\partial a_t}\frac{\partial a_t}{\partial s_t}\right)\frac{ds_t}{d\theta} + \frac{\partial r_t}{\partial a_t}\frac{\partial a_t}{\partial \theta} \\
\boldsymbol{\frac{ds_t}{d\theta}} &= \frac{\partial s_t}{\partial a_{t-1}}\frac{\partial a_{t-1}}{\partial \theta} + \left(\frac{\partial s_t}{\partial s_{t-1}} + \frac{\partial s_t}{\partial a_{t-1}}\frac{\partial a_{t-1}}{\partial s_{t-1}}\right)\boldsymbol{\frac{ds_{t-1}}{d\theta}} \\
\frac{ds_0}{d\theta} &:= \mathbf{0}
\end{aligned} \tag{4.1.2}$$

The same recurrent relationship (highlighted in bold) is found in value gradients as in traditional recurrent networks, described in equation 2.1.3, the only difference being the added policy $\pi(s_t)$ that must be differentiated through to obtain the state's relationship with θ. From this parallel, the problems of long-term credit assignment in RNNs, namely the *vanishing/exploding gradient problem*, also need to be addressed for value gradients. Thankfully, a lot of research has already been done for RNNs to improve long-term credit assignment, and given the similarities with value gradients, can be readily applied for reinforcement learning.

4.1.1. Forward Versus Backward Credit Assignment

It is worth noting the two different forms the value gradient may take: the forward version seen in equation 4.1.2, and the backward version derived in [**30**]:

$$\begin{aligned}
\nabla_\theta V^t(s_t;\theta) &= \sum_{k=t}^{T} \frac{\partial r_k}{\partial a_k}\frac{\partial a_k}{\partial \theta} + \frac{\partial s_{k+1}}{\partial a_k}\frac{\partial a_k}{\partial \theta}\frac{\partial V^{k+1}(s_{k+1};\theta)}{\partial s_{k+1}} \ , \\
\frac{\partial V^k(s_k;\theta)}{\partial s_k} &= \frac{\partial r_k}{\partial s_k} + \frac{\partial r_k}{\partial a_k}\frac{\partial a_k}{\partial s_k} + \left(\frac{\partial s_{t+k}}{\partial s_k} + \frac{\partial s_{t+k}}{\partial a_k}\frac{\partial a_k}{\partial s_k}\right)\frac{\partial V^{k+1}(s_{k+1};\theta)}{\partial s_{k+1}} \ , \\
\frac{\partial V^T(s_T;\theta)}{\partial s_T} &:= \mathbf{0} \ .
\end{aligned} \tag{4.1.3}$$

The main difference being that credit is assigned in the forward version by assessing how changing actions in the past may affect the current reward. The backward version instead assigns credit by determining how a current change in actions may affect future rewards. The implication of (4.1.3) is that current gradient estimates are dependent on the future, while (4.1.2) estimates current gradients with respect to the past. The same duality can be found between the popular REINFORCE estimator (2.2.8) and the separately derived GPOMDP[**5**] estimator (2.2.7). Again, the duality can also be said for reverse and forward mode AD. In every case, the *backward* algorithm ((4.1.3), REINFORCE, reverse AD, BPPT) use future values for credit assignment, while their *forward* counterparts ((4.1.2), GPOMDP, forward AD, RTRL[**78**]) use past values. Theoretically, the corresponding pairs always compute the same values, but offer different practical advantages [**68, 5, 78, 76, 20**]. Just like how

GPOMDP is generally more amenable to the online setting [**5**], forward SVG(∞) (4.1.2) inherits similar properties, because the gradient at t computed using only past Jacobians (instead of future Jacobians as in (4.1.3)).

4.2. Truncated Value Gradients

Gradient truncation is a simple solution to manage memory and unstable gradients in long recurrent networks. Given the shared recurrence relationship in value gradients, it is natural to extend gradient truncation to model-based reinforcement learning. An **n-truncated value gradient**, $\nabla_\theta V^t_{n-trunc}(s_t;\theta)$ can be derived from (4.1.2) and (2.4.1), where in the RL case s_{t-n} is considered a constant during backpropagation with respect to a reward r_t.

Definition 4.2.1 (n-truncated Value Gradients)**.** *Consider the model-based reinforcement learning setting, where we are interested in finding the gradient of the deterministic value function $V^t_\theta(s) = \sum_{k=t}^T r_k$, where r_k is defined in* (4.1.1)*. The **n-truncated value gradient** is*

$$\begin{aligned}
\nabla_\theta V^t(s_t;\theta) &= \sum_{k=t}^{T} \frac{dr_k}{d\theta} \\
\frac{dr_k}{d\theta} &= \left(\frac{\partial r_k}{\partial s_k} + \frac{\partial r_k}{\partial a_k}\frac{\partial a_k}{\partial s_k}\right)\frac{ds_k}{d\theta} + \frac{\partial r_k}{\partial a_k}\frac{\partial a_k}{\partial \theta} \\
\frac{ds_i}{d\theta} &= \frac{\partial s_i}{\partial a_{i-1}}\frac{\partial a_{i-1}}{\partial \theta} + \left(\frac{\partial s_i}{\partial s_{i-1}} + \frac{\partial s_i}{\partial a_{i-1}}\frac{\partial a_{i-1}}{\partial s_{i-1}}\right)\frac{ds_{i-1}}{d\theta} \quad , \; for\; i > k-1 \\
\frac{ds_{k-n}}{d\theta} &:= \mathbf{0}
\end{aligned}$$

The implication of an **n-truncated value gradient** being that the horizon is reduced to n during credit assignment, preventing any long-term credit assignment. We can visualize this truncation in Figure 4.2. Even though this may seem like a naive solution, it is not so far-fetched when compared to current common practices in RL, namely discounting.

4.2.1. Relationship with Discounted Critics

The truncated value gradient can also be written in backwards form as:

$$\begin{aligned}
\nabla_\theta V^t_{n-trunc}(s_t;\theta) &= \sum_{k=t}^{T} \frac{\partial r_k}{\partial a_k}\frac{\partial a_k}{\partial \theta} + \frac{\partial s_{k+1}}{\partial a_k}\frac{\partial a_k}{\partial \theta}\frac{\partial V^{k+1}(s_{k+1};\theta)}{\partial s_{k+1}} \quad , \\
\frac{\partial V^i(s_i;\theta)}{\partial s_i} &= \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \left(\frac{\partial s_{i+1}}{\partial s_i} + \frac{\partial s_{i+1}}{\partial a_i}\frac{\partial a_i}{\partial s_i}\right)\frac{\partial V^{i+1}(s_{i+1};\theta)}{\partial s_{i+1}} \quad , \; for \;\; i < T-n \\
\frac{\partial V^{T-n}(s_{T-n};\theta)}{\partial s_{T-n}} &:= \mathbf{0} \;\; .
\end{aligned} \tag{4.2.1}$$

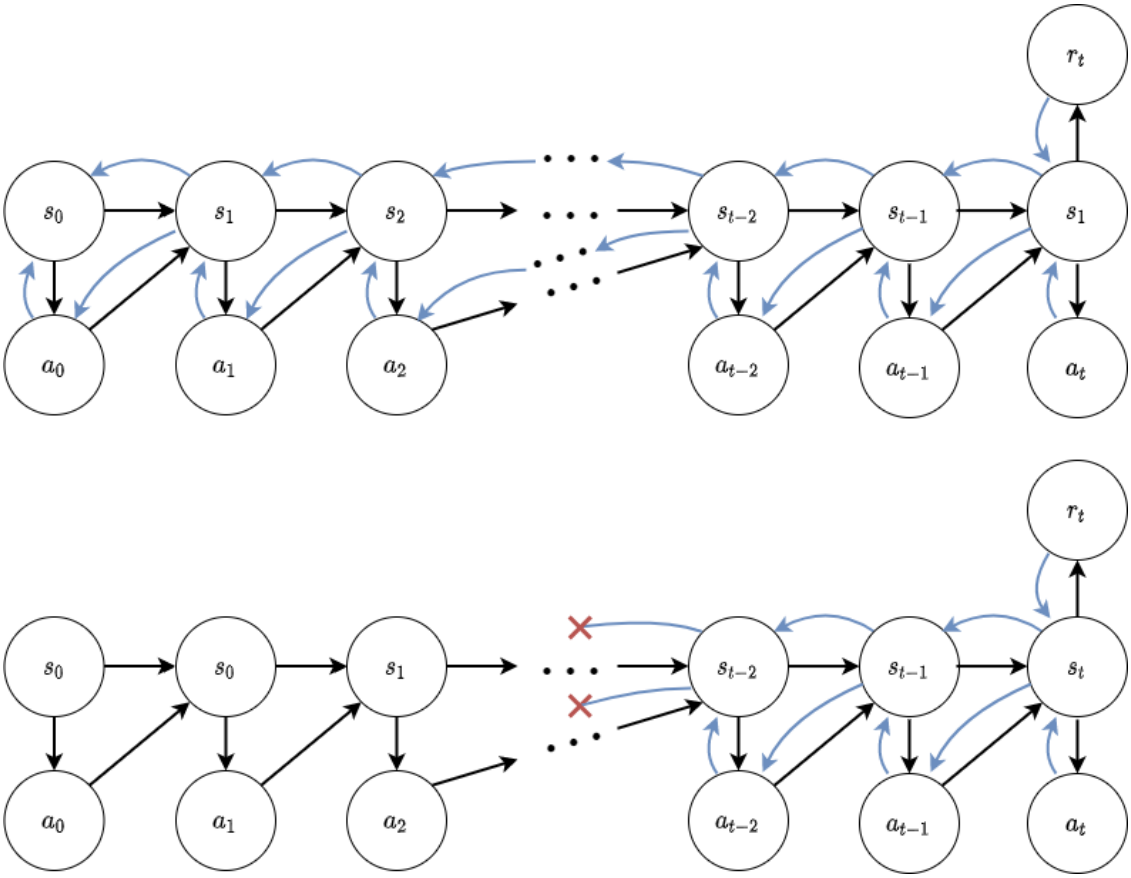

Fig. 4.2. A visualisation of how credit is assigned a time step t in traditional value gradients and truncated value gradients. Above is a full application of backpropagation, and below is an example of TBPPT ($n = 3$) in value gradients.

