Bias-Variance Trade-off and Overlearning in Dynamic Decision Problems

A. Max Reppen* H. Mete Soner†

November 19, 2020

Abstract

Modern Monte Carlo-type approaches to dynamic decision problems face the classical bias-variance trade-off. Deep neural networks can overlearn the data and construct feedback actions which are non-adapted to the information flow and hence, become susceptible to generalization error. We prove asymptotic overlearning for fixed training sets, but also provide a non-asymptotic upper bound on overperformance based on the Rademacher complexity demonstrating the convergence of these algorithms for sufficiently large training sets. Numerically studied stylized examples illustrate these possibilities, the dependence on the dimension and the effectiveness of this approach.

Key words: Deep Learning, Monte Carlo, Overlearning, Markov Decision Problems, Dynamic optimization, Bias-Variance Trade-off, Convergence.

Mathematics Subject Classification: 91G60, 49N35, 65C05.

1 Introduction

Recent advances in training of neural networks make high-dimensional numerical studies feasible for decision problems in uncertain environments. Although reinforcement learning has been widely used in optimal control for several decades [6], only recently Han and E [18], Han et al. [20] combine it with Monte Carlo type regression for the off-line construction of optimal feedback actions. In these problems, the randomness and the state are observable and a training set based on historical or simulated data is readily available. One then approximates the objective functions of these problems by the empirical averages over this training data, constructing a loss function which is minimized over the network parameters. The minimizer or a near-minimizer is the trained network and it is an approximation of the optimal feedback action. This approach,

*Questrom School of Business, Boston University, Boston, MA, 02215, USA, email: amreppen@bu.edu.

†Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ, 08540, USA, email: soner@princeton.edu

which we call dynamic *deep Monte Carlo optimization*, has proved to be highly effective in many closely related studies including Bachouch et al. [1], Becker et al. [4], Buehler et al. [10, 11], Henry-Labordère [22], Huré et al. [24] reporting impressive numerical results in problems with large number of states.

The generalization of the trained network or its out-of-sample performance is the key property determining its effectiveness. We study this central question by defining the pathwise cost of an action as a *deterministic function of the randomness*. Then, the performance of the action is the expected value of this deterministic function and the loss function described above is simply its empirical average. This novel reformulation of the optimal decision problem is our main observation allowing us to directly apply results from statistical machine learning without detailed analysis, providing useful estimates and tools for the analysis of dynamic deep Monte Carlo optimization. To articulate this general roadmap with clarity, we avoid technical constructs and only emphasize the fundamental structures and the connections between them.

It is well-known that the optimal feedback controls of dynamic optimization problems are determined by the conditional expectation of the value function evaluated at the future controlled random state and the above approach essentially uses a regression estimate of this conditional expectation [15]. It is therefore natural that these successful studies implicitly face the classical bias-variance trade-off described in the seminal paper of Geman et al. [16]. However, the dynamic nature of the decisions is an essential feature that separates stochastic optimal control from the classical regression or interpolation. The optimal or near-optimal decisions depend on a time-varying estimate of the randomness driving the dynamics of the state. In general there is no causality and in some applications the available data for training is limited in size. Hence, as opposed to interesting recent studies arguing the benefits of more complex networks and interpolation (cf. Belkin et al. [5] and the references therein), in dynamic settings overfitting causes the loss of the most salient restriction of the problem, namely, the adaptedness of the decisions to the information flow.

Indeed, the global minimum of the loss functions are achieved by feedback actions depending on the whole random path including the future rather than an estimate of the conditional expectation which is a function of only the past information. Therefore, sufficiently large networks may at every time *overlearn* the future training data instead of estimating it, causing them to generate output actions that look into the future. This renders the trained feedback actions *on the training set* to be *non-adapted* to the filtration generated by the observable variables. Thus, in-sample they *overperform* the original control problem, as they implicitly circumvent the essential restriction of the adaptedness of the decisions. Consequently, feedback actions constructed by sufficiently wide networks do not always generalize and may perform poorly out-of-sample. Examples 5.2, 5.3 below, illustrate the concept of overlearning and the consequent non-adaptedness of the actions clearly in non-technical settings.

More importantly, in the other direction, Sections 6 and 7 provide an analysis based on the Rademacher complexity ([3, 9]) and Theorem 7.1 establishes several non-asymptotic error estimates. These results state that with high probability the performance error of a randomly chosen training set is bounded by the *empirical* Rademacher complexity. As the complexity of a fixed network gets smaller with larger training data, for such an appropriate combination of network structure and data, overlearning would be small as well. On the other hand, for fixed training sets the complexity increases with the size of the networks. Thus, the performance error estimate (7.2) is an analytic manifestation of the bias-variance or more precisely, bias-complexity trade-off (cf. Shalev-Shwartz and Ben-David [34], Chp. 5) in this context.

The large sample size-limit of the complexity estimate proves the convergence of the deep Monte Carlo optimization. Indeed, Corollary 8.2 shows that for sufficiently large training sets the actions constructed by appropriately wide or deep neural networks are close to the desired solutions and the overlearning is negligible. Huré et al. [24] also establishes convergence results for controlled Markov processes including the above and several other hybrid algorithms that they propose. Han and Long [19] provides another convergence analysis for the backward stochastic differential equations which are closely connected to optimal control.

Our numerical experiments support these theoretical observations. In Section 9 we analyze a stylized Merton utility maximization problem of Example 4.1. Like the previous papers, our results also clearly demonstrate the effectiveness of the dynamic deep Monte Carlo optimization in handling high dimensional problems and the ease of including realistic aspects to the problems as done by Buehler et al. [10]. Additionally, our experiments illustrate the convergence of the algorithm and the dependence of overlearning on the dimension of the randomness driving the dynamics.

The potential overlearning is shown by comparing the in-sample and out-of-sample performances of the algorithm. Although we employ conservative stopping rules in the stochastic gradient algorithm used in the training, there is always some amount of overlearning. Our experiments with a training size of $100,000$ and three hidden layers of width 10 show in-sample to out-sample-sample performance differences ranging from 1.5% in 10 dimensions to 24% in 100 dimensions. Moreover, more aggressive minimization results in substantial overlearning. In 100 dimensions, we could achieve up to 30% over-performance over the known true solution in 100–200 epochs and more would be possible with longer tries. In these cases, the out-of-sample performance deteriorates rapidly.

It is well documented in the literature that the size of the training data is central to the performance of this approach. In 100 dimensions, we achieve a remarkable improvement in the accuracy of our numerical computations by increasing the size of the training data. Our estimates also indicate that more data points is an effective way to ensure the adaptedness of the feedback actions. So in applications with limited available data, one needs to enrich the training set by simulations as done in a financial application by Kondratyev and Schwarz [28] using Boltzmann machines [33] which is also discussed in Buehler et al. [11].

In lower dimensional examples with limited data, robust optimization as proposed by Bertsimas et al. [7, 8] also provides an effective approach. In similar studies, Esfahani and Kuhn [14] and Bartl et al. [2] use Wasserstein balls to regularize the problem. Although very effective in many problems, as the space dimension gets larger, this method becomes computationally more difficult. Still, we believe that one could incorporate these techniques to deep Monte Carlo optimization to further reduce the overlearning.

The paper is organized as follows. The decision problem is formulated in Section 2. The Monte Carlo optimization is outlined in Section 3. Two motivating examples are given in Section 4 and overlearning is defined and proved in Section 5. The classical Rademacher complexity is recalled in Section 6 and the error estimate is proved in Section 7. The convergence is discussed in Section 8. Section 9 outlines the specifics of the network structure, the optimization algorithm and the experiments. After the concluding remarks, in Appendix A we provide a generalization of the overlearning theorem.

2 Decision Problem

Consider a dynamic decision problem under uncertainty or, equivalently, a stochastic optimal control problem in discrete time with a finite horizon of T. Set

$$\mathcal{T} := \{0, 1, \ldots, T-1\}.$$

The state space $\mathcal{X}$, disturbance set $\mathcal{Z}$ and the control set $\mathcal{A}$ are closed subsets of Euclidean spaces. A finite sequence $z := (z_1, z_2, \ldots, z_T) \in \mathcal{Z}^T$ is a deterministic *trajectory* representing the realizations of the random input.

The source of randomness is an exogenous random trajectory $Z_1, Z_2, \ldots, Z_T$ with values in $\mathcal{Z}$ defined on a probability space $(\Omega, \mathbb{P})$. We set $Z_0 = 0$ and use the notation $Z = (Z_1, Z_2, \ldots, Z_T)$. For $t > 0$, $\mathcal{F}_t$ is the *smallest σ-algebra* (i.e., a collection of subsets of Ω that is closed under countably many usual set operations) so that the random variables $Z_1, \ldots, Z_t$ are all $\mathcal{F}_t$ *measurable* and $\mathcal{F}_0 = \{\emptyset, \Omega\}$. The collection of increasing sequence of σ-algebras $\mathbb{F} = (\mathcal{F}_t)_{t \in \mathcal{T}}$ is the *filtration*.

