Second-Order Sensitivity Methods for Robustly Training Recurrent Neural Network Models

Liam Johnston and Vivak Patel

***Abstract*—Adjoint methods are used in both control theory and machine learning (ML) to efficiently compute gradients of functionals. In ML, the adjoint method is a popular approach for training multilayer neural networks and is commonly referred to as backpropagation. Despite its importance in ML, the adjoint method suffers from two well documented shortcomings: (i) gradient decay/explosion and (ii) excessive training time. Until now, the gradient decay problem has primarily been addressed through modification to the network architecture with gating units that add additional parameters. This results in additional computational costs during evaluation and training which further exacerbates the excessive training time. In this letter, we introduce a powerful framework for addressing the gradient decay problem based on second-order sensitivity concepts from control theory. As a result, we are able to robustly train arbitrary network architectures without suffering from gradient decay. Furthermore, we demonstrate that this method is able to speed up training with respect to both wall-clock time and data efficiency. We demonstrate our method on a synthetic long time gap experiment, as well as three sequential modeling benchmarks with a simple recurrent neural network architecture.**

***Index Terms*—Backpropagation, machine learning, optimal control, recurrent neural networks.**

I. Introduction

MINIMIZING a functional to find an optimal parameter estimate of a discrete time-varying system is a central problem in control theory [1]. Typically, minimizing the functional requires efficiently and accurately computing its gradient [2]. The gradient can be efficiently computed using Largrangian formalism, which results in the adjoint method [2]. The adjoint method was first introduced in the 1970s as a way to perform identification of functional parameters for partial differential equations (PDE) [3], and was extended to a second-order method for efficiently computing matrix-free Hessian-vector products [4], [5].

Manuscript received March 7, 2020; revised May 25, 2020; accepted June 1, 2020. Date of publication June 10, 2020; date of current version July 7, 2020. The work of Vivak Patel was supported by UW-Madison WARF under Award AAD5914. Recommended by Senior Editor C. Seatzu. *(Corresponding author: Liam Johnston.)*

Liam Johnston is with the Department of Statistics, University of Wisconsin–Madison, Madison, WI, USA (e-mail: ljohnston2@wisc.edu).

Vivak Patel is with the Faculty of Statistics, University of Wisconsin–Madison, Madison, WI, USA (e-mail: vivak.patel@wisc.edu).

Digital Object Identifier 10.1109/LCSYS.2020.3001498

In machine learning (ML), the adjoint method is used to train multilayer neural networks under the name *back propagation* (BP) [6]. In fact, since its introduction, the adjoint method has become a central algorithm in computing the gradients required to find optimal parameters of Deep Neural Networks (DNNs) [7]. Despite its importance in ML, the adjoint method suffers from two shortcomings when training DNNs: (i) gradient decay/explosion [8], [9] and (ii) excessive training time [10].

Before now, the gradient decay problem has primarily been addressed by modifying the original network architecture with gating units [10], [11]. Unfortunately, these gating units add a substantial number of parameters to the system and, consequently, incur additional computation costs during evaluation and training [12], which further exacerbates the excessive training time.

In this letter, we introduce a powerful framework for addressing the gradient decay problem based on second-order sensitivity concepts from control theory. In particular, we introduce a penalty term on the decay of the adjoints in the training objective, and use the adjoint-over-adjoint method to efficiently compute (i.e., matrix-free) the gradient of the resulting objective function. As a result, we are able to overcome the gradient decay problem, which allows us to (a) train arbitrary network architectures without introducing the aforementioned complex gating units, and (b) speed up training. We demonstrate the effectiveness of our method using a simple recurrent neural network (RNN) architecture on a synthetic experiment and three common benchmarks used in sequence modeling.

