Game Theoretic Neural ODE Optimizer

Panagiotis Theodoropoulos [1] **Guan-Horng Liu** [*1] **Tiarnong Chen** [*1] **Evangelos A. Theodorou** [1]

Abstract

In this work, we present a novel Game Theoretic Neural Ordinary Differential Equation (Neural ODE) optimizer based on the minimax Differential Dynamic Programming paradigm. As neural networks and neural ODEs tend to be vulnerable to attacks, and their predictions are fragile in the presence of adversarial examples, we aim to design a robust game theoretic optimizer based on principles of Min-Max Optimal Control. By formulating Neural ODE optimization as a Min-Max Optimal Control Problem, our proposed algorithm aims to enhance the robustness of neural networks against adversarial attacks by finding policies that perform well under worst-case scenarios. Leveraging recent advances in the interpretation of Neural ODE training through an Optimal Control Problem perspective, we extend recent second-order optimization techniques to a game theoretic setting and adapt them to our proposed method. This allows our optimizer to efficiently handle the increased complexity stemming from the computation of double the amount of learnable parameters. The resulting optimizer, Game Theoretic Second-Order Neural Optimizer (GTSONO), enables more effective exploration of the control policy space, leading to improved robustness against adversarial attacks. Experimental evaluations on benchmark datasets demonstrate the superiority of GTSONO compared to existing state-of-the-art optimizers in terms of both performance and efficiency against state-of-the-art adversarial defense methods.

1. Introduction

Continuous deep learning architectures have recently gained quite significant popularity. This genre of models, where the ODE of the state derivative is parameterized by an DNN are called Neural ODEs (Chen et al., 2018) and concern the following optimization

$$\min_{\theta} \mathcal{L}(\mathrm{x}(\mathrm{t_f})), \text{where} \quad \frac{\mathrm{d}\mathbf{x}}{\mathrm{dt}} = \mathrm{F}(\mathrm{t}, \mathbf{x}(\mathrm{t}), \theta), \quad \mathrm{x}(\mathrm{t_0}) = \mathrm{x_0} \tag{1}$$

where in its general form $\mathbf{x}(t) \in \mathbb{R}^m$ is the state vector, and $(\cdot, \cdot, \theta)$ is the dynamics of the vector field of $\mathbf{x}$ parameterized by $\theta \in \mathbf{R}^n$.

Chen (2018) introduced the Adjoint Sensitivity Method (ASM) as a first order method to perform reverse-differentiation and train these models, which is based on Pontryagin's principle. Although the authors noted that it is possible to compute higher order derivatives, this causes accumulation of the integration error. Motivated by this, (Liu et al., 2021a) introduced a novel second-order framework. Their approach involved recasting the Neural ODE formulation, represented by equation 10, within the framework of continuous time Differential Dynamic Programming (DDP), which is a well-established second-order optimal control method. By formulating the training process as an optimal control problem, the authors successfully devised an efficient second order optimizer for updating the parameters of Neural ODEs. Intuitively, this extends the underlying connection between OCP and the training of discrete NNs. In (Liu et al., 2021b), a connection between the training process of NNs and DDP was established and it was further shown that the DDP update policy constitutes a generalization of Gradient Descent (GD) as the update rule of the weights and biases in NNs under some assumptions.

In their work to elucidate the underlying operations in Neural ODEs, (Massaroli et al., 2020) describe these continuous models as the latest instance in the pursuit of robustness in traditional neural networks. Machine learning (ML) models are often vulnerable to adversarial examples, deliberately perturbed inputs designed to mislead a model ((Tramèr et al., 2017), (Goodfellow et al., 2014)). Despite their superhuman performance in many computer vision tasks, recent findings demonstrate that deep neural networks tend to be more fragile (Liu et al., 2020). Adversarial training in machine learning aims to improve the robustness and generalization of models by training a model that can withstand such attacks on its input data. The robustness of Neural ODEs

[*]Equal contribution [1]Georgia Institute of Technology. Correspondence to: <ptheodor3@gatech.edu>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023.

against adversarial examples is discussed showing great performance against white box attacks (Chu et al., 2020), (Carrara et al., 2019). (Huang et al., 2022) proposed that the adversarial robustness in stabilized Neural ODEs might stem from obfuscated gradients, as the adaptive stepsize of numerical ODE solvers have a gradient masking effect that causes the PGD attacks to fail. In this vein, to further explore the robustness of Neural ODEs, (Liu et al., 2020) examined how the injection of noise in Neural ODEs can generate more stable predictions.

In the realm of OCP, it has been demonstrated that game-theoretic formulation enables the model handling uncertainties and external disturbances. The first min-max formulation was introduced by (Sun et al., 2015). This algorithm has been successfully tested in real-life robotic applications. For instance, it was shown that it can handle the steering and stabilization of a quadrotor subjected to external distrubances, and come up with the control policy for a biped robot to successfully walk under unknown disturbances (Sun et al., 2018), (Morimoto et al., 2003).

In this work, drawing inspiration by the novel connection of Neural ODE training and continuous time OCP, we adapt the Min-Max OCP paradigm into the training of Neural ODEs and introduce a second order Game Theoretic Neural Optimizer. The resulting framework, called GTSONO, based on the continuous time Min-Max DDP algorithm, employing the ability of this algorithm to handle well uncertainties and external disturbances. It is aimed to transfer this ability into the Neural ODE training seeking a control policy or equivalently a weight configuration that robustifies the model and immunizes it to external disturbances. The extension of the optimal control problem (OCP) perspective from discrete models, such as deep neural networks (DNNs), to continuous models like Neural ODEs necessitates specialized treatment based on continuous-time OCP theory (Todorov et al., 2006; Liu et al., 2021a), however this fundamental connection motivates the development of principled algorithms for the training of Neural ODEs.

The connection established between our methodology and DDP, a second order OCP method, motivates the derivation of second-order derivatives and effectively optimize the configuration for optimal performance. However, implementing second-order methods can be computationally intractable, especially for high-dimensional systems, such neural networks. Additionally,our algorithm entails seeking the optimal configuration of twice the number of weights, and storing more second order matrices. As a result, it becomes imperative to employ efficient second-order methods to effectively manage the heightened computational complexity. In this vein, by incorporating Kronecker factorization (Martens & Grosse, 2015), we are able to decompose the second-order matrices into vectors, facilitating more efficient computations. Therefore, our second-order method is found to come at no additional memory cost than counterpart first order methods. After the derivation of efficient preconditioning, we see that the resulting update law constitutes an approximation of the Natural Gradient Descent, as we approximate the Fischer Information Matrix through Gauss-Newton method. Furthermore, we also examine a generalization of this update law based on the open loop Min-Max DDP.

2. Preliminaries

2.1. Min-Max Differential Dynamic Programming

The discussion on the Min-Max Differential Dynamic Programming starts with the optimization problem.

$$\min_{\mathbf{u}} \max_{\mathbf{v}} J(\mathbf{u}, \mathbf{v}) \tag{2}$$

where $J(\mathbf{u}, \mathbf{v})$ is the cost function defined as: $J(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{x}, t_f) + \int L(\mathbf{x}, \mathbf{u}, \mathbf{v})dt$. We denote as $\mathbf{u}, \mathbf{v}$ the controls applied by the two antagonizing players. It is assumed that the player in control of the $\mathbf{u}$ variable wants to minimize the cost function, whereas the other player wants to maximize it. Consider the dynamics describing the aforementioned OCP problem being given by the following ODE:

$$\frac{\mathrm{d}\mathbf{x}(\mathrm{t})}{\mathrm{dt}} = F(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t), \quad \mathrm{x}(\mathrm{t}_0) = \mathrm{x}_0 \tag{3}$$

where $\mathbf{x}(t) \in \mathcal{X}$ is the state of the dynamic system. We proceed to define the value function for our problem as the function expressing the minmax value of the cost function at time $t = t_0$ and $x = x_0$.

$$\mathrm{V}(\mathrm{t}_0, \mathrm{x}_0) = \min_{\gamma_u} \max_{\gamma_v} \Bigg\{ \phi(t_f, \mathbf{x}(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t, \mathbf{x}(t)), \mathbf{v}(t, \mathbf{x}(t)), t)\mathrm{d}\tau \Bigg\} \tag{4}$$

Using the Bellman principle, we write (49), as follows:

$$\mathrm{V}(\mathrm{t}, \mathrm{x}(\mathrm{t})) = \min_{\mathbf{u}} \max_{\mathbf{v}} \Bigg\{ V(t + dt, \mathbf{x}(t + dt) + \int_{t}^{t+dt} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t, \mathbf{x}(t)), \mathbf{v}(t, \mathbf{x}(t)), t)\mathrm{d}\tau \Bigg\} \tag{5}$$

The detailed derivation of the Bellman equation is left in the Appendix B.2. From 5, we can derive the Hamilton-Jacobi-Bellman-Isaacs (HJBI) PDE

$$-\frac{\partial \mathrm{V}}{\partial t} = \min_{\mathbf{u}} \max_{\mathbf{v}} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t) + \mathrm{V}_{\mathbf{x}}(\mathbf{x}, \mathrm{t})^{\intercal} \mathrm{F}(\mathbf{x}, \mathbf{u}, \mathbf{v}, \mathrm{t}) \tag{6}$$

with boundary conditions $\mathrm{V}(\mathbf{x}(\mathrm{t_f}), \mathrm{t_f}) = \phi(\mathbf{x}_{\mathrm{t_f}}, \mathrm{t_f})$.

Subsequently, we define the Bellman objective function $\mathrm{Q}(\mathrm{t}, \mathbf{x}, \mathbf{u}, \mathbf{v})$ to be the optimization objective from 49. The Bellman principle allows us to interpret the optimization problem as a sequence of minimization over each $\mathbf{u}$ control and maximization over each $\mathbf{v}$ (Liu et al., 2021b). However, solving the Bellman equation for high-dimensional problems becomes infeasible. For this reason, given a nominal trajectory at time t, $(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t, \bar{\mathbf{v}}_t)$, during the backward pass, we approximate the Bellman equation locally around the nominal trajectory through a second order Taylor expansion. The goal of the min-max DDP is iteratively find the update laws of the control variables that minimize and maximize J, through the following minimization and maximizations

$$\delta\mathbf{u}_t^* = \arg\min_{\delta\mathbf{u}_t}\left\{ \begin{bmatrix} 1 \\ \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \\ \delta\mathbf{v}_t \end{bmatrix}^{\intercal} \begin{bmatrix} 0 & Q_{\mathbf{x}} & Q_{\mathbf{u}} & Q_{\mathbf{v}} \\ Q_{\mathbf{x}} & Q_{xx} & Q_{xu} & Q_{xv} \\ Q_{\mathbf{u}} & Q_{ux} & Q_{uu} & Q_{uv} \\ Q_{\mathbf{v}} & Q_{vx} & Q_{vu} & Q_{vv} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \\ \delta\mathbf{v}_t \end{bmatrix} \right\} \tag{7a}$$

$$\delta\mathbf{v}_t^* = \arg\min_{\delta\mathbf{v}_t}\left\{ \begin{bmatrix} 1 \\ \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \\ \delta\mathbf{v}_t \end{bmatrix}^{\intercal} \begin{bmatrix} 0 & Q_{\mathbf{x}} & Q_{\mathbf{u}} & Q_{\mathbf{v}} \\ Q_{\mathbf{x}} & Q_{xx} & Q_{xu} & Q_{xv} \\ Q_{\mathbf{u}} & Q_{ux} & Q_{uu} & Q_{uv} \\ Q_{\mathbf{v}} & Q_{vx} & Q_{vu} & Q_{vv} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \\ \delta\mathbf{v}_t \end{bmatrix} \right\} \tag{7b}$$

Finally, the optimization problems in 7a have a closed form solution, and can be compactly written as follows

$$\begin{bmatrix} \delta\mathbf{u}_t^* \\ \delta\mathbf{v}_t^* \end{bmatrix} = \begin{bmatrix} l_{\mathbf{u}} \\ l\mathbf{v} \end{bmatrix} + \begin{bmatrix} K_{\mathbf{u}} \\ K\mathbf{v} \end{bmatrix} \delta\mathbf{x}_t \tag{8}$$

where $\delta\mathbf{x}_t$ is the state differential, and the feed-forward gains $l_{\mathbf{u}}$, along with $l_{\mathbf{v}}$, and the feedback gains $K_{\mathbf{u}}$, $K_{\mathbf{v}}$ are given by

$$\begin{aligned} l_{\mathbf{u}} &= \tilde{Q}_{\mathbf{uu}}^{-1}(Q_{\mathbf{uv}}Q_{\mathbf{vv}}^{-1}Q_{\mathbf{v}} - Q_{\mathbf{u}}), \text{ and} \\ K_{\mathbf{u}} &= \tilde{Q}_{\mathbf{uu}}^{-1}(Q_{\mathbf{ux}} - Q_{\mathbf{uv}}Q_{\mathbf{vv}}^{-1}Q_{\mathbf{vx}}) \end{aligned} \tag{9a}$$

$$\begin{aligned} l_{\mathbf{v}} &= \tilde{Q}_{\mathbf{vv}}^{-1}(Q_{\mathbf{uv}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} - Q_{\mathbf{v}}), \text{ and} \\ K_{\mathbf{v}} &= \tilde{Q}_{\mathbf{vv}}^{-1}(Q_{\mathbf{vx}} - Q_{\mathbf{uv}}Q_{\mathbf{vv}}^{-1}Q_{\mathbf{vx}}) \end{aligned} \tag{9b}$$

with $\tilde{Q}_{uu} = (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu})$, and $\tilde{Q}_{vv} = (Q_{vv} - Q_{vu}Q_{uu}^{-1}Q_{uv})$. Substituting 23 into the Taylor expansion of the Bellman function, we obtain the backward ODEs to back-propagate $V_{\mathbf{x}}$, and $V_{\mathbf{xx}}$. A detailed derivation of the update laws and the backward ODEs of Min-Max DDP is provided in the Appendix B.2.