We can rewrite the recurrent relationship iteratively using similarly to equation 2.1.4 to better understand how it compares to the original value function:

$$\frac{\partial V^i_{n-trunc}(s_i;\theta)}{\partial s_i} = \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \sum_{j=i+1}^{i+n}\left(\prod_{l=i}^{j}(\frac{\partial s_{l+1}}{\partial s_l} + \frac{\partial s_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial s_l})\right)(\frac{\partial r_j}{\partial s_j} + \frac{\partial r_j}{\partial a_j}\frac{\partial a_j}{\partial s_j}) \quad (4.2.2)$$

The original un-truncated and undiscounted value function being:

$$V^i(s_i;\theta) = \sum_{k=i}^{T} r_k$$

$$\frac{\partial V^i(s_i;\theta)}{\partial s_i} = \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \sum_{j=i+1}^{T}\left(\prod_{l=i}^{j}(\frac{\partial s_{l+1}}{\partial s_l} + \frac{\partial s_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial s_l})\right)(\frac{\partial r_j}{\partial s_j} + \frac{\partial r_j}{\partial a_j}\frac{\partial a_j}{\partial s_j}) \ ,$$

and an **n-step** value function being:

$$V^i_{n-step}(s_i;\theta) = \sum_{k=i}^{i+n} r_k$$

$$\frac{\partial V^i_{n-step}(s_i;\theta)}{\partial s_i} = \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \sum_{j=i+1}^{i+n}\left(\prod_{l=i}^{j}(\frac{\partial s_{l+1}}{\partial s_l} + \frac{\partial s_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial s_l})\right)(\frac{\partial r_j}{\partial s_j} + \frac{\partial r_j}{\partial a_j}\frac{\partial a_j}{\partial s_j}) \ . \quad (4.2.3)$$

We can now appreciate the equivalence between equations 4.2.2 and 4.2.3, demonstrating that the **n-truncated value gradient** is equivalent to a **full value gradient** using an **n-step value function**. Given that in traditional actor-critic methods, the critic $\hat{V}^i(s_i)$ is only used derivative estimation [**13**], the **n-truncated value gradient** is not so different than policy gradient algorithms with $\gamma = 1 - \frac{1}{n}$, where future rewards are discounted smoothly instead of discretely. In fact, in the derivation for GPOMDP [**5**], the proposed discount factor, in their case β, was the first-order infinite impulse response filter alternative to the discrete n-th order finite impulse response filter of an n-step return . Given the success and popularity of the discount factor in RL, the logical parallel in value gradients would be to truncate the gradients just as in RNNs. There is therefore a strong precedent to use **n-truncated value gradients** in RL from both the deep learning, and the reinforcement learning literature.

4.2.2. Experiments

We run a simple set of experiments to verify the benefits in truncating gradient for model-based RL. As a baseline, we use the $SVG(\infty)$ proposed in [**30**], and compare its performance with a truncated version following equation 4.1.2. For completeness, a full description of the models and environments follows.

Openai Gym's Pendulum-v0 The Pendulum environment is the one implemented in the Openai Gym library [**9**], named *Pendulum-v0*. The task consists of swinging a hanging pendulum upwards, and keeping it stable while upright over a period of T time steps. Consider θ_t to be the angle between the desired upright position and the current position of the pendulum, $\dot{\theta}_t$ the current angular velocity of the pendulum, and the current action a_t to be the torque applied to the pendulum, then the reward is defined as

$$r_t = \theta_t^2 + 0.1(\dot{\theta}_t^2) + 0.001(a_t^2) \ .$$

Cancer environment The Cancer environment is taken from [**52**]. It models the growth of a tumour over time under the influence of chemotherapy. The state of the environment is x_t, the normalized density of the tumour, and the actions of an agent u_t is the strength of the drug used for chemotherapy. The reward and dynamics of the environment

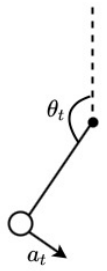

Fig. 4.3. Sketch of the *pendulum-v0* task as described in the OpenAI gym documentation . The action is a torque applied to the simple pendulum, and the state space is described as the angular position θ_t and the angular momentum $\dot{\theta}$.

can be identified as:

$$r_t := -(ax_t^2 + u_t^2) \tag{4.2.4}$$

$$x_{t+1} := x_t + (rx_t \ln(\frac{1}{x_t}) - \delta u_t x_t)\Delta \ , \tag{4.2.5}$$

where $a = 3$ is a cost coefficient related to the tumour density, $r = 0.3$ is the growth rate coefficient of the tumour, $\delta = 0.45$ describes the power of the chemotherapy, and $\Delta = \frac{1}{20}$ is a time scale coefficient.

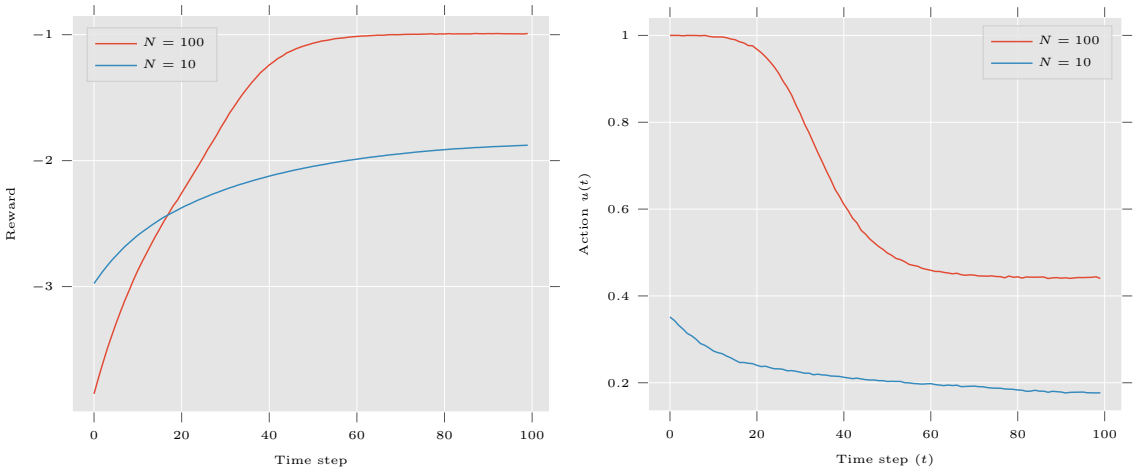

Fig. 4.4. An example trajectory is plotted for the optimal behavior of a long-term agent ($N = 100$) and a myopic agent ($N = 10$) in the cancer environment. On the left, early rewards are favored by the short-term agent at the cost of long-term performance. On the right, we can see that the payoff for early action is only observed much later.

SVG(∞) The stochastic value gradient implemented here follows (4.1.2). The policy and transition model are reparameterized using a Gaussian distribution with mean $\mu = 0$ and standard deviation σ_π and σ_f respectively. More precisely, the policy takes the deterministic form $\pi_\theta(s_t, \eta_t) = \bar{\pi}_\theta(s_t) + \eta_t \sigma_\pi$, where $\eta_t \sim \mathcal{N}(0,1)$. The transition model is similarly defined and is trained directly without noise with the following loss:

$$\text{Given } s_t, a_t, s_{t+1}$$
$$\mathcal{L}_f = \frac{1}{2}(s_{t+1} - \bar{f}(s_t, a_t))^2 \quad (4.2.6)$$

for any t, and noise is inferred during policy optimization to avoid compounding errors. The noise inference procedure is:

$$\text{Given } s_t, a_t, s_{t+1}$$
$$\xi = (s_{t+1} - \bar{f}(s_t,a_t))/\sigma_f \ , \quad (4.2.7)$$

following from the fact that $s_{t+1} = \bar{f}(s_t, a_t) + \sigma_f \xi$ for a Gaussian distribution. We assume the true differentiable reward function $r(s_t,a_t)$ is given.

Algorithm 1: T-SVG(∞)

Input: Horizon T, truncate length N, random process $\rho(\eta)$, true reward function $r(s_t,a_t)$

while *not converged* **do**
- Observe initial state s_1;
- **for** $t \leftarrow 1$ **to** T **do**
  - $a_t \leftarrow \pi_\theta(s_t, \eta_t), \eta_t \sim \rho(\eta)$;
  - Take action a_t and observe s_{t+1}, r_t from the environment ;
  - Insert (s_t, a_t, r_t, s_{t+1}) into $\mathcal{D}$;
- **end**
- Train transition model $f(s_t, a_t, \xi)$ with $\mathcal{D}$ using (4.2.6);
- Initialize return: $G \leftarrow 0$;
- Initialize predicted state: $\hat{s}_1 \leftarrow s_1$;
- **for** $t \leftarrow 1$ **to** T **do**
  - Infer $\xi | s_t, a_t, s_{t+1}$ using (4.2.7);
  - $\hat{a}_t \leftarrow \pi_\theta(\hat{s}_t, \eta_t)$;
  - $\hat{r}_t \leftarrow r(\hat{s}_t, \hat{a}_t)$;
  - $\hat{s}_{t+1} \leftarrow f(\hat{s}_t, \hat{a}_t, \xi)$;
  - **if** $t+1 \mod N == 0$ **then**
    - $\hat{s}_{t+1} \leftarrow$`Detach(`$\hat{s}_{t+1}$`)`;
  - **end**
  - $G \leftarrow G + \hat{r}_t$;
- **end**
- Update π_θ using $\nabla_\theta G$;

end

T-SVG(∞) The truncated stochastic value gradient (T-SVG) is a generalization of SVG(∞) with the added choice of gradient truncation hyper-parameter N. In the case that N exceeds the horizon T, T-SVG(∞) reduces to SVG(∞). In practice, it is easier to truncate the gradient by cutting the episode into sequences of length N instead of maintaining a moving window of length N.

The benefits and pitfalls of gradient truncation are demonstrated in the results of Figure 4.5. Depending on the environment, the added gradient stability from truncation can sometimes outweigh the bias of forgetting about long-term dependencies, as is the case in the

pendulum environment. However, in the case where long-term reasoning is necessary to find an optimal policy, as is the case for the cancer environment shown in Figure 4.4, gradient truncation impedes on performance.

We can also verify the relationship of discounting and gradient truncation discussed in Section 4.2.1 from the results in Figure 4.5. Notably, the qualitative behavior of discounting and gradient truncation are the same in both environment. In the pendulum task, performance improves for smaller values of γ and N, and gradient norms are reduced. In the cancer environment, performance suffers from weaker foresight, and gradient norms are not so easily reduced.