II. Background

A. The Adjoint Method in Learning

In deep learning, the goal is to identify the parameters of a T-horizon discrete-time system such that a given cost function is minimized [13]. To be specific, simplifying to the case of a single observation, the goal is to solve

$$\begin{aligned} \min_{\theta_0,\ldots,\theta_T} \quad & F(y,\hat{y}) \\ \text{s.t.} \quad & \hat{y} = h_{\text{out}}(x_T,\theta_T) \\ & x_{i+1} = h_i(x_i,\theta_i),\ \ i = 0,1,\ldots,T-1 \end{aligned} \tag{1}$$

where (y, x_0) is a given observation; F is the cost function; $\{x_i \in \mathbb{R}^{m_i} : i = 1,\ldots,T\}$ are the states; $\hat{y}$ is referred to as the prediction; $\{\theta_i \in \mathbb{R}^{n_i} : i = 0,\ldots,T\}$ are the parameters;

$\{h_i : \mathbb{R}^{m_i} \times \mathbb{R}^{n_i} \to \mathbb{R}^{m_{i+1}} : i = 0, \ldots, T-1\}$ are the system dynamics that are engineered to address the learning task at hand; and, similarly, h_{out} is a mapping from the final state, x_T, to the prediction, $\hat{y}$. While this problem is analogous to a discrete two point boundary problem commonly seen in the analysis of PDEs [1], [4], [5], there are several important differences in the learning case: (a) the dimension of states are usually identical over time (i.e., $m_i = m_j$) in the case of differential equation problems, whereas these dimensions are likely to vary in the learning case; and (b), in the learning case, the cost function is an average over individual cost functions for each observation, of which there are usually a substantial number.

Typically, solving (1) involves computing the partial derivative of the cost with respect to each of the parameters. For example, to compute the gradient with respect to θ_i, we use the chain rule to derive

$$\frac{\partial F}{\partial \theta_i} = (\frac{\partial x_i}{\partial \theta_i})^T (\frac{\partial x_{i+1}}{\partial x_i})^T \cdots (\frac{\partial \hat{y}}{\partial x_T})^T \frac{\partial F(y, \hat{y})}{\partial \hat{y}}, \tag{2}$$

where, as an abuse of notation, T refers to the time horizon or the transpose of a matrix. Unfortunately, as seen in (2), computing the gradient requires the computation and product of a sequence of Jacobian matrices, which can become prohibitively expensive as the dimension of the states or parameters grow.

To avoid this, the adjoint method can be used. The adjoint method efficiently addresses this expense by proceeding by duality, formulating a so-called adjoint system or "backward" discrete-time system, and leveraging the solution vectors—called the adjoint states or adjoint variables—of the adjoint system to compute derivatives using only matrix-vector products [2].

The adjoint system can be derived through the Lagrangian framework. The Lagrangian framework considers the state dynamics as additional constraints under study and introduces a set of adjoint variables (Lagrange multipliers), $\lambda_1, \ldots, \lambda_T$, corresponding to the state equation at each position in the state trajectory. Under this framework, the Lagrangian is defined as

$$\begin{aligned}\mathcal{L} = F(y, \hat{y}) + \delta'(\hat{y} - h_{\text{out}}(x_T, \theta_T)) \\ + \sum_{i=0}^{T-1} \lambda_i^T (x_{i+1} - h_i(x_i, \theta_i)).\end{aligned} \tag{3}$$

The Lagrangian's first term is interpreted as the cost incurred from predicting $\hat{y}$, and the Lagrangian's summation term is interpreted as the cost of abiding by the forward system dynamics. Note, the Lagrangian's partial derivatives with respect to the adjoint state are zero (i.e., $\frac{\partial \mathcal{L}}{\partial \lambda_i} = 0$) if and only if the forward, discrete-time dynamics are satisfied. Analogously, the Lagrangian's partial derivatives with respect to the state variables,

$$\frac{\partial \mathcal{L}}{\partial x_i} = 0 \in \mathbb{R}^{m_i}, \tag{4}$$

generate the backward or adjoint system, which, when satisfied, can be used to compute the derivative of the Lagrangian with respect to the parameters. Moreover, when the adjoint system is satisfied, the Lagrangian's partial derivatives with respect to the parameters are equal to the original cost function's derivatives with respect to the parameters.

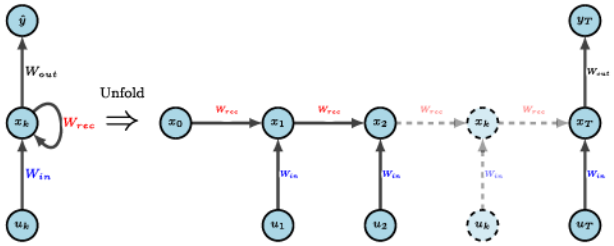

Fig. 1. Graphical representation of a folded and unfolded RNN architecture (many-to-one task).