2.2. Neural ODEs

In general, Neural ODEs concern the following optimization over an objective function: $\mathcal{L}$:

$$\min_{\theta} \mathcal{L}(\mathbf{x}(t)(t_f)), \text{where } \frac{d\mathbf{x}}{dt} = F(t, \mathbf{x}(t), \theta), \mathbf{x}(t)(t_0) = \mathbf{x}_0 \tag{10}$$

, where $\mathbf{x}(t) \in \mathbb{R}^m$, and $F(\cdot, \cdot, \theta)$ is a deep neural network parameterized by $\theta \in \mathbb{R}^n$. Solving the ODE involves a call of an ODE solver. In practice higher order integrations methods such as Runge-Kutta, Dopri5 are employed which provide with more accurate results than vanilla Euler discretization (Liu et al., 2021a). Back propagation of Neural ODEs involves obtaining the gradient $\frac{\partial\mathcal{L}}{\partial\theta}$. Naively, one could try to back-propagate through solving the ODE by calling the same solver mentioned above, however this results in intractable algorithms with very high memory complexity, as the computation graph can grow arbitrarily large for adaptive ODE solvers. For this reason, (Chen et al., 2018) proposed the Adjoint Sensitivity Method, which enables computing the gradient through the following integration

$$\frac{\partial\mathcal{L}}{\partial\theta} = -\int_{t_1}^{t_0} \mathrm{a(t)} \frac{\partial \mathrm{F(t, \mathbf{x}_t, \theta)}}{\partial\theta} \mathrm{dt}, \tag{11}$$

where $a(t) \in \mathbb{R}^m$ is the adjoint state, whose dynamics obey the backward ODE

$$-\frac{\mathrm{da(t)}}{\mathrm{dt}} = a(t)^{\intercal} \frac{\partial F(t, \mathbf{x}, \theta)}{\partial \mathbf{x}} \tag{12}$$

with terminal condition $a(t_1) = \frac{\partial\mathcal{L}}{\partial x(t_1)}$. These coupled ODEs in 11, and 12 are the equivalent continuous-time expression of the Back-propagation (Lecun, 1988). As mentioned in the 1, the authors mentioned that computation of higher order derivatives is possible via this formulation, by recursively calling $\frac{\partial^n\mathcal{L}}{\partial\theta^n} = \text{grad}(\frac{\partial^{n-1}\mathcal{L}}{\partial\theta^{n-1}}$, however this causes accumulation of the integration error as pointed out in (Liu et al., 2021a).

3. Methodology

3.1. Training Neural Dynamics using Game Theoretic Optimal Control Theory

Our proposed methodology , based on the continuous time minmax DDP algorithm, leverages the ability of this algorithm to handle well uncertainties and external disturbances and rendering second order methods for Neural ODEs more robust.

Converting (10) into an expression that is easier to implement through a min-max continuous time Optimal Control perspective, we obtain:

$$\begin{aligned} &\min_{\mathbf{u}} \max_{\mathbf{v}} \left[\Phi(x_{t_f}) + \int_{t_0}^{t_f} \ell(t, \mathbf{x}, \mathbf{u}, \mathbf{v}) \mathrm{dt} \right], \\ &\text{subjected to } \begin{cases} \frac{\mathrm{d}\mathbf{x}}{dt} = F(t, \mathbf{x}, \mathbf{u}, \mathbf{v}), & \mathbf{x}(t_0) = \mathbf{x}_0 \\ \frac{\mathrm{d}\mathbf{u}}{dt} = 0, & \mathbf{u}(t_0) = \theta \\ \frac{\mathrm{d}\mathbf{v}}{dt} = 0, & \mathbf{v}(t_0) = \eta \end{cases} \end{aligned} \tag{13}$$

where $\mathbf{x} \equiv \mathbf{x}(t) \in \mathbb{R}^m$, $\mathbf{u} \equiv \mathbf{u}(t) \in \mathbb{R}^n$, and $\mathbf{v} \equiv \mathbf{v}(t) \in \mathbb{R}^n$. It is clear that (27) describes (10) without loss of generality by taking $(\Phi, \ell) = (\mathcal{L}, 0)$. The function $F(t, \mathbf{x}, \mathbf{u}, \mathbf{v})$

characterizes the vector field and is parameterized by a Deep Neural Network (DNN). We consider that the dynamics of the system are symetric with respect to the two sets of weights $(\mathbf{u}, \mathbf{v})$. The functions Φ, and ℓ are known as the terminal and running cost in the context of OCP. This problem is understood as particular type of OCP that searches for the optimal initial condition of the time-invariant controls $\mathbf{u}_t, \mathbf{v}_t$. In the DNN setting, the terminal cost is equivalent to the loss function, for instance Categorical Cross Entropy for multi-label classification, and the running cost is equivalent to the weight decay, or some other regularization techniques that acts on the weights of the intermediate hidden layers. Next, the accumulated loss $Q(t, \mathbf{x}, \mathbf{u}, \mathbf{v})$ is defined as follows:

$$Q(t, \mathbf{x}, \mathbf{u}, \mathbf{v}) = \Phi(\mathbf{x}(t_f)) + \int_t^{t_f} \ell(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau), \mathbf{v}(\tau)) \mathrm{d}\tau \tag{14}$$

From this definition of the accumulated loss implies that Q we can readily obtain:

$$0 = \ell(t, \mathbf{x}, \mathbf{u}, \mathbf{v}) + \frac{\mathrm{dQ(t, \mathbf{x}, \mathbf{u}, \mathbf{v})}}{\mathrm{dt}} \tag{15}$$

At this point our goal is to compute higher-order derivatives with respect to $\mathbf{u}, \mathbf{v}$. Under the OC formulation the gradient of the loss function with respect to the parameters θ, and η $\frac{\partial \mathcal{L}}{\partial \theta}$, $\frac{\partial \mathcal{L}}{\partial \eta}$ respectively are equivalent to $\frac{\partial Q(t_0, \mathbf{x}_{t_0}, \mathbf{u}_{t_0}, \mathbf{v}_{t_0})}{\partial \mathbf{u}_{t_0}}$, and $\frac{\partial Q(t_0, \mathbf{x}_{t_0}, \mathbf{u}_{t_0}, \mathbf{v}_{t_0})}{\partial \mathbf{v}_{t_0}}$ in the context of Neural ODEs. In similar fashion, the hessian of the loss $\frac{\partial^2 \mathcal{L}}{\partial \theta^2}$ is identical to $\frac{\partial^2 Q(t_0, \mathbf{x}_{t_0}, \mathbf{u}_{t_0}, \mathbf{v}_{t_0})}{\partial u_{t_0}^2}$, similarly for the hessians of the loss with respect to η, and $\mathbf{v}(t_0)$. This is because we set $u_{t_0} = \theta$, and $\mathbf{v}(t_0) = \eta$ by construction. Intuitively, we can interpret $Q(t_0, \mathbf{x}_{t_0}, \mathbf{u}_{t_0}, \mathbf{v}_{t_0})$ as the accumulation of the loss over the entire interval $[t_0, t_f]$, thus it represents $\mathcal{L}$ satisfactorily. Now, we frame the equation above along a nominal path to derive the ODEs that will help us to compute the second order derivatives at t_0.

Theorem 3.1. *Consider nominal trajectory $(\bar{\mathbf{x}}, \bar{\mathbf{u}}, \bar{\mathbf{v}})$, that satisfies the ODEs in the constraints of (27) is considered. Then, the function Q along with its first and second derivatives expanded locally around the nominal trajectory obey the following backward ODEs:*

$$-\frac{\mathrm{dQ_x}}{\mathrm{dt}} = F_x Q_x + \ell_x, \tag{16a}$$

$$-\frac{\mathrm{dQ_{xx}}}{\mathrm{dt}} = \ell_{\mathbf{xx}} + F_{\mathbf{x}} Q_{\mathbf{xx}} + Q_{\mathbf{xx}} F_{\mathbf{x}}^\intercal, \tag{16b}$$

$$-\frac{\mathrm{dQ_{xu}}}{\mathrm{dt}} = \ell_{\mathbf{xu}} + F_{\mathbf{x}} Q_{\mathbf{xu}} + Q_{\mathbf{xx}} F_{\mathbf{u}}^\intercal \tag{16c}$$

$$-\frac{\mathrm{dQ_u}}{\mathrm{dt}} = F_{\mathbf{u}} Q_{\mathbf{x}} + \ell_{\mathbf{u}}, \tag{16d}$$

$$-\frac{\mathrm{dQ_{uu}}}{\mathrm{dt}} = \ell_{\mathbf{uu}} + F_{\mathbf{u}} Q_{\mathbf{xu}} + Q_{\mathbf{ux}} F_{\mathbf{u}}^\intercal, \tag{16e}$$

$$-\frac{\mathrm{dQ_{uv}}}{\mathrm{dt}} = \ell_{\mathbf{uv}} + F_{\mathbf{u}} Q_{\mathbf{xv}} + Q_{\mathbf{ux}} F_{\mathbf{v}}^\intercal \tag{16f}$$

$$-\frac{\mathrm{dQ_v}}{\mathrm{dt}} = F_{\mathbf{v}} Q_{\mathbf{x}} + \ell_{\mathbf{v}}, \tag{16g}$$

$$-\frac{\mathrm{dQ_{vv}}}{\mathrm{dt}} = \ell_{\mathbf{vv}} + F_{\mathbf{v}} Q_{\mathbf{xv}} + Q_{\mathbf{vx}} F_{\mathbf{v}}^\intercal \tag{16h}$$

$$-\frac{\mathrm{dQ_{xv}}}{\mathrm{dt}} = \ell_{\mathbf{xv}} + F_{\mathbf{x}} Q_{\mathbf{xv}} + Q_{\mathbf{xx}} F_{\mathbf{v}}^\intercal \tag{16i}$$

The terminal conditions are given by: $Q_{\mathbf{x}}(t_f) = \Phi_{\mathbf{x}}, Q_{\mathbf{xx}}(t_f) = \Phi_{\mathbf{xx}}$, and $Q_{\mathbf{u}}(t_f) = Q_{\mathbf{v}}(t_f) = Q_{\mathbf{uu}}(t_f) = Q_{\mathbf{vv}}(t_f) = Q_{\mathbf{uv}}(t_f) = Q_{\mathbf{xu}}(t_f) = Q_{\mathbf{xv}}(t_f) = 0$.

Remark This theorem implies that the proposed framework can obtain first and second order derivatives with a single pass of an ODE solver, without any recursive computations. This contributes to avoiding accumulating integration errors, and increased runtimes.

3.2. Second-order Matrix Factorization

Theorem 1 indeeds set the foundations for an efficient second order game theoretic framework. However, there is still the issue of the dimensionality. The number of learnable parameters in networks with highly complicated architectures grows easily to an unfavorable number, especially in our Game-Theoretic framework, dealing with two sets of weights, and every operation takes place twice, rendering the inversion of the Hessians intractable. Therefore, it is vital to establish an efficient methodology to approximate the second order matrices.

