Similarities between policy gradient methods (PGM) in reinforcement learning (RL) and supervised learning (SL)

Eric Benhamou*[1]

[1]A.I Square Connect, Lamsade PSL

May 20, 2019

Abstract

Reinforcement learning (RL) is about sequential decision making and is traditionally opposed to supervised learning (SL) and unsupervised learning (USL). In RL, given the current state, the agent makes a decision that may influence the next state as opposed to SL (and USL) where, the next state remains the same, regardless of the decisions taken, either in batch or online learning. Although this difference is fundamental between SL and RL, there are connections that have been overlooked. In particular, we prove in this paper that gradient policy method can be cast as a supervised learning problem where true label are replaced with discounted rewards. We provide a new proof of policy gradient methods (PGM) that emphasizes the tight link with the cross entropy and supervised learning. We provide a simple experiment where we interchange label and pseudo rewards. We conclude that other relationships with SL could be made if we modify the reward functions wisely.

keywords: Policy gradient, Supervised learning, Cross entropy, Kullback Leibler divergence, entropy.

1 Introduction

In RL, policy gradient methods (PGM) are frequently used [Wil92, SMSM99, SLH$^+$14, LHP$^+$15]. PGM are RL techniques that rely upon optimizing the parameters of policies with respect to the expected cumulative reward using gradient descent optimization. They are traditionally opposed to value learning (or value iterations) methods [SB98, WD92] that either keep improving the value function at each iteration until the value-function converges or optimize the parameters of the value function. The value function is defined as the expected value of the expected cumulative reward conditionally to an initial state and action. PGM principle is very simple. Improve gradually the policy through gradient descent. PGM have two important concepts. They are a policy method as the name emphasizes. They are also a gradient descent method. Policy means we observe and act. Gradient descent methods use the fact that the best move locally is along the gradient. As the move is at first order, a learning rate needs to ensure policy improvement is not too large at each step. PGM have been popularized in REINFORCE [Wil92] and in [SMSM99] and have received wider attention with Actor Critic methods [KT03, PS08] in particular when using deep PGM [MBM$^+$16] that combine policy and value methods. In addition, recently, it has been found that an entropy regularization term may fasten convergence [OMKM16, NNXS17, SAC17].

When looking in details in REINFORCE [Wil92], we can remark that the gradient term with respect to the policy can indeed be interpreted as the log term in the cross entropy in supervised learning. If in addition, we make the bridge between RL and SL, emphasizing that RL problem can be reformulated as a SL problem where true labels are changed by expected discounted future rewards, and estimated probabilities by policy probabilities, the link between RL and SL becomes obvious. In addition, leveraging the tight relationship between cross entropy and Kullback Leibler divergence, we can interpret the entropy regularization terms very naturally. This is precisely the objective of this short paper: call attention to the tight connection between RL and SL to give theoretical justifications of some of the techniques used in PGMs.

The paper is organized as follows. In section 3, we recall the various choices of functional losses in SL and exhibit that cross entropy is one of the main possibil-

*eric.benhamou@dauphine.eu

ities for loss functions. We flaunt the relationship between cross entropy and Kullback Leibler divergence, harping on the additional entropy term. In section 4, we present PGMs. We rub in the interpretation of RL problems as a modified cross entropy SL problem. We conclude in section 5 with a financial numerical experience using deep PGM, stressing that in the specific case of actions that do not influence the environment, the difference between RL and SL is very tenuous.

2 Related Work

Looking at similarities and synergies between RL and SL together was for a long time overlooked. This can be easily explained as the two research communities were different and thought they were labouring on incompatible or at least very different approaches. However, there has been one type of learning that promotes the similitude between RL and SL. This has been Imitation Learning.

Imitation learning [Sch96] is a classic technique for learning from human demonstration. Imitation learning uses a supervised approach to imitate an expert's behaviors, hence doing a RL task to accomplish a SL one. DAGGER [RGB11] is considered to be the mainstream imitation algorithm. It requests an action from the expert at each step. It uses an action sampled from a mixed distribution from the agent and the expert. It combines the observed states and demonstrated actions to train the agent successively. This has led to numerous extensions of this algorithm and in particular to deep version of it. Deeply AggreVaTeD [SVG+17] extends DAGGER with deep neural networks and continuous action spaces. Other approaches have been to stop opposing RL and SL and rather leverage SL in deep RL. [HVP+17] for instance have used SL tasks to train the agent better and created the method of Deep Q-learning from Demonstrations (DQfD). It tackles the problem of the necessity of huge amount of data for deep reinforcement learning (Deep RL) and the poor performance of Deep RL algorithms during initial phase of learning. The method of Deep Q-learning from Demonstrations (DQfD) solves the issue by combining RL techniques (temporal difference updates) with SL techniques (supervised classification of the demonstrator's actions) as the target network in Deep Q learning is initially trained with supervised learning.