In our notation, whenever possible, we use capital letters for random variables, lower case letters for deterministic quantities and sets are denoted by calligraphic letters. We assume all functions to be continuous.

2.1 Dynamics

We only consider controls or *actions* that are of feedback form, i.e., an action

$$a : \mathcal{T} \times \mathcal{X} \times \mathcal{Z} \to \mathcal{A}$$

is a continuous function[1] of time and the observed state and randomness. Let $\mathcal{C}$ be the set of all actions satisfying the constraints $a(t, \cdot) \in \mathcal{A}_t$ for every $t \in \mathcal{T}$,

[1]When the process Z is Markov, one does not gain from enlarging the controls to all adapted processes [15]. When Z is a general process, restricting the actions to be feedback type defines a well-defined problem whose optimal value might be different than the one obtained in the larger class of adapted controls.

where $\mathcal{A}_t \subset \mathcal{A}$ are given Borel sets.[2]

For an action $a \in \mathcal{C}$, an initial value $x \in \mathcal{X}$ and a (deterministic) trajectory $z = (z_1, z_2, \ldots, z_T)$, the controlled state process $x^a = (x^a_0, x^a_1, \ldots, x^a_T)$ is defined recursively by

$$x^a_{t+1} = f(t, x^a_t, z_{t+1}, a(t, x^a_t, z_t)), \quad t \in \mathcal{T},$$

where $x^a_0 = x$, $z_0 = 0$ and $f : \mathcal{T} \times \mathcal{X} \times \mathcal{Z} \times \mathcal{A} \to \mathcal{X}$ is a given smooth deterministic function. We fix the initial value x and let $x^a(z)$ be the state as a function of the trajectory z. For a given random trajectory $Z = (Z_1, \ldots, Z_T)$, we set

$$X^a := x^a(Z) \quad \text{with} \quad X^a_t = x^a_t(Z).$$

As $x^a_t(z)$ depends only on $(z_1, z_2, \ldots, z_t)$, X^a is adapted to the filtration $\mathbb{F} = (\mathcal{F}_t)_{t \in \mathcal{T}}$.

2.2 Performance Criteria

The performance criteria of a control process $a \in \mathcal{C}$ is given by,

$$v(a) := \mathbb{E}_\nu \left[\sum_{t \in \mathcal{T}} \psi(t, X^a_t, a(t, X^a_t, Z_t)) + \varphi(X^a_T) \right], \tag{2.1}$$

where ν is the distribution of the random process Z and $\psi : \mathcal{T} \times \mathcal{X} \times \mathcal{A} \to \mathcal{R}$, $\varphi : \mathcal{R}^d \to \mathcal{R}$ are given smooth functions.[3] Since $X^a = x^a(Z)$ is a deterministic function of the random trajectory Z, we may rewrite the above definition as follows,

$$v(a) := \mathbb{E}_\nu \left[\ell(a, Z) \right],$$

where $\ell : \mathcal{C} \times \mathcal{Z}^T \to \mathcal{R}$ is given by,

$$\ell(a, z) := \sum_{t \in \mathcal{T}} \psi(t, x^a_t(z), a(t, x^a_t(z)) + \varphi(x^a_T(z)). \tag{2.2}$$

The *dynamic stochastic decision problem* is to minimize $v(a)$ over all admissible controls $a \in \mathcal{C}$. The *optimal value* is given by,

$$v^* := \inf_{a \in \mathcal{C}} v(a). \tag{2.3}$$

We make the following simplifying assumption on the coefficients.

Assumption 2.1. *We assume that the functions ψ, φ in* (2.1) *are uniformly bounded and continuous so that ℓ defined in* (2.2) *is continuous in the control variable uniformly in the z-variable and there exist a constant c^* satisfying*

$$|\ell(a, z)| \le c^*, \qquad \forall\ z \in \mathcal{Z}^T,\ a \in \mathcal{C}.$$

In particular, if a sequence of feedback actions a_n converge to a pointwise, then $\lim_{n \to \infty} v(a_n) = v(a)$.

[2]More general structures, including state constraints can easily be incorporated.

[3]One may easily generalize the problem by allowing the cost functions ψ, φ to explicitly depend on the randomness. Although potentially important in some applications, we refrain from this easy generalization as it would unnecessarily further complicate the notation.

3 Deep Monte Carlo Optimization

In this section, we outline the deep Monte Carlo optimization of Han and E [18].

The *training set* is a collection of n observations of trajectories:

$$\mathcal{L}_n = \left\{ Z^{(1)}, Z^{(2)}, \ldots, Z^{(n)} \right\} \quad \text{where} \quad Z^{(i)} = (Z_1^{(i)}, Z_2^{(i)}, \ldots, Z_T^{(i)}).$$

For any $a \in \mathcal{C}$, this data generates n realizations of the state process $x^a(Z^{(i)})$ as well. We then use the training set to define the *loss function* for $a \in \mathcal{C}$ by,

$$L(a; \mathcal{L}_n) := \frac{1}{n} \sum_{i=1}^{n} \ell(a, Z^{(i)}). \tag{3.1}$$

In the applications that motivate this study, the exogenous process Z is observable and is the only source of randomness. In the example of portfolio management (Section 4.2), it is the stock price process and in the production planning problem (Section 4.1), it is the demand for a certain product. In both cases, historical data is available. In this study, we do not discuss how the training set is generated. Rather we take it as given and study its interaction with the neural networks.

We abstract the neural networks as a parametrized collection of functions. A parameter θ is a finite sequence of real numbers. For each parameter θ, a neural network is a continuous function

$$\Phi(\cdot; \theta) : \mathcal{T} \times \mathcal{X} \times \mathcal{Z} \to \mathcal{A}.$$

We consider a sequence of *compact* parameter sets $\mathcal{O}_k \subset \mathcal{R}^{d(k)}$ with increasing dimensions $d(k)$. We assume that Φ is *continuous.* Set

$$\mathcal{N}_k := \left\{ \Phi(\cdot; \theta) \ : \ \theta \in \mathcal{O}_k \right\}. \tag{3.2}$$

We fix the training set $\mathcal{L}_n$ and a set of neural networks $\mathcal{N}_k$, and

$$\text{minimize} \quad \theta \in \mathcal{O}_k \ \mapsto \ L(\Phi(\cdot; \theta); \mathcal{L}_n). \tag{3.3}$$

As L is continuous and $\mathcal{O}_k$ is compact, there exists a minimizer $\theta^*_{k,n} \in \mathcal{O}^*_k$. Then,

$$A^*_{k,n}(t, x, z) := \Phi(t, x, z; \theta^*_{k,n}), \qquad t \in \mathcal{T}, x \in \mathcal{X}, z \in \mathcal{Z},$$

is the *optimal feedback action* that could be constructed by the neural network $\mathcal{N}_k$ using $\mathcal{L}_n$. As the training data $\mathcal{L}_n$ is random, $A^*_{k,n}$ and its *in-sample performance*

$$L(A^*_{k,n}; \mathcal{L}_n) = \inf_{\theta \in \mathcal{O}_k} L(\Phi(\cdot; \theta); \mathcal{L}_n) \tag{3.4}$$

are both random as well. We are also interested in the *out-of-sample performance* $L(A^*_{k,n}; \hat{\mathcal{L}}_n)$ on an independently chosen set $\hat{\mathcal{L}}_n$ and its *average performance*,

$$v(A^*_{k,n}) = \mathbb{E}[\ell(A^*_{k,n}, Z)].$$

The effectiveness of this algorithm depends on the size and the architecture of the neural networks $\mathcal{N}_k$, the size of the training set $\mathcal{L}_n$, and also on their interactions. The main goal of this paper is to study this and the connections between $v^*, L(A^*_{k,n}; \mathcal{L}_n), v(A^*_{k,n})$ and $L(A^*_{k,n}; \hat{\mathcal{L}}_n)$.

Since numerically one can only construct an approximation of the above minimizer, the details of the approximating optimization procedure is an essential part of the algorithm. In our experiments, we use a standard stochastic gradient descent with commonly used stopping rules to better study the properties of the deep Monte Carlo optimization.

The only requirement we impose on $\mathcal{N}_k$ is to have the approximation capability. It is well known that the neural networks have this property as proved by Cybenko [12] and Hornik [23].

Assumption 3.1. *We assume that for any bounded continuous function $a : \mathcal{T} \times \mathcal{X} \times \mathcal{Z} \to \mathcal{A}$, there exists a sequence $\theta_k \in \mathcal{O}_k$ such that $\Phi(\cdot; \theta_k)$ converges to a locally uniformly.*

This assumption easily implies that the neural networks can approximate the optimal value. The more interesting question of the convergence of the computable minimum values $v(A^*_{k,n})$ is studied in Section 8.