Finally, in both environments, we can empirically observe the *exploding gradient problem* in Figure 4.7. Namely, the log-magnitude of the value gradient seems to increase linearly with the effective horizon N.

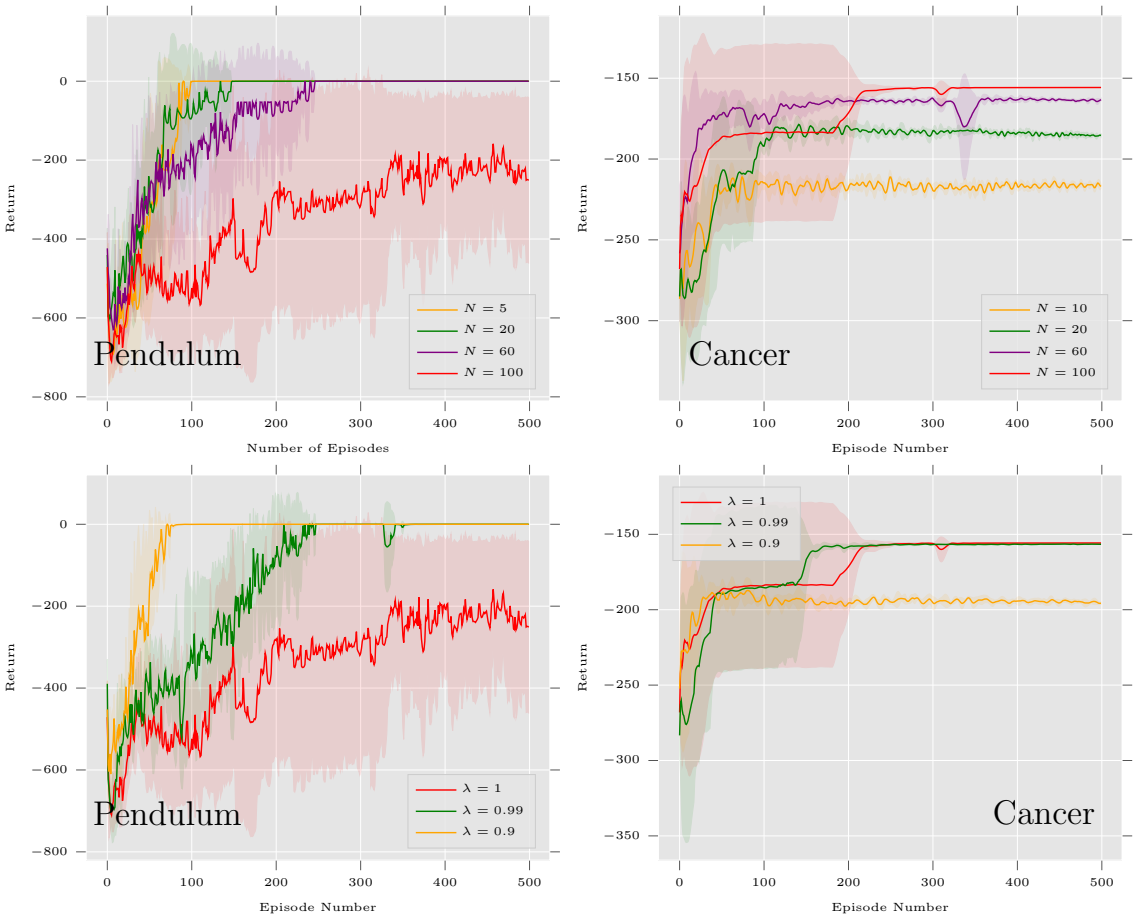

Fig. 4.5. Performance of T-SVG(∞) given different discount factors λ and truncation lengths N on the pendulum and cancer tasks. Both tasks are evaluated on episodes of length 100. When the truncation length is being changed (top two graphs), the discount factor is set to 1. Vice versa, when the discount factor is being changed (bottom 2 graphs), the truncation length is set to $N = 100$. For certain tasks that require long-term credit assignment, such as the cancer environment, too much truncation degrades performance.

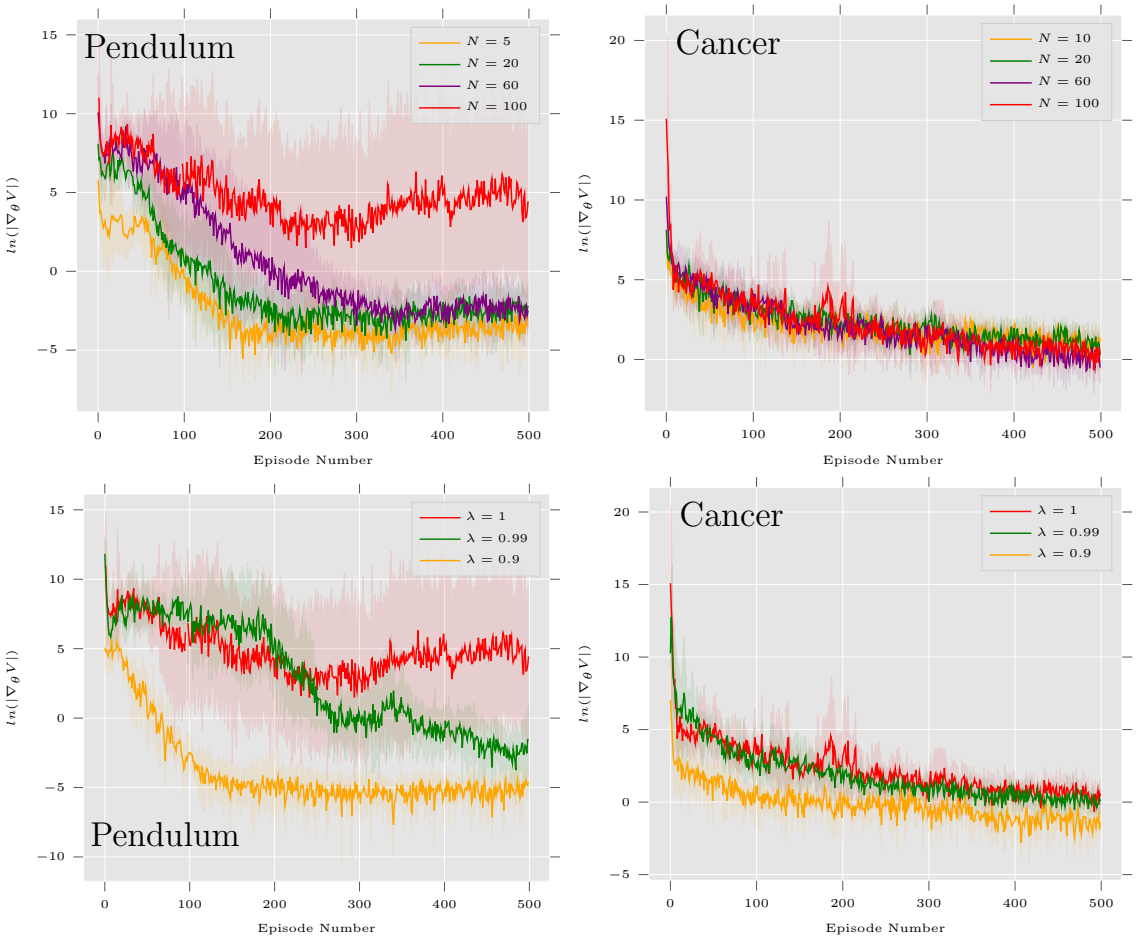

Fig. 4.6. The norm of the value gradient is plotted across experiments over time. Value gradient truncation has a larger effect on the pendulum environment, suggesting that truncating in the pendulum environment is more effective than the cancer environment. Discounting has a similar effect on the gradient behaviors.

4.3. Sparse Credit Assignment with Memory and Attention

The goal of the rest of this thesis is to design a method for long-term credit assignment that can perform long-term reasoning while applying gradient truncation. First, we move to the partially observable setting as it poses a more difficult credit assignment problem. More specifically, we seek to solve a so-called type 1 information acquisition task as described in [**33**], as opposed to a type 2 task. We give a detailed description of the corresponding environments used for each task in this work, which will better demonstrate the difficulties in each.

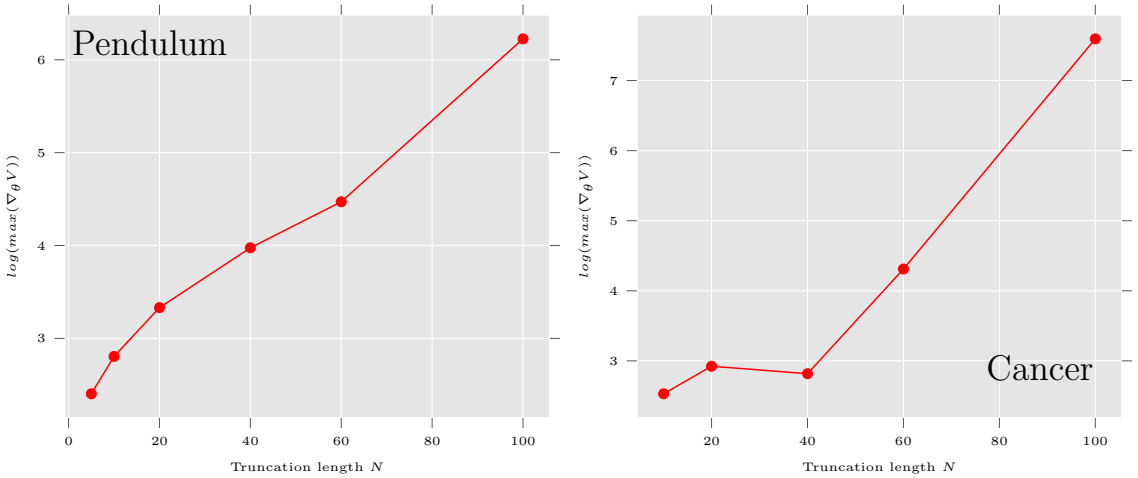

Fig. 4.7. The average log maximum gradients are plotted with respect to truncation length. The maximum gradient for each experiment is computed by taking the mean of the twenty largest gradients during training.

4.3.1. Environments

4.3.1.1. Passive Memory Task (Type 2 Information Acquisition). The passive memory task (PMT) is designed to test an agent's ability to learn adequate state representations in a partially observable setting where past observations, however distant, are necessary for future reward predictions. Conceptually inspired by the grid world environments in [**33**][**74**], the passive memory task in question here is a simplification of the aforementioned environments generalised in the continuous state-action space. The task is separated in three phases: remember, distractor, and recall.