All of these works show that RL and SL are not as opposed as one may have thought. We argue here that as nice as these works are, they do not emphasizes that, ignoring for a while the issue of feedback effect of action on next state environment, a PGM can be reformulated as a SL task where true labels are changed into future expected reward while the PG can be interpreted as a cross entropy loss minimization.

3 Supervised Learning

SL is quite general and encompasses both SL classification and SL regression. The goal of SL classification is to infer a function from labeled training data that maps inputs into labeled outputs. In psychology, this is sometimes analyzed as concept learning. The deduction of the function parameters is done traditionally through the optimization of a loss function. SL classifiers parameters are the ones of the optimal solution of the optimization program. To keep things simple, let us assume that we are looking at a binary classification problem. Let us assume we observe $D_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$ that are n independent random copies of $(X, Y) \in \mathcal{X} \times \mathcal{Y}$. The feature X lives in some abstract space $\mathcal{X}$ ($\mathbf{R}^d$ for instance) and Y is called label. Binary classification assumes that Y take two different values: $\mathcal{Y} = \{-1, 1\}$, while multi-class classification assumes $\mathcal{Y} = \{1, \dots, K\}$. To keep things simple, we will only look at binary classification. Naturally, one would like to find a function $f : \mathcal{X} \mapsto \mathbf{R}$ that best maps X to Y. We are also given a loss function $\ell : \{-1, 1\} \times \{-1, 1\} \mapsto \mathbf{R}$ that measures the error of a specific prediction. The loss function value at an arbitrary point $(Y, \hat{Y})$ reads as the cost incurred when predicting $\hat{Y}$ while true label is Y. In classification the loss function is often a zero-one loss, that is, $\ell(Y, \hat{Y})$ is zero when the predicted label matches the true label $Y = \hat{Y}$ and one otherwise. To find our best classifier, we look for the classifier with the smallest expected loss. In other words, we look up for the function f that minimizes the expected ℓ-risk, given by $\mathcal{R}_\ell(h) = \mathbf{E}_{X \times Y}[\ell(Y, f(X))]$.

Another naive approach is to minimize the empirical classification error $\mathbb{E}[\mathbb{1}_{\{-Yf(X) \geq 0\}}]$. To bypass the non convexity of $\mathbb{1}_{\mathbb{R}_+}$, we use convex risk minimization (CRM) [BBL05]. CRM defines a convex surrogate for the classification problem, called the cost function $\varphi : \mathbb{R} \mapsto \mathbb{R}_+$ convex, non-decreasing such that $\varphi(0) = 1$, hence $\varphi \geq \mathbb{1}_{\mathbb{R}_+}$. The classification problem consists in minimizing the expected φ-risk : $\mathbb{E}[\varphi(-Yf(X))]$. Typical loss functions are

- square loss: $\varphi(u) = (1+u)^2$ for $u \geq 0$ and $\varphi(u) = 1$ for $u \leq 0$, leading to regression methods.

- perplexity loss: $\varphi(u) = \log(e(1+u)))$ for $u > -1$ and $\varphi(u) = -\infty$ for $u \leq -1$.
- logit loss: $\varphi(u) = \log_2(1+e^u)$, leading to logistic regression methods and cross entropy.
- hinge loss: $\varphi(u) = max(0, 1+u)$. This leads to SVM methods.
- exponential loss: $\varphi(u) = e^u$.

