Decoupling Gradient-Like Learning Rules from Representations

Philip S. Thomas [1] **Christoph Dann** [2] **Emma Brunskill** [3]

Abstract

In machine learning, *learning* often corresponds to changing the parameters of a parameterized function. A *learning rule* is an algorithm or mathematical expression that specifies precisely how the parameters should be changed. When creating a machine learning system, we must make two decisions: what representation should be used (i.e., what parameterized function should be used) and what learning rule should be used to search through the resulting set of representable functions. In this paper we focus on gradient-like learning rules, wherein these two decisions are coupled in a subtle (and often unintentional) way. That is, using the same learning rule with two different representations that can represent the same sets of functions can result in two different outcomes. After arguing that this coupling is undesirable, particularly when using neural networks, we present a method for partially decoupling these two decisions for a broad class of gradient-like learning rules that span unsupervised learning, reinforcement learning, and supervised learning.

1. Introduction

Consider two challenges at the foundation of *machine learning* (ML) research and practice: representation selection and learning rule selection. *Representation selection* is the decision of how knowledge should be represented within an ML system. *Learning rule selection* is the decision of which algorithm should be used to modify the system's stored knowledge. These two challenges are central to most ML systems, including unsupervised, supervised, and reinforcement learning systems.

Representation and learning rule selection are intertwined—a learning rule can work well with some representations and work poorly with, or be incompatible with, others. Although this intertwining is unavoidable and is likely desirable, it can have unintended and undesirable effects. To see this, we split representation selection into two components: deciding *what* the system should be able to represent (e.g., normal distributions) and *how* the system will represent it (e.g., by storing the mean and standard deviation or variance). The intertwining of the learning rule with the decision of *what* the system should be able to represent is often considered by the designer of an ML system. However, the intertwining of the learning rule with the decision of *how* the system will represent knowledge is not necessarily desirable, is often overlooked, and can have significant ramifications.

Consider an example, which we adapt from an example presented by Amari (1985), where part of an ML system approximates an unknown distribution that generated some observed data, $X_1, X_2, \ldots, X_n$, where each $X_i \in \mathbb{R}$. We might decide that the system will estimate the distribution using a normal distribution (*what* the system can represent) and that the normal distribution will be represented by storing its mean, μ, and standard deviation, σ (*how* the system represents it). We might then decide to find the model, (μ, σ), that maximizes the log-likelihood: $L(\mu, \sigma | X_1, \ldots, X_n) = \ln\left(\Pr\left(X_1, \ldots, X_n | \mu, \sigma\right)\right)$. To maximize the log-likelihood, we might decide to use the gradient ascent learning rule with step size $\alpha \coloneqq .001/n$, resulting in the updates: $\mu_{i+1} = \mu_i + \frac{\alpha}{\sigma_i^2} \sum_{j=1}^{n} (X_j - \mu_i)$, and $\sigma_{i+1} = \sigma_i - \frac{\alpha n}{\sigma_i} + \frac{\alpha}{\sigma_i^3} \sum_{j=1}^{n} (X_j - \mu_i)^2$.

Clearly in this setting the decision to model the distribution that generated the data using a normal distribution will impact the behavior of the resulting system, and this decision would likely be carefully considered. Less obviously, our decision to parameterize normal distributions using μ and σ will also impact the behavior of the system. If we chose to parameterize normal distributions using the variance, σ^2 rather than the standard deviation, σ, our same learning rule would produce a different sequence of normal distributions. In general, if we represent normal distributions with two parameters, μ and σ^k, the resulting gradient ascent updates are: $\mu_{i+1} = \mu_i + \alpha / (\sigma_i^k)^{\frac{2}{k}} \sum_{j=1}^{n} (X_j - \mu_i)$, and $\sigma_{i+1}^k = \sigma_i^k - \frac{\alpha n}{k \sigma_i^k} + \frac{\alpha}{k} (\sigma_i^k)^{-\frac{k+2}{k}} \sum_{j=1}^{n} (X_j - \mu_i)^2$. Figure 1 shows the results of applying this algorithm to a fixed data set using various k. Notice that different choices of how

[1]University of Massachusetts Amherst [2]Carnegie Mellon University [3]Stanford University. Correspondence to: Philip S. Thomas <pthomas@cs.umass.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

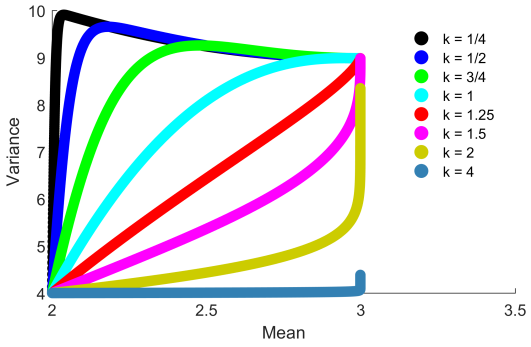

Figure 1: We generated a data set containing 100,000 samples from $\mathcal{N}(3, 9)$—the normal distribution with mean 3 and variance 9. Each curve shows the sequence of normal distributions produced by using gradient descent, $\theta_{i+1} \leftarrow \theta_i - \alpha \nabla L(\theta_i)$ (using abbreviated notation for L), to maximize the log-likelihood of a parameterized normal distribution for 200,000 iterations and starting from $\mathcal{N}(2, 4)$. The normal distributions are parameterized by μ, their mean, and σ^k, a power of their standard deviation. Each curve therefore corresponds to using a different (but equally representative) representation, but the same learning rule.

to represent normal distributions result in wildly different outcomes. A poor choice can result in a sequence of normal distributions that takes a circuitous path to the maximum likelihood distribution and produces poorly scaled updates. As a result, a poor choice of *how* to represent normal distributions can result in the likelihood of the model increasing slowly (notice that using σ^4, the model failed to approach the maximum log-likelihood model.

In this case, and many others, the underlying cause of the intertwining of the learning rule with the choice of how to represent knowledge stems from an implicit assumption hidden within the learning rule (stochastic gradient descent in this case). This implicit assumption is that distances between different parameter vectors, θ, should be measured using Euclidean distance. In Appendix A we review where this implicit assumption is made. In our example, $\theta = [\mu, \sigma^k]^\intercal$, and the distance between $\theta = [\mu, \sigma^k]^\intercal$ and $\theta' = [\mu', \sigma'^k]^\intercal$—the distance between $\mathcal{N}(\mu, \sigma^2)$ and $\mathcal{N}(\mu', \sigma'^2)$—is:

$$\text{dist}(\theta, \theta') = \sqrt{(\mu' - \mu)^2 + (\sigma'^k - \sigma^k)^2}.$$

Notice that this definition of distance is dependent on k, or, more generally, on *how* the parameter vector encodes knowledge. In this case, large k (e.g., $k = 4$) result in small changes to σ incurring large amounts of distance relative to similar changes to μ (given that $\sigma > 1$). As a result, using large k results in a sequence of normal distributions that focuses on changing the mean first, since changes to μ incur less distance than changes to σ. Similarly, small k results in a sequence of normal distributions that over-emphasizes adjusting the variance of the normal distribution.