(1) The remember phase: A random observation o_m is generated for the agent to see. Any actions in this phase bears zero consequences.

(2) The distractor phase: Any desired continuous task can be inserted here. It serves to distract the agent with an unrelated task, temporally distancing the first and second phase.

(3) The recall phase: Inconsequential and unrelated observations are shown here, but a reward is awarded to the agent for adequately remember the randomly generated observation o_m seen during the remember phase.

Consider t_1 to be the final time step of the remember phase, t_2 the transition step between phases 2 and 3, and a constant c as the maximum possible reward per time step. Let's also normalize [1] the action a_t and observation o_m, then the rewards specific to this simple

[1]The goal here is to make this reward work for any state and action dimensionality, regardless of their maximum and and minimum values. For example, it would be sufficient to do the following operations: sigmoid(mean($\vec{a_t}$)) and sigmoid(mean($\vec{o_t}$)).

memory task are defined :

$$r_{t<t_1} = 0$$
$$r_{t\geq t_2} = (a_t - o_m)^2 + c \tag{4.3.1}$$

4.3.1.2. Active Memory Task (Type 1 Information Acquisition). The active memory task (AMT) is designed to test an agent's ability to simultaneously learn adequate memory-based state representations, and perform long-term temporal credit assignment in the action-space. Concretely, the memory state is not passively observed in this setting. Instead, the agent must actively seek to obtain the memory state, and is only rewarded for it in the future when it needs to be recalled. Once again inspired by the grid world environments in [**33**], the active memory task here is a simplification of the Active Visual Match task generalised in the continuous state-action space. Similarly to the passive memory task, this problem is also separated in the same three phases: remember, distractor, and recall. The main difference is in the first phase, where the randomly generated memory observation o_m is only fully observed if the agent performs the correct action. The intuition is that credit needs to be assigned from the rewards in $t \geq t_2$ to the actions in $t < t_1$, which was not required in the passive memory task. Given the correct action is a_m, statically defined as part of the environment, then the observations seen during phase 1 are defined as:

$$o_{t<t_1} = o_m + (a_m - a_t)\mathcal{N}(0,1) \tag{4.3.2}$$

4.3.1.3. Distractor Environment. In the two previous problems, a core environment is needed to complete the tasks. We use two different candidates for this: a dummy environment, and the pendulum environment described in Section 4.2.2. The dummy environment just serves to extend the passive and active memory tasks into a larger horizon. This environment always returns $s_t = 0, r_t = 0$ for any t and any a_t.

4.3.2. Latent Stochastic Value Gradient

Latent stochastic value gradient is an extension of the original stochastic value gradient algorithm in a partially observable environment. Planning in a latent space using, model-based reinforcement learning has been explored more in depth previously [**26, 25, 73**]. Here, we employ a simplified version of those ideas, and apply them directly to T-SVG(∞) in algorithm 1.

At a high level, the goal of latent-SVG(∞) is to learn a latent representation of the state space such that the resulting latent space operates as a markovian process. That is, a latent model takes as input the current history of observations and actions, $(o_1 a_1 o_2 a_2, \ldots, a_{t-1} o_t)$, and outputs a latent state h_t that fully contains the information necessary to predict r_t and

h_{t+1} given a_t. After learning the correct latent representation, latent-SVG(∞) operates just as SVG(∞) would, but in this latent space.

Recall the dynamics of a partially observable environment defined as:

$$\begin{aligned} s_{t+1} &\sim p(s_{t+1}|s_t, a_t) \\ o_t &\sim p(o_t|s_t) \\ r_t &\sim p(r_t|s_t, a_t) \end{aligned}$$

Given only access to experience containing observations, actions, and rewards, we seek to learn a belief state h_t that can replace the true hidden state s_t. The models learned for latent planning are:

$$h_t = b(o_{1..t}a_{1..t-1}) \qquad \text{Encoder} \tag{4.3.3}$$

$$o_t = d(h_t) \qquad \text{Decoder} \tag{4.3.4}$$

$$h_t = f(h_{t-1}, a_{t-1}) \qquad \text{Transition model} \tag{4.3.5}$$

$$r_t = r(h_t, a_t) \qquad \text{Reward model} \tag{4.3.6}$$

Given a dataset of transition observations sampled from the environment $(o_{1..t}a_{1..t-1}, a_t, o_{t+1}, r_t)$ the above models are trained on the following loss function:

$$\begin{aligned} h_t =& b(o_{1..t}a_{1..t-1}) \ , \\ h_{t+1} =& b(o_{1..t+1}a_{1..t}) \ , \\ \mathcal{L}_{rew} =& \frac{1}{2}\Big[r_t(h_t, a_t) - r_t\Big]^2 \ , \\ \mathcal{L}_{decoder} =& \frac{1}{2}\Big[d(h_t) - o_t\Big]^2 \ , \\ \mathcal{L}_{trans} =& \frac{1}{2}||f(\text{StopGradient}(h_t), a_t) - \text{StopGradient}(h_{t+1})||^2 \ . \end{aligned}$$

The total loss sums all the above terms, and can be controlled by the constants $\alpha_{rew}, \alpha_{decoder}, \alpha_{trans}$,

$$\mathcal{L}_{models} = \alpha_{rew}\mathcal{L}_{rew} + \alpha_{decoder}\mathcal{L}_{decoder} + \alpha_{trans}\mathcal{L}_{trans} \ . \tag{4.3.7}$$

The decoder model acts as a regularizer, and provides a training signal beyond simple reward prediction for the latent model.

4.3.3. Latent Memory Stochastic Value Gradient

The latent memory stochastic value (LM-SVG) gradient algorithm is a special case of L-SVG, where the latent model is augmented by a memory module. Recent work using a memory-augmented representation model in deep learning [**35, 33, 74**] has shown the value

of using attention and memory in recurrent networks for sequence modeling, especially in long-horizon tasks. No restrictions were made to the structure of the latent model in the previous section, as long as it can model a sequence of observations into a single latent representation.

It is therefore natural to use a memory augmented recurrent network for the encoder b to model tasks for which memory might be beneficial. For the purposes of this work, we use the SAB-augmented LSTM as described in [**35**] for the encoder in LM-SVG. The high level parameterization of this encoder is

$$\begin{aligned} h_m^t &:= att(h_{i<t}) \ , \\ h_t &:= b(h_{t-1}, h_m^t, o_t a_{t-1}) \ , \end{aligned}$$

where $att(h_{i<t})$ selects k latent states from the past, summarizes them, and passes it along the encoder b in order to better predict h_t. A full description of the the SAB-augmented LSTM can be found in [**35**].

While it may be appealing to simply substitute the SAB-augmented LSTM as the encoder for L-SVG, the resulting latent states are no longer appropriate for latent planning under the Markovian assumption. That is, $p(h_t|h_{i<t}) \neq p(h_t|h_{t-1})$ under a memory augmented recurrent network. Moreover, we have already seen and discussed the benefits of gradient truncation during policy optimization. Truncating the gradients results in a myopic agent, no matter the accuracy of the model employed. To summarize, there are two fundamental problems with using an SAB-augmented LSTM for latent planning:

(1) **The resulting latent space is non-markovian**. The transition model is therefore ill-defined, and no longer has access to all the information needed for predicting the next latent state.

(2) **Fails to account for long-term credit assignment during policy optimization.** Learning a better model is an auxiliary task belonging to supervised learning. It does not directly address the long-term credit assignment issues of backpropagating through the value function.

4.3.4. Latent Sparse Attentive Value Gradient

The latent sparse attentive value gradient (LSAVG) algorithm fully integrates a memory and attention augmented recurrent network into policy optimization. By leveraging the rich structure learned during model fitting, we claim that LSAVG can perform efficient long-term credit assignment while being compatible with the discounting paradigm of modern RL. More precisely LSAVG seeks to **solve Type 1 information acquisition tasks using truncated value gradients**.

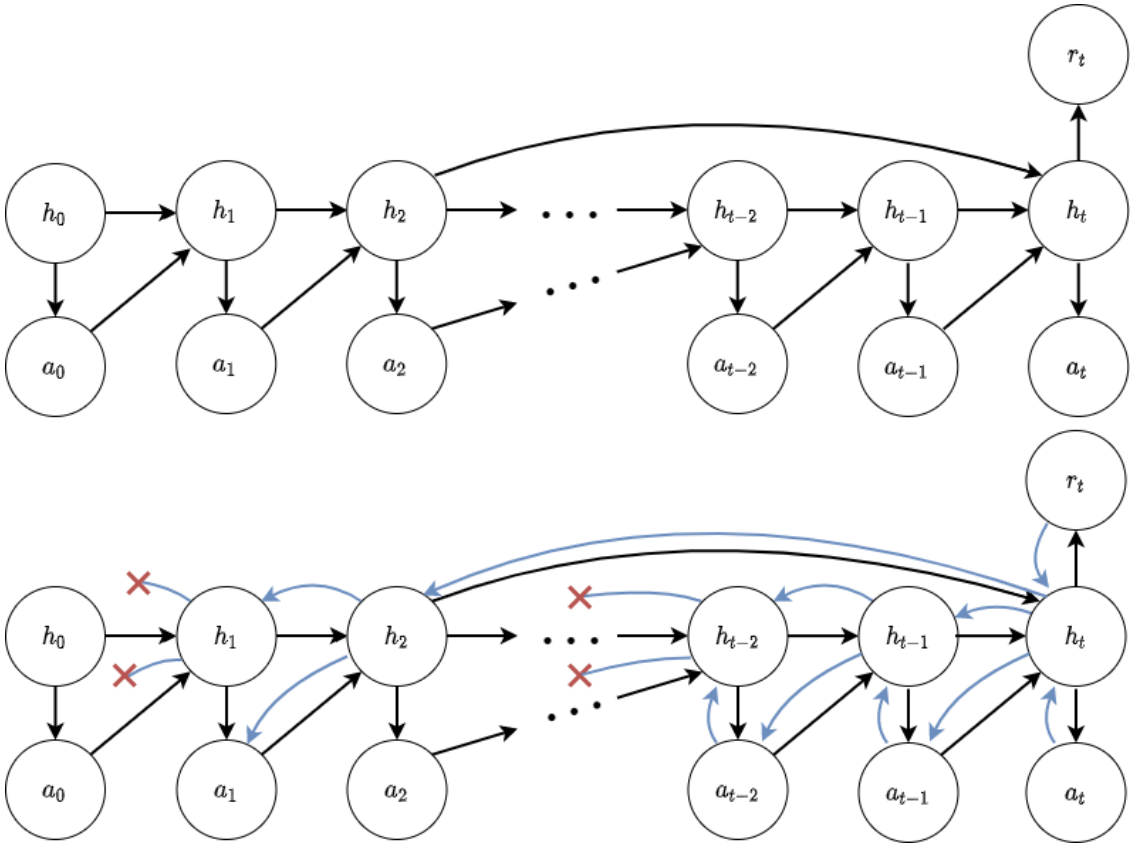

Fig. 4.8. LSAVG forward and backward passes. An access to memory from $m = 2$ to t results in a skip connection in the forward pass, allowing for credit to be directly assigned from m to t.