Lemma 3.2. *Suppose that the Assumptions 2.1, 3.1 holds. Then,*

$$\lim_{k\to\infty} \inf_{\theta \in \mathcal{O}_k} v(\Phi(\cdot; \theta)) = v^*.$$

Proof. Set $v^*_k := \inf_{\theta\in\mathcal{O}_k} v(\Phi(\cdot;\theta))$. Let $a^*_\epsilon \in \mathcal{C}$ be an ϵ-minimizer of v: $v(a^*_\epsilon) \le v^* + \epsilon$. In view of Assumption 3.1, there exists a sequence $\theta_k \in \mathcal{O}_k$ such that $\Phi(\cdot;\theta_k)$ converges to a^*_ϵ locally uniformly. Then, by Assumption 2.1, $\limsup_{k\to\infty} v^*_k \le \lim_{k\to\infty} v(\Phi(\cdot;\theta_k)) = v(a^*_\epsilon) \le v^* + \epsilon$. Since $\epsilon > 0$ is arbitrary and $v^* \le v^*_k$, we conclude that v^*_k converges to v^*. □

4 Examples

We briefly outline two classes of problems to clarify the model and the notation. Several other examples are also discussed in [18].

4.1 Production Planning

The above structure includes the multi-stage optimization problems introduced by Bertsimas et al. [7, 8]. For clarity, here we describe only a simple example of these problems which is very similar to Example 1 in Bertsimas et al. [7] and refer the reader to Bertsimas et al. [7, 8] for the more general problems.

Consider producers facing an optimal production decision. They observe the random demand Z_1, Z_2 in two stages. The production level is decided after observing Z_1 but before Z_2. The second component of the random demand Z_2 is observed at the final stage. The goal is to bring the final inventory level close to zero by properly choosing the production level at stage one. Let X^a be the inventory level. We assume the initial inventory is zero and no production is made initially. Then, $X_1^a = Z_1$ and with $a(\cdot) := a(1,\cdot)$, $X_2^a = X_1^a - a(Z_1) + Z_2$, and the problem is to minimize

$$v(a) = \mathbb{E}\left[\phi(ZX_2^a)\right] = \mathbb{E}\left[\phi(Z_1 + Z_2 - a(Z_1))\right]$$

over all production production functions $a \in \mathcal{C}$. The penalty function $\phi \geq 0$ is convex and equal to zero only at the origin. In our framework, $\mathcal{A}_0 = \{0\}$, $\mathcal{A}_1 = [0, \infty)$, $\psi \equiv 0$ and $f(t, x, z, a) = y - a + z$.

For $\phi(x) = x^2$ this is exactly the classical regression problem of estimating the total demand $Z_1 + Z_2$ after observing the first component Z_1. It is well-known that the optimal solution is $a^*(Z_1) = \mathbb{E}[Z_1 + Z_2 \mid Z_1]$, and this optimization problem reduces to the classical regression well-known to face the bias-variance trade-off. Although this connection may not be as explicit in other more complex problems, it is always inherent to the problem.

4.2 Merton Problem

Starting with Hutchinson et al. [25], neural networks have been employed in quantitative finance. Recently, Bachouch et al. [1], Becker et al. [4], Buehler et al. [10, 11], Henry-Labordère [22], Huré et al. [24] use them to obtain impressive results in high dimensional problems. Here we only outline a portfolio management problem in a financial market with d many assets. Although this example does not include many important modeling details, it must be clear that by choosing Z and X^a appropriately, one can cover essentially all classical Merton type utility maximization, portfolio optimization and hedging problems studied in the literature. Also problems with different structures such as free boundary problems studied by Becker et al. [4] and the hedging problem with transactions costs by Mulvey et al. [31] can be included in our framework.

Let $S_t \in \mathcal{R}_+^d$ denote the stock price process and assume that one-period interest rate r is constant. The control variable $\pi_t = (\pi_t^1, \ldots, \pi_t^d) \in \mathcal{R}^d$ is the amount of money to be invested in each of the stock. Classically, it is assumed that π_t^i could take any value. Starting with initial wealth of $x > 0$, the self-financing wealth dynamics for the portfolio choice $\pi_t \in A_t \subset \mathcal{R}^d$ is given by,

$$X_{t+1} = X_t + \pi_t \cdot Z_{t+1} + r(X_t - \pi_t \cdot \mathbf{1}) = (1+r)X_t + \pi_t \cdot (Z_{t+1} - r\mathbf{1}) \quad t \in \mathcal{T},$$

where $X_0 = x$, $\mathbf{1} = (1, \ldots, 1) \in \mathcal{R}^d$ and the return process Z is given by,

$$Z_{t+1} = \frac{S_{t+1} - S_t}{S_t} \in \mathcal{R}^d, \quad t \in \mathcal{T}.$$

We consider feedback controls $\pi_t = a(t, X_t^a, Z_t)$ and let X^a be the corresponding wealth process. Then, the classical problem is to maximize $v(a) := \mathbb{E}[u(X_T^a)]$ with a given utility function u.

In our formulation, $\mathcal{X} = \mathcal{R}$, $\mathcal{Z} \subset \mathcal{R}^d$ and $f(t, x, \zeta, a) := (1+r)x + a \cdot (\zeta - r\mathbf{1})$, $\psi \equiv 0$, $\varphi(x) = u(x)$. We refer the reader to Buehler et al. [10, 11]

and the references therein for more general financial markets with frictions and constraints.

Example 4.1. To illustrate the convergence of the algorithm, the potential overlearning, the loss of adaptedness, and their consequences, in Section 9 we study the following stylized example with an explicit solution in detail. We take $T = 2$, initial wealth $X_0 = x = 0$ and an exponential utility $u(x) = 1 - e^{-\lambda x}$ where $\lambda > 0$ is the risk-aversion parameter.

To simplify even further we assume that the initial portfolio $\pi_0 = (1/d, \ldots, 1/d)$ to be uncontrolled. Then, $X_1 = (Z_1 \cdot \mathbf{1})/d - r$ is also uncontrolled. Then, control problem is to choose $a(Z_1) := \pi_1(X_1, Z_1) \in \mathcal{R}^d$ so as to maximize

$$v(a) = \mathbb{E}\left[1 - \exp(-\lambda X_2^a)\right],$$

where $X_2^a = (1+r)X_1 + a(Z_1) \cdot (Z_2 - r\mathbf{1})$. The *certainty equivalent* of a utility value $v < 1$ given by

$$\text{ce}(v) := \frac{1}{\lambda} \ln(1 - v), \quad \Leftrightarrow \quad v = u(\text{ce}(v)),$$

is a more standard way of comparing different utility values. Indeed, the agents with expected utility preferences would be indifferent between the action a and the cash amount of $\text{ce}(v(a))$ as the expected utilities of both positions are equal to each other. Thus, for these agents the cash equivalent of the action a is $\text{ce}(a)$.

In the numerical experiments, to reduce the output noise we fix a unit vector $\eta \in \mathcal{R}^d$ and take $Z_2 = \zeta\eta$ independent of Z_1 with a real-valued Gaussian random variable ζ with mean m and volatility s. Then, with $r = 0$,

$$a^*(z) = a^* = \frac{m}{\lambda s^2}\,\eta, \quad \text{ce}(v^*) = -\frac{m^2}{2\lambda s^2}.$$

5 Overlearning

Given a vector $\alpha = (\alpha_0, \ldots, \alpha_{T-1}) \in \mathcal{A}^T$, we define a constant (in space) action

$$a_\alpha(t, x, \zeta) := \alpha_t, \quad t \in \mathcal{T},\ x \in \mathcal{X},\ \zeta \in \mathcal{Z}$$

which depends only on time. For brevity, we write $x^\alpha(z) := x^{a_\alpha}(z)$ and let

$$\hat{\ell}(\alpha, z) := \ell(a_\alpha, z), \quad \alpha \in \mathcal{A}^T, z \in \mathcal{Z}^T. \tag{5.1}$$

We need the following simple result.