Sometimes it is easy to control *how* knowledge is represented, like when deciding how to represent normal distributions in our example, and it may also be easy to select a representation such that Euclidean distance in the parameters is reasonable (e.g., using σ or σ^2, but not σ^{20}). However, in other cases it may not be clear how to define θ so that it can represent what we want while simultaneously ensuring that Euclidean distance between parameter vectors is a reasonable notion of distance.

This is particularly true when using deep *artificial neural networks* (ANNs), where the parameter vector, θ, corresponds to the weights of the ANN. If a weight near the output layer has a bigger impact on the function represented by the ANN than a weight in an early layer, then using Euclidean distance between weight vectors θ and θ' to measure the distance between the functions represented by an ANN with weights θ and θ' will place undue emphasis on weights near the output layer. For ANNs, this symptom of the underlying problem is called the problem of vanishing gradients (Hochreiter, 1998).

In this paper we study the intertwining of the decision of which learning rule to use and how knowledge should be represented, extending works by Amari (1985), Kakade (2002), and Bagnell & Schneider (2003). A common misconception is that the topic of this paper is representation selection or learning rule selection—it is neither. This paper is about the coupling of these two problems when using gradient-like learning rules, why it is undesirable, and how these two problems can be partially decoupled.

More specifically, for a broad class of gradient-like learning rules, which subsumes gradient, stochastic gradient, and accelerated gradient methods as well as temporal-difference methods (Sutton, 1988), we present a method for modifying the learning rule to use an explicit definition of how distances should be measured during learning rather than requiring the use of Euclidean distance between parameter vectors. Importantly, this method is *not* a learning rule itself, but rather a method that other researchers can use to enhance the learning rules that they design.

We begin by defining different forms of *covariance*, which capture different levels with which a learning rule can be independent of the choice of how knowledge will be represented. We then propose a method for converting any learning rule from within a broad class of gradient-like learning rules into a first-order covariant learning rule—the weakest form of decoupling—and show how this method can be approximated without increasing the computational complexity of the learning rule. Our method is closely related to natural gradient methods (Amari, 1998)—it extends them to a more general class of learning rules. We also

discuss why a second-order covariant learning rule would be desirable before presenting results that suggest there may not exist any practical second-order covariant learning rules.

We conclude with examples of how some existing learning rules can be converted into first-order covariant learning rules and discuss some of the subtleties and surprising properties of our method (including that using *estimates* of the Fisher information matrix results in a covariant learning rule). For compelling empirical evidence supporting our proposed approach, notice that the recently proposed *zap Q-learning* algorithm (Devraj & Meyn, 2017) and similar *natural temporal difference learning* algorithm (Dabney & Thomas, 2014), both of which are closely related to the *least squares policy evaluation* algorithm (Bertsekas & Ioffe, 1996; Bertsekas, 2009) and *fixed point Kalman filter* algorithm for reinforcement learning (Choi & Van Roy, 2006), can be immediately obtained by applying our method to the well-known *temporal difference learning* learning algorithm (Sutton & Barto, 1998). Thus, instead of serving as baselines with which to compare our approach, the performances of these algorithms show that our framework can produce algorithms that scale to challenging problems.

2. Notation and Definitions

Let a *parameter vector*, θ, be a vector that captures the current knowledge of an ML system. Let $\Theta \subseteq \mathbb{R}^n$ be the set of all possible values for θ. We restrict our discussion to ML systems that use a parameter vector to parameterize a function. Intuitively, a *parameterized function* is a function that, for each possible parameter vector, $\theta \in \Theta$, produces a mapping from elements of some set, $\mathcal{X}$, to elements of $\mathbb{R}^k$. More formally, $f : \mathcal{X} \times \Theta \to \mathbb{R}^k$ so that $f(x, \theta)$ denotes the parameterized function evaluated at $x \in \mathcal{X}$ using the parameter vector $\theta \in \Theta$. For example, in a regression system $f(x, \theta)$ might be an estimate of $f^\star(x)$ for some target function $f^\star : \mathcal{X} \to \mathbb{R}^k$ and all $x \in \mathcal{X}$. Similarly, in a reinforcement learning system $f(x, \theta)$ might be a scalar estimate of the value, $q^\pi(x)$, of a state-action pair, x, under policy, π. Let $\mathcal{P}$ be the set of all parameterized functions.

Importantly, we further restrict our discussion to systems where the parameterized function is in C^1: we only consider parameterized functions, f, such that for all $x \in \mathcal{X}$ and $\theta \in \Theta$, $\frac{\partial f}{\partial \theta}(x, \theta)$ exists. Although this assumption rules out some artificial intelligence and ML systems, like ones that use STRIPS (Fikes & Nilsson, 1971) or ID3 (Quinlan, 1986), it applies to many systems like those that use linear estimators or ANNs.

Learning is the iterative search for parameter vectors that cause $f(\cdot, \theta)$ to achieve some desirable property, starting from some set of initial parameter vectors, $\theta_0 := (\theta_0^1, \theta_0^2, \ldots, \theta_0^\iota) \in \Theta^\iota$, where $\iota \in \mathbb{N}_{\geq 0}$ denotes the number of initial parameter vectors required by the learning rule. Although $\iota = 1$ for most learning rules, like gradient descent, some learning rules (like the accelerated gradient methods discussed later) require multiple initial parameter vectors.

Let (Ω, Σ, p) be a probability space that captures all sources of randomness that occur during the learning process. Intuitively, each outcome, $\omega \in \Omega$, is a seed for the random number generator used by both the ML system and the environment with which it interacts. A *learning rule*, l, is a sequence of functions, $l := (l_i)_{i=1}^\infty$ where each $l_i : \mathcal{P} \times \Theta^\iota \times \Omega \to \Theta$ such that $l_i(f, \theta_0, \omega)$ denotes the parameter vector at the i^{th} iteration when learning rule l is used with the parameterized function f, the initial parameter vectors are $\theta_0 \in \Theta^\iota$, and all randomness is captured by the outcome $\omega \in \Omega$. To simplify notation for the base cases in recursive expressions, when $i \leq 0$ let $l_i(f, \theta_0, \omega) := \theta_0^{1-i}$.