B. Recurrent Neural Networks and Gradient Decay

We now focus our derivation to a particular type of neural network architecture, called Recurrent Neural Networks (RNNs) (see fig. (1)), designed for situations in which data is observed over time or in sequences of arbitrary length, and for which the aforementioned gradient decay problem is particularly insidious, as we will describe. In order to focus on RNNs, we will need to modify our description of the learning optimization problem, (1), in several ways:

1) The observation previously was (y, x_0). Now, the value of x_0 is randomly assigned, and we observe $(y, u_1, \ldots, u_T)$, where $u_i \in \mathbb{R}^p$ for $i = 1, \ldots, T$.
2) The observation u_i is now an input to the function h_{i-1} that is used to compute x_i.
3) We replace h_i with a single function σ that acts on each component of a vector argument independently.
4) Finally, we require that all of the h_i, for $i = 1, \ldots, T-1$ have a single common parameter.

To summarize, we replace the general feed forward neural network training problem, (1), with the RNN specific training problem

$$\begin{aligned}\min_{\theta} \quad & F(y, \hat{y}) \\ \text{s.t.} \quad & x_t = \sigma(W_{rec} x_{t-1} + W_{in} u_t + b_1), t = 1, \ldots, T \\ & \hat{y} = \sigma(W_{out} x_T + b_2),\end{aligned} \tag{5}$$

where $W_{rec} \in \mathbb{R}^{m \times m}$ is an unknown parameter called the (recurrent) weight matrix; $W_{in} \in \mathbb{R}^{m \times p}$ is an unknown matrix called the input weight matrix; $b_1 \in \mathbb{R}^m$ is an unknown vector called the bias; $W_{out} \in \mathbb{R}^{l \times m}$ is an unknown matrix called the output weight matrix; $b_2 \in \mathbb{R}^l$ is an unknown vector called the output bias; $\theta = \{W_{rec}, W_{in}, W_{out}, b_1, b_2\}$ is shorthand for all of the unknown parameters; and σ is a nonlinear activation function (e.g., sigmoid, tanh, etc.) that acts on each component of its argument independently.

Solving (5) is typically performed using a gradient-based optimization method. As previously described, the gradient of (5) can be computed using the adjoint method. The adjoint method produces the adjoint system

$$\delta = -\frac{\partial F(y_T, \hat{y}_T)}{\partial \hat{y}_T}, \tag{6}$$

$$\lambda_T = W_{out}^T \frac{\partial \sigma}{\partial Z}(W_{out} x_T + b_2) \circ \delta, \text{ and} \tag{7}$$

$$\lambda_k = W_{rec}^T \frac{\partial \sigma}{\partial Z}(W_{rec}x_k + W_{in}u_{k+1} + b_1) \circ \lambda_{k+1}, \quad (8)$$

for $k = T-1, \ldots, 1$, where $\circ$ is the Hadamard product (recall that σ acts entry-wise on the vector argument) and takes precedence over matrix-vector products and Z is used as a dummy variable for ease of notation that takes the value where the derivative is to be evaluated – for example in $\frac{\partial \sigma}{\partial Z}(W_{out}x_T + b_2)$ we have $Z = W_{out}x_T + b_2$. As described above, the adjoint system can now be used to efficiently compute the gradients of the Lagrangian with respect to the parameters and are equal to the gradients of the original objective function with respect to the parameters. The adjoint-based gradients are supplied by

$$\frac{\partial \mathcal{L}_1}{\partial W_{out}} = -\left(\delta \circ \frac{\partial \sigma}{\partial Z}(W_{out}x_T + b_2)\right)x_T^T \quad (9)$$

$$\frac{\partial \mathcal{L}_1}{\partial b_2} = -\frac{\partial \sigma}{\partial Z}(W_{out}x_T + b_2) \circ \delta \quad (10)$$

$$\frac{\partial \mathcal{L}_1}{\partial b_1} = -\sum_{k=1}^{T} \frac{\partial \sigma}{\partial Z}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \lambda_k \quad (11)$$

$$\frac{\partial \mathcal{L}_1}{\partial W_{rec}} = -\sum_{k=1}^{T} \left[\frac{\partial \sigma}{\partial Z}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \lambda_k\right]x_{k-1}^T \quad (12)$$

$$\frac{\partial \mathcal{L}_1}{\partial W_{in}} = -\sum_{k=1}^{T} \left[\frac{\partial \sigma}{\partial Z}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \lambda_k\right]u_k^T \quad (13)$$

Using these formula for the gradients, we can now describe and contextualize the gradient decay problem. First, notice that in (11)-(13), each time point contributes equally to computing the gradients for the recurrent weight matrix, the input weight matrix and the bias. Thus, at the beginning of training, each time point can influence the prediction quality of the RNN equally. Given that each time point corresponds to an input vector u_i, then each input vector can equally influence the prediction quality of the RNN at the beginning of training, and this would only be modified if the minimization induces the parameters to change the impact of each input.