We have also loss function defined directly between Y and $\hat{Y}$ as follows:

- mean square error: $\ell(Y, f(X)) = (Y - f(X))^2$ that leads to standard regression methods.
- mean absolute error: $\ell(Y, f(X)) = |Y - f(X)|$.
- cross entropy: $\ell(Y, f(X)) = -\frac{1+Y}{2} \log \frac{1+f(X)}{2}$ leading to logistic regression with the usual convention $0 \log 0 = 0$ justified by $\lim\limits_{x \to 0^+} x \log x = 0$ (trivially obtained by L'Hopital's rule).
- Huber loss: $\ell(Y, f(X)) = \frac{1}{2}(Y - f(X))^2$ if $|Y - f(X)| \leq \delta$ and $\ell(Y, f(X)) = \delta(|Y - f(X)| - \frac{1}{2}\delta^2$

We see from above that for SL there are numerous loss functions and that cross entropy is one criterion among others. We will see that this cross entropy has a nice interpretation in RL.

4 Reinforcement Learning Background

RL is usually modeled by an agent that interacts with an environment $\mathcal{E}$ over a number of discrete time steps. At each time step t, the agent levies a state s_t and picks an action a_t from a set of possible actions $\mathcal{A}$. This choice is done according to its policy π, where π is a mapping from states s_t to actions a_t. Once the action is decided and executed, the agent levies the next state s_{t+1} and a scalar reward r_t. This goes on until the agent reaches a terminal state. The expected cumulated discounted return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the sum of accumulated returns, where at each time step, future returns are discounted with the discount factor $\gamma \in (0, 1]$. At time t, a rational agent seeks to maximize its expected return given his current state s_t.

We traditionally define

- the value of state s under policy π is defined as $V^\pi(s) = \mathbb{E}\left[R_t | s_t = s\right]$ and is simply the expected return for following policy π from state s ([Wat89]).
- the action value function under policy π $Q^\pi(s, a) = \mathbb{E}\left[R_t | s_t = s, a\right]$ is defined as the expected return for selecting action a in state s and following policy π ([Wil92]).

Both the optimal value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ and the optimal value of state $V^*(s) = \max_\pi V^\pi(s)$ satisfy Bellmann equations.

Whenever states and action are too large, we are forced to represent the action value function with a function approximator, such as a neural network. Denoting the parameters θ, the state action function writes $Q(s, a; \theta)$

The updates to θ can be derived from a variety of reinforcement learning algorithms. In particular, in value-based methods, policy-based model-free methods directly parameterize the policy $\pi(a|s; \theta)$ and update the parameters θ by performing, typically approximate, gradient ascent on $\mathbb{E}[R_t]$.

An illustration of such a method is REINFORCE due to [Wil92]. Standard REINFORCE updates the policy parameters θ in the direction $R_t \nabla_\theta \log \pi(a_t|s_t; \theta)$, which is an unbiased estimate of $\nabla_\theta \mathbb{E}[R_t]$. It is possible to reduce the variance of this estimate while keeping it unbiased by subtracting a learned function of the state $b_t(s_t)$, known as a baseline [Wil92], from the return. The resulting gradient is $\nabla_\theta \log \pi(a_t|s_t; \theta) \, (R_t - b_t(s_t))$. This approach can be viewed as an actor-critic architecture where the policy π is the actor and the baseline b_t is the critic[SB98, DPS12].

We now prove a new formulation of REINFORCE.

Proposition 4.1. *The gradient descent in REINFORCE can also be computed by minimizing the following quantity*

$$\widetilde{J}(\theta) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} R_i(\tau) \log \pi_\theta(a_{i,t} \mid s_{i,t}) \qquad (1)$$

For Advantage Actor Critic method, the gradient descent can also be computed by minimizing the following quantity:

$$\widetilde{J}(\theta) = \frac{\sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T} A(s_{i,t}, a_{i,t}) \log \pi_\theta(a_{i,t} \mid s_{i,t})}{N} \qquad (2)$$

Proof. See supplementary materials 7.1 □

It is enlightening to see that the two formulations, traditional reinforce and actor critic are very close to SL method with cross entropy. Recall that cross entropy is given by $Y \log(\hat{Y})$ for labels with value in $\{0, 1\}$. This leads to the tables 1 and 2.