For example, consider the learning rule, l, for stochastic gradient descent when minimizing the expected squared error between $f(\cdot, \theta)$ and a target function, $f^\star : \mathcal{X} \to \mathbb{R}$:

$$
\begin{aligned}
&l_{i+1}(f, \theta_0, \omega) \\
=&l_i(f, \theta_0, \omega) - \alpha_i \Big(f^\star(X_i(\omega)) - f(X_i(\omega), l_i(f, \theta_0, \omega)) \Big) \\
&\times \frac{\partial f}{\partial l_i(f, \theta_0, \omega)}(X_i(\omega), l_i(f, \theta_0, \omega)),
\end{aligned}
$$

where $\times$ denotes scalar multiplication split across multiple lines, $X_i : \Omega \to \mathcal{X}$ is the input to $f^\star$ used during the i^{th} iteration, which is a random variable, and $(\alpha_i)_{i=1}^\infty$ is a sequence of positive real-valued step sizes. To obtain a more familiar notation, let $\theta_i := l_i(f, \theta_0, \omega)$, which gives the definition: $\theta_{i+1} = \theta_i - \alpha_i \big(f^\star(X_i(\omega)) - f(X_i(\omega), \theta_i) \big) \frac{\partial f}{\partial \theta_i}(X_i(\omega), \theta_i)$.

Intuitively, we say that a parameterized function, g, is congruent to a parameterized function, f, if g can represent everything that f can, and there is a smooth mapping from parameters for f to congruent parameters for g. More formally:

Definition 1 (Congruent Representations)**.** *Let $g : \mathcal{X} \times \Psi \to \mathbb{R}^k$ and $f : \mathcal{X} \times \Theta \to \mathbb{R}^k$ be two parameterized functions, where $\Psi \subseteq \mathbb{R}^m$ and $\Theta \subseteq \mathbb{R}^n$. We say that g is congruent to f if there exists a function $\psi : \Theta \to \Psi$, called a submersion, such that $f(x, \theta) = g(x, \psi(\theta))$ for all $x \in \mathcal{X}$ and $\theta \in \Theta$, $\frac{\partial \psi}{\partial \theta}(\theta)$ is full rank for all $\theta \in \Theta$, and $m \leq n$.*

Hereafter we reserve the symbols f, g, ψ, Θ, and Ψ to represent a parameterized function, $g : \mathcal{X} \times \Psi \to \mathbb{R}^k$, that is congruent to a parameterized function, $f : \mathcal{X} \times \Theta \to \mathbb{R}^k$, with submersion ψ, and we reserve the symbols n and m for the dimensions of Θ and Ψ, respectively. Also, notice that the congruency of functions is *not* symmetric. For example, if $g(x, \theta) = \theta$ and $f(x, \theta) = e^\theta$, where $\theta \in \mathbb{R}$, then g is congruent to f, but f is not congruent to g.

We can now state some earlier concepts more formally. The question of *what* an ML system should be able to represent is the question of what functions should be representable by the parameterized function that is used. The question of *how* an ML system should represent knowledge is the question of which parameterized function to use from a set of mutually congruent parameterized functions.

Intuitively, our goal is to ensure that a learning rule will produce the same sequence of functions regardless of which parameterized function is selected from a set of mutually congruent parameterized functions. Learning rules with this property are sometimes called *invariant to reparameterization*, since they do not change their behavior if the parameterized function that they use is parameterized in a different way. They are also sometimes called *covariant*, perhaps in reference to *covariant transformations* in physics. This usage of the word covariant has been attributed (MacKay, 1996) to Knuth (1968). To maintain consistency with prior work in machine learning (MacKay, 1996; Amari, 1998; Amari & Douglas, 1998; Kakade, 2002; Bagnell & Schneider, 2003; Peters & Schaal, 2008; Dabney & Thomas, 2014; Thomas et al., 2016), and for brevity, we adopt the latter terminology. Formally, we define a covariant learning rule as follows, where $\psi(\theta_0) \coloneqq (\psi(\theta_0^1), \psi(\theta_0^2), \ldots, \psi(\theta_0^\iota))$:

Definition 2 (Covariant Learning Rule)**.** *A learning rule, l, is a covariant learning rule if for all parameterized functions, f, all parameterized functions g that are congruent to f, all $i \in \mathbb{N}_{>0}$, all $\omega \in \Omega$, all $\theta_0 \in \Theta^\iota$, and all $x \in \mathcal{X}$, $f(x, l_i(f, \theta_0, \omega)) = g(x, l_i(g, \psi(\theta_0), \omega))$.*

Although covariant learning rules exist, they tend to be computationally impractical and are therefore rarely used. Instead of requiring a learning rule to produce the exact same sequence of functions, $(f(\cdot, l_i(f, \theta_0, \omega)))_{i=1}^\infty$ regardless of the parameterized function that is used, we can require a learning rule to produce a sequence of functions that, at each iteration, changes the parameter vector in a direction that is locally independent of the parameterization. More specifically, we define a j-order covariant learning rule to be a learning rule that ensures that a j-order Taylor approximation of $f(\cdot, l_i(f, \theta_0, \omega))$, centered around $l_{i-1}(f, \theta_0, \omega)$, is independent of f:

Definition 3 (j-Order Covariant Learning Rule With Respect to a Set, $\mathcal{G}$)**.** *A learning rule, l, is a j-order covariant learning rule with respect to a set $\mathcal{G} \subseteq \mathcal{P}$ if for all parameterized functions, $f \in \mathcal{P}$, all $g \in \mathcal{G}$ that are congruent to f, all $i \in \mathbb{N}_{>0}$, all $\omega \in \Omega$, all $\theta_0 \in \Theta^\iota$, and all $x \in \mathcal{X}$,*

$$\begin{aligned}&\tau_j(f(x, \cdot), l_{i-1}(f, \theta_0, \omega), l_i(f, \theta_0, \omega)) \\ &\quad = \tau_j\left(g(x, \cdot), \psi(l_{i-1}(f, \theta_0, \omega)), l_i(g, \psi(\theta_0), \omega)\right),\end{aligned}$$

where $\tau_j(h, y, y')$ denotes the j-order Taylor approximation of $h(y')$ centered around $h(y)$.

Notice that a covariant learning rule is *not* necessarily j-order covariant for any j, nor does j-order covariance imply $(j-k)$-order covariance for any $k > 0$. Also, although j-order covariance captures different levels of covariance, it is limited to learning rules of the form $\theta_i = \theta_{i-1} + \alpha_i \Delta_i$, where $\Delta_i \in \mathbb{R}^n$ is the update direction at step i. That is: it is restricted to updates that take a step from the previous parameters, θ_{i-1}. Some learning rules, like accelerated gradient methods (Nesterov, 1983), take steps from some other point: $\theta_i = \beta_i + \alpha_i \Delta_i$, where β_i might be some combination of previous parameter vectors, like $\beta_i = \gamma\theta_{i-1} + (1-\gamma)\theta_{i-2}$, where $\gamma \in [0, 1]$. To provide an achievable form of covariance for these updates, we require them to be equivalent under a j-order Taylor approximation centered around β_i. This leads to the more general definition of j-order covariance:

Definition 4 (j-Order Covariant Learning Rule with Respect to a Sequence and Set)**.** *Let $\beta_1, \beta_2, \ldots$ be a sequence of random variables where each $\beta_i : \Omega \to \mathbb{R}^n$. A learning rule, l, is a j-order covariant learning rule with respect to the sequence $(\beta_i(\omega))_{i=1}^\infty$ and a set, $\mathcal{G} \subseteq \mathcal{P}$, if for all parameterized functions, $f \in \mathcal{P}$, all $g \in \mathcal{G}$ that are congruent to f, all $i \in \mathbb{N}_{>0}$, all $\omega \in \Omega$, all $\theta_0 \in \Theta^\iota$, and all $x \in \mathcal{X}$,*

$$\begin{aligned}&\tau_j(f(x, \cdot), \beta_i(\omega), l_i(f, \theta_0, \omega)) \\ &\quad = \tau_j\left(g(x, \cdot), \psi(\beta_i(\omega)), l_i(g, \psi(\theta_0), \omega)\right).\end{aligned}$$

This is *not* consistent with previous definitions of a covariant learning rule (Bagnell & Schneider, 2003; Peters & Schaal, 2008; Dabney & Thomas, 2014; Thomas et al., 2016)—what these previous authors called "covariant" learning rules we call "first-order covariant". We adopt our alternate definition because later we discuss higher orders of covariance. Also, hereafter, for brevity, we write β_i as shorthand for $\beta_i(\omega)$.

Furthermore, previous works refer to covariant updates without specifying the set, $\mathcal{G}$, that the covariance is with respect to—we make this set explicit. As an example, Thomas et al. (2016) provide a proof of first-order covariance for a specific learning rule that requires the implicit assumption that $\mathcal{G}$ only includes g with positive definite *energetic information matrices*. Similarly, Amari (1985, page 16) restricts f and g to parameterized discrete probability distributions for which the *Fisher information matrix* is positive definite (it is in general only guaranteed to be positive semi-definite). We make these restrictions on g explicit in the definition of j-order covariant learning rules. Notice that the strength of the j-order covariance property scales not just with j, but also with the size of $\mathcal{G}$.

Lastly, let $\mathcal{F}$ be a σ-algebra on $\mathcal{X}$ so that $(\mathcal{X}, \mathcal{F})$ is a measurable space. Also, recall that $\mathbb{R}^\mathcal{X}$ denotes the set of all functions from $\mathcal{X}$ to $\mathbb{R}$. Let $\mathcal{F} \otimes \mathcal{F}$ denote the tensor-product σ-algebra on the product space, $\mathcal{X} \times \mathcal{X}$.

3. How to Make a Learning Rule First-Order Covariant

In Theorem 1, presented later in this section, we show how a broad class of gradient-like learning rules can be transformed into first-order covariant learning rules. Intuitively, the $\partial f(x,\theta)/\partial\theta$ terms within a learning rule make the implicit assumption that distances between parameterized functions, $f(\cdot,\theta)$ and $f(\cdot,\theta')$, should be measured using Euclidean distance between their parameter vectors. This implicit assumption has been described previously (Amari, 1998). In Appendix A we review exactly where this implicit assumption is made. We propose replacing the $\partial f(x,\theta)/\partial\theta$ terms within a learning rule with terms that respect a user-defined distance. Furthermore, we allow different distance measures over functions to be used when replacing each $\partial f(x,\theta)/\partial\theta$ term in a learning rule.

Theorem 1 is related to *natural gradient* methods (Amari, 1998). Amari (1998) argued that, if **1)** the goal is to minimize a loss function, $L : \mathbb{R}^n \to \mathbb{R}$, **2)** the arguments (inputs) of L lie on a Riemannian manifold characterized by the positive-definite metric tensor $G : \Theta \to \mathbb{R}^{n\times n}$, and **3)** one plans to use a gradient descent algorithm of the form $\theta_{i+1} \leftarrow \theta_i - \alpha_i \nabla L(\theta_i)$, then one should instead use the *natural gradient* update $\theta_{i+1} \leftarrow \theta_i - \alpha G(\theta_i)^{-1}\nabla L(\theta_i)$.[1] Theorem 1 does not require any of these three assumptions, although it also holds given these assumptions. Instead of these assumptions, our result relies on an assumption that the learning rule is gradient-like (which is formally defined later), which allows for updates like the temporal-difference learning update that is not the gradient of a function (Sutton, 1988), the use of stochastic gradient estimates, and accelerated gradient methods (Nesterov, 1983) that are more sophisticated than ordinary gradient methods.

Thomas (2014b) generalized the natural gradient to allow the parameters of L to lie on a semi-Riemannian manifold and established sufficient conditions for the convergence of natural gradient descent (which differ from those of ordinary gradient descent), but did not establish covariance properties. Intuitively, one can view our approach as replacing gradient terms, $\partial f(x,\theta)/\partial\theta$, within a gradient-like learning rule, with (generalized) natural gradients, $G(\theta)^+\partial f(x,\theta)/\partial\theta$, where $G(\theta)$ is automatically derived from a user-provided notion of the distance between parameterized functions. Unlike Amari's original natural gradient method, the gradient terms that we replace with natural gradient terms are *not* the gradients of a loss function, but rather terms within a learning rule. In Appendix A we show how the $G(\theta)$ that we use can be derived from the provided distance measure and review Amari's argument for why using $G(\theta)^+\partial f(\cdot,\theta)/\partial\theta$ terms in place of $\partial f(x,\theta)/\partial\theta$ terms corresponds to using the provided distance function rather than Euclidean distance (where $G(\theta)^+$ denotes the Moore-Penrose pseudoinverse of $G(\theta)$). Due to the strong influence natural gradients had on our approach, we refer to the learning rule, $\tilde{l}$, produced by applying our method to a learning rule l, as a *naturalized* version of l. Also, the naturalized version of gradient descent will be Amari's natural gradient algorithm.

The set of gradient-like learning rules that we consider includes learning rules that start from the previous weights, θ_{i-1}, and then move in a direction $\delta\frac{\partial f}{\partial\theta_i}(x,\theta_{i-1})$ for some x, where $\delta \in \mathbb{R}$ is a scalar error term that dictates whether $f(x,\theta_{i-1})$ should be increased or decreased. We also include learning rules that consider how to change $f(x,\theta_{i-1})$ for more than one x (e.g., when using mini-batches), by allowing learning rules that move in a direction $\sum_j \delta_j \frac{\partial f}{\partial\theta_{i-1}}(x_j,\theta_{i-1})$, where δ_j is the error term associated with x_j. We further generalize this by **1)** integrating over the x_j that are included, where the integral is with respect to a signed measure that captures the error term, **2)** allowing for the partial derivatives of f to be taken at β_i, **3)** allowing the update to start from a point that is more general than θ_{i-1}, and **4)** by allowing for stochastic gradient updates, where the x_j that are updated, and their associated errors, depend on ω. Formally we define this set of gradient-like learning rules as those that satisfy the following assumption:

Assumption 1. *The learning rule, l, can be written as:*

$$l_{i+1}(f,\theta_0,\omega) = l'_i(f,\theta_0,\omega) + \int_{\mathcal{X}} \frac{\partial f}{\partial\beta_i}(\cdot,\beta_i)\, d\mu_i(f(\cdot,\beta_i),\omega,\cdot),$$

where l' is a first-order covariant learning rule with respect to a set $\mathcal{G}$ and sequence $(\beta_i)_{i=1}^\infty$ and for all $i \in \mathbb{N}_{>0}$, $\mu_i : \mathbb{R}^{\mathcal{X}} \times \Omega \times X \to \mathbb{R}$ is a signed measure on $(\mathcal{X},\mathcal{F})$.

Notice that $l'_i(f,\theta_0,\omega) := \beta_i$ is a first order covariant learning rule with respect to $(\beta_i)_{i=1}^\infty$, and that this definition of l' will be the most common.

As an example, consider the *temporal difference* learning rule (Sutton, 1988) for making f approximate a function called a *state-value function*, and which has the form: $\theta_{t+1} \leftarrow \theta_t + \alpha(R_t + \gamma f(S_{t+1},\theta_t) - f(S_t,\theta_t))\frac{\partial f}{\partial\theta_i}(S_t,\theta_i)$, where S_t and R_t are random variables called the *state* and *reward* at time t, and α is a small positive step size (Sutton & Barto, 1998). This learning rule satisfies Assumption 1, where **1)** $t \leftarrow i$, **2)** $l_{t+1}(f,\theta_0,\omega) \leftarrow \theta_{t+1}$, **3)** $l'_t(f,\theta_0,\omega) \leftarrow \theta_t$, **4)** each $x \in \mathcal{X}$ is a state-reward-state tuple, i.e., $x = (s,a,s')$, **5)** $f((s,a,s'),\theta)$ does not depend on a and s', and so we write $f(s,\theta)$, and **6)** the signed measure $\mu_t(f(\cdot,\theta_t),\omega,\cdot)$ places a mass of $\alpha_t(R_t + \gamma f(S_{t+1},\theta_t) - f(S_t,\theta_t))$ on $x = (S_t,R_t,S_{t+1})$, where S_t and R_t are shorthand for $S_t(\omega)$ and $R_t(\omega)$.

As mentioned earlier, our method allows the designer of an

[1] For discussion of the relationship between natural gradients, Newton's method, approximate second-order methods, and other optimization methods, see, for example, the works of Pascanu & Bengio (2013); Martens (2014); Furmston et al. (2016).

algorithm to select an implicit distance over parameter space that is used by the resulting naturalized learning rule. More precisely, this "distance" may not satisfy the requirements of a measure, and so technically it is a dissimilarity function, not a distance measure. To define a notion of distance, the designer of a learning rule should select a sequence of functions, $p_1, p_2, \ldots$, such that each $p_i : \mathbb{R}^{\mathcal{X}} \times \Omega \times \mathcal{X} \times (\mathcal{F} \otimes \mathcal{F}) \to \mathbb{R}$ is a set of signed measures on $(\mathcal{X} \times \mathcal{X}, \mathcal{F} \otimes \mathcal{F})$. During the i^{th} update, the gradient term $\frac{\partial f}{\partial \beta_i}(\cdot, \beta_i)$ will be replaced with a generalized natural gradient term that measures the dissimilarity between θ and $\theta + \Delta$ as:

$$\sqrt{\frac{1}{2}\int_{\mathcal{X}^2}(f(x,\theta) - f(x,\theta+\Delta))(f(y,\theta) - f(y,\theta+\Delta)p_i(f(\cdot,\beta_i),\omega,dx,dy)}. \tag{1}$$

Thus, the choice of p_i should capture the desired notion of distance. Often (but not always) $p_i(f(\cdot,\beta_i),\omega,\cdot,\cdot)$ will be a joint *probability* measure over random variables that are always equal, and will also be a probability mass function (or probability density function), in which case (writing $p_i(x)$ for $p_i(f(\cdot,\beta_i),\omega,x,x)$) we can write the notion of distance as:

$$\begin{aligned}\text{dist}(\theta, \theta+\Delta) &= \sqrt{\frac{1}{2}\sum_{x\in\mathcal{X}} p_i(x)(f(x,\theta) - f(x,\theta+\Delta))^2} \\ &= \sqrt{\frac{1}{2}\mathbf{E}\left[(f(X,\theta) - f(X,\theta+\Delta))^2\right]},\end{aligned}$$

where $X \sim p_i$. This makes it clear that our notion of distance between $f(\cdot,\theta)$ and $f(\cdot,\theta+\Delta)$ is related to the expected squared difference between their values given that the inputs, x, are sampled from some distribution, $p_i(\cdot)$, which is chosen by the designer of the algorithm.

Consider again the temporal difference learning example from before, where $x = (s, r, s')$. In this case, we might define p_t to encode the distribution over states, rewards, and next-states that occur under the policy being evaluated by the temporal difference algorithm. This would cause $\text{dist}(\theta, \theta+\Delta)$ to be the square root of half the expected squared difference between state-value estimates when using θ and $\theta+\Delta$, and where the expected value assumes that states and rewards come from the *on-policy distribution over states and rewards* (Sutton & Barto, 2018).

We are now ready to present our main theorem, which takes a learning rule, l, and transforms it into a new learning rule, $\tilde{l}$, that is first-order covariant.

Theorem 1. *If a learning rule, l, satisfies Assumption 1, then for any sequence of p_i, the learning rule $\tilde{l}$ defined by:*

$$\tilde{l}_i(f,\theta_0,\omega) = l'_i(f,\theta_0,\omega) + \int_{\mathcal{X}} G_i^{\cdot}(f,\beta_i)^{+}\frac{\partial f}{\partial \beta_i}(\cdot,\beta_i)\, d\mu_i(f(\cdot,\beta_i),\omega,\cdot),$$

$$G_i^z(f,\beta_i) \coloneqq \int_{\mathcal{X}^2}\frac{\partial f}{\partial \beta_i}(x,\beta_i)\left(\frac{\partial f}{\partial \beta_i}(y,\beta_i)\right)^{\intercal} p_i(f(\cdot,\beta_i),\omega,z,dx,dy), \tag{2}$$

is a first-order covariant learning rule with respect to $(\beta_i)_{i=1}^{\infty}$ and $\mathcal{G}$, where $\mathcal{G}$ is the set of parameterized functions, $g \subseteq \mathcal{P}$ such that $G_i^z(g, \psi(\beta_i))$ is full rank for all $z \in \text{supp}(\mu_i(f(\cdot,\beta_i),\omega,\cdot))$.

Proof. *See Appendix B.* □

Theorem 1 produces the following "naturalized" form for our temporal difference learning example:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t G_t^{+}(R_t + \gamma f(S_{t+1},\theta_t) - f(S_t,\theta_t))\frac{\partial f}{\partial \theta_i}(S_t,\theta_i),$$

where $G_t = \mathbf{E}\left[\frac{\partial f}{\partial \theta_t}(S,\theta_t)\left(\frac{\partial f}{\partial \theta_t}(S,\theta_t)\right)^{\intercal}\right]$, where S is a state sampled from the distribution over states that occurs when running the policy being evaluated (averaged over all time steps—not for the time step t specifically). This algorithm is similar to the natural temporal difference learning algorithm (Dabney & Thomas, 2014), which can be derived by defining p_t to include a temporal difference error term.

4. Direct Estimation of the Update

In cases where $G_i^x(f,\beta_i)$ is not sparse, $\tilde{l}$ can have high computational complexity—$O(n^3)$ for naïve implementations. In this section we show that in some cases $\tilde{l}$ can be estimated directly without even requiring the estimation of the $n \times n$ matrix $G_i^x(f,\beta_i)$. Whereas Theorem 1 was inspired by Amari's work with natural gradients, the linear time approximation presented here generalizes the works of Kakade (2002) and Bhatnagar et al. (2009), which show that the natural policy gradient in reinforcement learning can be estimated in linear time by using *compatible function approximation* (Sutton et al., 2000).

In this section we assume that $k = 1$ so that $f(x,\theta) \in \mathbb{R}$. Let $\hat{1} : \mathcal{X} \times \mathbb{R}^n$ be a parameterized function defined by $\hat{1}(x,w) \coloneqq w^{\intercal}\frac{\partial f}{\partial \beta_i}(x,\beta_i)$.

Theorem 2. *If*

$$w^{\star} \in \arg\min_{w\in\mathbb{R}^n}\int_{\mathcal{X}}\left(1 - \hat{1}(\cdot,w)\right)^2 d\mu_i(f(\cdot,\beta_i),\omega,\cdot),$$

either $G_i^x(f,\beta_i) = \int_{\mathcal{X}}\frac{\partial f}{\partial \beta_i}(\cdot,\beta_i)\frac{\partial f}{\partial \beta_i}(\cdot,\beta_i)^{\intercal}\, d\mu_i(f(\cdot,\beta_i),\omega,\cdot)$, or $G_i^x(f,\beta_i) = \frac{\partial f}{\partial \beta_i}(x,\beta_i)\frac{\partial f}{\partial \beta_i}(x,\beta_i)^{\intercal}$, and $G_i^x(f,\beta_i)$ is full rank for all β_i and the single f that is being used, then the learning rule, $\tilde{l}$, in Theorem 1 can be written as $\tilde{l}_i(f,\theta_0,\omega) = l'_i(f,\theta_0,\omega) + w^{\star}$.

Proof. *See Appendix C.* □

Intuitively, Theorem 2 says that if $w^{\star} \in \mathbb{R}$ are parameters for $\hat{1}$ that minimize the average difference between $\hat{1}(x,w)$ and 1, weighted by the signed measure μ_i, then $\tilde{l}$ takes a step in the direction $w^{\star}$ from β_i. If $w^{\star}$ can be efficiently estimated, then it may be more computationally efficient to estimate $w^{\star}$ than it is to estimate G_i^x and compute the product of its pseudoinverse with $\partial f(x,\beta_i)/\partial \beta_i$.

For example, in the context of natural policy gradient methods for reinforcement learning, Bhatnagar et al. (2009)

suggest using a two-timescale approach to simultaneously estimate $w^\star$ using stochastic gradient descent on $\int_{\mathcal{X}} \left(1 - \hat{1}(\cdot, w)\right)^2 d\mu_i(f(\cdot, \beta_i), \omega, \cdot)$ and update the parameters θ in the direction $w^\star$. This approach results in computational complexity $O(n)$ per time step, since the stochastic gradient update for estimating $w^\star$ takes $O(n)$ time. Dabney & Thomas (2014) also show how a variant of temporal difference learning, called *residual gradient* (Baird, 1995), has a naturalized form that allows for this linear-time approximation. However, notice that methods of this form only *approximate* a first-order covariant learning rule, because an estimate of $w^\star$ is used instead of $w^\star$.

5. The Non-Existence of Second-Order Covariant Learning Rules

While investigating the proper use of Amari's natural gradient methods for policy search in reinforcement learning, Bagnell & Schneider (2003), noticed that a natural gradient method (using the Fisher information matrix for $G(\theta)$, which results in a first-order covariant update with respect to the set of parameterized probability distributions that have positive definite Fisher information matrices) did not act in a covariant way in practice. They concluded that this was due to their use of large step sizes, since first-order covariant learning rules will only behave in a covariant manner for step sizes that are sufficiently small for the first-order Taylor approximation to be accurate. This raises the question: can one develop a second-order covariant learning rule? If such a learning rule existed, then it might behave in a covariant manner when using larger step sizes.

Although we set out to construct a second-order covariant learning rule, we were unable to find any for non-degenerate $\mathcal{G}$, other than the trivial learning rule $l_i(f, \theta_0, \omega) \coloneqq \beta_i$. In Theorem 3 we give a one dimensional (i.e., $n = m = k = 1$) example of a reasonable class of $\mathcal{G}$ for which no second-order covariant learning rules exist. We conjecture that no second order learning rules exist for a far broader class of similar $\mathcal{G}$. We say that two functions, ρ and ϱ, both with domain $\mathcal{X}$, are *collinear* if there exists a constant $\gamma \in \mathbb{R}$ such that for all $x \in \mathcal{X}$, $\rho(x) = \gamma\varrho(x)$.

Theorem 3 (Nonexistence of Nontrivial Second-Order Covariant Learning Rules)**.** *Every learning rule, l, that is second-order covariant with respect to any sequence, $(\beta_i)_{i=1}^\infty$, and a set, $\mathcal{G}$, must use the trivial update, $l_i(f, \theta_0, \omega) \coloneqq \beta_i$ for all parameterized functions, f, where* **1)** *$n = k = 1$,* **2)** *both $g(x, \theta) \coloneqq f(x, \ln(\theta))$ and $h(x, \theta) \coloneqq f(x, \ln(\theta)/2)$ are in $\mathcal{G}$ and are congruent to f and* **3)** *both $\frac{\partial g}{\partial \theta}(\cdot, \beta_i)$ and $\frac{\partial^2 g}{\partial \theta^2}(\cdot, \beta_i)$ are not collinear and $\frac{\partial h}{\partial \theta}(\cdot, \beta_i)$ and $\frac{\partial^2 h}{\partial \theta^2}(\cdot, \beta_i)$ are not collinear.*

Proof. *See Appendix D.* □

6. Discussion and Conclusion

First, notice that Theorem 1 generalizes several existing results. If $\mathcal{P}$ contains parameterized log probability distributions, then p can be chosen to make $G_i^{\cdot}(f, \beta_i)$ be the Fisher information matrix or energetic information matrix of $f(\cdot, \beta_i)$. In these cases, the set, $\mathcal{G}$, that the naturalized algorithms are covariant with respect to includes all parameterized probability distributions with positive definite Fisher information matrices and energetic information matrices.

Furthermore, Theorem 1 allows for the naturalization of a broad class of learning rules. For example, accelerated gradient methods (Nesterov, 1983) use updates of the form:

$$\beta_i = l_{i-1}(f, \theta_0, \omega) + \frac{i-1}{i+1}\left(l_{i-1}(f, \theta_0, \omega) - l_{i-2}(f, \theta_0, \omega)\right),$$

$$l_i = \beta_i - \alpha_{i-1} \int_{\mathcal{X}} \frac{\partial f}{\partial \beta_i}(\cdot, \beta_i)\, d\mu_i(f(\cdot, \beta_i), \omega, \cdot),$$

which can be transformed into a first-order covariant learning rule with respect to $(\beta_i)_{i=1}^\infty$ and $\mathcal{G}$ using Theorem 1. The resulting naturalized accelerated gradient update is: $l_i = \beta_i - \alpha_{i-1} \int_{\mathcal{X}} G_i^{\cdot}(f, \beta_i)^+ \frac{\partial f}{\partial \beta_i}(\cdot, \beta_i)\, d\mu_i(f(\cdot, \beta_i), \omega, \cdot)$.

As shown throughout this paper, temporal difference algorithms also can be transformed into first order covariant learning rules. Dabney & Thomas (2014) presented such a naturalized temporal difference algorithm and showed that it produces state of the art performance on several classical benchmark problems. Devraj & Meyn (2017) showed that a variant of this algorithm, which they call zap Q-learning, produces state of the art performance on modern deep reinforcement learning benchmark problems. As yet another example of the strong performance of naturalized algorithms, since most discounted episodic policy gradient algorithms have been shown to *not* be gradient (or stochastic gradient) algorithms (Thomas, 2014a), natural policy gradient methods (Peters & Schaal, 2008) are also examples of the application of Theorem 1 to non-gradient learning rules.

Notice that the user of Theorem 1 is free to select what constitutes f in an algorithm. For example, one might select f to be the *probability density function* (PDF) for a normal distribution in the example from the introduction, or one might select f to be the natural logarithm of the PDF for a normal distribution (this latter choice can make G_i^x the Fisher information matrix).

Perhaps one of the most important properties of Theorem 1 is that $p_i(f, \theta_0, \omega, z, E, E')$ can have support only over some fixed small number of samples, $s > n$. This means that one can, for example, use a data-based estimate of the Fisher information matrix (constructed from s samples), and the resulting update will be first-order covariant. The catch here is that $\mathcal{G}$ will be small for small s (e.g., if $s < n$, then $\mathcal{G}$ will likely be empty). However, as s grows, $\mathcal{G}$ will

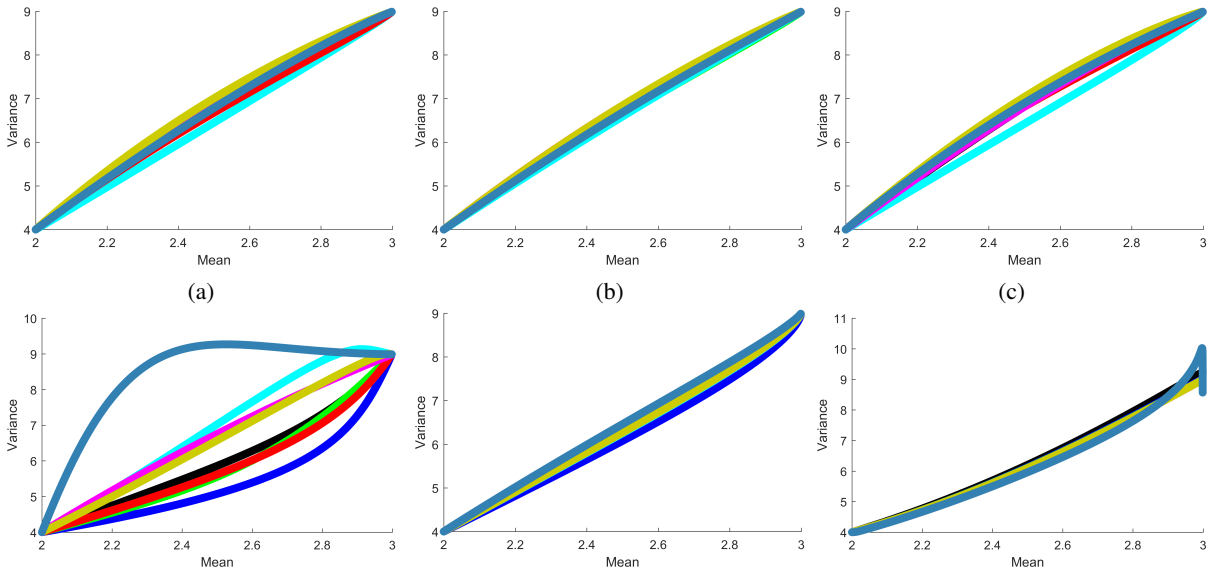

Figure 2: Reproduction of Figure 1 using naturalized gradient descent algorithms and with the legend suppressed. Each plot uses a fixed step size for all k, but step sizes vary between plots. (a)-(e) use the update $\theta_{i+1} = \theta_i - \alpha G(\theta_i)^{-1} \nabla L(\theta_i)$, each with different $G(\theta_i)$, but the same (precisely estimated) $\nabla L(\theta_i)$. **(a)** Where $f(x, \theta)$ is the log-probability of x and $p_i(f, \theta_0, \omega, x, x) = f(x, \theta_i)$ so that $G(\theta)$ is the Fisher information matrix of $f(, \cdot, \beta_i)$. The Fisher information matrix was estimated from 1,000 samples of x. Thus, $G(\theta) = (1/1000) \sum_{j=1}^{1000} (\partial \ln(\Pr(X_j|\theta))/\partial\theta)(\partial \ln(\Pr(X_j|\theta))/\partial\theta)^\intercal$, where here $\Pr$ denotes probability *density*. This shows that ordinary natural gradient descent using the Fisher information matrix exhibits covariant behavior for this problem. **(b)** The same as (a), except where $f(x, \theta)$ is the probability of x, and so $G(\theta) = (1/1000) \sum_{j=1}^{1000} (\partial \Pr(X_j|\theta)/\partial\theta)(\partial \Pr(X_j|\theta)/\partial\theta)^\intercal$. This shows that covariant behavior persists when changing the Fisher information matrix to be different (not include the log term). **(c)** the same as (b), but using only 100 samples to estimate $G(\theta)$ (the sum over j stops at 100). This shows that using a crude empirical estimate of the Fisher information matrix (without the log terms) maintains covariant behavior. **(d)** Same as (b), but using just 5 samples to estimate $G(\theta)$ (the sum over j stops at 5). This shows that using extremely low quality estimates of the Fisher information matrix (without the log terms) still shifts the algorithm's behavior towards covariant behavior (it is first-order covariant, but with respect to a set, $\mathcal{G}$, that may be small or empty). **(e)** The same as (b), but where the x used to estimate $G(\theta)$ were sampled from a continuous uniform distribution, p_i, rather than from $f(\cdot, \beta_i)$. This shows that estimating the Fisher information matrix using samples from a distribution other than the one being parameterized still results in covariant behavior, as suggested by our theoretical results. **(f)** The same setup as (a), but using the linear-time direct estimation technique from Theorem 2.

include more and more parameterizations of probability distributions until, in the limit as $s \to \infty$, $\mathcal{G}$ is the set of parameterized probability distributions whose Fisher information matrix is positive definite (the same $\mathcal{G}$ used implicitly in conventional covariance proofs (Amari, 1985)).

Importantly, notice that $p_i(f, \theta_0, \omega, z, E, E')$ can be unrelated to $\mu_i(f, \theta_0, \omega, E)$. This means, for example, that one can estimate the Fisher information matrix associated with one parameterized probability distribution using samples from a different distribution, and the resulting learning rule will be first-order covariant. This can be useful for applications where sampling from $f(\cdot, \theta)$ is expensive—for example in policy gradient applications where sampling states from the stationary distribution under a parameterized stochastic policy is expensive relative to sampling states from a uniform distribution. We empirically validated these various properties by applying various naturalized algorithms to the illustrative example from the introduction. The results, which support the theoretical discussion, are presented in Figure 2.

In summary, we have presented a method for converting a broad class of gradient-like learning rules into first-order covariant learning rules. This method, which we refer to as the *naturalization* of a learning rule, extends work on natural gradient methods beyond gradient descent, and ensures covariance for metric tensors, G_i^x, that generalize the Fisher information matrix and energetic information matrix, without sacrificing covariance. We also showed how the updates produced by naturalized learning rules can be directly estimated, in some cases in linear time. Finally, we presented initial findings that suggest that there may not exist any practical second-order covariant learning rules.

References

Amari, S. *Differential-geometrical methods in statistics*, volume 28. Springer Science & Business Media, 1985.

Amari, S. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

Amari, S. and Douglas, S. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pp. 1213–1216, 1998.

Bagnell, J. A. and Schneider, J. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1019–1024, 2003.

Baird, L. Residual algorithms: reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

Bertsekas, D. P. Projected equations, variational inequalities, and temporal difference methods. *Lab. for Information and Decision Systems Report LIDS-P-2808, MIT*, 2009.

Bertsekas, D. P. and Ioffe, S. Temporal differences-based policy iteration and applications in neuro-dynamic programming. *Lab. for Info. and Decision Systems Report LIDS-P-2349, MIT, Cambridge, MA*, 1996.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. Natural actor-critic algorithms. *Automatica*, 45(11): 2471–2482, 2009.

Choi, D. and Van Roy, B. A generalized kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006.

Dabney, W. and Thomas, P. S. Natural temporal difference learning. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Devraj, A. M. and Meyn, S. Zap q-learning. In *Advances in Neural Information Processing Systems*, pp. 2232–2241, 2017.

Fikes, R. E. and Nilsson, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Furmston, T., Lever, G., and Barber, D. Approximate Newton methods for policy search in Markov decision processes. *Journal of Machine Learning Research*, 17(227): 1–51, 2016.

Greville, T. and Nall, E. Note on the generalized inverse of a matrix product. *Siam Review*, 8(4):518–521, 1966.

Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Kakade, S. A natural policy gradient. In *Advances in Neural Information Processing Systems*, volume 14, pp. 1531–1538, 2002.

Knuth, D. The art of computer programming 1: Fundamental algorithms 2: Seminumerical algorithms 3: Sorting and searching. *MA: Addison-Wesley*, pp. 30, 1968.

MacKay, D. J. Maximum likelihood and covariant algorithms for independent component analysis. Technical report, Citeseer, 1996.

Martens, J. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

Nesterov, Y. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pp. 372–376, 1983.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Peters, J. and Schaal, S. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

Quinlan, J. R. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Sutton, R. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. Second edition, 2018. Unpublished draft.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pp. 1057–1063, 2000.

Thomas, P. S. Bias in natural actor-critic algorithms. In *Proceedings of the Thirty-First International Conference on Machine Learning*, 2014a.

Thomas, P. S. GeNGA: A generalization of natural gradient ascent with positive and negative convergence results. In *Proceedings of the Thirty-First International Conference on Machine Learning*, 2014b.

Thomas, P. S., da Silva, B. C., Dann, C., and Brunskill, E. Energetic natural gradient descent. In *International Conference on Machine Learning*, pp. 2887–2895, 2016.