Lemma 5.1. *For every $z \in \mathcal{Z}^T$, $\inf_{\alpha \in \mathcal{A}^T} \hat{\ell}(\alpha, z) = \inf_{a \in \mathcal{C}} \ell(a, z)$.*

Proof. For $z \in \mathcal{Z}^T, a \in \mathcal{C}$ and set $\alpha_t^{(a,z)} := a(t, x_t^a(z), z_t)$, $\alpha^{(a,z)} := (\alpha_0^{(a,z)}, \ldots, \alpha_{T-1}^{(a,z)})$. Then, on the trajectory z (and possibly not on the other trajectories), the original feedback action a and the constant action $\alpha^{(a,z)}$ defined through a, z yield the same state and performance. Indeed, for every $t \in \mathcal{T}$, trivially we have

$$a_{\alpha^{(a,z)}}(t, x, \zeta) = \alpha_t^{(a,z)} = a(t, x_t^a(z), z_t), \quad \forall x \in \mathcal{X}, \zeta \in \mathcal{Z}.$$

Therefore, $x^{\alpha^{(a,z)}}(z) = x^a(z)$, $\hat{\ell}(\alpha^{(a,z)}, z) = \ell(a, z)$. In particular,

$$\inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \hat{\ell}(\alpha, z) \leq \hat{\ell}(\alpha^{(a,z)}, z) = \ell(a, z).$$

Hence $\inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \hat{\ell}(\alpha, z) \leq \inf_{a \in \mathcal{C}} \ell(a, z)$. The opposite inequality is immediate. □

For the training data $\mathcal{L}_n$, set

$$V^*(\mathcal{L}_n) := \frac{1}{n} \sum_{i=1}^{n} \inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \hat{\ell}(\alpha; Z^{(i)}). \tag{5.2}$$

As by the above Lemma, $\inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \hat{\ell}(\alpha, Z^{(i)}) \leq \ell(a, Z^{(i)})$ for any $a \in \mathcal{C}$ and i,

$$V^*(\mathcal{L}_n) = \frac{1}{n} \sum_{i=1}^{n} \inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \hat{\ell}(\alpha, Z^{(i)}) \leq \frac{1}{n} \sum_{i=1}^{n} \ell(a, Z^{(i)}) = L(a; \mathcal{L}_n), \quad \forall\ a \in \mathcal{C}.$$

In view of law of large numbers, $L(a; \mathcal{L}_n)$ is close to $v(a) = \mathbb{E}[\ell(a, Z)]$ for large n. Also in most examples, the above inequality is strict as the minimization in the definition of $V^*(\mathcal{L}_n)$ is pointwise. Hence, typically, one has $V^*(\mathcal{L}_n) < v^*$.

Moreover, for every k,

$$V^*(\mathcal{L}_n) \leq \inf_{\theta \in \mathcal{O}_k} L(\Phi(\cdot; \theta); \mathcal{L}_n) = L(A^*_{k,n}; \mathcal{L}_n), \quad \forall\ k, \tag{5.3}$$

where $L(A^*_{k,n}; \mathcal{L}_n)$ is as in (3.4). Thus, the above inequality shows that the neural network $\mathcal{N}_k$ would try to approximate the optimizer or almost-optimizers of $\hat{\ell}(\cdot, z)$ on the training data $\mathcal{L}_n$. Since it is well-known that sufficiently wide or deep neural networks can learn any finite sequence (c.f. Assumption 3.1), the minimal value $L(A^*_{k,n}; \mathcal{L}_n)$ obtained by the neural networks would be close to $V^*(\mathcal{L}_n)$ as proved in Theorem 5.5 below. As a consequence, neural networks may potentially overperform v^*. We refer to this possibility as *overlearning* the randomness as the networks predict the future values instead of performing a regression analysis.

In optimal control, it is centrally important that the decisions are adapted to the information flow. Sufficiently deep or wide neural networks circumvent this restriction by overlearning the data and are thus able to overperform on the *training data*. However, as the output of the neural networks is in feedback form, technically the trained actions are always adapted. So the issue of non-adaptedness is a subtle and a data-dependent one. Indeed, the *coefficients* of the trained networks use the future data explicitly and therefore become *non-adapted* on the *training data* and overperform on this set. This observation is quite clear in the examples discussed in the next subsection.

5.1 Examples

We return to two examples from Section 4 to clarify the above notions.

Example 5.2. Consider the production planning problem of Section 4.1. The only feedback action in that context is the production decision. For a fixed control $\alpha \in \mathcal{R}$ and a given trajectory of demands $z = (z_1, z_2)$, the cost function $\ell(\alpha, z) = \phi(z_1 + z_2 - \alpha) \geq 0$ is zero at the origin. Hence, $\alpha^*(z) = z_1 + z_2$ is the optimizer and $V^*(\mathcal{L}_n) = 0 < v^*$ for any training set $\mathcal{L}_n = \{Z^{(1)}, Z^{(2)}, \ldots, Z^{(n)}\}$.

More importantly, as discussed above, neural networks may also overperform by getting close to $V^*(\mathcal{L}_n)$. Indeed, if the training data is distinct, any sufficiently deep and wide neural network constructs an approximation $\zeta : \mathcal{R} \to \mathcal{R}$ so that $\zeta(Z_1^{(i)})$ is uniformly close to $Z_2^{(i)}$ for each i. Then, the feedback action $a^*(z_1) := z_1 + \zeta(z_1)$ constructed by the neural network achieves an in-sample performance value of $L(a^*; \mathcal{L}_n)$ which is close to zero. As $v^* > 0$, this would be overlearning and a^* does not generalize. Also on the training set, the action a^* constructed by the network implicitly uses the non-observed demand Z_2.

Example 5.3. Consider the utility maximization problem discussed in Example 4.1 with one stock. Then, for a fixed control $\alpha \in \mathcal{R}$ and given returns $z = (z_1, z_2)$, the cost function is given by $\ell(\alpha, z) = 1 - \exp(((1 + r)(z_1 - r) + \alpha(z_2 - r))$. Then, the optimal portfolio takes unbounded positions depending on the sign of $z_2 - r$. Thus, $V^*(\mathcal{L}_n) = 1 = \sup_x u(x)$ for any training set. In financial terms, large enough neural networks predict the sign of the random variable $Z_2 - r$ by observing Z_1 and use this prediction to create a numerical arbitrage caused by the obvious non-adaptedness and overlearning. Additionally, on the training data the trained feedback actions almost achieve a performance value of one, thus overperforming $v^* < 1$.

5.2 Asymptotic Overlearning

We continue by proving asymptotic overlearning. We first prove the result under the following assumption and provide a general result in Appendix A.

Assumption 5.4. *We assume that the training data is distinct: for every $t \in \mathcal{T}$ and $i \neq j$, $Z_t^{(i)} \neq Z_t^{(j)}$.*

When $Z^{(i)}$'s are drawn independently from an atomless distribution, they would all be distinct. This assumption and the proof of Theorem 5.5 below show the importance of the dimension d. Indeed, in higher dimensions, the training data is 'more and more distinct' allowing easier overlearning. The separation between the training data is also a factor in the Rademacher complexity that is discussed in the next section.

Theorem 5.5. *Let $L(A_{k,n}^*; \mathcal{L}_n)$ be as in* (3.4) *and $V^*(\mathcal{L}_n)$ be as in* (5.2) *and suppose that Assumptions 2.1, 3.1 and 5.4 hold. Then, for every training set $\mathcal{L}_n$,*

$$\lim_{k \to \infty} L(A_{k,n}^*; \mathcal{L}_n) = V^*(\mathcal{L}_n).$$

Proof. Fix $n, \mathcal{L}_n, \epsilon > 0$ and choose $\alpha^\epsilon(\cdot)$ satisfying

$$\hat{\ell}(\alpha^\epsilon(z), z) \leq \inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \hat{\ell}(\alpha, z) + \epsilon,$$

For each i, consider the constant vector $\alpha^{\epsilon,i} = \alpha^\epsilon(Z^{(i)}) \in \mathcal{A}^T$. Since $Z^{(i)}$'s are distinct, there exists a bounded, smooth function $\tilde{a}^\epsilon : \mathcal{T} \times \mathcal{Z} \to \mathcal{A}$, satisfying

$$\tilde{a}^\epsilon(t, Z_t^{(i)}) = \alpha_t^{\epsilon,i} = \alpha_t^\epsilon(Z^{(i)}), \quad t \in \mathcal{T},\ i = 1, 2, \ldots, n.$$

We trivially extend $\tilde{a}^\epsilon$ to $\mathcal{T} \times \mathcal{X} \times \mathcal{Z}$ by setting $a^\epsilon(t, x, \zeta) := \tilde{a}^\epsilon(t, \zeta)$. As

$$a^\epsilon(t, x^{a^\epsilon}(Z^{(i)}), Z^{(i)}) = \tilde{a}^\epsilon(t, Z_t^{(i)}) = \alpha_t^{\epsilon,i}, \quad \forall t \in \mathcal{T},$$

we have $x^{a^\epsilon}(Z^{(i)}) = x^{\alpha^{\epsilon,i}}(Z^{(i)})$ and consequently, $\ell(a^\epsilon, Z^{(i)}) = \hat{\ell}(\alpha^{\epsilon,i}, Z^{(i)})$ for every i. Therefore,

$$L(a^\epsilon; \mathcal{L}_n) = \frac{1}{n}\sum_{i=1}^n \hat{\ell}(\alpha^{\epsilon,i}, Z^{(i)}) \le \frac{1}{n}\sum_{i=1}^n \inf_{\alpha\in\mathcal{A}^\mathcal{T}} \hat{\ell}(\alpha, Z^{(i)}) + \epsilon = V^*(\mathcal{L}_n) + \epsilon.$$

Moreover, by the universal approximation Assumption 3.1, the definition (3.1) of L and the continuity of ℓ as assumed in Assumption 2.1, there is a sequence of neural networks approximating the function a^ϵ, i.e., a sequence of parameters θ_k^ϵ (depending on fixed n) such that

$$\lim_{k\to\infty} L(\Phi(\cdot; \theta_k^\epsilon); \mathcal{L}_n) = L(a^\epsilon; \mathcal{L}_n).$$

Hence,

$$\limsup_{k\to\infty} L(A_{k,n}^*; \mathcal{L}_n) \le \lim_{k\to\infty} L(\Phi(\cdot; \theta_k^\epsilon); \mathcal{L}_n) = L(a^\epsilon; \mathcal{L}_n) \le V^*(\mathcal{L}_n) + \epsilon.$$

We now use a diagonal argument to construct a sequence θ_k satisfying

$$\limsup_{k\to\infty} L(A_{k,n}^*; \mathcal{L}_n) \le \lim_{k\to\infty} L(\Phi(\cdot; \theta_k); \mathcal{L}_n) \le V^*(\mathcal{L}_n).$$

This together with (5.3) completes the proof. □

6 Rademacher Complexity

We first recall several classical definitions and results: see Bartlett and Mendelson [3], Bousquet et al. [9], Mohri et al. [30], Shalev-Shwartz and Ben-David [34]. Let $\mathcal{G}$ be a hypothesis class of a set of real-valued functions defined on the set of trajectories.