However, owing to the nonlinearity of σ and the matrix-vector products in (6), the adjoint variables, λ_k, tend to zero as $k \downarrow 1$ (note, they can also explode towards infinity, but this is observed less frequently in practice). As a result, the early inputs (i.e., those observed closest to $t = 1$) have a substantially diminished or negligible influence on the choice of parameters, and, consequently, have a negligible influence on the predictions that are being made. If there is no prior knowledge about the relative importance of the inputs, then such a phenomenon would bias the RNN and could result in poorer prediction power. This phenomenon is referred to as the gradient decay or vanishing gradient problem, and is further detailed in [9]. In the following section, we use second-order sensitivity methods to address the gradient decay problem.

III. Using Second-Order Sensitivity Methods to Combat Gradient Decay

A. Our Coadjoint Method

Recall that the gradient decay problem is a consequence of the adjoint variables, defined in (6), tending to zero as $k \downarrow 1$. Thus, to combat the gradient decay problem, we introduce a term to the cost function, G, which will be specified later, that encourages specific behavior in the adjoint variables. Our adjoint variable control can be stated as

$$\begin{aligned} \min_{\theta} \ & F(y, \hat{y}) + G(\lambda_1, \ldots, \lambda_T) \\ \text{s.t.} \ & x_t = \sigma(W_{rec}x_{t-1} + W_{in}u_t + b_1) \\ & \hat{y} = \sigma(W_{out}x_T + b_2) \\ & \lambda_k = W_{rec}^T \frac{\partial \sigma}{\partial Z}(W_{rec}x_k + W_{in}u_{k+1} + b_1) \\ & \circ \lambda_{k+1} \\ & \lambda_T = W_{out}^T \frac{\partial \sigma}{\partial Z_{out}}(W_{out}x_T + b_2) \circ \delta \\ & \delta = -\frac{\partial F(y_T, \hat{y}_T)}{\partial \hat{y}_T} \end{aligned} \quad (14)$$

where $t = 1, \ldots, T$ and $k = 1, \ldots, T-1$; G is a differentiable function, which we call the *adjoint control function*; the first two constraints define the forward system dynamics; the latter three constraints are the adjoint dynamics from (6). Note that the additional constraints supplied from problem (5) to (14) are constraints on the gradients computed during training, not altering the forward dynamics of the network.

At first glance, the optimization problem proposed in (14) seems odd as it imposes constraints on the adjoint variables computed from the original optimization problem (5) under a different objective. However, the adjoint constraints are necessary due to the *sequential* nature of the new objective, $F + G$. Namely, the addition of G and its influence on the optimization path are determined "on top" of the original optimization problem (5). Thus, this new optimization problem proceeds by first computing F and the adjoint system by making a single forward and backward pass over the network, where the forward pass contributes the network's prediction, $\hat{y}$, and the backward pass provides the adjoint variables, λ_k and δ. After computing the prediction and adjoint variables, a second forward and backward pass over the network are used to evaluate G and imbue the characteristics of G into the adjoint variables.

To efficiently compute the gradient with respect to our cost functional, we proceed by introducing another set of adjoint variables and define the associated Lagrangian as in (3). In sensitivity analysis, the introduction of a second set of adjoint variables which act on the primary set of adjoint variables is referred to as *adjoint-over-adjoint* methods. For simplicity, we coin the second set of adjoint variables as coadjoints and the process of computing the parameter gradients with respect to $\mathcal{L}_2$ as the coadjoint method.