LSAVG starts by using the same encoder proposed for LM-SVG. The only modification we make to LM-SVG to obtain the latent sparse attentive value gradient algorithm is to include h_m^t as an argument to the transition model f. In doing so, (1) the belief state $h_t + h_m^t$ becomes Markovian again, and (2) skip connections are made during the forward pass of the value function for long-term credit assignment. The credit assignment procedure of LSAVG can be seen in Figure 4.8, which demonstrates how long-term dependencies can still be learned with gradient truncation through the skip connections learned by the memory augmented LSTM. The complete description of LSAVG is shown in Algorithm 2.

We claim that LSAVG is capable of **efficient long-term credit assignment in partially observable settings** if:

- LSAVG is compatible with value gradient truncation
- LSAVG is capable of solving type 2 information acquisition tasks
- LSAVG is capable of solving type 1 information acquisition tasks

We demonstrate these three points by running a truncated version of LSAVG on the previously described type 1 and type 2 memory tasks, and compare its performance with LM-SVG.

Algorithm 2: Latent Sparse Attentive Value Gradient

Input:

$b(h_t, w_t \mid o_{1..t}, a_{1..t-1})$	Attention Encoder	T	Episode length
$d(o_t \mid h_t)$	Decoder	N	Truncation length
$f(h_{t+1} \mid h_t, h_m, a_t, \xi)$	Transition Model		
$r(r_t \mid h_t, a_t)$	Reward Model		
$\pi_\theta(a_t \mid h_t, \eta)$	Policy		

while *not converged* **do**
  Observe initial observation o_1;
  for $t \leftarrow 1$ **to** T **do**
    $h_t \leftarrow b(o_{1..t}, a_{1..t-1})$ from Algorithm 3;
    $a_t \leftarrow \pi_\theta(h_t, \eta_t), \eta_t \sim \rho(\eta)$;
    Take action a_t and observe o_{t+1}, r_t from the environment ;
    Insert (o_t, a_t, r_t, o_{t+1}) into $\mathcal{D}$;
  end
  Train environment models $f(h_t, h_m, a_t, \xi)$ and $r(h_t, a_t)$ with $\mathcal{D}$;
  Initialize latent state $\hat{h}_1 \leftarrow b(o_1 a_0)$;
  Initialize memory: $\mathcal{M} \in \mathbb{R}^{T \times |h|}$, $\mathcal{M}[1] \leftarrow \hat{h}_1$;
  Initialize return: $G \leftarrow 0$;
  for $t \leftarrow 1$ **to** T **do**
    Infer $\xi, w_t \mid o_{t-1}, a_{t-1}, o_t$ following Algorithm 3 and 4.2.7;
    $h_m \leftarrow \sum_{w_t[i] \in w_t} w_t[i]\, \mathcal{M}[i]$;
    $\hat{a}_t \leftarrow \pi_\theta(\hat{h}_t, \eta_t)$, $\hat{r}_t \leftarrow r(\hat{h}_t, \hat{a}_t)$;
    $\hat{h}_{t+1} \leftarrow f(\hat{h}_t, h_m, \hat{a}_t, \xi)$;
    $G \leftarrow G + \hat{r}_t$;
    if $t + 1 \mod N == 0$ **then**
      $\hat{h}_t \leftarrow$`Detach(`$\hat{h}_t$`)`;
    end
    $\mathcal{M}[t+1] \leftarrow \hat{h}_t$;
  end
  Update π_θ using $\nabla_\theta G$;
end

4.4. Experiments For LSAVG

We test the capabilities of truncated LSAVG compared to truncated LM-SVG on three different memory tasks in order of complexity: dummy-PMT, dummy-AMT, and pendulum-AMT. For every task, $t_1 = 3, t_2 = 10$, and $T = 15$. That is, the remember phase is comprised of time steps 0 through 2, the distractor phase includes time steps 4 through 9, and the recall phase encompasses time steps 11 to 14. For both PMT and AMT tasks, the reward function is as defined in 4.3.1, where $c = 0.5$. This means that the maximum possible return for each

episode is $0.5 \times 4 = 2$. All algorithms are subjected to a truncation length of $N = 4$, which means that long-term credit assignment from the recall phase to the remember phase in type 2 information tasks is impossible in the traditional sense. The results of truncated LM-SVG and truncated LSAVG for all three tasks are shown in Figure 4.9.

4.4.1. Passive Memory Credit Assignment

In dummy-PMT, both LM-SVG and LSAVG are capable of solving the task. All that is required in this task is to produce state representations h_t such that $r(h_t, a_t)$ can make accurate predictions. No credit needs to be assigned to past time steps, since the memory state in phase 1 is observed no matter what. We can also see that the memory-augmented encoder is making proper use of attention during the recall phase. An argument can be made that LM-SVG outperforms LSAVG in terms of variance and efficiency in Figure 4.9. A possible explanation for this is due to the added connections in the transition model that effectively makes the model deeper than its LM-SVG counterpart, resulting in more unstable gradients. Until now, the behaviors of the algorithms are unsurprising, as [**33**] and [**74**] have already demonstrated that a memory augmented LSTM is capable of solving type 1 information acquisition tasks. Those results are extended here for a model-based algorithm.

4.4.2. Active Memory Credit Assignment

In both dummy-AMT and pendulum-AMT, the truncated LM-SVG model is incapable of obtaining returns beyond what a perfect myopic agent could achieve. The maximum return of a myopic agent in both settings is based on a policy that perfectly acts in phases 2 and 3 according to the noisy observation in 4.3.2, but acts completely randomly in the first phase. Therefore, this agent is able to act accordingly during the recall phase only if it randomly happened to perform the correct action(s) in phase 1. Interestingly, truncated LM-SVM seemingly approaches this myopic maximum, but is unable to surpass it. Conversely, truncated LSAVG reliably approaches the maximum possible return of the active memory task. Additionally, the attention weights for the remember phase obtained at $t = 12$ (the recall phase) are larger than those obtained at $t = 8$ (the distractor phase), suggesting that the agent is actively *recalling* appropriate memories when necessary.

If we direct our attention to Figure 4.10, we can start to appreciate how LM-SVG (1) has an ill-defined transition model, and (2) fails to account for long-term credit assignment. Regarding the first point, we can see that LM-SVG struggles to learn a good transition model when compared to LSAVG. Secondly, LM-SVG is just as capable as LSAVG in performing short-term credit assignment, since its performance on the distractor pendulum phase is comparable to LSAVG. The discrepancy in overall performance is therefore due to the active memory task.

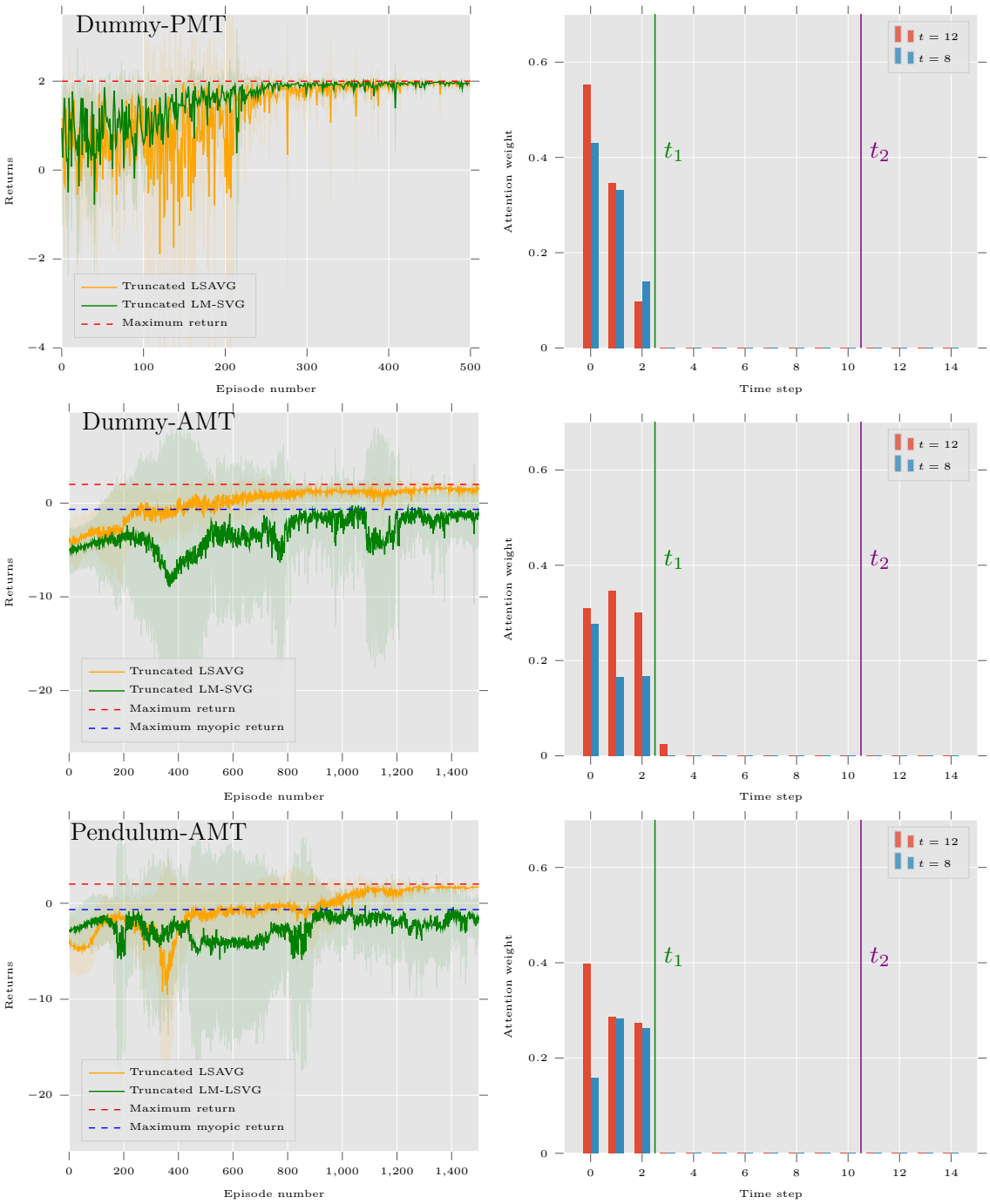

Fig. 4.9. Results of truncated LSAVG and truncated LM-SVG on dummy-PMT, dummy-AMT and pendulum-AMT. On the left, the return is plotted against the number of episodes during training. On the right, the attention weights for LSAVG are shown with their corresponding tasks. In red are the time steps recalled during the recall phase ($t = 12$), and in blue are the time steps recalled during the distractor phase ($t = 8$).