Theorem 3.2. *We assume the matrix $Q_{xx}(t_1)$ to be a symmetric matrix of rank $R \leq m$, it may be represented as: $Q_{xx} = \sum_{i=1}^{R} \mathbf{y}_i \mathbf{y}_i^\intercal$, where $\mathbf{y}_i \in \mathbb{R}^m$. Then $\forall t \in [t_0, t_f]$, the second order matrices in (33) that contain derivative with respect to the state can be decomposed as follows:*

$$Q_{\mathbf{xx}}(t) = \sum_{i=1}^{R} \mathbf{q}_i(t)\mathbf{q}_i(t)^\intercal, \quad Q_{\mathbf{xu}}(t) = \sum_{i=1}^{R} \mathbf{q}_i(t)\mathbf{p}_i(t)^\intercal$$

$$Q_{\mathbf{uu}}(t) = \ell_{uu} + \sum_{i=1}^{R} \mathbf{p}_i(t)\mathbf{p}_i(t)^\intercal, \quad Q_{\mathbf{uv}}(t) = \sum_{i=1}^{R} \mathbf{p}_i(t)\mathbf{s}_i(t)^\intercal,$$

$$Q_{\mathbf{xv}}(t) = \sum_{i=1}^{R} \mathbf{q}_i(t)\mathbf{s}_i(t)^\intercal, \quad Q_{\mathbf{vv}}(t) = \ell_{vv} + \sum_{i=1}^{R} \mathbf{s}_i(t)\mathbf{s}_i(t)^\intercal, \tag{17}$$

where the vectors $\mathbf{q}_i \in \mathbb{R}^m$, and $\mathbf{p}_i(t)$, and $\mathbf{s}_i(t) \in \mathbb{R}^n$ obey the following backward ODEs:

$$-\frac{\mathrm{d\mathbf{q}_i(t)}}{\mathrm{dt}} = F_{\mathbf{x}} \mathbf{q}_i(t), \quad -\frac{d\mathbf{p}_i(t)}{\mathrm{dt}} = F_{\mathbf{u}} \mathbf{q}_i(t), \quad -\frac{\mathrm{d\mathbf{s}_i(t)}}{\mathrm{dt}} = F_{\mathbf{v}} \mathbf{q}_i(t) \tag{18}$$

with the terminal condition given by $(\mathbf{q}_i(t_f), \mathbf{p}_i(t_f), \mathbf{s}_i(t_f)) = (\mathbf{y}_i, 0, 0)$.

Remark This theorem suggests that the coupled matrices back propagated through (33) can be decomposed into a set of vectors. As the number of vectors is dependent on the rank of $Q_{xx}(t_f)$; lower rank matrices observed in many Neural ODE applications (Chen et al., 2018) alleviate the computational load and provide great memory efficiency.

Assumption We assume that the running cost ℓ is twice differentiable and only dependent quadratic with respect to the antagonizing controls, namely: $\ell \equiv \ell(\mathbf{u}, \mathbf{v})$, with $\ell_{uu} = R_{\mathbf{u}} I$, and $\ell_{vv} = R_{\mathbf{v}} I$ and $\ell_{uv} = 0$, where I is the identity matrix, $R_{\mathbf{v}}$ and $R_{\mathbf{u}}$ are scalars, with $R_{\mathbf{v}}$ being larger than $R_{\mathbf{u}}$. The intuition for this selection is that we wish to penalize the disturbance more severely and prevent instability. Based on the above, substituting (18) in (17), the second order matrices can be formulated as follows:

$$Q_{\mathbf{uu}}(t) = R_{\mathbf{u}} I(t - t_f) + \tag{19a}$$

$$\sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{u}} \mathbf{q}_i \mathrm{dt} \Big) \Big(\int_{\mathrm{t_f}}^{\mathrm{t}} \mathrm{F_u q_i dt} \Big)^{\intercal} \tag{19b}$$

$$Q_{\mathbf{vv}}(t) = R_v I(t - t_f) + \tag{19c}$$

$$\sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{v}} \mathbf{q}_i \mathrm{dt} \Big) \Big(\int_{\mathrm{t_f}}^{\mathrm{t}} \mathrm{F_v q_i dt} \Big)^{\intercal} \tag{19d}$$

$$Q_{\mathbf{uv}}(t) = \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{u}} \mathbf{q}_i \mathrm{dt} \Big) \Big(\int_{\mathrm{t_f}}^{\mathrm{t}} \mathrm{F_v q_i dt} \Big)^{\intercal} \tag{19e}$$

$$Q_{\mathbf{ux}}(t) = \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{x}} \mathbf{q}_i \mathrm{dt} \Big) \Big(\int_{\mathrm{t_f}}^{\mathrm{t}} \mathrm{F_u q_i dt} \Big)^{\intercal} \tag{19f}$$

$$Q_{\mathbf{vx}}(t) = \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{x}} \mathbf{q}_i \mathrm{dt} \Big) \Big(\int_{\mathrm{t_f}}^{\mathrm{t}} \mathrm{F_v q_i dt} \Big)^{\intercal} \tag{19g}$$

To avoid dimensionality issues and tensor representations, the integrations in equations (39) are broken down into each layer j of the network structure. We demonstrate the layer-wise break down for the $\mathbf{p}_i(t)$ vector, the rest are shown in detail in the Appendix.

$$\begin{aligned} \int_{t_f}^{t_0} \Big(F_{\mathbf{u}} \mathbf{q}_i \Big) \mathrm{dt} &= \Big[\dots, \int_{\mathrm{t_f}}^{\mathrm{t_0}} \Big(\mathrm{F_{u_j} q_i} \Big) \mathrm{dt}, \dots \Big] \\ &= \Big[\dots, \int_{t_f}^{t_0} \Big(\mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \Big) \mathrm{dt}, \dots \Big] \end{aligned} \tag{20}$$

We denote $\mathbf{z}^j$, $\mathbf{h}^j$, $\mathbf{u}^j$, and $\mathbf{v}^j$ as the activation, pre-activation (linear combination), and the parameters of layer j, respectively. We consider the the preactivation vector $\mathbf{h}_j(t)$, as an affine combination of the weights with the input to the j^{th} layer. At this point it is emphasized that we considered symmetric dynamics with respect to $\mathbf{u}$, and $\mathbf{v}$, resulting in that $\mathbf{h}^j_{\mathbf{u}} = \mathbf{h}^j_{\mathbf{v}} = \mathbf{z}_j$, and $\mathbf{h}^j_{\mathbf{x}} = (\mathbf{u} + \mathbf{v})$. This implies that:

$$F_{u_j} \mathbf{q}_i = \mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \text{ for feed-forward layers} \tag{21a}$$

$$F_{u_j} \mathbf{q}_i = \mathbf{z}_j \hat{*} (\frac{\partial F_{u_j}}{\partial \mathbf{h}_j} \mathbf{q}_i) \text{ for convolution layers} \tag{21b}$$

where $\otimes$ denotes the Kronecker product, and $\hat{*}$ denotes the de-convolution operator. The partial derivatives of the system dynamics with respect to the other variables $(\mathbf{x}, \mathbf{v})$, are mentioned in the Appendix. Finally, the layer-wise precondition matrices Q_{uu}, $Q_{\mathbf{uv}}$ and Q_{vv} at time $t = t_0$ can be approximated through the Kronecker factorization as shown in (22). To derive these expressions, we first use the Kronecker product property: $(A \otimes B)(C \otimes D) = AC \otimes BD$, and it is also assumed that $\mathbf{z}(t)_j$, and $\mathbf{g}(t)_j = \frac{\partial F}{\partial \mathbf{h}_j}$ are uncorrelated across time, and that $\mathbf{z}_j(t)$, $\mathbf{g}_j(t)$ are pair-wise independent. A discussion over this assumptions takes place in the Appendix.

$$\begin{aligned} &Q_{\mathbf{u}_j \mathbf{u}_j}(t_0) \approx R_{\mathbf{u}} I(t - t_f) + \\ &\int_{t_f}^{t_0} \underbrace{\Big(\mathbf{z}_j \mathbf{z}_j^{\intercal} \Big) \mathrm{dt}}_{A_j(t)} \otimes \int_{t_f}^{t_0} \underbrace{\sum_{i=1}^{R} \Big((\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \Big((\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \Big)^{\intercal} \mathrm{dt}}_{B_j(t)} \end{aligned} \tag{22a}$$

$$Q_{\mathbf{v}_j \mathbf{v}_j}(t_0) \approx R_{\mathbf{v}} I(t - t_f) + A_j(t) \otimes B_j(t) \tag{22b}$$

$$Q_{\mathbf{u}_j \mathbf{v}_j}(t_0) \approx A_j(t) \otimes B_j(t) \tag{22c}$$

Our motivation to exploit Kronecker products is to utilize product properties, such as: $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$, and $(A \otimes B)\mathrm{vec}(X) = \mathrm{vec}(BXA^{\intercal})$. Primarily, the first property enables the efficient preconditioning, where instead of inverting the Hessians Q_{uu}, and Q_{vv} of dimensions $mn \times mn$, instead we can invert two matrices of dimensions $m \times m$, and $n \times n$.

3.3. Efficient Approximation of Update Laws

Update Law from Min-Max DDP Recall that Neural ODE analysis and OCP principles are deeply intertwined. That motivates us to view the optimization process of the Neural ODEs as a trajectory optimization task. Following this, we employ the control variable update laws derived in the 2. The factorization from the previous section enable us to derive tractable update expressions for the antagonizing control variables, based on the min-max DDP update rule introduced in (8).

$$\begin{bmatrix} \delta \mathbf{u}_t \\ \delta \mathbf{v}_t \end{bmatrix} = \begin{bmatrix} l_{\mathbf{u}} \\ l\mathbf{v} \end{bmatrix} + \begin{bmatrix} K_{\mathbf{u}} \\ K\mathbf{v} \end{bmatrix} \delta \mathbf{x}_t, \tag{23}$$

where:

$$l_{\mathbf{u}} = \tilde{Q}_{\mathbf{uu}}^{-1}(Q_{\mathbf{uv}}Q_{\mathbf{vv}}^{-1}Q_{\mathbf{v}} - Q_{\mathbf{u}}), \text{ and}$$
$$K_{\mathbf{u}} = \tilde{Q}_{\mathbf{uu}}^{-1}(Q_{\mathbf{ux}} - Q_{\mathbf{uv}}Q_{\mathbf{vv}}^{-1}Q_{\mathbf{vx}}) \quad (24a)$$
$$l_{\mathbf{v}} = \tilde{Q}_{\mathbf{vv}}^{-1}(Q_{\mathbf{uv}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} - Q_{\mathbf{v}}), \text{ and}$$
$$K_{\mathbf{v}} = \tilde{Q}_{\mathbf{vv}}^{-1}(Q_{\mathbf{vx}} - Q_{\mathbf{uv}}Q_{\mathbf{vv}}^{-1}Q_{\mathbf{vx}}) \quad (24b)$$

with: $\tilde{Q}_{uu} = (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu})$, and $\tilde{Q}_{vv} = (Q_{vv} - Q_{vu}Q_{uu}^{-1}Q_{uv})$.

At this point, we examine two cases for the update law. The first one follows curvature approximation of the loss curve using the factorizations showed above to efficiently invert the Hessian. The second update rule constitutes a generalization following open loop Min-Max DDP, for which we will introduce two more propositions below in order to efficiently and compactly represent the terms in 24.

Curvature Approximation Initially, setting the cross-terms: $Q_{\mathbf{uu}} = Q_{\mathbf{vu}} = Q_{\mathbf{xu}} = Q_{\mathbf{xv}} = 0$, in 23 it is easy to see that the update rule for the two control variables takes the following form: $\mathbf{u} \leftarrow \mathbf{u} - \eta Q_{\mathbf{uu}}^{-1}\nabla_{\mathbf{u}}Q$, and $\mathbf{v} \leftarrow \mathbf{v} + \eta Q_{\mathbf{vv}}^{-1}\nabla_{\mathbf{v}}Q$, where $\eta \in (0, 1)$ is the step size or learning rate of the optimizer. Note that our analysis, we applied to the terminal Hessian $Q_{\mathbf{xx}}(t_f)$ the Gauss-Newton approximation, enabling our the Hessians $Q_{\mathbf{uu}}$, and $Q_{\mathbf{vv}}$ to be computed and approximated in a similar manner. This implies that the resulting precondition matrices can be viewed as an approximation of the Fisher Information Matrix, thus the resulting update law can be thought as following the Natural Gradient Descent (George et al., 2018).

Open-Loop DDP Update Law The second update rule is a generalization of the first rule based on the open loop update rule of Min-Max DDP. For this update law, we lift the assumption of $Q_{\mathbf{uv}}$ to be equal to zero so the l_u, and l_v terms become slightly more complex. Recall that by design we impose that $\ell_{\mathbf{uv}} = 0$, as suggested by OCP literature (Sun et al., 2018). The role of the cross term $Q_{\mathbf{uv}}$ is to capture the implicit interaction of the two weights. It expresses the coupling of the two control variables describing how the two control variables of the system are coupled to each other and how changes in one can affect the update of other.

Proposition 3.3. *Using the property of the eigenvalue decomposition of a Kronecker product:* $(A \otimes B + \lambda I)^{-1} = (U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B + \lambda)^{-1}(U_A \otimes U_B)^\intercal$*, to each of the matrix terms in* $\tilde{Q}_{uu}, \tilde{Q}_{\mathbf{vv}}$*, we derive an equivalent expression for their eigenvalue decomposition.*

$$\tilde{Q}_{\mathbf{uu}} = (U_A \otimes U_B)(S_{\mathbf{u}} - S_{\mathbf{uv}}S_{\mathbf{v}}^{-1}S_{\mathbf{uv}})(U_A \otimes U_B)^\intercal$$
$$\tilde{Q}_{\mathbf{vv}} = (U_A \otimes U_B)(S_{\mathbf{v}} - S_{\mathbf{uv}}S_{\mathbf{v}}^{-1}S_{\mathbf{uv}})(U_A \otimes U_B)^\intercal \quad (25)$$

The eigen-value decomposition in of $\tilde{Q}_{\mathbf{uu}}$, $\tilde{Q}_{\mathbf{vv}}$ help us view the resulting Kronecker-factored preconditioning as the scaled down by $(\Sigma_A \otimes \Sigma_B + \lambda)$ projection of the corresponding gradient vector ($Q_{\mathbf{u}}$, or $Q_{\mathbf{v}}$) along the eigenbasis (inner product with $(U_A \otimes U_B)$, scaled down that basis by $((U_A \otimes U_B)^\intercal)$ which finally returns to the initial parameter space, as demonstrated in (George et al., 2018).