Definition 6.1. The *empirical Rademacher complexity* of $\mathcal{G}$ on the training set $\mathcal{L}_n$ is defined to be

$$R_e(\mathcal{G}, \mathcal{L}_n) := \mathbb{E}\left[\sup_{g\in\mathcal{G}} \frac{1}{n}\sum_{i=1}^n \sigma_i g(Z^{(i)})\right],$$

where the expectation is over the *Rademacher variables* σ_i's, which are identically and independently distributed taking values ± 1 with equal probability.

Definition 6.2. For a probability measure ν on $\mathcal{Z}^T$, the *Rademacher complexity* of $\mathcal{G}$ for the distribution ν is defined to be

$$r_\nu(\mathcal{G}, n) := \mathbb{E}_\nu \left[R_e(\mathcal{G}, \mathcal{L}_n) \right],$$

where the expectation is over the random training set $\mathcal{L}_n = \{Z^{(1)}, Z^{(2)}, \dots, Z^{(n)}\}$ whose elements are independently and identically drawn from ν.

Let $v : \mathcal{C} \to \mathcal{R}$ be as in (2.1), L be as in (3.1). For a neural network $\mathcal{N}_k$ and a training set $\mathcal{L}_n$, set

$$G_\nu(\mathcal{N}_k, \mathcal{L}_n) := \sup_{\theta \in \mathcal{O}_k} |v(\Phi(\cdot;\theta)) - L(\Phi(\cdot;\theta); \mathcal{L}_n)|.$$

The following result that uniformly connects empirical averages to expected values is classical. Since by Assumption 2.1, $|\ell|(a,\cdot)| \le c^*$ for every $a \in \mathcal{A}$, $|L|, |v| \le c^*$ as well. Then, for a given $\delta \in (0,1)$, with probability at least $1-\delta$ the following estimates hold,

$$G_\nu(\mathcal{N}_k, \mathcal{L}_n) \le c_\nu(\mathcal{N}_k, n, \delta) \le C_e(\mathcal{N}_k, \mathcal{L}_n, \delta), \tag{6.1}$$

where with $\ell(\mathcal{N}_k) = \{\ell(\Phi(\cdot;\theta) | \theta \in \mathcal{O}_k\}$,

$$c_\nu(\mathcal{N}_k, n, \delta) := 2 r_\nu(\ell(\mathcal{N}_k); n) + 2c^* \sqrt{\frac{\ln(2/\delta)}{2n}},$$
$$C_e(\mathcal{N}_k, \mathcal{L}_n, \delta) := 2 R_e(\ell(\mathcal{N}_k); \mathcal{L}_n) + 6c^* \sqrt{\frac{\ln(2/\delta)}{n}}.$$

One-sided version of these estimates for functions $0 \le g \le 1$ is proved in Theorem 3.3 by Mohri et al. [30] and elementary arguments yield the above two-sided estimates.

Remark 6.3. The random variable $C_e(\mathcal{N}_k, \mathcal{L}_n, \delta)$ is an empirical quantity independent of the distribution ν. Theoretically, it can be calculated once the training data is given.

The random variables $G_\nu(\mathcal{N}_k, \mathcal{L}_n), C_e(\mathcal{N}_k, \mathcal{L}_n, \delta)$ and the constant $c_\nu(\mathcal{N}_k, n, \delta)$ are increasing as the neural networks $\mathcal{N}_k$ get wider and deeper. The monotonicity of δ is also clear. One may obtain further estimates by using the Rademacher calculus as described in Section 26.1 by Shalev-Shwartz and Ben-David [34]. Indeed, if the mapping $a \in \mathcal{C} \mapsto \ell(a,z)$ is uniformly Lipschitz, then, the Kakade & Tewari composition Lemma (see Kakade and Tewari [26], also Lemma 26.9 in Shalev-Shwartz and Ben-David [34]) implies that one can estimate the complexities $R_e(\ell(\mathcal{N}_k); \mathcal{L}_n)$ and $r_\nu(\ell(\mathcal{N}_k); n)$, by the Rademacher complexities $R_e(\mathcal{N}_k; \mathcal{L}_n)$, $r_\nu(\mathcal{N}_k; n)$ of the neural networks.

Moreover, as a consequence of the Massart Lemma and the composition lemma, the Rademacher complexity $r_\nu(\mathcal{N}_k; n)$ of the neural networks converges to zero as the size n of the training data goes to infinity; see for example problem 3.11 in Mohri et al. [30] or Corollary 3.8 in the lecture notes Wolf [36]. In fact detailed estimates are also available in Golowich et al. [17], Neyshabur et al. [32]. Since the regularity of ℓ can be directly proven under Lipschitz assumptions on the coefficients of the decision problem, this procedure shows that under natural assumptions on the coefficients, the complexity $r_\nu(\ell(\mathcal{N}_k); n)$ also converges to zero.

We may also use the techniques developed by E et al. [13] for estimating the Rademacher complexity of residual networks to obtain upper bounds for the complexities appearing our analysis.

7 Complexity Estimates

Let v^* be as in (2.3) and $A^*_{k,n}$, $L(A^*_{k,n}; \mathcal{L}_n)$, $L(A^*_{k,n}; \hat{\mathcal{L}}_n)$, $v(A^*_{k,n})$ be as in Section 3. In this section we prove empirical bounds on their differences.

Theorem 7.1. *Under Assumptions 2.1 and 3.1, for every $\epsilon > 0$ there exists k_ϵ such that*

$$\left|v^* - L(A^*_{k,n}; \mathcal{L}_n)\right| \leq G_\nu(\mathcal{N}_k, \mathcal{L}_n) + \epsilon, \quad \forall\ k \geq k_\epsilon. \tag{7.1}$$

In particular, for all $\delta > 0$ the followings holds with at least $1-\delta$ probability for every $k \geq k_\epsilon$,

$$\begin{aligned} \left|v^* - L(A^*_{k,n}; \mathcal{L}_n)\right| &\leq c_\nu(\mathcal{N}_k, n, \delta) + \epsilon \leq C_e(\mathcal{N}_k, \mathcal{L}_n, \delta) + \epsilon, \\ \left|v^* - v(A^*_{k,n})\right| &\leq 2c_\nu(\mathcal{N}_k, n, \delta) + \epsilon \leq 2C_e(\mathcal{N}_k, \mathcal{L}_n, \delta) + \epsilon. \end{aligned} \tag{7.2}$$

Proof. For $\epsilon > 0$ choose $a_\epsilon \in \mathcal{C}$ satisfying $v(a_\epsilon) \leq v^* + \frac{1}{2}\epsilon$. By Assumption 3.1, there exists a sequence θ_k and k_ϵ so that

$$v(\Phi(\cdot; \theta_k)) \leq v(a_\epsilon) + \frac{1}{2}\epsilon \leq v^* + \epsilon, \quad \forall k \geq k_\epsilon.$$

Since $L(A^*_{k,n}; \mathcal{L}_n) \leq L(\Phi(\cdot; \theta); \mathcal{L}_n)$ for any $\theta \in \mathcal{O}_k$, the definition of G_ν implies that

$$\begin{aligned} L(A^*_{k,n}; \mathcal{L}_n) &\leq L(\Phi(\cdot; \theta_k); \mathcal{L}_n) \leq v(\Phi(\cdot; \theta_k)) + G_\nu(\mathcal{N}_k, \mathcal{L}_n) \\ &\leq v^* + G_\nu(\mathcal{N}_k, \mathcal{L}_n) + \epsilon, \quad \forall k \geq k_\epsilon. \end{aligned}$$

As $v^* \leq v(\Phi(\cdot, \theta))$ for any $\theta \in \mathcal{O}_k$ and $A^*_{k,n} = \Phi(\cdot, \theta^*_{k,n})$,

$$v^* \leq v(A^*_{k,n}) \leq L(A^*_{k,n}; \mathcal{L}_n) + G_\nu(\mathcal{N}_k, \mathcal{L}_n).$$

Now (7.1) follows directly from the above inequalities, and (7.2) from (7.1) and (6.1). Finally,

$$\left|v^* - v(A^*_{k,n})\right| \leq \left|v^* - L(A^*_{k,n}; \mathcal{L}_n)\right| + \left|v(A^*_{k,n}) - L(A^*_{k,n}; \mathcal{L}_n)\right| \leq 2G_\nu(\mathcal{N}_k, \mathcal{L}_n) + \epsilon.$$

□

Let $\hat{\mathcal{L}}_n$ be another data set drawn identically and independently from the distribution ν. By (6.1), the following holds with at least $1-\delta$ probability for every $\theta \in \mathcal{O}_k$,

$$\begin{aligned}\left|L(\Phi(\cdot;\theta);\hat{\mathcal{L}}_n) - L(\Phi(\cdot;\theta);\mathcal{L}_n))\right| \le & \, |v(\Phi(\cdot,\theta)) - L(\Phi(\cdot;\theta);\mathcal{L}_n))| \\ & + \left| \, v(\Phi(\cdot,\theta)) - L(\Phi(\cdot;\theta);\hat{\mathcal{L}}_n)\right| \\ \le & G_\nu(\mathcal{N}_k,\mathcal{L}_n) + G_\nu(\mathcal{N}_k,\hat{\mathcal{L}}_n) \le 2c_\nu(\mathcal{N}_k,n,\delta/2).\end{aligned}$$

This shows that the in and out-of-sample performance difference of any network provides an empirical lower bound for the Rademacher complexity with high probability. In fact, in many applications it is a standard practice to monitor this difference. Thus, also in view of the estimate (7.2), the following quantity, *maximal performance difference*, could be taken as a proxy for overlearning,

$$O(\mathcal{N}_k,\mathcal{L}_n,\hat{\mathcal{L}}_n) := \sup_{\theta\in\mathcal{O}_k} |L(\Phi(\cdot,\theta);\mathcal{L}_n) - L(\Phi(\cdot,\theta);\hat{\mathcal{L}}_n)|.$$

We restate that the following holds with at least $1-\delta$ probability,

$$|L(\Phi(\cdot,\theta);\mathcal{L}_n) - L(\Phi(\cdot,\theta);\hat{\mathcal{L}}_n)| \le O(\mathcal{N}_k,\mathcal{L}_n,\hat{\mathcal{L}}_n) \le 2c_\nu(\mathcal{N}_k,n,\delta/2), \quad \forall \theta \in \mathcal{O}_k. \tag{7.3}$$

8 Convergence

We make the following assumption on the complexity of the neural networks $\mathcal{N}_k$.

Assumption 8.1. *We assume that for each k, the Rademacher complexity $r_\nu(\ell(\mathcal{N}_k),n)$ converges to zero as the training size n tends to zero.*

As discussed in Remark 6.3, the above assumption holds under natural assumptions on the coefficients. We consider a sequence of training sets $\mathcal{L}_n$ and out-of-sample sets $\hat{\mathcal{L}}_n$ drawn identically and independently from ν. $A^*_{k,n}$ is the optimal feedback action that could be constructed by the neural network $\mathcal{N}_k$ using $\mathcal{L}_n$, c.f., (3.4). Set $v^*_k := \inf_{\theta\in\mathcal{O}_k} v(\Phi(\cdot;\theta))$.

Corollary 8.2. *Under the Assumptions 2.1 and 8.1, the following holds with probability one,*

$$\lim_{n\to\infty} L(A^*_{k,n};\mathcal{L}_n) = \lim_{n\to\infty} L(A^*_{k,n};\hat{\mathcal{L}}_n) = \lim_{n\to\infty} v(A^*_{k,n}) = v^*_k.$$

Under the approximation Assumption 3.1, v^*_k converges to v^* as proved in Lemma 3.2. Hence, the above result states that as the size of training data increases, the performance of the feedback actions constructed by the neural networks converge to the optimal value provided that the size of the neural networks also tends to infinity in a controlled manner.

Proof. By the definition of G_ν used at $A^*_{k,n}$ and at $\Phi(\cdot;\theta_k)$, we obtain the following

$$\begin{gathered} v^*_k \le v(A^*_{k,n}) \le L(A^*_{k,n};\mathcal{L}_n) + G_\nu(\mathcal{N}_k,\mathcal{L}_n), \\ L(A^*_{k,n};\mathcal{L}_n) \le L(\Phi(\cdot;\theta_k),\mathcal{L}_n) \le v^*_k + G_\nu(\mathcal{N}_k,\mathcal{L}_n). \end{gathered}$$

Hence,

$$\left|v^*_k - L(A^*_{k,n};\mathcal{L}_n)\right| \le G_\nu(\mathcal{N}_k,\mathcal{L}_n).$$

Fix $\epsilon > 0$ and set $\delta_n = 2\exp(-2n\epsilon^2/(6c^*)^2)$ so that

$$c_\nu(\mathcal{N}_k,n,\delta_n) = 2r_\nu(\ell(\mathcal{N}_k);n) + 6c^*\sqrt{\frac{\ln(2/\delta_n)}{2n}} = 2r_\nu(\ell(\mathcal{N}_k),n) + \epsilon.$$

Then, by (6.1), for every k with at least $1-\delta_n$ probability

$$\left|v^*_k - L(A^*_{k,n};\mathcal{L}_n)\right| \le C_e(\mathcal{N}_k,\mathcal{L}_n,\delta_n) = 2r_\nu(\ell(\mathcal{N}_k),n) + \epsilon.$$

Equivalently, $\mathbb{P}(\Omega_{k,n,\epsilon}) \le \delta_n$, where

$$\Omega_{k,n,\epsilon} := \left\{\left|v^*_k - L(A^*_{k,n};\mathcal{L}_n)\right| > 2r_\nu(\ell(\mathcal{N}_k),n) + \epsilon\right\}.$$

Since $\sum_n \delta_n < \infty$, by the Borel-Cantelli Lemma, for every k,

$$\limsup_{n\to\infty}\left|v^*_k - L(A^*_{k,n};\mathcal{L}_n)\right| \le \lim_{n\to\infty} 2r_\nu(\ell(\mathcal{N}_k),n) + \epsilon = \epsilon,$$

with probability one.

In view of (7.3), by at least $1-\delta_n$ probability

$$\left|L(A^*_{k,n};\hat{\mathcal{L}}_n) - L(A^*_{k,n};\mathcal{L}_n)\right| \le 2c_\nu(\mathcal{N}_k,n,\delta_n/2) = 4r_\nu(\ell(\mathcal{N}_k),n) + \ln(4)\epsilon.$$

The above Borel-Cantelli argument also implies that with probability one,

$$\limsup_{n\to\infty}\left|L(A^*_{k,n};\hat{\mathcal{L}}_n) - L(A^*_{k,n};\mathcal{L}_n))\right| \le \ln(4)\epsilon.$$

□

9 Numerical Experiments

In this section we present the numerical implementations of Example 4.1. We take $\lambda = 1, r = 0$ and as discussed in that example the return of the second period $Z_2 = \zeta\eta$ where ζ is Gaussian with mean 18% and volatility 0.44%[4] and is independent of Z_1. Then, the optimal solution given in Example 4.1

[4]We have chosen the μ and σ values randomly among those with a^* close to one and which are neither too small or large. For these parameter values, overlearning is not particularly easy.

is $a^* = 0.9297$. To focus the training on a compact input domain, Z_1 is distributed uniformly over $[-0.5, 0.5]^d$. We use the certainty equivalent ce defined in Example 4.1 to compare the performances of different actions.

As our main goal is to illustrate the potential overlearning, we try to strike a balance between avoiding unnecessary tuning parameters while still implementing commonly accepted best practices. The simple but representative structure of the chosen example allows us to easily evaluate the trained feedback actions by comparing them to explicit formulae and also provides an understanding of the performance of this algorithm on a general class of decision problems. We emphasize that our claim is not that overlearning cannot be alleviated in these problems, but that it does occur even with a seemingly reasonable learning setup and that one has to be aware of the possibility. Indeed some degree of tuning could possibly lead to improvement in this particular example, but such methods are not systematic, and it is not clear that they generalize when the ground truth is not available. Corollary 8.2 and Lemma 3.2 show that increasing the training set (and possibly the architecture complexity in a controlled manner) does provide a systematic method for improvement. This is also observed in the computations that follow.

To describe our findings succinctly, let a^* be the (constant) optimal feedback action and $A_{k,n}$ be the feedback action computed by the neural network $\mathcal{N}_k$ on the training set $\mathcal{L}_n$. Although the optimization algorithm is trying to compute the minimizer $A^*_{k,n}$ of (3.4), in actual computations, the stochastic gradient algorithm is stopped before reaching $A^*_{k,n}$. Thus, $A_{k,n}$ depends not only on the training data $\mathcal{L}_n$ and the network $\mathcal{N}_k$ but also on the optimization procedure, in particular, the stopping rule.