Now, our newly introduced coadjoint variables (ψ, ϕ, γ_k and α_k) are the vectors obtained using a procedure identical to the adjoint method, and are related by a corresponding coadjoint system. When the coadjoint system is satisfied, the gradient of the objective function in (14) with respect to the parameters

is equal to the derivative of $\mathcal{L}_2$ with respect to the parameters. The resulting gradients will be used to update the system's parameters and are given by

$$\frac{\partial \mathcal{L}_2}{\partial W_{out}} = \Bigg(-\phi \circ \frac{\partial \sigma}{\partial Z}(W_{out}x_T + b_2) - \delta \circ (W_{out}\gamma_T) \circ \frac{\partial^2 \sigma}{\partial Z^2}(W_{out}x_T + b_2) \Bigg) x_T^T - \Bigg(\delta \circ \frac{\partial \sigma}{\partial Z}(W_{out}x_T + b_2) \Bigg) \gamma_T^T \tag{15}$$

$$\frac{\partial \mathcal{L}_2}{\partial b_2} = -\phi \circ \frac{\partial \sigma}{\partial Z}(W_{out}x_t + b_2) - \delta \circ (W_{out}\gamma_T) \circ \frac{\partial^2 \sigma}{\partial Z^2}(W_{out}x_T + b_2) \tag{16}$$

$$\frac{\partial \mathcal{L}_2}{\partial b_1} = -\sum_{k=1}^{T} \frac{\partial \sigma}{\partial Z}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \alpha_k - \sum_{k=1}^{T-1} \Bigg(\frac{\partial^2 \sigma}{\partial Z^2}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \lambda_{k+1} \circ (W_{rec}\gamma_k) \Bigg) \tag{17}$$

$$\frac{\partial \mathcal{L}_2}{\partial W_{rec}} = -\sum_{k=1}^{T} \Bigg(\frac{\partial \sigma}{\partial Z}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \alpha_k \Bigg) x_{k-1}^T - \sum_{k=1}^{T-1} \Bigg(\frac{\partial^2 \sigma}{\partial Z^2}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \lambda_{k+1} \circ (W_{rec}\gamma_k) \Bigg) x_k^T + \Bigg(\frac{\partial \sigma}{\partial Z}(W_{rec}x_k + W_{in}u_{k+1} + b_1) \circ \lambda_{k+1} \Bigg) \gamma_k^T \tag{18}$$

$$\frac{\partial \mathcal{L}_2}{\partial W_{in}} = -\sum_{k=1}^{T} \Bigg(\frac{\partial \sigma}{\partial Z}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \alpha_k \Bigg) u_k^T - \sum_{k=1}^{T-1} \Bigg(\frac{\partial^2 \sigma}{\partial Z^2}(W_{rec}x_{k-1} + W_{in}u_k + b_1) \circ \lambda_{k+1} \circ (W_{rec}\gamma_k) \Bigg) u_{k+1}^T. \tag{19}$$

B. Selecting the Adjoint Control Function

In our formulation (14), we introduced a function that modifies the behavior of the adjoint variables, $G(\lambda_1, \ldots, \lambda_T)$, but we did not specify its form. Generally, we will choose G depending on characteristics that we wish to imbue the adjoint variables, which is task dependent; thus, we should choose the form of G to mold to the problem at hand.

To illustrate, we will choose a G that addresses the gradient decay problem in the RNNs formulated above. Upon initial inspection, a naive approach to formulating G is to penalize small adjoint lengths. For a simple example, we can choose G to penalize the sizes of $(\lambda_1, \ldots, \lambda_T)$,

$$G(\lambda_1, \ldots, \lambda_T) = -\sum_{k=1}^{T} \log(\|\lambda_k\|_2^2), \tag{20}$$

which would deter adjoint variables from decaying. However, choosing G of this form leads to numerical issues when evaluating the gradient of G due to adjoint decay. For example, the gradient of (20) with respect to λ_k is computed as:

$$\frac{\partial G}{\partial \lambda_k} = -\frac{2}{\|\lambda_k\|_2^2}. \tag{21}$$

Evaluating (21) as $\lambda_k \to 0$ we have the root of the numerical issue: $\lim_{\lambda_k \to 0} \frac{\partial G}{\partial \lambda_k} \to +\infty$. An alternative approach to control the size of the adjoint variables is to minimize the variance between the adjoint layers. With this in hand, G could be constructed as:

$$G(\lambda_1, \ldots, \lambda_T) = \frac{1}{T}\sum_{k=1}^{T} \|\lambda_k\|_2^2 - \left(\frac{1}{T}\sum_{k=1}^{T} \|\lambda_k\|_2 \right)^2. \tag{22}$$