Proposition 3.4. *After Eq. 25, the property:* $(A \otimes B)\mathrm{vec}(\mathrm{X}) = \mathrm{vec}(\mathrm{BXA}^\intercal)$ *allows us to rewrite the feed forward and feedback gains in 24 more compactly*

$$l_{\mathbf{u}} = \mathrm{vec}(\mathrm{U_B G_{uv} U_A^\intercal}) - \mathrm{vec}(\mathrm{U_B G_{uu} U_A^\intercal}),$$
$$K_{\mathbf{u}} = \mathrm{vec}(\mathrm{U_B Y_{uv} U_A^\intercal}) - \mathrm{vec}(\mathrm{U_B Y_{uu} U_A^\intercal}) \quad (26a)$$
$$l_{\mathbf{v}} = \mathrm{vec}(\mathrm{U_B G_{vu} U_A^\intercal}) - \mathrm{vec}(\mathrm{U_B G_{vv} U_A^\intercal}),$$
$$K_{\mathbf{v}} = \mathrm{vec}(\mathrm{U_B Y_{vu} U_A^\intercal}) - \mathrm{vec}(\mathrm{U_B Y_{vv} U_A^\intercal}) \quad (26b)$$

Following the property mentioned in the proposition , the only matrix inversions taking place involve the inversion of the diagonal matrices of the form $S = (\Sigma_A \otimes \Sigma_B + \lambda)$. Therefore, the computation of the terms in 26 is computationally tractable. The detailed proof is left in the Appendix A.1.

In this case, using the l terms from Eq. 26, our update law for the open-loop DDP GTSONO becomes $\mathbf{u}_t \leftarrow \mathbf{u}_t + \eta l_{\mathbf{u}_t}$, and $\mathbf{v}_t \leftarrow \mathbf{v}_t + \eta l_{\mathbf{v}_t}$, offering a generalization of the standard training scheme. From a Game Theoretic standpoint, the open loop information structure is equivalent to the two players (i.e. weights) having minimal access and knowledge to the game (i.e. network) structure.

4. Experiments

In this section, we assess the robustness of our suggested architecture compared with other state-of-the-art adversarial defense methods and the innate robustness of the continuous models. These models are tested in the task of image classification in 2 datasets. The tests performed examine the accuracy each model in the clean test set, along with the test set subjected to white box adversarial attacks and injection of Gaussian Noise. It is demonstrated that comparable performance to state-of-the-art adversarial training methods is achieved. We denote each model's accuracy to the clean test set (i.e. natural accuracy) with A_{nat}.

4.1. Experimental Setup

First, we introduce the datasets, the architectures and the white box attacks employed during our experiments. Note that generalized update rule following open loop Min-Max DDP is referred as DDP-GTSONO.

Attacks For the adversarial attacks, our emphasis lied on white box ℓ_∞-norm Projected Gradient Descent (PGD)

Table 1. Results on the MNIST. A_{nat} denotes each optimizer's accuracy to the clean test set (i.e. natural accuracy). $\mathcal{N}(0, \sigma)$ shows the robust accuracy of the optimizers after gaussian noise injection in the test set. PGD_s^ϵ denotes the accuracy of the optimizers under PGD attack in the test set with a perturbation distance ϵ that takes s steps in the direction of the gradient. $FGSM_s^\alpha$ describes the accuracy under FGSM attack whose single step in the direction of the gradient is multiplied with constant α.

Method	A_{nat}	$\mathcal{N}(0, 0.3)$	$\mathcal{N}(0, 0.4)$	$PGD_{40}^{0.2}$	$PGD_{40}^{0.3}$	$FGSM_{0.1}$	$FGSM_{0.2}$
Adam	97.54	97.05	96.88	90.92	85.44	94.62	91.44
SGD	98.00	97.42	96.85	90.04	85.19	95.48	92.12
SNOpt (Liu et al., 2021a)	97.67	97.23	97.11	91.34	87.04	94.68	91.36
DDP GTSONO (Ours)	99.25	98.91	98.67	93.88	93.26	95.38	91.33
GTSONO (Ours)	99.37	99.15	98.70	95.70	94.75	96.84	91.58

Table 2. Results on the SVHN

Method	A_{nat}	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.15)$	$PGD_{20}^{0.01}$	$PGD_{20}^{0.02}$	$FGSM_{0.02}$	$FGSM_{0.03}$
Adam	87.35	91.02	89.52	73.48	68.18	77.04	72.08
SGD	92.95	84.11	84.00	83.18	72.92	79.9	72.26
SNOpt (Liu et al., 2021a)	87.25	86.81	84.82	82.11	66.56	78.49	72.12
DDP GTSONO (Ours)	93.03	93.23	91.90	89.32	80.68	81.60	74.98
GTSONO (Ours)	94.17	93.41	92.39	92.92	82.08	82.72	74.64

(Madry et al., 2017) and Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014) to assess the robustness of our models. PGD attacks are an optimization-based attack used to undermine the estimations and predictions of machine learning models. It involves iteratively perturbing input data in the direction of the gradient, while constraining the perturbations to stay within a predefined projection set determined by parameter ϵ. For ℓ_∞ PGD attack, the update rule is defined as $\mathbf{x}' \leftarrow \Pi_{\mathbb{B}_\infty}(\mathbf{x}' + \eta_1 \text{ sign}(\nabla_{\mathbf{x}'}\mathcal{L}(F(\mathbf{x}'), y)))$. We denote PGD_s^ϵ the PGD attack with a perturbation distance ϵ that takes s steps in the direction of the gradient. Additionally, FGSM is another popular and computationally efficient attack technique. As opposed to taking multiple steps like PGD, FGSM operates by taking a single step in the direction of the gradient of the loss function with respect to the input. FGSM generates attacked sample through $\mathbf{x}' \leftarrow \mathbf{x} + \eta_2 \text{ sign}(\nabla_{\mathbf{x}}\mathcal{L}(F(\mathbf{x}), y)))$. We denote $FGSM_s^\alpha$ the FGSM attack whose single step in the direction of the gradient is multiplied with constant α. Finally, we also examined the robustness of our models, when Gaussian Noise was injected in the image. The perturbation of the images was carried out as follows: $\mathbf{x}' \leftarrow \mathbf{x} + \mathcal{N}(0, \sigma)$.

Datasets and training settings We carry out our experiments on two datasets the MNIST (Lecun et al., 1998), the SVHN (Netzer et al., 2011), with both datasets having 10 label classes. MNIST consists of 28×28 gray-scale images, while SVHN consists of 3×32×32 colour images.

The baseline optimizers used for our experiments are state-of-the-art optimizers widely used in Neural ODE applications. In particular, we consider Adam and SGD with momentum as first order benchmark optimizers, and SNOpt (Liu et al., 2021a) as a second order baseline. For training process of our proposed models, the training was carried out using only non-perturbed images for all optimizers. Additionally, the structure of the networks was identical for all experiments across all datasets and among every tested optimizer, with a batch size of 128 used for the training on both datasets. The networks trained on MNIST was trained for 5 epochs, while the network trained on SVHN was trained for 10 epochs. The ODE solver we used is the standard Runge-Kutta 4(5) adaptive solver (dopri5; (Dormand & Prince, 1980)) implemented by the torchdiffeq package, with the numerical tolerance set to 1e-3. All experiments are conducted on a TITAN RTX.

4.2. Results

Table 1 and 2 demonstrate the performance of our two methodologies compared with the benchmark optimizers on the two aforementioned datasets. In each Table, the performance of each optimizer in the clean test set is mentioned, along with the robust accuracy of each method under various attacks. Table 1 shows that all optimizers achieve near perfect accuracy on the MNIST clean test set, while our method outperforms the baseline optimizers under every attack and for all disturbances in the PGD attacks. We observe robust accuracy up to 94.75% for our optimizers under attacks with significant perturbation such as $PGD_{40}^{0.3}$. However, we can see that in the case of the FGSM attacks all classifiers have nearly identical performance. This could attributed to the

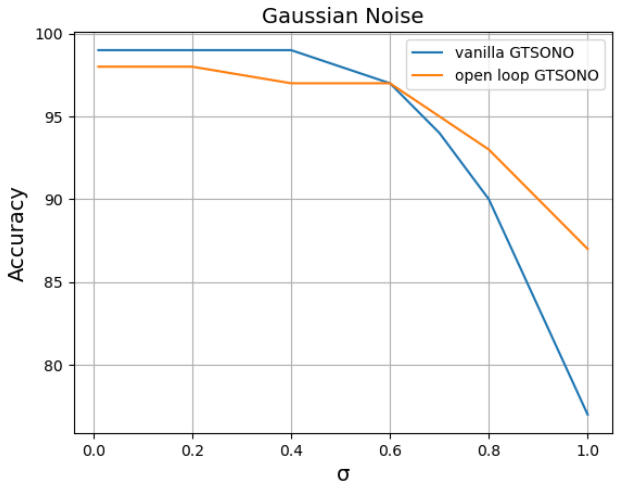

Figure 1. Robust Accuracy (%) of GTSONO and DDP-GTSONO for varying standard deviation of the injected Gaussian noise

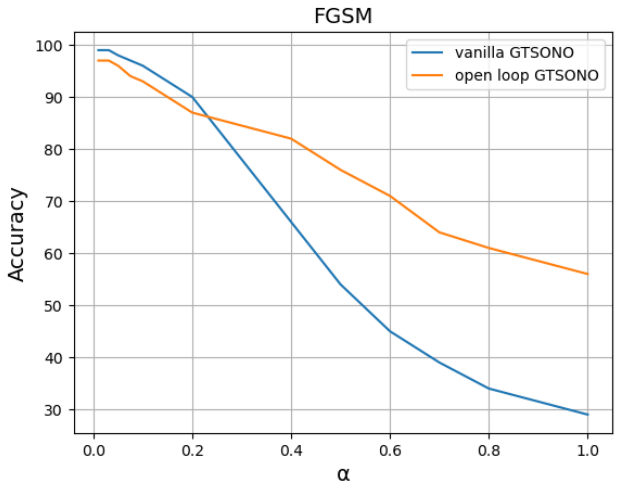

Figure 2. Robust Accuracy (%) of GTSONO and DDP-GTSONO for varying degree of pertubation generated from the FGSM attack

fact that while FGSM is a single step attack method not as potent as the iterative PGD, failing to effectively mislead the optimizers. Finally, regarding the efficiency of each optimizer their run-times per epoch on the MNIST dataset were comparable. More specifically, Adam and SGD completed an epoch in 13.6 and 12.2 seconds respectively. SNOpt required 15.5 seconds per epoch, while GTSONO required 15.7 seconds per epoch.

Similarly, we see that our proposed robust optimizers can significantly improve the robust accuracy of the model on the SVHN dataset. We observe robust accuracy up to almost 93% for PGD with $\epsilon = 0.02$, which is an outperforms almost by 10% the Adam optimizer, which is the more robust of the benchmark optimizers. It is shown that the performance gap widens as the disturbance in the test set increases under the PGD attack. Conversely for the FGSM attacks, we see that all optimizers have similar performance, verifying our comment about FGSM failing to generate perturbations that fool the optimizers. However, it is demonstrated that our optimizer still edges the rest optimizers delivering the best robust accuracy. Regarding the efficiency of each optimizer, it was observed that the run-times for each optimizer were comparable. Adam and SGD completed an epoch in 33.8 and 29.9 seconds respectively. SNOpt required 37.6 seconds per epoch, while GTSONO required 41.1 seconds per epoch.

4.3. Comparison between Open Loop and Vanilla GTSONO

We explore the robustness offered by the different update schemes of our methodology. Figures 1 and 2 demonstrate the accuracy of our methodology for varying degree of attack, under noise injection and FGSM attack on the MNIST dataset. We can see that for no-pertubation GTSONO edges DDP-GTSONO, which is aligned with what is demonstrated in Table 1, while this is also true for relatively small disturbances. However, we see that for large perturbations and severe noise injection, DDP-GTSONO is able to outperform its counterpart with a significant margin especially under the FGSM attack.

5. Conclusion

In this paper, we introduce a Game Theoretic Second Order Neural Optimizer. Drawing inspriration from OCP paradigms, such as min-max Differential Dynamic Programming, GTSONO showcases significant robustness against adversarial attacks, surpassing the performance of well-established optimizers like SGD and Adam. Experimental evaluations conducted in this study demonstrate the robustness of GTSONO against adversarial attacks, achieving higher accuracy rates and maintaining higher performance even when exposed to sophisticated adversarial techniques. Finally, two update rules are introduced for our optimizer. The first constitutes an approximation of Natural Gradient Descent, and a secondly a generalization of this update law based on the open loop Min-Max DDP. Our experiments showed that the additional terms in the latter capturing the interaction of the two control variables render the model more robust against large pertubations.