By taking advantage of the explicity available solution, we define the in-sample relative performance p_{in} and the out-of-sample relative performance p_{out} of the trained actions $A_{k,n}$ by,

$$p_{in} := \frac{\text{nn}_{\text{in-sample}} - \text{true}_{\text{in-sample}}}{\text{true}_{\text{in-sample}}},$$

$$p_{out} := \frac{\text{nn}_{\text{out-of-sample}} - \text{true}_{\text{out-of-sample}}}{\text{true}_{\text{out-of-sample}}},$$

where

$$\begin{aligned} \text{nn}_{\text{in-sample}} &:= \text{ce}(L(A_{k,n}; \mathcal{L}_n)), & \text{nn}_{\text{out-of-sample}} &:= \text{ce}(L(A_{k,n}; \hat{\mathcal{L}}_n)), \\ \text{true}_{\text{in-sample}} &:= \text{ce}(L(a^*; \mathcal{L}_n)), & \text{true}_{\text{out-of-sample}} &:= \text{ce}(L(a^*; \hat{\mathcal{L}}_n)), \end{aligned}$$

and $\mathcal{L}_n$ is the training set used to compute $A_{k,n}$ and $\hat{\mathcal{L}}_n$ is the training set chosen identically and independently of $\mathcal{L}_n$. Then, the appropriately normalized performance difference $p_{in} - p_{out}$ provides an understanding of the overlearning proxy $O(\mathcal{N}_k, \mathcal{L}_n, \hat{\mathcal{L}}_n)$ as in (7.3). Indeed, larger values of the difference imply larger values of O.

We focus on these measures, p_{in}, p_{out}, for two reasons. Firstly, although our samples are large enough to give a good representation of the distribution, the above formulae eliminate some dependency on the sample by subtracting the true optimizers performance on each sample. Secondly, there are circumstances where seemingly the training immediately tries to interpolate data instead of first approaching the true solution before starting to interpolate, as one might expect. This leads the out-of-sample performance to increase very early on, and with our stopping rule based on the out-of-sample performance, it thus leads to almost immediate stopping. In this sense, the combination of stopping rule and performance measure is relatively conservative for measuring overlearning.

9.1 Implementation Details

Our implementation is written in the programming language OCaml [29] using the library Owl [35] In all examples, the activation functions are set to ReLU and the parameters are optimized by stochastic gradient descent using the Adam scheme with parameters $(\alpha, \beta_1, \beta_2) = (0.001, 0.9, 0.999)$, as proposed by Kingma and Ba [27]. The neural networks are constructed with three hidden layers. This architecture is kept fixed regardless of data dimensionality in order to better isolate the dimensionality's impact on overlearning. The weights are initialized with a uniform centered distribution of width inversely proportional to the square root of the number neuron inputs.[5]

As the neural networks are capable of overlearning the data, we must employ stopping rules for early stopping. Such stopping rules are commonly used in practice as implicit regularizers. In our studies we mainly use a conservative stopping rule that monitors the out-of-sample performance after each epoch and terminates when the out-of-sample performance exceeds its past minimum.[6] To demonstrate the potential overlearning, we also performed some experiments running the stochastic gradient without stopping for a fixed number of epochs.

We train using minibatches sampled randomly from the training set. Overlearning can also be observed with batch gradient descent—equivalent to the extreme case of setting the minibatch size to the full training set—but we have opted to default to minibatches as it is far more common and computationally efficient[7]. On the issue of minibatch size, we use the Keras default of 32.

[5]The uniform He-initializer He et al. [21]—which differs only by a factor $\sqrt{6}$ in the width of the uniform distribution and is commonly recommended for training ReLU networks—has not shown qualitatively different results with regards to overlearning.

[6]As the parameter landscape is expected to have plateaus and the out-of-sample performance is not expected to be perfectly monotone, this calls for some tolerance, thereby introducing a tuning parameter. To be conservative, we keep this tolerance small to encourage early stopping and reduce overlearning.

[7]The computational burden of each gradient computation scales as $O(N)$ in the batch size N, but the accuracy is of order $O(1/\sqrt{N})$, leading to computational advantages of small batch sizes (but not too small, due to SIMD instructions in modern CPUs and GPUs). It is sometimes argued that the more 'chaotic' nature of small batches leads to beneficial regularization. However, due to the complex interaction between the batch size and the stopping rule, the effect of this is not clear-cut.

9.2 Results

Table 1 reports the neural network's average relative in-sample performance, and its comparison to the out-of-sample performance with the above described conservative stopping rule. For each dimension, the corresponding μ value is the average of 30 runs and σ is the standard deviation. We keep the data size of $N = 100,000$ and the network architecture of three hidden layers of width 10 fixed. Even though with this rule the stochastic gradient descent is stopped quite early, there is substantial overperformance increasing with dimension.

dims	p_{in} (%)		$p_{in} - p_{out}$ (%)	
	μ	σ	μ	σ
100	10.12820	1.09290	23.67080	2.01177
85	8.38061	1.35575	20.16440	2.30489
70	7.32720	0.86458	15.62060	1.94043
55	5.05783	0.81518	10.93950	1.54431
40	3.74648	0.62588	7.91105	1.32581
25	2.11501	0.43845	4.58954	0.88461
10	0.53982	0.34432	1.46138	0.39078

Table 1: Average relative in-sample performance, and its comparison to the out-of-sample performance with the above described conservative stopping rule. Everything is in % with training size of $N = 100,000$ and three hidden layers of width 10. The μ value is the average of 30 runs and σ is the standard deviation.

To isolate the impact of the dimension, in the second experiment, we keep all parameters except the width of layers as before. The last two hidden layers again have width 10. But the width of the first hidden layer is adjusted so that the number of parameters is equal to that of a neural network with three hidden layers of width 10 and input layer of dimension as in column 'parameters-equivalent'. There are three groups with parameters-equivalent dimensions of 40, 70 and 100. For example in the group with parameters-equivalent dimension 70, in the row with actual dimension 70, all layers have width 10. But in that group, the networks for the actual dimensions of 40 and 10 have wider first layer so that they all have the same number of parameters. Table 2 also shows a clear increase of overlearning with dimension. Although, the architecture is not exactly same, we believe that this experiment shows that the apparent dimensional dependence is not simply due to the increase in the number of parameters.

We also implemented an aggressive optimization by running the algorithm for 100 and 200 epochs in 100 dimensions without a stoping rule with other parameters as in Table 1. In these experiments the trained actions $A_{k,n}$ are closer to the optimal actions $A^*_{k,n}$ and Theorem 5.5 predicts a larger overperformance. Indeed, Table 3 shows that, overlearning is quite substantial even with a training size of $100,000$ and there is a noticeable deterioration in the out-of-sample performance.

Finally, Table 4 illustrates the convergence proved in Section 8. In 100

dims	params-equiv	p_{in} (%)		$p_{in} - p_{out}$ (%)	
		μ	σ	μ	σ
100	100	10.12820	1.09290	23.67080	2.01177
70	100	8.86214	1.45962	21.65000	3.12209
40	100	7.28550	1.19811	15.27540	2.10167
10	100	1.99793	0.54664	4.18041	1.22285
70	70	7.32720	0.86458	15.62060	1.94043
40	70	5.67500	0.84644	12.45610	1.90450
10	70	1.50328	0.93772	3.46245	1.19606
40	40	3.74648	0.62588	7.91105	1.32581
10	40	1.13566	0.65512	2.84677	0.78069

Table 2: All other parameters except the width of layers are as in Table 1. The last two hidden layers again have width 10 and the width of the first hidden layer is adjusted so that the number of parameters is equal to that of a neural network with three hidden layers of width 10 and the number of dimension is as in parameters-equivalent.

epochs	p_{in} (%)		$p_{in} - p_{out}$ (%)	
	μ	σ	μ	σ
200	30.3161	2.46850	315.875	540.4750
100	25.8374	1.72027	111.553	41.5841

Table 3: Longer iterations performance in 100 dimensions. Based on 15 runs. Especially the 200 epoch runs show signs of a heavy tail, as expected with high degrees of overlearning. All other parameters as in Table 1.

dimensions we increase the size of the training data from $100,000$ to twenty-fold keeping all the other parameters as in the first experiment. The results show a remarkable improvement in the accuracy demonstrating the power of the deep Monte Carlo optimization.

10 Conclusions

The deep neural-network optimization is a highly effective computational tool for the study of stochastic optimal control problems or equivalently, decision making under uncertainty. It can handle general random structures in high dimensions and complex dynamics with ease. The simplicity of the algorithm and the recent advances in deep neural networks are key to these properties. As one needs sufficient complexity of the neural networks to achieve appropriate accuracy, the size of the training set is critical and thus, one has to enrich it when the historical data is not large enough.

By numerical experimentation and also by theoretical results, we have demonstrated that the networks have the capability to overlearn the data and consequently construct forward looking feedback actions. These solutions may then

sample size	dims	p_{in} (%)		$p_{in} - p_{out}$ (%)	
		μ	σ	μ	σ
2,000,000	100	0.49597	0.11849	1.12846	0.32184
1,000,000	100	1.14094	0.14640	2.39532	0.27415
500,000	100	2.36352	0.20154	5.24018	0.81235
250,000	100	4.41388	0.37928	10.02040	1.45355

Table 4: Performance for larger sample sizes. Based on 15 runs. All other parameters as in Table 1.

be non-adapted to the flow of information on the training data. However, the estimates proved in Theorem 7.1 shows that overlearning is negligible when data compared to the complexity of the networks is sufficiently large.

Although, this approach is particularly valuable in high dimensions, the overlearning becomes easier in such problems requiring richer training sets as also clearly demonstrated by the numerical studies reported in Section 9. For optimal control, an in-depth-study of this dependence both numerically and theoretically remains an interesting question. The asymptotic overlearning result Theorem 5.5 provides an initial insight indicating that the average distance between the data points and the regularity of the networks are important for a better understanding of this dependence. Indeed, the Rademacher complexity which is present in the upper bound (7.2) is also influenced by both of them. The closely related covering numbers providing an upper for the Rademacher complexity [cf. 34, Lemma 27.4] could also be useful in better understanding of this dependence.

A Appendix: Asymptotic Overlearning

In this section, we prove an extension of Theorem 5.5 without Assumption 5.4.

Fix a training set $\mathcal{L}_n$. Let $\mathcal{K} = \{\mathcal{K}^{(1)}, \ldots, \mathcal{K}^{(m)}\}$ be a partition of $\mathcal{L}_n$ satisfying:

- $\mathcal{K}^{(j)}$'s are disjoint subsets of $\mathcal{L}_n$;
- $\cup_j \mathcal{K}^{(j)} = \mathcal{L}_n$;
- if $z \in \mathcal{K}^{(j)}$ for some j, and if there is a trajectory $\hat{z} \in \mathcal{L}_n$ and $t \in \mathcal{T}$ such that $z_t = \hat{z}_t$, then $\hat{z} \in \mathcal{K}^{(j)}$.

There are partitions satisfying the above conditions and one can even define and would like to use the maximal partition satisfying above conditions. As this is tangential to the main thrust of the paper, we do not pursue it here. When the data is distinct, the maximal partition is $\mathcal{K}^{(i)} = \{Z^{(i)}\}$ and we are back in the setting of Theorem 5.5.

For a constant control $\alpha = (\alpha_0, \dots, \alpha_{T-1}) \in \mathcal{A}^{\mathcal{T}}$, let $\hat{\ell}(\alpha, z)$ be as in (5.1). For $j = 1, \dots, m$, define

$$\overline{\ell}(\alpha, j) := \frac{1}{\left|\mathcal{K}^{(j)}\right|} \sum_{z \in \mathcal{K}^{(j)}} \hat{\ell}(\alpha, z).$$

Let $\alpha^{\epsilon}(j) \in \mathcal{A}^{\mathcal{T}}$ be an ϵ-minimizer of $\overline{\ell}(\cdot, j)$. Analogously to $V^*(\mathcal{L}_n)$ defined in Section 5, define

$$\overline{V}^*(\mathcal{L}_n, \mathcal{K}) := \lim_{\epsilon \downarrow 0} \frac{1}{m} \sum_{j=1}^{m} \overline{\ell}(\alpha^{\epsilon}(j), j) = \frac{1}{m} \sum_{j=1}^{m} \inf_{\alpha \in \mathcal{A}^{\mathcal{T}}} \overline{\ell}(\alpha, j).$$

For $z \in \mathcal{L}_n$, let $j(z)$ be the unique index so that $z \in \mathcal{K}^{(j(z))}$. We now follow the arguments of Theorem 5.5 *mutadis mutandis* to show that the neural networks can approximate the function

$$\overline{a}^*_{\epsilon}(z) := \alpha^{\epsilon}(j(z)), \quad z \in \mathcal{L}_n.$$

This implies the following extension of the overlearning result Theorem 5.5.

Lemma A.1. *Let $L(A^*_{k,n}; \mathcal{L}_n)$ be as in* (3.4). *Under the universal approximation Assumption 3.1,*

$$\lim_{k \to \infty} L(A^*_{k,n}; \mathcal{L}_n) \le \overline{V}^*(\mathcal{L}_n, \mathcal{K}).$$

When the partition $\mathcal{K}$ of $\mathcal{L}_n$ is non-trivial and if the number of partitions m is large, then we may have $\overline{V}^*(\mathcal{L}_n) < v^*$ and consequently potential overlearning. The robust approach used in Bertsimas et al. [7, 8] and also in Esfahani and Kuhn [14], Bartl et al. [2], essentially groups the elements of the training set into a small number of sets and identifies them by a representative element of these sets. If we then partition this processed data, this would result in a small number of partitions and the overlearning will not be possible even with modest size training sets.

Acknowledgments. Reppen is partly supported by the Swiss National Science Foundation grant SNF 181815. The authors thank Professors E, Fan and Han of Princeton University or their valuable comments.

References

[1] A. Bachouch, C. Huré, N. Langrené, and H. Pham. Deep neural networks algorithms for stochastic control problems on finite horizon, part 2: Numerical applications. *arXiv preprint:1812.05916*, 2018.

[2] D. Bartl, S. Drapeau, and L. Tangpi. Computational aspects of robust optimized certainty equivalents and option pricing. *Mathematical Finance*, 30(1):287–309, 2020.

[3] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[4] S. Becker, P. Cheridito, and A. Jentzen. Deep optimal stopping. *Journal of Machine Learning Research*, 20(4):1–25, 2019.

[5] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[7] D. Bertsimas, S. Shtern, and B. Sturt. A data-driven approach for multi-stage linear optimization. *Optimization Online*, 2018.

[8] D. Bertsimas, S. Shtern, and B. Sturt. Two-stage sample robust optimization. *arXiv preprint:1907.07142*, 2019.

[9] O. Bousquet, S. Boucheron, and G. Lugosi. Introduction to statistical learning theory. In *Summer School on Machine Learning*, pages 169–207. Springer, 2003.

[10] H. Buehler, L. Gonon, J. Teichmann, and B. Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.

[11] H. Buehler, L. Gonon, J. Teichmann, B. Wood, and B. Mohan. Deep hedging: hedging derivatives under generic market frictions using reinforcement learning. Technical report, Swiss Finance Institute, 2019.

[12] G. Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:183–192, 1989.

[13] W. E, C. Ma, and Q. Wang. A priori estimates of the population risk for residual networks. *arXiv preprint:1903.02154*, 2019.

[14] P. M. Esfahani and D. Kuhn. Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1-2):115–166, 2018.

[15] W. H. Fleming and H. M. Soner. *Controlled Markov processes and viscosity solutions*, volume 25. Springer Science & Business Media, 2006.

[16] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias-variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[17] N. Golowich, A. Rakhlin, and O. Shamir. Size-independent sample complexity of neural networks. *Information and Inference: A Journal of the IMA*, 9(2):473–504, 2020.

[18] J. Han and W. E. Deep learning approximation for stochastic control problems. In *Deep Reinforcement Learning Workshop, NIPS*, 2016.

[19] J. Han and J. Long. Convergence of the deep BSDE method for coupled FBSDEs. *arXiv preprint:1811.01165*, 2018.

[20] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[22] P. Henry-Labordère. Deep primal-dual algorithm for BSDEs: Applications of machine learning to CVA and IM. *SSRN 3071506*, 2017.

[23] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[24] C. Huré, H. Pham, A. Bachouch, and N. Langrené. Deep neural networks algorithms for stochastic control problems on finite horizon, part I: convergence analysis. *arXiv preprint:1812.04300*, 2018.

[25] J. M. Hutchinson, A. W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3):851–889, 1994.

[26] S. M. Kakade and A. Tewari. On the generalization ability of online strongly convex programming algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808, 2009.

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

[28] A. Kondratyev and C. Schwarz. The market generator. *Available at SSRN*, 2019.

[29] X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy, and J. Vouillon. *The OCaml system release 4.10*, 2 2020. https://caml.inria.fr/pub/docs/manual-ocaml/.

[30] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning.* MIT press, 2018.

[31] J. M. Mulvey, Y. Sun, M. Wang, and J. Ye. Optimizing a portfolio of mean-reverting assets with transaction costs via a feedforward neural network. *Quantitative Finance*, pages 1–23, 2020.

[32] B. Neyshabur, R. Tomioka, and N. Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.

[33] R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455, 2009.

[34] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[35] L. Wang. Owl: A general-purpose numerical library in OCaml. *arXiv preprint:1707.09616*, 2018.

[36] M. M. Wolf. Mathematical foundations of supervised learning, 2018. (Lecture notes from Technical University of Munich.).