Term	SL	RL (REINFORCE)
true label	Y	expected future rewards: $R(\tau)$
log term	$\log(\hat{Y})$	log of policy: $\log \pi_\theta(a_{i,t} \mid s_{i,t})$
cross entropy	$\frac{\sum_{i=1}^{N} Y_i \log \hat{Y}_i}{N}$	Monte Carlo expectation: $\frac{\sum_{i=1}^{N} \sum_{t=1}^{T} R_i(\tau) \log \pi_\theta(a_{i,t} \mid s_{i,t})}{N}$

Table 1: Comparing SL and RL for REINFORCE

Term	SL	RL (A2C)
true label	Y	expected advantage: $A(s,a) = Q(s,a) - V(s)$
log term	$\log(\hat{Y})$	log of policy: $\log \pi_\theta(a_{i,t} \mid s_{i,t})$
cross entropy	$\frac{\sum_{i=1}^{N} Y_i \log \hat{Y}_i}{N}$	Monte Carlo expectation: $\frac{\sum_{i=1}^{N} \sum_{t=1}^{T} A(s_{i,t}, a_{i,t}) \log \pi_\theta(a_{i,t} \mid s_{i,t})}{N}$

Table 2: Comparing SL and RL for AAC methods

Last but not least, recall that there is a connection between cross entropy and Kullback Leibler divergence. Recall that the cross entropy for the distributions p and q over a given set is defined as follows:

$$H(p, q) = \mathbb{E}_p[-\log q] \tag{3}$$

There is a straightforward connection to the Kullback–Leibler divergence $D_{\mathrm{KL}}(p\|q)$ of q from p, sometimes referred to as the relative entropy of p with respect to q given by

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p\|q) \tag{4}$$

where $H(p)$ is the entropy of p. As the entropy term $H(p)$ is constant given the true distribution p, minimizing the cross entropy or the Kullback Leibler divergence is equivalent. However, for RL, this gives another nice interpretation. As shown previously, PGM can be cast as a cross entropy minimization program. Since the difference between Kullback Leibler divergence and cross entropy is this entropy term, it makes sense to incorporate this entropy term in our gradient descent optimization. In theory, the entropy term should be multiply by one . However in practice as the entropy term is estimated by the empirical entropy, it makes sense somehow to multiply the entropy term by a regularization term λ, leading to a variation of PGMs where instead of computing a gradient on cross entropy as in REINFORCE, we add an additional term called an entropy regularization. Somehow, this changes the minimization problem to something that is a modified Kullback Leibler divergence. The cross entropy term itself is computed on future expected reward. The analogy holds between SL and RL as long as actions do not influence or slightly influence the environment.

Remark 4.1. *As a matter of fact, our tight connection between RL and SL makes a lot of sense and is very intuitive. Indeed, SL can be cast as an optimization problem, which can be solved by gradient descent (that's with stochastic gradient descent (SGD) and backprop in neural networks). Value estimation and value optimization in RL can also be cast as an optimization problem. It can also be solved by gradient descent (with TD-learning or Q-learning). Finally, policy optimization can also be solved by gradient descent (that's policy gradient). The only difference is really the loss function that is optimized.*

5 Experiments

We will apply our remark to a very specific environment where actions do not influence the environment. The considered reinforcement problem is a financial trading game concerning the Facebook stock (data were retrieved from https://finance.yahoo.com/quote/FB. We denote by $(P_t)_{t=1,...}$ the daily closing price of the Facebook stock in sequential order. For each day, we compute the daily return as follows $r_t = \frac{P_t}{P_{t-1}} - 1$

The environment is composed of the n last daily returns. We intentionally assume n last returns to emphasize that the choice of taking $n = 5$ (last week) or $n = 10$ (last two weeks) or $n = 20$ (last month) is a model design decision. As returns are continuous, our state space is $\mathbb{R}_+^n$, which by RL standard is very large. Our possible actions are each day threefold: either do nothing, buy or sell the Facebook stock. If we decide to enter in a new position at time t, this will only be materialized the next day and hence we will be initially having an open position only at time $t + 1$ initialized at the entering price p_{t+1}. Hence, if we only keep the position for one period, we will be facing the return r_{t+2} as our position will be only closed at time $t + 2$.

As for the reward, we take the Sharpe ratio of the trading strategy. As we compute the Sharpe ratio with daily returns, to compute the annual Sharpe ratio, we multiply the daily Sharpe ratio by a scaling factor equal to $\sqrt{250}$, assuming 250 trading days per year. We compute the daily Sharpe ratio as the mean of daily returns over their standard deviations. This implies in particular that we implicitly take a benchmark rate in the Sharpe ratio equal to 0.