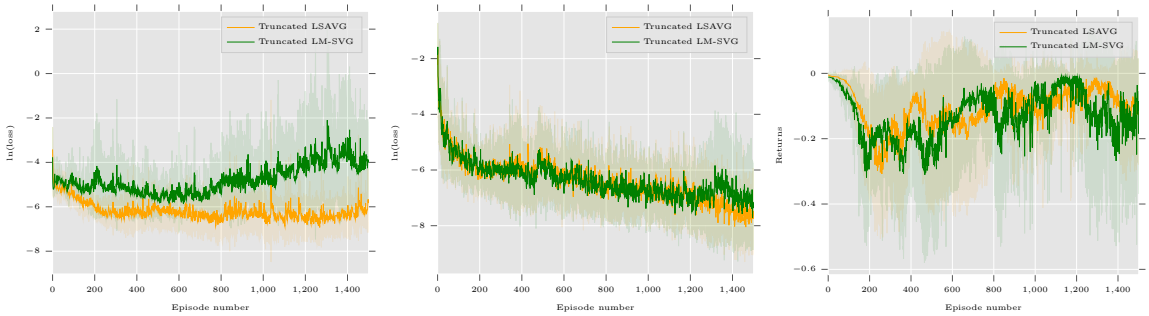

Fig. 4.10. From left to right, the $ln(loss)$ of the transition model, the $ln(loss)$ of the reward model, and the returns on the pendulum task within AMT are shown. LM-SVG is incapable of learning a good transition model, but sees similar performance for learning the reward model and acting in the distractor phase when compared to LSAVG.

4.5. Related Work

The work in this chapter builds on a number of of recent papers studying the problem of long-term credit assignment, most notably the sparse attentive backtracking network proposed in [**35**], and the temporal value transport mechanism of [**33**]. Let us elaborate further on the differences and similarities LSAVG has with SAB and TVT.

4.5.1. Sparse Attentive Backtracking

The computation graph for sparse attentive backtracking closely resembles that of LSAVG. As pointed out in section 4.1, we believe that a lot of the benefits of SAB for long-term credit assignment can be translated into value gradients, which is what we sought out to show in this chapter. Moreover, LSAVG directly makes use of a SAB network for representation learning. The key difference in LSAVG is that the skip connections are learned on an auxiliary task, reward prediction, and recycled during policy optimization. It is not yet clear at the moment if attention can be learned to directly optimize returns, and we leave that for future work.

4.5.2. Temporal Value Transport

Just like SAB and MERLIN [**74**], LSAVG and TVT use a memory augmented network to learn state representations. The novelty in TVT is in reusing the attention weights learned for latent representation to train the policy's action distribution. Given the attention weight w_t^m telling the latent network to use a memory at time step $m < t$ to predict the state s_t, TVT heuristically transports value from the t to m during policy optimization (in an undiscounted setting):

$$V_m = r_m + V_{m+1} + w_t^m V_t \tag{4.5.1}$$

How does this fit in the model-based setting? If we modify the value function accordingly in equation 4.2.1[2], then the recurrent relationship for the value gradient estimate becomes

$$\frac{\partial V^m(s_m;\theta)}{\partial s_m} = \frac{\partial r_m}{\partial s_m} + \frac{\partial r_m}{\partial a_m}\frac{\partial a_m}{\partial s_m} + \left(\frac{\partial s_{m+1}}{\partial s_m} + \frac{\partial s_{m+1}}{\partial a_m}\frac{\partial a_m}{\partial s_m}\right)\frac{\partial V^{m+1}(s_{m+1};\theta)}{\partial s_{m+1}} + \frac{\partial \mathbf{V^t}(\mathbf{s_t};\theta)}{\partial \mathbf{s_t}} \ ,$$

for the recalled time step m. LSAVG instead modifies the recurrent relationship for the forward value gradient. In the case where recalled information is strictly additive, $s_t = f(s_{t-1}, a_{t-1}) + s_m$, the recurrence for time step t changes to

$$\frac{ds_t}{d\theta} = \frac{\partial s_t}{\partial a_{t-1}}\frac{\partial a_{t-1}}{\partial \theta} + \left(\frac{\partial s_t}{\partial s_{t-1}} + \frac{\partial s_t}{\partial a_{t-1}}\frac{\partial a_{t-1}}{\partial s_{t-1}}\right)\frac{ds_{t-1}}{d\theta} + \frac{\mathbf{ds_m}}{\mathbf{d}\theta} \ .$$

LSAVG is also not forced into an additive recall step, as s_m is used however deemed necessary to better predict s_t, and the resulting function approximation is automatically differentiated through.

It is important to note that equation 4.5.1 results in a biased estimator of the policy gradient. Unlike TVT, backpropagating through the skip connections of the transition model does not bias the gradient, since we are not making any changes to the underlying auto-differentiation mechanism. Therefore, LSAVG can be seen as an unbiased generalisation of TVT in the model-based setting. Interestingly, the authors of TVT themselves note a biological bias towards model-based reasoning in humans, which might motivate the use of a model-based alternative.

[2]Remember that value functions for policy gradient based algorithms are only used for derivative estimation.

Chapter 5

Conclusion and Future Work

Credit assignment plays an integral role in machine learning. By correctly identifying what was responsible for certain outcomes, it guides autonomous agents towards optimal behaviors. In reinforcement learning, CA generally takes one of two forms: model-free learning and model-based learning. Unfortunately, whatever the approach, agents struggle to perform efficient and expressive credit assignment over long horizons. In the latter case, variance quickly grows over long time spans, and in the former, errors in the model propagating through time either exaggerate or underestimate long-term dependencies. Intuitively, humans are able to reason over extended periods of time because the effects of an event at time t on a future event at t' can be directly inferred without necessarily having recourse to every event between t and t'. Concretely, counterfactuals and attention allow for this form of sparse reasoning.

Counterfactuals answer questions like *what if x' had happened instead of x?* By reasoning solely about X, it is easier to isolate its effects, and consequently perform credit assignment over X. In fact, it turns out that counterfactuals of this form for policy evaluation in RL, called counterfactual policy evaluation, can be shown to be a variance reduction technique relating to the conditional Monte Carlo method and importance sampling. Existing approaches to policy evaluation can be seen through this lens, which helps better understand some of their theoretical properties. In this thesis, we showed that HCA [**28**] and tree backup [**56**] are specific cases of counterfactual policy evaluation. We leave for future work the possibility of discovering or rediscovering other estimators that may also be seen as a form counterfactual policy evaluation. We also leave room for a deeper theoretical analysis of when and how counterfactual policy evaluation is the most effective.

Attention mechanisms were originally proposed to allow models to focus on a specific subset of inputs to better assign credit where needed [**3**]. Latent sparse attentive value gradients leverage the rich structure induced by such attention mechanisms for sparse temporal credit assignment in model-based RL. To the best of our knowledge, this is the first of its kind, and it is demonstrated to be able to solve the same types of tasks TVT, its model-free

counterpart, was designed to solve. Value gradient based algorithms have historically had issues scaling to larger more complex environments [**30, 1**], and can benefit greatly from using an approximate value function. A natural followup to LSAVG is to scale it to more difficult problems, and to see how critics and value functions might translate into this framework.

References

[1] Brandon Amos, Samuel Stanton, Denis Yarats et Andrew Gordon Wilson : On the model-based stochastic value gradient for continuous reinforcement learning, 2021.

[2] Pierre-Luc Bacon, Jean Harb et Doina Precup : The option-critic architecture, 2016.

[3] Dzmitry Bahdanau, Kyunghyun Cho et Yoshua Bengio : Neural machine translation by jointly learning to align and translate, 2016.

[4] Elias Bareinboim, Andrew Forney et Judea Pearl : Bandits with unobserved confounders: A causal approach. *In* C. Cortes, N. Lawrence, D. Lee, M. Sugiyama et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[5] J. Baxter et P. L. Bartlett : Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, Nov 2001.

[6] Yoshua Bengio : How auto-encoders could provide credit assignment in deep networks via target propagation. *CoRR*, abs/1407.7906, 2014.

[7] Yoshua Bengio : The consciousness prior, 2019.

[8] Paul Bratley, Bennett L. Fox et Linus E. Schrage : *A Guide to Simulation (2Nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1987.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba : Openai gym, 2016.

[10] James A Bucklew : Conditional importance sampling estimators. *IEEE transactions on information theory*, 51(1):143–153, 2005.

[11] Lars Buesing, Theophane Weber, Yori Zwols, Sebastien Racaniere, Arthur Guez, Jean-Baptiste Lespiau et Nicolas Heess : Woulda, coulda, shoulda: Counterfactually-guided policy search, 2018.

[12] Joseph L. Doob : *Measure theory*. Springer, 1994.

[13] Pierluca D'Oro et Wojciech Jaśkowski : How to learn a useful critic? model-based action-gradient-estimator policy optimization, 2020.

[14] Jeffrey L. Elman : Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[15] Michael Fairbank et Eduardo Alonso : Value-gradient learning. *In The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

[16] A. Forney et E. Bareinboim : Counterfactual randomization: Rescuing experimental studies from obscured confounding. *In AAAI*, 2019.