References

Carrara, F., Caldelli, R., Falchi, F., and Amato, G. On the robustness to adversarial examples of neural ode image classifiers. In *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6. IEEE, 2019.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Chu, H., Wei, S., and Zhao, Y. Towards natural robustness against adversarial examples. *arXiv preprint arXiv:2012.02452*, 2020.

Dormand, J. R. and Prince, P. J. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. ISSN 0377-0427. doi: https://doi.org/10.1016/0771-050X(80)90013-3. URL https://www.sciencedirect.com/science/article/pii/0771050X80900133.

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast approximate natural gradient descent in a kronecker factored eigenbasis. *Advances in Neural Information Processing Systems*, 31, 2018.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Huang, Y., Yu, Y., Zhang, H., Ma, Y., and Yao, Y. Adversarial robustness of stabilized neural ode might be from obfuscated gradients. In *Mathematical and Scientific Machine Learning*, pp. 497–515. PMLR, 2022.

Lecun, Y. A theoretical framework for back-propagation. In Touretzky, D., Hinton, G., and Sejnowski, T. (eds.), *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pp. 21–28. Morgan Kaufmann, 1988.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Liu, G.-H., Chen, T., and Theodorou, E. Second-order neural ode optimizer. *Advances in Neural Information Processing Systems*, 34:25267–25279, 2021a.

Liu, G.-H., Chen, T., and Theodorou, E. A. Ddpnopt: Differential dynamic programming neural optimizer, 2021b.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. How does noise help robustness? explanation and exploration under the neural sde framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 282–290, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Massaroli, S., Poli, M., Park, J., Yamashita, A., and Asama, H. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

Morimoto, J., Zeglin, G., and Atkeson, C. G. Minimax differential dynamic programming: Application to a biped walking robot. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pp. 1927–1932. IEEE, 2003.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

Sun, W., Theodorou, E. A., and Tsiotras, P. Game theoretic continuous time differential dynamic programming. In *2015 American control conference (ACC)*, pp. 5593–5598. IEEE, 2015.

Sun, W., Pan, Y., Lim, J., Theodorou, E. A., and Tsiotras, P. Min-max differential dynamic programming: Continuous and discrete time formulations. *Journal of Guidance, Control, and Dynamics*, 41(12):2568–2580, 2018.

Todorov, E. et al. Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, pp. 268–298, 2006.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

A. Appendix

A.1. Missing Derivations from Section 3

We recall the formulation of the objective of the Neural ODE framework expressd in a fashion easily interpretable through the prism of game theoretic OCP.

$$\min_{\mathbf{u}}\max_{\mathbf{v}}\left[\Phi(x_{t_f})+\int_{t_0}^{t_f}\ell(t,\mathbf{x},\mathbf{u},\mathbf{v})dt\right], \text{subjected to } \begin{cases}\frac{d\mathbf{x}}{dt}=F(t,\mathbf{x},\mathbf{u},\mathbf{v}), & \mathbf{x}(t_0)=x_0\\ \frac{d\mathbf{u}}{dt}=0, & \mathbf{u}(t_0)=\theta\\ \frac{d\mathbf{v}}{dt}=0, & \mathbf{v}(t_0)=\eta\end{cases} \tag{27}$$

, where $\mathbf{x}\equiv\mathbf{x}(t)\in\mathbb{R}^m$, $\mathbf{u}\equiv\mathbf{u}(t)\in\mathbb{R}^n$, and $\mathbf{v}\equiv\mathbf{v}(t)\in\mathbb{R}^n$. It is clear that (27) describes (10) without loss of generality by taking $(\Phi,\ell)=(\mathcal{L},0)$. The function $F(t,\mathbf{x},\mathbf{u},\mathbf{v})$ characterizes the vector field and is parameterized by a Deep Neural Network (DNN). We consider that the dynamics of the system are symetric with respect to the two sets of weights $(\mathbf{u},\mathbf{v})$. The functions Φ, and ℓ are known as the terminal and running cost in the context of OCP. This problem is understood as particular type of OCP that searches for the optimal initial condition of the time-invariant controls $\mathbf{u}_t,\mathbf{v}_t$. In the DNN setting, the terminal cost is equivalent to the loss function, for instance Categorical Cross Entropy for multi-label classification, and the running cost is equivalent to the weight decay, or some other regularization techniques that acts on the weights of the intermediate hidden layers. Next, the accumulated loss $Q(t,\mathbf{x},\mathbf{u},\mathbf{v})$ is defined as follows:

$$Q(t,\mathbf{x},\mathbf{u},\mathbf{v})=\Phi(\mathbf{x}(t_f))+\int_t^{t_f}\ell(\tau,\mathbf{x}(\tau),\mathbf{u}(\tau),\mathbf{v}(\tau))d\tau \tag{28}$$

From this definition of the accumulated loss implies that Q we can readily obtain:

$$0=\ell(t,\mathbf{x},\mathbf{u},\mathbf{v})+\frac{dQ(t,\mathbf{x},\mathbf{u},\mathbf{v})}{dt} \tag{29}$$

Proof of Theorem 3.1 Considering the nominal trajectory $(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}})$, that satisfies the ODEs in the constraints of (27), we take the Taylor expansion of the terms in Eq.29 keeping up to second order terms:

$$\ell(\mathbf{x},\mathbf{u},\mathbf{v})=\ell(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}})+\ell_{\mathbf{x}}\delta\mathbf{x}_t+\ell_{\mathbf{u}}\delta\mathbf{u}+\ell_{\mathbf{v}}\delta\mathbf{v}+\frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}\ell_{xx}&\ell_{xu}&\ell_{xv}\\ \ell_{ux}&\ell_{uu}&\ell_{uv}\\ \ell_{vx}&\ell_{vu}&\ell_{vv}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix} \tag{30a}$$

$$Q(\mathbf{x},\mathbf{u},\mathbf{v})=Q(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}})+Q_{\mathbf{x}}\delta\mathbf{x}_t+Q_{\mathbf{u}}\delta\mathbf{u}+Q_{\mathbf{v}}\delta\mathbf{v}+\frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}Q_{xx}&Q_{xu}&Q_{xv}\\ Q_{ux}&Q_{uu}&Q_{uv}\\ Q_{vx}&Q_{vu}&Q_{vv}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix} \tag{30b}$$

We differentiate 30b, in order to utilize Eq. 29, and taking into account that $\frac{d\delta\mathbf{u}}{dt}=\frac{\delta\mathbf{v}}{dt}=0$, we obtain:

$$\begin{aligned}\frac{dQ}{dt}\approx&\frac{dQ(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}})}{dt}+(\frac{dQ_{\mathbf{x}}}{dt}\delta\mathbf{x}_t+Q_{\mathbf{x}}\frac{d\delta\mathbf{x}}{dt})+(\frac{dQ_{\mathbf{u}}}{dt}\delta\mathbf{u}+Q_{\mathbf{u}}\frac{d\delta\mathbf{u}}{dt})+(\frac{dQ_{\mathbf{v}}}{dt}\delta\mathbf{v}+Q_{\mathbf{v}}\frac{d\delta\mathbf{v}}{dt})+\\&+\frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}\frac{dQ_{\mathbf{xx}}}{dt}&\frac{dQ_{xu}}{dt}&\frac{Q_{xv}}{dt}\\ \frac{dQ_{ux}}{dt}&\frac{dQ_{uu}}{dt}&\frac{Q_{uv}}{dt}\\ \frac{dQ_{vx}}{dt}&\frac{Q_{vu}}{dt}&\frac{Q_{vv}}{dt}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix}+\frac{1}{2}\begin{bmatrix}\frac{\delta\mathbf{x}_t}{dt}\\ 0\\ 0\end{bmatrix}^\intercal\begin{bmatrix}Q_{xx}&Q_{xu}&Q_{xv}\\ Q_{ux}&Q_{uu}&Q_{uv}\\ Q_{vx}&Q_{vu}&Q_{vv}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix}\\&+\frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\\ \delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}Q_{xx}&Q_{xu}&Q_{xv}\\ Q_{ux}&Q_{uu}&Q_{uv}\\ Q_{vx}&Q_{vu}&Q_{vv}\end{bmatrix}\begin{bmatrix}\frac{\delta\mathbf{x}_t}{dt}\\ 0\\ 0\end{bmatrix}\end{aligned} \tag{31}$$

Next, we need to compute the term $\frac{\delta\mathbf{x}}{dt}$:

$$\frac{d\delta\mathbf{x}}{dt}=\frac{d\mathbf{x}}{dt}-\frac{\bar{\mathbf{x}}}{dt}=F_{\mathbf{x}}\delta\mathbf{x}+F_{\mathbf{u}}\delta\mathbf{u}+F_{\mathbf{v}}\delta\mathbf{v} \tag{32}$$

The second equation comes from the fact that the time derivative evaluated for fixed $\bar{\mathbf{x}}$ yields the dynamics $F(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}})$, whereas for the first time derivative we take the Taylor expansion around the nominal trajectory. Finally, substituting Eq. 32

into Eq. 31, and back into 29, we obtain the following ODE for Q and its derivatives

$$-\frac{dQ_{\mathbf{x}}}{dt} = F_x Q_x + \ell_x, \quad -\frac{dQ_{xx}}{dt} = \ell_{xx} + F_x Q_{xx} + Q_{xx} F_x^\intercal, \quad -\frac{dQ_{\mathbf{xu}}}{dt} = \ell_{\mathbf{xu}} + F_{\mathbf{x}} Q_{\mathbf{xu}} + Q_{\mathbf{xx}} F_{\mathbf{u}}^\intercal \tag{33a}$$

$$-\frac{dQ_{\mathbf{u}}}{dt} = F_{\mathbf{u}} Q_{\mathbf{x}} + \ell_{\mathbf{u}}, \quad -\frac{dQ_{\mathbf{uu}}}{dt} = \ell_{\mathbf{uu}} + F_{\mathbf{u}} Q_{\mathbf{xu}} + Q_{\mathbf{ux}} F_{\mathbf{u}}^\intercal, \quad -\frac{dQ_{\mathbf{uv}}}{dt} = \ell_{\mathbf{uv}} + F_{\mathbf{u}} Q_{\mathbf{xv}} + Q_{\mathbf{ux}} F_{\mathbf{v}}^\intercal \tag{33b}$$

$$-\frac{dQ\mathbf{v}}{dt} = F_{\mathbf{v}} Q_{\mathbf{x}} + \ell\mathbf{v}, \quad -\frac{dQ_{\mathbf{vv}}}{dt} = \ell_{\mathbf{vv}} + F_{\mathbf{v}} Q_{\mathbf{xv}} + Q_{\mathbf{vx}} F)_{\mathbf{v}}^\intercal \quad -\frac{dQ_{\mathbf{xv}}}{dt} = \ell_{\mathbf{xv}} + F_{\mathbf{x}} Q_{\mathbf{xv}} + Q_{\mathbf{xx}} F_{\mathbf{v}}^\intercal \tag{33c}$$

Proof Theorem 3.2 We recall 3.2, where it was assumed the matrix $Q_{xx}(t_1)$ to be a symmetric matrix of rank $R \leq m$, it may be represented as: $Q_{xx} = \sum_{i=1}^{R} \mathbf{y}_i \mathbf{y}_i^\intercal$, where $\mathbf{y}_i \in \mathbb{R}^m$. Additionally, we had that $Q_{uu}(t_f) = 0$, $Q_{vv}(t_f) = 0$. Then $\forall t \in [t_0, t_f]$, the second order matrices in (33) that contain derivative with respect to the state can be decomposed as follows:

$$\begin{aligned}
Q_{\mathbf{xx}}(t) &= \sum_{i=1}^{R} \mathbf{q}_i(t)\mathbf{q}_i(t)^\intercal, & Q_{\mathbf{xu}}(t) &= \sum_{i=1}^{R} \mathbf{q}_i(t)\mathbf{p}_i(t)^\intercal, & Q_{\mathbf{xv}}(t) &= \sum_{i=1}^{R} \mathbf{q}_i(t)\mathbf{s}_i(t)^\intercal, \\
Q_{\mathbf{uu}}(t) &= \ell_{uu} + \sum_{i=1}^{R} \mathbf{p}_i(t)\mathbf{p}_i(t)^\intercal, & Q_{\mathbf{uv}}(t) &= \sum_{i=1}^{R} \mathbf{p}_i(t)\mathbf{s}_i(t)^\intercal, & Q_{\mathbf{vv}}(t) &= \ell_{vv} + \sum_{i=1}^{R} \mathbf{s}_i(t)\mathbf{s}_i(t)^\intercal,
\end{aligned} \tag{34}$$