To compute our Mark to Market (the value of our trading strategy), we mark any open position to the last know price. We parametrize our policy with a deep network consisting of two fully connected layers composed of 16 ReLU nodes. We use ReLU activation function (as opposed to sigmoid) for two main reasons: sparsity and less chance to incur vanishing gradient. Recall a ReLU is defined as $h = max(0, a)$ where $a = Wx + b$. Vanishing gradient has less chance to vanish since the gradient is either zero or constant whenever the linear term is strictly positive ($a > 0$). In contrast, the gradient of sigmoid becomes increasingly small when the absolute value of x tends to be large. This constant character of the ReLU's gradient results in faster learning. The other benefit of ReLUs is sparsity. Sparsity arises when the linear term is non positive: $a \leq 0$. The more units with non positive term exist in a layer, the more sparse the resulting representation. In contrast, sigmoids always generates some non-zero value resulting in dense representations.

In our experience, we take two fully connected layers composed of 16 ReLU nodes and use A2C method to solve this reinforcement learning problem. For exploration purposes, we use three epsilon greedy method with epsilon equal to 5%, 25% and 50%. We compare these constant learning rates to two annealing method that decreases linearly the learning rate from 1 to 0.1% and from 0.5 to 0.1%. We provide the overall achieved Sharpe ratio with the Sharpe ratio reward choice for various learning rate and annealing rate. We obtained an overall Sharpe ratio higher than 1 which is quite a nice achievement compared to traditional investment strategies that typically do not perform better than a Sharpe ratio of 1. We see in the resulting experience that the exploration methods that works best are the two epsilon greedy annealed methods.

We can notice in figure 1 that the overall Sharpe ratio is above 3 which is very high by financial market standards. This could be explained by the fact that the Facebook stock has been incredibly raising over the last five years. Hence the algorithm has not much difficulty finding the optimal strategy that is to buy and hold the stock

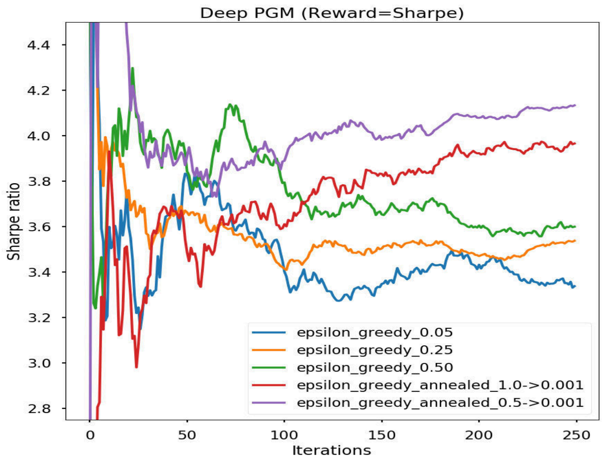

Figure 1: Comparison of various learning rate strategy for our experience. The best strategies are annealing learning rates. These strategies work well as they have the right balance between exploration exploitation. At first, as their learning rate is quite high, they explore more than their fix learning rate counterpart. As we progress in the algorithm, the learning rate decreases and this strategy progressively shift from exploration to exploitation.

6 Conclusion

We show in this article that there are tight connections between SL and RL. PGM in RL can be cast as cross entropy minimization problems where true labels are replaced by expected future reward or advantage while the log term is changed into the log policy term. This analogy takes its root from the minimization problem where we are looking for the parameters that maximizes the expected futures reward or advantage. Should this optimization objective changed, we conjecture that we could make other analogies between SL and RL

References

[BBL05] S. Boucheron, O. Bousquet, and G. Lugosi. Theory of classification: A survey of some recent advances. *ESAIM: Probability and Statistics*, 9:323, 2005.

[DPS12] Thomas Degris, Patrick M. Pilarski, and Richard S. Sutton. Model-free reinforcement learning with continuous action in practice. *IEEE In ACC*, 2012:2177–2182, 2012.

[HVP$^+$17] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017.

[KT03] Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM J. Control Optim.*, 42(4):1143–1166, April 2003.

[LHP$^+$15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ArXiv e-prints*, 2015.

[MBM$^+$16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, volume 48, pages 1928–1937, New York, New York, USA, 20-22 Jun 2016. PMLR.

[NNXS17] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *NIPS*, pages 2775–2785. Curran Associates, Inc., 2017.

[OMKM16] Brendan O'Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. PGQ: combining policy gradient and q-learning. *CoRR*, abs/1611.01626, 2016.

[PS08] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, March 2008.