[17] Andrew Forney, Judea Pearl et Elias Bareinboim : Counterfactual data-fusion for online reinforcement learners. *In* Doina Precup et Yee Whye Teh, éditeurs : *Proceedings of the 34th International Conference on Machine Learning*, volume 70 de *Proceedings of Machine Learning Research*, pages 1156–1164. PMLR, 06–11 Aug 2017.

[18] Scott Fujimoto, Herke Hoof et David Meger : Addressing function approximation error in actor-critic methods. *In International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[19] Carles Gelada et Marc G. Bellemare : Off-policy deep reinforcement learning by bootstrapping the covariate shift. *In The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 3647–3655, 2019.

[20] Ian Goodfellow, Yoshua Bengio et Aaron Courville : *Deep learning*. MIT Press, 2016.

[21] Google : Explore what the world is searching.

[22] Anirudh Goyal, Aniket Didolkar, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Heess, Michael Mozer et Yoshua Bengio : Neural production systems, 2021.

[23] A. Griewank : Who invented the reverse mode of differentiation. 2012.

[24] Andreas Griewank : A mathematical view of automatic differentiation. *Acta Numerica*, 12:321–398, 2003.

[25] Danijar Hafner, Timothy Lillicrap, Jimmy Ba et Mohammad Norouzi : Dream to control: Learning behaviors by latent imagination, 2020.

[26] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee et James Davidson : Learning latent dynamics for planning from pixels, 2019.

[27] Assaf Hallak et Shie Mannor : Consistent on-line off-policy evaluation. *In Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1372–1383, 2017.

[28] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup et Rémi Munos : Hindsight credit assignment. *In* Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox et Roman Garnett, éditeurs : *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 12467–12476, 2019.

[29] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield et Matthew Botvinick : Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

[30] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez et Yuval Tassa : Learning continuous control policies by stochastic value gradients. *In* C. Cortes, N. Lawrence, D. Lee, M. Sugiyama et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[31] Timothy C. Hesterberg : *Advances in Importance Sampling*. Thèse de doctorat, Stanford University, août 1988.

[32] Sepp Hochreiter et Jürgen Schmidhuber : Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 11 1997.

[33] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja et Greg Wayne : Optimizing agent behavior over long time scales by transporting value, 2018.

[34] Leslie Pack Kaelbling, Michael L. Littman et Anthony R. Cassandra : Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[35] Nan Rosemary Ke, Anirudh Goyal, Olexa Bilaniuk, Jonathan Binas, Michael C. Mozer, Chris Pal et Yoshua Bengio : Sparse attentive backtracking: Temporal creditassignment through reminding, 2018.

[36] Diederik P Kingma et Max Welling : Auto-encoding variational bayes, 2014.

[37] Vijay Konda et John Tsitsiklis : Actor-critic algorithms. *Society for Industrial and Applied Mathematics*, 42, 04 2001.

[38] Benjamin James Lansdell, Prashanth Ravi Prakash et Konrad Paul Kording : Learning to solve the credit assignment problem, 2020.

[39] Pierre Lecuyer : A unified view of the ipa, sf, and lr gradient estimation techniques. *Management Science*, 36(11):1364–1383, 1990.

[40] Pierre L'Ecuyer : Efficiency improvement and variance reduction. *In Proceedings of the 26th Conference on Winter Simulation*, WSC '94, pages 122–132, San Diego, CA, USA, 1994. Society for Computer Simulation International.

[41] David Lewis : *Counterfactuals*. Harvard University Press, 1973.

[42] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver et Daan Wierstra : Continuous control with deep reinforcement learning, 2019.

[43] S. Linnainmaa : Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16:146–160, 1976.

[44] Michael L. Littman, Richard S. Sutton et Satinder P. Singh : Predictive representations of state. *In* Thomas G. Dietterich, Suzanna Becker et Zoubin Ghahramani, éditeurs : *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1555–1561. MIT Press, 2001.

[45] Qiang Liu, Lihong Li, Ziyang Tang et Dengyong Zhou : Breaking the curse of horizon: Infinite-horizon off-policy estimation. *In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 5361–5371, 2018.

[46] Yao Liu, Pierre-Luc Bacon et Emma Brunskill : Understanding the curse of horizon in off-policy evaluation via conditional importance sampling, 2020.

[47] Ashique Rupam Mahmood, Huizhen Yu et Richard S. Sutton : Multi-step off-policy learning without importance sampling ratios. *CoRR*, abs/1702.03006, 2017.

[48] M.W. Mak, K.W. Ku et Y.L. Lu : On the improvement of the real time recurrent learning algorithm for recurrent neural networks. *Neurocomputing*, 24(1):13–36, 1999.

[49] Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, Marcus Hutter, Lars Buesing et Rémi Munos : Counterfactual credit assignment in model-free reinforcement learning, 2020.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski *et al.* : Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[51] Rémi Munos, Tom Stepleton, Anna Harutyunyan et Marc G. Bellemare : Safe and efficient off-policy reinforcement learning. *In* Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon et Roman Garnett, éditeurs : *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1046–1054, 2016.

[52] J. Panetta et K. Fister : Optimal control applied to competing chemotherapeutic cell-kill strategies. *SIAM J. Appl. Math.*, 63:1954–1971, 2003.

[53] Razvan Pascanu, Tomas Mikolov et Yoshua Bengio : On the difficulty of training recurrent neural networks, 2013.

[54] Judea Pearl : *Synthese*, 121(1/2):93–149, 1999.

[55] Judea Pearl et Dana Mackenzie : *The book of why: the new science of cause and effect*. Basic Books, 2020.

[56] Doina Precup, Richard S. Sutton et Satinder P. Singh : Eligibility traces for off-policy policy evaluation. *In* Pat Langley, éditeur : *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 759–766. Morgan Kaufmann, 2000.

[57] Martin L. Puterman : *Markov decision processes: discrete stochastic dynamic programming*. John Wiley Sons, 2010.

[58] F. Rosenblatt : *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory, 1957.

[59] Mark Rowland, Anna Harutyunyan, Hado van Hasselt, Diana Borsa, Tom Schaul, Rémi Munos et Will Dabney : Conditional importance sampling for off-policy learning. *In* Silvia Chiappa et Roberto Calandra, éditeurs : *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 de *Proceedings of Machine Learning Research*, pages 45–55. PMLR, 2020.

[60] Reuven Y. Rubinstein : *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., New York, NY, USA, 1st édition, 1981.

[61] David E. Rumelhart, Geoffrey E. Hinton et Ronald J. Williams : *Learning internal representations by error propagation*. Institute for Cognitive Science, University of California, San Diego, 1986.

[62] A. L. Samuel : Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, juillet 1959.

[63] Jürgen Schmidhuber : Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[64] Robert J. Serfling : *Approximation theorems of mathematical statistics*. John Wiley Sons, 2002.

[65] Richard S. Sutton : *Temporal credit assignment in reinforcement learning*. 1984.

[66] Richard S. Sutton : Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *In* Bruce Porter et Raymond Mooney, éditeurs : *Machine Learning Proceedings 1990*, pages 216–224. Morgan Kaufmann, San Francisco (CA), 1990.

[67] "Richard S. Sutton et Andrew G. Barto : *Reinforcement Learning, an Introduction*. The MIT Press, Cambridge, Massachusetts, 2018.

[68] Richard S. Sutton, David McAllester, Satinder Singh et Yishay Mansour : Policy gradient methods for reinforcement learning with function approximation. *In Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[69] Richard S. Sutton, Doina Precup et Satinder Singh : Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[70] Yujin Tang, Duong Nguyen et David Ha : Neuroevolution of self-interpretable agents. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, Jun 2020.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser et Illia Polosukhin : Attention is all you need, 2017.

[72] Wenguan Wang et Jianbing Shen : Deep visual attention prediction. *IEEE Transactions on Image Processing*, 27(5):2368–2378, May 2018.

[73] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker et Martin Riedmiller : Embed to control: A locally linear latent dynamics model for control from raw images, 2015.

[74] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis et Timothy Lillicrap : Unsupervised predictive memory in a goal-directed agent, 2018.

[75] N. A. Weiss, Paul T. Holmes et Michael Hardy : *A course in probability*. Pearson Addison Wesley, 2006.

[76] P.j. Werbos : Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[77] Ronald J. Williams : Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement Learning*, page 5–32, 1992.

[78] Ronald J. Williams et D. Zipser : Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1:87–111, 1989.

[79] Ronald J. Williams et David Zipser : *Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity*, page 433–486. L. Erlbaum Associates Inc., USA, 1995.

[80] Sanghyun Woo, Jongchan Park, Joon-Young Lee et In So Kweon : Cbam: Convolutional block attention module, 2018.

[81] Tengyang Xie, Yifei Ma et Yu-Xiang Wang : Optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. *In Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, 2019.

[82] Junzhe Zhang et E. Bareinboim : Markov decision processes with unobserved confounders : A causal approach. 2016.

[83] K.j Åström : Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

Appendix A

Proofs for Chapter 2

A.1. Proof of Theorem 3.3.1

Proof. Lemma 1. *Given any state s, action a, target policy π, and behavior policy μ for which $\mu(a_t|s_t) > 0$ for all a_t and s_t:*

$$Q^\pi(s,a) = \mathbb{E}_\mu \left[R_0 + \sum_{t=1}^{T} \gamma^T R_t \mathcal{P}_{1:t}^{\pi,\mu} \middle| S_0 = s, A_0 = a \right] = \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \mathcal{P}_{1:t}^{\pi,\mu} \middle| S_0 = s, A_0 = a \right]$$

Proof. By per decision importance sampling:

$$V^\pi(s) = \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \prod_{k=0}^{t} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \middle| S_0 = s \right]$$

By the law of total expectation

$$= \mathbb{E}_\mu \left[\mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \prod_{k=0}^{t} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \middle| S_0 = s, A_0 \right] \middle| S_0 = s \right]$$

$$= \sum_{a \in \mathcal{A}} \mu(a|s) \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \prod_{k=0}^{t} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \middle| S_0 = s, A_0 = a \right]$$

$$= \sum_{a \in \mathcal{A}} \mu(a|s) \mathbb{E}_\mu \left[\frac{\pi(A_0|S_0)}{\mu(A_0|S_0)} \sum_{t=0}^{T} \gamma^t R_t \prod_{k=1}^{t} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \middle| S_0 = s, A_0 = a \right]$$

$$= \sum_{a \in \mathcal{A}} \mu(a|s) \frac{\pi(a|s)}{\mu(a|s)} \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \prod_{k=1}^{t} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \middle| S_0 = s, A_0 = a \right]$$

$$= \sum_{a \in \mathcal{A}} \pi(a|s) \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \prod_{k=1}^{t} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} \middle| S_0 = s, A_0 = a \right]$$

By definition, $V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s,a)$, therefore

$$Q^\pi(s,a) = \mathbb{E}_\mu \left[\sum_{t=0}^{T} \gamma^t R_t \mathcal{P}_{1:t}^{\pi,\mu} \middle| S_0 = s, A_0 = a \right]$$

□

Lemma 2. *Consider the MDP setting, where random variables $S_i, A_i, \ldots, S_t, A_t$ are actions and states visited by a policy μ. Let $\mu(a|s)$ be the conditional probability of taking action a at state s, then:*

$$\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)\right]$$
$$= \mathbb{E}_\mu\left[\sum_{\alpha \neq A_i} \mu(\alpha|S_i)\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)\middle| A_i = \alpha\right] + \mu(A_i|S_i)\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)\right]$$

Proof.

$$\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)\right]$$

By the law of total expecation

$$= \mathbb{E}_\mu\left[\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)\middle| A_i\right]\right]$$
$$= \sum_{\alpha \in \mathcal{A}} \mathbb{P}_\mu(A_i' = \alpha)\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)|A_i = \alpha\right]$$
$$= \mathbb{E}_\mu\left[\sum_{\alpha \in \mathcal{A}} \mathbb{P}_\mu(A_i' = \alpha)\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)|A_i = \alpha\right]\right]$$
$$= \mathbb{E}_\mu\left[\sum_{\alpha \neq A_i} \mathbb{P}_\mu(A_i' = \alpha)\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)|A_i = \alpha\right] + \mathbb{P}_\mu(A_i' = A_i)\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)|A_i = A_i\right]\right]$$
$$= \mathbb{E}_\mu\left[\sum_{\alpha \neq A_i} \mu(\alpha|S_i)\mathbb{E}_\mu\left[\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)|A_i = \alpha\right] + \mu(A_i|S_i)\sum_{t=i}^{T} f(S_i, A_i, \ldots, S_t, A_t)\right]$$

□

Back to Theorem 1

Consider first conditioning on A_1

$$Q^\pi(s,a) = \mathbb{E}_\mu\left[R_0 + \sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{1:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\right]$$

By Lemma 2 on the second term, where $i = 1$

$$= \mathbb{E}_\mu\Bigg[R_0 + \sum_{\alpha\neq A_1}\mu(\alpha|S_1)\mathbb{E}_\mu\left[\sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{1:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a, A_1 = \alpha\right]$$

$$+ \mu(A_1|S_1)\sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{1:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\Bigg]$$

$$= \mathbb{E}_\mu\Bigg[R_0 + \sum_{\alpha\neq A_1}\mu(\alpha|S_1)\frac{\pi(\alpha|S_1)}{\mu(\alpha|S_1)}\mathbb{E}_\mu\left[\sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{2:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a, A_1 = \alpha\right]$$

$$+ \mu(A_1|S_1)\frac{\pi(A_1|S_1)}{\mu(A_1|S_1)}\sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{2:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\Bigg]$$

$$= \mathbb{E}_\mu\Bigg[R_0 + \sum_{\alpha\neq A_1}\pi(\alpha|S_1)\mathbb{E}_\mu\left[\sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{2:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a, A_1 = \alpha\right]$$

$$+ \pi(A_1|S_1)\sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{2:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\Bigg]$$

And by Lemma 1

$$= \mathbb{E}_\mu\Bigg[R_0 + \sum_{\alpha\neq A_1}\pi(\alpha|S_1)\gamma Q^\pi(S_1,\alpha) + \gamma\pi(A_1|S_1)\sum_{t=1}^{T}\gamma^{t-1} R_t \mathcal{P}_{2:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\Bigg]$$

Now, since $\pi(A_1|S_1)$ is independent of any future actions or states, we may do the same thing onto the last term, conditioning on A_2 now to obtain:

$$Q^\pi(s,a) = \mathbb{E}_\mu\left[R_0 + \sum_{t=1}^{T}\gamma^t R_t \mathcal{P}_{1:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\right]$$

$$= \Bigg[R_0 + \sum_{\alpha\neq A_1}\pi(\alpha|S_1)\gamma Q^\pi(S_1,\alpha)$$

$$+ \gamma\pi(A_1|S_1)\Bigg(R_1 + \sum_{\alpha\neq A_2}\gamma\pi(\alpha|S_2)Q^\pi(S_2,\alpha) + \gamma\pi(A_2|S_2)\sum_{t=2}^{T}\gamma^{t-2}R_t\mathcal{P}_{3:t}^{\pi,\mu}\Bigg)\Bigg| S_0 = s, A_0 = a\Bigg]$$

$$= \mathbb{E}_\mu\Bigg[\sum_{t=0}^{1}\gamma^t\prod_{k=0}^{t}\pi(A_k|S_k)(R_t + \gamma\sum_{\alpha\neq A_{t+1}}\pi(\alpha|S_{t+1})Q^\pi(S_{t+1},\alpha))$$

$$+ \gamma^2\prod_{k=1}^{2}\pi(A_k|S_k)\sum_{t=2}^{T}\gamma^{t-2}R_t\mathcal{P}_{3:t}^{\pi,\mu}\middle| S_0 = s, A_0 = a\Bigg]$$

Continuously applying the same logic up to A_T results in:

$$Q^\pi(s,a) = \mathbb{E}_\mu\left[\sum_{t=0}^{T}\gamma^t\prod_{k=0}^{t}\pi(A_k|S_k)(R_t + \gamma\sum_{\alpha\neq A_{t+1}}\pi(\alpha|S_{t+1})Q^\pi(S_{t+1},\alpha))\middle| S_0 = s, A_0 = a\right]$$

Notice that the conditional trick is not applied to terms in the time steps before A_i, therefore:

$$
\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}_{\mu}\left[\sum_{t=0}^{T}\gamma^{t}\mathbb{E}_{\mu}\left[R^{t}\mathcal{P}_{1:t}^{\pi,\mu}\middle|S_0=s,A_0=a,A_1,A_2,\ldots,A_t\right]\middle|S_0=s,A_0=a\right] \\
&= \mathbb{E}_{\mu}\left[\sum_{t=0}^{T}\gamma^{t}\prod_{k=1}^{t}\pi(A_k|S_k)(R_t+\gamma\sum_{\alpha\neq A_{t+1}}\pi(\alpha|S_{t+1})Q^{\pi}(S_{t+1},\alpha))\middle|S_0=s,A_0=a\right]
\end{aligned}
$$

□

Appendix B

Additional Results

B.1. Pendulum and Cancer

Horizon parameter	Solved time step	Max gradients	Final gradients
$N = 5$	116.40 ± 5.60	253.15 ± 55.70	0.07 ± 0.04
$N = 10$	97.8 ± 14.19	639.59 ± 88.50	0.10 ± 0.06
$N = 20$	120.4 ± 23.01	2142.58 ± 194.33	0.17 ± 0.06
$N = 40$	156.40 ± 19.02	9450.35 ± 2013.10	0.20 ± 0.04
$N = 60$	185.00 ± 32.52	29584.90 ± 9039.43	0.16 ± 0.06
$N = 100$	–	1680621.60 ± 992127.25	9777.83 ± 5939.89
$\lambda = 0.9$	94.20 ± 31.17	320.19 ± 168.80	0.01 ± 0.01
$\lambda = 0.99$	240.2 ± 33.38	60988.41 ± 12155.56	0.52 ± 0.46
$\lambda = 1$	–	1680621.60 ± 992127.25	9777.83 ± 5939.89

Table B.1. Results of T-SVG(∞) for different truncation lengths and discount factors in the pendulum environment. Episodes are 100 time steps long, and the confidence intervals represent a 90% confidence bound.

Horizon parameter	Return	Max gradients	Final gradients
$N = 10$	-216.37 ± 0.39	339.07 ± 44.37	3.59 ± 1.10
$N = 20$	-185.54 ± 0.36	838.79 ± 226.28	3.45 ± 0.43
$N = 40$	-163.82 ± 0.43	656.28 ± 88.05	7.13 ± 5.88
$N = 60$	-157.86 ± 0.09	20450.7 ± 20329.77	1.58 ± 0.19
$N = 100$	-155.74 ± 0.01	$39331804.0 \pm 39346094.6$	2.09 ± 0.77
$\lambda = 0.9$	-195.13 ± 0.73	3417.46 ± 3867.03	0.37 ± 0.08
$\lambda = 0.99$	-156.57 ± 0.08	$18711126.0 \pm 21392030.9$	1.36 ± 0.22
$\lambda = 1$	-155.74 ± 0.01	$39331804.0 \pm 39346094.6$	2.09 ± 0.77

Table B.2. Results of T-SVG(∞) for different truncation lengths and discount factors in the cancer environment. Episodes are 100 time steps long, and the confidence intervals represent a 90% confidence bound.

B.2. LSAVG and LM-SVG

B.2.1. Memory-augmented encoder

The memory-augmented encoder described and used for LSAVG is a simplification of the SAB network in [**35**]. In fact, we elect to use an architecture closer to a Transformer [**71**] given the simplicity of the tasks. The encoder is described in the following algorithm

Algorithm 3: Procedure for $b(h_t, w_t | o_{1..t}, a_{1..t-1})$

Input:

History $o_{1..t}, a_{1..t-1} \in \mathbb{R}^{t \times d}$

Sparsity k

Attention Key-gen $\mathbf{W} \in \mathbb{R}^{d \times d_k}$

MLP $mlp_\psi : \mathbb{R}^{1 \times d} \to \mathbb{R}^{1 \times d_k}$

$K \leftarrow o_{1..t}, a_{1..t-1}\mathbf{W}$;
$h_t \leftarrow mlp_\psi(o_t a_{t-1})$;
$attn_w \leftarrow h_t K^T$;
$attn_k \leftarrow \text{sorted}(attn_w)[k+1]$;
$attn_k \leftarrow \text{ReLU}(attn_w - attn_k)$;
$h_m \leftarrow \frac{\sum_{i=1}^{t} attn_k[i] o_i a_{i-1}}{\sum_i attn_k[i]}$;
return $h_t + h_m,\ attn_k$