Forming G as in (22) encourages the minimization of the adjoint variance. However, in practice (22) did not encourage the sizes of the adjoints to homogenize to an adequate level, but rather $||\lambda_k|| \to 0$ for all k; thus encouraging the behavior we wish to control, the decay of adjoints. As an alternative approach to avoid such issues, we can choose G inspired from the ideas employed in (20) and (22); namely, penalizing the output layer's adjoint state's size, λ_T, in addition to minimizing the variance between the norms of adjoint states,

$$G(\lambda_1, \ldots, \lambda_T) = \frac{1}{T}\sum_{k=1}^{T} \|\lambda_k\|_2^2 - \left(\frac{1}{T}\sum_{k=1}^{T} \|\lambda_k\|_2 \right)^2 - \log(\|\lambda_T\|_2^2). \tag{23}$$

Note that due to the recursive nature of the adjoint variables, solely penalizing the size of λ_T induces a less "aggressive" penalty function on the sizes of all T adjoint layers.

While (23) certainly induces the adjoint variables to homogenize and avoid being too small, we made further improvements by introducing a weighting function in the variance term that would induce the magnitudes of adjoint variables near the output layer to homogenize before inducing the magnitudes of the adjoint variables near the input layer to homogenize. While these improvements on the choice of G have meaningful practical consequences, we enumerate them here to emphasize several points. First, our coadjoint approach to inducing specific behavior in the adjoint variables is general and can be applied to a variety of choices of G. Second, the choice of the adjoint control function should be designed to address the task and challenges of the problem that is being considered. As a result, our coadjoint method is highly adaptive to the needs of the learning task at hand. We now demonstrate the effectiveness of the coadjoint method on a language processing task.

IV. Experimental Results

A. Long Time Gap Experiment (LTG)

We design a synthetic experiment to specifically highlight the control – or lack there of – of the standard adjoint-based training (i.e., backpropagation). For each example (i.e., each set of $(y, u_1, \ldots, u_{50})$) of eight thousand, the inputs, u_i, are

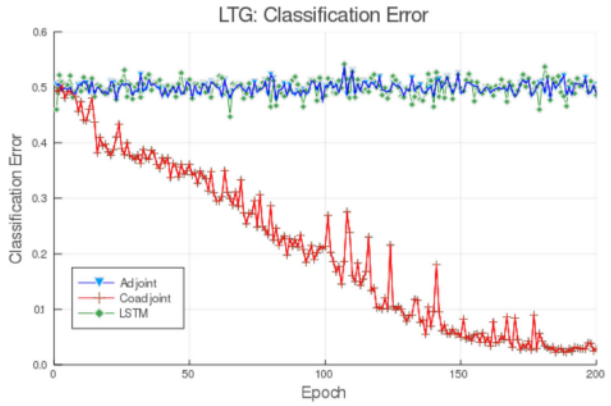

Fig. 2. Test classification error for LTG experiment comparing a RNN trained with the adjoint and coadjoint methods as well as a LSTM network.

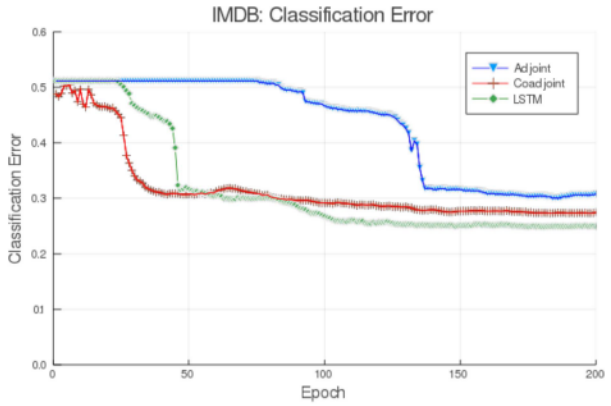

Fig. 3. Comparison of three RNNs: adjoint trained, coadjoint trained and LSTM on test classification error on the IMDB sentiment analysis data set.

independent five-dimensional standard Gaussian random variables. Moreover, for each example, the label, y, is a binary indicator determined by whether $\mathbf{1}^T u_4$ is positive or negative; thus, there exists a perfect classifier for each label given the input sequence. To perform the classification, we train a simple architecture RNN using the adjoint method; we train the same RNN architecture using the coadjoint method; and we train an LSTM network.