[RGB11] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *In: AISTATS, JMLR.org, JMLR Proceedings*, 15:627–635, 2011.

[SAC17] John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. *CoRR*, abs/1704.06440, 2017.

[SB98] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[Sch96] S. Schaal. Learning from demonstration. In *NIPS, MIT Press*, pages 1040–1046. NIPS, MIT Press, 1996.

[SLH$^+$14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, number 1 in Proceedings of Machine Learning Research, pages 387–395, Bejing, China, 22-24 Jun 2014. PMLR.

[SMSM99] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[SVG$^+$17] Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply AggreVaTeD: Differentiable imitation learning for sequential prediction. In *ICML*, volume 70, pages 3309–3318. PMLR, 06-11 Aug 2017.

[Wat89] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge England, 1989.

[WD92] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[Wil92] R. J. Williams. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, volume 8. Springer, 1992.

7 Supplementary materials

7.1 Proof of proposition 4.1

We provide here a quick proof of REINFORCE with modern notations. Let us denote by $r(s_t, a_t)$ the reward for a state s_t and action a_t. Let us assume we have some time horizon (that may be either finite or infinite). Let us denote by $\tau = (s_1, a_1, \dots, s_T, a_T)$ a trajectory generated by our policy approximator governed by a parameter θ. Using the Markov property of our MDP process, the probability of a given trajectory $\mathbb{P}(\tau|\theta)$ can be decomposed into a product of conditional probabilities as follows:

$$\mathbb{P}(\tau|\theta) = \mathbb{P}(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) \mathbb{P}(s_{t+1} \mid s_t, a_t) \tag{5}$$

Taking the log of the above equation (5), using the fact that the log of a product is the sum of the logs, we get:

$$\begin{aligned} \log \mathbb{P}(\tau|\theta) &= \log \mathbb{P}(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) \\ &\quad + \log \mathbb{P}(s_{t+1} \mid s_t, a_t) \end{aligned} \tag{6}$$

Differentiating with respect to theta gives (since most of terms do not depend on θ):

$$\nabla_\theta \log \mathbb{P}(\tau|\theta) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \tag{7}$$

Recall we want to minimize the expected return, $J(\theta)$, defined as

$$J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\tau|\theta)} \left[\sum_{t=1}^{T} r(s_t, a_t) \right] = \int_\tau R(\tau) \mathbb{P}(\tau|\theta) d\tau \tag{8}$$

The above notation $\tau \sim \mathbb{P}(\tau|\theta)$ indicates that we're sampling trajectories τ from the probability distribution of our policy approximator governed by θ and $R(\tau)$ is the sum of all the future (discounted) rewards.

To find the optimal θ, we do gradient descent and hence need to compute

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau R(\tau) \mathbb{P}(\tau|\theta) d\tau$$

Using the fact that we can interchange integral and expectation (9), assuming smooth functions and Lebesgue dominated convergence to justify that we can bring the gradient under the integral, we get:

$$\nabla_\theta J(\theta) = \int_\tau R(\tau) \nabla_\theta \mathbb{P}(\tau|\theta) d\tau \tag{9}$$

As the log gradient of a function is the quotient of the gradient and the function, we have:

$$\nabla_\theta J(\theta) = \int_\tau R(\tau) \nabla_\theta \log \mathbb{P}(\tau|\theta) \mathbb{P}(\tau|\theta) d\tau \tag{10}$$

Expressed the integral as an expectation, we conclude:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[R(\tau) \nabla_\theta \log \mathbb{P}(\tau|\theta) \right] \tag{11}$$

Finally, using the fact that the gradient of the log probability of the trajectory is the sum of the gradient of the log policy probabilities (7), we obtain the final expression:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[R(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right] \tag{12}$$

To turn this into something tractable, just express this as a Monte Carlo sum as follows:

$$\nabla_\theta J(\theta) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} R_i(\tau) \nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t}) \tag{13}$$

As the RHS does only depend on θ in the term $\log \pi_\theta(a_{i,t} \mid s_{i,t})$, minimizing $J(\theta)$ is the same as minimizing $\widetilde{J}(\theta)$ given by:

$$\widetilde{J}(\theta) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} R_i(\tau) \log \pi_\theta(a_{i,t} \mid s_{i,t}) \tag{14}$$

As for Advantage Actor Critic, the above proof is the same and leads to the result. $\square$