Taking the ODEs from 33, and substituting in 34, we yield:

$$\begin{aligned}
\frac{-dQ_{uu}}{dt}(t) &= \ell_{uu} + F_u Q_{xu} + Q_{ux} F_u^\intercal = R_{\mathbf{u}} + F_u Q_{xu} + Q_{ux} F_u^\intercal \Rightarrow \\
\int_{t_f}^{t} \frac{-dQ_{uu}}{dt}(t)d\tau &= \int_{t_f}^{t} \left[I + F_u Q_{xu} + Q_{ux} F_u^\intercal\right] d\tau \implies \\
Q_{uu}(t) &= I(t - tf) + \int_{t_f}^{t} \sum_{i=1}^{R} \Big(\mathbf{p}_i (F_u^\intercal \mathbf{q}_i) + (F_u^\intercal \mathbf{q}_i) \otimes \mathbf{p}_i\Big) d\tau
\end{aligned} \tag{35}$$

$$\begin{aligned}
\frac{-dQ_{uu}}{dt}(t) &= \ell_{uu} + F_u Q_{xu} + Q_{ux} F_u^\intercal = R_{\mathbf{u}} + F_u Q_{xu} + Q_{ux} F_u^\intercal \Rightarrow \\
\int_{t_f}^{t} \frac{-dQ_{uu}}{dt}(t)d\tau &= \int_{t_f}^{t} \left[I + F_u Q_{xu} + Q_{ux} F_u^\intercal\right] d\tau \implies \\
Q_{uu}(t) &= I(t - tf) + \int_{t_f}^{t} \sum_{i=1}^{R} \Big(\mathbf{p}_i (F_u^\intercal \mathbf{q}_i) + (F_u^\intercal \mathbf{q}_i) \otimes \mathbf{p}_i\Big) d\tau
\end{aligned} \tag{36}$$

Therefore, we define $Q_{uu}(t) = R_u I(t - tf) + \sum_{i=1}^{R} \mathbf{p}_i \otimes \mathbf{p}_i$, from which we can readily obtain that:

$$-\frac{d\mathbf{p}_i(t)}{dt} = F_{\mathbf{u}}^\intercal \mathbf{q}_i(t) \tag{37}$$

Similarly, we define $Q_{vv} = R_v I(t - tf) + \sum_{i=1}^{R} \mathbf{s}_i \otimes \mathbf{s}_i$, we obtain for $\frac{d\mathbf{s}_i(t)}{dt}$:

$$-\frac{d\mathbf{s}_i(t)}{dt} = F_{\mathbf{v}}^\intercal \mathbf{q}_i(t) \tag{38}$$

The second order matrices can be rewritten as follows:

$$Q_{\mathbf{uu}}(t) = R_{\mathbf{u}} I(t-t_f) + \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{u}} \mathbf{q}_i dt \Big) \Big(\int_{t_f}^{t} F_{\mathbf{u}} \mathbf{q}_i dt \Big)^{\intercal} \tag{39a}$$

$$Q_{\mathbf{vv}}(t) = R_{v} I(t-t_f) + \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{v}} \mathbf{q}_i dt \Big) \Big(\int_{t_f}^{t} F_{\mathbf{v}} \mathbf{q}_i dt \Big)^{\intercal} \tag{39b}$$

$$Q_{\mathbf{uv}}(t) = \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{u}} \mathbf{q}_i dt \Big) \Big(\int_{t_f}^{t} F_{\mathbf{v}} \mathbf{q}_i dt \Big)^{\intercal} \tag{39c}$$

$$Q_{\mathbf{ux}}(t) = \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{x}} \mathbf{q}_i dt \Big) \Big(\int_{t_f}^{t} F_{\mathbf{u}} \mathbf{q}_i dt \Big)^{\intercal}, Q_{\mathbf{vx}}(t) = \sum_{i=1}^{R} \Big(\int_{t_f}^{t} F_{\mathbf{x}} \mathbf{q}_i dt \Big) \Big(\int_{t_f}^{t} F_{\mathbf{v}} \mathbf{q}_i dt \Big)^{\intercal} \tag{39d}$$

We denote $\mathbf{z}^j$, $\mathbf{h}^j$, $\mathbf{u}^j$, and $\mathbf{v}^j$ as the activation, pre-activation (linear combination), and the parameters of layer j, respectively. Furthermore, we consider the the preactivation vector $\mathbf{h}_j(t)$, as an affine combination of the weights with the input to the j^{th} layer. We recall the symmetricity of our dynamics with respect to $\mathbf{u}$, and $\mathbf{v}$, resulting in the partial derivatives of the preactivation vector at the j^{th} layer with respect to the control variables: $\mathbf{h}^j_{\mathbf{u}} = \mathbf{h}^j_{\mathbf{v}} = \mathbf{z}_j$, and $\mathbf{h}^j_{\mathbf{x}} = (\mathbf{u} + \mathbf{v})$. This implies that:

$$F_{\mathbf{x}_j} \mathbf{q}_i = (u+v)^{\intercal} (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \text{ for Feed Forward layers} \tag{40a}$$

$$F_{\mathbf{u}_j} \mathbf{q}_i == \frac{\partial F}{\partial \mathbf{h}_j} (\mathbf{z}_j \otimes I) \mathbf{q}_i = \mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \text{ for Feed Forward layers} \tag{40b}$$

$$F_{\mathbf{x}_j} \mathbf{q}_i = (u+v) \hat{*} (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \text{ for the Convolution layers} \tag{40c}$$

$$F_{\mathbf{u}_j} \mathbf{q}_i = \mathbf{z}_j \hat{*} (\frac{\partial F_{u_j}}{\partial \mathbf{h}_j} \mathbf{q}_i) \text{ for the Convolution layers} \tag{40d}$$

where $\otimes$ denotes the Kronecker product, and $\hat{*}$ denotes the de-convolution operator. The layer-wise notation used above was adopted as a manner to circumvent dimensionality and tensor representations issues. In this vein, the integrations in equations (39) are broken down into each layer j of the network structure, where using the expression from 40, we obtain

$$\int_{t_f}^{t_0} \Big(F_{\mathbf{x}} \mathbf{q}_i \Big) dt = \Big[\dots, \int_{t_f}^{t_0} \Big(F_{\mathbf{x}^n} \mathbf{q}_i \Big) dt, \dots \Big] = \Big[\dots, \int_{t_f}^{t_0} \Big((\mathbf{u}+\mathbf{v})^{\intercal} (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \Big) dt, \dots \Big] \tag{41a}$$

$$\int_{t_f}^{t_0} \Big(F_{\mathbf{u}} \mathbf{q}_i \Big) dt = \Big[\dots, \int_{t_f}^{t_0} \Big(F_{u^n} \mathbf{q}_i \Big) dt, \dots \Big] = \Big[\dots, \int_{t_f}^{t_0} \Big(\mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \Big) dt, \dots \Big] \tag{41b}$$

$$\int_{t_f}^{t_0} \Big(F_{\mathbf{v}} \mathbf{q}_i \Big) dt = \Big[\dots, \int_{t_f}^{t_0} \Big(F_{v^n} \mathbf{q}_i \Big) dt, \dots \Big] = \Big[\dots, \int_{t_f}^{t_0} \Big(\mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j} \mathbf{q}_i) \Big) dt, \dots \Big] \tag{41c}$$

Derivation of Equation 22 Following the layer-wise representation, We recall that to derive 22 these expressions, we first use the Kronecker product property: $(A \otimes B)(C \otimes D) = AC \otimes BD$. Additionally, during the process of this derivation some approximations are necessary. First it is assumed that $\mathbf{z}(t)_j$, and $\mathbf{g}(t)_j = \frac{\partial F}{\partial \mathbf{h}_j}$ are uncorrelated across time, which is a rather strong assumption. Secondly, the last of our derivation entails an approximation that $\mathbf{z}_j(t)$, $\mathbf{g}_j(t)$ are pair-wise independent (Liu et al. 2021 NIPS). For the sake of brevity we demonstrate the analytic proof only for the Hessian $Q_{\mathbf{uu}}$, the

others follow similarly.

$$
\begin{aligned}
Q_{\mathbf{u}_j\mathbf{u}_j}(t_0) &= R_{\mathbf{u}} I(t-t_f) + \sum_{i=1}^{R}\Big(\int_{t_f}^{t} F_{\mathbf{u}}\mathbf{q}_i dt\Big)\Big(\int_{t_f}^{t} F_{\mathbf{u}}\mathbf{q}_i dt\Big)^{\intercal} \\
&= R_{\mathbf{u}} I(t-t_f) + \sum_{i=1}^{R}\Big(\int_{t_f}^{t} \mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i) dt\Big)\Big(\int_{t_f}^{t} \mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i) dt\Big)^{\intercal} \\
&\approx R_{\mathbf{u}} I(t-t_f) + \sum_{i=1}^{R}\int_{t_f}^{t}\Big(\mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i)\Big)\Big(\mathbf{z}_j \otimes (\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i) dt\Big)^{\intercal} \\
&= R_{\mathbf{u}} I(t-t_f) + \sum_{i=1}^{R}\int_{t_f}^{t}\Big(\mathbf{z}_j\mathbf{z}_j^{\intercal} \otimes (\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i)(\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i)^{\intercal}\Big) dt \\
&\approx R_{\mathbf{u}} I(t-t_f) + \int_{t_f}^{t_0} \underbrace{\Big(\mathbf{z}_j\mathbf{z}_j^{\intercal}\Big) dt}_{A_j(t)} \otimes \int_{t_f}^{t_0} \underbrace{\sum_{i=1}^{R}\Big((\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i)(\frac{\partial F}{\partial \mathbf{h}_j}\mathbf{q}_i)^{\intercal}\Big)}_{B_j(t)}
\end{aligned}
\tag{42}
$$

Remark The first assumption while maybe deemed as a strong assumption is almost essential in order to yield Kronecker factorized matrices (Liu et al. 2021). While relaxation of this assumption is possible via graphical models at the expense of computational cost (Martens et al. 2018), empirically the uncorrelated assumption has been shown to yield better performance (Laurent et al. 2018). The second approximation has been verified in (Wu et al. 2020) through an empirical study and can be made exact under certain conditions (Martens & Grosse 2015).

Derivation of Proposition 3.3 We begin by trying to simplifying the expressions for Q_{uu}, and Q_{vv}. Setting: $(\Sigma_A \otimes \Sigma_B) = S_{uv}$, $(\Sigma_A \otimes \Sigma_B + \lambda_u) = S_u$, and $(\Sigma_A \otimes \Sigma_B + \lambda_v) = S_v$, we obtain:

$$
\begin{aligned}
\tilde{Q}_{uu} &= (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu}) \\
&= (U_A \otimes U_B)S_u(U_A \otimes U_B)^{\intercal} - (U_A \otimes U_B)S_{uv}(U_A \otimes U_B)^{\intercal} \\
&(U_A \otimes U_B)S_v^{-1}(U_A \otimes U_B)^{\intercal}(U_A \otimes U_B)S_{uv}(U_A \otimes U_B)^{\intercal} \\
&= (U_A \otimes U_B)S_u(U_A \otimes U_B)^{\intercal} - (U_A \otimes U_B)(S_{uv}S_v^{-1}S_{uv})(U_A \otimes U_B)^{\intercal} \\
&= (U_A \otimes U_B)(S_u - S_{uv}S_v^{-1}S_{uv})(U_A \otimes U_B)^{\intercal}
\end{aligned}
\tag{43}
$$

Similarly, for $\tilde{Q}_{\mathbf{vv}} = (U_A \otimes U_B)(S_v - S_{uv}S_u^{-1}S_{uv})(U_A \otimes U_B)^{\intercal}$. At this point, using the following properties of the Kronecker product, based on the eigenvalue decomposition:

$$
(A \otimes B + \lambda I) = (U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B + \lambda)(U_A \otimes U_B)^{\intercal} \tag{44}
$$

$$
(A \otimes B + \lambda I)^{-1} = (U_A \otimes U_B)(\Sigma_A \otimes \Sigma_B + \lambda)^{-1}(U_A \otimes U_B)^{\intercal} \tag{45}
$$

we can easily obtain the inverse of the matrices in 43 as follows:

$$
\begin{aligned}
\tilde{Q}_{uu}^{-1} &= (U_A \otimes U_B)(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}(U_A \otimes U_B)^{\intercal}, \\
\tilde{Q}_{\mathbf{vv}}^{-1} &= (U_A \otimes U_B)(S_v - S_{uv}S_u^{-1}S_{uv})^{-1}(U_A \otimes U_B)^{\intercal}.
\end{aligned}
$$

Derivation of Proposition 3.4 Now, substituting into 23 the expression for 25, this formulation along with the property in $(A \otimes B)vec(X) = vec(BXA^{\intercal})$, and setting for brevity: $Q_{\mathbf{ux}}\delta\mathbf{x}_t = \delta\mathbf{q}_{\mathbf{u}}$, $Q_{\mathbf{vx}}\delta\mathbf{x}_t = \delta\mathbf{q}_{\mathbf{v}}$, the feed-forward and feedback

gains in (24) can be compactly rewritten as follows

$$l_{\mathbf{u}} = (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}(S_{uv}S_v^{-1})\text{vec}(U_B^\intercal \bar{Q}_{\mathbf{v}} U_A)}_{\mathbf{g}_{uv}} - (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}\text{vec}(U_B^\intercal \bar{Q}_{\mathbf{u}} U_A)}_{\mathbf{g}_{uu}} \tag{46a}$$

$$= \text{vec}(U_B G_{uv} U_A^\intercal) - \text{vec}(U_B G_{uu} U_A^\intercal) \tag{46b}$$

$$l_{\mathbf{v}} = (U_A \otimes U_B) \underbrace{(S_v - S_{uv}S_u^{-1}S_{uv})^{-1}(S_{uv}S_u^{-1})\text{vec}(U_B^\intercal \bar{Q}_{\mathbf{u}} U_A)}_{\mathbf{g}_{vu}} - (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}\text{vec}(U_B^\intercal \bar{Q}_{\mathbf{v}} U_A)}_{\mathbf{g}_{vv}} \tag{46c}$$

$$= \text{vec}(U_B G_{vu} U_A^\intercal) - \text{vec}(U_B G_{vv} U_A^\intercal) \tag{46d}$$

$$K_{\mathbf{u}} = (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}(S_{uv}S_v^{-1})\delta\mathbf{q}_{\mathbf{v}}}_{\mathbf{y}_{vv}} - (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}\delta\mathbf{q}_{\mathbf{u}}}_{\mathbf{y}_{vv}} \tag{46e}$$

$$= \text{vec}(U_B Y_{uv} U_A^\intercal) - \text{vec}(U_B Y_{uu} U_A^\intercal) \tag{46f}$$

$$K_{\mathbf{v}} = (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}(S_{uv}S_v^{-1})\delta\mathbf{q}_{\mathbf{u}}}_{\mathbf{y}_{vv}} - (U_A \otimes U_B) \underbrace{(S_u - S_{uv}S_v^{-1}S_{uv})^{-1}\delta\mathbf{q}_{\mathbf{v}}}_{\mathbf{y}_{vv}} \tag{46g}$$

$$= \text{vec}(U_B Y_{vu} U_A^\intercal) - \text{vec}(U_B Y_{vv} U_A^\intercal)$$

where $\text{vec}^{-1}(Q_i) = \bar{Q}_i, \quad \text{vec}^{-1}(g_{ij}) = G_{ij}, \quad \text{and} \quad \text{vec}^{-1}(y_{ij}) = Y_{ij} \quad i = \{u, v\}$, denote the inverse of the vectorization operation of the corresponding vectors. Inverting the diagonal S matrices is tractable, rendering the computation of the terms in 46 tractable to compute.

B. Differential Dynamic Programming in Continuous Time

B.1. Problem Formulation

Consider a min-max game problem, with dynamics described the following ODE:

$$\frac{d\mathbf{x}(t)}{dt} = F(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t), \quad x(t_0) = x_0 \tag{47}$$

where $x(t) \in \mathcal{X}$ is the state of the dynamic system at $t \in [t_0, t_f]$, and $\mathbf{u}(t) \in U_1 \subset \mathcal{U}$ and $\mathbf{v}(t) \in U_2 \subset \mathcal{V}$ denote conflicting controls, with U_1 and U_2 are convex sets containing all admissible controls of $\mathbf{u}$ and $\mathbf{v}$ respectively. Our goal is to find non-anticipating strategies for both players, namely we wish to maps: $\gamma_u : [t_0, t_f] \times \mathcal{X} \to U_1$ and $\gamma_v : [t_0, t_f] \times \mathcal{X} \to U_2$ for the two players respectively. Additionally, we define the cost function as follows:

$$J(\gamma_u, \gamma_v) = \phi(t_f, x(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau \tag{48}$$

where $\phi : [t_0, t_f] \times \mathcal{X} \to \mathbb{R}_+$ is the terminal cost, and $\mathcal{L} : \mathcal{X} \times \mathcal{U} \times \mathcal{V} \times [t_0, t_f] \to \mathbb{R}_+$ is the running cost incorporating the state and the control cost for both players. The conflict between the two players enters in our problem formulation as one tries through control $\mathbf{u}$ to minimize the above cost function, whereas the other one tries to maximize it through control $\mathbf{v}$.

We proceed to define the value function for our problem as the function expressing the minmax value of the cost function at time $t = t_0$ and $x = x_0$.

$$V(t_0, x_0) = \min_{\gamma_u} \max_{\gamma_v} \left\{ \phi(t_f, \mathbf{x}(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau \right\} \tag{49}$$

Using the Bellman principle, we write (49), as follows:

$$\begin{aligned} V(t, x(t)) &= \min_{\gamma_u(t \to t_f)} \max_{\gamma_v(t \to t_f)} \left\{ \phi(t_f, \mathbf{x}(t_f)) + \int_{t}^{t+dt} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau \right. \\ &\qquad \left. + \int_{t+dt}^{t_f} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau \right\} \\ &= \min_{\gamma_u(t \to t+dt)} \max_{\gamma_v(t \to t+dt)} \left\{ \min_{\gamma_u(t+dt \to t_f)} \max_{\gamma_v(t+dt \to t_f)} \left\{ \phi(t_f, \mathbf{x}(t_f)) + \int_{t+dt}^{t_f} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau \right\} \right. \\ &\qquad \left. + \int_{t}^{t+dt} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau \right\} \\ &= \min_{\gamma_u(t \to t+dt)} \max_{\gamma_v(t \to t+dt)} \left\{ \int_{t}^{t+dt} \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) d\tau + V(t + dt, \mathbf{x}(t + dt)) \right\} \end{aligned} \tag{50}$$

Then we can readily obtain:

$$0 = \min_{\gamma_u(t)} \max_{\gamma_v(t)} \Big[\mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) + \frac{d\mathcal{V}}{dt} \Big] \tag{51}$$

We can express dV as follows:

$$dV \approx \frac{\partial \mathcal{V}}{\partial t} dt + \frac{\partial \mathcal{V}}{\partial \mathbf{x}_t}^{\intercal} d\mathbf{x}_t = \frac{\partial \mathcal{V}}{\partial t} dt + \frac{\partial \mathcal{V}}{\partial \mathbf{x}_t}^{\intercal} F dt \tag{52}$$

This form enables us to obtain the min-max Hamilton Jacobi Bellman through substitution to (51):

$$\begin{aligned} 0 &= \min_{\gamma_u(t)} \max_{\gamma_v(t)} \Big[\mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) + \frac{\partial \mathcal{V}}{\partial t} + \frac{\partial \mathcal{V}}{\partial \mathbf{x}_t}^{\intercal} F \Big] \Rightarrow \\ -\frac{\partial \mathcal{V}}{\partial t} &= \min_{\gamma_u(t)} \max_{\gamma_v(t)} \Big[\mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) + \frac{\partial \mathcal{V}}{\partial \mathbf{x}_t}^{\intercal} F \Big] \end{aligned} \tag{53}$$

with its terminal condition being: $V(t_f, \mathbf{x}(t_f)) = \phi(t_f, \mathbf{x}(t_f))$.

B.2. Backwards Propagation

First, we consider equation function Q as: $Q(\mathbf{x}, \mathbf{u}, \mathbf{v}, t) = \mathcal{L}(\mathbf{x}(t), \gamma_u(t, \mathbf{x}(t)), \gamma_v(t, \mathbf{x}(t)), t) + \frac{d\mathcal{V}}{dt}$. We set $\gamma_u(t, \mathbf{x}(t) = \mathbf{u}(t)$, and $\gamma_v(t, \mathbf{x}(t) = \mathbf{v}(t)$ and expand the terms inside the minimization in (51) along with function Q up to second order terms, with respect to the nomimal trajectory $(\bar{\mathbf{x}}(t), \bar{\mathbf{u}}(t), \bar{\mathbf{v}}(t))$ and we obtain:

$$\begin{aligned} &\mathcal{L}(\bar{\mathbf{x}}(t) + \delta\mathbf{x}_t, \bar{\mathbf{u}}(t) + \delta\mathbf{u}_t, \bar{\mathbf{v}} + \delta\mathbf{v}_t) = \\ &\mathcal{L}(\bar{\mathbf{x}}(t), \bar{\mathbf{u}}(t), \bar{\mathbf{v}}(t)) + \mathcal{L}_x^{\intercal} \delta\mathbf{x} + \mathcal{L}_u^{\intercal} \delta\mathbf{u} + \mathcal{L}_v^{\intercal} \delta\mathbf{v}_t + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \\ \delta\mathbf{v}_t \end{bmatrix}^{\intercal} \begin{bmatrix} \mathcal{L}_{xx} & \mathcal{L}_{xu} & \mathcal{L}_{xv} \\ \mathcal{L}_{ux} & \mathcal{L}_{uu} & \mathcal{L}_{uv} \\ \mathcal{L}_{vx} & \mathcal{L}_{vu} & \mathcal{L}_{vv} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \\ \delta\mathbf{v}_t \end{bmatrix} \end{aligned} \tag{54}$$

$$\mathcal{V}(t + dt, \bar{\mathbf{x}}(t) + \delta\mathbf{x}_t) = \mathcal{V}(t, \bar{\mathbf{x}}(t) + \mathcal{V}_x(t, \mathbf{x}(t))\delta\mathbf{x}_t + \frac{1}{2} \delta\mathbf{x}_t^{\intercal} \mathcal{V}_{xx} \delta\mathbf{x}_t \tag{55}$$

At this point, we take the derivative with respect to time, of the expanded expression of the value function from (23). We note that $\mathcal{V}, \mathcal{V}_x, \mathcal{V}_{xx}$ are only functions of time, since they are expanded with respect to the nominal $\bar{\mathbf{x}}(t)$. Additionally, we also notice that from the system dynamics the infinitesimal perturbation around the nominal trajectory $\delta\mathbf{x}_t$ are also a function of time, so the differentiation of the second and third term below follow the product differentiation rule.

$$\begin{aligned}\frac{d\mathcal{V}}{dt} &= \frac{d\mathcal{V}}{dt}\Big|_{\substack{x=\bar{\mathbf{x}}(t)\\u=\bar{\mathbf{u}}(t)\\\mathbf{v}=\bar{\mathbf{v}}(t)}} + \frac{d}{dt}\Big(\mathcal{V}_x(t,\bar{\mathbf{x}}(t))\delta\mathbf{x}_t\Big) + \frac{d}{dt}\Big(\frac{1}{2}\delta\mathbf{x}^\intercal\mathcal{V}_{xx}\delta\mathbf{x}_t\Big)\\ &= \frac{d\mathcal{V}}{dt} + \Big(\frac{d\mathcal{V}}{dt}^\intercal\delta\mathbf{x}_t + \mathcal{V}_x^\intercal\frac{d\delta\mathbf{x}_t}{dt}\Big) + \frac{1}{2}\Big(\frac{d\delta\mathbf{x}}{dt}^\intercal\mathcal{V}_{xx}\delta\mathbf{x}_t + \delta\mathbf{x}_t^\intercal\frac{d\mathcal{V}_{xx}}{dt}\delta\mathbf{x}_t + \delta\mathbf{x}_t^\intercal\mathcal{V}_{xx}\frac{d\delta\mathbf{x}_t}{dt}\Big)\end{aligned} \tag{56}$$

Returning to the system dynamics, we can write $\frac{d\delta\mathbf{x}}{dt}$, as follows:

$$\begin{aligned}\frac{d\delta\mathbf{x}}{dt} &= \frac{d}{dt}\Big(\mathbf{x}(t) - \bar{\mathbf{x}}(t)\Big) = F(\mathbf{x}(t),\mathbf{u}(t),\mathbf{v}(t),t)) - F(\bar{\mathbf{x}}(t),\bar{\mathbf{u}}(t),\bar{\mathbf{v}}(t),t)\\ &= F(\bar{\mathbf{x}}(t),\bar{\mathbf{u}}(t),\bar{\mathbf{v}}(t),t) + \bar{F}_x\delta\mathbf{x}_t + \bar{F}_u\delta\mathbf{u}_t + \bar{F}_v\delta\mathbf{v}_t - F(\bar{\mathbf{x}}(t),\bar{\mathbf{u}}(t),\bar{\mathbf{v}}(t),t)\\ &= \bar{F}_x\delta\mathbf{x}_t + \bar{F}_u\delta\mathbf{u}_t + \bar{F}_v\delta\mathbf{v}_t\end{aligned} \tag{57}$$

where $\bar{F}_x, \bar{F}_u, \bar{F}_v$ stands for $F_x(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}}), F_u(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}}), F_v(\bar{\mathbf{x}},\bar{\mathbf{u}},\bar{\mathbf{v}})$ respectively. If we substitute (57) in (56), we obtain:

$$\begin{aligned}\frac{d\mathcal{V}}{dt} = \frac{d\mathcal{V}}{dt}\Big|_{\substack{x=\bar{\mathbf{x}}(t)\\u=\bar{\mathbf{u}}(t)\\\mathbf{v}=\bar{\mathbf{v}}(t)}} &+ \Big(\frac{d\mathcal{V}}{dt}^\intercal\delta\mathbf{x}_t + \mathcal{V}_x^\intercal(\bar{F}_x\delta\mathbf{x}_t + \bar{F}_u\delta\mathbf{u}_t + \bar{F}_v\delta\mathbf{v}_t)\Big)\\ &+ \frac{1}{2}\Big(((\bar{F}_x\delta\mathbf{x}_t + \bar{F}_u\delta\mathbf{u}_t + \bar{F}_v\delta\mathbf{v}_t))^\intercal\mathcal{V}_{xx}\delta\mathbf{x}_t + \delta\mathbf{x}_t^\intercal\frac{d\mathcal{V}_{xx}}{dt}\delta\mathbf{x}_t + \delta\mathbf{x}_t^\intercal\mathcal{V}_{xx}((\bar{F}_x\delta\mathbf{x}_t + \bar{F}_u\delta\mathbf{u}_t + \bar{F}_v\delta\mathbf{v}_t))\Big)\end{aligned} \tag{58}$$

Now, if we substitute (58) and (54) in (51), separate the terms that differentiate with respect to time the value function or any of its derivatives with respect to the state, we obtain:

$$\begin{aligned}&-\frac{d\mathcal{V}}{dt} - \frac{d\mathcal{V}_x}{dt}^\intercal\delta\mathbf{x}_t - \frac{1}{2}\delta\mathbf{x}_t^\intercal\frac{d\mathcal{V}_{xx}}{dt}\delta\mathbf{x}_t =\\ &= \min_{\delta\mathbf{u}(t)}\max_{\delta\mathbf{v}(t)}\Big[\mathcal{L}(\bar{\mathbf{x}}(t),\bar{\mathbf{u}}(t),\bar{\mathbf{v}}(t)) + \mathcal{L}_x^\intercal\delta\mathbf{x} + \mathcal{L}_u^\intercal\delta\mathbf{u} + \mathcal{L}_v^\intercal\delta\mathbf{v} + \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}\mathcal{L}_{xx} & \mathcal{L}_{xu} & \mathcal{L}_{xv}\\ \mathcal{L}_{ux} & \mathcal{L}_{uu} & \mathcal{L}_{uv}\\ \mathcal{L}_{vx} & \mathcal{L}_{vu} & \mathcal{L}_{vv}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}\\ &+ \mathcal{V}_x^\intercal F_x\delta\mathbf{x}_t + \mathcal{V}_x^\intercal F_u\delta\mathbf{u}_t + \mathcal{V}_x^\intercal F_v\delta\mathbf{v}_t + \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}\bar{F}_x^\intercal V_{xx} + V_{xx}\bar{F}_x & V_{xx}\bar{F}_u & V_{xx}\bar{F}_v\\ \bar{F}_u^\intercal V_{xx} & 0 & 0\\ \bar{F}_v^\intercal V_{xx} & 0 & 0\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}\Big]\end{aligned} \tag{59}$$

$$\begin{aligned}&= \min_{\delta\mathbf{u}(t)}\max_{\delta\mathbf{v}(t)}\Big[\mathcal{L}(\bar{\mathbf{x}}(t),\bar{\mathbf{u}}(t)) + (\mathcal{L}_x^\intercal + \mathcal{V}_x^\intercal F_x)\delta\mathbf{x}_t + (\mathcal{L}_u^\intercal + \mathcal{V}_x^\intercal F_u)\delta\mathbf{u}_t + (\mathcal{L}_v^\intercal + \mathcal{V}_x^\intercal F_v)\delta\mathbf{v}_t\\ &+ \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}\mathcal{L}_{xx} + \bar{F}_x^\intercal V_{xx} + V_{xx}\bar{F}_x & \mathcal{L}_{xu} + V_{xx}\bar{F}_u & V_{xx}\bar{F}_v\\ \mathcal{L}_{ux} + \bar{F}_u^\intercal V_{xx} & \mathcal{L}_{uu} & \mathcal{L}_{uv}\\ \mathcal{L}_{vx} + \bar{F}_v^\intercal V_{xx} & \mathcal{L}_{vu} & \mathcal{L}_{vv}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}\Big]\end{aligned}$$

Following the definition of Q and (51), we can expand Q up to second order terms as following:

$$0 = \min_{\gamma_u(t)}\max_{\gamma_v(t)}\Big[Q(\bar{x},\bar{u},\bar{v},t) + Q_x^\intercal\delta\mathbf{x} + Q_u^\intercal\delta\mathbf{u}_t + Q_v^\intercal\delta\mathbf{v}_t + \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix}^\intercal\begin{bmatrix}Q_{xx} & Q_{xu} & Q_{xv}\\ Q_{ux} & Q_{uu} & Q_{uv}\\ Q_{vx} & Q_{vu} & Q_{vv}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}_t\\\delta\mathbf{u}_t\\\delta\mathbf{v}_t\end{bmatrix} \tag{60}$$

Therefore, we can equate the terms from (59) with the terms of the quadratic expansion of (60), and yield the following:

$$
\begin{aligned}
Q_0(t) &= \mathcal{L}(\bar{\mathbf{x}}(t), \bar{\mathbf{u}}(t), t) \\
Q_x(t) &= (\mathcal{L}_x^\intercal + \mathcal{V}_x^\intercal F_x)^\intercal = \mathcal{L}_x + F_x^\intercal \mathcal{V}_x \\
Q_u(t) &= (\mathcal{L}_u^\intercal + \mathcal{V}_x^\intercal F_u)^\intercal = \mathcal{L}_u + F_u^\intercal \mathcal{V}_x \\
Q_v(t) &= (\mathcal{L}_v^\intercal + \mathcal{V}_x^\intercal F_v)^\intercal = \mathcal{L}_v + F_v^\intercal \mathcal{V}_x \\
Q_{xx}(t) &= \mathcal{L}_{xx} + \mathcal{V}_{xx} F_x + F_x^\intercal \mathcal{V}_{xx} \\
Q_{xu}(t) &= \mathcal{L}_{xu} + \mathcal{V}_{xx} F_u = Q_{ux}^\intercal \\
Q_{xv}(t) &= \mathcal{L}_{xv} + \mathcal{V}_{xx} F_v = Q_{vx}^\intercal \\
Q_{uu}(t) &= \mathcal{L}_{uu} \\
Q_{vv}(t) &= \mathcal{L}_{vv}
\end{aligned}
\tag{61}
$$

Additionally, taking the derivative in (60) with respect to $\delta \mathbf{u}_t$ and $\delta \mathbf{v}_t$ and setting them equal to 0, yields:

$$
\begin{aligned}
\delta \mathbf{u}^* &= -Q_{uu}^{-1}(Q_{ux}\delta \mathbf{x}_t + Q_{uv}\delta \mathbf{v}_t^* + Q_u) \\
\delta \mathbf{v}^* &= -Q_{vv}^{-1}(Q_{vx}\delta \mathbf{x}_t + Q_{uv}\delta \mathbf{u}_t^* + Q_v)
\end{aligned}
\tag{62}
$$

However, we notice that the optimal control of the opponent is present in the expression of the optimal control of each player, so we try to eliminate this dependency:

$$
\begin{aligned}
&\delta \mathbf{u}^* = -Q_{uu}^{-1}(Q_{ux}\delta \mathbf{x}_t + Q_{uv}(-Q_{vv}^{-1}(Q_{vx}\delta \mathbf{x}_t + Q_{uv}\delta \mathbf{u}_t^* + Q_v)) + Q_u) \\
&\Rightarrow (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu})\delta \mathbf{u}^* = -(Q_{ux} - Q_{uv}Q_{vv}^{-1}Q_{vx})\delta \mathbf{x}_t + Q_{uv}Q_{vv}^{-1}Q_v - Q_u \\
&\Rightarrow \delta \mathbf{u}^* = (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu})^{-1}(Q_{uv}Q_{vv}^{-1}Q_v - Q_v - (Q_{ux} - Q_{uv}Q_{vv}^{-1}Q_{vx})\delta \mathbf{x}_t) \\
&\Rightarrow \delta \mathbf{u}^* = l_u + K_u \delta \mathbf{x}_t
\end{aligned}
\tag{63}
$$

where $l_u = (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu})^{-1}(Q_{uv}Q_{vv}^{-1}Q_v - Q_u)$, and $K_u = (Q_{uu} - Q_{uv}Q_{vv}^{-1}Q_{vu})^{-1}((Q_{ux} - Q_{uv}Q_{vv}^{-1}Q_{vx})\delta \mathbf{x}_t)$. Similarly we can express $\delta \mathbf{v}^* = l_v + K_v \delta \mathbf{x}_t$, where equivalently the coefficients l_v, and K_v are defined as: $l_v = (Q_{vv} - Q_{vu}Q_{uu}^{-1}Q_{uv})^{-1}(Q_{uv}Q_{uu}^{-1}Q_u - Q_v)$, $\quad K_u = (Q_{vv} - Q_{vu}Q_{uu}^{-1}Q_{uv})^{-1}((Q_{vx} - Q_{uv}Q_{vv}^{-1}Q_{vx}))$. From (59) and (60), the value function and its first and second order derivatives with respect to $\mathbf{x}$ are expressed through the following backward ordinary differential equations:

$$
\begin{aligned}
&-\frac{d\mathcal{V}}{dt} - \frac{d\mathcal{V}_x}{dt}^\intercal \delta \mathbf{x}_t - \frac{1}{2}\delta \mathbf{x}_t^\intercal \frac{d\mathcal{V}_{xx}}{dt}\delta \mathbf{x}_t = \\
&= \min_{\gamma_u(t)} \max_{\gamma_v(t)} Q(\bar{x}, \bar{u}, \bar{v}, t) + Q_x^\intercal \delta \mathbf{x} + Q_u^\intercal \delta \mathbf{u}_t + Q_v^\intercal \delta \mathbf{v}_t + \frac{1}{2}\begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \\ \delta \mathbf{v}_t \end{bmatrix}^\intercal \begin{bmatrix} Q_{xx} & Q_{xu} & Q_{xv} \\ Q_{ux} & Q_{uu} & Q_{uv} \\ Q_{vx} & Q_{vu} & Q_{vv} \end{bmatrix} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \\ \delta \mathbf{v}_t \end{bmatrix} \\
&= Q(\bar{x}, \bar{u}, \bar{v}, t) + Q_x^\intercal \delta \mathbf{x} + Q_u^\intercal (l_u + K_u \delta \mathbf{x}_t) + Q_v^\intercal (l_v + K_v \delta \mathbf{x}_t) + \\
&+ \frac{1}{2}\begin{bmatrix} \delta \mathbf{x}_t \\ (l_u + K_u \delta \mathbf{x}_t) \\ (l_v + K_v \delta \mathbf{x}_t) \end{bmatrix}^\intercal \begin{bmatrix} Q_{xx} & Q_{xu} & Q_{xv} \\ Q_{ux} & Q_{uu} & Q_{uv} \\ Q_{vx} & Q_{vu} & Q_{vv} \end{bmatrix} \begin{bmatrix} \delta \mathbf{x}_t \\ (l_u + K_u \delta \mathbf{x}_t) \\ (l_v + K_v \delta \mathbf{x}_t) \end{bmatrix}
\end{aligned}
\tag{64}
$$

$$
= \begin{cases}
-\frac{d\mathcal{V}}{dt} = & \bar{\mathcal{L}} + l_u^\intercal Q_u + l_v^\intercal Q_v + \frac{1}{2} l_u^\intercal Q_{uu} l_u + \frac{1}{2} l_v^\intercal Q_{vv} l_v + l_u^\intercal Q_{uv} l_v \\
-\frac{d\mathcal{V}_x}{dt} = & Q_x + K_u^\intercal Q_u + K_v^\intercal Q_v + Q_{ux}^\intercal l_u + Q_{vx}^\intercal l_v + K_u^\intercal Q_{uu} l_u + \\
& + K_u^\intercal Q_{uv} lv + K_v^\intercal Q_{vu} l_u + K_v^\intercal Q_{vv} l_v \\
-\frac{d\mathcal{V}_{xx}}{dt} = & Q_{xx} + K_u^\intercal Q_{ux} + Q_{ux}^\intercal K_u + K_v^\intercal Q_{vx} + Q_{vx}^\intercal K_v + K_v^\intercal Q_{vu} K_u + \\
& + K_u^\intercal Q_{uv} K_v + K_u^\intercal Q_{uu} K_u + K_v Q_{vv} K_v
\end{cases}
\tag{65}
$$

under terminal conditions:

$$\begin{aligned} \bar{V}(t_f) &= \phi(\bar{x}(t_f), t_f) \\ \bar{V}_x(t_f) &= \phi_x(\bar{x}(t_f), t_f) \\ \bar{V}_{xx}(t_f) &= \phi_{xx}(\bar{x}(t_f), t_f) \end{aligned} \tag{66}$$