Recall that since u_4 is far away from the output layer (note, u_{50} is closest to the output layer), the gradient decay problem will prevent us from training an RNN with a simple architecture. Indeed, this phenomenon is observed in Fig. 2 (blue line): the classification error remains around 0.5, which is equivalent to guessing. Interestingly, this phenomenon is also observed the LSTM network—a complex architecture with multiple gating units—which is designed to avoid such issues; that is, the classification error also remains around 0.5, which is equivalent to guessing (see Fig. 2, green line).

On the other hand, the coadjoint method is effective at avoiding the vanishing gradient problem (see Fig. 2, red line). Thus, on this toy example, the coadjoint method is able to capture the long time-gap correlations for the same RNN for which the adjoint method fails. Furthermore, the coadjoint method is able to start minimizing classification error within the first few epochs, which indicates its data efficiency as well.

B. IMDB Sentiment Analysis

In this example, our goal is to determine the sentiment (either positive or negative) of a film review. To achieve this goal, we train RNNs on twenty five thousand labeled reviews from IMBD [14], and test each RNN's prediction quality on a separate set of twenty five thousand reviews from the same source. The input sequence is a single IMDB review truncated to the first fifty words where each word in the review is embedded in $\mathbb{R}^{100}$; thus, the input sequence is $\{u_i \in \mathbb{R}^{100}\}_{i=1}^{50}$. We train a simple RNN architecture using just the adjoint method; we train the same architecture using the coadjoint method; and, finally, we train the complex LSTM network using the adjoint method. For each resulting RNN, the classification error on the left-out twenty five thousand reviews are shown in Fig. 3.

TABLE I
COMPARISON OF THREE RNNS—ADJOINT TRAINED, COADJOINT TRAINED AND LSTM—SGD UPDATE TIME IN SECONDS AT DIFFERENT EPOCHS DURING TRAINING

Classifier	Parameter Update Time				
	Epoch 1	Epoch 25	Epoch 50	Epoch 150	Epoch 250
RNN Adjoint	0.0023	0.0011	0.0011	0.0011	0.0011
RNN Coadjoint	0.0091	0.0027	0.0027	0.0027	0.0027
LSTM Adjoint	0.0139	0.0260	0.0253	0.0245	0.0256

The results indicate that all three RNNs – adjoint trained, coadjoint trained and LSTM – are able to eventually reach similar classification rates if exposed to enough training examples. However, the coadjoint method is able to begin training with fewer examples (i.e., has greater data efficiency). Moreover, the coadjoint method also trains in less wall-clock time. While the coadjoint approach has greater data efficiency that will reduce the training time, the coadjoint method is also particularly computationally efficient as shown in Table I, which is the average time for a parameter to be updated at different points in the training regime.

In particular, the coadjoint method is about three fold slower than the adjoint approach for the simple RNN whereas the method is about ten fold faster than training an LSTM architecture. Thus, the coadjoint approach beats out the adjoint approach owing to its data efficiency; while it beats out the LSTM network training owing to the faster speed with which parameter updates can be made.

To reiterate, on this sentiment analysis task, the coadjoint method can train a simple RNN more data-efficiently than the standard approach for such a network, which results in faster training time. Moreover, the coadjoint method trains a simple RNN to a performance level that is comparable to an LSTM network but faster owing to the efficiency of the second-order sensitivity calculation of the gradients.

C. Sequential and Permuted MNIST

Using a subset of 28×28 images of handwritten digits taken from the MNIST dataset we define a sequential data task to

TABLE II
PERFORMANCE COMPARISON OF ADJOINT AND COADJOINT TRAINED RNNS ON THE SEQUENTIAL AND PERMUTED MNIST DATA TASKS

Classifier	MNIST Results		
	F	Test Error	$\|\|\lambda_1\|\|_2$
Seq. - Adjoint	14235.1	0.292	$O(10^{-7})$
Seq. - Coadjoint	14748.5	0.332	$O(10^{-6})$
Perm. - Adjoint	20268.3	0.652	$O(10^{-17})$
Perm. - Coadjoint	15449.1	0.333	$O(10^{-6})$

classify the digits {0, 1, 2} [15]. For the sequential MNIST task we define the i^{th} element of the input sequence, $\{u_i \in \mathbb{R}^{28}\}_{i=1}^{28}$, as the i^{th} column of the original input image. We train two RNNs with the same architecture: one with the adjoint method and one with the coadjoint method.

To add difficulty to the sequential MNIST task a fixed permutation is applied to each training image (permuted MNIST task). The permutation increases learning difficulty as it strips the input of associations provided from adjacent input. Now, the i^{th} element of the input sequence, $\{u_i \in \mathbb{R}^{28}\}_{i=1}^{28}$, corresponds to the *ith* permuted column of the original input image. Again, we train two RNNs with the same architecture: one with the adjoint method and one with the coadjoint method where we seek to learn to classify the handwritten digits 0, 1 and 2.

The results for both the sequential and permuted MNIST classification tasks are presented in Table II where F is the training cross-entropy loss and time refers to the number of seconds to complete an epoch. The results indicate that both the adjoint and coadjoint trained networks perform similarly for the sequential MNIST task. However, permuting the input sequence has drastically different impacts on the effectiveness of the adjoint and coadjoint methods. The coadjoint method performs almost identically to the sequential MNIST task, while the adjoint method is barely able to do better than random guessing which would result in a test accuracy of 0.66. This observation points to the coadjoint method being particularly advantageous in tasks where adjacent elements in the input sequence are not highly correlated.

V. CONCLUSION

In this letter, we introduced the coadjoint method for training deep neural networks. Our coadjoint method adds a penalty term on the adjoint states during the training process in order to induce certain properties to the deep neural networks and address certain computational challenges. Owing to its derivation from second-order sensitivity methods, our coadjoint method is general and can be applied to any number of learning tasks. Moreover, owing to its derivation from second-order sensitivity methods, our coadjoint method requires only a marginal increase in the computational burden relative to other state-of-the-art approaches.

Moreover, we demonstrated the effectiveness of our coadjoint methods on training a recurrent neural networks. In particular, we showed that the coadjoint method can successfully address the gradient decay problem that is endemic to such neural networks, and require minimal additional computational burdens in comparison to state-of-the-art approaches designed to address the gradient decay problem.

In the future, we will extend our coadjoint method to work with these more complex network architectures in order to reduce their overall training costs, and to extend this framework to other deep learning tasks. In addition, we will continue to demonstrate the effectiveness of our coadjoint method on more complex and more realistic learning tasks.

REFERENCES

[1] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. Cambridge, MA, USA: Cambridge Univ. Press, 2009.

[2] E. Laporte and P. L. Tallec, "Numerical methods in sensitivity analysis and shape optimization," in *Modeling and Simulation in Science, Engineering and Technology*. New York, NY, USA: Springer, 2002.

[3] G. Chavent, "Identification of functional parameters in partial differential equations," in *Identification of Parameters in Distributed Systems*, R. E. Goodson and M. Polis, Eds. New York, NY, USA: ASME, 1974, pp. 31–48.

[4] L. Métivier, R. Brossier, S. Operto, and J. Virieux, "Second-order adjoint state methods for full waveform inversion," Jun. 2012.

[5] R. Pratt, C. Shin, and G. J. Hicks, "Gauss–newton and full newton methods in frequency–space seismic waveform inversion," *Geophys. J. Int.*, vol. 133, pp. 341–362, Feb. 2002.

[6] Y. L. Cun, "A theoretical framework for back-propagation," in *Proc. Connectionist Models Summer School*, 1988, pp. 21–28.

[7] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[8] R. Sun, "Optimization for deep learning: Theory and algorithms," 2019. [Online]. Available: https://arxiv.org/abs/1912.08957.

[9] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[10] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[11] S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems," in *Advances in Neural Information Processing Systems 9*. Cambridge, MA, USA: MIT Press, 1997, pp. 473–479.

[12] T. Masuko, "Computational cost reduction of long short-term memory based on simultaneous compression of input and hidden state," in *Proc. IEEE Autom. Speech Recognit. Understanding Workshop (ASRU)*, Okinawa, Japan, Dec. 2017, pp. 126–133.

[13] V. N. Vapnik, *Statistical Learning Theory*. New York, NY, USA: Wiley, 1998.

[14] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguist. Human Lang. Technol.*, Portland, OR, USA, Jun. 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[15] Y. LeCun and C. Cortes. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist/